集群

概览

概览

在 **Alauda OS** 上创建

准备 裸金属 **Huawei DCS**

Huawei Cloud Stack **VMware vSphere** 故障排除

在非 **Alauda OS** 上创建

在非 **Alauda OS** 上创建

托管集群

托管集群

导入第三方集群

注册集群

公有云集群初始化

公有云集群初始化。

操作指南

操作指南

使用 **KubeConfig** 访问集群

为内置 **Registry** 添加外部地址

更新公共仓

概览

从这里开始，选择创建平台管理的集群，还是接入现有的 Kubernetes 环境。有关集群责任、控制平面拓扑和接入边界之间的平台级关系，请参阅集群管理模型。

目录

选择集群路径

平台管理的集群创建

安装程序置备的基础设施

用户置备的基础设施

托管控制平面

第三方集群接入

版本兼容性

升级先决条件行为

选择集群路径

如果您需要	从这里开始	关键边界
创建一个节点运行 Alauda OS、并通过受支持的基础设施提供商进行置备的集群。	在 Alauda OS 上创建	平台所有的生命周期管理仅适用于受支持的提供商和工作流范围。

如果您需要	从这里开始	关键边界
在您准备和维护的计算机上创建一个运行受支持的非 Alauda OS 的集群。	在非 Alauda OS 上创建	节点准备完成后，平台管理 Kubernetes；节点 OS 生命周期仍由用户负责。
接入现有的 Kubernetes 环境。	托管集群	外部所有者、发行版或提供商通常负责 Kubernetes、节点和基础设施生命周期。
在直接平台访问和反向连接接入之间进行选择。	导入第三方集群 和 注册集群	导入和注册是接入方法，而不是独立的 day-2 能力模型。

平台管理的集群创建

对于由 ACP 创建并进行生命周期管理的集群，请根据节点的操作系统选择创建路径。

节点操作系统	适用场景	继续操作
Alauda OS	平台应通过 Alauda OS 管理节点操作系统，并通过受支持的基础设施提供商置备节点。	在 Alauda OS 上创建
非 Alauda OS	您准备运行受支持操作系统的物理机或虚拟机，平台在这些节点上安装并管理 Kubernetes。	在非 Alauda OS 上创建

创建路径取决于节点操作系统，而不是下文所述的基础设施责任模型。例如，在裸金属环境中，您提供物理服务器，但节点运行 Alauda OS，因此裸金属集群使用 Alauda OS 路径。

安装程序置备的基础设施

在安装程序置备的基础设施（IPI）中，平台置备计算机，并通过 Alauda OS 管理节点操作系统。当平台负责所选提供商集成所支持的集群置备、扩展和升级生命周期时，会使用此模型。

目前，平台支持使用 Alauda OS 进行不可变节点管理。有关 Alauda OS 集群概念和特定于提供商的工作流，请参阅在 Alauda OS 上创建。

用户置备的基础设施

在用户置备的基础设施中，用户在创建集群前准备物理机或虚拟机。平台在这些节点上安装并管理 Kubernetes，而节点操作系统管理（包括置备、修补和替换）仍由用户负责。

当组织已经建立基础设施或操作系统管理流程时，此模型非常适用。有关集群创建工作流，请参阅在非 Alauda OS 上创建。

有关通过 API 创建和管理用户置备的基础设施集群，请参阅用户置备的基础设施提供商 API。

托管控制平面

Hosted Control Plane 是一种控制平面拓扑。每个托管集群都有自己的控制平面，多个托管控制平面作为工作负载运行在管理集群上。

在 ACP 中，HCP 通过 Kamaji（`TenantControlPlane`）实现。

第三方集群接入

第三方集群是由 灵雀云 之外的环境提供的现有 Kubernetes 环境。它们可以包括标准 Kubernetes 环境、外部 Kubernetes 发行版或公有云 Kubernetes 服务。在满足文档所述的先决条件和注意事项的情况下，ACP 可提供集中式治理和运维，但接入并不意味着 ACP 将负责外部集群的 Kubernetes 生命周期、节点生命周期或提供商基础设施生命周期。

第三方集群通过导入或注册工作流接入：

接入方式	连接模型	适用场景
导入集群	`global` 集群使用提供的地址、CA 和凭据连接到目标集群 API server。	平台可以访问目标集群 API server，且 operator 可以提供所需的集群信息。
注册集群	目标集群中的反向代理服务发起注册，并与平台建立隧道。	目标集群应发起连接，或对目标 API server 的平台直接访问受到限制。

提供商特定的注意事项可能涉及证书、审计数据、控制平面指标、入口、存储、节点操作、连接性和 Extension 兼容性。有关详细的先决条件，请参阅导入和注册工作流页面。

有关接入工作流，请参阅第三方集群接入、导入第三方集群和注册集群。

版本兼容性

请使用 Kubernetes 支持矩阵选择平台管理的集群创建目标版本、验证业务集群升级先决条件，并确认第三方集群接入版本。这些范围分别用于不同的决策。

升级先决条件行为

- 升级 `global` 集群之前，每个业务集群运行的 Kubernetes 版本都必须在目标版本的兼容版本范围内。
- 第三方集群接入范围独立于此升级先决条件所用的兼容版本范围。

