1 KCA

Generated on: 2026-01-13 15:04:58 **Version:** 1.0

1.1 Table of Contents

1.2 Overview

The **Kyverno Certified Associate (KCA)** exam demonstrates knowledge of Kubernetes policy management using Kyverno.

1.3 Exam Overview

Detail	Information
Exam Format	Multiple Choice
Number of Questions	60
Duration	90 minutes
Passing Score	75%
Certification Validity	3 years
Cost	$250 USD
Retake Policy	1 free retake

1.4 Exam Domains & Weights

Domain	Weight
Fundamentals	15%
Policy Authoring	30%
Policy Application	20%
Policy Operations	15%
Advanced Policy Concepts	20%

1.5 Key Topics

1.5.1 Policy Types

- Validation policies
- Mutation policies
- Generation policies
- Image verification

1.5.2 Policy Features

- Match and exclude
- Preconditions
- Variables and context
- Background scanning

1.6 Study Resources

- Kyverno Documentation
- KCA Curriculum
- Kyverno Playground

1.7 Navigation

- Next: Sample Questions →

1.8 Kyverno Fundamentals

Comprehensive guide to Kyverno policy engine for KCA certification.

1.9 Overview

Kyverno is a Kubernetes-native policy engine that can:

- **Validate** - Ensure resources meet requirements
- **Mutate** - Modify resources automatically
- **Generate** - Create additional resources
- **Verify Images** - Check image signatures

1.10 Installation

```
# Using kubectl
kubectl create -f https://github.com/kyverno/kyverno/releases/
        download/v1.10.0/install.yaml

# Using Helm
helm repo add kyverno https://kyverno.github.io/kyverno/
helm install kyverno kyverno/kyverno -n kyverno --create-namespace

# Verify
kubectl get pods -n kyverno
```

1.11 Policy Structure

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: policy-name
spec:
  validationFailureAction: Enforce  # or Audit
  background: true
  rules:
  - name: rule-name
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Error message"
      pattern:
        spec:
          containers:
          - name: "*"
            resources:
              limits:
                memory: "?*"
```

1.12 Validation Policies

1.12.1 Require Labels

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-labels
```

```
spec:
  validationFailureAction: Enforce
  rules:
  - name: check-team-label
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Label 'team' is required"
      pattern:
        metadata:
          labels:
            team: "?*"
```

1.12.2 Require Resource Limits

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-limits
spec:
  validationFailureAction: Enforce
  rules:
  - name: validate-resources
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "CPU and memory limits are required"
      pattern:
        spec:
          containers:
          - resources:
              limits:
                memory: "?*"
                cpu: "?*"
```

1.12.3 Disallow Privileged Containers

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged
spec:
  validationFailureAction: Enforce
  rules:
  - name: deny-privileged
```

```
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Privileged containers are not allowed"
      pattern:
        spec:
          containers:
          - securityContext:
              privileged: "!true"
```

1.13 Mutation Policies

1.13.1 Add Default Labels

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-labels
spec:
  rules:
  - name: add-default-labels
    match:
      any:
      - resources:
          kinds:
          - Pod
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            app.kubernetes.io/managed-by: kyverno
```

1.13.2 Add Resource Defaults

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-default-resources
spec:
  rules:
  - name: add-default-requests
    match:
      any:
      - resources:
          kinds:
          - Pod
```

```
    mutate:
      patchStrategicMerge:
        spec:
          containers:
          - (name): "*"
            resources:
              requests:
                memory: "64Mi"
                cpu: "100m"
```

1.13.3 Add Sidecar

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-sidecar
spec:
  rules:
  - name: inject-sidecar
    match:
      any:
      - resources:
          kinds:
          - Deployment
          selector:
            matchLabels:
              inject-sidecar: "true"
    mutate:
      patchStrategicMerge:
        spec:
          template:
            spec:
              containers:
              - name: sidecar
                image: busybox
                command: ['sh', '-c', 'sleep infinity']
```

1.14 Generate Policies

1.14.1 Generate NetworkPolicy

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-netpol
spec:
  rules:
  - name: generate-default-deny
    match:
```

```
      any:
      - resources:
          kinds:
          - Namespace
    generate:
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      name: default-deny
      namespace: "{{request.object.metadata.name}}"
      data:
        spec:
          podSelector: {}
          policyTypes:
          - Ingress
          - Egress
```

