Infrastructure as Code

Automatiser et gérer votre infrastructure

OBJECTIFS DES OUTILS D'AUTOMATISATION

Reproductibilité

Garantir des environnements identiques à chaque déploiement, éliminant les variations et les erreurs manuelles.

Rapidité et efficacité

Accélérer le déploiement d'infrastructure et réduire le temps nécessaire pour provisionner de nouveaux environnements.

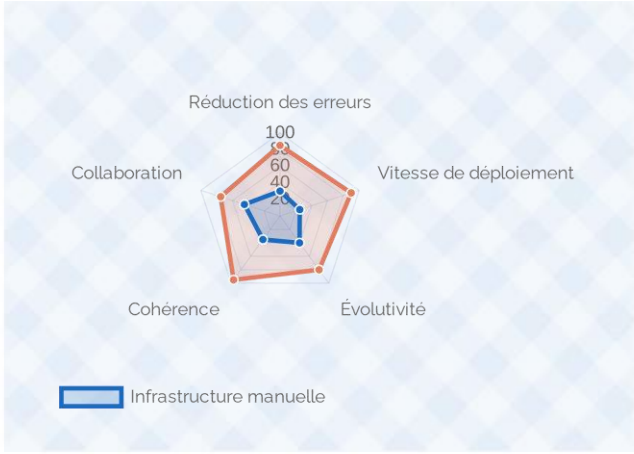
Versionnement et traçabilité

Suivre les modifications d'infrastructure au fil du temps et permettre de revenir à des versions antérieures si nécessaire.

Sécurité et conformité

Appliquer systématiquement les politiques de sécurité et garantir la conformité aux normes réglementaires.

Bénéfices de l'automatisation d'infrastructure



PRINCIPES DE L'INFRASTRUCTURE AS CODE

L'Infrastructure as Code (IaC) est une approche qui consiste à gérer et provisionner l'infrastructure informatique à travers des fichiers de configuration plutôt que par des processus manuels. Elle permet de traiter l'infrastructure de la même manière que les développeurs traitent le code source.

Infrastructure déclarative

Définir l'état souhaité de l'infrastructure plutôt que les étapes pour y parvenir, permettant aux outils de déterminer comment atteindre cet état.

Contrôle de version

Stocker les définitions d'infrastructure dans un système de contrôle de version pour suivre les modifications et faciliter la collaboration.

Modularité

Concevoir l'infrastructure en composants réutilisables qui peuvent être combinés pour créer des environnements complexes.

Idempotence

Garantir que l'application répétée du même code d'infrastructure produit toujours le même résultat, sans effets secondaires inattendus.

Tests automatisés

Valider les changements d'infrastructure avant leur déploiement pour garantir la qualité et éviter les régressions.

Collaboration DevOps

Favoriser la collaboration entre les équipes de développement et d'opérations en utilisant un langage commun pour l'infrastructure.

PROVISIONING ET INFRASTRUCTURE IMMUTABLE

Provisioning d'infrastructure

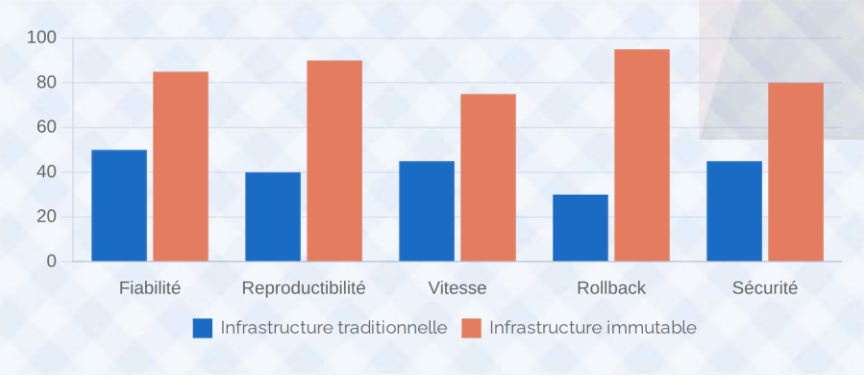
Le provisioning d'infrastructure est le processus d'allocation et de configuration des ressources informatiques nécessaires pour exécuter des applications et des services.

