Ansible : Introduction et Fondamentaux

Une approche moderne de l'automatisation IT et de la gestion de configuration pour les environnements DevOps

bstorm TRAINING

CHAPITRE 1

Qu'est-ce qu'Ansible ?

Ansible est un outil d'automatisation IT open-source qui simplifie radicalement la gestion de configuration, le déploiement d'applications et l'orchestration de tâches complexes. Créé par Michael DeHaan en 2012 et maintenant géré par Red Hat, Ansible s'est imposé comme l'une des solutions les plus populaires dans l'écosystème DevOps grâce à sa philosophie de simplicité et d'efficacité.

Contrairement aux outils traditionnels de gestion de configuration, Ansible adopte une approche déclarative qui permet aux équipes de définir l'état désiré de leur infrastructure plutôt que les étapes pour y parvenir. Cette philosophie "infrastructure as code" transforme la gestion d'infrastructure en un processus reproductible, versionnable et collaboratif.

L'outil utilise le langage YAML pour décrire les tâches d'automatisation, rendant les configurations accessibles même aux personnes sans compétences avancées en programmation. Cette accessibilité, combinée à sa puissance, fait d'Ansible un choix privilégié pour les organisations cherchant à accélérer leur transformation digitale et à adopter des pratiques DevOps modernes.

Le Concept Révolutionnaire : Architecture Agentless

L'une des caractéristiques les plus distinctives d'Ansible est son architecture **agentless** (sans agent). Contrairement à des outils comme Puppet ou Chef qui nécessitent l'installation d'un agent logiciel sur chaque machine gérée, Ansible fonctionne directement via les protocoles standards de l'infrastructure existante.

Comment ça fonctionne ?

SSH pour Linux/Unix : Ansible utilise le protocole SSH natif pour se connecter aux serveurs Linux et Unix, exploitant l'infrastructure de sécurité déjà en place

WinRM pour Windows : Pour les environnements Windows, Ansible s'appuie sur Windows Remote Management (WinRM), le protocole standard de Microsoft

Pas d'installation supplémentaire : Les machines cibles n'ont besoin d'aucun logiciel spécifique Ansible, seulement Python (généralement déjà présent)

Cette approche simplifie considérablement le déploiement initial et réduit la surface d'attaque sécuritaire. Il n'y a pas de démons à maintenir, pas de ports additionnels à ouvrir, et pas de problèmes de compatibilité d'agents à gérer lors des mises à jour.

Avantages Clés

- Déploiement immédiat sans préparation
- Moins de composants à maintenir
- Sécurité renforcée
- Réduction des coûts opérationnels
- Compatibilité universelle

Communication et Exécution : Le Flux de Travail Ansible

Le processus de communication d'Ansible suit un modèle push simple et efficace qui garantit la sécurité et la traçabilité de chaque opération.

1

Connexion SSH

Le nœud de contrôle initie une connexion SSH sécurisée vers les hôtes cibles

2

Transfert de Modules

Les modules Python nécessaires sont copiés temporairement sur les machines distantes

3

Exécution

Les modules s'exécutent localement sur chaque hôte avec les paramètres fournis

4

Nettoyage

Les fichiers temporaires sont supprimés et les résultats retournés au contrôleur

Cette approche garantit que rien ne persiste sur les machines gérées après l'exécution, maintenant un environnement propre et sécurisé. Chaque connexion est authentifiée via des clés SSH ou des méthodes d'authentification Kerberos, assurant un contrôle d'accès strict. Les logs détaillés permettent une traçabilité complète de toutes les opérations effectuées.

CHAPITRE 2

Pourquoi Choisir Ansible ?

Dans le paysage riche des outils de gestion de configuration et d'automatisation IT, Ansible se distingue par plusieurs avantages décisifs qui expliquent son adoption massive par des milliers d'entreprises à travers le monde. Comprendre ces avantages permet de prendre une décision éclairée lors du choix d'une solution d'automatisation pour votre infrastructure.

Le marché propose plusieurs alternatives sérieuses comme Puppet, Chef, et SaltStack, chacune avec ses forces et ses cas d'usage spécifiques. Cependant, Ansible a réussi à trouver un équilibre remarquable entre simplicité d'utilisation, puissance fonctionnelle et flexibilité d'adoption qui le rend particulièrement attractif pour les équipes de toutes tailles.

bstorm TRAINING

La Simplicité au Cœur : YAML et Lisibilité

Le Langage YAML

Ansible utilise YAML (Yet Another Markup Language) pour décrire l'infrastructure et les tâches d'automatisation. Ce choix n'est pas anodin : YAML est conçu pour être facilement lisible par les humains, avec une syntaxe minimale qui ressemble à de la prose structurée.

Avantages du YAML dans Ansible :

Courbe d'apprentissage douce : Pas besoin d'être développeur pour comprendre et écrire des playbooks

Collaboration facilitée : Les équipes ops et dev peuvent travailler ensemble sur le même code

Maintenance simplifiée : Le code reste compréhensible même des mois après l'avoir écrit

Moins d'erreurs : La syntaxe claire réduit les bugs de configuration

Exemple de Playbook Simple

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

  tasks:
    - name: Installer Apache
      apt:
        name: apache2
        state: present

