

Ansible for Teenagers

**CONVIENT AUSSI AUX
ADULTES**



Introduction



Ansible est un outil open source d'automatisation, de configuration et de gestion d'infrastructures informatiques. Il permet de gérer facilement de nombreux serveurs en utilisant le langage YAML pour décrire les tâches automatisées. Ansible fonctionne sans agent, ce qui signifie qu'il n'est pas nécessaire d'installer un logiciel supplémentaire sur les nœuds qu'il gère.

Ansible est populaire pour plusieurs raisons :

- Simplicité et facilité d'utilisation : Les playbooks (fichiers de configuration) sont écrits en YAML, un langage simple et lisible par l'homme.
- Puissant et flexible : Ansible peut gérer des tâches complexes et s'adapter à divers environnements.
- Agentless : Pas besoin d'installer de logiciel sur les nœuds gérés, ce qui simplifie la gestion et la sécurité.
- Idempotent : Les playbooks peuvent être exécutés plusieurs fois sans effets indésirables ou non souhaités.
- Large communauté et support : Ansible bénéficie d'une vaste communauté et d'une abondance de ressources et de modules.

Installation d'Ansible



```
● ● ●

# Installation d'Ansible sur une distribution basée sur
# Debian (comme Ubuntu)
sudo apt update && sudo apt install ansible

# Installation d'Ansible sur une distribution basée sur Red
# Hat
sudo yum install ansible

# Installation d'Ansible sur macOS en utilisant Homebrew
brew install ansible

# Pas de Windows, utilisez WSL
```

Inventaire



```
● ● ●  
[serveurs]  
serveur1 ansible_host=192.168.64.3
```

Le fichier d'inventaire est un document de configuration utilisé par Ansible pour énumérer et organiser les hôtes et groupes de serveurs sur lesquels les tâches et playbooks seront exécutés.

Ansible communique avec les nœuds distants en utilisant SSH pour les systèmes Unix/Linux et WinRM pour les systèmes Windows, permettant ainsi une gestion sécurisée et efficace des configurations et des automatisations à distance.

Ansible Ping



```
● ● ●
```

```
# Test de la connexion aux hôtes dans l'inventaire
# 'ansible' est la commande de base, 'all' désigne tous les
hôtes, '-i' spécifie le fichier d'inventaire, '-m ping'
utilise le module ping pour tester la connexion
ansible all -i inventory.ini -m ping
```

Cette commande envoie un ping à tous les hôtes définis dans `inventory.ini` pour vérifier que Ansible peut se connecter à eux correctement.

```
● ● ●
```

```
serveur1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Playbooks



Un playbook Ansible est un fichier YAML décrivant les tâches à exécuter sur les serveurs.

```
● ● ●  
—  
- hosts: serveurs  
  become: yes  
  tasks:  
    - name: Assurez-vous que Apache est installé  
      apt:  
        name: apache2  
        state: present
```

Ce playbook cible le groupe serveurs de votre fichier d'inventaire et exécute une tâche pour installer Apache sur les machines de ce groupe.

```
# Exécution d'un playbook Ansible  
# 'ansible-playbook' est la commande pour exécuter des  
playbooks, '-i' spécifie le fichier d'inventaire,  
'mon_playbook.yml' est le playbook à exécuter  
ansible-playbook -i inventory.ini mon_playbook.yml
```

Tasks



Les tâches sont les unités de base de l'action dans un playbook Ansible. Elles définissent ce que vous voulez réaliser sur les hôtes ciblés.

```
—  
- hosts: serveurs  
  become: yes  
  tasks:  
    - name: Mettre à jour tous les paquets  
      apt:  
        update_cache: yes  
        upgrade: dist
```

Cette tâche met à jour les paquets sur un serveur Ubuntu en utilisant le module apt. Ansible dispose d'une large gamme de modules pour diverses tâches. Par exemple, pour gérer des fichiers, utilisez le module file.

```
—  
- hosts: serveurs  
  become: yes  
  tasks:  
    - name: Créer un répertoire  
      file:  
        path: /mon/dossier  
        state: directory
```

Variables



```
● ○ ●  
—  
http_port: 80  
max_clients: 200
```

```
● ○ ●  
—  
- hosts: serveurs  
  vars_files:  
    - vars.yaml  
  tasks:  
    - name: Afficher la valeur de la variable 'http_port'  
      debug:  
        msg: "Le port HTTP est {{ http_port }}"
```

```
# '-e' permet de passer des variables extra (extra-vars), ici  
on redéfinit 'http_port'.  
# Ne pas le mettre permet d'utiliser le fichier du playbook  
ansible-playbook mon_playbook.yml -i inventory.ini -e  
"http_port=8080"
```

Facts



Les facts dans Ansible sont des variables générées automatiquement contenant des informations sur les systèmes distants. Ils sont recueillis par Ansible à chaque fois qu'il se connecte à un hôte cible, fournissant des détails comme le système d'exploitation, les adresses IP, les disques disponibles, etc.

