**17 - R710 Proxmox Ubuntu cloud-init image - Terraform - Ansible**

This sets up a bare-bones minimal system with a very small (~2GB disk) Ubuntu box via cloud-init and is then further provisioned / adjusted with Terraform and Ansible

Most instructions came from this link:

<https://austinsnerdythings.com/2021/08/30/how-to-create-a-proxmox-ubuntu-cloud-init-image/>

and:

<https://austinsnerdythings.com/2021/09/01/how-to-deploy-vms-in-proxmox-with-terraform/>

from this video:

<https://www.youtube.com/watch?v=1sPG3mFVafE>

… with number of things fixed .. .but is still not suitable for production … but fine for homelab.

Base system has Proxmox Version 7.1.2 installed.

Do these steps:

1. Open proxmox root terminal and as **root**, add ‘sudo’ to proxmox:  
   **apt-get install sudo**
2. Also as root, do:  
   **apt update -y && apt install libguestfs-tools -y**
3. Open proxmox root terminal and as root, create user ‘rhys’:  
   **adduser rhys**  
   **usermod -a -G sudo rhys**

**usermod -a -G kvm rhys**

1. Install midnight commander for filesystem easy navigation and file editing:  
   **apt install mc**
2. From new Terminal, log into ‘rhys’ and do:

**ssh-keygen -t rsa -b 4096**

1. Create directory named: ‘**test1**’ and ‘**cd**’ into it.
2. Create directory named: ‘original’ and ‘cd’ into this.  
   Then we download the base Ubuntu image with:  
   **wget** [**https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img**](https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img)
3. Do: ‘**cd ..**’
4. Create file: ‘create-template.sh’:

**touch create-template.sh**  
**chmod +x create-template.sh**

in this file put (and adjust user and root passwords to your values):

#!/usr/bin/bash

# run this with sudo

# change Data2 to local-lvm when needed

set -Eeuxo pipefail

image\_name = focal-server-cloudimg-amd64.img

[ -e ${image\_name} ] && rm ${image\_name}

echo "Copying original image to apply modifications against"

cp original/${ image\_name } ${ image\_name }

# Modify files to allow login

virt-edit -a ${image\_name} /etc/cloud/cloud.cfg -e 's/lock\_passwd: [Tt]rue/lock\_passwd: False/'

virt-edit -a ${image\_name} /etc/cloud/cloud.cfg -e 's/lock\_passwd: 1/lock\_passwd: 0/'

virt-edit -a ${image\_name} /etc/cloud/cloud.cfg -e 's/ssh\_pwauth: 0/ssh\_pwauth: 1/'

#virt-edit -a ${image\_name} /etc/ssh/sshd\_config -e 's/PasswordAuthentication no/PasswordAuthentication yes/'

virt-edit -a ${image\_name} /etc/ssh/sshd\_config -e 's/PubkeyAuthentication no/PubkeyAuthentication yes/'

virt-edit -a ${image\_name} /etc/ssh/sshd\_config -e 's/\#PubkeyAuthentication yes/PubkeyAuthentication yes/'

virt-customize -a ${ image\_name } --install qemu-guest-agent

# inject the SSH keys into the cloud image itself before turning it into a template and VM.

# You need to create a user first and the necessary folders:

virt-customize -a ${image\_name} --run-command 'useradd rhys'

virt-customize -a ${image\_name} --password rhys:password:userblafixme

virt-customize -a ${image\_name} --run-command 'mkdir -p /home/rhys/.ssh'

virt-customize -a ${image\_name} --ssh-inject rhys:file:/home/rhys/.ssh/id\_rsa.pub

virt-customize -a ${image\_name} --run-command 'chown -R rhys:rhys /home/rhys'

virt-customize -a ${image\_name} --run-command 'chmod 700 /home/rhys/.ssh'

virt-customize -a ${image\_name} --run-command 'usermod -aG sudo rhys'

virt-customize -a ${image\_name} --root-password password:blafixme

# Create Proxmox VM using modified image

qm create 9000 --name "ubuntu-2004-cloudinit-template" --memory 8096 --cores 2 --net0 virtio,bridge=vmbr0

