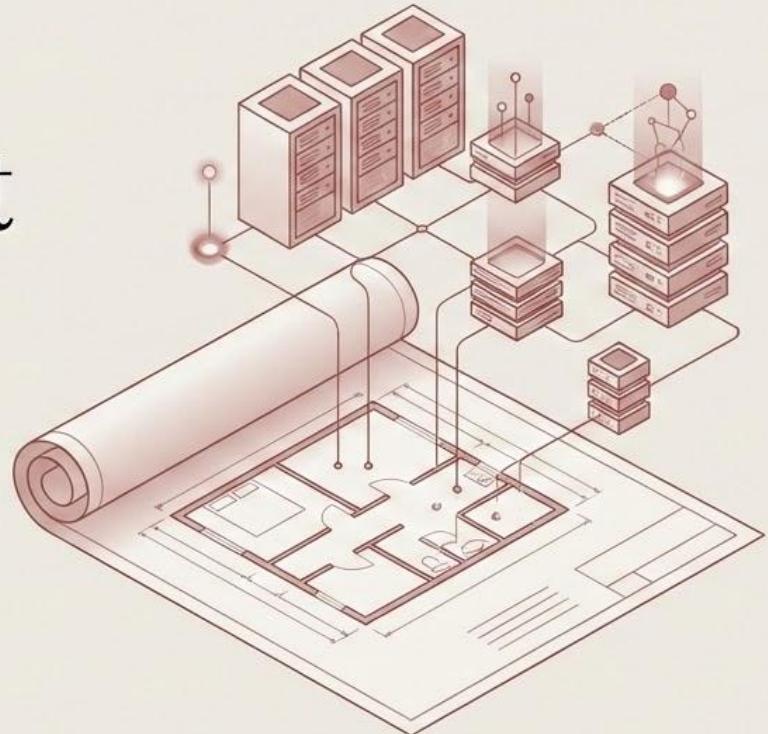


Provisionnement et Configuration d'Infrastructure

Avec Terraform et Ansible



Introduction à Terraform et Infrastructure as Code

Infrastructure as Code

Les outils d'infrastructure en tant que code (IaC) vous permettent de gérer l'infrastructure à l'aide de fichiers de configuration plutôt que via une interface utilisateur graphique.

Terraform c'est quoi ?

Terraform est une IaC vous permet de créer, de modifier et de gérer votre infrastructure de manière sûre, cohérente et reproductible en définissant des configurations de ressources que vous pouvez versionner, réutiliser et partager.

Intérêt de Terraform ?

