Alauda Container Platform Registry 管理

本节介绍如何安装、配置、公开和管理适用于 ACP 4.4 的 Alauda Container Platform Registry。

Alauda Container Platform Registry 部署在 `image-registry-system` 中，并由 `cluster-image-registry-operator` 进行协调。它使用 `Config/cluster`、`ImagePruner/cluster`、以及 `image.alauda.io/v1` Image API 和受管服务帐户拉取密钥。

> 重要：对于早于 `v4.4` 的 Alauda Container Platform Registry 版本，包括 ACP 4.3 及更早版本，请联系技术支持以获取升级和迁移计划。

目录

管理主题

管理主题

需求	继续阅读
查看 Registry 架构、仓库命名和常用术语。	Registry 概览
安装或升级 Alauda Container Platform Registry，或检查 Operator 状态。	Alauda Container Platform Registry

需求	继续阅读
配置存储、镜像限制、拉取密钥和计划清理。	设置和配置 Registry
为外部客户端公开 Registry。	公开 Registry
管理命名空间镜像访问、使用情况报告、清理、垃圾回收和签名验证。	管理访问和清理
从工作负载或开发人员工作站使用 Registry。	Registry 镜像使用

Alauda Container Platform Reg

集成镜像 Registry

仓库名称

常用术语

自动镜像清理

操作注意事项

Alauda Container Platform Reg

主要组件

安装 Alauda Container Platform Registry

升级 Alauda Container Platform Registry

更改 Registry 管理状态

ImagePruner 协调

检查 Operator 和 Registry 状态

检查 Registry 日志和指标访问

设置和配置 I

先决条件

配置开发存储

配置 PVC 存储

配置 S3 兼容存

配置云存储后端

配置请求处理

配置托管服务账

制

镜像

公开 Registry

前提条件

公开默认 Registry

公开自定义安全 Registry 主机

配置客户端信任

验证外部访问

故障排除

管理访问权限

任务索引

先决条件

授予命名空间权

查看使用情况

验证 Image 签名

清理 Image

运行 Registry 均

Alauda Container Platform Registry 概述

Alauda Container Platform Registry 是 ACP 4.4 集群的集成镜像 Registry。它提供 内部镜像存储、ImageStream 元数据、基于命名空间的访问控制、托管服务账户 拉取凭证以及镜像清理功能。

目录

集成镜像 Registry

仓库名称

常用术语

自动镜像清理

操作注意事项

集成镜像 Registry

Registry 作为集群工作负载运行于 `image-registry-system` 中，并由 Image Registry Operator 管理。`image-registry` Deployment 处理 OCI 推送和拉取流量，而镜像 元数据则通过聚合的 `image.alauda.io/v1` Image API 提供。

镜像数据和镜像元数据分别存储：

数据类型	存储位置
镜像 blob 和 manifest	`Config/cluster.spec.storage` 中配置的存储后端：`emptyDir`、`PVC`、`S3`、`Swift`、`GCS`、`IBMCOS` 或 `Azure`。
镜像元数据	ACP Image API 资源，例如 `Image`、`ImageStream`、`ImageStreamTag`、`ImageStreamImage` 和 `ImageSignature`。

Registry 与 ACP 身份验证和 Kubernetes 授权集成。 命名空间 RoleBindings 控制哪些用户可以拉取、推送、删除或列出镜像内容。镜像清理和 Registry 垃圾回收由管理员和 Operator 管理。

仓库名称

镜像仓库按命名空间划分，并使用以下格式：

```
<registry-host>/<namespace>/<repository>:<tag>
```

仓库名称在命名空间后不得包含其他路径段。例如，`registry.example.com/team-a/my-app:v1` 有效，但 `registry.example.com/team-a/backend/my-app:v1` 不受支持。

此限制由 Image API 强制执行，而不是由 Registry 强制执行。OCI Registry 接受推送到 `team-a/backend/my-app` 的请求并存储 manifest，但 Image API 无法表示该仓库，因为 ImageStream 名称必须是 DNS-1123 子域名。此类镜像没有 `ImageStream`、`Image` 或 `ImageStreamTag` 元数据，因此在 Registry 工作流中不可见。 命名空间后始终只使用一个仓库段。

常用术语

术语	含义
镜像仓库	按命名空间划分的镜像标签和摘要集合，地址为 `<namespace>/<repository>:<tag>`。

术语	含义
ImageStream	记录仓库标签规范和标签历史的 ACP Image API 资源。
Image	针对某个摘要的集群范围镜像元数据。
ImagePruner	配置计划清理作业的单例自定义资源。
托管拉取密钥	由 Operator 生成并注入的服务账户拉取凭证。
Registry 存储	存储镜像 blob 和 manifest 的后端。

自动镜像清理

Registry 使用 `imagepruners.imageregistry.operator.alauda.io/cluster` 配置计划清理。 Operator 渲染一个 `image-pruner` CronJob，该 CronJob 运行 `ac adm prune images`，并应用配置的保留策略。

清理会先移除未使用的镜像元数据。移除元数据后，Registry 垃圾回收会回收存储空间。

操作注意事项

- Registry 资源使用 API 组 `image.alauda.io/v1` 和 `imageregistry.operator.alauda.io/v1`。
- Registry 使用 `Image`、`ImageStream`、`ImageStreamTag` 和 `ImageSignature` 等 Image API 资源，以及 Operator 资源 `ImagePruner`。
- 对于 ACP Registry 工作流，请使用 `ac`。

Alauda Container Platform Registry

Image Registry Operator 安装并管理集群范围的 Alauda Container Platform Registry 实例。它根据 `Config/cluster` 协调 Registry 运行时资源，并根据 `ImagePruner/cluster` 协调计划的清理资源。

目录

主要组件

安装 Alauda Container Platform Registry

从 OperatorHub 安装

使用 YAML 安装

先决条件

升级 Alauda Container Platform Registry

更改 Registry 管理状态

ImagePruner 协调

检查 Operator 和 Registry 状态

检查 Registry 日志和指标访问

常见 Operator 问题

主要组件

组件	用途
`cluster-image-registry-operator` Deployment	根据 `Config/cluster` 和 `ImagePruner/cluster` 协调单例 Registry。
`image-registry` Deployment	提供 OCI 推送和拉取流量、身份验证、授权、存储访问、运行状况检查和指标。
`image-api-server` Deployment	通过 Kubernetes API 聚合提供 ACP Image API。
`APIService/v1.image.alauda.io`	向 Kubernetes API server 注册 `image.alauda.io/v1`。
`node-ca` DaemonSet	将 Registry CA 信任和 Registry 服务主机映射分发到节点。
`image-pruner` CronJob	运行计划的镜像清理，并在 Registry 受管理期间运行 Registry 垃圾回收。
受管理的 imagePullSecret controller	为内部 Registry 创建、注入、刷新和移除 service account 拉取 Secret。

