Alauda Container Platform Registry Administration

Use this section to install, configure, expose, and administer Alauda Container Platform Registry for ACP 4.4.

The Alauda Container Platform Registry is deployed in `image-registry-system` and reconciled by `cluster-image-registry-operator`. It uses `Config/cluster`, `ImagePruner/cluster`, the `image.alauda.io/v1` Image API, and managed service account pull secrets.

> **Important:** For Alauda Container Platform Registry versions earlier than `v4.4`, including ACP 4.3 and earlier, contact technical support for upgrade and migration plans.

TOC

Administration Topics

Administration Topics

Need	Continue with
Review Registry architecture, repository naming, and common terms.	Registry overview
Install or upgrade Alauda Container Platform Registry, or check Operator status.	Alauda Container Platform Registry
Configure storage, image limits, pull secrets, and scheduled pruning.	Setting up and configuring the registry
Expose the Registry for external clients.	Exposing the registry
Manage namespace image access, usage reporting, pruning, garbage collection, and signature verification.	Managing access and cleanup
Use the Registry from workloads or developer workstations.	Registry image use

Alauda Container Platform Reg

- Integrated Image Registry
- Repository Names
- Common Terms
- Automatic Image Pruning
- Operational Notes

Alauda Container Platform Reg

- Main Components
- Install Alauda Container Platform Registry
- Upgrade Alauda Container Platform Regis
- Change the Registry Management State
- Image Pruner Reconciliation
- Check Operator and Registry Status
- Check Registry Logs and Metrics Access
- Common Operator Issues

Setting Up a

- Prerequisites
- Configure Deve
- Configure PVC
- Configure S3-C
- Configure Clou
- Configure Requ
- Configure Mana
- Configure Imag
- Sche
- orag

Exposing the Registry

- Prerequisites
- Expose the Default Registry
- Expose a Custom Secure Registry Host
- Configure Client Trust
- Verify External Access

Troubleshooting

Managing Access and Cleanup

Task Index

Prerequisites

Grant Namespace Permissions

View Usage

Verify Image Signatures

Prune Images

Run Registry Garbage Collection

Alauda Container Platform Registry Overview

Alauda Container Platform Registry is the integrated image registry for ACP 4.4 clusters. It provides internal image storage, ImageStream metadata, namespace-based access control, managed service account pull credentials, and image pruning.

TOC

Integrated Image Registry
Repository Names
Common Terms
Automatic Image Pruning
Operational Notes

Integrated Image Registry

The Registry runs as a cluster workload in `image-registry-system` and is managed by the Image Registry Operator. The `image-registry` Deployment serves OCI push and pull traffic, while image metadata is served through the aggregated `image.alauda.io/v1` Image API.

Image data and image metadata are stored separately:

Data type	Storage location
Image blobs and manifests	The storage backend configured in `Config/cluster.spec.storage`: `emptyDir`, `PVC`, `S3`, `Swift`, `GCS`, `IBMCOS`, or `Azure`.
Image metadata	ACP Image API resources such as `Image`, `ImageStream`, `ImageStreamTag`, `ImageStreamImage`, and `ImageSignature`.

The Registry integrates with ACP authentication and Kubernetes authorization. Namespace RoleBindings control who can pull, push, delete, or list image content. Image pruning and registry garbage collection are administrator and Operator-managed operations.

Repository Names

Image repositories are namespace-scoped and use the following format:

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

Repository names must not contain additional path segments after the namespace. For example, `registry.example.com/team-a/my-app:v1` is valid, but `registry.example.com/team-a/backend/my-app:v1` is not supported.

This restriction is enforced by the Image API, not by the registry. The OCI registry accepts a push to `team-a/backend/my-app` and stores the manifest, but the Image API cannot represent the repository because an ImageStream name must be a DNS-1123 subdomain. Such an image has no `ImageStream`, `Image`, or `ImageStreamTag` metadata and is not visible to Registry workflows. Always use exactly one repository segment after the namespace.

Common Terms

Term	Meaning
Image repository	A namespace-scoped collection of image tags and digests, addressed as `<namespace>/<repository>:<tag>`.
ImageStream	ACP Image API resource that records tag specifications and tag history for a repository.
Image	Cluster-scoped image metadata for a digest.
ImagePruner	Singleton custom resource that configures scheduled prune jobs.
Managed pull secret	A service account pull credential generated and injected by the Operator.
Registry storage	The backend that stores image blobs and manifests.

