



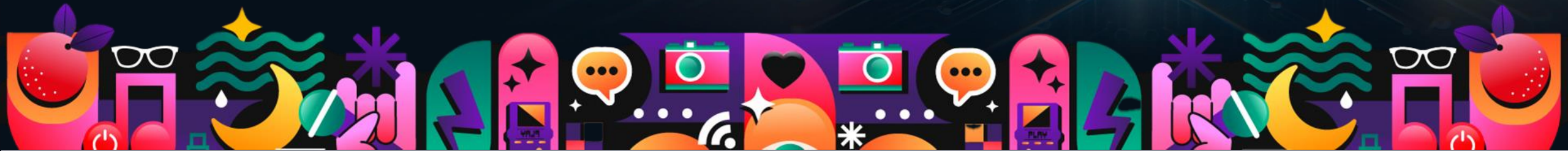
# KUBESPECTRA

## OPEN SOURCE INNOVATION – SECURE BY DESIGN

# Advanced Kyverno Workshop: Debugging, Testing und Überwachung

# Marie Padberg

**02.07.2025**



# Why KubeSpectra



We build, implement and develop Kubernetes-based container platforms.



**Containerization**



**DevOps**



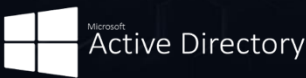
**Migration**

We enable the quick, secure and flexible use of applications in the cloud.



# Technology

UserManagement



Deployment



Security & Network



Observability



Storage



Source Code Management



Orchestration



# Team



Wolfgang Perzl  
Managing Director



Paul Schmidt  
Head of Technology



Marie Padberg  
DevOps Engineer



Alexander Preis  
DevOps Engineer



Nicole Philippczyk  
DevOps Engineer



Dr. Karol Palczynski  
DevOps Engineer



Emmanouil Goulidakis  
DevOps Engineer



Dr. Marcus Mende  
Business Development



Fabian Léglise  
DevOps Engineer



Andreas Platau  
DevOps Engineer



Christoph Seidel  
Lead DevOps Engineer



# Agenda

1. Schreiben von Kyverno Validation Policies
2. Testen der Policies
3. Reports der umgesetzten Policies

# Kyverno - Policy Engine



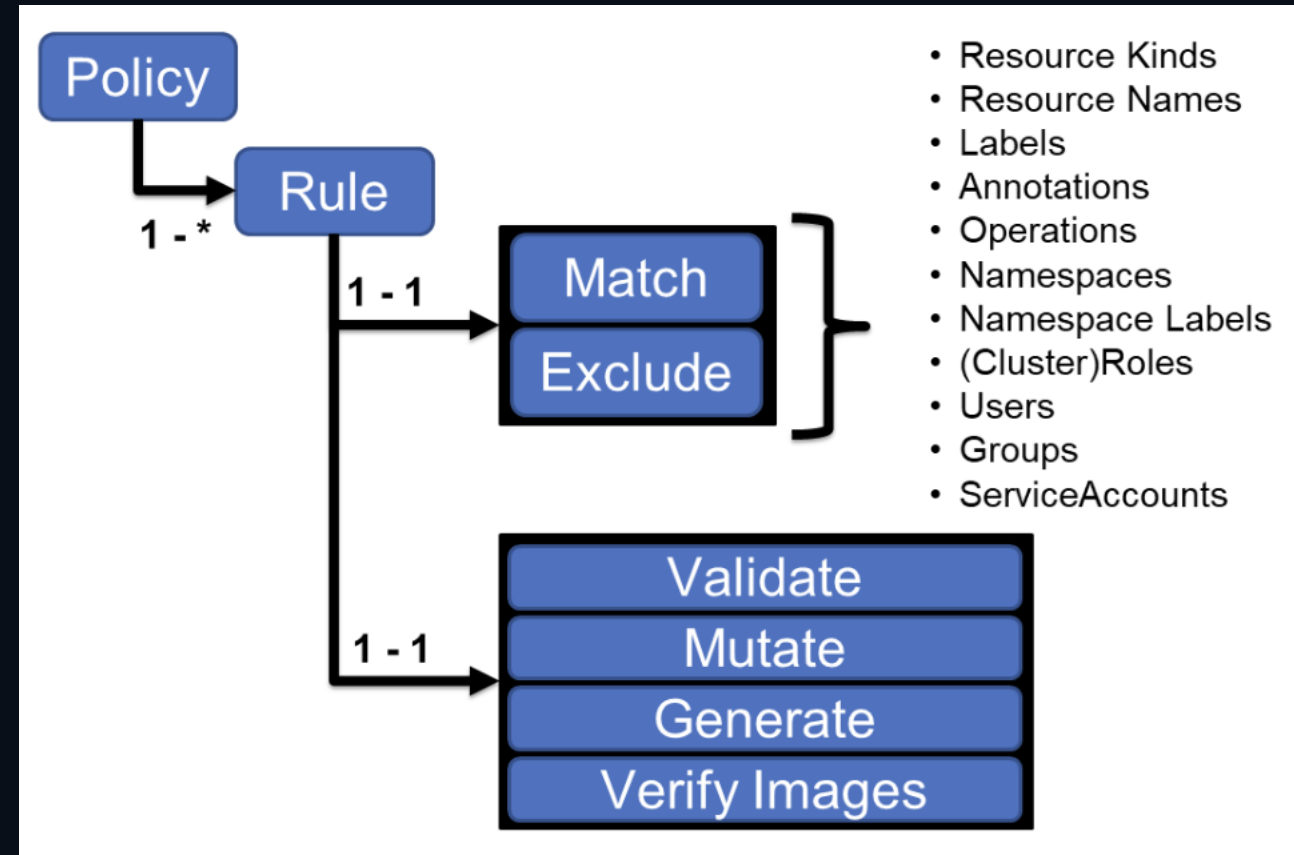
- Policy Engine für Kubernetes, die Richtlinien als Kubernetes Ressourcen managt
- Nutzt dynamische Kubernetes Admission Controller
- Kyverno umfasst 4 Arten von Richtlinien
  - Validierungen (validate)
  - Veränderungen (mutate)
  - Neue Kubernetes-Ressourcen basierend auf einer Richtlinie erstellen (generate)
  - Entfernen von Kubernetes Ressourcen (cleanup)
- Policy Reports bzgl. der Überprüfungen
- Anwendung auf sämtliche Kubernetes Objekte