在 Alauda OS 上创建集群

在此路径中，ACP 会创建节点运行 Alauda OS（不可变操作系统）的业务集群。与节点运行非 Alauda OS 的集群不同，所有节点配置都预先构建在镜像中，并在部署后保持不变。集群升级和配置更改通过使用新镜像替换节点来应用，从而确保整个集群生命周期内的一致性、可靠性和简化的运维。

此路径上的集群通过 Cluster API 以声明式方式创建和管理。每个受支持的 IaaS 平台都由对应的基础设施提供程序从 Alauda OS 镜像创建节点。在裸金属环境中，您需要提供物理服务器；这些服务器会启动由提供程序构建的安装 ISO，并安装 Alauda OS。

此路径遵循以下原则：

- 声明式配置：所有基础设施均以代码形式定义
- 版本控制：基础设施变更均有跟踪和版本记录
- 可复现部署：不同部署之间的环境保持一致
- 阶段感知恢复：通过可复现的机器替换安全地恢复模板；当 Kubernetes 次要版本升级更改了控制平面状态时，则使用基于备份的恢复

目录

架构

引导提供程序（Kubeadm）

控制平面提供程序（Kubeadm）

基础设施提供程序

支持的提供程序

控制平面端点

Web UI 支持（Fleet Essentials）

在 global 集群上安装提供程序

集群创建工作流

相关任务

准备

裸金属

Huawei DCS

Huawei Cloud Stack

VMware vSphere

故障排除

架构

Alauda OS 上的集群生命周期管理基于 Cluster API (CAPI) 框架构建，该框架提供声明式、原生 Kubernetes 的方法：

- 声明式 **API**：用于集群管理的原生 Kubernetes 资源
- 提供程序接口：适用于不同基础设施提供程序的标准化接口
- 生命周期管理：自动执行集群预配、扩缩和升级
- **GitOps** 集成：原生支持 GitOps 工作流

三个 Cluster API 提供程序协同工作，以两个软件包的形式交付：**Alauda Container Platform Cluster Bootstrap Provider** 负责 Kubernetes 引导和控制平面管理，基础设施提供程序则负责管理 IaaS 平台。

引导提供程序（**Kubeadm**）

适用于 kubeadm 的 Cluster API 引导提供程序负责初始化和配置 Kubernetes 节点：

- 节点初始化：自动将裸机转换为 Kubernetes 节点
- 配置管理：管理 kubelet 配置、证书和引导令牌

- 加入流程：处理工作节点加入集群的流程
- 证书管理：自动生成和轮换证书

控制平面提供程序（**Kubeadm**）

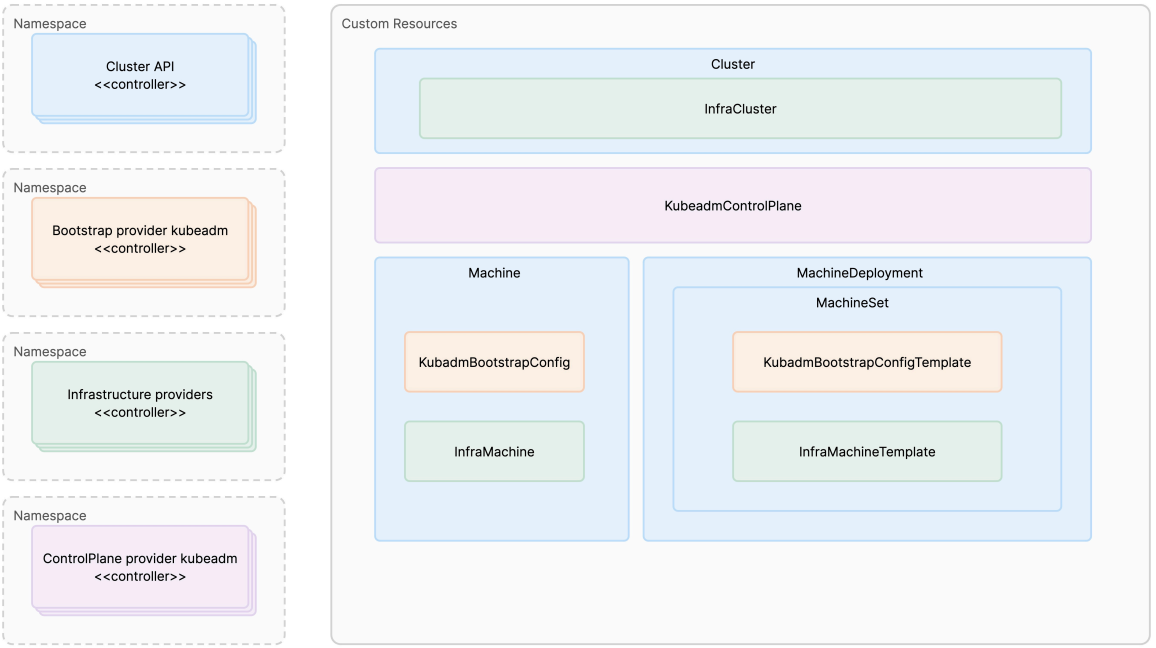
适用于 kubeadm 的 Cluster API 控制平面提供程序负责管理 Kubernetes 控制平面：