qm importdisk 9000 ${ image\_name } Data2

qm set 9000 --scsihw virtio-scsi-pci --scsi0 Data2:vm-9000-disk-0

qm set 9000 --boot c --bootdisk scsi0

qm set 9000 --ide2 Data2:cloudinit

qm set 9000 --serial0 socket --vga serial0

qm set 9000 --agent enabled=1

# Resize the primary boot disk (otherwise it will be around 2G by default)

# This step adds another 8G of disk space, but change this as you need to

# NOTE: this will get over-ridden by Terraform later

qm resize 9000 virtio0 +8G

# Convert VM to a template

qm template 9000

# Clone the template into a full VM and set some parameters

qm clone 9000 999 --name test-clone-cloud-init

#qm set 999 --sshkey ~/.ssh/id\_rsa.pub

qm set 999 --ipconfig0 ip=192.168.124.207/24,gw=192.168.124.1

qm start 999

1. Create a file: ‘delete-template.sh’  
     
   **touch delete-template.sh**

**chmod +x delete-template.sh**

in this file put:

#!/usr/bin/bash

# run this with sudo

qm stop 999

qm destroy 999

qm destroy 9000

1. To create the template, create VM from template and run, do:

**sudo ./create-template.sh**The above must be run as root to have access to the ‘**qm**’ commands in the script.

1. To log into the running machine, (from where one is currently logged in – on proxmox):

**ssh** [**rhys@192.168.124.207**](mailto:rhys@192.168.124.207)

**NOTE:** the above log in can not be done from WSL terminal -> see other document for Terraform usage of cloud-init image where it gives the new VM an initial ssh key that matches the one on the ‘machine’ running the terraform script.

1. When done, log out of the machine
2. To delete the VM and template, do:  
   **sudo ./delete-template.sh**

Before repeating and running the above process again, the last entry in **~/.ssh/known\_hosts** will need to be deleted with:  
**ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "192.168.124.207"**

1. Once you are happy with the template, don’t finish up with running: ‘**delete-template.sh**’, but instead to keep the template for use by terraform later on do:  
   **sudo qm stop 999**

**sudo qm destroy 999**

1. **-=-=-**

Using: **ubuntu-20.04.2-live-server-amd64.iso**

Create a Ubuntu server VM named ‘**u1**’ in which to do the following, also in this new VM create user rhys and do:  
  
**ssh-keygen -t rsa -b 4096**

1. put following into test2 directory:  
     
