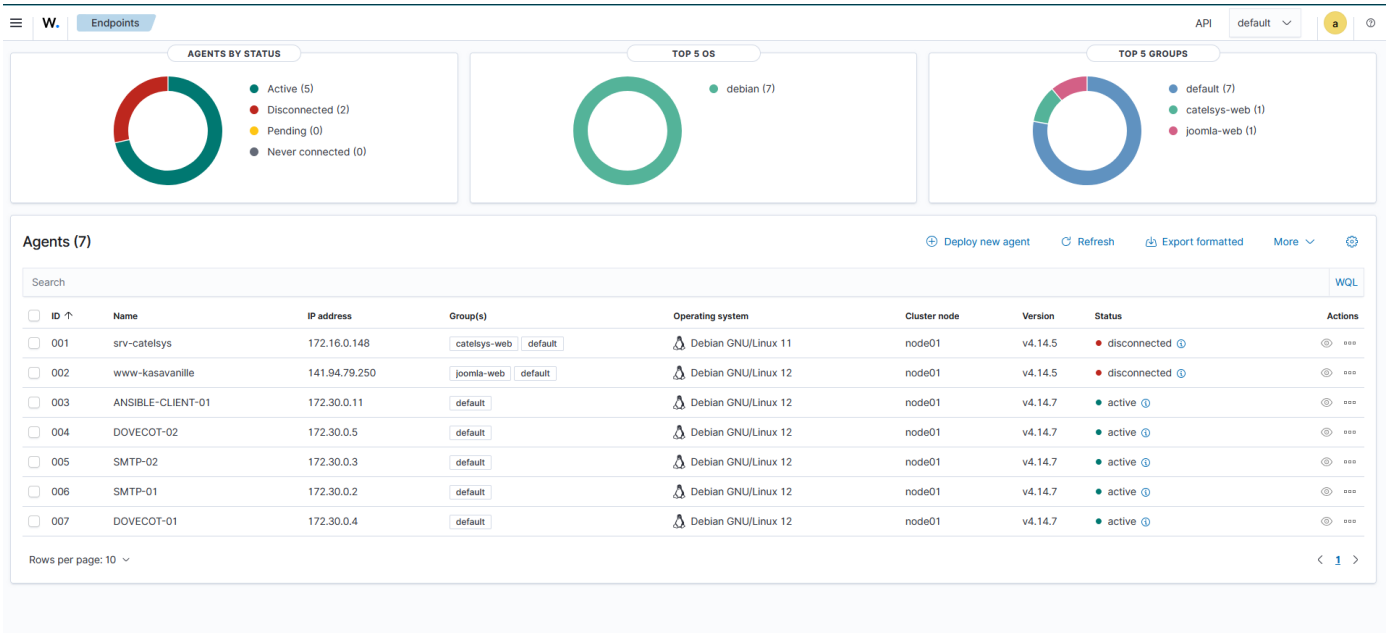
# ANSIBLE + WAZUH

## Déploiement automatisé et sécurisation d'une flotte de serveurs Debian

*Mémo technique, pédagogique et portfolio*




*Résultat final : plusieurs agents Debian enrôlés automatiquement dans Wazuh.*

**Réalisé par Adel Sadek**

Août 2026

# 1. Objectif du projet

L'objectif était de mettre en place une première base de gestion de flotte avec Ansible. Le besoin concret est de pouvoir recenser des serveurs Debian dispersés, les administrer depuis un point central et leur appliquer automatiquement une baseline commune. Le premier cas d'usage retenu a été l'installation et l'enrôlement de l'agent Wazuh.

**Idée centrale :** Ansible décrit l'état que l'on veut obtenir sur les machines. Il évite de se connecter manuellement à chaque serveur pour répéter les mêmes opérations.

# 2. Architecture du laboratoire

```
ANSIBLE-SRV-01
    |
    | SSH + clé Ed25519
    v
ANSIBLE-CLIENT-01 / autres Debian
    |
    | Wazuh Agent - TCP/1514
    v
WAZUH MANAGER : 192.168.10.123
    |
    v
Dashboard Wazuh
```

Le serveur Ansible est le nœud de contrôle. Les machines Debian sont les nœuds gérés. Ansible n'a pas besoin d'un agent spécifique installé sur les clients : il utilise principalement SSH et Python pour exécuter les modules.

