



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Policy Engines for Kubernetes: Picking One Without Losing Your Mind

Nabarun Pal, Broadcom



Raise your hand

... if you've ever had a **pod running as root** in **production** that shouldn't have been there.

... if you found out because of an incident,
not because of a policy.

The Deceptive Simplicity

- Block privileged containers? Check a field.
- Require resource limits? Another field.
- Enforce naming conventions? Easy, right?

... Then reality hits

- 42 teams deploying independently
- 3 different CI/CD pipelines
- That one legacy app that **needs** privileged mode
- The audit that asks *prove no PII was ever exposed*



KubeCon



CloudNativeCon

Europe 2026

Real-World Scenarios



Compliance Requirements

Framework	Common Kubernetes Requirements
SOC2	Access controls, audit logging, change management
PCI-DSS	Network segmentation, encryption, access restrictions
HIPAA	Data protection, audit trails, access controls
GDPR	Data residency, right to erasure, processing limits

"We have a policy" is nice

"Here's the audit log showing 847 blocked violations"
is what passes audits

Policy engines provide the ***evidence*** auditors need

Cost Control

Before policy engines,

- Average pod memory request: 2Gi
- Average pod memory usage: 340Mi
- Wasted capacity: 83%

Cost Control

After policy engines,

- Required justification for requests > 512Mi
- Automatic right-sizing recommendations
- Wasted capacity: 31%

Operational Sanity

Without policies,

- ✗ Pod crash: no readiness probe
- ✗ Deployment stuck: OOMKilled
- ✗ Can't reproduce: no image tag
- ✗ Security scan: 47 CVEs

Operational Sanity

With policies,

- ✓ Probe required at admission
- ✓ Limits enforced automatically
- ✓ Tags required, digests preferred
- ✓ Vulnerability scan gates deploy



KubeCon



CloudNativeCon

Europe 2026

The Parameters That Actually Matter



Policy Language & Expressiveness

Consideration	Questions to Ask
Learning curve	How long until a new team member writes their first policy?
Debugging	Can I test policies before deploying? How do I troubleshoot failures?
Reusability	Can I parameterize policies? Create libraries?
Limitations	What policies are impossible or impractical to express?

Operational Overhead

- ❑ How many new components am I deploying?
- ❑ What happens when the policy engine is unavailable?
- ❑ How do I upgrade without breaking existing policies?
- ❑ What is the memory/CPU footprint at scale?
- ❑ How do I backup and restore policy state?
- ❑ What is the disaster recovery story?

Failure Modes

failurePolicy: Fail

- Policy engine down = cluster operations blocked
- Maximum security guarantee
- Can cause cluster-wide outages

failurePolicy: Ignore

- Policy engine down = policies not enforced
- Cluster keeps functioning
- Security gap during outages

Another Failure Mode: Global Timeout Trap

Kubernetes has a **30-second global timeout** for admission webhooks.

If you have multiple webhooks (policy engine + cert-manager + others), each with 10-second timeouts, you can exceed the global limit and fail pod admission even with `failurePolicy: Ignore`

Integration & Ecosystem

- CI/CD integration: Can I validate policies in my pipeline before merging?
- GitOps compatibility: Does it work with Argo CD, Flux?
- Observability: Prometheus metrics? Audit logs? Policy reports?
- Multi-cluster: Can I manage policies across clusters?

Auditability & Reporting

- Historical record of policy decisions
- Evidence of enforcement (blocked violations)
- Ability to generate compliance reports
- Integration with audit logging infrastructure



KubeCon



CloudNativeCon

Europe 2026

The main choices





KubeCon



CloudNativeCon

Europe 2026

Kubernetes Native



ValidatingAdmissionPolicy (VAP)

- **Status:** GA in Kubernetes 1.30
- **Language:** CEL (Common Expression Language)

`ValidatingAdmissionPolicy` runs validation logic directly in the API server.

No webhooks. No external dependencies. No network calls.

