**What is Kyverno?**

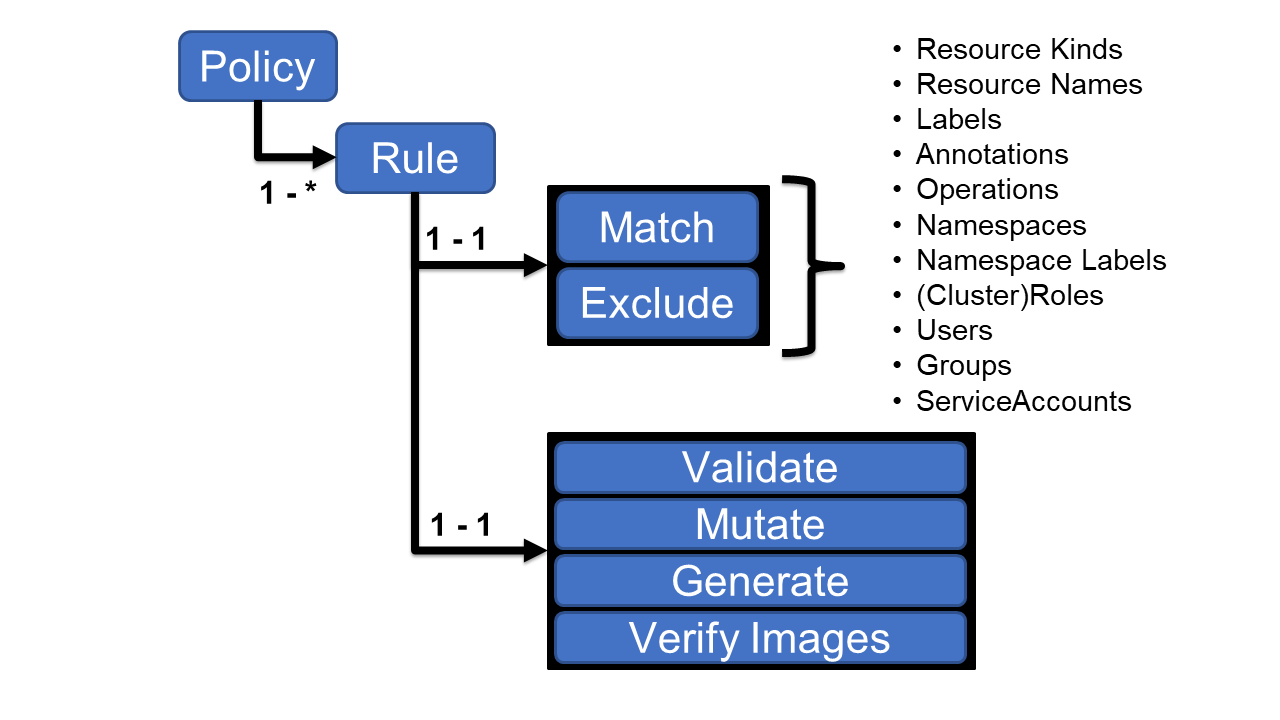
Kyverno (Greek for “govern”) is a policy engine explicitly designed for Kubernetes.

**Kyverno Policies in Kubernetes**

Kyverno is a policy engine designed specifically for Kubernetes, allowing cluster administrators to define, enforce, and validate policies for Kubernetes resources. It provides a way to ensure compliance, enforce best practices, and maintain security and governance in a Kubernetes environment. Kyverno is declarative and operates seamlessly with Kubernetes YAML manifests, making it developer-friendly.

**Kyverno can be used to:**

* Enhance cluster security
* Set default policies and rules
* Audit or enforce set rules
* Validate resources
* Mutate resources before applying
* Generate policies
* Manage pod security



**How Kyverno Works**

**Policy Types**

Kyverno policies are written as Kubernetes Custom Resource Definitions (CRDs) in YAML. They can perform the following actions:

* **Mutate**: Modify resource configurations (e.g., add labels, set defaults).
* **Validate**: Enforce rules to ensure resources meet certain criteria (e.g., disallow latest tag for container images).
* **Generate**: Automatically create or manage additional resources (e.g., generate a namespace-wide ConfigMap or NetworkPolicy).

**Workflow**

* **Admission Controller:** Kyverno operates as a Kubernetes Admission Controller. When a resource is created, updated, or deleted, Kyverno intercepts the request and applies the relevant policies.
* **Policy Validation:** Checks if the resource adheres to the defined policies.
* **Mutation/Generation:** Applies required changes or creates additional resources.
* **Policy Violation Reporting:** If a policy is violated, Kyverno logs it and can optionally block the operation based on configuration.

**Policy Structure**

A Kyverno policy typically includes:

**Metadata:** Name, namespace, and labels.

**Spec:**

* **rules:** Defines the validation, mutation, or generation logic.
* **match and exclude:** Specifies the resources to which the policy applies.
* **validationFailureAction**: Decides whether the policy violation should be enforced (enforce) or just reported (audit).

**Example**

apiVersion: kyverno.io/v1  
kind: ClusterPolicy  
metadata:  
 name: disallow-latest-tag  
**spec**:  
 rules:  
 - name: block-latest-tag  
 match:  
 resources:  
 kinds:  
 - Pod  
 validate:  
 message: "The 'latest' tag is not allowed. Use an immutable tag instead."  
 pattern:  
 **spec**:  
 containers:  
 - image: "!~ latest"

This policy prevents the use of the latest tag in container images.