```
# Collecte et affichage des facts pour un hôte spécifique
ansible serveur1 -i inventory.ini -m setup
```

Les facts peuvent être utilisés dans vos playbooks pour conditionner l'exécution de tâches en fonction des caractéristiques de l'hôte.

```
- hosts: serveurs
  become: yes
  tasks:
    - name: Assurez-vous que Apache est installé
      apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"
```



Rôles

Les rôles dans Ansible organisent des tâches, des fichiers, des templates et des variables en unités logiques, facilitant ainsi la réutilisation et la gestion.

```
nom_du_role/
└── defaults/          # Variables par défaut du rôle
    └── main.yml        # Fichier de variables par défaut
└── vars/              # Autres variables du rôle
    └── main.yml        # Fichier de variables supplémentaires
└── tasks/             # Fichiers de tâches principaux
    └── main.yml        # Fichier principal de tâches
└── handlers/          # Gestionnaires pour les tâches
    └── main.yml        # Fichier principal des gestionnaires
└── templates/          # Templates Jinja2 pour la configuration
    └── template.j2       # Exemple de fichier template
└── files/              # Fichiers statiques requis par le rôle
    └── exemple.txt      # Exemple de fichier statique
└── meta/               # Métadonnées du rôle
    └── main.yml        # Fichier de metadata, avec dépendances
```

```
# Création d'un nouveau rôle avec ansible-galaxy
ansible-galaxy init nom_du_role
```

Ansible Vault



Ansible Vault est un outil intégré à Ansible qui permet de chiffrer des fichiers de données sensibles pour les sécuriser. Il est particulièrement utile pour gérer des informations sensibles telles que des mots de passe ou des clés secrètes dans vos playbooks, rôles, ou fichiers de variables.

```
● ● ●

# Création d'un secret - Vous serez invité à entrer le mot de passe du vault
ansible-vault create secret.yml

# Édition d'un secret
ansible-vault edit secret.yml

# Voir un secret
ansible-vault view secret.yml

# Déchiffrer un secret
ansible-vault decrypt secret.yml

# Chiffrer un secret
ansible-vault encrypt secret.yml

# Utiliser un playbook contenant un secret (Demandera le mot de passe)
ansible-playbook site.yml --ask-vault-pass

# Utiliser un playbook en passant le mot de passe en tant que fichier
ansible-playbook playbook.yml --vault-password-file /path/to/password_file
```

Ansible Vault - Utilisation



Pour utiliser des variables chiffrées dans vos playbooks, vous devez d'abord inclure le fichier de secrets chiffré. Utilisez la directive `vars_files` dans votre playbook pour spécifier le fichier chiffré.

```
● ● ●  
  
db_password: supersecretpassword123  
api_key: "123456789abcdef"
```

```
● ● ●  
  
—  
- hosts: all  
  vars_files:  
    - secrets.yml  
  
  tasks:  
    - name: Afficher la clé API  
      debug:  
        msg: "La clé API est {{ api_key }}"
```

Ansible Galaxy



Ansible Galaxy est une plateforme où la communauté partage des rôles réutilisables : <https://galaxy.ansible.com/ui/>

```
● ● ●

# Installer un rôle pour Nginx depuis Ansible Galaxy
ansible-galaxy install geerlingguy.nginx

# Installer un rôle pour MySQL depuis Ansible Galaxy
ansible-galaxy install geerlingguy.mysql
```

Ce playbook cible deux groupes d'hôtes : `serveurs_web` pour le rôle Nginx et `serveurs_db` pour le rôle MySQL.

```
—
- hosts: serveurs_web
  become: yes
  roles:
    - geerlingguy.nginx

- hosts: serveurs_db
  become: yes
  roles:
    - geerlingguy.mysql
```

Erreurs



Parfois, vous voudrez peut-être continuer l'exécution du playbook même si une tâche échoue. Vous pouvez utiliser `ignore_errors`.