Automatic Image Pruning

The Registry uses `imagepruners.imageregistry.operator.alauda.io/cluster` to configure scheduled pruning. The Operator renders an `image-pruner` CronJob that runs `ac adm prune images` with the configured retention policy.

Pruning removes unused image metadata first. Registry garbage collection reclaims storage after metadata is removed.

Operational Notes

- Registry resources use the API groups `image.alauda.io/v1` and `imageregistry.operator.alauda.io/v1`.
- The Registry uses Image API resources such as `Image`, `ImageStream`, `ImageStreamTag`, and `ImageSignature`, plus the Operator resource `ImagePruner`.
- Use `ac` for ACP Registry workflows.

Alauda Container Platform Registry

The Image Registry Operator installs and manages the cluster-wide Alauda Container Platform Registry instance. It reconciles Registry runtime resources from `Config/cluster` and scheduled pruning resources from `ImagePruner/cluster`.

TOC

Main Components
Install Alauda Container Platform Registry
Install from OperatorHub
Install by Using YAML
Prerequisites
Upgrade Alauda Container Platform Registry
Change the Registry Management State
Image Pruner Reconciliation
Check Operator and Registry Status
Check Registry Logs and Metrics Access
Common Operator Issues

Main Components

Component	Purpose
`cluster-image-registry-operator` Deployment	Reconciles the singleton Registry from `Config/cluster` and `ImagePruner/cluster`.
`image-registry` Deployment	Serves OCI push and pull traffic, authentication, authorization, storage access, health checks, and metrics.
`image-api-server` Deployment	Serves the ACP Image API through Kubernetes API aggregation.
`APIService/v1.image.alauda.io`	Registers `image.alauda.io/v1` with the Kubernetes API server.
`node-ca` DaemonSet	Distributes Registry CA trust and Registry service host mappings to nodes.
`image-pruner` CronJob	Runs scheduled image pruning and, while the Registry is managed, registry garbage collection.
Managed imagePullSecret controller	Creates, injects, refreshes, and removes service account pull secrets for the internal registry.

Install Alauda Container Platform Registry

Install the Image Registry Operator to deploy Alauda Container Platform Registry from the ACP web console when the Operator package is available in OperatorHub. Use the YAML path only for controlled automation or support-guided installation when the web console is not available.

Both install paths require the `cluster-image-registry-operator` package to be present in the target CatalogSource. See Install by Using YAML for how to verify the package and upload it if it is missing.

Install from OperatorHub

1. Log in to ACP and navigate to the **Administrator** page.

2. In the left navigation bar, click **Marketplace** > **OperatorHub**.
3. Search for **Alauda Container Platform Registry** or `cluster-image-registry-operator`.
4. Click **Install**.
5. On the installation page, use the following settings unless your release guidance states otherwise:

Parameter	Recommended value
Channel	`stable`
Installation Mode	`All namespaces on the cluster`
Namespace	`image-registry-system`
Upgrade Strategy	`Manual`

6. Click **Install**.
7. If an approval prompt appears, review and approve the generated install plan.
8. Wait until the Operator status is **Installed**.

Verify the installation:

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

Expected results:

- The installed CSV is `Succeeded`.
- The `cluster-image-registry-operator` Deployment is available.

Install by Using YAML

Prerequisites

The `cluster-image-registry-operator` package must already be present in the target CatalogSource. A fresh platform installation does not include it, and the `check-subscription.cpaas.io` admission webhook rejects the `Subscription` when the package is missing:

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

Verify the package before continuing:

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

If it is missing, upload the Operator package first, following the platform's Operator package upload procedure. For example:

```
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>"
```

The push must use a registry account with `push` permission on the target repositories. An account with pull-only access fails with `requested access to the resource is denied`. The push does not configure image pull credentials for the Operator's own workloads; see the pull-secret note in the install steps below.

