TP 5 : Ansible (version locale avec Docker)

Prérequis pour le TP

Installer Ansible

- Installer Ansible avec pip :

  ```
  python3 -m pip install --user ansible --break-system-packages
  ```

Ajouter Ansible au PATH

- Trouver où sont installés vos packages Python :

  ```
  export USER_PYBASE=$(python3 -m site --user-base)
  ```

- Ajouter le répertoire binaire à votre variable PATH :

  ```
  export PATH="$USER_PYBASE/bin:$PATH"
  ```

  Note : Vous pouvez écrire ces commandes dans `~/.bashrc` pour qu'elles soient exécutées automatiquement au démarrage d'un terminal bash.

Tester votre installation

```
ansible --version
```

1 Création d'un mini-cluster local avec Docker

Pour ce TP, nous allons simuler plusieurs machines distantes (VMs) à l'aide de conteneurs Docker. Chaque conteneur jouera le rôle d'un noeud administré par Ansible.

1.1 Créer une image Docker compatible Ansible

Créez un fichier `Dockerfile` :

```
FROM ubuntu:22.04

# Install SSH and basic tools
RUN apt update && apt install -y openssh-server python3 sudo && \
    mkdir /var/run/sshd && \
    useradd -m -s /bin/bash ansible && \
    echo "ansible:ansible" | chpasswd && \
    adduser ansible sudo && \
    echo "ansible ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

# Enable SSH password login
RUN sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config && \
    sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

EXPOSE 22
CMD ["/usr/sbin/sshd","-D"]
```

Construisez l'image avec :

```
docker build -t ubuntu-ansible .
```

1.2 Déployer un premier conteneur

Créez un fichier `docker-compose.yaml` pour instancier le conteneur :

```yaml
version: "3"
services:
  node1:
    image: ubuntu-ansible
    container_name: ansible-node1
    hostname: node1
    ports:
      - "2221:22"
```

puis lancez le avec

```
docker compose up -d
```

1.3 Autoriser les connexions SSH

Ajoutez l'empreinte SSH localement :

```
ssh-keyscan -p 2221 127.0.0.1 >> ~/.ssh/known_hosts
```

2 La Voting App

Nous allons déployer une application distribuée composée de plusieurs services. L'objectif est de la déployer localement, d'abord sur une seule machine, puis sur plusieurs.

2.1 Description de l'application

- **vote :** application Python/Flask pour soumettre des votes.
- **result :** application Node.js affichant les résultats.
- **worker :** application .NET traitant les votes.
- **redis :** base NoSQL stockant temporairement les votes.
- **db :** base PostgreSQL stockant les résultats finaux.
- **nginx :** load balancer entre les services web.

3 Premiers pas avec Ansible

3.1 Fichier d'inventaire

Créez un fichier `inventory.ini`, dans lequel nous allons créer un groupe d'hôtes contenant notre conteneur :

```
[voteapp]
localhost ansible_host=127.0.0.1 ansible_port=2221 ansible_user=ansible ansible_password=ansible
```

Notez que le port donné à Ansible pour contacter notre hôte est le port 2221, associé au port 22 (port pour SSH) en interne du conteneur.
Testez la connexion avec un ping :

```
ansible all -i inventory.ini -m ping
```

3.2 Installer Docker sur les conteneurs avec Ansible

L'objectif de cette partie est de créer un playbook Ansible capable d'installer Docker sur l'ensemble de nos noeuds (les conteneurs). Nous allons décomposer la tâche étape par étape afin de comprendre la logique d'un playbook typique.

Préparation Dans votre répertoire de travail, créez un fichier nommé `install-docker.yaml`. Un playbook Ansible est un fichier YAML décrivant une série de tâches exécutées sur des hôtes définis.
Chaque tâche représente une action concrète : installation d'un paquet, création d'un fichier, exécution d'une commande, etc.

3.2.1 Structure du playbook

Un playbook suit généralement la structure suivante :

- **name** : un titre lisible qui décrit le but global du playbook.
- **hosts** : le ou les groupes de machines sur lesquels les tâches seront exécutées (ici, `all`).
- **become** : indique que les tâches nécessitent des privilèges administrateur (`sudo`).
- **tasks** : la liste des actions à exécuter, dans l'ordre.

