Upgrade

This document will provide all the information regarding the upgrading of ACP.

Upgrade Path Selection

Upgrade the `global` cluster first, then workload clusters. For each cluster, **Upgrade the Distribution Version** applies to both node operating systems. If the cluster runs Alauda OS, continue with **Upgrade Kubernetes on Alauda OS**:

- `global` cluster: Upgrade Kubernetes on Alauda OS
- Workload clusters: Upgrade Kubernetes on Alauda OS

For the upgrade model, see Overview.

Overview

Overview

Pre-Upgrade Preparation

Pre-Upgrade Preparation

Upgrade the global Cluster

Upgrade the Distribution Versic Upgrade Kubernetes on Alauda Upgrade in

Upgrade Workload Clusters

Upgrade the Distribution Versic Upgrade Kubernetes on Alauda OS

Upgrade Validation

Upgrade Validation

Upgrade Troubleshooting

Upgrade Troubleshooting

Overview

Starting with ACP 4.3, cluster upgrades use the Cluster Version Operator (CVO)-based workflow.

The upgrade request can be submitted from the Web Console, from ACP CLI, or by editing `ClusterVersionShadow` directly. The Web Console offers two cluster management interfaces: the built-in one that ships with the platform, and the newer one delivered by the **Alauda Container Platform Fleet Essentials** cluster plugin. Both are documented side by side on the global-cluster and workload-cluster pages.

When moving the platform to a new ACP Distribution Version, upgrade the `global` cluster first and workload clusters second:

1. Upgrade the global tier to the target Distribution Version by following the validated global-cluster procedure, including artifact preparation and preflight checks.
2. After the global tier reaches the target Distribution Version, upgrade workload clusters from the supported workload-cluster entry point and observe cluster status until each target cluster reaches the same Distribution Version.

A workload cluster can be upgraded only to a Distribution Version that the global tier has already reached. In environments with **global disaster recovery (DR)**, this means both the standby and primary global clusters must reach the target Distribution Version before workload clusters are upgraded to that Distribution Version. Before upgrading the global tier, verify that every workload cluster is within the target Distribution Version's **Compatible Versions** in the Kubernetes Support Matrix. This prerequisite is separate from the sequencing rule.

TOC

Upgrade Steps by Node Operating System
Distribution Version vs Kubernetes Version
Upgrade Workflow
Key Concepts
Upgrade Scope and Sequencing
CVO Management Scope
Upgrade Entry Points
Related Documentation

Upgrade Steps by Node Operating System

Each cluster is upgraded in one or two steps, depending on its node operating system:

1. **Upgrade the Distribution Version** applies to both node operating systems. CVO moves ACP Core and the installed Aligned plugins to the target Distribution Version. On a non-Alauda OS cluster, this step also upgrades Kubernetes in place on the existing nodes.
2. **Upgrade Kubernetes on Alauda OS** applies only to Alauda OS clusters. After the Distribution Version upgrade, Kubernetes and the Alauda OS image are rolled out by replacing nodes through Cluster API rolling updates. Nodes are not upgraded in place.

Distribution Version vs Kubernetes Version

Component	Description	Managed by
Distribution Version	The ACP Core release installed on the cluster.	CVO, through **Upgrade the Distribution Version**.
Kubernetes and OS baseline	On Alauda OS, a validated Kubernetes minor, component set, and Alauda OS image for the installed ACP release.	Cluster API and the provider-specific procedure in **Upgrade Kubernetes on Alauda OS**.

On a non-Alauda OS cluster, CVO upgrades Kubernetes as part of the Distribution Version upgrade. On an Alauda OS cluster, an ACP minor release line can offer more than one validated Kubernetes target over its patch lifecycle. Use OS Support Matrix for the exact patch-specific Kubernetes and component values offered for the Kubernetes upgrade.

Upgrade Workflow

Follow the pages in this order:

1. Pre-Upgrade Preparation: confirm the supported path, cluster readiness, and complete target-version package set.
2. Upgrade the `global` cluster: Upgrade the Distribution Version, or Upgrade in a DR Environment when global DR is enabled. If the `global` cluster runs Alauda OS, continue with Upgrade Kubernetes on Alauda OS. In a DR environment, run it on both `global` clusters.
3. Upgrade workload clusters after the global tier reaches the target Distribution Version: Upgrade the Distribution Version. For workload clusters that run Alauda OS, continue with Upgrade Kubernetes on Alauda OS.
4. Upgrade Validation: accept the Distribution Version, Kubernetes, Core, Aligned plugins, Pods, artifacts, CVO history, and Web UI.
5. Upgrade Troubleshooting: use this only when preflight or execution reports a block or stall.

Plan the maintenance timeline so preparation finishes before the change window:

Phase	Recommended timing	State change
Download and publish artifacts	Before the maintenance window	No running component is upgraded.
Run preflight and resolve blocks	1–2 weeks before the window	Read-only checks; remediation can change the environment separately.
Upgrade the global tier	First maintenance window	Core, Kubernetes, and installed Aligned plugins move to the target.

Phase	Recommended timing	State change
Upgrade workload clusters	After the global tier	Each selected workload cluster moves in its own controlled window.
Validate and complete follow-up work	Before closing each window	Confirms acceptance and completes required post-upgrade actions.

Key Concepts

- **ClusterVersionShadow (`cvsh`)**: The upgrade resource used to track current version, desired version, preflight results, execution stages, and history.
- **Distribution Version**: The ACP version currently reached by a cluster. A workload cluster can be upgraded only to a Distribution Version that the global tier has already reached.
- **Preflight**: Validation checks that run before the upgrade starts applying the target version. For workload clusters, review preflight results from the upgrade status output after the upgrade request is submitted.
- **Available upgrade targets**: The upgrade versions that are currently offered for a cluster. In the Web Console, the target version for the current upgrade flow is determined by the platform.
- **upgrade.sh**: The preparation script for global cluster upgrades. With the platform built-in Registry, it synchronizes Core artifacts and packages staged in `plugins/`. With an external Registry, synchronization is skipped and the target payload is uploaded separately. The script also runs preflight and deploys CVO at the appropriate stages.
- **Global DR environment**: An environment with both a primary global cluster and a standby global cluster.
- **Primary global cluster**: The global cluster that the platform access domain currently resolves to.
- **Standby global cluster**: The other global cluster in the DR pair. After a failover, the roles of the two clusters are swapped.

Upgrade Scope and Sequencing

A complete ACP upgrade is staged. Each cluster is upgraded in a single maintenance window, but the platform is moved cluster by cluster, with global first and workload clusters second.

A single cluster upgrade window covers four kinds of work:

Work item	Required?	How it is upgraded	If you run Alauda OS
ACP Core (the platform itself)	Required	CVO, driven by artifacts prepared through Sync upgrade artifacts	Same control flow; the Kubernetes step is handled separately, see below
Aligned plugins (cluster plugins and operators released together with the matching ACP version)	Required for each installed plugin	For the Aligned applications provided through **ACP Upgrade to v4.5**, manually copy their packages into `plugins/` and publish them through Sync upgrade artifacts. Publish other installed Aligned packages with `violet`. CVO then upgrades each installed Aligned plugin as part of the cluster upgrade.	Same
Kubernetes	Required in the same window	On non-Alauda OS clusters, CVO upgrades Kubernetes in place on the existing nodes as part of the same cluster upgrade; nodes are not replaced.	After the Distribution Version upgrade, Kubernetes is rolled out by replacing nodes from a new Alauda OS-based image through Cluster API rolling updates, not upgraded in place. See Upgrade Kubernetes on Alauda OS for the `global` cluster, and Huawei DCS, VMware vSphere, Huawei Cloud

Work item	Required?	How it is upgraded	If you run Alauda OS
			Stack, or Bare Metal for workload clusters
Agnostic plugins (independently released cluster plugins and operators)	Optional, per plugin	Upgrade each cluster plugin or operator from **Marketplace > Cluster Plugins** or the operator's own workflow after the cluster reaches the target Distribution Version	Same

A few rules govern when each piece moves:

- The platform requires that a cluster's Kubernetes version is upgraded together with its Core and Aligned versions; mixing a new Core release with an old Kubernetes minor version is not validated.
- Before requesting an upgrade, make the target-version package available for every Aligned plugin already installed on that cluster. Use the `plugins/` publication path in Sync upgrade artifacts for packages from **ACP Upgrade to v4.5**, and use `violet` for other Aligned packages. A missing package for an installed Aligned plugin stalls the CVO upgrade. Packages for Aligned plugins that are not installed on the cluster do not cause those plugins to be installed.
- Cluster plugins are global resources. A package made available through the global-tier artifact publication or `violet` workflow can be used by every cluster in the platform; you do not publish the same cluster plugin again per workload cluster.
- Operators are scoped to each cluster. Pushing operators to the global tier is recommended but not sufficient on its own; the same operator must be available on each workload cluster before the workload upgrade. If you forgot to push an operator during the global window, you can push it again before the workload upgrade as a fallback.
- Workload-cluster upgrades are only required when a workload cluster falls outside the target ACP release's Kubernetes Support Matrix. Workload clusters that stay within the supported range can keep their existing Kubernetes version after the global tier moves; the platform will continue to manage them.

CVO Management Scope

The Cluster Version Operator (CVO) decides which kinds of upgrades it drives end to end:

Lifecycle type	Driven by CVO?	Notes
Core	Yes, CVO is the only supported path	Core upgrades require the artifacts prepared through the global-cluster sync workflow; CVO consumes those artifacts.
Aligned plugins	Yes by default	The cluster plugin or operator workflow can also upgrade an individual Aligned plugin out of band — for example, to apply a critical security fix without moving the cluster to a new Distribution Version.
Agnostic plugins	No	CVO does not manage these. Upgrade each cluster plugin from **Marketplace > Cluster Plugins** and each operator from its own workflow after the cluster reaches the target Distribution Version.

Customers running real production maintenance windows usually move Core, Aligned, and the in-use Agnostic plugins together. The upgrade pages below document that path: prepare every needed artifact during pre-upgrade, drive Core and Aligned through CVO, and finish the in-use Agnostic plugins from Marketplace.

INFO

Kube-OVN on non-Alauda OS and Alauda OS clusters

On a **non-Alauda OS** cluster, Kube-OVN is part of ACP Core and is upgraded automatically by CVO together with the rest of Core — operators do not patch any per-cluster Kube-OVN version annotation and do not touch the Kube-OVN `AppRelease` by hand.

On **Alauda OS** clusters (Huawei DCS, VMware vSphere, Huawei Cloud Stack, and Bare Metal), Kube-OVN is driven separately by the infrastructure provider through the `cpaas.io/kube-ovn-version` annotation on the cluster's `Cluster` resource — see Upgrade Kube-OVN Before the Control Plane. That path does not apply to non-Alauda OS clusters.

Upgrade Entry Points

- **Global clusters**: Follow the validated `upgrade.sh`-based procedure. After the preparation phase completes, request the upgrade from the Web Console, use ACP CLI, or update `ClusterVersionShadow.spec.desiredUpdate` directly.
- **Workload clusters**: Use the Web Console or ACP CLI after the target Distribution Version becomes available for the workload cluster.

Related Documentation

- Pre-upgrade
- Upgrade the Distribution Version of the `global` cluster
- Upgrade Kubernetes on Alauda OS for the `global` cluster
- Upgrade global clusters in a DR environment
- Upgrade the Distribution Version of workload clusters
- Upgrade Kubernetes on Alauda OS for workload clusters
- Change the machine specification of Alauda OS nodes
- Upgrade validation
- Upgrade troubleshooting
- Global Cluster Disaster Recovery

Pre-Upgrade Preparation

MySQL-PXC Removed in Alauda Data Services Relational Database E1 v4.3.0

If you have **MySQL-PXC** instances, they will become **unmanaged** after upgrading to ACP v4.3.0 or later (when the MySQL operator is also upgraded). See MySQL Operator Upgrade Guide ↗ for migration instructions.

Supported upgrade paths:

At the minor-release policy level, ACP 4.2, 4.3, and 4.4 are within the n-3 direct-upgrade window for the ACP 4.5 release line. This policy does not make every source patch eligible for every target patch.

Upgrade only to a target 4.5.x version that the platform offers for the current source patch. After the target `ProductManifest` is registered, confirm that the target appears in `availableUpdates` in the Web Console or `ac adm upgrade`, and confirm that the `VersionUpgradePath` preflight check passes. If no 4.5.x target is offered, use a later 4.5.x target that includes the current source patch or first follow an intermediate target offered by the platform. Do not use an explicit upgrade request or disable `VersionUpgradePath` to bypass an unsupported source-to-target combination.

ACP 4.1.x and earlier releases are outside the direct-upgrade window for ACP 4.5. Follow the intermediate upgrade chain offered by the platform, and verify `availableUpdates` and `VersionUpgradePath` at every hop.

INFO

For clusters that use Alauda OS, the ACP Distribution Version must follow the same CVO-offered source-to-target path. When the target ACP version is more than one Kubernetes minor above the

cluster's current version, Upgrade Kubernetes on Alauda OS requires additional pre-staging of intermediate-version artifacts. See Preparing for Cross-Version Upgrades before starting the upgrade window.

TOC

Important Notes

Verify Module Stability Before You Upgrade

Kubernetes 1.35 or Later Node Readiness

Download the Packages for Offline Environments

- Step 1: Download the Core Package
- Step 2: Download ACP Upgrade to v4.5
- Step 3: Inventory other Extensions on every cluster in scope
- Step 4: Download the other Extension packages

Important Notes

- Ensure the directory `/cpaas/minio` on the control plane nodes of global cluster has at least **120 GB** of free disk space.
- Before upgrading the global tier, verify that all workload clusters remain within the target Distribution Version's **Compatible Versions** documented in Kubernetes Support Matrix.
- If any workload cluster is outside the target compatible range, upgrade that workload cluster first until it enters the range before upgrading the global tier.
- A workload cluster can be upgraded only to a Distribution Version that the global tier has already reached.

Verify Module Stability Before You Upgrade

The cluster upgrade preflight requires every installed **Core** and **Aligned** plugin module on the target cluster to be in a stable state. If any such module is mid-install, mid-upgrade, or failed, the `ModuleInfoStable` and `ClusterModuleStable` preflight checks block the upgrade until it is resolved. A common cause is a standalone plugin upgrade that is still running when the cluster upgrade is requested.

Before you request the upgrade, list the installed modules on the target cluster and confirm every one is in the `Running` phase:

```
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster> \
  -o custom-columns='MODULE:.metadata.labels.cpaas\.io/module-name,CURREN
T:.status.version,TARGET:.spec.version,PHASE:.status.phase'
```

Also confirm the cluster module itself is not in a deploy or upgrade failure state:

```
kubectl get clustermodule <cluster> -o jsonpath='{.status.base.deployStat
us}{"\n"}'
```

Every module `PHASE` must be `Running`, and `deployStatus` must not be `Upgrading`, `DeployFailed`, or `UpgradeFailed`. If a module is still upgrading, wait for it to reach `Running`; if a module is `Failed` or `Blocked`, resolve it before you request the cluster upgrade.

Kubernetes 1.35 or Later Node Readiness

If the target release is ACP 4.4 or later, upgrades the cluster to Kubernetes 1.35 or later, and reports an administrator acknowledgement gate for node readiness, verify every production node that will run the target cluster before acknowledging the gate.

Log in to each node through SSH or another administrative node access method available in your environment, and run the following checks directly on the node operating system. The account and credentials depend on how the cluster nodes were provisioned.

Verify the Linux kernel version:

```
uname -r
```

The kernel must be `5.8` or later and must also be within the ACP supported operating system and kernel matrix for the target release.

Verify that the node uses cgroup v2:

```
stat -fc %T /sys/fs/cgroup/
```

The expected output is:

```
cgroup2fs
```

On Alauda OS clusters, Kubernetes is upgraded by replacing nodes. On the hop to Kubernetes 1.35 or later, also apply the required kubelet patch to the control-plane and worker bootstrap configurations.

Download the Packages for Offline Environments

A production maintenance window typically upgrades **ACP Core**, the **Aligned plugins** (cluster plugins and operators released together with the same ACP version), and the in-use **Agnostic plugins** (independently released cluster plugins and operators) in the same window. Core and Extension packages are separate downloads. Prepare all three package sets before the maintenance window so that artifact synchronization can finish early.

Step 1: Download the Core Package

From the **Alauda Customer Portal**, download the **ACP 4.5 Core Package** that matches your target architecture. The Core Package contains Core artifacts only; it does not contain any Aligned or Agnostic Extension package.

Step 2: Download ACP Upgrade to v4.5

Use **Violet v4.2.13 or later** to download the scenario packages directly. Run `violet version` to verify the version, and use `violet ac login` to log in to the Alauda Customer Portal. For installation and login instructions, see Download Packages.

Set `<target-platform-version>` to the exact ACP Distribution Version of the Core Package, such as `v4.5.0`. Set `<arch>` to `amd64`, `arm64`, or `hybrid` to match the Core Package. List the scenarios first, and then download **ACP Upgrade to v4.5**:

```
violet ac scenarios --arch=<arch> --platformVersion=<target-platform-vers
ion>
violet ac download-app --arch=<arch> --platformVersion=<target-platform-v
ersion> --scenario="ACP Upgrade to v4.5"
```

Confirm that **ACP Upgrade to v4.5** appears in the scenario list before downloading it.

As an alternative to Violet, in the Customer Portal, go to **Marketplace > Batch Download > Install**. On the page, select the target Core Package version from **Your ACP Version**, select the matching **Architecture**, select **ACP Upgrade to v4.5** from **Scenarios**, and download the packages.

This package set provides the target-version packages for the following Aligned applications. These packages remain separate from the Core Package.

Downloaded files are named `<package-name>.<arch>.<version>.tgz`, which does not repeat the display name. Each entry below gives the package name so that you can confirm you received all eleven:

Application	Package name
Alauda Container Platform Web Console	`console`
Alauda Container Platform Web Console Central	`console-central`
Alauda Container Platform Marketplace Web Console	`console-marketplace`
Alauda Container Platform Application Catalog	`application-catalog`
Alauda Container Platform Observability Essentials	`aiops-base`

Application	Package name
Alauda Container Platform Essentials	`container-platform`
Alauda Container Platform Cluster Essentials	`acp-business`
Alauda Container Platform GatewayAPI Plugin	`gatewayapi-plugin`
Alauda Container Platform Basic Monitoring	`prometheus`
Alauda Language Pack for Chinese	`i18n-zh`
Alauda Container Platform Networking E1	`calico`

Before publishing upgrade artifacts, manually copy all eleven packages into the `plugins/` directory of the extracted Core Package. Sync upgrade artifacts describes the built-in and external Registry publication branches.

Step 3: Inventory other Extensions on every cluster in scope

Cluster plugins and operators are managed by different mechanisms. Both are needed in this step:

- **Cluster plugins** are global resources. A cluster plugin pushed to the global tier becomes installable and upgradable on every cluster in the platform; you only push each cluster plugin once.
- **Operators** are scoped to each cluster. Pushing an operator to the global tier is the recommended starting point, but the same operator package must be available on each workload cluster before that cluster's upgrade.

Use the same `violet` installation from Step 2 to inventory installed Extensions and publish packages that are not part of **ACP Upgrade to v4.5**. For more information, see Download Packages and Upload Packages.

On any machine with network access to the **Alauda** platform endpoint, run `violet list` to list the plugins (both Aligned and Agnostic) installed in the current environment and export the result to `./apps.yaml`:

```
violet list \
  --platform-address "https://<your-platform-domain>" \
  --platform-token "<platform_token>" \
  --output-file "./apps.yaml"
```

Prefer `--platform-token` over `--platform-password` to avoid exposing passwords in shell history and process listings (`ps aux`).

Violet v4.2.13 or later is required for this token-based inventory workflow. Earlier versions can fail with `failed to get user info from platform server`; upgrade Violet before continuing.

Step 4: Download the other Extension packages

Import the exported `apps.yaml` file into the **Alauda Customer Portal** to align the package list with what your clusters currently run. Download the matching target-version packages for the other Aligned and Agnostic Extensions that your clusters use.

If the result includes one of the eleven applications from **ACP Upgrade to v4.5**, do not publish that duplicate with `violet`. Use the package from **ACP Upgrade to v4.5** through the Core Package's `plugins/` directory instead. Use `violet push` only for the remaining Extension packages.

Download every package you will need so they are all available locally before the upgrade window starts. The next page, Upgrade the global cluster, describes how the selected Registry path publishes Core and the eleven manually staged packages, and how `violet` publishes the other Extensions. Artifact publication does not upgrade any already-running plugin, so complete it before the maintenance window.

Required Aligned Plugin Packages

Every Aligned plugin installed on a cluster being upgraded must have its target-version package available before the upgrade request. CVO does not require target packages for Aligned applications that are not installed, and synchronizing a package does not install an application that is not currently installed.

If a required package is unavailable, CVO blocks progress and keeps retrying the existing request. For an installed Aligned cluster plugin, the `ClusterVersionShadow` condition reports a missing or

non-Ready target `ModulePluginConfig`. For an installed Aligned operator, inspect its target catalog, `Subscription`, `InstallPlan`, and `ModuleInfo`. Add packages from **ACP Upgrade to v4.5** through the `plugins/` publication path in Sync upgrade artifacts; publish other packages with `violet`.

WARNING

Starting with v4.2, we introduced a new plugin named **Alauda Container Platform Log Essentials**. If you previously installed the log storage plugin, you also have to upload that plugin before starting the upgrade.

Upgrade the global Cluster

Upgrade the Distribution Versic Upgrade Kubernetes on Alauda Upgrade in

Upgrade the Distribution Version of the `global` Cluster

Upgrade Path

This page applies to both node operating systems. On a non-Alauda OS `global` cluster, this procedure also upgrades Kubernetes in place, and the `global` cluster upgrade is complete when it finishes. If your `global` cluster uses Alauda OS, complete this page first, then continue with Upgrade Kubernetes on Alauda OS to roll out Kubernetes by replacing nodes.

ACP consists of a **global** cluster and one or more workload clusters. To move the platform to a new ACP Distribution Version, upgrade the global tier to the target Distribution Version first, and then upgrade workload clusters to that same Distribution Version.

Cluster upgrades use the CVO-based workflow. A typical `global` cluster upgrade includes artifact preparation, preflight checks, upgrade request, and status observation.

Before upgrading the `global` cluster, verify that every workload cluster is within the target release's **Compatible Versions** in the Kubernetes Support Matrix. This prerequisite is separate from the third-party cluster onboarding range.

This Compatible Versions prerequisite applies whether or not the environment uses global DR. Global DR changes the procedure used to upgrade the global tier, but it does not change the requirement that workload clusters must remain within the compatible Kubernetes version range before the global tier is upgraded to the target Distribution Version.

Global cluster upgrades follow the validated `upgrade.sh`-based procedure documented on this page. You can request the global-cluster upgrade from the Web Console, by updating `ClusterVersionShadow.spec.desiredUpdate`, or by using ACP CLI with `--`

`cluster=global`. For the complete AC CLI workflow and output interpretation, see Upgrading Clusters. For full command and flag syntax, see AC CLI Administrator Command Reference.

If the environment uses **global DR**, follow Upgrade Global Clusters in a DR Environment. Otherwise, follow the standard workflow below.

TOC

- Standard Workflow
  - Sync upgrade artifacts
  - Run preflight checks
  - Handle preflight blocks
  - Deploy the cluster version operator
  - Request the upgrade
  - Observe execution
  - Upgrade Agnostic plugins from Marketplace
- Validate the Upgrade
- Related Documentation

Standard Workflow

A global cluster upgrade is staged across a timeline. Most of the work is completed **before** the maintenance window so the window itself stays short and predictable:

Phase	When	What happens	Cluster impact
1. Sync artifacts	Any time before the window	For the platform built-in Registry, `upgrade.sh --only-sync-image` uploads Core artifacts and the eleven packages that you manually place in	None — images and catalog entries are published, but no running plugin is

Phase	When	What happens	Cluster impact
		`plugins/`. For an external registry, manually upload the target Core payload before running `upgrade.sh`. In both cases, `violet` publishes the other Aligned and Agnostic packages needed for the window.	changed; no downtime.
2. Preflight	1–2 weeks before the window	`upgrade.sh --preflight` validates upgrade readiness. Resolve every blocking item before the window opens.	None — read-only validation.
3. Upgrade	During the maintenance window	`upgrade.sh --skip-sync-image` deploys or updates the cluster version operator (CVO); the upgrade is then requested and observed.	The cluster is upgraded.

Phase 1 and Phase 2 do not change cluster state — run them early so the maintenance window only contains Phase 3. A small or lab environment can instead run a single `bash upgrade.sh` that performs synchronization and CVO deployment together, but production windows usually keep them separate.

1 Sync upgrade artifacts

When: any time before the maintenance window. This step uploads artifacts to the registry and does not change cluster state.

First, record the configured registry address and determine whether the platform uses the built-in or an external registry:

```
kubectl get productbase base \
  -o jsonpath='{.spec.registry.address}{"\t"}{.spec.registry.external}{"\n"}'
```

The second value is `false` for the platform built-in Registry and `true` for an external registry. The publication command is selected after the target Aligned packages are copied into `plugins/`.

The Core Package does not contain the Aligned packages from **ACP Upgrade to v4.5**. Copy each of the eleven separately downloaded packages listed in Pre-Upgrade Preparation into the `plugins/` directory of the extracted Core Package. Run the following command once for each package:

```
cp <path-to-upgrade-extension-package> ./plugins/
```

Confirm that the directory contains all eleven packages before synchronization:

```
ls -1 ./plugins/
```

Publish the staged payload by following the branch that matches the value of `ProductBase.spec.registry.external`.

For the platform built-in Registry (`false`), run `upgrade.sh` in sync-only mode from the extracted Core Package directory:

```
bash upgrade.sh --only-sync-image
```

`--only-sync-image` uploads Core images and the packages that you manually placed in `plugins/`, without deploying the cluster version operator or changing running applications. The CVO is deployed later, inside the maintenance window — see Deploy the cluster version operator.

For an external registry (`true`), do not use `--only-sync-image` as proof of publication. `upgrade.sh` automatically skips image synchronization. From the target Core Package's `installer/` directory, run both `res/upload.sh all` and `res/upload.sh necessary` by following Prepare the Target-Version Payload. The modes may run in either order, but both must succeed.

The selected publication path makes the artifacts required by the CVO-based workflow available, including:

Type	Content	Purpose
Product images	`product-image`	Used to resolve the target version and image in `ProductManifest` and CVO.
CVO image	`cluster-version-operator`	Used to deploy or update the cluster version operator.
Plugin artifacts	`plugins/*.tgz`	Used by the upgrade plan when plugin artifacts are required.

For the eleven Aligned applications from **ACP Upgrade to v4.5**, always stage the packages in `plugins/`. The built-in Registry path publishes them with `upgrade.sh --only-sync-image`; the external Registry path publishes them through `res/upload.sh all`. CVO upgrades an application only when it is already installed; publishing a package for an application that is not installed does not install it.