# 3. Les termes à connaître

| Terme | Définition simple |
|---|---|
| **Control node / nœud de contrôle** | machine sur laquelle Ansible est installé et depuis laquelle les déploiements sont lancés. |
| **Managed node** | serveur administré par Ansible. |
| **Inventory** | liste structurée des machines et groupes de machines connus d'Ansible. |
| **Playbook** | fichier YAML décrivant quels rôles ou tâches appliquer à quels hôtes. |
| **Play** | association entre un groupe d'hôtes et les actions à exécuter. |
| **Task** | une étape du playbook : installer un paquet, copier un fichier, démarrer un service, etc. |
| **Module** | fonction Ansible spécialisée : apt, systemd, file, get_url, debug, etc. |
| **Role** | brique réutilisable qui regroupe tâches, variables, handlers, templates et autres fichiers autour d'une fonction. |
| **Variable** | valeur configurable utilisée par le code, par exemple l'adresse du Wazuh Manager. |
| **group_vars** | variables communes à un groupe d'hôtes. |
| **host_vars** | variables ou exceptions propres à un seul hôte. |

| | |
|---|---|
| **Handler** | tâche spéciale déclenchée par notify uniquement lorsqu'une modification la rend nécessaire. |
| **Idempotence** | capacité à relancer un playbook sans refaire inutilement les changements déjà appliqués. |
| **become** | élévation de privilèges, généralement vers root via sudo. |
| **Facts** | informations collectées automatiquement sur la machine distante : OS, architecture, interfaces, etc. |

# 4. Mise en place de la connexion SSH

Avant qu'Ansible puisse piloter un serveur, le nœud de contrôle doit pouvoir s'y connecter. Une paire de clés SSH Ed25519 a été utilisée. La clé privée reste sur ANSIBLE-SRV-01 ; seule la clé publique est copiée sur les clients.

```
ssh-keygen -t ed25519
ssh-copy-id root@172.30.0.11
ssh root@172.30.0.11
```

Pour plusieurs machines connues, une boucle Bash peut accélérer le bootstrap :

```
for ip in 172.30.0.2 172.30.0.3 172.30.0.4 172.30.0.5; do
  ssh-copy-id root@$ip
done
```

**À retenir :** ssh-copy-id autorise la connexion ; il n'ajoute pas automatiquement la machine dans l'inventaire Ansible.

# 5. Inventaire Ansible

Exemple d'inventaire INI :

```
[wazuh_clients]
client01 ansible_host=172.30.0.11 ansible_user=root
client02 ansible_host=172.30.0.1  ansible_user=root
client03 ansible_host=172.30.0.2  ansible_user=root
client04 ansible_host=172.30.0.3  ansible_user=root
client05 ansible_host=172.30.0.4  ansible_user=root
client06 ansible_host=172.30.0.5  ansible_user=root
```

Commandes de diagnostic importantes :

```
ansible wazuh_clients --list-hosts
ansible-inventory --graph
ansible wazuh_clients -m ping
```

Le module ping d'Ansible n'est pas un simple ping ICMP. Il vérifie qu'Ansible peut se connecter, exécuter du Python sur le serveur distant et récupérer une réponse.

# 6. Premier playbook Wazuh

Le premier playbook a été volontairement écrit à la main afin d'apprendre la structure YAML. Il installe les prérequis, importe la clé GPG, ajoute le dépôt Wazuh, installe l'agent et active son service.

```
---
- name: Installer et enregistrer Wazuh Agent
  hosts: wazuh_clients
  become: true

  vars:
    wazuh_manager: "192.168.10.123"

  tasks:
    - name: Installer les prerequis
      ansible.builtin.apt:
        name:
          - curl
          - gnupg
          - apt-transport-https
        state: present
        update_cache: true

    - name: Importer la cle GPG Wazuh
      ansible.builtin.shell: |
        curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | \
        gpg --no-default-keyring \
          --keyring gnupg-ring:/usr/share/keyrings/wazuh.gpg \
          --import
        chmod 644 /usr/share/keyrings/wazuh.gpg
      args:
        creates: /usr/share/keyrings/wazuh.gpg

    - name: Ajouter le depot Wazuh
      ansible.builtin.apt_repository:
        repo: "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.wazuh.com/4.x/apt/ stable main"
        filename: wazuh
        state: present

    - name: Mettre a jour le cache APT
      ansible.builtin.apt:
        update_cache: true

    - name: Installer Wazuh Agent
      ansible.builtin.apt:
        name: wazuh-agent
        state: present
      environment:
        WAZUH_MANAGER: "{{ wazuh_manager }}"

    - name: Activer Wazuh Agent au demarrage
      ansible.builtin.systemd:
        name: wazuh-agent
        enabled: true
        state: started
```