- Automatisation du déploiement des serveurs, réseaux et stockage
- Configuration des systèmes d'exploitation et applications
- Orchestration du déploiement dans différents environnements

Infrastructure immutable

L'infrastructure immutable est une approche où les composants d'infrastructure ne sont jamais modifiés après leur déploiement. Pour tout changement, de nouveaux composants sont déployés.

- Élimination de la dérive de configuration et des incohérences
- Déploiements plus fiables et prévisibles
- Rollbacks simplifiés et sécurité renforcée



AVANTAGES DE L'IAC

Réduction des coûts

Diminution des coûts opérationnels grâce à l'automatisation des tâches manuelles et à l'optimisation de l'utilisation des ressources.

Accélération du déploiement

Réduction significative du temps nécessaire pour déployer de nouveaux environnements ou mettre à jour des infrastructures existantes.

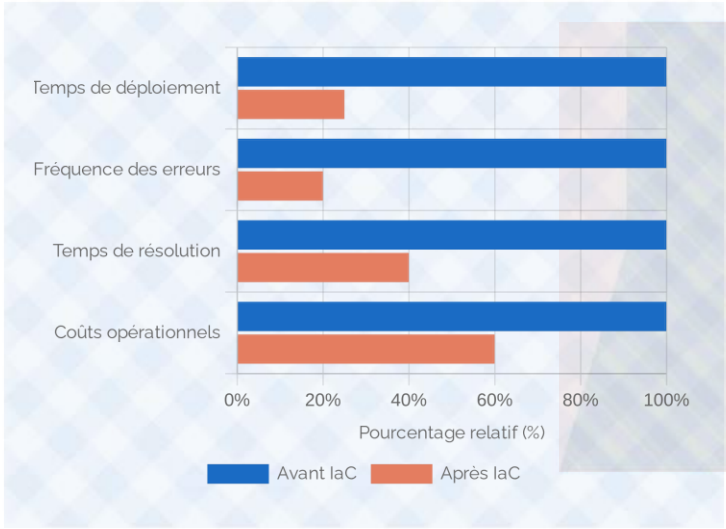
Réduction des erreurs

Élimination des erreurs humaines grâce à des processus standardisés et automatisés, garantissant la cohérence entre les environnements.

Évolutivité simplifiée

Capacité à faire évoluer rapidement l'infrastructure pour répondre aux besoins changeants de l'entreprise et aux pics de charge.

Impact de l'IaC sur les métriques clés



OUTILS DU MARCHÉ

Le marché propose une variété d'outils d'Infrastructure as Code, chacun avec ses spécificités et cas d'usage. Le choix dépend souvent de l'écosystème cloud, du langage préféré et des besoins spécifiques.

Terraform

Outil multi-cloud avec langage déclaratif HCL. Excellente gestion des dépendances et large écosystème de fournisseurs.

AWS CloudFormation

Solution native AWS utilisant JSON/YAML. Intégration profonde avec les services AWS et gestion des changements.

