Registry Administration

Overview

Overview

Registry Data Backup and Recovery

Registry Data Backup and Recovery

Registry v2 Administration

Registry v2: Image Registry Ov Registry v2: Image Registry Op Registry v2

Registry v2: Exposing the Regi Registry v2: Managing Access and Cleanup

Install

- Install by Using YAML
- Install by Using the Web Console

Upgrade

- Upgrade Registry

Overview

ACP Registry documentation has two administration tracks. Use **Registry v2** for the Operator-managed ACP Registry. Use the legacy Registry pages only for environments that still run the Registry Cluster Plugin.

Developers and application teams use the registry through developer image workflows.

TOC

Administration scope

Need	Continue with
Administer the Operator-managed Registry v2.	Registry v2
Install the legacy Registry Cluster Plugin.	Install
Upgrade the legacy Registry Cluster Plugin.	Upgrade
Back up or recover legacy Registry Cluster Plugin data.	Registry backup and recovery

Need	Continue with
Use images in workloads or developer workflows.	Images and registry image use

Namespace isolation and access

Registry image repositories are organized by namespace. Pull and push access depends on the platform roles and permissions granted to users or ServiceAccounts in the target namespace.

For common image usage commands and cross-namespace pull access, see Registry image use.

Alauda Container Platform Registry Data Backup and Recovery

TOC

Overview

Use this procedure to back up and recover ACP Registry data when registry image data is stored in S3-compatible object storage.

The recovery model separates image data stored in S3 from the Cluster Plugin configuration defined in the Kubernetes `ModuleInfo` custom resource.

- **Backup**: Retrieve S3 configuration from the `ModuleInfo` resource and back up data from the specified storage bucket.
- **Recovery**: After installing the Cluster Plugin in a new or repaired cluster, update its S3 configuration in the `ModuleInfo` resource to point to the bucket that contains the restored data.

This model keeps data backup and recovery independent from Cluster Plugin deployment and upgrade operations. All connection information is managed through the declarative `ModuleInfo` resource.

Prerequisites

- Have `kubectl` access and appropriate permissions to operate the target Kubernetes cluster.
- Have credentials and client tools (e.g., awscli, rclone, minio-client) to access and operate the S3-compatible storage used for image data.
- The ACP Registry Cluster Plugin is installed and configured, and its `ModuleInfo` resource exists and is in a healthy state.
- Have independent, sufficient storage capacity prepared for backup data (e.g., another S3 bucket).

Data Backup

During backup, obtain the current production S3 configuration and perform a full backup of the image data within the storage bucket.

Step 1: Obtain Current S3 Configuration

Extract the S3 storage configuration from the `ModuleInfo` resource that manages the ACP Registry Cluster Plugin. This information is the basis for the backup operation.

Run the following commands on the `global` cluster:

```
# 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}'
```

Key Variable Descriptions:

- `S3_BUCKET`: The source bucket name where image data is actually stored.
- `S3_ENDPOINT`: The endpoint URL to connect to the S3-compatible service.
- `S3_REGION`: The region identifier for the S3 service.
- `S3_SECRET_NAME`: The Kubernetes Secret name storing the authentication keys.

Step 2: Perform S3 Bucket Data Backup

Using your S3 client tool of choice, leverage the configuration obtained in the previous step to perform a full backup of the source bucket's data.

Use the following backup logic:

- Configure your client with the endpoint ($S3_ENDPOINT), region ($S3_REGION), and the decoded access key and secret key from the Secret.
- Execute a sync or copy command to back up all data from the source bucket ($S3_BUCKET) to your prepared independent backup location (e.g., another S3 bucket or path).
- Record the backup timestamp, the bucket name and endpoint used, and archive this information with the backup files.

Data Recovery

Before recovery, install the ACP Registry Cluster Plugin in the target environment, such as a new cluster or repaired cluster. Then modify the configuration so the registry can access the restored image data.

Step 1: Prepare Backup Data

Using your S3 client tool of choice, restore the backed-up image data into a **target S3 storage `bucket` that is definitively accessible**. For example, restore it into a new bucket named `registry-bucket-restored`. Ensure you have write permissions to this target bucket.

Step 2: Update ModuleInfo Configuration

The key to recovery is updating the `ModuleInfo` resource of the new `cluster plugin` to point its S3 configuration to the target bucket containing the backup data.