- 控制平面部署：部署和管理 Kubernetes API server、etcd 及控制平面组件
- 高可用性：支持通过负载均衡实现多节点控制平面部署
- 升级管理：处理控制平面组件的滚动升级
- 运行状况监控：监控控制平面运行状况并执行恢复操作

基础设施提供程序

目标 IaaS 平台的基础设施提供程序（例如 Huawei DCS）负责管理底层基础设施：

- 计算资源预配：在目标平台上创建和管理虚拟机或实例
- 网络配置：设置网络基础设施
- 资源管理：管理计算、存储和网络资源
- 提供程序集成：与提供程序的 API 和服务集成



支持的提供程序

每个基础设施提供程序都与一个 IaaS 平台集成，以提供集群生命周期管理。

提供程序	说明
裸金属	不使用虚拟化的裸金属服务器
Huawei DCS	Huawei Datacenter Virtualization Solution
Huawei Cloud Stack	Huawei Cloud Stack
VMware vSphere	VMware vSphere 虚拟化平台

控制平面端点

每个集群的 Kubernetes API server（`kube-apiserver`，端口 `6443`）都通过控制平面端点访问。该端点是一个虚拟 IP（VIP）或负载均衡器地址，位于控制平面节点前端。不同提供程序提供该端点的方式有所不同，因此在创建集群之前需要准备的内容也各不相同。

提供程序	控制平面端点	机制
裸金属	由提供程序创建（或由你提供）	`BaremetalCluster.spec.controlPlaneLoadBalancer.type=I` 会将 `alive` chart（`keepalived` + IPVS + `kube-lock` Leas 部署到控制平面节点；`type=External` 将 LB 委托给现有的外
Huawei DCS	由你提供，或由提供程序管理自建 VIP	外部负载均衡器、由提供程序管理的 `type: internal` 自建 VI 接访问单个控制平面节点
Huawei Cloud Stack	由提供程序创建	集群协调期间自动创建的 HCS 弹性负载均衡（ELB）实例
VMware vSphere	由提供程序创建（或由你提供）	`VSphereCluster.spec.controlPlaneLoadBalancer.type=int` 会在控制平面节点上部署 Alive（`keepalived` + IPVS）；`type=external` 或不设置该字段时，会将 LB 委托给现有的外 模式在创建集群时确定

Huawei Cloud Stack 是唯一会为你创建平台负载均衡器对象的 VM 提供程序。外部负载均衡器（包括其 VIP 分配和后端（真实服务器）维护）需在集群外部准备并由外部管理。在裸金属、Huawei DCS 和 VMware vSphere 上，提供程序也可以改为部署集群内 VIP 组件（`alive`）。各模式的要求请参阅规划控制平面端点。

Web UI 支持（Fleet Essentials）

Fleet Essentials 是一个集群插件，可为集群管理提供共享的 Web UI 框架。基础设施提供程序可以通过 UI 扩展锚点扩展该框架，以添加特定于平台的页面和工作流。

安装后，Fleet Essentials 可提供以下功能：

- 统一集群管理：通过单一界面管理所有平台上的集群
- 基础设施资源管理：配置云凭据、IP 池和机器模板
- 集群生命周期操作：通过引导式工作流创建、查看、升级和销毁集群
- 节点池管理：以声明式方式管理节点池，并支持扩缩和升级

实现这些 Fleet Essentials 扩展锚点的提供程序可以添加自己的集群创建工作流和管理页面：

- 裸金属提供程序：在物理服务器上提供仅基于 YAML 的集群生命周期管理。裸金属集群没有 Fleet Essentials UI；所有工作流均通过针对 `BaremetalCluster`、`BaremetalMachineTemplate`、`MachineInventoryPool` 和上游 CAPI 资源的 `kubectl apply` 驱动
- **DCS** 提供程序：在 Huawei DCS 上添加 DCS 专用的 UI 页面和工作流。用于池管理持久磁盘的 `DCSIpHostnamePool.spec.pool[].persistentDisk` 配置仅支持 YAML
- **HCS** 提供程序：安装 Fleet Essentials `1.0.2` 或更高版本后，支持在 Huawei Cloud Stack 上通过 Web UI 管理集群；同时支持基于 YAML 的集群生命周期管理，包括通过 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 管理池中的持久磁盘
- **vSphere** 提供程序：在 VMware vSphere 上提供基于 YAML 的集群生命周期管理。vSphere 集群的创建和管理没有 Fleet Essentials UI 扩展

在 global 集群上安装提供程序

安装 `global` 集群后，以及在 Bare Metal、Huawei DCS、Huawei Cloud Stack 或 VMware vSphere 上创建业务集群之前，请将该 IaaS 平台的基础设施提供程序与集群引导提供程序一起上传并安装到 `global` 集群上。无论 `global` 集群本身运行在哪个 IaaS 平台上，此要求均适用。

