TD - Automatisation d'une stack web avec Ansible

Objectifs

A l'issue de ce TD, vous serez capable de :

- Comprendre la séparation entre provisionnement (Terraform) et configuration (Ansible)
- Configurer une stack web multi-services avec Ansible
- Ecrire des templates Jinja2 pour générer des fichiers de configuration dynamiques
- Utiliser les handlers pour gérer les redémarrages conditionnels
- Structurer un projet Ansible en rôles réutilisables
- Gérer les dépendances entre services
- Utiliser des variables par environnement

Contexte

Vous intégrez l'équipe DevOps d'une plateforme e-commerce en forte croissance. L'infrastructure tourne sur AWS en production, mais l'equipe travaille en local avec Docker pour les environnements de developpement.

Votre mission : automatiser la configuration d'une stack composee de quatre serveurs :

- un reverse proxy Nginx
- une application Flask
- une base de donnees PostgreSQL
- un cache Redis

Terraform a deja ete utilise dans le TD precedent pour provisionner les conteneurs. Vous prenez le relai avec Ansible pour configurer chaque service.

0. Rôles Ansible

Un rôle est une façon de regrouper et d'organiser du contenu Ansible de manière réutilisable.

Sans rôle, un playbook grossit au fil du temps : les tâches, variables et handlers se retrouvent dans un seul fichier qui devient difficile à lire et à maintenir.

Un rôle découpe ce fichier en unités autonomes, une par service ou responsabilité.

Exemple concret :

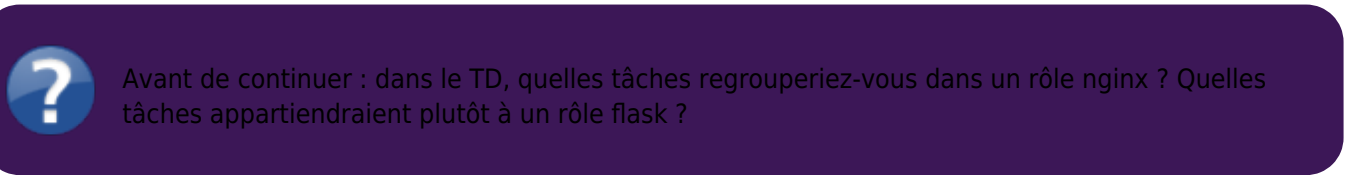
- un rôle `nginx` gère tout ce qui concerne le serveur web
- un rôle `flask` gère tout ce qui concerne l'application
- un rôle `postgresql` gère tout ce qui concerne la base de données

Le playbook principal devient alors une simple liste de rôles à appliquer sur quels hôtes. Il ne contient plus de logique, seulement de l'orchestration.

Avantages concrets :

- Un rôle peut être réutilisé dans plusieurs projets sans recopie
- Les modifications sont isolées : changer le rôle nginx n'impacte pas flask
- Le playbook principal devient lisible en quelques lignes

- Les rôles peuvent être versionnés et partagés via Ansible Galaxy



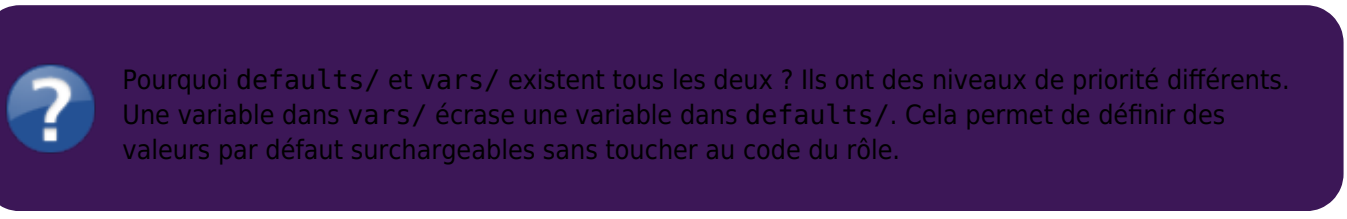
Structure imposée par Ansible pour un rôle

