Menu

安全

Alauda Container Security

Alauda Container Security

安全性与合规性

合规

API Refiner

关于 **Alauda Container Platform Compliance Service**

用户与角色

用户

用户组

角色

IDP

用户策略

多租户（项目）

介绍

Project

Namespaces

Clusters、Projects 和 Namespaces 之间的关系

功能指南

审计

介绍

前提条件

操作步骤

搜索结果

遥测

安装

前提条件

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

概述

工作原理

识别 cert-manager 管理的证书

相关资源

OLM 证书

证书监控

证书状态监控

内置告警规则

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护其容器基础设施的安全。

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

Note

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

Menu

安全性与合规性

合规

介绍

安装 **Alauda Container Platform Compliance with Kyverno**

通过控制台安装

通过 YAML 安装

卸载流程

使用指南

API Refiner

介绍

产品介绍

限制

安装 Alauda Container Platform API Refiner

通过控制台安装

通过 YAML 安装

卸载流程

默认配置

关于 Alauda Container Platform Compliance Service

关于 Alauda Container Platform Compliance Service

Menu

合规

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

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

Image Signature Verification Policy

什么是镜像签名验证？

快速开始

常见用例

使用 Secrets 的镜像签名验证策略

为什么使用 Secrets 存储公钥？

快速开始

Secret 创建方式

常见用例

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 Context Enforcement Policy

什么是安全上下文强制执行？

快速开始

核心安全上下文策略

高级场景

测试与验证

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，通过直观的界面展示资源级和策略级的合规违规情况，帮助团队快速识别不合规资源，并采取适当的补救措施，以保持期望的安全态势和法规合规性。

INFO

有关 Kyverno 的更多信息，请参阅 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

通过 YAML 安装

1. 检查可用版本
2. 创建 ModuleInfo

卸载流程

通过控制台安装

1. 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 部署插件

通过 **YAML** 安装

1. 检查可用版本

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

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

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

这表示集群中存在 ModulePlugin `kyverno`，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 for 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 1-3 定位插件
2. 点击 卸载 移除插件

Menu

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

Image Signature Verification Policy

什么是镜像签名验证？

快速开始

常见用例

使用 Secrets 的镜像签名验证策略

为什么使用 Secrets 存储公钥？

快速开始

Secret 创建方式

常见用例

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 Context Enforcement Policy

什么是安全上下文强制执行？

快速开始

核心安全上下文策略

高级场景

测试与验证

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？

快速开始

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

为什么 **Kyverno** 需要访问镜像仓库？

当 Kyverno 需要执行以下操作时，它需要访问镜像仓库：

- 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. 创建镜像仓库凭证

```
# 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** 使用该凭证（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
```

Image Signature Verification Policy

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

目录

什么是镜像签名验证？
快速开始
1. 生成密钥
2. 签名镜像
3. 创建基础验证策略
4. 测试
常见用例
场景 1：多个团队需要签署关键镜像
场景 2：不同环境使用不同规则
场景 3：使用证书代替密钥

什么是镜像签名验证？

镜像签名验证就像门口的保安检查身份证。它确保：

- 镜像真实可信：来源确实是其声称的发布者
- 镜像未被篡改：签名后没有人修改过镜像
- 仅允许可信镜像运行：阻止未签名或签名不正确的镜像

- 审计追踪：记录哪些镜像何时被验证过

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
# 这会生成：cosign.key（私钥，需保密）和 cosign.pub（公钥，可自由分享）
```

2. 签名镜像

```
# 签名镜像（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/N9sbqOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZA==
                -----END PUBLIC KEY-----
        mutateDigest: true  # 将标签转换为安全的摘要格式
```

4. 测试

```bash
# 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：多个团队需要签署关键镜像

对于关键应用，开发团队和安全团队都可能需要对镜像进行签名：

```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:
        # 两个团队都必须签名
        - count: 1  # 安全团队签名
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 安全团队公钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sbqOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZA==
                -----END PUBLIC KEY-----
        - count: 1  # 开发团队签名
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 开发团队公钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917hmKO6JWV
                GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J5==
                -----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  # 未签名时阻止
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                -----BEGIN PUBLIC KEY-----
                # 生产签名密钥
                MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8nXRh950IZbRj8Ra/N9sbqOPZrfM
                5/KAQN0/KjHcorm/J5yctVd7iEcnessRQjU917hmKO6JWVGHpDguIyakZA==
                -----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
              MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyctVd7iEcnessRQjU917hmKO6JWV
              GHpDguIyakZA8nXRh950IZbRj8Ra/N9sbqOPZrfM5/KAQN0/KjHcorm/J5==
              -----END PUBLIC KEY-----
      mutateDigest: true
```

场景 3：使用证书代替密钥

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
```

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: certificate-verification
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: verify-with-certificates
      match:
        any:
        - resources:
            kinds:
            - Pod
      verifyImages:
      - imageReferences:
        - "registry.company.com/*"
        attestors:
        - count: 1
          entries:
          - certificates:
              cert: |-
                -----BEGIN CERTIFICATE-----
                # 公司的签名证书（请替换为真实证书）
                MIIDXTCCAkWgAwIBAgIJAKoK/heBjcOuMA0GCSqGSIb3DQEBBQUAMEUxCzAJBgNV
                BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
                aWRnaXRzIFB0eSBMdGQwHhcNMTcwODI4MTExNzQwWhcNMTgwODI4MTExNzQwWjBF
                MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
                ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
                CgKCAQEAuuExVilGcXIZ3ulNuL7wLrA7VkqJoGpB1YPmYnlS7sobTggOGSqMUvqU
                BdLXcAo3ZCOXuKrBHBlltvcNdFHynfxOtkAOCZjirD6uQBrNPiQDlgMYMy14QIDAQAB
                o1AwTjAdBgNVHQ4EFgQUhKs8VQFhVLp5J4W1sFVLOVgnQxwwHwYDVR0jBBgwFoAU
                hKs8VQFhVLp5J4W1sFVLOVgnQxwwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQUF
                AAOCAQEAuuExVilGcXIZ3ulNuL7wLrA7VkqJoGpB1YPmYnlS7sobTggOGSqMUvqU
                -----END CERTIFICATE-----
              rekor:
                url: https://rekor.sigstore.dev
        mutateDigest: true
```

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，相较于直接在策略中嵌入密钥，提供了更好的安全性和密钥管理。

目录

为什么使用 Secrets 存储公钥？

快速开始

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

Secret 创建方式

方式一：从文件创建

方式二：从字符串字面量创建

方式三：从 YAML 清单创建

常见用例

场景 1：单团队使用一个 Secret

场景 2：多团队使用不同 Secret

场景 3：关键镜像需要多重签名

场景 4：离线环境使用 Secrets

为什么使用 **Secrets** 存储公钥？

使用 Kubernetes Secrets 存储公钥具有以下优势：

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

快速开始

1. 生成并存储密钥到 Secret

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

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

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

2. 为 Kyverno 配置 RBAC

创建 Kyverno 的 Service Account

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

创建访问 Secret 的 Role

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

将 Role 绑定到 Service Account

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

3. 使用 **Secret** 引用创建策略

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

4. 测试配置

```
# 签名镜像
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
```

常见用例

场景 1：单团队使用一个 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:
        # 安全团队签名（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：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: offline-verification-with-secret
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: verify-offline-images
      match:
        any:
        - resources:
            kinds: [Pod, Deployment, StatefulSet, DaemonSet]

      verifyImages:
      - imageReferences:
        - "registry.internal.com/*"
        - "airgap.company.com/*"

        failureAction: Enforce
        emitWarning: false

        attestors:
        - count: 1
          entries:
          - keys:
              secret:
                name: offline-cosign-key
                namespace: kyverno
                key: cosign.pub

              # 离线模式配置
              rekor:
                url: ""                    # 离线模式下为空 URL
                ignoreTlog: true           # 忽略透明日志
                ignoreSCT: true            # 忽略 SCT

              ctlog:
                ignoreTlog: true           # 忽略证书透明日志
                ignoreSCT: true            # 忽略 SCT

        mutateDigest: true
        verifyDigest: true
```

```
        required: true
```

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且可信的仓库的镜像。

目录

什么是镜像仓库验证？

快速开始

1. 阻止除公司仓库外的所有仓库

2. 测试策略

常见场景

场景 1：允许多个可信仓库

场景 2：不同环境不同规则

场景 3：阻止特定风险仓库

场景 4：团队专属仓库访问

高级模式

有效使用通配符

最佳实践

从警告模式开始

排除系统命名空间

常见问题

什么是镜像仓库验证？

仓库验证提供了对镜像来源的集中控制。它能够：

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

快速开始

1. 阻止除公司仓库外的所有仓库

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

2. 测试策略

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

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

# 这条命令会成功（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 Certified 镜像
    - name: production-strict-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "生产环境仅允许 Certified 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 1：使用拒绝列表方式
    - name: block-untrusted-registries
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: "不允许使用来自 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 2：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:*"                     # 来自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  # 从审计模式开始，不阻止
```