安装 Alauda Container Platform Registry

当 Operator 软件包在 OperatorHub 中可用时，从 ACP Web 控制台安装 Image Registry Operator，以部署 Alauda Container Platform Registry。仅当 Web 控制台不可用时，才使用 YAML 路径进行受控自动化安装或在支持人员指导下安装。

两种安装路径都要求目标 CatalogSource 中存在 `cluster-image-registry-operator` 软件包。有关如何验证软件包并在缺失时上传软件包，请参阅使用 YAML 安装。

从 OperatorHub 安装

1. 登录 ACP，然后导航到管理员页面。
2. 在左侧导航栏中，单击 **Marketplace** > **OperatorHub**。
3. 搜索 **Alauda Container Platform Registry** 或 `cluster-image-registry-operator`。
4. 单击安装。

5. 在安装页面上，除非版本指南另有说明，否则使用以下设置：

参数	建议值
Channel	`stable`
Installation Mode	`All namespaces on the cluster`
Namespace	`image-registry-system`
Upgrade Strategy	`Manual`

6. 单击安装。
7. 如果出现批准提示，请查看并批准生成的安装计划。
8. 等待 Operator 状态变为已安装。

验证安装：

```
kubectl -n image-registry-system get subscription,csv,installplan
kubectl -n image-registry-system get deployment cluster-image-registry-op
erator
```

预期结果：

- 已安装的 CSV 为 `Succeeded`。
- `cluster-image-registry-operator` Deployment 可用。

使用 YAML 安装

先决条件

目标 CatalogSource 中必须已存在 `cluster-image-registry-operator` 软件包。全新平台安装不包含该软件包；缺少该软件包时，`check-subscription.cpaas.io` admission webhook 会拒绝 `Subscription`：

```
admission webhook "check-subscription.cpaas.io" denied the request:
packagemanifests.packages.operators.coreos.com "cluster-image-registry-op
erator" not found
```

继续之前验证软件包：

```
kubectl -n cpaas-system get packagemanifests | grep cluster-image-registr
y-operator
```

如果缺少该软件包，请先按照平台的 Operator 软件包上传操作步骤 上传 Operator 软件包。例如：

```
violet push cluster-image-registry-operator.stable.amd64.<version>.tgz \
  --platform-address="https://<platform-host>" \
  --platform-username="<platform-admin-user>" \
  --platform-password="<platform-admin-password>" \
  --target-catalog-source=platform \
  --dest-repo="<global-registry-address>" \
  --username="<registry-user>" \
  --password="<registry-password>"
```

推送必须使用在目标仓库上具有 `push` 权限的 Registry 账户。仅具有拉取权限的账户会失败并显示 `requested access to the resource is denied`。推送不会为 Operator 自身的工作负载配置镜像拉取凭据；请参阅下面安装步骤中的拉取 Secret 说明。

如果安装命名空间不存在，请创建它：

```
kubectl get namespace image-registry-system >/dev/null 2>&1 || \
  kubectl create namespace image-registry-system

kubectl label namespace image-registry-system \
  cpaas.io/project=cpaas-system \
  pod-security.kubernetes.io/audit=privileged \
  pod-security.kubernetes.io/enforce=privileged \
  pod-security.kubernetes.io/warn=privileged \
  --overwrite
```

创建名为 `image-registry-operator-subscription.yaml` 的文件：

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  annotations:
    cpaas.io/target-namespaces: ""
  name: cluster-image-registry-operator
  namespace: image-registry-system
spec:
  channel: stable
  installPlanApproval: Manual
  name: cluster-image-registry-operator
  source: platform
  sourceNamespace: cpaas-system
```

应用 `Subscription`：

```
kubectl apply -f image-registry-operator-subscription.yaml
```

批准生成的 `InstallPlan`：

```
kubectl -n image-registry-system get installplan

kubectl -n image-registry-system patch installplan <installplan-name> \
  --type=merge \
  -p '{"spec":{"approved":true}}'
```

等待 Operator：

```
kubectl -n image-registry-system wait \
  --for=condition=Available \
  deployment/cluster-image-registry-operator \
  --timeout=300s
```

Operator 随后会创建 `image-registry`、`image-api-server` 和 `node-ca` 工作负载。这些工作负载从平台 Registry 拉取镜像，因此 Operator 使用的 ServiceAccount 必须携带可用的

Registry 拉取 Secret。如果工作负载一直处于 `ImagePullBackOff` 状态并显示 `insufficient_scope: authorization failed`，请确认平台 Registry 拉取 Secret 已附加到这些 ServiceAccount：

```
kubectl -n image-registry-system get serviceaccount registry node-ca imag
e-api-server \
  -o jsonpath='{range .items[*]}{.metadata.name}{" "}{.imagePullSecrets
[*].name}{"\n"}{end}'
```

如果缺少 Secret，请将其附加，然后重启工作负载：

```
kubectl -n image-registry-system patch serviceaccount registry node-ca im
age-api-server \
  --type=merge -p '{"imagePullSecrets":[{"name":"<platform-pull-secret
>"}]}'
kubectl -n image-registry-system rollout restart deployment/image-registr
y daemonset/node-ca
```

升级 Alauda Container Platform Registry

通过 OperatorHub 升级 Image Registry Operator，以升级 Alauda Container Platform Registry。Operator 升级会更新 `cluster-image-registry-operator` Deployment，然后协调现有的 `Config/cluster`、`ImagePruner/cluster`、Registry、Image API 及支持资源。

升级前，请确认当前 Registry 运行状况正常。

> 重要：对于早于 `v4.4` 的 Alauda Container Platform Registry 版本（包括 ACP 4.3 及更早版本），请联系技术支持以获取升级和迁移计划。

```
kubectl -n image-registry-system get subscription,csv
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
kubectl -n image-registry-system rollout status deployment/image-api-serv
er --timeout=300s
```

在提供新的 Operator package 之前，将 Subscription approval strategy 设置为 `Manual`：

```
kubectl -n image-registry-system get subscription
kubectl -n image-registry-system patch subscription <subscription-name> \
  --type=merge \
  -p '{"spec":{"installPlanApproval":"Manual"}}'
```

Placeholder	Description
`<subscription-name>`	Alauda Container Platform Registry Operator 的 Subscription。使用 `kubectl -n image-registry-system get subscription` 返回的名称。

在 **Administrator** > **Marketplace** > **OperatorHub** 中提供目标 Operator package。如果不存在该 package，请先按照平台的 Operator package 上传操作步骤 操作，然后再继续。

在 OperatorHub 中打开 **Alauda Container Platform Registry**，选择目标 channel 和 version，并查看待处理的升级。只有在确认 Subscription 使用 `Manual` approval 后，才批准生成的 InstallPlan：

