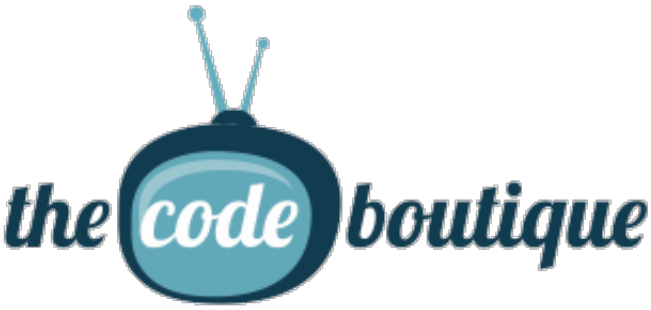


Mar 13, 2013

Creating Debian Squeeze Box for Vagrant (http://blog.codeboutique.com/post/creating-debian-squeeze-box-for-vagrant)

TL;DR. This article describes how you can setup a custom Debian base box for Vagrant (http://www.vagrantup.com/). Vagrant, or VagrantUp, is a system to create and configure lightweight, reproducible, and portable development environments. The documentation has a guide for setting up base boxes (http://docs-v1.vagrantup.com/v1/docs/base_boxes.html). It is a bit complex, perhaps because it aims too widely. This topic is labeled "only for advanced users" in the documentation, but I think the true power of Vagrant lies in working with your own custom boxes. For that reason I've created this guide.

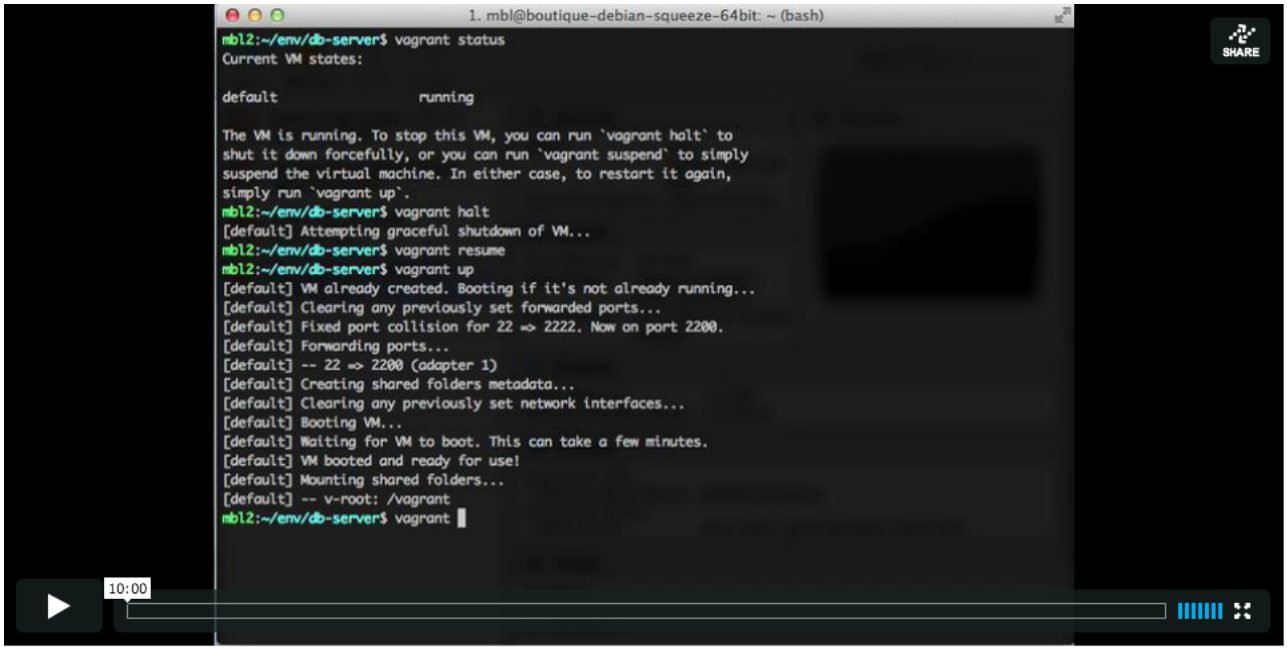
Background

More and more I find myself setting up a Debian box for work or for some big or small project. It can be a web-server a db-server or something else, but it is always the same. I keep telling myself to make a bare bone installation and clone on which I can build my future boxes. I never really came to that. Thanks to Vagrant, I don't have to.

