Vagrant - um tutorial prático

Paulo Jerônimo

Table of Contents

Chapter 1. Sobre

Olá! Este é um tutorial que estou desenvolvendo sobre o Vagrant[1] enquanto estudo um pouco mais sobre ele. Ele está público em http://paulojeronimo.github.io/tutorial-vagrant. Os fontes, um documento[2] no formato AsciiDoctor[3] e um script shell para sua construção[4], estão disponíveis num repositório GitHub[5]. Ele também possui uma versão em 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[1] e/ou ouça o GrokPodcast[2], nos episódios 100, 101 e 102. Nesses episódios são discutidos o Vagrant e o Docker[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 concentar 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[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 -v
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[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. Executemos o comando abaixo para observar sua saída:

[1] https://www.ruby-lang.org/en/
[2] http://j.mp/fedora-mac

```
vagrant
```

Temos várias opções disponíveis, não é? Por exemplo, para verificar a versão do Vagrant executemos o seguinte comando:

```
vagrant version
```

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

Vejamos a estrutura desse diretório:

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

A saída desse comando em meu ambiente é a seguinte:

```
/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 (VMs, na linguagem do Vagrant) disponíveis para a instalação acessemos a URL https://vagrantcloud.com/discover/featured. Observamos que nesse site são encontrados inúmeros boxes, produzidos por diferentes usuários. Podemos notar também que é possível criar nosso 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 um box Fedora 20 executando:

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

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

```
view Vagrantfile
```

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

```
vagrant up
```

A saída desse comando apresenta alguns detalhes interessantes. Dentre as linhas mostradas, notemos 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
```

Observemos 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! =)

Podemos perceber, pela 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 e 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. Observemos isso executando os comandos a seguir:

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

Para listar os boxes instalados (podemos estar qualquer diretório), executemos:

```
vagrant box list
```

Observemos, pela saída do comando executado acima, que são apresentadas as informações do nome do box, o `provider` que está executando-o e sua versão:

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

Experimentemos, agora, fazer um ssh para o box. Executemos:

```
vagrant ssh
```

No shell, dentro do box, executemos:

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

Para ver o status do box, executemos:

```
vagrant status
```

Observemos 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).

Suspendamos a execução da VM:

```
vagrant suspend
```

Se observarmos a tela de administração do VirtualBox, notaremos que a execução da VM foi suspensa. Poderíamos desligar o computador, ir tomar um café ou fazer qualquer outra coisa. Outra hora, poderíamos voltar e colocar a VM em funcionamento novamente. Mas, por agora, façamos isto:

```
vagrant up
```

Enviemos um comando para desligar a VM, via SSH:

```
ssh -p 2222 vagrant@localhost 'sudo shutdown -h now'
```

Será solicitada a senha do usuário vagrant que, por padrão, é vagrant. Um detalhe: para evitar passar essa senha a cada execução do comando ssh, gere sua chave pública com o comando ssh-keygen (se ainda não tiver uma) e utilize o comando ssh-copy-id para inserí-la no arquivo ~/.ssh/authorized_keys do usuário vagrant. Note também que o usuário vagrant tem permissões de root, conforme as configurações definidas no arquivo /etc/sudoers.

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

```
vagrant status
```

Agora, removamos o registro da VM. Para isso, serão necessários dois passos. O primeiro deles é executar e 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"
```

O que nos importa é o id do box. Com ele 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 destruímos e inicializamos uma nova VM, esse número muda.

O comando vagrant destroy <id> forcará a parada da VM e também removerá seu registro do `provider` (no caso, o VirtualBox). Notemos, 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, repitamos o comando tree ~/.vagrant.d e observemos 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
```

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

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

Agora, executemos:

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

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

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

Após termos 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 tinhamos, através do rsync:

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

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

É nesse ponto que entram as ferramentas Packer[3], ou o Veewee[4].

O Packer é um projeto da própria HashiCorp[5], a empresa fundada pelo Mitchell Hashimoto[6] e que está por trás do desenvolvimento do Vagrant. O Packer é

[3] https://github.com/mitchellh/packer
[4] https://github.com/jedi4ever/veewee
[5] http://www.hashicorp.com
[6] https://github.com/mitchellh

desenvovido em Go[7] e, se quiséssemos fazer seu uso sem ser através dos binários de instalação disponíveis[8], precisaríamos de montar um ambiente de desenvolvimento para o trabalho com essa linguagem. Em próximos tutoriais eu 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 você veja o projeto packer-templates[9] do Satashi Matsumoto[10].

O Veewee, por sua vez, é uma ferramenta Ruby. Utilizarei-o nesse primeiro tutorial.

