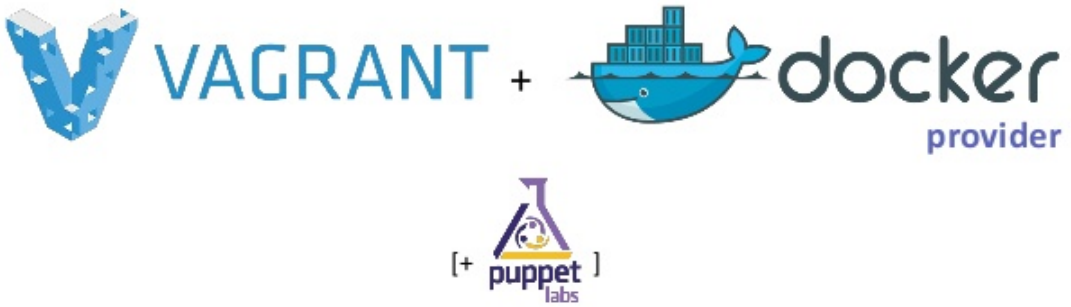
Building consistent Development Environments using Docker and Vagrant

Urs Oberdorf (141627) and Jan Kerkenhoff (141628)

November 25, 2014



Abstract

Supplying and managing consistent development environments across several different devices and different operating systems is a pain. Using Vagrant and Docker we set out to give an easy, consistent and scale able solution offering pre-configured Vagrant Images that contain the whole development machine, including docker containers to model the production environment you develop against. Only downside of our approach is the fact that developing now takes place in the virtual machine and not the developer machines native operating system

Contents

1 Introduction

One of the most annoying bug's while developing are always the "works on my machine" bugs, you spent numerous hours troubleshooting some developers environmental problems with mostly no added value to you product. But isn't there an easier way to handle this? There is, using to give every developer the same environment just the way you want it. But simply using virtual machine images produces a lot of overhead each time changes to the environment need to be made. Every change requires sending out several hundreds megabytes to every developer, while this can be done at a reasonable speed via internal networks, you are out of luck if your project span the entire globe and developers do not have access to Gigabit speeds anymore. To overcome this additional flaw of Virtual Machines (VMs) we set out to find a way to create and agile development environment work flow that does not require huge amounts of data to be send for every change. We finally settled on a combination VMs managed by Vagrant and Puppet as provisioner using VirtualBox as the provider and additionally Docker to handle the application environment inside the VM. While our system does not get rid of downloading a base VM image and Docker containers, this step is only required once.

2 Project proposal (SLA)

Our plan is to show how to use Vagrant to set up consistent development environments. We will show this for the case of web development. We also combine Vagrant with Docker to create a more production like environment, with separate instances and IP's for web server, database and other services. Using Docker we minimize the performance impact of each development environment. Since we run Docker containers instead of additional VM's for web server and databases, we save valuable resources on the developers system. Because Docker containers share resource with the host operating system they run on, it is possible to run more than one development environment at the same time. Also Vagrant reduces the effort to get the whole environment up and running to a single Vagrant up.

We will provide a development environment with the following software:

- Chrome, Opera and Firefox
- git
- Atom, Vim, Sublime as editors
- Ruby and Python
- Wireshark for debugging
- Tools for mysql(mysql-workbench,mysql-client)
- Filezilla for FTP
- htop for monitoring

Following Docker containers will also run in the Vagrant provided VM:

- Webserver (apache)
- Database (mysql)

3 Survey of similar projects

We looked for similar projects and while we found many projects that setup development environments using Vagrant and Puppet, none of them used Vagrant to create the VM your actually developing in. The normal approach with Vagrant is to use to create your production environment and then develop on your local machine. There exists some projects which aim to setup the local developer machines, most promising looking are Boxen by Github for Mac OS X and Boxstarter for Windows. There are even web services aimed at creating environments using Puppet and Vagrant, most notably is PuPHPet, which offers a sleek web GUI to setup web server images the way you want them to be.