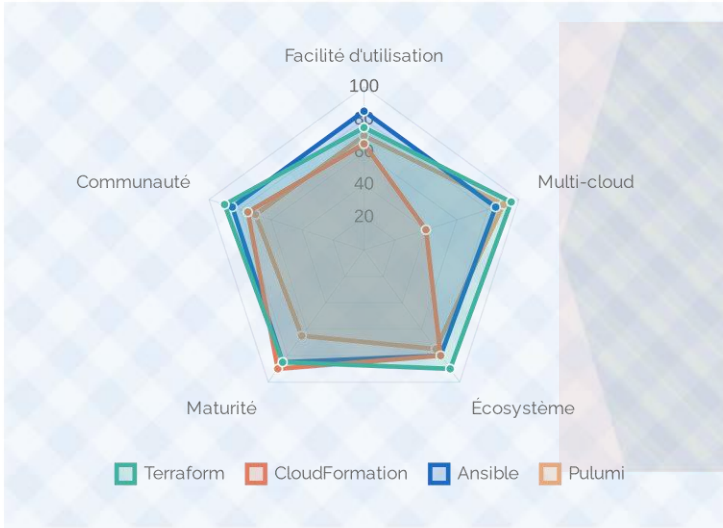
Ansible

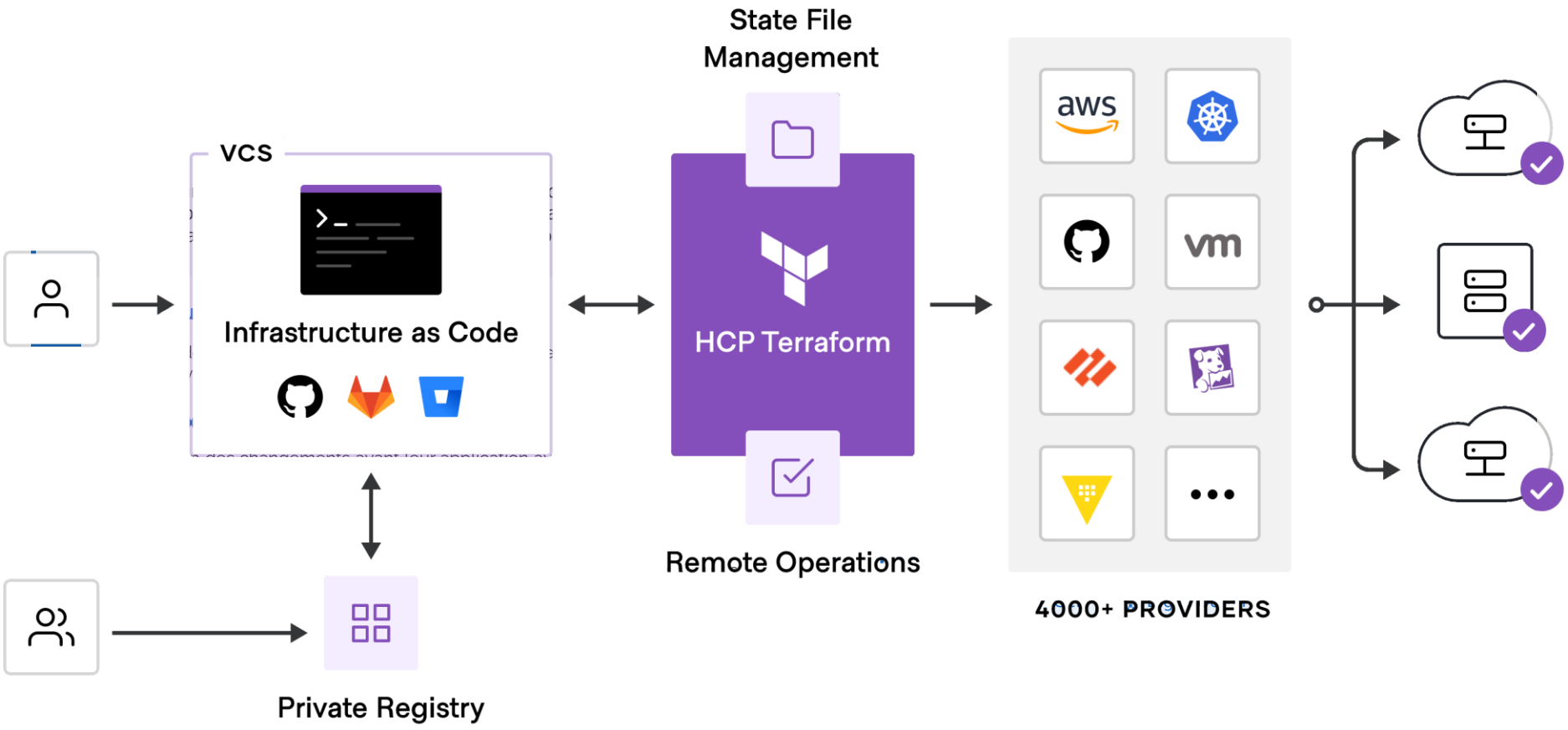
Approche sans agent utilisant YAML. Idéal pour la configuration et l'orchestration multi-plateformes.

