安装

安装路径选择

根据运行 `global` 集群节点的操作系统选择安装路径：

- 在 Alauda OS 上安装：节点运行 Alauda OS，并通过基础设施提供商进行配置。
- 在非 Alauda OS 上安装：节点运行 非 **Alauda OS**，例如 Ubuntu 或 RHEL。

概览

概览

规划

规划

安装准备

下载 准备外部平台镜像仓库 通信矩阵

在 **Alauda OS** 上安装

安装 **Global** 集群灾难恢复

在非 **Alauda OS** 上安装

前置要求 节点预处理 安装

安装程序参数 **Global** 集群灾难恢复 安装故障排查

验证

验证

后续步骤

后续步骤

概览

使用 Install 部分规划并完成 ACP 4.5 的安装。

安装将在 `global` 集群上部署 ACP Core 和所需的 Aligned Extensions。Core 提供平台管理平面、集群管理、用户和角色以及 Extension 管理框架。所需的 Aligned Extensions 提供 Web Console 和其他面向平台的功能。有关平台架构以及 `global` 集群和业务集群的定义，请参阅架构 和 术语表。

Core Package 和所需的 Aligned Extension 软件包需要分别下载。启动安装程序前，将 下载 中列出的八个必需 Extension 软件包复制到解压后的 Core Package 的 `plugins/` 目录中。其他 Extension 将在平台安装完成后通过 扩展 工作流进行安装。在 Alauda OS 上，安装还会将 Cluster Bootstrap Provider 和基础设施提供商软件包上传到临时引导集群；请参阅安装 global 集群。

INFO

ACP Core 安装不支持将 `global` 集群直接安装到现有 Kubernetes 环境中。在非 Alauda OS 环境中，开始安装前请准备满足硬件、网络、存储和 OS 要求的新节点。在 Alauda OS 上，安装会通过基础设施提供商配置 `global` 集群节点；在裸金属环境中，您需要提供主机。

目录

安装范围

选择安装路径

安装工作流

相关文档

安装范围

安装提供平台管理平面和所需的 Aligned Extensions。验证安装完成后，您可以根据平台场景创建或连接业务集群，并安装其他 Extension。

以下工作不属于此安装流程：

- 创建业务集群。
- 连接现有 Kubernetes 集群。
- 上传和安装八个必需的 Aligned Extensions 之外的其他 Extension。
- 安装可选的 Operator 或 Cluster Plugin。
- 配置可选的平台功能，例如存储、监控、备份、注册表或身份集成。

选择安装路径

根据 `global` 集群节点运行的操作系统选择路径：

节点操作系统	安装路径
Alauda OS，通过 Huawei DCS、VMware vSphere、Huawei Cloud Stack 或裸金属上的基础设施提供商进行配置。	在 Alauda OS 上安装
非 Alauda OS，例如 Ubuntu 或 RHEL，运行于您准备的节点上。	在非 Alauda OS 上安装

安装工作流

安装遵循以下高级流程：

1. 规划。选择部署架构，并确认安装后无法安全更改的决策。如有需要，请在准备资源前阅读适用于所选路径的 Global Cluster Disaster Recovery 页面：Alauda OS 或非 Alauda OS。DR 会影响地址、证书、负载均衡、注册表选择以及两次安装的一致性。
2. 准备安装。完成适用于所选路径的通用准备工作：
   - 下载。下载匹配的 Core Package 和所需的 Aligned Extension 软件包。
   - 如果选择使用外部平台镜像注册表，请准备外部平台镜像注册表，并上传完整的目标版本负载。
   - 如果在非 Alauda OS 节点上启用了主机防火墙，请使用通信矩阵构建允许规则。
3. 通过以下两种路径之一安装 `global` 集群：
   - 在 Alauda OS 上安装。在引导节点上运行临时引导集群，上传提供商软件包，应用特定于提供商的 `global` 清单，并触发平台安装。此路径还涵盖可选的主用和备用部署，以及移除引导集群。
   - 在非 Alauda OS 上安装。满足先决条件和节点预处理中的硬件、存储、网络和节点要求。然后解压 Core Package，暂存所需软件包，启动安装程序并完成 Web UI 工作流。查看安装程序中输入的值时，请使用安装程序参数。如果安装程序停滞或验证失败，请参阅安装故障排除。
4. 验证。只有在 Web UI、Core、所需的 Aligned Extensions、镜像源、制品和 Pod 均运行正常后，才接受安装。
5. 后续步骤。安装 Product Docs 插件，并继续进行特定于场景的配置。

Customer Portal 是 ACP 安装软件包的官方来源。有关下载要求和软件包架构选项，请参阅下载。

> **Separate Downloads**
>
> Core Package 不包含所需的 Aligned Extension 软件包。请单独下载，并在启动安装程序之前，手动将下载中列出的八个软件包复制到解压后的 Core Package 的 `plugins/` 目录中。

相关文档

- 架构

- 术语表
- Kubernetes 支持矩阵
- 磁盘配置要求
- 扩展

规划

开始安装前，请选择部署架构，并确认会影响后续操作的安装设置。安装路径包括在 Alauda OS 上安装和在非 Alauda OS 上安装。如果某项决策仅适用于其中一种路径，本页面会特别说明。

> **INFO**
>
> ACP 安装会在新节点上部署 `global` 集群：这些节点可以是在非 Alauda OS 上准备的节点，也可以是安装通过基础设施提供商配置的 Alauda OS 节点。不支持将 `global` 集群直接安装到现有 Kubernetes 环境中。

目录

选择部署架构

确认安装时的决策

安装前必读

选择部署架构

请选择与您的业务场景、管理模型、工作负载放置、资源规划和隔离要求相匹配的架构。

架构	适用场景	规划说明
多集群	您需要使用一个 `global` 集群管理多个工作负载集群。	避免在 `global` 集群上运行非平台工作负载。根据预期管理的工作负载集群数量规划资源。
单集群	您有意使用一个集群承载平台组件和应用工作负载。	当符合业务场景时，单集群支持生产环境使用。由于 `global` 集群还会运行应用工作负载，因此请在安装前规划资源以及平台与工作负载之间的隔离。
单节点	您需要测试环境或概念验证环境。	不要将单节点用于生产环境。在 Alauda OS 上，是否可以使用单个控制平面节点取决于提供商；请参阅以下列表。

在 Alauda OS 上，单个控制平面节点是否可用取决于提供商：

- **VMware vSphere**：不支持。`KubeadmControlPlane` 至少需要三个副本。请参阅提供商要求。
- 裸金属：不支持。请至少配置三个控制平面副本。请参阅第 3 步：创建控制平面集群资源。
- **Huawei Cloud Stack**：仅支持创建，不支持升级。请参阅在 Huawei Cloud Stack 上创建集群。

有关非 Alauda OS 上的资源要求和容量规划指导，请参阅先决条件。有关 Alauda OS，请参阅通用先决条件；有关 `global` 集群容量规划，请参阅评估 Global 集群资源。Bootstrap 节点要求仅适用于临时 Bootstrap 节点。

确认安装时的决策

开始安装前，请确认以下决策。最后两列链接到相应路径中记录各项决策的页面。

决策	需确认的内容	非 **Alauda OS**	**Alauda OS**
安装包	选择 x86、ARM 或混合架构的 Core Package，并下载适用于相同 ACP 版本和架构的所需 Aligned Extension 软件包。在 Alauda OS 上，还需准备提供商	下载	下载 和通用先决条件

决策	需确认的内容	非 **Alauda OS**	**Alauda OS**
	软件包以及 Alauda OS 镜像或 VM 模板。		
集群网络协议	在非 Alauda OS 上，请在启动安装程序前选择 IPv4、IPv6 或双栈。在 Alauda OS 上，请规划 Pod 和 Service CIDR 范围。	安装	通用先决条件
集群 Endpoint 和平台访问地址	准备集群组件、管理员和用户将使用的访问地址。	先决条件 和安装程序参数	通用先决条件 和规划控制平面 Endpoint
负载均衡	决定使用外部负载均衡器还是自建 VIP。在 Alauda OS 上，可用的 Endpoint 模式取决于提供商。	先决条件 和 LoadBalancer 转发规则	规划控制平面 Endpoint
证书	决定使用安装程序生成的自签名证书，还是您提供的受信任证书。	先决条件 和安装程序参数	触发平台安装
平台镜像源	决定使用平台内置 Registry，还是客户管理的外部平台镜像 Registry。对于新的生产部署，如果外部 Registry 满足文档规定的可用性和生命周期要求，请使用外部 Registry。	准备外部平台镜像 Registry 和安装程序参数	准备外部平台镜像 Registry 和通用先决条件
节点	确认节点名称、节点角色以及是否需要进行平台节点隔离。	节点预处理 和安装程序参数	通用先决条件
Global 集群灾难恢复	决定您的环境是否需要主用和备用 `global` 集群。	Global 集群灾难恢复	Global 集群灾难恢复 和可选灾难恢复部署

如果您计划使用 Global 集群灾难恢复，请在安装 ACP Core 前，阅读适用于您安装路径的 Global 集群灾难恢复页面。灾难恢复规划会影响域名、VIP、证书、负载均衡器规则、镜像 Registry 选择、软件包架构以及安装参数的一致性。

安装前必读

开始安装前，请将以下页面作为准确信息来源。

适用于两种安装路径：

- 磁盘配置要求：确认磁盘容量和性能要求。
- 下载：分别下载 Core Package 和 `Common ACP Extensions`，并确认二者的版本和架构一致。
- 准备外部平台镜像 Registry：如果选择使用外部 Registry，请在启动安装程序前上传目标版本的完整 Core 和 Extension 负载。

适用于非 Alauda OS：

- 先决条件：准备资源、网络和存储所需条件。
- 节点预处理：在安装前检查并准备每个节点。
- Global 集群灾难恢复：如果需要灾难恢复，请规划主用和备用 `global` 集群。
- 安装：上传 Core Package，暂存所需软件包，并启动安装程序。
- 安装程序参数：在 Web UI 中配置安装选项。

适用于 Alauda OS：

- 安装 global 集群：满足通用先决条件和引导节点要求，然后通过适用于您提供商的引导集群安装 `global` 集群。
- Global 集群灾难恢复：如果需要灾难恢复，请规划主用和备用 `global` 集群。

安装准备

下载 准备外部平台镜像仓库 通信矩阵

下载

安装之前，请从 灵雀云 **Customer Portal** 中分别下载 核心软件包 和所需的 Aligned Extension 软件包。

目录

关于 Customer Portal

下载安装软件包

使用 Violet CLI 下载（推荐）

从 Customer Portal 下载

所需的匹配扩展

选择软件包架构

从单架构迁移到 hybrid

关于 **Customer Portal**

灵雀云 **Customer Portal** 是 ACP 产品资源的官方服务与交付门户。您可以使用它获取经过验证的软件包以及用于安装、升级和维护活动的官方技术指导。

Customer Portal 提供以下资源：

- 产品下载内容，包括安装、升级和扩展软件包。
- 产品文档和技术指导。

- 用于提交、跟踪和管理支持请求的支持资源。
- 用于扩展平台功能的 Marketplace 软件包。
- 用于部署和维护规划的许可证相关资源。

使用授权账户下载环境所需的软件包。如果您没有注册账户，请联系技术支持。

对于客户交付环境和生产环境，请将 Customer Portal 上发布的软件包版本和文档作为官方部署和维护基线。

下载安装软件包

1. 下载与目标架构匹配的 ACP 4.5 核心软件包。
2. 下载与相同 ACP 版本和架构对应的 通用 **ACP** 扩展。推荐使用 Violet CLI，因为它可以直接定位并下载场景软件包。
3. 如果在 Alauda OS 上安装 `global` 集群，还需下载与节点架构和提供商匹配的 Alauda OS 镜像。请参阅下载 Alauda OS。

使用 **Violet CLI** 下载（推荐）

安装 **Violet v4.2.13** 或更高版本，运行 `violet version` 验证版本，并使用 `violet ac login` 登录 灵雀云 Customer Portal。有关安装和登录说明，请参阅下载软件包。

将 `<target-platform-version>` 设置为核心软件包的确切 ACP Distribution Version，例如 `v4.5.0`。将 `<arch>` 设置为 `amd64`、`arm64` 或 `hybrid`，使其与核心软件包匹配。先列出场景，然后下载 通用 **ACP** 扩展：

```
violet ac scenarios --arch=<arch> --platformVersion=<target-platform-vers
ion>
violet ac download-app --arch=<arch> --platformVersion=<target-platform-v
ersion> --scenario="Common ACP Extensions"
```

下载前，请确认场景列表中显示 通用 **ACP** 扩展。

从 **Customer Portal** 下载

您也可以不使用 Violet，而是在 Customer Portal 中转到 **Marketplace > Batch Download > Install**。在该页面上，从 **Your ACP Version** 中选择核心软件包版本，选择匹配的 **Architecture**，从 **Scenarios** 中选择 通用 **ACP** 扩展，然后下载软件包。

> **Separate Packages**
>
> 核心软件包仅包含 Core 构件，不包含通过 通用 **ACP** 扩展下载的任何扩展软件包。提取核心软件包后，必须先将所需的扩展软件包手动复制到其 `plugins/` 目录中，然后再启动安装程序。

所需的匹配扩展

将以下应用的软件包复制到已提取 Core Package 的 `plugins/` 目录中：

- **Alauda Container Platform Web Console**
- **Alauda Container Platform Web Console Central**
- **Alauda Container Platform Marketplace Web Console**
- **Alauda Container Platform Application Catalog**
- **Alauda Container Platform Observability Essentials**
- **Alauda Container Platform Essentials**
- **Alauda Container Platform Cluster Essentials**
- **Alauda Container Platform GatewayAPI Plugin**

不要将 **Common ACP Extensions** 中的其他软件包复制到已提取 Core Package 的 `plugins/` 目录中。请按照 扩展 中的说明，在平台安装完成后安装这些扩展。在 Alauda OS 上，安装还会使用 **Common ACP Extensions** 下列出的 Cluster Bootstrap Provider 和基础设施 provider 软件包；安装 global 集群 会将这些软件包上传到 bootstrap 集群，而不是上传到 `plugins/`。

选择软件包架构

软件包适用于 **x86**、**ARM** 和 **hybrid** 架构。hybrid 软件包包含 x86 和 ARM 两种架构的镜像，因此软件包更大。请选择与您的环境最匹配的软件包，并使用与 Core Package 架构匹配的扩

展软件包。

从单架构迁移到 **hybrid**

如果最初使用 x86 或 ARM Core Package 安装，但之后需要支持另一种架构，则必须重新下载 **hybrid Core Package** 并执行以下步骤：

1. 将新下载的 hybrid Core Package 上传到 global 集群的任意控制平面节点。
2. 提取软件包，并使用其中包含的 `upgrade.sh` 脚本将多架构镜像同步到镜像仓库：

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

3. 脚本完成后，检查 `cluster.platform.tkestack.io` 资源，确认是否存在标签 `cpaas.io/node-arch-constraint`。如果存在，则必须将其删除：

   ```
   kubectl get cluster.platform.tkestack.io global -oyaml | grep cpaas.io/node-arch-constraint
   # If there is output, edit the resource to remove the label; otherwise, you can skip this step.
   kubectl edit cluster.platform.tkestack.io global   ### Edit the labels field and delete cpaas.io/node-arch-constraint
   ```

准备外部平台镜像仓库

对于新的生产部署，您可以使用由客户管理的外部镜像仓库，作为 ACP Core、托管集群和 Extensions 的中央镜像源。启动安装程序之前，仓库中必须包含完整的目标版本负载。

> **安装前准备仓库**
>
> 选择外部仓库后，安装程序不会部署平台内置 Registry，也不会将缺失的镜像或软件包复制到外部仓库。在运行 `setup.sh` 之前，请完成本页所述的两种上传方式。

目录

了解仓库角色

Registry Requirements

Required Capabilities and Access

Production Recommendations

准备目标版本负载

故障排除

下一步

了解仓库角色

以下仓库角色彼此独立：

角色	用途	生命周期负责人
平台内置 Registry	在使用 **Platform Deployment** 作为镜像仓库安装 `global` 集群时，存储平台 Core 镜像和 Extension 工件。	ACP 部署该服务；平台管理员负责管理其容量和备份。
外部平台镜像仓库	取代平台内置 Registry，作为 Core、托管集群和 Extensions 的镜像源。	您负责提供和运营该仓库，并上传所有必需的目标版本负载。

本页配置供 ACP 组件和集群使用的平台镜像源。本页不会配置应用工作负载的镜像服务。

现有环境和非生产评估仍支持使用平台内置 Registry。本页不介绍从内置 Registry 在线迁移到外部仓库的方法。

Registry Requirements

Required Capabilities and Access

上传目标版本负载之前，请确认该仓库：

- 支持 Core Package 工具执行的标准容器镜像推送和拉取操作。
- 使用格式为 `<host-or-IP>[:port]` 的地址，例如 `registry.example.com:5000`。请勿在安装程序仓库地址中包含 `http://` 或 `https://`。
- 可通过所需的防火墙、代理和网络路由，从安装节点以及拉取平台镜像的每个集群节点访问。
- 提供可创建或更新所需仓库路径并推送镜像的上传凭据。传递给 `setup.sh` 的凭据必须具有拉取权限。上传凭据和拉取凭据可以属于不同的账户；向 `setup.sh` 进行身份验证时，请同时提供其用户名和密码。
- 具有足够的可用存储空间，用于存放您上传的目标版本 Core 和 Extension 负载。

Production Recommendations

对于生产仓库：

- 使用 HTTPS，并为仓库地址配置有效证书。确保安装节点以及拉取平台镜像的每个集群节点都信任签发该证书的 CA。
- 使用可从安装节点和所有集群节点解析的稳定 DNS 名称。
- 为每种所需的 CPU 架构，以及为升级、回滚和节点恢复而保留的版本规划容量。
- 根据您的环境提供适当的可用性、监控、备份和恢复机制。在启动新 Pod、替换节点、创建或升级集群，以及安装或升级 Extensions 时，都必须确保仓库可用。
- 配置保留策略，确保不会删除已安装平台版本、集群或 Extensions 仍在引用的清单或摘要。

> **Customer-Managed Service**
>
> ACP 使用外部仓库，但不会安装、升级、备份、监控或修复该仓库。有关这些操作，请遵循仓库供应商的管理文档。

准备目标版本负载

使用与要安装的 ACP Distribution Version 和 CPU 架构完全匹配的 Core Package。请勿重复使用其他版本或架构的负载。

1. 完成 Download，解压 Core Package，然后切换到其 `installer/` 目录。
2. 按照 Required Aligned Extensions 中的说明，将所需的 Aligned Extension 软件包复制到 `plugins/`。`all` 上传模式会处理当前位于此目录中的软件包。
3. 在当前 shell 中设置仓库地址和上传凭据。读取密码时不要将其写入 shell 历史记录：

   ```
   export REGISTRY_ADDRESS="registry.example.com"
   export REGISTRY_USERNAME="<registry-username>"
   read -rsp "Registry password: " REGISTRY_PASSWORD
   export REGISTRY_PASSWORD
   printf '\n'
   ```

   对于允许匿名推送的仓库，将 `REGISTRY_USERNAME` 和 `REGISTRY_PASSWORD` 都设置为空值。
4. 从解压后的 `installer/` 目录运行两种上传模式。您可以按任意顺序运行这些命令，但两者都必须成功完成：

```
bash res/upload.sh all \
  "$REGISTRY_ADDRESS" "$REGISTRY_USERNAME" "$REGISTRY_PASSWORD"

bash res/upload.sh necessary \
  "$REGISTRY_ADDRESS" "$REGISTRY_USERNAME" "$REGISTRY_PASSWORD"
```

两种模式的处理范围有所重叠，但两者互不替代：

- `all` 会上传产品镜像、集群版本 operator 镜像，以及当前位于 `plugins/` 中的每个 `*.tgz` 软件包。它不会上传安装程序工件。
- `necessary` 会上传产品镜像、集群版本 operator 镜像、安装程序工件，以及创建 `global` 集群所需的引导插件。它不会上传 `plugins/` 中的每个软件包。

5. 在仓库管理界面或 API 中，确认新上传的仓库和清单均已存在，并且不会立即符合清理条件。

故障排除

症状	可能原因	操作
`x509: certificate signed by unknown authority`	仓库 CA 或中间证书不受信任。	在安装节点以及拉取镜像的每个集群节点上安装完整的 CA 链，然后重试。请勿在生产环境中以不安全仓库替代 HTTPS 作为变通方案。
证书名称不匹配	仓库地址与证书的主题备用名称不匹配。	使用证书所涵盖的仓库 DNS 名称，或替换证书。
`unauthorized` 或 `authentication required`	上传凭据不正确或缺少推送权限。	验证 `res/upload.sh` 使用的账户，并确认该账户可以推送到所需的仓库路径。
`denied` 或仓库创建失败	上传账户无法创建或更新所需的仓库路径。	授予该账户所需的仓库权限，然后重新运行两种上传模式。

症状	可能原因	操作
报告未找到所需软件包	Core Package 不完整，解压了错误的版本或架构，或者未将所需的 Extension 软件包复制到 `plugins/`。	重试前，验证软件包校验和、版本、架构以及 `plugins/` 的内容。

下一步

两种上传模式均成功且仓库中已包含目标负载后，请继续执行所选的安装路径：

- 在非 Alauda OS 上安装。
- 在 Alauda OS 上安装 global 集群。在步骤 2 中使用 `--registry` 启动引导脚本，并在步骤 8 中设置 `registry.externalAddress`。

通信矩阵

此页面列出了在运行非 Alauda OS 的集群节点上，ACP 组件监听的端口。启用这些节点上的主机防火墙时，请使用此列表构建允许规则。

目录

此矩阵的范围
何时使用此页面
此矩阵未涵盖的内容
平台端口
可达性
NodePort 范围
条件端口
Calico

此矩阵的范围

此矩阵涵盖 ACP 核心组件和 Aligned 扩展使用的端口。Agnostic 扩展所需的端口由相应扩展维护，因此未在此处列出。

列出的部分组件仅在 `global` 集群节点上运行。业务集群节点会根据其运行的组件，监听适用的端口子集。

> **此页面不是完整的防火墙配置**
>
> 此矩阵是组件端口清单，而不是已完成的规则集。某些部署选项会增加一些要求，但这些要求不会体现为节点上的监听端口。应用任何防火墙规则之前，请参阅此矩阵未涵盖的内容。

何时使用此页面

仅当集群节点启用了主机防火墙（例如 `firewalld` 或 `ufw`）时，才需要此页面。默认且推荐的节点配置是禁用主机防火墙。在启用主机防火墙之前，请阅读强化节点安全配置，其中介绍了 `init.sh` 参数、运行安装程序的节点所使用的其他端口，以及应用配置的顺序。

此页面不适用于运行 Alauda OS 的集群，因为平台会维护节点端口配置。

此页面不同于节点预处理中的端口占用检查。该检查会在安装开始前确认没有其他进程已绑定平台所需的端口。此页面说明主机防火墙必须允许哪些端口。启用主机防火墙时，两者都适用。

> **在应用防火墙规则之前允许集群内部流量**
>
> 此矩阵列出了节点监听的端口。Pod 到 Pod 和 Pod 到 Service 的流量使用任意端口，不在此矩阵涵盖范围内，因此仅允许此处列出端口的防火墙会破坏集群网络。
>
> 启用主机防火墙时，还应将 Pod CIDR、Service CIDR 以及 CNI 插件创建的网络接口视为可信来源，以免集群内部流量按端口受到过滤。接口名称取决于 CNI 插件和环境，因此请从节点读取这些名称，而不要假定固定名称。

此矩阵未涵盖的内容

请根据您自己的部署逐项检查以下内容。即使某项要求适用于您的集群但未列入矩阵，仍必须允许该要求。

- **Pod** 和 **Service** 流量。请参阅上方的警告。
- 辅助网络和非默认容器网络。表中的 `6081/UDP` 条目是默认 Kube-OVN Geneve 隧道使用的端口。如果您更改 Kube-OVN 隧道类型，或使用 Multus 在默认网络之上附加辅助网络，

这些网络会使用自己的封装在节点 IP 地址之间传输 Pod 流量；此类流量未列于此处。请确认您的配置使用了哪些网络，并允许节点之间传输相应流量。

- `global` 集群与业务集群之间的流量。`global` 集群必须能够访问其管理的每个业务集群的控制平面节点，业务集群自身的负载均衡器也必须能够访问这些节点。有关涉及的端口，请参阅在非 Alauda OS 上创建集群。
- 负载均衡器转发。当负载均衡器位于平台访问地址前端时，转发流量的来源并非集群节点 IP 地址。有关涉及的端口，请参阅 LoadBalancer 转发规则。
- `Self-built VIP`。此选项会运行 keepalived，该程序使用 VRRP 协议，而不是 TCP 或 UDP 端口。`firewalld` 和 `ufw` 默认会阻止 VRRP，因此除非允许共享 VIP 的节点之间传输 VRRP 流量，否则 VIP 将无法启动。
- 全局灾难恢复。DR 对需要在主集群网络与备用集群网络之间开放多个端口，且两个方向都必须开放。请参阅全局灾难恢复中的网络要求。

平台端口

端口	协议	组件
`22`	TCP	`sshd`
`80`	TCP	`alb2`
`443`	TCP	`global-alb2`
`1936`	TCP	`alb2`
`2049`	TCP	`nfs`
`2379`	TCP	`etcd`
`2380`	TCP	`etcd`
`6081`	UDP	`kube-ovn`
`6443`	TCP	`kube-apiserver`
`6641`	TCP	`ovsdb-server`

端口	协议	组件
`6642`	TCP	`ovsdb-server`
`6643`	TCP	`ovsdb-server`
`6644`	TCP	`ovsdb-server`
`7473`	TCP	`frr-metrics`
`9000`	TCP	`minio`
`9100`	TCP	`node_exporter`
`10250`	TCP	`kubelet`
`10257`	TCP	`kube-controller-manager`
`10259`	TCP	`kube-scheduler`
`10660`	TCP	`kube-ovn-controller`
`10661`	TCP	`kube-ovn-monitor`
`10665`	TCP	`kube-ovn-daemon`
`11443`	TCP	`registry`
`11780`	TCP	`global-alb2`
`11782`	TCP	`global-alb2`
`17472`	TCP	`speaker`
`17946`	TCP	`speaker`
`30000-32767`	TCP	用户应用 `NodePort` 范围
`30000-32767`	UDP	用户应用 `NodePort` 范围

可达性

- 表中的每个端口都必须能在集群节点之间访问，运行安装程序的节点也必须能够访问这些端口。
- `443/tcp` 还承载平台访问地址，因此访问平台的用户和系统也必须能够访问该端口。
- 某些部署选项还要求从集群网络外部访问其中一些端口。请参阅此矩阵未涵盖的内容。

NodePort 范围

`30000-32767` 是为用户应用创建的 `NodePort` Services 预留的端口范围。部署使用该范围内端口的应用时，相应端口会在节点上开放；删除对应的 Service 后，该端口会再次关闭。只有当您的应用使用 `NodePort` Services 时，才应在主机防火墙上允许此范围，并将其限制为这些应用预期服务的来源。

条件端口

Calico

仅当集群使用 Calico CNI 插件时，才需要以下端口。

端口	协议	组件
`179`	TCP	`calico-node`、BGP
`9091`	TCP	`calico-node`
`9324`	TCP	`bird_exporter`

在 Alauda OS 上安装

`global` 集群是平台的控制平面：它托管 registry、基础 operator、集群 registry 和平台的身份堆栈，并编排每个业务集群的生命周期。本节介绍在节点运行 Alauda OS 时如何安装 `global` 集群，包括可选的灾难恢复部署。

要在 Ubuntu 或 RHEL 等非 Alauda OS 上安装 `global` 集群，请参阅在非 Alauda OS 上安装。

安装

适用场景
通用先决条件
Bootstrap 节点要求
兼容性和版本输入
安装生命周期
操作步骤
可选的灾难恢复部署
验证
弃用 Bootstrap 集群
后续步骤
示例：Huawei DCS `global` 完整清单

Global 集群灾难恢复

部署操作步骤
网络要求
可观测性部署
操作范围
故障转移
裸金属 DR 模型
Provider Notes
另请参阅

目录

本节不涵盖的内容

本节不涵盖的内容

- 在创建业务集群之前，将基础设施 provider 和 Cluster Bootstrap Provider 上传并安装到 `global` 集群：请参阅准备。
- 为业务集群配置基础设施资源：请参阅基础设施资源。
- 创建业务集群：请参阅在 Alauda OS 上创建。
- 升级 `global` 集群上的 Kubernetes：请参阅在 Alauda OS 上升级 Kubernetes。
- 升级业务集群：请参阅在 Alauda OS 上升级 Kubernetes。

安装 global 集群

本文介绍如何在 Alauda OS 上安装 `global` 集群。`global` 集群是平台控制平面，通过 Cluster API 进行置备。当平台控制平面必须运行在 Alauda OS 等不可变操作系统上时，请使用此路径。

对于 Huawei DCS 上的 `global` 集群，请使用为此 ACP 版本发布的 Alauda OS 镜像创建 VM 模板。请参阅下载 Alauda OS。

目录

- 适用场景
- 通用先决条件
- Bootstrap 节点要求
  - 硬件
  - 操作系统
  - 网络
- 兼容性和版本输入
- 安装生命周期
- 操作步骤
  - 步骤 1 — 准备通用变量
  - 步骤 2 — 创建引导集群
  - 步骤 3 — 上传并安装 Provider 软件包
  - 步骤 4 — 配置 Provider 特定的全局 Manifest
  - 步骤 5 — 应用 global 清单

步骤 6 — 等待控制平面
步骤 7 — 导入 Provider 资源
步骤 8 — 触发平台安装
步骤 9 — 监控安装
可观察到的阶段
安装期间的信号
常见停滞情况及查看位置
可选的灾难恢复部署
准备共享 DR 变量
Bare Metal：准备共享 ServiceAccount 签名密钥
将 DR etcd Server 证书 SAN 添加到 KubeadmControlPlane
添加 provider 特定的 DR 字段
安装主集群和备用集群
Bare Metal：在移除任一 bootstrap 集群前进行验证
Bare Metal：应用 Rule ConfigMap
安装 etcd-sync (ACP 4.4.0+)
Bare Metal：验证动态规则源
启动并验证同步
验证
弃用 Bootstrap 集群
后续步骤
示例：Huawei DCS `global` 完整清单
应用前：在 DCS 上准备以下内容
内部 Self-built VIP 变体
需要替换的值

适用场景

在满足以下所有条件时选择此安装路径：

- 您希望 `global` 集群运行在不可变操作系统上。目前支持的镜像是 Alauda OS。

- 您的基础设施属于有文档说明的 provider：Huawei DCS、VMware vSphere、Huawei Cloud Stack 或裸金属。
- 您可以运行一个临时 bootstrap 节点，该节点能够访问目标 IaaS 平台。

对于 Ubuntu 或 RHEL 等非 Alauda OS，请改用在非 Alauda OS 上安装。

通用先决条件

以下先决条件适用于每个 provider：

- 一个满足 Bootstrap 节点要求的 bootstrap 节点。灾难恢复部署需要两个节点，分别用于主集群和备用 `global` 集群；请参阅安装主集群和备用集群。
- 来自 Customer Portal 的 Core Package。
- 来自 **Common ACP Extensions** 的八个必需 Aligned Extension 软件包，其 ACP 版本和架构必须与 Core Package 相同。请参阅必需的 Aligned Extension。
- Alauda Container Platform Cluster Bootstrap Provider 软件包。
- 适用于目标平台的基础设施 provider 软件包。
- bootstrap 节点与目标 IaaS 平台 API 端点之间的网络可达性。请参阅网络。
- 为 `global` 控制平面和工作节点规划 IP 和主机名。有关各 provider 使用的资源模型，请参阅基础设施资源。
- `global` 集群的稳定 Kubernetes API 端点。创建集群前，请在规划控制平面端点中选择并验证特定于 provider 的端点模式。
- 平台访问地址、registry 地址以及 Pod 和 Service CIDR 范围。如果要使用客户管理的外部平台镜像 registry，请先完成准备；请参阅准备外部平台镜像 registry。
- 对于使用 ACP 提供的 Alauda OS 镜像的 x86_64 节点，其底层 CPU 必须支持 `x86-64-v2` ISA 基线。请参阅操作系统支持矩阵。

> **命名约定（必需）**
>
> 此规则适用于此安装路径支持的每个基础设施 provider，包括 Huawei DCS、Huawei Cloud Stack、VMware vSphere 以及未来添加的任何 provider。您在步骤 4 中编写的每个 manifest 都必须遵循此规则。这些资源命名错误会导致两种不同的故障模式，具体如下；一种会破坏初始置备，另一种只会在灾难恢复期间出现。

- CAPI `Cluster` 和 provider 的基础设施集群资源（例如 Huawei DCS 的 `DCSCluster` 或 Huawei Cloud Stack 的 `HCSCluster`；每个 provider 都有其对应资源）必须严格命名为 `global`。`cpaas-installer` 会按字面名称查找这些资源；只有在基础设施集群命名为 `global` 时，Huawei Cloud Stack provider 才会分配全局 ELB 监听器端口（registry 和控制台使用 `11443`，DR etcd 同步使用 `2379`，Web 访问使用 `443`）。使用其他名称会静默破坏 registry 拉取、DR etcd 同步和 Web 控制台。
- 每个其他 CAPI 资源（`KubeadmControlPlane`、`KubeadmConfigTemplate`、`MachineDeployment`）以及每个其他 provider 基础设施资源（machine template、IP/主机名池、machine config pool 和其他任何特定于 provider 的资源）都必须使用带有 `global-` 前缀的名称。DR（故障转移）机制使用此前缀来识别由 `global` 集群拥有的资源。不带 **`global-`** 前缀的 **`global`** 集群资源对 **DR** 不可见，并会导致备用集群的机器在故障转移时被删除——集群会正常完成置备并运行，但首次执行 DR 时会丢失节点。这是硬性要求，而非样式约定。
- `Cluster.spec.controlPlaneRef.name` 及其他所有交叉引用都必须与带前缀的名称完全匹配。

