升级

本文档将提供有关 ACP 升级的所有信息。

升级路径选择

本节记录在传统操作系统上运行的环境的升级信息。如果你的环境使用 Alauda OS：

- 升级 `global` 集群：在 Immutable Infrastructure 上升级 global 集群
- 升级工作负载集群：在 Immutable Infrastructure 上升级集群

概览

从 ACP 4.3 开始，集群升级采用基于 Cluster Version Operator (CVO) 的工作流。

在 Web Console 中，升级请求现在采用两步流程：先审查 RPCH 项目，然后在单独的确认步骤中提交升级请求。

将平台迁移到新的 ACP Distribution Version 时，升级通常分两个阶段进行：

1. 按照经过验证的 global 集群操作步骤，将 global 层升级到目标 Distribution Version，其中包括制品准备和预检。
2. 当 global 层达到目标 Distribution Version 后，从受支持的 workload cluster 入口升级业务集群，并观察集群状态，直到每个目标集群都达到相同的 Distribution Version。

业务集群只能升级到 global 层已经达到的 Distribution Version。在启用了 **global disaster recovery (DR)** 的环境中，这意味着在业务集群升级到该 Distribution Version 之前，备用和主 global 集群都必须先达到目标 Distribution Version。在升级 global 层之前，请确认每个业务集群都位于 Kubernetes 支持矩阵中该目标 Distribution Version 的 **Compatible Versions** 范围内。此先决条件与顺序规则是分开的。

目录

相关文档

升级工作流

请按以下顺序阅读页面：

1. 升级前准备：确认受支持的路径、集群就绪状态以及完整的目标版本包集合。
2. 升级 global 集群，或者在启用 global DR 时阅读在 DR 环境中升级 global 集群。
3. 在 global 层达到目标 Distribution Version 后，执行升级业务集群。
4. 升级验证：接受 Distribution Version、Kubernetes、Core、Aligned plugins、Pods、制品、CVO 历史记录和 Web UI。
5. 升级故障排查：仅当预检或执行报告阻塞或停滞时使用。

请规划维护时间线，确保准备工作在变更窗口之前完成：

阶段	推荐时间	状态变更
下载并发布制品	在维护窗口之前	不会升级任何正在运行的组件。
运行预检并解决阻塞	在窗口前 1–2 周	只读检查；修复操作可单独更改环境。
升级 global 层	第一个维护窗口	Core、Kubernetes 和已安装的 Aligned plugins 升级到目标版本。
升级业务集群	在 global 层之后	每个选定的业务集群都在各自受控的窗口内升级。
验证并完成后续工作	在关闭每个窗口之前	确认接受结果并完成所需的升级后操作。

关键概念

- **ClusterVersionShadow (`cvsh`)**：用于跟踪当前版本、目标版本、预检结果、执行阶段和历史记录的升级资源。

- **Distribution Version**：集群当前达到的 ACP 版本。业务集群只能升级到 global 层已经达到的 Distribution Version。
- **Preflight**：在升级开始应用目标版本之前运行的验证检查。对于业务集群，请在提交升级请求后，从升级状态输出中查看预检结果。
- 可用升级目标：当前为集群提供的升级版本。在 Web Console 中，当前升级流程的目标版本由平台决定。
- **upgrade.sh**：用于 global 集群升级的准备脚本。使用平台内置 Registry 时，它会同步 Core 制品和存放在 `plugins/` 中的包。使用外部 Registry 时，会跳过同步，并单独上传目标负载。该脚本还会在适当阶段运行预检并部署 CVO。
- **Global DR environment**：同时包含主 global 集群和备用 global 集群的环境。
- 主 **global** 集群：平台访问域当前解析到的 global 集群。
- 备用 **global** 集群：DR 对中的另一个 global 集群。发生故障切换后，这两个集群的角色会互换。

升级范围和顺序

完整的 ACP 升级采用分阶段方式进行。每个集群都在单个维护窗口内升级，但平台是按集群逐个推进的，先 global，后业务集群。

单个集群升级窗口涵盖四类工作：

工作项	必需？	升级方式	如果运行 **immutable OS**
ACP Core（平台本身）	必需	由通过同步升级制品准备的制品驱动的 CVO	控制流相同；Kubernetes 步骤单独处理，见下文
Aligned plugins（与匹配的 ACP 版本一起发布的集群插件和 operator）	对每个已安装插件均为必需	对于通过 **ACP Upgrade to v4.4** 提供的 Aligned 应用，请手动将其包复制到 `plugins/` 中，并通过同步升级制品发布。其他已安装的 Aligned 包请使用 `violet` 发布。随后，CVO 会在集群升级过程	相同

工作项	必需？	升级方式	如果运行 **immutable OS**
		中升级每个已安装的 Aligned plugin。	
Kubernetes	在同一窗口内必需	在传统 OS 集群上，CVO 会在同一次集群升级中就地升级现有节点上的 Kubernetes；不会替换节点。	Kubernetes 通过 Cluster API 滚动更新，使用新的基于 Alauda OS 的镜像替换节点来进行部署，而不是就地升级。相关操作步骤位于不可变基础设施文档中；请参见 在 Huawei DCS 上升级集群↗、在 VMware vSphere 上升级集群↗，或 在 Huawei Cloud Stack 上升级集群↗
Agnostic plugins（独立发布的集群插件和 operator）	可选，按插件分别处理	在集群达到目标 Distribution Version 后，从 **Marketplace > Cluster Plugins** 或通过 operator 自己的工作流升级每个 cluster plugin 或 operator	相同

每个部分何时迁移受以下规则约束：

- 平台要求集群的 Kubernetes 版本必须与其 Core 和 Aligned 版本一起升级；将新的 Core 版本与旧的 Kubernetes 次要版本混用未经过验证。
- 在请求升级之前，请为该集群上已安装的每个 Aligned plugin 准备好目标版本包。对于来自 **ACP Upgrade to v4.4** 的包，请使用同步升级制品中的 `plugins/` 发布路径；对于其他 Aligned 包，请使用 `violet`。已安装的 Aligned plugin 若缺少对应包，会使 CVO 升级停滞。对于集群上未安装的 Aligned plugin，其包不会导致这些插件被安装。
- Cluster plugins 是全局资源。通过 global 层制品发布或 `violet` 工作流提供的包，可以供平台中的每个集群使用；不需要针对每个业务集群重复发布同一个 cluster plugin。
- Operators 是按集群作用域划分的。虽然建议将 operator 推送到 global 层，但仅此并不充分；在业务升级之前，同一个 operator 必须在每个业务集群上都可用。如果你在 global 窗口期间忘记推送某个 operator，可以在业务升级前再次推送作为补救。

- 只有当业务集群超出目标 ACP 版本的 Kubernetes 支持矩阵时，才需要进行业务集群升级。仍处于受支持范围内的业务集群，在 global 层迁移后可以保留现有的 Kubernetes 版本；平台会继续管理它们。

