

Vagrant - um tutorial prático

Paulo Jerônimo

Table of Contents

1. Sobre	1
2. Introdução	2
3. Instalando e executando boxes	3
4. Geando um box próprio, utilizando o Veewee	11
5. Disponibilizando o box gerado no Vagrant Cloud	16
6. Conclusão	17

Chapter 1. Sobre

Olá! Este é um tutorial que estou desenvolvendo sobre o [Vagrant](https://www.vagrantup.com/)¹ enquanto estudo um pouco mais sobre ele. Ele está público em <http://paulojeronimo.github.io/tutorial-vagrant>. Os fontes, [um documento](https://github.com/paulojeronimo/tutorial-vagrant/blob/master/README.asciidoc)² no formato [AsciiDoctor](http://asciidoctor.org)³ e um [script shell para sua construção](https://github.com/paulojeronimo/tutorial-vagrant/blob/master/build)⁴, estão disponíveis num [repositório GitHub](http://github.com/paulojeronimo/tutorial-vagrant)⁵. Ele também possui uma [versão em PDF](http://paulojeronimo.github.io/tutorial-vagrant/index.pdf)⁶, caso você queira baixá-lo para impressão.

Ainda não gerei nenhum `release` desse tutorial. Portanto, considere esta versão como em desenvolvimento. Essa primeira `release` deverá ser publicada na primeira semana de setembro.

¹ <https://www.vagrantup.com/>

² <https://github.com/paulojeronimo/tutorial-vagrant/blob/master/README.asciidoc>

³ <http://asciidoctor.org>

⁴ <https://github.com/paulojeronimo/tutorial-vagrant/blob/master/build>

⁵ <http://github.com/paulojeronimo/tutorial-vagrant>

⁶ <http://paulojeronimo.github.io/tutorial-vagrant/index.pdf>

Chapter 2. Introdução

O Vagrant é muito utilizado em ambientes de desenvolvimento para automatizar a sua instalação. Neste tutorial eu não apresento os motivos para utilizá-lo pois foco na parte prática, pressupondo que você já tenha o embasamento teórico que inclui suas motivações de uso. Caso esse não seja o teu caso, eu sugiro que você leia a página [Why Vagrant](https://docs.vagrantup.com/v2/why-vagrant/index.html)¹ e/ou ouça o [GrokPodcast](http://www.grokpodcast.com/)², nos episódios 100, 101 e 102. Nesses episódios são discutidos o Vagrant e o [Docker](http://www.docker.com)³ que, *em breve, também pretendo apresentar um caso prático a respeito*⁴.

Antes de começar a utilizar o Vagrant eu criei um projeto pessoal e caseiro que utilizo para criar minhas próprias máquinas virtuais (VMs). Esse projeto ainda está ativo e disponibilizado em <http://github.com/paulojeronimo/vms>. Digo que vale a pena você dar uma lida nele para que compreenda o quanto pode ser interessante o uso do Vagrant. Dando uma olhada neste projeto você perceberá que eu criei, de minha forma e utilizando scripts Bash, boa parte das soluções que o Vagrant resolve. Mas, especificamente para a utilização de VMs executadas no [VirtualBox](#)⁵. Isso me deixa um pouco limitado além do fato de me fazer concentrar na criação de scripts para gerar estas VMs. Apesar de gostar muito dessa área de automação, tudo o que eu preciso, no final das contas, são de VMs pré-configuradas para utilizar em alguns laboratórios e tutoriais que crio. Dessa forma, para mim o Vagrant resolve parte de meus problemas pois acaba com certas "distrações" em meu processo de criar VMs.