    - name: Démarrer le service
      service:
        name: apache2
        state: started
        enabled: yes
```

Ce playbook est immédiatement compréhensible : il installe Apache sur les serveurs web et s'assure que le service est démarré. Pas de syntaxe complexe, pas de DSL obscur.

L'Idempotence : Un Principe Fondamental

L'idempotence est l'un des concepts les plus importants d'Ansible et un différenciateur majeur par rapport aux scripts shell traditionnels. Une opération idempotente peut être exécutée plusieurs fois sans changer le résultat au-delà de la première application.

Pourquoi l'idempotence est-elle cruciale ?

Sécurité

Vous pouvez réexécuter vos playbooks sans craindre d'endommager votre infrastructure ou de créer des doublons

Cohérence

L'état final de vos systèmes est prévisible et conforme à votre configuration, quel que soit l'état initial

Résilience

En cas d'échec partiel, relancer le playbook corrige automatiquement les problèmes sans intervention manuelle

Simplicité

Pas besoin de logique conditionnelle complexe pour vérifier si une action a déjà été effectuée

Concrètement, si votre playbook spécifie qu'un paquet doit être installé, Ansible vérifiera d'abord s'il est déjà présent. Si c'est le cas, aucune action ne sera entreprise. Cette approche intelligente économise du temps et réduit les risques d'erreurs lors des déploiements répétés.

Comparaison avec les Alternatives

Pour contextualiser la valeur d'Ansible, comparons-le avec les autres outils majeurs de gestion de configuration sur des critères clés qui impactent le quotidien des équipes DevOps.

Ansible vs Puppet : Philosophies Différentes

Puppet

Architecture : Basée sur des agents installés sur chaque nœud géré, avec un serveur maître central

Langage : Utilise un DSL (Domain Specific Language) basé sur Ruby, nécessitant un apprentissage spécifique

Modèle : Pull - les agents récupèrent périodiquement leur configuration depuis le maître

Courbe d'apprentissage : Raide, particulièrement pour les non-développeurs

Cas d'usage idéaux : Grandes infrastructures avec des politiques de conformité strictes et des besoins de reporting avancés

Puppet excelle dans les environnements d'entreprise où la gouvernance centralisée et les rapports détaillés de conformité sont prioritaires. Son modèle pull garantit que les machines restent conformes même si elles sont déconnectées temporairement.

Ansible

Architecture : Sans agent, utilise SSH natif pour la communication

Langage : YAML simple et lisible, accessible aux débutants

Modèle : Push - le contrôleur envoie les configurations quand nécessaire

Courbe d'apprentissage : Douce, démarrage possible en quelques heures

Cas d'usage idéaux : Automatisation rapide, orchestration complexe, déploiements d'applications, environnements cloud dynamiques

Ansible brille par sa simplicité de déploiement et son agilité. Parfait pour les équipes qui veulent des résultats rapides sans investir des semaines dans la formation. Le modèle push offre un contrôle immédiat et transparent.

Ansible vs Chef : Approches Contrastées

Points de Comparaison

- Infrastructure de déploiement
- Complexité de configuration
- Besoins en expertise
- Temps de mise en œuvre
- Flexibilité d'orchestration

Chef adopte une approche orientée développeur avec Ruby comme langage principal. Les "recipes" et "cookbooks" de Chef offrent une puissance remarquable mais exigent des compétences en programmation substantielles. L'infrastructure Chef Server/Client nécessite une planification et une maintenance continues.

Ansible privilégie la simplicité opérationnelle. Ses playbooks YAML sont immédiatement compréhensibles et modifiables par les équipes ops sans background de développement. L'absence de serveur central élimine un point de défaillance unique et simplifie l'architecture globale.

Verdict pratique : Chef convient aux organisations avec des équipes de développement fortes qui valorisent la programmabilité totale. Ansible s'impose pour les équipes ops qui veulent automatiser rapidement sans infrastructure lourde. Pour la plupart des cas d'usage, la simplicité d'Ansible l'emporte sur la puissance brute de Chef.

Ansible vs SaltStack : La Rapidité en Question

SaltStack se positionne comme l'alternative haute performance à Ansible, et cette affirmation mérite d'être examinée avec nuance.

Performance de SaltStack

SaltStack utilise ZeroMQ pour la communication, permettant une exécution parallèle ultra-rapide sur des milliers de nœuds simultanément. Pour les déploiements massifs et les réactions en temps réel, SaltStack montre effectivement des performances supérieures.

Performance d'Ansible

Ansible a considérablement amélioré ses performances avec les stratégies parallèles et le mode "mitogen". Pour la majorité des infrastructures (jusqu'à quelques centaines de serveurs), la différence de vitesse est négligeable dans la pratique quotidienne.

Le Compromis Réel

SaltStack offre plus de vitesse au prix d'une complexité accrue : agents à maintenir, architecture master/minion à gérer. Ansible privilégie la simplicité avec des performances largement suffisantes pour 95% des cas d'usage réels.

La question n'est pas "quel outil est le plus rapide" mais "quelle vitesse avez-vous réellement besoin ?". Pour orchestrer 50 serveurs web, la différence entre 30 secondes et 15 secondes est négligeable comparée au temps économisé sur la maintenance et la formation.

Les Avantages Décisifs d'Ansible

Démarrage Rapide

Installation en 5 minutes, premiers résultats en 30 minutes. Aucun outil concurrent n'offre une courbe d'adoption aussi rapide pour les équipes.

Accessibilité Universelle

YAML lisible par tous : développeurs, ops, managers. La documentation est exemplaire et la communauté très active pour aider les débutants.

Maintenance Minimale

Pas d'agents à mettre à jour, pas de serveur central à maintenir. Moins de pièces mobiles signifie moins de points de défaillance et de maintenance.

Polyvalence Exceptionnelle

Configuration, déploiement, orchestration, provisioning cloud. Un seul outil pour tous vos besoins d'automatisation infrastructure.

Écosystème Riche