Example: Block Privileged Pods

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: deny-privileged-containers
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:  [""]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources:  ["pods"]
  validations:
    - expression: >-
      object.spec.containers.all(c,
        !has(c.securityContext) ||
        !has(c.securityContext.privileged) ||
        c.securityContext.privileged == false
      )
      message: "Privileged containers are not allowed"
      reason: Forbidden
```

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: deny-privileged-containers-binding
spec:
  policyName: deny-privileged-containers
  validationActions: [Deny, Audit]
  matchResources:
    namespaceSelector:
      matchExpressions:
        - key: policy-enforcement
          operator: NotIn
          values: ["disabled"]
```

What CEL Can Do

- **Memory safe:** No buffer overflows
- **Terminating:** Guaranteed to complete
- **Side-effect free:** Pure evaluation, no external calls
- **Linear time:** Worst-case complexity is predictable

What CEL Cannot Do

1. **External data access:** Cannot query other resources during validation. You only see the object being admitted.
2. **Complex string operations:** No base64 encoding/decoding, no JSON parsing within strings, limited date/time handling.
3. **HTTP calls:** Cannot call external services for validation.
4. **Mutation:** ValidatingAdmissionPolicy is validation only. MutatingAdmissionPolicy adds mutation support.
5. **Image verification:** No built-in support for verifying container image signatures.

Cost-Based Restrictions

CEL has an estimated cost system. Complex expressions on unbounded data (like iterating over all annotations without limits) can exceed cost thresholds:

This might fail cost estimation if annotations are unbounded:

expression: >-

```
object.metadata.annotations.all(key, key.startsWith("allowed-"))
```

Solution: Add size limits in CRD schemas (`maxProperties`, `maxLength`) to reduce estimated cost.

When to Use Native Policies

- ✓ Simple validation rules (check field values, require labels)
- ✓ You want zero additional components
- ✓ Performance is critical (no webhook latency)
- ✓ You're on Kubernetes 1.30+

When NOT to Use Native Policies

- ✗ You need mutation with complex JSON-patch logic (*basic mutation now works via `MutatingAdmissionPolicy` beta in K8s 1.34+*)
- ✗ You need to verify image signatures
- ✗ You need to query cluster state during validation
- ✗ You need complex logic requiring external data

MutatingAdmissionPolicy (going GA in 1.36)

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingAdmissionPolicy
metadata:
  name: add-default-labels
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["CREATE"]
        resources: ["pods"]
  mutations:
    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: >-
          Object{
            metadata: Object.metadata{
              labels: {"managed-by": "platform-team"}
            }
          }
```



KubeCon



CloudNativeCon

Europe 2026

Kyverno



Why Platform Teams Love Kyverno

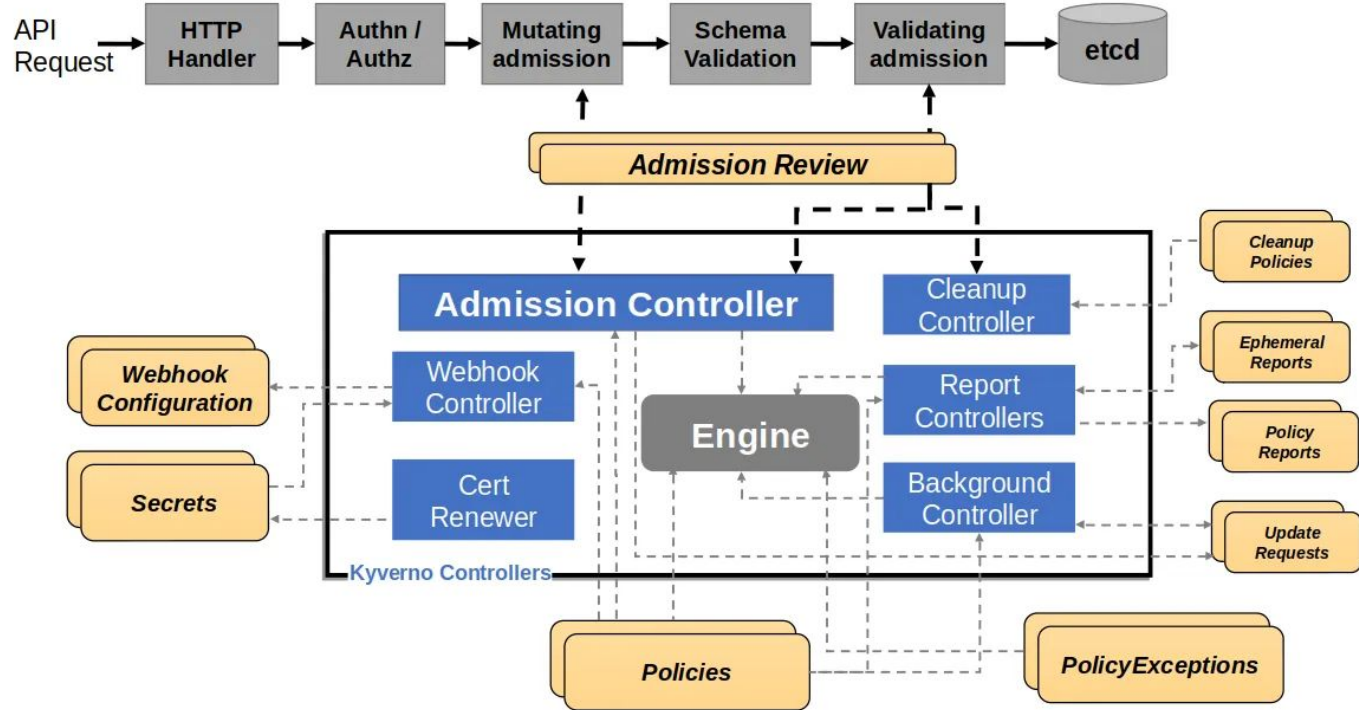
Kyverno's killer feature: **It's just YAML.** Just like native VAP, but more powerful.

Why Platform Teams Love Kyverno

Kyverno's killer feature: **It's just YAML.** Just like native VAP, but more powerful.

No new language. No compilation step. No separate testing framework. If you can write Kubernetes manifests, you can write Kyverno policies.

Architecture



Core Policy Types: **Validate**

