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

工作原理

运行注意事项

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

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

使用指南

安装 Alauda Container Platform

通过控制台安装

升级

兼容性矩阵

南

API Refiner

介绍

产品介绍

限制

安装 Alauda Container Platform

通过控制台安装

通过 YAML 安装

卸载流程

默认配置

升级

兼容性矩阵

升级路径指南

关于 Alauda Container Platform Compliance Service

关于 Alauda Container Platform Compliance Service

合规性

介绍

介绍

安装 Alauda Container Platform Compliance with Kyverno

安装 Alauda Container Platform Compliance with Kyverno

通过控制台安装

通过 YAML 安装

卸载流程

升级

升级

兼容性矩阵

升级路径指南

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

镜像签名验证策略

什么是镜像签名验证？

快速开始

常见用例

使用 Secret

为什么使用 Sec

快速开始

Secret 创建方式

常见使用场景

镜像仓库验证策略

什么是镜像仓库验证？

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

什么是安全上下

快速开始

核心安全上下文

高级场景

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

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 Compliance with Kyverno

Alauda Container Platform Compliance with Kyverno 是一项集成了 Kyverno 的平台服务，用于在 Alauda Container Platform 上管理合规策略。

目录

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. 搜索 **Alauda Container Platform Compliance with Kyverno** 并点击查看详情
4. 点击 安装 部署该插件

通过 YAML 安装

1. 检查可用版本

确保插件已经发布，可通过检查 `global` 集群中的 ModulePlugin 和 ModuleConfig 资源：

```
# 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`，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，且不带任何配置参数：

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

字段说明：

- `name`：集群插件的临时名称。平台会根据内容在创建后重命名，格式为 `<cluster-name>-<内容哈希>`，例如 `global-ee98c9991ea1464aaa8054bdacbab313`。

- `label cpaas.io/cluster-name`：指定插件安装的集群。
- `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`；缺失该字段会导致安装失败。
- `.spec.config`：如果对应的 ModuleConfig 为空，该字段可留空。
- `.spec.version`：指定安装的插件版本，必须与 ModuleConfig 中的 `.spec.version` 匹配。

卸载流程

1. 按安装流程中的步骤 1-3 定位插件
2. 点击 卸载 移除该插件

升级

INFO

本文档提供了 Alauda Container Platform Compliance for Kyverno 的升级路径原则和支持的版本兼容性。

目录

兼容性矩阵

下表列出了 Alauda Container Platform Compliance for Kyverno 支持的版本：

Compliance for Kyverno Version	Supported Alauda Container Platform Version
4.3.x	4.2.x, 4.3.x
4.2.x	4.2.x

升级路径指南

随 **ACP** 一起升级

- **Description**: 在 ACP 4.1.x 及更早版本中，Compliance for Kyverno plugins 的生命周期与 ACP 保持一致，并作为 ACP 发布的一部分进行升级。对于以下升级路径，在 ACP 升级过程中，plugins 会随 ACP 一起升级到 4.3.0。
- **Supported ACP Upgrade Paths**:
  - `4.0.x -> 4.3.0`
  - `4.1.x -> 4.3.0`
- **Result**: 升级完成后，所有 Compliance for Kyverno plugins 都会升级到 4.3.0。

次版本升级

- **Description**: 当 ACP 处于兼容版本时，Compliance for Kyverno plugins 支持次版本升级。
- **Prerequisite**: ACP 版本必须满足上方的兼容性矩阵要求。
- **Example**: `4.2.x -> 4.3.x`

补丁级升级

- **Description**: 在同一次版本内任意补丁版本之间的升级完全兼容，可直接执行。
- **Example**: `4.3.0 -> 4.3.x`

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

镜像签名验证策略

什么是镜像签名验证？

快速开始

常见用例

使用 Secret

为什么使用 Sec

快速开始

Secret 创建方式

常见使用场景

镜像仓库验证策略

什么是镜像仓库验证？

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

什么是安全上下

快速开始

核心安全上下文

高级场景

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

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？

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
```

