Vagrant Tutorial

Saturday, January 28, 2017 9:40 AM

Summary

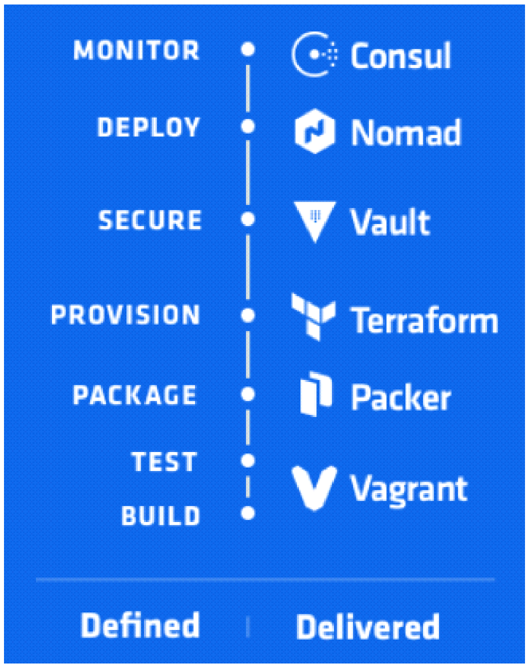
This tutorial is a guide to explain the use cases of Vagrant and Packer in regards to automation and enabling deployment of production environments to anyone in the organization who has a standard laptop. This enables testing, bug checking, development, and operations testing removing the need for multiple hardware requests.

What is the problem?
The problem is that we have high levels of computing available to us yet we are not really using any of it. Our own laptops are fully powered and used as primary workstations. Yet when it comes to developers writing code, QA testers, Operations and Security people, these resources are not being used--we have ad-hoc efforts to gain some hardware, any hardware, for testing. This leads to a fractured development and inconsistent experiences, which often lead to short cuts and compromises that your are forced to pay later.

Enter Mitchell Hashimoto and Hashicorp. They started in 2010 with a radical mission to automate everything they could into repeatable steps using single tools that handle abstractions. Hashicorp was platform agnostic and focused on the process of automation. They succeeded and are now considered the industry leaders in the arena. Quite frankly there are no other tools that even compete in this space.

https://www.hashicorp.com/files/DevOps-Defined.pdf <-- White paper on what problem we are trying to solve

They have a suite of tools at http://www.hashicorp.com but there are two programs we will primarily focus on.



Vagrant - https://vagrantup.com

Overview
Vagrant is a tool designed to abstract away desktop virtualization platforms. Most commonly used with Oracle's VirtualBox desktop program, it is not exclusive by a far reach to VirtualBox. Vagrant can operate as an interace natively to:

- VirtualBox
- Hyper-V (yes on a Windows System)
- Docker (yes it can interact with containers)
- Vmware (* there is a cost to purchase the plugin as Vmware desktop is not opensource - but 10% student discount)

This means you can interact with any of these back-ends using Vagrant commands - this streamlines your workflow allowing you to move to any platform and have the same commands.

Why Vagrant?

From - https://www.vagrantup.com/docs/why-vagrant/

How Vagrant Benefits You

If you are a **developer**, Vagrant will isolate dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of the tools you are used to working with (editors, browsers, debuggers, etc.). Once you or someone else creates a single Vagrantfile, you just need to vagrant up and everything is installed and configured for you to work. Other members of your team create their development environments from the same configuration, so whether you are working on Linux, Mac OS X, or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way. Say goodbye to "works on my machine" bugs.

If you are an **operations engineer**, Vagrant gives you a disposable environment and consistent workflow for developing and testing infrastructure management scripts. You can quickly test things like shell scripts, Chef cookbooks, Puppet modules, and more using local virtualization such as VirtualBox or VMware. Then, with the *same configuration*, you can test these scripts on remote clouds such as AWS or RackSpace with the *same workflow*. Ditch your custom scripts to recycle EC2 instances, stop juggling SSH prompts to various machines, and start using Vagrant to bring sanity to your life.

If you are a **designer**, Vagrant will automatically set everything up that is required for that web app in order for you to focus on doing what you do best: design. Once a developer configures Vagrant, you do not need to worry about how to get that app running ever again. No more bothering other developers to help you fix your environment so you can test designs. Just check out the code, vagrant up, and start designing.

