Vagrant tutorial

Bert Van Vreckem

LOADays, 5-6 April 2014

Whoami

Bert Van Vreckem

- Lecturer ICT at University College Ghent
  - Mainly Linux & open source
  - Coordinator Bachelor thesises
- @bertvanvreckem
- +BertVanVreckem
- http://be.linkedin.com/in/bertvanvreckem/
- http://youtube.com/user/bertvvhogent/
- http://hogentsysadmin.wordpress.com/

Have a question/remark? Please interrupt me!

Agenda

- Vagrant introduction
- Getting base boxes
- Configuring boxes
- Provisioning
  - shell, Ansible, Puppet
  - setting up a LAMP stack
- Creating base boxes

Introduction

What is Vagrant?

http://www.vagrantup.com/

- Written by Mitchell Hashimoto
- Command line tool
- Automates VM creation with
  - VirtualBox
  - VMWare

    - Hyper-V
- Integrates well with configuration management tools
    - Shell
    - Ansible
    - Chef
    - Puppet
- Runs on Linux, Windows, MacOS

Why use Vagrant?

- Create new VMs quickly and easily
    - Only one command! `vagrant up`
- Keep the number of VMs under control
- Reproducability
- Identical environment in development and production
- Portability
    - No more 4GB .ova files
    - `git clone` and `vagrant up`

Assumptions

- Git
- Vagrant 1.5.1
- VirtualBox 4.3.10
    - default Host-only network (192.168.56.0/24)
- `librarian-puppet`

```
$ vagrant --version
Vagrant 1.5.1
$ VBoxHeadless --version
Oracle VM VirtualBox Headless Interface 4.3.10
(C) 2008-2014 Oracle Corporation
All rights reserved.

4.3.10r93012
$ ifconfig vboxnet0
=> 192.168.56.1
```

Try it yourself

- Clone the repository `git clone git@github.com:bertvv/vagrant-example.git`
- When the slides mention “`checkpoint-nn`”, you can do `git checkout tags/checkpoint-nn`

Getting up and running

Minimal default setup:

```
$ vagrant init hashicorp/precise32
$ vagrant up
$ vagrant ssh
```

What happens under the hood?

```
$ vagrant init hashicorp/precise32
```

A Vagrantfile is created (that's all!)

What happens under the hood?

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'hashicorp/precise32' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'hashicorp/precise32'
    default: URL: https://vagrantcloud.com/hashicorp/precise32
==> default: Adding box 'hashicorp/precise32' (v1.0.0) for provider: virtualbox
    default: Downloading: https://vagrantcloud.com/hashicorp/precise32/version/1/provider/virtualbox.box
==> default: Successfully added box 'hashicorp/precise32' (v1.0.0) for 'virtualbox'!
==> default: Importing base box 'hashicorp/precise32'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'hashicorp/precise32' is up to date...
==> default: Setting the name of the VM: example_default_1395996714768_3176
==> default: Clearing any previously set network interfaces...
==> 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: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
```

```
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Mounting shared folders...
    default: /vagrant => /home/bert/CfgMgmt/vagrant-example
```

What happens under the hood?

```
$ vagrant init hashicorp/precise32
```

- The base box is downloaded and stored locally
  - in `~/.vagrant.d/boxes/`
- A new VM is created and configured with the base box as template
- The VM is booted
- The box is provisioned
  - only the first time, must be done manually afterwards

Done!

You now have a working VM, ready for use:

```
$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/
Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:22:31 2012 from 10.0.2.2
vagrant@precise32:~$
```

Configuring Vagrant boxes

Vagrantfile

Minimal Vagrantfile (`checkpoint-01`):

```
VAGRANTFILE_API_VERSION = '2'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = 'hashicorp/precise32'
end
```

Vagrantfile = Ruby

. . .

This is Ubuntu 12.04 LTS 32 bit,

Let's say we want CentOS 6.5 64 bit

Finding base boxes

- https://vagrantcloud.com/ (since 1.5)
- http://vagrantbox.es/ (pre-1.5 boxes)

Using another base box

From the command line (Vagrant cloud):

```
$ vagrant init alphainternational/centos-6.5-x64
```

. . .

From the command line (“old”, pre-1.5 style):

```
$ vagrant box add --name centos65 \
  http://packages.vstone.eu/vagrant-boxes/centos-6.x-64bit-latest.box
$ vagrant init centos65
```

. . .

In your Vagrantfile (only applies to “old” style):