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？

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？

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？

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

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

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

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？

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
```

API Refiner

介绍

- 产品介绍
- 限制

安装 Alauda Container Platform

- 通过控制台安装
- 通过 YAML 安装
- 卸载流程
- 默认配置

升级

- 兼容性矩阵
- 升级路径指南

介绍

目录

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

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`

升级

INFO

本文档提供了 Alauda Container Platform API Refiner 的升级路径原则和支持的版本兼容性。

目录

兼容性矩阵

下表列出了 Alauda Container Platform API Refiner 支持的版本：

API Refiner Version	Supported Alauda Container Platform Version
4.3.x	4.2.x, 4.3.x
4.2.x	4.2.x

升级路径指南

与 **ACP** 一起升级

- **Description**: 在 ACP 4.1.x 及更早版本中，API Refiner 插件与 ACP 的生命周期保持一致，并作为 ACP 发布的一部分进行升级。对于以下升级路径，在 ACP 升级过程中，插件会随 ACP 一起升级到 4.3.0。
- **Supported ACP Upgrade Paths**:
  - `4.0.x -> 4.3.0`
  - `4.1.x -> 4.3.0`
- **Result**: 升级完成后，所有 API Refiner 插件都会升级到 4.3.0。

次版本升级

- **Description**: 当 ACP 处于兼容版本时，API Refiner 插件支持次版本升级。
- **Prerequisite**: ACP 版本必须满足上述兼容性矩阵。
- **Example**: `4.2.x -> 4.3.x`

补丁级升级

- **Description**: 同一次版本内任意补丁版本之间的升级完全兼容，可直接执行。
- **Example**: `4.3.0 -> 4.3.x`

关于 Alauda Container Platform Compliance Service

Compliance Service 是一个平台模块，旨在支持 STIG 合规性扫描和 MicroOS 操作系统扫描。它提供开箱即用的合规性扫描功能，支持定时扫描和全面的报告生成。

> **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

ACP 使用 OIDC Client Secret 进行平台组件之间的身份验证。出于安全合规、定期凭证轮换或在发生潜在凭证泄露后，您可能需要轮换此密钥。在 global 集群上更新密钥后，平台会自动将其同步到所有成员集群。

目录

前提条件

- 具有 cluster-admin 权限的 `kubectl` 访问权限，可访问 **global** 集群。
- 平台管理的所有集群均已升级到 ACP v4.3。
- 以下具有 `Agnostic` 生命周期的插件已升级到与 ACP v4.3 兼容的版本：
  - `Alauda Container Platform Gitops`
  - `Alauda Build of Kiali`

- `Kubeflow Base`
- `Alauda AI`

- 已准备好新的 client secret 值。建议使用至少 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，从而造成临时身份验证中断。请在维护窗口期间执行此操作。

OIDC Secret 概览

平台 OIDC 凭证存储在一个 Kubernetes Secret 中，详细信息如下：

属性	值
Secret 名称	`cpaas-oidc-secret`
Namespace	`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。这是您将要轮换的值。

操作步骤

1 备份当前 **secret**

在进行更改之前，请先备份当前的 Secret 清单，以便在需要时回滚。

```
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。请将其存放在安全位置，并在操作完成后将其删除。

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

```
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 "当前 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该 Secret。这样做可能会导致整个平台出现级联身份验证失败。

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）

通过安全的交互式输入读取新密钥：

```
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该 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 "更新后的 client-secret SHA256: ${NEW_SECRET_SHA256}"
test "${OLD_SECRET_SHA256}" != "${NEW_SECRET_SHA256}" && \
  echo "验证通过：client-secret 已更改。"
unset OLD_SECRET_SHA256 NEW_SECRET_SHA256
```

3 验证组件健康状态

Secret 更新后，`base-operator` 会自动将其同步到所有成员集群。依赖 OIDC 凭证的某些组件会重启以加载新值。特别是，请确认 `frontend` 和 `apollo` 部署运行正常：

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

确认所有 Pod 均处于 `READY` 和 `UP-TO-DATE` 状态。重启完成可能需要几分钟。

INFO

> 如果某个组件启动失败，请确认 `cpaas-oidc-secret` Secret 是否存在于 `cpaas-system` Namespace 中，并且包含有效的 `client-id` 和 `client-secret` 值。

回滚

如果在更改 Secret 后出现问题，请从 步骤 1 创建的备份中恢复之前的值：

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

或者，如果您知道之前的密钥值：

```
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，确认所有组件均处于健康状态。

禁用 PKCE Plain 方法

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

一旦所有受影响的 `Agnostic` 插件都升级到与 ACP v4.3.0 兼容的版本，你应移除 `plain` 方法，以强制仅使用 `S256` 的 PKCE 验证。

目录

前提条件

- 具有 cluster-admin 权限的 `kubectl` 访问权限，可访问 **global** 集群。
- 平台管理的所有集群均已升级到 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 Build of Kiali`
  - `Kubeflow Base`

