**Chapter 2**

* Create a new Ubuntu Server VM using the Ubuntu Server 20.04 template
* Before powering on the VM, go to the VMs settings and click Processors
* Check the box for ‘Virtualize Intel VT-x/EPT or AMD-V-RVI’
* Power on the Virtual Machine
* If you get an error message that says ‘VMware Workstation does not support nested virtualization on this host’, open a powershell window as Administrator
* In powershell, run ‘bcdedit /set hypervisorlaunchtype off’ and then reboot the computer
* After rebooting, open VMWare Workstation and try booting the Ubuntu Server VM again. The VM should boot successfully now
* Sign in to the VM and update the machine
* Add the repository for ansible by running ‘sudo apt-add-repository -y ppa:ansible/ansible’
* Update the repository by running ‘sudo apt-get update’
* Install Ansible by running ‘sudo apt-get install -y ansible’
* Make sure Ansible is properly installed by running ‘ansible --version’
* Next, install VirtualBox by running ‘sudo apt install virtualbox’
* Now add the repo for vagrant by running ‘curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -’ and ‘sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"’
* Now update the repo and install vagrant by running ‘sudo apt-get update && sudo apt-get install vagrant’
* Create a directory to keep your vagrantfile and provisioning instructions
* Cd into the directory you just created and add a CentOS box by running the command ‘vagrant box add geerlingguy/centos7
* Create a virtual server configuration using the CentOS box you just downloaded by running ‘vagrant init geerlingguy/centos7’
* Boot the CentOS server by running ‘vagrant up’
* Edit the Vagrantfile to use Ansible Provision on the CentOS box by adding the following lines just before the final ‘end’

**# Provisioning configuration for Ansible.**

**config.vm.provision "ansible" do |ansible|**

**ansible.playbook = "playbook.yml"**

**end**

* Now create an Ansible playbook file by running ‘touch playbook.yml’ in the same directory as your Vagrantfile
* Edit the playbook.yml file and add the following contents

**---**

**- hosts: all**

**become: yes**

**tasks:**

**- name: Ensure chrony (for time synchronization) is installed.**

**yum:**

**name: chrony**

**state: present**

**- name: Ensure chrony is running.**

**service:**

**name: chronyd**

**state: started**

**enabled: yes**

* Save the playbook.yml file and then run ‘vagrant provision’ to provision the CentOS box using the Ansible playbook you just created

**Chapter 3**

* Create a new folder on the local drive of the VM and cd into it
* Create a new blank Vagrantfile by running ‘touch Vagrantfile’
* Open the Vagrantfile with a text editor and add the following:

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

VAGRANTFILE\_API\_VERSION = "2"

Vagrant.configure(VAGRANTFILE\_API\_VERSION) do |config|

# General Vagrant VM configuration.

config.vm.box = "geerlingguy/centos8"

config.ssh.insert\_key = false

config.vm.synced\_folder ".", "/vagrant", disabled: true

config.vm.provider :virtualbox do |v|

v.memory = 512

v.linked\_clone = true

end

# Application server 1.

config.vm.define "app1" do |app|

app.vm.hostname = "orc-app1.test"

app.vm.network :private\_network, ip: "192.168.60.4"

end

# Application server 2.

config.vm.define "app2" do |app|

app.vm.hostname = "orc-app2.test"

app.vm.network :private\_network, ip: "192.168.60.5"

end

# Database server.

config.vm.define "db" do |db|

db.vm.hostname = "orc-db.test"

db.vm.network :private\_network, ip: "192.168.60.6"

end

end

* This Vagrantfile defines three servers and gives each one a unique hostname, machine name, and IP
* Save the Vagrantfile and navigate to the directory where it is located
* Enter ‘vagrant up’ to let Vagrant build the three VMs
* Still in the same directory, run ‘touch ansible.cfg’ to create an empty config file
* Open ansible.cfg with a text editor and on the top line type ‘[defaults]’
* On the next line type ‘inventory = hosts.ini’, and on the third line type ‘interpreter\_python = /usr/libexec/platform-python’
* Save the ansible.cfg file
* Now create the hosts.ini file by running ‘touch hosts.ini’
* Copy and paste the following into the hosts.ini file and save

# Lines beginning with a # are comments, and are only included for

# illustration. These comments are overkill for most inventory files.

# Application servers

[app]

192.168.60.4

192.168.60.5

# Database server

[db]

192.168.60.6

# Group 'multi' with all servers

[multi:children]

app

db

# Variables that will be applied to all servers

[multi:vars]

ansible\_user=vagrant

ansible\_ssh\_private\_key\_file=~/.vagrant.d/insecure\_private\_key

* The first block puts both of the application servers into the ‘app’ group
* The second block puts the database server into the ‘db’ group
* The third block tells ansible to define a new group called ‘multi’, with the app and db groups as child groups
* The fourth block adds variables to the multi group that will be applied to all servers within multi and its children
* Save the inventory file
* Now run ‘ansible multi -a “hostname”’ to check the hostnames of the VMs
* Run ‘ansible multi -a “df -h”’ to make sure the servers have disk space available
* Run ‘ansible multi -a “free -m”’ to make sure there is enough memory on our servers
* Run ‘ansible multi -a “date”’ to make sure the date and time on each server is in sync
* Install the chrony daemon using Ansible’s yum module to keep the time in sync: ‘ansible multi -b -m yum -a “name=chrony state=present”
* Next, use Ansible’s service module to make sure the chrony daemon is started and set to run on boot: ‘ansible multi -b -m service -a “name=chronyd state started \ enabled=yes”’
* Now let’s check to make sure our servers are synced closely to the time on a time server: ‘ansible multi -b -a “chronyc tracking”’
* Now let’s configure the application servers
  + Install pip on the application servers by running ‘ansible app -b -m yum -a “name=python3-pip state=present”’
  + Next, install Django on the application servers by running ‘ansible app -b -m pip -a “name=django<4 state=present”’
  + Check to make sure Django is installed and working properly by running ‘ansible app -a “python -m django --version”
* Next let’s configure the database server
  + Install MariaDB: ‘ansible db -b -m yum -a “name=mariadb-server state=present”’
  + Start MariaDB: ‘ansible db -b -m service -a “name=mariadb state=started \ enabled=yes”’
  + Now configure the firewall of the database server to ensure only the app servers can access the database:
    - Install firewalld: ansible db -b -m yum -a “name=firewalld state=present”
    - Start firewalld: ansible db -b -m service -a “name=firewalld state=started \ enabled=yes”
    - ansible db -b -m firewalld -a “zone=database state=present permanent=\yes”
    - ansible db -b -m firewalld -a “source 192.168.60.0/24 \ zone=database state=enabled permanent=yes”
    - ansible db -b -m firewalld -a “port=3306/tcp zone=database \ state=enabled permanent=yes”
  + Install PyMySQL: ansible db -b -m yum -a “name=python3-PyMySQL state=present”
  + Allow MySQL access for one user from app servers: ansible db -b -m mysql\_user -a “name=django host=% password=12345 \ priv=\*.\*:ALL state=present”
* Now lets manage users and groups on our servers
  + Add an admin group to the app servers for server administrators: ansible app -b -m group -a “name=admin state=present”
  + Now add a user named johndoe with a home folder: ansible app -b -m user -a “name=johndoe group=admin createhome=yes”
  + To delete the user account run ansible app -b -m user -a “name=johndoe state=absent remove=yes”