Alauda Container Platform

Security

Alauda Container Security

Alauda Container Security

Alauda Cluster Authentication

Alauda Cluster Authentication

Security and Compliance

Compliance   **API Refiner**   **About Alau**

Users and Roles

User   **Group**   **Role**

IDP   **User Policy**

Multitenancy(Project)

Introduction   **Guides**

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

Audit

Introduction

Prerequisites

Procedure

Search Results

Telemetry

Install

Prerequisites

Installation Steps

Enable Online Operations

Uninstallation Steps

Certificates

Automated Kubernetes Certific

Installation

How it works

Operation Considerations

cert-manager

Overview

How it works

Identifying cert-manager Managed Certific

Related Resources

OLM Certifi

Certificate N

Certificate Statu

Built-in Alert Ru

Rotate TLS Certs of Platform Access Addresses

Prerequisites

Procedures

Alauda Container Platform

Alauda Container Security

Alauda Container Security is a comprehensive security solution designed for Kubernetes and containerized environments. It provides centralized management, automated vulnerability scanning, policy enforcement, and compliance checks to help organizations secure their container infrastructure across multiple clusters.

Alauda Container Security adopts a distributed, container-based architecture, consisting of Central Services (for management, API, and UI) and Secured Cluster Services (for monitoring, policy enforcement, and data collection). It integrates with CI/CD pipelines, SIEM, logging systems, and supports the built-in Scanner V4 vulnerability scanner.

> **Note**
>
> Because Alauda Container Security releases on a different cadence from Alauda Container Platform, the Alauda Container Security documentation is now available as a separate documentation set at Alauda Container Security ↗ .

Alauda Cluster Authentication

The Alauda Container Platform Cluster Authentication plugin provides an independent authentication integration capability for global cluster failure scenarios.

When the `global` cluster fails, users can still log in through this service to access and operate Kubernetes clusters, maintaining permissions consistent with the state before the global cluster failure.

> **Note**
>
> Because Cluster Authentication releases on a different cadence from Alauda Container Platform, the Cluster Authentication documentation is now available as a separate documentation set at [Cluster Authentication ↗](#).

Alauda Container Platform

Security and Compliance

Compliance

Introduction

HowTo

Install Alauda Container Platfor

Install via console

Upgrade

Compatibility M

ath C

API Refiner

Introduction

Product Introduction

Limitations

Install Alauda Container Platform API Refiner

Install via console

Install via YAML

Uninstallation Procedures

Default Configuration

About Alauda Container Platform Compliance Service

About Alauda Container Platform Compliance Service

Alauda Container Platform

Compliance

Introduction

Introduction

Install Alauda Container Platform Compliance with Kyverno

Install Alauda Container Platform Compliance with Kyverno

Install via console

Install via YAML

Uninstallation Procedures

Upgrade

Upgrade

Compatibility Matrix

Upgrade Path Guidelines

HowTo

Private Registry Access Config

Why Does Kyverno Need Registry Access

Quick Start

Image Signature Verification Po

What is Image Signature Verification?

Quick Start

Common Use Cases

Image Signa

Why Use Secre

Quick Start

Secret Creation

Common Use C

Image Registry Validation Policy

What is Image Registry Validation?

Quick Start

Common Scenarios

Advanced Patterns

Best Practices

Container Escape Prevention Policy

What is Container Escape Prevention?

Quick Start

Core Container Escape Prevention Policie

Advanced Scenarios

Testing and Validation

Best Practices

Security Co

What is Security

Quick Start

Core Security C

Advanced Scen

Testing and Val

Network Security Policy

What is Network Security?

Quick Start

Core Network Security Policies

Advanced Scenarios

Testing and Validation

Volume Sec

What is Volume

Quick Start

Core Volume S

Advanced Scen

Testing and Val

Alauda Container Platform

Introduction

ACP provides compliance functionality based on the open-source Kyverno component, enabling organizations to define and enforce policies across their Kubernetes clusters.

This feature addresses the challenge of maintaining consistent security, governance, and operational standards by allowing users to create custom policies using Kyverno's YAML syntax and automatically validate resources against these policies.

The compliance functionality provides comprehensive violation monitoring and reporting capabilities, offering both resource-level and policy-level views of compliance violations through an intuitive interface, helping teams quickly identify non-compliant resources and take appropriate remediation actions to maintain their desired security posture and regulatory compliance.

> **INFO**
>
> For more information about Kyverno, read the Kyverno Documentation ↗.

Alauda Container Platform

Install Alauda Container Platform Compliance with Kyverno

Alauda Container Platform Compliance with Kyverno is a platform service that integrates Kyverno for managing compliance policies on the Alauda Container Platform.

TOC

Install via console

1. Navigate to **Administrator**

2. In the left navigation bar, click **Marketplace** > **Cluster Plugins**

3. Search for **Alauda Container Platform Compliance with Kyverno** and click to view its details

4. Click **Install** to deploy the plugin

Install via YAML

1. Check available versions

Ensure the plugin has been published by checking for ModulePlugin and ModuleConfig resources, in `global` cluster :

```
# kubectl get moduleplugins kyverno
NAME       AGE
kyverno    4d20h


# kubectl get moduleconfigs -l cpaas.io/module-name=kyverno
NAME               AGE
kyverno-v4.0.4     4d21h
```

This indicates that the ModulePlugin `kyverno` exists in the cluster and version `v4.0.4` is published.

2. Create a ModuleInfo

Create a ModuleInfo resource to install the plugin without any configuration parameters:

```
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  annotations:
    cpaas.io/display-name: kyverno
    cpaas.io/module-name: '{"en": "Alauda Container Platform Compliance f
or Kyverno",
      "zh": "Alauda Container Platform Compliance for Kyverno"}'
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: kyverno
    cpaas.io/module-type: plugin
    cpaas.io/product: Platform-Center
  name: kyverno-global
spec:
  version: v4.2.0
```

Field explanations:

- `name` : Temporary name for the cluster plugin. The platform will rename it after creation based on the content, in the format `<cluster-name>-<hash of content>` , e.g., `global-ee98c9991ea1464aaa8054bdacbab313` .

- `label cpaas.io/cluster-name` : Specifies the cluster where the plugin should be installed.

- `label cpaas.io/module-name` : Plugin name, must match the ModulePlugin resource.

- `label cpaas.io/module-type` : Fixed field, must be `plugin` ; missing this field causes installation failure.

- `.spec.config` : If the corresponding ModuleConfig is empty, this field can be left empty.

- `.spec.version` : Specifies the plugin version to install, must match `.spec.version` in ModuleConfig.

Uninstallation Procedures

1. Follow steps 1-3 from the installation process to locate the plugin

2. Click **Uninstall** to remove the plugin

Upgrade

> **INFO**
>
> This document provides the upgrade path principles and supported version compatibility for Alauda Container Platform Compliance for Kyverno.

TOC

Compatibility Matrix

The table below lists supported versions of the Alauda Container Platform Compliance for Kyverno:

Compliance for Kyverno Version	Supported Alauda Container Platform Version
4.2.x	4.2.x

Upgrade Path Guidelines

Consecutive Upgrade (Recommended)

- **Description**: Upgrade step-by-step through consecutive minor versions.

- **Example**: 4.1.x → 4.2.x

Patch-Level Upgrade

- **Description**: Upgrades between any patch versions within the same minor version are fully compatible and can be performed directly.

- **Example**: 4.2.0 → 4.2.x

Alauda Container Platform

HowTo

Private Registry Access Config

Why Does Kyverno Need Registry Access

Quick Start

Image Signature Verification Po

What is Image Signature Verification?

Quick Start

Common Use Cases

Image Signa

Why Use Secre

Quick Start

Secret Creation

Common Use C

Image Registry Validation Policy

What is Image Registry Validation?

Quick Start

Common Scenarios

Advanced Patterns

Best Practices

Container Escape Prevention Policy

What is Container Escape Prevention?

Quick Start

Core Container Escape Prevention Policie

Advanced Scenarios

Testing and Validation

Best Practices

Security Co

What is Securit

Quick Start

Core Security C

Advanced Scer

Testing and Val

Network Security Policy

What is Network Security?

Quick Start

Core Network Security Policies

Advanced Scenarios

Testing and Validation

Volume Sec

What is Volume

Quick Start

Core Volume S

Advanced Scer

Testing and Val

Alauda Container Platform

Private Registry Access Configuration

This guide demonstrates how to configure Kyverno to access private container registries. When Kyverno needs to verify image signatures or check image details, it requires proper credentials to access private registries - just like a key card is needed to enter a secure building.

TOC

Why Does Kyverno Need Registry Access?

Kyverno needs to access registries when it:

- **Verifies image signatures**: Downloads signature data to check if images are properly signed

- **Checks image metadata**: Reads image labels, annotations, and manifest information

- **Scans for vulnerabilities**: Downloads images for security scanning

- **Validates image contents**: Inspects what's actually inside container images

Think of it like a security guard who needs to check ID - Kyverno needs to "see" the images to verify them.

Quick Start

1. Create Registry Secret

```
# For company's private registry
kubectl create secret docker-registry my-registry-secret \
  --docker-server=registry.company.com \
  --docker-username=<username> \
  --docker-password=<password> \
  --docker-email=<email@company.com> \
  -n kyverno
```

2. Configure Kyverno to Use the Secret (Recommended)

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kyverno
  namespace: kyverno
imagePullSecrets:
- name: my-registry-secret
```

3. Kyverno Deployment Configuration

If more control is needed, the Kyverno deployment can be modified directly:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kyverno
  namespace: kyverno
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kyverno
  template:
    metadata:
      labels:
        app: kyverno
    spec:
      serviceAccountName: kyverno
      imagePullSecrets:
      - name: my-registry-secret
      - name: gcr-secret
      - name: dockerhub-secret
      containers:
      - name: kyverno
        image: ghcr.io/kyverno/kyverno:latest
        env:
        - name: REGISTRY_CREDENTIAL_HELPERS
          value: "ecr-login,gcr,acr-env"  # Enable credential helpers
        # ... other configuration
```

Alauda Container Platform

Image Signature Verification Policy

This guide demonstrates how to configure Kyverno to verify that container images are properly signed before they can run in a Kubernetes cluster. Think of it like checking an ID card - only images with valid "signatures" are allowed in.

TOC

What is Image Signature Verification?

Image signature verification is like having a security guard check IDs at the door. It ensures:

- **Images are authentic**: They come from who they claim to come from

- **Images are untampered**: No one has modified them after signing

- **Only trusted images run**: Unsigned or improperly signed images are blocked

- **Audit trail**: Track which images were verified and when

Quick Start

1. Generate Keys

```
# Create a signing key pair (like creating an ID card system)
cosign generate-key-pair
# This creates: cosign.key (private, keep secret) and cosign.pub (public,
share freely)
```

2. Sign Images

```
# Sign images (like putting an official stamp on it)
cosign sign --key cosign.key registry.company.com/app:v1.0.0
```

3. Create Basic Verification Policy

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-signed-images
spec:
  validationFailureAction: Enforce  # Block unsigned images
  background: false
  rules:
    - name: check-signatures
      match:
        any:
        - resources:
            kinds:
            - Pod
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"  # Check images from company registry
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # Paste the cosign.pub content here
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        mutateDigest: true  # Convert tags to secure digest format
```

4. Test It

```
# Apply the policy
kubectl apply -f signature-policy.yaml

# Try to run an unsigned image (should fail)
kubectl run test --image=nginx:latest

# Try to run a signed image (should work)
kubectl run test --image=registry.company.com/app:v1.0.0
```

Common Use Cases

Scenario 1: Multiple Teams Need to Sign Critical Images

For critical applications, both the development team AND security team might need to sign images:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-dual-signatures
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: critical-app-signatures
      match:
        any:
        - resources:
            kinds:
            - Pod
      verifyImages:
      - imageReferences:
        - "registry.company.com/critical/*"
        attestors:
        # Both teams must sign
        - count: 1  # Security team signature
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # Security team's public key
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        - count: 1  # Development team signature
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # Development team's public key
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917h
mKO6JWV
                GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J
5==
                -----END PUBLIC KEY-----
        mutateDigest: true
```

Scenario 2: Different Rules for Different Environments

Production needs strict verification, development can be more relaxed:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-specific-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # Strict rules for production
    - name: production-must-be-signed
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
      verifyImages:
      - imageReferences:
        - "*"  # All images must be signed
        failureAction: Enforce  # Block if not signed
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # Production signing key
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        mutateDigest: true

    # Relaxed rules for development
    - name: development-warn-unsigned
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
```

```yaml
              - development
              - staging
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"   # Only check company images
          failureAction: Audit   # Audit but allow unsigned images
          attestors:
          - count: 1
            entries:
            - keys:
                publicKeys: |-
                    -----BEGIN PUBLIC KEY-----
                    # Development signing key
                    MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917h
mKO6JWV
                    GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J
5==
                    -----END PUBLIC KEY-----
          mutateDigest: true
```

Scenario 3: Using Certificates Instead of Keys

For enterprise environments, X.509 certificates might be used:

```bash
# Sign with certificate
cosign sign --cert company-cert.pem --cert-chain ca-chain.pem \
   registry.company.com/myapp:v1.0.0
```

Alauda Container Platform

Image Signature Verification Policy with Secrets

This guide demonstrates how to use Kubernetes Secrets to store public keys for Kyverno image signature verification, providing better security and key management compared to embedding keys directly in policies.

TOC

Why Use Secrets for Public Keys?

Using Kubernetes Secrets for storing public keys offers several advantages:

- **Enhanced Security**: Keys are stored securely in the Kubernetes Secret store

- **Easy Key Rotation**: Update keys without modifying policies

- **Access Control**: Use RBAC to control who can access the secrets

Quick Start

1. Generate and Store Keys in Secret

```
# Generate cosign key pair
cosign generate-key-pair

# Create secret from the public key file
kubectl create secret generic cosign-public-key \
  --from-file=cosign.pub=./cosign.pub \
  --namespace=kyverno

# Verify the secret was created
kubectl get secret cosign-public-key -n kyverno
```

2. RBAC Configuration for Keyverno

Create Service Account for Kyverno

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kyverno-secret-reader
  namespace: kyverno
```

Create Role for Secret Access

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: kyverno
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "list", "watch"]
  resourceNames: ["cosign-public-key", "team-keys"]  # Specific secrets only
```

Bind Role to Service Account

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: kyverno
subjects:
- kind: ServiceAccount
  name: kyverno-secret-reader
  namespace: kyverno
roleRef:
  kind: Role
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

3. Create Policy Using Secret Reference

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-with-secret
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: check-signatures
      match:
        any:
        - resources:
            kinds: [Pod]
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"
        attestors:
        - count: 1
          entries:
          - keys:
              secret:
                name: cosign-public-key
                namespace: kyverno
                key: cosign.pub
              rekor:
                url: https://rekor.sigstore.dev
        mutateDigest: true
```

4. Test the Configuration

```
# Sign an image
cosign sign --key cosign.key registry.company.com/app:v1.0.0

# Apply the policy
kubectl apply -f verify-with-secret.yaml

# Test with signed image (should work)
kubectl run test --image=registry.company.com/app:v1.0.0

# Test with unsigned image (should fail)
kubectl run test-fail --image=nginx:latest
```

Secret Creation Methods

Method 1: From File

```
# Create secret from existing cosign public key file
kubectl create secret generic cosign-public-key \
  --from-file=cosign.pub=./cosign.pub \
  --namespace=kyverno
```

Method 2: From Literal String

```
# Create secret with inline public key content
kubectl create secret generic cosign-public-key \
  --from-literal=cosign.pub="-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sbqOPZrfM
5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZA==
-----END PUBLIC KEY-----" \
  --namespace=kyverno
```

Method 3: From YAML Manifest

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: cosign-public-key
  namespace: kyverno
  labels:
    app: kyverno
    component: image-verification
type: Opaque
stringData:
  cosign.pub: |
    -----BEGIN PUBLIC KEY-----
    MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sbqOPZrfM
    5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZA==
    -----END PUBLIC KEY-----
```

```
kubectl apply -f cosign-secret.yaml
```

Common Use Cases

Scenario 1: Single Team with One Secret

Simple setup where one team manages all image signatures:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: single-team-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: verify-team-signatures
      match:
        any:
        - resources:
            kinds: [Pod, Deployment, StatefulSet, DaemonSet]
      exclude:
        any:
        - resources:
            namespaces: [kube-system, kyverno]

      verifyImages:
      - imageReferences:
        - "registry.company.com/*"
        - "gcr.io/myproject/*"

        failureAction: Enforce

        attestors:
        - count: 1
          entries:
          - keys:
              secret:
                name: team-cosign-key
                namespace: kyverno
                key: cosign.pub
              rekor:
                url: https://rekor.sigstore.dev

        mutateDigest: true
        verifyDigest: true
        required: true
```

Scenario 2: Multi-Team with Different Secrets

Different teams have their own signing keys and secrets:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: multi-team-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # Frontend team images
    - name: verify-frontend-images
      match:
        any:
        - resources:
            kinds: [Pod]
            namespaces: [frontend-*]

      verifyImages:
      - imageReferences:
        - "registry.company.com/frontend/*"

        attestors:
        - count: 1
          entries:
          - keys:
              secret:
                name: frontend-team-key
                namespace: kyverno
                key: cosign.pub
              rekor:
                url: https://rekor.sigstore.dev

        mutateDigest: true
        required: true

    # Backend team images
    - name: verify-backend-images
      match:
        any:
        - resources:
            kinds: [Pod]
            namespaces: [backend-*]

      verifyImages:
```
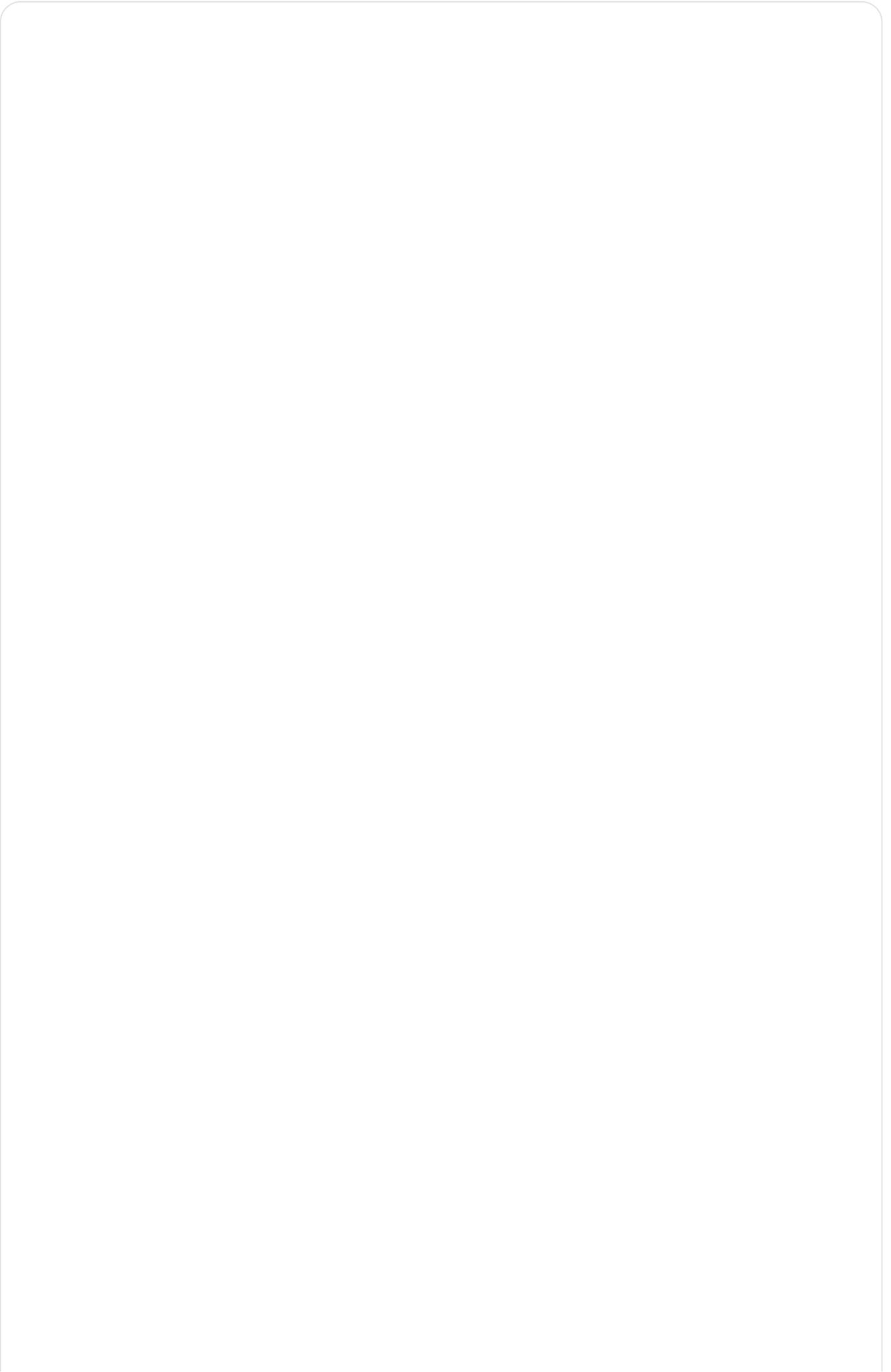
```
    - imageReferences:
      - "registry.company.com/backend/*"

      attestors:
      - count: 1
        entries:
        - keys:
            secret:
              name: backend-team-key
              namespace: kyverno
              key: cosign.pub
            rekor:
              url: https://rekor.sigstore.dev

      mutateDigest: true
      required: true
```

Scenario 3: Critical Images Requiring Multiple Signatures

High-security environments where multiple teams must sign critical images:

```
    - imageReferences:
      - "registry.company.com/backend/*"
```

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: critical-multi-signature
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: verify-critical-images
      match:
        any:
        - resources:
            kinds: [Pod]
            namespaces: [production]

      verifyImages:
      - imageReferences:
        - "registry.company.com/critical/*"

        failureAction: Enforce

        attestors:
        # Security team signature (required)
        - count: 1
          entries:
          - keys:
              secret:
                name: security-team-key
                namespace: kyverno
                key: security.pub
              rekor:
                url: https://rekor.sigstore.dev

        # Development team signature (required)
        - count: 1
          entries:
          - keys:
              secret:
                name: dev-team-key
                namespace: kyverno
                key: development.pub
              rekor:
                url: https://rekor.sigstore.dev
```

```yaml
    # Release team signature (required)
  - count: 1
    entries:
    - keys:
        secret:
          name: release-team-key
          namespace: kyverno
          key: release.pub
        rekor:
          url: https://rekor.sigstore.dev


    mutateDigest: true
    required: true
```

Scenario 4: Offline Environment with Secrets

Using secrets in air-gapped environments:

Alauda Container Platform

Image Registry Validation Policy

This guide demonstrates how to configure Kyverno to control which container registries can be used in a Kubernetes cluster. It implements registry access control policies to ensure only images from approved and trusted registries are deployed.

