Use Terraform to create On-prem Infrastructure

For the purposes of learning, let's assume the Linux server being used is Ubuntu 18.04.

Download and Install Terraform

1. Update the system packages

sudo apt-get update -y

2. Download wget and unzip

sudo apt-get install wget unzip -y

3. Download Terraform package

sudo wget https://releases.hashicorp.com/terraform/0.12.18/terraform_0.12.18_linux_amd64.zip

4. Unzip

sudo unzip terraform_0.12.18_linux_amd64.zip

5. Move terraform executable file to bin folder. This directory is where normal users will run terraform from. Placing the 'terraform' executable file will make it possible to call it from anywhere in the system. This is possible because the directory automatically has a path export.

sudo mv terraform /usr/local/bin/

6. Confirm if terraform is installed correctly

terraform -v

Install KVM and Terraform provider for libvirt

Steps to be taken so that a virtual machine can be provisioned. To achieve this, the KVM hypervisor will be used. Furthermore, the interaction with the virtualization library (libvirt) used to manage KVM needs to be added as a provider in Terraform.

We can also do this for hyper-v or vsphere.

Hyper-v - https://registry.terraform.io/providers/taliesins/hyperv/latest/docs

vSphere - https://registry.terraform.io/providers/hashicorp/vsphere/latest/docs

1. Download/Install KVM and the required packages with following command. This will install a lot of packages and could take quite some time. Besides the KVM packages some other useful tools will be installed.

Bridge-utils - Used for debugging issues in a network, mainly a bridge network.
Virtinst - Tool that can be used to create virtual machines.
libguestfs-tools - Has some dependencies for cloud init.

sudo apt -y install qemu-kvm libvirt-bin virt-top libguestfs-tools virtinst bridge-utils

2. Start the libvirt service and enable auto starting on next startup using the following command.

sudo systemctl enable — now libvirtd

3. Verify libvirt status by running the following command and searching for **active (running)**.

sudo systemctl status libvirtd

Install Terraform libvirt provider

1. Run the following command to install GOlang and Git. Git will be used to build terraform provider from GitHub. We only use the GO compiler to build and install the plugin.

sudo apt -y install golang git

2. Install libvirt-dev due to its dependency for installing terraform-provider-libvirt with go.

sudo apt -y install libvirt-dev

3. Create and go in to a new directory using the command below. The format described in the following code seems to be a standard of installing terraform providers as the provider will search for the build files in the directories shown below.

sudo mkdir -p /usr/lib/go/src/github.com/dmacvicar/

cd /usr/lib/go/src/github.com/dmacvicar/

4. Clone the official git repository

sudo git clone https://github.com/dmacvicar/terraform-provider-libvirt.git

5. Go into the provider’s directory and build it with the following commands. The ‘make’ command will compile the provider and ‘make install’ will install it

```
cd terraform-provider-libvirt

sudo make install
```

6. The result of the build is an executable which is saved in the directory. Verify this with following command,

```
cd /usr/lib/go/bin

./terraform-provider-libvirt -version
```

7. We want to use libvirt as our Terraform provider. The plugin needs to be specifically added into *‘~/.terraform.d/plugins’* directory. This is the directory in which terraform will search for third party plugins when initialized.

```
mkdir ~/.terraform.d/plugins

sudo ln -s /usr/lib/go/bin/terraform-provider-libvirt ~/.terraform.d/plugins/

ls -l ~/.terraform.d/plugins/
```

Create a Virtual Machine using Terraform and KVM

Setup permissions

1. Libvirt requires the host machine’s user rights to be able to allocate computing resources. Run the following command to add your user to groups libvirt and KVM.

```
sudo usermod -aG libvirt,kvm <host machine’s username>
```

2. The groups are not yet enabled for our current section. Re-login to your user with the ‘su — ${user}‘ command. After that list the groups the user is a member of with the id command.

```
su — <host machine’s username>

id -nG
```

3. Turn off SElinux security driver for libvirt by editing a file that can be found at ‘/etc/libvirt/qemu.conf’. The security driver needs to be turned off because it will prevent libvirt from creating a libvirt-domain. Open the file and change the line ‘security_driver=”selinux”’ to ‘security_driver = “none”

```
sudo nano /etc/libvirt/qemu.conf
```

