TP 5 : Introduction à la gestion automatisée avec OpenTofu et Ansible

Objectif :
Automatiser la création et la configuration des machines virtuelles et du réseau dans une infrastructure KVM avec OpenTofu pour la création des ressources et Ansible pour leur provisionnement.

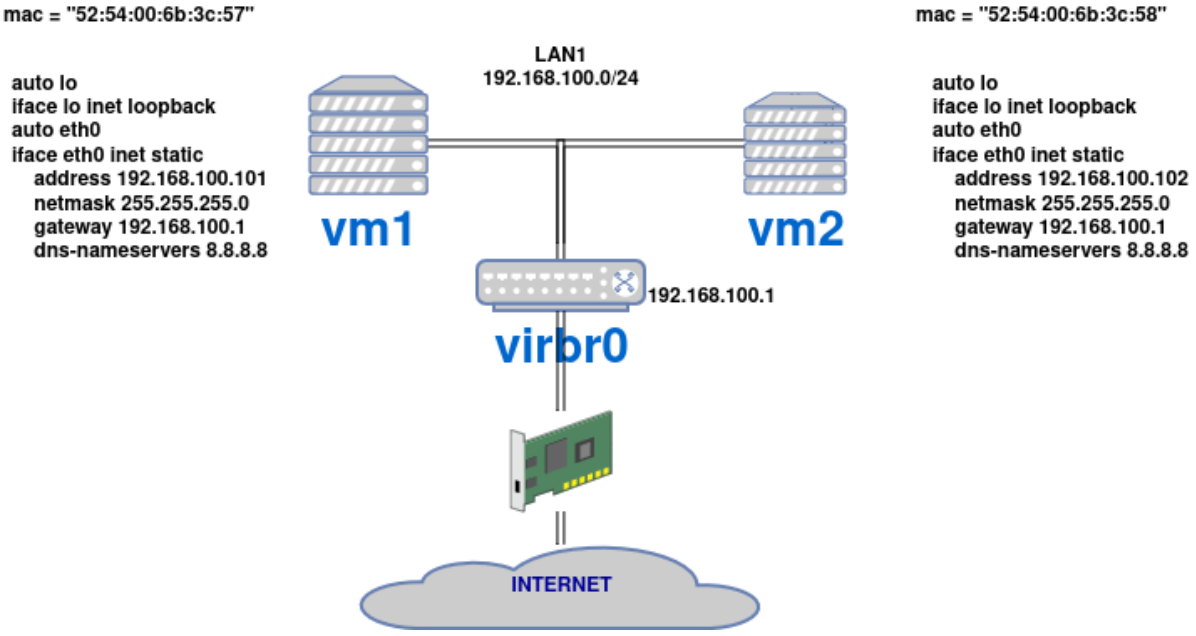
Outils :

- **OpenTofu** (alternative à Terraform)
- **Ansible**
- **KVM** (libvirt)

Prérequis :

- Lire attentivement le TP (J'ai résumé toutes les explications vues en cours)
- Ubuntu installé sur l'hôte avec KVM et libvirt configurés.
- Alpine Linux sera utilisé pour les machines virtuelles (VMs).
- Docs: https://docs.ansible.com/ansible/latest/index.html

Plan du TP :



Étape 1 : Préparation de l'environnement

1. **Installation d'OpenTofu sur l'hôte (Ubuntu)** :

```
#!/bin/bash


# Install opentofu
# Download the installer script:
curl --proto '=https' --tlsv1.2 -fsSL https://get.opentofu.org/install-
opentofu.sh -o install-opentofu.sh
```

```
# Alternatively: wget --secure-protocol=TLSv1_2 --https-only
https://get.opentofu.org/install-opentofu.sh -O install-opentofu.sh

# Give it execution permissions:
chmod +x install-opentofu.sh

# Please inspect the downloaded script

# Run the installer:
./install-opentofu.sh --install-method deb

# Remove the installer:
rm -f install-opentofu.sh

exit 0
```

2. **Installation d'Ansible sur l'hôte (Ubuntu)** :

```
sudo apt update
sudo apt install ansible -y
ansible --version
```

3. Tester le fonctionnement de ansible

```
ansible -m ping localhost
```

4. **Préparation des fichiers de configuration pour OpenTofu** : Créez un dossier de projet OpenTofu (par exemple `~/opentofu-tp`) et dans ce répertoire, créez les fichiers nécessaires :
    - `main.tf` : configuration d'OpenTofu pour créer les ressources.

