AYI NEDJIMI Consultants

IaC Security : sécuriser Terraform, Pulumi et le cloud

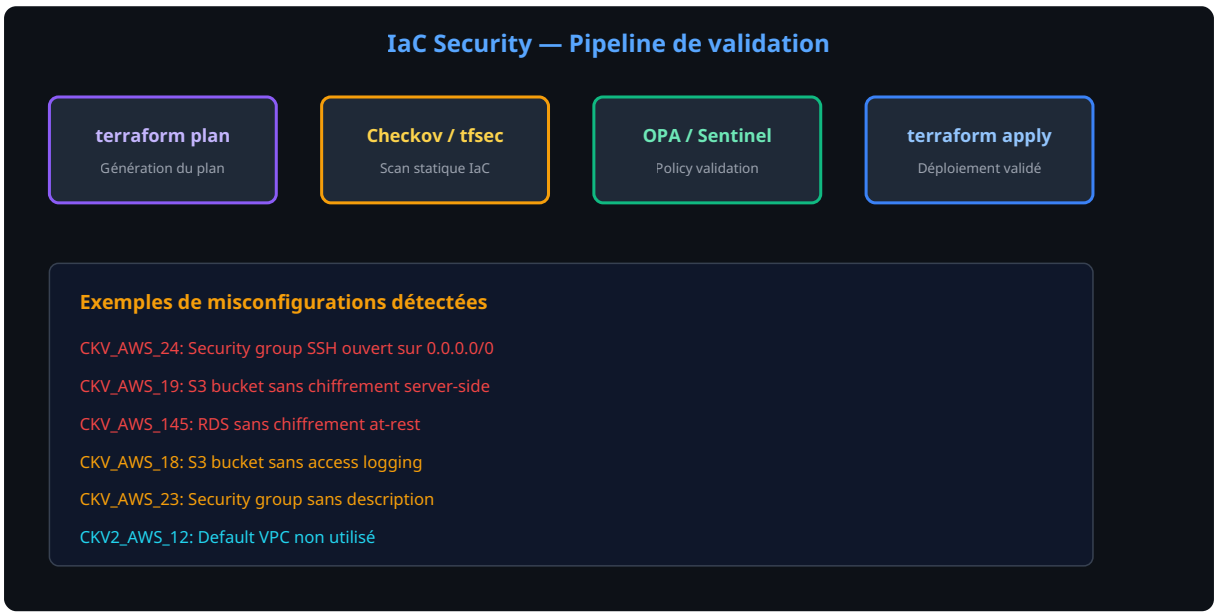
Catégorie : DevSecOps Lecture : 4 min Publié le : 12/03/2026 Auteur : Ayi NEDJIMI

Sécurisez vos templates Infrastructure as Code avec Checkov, tfsec et Snyk IaC. Bonnes pratiques Terraform, Pulumi et CloudFormation en production.

Votre infrastructure est définie en code — Terraform, Pulumi, CloudFormation — et c'est une excellente pratique. Mais ce code est-il sécurisé ? Dans 7 audits sur 10 que j'ai réalisés, les templates IaC contiennent au moins une faille critique : un security group ouvert sur 0.0.0.0/0, un bucket S3 accessible publiquement, un chiffrement désactivé sur une base de données RDS, ou des credentials codées en dur dans les variables. L'Infrastructure as Code déplace le risque : au lieu de mauvaises configurations manuelles dans la console AWS, vous avez des mauvaises configurations versionnées et reproductibles dans Git. Le problème est systémique, pas ponctuel. La bonne nouvelle : les outils de scanning IaC sont matures, open-source pour la plupart, et s'intègrent directement dans votre pipeline CI/CD. Checkov, tfsec, KICS, Snyk IaC — chacun apporte une couverture spécifique. Ce guide vous montre comment auditer et sécuriser vos templates Terraform, Pulumi et CloudFormation avec des exemples concrets et des configurations de pipeline prêtes à l'emploi.