1. **Determine New S3 Connection Information**:
   - `NEW_BUCKET`: The **target bucket name** where the backup data has been restored (e.g., registry-bucket-restored).
   - `NEW_ENDPOINT`: The **endpoint of the target S3 service**. This remains unchanged if the S3 service address is the same as during backup.
   - `NEW_REGION`: The **region of the target S3 service**.

- `NEW_SECRET_NAME`: The name of the Kubernetes Secret with read/write permissions to the **target bucket**. If the access keys are unchanged, this is still `$S3_SECRET_NAME`.

2. **Update ModuleInfo Resource**:
   Use the `kubectl patch` command to directly update the S3 configuration section of the `ModuleInfo`. The platform controller will automatically synchronize this change to all relevant Deployment, Pod, and other resources.

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

Key point: This operation triggers a rolling update of the ACP Registry Pods. Newly started Pods use the new configuration to connect to the specified target storage bucket.

Verification

After the update is complete, follow these steps to verify successful data recovery and normal service operation.

Check Module Status in the `global` Cluster

```
# 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
```

Verify Data Access (API Test)

Use the Registry's API interface to directly verify it can read the restored image data.

```
# 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",...]}
```

Functionality Test

Attempt to pull a known image from the restored registry or push a new image to it to fully verify read/write functionality.

Other Storage Backends

This procedure uses S3 storage as the example. The same recovery model can apply to other storage backends supported by Registry, such as local file system, StorageClass, or NAS, when the corresponding backup and restore tools are available.

Regardless of storage type, first extract storage connection parameters from the corresponding configuration block, such as `s3storage` or `persistence`, in the `ModuleInfo` resource. Then use the appropriate storage tools to back up data. During recovery, restore data to the target location and update the corresponding configuration fields in `ModuleInfo` so the newly deployed registry instance uses that location.

Registry v2 Administration

Use this section to install, configure, expose, and administer the Operator-managed ACP Registry.

Registry v2 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.

TOC

Administration Topics

Need	Continue with
Review Registry v2 architecture and compatibility notes.	Image registry overview
Install the Image Registry Operator or check Operator status.	Image Registry Operator
Configure storage, image limits, pull secrets, and scheduled pruning.	Setting up and configuring the registry
Expose Registry v2 for external clients.	Exposing the registry

Need	Continue with
Manage namespace image access, usage reporting, pruning, garbage collection, and signature verification.	Managing access and cleanup
Use Registry v2 from workloads or developer workstations.	Registry v2 image use

Registry v2: Image Registry Ov

- Integrated Image Registry
- What Changed from the Legacy Registry
- Common Terms
- Automatic Image Pruning
- Compatibility Notes

Registry v2: Image Registry Op

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

Registry v2

- Prerequisites
- Configure Deve
- Configure PVC
- Configure S3-C
- Configure Mana
- Configure Imag
- Configure Sche
- Operate Storag

Registry v2: Exposing the Registry

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

Registry v2: Managing Access and Cleanup

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

Registry v2: Image Registry Overview

Registry v2 is the Operator-managed integrated image registry for ACP clusters. It provides internal image storage, ImageStream metadata, namespace-based access control, managed service account pull credentials, and image pruning.

TOC

Integrated Image Registry

The registry runs as a cluster workload in `image-registry-system`. 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`, such as PVC, S3-compatible storage, `emptyDir`, or another supported backend.
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, list, or prune image content.

What Changed from the Legacy Registry

Area	Legacy Registry	Registry v2
Runtime namespace	Commonly `cpaas-system`	`image-registry-system`
Internal service address	`image-registry.cpaas-system.svc`	`image-registry.image-registry-system.svc:5000`
Lifecycle management	Platform plugin or chart-managed Registry resources	OLM and `cluster-image-registry-operator`
Desired state	Plugin configuration and legacy Registry configuration	`configs.imageregistry.operator.alauda.io/cluster` a `imagepruners.imageregistry.operator.alauda.io/clus`
Image metadata	Registry HTTP view and legacy metadata APIs	Aggregated `image.alauda.io/v1` Image API
External exposure	Legacy ingress or gateway configuration	`Config.spec.defaultRoute` and `Config.spec.routes[]` rendered as Kubernetes Ingress

Area	Legacy Registry	Registry v2
Pull credentials	Legacy service account pull secret automation	Built-in Operator imagePullSecret controller
Image limits	Legacy Registry gateway ConfigMap	Kubernetes `LimitRange` and `ResourceQuota` with `alauda.io` image resources
Pruning and GC	Legacy `ac` Registry commands and scripts	`ImagePruner/cluster`, `image-pruner` CronJob, and `ac adm prune images` / `ac adm registry gc`

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

Registry v2 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.

