**19 - R710 Proxmox Terraform VMs with different - CPU count - RAM and Disks**

These notes cover deploying multiple VM’s from the previously created run host.

The VM’s have different counts for CPU, different memory sizes, different disk sizes AND can all be ssh’d into.

It may not look pretty or be 100% perfect, but it works for me.

This document builds upon the previous two documents (and others):

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

18 - R710 Proxmox - Samba shares for W10 and W10 hosts file for WSL

# Terraform with modules, for multiple different VM deployments:

1. Log into rhys@run1
2. In the **public** folder, create directory **test6**  
   In **test6** directory create directory **terraform**  
   In **terraform** directory create directory **modules**
3. In the **modules** directory create file: **main.tf** with these contents:  
   (and adjust the highlighted values to your values):

terraform {

required\_version = "~> 1.1.5"

required\_providers {

proxmox = {

source = "telmate/proxmox"

version = "2.9.5"

}

}

}

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 'prox-1u'. 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

}

locals {

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

}

# 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=${local.ips}/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 = "${local.ips}"

}

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"

]

}

tags = "${var.hostname}"

}

1. In the **modules** directory create file: **vars.tf** with these contents:  
   (and adjust the highlighted values to your values):  
   That is, do: 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  
     
   variable "ssh\_key" {

default = "ssh-rsa **< your public ssh key here>** == rhys@run1"

}

variable "proxmox\_host" {

default = "prox3"

}

variable "template\_name" {

default = "ubuntu-2004-cloudinit-template"

}

variable "vmid" {

default = 159

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

}

variable "hostname" {

description = "VMs to be created"

type = string

default = "t1"

}

variable "cores" {

default = 2

}

variable "memory" {

default = 4096

}

variable "rootfs\_size" {

type = string

default = "10240M"

}

variable "second\_partition\_size" {

type = string

default = "4M"

}

variable "storage" {

type = string

default = "Data1"

}

variable "ssd" {

default = 0

}

variable "tags" {

type = string

default = "tag1"

}

variable "ip" {

type = string

default = "192.168.124.159"

}

1. In the **modules** directory create file: **outputs.tf** with these contents:  
   output "host\_name" {

value = var.hostname

}

output "public\_ips" {

value = local.ips

}

1. In the **terraform** directory create file: **main.tf** with these contents:  
   (you can un-comment whatever modules you want/need):  
     
   # NOTE:

# if the 'terraform apply' fails and its run again ... the disc size

# read back is in megabytes which does not then agree with specifying it as for example:

# "75G" for a disc size. It reads back as "75980M" ... so a second

# 'terraform apply' says that there is a change needed.

# So, disc sizes need to be expressed in Megabytes

module "man" {

source = "./modules"

hostname = "man"

vmid = 223

cores = 4

memory = 12288

ssd = 1

rootfs\_size = "25600M"

second\_partition\_size = "102400M"

storage = "Data2"

}

output "man\_ip" {

value = module.man.public\_ips

}

module "cw1" {

source = "./modules"

hostname = "cw1"

vmid = 224

cores = 2

memory = 8192

ssd = 1

rootfs\_size = "25600M"

# NOTE: This VM does not need a second partition, BUT giving it a size of "0M"

# caused the rootfs\_size partition to NOT be set up with the desired

# size of 25600M

# ... (this as of 11th Feb 2022 - i suspect a bug in the Telmate terraform provider)

#

# ... So, give second partition the minimum space (to not waste space) to ensure first

# partition is set up as desired:

#

second\_partition\_size = "4M"

storage = "Data2"

}

output "cw1\_ip" {

value = module.cw1.public\_ips

}

# module "cui" {

# source = "./modules"

# hostname = "cui"

# vmid = 225

# cores = 2

# memory = 4096

# ssd = 1

# rootfs\_size = "51200M"

# second\_partition\_size = "4M"

# storage = "Data2"

# }

# output "cui\_ip" {

# value = module.cui.public\_ips

# }

# module "bast" {

# source = "./modules"

# hostname = "bast"

# vmid = 226

# cores = 2

# memory = 4096

# ssd = 0

# rootfs\_size = "30720M"

# second\_partition\_size = "4M"

# storage = "local-lvm"

# }

# output "bast\_ip" {

# value = module.bast.public\_ips

# }

# module "web" {

# source = "./modules"

# hostname = "web"

# vmid = 227

# cores = 2

# memory = 8192

# ssd = 1

# rootfs\_size = "153600M"

# second\_partition\_size = "76800M"

# storage = "Data2"

# }

# output "web\_ip" {

# value = module.web.public\_ips

# }

# module "wasg1" {

# source = "./modules"

# hostname = "wasg1"

# vmid = 228

# cores = 2

# memory = 8192

# ssd = 1

# rootfs\_size = "51200M"

# second\_partition\_size = "76800M"

# storage = "Data2"

# }

# output "wasg1\_ip" {

# value = module.wasg1.public\_ips

# }

# module "z1" {

# source = "./modules"

# hostname = "z1"

# vmid = 229

# cores = 1

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "4M"