   in file **1-install-terraform.sh**

**curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -**

**sudo apt-get install software-properties-common**

**sudo apt-get update**

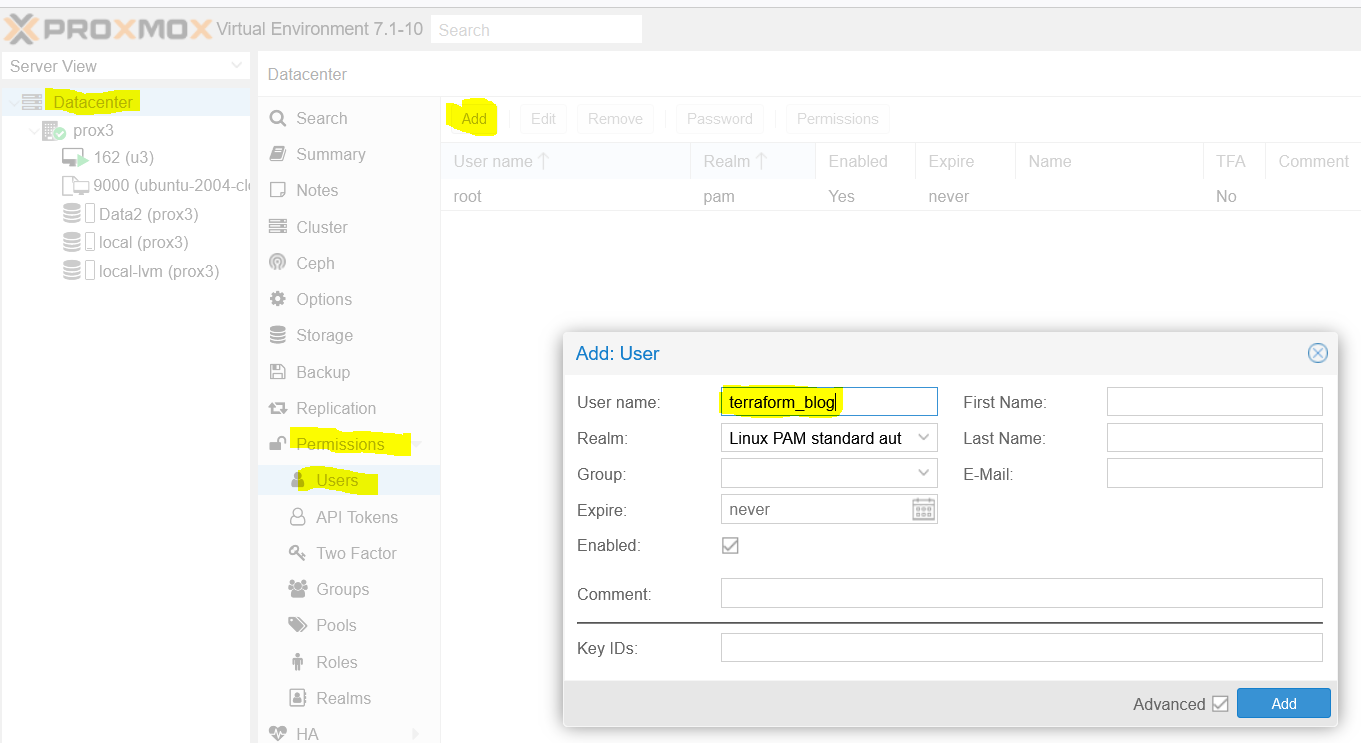
**sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"**

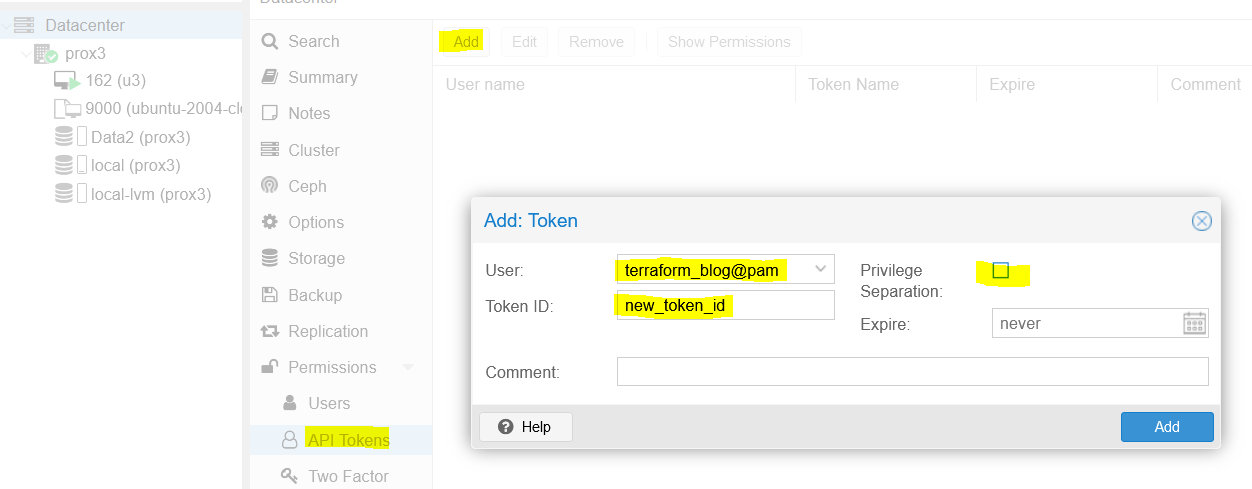
**sudo apt update**

**sudo apt install terraform**

make it executable and run it.  
  