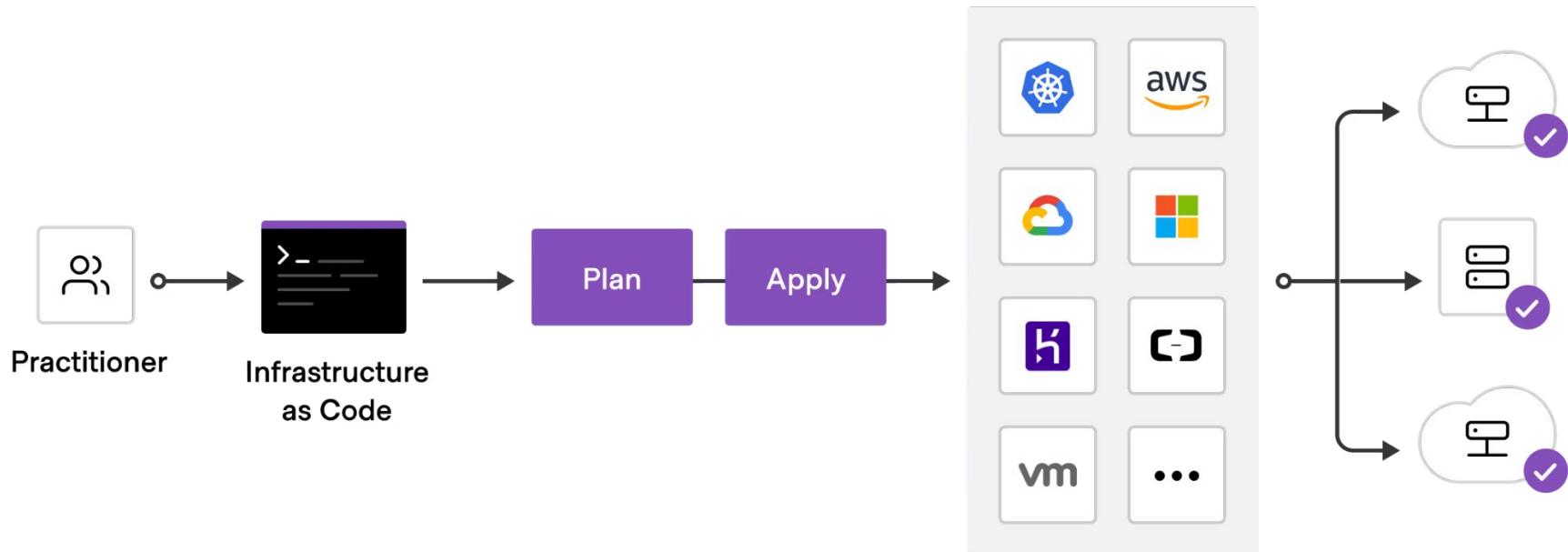
- peut gérer l'infrastructure sur plusieurs plates-formes cloud.
- langage de configuration lisible qui aide à écrire rapidement du code d'infrastructure.
- gestion d'état permet de suivre les modifications des ressources
- validation dans le contrôle de version

Standardisez le workflow de déploiement

Terraform est déclaratif, ce qui signifie qu'il décrit l'état final souhaité pour votre infrastructure.

Les fournisseurs Terraform calculent automatiquement les dépendances entre les ressources pour les créer ou les détruire dans le bon ordre.

Schéma de workflow



Déployer une infrastructure avec Terraform

1. identifiez l'infrastructure de votre projet.
2. écrivez la configuration de votre infrastructure.
3. installez les plugins dont Terraform a besoin pour gérer l'infrastructure.
4. prévisualisez les modifications que Terraform effectuera pour correspondre à votre configuration.
5. effectuez les modifications prévues.

registry/main.tf

```
provider "aws" {
  region = "eu-west-3" # Paris
}

# 1. AMI Ubuntu 24.04
data "aws_ami" "ubuntu" {
  most_recent = true
  owners       = ["099720109477"]

  filter {
    name    = "name"
    values  = ["ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-*"]
  }
}
```

registry/main.tf

```
# 2. Création de la clé SSH
resource "tls_private_key" "pk" {
    algorithm = "RSA"
    rsa_bits   = 4096
}

resource "aws_key_pair" "generated_key" {
    key_name     = "registry-key-terraform"
    public_key   = tls_private_key.pk.public_key_openssh
}

# Sauvegarde de la clé privée locale pour Ansible
resource "local_file" "ssh_key" {
    filename      = "${path.module}/registry-key-terraform.pem"
    content       = tls_private_key.pk.private_key_pem
    file_permission = "0400"
}
```

registry/main.tf

```
resource "aws_security_group" "registry_sg" {
  name      = "registry-sg-simple"
  description = "Allow SSH, HTTP (UI), Registry (5000)"
  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "Registry UI"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "Registry Docker API"
    from_port   = 5000
    to_port     = 5000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

registry/main.tf