# 7. Les erreurs rencontrées et ce qu’elles apprennent

## 7.1 Erreurs YAML

Les premières erreurs venaient principalement de l’indentation, de deux-points manquants et de noms de modules mal orthographiés. YAML dépend fortement des espaces. Les tabulations doivent être évitées.

```
ansible-playbook /etc/ansible/playbooks/wazuh-agent.yml --syntax-check
```

**Réflexe :** toujours lancer --syntax-check avant un déploiement important.

## 7.2 Clé GPG Wazuh

Le premier essai téléchargeait la clé directement vers /usr/share/keyrings/wazuh.gpg. APT ne la reconnaissait pas correctement et retournait NO_PUBKEY 96B3EE5F29111145. La clé a ensuite été importée avec gpg.

Une subtilité supplémentaire est apparue : le fichier wazuh.gpg existait déjà. L'option creates faisait donc croire à Ansible que l'import avait déjà été réalisé. La suppression du mauvais fichier a permis au nouvel import de s'exécuter.

```
ansible wazuh_clients -m file -a "path=/usr/share/keyrings/wazuh.gpg state=absent"
```

**Leçon :** l'idempotence dépend de bons critères. L'existence d'un fichier ne garantit pas que son contenu est correct.

## 7.3 Dépôt Wazuh oublié

Une autre exécution retournait « No package matching 'wazuh-agent' is available ». Le paquet existait, mais APT ne connaissait pas encore le dépôt Wazuh. L'ajout de apt_repository a corrigé le problème.

## 7.4 Erreur rc=137 sur un client

Lors du déploiement sur plusieurs machines, client02 a échoué avec « Killed » et rc=137 pendant l'installation des prérequis. Le code 137 correspond généralement à un processus tué par SIGKILL ; sur une petite VM/LXC, une cause fréquente est une pression mémoire ou un OOM killer. Ce client doit être diagnostiqué séparément (mémoire disponible, journal kernel, limites LXC), sans remettre en cause le succès du rôle sur les autres machines.

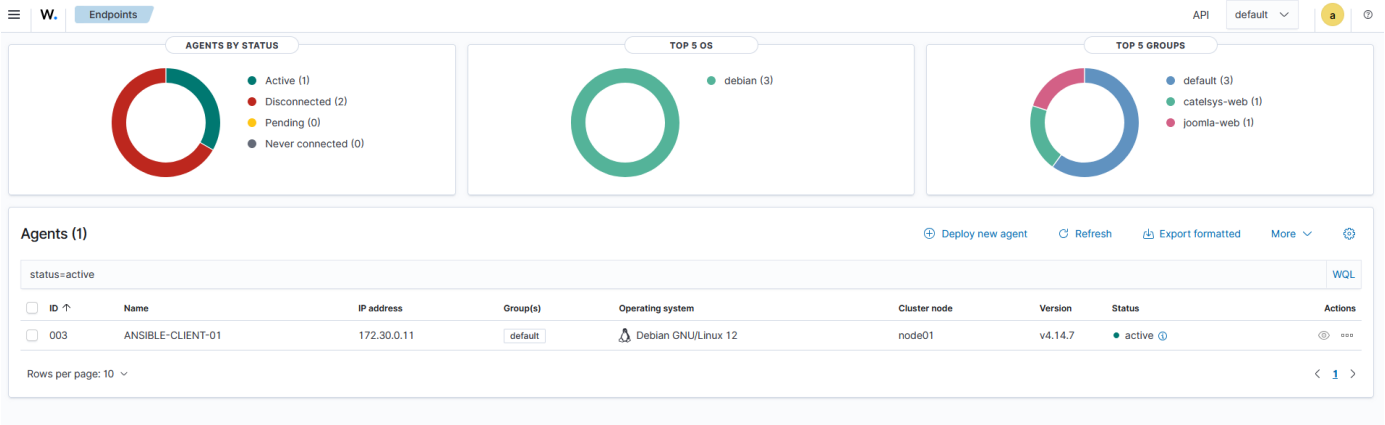
```
ansible client02 -m shell -a "free -h"
ansible client02 -m shell -a "dmesg | tail -n 50"
ansible client02 -m shell -a "journalctl -k -n 50 --no-pager"
ansible-playbook /etc/ansible/playbooks/baseline.yml --limit client02
```

