Upgrade

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

Upgrade Path Selection

This section documents upgrades for environments running on a **traditional operating system**. If your environment uses Alauda OS:

- Upgrade the `global` cluster: Upgrading the global Cluster on Immutable Infrastructure ↗
- Upgrade workload clusters: Upgrading Clusters on Immutable Infrastructure ↗

Overview

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

In the Web Console, the upgrade request now follows a two-step flow: review RPCH items first, and then submit the upgrade request in a separate confirmation step.

When moving the platform to a new ACP Distribution Version, the upgrade normally proceeds in two stages:

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 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, or Upgrade Global Clusters in a DR Environment when global DR is enabled.
3. Upgrade Workload Clusters after the global tier reaches the target Distribution Version.
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 immutable 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 v4.4**, 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 traditional-OS clusters, CVO upgrades Kubernetes in place on the existing nodes as part of the same cluster upgrade; nodes are not replaced.	Kubernetes is rolled out by replacing nodes from a new Alauda OS-based image through Cluster API rolling updates, not upgraded in place. The procedure lives in the immutable infrastructure documentation; see Upgrading Clusters on Huawei DCS ↗, Upgrading Clusters on

Work item	Required?	How it is upgraded	If you run immutable OS
			VMware vSphere ↗, or Upgrading Clusters on Huawei Cloud Stack ↗
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 v4.4**, 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 traditional vs. Immutable Infrastructure

On a **traditional operating system** cluster (this page's scope), 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 **Immutable Infrastructure** clusters (DCS, vSphere, HCS), Kube-OVN is driven separately by the IaaS provider through the `cpaas.io/kube-ovn-version` annotation on the workload `Cluster` resource — see Upgrading Clusters on Immutable Infrastructure ↗. That path does not apply to traditional-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 through the two-step RPCH review flow, use ACP CLI, or update `ClusterVersionShadow.spec.desiredUpdate` directly.
- **Workload clusters**: Use the Web Console through the two-step RPCH review flow or use ACP CLI after the target Distribution Version becomes available for the workload cluster.

Related Documentation

- Pre-upgrade
- Upgrade the global cluster
- Upgrade global clusters in a DR environment
- Upgrade workload clusters
- Upgrade validation
- Upgrade troubleshooting
- Global Cluster Disaster Recovery

Pre-Upgrade Preparation

Supported upgrade paths:

At the minor-release policy level, ACP 4.1, 4.2, and 4.3 are within the n-3 direct-upgrade window for the ACP 4.4 release line. This policy does not make every source patch eligible for every target patch.

Upgrade only to a target 4.4.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 4.4.x target is offered, use a later 4.4.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 4.0.x is outside the direct-upgrade window for ACP 4.4. 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, the Kubernetes upgrade requires additional pre-staging of intermediate-version artifacts. See Preparing for Cross-Version Upgrades ↗ before starting the upgrade window.

TOC

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
```

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 4.4 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 v4.4

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 `v4.4.0`. Set `<arch>` to `amd64`, `arm64`, or `hybrid` to match the Core Package. List the scenarios first, and then download **ACP Upgrade to v4.4**:

```
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 v4.4"
```

Confirm that **ACP Upgrade to v4.4** 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 v4.4** 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:

- **Alauda Container Platform Web Console**
- **Alauda Container Platform Web Console Central**
- **Alauda Container Platform Marketplace Web Console**
- **Alauda Container Platform Helm Application Catalog**
- **Alauda Container Platform Observability Essentials**
- **Alauda Container Platform Essentials**
- **Alauda Container Platform Cluster Essentials**
- **Alauda Container Platform GatewayAPI Plugin**
- **Alauda Container Platform Monitoring for Prometheus**
- **Alauda Language Pack for Chinese**
- **Alauda Container Platform Networking for 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.

> **Upgrading from ACP 4.1**
>
> Do not use `violet push` to publish these eleven packages before upgrading an ACP 4.1 environment. In ACP 4.1, publishing these packages can cause auto-install applications such as the

Web Console to deploy before their ACP 4.4 dependencies are available, which can make the Web Console unavailable.

Use only the documented `plugins/` publication path for these packages. The built-in Registry branch uses `upgrade.sh --only-sync-image`; the external Registry branch uses `res/upload.sh all` after the packages are staged.

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 v4.4**. 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 v4.4**, do not publish that duplicate with `violet`. Use the package from **ACP Upgrade to v4.4** 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 v4.4** 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 Path

This page covers the **traditional operating system** path for the `global` cluster. If your `global` cluster uses Alauda OS, the Kubernetes step lives in the Immutable Infrastructure documentation — see Upgrading the global Cluster on Immutable Infrastructure ↗. The Core, Aligned, and Agnostic steps described on this page still apply to immutable-OS clusters; only the way Kubernetes is rolled out differs.

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

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 `plugins/`. For an external	None — images and catalog entries are published, but no running plugin is changed; no downtime.

Phase	When	What happens	Cluster impact
		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.	
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 v4.4**. 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/
```