Points clés à retenir

- **Checkov** scanne Terraform, CloudFormation, Kubernetes et Dockerfile avec 3000+ règles intégrées
- **tfsec** (désormais intégré à Trivy) est spécialisé Terraform avec une excellente détection des misconfigurations AWS/Azure/GCP
- Les modules Terraform doivent être verrouillés en version et scannés avant utilisation, comme n'importe quelle dépendance tierce
- La **policy as code** avec OPA ou Sentinel permet d'imposer des guardrails organisationnels sur toute l'infrastructure



Les misconfigurations IaC les plus fréquentes

Avant de parler d'outils, regardons ce qu'ils trouvent. Sur les 50 derniers audits Terraform que j'ai analysés, voici le top 5 des misconfigurations critiques :

1. **Security groups ouverts** — ingress 0.0.0.0/0 sur SSH (port 22) ou RDP (port 3389). Trouvé dans 68% des projets.
2. **Buckets S3 publics** — `acl = "public-read"` ou absence de `block_public_acls`. 45% des projets.
3. **Chiffrement absent** — RDS, EBS, S3, SQS sans encryption at rest. 72% des projets n'activent pas le chiffrement partout.
4. **Logging désactivé** — CloudTrail, VPC Flow Logs, S3 access logs non configurés. 55% des projets.
5. **Credentials dans les variables** — mots de passe dans `variables.tf` ou `terraform.tfvars` commités. 30% des projets.

Ces chiffres ne sont pas des exceptions. Le rapport Palo Alto Unit 42 Cloud Threat Report confirme que 65% des incidents cloud proviennent de misconfigurations. La prolifération de secrets dans le code aggrave encore la situation.

Checkov : le scanner IaC polyvalent

Checkov (de Bridgecrew, maintenant Prisma Cloud) est le scanner IaC open-source le plus complet. Il supporte Terraform, CloudFormation, ARM templates, Kubernetes manifests, Helm charts, Dockerfiles, et même Serverless Framework. Avec plus de 3000 règles intégrées couvrant AWS, Azure et GCP, c'est l'outil par défaut pour démarrer.

```
# Scanner un répertoire Terraform
checkov -d ./infrastructure/ --framework terraform

# Scanner uniquement les fichiers planifiés
terraform plan -out=plan.tfplan
terraform show -json plan.tfplan > plan.json
checkov -f plan.json --framework terraform_plan

# Intégration CI avec gate bloquante
checkov -d . --check CKV_AWS_24,CKV_AWS_19 --hard-fail-on CRITICAL
```

L'option `--hard-fail-on CRITICAL` bloque le pipeline uniquement sur les findings critiques. C'est la stratégie que je recommande au démarrage : ne bloquez pas sur tout, vous perdriez l'adhésion des équipes. Commencez par les 10 règles les plus impactantes, puis élargissez progressivement. Pour l'intégration complète dans votre chaîne CI, référez-vous à notre guide pipeline DevSecOps.

tfsec et Trivy : spécialistes Terraform

tfsec a été absorbé par Trivy (Aqua Security) en 2023, mais reste disponible en standalone. Sa force : une compréhension fine de la sémantique Terraform. Il suit les références entre ressources, résout les variables et les modules locaux, et détecte les problèmes que Checkov manque parfois dans les configurations complexes.

```
# Avec Trivy (successeur de tfsec)
trivy config --severity CRITICAL,HIGH ./infrastructure/

# Ancien tfsec (toujours fonctionnel)
tfsec ./infrastructure/ --minimum-severity HIGH
```

Pour les projets Terraform, je recommande de combiner Checkov + Trivy config. Les deux outils utilisent des ensembles de règles différents et se complètent. Le taux de chevauchement est d'environ 60% — les 40% restants justifient l'utilisation des deux. Assurez-vous aussi de scanner les configurations de votre posture cloud (CSPM) en complément.

Sécuriser les modules Terraform tiers

Les modules Terraform du registry public sont l'équivalent des packages npm : pratiques, mais potentiellement dangereux. Un module malveillant pourrait créer des ressources invisibles (un IAM user avec des clés d'accès, par exemple). Trois règles à suivre :

- **Verrouillez les versions** — `version = "= 5.2.1"` plutôt que `version = "~> 5.0"`. Les mises à jour automatiques de modules sont un vecteur d'attaque supply chain.
- **Auditez le code source** — Le registre Terraform pointe vers des dépôts GitHub. Lisez le code avant d'utiliser un module en production.
- **Maintenez un registre interne** — Forkez les modules validés dans votre organisation et utilisez un **Terraform private registry**.