# 8. Validation du premier agent

Le service a été vérifié depuis Ansible :

```
ansible wazuh_clients -m shell -a "systemctl is-active wazuh-agent"
ansible wazuh_clients -m shell -a "grep -A5 '<server>' /var/ossec/etc/ossec.conf"
```

Résultat attendu : service active et adresse du manager 192.168.10.123 sur TCP/1514.




*Premier agent pilote ANSIBLE-CLIENT-01 actif dans Wazuh.*

# 9. Idempotence : test essentiel

Après le premier déploiement, le même playbook a été relancé. Toutes les tâches sont passées à ok et le récapitulatif a affiché changed=0. Cela prouve que le playbook ne réinstalle pas et ne redémarre pas inutilement ce qui est déjà conforme.

```
PLAY RECAP
client01 : ok=8 changed=0 unreachable=0 failed=0
```

**À retenir absolument :** changed=0 lors d'une seconde exécution est un excellent indicateur d'idempotence.

# 10. Passage à un rôle avec Ansible Galaxy

Le playbook monolithique a ensuite été transformé en rôle réutilisable. Ansible Galaxy n'est pas seulement un catalogue en ligne : la commande ansible-galaxy permet aussi de générer automatiquement l'arborescence standard d'un rôle.

```
cd /etc/ansible
ansible-galaxy role init roles/wazuh_agent

roles/wazuh_agent/
├── defaults/
│   └── main.yml
├── handlers/
│   └── main.yml
├── tasks/
│   └── main.yml
├── templates/
├── vars/
├── meta/
└── README.md
```

tasks/main.yml contient la logique d'installation. defaults/main.yml fournit les valeurs par défaut. handlers/main.yml contient les actions déclenchées lorsqu'une tâche notifie un changement.

# 11. Handler et notify

```
- name: Installer Wazuh Agent
  ansible.builtin.apt:
    name: wazuh-agent
    state: present
  environment:
    WAZUH_MANAGER: "{{ wazuh_manager }}"
  notify: Redemarrer Wazuh Agent

---
- name: Redemarrer Wazuh Agent
  ansible.builtin.systemd:
    name: wazuh-agent
    state: restarted
```

Le handler n'est exécuté que si la tâche qui le notifie a réellement changé quelque chose. Cela évite les redémarrages systématiques et inutiles.

# 12. group_vars : séparer le code de la configuration

L'adresse 192.168.10.123 ne doit pas être figée dans le rôle. Elle a donc été déplacée dans group_vars, ce qui permet au même rôle de fonctionner dans différents environnements.

```
# /etc/ansible/group_vars/wazuh_clients.yml
---
wazuh_manager: "192.168.10.123"

ansible wazuh_clients -m debug -a "var=wazuh_manager"
```

La séparation à retenir est :

```
inventory   -> QUI ?
group_vars  -> AVEC QUELLES VALEURS ?
role        -> COMMENT ?
playbook    -> QUOI APPLIQUER ET A QUI ?
```

# 13. Baseline simplifiée

Une fois le rôle créé, le playbook principal devient très lisible :

