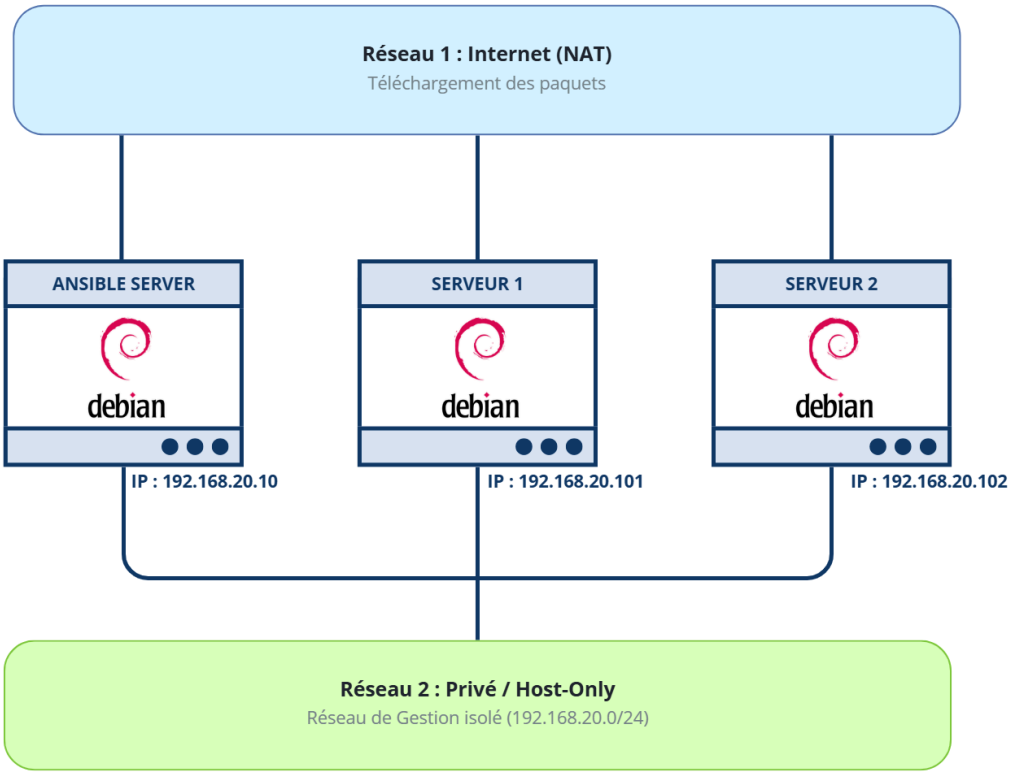
SOMMAIRE

I. Architecture et Prérequis Réseau

Pour mettre en place l'environnement Ansible, nous avons déployé trois machines virtuelles Debian. Afin de garantir à la fois l'accès aux mises à jour et une communication interne stable, nous avons configuré deux interfaces réseau distinctes sur chaque VM :

- Adaptateur 1 (NAT) : Permet l'accès à Internet (nécessaire pour le téléchargement des paquets ansible, sudo, etc.).

- Adaptateur 2 (Réseau Privé Hôte / Host-Only) : Dédié à la communication interne entre le Ansible Server Ansible et les nœuds gérés, assurant une isolation et des adresses IP fixes."



Configuration de l'adressage statique sur le Ansible Server. Comme illustré ci-dessus, nous avons fixé l'adresse IP 192.168.20.10 sur l'interface enp0s8 du nœud de gestion via le fichier /etc/network/interfaces. Une configuration similaire a été appliquée sur les nœuds clients avec les IPs .101 et .102.

VM 1 (Ansible Server) :
IP : 192.168.20.10

VM 2 (Serveur 1) :
IP : 192.168.20.101

VM 3 (Serveur 2) :
IP : 192.168.20.102

Partie A : Installation et Configuration du Nœud de Gestion

L'installation d'Ansible a été réalisée exclusivement sur la machine 'Ansible Serveur' via les dépôts officiels de Debian. La commande **ansible --version** nous permet de valider que l'installation s'est correctement déroulée et que le cœur du logiciel est fonctionnel.

Ouvre un terminal sur la machine 'Ansible Serveur (IP 192.168.20.10) et lance ces commandes pour installer le logiciel :

```
apt update
apt install ansible -y
```

Une fois terminé, vérifie que c'est bon avec :

```
ansible --version
```

```
root@debianVM:~# ansible --version
ansible [core 2.14.18]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modul
es']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.11.2 (main, Aug 26 2024, 07:20:54) [GCC 12.2.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

Cette commande valide l'installation d'Ansible et confirme l'utilisation de notre fichier de configuration personnalisé, garantissant ainsi l'application des paramètres définis précédemment.

2. Configuration d'Ansible (Sur Ansible Serveur)

Nous créons d'abord le répertoire de configuration /etc/ansible afin de centraliser les paramètres nécessaires au fonctionnement du nœud de gestion.