Étape 2 : Création des machines virtuelles avec OpenTofu

Fichier `main.tf` :

```
terraform {
  required_providers {
    libvirt = {
      source = "dmacvicar/libvirt"
      version = "0.8.0"
    }
  }
}

provider "libvirt" {
  uri = "qemu:///system"
}

# Réseau pour les VMs
resource "libvirt_network" "lan1" {
  name      = "lan1"
```

```
  addresses = ["192.168.100.0/24"]
}

# Pool de stockage pour les disques des VMs
resource "libvirt_pool" "new_pool" {
  name = "new_pool"
  type = "dir"
  path = "/var/lib/libvirt/new_pool"
}

# Image de base Alpine Cloud
resource "libvirt_volume" "alpine_cloud" {
  name   = "alpine-cloud.qcow2"
  source = "https://dl-
cdn.alpinelinux.org/alpine/v3.20/releases/cloud/nocloud_alpine-3.20.3-x86_64-
bios-cloudinit-r0.qcow2"
  format = "qcow2"
  pool   = libvirt_pool.new_pool.name
}

data "template_file" "cloud_init_vm1" {
  template = <<EOF
#cloud-config
users:
  - name: root
    ssh_authorized_keys:
        - <votre_cle_ssh_public>

chpasswd:
  list: |
    root:root
  expire: False
fqdn: vm1
write_files:
  - path: /etc/ssh/sshd_config.d/50-cloud-init.conf
    content: |
        PubkeyAuthentication yes
        PasswordAuthentication yes  # Si tu veux utiliser l'authentification par
mot de passe
        ChallengeResponseAuthentication yes
        PermitRootLogin yes
  - path: /etc/network/interfaces
    content: |
      auto lo
      iface lo inet loopback
      auto eth0
      iface eth0 inet static
          address 192.168.100.101
          netmask 255.255.255.0
          gateway 192.168.100.1
          dns-nameservers 8.8.8.8
runcmd:
  - resize2fs /dev/vda
  - /sbin/service networking restart

EOF
```

```
}

# Cloud-init ISO pour passer la configuration réseau à la VM
resource "libvirt_cloudinit_disk" "commoninit_vm1" {
  name           = "commoninit_vm1.iso"
  pool           = "default"
  user_data      = data.template_file.cloud_init_vm1.rendered
}


data "template_file" "cloud_init_vm2" {
  template = <<EOF
#cloud-config
users:
  - name: root
    ssh_authorized_keys:
        - <votre_cle_ssh_public>
chpasswd:
  list: |
    root:root
  expire: False
fqdn: vm2
write_files:
  - path: /etc/ssh/sshd_config.d/50-cloud-init.conf
    content: |
        PubkeyAuthentication yes
        PasswordAuthentication yes  # Si tu veux utiliser l'authentification par
mot de passe
        ChallengeResponseAuthentication yes
        PermitRootLogin yes
  - path: /etc/network/interfaces
    content: |
      auto lo
      iface lo inet loopback
      auto eth0
      iface eth0 inet static
          address 192.168.100.102
          netmask 255.255.255.0
          gateway 192.168.100.1
          dns-nameservers 8.8.8.8
runcmd:
  - resize2fs /dev/vda
  - /sbin/service networking restart

EOF
}

# Cloud-init ISO pour passer la configuration réseau à la VM
resource "libvirt_cloudinit_disk" "commoninit_vm2" {
  name           = "commoninit_vm2.iso"
  pool           = "default"
  user_data      = data.template_file.cloud_init_vm2.rendered
}


# Volume pour VM1
```

```
resource "libvirt_volume" "vm1_volume" {
  name           = "vm1.qcow2"
  base_volume_id = libvirt_volume.alpine_cloud.id
  format         = "qcow2"
  pool           = libvirt_pool.new_pool.name
  size           = 409715200
}

# Volume pour VM2
resource "libvirt_volume" "vm2_volume" {
  name           = "vm2.qcow2"
  base_volume_id = libvirt_volume.alpine_cloud.id
  format         = "qcow2"
  pool           = libvirt_pool.new_pool.name
  size           = 409715200
}

# VM1
resource "libvirt_domain" "vm1" {
  name   = "vm1"
  memory = 512
  vcpu   = 1
  network_interface {
    network_id = libvirt_network.lan1.id
    hostname   = "vm1"
    mac = "52:54:00:6b:3c:58"
  }
  disk {
    volume_id = libvirt_volume.vm1_volume.id
  }
 cloudinit = libvirt_cloudinit_disk.commoninit_vm1.id
  console {
    type        = "pty"
    target_port = "0"
    target_type = "serial"
  }

  # Configuration d'un affichage graphique (ici, Spice est utilisé)
  graphics {
    type = "spice"
  }
}

# VM2
resource "libvirt_domain" "vm2" {
  name   = "vm2"
  memory = 512
  vcpu   = 1
  network_interface {
    network_id = libvirt_network.lan1.id
    hostname   = "vm2"
    mac = "52:54:00:6b:3c:57"
  }
  disk {
    volume_id = libvirt_volume.vm2_volume.id
  }
```

```
 cloudinit = libvirt_cloudinit_disk.commoninit_vm2.id
  console {
    type        = "pty"
    target_port = "0"
    target_type = "serial"
  }

  # Configuration d'un affichage graphique (ici, Spice est utilisé)
  graphics {
    type = "spice"
  }
}
```