# Kyverno - Policy Engine



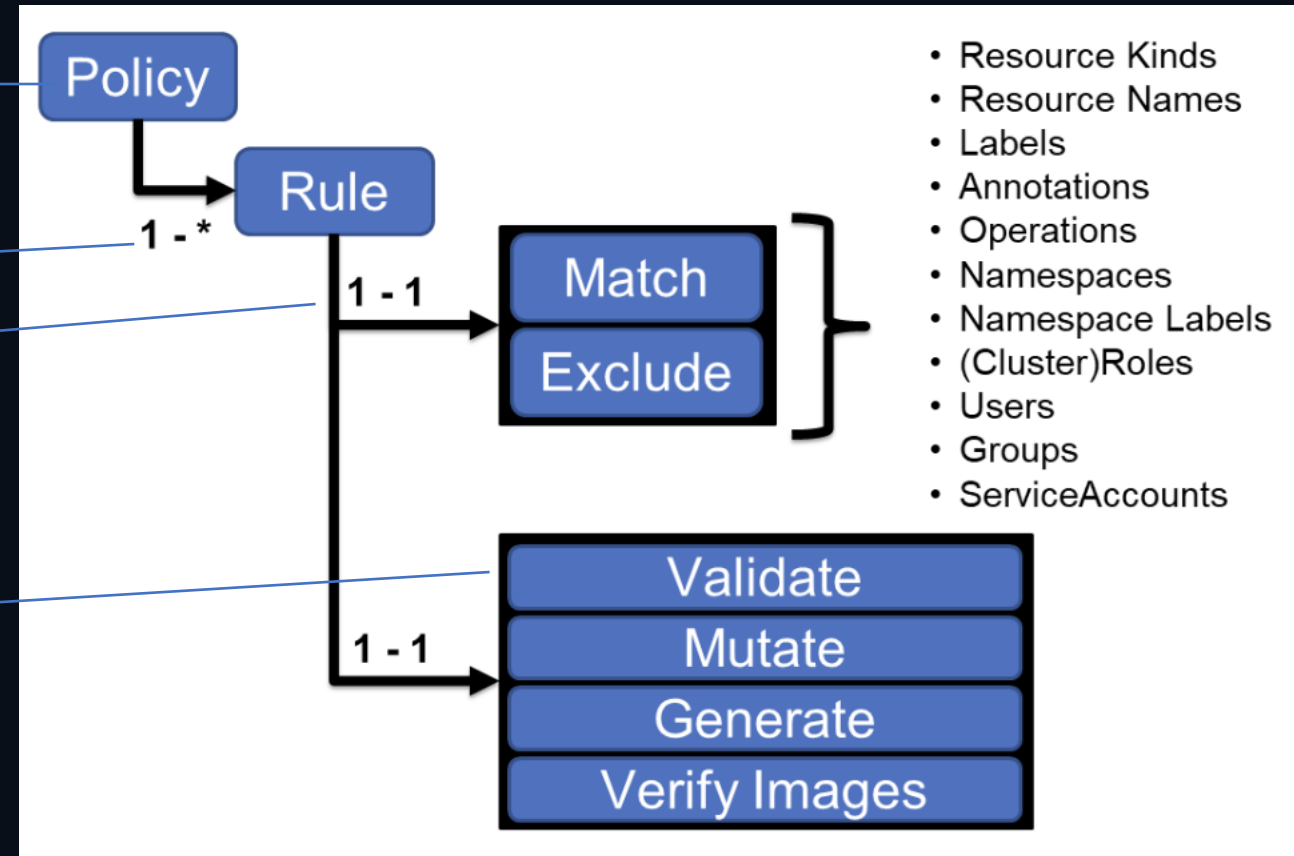
- Eine Kyverno Policy ist eine Sammlung von Regeln
- Jede Regel enthält eine match-Klausel und ggf. auch eine exclude-Klausel
- Clusterweit oder namespaceweit
- 2 Modi:
  - **Audit:** Verstöße gegen die Richtlinie werden vermerkt, sind aber ansonsten zulässig
  - **Enforce:** Verstöße gegen die Richtlinien führen zur Ablehnung



