# 注册表管理

## 概览

概览

## Registry 数据备份和恢复

Registry 数据备份和恢复

## Registry v2 管理

Registry v2：镜像仓库概览

Registry v2：镜像 Registry Ope

Registry v2

Registry v2：公开 Registry

Registry v2：管理访问和清理

## 安装

使用 **YAML** 安装

通过 **Web Console** 安装

## 升级

升级 **Registry**

# 概览

ACP Registry 文档有两条管理路径。对于由 operator 管理的 ACP Registry，请使用 **Registry v2**。仅当环境仍在运行 Registry 集群插件时，才使用旧版 Registry 页面。

开发人员和应用团队通过开发人员镜像工作流使用 registry。

## 目录



## 管理范围

| 需求 | 继续使用 |
| --- | --- |
| 管理由 operator 管理的 Registry v2。 | Registry v2 |
| 安装旧版 Registry 集群插件。 | 安装 |
| 升级旧版 Registry 集群插件。 | 升级 |
| 备份或恢复旧版 Registry 集群插件数据。 | Registry 备份和恢复 |
| 在工作负载或开发人员工作流中使用镜像。 | 镜像 和 registry 镜像使用 |

# 命名空间隔离和访问

Registry 镜像仓库按命名空间组织。拉取和推送访问取决于授予目标命名空间中用户或 ServiceAccounts 的平台角色和权限。

有关常用镜像使用命令和跨命名空间拉取访问，请参见 Registry 镜像使用。

# 灵雀云容器平台 **Registry** 数据备份和恢复

## 目录



## 概述

当 registry 镜像数据存储在 S3 兼容对象存储中时，可使用此操作步骤对 ACP Registry 数据进行备份和恢复。

此恢复模型将存储在 S3 中的镜像数据与 Kubernetes `ModuleInfo` 自定义资源中定义的 Cluster Plugin 配置分离开来。

- 备份：从 `ModuleInfo` 资源中获取 S3 配置，并备份指定存储桶中的数据。
- 恢复：在新集群或修复后的集群中安装 Cluster Plugin 后，更新 `ModuleInfo` 资源中的 S3 配置，使其指向包含已恢复数据的存储桶。

这种模型使数据备份和恢复独立于 Cluster Plugin 的部署和升级操作。所有连接信息都通过声明式 `ModuleInfo` 资源进行管理。

# 前提条件

- 已具备 `kubectl` 访问权限，并拥有在目标 Kubernetes 集群上执行操作的适当权限。
- 已具备用于访问和操作镜像数据所使用的 S3 兼容存储的凭据和客户端工具（例如 awscli、rclone、minio-client）。
- 已安装并配置 ACP Registry Cluster Plugin，且其 `ModuleInfo` 资源已存在并处于健康状态。
- 已准备好独立且容量充足的存储用于备份数据（例如另一个 S3 存储桶）。

# 数据备份

在备份过程中，获取当前生产环境的 S3 配置，并对存储桶中的镜像数据执行完整备份。

## 第 1 步：获取当前 S3 配置

从管理 ACP Registry Cluster Plugin 的 `ModuleInfo` 资源中提取 S3 存储配置。
此信息是执行备份操作的基础。

在 `global` 集群上运行以下命令：

```
# 1. Identify the ModuleInfo resource name for the image-registry module
MODULE_INFO_NAME=$(kubectl get moduleinfo -l cpaas.io/module-name=image-registry -o jsonpath='{.items[0].metadata.name}')
echo "Target ModuleInfo Resource: $MODULE_INFO_NAME"

# 2. Extract key S3 configuration information
S3_BUCKET=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.bucket}')
S3_ENDPOINT=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.regionEndpoint}')
S3_REGION=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.region}')
S3_SECRET_NAME=$(kubectl get moduleinfo $MODULE_INFO_NAME -o jsonpath='{.spec.config.s3storage.secretName}')

# 3. Obtain access keys from the Secret (typically access-key-id and secret-access-key)
# Note: The output is Base64 encoded and needs to be decoded accordingly.
kubectl get secret -n cpaas-system $S3_SECRET_NAME -o jsonpath='{.data}'
```

关键变量说明：

- `S3_BUCKET`：实际存储镜像数据的源存储桶名称。
- `S3_ENDPOINT`：连接 S3 兼容服务的端点 URL。
- `S3_REGION`：S3 服务的地域标识符。
- `S3_SECRET_NAME`：存储认证密钥的 Kubernetes Secret 名称。

## 第 2 步：执行 S3 存储桶数据备份

使用任意 S3 客户端工具，结合上一步获取的配置，对源存储桶数据执行完整备份。

使用以下备份逻辑：

- 使用端点（$S3_ENDPOINT）、地域（$S3_REGION），以及从 Secret 解码得到的 access key 和 secret key 配置客户端。
- 执行 sync 或 copy 命令，将源存储桶（$S3_BUCKET）中的所有数据备份到已准备好的独立备份位置（例如另一个 S3 存储桶或路径）。
- 记录备份时间戳、所使用的存储桶名称和端点，并将这些信息与备份文件一起归档。

# 数据恢复

在恢复之前，请先在目标环境中安装 ACP Registry Cluster Plugin，例如新集群或修复后的集群。然后修改配置，使 registry 能够访问已恢复的镜像数据。

## 第 1 步：准备备份数据

使用任意 S3 客户端工具，将已备份的镜像数据恢复到一个明确可访问的目标 **S3** 存储 `bucket` 中。例如，可恢复到名为 `registry-bucket-restored` 的新存储桶中。确保你对该目标存储桶具有写权限。