TOC

What is Image Registry Validation?

Registry validation provides centralized control over image sources. It enables:

- **Control image sources**: Only allow images from trusted registries

- **Block risky registries**: Prevent use of unknown or compromised registries

- **Enforce compliance**: Meet security requirements about image sources

- **Different rules per environment**: Strict rules for production, relaxed for development

- **Track usage**: Monitor which registries are being utilized

Quick Start

1. Block All Except Company Registry

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: company-registry-only
spec:
  validationFailureAction: Enforce  # Block non-approved images
  background: false
  rules:
    - name: check-registry
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "Only company registry allowed: registry.company.com"
        pattern:
          spec:
            containers:
            - image: "registry.company.com/*"
```

2. Test It

```
# Apply the policy
kubectl apply -f registry-policy.yaml

# This should fail (nginx from Docker Hub)
kubectl run test --image=nginx:latest

# This should work (if images exist in the registry)
kubectl run test --image=registry.company.com/nginx:latest
```

Common Scenarios

Scenario 1: Allow Multiple Trusted Registries

Organizations typically use several registries:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: multiple-trusted-registries
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: check-approved-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "Images must come from approved registries: company registry, GCR, or official Docker images"
        anyPattern:
        - spec:
            containers:
            - image: "registry.company.com/*"      # Company registry
        - spec:
            containers:
            - image: "gcr.io/project-name/*"      # Google Container Registry
        - spec:
            containers:
            - image: "docker.io/library/*"      # Official Docker images only
        - spec:
            containers:
            - image: "quay.io/organization/*"      # Red Hat Quay
```

Scenario 2: Different Rules for Different Environments

Production environments should be strict, development can be more flexible:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-based-registry-rules
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # Production: Only certified images
    - name: production-strict-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production only allows certified company images"
        pattern:
          spec:
            containers:
            - image: "registry.company.com/certified/*"

    # Development: More registries allowed
    - name: development-flexible-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - development
            - dev-*
            - staging
            - test-*
      validate:
        message: "Development can use company registry, GCR, or official
Docker images"
        anyPattern:
        - spec:
            containers:
```

```
      - image: "registry.company.com/*"
  - spec:
      containers:
      - image: "gcr.io/dev-project/*"
  - spec:
      containers:
      - image: "docker.io/library/*"
  - spec:
      containers:
      - image: "docker.io/organization/*"
```

Scenario 3: Block Specific Risky Registries

Block specific registries while allowing others:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: block-risky-registries
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # Method 1: Use deny list approach
    - name: block-untrusted-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "Images from untrusted-registry.com are not allowed"
        deny:
          conditions:
          - key: "{{ request.object.spec.containers[?contains(image, 'untrusted-registry.com')] | length(@) }}"
            operator: GreaterThan
            value: 0

    # Method 2: Use allow list for Docker Hub (only official images)
    - name: allow-only-official-dockerhub
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "Only official Docker Hub images are allowed (docker.io/library/*)"
        deny:
          conditions:
          - key: "{{ request.object.spec.containers[?starts_with(image, 'docker.io/') && !starts_with(image, 'docker.io/library/')] | length(@) }}"
            operator: GreaterThan
            value: 0
```

Scenario 4: Team-Specific Registry Access

Different teams can have access to different registries:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: team-specific-registries
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # Frontend team can use Node.js images
    - name: frontend-team-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - frontend-*
      validate:
        message: "Frontend team can use company registry and official Nod
e.js images"
        anyPattern:
        - spec:
            containers:
            - image: "registry.company.com/*"
        - spec:
            containers:
            - image: "docker.io/library/node:*"
        - spec:
            containers:
            - image: "docker.io/library/nginx:*"

    # Data team can use ML/AI registries
    - name: data-team-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - data-*
            - ml-*
      validate:
        message: "Data team can use company registry and ML/AI images"
```

```
        anyPattern:
          - spec:
              containers:
                - image: "registry.company.com/*"
          - spec:
              containers:
                - image: "docker.io/tensorflow/*"
          - spec:
              containers:
                - image: "docker.io/pytorch/*"
          - spec:
              containers:
                - image: "nvcr.io/nvidia/*"
```

Advanced Patterns

Using Wildcards Effectively

```
# Match patterns:
- image: "registry.company.com/*"          # Any image from this registry
- image: "registry.company.com/team-a/*"    # Only team-a images
- image: "*/database:*"                      # Any database image from any registry
- image: "gcr.io/project-*/app:*"            # Any app from project-* in GCR
```

Best Practices

Start with Warnings

```
spec:
  validationFailureAction: Audit   # Start with audit mode, not blocking
```

Exclude System Namespaces

```
rules:
  - name: check-registries
    match:
      any:
        - resources:
            kinds:
              - Pod
    exclude:
      any:
        - resources:
            namespaces:
              - kube-system
              - kyverno
              - kube-public
```

Common Issues

1. **Wrong image format**:

- ❌ `registry.company.com:5000/app` (missing protocol)
- ✅ `registry.company.com/app:latest`

2. **Wildcard confusion**:

- ❌ `registry.company.com*` (missing slash)
- ✅ `registry.company.com/*`

3. **Docker Hub format**:

- ❌ `nginx` (implicit docker.io)
- ✅ `docker.io/library/nginx`

Alauda Container Platform

Container Escape Prevention Policy

This guide demonstrates how to configure Kyverno to prevent container escape attacks by blocking high-risk container configurations that could allow containers to break out of their isolation boundaries.

TOC

Best Practices

    1. Start with Audit Mode

    2. Exclude System Namespaces

What is Container Escape Prevention?

Container escape prevention involves detecting and blocking dangerous container configurations that could allow attackers to escape container isolation and gain access to the host system. This includes:

- **Privileged containers**: Containers running with elevated privileges

- **Host namespace access**: Containers sharing host PID, network, or IPC namespaces

- **Host path mounts**: Containers mounting host filesystem paths

- **Dangerous capabilities**: Containers with excessive Linux capabilities

- **Host port access**: Containers binding to host network ports

Quick Start

1. Block Privileged Containers

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged-containers
  annotations:
    policies.kyverno.io/title: Disallow Privileged Containers
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Privileged mode disables most security mechanisms and must not be a
llowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: privileged-containers
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Privileged mode is disallowed. The fields spec.containers[*].se
curityContext.privileged,
          spec.initContainers[*].securityContext.privileged, and spec.eph
emeralContainers[*].securityContext.privileged
          must be unset or set to false.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(securityContext):
                  =(privileged): "false"
            =(initContainers):
              - =(securityContext):
                  =(privileged): "false"
            containers:
              - =(securityContext):
                  =(privileged): "false"
```

2. Test the Policy

```
# Apply the policy
kubectl apply -f disallow-privileged-containers.yaml

# Try to create a privileged container (should fail)
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-privileged
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      privileged: true
EOF

# Try to create a normal container (should work)
kubectl run test-normal --image=nginx

# Clean up
kubectl delete pod test-privileged test-normal --ignore-not-found
```

Core Container Escape Prevention Policies

Policy 1: Disallow Host Namespace Access

Prevent containers from accessing host namespaces:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-namespaces
  annotations:
    policies.kyverno.io/title: Disallow Host Namespaces
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Host namespaces (Process ID namespace, Inter-Process Communication
namespace, and
      network namespace) allow access to shared information and can be us
ed to elevate
      privileges. Pods should not be allowed access to host namespaces.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-namespaces
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Sharing the host namespaces is disallowed. The fields spec.host
Network,
          spec.hostIPC, and spec.hostPID must be unset or set to false.
        pattern:
          spec:
            =(hostPID): "false"
            =(hostIPC): "false"
            =(hostNetwork): "false"
```

Policy 2: Disallow Host Path Mounts

Block containers from mounting host filesystem paths:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-path
  annotations:
    policies.kyverno.io/title: Disallow Host Path
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      HostPath volumes let Pods use host directories and volumes in conta
iners.
      Using host resources can be used to access shared data or escalate
privileges
      and should not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-path
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          HostPath volumes are forbidden. The field spec.volumes[*].hostP
ath must be unset.
        pattern:
          spec:
            =(volumes):
              - X(hostPath): "null"
```

Policy 3: Disallow Host Ports

Prevent containers from binding to host network ports:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-ports
  annotations:
    policies.kyverno.io/title: Disallow Host Ports
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Access to host ports allows potential snooping of network traffic and should not be
      allowed, or at minimum restricted to a known list.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-ports-none
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host ports is disallowed. The fields spec.containers[*].ports[*].hostPort,
          spec.initContainers[*].ports[*].hostPort, and spec.ephemeralContainers[*].ports[*].hostPort
          must either be unset or set to 0.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(ports):
                  - =(hostPort): 0
            =(initContainers):
              - =(ports):
                  - =(hostPort): 0
            containers:
              - =(ports):
                  - =(hostPort): 0
```

Policy 4: Disallow Dangerous Capabilities

Block containers from adding dangerous Linux capabilities:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-capabilities-strict
  annotations:
    policies.kyverno.io/title: Disallow Capabilities (Strict)
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Adding capabilities other than `NET_BIND_SERVICE` is disallowed. In addition,
      all containers must explicitly drop `ALL` capabilities.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: require-drop-all
      match:
        any:
        - resources:
            kinds:
            - Pod
      preconditions:
        all:
        - key: "{{ request.operation || 'BACKGROUND' }}"
          operator: NotEquals
          value: DELETE
      validate:
        message: >-
          Containers must drop `ALL` capabilities.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers, containers][]
          deny:
            conditions:
              all:
              - key: ALL
                operator: AnyNotIn
                value: "{{ element.securityContext.capabilities.drop || `[]` }}"
    - name: adding-capabilities
      match:
```

```yaml
            any:
            - resources:
                kinds:
                - Pod
      preconditions:
        all:
        - key: "{{ request.operation || 'BACKGROUND' }}"
          operator: NotEquals
          value: DELETE
      validate:
        message: >-
          Any capabilities added other than NET_BIND_SERVICE are disallow
ed.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers,
containers][]
          deny:
            conditions:
              any:
              - key: "{{ element.securityContext.capabilities.add || `[]`
}}"
                operator: AnyNotIn
                value:
                - NET_BIND_SERVICE
```

Policy 5: Require Non-Root Containers

Ensure containers run as non-root users:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-run-as-nonroot
  annotations:
    policies.kyverno.io/title: Require Run As Non-Root User
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Containers must run as a non-root user. This policy ensures runAsNonRoot is set to true.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: run-as-non-root
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Running as root is not allowed. Either the field spec.securityContext.runAsNonRoot
          must be set to true, or the field spec.containers[*].securityContext.runAsNonRoot
          must be set to true.
        anyPattern:
        - spec:
            securityContext:
              runAsNonRoot: "true"
        - spec:
            containers:
            - securityContext:
                runAsNonRoot: "true"
```

Advanced Scenarios

Scenario 1: Environment-Specific Policies

Different security levels for different environments:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-container-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Production: Strict security
    - name: production-strict-security
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production environments require strict container security"
        pattern:
          spec:
            =(hostPID): "false"
            =(hostIPC): "false"
            =(hostNetwork): "false"
            securityContext:
              runAsNonRoot: "true"
            containers:
            - securityContext:
                privileged: "false"
                runAsNonRoot: "true"
                capabilities:
                  drop:
                  - ALL

    # Development: More permissive but still secure
    - name: development-basic-security
      match:
        any:
        - resources:
            kinds:
            - Pod
```

```yaml
        namespaces:
          - development
          - dev-*
          - staging
    validate:
      message: "Development environments require basic container securi
ty"
      pattern:
        spec:
          =(hostPID): "false"
          =(hostIPC): "false"
          containers:
          - securityContext:
              =(privileged): "false"
```

Scenario 2: Workload-Specific Exceptions

Allow specific workloads with controlled exceptions:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: workload-specific-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: system-workloads-exception
      match:
        any:
        - resources:
            kinds:
            - Pod
      exclude:
        any:
        - resources:
            namespaces:
            - kube-system
            - kyverno
        - resources:
            kinds:
            - Pod
            names:
            - "monitoring-*"
            - "logging-*"
      validate:
        message: "Container security policies apply to application worklo
ads"
        pattern:
          spec:
            =(hostNetwork): "false"
            containers:
            - securityContext:
                =(privileged): "false"
```

Testing and Validation

Test Privileged Container

```
# This should be blocked
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-privileged
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      privileged: true
EOF
```

Test Host Namespace Access

```
# This should be blocked
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-host-network
spec:
  hostNetwork: true
  containers:
  - name: test
    image: nginx
EOF
```

Test Host Path Mount

```
# This should be blocked
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-hostpath
spec:
  containers:
  - name: test
    image: nginx
    volumeMounts:
    - name: host-vol
      mountPath: /host
  volumes:
  - name: host-vol
    hostPath:
      path: /
EOF
```

Test Valid Secure Container

```
# This should be blocked
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
```

```
# This should be allowed
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-secure
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
  containers:
  - name: test
    image: nginx
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      runAsUser: 1000
EOF
```

Best Practices

1. Start with Audit Mode

```
spec:
  validationFailureAction: Audit   # Start with warnings, not blocking
```

2. Exclude System Namespaces

```yaml
exclude:
  any:
  - resources:
      namespaces:
      - kube-system
      - kyverno
      - kube-public
```

```yaml
exclude:
  any:
  - resources:
      namespaces:
```

Alauda Container Platform

Security Context Enforcement Policy

This guide demonstrates how to configure Kyverno to enforce proper security contexts for containers, ensuring they run with appropriate security settings and restrictions.

TOC

Test Missing Capabilities Drop (Should Fail)

Test Valid Secure Container (Should Pass)

What is Security Context Enforcement?

Security context enforcement involves controlling how containers run by setting security-related parameters. Proper security context configuration prevents:

- **Root privilege escalation**: Containers running as root user

- **Privilege escalation attacks**: Containers gaining elevated permissions

- **Insecure process execution**: Containers running with dangerous capabilities

- **Filesystem tampering**: Containers with writable root filesystems

- **Security bypass**: Containers circumventing security mechanisms

Quick Start

1. Require Non-Root Containers Policy

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-run-as-nonroot
  annotations:
    policies.kyverno.io/title: Require Run As Non-Root User
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Containers must run as a non-root user. This policy ensures runAsNonRoot is set to true.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: run-as-non-root
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Running as root is not allowed. Either the field spec.securityContext.runAsNonRoot
          must be set to true, or the field spec.containers[*].securityContext.runAsNonRoot
          must be set to true.
        anyPattern:
        - spec:
            securityContext:
              runAsNonRoot: "true"
        - spec:
            containers:
            - securityContext:
                runAsNonRoot: "true"
```

2. Test the Policy

```
# Apply the policy
kubectl apply -f require-run-as-nonroot.yaml

# Try to create a container explicitly running as root (should fail)
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-root
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      runAsUser: 0
      runAsNonRoot: false
EOF

# Try to create a container with non-root user (should work)
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-nonroot
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
  containers:
  - name: nginx
    image: nginx
EOF

# Clean up
kubectl delete pod test-root test-nonroot --ignore-not-found
```

Core Security Context Policies

Policy 1: Disallow Privilege Escalation

Prevent containers from escalating privileges:

Prevent containers from escalating privileges:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privilege-escalation
  annotations:
    policies.kyverno.io/title: Disallow Privilege Escalation
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Privilege escalation, such as via set-user-ID or set-group-ID file
mode, should not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: privilege-escalation
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Privilege escalation is disallowed. The fields
          spec.containers[*].securityContext.allowPrivilegeEscalation,
          spec.initContainers[*].securityContext.allowPrivilegeEscalatio
n,
          and spec.ephemeralContainers[*].securityContext.allowPrivilegeE
scalation
          must be set to false.
        pattern:
          spec:
            =(ephemeralContainers):
              - securityContext:
                  allowPrivilegeEscalation: "false"
            =(initContainers):
              - securityContext:
                  allowPrivilegeEscalation: "false"
            containers:
              - securityContext:
                  allowPrivilegeEscalation: "false"
```

Policy 2: Require Specific User ID Range

Ensure containers run with specific user IDs:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-user-id-range
  annotations:
    policies.kyverno.io/title: Require User ID Range
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Containers must run with a specific user ID range to prevent privilege escalation.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: user-id-range
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Containers must run with user ID between 1000 and 65535.
        deny:
          conditions:
            any:
            # Check pod-level security context
            - key: "{{ request.object.spec.securityContext.runAsUser || 0 }}"
              operator: LessThan
              value: 1000
            - key: "{{ request.object.spec.securityContext.runAsUser || 0 }}"
              operator: GreaterThan
              value: 65535
            # Check container-level security contexts
            - key: "{{ request.object.spec.containers[?securityContext.runAsUser && (securityContext.runAsUser < `1000` || securityContext.runAsUser > `65535`)] | length(@) }}"
              operator: GreaterThan
              value: 0
```

Policy 3: Require Non-Root Groups

Ensure containers run with non-root group IDs:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-non-root-groups
  annotations:
    policies.kyverno.io/title: Require Non-Root Groups
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Containers should be required to run with a non-root group ID or su
pplemental groups.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: non-root-groups
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Containers must run with non-root group ID. Either spec.securit
yContext.runAsGroup
          or spec.containers[*].securityContext.runAsGroup must be set an
d not be 0.
        deny:
          conditions:
            any:
            # Check if pod-level runAsGroup is 0
            - key: "{{ request.object.spec.securityContext.runAsGroup ||
0 }}"
              operator: Equals
              value: 0
            # Check if any container has runAsGroup set to 0
            - key: "{{ request.object.spec.containers[?securityContext.ru
nAsGroup == `0`] | length(@) }}"
              operator: GreaterThan
              value: 0
```

Policy 4: Restrict Seccomp Profiles

Enforce secure seccomp profiles:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-seccomp-strict
  annotations:
    policies.kyverno.io/title: Restrict Seccomp (Strict)
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Seccomp profile must be explicitly set to one of the allowed values.
      Both the Unconfined profile and the absence of a profile are prohibited.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: seccomp-strict
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of custom Seccomp profiles is disallowed. The field
          spec.securityContext.seccompProfile.type must be set to RuntimeDefault or Localhost.
        anyPattern:
        - spec:
            securityContext:
              seccompProfile:
                type: RuntimeDefault
        - spec:
            securityContext:
              seccompProfile:
                type: Localhost
        - spec:
            containers:
            - securityContext:
                seccompProfile:
                  type: RuntimeDefault
```

```
    - spec:
        containers:
        - securityContext:
            seccompProfile:
              type: Localhost
```

Policy 5: Require Dropping ALL Capabilities

Ensure containers drop all capabilities:

```
    - spec:
        containers:
        - securityContext:
```
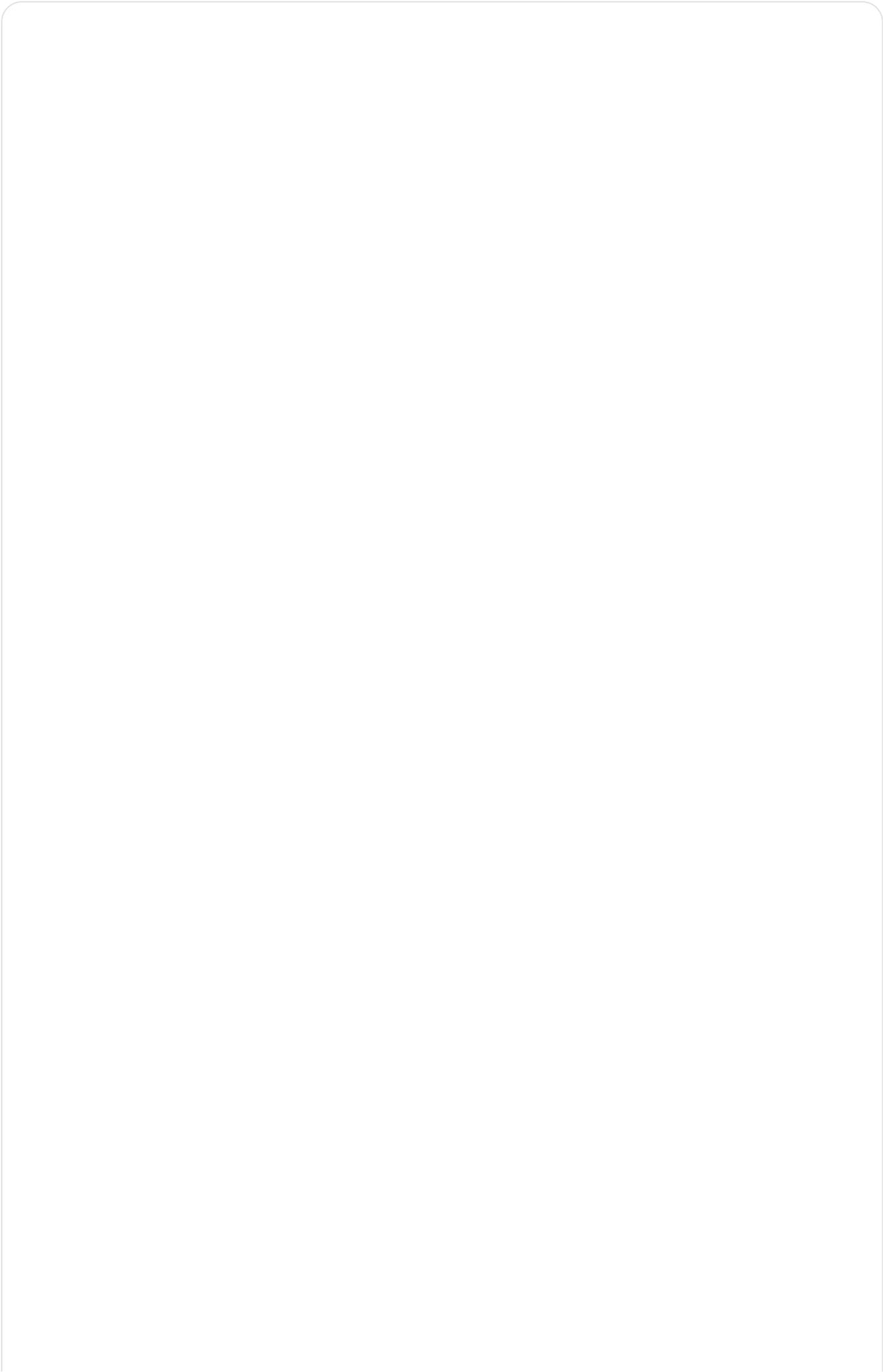
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-drop-all-capabilities
  annotations:
    policies.kyverno.io/title: Require Drop ALL Capabilities
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Containers must drop all capabilities and only add back those that
are specifically needed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: require-drop-all
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Containers must drop ALL capabilities.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers,
containers][]
          deny:
            conditions:
              all:
              - key: ALL
                operator: AnyNotIn
                value: "{{ element.securityContext.capabilities.drop || `
[]` }}"
```

Policy 6: Restrict AppArmor Profiles

Control AppArmor profile usage:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-apparmor-profiles
  annotations:
    policies.kyverno.io/title: Restrict AppArmor Profiles
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      On supported hosts, the runtime/default AppArmor profile is applied by default.
      The baseline policy should prevent overriding or disabling the default AppArmor profile.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: apparmor-profiles
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          AppArmor profile must be set to runtime/default or a custom profile.
          Unconfined profiles are not allowed.
        pattern:
          metadata:
            =(annotations):
              =(container.apparmor.security.beta.kubernetes.io/*): "!unconfined"
```