```
apiVersion: policies.kyverno.io/v1
kind: ValidatingPolicy
metadata:
  name: require-labels
spec:
  validationActions:
    - Deny # or Audit
  matchConstraints:
    resourceRules:
      - apiGroups: ['']
        apiVersions: ['v1']
        operations: ['CREATE', 'UPDATE']
        resources: ['pods']
  validations:
    - message: "The label 'team' is required."
      expression: >-
        'team' in object.metadata.labels.orValue({}) &&
        size(object.metadata.labels['team']) > 0
```

Core Policy Types: Mutate

```
[...]
```

```
mutations:
```

```
- patchType: ApplyConfiguration
```

```
  applyConfiguration:
```

```
    expression: |
```

```
      Object{
```

```
        spec: Object.spec{
```

```
          containers: object.spec.containers.map(c,
```

```
            Object.spec.containers{
```

```
              name: c.name,
```

```
              resources: Object.spec.containers.resources{
```

```
                requests: Object.spec.containers.resources.requests{
```

```
                  memory: c.?resources.?requests.?memory.orValue("64Mi"),
```

```
                  cpu:      c.?resources.?requests.?cpu.orValue("100m")
```

```
                }}}}}
```



KubeCon



CloudNativeCon

Europe 2026

Core Policy Types: Generate

```
apiVersion: policies.kyverno.io/v1
kind: GeneratingPolicy
metadata:
  name: create-network-policy
spec:
  evaluation:
    synchronize:
      enabled: true
  matchConstraints: [...]
  variables:
    - name: nsName
      expression: 'object.metadata.name'
    - name: downstream
      expression: |
        [{"kind": dyn("NetworkPolicy"), "apiVersion": dyn("networking.k8s.io/v1"),
          "metadata": dyn({"name": "default-deny", "namespace": string(variables.nsName)}),
          "spec": dyn({"podSelector": dyn({}), "policyTypes": dyn(["Ingress", "Egress"])}]}]

generate:
  - expression: generator.Apply(variables.nsName, variables.downstream)
```

Image Verification

```
[...]  
  