From <https://www.vagrantup.com/docs/why-vagrant/>

Installation

Vagrant is a self-contained executable file that is compiled for multiple platforms:

- MacOS
- Windows
- Debian based Linux distros (including Ubuntu)
- Centos based Linux distros (including RedHat)

You need to download this executable, put it somewhere safe, and then add the path to the vagrant executable to your SYSTEM PATH. **NOTE** - do not install Vagrant from an LINUX package manager - they are woefully out of data not supported packages - go right to the source at Vagrantup.com

SYSTEM PATH EDITING
MacOS
There is an unnofficial package manager maintained by the community called HOMEBREW - http://brew.sh/ - the reason why is MacOS won't add any software that is licensed under GPLv3+ so this essentially is the tool used to get (Bash 4.x) more recent software onto the Mac.

Following the instructions here http://sourabhbajaj.com/mac-setup/Vagrant/README.html - you can setup Vagrant on your Mac and the benefit of using this 3rd part package manager is that it will add the executable to your path. You can also skip the download step above as it will retrieve the executable for you.

Things to note
The version might be a bit behind, usually one or two points are ok. Also you can update Vagrant through homebrew as it is released.

https://coolestguidesontheplanet.com/add-shell-path-osx/ is a good guide to adding the Vagrant executable to your .bash_profile permanently

Windows

As above in the Mac section the best way to add the downloaded executable to your system path would be to use a 3rd party package manager (my recommended way). There is a quite excellent community managed tool called **Chocolatey** - https://chocolatey.org/ - this is a package manager similar to apt-get and rpm but made for Windows (similar to homebrew) to fill in the missing ability to automate package installs. You can install

Vagrant via a simple chocolatey command (after chocolatey is installed -- chocolatey required admin privileges). The command would be: `choco install vagrant` -- this command has the advantages of being easy to update as well as adding the vagrant executable to your system path.

Linux (all kinds)
This is a simple export exercise. Similar to the steps in the manual portion of the MacOS instructions, you simply need to add the directory location of the Vagrant executable to your PATH variable. This is done in your .bashrc file located in your home directory. Simply add a line similar to:
```
export PATH=$PATH:/path/to/vagrant
```

Vagrant Conventions - The box command

Vagrant provides a tutorial on https://www.vagrantup.com/docs/getting-started/

The term and concept you need to be familiar with is: ***.box** and **Vagrantfile**, these two components are what is needed to run and manage Vagrant boxes. You will interact with Vagrant boxes with the **box** command.

A Vagrant box is an abstraction (it's the collection of the components needed to run a virtual machine collected into a single file) For example a *.box file that was made for the Virtualbox provider would contain the *.vmdk (hard drive) and the *.ovf file (meta-data and Virtual Machine settings file)

There are two ways to obtain a Vagrant Box (*.box file). The first way would be to obtain pre-made images from a site you trust (remember you are potentially running other people's configuration and software in your place of work -- just be aware) The best most trustworthy place is from Vagrantup.com itself - https://atlas.hashicorp.com/boxes/search. Here you can search for boxes of other operating systems and versions even some opensource companies release a pre-configured Vagrant Box all setup for you to test their software all in one place. Using this facility you can simply run a command from the command line **to add** this box to your local system.

The second way to **is add** it manually

Adding Vagrant Boxes

When executing the `vagrant box` command from the command line (in Windows recommend using powershell) you will see this list of subcommands as the output:

```
 add
list
outdated
remove
repackage
update
```

Command: `vagrant box add`

The first command **add** is the command we will use to add boxes (either of the two methods) from above.

The tutorial on vagrantup.com will walk you through this but a small example (try any of these especially if you are not familiar with these platforms)

- `vagrant box add centos/7 (Official Centos 7 Vagrant box release)`
- `vagrant box add debian/jessie64 (Debian provided release of Jessie Debian 8 x64)`
- `vagrant box add terrywang/archlinux (user provided Arch Linux distro)`
- `vagrant box add concourse/lite   (concourse software package provided and preconfigured)`
- `vagrant box add freebsd/FreeBSD-11.0-CURRENT (official FreeBSD vagrant box)`
- `vagrant box add maier/alpine-3.4-x86_64  (user provided alpine Linux distro)`
- `vagrant box add ubuntu/xenial64 (Canonical--Ubuntu parent company - provided)`

You may need to use a full URL in the case of downloading a Vagrant box that is not provided from Hashicorp box repositories. This goes for 3rd party and for the boxes your create on your own. We will learn how to make our own in the **Packer.io** section of this document, but for all purposes the artifacts are the same; a ***.box** file. For installing a Devuan box (the distro that resulted from the Debian Civil War/systemd split) here are the commands:

(Command is all one line)
```
vagrant box add http://devuan.ksx4system.net/devuan_jessie_beta/devuan_jessie_1.0.0-beta_amd64
_vagrant.box --name alpine
```

NOTE*** - Adding a box via URL requires an additional parameter, `--name` (as seen above). The `--name` option is something you declare for your use, just don't put any spaces and its best to name the box something related to the actual box; `box1` or `thebox` are terrible names.

