Infrastructure As Code

Introduction aux concepts de virtualisation

Qu'est-ce que l'Infrastructure As Code (IaC)

Les problèmes courants

- **Configuration drift** :
  Les serveurs divergent de l'état désiré (configuration manuelle non tracée)
- **Processus lents** :
  Le provisionnement et les mises à jour sont manuels et sujets à erreur
- **Documentation défaillante** :
  L'état réel de l'infrastructure est inconnu

La solution IaC

Gérer l'infrastructure (réseaux, VM, services) via des fichiers de configuration, non via des processus manuels

- Décrire l’infrastructure dans du code versionné
- Automatiser la création, la configuration et la maintenance
- Garantir la cohérence entre environnements
- Faciliter la collaboration (DevOps)
- Réduire les erreurs humaines

Principes clefs

Déclaratif : décrire l'état final (ex: "Je veux un serveur Nginx installé")

Idempotence : Exécuter le code plusieurs fois a le même résultat (évite les réplications inutiles)

Avantages : Vitesse, Fiabilité, Versionnement (Git), Auditabilité

Ecosystème IaC

- **Solutions de Provisionning** : Terraform, CloudFormation, Pulumi, ...
- **Configuration Management** : Ansible, Chef, Puppet, SaltStack, ...

- **Cloud providers** : AWS, GCP, Azure, OVH, ...
- **Hyperviseurs** : VMWare, Proxmox, ...
- **Containers & Orchestration** : Docker, Kubernetes, ...
- **CI/CD & GitOps** : GitLab CI, ArgoCD, FluxCD, ...

Ansible

Qu'est-ce qu'Ansible

Ansible provides open-source automation that reduces complexity and runs everywhere. Using Ansible lets you automate virtually any task. Here are some common use cases for Ansible:

- Eliminate repetition and simplify workflows
- Manage and maintain system configuration
- Continuously deploy complex software
- Perform zero-downtime rolling updates

Principes d'Ansible

- **Agent-less architecture**
  Low maintenance overhead by avoiding the installation of additional software across IT infrastructure.

- **Simplicity**
  Automation playbooks use straightforward YAML syntax for code that reads like documentation. Ansible is also decentralized, using SSH with existing OS credentials to access to remote machines.

- **Scalability and flexibility**
  Easily and quickly scale the systems you automate through a modular design that supports a large range of operating systems, cloud platforms, and network devices.

- **Idempotence and predictability**
  When the system is in the state your playbook describes, Ansible does not change anything, even if the playbook runs multiple times.

Principes d'Ansible

Schéma d'Ansible avec les machines de l'inventaire
Lien avec Terraform qui provisionne les machines

Concepts Ansible

- **Control node** :
- **Managed node** :
- **Inventory** : le fichier qui liste et organise les machines cibles
- **Playbook** : fichiers YAML qui décrivent les tâches à exécuter, composés de :
  - **Plays** : cibles
  - **tasks** : actions
  - **roles** : des regroupements de configuration pour modulariser les playbook

- **modules** : programme qui effectuent le paramétrage

Inventory

https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

- Le plus souven **ini** ou **yaml**
- Objectif : identifier les machines gérées par Ansible

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

group

managed node

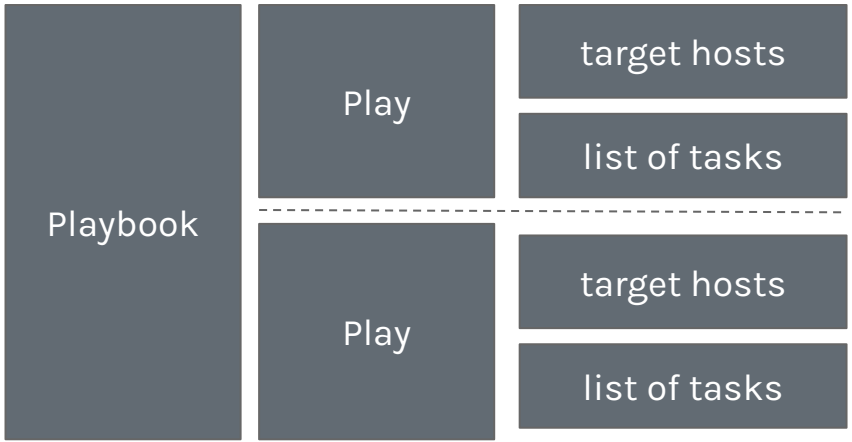
Par défaut, Ansible se connecte en SSH sur le port 22 avec le même user que celui utilisé sur le control node. Utiliser les variables pour modifier cela :

```
[webservers]
web1 ansible_host=192.168.1.10 ansible_port=2222 ansible_user=ubuntu
```

