FR

# OPNsense as Code avec Ansible

## Deployer et versionner toute l'infrastructure en une commande

Serie Stack OPNsense BOTUM — Billet 12/12

Mars 2026

BILLET 12

## Sommaire


1. Pourquoi Infrastructure as Code pour un homelab/PME ?
2. Prerequis : Ansible, SSH et API OPNsense
3. Collection ansibleguy.opnsense : installation et structure
4. Playbook complet : VLANs, firewall, WireGuard, CrowdSec
5. Versionner la config dans Git (GitOps pour OPNsense)
6. Strategie idempotente : relancer sans risque
7. Integration CI/CD : GitHub Actions et Gitea
8. Disaster Recovery : restaurer en une commande
9. Conclusion : recapitulatif du stack complet 12 billets




**BOTUM INC.** www.botum.ca contact@botum.ca | www.botum.ca

## 1. Pourquoi Infrastructure as Code pour un homelab/PME ?

Apres 11 billets a construire un stack de securite reseau complet, une question s'impose : comment **reproduire exactement ce travail** sur un nouveau firewall en cas de panne materielle, de migration ou d'expansion ? Reponse : l'**Infrastructure as Code (IaC)**.

Sans IaC, chaque changement de configuration OPNsense est un geste manuel. Avec IaC, toute la configuration est du **code versionne dans Git**, rejouable en une commande. Pour un homelab ou une PME, les benefices sont concrets :

```
- Reproductibilite  : nouveau firewall operationnel en < 10 minutes
- Versionning       : git log montre chaque changement (qui, quand, pourquoi)
- Idempotence       : relancer le playbook = meme etat, jamais de regression
- Doc vivante       : le code IS la documentation de votre reseau
- Disaster recovery : git clone + ansible-playbook = infrastructure restauree
- Collaboration     : partager la config sans donner acces au firewall
```

Ansible est l'outil ideal pour OPNsense : agentless (aucun daemon a installer), utilise SSH et l'API REST OPNsense, et dispose d'une collection specialisee — **ansibleguy.opnsense**.

## 2. Prerequis : Ansible, SSH et API OPNsense

Avant d'ecrire la premiere ligne de YAML, configurez trois choses : Ansible sur la machine de controle, l'acces SSH a OPNsense, et les credentials API.

### 2.1 Installer Ansible

```
# Ubuntu / Debian
sudo apt update && sudo apt install -y ansible python3-pip
pip3 install requests

# macOS
brew install ansible

# Verifier
ansible --version  # ansible [core 2.16+]
```

### 2.2 Activer SSH sur OPNsense

```
# OPNsense : System > Settings > Administration
# [x] Enable Secure Shell
# [x] SSH port : 22 (ou port personnalise)
# [ ] Permit password login : desactiver apres ajout cle

# Ajouter votre cle publique :
# System > Access > Users > admin > Authorized keys
# Coller le contenu de ~/.ssh/id_ed25519.pub

# Tester depuis la machine de controle :
ssh admin@192.168.1.1 'echo OK'
# > OK
```

### 2.3 Creer un utilisateur API OPNsense

```
# OPNsense : System > Access > Users
# Creer user 'ansible-api' avec droits admin
# System > Access > Users > ansible-api > API keys
# Generer une paire cle/secret -> noter les valeurs

# Variables d'environnement (ou vault Ansible) :
export OPNSENSE_API_KEY="votre-cle-api"
export OPNSENSE_API_SECRET="votre-secret-api"
export OPNSENSE_HOST="https://192.168.1.1"

# Tester l'API :
curl -k -u "${OPNSENSE_API_KEY}:${OPNSENSE_API_SECRET}" \
  "${OPNSENSE_HOST}/api/core/firmware/status"
# {"status":"none","last_check":"..."} -> OK
```