matchImageReferences:  
  - glob: 'ghcr.io/myorg/*'  
  
attestors:  
  - name: cosign  
    cosign:  
      keyless:  
        identities:  
          - subject: 'https://github.com/myorg/*'  
            issuer: 'https://token.actions.githubusercontent.com'  
        ctlog:  
          url: 'https://rekor.sigstore.dev'  
  
validationConfigurations:  
  mutateDigest: true  
  verifyDigest: true  
  
validations:  
  - expression: >=  
    images.containers.map(image,  
      verifyImageSignatures(image, [attestors.cosign])  
    ).all(e, e > 0)  
  message: "Image must be signed by the CI pipeline."
```

Background Scanning

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: check-deprecated-apis
spec:
  validationFailureAction: Audit # Don't block, just
report
  background: true # Enable background scanning
  rules:
    - name: check-ingress-version
      match:
        any:
          - resources:
              kinds:
                - Ingress
      validate:
        message: "Ingress should use networking.k8s.io/v1"
        deny:
          conditions:
            - key: "{{request.object.apiVersion}}"
              operator: Equals
              value: "extensions/v1beta1"
```

Unlike admission-only enforcement,
Kyverno continuously scans existing
resources.

Background Scanning

```
apiVersion: wgpolicyk8s.io/v1alpha2
kind: PolicyReport
metadata:
  name: polr-ns-default
  namespace: default
results:
  - message: "validation rule 'check-team-label' passed"
    policy: require-labels
    rule: check-team-label
    result: pass
    source: kyverno
  - message: "The label 'team' is required."
    policy: require-labels
    rule: check-team-label
    result: fail
    source: kyverno
```

Results are written to PolicyReports

Production Considerations

High Availability

```
replicaCount: 3  
  
resources:  
  limits:  
    memory: 512Mi  
  requests:  
    memory: 256Mi  
    cpu: 100m
```

Failure Mode

```
# Recommendation: Start with Ignore,  
move to Fail once stable  
failurePolicy: Ignore # or Fail
```

The Goods and the Tradeoffs

- Single controller pod (resource efficient)
- Validation, mutation, generation in one place
- Native Kubernetes experience
- Policy testing with **kyverno**
apply CLI
- Kubernetes-only (no use outside K8s)
- YAML-based policies can get verbose for complex logic



KubeCon



CloudNativeCon

Europe 2026

Gatekeeper/OPA



Clearing up the confusion

Open Policy Agent (OPA)

- General-purpose policy engine
- Rego language
- Works anywhere (not K8s specific)

Gatekeeper

- Kubernetes-native OPA integration
- CRD-based policy management
- Constraint Framework
- Audit capabilities

Related Projects

- Conftest: Policy testing for configs
- Managed Distribution: Controllers running inside provider implementations

Rego: What It Actually Looks Like

```
package k8srequiredlabels

violation[{"msg": msg, "details": {"missing_labels": missing}}] {
    provided := {label | input.review.object.metadata.labels[label]}
    required := {label | label := input.parameters.labels[_]}
    missing := required - provided

    # Only produce a violation if something is missing
    count(missing) > 0

    # Format the message
    msg := sprintf("You must provide labels: %v", [missing])
}
```

Rego: Key Concepts

- Rules produce sets or values: `violation[ {...} ]` adds to a set
- All statements in a rule must be true: Like implicit AND
- `input` contains the request: `input.review.object` is the resource
- `input.parameters` contains constraint config

The Constraint Framework: ConstraintTemplate

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema: [...]
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: [...]
```

The Constraint Framework: Constraint

```
apiVersion: constraints.gatekeeper.sh/v1beta1
```

```
kind: K8sRequiredLabels
```

```
metadata:
```

```
  name: pods-must-have-team
```

```
spec:
```

```
  match:
```

```
    kinds:
```

```
      - apiGroups: [""]
```

```
        kinds: ["Pod"]
```

```
        namespaces: ["production", "staging"]
```

```
parameters:
```

```
  labels: ["team", "cost-center"]
```

Rego: Advanced Concepts

```
apiVersion: config.gatekeeper.sh/v1alpha1
```

```
kind: Config
```

```
metadata:
```

```
  name: config
```

```
spec:
```

```
  sync:
```

```
    syncOnly:
```

- group: ""
 version: "v1"
 kind: "Namespace"
- group: "networking.k8s.io"
 version: "v1"
 kind: "Ingress"

Rego: Advanced Concepts