Playbook

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html

- Associe un groupe de machine à une série de tâche
- Exécute les tâches dans l'ordre



Les **play** s'exécutent séquentiellement

Au sein d'un play, les tâches s'exécutent séquentiellement

Les machines d'un même play sont traités en parallèle

```
ansible-playbook playbook.yml -f 10
```

nombre de *forks* = nombre d'exécution parallèles

Tasks

Au sein d'un playbook, chaque play définit une target et une série de tasks. Chaque task utilise un module pour appliquer la configuration demandée

playbook

```
---
- name: Write hostname
  hosts: all
  tasks:
  - name: write hostname using jinja2
    ansible.builtin.template:
      src: templates/test.j2
      dest: /tmp/hostname
```

targets, depuis l'inventaire

module à exécuter sur cette tâche

paramètres du module

Les modules les plus utiles

1. `apt` / `yum` / `package`

- **Objectif :** Installer, supprimer ou mettre à jour des paquets sur les systèmes Linux.

```yaml
- name: Installer nginx
  apt:
    name: nginx
    state: present
```

2. `copy`

- **Objectif :** Copier des fichiers depuis la machine de contrôle vers les hôtes gérés.

```yaml
- name: Copier un fichier de configuration
  copy:
    src: ./nginx.conf
    dest: /etc/nginx/nginx.conf
    owner: root
    group: root
    mode: '0644'
```

3. `template`

- **Objectif :** Copier des fichiers tout en traitant des **templates Jinja2** (utile pour les configurations dynamiques).

```yaml
- name: Déployer le fichier de configuration nginx
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
```

4. `service` / `systemd`

- **Objectif :** Démarrer, arrêter, activer ou redémarrer des services.

```yaml
- name: S'assurer que nginx est démarré
  service:
    name: nginx
    state: started
    enabled: yes
```

5. `user`

- **Objectif :** Gérer les utilisateurs et groupes sur les hôtes distants.

```yaml
- name: Créer un utilisateur
  user:
    name: deploy
    state: present
    shell: /bin/bash
    groups: sudo
```

Playbook : exemple

```yaml
---
- name: Déploiement d'un serveur Nginx avec Ansible 2.18
  hosts: webservers
  become: true
  gather_facts: true

  vars:
    nginx_user: www-data
    nginx_conf_src: templates/nginx.conf.j2
    nginx_conf_dest: /etc/nginx/nginx.conf

  tasks:
    - name: S'assurer que la liste des paquets est à jour
      ansible.builtin.apt:
        update_cache: true
        cache_valid_time: 3600

    - name: Installer Nginx
      ansible.builtin.apt:
        name: nginx
        state: present

    - name: Créer un utilisateur applicatif
      ansible.builtin.user:
        name: deploy
        shell: /bin/bash
        groups: sudo
        state: present

    - name: Déployer le fichier de configuration nginx (template Jinja2)
      ansible.builtin.template:
        src: "{{ nginx_conf_src }}"
        dest: "{{ nginx_conf_dest }}"
        owner: root
        group: root
        mode: '0644'
      notify: Redémarrer nginx

    - name: Copier une page d'accueil statique
      ansible.builtin.copy:
        src: files/index.html
        dest: /var/www/html/index.html
        owner: "{{ nginx_user }}"
        group: "{{ nginx_user }}"
        mode: '0644'

    - name: S'assurer que le service nginx est activé et démarré
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true

  handlers:
    - name: Redémarrer nginx
      ansible.builtin.service:
        name: nginx
        state: restarted
```

Facts

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_vars_facts.html

Utiliser des variables qui représentent le noeud en cours : adresse ip, hostname, ...

```yaml
- name: Exemple d'utilisation des facts Ansible
  hosts: all
  gather_facts: true   # ← active la collecte automatique des facts (par défaut à true)
  become: false

  tasks:
    - name: Afficher le nom d'hôte de la machine distante
      ansible.builtin.debug:
        msg: "Nom d'hôte détecté : {{ ansible_facts['hostname'] }}"

    - name: Afficher le système d'exploitation
      ansible.builtin.debug:
        msg: "Système d'exploitation : {{ ansible_facts['os_family'] }} {{ ansible_facts['distribution_version'] }}"

    - name: Afficher l'adresse IP principale
      ansible.builtin.debug:
        msg: "Adresse IP : {{ ansible_facts['default_ipv4']['address'] }}"

    - name: Utiliser une condition basée sur un fact
      ansible.builtin.debug:
        msg: "Cette machine est sous Debian ou Ubuntu"
      when: ansible_facts['os_family'] == "Debian"
```

