**Demonstration of Vagrant and Ansible-Built HA Web Servers with Proxy**

An example of an automated two web-server deployment with a single load-balancing proxy server presenting web content.

W.T.Hart – 01/12/19

Table of Contents

[Overview: 3](#_Toc26300657)

[Technical Detail: 3](#_Toc26300658)

[Prerequisites (before you begin): 4](#_Toc26300659)

[Installation and Running: 4](#_Toc26300660)

[Deployment Tests: 5](#_Toc26300661)

[Further Technical Details: 5](#_Toc26300662)

[Folder Structure: 7](#_Toc26300663)

[Additional Application(s): 8](#_Toc26300664)

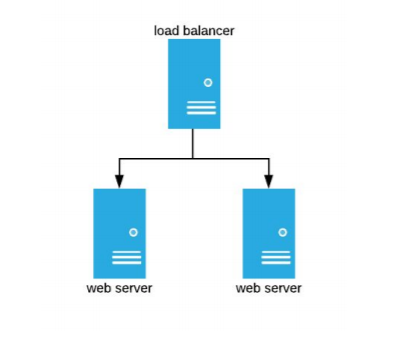
[Things To Do: 8](#_Toc26300665)

[Time Taken to Complete Project: 8](#_Toc26300666)

# Overview:

This repository contains all the required files required to automatically build a three-server HA cluster of Nginx, with a single load-balancer server running Nginx forwarding web requests to two backend Nginx web servers. Each web server is PHP-enabled, with the default web page presented as a PHP application. Additionally, a simple weather application – again configured with PHP – presents the present weather conditions in London.

Each web server is configured identically, with Ansible configuring all servers and Vagrant handling the initial host deployment. The front-end proxy is set to use both backend servers in round-robin fashion, ensuring that if one web server suffers a failure, the second will continue to present web pages.



# Technical Detail:

The repository is made up of the following files:

* Vagrantfile (build definition)
* Ansible inventory file (contains target hosts for Ansible)
* Ansible playbooks - one for the proxy and one for any number of backend web servers (default: 2). These are in the ‘provision’ folder and are ‘playbook-proxy.yml’ and ‘playbook-webserver.yml’ respectively.
* Playbook roles subfolders (located under provision/roles). See the section ‘Folder Structure’ for detailed information on the repository folder/file layout.
* The playbooks reference ‘Ansible roles’ (webserver, proxy and common), which correspond to the specific configuration options for webservers, the load-balancer (proxy) and options common to both proxy and web servers.

The Vagrantfile defines the build instructions for the base virtual machines, an ‘Ubuntu/bionic64’ base image. Ansible deploys each machine's configuration individually by loading Ansible locally on each box and running the corresponding playbook for each machine type, creating a ‘/vagranttest’ root folder on each host with a copy of the entire build folder. The results of the default configuration are:

|  |  |  |  |
| --- | --- | --- | --- |
| Server Name | Server Type | IP (Network) address | Application |
| Proxy | Load Balancer | 10.0.0.10 | Nginx (proxy) |
| Host1 | Web Server 1 | 10.0.0.11 | Nginx (web) |
| Host2 | Web Server 2 | 10.0.0.12 | Nginx (web) |

# Prerequisites (before you begin):

Before downloading this repository, ensure your machine has the following installed:

* Oracle Virtualbox - <https://www.virtualbox.org/wiki/Downloads>
* Vagrant - <https://www.vagrantup.com/downloads.html>

Follow the installation instructions on each of the links above for your particular environment (Windows, Mac or Linux). **Make sure you can run both applications in isolation before downloading the repository with the deployment files**. Run Virtualbox to ensure the installation has completed successfully, and open a terminal on your computer (CMD shell in Windows, bash shell in Linux and terminal on a Mac) and type ‘vagrant’ to ensure Vagrant is installed.

To ensure reasonable performance, please ensure your machine is sufficiently powerful and has enough memory and disk capacity to run three virtual servers.

In the case of Windows 10, the following must be requirements must be met:

* Hyper-V must be either uninstalled or disabled. *Note that this deployment will not run on Hyper-V!*
* If you have installed Docker for Windows, this must also be disabled/uninstalled.

# Installation and Running:

Once the prerequisites for your particular machine have been met, to install this example, clone this repository to a folder on your drive:

git clone https://github.com/billhartcivica/vagranttest.git

Alternatively copy the archived copy of the same folder (vagranttest.zip) to your computer and unzip it to a location on your computer:

unzip vagranttest.zip

Navigate to the newly downloaded/unzipped folder:

cd vagranttest

Assuming you have met the technical requirements (installed Vagrant and Oracle Virtualbox), all that is needed is to run:

vagrant up

After a period of about 5 to 6 minutes, depending on the number of backend web servers you’ve chosen to deploy, the server cluster should be up and running. Open your browser on your computer and enter 'http://10.0.0.10' in the address bar and press <enter>. You should see the following:

A screenshot of a cell phone

Description automatically generated

# Deployment Tests:

This deployment has been tested on both Linux and Windows 10 hosts.

Linux

The Linux installation was very simple:

* Installed Virtualbox
* Installed Vagrant

Windows

The Windows deployment was somewhat more involved, depending on your setup. To run the deployment on Windows 10, I found the following prerequisites **must** be met:

* Installed Oracle Virtualbox, which must be the **only** hypervisor running on the system.
* Installed Vagrant.
* Disabled/removed Hyper-V from the computer.
* Disabled/removed Docker (if installed) from the computer.

Macintosh

Unfortunately, I was unable to source a Macintosh host to test this on an OSX environment. However, it is entirely likely that as long as the prerequiresites above are met, this deployment should work correctly on a Mac. The process should be as with other hosts:

* Install Oracle Virtualbox.
* Install Vagrant.