Após ouvir um pouco de teoria sobre Vagrant e Docker no GrokPodcast, e também de dar uma olhada no projeto pessoal que citei, volte para cá e comece a explorar as coisas de forma prática, mão na massa! =)

¹ <https://docs.vagrantup.com/v2/why-vagrant/index.html>

² <http://www.grokpodcast.com/>

³ <http://www.docker.com>

⁴ <https://twitter.com/paulojeronimo/status/504904544166621184>

⁵ <https://docs.vagrantup.com/v2/why-vagrant/index.html>

Chapter 3. Instalando e executando boxes

Para começar este tutorial você precisará ter instalado o [Ruby](https://www.ruby-lang.org/en/)¹, o VirtualBox e o Vagrant, em teu ambiente. No momento em que escrevo esta versão do tutorial, estas são as versões para as ferramentas que tenho instaladas:

```
$ # Versão do OS X:
$ sw_vers -productVersion
10.9.4

$ # Versão do Ruby:
ruby 2.1.1p76 (2014-02-24 revision 45161) [x86_64-darwin12.0]

$ # Versão do VirtualBox:
$ VBoxManage -version
4.3.14r95030

$ # Versão do Vagrant:
$ vagrant -v
Vagrant 1.6.3
```

Nesta versão do tutorial eu estou apresentando a execução do Vagrant para gerar uma VM Fedora 20, utilizando o OS X instalado em meu Mac. Contudo, os passos que demonstro aqui também ser reproduzidos em Linux e, numa próxima versão deste tutorial, eu demonstrarei sua execução no [Fedora que tenho instalado em meu Mac](http://j.mp/fedora-mac)², para gerar uma VM Ubuntu.

Vamos começar criando um novo diretório para este tutorial:

```
d=~/tutorial-vagrant
rm -rf $d
mkdir -p $d && cd $d
```

A execução do comando vagrant sem nenhum parâmetro apresenta-nos várias possibilidades. Execute o comando abaixo e observe sua saída:

¹ <https://www.ruby-lang.org/en/>

² <http://j.mp/fedora-mac>

Instalando e executando boxes

```
vagrant
```

Temos várias opções disponíveis, não é? Por exemplo, para verificar a versão do Vagrant, execute:

```
vagrant version
```

Note também que, na primeira execução do Vagrant, será criado o diretório `~/.vagrant.d`.

Observe a estrutura desse diretório executando:

```
tree ~/.vagrant.d
```

Sua saída deverá ser semelhante a que eu recebo em meu ambiente:

```
/Users/pj/.vagrant.d
|-- boxes
|-- data
|   |-- machine-index
|   |-- index.lock
|-- gems
|   |-- ruby
|   |-- 2.0.0
|-- insecure_private_key
|-- rgloader
|   |-- loader.rb
|-- setup_version
|-- tmp

8 directories, 4 files
```

Para localizar os boxes (máquinas virtuais, na linguagem do Vagrant) disponíveis para a instalação acesse a URL <https://vagrantcloud.com/discover/featured>. Observe que nesse site você encontrará inúmeros boxes listados, produzidos por diferentes usuários. Note também que é possível criar o teu próprio box e publicá-lo pois há opções para isso (mais a frente discutiremos a esse respeito).

Localizado o box que gostaríamos de usar, vamos agora criar um novo diretório para iniciá-lo. Iniciaremos, por exemplo, um box Fedora 20. Execute:

Instalando e executando boxes

```
vagrant init olibob/fedora20
ls -la
```

Observe, pela saída do último comando acima, que isso gerará um arquivo Vagrantfile no diretório corrente. Esse arquivo nada mais é que um código Ruby. Veja o seu conteúdo:

```
view Vagrantfile
```

Para colocar o box que foi iniciado em execução, execute:

```
vagrant up
```

A saída desse comando apresenta alguns detalhes interessantes. Dentre as linhas mostradas, destaco as seguintes:

```
....
==> default: Preparing network interfaces based on configuration...
      default: Adapter 1: nat
==> default: Forwarding ports...
      default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
      default: SSH address: 127.0.0.1:2222
....
==> default: Checking for guest additions in VM...
==> default: Mounting shared folders...
      default: /vagrant => /Users/pj/tutorial-vagrant
```

Observe que só há um adaptador de rede nessa box e que ele está configurado para usar NAT. Dessa forma, só conseguiríamos fazer um ssh para esse box através do uso de um túnel SSH reverso. O interessante é que o Vagrant já cria esse túnel para nós, facilitando nossa vida! =)

Os curiosos também perceberão, fazendo a leitura da saída acima, que o SSH será feito do hospedeiro (HOST) para o box através da porta 2222. Num HOST Linux ou OS X, por exemplo, é possível verificar isso através do comando a seguir:

```
sudo lsof -n -P -i :2222
```

Também podemos notar que o Vagrant já monta uma pasta compartilhada /vagrant associando-a ao diretório onde o box foi iniciado ~/tutorial-vagrant. Isso facilita a troca de arquivos entre o box e o HOST.

