AYI NEDJIMI Consultants

Policy as Code : OPA, Kyverno et gouvernance cloud

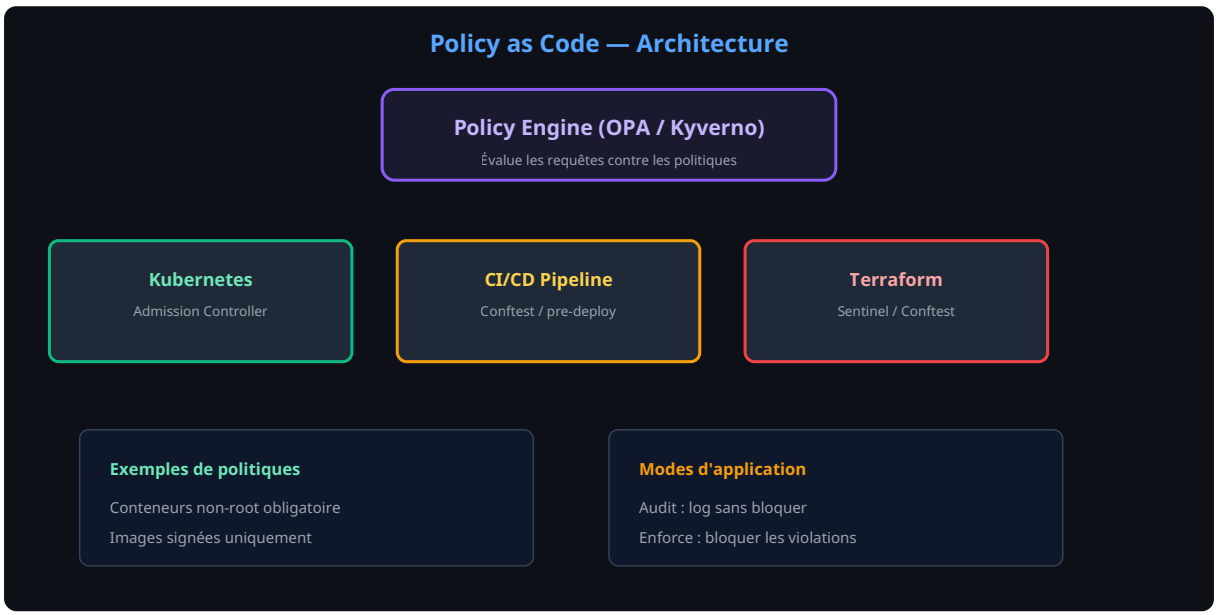
Catégorie : DevSecOps Lecture : 4 min Publié le : 12/03/2026 Auteur : Ayi NEDJIMI

Policy as Code avec OPA, Kyverno et Sentinel pour automatiser la gouvernance cloud et Kubernetes. Exemples de politiques et intégration CI/CD.

Votre équipe déploie 50 microservices sur Kubernetes. Chaque déploiement doit respecter des règles : pas de conteneur en root, pas d'image provenant d'un registry public non autorisé, des limites de ressources obligatoires, des labels de conformité présents. Qui vérifie tout cela ? Si la réponse est "personne" ou "l'équipe sécurité en audit trimestriel", vous avez un problème de gouvernance. La Policy as Code transforme vos règles de sécurité, de conformité et d'architecture en code exécutable, versionné et testé comme n'importe quel logiciel. Open Policy Agent (OPA) avec le langage Rego, Kyverno avec son approche YAML-native pour Kubernetes, et Sentinel pour l'écosystème HashiCorp — chaque outil apporte une réponse adaptée à un contexte spécifique. Ce guide vous montre comment concevoir, déployer et maintenir des politiques de gouvernance automatisées qui protègent votre infrastructure sans ralentir les équipes de développement. Vous repartirez avec des exemples de politiques prêts à l'emploi pour les scénarios les plus courants.

Points clés à retenir

- **OPA** avec Rego est le standard universel pour les politiques déclaratives — cloud, Kubernetes, CI/CD, API gateways
- **Kyverno** est l'option la plus accessible pour Kubernetes avec sa syntaxe YAML native et ses capacités de mutation
- Les politiques doivent être testées avec des suites de tests unitaires comme n'importe quel code
- Le mode **audit** (dry-run) est indispensable avant de passer en mode **enforce** (bloquant)



OPA et Rego : le standard universel

Open Policy Agent (CNCF graduated) est un moteur de politique générique qui évalue des requêtes JSON contre des règles écrites en **Rego**. Son universalité est sa force : il fonctionne avec Kubernetes (via Gatekeeper), les pipelines CI/CD (via Conftest), Terraform, Envoy, Kafka — tout ce qui produit ou consomme du JSON/YAML.

```
# Politique OPA : interdire les conteneurs root
package kubernetes.admission

deny[msg] {
  container := input.request.object.spec.containers[_]
  not container.securityContext.runAsNonRoot
  msg := sprintf("Le conteneur %v doit avoir runAsNonRoot: true", [container.name])
}

deny[msg] {
  container := input.request.object.spec.containers[_]
  container.securityContext.privileged
  msg := sprintf("Le conteneur %v ne peut pas tourner en mode privileged",
[container.name])
}
```

Rego a une courbe d'apprentissage. C'est un langage déclaratif qui déroute les développeurs habitués à l'impératif. Mon conseil : investissez 2-3 jours de formation pour votre équipe platform, et fournissez des templates de politiques pour les cas courants. Le guide Rego officiel est le meilleur point de départ.