Command: `vagrant box list`

You can check to see if the `vagrant box add` command was successful by issuing the `vagrant box list` command-- looking something like this: (note this is my system yours will vary but the structure will be the same)

```
PS C:\Users\Jeremy\Documents\vagrant> vagrant box list
centos-vanilla-1611   (virtualbox, 0)
ubuntu-vanilla-16041 (virtualbox, 0)
ubuntu/trusty64      (virtualbox, 20161121.0.0)
ubuntu/xenial64      (virtualbox, 20170119.1.0)
```

Here you notice that the last two boxes were added directly from the Hashicorp boxes repository (`vagrant box add ubuntu/trusty64` and `vagrant box add ubuntu/xenial64`)

The top two boxes were custom Vagrant boxes I created (we are getting to that part) that are treated as third part boxes. To add them I issued a command like this: (The vanilla term is my own convention, it just means this is a default OS install -- no extra packages)

```
vagrant box add ./centos-vanilla-1611-server-virtualbox-1485312680.box --name centos-7-vanilla
vagrant box add ./ubuntu-vanilla-16041-server-virtualbox-1485314496.box --name ubuntu-16041-vanilla
```

Command: vagrant box remove

The same way that you add boxes you can remove them from your list. You need to know the name of the box that was added, run a `vagrant box list` command and find the name that way. The below commands would remove the boxes added in the previous section.

- `vagrant box remove centos-7-vanilla`
- `vagrant box remove ubuntu-16041-vanilla`

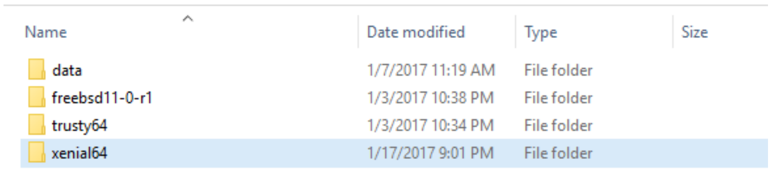
Command: vagrant init

Once your Vagrant boxes have been added to your system and Vagrant has them in a list, you can now create a `Vagrantfile`. You have one Vagrantfile per-Vagrant box. It would make the most sense to create a sub-directory with the same box name to house the Vagrantfile.

For instance if the output of the command:

```
PS C:\Users\Jeremy\Documents\vagrant> vagrant box list
centos-vanilla-1611   (virtualbox, 0)
ubuntu-vanilla-16041 (virtualbox, 0)
ubuntu/trusty64      (virtualbox, 20161121.0.0)
ubuntu/xenial64      (virtualbox, 20170119.1.0)
```

Then it would make sense to create a folder named after each of these boxes under the directory **vagrant**. NOTE - I arbitrarily created the directory **vagrant** under **Documents**. Why? It seems to make logical sense. See screenshot:



Name	Date modified	Type	Size
data	1/7/2017 11:19 AM	File folder	
freebsd11-0-r1	1/3/2017 10:38 PM	File folder	
trusty64	1/3/2017 10:34 PM	File folder	
xenial64	1/17/2017 9:01 PM	File folder	

Note--for good measure I added a directory called **data** which will be used for mounting shared drives--I will explain in a bit.

Once you have created these folders, `cd` into one. For instance take the **trusty64** and **xenial64**. You would `cd` into **trusty64** directory and type: `vagrant init trusty64`. This will create a file called `Vagrantfile` that points and works with the trusty64 vagrant box. The idea behind the `Vagrantfile` is that it has a shorthand syntax that is universally translated by Vagrant into specific virtualization platforms. The `Vagrantfile` handles all the properties that could be set (such as RAM, CPU, shared drives, port forwarding, networking, and so forth).