Explication détaillée du playbook Terraform

Ce fichier Terraform configure une infrastructure de virtualisation basée sur **libvirt** pour créer deux machines virtuelles (VMs) avec une configuration réseau statique. Explication détaillée de chaque section avec des commentaires pour mieux comprendre son fonctionnement.

---

1. Configuration du fournisseur libvirt

```
terraform {
  required_providers {
    libvirt = {
      source = "dmacvicar/libvirt"
      version = "0.8.0"
    }
  }
}
```

- **terraform** : Spécifie la configuration générale pour Terraform, notamment les fournisseurs nécessaires.
- **required_providers** : Indique le fournisseur à utiliser (ici `libvirt`, version 0.8.0), qui permet à Terraform d'interagir avec l'hyperviseur libvirt.

```
provider "libvirt" {
  uri = "qemu:///system"
}
```

- **provider "libvirt"** : Configure l'URI de connexion à l'hyperviseur libvirt. L'URI `qemu:///system` signifie que nous utilisons **QEMU** comme hyperviseur, avec des privilèges système.

---

2. Configuration du réseau et du stockage

Réseau pour les VMs

```
resource "libvirt_network" "lan1" {
  name      = "lan1"
  addresses = ["192.168.100.0/24"]
```

```
}
```

- **libvirt_network "lan1"** : Crée un réseau virtuel nommé `lan1` avec la plage d'adresses IP `192.168.100.0/24`, que les VMs vont utiliser pour communiquer.

Pool de stockage

```
resource "libvirt_pool" "new_pool" {
  name = "new_pool"
  type = "dir"
  path = "/var/lib/libvirt/new_pool"
}
```

- **libvirt_pool "new_pool"** : Crée un pool de stockage nommé `new_pool`, de type "répertoire", qui stockera les volumes des disques dans le chemin spécifié `/var/lib/libvirt/new_pool`.

---

3. Téléchargement et configuration de l'image Alpine

Image Alpine

```
resource "libvirt_volume" "alpine_cloud" {
  name   = "alpine-cloud.qcow2"
  source = "https://dl-
cdn.alpinelinux.org/alpine/v3.20/releases/cloud/nocloud_alpine-3.20.3-x86_64-
bios-cloudinit-r0.qcow2"
  format = "qcow2"
  pool   = libvirt_pool.new_pool.name
}
```

- **libvirt_volume "alpine_cloud"** : Télécharge une image **Alpine Linux** pour le cloud à partir d'une URL externe, au format `qcow2`. Cette image sera utilisée comme base pour créer des volumes de disques pour les VMs.