Use `violet push` for the other packages downloaded from the `violet list` inventory:

Package group	Publication path	Requirement
Other Aligned cluster plugins	`violet push` to the global tier	Required for each installed plugin that is not part of **ACP Upgrade to v4.5**.
Other Aligned operators	`violet push` to the global tier and each workload cluster where the operator is installed	Required before upgrading each cluster that runs the operator.
Agnostic cluster plugins and operators	`violet push` to the clusters where you plan to upgrade them	Optional for CVO; required before starting their separate Marketplace or operator upgrade.

```
# Publish another cluster plugin to the global tier
violet push <path-to-other-cluster-plugin-package> \
  --platform-address "https://<your-platform-domain>" \
  --platform-token "<platform_token>"

# Publish another operator to global and each workload cluster that r
uns it
violet push <path-to-other-operator-package> \
  --platform-address "https://<your-platform-domain>" \
  --platform-token "<platform_token>" \
  --clusters "global,<workload-cluster-1>,<workload-cluster-2>"
```

Pushing every operator package to every ACP cluster from the global window is the recommended pattern. If the global window has already passed and you discover a workload cluster missing an operator package, you can still push to that workload cluster before the workload upgrade — see Upgrade workload clusters.

WARNING

Before requesting the upgrade, make the target package available for every Aligned plugin installed on the cluster. Use `upgrade.sh` for applications from **ACP Upgrade to v4.5**, and `violet` for the other Aligned packages. Synchronizing a package does not install an application that is not currently installed. For an installed Aligned cluster plugin, CVO requires the matching target `ModulePluginConfig` to be Ready; otherwise, CVO cannot build the upgrade plan, reports the error through `ClusterVersionShadow`, and retries the existing request. An installed Aligned operator requires a matching target `InstallPlan` from its catalog.

Registry behavior depends on how the environment is configured:

Scenario	Behavior
`--registry` is specified	Use the provided registry directly.
`--registry` is not specified	Read the registry address from `ProductBase.spec.registry.address`.

Scenario	Behavior
Built-in platform registry	Rebuild the access address by using the global VIP.
External registry	Automatically set `SKIP_SYNC_IMAGE=true`; the target payload must already have been uploaded through the canonical target-version payload procedure.
Image upload required but credentials omitted	Read `username` and `password` from the `cpaas-system/registry-admin` Secret.

When the target registry is not the platform default, add registry parameters:

Parameter	Purpose
`--registry`	Specify the target registry address.
`--username` / `--password`	Specify registry credentials.

Artifact validation compares each package that is present in `plugins/` against the version the target Core Package expects, and fails on a version mismatch. It does not verify that the set is complete: a package that was never staged is simply not examined, and the run continues. Confirm the eleven packages against Download ACP Upgrade to v4.5 before relying on a successful validation.

> **WARNING**
>
> Do not open the maintenance window until image and plugin synchronization is complete.

Before running preflight, confirm that the registry configuration is still correct:

```
kubectl get productbase base \
  -o jsonpath='{.spec.registry.address}{"\t"}{.spec.registry.externa
l}{"\n"}'
```

Use successful completion of the selected publication commands and the registry administration interface or API as the Phase 1 evidence. Confirm that the expected target repositories and tags exist, and compare the published Extension set with the installed inventory exported to `apps.yaml`. Do not use `ProductBase.status.artifacts` as a zero-output target-version gate: it represents the current catalog and can legitimately contain `Absent` entries for optional or uninstalled packages.

2 Run preflight checks

When: 1–2 weeks before the maintenance window, so there is time to resolve any blocking items before the window opens.

Run `upgrade.sh` in preflight mode:

```
bash upgrade.sh --preflight
```

Preflight is read-only — it validates upgrade readiness and does not change cluster state.

> **Preflight requires the CVO custom resources to exist**
>
> On a cluster that has never had the cluster version operator deployed, `--preflight` cannot run yet and fails with:
>
> ```
> resolve desired release: get ProductManifest for version "<target-version>": no matches for kind "ProductManifest" in version "config.alauda.io/v1"
> ```
>
> The message names an API group, but the cause is ordering, not a version mismatch: the `ProductManifest` custom resource definition and the target-version `ProductManifest` are both created by Deploy the cluster version operator. Run that step first, then preflight. On a cluster whose previous upgrade already used the CVO workflow, the resources exist and preflight runs at any point after artifact synchronization.

Preflight returns two parts:

Output	Purpose
`Summary`	Shows the overall result, current version, desired version, and desired image.
`Checks`	Shows the result of each individual validation item.

The default check set includes:

- `ResourcePatchUpgradeable`
- `ClusterVersionUpgradeable`
- `AdminAckRequired`
- `VersionUpgradePath`
- `KubernetesVersionSupported`
- `DockerRuntimeUnsupported`
- `ClusterRunning`
- `ClusterModuleStable`
- `ControlPlaneStaticPodsPresent`
- `CustomEtcdBackupCronJobsAbsent`
- `CRIUpgradePodsAbsent`
- `ModuleInfoStable`
- `PlatformLicense`

3 Handle preflight blocks

When: before the maintenance window, as soon as preflight reports a block.

A clean first preflight run is not the norm on a customized environment. Treat resolving blocks as part of the preparation phase, not as an exception — that is why preflight is run one to two weeks ahead.

If any check does not pass, do not open the maintenance window. Resolve each block with Upgrade Troubleshooting, which covers:

- `ResourcePatchUpgradeable` — the most frequent block on a customized environment, and the target-version exemption it requires.
- Other common blocks — `KubernetesVersionSupported`, `VersionUpgradePath`, `ClusterRunning`, `DockerRuntimeUnsupported`, and module stability.
- Administrator acknowledgement gates — when `AdminAckRequired` fails.

Do not force an upgrade past a failed check. `VersionUpgradePath`, `KubernetesVersionSupported`, and `AdminAckRequired` gate the supported upgrade path itself. If a block cannot be resolved, contact technical support before continuing.

4 Deploy the cluster version operator

When: at the start of the maintenance window.

Run `upgrade.sh` in skip-sync mode. Synchronization is skipped because the artifacts were already uploaded in Sync upgrade artifacts:

```
bash upgrade.sh --skip-sync-image
```

This deploys or updates the cluster version operator (CVO) and completes the remaining preparation. The CVO drives the Core and Aligned plugin upgrade once the upgrade is requested in the next step.

After the command completes, inspect the target `ProductManifest` before requesting the upgrade:

```
kubectl get productmanifest v<target-version> \
  -o jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}
{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'

kubectl get productmanifest v<target-version> -o json | jq -r '
  .status.artifacts[] as $artifact
  | $artifact.channels[]
  | select(.artifactStatus != "Ready")
  | [$artifact.name, .channel, .tag, .artifactStatus]
  | @tsv'
```

The per-artifact output is diagnostic and is not required to be empty. The target manifest can include optional, Agnostic, or uninstalled entries that are legitimately not Ready. Compare the output with the installed Core and Aligned inventory. For an installed Aligned `ModulePlugin`, inspect the matching target `ModulePluginConfig` when its channel is not Ready:

```
kubectl get modulepluginconfig <modulepluginconfig-name> -o yaml
```

Use the component name and channel tag from the target `ProductManifest`, or the name reported in the `ClusterVersionShadow` error, to identify the `ModulePluginConfig`. The component version in this name is not necessarily the ACP Distribution Version. Do not treat `artifactStatus: Absent` alone as proof that the registry manifest is missing; use the `ModulePluginConfig` condition and its `.spec.image` to distinguish a missing manifest from authentication, CA, network, or package-content failures.

5 Request the upgrade

After the cluster version operator is deployed, request the upgrade through one of the following entry points. The entry points are equivalent; pick the one that fits your operating model.

The platform offers two cluster management interfaces in the Web Console. **Web Console** is the built-in interface that ships with the platform and is the default path. **Web Console (Fleet Essentials)** is the newer cluster management interface, delivered as a separate cluster plugin that must be installed before it can be used.

Web Console

This is the built-in cluster management interface. It is present on every ACP platform and requires nothing to be installed beyond the platform itself.

Requesting the upgrade requires update permission on the target cluster's `ClusterVersionShadow` and `ResourcePatch` resources.

1. In the left navigation bar, click **Clusters** > **Clusters**, then click the `global` cluster.

2. Open the **Functional Components** tab.
   The tab lists each component with its **Status** and **Version**. A component that has a newer version available is marked **Upgradeable**. A component that cannot move to the new version carries a block reason instead, and the page shows which components are not compatible with the new version.
3. Click **Upgrade**.
   The button is disabled when there is no upgradable version (**No upgradable version**), when an upgrade is already running (**Upgrading**), or when the cluster is not healthy (**Cluster Abnormality**). Hover over the button to read the reason.
4. On the **Personalized Configuration Confirmation** step, review the listed configurations, then click **Acknowledge**.
   This step appears **only when the cluster carries personalized configuration** — a customized `ModuleInfo`, a customized `ClusterModule`, or a `ResourcePatch`. It lists each item and renders the selected item's overrides as read-only YAML, under the warning *There are the following personalized configurations in your environment, please confirm with the technical support staff whether you need to delete or adjust, otherwise, your upgrade may fail!*
   When the cluster carries no personalized configuration, this step does not appear and the page opens directly on the confirmation form described in the next step.
5. On the **Confirm Component Upgrade** step, review **Current Version** and **Target Version**.
   The target version is the first entry of `ClusterVersionShadow.status.availableUpdates`. It is determined by the upgrade artifacts already prepared for the cluster and cannot be selected manually in this interface.
6. Click **Upgrade**.
7. In the **Are you sure you want to upgrade the cluster version?** dialog, set **Maintenance mode**, then click **Upgrade** to submit the request.

> **Maintenance mode is switched on by default in this dialog**
>
> The confirmation dialog opens with **Maintenance mode** already enabled. While maintenance mode is on, users who are not platform administrators are restricted from accessing platform features, and the text in **Platform maintenance announcements**

is shown to them. The announcement is required while the switch is on and is limited to 200 characters. Turn the switch off before submitting if the change window does not call for restricting access. The switch writes the platform-wide maintenance mode setting, so review that setting again after the change window closes.

After the request is submitted, the page returns to the **Functional Components** tab and shows *The upgrade has been initiated. It may take 5~10 minute for the status to update.* Click the magnifier icon in a component's **Status** column to follow that component's upgrade progress.

Web Console (Fleet Essentials)

This entry point belongs to the new cluster management interface, delivered as the **Alauda Container Platform Fleet Essentials** cluster plugin. It is **not installed by default**.

Prerequisite: install **Alauda Container Platform Fleet Essentials v1.0.4 or later** from **Marketplace** > **Cluster Plugins**. The plugin adds a **Clusters (New)** entry to the left navigation bar. When that entry is absent, the plugin is not installed on this platform — use the **Web Console** tab instead.

1. In the left navigation bar, click **Clusters (New)**, then click the `global` cluster.
2. Stay on the **Overview** tab and find the **Version & Update** card.
   The card shows **Distribution Version** and **Kubernetes Version**, and a status banner: **Up to date**, **Update Available**, **Upgrading**, **Update Blocked**, or **Upgrade Not Enabled**. **Upgrade Not Enabled** means the cluster version operator is not deployed on the cluster — complete Deploy the cluster version operator first.
   When a target version exists, the card also reports target-artifact readiness as **Target artifacts ready** or **Target artifacts not ready**, with a Core count and an Aligned count. Click **Show per-component** to list which Core and Aligned components are not Ready yet. Core artifacts are synced by `upgrade.sh --only-sync-image`; Aligned plugins are not bundled in the Core Package and need their own packages published with `violet push`. Readiness does not block the request, but a target version only appears once its artifacts are ready.
3. Click **Upgrade** to open the **Update Cluster** dialog.

The button is active only while an update is available. During a run the card shows a disabled **Upgrading**; when a preflight check blocks the upgrade, it shows a disabled **Update Blocked** and lists the blockers in the **Preflight Checks** section of the card. Re-requesting the upgrade does not clear a block — resolve the blocker and the upgrade resumes.

4. On the **Custom Configuration** step, review the resource patches that will be exempted, then click **Next**.
   The table lists each pending `ResourcePatch` with its **Name**, **Target Resource**, **Current Exempt Version**, and **Exempt To (after upgrade)**. Starting the upgrade annotates each of them with the exempt-for-version annotation so the upgrade does not overwrite them. When there is nothing to confirm, the step shows **No custom configuration to confirm**.
5. On the **Confirm & Upgrade** step, review **Current Version** and **Target Version**. On the `global` cluster, **Target Version** is a drop-down list when `ClusterVersionShadow.status.availableUpdates` contains more than one target; select the version whose artifacts you prepared. With a single available target the version is shown read-only. The step also restates the upgrade scope — Core, Aligned, Kubernetes, and Agnostic — and the caution that a cluster upgrade cannot be rolled back.
6. Click **Confirm & Start**.
   This interface has no second confirmation dialog and no maintenance mode switch. The card refreshes into the **Upgrading** state, and the **Preflight Checks**, **Upgrade Progress**, and **Update History** sections of the card track the run.

ACP CLI

Use this entry point when scripting the upgrade or when your current ACP session already targets the global cluster.

```
# Request upgrade to the highest version currently published in availableUpdates
ac adm upgrade --cluster=global --to-latest

# Request upgrade to a specific target version
ac adm upgrade --cluster=global --to=<target-version>

# Show summary, preflight, and stage progress for the global cluster upgrade
ac adm upgrade status --cluster=global
```

For the complete AC CLI workflow and output interpretation, see Upgrading Clusters. For full command and flag syntax, see AC CLI Administrator Command Reference.

kubectl

Use this entry point when you need to operate the underlying CVO resource directly. Patch `ClusterVersionShadow.spec.desiredUpdate` with the target version.

```
kubectl patch cvsh global -n cpaas-system --type merge -p '{
  "spec": {
    "desiredUpdate": {
      "version": "<target-version>"
    }
  }
}'
```

Or edit the resource directly:

```
kubectl edit cvsh global -n cpaas-system
```

Minimum configuration:

```
spec:
  desiredUpdate:
    version: <target-version>
```

6 Observe execution

Use the following command to inspect the overall status:

```
kubectl get cvsh -n cpaas-system
```

Important status fields:

Field	Purpose
`status.conditions`	Overall status entry point.
`status.preflight.observedAt`	Time of the latest preflight run.
`status.preflight.checks`	Detailed result of each preflight item.
`status.current`	Current applied version and image.
`status.desired`	Target version and image being reconciled.
`status.history`	Upgrade history, newest first.
`status.stages`	Upgrade stages and per-stage execution state.

Focus on these conditions first:

Condition	Interpretation
`PreflightReady`	`True` means preflight passed.
`Ready`	`True` means the cluster has reached the desired version.
`Reconciling`	`True` means the upgrade is still running.

Condition	Interpretation
`Stalled`	`True` means the upgrade is blocked and requires intervention.

If `Stalled=True`, follow Upgrade Troubleshooting. To observe an individual plugin or operator module, read its `ModuleInfo`:

```
# Current version, target, next available version, and phase per inst
alled module on this cluster
kubectl get moduleinfo -l cpaas.io/cluster-name=global \
  -o custom-columns='MODULE:.metadata.labels.cpaas\.io/module-name,CU
RRENT:.status.version,TARGET:.spec.version,NEW:.status.availableVersi
ons[0].version,PHASE:.status.phase'

# Conditions and any block reasons for one module
kubectl get moduleinfo <name> \
  -o jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}
{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

A module has reached its target when `status.phase` is `Running` and `status.version` equals the target version.

7 Upgrade Agnostic plugins from Marketplace

CVO drives Core and Aligned plugins. **Agnostic plugins are outside CVO's scope** and must be upgraded individually after the cluster has reached the target Distribution Version. Whether each Agnostic plugin needs to be upgraded depends on its own Kubernetes compatibility — see the plugin's release notes for compatibility with the target Kubernetes version.

For each in-use Agnostic plugin on the global cluster:

1. In the Web Console, switch to **Administrator** view.
2. Navigate to **Marketplace > Cluster Plugins** for Agnostic cluster plugins, or to the operator workflow for Agnostic operators.
3. Select the target plugin or operator and trigger the upgrade. The Marketplace upgrade flow reads the package previously pushed with `violet`.

> If you skipped pushing an Agnostic plugin during pre-upgrade and the Marketplace does not offer the target version, complete the `violet push` step first and then retry the Marketplace upgrade.

Validate the Upgrade

After the cluster reaches the desired version and the in-use Agnostic plugins are handled, complete Upgrade Validation. If the `global` cluster uses Alauda OS, first complete Upgrade Kubernetes on Alauda OS.

Related Documentation

- Overview
- Pre-upgrade
- Upgrade Kubernetes on Alauda OS for the `global` cluster
- Upgrade global clusters in a DR environment
- Upgrade the Distribution Version of workload clusters
- Upgrade validation
- Upgrade troubleshooting
- Upgrading Clusters
- Global Cluster Disaster Recovery

Upgrade Kubernetes on Alauda OS for the `global` Cluster

This document describes how to upgrade Kubernetes on a `global` cluster that runs Alauda OS. Upgrades replace nodes with new Alauda OS images managed by the Cluster API provider; in-place node upgrades are not used.

For a Huawei DCS `global` cluster, select the Alauda OS image published for the target ACP release. If the cluster's VM templates were made from an Alauda OS image published for a release earlier than ACP `v4.3.2`, they do not match DCS Provider `v1.0.21` or later. Upgrade the provider together with the Distribution Version of the `global` cluster, because the platform upgrades the installed Aligned plugins during that upgrade; see Pre-Upgrade Preparation. Then create the VM template from the Alauda OS image published for the target ACP release and replace the nodes by following this document. See Upgrade Kubernetes on Huawei DCS.

TOC

When to Use This Path

Two-Phase Upgrade Overview

Common Prerequisites

Procedure

Step 1 — Upgrade Kube-OVN

Step 2 — Update the Control Plane Resources

Step 3 — Apply the Control Plane Changes and Wait

Step 4 — Update the Worker Resources

Step 5 — Apply the Worker Changes

Step 6 — Monitor the Rolling Replacement

Verification

Recovery Considerations

Next Steps

When to Use This Path

Choose this upgrade path when:

- The `global` cluster was installed on Alauda OS. See Installing the global Cluster.
- Your infrastructure is one of the documented providers: Huawei DCS, VMware vSphere, Huawei Cloud Stack, or Bare Metal.

For a non-Alauda OS `global` cluster, Kubernetes is upgraded in place as part of Upgrade the Distribution Version; this page does not apply.

In a DR pair, run this Kubernetes upgrade on both `global` clusters, each with its own `global` kubeconfig. For the Distribution Version upgrade of the pair, see Upgrade in a DR Environment.

Two-Phase Upgrade Overview

Like workload clusters, an Alauda OS `global` cluster follows a two-phase upgrade.

1. **Phase 1 — ACP Core and Distribution Version**: prepare artifacts and plugin packages, then use CVO to move the global tier to the target Distribution Version. See Pre-Upgrade Preparation and Upgrade the Distribution Version.
2. **Phase 2 — Kubernetes and OS Image**: nodes are replaced with new Alauda OS images that contain the target Kubernetes version. This document focuses on Phase 2 for the `global` cluster.

Phase 1 Compatibility

Before starting Phase 2, verify that every workload cluster falls within the Compatible Versions matrix of the target Distribution Version. Workload clusters that are out of range must be upgraded first.

Common Prerequisites

- The `global` cluster has completed Phase 1 (Distribution Version upgrade).
- The infrastructure provider for the IaaS that the `global` cluster runs on and the Cluster Bootstrap Provider are uploaded and installed on the `global` cluster. If they are not, install them first by following the provider installation page for Bare Metal, Huawei DCS, Huawei Cloud Stack, or VMware vSphere. In a DR pair, install them on both `global` clusters.
- The infrastructure provider version, the Alauda OS image version, and the ACP version match. Phase 1 upgrades the provider together with the Distribution Version of the `global` cluster.
- An etcd backup of the `global` cluster has been taken and verified.
- The target row in OS Support Matrix is available, and its Kubernetes version, CoreDNS tag, etcd tag, and Kube-OVN chart are staged, together with the Alauda OS image published for that release.
- The previous machine templates and bootstrap templates are retained until Phase 2 is verified.
- A maintenance window plan that accounts for rolling control plane replacement.
- For cross-version upgrades that span more than one Kubernetes minor, the intermediate-version Core images and OS images are pre-staged. See Cross-Version Upgrade Preparation.

Procedure

The Cluster API controllers that manage the `global` cluster run on the `global` cluster itself. Use the `global` kubeconfig for the `kubectl` commands in this procedure.

> **Required when the target is Kubernetes 1.35**
>
> In Kubernetes 1.35, kubelet credential verification can apply to images that are already present on a node. On the hop to Kubernetes 1.35 or later, add `imagePullCredentialsVerificationPolicy: NeverVerify` to the kubelet patch used by every replacement node. Update the control-plane file in the same `KubeadmControlPlane` edit as `spec.version`. For workers, create a new `KubeadmConfigTemplate` with the updated patch and switch `MachineDeployment.spec.template.spec.bootstrap.configRef.name` in the same edit as the version. Do not add this parameter for Kubernetes 1.34 or earlier. See Required kubelet patch for Kubernetes 1.35.

Step 1 — Upgrade Kube-OVN

Read the **kube-ovn (chart)** value from the target OS Support Matrix row, then follow the shared provider-version-specific Kube-OVN procedure with `<cluster-name>` set to `global`.

That procedure first checks the installed provider revision, then selects the supported new or legacy flow, handles the Kube-OVN chart-name boundary at `v4.4`, and verifies the chart name, target revision, installed revision, phase, and conditions. Use the `global` kubeconfig for both the provider and `cni-kube-ovn` `AppRelease` checks. Do not begin control-plane replacement until all shared verification criteria pass.

Step 2 — Update the Control Plane Resources

Update the control plane resources of the `global` cluster to reference the new Alauda OS image and Kubernetes version. The fields to update are provider-specific. Update the worker node resources in Step 4.

Huawei DCS

Before you start, see Prerequisites.

For DCS, create new infrastructure templates instead of editing templates that are already referenced by running machines.

Update the control plane resources:

- Create a new `DCSMachineTemplate` for the target image and set `spec.template.spec.vmTemplateName` to the Alauda OS template that matches the target Kubernetes version.
- Keep preserved node-local data, including `/var/cpaas`, in `DCSIpHostnamePool.spec.pool[].persistentDisk`. Do not move preserved disks back into `DCSMachineTemplate`.
- Set `KubeadmControlPlane.spec.version` to the target Kubernetes version.
- Set the CoreDNS and etcd image tags in `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration` from the same OS Support Matrix row.
- When the target is Kubernetes 1.35 or later, update the control-plane kubelet patch in `KubeadmControlPlane.spec.kubeadmConfigSpec.files` in the same manifest edit.
- Point `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` to the new `DCSMachineTemplate`.
- Keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` when the cluster uses pool-managed persistent disks.

Pool-managed persistent disks are declared on the IP pool, not on the machine template:

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <global-pool-name>
  namespace: cpaas-system
spec:
  pool:
    - ip: <node-ip>
      hostname: <node-hostname>
      persistentDisk:
        - slot: 0
          quantityGB: 40
          datastoreName: <datastore-name>
          path: /var/cpaas
          format: xfs
          mountOptions:
            - defaults
```

Use `isThin` only when the DCS environment requires an explicit thin-provisioning value. If omitted, the provider does not send `isThin` and DCS uses the platform default. New persistent volumes are created as independent persistent normal volumes.

Use the IP pool status to confirm that preserved disks are detached from old VMs and attached to replacement VMs during the rolling replacement.

VMware vSphere

Before you start, see Check the manifests against the admission rules.

For vSphere, create new infrastructure templates instead of editing templates that are already referenced by running machines.

Update the control plane resources:

- Export the current `VSphereMachineTemplate`, give the copy a new name, remove server-generated metadata and `status`, and set `spec.template.spec.template` to the VM template created in vSphere from the Alauda OS image published for the target ACP release.

- Preserve the existing vCenter, folder, datastore, network, clone-mode, CPU, memory, and disk configuration unless the release procedure explicitly changes it. Leave `spec.template.spec.providerID` unset.
- Keep preserved node-local data, including `/var/cpaas`, in `VSphereMachineConfigPool.spec.configs[].persistentDisks[]`; do not move it into the machine template.
- Set `KubeadmControlPlane.spec.version`, CoreDNS, and etcd tags from the same OS Support Matrix row.
- When the target is Kubernetes 1.35 or later, update the control-plane kubelet patch in `KubeadmControlPlane.spec.kubeadmConfigSpec.files` in the same manifest edit.
- Point `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` to the new `VSphereMachineTemplate`.
- Keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` when the control plane pool uses fixed slots or pool-managed persistent disks.

Huawei Cloud Stack

Before you start, see Prerequisites.

For HCS, create new infrastructure templates instead of editing templates that are already referenced by running machines.

Update the control plane resources:

- Create a new `HCSMachineTemplate` for the target image and set `spec.template.spec.imageName` to the Alauda OS image that matches the target Kubernetes version.
- Keep preserved node-local data, including `/var/cpaas`, in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`. Do not move preserved disks back into `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.
- Leave runtime identity fields unset in the new template, including `spec.template.spec.providerID` and `spec.template.spec.serverId`.
- Set `KubeadmControlPlane.spec.version` to the target Kubernetes version.

- Set the CoreDNS and etcd image tags in `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration` from the same OS Support Matrix row.
- When the target is Kubernetes 1.35 or later, update the control-plane kubelet patch in `KubeadmControlPlane.spec.kubeadmConfigSpec.files` in the same manifest edit.
- Point `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` to the new `HCSMachineTemplate`.
- Keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` when the control plane pool uses pool-managed persistent disks.

Pool-managed persistent disks are declared on the machine configuration pool, not on the machine template:

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <global-pool-name>
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: global
spec:
  configs:
    - hostname: <node-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <node-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
          mountOptions:
            - defaults
            - noatime
```

Do not treat HCS `dataVolumes[]` as preserved state during node replacement. This rolling upgrade workflow supports highly available control planes that can replace `HCSMachineTemplate` and `KubeadmControlPlane` references while

`HCSMachineConfigPool` retains enough fixed identities and persistent disk slots for the rollout. Single-control-plane HCS clusters, including a `global` cluster with one control plane node, cannot be upgraded.

Bare Metal

Before you start, see Check COS_STATE Free Space and Update the Image Catalog.

Use this YAML workflow only after the Bare Metal provider is installed on the `global` cluster and the target OS Support Matrix row is available in the installed release. Before editing Cluster API resources, confirm that the provider `AppRelease` requested and installed revisions match and its chart phase is `Success`.

Import the target release's Bare Metal OS images into the `global` cluster's registry and add the target Kubernetes version to its `elemental-image-catalog`, as described in OS Images Imported and Image Catalog Populated. Neither the provider plugin nor the Distribution Version upgrade adds the version for you. In a DR pair, do this on both `global` clusters. Then verify the catalog:

```
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get configmap elemental-image-catalog -o yaml
```

The target version must appear as a key with a reachable base-image reference. A replacement `Machine` created while the key is missing leaves its `BaremetalMachine` in `Failed` with `Reason=ImageCatalogMiss`, and adding the key afterwards does not recover it. The Bare Metal provider resolves the replacement image from `Machine.spec.version`; a Kubernetes-only upgrade does not require a new `BaremetalMachineTemplate` unless the control plane or workers must move to a different `MachineInventoryPool`.

Update the existing `KubeadmControlPlane` manifest with values from the same OS Support Matrix row. Preserve all fields not shown here:

```
spec:
  version: <target-kubernetes-version>
  rolloutStrategy:
    rollingUpdate:
      maxSurge: 0
  kubeadmConfigSpec:
    clusterConfiguration:
      dns:
        imageTag: <target-coredns-tag>
      etcd:
        local:
          imageTag: <target-etcd-tag>
```

Retain the existing `spec.kubeadmConfigSpec.files` entries. For Kubernetes 1.35 or later, update the existing `kubeletconfiguration0+strategic.json` entry in that list as documented above; do not replace the list with an empty value.

Apply the complete retained control plane manifests in Step 3. The provider deletes each old machine, cleans its inventory, allocates an available inventory from the same pool, and writes a `reprovision` plan for the target image.

Step 3 — Apply the Control Plane Changes and Wait

Apply the updated control plane manifests against the `global` cluster.