Plus de 5000 modules communautaires, intégrations avec tous les clouds majeurs, support commercial Red Hat disponible si nécessaire.

Évolutivité Progressive

Commencez petit avec des scripts ad-hoc, évoluez vers des playbooks complexes, puis adoptez Ansible Tower/AWX pour l'entreprise sans changer d'outil.

CHAPITRE 3

Architecture Ansible : Vue d'Ensemble

Comprendre l'architecture d'Ansible est essentiel pour l'utiliser efficacement et concevoir des solutions d'automatisation robustes et maintenables. L'architecture d'Ansible se distingue par sa simplicité conceptuelle tout en offrant une flexibilité remarquable pour des cas d'usage complexes.

L'architecture repose sur quelques composants clés qui interagissent de manière élégante : un nœud de contrôle central d'où partent toutes les opérations, des nœuds gérés qui exécutent les tâches, un inventaire qui décrit votre infrastructure, des playbooks qui définissent vos automatisations, et des modules qui accomplissent les actions concrètes.

Cette séparation claire des responsabilités permet une scalabilité horizontale naturelle et une organisation logique du code d'automatisation. Explorons maintenant chaque composant en détail.

Le Nœud de Contrôle (Control Node)

Le **nœud de contrôle** est la machine depuis laquelle vous exécutez Ansible. C'est votre poste de commande central pour orchestrer toute votre infrastructure automatisée.

Caractéristiques et prérequis :

Système d'exploitation : Linux ou macOS uniquement (Ansible ne fonctionne pas nativement sur Windows comme contrôleur, bien qu'il puisse gérer des machines Windows)

Python : Version 3.8 ou supérieure recommandée pour les meilleures performances et le support complet

Connectivité réseau : Doit pouvoir atteindre tous les nœuds gérés via SSH (port 22 par défaut) ou WinRM pour Windows

Stockage : Espace pour les playbooks, rôles, inventaires et logs (généralement quelques Go suffisent)

Bonnes pratiques :

- Utilisez une machine dédiée ou une VM pour les environnements de production
- Implémentez le contrôle de version (Git) pour tous vos playbooks et inventaires
- Sécurisez l'accès au nœud de contrôle car il possède les clés SSH de votre infrastructure

Points Clés

Un seul nœud de contrôle peut gérer des milliers de machines simultanément grâce au parallélisme intégré.

Vous pouvez avoir plusieurs nœuds de contrôle pour différentes équipes ou environnements (dev, staging, prod).

Le nœud de contrôle ne nécessite aucun privilège root, sauf si vous souhaitez gérer des configurations système locales.

Il est possible d'exécuter Ansible depuis un conteneur Docker pour une portabilité maximale.

Les Nœuds Gérés (Managed Nodes)

Les **nœuds gérés** sont les machines cibles que vous automatisez avec Ansible : serveurs physiques, machines virtuelles, instances cloud, conteneurs, équipements réseau compatibles, etc.

Linux/Unix

Prérequis minimaux :

- Python 2.7 ou 3.5+ installé
- Serveur SSH actif et accessible
- Compte utilisateur avec privilèges sudo (pour les tâches système)

Windows

Configuration requise :

- PowerShell 3.0 ou supérieur
- WinRM configuré et activé
- .NET Framework 4.0+
- Compte avec droits administrateur

Réseau

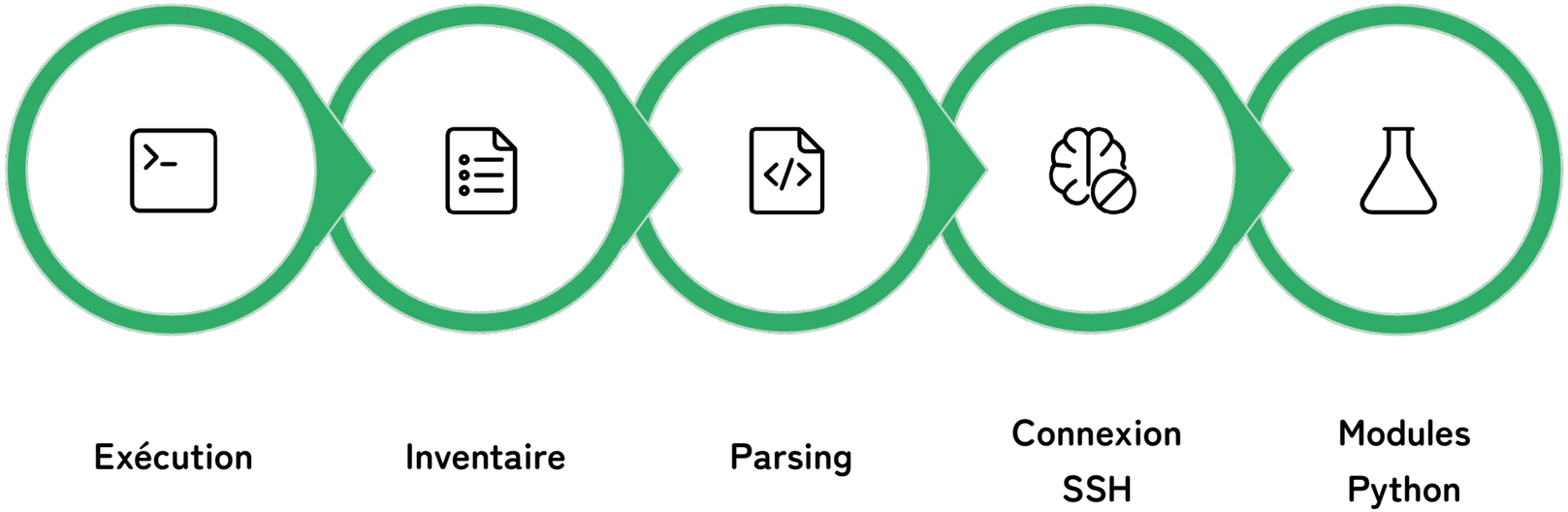
Équipements supportés :

- Cisco IOS, NX-OS, ASA
- Juniper Junos
- Arista EOS
- Configuration via SSH ou API

Un avantage majeur : les nœuds gérés n'ont besoin d'aucun logiciel spécifique Ansible. Si vous pouvez vous connecter en SSH à une machine Linux standard, Ansible peut la gérer immédiatement. Cette caractéristique agentless élimine le problème classique du "bootstrap" présent dans d'autres outils.

Le Flux d'Exécution Complet

Comprendre comment Ansible orchestre l'exécution des tâches est crucial pour déboguer les problèmes et optimiser les performances. Voici le parcours détaillé d'une commande Ansible de bout en bout.



Ce flux révèle plusieurs points d'optimisation : la phase de connexion SSH peut être accélérée avec ControlPersist, le transfert de modules peut être optimisé avec pipelining, et l'exécution parallèle peut être ajustée via le paramètre "forks" pour équilibrer vitesse et charge réseau.

Schéma d'Architecture Détaillé

Visualisons maintenant l'architecture complète d'un environnement Ansible typique, avec tous ses composants et leurs interactions.

Les Inventaires : Le Carnet d'Adresses d'Ansible

L'**inventaire** (inventory) est l'un des concepts fondamentaux d'Ansible. Il définit et organise les machines que vous souhaitez gérer. Sans inventaire, Ansible ne sait pas sur quelles machines exécuter vos tâches.

Les inventaires peuvent prendre plusieurs formes :

1

Inventaires Statiques

Fichiers INI ou YAML listant explicitement les hôtes et leurs groupes. Parfaits pour les infrastructures stables avec peu de changements.

2

Inventaires Dynamiques

Scripts ou plugins qui interrogent des sources externes (AWS, Azure, VMware, etc.) pour générer l'inventaire à la volée. Essentiels pour les environnements cloud dynamiques.

3

Inventaires Hybrides

Combinaison des deux approches, permettant de mélanger des hôtes statiques avec des sources dynamiques pour une flexibilité maximale.

L'inventaire supporte également des variables d'hôte et de groupe, permettant de personnaliser le comportement d'Ansible pour différents ensembles de machines. Cette flexibilité est cruciale pour gérer efficacement des infrastructures hétérogènes.

Les Playbooks : Le Cœur de l'Automatisation

Les **playbooks** sont les fichiers YAML où vous définissez vos automatisations. Ils décrivent l'état désiré de votre infrastructure de manière déclarative et lisible.

Structure d'un playbook :

Plays : Sections qui ciblent un groupe d'hôtes spécifique
Tasks : Actions individuelles à exécuter (appeler des modules)
Handlers : Tâches spéciales qui s'exécutent uniquement si déclenchées
Variables : Paramètres réutilisables dans le playbook
Roles : Regroupements réutilisables de tâches, variables et fichiers

Les playbooks peuvent être simples (quelques lignes) ou complexes (plusieurs centaines de tâches organisées en rôles). La philosophie reste la même : décrire l'état final souhaité plutôt que les étapes pour y parvenir.

Exemple de Playbook Structuré