4 Actual Project

4.1 Creating a base Image for Vagrant

Vagrant uses so called "boxes", which are basically VM images which are used as a base for the VM Vagrant creates, to setup the environment. There are already pre build boxes available for almost all operating systems, but they are all designed to be run headless and support no Graphical Interface. Since we wanted to go one step further and use Vagrant to create a virtual machine in which our developers would work, we need to create our own Ubuntu box. Creating a boxes with Vagrant is really simple you just to the following steps:

- Install your Operating system of choice in the provider of choice (Ubuntu and VirtualBox in our case)
- Install Puppet and every other thing you need to provision your VM (eg. Virtualbox guest additions)
- Add Vagrant user and copy over Vagrant SSH keys and enable password less sudo
- run `Vagrant package --base my-virtual-machine}` to let Vagrant pack the VM into a baseball

Full guide can be found here : `https://docs.Vagrantup.com/v2/VirtualBox/boxes.html`

4.2 Creating the environment using Vagrant

Vagrant uses our previous created box to set up the Ubuntu VM with everything we want to include. When we call `Vagrant up` it imports the specified base box and starts it, Vagrant then also takes care of mounting shared folder inside the VM, some of them are specified and some are automatically mounted like the ones needed for the Puppet run. When the machine successfully started, Puppet is run with our manifest to provision the VM with everything we want. In order to do this, Puppet installs the repositories needed for the browsers. Afterwards Puppet installs the packages for the browsers and tools from the repositories. After that Puppet takes care of Docker, by installing the package and downloading the Docker container images onto the host. Afterwards Puppet runs the Docker containers and the application runtime, in our case a simple html+php+mysql web

server, is ready to accept connections.After initially using `exec` commands in Puppet to add the repositorys to Ubuntu, we switched to the more elegant way of making use of the Puppet APT module. Using the Puppet APT module we added all the needed repositorys needed for the software we wanted to install so Puppet can handle them via the package manager. Most of the tools were already available in the standard repository shipping with Ubuntu, but some like the web browsers or Atom and Sublime required others. While working with the machine we noticed that the standard values for Vagrant VMs are not going to cut it in terms of memory requirements and CPU power. To combat this problem we simply added the following two lines in the Vagrant file to increase the memory to one Gigabyte and the CPU cores to two.

```
vb.memory = 1024
vb.cpus = 2
```

4.3 Using Docker to provide application run time environment inside the VM

Since we decided to run our web server and mysql server in Docker containers and not the actual VM, we looked for the best way this would be accomplished. We found the approved Docker module in the Puppet forge which made it really easy to manage Docker containers via Puppet. All we needed was to add

```
include Docker
```

followed by some

```
Docker::image
```

and

```
Docker::run
```

types to get our two containers up and running.The full sources are available in the appendix and on our github repository. The containers are linked to each other so that the web server can access the DB, without the need for exposed ports on the DB server itself. Also we shared the `www` directory into the web server container so that it can instantly start to serve the web page the developer created.

After having a enormous struggle to find a way to properly manage the Docker containers in regards to restarts or shutdowns, since using the standard Docker commands did not work in their expected way. We finally figured out the module generates and uses operating system build-in service files to manage the Docker containers and every container resembles its own service. This was one of the reasons why we couldn't properly shutdown the Docker containers, since the services were started via Upstart and init scripts they automatically restarted every time we shut them down. After this discovery we tried managing the containers via the `service` command, which yielded much better results. It took several hours to troubleshoot these issues which was partially caused by the very sparse documentation of the module. Some issues remain with the Docker module, when the web server get's killed and the mysql server is still running, its not possible for the module to start it again. For some reason the container exits right after starting. Using a manual run command gets the server running again so there seems to be some kind of problem with the init scripts the module is creating.

4.4 Welcome Module

To ease up the usage on start, we created a welcome module in puppet. The module ensures the existance of two files on the system. One is a html file, filled with some helpful usage commands and a list of the installed software. The other one is a simple shellscript which gets placed on the desktop and emerges the firefox browser on execution. The templates for the script and the html page can be found in the modules template folder. Due to a unpredictable delay on startup of the web server, we disagreed with the idea of putting the script on autostart.