排除系统命名空间

```
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. 通配符使用错误：

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

3. **Docker Hub** 格式：

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

容器逃逸防护策略

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

目录

什么是容器逃逸防护？

快速开始

1. 阻止特权容器

2. 测试策略

核心容器逃逸防护策略

策略 1：禁止访问宿主机命名空间

策略 2：禁止宿主机路径挂载

策略 3：禁止宿主机端口

策略 4：禁止危险能力

策略 5：要求非 root 用户运行容器

高级场景

场景 1：环境特定策略

场景 2：工作负载特定例外

测试与验证

测试特权容器

测试宿主机命名空间访问

测试宿主机路径挂载

测试安全合规容器

最佳实践

1. 从审计模式开始

2. 排除系统命名空间

什么是容器逃逸防护？

容器逃逸防护涉及检测并阻止可能允许攻击者突破容器隔离、访问宿主机系统的危险容器配置。包括：

- 特权容器：以提升权限运行的容器
- 宿主机命名空间访问：容器共享宿主机的 PID、网络或 IPC 命名空间
- 宿主机路径挂载：容器挂载宿主机文件系统路径
- 危险的能力：容器拥有过多的 Linux 能力
- 宿主机端口访问：容器绑定宿主机网络端口

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 allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: privileged-containers
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Privileged mode is disallowed. The fields
spec.containers[*].securityContext.privileged,
          spec.initContainers[*].securityContext.privileged, and
spec.ephemeralContainers[*].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

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

核心容器逃逸防护策略

策略 1：禁止访问宿主机命名空间

防止容器访问宿主机命名空间：

```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 used 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.hostNetwork,
          spec.hostIPC, and spec.hostPID must be unset or set to false.
        pattern:
          spec:
            =(hostPID): "false"
            =(hostIPC): "false"
            =(hostNetwork): "false"
```

策略 2：禁止宿主机路径挂载

阻止容器挂载宿主机文件系统路径：

```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 containers.
      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[*].hostPath must be
unset.
        pattern:
          spec:
            =(volumes):
              - X(hostPath): "null"
```

策略 3：禁止宿主机端口

防止容器绑定宿主机网络端口：

```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 and should not be
      allowed, or at minimum restricted to a known list.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-ports-none
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host ports is disallowed. The fields
spec.containers[*].ports[*].hostPort,
          spec.initContainers[*].ports[*].hostPort, and
spec.ephemeralContainers[*].ports[*].hostPort
          must either be unset or set to 0.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(ports):
                  - =(hostPort): 0
            =(initContainers):
              - =(ports):
                  - =(hostPort): 0
            containers:
              - =(ports):
                  - =(hostPort): 0
```

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 disallowed.
    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 runAsNonRoot is set to
true.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: run-as-non-root
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Running as root is not allowed. Either the field
spec.securityContext.runAsNonRoot
          must be set to true, or the field
spec.containers[*].securityContext.runAsNonRoot
          must be set to true.
        anyPattern:
        - spec:
            securityContext:
              runAsNonRoot: "true"
        - spec:
            containers:
            - securityContext:
                runAsNonRoot: "true"
```

高级场景

场景 1：环境特定策略

针对不同环境设置不同的安全级别：

```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 security"
        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 security"
        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 workloads"
        pattern:
          spec:
            =(hostNetwork): "false"
            containers:
            - securityContext:
                =(privileged): "false"
```

测试与验证

测试特权容器

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

测试宿主机命名空间访问

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

测试宿主机路径挂载

```
# 此操作应被阻止
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
```

测试安全合规容器

```
# 此操作应被允许
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  # 先以警告形式提示，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 以强制执行容器的安全上下文，确保容器以适当的安全设置和限制运行。

目录

什么是安全上下文强制执行？

快速开始

1. 要求非 Root 容器策略

2. 测试策略

核心安全上下文策略

策略 1：禁止权限提升

策略 2：要求特定用户 ID 范围

策略 3：要求非 Root 组

策略 4：限制 Seccomp 配置

策略 5：要求丢弃所有能力

策略 6：限制 AppArmor 配置

高级场景

场景 1：环境特定的安全上下文

场景 2：应用特定的安全上下文

场景 3：分级安全上下文强制执行

测试与验证

测试 Root 容器（应失败）

测试权限提升（应失败）

测试缺少丢弃所有能力（应失败）

测试有效的安全容器（应通过）

什么是安全上下文强制执行？

安全上下文强制执行涉及通过设置与安全相关的参数来控制容器的运行方式。正确配置安全上下文可以防止：

- **Root** 权限提升：容器以 root 用户身份运行
- 权限提升攻击：容器获得提升的权限
- 不安全的进程执行：容器运行具有危险能力
- 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 runAsNonRoot is set to
true.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: run-as-non-root
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Running as root is not allowed. Either the field
spec.securityContext.runAsNonRoot
          must be set to true, or the field
spec.containers[*].securityContext.runAsNonRoot
          must be set to true.
        anyPattern:
        - spec:
            securityContext:
              runAsNonRoot: "true"
        - spec:
            containers:
            - securityContext:
                runAsNonRoot: "true"
```

2. 测试策略

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

# 尝试创建明确以 root 身份运行的容器（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：

```
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.allowPrivilegeEscalation,
          and spec.ephemeralContainers[*].securityContext.allowPrivilegeEscalation
          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 privilege escalation.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: user-id-range
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Containers must run with user ID between 1000 and 65535.
        deny:
          conditions:
            any:
            # 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.runAsUser &&
(securityContext.runAsUser < `1000` || securityContext.runAsUser > `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 supplemental
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.securityContext.runAsGroup
          or spec.containers[*].securityContext.runAsGroup must be set and not be 0.
        deny:
          conditions:
            any:
            # 检查 Pod 级别的 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.runAsGroup == `0`]
| length(@) }}"
              operator: GreaterThan
              value: 0
```

策略 4：限制 Seccomp 配置

强制使用安全的 seccomp 配置：

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

```
            - securityContext:
                seccompProfile:
                  type: Localhost
```

策略 5：要求丢弃所有能力

确保容器丢弃所有能力：

```
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

控制 AppArmor 配置的使用：

```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

不同环境的安全需求不同：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: environment-security-contexts
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    # 生产环境：严格的安全上下文
    - name: production-strict-security
      match:
        any:
        - resources:
            kinds:
            - Pod
            namespaces:
            - production
            - prod-*
      validate:
        message: "Production environments require strict security contexts"
        pattern:
          spec:
            securityContext:
              runAsNonRoot: "true"
              runAsUser: "1000-65535"
              runAsGroup: "1000-65535"
              seccompProfile:
                type: RuntimeDefault
            containers:
            - securityContext:
                allowPrivilegeEscalation: "false"
                readOnlyRootFilesystem: "true"
                runAsNonRoot: "true"
                capabilities:
                  drop:
                  - ALL

    # 开发环境：基本安全要求
    - name: development-basic-security
      match:
        any:
        - resources:
            kinds:
```

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

场景 2：应用特定的安全上下文

不同应用类型的安全上下文不同：

```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 contexts"
        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：分级安全上下文强制执行

实现渐进式的安全上下文要求：

```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 contexts"
        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
```

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
```

测试缺少丢弃所有能力（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
```

测试有效的安全容器（应通过）

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

网络安全策略

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

目录

什么是网络安全？

快速开始

1. 禁止主机网络访问

2. 测试策略

核心网络安全策略

策略 1：禁止主机端口

策略 2：限制主机端口范围

策略 3：要求网络策略

策略 4：限制 Service 类型

策略 5：控制 Ingress 配置

策略 6：限制 DNS 配置

高级场景

场景 1：环境特定的网络策略

场景 2：应用特定的网络策略

场景 3：网络分段强制执行

测试与验证

测试主机网络访问（应失败）

测试主机端口绑定（应失败）

测试 NodePort Service（应失败）

测试有效的网络配置（应通过）

什么是网络安全？

网络安全涉及控制容器如何访问和交互网络资源。适当的网络安全可以防止：

- 主机网络访问：容器访问主机网络接口
- 通过网络的权限提升：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 traffic and should
not be allowed.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-network
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host network is disallowed. The field spec.hostNetwork must be unset or
set to false.
        pattern:
          spec:
            =(hostNetwork): "false"
```

2. 测试策略