```
sudo mkdir -p /etc/ansible
```

Ensuite, nous générons et éditons le fichier ansible.cfg pour y définir les règles de comportement par défaut de l'application.

```
sudo nano /etc/ansible/ansible.cfg
```

Nous insérons les directives permettant de désactiver la vérification des clés hôtes et de spécifier l'utilisation de l'interpréteur Python 3.

```
[defaults]
# Désactive la vérification des clés SSH (évite de taper "yes" à chaque connexion)
host_key_checking=False
# Définit l'interpréteur Python par défaut
interpreter_python=/usr/bin/python3
```

3. L'Inventaire (Sur VM 1)

Nous éditons le fichier d'inventaire /etc/ansible/hosts pour y référencer les machines clientes que notre serveur de gestion devra administrer.

Ouvre le fichier d'inventaire :

```
sudo nano /etc/ansible/hosts
```

Nous définissons un groupe [servers] associant des alias pratiques aux adresses IP respectives de nos deux machines virtuelles cibles.

```
[servers]
serv1 ansible_ssh_host=192.168.20.101
serv2 ansible_ssh_host=192.168.20.102
```

Enfin, nous exécutons une commande de listage pour confirmer qu'Ansible détecte correctement les deux serveurs déclarés dans notre inventaire.

```
ansible all --list-hosts
```

Le résultat de cette commande confirme la bonne prise en compte du fichier d'inventaire, listant correctement nos deux serveurs cibles serv1 et serv2.

```
root@debianVM:~# ansible all --list-hosts
  hosts (2):
    serv1
    serv2
```

4. Configuration SSH (La partie critique !)

Nous devons d'abord autoriser la connexion de l'utilisateur root sur les machines clientes pour permettre l'administration système par Ansible.

Nous éditons le fichier de configuration du démon SSH sur chaque client pour modifier la directive interdisant le login root.

```
sudo nano /etc/ssh/sshd_config
# Modifier la ligne : PermitRootLogin yes
```

Cette modification du fichier de configuration SSH autorise la connexion directe de l'utilisateur root, prérequis indispensable pour l'administration distante par Ansible

Ensuite, nous redémarrons le service SSH pour appliquer immédiatement ce changement de politique de sécurité sur les deux machines

```
sudo service ssh restart
```

Sur le Ansible Server, nous générons une paire de clés cryptographiques RSA qui servira de jeton d'authentification pour les connexions futures.

```
ssh-keygen
```

Nous déployons ensuite notre clé publique sur les serveurs distants, ce qui autorisera la connexion sans saisie de mot de passe.

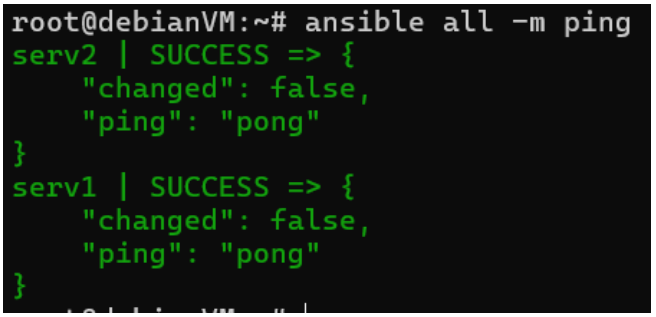
```
ssh-copy-id root@192.168.20.101
ssh-copy-id root@192.168.20.102
```

Le Test Final (Le Ping Ansible)

Pour finir, nous utilisons le module ping pour confirmer qu'Ansible parvient à contacter l'ensemble du parc sans erreur d'authentification.

```
ansible all -m ping
```

Le retour succès du module ping valide définitivement la connectivité SSH sans mot de passe entre le Ansible Server et les serveurs administrés.



Partie B : Ansible Playbook

Un playbook est un fichier au format YAML contenant une liste ordonnée de tâches à automatiser sur les machines cibles.

1. Installer des paquets (ntpdate et nmap)

Nous élaborons un premier playbook nommé ntp-nmap.yml ayant pour objectif l'installation automatisée des paquets logiciels ntpdate et nmap sur l'ensemble du parc.