Um último detalhe é que, após colocarmos o box em execução, também é criado o diretório .vagrant, no diretório de inicialização do box. Perceba isso executando os comandos a seguir:

```
ls -la
tree .vagrant
```

Para listar os boxes instalados (você pode estar qualquer diretório), execute:

```
vagrant box list
```

Observe que, a saída do comando vagrant executado acima apresenta as informações do nome do box, o `provider` que está executando-o e sua versão:

```
olibob/fedora20 (virtualbox, 0.1.0)
```

Experimente, agora, fazer um ssh para o box. Execute:

```
vagrant ssh
```

No shell, dentro do box, execute:

```
uname -a
cat /etc/redhat-release
ls -la /vagrant
logout
```

Para ver o status do box, execute:

```
vagrant status
```

Observe que a saída desse comando nos informa que, para parar a VM, podemos executar um `vagrant suspend` ou um `vagrant halt` (esse último é como se fosse pressionar o botão `power off` da máquina com ela em funcionamento).

Apenas suspenda a execução da VM:

```
.....  
vagrant suspend  
.....
```

Se você observar a tela de administração do VirtualBox, notará que a execução da VM foi suspensa. Você poderia desligar seu computador, ir tomar um café ou fazer qualquer outra coisa. Outra hora você poderia voltar e colocar a VM em funcionamento novamente. Mas, façamos isso agora:

```
.....  
vagrant up  
.....
```

Entre novamente no shell da VM (`vagrant ssh`) e desligue a VM (`shutdown -h now`). Será solicitada a senha do usuário `vagrant` que, por padrão, é `vagrant`. Um detalhe: se você executar `sudo shutdown -h now` não será necessário informar nenhuma senha pois o usuário `vagrant` tem permissões de root, conforme configurações no arquivo `/etc/sudoers`.

Execute mais uma vez o comando a seguir. Ele deverá informar que a VM está no estado de desligada.

```
.....  
vagrant status  
.....
```

Agora, removeremos o registro da VM. Para isso, serão necessários dois passos. O primeiro deles é observar a saída do comando a seguir:

```
.....  
vagrant global-status  
.....
```

No meu caso, sua saída foi essa:

```
.....  
id          name      provider  state    directory  
-----  
5d24c9b     default  virtualbox running  /Users/pj/vagrant-tutorial  
.....
```

The above shows information about all known Vagrant environments on this machine. This data is cached and may not be completely up-to-date. To interact with any of the machines, you can go to that directory and run Vagrant, or you can use the ID directly with Vagrant commands from any directory. For example:

```
"vagrant destroy 1a2b3c4d"
```

```
.....
```

De posse do id do box podemos solicitar sua destruição, executando o comando a seguir (informe o id apresentado em tua saída):

```
vagrant destroy 5d24c9b
```

Um detalhe a respeito desse id (5d24c9b): a cada vez que você destrói e inicializa uma nova VM, esse número muda.

O comando `vagrant destroy xxxxxx` forçará a parada da VM e também removerá seu registro do `provider` (no caso, o VirtualBox). Note, entretanto, que esse “destroy” na verdade só remove o box (VM) do VirtualBox (ou do `provider` em que ela estiver sendo utilizado). O box baixado ainda continua disponível para ser registrado novamente no VirtualBox pois ele permanece no diretório `~/.vagrant.d`.