Advanced Scenarios

Scenario 1: Environment-Specific Security Contexts

Different security requirements for different environments:
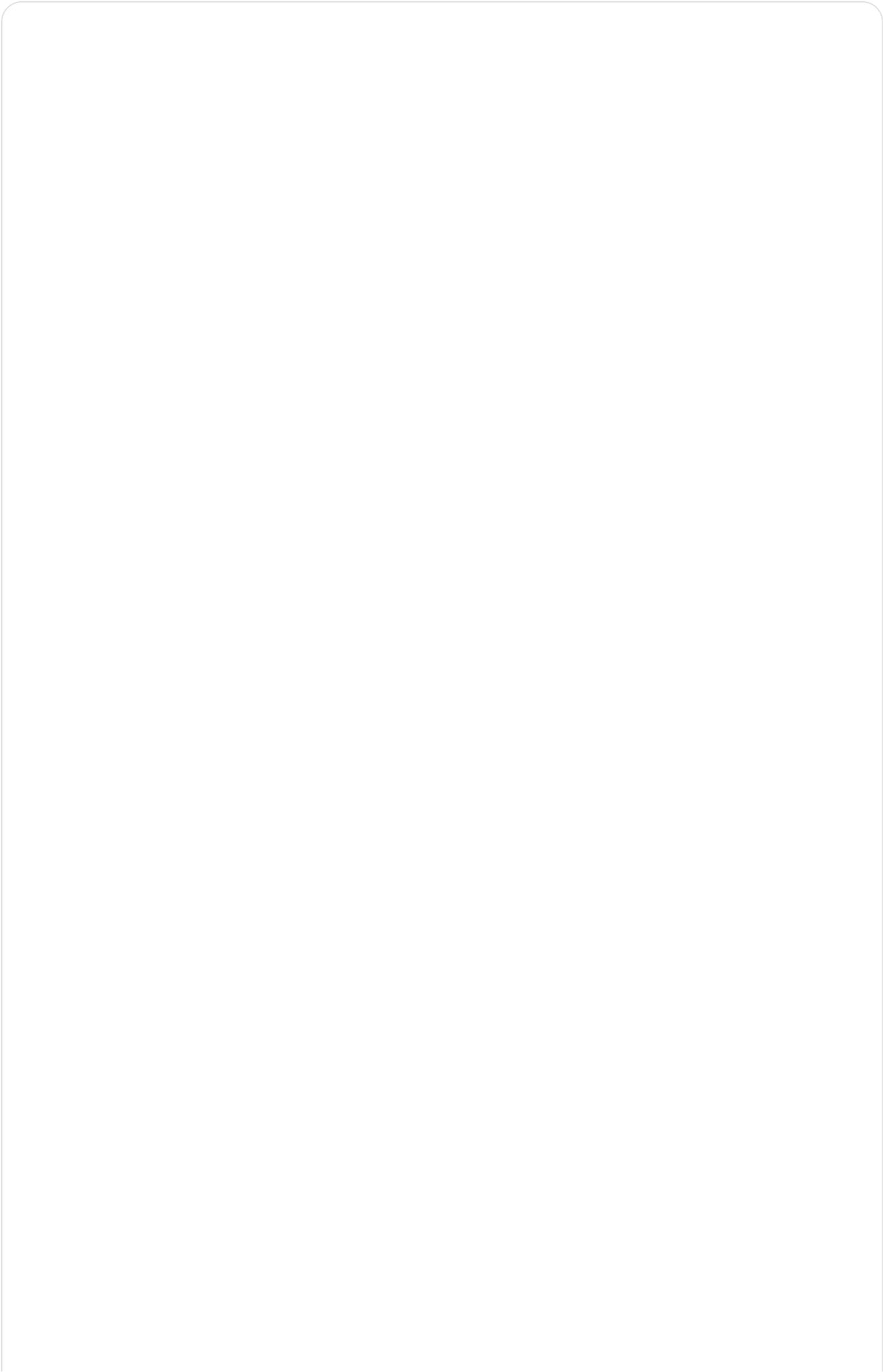
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Production: Strict security contexts
    - name: production-strict-security
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production environments require strict security contexts"
        pattern:
          spec:
            securityContext:
              runAsNonRoot: "true"
              runAsUser: "1000-65535"
              runAsGroup: "1000-65535"
              seccompProfile:
                type: RuntimeDefault
            containers:
            - securityContext:
                allowPrivilegeEscalation: "false"
                readOnlyRootFilesystem: "true"
                runAsNonRoot: "true"
                capabilities:
                  drop:
                  - ALL

    # Development: Basic security requirements
    - name: development-basic-security
      match:
        any:
        - resources:
```

```yaml
        kinds:
        - Pod
        namespaces:
        - development
        - dev-*
        - staging
    validate:
      message: "Development environments require basic security context
  s"
      pattern:
        spec:
          containers:
          - securityContext:
              allowPrivilegeEscalation: "false"
              runAsNonRoot: "true"
```

Scenario 2: Application-Specific Security Contexts

Different security contexts for different application types:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Database applications: Specific user/group IDs
    - name: database-security-context
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: database
      validate:
        message: "Database applications must use specific security contexts"
        pattern:
          spec:
            securityContext:
              runAsUser: "999"
              runAsGroup: "999"
              fsGroup: "999"
            containers:
            - securityContext:
                runAsNonRoot: "true"
                readOnlyRootFilesystem: "true"

    # Web applications: Standard security context
    - name: web-app-security-context
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: web
      validate:
```

```
      message: "Web applications must use standard security contexts"
      pattern:
        spec:
          containers:
          - securityContext:
              runAsNonRoot: "true"
              allowPrivilegeEscalation: "false"
              capabilities:
                drop:
                - ALL
                add:
                - NET_BIND_SERVICE
```

Scenario 3: Graduated Security Context Enforcement

Implement progressive security context requirements:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: graduated-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Level 1: Basic security (all namespaces)
    - name: basic-security-level
      match:
        any:
        - resources:
            kinds:
            - Pod
      exclude:
        any:
        - resources:
            namespaces:
            - kube-system
            - kyverno
      validate:
        message: "All containers must have basic security contexts"
        pattern:
          spec:
            containers:
            - securityContext:
                allowPrivilegeEscalation: "false"

    # Level 2: Enhanced security (sensitive namespaces)
    - name: enhanced-security-level
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - finance-*
            - hr-*
            - security-*
      validate:
        message: "Sensitive namespaces require enhanced security context
s"
```

```yaml
          pattern:
            spec:
              securityContext:
                runAsNonRoot: "true"
              containers:
                - securityContext:
                    readOnlyRootFilesystem: "true"
                    capabilities:
                      drop:
                        - ALL

      # Level 3: Maximum security (critical namespaces)
      - name: maximum-security-level
        match:
          any:
          - resources:
              kinds:
                - Pod
              namespaces:
                - critical-*
                - payment-*
        validate:
          message: "Critical namespaces require maximum security contexts"
          pattern:
            spec:
              securityContext:
                runAsNonRoot: "true"
                runAsUser: "1000-1999"
                runAsGroup: "1000-1999"
                seccompProfile:
                  type: RuntimeDefault
              containers:
                - securityContext:
                    allowPrivilegeEscalation: "false"
                    readOnlyRootFilesystem: "true"
                    runAsNonRoot: "true"
                    capabilities:
                      drop:
                        - ALL
```

Scenario 4: Specified Namespace Security Context Enforcement

Implementing label-based namespace security policy injection:
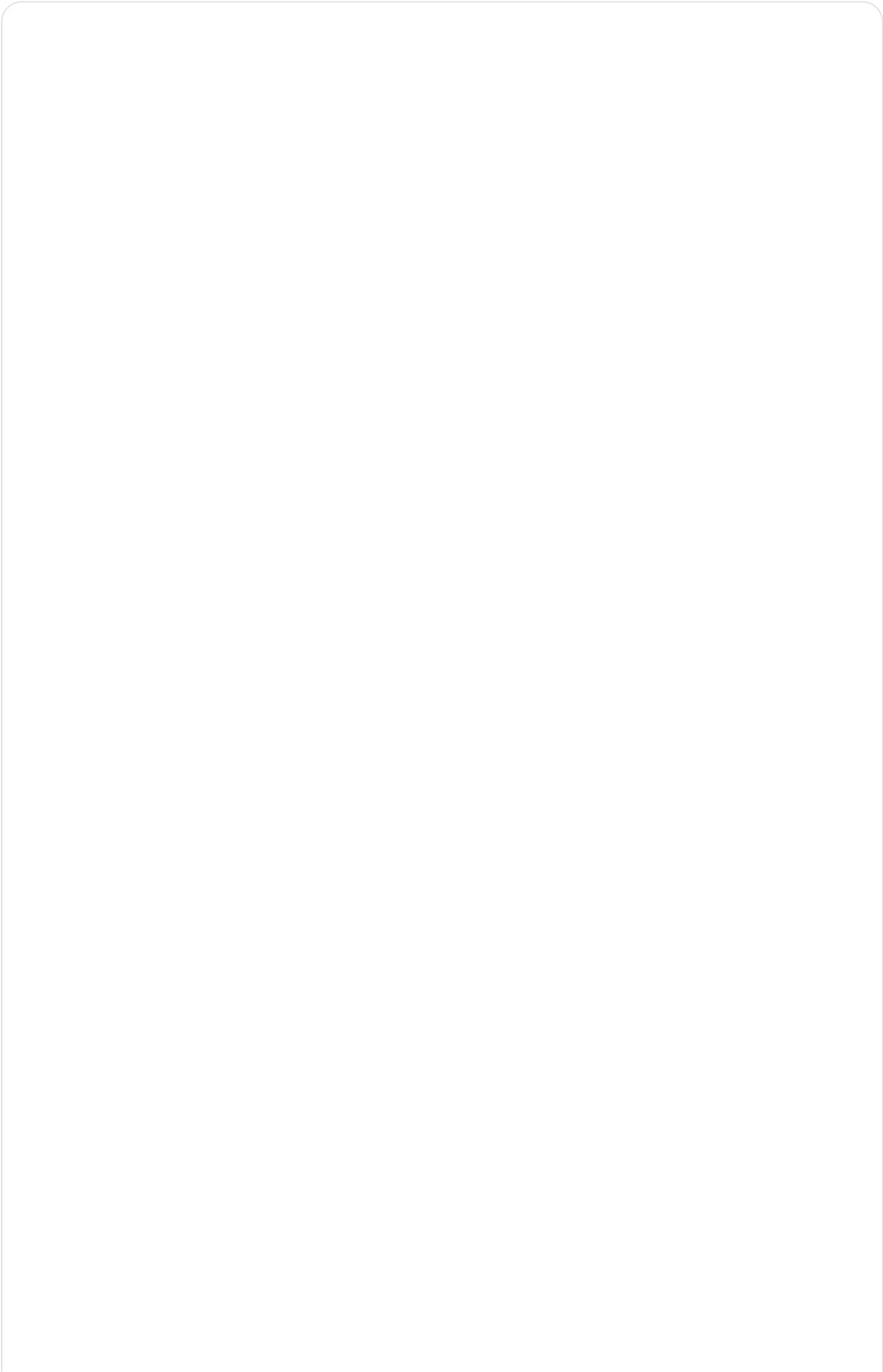
```yaml
apiVersion: kyverno.io/v1
kind: Policy
metadata:
  annotations:
    policies.kyverno.io/category: Pod Security Standards
    policies.kyverno.io/description: 'Strictly follow the Security Context configuration in the image to automatically add, allowPrivilegeEscalation:
      false, capabilities drop ALL,  runAsNonRoot: true, seccompProfile: RuntimeDefault'
    policies.kyverno.io/minversion: 1.13.0
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/title: Add precise Security Context configuration
  creationTimestamp: "2025-06-04T03:26:54Z"
  generation: 1
  labels:
    velero.io/backup-name: app-backup-20250604111354
    velero.io/restore-name: app-recovery
  name: add-exact-security-context
  namespace: test-1
spec:
  admission: true
  background: true
  emitWarning: false
  rules:
  - match:
      any:
      - resources:
          kinds:
          - Pod
    context:
    - name: namespaceInfo
      apiCall:
        urlPath: "/api/v1/namespaces/{{request.namespace}}"
        jmesPath: "metadata.labels.\"pod-security.kubernetes.io/enforce\"
|| 'restricted'"
    preconditions:
      all:
      - key: "{{ namespaceInfo }}"
        operator: NotEquals
        value: "privileged"
    mutate:
```

```yaml
      foreach:
      - list: request.object.spec.containers
        patchStrategicMerge:
          spec:
            containers:
            - name: '{{ element.name }}'
              securityContext:
                allowPrivilegeEscalation: false
                capabilities:
                  drop:
                  - ALL
                runAsNonRoot: true
                seccompProfile:
                  type: RuntimeDefault
    name: add-container-security-context
    skipBackgroundRequests: true
  - match:
      any:
      - resources:
          kinds:
          - Pod
    context:
    - name: namespaceInfo
      apiCall:
        urlPath: "/api/v1/namespaces/{{request.namespace}}"
        jmesPath: "metadata.labels.\"pod-security.kubernetes.io/enforce\"
|| 'restricted'"
    preconditions:
      all:
      - key: "{{ namespaceInfo }}"
        operator: NotEquals
        value: "privileged"
      - key: '{{ request.object.spec.initContainers || `[]` | length(@)
}}'
        operator: GreaterThan
        value: 0
    mutate:
      foreach:
      - list: request.object.spec.initContainers
        patchStrategicMerge:
          spec:
            initContainers:
            - name: '{{ element.name }}'
              securityContext:
```

```yaml
                    allowPrivilegeEscalation: false
                    capabilities:
                      drop:
                      - ALL
                    runAsNonRoot: true
                    seccompProfile:
                      type: RuntimeDefault
        name: add-init-container-security-context
        skipBackgroundRequests: true
    - match:
        any:
        - resources:
            kinds:
            - Pod
      context:
      - name: namespaceInfo
        apiCall:
          urlPath: "/api/v1/namespaces/{{request.namespace}}"
          jmesPath: "metadata.labels.\"pod-security.kubernetes.io/enforce\"
|| 'restricted'"
      preconditions:
        all:
        - key: "{{ namespaceInfo }}"
          operator: NotEquals
          value: "privileged"
        - key: '{{ request.object.spec.ephemeralContainers || `[]` | length
(@) }}'
          operator: GreaterThan
          value: 0
      mutate:
        foreach:
        - list: request.object.spec.ephemeralContainers
          patchStrategicMerge:
            spec:
              ephemeralContainers:
              - name: '{{ element.name }}'
                securityContext:
                  allowPrivilegeEscalation: false
                  capabilities:
                    drop:
                    - ALL
                  runAsNonRoot: true
                  seccompProfile:
                    type: RuntimeDefault
```

```
    name: add-ephemeral-container-security-context
    skipBackgroundRequests: true
  validationFailureAction: Audit
```

If you do not expect certain namespaces to be injected with security context detection, please add the label for namespace `pod-security.kubernetes.io/enforce: privileged`

Testing and Validation

Test Root Container (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-root-user
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      runAsUser: 0
EOF
```

Test Privilege Escalation (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-privilege-escalation
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      allowPrivilegeEscalation: true
EOF
```

Test Missing Capabilities Drop (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-missing-drop-all
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
EOF
```

Test Valid Secure Container (Should Pass)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-secure-context
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test
    image: nginx
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      runAsUser: 1000
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
EOF
```

Alauda Container Platform

Network Security Policy

This guide demonstrates how to configure Kyverno to enforce network security policies that control container network access and prevent network-based attacks.

TOC

Test Valid Network Configuration (Should Pass)

What is Network Security?

Network security involves controlling how containers access and interact with network resources. Proper network security prevents:

- **Host network access**: Containers accessing host network interfaces

- **Privilege escalation via networking**: Using network access to gain elevated permissions

- **Port scanning and reconnaissance**: Unauthorized network discovery activities

- **Lateral movement**: Containers accessing unintended network resources

- **Data exfiltration**: Unauthorized network communications

Quick Start

1. Disallow Host Network Access

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-network
  annotations:
    policies.kyverno.io/title: Disallow Host Network
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Access to the host network allows potential snooping of network traffic and should not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-network
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host network is disallowed. The field spec.hostNetwork must be unset or set to false.
        pattern:
          spec:
            =(hostNetwork): "false"
```

2. Test the Policy

```
# Apply the policy
kubectl apply -f disallow-host-network.yaml

# Try to create a pod with host network (should fail)
kubectl run test-hostnet --image=nginx --overrides='{"spec":{"hostNetwor
k":true}}'

# Try to create a normal pod (should work)
kubectl run test-normal --image=nginx
```

Core Network Security Policies

Policy 1: Disallow Host Ports

Prevent containers from binding to host network ports:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-ports
  annotations:
    policies.kyverno.io/title: Disallow Host Ports
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Access to host ports allows potential snooping of network traffic and should not be
      allowed, or at minimum restricted to a known list.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-ports-none
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host ports is disallowed. The fields spec.containers[*].ports[*].hostPort,
          spec.initContainers[*].ports[*].hostPort, and spec.ephemeralContainers[*].ports[*].hostPort
          must either be unset or set to 0.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(ports):
                  - =(hostPort): 0
            =(initContainers):
              - =(ports):
                  - =(hostPort): 0
            containers:
              - =(ports):
                  - =(hostPort): 0
```

Policy 2: Restrict Host Port Range

Allow specific host port ranges for controlled access:

Policy 2: Restrict Host Port Range

Allow specific host port ranges for controlled access:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-host-port-range
  annotations:
    policies.kyverno.io/title: Restrict Host Port Range
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Host ports, if used, must be within an allowed range to prevent conflicts and security issues.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-port-range
      match:
        any:
        - resources:
            kinds:
            - Pod
      preconditions:
        all:
        - key: "{{ request.object.spec.containers[].ports[?hostPort] | length(@) }}"
          operator: GreaterThan
          value: 0
      validate:
        message: >-
          Host ports must be within the allowed range 30000-32767.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers, containers][].ports[]
          preconditions:
            any:
            - key: "{{ element.hostPort }}"
              operator: GreaterThan
              value: 0
          deny:
            conditions:
              any:
              - key: "{{ element.hostPort }}"
```

```yaml
          operator: LessThan
          value: 30000
      - key: "{{ element.hostPort }}"
          operator: GreaterThan
          value: 32767
```

Policy 3: Require Network Policies

Ensure pods have associated NetworkPolicies for traffic control:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-network-policies
  annotations:
    policies.kyverno.io/title: Require Network Policies
    policies.kyverno.io/category: Network Security
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,NetworkPolicy
    policies.kyverno.io/description: >-
      Pods should have associated NetworkPolicies to control network traffic.
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: require-netpol
      match:
        any:
        - resources:
            kinds:
            - Pod
      exclude:
        any:
        - resources:
            namespaces:
            - kube-system
            - kyverno
      context:
      - name: netpols
        apiCall:
          urlPath: "/apis/networking.k8s.io/v1/namespaces/{{ request.namespace }}/networkpolicies"
          jmesPath: "items[?spec.podSelector.matchLabels.app == '{{ request.object.metadata.labels.app }}'] | length(@)"
      validate:
        message: >-
          Pods must have an associated NetworkPolicy. Create a NetworkPolicy that selects this pod.
        deny:
          conditions:
            all:
            - key: "{{ netpols }}"
              -
```