Another issue I've been facing is: co-workers. Some are permanent others come and go, but they all need a development environment and preferably fast. Vagrant comes to the help again, by letting you package your box and make it available to your co-workers. With Vagrant they can have a working environment in minutes:

```
$ vagrant box add base
http://files.vagrantup.com/lucid32.box
$ vagrant init
$ vagrant up
```

Lastly I've realised that while the Mac OSX is an awesome and powerful OS, it just isn't Linux. `port`, `brew`, `fink` or building from source, it is too error prone and tiresome. With Vagrant it becomes easier to work with your *local* development environment within a VM. Vagrant helps you manage networks and ssh. You can manage must of this yourself manually from VirtualBox or whatever virtualiser you use, but I always run into one problem or the other.



(http://vimeo.com/61865852)

Vagrant

This is how Mitchell Hashimoto (http://twitter.com/mitchellh), the creator of Vagrant, describes it:

> Development made easy. Create and configure lightweight, reproducible, and portable development environments.

Vagrant uses VirtualBox (http://www.virtualbox.org/) to manage boxes. Make sure have it installed.

The example above refers to a box that Vagrant provides: `http://files.vagrantup.com/lucid32.box`. You can find a long list of pre-build boxes on GitHub (http://vagrantbox.es), maintained by fellow Vagrant lovers, in this guide, however, I will be creating box of Debian Squeeze 64bit from scratch. The point is that it should reflect my (and your) development environment, with my preferred apps and packages installed and/or setup by the best practices of my company - the goal being to give developers a running start.

Creating a Virtual Machine

I'm using a business card iso `debian-6.0.7-amd64-businesscard.iso` to install Debian in VirtualBox. Vagrant recommends you set up the virtual machine following these guidelines:

- Make sure you allocate enough disk space in a **dynamically resizing drive**. Typically, we use a 40 GB drive, which is big enough for almost everything.
- Make sure the default memory allocation is not too high. Most people don't want to download a box to find it using 1 GB of RAM. We typically **set it at 360 MB to start**,

since that is the size of most small slices. The **RAM is configurable by the user at run-time** using their `Vagrantfile`.

- Disable audio, usb, etc. controllers unless they're needed. Most applications don't need to play music! So save resources by disabling these features.

IMPORTANT: When creating the machine make sure that the network controller is set to **NAT**. AFAIK this is the default setting of VirtualBox.

> Bridged connects are not supported since it requires the machine to specify which device it is bridged to, which is unknown.

*Note: The machine I'm creating is called **DebianVagrantBox**. This is the name you enter in the very first form when creating a new machine in VirtualBox. This is the name I will use to refer to the box through out the guide.*

So with the above in mind go ahead and install Debian. When configuring Debian during install, for simplicity of configuring Vagrant you can follow these conventions:

- Root password: `vagrant`
- Main account login: `vagrant`
- Main account password: `vagrant`

If you consider distributing your box later on you should also consider setting:

- Hostname: `vagrant-[os-name]` e.g. `vagrant-debian-squeeze` in our case
- Domain: `vagrantup.com`

When installing keep it to a minimal: disable everything but the bare system tools.

Upgrade Guest Additions

The Guest Editions that comes packaged with the Debian Squeeze 6.0.7 distribution are terribly outdated, so we will upgrade them before setting up our server.

First update and install build files:

```
root:~$ apt-get update
root:~$ apt-get upgrade
root:~$ apt-get install build-essential
module-assistant
root:~$ m-a prepare
```

From **Devices** menu select **Install Guest Additions** then run:

```
root:~$ mount /media/cdrom
root:~$ sh /media/cdrom/VBoxLinuxAdditions.run
```

It might ask you if you want to remove existing Guest Additions - just say yes.

Setting up your environment

It is time to configure your system. This is where you create any users you might have, install packages, etc. Boot up the machine from VirtualBox and start installing.