```bash
# 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":{"hostNetwork":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 and should not be
      allowed, or at minimum restricted to a known list.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-ports-none
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Use of host ports is disallowed. The fields
spec.containers[*].ports[*].hostPort,
          spec.initContainers[*].ports[*].hostPort, and
spec.ephemeralContainers[*].ports[*].hostPort
          must either be unset or set to 0.
        pattern:
          spec:
            =(ephemeralContainers):
              - =(ports):
                  - =(hostPort): 0
            =(initContainers):
              - =(ports):
                  - =(hostPort): 0
            containers:
              - =(ports):
                  - =(hostPort): 0
```

策略 2：限制主机端口范围

允许特定的主机端口范围以实现受控访问：

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: restrict-host-port-range
  annotations:
    policies.kyverno.io/title: Restrict Host Port Range
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Host ports, if used, must be within an allowed range to prevent conflicts and
security issues.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: host-port-range
      match:
        any:
        - resources:
            kinds:
            - Pod
      preconditions:
        all:
        - key: "{{ request.object.spec.containers[].ports[?hostPort] | length(@) }}"
          operator: GreaterThan
          value: 0
      validate:
        message: >-
          Host ports must be within the allowed range 30000-32767.
        foreach:
        - list: request.object.spec.[ephemeralContainers, initContainers, containers]
[].ports[]
          preconditions:
            any:
            - key: "{{ element.hostPort }}"
              operator: GreaterThan
              value: 0
          deny:
            conditions:
              any:
              - key: "{{ element.hostPort }}"
                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 traffic.
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: require-netpol
      match:
        any:
        - resources:
            kinds:
            - Pod
      exclude:
        any:
        - resources:
            namespaces:
            - kube-system
            - kyverno
      context:
      - name: netpols
        apiCall:
          urlPath: "/apis/networking.k8s.io/v1/namespaces/{{ request.namespace
}}/networkpolicies"
          jmesPath: "items[?spec.podSelector.matchLabels.app == '{{
request.object.metadata.labels.app }}'] | length(@)"
      validate:
        message: >-
          Pods must have an associated NetworkPolicy. Create a NetworkPolicy that selects
this pod.
        deny:
          conditions:
            all:
            - key: "{{ netpols }}"
              operator: Equals
```

```
            value: 0
```

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 annotations.
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 HSTS.
        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 exfiltration.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-dns-policy
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Custom DNS policy is not allowed. Use Default or ClusterFirst only.
        pattern:
          spec:
            =(dnsPolicy): "Default | ClusterFirst"
    - name: restrict-custom-dns
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Custom DNS configuration is not allowed in production environments.
        pattern:
          spec:
            X(dnsConfig): "null"
```

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 security"
        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 security"
        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 ports"
        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 ports"
        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 Service**（应失败）

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

测试有效的网络配置（应通过）

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

Volume Security Policy

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

目录

什么是卷安全？

快速开始

1. 限制卷类型
2. 测试策略

核心卷安全策略

策略 1：禁止 HostPath 卷

策略 2：限制 HostPath 卷（只读访问）

策略 3：禁止特权卷类型

策略 4：要求只读根文件系统

策略 5：控制卷挂载权限

高级场景

场景 1：环境特定的卷策略

场景 2：应用特定的卷策略

场景 3：卷大小和资源限制

测试与验证

测试 HostPath 卷（应失败）

什么是卷安全？

卷安全涉及控制容器可以挂载的卷类型及其访问方式。适当的卷安全可以防止：

- 主机文件系统访问：未经授权访问主机目录
- 权限提升：通过卷获取提升的权限
- 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 configMap, csi,
      downwardAPI, emptyDir, ephemeral, persistentVolumeClaim, projected, and secret.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
    - name: restrict-volume-types
      match:
        any:
        - resources:
            kinds:
            - Pod
      validate:
        message: >-
          Only the following types of volumes may be used: configMap, csi, downwardAPI,
          emptyDir, ephemeral, persistentVolumeClaim, projected, and secret.
        foreach:
        - list: "request.object.spec.volumes || []"
          deny:
            conditions:
              all:
              - key: "{{ element.keys(@) }}"
                operator: AnyNotIn
                value:
                - name
                - configMap
                - csi
                - downwardAPI
                - emptyDir
                - ephemeral
                - persistentVolumeClaim
                - projected
                - secret
```

2. 测试策略

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

# 尝试创建带有 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 containers.
      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[*].hostPath 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 HostPath 卷且必须为只读：

```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 paths.
        foreach:
        - list: "request.object.spec.volumes[?hostPath]"
          deny:
            conditions:
              any:
              # 如果路径不在允许列表则拒绝
              - 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

阻止绕过安全控制的特权卷类型：

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

```
    - flexVolume
    - cinder
    - cephFS
    - flocker
    - fc
    - azureFile
    - azureDisk
    - vsphereVolume
    - quobyte
    - portworxVolume
    - scaleIO
    - storageos
```

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 infrastructure strategy;
      the container only needs to write on the mounted volume that persists 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 to 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：限制为安全的卷类型
    - 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
```

Menu

API Refiner

介绍

产品介绍

限制

安装 Alauda Container Platform API Refiner

通过控制台安装

通过 YAML 安装

卸载流程

默认配置

介绍

目录

产品介绍

限制

产品介绍

ACP API Refiner 是由 Alauda Container Platform 提供的数据过滤服务，旨在增强 Kubernetes 环境中的多租户安全性和数据隔离。它根据用户权限、项目、集群和命名空间过滤 Kubernetes API 响应数据，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，并支持 API 响应中的字段排除、包含和脱敏。