```
---
- name: Configuration serveurs web
 hosts: webservers
 become: yes

 vars:
 http_port: 80
 max_clients: 200

 tasks:
 - name: Installer Apache
 apt:
 name: apache2
 state: latest
 notify: Redémarrer Apache

 - name: Copier configuration
 template:
 src: httpd.conf.j2
 dest: /etc/apache2/apache2.conf
 notify: Redémarrer Apache

 handlers:
 - name: Redémarrer Apache
 service:
 name: apache2
 state: restarted
```

Les Modules : Les Outils d'Action

Les **modules** sont les unités de travail d'Ansible. Chaque module accomplit une tâche spécifique : installer un paquet, copier un fichier, gérer un service, créer un utilisateur, etc.

Ansible inclut plus de 3000 modules couvrant virtuellement tous les besoins d'automatisation :

Gestion de Paquets

apt, yum, dnf, pip, npm, etc. Installation et mise à jour de logiciels sur différentes distributions.

Gestion de Fichiers

copy, template, file, lineinfile. Manipulation de fichiers et configuration avec templates Jinja2.

Gestion de Services

service, systemd, supervisorctl. Contrôle de démons et services système.

Gestion d'Utilisateurs

user, group, authorized_key. Administration des comptes et permissions.

Cloud & Containers

ec2, azure_rm, docker, kubernetes. Provisioning et orchestration d'infrastructure cloud.

Réseau

ios_config, nxos_command. Configuration d'équipements réseau Cisco, Juniper, etc.

Chaque module est idempotent et retourne des informations structurées sur son exécution, permettant une automatisation fiable et traçable.

CHAPITRE 4

Installation d'Ansible

L'installation d'Ansible est remarquablement simple comparée à d'autres outils d'automatisation. Pas d'infrastructure complexe à déployer, pas de base de données à configurer : quelques commandes suffisent pour être opérationnel.

Ansible peut être installé de plusieurs manières selon votre environnement et vos préférences. Nous allons explorer les méthodes les plus courantes, leurs avantages respectifs, et les prérequis à vérifier avant de commencer.

bstorm TRAINING

Prérequis Système

Configuration Minimale

OS : Linux (Ubuntu, RHEL, CentOS, Debian, etc.) ou macOS

Python : Version 3.8+ recommandée

Mémoire : 512 MB minimum, 2 GB recommandé

Disque : 2 GB d'espace libre

Réseau : Connectivité vers les hôtes cibles

Vérification des prérequis Python :

```
python3 --version
# Doit afficher Python 3.8 ou supérieur

python3 -m pip --version
# Vérifie que pip est disponible
```

Installation de Python si nécessaire :

```
# Ubuntu/Debian
sudo apt update
sudo apt install python3 python3-pip

# RHEL/CentOS
sudo yum install python3 python3-pip

# macOS (avec Homebrew)
brew install python3
```

Python est généralement déjà présent sur les systèmes Linux modernes. Sur macOS, Homebrew facilite grandement l'installation et la gestion des versions Python.

Installation via les Gestionnaires de Paquets

La méthode la plus simple et recommandée pour la plupart des utilisateurs est d'utiliser le gestionnaire de paquets natif de votre distribution Linux.

Ubuntu / Debian