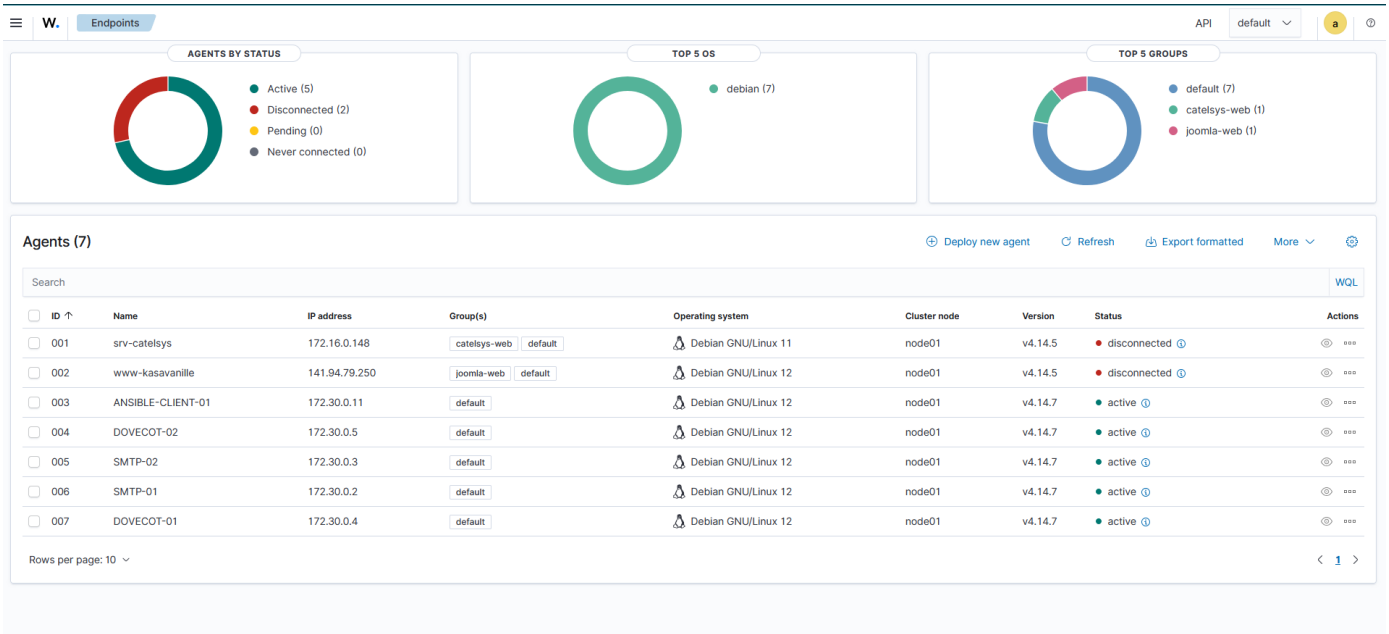
```
---
- name: Baseline Debian
  hosts: wazuh_clients
  become: true

  roles:
    - wazuh_agent
```

À terme, cette baseline pourra regrouper plusieurs rôles indépendants : wazuh_agent, fail2ban, auditd, ssh_hardening, monitoring, patching, sauvegarde, etc.

# 14. Déploiement sur plusieurs serveurs

Après validation sur le client pilote, plusieurs serveurs Debian ont été ajoutés à l'inventaire. Le même rôle a été appliqué en parallèle. Les machines déjà conformes sont restées à changed=0 ; les nouvelles ont reçu les changements nécessaires.

```
client01 : ok=7 changed=0 failed=0
client03 : ok=8 changed=6 failed=0
client04 : ok=8 changed=6 failed=0
client05 : ok=8 changed=6 failed=0
client06 : ok=7 changed=0 failed=0
```




*Dashboard Wazuh après déploiement multi-serveurs : 5 agents actifs, avec DOVECOT-01/02 et SMTP-01/02 enrôlés.*

# 15. Lecture du PLAY RECAP

| Champ | Signification |
|---|---|
| **ok** | tâche exécutée ou état déjà correct |
| **changed** | Ansible a modifié la machine |
| **unreachable** | la machine n'est pas joignable par Ansible |
| **failed** | une tâche a échoué |
| **skipped** | tâche ignorée à cause d'une condition |
| **rescued** | échec récupéré par un bloc rescue |
| **ignored** | échec explicitement ignoré |