Vagrant can do more for you than just setup a new machine using VirtualBox. You can also use it to configure the box, using what is referred to in the documentation as **Provisioners (http://docs-v1.vagrantup.com/v1/docs/provisioners/puppet.html)**. Provisioners are 3rd party configuration management and/or deployment tools like Puppet (http://www.puppetlabs.com/) (using manifests) and Chef (http://www.opscode.com/) (using cookbooks), that Vagrant can use when spinning up a box to make further configurations to, and install more packages on, the instance based on the box we are building here.

These are especially useful we you use the same base box to configure different kinds of servers like db-servers, web-servers, etc, that are all based on the same box. If you have no need for this functionality you don't have to install anything special, to support these provisioners.

You can also use the build in shell provisioner to run you shell scripts.

So whatever you choose to install at this step, whatever you choose to setup think of it as the common parts between all your systems, everything else, every little detail, can be configured later using provisioners.

Aside: For my work I use a third provisioner called ansible (http://ansible.cc) a python based command line tool that has a much more native feel to it than Puppet and Chef and needs a lot less configuration to work. Further more it doesn't require an agent on the client servers.

First install the necessary packages for Vagrant to worker properly:

```
root:~$ apt-get install openssh-server sudo zerofree
```

I like to have the following Debian packages installed as a bare minimum, but these are in no way required. This list is just something I've found that I always use. Since I use `ansible` (which relies on Python an SSH) there are no requirements as to what hast to be installed or setup on the box:

```
root:~$ apt-get install vim htop iotop rsync git tmux
python-dev python-pip curl
```

You can always use whatever provisioner you choose to use to install more specific packages, configure your network, change settings on configuration files, and so on. This is the reason why it is called **a base box** only the bare necessities. But if you know that you are always going to need Node.js or a (L)APM stack installed, then it would also be acceptable to do it here. So if you want to install a typical (L)AMP stack simply add:

```
root:~$ apt-get install apache2 mysql-server php5
php5-mysql
```

Of course you have to configure apache and php as well.

Setting Permissions

The `vagrant` user should be setup to have **password-less `sudo` privileges**.

The documentation tells you to add the `vagrant` user to the `admin` group and give the password-less sudo privileges to the that group. I think this is a bad practice. Assuming that you will later create actual users of the `admin` group, these users will be able to do anything with out any care in the world. The periodical password prompt is a good reminder that you have to be careful.

We will go ahead and create these privileges only for the `vagrant` user.

We will use `visudo` (why? (http://serverfault.com/questions/26303/why-do-i-have-to-edit-etc-sudoers-with-visudo)) to edit `/etc/sudoers` file. This is have we give the password-less privileges:

```
root:~$ visudo
```

This will open the editor. Add this line below the root definition (which is already there):

```
root    ALL=(ALL) ALL  <--------- don't add this,
already there
vagrant ALL=(ALL) NOPASSWD: ALL
```

Now save and exit `ctrl + o`, `enter`, `ctrl + x`. Be careful with the spelling, otherwise you'll get a syntax error, but luckily `visudo` will inform you about this - in that case just exit without saving and try again. Now restart the sudo-service:

```
root:~$ /etc/init.d/sudo restart
```

If change user to `vagrant` and test that you do not need to enter password:

```
root:~$ su vagrant            # you are now vagrant
vagrant:~$ sudo which sudo    # outputs: /usr/bin/sudo
```

SSH Access & Keys

Vagrant only supports **key-based authentication** for SSH and it uses SSH for most of its functionality. However you can login with password. First we will setup a port-forwarding. You can do this from the VirtualBox interface, but it is much faster from the command line. Make sure the box is turned off then and run:

```
you@laptop:~$ VBoxManage modifyvm "DebianVagrantBox" –
–natpf1 "ssh,tcp,,2222,,22"
```

Boot the box and now you should be able to access the box using ssh:

```
you@laptop:~$ ssh -p 2222 vagrant@127.0.0.1
```

We will need to append a public key to `./ssh/authorized_keys` in order to login without password. Before doing that read on.

If you use your own public key be aware that you cannot share this box with your co-workers, since the private key is for you alone.

In stead you could create a new key pair where the private key is shared amongst you and your co-workers.

Alternatively you could add all of your public keys to `~/.ssh/authorized_keys`. This however is problematic when people comes and goes, since the public key will be packaged with the box. You would have to repackage the box.

Vagrant supplies an "insecure" key pair (https://github.com/mitchellh/vagrant/tree/master/keys/) that you can use if you want to distribute your box publicly.