```
sudo apt update
sudo apt install ansible -y

# Vérifier l'installation
ansible --version
```

RHEL / CentOS

```
# Activer EPEL repository
sudo yum install epel-release -y

sudo yum install ansible -y

ansible --version
```

Fedora

```
sudo dnf install ansible -y

# Ansible est dans les repos officiels
ansible --version
```

Avantages de cette méthode :

- Installation simplifiée en une seule commande
- Mises à jour automatiques via le système de paquets
- Dépendances gérées automatiquement
- Version stable testée pour votre distribution

Inconvénient : La version disponible peut être légèrement plus ancienne que la dernière version upstream. Pour la dernière version, utilisez pip (méthode suivante).

Installation via pip (Recommandé pour les Dernières Versions)

Pour obtenir la toute dernière version d'Ansible ou installer dans un environnement virtuel Python, pip est la méthode privilégiée par la communauté Ansible.

Installation globale :

```
# Installation standard
sudo pip3 install ansible

# Ou pour l'utilisateur courant uniquement
pip3 install --user ansible

# Vérifier l'installation
ansible --version
```

Installation dans un environnement virtuel (recommandé) :

```
# Créer un environnement virtuel
python3 -m venv ~/ansible-venv

# Activer l'environnement
source ~/ansible-venv/bin/activate

# Installer Ansible
pip install ansible

# L'environnement isolé évite les conflits
```

Installation d'une version spécifique :

```
# Version exacte
pip install ansible==2.15.0

# Version minimum
pip install 'ansible>=2.14'
```

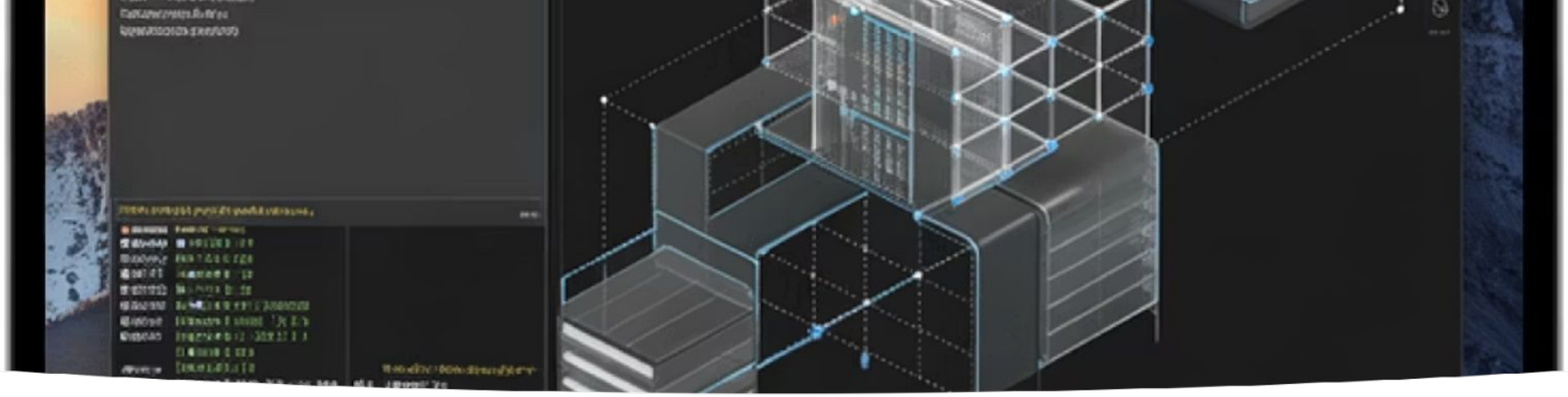
Mise à jour d'Ansible :

```
pip install --upgrade ansible
```

Avantages de pip :

- Dernières versions disponibles immédiatement
- Contrôle précis des versions
- Environnements virtuels pour l'isolation
- Installations multiples possibles (différentes versions)

Note importante : À partir d'Ansible 2.10, le paquet est divisé en `ansible-core` (moteur minimal) et `ansible` (core + collections). Pour la plupart des utilisateurs, installez le paquet `ansible` complet.



Installation sur macOS

Les utilisateurs macOS ont plusieurs options pour installer Ansible, Homebrew étant la plus populaire et pratique.

01

Installer Homebrew (si nécessaire)

```
/bin/bash -c "$(curl -fsSL \

https://raw.githubusercontent.com/Homebrew
/install/HEAD/install.sh)"
```

02

Installer Ansible via Homebrew

```
brew install ansible

# Vérifier l'installation
ansible --version
```

03

Alternative : Installation via pip

```
# macOS inclut Python 3
pip3 install --user ansible

# Ajouter au PATH si nécessaire
export
PATH="$HOME/Library/Python/3.x/bin:$PATH"
```

Homebrew gère automatiquement les dépendances et les mises à jour, ce qui en fait la méthode privilégiée sur macOS pour la plupart des utilisateurs.

Configuration Post-Installation

Une fois Ansible installé, quelques étapes de configuration amélioreront votre expérience quotidienne.

Fichier de Configuration Ansible

Créer un fichier `ansible.cfg` personnalisé :

```
# ~/.ansible.cfg ou ./ansible.cfg
[defaults]
inventory = ./inventory
host_key_checking = False
retry_files_enabled = False
gathering = smart
timeout = 30
forks = 10

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False

[ssh_connection]
pipelining = True
control_path = /tmp/ansible-ssh-%%h-%%p-%%r
```

Ces paramètres optimisent les performances et désactivent certaines vérifications strictes pour faciliter les premiers pas.

Configuration des Clés SSH

Pour une connexion sans mot de passe :

```
# Générer une paire de clés
ssh-keygen -t ed25519 -C "ansible"