Bootstrap 节点要求

bootstrap 节点是一台运行 Core Package 中 `setup.sh` 的临时机器。在安装期间，它承载基于 KIND 的 `minialauda` 管理集群、内置平台 registry、Cluster API bootstrap 和基础设施 provider，以及 `cpaas-installer`。它不会加入 `global` 集群，交接完成后即会移除——请参阅停用 Bootstrap 集群。

请根据其提供的镜像负载而非稳定运行时负载来规划规格：在集群置备期间，每个 `global` 节点都会从此主机拉取平台镜像集。

硬件

资源	最低要求（**x86_64**）
CPU	4 个核心
内存	8 GB

资源	最低要求（**x86_64**）
磁盘	300 GB 或更多，可供提取目录（`/root/cpaas-install`）和 `/var/cpaas` 使用

CPU 和内存最低要求来自此安装路径的标准交付实践。在非 Alauda OS 安装路径中，运行 `setup.sh` 的机器会成为第一个控制平面节点，因此必须满足前提条件中的控制平面最低要求；此处的 bootstrap 节点不会加入集群——它只提取 Core Package、运行临时的基于 KIND 的 bootstrap 集群，并在交接前提供平台 registry。这是较轻的临时负载，4 个核心 / 8 GB 是标准交付所采用并验证的规格。

在 arm64 上，请遵循同一页面中的平台 ARM 约定：至少使用 x86_64 规格的 1.5 倍，最好使用 2 倍，因此最低要求为 **6** 个核心 **/ 12 GB**，推荐配置为 **8** 个核心 **/ 16 GB**。

磁盘是最常导致 bootstrap 停滞的资源。请从以下三个方面规划磁盘空间：

使用者	空间要求
已提取的 Core Package	提取后至少 100 GB。请参阅在非 Alauda OS 上安装
`/var/cpaas` 下的内置 registry 数据	目标版本的平台镜像负载，以及复制到 `plugins/` 中的 Aligned Extension 软件包
暂存待上传的 provider 软件包和 OS 工件	Cluster Bootstrap Provider 和基础设施 provider 软件包，以及每种节点架构各一个 Alauda OS 镜像或 VM 模板工件

操作系统

bootstrap 节点不运行 Alauda OS。它需要一个带有 Bash 和 `root` 权限的 64 位 Linux 发行版，以便 `setup.sh` 能够启动步骤 2 中所述的 bootstrap 集群。

节点预处理中的要求——经过验证的 OS 和内核列表、SSH 用户以及 `sshd_config` 设置——不适用于此安装路径中的 bootstrap 节点。这些要求针对的是平台通过 SSH 加入集群的非 Alauda OS 机器，而 bootstrap 节点不会加入 `global` 集群。这与非 Alauda OS 安装路径不同，在该路径中，运行 `setup.sh` 的机器会成为第一个控制平面节点，因此必须满足这些要求。

在没有其他约束时，从该经过验证的列表中选择一个发行版仍然是安全的默认选择。

x86-64-v2 基线适用于集群节点，不适用于 bootstrap 节点

OS 支持矩阵中的 `x86-64-v2` ISA 基线适用于从 ACP 提供的 Alauda OS 镜像创建的集群节点。平台不会对运行用户提供的非 Alauda OS 的 x86_64 机器强制实施该基线，因此它不是 bootstrap 节点本身的要求。请参阅节点预处理。

如果 bootstrap 节点是运行集群节点的同一基础设施上的虚拟机，检查其 CPU 标志可以及早发现缺少基线的问题。但这不能替代在实际运行 Alauda OS 节点的主机上验证 `x86-64-v2`，因为来宾机可见的标志取决于 hypervisor 的 CPU 模型。

网络

bootstrap 节点需要一个在整个安装生命周期内保持不变的静态 **IP** 地址。`HOST_IP` 会写入置备节点用于拉取镜像的 `cpaas.io/registry-address` annotation；对于裸金属，还会写入注册端点和其服务证书的 SAN。安装过程中途更改地址会导致节点置备失败。

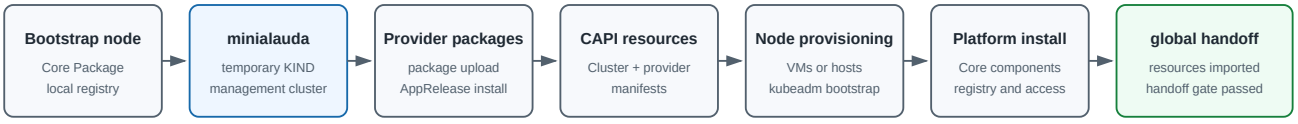
来源	目标	端口	用途
Bootstrap 节点	目标 IaaS 平台 API	provider 特定：DCS VRM `TCP/7443` 和 DCS 物理主机 MGMT（通常为 `TCP/8443`）；vCenter `TCP/443`；HCS API 和 IAM 端点	Cluster API provider 在 `minialauda` 中运行，并通过 IaaS API 调谐机器。对于 DCS，此路径还承载 Ignition ISO 上传；请参阅在 Huawei DCS 上创建集群
Bootstrap 节点	`global` 控制平面端点	`TCP/6443`	创建控制平面时，Cluster API 必须能够访问该端点。请参阅规划控制平面端点
正在置备的 `global` 节点	Bootstrap 节点 registry	`TCP/11443`	在 `global` 集群自身的 registry 接管之前，节点从 `NODE_REGISTRY_ADDRESS` 拉取平台镜像
`global` 主机，仅限裸金属	Bootstrap 节点	`TCP/12443`	bootstrap 阶段使用的 Elemental 注册端点

兼容性和版本输入

安装前，记录交付包支持的版本集：

输入	用途
Core Package 版本	提供安装程序、本地 registry 和基础平台负载。
Cluster Bootstrap Provider chart 版本	必须与全局 manifest 使用的 Cluster API 控制平面资源匹配。
基础设施 provider chart 版本	使用目标版本随附的 VMware vSphere、DCS、HCS 或 Bare Metal provider chart 版本。
Alauda OS 镜像或 VM 模板	必须包含 `K8S_VERSION` 使用的 Kubernetes 版本。
`K8S_VERSION`	使用与目标 Alauda OS 镜像匹配的带 v 前缀 semver，例如 `v<major>.<minor>.<patch>`。

安装生命周期



使用以下阶段图确定安装每个部分的输入、预期结果和首个诊断位置。

阶段	输入	应用	预期结果	首先诊断
引导环境	Core Package 和引导节点	运行 `setup.sh`	`minialauda` 管理集群、本地 registry 和安装程序 Pod 正在运行。	`cpaas-sy` 节点容器和

阶段	输入	应用	预期结果	首先诊断
Provider 交付	Cluster Bootstrap Provider 和基础设施 provider 软件包	推送软件包并应用 provider `AppRelease` 对象。	Provider CRD 和控制器均为 Ready。	`AppRelea` provider Pc 访问。
集群定义	Provider 凭证、端点、网络、节点和镜像值	应用完整的 Cluster API manifest。	`Cluster`、基础设施集群、控制平面和机器资源出现。	资源条件和 controller 日
节点置备	VM 模板或已注册主机	让基础设施 provider 调谐 Machine。	VM 或物理主机启动并接收 kubeadm 引导数据。	Provider M IaaS 事件和 日志。
控制平面	Kubernetes 和组件版本	等待 `KubeadmControlPlane`。	预期的控制平面副本均为 Ready，且 API 端点响应正常。	`KubeadmC` Machine、 状况。
平台安装	安装程序请求和导入资源	触发安装程序。	ACP Core 组件、registry、平台访问和基础模块变为健康状态。	`cpaas-sy` 程序进度 A
移交	已导入的 provider 资源和凭证	验证最终的 `global` 集群以及每个 provider 特定的移交门控，然后仅停用 `minialauda`。	最终集群持有已导入的 provider 资源和凭证，不再依赖临时管理集群。对于 Bare Metal，system-agent 移交 ConfigMap 与当前端点、身份验证配置文	已导入资源 证、最终集 以及 provic Job 和 Cor

阶段	输入	应用	预期结果	首先诊断
			件和完整目标集匹配。	

操作步骤

步骤 1 — 准备通用变量

在引导节点上设置通用变量。

```
export HOST_IP="<bootstrap-host-ip>"
export LOCAL_REGISTRY_ADDRESS="127.0.0.1:11443"
export BOOTSTRAP_REGISTRY_ADDRESS="172.18.0.1:11443"
export NODE_REGISTRY_ADDRESS="${HOST_IP}:11443"
export CONTROL_PLANE_VIP="<global-control-plane-vip>"
export PLATFORM_HOST="<platform-access-domain-or-vip>"
export REGISTRY_DOMAIN="<platform-registry-domain-or-vip>:11443"
export CLUSTER_CIDR="100.3.0.0/16"
export SERVICE_CIDR="100.4.0.0/16"
export KUBE_OVN_JOIN_CIDR="<kube-ovn-join-cidr>"
export K8S_VERSION="<target-kubernetes-version>"
export INGRESS_CLASS_NAME="global-alb2"
export PROVIDER_SECRET_NAME="global-secret"
# Use v-prefixed semver that matches the target Alauda OS image.
```

从引导节点推送软件包时使用 `LOCAL_REGISTRY_ADDRESS`。在 AppRelease chart 仓库值中使用 `BOOTSTRAP_REGISTRY_ADDRESS`，因为 provider Pod 会从引导集群的网络内部读取 chart 仓库。由于置备的 `global` 节点在置备期间必须通过其子网可访问的地址拉取镜像，因此在 Cluster API registry annotation 中使用 `NODE_REGISTRY_ADDRESS`（引导节点的 registry，即 `<bootstrap-host-ip>:11443`）。这是一个临时值：`global` 集群自身的 registry 启动后，安装程序会自动将 `Cluster` 和 `DCSCluster` 上的 `cpaas.io/registry-address` annotation 重写为永久平台 registry，因此后续调谐会从 `global` 集群而不是引导节点拉取镜像。

请区分 Registry 的格式：

用途	格式	示例
Registry HTTP API 请求	带有 `/v2/` 路径的 URL	`http://${LOCAL_REGISTRY_ADDRESS}/v2/acp/chart-cluster-api-provider-kubeadm/tags/list`
容器镜像引用	Registry、仓库以及 tag 或 digest	`${BOOTSTRAP_REGISTRY_ADDRESS}/acp/kubeadm-bootstrap-controller:<tag>`
`cpaas.io/registry-address` annotation	仅使用 `<host>:<port>`	`${NODE_REGISTRY_ADDRESS}`
AppRelease `repoURL`	不带 `/v2/` 的 Registry 地址	`${BOOTSTRAP_REGISTRY_ADDRESS}`

`/v2/` 片段仅属于 Registry HTTP API 请求。不要将其添加到镜像引用、`cpaas.io/registry-address` annotation 或 AppRelease `repoURL` 中。

步骤 2 — 创建引导集群

使用 Bash 运行 Core Package 提供的引导脚本。该脚本会在引导节点上创建一个名为 `minialauda` 的临时、基于 KIND 的引导集群——这是仅用于置备 `global` 集群的临时 Cluster API 管理集群。完成后，将引导控制平面容器中的匹配 `kubectl` 客户端提供给引导节点，并配置导出的 kubeconfig。

```
mkdir -p /root/cpaas-install
tar -xvf <core-package> -C /root/cpaas-install
cd /root/cpaas-install/installer
```

Core Package 不包含所需的 Aligned Extension 软件包。在运行 `setup.sh` 之前，将 Required Aligned Extensions 中列出的八个软件包分别复制到解压后的 Core Package 的 `plugins/` 目录中。如果缺少这些软件包，则无法安装所需的 Aligned Extensions。每个软件包运行一次以下命令：

```
cp <path-to-extension-package> /root/cpaas-install/installer/plugins/
```

下载的文件名为 `<package-name>.<arch>.<version>.tgz`。请保持文件名不变。这八个软件包名称为 `console`、`console-central`、`console-marketplace`、`application-catalog`、`aiops-base`、`container-platform`、`acp-business` 和 `gatewayapi-plugin`。保留 `plugins/` 中已有的文件，并确认八个软件包均已存在：

```
ls -1 /root/cpaas-install/installer/plugins/
```

不要将 **Common ACP Extensions** 中的其他软件包复制到此目录中。Cluster Bootstrap Provider 和基础设施 provider 软件包也不复制到此处；步骤 3 会将它们上传到引导 Registry。

然后运行引导脚本。要使用外部平台镜像 registry，请使用 `bash setup.sh --registry "<registry-address>"` 替代 `bash setup.sh`。Registry 地址采用 `<host-or-IP>[:port]` 格式，不包含 `http://` 或 `https://`。如果 registry 需要身份验证，还要同时提供拉取 `--username` 和 `--password`；有关如何在不暴露密码的情况下输入这些值，请参阅 Start the Installer。按照 Prepare an External Platform Image Registry 中的说明，在执行此步骤前准备好 registry。

```
bash setup.sh
mkdir -p "${HOME}/.local/bin" ~/.kube

if ! command -v kubectl >/dev/null 2>&1; then
  nerdctl cp minialauda-control-plane:/usr/bin/kubectl \
    "${HOME}/.local/bin/kubectl"
  chmod 0755 "${HOME}/.local/bin/kubectl"
  export PATH="${HOME}/.local/bin:${PATH}"
fi

cp /var/cpaas/data/alauda.kubeconfig ~/.kube/config
kubectl get nodes
```

引导脚本会置备内置 registry、Cluster API 控制平面以及驱动 `global` 集群安装的安装程序组件。

默认情况下，内置引导 Registry 为匿名访问，不会创建 `global-registry-auth` Secret。如果在引导设置期间同时配置了 Registry 用户名和密码，`setup.sh` 会在 `cpaas-system` 中创建该 Secret。

步骤 3 — 上传并安装 Provider 软件包

将 Cluster Bootstrap Provider 软件包和基础设施 provider 软件包上传到本地 registry。

下面的 AppRelease 示例使用默认的匿名引导 Registry，因此不引用 `global-registry-auth`。

如果引导 Registry 需要身份验证，请在创建任何 provider AppRelease 之前确认 Secret 存在：

```
kubectl -n cpaas-system get secret global-registry-auth -o name
```

然后在应用每个 provider AppRelease 之前添加这两个身份验证字段：

```
spec:
  source:
    chartPullSecret: global-registry-auth
  values:
    global:
      registry:
        imagePullSecrets:
          - global-registry-auth
```

如果 Registry 为匿名访问，则不要创建占位 Secret，也不要添加这些字段。引导 Secret 也不是凭证传输机制：默认情况下，业务集群通过 `global` 集群的 `public-registry-credential` 流程接收其 Registry 拉取 Secret；也可以按照 Choose the Image Registry for a Workload Cluster 中的说明，将业务集群绑定到专用 registry。不要将引导集群的 `global-registry-auth` Secret 复制到业务集群中。

> **为什么每个 provider 的 cluster.type 都是 Baremetal**
>
> 下面选项卡中的 AppRelease 值均设置了 `global.cluster.type: Baremetal`。这是 chart 内部的分类器，而不是 IaaS provider 名称。在 Huawei DCS、VMware vSphere、Huawei Cloud Stack 和 Bare Metal 的 `global` 安装中，都应保留 `Baremetal`。该值决定平台如何配置节点级组件；它不会选择基础设施 provider。

Huawei DCS

设置 provider 软件包路径和 chart 版本。

```
export DCS_PROVIDER_PACK="/root/cluster-api-provider-dcs.amd64.<version>.tgz"
export KUBEADM_PROVIDER_PACK="/root/cluster-api-provider-kubeadm.amd64.<version>.tgz"
export DCS_PROVIDER_VERSION="<dcs-provider-chart-version>"
export KUBEADM_PROVIDER_VERSION="<kubeadm-provider-chart-version>"
```

上传软件包。

```
/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${DCS_PROVIDER_PACK}"

/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${KUBEADM_PROVIDER_PACK}"
```

创建并应用 Cluster Bootstrap Provider 和 DCS provider 的 AppRelease 资源。

```
mkdir -p /root/yamls
export DCS_PROVIDER_APPRELEASES="/root/yamls/dcs-provider-apprelease
s.yaml"

cat > "${DCS_PROVIDER_APPRELEASES}" <<EOF
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-kubeadm
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-kubeadm
        releaseName: cluster-api-provider-kubeadm
        targetRevision: ${KUBEADM_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
```

```
      region: global
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-dcs
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-dcs
        releaseName: cluster-api-provider-dcs
        targetRevision: ${DCS_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
      region: global
```

```
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
EOF

kubectl apply -f "${DCS_PROVIDER_APPRELEASES}"

until kubectl get crd kubeadmcontrolplanes.controlplane.cluster.x-k8
s.io --ignore-not-found 2>/dev/null | grep -q kubeadmcontrolplanes.co
ntrolplane.cluster.x-k8s.io; do
  sleep 10
done

until kubectl get crd dcsclusters.infrastructure.cluster.x-k8s.io --i
gnore-not-found 2>/dev/null | grep -q dcsclusters.infrastructure.clus
ter.x-k8s.io; do
  sleep 10
done
```

VMware vSphere

设置 provider 软件包路径和 chart 版本。

```
export VSPHERE_PROVIDER_PACK="/root/cluster-api-provider-vsphere.amd6
4.<version>.tgz"
export KUBEADM_PROVIDER_PACK="/root/cluster-api-provider-kubeadm.amd6
4.<version>.tgz"
export VSPHERE_PROVIDER_VERSION="<vsphere-provider-chart-version>"
export KUBEADM_PROVIDER_VERSION="<kubeadm-provider-chart-version>"
```

上传软件包。

```
/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${VSPHERE_PROVIDER_PACK}"

/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${KUBEADM_PROVIDER_PACK}"
```

创建并应用 Cluster Bootstrap Provider 和 VMware vSphere provider 的 AppRelease 资源。

```
mkdir -p /root/yamls
export VSPHERE_PROVIDER_APPRELEASES="/root/yamls/vsphere-provider-app
releases.yaml"

cat > "${VSPHERE_PROVIDER_APPRELEASES}" <<EOF
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-kubeadm
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-kubeadm
        releaseName: cluster-api-provider-kubeadm
        targetRevision: ${KUBEADM_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
```

```
      region: global
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-vsphere
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-vsphere
        releaseName: cluster-api-provider-vsphere
        targetRevision: ${VSPHERE_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
      region: global
```

```
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
EOF

kubectl apply -f "${VSPHERE_PROVIDER_APPRELEASES}"

until kubectl get crd kubeadmcontrolplanes.controlplane.cluster.x-k8
s.io --ignore-not-found 2>/dev/null | grep -q kubeadmcontrolplanes.co
ntrolplane.cluster.x-k8s.io; do
  sleep 10
done

until kubectl get crd vsphereclusters.infrastructure.cluster.x-k8s.io
--ignore-not-found 2>/dev/null | grep -q vsphereclusters.infrastructu
re.cluster.x-k8s.io; do
  sleep 10
done
```

Huawei Cloud Stack

设置 provider 软件包路径和 chart 版本。

```
export HCS_PROVIDER_PACK="/root/cluster-api-provider-hcs.amd64.<versi
on>.tgz"
export KUBEADM_PROVIDER_PACK="/root/cluster-api-provider-kubeadm.amd6
4.<version>.tgz"
export HCS_PROVIDER_VERSION="<hcs-provider-chart-version>"
export KUBEADM_PROVIDER_VERSION="<kubeadm-provider-chart-version>"
```

上传软件包。

```
/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${HCS_PROVIDER_PACK}"

/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${KUBEADM_PROVIDER_PACK}"
```

创建并应用 Cluster Bootstrap Provider 和 HCS provider 的 AppRelease 资源。

```
mkdir -p /root/yamls
export HCS_PROVIDER_APPRELEASES="/root/yamls/hcs-provider-apprelease
s.yaml"

cat > "${HCS_PROVIDER_APPRELEASES}" <<EOF
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-kubeadm
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-kubeadm
        releaseName: cluster-api-provider-kubeadm
        targetRevision: ${KUBEADM_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
```

```
      region: global
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-hcs
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-hcs
        releaseName: cluster-api-provider-hcs
        targetRevision: ${HCS_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
      region: global
```

```
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
EOF

kubectl apply -f "${HCS_PROVIDER_APPRELEASES}"

until kubectl get crd kubeadmcontrolplanes.controlplane.cluster.x-k8
s.io --ignore-not-found 2>/dev/null | grep -q kubeadmcontrolplanes.co
ntrolplane.cluster.x-k8s.io; do
  sleep 10
done

until kubectl get crd hcsclusters.infrastructure.cluster.x-k8s.io --i
gnore-not-found 2>/dev/null | grep -q hcsclusters.infrastructure.clus
ter.x-k8s.io; do
  sleep 10
done
```

Bare Metal

设置 provider 软件包路径和 chart 版本。

```
export BAREMETAL_PROVIDER_PACK="/root/cluster-api-provider-baremetal.
amd64.<version>.tgz"
export KUBEADM_PROVIDER_PACK="/root/cluster-api-provider-kubeadm.amd6
4.<version>.tgz"
export BAREMETAL_PROVIDER_VERSION="<baremetal-provider-chart-version
>"
export KUBEADM_PROVIDER_VERSION="<kubeadm-provider-chart-version>"
```

上传软件包。

```
/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${BAREMETAL_PROVIDER_PACK}"

/root/cpaas-install/installer/res/amd64/packtool pack push \
  -r "${LOCAL_REGISTRY_ADDRESS}" -c "${KUBEADM_PROVIDER_PACK}"
```

将 Bare Metal Alauda OS 镜像导入同一本地 registry。provider 软件包不包含这些镜像。Bare Metal OS 下载文件是一个 OCI 镜像布局归档，其中包含两个具有相同 tag 的镜像，两个镜像都不可缺少——请参阅 Image Format by Provider：

镜像	用途
`baremetal-base-image`	当主机被重新置备为 `global` 节点时，由 `elemental-image-catalog` 使用。
`baremetal-base-image-iso`	构建 `global` 主机的安装 ISO 时，由 `SeedImage.spec.baseImage` 使用。

使用 `ctr` 将归档导入引导节点上的专用 containerd namespace，并列出它创建的镜像名称。`/var/lib/containerd` 需要至少等于归档大小的可用空间：

```
export OS_IMAGE_ARCHIVE="/root/<bare-metal-os-archive>.tar"

ctr -n bare-metal-os images import --all-platforms --no-unpack "${OS_IMAGE_ARCHIVE}"
ctr -n bare-metal-os images ls -q
```

输出会列出两个具有相同 tag 的镜像：

```
baremetal-base-image-iso:<os-image-tag>
baremetal-base-image:<os-image-tag>
```

将 `OS_IMAGE_TAG` 设置为该 tag，然后推送两个镜像，并确认 registry 为每个镜像列出了该 tag：

```
export OS_IMAGE_TAG="<os-image-tag>"

for repo in baremetal-base-image baremetal-base-image-iso; do
  ctr -n bare-metal-os images push --plain-http \
    "${LOCAL_REGISTRY_ADDRESS}/tkestack/${repo}:${OS_IMAGE_TAG}" "${r
epo}:${OS_IMAGE_TAG}"
  curl -s "http://${LOCAL_REGISTRY_ADDRESS}/v2/tkestack/${repo}/tags/
list"
done
```

如果引导 Registry 需要身份验证，请使用 `-u <username>:<password>` 将凭证传递给两个命令。将镜像保留在 `bare-metal-os` namespace 中：Upload the base image to the global Registry 会在安装后再次推送这些镜像。

创建并应用 Cluster Bootstrap Provider 和 Bare Metal provider 的 AppRelease 资源。

Bare Metal provider AppRelease 还会填充镜像目录。chart 创建的 `elemental-image-catalog` ConfigMap 不包含任何条目，因此 `provider.imageCatalog.images` 必须将 `${K8S_VERSION}` 映射到上面导入的 `baremetal-base-image`；chart 会为仓库添加 `global.registry.address` 前缀。缺少该条目时，每个 `global` `BaremetalMachine` 都会在带有 `ImageResolved=False / Reason=ImageCatalogMiss` 的情况下进入 `Failed`，不会置备任何节点。

> **Bare Metal 引导端点**
>
> 在引导期间，`global` 集群尚未将控制权移交给最终 VIP。将裸金属注册路径保留在引导节点上：`global.platformUrl` 指向引导节点，`elemental.server.url` 指向 `https://<bootstrap-host-ip>:12443`。在此阶段，不要在用于 `global` 机器的 `MachineRegistration` 上设置 `baremetal.cluster.io/system-agent-server-url`。对于 Bare Metal DR，将 `baremetal.cluster.io/system-agent-auth-scope: global` 添加到该引导 `MachineRegistration`；这是移交前唯一需要的 DR 特定 system-agent annotation。DR 移交 Job 稍后会将这些机器迁移到 `https://<CONTROL_PLANE_VIP>:6443` 的本地直接 API 端点。

在安装 Bare Metal provider 之前，准备 `global-alb2` 使用的引导 HTTPS 证书。`elemental-operator` 会从 `cpaas-system/dex.tls` 挂载 CA，而当

`elemental.tls.agentTLSMode` 为 `strict` 时，`elemental-system-agent` 会使用该 CA。Secret 必须包含 `tls.crt`、`tls.key` 以及在 AppRelease 中配置的 CA bundle key，下面显示为 `ca.crt`。由于引导端点为 `https://${HOST_IP}:12443`，服务证书必须将 `${HOST_IP}` 作为 IP SAN。

如果引导集群中有 cert-manager，请使用引导本地 CA 创建或刷新 `dex.tls`：

```
kubectl get crd certificates.cert-manager.io issuers.cert-manager.io

kubectl -n cpaas-system apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: baremetal-bootstrap-selfsigned
  namespace: cpaas-system
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: baremetal-bootstrap-ca
  namespace: cpaas-system
spec:
  secretName: baremetal-bootstrap-ca
  commonName: baremetal-bootstrap-ca
  duration: 87600h
  renewBefore: 720h
  isCA: true
  privateKey:
    algorithm: RSA
    size: 2048
  usages:
    - cert sign
    - crl sign
  issuerRef:
    name: baremetal-bootstrap-selfsigned
    kind: Issuer
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: baremetal-bootstrap-ca
  namespace: cpaas-system
spec:
  ca:
    secretName: baremetal-bootstrap-ca
---
apiVersion: cert-manager.io/v1
kind: Certificate
```

```
  metadata:
    name: dex-tls-bootstrap
    namespace: cpaas-system
  spec:
    secretName: dex.tls
    commonName: ${HOST_IP}
    duration: 87600h
    renewBefore: 720h
    privateKey:
      algorithm: RSA
      size: 2048
    usages:
      - digital signature
      - key encipherment
      - server auth
    ipAddresses:
      - "${HOST_IP}"
      - "127.0.0.1"
    dnsNames:
      - global-alb2
      - global-alb2.cpaas-system.svc
      - global-alb2.cpaas-system.svc.cluster.local
    issuerRef:
      name: baremetal-bootstrap-ca
      kind: Issuer
  EOF

  kubectl -n cpaas-system wait certificate/baremetal-bootstrap-ca \
    --for=condition=Ready \
    --timeout=120s
  kubectl -n cpaas-system wait certificate/dex-tls-bootstrap \
    --for=condition=Ready \
    --timeout=120s
```

验证 Secret 是否包含预期的 key，并确认该证书对引导端点有效：

```
kubectl -n cpaas-system get secret dex.tls \
  -o jsonpath='{.data.tls\.crt}{" "}{.data.tls\.key}{" "}{.data.ca\.crt}{"\n"}'

tmp_dir=$(mktemp -d)
trap 'rm -rf "${tmp_dir}"' EXIT
kubectl -n cpaas-system get secret dex.tls \
  -o jsonpath='{.data.ca\.crt}' | base64 -d > "${tmp_dir}/ca.crt"
kubectl -n cpaas-system get secret dex.tls \
  -o jsonpath='{.data.tls\.crt}' | base64 -d > "${tmp_dir}/tls.crt"

openssl verify -CAfile "${tmp_dir}/ca.crt" "${tmp_dir}/tls.crt"
openssl x509 -in "${tmp_dir}/tls.crt" -noout -text | grep "IP Address:${HOST_IP}"

echo | openssl s_client \
  -connect "${HOST_IP}:12443" \
  -servername "${HOST_IP}" \
  -CAfile "${tmp_dir}/ca.crt" \
  -verify_return_error 2>&1 | grep "Verify return code: 0 (ok)"
```

如果引导 ALB 仍提供旧证书，请重启它并再次验证：

```
kubectl -n cpaas-system rollout restart deploy/global-alb2
kubectl -n cpaas-system rollout status deploy/global-alb2
```

不要将此引导 `dex.tls` 包含在 `dcs-import-extra-resources` 中。它仅用于临时引导端点。最终的 `global` 集群的 `dex.tls` 由安装程序平台证书流程创建或维护。对于 DR，请按照步骤 8 中的 `thirdParty` 控制台证书指南操作，使两端都提供稳定平台域名的证书。只有当客户端有意通过该 VIP 直接访问平台 HTTPS 时，控制平面 VIP 才需要包含在该平台证书中。`11443` 端口上的 Global registry 提供其自身的 `global-registry-server` 证书链，不由 `console.cert` 或 `dex.tls` 配置。

```
mkdir -p /root/yamls
export BAREMETAL_PROVIDER_APPRELEASES="/root/yamls/baremetal-provider
-appreleases.yaml"

cat > "${BAREMETAL_PROVIDER_APPRELEASES}" <<EOF
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-kubeadm
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-kubeadm
        releaseName: cluster-api-provider-kubeadm
        targetRevision: ${KUBEADM_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${PLATFORM_HOST}
      protectSecretFiles:
        enabled: false
```

```
      region: global
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
---
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: cluster-api-provider-baremetal
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: ""
  source:
    charts:
      - name: acp/chart-cluster-api-provider-baremetal
        releaseName: cluster-api-provider-baremetal
        targetRevision: ${BAREMETAL_PROVIDER_VERSION}
    repoURL: ${BOOTSTRAP_REGISTRY_ADDRESS}
  timeout: 120
  values:
    global:
      albName: ${INGRESS_CLASS_NAME}
      auth:
        default_admin: admin@cpaas.io
      cluster:
        isGlobal: true
        name: global
        networkType: kube-ovn
        type: Baremetal
      host: ${PLATFORM_HOST}
      ingress:
        ingressClassName: ${INGRESS_CLASS_NAME}
        tls:
          secretName: dex.tls
      labelBaseDomain: cpaas.io
      namespace: cpaas-system
      platformUrl: https://${HOST_IP}
      protectSecretFiles:
```

```
        enabled: false
      region: global
      registry:
        address: ${BOOTSTRAP_REGISTRY_ADDRESS}
      replicas: 1
      scheme: https
    provider:
      imageCatalog:
        images:
          ${K8S_VERSION}:
            repository: tkestack/baremetal-base-image
            tag: ${OS_IMAGE_TAG}
    handoffHook:
      # Normal non-DR bootstrap baseline. Bare Metal DR must patch th
is to true
      # before calling the installer API, as shown in Optional Disast
er Recovery Deployment.
      directAPIServer: false
      controlPlaneVIP: ${CONTROL_PLANE_VIP}
      delivery:
        enabled: true
        mode: always
    elemental:
      server:
        url: https://${HOST_IP}:12443
      systemAgent:
        authMode: shared
        # Normal non-DR bootstrap baseline. Bare Metal DR must enable
the split
        # and set sharedAuthReadOnly according to this side's active/
standby role.
        splitAuthEnabled: false
        serviceAccountName: baremetal-system-agent
        globalServiceAccountName: baremetal-global-system-agent
        sharedAuthReadOnly: false
      tls:
        agentTLSMode: strict
        caCertSecretName: dex.tls
        caCertSecretKey: ca.crt
EOF

kubectl apply -f "${BAREMETAL_PROVIDER_APPRELEASES}"

until kubectl get crd kubeadmcontrolplanes.controlplane.cluster.x-k8
```

```
s.io --ignore-not-found 2>/dev/null | grep -q kubeadmcontrolplanes.co
ntrolplane.cluster.x-k8s.io; do
  sleep 10
done

until kubectl get crd baremetalclusters.infrastructure.cluster.x-k8s.
io --ignore-not-found 2>/dev/null | grep -q baremetalclusters.infrast
ructure.cluster.x-k8s.io; do
  sleep 10
done

until kubectl get crd machineinventories.elemental.cattle.io --ignore
-not-found 2>/dev/null | grep -q machineinventories.elemental.cattle.
io; do
  sleep 10
done
```

上面的完整 AppRelease 有意展示正常的非 DR 基线。对于 Bare Metal DR 安装，不要在这三个基线值保持不变的情况下调用安装程序 API。请先应用 Optional Disaster Recovery Deployment 中按角色划分的 DR patch，然后在该引导集群上验证生效的 AppRelease 值。

provider 启动后，验证 chart 值是否已被接受。如果出现带有未知 flag（例如 `--system-agent-auth-mode`）的 CrashLoopBackOff，则表示 AppRelease chart 与 elemental-operator 镜像不匹配；请先安装来自同一版本负载的 chart 和镜像，再继续操作。

```
kubectl -n cpaas-system get pods | grep -E 'cluster-api-provider-bare
metal|elemental'
kubectl -n cpaas-system logs deploy/elemental-operator --tail=100
kubectl -n cpaas-system get configmap elemental-image-catalog -o json
path='{.data}'
```

目录输出必须包含 `${K8S_VERSION}` 作为 key。空输出或 `{}` 表示 `provider.imageCatalog.images` 值未传递到 release；请修复 AppRelease，然后再应用 `global` manifest。