Compatibility Notes

- The legacy Registry how-to pages remain valid for environments that still use the legacy Registry.
- Registry v2 Image API resources use the API groups `image.alauda.io/v1` and `imageregistry.operator.alauda.io/v1`.
- Registry v2 uses ACP Image API resources such as `Image`, `ImageStream`, `ImageStreamTag`, and `ImagePruner`.
- Use `ac` for ACP Registry workflows.

Registry v2: Image Registry Operator

The Image Registry Operator installs and manages the cluster-wide Registry v2 instance. It reconciles registry runtime resources from `Config/cluster` and scheduled pruning resources from `ImagePruner/cluster`.

TOC

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 prune and garbage-collection workflows.
Managed imagePullSecret controller	Creates, injects, refreshes, and removes service account pull secrets for the internal registry.

Install the Operator

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

Do not use the legacy Registry Cluster Plugin to install Registry v2. The legacy Registry Cluster Plugin remains available only for legacy Registry deployments.

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 **Image Registry Operator** 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	`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-op
erator
```

Expected results:

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

Install by Using YAML

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
```

Change the Registry Management State

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

Expected result:

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

Setting the management state to `Removed` stops Registry v2 runtime components. Push and pull traffic, the Image API server, and scheduled pruning are unavailable 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.

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 v2 internal service URL by default.

Check Operator and Registry Status

```
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:

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

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.

Registry v2: 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 **Registry v2**.

TOC

Prerequisites

- Registry v2 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

`emptyDir` stores image data on ephemeral Pod storage. Use it only for development or test environments where all pushed image data can be discarded when the Registry Pod is recreated. 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
      },
      "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. Create a file named `image-registry-pvc.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,
      "storage": {
        "managementState": "Unmanaged",
        "emptyDir": null,
        "pvc": {
          "claim": "image-registry"
        },
        "s3": 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>
```

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.

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

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,
        "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 reachable from the Registry Pod, without the bucket name.
`<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.

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 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.

Configure Image Limits

In Registry v2, image size and tag-count limits are represented with Kubernetes `LimitRange` and `ResourceQuota` objects. Do not use the legacy Registry gateway limit ConfigMap for Registry v2 deployments.

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`:

```
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`.

Pruning removes unused image metadata. Run registry garbage collection separately when blob storage reclamation is required.

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.

Registry v2: Exposing the Registry

By default, use the internal registry service for workloads inside the cluster. Expose **Registry v2** only when developer machines, CI systems, or other external clients must push or pull images.

TOC

Prerequisites

- Registry v2 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_HOST="$(
  kubectl -n image-registry-system get ingress default-route \
    -o jsonpath='{.spec.rules[0].host}'
)"

echo "$REGISTRY_HOST"
```

Expected results:

- `Config/cluster.spec.defaultRoute` is `true`.
- The `default-route` Ingress exists in `image-registry-system`.
- `REGISTRY_HOST` contains the hostname assigned to the default route.

If the Ingress certificate is signed by a private CA, add the CA certificate to the trust store of the external client and the OCI client before logging in. Verify the TLS chain from the external client:

```
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[]`.

The TLS Secret must exist in `image-registry-system` when `secretName` is set.

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.
`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.

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.

Registry v2: Managing Access and Cleanup

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

TOC

Task Index

Task	Use when
Grant namespace permissions	A user or service account needs pull, push, delete, or prune 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

- Registry v2 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.

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.

Registry v2 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
Prune	`system:image-pruner`	Image API prune and layer inspection permissions

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.

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`:

```
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.

Run registry garbage collection after metadata cleanup:

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

Expected results:

- Without `--confirm`, the command previews garbage-collection candidates.
- With `--confirm`, the command removes eligible unreferenced blobs from 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`.

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

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

If a required blob was reclaimed, restore it from backend storage backup or copy the image again from a trusted source.

Install

Install by Using YAML

TOC

Use Cases

Use YAML installation for:

- **Advanced users** with Kubernetes expertise who prefer a manual approach.
- Deployments that require externally managed storage, such as NAS, S3-compatible object storage, or Ceph.
- Environments needing **fine-grained control** over TLS and ingress.

- **Full YAML customization** for advanced configurations.

Prerequisites

- **Install** the **Alauda Container Platform Registry** Cluster Plugin to a target cluster.
- **Access** to the target Kubernetes cluster with kubectl configured.
- **Cluster admin permissions** to create cluster-scoped resources.
- Obtain a registered **domain**, such as `registry.example.com`. For domain configuration, see Create a Domain.
- Provide valid **NAS storage**, such as NFS.
- Optional: Provide valid **S3-compatible storage**.