```
PLAN_NAME="$(kubectl -n image-registry-system get subscription <subscript
ion-name> \
  -o jsonpath='{.status.installPlanRef.name}')"
kubectl -n image-registry-system get installplan "$PLAN_NAME" -o yaml
kubectl -n image-registry-system patch installplan "$PLAN_NAME" \
  --type=merge \
  -p '{"spec":{"approved":true}}'
```

等待新的 CSV 和 Operator Deployment，然后验证 Registry data plane 是否完成协调：

```
kubectl -n image-registry-system get subscription,csv,installplan
kubectl -n image-registry-system rollout status deployment/cluster-image-registry-operator --timeout=300s
kubectl -n image-registry-system rollout status deployment/image-registry --timeout=300s
kubectl -n image-registry-system rollout status deployment/image-api-server --timeout=300s
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl get imagepruners.imageregistry.operator.alauda.io cluster -o yaml
```

预期结果：

- 新的 CSV 为 `Succeeded`，且 Subscription 报告新的 installed CSV。
- `cluster-image-registry-operator`、`image-registry` 和 `image-api-server` 可用。
- 现有存储、路由、Image API 元数据和 ImagePruner 配置仍保持协调状态。
- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。

> 注意：CSV 升级会重新应用 Operator 的 ServiceAccount。如果在安装期间已将 Registry pull Secret 附加到该 ServiceAccount，升级可能会清除该 Secret，导致新的 `cluster-image-registry-operator` Pod 保持在 `ImagePullBackOff` 状态并显示 `insufficient_scope: authorization failed`。请重新附加 Secret，然后重启 Deployment：
>
> ```
> kubectl -n image-registry-system patch serviceaccount cluster-image-registry-operator \
>   --type=merge -p '{"imagePullSecrets":[{"name":"<platform-pull-secret>"}]}'
> kubectl -n image-registry-system rollout restart deployment/cluster-image-registry-operator
> ```

Operator 升级期间，请勿删除 Subscription、`Config/cluster`、存储对象或 Image API 元数据。如果升级未完成，请先检查 Subscription、InstallPlan、CSV 事件、Operator 日志和 `Config/cluster.status.conditions`，再采取进一步操作。

更改 **Registry** 管理状态

启用 Registry 前，请配置一个受支持的存储后端。如果 `Config/cluster.spec.storage` 为空，将 `managementState` 设置为 `Managed` 会使 Operator 保持在 `StorageNotConfigured`，并且无法协调 Registry data plane。

将 `Config/cluster.spec.managementState` 设置为 `Managed`，以启用 Registry：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{"spec":{"managementState":"Managed"}}'
```

验证 Registry data plane 是否可用：

```
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
kubectl -n image-registry-system rollout status deployment/image-api-serv
er --timeout=300s
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
```

预期结果：

- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。

> 警告：将管理状态设置为 `Removed` 会停止 Registry 运行时组件。 Registry 被移除期间，推送和拉取流量以及 Image API 服务器均不可用。 Operator 也可能保留 `image-pruner` CronJob，但会禁用已移除 Registry 的 registry 垃圾回收。 Registry 被移除期间，请勿依赖计划清理。 仅应在维护窗口期间或根据支持人员指导进行恢复时使用此状态。

切换到 `Removed` 前，请备份所需的镜像数据，并检查存储 Management Mode：

```
kubectl get configs.imageregistry.operator.alauda.io cluster \
  -o jsonpath='{.spec.storage.managementState}{"\n"}'
```

仅当 `spec.storage.managementState` 为 `Unmanaged`，或存储管理员已确认后端存储回收策略会在 Registry 实例移除后保留数据时，才能假定镜像数据会被保留。在更改管理状态前，请备份所需的镜像数据。

要停止 Registry，请将管理状态设置为 `Removed`：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{"spec":{"managementState":"Removed"}}'
```

验证 data plane Deployment 是否已移除：

```
kubectl -n image-registry-system get deployment image-registry image-api-server --ignore-not-found
```

预期结果：

- Registry 被移除期间，`image-registry` 和 `image-api-server` Deployment 不存在。

ImagePruner 协调

Operator 会将单例 `ImagePruner/cluster` 协调为 `image-pruner` 中的 `image-registry-system` CronJob。在 `ImagePruner` 资源中配置保留策略。请参阅设置和配置 Registry。

CronJob 默认使用 Registry 内部服务 URL。

检查 Operator 和 Registry 状态

本节假定已按照更改 Registry 管理状态中的说明，将 `Config/cluster.spec.managementState` 设置为 `Managed`，从而启用 Registry。`spec.managementState` 没有默认值。未设置该值时，Operator 会报告原因为 `Removed` 的 `Available=True`，并且不存在 `image-registry` 和 `image-api-server` Deployment。

```
kubectl -n image-registry-system get subscription,csv,installplan
kubectl -n image-registry-system get deploy cluster-image-registry-operat
or image-registry image-api-server
kubectl -n image-registry-system get daemonset node-ca
kubectl -n image-registry-system get cronjob image-pruner
kubectl get apiservice v1.image.alauda.io
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl get imagepruners.imageregistry.operator.alauda.io cluster -o yaml
```

预期结果：

- Operator CSV 为 `Succeeded`。
- `cluster-image-registry-operator`、`image-registry` 和 `image-api-server` 可用。
- `node-ca` 已在目标节点上就绪。
- `APIService/v1.image.alauda.io` 为 `Available=True`。
- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。

检查 **Registry** 日志和指标访问

检查 Registry Pod：

```
kubectl -n image-registry-system get pods -l app.kubernetes.io/name=image
-registry
```

查看 Registry 日志：

```
kubectl -n image-registry-system logs deployment/image-registry -c regist
ry
```

检查监控 ServiceAccount 的指标访问权限。

`registry/metrics` 是仅用于 RBAC 的资源。`image.alauda.io/v1` API 发现文档不会公开该资源，而 `kubectl auth can-i` 会通过发现机制解析资源名称。因此，即使已授予 权限，`kubectl auth can-i get registry/metrics.image.alauda.io` 也始终报告 `no`。请改用 `SubjectAccessReview`：