步骤 4 — 配置 Provider 特定的全局 Manifest

为 `global` 集群创建一个 provider 特定的 manifest。该 manifest 使用与业务集群相同的 provider 资源，但还必须包含平台控制平面所需的全局专用标签、annotation、registry 值、与安装程序兼容的 kubeadm 设置以及持久化数据路径。

使用 provider 创建指南作为详细资源参考：

- Huawei DCS：Create a Cluster on Huawei DCS
- VMware vSphere：Infrastructure Preparation 和 Create a Cluster on VMware vSphere
- Huawei Cloud Stack：Create a Cluster on Huawei Cloud Stack

将 Common Prerequisites 中的命名约定 应用于你在下面编写的 manifest 中的每个资源。

将 `KubeadmControlPlane.spec.kubeadmConfigSpec.format` 设置为目标 provider 接受的值。这是一个 API 字段：输入 `cloud-config`，而不是 `cloud-init`。对于下面列出的 provider，`cloud-init` 是接收并应用生成的 `cloud-config` 数据的客户机操作系统软件。Provider controller 会强制执行该字段值：

Provider	引导 **userdata** 格式
Huawei DCS	`ignition`（由 provider 强制执行；DCS provider 会以 `invalid format, expected ignition, got <other>` 拒绝任何其他格式）。
VMware vSphere	`cloud-config`（CAPI 引导数据由客户机中的 `cloud-init` 软件处理；不支持设置 `ignition`）。
Huawei Cloud Stack	`cloud-config`（HCS provider 会拒绝 `ignition`；客户机会使用 `cloud-init` 处理生成的数据）。
Bare Metal	`cloud-config`（裸金属 provider 使用 CAPI 引导数据，并渲染由客户机使用 `cloud-init` 处理的 elemental plan）。

Huawei DCS

在渲染 DCS `global` manifest 之前，设置其输出路径。

```
export GLOBAL_DCS_YAML="/root/yamls/new-global.yaml"
```

按照Cloud Credentials中的说明，在组装 manifest 之前创建 DCS API 凭证 `Secret`。不要将其作为 YAML 写入 manifest：第 5 步使用 `kubectl apply` 应用 manifest，这会将凭证记录在 Secret 的 `kubectl.kubernetes.io/last-applied-configuration` annotation 中。manifest 通过 `DCSCluster.spec.credentialSecretRef` 引用该凭证。

然后，DCS `global` manifest 必须在 `cpaas-system` namespace 中包含以下资源：

资源	用途
用于控制平面节点的 `DCSIpHostnamePool`	分配静态 IP、主机名、网络设置以及由资源池管理的持久化磁盘。
用于控制平面节点的 `DCSMachineTemplate`	定义 DCS VM 模板、文件夹、CPU、内存和模板本地磁盘。
`KubeadmControlPlane`	引导 Kubernetes 控制平面。将 `spec.version` 设置为 `${K8S_VERSION}`。
`DCSCluster`	定义 DCS 基础设施集群和控制平面端点。
`Cluster`	将 Cluster API `Cluster` 连接到 `DCSCluster` 和 `KubeadmControlPlane`。
用于工作节点的 `DCSIpHostnamePool`、`DCSMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment`	创建工作节点。

使用在 Huawei DCS 上创建集群和Huawei DCS 基础设施资源中的 DCS 资源字段。对于 `global` 集群，还需满足以下要求：

- 将 `Cluster.metadata.name` 和 `DCSCluster.metadata.name` 设置为 `global`（基础设施集群与 CAPI 共享 `Cluster` 名称）。为其他每个 CAPI 资源和 provider 资源添加 `global-` 前缀；下面的连接片段使用 `KubeadmControlPlane.metadata.name: global-kcp`。

- 将 `DCSCluster.spec.credentialSecretRef.name` 设置为 `${PROVIDER_SECRET_NAME}`。第 7 步会将此 Secret 导入最终的 `global` 集群。
- 添加 `Cluster.metadata.labels.is-global: "true"` 和 `Cluster.metadata.labels.cluster-type: DCS`。
- 添加包含 `${NODE_REGISTRY_ADDRESS}` 的 `Cluster.metadata.annotations["cpaas.io/registry-address"]`。
- 为 Alauda OS 设置 `KubeadmControlPlane.spec.kubeadmConfigSpec.format: ignition`。
- 使用外部 LoadBalancer 并设置 `DCSCluster.spec.controlPlaneLoadBalancer.type: external`，或使用 `type: internal` Self-built VIP。对于内部模式，在控制平面 Layer 2 网络中预留 IPv4 VIP，并使用在 Huawei DCS 上创建集群中的 YAML 工作流。对于外部模式，按照规划控制平面端点中的说明，验证 listener、后端、运行状况检查和可达性。
- 保留 `KubeadmControlPlane.spec.kubeadmConfigSpec.users` 条目，并确保其 `sshAuthorizedKeys` 列表非空（`boot` 用户）。DCS `ignition` 格式会拒绝空的 SSH 密钥列表，因此，即使你不计划通过 SSH 访问 `global` 集群，也必须设置此字段。如果不需要交互式密钥，请参阅解析占位符值了解应提供的值。
- 保留非加密 kubeadm 文件、kubelet 补丁、审计策略和安装程序 RBAC 条目。文件内容（PodSecurity admission 配置、kubelet 补丁和审计策略）以及完整的 `clusterConfiguration`、`preKubeadmCommands`、`postKubeadmCommands`、init 和 join 节点注册补丁，都与业务集群相同。请从完整的 KubeadmControlPlane 配置附录中复制这些内容，或引用其中说明的 `dcs-kubernetes-<major.minor>-files` Secret。下面的连接片段仅展示叠加在该基础配置之上的 `global` 特定字段。
- 对于正常的非 DR 部署，不要设置 `DCSCluster.spec.encryptionProviderConfigRef`。按照附录中的配置保留 `clusterConfiguration.apiServer.extraArgs.encryption-provider-config`：DCS provider 会自行将 `/etc/kubernetes/encryption-provider.conf` 写入每个控制平面节点——第一个机器使用随机密钥，其余机器使用正在运行的 API server 文件——因此该参数可以解析，并且 `global` 集群会启用 etcd 静态数据加密。不要将该文件添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files`；provider 会丢弃该路径的任何条目，因此添加在其中的条目会被静默忽略。请参阅如何交付加密 provider 配置。
- 将 `/var/cpaas` 保留为平台状态。如果需要在滚动替换期间保留磁盘，请在 `DCSIpHostnamePool.spec.pool[].persistentDisk` 中声明它；不要依赖

`DCSMachineTemplate` 模板磁盘来保留状态。

- 对 DCS 本地存储使用具体的 `datastoreName` 值，除非你已确认所选 datastore 集群可以将卷放置在能够运行目标 VM 的主机上。

> **片段范围**
>
> 以下 YAML 是差异片段，不是可以直接应用的完整 manifest。将这些 `global` 特定变更合并到根据 DCS 创建集群参考资料准备的 manifest 中，然后应用完整的 manifest 文件。如果希望从完整文件开始，而不是从片段组装，请在本页末尾调整示例：Huawei DCS 的完整 `global` Manifest。

以下片段展示了 `global` 特定的 Cluster API 连接配置。使用上面的 DCS 创建集群参考资料填写 provider 资源字段。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: global
  namespace: cpaas-system
  labels:
    cluster-type: DCS
    is-global: "true"
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8
s.io/v1beta1
    capi.cpaas.io/resource-kind: DCSCluster
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - ${CLUSTER_CIDR}
    services:
      cidrBlocks:
        - ${SERVICE_CIDR}
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: global-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: DCSCluster
    name: global
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: global-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  replicas: 3
  version: ${K8S_VERSION}
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
```

```
      maxSurge: 0
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: global-cp-template
  kubeadmConfigSpec:
    format: ignition
    clusterConfiguration:
      etcd:
        local:
          serverCertSANs:
            - "${CONTROL_PLANE_VIP}"
            - "${PLATFORM_HOST}"
```

VMware vSphere

在渲染 VMware vSphere `global` manifest 之前，设置其输出路径。

```
export GLOBAL_VSPHERE_YAML="/root/yamls/new-global.yaml"
```

在组装 manifest 之前创建两个凭证 Secret，并将其排除在 manifest 之外。第 5 步使用 `kubectl apply` 应用 manifest，这会将它们的内容记录在每个 Secret 的 `kubectl.kubernetes.io/last-applied-configuration` annotation 中。

- vCenter 凭证，其名称必须为 `global-vsphere-credentials` ——第 7 步的导入 ConfigMap 和 DR 部分都硬编码了这一确切名称。按照在 VMware vSphere 上创建集群中的说明，使用该固定名称创建凭证，并从 `VSphereCluster.spec.identityRef.name` 引用它。
- vSphere CPI 凭证，由 `ClusterResourceSet` 引用。按照同一指南中的说明创建该凭证。

然后，VMware vSphere `global` manifest 必须在 `cpaas-system` namespace 中包含以下资源：

资源	用途
用于控制平面节点的 `VSphereMachineConfigPool`	分配静态主机名、数据中心值、IP 地址、网络设置和可选数据磁盘。
用于控制平面节点的 `VSphereMachineTemplate`	定义 VM 模板、克隆模式、CPU、内存、datastore、文件夹、网络设备和系统磁盘设置。
`KubeadmControlPlane`	引导 Kubernetes 控制平面。将 `spec.version` 设置为 `${K8S_VERSION}`。
`VSphereCluster`	定义 vCenter 端点、指纹、凭证引用和控制平面端点。
`Cluster`	将 Cluster API `Cluster` 连接到 `VSphereCluster` 和 `KubeadmControlPlane`。
用于 vSphere CPI 的 `ClusterResourceSet` 和 `ConfigMap`	API server 可访问后，将 vSphere CPI 资源交付到 `global` 集群。`ClusterResourceSet` 还会引用上面创建的 CPI 凭证 Secret。
用于工作节点的 `VSphereMachineConfigPool`、`VSphereMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment`	创建工作节点。

使用VMware vSphere 基础设施准备准备 vSphere 输入值。以在 VMware vSphere 上创建集群和VMware vSphere Provider为基础参考资料准备 `global` 集群 manifest。创建集群指南通过将 manifest 应用到现有的 `global` 管理集群来创建业务集群；它不会自行创建 `global` 集群。只有在应用以下 `global` 特定要求后，才能复用其中的 vSphere 资源定义：

- 将 `Cluster.metadata.name` 和 `VSphereCluster.metadata.name` 设置为 `global`（基础设施集群与 CAPI 共享 `Cluster` 名称）。为其他每个 CAPI 资源和 provider 资

源添加 `global-` 前缀；下面的连接片段使用 `KubeadmControlPlane.metadata.name: global-kcp`。

- 添加 `Cluster.metadata.labels.is-global: "true"` 和 `Cluster.metadata.labels.cluster-type: VSphere`。
- 添加包含 `${NODE_REGISTRY_ADDRESS}` 的 `Cluster.metadata.annotations["cpaas.io/registry-address"]`。
- 保留平台控制器所需的 VMware vSphere annotations，包括 VMware vSphere 创建集群指南中的网络和 CPI annotations。
- 将 `VSphereMachineTemplate.spec.template.spec.folder` 设置为 `/<datacenter>/vm/global`，以便 operator 在 vCenter 中识别 `global` 集群 VM。在 DR 部署中，为主集群和备用集群使用不同的子文件夹，例如 `/<datacenter>/vm/global/primary` 和 `/<datacenter>/vm/global/standby`。
- 将 `VSphereCluster.spec.identityRef.name` 设置为 `global-vsphere-credentials`。此固定 Secret 名称仅适用于 VMware vSphere `global` 安装路径；非 `global` VMware vSphere 集群遵循通用创建集群指南。
- 在应用 manifest 之前选择控制平面端点模式；之后无法更改。对于外部 LoadBalancer，按照规划控制平面端点中的约定先完成配置和验证。`global` 集群也可以使用 `internal` Self-built VIP：完成 Self-built VIP 先决条件，然后将启用 Self-built VIP 中的 `spec.controlPlaneLoadBalancer` 块添加到 `global` `VSphereCluster` 中。
  Self-built VIP 先决条件命令会从正在运行的 `global` 集群读取 `ModulePlugin/alive`，但在 bootstrap 安装期间该集群尚不存在。请改为从安装软件包中确认其包含 `alive` plugin。
- 设置 `KubeadmControlPlane.spec.kubeadmConfigSpec.format: cloud-config`，或者在 provider 将此 API 字段默认设置为 `cloud-config` 时不设置该字段。`cloud-init` 是处理数据的客户机软件，不是此字段的有效值。
- 保留 release manifest 的 kubeadm 文件，包括 VMware vSphere 的 `/etc/kubernetes/encryption-provider.conf` 文件条目、kubelet 补丁、审计策略和安装程序 RBAC 条目。VMware vSphere 通过 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 交付此文件；不要遵循 DCS 的 `DCSCluster.spec.encryptionProviderConfigRef` 模式。

> 片段范围

以下 YAML 是差异片段，不是可以直接应用的完整 manifest。将这些 `global` 特定变更合并到根据 VMware vSphere 创建集群指南准备的 manifest 中，然后应用完整的 manifest 文件。

以下片段展示了 `global` 特定的 Cluster API 连接配置。使用上面的 VMware vSphere 创建集群参考资料填写 provider 资源字段。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: global
  namespace: cpaas-system
  labels:
    cluster-type: VSphere
    is-global: "true"
    addons.cluster.x-k8s.io/vsphere-cpi: "enabled"
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8
s.io/v1beta1
    capi.cpaas.io/resource-kind: VSphereCluster
    cpaas.io/alb-address-type: ClusterAddress
    cpaas.io/network-type: kube-ovn
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - ${CLUSTER_CIDR}
    services:
      cidrBlocks:
        - ${SERVICE_CIDR}
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: global-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: VSphereCluster
    name: global
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: global-kcp
  namespace: cpaas-system
spec:
  replicas: 3
  version: "${K8S_VERSION}"
  rolloutStrategy:
    type: RollingUpdate
```

```
    rollingUpdate:
      maxSurge: 0
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: VSphereMachineTemplate
      name: global-cp-machine-template
  kubeadmConfigSpec:
    format: cloud-config
    clusterConfiguration:
      etcd:
        local:
          serverCertSANs:
            - "${CONTROL_PLANE_VIP}"
            - "${PLATFORM_HOST}"
```

Huawei Cloud Stack

在渲染 HCS `global` manifest 之前，设置其输出路径。

```
export GLOBAL_HCS_YAML="/root/yamls/new-global.yaml"
```

HCS `global` manifest 必须在 `cpaas-system` 命名空间中包含以下资源：

资源	用途
`Secret`	存储 `accessKey`、`secretKey`、`projectID`、`region`、`externalGlobalDomain` 以及可选的 `schema`。
`HCSMachineConfigPool` for control plane nodes	分配静态主机名、IP 地址以及由池管理的持久磁盘。
`Cluster`	将 Cluster API `Cluster` 连接到 `HCSCluster` 和 `KubeadmControlPlane`。

资源	用途
`HCSCluster`	定义 VPC、子网、安全组、ELB 和身份引用。
`KubeadmControlPlane`	引导 Kubernetes 控制平面。将 `spec.version` 设置为 `${K8S_VERSION}`。
`HCSMachineTemplate` for control plane nodes	定义镜像、规格、可用区、根卷和临时数据卷。
`HCSMachineConfigPool`、`HCSMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment` for workers	创建工作节点。

使用在 Huawei Cloud Stack 上创建集群和 Huawei Cloud Stack 基础设施资源中的 HCS 资源字段。对于 `global` 集群，保留以下额外要求：

- 将 `Cluster.metadata.name` 和 `HCSCluster.metadata.name` 设置为 `global`（基础设施集群共享 CAPI `Cluster` 名称）。为其他每个 CAPI 资源和 provider 资源添加 `global-` 前缀；下面的连接片段使用 `KubeadmControlPlane.metadata.name: global-kcp`。
- 将 `HCSCluster.spec.identityRef.name` 设置为 `${PROVIDER_SECRET_NAME}`。步骤 7 会将此 Secret 导入最终的 `global` 集群。
- 添加 `Cluster.metadata.labels.is-global: "true"` 和 `Cluster.metadata.labels.cluster-type: HCS`。
- 添加带有 `${NODE_REGISTRY_ADDRESS}` 的 `Cluster.metadata.annotations["cpaas.io/registry-address"]`。
- 设置 `KubeadmControlPlane.spec.kubeadmConfigSpec.format: cloud-config`，或者在 provider 将此 API 字段默认设置为 `cloud-config` 时不设置该字段。`cloud-init` 是客户机软件名称，而不是字段值。
- 保留 release manifest 的非加密 kubeadm 文件、kubelet 补丁、审计策略和安装程序 RBAC 条目。

- 对于正常的非 DR 部署，不要将 `/etc/kubernetes/encryption-provider.conf` 添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files`。
- 将 `/var/cpaas` 作为平台状态保留。当其必须在节点替换后继续存在时，在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中声明它；不要依赖 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 作为保留状态。
- 对于 `global` 集群，使用高可用控制平面。单控制平面 HCS 集群仅支持创建，无法升级。

> **片段范围**
>
> 以下 YAML 是差异片段，不是可以直接应用的完整 manifest。将这些 `global` 特定变更合并到根据 HCS 创建集群参考资料准备的 manifest 中，然后应用完整的 manifest 文件。

以下片段展示了 `global` 特定的 Cluster API 连接配置。使用上面的 HCS 创建集群参考资料填写 provider 资源字段。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: global
  namespace: cpaas-system
  labels:
    cluster-type: HCS
    is-global: "true"
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8
s.io/v1beta1
    capi.cpaas.io/resource-kind: HCSCluster
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - ${CLUSTER_CIDR}
    services:
      cidrBlocks:
        - ${SERVICE_CIDR}
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: global-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: HCSCluster
    name: global
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: global-kcp
  namespace: cpaas-system
spec:
  replicas: 3
  version: "${K8S_VERSION}"
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  machineTemplate:
```

```
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: HCSMachineTemplate
      name: global-cp-machine-template
  kubeadmConfigSpec:
    format: cloud-config
    clusterConfiguration:
      etcd:
        local:
          serverCertSANs:
            - "${CONTROL_PLANE_VIP}"
            - "${PLATFORM_HOST}"
```

裸金属

在渲染 Bare Metal `global` manifest 之前，设置其输出路径。

```
export GLOBAL_BAREMETAL_YAML="/root/yamls/new-global.yaml"
```

Bare Metal `global` manifest 必须在 `cpaas-system` 命名空间中包含以下资源：

资源	用途
`MachineRegistration`	为 `global` 主机提供 live ISO 注册端点和首次安装 elemental 配置。
`SeedImage`	构建用于安装 `global` 主机的 bootstrap ISO。此 ISO 属于 bootstrap 生命周期，交接后不再重复使用。
`MachineInventory`	由注册创建的长期主机身份。在应用 CAPI 对象之前，等待预期的清单。
`MachineInventoryPool` for control plane nodes	列出可以支持 `global` 控制平面的确切 `MachineInventory` 名称。
`BaremetalMachineTemplate` for control plane nodes	引用控制平面清单池。

资源	用途
`KubeadmControlPlane`	引导 Kubernetes 控制平面。将 `spec.version` 设置为 `${K8S_VERSION}`。
`BaremetalCluster`	定义控制平面 VIP、VIP 端口、VRID、端点以及可选的加密 provider config Secret 引用。
`Cluster`	将 Cluster API `Cluster` 连接到 `BaremetalCluster` 和 `KubeadmControlPlane`。
可选的 worker `MachineInventoryPool`、`BaremetalMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment`	当拓扑包含工作节点时，创建 `global` 工作节点。

将在裸金属上创建集群、管理裸金属节点和 Bare Metal Provider 用作资源参考。对于 `global` 集群，保留以下额外要求：

- 将 `Cluster.metadata.name` 和 `BaremetalCluster.metadata.name` 设置为 `global`。为其他每个 CAPI、裸金属和 elemental 资源添加 `global-` 前缀。
- 添加 `Cluster.metadata.labels.cluster-type: ProviderBaremetal`。
- 添加带有 `${NODE_REGISTRY_ADDRESS}` 的 `Cluster.metadata.annotations["cpaas.io/registry-address"]`。
- 将 `SeedImage.spec.baseImage` 设置为 `${BOOTSTRAP_REGISTRY_ADDRESS}/tkestack/baremetal-base-image-iso:${OS_IMAGE_TAG}`，即步骤 3 中导入的 ISO 镜像。重新置备镜像来自步骤 3 中设置的 `elemental-image-catalog` 条目；provider 会将该条目的 registry 部分替换为 `cpaas.io/registry-address` annotation，因此主机会从 `${NODE_REGISTRY_ADDRESS}` 拉取该镜像。
- 添加 `Cluster.metadata.annotations["cpaas.io/kube-ovn-join-cidr"]`、`Cluster.metadata.annotations["cpaas.io/sentry-deploy-type"]: Baremetal` 和 `Cluster.metadata.annotations["cpaas.io/alb-address-type"]: ClusterAddress`。

- 设置 `KubeadmControlPlane.spec.kubeadmConfigSpec.format: cloud-config`，或者在 provider 将此 API 字段默认设置为 `cloud-config` 时不设置该字段。生成的数据由节点上的 `cloud-init` 处理。
- 设置 `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0`。裸金属池无法对物理主机进行超额分配。
- 当 release manifest 使用 kube-ovn 时，在 `KubeadmControlPlane` 上保留 `controlplane.cluster.x-k8s.io/skip-kube-proxy: ""`。
- 将 `${CONTROL_PLANE_VIP}` 和 `${PLATFORM_HOST}` 放入 `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.serverCertSANs`。
- 在规划控制平面端点中选择 Bare Metal 端点模式。对于 `Internal`，设置 VIP、端口 `6443` 和唯一的 `vrid`。对于 `External`，设置外部负载均衡器前端地址和端口，并省略 `vrid`。
- 对于正常的非 DR 部署，可以省略 `BaremetalCluster.spec.encryptionProviderConfigRef`。对于 DR 部署，按照可选的灾难恢复部署中的说明进行设置；不要通过将 `/etc/kubernetes/encryption-provider.conf` 添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 来交付它。
- 对于 Bare Metal DR，将 `baremetal.cluster.io/system-agent-auth-scope: global` 添加到用于 bootstrap `global` 主机的 `MachineRegistration` 中。bootstrap 期间不要设置 `baremetal.cluster.io/system-agent-server-url`。bootstrap ISO 必须通过 bootstrap 节点注册；之后交接作业会将 `global` 机器迁移到本地直接 API 端点。
- 如果 `global` VM 或物理主机在 live ISO 启动期间没有 DHCP，请在等待 `MachineInventory` 注册之前，通过主机控制台手动配置 NIC。使用在裸金属上创建集群中所述的相同 NetworkManager 操作步骤，并将示例地址、网关、DNS 和连接名称替换为该主机的值。
- 在 live ISO 启动之前，从固件设置每个 `global` 主机的时钟，并确保主机之间的时间差保持在平台要求的 10 秒以内。请参阅主机时间同步。
- 不要依赖 OS 主机名的副作用。裸金属 provider 会根据 CAPI 和清单对象规范化 kubeadm 节点名称和 provider ID。
- 在 bootstrap 集群中创建的 `SeedImage` 是 bootstrap 工件。交接后，在活动的 `global` 集群上创建任何新的 `MachineRegistration` 或 `SeedImage`。

> **片段范围**
>
> 以下 YAML 是差异片段，不是可以直接应用的完整 manifest。将这些 `global` 特定变更合并到根据 Bare Metal 创建集群参考资料准备的 manifest 中，然后应用完整的 manifest 文件。

以下片段展示了 `global` 特定的 Cluster API 连接配置。使用 Bare Metal 创建集群参考资料填写清单名称、注册配置、镜像引用和可选的工作节点资源。

对于 provider 管理的 Alive，将 `<control-plane-load-balancer-type>` 设置为 `Internal`；对于用户置备的 LoadBalancer，设置为 `External`。使用 `External` 时，移除 `vrid`。有关每种模式的要求，请参阅规划控制平面端点。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: global
  namespace: cpaas-system
  labels:
    cluster-type: ProviderBaremetal
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8
s.io/v1beta1
    capi.cpaas.io/resource-kind: BaremetalCluster
    cpaas.io/kube-ovn-join-cidr: "${KUBE_OVN_JOIN_CIDR}"
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - ${CLUSTER_CIDR}
    services:
      cidrBlocks:
        - ${SERVICE_CIDR}
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: global-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: BaremetalCluster
    name: global
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalCluster
metadata:
  name: global
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    type: <control-plane-load-balancer-type>
    host: ${CONTROL_PLANE_VIP}
    port: 6443
    # Required only for Internal. Remove this field for External.
```

```
    vrid: <unique-vrid>
    # vipMode defaults to nic. Set it explicitly only when the enviro
nment
    # requires another supported mode, such as arp or policy_route.
    # vipMode: nic
  # Required for DR. Omit this field for a normal non-DR deployment u
nless
  # you need to provide a pre-existing encryption-provider.conf.
  # encryptionProviderConfigRef:
  #   name: global-encryption-provider-config
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: MachineInventoryPool
metadata:
  name: global-control-plane-pool
  namespace: cpaas-system
spec:
  clusterName: global
  machineInventories:
    - global-cp-1
    - global-cp-2
    - global-cp-3
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalMachineTemplate
metadata:
  name: global-control-plane-template
  namespace: cpaas-system
spec:
  template:
    spec:
      machineInventoryPoolRef:
        name: global-control-plane-pool
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: global-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  replicas: 3
  version: "${K8S_VERSION}"
```

```
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: BaremetalMachineTemplate
      name: global-control-plane-template
  kubeadmConfigSpec:
    format: cloud-config
    clusterConfiguration:
      etcd:
        local:
          serverCertSANs:
            - "${CONTROL_PLANE_VIP}"
            - "${PLATFORM_HOST}"
```

步骤 5 — 应用 global 清单

将 provider 特定的清单应用到 bootstrap 集群。

Huawei DCS

```
kubectl apply -f "${GLOBAL_DCS_YAML}"
```

VMware vSphere

```
kubectl apply -f "${GLOBAL_VSPHERE_YAML}"
```

Huawei Cloud Stack

```
kubectl apply -f "${GLOBAL_HCS_YAML}"
```

Bare Metal

```
kubectl apply -f "${GLOBAL_BAREMETAL_YAML}"
```

等待 bootstrap 注册生成预期的资源清单，然后再期待 Cluster API 协调取得进展。

```
kubectl -n cpaas-system get machineinventory.elemental.cattle.io
kubectl -n cpaas-system get machineinventorypool
kubectl -n cpaas-system get baremetalcluster,baremetalmachine
```

步骤 6 — 等待控制平面

等待 Cluster API provider 置备机器并启动 Kubernetes 控制平面。

```
kubectl get clusters.cluster.x-k8s.io -n cpaas-system
kubectl get kubeadmcontrolplane -n cpaas-system
kubectl get machines -n cpaas-system
```

当 `KubeadmControlPlane` 报告 `Ready: True`，且 `Cluster` 报告 `Phase: Provisioned` 时，控制平面即已就绪。

步骤 7 — 导入 **Provider** 资源

对于需要额外导入资源的 provider，请在触发安装程序之前，在 `cpaas-system` 命名空间中创建 `dcs-import-extra-resources` ConfigMap。即使 provider 不是 Huawei DCS，ConfigMap 名称仍保留 `dcs` 前缀，以兼容旧版安装程序。

每个 provider 都使用此 ConfigMap 将 IaaS 凭证 `Secret` 导入新的 `global` 集群。对于 VMware vSphere、Huawei Cloud Stack 和 Bare Metal，它还会导入 provider 的 Cluster API 资源；对于 Huawei DCS，这些资源由内置流程迁移，因此正常安装时，DCS ConfigMap 只需包含凭证 `Secret` 条目。正常安装和灾难恢复 `global` 安装都需要 VMware vSphere、Huawei Cloud Stack 和 Bare Metal 的 ConfigMap。

对于灾难恢复安装，通过 Secret 引用提供加密 provider 配置的两个 provider——Huawei DCS（`DCSCluster.spec.encryptionProviderConfigRef`）和 Bare Metal（`BaremetalCluster.spec.encryptionProviderConfigRef`）——还必须通过此 ConfigMap 导入该 Secret。内置的 DCS 迁移流程只迁移 Cluster API 资源，不会复制所引用的 Secret。

Huawei DCS

DCS provider 的 Cluster API 资源由内置流程迁移，因此正常安装时，ConfigMap 只需导入 `DCSCluster.spec.credentialSecretRef.name` 引用的凭证 `Secret`。使用在 `global` 清单中设置的相同 Secret 名称。

```
mkdir -p /root/yamls
cat > /root/yamls/dcs-import-extra-resources.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: dcs-import-extra-resources
  namespace: cpaas-system
data:
  resources.yaml: |
    resources:
    - resource: "secrets"
      names: ["${PROVIDER_SECRET_NAME}"]
      method: kubectl
EOF

kubectl apply -f /root/yamls/dcs-import-extra-resources.yaml
```

对于 DR 安装，请改用以下形式。它会添加 `DCSCluster.spec.encryptionProviderConfigRef.name` 引用的加密 provider `Secret`，该 provider 必须传递到最终的 `global` 集群。`DCS_ENCRYPTION_PROVIDER_SECRET` 来自准备共享 DR 变量；如果未导出该变量，第一行会明确报错，因为否则 heredoc 会将其展开为空的 Secret 名称。

```
mkdir -p /root/yamls
: "${DCS_ENCRYPTION_PROVIDER_SECRET:?export it first — see Prepare Sh
ared DR Variables}"
cat > /root/yamls/dcs-import-extra-resources.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: dcs-import-extra-resources
  namespace: cpaas-system
data:
  resources.yaml: |
    resources:
    - resource: "secrets"
      names:
        - "${PROVIDER_SECRET_NAME}"
        # The name must match the Secret referenced by
        # DCSCluster.spec.encryptionProviderConfigRef.
        - "${DCS_ENCRYPTION_PROVIDER_SECRET}"
      method: kubectl
EOF

kubectl apply -f /root/yamls/dcs-import-extra-resources.yaml
kubectl -n cpaas-system get cm dcs-import-extra-resources -o yaml
```

继续之前，请检查渲染后的 ConfigMap：两个 `names` 条目都必须非空，且第二个条目必须与 `DCSCluster.spec.encryptionProviderConfigRef.name` 完全匹配。

对于 DR，请在安装后验证最终的 `global` 集群中包含已导入的加密 provider Secret。

```
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get secret "${DCS_ENCRYPTION_PROVIDER_SECRET}" \
  -o jsonpath='{.data.encryption-provider\.conf}{"\n"}'
```

VMware vSphere

在触发安装程序之前，创建并应用 VMware vSphere 导入 ConfigMap。正常安装和灾难恢复 `global` 安装都需要此 ConfigMap。`global-vsphere-credentials` Secret 存储 vCenter 用户名和密码，其名称必须与 VMware vSphere `global` 清单中 `VSphereCluster.spec.identityRef.name` 引用的 Secret 名称相同。

```
mkdir -p /root/yamls
cat > /root/yamls/dcs-import-extra-resources.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: dcs-import-extra-resources
  namespace: cpaas-system
data:
  resources.yaml: |
    resources:
    - resource: "vsphereclusters.infrastructure.cluster.x-k8s.io"
      names: ["global"]
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/vsphere
clusters/cpaas-system/"
      method: etcdctl
    - resource: "vspheremachinetemplates.infrastructure.cluster.x-k8
s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/vsphere
machinetemplates/cpaas-system/"
      method: etcdctl
    - resource: "vspheremachines.infrastructure.cluster.x-k8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/vsphere
machines/cpaas-system/"
      method: etcdctl
    - resource: "vspherevms.infrastructure.cluster.x-k8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/vsphere
vms/cpaas-system/"
      method: etcdctl
    - resource: "vspheremachineconfigpools.infrastructure.cluster.x-k
8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/vsphere
machineconfigpools/cpaas-system/"
      method: etcdctl
    - resource: "secrets"
      names: ["global-vsphere-credentials"]
      method: kubectl
EOF

kubectl apply -f /root/yamls/dcs-import-extra-resources.yaml
```

Huawei Cloud Stack

在触发安装程序之前，创建并应用 HCS 导入 ConfigMap。正常安装和灾难恢复 `global` 安装都需要此 ConfigMap。将 `PROVIDER_SECRET_NAME` 设置为 `HCSCluster.spec.identityRef.name` 使用的相同 Secret 名称。

```
mkdir -p /root/yamls
cat > /root/yamls/dcs-import-extra-resources.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: dcs-import-extra-resources
  namespace: cpaas-system
data:
  resources.yaml: |
    resources:
    - resource: "secrets"
      names: ["${PROVIDER_SECRET_NAME}"]
      method: kubectl
    - resource: "hcsclusters.infrastructure.cluster.x-k8s.io"
      names: ["global"]
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/hcsclus
ters/cpaas-system/"
      method: etcdctl
    - resource: "hcsmachinetemplates.infrastructure.cluster.x-k8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/hcsmach
inetemplates/cpaas-system/"
      method: etcdctl
    - resource: "hcsmachineconfigpools.infrastructure.cluster.x-k8s.i
o"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/hcsmach
ineconfigpools/cpaas-system/"
      method: etcdctl
    - resource: "hcsmachines.infrastructure.cluster.x-k8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/hcsmach
ines/cpaas-system/"
      method: etcdctl
EOF

kubectl apply -f /root/yamls/dcs-import-extra-resources.yaml
```

Bare Metal

在触发安装程序之前，创建并应用 Bare Metal 导入 ConfigMap。此 ConfigMap 会导入持久化的裸金属和 elemental owner 资源，全新安装交接流程将使用这些资源枚举已安装的 Global 机器。不得导入 Elemental plan Secret 或 kubeadm bootstrap data Secret。

