Governance-as-Code pour Agentic AI : Implémentation Pas-à-Pas

Governance-as-Code permet de définir des politiques de sécurité pour les agents IA en code versionnable et auditable. Ce guide couvre OPA, Kyverno, les garde-fous CI/CD, la gestion des dépendances vulnérables (SCA) et le ROI mesurable pour les équipes DevSecOps travaillant avec des agents autonomes.

Governance-as-Code pour Agentic AI est l'approche qui consiste à définir les politiques de contrôle des agents IA autonomes sous forme de code versionnable, testable et auditable — au même titre que l'infrastructure-as-code pour les serveurs. En 2026, les organisations déploient des agents IA capables d'exécuter du code, d'accéder à des bases de données, d'envoyer des emails ou de modifier des fichiers en production, créant une surface d'attaque et des risques de conformité inédits. Selon le rapport OWASP Top 10 pour les LLM Applications 2025, les risques de prompt injection (LLM01), d'accès excessif aux ressources (LLM08) et de dépendances non sécurisées (LLM09) figurent parmi les menaces les plus critiques pour les agents IA en production. La Governance-as-Code répond à ces risques avec des outils comme OPA (Open Policy Agent), Kyverno pour Kubernetes, et des pipelines CI/CD intégrant SCA (Software Composition Analysis) pour détecter les dépendances vulnérables des agents IA. Ce guide pratique vous présente l'implémentation concrète de garde-fous techniques pour les agents IA : politiques OPA en Rego, admission controllers Kubernetes pour les pods exécutant des agents, scan SCA des dépendances Python/Node.js des agents, et métriques de ROI sur la réduction des incidents de sécurité.

À RETENIR

À retenir

Governance-as-Code = politiques de sécurité IA en dépôt Git : les politiques OPA/Rego et les contraintes Kyverno sont versionnées, reviewées via Pull Request et auditables — toute modification de politique laisse une trace dans l'historique Git.

OPA + Rego pour les décisions d'autorisation agents IA : Open Policy Agent évalue en temps réel si un agent IA peut exécuter une action (appel API, écriture fichier, exécution de code) selon des politiques définies en langage Rego — découplage entre la logique de l'agent et les règles de gouvernance.

SCA obligatoire pour les dépendances des agents : les frameworks d'agents IA (LangChain, LlamaIndex, AutoGen) ont des chaînes de dépendances complexes avec des vulnérabilités documentées dans la NVD — Trivy, Grype et OWASP Dependency-Check doivent scanner ces dépendances à chaque build.

Kyverno pour le contrôle des pods agents Kubernetes : les policies Kyverno bloquent le déploiement de pods agents IA sans limites de ressources, sans SecurityContext restrictif ou avec des mounts de secrets non autorisés — admission control déclaratif sans code Go.

Moindre privilège critique pour les agents autonomes : un agent IA avec accès lecture/écriture à une base de données de production constitue un risque de données massif en cas de prompt injection — chaque agent doit disposer d'un compte de service avec permissions strictement limitées à ses fonctions.

Pourquoi la Governance-as-Code est indispensable pour les agents IA ?

Les agents IA autonomes (Agentic AI) se distinguent des LLMs classiques par leur capacité à agir sur le monde réel via des outils (tool calls) : exécution de code Python, appels API, requêtes de bases de données, envoi d'emails, modifications de fichiers. Un agent autonome mal contrôlé peut, suite à une injection de prompt malveillante ou à une erreur d'inférence, exécuter des actions destructrices : supprimer des données, exfiltrer des informations sensibles ou escalader des privilèges. Selon l'OWASP, 71 % des incidents de sécurité liés aux LLM en 2025 impliquaient des agents avec des permissions excessives.

La Governance-as-Code applique les principes éprouvés de l'Infrastructure-as-Code aux politiques de sécurité des agents : déclaratif (les règles décrivent l'état désiré, pas les étapes), versionnable (Git comme source de vérité), testable (unit tests pour les politiques OPA), auditable (chaque changement de politique est tracé) et cohérent (la même politique s'applique en dev, staging et production). Cette approche répond aux exigences de conformité NIS 2 (article 21 sur les systèmes de gestion des risques) et ISO 27001 A.14 sur la sécurité du développement.

