Using Packer, Vagrant and Boxstarter to create Windows environments

Testing your application in different environments can be hard to do manually, as its time consuming, error prone, and not easily reproducible. This article will show you a way on how to automatically create Windows environments, where you can test your application.

**Tools**

You will use the following tools:

* [Packer](https://packer.io/) and [packer-windows](https://github.com/joefitzgerald/packer-windows) to create an up-to-date Windows base image from an ISO file.
* [Vagrant](https://www.vagrantup.com/) to launch and configure a base image tailored to a specific scenario.
* [Boxstarter](http://boxstarter.org/) to easily install software from [Chocolatey](https://chocolatey.org/) packages.
* [VirtualBox](https://www.virtualbox.org/) to run virtual machines.

**Install**

You will need to run several commands. These must be run on a bash shell, which I’ll assume that you have installed following the instructions from [Sane shell environment on Windows](https://blog.ruilopes.com/sane-shell-environment-on-windows.html).

Make sure you have previously installed Packer, Vagrant and VirtualBox and have them available on your PATH, e.g. with:

export PATH**=**"$PATH:/c/Development/packer"

export PATH**=**"$PATH:/c/Development/Vagrant/bin"

export PATH**=**"$PATH:/c/Program Files/Oracle/VirtualBox"

**Windows base image**

Packer is a tool to automatically create a machine image from a [template file](https://packer.io/docs/basics/terminology.html#Templates).

packer-windows is a repository with pre-configured template files for installing different Windows versions.

We will use them to drive the installation of Windows 2012 R2 into a virtual machine image. This image will have the bare minimum needed to be later used by Vagrant.

This base image will be automatically configured with:

* SSH server.
* [WinRM](https://msdn.microsoft.com/en-us/library/aa384426%28v=vs.85%29.aspx).
* vagrant user account with a vagrant password.
* Latest Windows updates.
* VirtualBox guest additions.
* 60 GB disk.

Start by getting the Windows template files:

git clone https://github.com/joefitzgerald/packer-windows.git

cd packer-windows

**NB** These templates are for the trial versions of Windows, namely Windows 2012 R2 Standard, but you can also [install from your own Windows ISO file](https://blog.ruilopes.com/using-packer-vagrant-and-boxstarter-to-create-windows-environments.html#install-from-your-own-windows-iso-file).

In order to makes things easier for us we will tweak a couple of files.

We want to see how the installation is going on, for that we need to change the template file to start the virtual machine in non-[headless mode](https://en.wikipedia.org/wiki/Headless_software). For that, edit the windows\_2012\_r2.json template file:

vim windows\_2012\_r2.json

...

{

"type"**:** "virtualbox-iso",

...

"headless"**:** **false**,

...

By default, the template is configured to install the latest Windows updates, but the default timeout might now be enough for the updates to finish in time, so increase the ssh\_timeout from the default 2h to 8h:

vim windows\_2012\_r2.json

...

{

"type"**:** "virtualbox-iso",

...

"ssh\_wait\_timeout"**:** "8h",

...

Also, sometimes, the [last provisioning step fails](https://github.com/joefitzgerald/packer-windows/issues/80), so you also need to change the line that has rm -rf to:

{

"type"**:** "shell",

"inline"**:** [

"rm -rf /tmp/\* || true"

]

}

Also consider removing the ./scripts/compact.bat (it takes ages to run) and the ./scripts/chef.bat (you might not need it) lines.

Finally, build the VirtualBox based image:

time packer build -only virtualbox-iso windows\_2012\_r2.json

Now be patient! It will download the Windows ISO file and install the latest Windows Updates. These steps take a lot of time to complete (about 2 hours in my environment). At the end you should have a .box file (about 1.8 GB), ready to be used by Vagrant.

