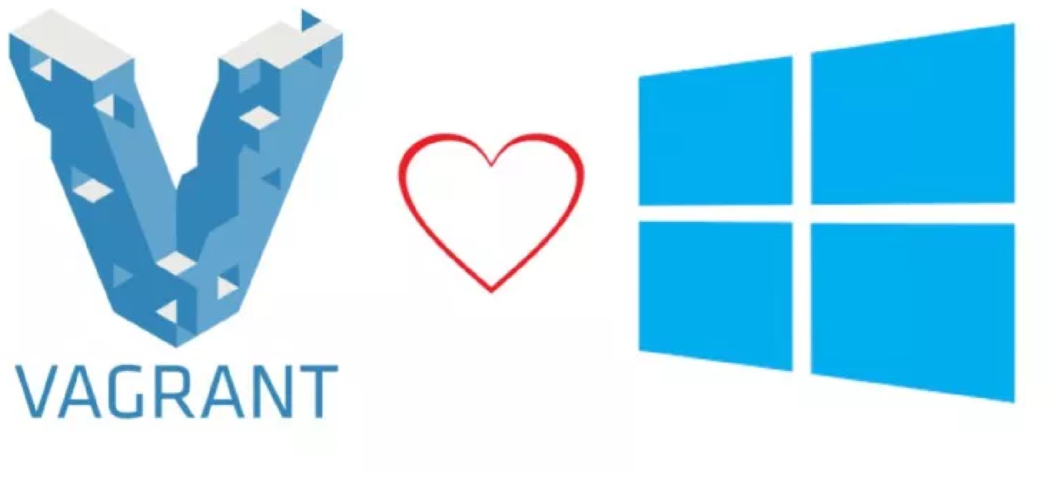
Dots and Brackets: Code Blog

Blog about DevOps, distributed applications and microservices

Using Vagrant for Windows VMs provisioning



Using Vagrant for creating Consul cluster on Linux probably was fun. But what about Windows hosts? Believe it or not, but more than half of developers are actually using Windows, so for most of the folks seeing how Vagrant creates Linux VMs is pretty useless.

However, you can create and provision Windows VMs with Vagrant with little to no problem. In fact, Windows support has been around for years. But there're some things to keep in mind though.

Things to keep in mind

Licensing

Because it's not particularly free OS, one cannot just pick random Windows box and put it into production. You can find something in evaluation mode to experiment on, or create your own box, but at some point you'll need to take care of Windows activation.

Remoting into VM

Connecting to Windows VM via `vagrant ssh` is doable, but not without SSH server installed onto guest OS. `vagrant powershell` on the other hand is more Windows friendly mean of remoting to guest OS. But even though PowerShell works on Linux now, the piece responsible for remote connections is not quite there yet, so unless you're using Windows host OS, this command is not for you. Surprisingly, `vagrant rdp` – remote desktop protocol client – is the only more or less cross platform (Linux, Mac) way of getting into Windows guest OS.

Performance

Unless Windows box creator indicated that VM should be created with doubled size of CPU cores and memory, you'll have to do that by yourself.

Provisioning

Provisioning should be done in PowerShell, hurrah! Don't get me wrong, I used to hate PowerShell, but spending so much time with it gave me some sort of Stockholm syndrome, so now I think it's quite OK. Another thing is that Vagrant communicates to Windows guest OS via WinRM, so again, unless box creator opened WinRM ports and configured Vagrant communicator to use them (`config.vm.communicator = "winrm"`) you'll need to do it by yourself in Vagrantfile.

Plan for today

We still have a cluster of three Linux hosts with Consul agents on them. What about adding the fourth one on Windows? That should be easy.