```
for SA in prometheus-sa vm-sa; do
  printf '%s: ' "${SA}"
  kubectl create -f - -o jsonpath='{.status.allowed}{"\n"}' <<EOF
apiVersion: authorization.k8s.io/v1
kind: SubjectAccessReview
spec:
  user: system:serviceaccount:cpaas-system:${SA}
  resourceAttributes:
    verb: get
    group: image.alauda.io
    resource: registry
    subresource: metrics
EOF
done
```

预期结果：

- 每次检查都报告 `true`。

Operator 通过 `registry-monitoring` ClusterRoleBinding 将 `system:image-registry-metrics-reader` 绑定到 `prometheus-sa` 和 `vm-sa`。这些 ServiceAccount 由监控 插件创建，因此在安装监控之前，各项检查都会报告 `false`。

常见 Operator 问题

症状	检查
CSV 不是 `Succeeded`	`Subscription`、`InstallPlan`、CSV 事件以及 `cpaas-system` catalog source。
`Config/cluster` 为 `Degraded=True`	`Config.status.conditions`、Operator 日志、Registry Pod 事件、存储、TLS Secret 和 RBAC。

症状	检查
Registry Pod 处于 pending 状态	PVC 绑定、节点资源、节点选择器、污点、容忍度和拓扑约束。
`node-ca` 未就绪	DaemonSet 调度、Pod 日志、节点信任配置和主机映射更新。

设置和配置 **Registry**

使用此页面为 Alauda Container Platform Registry 配置存储、Registry 请求行为、服务帐户拉取凭证、镜像 限制和计划的镜像清理。

目录

先决条件

配置开发存储

配置 PVC 存储

配置 S3 兼容存储凭证

配置云存储后端

配置请求处理

配置托管服务账户拉取 Secret

配置镜像限制

配置计划的镜像清理

操作存储

先决条件

- ACP Registry 已安装，并且 `Config/cluster` 已存在。
- 您有权更新 `configs.imageregistry.operator.alauda.io`、`imagepruners.imageregistry.operator.alauda.io`、`LimitRange`、`ResourceQuota`

以及相关 Kubernetes 资源。

- 对于持久性存储，请在配置 `Config/cluster` 之前准备好存储后端和所需凭证。

配置开发存储

> 警告：`emptyDir` 将镜像数据存储在临时 Pod 存储中。当 Registry Pod 从其节点中移除时，数据会被删除，并且无法与多个 Registry 副本结合使用。仅在可以丢弃所有推送的镜像数据的开发或测试环境中使用它。请勿将 `emptyDir` 用于生产环境或任何必须保留镜像的环境。

修补 `Config/cluster`。`null` 字段会从当前配置中移除互斥的存储后端：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "managementState": "Managed",
      "replicas": 1,
      "storage": {
        "emptyDir": {},
        "pvc": null,
        "s3": null,
        "swift": null,
        "gcs": null,
        "ibmcos": null,
        "azure": null
      },
      "resources": {
        "requests": {
          "cpu": "500m",
          "memory": "500Mi"
        },
        "limits": {
          "cpu": "500m",
          "memory": "500Mi"
        }
      }
    }
  }'
```

验证配置和滚动发布状态：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
```

预期结果：

- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。
- `image-registry` Deployment 成功完成滚动发布。

配置 PVC 存储

生产环境请使用持久化后端。此示例使用 `ReadWriteOnce`、一个 Registry 副本，以及 Operator 对 RWO 存储所要求的 `Recreate` 滚动发布策略。创建名为 `image-registry-pvc.yaml` 的文件：

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: image-registry
  namespace: image-registry-system
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: <storage-class-name>
```

占位符	描述
`<storage-class-name>`	为 Registry PVC 提供资源的 StorageClass。运行 `kubectl get storageclass` 以列出可用值。对于多副本 Registry 部署，请选择支持所需访问模式的存储。

应用 PVC：

```
kubectl apply -f image-registry-pvc.yaml
```

修补 `Config/cluster` 以使用 PVC。使用合并补丁，以免删除其他 `Config/cluster.spec` 字段，例如路由或拉取密钥设置。`null` 字段会从当前配置中移除互斥的存储后端：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "managementState": "Managed",
      "replicas": 1,
      "rolloutStrategy": "Recreate",
      "storage": {
        "managementState": "Unmanaged",
        "emptyDir": null,
        "pvc": {
          "claim": "image-registry"
        },
        "s3": null,
        "swift": null,
        "gcs": null,
        "ibmcos": null,
        "azure": null
      },
      "resources": {
        "requests": {
          "cpu": "500m",
          "memory": "500Mi"
        },
        "limits": {
          "cpu": "500m",
          "memory": "500Mi"
        }
      }
    }
  }'
```

验证 PVC、配置和滚动发布状态：

```
kubectl -n image-registry-system get pvc image-registry
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
```

预期结果：

- `image-registry` PVC 的状态为 `Bound`。
- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。
- `image-registry` Deployment 成功完成滚动发布。

配置 S3 兼容存储凭证

在配置 `Config/cluster` 之前，创建用户管理的存储 Secret。Operator 会将此 Secret 合并到 Registry 私有配置中：

```
kubectl -n image-registry-system create secret generic image-registry-pri
vate-configuration-user \
  --from-literal=REGISTRY_STORAGE_S3_ACCESSKEY=<access-key-id> \
  --from-literal=REGISTRY_STORAGE_S3_SECRETKEY=<secret-access-key> \
  --dry-run=client -o yaml | kubectl apply -f -
```

占位符	描述
`<access-key-id>`	S3 兼容存储账户的访问密钥 ID。请从存储管理员处获取。
`<secret-access-key>`	S3 兼容存储账户的秘密访问密钥。仅将其存储在 Kubernetes Secret 中。

在不打印其内容的情况下验证 Secret：

```
kubectl -n image-registry-system get secret image-registry-private-config
uration-user \
  -o jsonpath='{.type}{"\n"}'
kubectl -n image-registry-system get secret image-registry-private-config
uration-user \
  -o jsonpath='{.data}' | wc -c
```

预期结果：

- Secret 存在于 `image-registry-system` 中。
- 第二条命令返回非零字节数。不要打印或解码 Secret 数据。

当客户端无法直接访问对象存储端点，且所有内容都必须通过 Registry 提供时，请使用 `disableRedirect: true`。

如果 S3 端点使用私有 CA，请创建包含所需密钥的引用 ConfigMap：

```
kubectl -n image-registry-system create configmap <trusted-ca-configmap>
\
  --from-file=ca-bundle.crt=/path/to/ca-bundle.crt \
  --dry-run=client -o yaml | kubectl apply -f -
```

在不显示证书的情况下验证 ConfigMap 密钥：