# Kyverno - Beispiel Validation Policy



```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
name: validate-image-tag
spec:
  rules:
  - name: deny-latest-tag
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      failureAction: Audit
      message: "Using the mutable image tag
'latest' is not allowed."
      pattern:
        spec:
          =(containers):
          - image: "!*:latest"
```







Was ist JMESPath?

- JMESPath (ausgesprochen „James path“)
- Eine deklarative Abfragesprache für JSON-Daten
- Ermöglicht gezielte Extraktion & Filterung von Datenstrukturen

Warum ist das in Kyverno wichtig?

- JMESPath ist das Werkzeug, um Werte aus Objekten zu extrahieren
- Unverzichtbar in match, exclude, preconditions, foreach und patterns
- Strukturüberprüfung durch pattern ist nicht immer ausreichend



## Typische Einsatzbereiche

- Zugriff auf Labels, Annotations, Container-Images etc.
- Ermöglicht komplexe Bedingungen: `request.object.metadata.labels.app == 'frontend'`
- Iterationen & Filter: `spec.[initContainers, containers][].image`
- Message-Feld, falls eine Policy eine Ressource blockiert

## Syntax-Kurzüberblick

- Immer `{{ }}` für JMESPath-Ausdrücke
- Nur `"{{...}}"`, wenn das Feld **nur** der Ausdruck ist, z.B. Label - `appns: "{{request.namespace}}"`
- Ausdruck + Text → Quotes optional, z.B. `message: "Namespace is {{request.namespace}}"`
- `.` = Sub-Expression → Zugriff auf verschachtelte Felder



Was bedeutet {{request.namespace}}?

1. Erstellung Pod: `kubectl run nginx --image=nginx`
2. Kubernetes API-Server verarbeitet den Request
  - a) Validiert die YAML-Datei
  - b) Fügt systemseitige Standardwerte hinzu (z.B. Timestamps)
  - c) Verpackt die Anfrage/Request in ein AdmissionReview Objekt und sendet es an dynamische Admission Webhooks weiter
3. Das AdmissionReview-Objekt enthält u.A.
  - a) Die ursprüngliche Ressource
  - b) Informationen über die Aktion (CREATE, DELETE,...)
  - c) Namespace
  - d) Wer den Request ausgelöst hat
  - e) ...





- Auszug aus einem AdmissionReview

```
{  
  "kind": "AdmissionReview",  
  "apiVersion": "admission.k8s.io/v1",  
  "request": {  
    "name": "nginx",  
    "namespace": "psa-test",  
    "operation": "CREATE",  
    "userInfo": {  
      "username": "ubernetes-admin",  
    },  
    "object": {  
      "kind": "Pod",  
      "apiVersion": "v1",  
      "metadata": {  
        "name": "nginx",  
        "creationTimestamp": "2025-06-29T12:58:35Z"  
      },  
    },  
  },  
}
```



