Architektur

Vier Controller, eine Policy-Engine

Admission-Controller (Pflicht): nimmt die Webhook-Callbacks des API-Servers an, fuehrt validate/mutate aus, prueft PolicyExceptions.
Background-Controller: generate- und mutate-existing-Regeln auf Bestandsressourcen (reconciled UpdateRequests).
Reports-Controller: erzeugt und pflegt PolicyReports.
Cleanup-Controller: loescht Ressourcen nach Zeitplan.
Kyverno gehoert in einen dedizierten Namespace – nie nach `kube-system`. CNCF-Graduated-Projekt (seit Maerz 2026).

Dynamische Webhooks

Kyverno generiert seine Validating-/MutatingWebhookConfigurations selbst aus den match-Bloecken der installierten Policies – nur gematchte Ressourcen erzeugen ueberhaupt Admission-Calls.
Zertifikate: self-signed, automatische Rotation (CA 1 Jahr, TLS 150 Tage, Renewal ca. 15 Tage vor Ablauf).
HA: 3 Replicas fuer den Admission-Controller; mehr Replicas bedeutet nicht bei jedem Controller mehr Durchsatz.

```
helm repo add kyverno https://kyverno.github.io/kyverno/
helm install kyverno kyverno/kyverno \
  -n kyverno --create-namespace
kubectl -n kyverno get deploy   # 4 Controller
kubectl get validatingwebhookconfigurations | grep kyverno
```

Policy-Typen (Stand 1.18)

CEL-Typen: policies.kyverno.io/v1

Seit 1.18 stable: `ValidatingPolicy`, `MutatingPolicy`, `GeneratingPolicy`, `DeletingPolicy`, `ImageValidatingPolicy`. Jeder Typ auch namespaced (`NamespacedValidatingPolicy` usw.). Ein CR pro Zweck, CEL statt JMESPath/-Pattern.

Legacy: ClusterPolicy / Policy

`kyverno.io/v1`: validate-, mutate-, generate- und verifyImages-Regeln in einem CR. `ClusterPolicy` cluster-weit, `Policy` namespaced.
Deprecated seit 1.18 (nur noch Critical Fixes in 1.18/1.19), Removal fuer 1.20 (Okt 2026) geplant – Neues in CEL-Typen schreiben, Bestand nach Migrations-Guide umziehen.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata: {name: require-ns-purpose-label}
spec:
  rules:
    - name: require-ns-purpose-label
      match:
        any:
        - resources: {kinds: [Namespace]}
      validate:
        failureAction: Enforce   # oder Audit
        message: "Label `purpose` ist Pflicht."
        pattern:
          metadata:
            labels:
              purpose: production
```

Validieren mit CEL

```
apiVersion: policies.kyverno.io/v1
kind: ValidatingPolicy
metadata: {name: check-labels}
spec:
  validationActions: [Deny]
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: [v1]
        operations: [CREATE, UPDATE]
        resources: [pods]
  validations:
    - message: label 'environment' ist Pflicht
      expression: >-
        'environment' in
        object.metadata.?labels.orValue([])
```

Enforcement & Modi

`validationActions`: `Deny` blockt, `Audit` protokolliert nur.
`spec.evaluation`: `admission` und `background` getrennt schaltbar; `mode: JSON` prueft beliebige JSON-/YAML-Payloads auch ausserhalb des Clusters (CI-Pipeline).
`matchConstraints` folgt der API der K8s ValidatingAdmissionPolicy.

Mutieren & Generieren

MutatingPolicy

`patchType: ApplyConfiguration` (Server-Side-Apply-Merge) oder `JSONPatch`.
Bestandsressourcen: `spec.evaluation.mutateExisting.enabled: true` – der Background-Controller mutiert dann bereits existierende Objekte, nicht nur Admission-Traffic.

```
apiVersion: policies.kyverno.io/v1
kind: MutatingPolicy
metadata: {name: add-label}
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: [v1]
        operations: [CREATE]
        resources: [pods]
  mutations:
    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: >
          Object{metadata: Object.metadata{
            labels: Object.metadata.labels{
              team: "platform"}}}
```

GeneratingPolicy

Erzeugt oder klont Ressourcen auf Trigger – Klassiker: Default-NetworkPolicy oder ResourceQuota bei jedem neuen Namespace. Legacy-generate-Regeln laufen ueber UpdateRequests des Background-Controllers.

Image-Verifikation (Supply Chain)

ImageValidatingPolicy: Sigstore & Notary

Attestors: Cosign (Public Key, Keyless via OIDC issuer/subject, Transparency Log, TUF) und Notary (X.509-Zertifikate, optional TSA).
Attestations: SBOM- und in-toto-Predicates pruefen via `verifyAttestationSignatures()` und `extractPayload()`.
`mutateDigest`: ersetzt mutable Tags durch immutable Digests.