```
# 4. Instance EC2
resource "aws_instance" "registry_server" {
    ami          = data.aws_ami.ubuntu.id
    instance_type = "t3.micro"
    key_name      = aws_key_pair.generated_key.key_name

    vpc_security_group_ids = [aws_security_group.registry_sg.id]

    root_block_device {
        volume_size = 20
        volume_type = "gp3"
    }

    tags = {
        Name = "Terraform-Registry-Server"
    }
}

# 5. Output (Pour récupérer l'IP facilement)
output "instance_ip" {
    value = aws_instance.registry_server.public_ip
}
```

Phase de test

1. Initialiser les plugins AWS

terraform init

2. Créer l'infrastructure

terraform apply

3. Noter l'adresse IP

Présentation d'Ansible : configuration et déploiement applicatif



Introduction à Ansible

Ansible est un outil d'automatisation open source qui permet de gérer la configuration, le déploiement, l'orchestration, et le provisionnement de machines sans agent

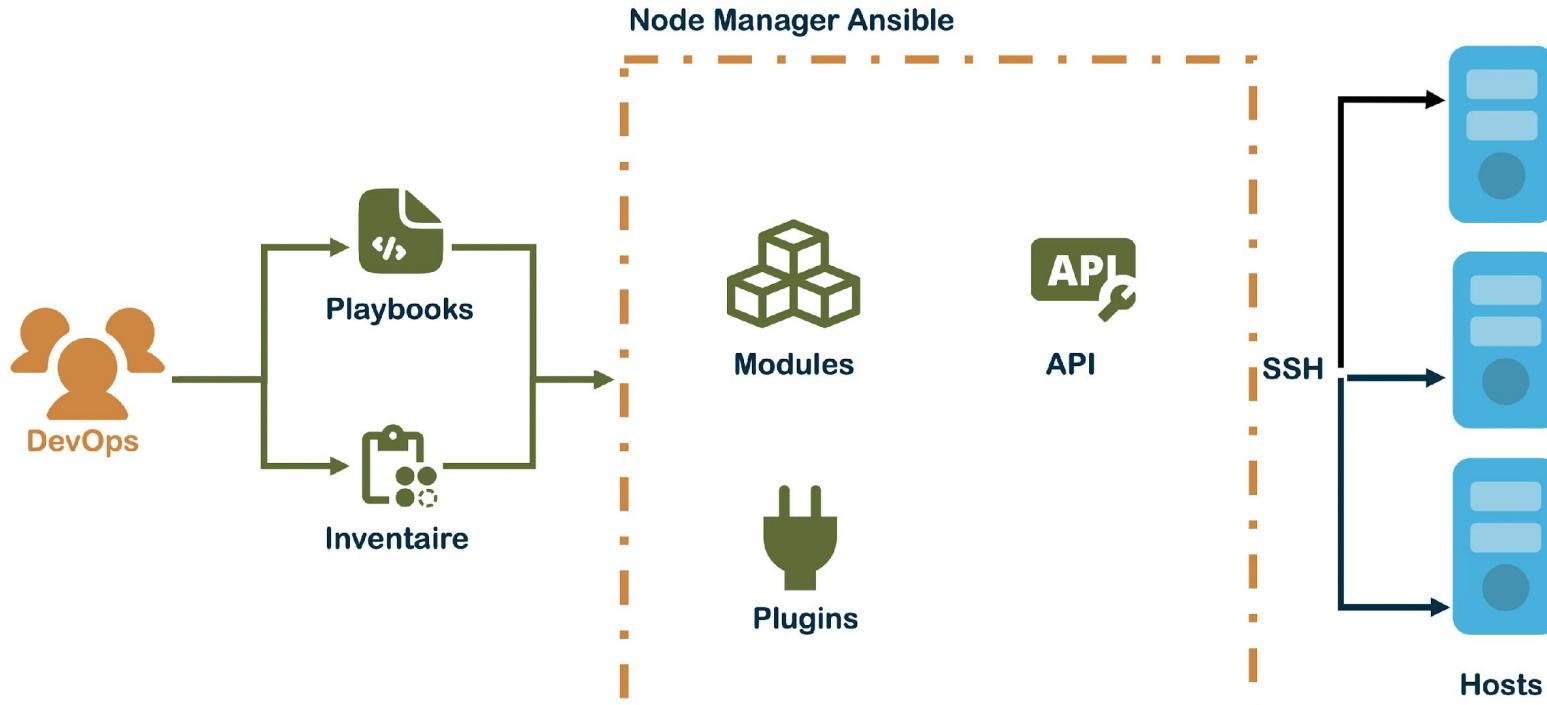
Principes

- Utilisation d'une approche « Infrastructure as code » et d'un langage déclaratif basé sur YAML.
- Fonctionne sans agent grâce à SSH : un seul serveur Ansible pilote l'ensemble des machines cibles.

Les composants clés

Le serveur Ansible
Les modules
Les playbooks

Les composants clés



Gestion de configuration avec Ansible

- Permet de définir et d'appliquer l'état souhaité sur les serveurs et services.
- Le contrôle centralisé évite la configuration manuelle, réduit les erreurs humaines et favorise la cohérence du parc.

Exemple de configuration