要安装提供程序，请从 Customer Portal 下载其软件包，提供程序会列在 **Common ACP Extensions** 下；使用 `violet push` 上传软件包，并将其作为集群插件安装。请参阅安装基础设施提供程序。

在 Alauda OS `global` 集群上升级 Kubernetes 时，也必须先将该 IaaS 平台的提供程序上传并安装到 `global` 集群上。

集群创建工作流

按以下顺序在 Alauda OS 上创建业务集群。准备按相同顺序介绍了前三个步骤。

1. 安装提供程序：在 `global` 集群上安装适用于 IaaS 平台的提供程序和集群引导提供程序。
2. 基础设施资源：配置提供程序所需的基础设施资源，例如云凭据、IP 池和机器模板。Bare Metal 没有单独的基础设施资源页面；其主机准备工作包含在提供程序安装和集群创建页面中。
3. 规划集群：阅读准备，了解适用于所有提供程序的决策，例如控制平面端点模式和集群的镜像仓库。此外，在创建集群之前还需检查以下内容：
   - 节点标签和污点：如果计划使用专用节点池（监控、日志记录、入口），请在创建节点池之前，阅读基础设施节点中关于如何在节点池上声明角色标签和污点的内容。
   - **CPU** 基线：对于使用 ACP 提供的 Alauda OS 镜像的 x86_64 集群节点，其底层 CPU 必须支持 `x86-64-v2` ISA 基线。详情请参阅OS 支持矩阵。
4. 创建集群：创建集群，包括其首批工作节点。

每个提供程序都有介绍这些步骤的专属页面：

提供程序	概述	安装提供程序	基础设施资源	创建集群
Bare Metal	Bare Metal 提供程序	安装 Bare Metal 提供程序	—	在 Bare Metal 上创建集群

提供程序	概述	安装提供程序	基础设施资源	创建集群
Huawei DCS	Huawei DCS 提供程序	安装 Huawei DCS 提供程序	Huawei DCS 的基础设施资源	在 Huawei DCS 上创建集群
Huawei Cloud Stack	Huawei Cloud Stack 提供程序	安装 Huawei Cloud Stack 提供程序	Huawei Cloud Stack 的基础设施资源	在 Huawei Cloud Stack 上创建集群
VMware vSphere	VMware vSphere 提供程序	安装 VMware vSphere 提供程序	VMware vSphere 基础设施准备	在 VMware vSphere 上创建集群

如果集群创建或删除停滞，请参阅故障排除。

相关任务

场景	文档
在 Alauda OS 上安装 `global` 集群	安装 global 集群
为 `global` 集群规划灾难恢复	global 集群灾难恢复
升级 `global` 集群	升级 Distribution 版本，然后在 Alauda OS 上升级 Kubernetes
升级业务集群	升级 Distribution 版本，然后在 Alauda OS 上升级 Kubernetes
管理集群节点	管理 Alauda OS 集群上的节点
配置机器（OS 或内核级别）	机器配置
查找 Cluster API 和提供程序资源	Cluster API 参考 和 提供程序 API

准备

规划控制平面端点　　为业务集群选择镜像仓库

裸金属

概览　　安装 **Provider**　　创建集群

在多磁盘裸金属主机上选择固定系统磁盘

Huawei DCS

概览　　安装 **Provider**　　基础设施资

创建集群　　排查卡在 **Deleting** 状态的 **DCSMachine**

Huawei Cloud Stack

概览　　安装 **Provider**　　基础设施资

创建集群　　排查 **Huawei Cloud Stack** 业务集群故障

VMware vSphere

概览 安装 **Provider** 基础设施资源

创建集群

故障排除

排查卡在 **Provisioned** 状态的业务集群 排查卡在删除中的集群

准备

在 Alauda OS 上创建业务集群之前完成以下准备工作。此流程适用于所有 provider。有关支持的 provider 列表，请参阅支持的 Provider。

目录

安装基础设施 Provider

配置基础设施资源

规划控制平面端点

选择镜像仓库

安装基础设施 Provider

安装 `global` 集群后，以及在裸金属、Huawei DCS、Huawei Cloud Stack 或 VMware vSphere 上创建业务集群之前，将相应 IaaS 平台的基础设施 provider 上传并安装到 `global` 集群。无论 `global` 集群自身运行在哪个 IaaS 平台上，此要求均适用。在 DR 对中，请在两个 `global` 集群上都执行此操作。

除基础设施 provider 外，每个 provider 还需要以下组件：

- **Alauda Container Platform Cluster Bootstrap Provider**：处理 Kubernetes 引导和控制平面管理

按以下步骤安装这两个组件：

1. 下载：从 Customer Portal 下载软件包。对于 ACP 4.5，provider 位于 **Common ACP Extensions** 下。请参阅下载。
2. 上传：使用 `violet push` 将软件包上传到平台。请参阅上架软件包。
3. 安装：在 `global` 集群上将软件包安装为集群插件。请参阅集群插件。

有关每个 provider 的软件包和验证步骤，请参阅：

- 裸金属
- Huawei DCS
- Huawei Cloud Stack
- VMware vSphere

配置基础设施资源

创建集群之前，配置 provider 所需的基础设施资源：

- 云凭据：存储 IaaS 平台访问信息
- **IP** 池：定义集群节点的网络配置
- 机器模板：指定虚拟机规格
- 网络清单和负载均衡器方案：在 provider 有要求时，准备子网清单和 API 服务器访问地址