5 Conclusions

We achieved all the things we planned for in a very efficient matter, the combination of Puppet and Vagrant helps so to streamline many things and the automation is really good. The enables you to scale this solution easily to several hundreds of workstations without any additional work. There are still some limitations when it comes to required hardware with our approach instead of the traditional Vagrant work flow of developing on your own machine and using Vagrant just for your emulating the production environment. The usage of Docker allowed us to reduce the hardware requirements significantly, since we are able to run the servers inside our VM and this way creates almost no overhead which allowed us to keep the requirements at a reasonable level. We also had to learn that using premade modules for Puppet always has its catch, on the one side they provide a fast solution to your problem, but in our case we spend a lot of time getting it to run they way we needed to and some issues are still present. Since the documentation of the module is kind of very parse its hard to determine if this are bugs or just desired behavior of the module. After all you can say that the combination of Vagrant is very powerful to make it easy to model the production environments of a company. We believe Docker and Vagrant have a bright future ahead, making the lives of system administrators and developers easier and give them more time to focus on the important things. Only downside of our approach is that we use an VM to develop in, this creates some performance drawbacks if computation intensive tasks need to be done regularly during the development of the application.

6 User Hand Book

This section aims to show how to setup our Vagrant environment so you can test and use it in the way intended by our project. This project requires the following things :

- VirtualBox, we tested this setup with 4.3.2 but newer versions should work too
- Vagrant, we used version 1.6.5 but newer versions should work too
- machine powerful enough to run a VM with at least 1GB of memory
- Git to download the project files
- Internet access to download the Vagrant base box

After you have installed Vagrant and Virtual box the first step is to download our project files from our project repository to a directory of your choice. After opening a shell in the directory where you would like to setup our project run:

```
git clone git@github.com:kerko/sysadm-project.git
```

The next last step takes the longest, go into your vagrant directory and just do:

```
vagrant up
```

Vagrant now takes care of everything, imports the base box, including downloading it from Amazon S3, creating the VM and running puppet to provision everything. The first run takes something between 20-40 minutes depending on your internet connection since puppet downloads the docker images the first time its run and also installs all the software.

7 Appendix

The full sources are also available at https://github.com/kerko/sysadm-project

7.1 Vagrantfile

```
VAGRANTFILE_API_VERSION = "2"

# SyncFolder on Host ist $HOME/vagrantWebDev
SYNCFOLDER_ROOT = [ENV['HOME'],"/vagrantWebDev"].join()

SYNCFOLDER_HOST_SHARE = SYNCFOLDER_ROOT + "/share"
SYNCFOLDER_GUEST_SHARE = "/home/vagrant/share"

SYNCFOLDER_HOST_WWW = SYNCFOLDER_ROOT + "/www"
SYNCFOLDER_GUEST_WWW = "/home/vagrant/src"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "webDev"
  config.vm.box_url = "https://s3.eu-central-1.amazonaws.com/vagrantsysadm/package.box"
  config.vm.synced_folder SYNCFOLDER_HOST_SHARE, SYNCFOLDER_GUEST_SHARE, create: true
  config.vm.synced_folder SYNCFOLDER_HOST_WWW, SYNCFOLDER_GUEST_WWW, create: true

  config.vm.provider "virtualbox" do |vb|
     # Start with Gui
     vb.gui = true
     # Set hostname
     vb.name = "webDevMachine"
     # Set RAM and CPU count
     vb.memory = 1024
     vb.cpus = 2
  end

   # Puppet
  config.vm.provision "puppet" do |puppet|
    puppet.options = "--verbose --debug"
    puppet.manifests_path = "../puppet/manifests"
    puppet.manifest_file  = "default.pp"
    puppet.module_path = "../puppet/modules"
  end


end
```

7.2 Puppetfiles

We used the `default.pp` as the main Puppet file run by Vagrant. Also we created some very simple modules for some thing like the bowsers that required more than

a few Puppet statements. Our self created modules are attached here and we also added a list of all modules from the Puppet forge we used.