## 第 2 步：更新 **ModuleInfo** 配置

恢复的关键在于更新新 `cluster plugin` 的 `ModuleInfo` 资源，使其 S3 配置指向包含备份数据的目标存储桶。

1. 确定新的 **S3** 连接信息：
   - `NEW_BUCKET`：已恢复备份数据的目标存储桶名称（例如，registry-bucket-restored）。
   - `NEW_ENDPOINT`：目标 **S3** 服务的端点。如果 S3 服务地址与备份时相同，则保持不变。
   - `NEW_REGION`：目标 **S3** 服务的地域。
   - `NEW_SECRET_NAME`：具有对目标存储桶读/写权限的 Kubernetes Secret 名称。如果 access key 未更改，则仍为 `$S3_SECRET_NAME`。
2. 更新 **ModuleInfo** 资源：
   使用 `kubectl patch` 命令直接更新 `ModuleInfo` 的 S3 配置部分。平台控制器将自动将此更改同步到所有相关的 Deployment、Pod 及其他资源。

```
# Execute the configuration update
kubectl patch moduleinfo $MODULE_INFO_NAME --type=merge -p '{
  "spec": {
    "config": {
      "s3storage": {
        "bucket": "'"$NEW_BUCKET"'",
        "regionEndpoint": "'"$NEW_ENDPOINT"'",
        "region": "'"$NEW_REGION"'",
        "secretName": "'"$NEW_SECRET_NAME"'"
      }
    }
  }
}'
```

要点：此操作会触发 ACP Registry Pods 的滚动更新。新启动的 Pods 将使用新配置连接到指定的目标存储桶。

# 验证

更新完成后，请按照以下步骤验证数据恢复是否成功以及服务是否正常运行。

## 检查 `global` 集群中的模块状态

```
# Check if Pods have successfully restarted and are running with the new configuration
kubectl get pods -n cpaas-system -l app=image-registry
# Observe Pod logs to confirm no S3 connection errors
kubectl logs -n cpaas-system -l app=image-registry -c registry --tail=50
```

## 验证数据访问（**API** 测试）

使用 Registry 的 API 接口直接验证其是否能够读取已恢复的镜像数据。

```
# Obtain the Registry Service access address (assuming ClusterIP type)
REGISTRY_SVC_IP=$(kubectl get svc -n cpaas-system image-registry -o jsonp
ath='{.spec.clusterIP}')

# Test 1: Query the catalog of repositories
curl -s http://$REGISTRY_SVC_IP/v2/_catalog | jq .
# Expected success return: {"repositories":["image1","image2",...]}

# Test 2: Query the tag list for a specific image (e.g., an image named `
myns/nginx`)
curl -s http://$REGISTRY_SVC_IP/v2/myns/nginx/tags/list | jq .
# Expected success return: {"name":"myns/nginx","tags":["v1.0","lates
t",...]}
```

## 功能测试

尝试从已恢复的 registry 中拉取一个已知镜像，或向其中推送一个新镜像，以完整验证读/写功能。

# 其他存储后端

本操作步骤以 S3 存储为示例。在具备相应备份和恢复工具的情况下，相同的恢复模型也可应用于 Registry 支持的其他存储后端，例如本地文件系统、StorageClass 或 NAS。

无论存储类型如何，首先都应从 `ModuleInfo` 资源中对应的配置块（例如 `s3storage` 或 `persistence`）提取存储连接参数。然后使用适当的存储工具备份数据。在恢复期间，将数据恢复到目标位置，并更新 `ModuleInfo` 中对应的配置字段，使新部署的 registry 实例使用该位置。

# Registry v2 管理

使用本节可安装、配置、暴露和管理由 Operator 管理的 ACP Registry。

Registry v2 部署在 `image-registry-system` 中，并由 `cluster-image-registry-operator` 进行协调。它使用 `Config/cluster`、`ImagePruner/cluster`、`image.alauda.io/v1` Image API 以及受管 service account 拉取 secret。

## 目录



## 管理主题

| 需求 | 继续查看 |
| --- | --- |
| 查看 Registry v2 架构和兼容性说明。 | 镜像仓库概览 |
| 安装 Image Registry Operator 或检查 Operator 状态。 | Image Registry Operator |
| 配置存储、镜像限制、拉取 secret 和计划清理。 | 设置并配置 registry |
| 将 Registry v2 暴露给外部客户端。 | 暴露 registry |
| 管理 namespace 镜像访问、用量报告、清理、垃圾回收和签名验证。 | 管理访问和清理 |

| 需求 | 继续查看 |
| --- | --- |
| 从工作负载或开发人员工作站使用 Registry v2。 | Registry v2 镜像使用 |

## Registry v2：镜像仓库概览

集成镜像仓库

与旧版 Registry 相比的变化

常用术语

自动镜像裁剪

兼容性说明

## Registry v2：镜像 Registry Ope

主要组件

安装 Operator

更改 Registry 管理状态

Image Pruner 协调

检查 Operator 和 Registry 状态

检查 Registry 日志和 metrics 访问

常见 Operator 问题

## Registry v2

前提条件

配置开发环境存

配置 PVC 存储

配置兼容 S3 的

配置受管的 Ser

配置镜像限制

配置计划的镜像

## Registry v2：公开 Registry

前提条件

公开默认 Registry

公开自定义安全 Registry Host

配置客户端信任

验证外部访问

故障排除

## Registry v2：管理访问和清理

任务索引

前提条件

授予命名空间权限

查看使用情况

验证镜像签名

清理镜像

运行 Registry 垃圾回收

# Registry v2：镜像仓库概览

**Registry v2** 是由 Operator 管理的、面向 ACP 集群的集成镜像仓库。它提供内部镜像存储、ImageStream 元数据、基于命名空间的访问控制、受管 service account pull 凭证以及镜像裁剪。

## 目录



## 集成镜像仓库

该仓库作为集群工作负载运行在 `image-registry-system` 中。`image-registry` Deployment 负责处理 OCI push 和 pull 流量，而镜像元数据则通过聚合后的 `image.alauda.io/v1` Image API 提供。

镜像数据和镜像元数据分开存储：

| 数据类型 | 存储位置 |
| --- | --- |
| 镜像 blob 和 manifest | 在 `Config/cluster.spec.storage` 中配置的存储后端，例如 PVC、S3 兼容存储、`emptyDir` 或其他受支持的后端。 |
| 镜像元数据 | ACP Image API 资源，例如 `Image`、`ImageStream`、`ImageStreamTag`、`ImageStreamImage` 和 `ImageSignature`。 |