Here is a sample `Vagrantfile` from my system that was build via the `vagrant init` command for you

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "ubuntu/xenial64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
   config.vm.network "forwarded_port", guest: 8080, host: 8080

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
   config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
   config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
   config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
     vb.memory = "2048"
   end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
  # such as FTP and Heroku are also available. See the documentation at
  # https://docs.vagrantup.com/v2/push/atlas.html for more information.
  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # Enable provisioning with a shell script. Additional provisioners such as
  # Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

Command: `vagrant up`

Once your Vagrantfile has been created the next step to launch the virtual machine via Vagrant is through the `vagrant up` command. You would issue the command from the same directory where the `Vagrantfile` is located. A vagrant up command looks in the local directory for a `Vagrantfile` to begin parsing. This command is akin to starting the virtual machine directly. On the first run the Vagrantfile will be parsed and any settings in the virtual machine platform (Virtual Box in our case) will be changed. On subsequent runs the Vagrantfile will be ignored.
Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant up --provision`

The --provision flag tells Vagrant to re-provision and re-read and parse the Vagrantfile and make any additional changes while launching the virtual machine. Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant reload`

This is akin to a reboot or restart of a virtual machine. Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant reload --provision`

Will restart the system as well as re-read and parse the Vagrantfile. Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant suspend`

This will put the virtual machine in suspend or pause move (standby) as opposed to running vagrant halt, which will power the virtual machine off. Very handy to quickly resume work. Don't expect the system to automatically put your virtual machine into standby if you are used to just closing the lid of your laptop. Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant halt`

Full shutdown of the virtual machine (power off). Note - This command is issues not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant ssh`

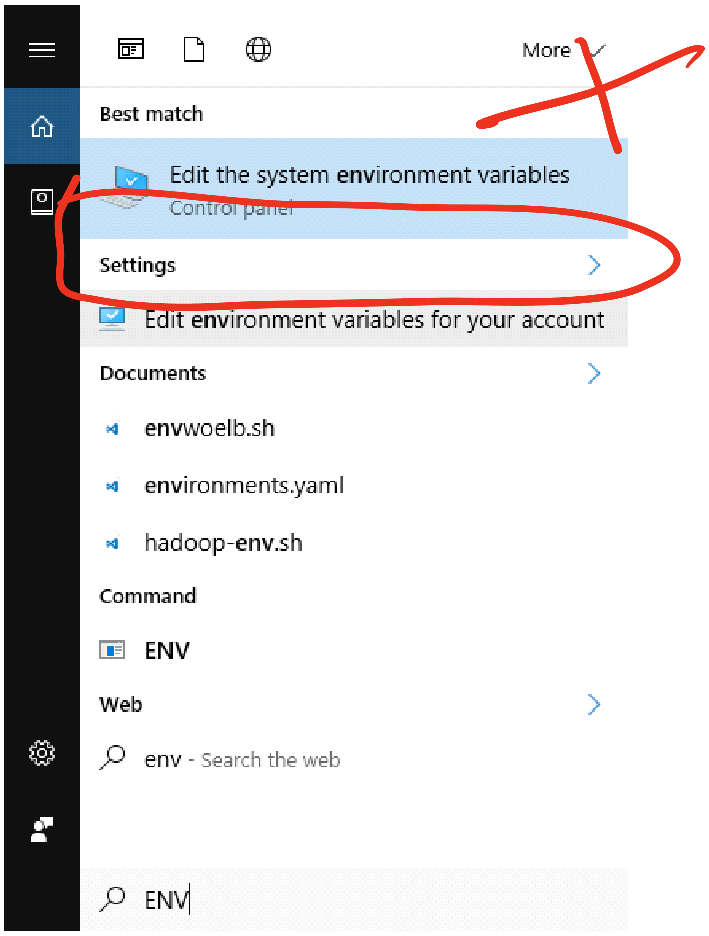
This command is issues after the vagrant up command and allows you to establish an SSH session directly into the vagrant box, with a pre-setup username and password, with NO ASK set in the sudoers file, making for seamless entry. You should never need to access and username or password in Vagrant as that defeats the purpose of Vagrant. But for completeness's sake it is vagrant:vagrant

NOTE - the `vagrant ssh` command works perfectly by default on all Linux and MacOS hosts. **NOT WINDOWS** by default. Why? Windows does not have a ssh client installed by default (they are working on a Win32 native port of SSH which is in beta, but it doesn't work well with Vagrant at the moment.) The solution in this case is not to use Putty, good thinking grasshopper, but brining in a 3rd party platform diminishes the power of Vagrant. Resist and I will explain what to do. Also MingGW64 is completely not the answer. You can use it for other things, but it is like dropping a nuclear weapon to light a match, too much complexity and overhead to add an entire sub-system. Simplicity is our goal.

