IT 445 – Capstone Implementation

Lab 03 - Advanced Infrastructure and Configuration Functions

1. **<u>Learning Objectives:</u>**
   - How to modify configuration files of different applications
2. **<u>Equipment/software:</u>**
   - Each member should have access to
     - Virtual Machine Images (Ubuntu 20.04 Server/Ubuntu 20.04 Desktop)
     - VMWare Workstation
3. **<u>Exercises</u>**

<u>MAKE SURE TO PROVIDE NETWORK DIAGRAMs AND TABLEs CONTAINING INFORMATION ABOUT THE MACHINES INVOLVED (SUCH AS, IP ADDRESSES, MAC ADDRESSES, ETC.)</u>

<u>NOTE THAT HAVING TO POST WIRESHARK PCAP/PCAPNG FILES IS NOT A SUBSITUTE FOR HAVING TO PROVIDE SCREEN SHOTS OF WIRESHARK CAPTURES AS EVIDENCE IN SUPPORT OF YOUR NARRATIVES AND ANSWERS.</u>

3.1 <u>Exercise 1: Preparation (it should be assigned to the students before class)</u>

Before starting, make sure that you have VMWare Workstation installed onto the machine currently used, whether that is the lab machine or your personal one. If you don't have VMWare installed, then go to the link below and follow the steps to install it onto your specific machine: https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion

<u>Step 1: What is Ansible?</u>

<u>T1.</u>

Appendix A shows the playbook that was used in Lab 1. There are tasks split up in between the playbook to identify actions that need to be done to a specific set of instances indicated in the hosts option. While tasks work as a way to split up what actions are being done, there are other methods that can be used that simplifies these playbooks and their tasks and make them easier to manage.

Roles are a feature in Ansible that lets you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure [4]. This gives the user an opportunity to seamlessly reuse the roles and the actions within them for other instances, which simplifies how the general playbook can be written. The process of setting up these roles are as followed.

Navigate to the directory where your ansible playbooks are and create the role structure needed with the command below:

ansible-galaxy init configure_routes

This command creates the role directory that will be used for this step.

Q1: Explain what was generated with the command and the usage for each folder created in the directory.

Next, we'll need to create a specific directory, named "group_vars", that organizes variables for specific groups of hosts or apply them globally. During the execution of a playbook, it loads the variables defined in the group_vars directory, with it then being applied to the hosts that are defined. In this case, with the way that it'll be set up, this will be applied to all of the hosts in our network architecture. This can be set up below:

Create the directory for the group_vars and the yml file for the roles:

mkdir group_vars
sudo vim all.yml

The all.yml file will contain the tasks that would occur on all of the vms. In this cas, we will set roles for configuring the routes for each of the vms so that universal connectivity can occur. The contents of this file can be seen in Appendix B.

Next, we will add the task for adding each route to the respective instance. To do this we will use a loop. A loop is a Ansible tool that execute a task multiple times. The statement "**loop: "{{ routes[inventory_hostname] }}"**" sets the loop, which sets the loop to execute for each host that is available in the all.yml file. Add these contents, which can be seen in Appendix C, to the main.yml which can be found in the configure_routes/tasks directory. Now that the roles are fully set up, add the role to the step4.yml as seen in Appendix D.

T2:

Now that the routes are set up using roles, lets do the same thing, but with the nginx configuration. Roles will be set up that configures nginx on the instances using created roles, We will do so as followed:

Generate the directory using the command below:

ansible-galaxy init geerlingguy.nginx

This will create the directory in the same way as before but will be set up a bit differently as the directory for the ip routes. Lets now set up a jinja2 template that will configure the nginx application. In the geerlingguy.nginx/handlers/ file directory, create a jinja2 file, nginx.conf.j2, that will replace the configuration file in the instances being targeted, with the contents of the j2 file being see in Appendix E.

Q2: What are the benefits of using a preexisting set up for tools such as geerlingguy.nginx?

Now, go to the main.yml file in the geerlingguy.nginx/tasks/ file directory and add the contents of Appendix F in the yml file. Then go to the main.yml file in the geerlingguy.nginx/handlers/ file directory and add the contents of Appendix G in the yml file.

In thr "group_vars/all.yml file, add the following line to set a port for the nginx application for the nginx configuration:

nginx_port: 80

Now, add the contents of Appendix H to step4.yml to add the nginx configuration changes.