OPA et Rego : définir les politiques d'autorisation des agents IA

Open Policy Agent (OPA) est le moteur de politiques open source de la CNCF, utilisé en production chez Netflix, Atlassian et Airbnb. Il évalue les requêtes d'autorisation selon des politiques écrites en Rego, un langage déclaratif spécialement conçu pour les décisions d'autorisation. Pour les agents IA, OPA sert de point de contrôle centralisé : avant qu'un agent exécute une action (outil), l'orchestrateur consulte OPA pour valider si l'action est autorisée selon les politiques en vigueur.

```
 # Installation OPA
curl -L -o /usr/local/bin/opa https://openpolicyagent.org/downloads/latest/opa_linux_amd64_static
chmod +x /usr/local/bin/opa

# Politique Rego : contrôle des actions d'un agent IA
cat > agent_policy.rego << 'EOF'
package agentic.authorization

import future.keywords.if
import future.keywords.in

# Règle principale : l'action est autorisée si elle passe toutes les vérifications
default allow := false

allow if {
    not action_requires_elevation
    agent_has_permission
    within_rate_limit
    not sensitive_data_access
}

# Bloquer les actions nécessitant des privilèges élevés
action_requires_elevation if {
    input.action.type in ["exec_shell", "write_system_file", "modify_firewall"]
}

# Vérifier que l'agent a la permission pour ce type d'action
agent_has_permission if {
    some perm in data.agents[input.agent_id].permissions
    perm == input.action.type
}

# Rate limiting : max 100 actions par heure par agent
within_rate_limit if {
    count(past_actions_last_hour) < 100
}

past_actions_last_hour := actions if {
    actions := [a | some a in data.action_log
        a.agent_id == input.agent_id
        a.timestamp > (time.now_ns() - (3600 * 1000000000))
    ]
}
```

```
# Bloquer l'accès aux données sensibles (PII, secrets)
sensitive_data_access if {
    input.action.target_resource in data.sensitive_resources
}

# Raisons du refus (pour logs d'audit)
deny_reasons := reasons if {
    reasons := concat(", ", [r |
        some rule in ["action_requires_elevation", "not agent_has_permission",
                      "not within_rate_limit", "sensitive_data_access"]
        _ := {rule: true}[rule]
        r := rule
    ])
}
EOF

# Tester la politique
cat > test_input.json << 'EOF'
{
  "agent_id": "agent-001",
  "action": {
    "type": "read_database",
    "target_resource": "users_table"
  },
  "timestamp": "2026-07-23T10:00:00Z"
}
EOF

opa eval -d agent_policy.rego -d data.json -i test_input.json "data.agentic.authorization.allow"
```

SCA pour les dépendances des agents IA : détecter les CVE avant production

Les frameworks d'agents IA accumulent des centaines de dépendances transitives. LangChain 0.2.x, par exemple, dépend de 47 packages directs et plus de 300 packages transitifs. En 2025, plusieurs CVE critiques ont été découvertes dans les frameworks d'agents : CVE-2024-46946