---

4. Cloud-init pour la configuration des VMs

Configuration pour VM1

```
data "template_file" "cloud_init_vm1" {
  template = <<EOF
#cloud-config
users:
  - name: root
    ssh_authorized_keys:
        - ssh-rsa AAAAB3NzaC... consultant@nepturne

chpasswd:
  list: |
    root:root
  expire: False
fqdn: vm1
write_files:
  - path: /etc/ssh/sshd_config.d/50-cloud-init.conf
```

```
    content: |
        PubkeyAuthentication yes
        PasswordAuthentication yes
        ChallengeResponseAuthentication yes
        PermitRootLogin yes
  - path: /etc/network/interfaces
    content: |
      auto lo
      iface lo inet loopback
      auto eth0
      iface eth0 inet static
          address 192.168.100.101
          netmask 255.255.255.0
          gateway 192.168.100.1
          dns-nameservers 8.8.8.8
runcmd:
  - resize2fs /dev/vda
  - /sbin/service networking restart

EOF
}
```

- **data "template_file" "cloud_init_vm1"** : Génère un fichier **cloud-init** pour configurer VM1.
  - **cloud-config** : Spécifie la configuration réseau, les utilisateurs SSH, et l'authentification SSH.
  - **write_files** : Ajoute des fichiers de configuration pour le SSH et le réseau.
  - **runcmd** : Exécute des commandes au démarrage (ajustement de la taille du disque et redémarrage du service réseau).

Création du disque cloud-init pour VM1

```
resource "libvirt_cloudinit_disk" "commoninit_vm1" {
  name           = "commoninit_vm1.iso"
  pool           = "default"
  user_data      = data.template_file.cloud_init_vm1.rendered
}
```

- **libvirt_cloudinit_disk "commoninit_vm1"** : Crée un disque cloud-init ISO pour VM1 à partir du fichier template généré précédemment.

Répétition pour VM2

Le même processus est répété pour **VM2**, en changeant les adresses IP, le FQDN, et d'autres détails spécifiques à la machine.

---

5. Création des volumes des VMs

```
resource "libvirt_volume" "vm1_volume" {
  name           = "vm1.qcow2"
  base_volume_id = libvirt_volume.alpine_cloud.id
  format         = "qcow2"
  pool           = libvirt_pool.new_pool.name
  size           = 409715200
}
```

- **libvirt_volume "vm1_volume"** : Crée un volume de disque de 409715200 octets pour VM1, basé sur l'image Alpine téléchargée.
- Le même processus est appliqué pour **VM2**.

---

6. Déploiement des machines virtuelles (VMs)

VM1

```
resource "libvirt_domain" "vm1" {
  name   = "vm1"
  memory = 512
  vcpu   = 1
  network_interface {
    network_id = libvirt_network.lan1.id
    hostname   = "vm1"
    mac        = "52:54:00:6b:3c:58"
  }
  disk {
    volume_id = libvirt_volume.vm1_volume.id
  }
  cloudinit = libvirt_cloudinit_disk.commoninit_vm1.id
  console {
    type        = "pty"
    target_port = "0"
    target_type = "serial"
  }
  graphics {
    type = "spice"
  }
}
```

- **libvirt_domain "vm1"** : Crée une machine virtuelle (VM1) avec 512 Mo de RAM et 1 CPU.
  - **network_interface** : Connecte la VM au réseau `lan1` avec un nom d'hôte `vm1` et une adresse MAC définie.
  - **disk** : Associe le volume de disque `vm1_volume`.
  - **cloudinit** : Associe l'ISO cloud-init `commoninit_vm1.iso` pour configurer VM1.
  - **graphics** : Configure un affichage graphique de type Spice.
  - La même configuration est répliquée pour **VM2**.

Résumé :

Le fichier Terraform :

- Crée un réseau virtuel et un pool de stockage pour les VMs.
- Télécharge une image Alpine Cloud et l'utilise comme base pour les disques des VMs.
- Crée deux machines virtuelles avec des configurations réseau statiques via **cloud-init**.
- Utilise **libvirt** pour gérer les ressources de virtualisation (réseau, disques, et VMs).