7.2.1 default.pp

```
# Puppet Manifest for Web Development Tools

# Tools
$tools = ['git' , 'ruby', 'python', 'wireshark', 'filezilla',
          'mysql-workbench','mysql-client','htop']
package { $tools: ensure => 'latest' }

#Editors
package { 'vim': ensure => 'latest' }
include atom
include sublime

# Browsers
package { 'firefox':
  ensure => 'latest'
}

include googlechrome
include opera

### Add Docker and Containers ####

include docker

# Docker volume folders
$ubuntu_syncFolder = '/home/vagrant/src'
$dockercontainer_apache_syncFolder = '/var/www/html'
$volumes_apache = "${ubuntu_syncFolder}:${dockercontainer_apache_syncFolder}"

## apache docker container

docker::image{'php':
  image_tag => 'apache',
  require   => CLASS['docker'],
}

## mysql docker container
docker::image{'mysql':
  require => CLASS['docker']
}

docker::run { 'mysql':
  image    => 'mysql',
  use_name => true,
```

```
45    # Must be set, otherwise SQL server wont run
46    env       => 'MYSQL_ROOT_PASSWORD=abc',
47  }
48
49  docker::run { 'webServer':
50    image     => 'php:apache',
51    use_name  => true,
52    ports     => '80',
53    expose    => '80',
54    links     => ['mysql:db'],
55    volumes   => $volumes_apache,
56    require   => DOCKER::RUN['mysql'],
57  }
58
59  # WelcomePage
60  include welcome
```

7.2.2 Module manifests

Manifest to install googlechrome:

```
1   # == Class: googlechrome
2       #
3       # Module to add a repository that contains Chrome to ubuntu 14.04 and
4       # install the latest version of Chrome.
5       #
6       # === Examples
7       #
8       # include googlechrome
9       #
10      # === Authors
11      #
12      # Urs Oberdorf <urs.oberdorf@hig.no>
13      #
14      # === Copyright
15      #
16      # Copyright 2014 Urs Oberdorf.
17      #
18  class googlechrome (
19    $url_key  = 'https://dl-ssl.google.com/linux/linux_signing_key.pub',
20    $url_repo = 'http://dl.google.com/linux/chrome/deb/',
21  ){
22    include apt
23
24    apt_key { 'googleChrome_key':
25      ensure => 'present',
26      source => $url_key,
27      id     => '7FAC5991', # Last 8 digits of Fingerprint
28    }
29
```

```
  apt::source { 'googleChrome_repository':
    location    => $url_repo,
    release     => 'stable',
    repos       => 'main',
    include_src => false,
  }

  package { 'google-chrome-stable':
    ensure  => 'latest',
    require => [
                APT_KEY['googleChrome_key'],
                APT::SOURCE['googleChrome_repository'],
               ],
  }
}
```

Manifest to install opera:

```
# == Class: opera
    #
    # Module to add a repository that contains Opera to ubuntu 14.04 and
    # install the latest version of Opera.
    #
    # === Examples
    #
    # include opera
    #
    # === Authors
    #
    # Urs Oberdorf <urs.oberdorf@hig.no>
    #
    # === Copyright
    #
    # Copyright 2014 Urs Oberdorf.
    #
class opera (
  $url_key  = 'http://deb.opera.com/archive.key',
  $url_repo = 'http://deb.opera.com/opera/',
){
  include apt

  apt_key { 'opera_key':
    ensure => 'present',
    source => $url_key,
    id     => 'A8492E35',  # Last 8 digits of Fingerprint
  }

  apt::source { 'opera_repository':
    location    => $url_repo,
    release     => 'stable',
```

```
    repos       => 'non-free',
    include_src => false,
  }

  package { 'opera':
    ensure  => 'latest',
    require => [
                APT_KEY['opera_key'],
                APT::SOURCE['opera_repository'],
               ],
  }
}
```

Manifest to install atom:

```
# == Class: atom
    #
    # Module to add a repository that contains atom to ubuntu and install the
    # latest version of Atom.
    #
    # === Examples
    #
    # include atom
    #
    # === Authors
    #
    # Jan Kerkenhoff <jan.kerkenhoff@gmail.com>
    #
    # === Copyright
    #
    # Copyright 2014 Jan Kerkenhoff.
    #
class atom {
  include apt

  apt::ppa { 'ppa:webupd8team/atom': }

  package{'atom':
    ensure  => latest,
    require => Apt::Ppa['ppa:webupd8team/atom']
  }
}
```