Pulumi

Utilise des langages de programmation standards (Python, TypeScript, Go). Forte intégration avec les pratiques DevOps modernes.

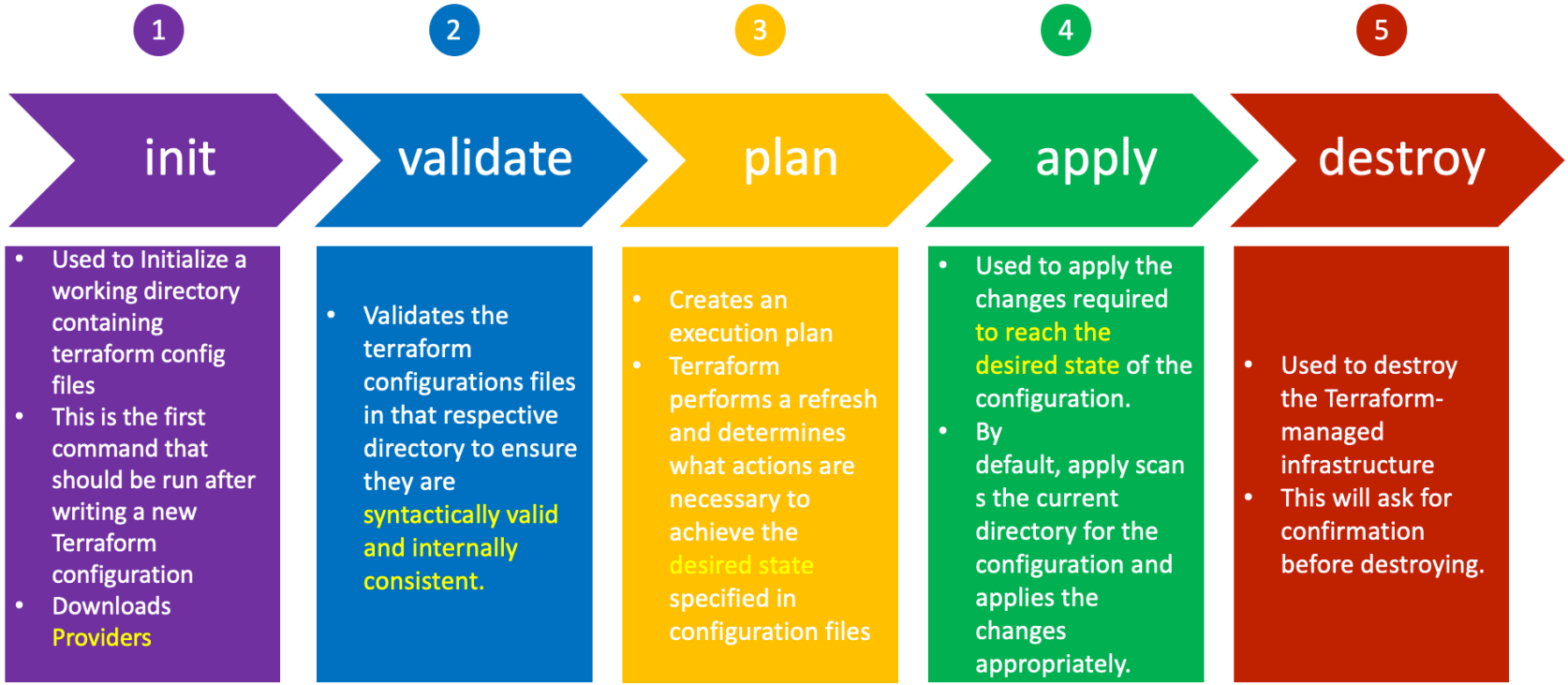
Comparaison des outils IaC



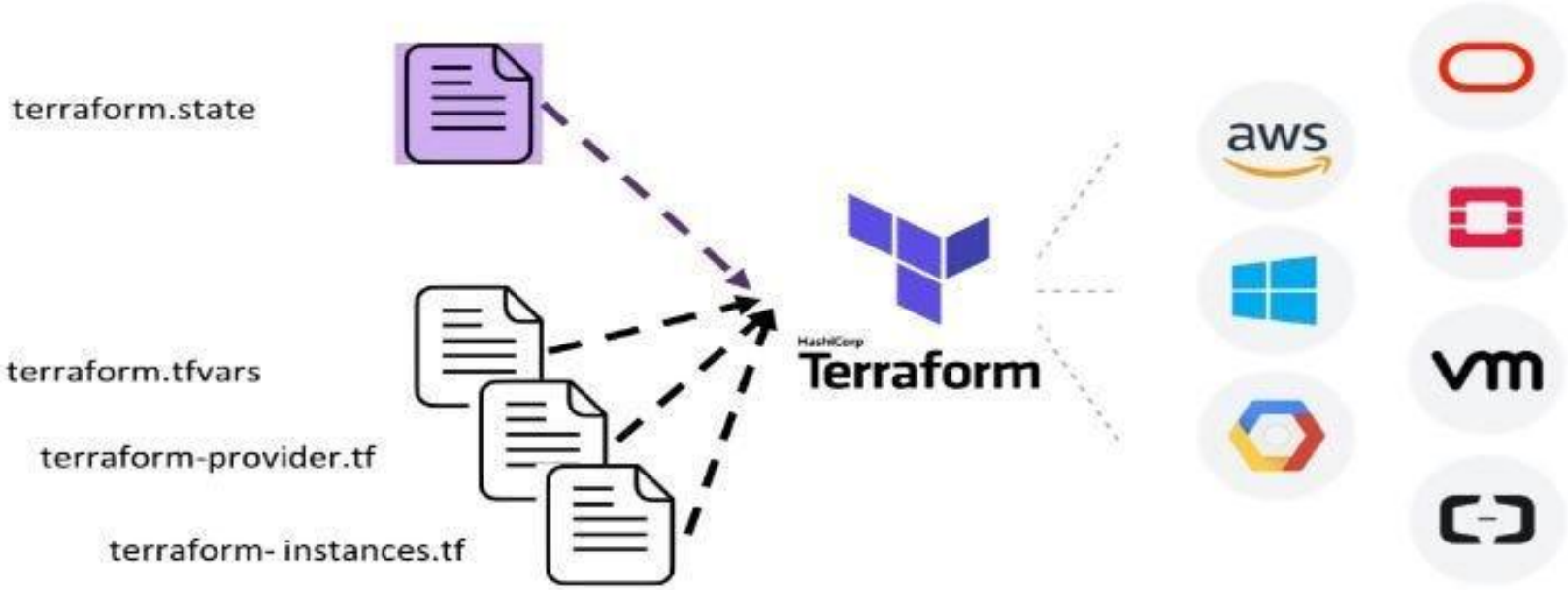

State File
Management
VCS
Infrastructure as Code
HCP Terraform
Remote Operations
aws
vm
4000+ PROVIDERS
Private Registry

TERRAFORM

Terraform Workflow



TERRAFORM


terraform.state
terraform.tfvars
terraform-provider.tf
terraform- instances.tf
HashiCorp
Terraform
aws
vm

CLOUDFORMATION

AWS CloudFormation est un service qui permet de modéliser et de provisionner des ressources d'infrastructure AWS à l'aide de fichiers de configuration au format JSON ou YAML. Il offre un moyen standardisé de créer et de gérer des ressources AWS de manière automatisée et sécurisée.

Intégration native avec AWS

Support complet de tous les services AWS avec mise à jour automatique lors du lancement de nouveaux services.

Gestion des dépendances

Création automatique des ressources dans le bon ordre et gestion des dépendances entre elles.

Rollback automatique

Retour automatique à l'état précédent en cas d'échec lors du déploiement d'une pile.