该仓库与 ACP 认证和 Kubernetes 授权集成。命名空间 RoleBinding 控制谁可以拉取、推送、删除、列出或裁剪镜像内容。

# 与旧版 **Registry** 相比的变化

| 领域 | 旧版 **Registry** | **Registry v2** |
| --- | --- | --- |
| 运行时命名空间 | 通常为 `cpaas-system` | `image-registry-system` |
| 内部服务地址 | `image-registry.cpaas-system.svc` | `image-registry.image-registry-system.svc:5000` |
| 生命周期管理 | 平台插件或 chart 管理的 Registry 资源 | OLM 和 `cluster-image-registry-operator` |
| 期望状态 | 插件配置和旧版 Registry 配置 | `configs.imageregistry.operator.alauda.io/cluster` 和 `imagepruners.imageregistry.operator.alauda.io/clus` |
| 镜像元数据 | Registry HTTP 视图和旧版元数据 API | 聚合后的 `image.alauda.io/v1` Image API |
| 外部暴露 | 旧版 ingress 或 gateway 配置 | `Config.spec.defaultRoute` 和 `Config.spec.routes[]` 染为 Kubernetes Ingress |
| pull 凭证 | 旧版 service account pull secret 自动化 | 内置的 Operator imagePullSecret controller |

| 领域 | 旧版 **Registry** | **Registry v2** |
| --- | --- | --- |
| 镜像限制 | 旧版 Registry gateway ConfigMap | 带有 `alauda.io` 镜像资源的 Kubernetes `LimitRange` 和 `ResourceQuota` |
| 裁剪与 GC | 旧版 `ac` Registry 命令和脚本 | `ImagePruner/cluster`、`image-pruner` CronJob，以及 `adm prune images` / `ac adm registry gc` |

# 常用术语

| 术语 | 含义 |
| --- | --- |
| Image repository | 以命名空间为范围的镜像 tag 和 digest 集合，地址格式为 `<namespace>/<repository>:<tag>`。 |
| ImageStream | ACP Image API 资源，用于记录某个 repository 的 tag 规范和 tag 历史。 |
| Image | 按 digest 作用域划分的集群级镜像元数据。 |
| ImagePruner | 用于配置定期裁剪作业的单例自定义资源。 |
| Managed pull secret | 由 Operator 生成并注入的 service account pull 凭证。 |
| Registry storage | 用于存储镜像 blob 和 manifest 的后端。 |

# 自动镜像裁剪

Registry v2 使用 `imagepruners.imageregistry.operator.alauda.io/cluster` 来配置定期裁剪。Operator 会渲染一个 `image-pruner` CronJob，并使用配置的保留策略运行 `ac adm prune images`。

裁剪会首先删除未使用的镜像元数据。Registry garbage collection 会在元数据被删除后回收存储空间。

# 兼容性说明

- 对于仍在使用旧版 Registry 的环境，旧版 Registry 的操作指南页面仍然有效。
- Registry v2 Image API 资源使用 API group `image.alauda.io/v1` 和 `imageregistry.operator.alauda.io/v1`。
- Registry v2 使用 ACP Image API 资源，例如 `Image`、`ImageStream`、`ImageStreamTag` 和 `ImagePruner`。
- 使用 `ac` 执行 ACP Registry 工作流。

# Registry v2：镜像 Registry Operator

Image Registry Operator 安装并管理集群范围的 Registry v2 实例。它会协调来自 `Config/cluster` 的 Registry 运行时资源，以及来自 `ImagePruner/cluster` 的计划清理资源。

## 目录



## 主要组件

| 组件 | 用途 |
| --- | --- |
| `cluster-image-registry-operator` Deployment | 协调来自 `Config/cluster` 和 `ImagePruner/cluster` 的单例 Registry。 |
| `image-registry` Deployment | 提供 OCI push 和 pull 流量、身份验证、授权、存储访问、健康检查和 metrics。 |
| `image-api-server` Deployment | 通过 Kubernetes API aggregation 提供 ACP Image API。 |
| `APIService/v1.image.alauda.io` | 在 Kubernetes API server 中注册 `image.alauda.io/v1`。 |
| `node-ca` DaemonSet | 将 Registry CA 信任和 Registry service 主机映射分发到各节点。 |
| `image-pruner` CronJob | 运行计划清理和垃圾回收工作流。 |
| Managed imagePullSecret controller | 为内部 registry 创建、注入、刷新和移除 service account pull secret。 |

# 安装 Operator

当 Operator package 在 OperatorHub 中可用时，请从 ACP web console 安装 Image Registry Operator。仅在 web console 不可用时，才将 YAML 路径用于受控自动化、恢复或支持指导下的安装。

不要使用旧版 Registry Cluster Plugin 来安装 Registry v2。旧版 Registry Cluster Plugin 仅对旧版 Registry 部署保留可用。

## 从 OperatorHub 安装

1. 登录 ACP，并进入 **Administrator** 页面。
2. 在左侧导航栏中，单击 **Marketplace** > **OperatorHub**。
3. 搜索 **Image Registry Operator** 或 `cluster-image-registry-operator`。
4. 单击 **Install**。

5. 在安装页面中，除非你的发布指南另有说明，否则使用以下设置：

| 参数 | 推荐值 |
| --- | --- |
| Channel | `stable` |
| Installation Mode | `Cluster` |
| Namespace | `image-registry-system` |
| Upgrade Strategy | `Manual` |

6. 单击 **Install**。
7. 如果出现批准提示，请审查并批准生成的安装计划。
8. 等待直到 Operator 状态为 **Installed**。

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

如果安装命名空间不存在，请先创建它：

```
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
```

创建一个名为 `image-registry-operator-subscription.yaml` 的文件：

```
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
```