```
kubectl -n image-registry-system get configmap <trusted-ca-configmap> \
  -o jsonpath='{.data.ca-bundle\.crt}' | wc -c
```

预期结果：

- 对于 `ca-bundle.crt`，命令返回非零字节数。

修补 `Config/cluster`。`null` 字段会从当前配置中移除互斥的存储后端：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "managementState": "Managed",
      "replicas": 2,
      "storage": {
        "managementState": "Unmanaged",
        "emptyDir": null,
        "pvc": null,
        "swift": null,
        "gcs": null,
        "ibmcos": null,
        "azure": null,
        "s3": {
          "bucket": "<bucket-name>",
          "region": "<region>",
          "regionEndpoint": "https://<s3-endpoint>",
          "trustedCA": {
            "name": "<trusted-ca-configmap>"
          }
        }
      },
      "disableRedirect": true
    }
  }'
```

占位符	描述
`<bucket-name>`	用于存储 Registry blob 的现有对象存储存储桶或等效容器。
`<region>`	存储后端所需的地域值。使用提供商的值，或 S3 兼容服务所需的值。
`<s3-endpoint>`	Registry Pod 可访问的 S3 兼容端点主机和可选端口，不包含 `https://`，也不包含存储桶名称。示例提供了 `https://` 方案。
`<trusted-ca-configmap>`	`image-registry-system` 中包含对象存储端点 CA 证书的 ConfigMap。当端点使用 Registry Pod 信任的公共 CA 时，省略 `trustedCA`。

当前 operator 支持以下存储字段：`emptyDir`、`pvc`、`s3`、`swift`、`gcs`、`ibmcos` 和 `azure`。不要配置其他存储字段。

> 注意：`Config` CRD 还公开了一个 `storage.oss` 字段，该字段继承自上游 image registry operator。此 operator 未实现该字段。设置 `storage.oss` 会被视为“未配置存储后端”：Operator 会报告原因为 `StorageNotConfigured` 的 `Degraded=True`，并且 Registry 数据平面不会进行协调。仅使用上面列出的七个受支持字段。

验证配置和滚动发布：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
kubectl -n image-registry-system logs deployment/image-registry -c regist
ry --tail=80
```

预期结果：

- `Config/cluster` 报告 `Available=True`、`Progressing=False` 和 `Degraded=False`。
- Registry 日志中不显示存储身份验证、存储桶、端点或证书错误。

配置云存储后端

当相应的平台集成和凭证可用时，Operator 可以使用 Swift、GCS、IBM COS 或 Azure 存储。在 `Config/cluster.spec.storage` 中只配置一个存储字段；使用上面所示的相同 merge-patch 模式，并将其他受支持的存储字段设置为 `null`。

后端	主要 `Config/cluster.spec.storage` 字段	所需的用户管理 **Secret** 数据
Swift	`swift.authURL`、`swift.authVersion`、`swift.container`、`swift.domain`、`swift.domainID`、`swift.tenant`、`swift.tenantID` 和 `swift.regionName`	`REGISTRY_STORAGE_SWIFT_USERNAME` 和 `REGISTRY_STORAGE_SWIFT_PASSWORD`，或应 `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` 或 `REGISTRY_STORAGE_SWIFT_TOKENID`。

后端	主要 `Config/cluster.spec.storage` 字段	所需的用户管理 **Secret** 数据
GCS	`gcs.bucket`、`gcs.region`、`gcs.projectID` 和可选的 `gcs.keyID`	包含服务账户密钥文件的 `REGISTRY_STORAGE`
IBM COS	`ibmcos.bucket`、`ibmcos.location`、`ibmcos.resourceGroupName`、`ibmcos.resourceKeyCRN` 和 `ibmcos.serviceInstanceCRN`	`REGISTRY_STORAGE_IBMCOS_IAMAPIKEY`。
Azure	`azure.accountName`、`azure.container`、`azure.cloudName` 和可选的 `azure.networkAccess`	`REGISTRY_STORAGE_AZURE_ACCOUNTKEY`。

提供商集成可以填充地域、项目、账户或端点信息等平台特定值。请与存储管理员确认所需的提供商资源和凭证。配置后端后，按照上述存储过程中的说明，验证 `Config/cluster` 条件和 Registry 滚动发布。排查问题时不要打印 Secret 数据。

配置请求处理

使用 `Config/cluster.spec.readOnly` 拒绝镜像推送和删除。使用 `requests` 设置限制并发读取和写入、排队请求数以及队列等待时间：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "readOnly": false,
      "requests": {
        "read": {
          "maxRunning": 100,
          "maxInQueue": 500,
          "maxWaitInQueue": "30s"
        },
        "write": {
          "maxRunning": 50,
          "maxInQueue": 200,
          "maxWaitInQueue": "30s"
        }
      }
    }
  }'
```

当集群代理设置不足时，`proxy.http`、`proxy.https` 和 `proxy.noProxy` 字段用于配置 Registry 出站流量。`logLevel` 接受 `Normal`、`Debug`、`Trace` 或 `TraceAll`。`Config/cluster.spec` 还支持资源请求和限制、节点选择、容忍度、亲和性以及拓扑分布。

验证请求处理设置和滚动发布：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system rollout status deployment/image-registry
--timeout=300s
```

预期结果：

- `Config/cluster.spec.readOnly` 和 `Config/cluster.spec.requests` 包含请求的值。
- Registry Deployment 成功完成滚动发布。

配置托管服务账户拉取 **Secret**

Operator 包含一个托管的 imagePullSecret 控制器。当 `Config/cluster` 由其管理时，该控制器可以为内部 Registry 创建、注入、刷新和移除服务账户拉取 Secret。

使用其他主机或要忽略的命名空间对 `Config/cluster` 应用补丁。此补丁中的数组字段会替换当前数组，因此请包含所有必须保留配置的主机和命名空间：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "managementState": "Managed",
      "imagePullSecret": {
        "managementState": "Managed",
        "additionalRegistryHosts": [
          "registry.example.com"
        ],
        "ignoredNamespaces": [
          "kube-public"
        ],
        "ignoreSystemNamespaces": true
      }
    }
  }'
```

验证配置：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
```

预期结果：

- `Config/cluster.spec.imagePullSecret` 包含配置的管理状态、其他主机和要忽略的命名空间设置。

控制器会异步协调拉取凭证。等待控制器有足够时间完成协调后，验证托管的 RoleBinding 和一个 ServiceAccount：