CVO 管理范围

Cluster Version Operator (CVO) 会决定它端到端驱动哪些类型的升级：

生命周期类型	是否由 **CVO** 驱动？	说明
Core	是，CVO 是唯一受支持的路径	Core 升级需要通过 global 集群同步工作流准备的制品；CVO 会消费这些制品。
Aligned plugins	默认是	cluster plugin 或 operator 工作流也可以脱离主流程升级单个 Aligned plugin，例如在不将集群迁移到新的 Distribution Version 的情况下应用关键安全修复。
Agnostic plugins	否	CVO 不管理这些内容。请在集群达到目标 Distribution Version 后，从 **Marketplace > Cluster Plugins** 升级每个 cluster plugin，并通过各自的工作流升级每个 operator。

在真实生产维护窗口中，客户通常会将 Core、Aligned 以及正在使用的 Agnostic plugins 一起迁移。下面的升级页面记录了这一路径：在升级前准备阶段准备所有必需的制品，通过 CVO 推进 Core 和 Aligned，并从 Marketplace 完成正在使用的 Agnostic plugins。

> **INFO**
>
> 传统环境与 **Immutable Infrastructure** 上的 **Kube-OVN**
>
> 在传统操作系统集群上（本页面的范围），Kube-OVN 是 ACP Core 的一部分，会随着其他 Core 内容一起由 CVO 自动升级——operator 不会为每个集群修补 Kube-OVN 版本注解，也不会手动修改 Kube-OVN `AppRelease`。
>
> 在 **Immutable Infrastructure** 集群（DCS、vSphere、HCS）上，Kube-OVN 由 IaaS 提供商通过业务 `Cluster` 资源上的 `cpaas.io/kube-ovn-version` 注解单独驱动——请参见 在 Immutable Infrastructure 上升级集群↗。该路径不适用于传统 OS 集群。

升级入口

- **global** 集群：按照经过验证的基于 `upgrade.sh` 的操作步骤执行。准备阶段完成后，可通过 Web Console 的两步 RPCH 审查流程、ACP CLI，或直接更新 `ClusterVersionShadow.spec.desiredUpdate` 来请求升级。
- 业务集群：在目标 Distribution Version 对业务集群可用后，可通过 Web Console 的两步 RPCH 审查流程或使用 ACP CLI。

相关文档

- 升级前准备
- 升级 global 集群
- 在 DR 环境中升级 global 集群
- 升级业务集群
- 升级验证
- 升级故障排查
- global 集群灾难恢复

升级前准备

支持的升级路径：

在次版本发布策略层面，ACP 4.1、4.2 和 4.3 均位于 ACP 4.4 发布线的 n-3 直接升级窗口内。此策略并不意味着每个源补丁都可以升级到每个目标补丁。

仅升级到平台针对当前源补丁提供的目标 4.4.x 版本。注册目标 `ProductManifest` 后，在 Web Console 或 `ac adm upgrade` 中确认目标已出现在 `availableUpdates` 中，并确认 `VersionUpgradePath` 预检通过。如果未提供任何 4.4.x 目标，请使用包含当前源补丁的更高 4.4.x 目标，或先跟随平台提供的中间目标。不要使用显式升级请求，也不要禁用 `VersionUpgradePath` 来绕过不受支持的源到目标组合。

ACP 4.0.x 超出了 ACP 4.4 的直接升级窗口。请按照平台提供的中间升级链进行升级，并在每一跳都验证 `availableUpdates` 和 `VersionUpgradePath`。

INFO

对于使用 Alauda OS 的集群，ACP Distribution Version 必须遵循 CVO 提供的相同源到目标路径。当目标 ACP 版本比集群当前版本高出一个以上的 Kubernetes minor 版本时，Kubernetes 升级需要额外预置中间版本制品。开始升级窗口之前，请参见 跨版本升级准备↗。

目录

重要说明

- 确保 global cluster 的控制平面节点上的 `/cpaas/minio` 目录至少有 **120 GB** 的可用磁盘空间。
- 在升级 global 层之前，验证所有业务集群是否仍处于 Kubernetes 支持矩阵 中记录的目标 Distribution Version 的 兼容版本 范围内。
- 如果任何业务集群超出了目标兼容范围，请先升级该业务集群，直到其进入该范围，然后再升级 global 层。
- 业务集群只能升级到 global 层已经达到的 Distribution Version。

在升级前验证模块稳定性

集群升级预检要求目标集群上每个已安装的 **Core** 和 **Aligned** 插件模块都处于稳定状态。如果任何此类模块处于安装中、升级中或失败状态，`ModuleInfoStable` 和 `ClusterModuleStable` 预检检查会阻止升级，直到问题解决。常见原因是在请求集群升级时，某个独立插件升级仍在运行。

在请求升级之前，列出目标集群上已安装的模块，并确认每个模块都处于 `Running` 阶段：

```
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'
```

同时确认集群模块本身不处于部署或升级失败状态：

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

每个模块的 `PHASE` 都必须为 `Running`，并且 `deployStatus` 不能是 `Upgrading`、`DeployFailed` 或 `UpgradeFailed`。如果某个模块仍在升级，请等待其达到 `Running`；如果某个模块处于 `Failed` 或 `Blocked` 状态，请先解决问题，再请求集群升级。

Kubernetes 1.35 或更高版本的节点就绪性

如果目标发行版是 ACP 4.4 或更高版本，会将集群升级到 Kubernetes 1.35 或更高版本，并且针对节点就绪性报告了管理员确认门禁，那么在确认该门禁之前，请验证将运行目标集群的每个生产节点。

通过 SSH 或环境中可用的其他管理员节点访问方式登录到每个节点，并直接在节点操作系统上执行以下检查。所使用的账号和凭据取决于集群节点的配置方式。

验证 Linux kernel 版本：

```
uname -r
```

kernel 必须是 `5.8` 或更高版本，并且还必须位于目标发行版的 ACP 支持的操作系统和 kernel 矩阵范围内。

验证节点是否使用 cgroup v2：

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

期望输出为：

```
cgroup2fs
```

为离线环境下载软件包

生产维护窗口通常会在同一个窗口内升级 **ACP Core**、**Aligned plugins**（与同一 ACP 版本一起发布的 cluster plugins 和 operators）以及正在使用的 **Agnostic plugins**（独立发布的 cluster plugins 和 operators）。Core 和 Extension 软件包是分别下载的。请在维护窗口之前准备好这三组软件包，以便制品同步可以尽早完成。

步骤 1：下载 Core Package

从 灵雀云 **Customer Portal** 下载与目标架构匹配的 **ACP 4.4 Core Package**。Core Package 只包含 Core 制品，不包含任何 Aligned 或 Agnostic Extension 软件包。