# 更改 **Registry** 管理状态

通过将 `Config/cluster.spec.managementState` 设置为 `Managed` 来启用 Registry：

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

验证 Registry 数据平面是否可用：

```
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`。

将管理状态设置为 `Removed` 会停止 Registry v2 运行时组件。在 Registry 被移除期间，push 和 pull 流量、Image API server 以及计划清理都不可用。仅在维护窗口或支持指导下的恢复过程中使用此状态。

在切换到 `Removed` 之前，请备份所需的图像数据，并检查存储管理模式：

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

仅当 `spec.storage.managementState` 为 `Unmanaged`，或者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，才假定图像数据会被保留。

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"}}'
```

验证数据平面 Deployment 已被移除：

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

预期结果：

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

# Image Pruner 协调

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

该 CronJob 默认使用 Registry v2 内部 service URL。

# 检查 Operator 和 Registry 状态

```
kubectl -n image-registry-system get subscription,csv,installplan
kubectl -n image-registry-system get deploy cluster-image-registry-operator 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` 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 metrics 访问

检查 Registry Pods：

```
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 registry
```

从 monitoring service account 检查 metrics 访问：

```
kubectl auth can-i get pods -n image-registry-system \
  --as=system:serviceaccount:cpaas-system:prometheus-sa
```

# 常见 Operator 问题

| 症状 | 检查 |
| --- | --- |
| CSV 不是 `Succeeded` | `Subscription`、`InstallPlan`、CSV events，以及 `cpaas-system` catalog source。 |
| `Config/cluster` 为 `Degraded=True` | `Config.status.conditions`、Operator 日志、Registry Pod events、存储、TLS Secret 和 RBAC。 |
| Registry Pod 处于 pending | PVC 绑定、节点资源、node selectors、taints、tolerations 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。 |

# Registry v2：设置和配置 Registry

使用此页面配置存储、Registry 请求行为、service account 拉取凭据、镜像限制以及计划的镜像清理，适用于 **Registry v2**。

## 目录



## 前提条件

- 已安装 Registry v2，且 `Config/cluster` 已存在。
- 你有权限更新 `configs.imageregistry.operator.alauda.io`、`imagepruners.imageregistry.operator.alauda.io`、`LimitRange`、`ResourceQuota` 以及相关的 Kubernetes 资源。
- 对于持久化存储，请在配置 `Config/cluster` 之前准备好存储后端和所需凭据。

# 配置开发环境存储

`emptyDir` 将镜像数据存储在临时 Pod 存储中。仅在所有已推送的镜像数据都可以在 Registry Pod 重建时丢弃的开发或测试环境中使用它。不要在生产环境或任何必须保留镜像的环境中使用 `emptyDir`。

Patch `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
      },
      "resources": {
        "requests": {
          "cpu": "500m",
          "memory": "500Mi"
        },
        "limits": {
          "cpu": "500m",
          "memory": "500Mi"
        }
      }
    }
  }'
```

验证配置并进行 rollout：

```
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 rollout。

# 配置 PVC 存储

生产环境请使用持久化后端。创建一个名为 `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 提供动态供给的 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
```

Patch `Config/cluster` 以使用 PVC。请使用 merge patch，这样 `Config/cluster.spec` 中的其他字段（例如 routes 或 pull-secret 设置）不会被移除。`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": {
        "managementState": "Unmanaged",
        "emptyDir": null,
        "pvc": {
          "claim": "image-registry"
        },
        "s3": null
      },
      "resources": {
        "requests": {
          "cpu": "500m",
          "memory": "500Mi"
        },
        "limits": {
          "cpu": "500m",
          "memory": "500Mi"
        }
      }
    }
  }'
```

验证 PVC、配置和 rollout：

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

- `image-registry` Deployment 成功完成 rollout。

# 配置兼容 S3 的存储凭据

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

```
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>
```

| 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 secret access key。请仅将其存储在 Kubernetes Secret 中。 |

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`。

Patch `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,
        "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 Registry blob 的现有对象存储 bucket 或等效容器。 |
| `<region>` | 存储后端所需的 region 值。请使用提供商的值，或 S3 兼容服务要求的值。 |
| `<s3-endpoint>` | 可从 Registry Pod 访问的 S3 兼容端点，不包含 bucket 名称。 |
| `<trusted-ca-configmap>` | `image-registry-system` 中包含对象存储端点 CA 证书的 ConfigMap。如果该端点使用 Registry Pod 信任的公共 CA，则省略 `trustedCA`。 |

验证配置并进行 rollout：

```
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 日志中没有存储认证、bucket、endpoint 或证书错误。

# 配置受管的 **Service Account Pull Secret**

Operator 包含一个受管的 imagePullSecret controller。当 `Config/cluster` 受管时，该 controller 可以为内部 Registry 创建、注入、刷新和移除 service account pull secret。

通过附加主机或忽略的命名空间来 patch `Config/cluster`。此 patch 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` 包含已配置的管理状态、附加主机和被忽略的命名空间设置。

# 配置镜像限制

在 Registry v2 中，镜像大小和标签数量限制通过 Kubernetes `LimitRange` 和 `ResourceQuota` 对象表示。不要在 Registry v2 部署中使用旧的 Registry gateway limit ConfigMap。

创建一个名为 `team-a-image-quota.yaml` 的文件，用于命名空间级配额：

```
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
```