目录

通过控制台安装

通过 YAML 安装

1. 检查可用版本
2. 创建 ModuleInfo

卸载流程

默认配置

过滤的资源

字段脱敏

通过控制台安装

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

通过 YAML 安装

1. 检查可用版本

确保插件已发布，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，无需任何配置参数：

```
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  annotations:
    cpaas.io/display-name: apirefiner
    cpaas.io/module-name: '{"en": "Alauda Container Platform API Refiner", "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`：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`；缺少此字段会导致安装失败。
- `.spec.config`：如果对应的 ModuleConfig 为空，此字段可留空。
- `.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

过滤的资源

默认过滤以下资源：

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

字段脱敏

默认脱敏以下字段：

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

关于 Alauda Container Platform 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。

Menu

用户与角色

用户

介绍

用户来源

用户管理规则

用户生命周期

功能指南

用户组

介绍

组介绍

组类型

功能指南

角色

介绍

角色介绍

系统角色

自定义角色

功能指南

IDP

介绍

概述

支持的集成方式

功能指南

故障排除

用户策略

介绍

概述

配置安全策略

可用策略

Menu

用户

介绍

介绍

用户来源

用户管理规则

用户生命周期

功能指南

管理用户角色

添加角色

移除角色

创建用户

步骤

用户管理

重置本地用户密码

更新用户到期日期

激活用户

禁用用户

将用户添加到本地用户组

删除用户

批量操作

介绍

该平台支持所有用户的身份验证和登录验证。

目录

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

用户来源

本地用户

- 在平台部署期间创建的管理员帐户
- 通过平台界面创建的帐户
- 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（身份提供者）集成导入的帐户
- 来源显示为 IDP 配置名称
- 通过 IDP 设置进行集成配置

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
- 通过 IDP 设置进行集成配置

> **WARNING**
>
> 对于在第一次登录之前已添加到项目中的 OIDC 用户：
>
> - 来源在成功登录平台之前显示为 "-"
> - 成功登录后，来源更改为 IDP 配置名称

其他第三方用户

- 通过支持的 dex 连接器（如 GitHub、Microsoft）进行身份验证的用户
- 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
> - 拥有相同用户名的第三方用户（OIDC/LDAP）将自动关联
> - 关联用户从现有帐户继承权限

- 用户可以通过各自的来源登录
- 平台中每个用户名只显示一个用户记录
- 用户来源由最近的登录方式决定

用户生命周期

下表描述了平台上用户的不同状态：

状态	描述
正常	用户帐户处于活动状态，可以登录平台
禁用	用户帐户处于非活动状态，无法登录。请联系平台管理员以重新激活。 可能原因： - 连续 90 天未登录 - 帐户过期 - 管理员手动禁用
锁定	由于 24 小时内五次登录失败，帐户暂时被锁定。 详细信息： - 锁定时长：20 分钟 - 可以由管理员手动解锁 - 锁定期结束后帐户将恢复可用
无效	已从 LDAP 服务器删除的 LDAP 同步帐户。 注意：无效帐户无法登录平台

Menu

功能指南

管理用户角色

添加角色

移除角色

创建用户

步骤

用户管理

重置本地用户密码

更新用户到期日期

激活用户

禁用用户

将用户添加到本地用户组

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（而不是他们自己的帐户）的角色，以授予或撤销权限。

目录

添加角色
步骤
移除角色
步骤

添加角色

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户的用户名
3. 滚动到 角色列表 部分
4. 点击 添加角色
5. 在角色分配对话框中：
   - 从 角色名称 下拉菜单中选择一个角色
   - 选择角色的权限范围（集群、项目或命名空间）
   - 点击 添加

NOTE

重要说明：

- 您可以为用户添加多个角色
- 每个角色每个用户只能添加一次
- 已分配的角色在下拉菜单中显示，但无法被选择
- 集群管理员 角色无法在 global 集群 中分配

移除角色

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户的用户名
3. 滚动到 角色列表 部分
4. 点击要移除的角色旁边的 移除
5. 确认移除

WARNING

角色管理权限：

- 只有平台管理员可以管理其他用户的角色
- 用户无法修改自己的帐户角色

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

步骤

步骤

1. 在左侧导航栏中，单击 用户 > 用户管理
2. 单击 创建用户
3. 配置以下参数：

参数	描述
密码类型	选择一种密码生成方法： 随机：系统生成一个安全的随机密码 自定义：用户手动输入密码
密码	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： - 单击眼睛图标以显示/隐藏密码 - 单击复制图标以复制密码
邮箱	用户的电子邮件地址： - 必须是唯一的 - 可以用作登录用户名 - 与用户的名称关联
有效期	设置用户帐户的有效期： 选项： - 永久：无限制 - 自定义：使用时间范围下拉菜单设置开始和结束时间
角色	为用户分配一个或多个角色
继续创建	切换开关以控制创建后行为： - 开启：重定向到新用户创建页面 - 关闭：显示用户详细信息页面

4. 单击 创建

NOTE

用户创建成功后：

- 如果“继续创建”启用，将重定向到创建另一个用户的页面
- 如果禁用，则会显示创建用户的详细信息页面

用户管理

该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，以提高特定场景（例如，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，地方来源）
- 当前登录用户无法管理自己的账户
- 对于个人账户修改（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

步骤

更新用户到期日期

步骤

激活用户

步骤

禁用用户

步骤

将用户添加到本地用户组

步骤

删除用户

步骤

批量操作

步骤

重置本地用户密码

拥有平台管理权限的用户可以重置其他本地用户的密码。

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户记录旁边的图标
3. 点击 重置密码
4. 在对话框中，选择密码类型：
   - 随机：系统生成一个安全的随机密码
   - 自定义：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 重置

更新用户到期日期

您可以更新 正常、禁用 或 锁定 状态用户的到期日期。超过到期日期的用户将被自动禁用。

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户旁边的 更新到期日期
3. 在对话框中，选择到期日期选项：
   - 永久：无限制
   - 自定义：使用时间范围下拉框设置开始和结束时间
4. 点击 更新

激活用户

您可以激活 禁用 或 锁定 状态的用户。

> **NOTE**
>
> 激活行为：
>
> - 如果用户在到期日期内：到期日期保持不变
> - 如果用户已过期：到期日期变为 永久

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户旁边的 激活
3. 在确认对话框中点击 激活
4. 用户状态将变为 正常

禁用用户

您可以在用户到期日期内禁用 正常 或 锁定 状态的用户。被禁用的用户无法登录，但可以重新激活。

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户旁边的图标
3. 点击 禁用 并确认

将用户添加到本地用户组

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

WARNING

组角色行为：

- 用户自动继承其组的角色
- 组角色仅在组的详细信息页面（配置角色选项卡）上可见
- 单个用户的角色列表仅显示直接分配的角色

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户旁边的图标
3. 点击 添加到用户组
4. 选择一个或多个本地用户组
5. 点击 添加

删除用户

平台管理员可以删除任何用户，但当前登录的账户除外，包括：

- IDP 配置的用户
- 源为 `-` 的用户
- 本地用户

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 点击目标用户旁边的图标
3. 点击 删除
4. 点击 确认

批量操作

您可以执行以下批量操作：

- 更新有效期
- 激活用户
- 禁用用户
- 删除用户

步骤

1. 在左侧导航栏中，点击 用户 > 用户管理
2. 使用复选框选择一个或多个用户
3. 点击 批量操作 并选择一个操作：
    - 更新有效性
    - 激活

- 禁用
- 删除

NOTE

批量操作详细信息：

- 更新有效性：设置永久或自定义时间范围
- 激活：在对话框中确认激活
- 禁用：在对话框中确认禁用
- 删除：输入当前账户密码并确认

Menu

用户组

介绍

介绍

组介绍

组类型

功能指南

管理用户组角色

向组中添加角色

从组中移除角色

创建本地用户组

创建用户组

管理用户组

管理本地用户组成员资格

前提条件

导入成员

移除成员

介绍

目录

组介绍
组类型
本地用户组
IDP 同步用户组

组介绍

该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：

- 同时向多个用户授予平台操作权限
- 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，当企业内部发生人事变动，并且您需要向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. 为该组配置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

该平台支持两种类型的组：

本地用户组

- 直接在平台上创建
- 源显示为 本地
- 可以更新或删除
- 支持：
  - 从任何来源添加或移除用户
  - 添加或移除角色

IDP 同步用户组

- 从连接的 IDP（LDAP、Azure AD）同步而来
- 源显示为连接的 ***IDP*** 名称
- 不能更新或删除
- 支持：
  - 添加或移除角色
  - 不能管理组成员（添加或移除）

Menu

功能指南

管理用户组角色

向组中添加角色

从组中移除角色

创建本地用户组

创建用户组

管理用户组

管理本地用户组成员资格

前提条件

导入成员

移除成员

管理用户组角色

拥有平台管理权限的用户可以管理本地用户组和与 IDP 同步的用户组的角色。

目录

向组中添加角色
步骤
从组中移除角色
步骤

向组中添加角色

步骤

1. 在左侧导航栏中，单击 用户 > 用户组管理
2. 点击目标用户组的名称
3. 在 配置角色 选项卡上，点击 添加角色
4. 点击以添加角色

NOTE

角色分配规则：

- 您可以向一个组添加多个角色

- 每个角色只能在同一组中添加一次

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 添加

从组中移除角色

WARNING

当您从组中移除角色时：

- 该角色授予组成员的所有权限将被撤销
- 此操作无法撤销

步骤

1. 在左侧导航栏中，单击 用户 > 用户组管理
2. 点击目标用户组的名称
3. 在 配置角色 选项卡上，点击角色旁边的 移除
4. 点击 确认 以移除角色

创建本地用户组

本地用户组允许您实现基于角色的访问控制，以便于管理来自任意来源的多个用户。

目录

创建用户组
步骤
管理用户组

创建用户组

步骤

1. 在左侧边栏中，单击 用户 > 用户组管理
2. 点击 创建用户组
3. 输入以下信息：
   - 名称：用户组的名称
   - 描述：对该组目的的描述
4. 点击 创建

管理用户组

您可以通过点击列表页面上的图标或在详细信息页面右上角点击 操作 来管理用户组。

操作	描述
更新用户组	根据组的来源更新组信息： - 对于 来源 为 `Local` 的组：可以更新名称和描述 - 对于 来源 为 `IDP name` 的组：只能更新描述
删除本地用户组	删除 来源 为 `Local` 的用户组

WARNING

删除组时：

- 所有组成员将被移除
- 所有分配给该组的角色将被移除
- 此操作无法撤销

管理本地用户组成员资格

仅具有平台管理权限的用户可以管理本地用户组成员资格。

目录

前提条件
导入成员
步骤
移除成员
步骤

前提条件

WARNING

在管理组成员资格之前，请注意以下限制：

- 仅具有平台管理权限的用户可以管理组及其成员
- 系统帐户和当前已登录帐户无法管理（不能导入到组中或从组中删除）
- 每个本地用户组最多可以拥有 5000 名成员
- 当组达到 5000 名成员的限制时，将不允许进一步导入

导入成员

您可以将平台中的用户导入本地用户组，以统一管理权限。

TIP

导入到组中的用户将自动继承分配给该组的所有操作权限。

步骤

1. 在左侧导航栏中，单击 用户 > 用户组管理
2. 单击要添加成员的本地用户组的名称
3. 在 组成员管理 标签下，单击 导入成员
4. 通过勾选用户名/显示名旁边的复选框选择一个或多个用户
5. 单击 导入

NOTE

- 只能选择当前不是该组成员的用户
- 使用 全部导入 按钮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，通过该组授予该用户的所有操作权限将被自动撤销。

步骤

1. 在左侧导航栏中，单击 用户 > 用户组管理