Install Alauda Container Platform Registry Using YAML

Procedure

1. **Create a ClusterPluginInstance manifest file** named `registry-plugin.yaml` with the following template:

```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. **Customize the following fields** according to your environment:

```
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. **How to create a secret** for S3 credentials:

```
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
```

   Replace `<S3-CREDENTIALS-SECRET>` with the name of your S3 credentials secret.

4. **Optional: enable registry push limits** for image size and tag count.

This capability is provided by the built-in Registry proxy.

To enable it:

- Set `spec.config.registryLimitConfig.enabled` to `true`.
- Set `spec.config.registryLimitConfig.configMapName` to the name of a ConfigMap that you create manually in the Registry namespace.

Example:

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

Notes:

- The ConfigMap is not created by the Registry plugin.
- For new deployments, the recommended ConfigMap name is `image-registry-limit-config`.
- The runtime still accepts the legacy ConfigMap name `registry-gateway-config` for backward compatibility.
- For detailed ConfigMap examples, rule behavior, and verification steps, see Configure Registry Push Limits.

When this feature is enabled, apply the ConfigMap before proceeding to step 5:

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

5. Apply the Registry plugin manifest to your cluster.

   This command creates or updates the `ClusterPluginInstance` resource named `image-registry`, which installs or updates the Registry cluster plugin.

   If you enabled `registryLimitConfig`, apply the limit ConfigMap described in Configure Registry Push Limits before this step.

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

Configuration Reference

Common Fields

Parameter	Description	Exam
`spec.config.global.oidc.ldapID`	LDAP ID for OIDC authentication	`ldap`
`spec.config.ingress.hosts[0].name`	Custom domain for registry access	`regi`
`spec.config.ingress.hosts[0].tlsCert`	TLS certificate secret reference (namespace/secret-name)	`cpaa` `tls`
`spec.config.ingress.ingressClassName`	Ingress class name for the registry	`clus`
`spec.config.persistence.size`	Storage size for the registry	`10Gi`
`spec.config.persistence.storageClass`	StorageClass name for the registry	`nfs-`
`spec.config.registryLimitConfig.enabled`	Enable registry push limits for image size and tag count	`true`
`spec.config.registryLimitConfig.configMapName`	Name of the manually created ConfigMap containing limit rules	`imag` `confi`
`spec.config.s3storage.bucket`	S3 bucket name for image storage	`prod`

Parameter	Description	Exam
`spec.config.s3storage.region`	Region identifier for S3-compatible storage	`us-w`
`spec.config.s3storage.regionEndpoint`	S3-compatible service endpoint URL	`http`
`spec.config.s3storage.secretName`	Secret containing S3 credentials	`s3-a`
`spec.config.s3storage.env.REGISTRY_STORAGE_S3_SKIPVERIFY`	Set to `true` for self-signed certs	`true`
`spec.config.infra.enabled`	Deploy components to infra nodes or all nodes	`fals`

Verification

1. Check plugin:

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

2. Verify registry pods:

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

Update Or Uninstall Alauda Container Platform Registry

Update

Execute the following command on the global cluster and update the values in the resource according to the parameter descriptions provided above to complete the update:

```
# <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)
```

Uninstall

Execute the following command on the global cluster:

```
# <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
```

Install by Using the Web Console

TOC

Use Cases

Use the web console installation path for:

- **First-time users** who prefer a guided, visual interface.
- **Quick proof-of-concept setups** in non-production environments.
- Teams with **limited Kubernetes expertise** seeking a simplified deployment process.
- Scenarios requiring **minimal customization** (e.g., default storage configurations).
- **Basic networking setups** without specific ingress rules.
- **StorageClass** configurations for high availability.

Use the YAML installation path when you need advanced S3-compatible storage configuration or specific ingress rules.

Prerequisites

- **Install** the **Alauda Container Platform Registry** Cluster Plugin to a target cluster using the Cluster Plugin mechanism.

Install Alauda Container Platform Registry Using the Web Console

Procedure

1. Log in and navigate to the **Administrator** page.
2. Click **Marketplace** > **Cluster Plugins** to access the **Cluster Plugins** list page.
3. Locate the **Alauda Container Platform Registry** Cluster Plugin, click **Install**, then proceed to the installation page.
4. Configure parameters according to the following specifications and click **Install** to complete the deployment.