步骤 2：下载 ACP Upgrade to v4.4

使用 **Violet v4.2.13** 或更高版本直接下载场景软件包。运行 `violet version` 验证版本，并使用 `violet ac login` 登录到 灵雀云 Customer Portal。有关安装和登录说明，请参见 下载软件包。

将 `<target-platform-version>` 设置为 Core Package 的精确 ACP Distribution Version，例如 `v4.4.0`。将 `<arch>` 设置为 `amd64`、`arm64` 或 `hybrid`，以匹配 Core Package。先列出场景，然后下载 **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"
```

下载之前，请确认 **ACP Upgrade to v4.4** 出现在场景列表中。

除了 Violet 之外，也可以在 Customer Portal 中进入 **Marketplace > Batch Download > Install**。在该页面中，从 **Your ACP Version** 选择目标 Core Package 版本，选择匹配的 **Architecture**，再从 **Scenarios** 中选择 **ACP Upgrade to v4.4**，然后下载软件包。

此软件包集合为以下 Aligned 应用提供目标版本软件包。这些软件包与 Core Package 分开：

- **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**

在发布升级制品之前，请将这 11 个软件包手动复制到已解压的 Core Package 的 `plugins/` 目录中。同步升级制品 会介绍内置 Registry 和外部 Registry 的发布分支。

> 从 **ACP 4.1** 升级
>
> 在升级 ACP 4.1 环境之前，不要使用 `violet push` 发布这 11 个软件包。在 ACP 4.1 中，发布这些软件包可能会导致 Web Console 等自动安装应用在其 ACP 4.4 依赖可用之前就开始部署，从而使 Web Console 不可用。
>
> 这些软件包只能使用本文档中说明的 `plugins/` 发布路径。内置 Registry 分支使用 `upgrade.sh --only-sync-image`；外部 Registry 分支则在软件包预置完成后使用 `res/upload.sh all`。

步骤 3：盘点作用范围内每个集群上的其他 Extension

Cluster plugins 和 operators 由不同机制管理。在这一步中，两者都需要处理：

- **Cluster plugins** 是全局资源。推送到 global 层的 cluster plugin 会变成平台内每个集群上都可安装和升级的资源；每个 cluster plugin 只需推送一次。
- **Operators** 作用域限定为每个集群。将 operator 推送到 global 层是推荐的起点，但在升级该集群之前，同一个 operator 软件包必须已在每个业务集群上可用。

使用第 2 步中相同的 `violet` 安装来盘点已安装的 Extension，并发布不属于 **ACP Upgrade to v4.4** 的软件包。更多信息请参见 下载软件包 和 上架软件包。

在任何能够访问 灵雀云 平台端点的机器上，运行 `violet list` 列出当前环境中安装的插件（包括 Aligned 和 Agnostic），并将结果导出到 `./apps.yaml`：

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

优先使用 `--platform-token` 而不是 `--platform-password`，以避免在 shell 历史记录和进程列表（`ps aux`）中暴露密码。

该基于令牌的盘点工作流要求使用 Violet v4.2.13 或更高版本。更早的版本可能会报错 `failed to get user info from platform server`；请先升级 Violet 再继续。

步骤 4：下载其他 **Extension** 软件包

将导出的 `apps.yaml` 文件导入 灵雀云 **Customer Portal**，使软件包列表与集群当前运行的内容保持一致。下载集群正在使用的其他 Aligned 和 Agnostic Extensions 对应的目标版本软件包。

如果结果中包含 **ACP Upgrade to v4.4** 中的 11 个应用之一，不要使用 `violet` 重复发布该软件包。请改为通过 Core Package 的 `plugins/` 目录使用 **ACP Upgrade to v4.4** 中的软件包。仅对其余 Extension 软件包使用 `violet push`。

请下载所有后续升级所需的软件包，确保在升级窗口开始前它们都已可本地获取。下一页 升级 global cluster 说明了所选 Registry 路径如何发布 Core 和那 11 个手动预置的软件包，以及 `violet` 如何发布其他 Extensions。制品发布不会升级任何已在运行的插件，因此请在维护窗口之前完成该步骤。

> ### 必需的 **Aligned** 插件软件包
>
> 正在升级的集群上已安装的每个 Aligned 插件，都必须在发起升级请求之前提供其目标版本软件包。CVO 不要求为未安装的 Aligned 应用提供目标软件包，且同步某个软件包并不会安装当前未安装的应用。
>
> 如果所需软件包不可用，CVO 会阻止进度并持续重试现有请求。对于已安装的 Aligned cluster plugin，`ClusterVersionShadow` 条件会报告缺失或非 Ready 的目标 `ModulePluginConfig`。对于已安装的 Aligned operator，请检查其目标 catalog、`Subscription`、`InstallPlan` 和

`ModuleInfo`。可通过 同步升级制品 中的 `plugins/` 发布路径添加 **ACP Upgrade to v4.4** 中的软件包；其他软件包请使用 `violet` 发布。

WARNING

从 v4.2 开始，我们引入了一个名为 **Alauda Container Platform Log Essentials** 的新插件。如果你之前安装了日志存储插件，在开始升级之前也必须上传该插件。

升级 global 集群

> **升级路径**
>
> 本页涵盖 `global` 集群的传统操作系统路径。如果你的 `global` 集群使用 Alauda OS，则 Kubernetes 步骤位于 Immutable Infrastructure 文档中——参见 在 Immutable Infrastructure 上升级 global 集群。本页描述的 Core、Aligned 和 Agnostic 步骤仍适用于 immutable-OS 集群；只有 Kubernetes 的发布方式不同。

ACP 由一个 **global** 集群和一个或多个业务集群组成。要将平台迁移到新的 ACP Distribution Version，请先将 global 层升级到目标 Distribution Version，然后再将业务集群升级到相同的 Distribution Version。

集群升级使用基于 CVO 的工作流。一个典型的 `global` 集群升级包括工件准备、预检、升级请求和状态观察。

在升级 `global` 集群之前，请确认每个业务集群都处于 Kubernetes Support Matrix 中目标 release 的 **Compatible Versions** 范围内。此先决条件与第三方集群接入范围是分开的。

无论环境是否使用 global DR，此 Compatible Versions 先决条件都适用。global DR 会改变用于升级 global 层的操作步骤，但不会改变在将 global 层升级到目标 Distribution Version 之前，业务集群必须保持在兼容的 Kubernetes 版本范围内这一要求。

global 集群升级遵循本页记录的经过验证的基于 `upgrade.sh` 的操作步骤。你可以从 Web Console 请求 global 集群升级，也可以通过更新 `ClusterVersionShadow.spec.desiredUpdate`，或者使用带有 `--cluster=global` 的 ACP CLI。有关完整的 AC CLI 工作流和输出解释，请参见 Upgrading Clusters。有关完整的命令和标志语法，请参见 AC CLI Administrator Command Reference。

如果环境使用 **global DR**，请按照 在 DR 环境中升级 global 集群 进行操作。否则，请遵循下面的标准工作流。

目录

标准工作流

global 集群升级会沿时间线分阶段进行。大部分工作会在维护窗口之前完成，因此窗口本身会保持较短且可预测：

阶段	时间	发生的操作	集群影响
1. 同步工件	在窗口之前的任何时间	对于平台内置 Registry，`upgrade.sh --only-sync-image` 会上传 Core 工件以及你手动放入 `plugins/` 中的 11 个包。对于外部 Registry，请在运行 `upgrade.sh` 之前手动上传目标 Core 负载。在这两种情况下，`violet` 会发布该窗口所需的其他 Aligned 和 Agnostic 包。	无——镜像和目录条目会被发布，但不会更改正在运行的插件；不会产生停机。

阶段	时间	发生的操作	集群影响
2. 预检	维护窗口前 1–2 周	`upgrade.sh --preflight` 会验证升级就绪状态。在窗口打开之前解决所有阻塞项。	无——只读验证。
3. 升级	在维护窗口期间	`upgrade.sh --skip-sync-image` 会部署或更新 cluster version operator（CVO）；随后会请求并观察升级。	集群将被升级。

阶段 1 和阶段 2 不会更改集群状态——请尽早运行它们，这样维护窗口中就只包含阶段 3。对于较小环境或实验室环境，也可以运行单个 `bash upgrade.sh`，它会同时执行同步和 CVO 部署，但生产窗口通常会将它们分开。

1 同步升级工件

时间： 维护窗口之前的任何时间。此步骤会将工件上传到 Registry，并且不会更改集群状态。

首先，记录已配置的 Registry 地址，并确定平台使用的是内置 Registry 还是外部 Registry：

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

第二个值在平台内置 Registry 时为 `false`，在外部 Registry 时为 `true`。在将目标 Aligned 包复制到 `plugins/` 之后，再选择发布命令。

Core Package 不包含 **ACP Upgrade to v4.4** 中的 Aligned 包。请将 Pre-Upgrade Preparation 中列出的 11 个单独下载的包复制到已解压的 Core Package 的 `plugins/` 目录中。对每个包运行一次以下命令：

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

在同步之前，确认目录中包含全部 11 个包：

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

> **从 ACP 4.1 升级**
>
> 当源环境运行 ACP 4.1 时，不要用 `violet push` 替代此复制步骤。通过 `violet` 发布这些包可能会导致 Web Console 等自动安装应用在其 ACP 4.4 依赖可用之前就开始部署。

按照与 `ProductBase.spec.registry.external` 值匹配的分支发布已暂存的负载。

对于平台内置 Registry（`false`），请在已解压的 Core Package 目录中以仅同步模式运行 `upgrade.sh`：

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

`--only-sync-image` 会上传 Core 镜像以及你手动放入 `plugins/` 中的包，而不会部署 cluster version operator 或更改正在运行的应用。CVO 会在稍后的维护窗口中部署——参见 部署 cluster version operator。

对于外部 Registry（`true`），不要将 `--only-sync-image` 作为发布完成的证明。`upgrade.sh` 会自动跳过镜像同步。请在目标 Core Package 的 `installer/` 目录中，按照 准备目标版本负载 同时运行 `res/upload.sh all` 和 `res/upload.sh necessary`。这两种模式可以按任意顺序运行，但都必须成功。

选定的发布路径会提供基于 CVO 的工作流所需的工件，包括：

类型	内容	目的
Product 镜像	`product-image`	用于在 `ProductManifest` 和 CVO 中解析目标版本和镜像。
CVO 镜像	`cluster-version-operator`	用于部署或更新 cluster version operator。
插件工件	`plugins/*.tgz`	当升级计划需要插件工件时使用。

对于来自 **ACP Upgrade to v4.4** 的 11 个 Aligned 应用，请始终将这些包暂存到 `plugins/` 中。内置 Registry 路径会通过 `upgrade.sh --only-sync-image` 发布它们；外部 Registry 路径会通过 `res/upload.sh all` 发布它们。CVO 只会升级已经安装的应用；为未安装的应用发布包不会安装它。

对于从 `violet list` 清单中下载的其他包，请使用 `violet push`：

包组	发布路径	要求
其他 Aligned 集群插件	对 global 层执行 `violet push`	对每个已安装且不属于 **ACP Upgrade to v4.4** 的插件都需要执行。
其他 Aligned operators	对 global 层以及安装了该 operator 的每个业务集群执行 `violet push`	在升级每个运行该 operator 的集群之前必须完成。
Agnostic 集群插件和 operators	对你计划升级它们的集群执行 `violet push`	对 CVO 可选；在启动它们各自的 Marketplace 或 operator 升级之前必须完成。