```
VAGRANTFILE_API_VERSION = '2'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = 'centos65'
  config.vm.box_url =
    'http://packages.vstone.eu/vagrant-boxes/centos-6.x-64bit-latest.box'
end
```

Applying the change

```
$ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
$ vagrant up
[...]
$ vagrant ssh
```

Configuring the VM

(checkpoint-02)

```
1  VAGRANTFILE_API_VERSION = '2'
2
3  HOST_NAME = 'box001'
4
5  Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
```

```
  config.vm.hostname = HOST_NAME
  config.vm.box = 'alphainternational/centos-6.5-x64'
  config.vm.network :private_network,
    ip: '192.168.56.65',
    netmask: '255.255.255.0'

  config.vm.provider :virtualbox do |vb|
    vb.name = HOST_NAME
    vb.customize ['modifyvm', :id, '--memory', 256]
  end
end
```

Configuring the VM

For more info,

- see the docs at https://docs.vagrantup.com/
- or the default `Vagrantfile`

Applying changes

When you change the `Vagrantfile`, do:

```
$ vagrant reload
```

Or, if the change is profound:

```
$ vagrant destroy -f
$ vagrant up
```

Setup with multiple VMs

Vagrantfile:

```
config.vm.define HOST_NAME do |node|
  node.vm.hostname = HOST_NAME
  [...]
end
```

Specify `HOST_NAME` after `vagrant` command:

```
$ vagrant status     # Status of *all* boxes
$ vagrant up box001  # Boot box001
$ vagrant up         # Boot *all* defined boxes
$ vagrant ssh box001
```

Setup with multiple VMs: Example

(checkpoint-03)

```
VAGRANTFILE_API_VERSION = '2'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  config.vm.define 'box001' do |node|
    node.vm.hostname = 'box001'
    node.vm.box = 'alphainternational/centos-6.5-x64'
    node.vm.network :private_network,
      ip: '192.168.56.65',
      netmask: '255.255.255.0'

    node.vm.provider :virtualbox do |vb|
      vb.name = 'box001'
    end
  end
```

Setup with multiple VMs: Example (cont'd)

```
  config.vm.define 'box002' do |node|
    node.vm.hostname = 'box002'
    node.vm.box = 'alphainternational/centos-6.5-x64'
    node.vm.network :private_network,
      ip: '192.168.56.66',
      netmask: '255.255.255.0'

    node.vm.provider :virtualbox do |vb|
      vb.name = 'box002'
    end
  end
end
```

Setup with multiple VMs: Example (cont'd)

Don't repeat yourself! (checkpoint-04)

```
hosts = [ { name: 'box001', ip: '192.168.56.65' },
          { name: 'box002', ip: '192.168.56.66' }]

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  hosts.each do |host|
    config.vm.define host[:name] do |node|
      node.vm.hostname = host[:name]
      node.vm.box = 'alphainternational/centos-6.5-x64'
      node.vm.network :private_network,
        ip: host[:ip],
        netmask: '255.255.255.0'
      node.vm.provider :virtualbox do |vb|
```

```
13          vb.name = host[:name]
14        end
15      end
16    end
17  end
```

Summary

```
$ vagrant init user/box   # Create Vagrantfile for specified base box
$ vim Vagrantfile         # Customize your box
$ vagrant up [host]       # Create VM(s) if needed and boot
$ vagrant reload [host]   # After every change to Vagrantfile
$ vagrant halt [host]     # Poweroff
$ vagrant destroy [host]  # Clean up!
$ vagrant ssh [host]      # log in
$ vagrant status [host]   # Status of your VM(s)
```

Provisioning

Provisioning

= From Just Enough Operating System to fully functional configured box

- **Shell script**
- **Ansible**
- **Puppet** (Apply + Agent)
- Chef (Solo + Client)
- Docker
- Salt

Shell provisioning

Shell provisioning

Add to your Vagrantfile

```
config.vm.provision 'shell', path: 'provision.sh'
```

Put the script into the same folder as `Vagrantfile`

Recommended workflow

- First do the installation manually (`vagrant ssh`)
- Make sure every command runs without user interaction!
- Record every command in the script
- If everything works: `vagrant destroy -f && vagrant up`

Provisioning script

(checkpoint-05)

Installs Apache and PHP