T3:

Next, we will work to enhance how the ping tasks will work. Instead of there being individual tasks that test the pings for each of the instances and its interfaces, we can set it up using loops and variables to have it be dynamically executed. This can be done as followed:

Appendix I sets the pings up dynamically by looping through to each interface and its ip address. For each instance in the inventory.ini file, it will ping all of the ip addresses in the loop in the order that they are added in the loop. This will then return a result with the "ping_result" addition to then show the results of the ping, whether it is a success or failure.

These results can be logged to debug potential errors that may occur during the ping by adding the contents of Appendix I.

Step 2: What is Terraform?

T1:

The main.tf file from Lab 2 shows how terraform can fully develop an infrastructure from the ground up, with it creating the networks, subnets, ports, and instances. While not needed for the lab, there are ways in which some of these actions can be reduced in size. It is possible to dynamically generate any of the networks, subnets, and ports by using dynamic blocks.

The count argument takes in a whole number, which in the argument added is the numeric value being pulled from the port_count variable. Appendix M shows the variable being declared, with it being set to 5 which will be the number of ports being dynamically created.

Appendix N shows the creation of a new network and subnet, as well as the creation of 5 dynamically created ports using fixed IP addresses. Apply these contents to the main.tf file in their respective areas.

Run the main.tf file to enact these changes:

```
terraform init
terraform plan
terraform apply
```

If successful, the new network and subnet should be created, with the 5 ports being created and associated to the created subnet.

T2:

The depends_on block is used to handle different resources that Terraform can't directly access. In the main.tf file, the depends on block is added three times, with each enaction of the block relying the previous one. The first block is within the “wait_for_ssh" resource, with it containing the instances being created. This means that the “wait_for_ssh" resource won’t get enacted until each instance has been created. Each subsequent depends_on block relies on the one before it, with it not running its main resource block until the contents of the depends on block have been complete.

Q3: Show how a depends_on block can be added to the main.tf file to enhance its functionality.

T3:

Run Triggers allows the user to run configuration blocks on a queue, base upon the assumption that the block is successfully ran. In the main.tf file the triggers command is added on the “wait_for_ssh” resource, enacting the proviosner block to run every time based on the timestamp.

Lets add another resource that enacts a trigger block to check whether the subnet we just previously created has been officially created. Appendix “X” shows how that block will look like. The trigger will get enacted if the subnet that was created has been officially created, and if it has, then it will trigger a text saying that the subnet is created, all based off of the subnet ID.

Run the main.tf file to enact these changes:

```
terraform init
terraform plan
terraform apply
```

If successful, text should appear stating that the subnet that was created was successful.

Q4: How do timestamps work in Trigger blocks and how is it used here.

T4:

OpenStack Horizon isn’t the only resource that shows the resources that were created. Terraform has a console that can be ran from the location of the main.tf file, where these resources can be checked.

To access the console, navigate to the directory where the main.tf is located. Use the command to access the console:

```
Terraform console
```

From this console, multiple different types of information can be accessed, ranging from the name of the instances to the id of the instances to the networks within the instances.

Q5: Pull the name, network, and id of one of the instances from the console.

Step 3: What is Vagrant?

T1:

The Vagrantfile created in Lab 1 shows the complete configuration of the virtual network in VMWare and the creation of the infrastructure.

Once these virtual machines are created, there are ways to transfer files from the host machine to the created virtual machines, without the need for any commands through the terminal.

The vagrant directory in use is the main hub where the files can be created on the host machine. These files can then be accessed in the "/vagrant" directory from the created virtual machines. Let's create a directory and file with the purpose of it being shared to the virtual machines.

In the Vagrant directory, create a new directory and a new test txt file.

```
mkdir shared
sudo vim test.txt
```

(If on windows, create the file through notepad)

On one of the created virtual machines, go to the vagrant directory to verify that the directory and file are present in the VM:

```
cd /vagrant
cd shared
ls
```

While the Vagrantfile has already been created and executed, and the VMs have already been created, the Vagrantfile can still be re-executed with modifications being reloaded onto the VMs.

Add the command below to one of the VMs configurations in the Vagrantfile:

```
r1.vm.synced_folder "./shared", "/home/vagrant/testR1", create: true
```

Reload the VM to apply the changes;

```
vagrant reload R1
```

Q6: Add a shared file for R2, and show how you go about that process.