2. 单击要移除成员的本地用户组的名称
3. 在 组成员管理 标签下，您可以通过两种方式移除成员：

- 单击成员名称旁的 移除 并确认
- 使用复选框选择一个或多个成员，然后单击 批量移除 并确认

Menu

角色

介绍

介绍

角色介绍

系统角色

自定义角色

功能指南

创建角色

基本信息配置

查看配置

权限配置

管理自定义角色

更新基本信息

更新角色权限

复制现有角色

删除自定义角色

介绍

目录

角色介绍
系统角色
自定义角色

角色介绍

该平台的用户角色管理是通过 Kubernetes RBAC（基于角色的访问控制）实现的。此系统通过将角色与用户关联，实现了灵活的权限配置。

角色表示一组在平台上操作 Kubernetes 资源所需的权限。这些权限包括：

- 创建资源
- 查看资源
- 更新资源
- 删除资源

角色对不同资源进行分类和组合权限。通过将角色分配给用户并设置权限范围，可以快速授予资源操作权限。

权限也可以同样简单地通过从用户中移除角色来撤销。

一个角色可以包含：

- 一个或多个资源类型
- 一个或多个操作权限
- 多个分配给它的用户

例如：

- 角色 A：只能查看和创建项目
- 角色 B：可以创建、查看、更新和删除用户、项目和命名空间

系统角色

为了满足常见的权限配置场景，该平台提供以下默认系统角色。这些角色使平台资源的访问控制灵活，并为用户提供高效的权限管理。

角色名称	描述	角色级别
平台管理员	拥有对平台上所有业务和资源的完全访问权限	平台
平台审计员	可以查看所有平台资源和操作记录，但没有其他权限	平台
集群管理员（**Alpha**）	管理和维护集群资源，对所有集群级别的资源具有完全访问权限	集群
项目管理员	管理命名空间管理员和命名空间配额	项目
namespace-admin-system	管理命名空间成员和角色分配	命名空间
开发者	在命名空间内开发、部署和维护自定义应用程序	命名空间

自定义角色

该平台支持自定义角色，以增强资源访问控制场景。自定义角色相较于系统角色提供了若干优势：

- 灵活的权限配置
- 更新角色权限的能力
- 可在不再需要时删除角色的选项

WARNING

更新或删除自定义角色时请谨慎。当删除自定义角色时，将自动撤销该角色授予所有绑定用户的权限。

Menu

功能指南

创建角色

基本信息配置

查看配置

权限配置

管理自定义角色

更新基本信息

更新角色权限

复制现有角色

删除自定义角色

创建角色

具有平台角色权限的用户可以根据实际使用场景创建其权限等于或低于自身角色权限的自定义角色。在创建角色时，可以配置：

- 平台功能模块操作权限
- 用户自定义资源的访问权限（Kubernetes CRD）

目录

基本信息配置
角色类型
查看配置
权限配置

基本信息配置

1. 在左侧导航栏中，点击 用户 > 角色。
2. 点击 创建角色。
3. 配置角色的基本信息：

角色类型

为用户分配角色时，权限范围将根据角色类型进行限制：

- 平台角色：显示所有平台权限
- 项目角色：显示以下权限：
  - 项目管理
  - 容器平台
  - 服务网格
  - DevOps
  - 中间件
- 命名空间角色：显示以下权限：
  - 项目管理
  - 容器平台
  - 服务网格
  - DevOps
  - 中间件

4. 点击 下一步。

查看配置

在查看配置部分，您可以控制角色访问特定视图的权限。未被选择的视图将不会在具有此角色的用户的顶部导航中显示。

NOTE

1. 您账户的角色权限限制了您可以配置哪些视图卡片。例如：
   - 如果您的账户没有 项目管理 视图权限
   - 在创建角色时， 项目管理 视图卡片将显示为灰色
   - 您只能创建与自己角色权限相等或更低的角色
2. 视图入口状态：
   - 如果某个视图的 显示入口 在 产品 功能中关闭

- 则该视图在 权限配置 中的权限仍将生效
- 该视图在入口启用之前将暂时无法访问
- 一旦启用，之前选择的权限将正常工作

权限配置

1. 点击页面左上角的 添加自定义权限。
2. 配置角色操作自定义资源（Kubernetes CRD）的权限：

参数	描述
组名称	权限组的名称。组在权限模块下按照添加的顺序显示。
资源名称	资源的名称。按 Enter 键添加多个自定义资源名称。
操作权限	操作该资源的权限。

3. 点击 创建。

管理自定义角色

本指南描述了如何在平台上管理自定义角色，包括：

- 更新基本信息和权限
- 复制现有角色以创建新角色
- 删除自定义角色

目录

更新基本信息

步骤

更新角色权限

步骤

复制现有角色

步骤

删除自定义角色

步骤

更新基本信息

您可以更新平台上自定义角色的显示名称和描述。

步骤

1. 在左侧导航栏中，点击 用户 > 角色
2. 点击要更新的 *角色名称*
3. 点击右上角的 操作 > 更新
4. 更新角色的：
   - 显示名称
   - 描述
5. 点击 更新

更新角色权限

您可以更新自定义角色的权限信息，包括：

- 为平台资源添加新的操作权限
- 删除现有权限
- 修改自定义资源的权限

步骤

1. 在左侧导航栏中，点击 用户 > 角色
2. 点击要更新的 *角色名称*
3. 点击权限区域右上角的 操作 > 更新角色权限
4. 在 更新角色权限 页面上进行更改
5. 点击 确认

复制现有角色

您可以通过复制现有角色（系统角色或自定义角色）来创建新角色。新角色将继承源角色的所有权限信息，您可以根据需要进行修改。

WARNING

新角色的权限不能超过创建者所属角色的权限。

步骤

1. 在左侧导航栏中，点击 用户 > 角色
2. 点击要复制的 *角色名称*
3. 点击右上角的 操作 > 复制为新角色
4. 在 复制为新角色 页面上，配置：
   - 名称
   - 显示名称
   - 描述
   - 类型
5. 点击 创建

删除自定义角色

您可以删除不再使用的自定义角色。

WARNING

当您删除自定义角色时：

- 该角色与用户的绑定关系将被移除
- 分配给该角色的用户将失去该角色所授予的所有权限
- 该角色将从用户的角色列表中移除

步骤

1. 在左侧导航栏中，点击 用户 > 角色
2. 点击要删除的 *角色名称*
3. 点击右上角的 操作 > 删除
4. 输入角色名称以确认删除
5. 点击 删除

Menu

IDP

介绍

介绍

概述

支持的集成方式

功能指南

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

概述
支持的集成方式
LDAP 集成
OIDC 集成

概述

该平台与 Dex 身份验证服务集成，使您能够通过 IDP 配置使用 Dex 预先实现的连接器进行平台账户认证。如需更多信息，请参阅 Dex 官方文档↗。

支持的集成方式

LDAP 集成

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，以连接到您企业的 LDAP 服务器。

LDAP 集成的好处：

- 启用平台与 LDAP 服务器之间的通信

- 允许企业用户使用 LDAP 凭据登录
- 自动将企业用户账户同步到平台

OIDC 集成

该平台支持使用 OpenID Connect（OIDC）协议与 IDP 服务集成，以进行第三方用户认证。

OIDC 集成的好处：

- 使用户能够使用第三方账户登录
- 支持企业 IDP 服务
- 通过 OIDC 协议提供安全认证

> **NOTE**
>
> 对于使用上述未提到的其他连接器进行认证，请联系技术支持。

Menu

功能指南

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

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

支持的 LDAP 类型

- OpenLDAP
- Active Directory

LDAP 术语

- OpenLDAP 常用术语
- Active Directory 常用术语

添加 LDAP

- 前提条件
- 步骤
  - 基本信息
  - 搜索设置

LDAP 配置示例

- LDAP 连接器配置
- 用户过滤器示例
- 组搜索配置示例
- AND(&) 和 OR(|) 运算符在 LDAP 筛选器中的示例

同步 LDAP 用户

- 操作程序

相关操作

LDAP 概述

LDAP（轻量级目录访问协议）是一种成熟、灵活且得到良好支持的标准机制，用于与目录服务器进行交互。它将数据组织成层次树结构，以存储企业用户和组织信息，主要用于实现单点登录（SSO）。

NOTE

LDAP 关键特性：

- 启用客户端与 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

有关更多信息，请参阅 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 并配置相关参数来使平台与 LDAP 服务进行通信。

NOTE

OpenLDAP 集成：

- 启用平台对 LDAP 用户的认证
- 支持标准 LDAP 协议
- 提供灵活的用户管理

有关 OpenLDAP 的更多信息，请参阅 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 服务进行通信。

NOTE

Active Directory 集成：

- 启用平台对 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 (域组件)	域组件	`dc=example,dc=com`
ou (组织单位)	组织单位	`ou=People,dc=example,dc=com`
cn (常用名)	常用名	`cn=admin,dc=example,dc=com`
uid (用户 ID)	用户 ID	`uid=example`
objectClass (对象类)	对象类	`objectClass=inetOrgPerson`
mail (邮件)	邮件	`mail=example@126.com`
givenName (名)	名	`givenName=xq`

术语	描述	示例
sn (姓)	姓	`sn=ren`
objectClass: groupOfNames	用户组	`objectClass: groupOfNames`
member (成员)	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 (域组件)	域组件	`dc=example,dc=com`
ou (组织单位)	组织单位	`ou=People,dc=example,dc=com`
cn (常用名)	常用名	`cn=admin,dc=example,dc=com`
sAMAccountName/userPrincipalName	用户标识符	`userPrincipalName=example` 或 `sAMAccountName=example`
objectClass: user	AD 用户	`objectClass=user`

术语	描述	示例
	对象类	
mail (邮件)	邮件	`mail=example@126.com`
displayName	显示名称	`displayName=example`
givenName (名)	名	`givenName=xq`
sn (姓)	姓	`sn=ren`
objectClass: group	用户组	`objectClass: group`
member (成员)	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：

- 用户可以使用其企业账户登录平台
- 多次添加相同的 LDAP 将覆盖先前同步的用户

前提条件

在添加 LDAP 之前，请准备以下信息：

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

步骤

1. 在左侧导航栏中，点击 用户 > 身份提供者（**IDP**）
2. 点击 添加 **LDAP**
3. 配置以下参数：

基本信息

参数	描述
服务器地址	LDAP 服务器访问地址（例如，`192.168.156.141:31758`）
用户名	LDAP 管理员 DN（例如，`cn=admin,dc=example,dc=com`）
密码	LDAP 管理员账户密码
登录框用户名提示	用户名输入提示消息（例如，“请输入您的用户名”）

搜索设置

NOTE

搜索设置目的：

- 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
对象类型	用户的 ObjectClass： - OpenLDAP: `inetOrgPerson` - Active Directory: `organizationalPerson` - 组: `posixGroup`
登录字段	用作登录用户名的属性： - OpenLDAP: `mail`（电子邮件地址） - Active Directory: `userPrincipalName`
筛选条件	用户/组过滤的 LDAP 筛选条件 示例: `(&(cn=John*)(givenName=*xq*))`
搜索起始点	用户/组搜索的基本 DN（例如，`dc=example,dc=org`）
搜索范围	搜索范围： - `sub`: 整个目录子树 - `one`: 从起始点向下一级
登录属性	唯一用户标识符： - OpenLDAP: `uid` - Active Directory: `distinguishedName`
名称属性	对象名称属性（默认: `cn`）
电子邮件属性	电子邮件属性： - OpenLDAP: `mail` - Active Directory: `userPrincipalName`

参数	描述
组成员属性	组成员标识符（默认: `uid`）
组属性	用户组关系属性（默认: `memberuid`）

4. 在 身份提供者（**IDP**）服务配置验证 部分：

- 输入有效的 LDAP 账户用户名和密码
- 用户名必须与 登录字段 设置匹配
- 点击验证配置

5. （可选）配置 **LDAP** 自动同步策略：

- 启用 自动同步用户 开关
- 设置同步规则
- 使用 在线工具↗ 验证 CRON 表达式

6. 点击 添加

NOTE

添加 LDAP 后：

- 用户可以在同步之前登录
- 用户信息将在首次登录时自动同步
- 自动同步将根据配置的规则进行

LDAP 配置示例

LDAP 连接器配置

以下示例展示如何配置一个 LDAP 连接器：

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

    # 登录账户提示。例如，用户名
    usernamePrompt: SSO Username

    # 用户搜索配置
    userSearch:
      # 从基本 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。映射到电子邮件的属性
      emailAttr: mail
      # 必填。映射到用户名的属性
      nameAttr: cn
      # 登录用户名属性
      # 筛选条件将转换为 "(<attr>=<username>)"，如 (uid=example)。
      username: uid

      # 扩展属性
      # phoneAttr: phone
```

```
    # 组搜索配置
    groupSearch:
      # 从基本 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
```