创建一个名为 `team-a-image-limits.yaml` 的文件，用于每个镜像和每个 ImageStream 的限制：

```yaml
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` 包含已配置的 Image API 限制。
- `LimitRange` 包含已配置的 `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。在启用计划之前，请先确认保留策略，因为清理会删除未使用的镜像元数据。

创建一个名为 `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 pruner 资源和渲染后的 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` 包含已配置的 schedule 和 retention policy。
- Operator 会在 `image-registry-system` 中渲染 `image-pruner` CronJob。

清理会删除未使用的镜像元数据。当需要回收 blob 存储空间时，请单独运行 registry garbage collection。

有关手动清理和 registry garbage collection 命令，请参见 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、节点附加以及后端存储可用性。
- 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`，或者存储后端是否已更改。
- 在确认数据保留决策之前，不要删除 PVC、PV 或对象存储数据。

# Registry v2：公开 Registry

默认情况下，请为集群内的 workload 使用内部 registry service。只有当开发人员机器、CI systems 或其他外部客户端必须 push 或 pull images 时，才公开 **Registry v2**。

## 目录



## 前提条件

- Registry v2 已安装并可用。
- 您有权限更新 `Config/cluster`，并在 `image-registry-system` 中创建或引用 TLS Secrets。
- 集群具有可将流量路由到 `image-registry-system` 的 Ingress controller。
- 对于自定义 host，请准备 DNS 记录和 TLS certificate，其 Subject Alternative Name 与 Registry hostname 匹配。

# 公开默认 **Registry**

在 `Config/cluster` 中启用默认 external endpoint：

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

Operator 会在 `image-registry-system` 中生成一个名为 `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 host：

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

echo "$REGISTRY_HOST"
```

预期结果：

- `Config/cluster.spec.defaultRoute` 为 `true`。
- `image-registry-system` 中存在 `default-route` Ingress。
- `REGISTRY_HOST` 包含分配给默认路由的 hostname。

如果 Ingress certificate 由私有 CA 签名，请在登录之前将 CA certificate 添加到 external client 和 OCI client 的信任存储中。请从 external client 验证 TLS chain：

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

为生成的 host 写入凭据：

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

# 公开自定义安全 **Registry Host**

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

当设置了 `secretName` 时，TLS Secret 必须存在于 `image-registry-system` 中。

生成的 Ingress 使用 registry service 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 merge patch，以免删除其他 `Config/cluster.spec` 字段，例如 storage 和 pull-secret 设置。此 patch 中的 `routes` 数组会替换当前的 routes 数组，因此请包含所有必须保留配置的自定义路由：

```
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 将其用作生成的 Ingress 名称。 |
| `registry.example.com` | 外部客户端可访问的 Registry hostname。DNS 记录必须解析到 Ingress controller。 |
| `registry-tls` | `image-registry-system` 中的 TLS Secret。证书的 Subject Alternative Name 必须与 Registry hostname 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。

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 certificate 由私有 CA 签名，请在登录以及 push 或 pull 操作之前，将 CA 添加到每个外部客户端的信任存储中。

对于使用 insecure registry 选项的测试客户端，请将不安全设置仅限定到 Registry host：

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

如果 OCI client 还连接到 HTTP endpoint 或不受信任的 certificate，请单独配置该 OCI client 的 trust 或 insecure-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
```

push 和 pull 一个测试 image：

```
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 |
| --- | --- |
| 缺少 Ingress | `Config.spec.defaultRoute`、`Config.spec.routes[]` 和 Operator 日志。 |
| Ingress 没有地址 | Ingress controller 状态、ALB project label 和 Ingress events。 |
| TLS 握手失败 | TLS Secret、客户端信任存储、certificate SANs 以及 private CA 配置。 |
| 登录失败 | ACP 凭据、namespace RoleBinding、OIDC runtime discovery 和 Registry 日志。 |
| 登录后 push 或 pull 失败 | image namespace 权限、repository path、storage backend 以及 Registry 就绪状态。 |

# Registry v2：管理访问和清理

本页用于执行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

| 任务 | 适用场景 |
| --- | --- |
| 授予命名空间权限 | 用户或 service account 需要在镜像命名空间中获得 pull、push、delete 或 prune 访问权限。 |
| 查看使用情况 | 管理员需要镜像或 ImageStream 的使用信息。 |

| 任务 | 适用场景 |
| --- | --- |
| 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 | 管理员需要在查看 dry run 之后移除未使用的镜像元数据。 |
| 运行 Registry 垃圾回收 | 管理员需要在元数据清理后，从 Registry 存储中回收未被引用的 blob。 |

# 前提条件

- 已安装并可用 Registry v2。
- 你已拥有到目标集群的 `kubectl` 和 `ac` 访问权限。
- 你有权限在镜像命名空间中创建 RoleBinding。
- 你拥有使用情况报告、镜像清理、Registry 垃圾回收和签名验证所需的管理员权限。

# 授予命名空间权限

向用户授予 pull 权限：

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

向用户授予 push 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 service account 授予 pull 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 service account 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：

- RoleBinding 存在于镜像命名空间中。
- `kubectl auth can-i` 命令对应该拉取镜像的 service account 返回 `yes`。

Registry v2 使用 ImageStream 层授权：

| 操作 | 常见角色 | **Image API** 权限 |
| --- | --- | --- |
| Pull | `system:image-puller` | `image.alauda.io` `imagestreams/layers` `get` |
| Push | `system:image-pusher` | `image.alauda.io` `imagestreams/layers` `update` |

| 操作 | 常见角色 | **Image API** 权限 |
| --- | --- | --- |
| Delete | `system:image-deleter` | 目标镜像元数据的 Image API delete 权限 |
| Prune | `system:image-pruner` | Image API prune 和层检查权限 |

# 查看使用情况

显示 Images 的存储和使用统计信息：

```
ac adm top images
```

显示 ImageStreams 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。

# 验证镜像签名

验证记录在 Image 对象上的镜像签名标识：

```
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 对象的摘要。使用 `ac get images` 或 `ac get imagestreamtags <name>:<tag> -n <namespace> -o wide` 查找该摘要。 |
| `registry.example.com/team-a/demo:latest` | 期望的已签名镜像标识。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 对象包含由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，或者从已知良好的备份中恢复该 Image 对象。

# 清理镜像

确认执行的清理可以永久移除未使用的镜像元数据。如果你还运行 Registry 垃圾回收，则可以从存储中永久回收未被引用的 blob。请在维护窗口期间运行这些命令，保留所需备份，并在添加 `--confirm` 之前查看 dry-run 输出。