Voici le squelette de départ :

```yaml
- name: Install Docker on nodes
  hosts: all
  become: yes
  tasks:
    # tasks go here
```

3.2.2 Mise à jour des paquets

Première étape : mettre à jour les index de paquets avant toute installation.

```yaml
    - name: Update packages
      apt:
        update_cache: yes
```

Cela permet à Ansible d'utiliser les versions les plus récentes des paquets disponibles.

3.2.3 Installation des dépendances

Docker nécessite plusieurs outils systèmes (curl, gnupg, etc.). On installe ces paquets en une seule tâche :

```yaml
    - name: Install dependencies
      apt:
        name:
          - ca-certificates
          - curl
          - gnupg
          - lsb-release
        state: present
```

3.2.4 Ajout de la clé GPG Docker

Pour s'assurer de l'authenticité du dépôt Docker, on doit ajouter sa clé GPG officielle. On utilise ici le module `shell` car cette étape nécessite une commande complexe.

```yaml
    - name: Add Docker GPG key
      ansible.builtin.shell: |
        install -m 0755 -d /etc/apt/keyrings
        curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
          gpg --dearmor -o /etc/apt/keyrings/docker.gpg
      args:
        creates: /etc/apt/keyrings/docker.gpg
```

Le paramètre `creates` garantit l'idempotence : la commande ne sera exécutée que si le fichier n'existe pas déjà.

3.2.5 Ajout du dépôt Docker officiel

On configure maintenant la source des paquets Docker. Le module `copy` écrit directement le fichier dans `/etc/apt/sources.list.d/`.

```yaml
    - name: Add Docker repository
      copy:
        dest: /etc/apt/sources.list.d/docker.list
        content: |
          deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
              jammy stable
```

3.2.6 Installation de Docker

Une fois le dépôt configuré, on installe les paquets Docker. Notez la présence du module `apt` avec `update_cache: yes` pour rafraîchir la liste des paquets.

```
    - name: Install Docker
      apt:
        name:
          - docker-ce
          - docker-ce-cli
          - containerd.io
          - docker-compose-plugin
        state: present
        update_cache: yes
```

3.2.7 Activation et démarrage du service

Enfin, on démarre et on active le service Docker pour qu'il se lance automatiquement au démarrage.

```
    - name: Enable and start Docker
      service:
        name: docker
        state: started
        enabled: yes
```

3.3 Exécution du playbook

Lancez le playbook avec :

```
ansible-playbook -i inventory.ini install-docker.yaml
```

Si tout se passe bien, vous devriez voir des lignes marquées `changed` indiquant que Docker a bien été installé. Vérifiez ensuite la version de docker sur les noeuds de votre inventaire :

```
ansible -i inventory.ini -m shell -a "docker --version" all
```

Si l'installation s'est correctement déroulée, votre version Docker installée sur votre noeud devrait apparaitre.

4 Déploiement de la Voting App

4.1 Copier le code de l'application

Après avoir télécharger l'application (depuis Celene), placez le dossier `voting-app` et les sous-dossiers applicatifs sur votre machine hôte. Créez un nouveau playbook Ansible, `deploy-voting-app.yaml`, pour le copier dans un répertoire sur votre noeud.

```
- name: Deploy Voting App
  hosts: voteapp
  become: yes
  tasks:
    - name: Copy app files
      copy:
        src: ./voting-app/
        dest: /opt/voting-app/
```

4.2 Lancer l'application

Nous allons ajouter une nouvelle tâche à notre playbook pour démarrer l'application :

```
    - name: Start containers
      ansible.builtin.shell: |
        cd /opt/voting-app
        docker compose up -d
```

Exécutez le playbook avec la commande :

```
ansible-playbook -i inventory.ini deploy-voting-app.yaml
```

Puis testez :

```
http://localhost:8000  (page de vote)
http://localhost:5050  (resultats)
```

5 Exercices d'ouvertures

Si vous avez réalisé les étapes précédentes avec sérieux, vous avez normalement compris comment fonctionne Ansible (l'inventaire, le playbook, les tâches, les modules). **A vous de jouer maintenant !**

5.1 Déployer PostgreSQL sur un autre noeud

Écrivez un playbook qui installe et configure PostgreSQL sur votre deuxième conteneur.