- Vollständiger Pod wird im AdmissionReview zusammen mit Metadaten dargestellt
- AdmissionReview ist die wichtigste Datenquelle für JMESPath-Ausdrücke in Kyverno
- Besonders relevant ist `request.object` (der Pod selbst)
- Inhalt von `AdmissionReview.request` kann in Kyverno-Logs angezeigt werden (Aktivierung Debug-Mode)
  - Helm: *`features.dumpPayload.enabled: true`*
  - Flag *`dumpPayload=true`* im Admission-Controller Deployment setzen
- Sollte nur bei der Entwicklung von Policies oder der Fehlersuche verwendet werden und nicht dauerhaft aktiviert sein



- Nützlich, um das Verhalten von Policies zu validieren und zu testen
- *jp* ist ein Unterbefehl der Kyverno CLI
- Ermöglicht das Testen von JMESPath-Ausdrücken
- Liste der verfügbaren benutzerdefinierten JMESPath-Filter von Kyverno: `kyverno jp function`
- Weitere Beispiele:
  - *jp* → Einstieg in den "JMESPath"-Modus
  - *query* → führt den JMESPath-Ausdruck aus
  - `<JMESPath-Ausdruck>` → Filter oder Zugriffspfad im JSON-Dokument

```
kubectl get pod nginx -o json | kyverno jp query 'spec.containers[0].image'
```

```
kyverno jp query -i admissionreview.json 'request.namespace'
```



# Kyverno Validation Policies – Selecting Resources



## Match/Exclude

- Bestimmen, welche Ressourcen eine Regel betreffen (match) oder welche explizit ausgeschlossen werden (exclude)
- Beide Blöcke nutzen dieselbe Struktur: entweder **any** (OR-Logik) oder **all** (AND-Logik)
- Mögliche Filterelemente:
  - kinds: Ressourcen-Typen
  - namespaces: Namespaces
  - selector: Label-basierte Auswahl
  - namespaceSelector: Labels auf Namespaces
  - annotations: Filter auf Basis von Annotationen
  - operations: Auf welche Admisson-Aktionen (z. B. CREATE, UPDATE) die Regel wirken soll – optional, Standard: CREATE & UPDATE



# Kyverno Validation Policies – Selecting Resources

- Beispiel mit feingranularem Match-Statement

```
match:  
  any:  
    # AND across kinds and namespaceSelector  
    - resources:  
      # OR inside list of kinds  
      kinds:  
        - Deployment  
        - StatefulSet  
      operations:  
        - CREATE  
        - UPDATE  
      selector:  
        matchLabels:  
          app: critical
```

# Kyverno Validation Policies – Deny Rules



- Erweiterung von validate-Rules: neben Pattern-Checks gibt es auch **deny**-Blöcke mit Bedingungen
- Wenn eine deny-Bedingung wahr (true) wird, wird die Ressource blockiert (Admission Request wird abgelehnt)
- Innerhalb eines deny-Blocks:
  - **key**: JSON-Pfad oder Variablenzugriff (z. B. request.object.spec...).
  - **operator**: Vergleiche wie Equals, NotEquals, LessThan, AnyNotIn, ...
  - **value**: Vergleichswert, auch Listentypen
  - **message**: Optionale Fehlermeldung – wird im Admission-Response angezeigt

Unterschied zu Pattern-Validation:

- **Pattern** (Overlay): prüft auf Vorhandensein/Werte, validiert, blockiert bei Nicht-Vorhandensein
- **Deny**: explizite Ausschlussregel – wenn Bedingung zutrifft, wird geblockt





# Kyverno Validation Policies – Deny Rules

- Beispiel-Snippet eines deny-Blocks

```
validate:
  deny:
    conditions:
      - key: "{{ request.namespace }}"
        operator: NotEquals
        value: "cattle-neuvector-system"
      - key: "{{ request.object.spec.http[*].match[0].uri.exact }}"
        operator: AnyIn
        value: ["/"]
    message: "The root path `/` is not allowed as uri.exact in a VirtualService."
```



# Kyverno Validation Policies – Foreach

- Erlaubt das Iterieren über Listen (Sub-Elemente) in Policies
- Vereinfacht Prüfungen auf einzelne Elemente, z. B. Container in einem Pod
- Nested foreach möglich für tiefer verschachtelte Arrays (z. B. `.spec.groups[].rules[]`)
- Falls Liste fehlt → kein Fehler, sondern wird übersprungen

