**Environment/Setup**

OS - Windows 10

**Env Variables -**

C:\Program Files\Git\usr\bin

C:\Program Files\Oracle\VirtualBox

C:\HashiCorp\Vagrant\bin

The following cost a few hours, and some hair.

**For Windows user’s : Enable hardware virtualization on your host, it's one of the few listed options in the BIOS.**

**Turn off Hyper V, this messes with virtualbox and VMware tools.**

Step 1

Install Virtualbox - Version - VirtualBox 5.1.12 platform packages.

URL : https://www.virtualbox.org/wiki/Downloads

Step 2

Install Vagrant - Version - 1.9.1

URL : https://www.vagrantup.com/downloads.html

Step 3

We want ssh, without messing around with OpenSSH or anything with big words like marmalade.

Install Git for windows - Version - 2.11.0

URL : https://git-scm.com/download/win

Step 4

Run vagrant box add - with the following options - puppetlabs/ubuntu-14.04-64-nocm

https://vagrantcloud.com/puppetlabs/boxes/ubuntu-14.04-64-nocm

Step 5

Run vagrant up

Spec comments.   
  
The solution should contain a Vagrantfile, all associated configuration management files, and a README that lists both the steps we should run to duplicate your solution and any assumed software version(s).

<https://github.com/BenCoughlan15/automationlogic>

1. The solution should only require vagrant installed on the host machine with all additional software installed on the virtual machines.

I’m using git to pull down the repo, and using the ssh binary in there, so I don’t need to mess with the fresh hell of configuring ssh on Windows.

I’ve also installed Virtualbox, and Vagrant. (see above).

1. · Create a Vagrantfile that creates a single machine using this box:[https://vagrantcloud.com/puppetlabs/boxes/ubuntu-14.04-64-nocm](https://vagrantcloud.com/puppetlabs/boxes/ubuntu-14.04-64-nocm" \t "_blank)  
   and installs the latest released version of your chosen configuration management tool.

see above.

1. Install the nginx webserver via configuration management.

Ansible installs this, as part of its playbook.

1. · Run a simple test using Vagrant's shell provisioner to ensure that nginx is listening on port 80

Emphasis on simple…. The last command of the Vagrantfile should show you the network settings.

This is pretty nasty, I’d have Nagios do this in future.



1. · Again, using configuration management, update contents of /etc/sudoers file so that Vagrant user can sudo without a password and that anyone in the admin group can sudo with a password.

Provision.sh runs from vagrant to do this substitution, it remains idemponent by checking for fields before executing the command, in hindsight I’d bomb much of it into ansible.

To test this, run

vagrant ssh -c "sudo grep 'admin\|vagrant' /etc/sudoers" loadbalancer

1. · Make the solution idempotent so that re-running the provisioning step will not restart nginx unless changes have been made

See above : ansible also does this by default.

1. · Create a simple "Hello World" web application in your favourite language of choice.

Go to 10.0.15.11 for the site.

Each application host is also available via their respective ip’s for your convenience.

Not overly chuffed with this, I had grand plans to have it connect to a web server populate a database full of 1990’s spinning gif’s, but I’m out of time, so..

 

1. · Ensure your web application is available via the nginx instance.

As I’m using haproxy to balance these, I may have made a mistake here, you can certainly access these application hosts independently, but the loadbalancer uses haproxy to manage the nginx services, re-reading this, I feel like an idiot, so I’m going to be a coward and blame the oversight on time constraints.

1. Extend the Vagrantfile to deploy this webapp to two additional vagrant machines and then configure the nginx to load balance between them.

See Vagrantfile

1. · Test (in an automated fashion) that both app servers are working, and that the nginx is serving the content correctly.

Vagrant does this on provision with connectiontest.sh.

Again, I’d have configured Nagios..

1. I’m interested in your working as much as your answers, so where you make a decision to go one way rather than another, please explain your thinking.  
   Your solution will be assessed on code quality and comprehensibility, not just on technical correctness.

## Things I’d change :

* I’d have given myself more time, I’ve had a few major releases in work this week so have been working around the clock, and haven’t had a lot of time to do this exercise.
* I’d have made the necessary connection settings to allow the application servers to connect to the DB on the loadbalancer.
* I’d pull and display data from the DB, and interact with a webserver, probably using something lightweight like json.
* I’d have done all of this from a linux machine, rather than this stock windows laptop, although I have to say, Window’s have vastly improved their Command Prompt.
* Changed the DNS settings to give it a proper hostname.
* I’d have used salt/chef/puppet or any other configuration tool out there. However, there are tonnes of examples of this exercise across the web that do this task with those tools. I wanted to learn from this experience, rather than to replicate somebody else’s work, so I have gone with Ansible.

Optional Extra credit:  
· Have the webapp be dynamic - e.g. perform a db query for inclusion in the response (such as picking a random quote from a database) or calling an API of your choice(e.g. weather).  
=> Any additional resources (e.g. a shared db server) should be set up by the Vagrant file   
· Include a section for possible improvements and compromises made during the development of your solution