Vous aurez besoin des modules suivants :

- `apt` pour installer les paquets Debian.
- `service` pour démarrer ou redémarrer les services systemd.
- La collection de modules `postgresql` pour effectuer des actions sur la base PostgreSQL.

En pratique, pour installer PostgreSQL, votre playbook devra :

1) Installer les paquets `postgresql-15` et `python3-psycopg2`. Le second est requis pour utiliser la collection de modules `postgresql`.

2) Démarrer le service `postgresql@15-main`.

3) Créer une base de données nommée `votingapp`.

4) Créer un utilisateur PostgreSQL nommé `votingapp` avec le mot de passe `verysecret`.

5) Donner les permissions à l'utilisateur `votingapp` en utilisant le module `postgresql_privs` (utiliser `type: database` et `privs: all`).

6) Autoriser la connexion à distance pour l'utilisateur `votingapp`, afin que l'application puisse se connecter à la base de données. Utilisez le module `postgresql_pg_hba` : il modifie le fichier de configuration `pg_hba.conf` spécifié via l'attribut `dest`.

- Pour obtenir le chemin de `pg_hba.conf`, utilisez la commande suivante sur la VM :

  ```
  sudo -u postgres psql -c "SHOW hba_file;"
  ```

- Vous devez ajouter des enregistrements du type :

  ```
  CONNEXION_TYPE        DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
  ```

  via `postgresql_pg_hba` en utilisant les attributs `contype`, `databases`, `users`, `address` et `method`. Utilisez `host` pour `contype`, `all` pour `address` et `md5` pour `method`.

7) Configurer PostgreSQL pour écouter sur toutes les interfaces. Le module `postgresql_set` vous permet de mettre à jour un autre fichier de configuration : `postgresql.conf`. Vous n'avez pas besoin de spécifier explicitement le chemin vers ce fichier. Définissez `listen_addresses` à `0.0.0.0`.

8) Enfin, redémarrer le service `postgresql`.

Astuce : lors de l'exécution des tâches utilisant les modules `postgresql`, celles-ci doivent s'exécuter sous l'utilisateur Unix `postgres`. Les autres tâches doivent s'exécuter en tant que `root`.

5.2 Deployer une application frontend-backend sur plusieurs noeuds

Dans cet exercice, vous allez déployer une application simple sur une infrastructure composée de **deux noeuds Docker**, administrés avec Ansible. (Note : l'application est celle présentée dans la demonstration du docker-compose vue en cours). L'application `basic-app` contient deux composants : un **frontend Node.js** (`port 3000`) et un **backend Python/Flask** (`port 5000`). L'objectif est de les faire fonctionner sur **deux noeuds distincts**.

Travail demandé :

1) **Préparer l'infrastructure :** créer deux conteneurs qui joueront le rôle de machines hôtes, comme au début du TP. Ils devront être accessibles en SSH et pouvoir communiquer entre eux.

2) **Configurer Ansible :** compléter l'inventaire pour distinguer les deux noeuds et vérifier la connexion à chacun.

3) **Installer Docker :** réutiliser votre playbook pour installer Docker sur les deux noeuds et vérifier son fonctionnement.

4) **Déployer l'application :** créer un playbook qui copie les fichiers, construit les images et démarre le frontend sur le premier noeud et le backend sur le second. Le déploiement des deux composants doit être réalisé avec Ansible.

5) **Tester :** vérifier que les deux services sont démarrés et que le frontend affiche bien la réponse du backend.

Note sur le réseau : placez les deux conteneurs hôtes sur un même réseau Docker. Le backend doit être joignable depuis le conteneur qui exécute le frontend : ouvrez le port `5000` du backend sur son noeud hôte et configurez la variable d'environnement `BACKEND_URL` du frontend avec une adresse effectivement accessible depuis ce dernier (`localhost` ne convient pas). Ouvrez également le port `3000` du frontend pour le rendre accessible depuis votre machine.
Note sur Docker imbriqué : pour exécuter Docker à l'intérieur des conteneurs hôtes, ceux-ci peuvent nécessiter le mode privilégié (`privileged: true`) ainsi qu'un démarrage explicite du démon `dockerd`, car l'image utilisée ne démarre pas `systemd`.

Résultat attendu : depuis votre machine, l'adresse `http://localhost:3000` doit afficher le message `Bonjour depuis le backend`.