```
kubectl --kubeconfig <global-kubeconfig> apply -f <updated-control-plane-manifest>
```

The Cluster API provider begins replacing control plane nodes by using the new image. When `maxSurge: 0` is set, each old node is drained and deleted before its replacement can reuse the same fixed identity, IP address, or preserved disk.

Wait until all control plane nodes have been replaced.

```
kubectl --kubeconfig <global-kubeconfig> get machines -A -o wide
kubectl --kubeconfig <global-kubeconfig> get kubeadmcontrolplane -A
```

The control plane replacement is complete when every control plane `Machine` reports the new Kubernetes version and `Phase: Running`, and the `KubeadmControlPlane` reports `Ready: True` against the new version.

Step 4 — Update the Worker Resources

Update the worker node resources of the `global` cluster to reference the new Alauda OS image and Kubernetes version. The fields to update are provider-specific.

Huawei DCS

Update worker node resources:

- Create a new worker `DCSMachineTemplate` with the target `vmTemplateName`.
- Set each `MachineDeployment.spec.template.spec.version` to the target Kubernetes version.
- When the target is Kubernetes 1.35 or later, create a new worker `KubeadmConfigTemplate` with the required kubelet patch and point `MachineDeployment.spec.template.spec.bootstrap.configRef.name` to it.
- Point each `MachineDeployment.spec.template.spec.infrastructureRef.name` to the new worker `DCSMachineTemplate`.
- Keep each `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` when the worker pool uses pool-managed persistent disks.

VMware vSphere

Update worker node resources:

- Create a new worker `VSphereMachineTemplate` with the target VM template and the retained provider configuration.
- Set each `MachineDeployment.spec.template.spec.version` to the target Kubernetes version.
- When the target is Kubernetes 1.35 or later, create a new worker `KubeadmConfigTemplate` with the required kubelet patch and point `MachineDeployment.spec.template.spec.bootstrap.configRef.name` to it.

- Point each `MachineDeployment.spec.template.spec.infrastructureRef.name` to the new worker `VSphereMachineTemplate`.
- Keep each `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` when the worker pool uses fixed slots or pool-managed persistent disks.

Do not edit a template that running machines reference. Keep the previous control-plane and worker templates until the new rollout is verified.

Huawei Cloud Stack

Update worker node resources:

- Create a new worker `HCSMachineTemplate` with the target `imageName`.
- Set each `MachineDeployment.spec.template.spec.version` to the target Kubernetes version.
- When the target is Kubernetes 1.35 or later, create a new worker `KubeadmConfigTemplate` with the required kubelet patch and point `MachineDeployment.spec.template.spec.bootstrap.configRef.name` to it.
- Point each `MachineDeployment.spec.template.spec.infrastructureRef.name` to the new worker `HCSMachineTemplate`.
- Keep each `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` when the worker pool uses pool-managed persistent disks.

Bare Metal

For each worker `MachineDeployment`, set the target Kubernetes version and keep `maxSurge: 0`. When the target is Kubernetes 1.35 or later, first create a new `KubeadmConfigTemplate` containing the required kubelet patch, then reference it in the same manifest change:

```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 0
  template:
    spec:
      version: <target-kubernetes-version>
      bootstrap:
        configRef:
          name: <target-kubeadm-config-template>
```

Omit the `bootstrap.configRef` change for an earlier target when no bootstrap setting changes. Apply the complete retained worker manifests in Step 5.

Step 5 — Apply the Worker Changes

Apply the updated worker manifests against the `global` cluster.

```
kubectl --kubeconfig <global-kubeconfig> apply -f <updated-worker-manifest>
```

The Cluster API provider begins replacing worker nodes by using the new image. When `maxSurge: 0` is set, each old node is drained and deleted before its replacement can reuse the same fixed identity, IP address, or preserved disk.

Step 6 — Monitor the Rolling Replacement

Watch the rolling replacement until all worker nodes have been replaced.

```
kubectl --kubeconfig <global-kubeconfig> get machines -A -o wide
kubectl --kubeconfig <global-kubeconfig> get kubeadmcontrolplane -A
```

The upgrade is complete when every `Machine` reports the new Kubernetes version and `Phase: Running`, and the `KubeadmControlPlane` reports `Ready: True` against the new version.

Verification

After the rolling replacement finishes, verify that the upgraded `global` cluster is healthy.

```
kubectl --kubeconfig <global-kubeconfig> get nodes -o wide
kubectl --kubeconfig <global-kubeconfig> get clusterversionshadow -o yaml
kubectl --kubeconfig <global-kubeconfig> get pods -n cpaas-system
```

All nodes must report the new Kubernetes version, the `ClusterVersionShadow` must reflect the target Distribution Version, and core platform pods must be `Running`.

Recovery Considerations

Do not treat a Kubernetes minor downgrade as an ordinary rollback. Choose the recovery path from the rollout stage:

1. **No target-version control-plane `Machine` has been created**: restore the previous Kube-OVN state and the previous manifest values. This cancels the target rollout before a new control-plane data format is introduced.
2. **Only the machine template or OS image changed, and the Kubernetes minor did not change**: point the controlling resource back to the previous template and let Cluster API perform another replacement rollout. Keep the Kubernetes minor unchanged.
3. **A control-plane `Machine` on the target Kubernetes minor has joined the cluster**: do not patch Kubernetes, CoreDNS, or etcd back to the previous minor. Stop further rollout and repair forward on the target minor, or recover the `global` cluster from the verified pre-upgrade etcd backup or Global DR procedure.

If a target-minor control-plane `Machine` was created but never joined, restore healthy etcd quorum and determine whether the failed replacement can be removed safely before changing manifests.

Huawei DCS

For DCS clusters that use pool-managed persistent disks, confirm disk state before rollback:

First, check `DCSIpHostnamePool.status.persistentDiskStatus` before deleting or recreating machines. Do not delete retained DCS volumes that are listed in `DCSIpHostnamePool.spec.pool[].persistentDisk`.

For stage 1, use the DCS rule in Restore Kube-OVN During Stage-1 Recovery; the action depends on the installed provider version. Keep `maxSurge: 0` while returning to previous machine templates in a same-minor replacement recovery. If a target-minor control-plane machine has joined, retain the target baseline and use forward recovery or backup restore.

VMware vSphere

For stage 1, use the vSphere rule in Restore Kube-OVN During Stage-1 Recovery. For a same-minor template recovery, keep `maxSurge: 0`, point the control plane and worker resources to the retained previous `VSphereMachineTemplate` resources, and verify that each fixed slot reattaches its pool-managed persistent disks. If a target-minor control-plane machine has joined, retain the target Kubernetes, CoreDNS, etcd, and Kube-OVN baseline and use forward recovery or backup restore.

Huawei Cloud Stack

Node-local data on HCS `dataVolumes[]` is not a recovery source because node replacement may delete the old VM and attached volumes. Data declared in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` is reattached during replacement. For stage 1, use the HCS rule in Restore Kube-OVN During Stage-1 Recovery; the action depends on the installed provider version. For a same-minor template recovery, keep `maxSurge: 0` and point the controlling resources to the retained previous `HCSMachineTemplate` resources. If a target-minor control-plane machine has joined, retain the target baseline and use forward recovery or backup restore.

Bare Metal

For stage 1, use the Bare Metal rule in Restore Kube-OVN During Stage-1 Recovery, then restore the previous manifests only if no target-minor control-plane `Machine` has been created.

Every host that completes a `reprovision` plan has already replaced its OS image and cleared Kubernetes-managed state. If a target-minor control-plane machine has joined, do not patch Kubernetes, CoreDNS, or etcd back to the previous minor. Repair forward, or recover the `global` cluster through the verified etcd backup or Global DR procedure.

Next Steps

- Upgrade workload clusters to the same Distribution Version: see Upgrade the Distribution Version of Workload Clusters, then Upgrade Kubernetes on Alauda OS for Workload Clusters.
- Review machine configuration changes that ship with the new image: see Machine Configuration.

Upgrade Global Clusters in a DR Environment

Use this procedure when the environment includes both a primary global cluster and a standby global cluster. The DR-specific steps are in addition to the standard CVO workflow in Upgrade the global cluster.

TOC

Upgrade Procedure

Verify the DR environment before upgrading

Uninstall the etcd synchronization plugin from the standby global cluster

Sync upgrade artifacts on both global clusters

Upgrade the standby global cluster

Upgrade the primary global cluster

Reinstall the etcd synchronization plugin and verify sync status

Upgrade Procedure

1 Verify the DR environment before upgrading

Follow your regular global DR inspection procedures to ensure that data in the **standby global cluster** is consistent with the **primary global cluster**. For background on the DR topology and synchronization workflow, see Global Cluster Disaster Recovery.

If inconsistencies are detected, do not uninstall the etcd synchronization plugin in the next step, and contact technical support before proceeding. Uninstalling the plugin while the standby global cluster is missing data that the primary holds can cause owner references to resolve incorrectly, and workload-cluster node `Machine` objects — including Alauda OS clusters, where this destroys the backing virtual machine — may be deleted.

On **both** global clusters, run the following command to ensure no `Machine` nodes are in a non-running state:

```
kubectl get machines.platform.tkestack.io
```

If any such nodes exist, resolve them before continuing.

2 Uninstall the etcd synchronization plugin from the standby global cluster

1. Access the Web Console of the **standby global cluster** through its IP or VIP.
2. Switch to **Administrator** view.
3. Navigate to **Marketplace > Cluster Plugins** and select the `global` cluster.
4. Find **Alauda Container Platform Global Cluster Disaster Recovery** and uninstall it.
5. Wait for the uninstallation to complete before proceeding.

3 Sync upgrade artifacts on both global clusters

Complete Sync upgrade artifacts on both the standby global cluster and the primary global cluster. Use the same registry type and exact target-version payload on both clusters. For an external registry, complete both upload modes for each registry endpoint that the two global clusters use.

4 Upgrade the standby global cluster

If you will use the Web Console on the standby global cluster, verify that the standby cluster `ProductBase` includes the standby VIP in `spec.alternativeURLs`:

```
apiVersion: product.alauda.io/v1alpha2
kind: ProductBase
metadata:
  name: base
spec:
  alternativeURLs:
    - https://<standby-cluster-vip>
```

After synchronization completes, run the remaining standard workflow steps on the **standby global cluster**:

1. Run preflight checks.
2. Deploy the cluster version operator.
3. Request the upgrade.
4. Observe execution until the standby global cluster reaches the desired version.

5 Upgrade the primary global cluster

After the standby global cluster reaches the desired version, run the remaining standard workflow steps on the **primary global cluster**:

1. Run preflight checks.
2. Deploy the cluster version operator.
3. Request the upgrade.
4. Observe execution until the primary global cluster reaches the desired version.

6 Reinstall the etcd synchronization plugin and verify sync status

Before reinstalling the plugin, verify that port `2379` is forwarded correctly from both global-cluster VIPs to their control plane nodes when that forwarding mode is used. Port

forwarding through a load balancer is not required if the standby global cluster can access the active global cluster directly.

Get the bearer token for the active global cluster API server from the active cluster, then use it to create or update the `etcd-sync-active-cluster-token` Secret on the standby cluster. Keep the token in the current shell only and clear it after the Secret is created.

```
# Run on the active cluster and copy the output without storing it in a document.
kubectl -n cpaas-system get secret k8sadmin -o jsonpath='{.data.token}' | base64 -d
```

Run on the standby cluster:

```
read -rsp "Active global cluster token: " ACTIVE_CLUSTER_TOKEN
printf '\n'
kubectl -n cpaas-system create secret generic etcd-sync-active-cluster-token \
  --from-literal=token="${ACTIVE_CLUSTER_TOKEN}" \
  --dry-run=client -o yaml | kubectl apply -f -
unset ACTIVE_CLUSTER_TOKEN
```

To reinstall the plugin:

1. Access the **standby global cluster** Web Console through its VIP and switch to **Administrator** view.
2. Navigate to **Marketplace > Cluster Plugins** and select the `global` cluster.
3. Find **Alauda Container Platform Global Cluster Disaster Recovery**, click **Install**, and configure the required parameters.

When you configure the plugin:

- Set **Active Global Cluster VIP** to the VIP of the active global cluster.
- When port `2379` is not forwarded through a load balancer, set **Active Global Cluster ETCD Endpoints** correctly.

- Set **Standby Cluster ETCD Endpoints** to the standby cluster etcd address. Use the default value unless the local etcd service is exposed through a different endpoint.
- Set **Active Global Cluster Token Secret** to `etcd-sync-active-cluster-token`.
- Use the default value of **Data Check Interval**.
- Leave **Print detail logs** disabled unless you are troubleshooting.

During reinstallation, the system runs the `etcd-sync-bootstrap` Job before the `etcd-sync` Deployment starts. Verify the bootstrap Job and runtime resources:

```
kubectl get job -n cpaas-system etcd-sync-bootstrap
kubectl logs -n cpaas-system job/etcd-sync-bootstrap
kubectl get secret -n cpaas-system remote-etcd-ca
kubectl get issuer -n cpaas-system remote-etcd-issuer
kubectl get certificate -n cpaas-system remote-etcd-client
kubectl get secret -n cpaas-system remote-etcd-client
```

Verify the sync Pods and current leader on the standby global cluster:

```
kubectl get po -n cpaas-system -l app=etcd-sync
kubectl get lease -n cpaas-system etcd-sync-mirror
leader_pod=$(kubectl get lease -n cpaas-system etcd-sync-mirror -o js
onpath='{.spec.holderIdentity}')
kubectl logs -n cpaas-system "$leader_pod" | grep -E "Acquired leader
lease|Start Sync update"
```

If resources with `ownerReference` dependencies need to be resynchronized, recreate the current leader Pod after `Start Sync update` appears:

```
leader_pod=$(kubectl get lease -n cpaas-system etcd-sync-mirror -o js
onpath='{.spec.holderIdentity}')
kubectl delete po -n cpaas-system "$leader_pod"
```

Check sync status:

```
mirror_svc=$(kubectl get svc -n cpaas-system etcd-sync-monitor -o jso
npath='{.spec.clusterIP}')
ipv6_regex="^[0-9a-fA-F:]+$"
if [[ $mirror_svc =~ $ipv6_regex ]]; then
  mirror_host="[$mirror_svc]"
else
  mirror_host="$mirror_svc"
fi
curl -g "http://${mirror_host}/check"
```

- `LOCAL ETCD missed keys`: Keys exist in the primary global cluster but are missing from the standby. This often resolves after restarting the current `etcd-sync` leader Pod.
- `LOCAL ETCD surplus keys`: Keys exist in the standby global cluster but not in the primary. Review these with your operations team before deleting them.

After verification succeeds, remove any remaining legacy plain-token configuration from the plugin settings or release values. If the active-cluster token or remote `etcd-ca` changes later, run the plugin upgrade or reinstall workflow again so `etcd-sync-bootstrap` refreshes the runtime credentials and certificates.

After both global clusters are healthy at the target version and synchronization is restored, continue with Upgrade Workload Clusters or Upgrade Validation. If the `global` clusters run Alauda OS, first run Upgrade Kubernetes on Alauda OS on both `global` clusters.

Upgrade Workload Clusters

Upgrade the Distribution Version

Upgrade the Distribution Version

Upgrade Kubernetes on Alauda OS

Bare Metal

Huawei DCS

Huawei Clo

VMware vSphere

Migrate Existing DCS Clusters t

Migrate Exi

Upgrade the Distribution Version of Workload Clusters

Upgrade Path

This page applies to both node operating systems. On a non-Alauda OS workload cluster, this procedure also upgrades Kubernetes in place, and the cluster upgrade is complete when it finishes. For workload clusters that use Alauda OS, complete this page first, then continue with Upgrade Kubernetes on Alauda OS to roll out Kubernetes by replacing nodes.

Workload clusters can be upgraded after the global tier has already reached the target ACP Distribution Version. The global tier does not need to be upgraded again if it is already at that Distribution Version.

Workload cluster upgrades use the same CVO-based workflow as the `global` cluster: confirm prerequisites, run preflight checks, request the upgrade, and observe execution. This workflow has two additional rules:

- If the platform uses **global DR**, both the standby and primary global clusters must reach the target Distribution Version before any workload cluster is upgraded to that Distribution Version.
- The target version normally becomes available for a workload cluster only after the global tier has already reached that same Distribution Version.

A workload cluster upgrade is staged like the global cluster upgrade, but shorter. A workload cluster does not have its own cluster version operator — the CVO runs on the global cluster and drives workload upgrades centrally — so a workload cluster has no artifact-synchronization or CVO-deployment step of its own:

Phase	When	What happens
1. Preflight	1–2 weeks before the window	Confirm prerequisites and run `upgrade.sh --cluster=<cluster> --preflight`; resolve every blocking item before the window opens.
2. Upgrade	During the maintenance window	Request the upgrade and observe execution until the cluster reaches the target version.

TOC

- Workflow
  - Confirm workload-cluster prerequisites
  - Ensure Aligned operator packages are available
  - Run preflight checks
  - Request the upgrade
  - Observe progress
  - Upgrade Agnostic plugins from Marketplace
- Validate the Upgrade
- Related Documentation

Workflow

1 Confirm workload-cluster prerequisites

Before requesting the upgrade, verify that:

- The workload cluster version is lower than the current `global` cluster version.
- The global tier has already reached the target ACP Distribution Version.

- In global DR environments, both the standby and primary global clusters have already reached that target Distribution Version.

2 Ensure Aligned operator packages are available

The recommended pattern in Upgrade the global cluster pushes operator packages to every ACP cluster during the global upgrade window. If the recommendation was followed, the workload cluster already has every operator package it needs and you can skip this section.

Cluster plugins do not need a per-workload-cluster publication. A cluster plugin made available to the global tier through `upgrade.sh` or `violet` is installable and upgradable on every cluster in the platform.

If the global window did not push operator packages to this workload cluster — for example, the cluster was offline during the global window or was added later — push the target-version package for each Aligned operator installed on this workload cluster before the upgrade request:

```
violet push <path/to/operator-package> \
  --platform-address "https://<your-platform-domain>" \
  --platform-token "<platform_token>" \
  --clusters "<workload-cluster-name>"
```

You only need to push packages for Aligned operators actually installed on the workload cluster. Aligned cluster plugin packages made available to the global tier are available to all workload clusters and do not need per-cluster publication.

If an installed Aligned operator's target package is unavailable, CVO can wait for a matching target `InstallPlan` and then report an error while retrying the existing request. Inspect the operator's catalog, `Subscription`, `InstallPlan`, and `ModuleInfo`; publish a missing package with the command above, or correct the reported catalog or registry problem. Continue observing the existing request; you do not need to submit it again. Packages for Aligned operators that are not installed on this workload cluster do not block its upgrade.

3 Run preflight checks

When: 1–2 weeks before the maintenance window, so there is time to resolve any blocking items before the window opens.

Run `upgrade.sh` in preflight mode for the target cluster:

```
bash upgrade.sh --cluster=<cluster> --preflight
```

Preflight is read-only — it validates upgrade readiness and does not change cluster state.

Preflight returns two parts:

Output	Purpose
`Summary`	Shows the overall result, current version, desired version, and desired image.
`Checks`	Shows the result of each individual validation item.

If preflight does not pass, do not submit the upgrade request until all blocking items are resolved.

For preflight block handling patterns, see Upgrade Troubleshooting.

4 Request the upgrade

When: during the maintenance window, after preflight passes.

Request the upgrade through one of the following entry points. The entry points are equivalent; pick the one that fits your operating model.

The platform offers two cluster management interfaces in the Web Console. **Web Console** is the built-in interface that ships with the platform and is the default path. **Web Console (Fleet Essentials)** is the newer cluster management interface, delivered as a separate cluster plugin that must be installed before it can be used.

Web Console

This is the built-in cluster management interface. The workload-cluster upgrade uses the same pages as the global-cluster upgrade, opened on the workload cluster instead of `global`.

Requesting the upgrade requires update permission on the target cluster's `ClusterVersionShadow` and `ResourcePatch` resources.

1. In the left navigation bar, click **Clusters** > **Clusters**, then click the workload cluster.
2. Open the **Functional Components** tab, and click **Upgrade**.
   The button is disabled when there is no upgradable version (**No upgradable version**), when an upgrade is already running (**Upgrading**), or when the cluster is not healthy (**Cluster Abnormality**). A workload cluster has no upgradable version until the global tier has reached the target Distribution Version.
3. On the **Personalized Configuration Confirmation** step, review the listed configurations, then click **Acknowledge**.
   This step appears **only when the cluster carries personalized configuration** — a customized `ModuleInfo`, a customized `ClusterModule`, or a `ResourcePatch`. When the cluster carries none, the step does not appear and the page opens directly on the confirmation form.
4. On the **Confirm Component Upgrade** step, review **Current Version** and **Target Version**.
   For a workload cluster the target version is the Distribution Version the global tier has already reached; it cannot be selected manually in this interface.
5. Click **Upgrade**, then confirm in the **Are you sure you want to upgrade the cluster version?** dialog.

> **Maintenance mode is switched on by default, and applies to the whole platform**
>
> The confirmation dialog is shared with the global-cluster upgrade and opens with **Maintenance mode** already enabled. The switch writes the **platform-wide** maintenance mode setting, not a workload-cluster setting: while it is on, users who are not platform administrators are restricted from accessing platform features across the platform, and the text in **Platform maintenance announcements** is shown to them.

Turn the switch off before submitting unless you intend to restrict the whole platform for the duration of this workload-cluster upgrade.

After the request is submitted, the page returns to the **Functional Components** tab and shows *The upgrade has been initiated. It may take 5~10 minute for the status to update.*

Web Console (Fleet Essentials)

This entry point belongs to the new cluster management interface, delivered as the **Alauda Container Platform Fleet Essentials** cluster plugin. It is **not installed by default**.

Prerequisite: install **Alauda Container Platform Fleet Essentials v1.0.4 or later** from **Marketplace** > **Cluster Plugins**. The plugin adds a **Clusters (New)** entry to the left navigation bar. When that entry is absent, the plugin is not installed on this platform — use the **Web Console** tab instead.

1. In the left navigation bar, click **Clusters (New)**, then click the workload cluster.
2. Stay on the **Overview** tab and find the **Version & Update** card.
   The banner reads **Target not ready** while the global tier has not yet reached the target Distribution Version. In a global DR environment, both the primary and the standby global cluster must reach the target version first.
3. Click **Upgrade** to open the **Update Cluster** dialog.
4. On the **Custom Configuration** step, review the pending `ResourcePatch` entries that will be exempted to the target version, then click **Next**.
5. On the **Confirm & Upgrade** step, review **Current Version** and **Target Version**. For a workload cluster the target equals the current global cluster version and is read-only — the drop-down list offered on the `global` cluster does not apply here.
6. Click **Confirm & Start**.
   This interface has no second confirmation dialog and no maintenance mode switch. The card refreshes into the **Upgrading** state, and the **Preflight Checks**, **Upgrade Progress**, and **Update History** sections of the card track the run.

ACP CLI

Use ACP CLI when your current ACP session can access the target workload cluster. To avoid ambiguity, specify the target cluster explicitly with `--cluster`. ACP CLI is not used to upgrade the `global` cluster.