```
          operator: Equals
          value: 0
```

Policy 4: Restrict Service Types

Control which service types can be created:
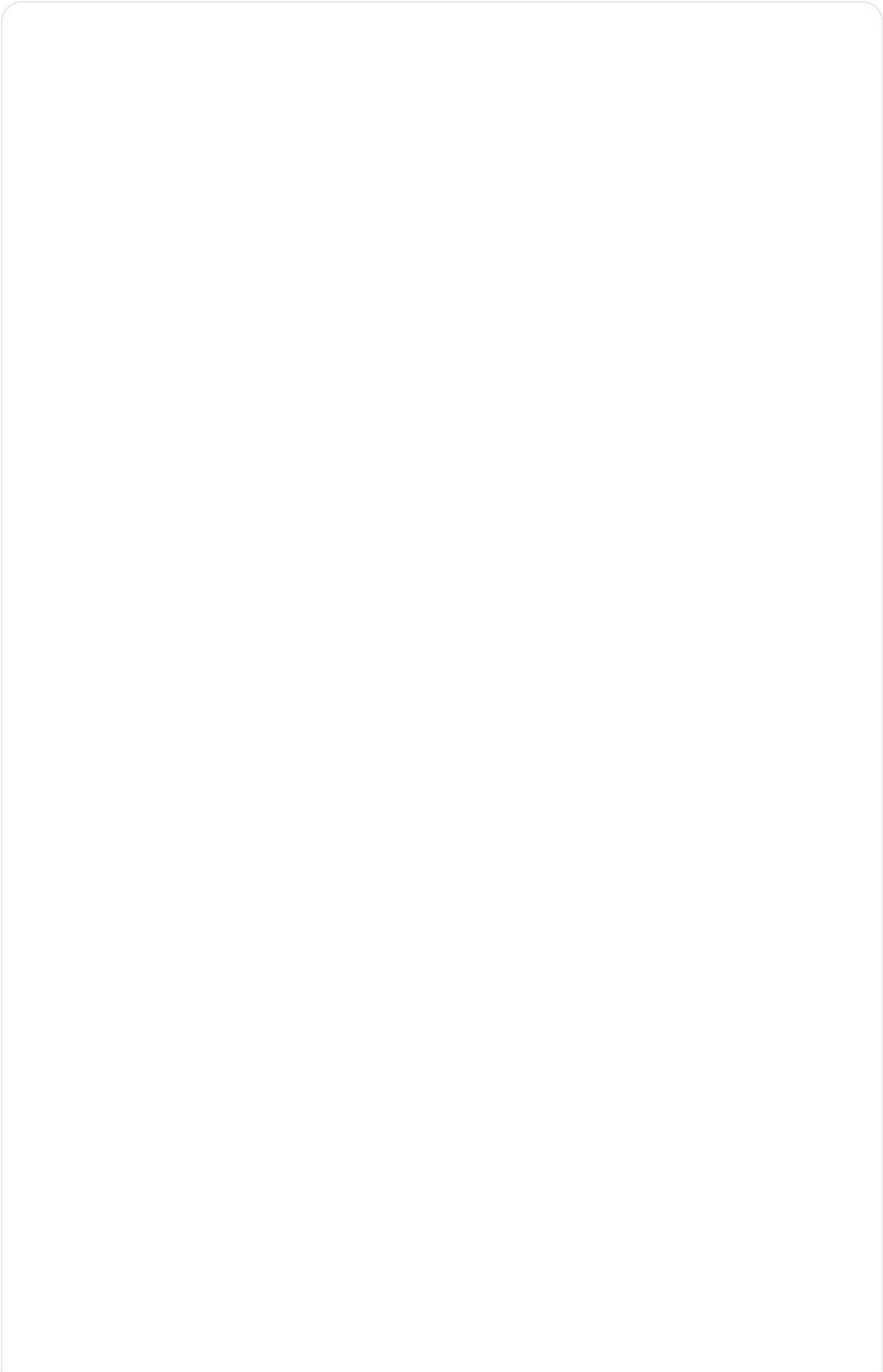
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-service-types
  annotations:
    policies.kyverno.io/title: Restrict Service Types
    policies.kyverno.io/category: Network Security
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Service
    policies.kyverno.io/description: >-
      Restrict Service types to prevent exposure of services to external
networks.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-nodeport
      match:
        any:
        - resources:
            kinds:
            - Service
      validate:
        message: >-
          NodePort services are not allowed. Use ClusterIP or LoadBalance
r instead.
        pattern:
          spec:
            type: "!NodePort"
    - name: restrict-loadbalancer
      match:
        any:
        - resources:
            kinds:
            - Service
            namespaces:
            - development
            - dev-*
            - staging
      validate:
        message: >-
          LoadBalancer services are not allowed in development environmen
ts.
```

```
      pattern:
        spec:
          type: "!LoadBalancer"
```

Policy 5: Control Ingress Configurations

Enforce secure Ingress configurations:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: secure-ingress-configuration
  annotations:
    policies.kyverno.io/title: Secure Ingress Configuration
    policies.kyverno.io/category: Network Security
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Ingress
    policies.kyverno.io/description: >-
      Ingress resources must be configured securely with TLS and proper a
nnotations.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: require-tls
      match:
        any:
        - resources:
            kinds:
            - Ingress
      validate:
        message: >-
          Ingress must use TLS. The field spec.tls must be specified.
        pattern:
          spec:
            tls:
            - hosts:
              - "*"
    - name: require-security-annotations
      match:
        any:
        - resources:
            kinds:
            - Ingress
      validate:
        message: >-
          Ingress must have security annotations for SSL redirect and HST
S.
        pattern:
          metadata:
            annotations:
```

```
        nginx.ingress.kubernetes.io/ssl-redirect: "true"
        nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
```

Policy 6: Restrict DNS Configuration

Control DNS settings to prevent DNS-based attacks:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-dns-configuration
  annotations:
    policies.kyverno.io/title: Restrict DNS Configuration
    policies.kyverno.io/category: Network Security
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Restrict DNS configuration to prevent DNS hijacking and data exfiltration.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-dns-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Custom DNS policy is not allowed. Use Default or ClusterFirst only.
        pattern:
          spec:
            =(dnsPolicy): "Default | ClusterFirst"
    - name: restrict-custom-dns
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Custom DNS configuration is not allowed in production environments.
        pattern:
          spec:
            X(dnsConfig): "null"
```

Advanced Scenarios

Scenario 1: Environment-Specific Network Policies

Different network restrictions for different environments:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-network-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Production: Strict network controls
    - name: production-network-restrictions
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production environments require strict network security"
        pattern:
          spec:
            hostNetwork: "false"
            dnsPolicy: "ClusterFirst"
            containers:
            - ports:
              - =(hostPort): 0

    # Development: Basic network security
    - name: development-network-restrictions
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - development
            - dev-*
            - staging
      validate:
        message: "Development environments require basic network security"
```

```
      pattern:
        spec:
          hostNetwork: "false"
```

Scenario 2: Application-Specific Network Policies

Different network policies for different application types:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-network-policies
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Database applications: No external network access
    - name: database-network-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: database
      validate:
        message: "Database applications cannot use host network or host ports"
        pattern:
          spec:
            hostNetwork: "false"
            containers:
            - ports:
              - =(hostPort): 0

    # Web applications: Controlled port access
    - name: web-app-network-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: web
      validate:
        message: "Web applications can only use standard HTTP/HTTPS ports"
        foreach:
        - list: request.object.spec.containers[].ports[]
```

```yaml
      deny:
        conditions:
          any:
          - key: "{{ element.containerPort }}"
            operator: AnyNotIn
            value:
            - 80
            - 443
            - 8080
            - 8443
```

Scenario 3: Network Segmentation Enforcement

Enforce network segmentation between different tiers:

```yaml
      deny:
        conditions:
          any:
          - key: "{{ element.containerPort }}"

            443
```
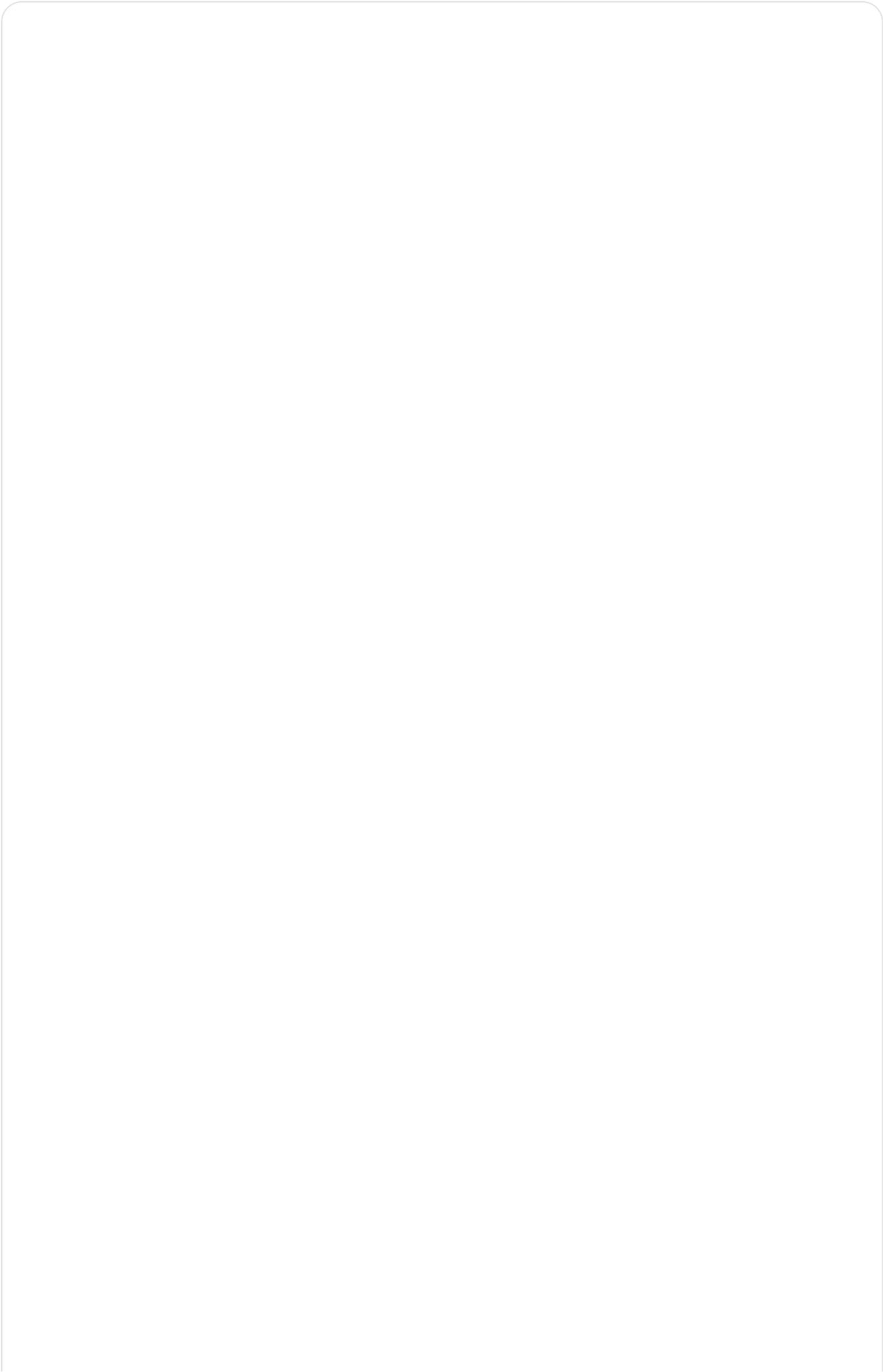
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: network-segmentation-enforcement
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: frontend-backend-separation
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                tier: frontend
      validate:
        message: "Frontend pods cannot access backend network directly"
        deny:
          conditions:
            any:
            - key: "{{ request.object.metadata.labels.tier }}"
              operator: Equals
              value: backend
    - name: require-network-labels
      match:
        any:
        - resources:
            kinds:
            - Pod
      exclude:
        any:
        - resources:
            namespaces:
            - kube-system
            - kyverno
      validate:
        message: "Pods must have network tier labels for segmentation"
        pattern:
          metadata:
            labels:
              tier: "frontend | backend | database"
```

Testing and Validation

Test Host Network Access (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-host-network
spec:
  hostNetwork: true
  containers:
  - name: test
    image: nginx
EOF
```

Test Host Port Binding (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-host-port
spec:
  containers:
  - name: test
    image: nginx
    ports:
    - containerPort: 80
      hostPort: 8080
EOF
```

Test NodePort Service (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: test-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
  selector:
    app: test
EOF
```

Test Valid Network Configuration (Should Pass)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
```

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-secure-network
  labels:
    app: web-app
    tier: frontend
spec:
  dnsPolicy: ClusterFirst
  containers:
  - name: test
    image: nginx
    ports:
    - containerPort: 80
      protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: test-service
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: web-app
EOF
```

```
cat <<EOF | kubectl apply -f -
```

Alauda Container Platform

Volume Security Policy

This guide demonstrates how to configure Kyverno to enforce volume security policies that restrict dangerous volume types and configurations that could compromise container security.

TOC

What is Volume Security?

Volume security involves controlling which types of volumes containers can mount and how they can access them. Proper volume security prevents:

- **Host filesystem access**: Unauthorized access to host directories

- **Privilege escalation**: Using volumes to gain elevated permissions

- **Data exfiltration**: Accessing sensitive host data through volume mounts

- **Container escape**: Breaking out of container isolation via volume access

- **Insecure volume types**: Using volume types that bypass security controls

Quick Start

1. Restrict Volume Types

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-volume-types
  annotations:
    policies.kyverno.io/title: Restrict Volume Types
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      Only allow safe volume types. This policy restricts volumes to configMap, csi,
      downwardAPI, emptyDir, ephemeral, persistentVolumeClaim, projected,
and secret.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-volume-types
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Only the following types of volumes may be used: configMap, csi, downwardAPI,
          emptyDir, ephemeral, persistentVolumeClaim, projected, and secret.
        foreach:
        - list: "request.object.spec.volumes || []"
          deny:
            conditions:
              all:
              - key: "{{ element.keys(@) }}"
                operator: AnyNotIn
                value:
                - name
                - configMap
                - csi
                - downwardAPI
                - emptyDir
```

```
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
```

2. Test the Policy

```
# Apply the policy
kubectl apply -f restrict-volume-types.yaml

# Try to create a pod with hostPath volume (should fail)
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-hostpath
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: host-vol
      mountPath: /host
  volumes:
  - name: host-vol
    hostPath:
      path: /
EOF

# Create a test ConfigMap first
kubectl create configmap test-config --from-literal=key=value

# Try to create a pod with allowed volume (should work)
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-configmap
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: config-vol
      mountPath: /config
  volumes:
  - name: config-vol
    configMap:
      name: test-config
EOF
```

```
# Clean up
kubectl delete pod test-hostpath test-configmap --ignore-not-found
kubectl delete configmap test-config --ignore-not-found
```

Core Volume Security Policies

Policy 1: Disallow HostPath Volumes

Prevent containers from mounting host filesystem paths:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-host-path
  annotations:
    policies.kyverno.io/title: Disallow Host Path
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      HostPath volumes let Pods use host directories and volumes in conta
iners.
      Using host resources can be used to access shared data or escalate
privileges
      and should not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-path
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          HostPath volumes are forbidden. The field spec.volumes[*].hostP
ath must be unset.
        pattern:
          spec:
            =(volumes):
              - X(hostPath): "null"
```

Policy 2: Restrict HostPath Volumes (Controlled Access)

Allow specific hostPath volumes with read-only access:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-host-path-readonly
  annotations:
    policies.kyverno.io/title: Restrict Host Path (Read-Only)
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      HostPath volumes which are allowed must be read-only and restricted
to specific paths.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-path-readonly
      match:
        any:
        - resources:
            kinds:
            - Pod
      preconditions:
        all:
        - key: "{{ request.object.spec.volumes[?hostPath] | length(@) }}"
          operator: GreaterThan
          value: 0
      validate:
        message: >-
          HostPath volumes must be read-only and limited to allowed path
s.
        foreach:
        - list: "request.object.spec.volumes[?hostPath]"
          deny:
            conditions:
              any:
              # Deny if path is not in allowed list
              - key: "{{ element.hostPath.path }}"
                operator: AnyNotIn
                value:
                - "/var/log"
                - "/var/lib/docker/containers"
                - "/proc"
```

```
          - "/sys"
      foreach:
      - list: "request.object.spec.containers[].volumeMounts[?name]"
        deny:
          conditions:
            any:
              # Deny if volume mount is not read-only
              - key: "{{ element.readOnly || false }}"
                operator: Equals
                value: false
```

Policy 3: Disallow Privileged Volume Types

Block volume types that can bypass security controls:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged-volumes
  annotations:
    policies.kyverno.io/title: Disallow Privileged Volume Types
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: high
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      Certain volume types are considered privileged and should not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: disallow-privileged-volumes
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Privileged volume types are not allowed: hostPath, gcePersistentDisk,
          awsElasticBlockStore, gitRepo, nfs, iscsi, glusterfs, rbd, flexVolume,
          cinder, cephFS, flocker, fc, azureFile, azureDisk, vsphereVolume, quobyte,
          portworxVolume, scaleIO, storageos.
        foreach:
        - list: "request.object.spec.volumes || []"
          deny:
            conditions:
              any:
              - key: "{{ element.keys(@) }}"
                operator: AnyIn
                value:
                - hostPath
                - gcePersistentDisk
                - awsElasticBlockStore
                - gitRepo
```

```
        - nfs
        - iscsi
        - glusterfs
        - rbd
        - flexVolume
        - cinder
        - cephFS
        - flocker
        - fc
        - azureFile
        - azureDisk
        - vsphereVolume
        - quobyte
        - portworxVolume
        - scaleIO
        - storageos
```

Policy 4: Require Read-Only Root Filesystem

Ensure containers use read-only root filesystems:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-readonly-rootfs
  annotations:
    policies.kyverno.io/title: Require Read-Only Root Filesystem
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      A read-only root file system helps to enforce an immutable infrastr
ucture strategy;
      the container only needs to write on the mounted volume that persis
ts the state.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: readonly-rootfs
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Root filesystem must be read-only. Set readOnlyRootFilesystem t
o true.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers,
containers][]
          deny:
            conditions:
              any:
              - key: "{{ element.securityContext.readOnlyRootFilesystem |
| false }}"
                operator: Equals
                value: false
```

Policy 5: Control Volume Mount Permissions

Restrict volume mount permissions and paths:

Restrict volume mount permissions and paths:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: control-volume-mounts
  annotations:
    policies.kyverno.io/title: Control Volume Mount Permissions
    policies.kyverno.io/category: Pod Security Standards (Restricted)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod,Volume
    policies.kyverno.io/description: >-
      Control where volumes can be mounted and with what permissions.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-mount-paths
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Volume mounts to sensitive paths are not allowed.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers, containers][].volumeMounts[]
          deny:
            conditions:
              any:
              # Block mounts to sensitive system paths
              - key: "{{ element.mountPath }}"
                operator: AnyIn
                value:
                - "/etc"
                - "/root"
                - "/var/run/docker.sock"
                - "/var/lib/kubelet"
                - "/var/lib/docker"
                - "/usr/bin"
                - "/usr/sbin"
                - "/sbin"
                - "/bin"
```

```yaml
    - name: require-readonly-sensitive-mounts
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Mounts to /proc and /sys must be read-only.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers,
containers][].volumeMounts[]
          preconditions:
            any:
            - key: "{{ element.mountPath }}"
              operator: AnyIn
              value:
              - "/proc"
              - "/sys"
          deny:
            conditions:
              any:
              - key: "{{ element.readOnly || false }}"
                operator: Equals
                value: false
```

Advanced Scenarios

Scenario 1: Environment-Specific Volume Policies

Different volume restrictions for different environments:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-volume-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Production: Strict volume controls
    - name: production-volume-restrictions
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production environments allow only secure volume types"
        foreach:
        - list: "request.object.spec.volumes || []"
          deny:
            conditions:
              all:
              - key: "{{ element.keys(@) }}"
                operator: AnyNotIn
                value:
                - name
                - configMap
                - secret
                - persistentVolumeClaim
                - emptyDir

    # Development: More permissive but still secure
    - name: development-volume-restrictions
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - development
```

```yaml
        - dev-*
        - staging
    validate:
      message: "Development environments allow additional volume types"
      foreach:
      - list: "request.object.spec.volumes || []"
        deny:
          conditions:
            any:
            - key: "{{ element.keys(@) }}"
              operator: AnyIn
              value:
              - hostPath   # Still block hostPath in dev
              - nfs        # Block network filesystems
```

Scenario 2: Application-Specific Volume Policies

Different volume policies for different application types:

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-volume-policies
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # Database applications: Allow persistent storage
    - name: database-volume-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: database
      validate:
        message: "Database applications must use persistent volumes"
        pattern:
          spec:
            volumes:
            - persistentVolumeClaim: {}

    # Web applications: Restrict to safe volumes
    - name: web-app-volume-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
            selector:
              matchLabels:
                app.type: web
      validate:
        message: "Web applications can only use safe volume types"
        foreach:
        - list: "request.object.spec.volumes || []"
          deny:
            conditions:
              all:
              - key: "{{ element.keys(@) }}"
```

```yaml
      operator: AnyNotIn
      value:
      - name
      - configMap
      - secret
      - emptyDir
      - projected
```

Scenario 3: Volume Size and Resource Limits

Control volume sizes and resource usage:

```yaml
      operator: AnyNotIn
      value:
      - name
      - configMap
      - secret
      - emptyDir
      - projected
```

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: volume-resource-limits
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: limit-emptydir-size
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "EmptyDir volumes must have size limits"
        foreach:
        - list: "request.object.spec.volumes[?emptyDir]"
          deny:
            conditions:
              any:
              - key: "{{ element.emptyDir.sizeLimit || '' }}"
                operator: Equals
                value: ""
    - name: limit-emptydir-memory
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "EmptyDir memory volumes are not allowed"
        foreach:
        - list: "request.object.spec.volumes[?emptyDir]"
          deny:
            conditions:
              any:
              - key: "{{ element.emptyDir.medium || '' }}"
                operator: Equals
                value: "Memory"
```

Testing and Validation

Test HostPath Volume (Should Fail)

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-hostpath
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: host-vol
      mountPath: /host
  volumes:
  - name: host-vol
    hostPath:
      path: /
EOF
```

Alauda Container Platform

API Refiner

Introduction

Product Introduction

Limitations

Install Alauda Container Platform API Refiner

Install via console

Install via YAML

Uninstallation Procedures

Default Configuration

Alauda Container Platform

Introduction

TOC

Product Introduction

ACP API Refiner is a data filtering service provided by the Alauda Container Platform that enhances multi-tenant security and data isolation in Kubernetes environments. It filters Kubernetes API response data based on user permissions, projects, clusters, and namespaces, while also supporting field-level filtering, inclusion, and data desensitization.

Limitations

The following limitations apply to ACP API Refiner:

- Resources must contain specific tenant-related labels for data isolation:

  - `cpaas.io/project`

  - `cpaas.io/cluster`

  - `cpaas.io/namespace`

  - `kubernetes.io/metadata.name`

  - Optional: `cpaas.io/creator`

- LabelSelector queries do not support logical OR operations

- Platform-level userbindings are not filtered

- Filtering is only applied to GET and LIST API operations

  - Optional: `cpaas.io/creator`

- LabelSelector queries do not support logical OR operations

- Platform-level userbindings are not filtered

- Filtering is only applied to GET and LIST API operations

Alauda Container Platform

Install Alauda Container Platform API Refiner

Alauda Container Platform API Refiner is a platform service that filters Kubernetes API response data. It provides filtering capabilities by project, cluster, and namespace, and supports field exclusion, inclusion, and desensitization in API responses.