Add the generated box file to vagrant:

vagrant box add --name windows\_2012\_r2 windows\_2012\_r2\_virtualbox.box

**NB** You can distribute the box file to other system, or you can delete it (its no longer needed once added to vagrant)

We are now almost ready to launch a Windows virtual machine with Vagrant, for that we need to create a Vagrantfile:

mkdir testing **&&** cd testing

cat**<<**"EOF"**>**Vagrantfile

*# -\*- mode: ruby -\*-*

*# vi: set ft=ruby :*

Vagrant**.**require\_version ">= 1.6.2"

Vagrant**.**configure("2") **do** **|**config**|**

config**.**vm**.**define "testing"

config**.**vm**.**box **=** "windows\_2012\_r2"

config**.**vm**.**provider :virtualbox **do** **|**v, override**|**

v**.**gui **=** **true**

v**.**customize **[**"modifyvm", :id, "--memory", 2048**]**

v**.**customize **[**"modifyvm", :id, "--cpus", 2**]**

**end**

**end**

EOF

And launch it:

time vagrant up

**NB** The initial run will take about 4m to boot into Windows. Later runs take about 40s. But YMMV.

**NB** To troubleshoot you need pass the --debug flag to vagrant up.

Once Windows boots, try it a bit to check whether things are working fine, and then stop the virtual machine with:

vagrant halt

And destroy it:

vagrant destroy

You are now ready to customize a vagrant environment to fit your needs. For that we will modify the Vagrantfile to use a provision script.

**Provisioning**

Customizing a vagrant environment is a mater of creating one or more provision scripts. Lets start simple, with one embedded within the Vagrantfile, it will just change the keyboard layout and timezone:

cat**<<**"EOF"**>**Vagrantfile

*# -\*- mode: ruby -\*-*

*# vi: set ft=ruby :*

Vagrant**.**require\_version ">= 1.6.2"

*# NB this is a PowerShell script that is run as Administrator.*

$root\_provision\_script **=** **<<**'ROOT\_PROVISION\_SCRIPT'

# set keyboard layout.

# NB you can get the name from the list:

# [System.Globalization.CultureInfo]::GetCultures('InstalledWin32Cultures') | out-gridview

Set-WinUserLanguageList pt-PT -Force

# set the date format, number format, etc.

Set-Culture pt-PT

# set the timezone.

# tzutil /l lists all available timezone ids

& $env:windir\system32\tzutil /s "GMT Standard Time"

ROOT\_PROVISION\_SCRIPT

Vagrant**.**configure("2") **do** **|**config**|**

config**.**vm**.**define "testing"

config**.**vm**.**box **=** "windows\_2012\_r2"

config**.**vm**.**provider :virtualbox **do** **|**v, override**|**

v**.**gui **=** **true**

v**.**customize **[**"modifyvm", :id, "--memory", 2048**]**

v**.**customize **[**"modifyvm", :id, "--cpus", 2**]**

**end**

config**.**vm**.**provision "shell", inline: $root\_provision\_script

**end**

EOF

And launch it:

vagrant up

If its running fine, stop the virtual machine with:

vagrant halt

Now its time to install applications. This is handled by Boxstarter. Lets modify the provision script to install Boxstarter and some applications:

vim Vagrantfile

**.**.**.**

$root\_provision\_script **=** **<<**'ROOT\_PROVISION\_SCRIPT'

...

# install Boxstarter.

# NB Do NOT install chocolatey before Boxstarter. If you do, strange things

# will happen...

# TODO this should all be abstracted in a "boxstarter" provisioner.

$boxstarterSetupPath = "$env:TEMP\Boxstarter-setup"

$boxstarterSetupZipPath = $boxstarterSetupPath + ".zip"

Invoke-WebRequest http://boxstarter.org/downloads/Boxstarter.2.4.209.zip -OutFile $boxstarterSetupZipPath

[System.Reflection.Assembly]::LoadWithPartialName("System.IO.Compression.FileSystem")

[System.IO.Compression.ZipFile]::ExtractToDirectory($boxstarterSetupZipPath, $boxstarterSetupPath)