- `Alauda AI`

> **WARNING**
>
> 在此上下文中，受影响插件是指那些使用平台 OIDC 授权流程的已列出 `Agnostic` 插件；在它们全部升级到与 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 客户端配置存储为 `cpaas-system` 命名空间中的 `OAuth2Client` 自定义资源。根据 client 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，并删除 `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"]
```

对每个受影响插件（`Alauda Container Platform Gitops`、`Alauda Build of Kiali`、`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

如果在移除 `plain` 方法后出现身份验证问题（例如，任何受影响插件的登录或授权失败），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

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

组

介绍

组介绍

组类型

操作指南

角色

介绍

角色介绍

操作指南

系统角色

自定义权限

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

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

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

- 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 - 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。

状态	描述
	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）

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

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： - 长度：8-32 个字符 - 必须包含字母和数字

参数	描述
	- 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

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

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

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

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

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

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

操作指南

介绍

目录

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. 支持的配置方
5. 配置
6. 复制与裁剪

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

- 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）

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

本文档介绍自定义角色的核心概念及如何配置 RoleTemplate。

2. 核心概念

- **RoleTemplate**：自定义角色模板。定义角色语义和权限集合，由控制器转换为 ClusterRoles。
- **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；可通过标签聚合成系统角色。
- **UserBinding**：用户与角色/作用域的绑定；控制器根据 UserBinding 生成 RoleBinding/ClusterRoleBinding 以实现最终授权。

> **NOTE**
>
> - `functionResourceRef` 的值来源于 FunctionResource 的 `metadata.name`。
> - 显示名称通常在 `metadata.annotations` 中，可用于映射到 UI 模块。

3. 技术变更

从 ACP v4.3 起，功能权限逐步从 FunctionResource 迁移到原生 K8s ClusterRole 管理。RoleTemplate 模块权限将通过 ClusterRole 标签聚合到系统角色中。基于 FunctionResource 的管理将在 v4.5 中废弃。

4. 支持的配置方式

- **FunctionResource** 方式：按产品模块选择权限，细粒度控制动词。
- **ClusterRole** 聚合方式（**aggregationRules**）：通过 ClusterRole 标签集中聚合，推荐 v4.2 及以上版本使用。
- **customRules**：使用原生 K8s RBAC 语法配置权限；规则格式与 ClusterRole 的 `rules` 相同。

5. 配置

5.1 RoleTemplate YAML

FunctionResource 方式（支持至 v4.5）

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

FunctionResource 示例：

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

ClusterRole 聚合方式（v4.2+）

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-aggrole
  annotations:
    cpaas.io/display-name: 示例聚合角色
    cpaas.io/description: ClusterRole 聚合示例
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-namespace-developer: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
  rules: []
```

带聚合标签的 ClusterRole 示例：

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

customRules 方式

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: demo-customrules
  annotations:
    cpaas.io/display-name: 示例自定义规则角色
    cpaas.io/description: customRules 示例
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  customRules:
    - apiGroups: [""]
      resources: ["configmaps"]
      verbs: ["get", "list", "watch"]
```

RoleTemplate 字段参考（含取值范围）：

字段	说明
`metadata.name`	角色名称
`metadata.labels.auth.cpaas.io/roletemplate.level`	角色层级
`metadata.annotations.cpaas.io/(display-name/description)`	显示名称和描述
`spec.rules[].functionResourceRef`	FunctionResource 引用
`spec.rules[].verbs`	动词
`spec.aggregationRules[].scope`	聚合作用域

字段	说明
`spec.aggregationRules[].clusterRoleSelectors`	标签选择器
`spec.aggregationRules[].clusterRoleSelectors.matchLabels`	标签键值
`spec.aggregationRules[].clusterRoleSelectors.matchExpressions`	表达式
`spec.customRules[].apiGroups`	API 组
`spec.customRules[].resources`	资源
`spec.customRules[].verbs`	动词
`spec.customRules[].resourceNames`	资源名称
`spec.customRules[].nonResourceURLs`	非资源 URL

NOTE

`spec.rules` 与 `spec.aggregationRules` 互斥。

系统匹配标签

系统角色聚合标签：

- `rbac.cpaas.io/aggregate-to-platform-admin: "true"`
- `rbac.cpaas.io/aggregate-to-platform-auditor: "true"`
- `rbac.cpaas.io/aggregate-to-cluster-admin: "true"`

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

作用域聚合标签：

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

> **NOTE**
>
> 也支持自定义标签进行聚合。

FunctionResource 字段参考：

字段	说明
`metadata.name`	FunctionResource 标识（用于 `functionResourceRef`）
`metadata.annotations`	显示名称（用于 UI 映射）
`metadata.labels`	模块/功能元数据（用于索引）
`spec.rules[].apiGroup`	API 组（`""` 表示核心组）
`spec.rules[].resources`	资源类型
`spec.rules[].bindScope`	作用域（`cluster / namespace`）
`spec.rules[].bindCluster`	集群作用域（`global / unlimit / business`）

字段	说明
`spec.rules[].bindNamespacePart`	命名空间部分（`common / system / kube-public / project_ns / cluster / ""`）

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

5.2 获取 RoleTemplate 生成的 ClusterRoles

通过标签查询 RoleTemplate 生成的 ClusterRoles：

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

6. 复制与裁剪（示例）

自定义角色建议复制内置系统模板并裁剪，避免遗漏模块权限。

6.1 裁剪系统 RoleTemplate YAML

步骤：

- 保留必要的 FunctionResources。
- 动词降为只读（get/list/watch）。
- 删除未使用模块。

角色示例：无 Secret 权限的 developer auditor

> **NOTE**

- 自定义角色中不配置命名空间资源管理（FunctionResource：`acp-namespace-resource-manage`）。
- 删除用户 Secret 字典（FunctionResource：`acp-user-secret`）。
- 动词全部设置为 get/list/watch。

示例 YAML：