```
violation[{"msg": msg}] {  
    # Check if ingress hostname conflicts with existing ingresses  
    input.review.kind.kind == "Ingress"  
    host := input.review.object.spec.rules[_].host  
  
    # Query synced data  
    other := data.inventory.namespace[ns][otherapiversion]["Ingress"][name]  
    other.spec.rules[_].host == host  
  
    # Ensure it's not the same ingress  
    other.metadata.uid != input.review.object.metadata.uid  
  
    msg := sprintf("Ingress host '%v' already in use", [host])  
}
```



KubeCon



CloudNativeCon

Europe 2026

Beyond Kubernetes Admission

This is OPA's *superpower*. **It works everywhere!**

Beyond Kubernetes Admission

Kubernetes Admission → Gatekeeper

Terraform Plans → Conftest + OPA

CI/CD Pipelines → OPA evaluation

Microservice AuthZ → OPA sidecar

API Gateway → Envoy + OPA

Infrastructure as Code → Pre-commit hooks

Gatekeeper Production Considerations

Resource Footprint

```
replicaCount: 3
resources:
  limits:
    memory: 512Mi
  requests:
    memory: 256Mi
    cpu: 100m

audit:
  # How often to scan existing resources
  interval: 3600 # seconds
  # Limit violations stored per constraint
  constraintViolationsLimit: 20
```

Learning Curve Reality

Task	O(Competency)
Basic constraints	1-2 days
Custom templates with Rego	1-2 weeks
Advanced data replication	2-4 weeks
Debugging complex policies	Ongoing

The Goods and the Tradeoffs

- Unified policy across layers
- Complex policies requiring full programming language
- Need to reference other cluster resources in decisions
- Good if organization already using OPA elsewhere
- Cloud provider managed options available
- Team unfamiliar with Rego and limited time to learn
- Mutation is a primary requirement (Gatekeeper supports it but *)
- Image signature verification needed is not built-in



KubeCon



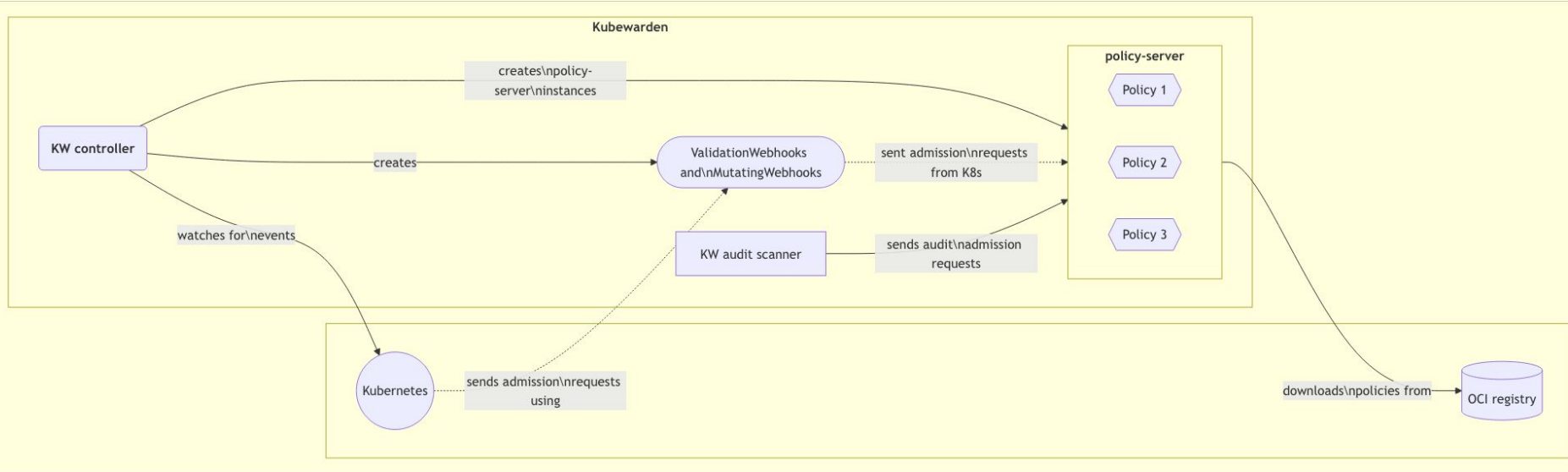
CloudNativeCon

Europe 2026

Kubewarden



Architecture



KubeCon



CloudNativeCon

Europe 2026

Language Flexibility: Rust