```
# 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>"
```

推荐的模式是在 global 窗口期间将每个 operator 包推送到每个 ACP 集群。如果 global 窗口已经过去，而你发现某个业务集群缺少 operator 包，你仍然可以在业务升级之前将其推送到该业务集群——参见 升级业务集群。

WARNING

在请求升级之前，请确保每个已安装在集群上的 Aligned 插件都已可用目标包。来自 **ACP Upgrade to v4.4** 的应用请使用 `upgrade.sh`，其他 Aligned 包请使用 `violet`。同步一个包不会安装当前未安装的应用。对于已安装的 Aligned 集群插件，CVO 需要匹配的目标 `ModulePluginConfig` 处于 Ready；否则，CVO 无法构建升级计划，会通过 `ClusterVersionShadow` 报告错误，并重试现有请求。已安装的 Aligned operator 则需要其目录中匹配的目标 `InstallPlan`。

Registry 行为取决于环境配置方式：

场景	行为
指定了 `--registry`	直接使用所提供的 Registry。
未指定 `--registry`	从 `ProductBase.spec.registry.address` 读取 Registry 地址。
平台内置 Registry	使用 global VIP 重新构建访问地址。
外部 Registry	自动设置 `SKIP_SYNC_IMAGE=true`；目标负载必须已经通过规范的目标版本负载操作步骤 上传。
需要上传镜像但未提供凭据	从 `cpaas-system/registry-admin` Secret 中读取 `username` 和 `password`。

当目标 Registry 不是平台默认值时，请添加 Registry 参数：

参数	目的
`--registry`	指定目标 Registry 地址。
`--username` / `--password`	指定 Registry 凭据。

如果已知工件校验是多余的，可以添加 `--skip-check-artifacts` 来绕过它。

WARNING

在镜像和插件同步完成之前，不要打开维护窗口。

在运行预检之前，请确认 Registry 配置仍然正确：

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

将所选发布命令成功完成的结果，以及 Registry 管理界面或 API 作为阶段 1 的证据。确认预期的目标仓库和 tag 已存在，并将已发布的 Extension 集与导出的已安装清单 `apps.yaml` 进行比较。不要将 `ProductBase.status.artifacts` 作为零输出的目标版本门控：它代表当前目录，且对于可选或未安装的包，合法地可能包含 `Absent` 条目。

2 运行预检

时间： 维护窗口前 1–2 周，这样在窗口打开之前就有时间解决任何阻塞项。

以预检模式运行 `upgrade.sh`：

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

预检是只读的——它会验证升级就绪状态，但不会更改集群状态。

预检返回两部分：

输出	目的
`Summary`	显示总体结果、当前版本、目标版本和目标镜像。
`Checks`	显示每个单独验证项的结果。

默认检查集包括：

- `ResourcePatchUpgradeable`
- `ClusterVersionUpgradeable`
- `AdminAckRequired`

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

如果任何检查未通过，请在维护窗口之前停止，并按照 升级故障排查 处理。不要通过禁用版本、Kubernetes 支持或管理员确认检查来强制执行不受支持的升级。

3 部署 cluster version operator

时间： 维护窗口开始时。

以跳过同步模式运行 `upgrade.sh`。由于工件已在 同步升级工件 中上传，因此会跳过同步：

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

这会部署或更新 cluster version operator（CVO）并完成剩余准备工作。CVO 会在下一步请求升级后驱动 Core 和 Aligned 插件的升级。

命令完成后，在请求升级之前检查目标 `ProductManifest`：