```yaml
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  annotations:
    cpaas.io/description: 负责命名空间内的开发、部署和维护。
    cpaas.io/display-name: Developer Copy
  labels:
    auth.cpaas.io/roletemplate.level: namespace
  name: namespace-developer-system-copy
spec:
  rules:
    - functionResourceRef: views-acp-userview
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-clusters
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-nodesmanage
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-domains
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: acp-subnet
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: acp-networkpolicies
      verbs:
        - get
        - list
        - watch
    - functionResourceRef: infrastructure-storageclasses
      verbs:
```

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

6.2 将 namespace-developer-system 裁剪为 auditor 角色

思路：

- 保持作用域结构（cluster / project-ns / business-ns / system-ns / kube-public）。
- 用 auditor 标签替换聚合标签（需对应 ClusterRole 已打标签）。

示例（节选）：

```
apiVersion: auth.alauda.io/v1beta1
kind: RoleTemplate
metadata:
  name: namespace-auditor
  annotations:
    cpaas.io/display-name: 命名空间审计员
    cpaas.io/description: 只读审计
  labels:
    auth.cpaas.io/roletemplate.level: namespace
spec:
  aggregationRules:
    - clusterRoleSelectors:
        - matchLabels:
            rbac.cpaas.io/aggregate-to-namespace-auditor: "true"
            rbac.cpaas.io/aggregate-to-scope-business-ns: "true"
      scope: business-ns
  rules: []
```

NOTE

`aggregate-to-namespace-auditor` 必须与已有 ClusterRole 标签匹配，否则请先创建只读 ClusterRole 并添加该标签。

IDP

介绍

介绍

Overview

Supported Integration Methods

操作指南

LDAP 管理

LDAP 概述

支持的 LDAP 类型

LDAP 术语

添加 LDAP

LDAP 配置示例

同步 LDAP 用户

相关操作

OIDC 管理

OIDC 概述

添加 OIDC

通过 YAML 添加 OIDC

相关操作

故障排除

删除用户

问题描述

解决方案

介绍

目录

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

OIDC 管理

LDAP 管理

平台管理员可以在平台上添加、更新和删除 LDAP 服务。

目录

LDAP 概述

LDAP（Lightweight Directory Access Protocol，轻量级目录访问协议）是一种成熟、灵活且支持良好的标准机制，用于与目录服务器交互。它以层级树状结构组织数据，用于存储企业用户和组织信息，主要用于实现单点登录（SSO）。

> **NOTE**
>
> LDAP 主要特性：
>
> - 支持客户端与 LDAP 服务器之间的通信
> - 支持数据存储、检索和搜索操作
> - 提供客户端认证能力
> - 便于与其他系统集成

更多信息请参考 LDAP 官方文档↗。

支持的 LDAP 类型

OpenLDAP

OpenLDAP 是 LDAP 的开源实现。如果您的组织使用开源 LDAP 进行用户认证，可以通过添加 LDAP 并配置相关参数，使平台与 LDAP 服务通信。

> **NOTE**
>
> OpenLDAP 集成：
>
> - 支持平台对 LDAP 用户的认证
> - 支持标准 LDAP 协议
> - 提供灵活的用户管理

更多关于 OpenLDAP 的信息，请参考 OpenLDAP 官方文档↗。

Active Directory

Active Directory 是微软基于 LDAP 的软件，用于在 Windows 系统中提供目录存储服务。如果您的组织使用微软 Active Directory 进行用户管理，可以配置平台与 Active Directory 服务通信。

> **NOTE**
>
> Active Directory 集成：
>
> - 支持平台对 AD 用户的认证
> - 支持 Windows 域集成
> - 提供企业级用户管理

LDAP 术语

OpenLDAP 常用术语

术语	描述	示例
dc (Domain Component)	域组件	`dc=example,dc=com`
ou (Organizational Unit)	组织单位	`ou=People,dc=example,dc=com`
cn (Common Name)	通用名称	`cn=admin,dc=example,dc=com`
uid (User ID)	用户 ID	`uid=example`
objectClass (Object Class)	对象类	`objectClass=inetOrgPerson`
mail (Mail)	邮箱	`mail=example@126.com`
givenName (Given Name)	名	`givenName=xq`

术语	描述	示例
sn (Surname)	姓	`sn=ren`