(LangChain, RCE via injection de template), CVE-2024-27441 (LlamaIndex, SSRF). Le SCA (Software Composition Analysis) automatique en CI/CD est non négociable.

```yaml
 # .github/workflows/sca-agent-deps.yml
# Pipeline SCA pour les dépendances d'agents IA

name: SCA Scan — Agent IA Dependencies

on:
  push:
    paths:
      - 'requirements*.txt'
      - 'pyproject.toml'
      - 'package*.json'
  pull_request:
  schedule:
    - cron: '0 6 * * *'  # Scan quotidien pour les nouvelles CVE

jobs:
  trivy-sca:
    name: Trivy SCA Scan
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Trivy : scan multi-format (Python requirements, npm, Cargo, Go)
      - name: Trivy — Scan dépendances agents IA
        uses: aquasecurity/trivy-action@master
        with:
          scan-type: 'fs'
          scan-ref: '.'
          format: 'sarif'
          output: 'trivy-sca.sarif'
          severity: 'HIGH,CRITICAL'
          vuln-type: 'library'
          ignore-unfixed: false

      # Upload résultats vers GitHub Security
      - name: Upload résultats SARIF
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: trivy-sca.sarif

  owasp-dependency-check:
    name: OWASP Dependency Check
    runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v4

      - name: OWASP Dependency Check
        uses: dependency-check/Dependency-Check_Action@main
        with:
          project: 'Agent-IA'
          path: '.'
          format: 'HTML'
          args: >
            --enableRetired
            --failOnCVSS 7
            --suppression .owasp-suppressions.xml

      - name: Upload rapport HTML
        uses: actions/upload-artifact@v4
        with:
          name: dependency-check-report
          path: reports/

  pip-audit:
    name: pip-audit (Python spécifique)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: {python-version: '3.12'}

      - name: pip-audit — Audit dépendances PyPI
        run: |
          pip install pip-audit
          pip-audit -r requirements.txt --format=json -o audit-results.json
          # Bloquer si des CVE CVSS >= 7.0 sont trouvées
          python3 -c "
          import json, sys
          data = json.load(open('audit-results.json'))
          vulns = [v for d in data.get('dependencies', [])
                   for v in d.get('vulns', [])
                   if v.get('fix_versions')]
          if vulns:
              print(f'BLOCKING: {len(vulns)} vulnérabilités avec fix disponible')
              [print(f'  - {v["id"]}: {v["description"][:80]}') for v in vulns[:5]]
              sys.exit(1)
```

```
          print('SCA OK: aucune vulnérabilité critique')
          "
```

Comment implémenter Kyverno pour contrôler les pods agents Kubernetes ?

Kyverno est un admission controller Kubernetes natif (pas de Rego requis) qui évalue les ressources Kubernetes selon des politiques YAML déclaratives. Pour les déploiements d'agents IA sur Kubernetes, Kyverno impose des standards de sécurité uniformes : pas de conteneurs en root, limits de ressources obligatoires, ban des images non signées et restriction des mounts de secrets.

```
 # kyverno-agent-security-policy.yaml
# Politique Kyverno pour sécuriser les déploiements d'agents IA

apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: secure-ai-agent-deployments
  annotations:
    policies.kyverno.io/title: "Sécurité des pods Agents IA"
    policies.kyverno.io/description: >
      Enforce les bonnes pratiques de sécurité pour tous les pods
      déployés avec le label app.kubernetes.io/component=ai-agent

spec:
  validationFailureAction: Enforce
  rules:
    # Règle 1 : Non-root obligatoire
    - name: require-non-root-agent
      match:
        any:
          - resources:
              kinds: [Pod]
              selector:
                matchLabels:
                  "app.kubernetes.io/component": "ai-agent"
      validate:
        message: "Les pods agents IA doivent s'exécuter en non-root (runAsNonRoot: true)"
        pattern:
          spec:
            securityContext:
              runAsNonRoot: true
              runAsUser: ">= 1000"

    # Règle 2 : Limits de ressources obligatoires (prévient les runaway agents)
    - name: require-resource-limits-agent
      match:
        any:
          - resources:
              kinds: [Pod]
              selector:
                matchLabels:
                  "app.kubernetes.io/component": "ai-agent"
      validate:
```

```
    message: "Les agents IA doivent avoir des limits CPU et mémoire définies"
    pattern:
      spec:
        containers:
          - (name): "*-agent"
            resources:
              limits:
                cpu: "?*"
                memory: "?*"
              requests:
                cpu: "?*"
                memory: "?*"

  # Règle 3 : Bloquer les privileged containers
  - name: block-privileged-agent
    match:
      any:
        - resources:
            kinds: [Pod]
            selector:
              matchLabels:
                "app.kubernetes.io/component": "ai-agent"
    validate:
      message: "Les agents IA ne peuvent pas être privileged"
      pattern:
        spec:
          containers:
            - (name): "*"
              =(securityContext):
                =(privileged): "false"

  # Règle 4 : Network Policy obligatoire pour les namespaces agents
  - name: require-network-policy-agent-ns
    match:
      any:
        - resources:
            kinds: [Namespace]
            selector:
              matchLabels:
                "env.type": "ai-agents"
    preconditions:
      all:
        - key: "{{ request.operation }}"
          operator: Equals
```

```
            value: "CREATE"
      generate:
        apiVersion: networking.k8s.io/v1
        kind: NetworkPolicy
        name: default-deny-agent-namespace
        namespace: "{{ request.object.metadata.name }}"
        data:
          spec:
            podSelector: {}
            policyTypes: [Ingress, Egress]
            egress:
              - to: []  # Interdire tout trafic sortant sauf exceptions explicites
```