# Copier la clé publique vers les hôtes
ssh-copy-id user@host1
ssh-copy-id user@host2

# Tester la connexion
ssh user@host1 'echo Connexion OK'
```

Variables d'environnement utiles :

```
# Ajouter à ~/.bashrc ou ~/.zshrc
export ANSIBLE_CONFIG=~/ansible.cfg
export ANSIBLE_INVENTORY=~/inventory
export ANSIBLE_NOCOWS=1 # Désactive les vaches ASCII
```

Vérification de l'Installation

Après l'installation, vérifiez que tout fonctionne correctement avec ces commandes de diagnostic.

Version et Configuration

```
ansible --version

# Affiche :
# - Version d'Ansible
# - Fichier de configuration utilisé
# - Emplacement des modules
# - Version de Python
# - Répertoire exécutable
```

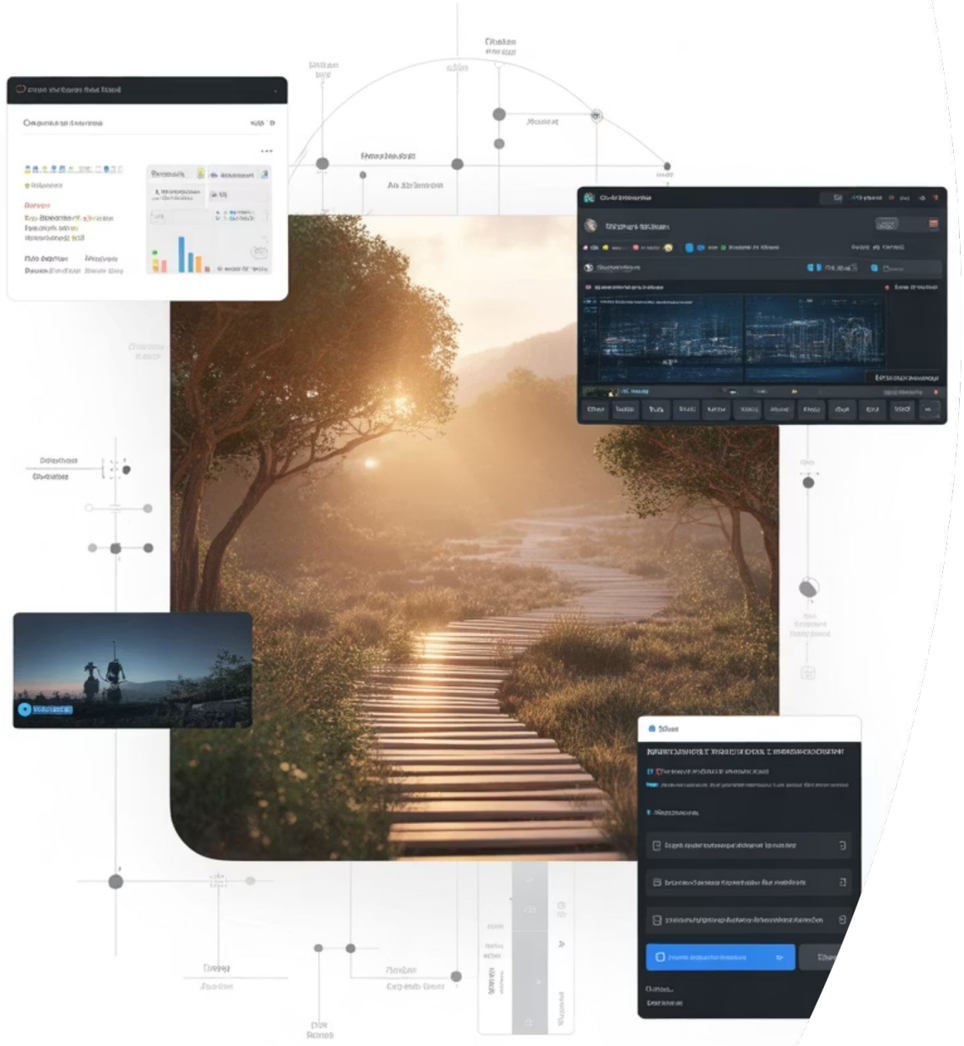
Liste des Modules Disponibles

```
ansible-doc -l | head -20

# Voir la doc d'un module spécifique
ansible-doc apt
ansible-doc copy
ansible-doc service
```

Test de Connectivité Local

```
# Tester sur localhost
ansible localhost -m ping

# Devrait retourner :
# localhost | SUCCESS => {
#     "ping": "pong"
# }
```

Si toutes ces commandes fonctionnent sans erreur, votre installation Ansible est opérationnelle et vous êtes prêt à commencer l'automatisation !



CHAPITRE 5

Premiers Pas avec Ansible

Maintenant qu'Ansible est installé, passons à la pratique avec des exemples concrets. Cette section vous guidera à travers la création de votre premier inventaire et l'exécution de vos premières commandes Ansible.

Nous allons adopter une approche progressive : d'abord des commandes ad-hoc simples pour comprendre les concepts de base, puis nous construirons vers des playbooks plus sophistiqués. L'objectif est d'obtenir des résultats rapidement tout en comprenant les fondamentaux.

Structure du Fichier d'Inventaire Statique

L'inventaire est le point de départ de toute automation Ansible. Il définit les machines que vous allez gérer et comment les organiser logiquement.

Format INI (Classique)

Le format INI est simple et lisible, parfait pour débuter :

```
# inventory.ini

# Serveurs individuels
web1.example.com
web2.example.com

# Groupe de serveurs web
[webservers]
web1.example.com
web2.example.com
web3.example.com

# Groupe de bases de données
[databases]
db1.example.com ansible_host=192.168.1.10
db2.example.com ansible_host=192.168.1.11