# 16. Commandes à connaître

| Commande | Usage |
|---|---|
| **ansible all -m ping** | tester la capacité d'Ansible à piloter les hôtes |
| **ansible wazuh_clients --list-hosts** | voir les hôtes réellement ciblés |
| **ansible-inventory --graph** | visualiser l'inventaire et ses groupes |
| **ansible-playbook fichier.yml --syntax-check** | vérifier la syntaxe |
| **ansible-playbook fichier.yml** | exécuter un playbook |

| | |
|---|---|
| **ansible-playbook fichier.yml --check** | simuler autant que possible sans appliquer |
| **ansible-playbook fichier.yml --limit client01** | limiter l'exécution à un hôte/groupe |
| **ansible-doc apt** | ouvrir la documentation locale du module apt |
| **ansible-doc systemd** | ouvrir la documentation locale du module systemd |
| **ssh-copy-id user@IP** | installer la clé publique SSH sur un hôte |

# 17. Ce qu'il faut mémoriser et ce qu'il ne faut pas apprendre par cœur

- Mémoriser la logique : inventory -> groupes -> playbook -> rôles -> tâches -> modules.
- Comprendre l'idempotence et vérifier changed/failed/unreachable.
- Savoir qu'un module existe probablement avant de recourir à shell.
- Savoir utiliser variables, group_vars, host_vars, handlers et notify.
- Savoir diagnostiquer SSH, APT, systemd et les erreurs de syntaxe YAML.
- Ne pas chercher à mémoriser tous les paramètres de tous les modules : utiliser ansible-doc et la documentation.

# 18. Bonnes pratiques pour la future flotte

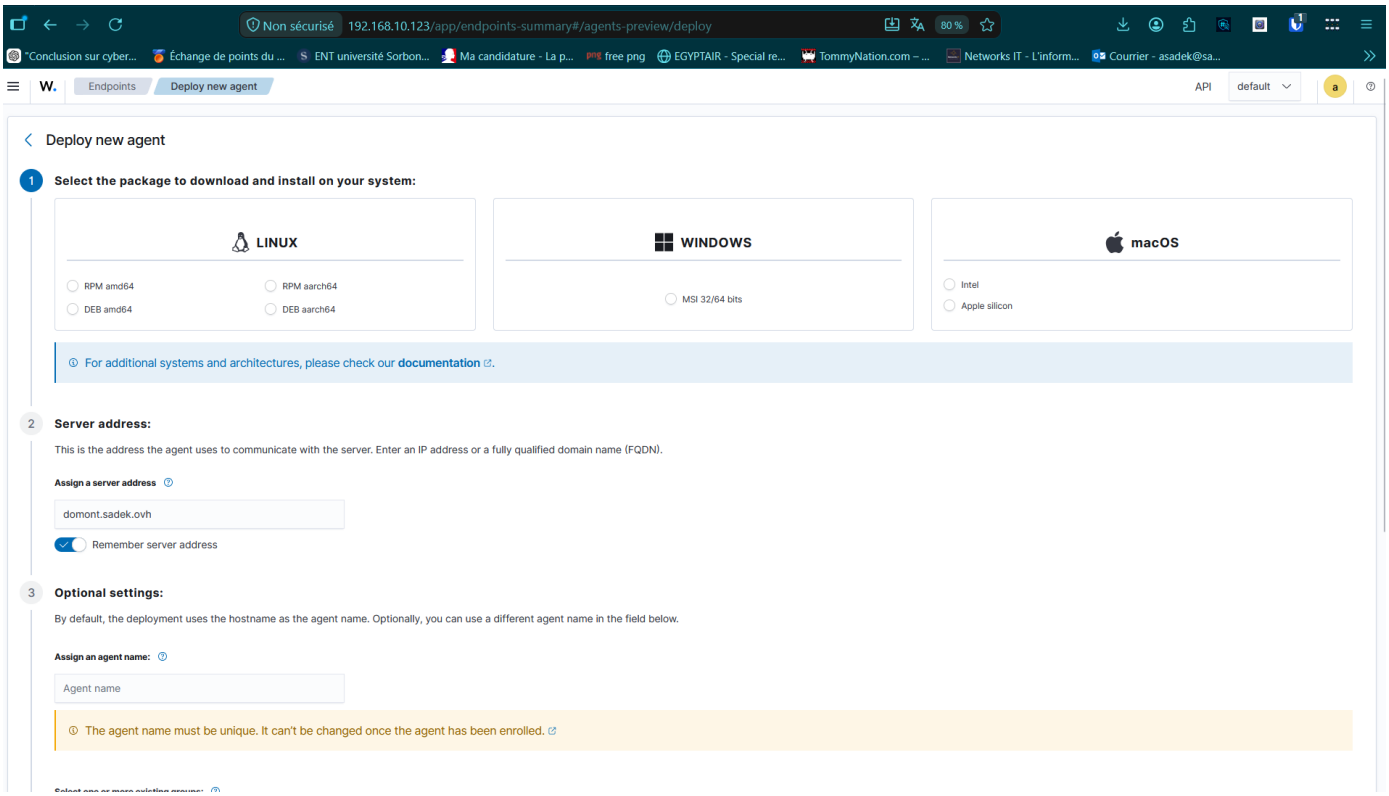
- Tester d'abord sur un client pilote, puis un petit groupe, puis la production.
- Versionner /etc/ansible ou mieux un dépôt de projet dédié dans Git.
- Ne jamais stocker de mots de passe ou secrets en clair ; utiliser Ansible Vault pour les secrets.
- Utiliser --limit lors des changements risqués.
- Utiliser --check et --diff lorsque les modules le permettent.
- Éviter shell/command lorsqu'un module idempotent existe.
- Nommer clairement les groupes : web, mail, docker, wazuh_clients, production, lab.
- Documenter les exceptions avec host_vars plutôt que modifier le rôle générique.
- Surveiller les ressources des petits LXC/VM : un rc=137 peut révéler un manque de mémoire.

