Chapter 2 - Local Infrastructure Development: Ansible and Vagrant

Prototyping and testing with local virtual machines

Ansible works well with any server to which you can connect—remote *or* local. For speedier testing and development of Ansible playbooks, and for testing in general, it's a very good idea to work locally. Local development and testing of infrastructure is both safer and faster than doing it on remote/live machines—especially in production environments!

In the past decade, test-driven development (TDD), in one form or another, has become the norm for much of the software industry. Infrastructure development hasn't been as organized until recently, and best practices dictate that infrastructure (which is becoming more and more important to the software that runs on it) should be thoroughly tested as well.

Changes to software are tested either manually or in some automated fashion; there are now systems that integrate both with Ansible and with other deployment and configuration management tools, to allow some amount of infrastructure testing as well. Even if it's just testing a configuration change locally before applying it to production, that approach is a thousand times better than what, in the software development world, would be called 'cowboy coding'—working directly in a production environment, not documenting or encapsulating changes in code, and not having a way to roll back to a previous version.

The past decade has seen the growth of many virtualization tools that allow for flexible and very powerful infrastructure emulation, all from your local workstation!

It's empowering to be able to play around with a config file, or to tweak the order of a server update to perfection, over and over again, with no fear of breaking an important server. If you use a local virtual machine, there's no downtime for a server rebuild; just re-run the provisioning on a new VM, and you're back up and running in minutes—with no one the wiser.

Vagrant[33], a server provisioning tool, and VirtualBox[34], a local virtualization environment, make a potent combination for testing infrastructure and individual server configurations locally. Both applications are free and open source, and work well on Mac, Linux, or Windows hosts.

We're going to set up Vagrant and VirtualBox for easy testing with Ansible to provision a new server.

Your first local server: Setting up Vagrant

To get started with your first local virtual server, you need to download and install Vagrant and VirtualBox, and set up a simple Vagrantfile, which will describe the virtual server.

1. Download and install Vagrant and VirtualBox (whichever version is appropriate for your OS): - Download Vagrant[35] - Download VirtualBox[36] (when installing, make sure the command line tools are installed, so Vagrant works with it)
2. Create a new folder somewhere on your hard drive where you will keep your Vagrantfile and provisioning instructions.
3. Open a Terminal or PowerShell window, then navigate to the folder you just created.
4. Add a CentOS 7.x 64-bit 'box' using the `vagrant box add`[37] command: `vagrant box add geerlingguy/centos7` (note: HashiCorp's Vagrant Cloud[38] has a comprehensive list of different pre-made Linux boxes. Also, check out the 'official' Vagrant Ubuntu boxes in Vagrant's Boxes documentation[39].

[33] http://www.vagrantup.com/
[34] https://www.virtualbox.org/
[35] http://www.vagrantup.com/downloads.html
[36] https://www.virtualbox.org/wiki/Downloads
[37] http://docs.vagrantup.com/v2/boxes.html
[38] https://app.vagrantup.com/boxes/search
[39] https://www.vagrantup.com/docs/boxes.html

5. Create a default virtual server configuration using the box you just downloaded: `vagrant init geerlingguy/centos7`
6. Boot your CentOS server: `vagrant up`

Vagrant downloaded a pre-built 64-bit CentOS 7 virtual machine image (you can build your own[40] virtual machine 'boxes', if you so desire), loaded the image into VirtualBox with the configuration defined in the default Vagrantfile (which is now in the folder you created earlier), and booted the virtual machine.

Managing this virtual server is extremely easy: `vagrant halt` will shut down the VM, `vagrant up` will bring it back up, and `vagrant destroy` will completely delete the machine from VirtualBox. A simple `vagrant up` again will re-create it from the base box you originally downloaded.

Now that you have a running server, you can use it just like you would any other server, and you can connect via SSH. To connect, enter `vagrant ssh` from the folder where the Vagrantfile is located. If you want to connect manually, or connect from another application, enter `vagrant ssh-config` to get the required SSH details.

Using Ansible with Vagrant

Vagrant's ability to bring up preconfigured boxes is convenient on its own, but you could do similar things with the same efficiency using VirtualBox's (or VMWare's, or Parallels') GUI. Vagrant has some other tricks up its sleeve:

- **Network interface management**[41]: You can forward ports to a VM, share the public network connection, or use private networking for inter-VM and host-only communication.
- **Shared folder management**[42]: Vagrant sets up shares between your host machine and VMs using NFS or (much slower) native folder sharing in VirtualBox.
- **Multi-machine management**[43]: Vagrant is able to configure and control multiple VMs within one Vagrantfile. This is important because, as stated in the

[40] https://www.vagrantup.com/docs/virtualbox/boxes.html
[41] https://www.vagrantup.com/docs/networking/index.html
[42] https://www.vagrantup.com/docs/synced-folders/index.html
[43] https://www.vagrantup.com/docs/multi-machine/index.html

documentation, "Historically, running complex environments was done by flattening them onto a single machine. The problem with that is that it is an inaccurate model of the production setup, which behaves far differently."

- **Provisioning**[44]: When running `vagrant up` the first time, Vagrant automatically *provisions* the newly-minted VM using whatever provisioner you have configured in the Vagrantfile. You can also run `vagrant provision` after the VM has been created to explicitly run the provisioner again.