```
# Request upgrade to the highest version currently published in availableUpdates
ac adm upgrade --cluster=<cluster> --to-latest

# Request upgrade to a specific target version
ac adm upgrade --cluster=<cluster> --to=<target-version>
```

ACP CLI submits the requested target for the specified workload cluster. Whether the upgrade can proceed is still determined by the cluster's available upgrade targets and the upgrade controller's preflight checks.

If you are new to ACP CLI, see Getting Started with ACP CLI. For session and cluster selection, see Managing CLI Profiles. For the complete AC CLI upgrade workflow (metadata preparation, available updates, request modes, and status interpretation), see Upgrading Clusters. For full command and flag syntax, see AC CLI Administrator Command Reference.

5 Observe progress

After the upgrade request is submitted, use `cvsh.status` to track current version, desired version, preflight results, stages, and history:

```
kubectl get cvsh <cluster> -n cpaas-system
kubectl get cvsh <cluster> -n cpaas-system -o yaml
```

If your current ACP context points to the target workload cluster, you can also check the CLI-reported status:

```
# Show summary, preflight, and stage progress for the target cluster upgrade
ac adm upgrade status
```

For detailed semantics of `ac adm upgrade status` output (preflight and stage interpretation), see Upgrading Clusters.

If the upgrade reports `Stalled=True`, use Upgrade Troubleshooting. Missing Aligned operator and cluster-plugin packages have different recovery paths.

The commands above track the cluster-level (Core and Aligned) upgrade. To observe an individual plugin or operator module — for example while upgrading an Agnostic plugin — read its `ModuleInfo`:

```
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster> \
  -o custom-columns='MODULE:.metadata.labels.cpaas\.io/module-name,CURRENT:.status.version,TARGET:.spec.version,NEW:.status.availableVersions[0].version,PHASE:.status.phase'
```

A module has reached its target when `status.phase` is `Running` and `status.version` equals the target version.

6 Upgrade Agnostic plugins from Marketplace

CVO drives Core and Aligned plugins. **Agnostic plugins are outside CVO's scope** and must be upgraded individually after this workload cluster has reached the target Distribution Version.

For each in-use Agnostic plugin on this workload cluster:

1. In the Web Console, switch to **Administrator** view.
2. Navigate to **Marketplace > Cluster Plugins** for Agnostic cluster plugins, or to the operator workflow for Agnostic operators.
3. Select the target plugin or operator and trigger the upgrade.

Whether each Agnostic plugin needs to be upgraded in this window depends on its own Kubernetes compatibility — see the plugin's release notes for compatibility with the new

> Kubernetes version that the workload cluster has reached.
>
> If the Marketplace does not offer the target version of an operator on this workload cluster, follow Ensure Aligned operator packages are available for that operator first and then retry the Marketplace upgrade.

Validate the Upgrade

After the workload cluster and its in-use Agnostic plugins reach their targets, complete Upgrade Validation. If the workload cluster uses Alauda OS, first complete Upgrade Kubernetes on Alauda OS.

Related Documentation

- Overview
- Upgrade the Distribution Version of the `global` cluster
- Upgrade Kubernetes on Alauda OS for workload clusters
- Upgrade validation
- Upgrade troubleshooting
- Upgrading Clusters

Upgrade Kubernetes on Alauda OS for Workload Clusters

This section provides instructions for upgrading Kubernetes on Alauda OS workload clusters managed by Cluster API.

TOC

- Upgrade Overview
  - Prerequisites
- Phase 1: Upgrade ACP Core and Distribution Version
  - Post-Upgrade Verification
- Phase 2: Upgrading Kubernetes Version
  - Upgrade Kube-OVN Before the Control Plane
    - Identify the Installed Provider Version
    - Apply the Provider-Version-Specific Procedure
    - Verify the Kube-OVN AppRelease
    - Restore Kube-OVN During Stage-1 Recovery
  - Required kubelet patch for Kubernetes 1.35
- Preparing for Cross-Version Upgrades
  - Step 1 — Confirm Core hops and pre-sync intermediate images
  - Step 2 — Stage intermediate OS images for the provisioning mechanism
  - Step 3 — Apply the platform-specific procedure for each intermediate Kubernetes/OS row
- Platform-Specific Instructions

Related Topics

Upgrade Overview

Cluster upgrades on Alauda OS follow a two-phase approach:

1. **Phase 1**: Upgrade ACP Core and the cluster Distribution Version through CVO. See Upgrade the Distribution Version.
2. **Phase 2**: Upgrade Kubernetes and replace nodes with the matching Alauda OS image. This section describes Phase 2.

> **INFO**
>
> Phase 1 prerequisites, supported ACP upgrade paths, artifact preparation, and CVO procedures are described in the pages listed under Phase 1 below. This section describes the Alauda OS Kubernetes and OS replacement procedures in Phase 2.

For how the Distribution Version relates to the Kubernetes and OS baseline, see Distribution Version vs Kubernetes Version. Use OS Support Matrix for the exact patch-specific Kubernetes and component values offered for Phase 2.

Prerequisites

- Complete Pre-Upgrade Preparation, including validation that the source ACP release can upgrade to the target Distribution Version.
- Complete the ACP Core and Distribution Version upgrade before starting Phase 2.
- Verify that the control plane is reachable and healthy.
- Verify that all nodes are in the `Ready` state.
- Take and verify the backups required by the provider-specific procedure.
- Confirm sufficient IP or machine-slot capacity for rolling replacement.

Phase 1: Upgrade ACP Core and Distribution Version

Use the following pages as the single procedure owner for this phase:

- Upgrade Overview
- Pre-Upgrade Preparation
- Upgrade the Distribution Version of the `global` Cluster
- Upgrade the Distribution Version of Workload Clusters

Follow that sequence to prepare artifacts and plugin packages, upgrade the `global` tier, and then move each workload cluster to the target Distribution Version through CVO. Do not use intermediate Kubernetes hops in Phase 2 to bypass an unsupported ACP Core upgrade path.

> **INFO**
>
> Fleet Essentials 1.0.4 and later can use the ACP 4.3-and-later CVO workflow to request a Distribution Version upgrade. That capability covers Phase 1 only. Fleet Essentials does not replace immutable nodes or perform the Kubernetes and Alauda OS rollout in Phase 2; use the provider-specific YAML procedure in this section.

Post-Upgrade Verification

After the Distribution Version upgrade completes:

- Verify that the cluster reports the target Distribution Version and CVO is no longer progressing or blocked.
- Confirm that the target row and all required intermediate rows exist in the OS Support Matrix.
- Proceed to Phase 2 only after the management-side Cluster API providers are healthy.

Phase 2: Upgrading Kubernetes Version

After upgrading the Distribution Version, upgrade Kubernetes and the Alauda OS image through the provider-specific Cluster API procedure.

For Huawei DCS, Huawei Cloud Stack, VMware vSphere, and bare-metal clusters, upgrade the Kube-OVN chart for the current upgrade hop **before** changing `KubeadmControlPlane`. Wait until the workload cluster's `cni-kube-ovn` `AppRelease` reports the target installed revision and `phase=Success`, then upgrade the control plane, followed by worker nodes. Repeat this Kube-OVN-first ordering for every intermediate Kubernetes minor in a cross-version upgrade.

Upgrade Kube-OVN Before the Control Plane

Use this section as the authoritative Kube-OVN procedure for workload and `global` clusters. For every Kubernetes and OS hop, read the **kube-ovn (chart)** version from the matching OS Support Matrix row, complete the provider-version-specific procedure below, and verify the resulting `AppRelease` before changing `KubeadmControlPlane`.

Run provider-version checks and `Cluster` or infrastructure-cluster commands against the management cluster. Run `cni-kube-ovn` `AppRelease` commands against the cluster being upgraded. When upgrading the `global` cluster, both resources are on the `global` cluster, so use the `global` kubeconfig for all commands.

Identify the Installed Provider Version

Do not infer the provider version from the ACP Distribution Version. Inspect the provider `AppRelease` on the management cluster. The resource name can vary, so the following command finds the provider by chart name and prints the requested and installed revisions:

```
kubectl -n cpaas-system get apprelease \
  -o custom-columns='APPRELEASE:.metadata.name,CHART:.spec.source.charts
[*].name,REQUESTED:.spec.source.charts[*].targetRevision,INSTALLED:.statu
s.charts[*].installedRevision,PHASE:.status.charts[*].phase'
```

Find the row whose chart name ends with `chart-cluster-api-provider-dcs`, `chart-cluster-api-provider-hcs`, `chart-cluster-api-provider-vsphere`, or `chart-cluster-api-provider-baremetal`, as appropriate. Use the **installed revision** to select a tab. If the requested and installed revisions differ, or the provider chart is not in `Success`, finish or repair the provider upgrade before starting the cluster upgrade.

Provider	Full-source reconciliation	Legacy behavior retained in this guide
Huawei DCS	`v1.0.22` and later	`v1.0.21` and earlier
Huawei Cloud Stack	`v1.0.4` and later	`v1.0.3` and earlier
VMware vSphere	`v1.0.16` and later	`v1.0.15` and earlier
Bare Metal	`v1.0.0` and later	None; `v1.0.0` is the first supported release for this procedure.

The Kube-OVN chart source also changes at chart version `v4.4`:

Target Kube-OVN chart version	Expected `spec.source.charts[0].name`
Earlier than `v4.4`	`acp/chart-cpaas-kube-ovn`
`v4.4` and later	`acp/chart-kube-ovn`

If the target OS Support Matrix row uses Kube-OVN chart `v4.4` or later, upgrade the infrastructure provider to the full-source reconciliation version in the table before continuing. A manual Kube-OVN source patch is not a substitute for that provider upgrade because the same provider releases also reconcile the required CoreDNS and kube-proxy repository changes. DCS Self-built VIP is a separate creation-time endpoint feature and requires DCS Provider `v1.0.22+` with ACP `v4.4+`; changing an existing external endpoint to internal is not part of this upgrade procedure.

Apply the Provider-Version-Specific Procedure

Huawei DCS

Verify that the DCS infrastructure cluster uses Kube-OVN:

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.spec.networkType}{"\n"}'
# Expected output: kube-ovn
```

DCS v1.0.22+

Update the target chart version on the CAPI `Cluster`. The provider selects the chart name from the target version and reconciles the complete `AppRelease` source and specification.

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e
```

Do not patch the `AppRelease` source manually. Continue with Verify the Kube-OVN AppRelease.

DCS v1.0.21 and earlier

WARNING

This legacy flow is valid only when the target Kube-OVN chart is earlier than `v4.4`. For chart `v4.4` or later, upgrade the DCS provider to `v1.0.22` or later first.

The legacy provider updates `spec.values` but preserves the existing source. Record the target on the `Cluster`, then patch only the existing legacy chart revision on the cluster being upgraded:

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e

kubectl patch apprelease cni-kube-ovn -n cpaas-system --type='json' \
  -p='[{"op":"replace","path":"/spec/source/charts/0/targetRevisio
n","value":"<kube-ovn-version-from-matrix>"}]'
```

Keep the legacy chart name `acp/chart-cpaas-kube-ovn`. Do not use this patch to move an old provider to chart `v4.4` or later.

Huawei Cloud Stack

Verify that the HCS infrastructure cluster uses Kube-OVN:

```
kubectl get hcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.spec.networkType}{"\n"}'
# Expected output: kube-ovn
```

HCS v1.0.4+

Update the target chart version on the CAPI `Cluster`. The provider selects the chart name from the target version and reconciles the complete `AppRelease` source and specification.

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e
```

Do not patch the `AppRelease` source manually. Continue with Verify the Kube-OVN AppRelease.

HCS v1.0.3 and earlier

> **WARNING**

This legacy flow is valid only when the target Kube-OVN chart is earlier than `v4.4`. For chart `v4.4` or later, upgrade the HCS provider to `v1.0.4` or later first.

The legacy provider updates `spec.values` but preserves the existing source. Record the target on the `Cluster`, then patch only the existing legacy chart revision on the cluster being upgraded:

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e

kubectl patch apprelease cni-kube-ovn -n cpaas-system --type='json' \
  -p='[{"op":"replace","path":"/spec/source/charts/0/targetRevisio
n","value":"<kube-ovn-version-from-matrix>"}]'
```

Keep the legacy chart name `acp/chart-cpaas-kube-ovn`. Do not use this patch to move an old provider to chart `v4.4` or later.

VMware vSphere

Verify that the CAPI `Cluster` enables Kube-OVN reconciliation:

```
kubectl get cluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.metadata.annotations.cpaas\.io/network-type}{"\n"}'
# Expected output: kube-ovn
```

vSphere v1.0.16+

Update the target chart version on the CAPI `Cluster`. The provider selects the chart name from the target version and reconciles the complete `AppRelease` source and specification.

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e
```

Do not patch the `AppRelease` source manually. Continue with Verify the Kube-OVN AppRelease.

vSphere v1.0.15 and earlier

> **WARNING**
>
> This legacy flow is valid only when the target Kube-OVN chart is earlier than `v4.4`. For chart `v4.4` or later, upgrade the vSphere provider to `v1.0.16` or later first. The legacy controller hardcodes the old chart name and can overwrite an unsupported manual source change.

For an earlier target chart, update the annotation only. The legacy provider reconciles the complete specification but keeps the legacy chart name.

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrit
e
```

Bare Metal

This procedure applies to Bare Metal provider `v1.0.0` and later. There is no earlier supported release flow to select.

Verify that the infrastructure cluster uses Kube-OVN, then update the target chart version on the CAPI `Cluster`:

```
kubectl get baremetalcluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.spec.networkType}{"\n"}'
# Expected output: kube-ovn

kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<kube-ovn-version-from-matrix> --overwrite
```

The provider selects the chart name from the target version and reconciles the complete `AppRelease` source and specification. Do not patch the `AppRelease` source manually.

Verify the Kube-OVN AppRelease

Run these commands against the cluster being upgraded:

```
kubectl get apprelease cni-kube-ovn -n cpaas-system \
  -o jsonpath='Chart: {.spec.source.charts[0].name}{"\n"}Target: {.spec.s
ource.charts[0].targetRevision}{"\n"}Installed: {.status.charts.*.install
edRevision}{"\n"}Phase: {.status.charts.*.phase}{"\n"}'

kubectl get apprelease cni-kube-ovn -n cpaas-system \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status} ({.reason})
{"\n"}{end}'
```

Before changing the control plane, verify all of the following:

- `spec.source.charts[0].name` matches the chart-name boundary above.
- `targetRevision` and `installedRevision` both match the OS Support Matrix row.
- `phase` is `Success`.
- The `Sync` and `Health` conditions are `True`.

The normal phase sequence is `Upgrading → HealthChecking → Success`. Do not continue based on `installedRevision` alone: it can change during `HealthChecking`, before Kube-OVN pods have been verified Ready. If the `AppRelease` reaches `DownloadFailed`, `DeployFailed`, or `NotReady`, stop and inspect its condition messages:

```
kubectl describe apprelease cni-kube-ovn -n cpaas-system
```

Restore Kube-OVN During Stage-1 Recovery

Use this recovery only when no control-plane `Machine` on the target Kubernetes minor has been created. If a target-minor control-plane machine has joined, keep the target Kube-OVN baseline and use forward recovery or the supported backup or DR restore procedure.

Restore the previous chart version on the CAPI `Cluster` in the management cluster:

```
kubectl annotate cluster <cluster-name> -n cpaas-system \
  cpaas.io/kube-ovn-version=<previous-kube-ovn-version> --overwrite
```

- **DCS `v1.0.22+`, HCS `v1.0.4+`, vSphere `v1.0.16+`, and Bare Metal `v1.0.0+`**: restore the previous `cpaas.io/kube-ovn-version` annotation. The provider restores the matching chart name, revision, and complete specification. Do not patch the source manually.
- **DCS `v1.0.21` and earlier or HCS `v1.0.3` and earlier, with a target earlier than chart `v4.4`**: after restoring the annotation, patch the existing `targetRevision` back to the previous chart revision on the cluster being recovered:

  ```
  kubectl patch apprelease cni-kube-ovn -n cpaas-system --type='json' \
    -p='[{"op":"replace","path":"/spec/source/charts/0/targetRevision","v
  alue":"<previous-kube-ovn-version>"}]'
  ```

- **vSphere `v1.0.15` and earlier, with a target earlier than chart `v4.4`**: restore the previous annotation only; the provider reconciles the complete legacy specification.

After restoring the previous target, repeat Verify the Kube-OVN AppRelease before changing any control-plane manifests.

Required kubelet patch for Kubernetes 1.35

When an upgrade hop targets Kubernetes 1.35 or later, the kubelet patch file at `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` must include the following setting:

```
{
  "imagePullCredentialsVerificationPolicy": "NeverVerify"
}
```

In Kubernetes 1.35, kubelet credential verification can apply even when an image is already present on the node. `NeverVerify` tells the kubelet not to verify pull credentials for these pre-pulled images. Add this field only to kubelet configurations that will run on Kubernetes 1.35 or later; do not add it for Kubernetes 1.34 or earlier.

Apply this change only on the hop to Kubernetes 1.35 or later:

- For control-plane nodes, update the matching file in `KubeadmControlPlane.spec.kubeadmConfigSpec.files` in the **same edit** that changes `spec.version` to 1.35. Do not roll out the policy separately while the target version is still 1.34.
- For worker nodes, create a new `KubeadmConfigTemplate` containing the updated file, then update `MachineDeployment.spec.template.spec.bootstrap.configRef.name` in the same edit as the target version and infrastructure template.
- If the file uses `contentFrom.secret`, verify that the referenced Secret key contains this parameter. An older version-specific Secret is not automatically suitable for the 1.35 hop.

For Kubernetes 1.34 or earlier, omit `imagePullCredentialsVerificationPolicy`. Add it only on the upgrade hop whose target is Kubernetes 1.35 or later.

See platform-specific guides:

- Huawei DCS
- Huawei Cloud Stack
- VMware vSphere
- Bare Metal

Preparing for Cross-Version Upgrades

When the target ACP Distribution Version is more than one Kubernetes minor above the cluster's current version, additional preparation is required before starting Phase 2. Single-minor upgrades skip this section.

On Alauda OS, each Kubernetes minor is baked into a matching OS image, and each Kubernetes minor maps to a specific ACP version row in OS Support Matrix. The Kubernetes version therefore moves forward one minor at a time through the required intermediate OS images. Keep these Kubernetes and OS hops separate from the ACP Core upgrade path: the Distribution Version must follow every source-to-target hop offered by CVO, while Phase 2 can require additional intermediate rows only to advance Kubernetes and the OS image.

Before starting Phase 2, identify every intermediate Kubernetes minor and its matching OS Support Matrix row. Confirm separately that Phase 1 completed every required Distribution Version hop according to Pre-Upgrade Preparation.

Step 1 — Confirm Core hops and pre-sync intermediate images

If CVO offers an intermediate Distribution Version as part of the supported Core path, perform the full ACP artifact preparation and Core/CVO upgrade for that hop during Phase 1. This is required for paths such as ACP 4.0.x to 4.4, and `--only-sync-image` does not replace it. Verify `availableUpdates` and `VersionUpgradePath` at every Core hop.

After all required Core hops are complete, some intermediate OS Support Matrix rows may still be needed only for the sequential Kubernetes and OS rollout in Phase 2. For each of these Kubernetes/OS-only rows, download the corresponding Core package and synchronize only its images into the registry. Do not request an additional Core upgrade solely because a row is used as a Kubernetes or OS stepping stone.

```
bash upgrade.sh --only-sync-image
```

See Sync upgrade artifacts for command options.

This places the intermediate-version CoreDNS, etcd, and Kube-OVN chart images in the registry so the staged Kubernetes rollout can pull them when `KubeadmControlPlane` is patched to the matching OS Support Matrix row. In an air-gapped environment, missing intermediate images prevent the new control plane components from starting.

For a required Core hop, prepare and upgrade Aligned plugins as directed by the Phase 1 procedure. For a Kubernetes/OS-only intermediate row, Aligned plugins do not need an extra `violet push`; the Kube-OVN chart used by that intermediate Kubernetes hop is covered by the Core image sync above.

Step 2 — Stage intermediate OS images for the provisioning mechanism

For each intermediate Kubernetes/OS row, make the Alauda OS image published for that ACP release available to the node provisioning mechanism so the Kubernetes step can replace nodes from each image in turn:

- **IaaS providers (Huawei DCS, VMware vSphere, Huawei Cloud Stack)**: upload the OS image and register it as a VM template (or VM image, depending on the platform). Record the name it is registered under — the machine templates reference that name.
- **Bare metal**: make the OS image available to the node re-provisioning flow used by your cluster.

The control plane and worker rollouts then step through the intermediate OS images one at a time, matched to the staged Kubernetes minor, until the cluster reaches the target ACP version row.

Step 3 — Apply the platform-specific procedure for each intermediate Kubernetes/OS row

Repeat the platform-specific Phase 2 procedure for each intermediate Kubernetes/OS row in turn, using that row in OS Support Matrix for Kube-OVN, `KubeadmControlPlane`, and `MachineDeployment` values. For each hop, upgrade Kube-OVN first and wait for `phase=Success`, then upgrade the control plane, followed by workers. On the hop to Kubernetes 1.35, also apply the required kubelet patch to the control-plane and worker bootstrap configurations. Complete the whole hop before starting the next one. After all intermediate hops succeed, apply the same procedure once more for the target row.

Platform-Specific Instructions

Select your platform for detailed upgrade instructions:

Bare Metal

Key Considerations

Huawei DCS

Upgrade Sequence

Huawei Clo

Overview

Upgrade Sequence
Storage Compatibility Across Provider Upg
Prerequisites
Check COS_STATE Free Space
Update the Image Catalog
Upgrade Kube-OVN Before the Control Plane
Upgrade the Control Plane
Upgrade Workers
Cross-Version Upgrades
Recovering From a Failed Phase 2 Upgra
Verification
Troubleshooting
Additional Resources

Prerequisites
Using YAML
Fleet Essentials Boundary
Additional Resources

Upgrade Seque
Prerequisites
Control Plane U
Worker Node U
Recovering Fro
Additional Resc

VMware vSphere

Upgrade Sequence
Prerequisites
Required Values From the OS Support Ma
Steps
OS Image Update Without a Kubernetes V
Rollout Holds That Are Not Failures
Recovering From a Failed Phase 2 Upgra
Verification
Troubleshooting
Next Steps

Migrate Exis

Overview
Before You Star
Inspect the Cur
Determine Whic
Update the DC
Update the DC
Trigger the Roll
n, De
and
pics

Migrate Existing HCS Clusters to Pool-Managed Persistent Disks

Overview
Before You Start
Inspect the Current Disk Layout
Decide Which Disks to Preserve
Create New Machine Templates
Prepare the Rollout Strategy
Update the Machine Configuration Pool
Trigger Rolling Replacement
Verify the Migration
Failure Handling
Limitations and Recovery Notes
Related Topics

Related Topics

- Create a Cluster on Alauda OS
- Manage Nodes on Alauda OS Clusters
- Infrastructure Resources

Upgrade Kubernetes on Bare Metal

This guide explains how to complete Phase 2 of the upgrade workflow for clusters on bare metal. Before you upgrade Kubernetes, complete the Distribution Version upgrade described in Upgrade the Distribution Version.

INFO

Where this page fits in the full ACP upgrade flow

This page covers only the Kubernetes step of the upgrade. The full ACP upgrade flow — including upgrade artifact synchronization, ACP Core upgrade through CVO, Aligned plugin upgrades, and Agnostic plugin upgrades from Marketplace — is documented in the following pages. Complete those steps before you start the Kubernetes step on this page:

- Upgrade Overview (scope and sequencing)
- Pre-Upgrade Preparation
- Upgrade the Distribution Version of the `global` cluster (Core, Aligned, Agnostic)
- Upgrade the Distribution Version of workload clusters (Core, Aligned, Agnostic)

Use this page when the cluster runs Alauda OS on bare-metal hosts, because the Kubernetes step on bare-metal replaces every node from a new `elemental upgrade` image rather than upgrading binaries in place.

TOC

Key Considerations

Upgrade Sequence
Storage Compatibility Across Provider Upgrades
Prerequisites
Check COS_STATE Free Space
Remove partial snapshots left by a failed upgrade
Update the Image Catalog
Upgrade Kube-OVN Before the Control Plane
Upgrade the Control Plane
Upgrade Workers
Cross-Version Upgrades
Recovering From a Failed Phase 2 Upgrade
Verification
Troubleshooting
Additional Resources

Key Considerations

Bare-metal upgrades replace every node — they do not run `kubeadm upgrade` on the existing OS. The mechanism is:

1. CAPI deletes one `Machine` according to the rollout strategy.
2. The provider writes a `clean` plan to the inventory's plan secret.
3. The host stops kubelet, deactivates any managed data mounts, clears CRI workload, stops containerd, then returns to `Available` in the same pool.
4. CAPI creates a replacement `Machine`.
5. The provider picks an `Available` inventory from the same pool, resolves the new Kubernetes version against `elemental-image-catalog`, and writes a `reprovision` plan.
6. The host stages managed-volume activation, runs `cloud-init clean` → `elemental upgrade --system <new-image>` → reboot. `initramfs` clears Kubernetes persistent

state; managed mount units start before kubelet, and cloud-init re-executes and runs `kubeadm init` / `kubeadm join` against the new control plane.

Two structural consequences operators must internalize before starting:

- **Kubernetes-managed system state is not preserved.** `/var/lib/kubelet`, `/var/lib/containerd`, `/var/lib/etcd`, and `/etc/kubernetes` are cleared by the initramfs cleanup step of every reprovision. Storage v2 does not permit these protected paths as managed data mounts.
- **Declared data volumes can remain on an inventory.** The clean plan deactivates them without wiping their filesystems, and reprovision activates the selected inventory's prepared volumes before kubelet. This is host-local retention, not a backup or data-migration mechanism.
- **The same `MachineInventory` may not be re-picked.** The provider does not guarantee that the inventory released by a `clean` plan is the same one re-allocated to the replacement `Machine`; data does not follow the replacement automatically. Replacements are performed as delete-then-add when `maxSurge=0` (no overlap between old and new node), so pool capacity sizing need only accommodate the desired replica count — old and new nodes do not coexist during the rollout.

Upgrade Sequence

Upgrade bare-metal clusters in the following order:

1. **(Prerequisite)** Upgrade the ACP platform on the `global` cluster. This brings the bare-metal provider, `elemental-operator`, and the related CAPI components to versions that understand the new schema. Trigger workload-cluster upgrades only after the management-side controllers have rolled out and become Ready.
2. Upgrade the Distribution Version (Aligned Extensions) on the workload cluster. See Upgrade the Distribution Version.
3. Import the target Alauda OS image pair into the platform registry and add the target Kubernetes version to `elemental-image-catalog`. See Update the Image Catalog.
4. Upgrade Kube-OVN to the chart version required by the target ACP release and wait for the `AppRelease` to reach `Success`.

5. Upgrade the control plane Kubernetes version (replaces all control-plane nodes one at a time).
6. Upgrade worker nodes to the target Kubernetes version (replaces all worker nodes within the `maxUnavailable` budget).

Cluster API orchestrates the rolling replacement.

> **WARNING**
>
> Skipping step 1 risks two failure modes: the old provider silently ignores new schema fields; or a controller image swap mid-rollout interrupts the plan secret state machine. Always settle the management-side upgrade before touching workload rollout.

Storage Compatibility Across Provider Upgrades

An existing cluster created before managed storage was introduced remains compatible when `MachineInventory.spec.storage` stays absent or has `volumes: []`. In this unmanaged mode, the storage controller performs no disk operation and does not rewrite the running Machine's plan merely because the management-side provider was upgraded.

This compatibility does **not** mean that managed storage can be enabled in place on every old host:

- Upgrading only the provider and `elemental-operator` on the `global` cluster does not install `elemental-storage-observer.service` or early-boot storage integration into an older host OS.
- Keep `spec.storage` empty on such hosts. To enable managed storage later, reinstall, reset, or replace the host with an approved observer-capable image, then verify the observer service and a fresh `status.observedStorage` report while the inventory is unallocated.
- Configure and Prepare managed volumes only after those host-image prerequisites are satisfied. Do not add a non-empty storage declaration to a running or allocated inventory.

For the preparation and validation procedure, see Manage Data Disks on Bare-Metal Hosts.

Prerequisites

Before you start, ensure all of the following prerequisites are met:

- The Distribution Version upgrade is complete.
- The installed Bare Metal provider `AppRelease` reports the same requested and installed revision with `phase=Success`.
- The control plane is reachable through the existing `<control-plane-vip>:<control-plane-port>`.
- All current nodes are healthy and `Ready`.
- Before changing Kube-OVN or reprovisioning any node, use ACP Cluster Enhancement to create a current on-demand etcd backup. Verify that the newest `status.records` entry reports `result: Success`, record its `backupTimestamp` and `fileName`, and confirm that the corresponding backup archive exists and will remain retrievable if any control-plane node is reprovisioned or becomes unavailable.
- The target Kubernetes version is a key in `elemental-image-catalog`. If it is not, add it before starting (see Update the Image Catalog).
- The target **kube-ovn (chart)** version has been read from the matching ACP row in the OS Support Matrix.
- `BaremetalCluster.spec.networkType` is `kube-ovn`; the provider skips Kube-OVN reconciliation for any other value.
- The platform registry is reachable from every host in the cluster.
- Every host the rollout may reprovision has at least 4 GiB free on its `state` partition, and carries no partial snapshot from an earlier failed upgrade (see Check COS_STATE Free Space).
- Both `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge = 0` and `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0` are set — bare-metal does not over-provision physical hosts.
- The relevant pools have enough capacity to replace one node at a time without falling below the desired replica count.
- Every inventory with a non-empty storage declaration reports `StoragePrepared=True`, and every currently allocated inventory reports `StorageActive=True`. Record each `BaremetalMachine.status.machineInventoryRef` and the managed filesystem UUIDs

before the rollout so that you can detect selection of a different inventory instead of assuming data moved with the Machine.