> **在 DCS API 之前导入**
>
> 在调用 `POST /cpaas-installer/api/config/dcs` 之前创建 `dcs-import-extra-resources`。如果缺少该对象，交接作业可能会使用空的目标列表运行，因为新的 `global` 集群中不包含用于描述 bootstrap `global` 机器的 `BaremetalMachine`、`MachineInventory` 或 `MachineRegistration` 对象。不要通过将这些机器的 plan Secret 或 kubeadm bootstrap data Secret 添加到 ConfigMap 来弥补。

从已协调的 `BaremetalMachine` 对象中收集确切的 Global `MachineInventory` 名称，并确定这些主机使用的 bootstrap `MachineRegistration`。对于 DR，还要收集 `BaremetalCluster.spec.encryptionProviderConfigRef` 引用的 Secret 名称，以便最终的 `global` 集群包含相同的加密 provider 配置。不要导入任意 platform 凭证 Secret 或 MachineRegistration token Secret。

```
kubectl -n cpaas-system get baremetalmachine \
  -l cluster.x-k8s.io/cluster-name=global \
  -o custom-columns='NAME:.metadata.name,INVENTORY:.status.machineInv
entoryRef.name'
kubectl -n cpaas-system get machineregistration.elemental.cattle.io
kubectl -n cpaas-system get baremetalcluster global \
  -o jsonpath='{.spec.encryptionProviderConfigRef.name}{"\n"}'
```

创建 ConfigMap。将 inventory 和 MachineRegistration 占位符替换为上述命令输出的值。对于 DR，还要替换加密 provider Secret 占位符；对于未设置 `BaremetalCluster.spec.encryptionProviderConfigRef` 的非 DR 安装，请移除该 Secret 条目。

```
mkdir -p /root/yamls
cat > /root/yamls/dcs-import-extra-resources.yaml <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: dcs-import-extra-resources
  namespace: cpaas-system
data:
  resources.yaml: |
    resources:
    - resource: "customresourcedefinitions.apiextensions.k8s.io"
      names:
        - baremetalclusters.infrastructure.cluster.x-k8s.io
        - baremetalmachines.infrastructure.cluster.x-k8s.io
        - baremetalmachinetemplates.infrastructure.cluster.x-k8s.io
        - machineinventorypools.infrastructure.cluster.x-k8s.io
        - machineinventories.elemental.cattle.io
        - machineregistrations.elemental.cattle.io
        - seedimages.elemental.cattle.io
      method: kubectl
    - resource: "baremetalclusters.infrastructure.cluster.x-k8s.io"
      names: ["global"]
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/baremet
alclusters/cpaas-system/"
      method: etcdctl
    - resource: "baremetalmachinetemplates.infrastructure.cluster.x-k
8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/baremet
almachinetemplates/cpaas-system/"
      method: etcdctl
    - resource: "baremetalmachines.infrastructure.cluster.x-k8s.io"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/baremet
almachines/cpaas-system/"
      method: etcdctl
    - resource: "machineinventorypools.infrastructure.cluster.x-k8s.i
o"
      etcdKeyBase: "/registry/infrastructure.cluster.x-k8s.io/machine
inventorypools/cpaas-system/"
      method: etcdctl
    - resource: "machineinventories.elemental.cattle.io"
      names:
        - "<global-machine-inventory-1>"
        - "<global-machine-inventory-2>"
```

```
        - "<global-machine-inventory-3>"
      etcdKeyBase: "/registry/elemental.cattle.io/machineinventories/
cpaas-system/"
      method: etcdctl
    - resource: "machineregistrations.elemental.cattle.io"
      names: ["<global-machine-registration>"]
      etcdKeyBase: "/registry/elemental.cattle.io/machineregistration
s/cpaas-system/"
      method: etcdctl
    # Required for DR when BaremetalCluster.spec.encryptionProviderCo
nfigRef is set.
    # The name must match the Secret referenced by BaremetalCluster.
    - resource: "secrets"
      names:
        - "<global-encryption-provider-config-secret>"
      method: kubectl
EOF

kubectl apply -f /root/yamls/dcs-import-extra-resources.yaml
kubectl -n cpaas-system get cm dcs-import-extra-resources -o yaml
```

> **不要导入 Plan 或 Kubeadm Bootstrap Secret**
>
> 安装程序会在批量 etcd 导入完成并创建其 owner 对象之前应用 `method: kubectl` extras。此时导入的 Elemental plan Secret 仍引用 bootstrap `MachineInventory` UID；该 owner UID 并不对应之后导入的 Global `MachineInventory`，因此 Kubernetes 垃圾回收可能会删除该 Secret。不要将 Elemental plan Secret、`KubeadmConfig` 对象或 `Machine.spec.bootstrap.dataSecretName` Secret 添加到此 ConfigMap。

第一版全新安装交接流程会显式处理此顺序。它会验证所有目标 Global `MachineInventory` 对象均已存在，从仍在运行的 bootstrap 集群读取每个源 plan Secret，并使用活动 Global `MachineInventory` UID 作为其控制器 owner 来创建 Global plan Secret。

第一版全新安装交接流程不会使用 `KubeadmConfig` 对象和 kubeadm bootstrap data Secret，不得将其添加到此 ConfigMap。

继续执行步骤 8 之前，请检查渲染后的 ConfigMap：

- 包含 Bare Metal CRD、`BaremetalCluster/global`、Bare Metal 机器资源、确切的 Global `MachineInventory` 对象以及 Global `MachineRegistration`。
- 不包含 `SeedImage` 对象、MachineRegistration token Secret、Elemental plan Secret、`KubeadmConfig` 对象或 kubeadm bootstrap data Secret。
- 对于 DR，仍包含加密 provider Secret，且其名称与 `BaremetalCluster.spec.encryptionProviderConfigRef.name` 完全匹配。

bootstrap `SeedImage` 会生成指向 bootstrap 环境的 ISO；在 `global` 集群完成交接后，它不再是正确的生命周期对象。导入 `seedimages.elemental.cattle.io` CRD 只是为了让新的 `global` 集群识别该 API 类型。

对于 DR，请在安装后验证最终的 `global` 集群中包含已导入的加密 provider Secret。

```
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get secret <global-encryption-provider-config-secret> \
  -o jsonpath='{.data.encryption-provider\.conf}{"\n"}'
```

步骤 8 — 触发平台安装

向嵌入式安装程序 REST API 提交平台安装请求。安装程序会将 Cluster API 资源导入新的 `global` 集群，部署基础 operator，并安装所选插件。

```
export INSTALLER_IP=$(kubectl get pods -n cpaas-system -l service_name=cpaas-installer \
  -o jsonpath='{.items[0].status.podIP}')
```

> **网络范围**
>
> `INSTALLER_IP` 是 bootstrap 集群中嵌入式安装程序的 Pod IP。该端点仅在安装期间使用。

在当前 bootstrap 节点上创建 provider 特定的安装程序配置 JSON 文件，然后将其提交到安装程序端点。此安装路径中的所有 provider 使用相同的端点路径，但请求正文不同。

字段	Huawei DCS	VMware vSphere
端点路径	`/cpaas-installer/api/config/dcs`	`/cpaas-installer/api/config/dcs`
`console.host`	`["${CONTROL_PLANE_VIP}"]`	`["${CONTROL_PLANE_VIP}"]`
`console.globalHost`	平台访问地址	平台访问地址
`cluster.clusterCIDR` 和 `cluster.serviceCIDR`	必需	未设置；集群 CIDR 在 VMware vSpher 单中声明
`cluster.features.ha`	必需，`vip` 为带有 `isThirdParty: true` 的 `${CONTROL_PLANE_VIP}`	必需，`vip` 为带有 `isThirdParty: t` `${CONTROL_PLANE_VIP}`。两种端点模 也在 `VSphereCluster.spec.controlPlane` 中声明
`hostIP`	当前 bootstrap 节点 IP	当前 bootstrap 节点 IP

每个 `console.host` 条目都是裸地址：不带 `https://` 前缀，也不带端口。端口由 `console.httpPort` 和 `console.httpsPort` 指定。

如果在步骤 2 中启动 bootstrap 脚本时使用了外部 platform 镜像 registry，请在请求的 `registry` 对象中，将 `registry.externalAddress` 设置为 registry 地址，以替代 `registry.domain`；不要同时设置两者。保留示例中的 `registry.username` 和 `registry.password`：它们是用于从 registry 拉取镜像的凭证。如果 registry 允许匿名拉取，则省略这两个字段。安装程序不会将镜像或软件包复制到外部 registry，因此目标版本的负载必须已上传；请参阅准备外部平台镜像 Registry。

Huawei DCS

DCS 安装程序请求包含 `global` 控制平面 VIP。使用外部负载均衡器时，这是外部预配的 VIP。使用 `Self-built VIP` 时，请使用 provider 管理的 `alive` 运行时所拥有的保留 VIP。

```
mkdir -p /root/yamls
export INSTALLER_CONFIG_JSON="/root/yamls/installer-config-dcs.json"

cat > "${INSTALLER_CONFIG_JSON}" <<EOF
{
  "basic": {
    "username": "admin@cpaas.io",
    "password": "<base64-platform-admin-password>"
  },
  "registry": {
    "domain": "${REGISTRY_DOMAIN}",
    "username": "<registry-username>",
    "password": "<base64-registry-password>"
  },
  "console": {
    "host": [
      "${CONTROL_PLANE_VIP}"
    ],
    "globalHost": "${PLATFORM_HOST}",
    "httpPort": 80,
    "httpsPort": 443,
    "cert": {
      "selfSigned": {}
    }
  },
  "cluster": {
    "clusterCIDR": "${CLUSTER_CIDR}",
    "serviceCIDR": "${SERVICE_CIDR}",
    "features": {
      "ha": {
        "vip": "${CONTROL_PLANE_VIP}",
        "vport": 6443,
        "isThirdParty": true
      }
    }
  },
  "product": [
    "base",
    "acp"
  ],
  "deployMode": "normal",
  "hostIP": "${HOST_IP}"
}
```

```
EOF

curl -k -X POST "http://${INSTALLER_IP}:8080/cpaas-installer/api/config/dcs" \
  -H 'Content-Type: application/json' \
  -d @"${INSTALLER_CONFIG_JSON}"
```

将 `console.host` 设置为本地 `global` 控制平面 VIP，即 `${CONTROL_PLANE_VIP}`。不要在 `console.host` 中使用 platform 域名；请使用 `console.globalHost` 作为 platform 访问地址。`cluster.features.ha.vip` 使用相同的地址。

VMware vSphere

VMware vSphere 使用与 DCS 相同的安装程序端点路径。请求正文在 `console.host` 和 `cluster.features.ha` 中携带控制平面 VIP；对于外部负载均衡器和 Self-built VIP，方式相同；相同的地址也在 `VSphereCluster.spec.controlPlaneEndpoint.host` 中声明。集群 CIDR 不在此处发送，而是在 VMware vSphere `Cluster` 清单中声明。

```
mkdir -p /root/yamls
export INSTALLER_CONFIG_JSON="/root/yamls/installer-config-vsphere.json"

cat > "${INSTALLER_CONFIG_JSON}" <<EOF
{
  "basic": {
    "username": "admin@cpaas.io",
    "password": "<base64-platform-admin-password>"
  },
  "registry": {
    "domain": "${REGISTRY_DOMAIN}",
    "username": "<registry-username>",
    "password": "<base64-registry-password>"
  },
  "console": {
    "host": [
      "${CONTROL_PLANE_VIP}"
    ],
    "globalHost": "${PLATFORM_HOST}",
    "httpPort": 80,
    "httpsPort": 443,
    "cert": {
      "selfSigned": {}
    }
  },
  "cluster": {
    "features": {
      "ha": {
        "vip": "${CONTROL_PLANE_VIP}",
        "vport": 6443,
        "isThirdParty": true
      }
    }
  },
  "product": [
    "base",
    "acp"
  ],
  "deployMode": "normal",
  "hostIP": "${HOST_IP}"
}
EOF
```

```
curl -k -X POST "http://${INSTALLER_IP}:8080/cpaas-installer/api/conf
ig/dcs" \
  -H 'Content-Type: application/json' \
  -d @"${INSTALLER_CONFIG_JSON}"
```

无论该地址由外部负载均衡器还是 Self-built VIP 提供，都将 `console.host` 和 `cluster.features.ha.vip` 设置为本地 `global` 控制平面 VIP，即 `${CONTROL_PLANE_VIP}`。不要在 `console.host` 中使用 platform 域名；请使用 `console.globalHost` 作为 platform 访问地址。`VSphereCluster.spec.controlPlaneEndpoint.host` 使用相同的地址。

Huawei Cloud Stack

HCS 使用与 DCS 相同的安装程序端点路径，但其请求正文不包含 `cluster.features.ha`。控制平面 VIP 由 `HCSCluster.spec.controlPlaneLoadBalancer` 中声明的 HCS ELB 所拥有。

```
mkdir -p /root/yamls
export INSTALLER_CONFIG_JSON="/root/yamls/installer-config-hcs.json"

cat > "${INSTALLER_CONFIG_JSON}" <<EOF
{
  "basic": {
    "username": "admin@cpaas.io",
    "password": "<base64-platform-admin-password>"
  },
  "registry": {
    "domain": "${REGISTRY_DOMAIN}",
    "username": "<registry-username>",
    "password": "<base64-registry-password>"
  },
  "console": {
    "host": [
      "${CONTROL_PLANE_VIP}"
    ],
    "globalHost": "${PLATFORM_HOST}",
    "httpPort": 80,
    "httpsPort": 443,
    "cert": {
      "selfSigned": {}
    }
  },
  "product": [
    "base",
    "acp"
  ],
  "deployMode": "normal",
  "hostIP": "${HOST_IP}"
}
EOF

curl -k -X POST "http://${INSTALLER_IP}:8080/cpaas-installer/api/conf
ig/dcs" \
  -H 'Content-Type: application/json' \
  -d @"${INSTALLER_CONFIG_JSON}"
```

将 `console.host` 设置为本地 `global` 控制平面 VIP，即 `${CONTROL_PLANE_VIP}`。不要在 `console.host` 中使用 platform 域名；请使用 `console.globalHost` 作为

platform 访问地址。`HCSCluster.spec.controlPlaneLoadBalancer.vipAddress` 使用相同的地址。

Bare Metal

Bare Metal 安装程序请求包含交接后使用的控制平面端点。在 `Internal` 模式下，Alive 提供 Self-built VIP。在 `External` 模式下，User-Provisioned 负载均衡器提供相同的端点，并维护控制平面后端。使用 platform 内置 Registry 时，将 `REGISTRY_DOMAIN` 设置为 `${PLATFORM_HOST}:11443`。保持 `registry.externalAddress` 未设置，以便安装程序部署并填充本地 platform Registry。不要在此字段中使用 bootstrap Registry 地址。

在使用 platform 内置 Registry 的 DR 对中，两个 `global` 集群使用相同的 `${PLATFORM_HOST}:11443` Registry 地址。这样可使两侧的 `ProductBase.spec.registry.address` 和两个 `cpaas.io/registry-address` Cluster 注解保持一致，因此 灵雀云容器平台 **Global Cluster Disaster Recovery** 会将其作为无操作进行复制，而不会使用主集群的值覆盖备用集群的值。每个 `global` 集群仍运行自己的 Registry；platform 域名只需解析到当前处于活动状态的集群。

> **Registry 凭证跟随 platform 域名**
>
> 每个 `global` 集群都有自己的 `cpaas-system/registry-admin` 凭证，但 platform 域名一次只解析到其中一个。将制品推送到 `${PLATFORM_HOST}:11443` 时（例如 provider 软件包或 灵雀云容器平台 **Global Cluster Disaster Recovery** 软件包），请使用当前域名所指向集群的 Registry 凭证进行身份验证。故障切换后，域名会指向另一个集群，因此所使用的凭证也会随之改变。错误的凭证会导致 `unable to retrieve auth token: invalid username/password`，但这并不表示你访问了错误的集群。跨站点的匿名拉取不受影响。

```
mkdir -p /root/yamls
export INSTALLER_CONFIG_JSON="/root/yamls/installer-config-baremetal.
json"

cat > "${INSTALLER_CONFIG_JSON}" <<EOF
{
  "basic": {
    "username": "admin@cpaas.io",
    "password": "<base64-platform-admin-password>"
  },
  "registry": {
    "domain": "${REGISTRY_DOMAIN}",
    "username": "<registry-username>",
    "password": "<base64-registry-password>"
  },
  "console": {
    "host": [
      "${CONTROL_PLANE_VIP}"
    ],
    "globalHost": "${PLATFORM_HOST}",
    "httpPort": 80,
    "httpsPort": 443,
    "cert": {
      "selfSigned": {}
    }
  },
  "cluster": {
    "clusterCIDR": "${CLUSTER_CIDR}",
    "serviceCIDR": "${SERVICE_CIDR}",
    "features": {
      "ha": {
        "vip": "${CONTROL_PLANE_VIP}",
        "vport": 6443,
        "isThirdParty": true
      }
    }
  },
  "product": [
    "base",
    "acp"
  ],
  "deployMode": "normal",
  "hostIP": "${HOST_IP}"
```

```
}
EOF

curl -k -X POST "http://${INSTALLER_IP}:8080/cpaas-installer/api/conf
ig/dcs" \
  -H 'Content-Type: application/json' \
  -d @"${INSTALLER_CONFIG_JSON}"
```

将 `console.host` 设置为本地 `global` 控制平面 VIP，即 `${CONTROL_PLANE_VIP}`。不要在 `console.host` 中使用 platform 域名；请使用 `console.globalHost` 作为 platform 访问地址。`cluster.features.ha.vip` 使用相同的地址。

第三方 **Console** 证书

示例使用自签名 Console 证书。如果环境要求使用第三方证书，请在提交安装程序请求前，将 `console.cert` 替换为 `thirdParty` 块，其中包含以 base64 编码的完整证书链、私钥以及可选的 PKCS#12 值。此证书仅用于 platform HTTPS 入口；不会配置由 `KubeadmControlPlane` 生成的 kube-apiserver 或 etcd 证书。端口 `11443` 上的 Global registry 使用独立管理的 `global-registry-server` 证书链，而不是 `console.cert` 或 `dex.tls`。

DR 证书要求

对于主/备用 Bare Metal `global` DR 部署，不要让两侧分别生成互不相关的自签名证书。请在两侧配置相同的受信任 `thirdParty` platform 证书。其必需的 SAN 是稳定的 `${PLATFORM_HOST}` 域名。默认情况下，不要添加主集群或备用集群控制平面 VIP，也不要添加内部 Service 名称。只有当客户端有意通过该 VIP 直接访问 platform HTTPS 时，才添加 VIP SAN。通过端口 `11443` 访问 Registry 不使用此证书；Registry 证书管理是独立的，不属于此证书步骤。交接给 `https://<control-plane-vip>:6443` 的 Global 主机将验证独立的 kube-apiserver 证书和 CA，而不是 `console.cert` 或 `dex.tls`。

步骤 9 — 监控安装

安装程序接受请求后，安装会经过多个阶段；这些阶段可从 bootstrap 节点观察。Alauda OS 上一个典型的 `global` 集群需要 30–60 分钟；总耗时取决于 IaaS 预配速度、镜像拉取时间以及所选插件数量。

可观察到的阶段

阶段	发生的情况	首先查看的位置
Bootstrap	bootstrap 集群、嵌入式 Registry 和 Cluster API provider 在 bootstrap 节点上运行。已在步骤 2 和步骤 3 中完成。	bootstrap 节点终端；`kubectl get pods -n cpaas-system`
基础设施预配	Cluster API provider 根据 Alauda OS 模板在目标 IaaS 平台上创建 VM。	`kubectl get machines -n cpaas-system`
控制平面引导	`KubeadmControlPlane` 引导第一个控制平面节点，etcd 启动，其他控制平面节点加入。	`kubectl get kubeadmcontrolplane -n cpaas-system`
网络和核心附加组件	CAPI provider 在新集群上协调 Kube-OVN、CoreDNS 和 kube-proxy。	`kubectl --kubeconfig <global-kubeconfig> get pods -n kube-system`
Platform 安装	安装程序将 Cluster API 资源导入新的 `global` 集群，部署基础 operator，并安装所选插件。	安装程序进度 API；安装程序日志
完成	安装程序将请求标记为 `Success`，并将最终集群状态写入 `ClusterModule/global`。	安装程序进度 API；`kubectl --kubeconfig <global-kubeconfig> get clustermodule global`

安装期间的信号

同时查看安装程序进度 API 和安装程序日志。如果其中一项看起来停滞，请直接在 bootstrap 节点上检查底层 Cluster API 资源。

```
# Installer progress and live log
curl "http://${INSTALLER_IP}:8080/cpaas-installer/api/progress"
tail -f /var/cpaas/data/installer.log

# Cluster API resources on the bootstrap node
kubectl get clusters.cluster.x-k8s.io -A
kubectl get kubeadmcontrolplane -A
kubectl get machines -A
```

安装程序日志会记录每次阶段转换。临时错误会按较短的时间间隔重试；持久性错误会保留在日志中，并在进度 API 中显示为停滞阶段。

安装程序报告成功后，检查 `global` 集群。

```
kubectl --kubeconfig <global-kubeconfig> get nodes
kubectl --kubeconfig <global-kubeconfig> get pods -n cpaas-system
kubectl --kubeconfig <global-kubeconfig> get clustermodule global
```

常见停滞情况及查看位置

症状	首先查看的位置	要查找的内容
机器保持在 `Pending`，或未出现	`kubectl describe machine -n cpaas-system <machine>`	机器 `Bootstrap` 和 `Infrastructure` 条件中的 provider 特定故障原因。IaaS 配额、网络和凭证问题会在此处显示。
`KubeadmControlPlane` 未达到 `Ready`	使用新集群 kubeconfig 的 `kubectl get nodes` 和 `kubectl describe kubeadmcontrolplane -n cpaas-system`	第一个控制平面节点上的 etcd 健康状况，以及其余节点的加入进度。
`kube-system` 中的 Pod 保持在 `Pending`，或无法拉取镜像	`kubectl --kubeconfig <global-kubeconfig> describe pod -n kube-system <pod>`	镜像拉取错误通常表示面向节点的 Registry 地址无法从新集群的子网访问。

症状	首先查看的位置	要查找的内容
安装程序进度 API 显示阶段停滞	`/var/cpaas/data/installer.log`	最近的阶段行和最近的错误消息。重试错误会按较短的时间间隔重复出现；持久性错误不会推进。
`ClusterModule/global` 未达到健康阶段	`kubectl --kubeconfig <global-kubeconfig> describe clustermodule global`	`Status.conditions` 会说明哪个模块阻止集群完成。

此处未列出的问题通常与环境特定原因有关。请收集安装程序日志、进度 API 响应以及相关的 `kubectl describe` 输出，然后进行升级处理。

可选的灾难恢复部署

当您为灾难恢复部署主集群和备用 `global` 集群时，请使用本节。在应用每个 `global` 集群的 provider 特定清单之前，请先完成这些附加步骤。

开始之前，请完成双向 DR 网络要求。在两个集群的 VIP 都配置了所需的负载均衡器侦听器，且集群间网络规则允许所需流量双向传输之前，请勿启动任何一侧的安装。正常运行期间的流量方向是从备用集群到主集群；故障转移后方向会反转。

> **Bare Metal Fresh-install Scope**
>
> 本节中的 Bare Metal 拆分身份验证操作步骤仅适用于全新安装主集群和备用 `global` 集群。本节不定义或验证现有 Bare Metal DR 集群对的原地升级或迁移，也不涵盖故障环境的修复或恢复。对于这些生命周期任务，请使用经过单独验证的运维手册。

以下并非每个小节都适用于所有 provider。请完成适用于您基础设施的小节：

小节	Huawei DCS	VMware vSphere	Huawei Cloud Stack	Bare Metal
准备共享 DR 变量	✓	✓	✓	✓
Bare Metal：准备共享 ServiceAccount 签名密钥	—	—	—	✓
添加 DR etcd Server 证书 SAN	✓	✓	✓	✓
添加 provider 特定的 DR 字段	✓	✓	✓	✓
安装主集群和备用集群	✓	✓	✓	✓
Bare Metal：在弃用前进行验证	—	—	—	✓
Bare Metal：应用规则 ConfigMap	—	—	—	✓
安装 etcd-sync	✓	✓	✓	✓
Bare Metal：验证动态规则来源	—	—	—	✓
启动并验证同步	✓	✓	✓	✓

将仅适用于 Bare Metal 的步骤应用于其他 provider，会更改该 provider 不期望出现的对象——尤其是不要将 Bare Metal `KubeadmControlPlane` 签名密钥 `files` 条目添加到 DCS、VMware vSphere 或 Huawei Cloud Stack 清单中。

所有 provider：主集群和备用集群必须使用相同的加密 provider 配置。对于 DCS 和 Bare Metal，provider 特定的集群资源会引用包含 `encryption-provider.conf` 的 Secret；对于 HCS，常规非 DR 部署不会将 `/etc/kubernetes/encryption-provider.conf` 添加到

`KubeadmControlPlane.spec.kubeadmConfigSpec.files`。VMware vSphere 保留发布清单中的 `/etc/kubernetes/encryption-provider.conf` 文件条目。

仅适用于 Bare Metal：主集群和备用集群还必须使用相同的 Kubernetes ServiceAccount 签名密钥，以便主集群上创建的固定 `baremetal-system-agent` 令牌在故障转移后能被备用 API server 接受。组成每个 `global` 集群的机器仍使用各自集群本地的身份。

准备共享 DR 变量

在主集群和备用集群的安装环境中设置相同的加密密钥值。

```
export ENCRYPTION_PROVIDER_CONF="/root/yamls/encryption-provider.conf"
export ENCRYPTION_PROVIDER_SECRET_B64="<base64-shared-etcd-encryption-key
>"
export PRIMARY_CLUSTER_VIP="<primary-ha-vip>"
export STANDBY_CLUSTER_VIP="<standby-ha-vip>"
export DCS_ENCRYPTION_PROVIDER_SECRET="encryption-provider-config"
export BAREMETAL_ENCRYPTION_PROVIDER_SECRET="global-encryption-provider-c
onfig"
export SERVICE_ACCOUNT_ISSUER="https://kubernetes.default.svc.cluster.loc
al"
```

在两个安装环境中创建加密 provider 配置文件。

```
mkdir -p "$(dirname "${ENCRYPTION_PROVIDER_CONF}")"
cat > "${ENCRYPTION_PROVIDER_CONF}" <<EOF_CONF
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
- resources:
  - secrets
  providers:
  - aescbc:
      keys:
      - name: key1
        secret: ${ENCRYPTION_PROVIDER_SECRET_B64}
EOF_CONF
```

Bare Metal：准备共享 ServiceAccount 签名密钥

> **Bare Metal only**
>
> 只有 Bare Metal 需要共享 ServiceAccount 签名密钥。Huawei DCS、VMware vSphere 和 Huawei Cloud Stack 部署请跳过本节——它们共享加密 provider 配置，但两侧各自保留自己的签名密钥。不要将这些 `files` 条目添加到非 Bare Metal 的 `KubeadmControlPlane` 中。

仅生成一次 ServiceAccount 签名密钥，并在主集群和备用 `KubeadmControlPlane` 清单中使用相同的文件。只有两侧使用相同密钥签名时，备用 API server 才会接受在主集群上创建的固定 `baremetal-system-agent` 令牌。

```
mkdir -p /root/global-dr-sa
openssl genrsa -out /root/global-dr-sa/sa.key 2048
openssl rsa -in /root/global-dr-sa/sa.key -pubout -out /root/global-dr-s
a/sa.pub
chmod 0600 /root/global-dr-sa/sa.key
chmod 0644 /root/global-dr-sa/sa.pub

kubectl -n cpaas-system create secret generic global-sa-signing-key \
  --from-file=sa.key=/root/global-dr-sa/sa.key \
  --from-file=sa.pub=/root/global-dr-sa/sa.pub \
  --dry-run=client -o yaml | kubectl apply -f -
```

将以下条目添加到主集群和备用集群的 `KubeadmControlPlane.spec.kubeadmConfigSpec` 中。两侧的文件内容以及 issuer/audience 值必须完全一致。

```
files:
  - path: /etc/kubernetes/pki/sa.key
    owner: root:root
    permissions: "0600"
    contentFrom:
      secret:
        name: global-sa-signing-key
        key: sa.key
  - path: /etc/kubernetes/pki/sa.pub
    owner: root:root
    permissions: "0644"
    contentFrom:
      secret:
        name: global-sa-signing-key
        key: sa.pub
clusterConfiguration:
  apiServer:
    extraArgs:
      service-account-key-file: /etc/kubernetes/pki/sa.pub
      service-account-signing-key-file: /etc/kubernetes/pki/sa.key
      service-account-issuer: https://kubernetes.default.svc.cluster.loca
l
      api-audiences: https://kubernetes.default.svc.cluster.local
  controllerManager:
    extraArgs:
      service-account-private-key-file: /etc/kubernetes/pki/sa.key
```

集群安装完成后，在两侧各选取一个控制平面节点，验证文件和 kubeadm 静态 Pod 参数。

```
sha256sum /etc/kubernetes/pki/sa.key /etc/kubernetes/pki/sa.pub
grep -E 'service-account-issuer|api-audiences|service-account-key-file|se
rvice-account-signing-key-file' \
  /etc/kubernetes/manifests/kube-apiserver.yaml
grep -E 'service-account-private-key-file' \
  /etc/kubernetes/manifests/kube-controller-manager.yaml
```

将 **DR etcd Server** 证书 **SAN** 添加到 **KubeadmControlPlane**

在第 4 步生成的清单中，将您正在安装的集群的控制平面 VIP、平台访问地址和 `etcd.kube-system` 添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.etcd.local.serverCertSANs` 中。每个安装环境使用自己的列表：主集群仅列出 `${PRIMARY_CLUSTER_VIP}`，备用集群仅列出 `${STANDBY_CLUSTER_VIP}`。这些值用于配置 kubeadm 生成的 etcd server 证书；它们独立于平台 `console.cert`，不得复制到其 `thirdParty` SAN 列表中。

Primary installation environment

```
serverCertSANs:
  - "${PRIMARY_CLUSTER_VIP}"
  - "${PLATFORM_HOST}"
  - "etcd.kube-system"
```

Standby installation environment

```
serverCertSANs:
  - "${STANDBY_CLUSTER_VIP}"
  - "${PLATFORM_HOST}"
  - "etcd.kube-system"
```

添加 **provider** 特定的 **DR** 字段

Huawei DCS

在两个安装环境中，分别在 bootstrap 集群中创建加密 provider Secret。该 Secret 必须与 `DCSCluster` 位于同一命名空间，并且必须包含一个名为 `encryption-provider.conf` 的键。

```
kubectl create secret generic "${DCS_ENCRYPTION_PROVIDER_SECRET}" \
  --from-file=encryption-provider.conf="${ENCRYPTION_PROVIDER_CONF}"
\
  -n cpaas-system \
  --dry-run=client -o yaml | kubectl apply -f -
```

在 `DCSCluster.spec` 中添加对该 Secret 的引用。

```
encryptionProviderConfigRef:
  name: encryption-provider-config
```

DCS 使用 `DCSCluster.spec.encryptionProviderConfigRef` 来传递灾难恢复加密 provider 配置。每次构建控制平面 bootstrap 数据时，DCS provider 都会读取此 Secret。对于 DCS DR 路径，不要将 `/etc/kubernetes/encryption-provider.conf` 添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 中。

还必须将此 Secret 包含在第 7 步的 DCS `dcs-import-extra-resources` ConfigMap 中。它不能只留在 bootstrap 集群中，因为移交后的 `global` 集群会保留 `DCSCluster` 对象及其 `encryptionProviderConfigRef`，并且还必须拥有被引用的 Secret，以供后续 provider 调和使用。内置的 DCS 迁移流程会迁移 Cluster API 资源，但不会复制此 Secret。

在两个安装环境中创建 DCS `dcs-import-extra-resources` ConfigMap，并使用第 7 步中 ConfigMap 的 DR 形式，而不是常规形式。将 `PROVIDER_SECRET_NAME` 设置为 `DCSCluster.spec.credentialSecretRef.name` 使用的同一 Secret 名称，并确保 `DCS_ENCRYPTION_PROVIDER_SECRET` 与 `DCSCluster.spec.encryptionProviderConfigRef.name` 保持一致。

> **缺少 Secret 不会立即报错，而是在替换控制平面机器时才会失败**
>
> 设置 `DCSCluster.spec.encryptionProviderConfigRef` 后，DCS provider 会要求存在被引用的 Secret，不会回退到生成或读取配置。不会创建控制平面机器的调和不受影响，因此，如果 Secret 留在 bootstrap 集群中，安装看起来会正常运行，直到需要新的控制平面机器时才会出问题——例如 Kubernetes 升级、`KubeadmControlPlane` 滚动更新或控制平面节点替换。随后，新建的 `Machine` 会停留在 `Provisioning` 中，且不会创建 VM；provider 日志会记录 `failed to create vm: failed to resolve user data: get encryption provider config secret`。
>
> 检查现有 DR 对的两侧；如果缺少 Secret，请补充创建：

```
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system get dc
scluster global \
  -o jsonpath='{.spec.encryptionProviderConfigRef.name}{"\n"}'
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system get se
cret <name>
```

使用安装时所用的同一 `encryption-provider.conf` 重新创建该 Secret，以确保新控制平面机器继续使用运行中的 API 服务器已在使用的密钥。如果该文件已不在 bootstrap 节点上，请从同一集群中正在运行的控制平面节点复制它（`/etc/kubernetes/encryption-provider.conf`）。不同的密钥会导致无法解密 DR 对已有的 etcd 数据。