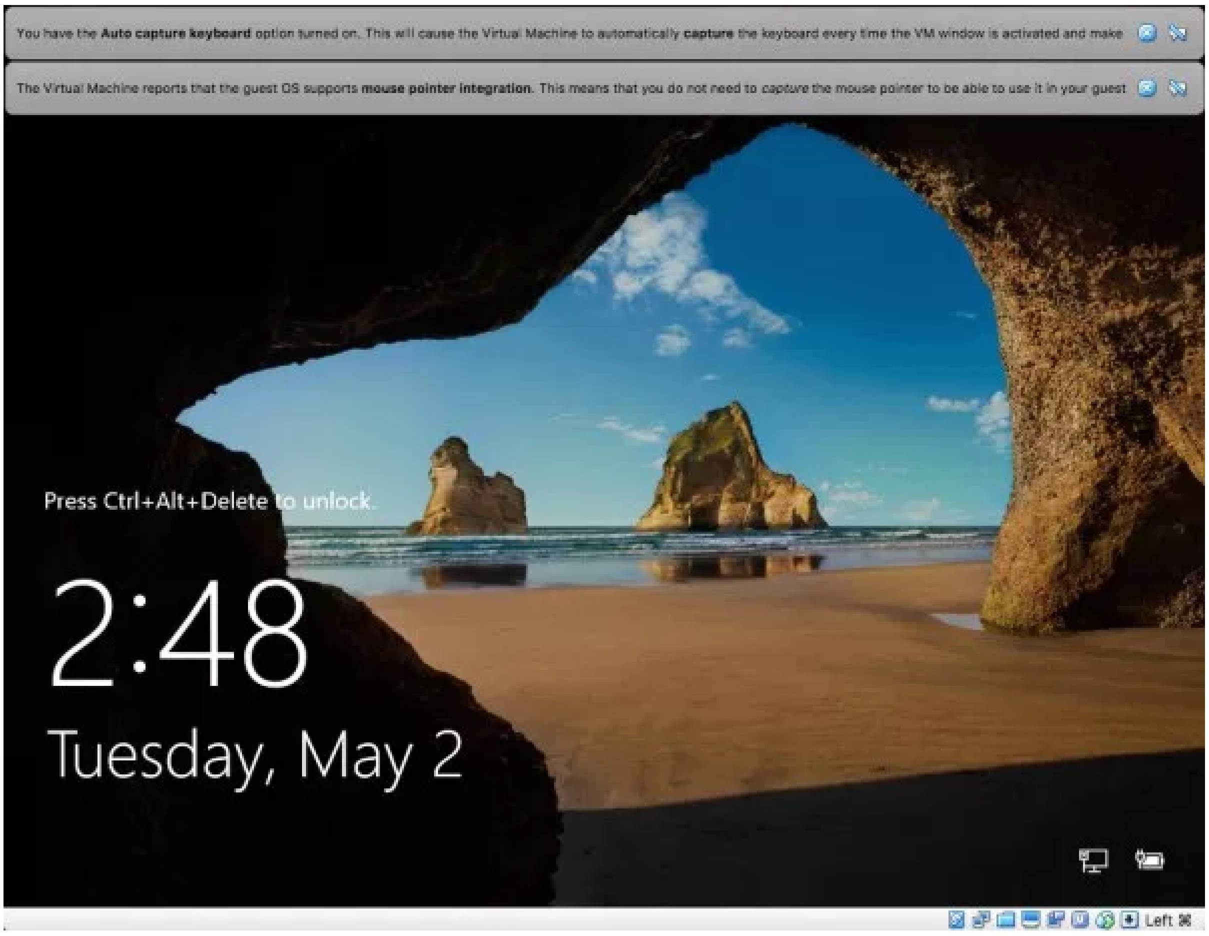
Step 0. Creating blank Windows VM

I found mwrock/Windows2016 evaluation box to be reasonably well maintained, so I'll use it for today. We can start with configuring a standalone Consul Windows VM and when it's done we can merge it with the rest of the cluster.

As with Linux VMs, `vagrant init mwrock/windows2016 --minimal` will produce pretty standard Vagrantfile:

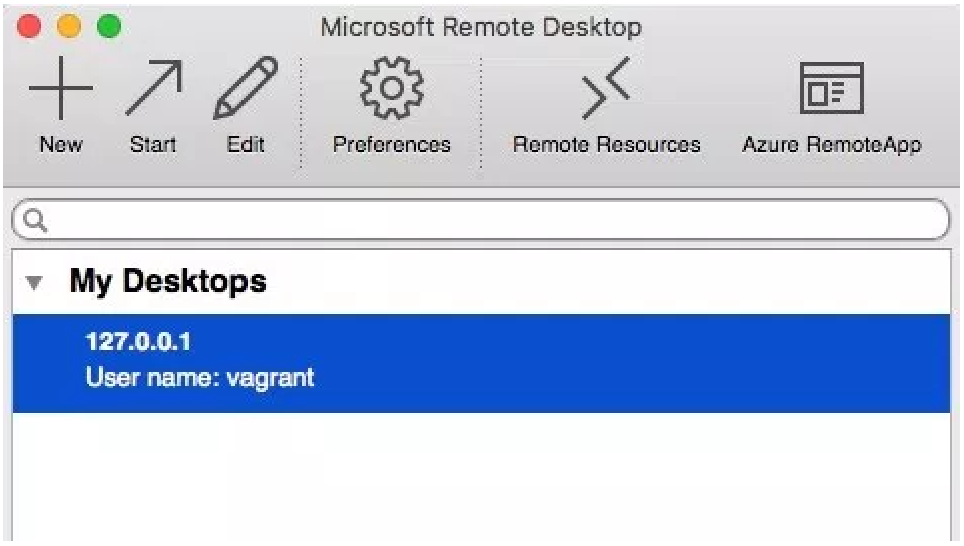
```ruby
Vagrant.configure("2") do |config|
  config.vm.box = "mwrock/windows2016"
end
```