objectClass: groupOfNames	用户组	`objectClass: groupOfNames`
member (Member)	组成员属性	`member=cn=admin,dc=example,dc=com`
memberOf	用户组成员属性	`memberOf=cn=users,dc=example,dc=com`

Active Directory 常用术语

术语	描述	示例
dc (Domain Component)	域组件	`dc=example,dc=com`
ou (Organizational Unit)	组织单位	`ou=People,dc=example,dc=com`
cn (Common Name)	通用名称	`cn=admin,dc=example,dc=com`
sAMAccountName/userPrincipalName	用户标识	`userPrincipalName=example` 或 `sAMAccountName=example`
objectClass: user	AD 用户对	`objectClass=user`

术语	描述	示例
	象类	
mail (Mail)	邮箱	`mail=example@126.com`
displayName	显示名称	`displayName=example`
givenName (Given Name)	名	`givenName=xq`
sn (Surname)	姓	`sn=ren`
objectClass: group	用户组	`objectClass: group`
member (Member)	组成员属性	`member=CN=Admin,DC=example,DC=com`
memberOf	用户组成员属性	`memberOf=CN=Users,DC=example,DC=com`

添加 **LDAP**

TIP

LDAP 集成成功后：

- 用户可使用企业账号登录平台
- 多次添加同一 LDAP 会覆盖之前同步的用户

前提条件

添加 LDAP 前，请准备以下信息：

- LDAP 服务器地址
- 管理员用户名
- 管理员密码
- 其他所需配置详情

操作步骤

1. 在左侧导航栏，点击 **Users** > **IDPs**
2. 点击 **Add LDAP**
3. 配置以下参数：

基本信息

参数	描述
Server Address	LDAP 服务器访问地址（例如：`192.168.156.141:31758`）
Username	LDAP 管理员 DN（例如：`cn=admin,dc=example,dc=com`）
Password	LDAP 管理员账号密码
Login Box Username Prompt	用户名输入提示信息（例如：“请输入您的用户名”）

搜索设置

NOTE

搜索设置作用：

- 根据指定条件匹配 LDAP 用户条目
- 提取关键用户和组属性
- 将 LDAP 属性映射到平台用户属性

参数	描述
Object Type	用户的 ObjectClass： - OpenLDAP: `inetOrgPerson` - Active Directory: `organizationalPerson` - 组: `posixGroup`
Login Field	用作登录用户名的属性： - OpenLDAP: `mail`（邮箱地址） - Active Directory: `userPrincipalName`
Filter Conditions	用户/组过滤的 LDAP 过滤条件 示例：`(&(cn=John*)(givenName=*xq*))`
Search Starting Point	用户/组搜索的 Base DN（例如：`dc=example,dc=org`）
Search Scope	搜索范围： - `sub`：整个目录子树 - `one`：起始点下一层
Login Attribute	唯一用户标识： - OpenLDAP: `uid` - Active Directory: `distinguishedName`
Name Attribute	对象名称属性（默认：`cn`）
Email Attribute	邮箱属性： - OpenLDAP: `mail` - Active Directory: `userPrincipalName`
Group Member Attribute	组成员标识（默认：`uid`）

参数	描述
Group Attribute	用户组关系属性（默认：`memberuid`）

4. 在 **IDP Service Configuration Validation** 区域：
   - 输入有效的 LDAP 账号用户名和密码
   - 用户名必须与 **Login Field** 设置匹配
   - 点击验证配置
5. （可选）配置 **LDAP** 自动同步策略：
   - 启用 **Auto-Sync Users** 开关
   - 设置同步规则
   - 使用 在线工具↗ 验证 CRON 表达式
6. 点击 **Add**

> **NOTE**
>
> 添加 LDAP 后：
>
> - 用户可在同步前登录
> - 首次登录时自动同步用户信息
> - 根据配置规则自动同步

LDAP 配置示例

LDAP Connector 配置示例

以下示例展示如何配置 LDAP connector：

```yaml
apiVersion: dex.coreos.com/v1
kind: Connector
id: ldap        # Connector ID
name: ldap      # Connector 显示名称
type: ldap      # Connector 类型为 LDAP
metadata:
  name: ldap
  namespace: cpaas-system
spec:
  config:
    # LDAP 服务器地址和端口
    host: ldap.example.com:636
    # connector 使用的服务账号的 DN 和密码。
    # 此 DN 用于搜索用户和组。
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    # 服务账号密码，创建 connector 时必填。
    bindPW: password

    # 登录账号提示，例如用户名
    usernamePrompt: SSO Username

    # 用户搜索配置
    userSearch:
      # 从 base DN 开始搜索
      baseDN: cn=users,dc=example,dc=com
      # LDAP 查询语句，用于搜索用户。
      # 例如："(&(objectClass=person)(uid=<username>))"
      filter: (&(objectClass=organizationalPerson))