# Further Technical Details:

The process of creating the cluster is as follows:

1. Vagrant initiates the build, using the directives found in the Vagrantfile in the root of this repository.
2. The Vagrantfile loops through a routine to set up the two backend web servers - host1 and host2. This is set by the variable SERVER\_COUNT (default: 2) in the Vagrantfile. This can be amended to create further backend web servers, limited only by the resources available on your own computer. *NOTE: Amending the SERVER\_COUNT value will automatically add the extra IP addresses for the backend web servers to the front-end proxy configuration.*
3. Each iteration of the webserver provisioning installs Ansible on the host and calls the same Ansible tasks in the corresponding role, defined in the provision/roles/webserver/tasks/main.yml file, including amending the default configuration, installing PHP and copying the index.php file to the /usr/share/nginx/html folder.
4. Additionally, the webserver provision process also calls the tasks in the provision/roles/common/tasks/main.yml file. These tasks are common to both proxy and web server, involving updating the apt cache, installing Nginx, amending the /etc/sudoers file and starting the Nginx daemon.
5. After each host is installed, local tests are run to check for the presence of the nginx service, the default listening port (80) and whether the expected 'Hello World' page is displayed. As each server is running Nginx and follows the same set of general rules, the tests defined are common to all. Because of this, the tests are defined in the provision/roles/common/tests/test.yml file.
6. Next, the web proxy, is created and Ansible installed locally on that host as before, calling on the provision/roles/common/tasks/main.yml file to run the common tasks for both proxy and web server.
7. Next the specific configuration for the proxy/load-balancer is implemented. This calls on the 'role' for the proxy held in the provision/roles/proxy folder which contains the tasks, the tests and the configuration files required to be copied to the host. This includes the dynamically created load-balancer.conf file, denoting the backend web servers the proxy must load-balance between.
8. The role's tasks (located in provison/roles/proxy/tasks/main.yml) define what changes are made to the host to configure it as the web proxy for the other two servers.
9. Next, the in-line tests for the proxy are run, as with the web servers checking for the presence of the Nginx service, the presence of port 80/TCP and checking that the default web page loads from the backend servers through the proxy correctly.

# Folder Structure:

├── hello.png # Image file for README.md

├── inventory # Ansible inventory of target hosts

├── provision # Top-level Ansible provision folder

│   ├── playbook-webserver.yml # Ansible playbook for backend web hosts

│   ├── playbook.yml # Ansible playbook for frontend proxy

│   └── roles # Ansible roles

│   ├── common # Common role folder (common to proxy and web)

│   │   ├── defaults # Default settings for provisioning

│   │   │   └── main.yml # Config file for default settings

│   │   ├── files # Files to be uploaded to target (empty)

│   │   ├── handlers

│   │   │   └── main.yml

│   │   ├── meta

│   │   │   └── main.yml

│   │   ├── README.md # README file for the common role

│   │   ├── tasks # Tasks common to both web and proxy

│   │   │   └── main.yml # Tasks file for common tasks

│   │   ├── templates

│   │   ├── tests # Post-installation tests folder

│   │   │   ├── inventory # Inventory file for tests

│   │   │   └── test.yml # Tests defined for all hosts

│   │   └── vars

│   │   └── main.yml

│   ├── proxy # Proxy role folder

│   │   ├── defaults # Default settings for provisioning

│   │   │   └── main.yml # Config file for default settings

│   │   ├── files # Files to be uploaded to target

│   │   │   └── load-balancer.conf # Load-balancer config for proxy (dynamically created)

│   │   ├── handlers

│   │   │   └── main.yml

│   │   ├── meta

│   │   │   └── main.yml

│   │   ├── README.md # README file for proxy role

│   │   ├── tasks # Tasks folder for proxy

│   │   │   └── main.yml # Tasks file for proxy provisioning

│   │   ├── templates

│   │   ├── tests

│   │   │   ├── inventory

│   │   │   └── test.yml

│   │   └── vars

│   │   └── main.yml

│   └── webserver

│   ├── defaults # Default settings for provisioning web server

│   │   └── main.yml # Config file for default settings

│   ├── files # Files to be copied to the target host(s)

│   │   ├── default # Default Nginx config file

│   │   ├── index.html # Default web page

│   │   └── index.php # Default web page for PHP handling (default)

│   ├── handlers

│   │   └── main.yml

│   ├── meta

│   │   └── main.yml

│   ├── README.md # README file for web server role

│   ├── tasks # Tasks folder for web servers

│   │   └── main.yml # Tasks file for web servers

│   ├── templates

│   ├── tests

│   │   ├── inventory

│   │   └── test.yml

│   └── vars

│   └── main.yml

├── README.md # README file for the repository

└── Vagrantfile # Config file for Vagrant deployment

# Additional Application(s):

As part of the exercise, I have included a very simple weather app, written in PHP and deployed on both web servers. This can be accessed by entering the following in your browser navigation bar:

```

http://10.0.0.10/weather.php

```

This displays some current weather conditions in the London area.

# Things To Do:

There may be more efficient, elegant ways to construct this deployment. Given the time I had to complete the task, this was done in the most expedient and elegant way I saw fit.

Further improvements might be to deploy Nginx within a Docker container, running Docker on each web host and on the proxy, configuring each using mapped configs to their respective internal configuration folders.

Taking containerisation further, another consideration might be to run the group as a Kubernetes cluster, with proxy and web servers presented as orchestrated containers. This would allow for automatic scaling of both backend web servers as well as the proxy, allowing the system to react more dynamically to web requests.

# Time Taken to Complete Project:

Overall, the total time taken to complete the configuration and documentation process, including testing, was roughly 9 and a half hours.