& $boxstarterSetupPath\setup.bat -Force

echo NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB

echo NB

echo 'NB Boxstarter might need to reboot the machine, in that case, vagrant will'

echo 'NB fail, but that is expected. you need to monitor the install yourself to'

echo 'NB known when its done...'

echo NB

echo NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB

# NB if any of the choco packages need to access the desktop, you need to force

# a reboot at start of the boxstarter script. when you do that Boxstarter

# will run the script again, but in a auto-logon session. the easiest way to

# do that, is to install google chrome.

$env:PSModulePath = "$([System.Environment]::GetEnvironmentVariable('PSModulePath', 'User'));$([System.Environment]::GetEnvironmentVariable('PSModulePath', 'Machine'))"

cp C:\vagrant\setup.ps1 $env:TEMP

Import-Module Boxstarter.Chocolatey

$credential = New-Object System.Management.Automation.PSCredential("vagrant", (ConvertTo-SecureString "vagrant" -AsPlainText -Force))

Install-BoxstarterPackage $env:TEMP\setup.ps1 -Credential $credential

ROOT\_PROVISION\_SCRIPT

**.**.**.**

Create the script to automatically install some software:

cat<<"EOF">setup.ps1

*# NB this file has to be idempotent. it will be run several times if the computer needs to be restarted.*

*# when that happens, Boxstarter schedules this script to run again with an auto-logon.*

*# NB always remember to pass -y to choco install!*

*# NB already installed packages will refuse to install again; so we are safe to run this entire script again.*

*# NB make sure this is the first software you install, because, as a side*

*# effect, this will trigger a reboot, which in turn, will fix the vagrant bug*

*# that prevents the machine from rebooting after setting the hostname.*

choco install -y google-chrome-x64

*# Enable Show Window Contents While Dragging*

reg ADD "HKCU\Control Panel\Desktop" /v DragFullWindows /t REG\_SZ /d 1 /f

taskkill /IM explorer.exe /F ; explorer.exe

*# Pin an application to the task bar.*

Install-ChocolateyPinnedTaskBarItem "$env:windir\system32\services.msc"

choco install -y notepad2

choco install -y nodejs.install

choco install -y visualstudiocode

EOF

And launch it:

vagrant up

You will notice that vagrant up seems to fail with something like:

The following WinRM command responded with a non-zero exit status. Vagrant assumes that this means the command failed!

powershell -ExecutionPolicy Bypass -OutputFormat Text -file c:\tmp\vagrant-shell.ps1

But that’s expected, you need to monitor the install yourself to known when its done. Doing that automatically would be nice, but I’ll leave that as an exercise for the reader (hint: You could create a new boxstarter Vagrant provisioner).

As a side note, you could also manually provision the machine and later export it as a new box file, e.g.:

vagrant package --base testing --output testing.box

Install from your own Windows ISO file

The templates contained in the windows-packer repository are configured to install the trial versions of windows, but you can also install from a specific ISO file. For that, you need to set the ISO path and its MD5 checksum, e.g.:

vim windows\_2012\_r2.json

...

"iso\_url"**:** "d:/Images/windows\_2012\_r2.iso",

"iso\_checksum\_type"**:** "md5",

"iso\_checksum"**:** "458ff91f8abc21b75cb544744bf92e6a",

...

The iso\_checksum value can be computed with md5sum:

md5sum d:/Images/windows\_2012\_r2.iso

458ff91f8abc21b75cb544744bf92e6a \*d:/Images/windows\_2012\_r2.iso

If you don’t have a volume license ISO, you also need to set the Product Key. For that open the answer\_files/2012\_r2/Autounattend.xml file, search for ProductKey and follow the instructions.

The End

And thats it! You should now be able to create Windows environments to fit your needs.

You should also look into configuration management tools like [Ansible](https://www.ansible.com/), [Chef](https://www.chef.io/), and [Puppet](https://puppetlabs.com/). These might useful to actually install and configure your own application.