**Pod YAML File** (Compliant Example):

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6

**Non-Compliant Example**:

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:latest

**Command to Deploy Policy**:

kubectl apply -f disallow-latest-tag.yaml

**Command to Deploy Pod**:

kubectl apply -f nginx-latest-complaint.yaml  
kubectl apply -f nginx-latest-non-complaint.yaml

**Key Benefits of Kyverno**

* **Kubernetes Native:** Policies are defined as YAML, no additional programming required.
* **Flexible and Extensible:** Supports complex rules and integrates easily with CI/CD pipelines.
* **Audit and Enforce:** Can audit existing resources or enforce policies on new changes.
* **Open Source:** Community-driven and actively maintained.

Kyverno makes it easier to enforce organizational policies and governance in Kubernetes clusters without introducing additional complexity.

# Set Up the Kubernetes Cluster

## Create a Kubernetes Cluster in Google Cloud (GKE)

* Open Google Cloud Console or use gcloud CLI.
* Use the following command to create a GKE cluster:

gcloud container clusters create kyverno-cluster \

--zone us-central1-c \

--num-nodes 3 \

--machine-type e2-medium

* Configure kubectl to use the cluster:

gcloud container clusters get-credentials kyverno-cluster --zone us-central1-c

## Verify the cluster

* Check cluster nodes:

kubectl get nodes

# Deploy Kyverno

Kyverno can be deployed using its Helm chart or by applying its manifest files.

## **Option 1: Using Helm**

1. **Install Helm**
2. **Add the Kyverno Helm Repository:**

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

helm repo update

1. **Install Kyverno Helm Chart:**

helm install kyverno kyverno/kyverno --namespace kyverno --create-namespace

1. **Verify Installation:**
   * Check if Kyverno pods are running:

kubectl get pods -n kyverno

* + You should see several Kyverno-related pods in the Running state.

## Policy Descriptions with Pod YAML Code Snippets

### **1. ENFORCE CPU AND MEMORY LIMITS**

**Description**: Ensures that all containers specify CPU and memory limits.

Ensures that resource limits for CPU and memory are specified for all containers, providing predictability and fairness in resource allocation across the cluster.

**Policy YAML File**:

apiVersion: kyverno.io/v1  
kind: ClusterPolicy  
metadata:  
 name: enforce-cpu-memory-limits  
spec:  
 rules:  
 - name: check-cpu-memory-limits  
 match:  
 resources:  
 kinds:  
 - Pod  
 validate:  
 message: "CPU and memory limits must be set."  
 pattern:  
 spec:  
 containers:  
 - resources:  
 limits:  
 cpu: "?\*"  
 memory: "?\*"

**Pod YAML File** (Compliant Example):

apiVersion: v1  
kind: Pod  
metadata:  
 name: resource-limits-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6  
 resources:  
 limits:  
 cpu: "500m"  
 memory: "256Mi"

**Non-Compliant Example**:

apiVersion: v1  
kind: Pod  
metadata:  
 name: no-resource-limits-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6

**Command to Deploy Policy**:

kubectl apply -f enforce-cpu-memory-limits.yaml

**Command to Deploy Pod**:

kubectl apply -f enforce-cpu-memory-limits-compliant-pod.yaml

kubectl apply -f enforce-cpu-memory-limits-non-compliant-pod.yaml

### **2. REQUIRE IMAGE PULL POLICY**

**Description**: Ensures that the imagePullPolicy is explicitly set for all containers.

Mandates that the imagePullPolicy attribute is explicitly defined in all container specifications, ensuring predictable behavior during container image pulls.

**Policy YAML File**:

apiVersion: kyverno.io/v1  
kind: ClusterPolicy  
metadata:  
 name: require-image-pull-policy  
spec:  
 rules:  
 - name: check-image-pull-policy  
 match:  
 resources:  
 kinds:  
 - Pod  
 validate:  
 message: "Image pull policy must be set explicitly."  
 pattern:  
 spec:  
 containers:  
 - imagePullPolicy: "Always"

**Pod YAML File** (Compliant Example):

apiVersion: v1  
kind: Pod  
metadata:  
 name: pull-policy-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6  
 imagePullPolicy: Always

**Non-Compliant Example**:

apiVersion: v1  
kind: Pod  
metadata:  
 name: missing-pull-policy-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6

**Command to Deploy Policy**:

kubectl apply -f require-image-pull-policy.yaml

**Command to Deploy Pod**:

kubectl apply -f require-image-pull-policy-compliant-pod.yaml

kubectl apply -f require-image-pull-policy-non-compliant-pod.yaml

### **2 FORCE READ-ONLY ROOT FILESYSTEM**

**Description**: Requires containers to have a readOnlyRootFilesystem security context.

Ensures that all container filesystems are mounted as read-only, reducing the risk of unauthorized or accidental modifications.

**Policy YAML File**:

apiVersion: kyverno.io/v1  
kind: ClusterPolicy  
metadata:  
 name: enforce-readonly-rootfs  
spec:  
 rules:  
 - name: readonly-rootfs  
 match:  
 resources:  
 kinds:  
 - Pod  
 validate:  
 message: "Containers must have 'readOnlyRootFilesystem' set to true."  
 pattern:  
 spec:  
 containers:  
 - securityContext:  
 readOnlyRootFilesystem: true

**Pod YAML File** (Compliant Example):

apiVersion: v1  
kind: Pod  
metadata:  
 name: readonly-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6  
 securityContext:  
 readOnlyRootFilesystem: true

**Non-Compliant Example**:

apiVersion: v1  
kind: Pod  
metadata:  
 name: writable-pod  
spec:  
 containers:  
 - name: nginx  
 image: nginx:1.21.6  
 securityContext:  
 readOnlyRootFilesystem: false

**Command to Deploy Policy**:

kubectl apply -f enforce-readonly-rootfs.yaml

**Command to Deploy Pod**:

kubectl apply -f readonly-pod.yaml  
kubectl apply -f writable-pod.yaml