# 19. Évolution prévue du projet

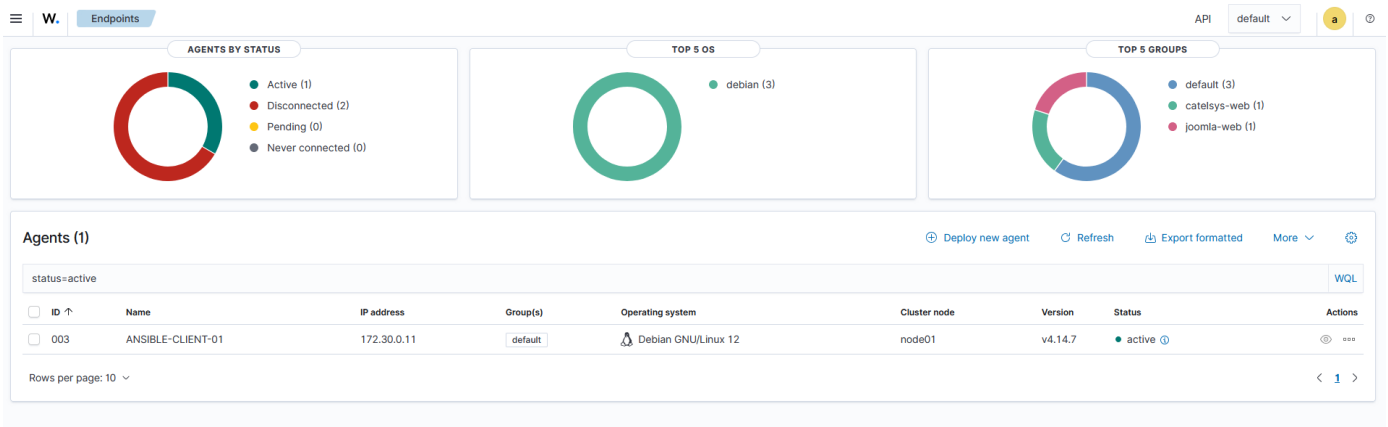
```
Phase actuelle
Ansible + inventaire statique + rôles + Wazuh

Prochaines étapes
├── rôle fail2ban
├── rôle auditd
├── rôle ssh_hardening
├── rôle patching / mises à jour
├── tags
├── host_vars
├── Ansible Vault
├── Git
├── ansible-lint
├── AWX
└── inventaire dynamique / NetBox / Proxmox
```

Terraform pourra ensuite compléter Ansible : Terraform provisionne l'infrastructure Proxmox, tandis qu'Ansible configure le système et les logiciels à l'intérieur des VM/LXC.