预览镜像清理：

```
ac adm prune images
```

预期结果：

- dry run 会列出可清理候选项，并且不会删除镜像元数据。

在查看 dry-run 输出后执行清理：

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

验证清理结果：

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

预期结果：

- 后续的 dry run 将不再列出由已确认命令清理掉的元数据。

通过使用 `--whitelist` 来排除匹配 allow-list 模式的仓库：

```
ac adm prune images \
  --whitelist='^cpaas-system/.*' \
  --confirm
```

有关计划性清理，请参阅 设置并配置 registry。

已确认的清理不可自动回滚。若需要恢复，请从备份中还原已清理的元数据、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 垃圾回收会回收 Registry 存储中未被引用的 blob。blob 回收在确认后无法从 Registry 中撤销。

在完成元数据清理后运行 Registry 垃圾回收：

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

预期结果：

- 不使用 `--confirm` 时，命令会预览垃圾回收候选项。
- 使用 `--confirm` 时，命令会从存储中移除符合条件的未引用 blob。

你也可以将 Registry 垃圾回收作为清理过程的一部分触发：

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

`ac adm prune images` 和 `ac adm registry gc` 默认都是 dry-run。请在添加 `--confirm` 之前查看预览结果。

在确认执行垃圾回收后，通过拉取或检查来验证关键镜像：

```
ac image info registry.example.com/team-a/demo:latest
```

如果回收了必需的 blob，请从后端存储备份中恢复，或再次从受信任的来源复制该镜像。

# 安装

# 使用 YAML 安装

## 目录



## 使用场景

在以下场景中使用 YAML 安装：

- 高级用户，具备 Kubernetes 经验，并且偏好手动方式。
- 需要外部管理存储的部署，例如 NAS、S3-compatible 对象存储或 Ceph。
- 需要对 TLS 和 ingress 进行细粒度控制的环境。
- 用于高级配置的完整 **YAML** 自定义。

# 前提条件

- 将 灵雀云容器平台 **Registry** 集群插件安装到目标集群。
- 已配置 kubectl，并且可以访问目标 Kubernetes 集群。
- 具有创建集群级资源的集群管理员权限。
- 获取已注册的域名，例如 `registry.example.com`。有关域名配置，请参见 创建域名。
- 提供有效的 **NAS** 存储，例如 NFS。
- 可选：提供有效的 **S3-compatible** 存储。

# 使用 **YAML** 安装 灵雀云容器平台 **Registry**

## 操作步骤

1. 创建一个名为 `registry-plugin.yaml` 的 **ClusterPluginInstance** 清单文件，并使用以下模板：

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ClusterPluginInstance
metadata:
  annotations:
    cpaas.io/display-name: image-registry
  labels:
    create-by: cluster-transformer
    manage-delete-by: cluster-transformer
    manage-update-by: cluster-transformer
  name: image-registry
spec:
  config:
    access:
      address: ''
      enabled: false
    fake:
      replicas: 2
    infra:
      enabled: false
    global:
      expose: false
      isIPv6: false
      replicas: 2
      oidc:
        ldapID: ''
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 250m
          memory: 256Mi
    ingress:
      enabled: true
      hosts:
        - name: <YOUR-DOMAIN> # [REQUIRED] Customize domain
          tlsCert: <NAMESPACE>/<TLS-SECRET> # [REQUIRED] Namespace/Secr
etName
      ingressClassName: '<INGRESS-CLASS-NAME>' # [REQUIRED] IngressClas
sName
      insecure: false
    persistence:
      accessMode: ReadWriteMany
```

```
      nodes: ''
      path: <YOUR-HOSTPATH> # [REQUIRED] Local path for LocalVolume
      size: <STORAGE-SIZE> # [REQUIRED] Storage size (e.g., 10Gi)
      storageClass: <STORAGE-CLASS-NAME> # [REQUIRED] StorageClass name
      type: StorageClass
    registryLimitConfig:
      enabled: false
      configMapName: image-registry-limit-config
    s3storage:
      bucket: <S3-BUCKET-NAME> # [REQUIRED] S3 bucket name
      enabled: false # Set false for local storage
      env:
        REGISTRY_STORAGE_S3_SKIPVERIFY: false # Set true for self-signe
d certs
      region: <S3-REGION> # S3 region
      regionEndpoint: <S3-ENDPOINT> # S3 endpoint
      secretName: <S3-CREDENTIALS-SECRET> # S3 credentials Secret
    service:
      nodePort: ''
      type: ClusterIP
  pluginName: image-registry
```

2. 根据你的环境自定义以下字段：

```
spec:
  config:
    global:
      oidc:
        ldapID: '<LDAP-ID>' # LDAP ID
    infra:
      enabled: false  # If you want to deploy components to the infra n
odes. Default is false means all nodes.
    ingress:
      hosts:
        - name: '<YOUR-DOMAIN>' # e.g., registry.your-company.com
          tlsCert: '<NAMESPACE>/<TLS-SECRET>' # e.g., cpaas-system/tls-
secret
      ingressClassName: '<INGRESS-CLASS-NAME>' # e.g., cluster-alb-1
    persistence:
      size: '<STORAGE-SIZE>' # e.g., 10Gi
      storageClass: '<STORAGE-CLASS-NAME>' # e.g., cpaas-system-storage
    registryLimitConfig:
      enabled: true # Set true to enable registry push limits
      configMapName: 'image-registry-limit-config' # Pre-created Config
Map name
    s3storage:
      bucket: '<S3-BUCKET-NAME>' # e.g., prod-registry
      region: '<S3-REGION>' # e.g., us-west-1
      regionEndpoint: '<S3-ENDPOINT>' # e.g., https://s3.amazonaws.com
      secretName: '<S3-CREDENTIALS-SECRET>' # Secret containing S3 acce
ss credentials
      env:
        REGISTRY_STORAGE_S3_SKIPVERIFY: 'true' # Set "true" for self-si
gned certs
```

3. 如何为 **S3** 凭据创建 **secret**：