VMware vSphere

无需引用 `VSphereCluster` 加密 Secret。对于 VMware vSphere，请在主、备用两个安装环境的 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 中保留此文件条目。两侧渲染出的 `/etc/kubernetes/encryption-provider.conf` 内容必须完全一致，包括 provider 顺序、键名和 base64 密钥值。还要在两个安装环境中创建第 7 步的 VMware vSphere `dcs-import-extra-resources` ConfigMap，以便安装程序导入 vSphere 基础设施资源和 `global-vsphere-credentials` Secret。

```
- path: /etc/kubernetes/encryption-provider.conf
  owner: "root:root"
  append: false
  permissions: "0644"
  content: |
    apiVersion: apiserver.config.k8s.io/v1
    kind: EncryptionConfiguration
    resources:
    - resources:
      - secrets
      providers:
      - aescbc:
          keys:
          - name: key1
            secret: ${ENCRYPTION_PROVIDER_SECRET_B64}
```

按照将 DR etcd Server 证书 SAN 添加到 KubeadmControlPlane 中的说明，使用各安装环境自己的 DR `serverCertSANs` 列表。

Huawei Cloud Stack

无需引用 `HCSCluster` 加密 Secret。对于 HCS，请在主、备用两个安装环境的 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 中追加此文件条目。两侧渲染出的 `/etc/kubernetes/encryption-provider.conf` 内容必须完全一致，包括 provider 顺序、键名和 base64 密钥值。

```
- path: /etc/kubernetes/encryption-provider.conf
  owner: "root:root"
  append: false
  permissions: "0644"
  content: |
    apiVersion: apiserver.config.k8s.io/v1
    kind: EncryptionConfiguration
    resources:
    - resources:
      - secrets
      providers:
      - aescbc:
          keys:
          - name: key1
            secret: ${ENCRYPTION_PROVIDER_SECRET_B64}
```

按照将 DR etcd Server 证书 SAN 添加到 KubeadmControlPlane 中的说明，使用各安装环境自己的 DR `serverCertSANs` 列表。

在两个安装环境中创建第 7 步的 HCS `dcs-import-extra-resources` ConfigMap。将 `PROVIDER_SECRET_NAME` 设置为 `HCSCluster.spec.identityRef.name` 使用的同一 Secret 名称。

Bare Metal

在主、备用两个安装环境的 `minialauda` 中创建加密 provider Secret。该 Secret 必须与 `BaremetalCluster` 位于同一命名空间，并且必须包含一个名为 `encryption-provider.conf` 的键。

```
kubectl create secret generic "${BAREMETAL_ENCRYPTION_PROVIDER_SECRET}" \
  --from-file=encryption-provider.conf="${ENCRYPTION_PROVIDER_CONF}" \
  -n cpaas-system \
  --dry-run=client -o yaml | kubectl apply -f -
```

在 `BaremetalCluster.spec.encryptionProviderConfigRef` 中引用该 Secret。

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalCluster
metadata:
  name: global
  namespace: cpaas-system
spec:
  encryptionProviderConfigRef:
    name: global-encryption-provider-config
```

裸金属 provider 会读取此 Secret，并将 `/etc/kubernetes/encryption-provider.conf` 注入生成的控制平面 bootstrap 数据中。对于 Bare Metal DR，不要再手动将该文件添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 中；`BaremetalCluster` 引用才是唯一可信来源。

还必须将此 Secret 包含在第 7 步的 Bare Metal `dcs-import-extra-resources` ConfigMap 中。它不能只留在 bootstrap 集群中，因为移交后的 `global` 集群会保留导入的 `BaremetalCluster` 对象，并且还必须拥有被引用的 Secret，以供后续 provider 调和使用。

按照将 DR etcd Server 证书 SAN 添加到 KubeadmControlPlane 中的说明，使用各安装环境自己的 DR `serverCertSANs` 列表。

还要保留准备共享 ServiceAccount 签名密钥中的共享 ServiceAccount 签名密钥配置。缺少此密钥时，备用 API 服务器无法验证现有主机在故障转移前收到的 `baremetal-system-agent` 令牌。

在两个安装环境中创建第 7 步的 Bare Metal `dcs-import-extra-resources` ConfigMap。该 ConfigMap 必须导入移交所需的持久化裸金属和 Elemental 所有者资源，包括 `BaremetalCluster.spec.encryptionProviderConfigRef` 所引用的加密 provider

Secret。不得导入 `SeedImage`、Elemental plan Secret、`KubeadmConfig` 对象或 kubeadm bootstrap 数据 Secret；只有在目标 Global 的所有 `MachineInventory` 对象都存在后，全新安装的移交流程才会创建初始 Global plan Secret。

两侧的 Bare Metal provider AppRelease 都必须启用 Global 本地身份、共享 workload 身份，以及 Global 机器的直接 kube-apiserver 移交。创建 AppRelease 后、调用安装程序 API 前，在各侧的 bootstrap 集群上应用以下某个 patch。

在主 bootstrap 集群上：

```
kubectl -n cpaas-system patch apprelease cluster-api-provider-baremet
al \
  --type=merge \
  -p '{"spec":{"values":{"handoffHook":{"directAPIServer":true},"elem
ental":{"systemAgent":{"authMode":"shared","splitAuthEnabled":true,"s
erviceAccountName":"baremetal-system-agent","globalServiceAccountNam
e":"baremetal-global-system-agent","sharedAuthReadOnly":false}}}}}'
```

在备用 bootstrap 集群上：

```
kubectl -n cpaas-system patch apprelease cluster-api-provider-baremet
al \
  --type=merge \
  -p '{"spec":{"values":{"handoffHook":{"directAPIServer":true},"elem
ental":{"systemAgent":{"authMode":"shared","splitAuthEnabled":true,"s
erviceAccountName":"baremetal-system-agent","globalServiceAccountNam
e":"baremetal-global-system-agent","sharedAuthReadOnly":true}}}}}'
```

在各自的 bootstrap 集群上验证生效值：

```
export EXPECTED_SHARED_AUTH_READ_ONLY=false # Use true on the standby
bootstrap cluster.

kubectl -n cpaas-system get apprelease cluster-api-provider-baremetal
-o json | \
  jq -e --argjson expected_read_only "${EXPECTED_SHARED_AUTH_READ_ONL
Y}" '
    .spec.values.handoffHook.directAPIServer == true and
    .spec.values.elemental.systemAgent.splitAuthEnabled == true and
    .spec.values.elemental.systemAgent.sharedAuthReadOnly == $expecte
d_read_only
  '
```

此命令成功前不要继续。主集群必须使用 `sharedAuthReadOnly: false`；在主集群到备用集群的同步处于活动状态时，备用集群必须使用 `sharedAuthReadOnly: true`。

生成的 bootstrap AppRelease 值如下：

```
handoffHook:
  controlPlaneVIP: <current-side-control-plane-vip>
  directAPIServer: true
  delivery:
    enabled: true
    mode: always

elemental:
  systemAgent:
    authMode: shared
    splitAuthEnabled: true
    serviceAccountName: baremetal-system-agent
    globalServiceAccountName: baremetal-global-system-agent
  tls:
    agentTLSMode: strict
    caCertSecretName: dex.tls
    caCertSecretKey: ca.crt
```

请将 bootstrap 和最终证书输入分开处理。第 3 步创建的 bootstrap `dex.tls` 有专用的 `ca.crt` 键，且仅用于 `https://<bootstrap-host-ip>:12443`。不要将其导入最终集群。在每个最终 Global 集群中，由安装程序管理的 `dex.tls` 在 `tls.crt` 中包含平台

完整证书链；因此，最终 Global 集群上的 Bare Metal provider 必须使用 `elemental.tls.caCertSecretKey: tls.crt`。

最终 Global 集群不会继承 bootstrap release 值。在每个最终 Global 集群上安装 Bare Metal provider 时，请按安装集群插件中的说明，将以下值及下文的 `sharedAuthReadOnly` 一并设置在 `acp/chart-cluster-api-provider-baremetal` 下该 provider 的 `ModuleInfo` 中的 `spec.valuesOverride`。将 `agentTLSMode` 和 `caCertSecretName` 保持为默认值（`strict` 和 `dex.tls`），不要复制其他 bootstrap 值，例如 `handoffHook.*`：

Final Global provider values

```yaml
elemental:
  tls:
    caCertSecretKey: tls.crt
  systemAgent:
    splitAuthEnabled: true
```

最终 operator 使用此证书链作为其注册端点。直接 API 移交仍会从 `Secret/cpaas-system/baremetal-global-system-agent-token` 的 `ca.crt` 键获取 kube-apiserver CA；不要在 Global 主机的直接 `kubeconfig` 中用 `dex.tls/tls.crt` 替换该 CA。

`elemental.systemAgent.splitAuthEnabled` 是 `elemental-operator` 和移交作业共用的唯一开关。不要单独配置移交身份验证启用开关。使用单一事实来源，可以避免 operator 已将共享 Role 的权限收窄至 workload plan 后，Global 主机仍保留共享令牌。

分别在两侧设置共享 bundle 所有权：在 bootstrap release 中应用上述 patch，并在每个最终 Global 集群的 provider `ModuleInfo` 中设置（请参阅安装集群插件）。每个最终 Global 集群保留自己的值；DR 同步不会复制 provider 的 `ModuleInfo`。活动的主集群会调和 workload 权限，而非活动的备用集群则将同步的共享 bundle 保持为只读：

Primary active Global

```yaml
elemental:
  systemAgent:
    sharedAuthReadOnly: false
```

Standby inactive Global

```
elemental:
  systemAgent:
    sharedAuthReadOnly: true
```

安装使用两组互不重叠的 system-agent 权限：

- `Role/cpaas-system/baremetal-global-system-agent` 仅包含属于本地 `Cluster/global` 的机器所对应的 plan Secret 名称。其 ServiceAccount 和 token Secret 位于本地 `global` 集群中，不得同步到对端。
- `Role/cpaas-system/baremetal-system-agent` 仅包含非 `global` workload plan Secret 名称。其 ServiceAccount、token Secret、Role 和 RoleBinding 会从主集群同步到备用集群。

使用命名空间范围的 `Role` 和 `RoleBinding` 对象，不要使用 `ClusterRoleBinding`。不要授予对整个命名空间的 Secret 访问权限，也不要包含 registry、bootstrap、集群访问凭证或平台凭证 Secret。

> **使用标准 system-agent 名称**
>
> 将 Global 本地对象命名为 `baremetal-global-system-agent` 和 `baremetal-global-system-agent-token`，将 workload 共享对象命名为 `baremetal-system-agent` 和 `baremetal-system-agent-token`。`etcd-sync` 的精确同步和忽略规则使用这些字面名称。除非同时更新并验证每条对应的 `etcd-sync` 规则，且在开始同步前完成验证，否则不要自定义这些名称。

对于此全新 DR 安装，每个最终 Global 集群都必须在 `ConfigMap/cpaas-system/baremetal-system-agent-handoff` 中报告 `data.ready: "true"`。这是清理 bootstrap 的信号，并非完整的验收结果：还要验证运行时端点、令牌、CA、plan 反馈和 RBAC 边界。全新安装不会使用 `RoleBinding/cpaas-system/baremetal-global-system-agent-handoff-bridge`；该对象必须不存在。不要手动 patch 完成信号。

安装主集群和备用集群

对主、备用 `global` 集群分别执行第 1 步至第 9 步。

使用两个独立的 **bootstrap** 节点

对每个 provider 使用两个独立的 bootstrap 节点，一个用于主集群安装，一个用于备用集群安装。不要在两侧重复使用同一个 bootstrap 集群：在移交之前，bootstrap 环境承载着安装程序、内嵌 Registry，以及其所安装的 `global` 集群的 provider 资源；对于 Bare Metal，还包括 AppRelease 对象、Registry Secret、`MachineRegistration`、`SeedImage` 和移交状态。共用一个 bootstrap 环境会污染两次 `global` 安装，并可能导致移交或清理操作作用于错误的一侧。

两侧均使用 provider 特定的安装程序配置差异：

Provider	主集群安装	备用集群安装
Huawei DCS	将 `console.host` 设置为主控制平面 VIP。`cluster.features.ha.vip` 使用相同地址。根据第 7 步创建 DCS `dcs-import-extra-resources` ConfigMap，并确保 `PROVIDER_SECRET_NAME` 与 `DCSCluster.spec.credentialSecretRef.name` 保持一致，且 `DCS_ENCRYPTION_PROVIDER_SECRET` 与 `DCSCluster.spec.encryptionProviderConfigRef.name` 保持一致。	将 `console.hos` `cluster.featu` DCS `dcs-impo` `PROVIDER_SECR` `DCSCluster.sp` 且 `DCS_ENCRYP` `DCSCluster.sp` 持一致。
VMware vSphere	将 `console.host` 设置为主控制平面 VIP。`cluster.features.ha.vip` 和 `VSphereCluster.spec.controlPlaneEndpoint.host` 使用相同地址。根据第 7 步创建 VMware vSphere `dcs-import-extra-resources` ConfigMap，并确保 `global-vsphere-credentials` 与 `VSphereCluster.spec.identityRef.name` 保持一致。	将 `console.hos` `cluster.featu` `VSphereCluste` 相同地址。根据 `extra-resource` `credentials` 与 保持一致。
Huawei Cloud Stack	将 `console.host` 设置为主控制平面 VIP。`HCSCluster.spec.controlPlaneLoadBalancer.vipAddress` 使用相同地址。根据第 7 步创建 HCS `dcs-import-extra-resources` ConfigMap，并确保 `HCS_SECRET_NAME` 与 `HCSCluster.spec.identityRef.name` 保持一致。	将 `console.hos` `HCSCluster.sp` 使用相同地址。 `resources` Con `HCSCluster.sp`
Bare Metal	将 `console.host` 设置为主控制平面 VIP。`cluster.features.ha.vip`、`BaremetalCluster.spec.controlPlaneLoadBalancer.host` 和 `handoffHook.controlPlaneVIP` 使用相同地址。设置 `handoffHook.directAPIServer: true`、	将 `console.hos` `cluster.featu` `BaremetalClus` 和 `handoffHoo` `handoffHook.d`

Provider	主集群安装	备用集群安装
	`elemental.systemAgent.splitAuthEnabled: true` 和 `elemental.systemAgent.sharedAuthReadOnly: false`。根据第 7 步创建 Bare Metal `dcs-import-extra-resources` ConfigMap。	`elemental.sys` `elemental.sys` 据第 7 步创建 Ba ConfigMap。

对于主集群，确保平台域名解析到主控制平面 VIP。在第 8 步中，将 `hostIP` 设置为主 bootstrap 节点 IP，并将 `console.host` 以及上表列出的 provider 字段设置为主控制平面 VIP。

主集群安装成功后，按照 DR 操作步骤将平台域名切换到备用控制平面 VIP。然后安装备用集群。备用集群安装前必须进行此 DNS 切换，因为多个平台资源会使用平台域名进行渲染，因此备用安装程序运行期间，该域名必须解析到备用入口。在备用 bootstrap 节点上执行第 8 步时，将 `hostIP` 设置为备用 bootstrap 节点 IP。将 `console.host` 以及上表列出的 provider 字段设置为备用控制平面 VIP。对于 Bare Metal，保持 `REGISTRY_DOMAIN` 为 `${PLATFORM_HOST}:11443`，即与主集群安装相同的值。从备用 bootstrap 节点上的 `cpaas-installer` Pod 获取 `INSTALLER_IP`；不要复用主 bootstrap 节点的值。

Bare Metal：在移除任一 bootstrap 集群前进行验证

> **仅适用于 Bare Metal**
>
> 从此处到本小节结束的所有内容仅适用于 Bare Metal 路径。Huawei DCS、 VMware vSphere 和 Huawei Cloud Stack 部署在两个安装均报告成功后， 继续执行 安装 etcd-sync。

在安装 `etcd-sync` 之前，验证每个最终 `global` 集群上的 Registry 传播情况。将 `ProductBase.spec.registry.preferPlatformURL` 保持为安装程序生成的值。两侧的 `spec.registry.address` 和两个 Cluster 注解中都必须报告相同的 `${PLATFORM_HOST}:11443` 值；只有该值一致，灵雀云容器平台 **Global Cluster Disaster Recovery** 才能复制这些键而不覆盖任何内容。

```
verify_baremetal_global_registry() {
  kubeconfig=$1
  expected_registry=$2

  test "$(kubectl --kubeconfig "${kubeconfig}" \
    get productbase.product.alauda.io base \
    -o jsonpath='{.spec.registry.address}')" = "${expected_registry}"
  test "$(kubectl --kubeconfig "${kubeconfig}" \
    get cluster.platform.tkestack.io global \
    -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}')" =
\
    "${expected_registry}"
  test "$(kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get cluster.cluster.x-k8s.io global \
    -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}')" =
\
    "${expected_registry}"
  test "$(kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get secret public-registry-credential -o jsonpath='{.data.registry}'
| \
    base64 -d)" = "${expected_registry}"

  # Only deployments that enabled Registry authentication have this Secre
t. On an
  # anonymous Registry it does not exist, and checking it unconditionally
aborts
  # the whole function under `set -e` before the remaining assertions ru
n.
  if kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get secret global-registry-auth >/dev/null 2>&1; then
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get secret global-registry-auth -o jsonpath='{.data.\.dockerconfigj
son}' | \
      base64 -d | jq -e --arg registry "${expected_registry}" \
        '.auths[$registry] != null' >/dev/null
  fi

  kubectl --kubeconfig "${kubeconfig}" get apprelease -A -o json | \
    jq -e --arg registry "${expected_registry}" '
      all(.items[];
        .spec.source.repoURL == $registry and
        .spec.values.global.registry.address == $registry)
    ' >/dev/null
```

```
}

export GLOBAL_REGISTRY_ADDRESS="${PLATFORM_HOST}:11443"

verify_baremetal_global_registry \
  "${PRIMARY_GLOBAL_KUBECONFIG}" "${GLOBAL_REGISTRY_ADDRESS}"
verify_baremetal_global_registry \
  "${STANDBY_GLOBAL_KUBECONFIG}" "${GLOBAL_REGISTRY_ADDRESS}"
```

在移除任一 bootstrap KIND 集群前，先在两侧完成完整的移交检查。在该侧已安装的 Global 机器中，`GLOBAL_HOSTS` 是以空格分隔的管理地址列表。请在能够使用两个 Global kubeconfig 并通过 SSH 连接到这些机器的安全主机上运行此操作。由于运行时连接文件包含 bearer token，请保持 shell 跟踪禁用。

```
set -euo pipefail
set +x

export PRIMARY_GLOBAL_KUBECONFIG="<path-to-primary-global-kubeconfig>"
export STANDBY_GLOBAL_KUBECONFIG="<path-to-standby-global-kubeconfig>"
export PRIMARY_BOOTSTRAP_KUBECONFIG="<path-to-primary-bootstrap-kubeconfi
g>"
export STANDBY_BOOTSTRAP_KUBECONFIG="<path-to-standby-bootstrap-kubeconfi
g>"
export PRIMARY_GLOBAL_HOSTS="<primary-global-host-1> <primary-global-host
-2> <primary-global-host-3>"
export STANDBY_GLOBAL_HOSTS="<standby-global-host-1> <standby-global-host
-2> <standby-global-host-3>"
export PRIMARY_GLOBAL_REGISTRATION_NAME="<primary-global-machine-registra
tion-name>"
export STANDBY_GLOBAL_REGISTRATION_NAME="<standby-global-machine-registra
tion-name>"

verify_global_auth_scope() {
  kubeconfig=$1
  registration_name=$2

  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get machineregistration.elemental.cattle.io "${registration_name}" -o
json | \
  jq -e '
    .metadata.annotations["baremetal.cluster.io/system-agent-auth-scope"]
==
      "global"
  ' >/dev/null

  global_inventory_names="$({
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get baremetalmachines.infrastructure.cluster.x-k8s.io -o json | \
    jq -c '[
      .items[]
      | select(
          .metadata.labels["cluster.x-k8s.io/cluster-name"] == "global"
        )
      | .status.machineInventoryRef.name? // empty
    ] | unique | sort'
  })"
  printf '%s\n' "${global_inventory_names}" | \
```

```
    jq -e 'length > 0' >/dev/null

  while IFS= read -r inventory_name; do
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get machineinventory.elemental.cattle.io "${inventory_name}" -o jso
n | \
    jq -e '
      .metadata.annotations["baremetal.cluster.io/system-agent-auth-scop
e"] ==
        "global"
    ' >/dev/null
  done < <(printf '%s\n' "${global_inventory_names}" | jq -r '.[]')
}

verify_baremetal_handoff() {
  bootstrap_kubeconfig=$1
  kubeconfig=$2
  control_plane_vip=$3
  global_hosts=$4
  registration_name=$5
  expected_endpoint="https://${control_plane_vip}:6443"

  kubectl --kubeconfig "${bootstrap_kubeconfig}" -n cpaas-system \
    wait --for=condition=complete \
    job/baremetal-system-agent-handoff --timeout=30m

  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get configmap baremetal-system-agent-handoff -o json | \
    jq -e '.data.ready == "true"' >/dev/null

  verify_global_auth_scope "${kubeconfig}" "${registration_name}"

  if kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get rolebinding baremetal-global-system-agent-handoff-bridge \
    >/dev/null 2>&1; then
    echo "temporary handoff bridge still exists" >&2
    return 1
  fi

  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system get \
    serviceaccount/baremetal-global-system-agent \
    secret/baremetal-global-system-agent-token \
    role/baremetal-global-system-agent \
    rolebinding/baremetal-global-system-agent >/dev/null
```

```
global_plan_names="$({
  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get role baremetal-global-system-agent -o json | \
  jq -c '[
    .rules[]?
    | select(any(.apiGroups[]?; . == ""))
    | select(any(.resources[]?; . == "secrets"))
    | .resourceNames[]?
  ] | unique | sort'
})"
printf '%s\n' "${global_plan_names}" | jq -e 'length > 0' >/dev/null

while IFS= read -r plan_secret; do
  test "$({
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get secret "${plan_secret}" -o jsonpath='{.type}'
  })" = "elemental.cattle.io/plan"
done < <(printf '%s\n' "${global_plan_names}" | jq -r '.[]')

expected_token_sha="$({
  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get secret baremetal-global-system-agent-token \
    -o jsonpath='{.data.token}' | base64 -d | \
  sha256sum | awk '{print $1}'
})"
expected_ca_sha="$({
  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
    get secret baremetal-global-system-agent-token \
    -o jsonpath='{.data.ca\.crt}' | base64 -d | \
  sha256sum | awk '{print $1}'
})"

for host in ${global_hosts}; do
  actual_endpoint="$({
    ssh "root@${host}" \
      'cat /var/lib/elemental/agent/elemental_connection.json' | \
    jq -r '.kubeConfig' | \
    awk '$1 == "server:" {print $2; exit}'
  })"
  test "${actual_endpoint}" = "${expected_endpoint}"

  actual_namespace="$({
    ssh "root@${host}" \
```

```
        'cat /var/lib/elemental/agent/elemental_connection.json' | \
      jq -r '.namespace'
    })"
    test "${actual_namespace}" = "cpaas-system"

    actual_plan_secret="$({
      ssh "root@${host}" \
        'cat /var/lib/elemental/agent/elemental_connection.json' | \
      jq -r '.secretName'
    })"
    printf '%s\n' "${global_plan_names}" | \
      jq -e --arg name "${actual_plan_secret}" 'index($name) != null' \
      >/dev/null

    actual_token_sha="$({
      ssh "root@${host}" \
        'cat /var/lib/elemental/agent/elemental_connection.json' | \
      jq -r '.kubeConfig' | \
      awk '$1 == "token:" {print $2; exit}' | \
      tr -d '\r\n' | sha256sum | awk '{print $1}'
    })"
    test "${actual_token_sha}" = "${expected_token_sha}"

    actual_ca_sha="$({
      ssh "root@${host}" \
        'cat /var/lib/elemental/agent/elemental_connection.json' | \
      jq -r '.kubeConfig' | \
      awk '$1 == "certificate-authority-data:" {print $2; exit}' | \
      base64 -d | sha256sum | awk '{print $1}'
    })"
    test "${actual_ca_sha}" = "${expected_ca_sha}"

    ssh "root@${host}" \
      "grep -Fq -- '${expected_endpoint}' /oem/elemental-system-agent.yam
l"
    ssh "root@${host}" \
      'test "$(stat -c %a /oem/elemental-system-agent.yaml)" = 600'
    ssh "root@${host}" \
      'grep -Fqx -- "CATTLE_AGENT_STRICT_VERIFY=\"true\"" /etc/rancher/el
emental/agent/envs'
    ssh "root@${host}" \
      'systemctl is-active --quiet elemental-system-agent.service'
  done
}
```

```
verify_baremetal_handoff \
  "${PRIMARY_BOOTSTRAP_KUBECONFIG}" "${PRIMARY_GLOBAL_KUBECONFIG}" \
  "${PRIMARY_CLUSTER_VIP}" \
  "${PRIMARY_GLOBAL_HOSTS}" "${PRIMARY_GLOBAL_REGISTRATION_NAME}"
verify_baremetal_handoff \
  "${STANDBY_BOOTSTRAP_KUBECONFIG}" "${STANDBY_GLOBAL_KUBECONFIG}" \
  "${STANDBY_CLUSTER_VIP}" \
  "${STANDBY_GLOBAL_HOSTS}" "${STANDBY_GLOBAL_REGISTRATION_NAME}"
```

还要验证两侧 Global 本地的权限边界。在首个非 `global` workload plan 存在之前，共享的 `baremetal-system-agent` ServiceAccount、token Secret、Role 和 RoleBinding 在两个集群上都可能不存在。不要仅为满足 Global 安装检查而手动创建或复制该资源集；当首个共享范围的注册被协调时，活动 operator 会创建该资源集，随后由 `etcd-sync` 将其复制到备用集群。

```
verify_global_local_system_agent_rbac() {
  kubeconfig=$1

  kubectl --kubeconfig "${kubeconfig}" -n cpaas-system get \
    serviceaccount/baremetal-global-system-agent \
    secret/baremetal-global-system-agent-token \
    role/baremetal-global-system-agent \
    rolebinding/baremetal-global-system-agent >/dev/null

  local_plan_names="$({
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get role baremetal-global-system-agent -o json | \
    jq -c '[
      .rules[]?
      | select(any(.apiGroups[]?; . == ""))
      | select(any(.resources[]?; . == "secrets"))
      | .resourceNames[]?
    ] | unique | sort'
  })"
  printf '%s\n' "${local_plan_names}" | jq -e 'length > 0' >/dev/null

  expected_local_plan_names="$({
    kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
      get machineinventories.elemental.cattle.io -o json | \
    jq -c '[
      .items[]
      | select(
          .metadata.annotations["baremetal.cluster.io/system-agent-auth-s
cope"] ==
            "global"
        )
      | .status.plan.secretRef.name? // empty
    ] | unique | sort'
  })"
  test "${local_plan_names}" = "${expected_local_plan_names}"

  while IFS= read -r plan_secret; do
    test "$({
      kubectl --kubeconfig "${kubeconfig}" -n cpaas-system \
        get secret "${plan_secret}" -o jsonpath='{.type}'
    })" = "elemental.cattle.io/plan"

    test "$({
```

```
      kubectl --kubeconfig "${kubeconfig}" \
        --as=system:serviceaccount:cpaas-system:baremetal-global-system-a
gent \
        -n cpaas-system auth can-i get "secret/${plan_secret}"
    })" = yes
    test "$({
      kubectl --kubeconfig "${kubeconfig}" \
        --as=system:serviceaccount:cpaas-system:baremetal-global-system-a
gent \
        -n cpaas-system auth can-i patch "secret/${plan_secret}"
    })" = yes
    test "$({
      kubectl --kubeconfig "${kubeconfig}" \
        --as=system:serviceaccount:cpaas-system:baremetal-system-agent \
        -n cpaas-system auth can-i get "secret/${plan_secret}"
    })" = no
  done < <(printf '%s\n' "${local_plan_names}" | jq -r '.[]')

  for verb in list create delete; do
    test "$({
      kubectl --kubeconfig "${kubeconfig}" \
        --as=system:serviceaccount:cpaas-system:baremetal-global-system-a
gent \
        -n cpaas-system auth can-i "${verb}" secrets
    })" = no
  done

  test "$({
    kubectl --kubeconfig "${kubeconfig}" \
      --as=system:serviceaccount:cpaas-system:baremetal-global-system-age
nt \
      -n cpaas-system auth can-i get secret/global-registry-auth
  })" = no
}

verify_global_local_system_agent_rbac "${PRIMARY_GLOBAL_KUBECONFIG}"
verify_global_local_system_agent_rbac "${STANDBY_GLOBAL_KUBECONFIG}"
```

不要仅凭 `ready` 信号判定验收通过。如果运行时 token 或 CA 不匹配、持久化 OEM 文件中的 bootstrap 端点过期、plan 探测失败或权限边界不正确，则即使 `data.ready` 为 `true`，全新移交也尚未完成。

备用集群安装成功后，将平台域名切回主入口。首次同步期间，主集群是活动源。

Bare Metal：应用 Rule ConfigMap

> **仅适用于 Bare Metal**
>
> 只有 Bare Metal 需要自定义 `etcd-sync` 规则源。Huawei DCS、VMware vSphere 和 Huawei Cloud Stack 部署可跳过本节，直接继续执行 安装 etcd-sync；其安装顺序不影响结果。

在安装 `etcd-sync` 之前，先在主集群和备用集群上应用此 ConfigMap。

将相同的 `baremetal-dr-rules` ConfigMap 应用到主集群和备用集群。使用 Bare Metal etcd-sync Rules 中的模板。将 plan 占位符替换为两个集群中当前所有 Global plan Secret 名称。两侧的移交都完成后，从 `MachineInventory.status.plan.secretRef.name` 读取这些名称；全新移交会创建 Global plan Secret，因此提前收集的名称已过期。插件 chart 会提供其余 Bare Metal 规则。每当添加或替换 Global plan Secret 时，都要更新此 ConfigMap。

> **在镜像启动前创建规则源**
>
> 如果安装 `etcd-sync` 时已设置 `active_cluster_vip` 和活动集群 token，同步路径会立即开启。如果 Bare Metal 规则源尚不存在，首次同步将不会应用 Global plan Secret 排除规则，并会用主集群的 Secret 覆盖备用集群的 plan Secret——这是 故障排查 中列出的 `A Global plan is overwritten` 故障模式。随后，备用 Global 主机将收到属于主集群的 plans。
>
> 先创建 ConfigMap 不会产生副作用：插件安装之前，它只是一个带标签的普通 ConfigMap，不会被任何组件读取。如果必须先安装插件，请将 `active_cluster_vip` 留空，仅在规则源报告 `accepted` 后再设置。

安装 etcd-sync (ACP 4.4.0+)

仅在当前备用 Global 上安装 v4.4.0 或更高版本的插件。不要在两侧同时安装或运行镜像。

安装前，将平台域名切回主控制平面 VIP——安装备用集群时，域名曾指向备用集群。主集群是首次同步的活动源；插件启动时会解析平台域名，如果域名指向的集群本身运行着 `etcd-sync`，插件将拒绝启动。若域名仍指向备用集群，即将执行的安装会触发此检查。

插件包发布为 `etcd-sync`，而不是 `global-etcd-sync`：从软件包服务器下载 `packages/etcd-sync/<minor>/etcd-sync.amd64.<version>.tgz`。它不包含在安装程序

bundle 中，也不在 bootstrap Registry 内，因此应像其他插件包一样将其上传到目标集群。

配置如下：

参数	值
活动 Global 集群 VIP	活动 `global` 集群的控制平面 VIP
活动 Global 集群 ETCD Endpoints	`https://<active-control-plane-vip>:2379`；除非负载均衡器转发端口 `2379`
备用集群 ETCD Endpoints	默认值
活动 Global 集群 Token Secret	`etcd-sync-active-cluster-token`
数据检查间隔	默认值
打印详细日志	除故障排查外禁用

安装前，在备用集群上创建 token Secret。其值为活动集群 `cpaas-system/k8sadmin` ServiceAccount 的 bearer token：

```
# On the active cluster:
kubectl -n cpaas-system get secret k8sadmin -o jsonpath='{.data.token}' |
base64 -d

# On the standby cluster, with the value copied from above:
ACTIVE_CLUSTER_TOKEN='<paste-the-token-from-the-active-cluster>'
kubectl -n cpaas-system create secret generic etcd-sync-active-cluster-to
ken \
  --from-literal=token="${ACTIVE_CLUSTER_TOKEN}" \
  --dry-run=client -o yaml | kubectl apply -f -
```

安装期间，`etcd-sync-bootstrap` Job 会在 `etcd-sync` Deployment 启动之前运行。只有在 Job 准备好 `remote-etcd-ca`、`remote-etcd-issuer` 和 `remote-etcd-client` 后，插件安装才会继续。

Bare Metal：验证动态规则源

开启同步路径前，确认规则源报告 `accepted`，且出现在 `ConfigMap/etcd-sync-rule-snapshot` 中：

```
kubectl --kubeconfig <standby-kubeconfig> -n cpaas-system \
  get configmap baremetal-dr-rules \
  -o jsonpath='{.metadata.annotations.etcd-sync\.cpaas\.io/status}{"\n"}'
kubectl --kubeconfig <standby-kubeconfig> -n cpaas-system \
  get configmap etcd-sync-rule-snapshot \
  -o jsonpath='{.data.snapshot\.json}'
```

被拒绝的规则源不会处于活动状态。

启动并验证同步

触发一次 etcd-sync 监控检查。收到 HTTP 425 时重试，直到检查获准执行，然后要求遗漏键和多余键均为零。确认共享 workload system-agent 资源集要么在两侧都不存在，要么在两侧都完整存在；只在一侧存在或部分存在均视为失败。

不要仅凭 AppRelease 就绪状态作为 DR 验收结果。有关故障切换顺序和故障切换后的测试，请继续阅读 Global Cluster Disaster Recovery。

重启需要重新加载 DR 和端点配置的 Pod。在主控制平面节点和备用控制平面节点上运行相同的命令。