(as of Feb 2022 the version of Terraform used is: Terraform v1.1.5 on linux\_amd64)

1. Copy steps from blog link:  
     
   **https://austinsnerdythings.com/2021/09/01/how-to-deploy-vms-in-proxmox-with-terraform/**  
     
   to create file: notes.txt, where I did:

  
and:



And copy and save the displayed token in notes.txt for use later in the terraform main.tf file in its section: ‘**provider "proxmox"**’:

Token Secret

Token ID:

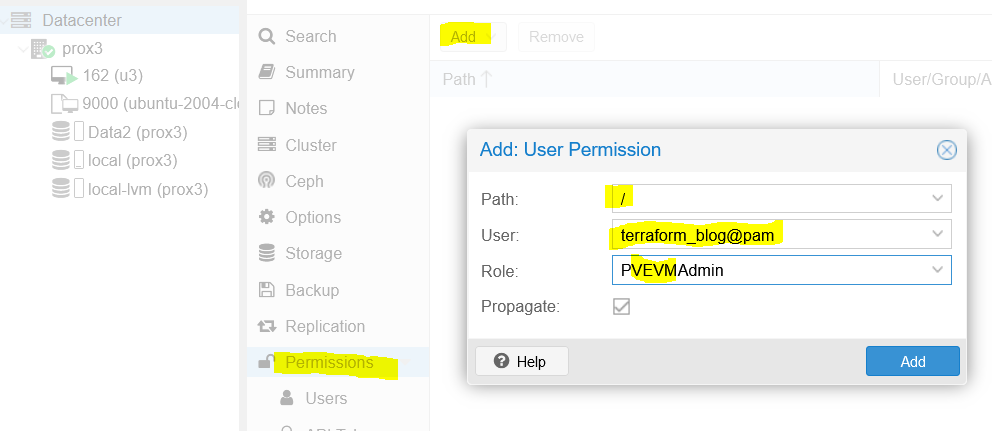
**terraform\_blog@pam!new\_token\_id**

Secret:

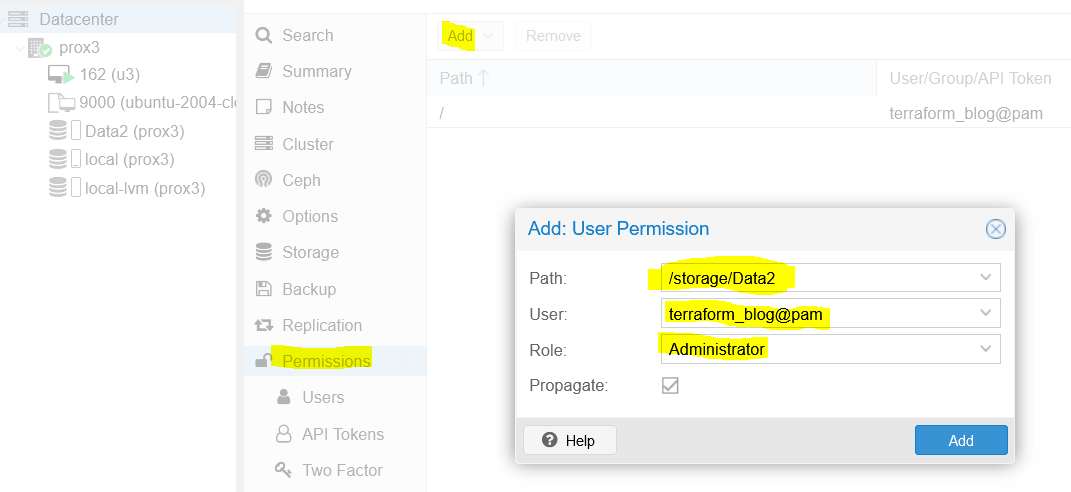
**0bca378d-8ce7-4282-9686-5046207b6ba3**