```
---
```

```
- name: Installer et démarrer Apache
  hosts: webservers
  become: yes
```

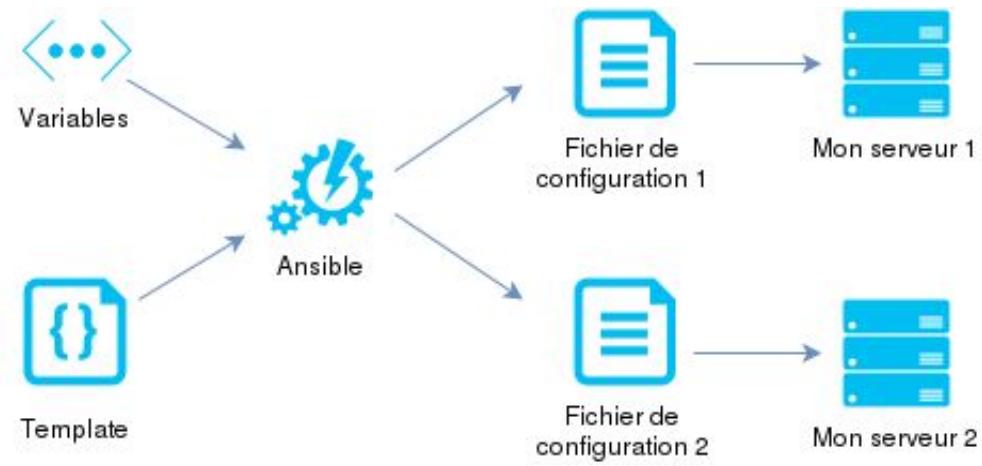
```
tasks:
```

```
  - name: Installer Apache
    apt:
      name: apache2
      state: present
      update_cache: yes
```

```
  - name: S'assurer que le service Apache est démarré
    service:
      name: apache2
      state: started
      enabled: yes
```

Les templates

Adapter vos fichiers de configuration avec des variables propres à chaque serveur.



Déploiement applicatif avec Ansible

- Automatise le déploiement d'applications sur une ou plusieurs machines.
- Les playbooks orchestrent les étapes : installation, configuration, démarrage des services, tests de bon fonctionnement.
- Support du déploiement multi-niveaux et orchestration de dépendances entre services applicatifs

Avantages et points forts

- Facile à prendre en main, pas de dépendance à des agents.
- Documentation et communauté active.
- Flexible et extensible (modules, rôles, Galaxy).
- Sécurité : gestion centralisée, support d'Ansible Vault

Bonnes pratiques

- Favoriser les rôles pour organiser les tâches, variabiliser au maximum, ne jamais placer de secrets en clair.
- Utiliser les fonctions de check et de debug pour fiabiliser les playbooks.
- Tenir à jour la version d'Ansible et la documentation.

docker compose