From within the box as `vagrant` user:

```
vagrant:~$ mkdir .ssh
vagrant:~$ chmod 700 .ssh
```

From your laptop:

```
you@laptop:~$ scp -P 2222 ~/.ssh/id_rsa.pub
vagrant@127.0.0.1:.ssh/authorized_keys
```

In the above example I copied my own public key into the box.

You should now be able to ssh into the box without password:

```
you@laptop:~$ ssh -p 2222 vagrant@127.0.0.1
```

Shrinking the Box

At this point everything has been installed and properly setup, the next step is to shrink the box to make it easier/faster to distribute.

The following step can reduce the size of our final packaged box from 550 to 240 Mb. You don't have to follow all the steps.

Remove shared docs (you'll still have your man pages):

```
root:~$ rm -rf /usr/share/doc
```

Remove guest addition source (not needed by VirtualBox anymore):

```
root:~$ rm -rf /usr/src/vboxguest*
root:~$ rm -rf /usr/src/virtualbox-ose-guest*
```

Remove linux headers:

```
root:~$ rm -rf /usr/src/linux-headers*
```

Remove cache:

```
root:~$ find /var/cache -type f -exec rm -rf {} \;
```

Remove locales:

```
# `en_US` and a few others are not removed
#
# remove your locale from the list below, e.g. your
locale is 'da' remove 'da'
# from the list

root:~$ rm -rf
```

```
/usr/share/locale/{af,am,ar,as,ast,az,bal,be,bg,bn,bn_I
N,br,bs,byn,ca,cr,cs,csb,cy,da,de,de_AT,dz,el,en_AU,en_
CA,eo,es,et,et_EE,eu,fa,fi,fo,fr,fur,ga,gez,gl,gu,haw,h
e,hi,hr,hu,hy,id,is,it,ja,ka,kk,km,kn,ko,kok,ku,ky,lg,l
t,lv,mg,mi,mk,ml,mn,mr,ms,mt,nb,ne,nl,nn,no,nso,oc,or,p
a,pl,ps,pt,pt_BR,qu,ro,ru,rw,si,sk,sl,so,sq,sr,sr*latin
,sv,sw,ta,te,th,ti,tig,tk,tl,tr,tt,ur,urd,ve,vi,wa,wal,
wo,xh,zh,zh_HK,zh_CN,zh_TW,zu}
```

Zerofree

`zerofree` is a tool that will fill un-allocated, non-zeroed blocks on the virtual disk with zeroes, making the disk more easily compressible (from Dom's Blog (http://dominique.broeglin.fr/2011/03/26/squeeze-64-vagrant-base-box.html)). Log in from VirtualBox as `root` **ssh'ing will not work** and execute:

```
root:~$ init 1
```

Enter root password and then:

```
root:~$ mount -o remount,ro /dev/sda1
root:~$ zerofree /dev/sda1
```

When done you can turn off the machine again and we are ready to box it.

Boxing

Vagrant will do all the boxing for us. We will just have to give it the name of the machine we created in the beginning - DebianVagrantBox.

First create a directory where the packaged box is going to end up:

```
you@laptop:~$ mkdir -p src/vagrant
you@laptop:~$ cd src/vagrant
```

Vagrant will export the box from VirtualBox and create a packed box called `package.box`. This is the file you want to share.

```
you@laptop:~/src/vagrant$ vagrant package --base
DebianVagrantBox
[DebianVagrantBox] Clearing any previously set
forwarded ports...
[DebianVagrantBox] Creating temporary directory for
export...
[DebianVagrantBox] Exporting VM...
[DebianVagrantBox] Compressing package to:
/Users/you/src/vagrant/package.box
```

Check the file size:

```
you@laptop:~/src/vagrant$ ls -lah
-rw-r--r--  1 mbl-private  staff   263M Mar 13 20:43
package.box
-rw-r--r--  1 mbl-private  staff   540M Mar 12 22:30
package.box-not-cleaned
```

Now let's add the newly created box to your repository of boxes. After this step you can use the box in your projects.

```
you@laptop:~/src/vagrant$ vagrant box add
MyDebianVagrantBox package.box
[vagrant] Downloading with
Vagrant::Downloaders::File...
[vagrant] Copying box to temporary location...
[vagrant] Extracting box...
[vagrant] Verifying box...
[vagrant] Cleaning up downloaded box...
```

You can confirm by listing added boxes:

```
you@laptop:~/src/vagrant$ vagrant box list
MyDebianVagrantBox
lucid32
```

There it is along with the `lucid32` from the example at the beginning of the guide.