Provider	描述
Huawei DCS	Huawei Datacenter Virtualization Solution
Huawei Cloud Stack	Huawei Cloud Stack
VMware vSphere	vCenter 清单、凭据、网络、模板、端点和存储准备

裸金属没有单独的基础设施资源页面。其主机准备工作属于安装裸金属 Provider 和在裸金属上创建集群的一部分。

规划控制平面端点

每个集群都需要一个用于 Kubernetes API 的稳定端点，并且必须在创建集群之前选择端点模式。请参阅规划控制平面端点。

选择镜像仓库

默认情况下，业务集群使用 `global` 集群的镜像仓库。也可以将集群绑定到专用镜像仓库，但只能在创建集群时进行绑定。请参阅为业务集群选择镜像仓库。

规划控制平面端点

为业务集群选择镜像仓库

规划控制平面端点

每个集群都需要一个用于 Kubernetes API 的稳定端点。在创建集群之前选择端点模式，并在 kubeadm 初始化控制平面后保持地址稳定。

目录

按 Provider 划分的端点模式

External LoadBalancer 契约

验证外部 LoadBalancer

自建 VIP 契约

按 **Provider** 划分的端点模式

此版本中可用的端点模式取决于 Provider。

Provider	端点模式
Huawei DCS	External LoadBalancer，或在与 ACP `v4.4+` 配对时使用 `type: internal` 自建 VIP。单个控制平面节点只能将自身 IP 用于开发或 PoC。
VMware vSphere	External LoadBalancer，或在与 ACP `v4.4+` 配对时使用 `internal` 自建 VIP，该模式会在控制平面节点上部署 Alive（keepalived 和 IPVS）。

Provider	端点模式
Bare Metal	`Internal` 自建 VIP 和 `External` LoadBalancer。
Huawei Cloud Stack	Provider 管理的 HCS ELB。使用 在 Huawei Cloud Stack 上创建集群 中的端点字段和 ELB 工作流。

对于 DCS，`type: internal` 需要 ACP `v4.4+`。请在创建集群之前进行规划。使用外部 LoadBalancer 的现有集群将继续使用它，除非你已有经过测试的迁移计划。

对于 vSphere，`type: internal` 需要 vSphere Provider `v1.0.17+` 和 ACP `v4.4+`，并且这是创建时的选择：只有以这种方式创建的集群才能使用它，而 Provider 从早期版本升级的集群会继续使用其外部 LoadBalancer。此外，还要求 `alive` 集群插件已存在，并且可以部署到 `global` 集群，因为 Provider 通过 `ModuleInfo` 安装 Alive，而不是自行提供它。在规划模式之前，请验证所有这些条件；请参阅控制平面端点模式。

External LoadBalancer 契约

在应用 Cluster API 资源之前，配置并验证负载均衡器。

要求	契约
侦听器	使用带 TLS 透传的第 4 层 TCP 转发。TLS 由 Kubernetes API server 而非负载均衡器终止。
前端	使用一个稳定的 VIP（IP 地址）和 TCP 端口 `6443`。在创建集群之前预留该地址。
后端	在 TCP 端口 `6443` 上添加每个控制平面节点的 IP。不要添加工作节点。
健康检查	优先对 `https://<control-plane-node-ip>:6443/healthz` 执行 HTTPS 检查，并要求返回 HTTP `200`。如果设备仅支持 TCP 检查，请记录该检查只能证明端口可达，不能证明 API server 健康。
后端所有权	负载均衡器所有者必须在控制平面节点添加、替换或移除时更新后端成员。DCS 和 vSphere Provider 不管理外部负载均衡器后端。

要求	契约
引导可达性	在 Cluster API 创建控制平面期间，引导节点或当前管理集群必须能够访问前端端点。
节点可达性	所有控制平面节点都必须能够访问该端点。业务集群端点还必须可从 `global` 集群访问。
名称和证书	不要在集群端点前使用 DNS 名称（FQDN）——不支持通过域名访问集群。API server 证书由集群自身的 CA 签发，因此无法在端点上提供客户签发的证书；而没有该证书的域名也无法发挥其通常的作用。

有关 ACP 端口要求，请参阅以下页面：

- 对于 `global` 集群，请参阅负载均衡器转发规则。
- 对于业务集群，请参阅连接 global 集群和业务集群。

> **WARNING**
>
> 集群初始化后，不要将更改控制平面端点作为常规负载均衡器编辑操作。该端点已嵌入 API server 证书、kubeconfig 和管理集群连接中。更改端点需要经过规划的证书和连接迁移。

验证外部 **LoadBalancer**

在创建集群之前，确认侦听器已存在，并且引导环境或管理环境可以打开前端端口：

```
nc -vz <control-plane-endpoint> 6443
```

至少有一个 API server 运行后，从代表负载均衡器的网络位置验证首选健康检查：

```
curl -kfsS https://<control-plane-node-ip>:6443/healthz
```