```
nano ntp-nmap.yml
```

Le contenu du fichier définit la cible (hosts: all), l'élévation de privilèges (become: yes) et l'état désiré des paquets via le module apt.

```
---
- hosts: all
  become: yes
  tasks:
    - name: Install packages
      apt:
        name:
          - ntpdate
          - nmap
        state: latest
        cache_valid_time: 3600
```

Nous exécutons ensuite ce playbook qui sollicite le module de gestion de paquets pour assurer la présence et la mise à jour des logiciels requis.

```
ansible-playbook update.yml
```

L'exécution du playbook affiche un état 'changed', validant l'installation effective des paquets ntpdate et nmap sur nos deux serveurs cibles.

```
root@debianVM:~# ansible-playbook ntp-nmap.yml

PLAY [all] *********************************************************************
****

TASK [Gathering Facts] *********************************************************
****
ok: [serv2]
ok: [serv1]

TASK [Install packages] ********************************************************
****
changed: [serv2]
changed: [serv1]

PLAY RECAP *********************************************************************
****
serv1                      : ok=2    changed=1    unreachable=0    failed=0
   skipped=0    rescued=0    ignored=0
serv2                      : ok=2    changed=1    unreachable=0    failed=0
   skipped=0    rescued=0    ignored=0
```

2. Exécution d'une commande Shell

Ce deuxième exercice exploite le module shell pour exécuter des commandes système natives, comme la redirection de flux vers des fichiers temporaires

```
nano shell-test.yml
```

Nous définissons deux tâches : la première stocke la date courante dans un fichier, la seconde affiche un message directement sur la console système.

```
---
- hosts: all
  become: yes
  tasks:
    - name: Echo the Date to a tmp file
      shell: echo "$(date)" > /tmp/date

    - name: Echo String to a tmp file
      shell: echo "Test Ansible RT" > /dev/tty0
```

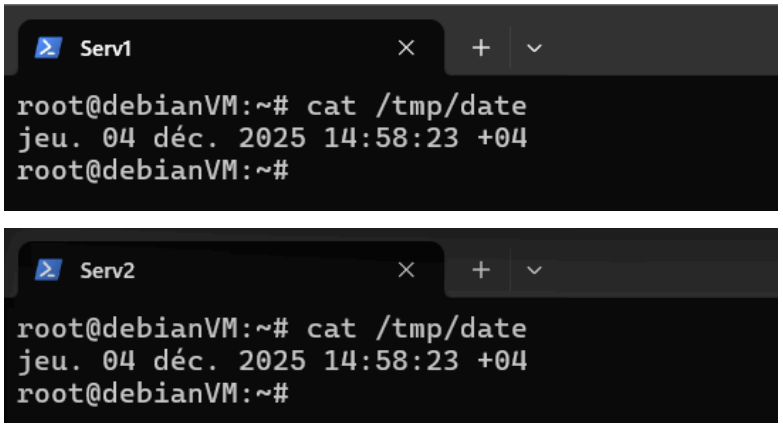
L'exécution de ce script génère le fichier daté sur les cibles et prouve la capacité d'Ansible à interagir avec le système d'exploitation.

```
ansible-playbook shell-test.yml
```

Les statuts 'changed' valident l'exécution des commandes Shell, confirmant la création du fichier daté et l'affichage du message sur les consoles distantes.

```
root@debianVM:~# ansible-playbook shell-test.yml

PLAY [all] ********************************************************************
****

TASK [Gathering Facts] ********************************************************
****
ok: [serv2]
ok: [serv1]

TASK [Echo the Date to a tmp file] ********************************************
****
changed: [serv2]
changed: [serv1]

TASK [Echo String to a tmp file] **********************************************
****
changed: [serv2]
changed: [serv1]

PLAY RECAP ********************************************************************
****
serv1                      : ok=3    changed=2    unreachable=0    failed=0
   skipped=0    rescued=0    ignored=0
serv2                      : ok=3    changed=2    unreachable=0    failed=0
   skipped=0    rescued=0    ignored=0
```

Nous vérifions manuellement la présence du fichier horodaté sur les clients, confirmant que le playbook a bien exécuté les commandes shell demandées.



3. Les Variables

Nous créons le fichier linux-scan.yml afin d'introduire la notion de variables, ce qui rend nos playbooks dynamiques et réutilisables pour différentes cibles.