# storage = "local-lvm"

# }

# output "z1\_ip" {

# value = module.z1.public\_ips

# }

# module "n1" {

# source = "./modules"

# hostname = "n1"

# vmid = 230

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "51200M"

# second\_partition\_size = "51200M"

# storage = "local-lvm"

# }

# output "n1\_ip" {

# value = module.n1.public\_ips

# }

# module "k1" {

# source = "./modules"

# hostname = "k1"

# vmid = 231

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "76800M"

# storage = "local-lvm"

# }

# output "k1\_ip" {

# value = module.k1.public\_ips

# }

# module "n2" {

# source = "./modules"

# hostname = "n2"

# vmid = 232

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "51200M"

# second\_partition\_size = "51200M"

# storage = "local-lvm"

# }

# output "n2\_ip" {

# value = module.n2.public\_ips

# }

# module "n3" {

# source = "./modules"

# hostname = "n3"

# vmid = 233

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "51200M"

# second\_partition\_size = "51200M"

# storage = "local-lvm"

# }

# output "n3\_ip" {

# value = module.n3.public\_ips

# }

# module "z2" {

# source = "./modules"

# hostname = "z2"

# vmid = 234

# cores = 1

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "4M"

# storage = "local-lvm"

# }

# output "z2\_ip" {

# value = module.z2.public\_ips

# }

# module "z3" {

# source = "./modules"

# hostname = "z3"

# vmid = 235

# cores = 1

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "4M"

# storage = "local-lvm"

# }

# output "z3\_ip" {

# value = module.z3.public\_ips

# }

# module "k2" {

# source = "./modules"

# hostname = "k2"

# vmid = 236

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "76800M"

# storage = "local-lvm"

# }

# output "k2\_ip" {

# value = module.k2.public\_ips

# }

# module "k3" {

# source = "./modules"

# hostname = "k3"

# vmid = 237

# cores = 2

# memory = 8192

# ssd = 0

# rootfs\_size = "76800M"

# second\_partition\_size = "76800M"

# storage = "local-lvm"

# }

# output "k3\_ip" {

# value = module.k3.public\_ips

# }

1. In the **terraform** directory create executable script file: **add-sshs.sh** with these contents:  
   (and adjust its contents to suit your purposes):  
     
   ssh -o StrictHostKeyChecking=no man "uptime"

ssh -o StrictHostKeyChecking=no cw1 "uptime"

ssh -o StrictHostKeyChecking=no cui "uptime"

ssh -o StrictHostKeyChecking=no bast "uptime"

ssh -o StrictHostKeyChecking=no web "uptime"

ssh -o StrictHostKeyChecking=no wasg1 "uptime"

ssh -o StrictHostKeyChecking=no z1 "uptime"

ssh -o StrictHostKeyChecking=no n1 "uptime"

ssh -o StrictHostKeyChecking=no k1 "uptime"

ssh -o StrictHostKeyChecking=no n2 "uptime"

ssh -o StrictHostKeyChecking=no n3 "uptime"

ssh -o StrictHostKeyChecking=no z2 "uptime"

ssh -o StrictHostKeyChecking=no z3 "uptime"

ssh -o StrictHostKeyChecking=no k2 "uptime"

ssh -o StrictHostKeyChecking=no k3 "uptime"

1. In the **terraform** directory create executable script file: **clear-known-hosts.sh** with these contents:  
   (and adjust its contents to suit your purposes):  
     
   sudo truncate -s 0 ~/.ssh/known\_hosts
2. In the **terraform** directory create executable script file: **clear-sshs.sh** with these contents:  
   (and adjust its contents to suit your purposes):  
     
   ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "man"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "cw1"

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

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

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

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "wasg1"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "z1"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "n1"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "k1"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "n2"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "n3"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "z2"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "z3"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "k2"

ssh-keygen -f "/home/rhys/.ssh/known\_hosts" -R "k3"