`vagrant up` on the other hand will **fetch 5** holy **gigs** of Windows box and after a few minutes of meditation on Microsoft masterpiece will end up with running Windows Server 2016 instance:


You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard every time the VM window is activated and make

The Virtual Machine reports that the guest OS supports **mouse pointer integration**. This means that you do not need to capture the mouse pointer to be able to use it in your guest

You can login into the VM using "vagrant" as user name and "vagrant" as a password. `vagrant rdp` command will also do, but depending on RDP gods' mood you might need to do an extra click.



127.0.0.1
User name: vagrant

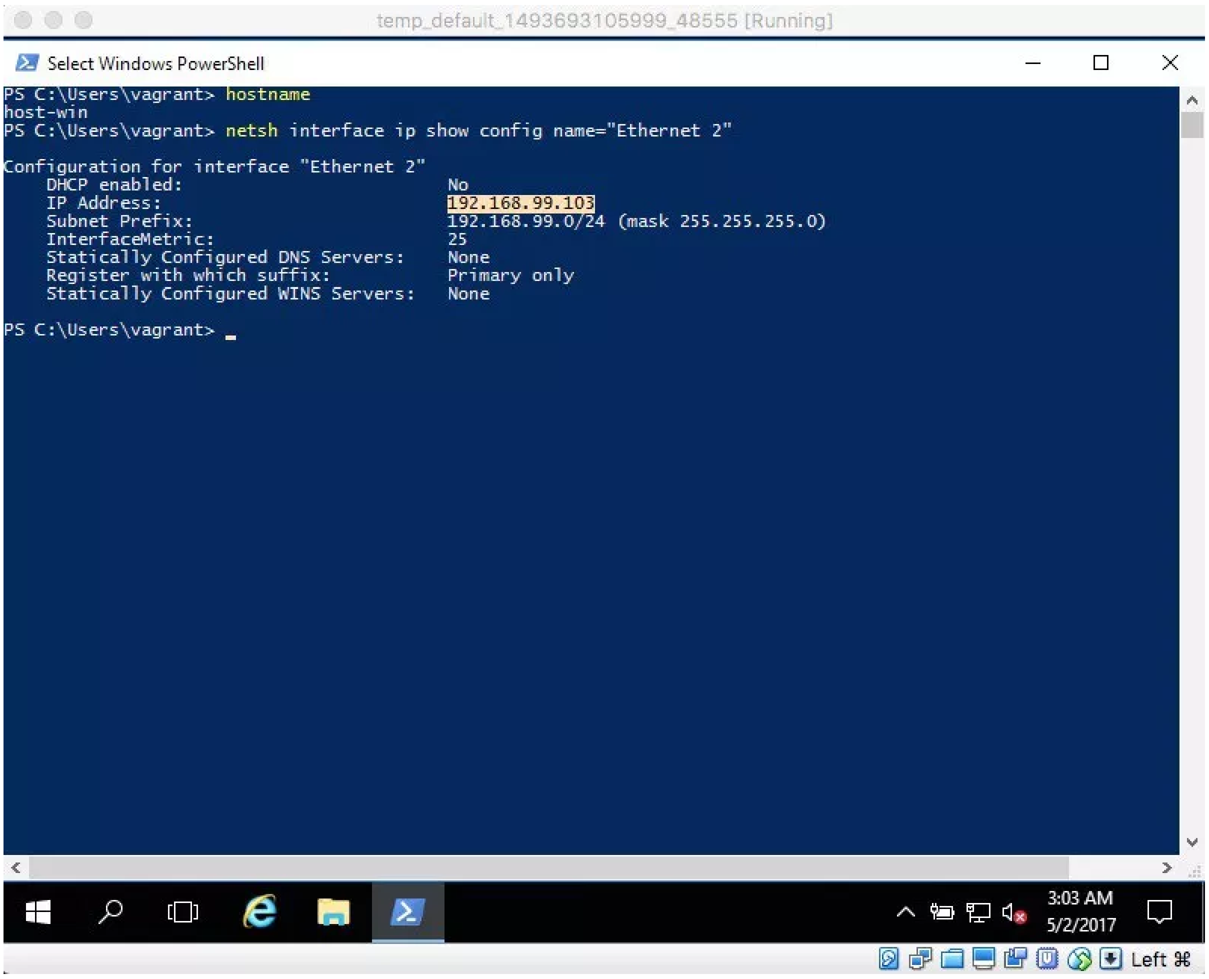
'vagrant rdp' command result on Mac

Step 1. Setting up a host name and static IP address

These steps are absolutely identical to ones we did for Linux hosts last time. That's the beauty of Vagrant. It's just a few more lines in Vagrantfile:

```
Vagrant.configure("2") do |config|
  config.vm.box = "mwrock/windows2016"
  config.vm.hostname = "host-win"
  winClientIP = "192.168.99.103"
  config.vm.network "private_network", ip: winClientIP
end
```

Execute `vagrant reload` to reconfigure the VM and then we can go to the VM itself to confirm that Vagrantfile changes indeed worked:


temp_default_1493693105999_48555 [Running]
Select Windows PowerShell
3:03 AM
5/2/2017
Left ⌘

```
PS C:\Users\vagrant> hostname
host-win
PS C:\Users\vagrant> netsh interface ip show config name="Ethernet 2"

Configuration for interface "Ethernet 2"
    DHCP enabled:                         No
    IP Address:                           192.168.99.103
    Subnet Prefix:                        192.168.99.0/24 (mask 255.255.255.0)
    InterfaceMetric:                      25
    Statically Configured DNS Servers:    None
    Register with which suffix:           Primary only
    Statically Configured WINS Servers:   None

PS C:\Users\vagrant> _