1.14.2 Generate ResourceQuota

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-quota
spec:
  rules:
  - name: generate-resourcequota
    match:
      any:
      - resources:
          kinds:
          - Namespace
    generate:
      apiVersion: v1
      kind: ResourceQuota
      name: default-quota
      namespace: "{{request.object.metadata.name}}"
      data:
        spec:
          hard:
            pods: "10"
            requests.cpu: "4"
            requests.memory: "8Gi"
```

1.15 Image Verification

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-images
spec:
  validationFailureAction: Enforce
```

```yaml
  rules:
  - name: verify-signature
    match:
      any:
      - resources:
          kinds:
          - Pod
    verifyImages:
    - imageReferences:
      - "myregistry.io/*"
      attestors:
      - entries:
        - keys:
            publicKeys: |-
              -----BEGIN PUBLIC KEY-----
              ...
              -----END PUBLIC KEY-----
```

1.16 Policy Reports

```bash
# View policy reports
kubectl get policyreport -A
kubectl get clusterpolicyreport

# Describe report
kubectl describe policyreport -n default
```

1.17 Useful Commands

```bash
# List policies
kubectl get clusterpolicy
kubectl get policy -A

# Test policy
kubectl apply -f policy.yaml --dry-run=server

# View policy details
kubectl describe clusterpolicy require-labels

# Check admission controller
kubectl get validatingwebhookconfiguration
kubectl get mutatingwebhookconfiguration
```

← Back to KCA Overview

1.18 Sample Practice Questions

1.19 Practice Resources

- Kyverno Documentation
- Kyverno Playground

1.20 Fundamentals (15%)

1.20.1 Question 1

What are the main policy types in Kyverno?

Show Solution

1. **Validate** - Check resources against rules, block non-compliant
2. **Mutate** - Modify resources automatically
3. **Generate** - Create additional resources
4. **VerifyImages** - Verify container image signatures

1.20.2 Question 2

What is the difference between ClusterPolicy and Policy?

Show Solution

- **ClusterPolicy** - Cluster-scoped, applies to all namespaces
- **Policy** - Namespace-scoped, applies only to its namespace

```yaml
# ClusterPolicy - cluster-wide
apiVersion: kyverno.io/v1
kind: ClusterPolicy

# Policy - namespace-scoped
apiVersion: kyverno.io/v1
kind: Policy
metadata:
  namespace: my-namespace
```

1.21 Policy Authoring (30%)

1.21.1 Question 3

Create a validation policy that requires all pods to have resource limits.

Show Solution

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-limits
spec:
  validationFailureAction: Enforce
  rules:
  - name: check-limits
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Resource limits are required"
      pattern:
        spec:
          containers:
          - resources:
              limits:
                memory: "?*"
                cpu: "?*"
```

1.21.2 Question 4

Create a mutation policy that adds a default label to all pods.

Show Solution

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-default-label
spec:
  rules:
  - name: add-team-label
    match:
      any:
      - resources:
          kinds:
          - Pod
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            team: default
```

1.21.3 Question 5

Create a policy that generates a NetworkPolicy for each new namespace.

Show Solution

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-netpol
spec:
  rules:
  - name: generate-default-deny
    match:
      any:
      - resources:
          kinds:
          - Namespace
    generate:
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      name: default-deny
      namespace: "{{request.object.metadata.name}}"
      data:
        spec:
          podSelector: {}
          policyTypes:
          - Ingress
          - Egress
```

1.21.4 Question 6

Create a policy to verify image signatures using cosign.

Show Solution

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-images
spec:
  validationFailureAction: Enforce
  rules:
  - name: verify-signature
    match:
      any:
      - resources:
          kinds:
          - Pod
    verifyImages:
    - imageReferences:
      - "ghcr.io/myorg/*"
      attestors:
      - entries:
        - keys:
            publicKeys: |
              -----BEGIN PUBLIC KEY-----
```

```
...
-----END PUBLIC KEY-----
```

1.22 Policy Application (20%)

1.22.1 Question 7

What is the difference between Enforce and Audit modes?