[7] http://golang.org/
[8] http://www.packer.io/downloads.html
[9] https://github.com/kaorimatz/packer-templates
[10] https://github.com/kaorimatz

Chapter 4. Gerando um box próprio, utilizando o Veewee

Esse é um tópico avançado deste tutorial, consome algum tempo e é um processo tedioso (assim como citado na página "Creating a Base Box"[1]. Eu não espero que você consiga seguir-me em sua execução pois, para isso, você precisaria criar um mirror local do Fedora[2] e, também, disponibilizá-lo através de um servidor HTTP em tua máquina, assim como eu faço em meu projeto vms[3]. Portanto, prosseguirei a execução deste tutorial sozinho e espero que você consiga voltar para cá e adaptar os comandos que apresento para o teu ambiente de forma que, em seguida, possa me dar um feedback positivo sobre sua execução.

Para iniciar a criação de um box próprio, meu primeiro passo é instalar o Veewee através do gems[4]:

```
gem install veewee
```

Para definir minha VM, eu utilizo um template. E, para listar os templates disponíveis para a geração de um Fedora, por exemplo, executo:

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

Utilizarei, como template de definição para o Fedora 20, o template Fedora-19-x86_64. Então, executo:

```
veewee vbox define Fedora-20-x86_64 heisenbug64
```

Heisenbug é o codenome oficial para o Fedora 20. Veja o histórico de codenomes do Fedora[5].

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

1 http://docs.vagrantup.com/v2/boxes/base.html
2 https://github.com/paulojeronimo/mirrors
3 https://github.com/paulojeronimo/vms/
4 http://guides.rubygems.org/command-reference/
5 https://fedoraproject.org/wiki/History_of_Fedora_release_names

```
tree definitions
```

Em meu ambiente, esta é a saída:

```
definitions/
`-- heisenbug64
    |-- base.sh
    |-- chef.sh
    |-- cleanup.sh
    |-- definition.rb
    |-- ks.cfg
    |-- puppet.sh
    |-- ruby.sh
    |-- vagrant.sh
    |-- virtualbox.sh
    |-- vmfusion.sh
    `-- zerodisk.sh

1 directory, 11 files
```

Precisamos ajustar esses arquivos de acordo com nossas necessidades.

No meu caso particular, onde gero minha vm-fedora[6] e disponiblizo-a para que as pessoas possam executar alguns dos meus tutoriais, eu edito os arquivos ks.cfg e o definitions.rb. Vou baixar o meu projeto veewee-definitions[7] para mostrar minhas alterações na estrutura que Veewee gerou localmente, na saída que apresentei acima. Farei isso executando os seguintes comandos:

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

A saída do diff, que apresentarei a seguir, poderá variar no futuro se o meu projeto veewee-definitions sofrer atualizações. Mas, no momento em que escrevo este tutorial, esta é a saída:

```
diff -r definitions/Fedora-20-x86_64/definition.rb pj/definitions/
Fedora-20-x86_64/definition.rb
4,5c4,5
```

[6] http://gdriv.es/vm-fedora
[7] http://github.com/paulojeronimo/veewee-definitions

```
<   :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 (veja como em meu projeto mirrors[8]) 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 (na estrutura de diretórios do mirror do Fedora), 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

[8] http://github.com/paulojeronimo/mirrors

totalmente automática (sem nehum input de nossa parte), pela execução do seguinte comando:

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

O processo de construção da VM leva cerca de 10 minutos em meu Mac. Quando esse processo termina, eu executo o seginte comando para encerrar a execução do box gerado:

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

Para disponibilizar o box para uso do Vagrant, executo:

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

Esse último comando gera o arquivo Fedora-20-x86_64.box, no diretório corrente, e apresenta uma instrução para que esse box possa ser importado pelo Vagrant. No meu ambinte, sua saída é esta:

```
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
```

Para testar, o box gerado, eu sigo as instruções passadas:

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

Como no diretório corrente já existe um Vagrantfile, eu preciso movê-lo para Vagrantfile.old, pois continuo a execução deste tutorial nesse mesmo diretório:

```
mv Vagrantfile Vagrantfile.old
```

Inicio o novo box que acabei de importar:

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

Executo o box e faço um ssh para ele:

```
vagrant up
vagrant ssh
```

Por fim, faço destruição do box:

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

`That's all!` Agora sei que o box está funcionando. Minha 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[1]. Dessa forma preciso criar minha conta nesse serviço[2] para poder utilizá-lo e após criá-la, posso configurar meus 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[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 Veewee 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.