Quand Ansible charge un rôle, il cherche des dossiers et fichiers avec des noms précis.

```
roles/
  nom_du_role/
    tasks/
      main.yml       # point d'entrée obligatoire : liste des tâches
    handlers/
      main.yml       # actions déclenchées par notify
    templates/
      *.j2           # fichiers Jinja2 générés dynamiquement
    files/
      *              # fichiers statiques copiés tels quels
    defaults/
      main.yml       # variables par défaut (priorité basse)
    vars/
      main.yml       # variables du rôle (priorité haute)
    meta/
      main.yml       # dépendances entre rôles
```

Règles importantes :

- Le point d'entrée est toujours `main.yml`, pas `tasks.yml`, pas `install.yml`
- Ansible ne charge que ce qu'il trouve dans ces dossiers aux noms fixes
- Seul `tasks/main.yml` est strictement nécessaire pour qu'un rôle fonctionne
- Les autres dossiers sont optionnels : Ansible ignore ceux qui sont absents



1. Structure du projet

Creez la structure suivante :

```
  td-ansible/
    docker/
      Dockerfile
```

```
    terraform/
      main.tf
    ansible/
      inventory/
        hosts.yml
      group_vars/
        all.yml
        frontend.yml
        backend.yml
        db.yml
        cache.yml
      roles/
        nginx/
          tasks/
            main.yml
          templates/
            nginx.conf.j2
          handlers/
            main.yml
        flask/
          tasks/
            main.yml
          handlers/
            main.yml
        postgresql/
          tasks/
            main.yml
          handlers/
            main.yml
        redis/
          tasks/
            main.yml
          handlers/
            main.yml
      playbook.yml
```

2. Construction de l'image Docker

Construisez l'image manuellement depuis le dossier docker.

Fichier : `docker/Dockerfile`

```
FROM ubuntu:22.04

RUN apt-get update -qq && \
    apt-get install -y -qq \
      openssh-server \
      python3 \
      python3-pip \
      curl \
```

```
    && mkdir -p /run/sshd \
    && echo 'root:root' | chpasswd \
    && sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/'
/etc/ssh/sshd_config

EXPOSE 22

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

```
cd docker
docker build -t ansible-node:local .
cd ..
```

3. Provisionnement avec Terraform

Créez le fichier suivant, puis appliquez la configuration.

Fichier : `terraform/main.tf`

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
  }
}

provider "docker" {}

resource "docker_image" "base" {
  name         = "ansible-node:local"
  keep_locally = true
}

resource "docker_network" "stack_network" {
  name = "stack_network"
}

resource "docker_container" "nginx" {
  name  = "nginx"
  image = docker_image.base.image_id
  networks_advanced {
    name = docker_network.stack_network.name
  }
  ports {
    internal = 22
    external = 2221
```

```
  }
  ports {
    internal = 80
    external = 8090
  }
}

resource "docker_container" "flask" {
  name  = "flask"
  image = docker_image.base.image_id
  networks_advanced {
    name = docker_network.stack_network.name
  }
  ports {
    internal = 22
    external = 2222
  }
}

resource "docker_container" "postgresql" {
  name  = "postgresql"
  image = docker_image.base.image_id
  networks_advanced {
    name = docker_network.stack_network.name
  }
  ports {
    internal = 22
    external = 2223
  }
}

resource "docker_container" "redis" {
  name  = "redis"
  image = docker_image.base.image_id
  networks_advanced {
    name = docker_network.stack_network.name
  }
  ports {
    internal = 22
    external = 2224
  }
}
```

```
cd terraform
terraform init
terraform apply
```

Si vous obtenez une erreur indiquant que le réseau ou des conteneurs existent déjà, supprimez-les avant de relancer :

```
docker rm -f nginx flask postgresql redis
docker network rm stack_network
terraform apply
```

Questions :

- Que fait terraform init ?
- Pourquoi expose-t-on le port 22 de chaque conteneur sur un port différent de la machine hôte ?
- Que se passe-t-il si deux conteneurs exposent le même port ?

4. Inventaire Ansible

L'inventaire indique à Ansible quels sont les hôtes à gérer et comment s'y connecter.

Creez le fichier suivant.