Nested Stacks

Possibilité de créer des piles imbriquées pour une meilleure organisation des ressources complexes.

Exemple de template CloudFormation

```yaml
# Template pour déployer une instance EC2
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Template pour créer une instance EC2'

Parameters:
  InstanceType:
    Description: Type d'instance EC2
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t2.medium

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref InstanceType
      ImageId: ami-0abcdef1234567890
      Tags:
        - Key: Name
          Value: WebServer
```

PULUMI

Pulumi

Pulumi est une plateforme d'Infrastructure as Code qui permet de créer, déployer et gérer des infrastructures cloud en utilisant des langages de programmation standards plutôt que des langages spécifiques ou des formats déclaratifs comme YAML.

Langages de programmation standards

Utilisation de langages comme Python, TypeScript, Go et C# pour définir l'infrastructure, permettant de tirer parti des IDE et des outils existants.

Multi-cloud natif

Support de tous les principaux fournisseurs cloud (AWS, Azure, GCP) et de nombreux services SaaS avec une API cohérente.

Composants réutilisables

Création de composants d'infrastructure réutilisables qui encapsulent les meilleures pratiques et peuvent être partagés entre les équipes.

Gestion d'état et sécurité

Gestion d'état robuste avec chiffrement et contrôle d'accès, disponible en mode SaaS ou auto-hébergé.

Exemple de code Pulumi (Python)

```python
# Infrastructure AWS avec Pulumi en Python
import pulumi
import pulumi_aws as aws

# Créer un groupe de sécurité
security_group = aws.ec2.SecurityGroup(
    'web-secgrp',
    description='Enable HTTP access',
    ingress=[
        aws.ec2.SecurityGroupIngressArgs(
            protocol='tcp',
            from_port=80,
            to_port=80,
            cidr_blocks=['0.0.0.0/0'],
        )
    ],
)

# Créer une instance EC2
server = aws.ec2.Instance(
    'web-server',
    instance_type='t2.micro',
    ami='ami-0c55b159cbfafe1f0',
    security_groups=[security_group.name],
    tags={
```

ANSIBLE - INTRODUCTION

Ansible

Ansible est un outil open-source d'automatisation IT qui permet de configurer des systèmes, déployer des logiciels et orchestrer des tâches informatiques avancées. Contrairement à d'autres outils d'IaC, Ansible utilise une approche sans agent et se concentre sur la simplicité et la facilité d'utilisation.

Sans agent

Fonctionne sans nécessiter l'installation d'agents sur les systèmes cibles, utilisant uniquement SSH pour les connexions.

YAML déclaratif

Utilise des fichiers YAML simples et lisibles (playbooks) pour décrire l'état souhaité des systèmes.

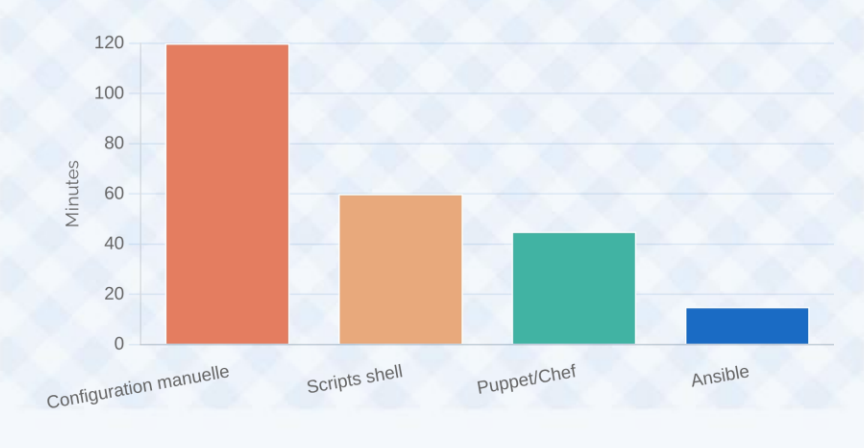
Modules extensibles

Large bibliothèque de modules pour interagir avec différents systèmes, services et plateformes.