Create the installation namespace if it does not exist:

```
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
```

Create a file named `image-registry-operator-subscription.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
```

Apply the `Subscription`:

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

Approve the generated `InstallPlan`:

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

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

Wait for the Operator:

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

The Operator then creates the `image-registry`, `image-api-server`, and `node-ca` workloads. These pull their images from the platform registry, so the ServiceAccounts the Operator uses must carry a working registry pull Secret. If a workload stays in `ImagePullBackOff` with `insufficient_scope: authorization failed`, confirm that the platform registry pull Secret is attached to those ServiceAccounts:

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

Attach the Secret if it is missing, then restart the workloads:

```
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
```

Upgrade Alauda Container Platform Registry

Upgrade Alauda Container Platform Registry by upgrading the Image Registry Operator through OperatorHub. The Operator upgrade updates the `cluster-image-registry-operator` Deployment and then reconciles the existing `Config/cluster`, `ImagePruner/cluster`, Registry, Image API, and supporting resources.

Before upgrading, confirm that the current Registry is healthy.

> **Important:** For Alauda Container Platform Registry versions earlier than `v4.4`, including ACP 4.3 and earlier, contact technical support for upgrade and migration plans.

```
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
```

Set the Subscription approval strategy to `Manual` before making a new Operator package available:

```
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>`	Subscription for the Alauda Container Platform Registry Operator. Use the name returned by `kubectl -n image-registry-system get subscription`.

Make the target Operator package available in **Administrator** > **Marketplace** > **OperatorHub**. If the package is not present, follow the platform's Operator package upload procedure before continuing.

In OperatorHub, open **Alauda Container Platform Registry**, select the target channel and version, and review the pending upgrade. Approve the generated InstallPlan only after verifying that the Subscription uses `Manual` approval:

```
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}}'
```

Wait for the new CSV and Operator Deployment, then verify that the Registry data plane reconciles:

```
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-serv
er --timeout=300s
kubectl get configs.imageregistry.operator.alauda.io cluster -o yaml
kubectl get imagepruners.imageregistry.operator.alauda.io cluster -o yaml
```

Expected results:

- The new CSV is `Succeeded` and the Subscription reports the new installed CSV.
- `cluster-image-registry-operator`, `image-registry`, and `image-api-server` are available.
- Existing storage, routes, Image API metadata, and ImagePruner configuration remain reconciled.
- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.

> **Note:** A CSV upgrade reapplies the Operator's ServiceAccount. If a registry pull Secret was attached to that ServiceAccount during installation, the upgrade can clear it, and the new `cluster-image-registry-operator` Pod then stays in `ImagePullBackOff` with `insufficient_scope: authorization failed`. Re-attach the Secret and restart the 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
```

Do not delete the Subscription, `Config/cluster`, storage objects, or Image API metadata as part of an Operator upgrade. If the upgrade does not complete, inspect the Subscription, InstallPlan, CSV events, Operator logs, and `Config/cluster.status.conditions` before taking further action.

Change the Registry Management State

Configure one supported storage backend before enabling the Registry. If `Config/cluster.spec.storage` is empty, setting `managementState` to `Managed` leaves the Operator in `StorageNotConfigured`, and the Registry data plane cannot be reconciled.

Enable the Registry by setting `Config/cluster.spec.managementState` to `Managed`:

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

Verify that the Registry data plane is available:

```
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
```

Expected result:

- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.

> **Warning:** Setting the management state to `Removed` stops the Registry runtime components. Push and pull traffic and the Image API server are unavailable while the Registry is removed. The Operator can also leave the `image-pruner` CronJob present, but disables registry garbage collection for the removed Registry. Do not rely on scheduled cleanup while the Registry is removed. Use this state only during a maintenance window or for support-guided recovery.

Before switching to `Removed`, back up required image data and inspect the storage management mode:

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

Only assume image data is retained when `spec.storage.managementState` is `Unmanaged`, or when the storage administrator has confirmed that the backend storage reclaim policy preserves the data after the Registry instance is removed. Back up required image data before changing the management state.

To stop the Registry, set the management state to `Removed`:

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

Verify that the data-plane deployments are removed:

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

Expected result:

- The `image-registry` and `image-api-server` Deployments are not present while the Registry is removed.

Image Pruner Reconciliation

The Operator reconciles the singleton `ImagePruner/cluster` into an `image-pruner` CronJob in `image-registry-system`. Configure the retention policy in the `ImagePruner` resource. See Setting up and configuring the registry.

The CronJob uses the Registry internal service URL by default.

Check Operator and Registry Status

This section assumes the Registry has been enabled by setting `Config/cluster.spec.managementState` to `Managed`, as described in Change the Registry Management State. `spec.managementState` has no default value. While it is unset the Operator reports `Available=True` with reason `Removed`, and the `image-registry` and `image-api-server` Deployments do not exist.