```
hosts: serveurs
become: yes
tasks:
  - name: Tâche pouvant échouer
    command: /un/commande/qui/peut/echouer
    ignore_errors: yes
```

Vous pouvez contrôler les conditions sous lesquelles une tâche est considérée comme ayant échoué ou réussi en utilisant `failed_when` et `changed_when`. Ici si le mot "ERREUR" est présent, la tâche échoue.

```
hosts: serveurs
become: yes
tasks:
  - name: Exécution d'un script qui retourne toujours 0
    command: /mon/script.sh
    register: resultat_script
    failed_when: "'ERREUR' in resultat_script.stdout"
    changed_when: resultat_script.rc ≠ 0
```

Gestion des Erreurs



Les blocs permettent de grouper plusieurs tâches et de gérer les erreurs de manière collective.

Dans cet exemple, si une tâche dans le bloc `block` échoue, le bloc `rescue` est exécuté. Le bloc `always` s'exécute dans tous les cas, que le bloc `block` réussisse ou échoue.

```
- hosts: serveurs
  become: yes
  tasks:
    - name: Bloc de tâches
      block:
        - debug:
            msg: 'Je vais exécuter une tâche'
        - command: /une/commande/qui/peut/echouer
      rescue:
        - debug:
            msg: 'Une erreur est survenue lors de l'exécution
de la commande'
        - always:
            - debug:
                msg: 'Ce message s'affiche toujours après le
bloc, succès ou échec'
```

Retries / Until



Parfois, vous voudrez peut-être réessayer une tâche qui a échoué après un court délai.

```
hosts: serveurs
become: yes
tasks:
  - name: Réessayer une tâche jusqu'à ce qu'elle réussisse
    command: /une/commande/temporairement/instable
    register: resultat
    until: resultat.rc = 0
    retries: 5
    delay: 10
```

Ici, la tâche est répétée jusqu'à 5 fois avec un délai de 10 secondes entre chaque tentative, jusqu'à ce qu'elle réussisse.

Assertions



Utilisez des assertions pour effectuer des vérifications avant d'exécuter des tâches critiques.

```
—  
- hosts: serveurs  
  become: yes  
  tasks:  
    - name: Vérifier que la variable 'mon_parametre' est définie  
      assert:  
        that:  
          - mon_parametre is defined  
        fail_msg: "'mon_parametre' n'est pas défini"  
        success_msg: "'mon_parametre' est défini"
```

Cette tâche s'assure que la variable `mon_parametre` est définie avant de poursuivre.

Handlers



Les handlers sont des tâches spéciales qui s'exécutent en réponse à un changement. Par exemple, redémarrer un service après sa mise à jour.

```
● ● ●

- name: Installer un paquet
  apt:
    name: mon_paquet
    state: latest
  notify:
    - redémarrer mon_service

handlers:
  - name: redémarrer mon_service
    service:
      name: mon_service
      state: restarted
```



Debug

Utilisez le module `debug` pour afficher des valeurs de variables ou des messages personnalisés.

```
hosts: serveurs
become: yes
tasks:
  - name: Afficher la valeur d'une variable
    debug:
      var: ma_variable
```

Exécutez des playbooks en mode verbose (`-v`, `-vv`, ou `-vvv` pour plus de détails) pour obtenir des informations supplémentaires.

```
# Exécuter un playbook en mode verbose
ansible-playbook mon_playbook.yml -i inventory.ini -v
```

Molecule



Molecule fournit une structure pour tester les rôles Ansible de manière systématique, utilisant des conteneurs ou des machines virtuelles pour créer un environnement de test. Il permet de vérifier que vos rôles fonctionnent comme prévu sur différentes plateformes et configurations.

```
# Installation de Molecule avec le support Docker
pip install 'molecule[docker]'

# Initialiser un nouveau rôle avec Molecule
molecule init role -r nom_du_role
```

Molecule permet d'écrire des tests pour vérifier que les rôles Ansible fonctionnent comme prévu. Ces tests sont généralement écrits en utilisant le module `testinfra` ou `ansible` lui-même pour la phase de vérification.

```
molecule/
  └── default/
    ├── converge.yml
    ├── molecule.yml
    └── verify.yml
```

Molecule avec Testinfra



Testinfra est une bibliothèque Python qui sert à tester l'infrastructure de vos serveurs. Elle permet d'écrire des tests unitaires pour vérifier si les services, paquets, fichiers, etc., sont dans l'état attendu sur vos serveurs. Testinfra peut être utilisé avec Molecule pour tester des rôles Ansible, en s'assurant que le rôle applique la configuration désirée sur un serveur ou un conteneur.