Idempotence

Garantit que l'application répétée des mêmes configurations produit toujours le même résultat.

Comparaison des temps de déploiement



Cas d'utilisation courants

- Configuration de serveurs
- Déploiement d'applications
- Orchestration continue
- Gestion de la sécurité
- Configuration réseau
- Provisioning cloud

ANSIBLE - ARCHITECTURE

Ansible utilise une architecture sans agent (agentless) qui fonctionne en se connectant aux nœuds gérés via SSH ou WinRM. Cette approche simplifie le déploiement et réduit les exigences sur les systèmes cibles.

Nœud de contrôle

Machine où Ansible est installé et à partir de laquelle les playbooks sont exécutés. Seul ce nœud nécessite l'installation d'Ansible.

Inventaire

Liste des hôtes gérés, organisés en groupes, avec leurs variables spécifiques. Peut être statique (fichier) ou dynamique (script).

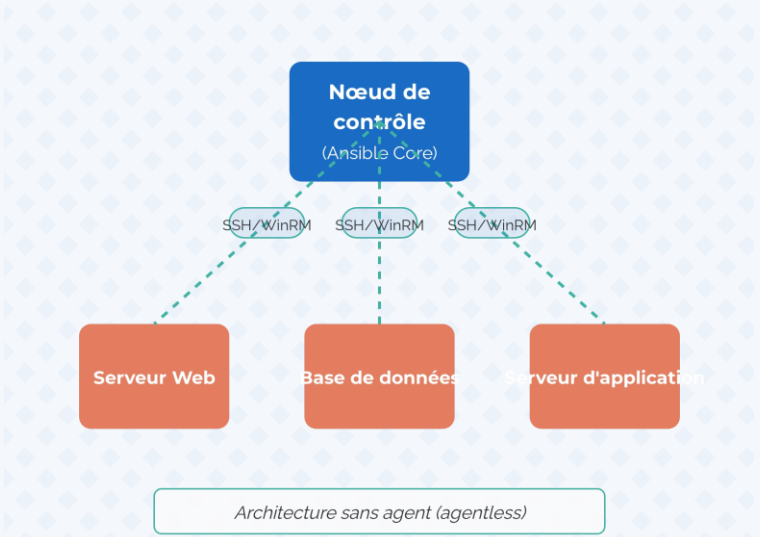
Playbooks

Fichiers YAML définissant les tâches à exécuter sur les hôtes cibles. Ils orchestrent le déploiement et la configuration.

Modules

Unités de code exécutant des actions spécifiques sur les hôtes cibles (gestion de fichiers, installation de paquets, etc.).

Architecture Ansible



ANSIBLE - PLAYBOOKS

Les playbooks sont au cœur d'Ansible. Ce sont des fichiers YAML qui décrivent une politique d'automatisation, un ensemble de tâches à exécuter sur des hôtes spécifiques dans un ordre défini.

Plays
Mappent un groupe d'hôtes à des tâches bien définies. Un playbook peut contenir plusieurs plays pour orchestrer des déploiements multi-niveaux.

Tasks
Unités d'action dans Ansible. Chaque tâche appelle un module avec des arguments spécifiques et s'exécute séquentiellement.

Modules
Unités de code qui effectuent des actions spécifiques, comme installer un paquet, copier un fichier ou redémarrer un service.

Roles
Permettent de regrouper le contenu de manière à faciliter le partage et la réutilisation du code. Ils organisent les playbooks en structures réutilisables.