INFO

The default local archive under `/var/cpaas/backup` can be used for recovery; S3-compatible storage is not required. However, bare-metal replacement can make the original node-local copy unavailable, and the replacement `Machine` is not guaranteed to use the same `MachineInventory`. If your recovery design cannot guarantee access to the local archive after node replacement or failure, keep an off-node copy in S3-compatible storage or another secure location outside the node before starting Phase 2.

Check COS_STATE Free Space

Every reprovision creates a new snapshot on `COS_STATE` and writes the complete target system into it. Because the upgrade targets a *different* image, the partition has to hold two unrelated systems of roughly 3.5 GiB at the same time. A host that is short on space fails part-way through with `no space left on device`, and the failed attempt leaves its partial snapshot behind — so each automatic retry starts with *less* free space than the one before it. Three retries on an 8 GiB partition is enough to fill it completely.

Check this before starting the rollout. Check every host in the affected pools, not only the ones you expect to be replaced: the inventory released by a `clean` plan is not guaranteed to be re-picked, so any `Available` inventory can receive the new image.

On each host:

```
# Free space on COS_STATE. Use btrfs, not df — df does not account for sn
apshot extents.
btrfs filesystem usage -h /

# Retained snapshots
snapper --no-dbus list
```

`Free (estimated)` must be at least **4 GiB** — the target system is roughly 3.5 GiB, plus headroom for btrfs metadata written during the copy. If it is lower, free space before starting the rollout.

Remove partial snapshots left by a failed upgrade

Elemental marks a snapshot `update-in-progress=yes` while it writes into it, and clears the flag only when the upgrade completes. A snapshot that still carries the flag, and is neither the default nor the running one, is the residue of a failed attempt and is safe to delete.

List the snapshots with the three columns that decide this:

```
snapper --no-dbus --csvout list --columns number,default,active,userdata
```

Delete a snapshot only when all three conditions hold — `default` is `no`, `active` is `no`, and the userdata column contains `update-in-progress=yes`:

```
snapper --no-dbus delete --sync <number>
```

> **WARNING**
>
> Never delete a snapshot whose `default` or `active` column is `yes`. Those are the snapshot the host will boot next and the one it is running now; deleting either leaves the host unable to boot.

Re-run `btrfs filesystem usage -h /` afterwards to confirm the space was actually reclaimed. On a partition that is already 100% full, `snapper delete` can itself fail for lack of metadata space — this is the same condition that let the partial snapshots accumulate in the first place. If deletion fails, treat the host as needing a reinstall rather than retrying the rollout against it.

If the host still has less than 4 GiB free after every partial snapshot is gone, the partition itself is too small. Hosts installed before COS_STATE sizing was applied carry an 8 GiB `COS_STATE`, and partition sizes are fixed at install time — growing it requires a reinstall. Plan that separately; do not start the rollout against such a host and expect it to succeed.

Update the Image Catalog

`elemental-image-catalog` is the resource that introduces a Kubernetes version to the bare-metal provider.

> **INFO**
>
> The bare-metal provider plugin does not carry any Kubernetes version or OS image. Its chart creates `elemental-image-catalog` with no entries, and upgrading the plugin or the Distribution Version does not add one. Every new Kubernetes version must be added explicitly, after its OS images are in the platform registry.

First import the target release's `baremetal-base-image` and `baremetal-base-image-iso` from the Bare Metal OS archive into the platform registry, as described in OS Images Imported and Image Catalog Populated. Then add the version with either option below.

Option A — Add a chart override. Append the new version under `provider.imageCatalog.images` in `spec.valuesOverride` of the provider's `ModuleInfo` on the `global` cluster (uses `global.registry.address` as the registry), next to the values already set there. Do not replace the whole `valuesOverride`: that drops `caCertSecretKey`, and `elemental-operator` cannot start. To find and edit the `ModuleInfo`, see Install the cluster plugins:

```
spec:
  valuesOverride:
    acp/chart-cluster-api-provider-baremetal:
      provider:
        imageCatalog:
          images:
            <new-kubernetes-version>:
              repository: tkestack/baremetal-base-image
              tag: <official-image-tag>
```

The platform re-renders the ConfigMap; the provider's in-process watch hot-reloads the cache without restarting. In a DR pair, do this on both `global` clusters.

Option B — Patch the ConfigMap directly. Use this when the plugin was installed without a chart override, or for an approved image reference that must be pinned directly in the catalog:

```
kubectl -n cpaas-system patch configmap elemental-image-catalog \
  --type='json' \
  -p='[{"op":"add","path":"/data/<new-kubernetes-version>","value":"<regi
stry-address>/tkestack/baremetal-base-image:<new-tag>"}]'
```

In either case, verify before continuing:

```
kubectl -n cpaas-system get configmap elemental-image-catalog -o yaml
```

The key must include the leading `v` (for example `v1.34.5`). If the key is missing when CAPI creates the replacement `Machine`, the resulting `BaremetalMachine` enters `Failed` with `ImageResolved=False / Reason=ImageCatalogMiss` and no `reprovision` plan is written. Adding the key later does **not** restart it: the `BaremetalMachine` stays `Failed` until you delete that `Machine` and its owner creates a new one. With `maxSurge: 0` the node being replaced is already gone at that point, so add the key before you change any version.

The bare-metal provider uses `base-image` for reprovisioning and does not rebuild SeedImages automatically. Before future host onboarding, refresh `MachineRegistration` / `SeedImage` with the `base-image-iso` imported from the same archive. Do not pair images from different archives or substitute an independently built image.

Upgrade Kube-OVN Before the Control Plane

Follow the Bare Metal flow in Upgrade Kube-OVN Before the Control Plane.

The shared procedure verifies `BaremetalCluster.spec.networkType`, updates the CAPI `Cluster` annotation, lets the provider reconcile the complete chart source and specification, and checks the chart name, requested revision, installed revision, phase, and conditions. Do not patch the `AppRelease` source manually.

Upgrade the Control Plane

Before continuing, complete the shared Upgrade Kube-OVN Before the Control Plane procedure and verify that `cni-kube-ovn` is at the target installed revision with `phase=Success`.

Patch `KubeadmControlPlane.spec.version` to the new Kubernetes version. Where component image tags are pinned (DNS, etcd), update them in the same edit:

```
kubectl -n cpaas-system edit kubeadmcontrolplane <cluster-name>-control-p
lane
```

- `spec.version` ← target Kubernetes version (must match an `elemental-image-catalog` key).
- `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageTag` ← matching CoreDNS image tag for the new release.
- `spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.imageTag` ← matching etcd image tag for the new release.
- When the target is Kubernetes 1.35 or later, update `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` in `spec.kubeadmConfigSpec.files` in this same edit, as described in Required kubelet patch for Kubernetes 1.35.

The bare-metal provider does **not** require a new `BaremetalMachineTemplate` for a Kubernetes-only upgrade: the template only carries the pool reference, and the upgrade image is resolved from `Machine.spec.version` through the catalog. A new template is only required when you want to move the control plane to a different pool.

Cluster API rolls control-plane nodes one at a time (because `maxSurge=0`):

```
kubectl -n cpaas-system get kubeadmcontrolplane <cluster-name>-control-pl
ane -w
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.
x-k8s.io <cluster-name>-control-plane-pool
kubectl get nodes -o wide
# workload cluster
```

The expected sequence on each node:

1. CAPI marks one old `Machine` for deletion; corresponding `BaremetalMachine` moves to `Preparing`.
2. The provider writes a `clean` plan; managed volumes are deactivated without being wiped, and the inventory ends up `Available` with cleared owner annotations and storage returned to `Prepared`.
3. CAPI creates a new `Machine` (with the target version); a new `BaremetalMachine` is created.
4. The provider allocates an `Available` inventory from the control-plane pool, resolves the new image, and writes a `reprovision` plan.
5. The host stages activation for that selected inventory's prepared volumes, runs `elemental upgrade --system <new-image>`, reboots, starts required mount units before kubelet, and `kubeadm join`s the surviving control plane. `BaremetalMachine.status.phase` becomes `Running` only after required storage is active.

Repeat steps 1–5 until every control-plane node has been replaced.

Throughout the rollout, `alive` continues to manage the VIP. As control-plane membership changes, the bare-metal provider re-renders the `alive` chart values (the `peer` list and `ipvs.ips`) and rolls out the static-pod manifests.

Upgrade Workers

Patch each `MachineDeployment.spec.template.spec.version` to the new Kubernetes version. CAPI replaces workers within the `maxUnavailable` budget — the bare-metal provider re-uses the same worker pool and the same image-catalog resolution logic as the control plane.

For Kubernetes 1.35 or later, first create a new worker `KubeadmConfigTemplate` containing the required kubelet patch, then update `spec.template.spec.bootstrap.configRef.name` in the same `MachineDeployment` edit as the version bump. The command below includes that reference. For an earlier target with no bootstrap change, omit the `bootstrap` object. Follow Updating Bootstrap Templates to create the new template.

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers \
  --type='merge' \
  -p='{"spec":{"template":{"spec":{"version":"<new-kubernetes-version
>","bootstrap":{"configRef":{"name":"<new-bootstrap-template>"}}}}}}'
```

Watch:

```
kubectl -n cpaas-system get machinedeployments.cluster.x-k8s.io -w
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io
kubectl get nodes -o wide
```

Worker upgrades carry fewer knobs than the control plane: there is no `clusterConfiguration` block on `MachineDeployment`, so no DNS / etcd tags to update. Kubernetes component versions on worker nodes follow the new base image.

For earlier versions, a bootstrap-template change is needed only when the release changes settings such as cloud-init or `kubeletExtraArgs`.

Cross-Version Upgrades

Kubernetes control planes must move through supported minor-version hops. Use Preparing for Cross-Version Upgrades to identify and pre-stage every intermediate ACP row.

For each intermediate row, and finally for the target row:

1. Import that row's OS image pair and add its Kubernetes version to `elemental-image-catalog`, as described in Update the Image Catalog.
2. Upgrade Kube-OVN to that row's **kube-ovn (chart)** version and wait for `phase=Success`.
3. Upgrade the control plane to that row's Kubernetes version and wait for every control-plane node to become `Ready`.
4. Upgrade every worker `MachineDeployment` to the same Kubernetes version and wait for the rollout to complete before starting the next hop.

The version values always come from the OS Support Matrix; they are not fixed examples in this guide. The bare-metal rollout strategy already serializes node replacement, so cross-version upgrades do not require additional pool capacity beyond what is needed for a same-minor upgrade.

Recovering From a Failed Phase 2 Upgrade

Do not treat a Kubernetes minor downgrade as an ordinary rollback:

1. If no target-version control-plane `Machine` has been created, restore Kube-OVN by using Restore Kube-OVN During Stage-1 Recovery, then restore the previous `KubeadmControlPlane` and `MachineDeployment` manifest values.
2. If a target-minor control-plane machine has joined, stop further rollout and repair forward on the target minor. If forward repair cannot restore cluster health, restore the cluster from the verified pre-upgrade backup by following the ACP etcd restore procedure. Do not patch Kubernetes, CoreDNS, or etcd back to the previous minor.

Restoring etcd is a disaster-recovery action, not a pre-upgrade validation step. Do not run the restore procedure on a healthy cluster merely to verify the backup.

Every host that completed a `reprovision` plan has already replaced its OS image and cleared Kubernetes-managed state. Changing version fields does not reconstruct that earlier node state.

Verification

After the rollout completes:

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io \
  -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o custom-columns='NAME:.metadata.name,PHASE:.status.phase,INVENTORY:.s
tatus.machineInventoryRef.name,PLAN_SECRET:.status.planSecretRef.name'
kubectl -n cpaas-system get machines.cluster.x-k8s.io \
  -l cluster.x-k8s.io/cluster-name=<cluster-name>
kubectl -n cpaas-system get apprelease cni-kube-ovn
kubectl get nodes -o wide
```

The `cni-kube-ovn` `AppRelease` should report the target installed revision with `phase=Success`. For the target cluster, every current `BaremetalMachine` should be `Running`, every CAPI `Machine` should report the new `spec.version`, and every workload `Node` should be `Ready` with the new `kubelet` version.

For each current `BaremetalMachine`, use the `INVENTORY` and `PLAN_SECRET` columns to verify the inventory and plan that back the running node:

```
kubectl -n cpaas-system get machineinventories.elemental.cattle.io <inven
tory-name> \
  -o jsonpath='{.status.plan.state}{" storage="}{.status.storage.phase}
{"\n"}'
kubectl -n cpaas-system get secret <plan-secret-name> \
  -o jsonpath='{.metadata.annotations.baremetal\.alauda\.io/plan\.type}
{"\n"}'
```

Each referenced `MachineInventory` should report `Applied`, and each referenced plan Secret should report `reprovision`. Apply this check only to inventories referenced by the current `BaremetalMachine` objects. A spare inventory that did not participate in the rollout may have no `reprovision` plan. An inventory that completed a `clean` plan but was not selected for a replacement Machine may still report `clean`; neither case means that the rollout is incomplete.

For an inventory with managed volumes, also verify `status.storage.phase=Active`, `StoragePrepared=True`, `StorageActive=True`, and `BaremetalMachine.status.conditions[StorageReady]=True/AllVolumesReady`. Compare its current filesystem UUIDs with the pre-upgrade record. A different UUID usually

means a different inventory was selected; it must not be interpreted as the original data having migrated.

`MachineInventoryPool.status` should satisfy `available + allocated + preparing + reprovisioning + unavailable = total` with `preparing = reprovisioning = 0`. `available` may remain greater than zero when the pool contains spare inventories; those inventories are outside the plan verification above.

Troubleshooting

Issue	What to check
The Kube-OVN annotation changed but the chart name, target revision, or installed revision did not converge	Inspect the `AppRelease` condition Follow the shared Kube-OVN verifi source manually.
Rollout stuck on the first new node	`BaremetalMachine.status.cond` — confirm the new key is in `eleme` and that the `Cluster` carries `cp address`.
Inventory does not become allocatable after clean	Check `MachineInventory.status` `status.observedStorage`. Mana confirmed inactive and Prepared fr observation before the inventory ca set.
Replacement Machine waits on `StorageReady`	Verify that the selected inventory is required device IDs and filesystem required mount units are active, an fresh post-reboot observation.
Host reboots into a state that never joins the cluster	`MachineInventory.status.plan` terminal); inspect the host's serial c `upgrade` errors. Verify the platform from the host.
`elemental upgrade` fails with `no space left on device`	The `state` partition is too small - cannot hold the running system an different image at the same time. E

Issue	What to check
	leaves a partial snapshot behind, s shrinking. Recover by removing the Remove partial snapshots left by a the plan retry. Hosts installed befor was applied need a reinstall to grov
New control-plane node fails `kubeadm join`	Confirm the VIP is still reachable; lock. Check `kubectl -n kube-sy` `ipvsadm -Ln`, `keepalived` logs
Worker rollout stalls with `MachineDeployment.spec.strategy.rollingUpdate.maxSurge > 0` set	Bare-metal pools cannot honour ov `maxSurge=0` and bump `maxUnav` parallel replacements within the po
Pool capacity exhausted mid-rollout	`MachineInventoryPool.status.` `allocated` still includes the not-y another `MachineInventory` to th or wait for in-flight `Preparing` no

For the full operator-side state machine reference (every condition reason and recovery action), see Provider Overview.

Additional Resources

- Cluster API Upgrade Documentation ↗
- Kubernetes Version Skew Policy ↗

Upgrade Kubernetes on Huawei DCS

This guide explains how to complete Phase 2 of the upgrade workflow for clusters on Huawei DCS. Before you upgrade Kubernetes, complete the Distribution Version upgrade described in Upgrade the Distribution Version.

INFO

Where this page fits in the full ACP upgrade flow

This page covers only the Kubernetes step of the upgrade. The full ACP upgrade flow — including upgrade artifact synchronization, ACP Core upgrade through CVO, Aligned plugin upgrades, and Agnostic plugin upgrades from Marketplace — is documented in the following pages. Complete those steps before you start the Kubernetes step on this page:

- Upgrade Overview (scope and sequencing)
- Pre-Upgrade Preparation
- Upgrade the Distribution Version of the `global` cluster (Core, Aligned, Agnostic)
- Upgrade the Distribution Version of workload clusters (Core, Aligned, Agnostic)

Use this page when the same cluster runs on Alauda OS, because the Kubernetes step there replaces nodes from a new Alauda OS-based VM template rather than upgrading binaries in place.

INFO

Alauda OS Image and DCS Provider Pairing

Select the Alauda OS image published for the ACP release installed on the cluster. See Downloading Alauda OS. The DCS provider version, the Alauda OS image version, and the ACP version must match.

If the VM templates of an existing cluster were made from an Alauda OS image published for a release earlier than ACP `v4.3.2`, they do not match DCS Provider `v1.0.21` or later. When you upgrade the `global` cluster, upgrade the provider together with its Distribution Version, because the platform upgrades the installed Aligned plugins during that upgrade; see Pre-Upgrade Preparation. Then create the VM template from the Alauda OS image published for the target ACP release and replace the nodes from it by following this page. See Required Values From the OS Support Matrix.

INFO

Existing Cluster Migration

If your cluster runs ACP `v4.2.1` or later and still keeps preserved data in the old template-disk layout, complete the migration procedure in Migrate Existing Huawei DCS Clusters to Pool-Managed Persistent Disks before you rely on upgrade-time disk preservation.

TOC

- Upgrade Sequence
- Prerequisites
- Using YAML
  - Required Values From the OS Support Matrix
  - Upgrade Kube-OVN Before the Control Plane
  - Upgrade Control Plane Infrastructure
    - Procedure
  - Upgrade Control Plane Kubernetes Version
    - Procedure
  - Upgrade Worker Nodes
    - Procedure
  - Recovering From a Failed Phase 2 Upgrade
- Fleet Essentials Boundary

Additional Resources

Upgrade Sequence

Upgrade DCS clusters in the following order:

1. **(Prerequisite)** Upgrade the ACP platform on the management cluster first. This brings the cluster-api-provider-dcs controller and the related CAPI components (core, KubeadmControlPlane provider, bootstrap provider) to versions that understand the new schema. Trigger workload-cluster upgrades only after the management-side controllers have rolled out and become Ready.
2. Upgrade the Distribution Version (Aligned Extensions) on the workload cluster. See Upgrade the Distribution Version.
3. Upgrade Kube-OVN to the chart version required by the target ACP release and wait for the `AppRelease` to reach `Success`.
4. Upgrade the control plane Kubernetes version.
5. Upgrade worker nodes to the target Kubernetes version.

Cluster API orchestrates rolling updates with built-in safety mechanisms to reduce service disruption.

> **WARNING**
>
> Skipping step 1 risks two failure modes: the old controller silently ignores new schema fields written to `DCSIpHostnamePool` / `DCSMachineTemplate`; or a controller image swap mid-rollout interrupts persistent-disk state-machine progression. Always settle the management-side upgrade before touching workload rollout.

Prerequisites

Before you start, ensure all of the following prerequisites are met:

- The Distribution Version upgrade is complete
- The control plane is reachable
- All nodes are healthy and in Ready state
- A current etcd backup has been taken and verified by using the supported ACP backup procedure
- The IP Pool has sufficient capacity for rolling updates
- The VM template supports the target Kubernetes version. See OS Support Matrix for version mapping
- For cross-version upgrades that span more than one Kubernetes minor, the intermediate-version Core images and VM templates are pre-staged. See Cross-Version Upgrade Preparation
- The target Kubernetes version is compatible with your workloads and add-ons
- DCS VM templates are `4.2.1` or later if you use pool-managed persistent disks, because safe shutdown and disk detach depend on guest tools
- If you rely on pool-managed persistent disks, keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge = 0` and each `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`
- If the cluster uses additional NICs, keep the required `DCSIpHostnamePool.spec.pool[].additionNic[]` entries in place and confirm that the DCS DVS and Port Groups are reachable from the hosts that may receive replacement VMs

WARNING

Disk Preservation Model

Upgrades rely on Cluster API's rolling update mechanism. Each cluster has four disk classes; only the pool-managed class survives a delete-recreate.

Disk class	Declared in
System disk (root volume)	The VM template image used for `vmTemplateName`
Template-local disks	`DCSMachineTemplate.spec.template.spec.vmConfig.dcsMachineDiskSp`
Pool-managed persistent disks	`DCSIpHostnamePool.spec.pool[].persistentDisk`
External CSI volumes (cinder, etc.)	Workload PVCs / CSI driver

"Preserved" means the same disk identity is reattached — it does not mean the disk's contents are time-traveled. Anything written to a pool-managed disk during the upgrade window stays after the upgrade and stays after a rollback.

Pool-managed preservation requires one-by-one replacement, so keep `maxSurge = 0` on both `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate` and `MachineDeployment.spec.strategy.rollingUpdate`.

If your existing cluster still keeps preserved data in the old template-disk layout, migrate it first by following Migrate Existing Huawei DCS Clusters to Pool-Managed Persistent Disks.

INFO

Additional NICs During Rolling Replacement

Additional NICs are declared in `DCSIpHostnamePool.spec.pool[].additionNic[]`, not in `DCSMachineTemplate`. During an upgrade, each replacement VM uses the additional NICs from

the Pool slot it claims. The upgrade template steps below should not move additional NIC settings into `DCSMachineTemplate`.

INFO

Use a New Machine Template for Each Upgrade Step

Every upgrade step on this page creates a new `DCSMachineTemplate` with a new `metadata.name`, applies it, and then patches the `infrastructureRef.name` of the `KubeadmControlPlane` or `MachineDeployment` to the new template. Keep the previous template until the new rollout is healthy in case rollback is required.

Using YAML

YAML-based upgrades do not depend on Fleet Essentials.

Required Values From the OS Support Matrix

The authoritative mapping between an ACP release, the Kubernetes version, and the matching CoreDNS, etcd, and Kube-OVN versions lives in OS Support Matrix. Locate the row that corresponds to the target ACP version before you start; the row supplies the component values the YAML steps below need.

The VM template name is not in the matrix. It is the name the template was created under on DCS from the Alauda OS image published for the target ACP release — see Downloading Alauda OS — and it is what `DCSMachineTemplate.spec.template.spec.vmTemplateName` must be set to on both the control plane and worker templates.

The cells you read from that row map to the upgrade manifests as follows:

OS Support Matrix column	Used to set
Kubernetes Version	`KubeadmControlPlane.spec.version` and `MachineDeployment.spec.template.`

OS Support Matrix column	Used to set
coredns	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.dns.im`
etcd	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.etcd.l`
kube-ovn (chart)	`Cluster.metadata.annotations["cpaas.io/kube-ovn-version"]` and the provic `cni-kube-ovn` `AppRelease`

The CoreDNS and etcd image tags are control-plane-only because `clusterConfiguration` is a `KubeadmControlPlane` field. Worker nodes inherit container image versions from the new VM template; the `MachineDeployment` does not carry its own `dns` / `etcd` tags. The Kube-OVN annotation lives on the `Cluster` resource, not on `KubeadmControlPlane`, because the DCS provider watches it independently of the Kubernetes control plane rollout.

Confirm with the cluster's platform owner that the Alauda OS image published for the target ACP release has already been uploaded to the DCS platform, and record the VM template name it was created under. The upgrade fails if that VM template is not present on DCS when the `DCSMachineTemplate` is applied.

Upgrade Kube-OVN Before the Control Plane

Follow Upgrade Kube-OVN Before the Control Plane and select the Huawei DCS procedure that matches the **installed** DCS provider version. That shared procedure owns the provider version boundary, the Kube-OVN chart-name migration at `v4.4`, legacy behavior, and the required `AppRelease` health checks.

Do not use the legacy direct `targetRevision` patch for a Kube-OVN `v4.4+` target. Upgrade the DCS provider to `v1.0.22` or later first so it can reconcile the complete source

and the related CoreDNS and kube-proxy repository changes. This provider upgrade does not migrate an existing external control-plane endpoint to Self-built VIP; endpoint mode remains unchanged during a routine Kubernetes upgrade.

Upgrade Control Plane Infrastructure

Upgrading the control plane machine template lets you roll out updated VM specifications, system patches, and infrastructure settings.

Procedure

1. **Create an updated machine template**
   Copy the existing `DCSMachineTemplate` referenced by `KubeadmControlPlane` and save it as a new file:

   ```
   kubectl get dcsmachinetemplate <current-template-name> -n cpaas-system
   -o yaml > new-cp-template.yaml
   ```

2. **Modify the template specifications**
   Update the new template as needed:

   - Set `metadata.name` to `<new-template-name>`
   - Update `spec.template.spec.vmTemplateName`
   - Update `spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`
   - Update `spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`
   - Update `spec.template.spec.vmConfig.dcsMachineDiskSpec` for system and template-local disks only
   - Strip server-generated metadata (`resourceVersion`, `uid`, `generation`, `creationTimestamp`, `managedFields`, `kubectl.kubernetes.io/last-applied-configuration` annotation) and the entire `status` field from the copied manifest.
   - Leave `spec.template.spec.providerID` unset. The DCS provider sets `providerID` to `dcs://<machine-name>` once the VM is created; pre-filling it in the template breaks the controller's identity binding.

Keep pool-managed persistent disks, including `/var/cpaas`, in `DCSIpHostnamePool.spec.pool[].persistentDisk`. Keep additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]`.

3. **Apply the updated template**

```
kubectl apply -f new-cp-template.yaml -n cpaas-system
```

4. **Update the control plane reference**

   Modify the `KubeadmControlPlane` resource to reference the new template:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='merge' -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<new-template-name>"}}}}'
```

5. **Monitor the rolling update**

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/control-plane
```

Upgrade Control Plane Kubernetes Version

Upgrading the control plane Kubernetes version on Alauda OS is a delete-recreate workflow. The control plane VMs are replaced one by one from a new `DCSMachineTemplate` that points at the target Alauda OS VM template, and the `KubeadmControlPlane` resource is patched to carry the matching Kubernetes version, CoreDNS image tag, and etcd image tag.

Before you start, complete the shared Upgrade Kube-OVN Before the Control Plane procedure and verify that the `cni-kube-ovn` `AppRelease` is at the target revision with `phase=Success`. Then collect every required control-plane value from the target ACP row in the OS Support Matrix as described in Required Values From the OS Support Matrix.