-=-=-

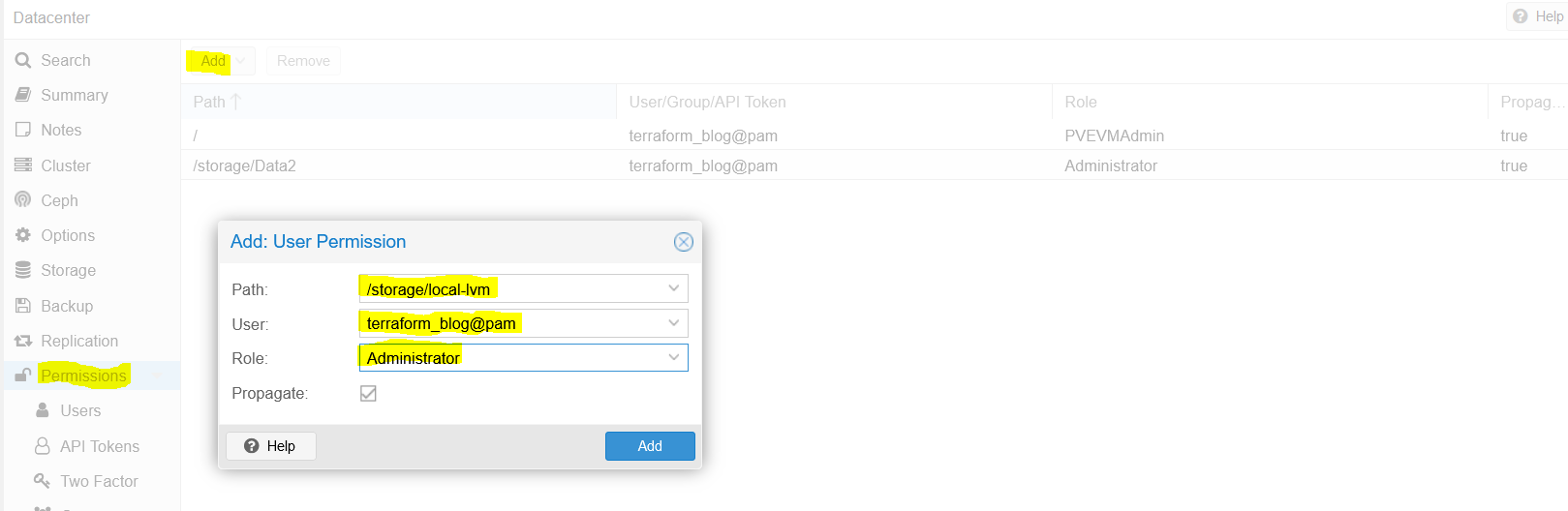
And:



And:



And:



-=-=-

1. In terraform directory, do:

create **main.tf** and **vars.tf** as per code further on.

Copy user ‘rhys’ (the user where this code is being placed and run from ) **id\_rsa.pub** file contents over the example in vars.tf

then run:

**terraform init**

**terraform plan**

**terraform apply**

then after logging in with:

**ssh rhys@192.168.124.163**

do:

**terraform destroy**

**ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "192.168.124.163"**

-=-=-

You can then log into the new vm as user 'rhys' from the terminal i had opened on prox1 as user 'rhys', utilising the ssh pub key.

**NOTE: More needs to be done in setting up the cloud-init for ‘user name/password’ log in without the ssh pub key stuff from any terminal on any machine.**

**NOTE: The Terraform script creates a second disc but does not put it into the file system.**  
**DETAILS:**  
When logged into new vm:

**df –h**

Filesystem Size Used Avail Use% Mounted on

udev 5.9G 0 5.9G 0% /dev

tmpfs 1.2G 1016K 1.2G 1% /run

/dev/sda1 25G 1.5G 23G 7% /

tmpfs 5.9G 0 5.9G 0% /dev/shm

tmpfs 5.0M 0 5.0M 0% /run/lock

tmpfs 5.9G 0 5.9G 0% /sys/fs/cgroup

/dev/loop0 62M 62M 0 100% /snap/core20/1328

/dev/loop1 68M 68M 0 100% /snap/lxd/21835

/dev/sda15 105M 5.2M 100M 5% /boot/efi

/dev/loop2 44M 44M 0 100% /snap/snapd/14549

tmpfs 1.2G 0 1.2G 0% /run/user/1000  
  
And the following shows the existence of second disc **sdb** (bits excluded to save space):