      # 以下字段为用户条目属性的直接映射。
      # 用户 ID 属性
      idAttr: uid
      # 必填。映射为邮箱的属性
      emailAttr: mail
      # 必填。映射为用户名的属性
      nameAttr: cn
      # 登录用户名属性
      # 过滤条件将转换为 "(<attr>=<username>)"，如 (uid=example)。
      username: uid

      # 扩展属性
      # phoneAttr: phone

      # 组搜索配置
```

```
    # 组搜索配置
    groupSearch:
      # 从 base DN 开始搜索
      baseDN: cn=groups,dc=freeipa,dc=example,dc=com
      # 组过滤条件
      # "(&(objectClass=group)(member=<user uid>))"。
      filter: "(objectClass=group)"
      # 用户组匹配字段
      # 组属性
      groupAttr: member
      # 用户组成员属性
      userAttr: uid
      # 组显示名称
      nameAttr: cn
```

用户过滤示例

```
# 1. 基础过滤：查找所有用户
(&(objectClass=person))

# 2. 多条件组合：查找特定部门的用户
(&(objectClass=person)(departmentNumber=1000))

# 3. 查找启用的用户（Active Directory）
(&(objectClass=user)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))

# 4. 查找特定邮箱域的用户
(&(objectClass=person)(mail=*@example.com))

# 5. 查找特定组成员
(&(objectClass=person)(memberOf=cn=developers,ou=groups,dc=example,dc=com))

# 6. 查找最近登录的用户（Active Directory）
(&(objectClass=user)(lastLogon>=20240101000000.0Z))

# 7. 排除系统账号
(&(objectClass=person)(!(uid=admin))(!(uid=system)))

# 8. 查找具有特定属性的用户
(&(objectClass=person)(mobile=*))

# 9. 查找多个部门的用户
(&(objectClass=person)(|(ou=IT)(ou=HR)(ou=Finance)))

# 10. 复杂条件组合示例
(&
  (objectClass=person)
  (|(department=IT)(department=Engineering))
  (!(title=Intern))
  (manager=cn=John Doe,ou=People,dc=example,dc=com)
)
```

组搜索配置示例

```
# 1. 基础过滤：查找所有组
(objectClass=groupOfNames)

# 2. 查找具有特定前缀的组
(&(objectClass=groupOfNames)(cn=dev-*))

# 3. 查找非空组
(&(objectClass=groupOfNames)(member=*))

# 4. 查找具有特定成员的组
(&(objectClass=groupOfNames)(member=uid=john,ou=People,dc=example,dc=com))

# 5. 查找嵌套组（Active Directory）
(&(objectClass=group)(|(groupType=-2147483646)(groupType=-2147483644)))

# 6. 查找具有特定描述的组
(&(objectClass=groupOfNames)(description=*admin*))

# 7. 排除系统组
(&(objectClass=groupOfNames)(!(cn=system*)))

# 8. 查找具有特定成员的组
(&(objectClass=groupOfNames)(|(cn=admins)(cn=developers)(cn=operators)))

# 9. 查找特定 OU 下的组
(&(objectClass=groupOfNames)(ou=IT))

# 10. 复杂条件组合示例
(&
  (objectClass=groupOfNames)
  (|(cn=prod-*)(cn=dev-*))
  (!(cn=deprecated-*))
  (owner=cn=admin,dc=example,dc=com)
)
```

LDAP 过滤器中 AND(&) 和 OR(|) 操作符示例

```
# AND 操作符 (&) - 所有条件必须满足
# 语法：(&(condition1)(condition2)(condition3)...)