Show Solution

- **Enforce** - Block non-compliant resources from being created
- **Audit** - Allow resources but report violations in policy reports

```yaml
spec:
  validationFailureAction: Enforce  # Block violations
  # or
  validationFailureAction: Audit    # Report only
```

Use Audit mode for testing policies before enforcement.

1.22.2 Question 8

How do you exclude certain resources from a policy?

Show Solution

```yaml
spec:
  rules:
  - name: my-rule
    match:
      any:
      - resources:
          kinds:
          - Pod
    exclude:
      any:
      - resources:
          namespaces:
          - kube-system
      - resources:
          selector:
            matchLabels:
              skip-policy: "true"
```

1.22.3 Question 9

How do you use preconditions in a policy?

Show Solution

```yaml
spec:
  rules:
  - name: check-image-tag
    match:
      any:
      - resources:
          kinds:
          - Pod
    preconditions:
      all:
      - key: "{{request.operation}}"
        operator: NotEquals
        value: DELETE
      - key: "{{request.object.metadata.labels.environment}}"
        operator: Equals
        value: production
    validate:
      message: "Production pods must use specific tags"
      pattern:
        spec:
          containers:
          - image: "*/myapp:v*"
```

1.23 Policy Operations (15%)

1.23.1 Question 10

How do you view policy reports?

Show Solution

```bash
# View cluster-wide policy reports
kubectl get clusterpolicyreport

# View namespace policy reports
kubectl get policyreport -n my-namespace

# Get detailed report
kubectl get policyreport -n my-namespace -o yaml

# Check specific policy violations
kubectl get policyreport -A -o jsonpath='{.items[*].results[?
        (@.result=="fail")]}'
```

1.23.2 Question 11

How do you troubleshoot a policy that isn't working?

Show Solution

```
# Check policy status
kubectl get clusterpolicy my-policy -o yaml

# Check Kyverno logs
kubectl logs -n kyverno -l app.kubernetes.io/name=kyverno

# Test policy with dry-run
kubectl apply -f pod.yaml --dry-run=server

# Use Kyverno CLI to test
kyverno apply policy.yaml --resource pod.yaml
```

1.24 Advanced Concepts (20%)

1.24.1 Question 12

How do you use variables and context in policies?

Show Solution

```yaml
spec:
  rules:
  - name: use-variables
    match:
      any:
      - resources:
          kinds:
          - Pod
    context:
    - name: allowedRegistries
      configMap:
        name: allowed-registries
        namespace: kyverno
    validate:
      message: "Image must be from allowed registry"
      deny:
        conditions:
          any:
          - key: "{{request.object.spec.containers[0].image}}"
            operator: AnyNotIn
            value: "{{allowedRegistries.data.registries}}"
```

1.24.2 Question 13

How do you use JMESPath expressions in Kyverno?

Show Solution

```yaml
spec:
  rules:
```

```
  - name: check-labels
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "Missing required labels"
      deny:
        conditions:
          any:
          - key: "{{ request.object.metadata.labels | keys(@) |
        length(@) }}"
            operator: LessThan
            value: 2
```

Common JMESPath functions: - `length()` - Get array/string length - `keys()` - Get object keys - `contains()` - Check if array contains value - `to_string()` - Convert to string

1.24.3 Question 14

Create a policy that validates pod security based on namespace labels.

Show Solution

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: enforce-pod-security
spec:
  validationFailureAction: Enforce
  rules:
  - name: restricted-namespace
    match:
      any:
      - resources:
          kinds:
          - Pod
          namespaceSelector:
            matchLabels:
              security: restricted
    validate:
      message: "Pods in restricted namespaces must run as non-
        root"
      pattern:
        spec:
          securityContext:
            runAsNonRoot: true
          containers:
          - securityContext:
              allowPrivilegeEscalation: false
```

1.25 Exam Tips

1. **Know policy types** - Validate, Mutate, Generate, VerifyImages
2. **Understand match/exclude** - How to target specific resources
3. **Practice pattern matching** - Wildcards, anchors, operators
4. **Know validation modes** - Enforce vs Audit
5. **Understand policy reports** - How to view and interpret

← Back to KCA Overview