Étapes :

- Créez ce fichier `main.tf` dans le dossier de votre projet.
- Initialisez le projet OpenTofu :

```
tofu init
```

- Appliquez la configuration pour créer les VMs :

```
tofu plan && tofu apply
```

Cela créera deux machines virtuelles (VM1 et VM2) basées sur l'image Alpine Linux.

Étape 3 : Configuration d'Ansible pour gérer les machines

1. **Configurer l'inventaire Ansible** : Créez un fichier `hosts` dans votre répertoire de projet :

```
[web_servers]
vm1 ansible_host=192.168.122.101 ansible_user=ares
ansible_ssh_private_key_file=/path/to/private_key
vm2 ansible_host=192.168.122.102 ansible_user=ares
ansible_ssh_private_key_file=/path/to/private_key
```

2. **Tester la connexion entre Ansible et les machines** : Testez la connexion avec la commande suivante :

```
ansible all -m ping -i hosts
```

```
[WARNING]: Platform linux on host vm2 is using the discovered Python
interpreter at /usr/bin/python, but future installation of another Python
interpreter
could change this. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discove
ry.html for more information.
vm2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

```
[WARNING]: Platform linux on host vm1 is using the discovered Python
interpreter at /usr/bin/python, but future installation of another Python
interpreter
could change this. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discove
ry.html for more information.
vm1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Étape 4 : Automatisation de la configuration avec Ansible

Rappels:

Le playbook Ansible est un fichier YAML qui définit un ensemble de tâches à exécuter sur un ou plusieurs serveurs (ici, les serveurs web). Il utilise le module Ansible pour gérer l'installation de paquets et est conçu pour fonctionner sur des serveurs Linux basés sur **Alpine Linux**, car il utilise le gestionnaire de paquets **apk**.

1. **Créer un playbook pour installer les outils réseau** : Créez un fichier `install_network_tools.yml` :

```yaml
---
- hosts: web_servers
  become: yes
  tasks:
    - name: Installer net-tools
      apk:
        name: net-tools
        state: present
```

Explication des différentes sections :

1. **---** :
   - Le triple tiret (`---`) est utilisé pour indiquer le début d'un document YAML. Il n'est pas strictement nécessaire dans tous les cas, mais il est généralement utilisé comme convention pour marquer le début du playbook.
2. **hosts: web_servers** :
   - Ce champ définit sur quels hôtes (ou groupes d'hôtes) les tâches du playbook seront exécutées. Ici, le groupe est nommé `web_servers`.
   - **`web_servers`** est probablement un groupe d'hôtes défini dans le fichier d'inventaire d'Ansible, qui pourrait contenir plusieurs serveurs qui sont utilisés pour héberger des applications web.
3. **become: yes** :

- Cela signifie que les tâches seront exécutées avec des privilèges **administratifs** (root). En Ansible, **become** permet d'élever les privilèges (similaire à l'utilisation de `sudo` sur un terminal Linux).
- `become: yes` indique qu'Ansible utilisera le compte courant, mais avec des privilèges root pour les tâches définies.

4. **tasks:** :
   - Cette section contient une liste de **tâches**. Une tâche représente une action spécifique à exécuter sur les hôtes spécifiés. Dans ce cas, il n'y a qu'une seule tâche, mais vous pouvez en avoir plusieurs dans un playbook.
5. **name: Installer net-tools** :
   - Chaque tâche dans un playbook peut avoir un nom. Le nom est une description lisible qui sera affichée lors de l'exécution du playbook. Ici, le nom est **"Installer net-tools"**, ce qui indique que la tâche va installer le paquet `net-tools` sur les hôtes ciblés.
6. **apk:** :
   - `apk` est le module utilisé pour gérer les paquets sur les systèmes Alpine Linux (le gestionnaire de paquets d'Alpine est `apk`).
   - Ce module permet d'installer, de mettre à jour ou de supprimer des paquets sur les hôtes basés sur Alpine Linux.
7. **name: net-tools** :
   - Cela spécifie le nom du paquet à installer, ici `net-tools`.
   - `net-tools` est un ensemble d'outils réseau tels que `ifconfig`, `netstat`, etc., qui sont souvent utilisés pour configurer et dépanner les interfaces réseau.
8. **state: present** :
   - Cela spécifie l'état dans lequel Ansible doit s'assurer que le paquet se trouve.
   - `present` signifie qu'Ansible va s'assurer que le paquet `net-tools` est installé. Si le paquet est déjà installé, Ansible ne fera rien. Si le paquet n'est pas installé, il sera installé.