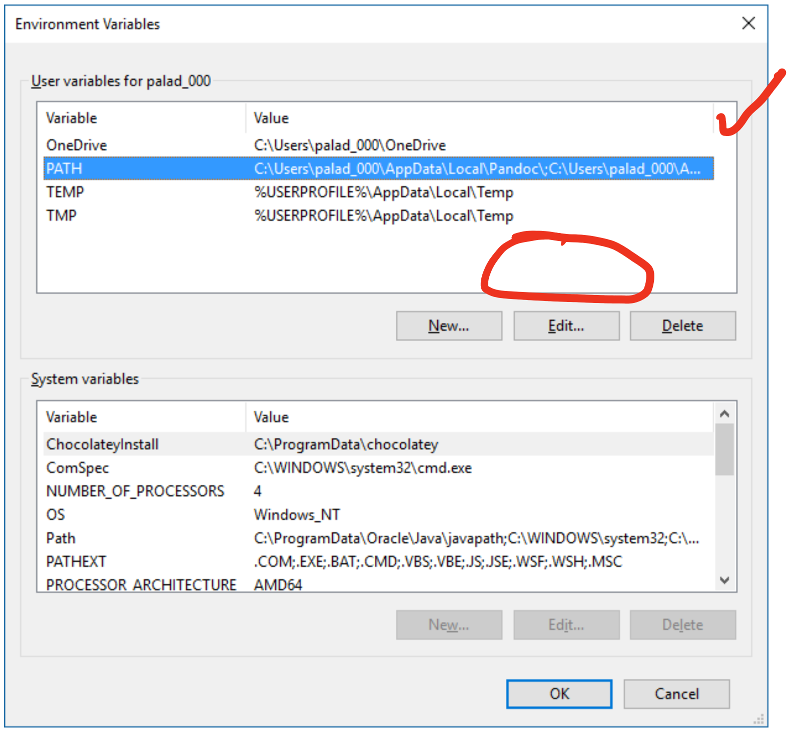
If you are using Vagrant then a good assumption is that you are also using Git for version control? Right? Included in the Git for Windows install package: https://git-scm.com/download/win there is a win32 port of SSH included in this package. Where is it? It is located (assuming a default install) at `C:\Program Files\Git\usr\bin` You will need to add this directory location to your SYSTEM PATH

WARNING - Be careful with modifying SYSETM PATH - you could destroy your computer. There is a safer way.

From the Windows Search Box (Cortana) type: **ENV** and you will get results similar to below: Select the option that mentions editing environment variables for **YOUR** account not **SYSTEM** account, there is no need to use the SYSTEM account in this case, once again overkill and could be dangerous.