4. Save and exit

5. Restart libvirt service to make the changes effective

sudo systemctl restart libvirtd

Setup the project workspace

1. Make a project directory

mkdir ~/kvm_project/

cd ~/kvm_project/

mkdir downloads

2. Create a new directory to save images. This where the disks of the virtual machine will be found.

sudo mkdir -p /libvirt_images/

3. Download the image to downloads directory in the project using the following command,

wget https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img

Generate an SSH key

1. Run the following command to generate the SSH keys,

ssh-keygen

2. Look into the permissions of the created keys to verify if permissions changed correctly by running the following command and comparing the output. You will see two keys. One of the keys is a public key and a private key. The difference can be seen in the ***.pub*** extension of the public key. Since this is the one we are going to share it also requires read writes

ls -al ~/.ssh/

<u>Setup a terraform Configuration to provision a guest virtual machine</u>

1. Go into the project directory.

cd ~/kvm_project/

2. Create a new terraform configuration file

vim libvirt.tf

3. Add following lines of code,

```
# libvirt.tf
# add the provider
provider “libvirt” {
 uri = “qemu:///system”
}
```

The first thing that is required is the provider ‘*terraform-libvirt-provider*’. To enable this in the code, a call needs to be added. This code will connect to QEMU Hypervisor.

```
# create pool
resource “libvirt_pool” “ubuntu” {
 name = “ubuntu-pool”
 type = “dir”
 path = “/libvirt_images/ubuntu-pool/”
}
```

Create a libvirt pool. This a quantity of storage set aside for use of virtual machine images. The *name* argument is used to identify a pool. The pool type ‘dir’ provides the means to manage files within a directory. These files can be a special disk format such as ***‘qcow’*** or ***‘vmdk’***. Finally, the path argument is where the storage is and where the images of the machine are going to be saved.

```
# create image
resource “libvirt_volume” “image-qcow2” {
 name = “ubuntu-amd64.qcow2”
 pool = libvirt_pool.ubuntu.name
 source =”${path.module}/downloads/bionic-server-
cloudimg-amd64.img”
 format = “qcow2”
}
```

Create a new resource for the virtual machine image disk. Add the following lines to create a new image resource. This will create a new image by cloning it from a given source and saving it in the path to the pool created in the last step. The source can be a directory or even an HTTP URL to the actual download.

```
# add cloudinit disk to pool
resource “libvirt_cloudinit_disk” “commoninit” {
 name = “commoninit.iso”
 pool = libvirt_pool.ubuntu.name
 user_data = data.template_file.user_data.rendered
}
```

Create a cloud init disk and load a cloud init configuration. This disk will be used to share user data found in the ‘cloud_init.cfg’ file with the guest VM. An image will be created and saved in the same storage pool as the virtual machine we are going to create. The machine’s disk (the libvirt volume) and the cloud init disk need to be in the same pool to share cloud init configuration.

This file is not automatically loaded by Terraform. To load the data a call needs to be made from a ‘.tf’ configuration file.

```
# read the configuration
data “template_file” “user_data” {
 template = file(“${path.module}/cloud_init.cfg”)
}
```

```
# Define KVM domain to create

resource “libvirt_domain” “test-domain” {

 # name should be unique!
   name = “test-vm-ubuntu”
   memory = “1024”
   vcpu = 1

 # add the cloud init disk to share user data
   cloudinit = libvirt_cloudinit_disk.commoninit.id

# set to default libvirt network
   network_interface {
   network_name = “default”
 }
console {
   type = “pty”
   target_type = “serial”
   target_port = “0”
 }
disk {
   volume_id = libvirt_volume.image-qcow2.id
 }
graphics {
   type = “spice”
   listen_type = “address”
   autoport = true
 }
}
```

Create a new libvirt domain. This where resources to support the guest machine are added. This is basically where the virtual machine is created and will determine the performance of the created machine but also the networking functionality. You can change the name of the memory and the vcpu depending on the resources available for you.

Setting up cloud-init for configuring the guest virtual machine system on creation

1. Go into the project directory and create a new file called ‘*cloud_init.cfg’.*

cd ~/kvm_project/

vim cloud_init.cfg

Cloud-init is the industry standard multi-distribution method for cross-platform cloud instance initialization. It is supported across all major public cloud providers, provisioning systems for private cloud infrastructure, and bare-metal installations.

2. Add the following code

```
#cloud-config
users:
 — name: <hostname>
 sudo: ALL=(ALL) NOPASSWD:ALL
 groups: users, admin
 home: /home/<hostname>
 shell: /bin/bash
 ssh_authorized_keys:
 — <public key>
```

The *name* argument is the name of a user that is going to be added. Adding more name arguments will result in multiple users.