预期响应为 `ok`，HTTP 状态为 `200`。

自建 VIP 契约

对于 Huawei DCS，仅将此契约用于 ACP `v4.4+`。在 VMware vSphere 上，它适用于 provider `v1.0.17` 及更高版本。对于裸金属，请遵循特定于 provider 的指南。

- 在与将承载 VIP 的控制平面接口处于同一 Layer-2 网络的网段中，预留一个未使用的 IPv4 地址。
- 使用在该 Layer-2 域中唯一的 VRID。
- 允许控制平面节点与周围网络之间的 VRRP 流量和免费 ARP 更新。
- 至少使用三个控制平面节点以实现高可用性。
- 确保节点镜像和受支持的 `alive` 软件包提供所需的 VIP 运行时和 IPVS 配置。
- 对于自建 VIP，请使用 IP 地址，而不是域名。
- 在控制平面节点被替换时，让 provider 和 Alive 协调 VIP 及后端成员关系。不要为同一 VIP 维护第二个手动后端列表。

在 VMware vSphere 上，还需遵循以下规则：

- VIP 必须为 IPv4。不接受 IPv6 和双栈端点。
- `vrid` 必须介于 `1` 和 `255` 之间。该范围由 provider 的准入 webhook 强制执行，而不是由 CRD schema 强制执行，因此超出范围的值会在应用时被拒绝，而不是由 `kubectl --dry-run=client` 拒绝。
- VIP 不得等于集群 `VSphereMachineConfigPool` 对象中声明的任何插槽地址（`network.primary.ip` 或 `network.additional[].ip`）。它还必须位于每个外部 IPAM 池范围之外；provider 无法读取这些池，也不会检测到该冲突。
- 集群创建时会写入 `spec.controlPlaneLoadBalancer`，或者永远不会写入。字段缺失已经表示使用外部 LoadBalancer，因此之后无法再填写；已写入的字段也无法编辑或清除。要纠正错误，必须重建集群。
- 对于多 NIC 节点，请显式设置 `interface`。自动检测会匹配节点的主 IP，且错误的选择之后无法通过修补进行更改。
- `VSphereCluster.status.conditions[SelfBuiltLoadBalancerReady]` 仅报告 VIP 链路。它不会控制 `VSphereCluster.status.ready`，因此基础设施集群可能在 VIP 仍处于收敛过程中时就报告已就绪。

有关 DCS 清单和验证，请参阅在 Huawei DCS 上创建集群。有关裸金属清单，请参阅在裸金属上创建集群。有关 vSphere，请参阅控制平面端点模式。

为业务集群选择镜像仓库

Alauda Container Platform 会在每个业务集群上安装并管理一组平台组件，包括 Kube-OVN 以及集群启动后交付的平台模块。这些组件的镜像来自镜像仓库，默认使用 `global` 集群所使用的同一仓库。

您也可以将业务集群绑定到专属仓库。常见原因是距离：当 `global` 集群与业务集群位于不同地域时，通过该连接拉取每个镜像会很慢且不稳定。将仓库部署在靠近业务集群的位置，可以消除这种依赖。

请在创建集群之前做出决定。绑定信息仅在创建集群时读取，现有集群无法迁移到其他仓库。

目录

选项

绑定涵盖的内容

继承 global 集群的仓库

将业务集群绑定到其专用仓库

开始之前

第 1 步 — 准备仓库

第 2 步 — 创建 RegistryCredential Secret

第 3 步 — 创建集群时引用凭证

第 4 步 — 验证绑定

操作现有绑定

故障排除

选项

选项	配置内容	适用场景
`global` 集群的仓库	无需配置。所有业务集群都会继承该仓库。	业务集群能够以可接受的吞吐量和可用性访问 `global` 集群所使用的仓库。
专用仓库	在 `cpaas-system` 中创建 RegistryCredential Secret，并通过 `cpaas.io/registry-reference` 标签从 `Cluster` 引用该 Secret。	业务集群位于不同地域或网络分段，需要从靠近它的仓库拉取镜像。

这两个选项适用于所有基础设施提供商。默认选项无需配置，因此也适用于通过 Web UI 创建的集群。专用仓库只能通过 YAML 进行配置：集群创建向导不提供可供选择的仓库凭证，因此当集群需要使用专用仓库时，请使用相应提供商集群创建指南中的 YAML 工作流。

绑定涵盖的内容

该绑定适用于 Alauda Container Platform 在业务集群上安装和管理的平台组件，例如 Kube-OVN 以及交付到该集群的平台模块。

以下内容不适用。每项内容都在其他位置进行配置，将集群指向专用仓库不会改变其中任何一项。

不涵盖的内容	其来源
kubeadm 管理的镜像：kube-apiserver、kube-controller-manager、kube-scheduler、etcd、	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageR` 每个提供商指南会独立设置该字段。在 Huawei DCS 和裸金属环境中，该字段为 `cloud.alauda.io/alauda`，与操作系统镜像中预加载的镜像一致。如果这些镜像必 仓库，请更改该字段，并确保引用仍与节点本地的镜像匹配。