```

Step 2. Download and “install” Consul

Declaring Windows provisioning steps in Vagrant is also almost identical to declaring them for Linux hosts. The only difference is that instead of `.sh` files we have to provide `.ps1` for PowerShell:

```
#...
config.vm.provision "shell", path: "provision.ps1"
#...
```

Provisioning Consul agent in PowerShell syntactically does look different from `provision.sh` we created the last time, but it has the same logic: download and unzip. I added a few extra steps like checking if destination directory exists, removing source archive after unzipping, creating directory for Consul data (we’ll need that later) and printing out Consul version to make sure it’s working. But only three lines are really, truly necessary:

```
$consulDir = "C:\Consul"
$consulExePath = "$consulDir\consul.exe"
$consulZipPath = "c:\tmp\consul.zip"
$version = "0.8.1"

```

```
if (-not (Test-Path $consulDir))
{
    # Use newest TLS1.2 protocol version for HTTPS connections
    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

    # Download consul
    Invoke-WebRequest -OutFile $consulZipPath -Uri https://releases.hashicorp.com/cc

    # Unzip
    Expand-Archive -Path $consulZipPath -DestinationPath $consulDir

    # Cleanup
    Remove-Item $consulZipPath

    # Create Consul data dir
    New-Item -Type Directory -Path "$consulDir\data"

    # Print Consul version number to confirm it was downloaded correctly
    & $consulExePath version
}
else
{
    Write-Host "Consul is already installed"
}
```

I also had to explicitly tell what TLS protocol to use for HTTPS requests, as HashiCorp's web site (as almost everybody else's) uses TLS 1.2 for encryption and that's not something spoken by PowerShell's `Invoke-WebRequest` by default. What a surprise.

But let's apply the provisioning file and see if it works:

```
$ vagrant provision
# ==> default: Running provisioner: shell...
# ...
# ==> default: Consul v0.8.1
# ...
```

Yup, it's all good.

