**Creating VMs with Packer and Vagrant**

[June 9, 2016](https://travislawrence.co/2016/06/09/creating-vms-with-packer-and-vagrant/) [tlawrence](https://travislawrence.co/author/tlaw2315/) [Code](https://travislawrence.co/category/code/), [Technology](https://travislawrence.co/category/technology/)

**Goal**

Utilize Packer and Vagrant with VMware Workstation on Windows to build and destroy Windows virtual machines in minutes.

**Overview**

1. Build a base VM programmatically with Packer
2. Package base VM as .box for use with Vagrant
3. Deploy VM in minutes with Vagrant to create test builds for infrastructure as code!
4. Suspend/DESTROY! (╯°□°)╯︵ ┻━┻ VM as needed for testing

**Process**

***Note***: This process uses the paid ($) [VMware Workstation Vagrant provider](https://www.vagrantup.com/docs/vmware/) rather than the standard, but free, VirtualBox provider. Though it should be easily modified for use with VirtualBox or other providers. In my experience Workstation is far superior to VirtualBox but YMMV.

Prerequisite: VMware Workstation

Install Packer via [Chocolatey](https://chocolatey.org/) (or manual installation)

cinst packer -y

Install Vagrant via [Chocolatey](https://chocolatey.org/) (or manual installation)

cinst vagrant -y

Install Git via [Chocolatey](https://chocolatey.org/) (or manual installation)

cinst git -y

Open a Windows command line (I prefer [cmder](http://cmder.net/)!) and create a new directory structure so we have a sandbox folder to play in.

cd \

mkdir Packer

cd Packer

mkdir win01

cd win01

We’ll use this awesome [packer-windows](https://github.com/StefanScherer/packer-windows) repo [Stefan Scherer](https://github.com/StefanScherer) has built for our Packer and Vagrant items to simplify the process for our first builds.

git clone <https://github.com/StefanScherer/packer-windows>

cd packer-windows

Build the base VM using Packer. We’ll build a Windows Server 2012 R2 box.

packer build --only=vmware-iso windows\_2012\_r2.json

The build process creates a VM using Packer, using an .iso and related Packer .json file, and then packages it as a Vagrant box. This process takes 30+ minutes. The box file contains the VMware Workstation required VM files (.vmx, .vmdk, etc.) along with a pre-created Vagrantfile which specifies how Vagrant builds the final VM. This also includes a ‘metadata.json’ file which tells us which Vagrant provider the box is for (Workstation in this case). If you are using a different provisioner you could specify a different ‘–only=’ flag.

Box files are .tar  files (and often also gzipped) that can be unzipped if you’d like to peer into the contents of what the Packer post processor result is for Vagrant. The ‘windows\_2012\_r2.json’ file “post-processors” section is where the Vagrant components are configured for Packer.

Box file contents Post-processors section in Packer .json file

Now that we have a completed box file we can add it to Vagrant. This isn’t necessarily required but makes this box available to multiple Vagrant systems. Vagrant will make a clone from the box file and create a VM from that clone giving us the flexibility to create multiple VMs from a single box.

Note the forward slashes in the box file path…yes on Windows.

|  |
| --- |
| vagrant box add win\_2012\_r2\_vmware c:/Packer/windows\_2012\_r2\_vmware.box |

Here we’ve provided a box name, use a standard that works for you, and the path to the box file we created with Packer. This path gives us the flexibility to store these large box files wherever we chose.

Now we run the ‘init’ command to provide Vagrant with the box file information and subsequently the Vagrantfile which is stored inside the box file.

Again, note the forward slashes in the file path.

vagrant init win01 C:/Packer/windows\_2012\_r2\_vmware.box

Finally we can run ‘vagrant up’ to build our VM. This clones the box file and automatically starts the VM in Workstation. This takes 3-4 minutes on my machine. Full Windows GUI VMs, as we have here, take a bit longer to deploy with Vagrant compared to Linux VMs due to their disk size. That changes with Server Core or Nano.

vagrant up

Now we can connect to the VM using RDP or PowerShell through Vagrant. You can also open the Workstation interface from the Workstation system tray icon to show the hidden VM by selecting “Open All Background Virtual Machines”. An easy way to test network connectivity without RDP is to connect via PowerShell and run test-netconnection.

***Note***: the default username/password for our build is…

username: **vagrant** password: **vagrant**

vagrant rdp

vagrant powershell

[192.168.72.147]: PS C:\Users\vagrant\Documents> **test-netconnection**

ComputerName : internetbeacon.msedge.net

RemoteAddress : 13.107.4.52

InterfaceAlias : Ethernet

SourceAddress : 192.168.72.147

PingSucceeded : True

PingReplyDetails (RTT) : 11 ms

As part of the ‘vagrant up’ process Vagrant has forwarded certain VM ports so we can access RDP and PowerShell.

Here we can test scripts, code, installing applications or Microsoft Desired State Configuration and when we’re done we can suspend or delete the VM. Best of all we can share the box file with team members and be assured we have the exact VM configuration for testing and development. Enjoy!

vagrant suspend

vagrant destroy