不涵盖的内容	其来源
CoreDNS 和 kube-proxy	
pause（沙箱）镜像	节点上的容器运行时配置。受支持的操作系统镜像会在 containerd 中设置该配置；它 Kubernetes 对象解析。
裸金属操作系统镜像	主机在 `elemental install` 期间以及每次重新置备时，从平台仓库拉取 `base-ima` `base-image-iso`，此时还不存在任何集群级绑定。
原生应用工作负载镜像	每个工作负载声明的镜像引用和拉取 Secret。绑定仅配置平台镜像源。

继承 **global** 集群的仓库

这是默认行为，无需配置。创建集群时不添加 `cpaas.io/registry-reference` 标签，该集群就会使用与 `global` 集群相同的仓库。

地址和凭证如何传递到业务集群，取决于平台仓库是否要求身份验证：

- 需要身份验证的仓库 — 平台将地址和凭证存储在 `global` 集群的 `cpaas-system` 命名空间中的 `public-registry-credential` Secret 内。每个未显式引用的业务集群都会从该 Secret 解析其仓库。尽管名称如此，该 Secret 保存的是平台自身使用的仓库，并不局限于面向互联网的仓库。
- 匿名仓库 — 不会填充凭证 Secret，平台组件会回退到安装时记录的平台级仓库地址。

在这两种情况下，operator 都不会针对每个集群进行配置。

如果业务集群节点无法访问 `global` 集群上 `Platform Built-in` 仓库的地址，请参阅为内置仓库添加外部地址。

将业务集群绑定到其专用仓库

开始之前

- 你可以在 `global` 集群的 `cpaas-system` 命名空间中创建 Secret。引用该 Secret 的集群必须在创建之前确保凭证 Secret 已存在。
- 专用仓库可从每个业务集群节点访问；如果希望从 `global` 集群验证，也必须能从该集群访问。
- 专用仓库已包含目标版本的 Alauda Container Platform Core，以及计划在该集群上安装的每个平台插件的软件包。请参阅准备仓库。

第 1 步 — 准备仓库

平台不会将镜像复制到专用仓库。请在创建集群之前自行填充仓库。

以下两点决定了仓库必须包含的内容：

- 始终需要目标版本的 **Alauda Container Platform Core**。 其中的镜像和软件包必须与即将创建的集群版本及 CPU 架构匹配。
- 除 **Core** 之外的所有内容取决于你计划在该集群上安装的平台插件。 只有你知道这组插件 — 平台不会推导这组插件，也不会在之后补齐缺失内容。请将这些插件的软件包与 Core 一起上传。

使用准备外部平台镜像仓库时采用的相同 Core Package 上传操作步骤，并将专用仓库作为目标地址。请参阅准备目标版本负载。

然后，从目标网络中的某个节点确认仓库能够响应：

```
export DEDICATED_REGISTRY="<registry-host>:<port>"
curl -sk "https://${DEDICATED_REGISTRY}/v2/" -o /dev/null -w '%{http_cod
e}\n'
```

这只能证明可达性和 TLS。请在你自己的仓库管理界面或 API 中确认已上传的仓库和标签。

`/v2/` 部分仅属于 Registry HTTP API 请求。请勿将其 — 或 scheme — 包含在 Secret 或任何集群字段中。

> **WARNING**
>
> 请在创建集群之前验证内容。绑定到不完整仓库的集群仍然可以启动 — kubeadm 会从其他仓库解析控制平面，因此节点可以访问 `Ready`，集群看起来也很健康。缺失内容只会在之后暴露出来，表现

为平台 Pod 卡在 `ImagePullBackOff`。这比预先失败的仓库检查难以诊断得多。

第 2 步 — 创建 RegistryCredential Secret

在 `global` 集群的 `cpaas-system` 命名空间中创建 Secret。集群上的引用仅携带名称，因此只会查询此命名空间；其他命名空间中同名的 Secret 会被忽略。

请使用编辑器编写凭证文件，而不是在 shell 提示符中输入值。Shell 会将整个命令（包括密码）记录到历史记录中；使用编辑器写入的文件不会留下这样的副本。

registry-credential.env

```
registry=<registry-host>:<port>
username=<username>
password=<password>
```

对于匿名仓库，只保留 `registry` 行，并省略另外两行。

限制文件权限，从文件创建 Secret，然后将其删除：

```
chmod 600 registry-credential.env

REGISTRY_CREDENTIAL_NAME="<registry-credential-name>"

kubectl create secret generic "${REGISTRY_CREDENTIAL_NAME}" \
  --namespace cpaas-system \
  --from-env-file=registry-credential.env

rm -f registry-credential.env
```

WARNING

请勿使用 `kubectl apply` 创建凭证 Secret。客户端 apply 会将其应用的清单（包括其中的每个值）存储在 Secret 自身的 `kubectl.kubernetes.io/last-applied-configuration` 注解中。`kubectl create` 不会写入此类注解。无论这些值是以 `stringData` 还是 base64 `data` 形式写入，这一点都适用；base64 是编码，而不是保护措施。

文件中的每一行都是 `key=value`。请勿为值添加引号 — `kubectl` 会按字面值处理引号字符，因此 `password="s3cret"` 会将包含引号的 `"s3cret"` 存储下来。值可以包含 `=` 和其他特殊字符；每行中只有第一个 `=` 用于分隔键和值。