# Kyverno Validation Policies – Foreach



- Beispiel-Snippet Foreach

```
validate:
  message: "Image registry must be set `to europe-west3-docker.pkg.dev`."
  foreach:
    - list: "request.object.spec.[containers, initContainers, ephemeralContainers]"
      preconditions:
        all:
          - key: "{{ element.name | | \" }}"
            operator: NotEquals
            value: istio-proxy
          - key: "{{ element.name | | \" }}"
            operator: NotEquals
            value: istio-validation
      pattern:
        image: "europe-west3-docker.pkg.dev/*"
```

# Kyverno Validation Policies – Variablen



- Machen Policies dynamisch und wiederverwendbar, z. B. in Templates, Patchstrategien oder Validierungen

```
rules:  
- name: check-hostpaths  
  match:  
    any:  
      - resources:  
          kinds:  
            - Pod  
  context:  
  - name: hostpathvolnames  
    variable:  
      jmesPath: request.object.spec.volumes[?hostPath].name  
  preconditions:  
    all:  
      - key: "{{ length(hostpathvolnames) }}"  
        operator: GreaterThan  
        value: 0
```





# Kyverno Validation Policies – Preconditions

- Steuern wann eine Regel ausgeführt wird
- Werden nach match/exclude, aber vor dem eigentlichen Regelkörper geprüft
- Da Preconditions oft Felder überprüfen, die optional sind, kann es notwendig sein, eine JMESPath-Syntax für Nicht-Existenzprüfungen (|| ") zu verwenden
- Können durch Kontextdaten erweitert werden
- context erlaubt das Hinzuladen externer Daten (z. B. über API-Calls oder ConfigMaps), um damit in Preconditions zu vergleichen

```
context:  
- name: namespacefilters  
  configMap:  
    name: namespace-filters  
    namespace: kyverno
```

# Kyverno Validation Policies – Auto-Gen Rules



- Pods sind zentrale Objekte in Kubernetes, werden aber selten direkt erstellt, stattdessen verwalten höhere Controller wie Deployments oder StatefulSets die Pods
- Kyverno vereinfacht Richtlinienmanagement, indem es automatisch Regeln für diese Controller aus einer einzigen Pod-Regel generiert
  - allerdings nur, wenn die Regel ausschließlich auf Pods zielt
- Wird durch Annotation gesteuert
  - `pod-policies.kyverno.io/autogen-controllers: Deployment, Job`
  - `pod-policies.kyverno.io/autogen-controllers: none`
- Kyverno generiert Regeln nur automatisch, wenn in *match* und *exclude* ausschließlich „Pod“ als Ressourcentyp angegeben ist
- *mutate*-Regeln mit JSON-Patches sind von der automatischen Generierung ausgeschlossen

# Kyverno Validation Policies – CEL



- Kyverno unterstützt CEL, die gleiche Sprache, die auch in Kubernetes ValidatingAdmissionPolicies genutzt wird
- CEL-Ausdrücke sind in der Regel kurze „Einzeiler“, die sich gut in die String-Felder von Kubernetes-API-Ressourcen einfügen
- CEL wird in `validate.cel.expressions` verwendet
  - jede Expression, die auf `false` ausgewertet, führt je nach `failureAction` zu einer Blockade oder einem Audit-Log

```
validate:  
  failureAction: Enforce  
  cel:  
    expressions:  
      - expression: "object.spec.replicas < 4"  
        message: "Deployment spec.replicas must be less than 4."
```

# Kyverno Validation Policies – CEL-Variablen



Variable	Beschreibung	Beispielausdruck	Kommentar
object	Neues Objekt aus der Admission-Anfrage	object.metadata.name.starts With("app-")	null bei DELETE-Anfragen
oldObject	Das existierende Objekt	oldObject.spec.replicas != object.spec.replicas	null bei CREATE-Anfragen
request	Attribute aus Admission Request	request.operation == "CREATE"	
params	Parameterressource, auf die sich cel.paramKind und cel.paramRef beziehen	params. <b>allowedUsers</b> .contains(request.userInfo.username)	Macht Policies dynamisch konfigurierbar
namespaceObject	Namespace-Ressource des eingehenden Objekts	namespaceObject.metadata.labels["env"] == "prod"	null bei Clusterscoped-Ressourcen
authorizer	Berechtigungsüberprüfungen	authorizer.requestResource("create") == true	Powerful: prüft Berechtigungen innerhalb der CEL
authorizer.requestResource	Kurzform für authorizer.resource(...) mit aktuellem Kontext	authorizer.requestResource("delete") == false	Nutzt GVK, Namespace und Name der Anfrage



# Hands-On (1/3)



- Voraussetzung: Eigenes Kubernetes-Cluster, Githubaccount, kubectl, helm
  - Falls nicht vorhanden, bitte nachholen