Kyverno : la politique Kubernetes en YAML natif

Si vous travaillez exclusivement avec Kubernetes et que Rego vous rebute, **Kyverno** est la solution. Les politiques sont écrites en YAML — le même langage que vos manifests Kubernetes. Kyverno offre trois capacités que Gatekeeper (OPA) n'a pas nativement : la **mutation** (modifier les resources à la volée), la **génération** (créer des resources automatiquement) et la **vérification d'images**.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-non-root
spec:
  validationFailureAction: Enforce
  rules:
    - name: check-runAsNonRoot
      match:
        any:
          - resources:
              kinds: ["Pod"]
      validate:
        message: "Les conteneurs doivent tourner en non-root"
        pattern:
          spec:
            containers:
              - securityContext:
                  runAsNonRoot: true
```

La politique de mutation est particulièrement puissante : Kyverno peut automatiquement ajouter les labels manquants, injecter un sidecar de monitoring, ou définir les limites de ressources par défaut. C'est la "carotte" qui accompagne le "bâton" de la validation. Pour le hardening complet de votre cluster, consultez notre guide sécurité conteneurs Docker et Kubernetes.

Tester vos politiques comme du code

Une politique mal écrite qui bloque des déploiements légitimes en production, c'est pire que pas de politique. Testez systématiquement :

```
# Tests OPA avec opa test
opa test ./policies/ -v

# Tests Kyverno avec kyverno CLI
kyverno test ./tests/
```

Chaque politique doit avoir au minimum deux types de tests : un test positif (la politique autorise un input conforme) et un test négatif (la politique bloque un input non conforme). Versionnez les politiques dans Git, intégrez les tests dans votre CI, et appliquez le même workflow de code review que pour le code applicatif. Les politiques passent par les mêmes étapes que vos pipelines CI/CD sécurisés.

Déploiement progressif : audit avant enforce

Déployer une politique directement en mode `Enforce` sans période d'observation est une recette pour le désastre. Procédez en trois phases :

1. **Audit** (2-4 semaines) — La politique est active mais ne bloque rien. Elle génère des rapports de violations. Analysez les résultats pour identifier les faux positifs et les cas légitimes à exempter.
2. **Warn** (1-2 semaines) — La politique affiche un avertissement à l'utilisateur mais autorise l'opération. Les équipes s'adaptent.
3. **Enforce** — La politique bloque activement les violations. Les exemptions sont documentées et revues trimestriellement.

Kyverno supporte nativement ces modes via `validationFailureAction: Audit` ou `Enforce`. Pour OPA/Gatekeeper, utilisez le paramètre `enforcementAction: dryrun`. La conformité de votre infrastructure cloud peut aussi bénéficier de cette approche, comme détaillé dans notre guide NIS2 et conformité cloud.

Politiques essentielles pour Kubernetes

Voici les 10 politiques que chaque cluster Kubernetes de production devrait appliquer :

Politique	Catégorie	Impact
Conteneurs non-root obligatoire	Sécurité runtime	Critique
Read-only root filesystem	Sécurité runtime	Élevé
Drop ALL capabilities	Sécurité runtime	Élevé
Images signées uniquement	Supply chain	Critique
Registry autorisé uniquement	Supply chain	Élevé
Limites CPU/mémoire obligatoires	Stabilité	Moyen
Labels conformité obligatoires	Gouvernance	Moyen
Pas de hostNetwork/hostPID	Isolation	Critique
NetworkPolicy par namespace	Réseau	Élevé
Pas de latest tag sur les images	Reproductibilité	Moyen

Le référentiel NIST SP 800-190 Application Container Security Guide fournit les recommandations détaillées pour chaque catégorie.

Sources et références : OWASP DevSecOps · NIST

Questions fréquentes sur la policy as code

OPA ou Kyverno, lequel choisir pour Kubernetes ?

Si votre besoin se limite à Kubernetes : Kyverno. Sa syntaxe YAML est accessible à tous les ingénieurs Kubernetes, ses capacités de mutation et génération sont uniques, et son adoption est plus rapide. Si vous avez besoin de politiques sur plusieurs plateformes (K8s + Terraform + API gateway) : OPA, car sa polyvalence justifie l'investissement dans Rego.

Comment gérer les exceptions aux politiques ?

Kyverno supporte les exceptions via des PolicyException resources ou des annotations sur les namespaces. OPA/Gatekeeper utilise des Config resources pour exempter des namespaces ou des resources spécifiques. Chaque exception doit être documentée avec un propriétaire, une justification et une date d'expiration. Revue trimestrielle obligatoire.

Peut-on utiliser la policy as code sans Kubernetes ?

Absolument. OPA avec Conftest valide n'importe quel fichier JSON/YAML : Terraform plans, Docker Compose, configurations Ansible, fichiers CI/CD. Sentinel fonctionne avec Terraform Cloud, Vault et Consul. La policy as code est un paradigme, pas un outil spécifique à Kubernetes. Pour la sécurisation de votre IaC Terraform, consultez notre guide IaC Security.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

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