```
#!/bin/bash -eu
# provision.sh -- Install Apache and a test PHP script

sudo rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
yum install -y httpd php

service httpd start
chkconfig httpd on

cat > /var/www/html/index.php << EOF
<?php phpinfo(); ?>
EOF
```

MySQL is left as an exercise for the reader ;-)

Synced folders

(checkpoint-06)

- Add to your `Vagrantfile`:

  ```
  config.vm.synced_folder 'html', '/var/www/html'
  ```

- Create folder `html` in your project root

  ```
  $ tree
  .
  |-- html
  |   `-- index.php
  |-- provision.sh
  `-- Vagrantfile
  ```

- `Vagrant reload`

Disadvantages of shell provisioning

- Not very flexible
- Script should be non-interactive
- Not scalable
  - Long Bash scripts are horrible!
- Idempotence not guaranteed
  - What happens when you run provision script multiple times?
  - Change to script is expensive: `vagrant destroy && vagrant up`

Provisioning with Ansible

Ansible

http://ansible.com/

- Configuration management tool written in Python
- Simple configuration (YAML)
- No agent necessary (but recommended for large setups)
- Idempotent

. . .

(of course, you know this, you went to the talks yesterday…)

Vagrant configuration

```
config.vm.define 'box001' do |node|
  [...]
  node.vm.provisioning 'ansible' do |ansible|
    ansible.playbook = 'ansible/site.yml'
  end
end
```

Pro tips:

- `define` directive is important to make automatic inventory work
  - See Vagrant/Ansible documentation
- try to mimic standard Ansible directory structure
  - See Ansible best practices

Let's build a LAMP stack!

First, on one box

Then, database on a separate machine

Vagrantfile

(checkpoint-07)

```
1 VAGRANTFILE_API_VERSION = '2'
2 hosts = [ { name: 'box001', ip: '192.168.56.65' },
3           { name: 'box002', ip: '192.168.56.66' } ]
4
5 Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
6   config.vm.box = 'alphainternational/centos-6.5-x64'
7   hosts.each do |host|
```

```
    config.vm.define host[:name] do |node|
      node.vm.hostname = host[:name]
      node.vm.network :private_network,
        ip: host[:ip],
        netmask: '255.255.255.0'
      node.vm.synced_folder 'html', '/var/www/html'

      node.vm.provider :virtualbox do |vb|
        vb.name = host[:name]
      end

      node.vm.provision 'ansible' do |ansible|
        ansible.playbook = 'ansible/site.yml'
      end
    end
  end
end
```

Ansible project structure

```
$ tree ansible/
ansible/
|-- group_vars
|   `-- all
|-- roles
|   |-- common
|   |   `-- tasks
|   |       `-- main.yml
|   |-- db
|   |   `-- tasks
|   |       `-- main.yml
|   `-- web
|       `-- tasks
|           `-- main.yml
`-- site.yml
```

Main Ansible config file: `site.yml`

```
---
- hosts: box001
  sudo: true
  roles:
    - common
    - web
    - db
```

Common role

```
---
# file common/tasks/main.yml
```

```
- name: Install base packages
  yum: pkg={{item}} state=installed
  with_items:
    - libselinux-python
```

Web role

```
---
# file web/tasks/main.yml
- name: Install Apache
  yum: pkg={{item}} state=installed
  with_items:
    - httpd
    - php
    - php-xml
    - php-mysql

- name: Start Apache service
  service: name=httpd state=running enabled=yes
```

Db role

```
---
# file db/tasks/main.yml
- name: Install MySQL
  yum: pkg={{item}} state=installed
  with_items:
    - mysql
    - mysql-server
    - MySQL-python

- name: Start MySQL service
  service: name=mysqld state=running enabled=yes

- name: Create application database
  mysql_db: name={{ dbname }} state=present

- name: Create application database user
  mysql_user: name={{ dbuser }} password={{ dbpasswd }}
                priv=*.*:ALL host='localhost' state=present
```

Variables

```
---
# file group_vars/all

# Application database
dbname: appdb
dbuser: appusr
dbpasswd: CaxWeikun6
```

Workflow

1. Write `Vagrantfile`
    - `vagrant up` and `vagrant reload` until you get it right
2. Write configuration
    - `vagrant provision` until you get it right
3. Think you're done?
    - `vagrant destroy -f` and `vagrant up`

Install a webapp

E.g. Mediawiki

1. Unpack latest mediawiki.tar.gz into `html/wiki/` directory
2. Surf to http://192.168.56.65/wiki and follow instructions
3. Enter values from `group_vars/all` in the install page
4. Download `LocalSite.php` and save in `html/wiki/`

Automating Mediawiki installation is left as an exercise to the reader... ;-)

How to use this for production

Inventory file, automatically created by Vagrant:

```
$ cat .vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory
# Generated by Vagrant

box001 ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222
box002 ansible_ssh_host=127.0.0.1 ansible_ssh_port=2200
```

In production, just use a different inventory file!

Move database to another box

(`checkpoint-08`)

What should change?

. . .