Procedure

1. **Create a new `DCSMachineTemplate` for the target Kubernetes version**

   Copy the existing control-plane template and update `metadata.name` to a new name and `spec.template.spec.vmTemplateName` to the VM template created on DCS from the

Alauda OS image published for the target ACP release. Keep pool-managed persistent disks in `DCSIpHostnamePool.spec.pool[].persistentDisk` and additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]` rather than reintroducing them as template fields.

```
kubectl get dcsmachinetemplate <current-cp-template-name> -n cpaas-syst
em -o yaml > new-cp-template.yaml
# edit metadata.name and spec.template.spec.vmTemplateName
kubectl apply -f new-cp-template.yaml -n cpaas-system
```

2. **Patch the `KubeadmControlPlane` with the target Kubernetes values**
   Update the `KubeadmControlPlane` resource in a single edit to keep `spec.version`, the CoreDNS image tag, the etcd image tag, and the infrastructure template reference consistent with the same Alauda OS release:

   - `spec.version` ← **Kubernetes Version** from the OS Support Matrix row
   - `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageTag` ← **coredns** column from the same row
   - `spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.imageTag` ← **etcd** column from the same row
   - `spec.machineTemplate.infrastructureRef.name` ← the new `DCSMachineTemplate` name created in step 1
   - When the target is Kubernetes 1.35 or later, update `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` in `spec.kubeadmConfigSpec.files` in this same edit, as described in Required kubelet patch for Kubernetes 1.35

     ```
     kubectl edit kubeadmcontrolplane <kcp-name> -n cpaas-system
     ```

   Updating only `spec.version` is not sufficient. The CoreDNS and etcd image tags must move together with the Kubernetes version because they are built from the same Alauda OS release; leaving them at the previous values can result in CoreDNS and etcd pods that do not match the new Kubernetes minor version.

3. **Monitor the rolling update**

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/control-plane
kubectl get nodes
```

`KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge` must remain `0` when the cluster relies on pool-managed persistent disks, so the control plane VMs are replaced one at a time.

Upgrade Worker Nodes

Worker node Kubernetes upgrades are managed through `MachineDeployment` resources. Worker upgrades carry fewer fields than the control plane: the CoreDNS and etcd image tags are part of `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration`, which `MachineDeployment` does not have. Worker nodes inherit Kubernetes component versions from the new VM template; the `MachineDeployment` only needs the target Kubernetes version and the new template reference.

Before you start, read the **Kubernetes Version** cell from the target ACP row in the OS Support Matrix and confirm the target VM template name on DCS, as described in Required Values From the OS Support Matrix.

Procedure

1. **Create a new `DCSMachineTemplate` for worker nodes**
   - Create a new `DCSMachineTemplate` with a `vmTemplateName` set to the VM template created on DCS from the Alauda OS image published for the target ACP release. For the export, rename, and apply commands, see Changing Machine Infrastructure
   - Keep `/var/cpaas` and any other upgrade-preserved disks in `DCSIpHostnamePool.spec.pool[].persistentDisk` rather than reintroducing them as template disks
   - Keep additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]`
2. **Update the `MachineDeployment`**
   - Set `spec.template.spec.version` to the **Kubernetes Version** value from the same OS Support Matrix row

- Set `spec.template.spec.infrastructureRef.name` to the new `DCSMachineTemplate` name created in step 1
- When the target is Kubernetes 1.35 or later, create a new `KubeadmConfigTemplate` containing the required kubelet patch (see Updating Bootstrap Templates) and set `spec.template.spec.bootstrap.configRef.name` to that template in the same edit; for earlier versions, update the bootstrap reference only when other bootstrap changes are required

3. **Monitor the rolling update**

   - Verify that the rolling update completes successfully
   - Verify that the new worker nodes join the cluster with the target Kubernetes version

   `MachineDeployment.spec.strategy.rollingUpdate.maxSurge` must remain `0` when the cluster relies on pool-managed persistent disks, so the worker nodes are replaced one at a time.

Recovering From a Failed Phase 2 Upgrade

Do not treat a Kubernetes minor downgrade as an ordinary rollback. Choose the recovery path from the rollout stage:

1. **No target-version control-plane `Machine` has been created**: restore the previous Kube-OVN state and the previous `KubeadmControlPlane` and `MachineDeployment` manifest values. This cancels the target rollout before a new control-plane data format is introduced.
2. **Only the machine template or OS image changed, and the Kubernetes minor did not change**: point the controlling resource back to the previous template. Cluster API performs another replacement rollout. Keep the Kubernetes minor unchanged.
3. **A control-plane `Machine` on the target Kubernetes minor has joined the cluster**: do not patch Kubernetes, CoreDNS, or etcd back to the previous minor. Stop further rollout, repair forward on the target minor, or restore the cluster from the verified pre-upgrade backup by using the supported ACP recovery procedure.

If a target-minor control-plane `Machine` was created but never joined, first restore healthy etcd quorum and determine whether the failed replacement can be removed safely. Do not assume that changing the version fields alone is sufficient.

Keep these infrastructure facts in mind during any recovery:

- **The old VMs are gone.** They were destroyed during the upgrade. Template recovery builds a fresh set of replacement machines; it does not restore the original VMs.
- **The old `DCSMachineTemplate` resource must still exist.** Do not delete the previous template until the new rollout is healthy. If you already deleted it, recreate it from version control or backup before attempting same-minor template recovery.
- **Pool-managed disk identity is preserved, but data state is not.** Disks declared in `DCSIpHostnamePool.spec.pool[].persistentDisk` reattach to the replacement machines at the same IP slot, but data written during the upgrade window remains on those disks.

For stage 1, use the DCS rule in Restore Kube-OVN During Stage-1 Recovery. The recovery action depends on the installed provider version: current providers restore the complete source from the annotation, while legacy providers require the earlier-chart revision patch. Wait until the restored `AppRelease` passes the shared verification before changing control-plane manifests.

The `KubeadmControlPlane` controller can block replacement while etcd is unhealthy. Recover quorum before retrying any safe replacement action.

Fleet Essentials Boundary

Fleet Essentials 1.0.4 and later can request the ACP 4.3-and-later Distribution Version upgrade through CVO. That is Phase 1 of this guide. Fleet Essentials does not perform the DCS Kubernetes and Alauda OS replacement in Phase 2, and it does not manage `DCSIpHostnamePool.spec.pool[].persistentDisk`. Use the YAML procedure on this page for the complete Phase 2 rollout.

Cluster creation and routine node-pool management through Fleet Essentials are outside this upgrade limitation.

Additional Resources

- Migrate Existing Huawei DCS Clusters to Pool-Managed Persistent Disks
- Cluster API Upgrade Documentation ↗
- Kubernetes Version Skew Policy ↗

Upgrade Kubernetes on Huawei Cloud Stack

This guide explains how to upgrade Kubernetes clusters on Huawei Cloud Stack with minimal downtime, while preserving stability and data integrity.

INFO

Where this page fits in the full ACP upgrade flow

This page covers only the Kubernetes step of the upgrade. The full ACP upgrade flow — including upgrade artifact synchronization, ACP Core upgrade through CVO, Aligned plugin upgrades, and Agnostic plugin upgrades from Marketplace — is documented in the following pages. Complete those steps before you start the Kubernetes step on this page:

- Upgrade Overview (scope and sequencing)
- Pre-Upgrade Preparation
- Upgrade the Distribution Version of the `global` cluster (Core, Aligned, Agnostic)
- Upgrade the Distribution Version of workload clusters (Core, Aligned, Agnostic)

Use this page when the same cluster runs on Alauda OS, because the Kubernetes step there replaces nodes from a new VM template rather than upgrading binaries in place.

INFO

Existing Cluster Migration

If your cluster runs ACP `v4.3.1` or later and still keeps preserved data in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`, complete the migration procedure

in Migrate Existing Huawei Cloud Stack Clusters to Pool-Managed Persistent Disks before you rely on upgrade-time disk preservation.

TOC

- Overview
- Upgrade Sequence
- Prerequisites
- Control Plane Upgrades
  - Infrastructure Image Updates
    - Procedure
  - Kubernetes Version Upgrades
    - Required Values From the OS Support Matrix
    - Upgrade Kube-OVN Before the Control Plane
    - Upgrade the Control Plane Kubernetes Version
- Worker Node Upgrades
  - Upgrade the Worker Kubernetes Version
- Recovering From a Failed Phase 2 Upgrade
- Additional Resources

Overview

Cluster upgrades on HCS encompass multiple components and follow a structured approach to ensure system reliability:

- **Control Plane Upgrades**: Update Kubernetes control plane components and underlying infrastructure

- **Worker Node Upgrades**: Upgrade worker nodes with new machine images and Kubernetes versions
- **Infrastructure Updates**: Modify virtual machine specifications, storage, and network configurations

Upgrade Sequence

Upgrade HCS clusters in the following order:

1. **(Prerequisite)** Upgrade the ACP platform on the management cluster first. This brings the `cluster-api-provider-hcs` controller and the related CAPI components to versions that understand the new schema. Trigger workload-cluster upgrades only after the management-side controllers have rolled out and become Ready.
2. Upgrade the Distribution Version on the workload cluster. See Upgrade the Distribution Version.
3. Upgrade Kube-OVN to the chart version required by the target ACP release and wait for the `AppRelease` to reach `Success`.
4. Upgrade the control plane Kubernetes version.
5. Upgrade worker nodes to the target Kubernetes version.

Cluster API orchestrates rolling updates with built-in safety mechanisms to reduce service disruption.

> **WARNING**
>
> Skipping step 1 risks two failure modes: the old controller silently ignores new schema fields written to `HCSMachineConfigPool` / `HCSMachineTemplate`; or a controller image swap mid-rollout interrupts persistent-disk state-machine progression. Always settle the management-side upgrade before touching workload rollout.

Prerequisites

Before you start, ensure all of the following prerequisites are met:

- The Distribution Version upgrade is complete.
- The control plane is reachable.
- All nodes are healthy and in `Ready` state.
- A current etcd backup has been taken and verified by using the supported ACP backup procedure.
- The VM image created from the Alauda OS image published for the target ACP release is present in the HCS environment. The upgrade fails if the image is not present when the new `HCSMachineTemplate` is applied.
- For cross-version upgrades that span more than one Kubernetes minor, the intermediate-version Core images and VM images are pre-staged. See Cross-Version Upgrade Preparation.
- The target Kubernetes version is compatible with your workloads and add-ons.
- Review the Kubernetes upgrade path and version skew policy.
- Any node-local state that must survive replacement is declared in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`, not in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.

For initial deployment, see the Create Cluster guide.

> **WARNING**
>
> **Single-Control-Plane Clusters**
>
> The upgrade workflow in this document applies to HCS clusters with a highly available control plane. Single-control-plane HCS clusters are supported for creation, but they cannot be upgraded.

> **WARNING**
>
> **Disk Preservation Model**
>
> Upgrades rely on Cluster API's rolling replacement mechanism. The HCS provider has four disk classes:

Disk class	Declared in	Survi upgra
System disk (root volume)	`HCSMachineTemplate.spec.template.spec.rootVolume`	❌ Ne
Data volumes	`HCSMachineTemplate.spec.template.spec.dataVolumes`	❌ Ne
Pool-managed persistent disks	`HCSMachineConfigPool.spec.configs[].persistentDisks[]`	✅ Ye
External CSI volumes (HCS EVS CSI, etc.)	Workload PVCs / CSI driver	✅ Un to noc lifecyc

Do not treat node-local data on HCS `dataVolumes[]` as preserved state. Move `/var/cpaas` and any other retained node-local paths to pool-managed persistent disks before the rolling replacement.

INFO

Use a New Machine Template for Each Upgrade Step

Every upgrade step on this page creates a new `HCSMachineTemplate` with a new `metadata.name`, applies it, and then patches the `infrastructureRef.name` of the `KubeadmControlPlane` or `MachineDeployment` to the new template. Keep the previous template until the new rollout is healthy in case rollback is required.

INFO

Fleet Essentials boundary

Fleet Essentials 1.0.4 and later can request the ACP 4.3-and-later Distribution Version upgrade through CVO. It does not perform the HCS Kubernetes and Alauda OS replacement described on this page. Complete Phase 1 with the ACP workflow, then use the YAML procedure below for Phase 2.

Control Plane Upgrades

Control plane upgrades update the Kubernetes API server, etcd, scheduler, and controller manager, along with the underlying VM infrastructure.

For HCS control planes backed by an `HCSMachineConfigPool` that uses pool-managed persistent disks, keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` during upgrades. Persistent disks are bound to fixed `(hostname, slot)` identities, so the rollout must remove the old machine before the replacement machine can reuse the same disk.

Infrastructure Image Updates

Upgrading the underlying machine images for control plane nodes provides security patches, performance improvements, and updated system components.

Procedure

1. **Create Updated Machine Template**
   Copy the existing `HCSMachineTemplate` referenced by `KubeadmControlPlane` and modify the required specifications:

   ```
   kubectl get hcsmachinetemplate <current-template-name> -n cpaas-system
   -o yaml > new-cp-template.yaml
   ```

2. **Modify Template Specifications**
   Modify the new template:

   - Set `metadata.name` to `<new-template-name>`
   - Remove server-generated metadata and status fields from the copied manifest.

- Leave runtime identity fields unset, including `spec.template.spec.providerID` and `spec.template.spec.serverId`. The HCS provider assigns these values when it creates instances.
- Keep preserved paths such as `/var/cpaas` out of `spec.template.spec.dataVolumes[]`. Declare those paths in the referenced `HCSMachineConfigPool.spec.configs[].persistentDisks[]`.
- Update as needed:
  - `spec.template.spec.imageName`
  - `spec.template.spec.flavorName`
  - `spec.template.spec.rootVolume.size`
  - `spec.template.spec.dataVolumes` for temporary disks only

3. **Deploy Updated Template**

   Apply the new machine template:

```
kubectl apply -f new-cp-template.yaml -n cpaas-system
```

4. **Update Control Plane Reference**

   Modify the `KubeadmControlPlane` resource to reference the new template:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='me
rge' -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<new-
template-name>"}}}}'
```

5. **Monitor Rolling Update**

   The control plane will automatically perform a rolling update:

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/control-plane
```

Kubernetes Version Upgrades

Upgrading the Kubernetes version involves updating both the control plane software and the supporting virtual machine images.

Required Values From the OS Support Matrix

The authoritative mapping between an ACP release, the Kubernetes version, and the matching CoreDNS, etcd, and Kube-OVN versions lives in OS Support Matrix. Locate the row that corresponds to the target ACP version before you start; the row supplies the component values the procedure below needs.

The ECS image name is not in the matrix. It is the name the image was uploaded under in HCS from the Alauda OS image published for the target ACP release — see Downloading Alauda OS — and it is what `HCSMachineTemplate.spec.template.spec.imageName` must be set to on both the control plane and worker templates.

The cells you read from that row map to the upgrade manifests as follows:

OS Support Matrix column	Used to set
Kubernetes Version	`KubeadmControlPlane.spec.version` and `MachineDeployment.spec.template.`
coredns	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.dns.im`
etcd	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.etcd.l`
kube-ovn (chart)	`Cluster.metadata.annotations["cpaas.io/kube-ovn-version"]` and the provic `cni-kube-ovn` `AppRelease`

The CoreDNS and etcd image tags are control-plane-only because `clusterConfiguration` is a `KubeadmControlPlane` field. Worker nodes inherit container image versions from the new VM template; the `MachineDeployment` does not carry its own `dns` / `etcd` tags. The

Kube-OVN annotation lives on the `Cluster` resource, not on `KubeadmControlPlane`, because the HCS provider watches it independently of the Kubernetes control plane rollout.

Upgrade Kube-OVN Before the Control Plane

Follow Upgrade Kube-OVN Before the Control Plane and select the Huawei Cloud Stack procedure that matches the **installed** HCS provider version. That shared procedure owns the provider version boundary, the Kube-OVN chart-name migration at `v4.4`, legacy behavior, and the required `AppRelease` health checks.

Do not use the legacy direct `targetRevision` patch for a Kube-OVN `v4.4+` target. Upgrade the HCS provider to `v1.0.4` or later first so it can reconcile the complete source and the related component repository changes.

Upgrade the Control Plane Kubernetes Version

Complete the shared Upgrade Kube-OVN Before the Control Plane procedure and verify that the `cni-kube-ovn` `AppRelease` is at the target revision with `phase=Success` before continuing.

1. **Create a new `HCSMachineTemplate` for the target Kubernetes version**
   Copy the existing control-plane template and apply it under a new `metadata.name` with the target `imageName`:

   ```
   kubectl get hcsmachinetemplate <current-cp-template-name> -n cpaas-syst
   em -o yaml > new-cp-template.yaml
   ```

   In `new-cp-template.yaml`:

   - Set `metadata.name` to `<new-template-name>`.
   - Set `spec.template.spec.imageName` to the name of the target Alauda OS image as uploaded to HCS.
   - Strip server-generated metadata (`resourceVersion`, `uid`, `generation`, `creationTimestamp`, `managedFields`, `kubectl.kubernetes.io/last-applied-configuration` annotation) and the entire `status` field.
   - Leave runtime identity fields unset, including `spec.template.spec.providerID` and `spec.template.spec.serverId`. The HCS provider sets `providerID` to

`hcs://<cluster-name>/<machine-name>` and `serverId` to the HCS ECS instance ID after the VM is created; pre-filling them in the template breaks the controller's identity binding.

```
kubectl apply -f new-cp-template.yaml -n cpaas-system
```

2. **Patch the `KubeadmControlPlane` with the target Kubernetes values**
   Update the `KubeadmControlPlane` resource in a single edit to keep `spec.version`, the CoreDNS image tag, the etcd image tag, and the infrastructure template reference consistent with the same Alauda OS release:

   - `spec.version` ← **Kubernetes Version** from the OS Support Matrix row
   - `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageTag` ← **coredns** column from the same row
   - `spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.imageTag` ← **etcd** column from the same row
   - `spec.machineTemplate.infrastructureRef.name` ← the new `HCSMachineTemplate` name created in step 1
   - When the target is Kubernetes 1.35 or later, update `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` in `spec.kubeadmConfigSpec.files` in this same edit, as described in Required kubelet patch for Kubernetes 1.35

   ```
   kubectl edit kubeadmcontrolplane <kcp-name> -n cpaas-system
   ```

   Updating only `spec.version` is not sufficient. The CoreDNS and etcd image tags must move together with the Kubernetes version because they are built from the same Alauda OS release; leaving them at the previous values can result in CoreDNS and etcd pods that do not match the new Kubernetes minor version.
   Keep `spec.rolloutStrategy.rollingUpdate.maxSurge: 0` when the referenced control plane pool uses persistent disks. The replacement machine must reuse the same fixed hostname and disk slot after the old machine is removed.

3. **Verify Upgrade Progress**
   Monitor the rolling upgrade process:

```
# Check control plane status
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system

# Monitor individual machines
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/control-plane

# Verify cluster health
kubectl get nodes
```

Worker Node Upgrades

Worker node upgrades are managed via `MachineDeployment` resources.

Upgrade the Worker Kubernetes Version

Kubernetes version upgrades require coordinated updates to both the `MachineDeployment` and the underlying VM template.

Note: Ensure the VM template's Kubernetes version matches the version specified in the `MachineDeployment`. Mismatched versions will cause node join failures.

Procedure:

1. **Update Machine Template**
   Create a new `HCSMachineTemplate` with an updated `imageName` that supports the target Kubernetes version. For the export, rename, and apply commands, see Changing Machine Infrastructure.
2. **Create a new bootstrap template for Kubernetes 1.35 or later**
   If the target is Kubernetes 1.35 or later, clone the current `KubeadmConfigTemplate` under a new name and add the required kubelet patch settings to `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`. For the export, edit, and apply steps, see Updating Bootstrap Templates. Skip this step for earlier versions unless the release has another bootstrap change.
3. **Update MachineDeployment**
   Modify the following fields:

- `spec.template.spec.version` - Target Kubernetes version
- `spec.template.spec.infrastructureRef.name` - New machine template name
- `spec.template.spec.bootstrap.configRef.name` - New bootstrap template name; required for Kubernetes 1.35 or later

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='merge' -p='{"spec":{"template":{"spec":{"version":"<kubernetes-version>","infrastructureRef":{"name":"<new-template>"},"bootstrap":{"configRef":{"name":"<new-bootstrap-template>"}}}}}}'
```

The command includes the bootstrap reference required for Kubernetes 1.35 or later. For an earlier target with no bootstrap change, omit the `bootstrap` object.

4. **Monitor Upgrade**
   Verify that new nodes join the cluster with the correct Kubernetes version:

```
kubectl get nodes
```

Recovering From a Failed Phase 2 Upgrade

Do not treat a Kubernetes minor downgrade as an ordinary rollback. Choose the recovery path from the rollout stage:

1. **No target-version control-plane `Machine` has been created**: restore the previous Kube-OVN state and the previous `KubeadmControlPlane` and `MachineDeployment` manifest values. This cancels the target rollout before a new control-plane data format is introduced.
2. **Only the machine template or OS image changed, and the Kubernetes minor did not change**: point the controlling resource back to the previous template. Cluster API performs another replacement rollout. Keep the Kubernetes minor unchanged.
3. **A control-plane `Machine` on the target Kubernetes minor has joined the cluster**: do not patch Kubernetes, CoreDNS, or etcd back to the previous minor. Stop further rollout, repair forward on the target minor, or restore the cluster from the verified pre-upgrade backup by using the supported ACP recovery procedure.

If a target-minor control-plane `Machine` was created but never joined, first restore healthy etcd quorum and determine whether the failed replacement can be removed safely. Do not assume that changing the version fields alone is sufficient.

Keep these infrastructure facts in mind during any recovery:

- **The old VMs are gone.** They were destroyed during the upgrade. Template recovery builds a fresh set of replacement machines; it does not restore the original VMs.
- **The old `HCSMachineTemplate` resource must still exist.** Do not delete the previous template until the new rollout is healthy. If you already deleted it, recreate it from version control or backup before attempting same-minor template recovery.
- **Only pool-managed persistent disks preserve node-local state.** Data written to `HCSMachineTemplate.spec.template.spec.dataVolumes[]` during the upgrade window is lost when that VM is replaced. Data written to disks declared in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` is retained and reattached to the replacement VM. Application data should still use external persistent storage such as HCS EVS CSI unless your operational design explicitly depends on node-local state.

For stage 1, use the HCS rule in Restore Kube-OVN During Stage-1 Recovery. The recovery action depends on the installed provider version: current providers restore the complete source from the annotation, while legacy providers require the earlier-chart revision patch. Wait until the restored `AppRelease` passes the shared verification before changing control-plane manifests.

The `KubeadmControlPlane` controller can block replacement while etcd is unhealthy. Recover quorum before retrying any safe replacement action.

Additional Resources

- Cluster API Upgrade Documentation ↗
- Kubernetes Version Skew Policy ↗

Upgrade Kubernetes on VMware vSphere

This document explains how to upgrade Kubernetes clusters on VMware vSphere after the platform-side distribution upgrade is complete. The documented workflow focuses on updating the control plane and worker nodes through Cluster API resources. The rollout rules are listed in Provider Requirements.

INFO

Where this page fits in the full ACP upgrade flow

This page covers only the Kubernetes step of the upgrade. The full ACP upgrade flow — including upgrade artifact synchronization, ACP Core upgrade through CVO, Aligned plugin upgrades, and Agnostic plugin upgrades from Marketplace — is documented in the following pages. Complete those steps before you start the Kubernetes step on this page:

- Upgrade Overview (scope and sequencing)
- Pre-Upgrade Preparation
- Upgrade the Distribution Version of the `global` cluster (Core, Aligned, Agnostic)
- Upgrade the Distribution Version of workload clusters (Core, Aligned, Agnostic)

Use this page when the same cluster runs on Alauda OS, because the Kubernetes step there replaces nodes from a new VM template rather than upgrading binaries in place.

TOC

Upgrade Sequence

Prerequisites
Required Values From the OS Support Matrix
Steps
Check the manifests against the admission rules
Upgrade Kube-OVN Before the Control Plane
Create the target machine templates
Upgrade the control plane
Upgrade the worker nodes
OS Image Update Without a Kubernetes Version Change
Rollout Holds That Are Not Failures
Recovering From a Failed Phase 2 Upgrade
Verification
Troubleshooting
Next Steps

Upgrade Sequence

Upgrade VMware vSphere clusters in the following order:

1. **(Prerequisite)** Upgrade the ACP platform on the management cluster first. This brings the `cluster-api-provider-vsphere` controller and the related CAPI components to versions that understand the new schema. The provider updates its CRDs and migrates existing resources to the new field layout itself — no manual YAML change is required. See Upgrading the Provider. Trigger workload-cluster upgrades only after the management-side controllers have rolled out and become Ready.
2. Complete the distribution-version upgrade described in Upgrade the Distribution Version.
3. Verify that the existing `KubeadmControlPlane` and `MachineDeployment` manifests satisfy the rollout rules. A `KubeadmControlPlane` that does not can block the very first edit of the upgrade, and a `MachineDeployment` that does not stalls the rollout. See Check the manifests against the admission rules.
4. Verify that the control plane is healthy and the current cluster is stable.

5. Upgrade Kube-OVN to the chart version required by the target ACP release and wait for the `AppRelease` to reach `Success`.
6. Upgrade the control plane Kubernetes version.
7. Upgrade worker nodes to the target Kubernetes version.

Prerequisites

Before you begin, ensure the following conditions are met:

- The distribution-version upgrade is complete.
- The control plane is healthy and reachable.
- All nodes are in the `Ready` state.
- A current etcd backup has been taken and verified by using the supported ACP backup procedure.
- The VM template built from the Alauda OS image published for the target ACP release is present in the vSphere environment. The upgrade fails if the template is not present when the new `VSphereMachineTemplate` is applied.
- For cross-version upgrades that span more than one Kubernetes minor, the intermediate-version Core images and VM templates are pre-staged. See Cross-Version Upgrade Preparation.
- The target Kubernetes version is compatible with your workloads and add-ons.
- The machine config pools have enough capacity for rolling updates.
- The `KubeadmControlPlane` sets `spec.rolloutStrategy.rollingUpdate.maxSurge: 0` and `spec.replicas` to `3` or more, and every `MachineDeployment` sets `spec.strategy.rollingUpdate.maxSurge: 0`. These values let the replacement reuse the same slot and disk identity. Verify them with Check the manifests against the admission rules **before** you start.
- The management cluster is not a disaster-recovery standby. Against a standby, the provider skips non-`global` objects and requeues every 30 seconds, so the rollout appears not to start rather than to fail. See Standby Management Clusters.
- You know the `KubeadmControlPlane` name for this cluster. Read it from `Cluster.spec.controlPlaneRef.name`; the manifests in Create a Cluster on VMware

vSphere name it `<cluster_name>-kcp`.

- If the cluster uses a Self-built VIP (`controlPlaneLoadBalancer.type: internal`), `VSphereCluster.status.conditions[SelfBuiltLoadBalancerReady]` is `True`, and the **new** VM template satisfies the Alive kernel prerequisites (the `ip_vs` module family is loadable, `net.ipv4.vs.conntrack=1` is permitted). Replacement control-plane nodes are built from that template, and the provider neither injects nor validates these settings.
- Review the Kubernetes upgrade path and version skew policy.