```
services:
  >Run Service
  registry:
    image: registry:2
    restart: always
    ports:
      - "5000:5000"
    environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/registry.password
      REGISTRY_STORAGE_DELETE_ENABLED: "true"
    volumes:
      - ./data:/var/lib/registry
      - ./auth:/auth:ro
  >Run Service
  ui:
    image: joxit/docker-registry-ui:main
    restart: always
    ports:
      - "80:80"
    environment:
      - REGISTRY_TITLE=Ansible Deployed Registry
      - REGISTRY_URL=http://{{ public_ip }}:5000
      - SINGLE_REGISTRY=true
      - DELETE_IMAGES=true
    depends_on:
      - registry
```

registry/playbook.yml

```
tasks:
  # -----
  # 1. INSTALLATION DES DÉPENDANCES
  # -----
  - name: Mise à jour du cache APT
    apt:
      update_cache: yes

  - name: Installation de Docker et outils
    apt:
      name:
        - docker.io
        - docker-compose-v2
        - apache2-utils # Pour htpasswd
        - openssl
      state: present

  - name: Ajout de l'utilisateur ubuntu au groupe docker
    user:
      name: ubuntu
      groups: docker
      append: yes
```

registry/playbook.yml

```
---
- name: Déploiement d'un Registry Docker Sécurisé
  hosts: registry_hosts
  become: true
  vars:
    project_dir: "/home/ubuntu/registry-stack"
    # On récupère l'IP publique automatiquement via les facts Ansible
    public_ip: "{{ ansible_host }}"
```

registry/playbook.yml

```
tasks:
  # 1. Installation Docker
  - name: Installation des paquets
    apt:
      update_cache: yes
      name:
        - docker.io
        - docker-compose-v2
        - apache2-utils # pour htpasswd
      state: present

  - name: Ajout user au groupe docker
    user:
      name: ubuntu
      groups: docker
      append: yes
```

registry/playbook.yml

```
- name: Création des dossiers
  file:
    path: "{{ item }}"
    state: directory
    owner: ubuntu
    group: ubuntu
    mode: '0755'
  loop:
    - "{{ project_dir }}"
    - "{{ project_dir }}/auth"
    - "{{ project_dir }}/data"
```

registry/playbook.yml

```
- name: Création des dossiers
  file:
    path: "{{ item }}"
    state: directory
    owner: ubuntu
    group: ubuntu
    mode: '0755'
  loop:
    - "{{ project_dir }}"
    - "{{ project_dir }}/auth"
    - "{{ project_dir }}/data"
```

registry/playbook.yml

```
- name: Génération mot de passe (admin/admin123)
  htpasswd:
    path: "{{ project_dir }}/auth/registry.password"
    name: admin
    password: admin123
    owner: ubuntu
    group: ubuntu
    mode: 0640
```

registry/playbook.yml

```
- name: Copie du docker-compose.yml
  template:
    src: docker-compose.yml
    dest: "{{ project_dir }}/docker-compose.yml"
    owner: ubuntu
    group: ubuntu
    mode: '0644'

# 5. Start
- name: Démarrage de la stack
  command: docker compose up -d
  args:
    chdir: "{{ project_dir }}"
```

registry/inventory.ini

```
[registry_hosts]
# Liste des IPs (une par ligne)
<IP_PUBLIQUE_AWS>

[registry_hosts:vars]
# Variables communes à tous les hôtes du groupe
ansible_user=ubuntu
ansible_ssh_private_key_file=registry-key-terraform.pem
# On désactive la vérification de la clé hôte pour éviter le prompt "Yes/No"
ansible_ssh_common_args='"-o StrictHostKeyChecking=no'"
```

Phase de test

#1 donner les droits au fichier .pem
chmod 400
registry-key-terraform.pem

#2 lancer le playbook

ansible-playbook -i inventory.ini
playbook.yml

#3 tester le registry

docker login <VOTRE_IP>:5000

User: admin

Password: admin123