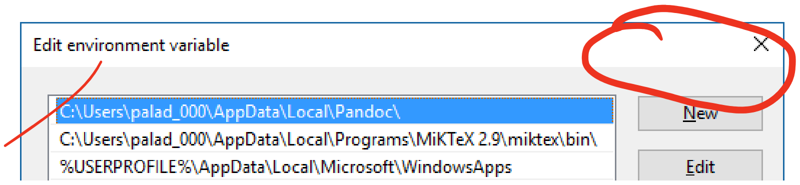


This screen will now appear

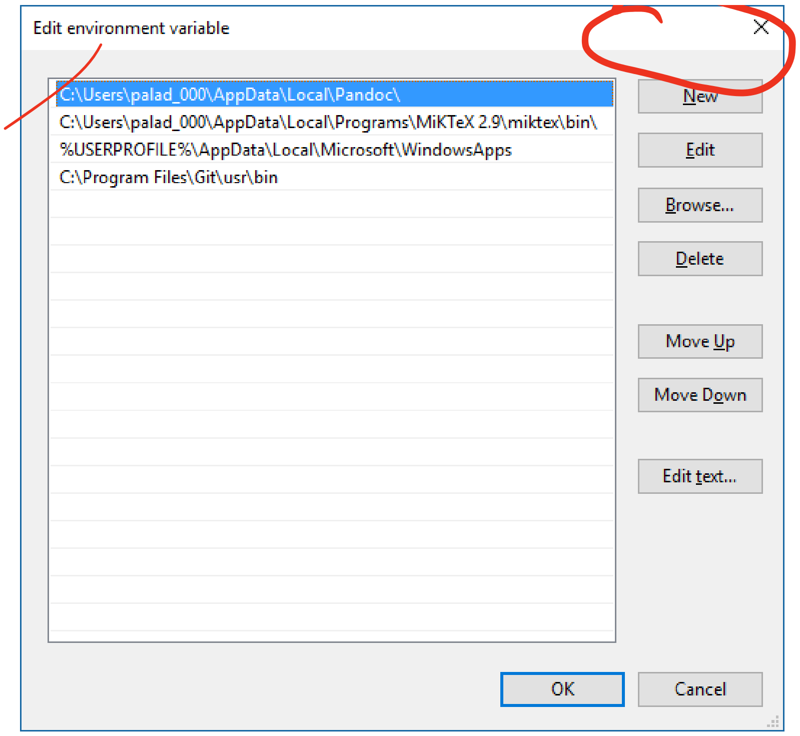


Variable	Value
OneDrive	C:\Users\palad_000\OneDrive
PATH	C:\Users\palad_000\AppData\Local\Pandoc\;C:\Users\palad_000\A...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\WINDOWS\system32\cmd.exe
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64



```
C:\Users\palad_000\AppData\Local\Pandoc\
C:\Users\palad_000\AppData\Local\Programs\MiKTeX 2.9\miktex\bin\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
```



Edit environment variable

```
C:\Users\palad_000\AppData\Local\Pandoc\
C:\Users\palad_000\AppData\Local\Programs\MiKTeX 2.9\miktex\bin\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Program Files\Git\usr\bin
```

Now vagrant ssh command will work natively on Windows.

Commands: **`vagrant plugin install vagrant-vbguest`**

If you are using Virtual Box and upon vagrant up command you receive an error about being unable to find extensions to mount a shared folder - you need to issue the command above to make it go away by properly installing the correct virtualbox additions (it is done automatically).

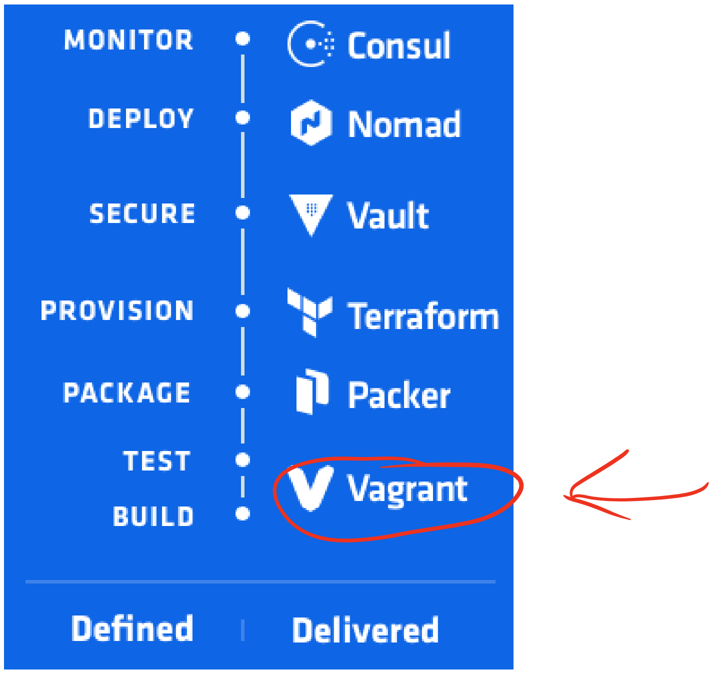
Packer.io Tutorial

Sunday, January 29, 2017 10:19 PM

What is Packer.io?

Packer is a tool for creating machine and container images for multiple platforms from a single source configuration.

From <https://www.packer.io/>



Packer is the companion tool to Vagrant for creating your own custom machine images and virtual machines.

Packer is an open source tool for creating identical machine images for multiple platforms from a single source configuration. Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms in parallel. Packer does not replace configuration management like Chef or Puppet. In fact, when building images, Packer is able to use tools like Chef or Puppet to install software onto the image.

A *machine image* is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines. Machine image formats change for each platform. Some examples include AMIs for EC2, VMDK/VMX files for VMware, OVF exports for VirtualBox, etc.

From <https://www.packer.io/intro/index.html>

Why use packer?

Pre-baked machine images have a lot of advantages, but most have been unable to benefit from them because images have been too tedious to create and manage. There were either no existing tools to automate the creation of machine images or they had too high of a learning curve. The result is that, prior to Packer, creating machine images threatened the agility of operations teams, and therefore aren't used, despite the massive benefits.

Packer changes all of this. Packer is easy to use and automates the creation of any type of machine image. It embraces modern configuration management by encouraging you to use a framework such as Chef or Puppet to install and configure the software within your Packer-made images.

In other words: Packer brings pre-baked images into the modern age, unlocking untapped potential and opening new opportunities.