Exécutez le playbook :

```
ansible-playbook -i hosts install_network_tools.yml
```

Fonctionnement :

1. Le playbook va cibler les hôtes appartenant au groupe `web_servers` dans votre inventaire Ansible.
2. Ensuite, il va exécuter toutes les tâches avec des privilèges administratifs, car `become: yes` est défini.
3. Il va vérifier si le paquet `net-tools` est installé sur ces serveurs. Si ce n'est pas le cas, Ansible l'installera à l'aide du gestionnaire de paquets `apk` propre à Alpine Linux.

2. **Configurer une adresse IP statique avec Ansible** : Créez un fichier `configure_network.yml` :

```
---
- hosts: web_servers
  become: yes
  tasks:
    - name: Configurer une adresse IP statique
      template:
        src: netplan.j2
        dest: /etc/network/interfaces

    - name: Appliquer la configuration réseau
      command: ifup eth0
```

Créez un template `netplan.j2` :

```
auto ens3
iface ens3 inet static
  address {{ ansible_host }}
  netmask 255.255.255.0
  gateway 192.168.100.1
  dns-nameservers 4.2.2.2
```

Exécutez le playbook :

```
ansible-playbook -i hosts configure_network.yml
```

3. Vérification de la Configuration Réseau:

Créez un fichier `check_config.yml` :

```
---
- name: Vérification de la configuration réseau
  hosts: all  # Vous pouvez cibler un groupe d'hôtes spécifiques comme
'web_servers'
  become: yes  # Pour exécuter les commandes avec des privilèges
administratifs
  tasks:
    - name: Afficher les interfaces réseau
      command: ip link show
      register: result_interfaces

    - name: Afficher la sortie des interfaces réseau
      debug:
        var: result_interfaces.stdout

    - name: Afficher les adresses IP des interfaces
      command: ip addr show
      register: result_ip_addr

    - name: Afficher la sortie des adresses IP
      debug:
        var: result_ip_addr.stdout
```

```
    - name: Afficher la table de routage
      command: ip route show
      register: result_routes

    - name: Afficher la sortie de la table de routage
      debug:
        var: result_routes.stdout

    - name: Vérifier l'état des connexions réseau actives
      command: netstat -tunlp
      register: result_netstat

    - name: Afficher la sortie de netstat
      debug:
        var: result_netstat.stdout

    - name: Vérifier si une interface spécifique (par ex. eth0) est active
      command: ip link show eth0
      register: result_eth0

    - name: Afficher l'état de l'interface eth0
      debug:
        var: result_eth0.stdout
```

Exécutez le playbook :

```
ansible-playbook -i hosts check_config.yml
```

Explication du Playbook :

1. **Définition des hôtes** :
   - `hosts: all` : Le playbook va s'exécuter sur tous les hôtes de l'inventaire. Vous pouvez spécifier un groupe particulier comme `web_servers` si vous ne voulez pas que cela s'exécute partout.
   - `become: yes` : Les tâches seront exécutées avec des privilèges administratifs (nécessaire pour certaines commandes réseau).
2. **Tâches principales** :
   - **Afficher les interfaces réseau** :
     - Utilise la commande `ip link show` pour lister toutes les interfaces réseau disponibles sur le système, qu'elles soient actives ou non.
     - Résultat enregistré dans `result_interfaces` et affiché via une tâche `debug`.
   - **Afficher les adresses IP des interfaces** :
     - Utilise la commande `ip addr show` pour afficher les adresses IP attribuées à chaque interface réseau.
     - Résultat enregistré dans `result_ip_addr` et affiché avec `debug`.
   - **Afficher la table de routage** :