```
kubectl -n <workload-namespace> get rolebinding image-registry-system-image-pullers -o yaml
kubectl -n <workload-namespace> get serviceaccount default \
  -o jsonpath='{.imagePullSecrets[*].name}{"\n"}'
kubectl -n <workload-namespace> get secret \
  -l imageregistry.operator.alauda.io/managed-pull-secret=true -o name
```

预期结果：

- RoleBinding 将 `system:image-puller` ClusterRole 授予该命名空间中的服务账户。
- 异步协调后，ServiceAccount 至少列出一个托管的拉取 Secret。
- 至少存在一个带有 managed-pull-secret 标签的 Secret。使用 `kubectl get secret` 仅检查元数据；不要打印或解码凭证数据。

占位符	描述
`<workload-namespace>`	应注入托管 ServiceAccount 拉取凭证的命名空间。该命名空间不得列在 `ignoredNamespaces` 中，并且必须满足 `ignoreSystemNamespaces`。

配置镜像限制

在 ACP Registry 中，镜像大小和标签数量限制通过 Kubernetes `LimitRange` 和 `ResourceQuota` 对象表示。

创建名为 `team-a-image-quota.yaml` 的文件以配置命名空间级别的配额：

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: image-registry-quota
  namespace: team-a
spec:
  hard:
    alauda.io/imagestreams: "20"
    alauda.io/images: "200"
    alauda.io/image-tags: "200"
```

应用配额：

```
kubectl apply -f team-a-image-quota.yaml
```

创建名为 `team-a-image-limits.yaml` 的文件以配置每个镜像和每个 ImageStream 的限制：

```
apiVersion: v1
kind: LimitRange
metadata:
  name: image-registry-limits
  namespace: team-a
spec:
  limits:
    - type: alauda.io/Image
      max:
        storage: 1Gi
    - type: alauda.io/ImageStream
      max:
        alauda.io/images: "100"
        alauda.io/image-tags: "100"
```

应用限制：

```
kubectl apply -f team-a-image-limits.yaml
```

验证配额和限制：

```
kubectl -n team-a get resourcequota image-registry-quota -o yaml
kubectl -n team-a get limitrange image-registry-limits -o yaml
```

预期结果：

- `ResourceQuota` 包含配置的 Image API 限制。
- `LimitRange` 包含配置的 `alauda.io/Image` 和 `alauda.io/ImageStream` 限制。

配置计划的镜像清理

创建或更新单例 `ImagePruner/cluster` 以配置计划的镜像清理。启用计划前确认保留策略，因为清理会移除未使用的镜像元数据。

创建名为 `image-pruner.yaml` 的文件：

```yaml
apiVersion: imageregistry.operator.alauda.io/v1
kind: ImagePruner
metadata:
  name: cluster
spec:
  schedule: "0 0 * * *"
  suspend: false
  keepTagRevisions: 3
  keepYoungerThanDuration: 60m
  resources:
    requests:
      cpu: 500m
      memory: 500Mi
    limits:
      cpu: 500m
      memory: 500Mi
```

应用配置：

```
kubectl apply -f image-pruner.yaml
```

验证清理器资源和生成的 CronJob：

```
kubectl get imagepruners.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system get cronjob image-pruner
```

预期结果：

- `ImagePruner/cluster` 包含配置的计划和保留策略。
- Operator 在 `image-registry-system` 中生成 `image-pruner` CronJob。

生成的 CronJob 运行 `ac adm prune images --confirm`。当 Registry 管理状态为 `Managed` 时，Operator 会添加 `--prune-registry=true`，因此计划任务还会在镜像元数据清理后执行 Registry 垃圾回收。对于手动运行的清理，仅当需要回收 blob 时添加 `--prune-registry`；否则，将垃圾回收作为单独的、经过审查的操作执行。

有关手动清理和 Registry 垃圾回收命令，请参阅管理访问和清理。

操作存储

对于基于 PVC 的 Registry 存储：

```
kubectl -n image-registry-system get pvc
kubectl -n image-registry-system describe pvc image-registry
kubectl get pv
```

常见操作：

- 如果 PVC 处于 pending 状态，请检查 StorageClass、访问模式、容量、配额和事件。
- 如果 Registry Pod 无法挂载存储，请检查 PV 绑定、节点附加状态和后端存储可用性。
- 如果镜像元数据存在但 blob 数据缺失，请确认 Registry 是否使用了 `emptyDir`，或存储后端是否已更改。
- 在确认数据保留决策前，不要删除 PVC、PV 或对象存储数据。

公开 Registry

默认情况下，对于集群内的工作负载，请使用内部服务。仅当开发人员机器、CI 系统或其他外部客户端必须推送或拉取镜像时，才公开 Alauda Container Platform Registry。

目录

前提条件

公开默认 Registry

公开自定义安全 Registry 主机

配置客户端信任

验证外部访问

故障排除

前提条件

- ACP Registry 已安装且可用。
- 你有权更新 `Config/cluster`，并在 `image-registry-system` 中创建或引用 TLS Secret。
- 集群具有能够将流量路由到 `image-registry-system` 的 Ingress 控制器。
- 对于自定义主机名，请准备 DNS 记录和 TLS 证书，其 Subject Alternative Name 与 Registry 主机名匹配。

公开默认 Registry

在 `Config/cluster` 中启用默认外部端点：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{"spec":{"defaultRoute":true}}'
```

Operator 会在 `image-registry-system` 中渲染名为 `default-route` 的 Kubernetes Ingress。

验证生成的 Ingress 和 Registry 配置：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system get ingress default-route
```

获取生成的主机名：

```
REGISTRY_RULE_HOST="$(
  kubectl -n image-registry-system get ingress default-route \
    -o jsonpath='{.spec.rules[0].host}'
)"

REGISTRY_STATUS_ADDRESS="$(
  kubectl -n image-registry-system get ingress default-route \
    -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'
)"
if [ -z "$REGISTRY_STATUS_ADDRESS" ]; then
  REGISTRY_STATUS_ADDRESS="$(
    kubectl -n image-registry-system get ingress default-route \
      -o jsonpath='{.status.loadBalancer.ingress[0].ip}'
  )"
fi

printf 'Ingress rule host: %s\n' "$REGISTRY_RULE_HOST"
printf 'Ingress status address: %s\n' "$REGISTRY_STATUS_ADDRESS"
```

预期结果：

- `Config/cluster.spec.defaultRoute` 为 `true`。
- `default-route` Ingress 存在于 `image-registry-system` 中。
- 非空的 `spec.rules[].host` 是可用的 Registry 主机名。空的规则主机表示 Ingress 是一条全匹配规则；状态地址标识 Ingress 控制器端点，不会自动成为 Registry DNS 名称。

