Security

Alauda Container Security

Alauda Container Security

Alauda Cluster Authentication

Alauda Cluster Authentication

Security and Compliance

Compliance API Refiner About Alau

Platform Security Configurations

Changing the Platform OIDC Cl

Prerequisites

OIDC Secret Overview

Procedure

Rollback

Disabling the PKCE Plain Method

Prerequisites

Procedure

Verification

Rollback

Users and Roles

User

Group

Role

IDP

User Policy

Multitenancy(Project)

Introduction

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

Guides

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 M

Certificate Statu

Built-in Alert Ru

Rotate TLS Certs of Platform Access Addresses

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 ↗.

Security and Compliance

Compliance

Introduction

Install Alauda Container Platfor HowTo

- Install via console
- Install via YAML
- Uninstallation Procedures

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

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

HowTo

Private Registry Access Config

Why Does Kyverno Need Registry Access

Apply Security Context Constra

Introduction

Image Signa

What is Image S

Quick Start

Who does what

Scenarios

Prerequisites

Steps

Quick Start

Common Use C

Image Signature Verification Policy with Secrets

Why Use Secrets for Public Keys?

Quick Start

Secret Creation Methods

Common Use Cases

Image Regi

What is Image I

Quick Start

Common Scena

Advanced Patte

Best Practices

Container Escape Prevention P

What is Container Escape Prevention?

Quick Start

Core Container Escape Prevention Policie

Advanced Scenarios

Testing and Validation

Best Practices

Security Context Enforcement P

What is Security Context Enforcement?

Quick Start

Core Security Context Policies

Advanced Scenarios

Testing and Validation

Network Se

What is Networ

Quick Start

Core Network S

Advanced Scer

Testing and Val

Kyverno Policy Configuration Use Cases

Use Kyverno Generate Rules

How Generate Works

Example Scenario

Prerequisites

Prepare the Target Resource Dependency

Prepare the Source Resource

Grant Kyverno Permission

Create the Generate Policy

ce/F

Val

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 ↗.

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

HowTo

Private Registry Access Config

- Why Does Kyverno Need Registry Access
- Quick Start

Apply Security Context Constra

- Introduction
- Who does what
- Scenarios
- Prerequisites
- Steps

Image Signa

- What is Image S
- Quick Start
- Common Use C

Image Signature Verification Policy with Secrets

- Why Use Secrets for Public Keys?
- Quick Start
- Secret Creation Methods
- Common Use Cases

Image Regis

- What is Image I
- Quick Start
- Common Scena
- Advanced Patte
- Best Practices

Container Escape Prevention P

- What is Container Escape Prevention?
- Quick Start
- Core Container Escape Prevention Policie
- Advanced Scenarios
- Testing and Validation

Security Context Enforcement P

- What is Security Context Enforcement?
- Quick Start
- Core Security Context Policies
- Advanced Scenarios
- Testing and Validation

Network Se

- What is Networ
- Quick Start
- Core Network S
- Advanced Scer

Best Practices

Testing and Val

Kyverno Policy Configuration Use Cases

Use Kyverno Generate Rules

ce/F

How Generate Works

Example Scenario

Prerequisites

Val

Prepare the Target Resource Dependency

Prepare the Source Resource

Grant Kyverno Permission

Create the Generate Policy

Verify the Generate Rule

Troubleshooting

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
```

Apply Security Context Constraints for Pod Security

This guide is for platform administrators and security administrators. It shows you how to install a SecurityContextConstraints (SCC) engine on top of an existing Kyverno deployment, and how to bind SCC profiles to ServiceAccounts, Users, and Groups so that Pod security boundaries are enforced automatically at admission time.

TOC

Introduction

OpenShift's SecurityContextConstraints (SCC) model lets cluster administrators define a library of pod security profiles, then grant subjects (ServiceAccounts, Users, Groups) the right to use specific profiles. When a Pod is admitted, the platform picks the most appropriate SCC the subject is allowed to use, fills in missing defaults, and validates the Pod against that profile. Workloads themselves do not need to declare every security field — the SCC profile does it for them.

Vanilla Kubernetes has no equivalent built-in. This guide installs a Kyverno-based engine that reproduces the SCC experience on any standard Kubernetes cluster that already runs Kyverno. It uses:

- A `SecurityContextConstraints` CRD (`security.alauda.io/v1alpha1`) to store SCC profiles.
- Standard Kubernetes RBAC (`use` verb plus `resourceNames`) to bind subjects to profiles, so the operator workflow matches OpenShift (`oc adm policy add-scc-to-user` patterns translate one-to-one).
- A pair of Kyverno admission policies — one mutating, one validating — that select the right SCC, fill in defaults, and reject Pods that no granted SCC can accept.
- Five `GlobalContextEntry` resources that cache the SCC profiles and the relevant RBAC objects in memory so admission decisions do not require additional API calls.

The result: application teams continue to write straightforward Pod manifests, the cluster automatically constrains them to a security profile their ServiceAccount is allowed to use, and migration from OpenShift requires no changes to the binding model.

SCC authorization is a security-control change. Application teams should not be granted permission to create or modify SCC RBAC bindings directly, because doing so lets them bypass the cluster security boundary. Application teams should describe the workload requirement, such as `anyuid`, `hostNetwork`, or `hostPath`; platform or security administrators review the request and bind the least-privilege SCC to the appropriate subject.

Who does what

Use the following table to decide which parts of this guide apply to you.

Role	What you do	What you should not do
Platform administrator or security administrator	Install the SCC engine, approve SCC requests, create SCC RBAC bindings, switch the validating policy from `Warn` to `Deny`, and audit exceptions.	Do not grant broad SCCs such as `privileged`, `hostaccess`, or `anyuid` without a workload-level reason and an owner.

Role	What you do	What you should not do
Application manager or application owner	Identify what the workload needs, such as root UID, host networking, host ports, host paths, user namespaces, or a fixed UID range. Provide the namespace, ServiceAccount, workload name, and reason to the platform or security administrator. Deploy the workload with the assigned ServiceAccount after approval.	Do not create SCC RBAC bindings or grant SCC permissions to your own ServiceAccount. Do not use `alauda.io/required-scc` unless the requested SCC has already been approved and bound.

If you are a platform or security administrator, follow Part 1 and Part 2. If you are an application manager, use Step 2.1 to prepare the SCC request, then use Step 2.5 and Step 2.6 only after the SCC has been approved and bound by an administrator. Do not apply the RBAC manifests in Step 2.2 through Step 2.4 yourself.

The normal workflow is:

1. The application manager identifies the workload requirement and target ServiceAccount.
2. The platform or security administrator selects the least-privilege SCC and creates the RBAC binding.
3. The application manager deploys the workload with the approved ServiceAccount, and adds `alauda.io/required-scc` only when the administrator asks for a specific SCC to be pinned.
4. The administrator verifies authorization with `kubectl auth can-i`, and the workload owner verifies the admitted Pod has the expected `alauda.io/scc` annotation.

Scenarios

Apply this guide when any of the following apply:

- You are migrating workloads from OpenShift and want to keep the existing `oc adm policy add-scc-to-*` binding model so that platform teams and audit tooling continue to work unchanged.

- You already use Kyverno and need a centrally managed security boundary that does not require every Pod manifest to declare a full `securityContext`.
- You operate a multi-tenant cluster and want different ServiceAccounts in different namespaces to receive different security ceilings — for example, an application SA limited to `restricted-v2`, a log-collector SA allowed `hostmount-anyuid`, and an ingress controller SA allowed `NET_BIND_SERVICE`.
- You want one cluster-wide place to express and audit "who is allowed to run privileged Pods" without scattering exemptions across every namespace.

Prerequisites

Before you start, make sure all of the following are true:

1. The Kubernetes cluster runs version **1.30 or later** (CEL admission is stable).
2. Kyverno is already installed and running, at version **v4.3.1 or later**, with the `MutatingPolicy`, `ValidatingPolicy`, and `GlobalContextEntry` CRDs available. You can verify with:

   ```
   kubectl get crd validatingpolicies.policies.kyverno.io mutatingpolicies.policies.kyverno.io globalcontextentries.kyverno.io
   ```

3. The `kyverno` namespace contains these ServiceAccounts (default Kyverno installation):
   - `kyverno-admission-controller`
   - `kyverno-background-controller`
   - `kyverno-reports-controller`
4. You have **cluster-admin** (or equivalent) permissions, because installing the engine requires creating a CRD, ClusterRoles, ClusterRoleBindings, GlobalContextEntries, and admission policies.
5. You have reviewed the Pod Security Admission (PSA) `enforce` label on each namespace where you intend to allow non-`restricted` Pods. PSA runs **before** Kyverno; if a namespace is labelled `pod-security.kubernetes.io/enforce: restricted`, that namespace will reject any Pod that matches a permissive SCC such as `anyuid` or `hostnetwork-v2` before Kyverno is consulted. Adjust the namespace label to `baseline`

or `privileged` where appropriate, or restrict the SCC profile set you offer in those namespaces.

> **Tip**
>
> The engine installation is one-time work and is normally performed by a platform administrator. Part 2 is also an administrator workflow: platform or security administrators bind SCC profiles after reviewing workload requirements. Application teams typically only provide those requirements and then use the assigned ServiceAccount.

Steps

The work splits into two parts:

- **Part 1** installs the SCC engine cluster-wide. Run it once per cluster.
- **Part 2** authorizes workloads by binding SCC profiles to ServiceAccounts, Users, and Groups, and pins specific workloads to specific SCCs when needed.

Part 1: Install the SCC engine

Step 1.1 — Install the `SecurityContextConstraints` CRD

Save the following manifest as `scc-crd.yaml`. It defines a cluster-scoped `SecurityContextConstraints` resource (short name `scc`) whose fields mirror OpenShift SCC semantics.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: securitycontextconstraints.security.alauda.io
spec:
  group: security.alauda.io
  names:
    plural: securitycontextconstraints
    singular: securitycontextconstraints
    kind: SecurityContextConstraints
    listKind: SecurityContextConstraintsList
    shortNames:
      - scc
  scope: Cluster
  versions:
    - name: v1alpha1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          description: |
            SecurityContextConstraints governs the ability to make reques
ts that affect
            container security context. This custom CRD mirrors OpenShift
SCC semantics
            while keeping fields under spec for Kyverno CEL consumption.
          type: object
          required:
            - spec
          properties:
            apiVersion:
              type: string
            kind:
              type: string
            metadata:
              type: object
            spec:
              type: object
              required:
                - runAsUser
              properties:
                allowHostPorts:
                  description: Determines if the profile allows host port
```

```
s in containers.
                  type: boolean
                priority:
                  description: Higher priority SCC is evaluated first.
                  type: integer
                  format: int32
                  nullable: true
                restrictiveScore:
                  description: Secondary sort key. Lower score means less
restrictive.
                  type: integer
                  format: int32
                  minimum: 0
                requiredDropCapabilities:
                  description: Capabilities that must be dropped.
                  type: array
                  nullable: true
                  items:
                    type: string
                  x-kubernetes-list-type: atomic
                allowPrivilegedContainer:
                  description: Determines if privileged containers are al
lowed.
                  type: boolean
                runAsUser:
                  description: Strategy controlling runAsUser.
                  type: object
                  nullable: true
                  properties:
                    type:
                      description: Strategy type for runAsUser.
                      type: string
                      enum:
                        - RunAsAny
                        - MustRunAs
                        - MustRunAsRange
                        - MustRunAsNonRoot
                        - MustRunAsNonRootOrSystem
                    uid:
                      description: Required when type=MustRunAs.
                      type: integer
                      format: int64
                      minimum: 0
                    uidRangeMin:
```

```
                description: Minimum uid for MustRunAsRange.
                type: integer
                format: int64
                minimum: 0
              uidRangeMax:
                description: Maximum uid for MustRunAsRange.
                type: integer
                format: int64
                minimum: 0
          users:
            description: Users who can use this SCC.
            type: array
            nullable: true
            items:
              type: string
            x-kubernetes-list-type: atomic
          groups:
            description: Groups who can use this SCC.
            type: array
            nullable: true
            items:
              type: string
            x-kubernetes-list-type: atomic
          allowHostDirVolumePlugin:
            description: Determines if hostPath-like volume plugin
usage is allowed.
            type: boolean
          seccompProfiles:
            description: Allowed seccomp profiles. '*' allows all.
            type: array
            nullable: true
            items:
              type: string
              pattern: "^(\\*|runtime/default|unconfined|localhos
t/.+)$"
            x-kubernetes-list-type: atomic
          allowHostIPC:
            description: Determines if host IPC is allowed.
            type: boolean
          forbiddenSysctls:
            description: Explicitly forbidden sysctls.
            type: array
            nullable: true
            items:
```

```
                    type: string
                  x-kubernetes-list-type: atomic
                seLinuxContext:
                  description: Strategy controlling SELinux labels.
                  type: object
                  nullable: true
                  properties:
                    type:
                      description: Strategy type for SELinux context.
                      type: string
                    seLinuxOptions:
                      description: Fixed SELinux options required by Must
RunAs.
                      type: object
                      properties:
                        user:
                          type: string
                        role:
                          type: string
                        type:
                          type: string
                        level:
                          type: string
                readOnlyRootFilesystem:
                  description: Forces readOnlyRootFilesystem when set to
true.
                  type: boolean
                fsGroup:
                  description: Strategy controlling fsGroup.
                  type: object
                  nullable: true
                  properties:
                    type:
                      type: string
                    ranges:
                      type: array
                      items:
                        type: object
                        properties:
                          min:
                            type: integer
                            format: int64
                          max:
                            type: integer
```

```
                format: int64
            x-kubernetes-list-type: atomic
        supplementalGroups:
          description: Strategy controlling supplemental groups.
          type: object
          nullable: true
          properties:
            type:
              type: string
            ranges:
              type: array
              items:
                type: object
                properties:
                  min:
                    type: integer
                    format: int64
                  max:
                    type: integer
                    format: int64
            x-kubernetes-list-type: atomic
        userNamespaceLevel:
          description: Controls host user namespace usage.
          type: string
          default: AllowHostLevel
          enum:
            - AllowHostLevel
            - RequirePodLevel
        defaultAddCapabilities:
          description: Capabilities added by default unless expli
citly dropped.
          type: array
          nullable: true
          items:
            type: string
          x-kubernetes-list-type: atomic
        allowedUnsafeSysctls:
          description: Explicitly allowed unsafe sysctls.
          type: array
          nullable: true
          items:
            type: string
          x-kubernetes-list-type: atomic
        allowedFlexVolumes:
```

```
                description: Allowed flex volume drivers.
                type: array
                nullable: true
                items:
                  type: object
                  required:
                    - driver
                  properties:
                    driver:
                      type: string
                x-kubernetes-list-type: atomic
              volumes:
                description: Allowed volume plugin types. '*' allows al
l.
                type: array
                nullable: true
                items:
                  type: string
                  enum:
                    - '*'
                    - none
                    - hostPath
                    - emptyDir
                    - gcePersistentDisk
                    - awsElasticBlockStore
                    - gitRepo
                    - secret
                    - nfs
                    - iscsi
                    - glusterfs
                    - persistentVolumeClaim
                    - rbd
                    - flexVolume
                    - cinder
                    - cephfs
                    - flocker
                    - downwardAPI
                    - fc
                    - azureFile
                    - configMap
                    - vsphereVolume
                    - quobyte
                    - azureDisk
                    - photonPersistentDisk
```

```
                  photonPersistentDisk
                - projected
                - portworxVolume
                - scaleIO
                - storageos
                - csi
                - ephemeral
                - image
              x-kubernetes-list-type: atomic
            allowHostPID:
              description: Determines if host PID is allowed.
              type: boolean
            allowHostNetwork:
              description: Determines if hostNetwork is allowed.
              type: boolean
            allowPrivilegeEscalation:
              description: Determines if privilege escalation can be
requested.
              type: boolean
              nullable: true
            defaultAllowPrivilegeEscalation:
              description: Default for allowPrivilegeEscalation when
container omits it.
              type: boolean
              nullable: true
            allowedCapabilities:
              description: Capabilities that may be added.
              type: array
              nullable: true
              items:
                type: string
              x-kubernetes-list-type: atomic
          x-kubernetes-validations:
            - rule: "!has(self.runAsUser) || self.runAsUser.type !=
'MustRunAs' || has(self.runAsUser.uid)"
              message: "runAsUser.uid is required when runAsUser.type
is MustRunAs."
            - rule: "!has(self.runAsUser) || self.runAsUser.type ==
'MustRunAs' || !has(self.runAsUser.uid)"
              message: "runAsUser.uid is only allowed when runAsUser.
type is MustRunAs."
            - rule: "!has(self.runAsUser) || self.runAsUser.type !=
'MustRunAsRange' || (has(self.runAsUser.uidRangeMin) && has(self.runAsUse
r.uidRangeMax))"
              message: "uidRangeMin and uidRangeMax are required when
```

```
                  message: "uidRangeMin and uidRangeMax are required when
runAsUser.type is MustRunAsRange."
                - rule: "!has(self.runAsUser) || self.runAsUser.type ==
'MustRunAsRange' || (!has(self.runAsUser.uidRangeMin) && !has(self.runAsU
ser.uidRangeMax))"
                  message: "uidRangeMin and uidRangeMax are only allowed
when runAsUser.type is MustRunAsRange."
                - rule: "!has(self.runAsUser) || !has(self.runAsUser.uidR
angeMin) || !has(self.runAsUser.uidRangeMax) || self.runAsUser.uidRangeMi
n <= self.runAsUser.uidRangeMax"
                  message: "uidRangeMin must be less than or equal to uid
RangeMax."
      additionalPrinterColumns:
        - name: Priv
          type: string
          description: Determines if privileged containers are allowed
          jsonPath: .spec.allowPrivilegedContainer
        - name: Caps
          type: string
          description: Allowed capabilities
          jsonPath: .spec.allowedCapabilities
        - name: SELinux
          type: string
          description: SELinux strategy
          jsonPath: .spec.seLinuxContext.type
        - name: RunAsUser
          type: string
          description: RunAsUser strategy
          jsonPath: .spec.runAsUser.type
        - name: FSGroup
          type: string
          description: FSGroup strategy
          jsonPath: .spec.fsGroup.type
        - name: SupGroup
          type: string
          description: SupplementalGroups strategy
          jsonPath: .spec.supplementalGroups.type
        - name: Priority
          type: string
          description: SCC sort priority
          jsonPath: .spec.priority
        - name: Score
          type: string
          description: Secondary restrictive score
```

```
        jsonPath: .spec.restrictiveScore
      - name: ReadOnlyRootFS
        type: string
        description: Force read-only root filesystem
        jsonPath: .spec.readOnlyRootFilesystem
      - name: Volumes
        type: string
        description: Allowed volume plugins
        jsonPath: .spec.volumes
  conversion:
    strategy: None
```

Apply it and wait for the CRD to be `Established` before continuing:

```
kubectl apply -f scc-crd.yaml
kubectl wait --for=condition=Established --timeout=120s \
  crd/securitycontextconstraints.security.alauda.io
```

Step 1.2 — Install the 13 built-in SCC profiles

Save the following manifest as `scc-profiles.yaml`. It defines 13 SCC profiles modelled on the OpenShift built-in set, ordered from most restrictive (`restrictiveScore: 100`) to least restrictive (`restrictiveScore: 0`). Higher `restrictiveScore` is preferred by the auto-pick policy when multiple SCCs are granted to the same subject.

> **Tip**
>
> You do not have to install every profile. Trim this manifest to the subset your platform offers — but you must keep at least one profile available to each subject, otherwise their Pods will be rejected at admission.

```yaml
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: restricted-v2
spec:
  priority: 0
  restrictiveScore: 100
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: false
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1
    uidRangeMax: 2147483647
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities:
    - NET_BIND_SERVICE
  requiredDropCapabilities:
    - ALL
  defaultAddCapabilities: []
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  seccompProfiles:
    - runtime/default
  readOnlyRootFilesystem: false
---
```

```yaml
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: restricted-v3
spec:
  priority: 0
  restrictiveScore: 100
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: false
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1000
    uidRangeMax: 65534
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
    ranges:
      - min: 1000
        max: 65534
  supplementalGroups:
    type: MustRunAs
    ranges:
      - min: 1000
        max: 65534
  userNamespaceLevel: RequirePodLevel
  allowedCapabilities:
    - NET_BIND_SERVICE
  requiredDropCapabilities:
    - ALL
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
```

```
    - secret
  seccompProfiles:
    - runtime/default
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: restricted
spec:
  priority: 0
  restrictiveScore: 98
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1
    uidRangeMax: 2147483647
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - KILL
    - MKNOD
    - SETUID
    - SETGID
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
```

```
    - secret
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: nonroot-v2
spec:
  priority: 0
  restrictiveScore: 95
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: false
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsNonRoot
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities:
    - NET_BIND_SERVICE
  requiredDropCapabilities:
    - ALL
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  seccompProfiles:
    - runtime/default
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
```

```yaml
metadata:
  name: nonroot
spec:
  priority: 0
  restrictiveScore: 92
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsNonRoot
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - KILL
    - MKNOD
    - SETUID
    - SETGID
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: hostnetwork-v2
spec:
  priority: 0
```

```
  restrictiveScore: 70
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: false
  allowHostNetwork: true
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: true
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1
    uidRangeMax: 2147483647
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
  supplementalGroups:
    type: MustRunAs
  allowedCapabilities:
    - NET_BIND_SERVICE
  requiredDropCapabilities:
    - ALL
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  seccompProfiles:
    - runtime/default
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: hostnetwork
spec:
  priority: 0
  restrictiveScore: 68
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
```

```
  allowPrivilegeEscalation: true
  allowHostNetwork: true
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: true
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1
    uidRangeMax: 2147483647
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
  supplementalGroups:
    type: MustRunAs
  allowedCapabilities: []
  requiredDropCapabilities:
    - KILL
    - MKNOD
    - SETUID
    - SETGID
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: anyuid
spec:
  priority: 10
  restrictiveScore: 60
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
```

```
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: RunAsAny
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - MKNOD
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: nested-container
spec:
  priority: 0
  restrictiveScore: 58
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: false
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 0
    uidRangeMax: 65534
```

```
  seLinuxContext:
    type: MustRunAs
    seLinuxOptions:
      type: container_engine_t
  fsGroup:
    type: MustRunAs
    ranges:
      - min: 0
        max: 65534
  supplementalGroups:
    type: MustRunAs
    ranges:
      - min: 0
        max: 65534
  userNamespaceLevel: RequirePodLevel
  allowedCapabilities:
    - SETUID
    - SETGID
  requiredDropCapabilities: []
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - image
    - persistentVolumeClaim
    - projected
    - secret
  seccompProfiles:
    - '*'
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: hostmount-anyuid
spec:
  priority: 0
  restrictiveScore: 55
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
```

```
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: true
  runAsUser:
    type: RunAsAny
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - MKNOD
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - hostPath
    - image
    - nfs
    - persistentVolumeClaim
    - projected
    - secret
  readOnlyRootFilesystem: false
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: hostmount-anyuid-v2
spec:
  priority: 0
  restrictiveScore: 50
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: false
  allowHostPID: false
  allowHostIPC: false
  allowHostPorts: false
  allowHostDirVolumePlugin: true
  runAsUser:
    type: RunAsAny
```

```
  seLinuxContext:
    type: RunAsAny
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - MKNOD
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - hostPath
    - image
    - nfs
    - persistentVolumeClaim
    - projected
    - secret
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: hostaccess
spec:
  priority: 0
  restrictiveScore: 40
  allowPrivilegedContainer: false
  allowPrivilegeEscalation: true
  allowHostNetwork: true
  allowHostPID: true
  allowHostIPC: true
  allowHostPorts: true
  allowHostDirVolumePlugin: true
  runAsUser:
    type: MustRunAsRange
    uidRangeMin: 1
    uidRangeMax: 2147483647
  seLinuxContext:
    type: MustRunAs
  fsGroup:
    type: MustRunAs
```

```
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities: []
  requiredDropCapabilities:
    - KILL
    - MKNOD
    - SETUID
    - SETGID
  volumes:
    - configMap
    - csi
    - downwardAPI
    - emptyDir
    - ephemeral
    - hostPath
    - image
    - persistentVolumeClaim
    - projected
    - secret
---
apiVersion: security.alauda.io/v1alpha1
kind: SecurityContextConstraints
metadata:
  name: privileged
spec:
  priority: 0
  restrictiveScore: 0
  allowPrivilegedContainer: true
  allowPrivilegeEscalation: true
  allowHostNetwork: true
  allowHostPID: true
  allowHostIPC: true
  allowHostPorts: true
  allowHostDirVolumePlugin: true
  runAsUser:
    type: RunAsAny
  seLinuxContext:
    type: RunAsAny
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  allowedCapabilities:
    - '*'
```

```
  requiredDropCapabilities: []
  volumes:
    - '*'
  seccompProfiles:
    - '*'
  allowedUnsafeSysctls:
    - '*'
```

Apply the profiles:

```
kubectl apply -f scc-profiles.yaml
kubectl get scc
```

You should see all 13 profiles listed with their `Priority` and `Score` columns populated (along with other SCC columns such as `Priv`, `RunAsUser`, and `Volumes`).

Step 1.3 — Install GlobalContextEntries, Kyverno reader RBAC, and admission policies

This step installs three things at once:

1. **GlobalContextEntries (GCE)** — five in-memory caches that Kyverno uses to look up SCC profiles, ClusterRoles, ClusterRoleBindings, RoleBindings, and Roles during admission, without making API calls per request.
2. **Reader RBAC** — a ClusterRole granting Kyverno's three service accounts read access to the SCC CRD, the four RBAC resources above, and the Pod / `pods/ephemeralcontainers` resources that the policies match.
3. **Two admission policies** — one `MutatingPolicy` that fills in defaults from the chosen SCC, and one `ValidatingPolicy` that rejects Pods that no granted SCC accepts.

> **Warning**
>
> The two policies contain the CEL logic that drives SCC selection and validation. You do **not** need to read or understand the CEL to use this engine — apply the manifests as-is. The expressions are long because they replicate the OpenShift SCC admission algorithm field-by-field.

Save the following as `scc-gce.yaml` and apply it:

```yaml
apiVersion: kyverno.io/v2alpha1
kind: GlobalContextEntry
metadata:
  name: scc-profiles
spec:
  kubernetesResource:
    group: security.alauda.io
    version: v1alpha1
    resource: securitycontextconstraints
  projections:
    - name: items
      jmesPath: "@"
---
apiVersion: kyverno.io/v2alpha1
kind: GlobalContextEntry
metadata:
  name: scc-clusterroles
spec:
  kubernetesResource:
    group: rbac.authorization.k8s.io
    version: v1
    resource: clusterroles
  projections:
    - name: items
      jmesPath: "@"
---
apiVersion: kyverno.io/v2alpha1
kind: GlobalContextEntry
metadata:
  name: scc-clusterrolebindings
spec:
  kubernetesResource:
    group: rbac.authorization.k8s.io
    version: v1
    resource: clusterrolebindings
  projections:
    - name: items
      jmesPath: "@"
---
apiVersion: kyverno.io/v2alpha1
kind: GlobalContextEntry
metadata:
  name: scc-rolebindings
```

```
  spec:
    kubernetesResource:
      group: rbac.authorization.k8s.io
      version: v1
      resource: rolebindings
    projections:
      - name: items
        jmesPath: "@"
---
apiVersion: kyverno.io/v2alpha1
kind: GlobalContextEntry
metadata:
  name: scc-roles
spec:
  kubernetesResource:
    group: rbac.authorization.k8s.io
    version: v1
    resource: roles
  projections:
    - name: items
      jmesPath: "@"
```

Save the following as `scc-reader-rbac.yaml` and apply it. The `pods` and `pods/ephemeralcontainers` read permissions are required because Kyverno checks read access to every matched resource during the policy-readiness gate (`RBACPermissionsGranted`); without them, the mutating policy stays `NotReady`.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kyverno-scc-reader
rules:
  - apiGroups:
      - security.alauda.io
    resources:
      - securitycontextconstraints
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - rbac.authorization.k8s.io
    resources:
      - clusterroles
      - clusterrolebindings
      - rolebindings
      - roles
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - pods
      - pods/ephemeralcontainers
    verbs:
      - get
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kyverno-scc-reader
subjects:
  - kind: ServiceAccount
    name: kyverno-admission-controller
    namespace: kyverno
  - kind: ServiceAccount
```

```
    name: kyverno-background-controller
    namespace: kyverno
  - kind: ServiceAccount
    name: kyverno-reports-controller
    namespace: kyverno
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kyverno-scc-reader
```

Save the following as `scc-auto-pick.yaml`. This is the `ValidatingPolicy` that rejects Pods which no granted SCC accepts.

> **Warning**
>
> The example below is configured with `validationActions: [Deny]`. On an existing cluster, change it to `validationActions: [Warn]` before the first apply, then switch it back to `Deny` after you have reviewed warnings and created the required SCC bindings. See Step 1.4 for the rollout process.

```yaml
apiVersion: policies.kyverno.io/v1alpha1
kind: ValidatingPolicy
metadata:
  name: scc-auto-pick
  labels:
    reports.kyverno.io/disabled: "true"
  annotations:
    policies.kyverno.io/title: SCC Auto-Pick (CEL, CRD + RBAC)
    pod-policies.kyverno.io/autogen-controllers: "none"
spec:
  autogen:
    podControllers:
      controllers: []
    validatingAdmissionPolicy:
      enabled: false
  evaluation:
    admission:
      enabled: true
    background:
      enabled: false
  failurePolicy: Fail
  validationActions:
    - Deny
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["pods"]
  matchConditions:
    - name: skip-system-ns
      expression: |
        !(request.namespace.startsWith('kube-') ||
          request.namespace.startsWith('cpaas-') ||
          request.namespace.startsWith('alauda-') ||
          request.namespace == 'kyverno' ||
          request.namespace == 'cattle-system' ||
          request.namespace == 'operators' ||
          request.namespace == 'default')

  variables:
    - name: containers
      expression: |
```

```
        object.spec.containers + object.spec.?initContainers.orValue([])
+
        object.spec.?ephemeralContainers.orValue([])

    - name: required
      expression: object.metadata.?annotations[?'alauda.io/required-sc
c'].orValue('')

    - name: profiles
      expression: |
        cel.bind(items, globalContext.Get('scc-profiles', 'items'),
          items == null ? [] : items)

    - name: subjectMatches
      expression: |
        [
          {'kind':'ServiceAccount',
           'name': string(object.spec.?serviceAccountName.orValue('defaul
t')),
           'namespace': string(request.namespace)},
          {'kind':'Group', 'name':'system:serviceaccounts'},
          {'kind':'Group', 'name':'system:serviceaccounts:'+request.names
pace},
          {'kind':'Group', 'name':'system:authenticated'},
          {'kind':'User',  'name': request.userInfo.username}
        ]
        + request.userInfo.groups.map(g, {'kind':'Group','name': g})

    - name: rolebindings
      expression: |
        cel.bind(rbs, globalContext.Get('scc-rolebindings','items'),
          rbs == null ? [] : rbs)

    - name: matchedClusterRoleRefsFromCRB
      expression: |
        cel.bind(crbs, globalContext.Get('scc-clusterrolebindings','item
s'),
          crbs == null ? [] : crbs)
        .filter(b, b.?roleRef.?kind.orValue('') == 'ClusterRole'
               && b.?subjects.orValue([]).exists(s,
            variables.subjectMatches.exists(m,
              s.kind == m.kind && s.name == m.name &&
              (s.kind != 'ServiceAccount' ||
               s.?namespace.orValue('') == m.?namespace.orValue(''))))))
```

```
        .map(b, b.roleRef.name)

    - name: matchedClusterRoleRefsFromRB
      expression: |
        variables.rolebindings
          .filter(b, b.?metadata.?namespace.orValue('') == request.namesp
ace
                  && b.?roleRef.?kind.orValue('') == 'ClusterRole'
                  && b.?subjects.orValue([]).exists(s,
              variables.subjectMatches.exists(m,
                s.kind == m.kind && s.name == m.name &&
                (s.kind != 'ServiceAccount' ||
                 s.?namespace.orValue('') == m.?namespace.orValue(''))))))
          .map(b, b.roleRef.name)

    - name: matchedRoleRefsFromRB
      expression: |
        variables.rolebindings
          .filter(b, b.?metadata.?namespace.orValue('') == request.namesp
ace
                  && b.?roleRef.?kind.orValue('') == 'Role'
                  && b.?subjects.orValue([]).exists(s,
              variables.subjectMatches.exists(m,
                s.kind == m.kind && s.name == m.name &&
                (s.kind != 'ServiceAccount' ||
                 s.?namespace.orValue('') == m.?namespace.orValue(''))))))
          .map(b, b.roleRef.name)

    - name: matchedClusterRoleRefs
      expression: |
        variables.matchedClusterRoleRefsFromCRB + variables.matchedCluste
rRoleRefsFromRB

    - name: allSccNames
      expression: |
        variables.profiles.map(p, p.metadata.name)

    - name: assignedFromClusterRoles
      expression: |
        cel.bind(crs, globalContext.Get('scc-clusterroles','items'),
          crs == null ? [] : crs)
          .filter(r, variables.matchedClusterRoleRefs.exists(n, n == r.me
tadata.name))
          .map(r, r.?rules.orValue([])
```

```
            .filter(ru,
              ru.?apiGroups.orValue([]).exists(g, g == 'security.alauda.i
o' || g == '*') &&
              ru.?resources.orValue([]).exists(x, x == 'securitycontextco
nstraints' || x == '*') &&
              ru.?verbs.orValue([]).exists(v, v == 'use' || v == '*'))
            .map(ru,
              ru.?resourceNames.orValue([]).size() == 0
                ? variables.allSccNames
                : ru.resourceNames)
          )
          .flatten()
          .flatten()

    - name: assignedFromRoles
      expression: |
        cel.bind(roles, globalContext.Get('scc-roles','items'),
          roles == null ? [] : roles)
          .filter(r,
            r.?metadata.?namespace.orValue('') == request.namespace
            && variables.matchedRoleRefsFromRB.exists(n, n == r.metadata.
name))
          .map(r, r.?rules.orValue([])
            .filter(ru,
              ru.?apiGroups.orValue([]).exists(g, g == 'security.alauda.i
o' || g == '*') &&
              ru.?resources.orValue([]).exists(x, x == 'securitycontextco
nstraints' || x == '*') &&
              ru.?verbs.orValue([]).exists(v, v == 'use' || v == '*'))
            .map(ru,
              ru.?resourceNames.orValue([]).size() == 0
                ? variables.allSccNames
                : ru.resourceNames)
          )
          .flatten()
          .flatten()

    - name: assigned
      expression: |
        (variables.assignedFromClusterRoles + variables.assignedFromRole
s)
          .filter(n, variables.allSccNames.exists(s, s == n))

    - name: safeSysctls
```

```
      expression: |
        ['kernel.shm_rmid_forced',
         'net.ipv4.ip_local_port_range',
         'net.ipv4.ip_unprivileged_port_start',
         'net.ipv4.tcp_syncookies',
         'net.ipv4.ping_group_range']

    - name: vtypes
      expression: |
        ['hostPath','emptyDir','gcePersistentDisk','awsElasticBlockStor
e','gitRepo',
         'secret','nfs','iscsi','glusterfs','persistentVolumeClaim','rb
d','flexVolume',
         'cinder','cephfs','flocker','downwardAPI','fc','azureFile','conf
igMap',
         'vsphereVolume','quobyte','azureDisk','photonPersistentDisk','pr
ojected',
         'portworxVolume','scaleIO','storageos','csi','ephemeral','imag
e']

    - name: ordered
      expression: |
        variables.assigned.sortBy(n,
          int(variables.profiles.filter(pr, pr.metadata.name == n)[?0].or
Value({}).?spec.?priority.orValue(0)) * -100000 +
          -int(variables.profiles.filter(pr, pr.metadata.name == n)[?0].o
rValue({}).?spec.?restrictiveScore.orValue(100))
        )
    - name: requiredExists
      expression: variables.required == '' || variables.profiles.exists(p
r, pr.metadata.name == variables.required)
    - name: requiredBound
      expression: variables.required == '' || variables.assigned.exists
(n, n == variables.required)
    - name: candidateNames
      expression: |
        variables.required != ''
          ? [variables.required]
          : variables.ordered

    - name: matched
      expression: |
        variables.candidateNames.exists(n,
          cel.bind(p, variables.profiles.filter(pr, pr.metadata.name ==
```

```
n)[?0].orValue({}).?spec.orValue({}),
            (p.?allowPrivilegedContainer.orValue(false)
              || !variables.containers.exists(c, c.?securityContext.?pri
vileged.orValue(false)))
          && (p.?allowPrivilegeEscalation.orValue(true)
              || !variables.containers.exists(c, c.?securityContext.?all
owPrivilegeEscalation.orValue(true)))
          && (p.?allowHostNetwork.orValue(false) || !object.spec.?hostNet
work.orValue(false))
          && (p.?allowHostPID.orValue(false)     || !object.spec.?hostPI
D.orValue(false))
          && (p.?allowHostIPC.orValue(false)     || !object.spec.?hostIP
C.orValue(false))
          && (p.?allowHostDirVolumePlugin.orValue(false)
              || !object.spec.?volumes.orValue([]).exists(v, has(v.hostP
ath)))
          && (
               p.?runAsUser.?type.orValue('RunAsAny') == 'RunAsAny'
               || (
                    (p.?runAsUser.?type.orValue('RunAsAny') in ['MustRunA
sNonRoot','MustRunAsNonRootOrSystem'])
                    && !variables.containers.exists(c, c.?securityContex
t.?runAsUser.orValue(
                         object.spec.?securityContext.?runAsUser.orValue
(1)) == 0)
                )
               || (
                    p.?runAsUser.?type.orValue('RunAsAny') == 'MustRunAs'
                    && variables.containers.all(c, c.?securityContext.?ru
nAsUser.orValue(
                         object.spec.?securityContext.?runAsUser.orValue
(1))
                         == p.?runAsUser.?uid.orValue(-1))
                )
               || (
                    p.?runAsUser.?type.orValue('RunAsAny') == 'MustRunAsR
ange'
                    && variables.containers.all(c,
                         c.?securityContext.?runAsUser.orValue(
                           object.spec.?securityContext.?runAsUser.orValu
e(1))
                           >= p.?runAsUser.?uidRangeMin.orValue(1)
                         && c.?securityContext.?runAsUser.orValue(
                              object.spec.?securityContext.?runAsUser.orV
```

```
                                object.spec.securityContext.runAsUser.v
alue(1))
                            <= p.?runAsUser.?uidRangeMax.orValue(214748364
7))
                    )
                )
            && (p.?allowedCapabilities.orValue([]).exists(t, t == '*')
                || variables.containers.all(c,
                    c.?securityContext.?capabilities.?add.orValue([]).all
(cap,
                      p.?allowedCapabilities.orValue([]).exists(a, a == c
ap))))
            && (p.?requiredDropCapabilities.orValue([]).size() == 0
                || variables.containers.all(c,
                    p.?requiredDropCapabilities.orValue([]).all(req,
                      c.?securityContext.?capabilities.?drop.orValue([]).
exists(d, d == req || d == 'ALL'))))
            && (p.?volumes.orValue(['*']).exists(t, t == '*')
                || object.spec.?volumes.orValue([]).all(v,
                    variables.vtypes.filter(t, v[?t].hasValue()).all(t,
                      p.?volumes.orValue([]).exists(a, a == t))))
            && (p.?allowHostPorts.orValue(false)
                || variables.containers.all(c,
                    c.?ports.orValue([]).all(port, port.?hostPort.orValue
(0) == 0)))
            && (p.?allowedUnsafeSysctls.orValue([]).exists(t, t == '*')
                || object.spec.?securityContext.?sysctls.orValue([]).all
(s,
                    variables.safeSysctls.exists(safe, safe == s.name)
                    || p.?allowedUnsafeSysctls.orValue([]).exists(a, a ==
s.name)))
            && (!p.?readOnlyRootFilesystem.orValue(false)
                || variables.containers.all(c, c.?securityContext.?readOnl
yRootFilesystem.orValue(false) == true))
            && (p.?seccompProfiles.orValue([]).size() == 0
                || p.?seccompProfiles.orValue([]).exists(t, t == '*')
                || variables.containers.all(c,
                    p.?seccompProfiles.orValue([]).exists(a,
                      (c.?securityContext.?seccompProfile.?type.orValue(
                         object.spec.?securityContext.?seccompProfile.?ty
pe.orValue('')) == 'RuntimeDefault'
                         && a == 'runtime/default')
                      || (c.?securityContext.?seccompProfile.?type.orValu
e(
                                object.spec.?securityContext.?seccompProfile.?ty
```

```
                        object.spec.?securityContext.?seccompProfile.?ty
pe.orValue('')) == 'Unconfined'
                        && a == 'unconfined')
                    || (c.?securityContext.?seccompProfile.?type.orValu
e(
                        object.spec.?securityContext.?seccompProfile.?ty
pe.orValue('')) == 'Localhost'
                        && a == 'localhost/' + c.?securityContext.?secco
mpProfile.?localhostProfile.orValue(
                            object.spec.?securityContext.?seccompProfil
e.?localhostProfile.orValue(''))))))
          && (p.?allowedFlexVolumes.orValue([]).size() == 0
               || object.spec.?volumes.orValue([]).filter(v, v.?flexVolum
e.hasValue()).all(v,
                    p.?allowedFlexVolumes.orValue([]).exists(d, d.?drive
r.orValue('') == v.flexVolume.driver)))
          )
        )

  validations:
    - expression: variables.requiredExists
      message: "required-scc does not exist"
      messageExpression: |
        "required SCC '" + variables.required + "' not found in scc-profi
les"
    - expression: variables.requiredBound
      message: "required-scc is not bound to ServiceAccount"
      messageExpression: |
        "required SCC '" + variables.required +
        "' is not bound to ServiceAccount '" +
        object.spec.?serviceAccountName.orValue('default') +
        "' in namespace '" + request.namespace + "'"
    - expression: variables.matched
      message: "Pod violates all SCCs assigned to its ServiceAccount"
      messageExpression: |
        variables.required != ''
        ? ("Pod " + object.metadata.name +
           " does not satisfy required SCC '" + variables.required + "'")
        : ("Pod " + object.metadata.name +
           " does not satisfy any SCC profile assigned to ServiceAccount
'" +
           object.spec.?serviceAccountName.orValue('default') +
           "' in namespace '" + request.namespace +
           "' (candidates: " + variables.ordered.join(",") + ")")
```

Save the following as `scc-fill-defaults.yaml` and apply it. This is the `MutatingPolicy` that records the selected SCC on the Pod (`alauda.io/scc` annotation) and fills in `runAsUser`, `seccompProfile`, and `allowPrivilegeEscalation` defaults inherited from that SCC.

```
apiVersion: policies.kyverno.io/v1alpha1
kind: MutatingPolicy
metadata:
  name: scc-fill-defaults
  labels:
    reports.kyverno.io/disabled: "true"
  annotations:
    policies.kyverno.io/title: SCC default value filler (CRD + RBAC, expl
icit-wins)
    pod-policies.kyverno.io/autogen-controllers: "none"
spec:
  autogen:
    podControllers:
      controllers: []
  evaluation:
    admission:
      enabled: true
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["CREATE"]
        resources: ["pods"]
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["UPDATE"]
        resources: ["pods/ephemeralcontainers"]
  matchConditions:
    - name: skip-system-ns
      expression: |
        !(request.namespace.startsWith('kube-') ||
          request.namespace.startsWith('cpaas-') ||
          request.namespace.startsWith('alauda-') ||
          request.namespace == 'kyverno' ||
          request.namespace == 'cattle-system' ||
          request.namespace == 'operators' ||
          request.namespace == 'default')

  variables:
    - name: containers
      expression: |
        object.spec.containers + object.spec.?initContainers.orValue([])
```

```
+
        object.spec.?ephemeralContainers.orValue([])
    - name: required
      expression: object.metadata.?annotations[?'alauda.io/required-sc
c'].orValue('')

    - name: profiles
      expression: |
        cel.bind(items, globalContext.Get('scc-profiles', 'items'),
          items == null ? [] : items)

    - name: subjectMatches
      expression: |
        [
          {'kind':'ServiceAccount',
           'name': string(object.spec.?serviceAccountName.orValue('defaul
t')),
           'namespace': string(object.metadata.namespace)},
          {'kind':'Group', 'name':'system:serviceaccounts'},
          {'kind':'Group', 'name':'system:serviceaccounts:'+object.metada
ta.namespace},
          {'kind':'Group', 'name':'system:authenticated'},
          {'kind':'User',  'name': request.userInfo.username}
        ]
        + request.userInfo.groups.map(g, {'kind':'Group','name': g})
    - name: rolebindings
      expression: |
        cel.bind(rbs, globalContext.Get('scc-rolebindings','items'),
          rbs == null ? [] : rbs)
    - name: matchedClusterRoleRefsFromCRB
      expression: |
        cel.bind(crbs, globalContext.Get('scc-clusterrolebindings','item
s'),
          crbs == null ? [] : crbs)
        .filter(b, b.?roleRef.?kind.orValue('') == 'ClusterRole'
               && b.?subjects.orValue([]).exists(s,
            variables.subjectMatches.exists(m,
              s.kind == m.kind && s.name == m.name &&
              (s.kind != 'ServiceAccount' ||
               s.?namespace.orValue('') == m.?namespace.orValue(''))))))
        .map(b, b.roleRef.name)
    - name: matchedClusterRoleRefsFromRB
      expression: |
        variables.rolebindings
```

```
          .filter(b, b.?metadata.?namespace.orValue('') == object.metadat
a.namespace
                  && b.?roleRef.?kind.orValue('') == 'ClusterRole'
                  && b.?subjects.orValue([]).exists(s,
              variables.subjectMatches.exists(m,
                s.kind == m.kind && s.name == m.name &&
                (s.kind != 'ServiceAccount' ||
                 s.?namespace.orValue('') == m.?namespace.orValue(''))))))
          .map(b, b.roleRef.name)
    - name: matchedRoleRefsFromRB
      expression: |
        variables.rolebindings
          .filter(b, b.?metadata.?namespace.orValue('') == object.metadat
a.namespace
                  && b.?roleRef.?kind.orValue('') == 'Role'
                  && b.?subjects.orValue([]).exists(s,
              variables.subjectMatches.exists(m,
                s.kind == m.kind && s.name == m.name &&
                (s.kind != 'ServiceAccount' ||
                 s.?namespace.orValue('') == m.?namespace.orValue(''))))))
          .map(b, b.roleRef.name)
    - name: matchedClusterRoleRefs
      expression: |
        variables.matchedClusterRoleRefsFromCRB + variables.matchedCluste
rRoleRefsFromRB
    - name: allSccNames
      expression: |
        variables.profiles.map(p, p.metadata.name)
    - name: assignedFromClusterRoles
      expression: |
        cel.bind(crs, globalContext.Get('scc-clusterroles','items'),
          crs == null ? [] : crs)
          .filter(r, variables.matchedClusterRoleRefs.exists(n, n == r.me
tadata.name))
          .map(r, r.?rules.orValue([])
            .filter(ru,
              ru.?apiGroups.orValue([]).exists(g, g == 'security.alauda.i
o' || g == '*') &&
              ru.?resources.orValue([]).exists(x, x == 'securitycontextco
nstraints' || x == '*') &&
              ru.?verbs.orValue([]).exists(v, v == 'use' || v == '*'))
            .map(ru,
              ru.?resourceNames.orValue([]).size() == 0
                ? variables.allSccNames
```

```
                : ru.resourceNames)
          )
          .flatten()
          .flatten()

    - name: assignedFromRoles
      expression: |
        cel.bind(roles, globalContext.Get('scc-roles','items'),
          roles == null ? [] : roles)
          .filter(r,
            r.?metadata.?namespace.orValue('') == object.metadata.namespa
ce
            && variables.matchedRoleRefsFromRB.exists(n, n == r.metadata.
name))
          .map(r, r.?rules.orValue([])
            .filter(ru,
              ru.?apiGroups.orValue([]).exists(g, g == 'security.alauda.i
o' || g == '*') &&
              ru.?resources.orValue([]).exists(x, x == 'securitycontextco
nstraints' || x == '*') &&
              ru.?verbs.orValue([]).exists(v, v == 'use' || v == '*'))
            .map(ru,
              ru.?resourceNames.orValue([]).size() == 0
                ? variables.allSccNames
                : ru.resourceNames)
          )
          .flatten()
          .flatten()

    - name: assigned
      expression: |
        (variables.assignedFromClusterRoles + variables.assignedFromRole
s)
          .filter(n, variables.allSccNames.exists(s, s == n))

    - name: safeSysctls
      expression: |
        ['kernel.shm_rmid_forced',
         'net.ipv4.ip_local_port_range',
         'net.ipv4.ip_unprivileged_port_start',
         'net.ipv4.tcp_syncookies',
         'net.ipv4.ping_group_range']
    - name: vtypes
      expression: |
```

```
        ['hostPath','emptyDir','gcePersistentDisk','awsElasticBlockStor
e','gitRepo',
         'secret','nfs','iscsi','glusterfs','persistentVolumeClaim','rb
d','flexVolume',
         'cinder','cephfs','flocker','downwardAPI','fc','azureFile','conf
igMap',
         'vsphereVolume','quobyte','azureDisk','photonPersistentDisk','pr
ojected',
         'portworxVolume','scaleIO','storageos','csi','ephemeral','imag
e']

    - name: ordered
      expression: |
        variables.assigned.sortBy(n,
          int(variables.profiles.filter(pr, pr.metadata.name == n)[?0].or
Value({}).?spec.?priority.orValue(0)) * -100000 +
          -int(variables.profiles.filter(pr, pr.metadata.name == n)[?0].o
rValue({}).?spec.?restrictiveScore.orValue(100))
        )
    - name: requiredExists
      expression: variables.required == '' || variables.profiles.exists(p
r, pr.metadata.name == variables.required)
    - name: requiredBound
      expression: variables.required == '' || variables.assigned.exists
(n, n == variables.required)
    - name: candidateNames
      expression: |
        variables.required != ''
          ? ((variables.requiredExists && variables.requiredBound) ? [var
iables.required] : [])
          : variables.ordered
    - name: isEphemeralSubresource
      expression: request.operation == 'UPDATE'
    - name: annotatedSelectedName
      expression: object.metadata.?annotations[?'alauda.io/scc'].orValue
('')

    - name: matchedNames
      expression: |
        variables.candidateNames.filter(n,
          cel.bind(p, variables.profiles.filter(pr, pr.metadata.name ==
n)[?0].orValue({}).?spec.orValue({}),
            cel.bind(defaultPE, p.?defaultAllowPrivilegeEscalation.orValu
e(
```

```
`
                  p.?allowPrivilegeEscalation.orValue(true)),
              cel.bind(podRunAsUserForFill,
                    object.spec.?securityContext.?runAsUser.orValue(
                      (p.?runAsUser.?type.orValue('') == 'MustRunAs' &&
p.?runAsUser.?uid.hasValue())
                        ? p.?runAsUser.?uid.orValue(1)
                        : 1),
                cel.bind(seccompFirstForFill,
                      p.?seccompProfiles.orValue([]).filter(s, s != '' &&
s != '*')[?0].orValue(''),
                  cel.bind(fillSeccompType,
                        seccompFirstForFill == 'runtime/default' ? 'Runti
meDefault' :
                        seccompFirstForFill.startsWith('localhost/') ? 'L
ocalhost' : '',
                    cel.bind(fillSeccompLocalhost,
                          fillSeccompType == 'Localhost'
                            ? seccompFirstForFill.substring('localhost/'.
size()) : '',
                      cel.bind(needPodSeccompFillForMatch,
                            !object.spec.?securityContext.?seccompProfil
e.hasValue() &&
                            object.spec.containers.all(c, !c.?securityCon
text.?seccompProfile.hasValue()) &&
                            object.spec.?initContainers.orValue([]).all
(c, !c.?securityContext.?seccompProfile.hasValue()),
                        (p.?allowPrivilegedContainer.orValue(false)
                          || !variables.containers.exists(c, c.?securityC
ontext.?privileged.orValue(false)))
                        && (p.?allowPrivilegeEscalation.orValue(true)
                            || !variables.containers.exists(c, c.?securit
yContext.?allowPrivilegeEscalation.orValue(defaultPE)))
                        && (p.?allowHostNetwork.orValue(false) || !objec
t.spec.?hostNetwork.orValue(false))
                        && (p.?allowHostPID.orValue(false)     || !objec
t.spec.?hostPID.orValue(false))
                        && (p.?allowHostIPC.orValue(false)     || !objec
t.spec.?hostIPC.orValue(false))
                        && (p.?allowHostDirVolumePlugin.orValue(false)
                            || !object.spec.?volumes.orValue([]).exists
(v, has(v.hostPath)))
                        && (
                            p.?runAsUser.?type.orValue('RunAsAny') == 'Ru
nAsAny'
```

```
RunAsAny
                            || (
                                (p.?runAsUser.?type.orValue('RunAsAny') i
n ['MustRunAsNonRoot','MustRunAsNonRootOrSystem'])
                                && !variables.containers.exists(c, c.?sec
urityContext.?runAsUser.orValue(
                                      podRunAsUserForFill) == 0)
                              )
                            || (
                                p.?runAsUser.?type.orValue('RunAsAny') ==
'MustRunAs'
                                && variables.containers.all(c, c.?securit
yContext.?runAsUser.orValue(
                                      podRunAsUserForFill) == p.?runAsUse
r.?uid.orValue(-1))
                              )
                            || (
                                p.?runAsUser.?type.orValue('RunAsAny') ==
'MustRunAsRange'
                                && variables.containers.all(c,
                                    c.?securityContext.?runAsUser.orValue
(podRunAsUserForFill)
                                      >= p.?runAsUser.?uidRangeMin.orValu
e(1)
                                    && c.?securityContext.?runAsUser.orVa
lue(podRunAsUserForFill)
                                      <= p.?runAsUser.?uidRangeMax.orValu
e(2147483647))
                              )
                          )
                        && (p.?allowedCapabilities.orValue([]).exists(t,
t == '*')
                            || variables.containers.all(c,
                                c.?securityContext.?capabilities.?add.orV
alue([]).all(cap,
                                  p.?allowedCapabilities.orValue([]).exis
ts(a, a == cap))))
                        && (p.?requiredDropCapabilities.orValue([]).size
() == 0
                            || variables.containers.all(c,
                                p.?requiredDropCapabilities.orValue([]).a
ll(req,
                                  c.?securityContext.?capabilities.?drop.
orValue([]).exists(d, d == req || d == 'ALL'))))
                        && (p.?volumes.orValue(['*']).exists(t, t == '*')
```

```
                        && (p.?volumes.orValue([]).exists(t, t ==
                            || object.spec.?volumes.orValue([]).all(v,
                                variables.vtypes.filter(t, v[?t].hasValue
()).all(t,
                                  p.?volumes.orValue([]).exists(a, a ==
t))))
                        && (p.?allowHostPorts.orValue(false)
                            || variables.containers.all(c,
                                c.?ports.orValue([]).all(port, port.?host
Port.orValue(0) == 0)))
                        && (p.?allowedUnsafeSysctls.orValue([]).exists(t,
t == '*')
                            || object.spec.?securityContext.?sysctls.orVa
lue([]).all(s,
                                variables.safeSysctls.exists(safe, safe =
= s.name)
                                || p.?allowedUnsafeSysctls.orValue([]).ex
ists(a, a == s.name)))
                        && (!p.?readOnlyRootFilesystem.orValue(false)
                            || variables.containers.all(c, c.?securityCon
text.?readOnlyRootFilesystem.orValue(false) == true))
                        && (p.?seccompProfiles.orValue([]).size() == 0
                            || p.?seccompProfiles.orValue([]).exists(t, t
== '*')
                            || variables.containers.all(c,
                                p.?seccompProfiles.orValue([]).exists(a,
                                  (c.?securityContext.?seccompProfile.?ty
pe.orValue(
                                      object.spec.?securityContext.?secco
mpProfile.?type.orValue(
                                        (needPodSeccompFillForMatch && fi
llSeccompType != '') ? fillSeccompType : '')) == 'RuntimeDefault'
                                    && a == 'runtime/default')
                                  || (c.?securityContext.?seccompProfil
e.?type.orValue(
                                      object.spec.?securityContext.?secco
mpProfile.?type.orValue(
                                        (needPodSeccompFillForMatch && fi
llSeccompType != '') ? fillSeccompType : '')) == 'Unconfined'
                                    && a == 'unconfined')
                                  || (c.?securityContext.?seccompProfil
e.?type.orValue(
                                      object.spec.?securityContext.?secco
mpProfile.?type.orValue(
```

```
                                        (needPodSeccompFillForMatch && fi
llSeccompType != '') ? fillSeccompType : '')) == 'Localhost'
                                    && a == 'localhost/' + c.?securityCon
text.?seccompProfile.?localhostProfile.orValue(
                                        object.spec.?securityContext.?sec
compProfile.?localhostProfile.orValue(
                                          (needPodSeccompFillForMatch &&
fillSeccompType == 'Localhost')
                                            ? fillSeccompLocalhost :
'')))))))
                        && (p.?allowedFlexVolumes.orValue([]).size() == 0
                            || object.spec.?volumes.orValue([]).filter(v,
v.?flexVolume.hasValue()).all(v,
                                p.?allowedFlexVolumes.orValue([]).exists
(d, d.?driver.orValue('') == v.flexVolume.driver)))
                      )
                    )
                  )
                )
              )
            )
          )
        )

    - name: selectedName
      expression: |
        variables.isEphemeralSubresource
          && variables.annotatedSelectedName != ''
          && variables.candidateNames.exists(n, n == variables.annotatedS
electedName)
          ? variables.annotatedSelectedName
          : variables.matchedNames[?0].orValue('')

    - name: selectedSpec
      expression: |
        variables.profiles.filter(pr, pr.metadata.name == variables.selec
tedName)[?0].orValue({}).?spec.orValue({})

    - name: defaultPE
      expression: |
        variables.selectedSpec.?defaultAllowPrivilegeEscalation.orValue(
          variables.selectedSpec.?allowPrivilegeEscalation.orValue(true))

    - name: seccompFirst
```

```
      expression: |
        variables.selectedSpec.?seccompProfiles.orValue([])
          .filter(s, s != '' && s != '*')[?0].orValue('')
    - name: defaultSeccompType
      expression: |
        variables.seccompFirst == 'runtime/default' ? 'RuntimeDefault' :
        variables.seccompFirst.startsWith('localhost/') ? 'Localhost' :
''
    - name: defaultSeccompLocalhostProfile
      expression: |
        variables.defaultSeccompType == 'Localhost'
          ? variables.seccompFirst.substring('localhost/'.size()) : ''
    - name: needPodSeccomp
      expression: |
        variables.selectedName != '' && variables.defaultSeccompType !=
'' &&
        !object.spec.?securityContext.?seccompProfile.hasValue() &&
        object.spec.containers.all(c, !c.?securityContext.?seccompProfil
e.hasValue()) &&
        object.spec.?initContainers.orValue([]).all(c, !c.?securityContex
t.?seccompProfile.hasValue())

    - name: hasLiteralUid
      expression: |
        variables.selectedName != '' &&
        variables.selectedSpec.?runAsUser.?type.orValue('') == 'MustRunA
s' &&
        variables.selectedSpec.?runAsUser.?uid.hasValue()
    - name: literalUid
      expression: |
        variables.hasLiteralUid ? variables.selectedSpec.?runAsUser.?uid.
orValue(-1) : -1
    - name: needPodRunAsUser
      expression: |
        variables.hasLiteralUid &&
        !object.spec.?securityContext.?runAsUser.hasValue()

  mutations:
    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (variables.isEphemeralSubresource || variables.selectedName ==
'') ? Object{} :
          Object{
```

```
            metadata: Object.metadata{
              annotations: {
                "alauda.io/scc": string(variables.selectedName)
              }
            }
          }

    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (variables.isEphemeralSubresource || !variables.needPodRunAsUse
r) ? Object{} :
          Object{
            spec: Object.spec{
              securityContext: Object.spec.securityContext{
                runAsUser: variables.literalUid
              }
            }
          }

    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (variables.isEphemeralSubresource || !variables.needPodSeccomp)
? Object{} :
          (variables.defaultSeccompType == 'Localhost') ?
          Object{
            spec: Object.spec{
              securityContext: Object.spec.securityContext{
                seccompProfile: Object.spec.securityContext.seccompProfil
e{
                  type: 'Localhost',
                  localhostProfile: variables.defaultSeccompLocalhostProf
ile
                }
              }
            }
          } :
          Object{
            spec: Object.spec{
              securityContext: Object.spec.securityContext{
                seccompProfile: Object.spec.securityContext.seccompProfil
e{
                  type: variables.defaultSeccompType
```

```
                }
              }
            }
          }

    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (variables.isEphemeralSubresource || variables.selectedName ==
'') ? Object{} :
          Object{
            spec: Object.spec{
              containers: object.spec.containers.map(c, Object.spec.conta
iners{
                name: c.name,
                securityContext: Object.spec.containers.securityContext{
                  allowPrivilegeEscalation:
                    c.?securityContext.?allowPrivilegeEscalation.hasValue
()
                      ? c.securityContext.allowPrivilegeEscalation
                      : variables.defaultPE
                }
              })
            }
          }

    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (variables.isEphemeralSubresource || variables.selectedName ==
'' || !object.spec.?initContainers.hasValue()) ? Object{} :
          Object{
            spec: Object.spec{
              initContainers: object.spec.initContainers.map(c, Object.sp
ec.initContainers{
                name: c.name,
                securityContext: Object.spec.initContainers.securityConte
xt{
                  allowPrivilegeEscalation:
                    c.?securityContext.?allowPrivilegeEscalation.hasValue
()
                      ? c.securityContext.allowPrivilegeEscalation
                      : variables.defaultPE
                }
```

```
            })
          }
        }

    - patchType: ApplyConfiguration
      applyConfiguration:
        expression: |
          (!variables.isEphemeralSubresource || variables.selectedName ==
'' || !object.spec.?ephemeralContainers.hasValue()) ? Object{} :
          Object{
            spec: Object.spec{
              ephemeralContainers: object.spec.ephemeralContainers.map(c,
Object.spec.ephemeralContainers{
                name: c.name,
                securityContext: Object.spec.ephemeralContainers.security
Context{
                  allowPrivilegeEscalation:
                    c.?securityContext.?allowPrivilegeEscalation.hasValue
()
                      ? c.securityContext.allowPrivilegeEscalation
                      : variables.defaultPE
                }
              })
            }
          }
```

Keep `reports.kyverno.io/disabled: "true"` on both policies. SCC selection depends on the admission request User, Groups, and ServiceAccount, while a background report scan does not have the equivalent request identity. The mutating policy also fills defaults only during admission. Excluding these admission-only policies from background reporting avoids misleading PolicyReports and unnecessary recompilation for every existing Pod.

Both policies skip the following namespaces by default: namespaces starting with `kube-`, `cpaas-`, or `alauda-`, plus `kyverno`, `cattle-system`, `operators`, and `default`. Adjust the `skip-system-ns` expression in both policies if your platform uses different system namespaces.

Step 1.4 — Roll out safely with Warn → Deny

The validating policy is delivered with `failurePolicy: Fail` and `validationActions: [Deny]`, which means it rejects non-compliant Pods immediately. On an existing cluster,

switching this on without preparation can break workloads whose ServiceAccounts have not yet been bound to any SCC.

Use a three-stage rollout:

1. **Warn before the first apply**. Before applying `scc-auto-pick.yaml` on an existing cluster, change `validationActions` to:

   ```
   validationActions:
     - Warn
   ```

   Apply the file. The policy now attaches a warning to every admission response that would have been rejected, but admits the Pod. Watch the Kyverno admission controller logs to collect the affected workloads:

   ```
   kubectl logs -n kyverno -l app.kubernetes.io/component=admission-contro
   ller \
     --tail=500 | grep -i 'scc-auto-pick'
   ```

2. **Fix**. For each warned workload, add or correct the RBAC binding so that its ServiceAccount can `use` an appropriate SCC (see Part 2). Confirm with:

   ```
   kubectl auth can-i use \
     securitycontextconstraints.security.alauda.io/<scc-name> \
     --as="system:serviceaccount:<namespace>:<sa-name>" -n <namespace>
   ```

3. **Deny**. Once warnings stop arriving for legitimate workloads, switch back to `Deny` and re-apply:

   ```
   validationActions:
     - Deny
   ```

Tip

If you need to exempt an entire namespace temporarily, you can either add it to the `skip-system-ns` expression in both policies or create a `PolicyException` resource. See **Learn More** below for the `PolicyException` pattern.

Step 1.5 — Verify the engine is ready

Run the following checks. All resources should be present and the two policies should be `READY=true`.

```
# 1. CRD is established and 13 profiles are loaded
kubectl get crd securitycontextconstraints.security.alauda.io
kubectl get scc

# 2. Five GCE caches exist
kubectl get globalcontextentry scc-profiles scc-clusterroles \
  scc-clusterrolebindings scc-rolebindings scc-roles

# 3. Two admission policies are ready
kubectl get validatingpolicy scc-auto-pick
kubectl get mutatingpolicy scc-fill-defaults

# 4. Reader RBAC is in place
kubectl get clusterrole kyverno-scc-reader
kubectl get clusterrolebinding kyverno-scc-reader
```

If `scc-fill-defaults` shows `READY=false`, the most common cause is missing read permission on `pods/ephemeralcontainers` — make sure Step 1.3's `kyverno-scc-reader` ClusterRole was applied in full.

Part 2: Authorize workloads to use SCCs

With the engine installed, no Pod is granted any SCC by default. Until an administrator creates an RBAC binding for a ServiceAccount (or User, or Group), Pods running under that subject in non-system namespaces will be rejected with the message `Pod violates all SCCs assigned to its ServiceAccount`.

Treat each SCC binding as a security authorization decision. Grant SCC binding privileges only to platform administrators or security administrators; ordinary application users and namespace owners should not be able to grant themselves higher Pod privileges.

Step 2.1 — Choose the right SCC profile

Match the workload's security needs against the table below. By default, the engine ranks granted SCCs by `priority` first and `restrictiveScore` second. Pick the least-privilege profile set the workload needs, and use `alauda.io/required-scc` when you must force one specific profile.

Workload characteristics	Recommended SCC
Stateless service, non-root, drops ALL capabilities, no host access	`restricted-v2`
Same as above, but needs to bind ports <1024	`restricted-v2` (the profile already allows `NET_BIND_SERVICE`)
Same as above, but uses a fixed UID range like 1000–65534 and user namespaces	`restricted-v3`
Service that runs as a non-root user but cannot drop all capabilities	`nonroot-v2` (drops ALL) or `nonroot` (older drop set)
Image that requires running as root (`USER root`)	`anyuid`
Ingress controller or other Pod that needs `hostNetwork` and host ports	`hostnetwork-v2` (drops ALL) or `hostnetwork` (older drop set)
Service that mounts `hostPath` for log / metrics collection, runs as non-root	`hostmount-anyuid`
Same as above, but does not require SELinux relabeling	`hostmount-anyuid-v2`
Diagnostic Pod that needs `hostNetwork`, `hostPID`, `hostIPC` and host paths	`hostaccess`

Workload characteristics	Recommended SCC
Container-in-container build sandbox using user namespaces	`nested-container`
Fully privileged workload (CNI, storage drivers, debug pods)	`privileged`

> **Warning**
>
> Always grant the least privilege required. A ServiceAccount bound to `privileged` can run any Pod, including those that escape the container boundary. Reserve `privileged` for infrastructure DaemonSets and avoid granting it to user workloads.

When an application manager requests SCC access, include:

- Namespace and ServiceAccount, for example `databases/postgres-sa`.
- Workload name and controller type, for example `StatefulSet/postgres`.
- The requested SCC or required capability, for example `anyuid` because the image runs as UID 0.
- Why a stricter SCC such as `restricted-v2` is not sufficient.
- Whether the workload must pin a specific SCC with `alauda.io/required-scc`.

Step 2.2 — Bind an SCC to a ServiceAccount

The most common administrator action is to bind an SCC to a workload ServiceAccount. Suppose you have an application running under `databases/postgres-sa` and the image runs as root (UID 0). You want this ServiceAccount to be allowed `anyuid`, while still keeping `restricted-v2` available for stricter workloads. In this root-UID example, `restricted-v2` does not match (`runAsUser.uidRangeMin: 1`), so admission selects `anyuid`. More generally, when a Pod satisfies both profiles, the default profile set in this guide prefers `anyuid` first because `anyuid` has higher `priority` than `restricted-v2` unless you adjust priorities or pin `alauda.io/required-scc`.

Save the following as `bind-postgres-sa.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scc-use-anyuid-restricted
  labels:
    rbac.alauda.io/scc-use: "true"
rules:
  - apiGroups: ["security.alauda.io"]
    resources: ["securitycontextconstraints"]
    resourceNames: ["anyuid", "restricted-v2"]
    verbs: ["use"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: postgres-sa-scc
  namespace: databases
  labels:
    rbac.alauda.io/scc-use: "true"
subjects:
  - kind: ServiceAccount
    name: postgres-sa
    namespace: databases
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: scc-use-anyuid-restricted
```

Apply:

```
kubectl apply -f bind-postgres-sa.yaml
```

The `rbac.alauda.io/scc-use=true` label is optional. It does not affect SCC selection, but lets you list all SCC-related RBAC objects with `kubectl get clusterrole,rolebinding -l rbac.alauda.io/scc-use=true -A`.

Note

You can equally well use a `ClusterRoleBinding` to grant this namespaced ServiceAccount cluster-scoped `use` permission. A namespaced `RoleBinding` is usually clearer when you want the grant to apply only within one namespace.

Step 2.3 — Bind an SCC to a User

When a trusted human operator (authenticated as a Kubernetes `User`, for example via OIDC or certificate) needs to launch Pods directly - for example, an SRE running `kubectl debug` or `kubectl run` - you can grant the SCC to the User principal.

Save as `bind-user-sre.yaml`, replacing `[email protected]` with your User name:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scc-use-hostaccess
  labels:
    rbac.alauda.io/scc-use: "true"
rules:
  - apiGroups: ["security.alauda.io"]
    resources: ["securitycontextconstraints"]
    resourceNames: ["hostaccess"]
    verbs: ["use"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sre-alice-hostaccess
  labels:
    rbac.alauda.io/scc-use: "true"
subjects:
  - kind: User
    name: [email protected]
    apiGroup: rbac.authorization.k8s.io
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: scc-use-hostaccess
```

Apply:

```
kubectl apply -f bind-user-sre.yaml
```

When `[email protected]` runs `kubectl run` directly (not via a controller's ServiceAccount), the Pod they create is admitted under their User identity and gets `hostaccess`.

Step 2.4 — Bind an SCC to a Group

Group bindings are useful for administrator-managed blanket policies, such as "every authenticated user can run `restricted-v2` Pods". Two synthesized groups are particularly relevant:

- `system:authenticated` — every authenticated principal.
- `system:serviceaccounts:<namespace>` — every ServiceAccount in a specific namespace.

Save as `bind-group-authenticated.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scc-use-restricted-v2
  labels:
    rbac.alauda.io/scc-use: "true"
rules:
  - apiGroups: ["security.alauda.io"]
    resources: ["securitycontextconstraints"]
    resourceNames: ["restricted-v2"]
    verbs: ["use"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: scc-use-restricted-v2-authenticated
  labels:
    rbac.alauda.io/scc-use: "true"
subjects:
  - kind: Group
    name: system:authenticated
    apiGroup: rbac.authorization.k8s.io
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: scc-use-restricted-v2
```

Warning

A `system:authenticated` group binding is a **fallback** that catches workloads whose ServiceAccount has no explicit SCC binding. It is acceptable as a migration safety net during the Warn-stage rollout in Step 1.4. Once every workload has an explicit binding, remove this fallback. Leaving it in place permanently widens your blast radius if a future SCC profile is added with permissive defaults.

To restrict the binding to a single namespace's ServiceAccounts, change `subjects` to:

```
subjects:
  - kind: Group
    name: system:serviceaccounts:my-namespace
    apiGroup: rbac.authorization.k8s.io
```

Step 2.5 — Pin a specific SCC with `alauda.io/required-scc`

By default the engine picks the most restrictive SCC a subject is allowed to use and which the Pod actually satisfies. If you have a workload that must always be admitted under one specific profile - for example, an audit-sensitive deployment that must use `restricted-v3` even though its ServiceAccount is also allowed `anyuid` - set the `alauda.io/required-scc` annotation on the Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: audited-app
  namespace: payments
  annotations:
    alauda.io/required-scc: restricted-v3
spec:
  serviceAccountName: payments-sa
  securityContext:
    runAsNonRoot: true
    runAsUser: 1500
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: app
      image: registry.example.com/payments/audited-app:1.2.3
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
```

The `alauda.io/required-scc` annotation only selects from SCCs the subject is already authorized to use. It does not grant SCC access. For this annotation to take effect, **both** of the following must hold:

- A SecurityContextConstraints named `restricted-v3` exists in the cluster.
- `payments/payments-sa` is bound to `restricted-v3` through a ClusterRole or Role that grants `use` on that resource name.

If either condition fails, the Pod is rejected. The validating policy emits specific messages for each case (see **Troubleshooting**).

When using PodTemplate-style controllers (Deployment, StatefulSet, Job), put the annotation in the Pod template's `metadata`, not on the controller:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: audited-app
  namespace: payments
spec:
  selector:
    matchLabels:
      app: audited-app
  template:
    metadata:
      labels:
        app: audited-app
      annotations:
        alauda.io/required-scc: restricted-v3
    spec:
      serviceAccountName: payments-sa
      # ...
```

Step 2.6 — Verify the binding takes effect

After applying any binding, run the following checks.

Administrator verification — confirm the subject can `use` the SCC:

```bash
kubectl auth can-i use \
  securitycontextconstraints.security.alauda.io/anyuid \
  --as="system:serviceaccount:databases:postgres-sa" -n databases
```

The expected output is `yes`. If you get `no`, recheck the `apiGroups`, `resources`, `resourceNames`, and `verbs` in your ClusterRole.

Application owner verification — after the administrator confirms the binding, create or redeploy the workload with the approved ServiceAccount, then inspect the admitted Pod annotation. For a quick probe:

```
kubectl -n databases run probe \
  --image=registry.example.com/library/pause:3.10 \
  --serviceaccount=postgres-sa \
  --overrides='{"spec":{"securityContext":{"runAsUser":999}}}' \
  --command -- /pause

kubectl -n databases get pod probe \
  -o jsonpath='{.metadata.annotations.alauda\.io/scc}{"\n"}'
```

The output should be the name of the SCC the engine selected (in this example, `anyuid`). If the application owner cannot create probe Pods, the administrator can perform this check or inspect a Pod from the real workload.

> **Note**
>
> GlobalContextEntries refresh on a list/watch basis and propagate new bindings to the admission cache typically within seconds, sometimes up to a minute under heavy load. If a Pod is rejected immediately after you apply a new binding, wait a moment and retry before assuming the binding is wrong.

Results

After completing Part 1 and at least one Part 2 binding, you should be able to verify all of the following:

- `kubectl get crd securitycontextconstraints.security.alauda.io` shows the CRD in state `Established=True`.
- `kubectl get scc` lists every SCC profile you installed.

- `kubectl get globalcontextentry` returns all five `scc-*` entries.
- `kubectl get validatingpolicy scc-auto-pick` and `kubectl get mutatingpolicy scc-fill-defaults` both show `READY=true`.
- Pods created under a bound ServiceAccount in a non-system namespace receive an `alauda.io/scc=<name>` annotation, naming the SCC the engine selected.
- Pods created under an unbound ServiceAccount in a non-system namespace are rejected at admission time with the message `Pod violates all SCCs assigned to its ServiceAccount`.

Troubleshooting

Use this table to map symptoms to causes and resolution steps.

For Pods created by controllers such as Deployments, StatefulSets, Jobs, and DaemonSets, the effective workload identity is normally the Pod's ServiceAccount. For Pods created directly by a trusted human operator, such as `kubectl run` or `kubectl debug`, User and Group SCC bindings can also match the admission request.

Symptom	Likely cause	What to check
`Pod violates all SCCs assigned to its ServiceAccount (candidates: ...)`	The Pod's ServiceAccount is bound to at least one SCC, but the Pod's spec violates every one of them. The candidate list at the end of the message names the SCCs that were considered.	For each candidate, comp fields. Common mismatch the allowed range, missin `requiredDropCapabilit` `seccompProfile.type` `runtime/default`.
`Pod violates all SCCs assigned to its ServiceAccount (candidates: )` (empty candidate list)	No SCC is bound to the Pod's ServiceAccount.	Run `kubectl auth can-` `securitycontextconstra` `--as=system:serviceacc` each SCC name from `ku` Add a binding per Step 2.
`required SCC '<name>' not found in scc-profiles`	The `alauda.io/required-scc` annotation references a SCC that does not exist.	Run `kubectl get scc <` install the missing profile.

Symptom	Likely cause	What to check
`required SCC '<name>' is not bound to ServiceAccount '<sa>' in namespace '<ns>'`	The annotation references a SCC the ServiceAccount has no `use` permission for.	Add a `RoleBinding` gra `securitycontextconstr` retry.
Binding was just added but Pods are still rejected	The Kyverno GlobalContextEntry caches RBAC objects asynchronously; new bindings take a few seconds to propagate.	Wait 10–30 seconds and r `globalcontextentry scc` `jsonpath='{.status.las` recent refresh.
Pod is admitted but `runAsUser` is unexpectedly set / unset	The mutating policy filled in a default from the selected SCC, or did not because the Pod already declared a value.	Inspect the `alauda.io/s` which SCC was chosen, t `runAsUser.type` and `r` their own `runAsUser` ar
`READY=false` on `scc-fill-defaults` or `scc-auto-pick`	Kyverno is missing read permission on a resource the policies match (most often `pods/ephemeralcontainers`).	Re-apply the `kyverno-sc` 1.3 in full.
Pods in a `pod-security.kubernetes.io/enforce: restricted` namespace are rejected before Kyverno sees them	The Kubernetes Pod Security Admission plugin runs ahead of Kyverno and enforces the namespace label independently.	Relax the namespace lab as appropriate for the wor restrict which SCCs you o

Learn More

Temporarily bypass the policy with `PolicyException`

When you need to allow a single ServiceAccount to exceed its current SCC for a short period (for example, an emergency debug session), and changing the RBAC binding is not appropriate, use a `PolicyException` resource. This requires Kyverno's admission controller to have been started with `--enablePolicyException=true`.

```yaml
apiVersion: policies.kyverno.io/v1alpha1
kind: PolicyException
metadata:
  name: postgres-debug-bypass
  namespace: policy-exceptions
spec:
  policyRefs:
    - name: scc-auto-pick
      kind: ValidatingPolicy
  matchConditions:
    - name: target-sa
      expression: |
        object.metadata.namespace == 'databases' &&
        object.spec.?serviceAccountName.orValue('') == 'postgres-sa'
    - name: must-be-debug-window
      expression: |
        object.metadata.?labels[?'debug-window'].orValue('') == 'open'
```

Best practice: place `PolicyException` resources in a dedicated namespace (for example, `policy-exceptions`) with restricted write access, label each exception with `owner` and `expire-at`, and audit them on a recurring schedule.

How the engine picks an SCC

When more than one SCC is granted to a subject and the Pod satisfies more than one of them, the validating policy ranks candidates in this order:

1. Higher `priority` first.
2. Higher `restrictiveScore` second.

The first candidate whose fields the Pod fully satisfies is the one chosen. The mutating policy uses the same ordering when picking which SCC's defaults to fill in. This matches OpenShift's intent of "the most restrictive acceptable SCC wins" while allowing operators to override the order with `priority` on a per-profile basis.

Mapping from OpenShift commands

If you are coming from OpenShift, the following `oc` commands translate to plain `kubectl apply` against the SCC engine. These operations grant SCC `use` permission and should be performed only by administrators who are allowed to change the cluster's Pod security boundary.

OpenShift command	Equivalent on this engine
`oc adm policy add-scc-to-user <scc> <user>`	Create a ClusterRole granting `use` on `securitycontextconstraints/<scc>`, then a ClusterRoleBinding with `subjects: [{kind: User, name: <user>}]`.
`oc adm policy add-scc-to-user <scc> -z <sa> -n <ns>`	Same ClusterRole as above, plus a `RoleBinding` in namespace `<ns>` with `subjects: [{kind: ServiceAccount, name: <sa>, namespace: <ns>}]`.
`oc adm policy add-scc-to-group <scc> <group>`	Same ClusterRole, plus a ClusterRoleBinding with `subjects: [{kind: Group, name: <group>}]`.
`oc get scc`	`kubectl get scc` (the CRD's `shortNames: [scc]` keeps the command identical).

Best Practice: Grant a Privileged Exception in an Existing Namespace

Use this procedure when an existing namespace normally follows the PSA `restricted` standard, but one workload in that same namespace must run a privileged container. The namespace remains unchanged; instead, the platform administrator removes PSA restricted enforcement from that namespace and makes the Kyverno SCC engine the enforcing admission boundary.

The isolation model is:

- Every ServiceAccount in the namespace can use `restricted-v2` as the default security ceiling.
- The exceptional workload uses a dedicated ServiceAccount.

- Only that ServiceAccount can use the built-in `privileged` SCC.
- The workload pins `privileged` with `alauda.io/required-scc`.
- Other ServiceAccounts are not granted `privileged` and continue to be rejected if they request privileged settings.

> **Warning**
>
> PSA and the Kyverno SCC engine are independent admission controls. An SCC grant cannot override `pod-security.kubernetes.io/enforce: restricted`. Complete the restricted SCC baseline and the exceptional workload binding before removing PSA enforcement, so the namespace never has an unprotected transition window.

Step 1 - Audit existing SCC grants

Before changing PSA, check whether ordinary ServiceAccounts already receive permissive SCCs through User, Group, RoleBinding, or ClusterRoleBinding grants. Pay particular attention to bindings for `system:authenticated`, `system:serviceaccounts`, and `system:serviceaccounts:payments`.

For example, the default ServiceAccount should not be able to use the built-in `privileged` SCC:

```
kubectl auth can-i use \
  securitycontextconstraints.security.alauda.io/privileged \
  --as=system:serviceaccount:payments:default \
  -n payments
```

The expected output is `no`. Remove unintended broad grants before proceeding. The optional `rbac.alauda.io/scc-use=true` label described earlier can help locate SCC-related RBAC objects, but do not rely on the label alone because an unlabeled Role can still grant `use` permission.

Step 2 - Bind the restricted baseline to all namespace ServiceAccounts

Create a namespaced RoleBinding from the synthesized `system:serviceaccounts:payments` group to the `scc-use-restricted-v2` ClusterRole from Step 2.4. This covers existing and future ServiceAccounts in the namespace.

Save as `payments-restricted-baseline.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: all-serviceaccounts-restricted-v2
  namespace: payments
  labels:
    rbac.alauda.io/scc-use: "true"
subjects:
  - kind: Group
    name: system:serviceaccounts:payments
    apiGroup: rbac.authorization.k8s.io
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: scc-use-restricted-v2
```

Apply the binding and verify an ordinary ServiceAccount can use only the restricted baseline:

```bash
kubectl apply -f payments-restricted-baseline.yaml

kubectl auth can-i use \
  securitycontextconstraints.security.alauda.io/restricted-v2 \
  --as=system:serviceaccount:payments:default \
  -n payments
```

The expected output is `yes`.

Step 3 - Create a dedicated ServiceAccount

SCC authorization is subject-based; it does not grant permissions to a Pod name. Model a single-workload exception with an exclusive ServiceAccount that is not shared by unrelated workloads.

Save as `payment-agent-serviceaccount.yaml`:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: payment-agent-privileged
  namespace: payments
automountServiceAccountToken: false
```

Apply the ServiceAccount:

```bash
kubectl apply -f payment-agent-serviceaccount.yaml
```

> **Warning**
>
> The built-in `privileged` SCC is intentionally unrestricted. In addition to privileged containers, it allows host networking, host namespaces, host ports, host paths, all Linux capabilities, all volume types, all seccomp profiles, and unsafe sysctls. Grant it only when the workload genuinely requires full privilege. If the workload needs a smaller exception, such as only `anyuid`, `hostNetwork`, or `hostPath`, bind one of the narrower built-in profiles instead.

Leave `automountServiceAccountToken: false` unless the workload must call the Kubernetes API. Enabling token automount adds API credentials to a container that already has elevated runtime privileges.

Step 4 - Grant `privileged` only to the dedicated ServiceAccount

Create a ClusterRole whose `resourceNames` contains only the built-in `privileged` SCC, then bind it to the dedicated ServiceAccount with a RoleBinding in the workload namespace.

Save as `payment-agent-scc-binding.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scc-use-privileged
  labels:
    rbac.alauda.io/scc-use: "true"
rules:
  - apiGroups: ["security.alauda.io"]
    resources: ["securitycontextconstraints"]
    resourceNames: ["privileged"]
    verbs: ["use"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: payment-agent-privileged-scc
  namespace: payments
  labels:
    rbac.alauda.io/scc-use: "true"
subjects:
  - kind: ServiceAccount
    name: payment-agent-privileged
    namespace: payments
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: scc-use-privileged
```

Apply and verify the binding:

```bash
kubectl apply -f payment-agent-scc-binding.yaml

kubectl auth can-i use \
  securitycontextconstraints.security.alauda.io/privileged \
  --as=system:serviceaccount:payments:payment-agent-privileged \
  -n payments

kubectl auth can-i use \
  securitycontextconstraints.security.alauda.io/privileged \
  --as=system:serviceaccount:payments:default \
  -n payments
```

The expected outputs are `yes` for `payment-agent-privileged` and `no` for `default`. The dedicated ServiceAccount also inherits `restricted-v2` from the namespace group binding.

Step 5 - Prevent other workloads from reusing the ServiceAccount

This procedure grants `privileged` to a ServiceAccount, not to a Pod or controller name. Kubernetes RBAC does not provide field-level authorization for `spec.serviceAccountName`. A user who can create arbitrary Pods, Deployments, StatefulSets, DaemonSets, Jobs, or CronJobs in `payments` may otherwise reference `payment-agent-privileged` from another workload and receive the same SCC access.

Before continuing, enforce at least one of these controls:

- Allow only a dedicated GitOps or deployment identity to create or update workloads that reference `payment-agent-privileged`, and prevent ordinary namespace users from modifying those workloads.
- Add an admission guard that rejects references to `payment-agent-privileged` unless the request is authorized. Apply the guard to direct Pods and every Pod-template controller enabled in the namespace.

Do not rely only on a workload label or annotation to authorize use of the dedicated ServiceAccount; users who can create workloads can normally copy those values. If neither deployment ownership nor admission-time ServiceAccount usage can be restricted, this pattern provides a ServiceAccount-level exception only and must not be represented as a single-Pod exception.

Do not continue to the PSA transition until one of these controls is enforced.

Step 6 - Transfer enforcement from PSA to Kyverno SCC

Remove only the PSA restricted enforcement labels. Keep restricted audit and warning labels so the API server continues to report workloads that would violate the restricted standard.

```
kubectl label namespace payments \
  pod-security.kubernetes.io/enforce- \
  pod-security.kubernetes.io/enforce-version- \
  --overwrite

kubectl label namespace payments \
  pod-security.kubernetes.io/audit=restricted \
  pod-security.kubernetes.io/audit-version=latest \
  pod-security.kubernetes.io/warn=restricted \
  pod-security.kubernetes.io/warn-version=latest \
  --overwrite
```

Removing namespace labels does not override a cluster-wide PSA default configured through the API server admission configuration. If the cluster default still enforces `restricted`, explicitly opt this namespace out of enforcement while retaining audit and warning visibility:

```
kubectl label namespace payments \
  pod-security.kubernetes.io/enforce=privileged \
  pod-security.kubernetes.io/enforce-version=latest \
  --overwrite
```

Here `enforce=privileged` means PSA does not impose additional restrictions. It does not grant privileged access to Pods; the Kyverno SCC bindings remain the enforcing authorization boundary.

Step 7 - Pin the approved SCC in the workload

Put `alauda.io/required-scc` in the Pod template metadata and use the dedicated ServiceAccount. Do not put the annotation only on the Deployment metadata because controller metadata is not copied automatically to its Pods.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payment-agent
  namespace: payments
spec:
  replicas: 1
  selector:
    matchLabels:
      app: payment-agent
  template:
    metadata:
      labels:
        app: payment-agent
      annotations:
        alauda.io/required-scc: privileged
    spec:
      serviceAccountName: payment-agent-privileged
      securityContext:
        seccompProfile:
          type: Unconfined
      containers:
        - name: agent
          image: registry.example.com/payments/agent:1.0.0
          securityContext:
            privileged: true
            allowPrivilegeEscalation: true
            runAsUser: 0
```

The annotation does not grant SCC permission. If the SCC is missing or the ServiceAccount does not have `use` permission, the validating policy rejects the Pod.

Step 8 - Verify that the exception is isolated

Run admission probes as the real application deployment identity, not as `cluster-admin`. SCC selection takes the request User and Groups into account for directly created Pods, so an administrator with separate broad SCC grants can produce a different result from the workload identity used in production.

Verify the exceptional workload is admitted under the intended SCC:

```
kubectl -n payments get pod -l app=payment-agent \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.annotations.alauda\.io/scc}{"\n"}{end}'
```

The SCC column should be `privileged`.

Next, confirm an ordinary restricted Pod remains admissible under `restricted-v2`:

```
cat <<'EOF' | kubectl create --dry-run=server -f - \
  -o jsonpath='{.metadata.annotations.alauda\.io/scc}{"\n"}'
apiVersion: v1
kind: Pod
metadata:
  name: restricted-probe
  namespace: payments
spec:
  serviceAccountName: default
  securityContext:
    runAsUser: 1000
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: probe
      image: registry.example.com/library/pause:3.10
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
EOF
```

The expected output is `restricted-v2`.

Finally, confirm an ordinary ServiceAccount cannot request `privileged`:

```
cat <<'EOF' | kubectl create --dry-run=server -f -
apiVersion: v1
kind: Pod
metadata:
  name: unauthorized-privileged-probe
  namespace: payments
  annotations:
    alauda.io/required-scc: privileged
spec:
  serviceAccountName: default
  containers:
    - name: probe
      image: registry.example.com/library/pause:3.10
      securityContext:
        privileged: true
EOF
```

When submitted by the normal application deployment identity, the request must be rejected with a message stating that the required SCC is not bound to ServiceAccount `default`. If this Pod is admitted, stop and inspect broad SCC RoleBindings, ClusterRoleBindings, and User or Group grants before allowing production workloads.

Roll back the exception

To restore PSA restricted enforcement safely:

1. Stop or replace the privileged workload with a `restricted`-compliant specification.
2. Remove the RoleBinding that grants `privileged` to the dedicated ServiceAccount:

   ```
   kubectl -n payments delete rolebinding payment-agent-privileged-scc
   ```

3. Confirm the ServiceAccount can no longer use `privileged`:

   ```
   kubectl auth can-i use \
     securitycontextconstraints.security.alauda.io/privileged \
     --as=system:serviceaccount:payments:payment-agent-privileged \
     -n payments
   ```

The authorization check must return `no`. Do not delete the built-in `privileged` SCC. Delete the reusable `scc-use-privileged` ClusterRole only when no other RoleBinding or ClusterRoleBinding references it.

4. Run server-side dry-run checks for the remaining workloads.
5. Restore PSA restricted enforcement last:

```
kubectl label namespace payments \
  pod-security.kubernetes.io/enforce=restricted \
  pod-security.kubernetes.io/enforce-version=latest \
  --overwrite
```

Next Steps

- Decide which SCC each existing namespace and ServiceAccount should be bound to after reviewing workload requirements, document the mapping, and apply the bindings through your GitOps workflow so that they are auditable and reproducible.
- Plan a periodic review of `PolicyException` resources — they are intended for short windows, not permanent exemptions.
- If you operate at large scale, monitor the Kyverno admission controller's `kyverno_admission_review_duration_seconds` metric to detect changes in admission latency as the number of SCC profiles or RBAC bindings grows.

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:
```

```
            - 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:

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

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

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

Create Role for Secret Access

```
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 o
nly
```

Bind Role to Service Account

```
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-----
```

```bash
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:

```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
```

```
    # 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:

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 regi
stry, GCR, or official Docker images"
        anyPattern:
        - spec:
            containers:
            - image: "registry.company.com/*"      # Company registry
        - spec:
            containers:
            - image: "gcr.io/project-name/*"       # Google Container Reg
istry
        - spec:
            containers:
            - image: "docker.io/library/*"         # Official Docker imag
es 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, 'unt
rusted-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 registr
y
- 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 GC
R
```

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`

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

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 a
nd 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.ephemeralCon
tainers[*].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:

```
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:
```

```
        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 runAsNo
nRoot 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.securityC
ontext.runAsNonRoot
          must be set to true, or the field spec.containers[*].securityCo
ntext.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 securi
ty"
        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
```

```
            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 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

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

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 Privilege Escalation (Should Fail)

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 runAsNo
nRoot 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.securityC
ontext.runAsNonRoot
          must be set to true, or the field spec.containers[*].securityCo
ntext.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:

```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 privil
ege 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.ru
nAsUser && (securityContext.runAsUser < `1000` || securityContext.runAsUs
er > `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:

```
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 value
s.
      Both the Unconfined profile and the absence of a profile are prohib
ited.
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 Runtime
Default 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:

```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 context
s"
        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:
```

```
            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 contex
ts"
        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"
```

```
      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:

```
apiVersion: kyverno.io/v1
kind: Policy
metadata:
  annotations:
    policies.kyverno.io/category: Pod Security Standards
    policies.kyverno.io/description: 'Strictly follow the Security Contex
t configuration in the image to automatically add, allowPrivilegeEscalati
on:
      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:
```

```
      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:
```

```
                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
```

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 NodePort Service (Should Fail)

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 tra
ffic 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 m
ust 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:

```
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 a
nd 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.ephemeralCon
tainers[*].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:

```
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 con
flicts 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] | le
ngth(@) }}"
          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 }}"
```

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

Policy 3: Require Network Policies

Ensure pods have associated NetworkPolicies for traffic control:

```
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 traf
fic.
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.name
space }}/networkpolicies"
          jmesPath: "items[?spec.podSelector.matchLabels.app == '{{ reque
st.object.metadata.labels.app }}'] | length(@)"
      validate:
        message: >-
          Pods must have an associated NetworkPolicy. Create a NetworkPol
icy 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 exfilt
ration.
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 o
nly.
        pattern:
          spec:
            =(dnsPolicy): "Default | ClusterFirst"
    - name: restrict-custom-dns
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Custom DNS configuration is not allowed in production environme
nts.
        pattern:
          spec:
            X(dnsConfig): "null"
```

Advanced Scenarios

Scenario 1: Environment-Specific Network Policies

Different network restrictions for different environments:

```
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 securit
y"
        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 securit
y"
```

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

Scenario 2: Application-Specific Network Policies

Different network policies for different application types:

```
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 p
orts"
        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 port
s"
        foreach:
        - list: request.object.spec.containers[].ports[]
```

```
      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
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: 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
```

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 conf
igMap, 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, cs
i, downwardAPI,
          emptyDir, ephemeral, persistentVolumeClaim, projected, and secr
et.
        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:

```
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 al
lowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: disallow-privileged-volumes
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Privileged volume types are not allowed: hostPath, gcePersisten
tDisk,
          awsElasticBlockStore, gitRepo, nfs, iscsi, glusterfs, rbd, flex
Volume,
          cinder, cephFS, flocker, fc, azureFile, azureDisk, vsphereVolum
e, 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:

```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"
```

```
    - 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
```

```
        - 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(@) }}"
```

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

Scenario 3: Volume Size and Resource Limits

Control volume sizes and resource usage:

```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
```

Kyverno Policy Configuration Use Cases

This document provides core policy configuration use cases based on Kyverno. It helps you implement automatic resource mutation, unified configuration, and automated injection of security templates and baseline environments based on Namespaces or Projects in your Kubernetes cluster.

TOC

1. Resource Mutation and Unified Configuration (Mutate)

Kyverno's Mutate rules can automatically modify submitted resources during the admission control phase. The following cases demonstrate how to inject unified labels into all Pods under a namespace and enforce a uniform `restartPolicy`.

1.1 Injecting Unified Labels into Pods

This policy will automatically append preset labels to all newly created Pods in the cluster (or within a specific namespace). This is commonly used for unified project management and billing scheduling.

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-default-pod-labels
  annotations:
    policies.kyverno.io/title: Add Default Pod Labels
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Automatically inject default labels into newly created Pods.
spec:
  rules:
    - name: inject-pod-labels
      match:
        any:
        - resources:
            kinds:
              - Pod
            # Optional: Uncomment and modify the following fields to appl
y only to specific namespaces
            # namespaces:
            #   - my-project-ns
      mutate:
        patchStrategicMerge:
          metadata:
            labels:
              # The +() syntax adds the label if it doesn't exist, and do
es not overwrite it if it does
              +(company.com/managed-by): "kyverno"
              +(company.com/environment): "production"
```

1.2 Enforcing Uniform RestartPolicy for Pods

This policy enforces that the default `restartPolicy` for all newly created Pods is `Always`. This is critical to ensure that business containers are automatically restarted if they exit

unexpectedly.

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: enforce-restart-policy
spec:
  rules:
    - name: set-restart-policy-always
      match:
        any:
        - resources:
            kinds:
              - Pod
      mutate:
        patchStrategicMerge:
          spec:
            # This will forcefully overwrite the restartPolicy to Always,
even if the user specifies Never or OnFailure
            restartPolicy: Always
```

---

2. Automated Template Configuration Based on Namespace/Project (Generate)

When a new namespace or project is created, Kyverno's Generate rules can detect this event and automatically generate related Kubernetes resources (such as NetworkPolicy, ConfigMap, Secret, RoleBinding, etc.). This acts as an **out-of-the-box security and unified configuration template**.

2.1 Automatically Injecting Default Isolated NetworkPolicy

This policy automatically generates a default `NetworkPolicy` when a new namespace is created. This policy denies all inbound (Ingress) requests by default, thereby overriding default network connectivity and achieving network isolation between namespaces.

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-default-networkpolicy
  annotations:
    policies.kyverno.io/title: Generate Default NetworkPolicy
    policies.kyverno.io/subject: Namespace, NetworkPolicy
    policies.kyverno.io/description: >-
      Automatically generates a NetworkPolicy that denies all cross-namespace inbound traffic when a new Namespace is created,
      achieving default network isolation between projects.
spec:
  rules:
    - name: generate-deny-all-networkpolicy
      match:
        any:
        - resources:
            kinds:
              - Namespace
      # Exclude specific system namespaces to prevent blocking system components
      exclude:
        any:
        - resources:
            namespaces:
              - kube-system
              - kyverno
              - monitoring
      generate:
        kind: NetworkPolicy
        apiVersion: networking.k8s.io/v1
        name: default-deny-all
        # Use a template variable to get the name of the newly created Namespace
        namespace: "{{request.object.metadata.name}}"
        # synchronize=true means that if this Kyverno policy changes, the generated resources will automatically sync and update
        synchronize: true
        data:
          metadata:
            labels:
              security.policy/type: "default-deny"
          spec:
```

```
            podSelector: {} # An empty selector matches all Pods in the n
amespace
            policyTypes:
            - Ingress
            # Without specifying Ingress rules (whitelists), all Ingress
traffic is denied
```

2.2 Automatically Initializing Project Configurations Based on Templates (DBS/Security Quotas)

This example demonstrates how to automatically prepare a series of underlying environments based on project attributes (e.g., labels applied when creating a Namespace). For instance, issuing DBS connection templates (for CLI or applications to read) and default security quotas (LimitRange).

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: setup-namespace-dbs-template
spec:
  rules:
    # Rule 1: Automatically generate DBS default configuration for projec
ts with specific labels
    - name: generate-dbs-configmap
      match:
        any:
        - resources:
            kinds:
              - Namespace
            selector:
              matchLabels:
                project-type: database # Triggers when the NS has this La
bel
      generate:
        kind: ConfigMap
        apiVersion: v1
        name: dbs-default-template
        namespace: "{{request.object.metadata.name}}"
        synchronize: true
        data:
          data:
            # Default DBS configuration for CLI support tools
            dbs-url: "jdbc:mysql://default-db-cluster:3306/db"
            dbs-cli-version: "v1.2.0"
            secure-mode: "true"

    # Rule 2: Uniformly generate LimitRange (security limits) for all new
ly created Namespaces
    - name: generate-default-limitrange
      match:
        any:
        - resources:
            kinds:
              - Namespace
      generate:
        kind: LimitRange
        apiVersion: v1
        name: default-limits
```

```
      namespace: "{{request.object.metadata.name}}"
      synchronize: true
      data:
        spec:
          limits:
          - default:
              cpu: 500m
              memory: 512Mi
            defaultRequest:
              cpu: 100m
              memory: 128Mi
            type: Container
```

Summary

1. **Mutate Capabilities**: Non-intrusively fixes and supplements YAML submitted by developers, easily achieving label and state control at the resource level (such as `RestartPolicy`).
2. **Generate Capabilities**: Acts as a declarative project generator. Once a Namespace creation event occurs, Kyverno automatically populates security policies (NetworkPolicy) and dependency templates (ConfigMap, Secret, LimitRange) in the background, providing a highly standardized and unified isolated environment for CLI tools and upper-level applications.

Use Kyverno Generate Rules

Use Kyverno `generate` rules to create Kubernetes resources automatically when a matching resource is created or updated. A generate rule defines:

- Which source resource to watch, using `match`.
- Which changes should trigger generation, using optional `preconditions`.
- Which target resource Kyverno should create, using `generate`.
- Whether Kyverno should keep the generated resource synchronized with the source resource.

This guide uses a ConfigMap update as the example workflow. When a specific ConfigMap in a specific namespace changes, Kyverno creates a platform `NotificationMessage` resource. The notification service then sends an email to the specified recipient.

For the upstream Kyverno reference, see the official Kyverno generate rules documentation ↗.

TOC

How Generate Works

A Kyverno generate rule is part of a `ClusterPolicy` or `Policy`.

```
rules:
  - name: <rule-name>
    match:
      any:
        - resources:
            kinds:
              - <source-kind>
            operations:
              - CREATE
              - UPDATE
    preconditions:
      all:
        - key: <expression>
          operator: <operator>
          value: <expression>
    generate:
      apiVersion: <target-api-version>
      kind: <target-kind>
      name: <target-name>
      namespace: <target-namespace>
      synchronize: false
      data:
        <target-resource-content>
```

Use `synchronize: false` when every trigger should create an independent target resource. Use `synchronize: true` only when the generated resource should be updated or removed along with the source resource.

Example Scenario

In this example:

- The source namespace is `app-team-a`.
- The source ConfigMap is `app-notification-source`.
- Only `UPDATE` operations are matched.
- Only changes to `ConfigMap.data` trigger generation.
- Kyverno creates a short-lived `NotificationMessage` in `cpaas-system`.
- The generated message sends an email directly to `platform-operator@example.com`.

The notification message API used in this example is `notificationmessages.aitextensions.alauda.io/v1beta1`. It is served by the platform notification API and is consumed quickly after creation.

Prerequisites

- Alauda Container Platform Compliance with Kyverno is installed.
- The platform notification capability is installed in the same cluster.
- An email notification server has been configured. The platform default email server is stored as the `platform-email-server` Secret in the `cpaas-system` namespace.
- A notification template for email messages exists in the `cpaas-system` namespace.
- Kyverno has permission to create `NotificationMessage` resources in the target namespace.

This example does not require a notification rule, notification policy, notification receiver, or notification sender resource. The recipient email address is specified directly in the generated `NotificationMessage`.

Prepare the Target Resource Dependency

The generated `NotificationMessage` references an email template. Create the template before applying the Kyverno policy.

```
apiVersion: ait.alauda.io/v1beta1
kind: NotificationTemplate
metadata:
  name: configmap-change-email-template
  namespace: cpaas-system
  labels:
    cpaas.io/template.email.body.type: Html
    cpaas.io/template.language: EN
spec:
  templates:
    - type: Email
      subject: "ConfigMap {{ .labels.namespace }}/{{ .labels.name }} chan
ged"
      content: |-
        The watched ConfigMap was updated.

        Namespace: {{ .labels.namespace }}
        Name: {{ .labels.name }}
        Resource version: {{ .labels.resourceVersion }}
        Application: {{ .labels.application }}
        Owner: {{ .labels.owner }}
        Severity: {{ .labels.severity }}
```

Prepare the Source Resource

Create the namespace and ConfigMap that Kyverno will watch.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: app-team-a
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-notification-source
  namespace: app-team-a
data:
  application: payment
  owner: platform-ops
  severity: warning
```

Grant Kyverno Permission

Kyverno creates generated resources through its background controller. Grant the controller permission to create `NotificationMessage` resources in the target namespace.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kyverno-generate-notificationmessages
  namespace: cpaas-system
rules:
  - apiGroups:
      - aitextensions.alauda.io
      - aiops.alauda.io
    resources:
      - notificationmessages
    verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kyverno-generate-notificationmessages
  namespace: cpaas-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kyverno-generate-notificationmessages
subjects:
  - kind: ServiceAccount
    name: kyverno-background-controller
    namespace: kyverno
```

If Kyverno is installed in a different namespace or uses a different background controller ServiceAccount name, update the `subjects` section before applying the YAML.

Create the Generate Policy

Create a `ClusterPolicy` that generates a `NotificationMessage` only when `app-team-a/app-notification-source` is updated and its `data` field changes.

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-notificationmessage-on-configmap-change
  annotations:
    policies.kyverno.io/title: Generate NotificationMessage on ConfigMap
Change
    policies.kyverno.io/category: Operations
    policies.kyverno.io/subject: ConfigMap, NotificationMessage
    policies.kyverno.io/description: >-
      Generates a platform NotificationMessage resource when the watched
ConfigMap data changes.
spec:
  background: false
  rules:
    - name: generate-email-notificationmessage
      match:
        any:
          - resources:
              kinds:
                - ConfigMap
              namespaces:
                - app-team-a
              names:
                - app-notification-source
              operations:
                - UPDATE
      preconditions:
        all:
          - key: "{{ request.object.data || `{}` }}"
            operator: NotEquals
            value: "{{ request.oldObject.data || `{}` }}"
      generate:
        apiVersion: aitextensions.alauda.io/v1beta1
        kind: NotificationMessage
        name: "cm-change-{{ request.object.metadata.namespace }}-{{ reque
st.object.metadata.name }}-{{ request.object.metadata.resourceVersion }}"
        namespace: cpaas-system
        synchronize: false
        data:
          metadata:
            labels:
              app.kubernetes.io/managed-by: kyverno
```

```
              app.kubernetes.io/part-of: configmap-change-notification
              configmap-change/source-namespace: "{{ request.object.metad
ata.namespace }}"
              configmap-change/source-name: "{{ request.object.metadata.n
ame }}"
            annotations:
              configmap-change/resource-version: "{{ request.object.metad
ata.resourceVersion }}"
          spec:
            body:
              labels:
                namespace: "{{ request.object.metadata.namespace }}"
                name: "{{ request.object.metadata.name }}"
                resourceVersion: "{{ request.object.metadata.resourceVers
ion }}"
                application: "{{ request.object.data.application || ''
}}"
                owner: "{{ request.object.data.owner || '' }}"
                severity: "{{ request.object.data.severity || '' }}"
              notifications:
                - name: "configmap-change-email-{{ request.object.metadata.
resourceVersion }}"
                  ephemeral:
                    methods:
                      - Email
                    template: configmap-change-email-template
                  receivers:
                    - destination: platform-operator@example.com
```

The generate rule uses these key fields:

- `match.resources`: Watches only the specified ConfigMap and only `UPDATE` operations.
- `preconditions`: Compares the new and old `data` fields so metadata-only updates do not generate messages.
- `generate.apiVersion`, `kind`, `name`, and `namespace`: Define the target resource identity.
- `generate.data`: Defines the full generated resource body.
- `synchronize: false`: Creates a separate target resource for each matching update.

The generated `NotificationMessage` uses `spec.body.labels` as template variables, `ephemeral.methods` to select email delivery, `ephemeral.template` to reference the email

template, and `receivers[].destination` to specify the recipient email address.

Verify the Generate Rule

Update the watched ConfigMap.

```
kubectl patch configmap app-notification-source \
  -n app-team-a \
  --type merge \
  -p '{"data":{"severity":"critical"}}'
```

Check the Kyverno background controller logs. The logs show that Kyverno created the generated target resource.

```
kubectl logs -n kyverno deploy/kyverno-background-controller --since=5m
```

The log output contains a generated target similar to the following:

```
created generate target resource target=aitextensions.alauda.io/v1beta1/N
otificationMessage/cpaas-system/cm-change-app-team-a-app-notification-sou
rce-19377570
```

For this example, also check the notification API logs. A successful email delivery shows one task, the email sender and recipient, and a `Done` status.

```
kubectl logs -n cpaas-system deploy/courier-api --since=5m
```

The log output contains entries similar to the following:

```
Try to exec {"tasks": 1}
Email begin {"from": ["user@example.com"], "to": ["platform-operator@exam
ple.com"]}
Email end {"from": ["user@example.com"], "to": ["platform-operator@exampl
e.com"]}
"status":{"conditions":[{"method":"Email","addresses":["platform-operator
@example.com"],"type":"Ready","status":"True"}],"phase":"Done"}
```

`NotificationMessage` resources are short-lived and are consumed by the notification API. A normal `kubectl get` command may not show the generated message after it has been accepted.

Troubleshooting

If no target resource is generated, check the Kyverno update requests.

```
kubectl get updaterequests -A
```

If an update request failed, inspect it for the detailed error.

```
kubectl describe updaterequest -n kyverno <update-request-name>
```

Common causes include:

- The source resource update did not match the `match.resources` filter.
- The ConfigMap update did not change the `data` field, so the precondition was not met.
- The Kyverno background controller does not have permission to create `notificationmessages.aitextensions.alauda.io` resources in `cpaas-system`.
- The `notificationmessages.aitextensions.alauda.io/v1beta1` API is not available in the target cluster.
- The generated resource name exceeds the Kubernetes DNS label length limit.
- The email template referenced by `ephemeral.template` does not exist.
- The email notification server is not configured or is not available.

API Refiner

Introduction

Product Introduction

Limitations

Install Alauda Container Platform API Refiner

Install via console

Install via YAML

Uninstallation Procedures

Default Configuration

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

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`

About Alauda Container Platform Compliance Service

Compliance Service is a platform module designed to support STIG compliance scanning and Alauda OS 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 ↗.

Platform Security Configurations

Changing the Platform OIDC Client Secret

ACP uses an OIDC Client Secret for authentication between platform components. You may need to rotate this secret for security compliance, periodic credential rotation, or after a potential credential leak. After updating the secret on the global cluster, the platform automatically synchronizes it to all member clusters.

TOC

Prerequisites

- `kubectl` access to the **global cluster** with cluster-admin privileges.
- All clusters managed by the platform have been upgraded to ACP v4.3.
- The following plugins with the `Agnostic` life cycle have been upgraded to versions compatible with ACP v4.3:

- `Alauda Container Platform Gitops`
- `Alauda Build of Kiali`
- `Kubeflow Base`
- `Alauda AI`

- A new client secret value prepared. It is recommended to use a cryptographically random string of at least 32 characters.

> **WARNING**
>
> Changing the Client Secret causes affected components to restart, resulting in temporary authentication disruptions. Plan this operation during a maintenance window.

OIDC Secret Overview

The platform OIDC credential is stored in a Kubernetes Secret with the following details:

Property	Value
Secret Name	`cpaas-oidc-secret`
Namespace	`cpaas-system`

The Secret contains two data keys:

- `client-id` — The OIDC client identifier, with the default value `alauda-auth`. **Do not modify this value.**
- `client-secret` — The OIDC client secret. This is the value you will rotate.

Procedure

1 Back up the current secret

Before making changes, back up the current Secret manifest so you can roll back if needed.

```
kubectl get secret cpaas-oidc-secret -n cpaas-system -o yaml > cpaas-oidc-secret-backup.yaml
```

WARNING

The backup file contains sensitive credential data. Store it in a secure location and delete it after the operation is complete.

Record the current `client-secret` hash for non-plaintext verification after the update:

```
OLD_SECRET_SHA256="$(kubectl get secret cpaas-oidc-secret -n cpaas-system \
  -o jsonpath='{.data.client-secret}' | base64 -d | openssl dgst -sha256 -r | awk '{print $1}')"
echo "Current client-secret SHA256: ${OLD_SECRET_SHA256}"
```

2 Update the Client Secret

DANGER

Do not modify the `client-id` value, and do not delete and recreate the Secret. Doing so may cause cascading authentication failures across the platform.

Use one of the following methods to update only the `client-secret` data field.

Method 1: Using `kubectl patch` (recommended)

Read the new secret from secure interactive input:

```
read -rs NEW_CLIENT_SECRET
echo
kubectl patch secret cpaas-oidc-secret -n cpaas-system \
  --type merge \
  -p "{\"data\":{\"client-secret\":\"$(printf %s "$NEW_CLIENT_SECRET"
| base64 | tr -d '\n')\"}}"
unset NEW_CLIENT_SECRET
```

Method 2: Using `kubectl edit`

Open the Secret in your default editor and replace the `client-secret` value with the new base64-encoded value.

```
kubectl edit secret cpaas-oidc-secret -n cpaas-system
```

TIP

You can generate the base64-encoded value in advance:

```
read -rs NEW_CLIENT_SECRET
echo
printf %s "$NEW_CLIENT_SECRET" | base64
unset NEW_CLIENT_SECRET
```

Verify that the Secret has been updated on the global cluster without exposing plaintext:

```
NEW_SECRET_SHA256="$(kubectl get secret cpaas-oidc-secret -n cpaas-sy
stem \
  -o jsonpath='{.data.client-secret}' | base64 -d | openssl dgst -sha
256 -r | awk '{print $1}')"
echo "Updated client-secret SHA256: ${NEW_SECRET_SHA256}"
test "${OLD_SECRET_SHA256}" != "${NEW_SECRET_SHA256}" && \
  echo "Verification passed: client-secret has changed."
unset OLD_SECRET_SHA256 NEW_SECRET_SHA256
```

3 Verify component health

After the Secret is updated, the `base-operator` automatically synchronizes it to all member clusters. Some components that depend on the OIDC credential will restart to load the new value. In particular, verify that the `frontend` and `apollo` deployments are running normally:

```
kubectl get deploy -n cpaas-system frontend apollo
```

Verify that all Pods are `READY` and `UP-TO-DATE`. It may take several minutes for the restart to complete.

INFO

If a component fails to start, verify that the `cpaas-oidc-secret` Secret exists in the `cpaas-system` namespace and contains valid `client-id` and `client-secret` values.

Rollback

If issues occur after the Secret change, restore the previous value from the backup created in Step 1:

```
kubectl apply -f cpaas-oidc-secret-backup.yaml
```

Alternatively, if you know the previous secret value:

```
read -rs PREVIOUS_CLIENT_SECRET
echo
kubectl patch secret cpaas-oidc-secret -n cpaas-system \
  --type merge \
  -p "{\"data\":{\"client-secret\":\"$(printf %s "$PREVIOUS_CLIENT_SECRE
T" | base64 | tr -d '\n')\"}}"
unset PREVIOUS_CLIENT_SECRET
```

After rolling back, repeat the verification steps to confirm that all components are healthy.

Disabling the PKCE Plain Method

Starting from ACP v4.3.0, PKCE (Proof Key for Code Exchange) verification is enabled by default for login authorization. To maintain backward compatibility with affected plugins that use the `Agnostic` life cycle and are not yet upgraded, the platform retains the `plain` code challenge method alongside the secure `S256` method after upgrade.

Once all affected `Agnostic` plugins have been upgraded to versions compatible with ACP v4.3.0, you should remove the `plain` method to enforce `S256`-only PKCE verification.

TOC

Prerequisites

- `kubectl` access to the **global cluster** with cluster-admin privileges.
- All clusters managed by the platform have been upgraded to ACP v4.3.
- The following affected plugins with the `Agnostic` life cycle have been upgraded to versions compatible with ACP v4.3:

- `Alauda Container Platform Gitops`
- `Alauda Build of Kiali`
- `Kubeflow Base`
- `Alauda AI`

> **WARNING**
>
> In this context, affected plugins are the listed `Agnostic` plugins that use the platform OIDC authorization flow; removing `plain` before all of them are upgraded to ACP v4.3-compatible versions will cause authentication failures.

Procedure

The OIDC client configuration is stored as an `OAuth2Client` custom resource in the `cpaas-system` namespace. Determine the target resource name by client ID:

```
OIDC_CLIENT_NAME="$(kubectl get oauth2client -n cpaas-system \
  -o jsonpath='{range .items[?(@.id=="alauda-auth")]}{.metadata.name}
{"\n"}{end}' | head -n 1)"
test -n "${OIDC_CLIENT_NAME}" || { echo "Failed to find OAuth2Client for
id=alauda-auth"; exit 1; }
echo "Target OAuth2Client: ${OIDC_CLIENT_NAME}"
```

Edit the `OAuth2Client` resource:

```
kubectl edit oauth2client "${OIDC_CLIENT_NAME}" -n cpaas-system
```

Locate the `codeChallengeMethods` field and remove the `plain` entry, keeping only `S256`:

```
# Before
codeChallengeMethods:
- S256
- plain

# After
codeChallengeMethods:
- S256
```

Save and exit the editor. The change takes effect immediately.

Verification

Confirm that the `plain` method has been removed:

```
kubectl get oauth2client "${OIDC_CLIENT_NAME}" -n cpaas-system \
  -o jsonpath='{.codeChallengeMethods}'
```

The output should be:

```
["S256"]
```

Run at least one login/authorization verification for each affected plugin (`Alauda Container Platform Gitops`, `Alauda Build of Kiali`, `Kubeflow Base`, and `Alauda AI`) to confirm there are no PKCE-related authentication failures.

Rollback

If authentication issues occur after removing the `plain` method (for example, any affected plugin login or authorization fails), add it back:

```
kubectl patch oauth2client "${OIDC_CLIENT_NAME}" -n cpaas-system \
  --type merge \
  -p '{"codeChallengeMethods":["S256","plain"]}'
```

After rollback, verify affected plugin logins again and then `unset OIDC_CLIENT_NAME`.

Users and Roles

User

Introduction

User Sources

User Management Rules

Role Assignment Views

User Lifecycle

Guides

HowTo

Group

Introduction

Group Introduction

Group Types

Guides

Role

Introduction

- Role Introduction
- System Roles
- Custom Permissions

Guides

IDP

Introduction

- Overview
- Supported Integration Methods

Guides

Troublesho

User Policy

Introduction

- Overview
- Configure Security Policy
- Available Policies

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 Operatio

HowTo

Obtain an API Access Token

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
Disabled	User account is inactive and cannot log in. Contact platform administrator for activation.

Status	Description
	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

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 Operatio

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.

Parameter	Description
	Password Requirements: - Length: 8-32 characters - 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
```

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

HowTo

Obtain an API Access Token

Obtain an API Access Token

This guide explains how to obtain an ACP platform access token via API calls. It is intended for scenarios where you need to access platform APIs programmatically.

TOC

Overview

ACP uses a Dex-based OIDC authentication system. The login flow follows the OAuth 2.0 Authorization Code Flow and consists of five API calls that **must be completed within the same HTTP session (sharing cookies)**.

```
Step 1: Retrieve login metadata
    GET /console-platform/api/v1/token/login

Step 2: Obtain the Dex request ID (req_id)
    GET /dex/api/v1/authorize  ← uses query parameters from the auth_url
returned in Step 1

Step 3: Retrieve the RSA public key
    GET /dex/pubkey

Step 4: Submit encrypted credentials and obtain the authorization code
    POST /dex/api/v1/authorize/{idp}?req={req_id}

Step 5: Exchange the authorization code for a token
    GET /console-platform/api/v1/token/callback
```

WARNING

The entire flow must use the same HTTP session (shared cookie jar). Step 2 sets the `cpaas_oidc_auth_flow` session cookie, which Step 5 requires to issue a token. Without this cookie, Step 5 returns `invalid authentication session`.

Prerequisites

Parameter	Description	Value
`PLATFORM_URL`	Platform HTTPS base URL (no trailing path)	`https://<platform>`
`CLIENT_ID`	OAuth client ID (fixed)	`alauda-auth`
`SCOPE`	Requested permission scopes (fixed)	`openid profile offline_access email groups ext`

Parameter	Description	Value
`REDIRECT_URI`	Callback URL (fixed)	`{PLATFORM_URL}/dex/callback`
`IDP`	Identity provider ID	`local` (default for local accounts), `ldap`, `oidc`, etc.

Steps

Step 1: Retrieve Login Metadata

Request:

```
GET /console-platform/api/v1/token/login
```

Query parameters:

Parameter	Value
`client_id`	`alauda-auth`
`redirect_uri`	`https://{platform}/dex/callback`
`response_type`	`code`
`scope`	`openid profile offline_access email groups ext`

Response (200 OK):

```
{
  "auth_url": "https://{platform}/console-dex/auth?access_type=offline&client_id=alauda-auth&code_challenge=...&state=...&...",
  "state": "VpweqrTxqHxpXZzjNOUe1Zbqe7fPIYFN3_HoEYAxWnA",
  "logout_url": "https://{platform}/dex/logout?post_logout_redirect_uri=...&state=..."
}
```

NOTE

The `code_challenge` and related parameters in `auth_url` are generated server-side for the server-to-Dex PKCE exchange and stored in the `cpaas_oidc_auth_flow` cookie. You do not need to interpret them. In Step 2, pass the **complete query string** from `auth_url` as-is.

Step 2: Obtain the Dex Request ID

Use the query parameters from the `auth_url` returned in Step 1.

This step sets the `cpaas_oidc_auth_flow` session cookie. All subsequent requests must carry this cookie.

Request:

```
GET /dex/api/v1/authorize?{query_from_auth_url}
```

Where `{query_from_auth_url}` is the complete query string extracted from the Step 1 `auth_url` (forward it unchanged):

```
access_type=offline&client_id=alauda-auth&code_challenge=...&code_challenge_method=S256
&nonce=...&redirect_uri=...&response_type=code&scope=...&state=...
```

Response (200 OK):

```
{
  "req": "kyrqxpv6tvnj5a3efhkou3seocnehfxout42rb3p75q3g3oue6u"
}
```


Step 3: Retrieve the RSA Public Key

Request:

```
GET /dex/pubkey
```

Response (200 OK):


```
{
  "ts": "1779940350",
  "pubkey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8
A...\n-----END PUBLIC KEY-----\n",
  "pubkey_encode": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0K..."
}
```


Field	Description
`ts`	Timestamp. Must be included in the encrypted payload. Fetch a fresh value for each login attempt.
`pubkey`	RSA public key in PEM format, used to encrypt the password.
`pubkey_encode`	Base64-encoded form of `pubkey`. Equivalent to `pubkey`.

Step 4: Submit Encrypted Credentials and Obtain the Authorization Code

Password encryption:

1. Construct a JSON payload: `{"ts": "<ts from Step 3>", "password": "<plaintext password>"}`
2. Encrypt the JSON bytes using the RSA public key with **PKCS#1 v1.5** padding
3. Base64-encode the encrypted result (standard encoding, not URL-safe)

Request:

```
POST /dex/api/v1/authorize/{idp}?req={req_id}
Content-Type: application/json
Cookie: cpaas_oidc_auth_flow=...
```

Path parameter:

Parameter	Description
`{idp}`	Identity provider ID: `local` (local accounts), `ldap`, etc.

Query parameter:

Parameter	Value
`req`	The `req` ID returned in Step 2

Request body:

```
{
  "account": "admin",
  "password": "<RSA-encrypted Base64 string>"
}
```

Response (200 OK):

```
{
  "session_state": "",
  "redirect_url": "https://{platform}/dex/callback?code=qdul75ionlvlsn3ggg2m...&state=VpweqrTxqH..."
}
```


Extract the `code` and `state` parameters from `redirect_url` for use in Step 5.

Step 5: Exchange the Authorization Code for a Token

Request:

```
GET /console-platform/api/v1/token/callback
Cookie: cpaas_oidc_auth_flow=...
```

Query parameters:

Parameter	Source	Required
`code`	Extracted from `redirect_url` in Step 4	Yes
`state`	Extracted from `redirect_url` in Step 4	Yes

Response (200 OK):

```
{
  "token_type": "bearer",
  "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6...",
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6...",
  "refresh_token": "CjNuY3V3bmRvcW90NjRwaHl4NXB5bndiY2Zuc2MzcXhkdGhud
2...",
  "expire_at": "2025-01-02T12:00:00Z",
  "issued_at": "2025-01-01T12:00:00Z",
  "token_storage": "local"
}
```


Field	Description
`access_token`	JWT for API access. Pass as `Authorization: Bearer <token>` in request headers.
`id_token`	JWT containing user identity claims (email, username, etc.).
`refresh_token`	Opaque string used to refresh the `access_token`.
`expire_at`	Expiry time of the `access_token` (UTC).
`issued_at`	Issue time of the `access_token` (UTC).
`token_storage`	Token storage location. Fixed value: `local`.

Complete Example

Shell Script (curl)

curl maintains the session by sharing a cookie jar file via `-c` (write cookies) and `-b` (read cookies).

Dependencies: `curl`, `jq` (JSON parsing), `openssl` (RSA encryption)

Usage:

```
# Option 1: edit the configuration section in the script, then run it
bash acp-login.sh

# Option 2: pass credentials via environment variables (recommended — avoids plaintext passwords in the script)
ACP_PLATFORM=https://<platform> ACP_USERNAME=admin ACP_PASSWORD=xxx bash acp-login.sh

# Extract the access_token
bash acp-login.sh | jq -r '.access_token'
```

```
#!/usr/bin/env bash
# ACP platform API login script — obtains an access token via the OAuth2 Authorization Code Flow
# Dependencies: curl jq openssl

set -euo pipefail

# ============================================================
# Configuration (can be overridden by environment variables of the same name)
# ============================================================
PLATFORM="${ACP_PLATFORM:-https://your.platform.com}"   # Platform base URL, no trailing path
USERNAME="${ACP_USERNAME:-admin}"                       # Username
PASSWORD="${ACP_PASSWORD:-}"                            # Password (recommended: pass via ACP_PASSWORD env var)
IDP="${ACP_IDP:-local}"                                 # Identity provider: local | ldap | oidc
# ============================================================

CLIENT_ID="alauda-auth"
SCOPE="openid profile offline_access email groups ext"
REDIRECT_URI="${PLATFORM}/dex/callback"

log() { echo "$*" >&2; }
die() { echo "ERROR: $*" >&2; exit 1; }

check_deps() {
    local missing=()
    for cmd in curl jq openssl; do
        command -v "$cmd" &>/dev/null || missing+=("$cmd")
    done
    [[ ${#missing[@]} -eq 0 ]] || die "Missing required tools: ${missing[*]}"
}

check_vars() {
    [[ -n "${PLATFORM}" ]] || die "PLATFORM must not be empty"
    [[ -n "${USERNAME}" ]] || die "USERNAME must not be empty"
    [[ -n "${PASSWORD}" ]] || die "PASSWORD must not be empty (set the ACP_PASSWORD environment variable or fill it in the script)"
}
```

```
urlencode() { jq -rn --arg v "$1" '$v|@uri'; }

# Extract a field from a JSON response; print the response and exit if th
e field is missing
get_field() {
    local json="$1" field="$2" label="$3"
    local val
    val=$(printf '%s' "$json" | jq -r --arg f "$field" '.[$f] // empty' 2
>/dev/null) \
      || die "${label}: jq parse failed"
    [[ -n "$val" ]] || die "${label}: field '${field}' missing from respo
nse
$(printf '%s' "$json" | jq . 2>/dev/null || printf '%s' "$json")"
    printf '%s' "$val"
}

# Extract a query parameter value from a URL
query_param() {
    printf '%s' "$1" | grep -o "${2}=[^&]*" | cut -d= -f2-
}

main() {
    check_deps
    check_vars

    local COOKIE_JAR PUBKEY_RESP PUBKEY_PEM PAYLOAD
    COOKIE_JAR=$(mktemp /tmp/acp-login-XXXXXX.jar)
    PUBKEY_RESP=$(mktemp /tmp/acp-pubkey-XXXXXX.json)
    PUBKEY_PEM=$(mktemp /tmp/acp-key-XXXXXX.pem)
    PAYLOAD=$(mktemp /tmp/acp-payload-XXXXXX.json)
    trap "rm -f '$COOKIE_JAR' '$PUBKEY_RESP' '$PUBKEY_PEM' '$PAYLOAD'" EX
IT

    # Step 1: Retrieve login metadata
    log "[1/5] Retrieving login metadata..."
    local STEP1 AUTH_URL
    STEP1=$(curl -sk -c "$COOKIE_JAR" \
      "${PLATFORM}/console-platform/api/v1/token/login?client_id=${CLIENT
_ID}&redirect_uri=$(urlencode "${REDIRECT_URI}")&response_type=code&scope
=$(urlencode "${SCOPE}")") \
      || die "Step 1: curl request failed"
    AUTH_URL=$(get_field "$STEP1" "auth_url" "Step 1")

    # Step 2: Obtain the Dex request ID (also writes the cpaas_oidc_auth_
```

```
flow cookie)
    log "[2/5] Obtaining Dex request ID..."
    local AUTH_QUERY STEP2 REQ_ID
    AUTH_QUERY="${AUTH_URL#*\?}"
    STEP2=$(curl -sk -c "$COOKIE_JAR" -b "$COOKIE_JAR" \
      "${PLATFORM}/dex/api/v1/authorize?${AUTH_QUERY}") \
      || die "Step 2: curl request failed"
    REQ_ID=$(get_field "$STEP2" "req" "Step 2")

    # Step 3: Retrieve the RSA public key (save response to a file to avo
id newline truncation in bash variables)
    log "[3/5] Retrieving RSA public key..."
    curl -sk -b "$COOKIE_JAR" "${PLATFORM}/dex/pubkey" -o "$PUBKEY_RESP"
\
      || die "Step 3: curl request failed"
    local TS
    TS=$(jq -r '.ts // empty' "$PUBKEY_RESP")
    [[ -n "$TS" ]] || die "Step 3: 'ts' field missing from response"
    jq -r '.pubkey_encode' "$PUBKEY_RESP" | base64 -d > "$PUBKEY_PEM"
    [[ -s "$PUBKEY_PEM" ]] || die "Step 3: public key decode failed"

    # Step 4: Encrypt credentials, submit them, and obtain the authorizat
ion code
    log "[4/5] Submitting credentials and obtaining authorization cod
e..."
    jq -cn --arg ts "$TS" --arg pwd "$PASSWORD" '{"ts":$ts,"password":$pw
d}' > "$PAYLOAD"
    local ENCRYPTED STEP4 REDIRECT_URL CODE STATE
    ENCRYPTED=$(openssl pkeyutl -encrypt -pubin -inkey "$PUBKEY_PEM" \
      -pkeyopt rsa_padding_mode:pkcs1 -in "$PAYLOAD" 2>/dev/null | base64
| tr -d '\n') \
      || die "Step 4: password encryption failed"
    STEP4=$(curl -sk -b "$COOKIE_JAR" -c "$COOKIE_JAR" \
      -X POST "${PLATFORM}/dex/api/v1/authorize/${IDP}?req=${REQ_ID}" \
      -H "Content-Type: application/json" \
      -d "$(jq -cn --arg account "$USERNAME" --arg password "$ENCRYPTED"
'{"account":$account,"password":$password}')") \
      || die "Step 4: curl request failed"
    REDIRECT_URL=$(get_field "$STEP4" "redirect_url" "Step 4 (check usern
ame/password and IDP parameter)")
    CODE=$(query_param "$REDIRECT_URL" "code")
    STATE=$(query_param "$REDIRECT_URL" "state")
    [[ -n "$CODE"  ]] || die "Step 4: 'code' parameter missing from redir
ect_url"
```

```
    [[ -n "$STATE" ]] || die "Step 4: 'state' parameter missing from redi
rect_url"

    # Step 5: Exchange the authorization code for an access token
    log "[5/5] Exchanging authorization code for access token..."
    local TOKENS
    TOKENS=$(curl -sk -b "$COOKIE_JAR" \
      "${PLATFORM}/console-platform/api/v1/token/callback?code=${CODE}&st
ate=$(urlencode "${STATE}")") \
      || die "Step 5: curl request failed"
    get_field "$TOKENS" "access_token" "Step 5" > /dev/null

    local EXPIRE_AT
    EXPIRE_AT=$(printf '%s' "$TOKENS" | jq -r '.expire_at // "unknown"')
    log ""
    log "Login successful | User: ${USERNAME} | Token expires at: ${EXPIR
E_AT}"
    log ""
    printf '%s\n' "$TOKENS" | jq .
}

main "$@"
```

Using the Token

Include the `access_token` in the `Authorization` header of all subsequent API requests:

```
curl -sk https://your.platform.com/kubernetes/{cluster}/api/v1/namespaces
\
  -H "Authorization: Bearer <access_token>"
```

Important Notes

1. **Session cookie must be shared**: The entire login flow must use the same HTTP session (shared cookie jar) so that the `cpaas_oidc_auth_flow` cookie is automatically carried through each step. Without this cookie, Step 5 returns `invalid authentication session (400)`.

2. **The `ts` timestamp cannot be reused**: The `ts` returned in Step 3 is unique per request. It must be combined with the password into a JSON payload before encryption. Reusing a previous `ts` value will cause Step 4 authentication to fail.
3. **Choosing the `IDP` parameter**:
   - `local`: ACP local accounts (default)
   - `ldap`: LDAP/AD domain accounts; the exact ID depends on your platform configuration
4. **TLS certificate verification**: The `-k` flag in the example script skips certificate verification. This is only suitable for test environments with self-signed certificates. In production, remove `-k` and either configure a valid CA certificate or add the platform CA to your system trust store.

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 Loc

Prerequisites

Import Member

Remove Memb

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)

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

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

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

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

Role

Introduction

Introduction

Guides

Create Kubernetes Roles

Manage Roles

How to Crea

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.

Guides

Create Kubernetes Roles

Manage Roles

How to Crea

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.

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.

How to Create Custom Platform Role

TOC

1. Overview

This document explains the core concepts of custom roles and how to configure a RoleTemplate. There are two main ways to build a working custom role — pick either and follow it end to end:

- **FunctionResource method** (5.1): select permissions from the product's built-in permission catalog. Simplest option, no RBAC authoring required.
- **ClusterRole aggregation method** (5.2): author your own native `ClusterRole` and pull it into a RoleTemplate via label matching. More setup, but works independently of what any product module has migrated, and is the platform's long-term direction.

`customRules` (5.3) is a third, lower-level option for one-off native RBAC rules.

Not sure which to pick? Start with **Method A (5.1)** — it's the simplest and covers almost every need. Choose **Method B (5.2)** only when you specifically need a role built from your own `ClusterRole`, or one that must span multiple clusters.

Once a RoleTemplate exists, grant it to users via UserBinding — see Manage Roles for the console steps.

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. A hand-written ClusterRole can be aggregated into a RoleTemplate (or into a built-in system role) 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

Since ACP v4.3, feature permissions are gradually migrating from FunctionResource to native K8s ClusterRole management, aggregated into system roles via ClusterRole labels; FunctionResource-based management will be retired in v4.5. You don't need this background to create a role — skip to section 5 for the steps.

NOTE

The migration is still in progress: **most product modules haven't published aggregation-ready ClusterRoles yet**, so the built-in system labels mentioned in 5.4 (`aggregate-to-namespace-developer`, etc.) resolve to nothing today. Don't build a custom role by pointing `aggregationRules` at one and hoping — 5.2 shows the pattern that works, and 5.5 lets you verify. (You may also see hidden `legacy-*` RoleTemplates in `kubectl` output; these are compatibility bridges for the built-in roles and can be ignored.)

4. Choosing a Method

All three methods produce a working RoleTemplate; they differ in what you author and what they can express.

	Method A: FunctionResource (5.1)	**Method B: ClusterRole aggregation (5.2)**	**Method C: customRules (5.3)**
Best for	Reusing permissions a product already exposes	An independent role built from your own ClusterRole, especially across clusters	A quick one-off native RBAC rule

	Method A: FunctionResource (5.1)	Method B: ClusterRole aggregation (5.2)	Method C: customRules (5.3)
You author	Nothing — pick from the catalog	A ClusterRole plus its aggregation labels	Inline RBAC rules
Multi-cluster	Handled by the platform	Add `sync.scope: "platform"` to your ClusterRole	Handled by the platform
Cross-scope rules	Yes	Yes — one ClusterRole per scope	No — base ClusterRole only

5. Configuration

CAUTION

Always set `rbac.cpaas.io/version: "v2"` and `auth.cpaas.io/roletemplate.frozen: "false"` on a custom RoleTemplate. For custom roles the controller auto-fills a missing `version` with `v1` (which **ignores** `aggregationRules`) and a missing `frozen` with `true` (which **stops later** `spec` **edits from updating the ClusterRoles**), handling the two independently — so setting only one still leaves the other at the legacy default. The examples below also set `auth.cpaas.io/roletemplate.official: "false"` (optional, but never set it to `"true"` on a custom role — that flips it to a built-in system role) and `auth.cpaas.io/roletemplate.level`, which decides where the role is granted — pick the smallest that fits: `namespace` (per namespace), `project` (all namespaces in a project), `cluster` (a whole cluster), or `platform` (the whole platform).

5.1 Method A: FunctionResource

Use this when the permissions you need are already exposed as FunctionResources — this is true for almost all current product modules.

Step 1 — find the FunctionResources you need

```
kubectl get functionresource -o custom-columns='NAME:.metadata.name,MODUL
E:.metadata.labels.auth\.cpaas\.io/functionresource\.module,DISPLAY:.meta
data.annotations.cpaas\.io/functionresource\.function\.display-name'
```

`metadata.name` is what you'll put in `functionResourceRef`; the display-name annotation tells you which UI feature it maps to.

Step 2 — write the RoleTemplate

```
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
    auth.cpaas.io/roletemplate.official: "false"
    rbac.cpaas.io/version: "v2"
    auth.cpaas.io/roletemplate.frozen: "false"
spec:
  rules:
    - functionResourceRef: acp-app
      verbs: [get, list, watch]
```

Step 3 — apply and verify with 5.5.

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

5.2 Method B: ClusterRole aggregation (independent custom role)

Unlike the FunctionResource method, `aggregationRules` doesn't define permissions by itself — it only pulls `rules` in from existing ClusterRoles that carry matching labels. To build

a genuinely independent custom role this way, you provide both halves yourself: the ClusterRole with real rules, and the label that connects it to your RoleTemplate.

Step 1 — write a native ClusterRole with the real permissions, and label it with your own aggregation label plus a scope label. Don't reuse the built-in system-role labels (5.4) unless you deliberately want to grant these permissions to everyone who already holds that built-in role — for an independent role, invent your own label name.

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

Step 2 — if this role needs to work in business clusters too, and the ClusterRole above was only created in the global cluster, add `rbac.cpaas.io/sync.scope: "platform"` so the platform replicates it to every cluster. Without this label the ClusterRole only exists wherever you created it, and the RoleTemplate below will resolve to empty permissions on any cluster that doesn't have it.

```yaml
  labels:
    rbac.cpaas.io/aggregate-to-demo-auditor: "true"
    rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
    rbac.cpaas.io/sync.scope: "platform"
```

Step 3 — create the RoleTemplate, referencing the same labels via `aggregationRules`:

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-auditor
  annotations:
    cpaas.io/display-name: Example Aggregation Role
    cpaas.io/description: ClusterRole aggregation example
  labels:
    auth.cpaas.io/roletemplate.level: namespace
    auth.cpaas.io/roletemplate.official: "false"
    rbac.cpaas.io/version: "v2"
    auth.cpaas.io/roletemplate.frozen: "false"
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-demo-auditor: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
  rules: []
```

To cover more than one scope (e.g. also `cluster` or `system-ns`), add one ClusterRole per scope with the matching scope label, and one more entry under `aggregationRules` per scope — see 6.2 for a worked multi-scope example.

Step 4 — apply and verify with 5.5. An empty result means either the labels on the ClusterRole and in `clusterRoleSelectors` don't actually match, or the ClusterRole hasn't synced to the cluster you're checking (step 2).

> **CAUTION**
>
> Don't point `aggregationRules` at a system/product label you *hope* exists (e.g. `rbac.cpaas.io/aggregate-to-namespace-auditor`) — most modules haven't published theirs yet, so it resolves to nothing and the role silently grants no permissions. Your own ClusterRole plus your own label, as above, always works.

5.3 Method C: customRules

For a one-off native RBAC rule that doesn't warrant a separate ClusterRole object, inline it directly in the RoleTemplate:

```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
    auth.cpaas.io/roletemplate.official: "false"
    rbac.cpaas.io/version: "v2"
    auth.cpaas.io/roletemplate.frozen: "false"
spec:
  customRules:
    - apiGroups: [""]
      resources: ["configmaps"]
      verbs: ["get", "list", "watch"]
```

NOTE

`customRules` are written only into the role's base ClusterRole — they can't be targeted at a specific scope like `cluster` or `system-ns` (there is no `scope` field on `customRules`). If you need cross-scope rules, use Method A or Method B.

5.4 Field Reference

The first table covers metadata and the labels every custom role needs; the second covers the permission fields, grouped by the method they belong to (5.1–5.3).

Metadata and labels (on every custom RoleTemplate)

Field	Description	Values
`metadata.name`	The RoleTemplate's unique name; a	Custom st

Field	Description	Values
	UserBinding references it when granting the role to users.	
`metadata.annotations.cpaas.io/display-name`, `.../description`	Human-readable name and description shown in the console.	Custom st
`metadata.labels.auth.cpaas.io/roletemplate.level`	Where the role is granted, and which ClusterRoles the controller generates for it.	`platform` `cluster` `project` `namespac`
`metadata.labels.auth.cpaas.io/roletemplate.official`	Whether this is a built-in system role. A custom role must not claim to be official.	`"false"` omit; **neve** `"true"` custom ro
`metadata.labels.rbac.cpaas.io/version`	Which conversion logic the controller applies. `v2` understands `aggregationRules`; the legacy `v1` ignores it.	`"v2"` (current); omitting defaults to `"v1"`
`metadata.labels.auth.cpaas.io/roletemplate.frozen`	Whether the generated ClusterRoles are locked. `"false"` lets later `spec` edits take effect.	`"false"` omitting defaults to `"true"` (frozen)

Permission fields (use the set for your method)

Field	Method	Description
`spec.rules[].functionResourceRef`	A	Name of a FunctionResource whose permissions to include. Find names with `kubectl get functionresource`.
`spec.rules[].verbs`	A	Actions the role allows on that FunctionResource's resources.
`spec.aggregationRules[].scope`	B	Which generated ClusterRole the matched rules land in — i.e. where they apply: `business-ns` (namespaces created under a project), `project-ns` (a project's own namespace), `system-ns` (`cpaas-system`), `cluster` (cluster-wide), `kube-public`. Must be compatible with `level` (see note below).
`spec.aggregationRules[].clusterRoleSelectors`	B	Picks which existing ClusterRoles to pull rules from, by their labels.

Field	Method	Description
`spec.customRules[].apiGroups`	C	K8s API groups the rule covers.
`spec.customRules[].resources`	C	Resource types the rule covers.
`spec.customRules[].verbs`	C	Actions allowed on those resources.
`spec.customRules[].resourceNames`	C	Optional: restrict the rule to specific named objects.
`spec.customRules[].nonResourceURLs`	C	Optional: grant access to non-resource URLs (e.g. `/healthz`); for cluster-level roles.

NOTE

- Use either direct rules (`spec.rules` / `spec.customRules`) or `spec.aggregationRules` for a given scope, not both — where they overlap on the same generated ClusterRole, the aggregation rule wins and the direct rules are dropped.
- `auth.cpaas.io/roletemplate.level` (where the role is *granted* — `platform` / `cluster` / `project` / `namespace`) and `spec.aggregationRules[].scope` (which generated ClusterRole each aggregated rule lands in) are different things. The controller enforces one constraint between them: **`platform`- and `cluster`-level roles may only use `scope: cluster`**; `project`- and `namespace`-level roles may use any of the five scopes (including `cluster`). Breaking this rule puts the RoleTemplate into a `Failed` state.

Labels for matchLabels

For an independent custom role, invent your own aggregation label — any `rbac.cpaas.io/aggregate-to-<name>` string works (5.2). Pair it with one **scope label**, which controls *where* the rules apply:

- `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"`

To instead **add permissions to a built-in system role** (advanced — this grants them to everyone who already holds that role), label your ClusterRole with `rbac.cpaas.io/aggregate-to-<built-in-role>`, e.g. `aggregate-to-namespace-developer` or `aggregate-to-platform-admin`.

Multi-cluster distribution (5.2 step 2): add `rbac.cpaas.io/sync.scope: "platform"` to a ClusterRole you create by hand to replicate it from the global cluster to all clusters. It has no effect on a RoleTemplate or on the ClusterRoles a RoleTemplate generates.

5.5 Verify a configuration actually takes effect

Check the RoleTemplate status

The RoleTemplate reports what it generated and whether it synced — start here:

```
kubectl get roletemplate <name> -o jsonpath='{.status.phase}{"\n"}{.statu
s.clusterRoles}{"\n"}'
kubectl get roletemplate <name> -o jsonpath='{.status.conditions}'
```

`status.phase` should be `Ready`; `status.clusterRoles` lists the ClusterRoles it produced; `status.conditions` carries the reason/message when something fails. If the phase isn't `Ready`, `status.clusterRoles` is empty, or the ClusterRoles it names have empty `rules` (`kubectl get clusterrole <name> -o yaml`), the role isn't granting what you expect. Common causes, in rough order of frequency:

- The RoleTemplate is missing `rbac.cpaas.io/version: "v2"`, or was auto-frozen — see the required-labels caution in section 5. This is the most frequent reason `aggregationRules` appears to do nothing.
- A `functionResourceRef` name is misspelled — the controller skips unknown references silently and produces an empty ClusterRole with no error.
- Your `aggregationRules` labels don't match any ClusterRole, or the source ClusterRole hasn't synced to this cluster.

Confirm a user actually gets the permission

Generating ClusterRoles isn't the end goal — a user having access is. After granting the role via UserBinding (see Manage Roles), confirm the real outcome with an impersonated check:

```
kubectl auth can-i get deployments --as=<user> -n <namespace>
```

Check whether an aggregation label is usable before writing `aggregationRules`

Before referencing an `rbac.cpaas.io/aggregate-to-*` label in `clusterRoleSelectors`, confirm it actually resolves to a ClusterRole:

```
kubectl get clusterrole -l rbac.cpaas.io/aggregate-to-<your-label>=true \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.metadata.labels.auth\.cpaas\.io/roletemplate}{"\n"}{end}'
```

- No output → no ClusterRole carries this label. If you're using your own custom label (5.2), double-check the label was actually applied to your ClusterRole and that it synced to this cluster. If you're using a system label (5.4) hoping a product module published it, it likely hasn't — see 3.
- Output shows a `legacy-*` value for `auth.cpaas.io/roletemplate` → this ClusterRole comes from the FunctionResource-based compatibility bridge described in 3, not from a module that has actually migrated. The permissions still originate from FunctionResource, and this bridge is expected to disappear in v4.5.

5.6 Update and delete

- **Update**: edit the RoleTemplate `spec`. Changes only take effect while `auth.cpaas.io/roletemplate.frozen` is `"false"` (see the caution in section 5).
- **Delete**: delete the RoleTemplate. The controller garbage-collects the ClusterRoles it generated across all clusters — you don't clean those up by hand. (A ClusterRole you hand-authored for Method B is yours to delete.)

6. Copy & Trim (Examples)

Two worked examples: trimming a built-in template (Method A, 6.1), and building a multi-scope role from your own ClusterRoles (Method B, 6.2). For a FunctionResource role, copying a built-in system template and trimming it is the quickest way to avoid missing module permissions.

6.1 Trim a system RoleTemplate YAML (Method A)

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:

```
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  annotations:
    cpaas.io/description: Responsible for development, deployment, and ma
intenance within the namespace.
    cpaas.io/display-name: Developer Copy
  labels:
    auth.cpaas.io/roletemplate.level: namespace
    auth.cpaas.io/roletemplate.official: "false"
    rbac.cpaas.io/version: "v2"
    auth.cpaas.io/roletemplate.frozen: "false"
  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 Build a multi-scope auditor role (Method B)

This is the aggregation-method equivalent of 6.1: a namespace-level read-only role that spans both the business and project namespaces, built from your own ClusterRoles and your own label — following the pattern from 5.2.

ClusterRoles — one per scope you want to cover, all sharing the same custom label, each with its own scope label:

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

RoleTemplate — one `aggregationRules` entry per scope, reusing the same custom label:

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-namespace-auditor
  annotations:
    cpaas.io/display-name: Custom Namespace Auditor
    cpaas.io/description: Read-only audit across business and project nam
espaces
  labels:
    auth.cpaas.io/roletemplate.level: namespace
    auth.cpaas.io/roletemplate.official: "false"
    rbac.cpaas.io/version: "v2"
    auth.cpaas.io/roletemplate.frozen: "false"
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-demo-auditor: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-demo-auditor: "true"
            rbac.cpaas.io/aggregate-to-scope-project-ns: "true"
      scope: project-ns
  rules: []
```

Add `system-ns` the same way if you need it, and remember `rbac.cpaas.io/sync.scope: "platform"` on each ClusterRole (5.2 step 2) if it's only created in the global cluster but the role must work in business clusters too.

IDP

Introduction

Introduction

Overview

Supported Integration Methods

Guides

LDAP Management

Before You Begin

Discover the Directory Settings

Choose a Tested Mapping

Add LDAP

Verify the Integration

Synchronization and Lifecycle

Configure LDAP with YAML

Troubleshoot by Symptom

On-Site Diagnostic Tools

OIDC Management

Before You Begin

Add OIDC Using the Form

Verify the Integration

Add OIDC Using YAML

Login, Update, and Delete Behavior

Logout Behavior

Troubleshoot by Stage

Troubleshooting

Delete User

Problem Description

Solution

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.

Guides

LDAP Management

OIDC Management

LDAP Management

ACP can authenticate users from an LDAP directory and can optionally synchronize directory groups. The console workflow supports common OpenLDAP and Microsoft Active Directory layouts.

An integration is complete only after all of the following checks pass:

- ACP creates the LDAP IDP.
- The expected users and, when enabled, groups are synchronized.
- A representative directory user can log in and receives the intended ACP role.

This guide starts with the shortest console workflow. If the directory administrator has not provided the Base DN, Filter, or attribute mappings, use Discover the Directory Settings first.

TOC

Before You Begin

Prepare these values before configuring ACP:

- LDAP server hostname and port reachable from the ACP cluster.
- Bind DN and password with read permission for intended users and groups.
- User Base DN and user Filter.
- A stable, unique, non-empty user identity attribute.

- One or more login attributes.
- Optional group Base DN, group Filter, member attribute, user matching attribute, and group name attribute.
- One representative user account for final login verification.

A successful query from an engineer's laptop is not proof that ACP can connect. ACP Connector creation and synchronization are the cluster-side checks. If the directory administrator has already supplied and verified all required values, continue with Add LDAP.

Discover the Directory Settings

Use this workflow when the Base DN, Filter, or attribute mappings are unknown. The command-line tools referenced here are external pre-integration diagnostic tools, not ACP features.

1. Run Checks from the Correct Network

Run directory queries from a diagnostic host or Pod whose DNS, routing, firewall, and certificate trust are equivalent to the ACP cluster whenever possible. A laptop query is useful preliminary evidence, but ACP Connector creation and synchronization remain the authoritative cluster-side checks.

2. Find the Directory Root

Query the directory Root DSE and record the naming root:

- OpenLDAP and compatible directories commonly publish `namingContexts`.
- Active Directory publishes `defaultNamingContext`.

Use the Root DSE examples in the final **On-Site Diagnostic Tools** section. The returned value, such as `dc=example,dc=com`, is the starting point for locating the user and group branches; it is not automatically the best User Base DN.

3. Choose the User and Group Base DNs

Treat the Base DN as the boundary below which ACP searches. For example:

```
dc=example,dc=com
|- ou=People
|  |- ou=Engineering
|  `- ou=Operations
`- ou=Groups
```

- If all intended users are under `ou=People,dc=example,dc=com`, use that as the User Base DN.
- If intended users span branches outside `ou=People`, use their lowest common parent.
- Use a separate Group Base DN when groups are stored under `ou=Groups`.

Before running a subtree query, use an external base-scope query to confirm that the bind account can read the candidate Base DN. The ACP form uses a subtree search below the configured Base DN. It does not expose a Search Scope field. YAML may use `scope: sub` or `scope: one`; an omitted scope defaults to `sub` in the current Connector runtime.

4. Inspect Representative Users

Inspect at least three entries before choosing mappings:

- A normal user who is expected to log in.
- A user in another intended OU or directory branch.
- A boundary case, such as a disabled user, a user with no groups, or a user missing an optional attribute.

For OpenLDAP, request the DN, `objectClass`, `uid`, `cn`, and `mail`. For Active Directory, request the DN, `objectClass`, `sAMAccountName`, `displayName`, `mail`, `userAccountControl`, and `memberOf`. Confirm that the intended population is below the candidate Base DN before narrowing the Filter.

5. Choose the Identity and Login Attributes

Purpose	OpenLDAP candidates	Active Directory candidates	Selection rule
ACP user identity	`uid`	`sAMAccountName`	Present for every intended user, unique, stable, and non-empty
Login Field	`uid`, `mail`	`sAMAccountName`	Known to users and returns exactly one entry
Display-only information	`cn`	`displayName`	May change and should not be the default identity
Group member value	`member`, `memberUid`	`member`	Must match the value stored in a sample group entry
Group name	`cn`	`cn`	Readable and stable for administrators

Do not use `cn`, `displayName`, or a user's DN as the default ACP identity merely because it is readable. These values can be duplicated, and they can change when a user is renamed or moved. Resolve missing or duplicate identity values in the directory before creating the Connector.

6. Build the User Filter Incrementally

Build and retest the Filter in this order:

1. Select user entries by `objectClass`.
2. Require the ACP identity and login attributes with `(attribute=*)`.
3. Exclude computers, disabled accounts, service accounts, or system accounts when the customer's directory policy requires it.
4. Add business restrictions only after the broad result matches the expected population.

After each condition, rerun the external directory query and compare the count and representative entries with the expected population. Use these Filters to find entries that would be skipped because the proposed identity is absent:

```
# OpenLDAP entries that would be skipped because uid is missing
(&(objectClass=inetOrgPerson)(!(uid=*)))

# Active Directory user objects missing sAMAccountName
(&(objectClass=user)(!(objectClass=computer))(!(sAMAccountName=*)))

# Optional Active Directory exclusion for disabled accounts
(&(objectClass=user)(!(objectClass=computer))(sAMAccountName=*)(!(userAcc
ountControl:1.2.840.113556.1.4.803:=2)))
```

The last expression is a copyable Active Directory LDAP Filter for excluding disabled accounts. Apply it only when that exclusion matches the customer's directory policy.

LDAP Filter Syntax Quick Reference

Build Filters from the smallest expression that returns the intended entries. The operators below were exercised against the test directories used for this guide.

Need	Syntax	Example
Exact value	`(attribute=value)`	`(uid=alice)`
Attribute is present	`(attribute=*)`	`(mail=*)`
All conditions match	`(&(condition1)(condition2))`	`(&(objectClass=inetOr(uid=*))`
Any condition matches	`(|(condition1)(condition2))`	`(|(uid=alice)(mail=alice@test.co`
Exclude a condition	`(!(condition))`	`(!(objectClass=com`
Exclude disabled Active Directory accounts	`(!(userAccountControl:1.2.840.113556.1.4.803:=2))`	Use inside the Active D user Filter shown abov

Do not put a `<username>` placeholder in the configured user Filter. Synchronization runs the configured Filter as written. During login, ACP combines that Filter with an equality lookup for each configured **Login Field** attribute.

7. Identify the Group Membership Model

Inspect one known group and compare its stored membership values with the representative user:

- When `member` contains full user DNs, use **Group Attr** `member` and **User Attr** `DN`.
- When `memberUid` contains short user IDs, use **Group Attr** `memberUid` and **User Attr** `uid`.

ACP evaluates group membership for each user as `(<Group Attr>=<User Attr value>)`. Reproduce that lookup with an external directory query, for example `(member=uid=alice,ou=People,dc=example,dc=com)` or `(memberUid=alice)`. Do not rely only on a user's `memberOf` value when validating this mapping.

Choose a Tested Mapping

These mappings are validated starting points for the listed layouts. The Active Directory Filter excludes computer objects and disabled accounts. Change that Filter only when the customer's directory policy requires a different population, and repeat the external count, ACP synchronization, and login checks after the change.

Directory layout	User Filter	User Name Attr
OpenLDAP `groupOfNames`	`(&(objectClass=inetOrgPerson)(uid=*))`	`uid`
OpenLDAP `posixGroup`	`(&(objectClass=inetOrgPerson)(uid=*))`	`uid`

Directory layout	User Filter	User Name Attr
Active Directory	`(&(objectClass=user)(!(objectClass=computer))(sAMAccountName=*)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))`	`sAMAccountName`

When Active Directory group synchronization uses **Group Attr** `member` and **User Attr** `DN`, use `(&(objectClass=group)(member=*))` as the Group Filter. ACP validates `member` against sampled group entries during creation. A broader `(objectClass=group)` Filter can sample empty groups that do not return a `member` attribute and reject an otherwise valid mapping.

Add LDAP

1. Go to **Users > IDP**.
2. Click **Add LDAP**.
3. Complete **Basic Info**: **Name**, **Display Name**, and optional **Description**.
4. Complete **LDAP Server Setting**: **Server Address**, **Admin Account**, and **Admin Password**.
5. Complete **Search Setting**: **Filter** and **Base DN**.
6. If group synchronization is required, enable **Group Search Setting** and enter **Filter**, **Base DN**, **Group Attr**, and **User Attr**.
7. Complete **Field Mapping**: **User Name Attr**, optional **Group Name Attr**, **Login Field**, and **Username Tip In Login Box**. **User Name Attr** becomes the stable identity value used for the synchronized ACP user; it is not merely a display label. **Login Field** determines how ACP searches for a login account and supports comma-separated alternatives.
8. Optionally open **Advanced Settings**, enable **Auto Sync**, and enter a five-field cron expression in **Sync Rules**. ACP evaluates the rule in UTC.
9. Click **Add**.

> **NOTE**

During creation, ACP validates server connectivity, bind, Base DN, Filter, and mapped attributes. The current form does not authenticate a normal directory user during creation because it does not collect test-user credentials. The submitted bind password is moved to a managed Secret after creation.

Console Field Reference

Use this table to translate values discovered with `ldapsearch` into the current ACP form and, when needed, the YAML fields described later.

ACP console field	YAML field	How to choose the value
Name	`id` and `metadata.name`	Use the same DNS-compatible value for both fields.
Display Name	`name`	Name shown on the ACP IDP chooser.
Description	`metadata.annotations["cpaas.io/description"]`	Optional operator-facing description.
Server Address	`spec.config.host`	Directory hostname or IP and port reachable from ACP.
Admin Account	`spec.config.bindDN`	Bind account DN with read access to intended entries.

ACP console field	YAML field	How to choose the value
Admin Password	`spec.config.bindPW`	Initial-creation input; ACP stores it in a managed Secret.
User **Filter**	`spec.config.userSearch.filter`	Filter that returns only the intended user population.
User **Base DN**	`spec.config.userSearch.baseDN`	Lowest directory branch containing all intended users.
Group **Filter**	`spec.config.groupSearch.filter`	Filter for intended groups; require the member attribute when empty groups exist.
Group **Base DN**	`spec.config.groupSearch.baseDN`	Lowest directory branch containing the intended groups.
Group Attr	`spec.config.groupSearch.groupAttr`	Attribute on a group that stores member values, such as `member` or `memberUid`.

ACP console field	YAML field	How to choose the value
User Attr	`spec.config.groupSearch.userAttr`	User value comparable with **Group Attr**, such as `DN` or `uid`.
User Name Attr	`spec.config.userSearch.nameAttr`	Stable, unique, non-empty identity used for the v2 ACP username and primary identifier.
Group Name Attr	`spec.config.groupSearch.nameAttr`	Readable group name, normally `cn`.
Login Field	`spec.config.userSearch.username`	One or more comma-separated login attributes. Configure only attributes present on the filtered entries you validated.
Username Tip In Login Box	`spec.config.usernamePrompt`	Prompt displayed on the LDAP login page.
Auto Sync	`metadata.labels["cpaas.io/ldap.autoSync"]`	Enable only after manual synchronization is correct.

ACP console field	YAML field	How to choose the value
Sync Rules	`metadata.annotations["cpaas.io/ldap.autoSyncRule"]`	Five-field cron expression evaluated in UTC.

The form does not expose `scope`, TLS flags, CA data, or optional profile mappings. Configure those fields through YAML only when required.

Verify the Integration

1. Open the new LDAP IDP and select **Actions > Sync user**.
2. Confirm the sync result and compare the synchronized user and group counts with the expected directory population.
3. Open one synchronized user and verify that its source, identity value, and state are correct.
4. Assign a minimal role using Manage User Roles or Manage User Group Roles.
5. Use a separate browser session without an ACP login state and log in with the representative directory account.
6. Confirm that authentication succeeds and the assigned ACP permission is available.

> **WARNING**
>
> Overall synchronization can report success while entries missing **User Name Attr** are skipped. A user with no matching directory group does not by itself mean that user synchronization failed. Compare counts and inspect representative entries rather than relying only on the overall result.

Synchronization and Lifecycle

First-Login Synchronization

A directory user can log in before a full manual synchronization. After successful authentication, ACP synchronizes that individual LDAP user. Use the full acceptance path above to verify the expected inventory, group mapping, and authorization rather than treating first login as complete integration acceptance.

Manual Synchronization

Open the LDAP IDP and select **Actions > Sync user** to synchronize and reconcile the directory population on demand. Review the result, counts, skipped-entry messages, and representative users and groups.

Newly synchronized LDAP users are active, valid, and have a default validity period of **Permanent** unless an administrator changes their validity later.

Before synchronization starts, ACP warns that an IDP user with the same name as an existing local user can overwrite or reassociate that user while retaining the existing role assignments. Check the proposed **User Name Attr** for collisions with current ACP users before the first full synchronization. Do not use the built-in local administrator account as an integration test identity.

Automatic Synchronization

Enable **Auto Sync** under **Advanced Settings** when scheduled reconciliation is required, and provide **Sync Rules**. Automatic synchronization uses the configured Base DNs, Filters, and mappings; it does not authenticate directory user passwords. ACP evaluates the five-field cron expression in UTC. For an acceptance test, `*/1 * * * *` runs every minute; replace this temporary rule after the scheduled run is observed.

Update the Connector

Update the LDAP IDP when the server, search boundary, Filter, mapping, or automatic synchronization setting changes. Run a manual synchronization after the update to confirm the new configuration and reconcile existing source users and groups.

Reconcile Upstream-Deleted Users

After manual or automatic reconciliation, a user deleted from the upstream directory becomes invalid in ACP and can no longer authenticate through that source.

Delete the Connector

Deleting the Connector without cleanup retains users and groups from that source and marks the users invalid. If **Clean up IDP users and User Groups** is selected during deletion, ACP deletes the users and groups generated from that Connector.

Configure LDAP with YAML

The following v2 `Connector` templates use `bindPW` only as an initial-creation input. Submit them through the ACP console YAML tab. ACP moves the submitted password to a managed Secret, and stored YAML later contains a managed `clientSecretRef` instead of `bindPW`. Do not use `kubectl apply` with an inline `bindPW`: the Kubernetes last-applied annotation can retain the submitted password in the Connector metadata. Replace every example value before use.

> **Fields deliberately shown in the templates**
>
> ACP adds `metadata.labels["cpaas.io/idp.version"]: v2` when a Connector is created. For a plain LDAP connection without `rootCAData`, ACP also persists `insecureNoSSL: true`, `insecureSkipVerify: true`, and `startTLS: false`. The templates show these values explicitly so that the submitted YAML describes the effective connection mode.
>
> ACP can infer some user and group attributes from the first `objectClass` in a Filter. Do not depend on that inference for customer delivery: it cannot determine the directory's actual identity and membership model. Set `idAttr`, `nameAttr`, `username`, `groupAttr`, `userAttr`, and the group `nameAttr` explicitly after inspecting representative entries. The templates also write `scope: sub` explicitly, although `sub` is the runtime default. `clientSecretRef` is not an input field in these templates because ACP writes it only after moving `bindPW` to the managed Secret.

Strict-Verification OpenLDAP Template

Certificate verification limitation on port 636

In the ACP environment validated for this guide (`auth-controller2` v4.3.6), a Connector whose `host` ends in `:636` is stored with `insecureSkipVerify: true`, even when the submitted value is `false` and `rootCAData` contains the correct CA. Standard-port LDAPS traffic is encrypted, but ACP does not verify the LDAP server certificate.

The template below is for a directory that exposes LDAPS on a non-636 port. This configuration was validated with `insecureSkipVerify: false` and `rootCAData` on a non-636 port. If certificate verification is required, coordinate the port with the directory and network administrators, then complete Connector creation, synchronization, and a real login before rollout.

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: corporate-ldap
name: Corporate LDAP
type: ldap
metadata:
  name: corporate-ldap
  namespace: cpaas-system
  labels:
    cpaas.io/idp.version: v2
spec:
  config:
    host: "<LDAPS_HOST>:<NON_636_LDAPS_PORT>"
    bindDN: cn=reader,dc=example,dc=com
    bindPW: <BIND_PASSWORD>
    insecureNoSSL: false
    insecureSkipVerify: false
    startTLS: false
    rootCAData: <BASE64_ENCODED_CA_CERTIFICATE_PEM>
    usernamePrompt: Directory username
    userSearch:
      baseDN: ou=People,dc=example,dc=com
      filter: "(&(objectClass=inetOrgPerson)(uid=*))"
      scope: sub
      idAttr: uid
      nameAttr: uid
      username: uid
    groupSearch:
      baseDN: ou=Groups,dc=example,dc=com
      filter: "(objectClass=groupOfNames)"
      scope: sub
      groupAttr: member
      userAttr: DN
      nameAttr: cn
```

Active Directory Template

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: corporate-ad
name: Corporate Active Directory
type: ldap
metadata:
  name: corporate-ad
  namespace: cpaas-system
  labels:
    cpaas.io/idp.version: v2
spec:
  config:
    host: ad.example.com:389
    bindDN: CN=ACP Reader,OU=Service Accounts,DC=example,DC=com
    bindPW: <BIND_PASSWORD>
    insecureNoSSL: true
    insecureSkipVerify: true
    startTLS: false
    usernamePrompt: AD username
    userSearch:
      baseDN: OU=Employees,DC=example,DC=com
      filter: "(&(objectClass=user)(!(objectClass=computer))(sAMAccountNa
me=*)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))"
      scope: sub
      idAttr: sAMAccountName
      nameAttr: sAMAccountName
      username: sAMAccountName
    groupSearch:
      baseDN: OU=Groups,DC=example,DC=com
      filter: "(&(objectClass=group)(member=*))"
      scope: sub
      groupAttr: member
      userAttr: DN
      nameAttr: cn
```

Complete LDAP YAML Field Reference

The following skeleton contains the ACP integration fields that are useful for a v2 LDAP Connector. Optional profile mappings are commented out so that the example remains valid

when the directory does not provide those attributes. Remove unused optional fields instead of naming attributes that are absent from the filtered users.

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: corporate-ldap
name: Corporate LDAP
type: ldap
metadata:
  name: corporate-ldap
  namespace: cpaas-system
  annotations:
    cpaas.io/description: Corporate directory
    cpaas.io/ldap.autoSyncRule: "0 2 * * *"
    # cpaas.io/idp.validation: "false"
  labels:
    cpaas.io/idp.version: v2
    cpaas.io/ldap.autoSync: "true"
spec:
  config:
    host: ldap.example.com:1636
    bindDN: cn=reader,dc=example,dc=com
    bindPW: <BIND_PASSWORD>
    insecureNoSSL: false
    insecureSkipVerify: false
    startTLS: false
    rootCAData: <BASE64_ENCODED_CA_CERTIFICATE_PEM>
    usernamePrompt: Directory username
    userSearch:
      baseDN: ou=People,dc=example,dc=com
      filter: "(&(objectClass=inetOrgPerson)(uid=*))"
      scope: sub
      idAttr: uid
      nameAttr: uid
      username: uid
      # emailAttr: mail
      # preferredUsernameAttr: cn
      # phoneAttr: telephoneNumber
      # emailSuffix: test.com
    groupSearch:
      baseDN: ou=Groups,dc=example,dc=com
      filter: "(&(objectClass=groupOfNames)(member=*))"
      scope: sub
      groupAttr: member
      userAttr: DN
      nameAttr: cn
```

Top-level and metadata fields:

Field	Required	Product behavio
`apiVersion`	Yes	Use `dex.coreos.com`
`kind`	Yes	Use `Connector`
`id`	Yes	Connector identity Keep it equal to `metadata.name`
`name`	Yes	Display name sho on the ACP login chooser.
`type`	Yes	Use `ldap`.
`metadata.name`	Yes	Kubernetes resou name and ACP Connector lookup
`metadata.namespace`	Yes	Use `cpaas-syst`
`metadata.annotations["cpaas.io/description"]`	No	Description show ACP.
`metadata.annotations["cpaas.io/idp.validation"]`	Temporary troubleshooting only	String value `"false"` skips directory connecti validation for this creation or spec-changing update; format validation runs. Do not set it `"true"`.
`metadata.labels["cpaas.io/idp.version"]`	Yes	Use `v2` for the mappings docum here.

Field	Required	Product behavio
`metadata.labels["cpaas.io/ldap.autoSync"]`	No	String value `"tr` or `"false"`; ena scheduled synchronization.
`metadata.annotations["cpaas.io/ldap.autoSyncRule"]`	When auto sync is enabled	Five-field cron expression evalua in UTC.

Connection and transport fields:

Field	Required	Product behavior
`host`	Yes	Directory host and port reachable from the ACP cluster.
`bindDN`	Yes	Bind account DN used to read users and groups.
`bindPW`	Initial creation	Plain input accepted by the ACP YAML page. ACP removes it and stores the password in a managed Secret.
`clientSecretRef`	Stored output	Managed Secret reference written by ACP. Do not submit it together with `bindPW` or edit it manually.
`insecureNoSSL`	Yes	`true` selects plain LDAP; `false` selects TLS according to `startTLS`.
`startTLS`	Yes	`false` with `insecureNoSSL: false` selects LDAPS. `true` selects StartTLS over LDAP.
`insecureSkipVerify`	TLS connections	`false` requests server certificate verification, subject to the port-636 limitation below.
`rootCAData`	Private-CA strict TLS	Base64-encoded PEM CA certificate used by the Connector trust pool.

Field	Required	Product behavior
`usernamePrompt`	No	Text shown above the LDAP username field on the login page.

`userSearch` fields:

Field	Required	Product behavior
`baseDN`	Yes	Search boundary for user entries.
`filter`	Yes	LDAP Filter used to select the synchronized user population.
`scope`	No	`sub` searches the full subtree and is the runtime default; `one` searches only one level below `baseDN`.
`nameAttr`	Yes for v2	Stable unique value used as the ACP username and primary identifier during synchronization.
`idAttr`	Yes	Validation and compatibility identity field. For v2, set it to the same stable attribute as `nameAttr`.
`username`	Yes	One or more comma-separated attributes used to find a user during login.
`emailAttr`	No	Copies the directory value to the ACP user's `mail` field. Configure it only when the filtered users provide that attribute.
`preferredUsernameAttr`	No	Supplies the preferred-username Claim during LDAP login; it does not replace the v2 synchronized identity from `nameAttr`.
`phoneAttr`	No	Copies the directory value to the ACP user's mobile field. Configure it only when the filtered users provide that attribute.

Field	Required	Product behavior
`emailSuffix`	No	During LDAP login, constructs the identity email as `<nameAttr>@<suffix>`; enter the suffix without `@`.

> **Warning**
>
> `emailSuffix` and v2 synchronization
>
> In the validated v2 behavior, `emailSuffix: test.com` produced `admin@test.com` for a first-login identity, while manual synchronization still stored `nameAttr` without the suffix as the primary identifier. Leave `emailSuffix` unset unless this difference has been tested against the customer's synchronization and collision requirements.

`groupSearch` fields:

Field	Required when group synchronization is enabled	Product behavior
`baseDN`	Yes	Search boundary for group entries.
`filter`	Yes	LDAP Filter used to select groups. Requiring the membership attribute avoids validation against empty groups.
`scope`	No	`sub` is the runtime default; `one` searches one level below `baseDN`.
`groupAttr`	Yes	Attribute on the group entry containing member values.
`userAttr`	Yes	Attribute or `DN` value from the user entry compared with `groupAttr`.
`nameAttr`	Yes	Attribute used as the ACP group display name, normally `cn`.

Omit `groupSearch` entirely when ACP group synchronization is not required.

Temporary validation bypass

Use `cpaas.io/idp.validation: "false"` only when the Connector must be staged while the directory is temporarily unreachable. ACP still checks the configuration structure, but it skips network, TLS, bind, Base DN, Filter, and mapped-attribute checks. The resulting Connector can therefore exist even though synchronization and login cannot work.

The admission webhook removes this annotation from the stored Connector and copies the skip marker to the managed credential Secret. A later spec-changing Connector update without the annotation runs normal Connector validation. To restore validation for future credential changes, update the credential through the Connector without the annotation; do not edit the managed Secret directly. Do not use `cpaas.io/idp.validation: "true"`; omit the annotation for normal validation.

Before rollout, complete a normally validated creation or real corrective update, then run manual synchronization and a representative login. A metadata-only update is not evidence that normal validation ran.

Choose the transport flags as a set:

Transport	Typical port	`insecureNoSSL`	`startTLS`	Certificate verification
Plain LDAP	389	`true`	`false`	Not applicable; use only on a trusted network when policy permits
LDAPS on the standard port	636	`false`	`false`	The validated ACP implementation store `insecureSkipVeri` `true`; traffic is encrypted, but the LDAP server certific is not verified
LDAPS with strict certificate	A configured non-636	`false`	`false`	Keep `insecureSkipVeri`

Transport	Typical port	`insecureNoSSL`	`startTLS`	Certificate verification
verification	LDAPS port			`false` and provide trusted CA through `rootCAData`; this v validated on a non-6 port

The Active Directory template above uses the plain LDAP transport that was validated with ACP. Plain LDAP does not protect the bind password on the network. Use it only when the customer's network and security policy permit it. When TLS is required, first determine whether the policy also requires certificate verification: standard-port 636 provides encryption without server certificate verification in the validated implementation, while strict verification requires a non-636 LDAPS port and `rootCAData`. Complete an ACP LDAPS creation, synchronization, and login test before rollout.

For strict certificate verification on a non-636 LDAPS port, encode the PEM CA certificate without line breaks and use the output as `rootCAData`:

```
openssl base64 -A -in ldap-ca.pem
```

For scheduled synchronization in YAML, add the supported metadata to the Connector:

```
metadata:
  labels:
    cpaas.io/ldap.autoSync: "true"
  annotations:
    cpaas.io/ldap.autoSyncRule: "*/1 * * * *"
```

The example rule is suitable only for observing an acceptance-test run. Replace it with the required UTC schedule after validation. The top-level `id` must equal `metadata.name`; current scheduled synchronization looks up the Connector by that identity.

Troubleshoot by Symptom

Symptom	Product stage	Likely cause	Concrete check
ACP cannot create the Connector because it cannot connect.	Connector creation	Cluster DNS, routing, firewall, port, TLS mode, or certificate trust is incorrect.	From a host or Pod with ACP-equivalent reachability, run the external TCP or TLS checks below, then inspect `auth-controller2` logs.
Bind succeeds but Base DN or Filter validation fails.	Connector creation and search validation	The Base DN does not exist for the bind account, the Filter is invalid, or the Filter returns no usable entries.	Run an external base-scope query for the Base DN, then run the exact subtree Filter and inspect `auth-controller2` logs.
ACP rejects Active Directory **Group Attr** `member` and also suggests `member`.	Connector creation and group attribute validation	The Group Filter sampled empty groups that did not return a `member` attribute.	Use `(&(objectClass=group)(member=*))`, confirm that the external query returns the intended groups, and retry creation.
Sync reports success but returns fewer users than expected.	Manual or scheduled synchronization	The Filter excludes users, users are outside the Base DN, or required **User Name Attr** values are missing.	Compare the external subtree result with the expected count, run the missing-attribute Filters above, and inspect skipped-entry messages in `apollo` logs.
Connector creation succeeds but a directory user cannot log in.	Login	**Login Field** does not find exactly one entry, the account is excluded or disabled, or the representative credentials are invalid.	Query the representative login value with each configured login attribute, confirm exactly one result, retry in a separate browser session, and inspect `apollo` logs.
Users synchronize but groups do not.	Group synchronization	**Group Attr** and **User Attr** do not represent the values stored in the group, or the	Inspect one known group and reproduce `(<Group Attr>=<User Attr value>)` with an external

Symptom	Product stage	Likely cause	Concrete check
		Group Base DN or Filter is wrong.	subtree query, then inspect `apollo` group lookup messages.
Identities collide or change unexpectedly.	User synchronization and reconciliation	**User Name Attr** is duplicated, empty, mutable, or based on a DN that changed after a rename or move.	Query the proposed identity attribute across the full User Base DN, check missing and duplicate values, and select a stable unique attribute.
Automatic synchronization does not run.	Scheduled synchronization	Auto Sync metadata is absent or invalid, or top-level `id` differs from `metadata.name`.	Inspect the stored Connector metadata and identity, then inspect scheduled synchronization messages in `apollo` logs.

On-Site Diagnostic Tools

The tools in this section are external pre-integration diagnostics or advanced ACP evidence collection. They are not built-in ACP functions.

External: Linux and macOS LDAP Tools

The LDAP commands require an installed OpenLDAP client such as `ldapsearch` and `ldapwhoami`. They prompt for the bind password with `-W`. This option only prevents the password from appearing in the command line; it does not encrypt the network transport.

The private-CA environment variables below were validated with a Linux OpenLDAP client. The built-in macOS `/usr/bin/ldapsearch` and `/usr/bin/ldapwhoami` use the macOS system trust store instead; use a Linux diagnostic host or Pod, or configure the CA in the macOS Keychain before using those built-in clients.

For every `ldapwhoami` and `ldapsearch` command, use the same transport combination as the Connector being investigated:

Connector transport	Connector settings	OpenLDAP client transport	Bind credential protection
Plain LDAP	`insecureNoSSL: true`, `startTLS: false`	`-H ldap://ldap.example.com:389` without `-ZZ`	No TLS protection; use only when network policy permits
LDAPS	`insecureNoSSL: false`, `startTLS: false`	`-H ldaps://ldap.example.com:636` without `-ZZ`	TLS starts when the connection is established

The query examples below use plain LDAP and therefore match only a Connector with `insecureNoSSL: true`. For LDAPS, replace the `-H` value with the matching row before running each example. If the LDAP certificate is signed by a private CA, the diagnostic client must also trust that CA. On Linux OpenLDAP clients, prefix each `ldapwhoami` or `ldapsearch` command as follows:

```
LDAPTLS_REQCERT=demand \
LDAPTLS_CACERT=/path/to/ldap-ca.pem \
ldapwhoami -x -H ldaps://ldap.example.com:636 \
  -D 'cn=reader,dc=example,dc=com' -W
```

Without the `LDAPTLS_CACERT` setting, the validated Linux client failed with `unable to get local issuer certificate`; with the CA file, the same bind succeeded.

External command: test TCP port reachability only

```
nc -vz ldap.example.com 389
```

This command checks only whether the TCP port is reachable. It does not validate TLS, bind credentials, or LDAP search behavior. Use port `636` instead when checking LDAPS port reachability.

External command: validate the server, transport, and bind account

```
ldapwhoami -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W
```

External command: read the directory Root DSE

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b '' -s base '(objectClass=*)' namingContexts defaultNamingContext
```

External command: confirm that the candidate Base DN is readable

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b 'ou=People,dc=example,dc=com' -s base \
  '(objectClass=*)' dn
```

External command: reproduce the OpenLDAP user subtree search

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b 'ou=People,dc=example,dc=com' -s sub \
  '(&(objectClass=inetOrgPerson)(uid=*))' \
  dn uid cn mail objectClass
```

External command: find OpenLDAP entries missing the proposed identity

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b 'ou=People,dc=example,dc=com' -s sub \
  '(&(objectClass=inetOrgPerson)(!(uid=*)))' dn cn mail
```

External command: print duplicate values for the proposed OpenLDAP identity

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b 'ou=People,dc=example,dc=com' -s sub \
  '(&(objectClass=inetOrgPerson)(uid=*))' uid | \
  awk -F ': ' '$1 == "uid" { print $2 }' | \
  LC_ALL=C sort -f | uniq -di
```

Replace `uid`, the Base DN, and the Filter when evaluating another candidate identity attribute. No output means that no duplicate values were found among the returned entries. As with the other `ldapsearch` examples, use the transport combination that matches the Connector.

External command: reproduce a `groupOfNames` membership lookup

```
ldapsearch -LLL -x -H ldap://ldap.example.com:389 \
  -D 'cn=reader,dc=example,dc=com' -W \
  -b 'ou=Groups,dc=example,dc=com' -s sub \
  '(&(objectClass=groupOfNames)(member=uid=alice,ou=People,dc=example,dc=
com))' \
  dn cn member
```

External command: verify the LDAPS certificate chain with the intended CA

```
openssl s_client -connect ldap.example.com:636 \
  -servername ldap.example.com \
  -CAfile ldap-ca.pem -verify_return_error
```

The certificate check passes only when the output reports successful verification, such as `Verify return code: 0 (ok)` or `Verification: OK`.

External: Active Directory Queries with `ldapsearch`

The same OpenLDAP command-line client can inspect Active Directory without requiring Windows or RSAT. Use a directory account with read permission.

External command: inspect the intended Active Directory users

```
ldapsearch -LLL -x -H ldap://ad.example.com:389 \
  -D 'CN=ACP Reader,OU=Service Accounts,DC=example,DC=com' -W \
  -b 'OU=Employees,DC=example,DC=com' -s sub \
  '(&(objectClass=user)(!(objectClass=computer))(sAMAccountName=*)(!(user
AccountControl:1.2.840.113556.1.4.803:=2)))' \
  dn sAMAccountName displayName mail userAccountControl memberOf
```

External command: reproduce an Active Directory group membership lookup

```
ldapsearch -LLL -x -H ldap://ad.example.com:389 \
  -D 'CN=ACP Reader,OU=Service Accounts,DC=example,DC=com' -W \
  -b 'OU=Groups,DC=example,DC=com' -s sub \
  '(&(objectClass=group)(member=CN=Alice,OU=Employees,DC=example,DC=co
m))' \
  dn cn member
```

Advanced: ACP Component Logs

These deployment names are the current ACP component names. The commands require permission to read cluster logs.

Advanced ACP command: inspect Connector creation validation

```
kubectl -n cpaas-system logs deployment/auth-controller2 --since=10m
```

`auth-controller2` covers Connector creation validation, including network connection, TLS, bind, Base DN, Filter, and mapped attributes.

Advanced ACP command: inspect synchronization and login behavior

```
kubectl -n cpaas-system logs deployment/apollo --since=10m
```

`apollo` covers manual and scheduled synchronization, skipped entries, group lookups, and login behavior.

NOTE

Success with an external tool does not prove that ACP can reach or use the directory. Complete the evidence chain with ACP Connector creation, expected synchronization results, and a real directory login with the intended ACP permission.

OIDC Management

ACP can redirect users to a standards-based OIDC identity provider and create or update their platform identity after successful login.

Use the **Form** tab for the shortest initial integration. Use the **YAML** tab when the returned username Claim must be selected explicitly, Claims must be remapped, UserInfo must be queried, or OIDC groups and extra fields are required. See Add OIDC Using YAML.

TOC

Before You Begin

Prepare these values:

- Issuer URL, labeled in ACP as **Server Provider URL**.
- Client ID.
- Client Secret, labeled in ACP as **Client Key**.
- One representative test account with an email address that is not already used by an ACP local user or another IDP.
- A stable username Claim returned for that test account. The tested YAML fallback uses `preferred_username`.

Register this exact callback with the upstream identity provider:

```
https://<ACP_ADDRESS>/dex/callback
```

ACP derives the callback from the platform address. The form does not contain a Redirect URI field.

Server Provider URL must be the Issuer root, not an authorization endpoint, token endpoint, or login page. Its discovery document is normally available at this path:

```
<ISSUER_URL>/.well-known/openid-configuration
```

You can inspect discovery from a diagnostic host with ACP-equivalent reachability:

```
curl -fsS 'https://idp.example.com/.well-known/openid-configuration'
```

For an upstream service that uses a private CA, independently verify the certificate chain with that CA:

```
curl -fsS --cacert /path/to/oidc-ca.pem \
  'https://idp.example.com/.well-known/openid-configuration'
```

This external query is preliminary evidence. ACP Connector creation is the cluster-side discovery and reachability check.

Confirm the Returned Claims

Ask the upstream identity administrator to provide the ID token or UserInfo Claim names for one representative account. Claim names are case-sensitive. ACP Connector creation validates discovery and client reachability, but it does not prove that a real user's token contains the required Claims.

Claim or setting	Required when	ACP use
`sub`	Always	Stable upstream subject used by the OIDC protocol. It must be a string.
`name`	No `userNameKey` is configured	Default source for the ACP username.
Claim selected by `userNameKey`	The default `name` Claim is absent or unsuitable	Source for the ACP username. The validated fallback is `preferred_username`.
`email` or the Claim selected by `claimMapping.email`	The `email` scope is requested	ACP user identity. It must be a string and should not collide with an existing ACP user.
`email_verified`	The `email` scope is requested	Boolean verification flag required by the current login flow.
Claim selected by `claimMapping.groups`	OIDC group processing is enabled	String or array of group names used to create and associate ACP groups.
Claims selected by `claimMapping.phone` and `claimMapping.mail`	Those optional profile fields are configured	String values copied to the ACP user's mobile and mail fields.
Claim listed in `claimExtra`	The extra value must be stored in ACP	Value copied to `spec.extra` when its configured type matches. This guide documents the validated `string` type.

Complete one real browser login before rollout and inspect the generated ACP user. That login is the authoritative check for Claim availability, mapping, collision behavior, and group

restrictions.

> **Upstream OIDC certificate verification limitation**
>
> In the ACP environment validated for this guide (`auth-controller2` and `apollo` v4.3.6), Connector creation accepted an HTTPS Issuer whose self-signed CA was not trusted by a normal client, and the stored Connector contained `insecureSkipVerify: true`. ACP therefore does not verify the upstream OIDC server certificate in this implementation. HTTPS still encrypts the connection, but successful Connector creation is not evidence that the certificate chain is trusted.
>
> If the customer's security policy requires upstream server certificate verification, the validated implementation does not meet that requirement. Resolve the product-version or security-policy requirement before rollout.

Add OIDC Using the Form

1. Go to **Users > IDP**.
2. Click **Add OIDC**.
3. Complete **Basic Info**: **Name**, **Display Name**, and optional **Description**.
4. Complete **Server Setting**: **Server Provider URL**, **Client ID**, and **Client Key**.
5. Leave **Logout URL** empty for this validated integration workflow.
6. Click **Add**.

During creation, ACP reads OIDC discovery from the Issuer URL. An invalid or unreachable Issuer is rejected at this stage. The current creation check does not prove that **Client Key** is correct or that the upstream TLS certificate is trusted: a Connector with an incorrect key can still be created and then fail during the browser callback, while an untrusted self-signed certificate can be accepted. A real login is therefore required for authentication, and certificate compliance must be assessed separately as described above.

After creation, ACP moves the submitted **Client Key** to a managed Secret. The stored Connector references that Secret instead of retaining `clientSecret` in its configuration.

Console Field Reference

ACP console field	YAML field	Product behavior
Name	`id` and `metadata.name`	Use the same DNS-compatible value for both fields.
Display Name	`name`	Name shown on the ACP IDP chooser.
Description	`metadata.annotations["cpaas.io/description"]`	Optional operator-facing description.
Server Provider URL	`spec.config.issuer`	Issuer root used for OIDC discovery.
Client ID	`spec.config.clientID`	Client identifier registered upstream.
Client Key	`spec.config.clientSecret`	Initial-creation input; ACP stores it in a managed Secret.
Logout URL	`logoutUrl`	Leave empty. The validated runtime did not use it as the post-logout redirect.

The form does not expose `redirectURI`, scopes, UserInfo, Claim mappings, group processing, or extra Claims. Configure those fields through YAML only after confirming the upstream response.

Verify the Integration

1. Start a separate browser session without an ACP login state.
2. Open ACP and select the configured OIDC **Display Name**.
3. Complete authentication at the upstream identity provider.
4. Confirm that the browser returns to ACP instead of an HTTP 500 error.
5. In **Users**, find the generated user and verify its IDP source, email, username, and active state.
6. If the username is empty because the upstream response does not contain the default `name` Claim, recreate or update the Connector with the YAML example below and repeat the login.
7. Assign a minimal role with Manage User Roles.
8. Log in again and confirm that the assigned ACP permission is available.

> **Email identity collisions**
>
> In the validated ACP behavior, logging in through another OIDC Connector with the same mapped email updated and reassociated the existing ACP user with the newer Connector. It did not create a second user or reject the collision. Use a dedicated, non-colliding account for acceptance testing, and check existing ACP users before rollout.

Connector creation, upstream authentication, ACP user creation, and ACP authorization are separate checkpoints. Verify each checkpoint independently.

Add OIDC Using YAML

Use **Users > IDP > Add OIDC > YAML** when the upstream response requires explicit Claim selection or advanced mapping. The complete example below assumes that the upstream response contains `sub`, `preferred_username`, `email`, `email_verified`, `groups`, and `phone_number`, and that the test user belongs to `acp-users`. Remove optional mappings and group restrictions that the real upstream response does not satisfy.

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: corporate-oidc
name: Corporate OIDC
type: oidc
metadata:
  name: corporate-oidc
  namespace: cpaas-system
  annotations:
    cpaas.io/description: Corporate OIDC
    # cpaas.io/idp.validation: "false"
  labels:
    cpaas.io/idp.version: v2
spec:
  config:
    issuer: https://idp.example.com
    clientID: <CLIENT_ID>
    clientSecret: <CLIENT_SECRET>
    redirectURI: https://acp.example.com/dex/callback
    scopes:
      - profile
      - email
    getUserInfo: true
    userNameKey: preferred_username
    overrideClaimMapping: true
    claimMapping:
      preferred_username: preferred_username
      email: email
      groups: groups
      phone: phone_number
      mail: email
    claimExtra:
      - field: preferred_username
        type: string
    insecureEnableGroups: true
    allowedGroups:
      - acp-users
```

Replace the Issuer, credentials, ACP address, Claim names, and allowed group before submission. If the provider does not expose a UserInfo endpoint or all required Claims are

already in the ID token, set `getUserInfo: false`. If group processing is not required, remove `insecureEnableGroups`, `allowedGroups`, and `claimMapping.groups`.

Do not add `openid` to `scopes`; the Connector adds it to the authorization request. If the upstream provider requires a provider-specific scope for group Claims, add that scope only after confirming it in the discovery and provider configuration.

Submit this initial configuration through the ACP console YAML tab. ACP moves `clientSecret` to a managed Secret, and the stored Connector contains `clientSecretRef` instead. Do not use `kubectl apply` with an inline `clientSecret`: the Kubernetes last-applied annotation can retain the submitted secret in the Connector metadata.

Complete OIDC Field Reference

Top-level and metadata fields:

Field	Required	Product behavior
`apiVersion`	Yes	Use `dex.coreos.com/v1`
`kind`	Yes	Use `Connector`.
`id`	Yes	Connector identity. Keep it equal to `metadata.name`.
`name`	Yes	Display name shown on the ACP login chooser.
`type`	Yes	Use `oidc`.
`metadata.name`	Yes	Kubernetes resource name and ACP Connector lookup key
`metadata.namespace`	Yes	Use `cpaas-system`

Field	Required	Product behavior
`metadata.annotations["cpaas.io/description"]`	No	Description shown by ACP.
`metadata.annotations["cpaas.io/idp.validation"]`	Temporary troubleshooting only	String value `"false"` skips upstream connectivity validation for this creation or spec-changing update; format validation still runs. Do not set it to `"true"`.
`metadata.labels["cpaas.io/idp.version"]`	Recommended	Use `v2`; current AC also adds this label during creation.

Connection and login fields:

Field	Required	Product behavior
`issuer`	Yes	Issuer root used to retrieve `/.well-known/openid-configuration`.
`clientID`	Yes	Client identifier registered upstream.
`clientSecret`	Initial creation	Plain input accepted by the ACP YAML page. ACP removes it and stores it in a managed Secret.
`clientSecretRef`	Stored output	Managed Secret reference written by ACP. Do not submit it together with `clientSecret` or edit it manually.
`redirectURI`	Yes	Exact ACP callback registered upstream: `https://<ACP_ADDRESS>/dex/callback`.
`scopes`	No	Additional scopes such as `profile` and `email`. ACP automatically adds `openid`.

Field	Required	Product behavior
`getUserInfo`	No	When `true`, requests the UserInfo endpoint and merges returned Claims before mapping. Verify that the provider supports this endpoint.
`userNameKey`	No	Claim used as the ACP username instead of the default `name`; `preferred_username` is the validated fallback.

Claim and group fields:

Field	Required	Product behavior
`overrideClaimMapping`	When forcing a custom mapping	Set to `true` when a standard Claim is also present but ACP must use the Claim selected below.
`claimMapping.preferred_username`	No	Claim used as the preferred username.
`claimMapping.email`	When remapping email	Claim used as the ACP primary identity. Check for collisions before rollout.
`claimMapping.groups`	When group processing is enabled	Claim containing a string or array of group names. This nested field is the runtime group mapping.
`claimMapping.phone`	No	String Claim copied to the ACP user's mobile field.
`claimMapping.mail`	No	String Claim copied to the ACP user's mail field.
`claimExtra[].field`	No	Claim copied to the ACP user's `spec.extra` map.

Field	Required	Product behavior
`claimExtra[].type`	With `claimExtra[].field`	Use `string` for the validated workflow in this guide. Remove the entry when the Claim is absent or has another type.
`insecureEnableGroups`	When group processing is required	Set to `true` to read the mapped group Claim and create or associate ACP groups during login.
`allowedGroups`	No	When non-empty, login is rejected unless at least one mapped group matches. ACP retains only the matching allowed groups.

If `email` is present in `scopes`, the selected email Claim must be a string and `email_verified` must be a Boolean in the merged response. Missing either value causes login to fail.

Temporary validation bypass

Use `cpaas.io/idp.validation: "false"` only when the Connector must be staged while the upstream service is temporarily unreachable. ACP still checks the configuration structure, but it skips discovery and upstream connectivity checks. The resulting Connector can therefore exist even though browser login cannot work.

The admission webhook removes this annotation from the stored Connector and copies the skip marker to the managed credential Secret. A later spec-changing Connector update without the annotation runs normal Connector validation. To restore validation for future Client Key changes, update the credential through the Connector without the annotation; do not edit the managed Secret directly. Do not use `cpaas.io/idp.validation: "true"`; omit the annotation for normal validation.

Before rollout, complete a normally validated creation or real corrective update, then perform a real browser login and inspect the generated ACP user. A metadata-only update is not evidence that

normal validation ran.

ACP-managed and ineffective fields

Stored YAML can contain controller-added defaults such as `issuerAlias: ""`, top-level `groupsKey: groups`, and `insecureSkipVerify: true`.

- Configure group mapping with `claimMapping.groups`. The current runtime does not use the controller-added top-level `groupsKey` as the group Claim selector.
- Do not use `insecureSkipVerify: false` as a certificate-trust control. The validated runtime does not verify the upstream certificate, as described in **Before You Begin**.
- Leave `issuerAlias` out of the validated workflow unless a product-specific integration procedure explicitly requires it.

Login, Update, and Delete Behavior

- OIDC users are created or updated after a successful browser login. OIDC does not provide LDAP-style manual or scheduled user synchronization.
- After updating the username mapping, start a new browser session and log in again to verify the updated user.
- Deleting the Connector through ACP without cleanup retains source users and marks them invalid.
- Selecting **Clean up IDP users and User Groups** deletes the generated source users and the managed Secret with the Connector.

Use the ACP console action or ACP product API for deletion. Deleting the underlying Connector custom resource directly with `kubectl` bypasses the tested ACP user-invalidation and cleanup behavior.

Logout Behavior

In the validated current flow, ACP logout ended the ACP session and returned the browser to the ACP IDP chooser. Selecting the same OIDC entry again reused the still-active upstream session and signed the user in without another credential prompt. ACP logout therefore does not prove that the upstream session has ended.

The tested flow did not use the configured **Logout URL** as a browser redirect. Leave this field empty and validate upstream session termination separately when the customer's security requirements require it.

Troubleshoot by Stage

Symptom	Proven stage	Next check
ACP rejects the Connector during creation.	Discovery or cluster-side reachability did not complete.	Confirm that **Server Provider URL** is the Issuer root, retrieve its discovery document from an ACP-equivalent network path, and retry creation.
Connector creation succeeds, but the callback returns HTTP 500 with `unauthorized_client` or `Invalid client secret`.	Discovery and the upstream authorization page worked; client authentication at the token endpoint failed.	Correct **Client Key**, then repeat the full login in a new browser session. Creation alone does not validate this value.
Login succeeds, but the ACP username is empty.	ACP accepted the upstream identity and created the user.	Confirm that `preferred_username` is present, set `userNameKey: preferred_username` through YAML, and log in again.
Login fails with a missing email or `email_verified` message.	Authorization and token exchange completed, but required Claims were absent or had the wrong type.	Inspect the ID token and UserInfo response, correct `scopes` or `claimMapping.email`, and repeat login.
Login fails with `user not a member of allowed groups`.	The mapped group Claim was read, but no	Confirm the exact case-sensitive group values, then correct the upstream membership,

Symptom	Proven stage	Next check
	value matched `allowedGroups`.	`claimMapping.groups`, or `allowedGroups`.
Login succeeds, but no ACP groups are associated.	Authentication and user mapping completed without a usable group result.	Confirm that `insecureEnableGroups: true`, inspect the mapped group Claim type and values, and verify `claimMapping.groups`.
A configured extra Claim is absent from the ACP user.	Login completed, but the Claim was missing or did not match the configured type.	Inspect the merged ID token and UserInfo Claims; for this guide, use `type: string` only for a string Claim.
An existing user's IDP source or mapping changes after the test login.	The callback mapped an email already used by an ACP user.	Stop the rollout, choose a non-colliding test account, and review existing ACP users before retrying.
ACP logout returns to the IDP chooser, but choosing OIDC signs in without another prompt.	The ACP session ended, while the upstream session remained active.	Treat ACP and upstream logout as separate acceptance checks; do not rely on the unverified **Logout URL** field.
Users remain active after the Connector custom resource is deleted with `kubectl`.	The ACP product deletion workflow was bypassed.	Restore the test Connector if needed, then delete it with the ACP console action and select the required cleanup option.

For callback and user-mapping evidence, inspect the current ACP login component logs:

```
kubectl -n cpaas-system logs deployment/apollo --since=10m
```

Troubleshooting

Delete User

Problem Description

Solution

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'
   ```

User Policy

Introduction

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:

Policy	Description
	- Requires password change after specified period (e.g., 90 days) - Prevents login until password is updated
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

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

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.

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

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

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**.

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?

- 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.

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**.

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.

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

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

Telemetry

Install

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

Certificates

Automated Kubernetes Certific cert-manager OLM Certifi

Certificate Monitoring Rotate TLS

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
```

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 ↗

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

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

- 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-found=true -o yaml > /cpaas/dex-serving-cert.yaml
kubectl get secret -n cpaas-system dex.tls -o yaml > /cpaas/dex.tls.yaml
```

2. Delete the current certificates:

```
kubectl delete certificate -n cpaas-system dex-serving-cert --ignore-not-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/to/tls.key -n cpaas-system
```