## 3. Collection ansibleguy.opnsense : installation et structure

La collection **ansibleguy.opnsense** est la reference pour automatiser OPNsense via Ansible. Elle couvre interfaces, VLANs, regles firewall, VPN WireGuard, DNS, CrowdSec, et bien plus.

```
# Installer la collection
ansible-galaxy collection install ansibleguy.opnsense

# Structure recommandee du projet
opnsense-ansible/
+-- inventory/
|   +-- hosts.yml           # Hotes OPNsense
|   +-- group_vars/
|       +-- opnsense.yml    # Variables globales + credentials API
+-- playbooks/
|   +-- site.yml            # Playbook principal (tout le stack)
|   +-- vlans.yml           # VLANs uniquement
|   +-- firewall.yml        # Regles firewall
|   +-- wireguard.yml       # VPN WireGuard
|   +-- crowdsec.yml        # CrowdSec bouncer
+-- roles/
|   +-- opnsense_base/      # Role de base (NTP, DNS, SMTP)
+-- ansible.cfg
+-- README.md
```

```
# inventory/hosts.yml
all:
  children:
    opnsense:
      hosts:
        fw-primary:
          ansible_host: 192.168.1.1
          ansible_user: admin
          ansible_ssh_private_key_file: ~/.ssh/id_ed25519
        fw-secondary:           # CARP HA (Billet 10)
          ansible_host: 192.168.1.2
          ansible_user: admin

# inventory/group_vars/opnsense.yml
opnsense_api_host: "https://192.168.1.1"
opnsense_api_key: "{{ lookup('env', 'OPNSENSE_API_KEY') }}"
opnsense_api_secret: "{{ lookup('env', 'OPNSENSE_API_SECRET') }}"
opnsense_ssl_verify: false     # Certificat auto-signe homelab
```

## 4. Playbook complet : VLANs, firewall, WireGuard, CrowdSec

Voici le playbook principal qui configure l'integralite du stack en une seule commande. Chaque section correspond a un role du stack BOTUM.

### 4.1 VLANs et interfaces

```
# playbooks/site.yml
---
- name: "BOTUM Stack - Configuration complete OPNsense"
  hosts: opnsense
  gather_facts: false
  collections:
    - ansibleguy.opnsense

  tasks:
    - name: "VLAN 10 - Management"
      ansibleguy.opnsense.vlan:
        vlan_id: 10
        interface: em0
        description: "VLAN Management"
        state: present

    - name: "VLAN 20 - Servers"
      ansibleguy.opnsense.vlan:
        vlan_id: 20
        interface: em0
        description: "VLAN Servers"
        state: present

    - name: "VLAN 30 - IoT"
      ansibleguy.opnsense.vlan:
        vlan_id: 30
        interface: em0
        description: "VLAN IoT isole"
        state: present

    - name: "VLAN 40 - DMZ"
      ansibleguy.opnsense.vlan:
        vlan_id: 40
        interface: em0
        description: "VLAN DMZ"
        state: present
```

## 4.2 Regles firewall Zero-Trust

```
    # -- REGLES FIREWALL -------------------------------------------------
    - name: "Regle - Autoriser Management vers WAN"
      ansibleguy.opnsense.rule:
        interface: "vlan10"
        source: "vlan10 net"
        destination: "any"
        protocol: "any"
        action: "pass"
        description: "MGMT vers WAN autorise"
        state: present

    - name: "Regle - Bloquer IoT vers LAN"
      ansibleguy.opnsense.rule:
        interface: "vlan30"
        source: "vlan30 net"
        destination: "vlan10 net"
        protocol: "any"
        action: "block"
        description: "IoT ne peut pas acceder LAN/MGMT"
        log: true
        state: present

    - name: "Regle - DNS IoT vers AdGuard uniquement"
      ansibleguy.opnsense.rule:
        interface: "vlan30"
        source: "vlan30 net"
        destination: "192.168.20.5"
        destination_port: "53"
        protocol: "tcp/udp"
        action: "pass"
        state: present
```

## 4.3 WireGuard VPN et CrowdSec

```
    # -- WIREGUARD ----------------------------------------------------
    - name: "WireGuard - Instance principale"
      ansibleguy.opnsense.wireguard_server:
        name: "botum-vpn"
        port: 51820
        private_key: "{{ lookup('env', 'WG_PRIVATE_KEY') }}"
        dns: "192.168.20.5"
        tunnel_address: "10.10.0.1/24"
        state: present

    - name: "WireGuard - Peer mobile"
      ansibleguy.opnsense.wireguard_peer:
        name: "phone-mobile"
        public_key: "{{ lookup('env', 'WG_PEER_PUBKEY') }}"
        allowed_ips: "10.10.0.2/32"
        instance: "botum-vpn"
        state: present

    # -- CROWDSEC -----------------------------------------------------
    - name: "CrowdSec - activer le paquet"
      ansibleguy.opnsense.package:
        name: "os-crowdsec"
        state: present

    - name: "CrowdSec - configurer bouncer"
      ansibleguy.opnsense.crowdsec:
        enabled: true
        bouncer_api_key: "{{ lookup('env', 'CROWDSEC_BOUNCER_KEY') }}"
        crowdsec_api_url: "http://192.168.20.10:8080"
        state: present

# -- LANCER ------------------------------------------------------------
# ansible-playbook -i inventory/hosts.yml playbooks/site.yml
# ansible-playbook -i inventory/hosts.yml playbooks/site.yml --check  # dry-run
```