**sudo fdisk -l**

**Disk /dev/sdb: 100 GiB, 107374182400 bytes, 209715200 sectors**

Disk model: QEMU HARDDISK

Units: sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

**Disk /dev/sda: 25 GiB, 26843545600 bytes, 52428800 sectors**

Disk model: QEMU HARDDISK

Units: sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: gpt

Disk identifier: F939C69C-B7F6-4E7D-90FC-BD42C215FA09

Device Start End Sectors Size Type

/dev/sda1 227328 52428766 52201439 24.9G Linux filesystem

/dev/sda14 2048 10239 8192 4M BIOS boot

/dev/sda15 10240 227327 217088 106M EFI System

Partition table entries are not in disk order.

-=-=-

**Later there is a section that uses Ansible to setup: /dev/sdb**

The purpose of this second partition is for example a database to use.

Doing this saves on the OS having to waste resources via ‘quota’ to limit how much disk space the database is using.

If we only had one partition that the database shared with the OS it filled the disc, then the OS would crash.

But if the database fills the second partition, it would crash, but the OS would survive and have other software running that can report the failure and possibly apply any recovery procedures.

1. Then in folder: terraform

In file: **main.tf**  
  
terraform {

required\_providers {

proxmox = {

source = "telmate/proxmox"

version = "2.7.4"

}

}

}

provider "proxmox" {

# url is the hostname (FQDN if you have one) for the proxmox host you'd like to connect to to issue the commands. my proxmox host is 'a number'. Add /api2/json at the end for the API

pm\_api\_url = "https://192.168.124.161:8006/api2/json"

# api token id is in the form of: <username>@pam!<tokenId>

pm\_api\_token\_id = "terraform\_blog@pam!new\_token\_id"

# this is the full secret wrapped in quotes. don't worry, I've already deleted this from my proxmox cluster by the time you read this post

pm\_api\_token\_secret = "<the one you saved earlier>"

# leave tls\_insecure set to true unless you have your proxmox SSL certificate situation fully sorted out (if you do, you will know)

pm\_tls\_insecure = true

}

# resource is formatted to be "[type]" "[entity\_name]" so in this case

# we are looking to create a proxmox\_vm\_qemu entity named test\_server

resource "proxmox\_vm\_qemu" "test\_server" {

name = var.hostname

target\_node = var.proxmox\_host

# another variable with contents "ubuntu-2004-cloudinit-template"

clone = var.template\_name

vmid = var.vmid

full\_clone = true

# basic VM settings here. agent refers to guest agent

agent = 1

os\_type = "cloud-init"

cores = var.cores

sockets = 1

numa = true

cpu = "host"

memory = var.memory

scsihw = "virtio-scsi-pci"

onboot = false # !!! change this to true if eventually want all machines to start at power up

bootdisk = "scsi0"

disk {

# set disk size here. leave it small for testing because expanding the disk takes time.

ssd = var.ssd

size = var.rootfs\_size

type = "scsi"

storage = var.storage

iothread = 1

}

disk {

# set disk size here. leave it small for testing because expanding the disk takes time.

ssd = var.ssd

size = var.second\_partition\_size

type = "scsi"

storage = var.storage

iothread = 1

}

# if you want two NICs, just copy this whole network section and duplicate it

network {

model = "virtio"

bridge = "vmbr0"

}

# not sure exactly what this is for. presumably something about MAC addresses and ignore network changes during the life of the VM

lifecycle {

ignore\_changes = [

network,

]

}

# Cloud Init Settings

ipconfig0 = "ip=192.168.124.${var.vmid}/24,gw=192.168.124.1"