It's this last feature that is most important for us. Ansible is one of many provisioners integrated with Vagrant (others include basic shell scripts, Chef, Docker, Puppet, and Salt). When you call `vagrant provision` (or `vagrant up` the first time), Vagrant passes off the VM to Ansible, and tells Ansible to run a defined Ansible playbook. We'll get into the details of Ansible playbooks later, but for now, we're going to edit our Vagrantfile to use Ansible to provision our virtual machine.

Open the Vagrantfile that was created when we used the `vagrant init` command earlier. Add the following lines just before the final 'end' (Vagrantfiles use Ruby syntax, in case you're wondering):

```
# Provisioning configuration for Ansible.
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "playbook.yml"
end
```

This is a very basic configuration to get you started using Ansible with Vagrant. There are many other Ansible options[45] you can use once we get deeper into using Ansible. For now, we just want to set up a very basic playbook—a simple file you create to tell Ansible how to configure your VM.

Your first Ansible playbook

Let's create the Ansible `playbook.yml` file now. Create an empty text file in the same folder as your Vagrantfile, and put in the following contents:

[44] https://www.vagrantup.com/docs/provisioning/index.html
[45] https://www.vagrantup.com/docs/provisioning/ansible.html

```
---
- hosts: all
  become: yes
  tasks:
  - name: Ensure NTP (for time synchronization) is installed.
    yum: name=ntp state=present
  - name: Ensure NTP is running.
    service: name=ntpd state=started enabled=yes
```

I'll get into what this playbook is doing in a minute. For now, let's run the playbook on our VM. Make sure you're in the same directory as the Vagrantfile and new playbook.yml file, and enter `vagrant provision`. You should see status messages for each of the 'tasks' you defined, and then a recap showing what Ansible did on your VM—something like the following:

```
PLAY RECAP *********************************************************
default                    : ok=3    changed=1    unreachable=0    failed=0
```

Ansible just took the simple playbook you defined, parsed the YAML syntax, and ran a bunch of commands via SSH to configure the server as you specified. Let's go through the playbook, step by step:

```
---
```

This first line is a marker showing that the rest of the document will be formatted in YAML (read a getting started guide for YAML[46]).

```
- hosts: all
```

This line tells Ansible to which hosts this playbook applies. `all` works here, since Vagrant is invisibly using its own Ansible inventory file (instead of the one we created earlier in `/etc/ansible/hosts`), which just defines the Vagrant VM.

[46] http://www.yaml.org/start.html

```
3   become: yes
```

Since we need privileged access to install NTP and modify system configuration, this line tells Ansible to use `sudo` for all the tasks in the playbook (you're telling Ansible to 'become' the root user with `sudo`, or an equivalent).

```
4   tasks:
```

All the tasks after this line will be run on all hosts (or, in our case, our one VM).

```
5   - name: Ensure NTP daemon (for time synchronization) is installed.
6     yum: name=ntp state=present
```

This command is the equivalent of running `yum install ntp`, but is much more intelligent; it will check if ntp is installed, and, if not, install it. This is the equivalent of the following shell script:

```
if ! rpm -qa | grep -qw ntp; then
    yum install -y ntp
fi
```

However, the above script is still not quite as robust as Ansible's `yum` command. What if ntpdate is installed, but not ntp? This script would require extra tweaking and complexity to match the simple Ansible yum command, especially after we explore the yum module more intimately (or the `apt` module for Debian-flavored Linux, or `package` for OS-agnostic package installation).

```
7   - name: Ensure NTP is running.
8     service: name=ntpd state=started enabled=yes
```

This final task both checks and ensures that the ntpd service is started and running, and sets it to start at system boot. A shell script with the same effect would be:

```
# Start ntpd if it's not already running.
if ps aux | grep -q "[n]tpd"
then
    echo "ntpd is running." > /dev/null
else
    systemctl start ntpd.service > /dev/null
    echo "Started ntpd."
fi
# Make sure ntpd is enabled on system startup.
systemctl enable ntpd.service
```

You can see how things start getting complex in the land of shell scripts! And this shell script is still not as robust as what you get with Ansible. To maintain idempotency and handle error conditions, you'll have to do a lot more extra work with basic shell scripts than you do with Ansible.

We could be even more terse (and really demonstrate Ansible's powerful simplicity) and not use Ansible's `name` module to give human-readable names to each command, resulting in the following playbook:

```
---
- hosts: all
  become: yes
  tasks:
  - yum: name=ntp state=present
  - service: name=ntpd state=started enabled=yes
```

Just as with code and configuration files, documentation in Ansible (e.g. using the `name` function and/or adding comments to the YAML for complicated tasks) is not absolutely necessary. However, I'm a firm believer in thorough (but concise) documentation, so I almost always document what my tasks will do by providing a `name` for each one. This also helps when you're running the playbooks, so you can see what's going on in a human-readable format.

Cleaning Up

Once you're finished experimenting with the CentOS Vagrant VM, you can remove it from your system by running `vagrant destroy`. If you want to rebuild the VM again, run `vagrant up`. If you're like me, you'll soon be building and rebuilding hundreds of VMs and containers per week using Vagrant and Ansible!

Summary

Your workstation is on the path to becoming an "infrastructure-in-a-box," and you can now ensure your infrastructure is as well-tested as the code that runs on top of it. With one small example, you've got a glimpse at the simple-yet-powerful Ansible playbook. We'll dive deeper into Ansible playbooks later, and we'll also explore Vagrant a little more as we go.

```
 _______________________________________
/ I have not failed, I've just found   \
| 10,000 ways that won't work. (Thomas |
\ Edison)                              /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```