# 多属性 AND 示例
(&
  (objectClass=person)
  (mail=*@example.com)
  (title=Engineer)
  (manager=*)
)

# OR 操作符 (|) - 至少满足一个条件
# 语法：(|(condition1)(condition2)(condition3)...)

# 多属性 OR 示例
(|
  (department=IT)
  (department=HR)
  (department=Finance)
)

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

# 复杂条件组合
(&
  (objectClass=person)
  (|
    (&
      (department=IT)
      (title=*Engineer*)
    )
    (&
      (department=R&D)
      (title=*Developer*)
    )
```

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

同步 LDAP 用户

成功将 LDAP 用户同步到平台后，可在用户列表中查看同步的用户。

您可以在添加 LDAP 时配置自动同步策略（后续可更新），也可以在成功添加 LDAP 后手动触发同步操作。以下是手动触发同步的操作步骤。

注意事项：

- 集成到平台的 LDAP 中新增的用户，在执行用户同步操作前即可登录平台。用户首次成功登录平台后，其信息会自动同步到平台。
- 从 LDAP 删除的用户，同步后状态为 `Invalid`。
- 新同步用户的默认有效期为 永久。
- 同步的用户若与现有用户（本地用户、IDP 用户）同名，将自动关联。其权限和有效期与现有用户保持一致，可使用对应来源的登录方式登录平台。

操作步骤

1. 在左侧导航栏，点击 **Users** > **IDPs**。
2. 点击要手动同步的 ***LDAP** 名称*。
3. 点击右上角 **Actions** > **Sync user**。
4. 点击 **Sync**。
   注意：若您手动关闭同步提示对话框，系统会弹出确认关闭的提示。关闭同步提示对话框后，系统仍会继续同步用户。若您停留在用户列表页面，将收到同步结果反馈；若离开用户列表页面，则不会收到同步结果。

相关操作

您可以在列表页右侧点击
或在详情页右上角点击 **Actions**，根据需要更新或删除 LDAP。

操作	描述
更新 **LDAP**	更新已添加 LDAP 的配置信息或 **LDAP** 自动同步策略。 注意：更新 LDAP 后，通过该 LDAP 同步到平台的现有用户也会被更新。LDAP 中被移除的用户在平台用户列表中将变为无效。您可以执行清理无效用户操作来清理垃圾数据。
删除 **LDAP**	删除 LDAP 后，通过该 LDAP 同步到平台的所有用户状态将变为 **Invalid**（用户与角色的绑定关系保持不变），且无法登录平台。重新集成后需重新执行同步以激活用户。 提示：删除 IDP 后，如需删除通过 LDAP 同步到平台的用户和用户组，请勾选提示框下方的 **Clean IDP Users and User Groups** 复选框。

OIDC 管理

平台支持 OIDC（OpenID Connect）协议，平台管理员在添加 OIDC 配置后，可以使用第三方账号登录平台。平台管理员还可以更新和删除已配置的 OIDC 服务。

目录

OIDC 概述

OIDC（OpenID Connect）是一种基于 OAuth 2.0 协议的身份认证标准协议。它使用 OAuth 2.0 授权服务器为第三方客户端提供用户身份认证，并将相应的身份认证信息传递给客户端。

OIDC 允许所有类型的客户端（包括服务器端、移动端和 JavaScript 客户端）请求并接收经过认证的会话和终端用户信息。该规范套件具有可扩展性，允许参与者在有意义时使用可选功能，如身份数据加密、OpenID Provider 发现和会话管理。更多信息请参见 OIDC 官方文档↗。

添加 OIDC

通过添加 OIDC，可以使用第三方平台账号登录平台。

注意：OIDC 用户成功登录平台后，平台会使用用户的 email 属性作为唯一标识。支持 OIDC 的第三方平台用户必须拥有 **email** 属性，否则无法登录平台。

操作步骤

1. 在左侧导航栏点击 **Users** > **IDPs**。
2. 点击 **Add OIDC**。
3. 配置 **Basic Information** 参数。
4. 配置 **OIDC Server Configuration** 参数：
   - **Identity Provider URL**：发行者 URL，即 OIDC 身份提供者的访问地址。
   - **Client ID**：OIDC 客户端的客户端标识。
   - **Client Secret**：OIDC 客户端的密钥。
   - **Redirect URI**：登录第三方平台后的回调地址，即 dex 发行者的 URL + `/callback`。
   - **Logout URL**：用户执行 **Logout** 操作后访问的地址，若为空，则登出地址为平台的初始登录页。
5. 在 **IDP Service Configuration Validation** 区域，输入有效 OIDC 账号的 **Username** 和 **Password** 以验证配置。
   提示：若用户名或密码错误，添加时会报错提示凭证无效，无法添加 OIDC。
6. 点击 **Create**。

通过 YAML 添加 OIDC

除了表单配置外，平台还支持通过 YAML 添加 OIDC，支持更灵活地配置认证参数、声明映射、用户组同步等高级功能。

示例：配置 OIDC Connector

以下示例展示了如何配置 OIDC connector 以集成 OIDC 身份认证服务。该配置示例适用于以下场景：

1. 需要集成 OIDC 作为身份认证服务器。
2. 需要支持用户组信息同步。
3. 需要自定义登出重定向地址。
4. 需要配置特定的 OIDC scopes。
5. 需要自定义声明映射。

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

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

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

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

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

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

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

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

相关操作

您可以在列表页面点击右侧的

或在详情页面右上角点击 **Actions**，根据需要更新或删除 OIDC。

操作	描述
更新 **OIDC**	更新已添加的 OIDC 配置。更新 OIDC 配置信息后，原有用户和认证方式将被重置，并根据当前配置进行同步。
删除 **OIDC**	删除平台不再使用的 OIDC。删除 OIDC 后，通过该 OIDC 同步到平台的所有用户状态将变为 **Invalid**（用户与角色的绑定关系保持不变），且无法登录平台。重新集成后，用户通过成功登录平台可被激活。 提示：删除 IDP 后，如需删除通过 OIDC 同步到平台的用户和用户组，请勾选提示框下方的 **Clean IDP Users and User Groups** 复选框。

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

问题：在创建或同步新用户时，系统提示用户已存在。该如何处理？

出于安全考虑，平台禁止创建与之前已删除用户同名的新用户（包括本地用户和 IDP 用户）。此限制适用于：

- 创建与已删除用户同名的本地新用户
- 同步与已删除用户同名的 IDP 用户

升级到当前版本后，您可能会遇到以下情况导致该问题：

- 创建与升级前已删除用户同名的新用户
- 同步与升级前已删除用户同名的新用户

解决方案

为解决该问题，您需要在 global 集群的任一控制节点上执行特定脚本，清理已删除用户信息。

清理已删除的 IDP 用户

在 global 集群的任一控制节点执行以下命令：