- Installation von Kyverno:

```
helm repo add kyverno https://kyverno.github.io/kyverno/
```

```
helm repo update
```

```
helm install kyverno kyverno/kyverno -n kyverno --create-namespace --version 3.3.9 --set features.dumpPayload.enabled=true
```

- Installation Kyverno-CLI: <https://release-1-13-0.kyverno.io/docs/kyverno-cli/install/>
- Ziel: Nach diesem Kapitel haben alle Teilnehmenden 4 Kyverno Policies in ihrem Cluster deployed. Dies ist wichtig für die folgenden Kapitel, da wir diese noch testen und monitoren werden.
- Git Repository für den Workshop: <https://github.com/kubespectra/kyverno-workshop>

# Hands-On (2/3)



Implementiere folgende Kyverno Policies:

- Policy 1 – validate-emptydir-sizelimit
  - Volumes vom Typ emptyDir müssen ein sizeLimit haben
  - Folgende Volumes sollen ausgenommen werden: istio-envoy, istio-data, workload-certs, credential-socket, workload-socket
  - Füge eine aussagekräftige Fehlermeldung ein, falls deine Policy anschlägt. Diese soll u.A. den Namen des entsprechenden Volumes ausgeben
- Policy 2 – validate-allowed-namespace
  - Unter folgendem Link (<https://github.com/kubespectra/kyverno-workshop/blob/main/configmap.yaml>) liegt eine configmap, in der Namespaces ProjectIds zugeordnet sind
  - Bei Erstellung neuer Namespaces soll sichergestellt werden, dass diese die korrekte ProjectId als Label enthalten wie in dem Beispiel

```
metadata:  
  name: team-x  
  labels:  
    projectId: pid12345
```

# Hands-On (3/3)



Implementiere folgende Policies

- Policy 3: disallow-root-user-except-admins
  - Standardmäßig soll kein Pod als RootUser mit runAsUser: 0 (root) laufen
  - Schließe den Namespace *kube-system* von dieser Regel aus
  - Verwende CEL in deinem Validate-Statement
- Policy 4: Implementiere eine validierende Kyverno Policy, die du für sinnvoll hältst

# Testen

# Testen – Manuell mit CLI



```
kyverno test /path/to/folderContainingTestYamls
```

- Testet einen bestimmten Satz von Ressourcen gegen eine oder mehrere Policies, um die gewünschten Ergebnisse, die im Voraus in einer separaten Testmanifestdatei angegeben wurden, mit den tatsächlichen Ergebnissen zu vergleichen
- Testet auch Auto-gen Policies
- *kyverno test* sucht nach einer Datei *kyverno-test.yaml* und führt die darin enthaltenen Tests aus
- In den Tests kann zwischen 4 Optionen entschieden werden (pass, skip, fail, warn). Wenn das Ergebnis des Tests die gewünschte Option ausgibt, gilt dieser als bestanden

# Testen – Aufbau Test



- Snippet kyverno-test.yaml

```
apiVersion: cli.kyverno.io/v1alpha1
kind: Test
metadata:
  name: validate-nonroot
policies:
  - ../../policies/policy-3_cel.yaml
resources:
  - pod-success.yaml
  - pod-fail.yaml
results:
  - policy: restrict-root-user
    rule: validate-non-root
    resources:
      - test/busybox-pod
kind: Pod
result: pass
```



# Testen – Automatisierung mittels CI



- Zwei wesentliche Anwendungsfälle
  1. Entwicklungsteams sind verantwortlich für ihre Manifeste und möchten bei einem Pull-Request überprüfen, ob diese regelkonform sind, bevor ein Deployment stattfindet
    - Pipeline ist nur erfolgreich, wenn alle Tests bestanden sind
    - Tests, die hier nicht bestanden werden, werden auch im Kubernetes Cluster anschlagen
  2. Testen der Regeln
    - Verhalten der Policies überprüfen
    - Erwartetes Verhalten nach Änderungen sicherstellen
    - Neue Kyverno Versionen testen
- Umsetzung durch Nutzung einer der Github Action: [kyverno/action-install-cli@v0.2.0](#)
- Das Git Repository muss die Policies, die zu testenden Ressourcen und die entsprechenden Tests enthalten