Advantages of Using Packer

Super fast infrastructure deployment. Packer images allow you to launch completely provisioned and configured machines in seconds, rather than several minutes or hours. This benefits not only production, but development as well, since development virtual machines can also be launched in seconds, without waiting for a typically much longer provisioning time.

Multi-provider portability. Because Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox. Each environment is running an identical machine image, giving ultimate portability.

Improved stability. Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they'll be caught early, rather than several minutes after a machine is launched.

Greater testability. After a machine image is built, that machine image can be quickly launched and smoke tested to verify that things appear to be working. If they are, you can be confident that any other machines launched from that image will function properly.

Packer makes it extremely easy to take advantage of all these benefits.

What are you waiting for? Let's get started!

From <https://www.packer.io/intro/why.html>

Installation:

Packer is a self-contained standalone executable for multiple platforms -- you don't install it. https://www.packer.io/downloads.html

- MacOS
- Windows
- Linux (All kinds)
- OpenBSD
- FreeBSD

Windows

You can use package managers such as Chocolatey on Windows to download the packer executable and automatically add the location to the SYSTEM PATH. For Windows this is highly recommended (and basiclly the best and only way you should do this).

MacOS

Homebrew versions of packer are available but not always up to date. A few point versions behind should be ok.

Linux/*BSD

Do not use Linux default package managers - down load the executable and place it in a safe location. Then add that location to your SYSTEM PATH in your ~/.bashrc file via an export command.

Workflow & Explanation

Packer's goal is to take the manual parts of installing an operating system, all the times you have to hit the <enter> key, and simply automate this. One would think that this would be an easy operation and that it would have been a solved problem by the year 2014, but when you think about it, operating systems are not really engineered to be automatically installed. It defeats the entire PC (personal computer) experience ideal doesn't it?

Hashicorp essentially built a tool that captures each install step and places it into a JSON based template file. Combined with using Linux/BSD/Windows based "Windows Installation Answer Files, Debian/Ubuntu Presees, and RedHat/CentOS Kickstart Files" these concepts will be covered later in the tutorial.

Let us look at a JSON template file: Source can be retrieved from here:
https://github.com/illinoistech-itm/itmo453-553

```
{
    "builders": [{
        "name": "ubuntu-vanilla-14045-server",
        "type": "virtualbox-iso",
        "guest_os_type": "Ubuntu_64",
        "guest_additions_mode": "disable",
        "iso_url": "http://mirrors3.kernel.org/ubuntu-releases/14.04.5/ubuntu-14.04.5-
server-amd64.iso",
        "iso_checksum": "dde07d37647a1d2d9247e33f14e91acb10445a97578384896b4e1d985f754cc1",
        "iso_checksum_type": "sha256",
        "http_directory" : ".",
        "http_port_min" : 9001,
        "http_port_max" : 9001,
        "ssh_username": "vagrant",
        "ssh_password": "vagrant",
        "ssh_wait_timeout": "30m",
        "communicator": "ssh",
        "ssh_pty": "true",
        "shutdown_command": "echo 'vagrant' | sudo -S shutdown -P now",
        "vm_name": "ubuntu-vanilla-14045-server",
        "hard_drive_interface": "sata",
        "disk_size": 20000,
        "boot_wait": "5s",
        "boot_command" : [
            "<esc><esc><enter><wait>",
            "/install/vmlinuz noapic ",
            "preseed/url=http://{{ .HTTPIP }}:{{ .HTTPPort }}/preseed/preseed.cfg ",
            "debian-installer=en_US auto locale=en_US kbd-chooser/method=us ",
            "hostname=pleasechangeme ",
            "fb=false debconf/frontend=noninteractive ",
            "keyboard-configuration/modelcode=SKIP keyboard-configuration/layout=USA ",
            "keyboard-configuration/variant=USA console-setup/ask_detect=false ",
            "initrd=/install/initrd.gz -- <enter>"
        ],
  "vboxmanage": [
        [
          "modifyvm",
          "{{.Name}}",
          "--memory",
          "2048"
        ]

    ]
  }],
  "provisioners": [
  {
    "type": "shell",
  "execute_command" : "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
    "script": "../scripts/post_install_vagrant.sh"
  }
],
  "post-processors": [
 {
  "type": "vagrant",
 "keep_input_artifact": true,
 "output": "../build/{{.BuildName}}-{{.Provider}}-{{timestamp}}.box"
 }
]
}
```

Builders:
The majority of this information is taken from https://www.packer.io/docs/

Builders are the initial syntax needed to build for a single platform. This forms the bulk of the JSON Key-Value pairs you see above in the sample template I provided. The Builder of choice above is for VirtualBox initially, but if my target platform had been something else then I could have switched. Note the syntax will be different for each builder as some require things others do not (Amazon and Azure require account keys for instance).