This argument requires a list of public keys which are going to be added under the ‘*authorized_keys’* in directory ‘*~/.ssh/authorized_keys’* of the guest machine. This way we will be able to login into the user using our private key in the same directory.

```
# install packages
packages:
 — qemu-guest-agent
 — git
 — python
 — wget
```

This will attempt to install packages after the machine is created.
Note that we added python wget, two programs used to install the PiCalc service.

```
# run command after boot
runcmd:
 — [“cd”, “/home/terraform_guest”]
 — [“git”, “clone”, “https://github.com/krebsalad/
PiCalcPy.git"]
 — [“cd”, “PiCalcPy”]
 — [“python”, “install_picalc.py”]
 — [“python”, “run.py”, “mode=server”,”&”]
```

PiCalc is a HTTP server with which Pi can be calculated to a given precision.
To start the server after installing, we will have to run some *bash* commands. These steps should be enough to run the server automatically when it’s started.

Provisioning the guest virtual machine

1. Initialize the project workspace

terraform init

2. Apply the changes with following command

terraform apply

Verify if all our resources are created correctly

1. Run the following command to list all virtual machines (Domains) created with libvirt

virsh list — all

‘virsh’ is a tool used from command line to create and manage libvirt resources.

2. To list the existing volumes run in a certain pool run the following command.

virsh vol-list <pool name>

You should see both the VM disk and cloudinit disk in this pool.

3. List all libvirt networks with the command below.

virsh net-list --all

4. Find the IP-address DHCP leased to the VM

virsh net-dhcp-leases <network name>

ping <IP>

5. When the installation is done, test the server by sending a HTTP request to the IP address from the CLI or from your host's browser.

curl <IP>:8080/PiCalc/100

Scalability

We can use variables to accommodate different terraform configuration for different machines. We can use modules to set the values for different variables.

1. Create a new module directory called '*ubuntu-module'*. This directory will have the required files to create a virtual machine.

mkdir -p ~/kvm_project/modules/ubuntu-module/

2. Move the file '*libvirt.tf'* to the module directory. We will be changing the *'libvirt.tf'* file so that its reusable.

mv ~/kvm_project/libvirt.tf ~/kvm_project/modules/ubuntu-module

3. Edit the libvirt.tf file as given below,

```
#libvirt.tf

provider “libvirt” {
 uri = “qemu:///system”
}

variable “machine_name” {
type = string
}

variable “network_name” {
type = string
}

variable “mac_address” {
}

variable “user_data_path”{
}


resource “libvirt_pool” “ubuntu” {
 name = “${var.machine_name}_pool”
 type = “dir”
 path = “/libvirt_images/${var.machine_name}_pool/”
```

```
}

resource “libvirt_volume” “image-qcow2”{
 name = “${var.machine_name}_image.qcow2”
 pool = libvirt_pool.ubuntu.name
 source = “${path.module}/../../downloads/bionic-server-cloudimg-amd64.img”
 format = “qcow2”
}




resource “libvirt_domain” “test-domain” {
   name = “${var.machine_name}_domain”
   memory = “1024”
   vcpu = 1

network_interface {
   network_name = “${var.network_name}”
   mac = “${var.mac_address}”
}

data “template_file” “user_data” {
   template = file(“${var.user_data_path}”)
}
```

4. Save the *‘libvirt.tf’* file and close

5. To make use of the module a new file will be created in the main directory of the project. This file will be used to call multiple modules, therefore it will be called *main.tf*.

cd ~/kvm_project/

vim main.tf

6. In this file we will call the module and set the variables we added earlier.

```
#main.tf
# picalc server 1
module “picalc-server-1” {
   # load the module
   source = “./modules/ubuntu-module/”
   # set the variables
   machine_name = “server_1”
   network_name = “picalc_net”
   mac_address = “52:54:00:6c:3c:03”
   user_data_path = “${path.module}/cloud_init.cfg”
}
# picalc server 2
module “picalc-server-2” {
   # load the module
   source = “./modules/ubuntu-module/”
   # set the variables
   machine_name = “server_2”
   network_name = “picalc_net”
   mac_address = “52:54:00:6c:3c:04”
   user_data_path = “${path.module}/cloud_init.cfg” }
```

The path to the cloud init is in the directory of this file. This means we can use ‘*path.module’* function to get our current directory.
This will be useful if we are trying to run different types of servers/applications on each VM.
Each cloudinit config file can be customised to accommodate different applications.

References

1. How To Provision VMs on KVM with Terraform - https://computingforgeeks.com/how-to-provision-vms-on-kvm-with-terraform/

2. How To Enable Virsh Console Access For KVM Guests - https://ostechnix.com/how-to-enable-virsh-console-access-for-kvm-guests/

3. Cloud-init Documentation - https://cloudinit.readthedocs.io/en/latest/