# Groupe avec variables
[appservers]
app1.example.com
app2.example.com

[appservers:vars]
ansible_user=deploy
ansible_port=2222

# Groupe de groupes
[production:children]
webservers
databases
appservers
```

Format YAML (Moderne)

YAML offre plus de flexibilité et de structure :

```
# inventory.yml
all:
 children:
 webservers:
 hosts:
 web1.example.com:
 web2.example.com:
 web3.example.com:
 vars:
 http_port: 80

 databases:
 hosts:
 db1.example.com:
 ansible_host: 192.168.1.10
 db2.example.com:
 ansible_host: 192.168.1.11
 vars:
 db_port: 5432

 appservers:
 hosts:
 app1.example.com:
 app2.example.com:
 vars:
 ansible_user: deploy
 ansible_port: 2222
```

Variables et Paramètres d'Inventaire

Les inventaires Ansible supportent de nombreuses variables pour personnaliser le comportement de connexion et d'exécution.

Connexion

`ansible_host` : Adresse IP ou hostname réel

`ansible_port` : Port SSH (défaut: 22)

`ansible_user` : Utilisateur de connexion

`ansible_connection` : Type (ssh, local, docker, etc.)

Authentification

`ansible_ssh_private_key_file` : Chemin vers clé SSH

`ansible_password` : Mot de passe (déconseillé)

`ansible_become` : Activer sudo (true/false)

`ansible_become_user` : Utilisateur sudo cible

Python

`ansible_python_interpreter` : Chemin vers Python

- Utile si Python est dans un emplacement non-standard

Exemple : `/usr/bin/python3.9`

Exemple d'hôte avec paramètres complets :

```
server1.example.com ansible_host=10.0.1.50 ansible_port=2222 ansible_user=admin
ansible_python_interpreter=/usr/bin/python3
```

Création de Votre Premier Inventaire

Créons ensemble un inventaire pratique pour un environnement type application web trois-tiers.

Architecture Cible

Notre infrastructure d'exemple :

- 2 serveurs web (nginx)
- 2 serveurs d'application (Node.js)
- 1 serveur de base de données (PostgreSQL)
- 1 serveur de monitoring (Prometheus)

Tous les serveurs sont dans le même réseau local et accessibles via SSH sur le port standard.

Sauvegardez ce fichier comme `inventory.ini` dans votre répertoire de travail Ansible.

inventory.ini

```
# Serveurs web frontend
[web]
web1.local ansible_host=192.168.1.10
web2.local ansible_host=192.168.1.11

[web:vars]
nginx_port=80
ansible_user=webadmin

# Serveurs applicatifs
[app]
app1.local ansible_host=192.168.1.20
app2.local ansible_host=192.168.1.21

[app:vars]
app_port=3000
node_env=production

# Base de données
[database]
db1.local ansible_host=192.168.1.30

[database:vars]
postgres_port=5432
ansible_user=dbadmin

# Monitoring
[monitoring]
monitor.local ansible_host=192.168.1.40

# Tous les serveurs de production
[production:children]
web
app
database
monitoring

[production:vars]
environment=prod
ansible_become=true
```

Commandes Ad-Hoc : Introduction

Les **commandes ad-hoc** sont des commandes Ansible one-liner qui vous permettent d'exécuter des tâches rapides sans écrire de playbook. Elles sont parfaites pour :

- Tester la connectivité vers vos hôtes
- Effectuer des vérifications rapides
- Exécuter des commandes ponctuelles
- Collecter des informations système
- Gérer des services d'urgence

Syntaxe générale d'une commande ad-hoc :

```
ansible <pattern> -i <inventory> -m <module> -a "<arguments>"
```

Où :

`<pattern>` : Cible (groupe ou hôte) de l'inventaire

`-i <inventory>` : Fichier d'inventaire (optionnel si défini dans ansible.cfg)

`-m <module>` : Module Ansible à utiliser

`-a "<arguments>"` : Arguments à passer au module

Première Commande : Le Module Ping

Le module `ping` est traditionnellement la première commande qu'on exécute avec Ansible. Il vérifie la connectivité et l'authentification.

01

Tester tous les hôtes

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

# Résultat attendu pour chaque hôte :
# web1.local | SUCCESS => {
#     "changed": false,
#     "ping": "pong"
# }
```

02

Tester un groupe spécifique

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

# Teste uniquement les serveurs du
groupe [web]
```

03

Tester un hôte individuel

```
ansible web1.local -i inventory.ini -m
ping

# Teste un seul serveur
```

Interprétation des résultats :

`SUCCESS` : L'hôte est accessible, Python fonctionne, authentification OK

`UNREACHABLE` : Problème de connexion réseau ou SSH

`FAILED` : Connexion établie mais problème d'exécution (souvent Python manquant)

Note importante : Le module ping d'Ansible n'est PAS un ping ICMP. Il établit une connexion SSH complète et vérifie que Python peut s'exécuter.

Commandes Ad-Hoc Essentielles

Explorons les commandes ad-hoc les plus utiles pour la gestion quotidienne de votre infrastructure.

Exécuter une Commande Shell

```
# Vérifier l'uptime
ansible all -m command -a "uptime"

# Vérifier l'espace disque
ansible web -m command -a "df -h"

# Lister les processus
ansible database -m shell -a "ps aux
| grep postgres"
```

Note : `command` n'interprète pas les pipes/redirections. Utilisez `shell` pour ça.

Gérer des Fichiers

```
# Copier un fichier
ansible web -m copy -a
"src=/tmp/index.html
dest=/var/www/html/"

# Créer un répertoire
ansible all -m file -a "path=/opt/app
state=directory mode=0755"

# Supprimer un fichier
ansible app -m file -a
```

Gérer des Paquets

```
# Installer un paquet (Ubuntu/Debian)
ansible web -m apt -a "name=nginx
state=present" --become