> **Upgrading from ACP 4.1**
>
> Do not replace this copy step with `violet push` when the source environment runs ACP 4.1. Publishing these packages through `violet` can cause auto-install applications such as the Web Console to deploy before their ACP 4.4 dependencies are available.

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 v4.4**, 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 v4.4**.
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 v4.4**, 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.

To bypass artifact validation when it is known to be redundant, add `--skip-check-artifacts`.

> **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.external}{"\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 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`

If any check does not pass, stop before the maintenance window and follow Upgrade Troubleshooting. Do not disable version, Kubernetes support, or administrator acknowledgement checks to force an unsupported upgrade.

3 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.

4 Request the upgrade

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

Web Console

Use this entry point after the target version becomes available for the cluster. The request follows a two-step flow:

- In **Step 1**, review the RPCH list.
- Click **Acknowledge** to continue to **Step 2**.
- In **Step 2**, review **Current Version** and **Target Version**. The page does not display a plugin list or a warning panel at this stage.
- The target version is determined by the prepared upgrade artifacts and cannot be selected manually in the Web Console.
- Click **Start Upgrade**.
- Confirm the action in the dialog.
- After confirmation, the page shows that the upgrade request has been submitted and the action enters an in-progress state.

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 clu
ster 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>
```

5 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.

Field	Purpose
`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.
`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.

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 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.

Related Documentation

- Overview
- Pre-upgrade
- Upgrade global clusters in a DR environment
- Upgrade workload clusters
- Upgrade validation
- Upgrade troubleshooting
- Upgrading Clusters
- Global Cluster Disaster Recovery

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

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 immutable-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 etcd Synchronizer** 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`:

```yaml
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.toke
n}' | 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-cluste
r-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 etcd Synchronizer**, 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.

Upgrade Workload Clusters

Upgrade Path

This page covers the **traditional operating system** path for workload clusters. For workload clusters that use Alauda OS, the Kubernetes step lives in the Immutable Infrastructure documentation — see Upgrading Clusters on Immutable Infrastructure ↗. The Core, Aligned, and Agnostic steps described on this page still apply to immutable-OS clusters; only the way Kubernetes is rolled out differs.

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

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 two entry points are equivalent; pick the one that fits your operating model.

Web Console

Start the upgrade from the workload cluster in the Web Console. The request follows a two-step flow:

- In **Step 1**, review the RPCH list.
- Click **Acknowledge** to continue to **Step 2**.

- In **Step 2**, review **Current Version** and **Target Version**. The page does not display a plugin list or a warning panel at this stage.
- The target version is the current `global` cluster version and cannot be selected manually in the Web Console.
- Click **Start Upgrade**.
- After the request is submitted, the page shows that the upgrade request has been submitted and the action enters an in-progress state.

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.

Related Documentation

- Overview
- Upgrade the global cluster
- Upgrade validation
- Upgrade troubleshooting
- Upgrading Clusters

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.

TOC

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

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

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
```

Common preflight blocks:

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 disable `VersionUpgradePath`, `KubernetesVersionSupported`, or `AdminAckRequired` to force an unsupported upgrade. If technical support instructs you to disable another check temporarily, disable only the named check in `cpaas-system/cvo-config`.

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.

Copy the applicable key from `admin-gates`, record the acknowledgement, 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
```

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.
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 v4.4**, 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.