> En implémentation Governance-as-Code pour une fintech utilisant des agents LangChain pour l'analyse de transactions (contexte PCI DSS), les politiques OPA ont bloqué 23 tentatives d'injection de prompt lors des tests de pénétration, dont 3 qui auraient potentiellement exfiltré des données de cartes bancaires. L'audit de la chaîne de dépendances via Trivy a révélé 7 CVE HIGH dans les dépendances transitives de LangChain non signalées dans les releases notes. Le ROI calculé : 0 incident de sécurité lié aux agents en 6 mois de production vs 2,3 incidents/mois estimés sans governance, pour un investissement de 15 jours de mise en place initiale.
>
> *— Retour de mission Governance-as-Code fintech, juin 2026*

ROI mesurable de la Governance-as-Code pour les agents IA

Quantifier le ROI de la Governance-as-Code est essentiel pour convaincre les directions techniques et métier d'investir dans ces outils. Les métriques clés à mesurer avant et après implémentation :

```
 # Script de mesure du ROI Governance-as-Code
cat > /opt/governance/roi_metrics.sh << 'EOF'
#!/bin/bash
# Métriques ROI Governance-as-Code — à exécuter mensuellement

echo "=== MÉTRIQUES ROI GOVERNANCE-AS-CODE ==="
echo "Période : $(date -d 'last month' +%B\ %Y)"
echo ""

# 1. Vulnérabilités bloquées par SCA avant production
VULNS_BLOCKED=$(grep "BLOCKING" /var/log/ci/sca-*.log 2>/dev/null | wc -l)
echo "CVE bloquées avant production : $VULNS_BLOCKED"
echo "→ Coût évité (estimation NIST : 4x plus cher en prod) : $(($VULNS_BLOCKED * 800))€"

# 2. Policy violations bloquées par OPA
OPA_BLOCKS=$(grep '"allow": false' /var/log/opa/decisions.log 2>/dev/null | wc -l)
echo "Actions agents bloquées par OPA : $OPA_BLOCKS"

# 3. Kyverno admissions refusées
KYVERNO_DENIED=$(kubectl get events --all-namespaces -A 2>/dev/null |     grep "kyverno\|denied"
echo "Admissions Kubernetes refusées (Kyverno) : $KYVERNO_DENIED"

# 4. MTTR (Mean Time to Remediate) sur les politiques
echo ""
echo "=== MÉTRIQUES QUALITÉ ==="
echo "Politiques OPA en production : $(find /etc/opa/policies -name '*.rego' | wc -l)"
echo "Tests unitaires politiques : $(find /etc/opa/tests -name '*_test.rego' | wc -l)"
echo "Coverage des politiques : $(opa test /etc/opa/policies/ 2>&1 | grep PASS | wc -l) tests PAS
EOF

chmod +x /opt/governance/roi_metrics.sh
/opt/governance/roi_metrics.sh
```

Tableau de bord des politiques de gouvernance

Outil	Scope	Format politique	Intégration CI/CD	Effort déploiement
OPA (Open Policy Agent)	API, microservices, agents IA	Rego	GitHub Actions, GitLab CI	2-3 jours
Kyverno	Kubernetes uniquement	YAML natif K8s	kubectl apply, Helm	1 jour
Trivy SCA	Python, npm, Go, Rust, Java	Règles OPA/ Rego	GitHub Actions, GitLab, Jenkins	2h
OWASP DepCheck	Java, .NET, Python, npm	Suppressions XML	Maven, Gradle, CLI	4h
Semgrep	Code source (SAST)	YAML/patterns	GitHub Actions, pre-commit	4h

Questions fréquentes

Quelle est la différence entre OPA et Kyverno pour Kubernetes ?