> **WARNING**
>
> **The rollout rules can reject the first edit of an existing cluster**
>
> The `KubeadmControlPlane` rules apply to `UPDATE` as well as `CREATE`, so an existing cluster can hold values that are no longer accepted:
>
> - a `KubeadmControlPlane` whose `rolloutStrategy.rollingUpdate.maxSurge` is unset — the Cluster API default is `1`;
> - a `KubeadmControlPlane` with `replicas: 1`.
>
> The rejection lands on the version bump itself, so the upgrade stops before anything is replaced. Correct `maxSurge` in a separate, earlier edit; see Check the manifests against the admission rules. A cluster with a single control plane node (`replicas: 1`) cannot be upgraded.
>
> A `MachineDeployment` that does not set `strategy.rollingUpdate.maxSurge: 0` is not rejected, but its rollout stalls: with the default `maxSurge` of `1`, the rollout needs an extra slot that a full pool does not have. Correct it in the same earlier edit.

> **WARNING**
>
> **Disk Preservation Model**
>
> Each cluster has five disk classes; only the pool-managed persistent class and external CSI volumes survive a delete-recreate.

Disk class	Declared in	S u
System disk (root volume)	The VM template used for `spec.template.spec.template`	❌
Template-local disks	`VSphereMachineTemplate.spec.template.spec.*` (additional disks declared in the template)	❌
Pool-managed ephemeral disks	`VSphereMachineConfigPool.spec.configs[].ephemeralDisks[]`	❌ th d th n d fo re
Pool-managed persistent disks	`VSphereMachineConfigPool.spec.configs[].persistentDisks[]`	✅ fr a to a s
External CSI volumes (vSphere CSI, etc.)	Workload PVCs / CSI driver	✅ to li

Unlike pool-managed persistent disks, ephemeral disks take no part in reclaim and never hold a slot release, so they never delay a rollout. See Choose persistent or ephemeral.

Which class your disks are in is fixed at cluster creation. `ephemeralDisks[]` is new in provider `v1.0.17`. A cluster created on an earlier provider declares `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` as persistent disks, with `wipeFilesystem: true` on the etcd disk, and an upgrade does not change that: the provider migrates no disk. Do not restructure the pool's disk lists as part of an upgrade — the standard persistent-plus-ephemeral split applies to newly created clusters only.

"Preserved" means the same disk identity is reattached — it does not mean the disk's contents are time-traveled. Anything written to a pool-managed disk during the upgrade window stays after the upgrade and stays after a rollback.

WARNING

Create a New Template for Every Change

Every upgrade step on this page creates a new `VSphereMachineTemplate` with a new `metadata.name`, applies it, and then patches the controlling resource's `infrastructureRef.name` to the new template. Do not edit an existing template. Keep the previous template until the new rollout is healthy in case rollback is required.

INFO

Fleet Essentials boundary

Fleet Essentials 1.0.4 and later can request the ACP 4.3-and-later Distribution Version upgrade through CVO. It does not perform the vSphere Kubernetes and Alauda OS replacement described on this page. Complete Phase 1 with Upgrade the Distribution Version, then use the YAML procedure below for Phase 2.

WARNING

An upgraded cluster cannot use the Self-built VIP

The Self-built VIP is available only to clusters created with provider `v1.0.17` or later on ACP `v4.4` or later. A cluster that was created on an earlier provider keeps its external LoadBalancer after the provider is upgraded, and cannot be moved onto a Self-built VIP: the provider injects the VIP while the first control-plane node bootstraps, and there is no in-place migration for a running control plane. See Control Plane Endpoint Modes.

The endpoint itself does not move, and an upgrade is not an opportunity to change it. The control plane endpoint — the mode and the VIP or LoadBalancer address — is chosen when the cluster is deployed and is fixed from then on; it cannot be adjusted while the cluster runs or during an upgrade. An existing cluster needs no action and gets none: `spec.controlPlaneLoadBalancer` stays unset, which is exactly what an external LoadBalancer looks like, and writing it now is

rejected. The field is optional for this reason — a cluster created before the Self-built VIP existed keeps its endpoint configuration untouched across the upgrade. See Upgrading the Provider.

Required Values From the OS Support Matrix

The authoritative mapping between an ACP release, the Kubernetes version, and the matching CoreDNS, etcd, and Kube-OVN versions lives in OS Support Matrix. Locate the row that corresponds to the target ACP version before you start; the row supplies the component values the steps below need.

The VM template name is not in the matrix. It is the name the template was created under in vSphere from the Alauda OS image published for the target ACP release — see Downloading Alauda OS — and it is what `VSphereMachineTemplate.spec.template.spec.template` must be set to on both the control plane and worker templates.

The cells you read from that row map to the upgrade manifests as follows:

OS Support Matrix column	Used to set
Kubernetes Version	`KubeadmControlPlane.spec.version` and `MachineDeployment.spec.template.`
coredns	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.dns.im`
etcd	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.etcd.l`

OS Support Matrix column	Used to set
kube-ovn (chart)	`Cluster.metadata.annotations["cpaas.io/kube-ovn-version"]` and the provic `cni-kube-ovn` `AppRelease`

The CoreDNS and etcd image tags are control-plane-only because `clusterConfiguration` is a `KubeadmControlPlane` field. Worker nodes inherit container image versions from the new VM template; the `MachineDeployment` does not carry its own `dns` / `etcd` tags. The Kube-OVN annotation lives on the `Cluster` resource, not on `KubeadmControlPlane`.

Steps

1 Check the manifests against the admission rules

Read the current values before you change anything:

```
kubectl -n <namespace> get kubeadmcontrolplane <cluster_name>-kcp \
  -o jsonpath='replicas={.spec.replicas} maxSurge={.spec.rolloutStrat
egy.rollingUpdate.maxSurge}{"\n"}'

kubectl -n <namespace> get machinedeployment -o custom-columns='NAM
E:.metadata.name,TYPE:.spec.strategy.type,SURGE:.spec.strategy.rollin
gUpdate.maxSurge,UNAVAIL:.spec.strategy.rollingUpdate.maxUnavailable'
```

An empty `maxSurge` value means the field is unset, and unset means the Cluster API default `1`, which is not valid for a pool-backed controller. The `MachineDeployment` example uses `maxUnavailable: 1`. Correct the values before the version bump:

```
# replicas is 3 or more: correct maxSurge.
kubectl -n <namespace> patch kubeadmcontrolplane <cluster_name>-kcp -
-type='merge' \
  -p='{"spec":{"rolloutStrategy":{"type":"RollingUpdate","rollingUpda
te":{"maxSurge":0}}}}'

kubectl -n <namespace> patch machinedeployment <cluster_name>-md-0 --
type='merge' \
  -p='{"spec":{"strategy":{"type":"RollingUpdate","rollingUpdate":{"m
axSurge":0,"maxUnavailable":1}}}}'
```

If `replicas` is less than 3, the cluster cannot be upgraded.

A `MachineDeployment` that uses `strategy.type: OnDelete` needs no surge setting, because `OnDelete` already replaces machines delete-first.

Verify that the pools are healthy before you start:

```
kubectl -n <namespace> get vspheremachineconfigpool
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp
-pool \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status} ({.reas
on}){"\n"}{end}'
```

Expect `Ready=True`, `MembersValid=True`, `MembersUnique=True`, and `PersistentDisksReady=True`. `SlotAvailable` can legitimately be `False` on a fully allocated pool; it is not part of the `Ready` summary.

2 Upgrade Kube-OVN Before the Control Plane

Follow Upgrade Kube-OVN Before the Control Plane and select the VMware vSphere tab that matches the **installed** provider version: a cluster still on `v1.0.15` or earlier follows the legacy tab, and a Kube-OVN `v4.4` or later target requires upgrading the provider to `v1.0.16` or later first. The shared procedure includes the vSphere-specific network annotation check, the chart-name migration at Kube-OVN `v4.4`, and the required `AppRelease` health checks. Do not patch the chart source by hand; the provider reconciles it from the annotation.

3 Create the target machine templates

Before you start the rolling upgrade, create new `VSphereMachineTemplate` resources for the control plane and workers.

1. **Export the existing control plane template**

```
kubectl get vspheremachinetemplate <cluster_name>-control-plane -n <namespace> -o yaml > new-cp-template.yaml
```

2. **Modify the control plane template**

   Edit `new-cp-template.yaml`:

   - Set `metadata.name` to a new unique name (for example, `<cluster_name>-control-plane-v2`)
   - Update `spec.template.spec.template` to the target VM template name
   - Update CPU, memory, or disk settings if needed
   - Remove server-generated fields: `metadata.resourceVersion`, `metadata.uid`, `metadata.generation`, `metadata.creationTimestamp`, `metadata.managedFields`, `metadata.annotations["kubectl.kubernetes.io/last-applied-configuration"]`, and `status`
   - Leave `spec.template.spec.providerID` unset. The vSphere provider sets `providerID` to the VM's BIOS UUID once the VM is created; pre-filling it in the template breaks the controller's identity binding.

3. **Export and modify the worker template**

```
kubectl get vspheremachinetemplate <cluster_name>-worker -n <namespace> -o yaml > new-worker-template.yaml
```

   Edit `new-worker-template.yaml`:

   - Set `metadata.name` to a new unique name (for example, `<cluster_name>-worker-v2`)
   - Update `spec.template.spec.template` to the target VM template name

- Update CPU, memory, or disk settings if needed
- Remove the same server-generated fields listed above

4. **Apply both new templates**

```
kubectl apply -f new-cp-template.yaml
kubectl apply -f new-worker-template.yaml
```

4 Upgrade the control plane

Before you start, complete the shared Upgrade Kube-OVN Before the Control Plane procedure and verify that the `cni-kube-ovn` `AppRelease` is at the target revision with `phase=Success`. Then collect every required control-plane value from the target ACP row in the OS Support Matrix as described in Required Values From the OS Support Matrix.

1. **Patch the `KubeadmControlPlane` with the target Kubernetes values**
   Update the `KubeadmControlPlane` resource in a single edit to keep `spec.version`, the CoreDNS image tag, the etcd image tag, and the infrastructure template reference consistent with the same VM template:
   - `spec.version` ← **Kubernetes Version** from the OS Support Matrix row
   - `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageTag` ← **coredns** column from the same row
   - `spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.imageTag` ← **etcd** column from the same row
   - `spec.machineTemplate.infrastructureRef.name` ← the new `VSphereMachineTemplate` name created above
   - When the target is Kubernetes 1.35 or later, update `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` in `spec.kubeadmConfigSpec.files` in this same edit, as described in Required kubelet patch for Kubernetes 1.35

```
kubectl edit kubeadmcontrolplane <cluster_name>-kcp -n <namespac
e>
```

Updating only `spec.version` is not sufficient. The CoreDNS and etcd image tags must move together with the Kubernetes version because they are built from the same release; leaving them at the previous values can result in CoreDNS and etcd pods that do not match the new Kubernetes minor version.

> **INFO**
>
> After the provider reconciles, expect the CoreDNS and kube-proxy skip annotations and a controller-set `dns.imageRepository` on the object you just edited. Do not remove them, and do not re-apply a stored manifest that lacks them. The workload cluster's kube-proxy DaemonSet and CoreDNS Deployment repositories are reconciled separately and only after this rollout completes, so a lag there is expected. See Provider-managed fields on your manifests.
>
> On a Self-built VIP cluster, Alive keeps the VIP throughout the rollout. The bootstrap VIP is injected only on the original `kubeadm init` node; replacement control-plane nodes join through the VIP and never receive it. As control-plane membership changes, the provider updates the Alive backend list from the current control-plane node addresses. Keep `maxSurge: 0` — the VIP and etcd quorum both depend on one-at-a-time replacement.

2. **Monitor the control plane rollout**

```
kubectl -n <namespace> get kubeadmcontrolplane <cluster_name>-kcp
-w
kubectl -n <namespace> get machine -l cluster.x-k8s.io/control-pla
ne
```

5 Upgrade the worker nodes

After the control plane upgrade completes, update the `MachineDeployment` to reference the new worker template and the target Kubernetes version.

Typical changes include:

- `spec.template.spec.version` — the target Kubernetes version

- `spec.template.spec.infrastructureRef.name` — the new `VSphereMachineTemplate` name
- `spec.template.spec.bootstrap.configRef.name` — the new `KubeadmConfigTemplate` name. This is required for Kubernetes 1.35 or later so the worker receives the required kubelet patch; for earlier versions, change it only when other bootstrap settings must change. See Updating Bootstrap Templates.

The patch below changes only `spec.template.spec`. If the pre-flight check showed that the `MachineDeployment` strategy needed correcting, that correction must already be applied before you patch.

Apply the changes:

The following command includes the bootstrap reference required for Kubernetes 1.35 or later. For an earlier target with no bootstrap change, omit the `bootstrap` object.

```
kubectl patch machinedeployment <cluster_name>-md-0 -n <namespace> \
  --type='merge' -p='{
    "spec": {
      "template": {
        "spec": {
          "version": "<target_kubernetes_version>",
          "infrastructureRef": {
            "name": "<new-worker-template-name>"
          },
          "bootstrap": {
            "configRef": {
              "name": "<new-bootstrap-template-name>"
            }
          }
        }
      }
    }
  }'
```

Monitor the worker rollout:

```
kubectl -n <namespace> get machinedeployment <cluster_name>-md-0 -w
kubectl -n <namespace> get machine
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
```

OS Image Update Without a Kubernetes Version Change

Use this procedure to move nodes onto a new Alauda OS image while the Kubernetes version stays the same — for example, to pick up an OS security fix, or to retrofit the pod log directory relocation onto existing nodes.

1. Create new `VSphereMachineTemplate` resources with the export, scrub, and rename procedure in Create the target machine templates. Change **only** `spec.template.spec.template` to the new VM template name.
2. Apply both new templates.
3. Patch `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name`, wait for the control-plane rollout to finish, then patch `MachineDeployment.spec.template.spec.infrastructureRef.name`.

Do not touch `spec.version`, `clusterConfiguration.dns.imageTag`, `clusterConfiguration.etcd.local.imageTag`, or the `cpaas.io/kube-ovn-version` annotation. An OS-image rollout is not a Kubernetes hop, so it does not need the Kube-OVN-first ordering.

The same rollout rules apply: `maxSurge: 0` on both controllers. Pool-managed persistent disks reattach exactly as they do in a version upgrade, and pool-managed ephemeral disks are destroyed and recreated empty.

To recover, point the controlling resource back at the previous template. This is stage 2 of Recovering From a Failed Phase 2 Upgrade.

Rollout Holds That Are Not Failures

Some rollout pauses are the provider protecting data. Do not clear them by hand.

Attachment-safety hold. When a `VSphereVM` is deleted, the provider scans the datacenter for VMs that still have the slot's persistent-disk backings attached. While any of them is attached, it keeps the finalizer and requeues. The same check guards pool reclaim. A rollout that appears stalled at the deletion of the first old machine is frequently this hold rather than a defect. **Do not remove the finalizer.**

Read the current state:

```
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> \
  -o jsonpath='{range .status.persistentDiskStatuses[*]}{.hostname}{" "}
{.name}{" "}{.phase}{" owner="}{.ownerMachineName}{" err="}{.lastError}
{"\n"}{end}'
kubectl -n <namespace> get vspherevm -o custom-columns='NAME:.metadata.na
me,DELETED:.metadata.deletionTimestamp,FINALIZERS:.metadata.finalizers'
```

The normal handoff is `Attached`, then `Available` once the old VM is gone, then `Attached` again on the replacement. `ownerMachineName` and `ownerMachineUID` are preserved across the delete and recreate so the replacement reattaches the same disks.

`Reclaimed` is a tombstone. Do not delete a `phase: Reclaimed` entry to unstick a rollout. See Troubleshooting on the cluster-creation page for why removing it produces a re-seed loop.

`SlotAvailable=False` is expected. With `maxSurge: 0` and a fully allocated pool, no slot is free until an old machine is deleted. `SlotAvailable` is excluded from the pool's `Ready` summary precisely so this does not read as a failure.

A node powered off in vCenter stays off. Once `VSphereVM` latches `InitialPowerOnCompleted=True`, the controller no longer forces a powered-off VM back on. A replacement VM created during a rollout has an unset latch, so a node that was deliberately powered off before the upgrade comes back powered on after replacement.

Recovering From a Failed Phase 2 Upgrade

Do not treat a Kubernetes minor downgrade as an ordinary rollback. Choose the recovery path from the rollout stage:

1. **No target-version control-plane `Machine` has been created**: restore the previous Kube-OVN annotation and the previous `KubeadmControlPlane` and `MachineDeployment` manifest values. This cancels the target rollout before a new control-plane data format is introduced.
2. **Only the machine template or OS image changed, and the Kubernetes minor did not change**: point the controlling resource back to the previous template. Cluster API performs another replacement rollout. Keep the Kubernetes minor unchanged.
3. **A control-plane `Machine` on the target Kubernetes minor has joined the cluster**: do not patch Kubernetes, CoreDNS, or etcd back to the previous minor. Stop further rollout, repair forward on the target minor, or restore the cluster from the verified pre-upgrade backup by using the supported ACP recovery procedure.

If a target-minor control-plane `Machine` was created but never joined, first restore healthy etcd quorum and determine whether the failed replacement can be removed safely. Do not assume that changing the version fields alone is sufficient.

Keep these infrastructure facts in mind during any recovery:

- **The old VMs are gone.** They were destroyed during the upgrade. Template recovery builds a fresh set of replacement machines; it does not restore the original VMs.
- **The old `VSphereMachineTemplate` resource must still exist.** Do not delete the previous template until the new rollout is healthy. If you already deleted it, recreate it from version control or backup before attempting same-minor template recovery.
- **Pool-managed disk identity is preserved, but data state is not.** Disks declared in `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` reattach to the replacement machines at the same slot, but data written during the upgrade window remains on those disks.
- **Ephemeral disks are gone.** Data on `VSphereMachineConfigPool.spec.configs[].ephemeralDisks[]` is destroyed with every replaced VM and cannot be recovered from the pool. Only pool-managed persistent disks reattach.
- **Do not reverse a rollout-rule correction.** If stage 1 requires restoring earlier manifest values, leave `maxSurge` at `0`.

For stage 1, use the vSphere rule in Restore Kube-OVN During Stage-1 Recovery. The provider restores the chart from the annotation. Wait until the restored `AppRelease` passes the shared verification before changing control-plane manifests.

The `KubeadmControlPlane` controller can block replacement while etcd is unhealthy. Recover quorum before retrying any safe replacement action.

Verification

Read the pool state back after the upgrade:

```
kubectl -n <namespace> get vspheremachineconfigpool
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> \
  -o jsonpath='{range .status.persistentDiskStatuses[*]}{.hostname}{" "}
{.name}{" "}{.phase}{"\n"}{end}'
```

Confirm the following results:

- `KubeadmControlPlane` reaches the target version and desired replica count.
- `MachineDeployment` reaches the target version and desired replica count.
- Control plane and worker nodes return to the `Ready` state.
- The vSphere CPI daemonset remains available in the workload cluster.
- Each `VSphereMachineConfigPool` reports `Ready=True` and `PersistentDisksReady=True`, its `ALLOCATED` count equals the number of running Machines drawn from it, and its `TOTAL` count is unchanged.
- Every persistent disk on an allocated slot reports `phase: Attached`.

Troubleshooting

Issue	What to check
The `kubectl edit` or `kubectl patch` on	Run the pre-flight check. For the full message-to-cause map, see Admission rejections.

Issue	What to check
`KubeadmControlPlane` is rejected	
The rollout stalls after the first machine is deleted	Usually the attachment-safety hold rather than a defect. See Rollout Holds That Are Not Failures.
The pool reports `Ready=True` but `SlotAvailable=False`	Normal on a fully allocated pool during a `maxSurge: 0` rollout. `SlotAvailable` is excluded from the `Ready` summary.
A replaced node's disks are empty	Confirm which list the disk is declared in. On a cluster created on `v1.0.17` or later, `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` are ephemeral by design and start empty on every replacement. On an older cluster they are persistent disks, so also check `wipeFilesystem`, which is `true` on the etcd disk by design.
Nothing reconciles, no conditions change, and no error appears	The management cluster may be a disaster-recovery standby. See Standby Management Clusters.
A workload node is powered off and does not come back	Expected once `InitialPowerOnCompleted` has latched. Power it on in vCenter. See Rollout Holds That Are Not Failures.
A new control-plane machine never bootstraps	Read the `BootstrapReady` condition on `VSphereVM` and `VSphereMachine`; the reasons are `BootstrapSecretGetFailed` and `BootstrapSecretContentInvalid`.
The Kube-OVN annotation changed but the chart name or revision did not converge	Follow the shared Kube-OVN procedure. Do not patch the chart source manually.
The workload `kube-proxy` or `coredns` is still on the old image repository after the control plane reaches the target version	Expected lag — those are reconciled only after the control-plane rollout completes. See Provider-managed fields on your manifests.

Issue	What to check
A `VSphereMachineTemplate` update is rejected as immutable	Create a new template under a new name. See the warning near the top of this page.

Next Steps

After the Kubernetes upgrade is complete, continue with routine node operations in Manage Nodes on VMware vSphere.

Migrate Existing Huawei DCS Clusters to Pool-Managed Persistent Disks

Use this guide when you upgrade an existing Huawei DCS cluster from the older template-disk layout to the current pool-managed persistent-disk model.

This migration is YAML-driven because `DCSIpHostnamePool.spec.pool[].persistentDisk` is not exposed in the web UI.

INFO

Version

Use this procedure when the cluster runs ACP `v4.2.1` or later.

This procedure currently assumes all of the following:

- The DCS VM templates are `4.2.1` or later.
- Guest tools (`vmtools`) are working inside the guest OS so safe shutdown and disk detach can complete.

TOC

Overview

Before You Start

Inspect the Current Disk Layout

Determine Which Disks Are Claimable

Worked Example
Update the `DCSMachineTemplate`
Update the `DCSIpHostnamePool`
Trigger the Rolling Upgrade
Verify Claim, Detach, Conversion, and Reattach
Limitations and Recovery Notes
Related Topics

Overview

Older DCS clusters created reusable data disks through `DCSMachineTemplate`. That layout does not give the controller enough information to preserve disks safely during delete-recreate replacement.

The current model moves upgrade-preserved disks into `DCSIpHostnamePool.spec.pool[].persistentDisk`. Each disk is bound to an `(ip, slot)` identity. During rolling replacement, the controller:

1. Claims the existing disk from the old VM.
2. Safely stops the old VM.
3. Detaches the disk.
4. Converts stock volumes to independent persistent volumes when needed.
5. Deletes the old VM.
6. Reattaches the disk to the replacement VM.
7. Boots the replacement VM, which mounts the existing filesystem without reformatting it.

This is also the documented model for the platform-required `/var/cpaas` disk.

Before You Start

Verify all of the following before you begin:

- The cluster is healthy and currently stable.
- Because pool-managed persistent disks require one-by-one replacement, the relevant control plane and worker rollout strategies use `maxSurge: 0`.
- You can identify the current disk `sequenceNum` values on the old VMs from the DCS UI or by querying VM details through the DCS API.
- You know which disks must be preserved and which disks can still be recreated with the VM.
- The target `DCSIpHostnamePool` already exists and maps each node to a fixed IP slot.

Inspect the Current Disk Layout

First, identify the management-cluster objects and the DCS VM that backs each node:

```
kubectl get kubeadmcontrolplane -n cpaas-system
kubectl get machinedeployment -n cpaas-system
kubectl get machine -n cpaas-system
kubectl get dcsmachine -n cpaas-system
kubectl get dcsiphostnamepool -n cpaas-system
```

For any `DCSMachine` you plan to migrate, inspect the current VM details and record the disk `sequenceNum`, size, datastore, and PCI type for each disk you want to preserve.

You can gather that information from:

- The DCS platform UI.
- Your existing operational tooling that wraps `QueryVmInfo`.
- Direct API inspection if your environment already exposes that workflow.

You need the following values for each preserved disk:

- Old `sequenceNum`
- `quantityGB`
- `datastoreName` or `datastoreClusterName`
- `path`

- `format`
- `pciType`

Determine Which Disks Are Claimable

Existing clusters can only claim disks that sit in the tail-contiguous region of the old VM disk layout.

Use the following formula:

```
slot = oldSequenceNum - systemDiskCount - newTemplateDataDiskCount - 1
```

Use these constants when you apply the formula:

- `systemDiskCount = 1`
- `newTemplateDataDiskCount` = the number of non-system disks that remain in the **new** `DCSMachineTemplate`

The computed `slot` must:

- Be greater than or equal to `0`
- Be unique within the same IP entry

If a disk is not in the tail-contiguous region, you must either:

- Move the disks between it and the old template tail into the pool-managed persistent-disk list as well, or
- Accept that the non-claimable disk will still be lost with the old VM

Worked Example

Assume the old template disk order is:

Old Sequence	Old Disk
1	system disk
2	`/var/lib/kubelet`
3	`/var/lib/etcd`
4	`/var/lib/containerd`
5	`/var/cpaas`

If the new template keeps only `system + /var/lib/kubelet + /var/lib/containerd`, then `newTemplateDataDiskCount = 2`.

Disk You Want to Preserve	Old `sequenceNum`	New Template Data Disk Count	Computed `slot`	Claimable
`/var/cpaas`	5	2	`5 - 1 - 2 - 1 = 1`	Yes
`/var/lib/containerd` and `/var/cpaas`	4, 5	1	`4 - 1 - 1 - 1 = 1`, `5 - 1 - 1 - 1 = 2`	Yes
`/var/lib/etcd` only	3	2	`3 - 1 - 2 - 1 = -1`	No

Update the `DCSMachineTemplate`

Edit the currently referenced `DCSMachineTemplate` in place so it no longer declares the disks you want to preserve.

1. Export the current template:

```
kubectl get dcsmachinetemplate <template-name> -n cpaas-system -o yaml
> current-template.yaml
```

2. Update the exported manifest:

   - Keep the system disk.
   - Keep only the template-local disks that should still be recreated with the VM.
   - Remove all disks you want to preserve through the IP pool.
   - If a target disk is only claimable when trailing disks are moved as well, remove those trailing disks from the template too.
   - Keep the original `metadata.name`, because this migration updates the currently referenced template in place.
   - Remove transient metadata fields such as `resourceVersion`, `uid`, `creationTimestamp`, and `managedFields`.

3. Apply the updated template:

```
kubectl apply -f current-template.yaml -n cpaas-system
```

Update the `DCSIpHostnamePool`

Add `persistentDisk` entries to the matching IP slot for every preserved disk.

The spec interacts with the live disk attributes in three ways:

Strict claim match. A mismatch on any of these fields fails the claim and sets `phase=Error` with `lastError`. The controller retries on a slow loop until the spec is corrected:

- `quantityGB` — must match the live disk size exactly
- `datastoreName` or `datastoreClusterName` — must point to the same storage target as the live disk
- `pciType` — must match the live disk PCI type. If omitted, the provider uses the default `VIRTIO`; verify the live disk PCI type before omitting this field, because a non-`VIRTIO` live disk can fail the strict claim match

`isThin` is creation-only. It is sent only when the provider creates a new DCS persistent volume. It is not compared during existing-volume claim and does not convert existing volumes.