```
kubectl create secret generic <S3-CREDENTIALS-SECRET> \
  --from-literal=access-key-id=<YOUR-S3-ACCESS-KEY-ID> \
  --from-literal=secret-access-key=<YOUR-S3-SECRET-ACCESS-KEY> \
  -n cpaas-system
```

将 `<S3-CREDENTIALS-SECRET>` 替换为你的 S3 凭据 secret 名称。

4. 可选：启用 **registry push** 限制，用于限制镜像大小和 tag 数量。

此能力由内置的 Registry proxy 提供。
要启用它：

- 将 `spec.config.registryLimitConfig.enabled` 设置为 `true`。
- 将 `spec.config.registryLimitConfig.configMapName` 设置为你在 Registry 命名空间中手动创建的 ConfigMap 名称。

示例：

```
spec:
  config:
    registryLimitConfig:
      enabled: true
      configMapName: image-registry-limit-config
```

说明：

- 该 ConfigMap 不由 Registry 插件创建。
- 对于新部署，推荐的 ConfigMap 名称为 `image-registry-limit-config`。
- 运行时仍然出于向后兼容性接受旧的 ConfigMap 名称 `registry-gateway-config`。
- 有关详细的 ConfigMap 示例、规则行为和验证步骤，请参见 配置 Registry Push 限制。

启用此功能后，请在继续执行第 5 步之前应用 ConfigMap：

```
kubectl apply -f image-registry-limit-config.yaml
```

5. 将 Registry 插件清单应用到你的集群。
   此命令会创建或更新名为 `image-registry` 的 `ClusterPluginInstance` 资源，从而安装或更新 Registry 集群插件。
   如果启用了 `registryLimitConfig`，请在此步骤之前应用 配置 Registry Push 限制 中所述的限制 ConfigMap。

```
kubectl apply -f registry-plugin.yaml
```

# 配置参考

## 常用字段

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| `spec.config.global.oidc.ldapID` | 用于 OIDC 身份验证的 LDAP ID | `lda` |
| `spec.config.ingress.hosts[0].name` | 用于访问 registry 的自定义域名 | `reg` |
| `spec.config.ingress.hosts[0].tlsCert` | TLS 证书 secret 引用（namespace/secret-name） | `cpa` `tls` |
| `spec.config.ingress.ingressClassName` | registry 的 Ingress class 名称 | `clu` |
| `spec.config.persistence.size` | registry 的存储大小 | `10G` |
| `spec.config.persistence.storageClass` | registry 的 StorageClass 名称 | `nfs` |
| `spec.config.registryLimitConfig.enabled` | 启用用于限制镜像大小和 tag 数量的 registry push 限制 | `tru` |
| `spec.config.registryLimitConfig.configMapName` | 包含限制规则的手动创建 ConfigMap 名称 | `ima` `conf` |
| `spec.config.s3storage.bucket` | 用于镜像存储的 S3 bucket 名称 | `pro` |
| `spec.config.s3storage.region` | S3-compatible 存储的地域标识符 | `us-` |
| `spec.config.s3storage.regionEndpoint` | S3-compatible 服务端点 URL | `htt` |
| `spec.config.s3storage.secretName` | 包含 S3 凭据的 secret | `s3-` |

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| `spec.config.s3storage.env.REGISTRY_STORAGE_S3_SKIPVERIFY` | 自签名证书时设置为 `true` | `tru` |
| `spec.config.infra.enabled` | 将组件部署到 infra 节点或所有节点 | `fal` |

### 验证

1. 检查插件：

```
kubectl get clusterplugininstances image-registry -o yaml
```

2. 验证 registry Pod：

```
kubectl get pods -n cpaas-system -l app=image-registry
```

## 更新或卸载 灵雀云容器平台 **Registry**

### 更新

在 global 集群上执行以下命令，并根据上面提供的参数说明更新资源中的值，以完成更新：

```
# <CLUSTER-NAME> is the cluster where the plugin is installed
kubectl edit -n cpaas-system \
  $(kubectl get moduleinfo -n cpaas-system -l cpaas.io/cluster-name=<CLUS
TER-NAME>,cpaas.io/module-name=image-registry -o name)
```

### 卸载

在 global 集群上执行以下命令：