Agora, repita o comando `tree ~/.vagrant.d` e observe a estrutura final do `dotfile` do Vagrant:

Em meu ambiente, esta é a saída produzida:

```
/Users/pj/.vagrant.d/
|-- boxes
|   |-- olibob-VAGRANTSLASH-fedora20
|       |-- 0.1.0
|           |-- virtualbox
|               |-- Vagrantfile
|               |-- box-disk2.vmdk
|               |-- box.ovf
|               |-- metadata.json
|               |-- metadata_url
|-- data
|   |-- fp-leases
|   |-- lock.dotlock.lock
|   |-- machine-index
|       |-- index
|       |-- index.lock
|-- gems
|   |-- ruby
|       |-- 2.0.0
|-- insecure_private_key
|-- rgloader
|   |-- loader.rb
|-- setup_version
|-- tmp
```

```
12 directories, 11 files
```

Faça um backup do diretório `dotfile` para testarmos um comando, logo a seguir:

```
cp -r ~/.vagrant.d/ ~/.vagrant.d.backup
```

Agora, execute:

```
vagrant box remove olibob/fedora20
```

Observe, pela saída do comando a seguir, que o box foi eliminado do diretório `~/.vagrant.d`:

```
tree ~/.vagrant.d/boxes/
```

Após ter feito isso, se for necessário executar novamente esse box o Vagrant terá que fazer o seu download outra vez. Como não queremos isso nesse instante, vamos simplesmente voltar o que tínhamos, através do `rsync`:

```
rsync -av ~/.vagrant.d.backup/ ~/.vagrant.d/
```

Isso é o básico de Vagrant! Utilizando-o perceberemos o quanto fica simples trabalhar com VMs, baixando-as de um repositório de VMs já existentes. Mas, agora vamos resolver outra questão: e se quisermos ter a nossa própria VM registrada nesse repositório?

É nesse ponto que entram as ferramentas [Packer](https://github.com/mitchellh/packer)³, ou o [Veewee](https://github.com/jedi4ever/veewee)⁴.

O Packer é um projeto da própria [HashiCorp](http://www.hashicorp.com)⁵, a empresa fundada pelo [Mitchell Hashimoto](https://github.com/mitchellh)⁶ e que está por trás do desenvolvimento do Vagrant. O Packer é desenvolvido em [Go](http://golang.org/)⁷ e, se quiséssemos fazer seu uso sem ser através dos [binários de instalação disponíveis](http://www.packer.io/downloads.html)⁸, precisaríamos de montar um ambiente de desenvolvimento

³ <https://github.com/mitchellh/packer>

⁴ <https://github.com/jedi4ever/veewee>

⁵ <http://www.hashicorp.com>

⁶ <https://github.com/mitchellh>

⁷ <http://golang.org/>

⁸ <http://www.packer.io/downloads.html>

para o trabalho com essa linguagem. Em próximos tutoriais falarei sobre o uso do Packer mas, se você já quiser ver exemplos de como construir VMs com o seu uso eu sugiro, por exemplo, que veja o projeto [packer-templates](https://github.com/kaorimatz/packer-templates)⁹ do [Satashi Matsumoto](https://github.com/kaorimatz)¹⁰.

O Veewee, por sua vez, é uma ferramenta Ruby e será utilizada nesse primeiro tutorial.

⁹ <https://github.com/kaorimatz/packer-templates>

¹⁰ <https://github.com/kaorimatz>

Chapter 4. Geando um box próprio, utilizando o Veewee

Nosso primeiro passo será instalá-lo via [gems](http://guides.rubygems.org/command-reference/)¹:

```
gem install veewee
```

Para iniciar a definição de nossa VM, utilizaremos um template. E, para listar os templates disponíveis para a geração de um Fedora, por exemplo, executamos:

```
veewee vbox templates | grep -i fedora
```

Iremos utilizar, como template de definição para o Fedora 20, o template Fedora-19-x86_64. Então, executaremos:

```
veewee vbox define Fedora-20-x86_64 Fedora-19-x86_64
```

Esse comando gerará diversos arquivos, dentro do diretório corrente no diretório definitions. Vejamos quais:

```
tree definitions
```

Em meu ambiente, são apresentados os seguintes:

```
definitions/  
|-- Fedora-20-x86_64  
    |-- base.sh  
    |-- chef.sh  
    |-- cleanup.sh  
    |-- definition.rb  
    |-- ks.cfg  
    |-- puppet.sh  
    |-- ruby.sh  
    |-- vagrant.sh  
    |-- virtualbox.sh  
    |-- vmfusion.sh
```

¹ <http://guides.rubygems.org/command-reference/>

```
`-- zerodisk.sh

1 directory, 11 files
```

Agora, precisamos ajustar esses arquivos de acordo com nossas necessidades.