```
fn validate(payload: &[u8]) -> CallResult {
    let validation_request: ValidationRequest<Settings> =
        ValidationRequest::new(payload)?;

    match serde_json::from_value::<apicore::Pod>(validation_request.request.object) {
        Ok(pod) => {
            for container in pod.spec.containers.unwrap_or_default() {
                if container.security_context
                    .and_then(|sc| sc.privileged)
                    .unwrap_or(false)
                {
                    return kubewarden::reject_request(
                        Some("Privileged containers are not allowed".to_string()),
                        None,
                        None,
                        None,
                    );
                }
            }
            kubewarden::accept_request()
        }
        Err(_) => kubewarden::accept_request(),
    }
}
```



KubeCon



CloudNativeCon

Europe 2026

Language Flexibility: Go

[...]

```
func validate(payload []byte) ([]byte, error) {  
    var request kubewarden_protocol.ValidationRequest  
    if err := json.Unmarshal(payload, &request); err != nil {  
        return kubewarden.RejectRequest(  
            kubewarden.Message("Invalid request"),  
            kubewarden.Code(400))  
    }  
  
    // Policy logic here  
  
    return kubewarden.AcceptRequest()  
}
```

CEL and Rego

[...]

```
func validate(payload []byte) ([]byte, error) {  
    var request kubewarden_protocol.ValidationRequest  
    if err := json.Unmarshal(payload, &request); err != nil {  
        return kubewarden.RejectRequest(  
            kubewarden.Message("Invalid request"),  
            kubewarden.Code(400))  
    }  
  