Les deux outils peuvent contrôler les admissions Kubernetes, mais leurs approches diffèrent. OPA (via Gatekeeper) utilise le langage Rego pour exprimer des politiques complexes avec logique conditionnelle, et est plus flexible pour les cas avancés (validation croisée de ressources, intégration avec des données externes). Kyverno utilise un format YAML natif Kubernetes sans langage de requête externe, ce qui le rend plus accessible aux équipes Kubernetes sans expertise Rego. Kyverno offre également des fonctionnalités de mutation et de génération de ressources que OPA Gatekeeper ne propose pas nativement. En pratique, les deux outils peuvent coexister : OPA pour les politiques complexes côté API/services, Kyverno pour les politiques Kubernetes déclaratives.

Comment tester les politiques OPA avant déploiement en production ?

OPA dispose d'un framework de test intégré avec la commande `opa test`. Les tests unitaires s'écrivent en Rego dans des fichiers `*_test.rego` et utilisent la convention `test_nom_du_test` pour les règles à tester. L'intégration dans les pipelines CI/CD via `opa test --coverage ./policies/` retourne un code de sortie non-nul en cas d'échec, bloquant le merge. Le coverage des branches Rego est affiché, permettant d'identifier les cas non couverts par les tests. L'outil `opa check` valide la syntaxe Rego sans exécuter les tests.

Comment gérer les faux positifs dans les scans SCA des agents IA ?

Les faux positifs SCA (vulnérabilités reportées mais non exploitables dans le contexte d'utilisation) sont gérés via des fichiers de suppression. OWASP Dependency-Check utilise un fichier XML de suppressions, Trivy supporte un fichier `.trivyignore`, et pip-audit permet des suppressions via `pip-audit --ignore-vuln CVE-XXXX-XXXXX`. Chaque suppression doit être documentée avec une justification (CVE non applicable car le code vulnérable n'est pas dans le chemin d'exécution, fix en cours, etc.) et une date d'expiration. Les suppressions permanentes sans justification constituent un anti-pattern DevSecOps sévère.

Peut-on utiliser Governance-as-Code avec des agents LangGraph ou CrewAI ?

Oui, OPA s'intègre avec n'importe quel framework d'agents IA. Pour LangGraph et CrewAI, le pattern recommandé est d'intercepter les appels d'outils (tool calls) avant exécution via un middleware custom qui consulte OPA. LangChain propose un mécanisme de callbacks (`BaseCallbackHandler`) qui permet d'intercepter chaque tool call. CrewAI supporte des hooks similaires via son système de Crew et Task hooks. OPA retourne une décision JSON (`{"allow": true/false}`) en moins de 1ms, l'impact sur la latence des agents est négligeable.

Comment implémenter le principe du moindre privilège pour les agents IA accédant à une base de données ?

Chaque agent IA doit disposer d'un compte de service applicatif avec des droits strictement limités à ses fonctions. Pour un agent de lecture d'analytics, un compte SELECT-only sur les

tables spécifiques suffit. Pour un agent de génération de rapports, READ sur les vues exposées uniquement. Les connexions de base de données des agents doivent passer par un proxy (PgBouncer, ProxySQL) qui trace toutes les requêtes. Le principe de segregation des devoirs (SoD) s'applique : aucun agent ne doit avoir à la fois des droits de lecture sur les données sensibles et des droits d'écriture vers des systèmes externes. Un agent compromis par prompt injection ne peut ainsi exfiltrer que les données auxquelles il accède légitimement.

Tests unitaires des politiques OPA avec opa test

Les politiques OPA doivent être couvertes par des tests unitaires avant déploiement en production. OPA intègre un framework de test natif avec la commande `opa test`. Les tests s'écrivent en Rego dans des fichiers `*_test.rego` et utilisent des données JSON simulées pour valider le comportement de chaque règle. Le coverage OPA mesure le taux de branches Rego couvertes par les tests, similaire au coverage de code source classique.

```
 # Fichier de test pour la politique agent (agent_policy_test.rego)
cat > agent_policy_test.rego << 'EOF'
package agentic.authorization

import future.keywords.if

# Test 1 : action autorisée dans les règles
test_allow_read_database if {
    allow with input as {
        "agent_id": "agent-reader",
        "action": {"type": "read_database", "target_resource": "users_view"}
    } with data.agents as {
        "agent-reader": {"permissions": ["read_database"]}
    } with data.action_log as []
    with data.sensitive_resources as ["salary_table"]
}

# Test 2 : blocage des actions d'exécution shell
test_deny_shell_exec if {
    not allow with input as {
        "agent_id": "agent-reader",
        "action": {"type": "exec_shell", "target_resource": "/bin/bash"}
    } with data.agents as {
        "agent-reader": {"permissions": ["read_database", "exec_shell"]}
    } with data.action_log as []
    with data.sensitive_resources as []
}

# Test 3 : blocage accès données sensibles
test_deny_sensitive_data if {
    not allow with input as {
        "agent_id": "agent-reader",
        "action": {"type": "read_database", "target_resource": "salary_table"}
    } with data.agents as {
        "agent-reader": {"permissions": ["read_database"]}
    } with data.action_log as []
    with data.sensitive_resources as ["salary_table"]
}
EOF

# Exécuter les tests
opa test . --verbose
# Output :
```

```
# PASS: test_allow_read_database (1.2ms)
# PASS: test_deny_shell_exec (0.8ms)
# PASS: test_deny_sensitive_data (0.9ms)
# ---
# PASS: 3/3 (0s)

# Coverage des politiques
opa test . --coverage | python3 -m json.tool | grep -A2 'coverage'
```