TOC

Install via console

1. Navigate to **Administrator**

2. In the left navigation bar, click **Marketplace** > **Cluster Plugins**

3. Select the **global** cluster in the top navigation bar

4. Search for **Alauda Container Platform API Refiner** and click to view its details

5. Click **Install** to deploy the plugin

Install via YAML

1. Check available versions

Ensure the plugin has been published by checking for ModulePlugin and ModuleConfig resources, in `global` cluster :

```
# kubectl get moduleplugins apirefiner
NAME        AGE
apirefiner   4d20h

# kubectl get moduleconfigs -l cpaas.io/module-name=apirefiner
NAME                AGE
apirefiner-v4.0.4    4d21h
```

This indicates that the ModulePlugin `apirefiner` exists in the cluster and version `v4.0.4` is published.

2. Create a ModuleInfo

Create a ModuleInfo resource to install the plugin without any configuration parameters:

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  annotations:
    cpaas.io/display-name: apirefiner
    cpaas.io/module-name: '{"en": "Alauda Container Platform API Refine
r", "zh": "Alauda
      Container Platform API Refiner"}'
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: apirefiner
    cpaas.io/module-type: plugin
    cpaas.io/product: Platform-Center
  name: apirefiner-global
spec:
  version: v4.2.0-default.1.g8f0543e4
```

Field explanations:

- `name` : Temporary name for the cluster plugin. The platform will rename it after creation based on the content, in the format `<cluster-name>-<hash of content>` , e.g., `global-ee98c9991ea1464aaa8054bdacbab313` .

- `label cpaas.io/cluster-name` : API Refiner only can be installed in the `global` cluster, keep this filed as global.

- `label cpaas.io/module-name` : Plugin name, must match the ModulePlugin resource.

- `label cpaas.io/module-type` : Fixed field, must be `plugin` ; missing this field causes installation failure.

- `.spec.config` : If the corresponding ModuleConfig is empty, this field can be left empty.

- `.spec.version` : Specifies the plugin version to install, must match `.spec.version` in ModuleConfig.

Uninstallation Procedures

1. Follow steps 1-4 from the installation process to locate the plugin

2. Click **Uninstall** to remove the plugin

Default Configuration

Filtered Resources

The following resources are filtered by default:

Resource	API Version
namespaces	v1
projects	auth.alauda.io/v1
clustermodules	cluster.alauda.io/v1alpha2
clusters	clusterregistry.k8s.io/v1alpha1

Field Desensitization

By default, the following field is desensitized:

- `metadata.annotations.cpaas.io/creator`

Alauda Container Platform

About Alauda Container Platform Compliance Service

Compliance Service is a platform module designed to support STIG compliance scanning and MicroOS operating system scanning. It provides out-of-the-box compliance scanning capabilities with support for scheduled scanning and comprehensive reporting.

> **Note**
>
> Because Compliance Service releases on a different cadence from Alauda Container Platform, the Compliance Service documentation is now available as a separate documentation set at [Compliance Service ↗](link).

Alauda Container Platform

Users and Roles

User

Introduction

User Sources

User Management Rules

Role Assignment Views

User Lifecycle

Guides

Group

Introduction

Group Introduction

Group Types

Guides

Role

Introduction

Role Introduction

Guides

System Roles

Custom Permissions

IDP

Introduction

Guides

Troublesho

Overview

Supported Integration Methods

User Policy

Introduction

Overview

Configure Security Policy

Available Policies

Alauda Container Platform

User

Introduction

Introduction

User Sources

User Management Rules

Role Assignment Views

User Lifecycle

Guides

Manage User Roles

Assign Platform Roles (System Templates

Bind Kubernetes Roles (RoleBinding / Clu

Create User

Create User via Console

Create User via YAML

User Manag

Reset Local Us

Update User Ex

Activate User

Disable User

Add User to Lo

View User Role

Delete User

Batch Operation

Manage User Roles

Assign Platform Roles (System Templates

Bind Kubernetes Roles (RoleBinding / Clu

Create User

Create User via Console

Create User via YAML

User Manag

Reset Local Us

Alauda Container Platform

Introduction

The platform supports user authentication and login verification for all users.

TOC

User Sources

Local Users

- Administrator account created during platform deployment

- Accounts created through the platform interface

- Users added through local dex configuration file

Third-Party Users

LDAP Users

- Enterprise users synchronized from LDAP servers

- Accounts are imported through IDP (Identity Provider) integration

- Source is displayed as the IDP configuration name

- Integration is configured through IDP settings

OIDC Users

- Third-party platform users authenticated via OIDC protocol

- Source is displayed as the IDP configuration name

- Integration is configured through IDP settings

> **WARNING**
>
> For OIDC users added to a project before their first login:
>
> - Source is displayed as "-" until successful platform login
>
> - After successful login, source changes to the IDP configuration name

Other Third-Party Users

- Users authenticated through supported dex connectors (e.g., GitHub, Microsoft)

- For more information, refer to the dex official documentation ↗

User Management Rules

> **WARNING**
>
> Please note the following important rules:
>
> - Local usernames must be unique across all user types

- Third-party users (OIDC/LDAP) with matching usernames are automatically associated

- Associated users inherit permissions from existing accounts

- Users can log in through their respective sources

- Only one user record is displayed per username in the platform

- User source is determined by the most recent login method

Role Assignment Views

Every user detail page now contains two dedicated tabs for role visibility and maintenance:

- **Platform Roles** (read-only): Lists system-provided platform/project/namespace roles that are bound to the user. These roles cannot be edited or duplicated in the UI but can be unbound if necessary.

- **Kubernetes Roles**: Displays all RoleBinding/ClusterRoleBinding objects that reference the user (across clusters). Administrators can create or remove bindings here to grant native Kubernetes permissions.

Use the Platform Roles tab for default access templates, and use the Kubernetes Roles tab when you need fine-grained control through native roles.

User Lifecycle

The following table describes different user statuses on the platform:

Status	Description
Normal	User account is active and can log in to the platform

Status	Description
Disabled	User account is inactive and cannot log in. Contact platform administrator for activation. Possible reasons: - No login for 90+ consecutive days - Account expiration - Manual disable by administrator
Locked	Account is temporarily locked due to 5 failed login attempts within 24 hours. Details: - Lock duration: 20 minutes - Can be manually unlocked by administrator - Account becomes available after lock period
Invalid	LDAP-synchronized account that has been deleted from the LDAP server. Note: Invalid accounts cannot log in to the platform

Alauda Container Platform

Guides

Manage User Roles

Assign Platform Roles (System Templates

Bind Kubernetes Roles (RoleBinding / Clu

Create User

Create User via Console

Create User via YAML

User Manag

Reset Local Us

Update User E>

Activate User

Disable User

Add User to Lo

View User Role

Delete User

Batch Operatior

Alauda Container Platform

Manage User Roles

Platform administrators can manage roles for other users (not their own account) to grant or revoke permissions. After the RBAC refactor, role assignments are split into **Platform Roles** (system templates) and **Kubernetes Roles** (native RoleBinding objects).

TOC

Assign Platform Roles (System Templates)

Use this tab to bind or unbind the predefined platform/project/namespace roles that the product ships with.

Steps

1. In the left navigation bar, click **Users** > **User Management**.

2. Click the username of the target user.

3. Open the **Platform Roles** tab.

4. Click **Add Role**.

5. In the dialog:

- Select a role from the **Role Name** dropdown.

- Choose the scope (Cluster / Project / Namespace) if prompted.

- Click **Add**.

> **NOTE**
>
> Notes for platform roles:
>
> - Only the predefined system roles are available here; they cannot be edited or duplicated.
>
> - A role can only be bound once per scope. Already-bound roles are disabled in the dropdown.
>
> - The built-in Cluster Administrator role cannot be reassigned for the global cluster.

Remove Platform Roles

1. Stay on the **Platform Roles** tab.

2. Click **Remove** next to the role you want to unbind.

3. Confirm the removal.

Bind Kubernetes Roles (RoleBinding / ClusterRoleBinding)

Use this tab to grant fine-grained permissions through native Kubernetes roles that exist inside specific clusters.

Steps

1. On the user detail page, switch to the **Kubernetes Roles** tab.

2. Click **Add RoleBinding**.

3. Configure the binding:

- **Cluster**: Target cluster that hosts the role.

- **Binding Type**: `RoleBinding` (namespace scope) or `ClusterRoleBinding` .

- **Namespace**: Required when `RoleBinding` is selected.

- **Role Name**: Choose an existing `Role` or `ClusterRole` .

- **Subject**: Confirm the current user as the binding subject.

4. Click **Create**.

Remove Kubernetes RoleBindings

1. Remain on the **Kubernetes Roles** tab.

2. Locate the binding (filter by cluster, namespace, or role if needed).

3. Click **Remove** and confirm.

> **WARNING**
>
> Role management permissions:
>
> - Only platform administrators can manage other users' roles.
>
> - Users cannot modify roles or bindings for their own account.

Create User

Users with platform administrator roles can create local users and assign roles to them through the platform interface.

TOC

Create User via Console

1. In the left navigation bar, click **Users** > **User Management**

2. Click **Create User**

3. Configure the following parameters:

Parameter	Description
Password Type	Select a password generation method: **Random**: System generates a secure random password **Custom**: User manually enters a password
Password	Enter or generate a password based on the selected type. Password Requirements: - Length: 8-32 characters

Parameter	Description
	- Must contain letters and numbers - Must contain special characters (`~!@#$%^&*() -_=+?`) Password Field Features: - Click the eye icon to show/hide password - Click the copy icon to copy password
Mailbox	User's email address: - Must be unique - Can be used as login username - Associated with user's name
Validity Period	Set the user's account validity period: Options: - **Permanent**: No time limit - **Custom**: Set start and end times using the Time Range dropdown
Roles	Assign one or more roles to the user
Continue Creating	Toggle switch to control post-creation behavior: - **On**: Redirects to new user creation page - **Off**: Shows user details page

4. Click **Create**

> **NOTE**
>
> After successful user creation:
>
> - If "Continue Creating" is enabled, you'll be redirected to create another user
>
> - If disabled, you'll see the created user's details page

Create User via YAML

You can submit the following YAML in the `global` cluster to create a user.

```yaml
apiVersion: auth.alauda.io/v1
kind: User
metadata:
  labels:
    auth.cpaas.io/user.connector_id: ""                    # Connector ID
for external authentication (leave empty for local users)
    auth.cpaas.io/user.connector_type: ""                  # Connector t
ype for external authentication (leave empty for local users)
    auth.cpaas.io/user.email: c18c9911faaac4e1051a599b88c62af2   # MD5 has
h of the username (spec.email)
    auth.cpaas.io/user.state: active                       # User state;
must match spec.state
    auth.cpaas.io/user.username: ""                        # User displa
y name; must match spec.username
    auth.cpaas.io/user.valid: "true"                       # Whether the
user is valid; must match spec.valid
  name: c18c9911faaac4e1051a599b88c62af2                   # Name of the
User resource; MD5 hash of spec.email
spec:
  connector_name: ""                                       # Name of the
external authentication connector (leave empty for local users)
  connector_type: ""                                       # Type of the
external authentication connector (leave empty for local users)
  email: leizhuaaa                                         # User identi
fier; can be an email address or any unique string
  is_admin: false                                          # Whether the
user is an initial admin user; must be set to false
  state: active                                            # User accoun
t state: active or inactive
  username: ""                                             # Display nam
e for the user
  valid: true                                              # Whether the
user account is valid; should be set to true
```

Alauda Container Platform

User Management

The platform provides flexible user management capabilities, supporting both individual user management and batch operations for improved efficiency in specific scenarios (e.g., on-site or off-site teams).

> **WARNING**
>
> Important Restrictions:
>
> - System-generated accounts cannot be managed (platform administrator role, local source)
>
> - Currently logged-in users cannot manage their own accounts
>
> - For personal account modifications (display name, password), please use the personal information page

TOC

Reset Local User Password

Users with platform management permissions can reset passwords for other local users.

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click the icon next to the target user's record

3. Click **Reset Password**

4. In the dialog box, select a password type:

   - **Random**: System generates a secure random password

   - **Custom**: Enter a new password manually

> **NOTE**
>
> Password Requirements:
>
> - Length: 8-32 characters
>
> - Must contain letters and numbers
>
> - Must contain special characters (`~!@#$%^&*()-_=+?`)
>
> Password Field Features:
>
> - Click eye icon to show/hide password
>
> - Click copy icon to copy password

5. Click **Reset**

Update User Expiry Date

You can update expiry dates for users in **normal**, **disabled**, or **locked** status. Users exceeding their expiry date will be automatically disabled.

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click **Update Expiry Date** next to the target user

3. In the dialog box, select an expiry date option:

   - **Permanent**: No time limit

   - **Custom**: Set start and end times using the Time Range dropdown

4. Click **Update**

Activate User

You can activate users in **disabled** or **locked** status.

> **NOTE**
>
> Activation Behavior:
>
> - If user is within expiry date: expiry date remains unchanged
>
> - If user has expired: expiry date becomes **Permanent**

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click **Activate** next to the target user

3. Click **Activate** in the confirmation dialog

4. User status will change to **normal**

Disable User

You can disable users in **normal** or **locked** status within their expiry date. Disabled users cannot log in but can be reactivated.

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click the icon next to the target user

3. Click **Disable** and confirm

Add User to Local User Group

You can add users with **Source** as **Local** or `LDAP` to one or more local user groups.

> **WARNING**
>
> Group Role Behavior:
>
> - Users automatically inherit roles from their groups
>
> - Group roles are only visible on the group's details page (**Platform Roles** tab)
>
> - Individual user role lists only show directly assigned roles

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click the icon next to the target user

3. Click **Add to User Group**

4. Select one or more local user groups

5. Click **Add**

View User Roles

After the RBAC refactor, each user detail page shows two tabs:

- **Platform Roles**: Displays the system-provided roles bound to the user. You can add or remove bindings but cannot edit role definitions.

- **Kubernetes Roles**: Lists every RoleBinding/ClusterRoleBinding that references the user across clusters. You can create or remove bindings directly on this tab.

See Manage User Roles for detailed procedures.

Delete User

Platform administrators can delete any user except the currently logged-in account, including:

- IDP-configured users

- Users with source -

- Local users

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Click the icon next to the target user

3. Click **Delete**

4. Click **Confirm**

Batch Operations

You can perform batch operations for:

- Updating validity periods

- Activating users

- Disabling users

- Deleting users

Steps

1. In the left navigation bar, click **Users** > **User Management**

2. Select one or more users using checkboxes

3. Click **Batch Operations** and select an action:

   - **Update Validity**

   - **Activate**

   - **Deactivate**

   - **Delete**

   > **NOTE**
   >
   > Batch Operation Details:
   >
   > - **Update Validity**: Set permanent or custom time range
   >
   > - **Activate**: Confirm activation in dialog
   >
   > - **Deactivate**: Confirm deactivation in dialog
   >
   > - **Delete**: Enter current account password and confirm

Alauda Container Platform

Group

Introduction

Introduction

Group Introduction

Group Types

Guides

Manage User Group Roles

Add Role to Group

Remove Role from Group

Create Local User Group

Create User Group

Manage User Groups

Manage Lo

Prerequisites

Import Member

Remove Memb

Alauda Container Platform

Introduction

TOC

Group Introduction

The platform supports user management through user groups. By managing group roles, you can efficiently:

- Grant platform operation permissions to multiple users simultaneously

- Revoke permissions from multiple users at once

- Implement batch role-based access control

For example, when personnel changes occur within an enterprise and you need to grant new project or namespace operation permissions to multiple users, you can:

1. Create a user group

2. Import relevant users as group members

3. Configure project and namespace roles for the group

4. Apply unified permissions to all group members

Group Types

The platform supports two types of groups:

Local User Group

- Created directly on the platform

- Source is displayed as **Local**

- Can be updated or deleted

- Supports:

  - Adding or removing users from any source

  - Adding or removing roles

IDP-Synchronized User Group

- Synchronized from connected IDP (LDAP, Azure AD)

- Source is displayed as the connected *IDP* name

- Cannot be updated or deleted

- Supports:

  - Adding or removing roles

  - Cannot manage group members (add or remove)

Alauda Container Platform

Guides

Manage User Group Roles

Add Role to Group

Remove Role from Group

Create Local User Group

Create User Group

Manage User Groups

Manage Loc

Prerequisites

Import Member

Remove Memb

Alauda Container Platform

Manage User Group Roles

Users with platform management permissions can manage roles for both local user groups and IDP-synchronized user groups.

TOC

Add Role to Group

Steps

1. In the left navigation bar, click **Users** > **User Group Management**

2. Click the name of the target user group

3. On the **Configure Role** tab, click **Add Role**

4. Click to add a role

> **NOTE**
>
> Role Assignment Rules:

- You can add multiple roles to a group

- Each role can only be added once to the same group

5. Select the role name from the dropdown

6. Choose the role's permission scope (cluster, project, or namespace)

7. Click **Add**

Remove Role from Group

> **WARNING**
>
> When you remove a role from a group:
>
> - All permissions granted by that role to group members will be revoked
>
> - This action cannot be undone

Steps

1. In the left navigation bar, click **Users** > **User Group Management**

2. Click the name of the target user group

3. On the **Configure Role** tab, click **Remove** next to the role

4. Click **Confirm** to remove the role

Alauda Container Platform

Create Local User Group

Local user groups allow you to implement role-based access control for multiple users from any source.

TOC

Create User Group

Steps

1. In the left sidebar, click **Users** > **User Group Management**

2. Click **Create User Group**

3. Enter the following information:

   - **Name**: The name of the user group

   - **Description**: A description of the group's purpose

4. Click **Create**

Manage User Groups

You can manage user groups by clicking the icon on the list page or clicking **Operations** in the upper right corner on the details page.

Operation	Description
Update User Group	Update group information based on the group source: - For groups with **Source** as `Local` : Can update both name and description - For groups with **Source** as `IDP name` : Can only update description
Delete Local User Group	Delete user groups with **Source** as `Local`

> **WARNING**
>
> When you delete a group:
>
> - All group members will be removed
>
> - All roles assigned to the group will be removed
>
> - This action cannot be undone

Alauda Container Platform

Manage Local User Group Membership

Only users with Platform Management permissions can manage local user group memberships.

TOC

Prerequisites

> **WARNING**
>
> Before managing group memberships, please note the following limitations:
>
> - Only users with Platform Management permissions can manage groups and their members
>
> - System accounts and currently logged-in accounts cannot be managed (imported to or removed from groups)
>
> - Each local user group can have a maximum of 5000 members
>
> - When a group reaches the 5000-member limit, no further imports are allowed

Import Members

You can import users from the platform into local user groups for unified permission management.

> **TIP**
>
> Users imported into a group will automatically inherit all operational permissions assigned to that group.

Steps

1. In the left navigation bar, click **Users** > **User Group Management**

2. Click the name of the local user group where you want to add members

3. On the **Group Member Management** tab, click **Import Member**

4. Select one or more users from the platform by checking the boxes next to their usernames/display names

5. Click **Import**

> **NOTE**
>
> - You can only select users who are not currently members of the group
>
> - Use the **Import All** button to import all users in the list at once

Remove Members

When you remove a user from a group, all operational permissions granted to that user through the group will be automatically revoked.

Steps

1. In the left navigation bar, click **Users** > **User Group Management**

2. Click the name of the local user group where you want to remove members

3. On the **Group Member Management** tab, you can remove members in two ways:

   - Click **Remove** next to the member's name and confirm

   - Select one or more members using checkboxes, then click **Batch Remove** and confirm

Alauda Container Platform

Role

Introduction

Introduction

Role Introduction

System Roles

Custom Permissions

Guides

Create Kubernetes Roles

Prerequisites

Create a Role or ClusterRole

Edit Role YAML (Optional)

Create RoleBindings

Verify

Manage Roles

View Platform Roles (Read-Only)

Grant a Platform Role to Users via Consol

Grant a Platform Role to Users via YAML

View and Update a Kubernetes Role via Y

Delete a Kubernetes Role

Manage RoleBindings

Best Practices

How to Crea

1. Overview

2. Core Concep

3. Technical Ch

4. Supported C

5. Configuration

6. Copy & Trim

Alauda Container Platform

Introduction

TOC

Role Introduction

The platform's user role management is implemented on top of Kubernetes RBAC (Role-Based Access Control). After ACP 4.2, the model is split into two complementary layers:

- **Platform Roles**: System-provided templates that guarantee core product scenarios. They are read-only in the UI; you can only bind or unbind them for users and groups.
- **Kubernetes Roles**: Native Kubernetes `Role` and `ClusterRole` objects that you can create per cluster to satisfy bespoke permission requirements. These roles are managed from the **Kubernetes Roles** page and bound through RoleBinding/ClusterRoleBinding.

Both layers ultimately translate into Kubernetes permissions. Assigning or removing a role immediately grants or revokes the associated operations (create, view, update, delete, etc.) on targeted resources.

System Roles

To meet common permission configuration scenarios, the platform provides the following default system roles. These roles enable flexible access control for platform resources and efficient permission management for users.

Role Name	Description	Role Level
Platform Administrator	Has full access to all business and resources on the platform	Platform
Platform Auditors	Can view all platform resources and operation records, but has no other permissions	Platform
Cluster Administrator (Alpha)	Manages and maintains cluster resources with full access to all cluster-level resources	Cluster
Project Administrator	Manages namespace administrators and namespace quotas	Project
namespace-admin-system	Manages namespace members and role assignments	Namespace
Developers	Develops, deploys, and maintains custom applications within namespaces	Namespace

Custom Permissions

The legacy "checkbox-based" custom role creation experience has been removed. To implement new authorization scenarios you must:

1. Use the **Kubernetes Roles** page to create native roles (Role/ClusterRole) in the desired cluster, or

2. Request role templates from the owning plug-in team and bind them through RoleBinding/ClusterRoleBinding.

> **WARNING**

Deleting or editing a native role affects every RoleBinding/ClusterRoleBinding that references it.

Review existing bindings and validate changes in a staging environment when possible.

Alauda Container Platform

Guides

Create Kubernetes Roles

Prerequisites

Create a Role or ClusterRole

Edit Role YAML (Optional)

Create RoleBindings

Verify

Manage Roles

View Platform Roles (Read-Only)

Grant a Platform Role to Users via Consol

Grant a Platform Role to Users via YAML

View and Update a Kubernetes Role via Y,

Delete a Kubernetes Role

Manage RoleBindings

Best Practices

How to Crea

1. Overview

2. Core Concep

3. Technical Ch

4. Supported C

5. Configuratior

6. Copy & Trim

Alauda Container Platform

Create Kubernetes Roles

Starting from ACP 4.2, custom permissions are delivered through native Kubernetes roles. Use the **Kubernetes Roles** page (located under **Users > Platform Roles > Kubernetes Roles**) to create or manage `Role` and `ClusterRole` objects in the currently selected cluster.

TOC

Prerequisites

- You are assigned a platform role that grants access to the Kubernetes Roles feature.

- You have selected the target cluster in the global cluster switcher.

- The cluster already contains any namespaces that your role will scope to.

Create a Role or ClusterRole

1. In the left navigation bar, click **Users** > **Platform Roles** > **Kubernetes Roles**.

2. Click **Create Role**.

3. In the drawer:

- Enter the **Name** (must satisfy Kubernetes naming rules).

- Choose the **Type** (`Role` or `ClusterRole`).

- If you selected `Role` , pick the **Namespace** that scopes the permissions.

4. Configure rules by adding one or more entries with:

- **API Groups**

- **Resources**

- **Resource Names** (optional)

- **Verbs** (`get` , `list` , `watch` , `create` , `update` , `patch` , `delete`)

5. Click **Create**.

The role is created directly inside the cluster and becomes available for RoleBinding operations immediately.

Edit Role YAML (Optional)

1. Open the **Kubernetes Roles** tab and click the role name.

2. On the **YAML** tab, click **Edit**.

3. Update fields such as labels, annotations, or rules.

4. Click **Save** to apply the changes.

Create RoleBindings

To grant the newly created role to users or groups:

1. While viewing a role, switch to the **RoleBindings** tab.

2. Click **Create RoleBindings**.

3. Provide:

- **Name**

- **Binding Type** (`RoleBinding` or `ClusterRoleBinding` — only `RoleBinding` is available when the source is a namespace-scoped role)

- **Namespace** (for `RoleBinding`)

- **Subjects** (User, Group, or ServiceAccount with the corresponding name)

4. Click **Create**.

Alternatively, open the **Users** or **User Groups** page, switch to the **Kubernetes Roles** tab, and create bindings directly from the user perspective.

Verify

Use one of the following methods to confirm the role exists:

```
kubectl get role <role-name> -n <namespace>
kubectl get clusterrole <clusterrole-name>
```

Or refresh the **Kubernetes Roles** list and use the built-in search (by name or label) to locate the role.

Alauda Container Platform

Manage Roles

TOC

View Platform Roles (Read-Only)

Platform roles remain the canonical templates for core functionality.

1. In the left navigation bar, click **Users** > **Platform Roles**.

2. Use the list filters to locate a role. The **Role Type** column now shows `Platform`, `Project`, `Namespace`, or `Cluster`.

3. Click the role name to open the detail page.

4. Switch to the **YAML** tab to inspect the exact definition. Use **Download YAML** if you need to archive the spec.

Grant a Platform Role to Users via Console

1. In the left navigation bar, click **Users** > **Platform Roles**.

2. Click the role name to open the detail page.

3. Switch to the **Members** tab.

4. Click **Import Members**.

5. You can select users from the platform and import them to the role as members.

Grant a Platform Role to Users via YAML

You can submit the following YAML in the `global` cluster to grant a specific platform role to a user.

apiVersion: auth.alauda.io/v1 kind: UserBinding metadata: annotations: auth.cpaas.io/role.display-name: Platform Admin # Display name of the role to be assigned auth.cpaas.io/user.email: bxliu@alauda.io ↗ # Username of the user to grant the role to labels: auth.cpaas.io/role.display-name: "" # Display name of the role to be assigned auth.cpaas.io/role.level: platform # Scope of the role: platform, project, namespace, or cluster auth.cpaas.io/role.name: acp-platform-admin # Name of the role to be assigned auth.cpaas.io/user.email: 569526aac97a17ce8c1c185d7544aae4 # MD5 hash of the Username cpaas.io/cluster: "" # Name of the cluster; required when role level is namespace or cluster, leave empty for platform or project cpaas.io/namespace: "" # Name of the namespace; required when role level is namespace, leave empty for platform, project, or cluster cpaas.io/project: "" # Name of the project; required when role level is project or namespace, leave empty for platform or cluster name: dc30204c17c7fe8b15383f4ed7798c88 # Name of the UserBinding resource; can be customized

View and Update a Kubernetes Role via YAML

1. Navigate to **Users** > **Platform Roles** > **Kubernetes Roles**.

2. Search by name or label.

3. Click the role name, then open the **YAML** tab.

4. Click **Edit**, modify the manifest (labels, annotations, or `rules`), and click **Save**.

5. Review the **RoleBindings** tab to ensure existing bindings still meet your expectations.

Delete a Kubernetes Role

1. On the **Kubernetes Roles** list, click the overflow menu (…) next to the role.

2. Select **Delete Role**.

3. Confirm the role name to proceed.

Deleting a role removes it from the cluster. You must also clean up any RoleBindings that referenced the role. The UI will show a warning if bindings are still present.

Manage RoleBindings

From the Role Perspective

1. Open a role (Role or ClusterRole) from the **Kubernetes Roles** tab.

2. Go to the **RoleBindings** tab.

3. Use the search bar (supports name and label filters) to locate existing bindings.

4. Actions:

- **Create RoleBindings**: Launches the creation wizard.

- **Update Role**: Opens the YAML editor for the role itself.

- **Delete Binding**: Removes the RoleBinding/ClusterRoleBinding after confirmation.

From Users or User Groups

1. Open **Users** (or **User Groups**) and select the desired entry.

2. Switch to the **Kubernetes Roles** tab.

3. Review all RoleBindings associated with the user/group across clusters.

4. Click **Add RoleBinding**, choose:

- Cluster

- Binding type (RoleBinding/ClusterRoleBinding)

- Role/ClusterRole

- Namespace (for RoleBinding)

- Subject details

5. Save the binding.

This workflow complements the existing **Platform Roles** tab, which is still used to attach system roles to users.

Best Practices

- Use staging clusters to validate YAML changes before applying them to production.

- Keep role definitions under version control (for example, export them into Git) so that changes remain auditable.

- When in doubt about required permissions, start from a system role`s YAML, copy it locally, and adapt it as a Kubernetes role through the new UI.

Alauda Container Platform

How to Create Custom Platform Role

TOC

1. Overview

This document explains the core concepts of custom roles and how to configure a RoleTemplate.

2. Core Concepts

- **RoleTemplate**: Custom role template. It defines role semantics and permission sets, and is converted by the controller into ClusterRoles.

- **FunctionResource**: An abstraction of K8s resources used by product features; referenced by `functionResourceRef` in RoleTemplate.

- **ClusterRole**: RBAC rule set; can be aggregated into system roles via labels.

- **UserBinding**: Binding between users and roles/scopes; the controller generates RoleBinding/ClusterRoleBinding based on UserBinding for final authorization.

> **NOTE**
>
> - The value of `functionResourceRef` comes from FunctionResource `metadata.name`.
>
> - Display names are typically in `metadata.annotations`, which you can use to map to UI modules.

3. Technical Changes

Starting from ACP v4.3, feature permissions are gradually migrated from FunctionResource to native K8s ClusterRole management. RoleTemplate module permissions will be aggregated into system roles via ClusterRole labels. FunctionResource-based management will be retired in v4.5.

4. Supported Configuration Methods

- **FunctionResource method**: Select permissions by product modules and control verbs at fine granularity.

- **ClusterRole aggregation method (aggregationRules)**: Centralized aggregation via ClusterRole labels; recommended for v4.2+.

- **customRules**: Configure permissions using native K8s RBAC syntax; same rule format as ClusterRole `rules`.

5. Configuration

5.1 RoleTemplate YAML

FunctionResource method (supported until v4.5)

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-funcrole
  annotations:
    cpaas.io/display-name: Example Function Role
    cpaas.io/description: FunctionResource example
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  rules:
    - functionResourceRef: acp-app
      verbs: [get, list, watch]
```

FunctionResource example:

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: FunctionResource
metadata:
  name: acp-app
  annotations:
    cpaas.io/functionresource.module.display-name: Container Platform
    cpaas.io/functionresource.function.display-name: Application
  labels:
    auth.cpaas.io/functionresource.module: acp
    auth.cpaas.io/functionresource.function: app
    auth.cpaas.io/product: console-acp
spec:
  rules:
    - apiGroup: app.k8s.io
      resources: ["*"]
      bindScope: namespace
      bindCluster: unlimit
      bindNamespacePart: common
```

ClusterRole aggregation method (v4.2+)

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-aggrole
  annotations:
    cpaas.io/display-name: Example Aggregation Role
    cpaas.io/description: ClusterRole aggregation example
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-namespace-developer: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
  rules: []
```

ClusterRole with aggregation labels:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cpaas:demo:business-ns:view
  labels:
    rbac.cpaas.io/aggregate-to-namespace-developer: "true"
    rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
rules:
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get", "list", "watch"]
```

customRules method

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-customrules
  annotations:
    cpaas.io/display-name: Example Custom Rules Role
    cpaas.io/description: customRules example
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  customRules:
    - apiGroups: [""]
      resources: ["configmaps"]
      verbs: ["get", "list", "watch"]
```

RoleTemplate field reference (with ranges):

Field	Description
`metadata.name`	Role name
`metadata.labels.auth.cpaas.io/roletemplate.level`	Role level

Field	Description
`metadata.annotations.cpaas.io/(display-name/description)`	Display name and description
`spec.rules[].functionResourceRef`	FunctionResource reference
`spec.rules[].verbs`	Verbs
`spec.aggregationRules[].scope`	Aggregation scope
`spec.aggregationRules[].clusterRoleSelectors`	Label selectors
`spec.aggregationRules[].clusterRoleSelectors.matchLabels`	Label key/value
`spec.aggregationRules[].clusterRoleSelectors.matchExpressions`	Expressions
`spec.customRules[].apiGroups`	API groups
`spec.customRules[].resources`	Resources
`spec.customRules[].verbs`	Verbs
`spec.customRules[].resourceNames`	Resource names

Field	Description
`spec.customRules[].nonResourceURLs`	Non-resource URLs

> **NOTE**
>
> `spec.rules` and `spec.aggregationRules` are mutually exclusive.

System labels for matchLabels

System role aggregation labels:

- `rbac.cpaas.io/aggregate-to-platform-admin: "true"`
- `rbac.cpaas.io/aggregate-to-platform-auditor: "true"`
- `rbac.cpaas.io/aggregate-to-cluster-admin: "true"`
- `rbac.cpaas.io/aggregate-to-project-admin: "true"`
- `rbac.cpaas.io/aggregate-to-namespace-admin: "true"`
- `rbac.cpaas.io/aggregate-to-namespace-developer: "true"`
- `rbac.cpaas.io/aggregate-to-basic-user: "true"`

Scope aggregation labels:

- `rbac.cpaas.io/aggregate-to-scope-cluster: "true"`
- `rbac.cpaas.io/aggregate-to-scope-project-ns: "true"`
- `rbac.cpaas.io/aggregate-to-scope-business-ns: "true"`
- `rbac.cpaas.io/aggregate-to-scope-system-ns: "true"`
- `rbac.cpaas.io/aggregate-to-scope-kube-public: "true"`

> **NOTE**
>
> Custom labels are also supported for aggregation.

FunctionResource field reference:

Field	Description
`metadata.name`	FunctionResource identifier (for `functionResourceRef`)
`metadata.annotations`	Display names (for UI mapping)
`metadata.labels`	Module/function metadata (for indexing)
`spec.rules[].apiGroup`	API group (`""` for core)
`spec.rules[].resources`	Resource types
`spec.rules[].bindScope`	Scope (`cluster / namespace`)
`spec.rules[].bindCluster`	Cluster scope (`global / unlimit / business`)
`spec.rules[].bindNamespacePart`	Namespace part (`common / system / kube-public / project_ns / cluster / ""`)

> **NOTE**
>
> `acp-namespace-resource-manage` is **not allowed** in custom roles.

5.2 Get ClusterRoles generated from a RoleTemplate

Query ClusterRoles generated by a RoleTemplate using labels:

```
kubectl get clusterrole -l auth.cpaas.io/role.relative=<roletemplate-name>
```

6. Copy & Trim (Examples)

For custom roles, it is recommended to copy a built-in system template and trim it to avoid missing module permissions.

6.1 Trim a system RoleTemplate YAML

Steps:

- Keep the required FunctionResources.

- Reduce verbs to read-only (get/list/watch).

- Remove unused modules.

Role example: developer auditor without secret permissions

> **NOTE**
>
> - In custom roles, do not configure Namespace Resource Management (FunctionResource: `acp-namespace-resource-manage`).
>
> - Remove User Secret Dictionary (FunctionResource: `acp-user-secret`).
>
> - Set all verbs to get/list/watch.

Example YAML:
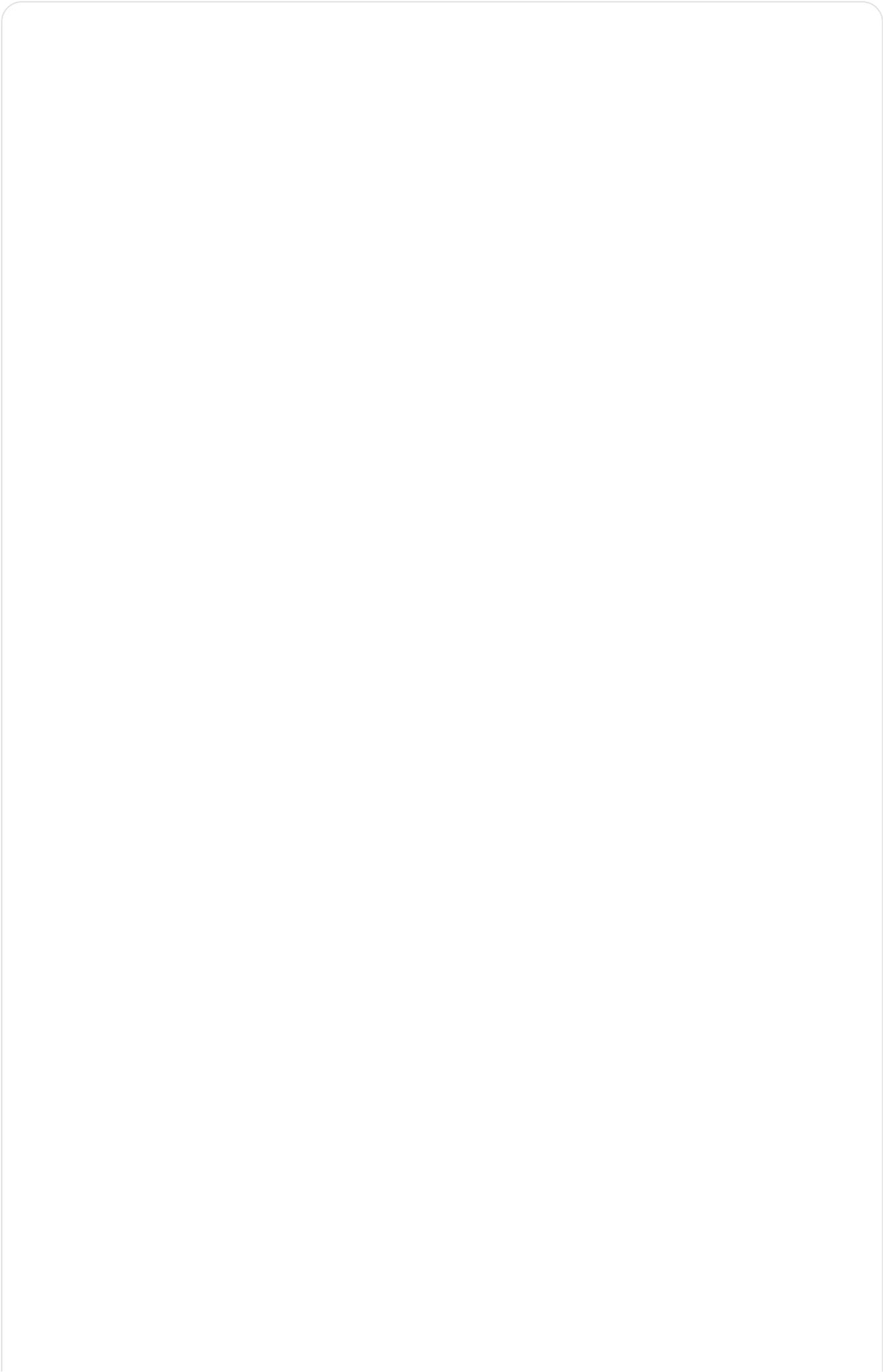
```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  annotations:
    cpaas.io/description: Responsible for development, deployment, and maintenance within the namespace.
    cpaas.io/display-name: Developer Copy
  labels:
    auth.cpaas.io/roletemplate.level: namespace
  name: namespace-developer-system-copy
spec:
  rules:
    - functionResourceRef: views-acp-userview
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-clusters
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-nodesmanage
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-domains
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: acp-subnet
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: acp-networkpolicies
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-storageclasses
```

```
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-persistentvolumes
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: acp-virtualmachineimagetemplates
      verbs:
        - get
        - list
        - watch
```

6.2 Trim namespace-developer-system into an auditor role

Approach:

- Keep the scope structure (cluster / project-ns / business-ns / system-ns / kube-public).

- Replace aggregation labels with auditor labels (requires labeled ClusterRoles).

Example (excerpt):

```
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: namespace-auditor
  annotations:
    cpaas.io/display-name: Namespace Auditor
    cpaas.io/description: Read-only audit
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-namespace-auditor: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
  rules: []
```

NOTE

`aggregate-to-namespace-auditor` must match labels on existing ClusterRoles. If not, create a read-only ClusterRole and add the label first.

Alauda Container Platform

IDP

Introduction

Introduction

Overview

Supported Integration Methods

Guides

LDAP Management

LDAP Overview

Supported LDAP Types

LDAP Terminology

Add LDAP

LDAP Configuration Examples

Synchronize LDAP Users

Relevant Operations

OIDC Management

Overview of OIDC

Adding OIDC

Adding OIDC via YAML

Relevant Operations

Troubleshooting

Delete User

Problem Description

Solution

Alauda Container Platform

Introduction

TOC

Overview

The platform integrates with Dex identity authentication service, enabling you to use Dex's pre-implemented connectors for platform account authentication through IDP configuration. For more information, refer to the Dex official documentation ↗ .

Supported Integration Methods

LDAP Integration

If your enterprise uses **LDAP** (Lightweight Directory Access Protocol) for user management, you can configure LDAP on the platform to connect with your enterprise's LDAP server.

LDAP Integration Benefits:

- Enables communication between platform and LDAP server

- Allows enterprise users to log in with LDAP credentials

- Automatically synchronizes enterprise user accounts to the platform

OIDC Integration

The platform supports integration with IDP services using the OpenID Connect (OIDC) protocol for third-party user authentication.

OIDC Integration Benefits:

- Enables users to log in with third-party accounts

- Supports enterprise IDP services

- Provides secure authentication through OIDC protocol

> **NOTE**
>
> For authentication using other connectors not mentioned above, please contact technical support.

Alauda Container Platform

Guides

LDAP Management

LDAP Overview

Supported LDAP Types

LDAP Terminology

Add LDAP

LDAP Configuration Examples

Synchronize LDAP Users

Relevant Operations

OIDC Management

Overview of OIDC

Adding OIDC

Adding OIDC via YAML

Relevant Operations

Alauda Container Platform

LDAP Management

Platform administrators can add, update, and delete LDAP services on the platform.

TOC

LDAP Overview

LDAP (Lightweight Directory Access Protocol) is a mature, flexible, and well-supported standard mechanism for interacting with directory servers. It organizes data in a hierarchical tree structure to store enterprise user and organization information, primarily used for implementing single sign-on (SSO).

> **NOTE**
>
> LDAP Key Features:
>
> - Enables communication between clients and LDAP servers
>
> - Supports data storage, retrieval, and search operations
>
> - Provides client authentication capabilities
>
> - Facilitates integration with other systems

For more information, refer to the LDAP official documentation ↗ .

Supported LDAP Types

OpenLDAP

OpenLDAP is an open-source implementation of LDAP. If your organization uses open-source LDAP for user authentication, you can configure the platform to communicate with the LDAP service by adding LDAP and configuring relevant parameters.

> **NOTE**
>
> OpenLDAP Integration:
>
> - Enables platform authentication for LDAP users
>
> - Supports standard LDAP protocols

- Provides flexible user management

For more information about OpenLDAP, refer to the [OpenLDAP official documentation ↗](#).

Active Directory

Active Directory is Microsoft's LDAP-based software for providing directory storage services in Windows systems. If your organization uses Microsoft Active Directory for user management, you can configure the platform to communicate with the Active Directory service.

> **NOTE**
>
> Active Directory Integration:
>
> - Enables platform authentication for AD users
>
> - Supports Windows domain integration
>
> - Provides enterprise-level user management

LDAP Terminology

OpenLDAP Common Terms

Term	Description	Example
dc (Domain Component)	Domain component	`dc=example,dc=com`
ou (Organizational Unit)	Organizational unit	`ou=People,dc=example,dc=com`
cn (Common Name)	Common name	`cn=admin,dc=example,dc=com`
uid (User ID)	User ID	`uid=example`

Term	Description	Example
objectClass (Object Class)	Object class	`objectClass=inetOrgPerson`
mail (Mail)	Mail	`mail=example@126.com`
givenName (Given Name)	Given name	`givenName=xq`
sn (Surname)	Surname	`sn=ren`
objectClass: groupOfNames	User group	`objectClass: groupOfNames`
member (Member)	Group member attribute	`member=cn=admin,dc=example,dc=com`
memberOf	User group membership attribute	`memberOf=cn=users,dc=example,dc=com`

Active Directory Common Terms

Term	Description	Example
dc (Domain Component)	Domain component	`dc=example,dc=com`
ou (Organizational Unit)	Organizational unit	`ou=People,dc=example,dc=com`
cn (Common Name)	Common name	`cn=admin,dc=example,dc=com`
sAMAccountName/userPrincipalName	User identifier	`userPrincipalName=example` or `sAMAccountName=example`
objectClass: user	AD user object class	`objectClass=user`

Term	Description	Example
mail (Mail)	Mail	`mail=example@126.com`
displayName	Display name	`displayName=example`
givenName (Given Name)	Given name	`givenName=xq`
sn (Surname)	Surname	`sn=ren`
objectClass: group	User group	`objectClass: group`
member (Member)	Group member attribute	`member=CN=Admin,DC=example,DC=`
memberOf	User group membership attribute	`memberOf=CN=Users,DC=example,`

Add LDAP

> **TIP**
>
> After successful LDAP integration:
>
> - Users can log in to the platform using their enterprise accounts
>
> - Multiple additions of the same LDAP will overwrite previously synchronized users

Prerequisites

Before adding LDAP, prepare the following information:

- LDAP server address

- Administrator username

- Administrator password

- Other required configuration details

Steps

1. In the left navigation bar, click **Users** > **IDPs**

2. Click **Add LDAP**

3. Configure the following parameters:

Basic Information

Parameter	Description
Server Address	LDAP server access address (e.g., `192.168.156.141:31758`)
Username	LDAP administrator DN (e.g., `cn=admin,dc=example,dc=com`)
Password	LDAP administrator account password
Login Box Username Prompt	Prompt message for username input (e.g., "Please enter your username")

Search Settings

> **NOTE**
>
> Search Settings Purpose:
>
> - Matches LDAP user entries based on specified conditions
>
> - Extracts key user and group attributes
>
> - Maps LDAP attributes to platform user attributes

Parameter	Description
Object Type	ObjectClass for users: - OpenLDAP: `inetOrgPerson`

Parameter	Description
	- Active Directory: `organizationalPerson` - Groups: `posixGroup`
Login Field	Attribute used as login username: - OpenLDAP: `mail` (email address) - Active Directory: `userPrincipalName`
Filter Conditions	LDAP filter conditions for user/group filtering Example: `(&(cn=John*)(givenName=*xq*))`
Search Starting Point	Base DN for user/group search (e.g., `dc=example,dc=org`)
Search Scope	Search scope: - `sub` : entire directory subtree - `one` : one level below starting point
Login Attribute	Unique user identifier: - OpenLDAP: `uid` - Active Directory: `distinguishedName`
Name Attribute	Object name attribute (default: `cn`)
Email Attribute	Email attribute: - OpenLDAP: `mail` - Active Directory: `userPrincipalName`
Group Member Attribute	Group member identifier (default: `uid`)
Group Attribute	User group relationship attribute (default: `memberuid`)

4. In the **IDP Service Configuration Validation** section:

- Enter a valid LDAP account username and password

- The username must match the **Login Field** setting

- Click to verify the configuration

5. (Optional) Configure **LDAP Auto-Sync Policy**:

- Enable **Auto-Sync Users** switch

- Set synchronization rules

- Use online tool ↗ to verify CRON expressions

6. Click **Add**

> **NOTE**
>
> After adding LDAP:
>
> - Users can log in before synchronization
>
> - User information syncs automatically on first login
>
> - Auto-sync occurs based on configured rules

LDAP Configuration Examples

LDAP Connector Configuration

The following example shows how to configure an LDAP connector:

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: ldap          # Connector ID
name: ldap        # Connector display name
type: ldap        # Connector type is LDAP
metadata:
  name: ldap
  namespace: cpaas-system
spec:
  config:
    # LDAP server address and port
    host: ldap.example.com:636
    # DN and password for the service account used by the connector.
    # This DN is used to search for users and groups.
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    # Service account password, required when creating a connector.
    bindPW: password

    # Login account prompt. For example, username
    usernamePrompt: SSO Username

    # User search configuration
    userSearch:
      # Start searching from the base DN
      baseDN: cn=users,dc=example,dc=com
      # LDAP query statement, used to search for users.
      # For example: "(&(objectClass=person)(uid=<username>))"
      filter: (&(objectClass=organizationalPerson))

      # The following fields are direct mappings of user entry attribute
s.
      # User ID attribute
      idAttr: uid
      # Required. Attribute to map to email
      emailAttr: mail
      # Required. Attribute to map to username
      nameAttr: cn
      # Login username attribute
      # Filter condition will be converted to "(<attr>=<username>)", such
 as (uid=example).
      username: uid

      # Extended attributes
```

```
    # phoneAttr: phone


  # Group search configuration
  groupSearch:
    # Start searching from the base DN
    baseDN: cn=groups,dc=freeipa,dc=example,dc=com
    # Group filter condition
    # "(&(objectClass=group)(member=<user uid>))".
    filter: "(objectClass=group)"
    # User group matching field
    # Group attribute
    groupAttr: member
    # User group member attribute
    userAttr: uid
    # 组显示名称
    nameAttr: cn
```

User Filter Examples

```
# 1. Basic filter: Find all users
(&(objectClass=person))


# 2. Multiple conditions combination: Find users in a specific department
(&(objectClass=person)(departmentNumber=1000))


# 3. Find enabled users (Active Directory)
(&(objectClass=user)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))


# 4. Find users with a specific email domain
(&(objectClass=person)(mail=*@example.com))


# 5. Find members of specific group
(&(objectClass=person)(memberOf=cn=developers,ou=groups,dc=example,dc=co
m))


# 6. Find recently logged in users (Active Directory)
(&(objectClass=user)(lastLogon>=20240101000000.0Z))


# 7. Exclude system accounts
(&(objectClass=person)(!(uid=admin))(!(uid=system)))


# 8. Find users with a specific attribute
(&(objectClass=person)(mobile=*))


# 9. Find users in multiple departments
(&(objectClass=person)(|(ou=IT)(ou=HR)(ou=Finance)))


# 10. Complex condition combination example
(&
  (objectClass=person)
  (|(department=IT)(department=Engineering))
  (!(title=Intern))
  (manager=cn=John Doe,ou=People,dc=example,dc=com)
)
```

Group Search Configuration Examples

```
# 1. Basic filter: Find all groups
(objectClass=groupOfNames)

# 2. Find groups with a specific prefix
(&(objectClass=groupOfNames)(cn=dev-*))

# 3. Find non-empty groups
(&(objectClass=groupOfNames)(member=*))

# 4. Find groups with a specific member
(&(objectClass=groupOfNames)(member=uid=john,ou=People,dc=example,dc=co
m))

# 5. Find nested groups (Active Directory)
(&(objectClass=group)(|(groupType=-2147483646)(groupType=-2147483644)))

# 6. Find groups with a specific description
(&(objectClass=groupOfNames)(description=*admin*))

# 7. Exclude system groups
(&(objectClass=groupOfNames)(!(cn=system*)))

# 8. Find groups with specific members
(&(objectClass=groupOfNames)(|(cn=admins)(cn=developers)(cn=operators)))

# 9. Find groups in a specific OU
(&(objectClass=groupOfNames)(ou=IT))

# 10. Complex condition combination example
(&
  (objectClass=groupOfNames)
  (|(cn=prod-*)(cn=dev-*))
  (!(cn=deprecated-*))
  (owner=cn=admin,dc=example,dc=com)
)
```

Examples of AND(&) and OR(|) Operators in LDAP Filters

```
# AND operator (&) - All conditions must be met
# Syntax: (&(condition1)(condition2)(condition3)...)

# Multiple attribute AND example
(&
  (objectClass=person)
  (mail=*@example.com)
  (title=Engineer)
  (manager=*)
)

# OR operator (|) - At least one condition must be met
# Syntax: (|(condition1)(condition2)(condition3)...)

# Multiple attribute OR example
(|
  (department=IT)
  (department=HR)
  (department=Finance)
)

# Combining AND and OR
(&
  (objectClass=person)
  (|
    (department=IT)
    (department=R&D)
  )
  (employeeType=FullTime)
)

# Complex condition combination
(&
  (objectClass=person)
  (|
    (&
      (department=IT)
      (title=*Engineer*)
    )
    (&
      (department=R&D)
      (title=*Developer*)
    )
```

```
    )
    (!(status=Inactive))
    (|(manager=*)(isManager=TRUE))
)
```

Synchronize LDAP Users

After successfully synchronizing LDAP users to the platform, you can view the synchronized users in the user list.

You can configure an automatic synchronization policy when adding LDAP (which can be updated later) or manually trigger synchronization after adding LDAP successfully. Here's how to manually trigger a synchronization operation.

Notes:

- Newly added users in the LDAP integrated with the platform can log in to the platform before performing the user synchronization operation. Once they successfully log in to the platform, their information will be automatically synchronized to the platform.

- Users deleted from LDAP will have an `Invalid` status after synchronization.

- The default validity period for newly synchronized users is **Permanent**.

- Synchronized users with the same name as existing users (local users, IDP users) are automatically associated. Their permissions and validity period will be consistent with existing users. They can log in to the platform using the login method corresponding to their respective sources.

Procedure of Operation

1. In the left navigation bar, click **Users** > **IDPs**.

2. Click the **_LDAP name_** that you want to manually synchronize.

3. Click **Actions** > **Sync user** in the upper-right corner.

4. Click **Sync**.
   Notes: If you manually close the synchronization prompt dialog, a confirmation dialog will appear to confirm the closure. After closing the synchronization prompt dialog, the system will continue to synchronize users. If you remain on the user list page, you will receive

synchronization result feedback. If you leave the user list page, you will not receive synchronization results.

Relevant Operations

You can click the
on the right in the list page or click **Actions** in the upper-right corner on the details page to update or delete LDAP as needed.

Operation	Description
Update LDAP	Update the configuration information of the added LDAP or the **LDAP Auto-Sync Policy**. **Note**: After updating LDAP, users currently synchronized to the platform through this LDAP will also be updated. Users removed from LDAP will become invalid in the platform user list. You can clean up junk data by executing the operation to clean up invalid users.
Delete LDAP	After deleting LDAP, all users synchronized to the platform through this LDAP will have an **Invalid** status (the binding relationship between users and roles remains unchanged), and they cannot log in to the platform. After re-integrating, synchronization needs to be re-executed to activate users. **Tips**: After deleting IDP, if you need to delete users and user groups synchronized to the platform through LDAP, check the checkbox **Clean IDP Users and User Groups** below the prompt box.

Alauda Container Platform

OIDC Management

The platform supports the OIDC (OpenID Connect) protocol, enabling platform administrators to log in using third-party accounts after adding OIDC configuration. Platform administrators can also update and delete configured OIDC services.

TOC

Overview of OIDC

OIDC (OpenID Connect) is an identity authentication standard protocol based on the OAuth 2.0 protocol. It uses an OAuth 2.0 authorization server to provide user identity authentication for third-party clients and passes the corresponding identity authentication information to the client.

OIDC allows all types of clients (including server-side, mobile, and JavaScript clients) to request and receive authenticated sessions and end-user information. This specification suite is extensible, allowing participants to use optional features such as identity data encryption,

OpenID Provider discovery, and session management when meaningful. For more information, refer to the OIDC official documentation ↗ .

Adding OIDC

By adding OIDC, you can use third-party platform accounts to log in to the platform.

Note: After OIDC users successfully log in to the platform, the platform will use the user's email attribute as the unique identifier. OIDC-supported third-party platform users must have an **email** attribute; otherwise, they will not be able to log in to the platform.

Procedure of Operation

1. In the left navigation bar, click **Users** > **IDPs**.

2. Click **Add OIDC**.

3. Configure the **Basic Information** parameters.

4. Configure the **OIDC Server Configuration** parameters:

   - **Identity Provider URL**: The issuer URL, which is the access address of the OIDC identity provider.

   - **Client ID**: The client identifier for the OIDC client.

   - **Client Secret**: The secret key for the OIDC client.

   - **Redirect URI**: The callback address after logging in to the third-party platform, which is the URL of the dex issuer + `/callback` .

   - **Logout URL**: The address visited by the user after performing the **Logout** operation. If empty, the logout address will be the platform's initial login page.

5. In the **IDP Service Configuration Validation** area, enter the **Username** and **Password** of a valid OIDC account to verify the configuration.
   Tip: If the username and password are incorrect, an error will be reported during addition, indicating invalid credentials, and OIDC cannot be added.

6. Click **Create**.

Adding OIDC via YAML

In addition to form configuration, the platform also supports adding OIDC through YAML, which allows for more flexible configuration of authentication parameters, claim mappings, user group synchronization, and other advanced features.

Example: Configuring OIDC Connector

The following example demonstrates how to configure an OIDC connector for integrating with OIDC identity authentication services. This configuration example is suitable for the following scenarios:

1. Need to integrate OIDC as an identity authentication server.

2. Need to support user group information synchronization.

3. Need to customize logout redirect address.

4. Need to configure specific OIDC scopes.

5. Need to customize claim mappings.

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
# Connector basic information
id: oidc                # Connector unique identifier
name: oidc              # Connector display name
type: oidc              # Connector type is OIDC
metadata:
  annotations:
    cpaas.io/description: "11"  # Connector description
  name: oidc
  namespace: cpaas-system
spec:
  config:
    # OIDC server configuration
    # Configure server connection information, including server address,
client credentials, and callback address
    issuer: http://auth.com/auth/realms/master          # OIDC serve
r address
    clientID: dex                                       # Client ID
    # Service account secret key, valid when creating Connector resources
for the first time
    clientSecret: xxxxxxx
    redirectURI: https://example.com/dex/callback       # Callback a
ddress, must match the address registered by the OIDC client

    # Security configuration
    # Configure SSL verification and user information acquisition method
    insecureSkipVerify: true                            # Whether to
skip SSL verification, it is recommended to set to false in a production
environment
    getUserInfo: false                                  # Whether to
obtain additional user information through the UserInfo endpoint

    # Logout configuration
    # Configure the redirect address after user logout
    logoutURL: https://test.com                         # Logout redir
ect address, can be customized to the page jumped after user logout

    # Scope configuration
    # Configure the required authorization scope, ensure that the OIDC se
rver supports these scopes
    scopes:
      - openid                                          # Required, us
```

```
ed for OIDC basic authentication
      - profile                                    # Optional, us
ed to obtain user basic information
      - email                                      # Optional, us
ed to obtain user email


    # Claim mapping configuration
    # Configure the mapping relationship between OIDC returned claims and
platform user attributes
    claimMapping:
      email: email                                 # Email mappin
g, used for user unique identification
      groups: groups                               # User group m
apping, used for organization structure
      phone: ""                                    # Phone mappin
g, optional
      preferred_username: preferred_username       # Username map
ping, used for display name


    # Custom claimextra configuration
    # External custom fields will be dynamically added to the user object
spec.extra field
    claimExtra:
      - field: xxx            # Custom field name
        type: string         # Field type value is consistent with the d
efinition of golang language type. For example: string, int, bool, map[st
ring]string, []string, []int


    # User group configuration
    # Configure user group synchronization related parameters, ensure tha
t the token contains group information
    groupsKey: groups                              # Specify the
key name of group information
    insecureEnableGroups: false                    # Whether to en
able group synchronization function
```

Relevant Operations

You can click the
on the right in the list page or click **Actions** in the upper-right corner on the details page to
update or delete OIDC as needed.

Operation	Description
Update OIDC	Update the added OIDC configuration. After updating the OIDC configuration information, the original users and authentication methods will be reset and synchronized according to the current configuration.
Delete OIDC	Delete OIDC that is no longer used by the platform. After deleting OIDC, all users synchronized to the platform through this OIDC will have an **Invalid** status (the binding relationship between users and roles remains unchanged), and they cannot log in to the platform. After re-integrating, users can be activated by successfully logging in to the platform. **Tip**: After deleting IDP, if you need to delete users and user groups synchronized to the platform through OIDC, check the checkbox **Clean IDP Users and User Groups** below the prompt box.

Alauda Container Platform

Troubleshooting

Delete User

Problem Description

Solution

Alauda Container Platform

Delete User

TOC

Problem Description

Issue: When creating or synchronizing a new user, the system indicates that the user already exists. How should you handle this?

For security reasons, the platform prevents creating new users (both local and IDP users) with names that match previously deleted users. This limitation applies to:

- Creating new local users with names matching deleted users

- Synchronizing IDP users with names matching deleted users

After upgrading to the current version, you may encounter this issue when:

- Creating new users with names that match users deleted before the upgrade

- Synchronizing new users with names that match users deleted before the upgrade

Solution

To resolve this issue, you need to clean up the deleted user information by executing specific scripts on your global cluster control nodes.

Clean up deleted IDP users

Execute the following command on any control node of your global cluster:

```
kubectl delete users -l 'auth.cpaas.io/user.connector_id=<IDP Name>,auth.cpaas.io/user.state=deleted'
```

Example:

```
kubectl delete users -l 'auth.cpaas.io/user.connector_id=github,auth.cpaas.io/user.state=deleted'
```

Clean up deleted local users

Execute these two scripts in sequence on any control node of your global cluster:

1. Clean up user passwords:

```
kubectl get users -l 'auth.cpaas.io/user.connector_id=local,auth.cpaas.io/user.state=deleted' | awk '{print $1}' | xargs kubectl delete password -n cpaas-system
```

2. Clean up users:

```
kubectl delete users -l 'auth.cpaas.io/user.connector_id=local,auth.cpaas.io/user.state=deleted'
```

Alauda Container Platform

User Policy

Introduction

Overview

Configure Security Policy

Available Policies

Alauda Container Platform

Introduction

The platform provides comprehensive user security policies to enhance login security and protect against malicious attacks.

TOC

Overview

The platform supports the following security policies:

- Password security management

- User account disablement

- User account locking

- User notifications

- Access control

Configure Security Policy

Steps

1. In the left navigation bar, click **User Role Management** > **User Security Policy**

2. Click **Update** in the top right corner

3. Configure the security policies as needed

4. Click **Update** to save changes

> **WARNING**
>
> Policy Configuration Notes:
>
> - Check the box before a policy to enable it
>
> - Uncheck the box to disable a policy
>
> - Disabled policies retain their configuration data
>
> - Previous settings are restored when re-enabling a policy

Available Policies

Policy	Description
User Authentication Policy	Enables dual authentication for password-based login: - Users receive verification codes via specified notification methods - Supports various notification servers (e.g., Enterprise Communication Tool Server)
Password Security Policy	Manages password requirements: First Login: - Forces password change on first platform login Regular Updates: - Requires password change after specified period (e.g., 90 days) - Prevents login until password is updated

Policy	Description
User Disablement Policy	Automatically disables inactive accounts: - Triggers after specified period of no login
User Locking Policy	Protects against brute force attacks: Lock Conditions: - Triggers after specified number of failed login attempts within 24 hours Lock Duration: - Account remains locked for specified minutes - Automatically unlocks after lock period expires
Notification Policy	Manages user notifications: - Sends initial password via email after user creation
Access Control	Manages user sessions and access: Session Management: - Auto-logs out inactive sessions after specified time - Limits maximum concurrent online users Browser Control: - Ends session when all product tabs are closed - Prevents multiple logins from same client :::note Important Notes: - Access Control only affects new logins after policy update - Browser tab restoration may not trigger session end - Only last login is allowed per client when preventing repeated login :::

Multitenancy(Project)

Introduction

Introduction

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

Guides

Create Project

Procedure

Manage Project Quotas

What is ProjectQuota?

How it works

Manage Pro

Update Basic P

Delete Project

Manage Project Cluster

Introduction

Add a Cluster

Remove a Cluster

Manage Pro

Import Member

Remove Memb

Alauda Container Platform

Introduction

TOC

Project

A project is a resource isolation unit that enables multi-tenant usage scenarios in enterprises. It divides resources from one or more clusters into isolated environments, ensuring both resource and personnel isolation. Projects can represent different subsidiaries, departments, or project teams within an enterprise. Through project management, you can achieve:

- Resource isolation between project teams

- Quota management within tenants

- Efficient resource allocation and control

Namespaces

Namespaces are smaller, mutually isolated resource spaces within a project. They serve as workspaces for users to implement their production workloads. Key characteristics of namespaces include:

- Multiple namespaces can be created under a project

- Total resource quota of all namespaces cannot exceed the project quota

- Resource quotas are allocated more granularly at the namespace level

- Container sizes (CPU, memory) are limited at the namespace level

- Improved resource utilization through fine-grained control

Relationship Between Clusters, Projects, and Namespaces

The platform's resource hierarchy follows these rules:

- A project can utilize resources (CPU, memory, storage) from multiple clusters, and a cluster can allocate resources to multiple projects.

- Multiple namespaces can be created under a project, with their combined resource quotas not exceeding the total project resources.

- A namespace's resource quota must come from a single cluster, and a namespace can only belong to one project.

Alauda Container Platform

Guides

Create Project

Procedure

Manage Project Quotas

What is ProjectQuota?

How it works

Manage Pro

Update Basic P

Delete Project

Manage Project Cluster

Introduction

Add a Cluster

Remove a Cluster

Manage Pro

Import Member

Remove Memb

Alauda Container Platform

Create Project

Before your project team starts working, you can create a project based on the existing cluster resources on the platform. The project will be isolated from other projects (tenants) in terms of both resources and personnel. When creating a project, you can allocate resources according to your project scale and actual business needs. The project can utilize resources from multiple clusters on the platform.

> **WARNING**
>
> When creating a project, the platform will automatically create a namespace with the same name as the project in the associated clusters to isolate platform-level resources. Please do not modify this namespace or its resources.

TOC

Procedure

Procedure

1. In the **Project Management** view, click **Create Project**.

2. On the **Basic information** page, configure the following parameters:

Parameter	Description
Name	The name of the project, which cannot be the same as an existing project name or any name in the project name blacklist. Otherwise, the project cannot be created. **Note**: The project name blacklist includes special namespace names under platform clusters: `cpaas-system`, `cert-manager`, `default`, `global-credentials`, `kube-ovn`, `kube-public`, `kube-system`, `nsx-system`, `alauda-system`, `kube-federation-system`, `ALL-ALL`, and `true`.
Cluster	The cluster(s) associated with the project, where the administrator can allocate resource quotas. Click the drop-down selection box to select one or more clusters. **Note**: Clusters in abnormal state cannot be selected.

3. Click **Next** and in the project quota setting step, read Manage Resource Quotas to set the resource quotas to be allocated to the current project for the selected clusters.

4. Click **Create Project**.

Alauda Container Platform

Manage Project Quotas

This guide explains how ACP extends Kubernetes ResourceQuota with a project-level aggregate quota (ProjectQuota). ProjectQuota lets you cap the sum of ResourceQuotas across all namespaces in a project, so you can plan and govern capacity at the project level while still delegating limits to individual namespaces.

TOC

What is ProjectQuota?

How it works

When to use ProjectQuota

Quota keys and units

Allocation strategy tips

Best practices and FAQs

What is ProjectQuota?

- ResourceQuota (Kubernetes native) limits resources per namespace (CPU, memory, object counts, etc.). For concepts, keys, and usage, please refer to:

  - Resource Quotas

- ProjectQuota defines a project-wide upper bound: the total of all namespace ResourceQuotas within the project must not exceed the project's hard limits for the same keys.