```
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'
```

按工件输出的结果是诊断信息，不要求为空。目标 manifest 可能包含可选、Agnostic 或未安装的条目，这些条目不 Ready 是合理的。将输出与已安装的 Core 和 Aligned 清单进行比较。对于已安装的 Aligned `ModulePlugin`，如果其 channel 不是 Ready，请检查匹配的目标 `ModulePluginConfig`：

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

使用目标 `ProductManifest` 中的组件名称和 channel tag，或者 `ClusterVersionShadow` 错误中报告的名称，来定位 `ModulePluginConfig`。此名称中的组件版本不一定是 ACP Distribution Version。不要仅将 `artifactStatus: Absent` 视为 Registry manifest 缺失的证据；请使用 `ModulePluginConfig` 条件及其 `.spec.image` 来区分 manifest 缺失与认证、CA、网络或包内容失败。

4 请求升级

在部署 cluster version operator 后，通过以下任一入口请求升级。这三个入口是等价的；请选择最符合你的操作模型的方式。

Web Console

当目标版本对集群可用后，使用此入口。请求采用两步流程：

- 在 **Step 1** 中，查看 RPCH 列表。
- 点击 **Acknowledge**，继续到 **Step 2**。
- 在 **Step 2** 中，查看 **Current Version** 和 **Target Version**。此时页面不会显示插件列表或警告面板。

- 目标版本由已准备好的升级工件决定，无法在 Web Console 中手动选择。
- 点击 **Start Upgrade**。
- 在对话框中确认操作。
- 确认后，页面会显示升级请求已提交，操作进入进行中状态。

ACP CLI

当你以脚本方式执行升级，或者当前 ACP 会话已经指向 global 集群时，使用此入口。

```
# 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
```

有关完整的 AC CLI 工作流和输出解释，请参见 Upgrading Clusters。有关完整的命令和标志语法，请参见 AC CLI Administrator Command Reference。

kubectl

当你需要直接操作底层 CVO 资源时，使用此入口。使用目标版本补丁更新 `ClusterVersionShadow.spec.desiredUpdate`。

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

或者直接编辑该资源：

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

最低配置：

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

5 观察执行情况

使用以下命令查看总体状态：

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

重要状态字段：

字段	目的
`status.conditions`	总体状态入口。
`status.preflight.observedAt`	最近一次预检运行时间。
`status.preflight.checks`	每个预检项的详细结果。
`status.current`	当前已应用的版本和镜像。
`status.desired`	正在协调的目标版本和镜像。
`status.history`	升级历史，最新的在前。
`status.stages`	升级阶段及各阶段的执行状态。

首先关注以下条件：

条件	解释
`PreflightReady`	`True` 表示预检已通过。
`Ready`	`True` 表示集群已达到目标版本。
`Reconciling`	`True` 表示升级仍在运行。
`Stalled`	`True` 表示升级被阻塞，需要介入。

如果 `Stalled=True`，请按照 升级故障排查 进行处理。要观察单个插件或 operator 模块，请读取其 `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}'
```

当 `status.phase` 为 `Running` 且 `status.version` 等于目标版本时，模块就已达到目标状态。

6 从 Marketplace 升级 Agnostic 插件

CVO 负责驱动 Core 和 Aligned 插件。**Agnostic** 插件不在 **CVO** 的范围内，必须在集群达到目标 Distribution Version 后单独升级。每个 Agnostic 插件是否需要升级，取决于其自身的 Kubernetes 兼容性——请参阅该插件的 release notes 以确认其与目标 Kubernetes 版本的兼容性。

对于 global 集群中每个正在使用的 Agnostic 插件：

1. 在 Web Console 中切换到 **Administrator** 视图。

2. 导航到 **Marketplace > Cluster Plugins** 以查看 Agnostic 集群插件，或者进入 Agnostic operator 的工作流。
3. 选择目标插件或 operator 并触发升级。Marketplace 升级流程会读取此前通过 `violet` 推送的包。

如果你在升级前跳过了某个 Agnostic 插件的推送，而 Marketplace 没有提供目标版本，请先完成 `violet push` 步骤，然后重新尝试 Marketplace 升级。

验证升级

在集群达到目标版本且已处理正在使用的 Agnostic 插件之后，完成 升级验证。

相关文档

- 概览
- 预升级
- 在 DR 环境中升级 global 集群
- 升级业务集群
- 升级验证
- 升级故障排查
- Upgrading Clusters
- Global Cluster Disaster Recovery

在 DR 环境中升级 global 集群

当环境同时包含主 global 集群和备用 global 集群时，请使用此操作步骤。DR 特定步骤是在 Upgrade the global cluster 中标准 CVO 工作流的基础上附加的。

目录

升级操作步骤

1 在升级前验证 DR 环境

按照常规的 global DR 检查操作步骤，确保 备用 **global** 集群 中的数据与 主 **global** 集群 保持一致。有关 DR 拓扑和同步工作流的背景信息，请参见 Global Cluster Disaster Recovery。

如果检测到不一致，请不要在下一步中卸载 etcd 同步插件，并在继续之前联系技术支持。当备用 global 集群缺少主集群持有的数据时，卸载该插件可能会导致 owner reference 解析错误，并且工作负载集群的 `Machine` 对象——包括 immutable-OS 集群，在这些集群中这会销毁其底层虚拟机——可能会被删除。

在 两个 global 集群上运行以下命令，确保没有处于非运行状态的 `Machine` 节点：

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

如果存在此类节点，请先解决这些问题再继续。

2 从备用 **global** 集群卸载 **etcd** 同步插件

1. 通过 IP 或 VIP 访问 备用 **global** 集群 的 Web Console。
2. 切换到 **Administrator** 视图。
3. 导航到 **Marketplace > Cluster Plugins** 并选择 `global` 集群。
4. 找到 灵雀云容器平台 **etcd Synchronizer** 并将其卸载。
5. 等待卸载完成后再继续。