# sshkeys set using variables. the variable contains the text of the key.

# NOTE: this puts the key into the default 'ubuntu' user in file /home/ubuntu/.ssh/authorized\_keys

sshkeys = <<EOF

${var.ssh\_key}

EOF

connection {

type = "ssh"

user = "ubuntu"

# specify the key from 'this' host machine to establish ssh connection

private\_key = file("~/.ssh/id\_rsa")

agent = false

timeout = "3m"

host = "192.168.124.${var.vmid}"

}

provisioner "remote-exec" {

inline = [

"echo 'Cool, we are ready for ansible'",

"ls -alt /home",

# NOTE: the following shows that user rhys has the key from the original user on the proxmox hypervisor

# when the template was created that this terraform script uses as its 'base' image

"sudo cat /home/rhys/.ssh/authorized\_keys",

# So, now we replace with the key that we actually want that was placed in the user 'ubuntu'

# via the 'sshkeys' directive earlier on

"sudo cp /home/ubuntu/.ssh/authorized\_keys /home/rhys/.ssh/authorized\_keys",

# and show again to confirm change

"sudo cat /home/rhys/.ssh/authorized\_keys"

]

}

}  
  
-=-=-  
in file: **vars.tf**  
  
variable "ssh\_key" {

default = "ssh-rsa **< your public ssh key here>** == rhys@u1"

}

variable "proxmox\_host" {

default = "**prox1**"

}

variable "template\_name" {

default = "ubuntu-2004-cloudinit-template"

}

variable "vmid" {

default = 163

description = "Starting ID for the CTs, and also the IP address within the subnet"

}

variable "hostname" {

description = "VMs to be created"

type = string

default = "man"

}

variable "cores" {

default = 4

}

variable "memory" {

default = 12288

}

variable "rootfs\_size" {

type = string

default = "25G"

}

variable "second\_partition\_size" {

type = string

default = "100G"

}

variable "storage" {

type = string

default = "Data2"

}

variable "ssd" {

default = 1

}

1. If Terraform can’t seem to create / access a VM it might be that I’ve confused things by interrupting a previous creation / destroy.  
   Determine the proxmox ID of the problematic machine that can’t be stopped or deleted, say for example an ID of **224**, using this command:

**fuser /var/lock/qemu-server/lock-224.conf**  
  
to get its PID, with a return value of say **1872**, then do:

**kill -9 1872**

**qm stop 224**  
  
The VM should then be able to be deleted.

1. **-=-=-**  
   Ansibe:  
     
   On the base machine ‘u1’ do:

**sudo apt install ansible**

**sudo apt install sshpass**

(as of feb 2022 this installed version 2.9.6 of ansible)

1. From the previous terraform steps, ensure you have machine ‘man’ (192.168.124.163) running.
2. (useful links on how ansible.cfg, hosts and inventory files should be formatted):  
   https://github.com/ansible/ansible/blob/stable-2.9/examples/ansible.cfg

See lower section of this page for good example of hosts and inventory files:

<https://riptutorial.com/ansible>

and this:

<https://riptutorial.com/ansible/topic/1764/inventory>

and this makes a lot of sense:

<https://riptutorial.com/ansible/example/22593/hosts-file>

and this for reference:

<https://github.com/ansible/ansible/blob/stable-2.9/examples/ansible.cfg>

1. In the **test2** directory:
2. **mkdir ansible**

**cd ansible/**

**touch hosts**

and put the following in file ‘hosts’:

192.168.124.163

1. If you have not already ssh’d into 192.168.124.163 the do so now.
2. Run following command:  
   **ansible-inventory --list -y -i hosts**  
   to check a few things are OK to see:

*all:*

*children:*

*ungrouped:*

*hosts:*

*192.168.124.163: {}*

1. Next test, enter:  
   **ansible all -i hosts -m ping -u rhys**

To see:

*man | SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

If that does not work, you may need to try:  
**sshpass -p red ansible all -m ping -k**

1. Doing the following should get the same output as the last command above:  
   **ansible all -i hosts -m ping**
2. Create file **ansible.config** with contents:  
   [defaults]