T2: Modular Network Configuration

While the way that the networks were set up before applied the networks to the correct VMs, it isn't the most efficient way to go about it. Defining the network configurations separately increases the efficiency of the vagrantfile, as it helps making changes to the network easier. Instead of having the details omf the network defined in each individual VM, it can be declared in a separate block and applied to a created vm, new or existing.

Add the contents of Appendix Q to the beginning of the Vagrantfile, outside of the "**Vagrant.configure("2") do |config|**" block. This sets the network configuration before the VMs get created, so if a new VM needs to get created, then it can get added in the new block and within the "**Vagrant.configure("2") do |config|**" block.

Q7: What makes this easier than setting it all in the vagrant config block?

T3: New VM Definition

In the original Vagrantfile, each VM gets defined within the "**Vagrant.configure("2") do |config|**" block with each option and detail being added to each VM block. This can be defined in its own block outside of the main "**Vagrant.configure("2") do |config|**" block. This makes the creation of the VM easier, as it can just call the VM creation block that's outside of the main block, and use the configuration that was set within it.

Use the contents of Appendix R to add the block to the outside of the "**Vagrant.configure("2") do |config|**" block. This sets up the details needed for each VM without it needing to be included in the VM configuration block.

Now use the contents of Appendix S to modify the Vagrantfile to use the new blocks added. This changes how the VMs are called, as it only contains the box needed to create the vm, the IP address(es) needed for the VMs and the settings based on the VMWare environment.

Run the Vagrantfile:

```
vagrant up
```

If successful, the VMs will get created in the same way as they would in the previous configuration.

Q8: What is still needed in the vagrantfile to create the instance in the main configure block?

Appendix A: step4.yml file

```
---
- name: Configure all VMs
  hosts: all
  become: yes
  tasks:
    - name: Install nginx
      apt:
        name: nginx
        state: present

    - name: Add default route for HostA
      when: inventory_hostname == 'HostA'
      shell: sudo ip route add default via 10.0.10.1

    - name: Add default route for HostB
      when: inventory_hostname == 'HostB'
      shell: sudo ip route add default via 10.0.30.1

    - name: Add routes for R1
      when: inventory_hostname == 'R1'
      shell: sudo ip route add 10.0.30.0/24 via 10.0.20.2 dev eth2

    - name: Add routes for R2
      when: inventory_hostname == 'R2'
      shell: sudo ip route add 10.0.10.0/24 via 10.0.20.1 dev eth1
```

```
- name: Enable IP forwarding on R1 and R2
  hosts: R1,R2
  become: yes
  tasks:
    - name: Ensure IP forwarding is enabled on boot
      lineinfile:
        path: /etc/sysctl.conf
        regexp: '^net.ipv4.ip_forward='
        line: 'net.ipv4.ip_forward=1'
        state: present

    - name: Apply sysctl changes
      shell: |
        sysctl -p

- name: Check connectivity between VMs
  hosts: all
  become: yes
  tasks:
    - name: Ping HostA from all VMs
      shell: ping -c 2 10.0.10.2
      ignore_errors: yes
      changed_when: false

    - name: Ping HostB from all VMs
      shell: ping -c 2 10.0.30.2
      ignore_errors: yes
      changed_when: false

    - name: Ping R1 eth1 from all VMs
      shell: ping -c 2 10.0.10.1
      ignore_errors: yes
      changed_when: false

    - name: Ping R1 eth2 from all VMs
      shell: ping -c 2 10.0.10.1
      ignore_errors: yes
      changed_when: false

    - name: Ping R2 eth1 from all VMs
      shell: ping -c 2 10.0.20.2
      ignore_errors: yes
      changed_when: false

    - name: Ping R2 eth2 from all VMs
      shell: ping -c 2 10.0.30.1
      ignore_errors: yes
      changed_when: false
```

Appendix B:

```
routes:
  HostA:
```

```
  - { dest: "default", via: "10.0.10.1" }
 HostB:
  - { dest: "default", via: "10.0.30.1" }
 R1:
  - { dest: "10.0.30.0/24", via: "10.0.20.2", dev: "eth2" }
 R2:
  - { dest: "10.0.10.0/24", via: "10.0.20.1", dev: "eth1" }
```