用户过滤器示例

```
# 1. 基本过滤器：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. 查找特定电子邮件域的用户
(&(objectClass=person)(mail=*@example.com))

# 5. 查找特定组的成员
(&(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. 排除系统账户
(&(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. 基本过滤器：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 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)
)
```

AND(&) 和 OR(|) 运算符在 LDAP 筛选器中的示例

```
# AND 运算符 (&) - 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 运算符 (|) - 至少一个条件必须满足
# 语法: (|(condition1)(condition2)(condition3)...)

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

# 组合 AND 和 OR
(&
  (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

在成功将 LDAP 用户同步到平台后，您可以在用户列表中查看已同步的用户。

您可以在 添加 LDAP 时配置自动同步策略（可以稍后更新）或在成功添加 LDAP 后手动触发同步。以下是手动触发同步操作的方法。

注意：

- 在与平台集成的 LDAP 中新增的用户可以在执行用户同步操作之前登录平台。一旦他们成功登录平台，其信息将自动同步到平台。
- 从 LDAP 中删除的用户在同步后将显示为 `无效` 状态。
- 新同步用户的默认有效期为 永久。
- 与现有用户（本地用户、IDP 用户）同名的同步用户会自动关联。它们的权限和有效期将与现有用户保持一致。它们可以使用对应来源的登录方式登录平台。

操作程序

1. 在左侧导航栏中，点击 用户 > 身份提供者（**IDP**）。
2. 点击要手动同步的 ***LDAP** 名称*。
3. 点击右上角的 操作 > 同步用户。
4. 点击 同步。

   注意：如果您手动关闭同步提示对话框，将出现确认对话框以确认关闭。关闭同步提示对话框后，系统将继续同步用户。如果您仍在用户列表页面，将收到同步结果反馈。如果您离开用户列表页面，将不会收到同步结果。

相关操作

您可以点击列表页面右侧的
或在详情页面右上角点击 操作 以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，通过此 LDAP 目前同步到平台的用户也将被更新。从 LDAP 中删除的用户将在平台用户列表中变为无效。您可以通过执行清理无效用户的操作来清理垃圾数据。
删除 **LDAP**	删除 LDAP 后，通过此 LDAP 同步到平台的所有用户将变为 无效 状态（用户与角色之间的绑定关系保持不变），并且他们无法登录平台。重新集成后，需要重新执行同步以激活用户。 提示：删除 IDP 后，如果需要删除通过 LDAP 同步到平台的用户和用户组，请在提示框下方勾选 清理 **IDP** 用户和用户组 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

添加 OIDC

操作步骤

通过 YAML 添加 OIDC

示例：配置 OIDC Connector

相关操作

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，允许参与方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，平台将使用用户的 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：若用户名和密码错误，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，平台还支持通过 YAML 添加 OIDC，允许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**

以下示例演示如何配置 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, used for OIDC basic authentication
      - profile                                             # Optional, used to obtain
```

```
user basic information
      - email                                                  # Optional, used to obtain
user email

    # Claim mapping configuration
    # Configure the mapping relationship between OIDC returned claims and platform user
attributes
    claimMapping:
      email: email                                             # Email mapping, used for user
unique identification
      groups: groups                                           # User group mapping, used for
organization structure
      phone: ""                                                # Phone mapping, optional
      preferred_username: preferred_username                   # Username mapping, 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 definition of
golang language type. For example: string, int, bool, map[string]string, []string, []int

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

相关操作

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

操作	说明
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，并根据当前配置重新同步。

操作	说明
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，用户可通过成功登录平台激活。 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。

Menu

故障排除

删除用户

问题描述

解决方案

删除用户

目录

问题描述
解决方案
清理已删除的 IDP 用户
清理已删除的本地用户

问题描述

问题：创建或同步新用户时，系统提示用户已存在。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

为解决此问题，您需要在 global 集群的任意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 集群任意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 集群任意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'
   ```