To remove the box again:

```
you@laptop:~/src/vagrant$ vagrant box remove
MyDebianVagrantBox
[vagrant] Deleting box 'MyDebianVagrantBox'...
```

Lets try it out (remember to add it again if you deleted it).

Test

Our work is done, the our neatly shrunk and development friendly box is ready to be used by you and your co-workers. Lets create a new developing environment on Jim's laptop. First make sure you've provided Jim with: the private key for the public key you installed, and the box itself.

First Jim will have to add the box like we did:

```
jim@laptop:~$ vagrant add MyDebianVagrantBox
~/downloads/BigStartupDebianDevBox.box
[vagrant] Downloading with
Vagrant::Downloaders::File...
[vagrant] Copying box to temporary location...
[vagrant] Extracting box...
[vagrant] Verifying box...
[vagrant] Cleaning up downloaded box...
```

Jim will use the box to create web-server for a new project:

```
jim@laptop:~$ mkdir -p env/web
jim@laptop:~$ cd env/web
```

Vagrant uses `Vagrantfile`s to configure the box. You can create these yourself or you can just use the the build in `init` function the will build a simple file for you:

```
jim@laptop:~/env/web$ vagrant init MyDebianVagrantBox
A `Vagrantfile` has been placed in this directory. You
are now
ready to `vagrant up` your first virtual environment!
Please read
the comments in the Vagrantfile as well as
documentation on
`vagrantup.com` for more information on using Vagrant.
```

Ready to boot it up:

```
jim@laptop:~/env/web$ vagrant up
```

If the Guest Additions are outdated you will see a message like this:

```
[default] The guest additions on this VM do not match
the install version of
VirtualBox! This may cause things such as forwarded
ports, shared
folders, and more to not work properly. If any of those
things fail on
this machine, please update the guest additions and
repackage the
box.

Guest Additions Version: 3.2.10
VirtualBox Version: 4.2.6
```

(rewind and go back and update your Guest Additions)

If your Guest Additions is up to date you should something like this:

```
jim@laptop:~/env/web$ vagrant up
[default] Matching MAC address for NAT networking...
[default] Clearing any previously set forwarded
```

```
ports...
[default] Forwarding ports...
[default] -- 22 => 2222 (adapter 1)
[default] Creating shared folders metadata...
[default] Clearing any previously set network
interfaces...
[default] Booting VM...
[default] Waiting for VM to boot. This can take a few
minutes.
```

If you didn't install a ssh public key you will see this warning:

```
SSH authentication failed! This is typically caused by
the public/private
keypair for the SSH user not being properly set on the
guest VM. Please
verify that the guest VM is setup with the proper
public key, and that
the private key path for Vagrant is setup properly as
well.
```

"But I did", you say. This is also the error Vagrant shows if it cannot find the private key. So lets tell it where to find it. Lets open op the Vagrantfile that was created just before:

```
jim@laptop:~/env/web$ vim Vagrantfile
```

Removing all the comments, the file contains this:

```
Vagrant::Config.run do |config|
    config.vm.box = "MyDebianVagrantBox"
end
```

This is the simplest bit of code the that tells Vagrant how to startup and otherwise configure the box. This is where you put provisioners' code as well. Check the

Vagrantfile section (http://docs-v1.vagrantup.com/v1/docs/vagrantfile.html) of the documentation for info on what you can put in here. In our case the setting we are looking for is `config.ssh.private_key_path`. This setting will tell Vagrant where to look for the private key. By default Vagrant uses the mentioned insure private key.

```
Vagrant::Config.run do |config|
    config.vm.box = "MyDebianVagrantBox"
    config.ssh.private_key_path = "private-key-shared-
between-co-workers"
end
```

This will Vagrant look for a private key named `private-key-shared-between-co-workers` in `~/env/web` which was were the Vagrantfile resides.

If you added your own private key enter `~/.ssh/id_rsa` instead.