3 在两个 **global** 集群上同步升级工件

在备用 global 集群和主 global 集群上都完成 Sync upgrade artifacts。在两个集群上使用相同的 registry 类型和完全一致的目标版本负载。对于外部 registry，请为两个 global 集群使用的每个 registry endpoint 完成两种上传模式。

4 升级备用 **global** 集群

如果您将使用备用 global 集群上的 Web Console，请验证备用集群 `ProductBase` 的 `spec.alternativeURLs` 中包含备用 VIP：

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

同步完成后，在 备用 **global** 集群 上执行剩余的标准工作流步骤：

1. 运行预检检查。
2. 部署 cluster version operator。
3. 请求升级。
4. 观察执行过程，直到备用 global 集群达到目标版本。

5 升级主 **global** 集群

在备用 global 集群达到目标版本后，在 主 **global** 集群 上执行剩余的标准工作流步骤：

1. 运行预检检查。
2. 部署 cluster version operator。
3. 请求升级。
4. 观察执行过程，直到主 global 集群达到目标版本。

6 重新安装 **etcd** 同步插件并验证同步状态

在重新安装插件之前，请验证在使用端口转发模式时，端口 `2379` 是否已从两个 global 集群的 VIP 正确转发到各自的控制平面节点。如果备用 global 集群能够直接访问活动 global 集群，则不需要通过负载均衡器进行端口转发。

从活动集群获取活动 global 集群 API server 的 bearer token，然后使用它在备用集群上创建或更新 `etcd-sync-active-cluster-token` Secret。请仅将 token 保留在当前 shell 中，并在创建 Secret 后将其清除。

```
# 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
```

在备用集群上运行：

```
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
```

要重新安装该插件：

1. 通过其 VIP 访问 备用 **global** 集群 的 Web Console，并切换到 **Administrator** 视图。
2. 导航到 **Marketplace > Cluster Plugins** 并选择 `global` 集群。
3. 找到 灵雀云容器平台 **etcd Synchronizer**，单击 **Install**，并配置所需参数。

配置插件时：

- 将 **Active Global Cluster VIP** 设置为活动 global 集群的 VIP。
- 当端口 `2379` 未通过负载均衡器转发时，请正确设置 **Active Global Cluster ETCD Endpoints**。
- 将 **Standby Cluster ETCD Endpoints** 设置为备用集群 etcd 地址。除非本地 etcd service 通过不同的 endpoint 暴露，否则请使用默认值。
- 将 **Active Global Cluster Token Secret** 设置为 `etcd-sync-active-cluster-token`。
- 使用 **Data Check Interval** 的默认值。
- 除非您正在排查问题，否则请保持 **Print detail logs** 处于禁用状态。

在重新安装期间，系统会在 `etcd-sync` Deployment 启动之前运行 `etcd-sync-bootstrap` Job。验证 bootstrap Job 和运行时资源：

```
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
```

验证备用 global 集群上的同步 Pods 和当前 leader：

```
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"
```

如果需要重新同步具有 `ownerReference` 依赖关系的资源，请在出现 `Start Sync update` 后重新创建当前 leader Pod：

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

检查同步状态：

```
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`：这些键存在于主 global 集群中，但在备用集群中缺失。通常在重启当前 `etcd-sync` leader Pod 后可以解决。

> - `LOCAL ETCD surplus keys`：这些键存在于备用 global 集群中，但在主集群中不存在。在删除这些键之前，请与您的运维团队一起审查。
>
> 验证成功后，从插件设置或 release values 中移除任何剩余的旧式纯 token 配置。如果活动集群 token 或远程 `etcd-ca` 之后发生更改，请再次运行插件升级或重新安装工作流，以便 `etcd-sync-bootstrap` 刷新运行时凭据和证书。

在两个 global 集群都以目标版本保持健康且同步已恢复后，请继续执行 Upgrade Workload Clusters 或 Upgrade Validation。

升级业务集群

> **升级路径**
>
> 本页介绍业务集群的传统操作系统路径。对于使用 Alauda OS 的业务集群，Kubernetes 步骤位于 Immutable Infrastructure 文档中——请参见 在 Immutable Infrastructure 上升级集群↗。本页描述的 Core、Aligned 和 Agnostic 步骤仍适用于 immutable-OS 集群；仅 Kubernetes 的部署方式不同。

在 global 层已经达到目标 ACP Distribution Version 之后，可以升级业务集群。如果 global 层已经处于该 Distribution Version，则无需再次升级。

业务集群升级使用与 `global` 集群相同的基于 CVO 的工作流：确认先决条件、运行 preflight 检查、发起升级请求并观察执行过程。该工作流还有两条额外规则：

- 如果平台使用 **global DR**，则在任何业务集群升级到该 Distribution Version 之前，备用和主用 global 集群都必须先达到目标 Distribution Version。
- 目标版本通常只有在 global 层已经达到相同的 Distribution Version 之后，才会对业务集群开放。

业务集群升级与 global 集群升级一样会分阶段进行，但流程更短。业务集群没有自己的 cluster version operator——CVO 运行在 global 集群上，并集中驱动业务集群升级——因此业务集群没有自己的 artifact 同步或 CVO 部署步骤：

阶段	时间	发生的内容
1. Preflight	窗口前 1–2 周	确认先决条件并运行 `upgrade.sh --cluster=<cluster> --preflight`；在窗口开启前解决所有阻塞项。
2. Upgrade	维护窗口期间	发起升级请求并观察执行过程，直到集群达到目标版本。

目录

工作流

1 确认业务集群先决条件

在发起升级请求之前，验证以下内容：

- 业务集群版本低于当前 `global` 集群版本。
- global 层已经达到目标 ACP Distribution Version。
- 在 global DR 环境中，备用和主用 global 集群都已经达到该目标 Distribution Version。

2 确保 **Aligned operator** 包可用

升级 global 集群 中推荐的模式会在 global 升级窗口期间将 operator 包推送到每个 ACP 集群。如果已遵循该建议，则业务集群已经拥有所需的所有 operator 包，可跳过本节。

集群插件不需要按业务集群单独发布。通过 `upgrade.sh` 或 `violet` 向 global 层提供的集群插件，可在平台中的每个集群上安装和升级。

如果 global 窗口没有将 operator 包推送到该业务集群——例如，该集群在 global 窗口期间处于离线状态，或是后来才加入——则在升级请求之前，为此业务集群上已安装的每个 Aligned operator 推送目标版本包：

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

你只需要为业务集群上实际安装的 Aligned operators 推送包。向 global 层提供的 Aligned 集群插件包对所有业务集群都可用，不需要按集群发布。

如果已安装的 Aligned operator 的目标包不可用，CVO 可能会等待匹配的目标 `InstallPlan`，然后在重试现有请求时报告错误。请检查该 operator 的 catalog、`Subscription`、`InstallPlan` 和 `ModuleInfo`；使用上面的命令发布缺失的包，或修正报告中的 catalog 或 registry 问题。继续观察现有请求即可，无需重新提交。未安装在该业务集群上的 Aligned operator 包不会阻止其升级。

3 运行 **preflight** 检查

时间：维护窗口前 1–2 周，以便在窗口开启前有时间解决任何阻塞项。

以 preflight 模式对目标集群运行 `upgrade.sh`：

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

Preflight 为只读操作——它用于验证升级就绪状态，不会更改集群状态。

Preflight 返回两部分内容：

输出	目的
`Summary`	显示总体结果、当前版本、期望版本和期望镜像。
`Checks`	显示每个单独验证项的结果。

如果 preflight 未通过，请不要提交升级请求，直到所有阻塞项都已解决。

有关 preflight 阻塞处理模式，请参见升级故障排查。

4 发起升级请求

时间：在维护窗口期间，并且在 preflight 通过之后。

通过以下入口之一发起升级请求。这两个入口是等价的；请选择适合你的操作模型的方式。

Web Console

在 Web Console 中从业务集群开始升级。该请求遵循两步流程：

- 在 步骤 **1** 中，查看 RPCH 列表。
- 点击 确认 继续到 步骤 **2**。
- 在 步骤 **2** 中，查看 当前版本 和 目标版本。此阶段页面不会显示 plugin 列表或警告面板。
- 目标版本是当前 `global` 集群版本，不能在 Web Console 中手动选择。
- 点击 开始升级。
- 提交请求后，页面会显示升级请求已提交，并且该操作进入进行中状态。

ACP CLI

当你当前的 ACP 会话可以访问目标业务集群时，使用 ACP CLI。为避免歧义，请使用 `--cluster` 显式指定目标集群。ACP CLI 不用于升级 `global` 集群。