```
---
# file site.yml
- hosts: box001
  sudo: true
  roles:
    - common
    - web
```

```
- hosts: box002
  sudo: true
  roles:
    - common
    - db
```

Move database to another box (cont'd)

What should change?

```
---
# db/tasks/main.yml
[...]
- name: Create application database user
  mysql_user: name={{ dbuser }} password={{ dbpasswd }}
              priv=*.*:ALL host='%' state=present
```

This should be easy to automate

Provisioning with Puppet

Puppet

http://puppetlabs.com/

- One of the market leaders in configuration management
- Has its own configuration language
- Many reusable modules available
- Needs an agent on hosts under control
- Usually set up with a central server (puppet master)
- Puppet should be already on your base box!

. . .

Do I have to introduce Puppet at all?

Vagrant configuration

```
config.vm.define HOST_NAME do |node|
  node.vm.synced_folder 'puppet', '/etc/puppet'
  node.vm.provision 'puppet' do |puppet|
    puppet.manifests_path = 'puppet/manifests'
    puppet.manifest_file = 'site.pp'
  end
end
```

Pro tips:

- The `synced_folder` directive makes Puppet “just work”
  - No other directives needed (e.g. `module_path`, `manifest_path`)
  - Installing files outside of modules
  - Same `hiera.yml` for Vagrant and production
  - Easier to reuse in production environment
- Mimic Puppet directory structure and best practices

Let’s build a LAMP stack!

Vagrantfile

(checkpoint-09)

```
VAGRANTFILE_API_VERSION = '2'
HOST_NAME = 'box001'
DOMAIN = 'example.com'
HOST_IP = '192.168.56.65'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = 'alphainternational/centos-6.5-x64'
  config.vm.define HOST_NAME do |node|
    node.vm.hostname = "#{HOST_NAME}.#{DOMAIN}"
    node.vm.network :private_network,
      ip: HOST_IP,
      netmask: '255.255.255.0'
    node.vm.synced_folder 'html', '/var/www/html'
    node.vm.synced_folder 'puppet', '/etc/puppet'
```

Vagrantfile (cont’d)

```
    node.vm.provider :virtualbox do |vb|
      vb.name = HOST_NAME
      vb.customize ['modifyvm', :id, '--memory', 256]
    end

    node.vm.provision 'puppet' do |puppet|
      puppet.manifests_path = 'puppet/manifests'
      puppet.manifest_file = 'site.pp'
    end
  end
end
```

Puppet project structure

```
$ tree -I modules --prune puppet/
puppet/
|-- manifests
|   |-- nodes
|   |   |-- box001.pp
```

```
|   |   '-- default.pp
|   '-- site.pp
'-- Puppetfile
```

Main Puppet files

```
# file manifests/site.pp

# Load node definitions
import 'nodes/*'


# file manifests/nodes/default.pp

node default {
  notice("I'm node ${::hostname} with IP ${::ipaddress_eth1}")

}
```

Managing 3rd party modules

Here, we use librarian-puppet

```
# Puppetfile -- Configuration for librarian-puppet
# Bootstrap by running 'librarian-puppet init'

forge "http://forge.puppetlabs.com"

mod "puppetlabs/stdlib"
mod "puppetlabs/concat"

mod "puppetlabs/apache"
mod "puppetlabs/mysql"
```

Working with Git submodules is also common, e.g.