Fichier : ansible/inventory/hosts.yml

```
all:
  vars:
    ansible_user: root
    ansible_password: root
    ansible_ssh_common_args: '-o StrictHostKeyChecking=no'

  children:
    frontend:
      hosts:
        nginx:
          ansible_host: 127.0.0.1
          ansible_port: 2221

    backend:
      hosts:
        flask:
          ansible_host: 127.0.0.1
          ansible_port: 2222

    db:
      hosts:
        postgresql:
          ansible_host: 127.0.0.1
          ansible_port: 2223
```

```
    cache:
      hosts:
        redis:
          ansible_host: 127.0.0.1
          ansible_port: 2224
```

Testez la connexion aux conteneurs :

Installez au préalable sshpass sur votre machine :

```
sudo apt-get install -y sshpass
```

puis :

```
  cd ansible/
  ansible all -i inventory/hosts.yml -m ping
```

Questions :

- Que signifie le parametre StrictHostKeyChecking=no ?
- Pourquoi est-ce acceptable en developpement mais pas en production ?
- Que retourne la commande ping d'Ansible ?

5. Variables par groupe

Les variables permettent de centraliser la configuration et de la rendre reutilisable.

Creez les fichiers suivants.

Fichier : ansible/group_vars/all.yml

```
app_name: ecommerce
app_env: development
app_domain: localhost
```

Fichier : ansible/group_vars/db.yml

```
db_name: ecommerce_db
db_user: app_user
```

```
db_password: changeme
db_port: 5432
```

Fichier : ansible/group_vars/cache.yml

```
redis_port: 6379
redis_maxmemory: 256mb
```

Fichier : ansible/group_vars/backend.yml

```
flask_port: 5000
flask_debug: true
db_host: postgresql
cache_host: redis
```

Fichier : ansible/group_vars/frontend.yml

```
nginx_port: 80
backend_host: flask
backend_port: 5000
```

6. Role PostgreSQL

On commence par la base de donnees car les autres services en dependent.

Creez le fichier suivant.

Fichier : ansible/roles/postgresql/tasks/main.yml

```
---
- name: Installation de PostgreSQL
  apt:
    name:
      - postgresql
      - postgresql-contrib
      - python3-psycopg2
    state: present
    update_cache: yes

- name: Demarrage de PostgreSQL
  service:
    name: postgresql
    state: started
```

```
    enabled: yes

- name: Creation de la base de donnees
  become: yes
  become_user: postgres
  postgresql_db:
    name: "{{ db_name }}"
    state: present

- name: Creation de l'utilisateur applicatif
  become: yes
  become_user: postgres
  postgresql_user:
    name: "{{ db_user }}"
    password: "{{ db_password }}"
    priv: "{{ db_name }}.*:ALL"
    state: present
```

7. Role Redis

Creez le fichier suivant.

Fichier : ansible/roles/redis/tasks/main.yml

```
---
- name: Installation de Redis
  apt:
    name: redis-server
    state: present
    update_cache: yes

- name: Configuration de Redis
  lineinfile:
    path: /etc/redis/redis.conf
    regexp: '^maxmemory '
    line: "maxmemory {{ redis_maxmemory }}"
  notify: restart redis

- name: Demarrage de Redis
  service:
    name: redis-server
    state: started
    enabled: yes
```

Fichier : ansible/roles/redis/handlers/main.yml

```
---
- name: restart redis
```

```
  service:
    name: redis-server
    state: restarted
```

Questions :

- Qu'est-ce qu'un handler dans Ansible ?
- Quand est-il declenche ?
- Quelle est la difference entre notify et un appel direct a service ?

8. Role Flask

Creez le fichier suivant.

Fichier : ansible/roles/flask/tasks/main.yml

```
---
- name: Installation des dependances Python
  apt:
    name:
      - python3
      - python3-pip
      - python3-venv
    state: present
    update_cache: yes

- name: Creation du repertoire applicatif
  file:
    path: /opt/flask
    state: directory
    mode: '0755'

- name: Creation de l'application Flask
  copy:
    dest: /opt/flask/app.py
    content: |
      from flask import Flask, jsonify
      import os
      import redis
      import psycopg2

      app = Flask(__name__)

      @app.route('/health')
      def health():
          return jsonify(status='ok', env=os.environ.get('APP_ENV', 'unknown'))

      if __name__ == '__main__':
```

```
          app.run(host='0.0.0.0', port={{ flask_port }}, debug={{ flask_debug |
lower }})
  notify: restart flask

- name: Installation de Flask et des dependances
  pip:
    name:
      - flask
      - redis
      - psycopg2-binary
    executable: pip3

- name: Creation du service systemd Flask
  copy:
    dest: /etc/systemd/system/flask.service
    content: |
      [Unit]
      Description=Flask Application
      After=network.target

      [Service]
      Environment=APP_ENV={{ app_env }}
      Environment=DB_HOST={{ db_host }}
      Environment=CACHE_HOST={{ cache_host }}
      ExecStart=/usr/bin/python3 /opt/flask/app.py
      Restart=always

      [Install]
      WantedBy=multi-user.target
  notify: restart flask

- name: Demarrage de Flask
  systemd:
    name: flask
    state: started
    enabled: yes
    daemon_reload: yes
```