如果默认 Ingress 没有可用的主机名，请改为配置自定义安全主机名。不要将空的规则主机作为 `--registry` 的值。

有可用主机名后，将其赋值给 `REGISTRY_HOST`。如果客户端使用私有 CA，请在运行 TLS 检查前配置该 CA。在登录前，使用客户端的常规信任设置运行检查，并确保检查成功：

```
REGISTRY_HOST="$REGISTRY_RULE_HOST"

test -n "$REGISTRY_HOST" || {
  echo "No usable Registry hostname was found; configure a custom secure
host." >&2
  exit 1
}

openssl s_client \
  -connect "$REGISTRY_HOST:443" \
  -servername "$REGISTRY_HOST" \
  -verify_return_error </dev/null
```

为生成的主机名写入凭据：

```
ac registry login \
  --registry "$REGISTRY_HOST"
```

公开自定义安全 **Registry** 主机

在 `image-registry-system` 中创建或提供 TLS Secret，然后配置 `Config.spec.routes[]`。

使用 Subject Alternative Name 与自定义主机名匹配的证书创建 Secret：

```
kubectl -n image-registry-system create secret tls registry-tls \
  --cert=/path/to/registry.example.com.crt \
  --key=/path/to/registry.example.com.key
```

当设置 `secretName` 时，TLS Secret 必须存在于 `image-registry-system` 中。为 `registry.example.com` 创建 DNS 记录，使其解析到 Ingress 控制器。

生成的 Ingress 使用 Registry 服务作为后端，并将后端协议设置为 HTTPS。

使用自定义路由修补 `Config/cluster`。请使用合并补丁，以免删除其他 `Config/cluster.spec` 字段（例如存储和 pull-secret 设置）。此补丁中的 `routes` 数组会替换当前的路由数组，因此请包含所有必须继续配置的自定义路由：

```
kubectl patch configs.imageregistry.operator.alauda.io cluster \
  --type=merge \
  -p '{
    "spec": {
      "managementState": "Managed",
      "defaultRoute": false,
      "routes": [
        {
          "name": "public-registry",
          "hostname": "registry.example.com",
          "secretName": "registry-tls"
        }
      ]
    }
  }'
```

值	描述
`public-registry`	路由名称。Operator 使用该名称作为生成的 Ingress 名称。
`registry.example.com`	外部客户端可访问的 Registry 主机名。DNS 记录必须解析到 Ingress 控制器。
`registry-tls`	`image-registry-system` 中的 TLS Secret。证书的 Subject Alternative Name 必须与 Registry 主机名匹配。

验证配置和生成的 Ingress：

```
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl -n image-registry-system get ingress public-registry
kubectl -n image-registry-system describe ingress public-registry
```

预期结果：

- `Config/cluster.spec.routes[]` 包含 `public-registry`。
- `public-registry` Ingress 使用 `registry.example.com`。
- Ingress 引用了 `registry-tls` Secret。

按照配置客户端信任中的说明配置客户端信任，然后验证 TLS 信任并登录：

```
openssl s_client \
  -connect registry.example.com:443 \
  -servername registry.example.com \
  -verify_return_error </dev/null

ac registry login \
  --registry registry.example.com
```

配置客户端信任

如果 Ingress 证书由私有 CA 签名，请在登录以及推送或拉取操作之前，将该 CA 添加到每个外部客户端的信任存储中。

对于使用不安全 Registry 选项的测试客户端，请仅将不安全设置限定到 Registry 主机：

```
ac registry login \
  --registry registry.example.com \
  --insecure
```

如果 OCI 客户端还连接到 HTTP 端点或不受信任的证书，请单独配置该客户端的信任设置或不安全 Registry 设置。

验证外部访问

检查 Ingress：

```
kubectl -n image-registry-system get ingress
kubectl -n image-registry-system describe ingress public-registry
```

从外部客户端写入凭据：

```
ac registry login \
  --registry registry.example.com
```

推送和拉取测试镜像：

```
nerdctl tag my-app:latest registry.example.com/team-a/my-app:external-tes
t
nerdctl push registry.example.com/team-a/my-app:external-test
nerdctl pull registry.example.com/team-a/my-app:external-test
```

故障排除

症状	检查项
Ingress 缺失	`Config.spec.defaultRoute`、`Config.spec.routes[]` 和 Operator 日志。
Ingress 没有地址	Ingress 控制器状态、ALB 项目标签和 Ingress 事件。
TLS 握手失败	TLS Secret、客户端信任存储、证书 SAN 和私有 CA 配置。
登录失败	ACP 凭据、命名空间 RoleBinding、OIDC 运行时发现和 Registry 日志。
登录后推送或拉取失败	镜像命名空间权限、仓库路径、存储后端和 Registry 就绪状态。

管理访问权限和清理

使用此页面执行 Alauda Container Platform Registry 管理员和命名空间管理员任务， 包括命名空间镜像访问、使用情况报告、镜像清理、Registry 垃圾回收以及 镜像签名验证。

目录

任务索引
先决条件
授予命名空间权限
查看使用情况
验证 Image 签名
清理 Image
运行 Registry 垃圾回收

任务索引

任务	使用场景
授予命名空间权限	用户或服务帐户需要在镜像命名空间中执行拉取、推送、删除或访问镜像目录。
查看使用情况	管理员需要镜像或 ImageStream 使用情况信息。

任务	使用场景
验证镜像签名	管理员需要检查或保存受信任的镜像签名条件。
清理镜像	管理员在查看试运行结果后需要删除未使用的镜像元数据。
运行 Registry 垃圾回收	管理员在清理元数据后需要从 Registry 存储中回收未引用的 blob。

先决条件

- ACP Registry 已安装并可用。
- 你具有对目标集群的 `kubectl` 和 `ac` 访问权限。
- 你具有在镜像命名空间中创建 RoleBinding 的权限。
- 你具有使用情况报告、镜像清理、Registry 垃圾回收和签名验证的管理员权限。

使用 `ac` 管理命令前，选择 ACP 4.4 Registry 后端：

```
ac config set-registry-mode modern
```

预期结果：

- 后续的 `ac` Image API、清理、垃圾回收和签名命令将使用 ACP 4.4 Registry。

授予命名空间权限

向用户授予拉取权限：

```
kubectl create rolebinding image-puller-user \
  --clusterrole=system:image-puller \
  --user=<username> \
  -n <image-namespace>
```

向用户授予推送权限：

```
kubectl create rolebinding image-pusher-user \
  --clusterrole=system:image-pusher \
  --user=<username> \
  -n <image-namespace>
```

向另一个命名空间中的 ServiceAccount 授予拉取权限：

```
kubectl create rolebinding image-puller-sa \
  --clusterrole=system:image-puller \
  --serviceaccount=<workload-namespace>:<serviceaccount-name> \
  -n <image-namespace>
```

占位符	描述
`<username>`	需要访问目标命名空间中镜像的 ACP 或 Kubernetes 用户名。
`<image-namespace>`	拥有镜像仓库的命名空间，例如 `team-a`。
`<workload-namespace>`	工作负载 ServiceAccount 运行所在的命名空间。
`<serviceaccount-name>`	拉取镜像的工作负载所使用的 ServiceAccount 名称。