```
# 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 会为指定的业务集群提交所请求的目标版本。升级是否可以继续，仍由集群可用的升级目标以及升级控制器的 preflight 检查来决定。

如果你是第一次使用 ACP CLI，请参见ACP CLI 入门。有关会话和集群选择，请参见管理 CLI 配置文件。有关完整的 AC CLI 升级工作流（元数据准备、可用更新、请求模式和状态解读），请参见升级集群。有关完整的命令和标志语法，请参见AC CLI 管理员命令参考。

5 观察进度

提交升级请求后，使用 `cvsh.status` 跟踪当前版本、目标版本、preflight 结果、阶段和历史记录：

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

如果当前 ACP 上下文指向目标业务集群，也可以检查 CLI 报告的状态：

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

有关 `ac adm upgrade status` 输出的详细语义（preflight 和阶段解读），请参见升级集群。

如果升级报告 `Stalled=True`，请使用升级故障排查。缺失的 Aligned operator 包和集群插件包有不同的恢复路径。

上述命令跟踪的是集群级别（Core 和 Aligned）的升级。要观察单个 plugin 或 operator 模块——例如在升级 Agnostic plugin 时——请读取其 `ModuleInfo`：

```
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster> \
  -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'
```

当 `status.phase` 为 `Running` 且 `status.version` 等于目标版本时，说明该模块已达到目标状态。

6 从 Marketplace 升级 Agnostic plugins

CVO 负责驱动 Core 和 Aligned plugins。**Agnostic plugins** 不在 **CVO** 的范围内，并且必须在该业务集群达到目标 Distribution Version 之后单独升级。

对该业务集群中每个正在使用的 Agnostic plugin：

1. 在 Web Console 中切换到 管理员 视图。
2. 导航到 **Marketplace > Cluster Plugins**，用于 Agnostic cluster plugins；或者进入 Agnostic operators 的 operator 工作流。
3. 选择目标 plugin 或 operator 并触发升级。

每个 Agnostic plugin 是否需要在本窗口中升级，取决于其自身的 Kubernetes 兼容性——请查看该 plugin 的 release notes，以了解其与业务集群已达到的新 Kubernetes 版本的兼容性。

如果 Marketplace 未在该业务集群上提供某个 operator 的目标版本，请先为该 operator 按照确保 Aligned operator 包可用处理，然后重试 Marketplace 升级。

验证升级

在业务集群及其正在使用的 Agnostic plugins 达到目标后，完成升级验证。

相关文档

- 概述
- 升级 global 集群
- 升级验证
- 升级故障排查
- 升级集群

升级验证

在关闭维护窗口之前，验证每个已升级的集群。在 global DR 环境中，在升级工作负载集群之前，分别验证备用和主 global 集群。

目录

确认 Distribution Version 和 CVO 完成情况

在 global 集群中，检查目标集群的 `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}'
```

当前和期望的 Distribution Version 必须与目标版本一致。`Ready` 必须为 `True`，`Reconciling` 不应再为 `True`，且 `Stalled` 不能为 `True`。最新的 history 条目必须显示目标版本已成功完成。

验证 Kubernetes、节点、Core 和已对齐插件

在 global-cluster 管理上下文中，确认 Core 和已安装的 Aligned 模块状态：

```
kubectl get clustermodule <cluster> -o yaml
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster>
```

然后使用升级后集群的 `kubectl` 上下文，确认 Kubernetes server 版本、节点就绪状态以及 Pods：

```
kubectl version
kubectl get nodes
kubectl get pods --all-namespaces \
  | awk '{if ($4 != "Running" && $4 != "Completed")print}' \
  | awk -F'[/ ]+' '{if ($3 != $4)print}'
```

所有节点都必须处于 `Ready`；Core 和已安装的 Aligned 模块必须在健康阶段报告目标版本；关键 Pods 必须处于 `Running` 或 `Completed` 状态。

验证已安装扩展的可用性

在 global 集群中，当你需要确认升级后的原生应用仍然可被发现时，请查看 cluster plugin 目录：