The parameter descriptions are as follows:

Parameter	Description
Expose Service	Once enabled, administrators can manage the image repository externally using the access address. This poses significant security risks and should be enabled with extreme caution.
Enable IPv6	Enable this option when the cluster uses IPv6 single-stack networking.
NodePort	When Expose Service is enabled, configure NodePort to allow external access to the Registry via this port.
Storage Type	Select a storage type. Supported types: LocalVolume and StorageClass.

Parameter	Description
Nodes	Select a node to run the Registry service for image storage and distribution. (Available only when Storage Type is LocalVolume)
StorageClass	Select a StorageClass. When replicas exceed 1, select storage with RWX (ReadWriteMany) capability (e.g., File Storage) to ensure high availability. (Available only when Storage Type is StorageClass)
Storage Size	Storage capacity allocated to the Registry (Unit: Gi).
Replicas	Configure the number of replicas for the Registry Pod: • **LocalVolume**: Default is 1 (fixed) • **StorageClass**: Default is 3 (adjustable)
Resource Requirements	Define CPU and Memory resource requests and limits for the Registry Pod.

Verification

1. Navigate to **Marketplace** > **Cluster Plugins** and confirm the plugin status shows **Installed**.
2. Click the plugin name to view its details.
3. Copy the **Registry Address** and use an OCI client (e.g., `nerdctl`) to push or pull images.

Update Or Uninstall Alauda Container Platform Registry

You can update or uninstall the **Alauda Container Platform Registry** plugin from either the list page or details page.

Upgrade

Upgrade Registry

Upgrade Alauda Container Platform Registry

TOC

Prerequisites

1. Administrator privileges for ACP.
2. The old plugin refers to versions of ACP v4.1 and earlier.
3. The new plugin refers to versions of ACP v4.2 and later.

Overview

Upgrade the old registry plugin to the target registry plugin according to the registry storage type. Because the Cluster Plugin name changed, manual intervention is required.

In the commands below, replace `<LEGACY-PLUGIN-NAME>` with the registry plugin name used in ACP v4.1 and earlier.

You can identify it by checking the existing registry plugin resources (Execute in Global cluster):

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

Upgrade Steps

Environment Check and Preprocessing

Old Plugin Not Installed

If the old plugin was never installed, you just need to clean up the old plugin from [Marketplace/Cluster Plugins]:

```
# 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
```

Old Plugin Installed

If the old plugin is installed in any cluster, follow the migration procedure based on your **storage type**.

S3 Storage Migration

Characteristics: Automatic data migration, no manual operation needed

Procedure:

1. Backup Old Plugin Configuration: (Execute in workload cluster where the old plugin is installed)

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

2. Uninstall Old Plugin: (Execute in Global cluster)

```
# 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. Install New Plugin (Execute in workload cluster where the plugin will be installed)

- Edit backup file `backup-clusterplugininstances.yaml`, replace all occurrences of `<LEGACY-PLUGIN-NAME>` with `image-registry`
- Apply new configuration:

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

4. Cleanup Old Plugin From [Marketplace/Cluster Plugins]: (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
```

NFS Storage Migration

Characteristics: Manual PV retention and reclaim policy modification required.

Procedure:

1. Record PV Information: (Execute in workload cluster where the old plugin is installed)

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

2. Modify PV Reclaim Policy: (Execute in workload cluster where the old plugin is installed)

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

3. Backup Old Plugin Configuration (Same as S3 Step 1)
4. Uninstall Old Plugin (Same as S3 Step 2)
5. Release PV Binding: (Execute in workload cluster where the old plugin is installed)

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

6. Install New Plugin (Important: You must set the `config.persistence.volumeName` for the new plugin to retain the old data)

- Edit configuration file `backup-clusterplugininstances.yaml`, set config.persistence.volumeName: `<OLD_PV>`
- Apply configuration (Same as S3 Step 3)

7. Cleanup Old Plugin From [Marketplace/Cluster Plugins] (Same as S3 Step 4)

Local Storage Migration

Characteristics: Manual data copy required

Procedure:

1. Backup Old Plugin Configuration (Same as S3 Step 1)
2. Uninstall Old Plugin (Same as S3 Step 2)
3. Install New Plugin (Same as S3 Step 3)
4. Data Migration:

- ssh login to old plugin Pod node, copy data from old directory to new directory with all file permissions:

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

5. Cleanup Old Plugin From [Marketplace/Cluster Plugins] (Same as S3 Step 4)