验证 RoleBinding 和有效访问权限：

```
kubectl -n <image-namespace> get rolebinding

kubectl auth can-i get imagestreams/layers.image.alauda.io \
  -n <image-namespace> \
  --as=system:serviceaccount:<workload-namespace>:<serviceaccount-name>
```

预期结果：

- Image 命名空间中存在 RoleBinding。
- 对于应拉取镜像的 ServiceAccount，`kubectl auth can-i` 命令返回 `yes`。

Registry 使用 ImageStream 层授权：

操作	典型角色	**Image API** 权限
拉取	`system:image-puller`	`image.alauda.io` `imagestreams/layers` `get`
推送	`system:image-pusher`	`image.alauda.io` `imagestreams/layers` `update`
删除	`system:image-deleter`	针对目标镜像元数据的 Image API 删除权限
写入签名	`system:image-signature-writer`	`image.alauda.io` `imagesignatures` `create`、`delete`
读取镜像目录	`system:image-registry-catalog-reader`	`image.alauda.io` `imagestreams` `list`
读取 Registry 指标	`system:image-registry-metrics-reader`	`image.alauda.io` `registry/metrics` `get`

`system:image-pruner` ClusterRole 专用于由 Operator 管理的 `pruner` ServiceAccount。它不是命名空间访问角色。手动清理和垃圾回收需要相应的管理员权限。

对于 Pull 和 Push 以外的角色，请根据你的授权模型授予相应的平台角色或 ClusterRole，然后验证有效权限。指标权限是 集群范围的；其他检查（如适用）限定在镜像命名空间中：

```
kubectl auth can-i delete images.image.alauda.io \
  --as=<username>
kubectl auth can-i create imagesignatures.image.alauda.io \
  --as=<username>
kubectl auth can-i list imagestreams.image.alauda.io \
  -n <image-namespace> \
  --as=<username>
kubectl auth can-i get registry/metrics.image.alauda.io \
  --as=<username>
```

预期结果：

- 对于具有相应有效权限的主体，每个命令均返回 `yes`。该 权限也可以由其他角色授予。

查看使用情况

显示 Image 的存储和使用统计信息：

```
ac adm top images
```

显示 ImageStream 的存储和使用统计信息：

```
ac adm top imagestreams
```

预期结果：

- 命令会输出当前管理员可见的 Image 或 ImageStream 资源的使用情况行。

验证 **Image** 签名

验证 Image 对象上记录的镜像签名身份：

```
ac adm verify-image-signature sha256:<digest> \
  --expected-identity=registry.example.com/team-a/demo:latest
```

占位符	描述
`sha256:<digest>`	要验证的 Image 对象摘要。使用 `ac get images` 或 `ac get imagestreamtags <name>:<tag> -n <namespace> -o wide` 查找摘要。
`registry.example.com/team-a/demo:latest`	预期的已签名镜像身份。使用签名策略所要求的镜像引用。

将受信任条件保存回 Image 对象：

```
ac adm verify-image-signature sha256:<digest> \
  --expected-identity=registry.example.com/team-a/demo:latest \
  --save
```

验证已保存的条件：

```
ac get images.image.alauda.io sha256:<digest> -o yaml
```

预期结果：

- Image 对象包含验证命令保存的受信任签名条件。

保存信任条件会更改 Image API 元数据。要回滚已保存的条件，请编辑 Image 对象并移除已保存的条件，或从已知良好的备份中恢复 Image 对象。

仅当不再需要签名记录时，才从 Image 对象中移除所有签名记录：

```
ac adm verify-image-signature sha256:<digest> --remove-all
```

> 警告：`--remove-all` 会移除 Image 对象中的每条签名记录。通过 Registry 无法撤销此操作。执行命令后验证 Image 对象；如果错误地移除了记录，请从已知良好的备份中恢复 元数据。

```
ac get images.image.alauda.io sha256:<digest> -o yaml
```

清理 **Image**

确认执行清理可能会永久移除未使用的镜像元数据。如果同时运行 Registry 垃圾回收，则未引用的 Blob 可能会从存储中永久回收。请在维护时段运行这些命令，保留所需的备份，并在添加 `--confirm` 前查看试运行输出。

预览镜像清理：

```
ac adm prune images
```

预期结果：

- 试运行会列出清理候选项，但不会删除镜像元数据。

查看试运行输出后运行清理：

```
ac adm prune images \
  --keep-tag-revisions=5 \
  --keep-younger-than=72h \
  --confirm
```

验证清理：

```
ac adm prune images \
  --keep-tag-revisions=5 \
  --keep-younger-than=72h
```

预期结果：

- 后续试运行不再列出已由确认命令清理的元数据。

使用 `--whitelist` 和允许列表模式排除存储库：

```
# Preview the effect of the allow-list pattern
ac adm prune images \
  --whitelist='^cpaas-system/.*'

# Confirm pruning only after reviewing the preview
ac adm prune images \
  --whitelist='^cpaas-system/.*' \
  --confirm
```

如需配置计划清理，请参阅设置和配置 Registry。

确认执行的清理不会自动还原。需要恢复时，请从备份中恢复已清理的元数据，从受信任的镜像源重新创建元数据，或重新导入镜像。

运行 **Registry** 垃圾回收

Registry 垃圾回收会从 Registry 存储中回收未被引用的 blob。确认后，无法从 Registry 中还原已回收的 blob。

在清理元数据后预览 Registry 垃圾回收：

```
ac adm registry gc
```

预期结果：

- 命令会预览符合条件的未引用 blob，但不会删除它们。

仅在查看预览后运行 Registry 垃圾回收：

```
ac adm registry gc --confirm
```

预期结果：

- 命令会从 Registry 存储中删除符合条件的未引用 blob。

也可以将 Registry 垃圾回收作为清理的一部分触发：

```
ac adm prune images --confirm --prune-registry
```

`ac adm prune images` 和 `ac adm registry gc` 默认都以试运行模式执行。请先查看预览，然后再添加 `--confirm`。确认回收 blob 后，无法从 Registry 中还原；如果所需的 blob 已被回收，请从存储备份中恢复数据，或从受信任的源重新复制镜像。

确认完成垃圾回收后，通过拉取或检查关键镜像来验证它们：

```
mkdir -p /tmp/registry-auth
ac registry login \
  --registry registry.example.com \
  --to /tmp/registry-auth/config.json

ac image info registry.example.com/team-a/demo:latest \
  --registry-config=/tmp/registry-auth/config.json
```