```
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
```

Expected results:

- The Operator CSV is `Succeeded`.
- `cluster-image-registry-operator`, `image-registry`, and `image-api-server` are available.
- `node-ca` is ready on target nodes.
- `APIService/v1.image.alauda.io` is `Available=True`.
- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.

Check Registry Logs and Metrics Access

Check Registry Pods:

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

View Registry logs:

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

Check metrics access from a monitoring service account.

`registry/metrics` is an RBAC-only resource. It is not exposed by the `image.alauda.io/v1` API discovery document, and `kubectl auth can-i` resolves resource names through discovery. As a result `kubectl auth can-i get registry/metrics.image.alauda.io` always reports `no`, even when the permission is granted. Use a `SubjectAccessReview` instead:

```
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
```

Expected result:

- Each review reports `true`.

The Operator binds `system:image-registry-metrics-reader` to `prometheus-sa` and `vm-sa` through the `registry-monitoring` ClusterRoleBinding. Those ServiceAccounts are created by the monitoring plugin, so the reviews report `false` until monitoring is installed.

Common Operator Issues

Symptom	Check
CSV is not `Succeeded`	`Subscription`, `InstallPlan`, CSV events, and the `cpaas-system` catalog source.
`Config/cluster` is `Degraded=True`	`Config.status.conditions`, Operator logs, Registry Pod events, storage, TLS Secret, and RBAC.
Registry Pod is pending	PVC binding, node resources, node selectors, taints, tolerations, and topology constraints.
`node-ca` is not ready	DaemonSet scheduling, Pod logs, node trust configuration, and host mapping updates.

Setting Up and Configuring the Registry

Use this page to configure storage, Registry request behavior, service account pull credentials, image limits, and scheduled image cleanup for Alauda Container Platform Registry.

TOC

Prerequisites
Configure Development Storage
Configure PVC Storage
Configure S3-Compatible Storage Credentials
Configure Cloud Storage Backends
Configure Request Handling
Configure Managed Service Account Pull Secrets
Configure Image Limits
Configure Scheduled Image Pruning
Operate Storage

Prerequisites

- ACP Registry is installed, and `Config/cluster` exists.

- You have permission to update `configs.imageregistry.operator.alauda.io`, `imagepruners.imageregistry.operator.alauda.io`, `LimitRange`, `ResourceQuota`, and related Kubernetes resources.
- For persistent storage, prepare the storage backend and required credentials before configuring `Config/cluster`.

Configure Development Storage

> **Warning:** `emptyDir` stores image data on ephemeral Pod storage. It is deleted when the Registry Pod is removed from its node and cannot be used with more than one Registry replica. Use it only for development or test environments where all pushed image data can be discarded. Do not use `emptyDir` for production or for any environment that must retain images.

Patch `Config/cluster`. The `null` fields remove mutually exclusive storage backends from the current configuration:

```
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"
        }
      }
    }
  }'
```

Verify the configuration and rollout:

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

Expected results:

- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.
- The `image-registry` Deployment rolls out successfully.

Configure PVC Storage

Use a persistent backend for production. The example uses `ReadWriteOnce`, one Registry replica, and the `Recreate` rollout strategy required by the Operator for RWO storage. Create a file named `image-registry-pvc.yaml`:

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

Placeholder	Description
`<storage-class-name>`	StorageClass that provisions the Registry PVC. Run `kubectl get storageclass` to list available values. For multi-replica Registry deployments, choose storage that supports the required access mode.

Apply the PVC:

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

Patch `Config/cluster` to use the PVC. Use a merge patch so that other `Config/cluster.spec` fields, such as routes or pull-secret settings, are not removed. The `null` fields remove mutually exclusive storage backends from the current configuration:

```
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"
        }
      }
    }
  }'
```

Verify the PVC, configuration, and 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
```

Expected results:

- The `image-registry` PVC is `Bound`.
- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.
- The `image-registry` Deployment rolls out successfully.

Configure S3-Compatible Storage Credentials

Create the user-managed storage Secret before configuring `Config/cluster`. The Operator merges this Secret into the Registry private configuration:

```
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 -
```

Placeholder	Description
`<access-key-id>`	Access key ID for the S3-compatible storage account. Obtain it from the storage administrator.
`<secret-access-key>`	Secret access key for the S3-compatible storage account. Store it only in the Kubernetes Secret.

Verify the Secret without printing its contents:

```
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
```

Expected results:

- The Secret exists in `image-registry-system`.

- The second command returns a non-zero byte count. Do not print or decode the Secret data.