Flux d'exécution d'un playbook

DÉMONSTRATION ANSIBLE

Exemple pratique : Déploiement d'une application web avec Ansible. Ce playbook installe et configure un serveur web NGINX, déploie une application et configure un pare-feu de base.

web_deploy.yml

```
# Playbook pour déployer une application web
---
- name: Déploiement d'une application web
  hosts: web_servers
  become: yes
  tasks:
    - name: Installation de NGINX
      apt:
        name: nginx
        state: present
        update_cache: yes

    - name: Démarrage du service NGINX
      service:
        name: nginx
        state: started
        enabled: yes

    - name: Copie des fichiers de l'application
      copy:
```

inventory.ini

```
# Fichier d'inventaire pour le déploiement web
[web_servers]
web1 ansible_host=192.168.1.101
web2 ansible_host=192.168.1.102

[web_servers:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=~/.ssh/id_rsa
http_port=80
app_env=production
```

1. **Création de l'inventaire**

   Définir les serveurs cibles dans un fichier d'inventaire (inventory.ini) avec leurs adresses IP et variables.

2. **Exécution du playbook**

Résultat de l'exécution

Le playbook s'exécute sur les serveurs cibles et effectue les tâches suivantes :

- Installation du serveur web NGINX
- Configuration et démarrage du service
- Déploiement des fichiers de l'application
- Configuration du pare-feu pour autoriser le trafic HTTP

Chaque tâche affiche son statut (OK, Changed, Failed) et des informations détaillées sur les modifications effectuées.

BONNES PRATIQUES

L'adoption de bonnes pratiques pour l'Infrastructure as Code est essentielle pour garantir la fiabilité, la maintenabilité et la sécurité de vos déploiements d'infrastructure. Voici les recommandations clés pour une implémentation réussie.

Contrôle de version

Stockez tout le code d'infrastructure dans un système de contrôle de version comme Git. Traitez votre infrastructure comme du code source avec des revues de code et des branches.

Modularité

Organisez votre code en modules réutilisables avec des interfaces bien définies. Cela facilite la maintenance et permet de composer des infrastructures complexes.

Tests automatisés

Implémentez des tests pour valider votre infrastructure : tests unitaires pour les modules, tests d'intégration pour les déploiements complets, et tests de sécurité.

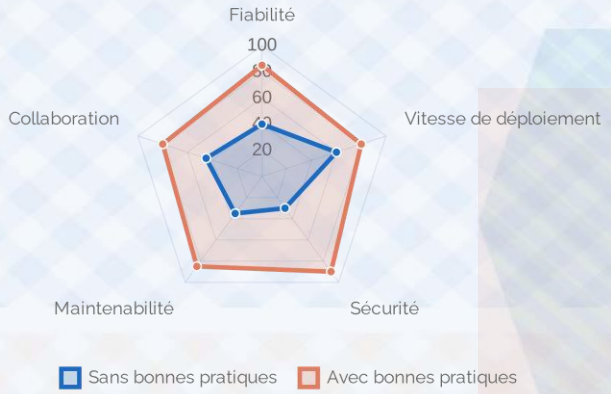
Gestion des secrets

Ne stockez jamais de secrets (mots de passe, clés API) dans votre code. Utilisez des coffres-forts de secrets comme HashiCorp Vault ou AWS Secrets Manager.

Documentation

Documentez votre infrastructure, les modules et leurs interfaces. Une documentation claire facilite l'adoption et la maintenance par l'équipe.

Impact des bonnes pratiques IaC



"L'Infrastructure as Code n'est pas seulement un outil technique, c'est une discipline qui nécessite rigueur et bonnes pratiques. Traitez votre infrastructure avec le même soin que votre code applicatif."

— Kief Morris, auteur de "Infrastructure as Code"

Conclusion

Infrastructure as Code

Gérer l'infrastructure avec la même rigueur que le code source pour une meilleure reproductibilité et fiabilité.

Diversité des outils

Choisir l'outil adapté à vos besoins spécifiques : Terraform pour le multi-cloud, CloudFormation pour AWS, Ansible pour la configuration.

Infrastructure immutable

Privilégier le remplacement plutôt que la modification pour une meilleure stabilité et sécurité des environnements.

Collaboration DevOps

Favoriser la collaboration entre développeurs et opérations grâce à un langage commun pour l'infrastructure.

Prochaines étapes

Commencez par un projet pilote simple, adoptez progressivement les pratiques IaC, et intégrez-les dans votre pipeline CI/CD pour une automatisation complète.

Pour plus d'informations : contact@example.com