# Mettre à jour tous les paquets
ansible all -m apt -a "upgrade=dist"
--become

# Installer plusieurs paquets
ansible app -m apt -a
"name=nodejs,npm state=latest"
```

Gérer des Services

```
# Démarrer un service
ansible web -m service -a "name=nginx
state=started" --become

# Redémarrer un service
ansible app -m service -a
"name=nodejs state=restarted"
--become

# Activer au démarrage
ansible database -m service -a
```

Gérer des Utilisateurs

```
# Créer un utilisateur
ansible all -m user -a "name=deploy shell=/bin/bash" --become

# Ajouter une clé SSH
ansible all -m authorized_key -a "user=deploy key='{{lookup('file',
'~/.ssh/id_rsa.pub')}}'" --become
```

Collecter des Informations

```
# Récupérer tous les facts
ansible web1.local -m setup

# Filtrer les facts réseau
ansible web -m setup -a "filter=ansible_eth*"

# Facts personnalisés uniquement
ansible all -m setup -a "gather_subset=!all,network"
```

Options Avancées des Commandes Ad-Hoc

Les commandes ad-hoc supportent de nombreuses options pour contrôler précisément leur comportement.

Options de connexion :

```
# Spécifier un utilisateur
ansible all -m ping -u admin

# Utiliser sudo
ansible all -m command -a "cat /etc/shadow" --become

# Spécifier l'utilisateur sudo
ansible all -m command -a "whoami" --become --become-user=postgres

# Demander le mot de passe sudo
ansible all -m ping --become --ask-become-pass
```

Options de performance :

```
# Exécution parallèle (10 hôtes à la fois)
ansible all -m ping -f 10

# Timeout personnalisé
ansible all -m ping -T 30
```

Options de débogage :

```
# Mode verbeux (1 à 4 'v')
ansible all -m ping -v
ansible all -m ping -vvv

# Vérifier sans exécuter
ansible all -m command -a "reboot" --check

# Mode diff (montre les changements)
ansible all -m copy -a "src=file dest=/tmp/" --diff
```

Options de ciblage :

```
# Limiter à certains hôtes
ansible all -m ping --limit web1.local

# Exclure des hôtes
ansible all -m ping --limit '!database'

# Pattern complexe
ansible 'web*:&production' -m ping
```

Exercice Pratique : Votre Première Automation

Mettons en pratique ce que nous avons appris avec un exercice complet de bout en bout.

Objectif : Préparer un nouveau serveur web en installant et configurant nginx, puis déployer une page d'accueil personnalisée.

Étape 1 : Vérifier la

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

Étape 2 : Mettre à jour le

```
ansible web -m apt -a "update_cache=yes" --become -i inventory.ini
```

Étape 3 : Installer nginx

```
ansible web -m apt -a "name=nginx state=present" --become -i
inventory.ini
```

Étape 4 : Démarrer le

```
ansible web -m service -a "name=nginx state=started enabled=yes"
--become -i inventory.ini
```

Étape 5 : Déployer une page HTML

```
echo "<h1>Serveur géré par Ansible</h1>" > /tmp/index.html
ansible web -m copy -a "src=/tmp/index.html
dest=/var/www/html/index.html" --become -i inventory.ini
```

Étape 6 : Vérifier le

```
ansible web -m uri -a "url=http://localhost return_content=yes"
```

Félicitations ! Vous venez d'automatiser le déploiement complet d'un serveur web avec Ansible en quelques commandes.

Bonnes Pratiques pour Débuter

Organisation des Fichiers

Créez une structure de répertoires claire dès le début : inventaires séparés par environnement, playbooks regroupés par fonction, variables externalisées.

Versionner Tout

Utilisez Git pour tous vos playbooks, inventaires et configurations. Cela permet la collaboration, la traçabilité et les rollbacks en cas de problème.

Tester en Dev d'Abord

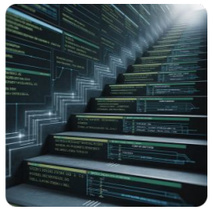
Toujours tester vos playbooks sur un environnement de développement ou staging avant de les exécuter en production. Utilisez --check pour simuler.

Documenter

Ajoutez des commentaires dans vos playbooks et maintenant un README expliquant la structure et l'usage. Votre futur vous dira merci.

Gérer les Secrets

Ne jamais commiter de mots de passe en clair. Utilisez Ansible Vault pour chiffrer les données sensibles ou des solutions externes comme HashiCorp Vault.



Commencer Petit

Ne tentez pas d'automatiser toute votre infrastructure d'un coup. Commencez par des tâches simples et répétitives, puis montez en complexité progressivement.

Ressources pour Aller Plus Loin

Documentation Officielle

docs.ansible.com : Documentation complète et à jour

Module Index : Référence exhaustive de tous les modules

Best Practices : Guide officiel des bonnes pratiques

Communauté

Ansible Galaxy : galaxy.ansible.com - Rôles communautaires

Forums : groups.google.com/g/ansible-project

Reddit : r/ansible - Discussions et aide

IRC/Matrix : #ansible sur Libera.Chat

Formation Continue

Red Hat Training : Formations officielles certifiantes

Ansible Fest : Conférence annuelle

YouTube : Nombreux tutoriels vidéo

Livres Recommandés

- "Ansible for DevOps" par Jeff Geerling
- "Ansible: Up and Running" par Lorin Hochstein
- "Mastering Ansible" par James Freeman

Outils Complémentaires

Ansible Lint : Vérification de qualité des playbooks

Molecule : Framework de test pour rôles Ansible

Ansible Tower/AWX : Interface web et orchestration entreprise

Semaphore : Alternative open-source à Tower

Projets Pratiques

Rien ne vaut la pratique ! Automatisez votre infrastructure personnelle, contribuez à des projets open-source, ou créez vos propres rôles sur Galaxy.