Menu

用户策略

介绍

概述

配置安全策略

可用策略

介绍

该平台提供全面的用户安全策略，以增强登录安全性并防止恶意攻击。

目录

概述

配置安全策略

步骤

可用策略

概述

平台支持以下安全策略：

- 密码安全管理
- 用户账户禁用
- 用户账户锁定
- 用户通知
- 访问控制

配置安全策略

步骤

1. 在左侧导航栏中，点击 用户角色管理 > 用户安全策略
2. 在右上角点击 更新
3. 根据需要配置安全策略
4. 点击 更新 保存更改

WARNING

策略配置注意事项：

- 勾选框以启用相应策略
- 取消勾选框以禁用相应策略
- 被禁用的策略会保留其配置数据
- 重新启用策略时，先前的设置将被恢复

可用策略

策略	描述
用户身份验证策略	启用基于密码登录的双重身份验证： - 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，企业通信工具服务器）
密码安全策略	管理密码要求： 首次登录： - 在首次平台登录时强制用户更改密码 定期更新： - 在规定期限内要求用户更改密码（例如，90天） - 密码未更新前禁止登录

策略	描述
用户禁用策略	自动禁用不活跃账户： - 在规定的无登录期限后触发
用户锁定策略	防止暴力破解攻击： 锁定条件： - 在24小时内指定次数的登录失败后触发 锁定时长： - 账户将锁定指定分钟数 - 锁定期过后自动解锁
通知策略	管理用户通知： - 用户创建后通过电子邮件发送初始密码
访问控制	管理用户会话和访问： 会话管理： - 在指定时间后自动注销不活跃会话 - 限制最大并发在线用户数 浏览器控制： - 当所有产品标签页关闭时结束会话 - 防止同一客户端重复登录 - 重要注意事项： - 访问控制仅影响更新策略后的新登录 - 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每个客户端最后一次登录。

Menu

多租户（项目）

介绍

介绍

Project

Namespaces

Clusters、Projects 和 Namespaces 之间的关系

功能指南

创建项目

操作步骤

管理项目配额

什么是 ProjectQuota？

工作原理

何时使用 ProjectQuota

配额键及单位

分配策略建议

最佳实践与常见问题

Manage Project

Update Basic Project Information

Delete Project

管理项目集群

介绍

添加集群

移除集群

管理项目成员

导入成员

移除成员

介绍

目录

Project

Namespaces

Clusters、Projects 和 Namespaces 之间的关系

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

Namespaces 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。Namespaces 的主要特点包括：

- 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

Clusters、Projects 和 Namespaces 之间的关系

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），一个集群也可以向多个 Project 分配资源。
- 一个 Project 下可以创建多个 Namespace，且它们的资源配额总和不得超过 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。

Menu

功能指南

创建项目

操作步骤

管理项目配额

什么是 ProjectQuota？

工作原理

何时使用 ProjectQuota

配额键及单位

分配策略建议

最佳实践与常见问题

Manage Project

Update Basic Project Information

Delete Project

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。该项目将在资源和人员方面与其他项目（租户）隔离。创建项目时，您可以根据项目规模和实际业务需求分配资源。项目可以利用平台上多个集群的资源。

WARNING

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

目录

操作步骤

操作步骤

1. 在 **Project Management** 视图中，点击 **Create Project**。

2. 在 **Basic information** 页面，配置以下参数：

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

参数	描述
	注意：项目名称黑名单包括平台集群下的特殊命名空间名称：`cpaas-system`、`cert-manager`、`default`、`global-credentials`、`kube-ovn`、`kube-public`、`kube-system`、`nsx-system`、`alauda-system`、`kube-federation-system`、`ALL-ALL` 和 `true`。
Cluster	与项目关联的集群，管理员可以在此分配资源配额。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被选择。

3. 点击 **Next**，在项目配额设置步骤中，阅读 Manage Resource Quotas 来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，从而在项目级别规划和管理容量，同时仍将限制委托给各个命名空间。

目录

什么是 ProjectQuota？
工作原理
何时使用 ProjectQuota
配额键及单位
分配策略建议
最佳实践与常见问题

什么是 **ProjectQuota**？

- ResourceQuota（Kubernetes 原生）限制每个命名空间的资源（CPU、内存、对象数量等）。有关概念、键和用法，请参见：
  - Resource Quotas
- ProjectQuota 定义了项目范围的上限：项目内所有命名空间的 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为各命名空间分配 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。
- 聚合强制执行（准入时）：
  - 创建或更新命名空间的 ResourceQuota 时，平台会计算该项目内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说明错误。
- 执行模型：
  - ProjectQuota 约束通过命名空间 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悄然膨胀。

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 以核为单位（例如 `2`、`500m`）

- 内存以字节为单位（例如 `8Gi`）
- 对象类键使用整数计数

当所有命名空间对应键的总和接近或超过 ProjectQuota 硬限制时，ACP 会阻止该键的 ResourceQuota 进一步创建或扩容。

分配策略建议

- 先定义项目级“大桶”（ProjectQuota），再细分为各命名空间的 ResourceQuota，分配给团队或环境。
- 保留 10% - 30% 的余量以应对突发和弹性扩缩。
- 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，配合容量规划（节点/存储）和预算管理。

Manage Project

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

Update Basic Project Information
Procedure
Delete Project
Procedure

Update Basic Project Information

更新指定项目的基本信息，例如显示名称和描述。

Procedure

1. 在 **Project Management** 视图中，点击要更新的项目名称。
2. 在左侧导航栏中，点击 **Details**。
3. 点击右上角的 **Actions** > **Update Basics**。
4. 修改或输入 **Display name** 和 **Description**。
5. 点击 **Update**。

Delete Project

删除不再使用的项目。

> **WARNING**
>
> 项目删除后，集群中该项目占用的资源将被释放。

Procedure

1. 在 **Project Management** 视图中，点击要删除的项目名称。
2. 在左侧导航栏中，点击 **Details**。
3. 点击右上角的 **Actions** > **Delete Project**。
4. 输入项目名称并点击 **Delete**。

管理项目集群

本指南解释了如何管理项目的集群关联。您可以添加集群以将其资源分配给项目，或者移除集群以回收已分配的资源。

目录

介绍
添加集群
过程
移除集群
过程

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。此功能在以下场景中非常有用：

- 当项目资源不足以支持业务运营时
- 当新创建或添加的集群需要分配到现有项目时
- 当需要从项目中回收集群资源时
- 当特定项目需要对某个集群的独占访问时

添加集群

将集群添加到项目并设置其资源配额。

过程

1. 在 项目管理 视图中，点击您希望添加集群的项目名称。
2. 在左侧导航栏中，点击 详细信息。
3. 在右上角点击 操作 > 添加集群。
4. 选择集群并设置将分配给当前项目的资源配额。可以配置以下资源：
   - CPU（核心）
   - 内存（Gi）
   - 存储（Gi）
   - PVC 数量（数量）
   - Pods（数量）
   - vGPU（虚拟 GPU）/MPS/pGPU（物理 GPU，核心）
   - 视频内存配额

NOTE

- 仅在集群中部署了 GPU 插件时，才能配置 GPU 资源配额。

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

  GPU 单位：100 个虚拟核心等于 1 个物理核心（1 pGPU = 1 核心 = 100 GPU-管理核心 = 100 MPS 核心），且 pGPU 单位仅能以整数分配。GPU-管理 1 单位内存等于 256 Mi，MPS GPU 1 单位内存等于 1 Gi，1024 Mi = 1 Gi。
- 如果未设置某种类型资源的配额，则默认为 无限。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，而没有上限。
- 设置的项目配额值应在页面上显示的配额范围内。在每个资源配额输入框下，会显示已分配的配额和该资源的总量以供参考。