1. Then in the **terraform** directory, run:  
   (NOTE: when creating and destroying VM’s multiple times with terraform you need to remove previous ssh info before creation and add new after creation)

**./clear-sshs**

**terraform init**

**terraform plan**

**terraform apply**

NOTE: Sometimes the apply fails to deploy all 15 machines … running terraform apply again seems to finish the job off ok. ( on the second apply, you can ignore the displayed differences in disk sizes, as it seems to work OK … though me thinks the telmate provider could be improved in this area)

1. Then run:  
     
   **./add-sshs.sh**
2. Then add the following to the file /etc/hosts for the run host “run1”:  
   (adjust this to suit your needs):

192.168.124.223 man

192.168.124.224 cw1

192.168.124.225 cui

192.168.124.226 bast

192.168.124.227 web

192.168.124.228 wasg1

192.168.124.229 z1

192.168.124.230 n1

192.168.124.231 k1

192.168.124.232 n2

192.168.124.233 n3

192.168.124.234 z2

192.168.124.235 z3

192.168.124.236 k2

192.168.124.237 k3

1. You can then log into the any of the new VM’s as user 'rhys' – i.e:

**ssh rhys@man**

1. To see what individual terraform resources can be created/destroyed, do:  
   **terraform state list**  
     
   which for say just two VM’s returns:  
   *module.cw1.proxmox\_vm\_qemu.test\_server*

*module.man.proxmox\_vm\_qemu.test\_server*

1. With many VM’s created, you can destroy a single VM with, for example:  
   **terraform destroy -target=module.cw1.proxmox\_vm\_qemu.test\_server**
2. You can deploy that single VM with, for example:  
   **terraform apply -target=module.cw1.proxmox\_vm\_qemu.test\_server**

1. You can also create and destroy more than one VM at a time with for example:  
   **terraform apply -target=module.cw1.proxmox\_vm\_qemu.test\_server -target=module.man.proxmox\_vm\_qemu.test\_server**

or:

**terraform destroy -target=module.cw1.proxmox\_vm\_qemu.test\_server -target=module.man.proxmox\_vm\_qemu.test\_server**

1. See the next section to finish off setting up the second disks with Ansible
2. If you need to tear down the VM’s, do:

**terraform destroy**

# Ansible setup of /dev/sdb:

1. This section assumes you have previously installed and set up Ansible as per the Ansible section in document: “17 - R710 Proxmox Ubuntu cloud-init image - Terraform - Ansible”
2. Copy the previous **ansible** directory to: **ansible-2**
3. In the **ansible-2** directory create file **inventory** with contents:  
   (adjust to suit your needs)  
     
   # Servers

[servers]

man hostname=man ansible\_ssh\_host=192.168.124.223 ansible\_ssh\_user=rhys

cw1 hostname=cw1 ansible\_ssh\_host=192.168.124.224 ansible\_ssh\_user=rhys

cui hostname=cui ansible\_ssh\_host=192.168.124.225 ansible\_ssh\_user=rhys

bast hostname=bast ansible\_ssh\_host=192.168.124.226 ansible\_ssh\_user=rhys

web hostname=web ansible\_ssh\_host=192.168.124.227 ansible\_ssh\_user=rhys

wasg1 hostname=wasg1 ansible\_ssh\_host=192.168.124.228 ansible\_ssh\_user=rhys

z1 hostname=z1 ansible\_ssh\_host=192.168.124.229 ansible\_ssh\_user=rhys

n1 hostname=n1 ansible\_ssh\_host=192.168.124.230 ansible\_ssh\_user=rhys

k1 hostname=k1 ansible\_ssh\_host=192.168.124.231 ansible\_ssh\_user=rhys

n2 hostname=n2 ansible\_ssh\_host=192.168.124.232 ansible\_ssh\_user=rhys

n3 hostname=n3 ansible\_ssh\_host=192.168.124.233 ansible\_ssh\_user=rhys

z2 hostname=z2 ansible\_ssh\_host=192.168.124.234 ansible\_ssh\_user=rhys

z3 hostname=z3 ansible\_ssh\_host=192.168.124.235 ansible\_ssh\_user=rhys

k2 hostname=k2 ansible\_ssh\_host=192.168.124.236 ansible\_ssh\_user=rhys

k3 hostname=k3 ansible\_ssh\_host=192.168.124.237 ansible\_ssh\_user=rhys

1. In the **ansible-2** directory create file **disk-setup.yml.yml** with contents:  
   (adjust to suit your needs)  
     
   ---

- name: "Create partitions on sdb"

hosts: all

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. For the “**man**” VM 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**