The builders available are:

1. Amazon EC2 (AMI)
2. Azure
3. Cloudstack
4. Digital Ocean
5. Docker
6. Google Compute Engine
7. Hyper-V
8. OpenStack
9. Parallels
10. QEMU
11. Triton (Joyent/Samsung Public Cloud)
12. VirtualBox
13. Vmware

Provisioners:

Provisioner are tools that you can use to customize your machine image after the base install is finished. Though tempting to just use the Kickstart or Pressed files to do the custom install--this is not a good idea. You should leave the "answer files" as clean or basic as possible so that you may reuse them and do your customization here via a provisionser.

```
  "provisioners": [
  {
    "type": "shell",
    "execute_command" : "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
    "script": "../scripts/post_install_vagrant.sh"
  }
],
```

In the sample above I chose to implement an inline shell command, "execute command" and then via a shell script. Shell scripts are very easy to use and flexible. Provisioners can also be connected to use Provisioning 3rd party tools such as Puppet, Chef, Salt, Ansible, as well as PowerShell. These tools are called Orchestration tools and I would recommend checking them out if your interest or job lies in this domain.

Post-Processors:

Packer has the ability to build a virtual machine or OS Container once and export it to many different types of platforms in a single execution stretch. The initial artifact can be exported and converted across all of the formats listed below. Therein lies the power of Packer as you can deploy your production environment to any platform for any person: Dev, QA, Test, Ops, Sec, and so forth.

Once the Build step and Provision step are complete the last step (which is optional) is the Post-Proceesor step. This is where you can convert your base image you built into various other formats.

```
  "post-processors": [
 {
  "type": "vagrant",
 "keep_input_artifact": true,
 "output": "../build/{{.BuildName}}-{{.Provider}}-{{timestamp}}.box"
 }
]
```

Here you see I am converting the VirtualBox *.ovf file into a Vagrant Box file *.box. If you leave off the keep_input_artifact option, the initial artifact will be deleted and only the post-processor result will remain.

Types of Post-Processing include:
1. Amazon AMI
2. Atlas (Hashicorp artifact enterprise deployment tool)
3. Compress (simply a compressed archive)
4. Docker
5. Google Compute Engine
6. Vagrant and Vagrant Cloud (cloud based storage and management of Vagrant Boxes)
7. vShpere

You can use multiple post-processors if desired.

Tutorial

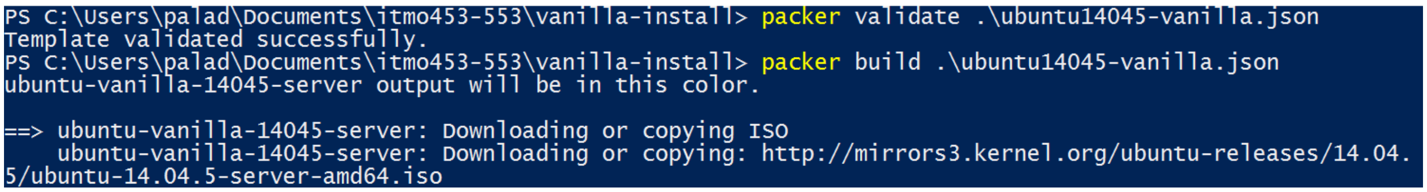
There are two key commands to use in Packer (assuming you have added the packer executable to your system path) First type: `packer -v` and see if you get version information output.

Command: `packer validate`

This command will check the syntax of your *.json packer template for syntax errors or missing brackets. It will not check logic but just syntax. Good idea to run it to make sure everything is in order.

Command: `packer build`

This command will be what is used to execute and run the packer *.json template.



```
PS C:\Users\palad\Documents\itmo453-553\vanilla-install> packer validate .\ubuntu14045-vanilla.json
Template validated successfully.
PS C:\Users\palad\Documents\itmo453-553\vanilla-install> packer build .\ubuntu14045-vanilla.json
ubuntu-vanilla-14045-server output will be in this color.

==> ubuntu-vanilla-14045-server: Downloading or copying ISO
    ubuntu-vanilla-14045-server: Downloading or copying: http://mirrors3.kernel.org/ubuntu-releases/14.04.
5/ubuntu-14.04.5-server-amd64.iso
```