INVENTORY = inventory

1. Create file **inventory** with contents:  
   # Servers

[servers]

man hostname=man ansible\_ssh\_host=192.168.124.163 ansible\_ssh\_user=rhys

# other

#192.168.124.164

and test with:

**ansible -m ping man**

1. Create file **provision.yml** with contents:  
   ---

- hosts: all

become: yes

tasks:

- name: Update and upgrade apt packages

become: yes

ignore\_errors: yes

register: command\_result

failed\_when: "'FAILED' in command\_result.stderr"

apt:

upgrade: "yes"

update\_cache: yes

cache\_valid\_time: 86400 #One day

Run the above with (and enter the sudo password that was embedded within the cloud init much earlier on):

**ansible-playbook --ask-become-pass provision.yml**

1. Create file **ntp.yml** with contents:  
   ---

- name: Set up NTP on all servers.

hosts: all

become: yes

tasks:

- name: Ensure NTP is installed.

apt: pkg=ntp state=present

- name: Ensure NTP is running.

service: name=ntp state=started enabled=yes

Run the above with (and enter the sudo password that was embedded within the cloud init much earlier on):  
**ansible-playbook --ask-become-pass ntp.yml**

Now ssh into 192.168.124.163 and enter:

**sudo su –**

and do this install:  
**apt install ntpstat**

then exit out of root and run command:

**ntpstat**

to see something like:

*synchronised to NTP server (185.83.169.27) at stratum 2*

*time correct to within 23 ms*

*polling server every 64 s*

1. -=-=-

Using Ansible:

Configure second disc /dev/sdb  
  
To look at disc information before we set up directory ‘**/work**’ do:  
**ansible man -m shell -a 'ls -alt /' --ask-become-pass**

1. Create file **disk-setup.yml.yml** with contents:  
   ---

- name: "Create partitions on sdb"

hosts: man

become: yes

gather\_facts: True

tasks:

- name: install parted

package:

name: parted

state: present

- name: Add new partition /dev/sdb1

parted:

device: /dev/sdb

number: 1

state: present

- filesystem:

fstype: ext4

dev: /dev/sdb1

- mount:

fstype: ext4

src: /dev/sdb1

path: /work

state: mounted

1. Run the above with:  
   **ansible-playbook --ask-become-pass disk-setup.yml**
2. Run the following to now see that the directory ‘**/work**’ now exists:  
   **ansible man -m shell -a 'ls -alt /' --ask-become-pass**
3. You can also see that /dev/sdb1 exists with:

**ansible man -m shell -a 'fdisk -l' -become --ask-become-pass**

1. You can see the free space on **/dev/sdb1** with:  
   **ansible man -m shell -a 'df -h' --ask-become-pass**
2. **APPENDIX**  
   other articles to investigate:

<https://whattheserver.com/proxmox-cloud-init-os-template-creation/>

<https://registry.terraform.io/providers/Telmate/proxmox/latest/docs/guides/cloud_init>

<https://registry.terraform.io/modules/sdhibit/cloud-init-vm/proxmox/latest>

<https://registry.terraform.io/providers/Telmate/proxmox/latest/docs/resources/vm_qemu>

<https://github.com/Telmate/terraform-provider-proxmox/blob/master/examples/cloudinit_example.tf>

See these for tips on enabling ssh password:  
<https://serverpilot.io/docs/how-to-enable-ssh-password-authentication/>

Post-install script to disable SSH password authentication, install latest Docker with AUFS on Ubuntu 14.04 VMs:  
<https://gist.github.com/parente/0227cfbbd8de1ce8ad05>

How to set "PasswordAuthentication" as yes through automation:

<https://askubuntu.com/questions/988845/how-to-set-passwordauthentication-as-yes-through-automation>

Using Packer: (do this exercise)

<https://salmonsec.com/blogs/home_lab_3>

the following seems useful to work through (**do it all to learn !!!**)

<https://yetiops.net/posts/proxmox-terraform-cloudinit-saltstack-prometheus/>

1. AOB …