Manifest to install sublime:

```
# == Class: sublime
    #
    # Module to add a repository that contains Sublime 3 to ubuntu 14.04 and
    # install the latest version of Sublime.
    #
    # === Examples
    #
```

```
8       # include sublime
9       #
10      # === Authors
11      #
12      # Jan Kerkenhoff <jan.kerkenhoff@gmail.com>
13      #
14      # === Copyright
15      #
16      # Copyright 2014 Jan Kerkenhoff.
17      #
18  class sublime {
19    include apt
20
21    apt::ppa { 'ppa:webupd8team/sublime-text-3': }
22
23    package{'sublime-text-installer':
24      ensure  => latest,
25      require => Apt::Ppa['ppa:webupd8team/sublime-text-3']
26    }
27  }
```

Manifest to add Welcome Page:

```
1   #
2   # == Class: welcome
3       #
4       # Module to create a welcome html page with user informations
5       #
6       # === Examples
7       #
8       # include welcome
9       #
10      # === Authors
11      #
12      # Urs Oberdorf <urs.oberdorf@hig.no>
13      #
14      # === Copyright
15      #
16      # Copyright 2014 Urs Oberdorf.
17      #
18  class welcome (
19    $mode                   = '0755',
20    $welcomePage_path       = '/home/vagrant/src/index.html',
21    $welcomePage_template   = 'welcome/welcomePage.html',
22
23    $welcomeScript_path     = '/home/vagrant/Desktop/welcome.sh',
24    $welcomeScript_template = 'welcome/welcomeScript.sh',
25  ) {
26
27    file { '/home/vagrant/src':
```

```
    ensure => directory,
    mode   => $mode,
  }

  file { $welcomePage_path:
    ensure  => present,
    content => template($welcomePage_template),
    mode    => $mode,
    require => FILE['/home/vagrant/src'],
  }

  file { $welcomeScript_path:
    ensure  => present,
    content => template($welcomeScript_template),
    mode    => $mode,
  }
}
```

Htmlpage template for welcome page

```
<!-- Static page with some help for using the web developer vagrant machine -->
<html>
  <body>
    <h1><center>Welcome web developer.</center></h1>

    <h2>Here are some helpful commands. </h2>

    <ul>
      <li>Show informations on the state of docker containers
        <ul><li>sudo docker ps -a</ul><br>
      <li>Show the IP address of the apache server
        <ul><li>"sudo docker inspect -f "{{.NetworkSettings.IPAddress}}" 'container name'"</
    </ul>

    <h2>A list of the installed software.</h2>
    <ul>
      <li>Chrome, Opera and Firefox
      <li> git
      <li> Atom, Vim, Sublime as editors
      <li> Ruby and Python
      <li> Wireshark for debugging
      <li> Tools for mysql( mysql-workbench,mysql-client)
      <li> Filezilla for FTP
      <li> htop for monitoring
    </ul><br>
    <h2>Following Docker containers will also run in the Vagrant provided VM</h2>
      <ul>
        <li> Webserver (apache)
        <li> Database (mysql)
      </ul>
```

```
31    </body>
32  </html>
```

bash template for welcome.sh

```
1  #!/bin/bash
2  # Welcome script for web developers
3
4  firefox http://$(sudo docker inspect -f "{{.NetworkSettings.IPAddress}}" webServer)
```

7.2.3 List of used modules from the Puppet forge:

Module name	Version	Author	URL
Docker	2.0.0	Gareth Rushgrove	https://forge.Puppetlabs.com/garethr/Docker
apt	1.7.0	Puppet Labs	https://forge.Puppetlabs.com/Puppetlabs/apt
Dependency modules			
epel	1.0.0	Michael Stahnke	https://forge.Puppetlabs.com/stahnma/epel
stdlib	4.4.0	Puppet Labs	https://forge.Puppetlabs.com/Puppetlabs/stdlib

Acronyms

VM Virtual Machine.

Glossary

box

is a base virtual machine used as the base for setting up new virtual machines, also called ”base box”.

provider

the program used by Vagrant to run the virtual machine, several options are available here.

provisioner

is a tool used to configure a virtual machine, so it gets into the state you want it to have. This includes installing software and changing configurations.

repository

a directory that contains software packages which can be installed on the system using a package manager.