```
sudo kubectl delete po -n cpaas-system -l 'service_name in (alertmanager,
vmselect,vminsert)'
sudo kubectl delete po -n cpaas-system -l service_name=cpaas-elasticsearc
h
sudo kubectl delete po -n cpaas-system -l service_name=cluster-transforme
r
```

有关安装后的 DR 生命周期，请参阅 Global Cluster Disaster Recovery。

验证

安装程序报告完成后，请按照 验证 中的说明，确认 `global` 集群运行正常，其中包括 Alauda OS 检查。

如果 `global` 集群安装在裸金属上，请接下来完成将 global 集群准备为管理集群。这是弃用 bootstrap 集群之前的必要步骤；交接完成后重新配置 `global` 节点时，会从 `global` Registry 拉取该节点上传的基础映像。

弃用 Bootstrap 集群

安装程序报告成功后，并且在裸金属环境中完成将 global 集群准备为管理集群后，`global` 集群便不再依赖 bootstrap 集群。常规验证以及下面针对特定 provider 的交接门槛均通过后，从 bootstrap 节点移除临时 bootstrap 集群——只删除本地 bootstrap 集群（`minialauda`）及其 KIND 容器网络。

> **裸金属：必须通过最终交接门槛**
>
> 不要仅因为 `minialauda` 处于 Ready 状态、安装程序报告成功，或 `baremetal-system-agent-handoff` Job 处于 `Complete` 状态，就移除 `KubeadmControlPlane`。每台已导入的 Global 主机都必须先通过最终 `global` 集群执行的探测，之后才能停用 bootstrap 端点。
>
> 从新建的 `global` 集群读取最终交接记录：
>
> ```
> kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
>   get configmap baremetal-system-agent-handoff \
>   -o jsonpath='{.data.ready}{"\n"}'
> ```
>
> 完成状态 ConfigMap 恰好包含一个键：`ready`。它仅用作安装程序清理门槛：记录交接和 global 侧探测已完成，不代表端点、TLS 输入或各主机最终使用的身份状态。不要手动修改 `data.ready`，也不要单独将其视为身份验证分离正确的证明——以下检查才能确认这一点。
>
> 只有满足以下所有条件后，才能继续：
>
> - 当前安装程序交接钩子使用已部署的 provider 配置运行了该 Job，且 Job 成功完成。
> - 在最终的 `global` 集群中，`data.ready` 为 `true`。
> - 在最终的 `global` 集群中，每个已导入的目标 `MachineInventory` 均报告 `status.plan.state: Applied`。这是 global 侧探测结果，也是证明每台主机现在都向新集群报告的检查项。

- 最终集群中存在每个已导入的目标对象，以及该对象所引用的计划 Secret。
- 最终集群中的 system-agent 身份与已部署的模式一致，并且临时跨身份 RoleBinding 已被移除。

使用以下命令检查最后三项条件：

```
kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get machineinventories.elemental.cattle.io \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.pla
n.state}{"\n"}{end}'

kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get serviceaccount,secret,role,rolebinding \
  | grep -E 'baremetal-(global-)?system-agent'

kubectl --kubeconfig <global-kubeconfig> -n cpaas-system \
  get rolebinding baremetal-global-system-agent-handoff-bridge
```

对于 `elemental.systemAgent.splitAuthEnabled: false`，最终集群仅包含共享的 `baremetal-system-agent` 身份。对于 `true`，集群还包含 Global-local `baremetal-global-system-agent` 身份，并且两个 Role 不得指定同一个计划 Secret。最后一条命令必须报告 `NotFound`。

如果当前 Job 失败或超时、目标集不完整、缺少被引用的计划 Secret，或探测计划不是 `Applied`，请保持 `minialauda` 运行，并检查交接 Job 和 provider 日志。

确认 DCS Secret 已到达 global 集群

安装期间，你在步骤 7 中创建的 `dcs-import-extra-resources` ConfigMap 会将 DCS API 凭证 `Secret` 复制到 `global` 集群——它会导入 `DCSCluster.spec.credentialSecretRef` 中指定的 Secret。移除 bootstrap 节点之前，请确认该 Secret 已存在：`kubectl --kubeconfig <global-kubeconfig> get secret <name> -n cpaas-system`。如果缺少，请先将其复制过去——否则，你在 `global` 集群上安装的 DCS provider 将没有 DCS API 凭证，无法协调资源（例如，之后扩容会失败）。

对于 DR 安装，请对 `DCSCluster.spec.encryptionProviderConfigRef` 中指定的 Secret 执行相同检查。`DCSCluster` 对象会在交接后保留该引用，因此 bootstrap 集群移除后，如果缺少该 Secret，之后每次创建控制平面机器都会失败——请参阅添加特定于 provider 的 DR 字段。

> **不要通过删除 Cluster API 对象来清理**
>
> 不要运行 `kubectl delete cluster global`，也不要将删除 `Cluster`、`KubeadmControlPlane` 或 provider 基础设施对象作为清理步骤。安装完成后，这些对象会管理正在运行的 `global` 控制平面机器；删除它们会连带删除控制平面 VM，并破坏刚安装的集群。弃用操作仅限于在 bootstrap 节点上移除本地 bootstrap 集群（其 KIND 容器）；请保留 Cluster API 对象。

后续步骤

- 在创建业务集群之前，请先在 `global` 集群上上传并安装目标 IaaS 平台的基础设施 provider 和 Cluster Bootstrap Provider，无论 `global` 集群运行在哪个 IaaS 平台上：请参阅准备。对于 DR 集群对，请在两个 `global` 集群上都执行此操作。
- 配置业务集群的基础设施资源：请参阅基础设施资源。
- 创建第一个业务集群：请参阅在 Alauda OS 上创建。
- 规划灾难恢复：请参阅 Global 集群灾难恢复。

示例：Huawei DCS `global` 完整清单

这是一个完整的单文件清单，用于在 Huawei DCS 上创建具有三个副本的控制平面 `global` 集群。它包含步骤 4 中所述的同一组资源，已整合在一起，因此你无需从不同页面合并片段。示例中使用的值仅用于文档演示：请替换每个 `<placeholder>`，并复用在步骤 1 中导出的 `${...}` 变量。在步骤 5 中应用该清单。

此示例适用于非 DR 集群。为避免维护两份副本，这里不再重复 `KubeadmControlPlane` `kubeadmConfigSpec` 主体——它与业务集群相同，取自完整的 KubeadmControlPlane 配置附录，资源 3 中内联注明了两个 `global` / 非 DR 差异。

该清单不会创建 DCS API 凭证 `Secret`。该 Secret 已在上述先决条件中创建，并在此处通过 `DCSCluster.spec.credentialSecretRef` 按名称引用。步骤 7 会通过 `dcs-import-extra-resources` ConfigMap 将其复制到 `global` 集群；你必须确保其中的 `names` 列表与同一 Secret 名称保持一致。

应用前：在 DCS 上准备以下内容

清单会引用以下内容，但不会创建它们。请先按顺序准备好：

1. **DCS API** 访问权限及据此创建的凭证 **Secret**——端点（`https://<host>:7443`）、一个 `administrator` 用户及其密码，以及站点 ID。四项信息都需写入凭证 Secret；如果该字段留空，provider 会从中读取站点 ID 并将其写入 `DCSCluster.spec.site`，因此下面的清单不会设置该字段。
   按照云凭证中的说明，在 `cpaas-system` 中创建名为 `${PROVIDER_SECRET_NAME}` 的 Secret。不要将其编写为 YAML 后再应用：`kubectl apply` 会将密码记录在 Secret 的 `kubectl.kubernetes.io/last-applied-configuration` 注解中。
   `global` 安装还要求添加清单中所有其他资源都带有的集群名称标签：

   ```
   kubectl label secret "${PROVIDER_SECRET_NAME}" --namespace cpaas-system
   \
     --overwrite cpaas.io/cluster-name=global
   ```

2. **VM** 模板——上传 Alauda OS 镜像并据此创建 VM 模板；记录其名称，以供 `vmTemplateName` 使用。请使用 `4.2.1+` 模板，以便在替换节点期间可以分离并重新挂载 `/var/cpaas` 持久磁盘。请参阅机器模板。

3. 计算集群、分布式虚拟交换机、端口组和数据存储——选择目标 DCS 计算集群、一个分布式虚拟交换机、一个端口组，以及具有足够可用容量的数据存储。它们分别对应 `resource`、`dvSwitchName`、`portGroupName` 和 `datastoreName`。请参阅DCS 平台容量与放置。

4. **IP** 和 **API** 端点——为控制平面准备三个可用的节点 IP（包含网关和 DNS）；它们对应 `DCSIpHostnamePool`。然后选择一个符合控制平面端点约定的外部 LoadBalancer，并将 TCP `6443` 转发到这三个节点；或者为 Self-built VIP 选择同一 Layer 2 网络中的独立 IPv4 VIP。所选端点的主机和端口分别对应 `DCSCluster.spec.controlPlaneLoadBalancer` 和 `controlPlaneEndpoint`。请参阅在 Huawei DCS 上创建集群。

5. 需要查阅的版本和 **ID**—— `${K8S_VERSION}`，以及从 `cpaas.io/dcs-vm-template` ConfigMap 获取的 CoreDNS 和 etcd 镜像标签（请参阅解析占位值）；从OS 支持矩阵获取的 kube-ovn chart 版本；以及你在步骤 1 中导出的 registry 地址、VIP 和 CIDR 值。

准备好这些内容后，即可应用下面的清单。示例使用外部 LoadBalancer。如果选择 `Self-built VIP`，请在创建 `global` 集群之前，先应用清单之后所示的内部模式替换项。

```
---
# 1. Control-plane IP / hostname pool (one entry per control-plane replic
a).
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: global-cp-pool
  namespace: cpaas-system
  labels:
    cpaas.io/cluster-name: "global"
spec:
  pool:
  # /var/cpaas holds platform state and must survive node replacement, so
it is
  # declared here as a persistentDisk bound to the IP slot (not as a
  # DCSMachineTemplate disk). Requires a DCS VM template 4.2.1+ and maxSu
rge: 0.
  - ip: "192.0.2.11"
    mask: "24"
    gateway: "192.0.2.1"
    dns: "192.0.2.2"
    hostname: "global-cp-1"
    machineName: "global-cp-1"
    persistentDisk:
    - {slot: 0, quantityGB: 100, datastoreName: <datastore-name>, path: /
var/cpaas, format: xfs, mountOptions: [defaults]}
  - ip: "192.0.2.12"
    mask: "24"
    gateway: "192.0.2.1"
    dns: "192.0.2.2"
    hostname: "global-cp-2"
    machineName: "global-cp-2"
    persistentDisk:
    - {slot: 0, quantityGB: 100, datastoreName: <datastore-name>, path: /
var/cpaas, format: xfs, mountOptions: [defaults]}
  - ip: "192.0.2.13"
    mask: "24"
    gateway: "192.0.2.1"
    dns: "192.0.2.2"
    hostname: "global-cp-3"
    machineName: "global-cp-3"
    persistentDisk:
    - {slot: 0, quantityGB: 100, datastoreName: <datastore-name>, path: /
```

```
var/cpaas, format: xfs, mountOptions: [defaults]}
---
# 2. Control-plane VM spec.
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: global-cp-template
  namespace: cpaas-system
  labels:
    cpaas.io/cluster-name: "global"
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      # Places the cloned VMs in a DCS compute cluster.
      resource:
        type: cluster
        name: <dcs-cluster-name>
      vmConfig:
        dvSwitchName: <dvswitch-name>
        portGroupName: <port-group-name>
        dcsMachineCpuSpec: {quantity: 16}
        dcsMachineMemorySpec: {quantity: 32768}   # MB
        dcsMachineDiskSpec:
        - {quantity: 0,   datastoreName: <datastore-name>, systemVolume:
true}
        - {quantity: 10,  datastoreName: <datastore-name>, path: /var/li
b/etcd,       format: xfs}
        - {quantity: 100, datastoreName: <datastore-name>, path: /var/li
b/kubelet,    format: xfs}
        - {quantity: 100, datastoreName: <datastore-name>, path: /var/li
b/containerd, format: xfs}
        # /var/cpaas is intentionally NOT a template disk — it is declare
d as a
        # persistentDisk on the IP pool above so it survives node replace
ment.
      ipHostPoolRef:
        name: global-cp-pool
---
# 3. Control plane.
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: global-kcp
```

```
  namespace: cpaas-system
  labels:
    cpaas.io/cluster-name: "global"
  annotations:
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  replicas: 3
  version: ${K8S_VERSION}
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate: {maxSurge: 0}
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: global-cp-template
  kubeadmConfigSpec:
    format: ignition
    users:
    - name: boot
      sshAuthorizedKeys:
      - "ssh-ed25519 AAAA...replace-with-your-public-key... global-boot"
    # The rest of kubeadmConfigSpec — files, clusterConfiguration,
    # preKubeadmCommands, postKubeadmCommands, initConfiguration,
    # joinConfiguration — is identical to a workload cluster, so it is no
t
    # duplicated here. Take it verbatim from the Complete KubeadmControlP
lane
    # Configuration appendix in the DCS create-cluster guide
    # (#complete-kubeadmcontrolplane-configuration). The three large file
s
    # (psa-config.yaml, control-plane-kubelet-patch.json, audit-policy.ya
ml) may
    # use contentFrom the dcs-kubernetes-<major.minor>-files Secret. Appl
y these
    # global / non-DR deltas to that body:
    #   1. Add clusterConfiguration.etcd.local.serverCertSANs:
    #        ["${CONTROL_PLANE_VIP}", "${PLATFORM_HOST}"]
    #        A DR pair uses a different list — see Optional Disaster Reco
very
    #        Deployment. etcd.kube-system belongs to that list, not this
one.
```

```
    #   2. Omit the /etc/kubernetes/encryption-provider.conf files entry: the
    #       provider writes that file on control-plane nodes and drops any
    #       files entry for it. Keep apiServer.extraArgs.encryption-provider-
    #       config — see the note after this example.
---
# 4. DCS infrastructure cluster.
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSCluster
metadata:
  name: "global"
  namespace: cpaas-system
  labels:
    cpaas.io/cluster-name: "global"
  annotations:
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
spec:
  controlPlaneLoadBalancer: {host: "${CONTROL_PLANE_VIP}", port: 6443, type: external}
  controlPlaneEndpoint: {host: "${CONTROL_PLANE_VIP}", port: 6443}
  credentialSecretRef: {name: "${PROVIDER_SECRET_NAME}"}   # created in the prerequisites above
  networkType: kube-ovn
  site: <dcs-site-id>
---
# 5. Top-level CAPI Cluster: global wiring, labels, and annotations.
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: global
  namespace: cpaas-system
  labels:
    cpaas.io/cluster-name: "global"
    cluster-type: DCS
    is-global: "true"
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: DCSCluster
    capi.cpaas.io/kubernetes: ${K8S_VERSION}   # same value as KubeadmControlPlane.spec.version
    cpaas.io/registry-address: "${NODE_REGISTRY_ADDRESS}"
```

```
    cpaas.io/nodes-mode: self-managed   # node lifecycle managed by CAPI
+ the DCS provider
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>   # a /16 you choos
e; must not overlap the pod / service CIDRs or another cluster's join CID
R
    cpaas.io/kube-ovn-version: <kube-ovn-chart-version>
spec:
  clusterNetwork:
    pods:     {cidrBlocks: ["${CLUSTER_CIDR}"]}
    services: {cidrBlocks: ["${SERVICE_CIDR}"]}
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: global-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: DCSCluster
    name: global
```

内部 **Self-built VIP** 变体

上述示例使用 `type: external`。应用前，可以将资源 4 的端点字段替换为以下内部配置。VIP 必须预留在控制平面 Layer 2 网络中，并且 `vrid` 在该网络中必须唯一：

```
spec:
  controlPlaneLoadBalancer:
    host: "${CONTROL_PLANE_VIP}"
    port: 6443
    type: internal
    vrid: <unique-vrid-1-255>
    # interface: eth0  # optional; omit for provider/alive auto-detection
  controlPlaneEndpoint: {host: "${CONTROL_PLANE_VIP}", port: 6443}
```

需要替换的值

占位符 / 变量	来源
`${K8S_VERSION}`、`${CONTROL_PLANE_VIP}`、`${PLATFORM_HOST}`、`${NODE_REGISTRY_ADDRESS}`、	在步骤 1 中导出。

占位符 / 变量	来源
`${CLUSTER_CIDR}`、`${SERVICE_CIDR}`、`${PROVIDER_SECRET_NAME}`	
`<dcs-api-user>` / `<dcs-api-password>` / `<dcs-api-host>` / `<dcs-site-id>`	DCS 平台凭证和站点信息。它们不会出现在清单中——而是写入用于在先决条件中创建 `${PROVIDER_SECRET_NAME}` 的凭证文件。请参阅云凭证。
`<vm-template-name>`、`<dns-image-tag>`、`<etcd-image-tag>`	`cpaas.io/dcs-vm-template` ConfigMap。请参阅解析占位值。
`<dcs-cluster-name>`、`<dvswitch-name>`、`<port-group-name>`、`<datastore-name>`	DCS 平台对象。请向 DCS 管理员确认。
`192.0.2.x`、`<kube-ovn-join-cidr>`、`<kube-ovn-chart-version>`	网络的节点 IP/网关/DNS；一个不会与 Pod / Service CIDR 或其他集群的 join CIDR 重叠的 `/16` Kube-OVN join CIDR；以及从 OS 支持矩阵获取的 kube-ovn chart 版本。
`ssh-ed25519 AAAA...`	有效的 OpenSSH 公钥。`ignition` 格式不接受空列表；即使你不打算通过 SSH 登录，也请提供任意有效密钥。

Secret 加密与灾难恢复

此示例启用了 etcd 静态 Secret 加密。`apiServer.extraArgs.encryption-provider-config` 保留在你从附录复制的 `clusterConfiguration` 中，DCS provider 会提供它所指向的文件：为第一台控制平面机器生成一个随机密钥，之后的每台机器则使用正在运行的 API server 自己的文件。不要将 `/etc/kubernetes/encryption-provider.conf` 添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files`——provider 会丢弃指向该路径的任何条目。请参阅如何交付加密 provider 配置。

只有在你明确希望停用静态加密时，才移除此参数。对于 DR 集群对，两侧的密钥必须相同，因此必须按照可选的灾难恢复部署中的说明，使用 `DCSCluster.spec.encryptionProviderConfigRef` 固定该密钥。

Global 集群灾难恢复

此页面是 `global` 集群的灾难恢复入口，该集群的节点运行 Alauda OS。部署时的灾难恢复配置属于 `global` 集群安装操作步骤的一部分。

目录

- 部署操作步骤
- 网络要求
- 可观测性部署
- 操作范围
- 故障转移
- 裸金属 DR 模型
  - 端点和身份
  - 同步规则
  - 启动和验证
  - 验收
  - 故障排除
- Provider Notes
- 另请参阅

部署操作步骤

创建主集群和备用 `global` 集群时，请在安装指南中使用可选灾难恢复部署。

该部署操作步骤是安装时 DR 配置的权威来源，包括：

- 每一侧都从各自的 bootstrap 节点进行安装：每个提供商总共使用两个 bootstrap 节点。
- 主集群和备用集群使用相同的 Kubernetes API server 加密提供商配置。
- 每个集群的 `KubeadmControlPlane` etcd server 证书 SAN 列表包含该集群自己的控制平面 VIP、Platform Access Address 和 `etcd.kube-system`；主集群和备用集群的列表不同（请参阅将 DR etcd Server 证书 SAN 添加到 KubeadmControlPlane）。此列表与平台 `console.cert` SAN 列表相互独立。
- Huawei DCS 部署从 `DCSCluster.spec.encryptionProviderConfigRef` 引用共享的加密提供商 Secret。该 Secret 必须存在于每个最终的 `global` 集群中，而不仅仅是安装它的 bootstrap 集群。
- VMware vSphere 部署通过 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 写入相同的 `/etc/kubernetes/encryption-provider.conf` 文件。
- Huawei Cloud Stack 部署通过 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 写入相同的 `/etc/kubernetes/encryption-provider.conf` 文件。
- Bare Metal 部署从 `BaremetalCluster.spec.encryptionProviderConfigRef` 引用共享的加密提供商 Secret，并在两侧使用相同的 Kubernetes ServiceAccount 签名密钥。
- Huawei DCS、VMware vSphere 和 Huawei Cloud Stack 会在 installer 导入前创建 `dcs-import-extra-resources` ConfigMap，以便安装过程能够导入被引用的凭证 Secret，以及适用时提供商特定的基础设施资源。该名称保留 `dcs` 前缀，以保持历史 installer 兼容性。
- Bare Metal 会在 installer 导入前创建 `dcs-import-extra-resources` ConfigMap，以便安装过程能够保留交接所需的持久化裸金属和 elemental owner 资源。它不会导入 Global plan Secret 或 kubeadm bootstrap data Secret；在所有目标 Global `MachineInventory` 对象存在后，新的交接会创建初始 Global plan Secret。
- 两个集群安装成功后，备用集群会使用指向当前活动主集群的连接值安装 `global-etcd-sync`。

两个集群均安装完成后，请在两个 `global` 集群上安装基础设施提供商和 Cluster Bootstrap Provider，以便备用集群能够在故障转移后协调业务集群。请参阅 Huawei DCS、VMware vSphere、Huawei Cloud Stack 或 Bare Metal。

网络要求

灾难恢复的前提条件是使用 DNS 名称的稳定 Platform Access Address。在正常运行期间，该域名仅解析到主 `global` 集群 VIP。备用 `global` 集群使用该域名访问主 `global` 集群上的平台服务。

安装任一集群之前，请完成以下两项网络准备工作：

- 配置每个集群的负载均衡器，将其 VIP 上所需的 TCP 端口转发到该 VIP 后方的控制平面节点。仅当用户通过 HTTP 访问平台时，才配置端口 `80`。
- 在主集群和备用集群网络之间双向开放所需的 TCP 端口。将规则应用于防火墙、安全组、路由器 ACL 以及其他任何站点间网络控制措施。

TCP 端口	服务	所需原因
`443`	ACP 平台 HTTPS	备用集群使用 Platform Access Address 访问活动平台。灵雀云容器平台 **Global Cluster Disaster Recovery** 使用此地址，日志和监控组件也会向此地址发送告警回调。
`80`	ACP 平台 HTTP	仅当用户通过 HTTP 访问平台时才需要。其他情况下，连通性要求与端口 `443` 相同。
`6443`	Kubernetes API server	安装或重新安装 灵雀云容器平台 **Global Cluster Disaster Recovery** 时，备用集群会连接活动集群的 API server，以获取同步所需的 etcd CA 材料。
`11443`	内置镜像仓库	备用集群从通过 Platform Access Address 配置的镜像仓库拉取平台和插件镜像。此要求仅适用于平台内置的 Registry。
`2379`	etcd	备用集群上的 灵雀云容器平台 **Global Cluster Disaster Recovery** 从主集群读取 etcd 数据，并将其写入本地备用 etcd。

网络策略必须是双向的，因为故障转移后集群角色会互换。故障转移前，有效服务流量通常从备用集群流向主集群。故障转移后，原主集群成为新的备用集群，必须通过相同端口访问新的主集群。这项要求并不意味着 etcd 复制是双向的：灵雀云容器平台 **Global Cluster Disaster Recovery** 仅在当前备用集群上运行，并将同步的数据写入其本地 etcd。

可观测性部署

灾难恢复仅复制 `global` 集群的 etcd 数据，`cpaas-system` 等命名空间不在复制范围内。因此，监控和日志数据永远不会复制到备用集群。在安装任一集群之前，请规划可观测性部署。

- 每个 `global` 集群收集、存储和查询自身的指标与日志。主集群和备用集群不会使用彼此的监控或日志存储，因此备用集群在故障转移后可以立即监控自身。
- 业务集群的监控和日志不得写入任一 `global` 集群。请在专用业务集群中汇总业务集群的指标和日志，并在该集群上运行 VictoriaMetrics Center 和日志存储，然后将每个业务集群的采集智能体指向该集群。
- 如果某个网络区域无法访问该汇总集群，请为该区域提供单独的汇总集群，而不要回退到 `global` 集群。
- 存储在 `global` 集群中的业务集群可观测性数据会在该集群的角色发生变化时变得不可用，因为这些数据不会被复制，另一个 `global` 集群也无法提供这些数据。

有关与非 Alauda OS 路径共用的同步范围和排除范围，请参阅 Global Cluster Disaster Recovery。

操作范围

安装主集群和备用集群后，请将灾难恢复作为独立的生命周期流程进行操作。使安装清单与安装指南保持一致，然后使用经过批准的操作运行手册执行以下任务：

- 验证 灵雀云容器平台 **Global Cluster Disaster Recovery** 的运行状况和复制堆积量。
- 验证备用集群能否解密在主集群上创建的 Kubernetes Secrets。
- 在计划内故障转移之前，验证主集群和备用集群的控制平面 VIP 以及平台访问路径。
- 在计划内故障转移或故障恢复之前，验证两个集群网络之间所需的端口是否已双向放行。切换后，原备用集群会成为新的主集群，原主集群会成为新的备用集群，因此服务流量的方向会反转。
- 使用经过批准的操作步骤执行故障转移和故障恢复。
- 故障转移后协调特定于 provider 的资源。

> 卸载 **Alauda Container Platform Global Cluster Disaster Recovery** 前验证一致性

灾难恢复切换和支持灾难恢复的 `global` 集群升级都会卸载 灵雀云容器平台 **Global Cluster Disaster Recovery**。在卸载之前，请确认备用 `global` 集群数据与主集群一致。在 Alauda OS 集群上，业务集群节点由 Cluster API `Machine` 对象提供支持，因此，不一致同步后错误的所有者引用解析可能会删除这些 `Machine` 对象，并销毁其后端虚拟机。如果一致性检查报告缺失或多余的键，请勿卸载插件；先解决不一致问题，或联系技术支持。有关与非 Alauda OS 路径共用的详细切换和升级操作步骤，请参阅 Global Cluster Disaster Recovery 和 Upgrade in a DR Environment。

故障转移

以下序列与 provider 无关：它由平台域和 `etcd-sync` 生命周期驱动，而不是由任何 provider API 驱动。特定于 provider 的 同步源和故障转移后检查请参阅 Provider Notes；对于 Bare Metal，请参阅 Bare Metal DR Model 部分。

1. 隔离主集群，使其无法写入业务资源。
2. 等待监控器显示无差异，然后从旧备用集群卸载 `etcd-sync`。移除其 `ModuleInfo` 才会卸载插件；仅将 Deployment 缩容到零还不够（请参阅下面的警告）。对于 Bare Metal，随后在旧备用集群的 Bare Metal provider 的 `ModuleInfo` 中设置 `elemental.systemAgent.sharedAuthReadOnly: false`；该集群会成为活动集群，然后等待 `elemental-operator` Deployment 不再包含 `--system-agent-shared-auth-read-only` 参数（使用 `kubectl -n cpaas-system get deploy elemental-operator -o yaml | grep -- --system-agent-shared-auth-read-only` 检查）。在此之前，新的工作节点无法获取其计划。
3. 将平台域切换到备用集群的控制平面入口。
4. 验证备用集群 API 和平台端点。
5. 确认业务主机通过平台域重新连接，并能够更新计划反馈。
6. 对于 Bare Metal，先在新备用集群的 provider `ModuleInfo` 中设置 `elemental.systemAgent.sharedAuthReadOnly: true`，并使用相同的命令确认 `elemental-operator` Deployment 包含 `--system-agent-shared-auth-read-only` 参数。然后，在新的备用集群（被隔离的原主集群）上安装 `etcd-sync` 插件，并使用反转后的活动/备用值；对于 Bare Metal，还要使用相同的 `baremetal-dr-rules` ConfigMap。请注意，活动集群令牌现在必须来自新的活动集群，并且推送插件包时使用的是平台域当前指向的集群的 Registry 凭证。

> **卸载镜像，不要仅将其缩容**
>
> 启动时，`etcd-sync` 会解析平台域，并检查该域所指向的集群是否未安装 `etcd-sync`。这可以防止镜像同时在两侧运行。如果步骤 2 仅将 Deployment 缩容到零，`ModuleInfo` 仍会存在，新的备用集群会拒绝启动，并显示：
>
> ```
> check failed, error: moduleinfo <name> is found, it shouldn't happe
> n,
>   please remove it or check the DNS resolution of platformURL: <pla
> tform-domain>
> ```
>
> 尽管提示文本如此表述，但这是安全检查按预期工作，而不是缺陷。在新的活动集群上删除 `ModuleInfo` 后，新的备用集群便会正常启动。

保持被隔离的集群处于隔离状态，直到单独规划故障恢复。切勿让两个 provider 同时针对同一业务清单运行。

切换后，在新的活动集群上验证每个业务 `Cluster` 和 `Machine` 对象均完好无损——在 Alauda OS 集群上，这些对象代表真实机器，因此在切换过程中丢失资源会导致节点被删除。

裸金属 DR 模型

本节介绍 ACP 4.4.0 及更高版本的裸金属 DR 操作步骤。涵盖全新主集群/备用集群安装；使用可选灾难恢复部署创建两个集群。

Bare Metal 将物理主机及其计划 Secret 保存在 MachineInventory 对象中。因此，DR 会将 Global 主机与业务主机分开。

端点和身份

主机	端点	身份	同步
本地 `Cluster/global` 主机	`https://<local-control-plane-vip>:6443`	`baremetal-global-system-agent`	仅本地

主机	端点	身份	同步
非 `global` 业务主机	`https://<platform-domain>/kubernetes/global`	`baremetal-system-agent`	主集群到备用集群

Global 行专用于 DR 以及下文所述的交接后 Global 注册。业务行是标准的 Bare Metal 注册路径；有关业务集群路径分阶段的网络要求（包括 DNS、NTP、Registry 和集群内部流量），请参阅网络连接。

在两侧的 Bare Metal provider 上设置 `elemental.systemAgent.splitAuthEnabled: true`。活动 Global 使用 `sharedAuthReadOnly: false`；备用集群接收数据时使用 `sharedAuthReadOnly: true`。按照添加 provider 特定的 DR 字段中的说明，在每一侧的 bootstrap release 上设置这些值，并在每个最终的 `global` 集群的 provider `ModuleInfo` 中再次设置，同时设置 `elemental.tls.caCertSecretKey`；请参阅安装集群插件。保留标准的 ServiceAccount、Secret、Role 和 RoleBinding 名称。切勿授予命名空间范围的 Secret 访问权限。

交接后，每个 Global 注册都必须使用本地 API server：

```
metadata:
  annotations:
    baremetal.cluster.io/system-agent-auth-scope: global
    baremetal.cluster.io/system-agent-server-url: https://<local-control-plane-vip>:6443
    baremetal.cluster.io/system-agent-endpoint-mode: direct-apiserver
```

业务注册使用平台域。交接后不要重复使用 bootstrap Global ISO；请在活动 Global 上创建新的 Global 注册和 SeedImage。仅当业务 ISO 的注册 URL 使用平台域时，该 ISO 才可继续重复使用。

同步规则

`etcd-sync` 在备用集群上运行，并读取主集群。支持 DR 的 v4.4 chart 已提供标准的 Bare Metal 前缀和 Global 资源排除项，并接受来自带标签 ConfigMap 的其他规则。自定义规则源必须存在于两个 Global 集群上，以便在角色切换后可用。

使用权重 `100` 创建 `baremetal-dr-rules`：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: baremetal-dr-rules
  namespace: cpaas-system
  labels:
    etcd-sync.cpaas.io/enabled: "true"
  annotations:
    etcd-sync.cpaas.io/weight: "100"
data:
  ignore-equal.txt: |
    # Add one line for every Global plan Secret on both sides.
    /registry/secrets/cpaas-system/<primary-global-plan-secret>
    /registry/secrets/cpaas-system/<standby-global-plan-secret>
```

内置 chart 规则已经涵盖 Bare Metal 资源前缀、Global 清单/注册筛选、`SeedImage`、Global 本地 system-agent bundle、监控器以及本地 Registry 凭证。不要在此 ConfigMap 中重复添加这些条目。开始同步前，已安装的 chart 必须包含这些内置规则。

将计划占位符替换为实际值：

- 从 `MachineInventory.status.plan.secretRef.name` 读取计划名称。
- 为两侧的每个 Global 计划 Secret 添加一行。
- 每当添加或替换 Global 计划 Secret 时，更新两侧的 ConfigMap。

首次同步前，要求规则源注解为 `etcd-sync.cpaas.io/status: accepted`，并确认 `etcd-sync-rule-snapshot` 包含该源：

```
kubectl --kubeconfig <standby-kubeconfig> -n cpaas-system \
  get configmap baremetal-dr-rules \
  -o jsonpath='{.metadata.annotations.etcd-sync\.cpaas\.io/status}{"\n"}'
kubectl --kubeconfig <standby-kubeconfig> -n cpaas-system \
  get configmap etcd-sync-rule-snapshot \
  -o jsonpath='{.data.snapshot\.json}'
```

被拒绝的源不会生效。在允许镜像写入之前修复该问题。

插件 chart 负责标准的 CAPI、Bare Metal、Elemental、Global 资源、system-agent、SeedImage、Registry 和监控规则。此 ConfigMap 仅添加渲染 chart 时无法获知的运行时

Global 计划 Secret 密钥。

`dcs-import-extra-resources` 是安装时的交接输入。它必须包含持久化的 provider 资源，以及 DCS 和 Bare Metal 所引用的加密 Secret，但它不是 etcd 同步规则源。

启动和验证

1. 仅在当前备用集群上安装 `global-etcd-sync`。
2. 将 `active_cluster_vip` 设置为主集群 VIP。
3. 将 `active_cluster_token_secret_ref` 设置为备用集群本地的 Secret，其中包含主集群令牌。
4. 等待 AppRelease 和两个 Deployment 均变为 Ready。
5. 运行监控器检查，并要求缺失密钥数和多余密钥数均为零。
6. 将规则快照和监控器结果作为验收门禁的一部分；仅凭 AppRelease 状态不足以完成验收。