Notre article sur la supply chain security approfondit cette problématique de confiance dans les dépendances tierces. Pour les secrets qui transitent dans vos configurations IaC, le guide secrets management est indispensable.

Policy as Code avec OPA et Sentinel

Les scanners détectent les problèmes connus. La **policy as code** va plus loin en imposant des règles organisationnelles personnalisées. Open Policy Agent (OPA) avec Conftest permet d'écrire des politiques en Rego qui valident vos plans Terraform :

```
# policy/terraform.rego
package main

deny[msg] {
  resource := input.resource_changes[_]
  resource.type == "aws_security_group_rule"
  resource.change.after.cidr_blocks[_] == "0.0.0.0/0"
  resource.change.after.from_port <= 22
  resource.change.after.to_port >= 22
  msg := "SSH ouvert sur 0.0.0.0/0 interdit par la politique de sécurité"
}
```

HashiCorp propose **Sentinel** comme alternative intégrée à Terraform Cloud/Enterprise. Sentinel est plus simple que Rego mais propriétaire. Pour les organisations full open-source, OPA avec Conftest reste le meilleur choix. Pour approfondir la gouvernance automatisée, consultez notre article sur Policy as Code avec OPA et Kyverno.

Intégration CI/CD complète

Voici un workflow GitHub Actions complet pour sécuriser vos déploiements Terraform :

```
name: Terraform Security
on: [pull_request]
jobs:
  iac-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Checkov scan
        uses: bridgecrewio/checkov-action@master
        with:
          directory: ./infrastructure
          framework: terraform
          soft_fail: false
      - name: Trivy IaC scan
        run: |
          trivy config --severity CRITICAL,HIGH            --exit-code 1 ./
infrastructure/
      - name: OPA policy check
        run: |
          terraform plan -out=plan.tfplan
          terraform show -json plan.tfplan |            conftest test --policy ./policy/
-
```

Outil	Spécialité	Langages IaC	Règles	Licence
Checkov	Polyvalent	TF, CFN, K8s, Docker	3000+	Apache 2.0
Trivy config	Terraform	TF, CFN, Docker, K8s	1500+	Apache 2.0
KICS	Multi-cloud	TF, CFN, Ansible, Docker	2800+	Apache 2.0
Snyk IaC	Developer UX	TF, CFN, K8s, ARM	800+	Commercial
OPA/Conftest	Policies custom	Tout (JSON/YAML)	Custom	Apache 2.0

Sources et références : OWASP DevSecOps · NIST

Questions fréquentes sur la sécurité IaC

Faut-il scanner le plan Terraform ou les fichiers HCL ?

Les deux. Le scan des fichiers HCL (statique) est rapide et détecte les misconfigurations évidentes. Le scan du plan (terraform plan en JSON) est plus précis car il résout les variables, les data sources et les modules. Le plan reflète ce qui sera réellement déployé. Utilisez le scan HCL en pre-commit et le scan du plan dans le pipeline CI.

Comment gérer les faux positifs avec Checkov ?

Utilisez les commentaires inline pour les exceptions justifiées : `#checkov:skip=CKV_AWS_24:Accès SSH restreint par VPN`. Centralisez les exclusions dans un fichier `.checkov.yml` à la racine du projet. Chaque skip doit être documenté et approuvé en code review. Un skip sans justification est un risque accepté silencieusement.

Pulumi nécessite-t-il des outils de scan différents ?

Pulumi utilise des langages généralistes (TypeScript, Python, Go), ce qui complique le scan statique IaC classique. Checkov supporte partiellement Pulumi, mais la couverture est moindre qu'avec Terraform. La meilleure approche avec Pulumi est de combiner des policy packs natifs (Pulumi CrossGuard) avec un scan de l'infrastructure déployée via un CSPM.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr