安全

Alauda Container Security

Alauda Container Security

Alauda 集群认证

Alauda 集群认证

安全与合规

合规性 API Refiner 关于 Alauda

平台安全配置

更改平台 OIDC Client Secret

前提条件

OIDC Secret 概览

操作步骤

回滚

禁用 PKCE Plain 方法

前提条件

操作步骤

验证

回滚

用户与角色

用户

组

角色

IDP

用户策略

多租户（项目）

介绍

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

操作指南

审计

介绍

先决条件

操作步骤

搜索结果

遥测

安装

先决条件

安装步骤

启用在线运维

卸载步骤

证书

自动化 Kubernetes 证书轮换

安装

配置

工作原理

操作注意事项

cert-manager

Overview

How it works

Identifying cert-manager Managed Certific

Related Resources

OLM 证书

证书监控

证书状态监控

内置告警规则

轮换平台访问地址的 TLS 证书

前提条件

操作步骤

Alauda Container Security

Alauda Container Security 是一款面向 Kubernetes 和容器化环境的综合安全解决方案。它提供集中管理、自动化漏洞扫描、策略执行和合规性检查，帮助组织在多个集群中保障容器基础设施的安全。

Alauda Container Security 采用分布式、基于容器的架构，由中央服务（用于管理、API 和 UI）和安全集群服务（用于监控、策略执行和数据收集）组成。它可与 CI/CD 流水线、SIEM、日志系统集成，并支持内置的 Scanner V4 漏洞扫描器。

Note

因为 Alauda Container Security 的发版周期与灵雀云容器平台不同，所以 Alauda Container Security 的文档现在作为独立的文档站点托管在 Alauda Container Security↗。

Alauda 集群认证

Alauda Container Platform 集群认证插件为 global 集群故障场景提供独立的认证集成能力。

当 `global` 集群故障时，用户仍可通过该服务登录访问和操作 Kubernetes 集群，权限保持与 global 集群故障前一致。

> **Note**
>
> 因为 Cluster Authentication 的发版周期与灵雀云容器平台不同，所以 Cluster Authentication 的文档现在作为独立的文档站点托管在 Cluster Authentication ↗。

安全与合规

合规性

介绍

安装 Alauda Container Platform 使用指南

通过控制台安装

通过 YAML 安装

卸载流程

API Refiner

介绍

产品介绍

限制

安装 Alauda Container Platform API Refiner

通过控制台安装

通过 YAML 安装

卸载流程

默认配置

关于 **Alauda Container Platform** 合规服务

关于 **Alauda Container Platform** 合规服务

合规性

介绍

介绍

安装 Alauda Container Platform Policy Management

安装 Alauda Container Platform Policy Management

通过控制台安装

通过 YAML 安装

卸载流程

使用指南

私有镜像仓库访问配置

为什么 Kyverno 需要访问镜像仓库？

为 Pod 安全性应用 Security Con

简介

镜像签名验证

什么是镜像签名

快速开始

各角色职责

适用场景

先决条件

快速开始

常见用例

使用 Secrets 的镜像签名验证策略

为什么使用 Secrets 存储公钥？

快速开始

Secret 创建方式

常见使用场景

镜像仓库验证

什么是镜像仓库

快速开始

常见场景

高级模式

最佳实践

容器逃逸防护策略

什么是容器逃逸防护？

快速开始

核心容器逃逸防护策略

高级场景

测试与验证

最佳实践

Security Co

网络安全策略

什么是网络安全？

快速开始

核心网络安全策略

高级场景

测试与验证

Volume Security Policy

什么是卷安全？

快速开始

核心卷安全策略

高级场景

测试与验证

使用 Kyverno Generate 规则

Generate 的工作原理

示例场景

前提条件

准备目标资源依赖项

准备源资源

授予 Kyverno 权限

创建生成策略

验证生成规则

故障排查

介绍

ACP 基于开源组件 Kyverno 提供合规功能，使组织能够在其 Kubernetes 集群中定义和执行策略。

该功能解决了维护一致的安全性、治理和运营标准的挑战，允许用户使用 Kyverno 的 YAML 语法创建自定义策略，并自动验证资源是否符合这些策略。

合规功能提供全面的违规监控和报告能力，通过直观的界面提供资源级和策略级的合规违规视图，帮助团队快速识别不合规资源并采取适当的补救措施，以维持其期望的安全态势和法规合规性。

INFO

有关 Kyverno 的更多信息，请阅读 Kyverno Documentation ↗。

安装 Alauda Container Platform Policy Management

Alauda Container Platform Policy Management 是一项平台服务，用于集成 Kyverno，以管理 Alauda Container Platform 上的合规策略。

目录

通过控制台安装

通过 YAML 安装

1. 检查可用版本
2. 创建 ModuleInfo

卸载流程

通过控制台安装

1. 导航至 管理员
2. 在左侧导航栏中，点击 **Marketplace** > 集群插件
3. 搜索 **Alauda Container Platform Policy Management**，然后点击查看其详细信息
4. 点击 安装 以部署插件

通过 YAML 安装

1. 检查可用版本

在 `global` 集群中检查 ModulePlugin 和 ModuleConfig 资源，确保插件已发布：

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

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

这表示集群中存在 ModulePlugin `kyverno`，并且已发布版本 `v4.0.4`。

2. 创建 ModuleInfo

创建 ModuleInfo 资源，无需任何配置参数即可安装插件：

```
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  annotations:
    cpaas.io/display-name: kyverno
    cpaas.io/module-name: '{"en": "Alauda Container Platform Policy Manag
ement",
      "zh": "Alauda Container Platform Policy Management"}'
  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
```

字段说明：

- `name`：集群插件的临时名称。平台将在创建后根据内容重命名，格式为 `<cluster-name>-<hash of content>`，例如 `global-ee98c9991ea1464aaa8054bdacbab313`。
- `label cpaas.io/cluster-name`：指定应安装插件的集群。
- `label cpaas.io/module-name`：插件名称，必须与 ModulePlugin 资源匹配。
- `label cpaas.io/module-type`：固定字段，必须为 `plugin`；缺少此字段会导致安装失败。
- `.spec.config`：如果对应的 ModuleConfig 为空，此字段可以留空。
- `.spec.version`：指定要安装的插件版本，必须与 ModuleConfig 中的 `.spec.version` 匹配。

卸载流程

1. 按照安装流程中的步骤 1-3 找到插件
2. 点击 卸载 以移除插件

使用指南

私有镜像仓库访问配置

为什么 Kyverno 需要访问镜像仓库？

快速开始

为 Pod 安全性应用 Security Con

简介

各角色职责

适用场景

先决条件

镜像签名验证

什么是镜像签名

快速开始

常见用例

使用 Secrets 的镜像签名验证策略

为什么使用 Secrets 存储公钥？

快速开始

Secret 创建方式

常见使用场景

镜像仓库验证

什么是镜像仓库

快速开始

常见场景

高级模式

最佳实践

容器逃逸防护策略

什么是容器逃逸防护？

快速开始

核心容器逃逸防护策略

高级场景

测试与验证

最佳实践

Security Co

网络安全策略

什么是网络安全？

快速开始

核心网络安全策略

高级场景

测试与验证

Volume Security Policy

什么是卷安全？

快速开始

核心卷安全策略

高级场景

测试与验证

使用 Kyverno Generate 规则

Generate 的工作原理

示例场景

前提条件

准备目标资源依赖项

准备源资源

授予 Kyverno 权限

创建生成策略

验证生成规则

故障排查

私有镜像仓库访问配置

本指南演示如何配置 Kyverno 以访问私有容器镜像仓库。当 Kyverno 需要验证镜像签名或检查镜像详情时，它需要适当的凭证来访问私有仓库——就像进入安全建筑需要门禁卡一样。

目录

为什么 Kyverno 需要访问镜像仓库？

快速开始

1. 创建镜像仓库 Secret
2. 配置 Kyverno 使用该 Secret（推荐）
3. Kyverno Deployment 配置

为什么 **Kyverno** 需要访问镜像仓库？

当 Kyverno 需要执行以下操作时，会访问镜像仓库：

- 验证镜像签名：下载签名数据以检查镜像是否被正确签名
- 检查镜像元数据：读取镜像标签、注解和清单信息
- 扫描漏洞：下载镜像进行安全扫描
- 验证镜像内容：检查容器镜像内部的实际内容

可以把它想象成一个需要检查身份证的保安——Kyverno 需要“看到”镜像才能验证它们。

快速开始

1. 创建镜像仓库 Secret

```
# 针对公司的私有镜像仓库
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. 配置 Kyverno 使用该 Secret（推荐）

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

3. Kyverno Deployment 配置

如果需要更细粒度的控制，可以直接修改 Kyverno 的 Deployment：

```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"  # 启用凭证助手
        # ... 其他配置
```

为 Pod 安全性应用 Security Context Constraints

本指南面向平台管理员和安全管理员。它介绍如何在现有 Kyverno 部署之上安装一个 SecurityContextConstraints (SCC) 引擎，以及如何将 SCC 配置文件绑定到 ServiceAccounts、Users 和 Groups，从而在准入时自动强制执行 Pod 安全边界。

目录

简介

各角色职责

适用场景

先决条件

步骤

第 1 部分：安装 SCC 引擎

步骤 1.1 — 安装 `SecurityContextConstraints` CRD

步骤 1.2 — 安装 13 个内置 SCC 配置文件

步骤 1.3 — 安装 GlobalContextEntries、Kyverno 读取 RBAC，以及 admission policies

步骤 1.4 — 以 Warn → Deny 的方式安全上线

步骤 1.5 — 验证引擎已就绪

第 2 部分：授权工作负载使用 SCC

步骤 2.1 — 选择合适的 SCC 配置文件

步骤 2.2 — 将 SCC 绑定到 ServiceAccount

步骤 2.3 — 将 SCC 绑定到 User

步骤 2.4 — 将 SCC 绑定到 Group

步骤 2.5 — 使用 `alauda.io/required-scc` 固定到特定 SCC

步骤 2.6 — 验证绑定是否生效

结果

故障排查

了解更多

使用 `PolicyException` 临时绕过策略

引擎如何选择 SCC

OpenShift 命令映射

最佳实践：在现有 namespace 中授予特权例外

第 1 步 - 审计现有 SCC 授权

第 2 步 - 将 restricted 基线绑定到所有 namespace ServiceAccounts

第 3 步 - 创建专用 ServiceAccount

第 4 步 - 仅向专用 ServiceAccount 授予 `privileged`

第 5 步 - 防止其他工作负载复用该 ServiceAccount

第 6 步 - 将强制执行从 PSA 转移到 Kyverno SCC

第 7 步 - 在工作负载中固定获批的 SCC

第 8 步 - 验证例外是否隔离

回滚该例外

后续步骤

简介

OpenShift 的 SecurityContextConstraints (SCC) 模型允许集群管理员定义一组 Pod 安全配置文件库，然后授予受试对象（ServiceAccounts、Users、Groups）使用特定配置文件的权限。当一个 Pod 被准入时，平台会选择该受试对象被允许使用的最合适的 SCC，补齐缺失的默认值，并根据该配置文件验证 Pod。工作负载本身无需声明每一个安全字段——SCC 配置文件会替它完成这些工作。

原生 Kubernetes 没有内置等效能力。本指南会安装一个基于 Kyverno 的引擎，在任何已运行 Kyverno 的标准 Kubernetes 集群上复现 SCC 体验。它使用：

- 一个 `SecurityContextConstraints` CRD（`security.alauda.io/v1alpha1`）来存储 SCC 配置文件。
- 标准 Kubernetes RBAC（`use` 动词加 `resourceNames`）将受试对象绑定到配置文件，因此操作流程与 OpenShift 保持一致（`oc adm policy add-scc-to-user` 模式可一一对应）。
- 一对 Kyverno 准入策略——一个变更型、一个校验型——用于选择正确的 SCC、补齐默认值，并拒绝任何没有可接受 SCC 的 Pod。
- 五个 `GlobalContextEntry` 资源，它们在内存中缓存 SCC 配置文件以及相关 RBAC 对象，因此准入决策不需要额外的 API 调用。

结果是：应用团队可以继续编写简单直接的 Pod 清单，集群会自动将其约束到其 ServiceAccount 被允许使用的安全配置文件中，而从 OpenShift 迁移时也无需更改绑定模型。

SCC 授权是一项安全控制变更。应用团队不应被授予直接创建或修改 SCC RBAC 绑定的权限，因为这样会让他们绕过集群安全边界。应用团队应描述工作负载需求，例如 `anyuid`、`hostNetwork` 或 `hostPath`；平台或安全管理员审查请求，并将最小权限 SCC 绑定到相应受试对象。

各角色职责

使用下表来判断本指南中的哪些部分适用于你。

角色	你的工作	你不应做的事
平台管理员或安全管理员	安装 SCC 引擎、审批 SCC 请求、创建 SCC RBAC 绑定、将校验策略从 `Warn` 切换到 `Deny`，以及审计例外情况。	不要在没有工作负载层面的明确理由和负责人时授予过宽的 SCC，例如 `privileged`、`hostaccess` 或 `anyuid`。
应用经理或应用所有者	识别工作负载所需能力，例如 root UID、host networking、host ports、host paths、user namespaces 或固定 UID 范围。向平台或安全管理员提供 namespace、ServiceAccount、工作负载名称和原因。审批后，使用分配的 ServiceAccount 部署工作负载。	不要创建 SCC RBAC 绑定，也不要向自己的 ServiceAccount 授予 SCC 权限。除非请求的 SCC 已经获批并完成绑定，否则不要使用 `alauda.io/required-scc`。

如果你是平台或安全管理员，请按照第 1 部分和第 2 部分执行。如果你是应用经理，请先使用步骤 2.1 准备 SCC 请求，然后仅在管理员批准并完成绑定后，才使用步骤 2.5 和步骤 2.6。不要自行应用步骤 2.2 到步骤 2.4 中的 RBAC 清单。

标准工作流如下：

1. 应用经理识别工作负载需求和目标 ServiceAccount。
2. 平台或安全管理员选择最小权限 SCC 并创建 RBAC 绑定。
3. 应用经理使用获批的 ServiceAccount 部署工作负载，只有在管理员要求固定到某个特定 SCC 时，才添加 `alauda.io/required-scc`。
4. 管理员使用 `kubectl auth can-i` 验证授权，而工作负载所有者验证准入后的 Pod 具有预期的 `alauda.io/scc` 注解。

适用场景

当出现以下任一情况时，应用本指南：

- 你正在将工作负载从 OpenShift 迁移过来，并希望保留现有的 `oc adm policy add-scc-to-*` 绑定模型，以便平台团队和审计工具继续按原样工作。
- 你已经在使用 Kyverno，并且需要一个集中管理的安全边界，而不要求每个 Pod 清单都声明完整的 `securityContext`。
- 你运行的是多租户集群，并希望不同 namespace 中的不同 ServiceAccount 具有不同的安全上限——例如，应用 SA 仅限 `restricted-v2`，日志收集 SA 允许 `hostmount-anyuid`，入口控制器 SA 允许 `NET_BIND_SERVICE`。
- 你希望在集群范围内有一个统一位置来表达和审计“谁被允许运行 privileged Pod”，而不是把例外分散到每个 namespace 中。

先决条件

开始前，请确保满足以下所有条件：

1. Kubernetes 集群运行 **1.30** 或更高版本（CEL admission 已稳定）。

2. Kyverno 已经安装并运行，版本为 **v4.3.1** 或更高，并且可用 `MutatingPolicy`、`ValidatingPolicy` 和 `GlobalContextEntry` CRD。你可以使用以下命令验证：

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

3. `kyverno` namespace 中包含以下 ServiceAccounts（默认 Kyverno 安装）：
   - `kyverno-admission-controller`
   - `kyverno-background-controller`
   - `kyverno-reports-controller`
4. 你拥有 **cluster-admin**（或等效）权限，因为安装引擎需要创建 CRD、ClusterRoles、ClusterRoleBindings、GlobalContextEntries 和 admission policies。
5. 你已经审查了打算允许非 `restricted` Pod 的每个 namespace 上的 Pod Security Admission (PSA) `enforce` 标签。PSA 会在 Kyverno 之前运行；如果某个 namespace 标记为 `pod-security.kubernetes.io/enforce: restricted`，那么在 Kyverno 介入前，该 namespace 会拒绝任何与宽松 SCC（例如 `anyuid` 或 `hostnetwork-v2`）匹配的 Pod。请在适当的 namespace 中将标签调整为 `baseline` 或 `privileged`，或者限制你在这些 namespace 中提供的 SCC 配置文件集合。

> **Tip**
>
> 引擎安装是一次性工作，通常由平台管理员执行。第 2 部分同样属于管理员工作流：平台或安全管理员在审查工作负载需求后绑定 SCC 配置文件。应用团队通常只需提供这些需求，然后使用分配的 ServiceAccount。

步骤

工作分为两部分：

- 第 **1** 部分 在集群范围内安装 SCC 引擎。每个集群执行一次。
- 第 **2** 部分 通过将 SCC 配置文件绑定到 ServiceAccounts、Users 和 Groups 来授权工作负载在需要时固定到特定 SCC。

第 1 部分：安装 SCC 引擎

步骤 1.1 — 安装 `SecurityContextConstraints` CRD

将以下清单保存为 `scc-crd.yaml`。它定义了一个集群级别的 `SecurityContextConstraints` 资源（短名 `scc`），其字段与 OpenShift SCC 语义一致。

```yaml
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
```

应用它并等待 CRD 变为 `Established` 后再继续：

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

步骤 1.2 — 安装 13 个内置 SCC 配置文件

将以下清单保存为 `scc-profiles.yaml`。它定义了 13 个基于 OpenShift 内置集合建模的 SCC 配置文件，按限制程度从高到低排序（`restrictiveScore: 100` 到 `restrictiveScore: 0`）。当同一受试对象被授予多个 SCC 时，自动选择策略会优先选择更高的 `restrictiveScore`。

> **Tip**
>
> 你不必安装全部配置文件。可以将此清单裁剪为平台实际提供的子集——但你必须至少为每个受试对象保留一个可用配置文件，否则它们的 Pod 会在准入时被拒绝。

```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
```

```
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:
    - '*'
```

应用这些配置文件：

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

你应该会看到全部 13 个配置文件列出，并且其 `Priority` 和 `Score` 列已填充（以及其他 SCC 列，如 `Priv`、`RunAsUser` 和 `Volumes`）。

步骤 1.3 — 安装 GlobalContextEntries、Kyverno 读取 RBAC，以及 admission policies

此步骤一次性安装三项内容：

1. **GlobalContextEntries (GCE)** —— 五个内存缓存，Kyverno 在准入过程中使用它们查找 SCC 配置文件、ClusterRoles、ClusterRoleBindings、RoleBindings 和 Roles，而无需为每次请求发起 API 调用。
2. **Reader RBAC** —— 一个 ClusterRole，授予 Kyverno 的三个服务账户对 SCC CRD、上述四类 RBAC 资源，以及策略所匹配的 Pod / `pods/ephemeralcontainers` 资源的只读权限。
3. 两个 **admission policies** —— 一个 `MutatingPolicy`，用于用所选 SCC 的默认值填充字段；以及一个 `ValidatingPolicy`，用于拒绝任何没有可接受 SCC 的 Pod。

> **Warning**
>
> 这两个策略包含驱动 SCC 选择与验证的 CEL 逻辑。你不需要阅读或理解 CEL 也能使用该引擎——按原样应用这些清单即可。之所以表达式很长，是因为它们逐字段复现了 OpenShift SCC 的准入算法。

将以下内容保存为 `scc-gce.yaml` 并应用：

```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: "@"
```

将以下内容保存为 `scc-reader-rbac.yaml` 并应用。由于 Kyverno 在策略就绪门控（`RBACPermissionsGranted`）期间会检查每个匹配资源的读取权限，因此必须包含 `pods` 和 `pods/ephemeralcontainers` 的读取权限；否则变更型策略会保持 `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
```

将以下内容保存为 `scc-auto-pick.yaml`。这是一个 `ValidatingPolicy`，用于拒绝任何没有可接受 SCC 的 Pod。

> **Warning**
>
> 下面的示例配置为 `validationActions: [Deny]`。在现有集群上，第一次应用前请将其改为 `validationActions: [Warn]`，然后在审查警告并创建所需 SCC 绑定后，再改回 `Deny`。有关滚动发布流程，请参见步骤 1.4。

```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.orV
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(",") + ")")
```

将以下内容保存为 `scc-fill-defaults.yaml` 并应用。这是一个 `MutatingPolicy`，它会在 Pod 上记录所选 SCC（`alauda.io/scc` 注解），并填充从该 SCC 继承而来的 `runAsUser`、`seccompProfile` 和 `allowPrivilegeEscalation` 默认值。

```
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
                }
              })
            }
          }
```

在两个策略上都保留 `reports.kyverno.io/disabled: "true"`。SCC 选择依赖准入请求中的 User、Groups 和 ServiceAccount，而后台报告扫描没有等效的请求身份。变更型策略也只会在准入期间填充默认值。将这些仅限准入的策略排除在后台报告之外，可以避免产生误导性的 PolicyReports，也避免对每个现有 Pod 进行不必要的重新编译。

两个策略默认都会跳过以下 namespace：以 `kube-`、`cpaas-` 或 `alauda-` 开头的 namespace，以及 `kyverno`、`cattle-system`、`operators` 和 `default`。如果你的平台使用不同的系统 namespace，请相应调整两个策略中的 `skip-system-ns` 表达式。

步骤 1.4 — 以 Warn → Deny 的方式安全上线

校验策略通过 `failurePolicy: Fail` 和 `validationActions: [Deny]` 交付，这意味着它会立即拒绝不合规 Pod。在现有集群上，如果在未做准备的情况下直接启用，可能会破坏那些 ServiceAccount 还未绑定任何 SCC 的工作负载。

请采用三阶段滚动发布：

1. 在首次应用前先使用 **Warn**。在现有集群上应用 `scc-auto-pick.yaml` 之前，将 `validationActions` 改为：

   ```
   validationActions:
     - Warn
   ```

   然后应用该文件。此时，策略会在每个原本会被拒绝的准入响应中附加警告，但仍然允许 Pod 通过。观察 Kyverno admission controller 日志以收集受影响的工作负载：

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

2. 修复。针对每个被警告的工作负载，添加或修正 RBAC 绑定，使其 ServiceAccount 可以 `use` 一个合适的 SCC（参见第 2 部分）。使用以下命令确认：

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

3. **Deny**。当合法工作负载不再产生警告后，将其切回 `Deny` 并重新应用：

   ```
   validationActions:
     - Deny
   ```

> **Tip**
>
> 如果你需要临时豁免整个 namespace，可以将其添加到两个策略中的 `skip-system-ns` 表达式，或者创建一个 `PolicyException` 资源。有关 `PolicyException` 模式，请参见下方 了解更多。

步骤 1.5 — 验证引擎已就绪

运行以下检查。所有资源都应该存在，并且两个策略都应该是 `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
```

如果 `scc-fill-defaults` 显示 `READY=false`，最常见的原因是缺少对 `pods/ephemeralcontainers` 的读取权限——请确保完整应用了步骤 1.3 中的 `kyverno-scc-reader` ClusterRole。

第 2 部分：授权工作负载使用 SCC

安装引擎后，默认情况下没有任何 Pod 被授予 SCC。除非管理员为某个 ServiceAccount（或 User，或 Group）创建了 RBAC 绑定，否则在非系统 namespace 中以该受试对象运行的 Pod 会被拒绝，并提示 `Pod violates all SCCs assigned to its ServiceAccount`。

请将每个 SCC 绑定视为一个安全授权决策。只有平台管理员或安全管理员才应拥有授予 SCC 绑定权限；普通应用用户和 namespace 所有者不应能够为自己提升 Pod 权限。

步骤 2.1 — 选择合适的 SCC 配置文件

将工作负载的安全需求与下表进行匹配。默认情况下，引擎会先按 `priority`、再按 `restrictiveScore` 对已授予的 SCC 进行排序。选择工作负载所需的最小权限配置文件集合，并在必须强制使用某个特定配置文件时使用 `alauda.io/required-scc`。

工作负载特征	推荐 **SCC**
无状态服务、非 root、丢弃所有 capabilities、无 host 访问	`restricted-v2`
与上相同，但需要绑定 1024 以下端口	`restricted-v2`（该配置文件已允许 `NET_BIND_SERVICE`）
与上相同，但使用固定 UID 范围，例如 1000–65534，并使用 user namespaces	`restricted-v3`
以非 root 用户运行，但不能丢弃所有 capabilities 的服务	`nonroot-v2`（丢弃 ALL）或 `nonroot`（较早的丢弃集合）
需要以 root 运行的镜像（`USER root`）	`anyuid`
需要 `hostNetwork` 和 host ports 的 Ingress controller 或其他 Pod	`hostnetwork-v2`（丢弃 ALL）或 `hostnetwork`（较早的丢弃集合）
挂载 `hostPath` 用于日志 / 指标收集、以非 root 运行的服务	`hostmount-anyuid`
与上相同，但不需要 SELinux 重新标记	`hostmount-anyuid-v2`
需要 `hostNetwork`、`hostPID`、`hostIPC` 和 host paths 的诊断 Pod	`hostaccess`
使用 user namespaces 的 container-in-container 构建沙箱	`nested-container`
完全特权工作负载（CNI、存储驱动、调试 Pod）	`privileged`

Warning

始终只授予所需的最小权限。绑定到 `privileged` 的 ServiceAccount 可以运行任何 Pod，包括那些能够突破容器边界的 Pod。`privileged` 应仅保留给基础设施 DaemonSets，不要授予用户工作负载。

当应用经理请求 SCC 访问时，请提供：

- Namespace 和 ServiceAccount，例如 `databases/postgres-sa`。
- 工作负载名称和控制器类型，例如 `StatefulSet/postgres`。
- 请求的 SCC 或所需能力，例如因为镜像以 UID 0 运行而需要 `anyuid`。
- 为什么更严格的 SCC（如 `restricted-v2`）不足以满足需求。
- 工作负载是否必须使用 `alauda.io/required-scc` 固定到某个特定 SCC。

步骤 2.2 — 将 SCC 绑定到 ServiceAccount

管理员最常见的操作是将某个 SCC 绑定到工作负载的 ServiceAccount。假设你有一个在 `databases/postgres-sa` 下运行的应用，并且镜像以 root（UID 0）运行。你希望允许该 ServiceAccount 使用 `anyuid`，同时仍保留 `restricted-v2` 供更严格的工作负载使用。在这个 root UID 示例中，`restricted-v2` 不匹配（`runAsUser.uidRangeMin: 1`），因此准入会选择 `anyuid`。更一般地说，当一个 Pod 同时满足两个配置文件时，本指南中的默认配置会优先选择 `anyuid`，因为除非你调整优先级或固定 `alauda.io/required-scc`，否则 `anyuid` 的 `priority` 高于 `restricted-v2`。

将以下内容保存为 `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
```

应用：

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

`rbac.alauda.io/scc-use=true` 标签是可选的。它不会影响 SCC 选择，但可让你通过 `kubectl get clusterrole,rolebinding -l rbac.alauda.io/scc-use=true -A` 列出所有与 SCC 相关的 RBAC 对象。

Note

你同样可以使用 `ClusterRoleBinding` 为这个有命名空间的 ServiceAccount 授予集群级 `use` 权限。通常当你希望授权只在某一个 namespace 内生效时，使用有命名空间的 `RoleBinding` 更清晰。

步骤 2.3 — 将 SCC 绑定到 User

当一名受信任的人类操作员（以 Kubernetes `User` 身份认证，例如通过 OIDC 或证书）需要直接启动 Pod 时——例如运行 `kubectl debug` 或 `kubectl run` 的 SRE——你可以将 SCC 授予该 User 主体。

将其保存为 `bind-user-sre.yaml`，并将 `[email protected]` 替换为你的 User 名称：

```yaml
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
```

应用：

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

当 `[email protected]` 直接运行 `kubectl run`（而不是通过某个控制器的 ServiceAccount）时，他们创建的 Pod 会以其 User 身份被准入，并获得 `hostaccess`。

步骤 2.4 — 将 SCC 绑定到 Group

Group 绑定适用于管理员管理的整体策略，例如“每个已认证用户都可以运行 `restricted-v2` Pod”。两个合成 Group 尤其相关：

- `system:authenticated` —— 每个已认证主体。
- `system:serviceaccounts:<namespace>` —— 某个特定 namespace 中的每个 ServiceAccount。

将其保存为 `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

`system:authenticated` Group 绑定是一个兜底机制，用于捕获那些 ServiceAccount 没有显式 SCC 绑定的工作负载。它可作为步骤 1.4 中 Warn 阶段上线时的迁移安全网。一旦每个工作负载都有了显式绑定，就应移除此兜底。如果永久保留它，未来一旦新增带有宽松默认值的 SCC 配置文件，你的影响范围会被扩大。

若要将绑定限制为单个 namespace 的 ServiceAccounts，请将 `subjects` 改为：

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

步骤 2.5 — 使用 `alauda.io/required-scc` 固定到特定 SCC

默认情况下，引擎会选择受试对象被允许使用的、且 Pod 实际满足条件的最严格 SCC。如果你有一个必须始终以某个特定配置文件准入的工作负载——例如，一个审计敏感的部署，必须使用 `restricted-v3`，即使其 ServiceAccount 也被允许 `anyuid`——就在 Pod 上设置 `alauda.io/required-scc` 注解：

```
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"]
```

`alauda.io/required-scc` 注解只会从该受试对象已被授权使用的 SCC 中进行选择，并不会授予 SCC 访问权限。该注解要生效，必须同时满足以下条件：

- 集群中存在名为 `restricted-v3` 的 SecurityContextConstraints。

- `payments/payments-sa` 已通过一个为该资源名授予 `use` 权限的 ClusterRole 或 Role 绑定到 `restricted-v3`。

如果任一条件不满足，Pod 将被拒绝。校验策略会针对每种情况输出具体消息（参见 故障排查）。

当使用 PodTemplate 风格控制器（Deployment、StatefulSet、Job）时，请将注解放在 Pod template 的 `metadata` 中，而不是放在控制器上：

```
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
      # ...
```

步骤 2.6 — 验证绑定是否生效

应用任意绑定后，请运行以下检查。

管理员验证 — 确认该主体可以 `use` 该 SCC：

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

期望输出为 `yes`。如果得到 `no`，请重新检查 ClusterRole 中的 `apiGroups`、`resources`、`resourceNames` 和 `verbs`。

应用所有者验证 — 在管理员确认绑定后，使用获批的 ServiceAccount 创建或重新部署工作负载，然后检查准入后的 Pod 注解。可快速探测如下：

```
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"}'
```

输出应为引擎所选择的 SCC 名称（在本例中为 `anyuid`）。如果应用所有者无法创建探测 Pod，管理员可以执行此检查，或直接检查真实工作负载中的某个 Pod。

> **Note**
>
> GlobalContextEntries 通过 list/watch 刷新，通常会在几秒内将新绑定传播到准入缓存，在高负载情况下可能最长需要一分钟。如果你在应用新绑定后立即遭到拒绝，请稍等片刻后重试，再判断是否为绑定错误。

结果

完成第 1 部分并至少完成一个第 2 部分绑定后，你应该能够验证以下所有内容：

- `kubectl get crd securitycontextconstraints.security.alauda.io` 显示 CRD 状态为 `Established=True`。
- `kubectl get scc` 列出你安装的每个 SCC 配置文件。
- `kubectl get globalcontextentry` 返回全部五个 `scc-*` 条目。
- `kubectl get validatingpolicy scc-auto-pick` 和 `kubectl get mutatingpolicy scc-fill-defaults` 都显示 `READY=true`。

- 在非系统 namespace 中，以已绑定 ServiceAccount 创建的 Pod 会收到 `alauda.io/scc=<name>` 注解，其中 `<name>` 是引擎选择的 SCC。
- 在非系统 namespace 中，以未绑定 ServiceAccount 创建的 Pod 会在准入时被拒绝，并提示 `Pod violates all SCCs assigned to its ServiceAccount`。

故障排查

使用下表将症状映射到原因和解决步骤。

对于由 Deployment、StatefulSet、Job 和 DaemonSet 等控制器创建的 Pod，实际生效的工作负载身份通常是 Pod 的 ServiceAccount。对于由受信任的人类操作员直接创建的 Pod，例如 `kubectl run` 或 `kubectl debug`，User 和 Group 的 SCC 绑定也可能匹配准入请求。

症状	可能原因	检查内容
`Pod violates all SCCs assigned to its ServiceAccount (candidates: ...)`	Pod 的 ServiceAccount 至少绑定了一个 SCC，但 Pod 的 spec 违反了这些 SCC 中的每一个。消息末尾的候选列表会列出被考虑的 SCC。	对每个候选项，将 Pod 配：容器 `runAsUser` `requiredDropCapabi` `[ALL]`；当 SCC 要求 `seccompProfile.typ`
`Pod violates all SCCs assigned to its ServiceAccount (candidates: )`（候选列表为空）	没有任何 SCC 绑定到该 Pod 的 ServiceAccount。	对 `kubectl get scc` `kubectl auth can-i` `securitycontextcons` `--as=system:service` 全部返回 `no`，请按照
`required SCC '<name>' not found in scc-profiles`	`alauda.io/required-scc` 注解引用了一个不存在的 SCC。	运行 `kubectl get sc` 配置文件。
`required SCC '<name>' is not bound to ServiceAccount '<sa>' in namespace '<ns>'`	注解引用了该 ServiceAccount 没有 `use` 权限的 SCC。	添加一个 `RoleBindin` `securitycontextcon` 予该 SA，然后重试。
刚添加绑定，但 Pod 仍然被拒绝	Kyverno GlobalContextEntry 缓存会异步刷新 RBAC 对象；新绑定需要几秒钟才能传播。	等待 10–30 秒后重试。`globalcontextentry` `jsonpath='{.status.` 最近刷新。

症状	可能原因	检查内容
Pod 已准入，但 `runAsUser` 意外地被设置 / 未设置	变更型策略从所选 SCC 中填充了默认值，或者因为 Pod 已经声明了值而没有填充。	检查 Pod 上的 `alauda` SCC，然后查看该 SC `runAsUser.uid`。Po 覆盖。
`scc-fill-defaults` 或 `scc-auto-pick` 上显示 `READY=false`	Kyverno 缺少策略所匹配某个资源的读取权限（最常见的是 `pods/ephemeralcontainers`）。	重新完整应用步骤 1.3 ClusterRole。
位于 `pod-security.kubernetes.io/enforce: restricted` namespace 中的 Pod 在 Kyverno 看到之前就被拒绝	Kubernetes Pod Security Admission 插件先于 Kyverno 运行，并独立强制执行 namespace 标签。	根据该 namespace 中的 为 `baseline` 或 `pri` SCC。

了解更多

使用 `PolicyException` 临时绕过策略

当你需要在短时间内允许某个 ServiceAccount 超出其当前 SCC（例如紧急调试会话），且修改 RBAC 绑定并不合适时，可以使用 `PolicyException` 资源。这要求 Kyverno 的 admission controller 启动时启用了 `--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'
```

最佳实践：将 `PolicyException` 资源放在专用 namespace 中（例如 `policy-exceptions`），并限制写入权限；为每个例外添加 `owner` 和 `expire-at` 标签，并按周期进行审计。

引擎如何选择 SCC

当多个 SCC 被授予给某个受试对象，且 Pod 同时满足其中多个时，校验策略按以下顺序对候选项排序：

1. 先看更高的 `priority`。
2. 再看更高的 `restrictiveScore`。

Pod 完整满足的第一个候选项就是最终选择的 SCC。变更型策略在填充默认值时使用相同的排序方式。这与 OpenShift 的意图一致，即“在可接受的 SCC 中，限制最严格者胜出”，同时允许运维人员通过每个配置文件的 `priority` 覆盖顺序。

OpenShift 命令映射

如果你来自 OpenShift，下面这些 `oc` 命令可以直接映射为对 SCC 引擎的 `kubectl apply`。这些操作会授予 SCC 的 `use` 权限，并且只应由有权更改集群 Pod 安全边界的管理

员执行。

OpenShift 命令	在本引擎中的等效操作
`oc adm policy add-scc-to-user <scc> <user>`	创建一个在 `securitycontextconstraints/<scc>` 上授予 `use` 的 ClusterRole，然后创建一个 `subjects: [{kind: User, name: <user>}]` 的 ClusterRoleBinding。
`oc adm policy add-scc-to-user <scc> -z <sa> -n <ns>`	与上面相同的 ClusterRole，再加上一个位于 namespace `<ns>` 的 `RoleBinding`，其中 `subjects: [{kind: ServiceAccount, name: <sa>, namespace: <ns>}]`。
`oc adm policy add-scc-to-group <scc> <group>`	相同的 ClusterRole，再加上一个 `subjects: [{kind: Group, name: <group>}]` 的 ClusterRoleBinding。
`oc get scc`	`kubectl get scc`（该 CRD 的 `shortNames: [scc]` 保持了命令一致性）。

最佳实践：在现有 **namespace** 中授予特权例外

当一个现有 namespace 通常遵循 PSA `restricted` 标准，但同一 namespace 中的某个工作负载必须运行 privileged 容器时，请使用此操作步骤。namespace 保持不变；平台管理员将移除该 namespace 上的 PSA restricted 强制执行，并让 Kyverno SCC 引擎成为强制执行的准入边界。

隔离模型如下：

- namespace 中的每个 ServiceAccount 都可以使用 `restricted-v2` 作为默认安全上限。
- 例外工作负载使用专用 ServiceAccount。
- 只有该 ServiceAccount 可以使用内置的 `privileged` SCC。
- 工作负载通过 `alauda.io/required-scc` 固定到 `privileged`。
- 其他 ServiceAccounts 不会被授予 `privileged`，如果请求特权设置，仍会被拒绝。

> **Warning**

PSA 和 Kyverno SCC 引擎是彼此独立的 admission controls。SCC 授权无法覆盖 `pod-security.kubernetes.io/enforce: restricted`。在移除 PSA 强制执行之前，请先完成 restricted SCC 基线和例外工作负载绑定，这样 namespace 就不会出现未受保护的过渡窗口。

第 1 步 - 审计现有 SCC 授权

在更改 PSA 之前，请检查普通 ServiceAccounts 是否已经通过 User、Group、RoleBinding 或 ClusterRoleBinding 获得了宽松 SCC。特别注意 `system:authenticated`、`system:serviceaccounts` 和 `system:serviceaccounts:payments` 的绑定。

例如，默认 ServiceAccount 不应能够使用内置的 `privileged` SCC：

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

期望输出为 `no`。在继续之前，先移除任何非预期的宽泛授权。前面提到的可选标签 `rbac.alauda.io/scc-use=true` 可以帮助定位与 SCC 相关的 RBAC 对象，但不要只依赖标签，因为未打标签的 Role 仍然可以授予 `use` 权限。

第 2 步 - 将 restricted 基线绑定到所有 namespace ServiceAccounts

创建一个从合成的 `system:serviceaccounts:payments` Group 到步骤 2.4 中 `scc-use-restricted-v2` ClusterRole 的 namespaced RoleBinding。这会覆盖该 namespace 中现有和未来的所有 ServiceAccounts。

保存为 `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
```

应用该绑定，并验证普通 ServiceAccount 只能使用 restricted 基线：

```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
```

期望输出为 `yes`。

第 3 步 - 创建专用 ServiceAccount

SCC 授权是基于主体的；它不会授予某个 Pod 名称权限。使用一个不与其他无关工作负载共享的专用 ServiceAccount，来建模单一工作负载例外。

保存为 `payment-agent-serviceaccount.yaml`：

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

应用该 ServiceAccount：

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

> **Warning**
>
> 内置的 `privileged` SCC 故意不加限制。除了特权容器之外，它还允许 host networking、host namespaces、host ports、host paths、所有 Linux capabilities、所有 volume 类型、所有 seccomp profiles 和 unsafe sysctls。只有当工作负载确实需要完全特权时才授予它。如果工作负载只需要更小的例外，例如仅 `anyuid`、`hostNetwork` 或 `hostPath`，请改为绑定更窄的内置配置文件。

除非工作负载必须调用 Kubernetes API，否则请保持 `automountServiceAccountToken: false`。启用 token 自动挂载会将 API 凭据添加到一个已经拥有较高运行时权限的容器中。

第 4 步 - 仅向专用 ServiceAccount 授予 `privileged`

创建一个 `resourceNames` 仅包含内置 `privileged` SCC 的 ClusterRole，然后通过位于工作负载 namespace 中的 RoleBinding 将其绑定到专用 ServiceAccount。

保存为 `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
```

应用并验证该绑定：

```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
```

期望输出为：`payment-agent-privileged` 为 `yes`，`default` 为 `no`。该专用 ServiceAccount 还会从 namespace Group 绑定继承 `restricted-v2`。

第 5 步 - 防止其他工作负载复用该 ServiceAccount

此操作会将 `privileged` 授予某个 ServiceAccount，而不是某个 Pod 或控制器名称。Kubernetes RBAC 不提供针对 `spec.serviceAccountName` 的字段级授权。如果某个用户可以在 `payments` 中创建任意 Pod、Deployment、StatefulSet、DaemonSet、Job 或 CronJob，那么他就可能从其他工作负载引用 `payment-agent-privileged`，并获得相同的 SCC 访问权限。

在继续之前，请至少实施以下一种控制：

- 仅允许专用的 GitOps 或部署身份创建或更新引用 `payment-agent-privileged` 的工作负载，并阻止普通 namespace 用户修改这些工作负载。
- 添加一个 admission guard，拒绝对 `payment-agent-privileged` 的引用，除非请求已获授权。将该 guard 同时应用于直接 Pod 和该 namespace 中启用的每一种 Pod-template 控制器。

不要仅依赖工作负载标签或注解来授权使用专用 ServiceAccount；能够创建工作负载的用户通常也可以复制这些值。如果既无法限制部署所有权，也无法限制准入时的 ServiceAccount 使用，则此模式只能提供 ServiceAccount 级别的例外，不得将其表示为单 Pod 例外。

在强制实施其中一种控制之前，不要继续执行 PSA 迁移。

第 6 步 - 将强制执行从 PSA 转移到 Kyverno SCC

只移除 PSA restricted 强制执行标签。保留 restricted 的 audit 和 warning 标签，以便 API server 继续报告那些会违反 restricted 标准的工作负载。

```
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
```

移除 namespace 标签不会覆盖通过 API server admission configuration 配置的集群级 PSA 默认值。如果集群默认值仍然强制执行 `restricted`，请显式将该 namespace 从强制执行中排除，同时保留 audit 和 warning 可见性：

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

这里 `enforce=privileged` 的意思是 PSA 不再施加额外限制。它并不会向 Pod 授予 privileged 访问；Kyverno SCC 绑定仍然是强制执行的授权边界。

第 7 步 - 在工作负载中固定获批的 SCC

将 `alauda.io/required-scc` 放入 Pod template 的 metadata 中，并使用专用 ServiceAccount。不要只把注解放在 Deployment metadata 上，因为控制器的 metadata 不会自动复制到其 Pod。

```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
```

该注解不会授予 SCC 权限。如果 SCC 不存在，或者 ServiceAccount 没有 `use` 权限，校验策略会拒绝该 Pod。

第 8 步 - 验证例外是否隔离

请使用真实的应用部署身份运行 admission 探测，而不是使用 `cluster-admin`。对于直接创建的 Pod，SCC 选择会考虑请求中的 User 和 Groups，因此拥有额外宽泛 SCC 授权的管理员，可能会得到与生产中工作负载身份不同的结果。

验证该例外工作负载是否以预期 SCC 被准入：

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

SCC 列应为 `privileged`。

接下来，确认普通 restricted Pod 仍然可以以 `restricted-v2` 被准入：

```
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
```

期望输出为 `restricted-v2`。

最后，确认普通 ServiceAccount 不能请求 `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
```

当由正常的应用部署身份提交时，请求必须被拒绝，并提示所需 SCC 未绑定到 ServiceAccount `default`。如果该 Pod 被准入，请停止并检查宽泛的 SCC RoleBindings、ClusterRoleBindings，以及 User 或 Group 授权，然后再允许生产工作负载继续运行。

回滚该例外

要安全恢复 PSA restricted 强制执行，请按以下步骤操作：

1. 停止或替换特权工作负载，使其变为符合 `restricted` 的规格。
2. 移除授予专用 ServiceAccount `privileged` 的 RoleBinding：

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

3. 确认该 ServiceAccount 不再能够使用 `privileged`：

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

授权检查必须返回 `no`。不要删除内置的 `privileged` SCC。只有在没有其他 RoleBinding 或 ClusterRoleBinding 引用它时，才删除可复用的 `scc-use-privileged` ClusterRole。

4. 对剩余工作负载执行 server-side dry-run 检查。
5. 最后恢复 PSA restricted 强制执行：

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

后续步骤

- 在审查工作负载需求后，确定每个现有 namespace 和 ServiceAccount 应绑定到哪个 SCC，记录该映射，并通过 GitOps 工作流应用这些绑定，以便实现可审计、可复现。
- 计划对 `PolicyException` 资源进行定期审查——它们的用途是短期窗口，而不是永久例外。
- 如果你的规模较大，请监控 Kyverno admission controller 的 `kyverno_admission_review_duration_seconds` 指标，以便在 SCC 配置文件或 RBAC 绑定数量增长时检测准入延迟变化。

镜像签名验证策略

本指南演示如何配置 Kyverno，以确保容器镜像在 Kubernetes 集群中运行前已正确签名。可以将其类比为检查身份证——只有带有有效“签名”的镜像才被允许使用。

目录

什么是镜像签名验证？

快速开始

1. 生成密钥
2. 签署镜像
3. 创建基础验证策略
4. 测试

常见用例

场景 1：多个团队需签署关键镜像

场景 2：不同环境使用不同规则

场景 3：使用证书替代密钥

什么是镜像签名验证？

镜像签名验证就像门口的安保人员检查身份证。它确保：

- 镜像真实可信：来自其声称的来源

- 镜像未被篡改：签名后无人修改
- 仅允许可信镜像运行：阻止未签名或签名不当的镜像
- 审计追踪：记录哪些镜像何时被验证

快速开始

1. 生成密钥

```
# 创建签名密钥对（类似创建身份证系统）
cosign generate-key-pair
# 这将生成：cosign.key（私钥，保密）和 cosign.pub（公钥，可自由分享）
```

2. 签署镜像

```
# 签署镜像（类似盖上官方印章）
cosign sign --key cosign.key registry.company.com/app:v1.0.0
```

3. 创建基础验证策略

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-signed-images
spec:
  validationFailureAction: Enforce  # 阻止未签名镜像
  background: false
  rules:
    - name: check-signatures
      match:
        any:
        - resources:
            kinds:
            - Pod
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"  # 检查公司镜像仓库的镜像
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 在此粘贴 cosign.pub 内容
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        mutateDigest: true  # 将标签转换为安全的摘要格式
```

4. 测试

```
# 应用策略
kubectl apply -f signature-policy.yaml

# 尝试运行未签名镜像（应失败）
kubectl run test --image=nginx:latest

# 尝试运行已签名镜像（应成功）
kubectl run test --image=registry.company.com/app:v1.0.0
```

常见用例

场景 1：多个团队需签署关键镜像

对于关键应用，开发团队和安全团队都可能需要签署镜像：

```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:
        # 两个团队都必须签署
        - count: 1  # 安全团队签名
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 安全团队公钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        - count: 1  # 开发团队签名
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 开发团队公钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917h
mKO6JWV
                GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J
5==
                -----END PUBLIC KEY-----
        mutateDigest: true
```

场景 2：不同环境使用不同规则

生产环境需要严格验证，开发环境可以更宽松：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-specific-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # 生产环境严格规则
    - name: production-must-be-signed
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
      verifyImages:
      - imageReferences:
        - "*"  # 所有镜像必须签名
        failureAction: Enforce  # 未签名则阻止
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 生产签名密钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sb
qOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZ
A==
                -----END PUBLIC KEY-----
        mutateDigest: true

    # 开发环境宽松规则
    - name: development-warn-unsigned
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
```

```
            - development
            - staging
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"  # 仅检查公司镜像
        failureAction: Audit  # 审计但允许未签名镜像
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 开发签名密钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917h
mKO6JWV
                GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J
5==
                -----END PUBLIC KEY-----
        mutateDigest: true
```

场景 3：使用证书替代密钥

在企业环境中，可能使用 X.509 证书：

```
# 使用证书签名
cosign sign --cert company-cert.pem --cert-chain ca-chain.pem \
  registry.company.com/myapp:v1.0.0
```

使用 Secrets 的镜像签名验证策略

本指南演示如何使用 Kubernetes Secrets 存储 Kyverno 镜像签名验证的公钥，相较于将密钥直接嵌入策略中，这种方式提供了更好的安全性和密钥管理。

目录

为什么使用 Secrets 存储公钥？

快速开始

1. 生成并存储密钥到 Secret
2. 为 Kyverno 配置 RBAC
3. 使用 Secret 引用创建策略
4. 测试配置

Secret 创建方式

方式一：从文件创建

方式二：从字符串字面量创建

方式三：从 YAML 清单创建

常见使用场景

场景 1：单团队使用单个 Secret

场景 2：多团队使用不同 Secret

场景 3：关键镜像需要多重签名

场景 4：离线环境使用 Secrets

为什么使用 **Secrets** 存储公钥？

使用 Kubernetes Secrets 存储公钥具有以下优势：

- 增强安全性：密钥安全地存储在 Kubernetes Secret 存储中
- 便捷的密钥轮换：无需修改策略即可更新密钥
- 访问控制：使用 RBAC 控制谁可以访问 Secrets

快速开始

1. 生成并存储密钥到 **Secret**

```
# 生成 cosign 密钥对
cosign generate-key-pair

# 从公钥文件创建 Secret
kubectl create secret generic cosign-public-key \
  --from-file=cosign.pub=./cosign.pub \
  --namespace=kyverno

# 验证 Secret 是否创建成功
kubectl get secret cosign-public-key -n kyverno
```

2. 为 **Kyverno** 配置 **RBAC**

创建 Kyverno 的 Service Account

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

创建访问 Secret 的 Role

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

将 Role 绑定到 Service Account

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

3. 使用 **Secret** 引用创建策略

```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. 测试配置

```
# 签署镜像
cosign sign --key cosign.key registry.company.com/app:v1.0.0

# 应用策略
kubectl apply -f verify-with-secret.yaml

# 使用已签名镜像测试（应成功）
kubectl run test --image=registry.company.com/app:v1.0.0

# 使用未签名镜像测试（应失败）
kubectl run test-fail --image=nginx:latest
```

Secret 创建方式

方式一：从文件创建

```
# 从已有 cosign 公钥文件创建 Secret
kubectl create secret generic cosign-public-key \
  --from-file=cosign.pub=./cosign.pub \
  --namespace=kyverno
```

方式二：从字符串字面量创建

```
# 使用内联公钥内容创建 Secret
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
```

方式三：从 YAML 清单创建

```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
```

常见使用场景

场景 1：单团队使用单个 Secret

简单场景，一个团队管理所有镜像签名：

```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
```

场景 **2**：多团队使用不同 **Secret**

不同团队拥有各自的签名密钥和 Secret：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: multi-team-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # 前端团队镜像
    - 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

    # 后端团队镜像
    - 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
```

场景 3：关键镜像需要多重签名

高安全环境，多个团队必须对关键镜像签名：

```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:
        # 安全部门签名（必需）
        - count: 1
          entries:
          - keys:
              secret:
                name: security-team-key
                namespace: kyverno
                key: security.pub
              rekor:
                url: https://rekor.sigstore.dev

        # 开发团队签名（必需）
        - count: 1
          entries:
          - keys:
              secret:
                name: dev-team-key
                namespace: kyverno
                key: development.pub
              rekor:
                url: https://rekor.sigstore.dev
```

```
    # 发布团队签名（必需）
    - count: 1
      entries:
      - keys:
          secret:
            name: release-team-key
            namespace: kyverno
            key: release.pub
          rekor:
            url: https://rekor.sigstore.dev

    mutateDigest: true
    required: true
```

场景 4：离线环境使用 Secrets

在隔离环境中使用 Secrets：

镜像仓库验证策略

本指南演示如何配置 Kyverno 来控制 Kubernetes 集群中可使用的容器镜像仓库。它实现了仓库访问控制策略，确保只部署来自批准和可信仓库的镜像。

目录

什么是镜像仓库验证？

快速开始

1. 阻止除公司仓库外的所有仓库
2. 测试策略

常见场景

场景 1：允许多个可信仓库

场景 2：不同环境不同规则

场景 3：阻止特定风险仓库

场景 4：团队专属仓库访问

高级模式

有效使用通配符

最佳实践

从警告模式开始

排除系统命名空间

常见问题

什么是镜像仓库验证？

仓库验证提供了对镜像来源的集中控制。它能够：

- 控制镜像来源：仅允许来自可信仓库的镜像
- 阻止风险仓库：防止使用未知或被攻破的仓库
- 强制合规：满足关于镜像来源的安全要求
- 针对不同环境制定不同规则：生产环境严格，开发环境宽松
- 跟踪使用情况：监控使用了哪些仓库

快速开始

1. 阻止除公司仓库外的所有仓库

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: company-registry-only
spec:
  validationFailureAction: Enforce  # 阻止非批准镜像
  background: false
  rules:
    - name: check-registry
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "仅允许公司仓库：registry.company.com"
        pattern:
          spec:
            containers:
            - image: "registry.company.com/*"
```

2. 测试策略

```
# 应用策略
kubectl apply -f registry-policy.yaml

# 这条命令应失败（nginx 来自 Docker Hub）
kubectl run test --image=nginx:latest

# 这条命令应成功（如果镜像存在于仓库中）
kubectl run test --image=registry.company.com/nginx:latest
```

常见场景

场景 1：允许多个可信仓库

组织通常使用多个仓库：

```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: "镜像必须来自批准的仓库：公司仓库、GCR 或官方 Docker 镜像"
        anyPattern:
        - spec:
            containers:
            - image: "registry.company.com/*"      # 公司仓库
        - spec:
            containers:
            - image: "gcr.io/project-name/*"       # Google Container Registry
        - spec:
            containers:
            - image: "docker.io/library/*"         # 仅官方 Docker 镜像
        - spec:
            containers:
            - image: "quay.io/organization/*"      # Red Hat Quay
```

场景 2：不同环境不同规则

生产环境应严格，开发环境可更灵活：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-based-registry-rules
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # 生产环境：仅允许认证镜像
    - name: production-strict-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "生产环境仅允许认证的公司镜像"
        pattern:
          spec:
            containers:
            - image: "registry.company.com/certified/*"

    # 开发环境：允许更多仓库
    - name: development-flexible-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - development
            - dev-*
            - staging
            - test-*
      validate:
        message: "开发环境允许使用公司仓库、GCR 或官方 Docker 镜像"
        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/*"
```

场景 3：阻止特定风险仓库

阻止特定仓库，同时允许其他仓库：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: block-risky-registries
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # 方法一：使用拒绝列表
    - name: block-untrusted-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "不允许使用 untrusted-registry.com 的镜像"
        deny:
          conditions:
          - key: "{{ request.object.spec.containers[?contains(image, 'untrusted-registry.com')] | length(@) }}"
            operator: GreaterThan
            value: 0

    # 方法二：Docker Hub 仅允许官方镜像
    - name: allow-only-official-dockerhub
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "仅允许官方 Docker Hub 镜像（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
```

场景 4：团队专属仓库访问

不同团队可访问不同仓库：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: team-specific-registries
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    # 前端团队可使用 Node.js 镜像
    - name: frontend-team-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - frontend-*
      validate:
        message: "前端团队可使用公司仓库和官方 Node.js 镜像"
        anyPattern:
        - spec:
            containers:
            - image: "registry.company.com/*"
        - spec:
            containers:
            - image: "docker.io/library/node:*"
        - spec:
            containers:
            - image: "docker.io/library/nginx:*"

    # 数据团队可使用 ML/AI 镜像仓库
    - name: data-team-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - data-*
            - ml-*
      validate:
        message: "数据团队可使用公司仓库和 ML/AI 镜像"
        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/*"
```

高级模式

有效使用通配符

```
# 匹配模式：
- image: "registry.company.com/*"            # 来自该仓库的任意镜像
- image: "registry.company.com/team-a/*"     # 仅 team-a 的镜像
- image: "*/database:*"                      # 任意仓库的数据库镜像
- image: "gcr.io/project-*/app:*"           # GCR 中 project-* 下的任意 app
镜像
```

最佳实践

从警告模式开始

```
spec:
  validationFailureAction: Audit  # 先使用审计模式，不阻止
```

排除系统命名空间

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

常见问题

1. 镜像格式错误：

- ❌ `registry.company.com:5000/app`（缺少协议）
- ✅ `registry.company.com/app:latest`

2. 通配符误用：

- ❌ `registry.company.com*`（缺少斜杠）
- ✅ `registry.company.com/*`

3. **Docker Hub** 格式：

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

容器逃逸防护策略

本指南演示如何配置 Kyverno，通过阻止可能导致容器突破隔离边界的高风险容器配置，来防止容器逃逸攻击。

目录

什么是容器逃逸防护？

快速开始

1. 阻止特权容器

2. 测试策略

核心容器逃逸防护策略

策略 1：禁止访问宿主命名空间

策略 2：禁止宿主路径挂载

策略 3：禁止宿主端口

策略 4：禁止危险能力

策略 5：要求非 root 用户运行容器

高级场景

场景 1：环境特定策略

场景 2：工作负载特定例外

测试与验证

测试特权容器

测试宿主命名空间访问

测试宿主路径挂载

测试有效安全容器

最佳实践

1. 从审计模式开始
2. 排除系统命名空间

什么是容器逃逸防护？

容器逃逸防护涉及检测并阻止危险的容器配置，这些配置可能允许攻击者逃离容器隔离并访问宿主系统。包括：

- 特权容器：以提升权限运行的容器
- 宿主命名空间访问：容器共享宿主的 PID、网络或 IPC 命名空间
- 宿主路径挂载：容器挂载宿主文件系统路径
- 危险能力：容器拥有过多的 Linux 能力
- 宿主端口访问：容器绑定宿主网络端口

快速开始

1. 阻止特权容器

```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. 测试策略

```
# 应用策略
kubectl apply -f disallow-privileged-containers.yaml

# 尝试创建特权容器（应失败）
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-privileged
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      privileged: true
EOF

# 尝试创建普通容器（应成功）
kubectl run test-normal --image=nginx

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

核心容器逃逸防护策略

策略 1：禁止访问宿主命名空间

防止容器访问宿主命名空间：

```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"
```

策略 2：禁止宿主路径挂载

阻止容器挂载宿主文件系统路径：

```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"
```

策略 3：禁止宿主端口

防止容器绑定宿主网络端口：

```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
```

策略 4：禁止危险能力

阻止容器添加危险的 Linux 能力：

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

```
        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
```

策略 **5**：要求非 **root** 用户运行容器

确保容器以非 root 用户身份运行：

```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"
```

高级场景

场景 1：环境特定策略

针对不同环境设置不同安全级别：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-container-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 生产环境：严格安全
    - 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

    # 开发环境：更宽松但仍安全
    - 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"
```

场景 2：工作负载特定例外

允许特定工作负载的受控例外：

```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"
```

测试与验证

测试特权容器

```
# 应该被阻止
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-privileged
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      privileged: true
EOF
```

测试宿主命名空间访问

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

测试宿主路径挂载

```
# 应该被阻止
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
```

测试有效安全容器

```
# 应该被允许
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
```

最佳实践

1. 从审计模式开始

```
spec:
  validationFailureAction: Audit  # 先以警告模式启动，不阻止
```

2. 排除系统命名空间

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

Security Context Enforcement Policy

本指南演示如何配置 Kyverno 以强制执行容器的正确安全上下文，确保容器以适当的安全设置和限制运行。

目录

什么是安全上下文强制？

快速开始

1. 要求非 Root 容器策略

2. 测试策略

核心安全上下文策略

策略 1：禁止权限提升

策略 2：要求特定用户 ID 范围

策略 3：要求非 Root 组

策略 4：限制 Seccomp 配置文件

策略 5：要求丢弃所有能力

策略 6：限制 AppArmor 配置文件

高级场景

场景 1：环境特定的安全上下文

场景 2：应用特定的安全上下文

场景 3：分阶段安全上下文强制

场景 4：指定命名空间安全上下文强制

测试与验证

测试 Root 容器（应失败）

测试权限提升（应失败）

测试缺少能力丢弃（应失败）

测试有效安全容器（应通过）

什么是安全上下文强制？

安全上下文强制涉及通过设置与安全相关的参数来控制容器的运行方式。正确的安全上下文配置可以防止：

- **Root** 权限提升：容器以 root 用户运行
- 权限提升攻击：容器获得提升的权限
- 不安全的进程执行：容器以危险的能力运行
- 文件系统篡改：容器具有可写的根文件系统
- 安全绕过：容器绕过安全机制

快速开始

1. 要求非 **Root** 容器策略

```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. 测试策略

```
# 应用策略
kubectl apply -f require-run-as-nonroot.yaml

# 尝试创建一个明确以 root 运行的容器（应失败）
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

# 尝试创建一个以非 root 用户运行的容器（应成功）
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

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

核心安全上下文策略

策略 1：禁止权限提升

防止容器提升权限：

```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"
```

策略 **2**：要求特定用户 **ID** 范围

确保容器以特定用户 ID 范围运行：

```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:
            # 检查 Pod 级别的安全上下文
            - key: "{{ request.object.spec.securityContext.runAsUser || 0
}}"
              operator: LessThan
              value: 1000
            - key: "{{ request.object.spec.securityContext.runAsUser || 0
}}"
              operator: GreaterThan
              value: 65535
            # 检查容器级别的安全上下文
            - key: "{{ request.object.spec.containers[?securityContext.ru
nAsUser && (securityContext.runAsUser < `1000` || securityContext.runAsUs
er > `65535`)] | length(@) }}"
              operator: GreaterThan
              value: 0
```

策略 **3**：要求非 **Root** 组

确保容器以非 root 组 ID 运行：

```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:
            # 检查 Pod 级别 runAsGroup 是否为 0
            - key: "{{ request.object.spec.securityContext.runAsGroup ||
0 }}"
              operator: Equals
              value: 0
            # 检查是否有容器的 runAsGroup 设置为 0
            - key: "{{ request.object.spec.containers[?securityContext.ru
nAsGroup == `0`] | length(@) }}"
              operator: GreaterThan
              value: 0
```

策略 **4**：限制 **Seccomp** 配置文件

强制使用安全的 seccomp 配置文件：

```
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
```

策略 5：要求丢弃所有能力

确保容器丢弃所有能力：

```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 || `
[]` }}"
```

策略 6：限制 AppArmor 配置文件

控制 AppArmor 配置文件的使用：

```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"
```

高级场景

场景 1：环境特定的安全上下文

针对不同环境的不同安全需求：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 生产环境：严格安全上下文
    - 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

    # 开发环境：基础安全要求
    - 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"
```

场景 2：应用特定的安全上下文

针对不同应用类型的不同安全上下文：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 数据库应用：特定用户/组 ID
    - 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 应用：标准安全上下文
    - 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
```

场景 3：分阶段安全上下文强制

实现渐进式安全上下文要求：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: graduated-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 级别 1：基础安全（所有命名空间）
    - 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"

    # 级别 2：增强安全（敏感命名空间）
    - 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

  # 级别 3：最高安全（关键命名空间）
  - 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
```

场景 4：指定命名空间安全上下文强制

基于标签的命名空间安全策略注入实现：

```
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
```

如果您不希望某些命名空间被注入安全上下文检测，请为该命名空间添加标签 `pod-security.kubernetes.io/enforce: privileged`

测试与验证

测试 **Root** 容器（应失败）

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

测试权限提升（应失败）

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

测试缺少能力丢弃（应失败）

```
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
```

测试有效安全容器（应通过）

```
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
```

网络安全策略

本指南演示如何配置 Kyverno 以强制执行网络安全策略，控制容器的网络访问并防止基于网络的攻击。

目录

什么是网络安全？

快速开始

1. 禁止主机网络访问

2. 测试策略

核心网络安全策略

策略 1：禁止主机端口

策略 2：限制主机端口范围

策略 3：要求网络策略

策略 4：限制 Service 类型

策略 5：控制 Ingress 配置

策略 6：限制 DNS 配置

高级场景

场景 1：环境特定的网络策略

场景 2：应用特定的网络策略

场景 3：网络分段强制执行

测试与验证

测试主机网络访问（应失败）

测试主机端口绑定（应失败）

测试 NodePort 服务（应失败）

测试有效的网络配置（应通过）

什么是网络安全？

网络安全涉及控制容器如何访问和交互网络资源。适当的网络安全可以防止：

- 主机网络访问：容器访问主机网络接口
- 通过网络权限提升：利用网络访问获得提升权限
- 端口扫描和侦察：未经授权的网络发现活动
- 横向移动：容器访问非预期的网络资源
- 数据外泄：未经授权的网络通信

快速开始

1. 禁止主机网络访问

```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. 测试策略

```
# 应用策略
kubectl apply -f disallow-host-network.yaml

# 尝试创建使用主机网络的 Pod（应失败）
kubectl run test-hostnet --image=nginx --overrides='{"spec":{"hostNetwor
k":true}}'

# 尝试创建普通 Pod（应成功）
kubectl run test-normal --image=nginx
```

核心网络安全策略

策略 1：禁止主机端口

防止容器绑定主机网络端口：

```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
```

策略 2：限制主机端口范围

允许特定主机端口范围以实现受控访问：

```
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
```

策略 3：要求网络策略

确保 Pod 关联有 NetworkPolicies 以控制流量：

```
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
```

策略 **4**：限制 **Service** 类型

控制允许创建的 Service 类型：

```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 LoadBalancer 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 environments.
```

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

策略 **5**：控制 **Ingress** 配置

强制安全的 Ingress 配置：

```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"
```

策略 6：限制 DNS 配置

控制 DNS 设置以防止基于 DNS 的攻击：

```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"
```

高级场景

场景 1：环境特定的网络策略

针对不同环境的不同网络限制：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-network-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 生产环境：严格的网络控制
    - 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

    # 开发环境：基础网络安全
    - 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"
```

场景 2：应用特定的网络策略

针对不同应用类型的不同网络策略：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-network-policies
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 数据库应用：禁止外部网络访问
    - 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 应用：受控端口访问
    - 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
```

场景 3：网络分段强制执行

强制不同层级之间的网络分段：

```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"
```

测试与验证

测试主机网络访问（应失败）

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

测试主机端口绑定（应失败）

```
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
```

测试 **NodePort** 服务（应失败）

```
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
```

测试有效的网络配置（应通过）

```
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

本指南演示如何配置 Kyverno 以强制执行卷安全策略，限制可能危及容器安全的危险卷类型和配置。

目录

什么是卷安全？

快速开始

1. 限制卷类型
2. 测试策略

核心卷安全策略

策略 1：禁止 HostPath 卷

策略 2：限制 HostPath 卷（只读访问）

策略 3：禁止特权卷类型

策略 4：要求只读根文件系统

策略 5：控制卷挂载权限

高级场景

场景 1：环境特定的卷策略

场景 2：应用特定的卷策略

场景 3：卷大小和资源限制

测试与验证

测试 HostPath 卷（应失败）

什么是卷安全？

卷安全涉及控制容器可以挂载的卷类型及其访问方式。适当的卷安全可防止：

- 主机文件系统访问：未经授权访问主机目录
- 权限提升：通过卷获取提升权限
- 数据外泄：通过卷挂载访问敏感主机数据
- 容器逃逸：通过卷访问突破容器隔离
- 不安全的卷类型：使用绕过安全控制的卷类型

快速开始

1. 限制卷类型

```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. 测试策略

```
# 应用策略
kubectl apply -f restrict-volume-types.yaml

# 尝试创建带 hostPath 卷的 Pod（应失败）
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

# 先创建测试 ConfigMap
kubectl create configmap test-config --from-literal=key=value

# 尝试创建允许的卷类型 Pod（应成功）
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
```

```
# 清理资源
kubectl delete pod test-hostpath test-configmap --ignore-not-found
kubectl delete configmap test-config --ignore-not-found
```

核心卷安全策略

策略 1：禁止 HostPath 卷

防止容器挂载主机文件系统路径：

```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"
```

策略 2：限制 **HostPath** 卷（只读访问）

允许特定的只读 HostPath 卷：

```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:
              # 如果路径不在允许列表中则拒绝
              - 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:
          # 如果卷挂载不是只读则拒绝
          - key: "{{ element.readOnly || false }}"
            operator: Equals
            value: false
```

策略 3：禁止特权卷类型

阻止可绕过安全控制的卷类型：

```
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
```

策略 4：要求只读根文件系统

确保容器使用只读根文件系统：

```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
```

策略 **5**：控制卷挂载权限

限制卷挂载的权限和路径：

```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:
              # 阻止挂载到敏感系统路径
              - 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
```

高级场景

场景 1：环境特定的卷策略

针对不同环境设置不同的卷限制：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-volume-security
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 生产环境：严格的卷控制
    - 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

    # 开发环境：更宽松但仍安全
    - 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  # 开发环境仍禁止 hostPath
              - nfs       # 禁止网络文件系统
```

场景 2：应用特定的卷策略

针对不同应用类型设置不同的卷策略：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: application-volume-policies
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 数据库应用：允许持久存储
    - 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 应用：限制为安全卷类型
    - 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
```

场景 3：卷大小和资源限制

控制卷大小和资源使用：

```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"
```

测试与验证

测试 **HostPath** 卷（应失败）

```
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 配置使用场景

本文档基于 Kyverno 提供核心的 policy 配置使用场景。它帮助你在 Kubernetes 集群中基于 Namespace 或 Project 实现自动化资源 mutation、统一配置，以及安全模板和基础环境的自动注入。

目录

1. 资源 Mutation 与统一配置（Mutate）
   1.1 向 Pod 注入统一标签
   1.2 强制 Pod 使用统一的 RestartPolicy
2. 基于 Namespace/Project 的自动化模板配置（Generate）
   2.1 自动注入默认隔离 NetworkPolicy
   2.2 基于模板自动初始化 Project 配置（DBS/安全配额）
总结

---

1. 资源 Mutation 与统一配置（Mutate）

Kyverno 的 Mutate 规则可以在 admission control 阶段自动修改已提交的资源。以下案例演示了如何为某个 namespace 下的所有 Pod 注入统一的标签，并强制统一的 `restartPolicy`。

1.1 向 Pod 注入统一标签

此 policy 会自动为集群中新创建的所有 Pod（或特定 namespace 内的 Pod）追加预设标签。这通常用于统一的项目管理和计费调度。

```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 强制 Pod 使用统一的 RestartPolicy

此 policy 强制所有新创建的 Pod 的默认 `restartPolicy` 为 `Always`。这对于确保业务容器在意外退出时能够自动重启至关重要。

```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. 基于 Namespace/Project 的自动化模板配置（Generate）

当创建新的 namespace 或 project 时，Kyverno 的 Generate 规则可以检测到该事件，并自动生成相关的 Kubernetes 资源（例如 NetworkPolicy、ConfigMap、Secret、RoleBinding 等）。这相当于一个开箱即用的安全与统一配置模板。

2.1 自动注入默认隔离 NetworkPolicy

当创建新的 namespace 时，此 policy 会自动生成一个默认的 `NetworkPolicy`。该 policy 默认拒绝所有入站（Ingress）请求，从而覆盖默认的网络连通性，实现 namespace 之间的网络隔离。

```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 基于模板自动初始化 Project 配置（DBS/安全配额）

此示例演示了如何根据 project 属性（例如创建 Namespace 时附加的标签）自动准备一系列底层环境。例如，发放 DBS 连接模板（供 CLI 或应用读取）以及默认的安全配额（LimitRange）。

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: setup-namespace-dbs-template
spec:
  rules:
    # Rule 1: Automatically generate DBS default configuration for projects with specific labels
    - name: generate-dbs-configmap
      match:
        any:
        - resources:
            kinds:
              - Namespace
            selector:
              matchLabels:
                project-type: database # Triggers when the NS has this Label
      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 newly 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
```

总结

1. **Mutate** 能力：以非侵入方式修正和补充开发者提交的 YAML，轻松实现资源级别的标签和状态控制（例如 `RestartPolicy`）。
2. **Generate** 能力：作为声明式的 project 生成器。一旦发生 Namespace 创建事件，Kyverno 会在后台自动填充安全策略（NetworkPolicy）和依赖模板（ConfigMap、Secret、LimitRange），为 CLI 工具和上层应用提供高度标准化、统一的隔离环境。

使用 Kyverno Generate 规则

使用 Kyverno `generate` 规则，在创建或更新匹配的资源时自动创建 Kubernetes 资源。Generate 规则定义以下内容：

- 要监视的源资源，使用 `match`。
- 应触发生成的变更，使用可选的 `preconditions`。
- Kyverno 应创建的目标资源，使用 `generate`。
- Kyverno 是否应使生成的资源与源资源保持同步。

本指南以更新 ConfigMap 作为示例流程。当特定命名空间中的特定 ConfigMap 发生变更时，Kyverno 会创建平台 `NotificationMessage` 资源。随后，通知服务会向指定的收件人发送电子邮件。

有关上游 Kyverno 参考信息，请参阅官方的 Kyverno generate 规则文档↗。

目录

Generate 的工作原理
示例场景
前提条件
准备目标资源依赖项
准备源资源
授予 Kyverno 权限
创建生成策略
验证生成规则

故障排查

Generate 的工作原理

Kyverno generate 规则是 `ClusterPolicy` 或 `Policy` 的一部分。

```
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>
```

当每次触发都应创建独立的目标资源时，使用 `synchronize: false`。仅当生成的资源应随源资源一同更新或删除时，才使用 `synchronize: true`。

示例场景

在此示例中：

- 源命名空间为 `app-team-a`。
- 源 ConfigMap 为 `app-notification-source`。
- 仅匹配 `UPDATE` 操作。
- 仅对 `ConfigMap.data` 的变更会触发生成。
- Kyverno 会在 `cpaas-system` 中创建一个短生命周期的 `NotificationMessage`。
- 生成的消息会直接向 `platform-operator@example.com` 发送电子邮件。

此示例使用的通知消息 API 是 `notificationmessages.aitextensions.alauda.io/v1beta1`。该 API 由平台通知 API 提供，并会在创建后很快被消费。

前提条件

- 已安装 Alauda Container Platform Policy Management。
- 同一集群中已安装平台通知功能。
- 已配置电子邮件通知服务器。平台默认电子邮件服务器以 `platform-email-server` Secret 的形式存储在 `cpaas-system` 命名空间中。
- `cpaas-system` 命名空间中存在用于电子邮件消息的通知模板。
- Kyverno 具有在目标命名空间中创建 `NotificationMessage` 资源的权限。

此示例不需要通知规则、通知策略、通知接收器或通知发送器资源。收件人的电子邮件地址直接在生成的 `NotificationMessage` 中指定。

准备目标资源依赖项

生成的 `NotificationMessage` 引用了一个电子邮件模板。在应用 Kyverno 策略之前创建该模板。

```
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 }}
```

准备源资源

创建 Kyverno 将监视的命名空间和 ConfigMap。

```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
```

授予 **Kyverno** 权限

Kyverno 通过其后台控制器创建生成的资源。授予该控制器在目标命名空间中创建 `NotificationMessage` 资源的权限。

```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
```

如果 Kyverno 安装在其他命名空间中，或使用不同的后台控制器 ServiceAccount 名称，请在应用 YAML 之前更新 `subjects` 部分。

创建生成策略

创建一个 `ClusterPolicy`，仅当 `app-team-a/app-notification-source` 更新且其 `data` 字段发生变化时，才生成 `NotificationMessage`。

```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
```

生成规则使用以下关键字段：

- `match.resources`：仅监视指定的 ConfigMap，并且仅监视 `UPDATE` 操作。
- `preconditions`：比较新旧 `data` 字段，因此仅元数据更新不会生成消息。
- `generate.apiVersion`、`kind`、`name` 和 `namespace`：定义目标资源的标识。
- `generate.data`：定义生成资源的完整内容。
- `synchronize: false`：为每次匹配的更新创建单独的目标资源。

生成的 `NotificationMessage` 使用 `spec.body.labels` 作为模板变量，使用 `ephemeral.methods` 选择电子邮件传递方式，使用 `ephemeral.template` 引用电子邮件模板，并使用 `receivers[].destination` 指定收件人的电子邮件地址。

验证生成规则

更新受监视的 ConfigMap。

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

检查 Kyverno 后台控制器日志。日志显示 Kyverno 创建了生成的目标资源。

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

日志输出包含类似于以下内容的生成目标：

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

对于此示例，还应检查通知 API 日志。成功发送电子邮件时，会显示一个任务、电子邮件发件人和收件人，以及 `Done` 状态。

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

日志输出包含类似于以下内容的条目：

```
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` 资源的生命周期很短，会由通知 API 消费。在消息被接受后，普通的 `kubectl get` 命令可能不会显示生成的消息。

故障排查

如果未生成目标资源，请检查 Kyverno 更新请求。

```
kubectl get updaterequests -A
```

如果更新请求失败，请检查详细错误信息。

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

常见原因包括：

- 源资源更新与 `match.resources` 筛选条件不匹配。
- ConfigMap 更新未更改 `data` 字段，因此未满足前置条件。
- Kyverno 后台控制器没有权限在 `cpaas-system` 中创建 `notificationmessages.aitextensions.alauda.io` 资源。
- 目标集群中没有 `notificationmessages.aitextensions.alauda.io/v1beta1` API。
- 生成的资源名称超过了 Kubernetes DNS 标签长度限制。
- `ephemeral.template` 引用的电子邮件模板不存在。
- 电子邮件通知服务器未配置或不可用。

API Refiner

介绍

产品介绍

限制

安装 Alauda Container Platform API Refiner

通过控制台安装

通过 YAML 安装

卸载流程

默认配置

介绍

目录

产品介绍

限制

产品介绍

ACP API Refiner 是由 Alauda Container Platform 提供的数据过滤服务，旨在增强 Kubernetes 环境中的多租户安全性和数据隔离。它基于用户权限、项目、集群和命名空间对 Kubernetes API 响应数据进行过滤，同时支持字段级过滤、包含和数据脱敏。

限制

以下限制适用于 ACP API Refiner：

- 资源必须包含特定的租户相关标签以实现数据隔离：
  - `cpaas.io/project`
  - `cpaas.io/cluster`
  - `cpaas.io/namespace`
  - `kubernetes.io/metadata.name`

  - 可选：`cpaas.io/creator`

- LabelSelector 查询不支持逻辑或操作
- 平台级别的 userbindings 不会被过滤
- 过滤仅应用于 GET 和 LIST API 操作

安装 Alauda Container Platform API Refiner

Alauda Container Platform API Refiner 是一个过滤 Kubernetes API 响应数据的平台服务。它提供按项目、集群和命名空间的过滤能力，支持在 API 响应中进行字段排除、包含和脱敏。

目录

通过控制台安装
通过 YAML 安装
1. 检查可用版本
2. 创建 ModuleInfo
卸载流程
默认配置
过滤资源
字段脱敏

通过控制台安装

1. 进入 管理员
2. 在左侧导航栏点击 **Marketplace** > 集群插件
3. 在顶部导航栏选择 **global** 集群
4. 搜索 **Alauda Container Platform API Refiner** 并点击查看详情

5. 点击 安装 以部署该插件

通过 YAML 安装

1. 检查可用版本

确认插件已发布，通过检查 `global` 集群中的 ModulePlugin 和 ModuleConfig 资源：

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

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

这表示集群中存在 ModulePlugin `apirefiner`，且版本 `v4.0.4` 已发布。

2. 创建 ModuleInfo

创建一个 ModuleInfo 资源以安装插件，无需配置参数：

```
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
```

字段说明：

- `name`：集群插件的临时名称。平台会根据内容重命名，格式为 `<cluster-name>-<内容哈希>`，例如 `global-ee98c9991ea1464aaa8054bdacbab313`。
- `label cpaas.io/cluster-name`：API Refiner 只能安装在 `global` 集群，此字段保持为 global。
- `label cpaas.io/module-name`：插件名称，必须与 ModulePlugin 资源匹配。
- `label cpaas.io/module-type`：固定字段，必须为 `plugin`；缺失此字段会导致安装失败。
- `.spec.config`：如果对应的 ModuleConfig 为空，此字段可留空。
- `.spec.version`：指定要安装的插件版本，必须与 ModuleConfig 中的 `.spec.version` 匹配。

卸载流程

1. 按安装流程的步骤 1-4 定位插件
2. 点击 卸载 移除插件

默认配置

过滤资源

默认过滤以下资源：

资源	**API** 版本
namespaces	v1
projects	auth.alauda.io/v1
clustermodules	cluster.alauda.io/v1alpha2
clusters	clusterregistry.k8s.io/v1alpha1

字段脱敏

默认脱敏以下字段：

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

关于 Alauda Container Platform 合规服务

合规服务是一个平台模块，旨在支持 STIG 合规扫描和 Alauda OS 操作系统扫描。它提供开箱即用的合规扫描能力，并支持定时扫描和全面报告。

Note

因为 Compliance Service 的发版周期与灵雀云容器平台不同，所以 Compliance Service 的文档现在作为独立的文档站点托管在 Compliance Service ↗。

平台安全配置

更改平台 OIDC Client Secret

- 前提条件
- OIDC Secret 概览
- 操作步骤
- 回滚

禁用 PKCE Plain 方法

- 前提条件
- 操作步骤
- 验证
- 回滚

更改平台 OIDC Client Secret

ACP 使用 OIDC Client Secret 在平台组件之间进行身份验证。出于安全合规要求、定期凭证轮换或潜在凭证泄露后的处理需要，您可能需要轮换此 Secret。在 global 集群上更新 Secret 后，平台会自动将其同步到所有工作负载集群。

目录

前提条件
OIDC Secret 概览
操作步骤
备份当前 Secret
更新 Client Secret
验证组件运行状况
回滚

前提条件

- 具有 `kubectl` 访问权限，并拥有 global 集群的 cluster-admin 权限。
- 平台管理的所有集群均已升级到 ACP v4.3。
- 具有 `Agnostic` 生命周期的以下插件已升级到与 ACP v4.3 兼容的版本：
  - `Alauda Container Platform Gitops`

- `Alauda Service Mesh Console`
- `Kubeflow Base`
- `Alauda AI`

- 已准备新的 Client Secret 值。建议使用至少包含 32 个字符的密码学随机字符串。

> **WARNING**
>
> 更改 Client Secret 会导致受影响的组件重启，从而造成暂时的身份验证中断。请在维护时段内规划此操作。

OIDC Secret 概览

平台 OIDC 凭证存储在 Kubernetes Secret 中，详细信息如下：

属性	值
Secret 名称	`cpaas-oidc-secret`
命名空间	`cpaas-system`

该 Secret 包含两个数据键：

- `client-id` — OIDC 客户端标识符，默认值为 `alauda-auth`。请勿修改此值。
- `client-secret` — OIDC Client Secret。这是您要轮换的值。

操作步骤

1 备份当前 Secret

在进行更改之前，请备份当前的 Secret manifest，以便在需要时执行回滚。

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

> **WARNING**
>
> 备份文件包含敏感凭证数据。请将其存储在安全位置，并在操作完成后将其删除。

记录当前的 `client-secret` 哈希，以便在更新后进行非明文验证：

```
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 更新 Client Secret

> **DANGER**
>
> 请勿修改 `client-id` 值，也不要删除并重新创建 Secret。这样做可能会导致整个平台出现连锁身份验证故障。

使用以下任一方法，仅更新 `client-secret` 数据字段。

方法 **1**：使用 **`kubectl patch`**（推荐）

从安全的交互式输入中读取新 Secret：

```
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
```

方法 **2**：使用 `kubectl edit`

在默认编辑器中打开 Secret，并将 `client-secret` 值替换为新的 base64 编码值。

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

TIP

您可以提前生成 base64 编码值：

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

在不暴露明文的情况下，验证 global 集群上的 Secret 已更新：

```
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 验证组件运行状况

Secret 更新后，`base-operator` 会自动将其同步到所有工作负载集群。依赖 OIDC 凭证的某些组件将重启以加载新值。请特别验证 `frontend` 和 `apollo` 部署是否正常运行：

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

验证所有 Pod 均为 `READY` 和 `UP-TO-DATE`。重启完成可能需要几分钟。

> **INFO**
>
> 如果某个组件无法启动，请确认 `cpaas-oidc-secret` Secret 存在于 `cpaas-system` 命名空间中，并且包含有效的 `client-id` 和 `client-secret` 值。

回滚

如果更改 Secret 后出现问题，请从在步骤 1 中创建的备份恢复之前的值：

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

或者，如果你知道之前的 Secret 值：

```
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
```

回滚后，请重复验证步骤，确认所有组件均运行正常。

禁用 PKCE Plain 方法

从 ACP v4.3.0 开始，默认启用 PKCE（Proof Key for Code Exchange）验证，以进行登录授权。为了保持与使用 `Agnostic` 生命周期且尚未升级的受影响插件的向后兼容性，平台在升级后会保留 `plain` code challenge 方法，同时保留安全的 `S256` 方法。

所有受影响的 `Agnostic` 插件都升级到与 ACP v4.3.0 兼容的版本后，应移除 `plain` 方法，以强制仅使用 `S256` 进行 PKCE 验证。

目录

前提条件

操作步骤

验证

回滚

前提条件

- 具有以集群管理员权限访问 **global** 集群 的 `kubectl`。
- 平台管理的所有集群都已升级到 ACP v4.3。
- 以下使用 `Agnostic` 生命周期的受影响插件已升级到与 ACP v4.3 兼容的版本：
  - `Alauda Container Platform Gitops`

- `Alauda Service Mesh Console`
- `Kubeflow Base`
- `Alauda AI`

> **WARNING**
>
> 在此上下文中，受影响插件是所列出的使用平台 OIDC 授权流程的 `Agnostic` 插件；在所有这些插件都升级到与 ACP v4.3 兼容的版本之前移除 `plain` 将导致身份验证失败。

操作步骤

OIDC 客户端配置以 `OAuth2Client` 自定义资源的形式存储在 `cpaas-system` 命名空间中。根据客户端 ID 确定目标资源名称：

```
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}"
```

编辑 `OAuth2Client` 资源：

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

找到 `codeChallengeMethods` 字段并移除 `plain` 条目，仅保留 `S256`：

```
# Before
codeChallengeMethods:
- S256
- plain

# After
codeChallengeMethods:
- S256
```

保存并退出编辑器。更改会立即生效。

验证

确认已移除 `plain` 方法：

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

输出应为：

```
["S256"]
```

对每个受影响的插件（`Alauda Container Platform Gitops`、`Alauda Service Mesh Console`、`Kubeflow Base` 和 `Alauda AI`）至少执行一次登录/授权验证，以确认不存在与 PKCE 相关的身份验证失败。

回滚

如果移除 `plain` 方法后出现身份验证问题（例如，任何受影响的插件登录或授权失败），请将其重新添加：

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

回滚后，再次验证受影响插件的登录，然后执行 `unset OIDC_CLIENT_NAME`。

用户与角色

用户

介绍	操作指南	操作指南
用户来源		
用户管理规则		
角色分配视图		
用户生命周期		

组

介绍	操作指南
组介绍	
组类型	

角色

介绍

角色介绍

系统角色

自定义权限

操作指南

IDP

介绍

Overview

Supported Integration Methods

操作指南

故障排除

用户策略

介绍

Overview

Configure Security Policy

Available Policies

用户

介绍

介绍

用户来源

用户管理规则

角色分配视图

用户生命周期

操作指南

管理用户角色

分配平台角色（系统模板）

绑定 Kubernetes 角色（RoleBinding / Clus

创建用户

通过控制台创建用户

通过 YAML 创建用户

用户管理

重置本地用户密

更新用户过期时

激活用户

禁用用户

将用户添加到本

查看用户角色

删除用户

批量操作

操作指南

获取 API 访问令牌

概述

前提条件

步骤

完整示例

使用令牌

重要说明

介绍

平台支持所有用户的身份认证和登录验证。

目录

用户来源
本地用户
第三方用户
LDAP 用户
OIDC 用户
其他第三方用户
用户管理规则
角色分配视图
用户生命周期

用户来源

本地用户

- 平台部署时创建的管理员账号
- 通过平台界面创建的账号

- 通过本地 dex 配置文件添加的用户

第三方用户

LDAP 用户

- 从 LDAP 服务器同步的企业用户
- 通过 IDP (Identity Provider) 集成导入账号
- 来源显示为 IDP 配置名称
- 集成通过 IDP 设置进行配置

OIDC 用户

- 通过 OIDC 协议认证的第三方平台用户
- 来源显示为 IDP 配置名称
- 集成通过 IDP 设置进行配置

> **WARNING**
>
> 对于首次登录前已添加到项目的 OIDC 用户：
>
> - 来源显示为 "-"，直到成功登录平台
> - 成功登录后，来源变更为 IDP 配置名称

其他第三方用户

- 通过支持的 dex 连接器（如 GitHub、Microsoft）认证的用户
- 更多信息请参见 dex 官方文档↗

用户管理规则

> **WARNING**

请注意以下重要规则：

- 本地用户名在所有用户类型中必须唯一
- 第三方用户（OIDC/LDAP）若用户名匹配，则自动关联
- 关联用户继承已有账号的权限
- 用户可通过各自来源登录
- 平台中每个用户名仅显示一条用户记录
- 用户来源由最近一次登录方式决定

角色分配视图

每个用户详情页现包含两个专用标签页，用于角色的查看和维护：

- **Platform Roles**（只读）：列出绑定到用户的系统提供的 platform/project/namespace 角色。此类角色在 UI 中不可编辑或复制，但可根据需要解绑。
- **Kubernetes Roles**：显示所有引用该用户的 RoleBinding/ClusterRoleBinding 对象（跨集群）。管理员可在此创建或移除绑定，以授予原生 Kubernetes 权限。

默认访问模板请使用 Platform Roles 标签页，需通过原生角色进行细粒度控制时请使用 Kubernetes Roles 标签页。

用户生命周期

下表描述了平台上不同用户状态：

状态	描述
正常	用户账号处于激活状态，可登录平台
禁用	用户账号处于非激活状态，无法登录。请联系平台管理员激活。 可能原因： - 连续 90 天以上未登录

状态	描述
	- 账号过期 - 管理员手动禁用
锁定	由于 24 小时内连续 5 次登录失败，账号被临时锁定。 详情： - 锁定时长：20 分钟 - 可由管理员手动解锁 - 锁定期满后账号自动恢复可用
无效	LDAP 同步账号已从 LDAP 服务器删除。 注意：无效账号无法登录平台

操作指南

管理用户角色

分配平台角色（系统模板）

绑定 Kubernetes 角色（RoleBinding / Clus

创建用户

通过控制台创建用户

通过 YAML 创建用户

用户管理

重置本地用户密

更新用户过期时

激活用户

禁用用户

将用户添加到本

查看用户角色

删除用户

批量操作

管理用户角色

平台管理员可以管理其他用户（非自身账户）的角色，以授予或撤销权限。RBAC 重构后，角色分配分为 平台角色（系统模板）和 **Kubernetes** 角色（原生 RoleBinding 对象）。

目录

分配平台角色（系统模板）

步骤

移除平台角色

绑定 Kubernetes 角色（RoleBinding / ClusterRoleBinding）

步骤

移除 Kubernetes RoleBindings

分配平台角色（系统模板）

使用此标签页绑定或解绑产品自带的预定义平台/项目/命名空间角色。

步骤

1. 在左侧导航栏，点击 **Users** > **User Management**。
2. 点击目标用户的用户名。
3. 打开 **Platform Roles** 标签页。

4. 点击 **Add Role**。
5. 在弹窗中：
   - 从 **Role Name** 下拉菜单中选择一个角色。
   - 如有提示，选择作用域（Cluster / Project / Namespace）。
   - 点击 **Add**。

> **NOTE**
>
> 平台角色注意事项：
>
> - 此处仅提供预定义的系统角色，无法编辑或复制。
> - 每个作用域内角色只能绑定一次。已绑定的角色在下拉菜单中禁用。
> - 内置的 Cluster Administrator 角色无法在 global 集群中重新分配。

移除平台角色

1. 保持在 **Platform Roles** 标签页。
2. 点击要解绑角色旁的 **Remove**。
3. 确认移除操作。

绑定 **Kubernetes** 角色（**RoleBinding / ClusterRoleBinding**）

使用此标签页通过特定集群内的原生 Kubernetes 角色授予细粒度权限。

步骤

1. 在用户详情页，切换到 **Kubernetes Roles** 标签页。
2. 点击 **Add RoleBinding**。
3. 配置绑定信息：

- **Cluster**：承载该角色的目标集群。
- **Binding Type**：`RoleBinding`（命名空间作用域）或 `ClusterRoleBinding`。
- **Namespace**：选择 `RoleBinding` 时必填。
- **Role Name**：选择已有的 `Role` 或 `ClusterRole`。
- **Subject**：确认当前用户为绑定主体。

4. 点击 **Create**。

移除 Kubernetes RoleBindings

1. 保持在 **Kubernetes Roles** 标签页。
2. 定位绑定（可按集群、命名空间或角色筛选）。
3. 点击 **Remove** 并确认。

> **WARNING**
>
> 角色管理权限：
>
> - 仅平台管理员可以管理其他用户的角色。
> - 用户无法修改自身账户的角色或绑定。

创建用户

具有平台管理员角色的用户可以通过平台界面创建本地用户并为其分配角色。

目录

通过控制台创建用户

通过 YAML 创建用户

通过控制台创建用户

1. 在左侧导航栏中，点击 **Users** > **User Management**
2. 点击 **Create User**
3. 配置以下参数：

参数	描述
Password Type	选择密码生成方式： **Random**：系统生成安全随机密码 **Custom**：用户手动输入密码
Password	根据所选类型输入或生成密码。 密码要求：

参数	描述
	- 长度：8-32 个字符 - 必须包含字母和数字 - 必须包含特殊字符（`~!@#$%^&*() -_=+?`） 密码字段功能： - 点击眼睛图标显示/隐藏密码 - 点击复制图标复制密码
Mailbox	用户邮箱地址： - 必须唯一 - 可作为登录用户名 - 与用户名关联
Validity Period	设置用户账户有效期： 选项： - **Permanent**：无限期 - **Custom**：通过时间范围下拉框设置起止时间
Roles	为用户分配一个或多个角色
Continue Creating	切换开关控制创建后行为： - 开启：跳转到新用户创建页面 - 关闭：显示用户详情页面

4. 点击 **Create**

> **NOTE**
>
> 用户创建成功后：
>
> - 若“Continue Creating”开启，将跳转到创建新用户页面
> - 若关闭，则显示已创建用户的详情页面

通过 YAML 创建用户

您可以在 `global` 集群中提交以下 YAML 来创建用户。

```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 type for external authentication (leave empty for local users)
    auth.cpaas.io/user.email: c18c9911faaac4e1051a599b88c62af2  # MD5 hash of the username (spec.email)
    auth.cpaas.io/user.state: active                       # User state; must match spec.state
    auth.cpaas.io/user.username: ""                        # User display 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 identifier; 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 account state: active or inactive
  username: ""                                             # Display name for the user
  valid: true                                              # Whether the user account is valid; should be set to true
```

用户管理

平台提供灵活的用户管理功能，支持单个用户管理和批量操作，以提升特定场景（如现场或远程团队）的效率。

WARNING

重要限制：

- 系统生成的账户无法管理（平台管理员角色，本地来源）
- 当前登录用户无法管理自己的账户
- 个人账户修改（显示名称、密码）请使用个人信息页面

目录

重置本地用户密码
操作步骤
更新用户过期时间
操作步骤
激活用户
操作步骤
禁用用户
操作步骤
将用户添加到本地用户组
操作步骤

查看用户角色
删除用户
操作步骤
批量操作
操作步骤

重置本地用户密码

具有平台管理权限的用户可以重置其他本地用户的密码。

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户记录旁的图标
3. 点击 **Reset Password**
4. 在弹窗中选择密码类型：
   - **Random**：系统生成安全随机密码
   - **Custom**：手动输入新密码

> **NOTE**
>
> 密码要求：
>
> - 长度：8-32 个字符
> - 必须包含字母和数字
> - 必须包含特殊字符（`~!@#$%^&*() -_=+?`）
>
> 密码字段功能：
>
> - 点击眼睛图标显示/隐藏密码
> - 点击复制图标复制密码

5. 点击 **Reset**

更新用户过期时间

您可以更新处于 **normal**、**disabled** 或 **locked** 状态用户的过期时间。超过过期时间的用户将自动被禁用。

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户旁的 **Update Expiry Date**
3. 在弹窗中选择过期时间选项：
   - **Permanent**：无限期
   - **Custom**：通过时间范围下拉框设置开始和结束时间
4. 点击 **Update**

激活用户

您可以激活处于 **disabled** 或 **locked** 状态的用户。

> **NOTE**
>
> 激活行为：
>
> - 用户在有效期内：过期时间保持不变
> - 用户已过期：过期时间变为 **Permanent**

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户旁的 **Activate**

3. 在确认弹窗点击 **Activate**
4. 用户状态将变为 **normal**

禁用用户

您可以禁用处于 **normal** 或 **locked** 状态且在有效期内的用户。被禁用用户无法登录，但可以重新激活。

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户旁的图标
3. 点击 **Disable** 并确认

将用户添加到本地用户组

您可以将 **Source** 为 **Local** 或 `LDAP` 的用户添加到一个或多个本地用户组。

> **WARNING**
>
> 组角色行为：
>
> - 用户会自动继承其所在组的角色
> - 组角色仅在组详情页的 **Platform Roles** 标签页可见
> - 个人用户角色列表仅显示直接分配的角色

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户旁的图标
3. 点击 **Add to User Group**

4. 选择一个或多个本地用户组
5. 点击 **Add**

查看用户角色

RBAC 重构后，每个用户详情页显示两个标签页：

- **Platform Roles**：展示绑定给用户的系统内置角色。您可以添加或移除绑定，但无法编辑角色定义。
- **Kubernetes Roles**：列出所有引用该用户的 RoleBinding/ClusterRoleBinding（跨集群）。您可以直接在此标签页创建或删除绑定。

详细操作请参见 Manage User Roles。

删除用户

平台管理员可以删除除当前登录账户外的任意用户，包括：

- IDP 配置的用户
- 来源为 `-` 的用户
- 本地用户

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 点击目标用户旁的图标
3. 点击 **Delete**
4. 点击 **Confirm**

批量操作

您可以对用户执行批量操作：

- 更新有效期
- 激活用户
- 禁用用户
- 删除用户

操作步骤

1. 在左侧导航栏点击 **Users** > **User Management**
2. 通过复选框选择一个或多个用户
3. 点击 **Batch Operations** 并选择操作：
   - **Update Validity**
   - **Activate**
   - **Deactivate**
   - **Delete**

> **NOTE**
>
> 批量操作详情：
>
> - **Update Validity**：设置为永久或自定义时间范围
> - **Activate**：在弹窗确认激活
> - **Deactivate**：在弹窗确认禁用
> - **Delete**：输入当前账户密码并确认

操作指南

获取 **API** 访问令牌

概述

前提条件

步骤

完整示例

使用令牌

重要说明

获取 API 访问令牌

本指南说明如何通过 API 调用获取 ACP 平台访问令牌。适用于需要以编程方式访问平台 API 的场景。

目录

概述
前提条件
步骤
第 1 步：获取登录元数据
第 2 步：获取 Dex 请求 ID
第 3 步：获取 RSA 公钥
第 4 步：提交加密凭据并获取授权码
第 5 步：将授权码交换为令牌
完整示例
Shell Script（curl）
使用令牌
重要说明

概述

ACP 使用基于 Dex 的 OIDC 身份验证系统。登录流程遵循 OAuth 2.0 Authorization Code Flow，由五个 API 调用组成，且必须在同一个 **HTTP** 会话中完成（共享 **cookie**）。

```
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

整个流程必须使用同一个 HTTP 会话（共享 cookie jar）。第 2 步会设置 `cpaas_oidc_auth_flow` 会话 cookie，第 5 步需要该 cookie 才能签发令牌。如果没有此 cookie，第 5 步将返回 `invalid authentication session`。

前提条件

参数	描述	值
`PLATFORM_URL`	平台 HTTPS 基础 URL（不包含尾随路径）	`https://<platform>`
`CLIENT_ID`	OAuth client ID（固定）	`alauda-auth`
`SCOPE`	请求的权限范围（固定）	`openid profile offline_access email groups ext`

参数	描述	值
`REDIRECT_URI`	回调 URL（固定）	`{PLATFORM_URL}/dex/callback`
`IDP`	Identity Provider ID	`local`（本地账号默认）、`ldap`、`oidc` 等。

步骤

第 1 步：获取登录元数据

请求：

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

查询参数：

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

响应（**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

`auth_url` 中的 `code_challenge` 及相关参数由服务端生成，用于 server-to-Dex PKCE 交换，并存储在 `cpaas_oidc_auth_flow` cookie 中。你无需解析它们。在第 2 步中，请原样传递 `auth_url` 中的完整查询字符串。

第 2 步：获取 Dex 请求 ID

使用第 1 步返回的 `auth_url` 中的查询参数。

此步骤会设置 `cpaas_oidc_auth_flow` 会话 **cookie**。后续所有请求都必须携带该 **cookie**。

请求：

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

其中 `{query_from_auth_url}` 是从第 1 步 `auth_url` 中提取的完整查询字符串（请保持原样传递）：

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

响应（**200 OK**）：

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

第 3 步：获取 RSA 公钥

请求：

```
GET /dex/pubkey
```

响应（**200 OK**）：

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

字段	描述
`ts`	时间戳。必须包含在加密负载中。每次登录尝试都需要获取一个新的值。
`pubkey`	PEM 格式的 RSA 公钥，用于加密密码。
`pubkey_encode`	`pubkey` 的 Base64 编码形式。与 `pubkey` 等价。

第 4 步：提交加密凭据并获取授权码

密码加密：

1. 构造 JSON 负载：`{"ts": "<第 3 步中的 ts>", "password": "<明文密码>"}`
2. 使用 RSA 公钥并采用 **PKCS#1 v1.5** 填充对 JSON 字节进行加密
3. 对加密结果进行 Base64 编码（标准编码，不是 URL-safe）

请求：

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

路径参数：

参数	描述
`{idp}`	Identity Provider ID：`local`（本地账号）、`ldap` 等。

查询参数：

参数	值
`req`	第 2 步返回的 `req` ID

请求体：

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

响应（**200 OK**）：

```
{
  "session_state": "",
  "redirect_url": "https://{platform}/dex/callback?code=qdul75ionlvlsn3gg
g2m...&state=VpweqrTxqH..."
}
```

从 `redirect_url` 中提取 `code` 和 `state` 参数，用于第 5 步。

第 5 步：将授权码交换为令牌

请求：

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

查询参数：

参数	来源	必填
`code`	从第 4 步的 `redirect_url` 中提取	是
`state`	从第 4 步的 `redirect_url` 中提取	是

响应（**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"
}
```

字段	描述
`access_token`	用于 API 访问的 JWT。在请求头中以 `Authorization: Bearer <token>` 传递。

字段	描述
`id_token`	包含用户身份声明（email、username 等）的 JWT。
`refresh_token`	用于刷新 `access_token` 的不透明字符串。
`expire_at`	`access_token` 的过期时间（UTC）。
`issued_at`	`access_token` 的签发时间（UTC）。
`token_storage`	令牌存储位置。固定值：`local`。

完整示例

Shell Script（curl）

curl 通过 `-c`（写入 cookie）和 `-b`（读取 cookie）共享 cookie jar 文件来维护会话。

依赖项：`curl`、`jq`（JSON 解析）、`openssl`（RSA 加密）

用法：

```
# 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 "$@"
```

使用令牌

在后续所有 API 请求的 `Authorization` 头中包含 `access_token`：

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

重要说明

1. 必须共享会话 **cookie**：整个登录流程必须使用同一个 HTTP 会话（共享 cookie jar），以便 `cpaas_oidc_auth_flow` cookie 能在每一步中自动携带。没有此 cookie，第 5 步将返回 `invalid authentication session (400)`。
2. **`ts`** 时间戳不能复用：第 3 步返回的 `ts` 对每次请求都是唯一的。它必须在加密前与密码一起组合成 JSON 负载。复用之前的 `ts` 值会导致第 4 步认证失败。

3. 选择 `IDP` 参数：
   - `local`：ACP 本地账号（默认）
   - `ldap`：LDAP/AD 域账号；具体 ID 取决于你的平台配置
4. **TLS** 证书校验：示例脚本中的 `-k` 标志会跳过证书校验。这仅适用于使用自签名证书的测试环境。在生产环境中，请移除 `-k`，并配置有效的 CA 证书，或将平台 CA 添加到系统信任库中。

组

介绍

介绍

组介绍

组类型

操作指南

管理用户组角色

向用户组添加角色

从用户组移除角色

创建本地用户组

创建用户组

管理用户组

管理本地用户

前提条件

导入成员

移除成员

介绍

目录

组介绍
组类型
本地用户组
IDP 同步用户组

组介绍

平台通过用户组支持用户管理。通过管理组角色，您可以高效地：

- 同时授予多个用户平台操作权限
- 一次性撤销多个用户的权限
- 实现基于角色的批量访问控制

例如，当企业内部人员变动，需要为多个用户授予新的项目或命名空间操作权限时，您可以：

1. 创建一个用户组
2. 导入相关用户作为组成员
3. 配置该组的项目和命名空间角色
4. 将统一权限应用于所有组成员

组类型

平台支持两种类型的组：

本地用户组

- 直接在平台上创建
- 来源显示为 **Local**
- 可以更新或删除
- 支持：
  - 添加或移除任何来源的用户
  - 添加或移除角色

IDP 同步用户组

- 从已连接的 IDP（LDAP、Azure AD）同步
- 来源显示为已连接的 ***IDP*** 名称
- 不可更新或删除
- 支持：
  - 添加或移除角色
  - 不支持管理组成员（添加或移除）

操作指南

管理用户组角色

向用户组添加角色

从用户组移除角色

创建本地用户组

创建用户组

管理用户组

管理本地用户

前提条件

导入成员

移除成员

管理用户组角色

具有平台管理权限的用户可以管理本地用户组和 IDP 同步用户组的角色。

目录

向用户组添加角色

步骤

从用户组移除角色

步骤

向用户组添加角色

步骤

1. 在左侧导航栏中，点击 **Users** > **User Group Management**
2. 点击目标用户组的名称
3. 在 **Configure Role** 选项卡中，点击 **Add Role**
4. 点击添加角色

> **NOTE**
>
> 角色分配规则：

- 可以向一个用户组添加多个角色
- 同一个角色在同一用户组中只能添加一次

5. 从下拉菜单中选择角色名称
6. 选择角色的权限范围（集群、项目或命名空间）
7. 点击 **Add**

从用户组移除角色

WARNING

当您从用户组移除角色时：

- 该角色授予组成员的所有权限将被撤销
- 此操作不可撤销

步骤

1. 在左侧导航栏中，点击 **Users** > **User Group Management**
2. 点击目标用户组的名称
3. 在 **Configure Role** 选项卡中，点击角色旁的 **Remove**
4. 点击 **Confirm** 以移除角色

创建本地用户组

本地用户组允许您为来自任何来源的多个用户实施基于角色的访问控制。

目录

创建用户组

步骤

管理用户组

创建用户组

步骤

1. 在左侧边栏，点击 **Users** > **User Group Management**
2. 点击 **Create User Group**
3. 输入以下信息：
   - **Name**：用户组名称
   - **Description**：用户组用途描述
4. 点击 **Create**

管理用户组

您可以通过点击列表页上的图标或在详情页右上角点击 **Operations** 来管理用户组。

操作	描述
Update User Group	根据用户组来源更新组信息： - 对于 **Source** 为 `Local` 的组：可以更新名称和描述 - 对于 **Source** 为 `IDP name` 的组：只能更新描述
Delete Local User Group	删除 **Source** 为 `Local` 的用户组

WARNING

当您删除用户组时：

- 所有组成员将被移除
- 分配给该组的所有角色将被移除
- 此操作不可撤销

管理本地用户组成员

只有具有平台管理权限的用户才能管理本地用户组成员。

目录

前提条件
导入成员
操作步骤
移除成员
操作步骤

前提条件

WARNING

在管理组成员之前，请注意以下限制：

- 只有具有平台管理权限的用户才能管理组及其成员
- 系统账户和当前登录账户不能被管理（导入到组或从组中移除）
- 每个本地用户组最多可包含 5000 名成员
- 当组达到 5000 名成员的限制时，不允许再导入成员

导入成员

您可以将平台中的用户导入到本地用户组中，实现统一的权限管理。

> **TIP**
>
> 导入到组中的用户将自动继承该组分配的所有操作权限。

操作步骤

1. 在左侧导航栏中，点击 **Users** > **User Group Management**
2. 点击要添加成员的本地用户组名称
3. 在 **Group Member Management** 标签页，点击 **Import Member**
4. 通过勾选用户名/显示名旁的复选框，选择一个或多个平台用户
5. 点击 **Import**

> **NOTE**
>
> - 只能选择当前不属于该组的用户
> - 使用 **Import All** 按钮可一次性导入列表中的所有用户

移除成员

当您从组中移除用户时，该用户通过该组获得的所有操作权限将被自动撤销。

操作步骤

1. 在左侧导航栏中，点击 **Users** > **User Group Management**
2. 点击要移除成员的本地用户组名称
3. 在 **Group Member Management** 标签页，您可以通过以下两种方式移除成员：

- 点击成员名称旁的 **Remove** 并确认
- 通过复选框选择一个或多个成员，点击 **Batch Remove** 并确认

角色

介绍

介绍

角色介绍

系统角色

自定义权限

操作指南

创建 **Kubernetes** 角色

前提条件

创建 Role 或 ClusterRole

编辑 Role YAML（可选）

创建 RoleBindings

验证

管理角色

查看平台角色（只读）

通过控制台授予用户平台角色

通过 YAML 授予用户平台角色

通过 YAML 查看和更新 Kubernetes 角色

删除 Kubernetes 角色

管理 RoleBindings

最佳实践

如何创建自定

1. 概述

2. 核心概念

3. 技术变更

4. 选择方法

5. 配置

6. 复制并精简

介绍

目录

角色介绍

系统角色

自定义权限

角色介绍

平台的用户角色管理基于 Kubernetes RBAC（基于角色的访问控制）实现。自 ACP 4.2 版本起，该模型被拆分为两个互补的层次：

- 平台角色：系统提供的模板，保障核心产品场景。它们在 UI 中为只读状态；您只能为用户和用户组绑定或解绑这些角色。
- **Kubernetes** 角色：原生 Kubernetes 的 `Role` 和 `ClusterRole` 对象，您可以针对每个集群创建以满足定制的权限需求。这些角色在 **Kubernetes Roles** 页面进行管理，并通过 RoleBinding/ClusterRoleBinding 绑定。

这两个层次最终都会转换为 Kubernetes 权限。分配或移除角色会立即授予或撤销对目标资源的相关操作权限（创建、查看、更新、删除等）。

系统角色

为满足常见的权限配置场景，平台提供以下默认系统角色。这些角色支持对平台资源的灵活访问控制和对用户的高效权限管理。

角色名称	描述	角色层级
Platform Administrator	拥有对平台上所有业务和资源的完全访问权限	Platform
Platform Auditors	可以查看所有平台资源和操作记录，但无其他权限	Platform
Cluster Administrator (Alpha)	管理和维护集群资源，拥有对所有集群级资源的完全访问权限	Cluster
Project Administrator	管理命名空间管理员和命名空间配额	Project
namespace-admin-system	管理命名空间成员和角色分配	Namespace
Developers	在命名空间内开发、部署和维护自定义应用	Namespace

自定义权限

已移除传统的“基于复选框”的自定义角色创建体验。要实现新的授权场景，您必须：

1. 使用 **Kubernetes Roles** 页面在目标集群中创建原生角色（Role/ClusterRole），或
2. 向所属插件团队申请角色模板，并通过 RoleBinding/ClusterRoleBinding 进行绑定。

> **WARNING**
>
> 删除或编辑原生角色会影响所有引用该角色的 RoleBinding/ClusterRoleBinding。请务必审查现有绑定，并在可能的情况下在预发布环境中验证更改。

操作指南

创建 Kubernetes 角色

前提条件

创建 Role 或 ClusterRole

编辑 Role YAML（可选）

创建 RoleBindings

验证

管理角色

查看平台角色（只读）

通过控制台授予用户平台角色

通过 YAML 授予用户平台角色

通过 YAML 查看和更新 Kubernetes 角色

删除 Kubernetes 角色

管理 RoleBindings

最佳实践

如何创建自定

1. 概述

2. 核心概念

3. 技术变更

4. 选择方法

5. 配置

6. 复制并精简

创建 Kubernetes 角色

从 ACP 4.2 开始，自定义权限通过原生 Kubernetes 角色交付。使用 **Kubernetes Roles** 页面（位于 **Users > Platform Roles > Kubernetes Roles** 下）来创建或管理当前选定集群中的 `Role` 和 `ClusterRole` 对象。

目录

前提条件

创建 Role 或 ClusterRole

编辑 Role YAML（可选）

创建 RoleBindings

验证

前提条件

- 您被分配了一个授予访问 Kubernetes Roles 功能的平台角色。
- 您已在 global 集群切换器中选择了目标集群。
- 集群中已包含您的角色将作用于的任何命名空间。

创建 Role 或 ClusterRole

1. 在左侧导航栏中，点击 **Users** > **Platform Roles** > **Kubernetes Roles**。
2. 点击 **Create Role**。
3. 在抽屉中：
   - 输入 **Name**（必须符合 Kubernetes 命名规则）。
   - 选择 **Type**（`Role` 或 `ClusterRole`）。
   - 如果选择了 `Role`，请选择作用域权限的 **Namespace**。
4. 通过添加一条或多条规则来配置：
   - **API Groups**
   - **Resources**
   - **Resource Names**（可选）
   - **Verbs**（`get`、`list`、`watch`、`create`、`update`、`patch`、`delete`）
5. 点击 **Create**。

角色会直接在集群内创建，并立即可用于 RoleBinding 操作。

编辑 Role YAML（可选）

1. 打开 **Kubernetes Roles** 标签页，点击角色名称。
2. 在 **YAML** 标签页，点击 **Edit**。
3. 更新标签、注解或规则等字段。
4. 点击 **Save** 以应用更改。

创建 RoleBindings

将新创建的角色授予用户或组：

1. 在查看角色时，切换到 **RoleBindings** 标签页。
2. 点击 **Create RoleBindings**。
3. 提供：

- **Name**
- **Binding Type**（`RoleBinding` 或 `ClusterRoleBinding` —— 当来源为命名空间作用域角色时，仅支持 `RoleBinding`）
- **Namespace**（针对 `RoleBinding`）
- **Subjects**（用户、组或 ServiceAccount 及对应名称）

4. 点击 **Create**。

或者，打开 **Users** 或 **User Groups** 页面，切换到 **Kubernetes Roles** 标签页，从用户视角直接创建绑定。

验证

使用以下方法之一确认角色存在：

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

或者刷新 **Kubernetes Roles** 列表，使用内置搜索（按名称或标签）定位角色。

管理角色

目录

查看平台角色（只读）
通过控制台授予用户平台角色
通过 YAML 授予用户平台角色
通过 YAML 查看和更新 Kubernetes 角色
删除 Kubernetes 角色
管理 RoleBindings
从角色视角
从用户或用户组视角
最佳实践

查看平台角色（只读）

平台角色仍然是核心功能的权威模板。

1. 在左侧导航栏中，点击 **Users** > **Platform Roles**。
2. 使用列表过滤器定位角色。**Role Type** 列现在显示 `Platform`、`Project`、`Namespace` 或 `Cluster`。
3. 点击角色名称以打开详情页面。
4. 切换到 **YAML** 标签页查看精确定义。如需存档规范，可使用 **Download YAML**。

通过控制台授予用户平台角色

1. 在左侧导航栏中，点击 **Users** > **Platform Roles**。
2. 点击角色名称以打开详情页面。
3. 切换到 **Members** 标签页。
4. 点击 **Import Members**。
5. 你可以从平台中选择用户并导入到该角色作为成员。

通过 YAML 授予用户平台角色

你可以在 `global` 集群中提交以下 YAML，向用户授予特定的平台角色。

apiVersion: auth.alauda.io/v1 kind: UserBinding metadata: annotations: auth.cpaas.io/role.display-name: Platform Admin # 要分配的角色显示名称 auth.cpaas.io/user.email: bxliu@alauda.io ↗ # 授予角色的用户名 labels: auth.cpaas.io/role.display-name: "" # 要分配的角色显示名称 auth.cpaas.io/role.level: platform # 角色作用域：platform、project、namespace 或 cluster auth.cpaas.io/role.name: acp-platform-admin # 要分配的角色名称 auth.cpaas.io/user.email: 569526aac97a17ce8c1c185d7544aae4 # 用户名的 MD5 哈希 cpaas.io/cluster: "" # 集群名称；当角色级别为 namespace 或 cluster 时必填，platform 或 project 时留空 cpaas.io/namespace: "" # 命名空间名称；当角色级别为 namespace 时必填，platform、project 或 cluster 时留空 cpaas.io/project: "" # 项目名称；当角色级别为 project 或 namespace 时必填，platform 或 cluster 时留空 name: dc30204c17c7fe8b15383f4ed7798c88 # UserBinding 资源名称；可自定义

通过 YAML 查看和更新 Kubernetes 角色

1. 进入 **Users** > **Platform Roles** > **Kubernetes Roles**。
2. 按名称或标签搜索。
3. 点击角色名称，打开 **YAML** 标签页。
4. 点击 **Edit**，修改清单（标签、注解或 `rules`），然后点击 **Save**。
5. 查看 **RoleBindings** 标签页，确保现有绑定仍符合预期。

删除 **Kubernetes** 角色

1. 在 **Kubernetes Roles** 列表中，点击角色旁的更多菜单（…）。
2. 选择 **Delete Role**。
3. 确认角色名称以继续。

删除角色会将其从集群中移除。你还必须清理引用该角色的任何 RoleBindings。若绑定仍存在，UI 会显示警告。

管理 **RoleBindings**

从角色视角

1. 在 **Kubernetes Roles** 标签页打开一个角色（Role 或 ClusterRole）。
2. 进入 **RoleBindings** 标签页。
3. 使用搜索栏（支持名称和标签过滤）定位现有绑定。
4. 操作：

- **Create RoleBindings**：启动创建向导。
- **Update Role**：打开角色本身的 YAML 编辑器。
- **Delete Binding**：确认后删除 RoleBinding/ClusterRoleBinding。

从用户或用户组视角

1. 打开 **Users**（或 **User Groups**），选择目标条目。
2. 切换到 **Kubernetes Roles** 标签页。
3. 查看该用户/组在各集群中的所有 RoleBindings。
4. 点击 **Add RoleBinding**，选择：

- 集群
- 绑定类型（RoleBinding/ClusterRoleBinding）
- 角色/ClusterRole

- 命名空间（针对 RoleBinding）
- 主题详情

5. 保存绑定。

此流程补充了现有的 **Platform Roles** 标签页，该标签页仍用于将系统角色附加给用户。

最佳实践

- 使用预发布集群验证 YAML 变更，再应用到生产环境。
- 将角色定义纳入版本控制（例如导出到 Git），确保变更可审计。
- 对所需权限有疑问时，可从系统角色的 YAML 开始，复制到本地，并通过新 UI 适配为 Kubernetes 角色。

如何创建自定义平台角色

目录

1. 概述
2. 核心概念
3. 技术变更
4. 选择方法
5. 配置
   5.1 方法 A：FunctionResource
   5.2 方法 B：ClusterRole 聚合（独立自定义角色）
   5.3 方法 C：customRules
   5.4 字段参考
   5.5 验证配置是否实际生效
      检查 RoleTemplate 状态
      确认用户确实获得权限
      在写入 `aggregationRules` 前检查聚合标签是否可用
   5.6 更新和删除
6. 复制并精简（示例）
   6.1 精简系统 RoleTemplate YAML（方法 A）
   6.2 构建多范围审计员角色（方法 B）

1. 概述

本文介绍自定义角色的核心概念，以及如何配置 RoleTemplate。构建可用自定义角色主要有两种方式——任选一种并完整执行：

- **FunctionResource** 方法（5.1）：从产品内置的权限目录中选择权限。最简单的选项，无需编写 RBAC。
- **ClusterRole** 聚合方法（5.2）：编写自己的原生 `ClusterRole`，并通过标签匹配将其引入 RoleTemplate。配置较为复杂，但不受任何产品模块迁移进度的影响，并且是平台的长期发展方向。

`customRules`（5.3）是第三种更底层的选项，用于一次性使用的原生 RBAC 规则。

不知道该选择哪一种？ 请从方法 **A**（**5.1**）开始——这是最简单的方式，几乎可以满足所有需求。只有在明确需要使用自有 `ClusterRole` 构建角色，或需要构建跨多个集群的角色时，才选择方法 **B**（**5.2**）。

RoleTemplate 创建后，可通过 UserBinding 将其授予用户——控制台操作步骤请参阅管理角色。

2. 核心概念

- **RoleTemplate**：自定义角色模板。它定义角色语义和权限集，控制器会将其转换为 ClusterRole。
- **FunctionResource**：产品功能使用的 K8s 资源抽象；在 RoleTemplate 中通过 `functionResourceRef` 引用。
- **ClusterRole**：RBAC 规则集。手动编写的 ClusterRole 可以通过标签聚合到 RoleTemplate（或内置系统角色）中。
- **UserBinding**：用户与角色/作用域之间的绑定；控制器会根据 UserBinding 生成 RoleBinding/ClusterRoleBinding，以完成最终授权。

> **NOTE**
>
> - `functionResourceRef` 的值来自 FunctionResource `metadata.name`。

- 显示名称通常位于 `metadata.annotations` 中，可用于映射到 UI 模块。

3. 技术变更

自 ACP v4.3 起，功能权限正逐步从 FunctionResource 迁移到原生 K8s ClusterRole 管理，并通过 ClusterRole 标签聚合到系统角色中；基于 FunctionResource 的管理方式将在 v4.5 中弃用。创建角色不需要了解这些背景——请跳转到第 5 节查看具体步骤。

> **NOTE**
>
> 迁移仍在进行中：大多数产品模块尚未发布可用于聚合的 **ClusterRole**，因此 5.4 中提到的内置系统标签（`aggregate-to-namespace-developer` 等）目前不会匹配到任何内容。不要将 `aggregationRules` 指向其中一个标签并寄希望于它能生效——5.2 展示了可行的模式，5.5 介绍了验证方法。（你还可能会在 `kubectl` 输出中看到隐藏的 `legacy-*` RoleTemplate；这些是内置角色的兼容性桥接对象，可以忽略。）

4. 选择方法

这三种方法都可以生成可正常工作的 RoleTemplate；区别在于需要编写的内容以及它们能够表达的权限范围。

	方法 **A**：**FunctionResource**（**5.1**）	方法 **B**：**ClusterRole** 聚合（**5.2**）	方法 **C**：**customRules**（**5.3**）
适用场景	重用产品已经公开的权限	使用自有 ClusterRole 构建独立角色，尤其适用于跨集群场景	快速添加一次性的原生 RBAC 规则
需要编写的内容	无需编写任何内容——从目录中选择	一个 ClusterRole 及其聚合标签	内联 RBAC 规则
多集群	由平台处理	将 `sync.scope: "platform"` 添加到	由平台处理

	方法 **A**：**FunctionResource**（**5.1**）	方法 **B**：**ClusterRole** 聚合（**5.2**）	方法 **C**：**customRules**（**5.3**）
		ClusterRole	
跨作用域规则	是	是——每个作用域使用一个 ClusterRole	否——仅支持基础 ClusterRole

5. 配置

CAUTION

始终在自定义 **RoleTemplate** 上设置 `rbac.cpaas.io/version: "v2"` 和 `auth.cpaas.io/roletemplate.frozen: "false"`。 对于自定义角色，控制器会将缺失的 `version` 自动填充为 `v1`（该值会忽略 `aggregationRules`），并将缺失的 `frozen` 自动填充为 `true`（该值会阻止后续对 `spec` 的编辑更新 **ClusterRole**），且两者独立处理——因此只设置其中一个，另一个仍会保留旧版默认值。以下示例还设置了 `auth.cpaas.io/roletemplate.official: "false"`（可选，但在自定义角色上切勿将其设置为 `"true"`——这会将其切换为内置系统角色）和 `auth.cpaas.io/roletemplate.level`，后者决定角色的授予范围——请选择满足需求的最小范围：`namespace`（每个命名空间）、`project`（项目中的所有命名空间）、`cluster`（整个集群）或 `platform`（整个平台）。

5.1 方法 A：FunctionResource

当所需权限已经通过 FunctionResource 公开时，请使用此方法——目前几乎所有产品模块都属于这种情况。

步骤 **1**——查找所需的 **FunctionResource**

```
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` 就是要填入 `functionResourceRef` 的值；显示名称注解会告诉你它映射到哪个 UI 功能。

步骤 **2**——编写 **RoleTemplate**

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-funcrole
  annotations:
    cpaas.io/display-name: Example Function Role
    cpaas.io/description: FunctionResource example
  labels:
    auth.cpaas.io/roletemplate.level: namespace
    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]
```

步骤 **3**——应用并验证，请参阅 5.5。

> **NOTE**
>
> 自定义角色中不允许使用 `acp-namespace-resource-manage`。

5.2 方法 B：ClusterRole 聚合（独立自定义角色）

与 FunctionResource 方法不同，`aggregationRules` 本身不定义权限——它只会从带有匹配标签的现有 ClusterRole 中引入 `rules`。要通过这种方式构建真正独立的自定义角色，需要自行提供两个部分：包含实际规则的 ClusterRole，以及将其连接到 RoleTemplate 的标签。

步骤 **1**——使用实际权限编写原生 **ClusterRole**，并为其添加自定义聚合标签和作用域标签。除非你明确希望将这些权限授予已经持有该内置角色的所有用户，否则不要重复使用内置系统角色标签（5.4）；对于独立角色，请自行创建标签名称。

```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"]
```

步骤 **2**——如果该角色还需要在工作负载集群中生效，且上面的 ClusterRole 只在 global 集群中创建，请添加 `rbac.cpaas.io/sync.scope: "platform"`，以便平台将其复制到每个集群。如果没有此标签，ClusterRole 只会存在于创建它的位置，而下面的 RoleTemplate 在任何没有该 ClusterRole 的集群上都会解析为空权限。

```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"
```

步骤 **3**——创建 **RoleTemplate**，通过 `aggregationRules` 引用相同的标签：

```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: []
```

要涵盖多个作用域（例如还包括 `cluster` 或 `system-ns`），请为每个作用域添加一个带有匹配作用域标签的 ClusterRole，并在 `aggregationRules` 下为每个作用域再添加一项——请参阅 6.2 中完整的多作用域示例。

步骤 **4**——应用并验证，请参阅 5.5。结果为空表示 ClusterRole 上的标签与 `clusterRoleSelectors` 中的标签实际上不匹配，或者 ClusterRole 尚未同步到你正在检查的集群（步骤 2）。

> **CAUTION**
>
> 不要将 `aggregationRules` 指向你*希望存在*的系统/产品标签（例如 `rbac.cpaas.io/aggregate-to-namespace-auditor`）——大多数模块尚未发布自己的标签，因此解析结果为空，角色也会静默地不授予任何权限。如上所示，使用自有 ClusterRole 和自有标签始终有效。

5.3 方法 C：customRules

对于无需单独 ClusterRole 对象的一次性原生 RBAC 规则，可以直接将其内联到 RoleTemplate 中：

```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` 只会写入角色的基础 ClusterRole——无法将其定向到特定作用域，例如 `cluster` 或 `system-ns`（`customRules` 上不存在 `scope` 字段）。如果需要跨作用域规则，请使用方法 A 或方法 B。

5.4 字段参考

第一个表格介绍元数据以及每个自定义角色都需要的标签；第二个表格介绍权限字段，并按照所属方法（5.1–5.3）进行分组。

元数据和标签（每个自定义 RoleTemplate 都需要）

字段	说明	值
`metadata.name`	RoleTemplate 的唯一名称；UserBinding 在向	自定义字

字段	说明	值
	用户授予角色时会引用该名称。	
`metadata.annotations.cpaas.io/display-name`、`.../description`	控制台中显示的可读名称和描述。	自定义字
`metadata.labels.auth.cpaas.io/roletemplate.level`	角色的授予范围，以及控制器为其生成哪些 ClusterRole。	`platfo` `cluste` `projec` `namesp`
`metadata.labels.auth.cpaas.io/roletemplate.official`	是否为内置系统角色。自定义角色不得声称自己是官方角色。	`"false` 省略；自 角色上绝 使用 `"t`
`metadata.labels.rbac.cpaas.io/version`	控制器应用哪种转换逻辑。`v2` 理解 `aggregationRules`；旧版 `v1` 会忽略它。	`"v2"` 版本）；时默认为 `"v1"`
`metadata.labels.auth.cpaas.io/roletemplate.frozen`	生成的 ClusterRole 是否锁定。`"false"` 允许后续对 `spec` 的编辑生效。	`"false` 略时默认 `"true"` 结）

权限字段（根据所用方法选择相应字段集）

字段	方法	说明
`spec.rules[].functionResourceRef`	A	要包含其权限的 FunctionResource 名称。使用 `kubectl get functionresource` 查找名称。

字段	方法	说明
`spec.rules[].verbs`	A	角色允许对该 FunctionResource 资源执行的操作。
`spec.aggregationRules[].scope`	B	匹配的规则要写入哪个生成的 ClusterRole——即规则的适用范围：`business-ns`（在项目下创建的命名空间）、`project-ns`（项目自身的命名空间）、`system-ns`（`cpaas-system`）、`cluster`（集群范围）、`kube-public`。必须与 `level` 兼容（请参阅下面的说明）。
`spec.aggregationRules[].clusterRoleSelectors`	B	根据标签选择要从中引入规则的现有 ClusterRole。
`spec.customRules[].apiGroups`	C	规则涵盖的 K8s API 组。
`spec.customRules[].resources`	C	规则涵盖的资源类型。

字段	方法	说明
`spec.customRules[].verbs`	C	允许对这些资源执行的操作。
`spec.customRules[].resourceNames`	C	可选：将规则限制为特定名称的对象。
`spec.customRules[].nonResourceURLs`	C	可选：授予对非资源 URL 的访问权限（例如 `/healthz`）；用于集群级角色。

NOTE

- 对于给定作用域，请使用直接规则（`spec.rules` / `spec.customRules`）或 `spec.aggregationRules`，不要同时使用两者——当它们在同一个生成的 ClusterRole 上发生重叠时，聚合规则优先，直接规则会被丢弃。
- `auth.cpaas.io/roletemplate.level`（角色被*授予*的范围—— `platform` / `cluster` / `project` / `namespace`）与 `spec.aggregationRules[].scope`（每个聚合规则要写入的生成 ClusterRole）是不同的概念。控制器会强制执行以下约束：**`platform`** 级和 **`cluster`** 级角色只能使用 **`scope: cluster`**；`project` 级和 `namespace` 级角色可以使用五种作用域中的任意一种（包括 `cluster`）。违反此规则会使 RoleTemplate 进入 `Failed` 状态。

用于 **matchLabels** 的标签

对于独立自定义角色，请自行创建聚合标签——任何 `rbac.cpaas.io/aggregate-to-<name>` 字符串都可以（5.2）。将其与一个作用域标签配对，该标签控制规则的适用范围：

- `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"`

若要向内置系统角色添加权限（高级操作——这会将权限授予所有已持有该角色的用户），请为您的 ClusterRole 添加 `rbac.cpaas.io/aggregate-to-<built-in-role>` 标签，例如 `aggregate-to-namespace-developer` 或 `aggregate-to-platform-admin`。

多集群分发（5.2 第 2 步）：将 `rbac.cpaas.io/sync.scope: "platform"` 添加到您手动创建的 ClusterRole，以便将其从 global 集群复制到所有集群。它对 RoleTemplate 或 RoleTemplate 生成的 ClusterRole 没有影响。

5.5 验证配置是否实际生效

检查 **RoleTemplate** 状态

RoleTemplate 会报告其生成的内容以及是否已同步——请从这里开始：

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

`status.phase` 应为 `Ready`；`status.clusterRoles` 列出其生成的 ClusterRole；`status.conditions` 在发生故障时包含原因/消息。如果阶段不是 `Ready`、`status.clusterRoles` 为空，或其列出的 ClusterRole 的 `rules`（`kubectl get clusterrole <name> -o yaml`）为空，则该角色未授予您预期的权限。常见原因如下，按大致发生频率排序：

- RoleTemplate 缺少 `rbac.cpaas.io/version: "v2"`，或已被自动冻结——请参阅第 5 节中关于必需标签的注意事项。这是 `aggregationRules` 看似不起作用的最常见原因。
- `functionResourceRef` 名称拼写错误——控制器会静默跳过未知引用，并生成一个没有错误但内容为空的 ClusterRole。
- 您的 `aggregationRules` 标签未匹配任何 ClusterRole，或者源 ClusterRole 尚未同步到此集群。

确认用户确实获得权限

生成 ClusterRole 不是最终目标——最终目标是让用户获得访问权限。通过 UserBinding 授予角色后（请参阅管理角色），使用模拟用户检查确认实际结果：

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

在写入 `aggregationRules` 前检查聚合标签是否可用

在 `clusterRoleSelectors` 中引用 `rbac.cpaas.io/aggregate-to-*` 标签之前，请确认它确实能解析为 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}'
```

- 无输出 → 没有 ClusterRole 携带此标签。如果您使用的是自己的自定义标签（5.2），请仔细检查该标签是否确实已应用于您的 ClusterRole，以及它是否已同步到此集群。如果您使用的是系统标签（5.4），并希望某个产品模块已发布该标签，则它很可能尚未发布——请参阅第 3 节。
- 输出显示 `auth.cpaas.io/roletemplate` 的 `legacy-*` 值 → 此 ClusterRole 来自第 3 节所述的基于 FunctionResource 的兼容性桥接，而不是来自已实际迁移的模块。权限仍来源于 FunctionResource，并且预计此桥接将在 v4.5 中移除。

5.6 更新和删除

- 更新：编辑 RoleTemplate 的 `spec`。只有在 `auth.cpaas.io/roletemplate.frozen` 为 `"false"` 时，更改才会生效（请参阅第 5 节中的注意事项）。
- 删除：删除 RoleTemplate。控制器会在所有集群中垃圾回收其生成的 ClusterRole——您无需手动清理这些资源。（通过方法 B 手动创建的 ClusterRole 由您负责删除。）

6. 复制并精简（示例）

两个完整示例：精简内置模板（方法 A，6.1），以及使用您自己的 ClusterRole 构建多范围角色（方法 B，6.2）。对于 FunctionResource 角色，复制内置系统模板并精简，是避免遗漏模块权限的最快方法。

6.1 精简系统 RoleTemplate YAML（方法 A）

步骤：

- 保留必需的 FunctionResource。
- 将动词缩减为只读操作（get/list/watch）。
- 移除未使用的模块。

角色示例：没有 Secret 权限的开发人员审计员

> **NOTE**
>
> - 在自定义角色中，不要配置 Namespace Resource Management（FunctionResource：`acp-namespace-resource-manage`）。
> - 移除 User Secret Dictionary（FunctionResource：`acp-user-secret`）。
> - 将所有动词设置为 get/list/watch。

YAML 示例：

```
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 构建多范围审计员角色（方法 B）

这是 6.1 的聚合方法版本：一个命名空间级只读角色，涵盖业务命名空间和项目命名空间，并使用您自己的 ClusterRole 和标签构建，遵循 5.2 中的模式。

ClusterRole — 每个要涵盖的范围对应一个 ClusterRole，所有 ClusterRole 使用相同的自定义标签，并分别具有自己的范围标签：

```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 — 每个范围配置一个 `aggregationRules` 条目，并重复使用相同的自定义标签：

```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: []
```

如果需要，也可以用同样的方式添加 `system-ns`。另外请注意，如果 ClusterRole 仅在 global 集群中创建，但角色也必须在工作负载集群中生效，则需在每个 ClusterRole 上设置 `rbac.cpaas.io/sync.scope: "platform"`（5.2 第 2 步）。

IDP

介绍

介绍

Overview

Supported Integration Methods

操作指南

LDAP 管理

开始之前

发现目录设置

选择经过测试的映射

添加 LDAP

验证集成

同步与生命周期

使用 YAML 配置 LDAP

按症状排查

站点诊断工具

OIDC 管理

开始之前

使用 Form 添加 OIDC

验证集成

使用 YAML 添加 OIDC

登录、更新和删除行为

注销行为

按阶段排查

故障排除

删除用户

问题描述

解决方案

介绍

目录

Overview

Supported Integration Methods

LDAP Integration

OIDC Integration

Overview

平台集成了 Dex 身份认证服务，允许您通过 IDP 配置使用 Dex 预先实现的连接器进行平台账号认证。更多信息请参阅 Dex 官方文档↗。

Supported Integration Methods

LDAP Integration

如果您的企业使用 **LDAP**（轻量级目录访问协议）进行用户管理，您可以在平台上配置 LDAP，以连接企业的 LDAP 服务器。

LDAP 集成优势：

- 实现平台与 LDAP 服务器之间的通信
- 允许企业用户使用 LDAP 凭据登录
- 自动同步企业用户账号至平台

OIDC Integration

平台支持通过 OpenID Connect（OIDC）协议与 IDP 服务集成，实现第三方用户认证。

OIDC 集成优势：

- 允许用户使用第三方账号登录
- 支持企业 IDP 服务
- 通过 OIDC 协议提供安全认证

> **NOTE**
>
> 对于使用上述未提及的其他连接器进行认证的需求，请联系技术支持。

操作指南

LDAP 管理

开始之前
发现目录设置
选择经过测试的映射
添加 LDAP
验证集成
同步与生命周期
使用 YAML 配置 LDAP
按症状排查
站点诊断工具

OIDC 管理

开始之前
使用 Form 添加 OIDC
验证集成
使用 YAML 添加 OIDC
登录、更新和删除行为
注销行为
按阶段排查

LDAP 管理

ACP 可以使用 LDAP 目录对用户进行身份验证，并且可选地同步目录组。控制台流程支持常见的 OpenLDAP 和 Microsoft Active Directory 布局。

只有在以下所有检查都通过后，集成才算完成：

- ACP 创建了 LDAP IDP。
- 预期的用户以及（在启用时）组已同步。
- 代表性的目录用户可以登录，并获得预期的 ACP 角色。

本指南从最短的控制台流程开始。如果目录管理员未提供 Base DN、Filter 或属性映射，请先使用 发现目录设置。

目录

开始之前

发现目录设置

1. 从正确的网络运行检查
2. 查找目录根
3. 选择 User 和 Group Base DN
4. 检查代表性用户
5. 选择身份和登录属性
6. 逐步构建 User Filter

LDAP Filter 语法速查

7. 确定组成员模型

选择经过测试的映射
添加 LDAP
控制台字段参考
验证集成
同步与生命周期
首次登录同步
手动同步
自动同步
更新 Connector
协调上游已删除的用户
删除 Connector
使用 YAML 配置 LDAP
严格验证的 OpenLDAP 模板
Active Directory 模板
完整 LDAP YAML 字段参考
按症状排查
站点诊断工具
外部：Linux 和 macOS LDAP 工具
外部：使用 `ldapsearch` 查询 Active Directory
高级：ACP 组件日志

开始之前

在配置 ACP 之前，请准备以下值：

- 从 ACP 集群可访问的 LDAP 服务器主机名和端口。
- 具有目标用户和组只读权限的 Bind DN 和密码。
- User Base DN 和 user Filter。
- 稳定、唯一且非空的用户身份属性。
- 一个或多个登录属性。

- 可选的 group Base DN、group Filter、member 属性、user matching 属性和 group name 属性。
- 一个用于最终登录验证的代表性用户账户。

工程师笔记本电脑上的查询成功，并不证明 ACP 可以连接。ACP Connector 创建和同步才是集群侧的检查。如果目录管理员已经提供并验证了所有必需值，请继续执行 添加 LDAP。

发现目录设置

当 Base DN、Filter 或属性映射未知时，请使用此流程。这里引用的命令行工具是外部集成前诊断工具，不是 ACP 功能。

1. 从正确的网络运行检查

尽可能从诊断主机或 Pod 执行目录查询，其 DNS、路由、防火墙和证书信任应与 ACP 集群等效。笔记本电脑查询可作为有用的初步证据，但 ACP Connector 创建和同步仍然是权威的集群侧检查。

2. 查找目录根

查询目录 Root DSE 并记录命名根：

- OpenLDAP 和兼容目录通常会发布 `namingContexts`。
- Active Directory 会发布 `defaultNamingContext`。

请使用最后的 **On-Site Diagnostic Tools** 部分中的 Root DSE 示例。返回值（例如 `dc=example,dc=com`）是定位用户和组分支的起点；它不会自动成为最佳的 User Base DN。

3. 选择 User 和 Group Base DN

将 Base DN 视为 ACP 搜索的边界。例如：

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

- 如果所有目标用户都位于 `ou=People,dc=example,dc=com` 下，则将其用作 User Base DN。
- 如果目标用户分布在 `ou=People` 之外的多个分支，请使用它们的最低公共父级。
- 当组存储在 `ou=Groups` 下时，请使用单独的 Group Base DN。

在运行子树查询之前，请先使用外部 base-scope 查询确认绑定账户可以读取候选 Base DN。ACP 表单会在已配置的 Base DN 下使用 subtree 搜索。它不会暴露 Search Scope 字段。YAML 可以使用 `scope: sub` 或 `scope: one`；在当前 Connector 运行时中，省略 scope 时默认为 `sub`。

4. 检查代表性用户

在选择映射之前，至少检查三个条目：

- 一个预期可以登录的普通用户。
- 另一个目标 OU 或目录分支中的用户。
- 一个边界情况，例如已禁用用户、没有组的用户，或者缺少可选属性的用户。

对于 OpenLDAP，请请求 DN、`objectClass`、`uid`、`cn` 和 `mail`。对于 Active Directory，请请求 DN、`objectClass`、`sAMAccountName`、`displayName`、`mail`、`userAccountControl` 和 `memberOf`。在缩小 Filter 之前，确认目标用户群体位于候选 Base DN 之下。

5. 选择身份和登录属性

目的	**OpenLDAP** 候选项	**Active Directory** 候选项	选择规则
ACP 用户身份	`uid`	`sAMAccountName`	对每个目标用户都存在，且唯一、稳定、非空
登录字段	`uid`，`mail`	`sAMAccountName`	用户已知，并且只返回一个条目
仅用于显示的信息	`cn`	`displayName`	可能变化，不应作为默认身份
组成员值	`member`，`memberUid`	`member`	必须与示例组条目中存储的值匹配
组名	`cn`	`cn`	对管理员而言可读且稳定

不要仅因为 `cn`、`displayName` 或用户的 DN 易于阅读，就将其用作默认 ACP 身份。这些值可能重复，并且在用户重命名或移动时会发生变化。在创建 Connector 之前，请在目录中解决缺失或重复的身份值。

6. 逐步构建 User Filter

按以下顺序构建并重新测试 Filter：

1. 使用 `objectClass` 选择用户条目。
2. 使用 `(attribute=*)` 约束 ACP 身份和登录属性必须存在。
3. 如果客户的目录策略要求，排除计算机、已禁用账户、服务账户或系统账户。
4. 只有在宽泛结果与预期用户群体匹配后，才添加业务限制。

每添加一个条件后，都重新运行外部目录查询，并将计数和代表性条目与预期用户群体进行比较。使用这些 Filter 查找会因为缺少建议的身份而被跳过的条目：

```
# 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)))
```

最后一个表达式是可复制的用于排除已禁用账户的 Active Directory LDAP Filter。仅当该排除与客户的目录策略一致时才使用。

LDAP Filter 语法速查

请使用能返回目标条目的最小表达式构建 Filter。以下运算符已在本指南使用的测试目录上进行过验证。

需求	语法	示例
精确值	`(attribute=value)`	`(uid=alice)`
属性存在	`(attribute=*)`	`(mail=*)`
所有条件都匹配	`(&(condition1)(condition2))`	`(&(objectClass=inetOr(uid=*))`
任一条件匹配	`(|(condition1)(condition2))`	`(|(uid=alice)(mail=alice@test.cc`
排除某个条件	`(!(condition))`	`(!(objectClass=com`
排除已禁用的 Active Directory 账户	`(!(userAccountControl:1.2.840.113556.1.4.803:=2))`	仅在上方所示的 Active user Filter 中使用

不要在已配置的 user Filter 中放置 `<username>` 占位符。同步会按原样运行已配置的 Filter。在登录期间，ACP 会将该 Filter 与每个已配置 **Login Field** 属性的相等性查找组合起来。

7. 确定组成员模型

检查一个已知组，并将其存储的成员值与代表性用户进行比较：

- 当 `member` 包含完整用户 DN 时，使用 **Group Attr** `member` 和 **User Attr** `DN`。
- 当 `memberUid` 包含简短用户 ID 时，使用 **Group Attr** `memberUid` 和 **User Attr** `uid`。

ACP 会按 `(<Group Attr>=<User Attr value>)` 为每个用户评估组成员关系。使用外部目录查询复现该查找，例如 `(member=uid=alice,ou=People,dc=example,dc=com)` 或 `(memberUid=alice)`。在验证此映射时，不要只依赖用户的 `memberOf` 值。

选择经过测试的映射

以下映射是针对所列布局验证过的起点。Active Directory Filter 会排除计算机对象和已禁用账户。仅当客户的目录策略要求不同的用户群体时，才修改该 Filter，并在修改后重复外部计数、ACP 同步和登录检查。

目录布局	User Filter	User Name Attr
OpenLDAP `groupOfNames`	`(&(objectClass=inetOrgPerson)(uid=*))`	`uid`
OpenLDAP `posixGroup`	`(&(objectClass=inetOrgPerson)(uid=*))`	`uid`
Active Directory	`(&(objectClass=user)(!(objectClass=computer))(sAMAccountName=*)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))`	`sAMAccountName`

当 Active Directory group synchronization 使用 **Group Attr** `member` 和 **User Attr** `DN` 时，请使用 `(&(objectClass=group)(member=*))` 作为 Group Filter。ACP 在创建期间会针对采样组条目验证 `member`。更宽泛的 `(objectClass=group)` Filter 可能会采样到不返回 `member` 属性的空组，从而拒绝一个原本有效的映射。

添加 **LDAP**

1. 转到 **Users > IDP**。
2. 单击 **Add LDAP**。
3. 填写 **Basic Info**：**Name**、**Display Name** 和可选的 **Description**。
4. 填写 **LDAP Server Setting**：**Server Address**、**Admin Account** 和 **Admin Password**。
5. 填写 **Search Setting**：**Filter** 和 **Base DN**。
6. 如果需要组同步，请启用 **Group Search Setting**，并输入 **Filter**、**Base DN**、**Group Attr** 和 **User Attr**。
7. 填写 **Field Mapping**：**User Name Attr**、可选的 **Group Name Attr**、**Login Field** 和 **Username Tip In Login Box**。**User Name Attr** 会成为同步后的 ACP user 所使用的稳定身份值；它不只是一个显示标签。**Login Field** 决定 ACP 如何搜索登录账户，并支持以逗号分隔的多个备选项。
8. 可选地打开 **Advanced Settings**，启用 **Auto Sync**，并在 **Sync Rules** 中输入五字段 cron 表达式。ACP 会以 UTC 评估该规则。
9. 单击 **Add**。

> **NOTE**
>
> 在创建期间，ACP 会验证服务器连通性、bind、Base DN、Filter 和已映射属性。当前表单在创建时不会对普通目录用户进行身份验证，因为它不收集测试用户凭证。提交的 bind 密码会在创建后转移到受管 Secret 中。

控制台字段参考

使用此表将通过 `ldapsearch` 发现的值转换为当前 ACP 表单，以及后文所述的 YAML 字段（如有需要）。

ACP console field	YAML field	如何选择该值
Name	`id` 和 `metadata.name`	两个字段都使用相同的、符

ACP console field	YAML field	如何选择该值
		合 DNS 规范的值。
Display Name	`name`	ACP IDP 选择器中显示的名称。
Description	`metadata.annotations["cpaas.io/description"]`	可选的面向 operator 的描述。
Server Address	`spec.config.host`	从 ACP 可访问的目录主机名或 IP 和端口。
Admin Account	`spec.config.bindDN`	具有目标条目读取权限的 Bind 账户 DN。
Admin Password	`spec.config.bindPW`	初始创建时输入；ACP 会将其存储在受管 Secret 中。
User **Filter**	`spec.config.userSearch.filter`	仅返回目标用户群体的 Filter。
User **Base DN**	`spec.config.userSearch.baseDN`	包含所有目标用户的最低目录分支。
Group **Filter**	`spec.config.groupSearch.filter`	目标组的 Filter；如果存在空组，请要求成员属性存在。

ACP console field	**YAML field**	如何选择该值
Group **Base DN**	`spec.config.groupSearch.baseDN`	包含目标组的最低目录分支。
Group Attr	`spec.config.groupSearch.groupAttr`	组上存储成员值的属性，例如 `member` 或 `memberUid`。
User Attr	`spec.config.groupSearch.userAttr`	可与 **Group Attr** 比较的用户值，例如 `DN` 或 `uid`。
User Name Attr	`spec.config.userSearch.nameAttr`	稳定、唯一且非空的身份值，用作 v2 ACP username 和主标识符。
Group Name Attr	`spec.config.groupSearch.nameAttr`	可读的组名，通常是 `cn`。
Login Field	`spec.config.userSearch.username`	一个或多个以逗号分隔的登录属性。只配置你已验证过的、在过滤后的条目上存在的属性。
Username Tip In Login Box	`spec.config.usernamePrompt`	LDAP 登录页面上显示的提示文本。
Auto Sync	`metadata.labels["cpaas.io/ldap.autoSync"]`	仅在手动同步正确后启用。

ACP console field	YAML field	如何选择该值
Sync Rules	`metadata.annotations["cpaas.io/ldap.autoSyncRule"]`	以 UTC 评估的五字段 cron 表达式。

表单不暴露 `scope`、TLS 标志、CA 数据或可选的 profile 映射。仅在需要时通过 YAML 配置这些字段。

验证集成

1. 打开新的 LDAP IDP，并选择 **Actions > Sync user**。
2. 确认同步结果，并将同步后的用户和组数量与预期目录群体进行比较。
3. 打开一个已同步用户，验证其来源、身份值和状态是否正确。
4. 使用 Manage User Roles 或 Manage User Group Roles 分配一个最小角色。
5. 使用一个不带 ACP 登录状态的单独浏览器会话，并使用代表性目录账户登录。
6. 确认身份验证成功，并且已分配的 ACP 权限可用。

> **WARNING**
>
> 总体同步可能报告成功，但缺少 **User Name Attr** 的条目会被跳过。没有匹配目录组的用户本身并不意味着该用户同步失败。请比较计数并检查代表性条目，而不要只依赖总体结果。

同步与生命周期

首次登录同步

目录用户可以在完整手动同步之前先登录。身份验证成功后，ACP 会同步该单个 LDAP user。请使用上面的完整验收路径来验证预期清单、组映射和授权，而不要将首次登录视为完整的集成验收。

手动同步

打开 LDAP IDP 并选择 **Actions > Sync user**，即可按需同步和协调目录用户群体。请查看结果、计数、跳过条目消息，以及代表性用户和组。

新同步的 LDAP 用户是 active、valid 的，且默认 validity period 为 **Permanent**，除非管理员之后修改其有效期。

在同步开始之前，ACP 会警告：如果一个 IDP user 与现有本地用户同名，可能会覆盖或重新关联该用户，同时保留现有角色分配。在第一次完整同步之前，请检查建议的 **User Name Attr** 是否与当前 ACP 用户冲突。不要使用内置本地管理员账户作为集成测试身份。

自动同步

当需要定时协调时，请在 **Advanced Settings** 下启用 **Auto Sync**，并提供 **Sync Rules**。自动同步会使用已配置的 Base DN、Filter 和映射；它不会对目录用户密码进行身份验证。ACP 会以 UTC 评估五字段 cron 表达式。对于验收测试，`*/1 * * * *` 每分钟运行一次；在观察到计划运行后，请替换该临时规则。

更新 **Connector**

当服务器、搜索边界、Filter、映射或自动同步设置发生变化时，请更新 LDAP IDP。更新后运行一次手动同步，以确认新配置并协调现有源用户和组。

协调上游已删除的用户

在手动或自动协调之后，从上游目录中删除的用户会在 ACP 中变为无效，并且无法再通过该源进行身份验证。

删除 **Connector**

删除 Connector 而不进行清理会保留该源中的用户和组，并将用户标记为无效。如果在删除时选择 **Clean up IDP users and User Groups**，ACP 会删除由该 Connector 生成的用户和组。

使用 YAML 配置 LDAP

以下 v2 `Connector` 模板仅将 `bindPW` 作为初始创建输入。请通过 ACP 控制台的 YAML 选项卡提交它们。ACP 会将提交的密码移至受管 Secret，之后存储的 YAML 会包含受管 `clientSecretRef`，而不是 `bindPW`。不要使用带有内联 `bindPW` 的 `kubectl apply`：Kubernetes last-applied 注解可能会在 Connector 元数据中保留所提交的密码。在使用前，请替换所有示例值。

> **模板中有意显示的字段**
>
> 创建 Connector 时，ACP 会添加 `metadata.labels["cpaas.io/idp.version"]: v2`。对于没有 `rootCAData` 的普通 LDAP 连接，ACP 还会持久化 `insecureNoSSL: true`、`insecureSkipVerify: true` 和 `startTLS: false`。模板显式展示这些值，以便所提交的 YAML 描述实际生效的连接模式。
>
> ACP 可以从 Filter 中的第一个 `objectClass` 推断出某些用户和组属性。对于客户交付，不要依赖这种推断：它无法确定目录真实的身份和成员模型。在检查代表性条目后，请显式设置 `idAttr`、`nameAttr`、`username`、`groupAttr`、`userAttr` 以及组的 `nameAttr`。模板还显式写入了 `scope: sub`，尽管 `sub` 是运行时默认值。`clientSecretRef` 不是这些模板中的输入字段，因为 ACP 只会在将 `bindPW` 移至受管 Secret 后写入它。

严格验证的 OpenLDAP 模板

> **636 端口上的证书验证限制**
>
> 在本指南验证过的 ACP 环境（`auth-controller2` v4.3.6）中，`host` 以 `:636` 结尾的 Connector 会以 `insecureSkipVerify: true` 方式存储，即使提交值为 `false` 且 `rootCAData` 包含正确的 CA 也是如此。标准端口 LDAPS 流量是加密的，但 ACP 不会验证 LDAP 服务器证书。
>
> 下面的模板适用于在非 636 端口上提供 LDAPS 的目录。该配置已在非 636 端口上使用 `insecureSkipVerify: false` 和 `rootCAData` 进行验证。如果需要证书验证，请与目录和网络管理员协调端口，然后完成 Connector 创建、同步和真实登录后再推广。

```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 模板

```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
```

完整 **LDAP YAML** 字段参考

以下骨架包含了对 v2 LDAP Connector 有用的 ACP 集成字段。可选的 profile 映射已注释掉，以便示例在目录不提供这些属性时仍然有效。请删除未使用的可选字段，而不要为过滤后的用户中不存在的属性命名。

```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
```

顶层和 metadata 字段：

字段	是否必需	产品行为
`apiVersion`	是	使用 `dex.coreos.com`
`kind`	是	使用 `Connector`
`id`	是	Connector 标识。与 `metadata.nam` 同。
`name`	是	ACP 登录选择器中的名称。
`type`	是	使用 `ldap`。
`metadata.name`	是	Kubernetes 资源名 ACP Connector 查键。
`metadata.namespace`	是	使用 `cpaas-syst`
`metadata.annotations["cpaas.io/description"]`	否	ACP 显示的描述。
`metadata.annotations["cpaas.io/idp.validation"]`	仅临时故障排查时使用	字符串值 `"false` 在本次创建或更改的更新中跳过目录性验证；格式验证运行。不要将其设 `"true"`。
`metadata.labels["cpaas.io/idp.version"]`	是	对本文档所述映射 `v2`。
`metadata.labels["cpaas.io/ldap.autoSync"]`	否	字符串值 `"true"` `"false"`；启用步。

字段	是否必需	产品行为
`metadata.annotations["cpaas.io/ldap.autoSyncRule"]`	启用 auto sync 时必需	以 UTC 评估的五 cron 表达式。

连接和传输字段：

字段	是否必需	产品行为
`host`	是	从 ACP 集群可达的目录主机和端口。
`bindDN`	是	用于读取用户和组的 Bind 账户 DN。
`bindPW`	初始创建时需要	ACP YAML 页面接受的明文输入。ACP 会移除它并将密码存储到受管 Secret 中。
`clientSecretRef`	存储输出	ACP 写入的受管 Secret 引用。不要将其与 `bindPW` 一起提交，也不要手动编辑。
`insecureNoSSL`	是	`true` 选择普通 LDAP；`false` 根据 `startTLS` 选择 TLS。
`startTLS`	是	`false` 且 `insecureNoSSL: false` 时选择 LDAPS。`true` 时选择通过 LDAP 的 StartTLS。
`insecureSkipVerify`	TLS 连接时	`false` 请求服务器证书验证，但受下面 636 端口限制影响。
`rootCAData`	私有 CA 严格 TLS	Connector 信任池使用的 Base64 编码 PEM CA 证书。
`usernamePrompt`	否	登录页面上 LDAP username 字段上方显示的文本。

`userSearch` 字段：

字段	是否必需	产品行为
`baseDN`	是	用户条目的搜索边界。

字段	是否必需	产品行为
`filter`	是	用于选择已同步用户群体的 LDAP Filter。
`scope`	否	`sub` 会搜索完整子树，是运行时默认值；`one` 仅搜索 `baseDN` 下一级。
`nameAttr`	v2 必需	用作 ACP username 和同步期间主标识符的稳定唯一值。
`idAttr`	是	验证和兼容性身份字段。对于 v2，请将其设置为与 `nameAttr` 相同的稳定属性。
`username`	是	在登录时用于查找用户的一个或多个以逗号分隔的属性。
`emailAttr`	否	将目录值复制到 ACP 用户的 `mail` 字段。仅在过滤后的用户提供该属性时配置。
`preferredUsernameAttr`	否	在 LDAP 登录期间提供 preferred-username Claim；它不会替代 v2 从 `nameAttr` 同步得到的身份。
`phoneAttr`	否	将目录值复制到 ACP 用户的 mobile 字段。仅在过滤后的用户提供该属性时配置。
`emailSuffix`	否	在 LDAP 登录期间，将身份 email 构造成 `<nameAttr>@<suffix>`；输入 suffix 时不要包含 `@`。

Warning

`emailSuffix` 与 v2 同步

在已验证的 v2 行为中，`emailSuffix: test.com` 在首次登录身份中生成了 `admin@test.com`，而手动同步仍将不带后缀的 `nameAttr` 作为主标识符存储。除非已针对客户的同步和冲突要求测试过这种差异，否则请不要设置 `emailSuffix`。

`groupSearch` 字段：

字段	启用组同步时是否必需	产品行为
`baseDN`	是	组条目的搜索边界。
`filter`	是	用于选择组的 LDAP Filter。要求成员属性存在可以避免对空组进行验证。
`scope`	否	`sub` 是运行时默认值；`one` 会搜索 `baseDN` 下一级。
`groupAttr`	是	组条目中包含成员值的属性。
`userAttr`	是	与 `groupAttr` 比较的用户条目中的属性或 `DN` 值。
`nameAttr`	是	用作 ACP 组显示名称的属性，通常是 `cn`。

当 ACP 不需要组同步时，请完全省略 `groupSearch`。

> **临时验证绕过**
>
> 仅在目录暂时不可达且必须先分阶段创建 Connector 时，才使用 `cpaas.io/idp.validation: "false"`。ACP 仍会检查配置结构，但会跳过网络、TLS、bind、Base DN、Filter 和映射属性检查。因此，即使同步和登录无法工作，Connector 也可能被创建出来。
>
> admission webhook 会从已存储的 Connector 中删除此注解，并将跳过标记复制到受管凭证 Secret。之后若进行不带该注解的 spec 变更更新，则会执行正常的 Connector 验证。若要在未来的凭证变更中恢复验证，请通过不带该注解的 Connector 更新凭证；不要直接编辑受管 Secret。不要使用 `cpaas.io/idp.validation: "true"`；正常验证时请省略该注解。
>
> 在推广之前，请完成一次正常验证的创建或真实修正更新，然后运行手动同步和代表性登录。仅做 metadata 更新，不能证明正常验证已运行。

将传输标志作为一个整体来选择：

传输方式	常见端口	`insecureNoSSL`	`startTLS`	证书验证
普通 LDAP	389	`true`	`false`	不适用；仅在网络策 允许且处于受信任网 时使用
标准端口上的 LDAPS	636	`false`	`false`	已验证的 ACP 实现会 存储 `insecureSkipVeri` `true`；流量已加密 但不会验证 LDAP 服 器证书
具有严格证书验证的 LDAPS	已配置的非 636 LDAPS 端口	`false`	`false`	保持 `insecureSkipVeri` `false`，并通过 `rootCAData` 提供受 信任 CA；该配置已在 非 636 端口上验证

上面的 Active Directory 模板使用的是 ACP 验证过的普通 LDAP 传输。普通 LDAP 不会在网络上传输保护 bind 密码。仅当客户的网络和安全策略允许时才使用它。当需要 TLS 时，首先要确定策略是否还要求证书验证：在已验证实现中，标准端口 636 提供加密但不验证服务器证书，而严格验证则需要非 636 的 LDAPS 端口和 `rootCAData`。在推广前，请完成一次 ACP LDAPS 创建、同步和登录测试。

若要在非 636 的 LDAPS 端口上进行严格证书验证，请将 PEM CA 证书编码为不带换行符的 Base64，并将输出用作 `rootCAData`：

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

对于 YAML 中的定时同步，请向 Connector 添加受支持的 metadata：

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

示例规则仅适用于观察验收测试运行。在验证后，请将其替换为所需的 UTC 调度。顶层 `id` 必须等于 `metadata.name`；当前的定时同步会按该标识查找 Connector。

按症状排查

症状	产品阶段	可能原因	具体检查
ACP 无法创建 Connector，因为无法连接。	Connector 创建	集群 DNS、路由、防火墙、端口、TLS 模式或证书信任配置不正确。	从具有与 ACP 等效可达性的主机或 Pod 运行下面的外部 TCP 或 TLS 检查，然后检查 `auth-controller2` 日志。
Bind 成功，但 Base DN 或 Filter 验证失败。	Connector 创建和搜索验证	该 Base DN 对 Bind 账户不存在，Filter 无效，或者 Filter 没有返回可用条目。	对 Base DN 运行外部 base-scope 查询，然后运行精确的 subtree Filter，并检查 `auth-controller2` 日志。
ACP 拒绝 Active Directory **Group Attr** `member`，同时还建议使用 `member`。	Connector 创建和组属性验证	Group Filter 采样到了未返回 `member` 属性的空组。	使用 `(&(objectClass=group)(member=*))`，确认外部查询返回的是目标组，然后重试创建。
同步报告成功，但返回的用户少于预期。	手动或定时同步	Filter 排除了用户、用户不在 Base DN 范围内，或者缺少必需的 **User Name Attr** 值。	将外部子树结果与预期计数进行比较，运行上方的缺失属性 Filter，并检查 `apollo` 日志中的跳过条目消息。
Connector 创建成功，但目录用户无法登录。	登录	**Login Field** 没有找到恰好一个条目、账户被排除或禁用，或者代表性凭证无效。	使用每个已配置的登录属性查询代表性的登录值，确认结果恰好一个，在单独浏览器会话中重试，并检查 `apollo` 日志。
用户已同步，但组未同步。	组同步	**Group Attr** 和 **User Attr** 无法表示组中存储的值，或者 Group	检查一个已知组，并通过外部子树查询复现 `(<Group Attr>=<User Attr`

症状	产品阶段	可能原因	具体检查
		Base DN / Filter 错误。	`value>)`，然后检查 `apollo` 的组查找消息。
身份发生冲突或意外变化。	用户同步和协调	**User Name Attr** 重复、为空、可变，或者基于在重命名或移动后发生变化的 DN。	在整个 User Base DN 范围内查询建议的身份属性，检查缺失和重复值，并选择一个稳定唯一的属性。
自动同步未运行。	定时同步	Auto Sync metadata 缺失或无效，或者顶层 `id` 与 `metadata.name` 不一致。	检查存储的 Connector metadata 和身份，然后检查 `apollo` 日志中的定时同步消息。

站点诊断工具

本节中的工具属于外部集成前诊断或高级 ACP 证据收集。它们不是内置的 ACP 功能。

外部：**Linux** 和 **macOS LDAP** 工具

LDAP 命令需要已安装的 OpenLDAP client，例如 `ldapsearch` 和 `ldapwhoami`。它们会使用 `-W` 提示输入 bind 密码。该选项只能防止密码出现在命令行中；它不会加密网络传输。

下面的私有 CA 环境变量已在 Linux OpenLDAP client 上验证。内置的 macOS `/usr/bin/ldapsearch` 和 `/usr/bin/ldapwhoami` 使用的是 macOS 系统信任存储；请使用 Linux 诊断主机或 Pod，或者在使用这些内置 client 之前先在 macOS Keychain 中配置 CA。

对于每个 `ldapwhoami` 和 `ldapsearch` 命令，请使用与被调查 Connector 相同的传输组合：

Connector 传输	**Connector** 设置	**OpenLDAP client** 传输	**Bind** 凭证保护
普通 LDAP	`insecureNoSSL: true`，	`-H ldap://ldap.example.com:389`，不带 `-ZZ`	无 TLS 保护；仅在网络策略允许时使用

Connector 传输	Connector 设置	OpenLDAP client 传输	Bind 凭证保护
	`startTLS: false`		
LDAPS	`insecureNoSSL: false`，`startTLS: false`	`-H ldaps://ldap.example.com:636`，不带 `-ZZ`	连接建立时即开始 TLS

下面的查询示例使用普通 LDAP，因此只匹配 `insecureNoSSL: true` 的 Connector。对于 LDAPS，请在运行每个示例之前将 `-H` 值替换为匹配的行。如果 LDAP 证书由私有 CA 签名，诊断 client 也必须信任该 CA。在 Linux OpenLDAP client 上，请按如下方式为每个 `ldapwhoami` 或 `ldapsearch` 命令添加前缀：

```
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
```

如果没有 `LDAPTLS_CACERT` 设置，已验证的 Linux client 会失败并显示 `unable to get local issuer certificate`；使用 CA 文件后，相同的 bind 会成功。

外部命令：仅测试 **TCP** 端口可达性

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

该命令仅检查 TCP 端口是否可达。它不会验证 TLS、bind 凭证或 LDAP 搜索行为。检查 LDAPS 端口可达性时，请改用端口 `636`。

外部命令：验证服务器、传输和 **bind** 账户

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

外部命令：读取目录 **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
```

外部命令：确认候选 **Base DN** 可读

```
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
```

外部命令：复现 **OpenLDAP** 用户子树搜索

```
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
```

外部命令：查找缺少建议身份的 **OpenLDAP** 条目

```
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
```

外部命令：打印建议的 **OpenLDAP** 身份的重复值

```
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
```

在评估其他候选身份属性时，请替换 `uid`、Base DN 和 Filter。没有输出表示返回的条目中未发现重复值。与其他 `ldapsearch` 示例一样，请使用与 Connector 匹配的传输组合。

外部命令：复现 `groupOfNames` 成员查找

```
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
```

外部命令：使用目标 **CA** 验证 **LDAPS** 证书链

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

只有当输出报告验证成功时，证书检查才通过，例如 `Verify return code: 0 (ok)` 或 `Verification: OK`。

外部：使用 `ldapsearch` 查询 Active Directory

同一个 OpenLDAP 命令行 client 无需 Windows 或 RSAT 即可检查 Active Directory。请使用具有读取权限的目录账户。

外部命令：检查目标 **Active Directory** 用户

```
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=*)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))' \
  dn sAMAccountName displayName mail userAccountControl memberOf
```

外部命令：复现 **Active Directory** 组成员查找

```
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
```

高级：**ACP** 组件日志

这些部署名称是当前 ACP 组件名称。运行这些命令需要读取集群日志的权限。

高级 **ACP** 命令：检查 **Connector** 创建验证

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

`auth-controller2` 负责 Connector 创建验证，包括网络连接、TLS、bind、Base DN、Filter 和已映射属性。

高级 **ACP** 命令：检查同步和登录行为

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

`apollo` 负责手动和定时同步、跳过的条目、组查找以及登录行为。

> **NOTE**
>
> 使用外部工具成功，并不能证明 ACP 能够访问或使用该目录。请通过 ACP Connector 创建、预期的同步结果，以及使用目标 ACP 权限进行真实目录登录来完成证据链。

OIDC 管理

ACP 可以将用户重定向到符合标准的 OIDC Identity Provider，并在成功登录后创建或更新其平台身份。

初始集成若要尽可能简短，请使用 **Form** 选项卡。 当返回的用户名 Claim 必须显式选择、Claim 必须重映射、必须查询 UserInfo，或者需要 OIDC 组和额外字段时，请使用 **YAML** 选项卡。参见 使用 YAML 添加 OIDC。

目录

开始之前
确认返回的 Claims
使用 Form 添加 OIDC
控制台字段参考
验证集成
使用 YAML 添加 OIDC
OIDC 字段完整参考
登录、更新和删除行为
注销行为
按阶段排查

开始之前

准备以下值：

- Issuer URL，在 ACP 中标记为 **Server Provider URL**。
- Client ID。
- Client Secret，在 ACP 中标记为 **Client Key**。
- 一个具有电子邮件地址的代表性测试账户，该地址尚未被 ACP 本地用户或其他 IDP 使用。
- 为该测试账户返回的一个稳定的 username Claim。测试过的 YAML 回退使用 `preferred_username`。

向上游 Identity Provider 注册以下精确回调：

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

ACP 会根据平台地址派生回调地址。表单中不包含 Redirect URI 字段。

Server Provider URL 必须是 Issuer 根地址，而不是 authorization endpoint、token endpoint 或登录页。其 discovery document 通常可在以下路径获取：

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

你可以在一个与 ACP 具有等效可达性的诊断主机上检查 discovery：

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

对于使用私有 CA 的上游服务，请使用该 CA 独立验证证书链：

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

该外部查询只是初步证据。ACP Connector 创建才是集群侧的 discovery 和可达性检查。

确认返回的 **Claims**

请上游 Identity 管理员提供一个代表性账户的 ID token 或 UserInfo Claim 名称。Claim 名称区分大小写。ACP Connector 创建会验证 discovery 和客户端可达性，但不能证明真实用户的 token 包含所需的 Claims。

Claim 或设置	需要的条件	**ACP** 用途
`sub`	始终需要	OIDC 协议使用的稳定上游 subject。它必须是字符串。
`name`	未配置 `userNameKey`	ACP 用户名的默认来源。
由 `userNameKey` 选择的 Claim	默认 `name` Claim 缺失或不适用时	ACP 用户名的来源。已验证的回退是 `preferred_username`。
`email` 或由 `claimMapping.email` 选择的 Claim	请求了 `email` scope	ACP 用户身份。它必须是字符串，并且不应与现有 ACP 用户冲突。
`email_verified`	请求了 `email` scope	当前登录流程所需的布尔验证标志。
由 `claimMapping.groups` 选择的 Claim	启用了 OIDC 组处理	用于创建并关联 ACP 组的字符串或组名数组。
由 `claimMapping.phone` 和 `claimMapping.mail` 选择的 Claim	配置了这些可选的 profile 字段	复制到 ACP 用户 mobile 和 mail 字段的字符串值。
`claimExtra` 中列出的 Claim	额外值必须存储到 ACP	当其配置类型匹配时，值会复制到 `spec.extra`。本文档记录了已验证的 `string` 类型。

在发布前完成一次真实的浏览器登录并检查生成的 ACP 用户。该登录是验证 Claim 可用性、映射、冲突行为和组限制的权威检查。

上游 **OIDC** 证书验证限制

在本文档验证的 ACP 环境（`auth-controller2` 和 `apollo` v4.3.6）中，Connector 创建接受了一个 HTTPS Issuer，其自签名 CA 未被普通客户端信任，并且存储的 Connector 包含

`insecureSkipVerify: true`。因此，在该实现中 ACP 不会验证上游 OIDC 服务器证书。HTTPS 仍然会加密连接，但 Connector 创建成功并不能证明该证书链已被信任。

如果客户的安全策略要求验证上游服务器证书，则已验证的实现不满足该要求。请在发布前解决产品版本或安全策略要求。

使用 **Form** 添加 **OIDC**

1. 转到 **Users > IDP**。
2. 单击 **Add OIDC**。
3. 填写 **Basic Info**：**Name**、**Display Name** 和可选的 **Description**。
4. 填写 **Server Setting**：**Server Provider URL**、**Client ID** 和 **Client Key**。
5. 对于此已验证的集成流程，请将 **Logout URL** 留空。
6. 单击 **Add**。

在创建过程中，ACP 会从 Issuer URL 读取 OIDC discovery。无效或不可达的 Issuer 会在此阶段被拒绝。当前的创建检查不能证明 **Client Key** 正确，也不能证明上游 TLS 证书受信任：即使 key 错误，Connector 也可能被创建，随后在浏览器回调时失败；同时，不受信任的自签名证书也可能被接受。因此，身份验证仍需要一次真实登录，而证书合规性必须按上文所述单独评估。

创建完成后，ACP 会将提交的 **Client Key** 迁移到受管理的 Secret。存储的 Connector 引用该 Secret，而不是在其配置中保留 `clientSecret`。

控制台字段参考

ACP 控制台字段	**YAML** 字段	产品行为
Name	`id` 和 `metadata.name`	两个字段使用相同的、符合 DNS 规范的值。
Display Name	`name`	显示在 ACP IDP 选择器上的名称。

ACP 控制台字段	YAML 字段	产品行为
Description	`metadata.annotations["cpaas.io/description"]`	可选的、面向 operator 的描述。
Server Provider URL	`spec.config.issuer`	用于 OIDC discovery 的 Issuer 根地址。
Client ID	`spec.config.clientID`	在上游注册的客户端标识符。
Client Key	`spec.config.clientSecret`	初始创建时的输入；ACP 会将其存储到受管理的 Secret 中。
Logout URL	`logoutUrl`	保持为空。已验证的运行时未将其用作注销后的重定向。

表单不会暴露 `redirectURI`、scopes、UserInfo、Claim mappings、组处理或额外 Claims。只有在确认上游响应后，才通过 YAML 配置这些字段。

验证集成

1. 在没有 ACP 登录状态的情况下启动一个单独的浏览器会话。
2. 打开 ACP 并选择已配置的 OIDC **Display Name**。
3. 在上游 Identity Provider 完成身份验证。
4. 确认浏览器返回到 ACP，而不是 HTTP 500 错误。
5. 在 **Users** 中找到生成的用户，并验证其 IDP 来源、email、username 和 active 状态。
6. 如果由于上游响应不包含默认的 `name` Claim 而导致 username 为空，请使用下面的 YAML 示例重新创建或更新 Connector，然后再次登录。
7. 分配一个最小角色，参见 Manage User Roles。

8. 再次登录并确认已分配的 ACP 权限可用。

> **电子邮件身份冲突**
>
> 在已验证的 ACP 行为中，通过另一个 OIDC Connector 使用相同的映射 email 登录，会将现有 ACP 用户更新并重新关联到较新的 Connector。它不会创建第二个用户，也不会拒绝该冲突。请使用专用的、不会冲突的账户进行验收测试，并在发布前检查现有 ACP 用户。

Connector 创建、上游身份验证、ACP 用户创建以及 ACP 授权是彼此独立的检查点。请分别验证每个检查点。

使用 YAML 添加 OIDC

当上游响应需要显式 Claim 选择或高级映射时，请使用 **Users > IDP > Add OIDC > YAML**。下面的完整示例假设上游响应包含 `sub`、`preferred_username`、`email`、`email_verified`、`groups` 和 `phone_number`，并且测试用户属于 `acp-users`。请移除真实上游响应不满足的可选映射和组限制。

```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
```

在提交前，请替换 Issuer、凭证、ACP 地址、Claim 名称以及允许的组。如果提供方未暴露 UserInfo endpoint，或者所有必需的 Claims 都已包含在 ID token 中，请将 `getUserInfo` 设

为 `false`。如果不需要组处理，请移除 `insecureEnableGroups`、`allowedGroups` 和 `claimMapping.groups`。

不要在 `scopes` 中添加 `openid`；Connector 会自动将其添加到 authorization request 中。如果上游提供方需要一个特定于该提供方的 scope 来获取组 Claims，请仅在 discovery 和提供方配置确认后再添加该 scope。

请通过 ACP 控制台的 YAML 选项卡提交此初始配置。ACP 会将 `clientSecret` 迁移到受管理的 Secret，存储的 Connector 将改为包含 `clientSecretRef`。不要使用带有内联 `clientSecret` 的 `kubectl apply`：Kubernetes 的 last-applied annotation 可能会在 Connector metadata 中保留提交的 secret。

OIDC 字段完整参考

顶层和 metadata 字段：

字段	是否必需	产品行为
`apiVersion`	是	使用 `dex.coreos.com/v1`
`kind`	是	使用 `Connector`。
`id`	是	Connector 标识。保持 `metadata.name` 相同。
`name`	是	显示在 ACP 登录选择上的显示名称。
`type`	是	使用 `oidc`。
`metadata.name`	是	Kubernetes 资源名和 ACP Connector 查询键。
`metadata.namespace`	是	使用 `cpaas-system`
`metadata.annotations["cpaas.io/description"]`	否	ACP 显示的描述。

字段	是否必需	产品行为
`metadata.annotations["cpaas.io/idp.validation"]`	仅用于临时排障	字符串值 `"false"` 在本次创建或变更 sp 的更新中跳过上游连 性验证；格式验证仍 运行。不要将其设为 `"true"`。
`metadata.labels["cpaas.io/idp.version"]`	推荐	使用 `v2`；当前 ACP 创建时也会添加该 label。

连接和登录字段：

字段	是否必需	产品行为
`issuer`	是	用于获取 `/.well-known/openid-configuration` 的 Issuer 根地址。
`clientID`	是	在上游注册的客户端标识符。
`clientSecret`	初始创建	ACP YAML 页面接受的明文输入。ACP 会将其移除并存储到受管理的 Secret 中。
`clientSecretRef`	存储后的输出	ACP 写入的受管理 Secret 引用。不要将其与 `clientSecret` 一并提交，也不要手动编辑它。
`redirectURI`	是	在上游注册的精确 ACP 回调：`https://<ACP_ADDRESS>/dex/callback`。
`scopes`	否	例如 `profile` 和 `email` 之类的附加 scope。ACP 会自动添加 `openid`。
`getUserInfo`	否	设为 `true` 时，请求 UserInfo endpoint，并在映射前合并返回的 Claims。请确认提供方支持该 endpoint。
`userNameKey`	否	作为 ACP 用户名的 Claim，替代默认的 `name`；已验证的回退是 `preferred_username`。

Claim 和组字段：

字段	是否必需	产品行为
`overrideClaimMapping`	强制自定义映射时	当标准 Claim 也存在，但 ACP 必须使用下面选择的 Claim 时，设为 `true`。
`claimMapping.preferred_username`	否	用作首选用户名的 Claim。
`claimMapping.email`	重映射 email 时	用作 ACP 主身份的 Claim。发布前请检查冲突。
`claimMapping.groups`	启用组处理时	包含字符串或组名数组的 Claim。这个嵌套字段是运行时的组映射。
`claimMapping.phone`	否	复制到 ACP 用户 mobile 字段的字符串 Claim。
`claimMapping.mail`	否	复制到 ACP 用户 mail 字段的字符串 Claim。
`claimExtra[].field`	否	复制到 ACP 用户 `spec.extra` map 的 Claim。
`claimExtra[].type`	与 `claimExtra[].field` 一起使用	对于本文档中的已验证流程，请使用 `string`。当 Claim 缺失或类型不同时，请移除该条目。
`insecureEnableGroups`	需要组处理时	设为 `true`，以便读取已映射的组 Claim，并在登录时创建或关联 ACP 组。
`allowedGroups`	否	当该字段非空时，只有至少一个映射组匹配时才允许登录。ACP 仅保留匹配的允许组。

如果 `scopes` 中包含 `email`，则所选 email Claim 必须是字符串，并且合并后的响应中 `email_verified` 必须是 Boolean。任一值缺失都会导致登录失败。

> **临时绕过验证**
>
> 仅当必须在上游服务临时不可达时暂存 Connector，才使用 `cpaas.io/idp.validation: "false"`。ACP 仍会检查配置结构，但会跳过 discovery 和上游连通性检查。因此，即使浏览器登录无法工作，该 Connector 也可能存在。
>
> admission webhook 会从已存储的 Connector 中移除该 annotation，并将跳过标记复制到受管理的 credential Secret 中。后续在没有该 annotation 的 spec 变更型 Connector 更新将执行正常的 Connector 验证。要为未来的 Client Key 变更恢复验证，请通过不带该 annotation 的 Connector 更新凭证；不要直接编辑受管理的 Secret。不要使用 `cpaas.io/idp.validation: "true"`；正常验证时请省略该 annotation。
>
> 在发布前，请完成一次正常验证的创建或真实的纠正性更新，然后执行一次真实的浏览器登录并检查生成的 ACP 用户。仅 metadata 更新不能证明正常验证已运行。

> **ACP 管理且无效的字段**
>
> 存储的 YAML 可能包含 controller 添加的默认值，例如 `issuerAlias: ""`、顶层 `groupsKey: groups` 和 `insecureSkipVerify: true`。
>
> - 通过 `claimMapping.groups` 配置组映射。当前运行时不会将 controller 添加的顶层 `groupsKey` 用作组 Claim 选择器。
> - 不要将 `insecureSkipVerify: false` 作为证书信任控制手段。正如 开始之前 所述，已验证的运行时不会验证上游证书。
> - 除非某个产品特定的集成流程明确要求，否则请不要在已验证流程中使用 `issuerAlias`。

登录、更新和删除行为

- OIDC 用户会在成功的浏览器登录后创建或更新。OIDC 不提供类似 LDAP 的手动或计划性用户同步。
- 在更新用户名映射后，请启动新的浏览器会话并再次登录，以验证更新后的用户。

- 通过 ACP 删除 Connector 而不进行清理，会保留源用户并将其标记为无效。
- 选择 **Clean up IDP users and User Groups** 会删除生成的源用户以及与 Connector 一起管理的 Secret。

请使用 ACP 控制台操作或 ACP product API 进行删除。直接使用 `kubectl` 删除底层 Connector custom resource 会绕过已测试的 ACP 用户失效和清理行为。

注销行为

在已验证的当前流程中，ACP 注销会结束 ACP 会话，并将浏览器返回到 ACP IDP 选择器。再次选择同一个 OIDC 条目会复用仍然有效的上游会话，并在不再次提示凭证的情况下完成登录。因此，ACP 注销并不能证明上游会话已经结束。

经过测试的流程没有将配置的 **Logout URL** 用作浏览器重定向。请将该字段留空，并在客户安全要求需要时单独验证上游会话终止。

按阶段排查

症状	已证实阶段	下一步检查
ACP 在创建期间拒绝 Connector。	discovery 或集群侧可达性未完成。	确认 **Server Provider URL** 是 Issuer 根地址，从与 ACP 等效的网络路径获取其 discovery document，然后重试创建。
Connector 创建成功，但回调返回 HTTP 500，错误为 `unauthorized_client` 或 `Invalid client secret`。	discovery 和上游授权页正常；token endpoint 上的客户端认证失败。	更正 **Client Key**，然后在新的浏览器会话中重复完整登录。仅创建并不能验证该值。
登录成功，但 ACP 用户名为空。	ACP 接受了上游身份并创建了用户。	确认 `preferred_username` 存在，通过 YAML 设置 `userNameKey: preferred_username`，然后再次登录。

症状	已证实阶段	下一步检查
登录失败，并提示缺少 email 或 `email_verified`。	授权和 token 交换已完成，但所需 Claims 缺失或类型错误。	检查 ID token 和 UserInfo 响应，更正 `scopes` 或 `claimMapping.email`，然后重复登录。
登录失败，并提示 `user not a member of allowed groups`。	已读取映射的组 Claim，但没有任何值与 `allowedGroups` 匹配。	确认组值的大小写完全正确，然后更正上游成员关系、`claimMapping.groups` 或 `allowedGroups`。
登录成功，但没有关联任何 ACP 组。	身份验证和用户映射完成，但没有得到可用的组结果。	确认 `insecureEnableGroups: true`，检查映射的组 Claim 类型和值，并验证 `claimMapping.groups`。
配置的额外 Claim 未出现在 ACP 用户中。	登录完成，但 Claim 缺失或与配置的类型不匹配。	检查合并后的 ID token 和 UserInfo Claims；对于本文档，请仅对字符串 Claim 使用 `type: string`。
测试登录后，现有用户的 IDP 来源或映射发生变化。	回调映射到了一个已经被 ACP 用户使用的 email。	停止发布，选择一个不会冲突的测试账户，然后在重试前检查现有 ACP 用户。
ACP 注销后返回 IDP 选择器，但再次选择 OIDC 时无需额外提示即可登录。	ACP 会话已结束，而上游会话仍然有效。	将 ACP 和上游注销视为两个独立的验收检查；不要依赖未经验证的 **Logout URL** 字段。
使用 `kubectl` 删除 Connector custom resource 后，用户仍保持 active。	绕过了 ACP product deletion 流程。	如有需要，先恢复测试 Connector，然后使用 ACP 控制台操作删除，并选择所需的清理选项。

如需查看回调和用户映射证据，请检查当前 ACP 登录组件日志：

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

故障排除

删除用户

问题描述

解决方案

删除用户

目录

问题描述

解决方案

清理已删除的 IDP 用户

清理已删除的本地用户

问题描述

问题：创建或同步新用户时，系统提示用户已存在。应如何处理？

出于安全原因，平台禁止创建名称与之前已删除用户匹配的新用户（包括本地用户和 IDP 用户）。此限制适用于：

- 创建名称与已删除用户匹配的新本地用户
- 同步名称与已删除用户匹配的 IDP 用户

升级到当前版本后，您可能会在以下情况下遇到此问题：

- 创建名称与升级前删除的用户匹配的新用户
- 同步名称与升级前删除的用户匹配的新用户

解决方案

要解决此问题，您需要通过在 global 集群控制平面节点上执行特定脚本来清理已删除的用户信息。

清理已删除的 IDP 用户

在 global 集群的任意控制平面节点上执行以下命令：

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

示例：

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

清理已删除的本地用户

在 global 集群的任意控制平面节点上依次执行以下两个脚本：

1. 清理用户密码：

   ```
   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. 清理用户：

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

用户策略

介绍

Overview

Configure Security Policy

Available Policies

介绍

平台提供全面的用户安全策略，以增强登录安全性并防范恶意攻击。

目录

Overview

Configure Security Policy

Steps

Available Policies

Overview

平台支持以下安全策略：

- 密码安全管理
- 用户账号禁用
- 用户账号锁定
- 用户通知
- 访问控制

Configure Security Policy

Steps

1. 在左侧导航栏，点击 **User Role Management** > **User Security Policy**
2. 点击右上角的 **Update**
3. 根据需要配置安全策略
4. 点击 **Update** 保存更改

> **WARNING**
>
> 策略配置说明：
>
> - 勾选策略前的复选框以启用该策略
> - 取消勾选以禁用该策略
> - 禁用的策略会保留其配置数据
> - 重新启用策略时会恢复之前的设置

Available Policies

Policy	Description
User Authentication Policy	启用基于密码登录的双重认证： - 用户通过指定的通知方式接收验证码 - 支持多种通知服务器（例如企业通讯工具服务器）
Password Security Policy	管理密码要求： 首次登录： - 强制首次登录平台时修改密码 定期更新： - 要求在指定周期（例如 90 天）后修改密码 - 未更新密码前禁止登录
User Disablement Policy	自动禁用不活跃账号： - 在指定时间段内无登录行为后触发

Policy	Description
User Locking Policy	防范暴力破解攻击： 锁定条件： - 在24小时内达到指定失败登录次数后触发 锁定时长： - 账号锁定指定分钟数 - 锁定期满后自动解锁
Notification Policy	管理用户通知： - 用户创建后通过邮件发送初始密码
Access Control	管理用户会话和访问： 会话管理： - 指定时间内无操作自动登出 - 限制最大并发在线用户数 浏览器控制： - 关闭所有产品标签页时结束会话 - 防止同一客户端多次登录 :::note 重要说明： - 访问控制仅影响策略更新后的新登录 - 浏览器标签页恢复可能不会触发会话结束 - 防止重复登录时仅允许客户端最后一次登录 :::

多租户（项目）

介绍

介绍

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

操作指南

创建项目

操作步骤

管理项目配额

什么是 ProjectQuota？

工作原理

管理项目

更新项目基本信

删除项目

管理项目集群

介绍

添加集群

移除集群

管理项目成员

导入成员

移除成员

介绍

目录

Project

Namespaces

Relationship Between Clusters, Projects, and Namespaces

Project

Project 是一个资源隔离单元，支持企业中的多租户使用场景。它将一个或多个集群的资源划分为相互隔离的环境，确保资源和人员的隔离。Project 可以代表企业中的不同子公司、部门或项目团队。通过 Project 管理，可以实现：

- 项目团队之间的资源隔离
- 租户内的配额管理
- 高效的资源分配与控制

Namespaces

Namespace 是 Project 内更小的、相互隔离的资源空间，作为用户实现生产工作负载的工作区。Namespace 的主要特性包括：

- 一个 Project 下可以创建多个 Namespace
- 所有 Namespace 的资源配额总和不能超过 Project 的配额
- 资源配额在 Namespace 级别进行更细粒度的分配
- 容器规格（CPU、内存）在 Namespace 级别受限
- 通过细粒度控制提升资源利用率

Relationship Between Clusters, Projects, and Namespaces

平台的资源层级遵循以下规则：

- 一个 Project 可以使用来自多个集群的资源（CPU、内存、存储），一个集群也可以将资源分配给多个 Project。
- 一个 Project 下可以创建多个 Namespace，且它们的资源配额总和不能超过 Project 的总资源。
- 一个 Namespace 的资源配额必须来自单个集群，且一个 Namespace 只能属于一个 Project。

操作指南

创建项目

操作步骤

管理项目配额

什么是 ProjectQuota？

工作原理

管理项目

更新项目基本信

删除项目

管理项目集群

介绍

添加集群

移除集群

管理项目成员

导入成员

移除成员

创建项目

在您的项目团队开始工作之前，您可以基于平台上现有的集群资源创建一个项目。该项目在资源和人员方面将与其他项目（租户）隔离。创建项目时，您可以根据项目规模和实际业务需求分配资源。项目可以利用平台上多个集群的资源。

> **WARNING**
>
> 创建项目时，平台会在关联的集群中自动创建一个与项目同名的命名空间，用于隔离平台级资源。请勿修改该命名空间或其资源。

目录

操作步骤

操作步骤

1. 在 **Project Management** 视图中，点击 **Create Project**。
2. 在 **Basic information** 页面，配置以下参数：

参数	描述
Name	项目名称，不能与已有项目名称或项目名称黑名单中的任何名称相同，否则无法创建项目。

参数	描述
	注意：项目名称黑名单包括平台集群下的特殊命名空间名称：`cpaas-system`、`cert-manager`、`default`、`global-credentials`、`kube-ovn`、`kube-public`、`kube-system`、`nsx-system`、`alauda-system`、`kube-federation-system`、`ALL-ALL` 和 `true`。
Cluster	与项目关联的集群，管理员可以在其中分配资源配额。点击下拉选择框选择一个或多个集群。 注意：异常状态的集群不可选择。

3. 点击 **Next**，在项目配额设置步骤中，阅读 Manage Resource Quotas 以设置为当前项目在所选集群中分配的资源配额。
4. 点击 **Create Project**。

管理项目配额

本指南介绍了 ACP 如何扩展 Kubernetes 的 ResourceQuota，提供项目级别的聚合配额（ProjectQuota）。ProjectQuota 允许您限制一个项目中所有命名空间的 ResourceQuota 之和，从而在项目层面规划和管理容量，同时仍然可以将限制委托给各个命名空间。

目录

什么是 ProjectQuota？

工作原理

何时使用 ProjectQuota

配额键及单位

分配策略建议

最佳实践与常见问题

什么是 **ProjectQuota**？

- ResourceQuota（Kubernetes 原生）限制每个命名空间的资源（CPU、内存、对象数量等）。有关概念、键和用法，请参阅：
  - Resource Quotas
- ProjectQuota 定义了项目范围内的上限：项目中所有命名空间的 ResourceQuota 总和不得超过该项目对应键的硬限制。

简而言之：ResourceQuota 限制单个命名空间；ProjectQuota 限制一个项目中所有命名空间的总和。

工作原理

- 工作流程顺序：先定义或调整 ProjectQuota，然后在该项目预算内分配每个命名空间的 ResourceQuota。
- 作用范围：ProjectQuota 适用于平台项目，管理属于该项目的所有命名空间。
- 聚合执行于准入时：
  - 在创建或更新命名空间的 ResourceQuota 时，平台会计算项目中所有命名空间对应键（例如 `limits.cpu`、`requests.memory`、`pods`）的聚合值，包括此次变更。
  - 仅当新的聚合值小于或等于对应的 ProjectQuota 硬限制时，才允许该请求。否则，变更会被拒绝并返回错误说明。
- 执行模型：
  - ProjectQuota 限制通过命名空间 ResourceQuota 可分配的资源（预分配），而非瞬时运行时使用量。实际消耗仍由各命名空间的 ResourceQuota 和调度器管理。

何时使用 **ProjectQuota**

- 按项目进行预算/容量管理：分配固定的 CPU/内存/对象预算，再细分到各命名空间。
- 多团队或多环境项目（例如 `dev` / `staging` / `prod`）共享统一上限。
- 防止配额漂移：在项目层保持一个“大桶”，避免命名空间配额随时间无声膨胀。

配额键及单位

ProjectQuota 支持与 ResourceQuota 相同的常用键（非详尽）：

- 计算和内存：`limits.cpu`、`limits.memory`、`requests.cpu`、`requests.memory`
- 工作负载/对象计数：`pods`、`services`、`configmaps`、`secrets`、`pvc` 等

单位及计数规则：

- CPU 使用核心数（例如 `2`、`500m`）
- 内存使用字节（例如 `8Gi`）
- 对象类键使用整数计数

如果所有命名空间对应键的总和接近或超过 ProjectQuota 硬限制，ACP 会阻止该键的 ResourceQuota 创建或扩容。

分配策略建议

- 先定义项目级“大桶”（ProjectQuota），再将其拆分为各命名空间的 ResourceQuota，分配给团队或环境。
- 保留 10% - 30% 的余量以应对峰值和弹性扩缩。
- 定期审查：回收未充分使用的配额并重新分配；提升持续受限的命名空间配额，并相应调整项目上限。

最佳实践与常见问题

- 问：增加某命名空间的 `limits.memory` 时失败，提示超出项目配额，为什么？
  - 答：请求的变更会导致该键的 ProjectQuota 硬限制被超出。请先减少其他命名空间的配额，或先提升项目上限，再重试命名空间变更。
- 问：我提升了 ProjectQuota，但工作负载仍无法调度。
  - 答：请确保各命名空间的 ResourceQuota 也相应提升，并检查底层集群/节点容量。
- 建议：将 ProjectQuota 管理纳入常规变更控制流程，结合容量规划（节点/存储）和预算管理同步进行。

管理项目

本指南说明如何更新指定项目的基本信息和项目配额，或删除项目。

目录

更新项目基本信息
操作步骤
删除项目
操作步骤

更新项目基本信息

更新指定项目的基本信息，如显示名称和描述。

操作步骤

1. 在项目管理视图中，点击要更新的项目名称。
2. 在左侧导航栏中，点击详情。
3. 点击右上角的操作 > 更新基本信息。
4. 修改或输入显示名称和描述。
5. 点击更新。

删除项目

删除不再使用的项目。

> **WARNING**
>
> 项目删除后，项目在集群中占用的资源将被释放。

操作步骤

1. 在项目管理视图中，点击要删除的项目名称。
2. 在左侧导航栏中，点击详情。
3. 点击右上角的操作 > 删除项目。
4. 输入项目名称并点击删除。

管理项目集群

本指南说明如何管理项目的集群关联。您可以添加集群以分配其资源给项目，或移除集群以回收已分配的资源。

目录

介绍

添加集群

操作步骤

移除集群

操作步骤

介绍

您可以向项目添加集群以分配其资源，或移除集群以回收已分配的资源。此功能适用于以下场景：

- 当项目资源不足以支持业务运行时
- 当新创建或新增的集群需要分配给现有项目时
- 当需要从项目中回收集群资源时
- 当某个特定项目需要独占访问某个集群时

添加集群

向项目添加集群并设置其资源配额。

操作步骤

1. 在 **Project Management** 视图中，点击要添加集群的项目名称。
2. 在左侧导航栏中，点击 **Details**。
3. 点击右上角的 **Actions** > **Add Cluster**。
4. 选择集群并设置分配给当前项目的资源配额。可配置的资源包括：

   - CPU（核）
   - 内存（Gi）
   - 存储（Gi）
   - PVC 数量（个）
   - Pods（个）
   - vGPU（虚拟 GPU）/MPS/pGPU（物理 GPU，核）
   - 显存配额

NOTE

- 只有在集群中部署了 GPU 插件时，才能配置 GPU 资源配额。

  当 GPU 资源为 **GPU-Manager** 或 **MPS GPU** 时，也可以配置 **vMemory** 配额。

  GPU 单位：100 个虚拟核等于 1 个物理核（1 pGPU = 1 核 = 100 个 GPU-Manager 核 = 100 个 MPS 核），pGPU 单位只能以整数分配。GPU-Manager 1 单位内存等于 256 Mi，MPS GPU 1 单位内存等于 1 Gi，且 1024 Mi = 1 Gi。
- 如果某类资源未设置配额，则默认为 无限制。这意味着项目可以根据需要使用集群中该类型的可用资源，无最大限制。
- 设置的项目配额值应在页面显示的配额范围内。每个资源配额输入框下方会显示该资源的已分配配额和总量，供参考。

5. 点击 **Add**。

移除集群

移除与项目关联的集群。

> **WARNING**
>
> - 移除集群后，项目将无法使用被移除集群下的业务资源。
> - 当要移除的集群异常时，无法清理该异常集群下的资源，建议先修复集群后再移除。

操作步骤

1. 在 **Project Management** 视图中，点击要移除集群的项目名称。
2. 在左侧导航栏中，点击 **Details**。
3. 点击右上角的 **Actions** > **Remove Cluster**。
4. 在弹出的 **Remove Cluster** 对话框中，输入要移除的集群名称，然后点击 **Remove** 按钮，即可成功移除集群。

管理项目成员

本指南介绍如何管理项目成员，包括导入成员和分配项目相关角色。

目录

导入成员
约束与限制
操作步骤
从成员列表导入
导入 OIDC 用户
移除成员
操作步骤

导入成员

您可以通过导入已有的平台用户或添加 OIDC 用户，授予用户对项目及其命名空间的操作权限。分配平台提供的角色，如项目管理员、命名空间管理员或开发者。如果需要定制权限，可在 **Users > Platform Roles > Kubernetes Roles** 下创建原生 Kubernetes Role/ClusterRole，并通过 RoleBinding/ClusterRoleBinding 绑定到目标用户。

约束与限制

- 当平台未配置 OIDC IDP 时：
  - 只能导入已有的平台用户作为项目成员，包括：
    - 已成功登录的 OIDC 用户
    - 通过 LDAP 同步的用户
    - 本地用户
    - 作为 OIDC 用户添加到其他项目的用户（来源标记为 `-`）
- 当配置了 OIDC IDP 时：
  - 可添加符合输入要求的有效 OIDC 账号
  - 添加时无法验证账号有效性
  - 请确保账号有效，否则无法正常登录
- 系统默认管理员用户及当前登录用户不可导入

操作步骤

1. 在 **Project Management** 视图中，点击要管理的项目名称。
2. 在左侧导航栏点击 **Members**。
3. 点击 **Import Member**。
4. 选择 **Member List** 或 **OIDC Users**。

从成员列表导入

您可以导入全部用户或选定用户。

> **TIP**
>
> 使用右上角的用户组下拉菜单和搜索框按用户名筛选用户。

导入全部用户：

1. 选择 **Member List**。

2. 点击 **Bind** 下拉菜单，选择要分配给所有用户的角色。
   如果角色需要命名空间，请从 **Namespaces** 下拉菜单中选择。
3. 点击 **Import All**。

导入指定用户：

1. 选择 **Member List**。
2. 通过复选框选择一个或多个用户。
3. 点击 **Bind** 下拉菜单，选择要分配给选中用户的角色。
   如果角色需要命名空间，请从 **Namespaces** 下拉菜单中选择。
4. 点击 **Import**。

导入 **OIDC** 用户

1. 选择 **OIDC Users**。
2. 点击 **Add** 创建成员记录（可多次重复添加）。
3. 在 **Name** 字段输入通过 OIDC 认证的用户名。

> **WARNING**
>
> 请确保用户名对应已配置 OIDC 系统可认证的账号，否则登录将失败。

4. 从 **Roles** 下拉菜单选择角色。
   如果角色需要命名空间，请从 **Namespaces** 下拉菜单中选择。
5. 点击 **Import**。

导入成功后，您可以查看：

- 项目成员列表中的成员
- **Platform Management > Users** 中的用户
  - 来源显示为 "-"，直到首次登录/同步
  - 登录/同步成功后来源会更新

移除成员

移除项目成员以撤销其权限。

操作步骤

1. 在 **Project Management** 视图中，点击项目名称。
2. 在左侧导航栏点击 **Members**。

> **TIP**
>
> 使用搜索框旁的下拉列表按 **Name**、**Display name** 或 **User Group** 筛选成员。

3. 点击要移除成员旁的 **Remove**。
4. 在提示对话框中确认移除。

审计

介绍

先决条件

操作步骤

搜索结果

介绍

平台的审计功能提供与用户和系统安全相关的按时间顺序排列的操作记录。这有助于您分析具体问题，并快速解决集群、自定义应用及其他领域中发生的问题。

通过审计，您可以跟踪 Kubernetes 集群中的各种变更，包括：

- 在特定时间段内集群发生了哪些变更
- 谁执行了这些变更（系统组件或用户）
- 重要变更事件的详细信息（例如 POD 参数更新）
- 事件结果（成功或失败）
- 操作者位置（集群内或集群外）
- 用户操作记录（更新、删除、管理操作）及其结果

目录

先决条件

操作步骤

搜索结果

先决条件

- 您的账号必须具有平台管理或平台审计权限。

- 此功能依赖于日志服务。请确保平台内已预先安装 ACP Log Essentails、ACP Log Collector 和 ACP Log Storage 插件。

> **Note**
>
> 因为 Logging Service 的发版周期与灵雀云容器平台不同，所以 Logging Service 的文档现在作为独立的文档站点托管在 Logging Service ↗。

操作步骤

1. 在左侧导航栏中，点击 审计。
2. 从标签页中选择审计范围：
   - 用户操作：查看已登录平台用户的操作记录
   - 系统操作：查看系统操作记录（操作者以 `system:` 开头）
3. 配置查询条件以筛选审计事件：

查询条件	说明
操作者	操作者的用户名或系统账号名称（默认：`全部`）
操作类型	操作类型（创建、更新、删除、管理、回滚、停止等，默认：`全部`）
集群	所操作资源所在的集群（默认：`全部`）
资源类型	所操作资源的类型（默认：`全部`）
资源名称	所操作资源的名称（支持模糊搜索）

4. 点击 搜索。

> **TIP**
>
> - 使用 时间范围 下拉框设置审计时间范围（默认：`最近30分钟`）。您可以选择预设范围或自定义时间段。

- 点击刷新图标以更新搜索结果。
- 点击导出图标将结果下载为 `.csv` 文件。

搜索结果

搜索结果显示以下信息：

参数	说明
操作者	操作者的用户名或系统账号名称
操作类型	操作类型（创建、更新、删除、管理、回滚、停止等）
资源名称/类型	所操作资源的名称和类型
集群	所操作资源所在的集群
命名空间	所操作资源所在的命名空间
客户端 IP	用于执行操作的客户端 IP 地址
操作结果	基于 API 返回码的操作结果（2xx = 成功，其他 = 失败）
操作时间	操作的时间戳
详情	点击 详情 按钮，在 审计详情 弹窗中以 JSON 格式查看完整审计记录

遥测

安装

先决条件

安装步骤

启用在线运维

卸载步骤

安装

ACP Telemetry 是一个平台服务，用于收集集群的遥测数据以支持在线运维。它收集系统指标并上传至 Alauda Cloud 进行监控和分析。

目录

先决条件

安装步骤

启用在线运维

卸载步骤

先决条件

安装前，请确保：

- Alauda Container Platform 拥有有效的许可证
- global 集群具备互联网访问能力

安装步骤

1. 进入 管理员

2. 在左侧导航栏点击 **Marketplace** > **Cluster Plugins**
3. 在顶部导航栏选择 **global** 集群
4. 搜索 **ACP Telemetry** 并点击查看详情
5. 点击 安装 以部署插件

启用在线运维

1. 在左侧导航栏点击 **System Settings** > **Platform Maintenance**
2. 点击 在线运维 的 开启 按钮

卸载步骤

1. 按照安装流程中的步骤 1-4 定位插件
2. 点击 卸载 以移除插件

证书

自动化 **Kubernetes** 证书轮换

cert-manager

OLM 证书

证书监控

轮换平台访问

自动化 Kubernetes 证书轮换

本指南将帮助您在 ACP 中安装、了解和运行集群证书轮换，以自动轮换集群中的 Kubernetes 证书。

CAUTION

仅在需要证书轮换时使用此插件

Kubernetes 证书默认有效期为 10 年。启用此插件后，插件管理的证书将变为短期证书：kubelet 证书为 91 天，控制平面证书为 365 天。

自动续订会重启 API server 和 kubelet 等组件，这可能会影响集群可用性和业务工作负载。如果轮换时间超过 30 秒，节点可能变为 `NotReady`，其 Pod 可能会被驱逐。

如果不需要证书轮换，请勿安装此插件。

目录

- 安装
- 配置
  - 证书设置
  - 运行设置
  - 覆盖证书设置
- 工作原理
  - 轮换流程
- 操作注意事项

安装

请参阅集群插件，了解安装说明。

> 注意：
>
> - 当前支持：
>
> - On-Premises 集群
> - DCS 集群

配置

以下是插件的默认证书设置。控制器使用为每种证书类型配置的续订阈值，并按配置的间隔检查证书。

证书设置

证书	有效期	默认续订阈值	大致轮换时间点
控制平面	`8760h`（365 天）	`1752h`（剩余 73 天）	签发后约 292 天
Kubelet	`2184h`（91 天）	`720h`（剩余 30 天）	签发后约 61 天

使用默认设置时，当剩余有效期少于 20% 或 30 天时（以先达到的阈值为准），证书会被续订。这对应于控制平面证书签发后约 292 天，以及 kubelet 证书签发后约 61 天。实际时间可能因一次检查间隔而有所差异。如果覆盖部署值，则以配置的续订阈值为准。

运行设置

设置	默认值	描述
检查间隔	`24h`	两次证书检查之间的间隔。

设置	默认值	描述
续订 kubelet 客户端证书	已禁用	同时续订 `kubelet.conf` 引用的客户端证书。

覆盖证书设置

控制器接受以下插件参数。时长值使用 Go 的 `time.Duration` 格式；请使用小时，例如 `8760h` 或 `720h`（该格式支持 `ns`、`us`、`ms`、`s`、`m` 和 `h` 等单位）。

参数	默认值	描述
`controlPlaneCertificateDuration`	`8760h`	为控制平面证书请求的有效期。
`controlPlaneCertificateRenewBefore`	`1752h`	触发控制平面证书续订的剩余有效期。
`kubeletCertificateDuration`	`2184h`	为 kubelet 证书请求的有效期。
`kubeletCertificateRenewBefore`	`720h`	触发 kubelet 证书续订的剩余有效期。
`checkInterval`	`24h`	两次证书检查之间的间隔。
`enableRenewKubeletClientCertificate`	`false`	是否续订 `kubelet.conf` 引用的客户端证书。

对于已安装的插件，请在 global 集群中编辑相应的 `ModuleInfo`。插件的 `spec.config` 是受支持的配置输入；cert-rotator `valuesTemplates` 会将这些字段映射到控制器使用的 chart 值：

```
spec:
  config:
    controlPlaneCertificateDuration: "8760h"
    controlPlaneCertificateRenewBefore: "1752h"
    kubeletCertificateDuration: "2184h"
    kubeletCertificateRenewBefore: "720h"
    checkInterval: "24h"
    enableRenewKubeletClientCertificate: true
```

找到已安装的实例并进行编辑：

```
kubectl get moduleinfo \
  -l cpaas.io/cluster-name=<CLUSTER-NAME>,cpaas.io/module-name=cluster-cert-rotator
kubectl edit moduleinfo <MODULEINFO-NAME>
```

有关高级插件 YAML 安装和配置，请参阅集群插件。请确保每个续订阈值都短于其对应的证书有效期。更改有效期也会导致续订有效期不同的证书。

工作原理

此插件会自动轮换以下证书。

证书文件	功能	节点类型
apiserver.crt	kube-apiserver 的服务器证书	控制平面节点
apiserver-etcd-client.crt	kube-apiserver 访问 etcd 的客户端证书	控制平面节点
apiserver-kubelet-client.crt	kube-apiserver 访问 kubelet 的客户端证书	控制平面节点
front-proxy-client.crt	kube-apiserver 访问聚合 API 服务器的客户端证书	控制平面节点
etcd/server.crt	etcd 的服务器证书	控制平面节点

证书文件	功能	节点类型
etcd/peer.crt	etcd 成员之间进行对等通信的证书	控制平面节点
/root/.kube/config, admin.conf, super-admin.conf	用于集群管理的 kubeconfig 中的客户端证书	控制平面节点
controller-manager.conf	kube-controller-manager 的 kubeconfig 中的客户端证书	控制平面节点
scheduler.conf	kube-scheduler 的 kubeconfig 中的客户端证书	控制平面节点
kubelet.crt	kubelet 的服务器证书	所有节点
kubelet-client-current.pem	kubelet 的客户端证书（由 kubelet.conf 引用）	所有节点

轮换流程

1. 加载证书信息
   初始步骤包括收集所有目标证书的元数据。由于这些证书存储在主机上的不同路径中，因此必须从相应文件中读取其内容。为此，会在目标节点上创建一个临时 Pod，并挂载证书目录，使 Pod 能够读取相关信息。证书信息每天收集一次。证书详细信息（路径、到期时间）保存在 ConfigMap `cpaas-system/node-local-certs-<node-name>` 中。加密的 CA 证书存储在 Secret `cpaas-system/kubernetes-ca` 中。
2. 轮换触发条件
   证书的 `notBefore` 和 `notAfter` 字段表示有效期。默认情况下，当剩余有效期少于 20% 或 30 天时（以先达到的阈值为准），会触发轮换。控制器会对每种证书应用配置的续订阈值，因此，覆盖值优先。如果证书的有效期与预期持续时间不同，也会对其进行续订。
3. 轮换队列
   需要轮换的证书会加入处理队列。轮换程序会评估近期的轮换活动和待处理任务的紧迫程度，以决定是否立即处理这些任务。这可以防止同时轮换多个证书而导致潜在的集群运行状况问题。
4. 生成新证书
   轮换程序会根据内部存储的 CA 信息生成新证书。轮换过程会在目标节点上创建一个临时 Pod，并挂载必要的证书目录，以便以受控方式修改文件。

5. 重启组件
   需要重启：

   - `kube-apiserver`：需要重启以加载新证书。重启期间，它会重新生成其内部回环证书（有效期为一年，仅供内部使用，无法在外部轮换）。
   - `kube-controller-manager`：需要重启以重新加载 kubeconfig 文件。
   - `kube-scheduler`：需要重启以重新加载 kubeconfig 文件。
   - `kubelet`：需要重启以重新加载服务器证书。

   重启方法：向相应静态 Pod 的 YAML 文件添加注解，以触发 kubelet 重新创建 Pod。要重启 kubelet，请使用 `hostPID is true` 挂载主机文件系统，并在容器中运行 "systemctl restart kubelet"。
   自动重新加载：

   - Etcd 可以自动重新加载证书。

操作注意事项

如果 `kubelet` 在轮换窗口期间处于异常状态，无法自动轮换证书，则需要手动轮换：

操作员必须手动续订证书。

运行以下命令手动续订证书：

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

例如，续订 `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
```

运行以下命令下载并准备 `cert-renew` 工具：

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

也可以下载 `renew-all.sh`，以续订节点上的所有证书：

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

cert-manager

每个集群将自动部署 **Certificate for cert-manager**

cert-manager 是一个原生的 Kubernetes 证书管理控制器，基于 `Certificate` 资源自动生成和管理 TLS 证书。Kubernetes 集群中的许多组件使用 cert-manager 来管理它们的 TLS 证书，确保通信安全。

目录

Overview

How it works

Identifying cert-manager Managed Certificates

Common Labels and Annotations

Related Resources

Overview

Cert-manager 通过 Kubernetes 自定义资源定义（CRDs）管理证书的生命周期：

- **Certificate**：定义需要管理的证书
- **Issuer/ClusterIssuer**：定义证书颁发者
- **CertificateRequest**：用于处理证书请求的内部资源

How it works

当创建 `Certificate` 资源时，cert-manager 会自动：

1. 生成私钥和证书签名请求
2. 从指定的 Issuer 获取签发的证书
3. 将证书和私钥存储在 Kubernetes Secret 中

此外，cert-manager 会监控证书的有效期，并在证书过期前进行续期，以确保服务的持续可用性。

Identifying cert-manager Managed Certificates

由 cert-manager 管理的证书对应的 `Secret` 资源类型为 `kubernetes.io/tls`，并带有特定的标签和注解。

Common Labels and Annotations

cert-manager 管理的 `Secret` 资源通常包含以下标签和注解：

标签：

- `controller.cert-manager.io/fao: "true"`：标识该 Secret 由 cert-manager 管理，并启用控制器的过滤 Secret 缓存。

注解：

- `cert-manager.io/certificate-name`：证书名称
- `cert-manager.io/common-name`：证书的通用名称
- `cert-manager.io/alt-names`：证书的备用名称
- `cert-manager.io/ip-sans`：证书的 IP 地址
- `cert-manager.io/issuer-kind`：证书颁发者类型
- `cert-manager.io/issuer-name`：证书颁发者名称
- `cert-manager.io/issuer-group`：颁发者的 API 组

- `cert-manager.io/uri-sans`：URI 主题备用名称

Related Resources

- cert-manager Official Documentation ↗

OLM 证书

所有 **Operator Lifecycle Manager (OLM)** 组件的证书——包括 `olm-operator`、`catalog-operator`、`packageserver` 和 `marketplace-operator`——均由系统自动管理。

当安装在其 **ClusterServiceVersion (CSV)** 对象中定义了 **webhooks** 或 **API services** 的 Operators 时，OLM 会自动生成并轮换所需的证书。

证书监控

Cluster Enhancement 为 Kubernetes 集群中使用的证书提供监控能力。监控范围包括：

1. **Kubernetes** 组件证书，包括控制平面和 kubelet 服务器/客户端证书（包括 kubeconfig 客户端证书）
2. 集群中运行的组件的证书，通过检查所有类型为 `kubernetes.io/tls` 的 Secret 实现
3. **kube-apiserver** 实际使用的服务器证书（包括用于自访问的内部回环证书），通过访问 `kubernetes` Endpoints 获取

用户可以在左侧导航中依次进入 **Marketplace** > **Cluster Plugins**，在 管理员 视图中查找并安装 **Cluster Enhancement**。

目录

证书状态监控

内置告警规则

Kubernetes 证书告警

平台组件证书告警

证书状态监控

可以通过指标 `certificate_expires_status` 查看证书的过期状态。可以通过指标 `certificate_expires_time` 查看证书的过期时间。

可以在 证书状态 子选项卡中查看当前证书状态和过期时间。要访问此子选项卡，请进入 管理员 视图，导航至 集群 **>** 集群，选择特定集群，然后进入 监控 选项卡。

内置告警规则

Cluster Enhancement 提供内置告警规则 `cpaas-certificates-rule`，包括以下告警：

Kubernetes 证书告警

规则	级别
Kubernetes 证书即将过期（少于 30 天）<= 30d，持续 1 分钟	中
Kubernetes 证书即将过期（少于 10 天）<= 10d，持续 1 分钟	高
Kubernetes 证书已过期 <= 0d，持续 1 分钟	严重

平台组件证书告警

规则	级别
平台组件证书即将过期（少于 30 天）<= 30d，持续 1 分钟	中
平台组件证书即将过期（少于 10 天）<= 10d，持续 1 分钟	高
平台组件证书已过期 <= 0d，持续 1 分钟	严重

轮换平台访问地址的 TLS 证书

INFO

对于 ACP 版本 v4.0.x，请按照此处描述的操作步骤，分别在主集群和备用集群（灾备配置中的术语）中执行相同的操作。

目录

前提条件

操作步骤

前提条件

- 一对 TLS 证书及其私钥。

操作步骤

1. 在 global 集群中的任一 control-plane 节点上，导出 ACP 平台访问地址所使用的 TLS 证书备份：

```
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. 删除当前证书：

```
kubectl delete certificate -n cpaas-system dex-serving-cert --ignore-not-found=true
kubectl delete secret -n cpaas-system dex.tls
```

3. 引入新证书：

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