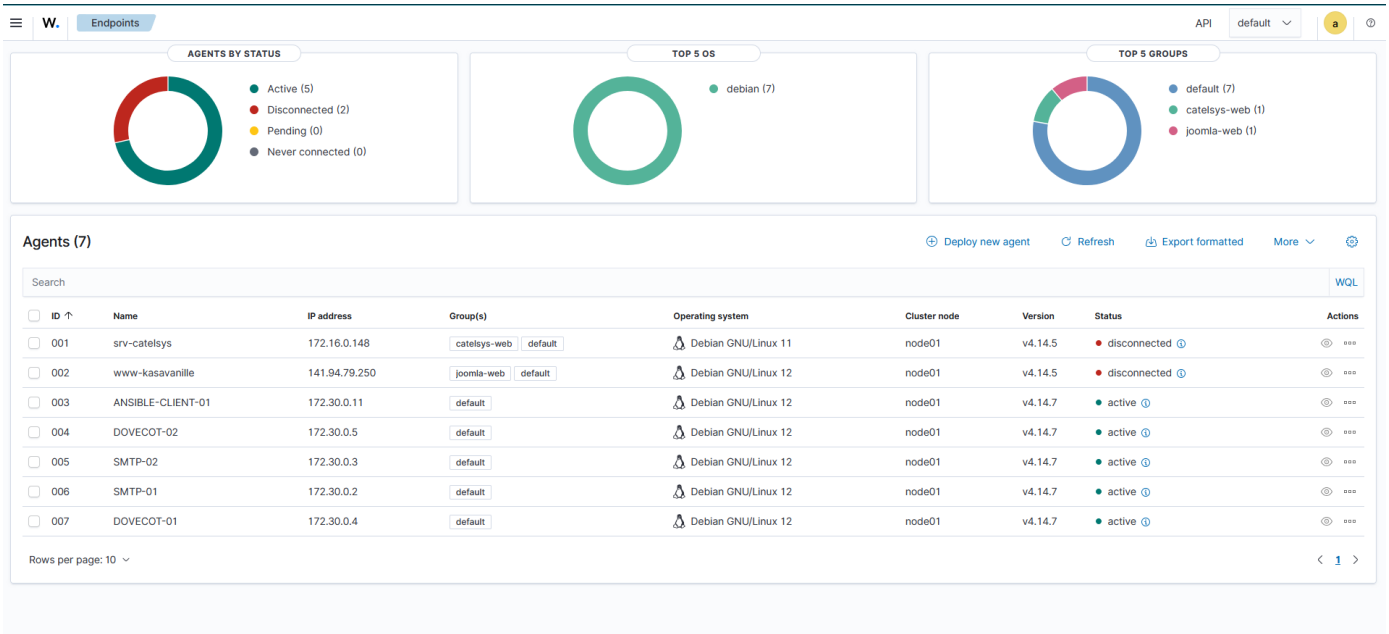
# 20. Captures du projet




*Interface Wazuh - écran de déploiement d'un nouvel agent et configuration du manager.*




*Validation du premier client pilote dans Wazuh.*




*État de la flotte après automatisation de plusieurs serveurs Debian.*

# 21. Conclusion

Ce laboratoire a permis de passer d'une administration manuelle serveur par serveur à une première approche Infrastructure as Code. Le déploiement Wazuh a servi de cas concret pour apprendre SSH, inventaires, YAML, modules, variables, group_vars, rôles, handlers, idempotence et diagnostic d'erreurs. Le résultat est une base réutilisable : ajouter un serveur à l'inventaire et lui donner l'accès SSH permet désormais de lui appliquer la même baseline de manière contrôlée et reproductible.

**Adel Sadek**

*Projet DevOps / Administration Linux - Août 2026*