```
kubectl get moduleplugins
```

升级验收适用于实际安装在升级后集群上的 Core 和 Aligned 原生应用。请使用上一节中的 `ClusterVersionShadow` 完成状态和 `ModuleInfo` 版本作为权威结果，并确认没有相关 Pod 报告 `ImagePullBackOff` 或 `ErrImagePull`。

不要要求 `ProductBase.status.artifacts` 中的每一项都必须为 `Ready`。`ProductBase` 包含可选或未安装的目录包，包括 Agnostic 应用，这些项目在集群升级成功后合理地可能仍然保持 `Absent`。

验证 Web UI 和正在使用的 Agnostic 插件

打开平台 Web UI，确认登录、集群页面和 Marketplace 页面可以正常加载。升级目标 Kubernetes 版本所需的每个正在使用的 Agnostic 插件，然后验证其 UI 和主要功能。

当从早于 ACP 4.3 的版本升级时，请使用与目标版本兼容的这些 Agnostic 插件，否则其页面可能无法打开：

- `Alauda DevOps v3`
- `Alauda AI Essentials`
- `Alauda Hyperflux`
- `Alauda Container Platform Data Services Essentials`

完成所有适用的产品特定流程：

- 升级 Alauda AI
- 升级 Alauda DevOps

如果源版本早于 ACP 4.3，且尚未完成 PKCE 加固，请等待 global 层和所有工作负载集群都达到目标 Distribution Version，然后按照 禁用 PKCE Plain 方法 进行操作。

如果任何验收检查失败，请保持维护窗口开启，并使用 升级故障排查。

升级故障排查

当 preflight 报告阻塞性检查，CVO 报告 `Ready=False`、`Reconciling=True` 或 `Stalled=True`，或者 Core 或 Aligned 模块未达到其目标版本时，请使用本页。

目录

检查升级状态

将 `<cluster>` 替换为 `global` 或业务集群名称：

```
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}'
```

在更改升级请求之前，先解决第一个失败的 preflight 检查，或相关的 `Ready`、`Reconciling`、`Stalled` 条件。

处理预检阻塞

如果 `ResourcePatchUpgradeable` 因 `reason=UnexemptResourcePatches` 失败，请检查命名的 `ResourcePatch`，并且仅在审查该 patch 之后，添加所需的目标版本豁免：

```
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
```

常见的 preflight 阻塞：

检查	验证内容	解决方法
`KubernetesVersionSupported`	集群的 Kubernetes 版本位于目标发行版支持的范围内。	按照 Kubernetes Support Matrix 操作；不要禁用该检查。

检查	验证内容	解决方法
`VersionUpgradePath`	当前 patch 可以通过受支持的边路径到达请求的目标版本。	使用 `availableUpdates` 中的目标，或遵循提供的中间目标。
`ClusterRunning`	集群处于健康且已协调状态。	检查 `kubectl get clusterview <cluster>`，并解决不健康的节点或组件。
`DockerRuntimeUnsupported`	没有节点使用不受支持的 Docker runtime。	将受影响的节点迁移到 `containerd`。
`ClusterModuleStable`，`ModuleInfoStable`	Core 和已安装的 Aligned 模块处于稳定状态。	按照 在升级前验证模块稳定性 操作。

不要禁用 `VersionUpgradePath`、`KubernetesVersionSupported` 或 `AdminAckRequired` 来强制执行不受支持的升级。如果技术支持指示你临时禁用其他检查，则仅在 `cpaas-system/cvo-config` 中禁用所指定的检查。

处理管理员确认门控

如果 `AdminAckRequired` 失败，请检查目标发行版提供的键：

```
kubectl -n cpaas-system get configmap admin-gates -o yaml
```

完成适用门控所描述的操作。对于 Kubernetes 1.35 或更高版本的节点就绪门控，请在每个生产节点上完成 Kubernetes 1.35 或更高版本节点就绪。

从 `admin-gates` 中复制适用的键，记录确认，然后重新运行 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
```

恢复缺失的 Aligned 包

不可用的 Aligned 包不会更改应用当前已安装的状态。根据包类型和协调阶段，CVO 可能会通过 `Ready=False` 或 `Reconciling=True` 而不是 `Stalled=True` 来报告问题。

Aligned 集群插件

如果某个 `ClusterVersionShadow` 条件包含 `required ready ModulePluginConfig for installed platform-aligned component`，请识别目标 `ProductManifest` 以及命名的 `ModulePluginConfig`：

```
kubectl get productmanifest v<target-version> -o yaml

kubectl get moduleplugin config <moduleplugin config-name> \
  -o jsonpath='{.spec.image}{"\n"}{range .status.conditions[*]}{.type}
{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

对于 ModulePlugin channel，目标 `ProductManifest` 中的 `artifactStatus: Absent` 表示匹配的目标 `ModulePluginConfig` 尚未完全就绪。这并不表示当前已安装的应用已变为缺失，也不能仅凭这一点证明 registry 中缺少该 image。

使用 `ModulePluginConfig` 条件以及 `.spec.image` 中的确切 image 来选择恢复方式：

依据	恢复方法
命名的 `ModulePluginConfig` 不存在	检查目标 `ProductManifest` 条件和 controller 的协调情况。对象缺失本身并不能证明包 image 不存在。
`ResolveDigestFailed` 报告 `manifest unknown` 或 `not found`，或者 registry 确认 `.spec.image` 不存在	使用下面适用的方法重新发布确切的目标包。
条件报告 `unauthorized`、`denied` 或其他认证错误	修正 registry 凭据或仓库权限，然后让现有请求重试。
条件报告 `x509` 或证书信任错误	修正 registry 证书链或节点信任配置。

依据	恢复方法
条件报告 DNS、路由、超时或连接错误	恢复执行解析的组件到 registry 的可达性。
`LoadModulePluginFailed`	检查包内容和 `module-plugin.yaml`；仅在确认包无效后，发布已修正的包。

- 对于 **ACP Upgrade to v4.4** 中的应用，将其 package 复制到目标 Core Package 的 `plugins/` 目录中。对于内置 Registry，重新运行 `upgrade.sh --only-sync-image`。对于外部 registry，重新运行 准备目标版本有效负载 中的两种模式。
- 对于其他 Aligned 集群插件，使用 `violet push` 将目标包发布到 global 层。

Aligned operators

Aligned operators 不使用 `ModulePluginConfig`。如果已安装的 operator 未继续升级，请检查受影响集群上的 `Subscription`、目标 `InstallPlan`、catalog 和 `ModuleInfo`：

```
kubectl get subscription -A
kubectl get installplan -A
kubectl get moduleinfo -l cpaas.io/cluster-name=<cluster>
```

如果目标 operator 包缺失，请将其发布到安装了该 operator 的每个集群。如果只有一个业务集群缺失，则使用 `--clusters "<workload-cluster-name>"` 将其推送到该集群。如果包已存在，请解决 CVO 实际报告的 `Subscription`、`InstallPlan`、catalog 或 `ModuleInfo` 条件。

在修正问题后，继续观察现有请求。CVO 会自动重试；不要清除或重新提交 `desiredUpdate`。

诊断未推进的模块

```
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,NEW:.status.availableVersions[0].v
ersion,PHASE:.status.phase'

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

检查命名资源及相关 Events。对于 image-pull 失败，请验证 registry 可达性、CA 信任、pull 凭据，以及受影响节点上引用的 manifest。

在升级升级前收集证据

请捕获 `cvsh` 条件、preflight 检查、阶段和历史记录、受影响的 `ModuleInfo` 或 `ModulePluginConfig`、相关 Events，以及精确的源版本和目标版本。不要包含平台 token、registry 密码或 Secret 内容。