Appendix C: configure_routes/tasks/main.yml

```
- name: Add routes dynamically
 when: routes[inventory_hostname] is defined
 loop: "{{ routes[inventory_hostname] }}"
 command: >
  ip route add {{ item.dest == 'default' | ternary('', item.dest) }}
  via {{ item.via }}
  {% if item.dev is defined %} dev {{ item.dev }} {% endif %}
```

Appendix D: New Role to Configure Routes

```
- name: Configure routes
 hosts: all
 become: yes
 roles:
  - configure_routes
```

Appendix E: roles/geerlingguy.nginx/templates/nginx.conf.j2

```
server {
  listen {{ nginx_port }};
  server_name {{ inventory_hostname }};
  location / {
    root /var/www/html;
    index index.html index.htm;
  }
}
```

Appendix F: roles/geerlingguy.nginx/tasks/main.yml

```
- name: Deploy Nginx configuration
 template:
  src: templates/nginx.conf.j2
  dest: /etc/nginx/nginx.conf
 notify: Restart Nginx
```

Appendix G: roles/geerlingguy.nginx/handlers/main.yml

```
- name: Restart Nginx
 service:
  name: nginx
  state: restarted
```

Appendix H: Updated config for nginx

```
- name: Configure Nginx and routes
 hosts: all
 become: yes
 roles:
  - geerlingguy.nginx
```

Appendix I: Update to pings

```
- name: Verify network connectivity dynamically
  hosts: all
  become: yes
  tasks:
    - name: Ping other hosts
      loop:
        - { target: "10.0.10.2", name: "HostA" }
        - { target: "10.0.30.2", name: "HostB" }
        - { target: "10.0.10.1", name: "R1 eth1" }
        - { target: "10.0.20.2", name: "R2 eth1" }
        - { target: "10.0.50.1", name: "Non-existent Host" }
      shell: ping -c 2 {{ item.target }}
      ignore_errors: yes
      register: ping_result

    - name: Log ping results
      debug:
        msg: "Ping to {{ item.name }} ({{ item.target }}) {{ 'succeeded' if ping_result.rc == 0 else 'failed' }}"
```

Appendix J: main.tf updated with dynamic ports

```
resource "openstack_networking_network_v2" "Lab2-40" {
  name = "Lab2-40"
}
```

```
resource "openstack_networking_subnet_v2" "subnet_4" {
  name       = "subnet_4"
  network_id = openstack_networking_network_v2.Lab2-40.id
  cidr       = "10.0.40.0/24"
  ip_version = 4
  gateway_ip = "10.0.40.1"
}

resource "openstack_networking_port_v2" "dynamic_ports" {
  count      = var.port_count
  name       = "dynamic_port_${count.index}"
  network_id = openstack_networking_network_v2.Lab2-40.id

  fixed_ip {
    subnet_id  = openstack_networking_subnet_v2.subnet_4.id
    ip_address = "10.0.40.${count.index + 2}"
  }
}
```

Appendix K: main.tf with updated depends_on block

```
resource "null_resource" "configure_router" {
  depends_on = [openstack_networking_port_v2.dynamic_ports]

  provisioner "local-exec" {
    command = "echo 'Router configuration complete!'"
```

```
 }
}
```

Appendix L: triggers block

```
resource "null_resource" "check_subnet" {
 triggers = {
  subnet_created = openstack_networking_subnet_v2.subnet_4.id
 }

 provisioner "local-exec" {
  command = "echo 'Subnet ${openstack_networking_subnet_v2.subnet_4.id} exists!'"
 }
}
```

Appendix M: port_count

```
variable "port_count" {
 description = "ports to create dynamically"
 type       = number
 default    = 5
}
```

Appendix N: Creation of new network, subnet and dynamically created ports

```
resource "openstack_networking_network_v2" "Lab2-40" {
 name = "Lab2-40"
}
```

```
resource "openstack_networking_subnet_v2" "subnet_4" {
  name       = "subnet_4"
  network_id = openstack_networking_network_v2.Lab2-40.id
  cidr       = "10.0.40.0/24"
  ip_version = 4
  gateway_ip = "10.0.40.1"
}

resource "openstack_networking_port_v2" "dynamic_ports" {
  count      = var.port_count
  name       = "dynamic_port_${count.index}"
  network_id = openstack_networking_network_v2.Lab2-40.id

  fixed_ip {
    subnet_id  = openstack_networking_subnet_v2.subnet_4.id
    ip_address = "10.0.40.${count.index + 2}"
  }
}
```