5. 点击 添加。

移除集群

移除与项目关联的集群。

> **WARNING**
>
> - 移除集群后，项目无法使用已移除集群下的业务资源。
> - 当要移除的集群出现异常时，异常集群下的资源无法清理。建议在移除前修复该集群。

过程

1. 在 项目管理 视图中，点击您希望移除集群的项目名称。
2. 在左侧导航栏中，点击 详细信息。
3. 在右上角点击 操作 > 移除集群。
4. 在弹出的 移除集群 对话框中，输入要移除的集群名称，然后点击 移除 按钮以成功移除该集群。

管理项目成员

本指南解释了如何管理项目成员，包括导入成员和分配与项目相关的角色。

目录

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

移除成员
- 操作步骤

导入成员

您可以通过导入现有平台用户或添加 OIDC 用户来授予用户对项目及其命名空间的操作权限。您可以分配角色，如项目管理员、命名空间管理员、开发人员或具有项目和命名空间管理权限的自定义角色。

约束与限制

- 当平台上未配置 OIDC IDP 时：
  - 只能将现有平台用户导入为项目成员，包括：

    - 已成功登录的 OIDC 用户
    - 通过 LDAP 同步的用户
    - 本地用户
    - 以 OIDC 用户身份添加到其他项目的用户（来源标记为 `-`）

- 当配置了 OIDC IDP 时：
    - 您可以添加符合输入要求的有效 OIDC 帐户
    - 添加时无法验证帐户的有效性
    - 确保帐户有效，否则将无法正常登录
- 系统默认管理员用户和当前登录用户无法被导入

操作步骤

1. 在 项目管理 视图中，点击要管理的项目名称。
2. 在左侧导航栏中，点击 成员。
3. 点击 导入成员。
4. 选择 成员列表 或 **OIDC** 用户。

从成员列表导入

您可以从成员列表中导入所有用户或选定用户。

> **TIP**
>
> 使用右上角的用户组下拉菜单和搜索框按用户名过滤用户。

导入所有用户：

1. 选择 成员列表。
2. 点击 绑定 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 命名空间 下拉菜单中选择。

3. 点击 导入全部。

导入特定用户：

1. 选择 成员列表。

2. 使用复选框选择一个或多个用户。

3. 点击 绑定 下拉菜单，选择要分配给选定用户的角色。
   如果角色需要命名空间，请从 命名空间 下拉菜单中选择。

4. 点击 导入。

导入 **OIDC** 用户

1. 选择 **OIDC** 用户。

2. 点击 添加 创建成员记录（对于多个成员重复此操作）。

3. 在 名称 字段中输入 OIDC 认证的用户名。

> **WARNING**
>
> 确保用户名对应于可以被配置的 OIDC 系统认证的帐户，否则登录将失败。

4. 从 角色 下拉菜单中选择角色。
   如果角色需要命名空间，请从 命名空间 下拉菜单中选择。

5. 点击 导入。

成功导入后，您可以查看：

- 项目成员列表中的成员
- 平台管理 **>** 用户 中的用户
  - 来源显示为 "-"，直到首次登录/同步
  - 来源在成功登录/同步后更新

移除成员

移除项目成员以撤销其权限。

操作步骤

1. 在 项目管理 视图中，点击项目名称。

2. 在左侧导航栏中，点击 成员。

> **TIP**
>
> 使用搜索框旁边的下拉列表按 姓名、显示名称 或 用户组 过滤成员。

3. 点击要移除的成员旁边的 移除。

4. 在提示对话框中确认移除。

Menu

审计

介绍

前提条件

操作步骤

搜索结果

介绍

该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记录的操作记录。这有助于您分析特定问题，并快速解决在集群、自定义应用及其他领域发生的问题。

通过审计，您可以跟踪 Kubernetes 集群中的各种变化，包括：

- 在特定时间段内集群发生了哪些变化
- 谁实施了这些变化（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）
- 操作员位置（集群内部或外部）
- 用户操作记录（更新、删除、管理操作）及其结果

目录

前提条件

操作步骤

搜索结果

前提条件

您的账户必须具备平台管理或平台审计权限。

操作步骤

1. 在左侧导航栏中，点击 审计。

2. 从标签中选择审计范围：

   - 用户操作：查看已登录平台的用户的操作记录
   - 系统操作：查看系统操作记录（操作员以 `system:` 开头）

3. 配置查询条件以过滤审计事件：

查询条件	描述
操作员	操作员的用户名或系统账户名（默认：`所有`）
操作类型	操作的类型（创建、更新、删除、管理、回滚、停止等，默认：`所有`）
集群	包含被操作资源的集群（默认：`所有`）
资源类型	被操作资源的类型（默认：`所有`）
资源名称	被操作资源的名称（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：`过去 30 分钟`）。您可以选择预设范围或自定义范围。
- 点击刷新图标以更新搜索结果。
- 点击导出图标以下载结果为 `.csv` 文件。

搜索结果

搜索结果显示以下信息：

参数	描述
操作员	操作员的用户名或系统账户名
操作类型	操作的类型（创建、更新、删除、管理、回滚、停止等）
资源名称/类型	被操作资源的名称和类型
集群	包含被操作资源的集群
命名空间	包含被操作资源的命名空间
客户端 IP	执行操作所使用的客户端的 IP 地址
操作结果	基于 API 返回代码的操作结果（2xx = 成功，其他 = 失败）
操作时间	操作的时间戳
详细信息	点击 详细信息 按钮以在 审计详细信息 对话框中以 JSON 格式查看完整的审计记录

Menu

遥测

安装

前提条件

安装步骤

启用在线运维

卸载步骤

安装

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

目录

前提条件
安装步骤
启用在线运维
卸载步骤

前提条件

安装前，请确保：

- Alauda Container Platform 拥有有效的许可证
- global 集群能够访问互联网

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 以部署该插件

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 以移除该插件

Menu

证书

自动化 **Kubernetes** 证书轮换

cert-manager

OLM 证书

证书监控

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

工作原理

轮换流程

运行注意事项

安装

请参阅 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	集群管理使用的 kubeconfig 中的客户端证书	控制平面节点
controller-manager.conf	kube-controller-manager 使用的 kubeconfig 中的客户端证书	控制平面节点
scheduler.conf	kube-scheduler 使用的 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宿主机的不同路径，需要从相应文件中读取内容。为此，在目标节点上创建一个临时 Pod，并挂载证书目录，使 Pod 能读取证书信息。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

   需要轮换的证书会被放入队列等待处理。轮换程序会评估近期的轮换活动及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

   轮换程序根据内部存储的 CA 信息生成新证书。轮换过程中，在目标节点创建临时 Pod 并挂载所需证书目录，以便受控地修改文件。

5. 重启组件

   需要重启的组件：

   - `kube-apiserver`：需重启以加载新证书。重启时会重新生成其内部的环回证书（有效期一年，仅内部使用，无法外部轮换）。
   - `kube-controller-manager`：需重启以重新加载 kubeconfig 文件。
   - `kube-scheduler`：需重启以重新加载 kubeconfig 文件。
   - `kubelet`：需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，需以 `hostPID` 为 true 挂载宿主机文件系统，并在容器中运行 "systemctl restart kubelet"。

   自动重载：

- Etcd 可自动重载证书。

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='{.spec.clusterIP}'):8080/cluster-cert-rotator/download/cert-renew" -o ./cert-renew && 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='{.spec.clusterIP}'):8080/cluster-cert-rotator/download/renew-all.sh" -o ./renew-all.sh
```

cert-manager

每个集群将自动部署 **cert-manager** 的证书

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 来管理其 TLS 证书，确保通信安全。

目录

概述

工作原理

识别 cert-manager 管理的证书

常见标签和注解

相关资源

概述

Cert-manager 通过 Kubernetes 自定义资源定义（CRD）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**：处理证书请求的内部资源

工作原理

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签，以确保服务的持续可用性。

识别 **cert-manager** 管理的证书

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。

常见标签和注解

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 备用名称

相关资源

- cert-manager Official Documentation ↗

Menu

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 的服务器/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
内置告警规则
Kubernetes 证书告警
平台组件证书告警

证书状态监控

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

当前证书状态和过期时间可以在 **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，然后进入 **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