# Testen – Hands-On



- Forke das Git-Repository in deinen Account:
  - <https://github.com/kubespectra/kyverno-workshop>
  - Erklärung wie man forkt: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo>
- Kopiere deine Policies in den Ordner /policies
  - Passe die Tests auf deine Namen an
  - Führe den Test manuell mit dem Kyverno CLI aus: *kyverno test* .
  - Ergänze fehlende Testfälle (vor allem für Policy 4)
  - Push alle Policies in Git und überprüfe die Ergebnisse der Github-Action
- Erstelle Ressourcen auf deinem Cluster mit folgenden Bedingungen
  1. Soll die Validierungen nicht bestehen
  2. Soll die Validierungen bestehen
- Command, um ein Deployment Manifest zu generieren:  
`kubectl create deployment nginx --image=nginx:latest -n <namespace> -oyaml --dry-run='client'`

# Reports



# Reports - Beispiel Validation Policy



```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
  metadata:
name: validate-image-tag
spec:
  validationFailureAction: Audit
  rules:
    - name: deny-latest-tag
      match:
        any:
          - resources:
              kinds:
                - Pod
      validate:
        message: "Using the mutable image tag 'latest' is not allowed."
        pattern:
          spec:
            =(containers):
              - image: "!*:latest"
```

# Reports - Beispiel Validation Policy



```
$ kubectl create deployment busybox --image=busybox:latest -n psa-test
$ kubectl get policyreport -n psa-test
```

NAME		KIND		NAME		PASS	FAIL	WARN	ERROR
SKIP	AGE								
43c9a204-37a4-4960-ba6b-c974ee29fb6d		Pod		deployment-check-example-7f854fdc76-					
zrdxz	0	1	0	0	0				99s

```
apiVersion: wgpolicyk8s.io/v1alpha2
kind: PolicyReport
results:
- message: 'validation error: Using the mutable image tag "latest" is not allowed.
  rule deny-latest-tag failed at path /spec/containers/0/image/'
  policy: validate-image-tag
  result: fail
  rule: deny-latest-tag
  scored: true
  source: kyverno
scope:
  apiVersion: v1
  kind: Pod
  name: deployment-check-example-7f854fdc76-zrdxz
  namespace: psa-test
summary:
  error: 0
  fail: 1
  pass: 0
  skip: 0
  warn: 0
```



# Reports – Background Scans

- Die Anwendung von Policies auf vorhandene Ressourcen wird als Background Scan bezeichnet und ist standardmäßig aktiviert
- Nützlich, um potenzielle Auswirkungen neuer Richtlinien zu bewerten
  - Es werden keine vorhandenen Ressourcen blockiert, auch nicht im Enforce-Modus
- Der Background Scan erfolgt in regelmäßigen Abständen (standardmäßig stündlich).
- Bei aktivem Background Scan werden die Ergebnisse in einem Report aufgezeichnet, unabhängig davon, ob der Modus Enforce oder Audit ist

```
spec:  
  background: true  
  rules:  
  - name: deny-image-tag
```



# Reports – Policy Reporter



- Policy Reports
  - über alle Namespaces verteilt
  - Unübersichtlichkeit in Reports, da viele Ergebnisse in einem Bericht stehen
  - Aufwändig fehlgeschlagene Regeln zu finden
- Policy Reporter
  - Wurde entwickelt, um die Ergebnisse der validierenden Kyverno Policies besser sichtbar und beobachtbar zu machen
  - Der optionale Metrik-Endpunkt kann verwendet werden, um Verstöße in Monitoring-Tools wie Grafana zu beobachten
  - Bietet auch ein eigenständiges Dashboard, um einen grafischen Überblick über alle Ergebnisse und Informationen über Kyverno-Richtlinien zu erhalten

# Reports – Hands-On



- Installation von: Policy Reporter core application, Policy Reporter Kyverno Plugin, and the Policy Reporter UI with the Kyverno views enabled

```
helm repo add policy-reporter https://kyverno.github.io/policy-reporter
helm repo update
helm upgrade policy-reporter policy-reporter/policy-reporter -f reporter-values.yaml --create-namespace -n policy-reporter --version=3.0.5
kubectl port-forward service/policy-reporter-ui 8081:8080 -n policy-reporter
```

- *reporter-values.yaml* - <https://github.com/kubespectra/kyverno-workshop/blob/main/reporter-values.yaml>
- Analysiere die Ergebnisse deiner Kyverno Policies
- Erstelle ein personalisiertes Dashboard
- <https://artifacthub.io/packages/helm/policy-reporter/policy-reporter/3.0.5>