Fichier : ansible/roles/flask/handlers/main.yml

```
---
- name: restart flask
  systemd:
    name: flask
    state: restarted
    daemon_reload: yes
```

9. Role Nginx

Creez les fichiers suivants.

Fichier : ansible/roles/nginx/tasks/main.yml

```
---
- name: Installation de Nginx
  apt:
    name: nginx
    state: present
    update_cache: yes

- name: Configuration de Nginx
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/sites-available/default
    mode: '0644'
  notify: restart nginx

- name: Demarrage de Nginx
  service:
    name: nginx
    state: started
    enabled: yes
```

Fichier : ansible/roles/nginx/templates/nginx.conf.j2

```
upstream backend {
    server {{ backend_host }}:{{ backend_port }};
}

server {
    listen {{ nginx_port }};
    server_name {{ app_domain }};

    location / {
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /health {
        proxy_pass http://backend/health;
    }
}
```

Fichier : ansible/roles/nginx/handlers/main.yml

```
---
- name: restart nginx
  service:
```

```
    name: nginx
    state: restarted
```

Questions :

- Quelle est la difference entre copy et template dans Ansible ?
- Que fait la directive proxy_pass ?
- Pourquoi utilise-t-on le nom du conteneur plutot que son adresse IP ?

10. Playbook principal

Creez le fichier suivant.

Fichier : ansible/playbook.yml

```
---
- name: Configuration de la base de donnees
  hosts: db
  become: yes
  roles:
    - postgresql

- name: Configuration du cache
  hosts: cache
  become: yes
  roles:
    - redis

- name: Configuration de l'application
  hosts: backend
  become: yes
  roles:
    - flask

- name: Configuration du reverse proxy
  hosts: frontend
  become: yes
  roles:
    - nginx
```

Questions :

- Pourquoi l'ordre des plays dans le playbook est-il important ?
- Que se passe-t-il si on configure Nginx avant Flask ?
- A quoi sert le parametre become ?

11. Execution et verification

Executez le playbook :

```
cd ansible/
ansible-playbook -i inventory/hosts.yml playbook.yml
```

Verifiez que l'application repond :

```
curl http://localhost:8080/health
```

12. Probleme volontaire

Modifiez le fichier group_vars/frontend.yml et changez backend_host :

```
backend_host: wronghost
```

Relancez le playbook, puis testez :

```
curl http://localhost:8080/health
```

Questions :

- Que se passe-t-il et pourquoi ?
- Comment Ansible vous aide-t-il a identifier le probleme ?
- Ou dans les logs trouvez-vous l'erreur ?
- Comment corriger ?

Corrigez la valeur de backend_host et relancez le playbook. Verifiez que l'application repond correctement.

13. Challenge final

Vous devez ajouter un nouveau service de monitoring.

Sans aide, creez un role Ansible pour installer et configurer Netdata sur le conteneur flask.

Contraintes :

- Le service doit demarrer automatiquement

- Le port d'ecoute doit etre defini dans une variable
- Un handler doit gerer le redemarrage
- Le conteneur flask doit exposer le port correspondant dans main.tf

Verifiez que Netdata est accessible depuis votre navigateur.

Bonus

- Ajoutez un tag Ansible sur chaque play pour pouvoir executer uniquement un role specifique
- Creez un fichier ansible.cfg pour eviter de specifier l'inventaire a chaque commande
- Chiffrez le fichier group_vars/db.yml avec ansible-vault pour proteger le mot de passe