```
# <CLUSTER-NAME> is the cluster where the plugin is installed
kubectl get moduleinfo -n cpaas-system -l cpaas.io/cluster-name=<CLUSTER-NAME>,cpaas.io/module-name=image-registry -o name | xargs kubectl delete -n cpaas-system
```

# 通过 Web Console 安装

## 目录



## 适用场景

在以下场景中，建议使用 Web Console 安装路径：

- 首次使用的用户，希望采用引导式、可视化界面。
- 在非生产环境中进行快速概念验证部署。
- 团队对 **Kubernetes** 的经验有限，希望采用更简化的部署流程。
- 需要最少自定义的场景（例如默认存储配置）。
- 基础网络配置，且没有特定的 ingress 规则。
- 用于高可用的 **StorageClass** 配置。

当你需要高级的 S3 兼容存储配置或特定的 ingress 规则时，请使用 YAML 安装路径。

## 前提条件

- 使用 Cluster Plugin 机制，将 灵雀云容器平台 **Registry** 集群插件安装到目标集群。

# 使用 **Web Console** 安装 灵雀云容器平台 **Registry**

## 操作步骤

1. 登录并进入 **Administrator** 页面。
2. 单击 **Marketplace** > **Cluster Plugins**，进入 **Cluster Plugins** 列表页。
3. 找到 灵雀云容器平台 **Registry** 集群插件，单击 **Install**，然后进入安装页面。
4. 按照以下说明配置参数，并单击 **Install** 完成部署。

参数说明如下：

| 参数 | 说明 |
| --- | --- |
| **Expose Service** | 启用后，管理员可以通过访问地址在外部管理镜像仓库。这会带来显著的安全风险，请务必谨慎启用。 |
| **Enable IPv6** | 当集群使用 IPv6 单栈网络时，启用此选项。 |
| **NodePort** | 启用 Expose Service 后，配置主机端口，以允许通过此端口从外部访问 Registry。 |
| **Storage Type** | 选择存储类型。支持的类型：LocalVolume 和 StorageClass。 |
| **Nodes** | 选择用于运行 Registry 服务、执行镜像存储和分发的节点。（仅当 Storage Type 为 LocalVolume 时可用） |
| **StorageClass** | 选择一个 StorageClass。当副本数超过 1 时，请选择支持 RWX（ReadWriteMany）能力的存储（例如文件存储），以确保高可用性。（仅当 Storage Type 为 StorageClass 时可用） |
| **Storage Size** | 分配给 Registry 的存储容量（单位：Gi）。 |

| 参数 | 说明 |
| --- | --- |
| **Replicas** | 配置 Registry Pod 的副本数：<br>• **LocalVolume**：默认值为 1（固定）<br>• **StorageClass**：默认值为 3（可调整） |
| **Resource Requirements** | 定义 Registry Pod 的 CPU 和 Memory 资源请求与限制。 |

## 验证

1. 进入 **Marketplace** > **Cluster Plugins**，确认插件状态显示为 **Installed**。
2. 单击插件名称查看其详情。
3. 复制 **Registry Address**，并使用 OCI 客户端（例如 `nerdctl`）推送或拉取镜像。

# 更新或卸载 灵雀云容器平台 **Registry**

你可以从列表页或详情页更新或卸载 灵雀云容器平台 **Registry** 插件。

# 升级

## 升级 **Registry**

# 升级 灵雀云容器平台 **Registry**

## 目录



## 先决条件

1. ACP 的管理员权限。
2. 旧插件是指 ACP v4.1 及更早版本。
3. 新插件是指 ACP v4.2 及更高版本。

## 概述

根据 Registry 存储类型，将旧 Registry 插件升级到目标 Registry 插件。由于集群插件名称已更改，因此需要手动干预。

在下面的命令中，请将 `<LEGACY-PLUGIN-NAME>` 替换为 ACP v4.1 及更早版本中使用的 Registry 插件名称。

你可以通过检查现有的 Registry 插件资源来识别它（在 global 集群中执行）：

```
kubectl get moduleplugins -n cpaas-system
```

# 升级步骤

## 环境检查和预处理

### 未安装旧插件

如果从未安装过旧插件，你只需从 [Marketplace/集群插件] 中清理旧插件：

```
# Execute in Global cluster
kubectl delete moduleplugins <LEGACY-PLUGIN-NAME>
kubectl get moduleconfig -l cpaas.io/module-name=<LEGACY-PLUGIN-NAME> -o
name | xargs kubectl delete
```

### 已安装旧插件

如果旧插件已安装在任意集群中，请根据你的存储类型执行相应的迁移操作。

## S3 存储迁移

特性：自动迁移数据，无需手动操作

操作步骤：

1. 备份旧插件配置：（在安装了旧插件的业务集群中执行）

```
kubectl get clusterplugininstances <LEGACY-PLUGIN-NAME> -o yaml > backup-clusterplugininstances.yaml
```

2. 卸载旧插件：（在 global 集群中执行）

```
# Replace <CLUSTER-NAME> with actual cluster name which has the plugin installed
kubectl get moduleinfo -l cpaas.io/cluster-name=<CLUSTER-NAME>,cpaas.io/module-name=<LEGACY-PLUGIN-NAME> -o name | xargs kubectl delete
```

3. 安装新插件（在将安装该插件的业务集群中执行）

- 编辑备份文件 `backup-clusterplugininstances.yaml`，将所有 `<LEGACY-PLUGIN-NAME>` 替换为 `image-registry`
- 应用新配置：

```
kubectl apply -f backup-clusterplugininstances.yaml
```

4. 从 [Marketplace/集群插件] 中清理旧插件：（在 global 集群中执行）

```
kubectl delete moduleplugins <LEGACY-PLUGIN-NAME>
kubectl get moduleconfig -l cpaas.io/module-name=<LEGACY-PLUGIN-NAME> -o name | xargs kubectl delete
```

## NFS 存储迁移

特性：需要手动保留 PV 并修改回收策略。

操作步骤：

1. 记录 PV 信息：（在安装了旧插件的业务集群中执行）

```
OLD_PV=$(kubectl get pvc <LEGACY-PLUGIN-NAME> -n cpaas-system -o jsonpath='{.spec.volumeName}')
echo "Old PV Name: $OLD_PV"
```

2. 修改 PV 回收策略：（在安装了旧插件的业务集群中执行）

```
kubectl patch pv $OLD_PV -p '{"spec":{"persistentVolumeReclaimPolic
y":"Retain"}}'
```

3. 备份旧插件配置（同 S3 第 1 步）
4. 卸载旧插件（同 S3 第 2 步）
5. 释放 PV 绑定：（在安装了旧插件的业务集群中执行）

```
kubectl patch pv $OLD_PV -p '{"spec":{"claimRef":null}}'
```

6. 安装新插件（重要：必须为新插件设置 `config.persistence.volumeName`，以保留旧数据）

- 编辑配置文件 `backup-clusterpluginins tances.yaml`，设置 `config.persistence.volumeName: <OLD_PV>`
- 应用配置（同 S3 第 3 步）

7. 从 [Marketplace/集群插件] 中清理旧插件（同 S3 第 4 步）

## 本地存储迁移

特性：需要手动复制数据

操作步骤：

1. 备份旧插件配置（同 S3 第 1 步）
2. 卸载旧插件（同 S3 第 2 步）
3. 安装新插件（同 S3 第 3 步）
4. 数据迁移：

- 使用 ssh 登录到旧插件 Pod 所在节点，将旧目录中的数据复制到新目录，并保留所有文件权限：

```
sudo cp -a /cpaas/<LEGACY-PLUGIN-NAME>/* /cpaas/image-registry/
```

5. 从 [Marketplace/集群插件] 中清理旧插件（同 S3 第 4 步）