Appendix O:

```
resource "null_resource" "configure_router" {
  depends_on = [openstack_networking_port_v2.dynamic_ports]

  provisioner "local-exec" {
    command = "echo 'Router configuration complete!'"
  }
}
```

Appendix P:

```
resource "null_resource" "check_subnet" {
  triggers = {
    subnet_created = openstack_networking_subnet_v2.subnet_4.id
  }

  provisioner "local-exec" {
    command = "echo 'Subnet ${openstack_networking_subnet_v2.subnet_4.id} exists!'"
  }
}
```

Appendix Q:

```
# Network Configurations
private_network_config = {
  "HostA" => { ip: "10.0.10.2", nic_type: "virtio", network: "VMnet2" },
  "HostB" => { ip: "10.0.30.2", nic_type: "virtio", network: "VMnet4" },
  "R1" => [
    { ip: "10.0.10.1", network: "VMnet2" },
    { ip: "10.0.20.1", network: "VMnet3" }
  ],
  "R2" => [
    { ip: "10.0.20.2", network: "VMnet3" },
    { ip: "10.0.30.1", network: "VMnet4" }
```

```
 ]
}
```

Appendix R:

```
# VM Provider Setup
def setup_vm(vm, box, ip, hostname, networks)
 vm.vm.box = box
 vm.vm.hostname = hostname
 networks.each do |net|
  vm.vm.network "private_network", ip: net[:ip], virtualbox__intnet: net[:network],
auto_config: true, nic_type: net[:nic_type]
 end
 vm.vm.provider "vmware_desktop" do |v|
  v.gui = true
  v.memory = 1024
  v.cpus = 1
 end
 vm.vm.provision "file", source: ssh_key_path, destination:
"/home/vagrant/.ssh/authorized_keys"
end
```

Appendix S: Full Vagrantfile

```
private_network_config = {
 "HostA" => { ip: "10.0.10.2", nic_type: "virtio", network: "VMnet2" },
 "HostB" => { ip: "10.0.30.2", nic_type: "virtio", network: "VMnet4" },
 "R1" => [
  { ip: "10.0.10.1", network: "VMnet2" },
  { ip: "10.0.20.1", network: "VMnet3" }
 ],
 "R2" => [
  { ip: "10.0.20.2", network: "VMnet3" },
  { ip: "10.0.30.1", network: "VMnet4" }
 ]
}

def setup_vm(vm, box, hostname, networks)
 vm.vm.box = box
 vm.vm.hostname = hostname
 networks.each do |net|
  vm.vm.network "private_network", ip: net[:ip], virtualbox__intnet: net[:network],
auto_config: true, nic_type: net[:nic_type]
 end
 vm.vm.provider "vmware_desktop" do |v|
  v.gui = true
  v.memory = 1024
  v.cpus = 1
 end

end

Vagrant.configure("2") do |config|
 config.vm.define "HostA" do |desktop2|
  setup_vm(desktop2, "gusztavvargadr/ubuntu-desktop-2004-lts", "HostA", [{ip:
"10.0.10.2", network: "VMnet2", nic_type: "virtio"}])
```

```
 end

 config.vm.define "HostB" do |desktop3|
  setup_vm(desktop3, "gusztavvargadr/ubuntu-desktop-2004-lts", "HostB", [{ip:
"10.0.30.2", network: "VMnet4", nic_type: "virtio"}])
 end

 config.vm.define "R1" do |r1|
  setup_vm(r1, "gusztavvargadr/ubuntu-server-2004-lts", "R1", [
   {ip: "10.0.10.1", network: "VMnet2", nic_type: "virtio"},
   {ip: "10.0.20.1", network: "VMnet3", nic_type: "virtio"}
  ])
  r1.vm.synced_folder "./shared", "/home/vagrant/testR1", create: true
 end

 config.vm.define "R2" do |r2|
  setup_vm(r2, "gusztavvargadr/ubuntu-server-2004-lts", "R2", [
   {ip: "10.0.20.2", network: "VMnet3", nic_type: "virtio"},
   {ip: "10.0.30.1", network: "VMnet4", nic_type: "virtio"}
  ])
 end
end
```