Intégration Governance-as-Code dans le pipeline GitOps

La Governance-as-Code s'intègre naturellement dans un workflow GitOps (Argo CD, Flux CD) où toutes les ressources, y compris les politiques de sécurité, sont définies dans des dépôts Git et appliquées automatiquement au cluster. Les politiques Kyverno et les configurations OPA sont commited dans Git, reviewées via Pull Request avec les tests automatiques, puis synchronisées vers le cluster par l'opérateur GitOps.

```
 # Application Argo CD pour synchroniser les politiques Kyverno
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: security-policies
  namespace: argocd
spec:
  project: security
  source:
    repoURL: https://github.com/mon-org/k8s-policies.git
    targetRevision: main
    path: policies/kyverno/
  destination:
    server: https://kubernetes.default.svc
    namespace: kyverno
  syncPolicy:
    automated:
      prune: true      # Supprimer les politiques retirées du Git
      selfHeal: true   # Re-appliquer si modifiées manuellement
    syncOptions:
      - CreateNamespace=true
      - ServerSideApply=true
```

La Governance-as-Code pour Agentic AI s'appuie sur les mêmes outils que le DevSecOps classique, mais avec des politiques adaptées aux spécificités des agents IA (rate limiting des tool calls, contrôle des accès aux données sensibles, audit des actions autonomes). Pour aller plus loin sur la sécurisation des pipelines qui déploient ces agents, l'article sur les attaques CI/CD et la sécurité des pipelines GitHub est complémentaire. L'intégration Kubernetes de ces politiques s'appuie sur les outils de sécurité Kubernetes présentés dans notre guide dédié. La conformité NIS 2 pour les systèmes d'IA impose des exigences de traçabilité des actions des agents que la Governance-as-Code fournit nativement via les audit logs OPA et Kyverno.

La Governance-as-Code pour les agents IA est un domaine en évolution rapide. Les frameworks émergents comme LangChain v0.3 et LlamaIndex v0.12 intègrent progressivement des hooks natifs pour l'auditabilité et le contrôle des actions d'agents, simplifiant l'intégration avec OPA. La CNCF publie des guidelines spécifiques sur la sécurisation des pipelines d'IA via son groupe de travail Security TAG. Pour les équipes DevSecOps qui débutent avec la Governance-as-Code IA, le point d'entrée recommandé est l'implémentation des tests SCA avec `pip-audit` et `trivy` sur les dépendances des agents, avant de progresser vers les politiques OPA et les admission

controllers Kyverno. La documentation officielle OPA sur openpolicyagent.org et le OWASP Top 10 pour les LLM constituent les deux références fondamentales pour construire une stratégie de gouvernance IA solide.

En 2026, la Governance-as-Code pour les agents IA n'est plus une option mais une obligation pour les organisations déployant des systèmes d'IA autonomes dans des environnements réglementés. L'AI Act européen, entré en force en 2025, impose des exigences de traçabilité et d'audit des systèmes d'IA à haut risque qui correspondent exactement aux capacités de Governance-as-Code présentées dans ce guide. Les politiques OPA, les tests Kyverno et les scans SCA automatisés constituent des preuves techniques exploitables lors des audits de conformité réglementaire.