```
nano linux-scan.yml
```

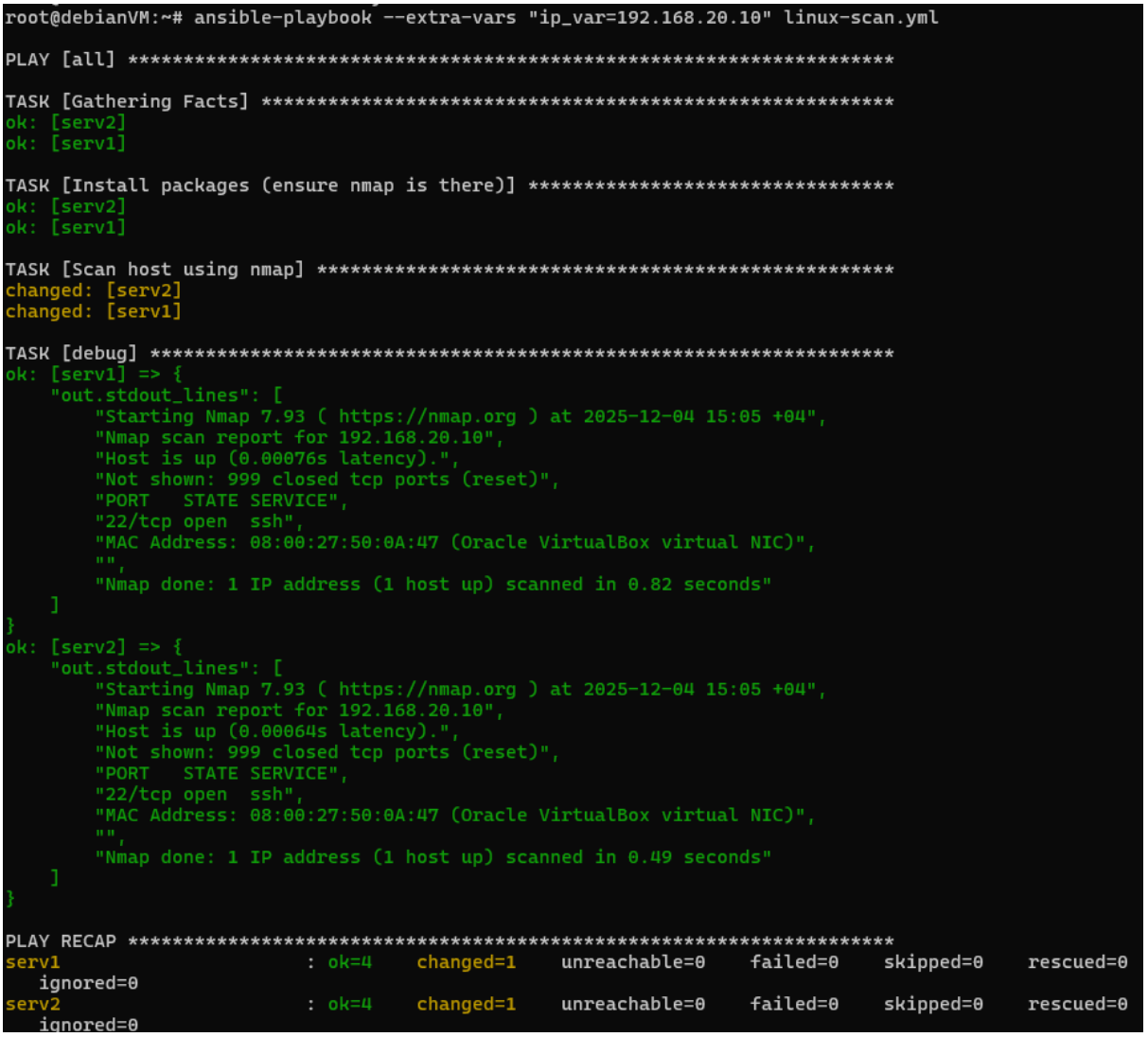
Ce script assure d'abord l'installation de nmap, puis exécute un scan réseau vers une cible définie par la variable {{ ip_var }}.

```
---
- hosts: all
  become: yes
  tasks:
    - name: Install packages (ensure nmap is there)
      apt:
        name:
          - nmap
        state: latest
        cache_valid_time: 3600

    - name: Scan host using nmap
      shell: nmap "{{ ip_var }}"
      register: out

    - debug: var=out.stdout_lines
```

Nous exécutons le playbook en injectant l'adresse IP du Ansible Server via l'argument --extra-vars, déclenchant ainsi le scan réseau depuis les clients.