Variables

Les variables peuvent être définies à plusieurs niveaux :

- inventaire :
- Fichiers group_vars / host_vars : chargé automatiquement en fonction de l'hote
- Playbook (vars) : dans le bloc var du playbook
- Tâche (vars) : dans le bloc var d'une tâche
- vars_files : inclusion d'un fichier yaml contenant des variables
- Facts
- Extra-vars (--extra-vars) : passés en ligne de commande
- Rôles (defaults / vars) : définies au niveau du rôle
- Environment variables : via le système du control node

Exemples

inventaire

```
[webservers]
web1 ansible_host=10.0.0.10 ansible_user=ubuntu
web2 ansible_host=10.0.0.11 ansible_user=ubuntu

[webservers:vars]
http_port=80
```

group vars

```
# group_vars/webservers.yml
http_port: 8080
nginx_package: nginx
```

Fichier de variables

```
- hosts: all
  vars_files:
    - vars/common.yml
```

Ligne de commande

```
ansible-playbook site.yml -e "env=prod version=2.1.0"
```

Rôles

Rôles - exemple

```yaml
---
- name: Déployer un serveur web simple
  hosts: webservers
  become: true
  tasks:
    - name: Installer nginx
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: true

    - name: Copier la page d'accueil
      ansible.builtin.copy:
        src: files/index.html
        dest: /var/www/html/index.html
        mode: '0644'

    - name: Démarrer le service nginx
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
```

Playbook simple, peu réutilisable => on va sortir la logique de "serveur web"

Rôles - exemple

```
project/
├── site.yml
└── roles/
    └── nginx/
        ├── tasks/
        │   └── main.yml
        ├── files/
        ├── templates/
        ├── vars/
        │   └── main.yml
        ├── handlers/
        │   └── main.yml
        └── defaults/
            └── main.yml
```

```yaml
---
- name: Déployer un serveur web avec un rôle
  hosts: webservers
  become: true
  roles:
    - nginx
```

```yaml
---
- name: Installer nginx
  ansible.builtin.apt:
    name: nginx
    state: present
    update_cache: true

- name: Copier la page d'accueil
  ansible.builtin.copy:
    src: index.html
    dest: /var/www/html/index.html
    mode: '0644'
  notify: Redémarrer nginx

- name: Démarrer nginx
  ansible.builtin.service:
    name: nginx
    state: started
    enabled: true
```

```yaml
---
- name: Redémarrer nginx
  ansible.builtin.service:
    name: nginx
    state: restarted
```

Utilisation de plusieurs rôles

```yaml
---
- name: Déployer un serveur web complet
  hosts: webservers
  become: true
  roles:
    - firewall
    - nginx
```

On peut utiliser plusieurs rôles, pour découper la configuration en blocs réutilisables

Variabilisation d'un rôle

```
roles/
└── webserver/
    ├── tasks/
    │   └── main.yml
    └── vars/
        └── main.yml
```

On veut sélectionner le **package** de serveur web en fonction de la **distribution**

- une **variable** dictionnaire définit dans le rôle
- les variables **facts** pour sélectionner la distrib

```yaml
---
# Nom du package Nginx selon la distribution
nginx_package_name:
  Debian: nginx
  Ubuntu: nginx
  RedHat: httpd
  CentOS: httpd
  Rocky: httpd
  Fedora: nginx
  Archlinux: nginx

# Nom du service selon la distribution
nginx_service_name:
  Debian: nginx
  Ubuntu: nginx
  RedHat: httpd
  CentOS: httpd
  Rocky: httpd
  Fedora: nginx
  Archlinux: nginx
```

```yaml
---
- name: Installer le package Nginx ou Httpd selon la distro
  package:
    name: "{{ nginx_package_name[ansible_os_family] | default('nginx') }}"
    state: present

- name: Assurer que le service est démarré et activé
  service:
    name: "{{ nginx_service_name[ansible_os_family] | default('nginx') }}"
    state: started
    enabled: true
```