```
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: instance-ubuntu
    image: "docker.io/library/ubuntu:20.04"
    pre_build_image: true
provisioner:
  name: ansible
  playbooks:
    converge: converge.yml
verifier:
  name: testinfra
  options:
    v: 2
  directory: ..tests/
```

molecule.yml

Molecule avec Testinfra



Molecule utilisant Testinfra comme vérificateur, les tests ne sont pas définis dans un fichier verify.yml mais plutôt dans des fichiers de test Python séparés, tels que test_default.py.

```
● ● ●

def test_nginx_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed

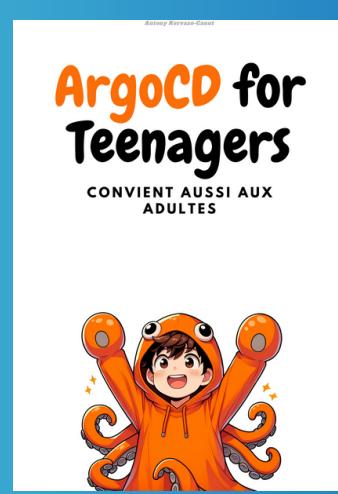
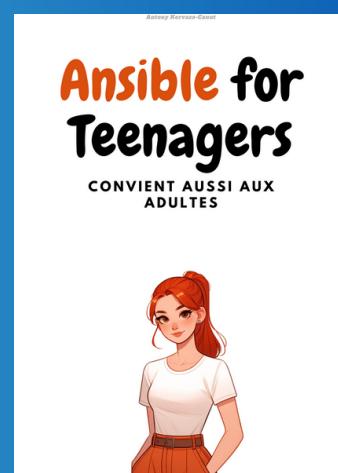
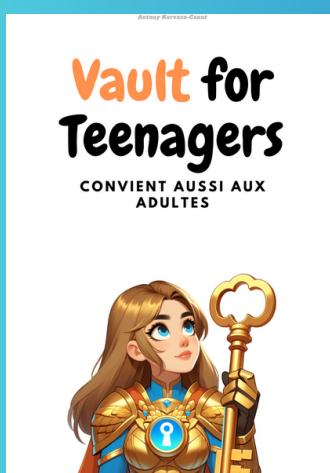
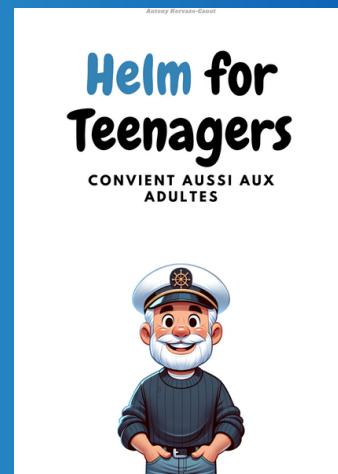
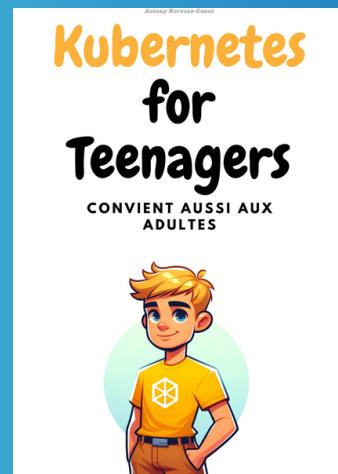
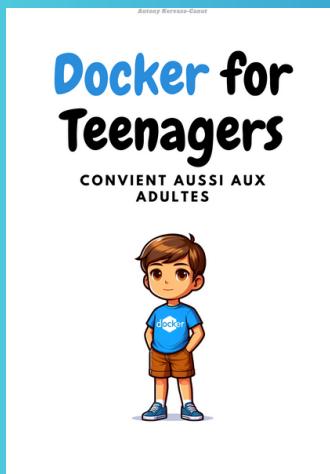
def test_nginx_running_and_enabled(host):
    nginx = host.service("nginx")
    assert nginx.is_running
    assert nginx.is_enabled
```

tests/test_default.py

Molecule va créer une instance Docker pour le test. Appliquer le rôle, exécuter les tests et détruire l'instance docker.

```
● ● ●
molecule test
```

Dans la même collection



ANTONYCANUT



ANTONY KERVAZO- CANUT