    - Utilise la commande `ip route show` pour afficher la table de routage du système.
    - Cela permet de voir comment les paquets réseau sont routés à travers le réseau.
    - Résultat enregistré dans `result_routes` et affiché.
- **Vérifier l'état des connexions réseau actives** :
    - Utilise la commande `netstat -tunlp` pour afficher toutes les connexions actives (TCP et UDP), ainsi que les ports ouverts et les programmes qui les écoutent.
    - Résultat enregistré dans `result_netstat` et affiché avec `debug`.
- **Vérifier l'état d'une interface spécifique (eth0)** :
    - Si vous souhaitez vérifier l'état d'une interface particulière (par exemple, `eth0`), cette tâche utilise la commande `ip link show eth0`.
    - Résultat enregistré dans `result_eth0` et affiché.

Pour vérifier si une machine accède à Internet à partir d'un playbook Ansible, vous pouvez utiliser la commande `ping` ou essayer d'accéder à un site web (comme Google) via une requête HTTP avec `curl` ou `wget`. Voici un exemple de playbook qui effectue cette vérification.

4. Vérification de la connectivité Internet

Créez un fichier `check_access.yml` :

```
---
- name: Vérification de la connectivité Internet
  hosts: all
  become: yes
  tasks:
    - name: Vérifier la connectivité en pingant une IP externe (Google DNS)
      command: ping -c 4 8.8.8.8
      register: result_ping
      ignore_errors: yes

    - name: Afficher le résultat du test de ping
      debug:
        var: result_ping.stdout

    - name: Vérifier la connectivité en accédant à un site web via HTTP
(Google)
      command: curl -s https://www.google.com
      register: result_http
      ignore_errors: yes

    - name: Afficher le résultat du test HTTP
      debug:
        var: result_http.stdout

    - name: Vérifier si la machine a accès à Internet
      fail:
        msg: "La machine n'a pas accès à Internet."
```

```
      when: result_ping.rc != 0 and result_http.rc != 0
```

Exécutez le playbook :

```
ansible-playbook -i hosts check_config.yml
```

Explication du Playbook :

1. **Vérification de la connectivité avec `ping`** :
   - La première tâche utilise la commande `ping` pour tester la connectivité en envoyant 4 paquets ICMP à l'adresse IP de Google DNS (`8.8.8.8`).
   - Le résultat de la commande est enregistré dans la variable `result_ping`.
   - Si `ping` échoue, la tâche ne s'arrêtera pas immédiatement (`ignore_errors: yes`), permettant au playbook de continuer à la tâche suivante.
2. **Affichage du résultat du `ping`** :
   - Cette tâche affiche le résultat du test `ping` (contenu de `result_ping.stdout`), ce qui permet de voir les détails du test (temps de réponse, paquets perdus, etc.).
3. **Vérification de la connectivité HTTP avec `curl`** :
   - La commande `curl` est utilisée pour vérifier si un site web (ici, `https://www.google.com`) est accessible. Si la machine a accès à Internet, `curl` pourra récupérer la page web.
   - Le résultat de la commande est enregistré dans `result_http`.
4. **Affichage du résultat de `curl`** :
   - Cette tâche affiche le contenu de la réponse de `curl` (contenu de `result_http.stdout`), ce qui permet de vérifier si la page a été correctement récupérée.
5. **Vérification finale : Accès à Internet** :
   - La dernière tâche vérifie si l'accès à Internet est fonctionnel en testant les codes de retour (`rc`) des deux tests précédents (`ping` et `curl`).
   - Si les deux tests échouent (i.e., `result_ping.rc != 0` et `result_http.rc != 0`), cela indique que la machine n'a pas accès à Internet, et la tâche `fail` s'exécutera pour signaler une erreur.
   - Si au moins un des tests réussit, le playbook continuera sans erreur.

Étape 5 : Questions de réflexion et débriefing

- Pourquoi automatiser la configuration réseau avec Ansible ? Developper votre réponse