Falando de meu caso particular, onde gero [minha vm-fedora](http://gdriv.es/vm-fedora)² e disponibilizo-a para que as pessoas possam executar alguns dos meus tutoriais, eu edito os arquivos `ks.cfg` e `definitions.rb`. Baixe o [meu projeto vagrant-definitions](http://github.com/paulojeronimo/vagrant-definitions)³ para poder comparar minhas alterações com relação a estrutura que o Veewee acabou de gerar no diretório corrente. Para isso, execute:

```
git clone http://github.com/paulojeronimo/vagrant-definitions pj
diff -r definitions pj/definitions
```

A saída do diff acima, no momento em que escrevo este tutorial, é esta:

```
diff -r definitions/Fedora-20-x86_64/definition.rb pj/definitions/
Fedora-20-x86_64/definition.rb
4,5c4,5
<  :memory_size=> '512',
<  :disk_size => '10140',
---
>  :memory_size=> '1024',
>  :disk_size => '8192',
10,12c10,12
<  :iso_file => "Fedora-19-x86_64-DVD.iso",
<  :iso_src => "http://download.fedoraproject.org/pub/fedora/linux/
releases/19/Fedora/x86_64/iso/Fedora-19-x86_64-DVD.iso",
<  :iso_sha1 => "73e45acf91d73146c7a71f7e8ca72762833aeadd",
---
>  :iso_file => "Fedora-20-x86_64-DVD.iso",
>  :iso_src => "http://localhost/Fedora-20-x86_64/releases/20/Fedora/
x86_64/iso/Fedora-20-x86_64-DVD.iso",
>  :iso_sha1 => "36dd25d7a6df45cdf19b85ad1bf2a2ccbf34f991",
```

As mudanças que eu faço no arquivo `definition.rb` são apenas para aumentar a memória da VM, diminuir o tamanho do seu disco e, na sua geração, utilizar o ISO e o mirror que tenho locais em minha máquina. Detalhe: eu crio e utilizo esse mirror local

² <http://gdriv.es/vm-fedora>

³ <http://github.com/paulojeronimo/vagrant-definitions>

(veja como em [meu projeto mirrors⁴](#)) pois, sem ele, o Veewee teria que baixar vários pacotes pela Internet e isso tornaria o processo de geração da VM ainda mais lento.

Outro detalhe é que, antes de iniciar a construção da VM, o Veewee fará uma cópia de alguns ISOs para o diretório iso, casos eles já não estejam lá. Como, em meu caso, eu já possuo estes isos localmente no sistema de arquivos do meu Mac, eu executo os seguintes passos:

```
mkdir iso && cd iso
ln -s /Applications/VirtualBox.app/Contents/MacOS/VBoxGuestAdditions.iso
    VBoxGuestAdditions_4.3.14.iso
ln -s /PJ-HFS/mirrors/Fedora-20-x86_64/releases/20/Fedora/x86_64/iso/
    Fedora-20-x86_64-DVD.iso
```

Para encontrar o valor de iso_sha1 que configurei no arquivo definition.rb, no Mac, eu executo:

```
shasum Fedora-20-x86_64-DVD.iso
```

Finalmente, terminadas as edições dos arquivos em definitions (e a geração, opcional, dos links para os isos), a construção da VM através do Veewee é feita de forma totalmente automática (sem nenhum input de nossa parte), pela execução do seguinte comando:

```
cd ..
veewee vbox build Fedora-20-x86_64
```

O processo de construção da VM levará algum tempo (de 5 a 15 minutos dependendo da velocidade de teu computador). Quando esse processo terminar, você pode executar o seguinte comando para encerrar a execução do box gerado:

```
veewee vbox halt Fedora-20-x86_64
```

Para disponibilizar o box para uso do Vagrant, precisamos executar o comando a seguir:

```
veewee vbox export Fedora-20-x86_64
```