In short: ResourceQuota caps a single namespace; ProjectQuota caps the sum across all namespaces in a project.

How it works

- Workflow order: define or adjust the ProjectQuota first, then allocate per-namespace ResourceQuotas within that project budget.

- Scope: ProjectQuota applies to a platform project and governs all namespaces that belong to it.

- Aggregate enforcement at admission time:

  - When creating or updating a namespace's ResourceQuota, the platform computes the aggregate for the same keys (for example, `limits.cpu`, `requests.memory`, `pods`) across all namespaces in the project, including the incoming change.

  - The request is allowed only if the new aggregate remains less than or equal to the corresponding ProjectQuota hard limits. Otherwise, the change is rejected with an explanatory error.

- Execution model:

  - ProjectQuota constrains what can be allocated via namespace ResourceQuotas (pre-allocation), not the instantaneous runtime usage. Actual consumption remains governed by each namespace's ResourceQuota and the scheduler.

When to use ProjectQuota

- Budget/capacity governance per project: allocate a fixed CPU/memory/object budget, then subdivide across namespaces.

- Multi-team or multi-environment projects (for example, `dev` / `staging` / `prod`) that share a common upper bound.

- Preventing quota drift: keep a single "big bucket" at the project layer so namespace quotas do not silently inflate over time.

Quota keys and units

ProjectQuota supports the same common keys as ResourceQuota (non-exhaustive):

- Compute and memory: `limits.cpu`, `limits.memory`, `requests.cpu`, `requests.memory`
- Workload/object counts: `pods`, `services`, `configmaps`, `secrets`, `pvc`, and more

Units and counting rules:

- CPU uses cores (for example, `2`, `500m`)
- Memory uses bytes (for example, `8Gi`)
- Object-style keys use integer counts

If the sum of the corresponding keys across all namespaces approaches or exceeds the ProjectQuota hard limit, ACP blocks further ResourceQuota creation or expansion for that key.

Allocation strategy tips

- Define the project "big bucket" first (ProjectQuota), then split it into per-namespace ResourceQuotas for teams/environments.
- Keep 10% - 30% headroom for spikes and elastic scaling.
- Review regularly: reclaim underused quota and reassign; raise consistently constrained namespaces, and adjust the project cap accordingly.

Best practices and FAQs

- Q: Increasing a namespace's `limits.memory` fails with an error about exceeding project quota. Why?

  - A: The project's ProjectQuota hard limit for that key would be exceeded by the requested change. Reduce other namespaces' quotas, or raise the project cap first and then retry the namespace change.

- Q: I raised the ProjectQuota, but workloads still won't schedule.

  - A: Ensure each namespace's ResourceQuota is also increased appropriately and verify underlying cluster/node capacity.

- Recommendation: Manage ProjectQuota as part of your normal change control, aligned with capacity planning (nodes/storage) and budget management.

Alauda Container Platform

Manage Project

This guide explains how to update basic information and project quotas for a specified project, or delete the project.

TOC

Update Basic Project Information

Update basic information for a specified project, such as display name and description.

Procedure

1. In the **Project Management** view, click on the project name to be updated.

2. In the left navigation pane, click **Details**.

3. Click **Actions** > **Update Basics** in the upper right corner.

4. Modify or enter the **Display name** and **Description**.

5. Click **Update**.

Delete Project

Delete projects that are no longer in use.

> **WARNING**
>
> After the project is deleted, the resources occupied by the project in the cluster will be released.

Procedure

1. In the **Project Management** view, click on the project name to be deleted.

2. In the left navigation bar, click **Details**.

3. Click **Actions** > **Delete Project** in the upper right corner.

4. Enter the name of the project and click **Delete**.

Alauda Container Platform

Manage Project Cluster

This guide explains how to manage cluster associations for a project. You can add clusters to allocate their resources to the project, or remove clusters to reclaim the allocated resources.

TOC

Introduction

You can add clusters to a project to allocate their resources, or remove clusters to reclaim the allocated resources. This functionality is useful in the following scenarios:

- When project resources are insufficient for business operations

- When a newly created or added cluster needs to be allocated to an existing project

- When cluster resources need to be reclaimed from a project

- When a specific project needs exclusive access to a cluster

Add a Cluster

Add a cluster to a project and set its resource quota.

Procedure

1. In the **Project Management** view, click on the project name where you want to add the cluster.

2. In the left navigation bar, click **Details**.

3. Click **Actions** > **Add Cluster** in the upper right corner.

4. Select the cluster and set the resource quota to be allocated to the current project. The following resources can be configured:

   - CPU (cores)

   - Memory (Gi)

   - Storage (Gi)

   - PVC count (number)

   - Pods (number)

   - vGPU (virtual GPU)/MPS/pGPU (physical GPU, cores)

   - Video memory quota

> **NOTE**
>
> - GPU resource quota can only be configured when GPU plugins are deployed in the cluster.
>
>   When GPU resources are **GPU-Manager or MPS GPU**, **vMemory** quota can also be configured.
>
>   **GPU Units**: 100 units of virtual cores are equivalent to 1 physical core (1 pGPU = 1 core = 100 GPU-Manager core = 100 MPS core), and pGPU units can only be allocated in whole numbers. GPU-Manager 1 unit of memory is equal to 256 Mi, MPS GPU 1 unit of memory is equal to 1 Gi, and 1024 Mi = 1 Gi.
>
> - If no quota is set for a certain type of resource, it defaults to **Unlimited**. This means that the project can use the available resources of the corresponding type in the cluster as needed, without a maximum limit.

- The value of the project quota set should be within the quota range displayed on the page. Under each resource quota input box, the allocated quota and total amount of that resource will be displayed for reference.

5. Click **Add**.

Remove a Cluster

Remove a cluster associated with a project.

> **WARNING**
>
> - After removing a cluster, the project cannot use the business resources under the removed cluster.
>
> - When the cluster to be removed is abnormal, the resources under the abnormal cluster cannot be cleaned up. It is recommended to fix the cluster before removing it.

Procedure

1. In the **Project Management** view, click on the project name where you want to remove the cluster.

2. In the left navigation bar, click **Details**.

3. Click **Actions** > **Remove Cluster** in the upper right corner.

4. In the pop-up **Remove Cluster** dialog box, enter the name of the cluster to be removed, and then click the **Remove** button to successfully remove the cluster.

Alauda Container Platform

Manage Project Members

This guide explains how to manage project members, including importing members and assigning project-related roles.

TOC

Import Members

You can grant users operation permissions for the project and its namespaces by importing existing platform users or adding OIDC users. Assign platform-provided roles such as project administrators, namespace administrators, or developers. If you need tailor-made permissions, create a native Kubernetes Role/ClusterRole under **Users > Platform Roles > Kubernetes Roles** and bind it to the desired users through RoleBinding/ClusterRoleBinding.

Constraints and Limitations

- When no OIDC IDP is configured on the platform:

  - Only existing platform users can be imported as project members, including:

    - OIDC users who have successfully logged in

    - Users synchronized through LDAP

    - Local users

    - Users added to other projects as OIDC users (with source marked as -)

- When an OIDC IDP is configured:

  - You can add valid OIDC accounts that meet the input requirements

  - Account validity cannot be verified during addition

  - Ensure the account is valid, or it won't be able to log in normally

- System default administrator users and the currently logged-in user cannot be imported

Procedure

1. In the **Project Management** view, click on the project name to be managed.

2. In the left navigation bar, click **Members**.

3. Click **Import Member**.

4. Choose either **Member List** or **OIDC Users**.

Import from Member List

You can import either all users or selected users from the member list.

> **TIP**
>
> Use the user group dropdown menu in the upper right corner and the search box to filter users by username.

To import all users:

1. Select **Member List**.

2. Click the **Bind** dropdown menu and select the role to assign to all users.

   If the role requires a namespace, select it from the **Namespaces** dropdown menu.

3. Click **Import All**.

To import specific users:

1. Select **Member List**.

2. Select one or more users using the checkboxes.

3. Click the **Bind** dropdown menu and select the role to assign to the selected users.

   If the role requires a namespace, select it from the **Namespaces** dropdown menu.

4. Click **Import**.

Import OIDC Users

1. Select **OIDC Users**.

2. Click **Add** to create a member record (repeat for multiple members).

3. Enter the OIDC-authenticated username in the **Name** field.

> **WARNING**
>
> Ensure the username corresponds to an account that can be authenticated by the configured OIDC system, or login will fail.

4. Select the role from the **Roles** dropdown menu.

   If the role requires a namespace, select it from the **Namespaces** dropdown menu.

5. Click **Import**.

After successful import, you can view:

- The member in the project member list

- The user in **Platform Management > Users**

  - Source shows as "-" until first login/sync

  - Source updates after successful login/sync

Remove Members

Remove a project member to revoke their permissions.

Procedure

1. In the **Project Management** view, click on the project name.

2. In the left navigation bar, click **Members**.

> **TIP**
>
> Use the dropdown list next to the search box to filter members by **Name**, **Display name**, or **User Group**.

3. Click **Remove** next to the member you want to remove.

4. Confirm removal in the prompt dialog.

Audit

Introduction

Prerequisites

Procedure

Search Results

Alauda Container Platform

Introduction

The platform's auditing function provides time-ordered operation records related to users and system security. This helps you analyze specific issues and quickly resolve problems that occur in clusters, custom applications, and other areas.

Through auditing, you can track various changes in the Kubernetes cluster, including:

- What changes occurred in the cluster during a specific time period

- Who performed these changes (system components or users)

- Details of important change events (e.g., POD parameter updates)

- Event outcomes (success or failure)

- Operator location (internal or external to the cluster)

- User operation records (updates, deletions, management operations) and their results

TOC

Prerequisites

- Your account must have platform management or platform auditing permissions.

- This feature relies on the logging service. Please ensure that the ACP Log Essentails , ACP Log Collector and ACP Log Storage plugins are installed within the platform beforehand.

> **Note**
>
> Because Logging Service releases on a different cadence from Alauda Container Platform, the Logging Service documentation is now available as a separate documentation set at [Logging Service ↗](#).

Procedure

1. In the left navigation bar, click **Auditing**.

2. Select the auditing scope from the tabs:

   - **User Operations**: View operation records of users who have logged in to the platform
   - **System Operations**: View system operation records (operators start with `system:`)

3. Configure query conditions to filter auditing events:

Query Condition	Description
Operator	Username or system account name of the operator (default: `All`)
Actions	Type of operation (create, update, delete, manage, rollback, stop, etc., default: `All`)
Clusters	Cluster containing the operated resource (default: `All`)
Resource Type	Type of the operated resource (default: `All`)
Resource Name	Name of the operated resource (supports fuzzy search)

4. Click **Search**.

> **TIP**
>
> - Use the **Time Range** dropdown to set the audit time range (default: `Last 30 Minutes`). You can select a preset range or customize one.
>
> - Click the refresh icon to update search results.
>
> - Click the export icon to download results as a `.csv` file.

Search Results

The search results display the following information:

Parameter	Description
Operator	Username or system account name of the operator
Actions	Type of operation (create, update, delete, manage, rollback, stop, etc.)
Resource Name/Type	Name and type of the operated resource
Clusters	Cluster containing the operated resource
Namespaces	Namespace containing the operated resource
Client IP	IP address of the client used to execute the operation
Operation Result	Operation outcome based on API return code (2xx = success, other = failure)
Operation Time	Timestamp of the operation
Details	Click the **Details** button to view the complete audit record in JSON format in the **Audit Details** dialog box

Alauda Container Platform

Telemetry

Install

Prerequisites

Installation Steps

Enable Online Operations

Uninstallation Steps

Alauda Container Platform

Install

ACP Telemetry is a platform service that collects telemetry data from clusters for online operations and maintenance. It collects system metrics and uploads them to Alauda Cloud for monitoring and analysis.

TOC

Prerequisites

Before installation, ensure that:

- The Alauda Container Platform has a valid license

- The global cluster has internet access

Installation Steps

1. Navigate to **Administrator**

2. In the left navigation bar, click **Marketplace** > **Cluster Plugins**

3. Select the **global** cluster in the top navigation bar

4. Search for **ACP Telemetry** and click to view its details

5. Click **Install** to deploy the plugin

Enable Online Operations

1. In the left navigation bar, click **System Settings** > **Platform Maintenance**

2. Click the **On** button for **Online Operations**

Uninstallation Steps

1. Follow steps 1-4 from the installation process to locate the plugin

2. Click **Uninstall** to remove the plugin

Alauda Container Platform

Certificates

Automated Kubernetes Certific   **cert-manager**   **OLM Certifi**

Certificate Monitoring   **Rotate TLS**

Alauda Container Platform

Automated Kubernetes Certificate Rotation

This guide helps you install, understand, and operate the Kubernetes Certificate Rotator in ACP to automate the rotation of Kubernetes certificates within your clusters.

TOC

Installation

See Cluster Plugin for installation instructions.

> **Note:**
>
> - Currently supported:
>
>   - On-Premises clusters
>
>   - DCS clusters

How it works

This plugin handles automatic rotation for the following certificates.

Certificate file	Function	Node Type
apiserver.crt	Server certificate for kube-apiserver	Control Plane Node
apiserver-etcd-client.crt	Client certificate for kube-apiserver to access etcd	Control Plane Node
apiserver-kubelet-client.crt	Client certificate for kube-apiserver to access kubelet	Control Plane Node
front-proxy-client.crt	Client certificate for kube-apiserver to access aggregated API servers	Control Plane Node
etcd/server.crt	Server certificate for etcd	Control Plane Node
etcd/peer.crt	Peer communication certificate between etcd members	Control Plane Node
/root/.kube/config, admin.conf, super-admin.conf	Client certificate in kubeconfig for cluster administration	Control Plane Node
controller-manager.conf	Client certificate in kubeconfig for kube-controller-manager	Control Plane Node
scheduler.conf	Client certificate in kubeconfig for kube-scheduler	Control Plane Node
kubelet.crt	Server certificate for kubelet	All Nodes
kubelet-client-current.pem	Client certificate for kubelet (referenced by kubelet.conf)	All Nodes

Rotation Process

1. **Load certificate information**

The initial step involves gathering metadata for all target certificates. Since these certificates are stored in different paths on the host, their contents must be read from the respective files. To achieve this, a temporary Pod is created on the target node with the certificate directories mounted, allowing the Pod to read the information. The certificate's information is collected once per day. Certificate details (paths, expiration) are maintained in the ConfigMap `cpaas-system/node-local-certs-<node-name>`. The encrypted CA certificate is stored in Secret `cpaas-system/kubernetes-ca`.

2. **Rotation Trigger Condition**

   The `notBefore` and `notAfter` fields of the certificate indicate the validity period. Rotation is triggered if the remaining validity period is less than 20% or 30 days.

3. **Rotation queue**

   Certificates requiring rotation are placed in a queue for processing. The rotation program evaluates recent rotation activities and the urgency of pending tasks to decide whether to process them immediately. This prevents potential cluster health issues caused by the simultaneous rotation of multiple certificates.

4. **Generate new certificates**

   The rotation program generates new certificates based on internally stored CA information. The rotation process creates a temporary Pod on the target node with the necessary certificate directories mounted, allowing for controlled file modifications.

5. **Restart the components**

   Requiring restart:

   - `kube-apiserver`: It needs to be restarted to load the new certificates. During restart, it regenerates its internal loopback certificate (valid for one year, used only internally and can not be externally rotated).

   - `kube-controller-manager`: It needs to be restarted to reload the kubeconfig file.

   - `kube-scheduler`: It needs to be restarted to reload the kubeconfig file.

   - `kubelet`: It needs to be restarted to reload the server certificate.

   Restart method: Add annotations to the respective static Pods' YAML files to trigger the kubelet to recreate the Pods. To restart kubelet, mount the host filesystem with `hostPID is true` and run "systemctl restart kubelet" in the container.

   Auto-reloading:

   - Etcd can auto-reload the certificates.

6. **Rotation Timelines**

- `kubelet` certificates: Rotate at 61 days (91-day validity)

- Control plane certificates: Rotate at 292 days (365-day validity)

Operation Considerations

If `kubelet` is in an abnormal state during the rotation window and cannot rotate certificates automatically, manual rotation is required:

Operators must manually renew the certificates.

Run the following commands to renew the certificates manually:

```
cert-renew --ca-cert <ca-cert-path> --ca-key <ca-key-path> --days <days>
<certificate or kubeconfig 1> <certificate or kubeconfig 2> ...
```

For example to renew the `kubelet.crt`:

```
cert-renew --ca-cert /etc/kubernetes/pki/ca.crt --ca-key /etc/kubernetes/
pki/ca.key --days 91 /etc/kubernetes/pki/kubelet.crt
```

To download and prepare the `cert-renew` tool, run:

```
curl "$(kubectl get services -n cpaas-system frontend -o jsonpath='{.spe
c.clusterIP}'):8080/cluster-cert-rotator/download/cert-renew" -o ./cert-r
enew && chmod +x ./cert-renew
```

Optionally, download `renew-all.sh` to renew all certificates on the node:

```
curl "$(kubectl get services -n cpaas-system frontend -o jsonpath='{.spe
c.clusterIP}'):8080/cluster-cert-rotator/download/renew-all.sh" -o ./rene
w-all.sh
```

Alauda Container Platform

cert-manager

Each cluster will automatically deploy **Certificate for cert-manager**

cert-manager is a native Kubernetes certificate management controller that automatically generates and manages TLS certificates based on `Certificate` resources. Many components in Kubernetes clusters use cert-manager to manage their TLS certificates, ensuring secure communication.

TOC

Overview

Cert-manager manages the lifecycle of certificates through Kubernetes Custom Resource Definitions (CRDs):

- **Certificate**: Defines the certificates that need to be managed

- **Issuer/ClusterIssuer**: Defines certificate issuers

- **CertificateRequest**: Internal resource for processing certificate requests

How it works

When a `Certificate` resource is created, cert-manager automatically:

1. Generates private keys and certificate signing requests

2. Obtains signed certificates from the specified Issuer

3. Stores certificates and private keys in Kubernetes Secrets

Additionally, cert-manager monitors the validity period of certificates and renews them before they expire to ensure continuous service availability.

Identifying cert-manager Managed Certificates

Certificates managed by cert-manager have corresponding `Secret` resources with type `kubernetes.io/tls` and specific labels and annotations.

Common Labels and Annotations

`Secret` resources managed by cert-manager typically contain the following labels and annotations:

Labels:

- `controller.cert-manager.io/fao: "true"` : Identifies that this Secret is managed by cert-manager and enables filtered Secret caching by the controller.

Annotations:

- `cert-manager.io/certificate-name` : Certificate name
- `cert-manager.io/common-name` : Common name of the certificate
- `cert-manager.io/alt-names` : Alternative names of the certificate
- `cert-manager.io/ip-sans` : IP addresses of the certificate
- `cert-manager.io/issuer-kind` : Type of certificate issuer
- `cert-manager.io/issuer-name` : Name of certificate issuer

- `cert-manager.io/issuer-group` : API group of the issuer

- `cert-manager.io/uri-sans` : URI Subject Alternative Names

Related Resources

- [cert-manager Official Documentation ↗](#)

Alauda Container Platform

OLM Certificates

All certificates for **Operator Lifecycle Manager (OLM)** components — including `olm-operator`, `catalog-operator`, `packageserver`, and `marketplace-operator` — are automatically managed by the system.

When installing Operators that define **webhooks** or **API services** in their **ClusterServiceVersion (CSV)** object, OLM automatically generates and rotates the required certificates.

Certificate Monitoring

Cluster Enhancer provides monitoring capabilities for certificates used in Kubernetes clusters. The monitoring scope includes:

1. **Kubernetes component certificates**, including control plane and kubelet server/client certificates (including kubeconfig client certificates)
2. **Certificates of components running in the cluster**, implemented by inspecting all Secrets with type `kubernetes.io/tls`
3. **Server certificates actually used by kube-apiserver** (including internal loopback certificates for self-access) by accessing the `kubernetes` Endpoints

Users can find and install **Cluster Enhancer** in the **Administrator** view by navigating to **Marketplace** > **Cluster Plugins** in the left navigation.

TOC

Certificate Status Monitoring

The expiration status of certificates can be viewed through the metric `certificate_expires_status`. The expiration time of certificates can be viewed through

the metric `certificate_expires_time` .

The current certificate status and expiration time can be viewed in the **Certificate Status** sub-tab. To access this sub-tab, go to the **Administrator** view, navigate to **Clusters > Clusters**, select a specific cluster, then go to the **Monitoring** tab.

Built-in Alert Rules

Cluster Enhancer provides built-in alert rules `cpaas-certificates-rule` with the following alerts:

Kubernetes Certificate Alerts

Rule	Level
The expiration time of the kubernetes certificate is about to expire (less than 30 days) <= 30d and last 1 minutes	Medium
The expiration time of the kubernetes certificate is about to expire (less than 10 days) <= 10d and last 1 minutes	High
Kubernetes certificate has expired <= 0d and last 1 minutes	Critical

Platform Components Certificate Alerts

Rule	Level
The expiration time of the platform components certificate is about to expire (less than 30 days) <= 30d and last 1 minutes	Medium
The expiration time of the platform components certificate is about to expire (less than 10 days) <= 10d and last 1 minutes	High
Platform components certificate has expired <= 0d and last 1 minutes	Critical

Alauda Container Platform

Rotate TLS Certs of Platform Access Addresses

> **INFO**
>
> For ACP version v4.0.x, apply the same procedure to both the primary cluster and the standby cluster (terms of the Disaster Recovery setup) as described here.

TOC

Prerequisites

Procedures

Prerequisites

- A pair of TLS certificates and its private key.

Procedures

1. On any control-plane node in the global cluster, export backups of the TLS certificates used by ACP's platform access addresses:

```
kubectl get certificate -n cpaas-system dex-serving-cert --ignore-not-f
ound=true -o yaml > /cpaas/dex-serving-cert.yaml
kubectl get secret -n cpaas-system dex.tls -o yaml > /cpaas/dex.tls.yam
l
```

2. Delete the current certificates:

```
kubectl delete certificate -n cpaas-system dex-serving-cert --ignore-no
t-found=true
kubectl delete secret -n cpaas-system dex.tls
```

3. Introduce the new certificate:

```
kubectl create secret tls dex.tls --cert=/path/to/tls.crt --key=/path/t
o/tls.key -n cpaas-system
```