```
apiVersion: policies.kyverno.io/v1
kind: ImageValidatingPolicy
metadata: {name: verify-images}
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: [v1]
        operations: [CREATE]
        resources: [pods]
  matchImageReferences:
    - glob: "registry.example.com/*"
  attestors:
    - name: cosign
      cosign:
        key:
          data: |
            -----BEGIN PUBLIC KEY-----
            ...
            -----END PUBLIC KEY-----
  validations:
    - expression: >-
        images.containers.map(image,
        verifyImageSignatures(image,
        [attestors.cosign])).all(e, e > 0)
      message: "Signatur-Check fehlgeschlagen"
```

Abgrenzung: K8s-native CEL-Policies

Was der API-Server selbst kann

ValidatingAdmissionPolicy: GA seit K8s 1.30. MutatingAdmissionPolicy: GA seit K8s 1.36.
CEL laeuft in-process im kube-apiserver: keine Webhooks, keine Zertifikate, keine Verfuegbarkeits-Abhaengigkeit von einem Policy-Deployment.

Was Kyverno darueber legt

Background-Scans auf Bestandsressourcen, PolicyReports, PolicyExceptions, generate/cleanup, Image-Signatur-Verifikation, JSON-Mode, Autogen.
`ValidatingPolicy` ist ein Superset der VAP – und generiert via `spec.autogen.validatingAdmissionPolicy.enabled: true` eine native VAP, die im API-Server laeuft: Kyverno als Management-Layer, Enforcement ohne Webhook-Hop.

PolicyException & Autogen

PolicyException

Gezielte Ausnahme statt aufgeweichter Policy: referenziert Policy + Regel(n) (Wildcard `*` ok) + Ressourcen-Match. Legacy-Engine: `kyverno.io/v2`; CEL-Typen: `policies.kyverno.io/v1alpha1`.
Default: deaktiviert. Aktivierung per Flag `enablePolicyException` + `exceptionNamespace` (nur dort duerfen Exceptions liegen) – Erstellung per RBAC und GitOps kontrollieren.

Autogen fuer Pod-Controller

Pod-Regeln werden automatisch auf Deployment, DaemonSet, StatefulSet, Job, ReplicaSet, ReplicationController und CronJob (via **`jobTemplate`**) ausgeweitet – eine Policy statt sieben.
CEL: **`spec.autogen.podControllers.controllers`**. Legacy: Annotation **`pod-policies.kyverno.io/autogen-controllers`** (**`none`** schaltet ab).

PolicyReports (Compliance)

Audit-Evidenz im Cluster

`wgpolicyk8s.io/v1alpha2`: **`PolicyReport`** (namespaced) und **`ClusterPolicyReport`** – offenes Format der Kubernetes Policy WG, nicht Kyverno-proprietaer.
Results: **`pass / fail / warn / error / skip`**. Gefuettert aus Admission-Requests und Background-Scans (Default: stuendlich) – deckt auch Bestandsressourcen ab, die nie durch Admission liefen.

```
kubectl get policyreport -A       # namespaced
kubectl get clusterpolicyreport   # cluster-scoped
kubectl get polr -o wide          # Kurzform
kubectl get polr <name> -o jsonpath=\
'{.results[?(@.result=="fail")]}'
```

Webhook-Betrieb

failurePolicy: bewusst entscheiden

Default Fail (fail-closed): ist Kyverno down, lehnt der API-Server alle gematchten Requests ab. Sicher, aber ein Verfuegbarkeits-Risiko – HA-Replicas und PDB einplanen.
Fail-open global per **`--forceFailurePolicyIgnore`**, besser pro Policy abwaegen: Security-Gates **`Fail`**, Komfort-Mutationen **`Ignore`**.

Performance-Tuning

Webhook-Timeout: Default 10s (1–30s) – knapp halten, haengende Admission bremst jedes Deployment.
Namespace-Filter: **`webhooks.namespaceSelector`** in der Kyverno-ConfigMap; **`kube-system`** und **`kyverno`** sind default ausgenommen.
resourceFilters **`[Kind,Namespace,Name]`**: letzte Verteidigungslinie hinter dem Webhook – gilt nicht fuer Background-Scans.
Match-Bloecke eng schneiden: was nicht matcht, erzeugt keinen Admission-Call.

Anti-Patterns

Was du nicht tun solltest

Neue Policies als ClusterPolicy: deprecated seit 1.18, Removal fuer 1.20 geplant – Neues gehoert in die CEL-Typen.
Enforce ohne Audit-Phase: neue Policies erst im Audit-Modus fahren, PolicyReports lesen, dann auf Deny/Enforce drehen.
Pauschal fail-open: **`Ignore`** ueberall hebelt Security-Policies bei jedem Kyverno-Ausfall aus.
Wildcard-Matches: jede Admission geht durch den Webhook – Latenz auf jedem API-Call, maximaler Blast-Radius bei Ausfall.
Exceptions ohne Governance: wer PolicyExceptions erstellen darf, hebelt Policies aus – **`exceptionNamespace`** + RBAC + Review-Pflicht.
Kyverno neben andere Workloads: dedizierter Namespace ist Vorgabe der Doku – nie nach **`kube-system`** deployen.

kyverno-cheatsheet.de

Kyverno Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

Policy-as-Code & Admission-Governance

OMNI52™ hilft Plattform-Teams, Kyverno fuer regulierte Enterprise-Umgebungen sauber aufzustellen.

Policy-Baseline nach Pod-Security-Standards, Migration von ClusterPolicy-Bestaenden auf die CEL-Engine, Exception-Governance mit RBAC und GitOps, Supply-Chain-Verifikation mit Sigstore, PolicyReports als Audit-Evidenz fuer Compliance-Nachweise. Output landet als GitOps-Repo bei euch im Cluster: Policies, Tests, Runbooks. Euer Team operiert weiter, nicht wir.

OMNI52 GmbH, Ebern · istio-consulting.de · kostenloses 30-min Initial-Assessment

Aus der OMNI52 Cheatsheet-Fabrik

Verwandte Sheets: kubernetes-cheatsheet.de (Kubernetes Core) · rke2-cheatsheet.de (RKE2-Distribution) · rancher-cheatsheet.de (Rancher-Management) · istio-cheatsheet.de (Service-Mesh-Layer).