```
ansible-playbook --extra-vars "ip_var=192.168.20.10"
linux-scan.yml
```

Cette exécution avec une variable dynamique prouve que le scan fonctionne, le module debug confirmant l'ouverture du port SSH sur la cible



4. Mise à jour du système (Update & Upgrade)

Nous rédigeons le fichier update.yml pour automatiser la maintenance du système, combinant la mise à jour des dépôts et la mise à niveau des paquets.

```
nano update.yml
```

Le paramètre cache_valid_time optimise l'exécution en évitant de rafraîchir le cache apt si la dernière vérification date de moins de 24 heures

```
---
- hosts: all
  become: yes
  tasks:
    - name: Update and upgrade
      apt:
        upgrade: yes
        update_cache: yes
        cache_valid_time: 86400
```

Nous lançons la commande suivante pour appliquer les mises à jour de sécurité et logicielles sur l'ensemble du parc de machines simultanément.

```
ansible-playbook update.yml
```

5. Tâches parallèles (Async)

Ce dernier exercice illustre la gestion des tâches longues via le mode asynchrone, permettant de lancer des processus sans bloquer l'exécution du playbook.

```
nano async.yml
```

Nous utilisons une boucle with_items pour simuler des délais variables, associée à async pour définir un temps limite d'exécution global.

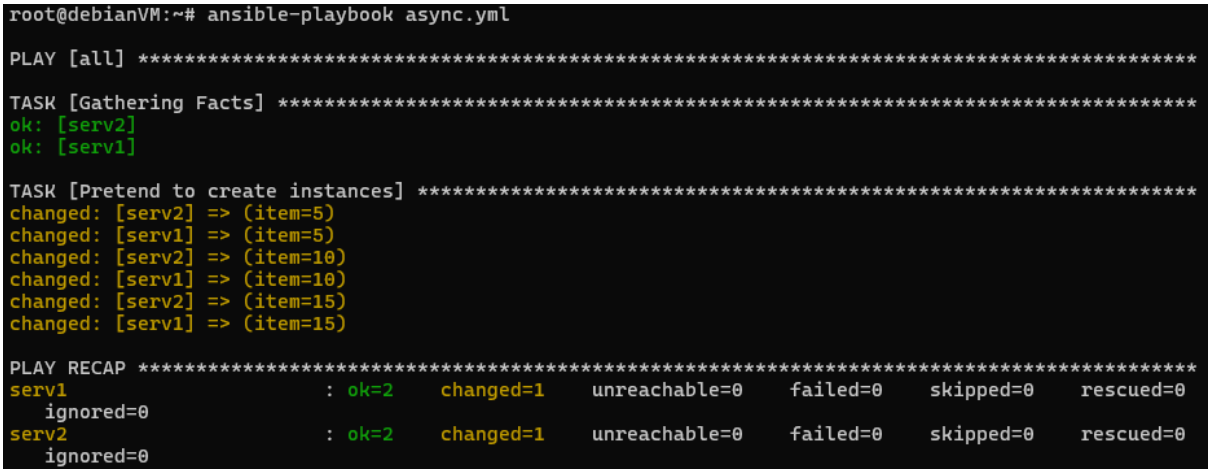
```
---
- hosts: all
  become: yes
  tasks:
    - name: Pretend to create instances
      shell: "echo `date` >> /tmp/date; sleep {{ item }}"
      with_items:
        - 5
        - 10
        - 15
      register: _create_instances
```

```
        async: 30
        poll: 0
```

La directive poll: 0 active le mode « tir et oubli », où Ansible passe immédiatement à la tâche suivante sans attendre la fin des processus.

```
ansible-playbook async.yml
```

Le mode asynchrone permet de lancer les tâches en parallèle, libérant le Ansible Server immédiatement sans attendre la fin des délais de simulation.



Conclusion

Ce travail pratique nous a permis de déployer et de configurer avec succès une infrastructure d'automatisation basée sur Ansible. À travers la mise en place d'un nœud de gestion et de deux serveurs cibles, nous avons validé les concepts fondamentaux de l'outil : une architecture « sans agent » reposant sur SSH et une gestion centralisée via un fichier d'inventaire.

La réalisation des différents Playbooks nous a permis de maîtriser plusieurs aspects clés de l'administration système automatisée :

- L'installation et la maintenance de paquets via le module apt.
- L'interaction directe avec le système via le module shell.
- L'utilisation de variables pour rendre les scripts dynamiques et réutilisables.
- L'optimisation des performances grâce à la gestion des tâches asynchrones.

En résumé, ce TP a démontré la capacité d'Ansible à garantir la cohérence des configurations et à accélérer le déploiement sur un parc de machines, confirmant son rôle central dans les pratiques DevOps modernes.