Use `disableRedirect: true` when clients cannot reach the object storage endpoint directly and all content must be served through the Registry.

If the S3 endpoint uses a private CA, create the referenced ConfigMap with the required key:

```
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 -
```

Verify the ConfigMap key without displaying the certificate:

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

Expected result:

- The command returns a non-zero byte count for `ca-bundle.crt`.

Patch `Config/cluster`. The `null` fields remove mutually exclusive storage backends from the current configuration:

```
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
    }
  }'
```

Placeholder	Description
`<bucket-name>`	Existing object storage bucket or bucket-equivalent container used for Registry blobs.
`<region>`	Region value required by the storage backend. Use the provider value, or the value required by the S3-compatible service.
`<s3-endpoint>`	S3-compatible endpoint host and optional port reachable from the Registry Pod, without `https://` and without the bucket name. The example supplies the `https://` scheme.

Placeholder	Description
`<trusted-ca-configmap>`	ConfigMap in `image-registry-system` that contains the CA certificate for the object storage endpoint. Omit `trustedCA` when the endpoint uses a public CA trusted by the Registry Pod.

The current operator supports the following storage fields: `emptyDir`, `pvc`, `s3`, `swift`, `gcs`, `ibmcos`, and `azure`. Do not configure other storage fields.

> **Note:** The `Config` CRD also exposes a `storage.oss` field, inherited from the upstream image registry operator. This operator does not implement it. Setting `storage.oss` is treated as "no storage backend configured": the Operator reports `Degraded=True` with reason `StorageNotConfigured`, and the Registry data plane does not reconcile. Use only the seven supported fields listed above.

Verify the configuration and 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
```

Expected results:

- `Config/cluster` reports `Available=True`, `Progressing=False`, and `Degraded=False`.
- The Registry logs do not show storage authentication, bucket, endpoint, or certificate errors.

Configure Cloud Storage Backends

The Operator can use Swift, GCS, IBM COS, or Azure storage when the corresponding platform integration and credentials are available. Configure exactly one storage field in `Config/cluster.spec.storage`; use the same merge-patch pattern shown above and set the other supported storage fields to `null`.

Backend	Main `Config/cluster.spec.storage` fields	Required user-managed Secret data
Swift	`swift.authURL`, `swift.authVersion`, `swift.container`, `swift.domain`, `swift.domainID`, `swift.tenant`, `swift.tenantID`, and `swift.regionName`	`REGISTRY_STORAGE_SWIFT_USERNAME` and `REGISTRY_STORAGE_SWIFT_PASSWORD`, or th credential keys `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` and `REGISTRY_STORAGE_SWIFT_APPLICATIONCRE` or `REGISTRY_STORAGE_SWIFT_TOKENID`.
GCS	`gcs.bucket`, `gcs.region`, `gcs.projectID`, and optional `gcs.keyID`	`REGISTRY_STORAGE_GCS_KEYFILE` containin account key file.
IBM COS	`ibmcos.bucket`, `ibmcos.location`, `ibmcos.resourceGroupName`, `ibmcos.resourceKeyCRN`, and `ibmcos.serviceInstanceCRN`	`REGISTRY_STORAGE_IBMCOS_IAMAPIKEY`.
Azure	`azure.accountName`, `azure.container`, `azure.cloudName`, and optional `azure.networkAccess`	`REGISTRY_STORAGE_AZURE_ACCOUNTKEY`.

Provider integrations can populate platform-specific values such as region, project, account, or endpoint information. Confirm the required provider resources and credentials with the storage administrator. After configuring a backend, verify the `Config/cluster` conditions and the Registry rollout as shown in the storage procedures above. Do not print Secret data while troubleshooting.

Configure Request Handling

Use `Config/cluster.spec.readOnly` to reject image pushes and deletes. Use the `requests` settings to limit concurrent reads and writes, queued requests, and queue wait time:

```
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"
        }
      }
    }
  }'
```

The `proxy.http`, `proxy.https`, and `proxy.noProxy` fields configure Registry egress when the cluster proxy settings are not sufficient. `logLevel` accepts `Normal`, `Debug`, `Trace`, or `TraceAll`. Resource requests and limits, node selection, tolerations, affinity, and topology spread are also supported by `Config/cluster.spec`.

Verify request handling settings and the rollout:

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

Expected results:

- `Config/cluster.spec.readOnly` and `Config/cluster.spec.requests` contain the requested values.
- The Registry Deployment rolls out successfully.

Configure Managed Service Account Pull Secrets

The Operator includes a managed imagePullSecret controller. When `Config/cluster` is managed, the controller can create, inject, refresh, and remove service account pull secrets for the internal registry.

Patch `Config/cluster` with additional hosts or ignored namespaces. The array fields in this patch replace the current arrays, so include every host and namespace that must remain configured:

```
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
      }
    }
  }'
```

Verify the configuration:

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

Expected result:

- `Config/cluster.spec.imagePullSecret` contains the configured management state, additional hosts, and ignored namespace settings.

The controller reconciles pull credentials asynchronously. Verify the managed RoleBinding and a ServiceAccount after the controller has had time to reconcile:

```
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
```

Expected results:

- The RoleBinding grants the `system:image-puller` ClusterRole to service accounts in the namespace.
- The ServiceAccount lists at least one managed pull Secret after asynchronous reconciliation.
- At least one Secret with the managed-pull-secret label exists. Use `kubectl get secret` to inspect metadata only; do not print or decode credential data.

Placeholder	Description
`<workload-namespace>`	Namespace where managed ServiceAccount pull credentials should be injected. It must not be listed in `ignoredNamespaces` and must satisfy `ignoreSystemNamespaces`.

Configure Image Limits

In ACP Registry, image size and tag-count limits are represented with Kubernetes `LimitRange` and `ResourceQuota` objects.

Create a file named `team-a-image-quota.yaml` for namespace-level quota:

```yaml
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"
```

Apply the quota:

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

Create a file named `team-a-image-limits.yaml` for per-image and per-ImageStream limits:

```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"
```

Apply the limits:

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

Verify the quota and limits:

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

Expected results:

- The `ResourceQuota` contains the configured Image API limits.
- The `LimitRange` contains the configured `alauda.io/Image` and `alauda.io/ImageStream` limits.

Configure Scheduled Image Pruning

Create or update the singleton `ImagePruner/cluster` to configure scheduled image pruning. Confirm the retention policy before enabling the schedule because pruning removes unused image metadata.

Create a file named `image-pruner.yaml`:

```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
```

Apply the configuration:

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

Verify the pruner resource and rendered CronJob:

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

Expected results:

- `ImagePruner/cluster` contains the configured schedule and retention policy.
- The Operator renders the `image-pruner` CronJob in `image-registry-system`.

The generated CronJob runs `ac adm prune images --confirm`. When the Registry management state is `Managed`, the Operator adds `--prune-registry=true`, so the scheduled job also performs registry garbage collection after image metadata pruning. For a manually run prune, add `--prune-registry` only when blob reclamation is intended; otherwise run garbage collection as a separate reviewed operation.

For manual pruning and registry garbage collection commands, see Managing access and cleanup.

Operate Storage

For PVC-backed Registry storage:

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

Common actions:

- If a PVC is pending, check StorageClass, access mode, capacity, quotas, and events.
- If a Registry Pod cannot mount storage, check PV binding, node attachment, and backend storage availability.

- If image metadata exists but blob data is missing, verify whether the Registry used `emptyDir` or whether the storage backend was changed.
- Do not delete PVCs, PVs, or object storage data until the data retention decision is confirmed.

Exposing the Registry

By default, use the internal service for workloads inside the cluster. Expose Alauda Container Platform Registry only when developer machines, CI systems, or other external clients must push or pull images.

TOC

Prerequisites
Expose the Default Registry
Expose a Custom Secure Registry Host
Configure Client Trust
Verify External Access
Troubleshooting

Prerequisites

- ACP Registry is installed and available.
- You have permission to update `Config/cluster` and create or reference TLS Secrets in `image-registry-system`.
- The cluster has an Ingress controller that can route traffic to `image-registry-system`.
- For a custom host, prepare a DNS record and a TLS certificate whose Subject Alternative Name matches the Registry hostname.

Expose the Default Registry

Enable the default external endpoint in `Config/cluster`:

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

The Operator renders a Kubernetes Ingress named `default-route` in `image-registry-system`.

Verify the generated Ingress and Registry configuration:

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

Get the generated host:

```
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"
```

Expected results:

- `Config/cluster.spec.defaultRoute` is `true`.
- The `default-route` Ingress exists in `image-registry-system`.
- A non-empty `spec.rules[].host` is a usable Registry hostname. An empty rule host means that the Ingress is a catch-all rule; a status address identifies the Ingress controller endpoint and is not automatically a Registry DNS name.

If the default Ingress has no usable hostname, configure a custom secure host instead. Do not use an empty rule host as the value of `--registry`.

When a usable host is available, assign it to `REGISTRY_HOST`. If the client uses a private CA, configure that CA before running the TLS check. The check must succeed with the client's normal trust settings before logging in:

```
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
```

Write credentials for the generated host:

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

Expose a Custom Secure Registry Host

Create or provide a TLS Secret in `image-registry-system`, then configure `Config.spec.routes[]`.

Create the Secret from a certificate whose Subject Alternative Name matches the custom hostname:

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

The TLS Secret must exist in `image-registry-system` when `secretName` is set. Create a DNS record for `registry.example.com` that resolves to the Ingress controller.

The generated Ingress uses the registry service as the backend and sets the backend protocol to HTTPS.

Patch `Config/cluster` with the custom route. Use a merge patch so other `Config/cluster.spec` fields, such as storage and pull-secret settings, are not removed. The `routes` array in this patch replaces the current routes array, so include every custom route that must remain configured:

```
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"
        }
      ]
    }
  }'
```

Value	Description
`public-registry`	Route name. The Operator uses it as the generated Ingress name.

Value	Description
`registry.example.com`	Registry hostname reachable by external clients. The DNS record must resolve to the Ingress controller.
`registry-tls`	TLS Secret in `image-registry-system`. The certificate Subject Alternative Name must match the Registry hostname.

Verify the configuration and generated 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
```

Expected results:

- `Config/cluster.spec.routes[]` contains `public-registry`.
- The `public-registry` Ingress uses `registry.example.com`.
- The Ingress references the `registry-tls` Secret.

Configure client trust as described in Configure Client Trust, then verify TLS trust and login:

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

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

Configure Client Trust

If the Ingress certificate is signed by a private CA, add the CA to each external client's trust store before login and push or pull operations.

For a test client that uses an insecure registry option, scope the insecure setting to the registry host only:

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

Configure the OCI client's trust or insecure-registry setting separately if that client also connects to an HTTP endpoint or an untrusted certificate.

Verify External Access

Check the Ingress:

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

Write credentials from an external client:

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

Push and pull a test 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
```

Troubleshooting

Symptom	Check
Ingress is missing	`Config.spec.defaultRoute`, `Config.spec.routes[]`, and Operator logs.
Ingress has no address	Ingress controller status, ALB project label, and Ingress events.
TLS handshake fails	TLS Secret, client trust store, certificate SANs, and private CA configuration.
Login fails	ACP credentials, namespace RoleBinding, OIDC runtime discovery, and Registry logs.
Push or pull fails after login	Image namespace permissions, repository path, storage backend, and Registry readiness.

Managing Access and Cleanup

Use this page for Alauda Container Platform Registry administrator and namespace administrator tasks, including namespace image access, usage reporting, image pruning, registry garbage collection, and image signature verification.

TOC

Task Index
Prerequisites
Grant Namespace Permissions
View Usage
Verify Image Signatures
Prune Images
Run Registry Garbage Collection

Task Index

Task	Use when
Grant namespace permissions	A user or service account needs pull, push, delete, or image catalog access in an image namespace.

Task	Use when
View usage	An administrator needs image or ImageStream usage information.
Verify image signatures	An administrator needs to check or save trusted image signature conditions.
Prune images	An administrator needs to remove unused image metadata after reviewing a dry run.
Run registry garbage collection	An administrator needs to reclaim unreferenced blobs from Registry storage after metadata cleanup.

Prerequisites

- ACP Registry is installed and available.
- You have `kubectl` and `ac` access to the target cluster.
- You have permission to create RoleBindings in the image namespace.
- You have administrator permissions for usage reporting, image pruning, registry garbage collection, and signature verification.

Select the ACP 4.4 Registry backend before using the `ac` administrative commands:

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

Expected result:

- Subsequent `ac` Image API, pruning, garbage-collection, and signature commands use the ACP 4.4 Registry.

Grant Namespace Permissions

Grant pull permission to a user:

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

Grant push permission to a user:

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

Grant pull permission to a service account in another namespace:

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

Placeholder	Description
`<username>`	ACP or Kubernetes username that needs access to images in the target namespace.
`<image-namespace>`	Namespace that owns the image repository, for example `team-a`.
`<workload-namespace>`	Namespace where the workload service account runs.
`<serviceaccount-name>`	ServiceAccount name used by the workload that pulls the image.

Verify the RoleBinding and effective access:

```
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>
```

Expected results:

- The RoleBinding exists in the image namespace.
- The `kubectl auth can-i` command returns `yes` for the service account that should pull images.

The Registry uses ImageStream layer authorization:

Operation	Typical role	Image API permission
Pull	`system:image-puller`	`image.alauda.io` `imagestreams/layers` `get`
Push	`system:image-pusher`	`image.alauda.io` `imagestreams/layers` `update`
Delete	`system:image-deleter`	Image API delete permissions for the target image metadata
Write signatures	`system:image-signature-writer`	`image.alauda.io` `imagesignatures` `create`, `delete`
Read the image catalog	`system:image-registry-catalog-reader`	`image.alauda.io` `imagestreams` `list`
Read Registry metrics	`system:image-registry-metrics-reader`	`image.alauda.io` `registry/metrics` `get`

The `system:image-pruner` ClusterRole is reserved for the Operator-managed `pruner` ServiceAccount. It is not a namespace access role. Manual pruning and garbage collection require the corresponding administrator permissions.

For roles other than Pull and Push, grant the corresponding platform role or ClusterRole according to your authorization model, then verify the effective permission. The metrics permission is cluster-scoped; the other checks are scoped to the image namespace where applicable:

```
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>
```

Expected result:

- Each command returns `yes` for a principal with the corresponding effective permission. The permission can also be granted by another role.

View Usage

Show storage and usage statistics for Images:

```
ac adm top images
```

Show storage and usage statistics for ImageStreams:

```
ac adm top imagestreams
```

Expected result:

- The command prints usage rows for Image or ImageStream resources visible to the current administrator.

Verify Image Signatures

Verify the image signature identity recorded on an Image object:

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

Placeholder	Description
`sha256:<digest>`	Digest of the Image object to verify. Use `ac get images` or `ac get imagestreamtags <name>:<tag> -n <namespace> -o wide` to find the digest.
`registry.example.com/team-a/demo:latest`	Expected signed image identity. Use the image reference required by your signing policy.

Save trusted conditions back to the Image object:

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

Verify the saved condition:

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

Expected result:

- The Image object contains the trusted signature condition saved by the verification command.

Saving trust conditions changes Image API metadata. To roll back a saved condition, edit the Image object and remove the saved condition, or restore the Image object from a known-good backup.

Remove all signature records from an Image object only when the records are no longer required:

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

> **Warning:** `--remove-all` removes every signature record from the Image object. This operation is not reversible through the Registry. Verify the Image object after the command and restore the metadata from a known-good backup if the records were removed by mistake.

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

Prune Images

Confirmed pruning can permanently remove unused image metadata. If you also run registry garbage collection, unreferenced blobs can be permanently reclaimed from storage. Run these commands during a maintenance window, keep required backups, and review dry-run output before adding `--confirm`.

Preview image pruning:

```
ac adm prune images
```

Expected result:

- The dry run lists prune candidates and does not delete image metadata.

Run pruning after reviewing the dry-run output:

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

Verify pruning:

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

Expected result:

- A follow-up dry run no longer lists the metadata that was pruned by the confirmed command.

Exclude repositories with an allow-list pattern by using `--whitelist`:

```
# 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
```

For scheduled pruning, see Setting up and configuring the registry.

Confirmed pruning is not automatically reversible. Restore pruned metadata from backup, recreate metadata from a trusted image source, or re-import the image when recovery is required.

Run Registry Garbage Collection

Registry garbage collection reclaims unreferenced blobs from Registry storage. Blob reclamation is not reversible from the Registry after it is confirmed.

Preview registry garbage collection after metadata cleanup:

```
ac adm registry gc
```

Expected result:

- The command previews eligible unreferenced blobs and does not delete them.

Run registry garbage collection only after reviewing the preview:

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

Expected result:

- The command removes eligible unreferenced blobs from Registry storage.

You can also trigger registry garbage collection as part of pruning:

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

Both `ac adm prune images` and `ac adm registry gc` are dry-run by default. Review the preview before adding `--confirm`. Confirmed blob reclamation is irreversible from the Registry; restore data from a storage backup or copy the image again from a trusted source if a required blob was reclaimed.

After confirmed garbage collection, verify critical images by pulling or inspecting them:

```
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
```