# Abschlussrunde



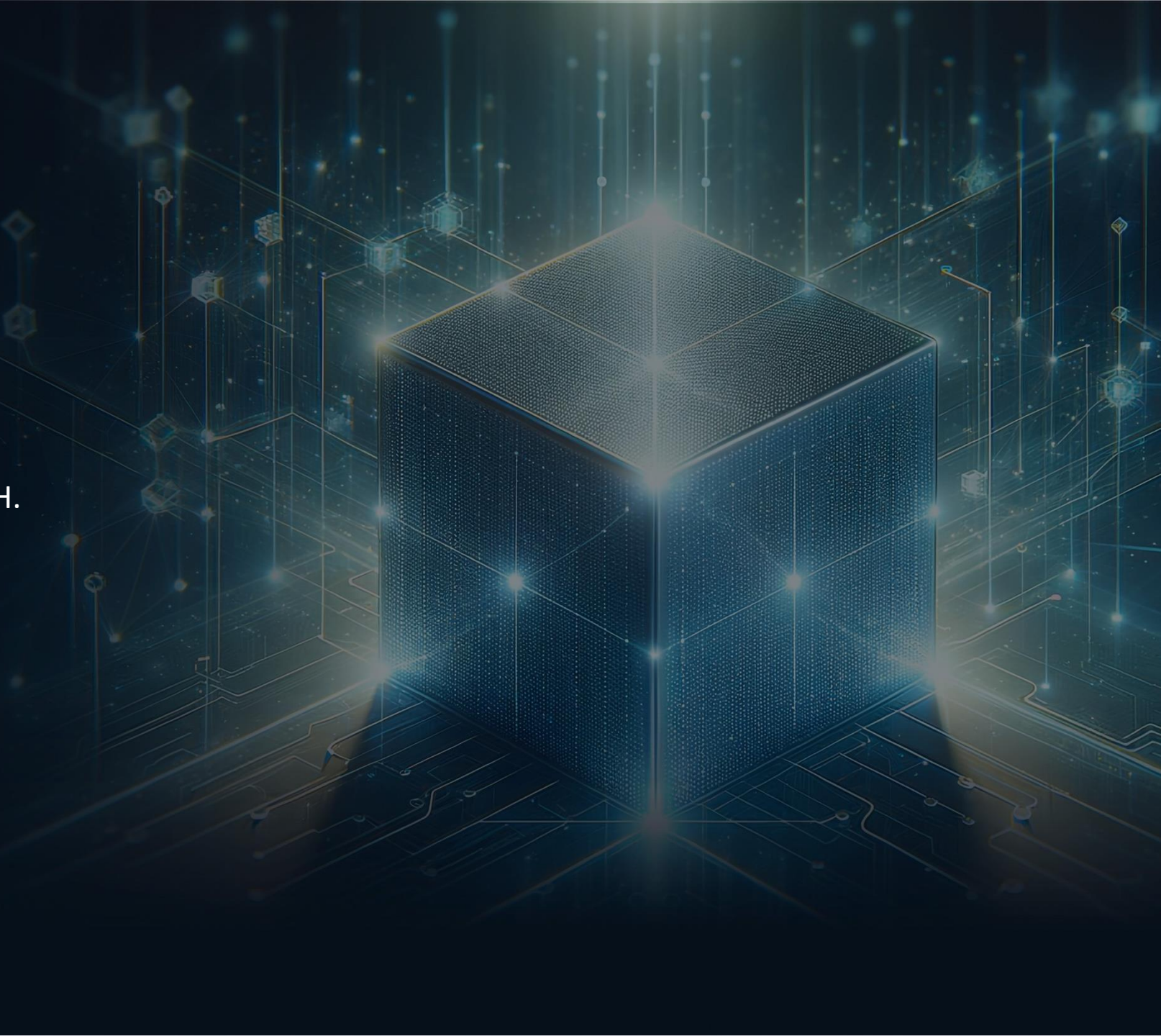
- Gibt es noch Fragen?
- Vortragstipp: Unlocking Cloud & DevOps in Regulated Industries

Donnerstag 18 Uhr, Schatzinsel – Nicole Philippczyck

- Umsetzung von Compliance Vorgaben in regulierten Branchen wie im Banken- und Versicherungswesen

LinkedIn





KubeSpectra is a brand of infologistix GmbH.  
All rights reserved.

infologistix GmbH  
Gutenbergstraße 7  
85748 Garching  
Germany

[www.kubespectra.io](http://www.kubespectra.io) / [www.infologistix.de](http://www.infologistix.de)  
[kubespectra@infologistix.de](mailto:kubespectra@infologistix.de)  
T +49 89 81885979