验收

故障切换后，验证：

- 业务 `Cluster` 和 `Machine` 对象保持健康。
- 每个 `BaremetalMachine` 仍引用预期的 `MachineInventory`。
- 业务计划达到 `Applied`，且其反馈已更新。
- 工作节点扩容会创建计划并更新共享 Role。
- 工作节点缩容完成其清理计划并释放清单。
- URL 使用平台域的业务 ISO 可以注册新主机。
- Global 主机仍使用本地 VIP、本地令牌和本地计划 Secret。

故障排除

症状	首要检查
规则源为 `rejected`	规则数据密钥、语法以及 `message` 注解

症状	首要检查
Global 计划被覆盖	`ignore-equal.txt` 中缺少其确切的 Secret 密钥；停止同步并更新两个规则 ConfigMap
Global 清单被删除	已安装的 chart 缺少其内置 Global 排除项；停止同步，并在恢复本地对象前验证渲染后的 chart
业务主机获得 `401` 或 `403`	签名密钥、issuer/audience、共享令牌 Secret 以及 Role `resourceNames`
监控器报告存在差异	活动/备用方向、规则快照和隔离状态

Provider Notes

Huawei DCS

按照可选灾难恢复部署中的 DCS 步骤操作。两侧的 DCS 安装必须使用相同的加密提供程序 Secret 和 `DCSCluster.spec.encryptionProviderConfigRef`。对于 DCS，不要将加密提供程序文件添加到 `KubeadmControlPlane.spec.kubeadmConfigSpec.files`。在两侧创建 DCS `dcs-import-extra-resources` ConfigMap，以便安装程序导入 `DCSCluster.spec.credentialSecretRef.name` 引用的 Secret 和 `DCSCluster.spec.encryptionProviderConfigRef.name` 引用的 Secret。DCS 提供程序资源由内置流程迁移，但该流程不会复制任一 Secret。

每个最终的 `global` 集群中都必须存在加密提供程序 Secret。如果它仅保留在引导集群中，安装仍会完成，且这对集群看起来健康，但引导集群消失后，后续每次控制平面计算机创建都会失败——新的 `Machine` 会停留在 `Provisioning` 中，并且提供程序会记录 `get encryption provider config secret ... not found`。在停用任一引导集群之前，验证两侧均存在该 Secret。

对于计划内切换，请运行故障切换序列。`baremetal-dr-rules` ConfigMap 和注册检查仅适用于 Bare Metal；其他步骤均按原样适用于 DCS。切换后，在新的活动集群上验证业务 `Cluster`、`Machine` 和 `DCSMachine` 对象完整，且没有虚拟机被删除：故障切换期间，安装指南命名约定中的 `global-` 名称前缀用于标识属于 `global` 对的资源，因此

两侧的每个 DCS 资源都必须带有该前缀。同时确认 DCS 凭证 Secret 和加密提供程序 Secret 存在于当前活动的一侧。

VMware vSphere

按照可选灾难恢复部署中的 VMware vSphere 步骤操作。VMware vSphere 安装使用与主安装相同的 `KubeadmControlPlane` 加密文件条目，以及其自身的 etcd SAN 列表。不需要 `VSphereCluster` 加密 Secret 引用。创建 VMware vSphere `dcs-import-extra-resources` ConfigMap，以便安装程序导入 vSphere 基础设施资源和 `VSphereCluster.spec.identityRef.name` 引用的 `global-vsphere-credentials` Secret。

Huawei Cloud Stack

按照可选灾难恢复部署中的 HCS 步骤操作。HCS 安装使用与主安装相同的 `KubeadmControlPlane` 加密文件条目，以及其自身的 etcd SAN 列表。不需要 `HCSCluster` 加密 Secret 引用。创建 HCS `dcs-import-extra-resources` ConfigMap，以便安装程序导入 HCS 基础设施资源和 `HCSCluster.spec.identityRef.name` 引用的 Secret。

Bare Metal

按照可选灾难恢复部署中的 Bare Metal 步骤操作。使用两个引导 KIND 主机，一个用于主集群，另一个用于备用集群。安装备用集群之前，将平台域切换到备用入口，以便平台渲染的 URL 解析到正在安装的一侧。

Bare Metal 要求以下所有内容保持一致：

- 两侧使用相同的 Kubernetes 加密提供程序 Secret 内容，并由 `BaremetalCluster.spec.encryptionProviderConfigRef` 引用。
- 通过 `dcs-import-extra-resources` 将加密提供程序 Secret 导入每个最终的 `global` 集群；该 Secret 不能只存在于引导 KIND 集群中。
- 两侧使用相同的 ServiceAccount 签名密钥文件，以及相同的 issuer/audience 值。
- 两侧使用相同的受信任平台 HTTPS 证书，且其 SAN 列表中包含稳定的平台域。仅当平台 HTTPS 有意使用控制平面 VIP 时，才添加控制平面 VIP SAN；不要将内部

Service 名称添加到 `console.cert`。使用平台内置 Registry 时，端口 `11443` 上的 registry 使用独立管理的 `global-registry-server` 证书链。

- 在两侧调用安装程序 API 之前创建 `dcs-import-extra-resources`，其中包含持久的 Global Bare Metal 和 Elemental owner 资源，以及所引用的加密提供程序 Secret。它必须排除 Global 计划 Secret、`KubeadmConfig` 对象和 kubeadm 引导数据 Secret；全新移交会使用活动 Global `MachineInventory` UID 创建初始 Global 计划 Secret。
- 在每个引导集群上，使用本地 `handoffHook.controlPlaneVIP`、`handoffHook.directAPIServer: true`、`elemental.systemAgent.authMode: shared`、`elemental.systemAgent.splitAuthEnabled: true` 和 `elemental.systemAgent.globalServiceAccountName: baremetal-global-system-agent` 设置 Bare Metal 提供程序 AppRelease 值。活动侧设置 `elemental.systemAgent.sharedAuthReadOnly: false`；非活动侧将其设置为 `true`。在每个最终的 `global` 集群中，仅在提供程序的 `ModuleInfo` 的 `spec.valuesOverride` 中设置 `elemental.tls.caCertSecretKey`、`elemental.systemAgent.splitAuthEnabled: true` 和该侧的 `sharedAuthReadOnly` 值；参见安装集群插件。
- 已部署的 `elemental-operator` 镜像以及新生成的每个 `baremetal-base-image` 和 `baremetal-base-image-iso` 都基于相同的固定 elemental 提交构建。该提交发生变化时，重新构建安装介质。

不要将 `SeedImage` 从主集群同步到备用集群。当业务 ISO 需要在故障切换后继续有效时，同步非 `global` 的 `MachineRegistration` 资源。不要将主集群的 Global `MachineRegistration` 资源同步到备用集群。使用 `baremetal.cluster.io/system-agent-auth-scope: global`、`baremetal.cluster.io/system-agent-server-url: https://<local-global-control-plane-vip>:6443` 和 `baremetal.cluster.io/system-agent-endpoint-mode: direct-apiserver` 在本地创建 Global 注册。

故障切换后，在执行单独的故障恢复计划之前，保持主集群处于隔离状态。不要让主集群和备用集群的 Bare Metal 提供程序同时针对同一组业务清单运行。

另请参阅

有关在非 Alauda OS 上对 `global` 集群进行灾难恢复，请参阅Global Cluster Disaster Recovery。

有关在 Alauda OS 上安装和升级 `global` 集群，请参阅：

- 安装 Global 集群
- 升级发行版版本，然后参阅Alauda OS 上的 Kubernetes 升级
- 在 DR 环境中升级

在非 Alauda OS 上安装

前置要求 节点预处理 安装

安装程序参数 Global 集群灾难恢复 安装故障排查

前置要求

在安装 `global` 集群之前，您需要准备符合要求的硬件、网络和操作系统。

INFO

1. 当前平台不支持直接在现有 Kubernetes 环境中安装 `global` 集群。如果您的环境中已经存在 Kubernetes 集群，请在安装前备份数据并清理环境。
2. 如果您计划使用 global 集群灾难恢复，请先阅读 global 集群灾难恢复。
3. 确保所有新节点均满足节点要求。
4. 磁盘的性能和容量也必须满足磁盘配置要求。

目录

- 资源规划
  - 部署架构
  - 单节点
  - 单集群
  - 多集群
- 网络
  - 网络资源
  - 网络配置
  - LoadBalancer 转发规则
- 外部平台镜像仓库

资源规划

本节介绍安装 ACP 前进行资源规划的指导原则。 请根据您的环境和使用需求选择合适的部署场景，并相应地准备资源。

> **INFO**
>
> 以下建议仅涵盖成功安装 **Global** 集群 所需的最小资源。
>
> 其中不包括在 Global 集群上部署的任何额外扩展或组件所需的资源。
>
> 有关各个扩展的详细要求，请参阅相应的组件文档。

部署架构

> **WARNING**
>
> 对于 ARM 架构（例如 Kunpeng 920），建议将配置提升至 x86 最低配置的 **2** 倍，但不得低于 **1.5** 倍。
>
> 例如：如果 x86 需要 8 核 16GB，则 ARM 至少应达到 12 核 24GB，建议配置为 16 核 32GB。

安装前，您必须确定最适合您使用场景的部署架构。 ACP 支持以下三种常见的部署架构：

- 多集群
  如果您需要集中管理多个 Kubernetes 集群，请选择此架构。 在此模式下，ACP 由一个 **Global** 集群和多个业务集群组成。 在 Global 集群上运行非平台业务负载可能会降低平台的稳定性和性能，应予以避免。
- 单集群
  如果您计划仅安装一个集群并直接在其上运行业务负载，请选择此架构。 在此模式下，Global 集群同时充当业务集群，因此与多集群模式下纯 Global 集群的部署相比，需要更多资源。
- 单节点

> **WARNING**
>
> 此架构仅用于测试或概念验证，不应在生产环境中使用。

单节点

下表列出了以单节点模式安装 ACP 的最低硬件要求。

资源	最低要求
CPU	12 核
内存	24GB
存储	存储容量

单集群

在此模式下，Global 集群同时充当控制平面和业务集群。 Global 集群的控制平面节点数必须为 3。

总资源要求由两部分组成：

- Global 集群自身所需的基础资源
- 在同一集群上运行业务负载所需的额外资源

高可用 **Global** 集群所需的资源如下：

资源	最低要求
CPU	8 核
内存	16GB
存储	存储容量

要估算业务负载所需的额外资源，请参阅 评估业务集群资源

多集群

管理多个业务集群时，Global 集群的资源使用量会随着所管理集群数量的增加而按比例增加。额外开销主要来自集群注册、监控和控制平面同步。

要根据所管理集群的数量估算 Global 集群所需的资源，请参阅评估 **Global** 集群资源

网络

在安装之前，请确保已准备好所需的网络资源。

如果您的环境中有可用的硬件 LoadBalancer，建议使用它。如果没有，您可以启用 `Self-built VIP`，通过 keepalived 提供软件负载均衡。

注意：`Self-built VIP` 不支持域名配置。

网络资源

资源	必需	数量	描述
`global` VIP	必需	1	用于集群中的节点访问 kube-apiserver，需要在负载均衡设备中进行配置以确保高可用。 此 IP 也可用作平台 Web UI 的访问地址。 与 `global` 集群处于同一网络中的业务集群，也可以通过此 IP 访问 `global` 集群。
外部 IP	可选	按需	当存在与 `global` 集群不在同一网络中的业务集群时（例如混合云场景），必须提供此 IP。其他网络中的业务集群通过此 IP 访问 `global` 集群。 需要在负载均衡设备中配置此 IP，以确保高可用。 此 IP 也可用作平台 Web UI 的访问地址。

资源	必需	数量	描述
域名	建议	按需	建议将其用于集群端点和平台访问地址。请提前提供域名，并确保域名解析正确。 建议使用域名，因为如果在集群安装后需要更改 VIP，您可以轻松更新 DNS 记录，而不会影响集群运行。 在以下情况下，必须使用域名： • `global` 集群需要支持 IPv6 访问； • 计划为 `global` 集群制定灾难恢复方案。
证书	可选	按需	建议使用受信任的证书，以避免浏览器安全警告；如果未提供证书，安装程序将生成自签名证书，但使用 HTTPS 时可能存在安全风险。

NOTE

如果平台需要配置多个访问地址（例如内部网络和外部网络的地址），请根据上表提前准备相应的 IP 地址或域名。您可以稍后在安装参数中配置这些地址，也可以在安装后根据产品文档添加这些地址。

网络配置

类型	要求说明
网络带宽	集群内带宽必须 ≥1 Gbps（建议 10 Gbps）。 跨集群带宽必须 ≥100 Mbps（建议 1 Gbps）。 带宽不足可能会显著降低数据查询性能。
网络延迟	集群内延迟必须 ≤10 ms。 跨集群延迟必须 ≤100 ms（建议 ≤30 ms）。
网络策略	请参阅 LoadBalancer 转发规则，确保必要的端口已开放。
IP 地址范围	`global` 集群节点应避免使用 172.17-18 网段。如果已使用，请调整 nerdctl 配置（添加 bip 参数）以避免冲突。

LoadBalancer 转发规则

此规则用于确保 `global` 集群能够正常接收来自 LoadBalancer 的流量。请根据下表检查网络策略，确保相关端口已开放。

源 IP	协议	目标 IP	目标端口	描述
`global` 集群节点 IP、VIP、外部 IP	TCP	所有控制平面节点 IP	443	通过 HTTPS 协议为平台 Web UI、镜像仓库和 Kubernetes API Server 提供访问服务。默认端口为 `443`。如果需要使用自定义 HTTPS 端口，请执行以下操作： • 将端口转发规则中的目标端口替换为自定义端口号。

源 IP	协议	目标 IP	目标端口	描述
				• 稍后在 Web UI 安装参数中填写自定义端口号。
`global` VIP、外部 IP	TCP	所有控制平面节点 IP	6443	此端口为集群内的节点提供 Kubernetes API Server 访问服务。
`global` VIP、外部 IP	TCP	所有控制平面节点 IP	11443	此端口为集群内的节点提供镜像仓库访问服务。注意：如果计划使用外部平台镜像仓库，而不是使用 `global` 集群提供的 Registry，则无需配置此端口。有关外部 Registry 的网络要求，请参阅准备外部平台镜像仓库。

TIP

- 建议在 LoadBalancer 上配置健康检查，以监控端口状态。
- Global 集群灾难恢复要求在两个集群 VIP 上配置负载均衡器监听器，并确保 Primary 集群网络与 Standby 集群网络之间能够双向访问。请参阅网络要求。

- 平台默认仅支持 HTTPS。如果需要支持 HTTP，则需要为所有控制平面节点开放 HTTP 端口。

外部平台镜像仓库

对于新的生产部署，如果组织要求独立管理镜像仓库的可用性、容量、备份和生命周期，请准备一个由客户管理的外部平台镜像仓库。

运行 `setup.sh` 之前，请完成准备外部平台镜像仓库。该页面定义了所需的地址、网络访问、存储以及推送和拉取权限；区分了这些要求与生产建议；并说明了如何上传完整的目标版本负载。

节点预处理

在安装 `global` 集群之前，所有节点（控制平面节点和工作节点）都必须完成预处理。

> **INFO**
>
> 此页面适用于运行非 **Alauda OS** 的节点，例如 Kylin Linux Advanced Server、RHEL 或 Ubuntu，并且由 ACP 通过 SSH 进行置备。下面的多项检查（例如 SSH 用户和 `/etc/ssh/sshd_config` 设置）用于确保基于 SSH 的节点加入操作正常运行。如果您的集群使用 Alauda OS，则本页面中的节点预处理步骤不适用。请改为遵循 在 Alauda OS 上安装 global 集群。

目录

- 支持的 OS 和内核版本
  - x86
  - ARM
  - 操作系统支持范围变更
- 执行快速配置脚本
- 节点检查
- 附录
  - 为什么必须禁用 SELinux 和 AppArmor
  - 加固节点安全配置
    - 保持主机防火墙启用，而不是重新启用
    - 操作顺序
    - 如果节点已经运行过完整的 `init.sh`

主机防火墙

删除冲突的软件包

配置搜索域

支持的 OS 和内核版本

下表列出了支持的操作系统、经过验证的版本以及相应的测试内核版本。

平台会强制执行支持列表中声明的版本匹配粒度：

- **OS** 版本：使用下面列出的发行版版本。除非同时列出，否则不正式支持其他主版本或次版本。
- 内核版本：使用操作系统供应商提供的官方内核。当列表给出确切的 `x.y.z-build` 值时，构建后缀可以不同，但 `x.y.z` 必须匹配。当列表列出操作系统版本但未指定内核值时，请使用该版本随附的官方内核。不要假定任意更新的内核都受支持。

> **INFO**
>
> `x86-64-v2` 是 CPU 指令集基线，但 ACP 不会将其作为运行非 Alauda OS 的 x86 节点的统一要求。CPU 仍必须满足所选操作系统和待部署组件的要求。如果无法确认兼容性，请联系技术支持。

x86

Red Hat Enterprise Linux (RHEL)

- RHEL 9.6: `5.14.0-570.12.1`

Ubuntu

- Ubuntu 22.04 LTS: `5.15.0-56-generic`

注意：不支持 Ubuntu HWE（Hardware Enablement）版本。

Kylin Linux Advanced Server

- Kylin Linux Advanced Server V11 2503：该版本随附的官方 6.6 系列内核

ARM

Kylin Linux Advanced Server

- Kylin Linux Advanced Server V11 2503：该版本随附的官方 6.6 系列内核

注意：ARM 架构仅支持 `Kunpeng 920`。对于其他 CPU 型号，请联系技术支持。

操作系统支持范围变更

ACP 4.4 引入了 Kubernetes 1.35，而 Kubernetes 1.35 移除了 cgroup v1。运行 Kubernetes 1.35 或更高版本的集群中的节点必须使用 cgroup v2 和 Linux 内核 5.8 或更高版本。

由于以下操作系统的内核早于 5.8，这些操作系统在 ACP 4.3 及更早版本中受支持，但从 ACP 4.4 起不再属于支持范围：

- Kylin Linux Advanced Server V10 和 V10 SP3（内核 4.19）
- Ubuntu 20.04 LTS（内核 5.4）
- RHEL 7.x 和 RHEL 8.x（内核 3.10 和内核 4.18）
- CentOS 7.x（内核 3.10）

如果您的节点运行上述某个操作系统，请选择以下路径之一。迁移到同一操作系统供应商的较新版本通常是成本最低的选项，因为这样可以保留现有的供应商关系、工具和运维实践：

- 将 RHEL 7.x 或 RHEL 8.x 节点迁移到 RHEL 9.6。
- 将 Ubuntu 20.04 LTS 节点迁移到 Ubuntu 22.04 LTS。
- 将 Kylin Linux Advanced Server V10 或 V10 SP3 节点迁移到 Kylin Linux Advanced Server V11 2503。
- 将节点迁移到 Alauda OS。对于 `global` 集群，请参阅 Alauda OS 上的安装；对于工作负载集群，请参阅 Alauda OS 上的创建。Alauda OS 是不可变操作系统，因此此路径会改变节点的管理方式，通常也是成本最高的选项。

- 继续使用 ACP 4.3.x，直到能够完成操作系统变更。

CentOS 7.x 在支持列表中没有后续版本。请将这些节点迁移到上述受支持的操作系统之一。

> **WARNING**
>
> 在集群仍运行当前 ACP 版本时变更操作系统，并且仅在所有节点都运行受支持的操作系统后，再升级到 ACP 4.4 或更高版本。此变更通常需要重建节点或更换机器。升级 ACP 不会将节点迁移到其他操作系统。

执行快速配置脚本

ACP 安装包提供了用于快速配置节点的脚本。

解压安装包，在 `res` 目录中获取 `init.sh` 脚本文件。将脚本文件复制到节点，并确保您具有 `root` 权限。

执行脚本：

```
bash init.sh
```

> **WARNING**
>
> `init.sh` 无法保证正确处理以下所有检查。您仍需继续执行下面的操作步骤。

节点检查

以下列出了节点上必须完成的所有检查。所需的检查因节点角色而异。例如，某些检查仅适用于控制平面节点。

检查分为两类：

- ✅ 表示检查必须通过。

- ⚠️ 表示检查必须在特定场景下满足。请根据说明确定相应条件是否满足。如果满足，则必须解决相关问题。

以下是检查列表：

- 操作系统和内核
  - ✅ 机器的 grub 启动配置必须包含 `transparent_hugepage=never` 参数。
  - ✅ 检查内核模块 `ip_vs`、`ip_vs_rr`、`ip_vs_wrr` 和 `ip_vs_sh` 是否已启用。
  - ⚠️ 如果 `global` 集群计划使用 `Kube-OVN` CNI，则必须启用内核模块 `geneve` 和 `openvswitch`。
  - ✅ 必须禁用 SELinux 和 AppArmor。在运行非 Alauda OS 的节点上，不能保持启用状态；请参阅为何必须禁用 SELinux 和 AppArmor。
  - ✅ 必须禁用主机防火墙，除非您的安全策略要求保持启用。在这种情况下，必须按照强化节点安全配置中的说明配置节点，然后再安装平台。
    - `init.sh` 会禁用主机防火墙。若要保持其启用状态，请向 `init.sh` 传递匹配的 `--skip-` 参数，而不是在之后重新启用防火墙。
    - 保持主机防火墙启用属于非默认配置。平台不会管理非 Alauda OS，因此启用的主机防火墙可能导致节点或集群中出现意外故障。您需要负责在此配置下运维和排查节点问题。
    - 启用主机防火墙后，您需要在每个节点上配置和维护端口规则。平台所需的端口请参阅通信矩阵。
  - ✅ 禁用 `swap`。
- 用户和权限
  - ✅ 节点的 SSH 用户具有 `root` 权限，并且可以免密码使用 `sudo`。
  - ✅ `/etc/ssh/sshd_config` 中的 `UseDNS` 参数必须设置为 `no`。
  - ✅ 在添加节点之前，将 `/etc/ssh/sshd_config` 中的 `UsePAM` 参数设置为 `no`，然后重启 `sshd`。否则，PAM 会话策略（例如强制修改密码、`pam_access`、`faillock` 或 `pam_limits`）可能会阻止基于 SSH 的节点加入。节点达到 `Ready` 状态后，可以恢复 `UsePAM yes`。在使用密码认证且 SELinux 处于强制执行模式的系统上，`UsePAM no` 本身可能会导致 SSH 登录失败；在这种情况下，请使用基于密钥的认证。

- ✅ `systemctl show --property=DefaultTasksMax` 必须返回 `infinity`；限制过低可能导致繁忙的容器创建线程失败。如果不是 `infinity`，请在 `/etc/systemd/system.conf` 中设置 `DefaultTasksMax=infinity`，然后运行 `systemctl daemon-reexec`。

- 节点网络
  - ✅ `hostname` 必须符合以下规则：
    - 不超过 36 个字符。
    - 以字母或数字开头和结尾。
    - 只能包含小写字母、数字、`-` 和 `.`，但不能包含 `.-`、`..` 或 `-.`。
  - ✅ `/etc/hosts` 中的 `localhost` 必须解析为 `127.0.0.1`。
  - ✅ `/etc/resolv.conf` 文件必须存在，并包含 `nameserver` 配置，但不得包含以 172 开头的地址（禁用 systemd-resolved）。
  - ⚠️ `/etc/resolv.conf` 文件不应配置搜索域（如必须配置，请参阅配置搜索域）。
  - ✅ 机器的 IP 地址不能是环回地址、多播地址、链路本地地址、全 0 地址或广播地址。
  - ✅ 执行 `ip route` 必须返回默认路由或指向 `0.0.0.0` 的路由。
  - ✅ 节点不得占用以下端口。此检查用于确认安装开始前端口处于空闲状态，而不是列出主机防火墙必须允许的端口。端口列表请参阅通信矩阵。
    - 控制平面节点：`2379`、`2380`、`6443`、`10249` ~ `10256`
    - 安装程序所在的节点：`8080`、`12080`、`12443`、`16443`、`2379`、`2380`、`6443`、`10249` ~ `10256`
    - 工作节点：`10249` ~ `10256`
  - ✅ 如果集群使用 **Kube-OVN** 或 **Calico**，请确保以下端口未被占用：
    - **Kube-OVN**：`6641`、`6642`
    - **Calico**：`179`
  - ⚠️ 确保 nerdctl 所需网络段 `172.17.x.x` ~ `172.18.x.x` 中的 IP 地址未被占用。如果此网络段中的 IP 已被占用且无法更改，请联系技术支持。

- 软件和目录要求：

- ✅ 必须安装以下软件：`ip`、`ss`、`tar`、`swapoff`、`modprobe`、`sysctl`、`md5sum`，以及 `scp` 或 `sftp`。
- ⚠️ 如果计划使用本地存储 **TopoLVM** 或 **Rook**，则需要安装 `lvm2`。
- ✅ 不允许存在 `/etc/systemd/system/kubelet.service` 文件。
- ✅ `/tmp` 的挂载参数不得包含 `noexec`。
- ✅ 删除与 `global` 集群组件冲突的软件包（请参阅删除冲突的软件包）。
- ✅ 如果存在，必须删除以下文件：
  - `/var/lib/docker`
  - `/var/lib/nerdctl`
  - `/opt/nerdctl/`
  - `/var/lib/containerd`
  - `/var/log/pods`
  - `/var/lib/kubelet/pki`
- 跨节点检查
  - ✅ `global` 集群中节点之间的网络通信不得受到限制：
    - 在未启用主机防火墙的默认配置中，节点之间不得存在网络防火墙限制。
    - 在节点上启用主机防火墙时，通信矩阵中的每个端口都必须允许节点之间进行通信，并且集群内部的 Pod 和 Service 流量不得按端口进行过滤。
  - ✅ 集群中每个节点的 `hostname` 必须唯一。
  - ✅ 所有节点的时区必须统一，时间同步误差必须 ≤ 10 秒。

附录

为什么必须禁用 SELinux 和 AppArmor

在运行非 Alauda OS 的节点上，必须禁用 SELinux 和 AppArmor。这是此版本的平台要求，而不是加固建议。`init.sh` 会禁用这两项功能，并且节点加入集群时，节点加入操作步骤会再次禁用它们。

SELinux。 平台组件会将主机路径挂载到容器中。镜像仓库安装程序会将主机上的 `/proc/mounts` 文件绑定挂载到其容器中，而镜像仓库和对象存储组件会将数据存储在主机上的 `/cpaas`、`/etc/kubernetes` 等目录下，这些目录带有操作系统的默认文件标签。在非 Alauda OS 上，容器运行时配置为不使用 SELinux 标签，因此这些挂载会根据主机策略进行评估；当 SELinux 以 `enforcing` 模式运行时，这些挂载会被拒绝。随后，容器运行时会在容器进程启动前创建容器失败，镜像仓库组件也无法完成安装，导致平台安装或升级停止。

`permissive` 模式并不是替代方案：它只会改变记录的日志内容，并不会改变这一要求。安装完成后再切换回 `enforcing` 的节点不属于受支持的配置。

AppArmor。 节点预处理会禁用 AppArmor，并且平台未经验证，不支持在强制执行 AppArmor 配置文件的集群节点上运行。在启用 AppArmor 的节点上运行不属于此版本支持的配置。

加固的操作系统配置文件（例如 Ubuntu Security Guide 提供的 STIG 配置文件）会启用这些机制。由于相同原因，将此类配置文件应用到集群节点不属于受支持的配置。

如果安全策略要求保持启用这些机制：请使用 Alauda OS，这是平台提供的不可变操作系统。在 Alauda OS 上，SELinux 和主机防火墙默认处于启用状态，平台会为你维护策略和端口规则。请参阅Alauda OS 上的安装和在 Alauda OS 上创建。

计划在未来版本中支持以 `enforcing` 模式运行启用 SELinux 的 Red Hat Enterprise Linux 9.6。有关其状态，请联系技术支持。

加固节点安全配置

仅当强制性安全策略（例如 STIG 基线）要求在集群节点上保持启用主机防火墙时，才使用本节。保持主机防火墙禁用是默认且推荐的节点配置，也是直接运行不带参数的 `init.sh` 所产生的配置。启用主机防火墙属于非默认配置，由你负责，因此请先在非生产环境中验证，然后再应用到生产节点。

SELinux 和 AppArmor 的情况不同：它们必须保持禁用。请参阅为什么必须禁用 SELinux 和 AppArmor。

保持主机防火墙启用，而不是重新启用

`init.sh` 可以跳过单个操作步骤。传入匹配的 `--skip-` 参数，以确保脚本不会禁用策略要求保持启用的主机防火墙：

实现方式	参数
`firewalld`	`--skip-disable_firewalld`
`ufw`	`--skip-disable_ufw`

例如，要在 RHEL 节点上保持 `firewalld` 启用：

```
bash init.sh --skip-disable_firewalld
```

这是推荐的方式。节点会保留安装时的状态，同时 `init.sh` 仍会执行其他所有节点预处理操作步骤。

操作顺序

请在平台安装之前配置节点，而不是在安装之后配置。这样，防火墙问题会在安装程序运行期间暴露，而不是在集群承载工作负载后才出现。

1. 使用你所用主机防火墙实现的 `--skip-` 参数运行 `init.sh`。
2. 配置主机防火墙规则。
3. 使用防火墙实现的状态命令验证节点上的最终状态，例如 `firewall-cmd --list-all` 或 `ufw status verbose`。
4. 安装平台。

如果节点已经运行过完整的 `init.sh`

仅当主机防火墙已经被禁用且必须重新启用时，才使用此方式。这比一开始就跳过该步骤需要更多操作。

1. 启用并启动服务，例如使用 `systemctl enable --now firewalld`，或在 Ubuntu 上使用 `ufw enable`。
2. 按照下文所述配置端口规则。
3. 使用 `firewall-cmd --list-all` 或 `ufw status verbose` 确认最终规则集。

主机防火墙

平台不要求使用特定的主机防火墙实现。RHEL 和 Kylin Linux Advanced Server 节点通常使用 `firewalld`，Ubuntu 节点通常使用 `ufw`。无论使用哪种实现，都必须允许相同的端口集合。端口列表请参阅通信矩阵。

> 危险
>
> 通信矩阵列出了节点侦听的端口。Pod 到 Pod 和 Pod 到 Service 的流量使用任意端口，不在该列表范围内，因此仅允许列表中端口的防火墙会破坏集群网络。启用主机防火墙时，还必须将 Pod CIDR、Service CIDR 以及 CNI 插件创建的网络接口视为可信来源，以免按端口过滤集群内部流量。接口名称取决于 CNI 插件和环境，因此请从节点读取接口名称，不要假定使用固定名称。

运行安装程序的节点还会使用 `8080`、`12080`、`12443` 和 `16443`。开始安装前，请在该节点上允许这些端口。

删除冲突的软件包

安装之前，节点上的 docker/nerdctl/containerd 环境中可能已经运行应用，或者可能已经安装了与 `global` 集群冲突的软件。因此，需要检查并卸载冲突的软件包。

> 危险
>
> - 为避免应用中断或数据丢失，请务必确认是否存在冲突的软件包。发现冲突时，请在卸载前制定应用切换计划并备份数据。
> - 卸载冲突的软件包后，仍需检查 `/usr/local/bin/` 等目录中是否存在其他可能冲突的二进制文件（例如与 docker、nerdctl、containerd、runc、podman、容器网络、容器运行时或 Kubernetes 相关的软件）。

以下命令可供参考。

RHEL

检查：

```
for x in \
    docker docker-client docker-common docker-latest \
    podman-docker podman \
    runc \
    containernetworking-plugins \
    apptainer \
    kubernetes kubernetes-master kubernetes-node kubernetes-client \
    ; do
    rpm -qa | grep -F "$x"
done
```

卸载：

```
for x in \
    docker docker-client docker-common docker-latest \
    podman-docker podman \
    runc \
    containernetworking-plugins \
    apptainer \
    kubernetes kubernetes-master kubernetes-node kubernetes-client \
    ; do
    yum remove "$x"
done
```

Ubuntu

检查：

```
for x in \
    docker.io \
    podman-docker \
    containerd \
    rootlesskit \
    rkt \
    containernetworking-plugins \
    kubernetes \
    ; do
    dpkg-query -l | grep -F "$x"
done

for x in \
    kubernetes-worker \
    kubectl kube-proxy kube-scheduler kube-controller-manager kube-ap
iserver \
    k8s microk8s \
    kubeadm kubelet \
    ; do
    snap list | grep -F "$x"
done
```

卸载：

```
for x in \
    docker.io \
    podman-docker \
    containerd \
    rootlesskit \
    rkt \
    containernetworking-plugins \
    kubernetes \
    ; do
    apt-get purge "$x"
done

for x in \
    kubernetes-worker \
    kubectl kube-proxy kube-scheduler kube-controller-manager kube-ap
iserver \
    k8s microk8s \
    kubeadm kubelet \
    ; do
    snap remove --purge "$x"
done
```

Kylin

检查：

```
for x in \
    docker docker-client docker-common \
    docker-engine docker-proxy docker-runc \
    podman-docker podman \
    containernetworking-plugins \
    apptainer \
    containerd \
    kubernetes kubernetes-master kubernetes-node kubernetes-client ku
bernetes-kubeadm \
    ; do
    rpm -qa | grep -F "$x"
done
```

卸载：

```
for x in \
    docker docker-client docker-common \
    docker-engine docker-proxy docker-runc \
    podman-docker podman \
    containernetworking-plugins \
    apptainer \
    containerd \
    kubernetes kubernetes-master kubernetes-node kubernetes-client ku
bernetes-kubeadm \
    ; do
    yum remove "$x"
done
```