```
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 集群的任一控制节点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

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

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

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

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**？

- 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

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

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

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

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

- 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

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

本指南帮助您在 ACP 中安装、理解和操作 Kubernetes 证书轮换器，以实现集群内 Kubernetes 证书的自动轮换。

目录

安装

请参见 Cluster Plugin 获取安装说明。

> 注意：
>
> - 当前支持：
>   - 本地部署集群
>   - DCS 集群

工作原理

该插件负责以下证书的自动轮换。

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
etcd/peer.crt	etcd 成员间的对等通信证书	控制平面节点
/root/.kube/config, admin.conf, super-admin.conf	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

首先收集所有目标证书的元数据。由于这些证书存储在主机的不同路径，需要从相应文件中读取内容。为此，会在目标节点创建一个临时 Pod，挂载证书目录，使 Pod 能读取信息。证书信息每天收集一次。证书详情（路径、过期时间）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。当剩余有效期少于 20% 或 30 天时，触发轮换。

3. 轮换队列
   需要轮换的证书进入队列等待处理。轮换程序会评估近期的轮换活动和待处理任务的紧急程度，决定是否立即处理，以避免多个证书同时轮换导致集群健康问题。

4. 生成新证书
   轮换程序基于内部存储的 CA 信息生成新证书。轮换过程中会在目标节点创建临时 Pod，挂载必要的证书目录，以便受控地修改文件。

5. 重启组件
   需要重启的组件：

   - `kube-apiserver`：需要重启以加载新证书。重启时会重新生成其内部的 loopback 证书（有效期一年，仅内部使用，无法外部轮换）。
   - `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。

   重启方式： 在相应静态 Pod 的 YAML 文件中添加注解，触发 kubelet 重新创建 Pod。重启 kubelet 时，需挂载主机文件系统并设置 `hostPID` 为 true，在容器内执行 "systemctl restart kubelet"。
   自动重载：

   - Etcd 支持证书自动重载。

6. 轮换时间节点

   - `kubelet` 证书：在 61 天时轮换（证书有效期 91 天）
   - 控制平面证书：在 292 天时轮换（证书有效期 365 天）

运行注意事项

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：

运维人员必须手动更新证书。

运行以下命令手动更新证书：

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

例如更新 `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
```

下载并准备 `cert-renew` 工具，运行：

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

可选地，下载 `renew-all.sh` 脚本以更新节点上的所有证书：

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

cert-manager

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

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 Enhancer 提供对 Kubernetes 集群中使用的证书的监控能力。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. 集群中运行组件的证书，通过检查所有类型为 `kubernetes.io/tls` 的 Secrets 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 **Administrator** 视图中，通过左侧导航进入 **Marketplace** > **Cluster Plugins** 查找并安装 **Cluster Enhancer**。

目录

证书状态监控

证书的过期状态可通过指标 `certificate_expires_status` 查看。证书的过期时间可通过指标 `certificate_expires_time` 查看。

当前证书状态和过期时间可在 **Certificate Status** 子标签页中查看。访问该子标签页的方法是进入 **Administrator** 视图，导航至 **Clusters > Clusters**，选择具体集群后，进入 **Monitoring** 标签页。

内置告警规则

Cluster Enhancer 提供内置告警规则 `cpaas-certificates-rule`，包含以下告警：

Kubernetes 证书告警

规则	等级
kubernetes 证书的过期时间即将到期（少于 30 天） <= 30d 且持续 1 分钟	中等
kubernetes 证书的过期时间即将到期（少于 10 天） <= 10d 且持续 1 分钟	高
kubernetes 证书已过期 <= 0d 且持续 1 分钟	严重

平台组件证书告警

规则	等级
平台组件证书的过期时间即将到期（少于 30 天） <= 30d 且持续 1 分钟	中等
平台组件证书的过期时间即将到期（少于 10 天） <= 10d 且持续 1 分钟	高
平台组件证书已过期 <= 0d 且持续 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

- 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
```