Let's do a clean start and destroy, or tear down, the box we just started:

```
jim@laptop:~/env/web$ vagrant destroy
Are you sure you want to destroy the 'default' VM?
[Y/N] Y
[default] Forcing shutdown of VM...
[default] Destroying VM and associated drives...
```

.. and then up again:

```
jim@laptop:~/env/web$ vagrant up
[default] Importing base box 'MyDebianVagrantBox'...
[default] Matching MAC address for NAT networking...
[default] Clearing any previously set forwarded
ports...
[default] Forwarding ports...
[default] -- 22 => 2222 (adapter 1)
[default] Creating shared folders metadata...
[default] Clearing any previously set network
interfaces...
```

```
[default] Booting VM...
[default] Waiting for VM to boot. This can take a few
minutes.
[default] VM booted and ready for use!
[default] Mounting shared folders...
[default] -- v-root: /vagrant
```

This time it was started without errors or warnings. Yeah!!!

Since Vagrant was now able to login using the key pair you can see it has mounted the project directory inside box. Let's check it out:

```
jim@laptop:~/env/web$ touch foobar.txt
jim@laptop:~/env/web$ vagrant ssh
vagrant@vagrant-debian-squeeze:~$ ls /vagrant
foobar.txt  Vagrantfile
```

Sometimes you want to login in using SSH using `ssh` instead of `vagrant ssh`. In this can run the following to check Vagrants ssh config for the box:

```
jim@laptop:~/env/web$ vagrant ssh-config
Host default
  HostName 127.0.0.1
  User vagrant
  Port 2222
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile
/Users/jim/env/web/private-key-shared-between-co-worker
s
  IdentitiesOnly yes
```

You can see the ssh is mapped to port 2222 on 127.0.0.1 so you can login like this:

```
$ ssh -p 2222 -i
```

```
~/env/web/private-key-shared-between-co-workers
vagrant@127.0.0.1
```

.. or simply:

```
$ ssh -p 2222 vagrant@127.0.0.1
```

.. if you used your own private key. You could also do it automatically like this:

```
$ vagrant ssh-config | ssh -F /dev/stdin default
```

If you get the error `Pseudo-terminal will not be allocated because stdin is not a terminal.` then try to add `-t -t` to the ssh part:

```
$ vagrant ssh-config | ssh -t -t -F /dev/stdin default
```

What Now

We've successfully created a new box that we can use ourselves or distribute amongst our co-workers and friends.

But we haven't really setup a complete developing environment yet. We have the box and we can create a new one in a matter of seconds.

From this we can start learning about the Vagrantfile (http://docs-v1.vagrantup.com/v1/docs/vagrantfile.html) and about Provisioners (http://docs-v1.vagrantup.com/v1/docs/provisioners/puppet.html).

Also if we are serious about setting up an environment with multiples servers, checkout the Multiple VM Environments (http://docs-v1.vagrantup.com/v1/docs/multivm.html). For this purpose we can use either host-only or bridge networks to communicate between the VMs. This is especially relevant if for instance you have a web-server and a db-server on two different VMs.

You can use Dropbox to share your box, even if you don't have a shared folder. Put your file on Dropbox and grab the share link. To the end of your share link append:

```
?dl=1
```

This will tell Dropbox to server the file instead of presenting the web interface.

Now go issue some commands and read the documentation.

.. to be continued.

chef (http://blog.codeboutique.com/tag/chef)

vagrant (http://blog.codeboutique.com/tag/vagrant)

image (http://blog.codeboutique.com/tag/image)

configuration-management (http://blog.codeboutique.com/tag/configuration-management)

guide (http://blog.codeboutique.com/tag/guide)

development (http://blog.codeboutique.com/tag/development)

virtualbox (http://blog.codeboutique.com/tag/virtualbox)

advanced (http://blog.codeboutique.com/tag/advanced)

deployment (http://blog.codeboutique.com/tag/deployment)

puppet (http://blog.codeboutique.com/tag/puppet)

article (http://blog.codeboutique.com/tag/article)

distro (http://blog.codeboutique.com/tag/distro)

ansible (http://blog.codeboutique.com/tag/ansible)

debian (http://blog.codeboutique.com/tag/debian)

dropbox (http://blog.codeboutique.com/tag/dropbox)