    // Policy logic here  
  
    return kubewarden.AcceptRequest()  
}
```

Key Benefits: Distribution

```
apiVersion: policies.kubewarden.io/v1
kind: ClusterAdmissionPolicy
metadata:
  name: privileged-pods
spec:
  module: ghcr.io/your-registry/policies/pod-privileged:v0.3.0
  rules:
    - apiGroups: [""]
      apiVersions: ["v1"]
      resources: ["pods"]
      operations: ["CREATE", "UPDATE"]
  settings: {}
```

Key Benefits: Sandboxed Execution

- **Memory isolation:** Policies can't escape sandbox.
- **CPU limits:** Deterministic execution
- No filesystem or network access

Key Benefits: Language Choice

- Teams can use familiar languages
- Different teams can use different languages
- Leverage existing testing frameworks

Limitations: Compilation Complexity

- Build toolchain setup (rustup, TinyGo, etc.)
- Compilation to Wasm
- Publishing to registry
- More complex CI/CD pipeline
- TinyGo limitations: Has larger Wasm binary sizes than Rust, Not 100% Go standard library compatible

Limitations: Debugging Experience

- Debugging Wasm is harder than debugging Rego or CEL
- Limited runtime introspection
- printf debugging is common
- IDE support varies by language

Limitations: Ecosystem Maturity

- Smaller community than Kyverno or OPA
- Fewer pre-built policies available
- Less third-party tooling

The Goods and the Tradeoffs

- Teams with Rust expertise
- Complex policy logic benefiting from a full programming language
- Need for multi-language policy development
- Want to use existing OCI infrastructure for policy distribution
- High security requirements
- Simple policies where YAML would suffice
- Teams without systems programming experience
- Need for quick policy iteration
- Want large pre-built policy library



KubeCon



CloudNativeCon

Europe 2026

Comparison



	Native (VAP)	Kyverno	OPA/Gatekeeper	Kubewarden
Validation	✓	✓	✓	✓
Mutation	⚠ Beta (K8s 1.34+)	✓	⚠ Stable but via separate CRDs (not Rego)	✓
Generation	✗	✓	✗	✗
Image Verification	✗	✓	✗	✓
Background Scan	✗	✓	✓	✓
External Dependencies	None	Webhook	Webhook	Webhook
Policy Language	CEL	YAML/CEL	Rego	Any→Wasm
Learning Curve	Low	Low-Medium	High	Medium-High
Cross-Resource Checks	✗	Limited	✓	Limited
Beyond K8s	✗	✗	✓	✗



KubeCon

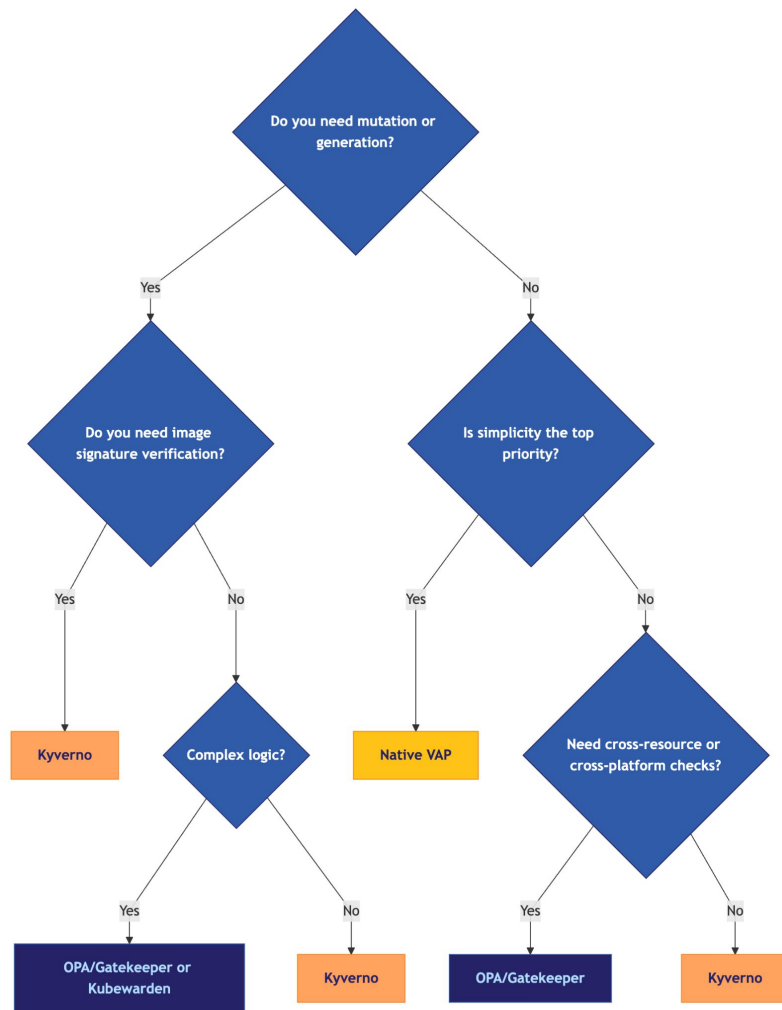


CloudNativeCon

Europe 2026

Decision Tree







KubeCon



CloudNativeCon

Europe 2026

Real-world Recommendations



Scenario 1: Platform Team at a Startup

Team Size: 5 engineers

Kubernetes experience: Moderate

Compliance requirements: SOC 2
incoming

Budget for tools: Limited

Recommendation: **Kyverno**

- Quick to adopt (YAML-native)
- Covers validation, mutation, generation
- PolicyReports for compliance
- Good docs, active community

Scenario 2: Enterprise Multi-Cloud

Team Size: 50+ engineers

Kubernetes experience: Expert

Compliance requirements: PCI-DSS, SOC 2

Existing tools: Terraform, multiple clouds - private/public/hybrid

Recommendation: **OPA/Gatekeeper**

- Unified policy language across stack
- Cloud provider integrations (Azure Policy, GKE)
- Mature, battle-tested
- Handles complex cross-resource policies

Scenario 3: Security-First Fintech

Team Size: 20 engineers

Kubernetes experience: Expert

Compliance requirements: Extensive

Primary Concern: Supply chain security

Recommendation:

- **Kyverno** (for image verification)
- **Native VAP** (for simple rules)
- Image signature verification built-in
- Background scanning for drift detection
- Native policies for performance-critical checks
- Layered approach

Scenario 4: Greenfield with Simple Needs

Team Size: 3 engineers

Kubernetes experience: Beginner

Compliance requirements: Basic

Primary Concern: Prevent common mistakes

Recommendation: **Native VAP**

- Zero additional components
- Built into K8s
- Covers common validation needs
- Graduate to Kyverno/OPA when needed



KubeCon



CloudNativeCon

Europe 2026

Migration Paths



Native VAP → Kyverno

- Kyverno supports CEL, can often port expressions directly
- Add mutation/generation as needed
- Run both during transition

Kyverno → OPA/Gatekeeper

- Port YAML patterns to Rego → more work
- Consider keeping Kyverno for mutation
- OPA has community-supported Kyverno policy converters

Any → Kubewarden

- Can import Rego policies (compiles to Wasm)
- CEL policies supported
- Gradual migration possible



KubeCon



CloudNativeCon

Europe 2026

Lessons That Don't Fit in Marketing Materials



Start in Audit Mode

Wrong approach:

`validationFailureAction: Enforce`

From day one

Right approach:

`validationFailureAction: Audit`

First 2-4 weeks

Analyze violations, fix false positives, then:

`validationFailureAction: Enforce`

Deploy in audit mode first. You will find:

- Legitimate workloads that need exceptions
- Policies that are too strict
- Edge cases you didn't consider

Exception Management is Half the Work