⁴ <http://github.com/paulojeronimo/mirrors>

Geando um box próprio, utilizando o Veewee

Esse comando gerará o arquivo `Fedora-20-x86_64.box` no diretório corrente e apresentará a instrução para que esse box possa ser importado pelo Vagrant. Em meu caso, sua saída é essa:

```
.....
Creating a temporary directory for export
Adding additional files
Creating Vagrantfile
Exporting the box
Executing VBoxManage export Fedora-20-x86_64 --output /var/folders/7h/
tbnf44cx6dsdc0mbptm0k_580000gn/T/d20140831-9085-3gfo8j/box.ovf
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Successfully exported 1 machine(s).
Packaging the box
Cleaning up temporary directory

To import it into vagrant type:
vagrant box add 'Fedora-20-x86_64' '/Users/pj/tutorial-vagrant/Fedora-20-
x86_64.box'

To use it:
vagrant init 'Fedora-20-x86_64'
vagrant up
vagrant ssh
.....
```

Apenas para fazer um teste, seguiremos as instruções passadas:

```
.....
vagrant box add Fedora-20-x86_64 Fedora-20-x86_64.box
.....
```

Como no diretório corrente já existe um Vagrantfile, precisaremos movê-lo para `Vagrantfile.old`, pois continuaremos o tutorial nesse mesmo diretório:

```
.....
mv Vagrantfile Vagrantfile.old
.....
```

Agora iniciaremos o novo box que acabamos de importar:

```
.....
vagrant init Fedora-20-x86_64
.....
```

Para testar, executaremos o box e faremos um ssh para ele:

```
.....
vagrant up
vagrant ssh
.....
```


Geando um box próprio, utilizando o Veewee

Por fim, faremos a destruição do box:

```
.....  
vagrant destroy $(vagrant global-status | awk '/tutorial-vagrant/{print  
$1}')
```

```
.....
```

That's all! Sabemos que o box está funcionando. Nossa última tarefa é publicá-lo para que outras pessoas possam utilizá-lo.

Chapter 5. Disponibilizando o box gerado no Vagrant Cloud

O repositório de onde o Vagrant baixa as VMs (boxes) é o [Vagrant Cloud](https://vagrantcloud.com/)¹. Dessa forma precisamos [criar nossa conta nesse serviço](https://vagrantcloud.com/account/new)² para poder utilizá-lo. Após criá-la, podemos configurar nossos próprios boxes.

O site Vagrant Cloud apresenta um passo a passo muito simples para que possamos publicar nossos próprios boxes. É só segui-lo, não tem segredos e os passos são descritos na página [Creating a new Box](https://vagrantcloud.com/help/boxes/create)³! =) Num dos últimos passos é necessário informar uma URL para a localização do box (o arquivo com a extensão .box). No meu caso, eu informo o link público para o box que publico em minha conta no Google Drive. Por fim, depois de ter meu box publicado no Vagrant, eu posso utilizá-lo através do comando a seguir:

```
vagrant init paulojeronimo/heisenbug64
```

Daí é só usar, como eu expliquei anteriormente!

Todas as VMs que criamos no Vagrant Cloud também ficam acessíveis através de uma URL contendo nosso nome de usuário no Vagrant Cloud. A minha URL é <https://vagrantcloud.com/paulojeronimo/>.

¹ <https://vagrantcloud.com/>

² <https://vagrantcloud.com/account/new>

³ <https://vagrantcloud.com/help/boxes/create>

Chapter 6. Conclusão

O Vagrant pode ser extremamente útil para vários ambientes (desenvolvimento, integração, ...). Especificamente no caso de ambientes de desenvolvimento, sua utilização pode facilitar enormemente a inclusão de um novo desenvolvedor num projeto pois reduz, de forma significativa, o tempo de setup inicial que esse desenvolvedor teria para montar o seu ambiente. Além disso, cria um ambiente comum e de configurações iguais para todos os desenvolvedores de uma mesma equipe. A utilização do Vagrant oferece visibilidade a todo o processo de montagem da VM e apresenta, de forma clara, todas as etapas e configurações necessárias para a criação de uma VM.