## 5. Versionner la config dans Git (GitOps pour OPNsense)

La puissance de l'IaC vient du **versioning Git** : chaque modification est tracee, reversible, et documentee automatiquement.

```
# Initialiser le depot Git
cd opnsense-ansible/
git init
git add .
git commit -m "feat: initial stack BOTUM complet (12 billets)"

# .gitignore -- ne jamais committer les secrets
cat > .gitignore << 'EOF'
*.vault
.env
secrets/
inventory/group_vars/vault.yml
EOF

# Utiliser ansible-vault pour chiffrer les secrets
ansible-vault encrypt inventory/group_vars/vault.yml
# Vault password stocke dans : ~/.vault_pass (chmod 600)

# ansible.cfg
[defaults]
vault_password_file = ~/.vault_pass
inventory = inventory/hosts.yml

# Workflow GitOps quotidien :
# 1. Modifier un playbook
# 2. git add + git commit -m "fix: regle IoT plus restrictive"
# 3. git push origin main
# 4. CI/CD applique automatiquement (voir section 7)
```

Chaque commit documente l'histoire de votre infrastructure reseau. **git log --oneline** vous dit exactement quand vous avez ajoute WireGuard, durci les regles IoT, ou active CrowdSec.

## 6. Strategie idempotente : relancer sans risque

L'idempotence est la propriete fondamentale d'Ansible : relancer le meme playbook 10 fois produit **toujours le meme etat final**, sans creer de doublons ni casser ce qui fonctionne.

```
# Tester l'idempotence avant de deployer
ansible-playbook -i inventory/hosts.yml playbooks/site.yml --check
# -> "changed=0 unreachable=0 failed=0" = deja a l'etat desire

# Mode diff : voir exactement ce qui changerait
ansible-playbook -i inventory/hosts.yml playbooks/site.yml --check --diff

# Exemple de sortie idempotente :
# TASK [VLAN 10 - Management]
#   ok: [fw-primary]     <- deja correct, rien a faire
# TASK [Regle - Bloquer IoT vers LAN]
#   changed: [fw-primary] <- nouvelle regle a appliquer

# Tags pour cibler une partie specifique
ansible-playbook site.yml --tags "vlans"         # VLANs seulement
ansible-playbook site.yml --tags "firewall"      # Regles seulement
ansible-playbook site.yml --tags "wireguard"     # VPN seulement
ansible-playbook site.yml --skip-tags "crowdsec" # Tout sauf CrowdSec
```

La regle d'or : **toujours lancer en --check avant un vrai deploiement**. Vous voyez exactement ce qui va changer, sans risque.

## 7. Integration CI/CD : GitHub Actions et Gitea

Automatiser le deploiement a chaque commit. Des que vous poussez une modification sur la branche **main**, le pipeline valide et deploie automatiquement.

### 7.1 GitHub Actions

```
# .github/workflows/deploy-opnsense.yml
name: Deploy OPNsense Stack

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install ansible-lint
        run: pip install ansible-lint
      - name: Lint playbooks
        run: ansible-lint playbooks/site.yml

  deploy:
    runs-on: ubuntu-latest
    needs: lint
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4
      - name: Install Ansible + collection
        run: |
          pip install ansible
          ansible-galaxy collection install ansibleguy.opnsense
      - name: Deploy to OPNsense
        env:
          OPNSENSE_API_KEY: ${{ secrets.OPNSENSE_API_KEY }}
          OPNSENSE_API_SECRET: ${{ secrets.OPNSENSE_API_SECRET }}
          ANSIBLE_VAULT_PASSWORD: ${{ secrets.VAULT_PASSWORD }}
        run: |
          echo "$ANSIBLE_VAULT_PASSWORD" > /tmp/.vault_pass
          ansible-playbook -i inventory/hosts.yml playbooks/site.yml \
            --vault-password-file /tmp/.vault_pass
      - name: Notify Telegram
        if: always()
        run: |
          MSG="OPNsense deploy: ${{ job.status }} - ${{ github.sha }}"
          curl -s "https://api.telegram.org/bot${BOT_TOKEN}/sendMessage" \
            -d "chat_id=${CHAT_ID}&text=${MSG}" > /dev/null
```