```
# Kyverno exception example
apiVersion: kyverno.io/v2
kind: PolicyException
metadata:
  name: allow-istio-privileged
  namespace: istio-system
spec:
  exceptions:
    - policyName: deny-privileged
      ruleNames:
        - check-privileged
  match:
    any:
      - resources:
          kinds:
            - Pod
          namespaces:
            - istio-system
```

Document every exception.

Review quarterly.

Exceptions accumulate.

Webhook Timeout Tuning

```
# Default: 10 seconds
```

```
# Problem:
```

```
Complex policies + slow network =  
timeouts
```

```
webhookTimeoutSeconds: 30 # Max allowed
```

Global Timeout Trap

Multiple webhooks compound

Consider policy complexity

Resource Limits for Policy Engines

```
# Kyverno in a 500-node cluster
resources:
  limits:
    memory: 2Gi  # Increased from default
  requests:
    memory: 512Mi
    cpu: 500m
```

Monitor memory usage

Policy engines cache data

Large clusters = more cache

Policy Testing in CI/CD

Kyverno CLI

```
kyverno apply policy.yaml --resource  
test-pod.yaml
```

OPA/Conftest

```
conftest test deployment.yaml -p policies/
```

Kubewarden

```
kwctl run policy.wasm --request-path  
test-request.json
```

Don't deploy untested policies

The "Break Glass" Procedure

```
# Option 1: Label namespace for exclusion (if  
configured)
```

```
kubectl label namespace emergency  
policy-enforcement=disabled
```

Document this

```
# Option 2: Scale down webhook (nuclear option)
```

```
kubectl scale deployment kyverno -n kyverno  
--replicas=0
```

Practice this

```
# Option 3: Delete webhook configuration (very  
nuclear)
```

```
kubectl delete validatingwebhookconfiguration  
kyverno-resource-validating-webhook-cfg
```

Don't learn during an incident

Closing Notes

1. Policy engines are infrastructure
2. There's no universal best choice
3. Start simple, grow as needed
4. Audit mode first
5. Plan for exceptions



KubeCon



CloudNativeCon

Europe 2026

Thank You!

Let's take the Q&A offline

Resources

- **Kubernetes VAP:**
<https://kubernetes.io/docs/reference/access-authn-authz/validating-admission-policy/>
- **Kyverno:** <https://kyverno.io/>
- **OPA Gatekeeper:** <https://open-policy-agent.github.io/gatekeeper/>
- **Kubewarden:** <https://kubewarden.io/>



KubeCon



CloudNativeCon

Europe 2026