配置搜索域

在 Linux OS 中，`/etc/resolv.conf` 文件用于配置 DNS 客户端域名解析设置。`search` 行指定 DNS 查询的域搜索路径。

配置要求

- 域数量：`search` 行中的域数量应小于 `domainCountLimit - 3`（默认 `domainCountLimit` 为 32）。
- 单个域的长度：每个域名不得超过 253 个字符。
- 总字符长度：所有域名及空格的字符总数不得超过 `MaxDNSSearchListChar`（默认值为 2048）。

示例

```
search domain1.com domain2.com domain3.com
```

- 域总数为 3。
- 单个域的长度（例如 `domain1.com`）为 11。
- 总字符长度为 35，即 11 + 11 + 11 + 2（两个空格）。

> 警告

- 如果 `search` 文件中的 `/etc/resolv.conf` 行不符合上述限制，可能会导致 DNS 查询失败或性能下降。
- 修改 `/etc/resolv.conf` 文件前，建议先备份该文件。

安装

本节介绍如何安装 ACP Core 和所需的 Aligned Extensions，并部署 `global` 集群。

> **安装路径**
>
> 此安装路径适用于运行在非 **Alauda OS** 上的集群。如果您的环境使用 Alauda OS，请改为参阅 Alauda OS 上安装 global 集群。

开始安装前，请确保已完成先决条件检查，下载并验证 Core Package 和所需的 Aligned Extension 软件包，完成节点预处理以及其他准备工作。如果您选择了外部平台镜像 Registry，则必须在运行 `setup.sh` 前准备好完整的目标版本负载。

目录

- 流程
  - 上传并解压安装包
  - 准备所需的 Aligned Extension 软件包
  - 启动安装程序
    - IP 系列
  - 参数配置
- 验证安装
- 安装程序参数参考
- 安装故障排除
- 其他资源

流程

1 上传并解压安装包

将 Core Package 安装包上传到 `global` 集群控制平面节点中的任意一台机器，并根据以下命令进行解压：

```
# Assume that the /root/cpaas-install folder already exists on the machine
tar -xvf {Path to Core Package File}/{Core Package File Name} -C /root/cpaas-install
cd /root/cpaas-install/installer || exit 1
```

> 信息
>
> - `global` 集群安装完成后，此机器将成为第一个控制平面节点。
> - 解压 Core Package 后，至少需要 **100GB** 的磁盘空间。请确保有足够的存储资源。
> - Core Package 不包含所需的 Aligned Extension 软件包。请按照下载中的说明单独准备这些软件包。

2 准备所需的 **Aligned Extension** 软件包

将所需的 Aligned Extensions 中列出的八个软件包分别复制到解压后的 Core Package 的 `plugins/` 目录中。对每个软件包运行一次以下命令：

```
cp <path-to-extension-package> /root/cpaas-install/installer/plugins/
```

启动安装程序前，请确认该目录包含所需的八个软件包：

```
ls -1 /root/cpaas-install/installer/plugins/
```

请勿将 **Common ACP Extensions** 中的其他软件包复制到此目录。安装程序只处理您手动放入 `plugins/` 的软件包；它不会从 Customer Portal 下载软件包。

3 启动安装程序

选择与平台镜像源匹配的命令。安装程序成功启动后，终端会输出 Web 控制台访问地址。

等待约 5 分钟后，您可以使用 PC 上的浏览器访问安装程序提供的 Web 控制台。

对于平台内置 Registry：

```
bash setup.sh
```

对于允许匿名拉取的外部 Registry：

```
bash setup.sh --registry "<registry-address>"
```

对于经过身份验证的外部 Registry，请同时提供拉取用户名和密码。`setup.sh` 使用的拉取账户可以不同于用于上传负载的推送账户：

```
export REGISTRY_ADDRESS="registry.example.com"
export REGISTRY_USERNAME="<registry-pull-username>"
read -rsp "Registry password: " REGISTRY_PASSWORD
export REGISTRY_PASSWORD
printf '\n'

bash setup.sh \
  --registry "$REGISTRY_ADDRESS" \
  --username "$REGISTRY_USERNAME" \
  --password "$REGISTRY_PASSWORD"

unset REGISTRY_PASSWORD
```

Registry 地址必须使用 `<host-or-IP>[:port]` 的形式，不能包含 `http://` 或 `https://`。请勿只提供 `--username` 和 `--password` 中的一个。

> **警告**
>
> 请确保可以正常访问安装程序所在节点的 IP 地址和 8080 端口，以便安装程序成功启动后能够顺利访问其提供的 Web 控制台。

IP 系列

```
bash setup.sh --ip-family ipv6
```

如果您计划创建使用单栈网络 IPv6 的 `global` 集群，则必须在启动安装程序时显式指定 `--ip-family ipv6`。如果不指定此参数，安装程序创建的 `global` 集群默认支持单栈网络 IPv4 和双栈网络。

4 参数配置

在 Web UI 中配置安装，检查每个值，然后确认安装。有关各字段的参考信息，请参阅安装程序参数。

验证安装

安装程序报告安装完成后，在继续执行安装后任务之前，请验证 Core 和所需的 Aligned Extensions。有关检查清单和命令，请参阅验证。

安装程序参数参考

有关字段的详细参考信息，请参阅安装程序参数。

安装故障排除

有关安装程序停滞、失败的资源、镜像拉取失败、缺少构件和清理，请参阅安装故障排除。

其他资源

- 上传和安装扩展
- 安装程序参数
- 验证
- 安装故障排除
- 后续步骤

安装程序参数

在安装程序 Web UI 中配置 `global` 集群时，请参考此内容。提交安装前，请确认网络、访问方式、镜像源和节点相关设置。

参数	说明
Kubernetes 版本	所有可选版本都经过严格测试，以确保稳定性和兼容性。 建议：选择最新版本，以获得最佳功能和支持。
集群网络协议	支持三种模式：IPv4 单栈、IPv6 单栈、IPv4/IPv6 双栈。 注意：如果选择双栈模式，请确保所有节点都已正确配置 IPv6 地址；设置后无法更改网络协议。
集群端点	输入预先准备好的域名。 如果没有可用域名，请输入预先准备好的 `global VIP`。 默认情况下禁用 `自建 VIP`，只有在未提供负载均衡器时才启用此选项。 使用 `自建 VIP` 时，必须满足以下条件： • 有可用的 VRID； • 主机网络支持 VRRP 协议； • 所有控制平面节点和 VIP 必须位于同一子网。

	提示：在功能体验场景中进行单节点部署时，可直接输入节点 IP。无需启用 `自建 VIP`，也无需准备 `global VIP` 等网络资源。
平台访问地址	如果无需区分集群端点和平台访问地址，请输入与集群端点相同的地址。 如果需要区分，例如 `global` 集群仅供内网访问，而平台需要提供外网访问，请输入预先准备好的域名或 `外部 IP`。 平台默认使用 HTTPS 访问，不启用 HTTP。如果需要启用 HTTP 访问，请在高级设置中启用（不推荐）。 注意：在以下情况下必须输入域名： • 计划为 `global` 集群制定灾难恢复方案； • 平台需要支持 IPv6 访问。 提示：如果需要配置更多平台访问地址，可在下一步的其他设置 **>** 其他平台访问地址中添加。也可以在安装后根据用户手册在平台管理中添加。
证书	平台默认提供自签名证书以支持 HTTPS 访问。 如果需要使用自定义证书，可以上传已有证书。
镜像仓库	`平台部署` 使用与 `global` 集群一同部署的 Registry。 `外部` 使用由客户管理的平台镜像仓库。其地址和拉取凭据必须与传递给 `setup.sh` 的值一致，且该仓库必须已包含目标版本的完整 Core 和 Extension 负载。
容器网络	集群的默认子网与 Service 网络段不能重叠。 使用 Kube-OVN Overlay 网络时，请确保容器网络与主机网络不在同一网段，否则可能导致网络异常。

节点名称	如果选择 `主机名作为节点名称`，请确保所有节点的主机名唯一。
`global` 集群平台节点隔离	仅当计划在 `global` 集群中运行原生应用工作负载时启用。 启用后： • 节点可设置为 `平台专用`，即仅运行平台组件，以确保平台工作负载与原生应用工作负载相互隔离； • DaemonSet 类型的工作负载不受此设置影响。
添加节点	控制平面节点： • 支持添加 1 个或 3 个控制平面节点（高可用配置需添加 3 个）； • 如果启用 `平台专用`，则会强制禁用 `可部署原生应用`，控制平面节点仅运行平台组件； • 如果禁用 `平台专用`，可以选择是否启用 `可部署原生应用`，以允许控制平面节点运行原生应用工作负载。 工作节点： • 如果启用 `平台专用`，则会强制禁用 `可部署原生应用`； • 如果禁用 `平台专用`，则会强制启用 `可部署原生应用`。 使用 Kube-OVN 时，可通过输入网关名称指定节点网卡。 如果节点可用性检查失败，请根据页面提示进行调整，然后重新添加。

查看参数后，返回安装提交安装。

Global 集群灾难恢复

> **灾难恢复路径**
>
> 本页面介绍在非 **Alauda OS** 上运行的 `global` 集群的灾难恢复。如果 `global` 集群使用 Alauda OS，请改为参阅 Alauda OS 上的 Global 集群灾难恢复。

目录

- 概述
  - 支持的灾难场景
  - 不支持的灾难场景
- 注意事项
- 流程概述
- 所需资源
  - 网络要求
- 操作步骤
  - 步骤 1：安装主集群
  - 步骤 2：安装备用集群
  - 步骤 3：启用 etcd 同步
- 灾难恢复流程
- 例行检查
- 上架软件包

概述

此解决方案适用于涉及 `global` 集群的灾难恢复场景。`global` 集群作为平台的控制平面，负责管理其他集群。为确保在 `global` 集群发生故障时平台服务持续可用，此解决方案会部署两个 `global` 集群：主集群和备用集群。

灾难恢复机制基于将主集群的 etcd 数据实时同步到备用集群。如果主集群因故障而不可用，服务可以快速切换到备用集群。

支持的灾难场景

- 主集群发生不可恢复的系统级故障，导致其无法运行；
- 承载主集群的物理机或虚拟机发生故障，导致集群无法访问；
- 主集群所在位置发生网络故障，导致服务中断；

不支持的灾难场景

- 部署在 `global` 集群中的应用发生故障；
- 存储系统故障导致的数据丢失（不在 etcd 同步范围内）；

主集群和备用集群的角色是相对的：当前为平台提供服务的集群是主集群（DNS 指向该集群），另一个集群则是备用集群。发生故障转移后，这两个角色会互换。

注意事项

- 此解决方案仅同步 `global` 集群的 etcd 数据，不包括 registry、chartmuseum 或其他组件的数据；
- 监控和日志数据也不会同步。请勿将业务集群的监控或日志数据存储在主集群或备用集群中：应在专用业务集群中安装 VictoriaMetrics Center 和日志存储，以汇总业务集群数据，并让每个 `global` 集群仅收集、存储和查询自身的指标和日志。故障转移后，主集群中的可观测性数据将不可用，因此依赖这些数据的备用集群无法观测其刚接管的平台；
- 为便于故障排查和管理，建议采用类似 `standby-global-m1` 的节点命名方式，以指明节点所属的集群（主集群或备用集群）。

- 不支持集群内应用数据的灾难恢复；
- 两个集群之间需要稳定的网络连接，以确保 etcd 可靠同步；
- 如果集群基于异构架构（例如 x86 和 ARM），请使用双架构安装包；
- 以下命名空间不会进行 etcd 同步。如果在这些命名空间中创建资源，用户必须手动备份：

```
cpaas-system
cert-manager
default
global-credentials
cpaas-system-global-credentials
kube-ovn
kube-public
kube-system
nsx-system
cpaas-solution
kube-node-lease
kubevirt
nativestor-system
operators
```

- 如果两个集群都配置为使用内置镜像仓库，则必须分别向每个集群上传容器镜像；
- 如果主集群部署了 灵雀云 **DevOps Eventing v3** (knative-operator) 及其实例，则必须在备用集群中预先部署相同的组件。

流程概述

1. 为平台访问准备一个统一域名；
2. 将该域名指向主集群的 VIP，并安装主集群；
3. 临时将 DNS 解析切换到备用 VIP，以安装备用集群；
4. 将主集群的 ETCD 加密密钥复制到之后将作为备用集群控制平面节点的节点上；
5. 安装并启用 灵雀云容器平台 **Global Cluster Disaster Recovery**；
6. 验证同步状态并定期检查；
7. 发生故障时，将 DNS 切换到备用集群以完成灾难恢复。

所需资源

- 一个统一域名，该域名将作为 `Platform Access Address`，以及用于在该域名上提供 HTTPS 服务的 TLS 证书和私钥；
- 每个集群各需一个专用虚拟 IP 地址——一个用于主集群，另一个用于备用集群；

网络要求

统一域名是灾难恢复的前提条件。正常运行期间，该域名仅解析到主集群 VIP。备用集群使用该域名访问主集群上的平台服务。

安装任一集群之前，请完成以下两项网络准备：

- 配置每个集群的负载均衡器，将 VIP 上所需的 TCP 端口转发到该 VIP 后端的控制平面节点。仅当用户通过 HTTP 访问平台时，才配置端口 `80`。
- 在主集群和备用集群的网络之间，双向开放所需的 TCP 端口。将规则应用于防火墙、安全组、路由器 ACL 以及任何其他站点间网络控制措施。

TCP 端口	服务	所需原因
`443`	ACP platform HTTPS	备用集群使用平台访问地址访问当前活动的平台。灵雀云容器平台 **Global Cluster Disaster Recovery** 使用此地址，日志和监控组件也会向该地址发送告警回调。
`80`	ACP platform HTTP	仅当用户通过 HTTP 访问平台时才需要。其他情况下的连通性要求与端口 `443` 相同。
`6443`	Kubernetes API server	安装或重新安装 灵雀云容器平台 **Global Cluster Disaster Recovery** 期间，备用集群会连接到当前活动集群的 API server，以获取同步所需的 etcd CA 材料。
`11443`	内置镜像仓库	备用集群从通过平台访问地址配置的镜像仓库拉取平台和插件镜像。
`2379`	etcd	备用集群上的 灵雀云容器平台 **Global Cluster Disaster Recovery** 从主集群读取 etcd 数据，并将其写入本地备用 etcd。

网络策略必须是双向的，因为故障转移后集群角色会互换。故障转移前，实际服务流量通常从备用集群流向主集群。故障转移后，原主集群成为新的备用集群，必须通过相同端口访问新的主集群。这项要求并不意味着 etcd 复制是双向的：灵雀云容器平台 **Global Cluster Disaster Recovery** 仅在当前备用集群上运行，并将同步的数据写入其本地 etcd。

操作步骤

步骤 1：安装主集群

> 灾难恢复环境安装注意事项
>
> 安装灾难恢复环境的主集群时：
>
> - 首先，记录按照安装 Web UI 指引设置的所有参数。安装备用集群时，部分选项必须保持一致。
> - 必须预先配置 **User-provisioned** 负载均衡器，以路由发送到虚拟 IP 的流量。`Self-built VIP` 选项不可用。
> - `Platform Access Address` 字段必须填写域名，而 `Cluster Endpoint` 必须填写虚拟 IP 地址。
> - 两个集群都必须配置为使用 `An Existing Certificate`（必须相同），必要时申请有效证书。`Self-signed Certificate` 选项不可用。
> - 当 `Image Repository` 设置为 `Platform Deployment` 时，`Username` 和 `Password` 字段都不能为空；`IP/Domain` 字段必须设置为用作 `Platform Access Address` 的域名。
> - `Platform Access Address` 的 `HTTP Port` 和 `HTTPS Port` 字段都必须设置为 80 和 443。
> - 进入安装指南的第二页（步骤：`Advanced`）时，`Other Platform Access Addresses` 字段必须包含当前集群的虚拟 IP。

请参阅以下文档以完成安装：

- 安装准备
- 安装

步骤 2：安装备用集群

1. 暂时将域名指向备用集群的 VIP；
2. 登录到主集群的第一个控制平面节点，并将 etcd 加密配置复制到备用集群的所有控制平面节点：

```
# Assume the primary cluster control plane nodes are 1.1.1.1, 2.2.2.2 &
3.3.3.3
# and the standby cluster control plane nodes are 4.4.4.4, 5.5.5.5 & 6.
6.6.6
for i in 4.4.4.4 5.5.5.5 6.6.6.6  # Replace with standby cluster contro
l plane node IPs
do
  ssh "<user>@$i" "sudo mkdir -p /etc/kubernetes/"
  scp /etc/kubernetes/encryption-provider.conf "<user>@$i:/tmp/encrypti
on-provider.conf"
  ssh "<user>@$i" "sudo install -o root -g root -m 600 /tmp/encryption-
provider.conf /etc/kubernetes/encryption-provider.conf && rm -f /tmp/en
cryption-provider.conf"
done
```

3. 按照与主集群相同的方式安装备用集群

> **安装备用集群注意事项**
>
> 安装灾难恢复环境的备用集群时， 必须将以下选项设置为与主集群相同：
>
> - `Platform Access Address` 字段。
> - `Certificate` 的所有字段。
> - `Image Repository` 的所有字段。
> - 重要：确保镜像仓库凭据和 ACP 管理员用户的凭据与主集群上设置的凭据一致。
>
> 并确保已按照步骤 1 中的 `NOTES OF DR (Disaster Recovery Environment) INSTALLING` 操作。

请参阅以下文档以完成安装：

- 安装准备
- 安装

步骤 3：启用 etcd 同步

1. 安装插件之前，从活动 global 集群获取用于访问其 API server 的 bearer token。复制命令输出，并在备用 global 集群的 `cpaas-system` 命名空间中创建 `etcd-sync-active-cluster-token` Secret 时使用该输出。该 Secret 使用数据键 `token` 存储令牌。

   ```
   # Run this command on the active cluster.
   kubectl -n cpaas-system get secret k8sadmin -o jsonpath='{.data.token}'
   | base64 -d
   ```

   复制输出值，然后在备用集群上运行此命令：

   ```
   ACTIVE_CLUSTER_TOKEN='<paste-the-token-from-the-active-cluster>'
   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 -
   ```

2. 如适用，配置负载均衡器，将端口 `2379` 转发到相应集群的控制平面节点。仅支持 TCP 模式；不支持 L7 转发。

   > **INFO**
   >
   > 无需通过负载均衡器转发端口。 如果备用集群可以直接访问活动 global 集群，请通过活动 **Global** 集群 **ETCD** 端点指定 etcd 地址。

3. 使用 VIP 访问备用 **global** 集群 Web Console，并切换到管理员视图。
4. 转到 **Marketplace > Cluster Plugins**，选择 `global` 集群。
5. 找到 灵雀云容器平台 **Global Cluster Disaster Recovery**，单击安装，并配置以下参数：

   - 将活动 **Global** 集群 **VIP**设置为活动 global 集群的 VIP。
   - 如果未通过负载均衡器转发端口 `2379`，请正确配置活动 **Global** 集群 **ETCD** 端点。
   - 将备用集群 **ETCD** 端点设置为备用集群的 etcd 地址。除非本地 etcd 服务通过其他端点公开，否则请使用默认值。
   - 将活动 **Global** 集群令牌 **Secret**设置为 `etcd-sync-active-cluster-token`。

- 数据检查间隔使用默认值。
- 除非正在排查问题，否则请保持禁用打印详细日志。

安装期间，系统会先运行 `etcd-sync-bootstrap` Job，然后启动 `etcd-sync` Deployment。Job 准备好 `remote-etcd-ca`、`remote-etcd-issuer` 和 `remote-etcd-client` 后，插件安装才会继续。

验证引导 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
```

验证同步 Pod 在备用集群上运行，并确定当前 leader：

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

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

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

检查同步状态：

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

输出说明：

- `LOCAL ETCD missed keys`：主集群中存在但备用集群中缺失的键。通常是由于同步期间资源顺序导致 GC 引起的。重启当前 etcd-sync leader Pod 可解决此问题；
- `LOCAL ETCD surplus keys`：仅存在于备用集群中的额外键。删除这些键之前，请先与运维团队确认。

如果已安装以下组件，请重启其服务：

- 灵雀云容器平台 Elasticsearch 日志存储：

  ```
  kubectl delete po -n cpaas-system -l service_name=cpaas-elasticsearch
  ```

- 灵雀云容器平台 高级监控：

  ```
  kubectl delete po -n cpaas-system -l 'service_name in (alertmanager,vmselect,vminsert)'
  ```

灾难恢复流程

> **卸载 Alauda Container Platform Global Cluster Disaster Recovery 前，请验证主集群与备用集群的一致性**
>
> 此操作将卸载 灵雀云容器平台 **Global Cluster Disaster Recovery**。卸载前，请确认备用集群的数据与主集群一致。如果主集群持有某些数据而备用集群缺少这些数据，此时卸载插件可能导致 owner 引用解析错误，并可能删除业务集群节点的 `Machine` 对象，包括 Alauda OS 集群；在这种情况

下，支撑虚拟机也会被销毁。如果一致性检查报告存在缺失或多余的键，请勿卸载插件；请先解决不一致问题或联系技术支持。

1. 如有必要，重启备用集群上的 Elasticsearch：

```
# Copy installer/res/packaged-scripts/for-upgrade/ensure-asm-template.sh to /root:
# DO NOT skip this step

# switch to the root user if necessary
sudo -i

# check whether the Log Storage for Elasticsearch is installed on global cluster
_es_pods=$(kubectl get po -n cpaas-system | grep cpaas-elasticsearch | awk '{print $1}')
if [[ -n "${_es_pods}" ]]; then
    # In case the script returned the 401 error, restart Elasticsearch
    # then execute the script to check the cluster again
    bash /root/ensure-asm-template.sh

    # Restart Elasticsearch
    xargs -r -t -- kubectl delete po -n cpaas-system <<< "${_es_pods}"
fi
```

2. 验证备用集群中的数据一致性（与步骤 3中的检查相同）。如果检查报告存在缺失或多余的键，则备用集群与主集群不一致：请勿继续下一步。请先解决不一致问题或联系技术支持。在两个集群上，还需确认没有处于非运行状态的 `Machine` 节点，并在继续操作前解决所有此类问题：

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

3. 卸载 灵雀云容器平台 **Global Cluster Disaster Recovery**；
4. 移除两个 VIP 上针对 `2379` 的端口转发；
5. 将平台域名 DNS 切换到备用 VIP；此时备用集群将成为主集群；
6. 验证 DNS 解析：

```
kubectl exec -it -n cpaas-system deployments/sentry -- nslookup <platfo
rm access domain>
# If not resolved correctly, restart coredns Pods and retry until succe
ss
```

7. 清除浏览器缓存并访问平台页面，确认页面显示的是原备用集群；
8. 重启以下服务（如果已安装）：
   - 灵雀云容器平台 Elasticsearch 日志存储：

     ```
     kubectl delete po -n cpaas-system -l service_name=cpaas-elasticsearch
     ```

   - 灵雀云容器平台 高级监控：

     ```
     kubectl delete po -n cpaas-system -l 'service_name in (alertmanager,v
     mselect,vminsert)'
     ```

   - cluster-transformer：

     ```
     kubectl delete po -n cpaas-system -l service_name=cluster-transformer
     ```

9. 如果业务集群将监控数据发送到主集群，请在业务集群中重启 warlock：

   ```
   kubectl delete po -n cpaas-system -l service_name=warlock
   ```

10. 在原主集群上，重复启用 etcd 同步的步骤，将其转换为新的备用集群。

例行检查

定期在备用集群上检查同步状态：

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

如果存在缺失或多余的键，请按照输出中的说明解决。

上架软件包

有关 `violet push` 子命令的详细信息，请参阅上架软件包。

安装故障排查

当安装程序无法继续执行、安装资源失败，或安装后验证报告映像或构件问题时，请使用此页面。

目录

常见卡顿及查看位置

映像拉取和构件失败

安装程序清理

在提交支持请求前收集证据

常见卡顿及查看位置

在更改环境或提交支持工单之前，先收集第二列中的信息。

症状	首先查看的位置	要查找的内容
Web UI 无法加载	安装节点上 `setup.sh` 和 `nerdctl ps` 的终端输出	`minialauda-control-plane` 容器处于 Running 状态，并且浏览器可以访问端口 `8080`。
安装程序日志停止推进	`tail -f /var/cpaas/data/installer.log`	最后完成的阶段，以及第一个重复或失败的操作。

症状	首先查看的位置	要查找的内容
卡在控制平面预配	非 `Ready` 机器上的 `kubectl get machines -A` 和 `kubectl describe`	`Bootstrap` 和 `Infrastructure` 条件，以及 `status.addresses` 中的节点地址。
卡在网络和 Core 附加组件	新集群上的 `kubectl -n kube-system get pods`	Kube-OVN、CoreDNS 和 kube-proxy Pod 处于 `Running` 状态；映像拉取失败通常表明存在 registry 路径问题。
AppRelease 显示 `Failed`	`kubectl describe apprelease <name> -n <namespace>`	用于标识 chart 或依赖项故障的状态条件和近期 Events。
`ClusterModule/global` 未变为健康状态	`kubectl describe clustermodule global`	用于标识阻塞 Core 或 Aligned 模块的状态条件。

映像拉取和构件失败

症状	检查	恢复
Pod 处于 `ImagePullBackOff` 或 `ErrImagePull` 状态	运行 `kubectl describe pod <pod> -n <namespace>`，并从发生故障的节点测试配置的 registry。	修正报告的 DNS、路由、防火墙、CA 信任、拉取凭证、registry 可用性或缺少 manifest 的问题。
安装后所需的 Core 或 Aligned 原生应用不可用	检查对应的 `AppRelease`、`ModuleInfo`、`ModulePlugin` 和 Pod Events。如果仍不清楚软件包是否可用，请检查 `kubectl get productbase base -o yaml` 中的匹配条目，以及 registry 中准确的 `repository:tag`。	如果 registry 确认缺少所需的 manifest，请上传匹配的安装版本软件包。对于授权、CA、DNS、路由或连接错误，请修正报告的 registry 问题，而不是重新发布软件包。忽略可选或未安装 catalog 软件包的无关 `Absent` 条目。

症状	检查	恢复
已恢复或替换的节点无法拉取已安装版本的映像	检查发生故障的 Pod 和 registry 保留记录。	恢复引用的 digest，然后在重试恢复之前修正 registry 可用性、备份或保留策略。

对于在安装程序启动前发生的上传命令失败，请使用External Platform Image Registry Troubleshooting。

安装程序清理

安装程序通常会在安装完成后自动删除自身。如果安装完成 30 分钟后安装程序容器仍然存在，请在安装节点上运行以下命令：

```
nerdctl rm -f minialauda-control-plane
```

在提交支持请求前收集证据

收集 `/var/cpaas/data/installer.log`、相关的 `kubectl describe` 输出、失败资源的状况，以及准确的 Core Package 版本和架构。请勿包含 registry 密码或 Secret 内容。

验证

安装完成后、继续执行安装后任务之前，请验证 ACP Core 和必需的 Aligned Extensions。这些检查适用于两种安装路径。如果 `global` 集群运行的是 Alauda OS，还需完成 Alauda OS 的其他检查。

目录

前提条件

验证 Web UI 访问

验证核心应用、扩展和 Pod

验证平台映像源

Alauda OS 的其他检查

后续步骤

前提条件

- ACP 安装已完成。
- 您可以访问平台 Web UI 地址：在非 Alauda OS 上，该地址由安装程序显示；在 Alauda OS 上，该地址是在 `console.globalHost` 中设置的平台访问地址。
- 您可以对新的 `global` 集群运行 `kubectl`：在非 Alauda OS 上，从安装节点运行；在 Alauda OS 上，使用新 `global` 集群的 kubeconfig 运行。在 Alauda OS 上，请在本页的命令中添加 `--kubeconfig <global-kubeconfig>`。

验证 **Web UI** 访问

安装完成后，打开平台 Web UI，并确认平台登录页面可以访问。在非 Alauda OS 上，安装程序会显示平台访问 URL；点击 访问 将其打开。在 Alauda OS 上，打开 `https://<platform-access-address>`，即在 `console.globalHost` 中设置的地址。

验证核心应用、扩展和 **Pod**

对新的 `global` 集群运行以下命令，以确认安装状态。在非 Alauda OS 上，请在控制平面节点上运行这些命令。

```
# All nodes are Ready
kubectl get nodes

# ClusterModule/global reports the base module as healthy
kubectl get clustermodule global -o jsonpath='{.status.phase}'

# No AppRelease is in a failed state
kubectl get apprelease -A

# Core and Aligned modules report their current phase and version
kubectl get moduleinfo -l cpaas.io/cluster-name=global

# No Pod is stuck outside Running or Completed
kubectl get pod --all-namespaces | awk '{if ($4 != "Running" && $4 != "Co
mpleted")print}' | awk -F'[/ ]+' '{if ($3 != $4)print}'
```

以下情况表明安装运行正常：

- 所有 `global` 集群节点均处于 `Ready` 状态。
- `ClusterModule/global` 报告的阶段状态正常。
- 所有 AppRelease 均未处于失败状态。
- 已安装以下应用，且对应的 `ModuleInfo` 资源在目标版本上报告 `Running`：
  - **Alauda Container Platform Web Console**
  - **Alauda Container Platform Web Console Central**

- **Alauda Container Platform Marketplace Web Console**
- **Alauda Container Platform Application Catalog**
- **Alauda Container Platform Observability Essentials**
- **Alauda Container Platform Essentials**
- **Alauda Container Platform Cluster Essentials**
- **Alauda Container Platform GatewayAPI Plugin**

- `cpaas-system` 中的关键 Pod 处于 `Running` 或 `Completed` 状态。

在 Web Console 中，打开 管理员 **> Marketplace >** 集群插件，并确认所需应用均可用且未发生安装失败。

验证平台映像源

确认 `ProductBase` 记录了预期的平台映像源：

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

对于外部 registry，地址必须与准备好的 registry 匹配，且第二个值必须为 `true`。

对于需要身份验证的外部 registry，确认拉取 Secret 存在，但不要打印其内容：

```
kubectl -n cpaas-system get secret global-registry-auth
```

确认 `global` 集群记录了相同的 registry 地址：

```
kubectl -n cpaas-system get cluster global \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}{"\n"}'
```

`ProductBase` 还包含可选或未安装 Extension 的目录条目。不要要求 `ProductBase.status.artifacts` 中的每个条目都为 `Ready`；无关的 `Absent` 条目本身并不表示安装失败。

确认所需的 Extension 目录条目存在，并且 Pod 未因映像拉取而受阻：

```
kubectl get moduleplugins

kubectl get pods --all-namespaces \
  | awk '$4 ~ /ImagePullBackOff|ErrImagePull/'
```

Pod 命令不应产生任何输出。安装验收依据是所需应用、其 `ModuleInfo` 和 `AppRelease` 状态，以及其运行中的 Pod。如果所需的已安装应用不可用，请参阅安装故障排除，以区分清单缺失与 DNS、路由、防火墙、CA 信任、拉取凭证或 registry 可用性问题。

Alauda OS 的其他检查

如果 `global` 集群运行 Alauda OS，还应使用新 `global` 集群的 kubeconfig 执行以下验证。在停用引导集群之前完成这些检查。

```
kubectl --kubeconfig <global-kubeconfig> get nodes
kubectl --kubeconfig <global-kubeconfig> get clusters.platform.tkestack.i
o global \
  -o jsonpath='{.status.phase}'
kubectl --kubeconfig <global-kubeconfig> get pods -n cpaas-system
kubectl --kubeconfig <global-kubeconfig> get clustermodule global
```

满足以下所有条件时，表示安装成功：

- 安装程序进度 API 报告 `status: Success` 和 `type: Complete`。请参阅安装期间的信号。
- 所有 `global` 集群节点均为 `Ready`。
- `cpaas-system` 中的关键 Pod 为 `Running` 或 `Completed`。
- `ClusterModule/global` 报告基础模块运行正常。

后续步骤

验证成功后，继续执行后续步骤。在 Alauda OS 上，先停用引导集群。

后续步骤

Core 和所需的 Aligned Extensions 通过验证后，请根据您的场景完成以下安装后任务。

优先级	任务	使用时机	继续执行
建议立即执行	安装 Product Docs 插件。	安装此插件，以便控制台中的帮助链接可以打开产品文档。	安装 Product Docs 插件
有条件	创建工作负载集群或连接现有集群。	当您的架构要求工作负载集群与 `global` 集群分离时，使用此路径。要在 Alauda OS 上创建工作负载集群，请先在 `global` 集群上为目标 IaaS 平台上传并安装基础设施提供商和 Cluster Bootstrap Provider，无论 `global` 集群运行在哪个 IaaS 平台上。	集群
有条件	上传并安装其他 Extensions Packages。	对于 **Common ACP Extensions** 中未包含在八个必需安装包内的软件包，或当您需要其他平台功能时，使用此路径。	扩展
建议在广泛引入用户之前执行	配置身份提供商、用户、角色、项目和项目成员关系。	在引入管理员、项目用户或应用团队之前，使用此路径。	用户和角色 和 项目
建议在应用工作负载之前执行	准备存储、网络、Registry 访问权限和工作负载前置条件。	在应用团队部署工作负载之前，使用此路径。	存储、网络、Registry 和 工作负载集群

优先级	任务	使用时机	继续执行
建议在生产运行之前执行	查看备份、升级、监控和运维文档。	在针对生产工作负载运行平台之前，使用此路径。	备份、升级 和 可观测性

这些任务并不适用于每种安装。请遵循与您的部署架构和运维要求相匹配的路径。

目录

安装 Product Docs 插件

打开集群插件

安装插件

安装 **Product Docs** 插件

灵雀云容器平台 **Product Docs** 插件可在平台内访问产品文档。平台中的所有帮助链接都会指向此文档。如果未安装此插件，点击平台中的帮助链接会导致 404 访问错误。

1 打开集群插件

导航至管理员。在左侧边栏中，点击 **Marketplace** > 集群插件，然后选择 `global` 集群。

2 安装插件

找到 灵雀云容器平台 **Product Docs** 插件，然后点击安装。