Step 3. Configure and start Consul agent

With little adaptation agent's config JSON from last article can be used for Windows agent as well. It's mostly about the slashes:

```
winClientIP = "192.168.99.103"
serverIP = "192.168.99.100" # Consul server IP from last article

clientInit = %(
  {
    "advertise_addr": "#{winClientIP}",
    "retry_join": ["#{serverIp}"],
    "data_dir": "C:\\\\Consul\\\\data"
  }
)
```

```

config.vm.provision "shell", inline: "Set-Content -Value '#{clientInit}' -Path C:\\C
```

And finally, starting the executable:

```
#...
config.vm.provision "shell", inline: "C:\\Consul\\consul.exe agent -config-dir=C:\\Co
#...
```

The line above will start Consul agent as a foreground process, so provisioning step will never end. That's OK for now, but we could've easily fixed that with something like PowerShell's `Start-Process` cmdlet:

```
#...
config.vm.provision "shell", inline: "Start-Process consul.exe -WorkingDirectory C:\\
#...
```

Now, `vagrant provision` one more time and enjoy running Consul agent that can't find the rest of the cluster:

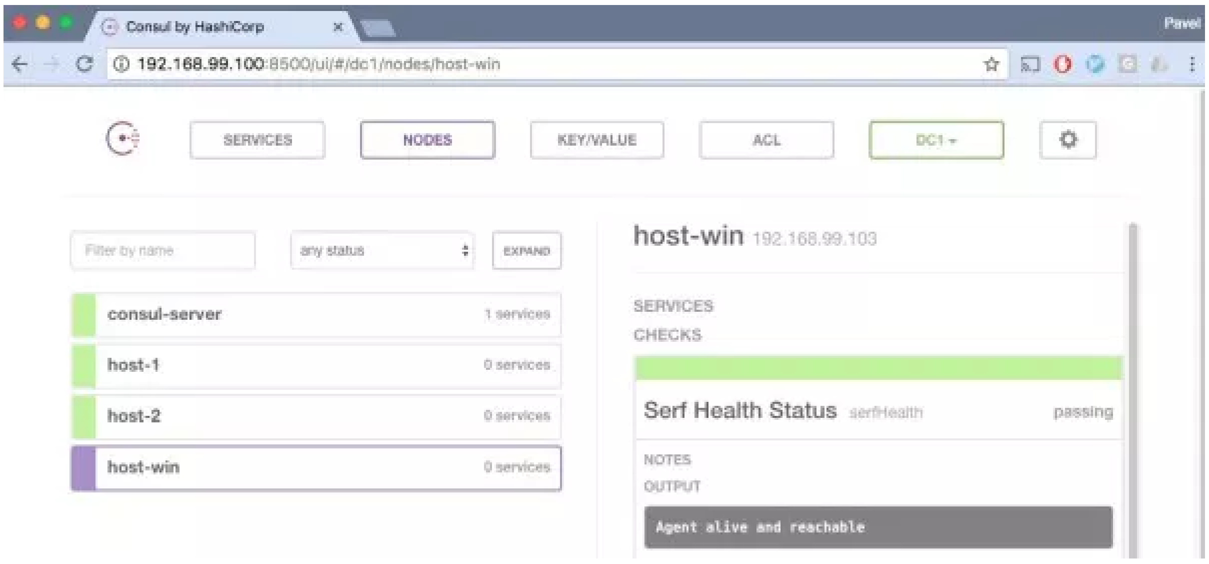
```
$ vagrant provision
#...
# ==> default: ==> Consul agent running!
#...
# ==> default:     2017/05/02 03:46:05 [WARN] manager: No servers available
```

We can fix that by starting Linux cluster we created before, or by implementing step 4.

Step 4. Optional. Merge Vagrantfiles

I already merged today's Vagrantfile for Windows host with one created the last time. The result is on github, "winhost" branch. There were nothing difficult or interesting enough to put it here, but resulting Vagrantfile can do some magical stuff. It can create four virtual machines with different operating systems and configure them to behave as one Consul cluster. Imagine how long would it take to do that manually. Especially the Windows part.

That's the beauty of automation: `vagrant up`, wait for few minutes and boom:



```
Agent alive and reachable
```

Conclusion

Automating Windows VMs is not a myth. It’s something very doable and surprisingly easy to follow. Yes, there’re some problems with licensing and high resource constraints, but if you’re configuring VMs in a cloud, it already provides images with license taken care of, so configuring those hosts with Vagrant is no way different from configuring ones on Linux. There’s just no reason to do that manually anymore.

Share this:

pav / May 3, 2017 / Development / consul, quick guide, Vagrant, windows server

Dots and Brackets: Code Blog / Proudly powered by WordPress