Filesystem (affects guest-side initialization, not the claim check):

- `format` — used only when initializing a fresh disk. If the live disk already has a filesystem, the existing format is preserved and `mkfs` is skipped.

Guest-side (applied on replacement VMs only, not part of the claim check):

- `path` — mount path inside the guest
- `mountOptions` — mount options
- `options` — `mkfs` options applied only on first format

For the platform-required `/var/cpaas` disk, move it into the pool-managed layout as part of this migration.

Set `slot` to the value you calculated in the previous section. Do not reuse a fixed example value across different disk layouts.

Example:

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <iphostname-pool-name>
  namespace: cpaas-system
spec:
  pool:
  - ip: "<node-ip>"
    mask: "<mask>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname>"
    machineName: "<machine-name>"
    persistentDisk:
    - slot: <calculated-slot>
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
      pciType: VIRTIO
```

Apply the pool update:

```
kubectl apply -f <updated-pool-file>.yaml -n cpaas-system
```

Trigger the Rolling Upgrade

Before you trigger replacement:

- Confirm `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge = 0`
- Confirm each `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`

These settings are prerequisites for the migration and for later upgrade-time reuse of pool-managed persistent disks.

Then trigger the rollout:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec": {"rolloutAfter": "'"$(date -u +%Y-%m-%dT%H:%M:%SZ)"'"}}'
```

```
kubectl patch machinedeployment <md-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec": {"rolloutAfter": "'"$(date -u +%Y-%m-%dT%H:%M:%SZ)"'"}}'
```

Verify Claim, Detach, Conversion, and Reattach

Watch the management-cluster resources during the rollout:

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machinedeployment <md-name> -n cpaas-system -w
kubectl get machine -n cpaas-system -w
```

Inspect the pool status to confirm the controller has claimed and tracked the disks:

```
kubectl get dcsiphostnamepool <iphostname-pool-name> -n cpaas-system -o y
aml
```

During the transition, each record appears under `status.persistentDiskStatus`. The stable phases to watch for are:

- `phase: Attached` while the old VM still owns the disk
- `phase: Available` after the disk is detached (and converted from a stock volume to an independent persistent volume when needed)
- `phase: Attached` again after the replacement VM reattaches the disk

Transient phases (`Attaching`, `Detaching`) may briefly appear during the corresponding operations; `Deleting` appears when a disk is being permanently removed, for example during pool or cluster cleanup. The full phase set is `Creating`, `Available`, `Attaching`, `Attached`, `Detaching`, `Deleting`, `Error`.

If a disk enters `phase: Error`, inspect `lastError` before retrying.

Limitations and Recovery Notes

- Only tail-contiguous disks are claimable in the existing-cluster migration path.
- The controller only protects disks that are declared in `persistentDisk`. Any undeclared disk still follows VM lifecycle and may be deleted with the old VM.
- This migration changes the ownership model of preserved disks. Do not keep the same disk defined in both `DCSMachineTemplate` and `DCSIpHostnamePool`.
- If you need to preserve `/var/cpaas`, move it into the IP pool as part of this migration instead of leaving it in the template.
- This runbook applies to clusters on ACP `v4.2.1` or later.

Related Topics

- Infrastructure Resources for Huawei DCS
- Upgrade Kubernetes on Huawei DCS

Migrate Existing Huawei Cloud Stack Clusters to Pool-Managed Persistent Disks

Use this guide when you upgrade an existing Huawei Cloud Stack (HCS) cluster from the older `HCSMachineTemplate` data-volume layout to the pool-managed persistent-disk model.

Disks that must survive node replacement are declared in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`. This includes the platform-required `/var/cpaas` disk.

> **INFO**
>
> **Version**
>
> Use this procedure when the cluster runs ACP `v4.3.1` or later.

TOC

- Overview
- Before You Start
- Inspect the Current Disk Layout
- Decide Which Disks to Preserve
- Create New Machine Templates
- Prepare the Rollout Strategy
- Update the Machine Configuration Pool

Trigger Rolling Replacement

Verify the Migration

Failure Handling

Limitations and Recovery Notes

Related Topics

Overview

Older HCS clusters commonly placed `/var/cpaas` in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`. That layout creates data volumes with the ECS. During rolling replacement, the old ECS and its template-owned data volumes may be deleted together.

The pool-managed model moves upgrade-preserved disks into `HCSMachineConfigPool.spec.configs[].persistentDisks[]`. Each persistent disk is bound to a fixed `(hostname, slot)` identity. During rolling replacement, the provider:

1. Claims the existing EVS disk from the old ECS when it matches the pool declaration.
2. Stops the old ECS.
3. Detaches the EVS disk and waits until it is available.
4. Deletes the old ECS.
5. Creates the replacement ECS with the same EVS disk attached before first boot.
6. Starts the replacement ECS, which mounts the existing file system without reformatting it.

Before You Start

Verify all of the following before you begin:

- The HCS provider is installed on the management cluster.
- The workload cluster is healthy and all nodes are `Ready`.
- The cluster uses `HCSMachineConfigPool` to assign fixed hostnames and IP addresses.

- The preserved disks have non-empty mount paths in the old `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.
- The relevant rollout strategies use `maxSurge: 0`.
- You have a maintenance window for one-by-one node replacement.
- You have a verified backup of etcd and platform configuration.

> **WARNING**
>
> Do not declare the same mount path in both `HCSMachineTemplate.spec.template.spec.dataVolumes[]` and `HCSMachineConfigPool.spec.configs[].persistentDisks[]`.

Inspect the Current Disk Layout

Identify the management-cluster objects that control the cluster:

```
kubectl get cluster -n cpaas-system
kubectl get kubeadmcontrolplane -n cpaas-system
kubectl get machinedeployment -n cpaas-system
kubectl get hcsmachinetemplate -n cpaas-system
kubectl get hcsmachineconfigpool -n cpaas-system
kubectl get hcsmachine -n cpaas-system
```

Inspect the current machine templates:

```
kubectl get hcsmachinetemplate <template-name> -n cpaas-system -o yaml
```

Record every `dataVolumes[]` entry that must be preserved. For each disk, record:

Field	Source
`mountPath`	`HCSMachineTemplate.spec.template.spec.dataVolumes[].mountPath`
`size`	`HCSMachineTemplate.spec.template.spec.dataVolumes[].size`

Field	Source
`type`	`HCSMachineTemplate.spec.template.spec.dataVolumes[].type`
`format`	`HCSMachineTemplate.spec.template.spec.dataVolumes[].format`

Decide Which Disks to Preserve

Move only the disks that must survive node replacement to the pool-managed model.

Use the following split:

Path	Recommended declaration
`/var/cpaas`	`HCSMachineConfigPool.spec.configs[].persistentDisks[]`
Monitoring or log data stored under `/var/cpaas`	`HCSMachineConfigPool.spec.configs[].persistentDisks[]`
`/var/lib/kubelet`	`HCSMachineTemplate.spec.template.spec.dataVolumes[]` unless your operational design requires retaining it
`/var/lib/containerd`	`HCSMachineTemplate.spec.template.spec.dataVolumes[]` unless your operational design requires retaining it
`/var/lib/etcd`	Use etcd backup and restore procedures for disaster recovery. Do not rely only on node-local disk retention.

For automatic migration, the provider claims an existing data volume by matching `mountPath`. If a preserved disk has no `mountPath`, automatic claim is not available. Use a supported operational procedure to record the existing EVS `volumeID` in `status.persistentDiskStatus[]`, or migrate the data to a disk with a declared mount path before you trigger replacement.

Create New Machine Templates

Create new `HCSMachineTemplate` resources for the replacement nodes. Do not edit the existing templates in place.

Copy the current template:

```
kubectl get hcsmachinetemplate <current-template-name> -n cpaas-system -o yaml > new-template.yaml
```

Edit `new-template.yaml`:

1. Set `metadata.name` to a new template name.
2. Remove server-generated metadata, such as `resourceVersion`, `uid`, `creationTimestamp`, `managedFields`, and `status`.
3. Leave runtime identity fields unset, including `spec.template.spec.providerID` and `spec.template.spec.serverId`.
4. Remove the preserved paths from `spec.template.spec.dataVolumes[]`.
5. Keep only temporary data volumes that may be recreated with each ECS.
6. Update `spec.template.spec.imageName` and other upgrade fields when this migration is part of a Kubernetes or image upgrade.

For example, after `/var/cpaas` moves to the pool, the template keeps temporary disks only:

hcs-machine-template-temporary-data-volumes.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: <new-template-name>
  namespace: cpaas-system
spec:
  template:
    spec:
      imageName: <vm-image-name>
      flavorName: <instance-flavor>
      availabilityZone: <availability-zone>
      rootVolume:
        type: SSD
        size: 100
      configPoolRef:
        name: <pool-name>
      dataVolumes:
        - size: 100
          type: SSD
          mountPath: /var/lib/containerd
          format: xfs
```

Apply the new template:

```
kubectl apply -f new-template.yaml -n cpaas-system
```

Prepare the Rollout Strategy

Before you update the pool, confirm the rollout strategy for each controller that will use the updated pool. Skip the `MachineDeployment` command if the cluster has no worker pool in this migration.

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system \
  -o jsonpath='{.spec.rolloutStrategy.rollingUpdate.maxSurge}{"\n"}'

kubectl get machinedeployment <md-name> -n cpaas-system \
  -o jsonpath='{.spec.strategy.rollingUpdate.maxSurge}{"\n"}'
```

Each returned value must be `0` for pools that use persistent disks.

If any returned value is not `0`, patch the affected rollout strategy before you update the pool:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec":{"rolloutStrategy":{"type":"RollingUpdate","rollingUpdate":
{"maxSurge":0}}}}'

kubectl patch machinedeployment <md-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec":{"strategy":{"type":"RollingUpdate","rollingUpdate":{"maxSu
rge":0}}}}'
```

Update the Machine Configuration Pool

Edit the `HCSMachineConfigPool` that is referenced by the old and new `HCSMachineTemplate.spec.template.spec.configPoolRef.name`.

Add one `persistentDisks[]` entry under each hostname that must preserve the disk:

hcs-pool-persistent-disk.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <pool-name>
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: <node-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <node-ip>
      persistentDisks:
        - slot: 0
          size: 40
          type: SSD
          mountPath: /var/cpaas
          format: xfs
          mountOptions:
            - defaults
            - noatime
```

Use these rules when you edit the pool:

- Start `slot` at `0` for each hostname.
- Keep slots contiguous for each hostname.
- Set `size`, `type`, `mountPath`, and `format` to match the old data volume that you want to claim.
- Each pool `mountPath` must match the `mountPath` of an existing data volume exactly. Otherwise a new empty disk is created instead of the old one, and the old disk is deleted together with the old ECS.
- Add `cluster.x-k8s.io/cluster-name: <cluster-name>` if the pool does not already have it.
- Keep the same persistent-disk declaration across all nodes that must preserve the same path.

Apply the updated pool only after the replacement template exists, has removed the preserved paths from `dataVolumes[]`, and the rollout strategy uses `maxSurge: 0`:

```
kubectl apply -f hcs-pool-persistent-disk.yaml -n cpaas-system
```

Trigger Rolling Replacement

Apply the updated pool before any old machine is replaced. If the pool does not yet declare `persistentDisks[]` when an old machine is replaced, its old data disk is deleted together with the ECS.

After you apply the updated pool, immediately point the control plane or worker controller to the new template in the same maintenance window. Do not leave a pool that declares `/var/cpaas` persistent disks while the active rollout still points to an old template that also declares `/var/cpaas` in `dataVolumes[]`.

For a control plane migration, point the `KubeadmControlPlane` to the new template:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<new-templ
ate-name>"}}}}'
```

For a worker migration, point the `MachineDeployment` to the new template:

```
kubectl patch machinedeployment <md-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<new-temp
late-name>"}}}}}'
```

If the template reference already points to the target template and you need to force a one-by-one replacement, set `rolloutAfter`:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec": {"rolloutAfter": "'"$(date -u +%Y-%m-%dT%H:%M:%SZ)"'"}}'

kubectl patch machinedeployment <md-name> -n cpaas-system \
  --type='merge' \
  -p='{"spec": {"rolloutAfter": "'"$(date -u +%Y-%m-%dT%H:%M:%SZ)"'"}}'
```

Verify the Migration

Watch the rolling replacement:

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machinedeployment <md-name> -n cpaas-system -w
kubectl get machine -n cpaas-system -w
kubectl get hcsmachine -n cpaas-system -w
```

Inspect the pool status:

```
kubectl get hcsmachineconfigpool <pool-name> -n cpaas-system -o yaml
```

Each migrated disk appears under `status.persistentDiskStatus`:

Phase	Meaning
`Creating`	The provider is creating the EVS volume for this hostname and slot.
`Available`	The EVS volume is ready and is not attached to any ECS.
`Attaching`	The replacement ECS exists; the provider is confirming the attachment.
`Attached`	The EVS disk is attached to the current ECS.
`Detaching`	The disk is being detached before the old ECS is removed.
`Deleting`	Pool deletion is in progress; the EVS volume is being removed.

Phase	Meaning
`Error`	The provider cannot safely continue. Inspect `lastError`.

Confirm the replacement node can read data from the preserved path:

```
kubectl --kubeconfig <workload-kubeconfig> debug node/<node-name> --image
=busybox -- \
  chroot /host sh -c 'mount | grep /var/cpaas && ls -la /var/cpaas'
```

For a stronger data-retention check, write a marker before the rollout and read it after the replacement node becomes `Ready`:

```
kubectl --kubeconfig <workload-kubeconfig> debug node/<node-name> --image
=busybox -- \
  chroot /host sh -c 'cat /var/cpaas/<marker-file>'
```

Failure Handling

Use the pool status to decide the next action when a disk enters `phase: Error`.

Symptom	Likely cause	Action
`lastError` reports a mount-path conflict	The same path exists in both `dataVolumes[]` and `persistentDisks[]`	Remove the preserved path from the new `HCSMachineTemplate`, then retry the rollout.
`lastError` reports size or type mismatch	The pool declaration does not match the existing EVS disk	Correct the pool entry so `size` and `type` match the existing disk.
`lastError` reports an availability zone conflict	The existing EVS disk and the target ECS are in different availability zones	Use an ECS availability zone that matches the EVS disk, or migrate the disk to the target zone before retrying.

Symptom	Likely cause	Action
`lastError` reports that the EVS disk is attached to another ECS	The disk is attached outside the current replacement flow	Detach the disk manually only after you confirm ownership, then let the provider reconcile again.
`lastError` reports that `volumeID` was not found	The tracked EVS disk was deleted	Restore or recreate the expected EVS disk with the correct ownership tags before retrying.

Do not delete the old `HCSMachine` or force-remove finalizers while a persistent disk is in an unresolved error state. The provider blocks deletion to avoid deleting the old ECS before it can safely claim and detach the disk.

Limitations and Recovery Notes

- This procedure applies to clusters that use `HCSMachineConfigPool` with fixed hostnames and IP addresses.
- Pool-managed persistent disks require one-by-one replacement. Keep `maxSurge: 0` for each control plane or worker rollout that uses persistent disks.
- The provider automatically claims existing data volumes by matching a non-empty `mountPath`.
- `dataVolumes[]` that are not declared in `persistentDisks[]` remain template-owned and may be deleted with the old ECS.
- After the provider accepts a persistent disk entry, treat `slot`, `size`, `type`, `format`, and `mountPath` as immutable.
- `mountOptions` can change, but the change takes effect only on a replacement VM.
- Single-control-plane HCS clusters are creation-only topologies and cannot be upgraded. Do not use this rolling migration procedure for a single-control-plane cluster.

Related Topics

- Infrastructure Resources for Huawei Cloud Stack
- Create a Cluster on Huawei Cloud Stack
- Upgrade Kubernetes on Huawei Cloud Stack

Upgrade Validation

Validate each upgraded cluster before closing its maintenance window. In a global DR environment, validate the standby and primary global clusters separately before upgrading workload clusters. For an Alauda OS cluster, run this validation after Upgrade Kubernetes on Alauda OS completes for the `global` cluster or the workload cluster.

TOC

Confirm Distribution Version and CVO Completion

Validate Kubernetes, Nodes, Core, and Aligned Plugins

Validate Installed Extension Availability

Validate the Web UI and In-Use Agnostic Plugins

Confirm Distribution Version and CVO Completion

From the global cluster, inspect the target cluster's `ClusterVersionShadow`:

```
kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{.status.current.version}{"\t"}{.status.desired.version}
{"\n"}{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.reason}
{"\t"}{.message}{"\n"}{end}'

kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{range .status.history[*]}{.version}{"\t"}{.state}{"\t"}{.
startedTime}{"\t"}{.completionTime}{"\n"}{end}'
```

The current and desired Distribution Versions must match the target. `Ready` must be `True`, `Reconciling` must no longer be `True`, and `Stalled` must not be `True`. The newest history entry must show the target version completed successfully.

Validate Kubernetes, Nodes, Core, and Aligned Plugins

From the global-cluster management context, confirm the Core and installed Aligned module state:

```
kubectl get clustermodule <cluster> -o yaml
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster>
```

Then use a `kubectl` context for the upgraded cluster to confirm the Kubernetes server version, node readiness, and Pods:

```
kubectl version
kubectl get nodes
kubectl get pods --all-namespaces \
  | awk '{if ($4 != "Running" && $4 != "Completed")print}' \
  | awk -F'[/ ]+' '{if ($3 != $4)print}'
```

All nodes must be `Ready`; Core and installed Aligned modules must report the target version in a healthy phase; critical Pods must be `Running` or `Completed`.

Validate Installed Extension Availability

From the global cluster, review the cluster plugin catalog when you need to confirm that an upgraded application remains discoverable:

```
kubectl get moduleplugins
```

Upgrade acceptance applies to the Core and Aligned applications actually installed on the upgraded cluster. Use the `ClusterVersionShadow` completion state and the `ModuleInfo` versions from the previous sections as the authoritative result, and confirm that no related Pod reports `ImagePullBackOff` or `ErrImagePull`.

Do not require every entry in `ProductBase.status.artifacts` to be `Ready`. `ProductBase` includes optional or uninstalled catalog packages, including Agnostic applications, which can legitimately remain `Absent` after a successful cluster upgrade.

Validate the Web UI and In-Use Agnostic Plugins

Open the platform Web UI and confirm that login, cluster pages, and Marketplace pages load. Upgrade each in-use Agnostic plugin required for the target Kubernetes version, then verify its UI and primary function.

When upgrading from a release earlier than ACP 4.3, use target-compatible versions of these Agnostic plugins or their pages can fail to open:

- `Alauda DevOps v3`
- `Alauda AI Essentials`
- `Alauda Hyperflux`
- `Alauda Container Platform Data Services Essentials`

Complete any product-specific procedures that apply:

- Upgrade Alauda AI ↗
- Upgrade Alauda DevOps ↗

If the source release is earlier than ACP 4.3 and PKCE hardening has not already been completed, wait until the global tier and all workload clusters reach the target Distribution Version, then follow Disabling the PKCE Plain Method.

If any acceptance check fails, keep the maintenance window open and use Upgrade Troubleshooting.

Upgrade Troubleshooting

Use this page when preflight reports a blocking check, CVO reports `Ready=False`, `Reconciling=True`, or `Stalled=True`, or a Core or Aligned module does not reach its target version.

TOC

- Inspect Upgrade State
- Handle Preflight Blocks
  - Resolve Other Common Preflight Blocks
- Handle Administrator Acknowledgement Gates
- Recover a Missing Aligned Package
  - Aligned Cluster Plugins
  - Aligned Operators
- Diagnose a Module That Does Not Advance
- Collect Evidence Before Escalation

Inspect Upgrade State

Replace `<cluster>` with `global` or the workload cluster name:

```
kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}
{.reason}{"\t"}{.message}{"\n"}{end}'

kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{.status.preflight.observedAt}{"\n"}{range .status.preflig
ht.checks[*]}{.name}{"\t"}{.policy}{"\t"}{.state}{"\t"}{.reason}{"\t"}{.m
essage}{"\n"}{end}'

kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{range .status.history[*]}{.version}{"\t"}{.state}{"\t"}{.
startedTime}{"\t"}{.completionTime}{"\n"}{end}'
```

Resolve the first failing preflight check or the relevant `Ready`, `Reconciling`, or `Stalled` condition before changing the upgrade request.

Handle Preflight Blocks

A `ResourcePatch` records a customization applied to a platform resource. The upgrade refuses to overwrite one unless it is exempted for the target version, so `ResourcePatchUpgradeable` is the most frequent block on a customized environment.

If `ResourcePatchUpgradeable` fails with `reason=UnexemptResourcePatches`, inspect the named `ResourcePatch` and add the required target-version exemption only after reviewing the patch:

```
kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{range .status.preflight.checks[?(@.name=="ResourcePatchUp
gradeable")]}{.state}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'

kubectl get resourcepatches <rp-name> -o yaml
kubectl annotate resourcepatches <rp-name> \
  config.cpaas.io/exempt-for-ver=<target-version> \
  --overwrite
```

The exemption annotation key is `config.cpaas.io/exempt-for-ver`, and its value is the target version. The exemption is matched on the major and minor version, so an exemption

recorded for one patch version also covers the other patch versions of the same minor release.

Exempting a patch tells the upgrade to leave that customization in place. It does not verify that the customization is still valid on the target version — confirm that with the technical support contact who supplied it.

Resolve Other Common Preflight Blocks

List every check that is not `Passed` with the command in Inspect Upgrade State, then resolve each one:

Check	What it validates	Resolution
`KubernetesVersionSupported`	The cluster's Kubernetes version is within the target release's supported range.	Follow the Kubernetes Support Matrix; do not disable the check.
`VersionUpgradePath`	The current patch can reach the requested target through a supported edge.	Use a target in `availableUpdates` or follow an offered intermediate target.
`ClusterRunning`	The cluster is healthy and reconciled.	Inspect `kubectl get clusterview <cluster>` and resolve unhealthy nodes or components.
`DockerRuntimeUnsupported`	No node uses the unsupported Docker runtime.	Migrate affected nodes to `containerd`.
`ClusterModuleStable`, `ModuleInfoStable`	Core and installed Aligned modules are stable.	Follow Verify Module Stability Before You Upgrade.

Do not force an upgrade past a failed check. `VersionUpgradePath`, `KubernetesVersionSupported`, and `AdminAckRequired` gate the supported upgrade path

itself, and an upgrade that bypasses them reaches a state the platform has not validated. If a block cannot be resolved from this page, contact technical support before continuing.

Handle Administrator Acknowledgement Gates

If `AdminAckRequired` fails, inspect the keys supplied by the target release:

```
kubectl -n cpaas-system get configmap admin-gates -o yaml
```

Complete the action described by the applicable gate. For a Kubernetes 1.35 or later node-readiness gate, complete Kubernetes 1.35 or Later Node Readiness on every production node.

The first gate introduced for that requirement, for the ACP 4.4 and Kubernetes 1.35 target, is `ack-4.4-kubernetes-1.35-kernel-update`. Treat it as an example — a later target release can supply a different key, so always read the key out of `admin-gates` rather than reusing this one.

Copy the applicable key from `admin-gates`, record the acknowledgement in `admin-acks`, and rerun preflight:

```
ACK_KEY='<key-from-admin-gates>'
kubectl -n cpaas-system patch configmap admin-acks --type merge \
  -p "{\"data\":{\"${ACK_KEY}\":\"true\"}}"

bash upgrade.sh --preflight
```

Confirm that the check now passes:

```
kubectl -n cpaas-system get cvsh <cluster> \
  -o jsonpath='{range .status.preflight.checks[?(@.name=="AdminAckRequire
d")]}{.state}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

Recover a Missing Aligned Package

An unavailable Aligned package does not change the application's current installed state. Depending on the package type and reconciliation stage, CVO can report the problem through `Ready=False` or `Reconciling=True` rather than `Stalled=True`.

Aligned Cluster Plugins

If a `ClusterVersionShadow` condition contains `required ready ModulePluginConfig for installed platform-aligned component`, identify the target `ProductManifest` and the named `ModulePluginConfig`:

```
kubectl get productmanifest v<target-version> -o yaml

kubectl get modulepluginconfig <modulepluginconfig-name> \
  -o jsonpath='{.spec.image}{"\n"}{range .status.conditions[*]}{.type}
{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

For a ModulePlugin channel, `artifactStatus: Absent` in the target `ProductManifest` means that the matching target `ModulePluginConfig` is not fully Ready. It does not mean that the currently installed application became absent, and it does not by itself prove that the image is missing from the registry.

Use the `ModulePluginConfig` condition and the exact image in `.spec.image` to select the recovery:

Evidence	Recovery
The named `ModulePluginConfig` does not exist	Inspect the target `ProductManifest` conditions and controller reconciliation. A missing object does not by itself prove that the package image is absent.
`ResolveDigestFailed` reports `manifest unknown` or `not found`, or the registry confirms that `.spec.image` is absent	Republish the exact target package by using the applicable method below.
The condition reports `unauthorized`, `denied`, or another authentication error	Correct the registry credentials or repository permissions, then let the existing request retry.

Evidence	Recovery
The condition reports an `x509` or certificate trust error	Correct the registry certificate chain or node trust configuration.
The condition reports DNS, routing, timeout, or connection errors	Restore registry reachability from the component that performs the resolution.
`LoadModulePluginFailed`	Inspect the package contents and `module-plugin.yaml`; publish a corrected package only after confirming the package is invalid.

- For an application in **ACP Upgrade to v4.5**, copy its package into the target Core Package's `plugins/` directory. With the platform built-in Registry, rerun `upgrade.sh --only-sync-image`. With an external registry, rerun both modes in Prepare the Target-Version Payload.
- For another Aligned cluster plugin, publish the target package to the global tier with `violet push`.

Aligned Operators

Aligned operators do not use `ModulePluginConfig`. If an installed operator does not advance, inspect its `Subscription`, target `InstallPlan`, catalog, and `ModuleInfo` on the affected cluster:

```
kubectl get subscription -A
kubectl get installplan -A
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster>
```

If the target operator package is missing, publish it to every cluster where that operator is installed. If only one workload cluster is missing it, push to that cluster with `--clusters "<workload-cluster-name>"`. If the package exists, resolve the `Subscription`, `InstallPlan`, catalog, or `ModuleInfo` condition actually reported by CVO.

Continue observing the existing request after correcting the problem. CVO retries automatically; do not clear or resubmit `desiredUpdate`.

Diagnose a Module That Does Not Advance

```
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster> \
  -o custom-columns='MODULE:.metadata.labels.cpaas\.io/module-name,CURRENT:.status.version,TARGET:.spec.version,NEW:.status.availableVersions[0].version,PHASE:.status.phase'

kubectl get moduleinfo <name> \
  -o jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

Inspect the named resource and related Events. For image-pull failures, verify registry reachability, CA trust, pull credentials, and the referenced manifest from the affected node.

Collect Evidence Before Escalation

Capture the `cvsh` conditions, preflight checks, stages and history, the affected `ModuleInfo` or `ModulePluginConfig`, related Events, and the exact source and target versions. Do not include platform tokens, registry passwords, or Secret contents.