```
$ git submodule add git@github.com:puppetlabs/puppetlabs-mysql.git modules/mysql
$ cd modules/mysql
$ git checkout tags/2.2.3
```

Definition of `box001`

```
# file manifests/nodes/box001.pp

node box001 inherits default {
  # Apache and PHP
  class { 'apache': }
  class { 'apache::mod::php': }

  package { [ 'php-mysql', 'php-xml' ]:
```

```
    ensure => installed,
  }

  # MySQL
  include '::mysql::server'

  mysql::db { 'appdb':
    user     => 'dbusr',
    password => 'vaygDeesh1',
    host     => 'localhost',
  }

}
```

Development vs Production

(checkpoint-10)

How to handle differences between development and production?

Puppet's answer: Hiera

Hiera configuration

puppet/hiera.yaml:

```
---
:backends:
  - yaml
:hierarchy:
  - '%{::environment}/%{::clientcert}'
  - 'common'
:yaml:
  :datadir: '/etc/puppet/hiera'
```

```
$ tree puppet/hiera
puppet/hiera
|-- common.yaml
|-- development
|   `-- box001.example.com.yaml
`-- production
    `-- box001.example.com.yaml
```

Hiera data

```
---
# file hiera/common.yaml
mysql::host: localhost
```

```
---
# puppet/hiera/development/box001.example.com.yaml
```

```
mysql::appdb: 'appdb'
mysql::user: 'dbusr'
mysql::password: 'letmein'


---
# file puppet/hiera/production/box001.example.com.yaml
mysql::appdb: 'db72437'
mysql::user:  'u440380'
mysql::password: 'ifwoHaffEtHafwivIj7'
```

Using Hiera data

`Vagrantfile:`

```
node.vm.provision 'puppet' do |puppet|
  puppet.manifests_path = 'puppet/manifests'
  puppet.manifest_file = 'site.pp'
  puppet.options = [ '--environment development' ]
end
```

`puppet/manifests/nodes/box001.pp`

```
  $appdb = hiera('mysql::appdb')

  mysql::db { $appdb:
    user     => hiera('mysql::user'),
    password => hiera('mysql::password'),
    host     => hiera('mysql::host'),
  }
```

Best practices

Best practices

- Follow guidelines of CfgMgmt tool
  - so you can use your box outside of Vagrant
- Keep `Vagrantfile` minimal
  - change `Vagrantfile` => `vagrant reload`
  - more expensive than `vagrant provision`

`Vagrantfile` bloat

```
1  # Enable provisioning with chef solo
2  config.vm.provision :chef_solo do |chef|
3    chef.cookbooks_path = "cookbooks"
4    chef.add_recipe "yum"
5    chef.add_recipe "yum::epel"
6    chef.add_recipe "openssl"
```

```
7       chef.add_recipe ”apache2”
8       chef.add_recipe ”apache2::default”
9       chef.add_recipe ”apache2::mod_ssl”
10      chef.add_recipe ”mysql”
11      chef.add_recipe ”mysql::server”
12      chef.add_recipe ”php”
13      chef.add_recipe ”php::module_apc”
14      chef.add_recipe ”php::module_curl”
15      chef.add_recipe ”php::module_mysql”
16      chef.add_recipe ”apache2::mod_php5”
17      chef.add_recipe ”apache2::mod_rewrite”
18      chef.json = {
19          :mysql => {
20                :server_root_password => ’root’,
21                :bind_address => ’127.0.0.1’
22          }
23      }
24    end
```

Creating base boxes

Creating base boxes

Sometimes, the available base boxes just aren’t good enough…

Manually

1. Create a VM, and take some requirements into account
    - a.o. `vagrant` user with sudo, ssh, package manager, Guest Additions
    - if you want: Puppet, Chef, …
2. Execute `vagrant package –base my-vm`
    - Result: file `my-vm.box`

Disadvantages

- It’s manual
- Not quite reproducable for other provider (e.g. VMWare, Hyper-V, bare metal)

Enter Packer

http://www.packer.io/

> Packer is a tool for creating identical machine images for multiple platforms from a single source configuration.

Packer template

- JSON file with settings
  - e.g. ISO download URL, VM type, provisioner
- Kickstart file
  - Automates installation from ISO
- Post-installation scripts
  - e.g. Configure for Vagrant, install Puppet, clean up yum repository, zerodisk (smaller disk images)
- Find loads of Packer templates at https://github.com/misheska/basebox-packer
  - Cr*p, only for Chef & Salt…

That's it!

What I didn't cover

- Provisioning with Chef
- Security (SELinux, firewall)
- Testing

Thank you!

Presentation slides: https://github.com/bertvv/vagrant-presentation

Code: https://github.com/bertvv/vagrant-example

More at:

https://github.com/bertvv/ https://bitbucket.org/bertvanvreckem/ https://www.youtube.com/user/bertvvrhogent/

@bertvanvreckem

Figure 1: CC-BY