## 7.2 Gitea self-hosted

```
# .gitea/workflows/deploy-opnsense.yml
name: Deploy OPNsense

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Ansible
        run: |
          pip install ansible
          ansible-galaxy collection install ansibleguy.opnsense
      - name: Dry-run (check)
        env:
          OPNSENSE_API_KEY: ${{ secrets.OPNSENSE_API_KEY }}
          OPNSENSE_API_SECRET: ${{ secrets.OPNSENSE_API_SECRET }}
        run: ansible-playbook -i inventory/hosts.yml playbooks/site.yml --check
      - name: Deploy
        env:
          OPNSENSE_API_KEY: ${{ secrets.OPNSENSE_API_KEY }}
          OPNSENSE_API_SECRET: ${{ secrets.OPNSENSE_API_SECRET }}
        run: ansible-playbook -i inventory/hosts.yml playbooks/site.yml

# Runner self-hosted : docker run gitea/act_runner register
```

## 8. Disaster Recovery : restaurer en une commande

Le scenario le plus redoute : le firewall grille. Avec le stack IaC, le disaster recovery devient un exercice de routine.

```
# SCENARIO : OPNsense principal hors service
# Temps de restauration cible : < 15 minutes

# Etape 1 : Provisionner un nouveau OPNsense (Proxmox, ~5 min)
# -> Installer l'ISO depuis https://opnsense.org/download/
# -> Configuration reseau de base via console (IP, SSH)

# Etape 2 : Restaurer la config Ansible
git clone https://gitea.botum.ca/botum/opnsense-ansible.git
cd opnsense-ansible

# Etape 3 : Deployer le stack complet (~8 min)
OPNSENSE_API_KEY=xxx OPNSENSE_API_SECRET=yyy \
  ansible-playbook -i inventory/hosts.yml playbooks/site.yml

# Resultat apres execution :
# [ok] VLANs configures        (Billet 2)
# [ok] Regles firewall          (Billet 2 + 3)
# [ok] WireGuard restaure       (Billet 4)
# [ok] CrowdSec active          (Billet 3)
# [ok] Suricata IDS/IPS         (Billet 7)

# Etape 4 : Verifier l'idempotence
ansible-playbook -i inventory/hosts.yml playbooks/site.yml --check
# changed=0 unreachable=0 failed=0 -> Stack 100% conforme
```

En comparaison, une restauration manuelle prendrait 2 a 4 heures, avec le risque d'oublier des regles critiques. L'IaC transforme le disaster recovery en **procedure documentee et testable**.

## 9. Conclusion : recapitulatif du stack complet 12 billets

Avec ce Billet 12, le **Stack OPNsense BOTUM** est complet. En 12 billets, nous avons construit une infrastructure de securite reseau de niveau enterprise, reproductible, versionnee et automatisee :

- **Billet 1** — OPNsense sur Proxmox — fondation hyperviseur
- **Billet 2** — VLANs Zero-Trust — segmentation reseau
- **Billet 3** — CrowdSec IPS collaboratif — blocage communautaire
- **Billet 4** — WireGuard VPN — acces securise a distance
- **Billet 5** — fail2ban renforce — protection brute-force
- **Billet 6** — NAC — controle d'acces au reseau
- **Billet 7** — Suricata IDS/IPS — detection d'intrusion profonde
- **Billet 8** — AdGuard Home + DoH/DoT — DNS filtre et chiffre
- **Billet 9** — Grafana + InfluxDB — monitoring reseau
- **Billet 10** — CARP haute disponibilite — resilience infrastructure
- **Billet 11** — Wazuh SIEM — correlation et detection
- **Billet 12** — Ansible as Code — automatisation et GitOps [ce guide]

Le stack est maintenant **entierement reproductible** : un nouveau collaborateur peut deployer l'integralite de cette infrastructure en moins de 30 minutes, depuis zero, grace a Ansible.

**Prochains billets (13-15) :** Backup automatise de la config OPNsense, certificats Let's Encrypt avec ACME, et analyse de trafic avec Netflow + ntopng.

---

**Article complet :** blog.botum.ca/opnsense-ansible-as-code-infrastructure

Site web : www.botum.ca • contact@botum.ca