该 Secret 创建时使用的类型为 `Opaque`，此处这样设置是正确的。平台通过标签而不是类型识别仓库凭证，因此请在创建任何引用该 Secret 的集群之前应用以下两个标签：

```
kubectl label secret "${REGISTRY_CREDENTIAL_NAME}" --namespace cpaas-syst
em --overwrite \
  capi.cpaas.io/provider=registry-credential \
  cpaas.io/registry-credential-format=direct
```

键	必需	备注
Namespace	是	必须为 `cpaas-system`。
`capi.cpaas.io/provider` 标签	是	必须为 `registry-credential`。
`cpaas.io/registry-credential-format` 标签	是，本操作步骤需要	对于你创建的凭证，将其设置为 `direct`。该标签对于平台本身是可选的，但省略该标签会选择旧版加密格式，该格式要求上传身份验证文件，而不是使用下面的键。
`registry`	是	仅限 `<host>[:<port>]`。不得包含 scheme 或路径。
`username`、`password`	一起设置	两者必须同时设置或同时省略。匿名仓库请同时省略。只设置其中一个会被拒绝。

WARNING

请勿跳过 `capi.cpaas.io/provider` 标签。由于创建路径会直接读取这三个键，因此即使 Secret 缺少该标签，仍然可以基于它创建集群。但平台的其余部分随后不会将该 Secret 识别为凭证：其值的后续更改永远不会传播到引用它的集群，并且在仍被使用时也不会受到删除保护。

第 3 步 — 创建集群时引用凭证

将 `cpaas.io/registry-reference` 标签添加到 `Cluster` 资源，并将 Secret 名称作为其值：

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  labels:
    cpaas.io/registry-reference: <registry-credential-name>
```

按照提供商集群创建指南中的说明应用其余集群清单。

执行此操作时，Secret 必须已经存在。平台会在注册新集群时解析该引用，并在此时从 `cpaas-system` 按名称读取 Secret。如果 Secret 不存在，注册将无法完成，集群也不会继续推进。这不是永久性失败 — 平台会持续重试，因此之后创建 Secret 即可解除集群阻塞 — 但在此期间不会发生任何进展，并且原因不会报告在 `Cluster` 资源本身上。请先创建 Secret。

> **WARNING**
>
> 该标签只会在首次创建集群时读取。在现有 `Cluster` 上添加、更改或删除该标签都不会影响该集群。要将现有集群迁移到另一个仓库，必须重新创建集群。

第 4 步 — 验证绑定

在 `global` 集群上运行以下检查。确认引用已传递到平台集群记录中：

```bash
export CLUSTER_NAME="<cluster-name>"
kubectl get cluster.platform.tkestack.io "${CLUSTER_NAME}" \
  -o jsonpath='{.metadata.labels.cpaas\.io/registry-reference}{"\n"}'
```

确认平台已从凭证中解析仓库地址。该注解由平台写入；请勿自行设置：

```
kubectl get cluster.platform.tkestack.io "${CLUSTER_NAME}" \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}{"\n"}'
```

该值必须是 Secret 中的地址。如果为空或仍显示其他仓库，请参阅故障排除。

最后，在业务集群上确认平台 Pod 从预期仓库拉取镜像：

```
kubectl --kubeconfig <workload-cluster-kubeconfig> get pods -A \
  -o jsonpath='{range .items[*]}{range .spec.containers[*]}{.image}{"\n"}
{end}{end}' \
  | sort -u
```

平台组件镜像使用专用仓库。kubeadm 管理的镜像和 pause 镜像不会使用该仓库 — 请参阅绑定涵盖的内容。某些镜像仍解析到其他仓库是预期行为；但每个仍解析到旧仓库的平台组件镜像则不是。

操作现有绑定

任务	是否支持	操作
轮换仓库密码或切换到其他账户	是	修改 Secret，然后删除补丁文件。平台会将更改传播到引用该 Secret 的每个集群，无需重启集群。
更正凭证中的仓库地址	是，但需谨慎	以相同方式修改。它会重新指向引用此 Secret 的每个集群，因此请确认所有这些集群都能访问新地址，并且该地址包含每个集群版本所需的内容。
将现有集群迁移到其他仓库	否	引用在创建时即固定。请使用预期的引用重新创建集群。
在多个集群之间共享一个凭证	是	任意数量的集群都可以引用同一个 Secret。一次更新会影响所有这些集群。不同的 Secret 之间互不影响。
删除凭证	仅限未被引用时	只要仍有集群引用该 Secret，平台就会拒绝删除，并在错误信息中列出这些集群。请先删除这些引用：删除这些集群，或使用其他凭证重新创建它们。

请使用 `kubectl patch` 而不是 `kubectl apply` 更新凭证，原因与使用 `kubectl create` 创建凭证相同：

registry-credential-patch.yaml

```yaml
stringData:
  username: <new-username>
  password: <new-password>
```

```
chmod 600 registry-credential-patch.yaml

kubectl patch secret "${REGISTRY_CREDENTIAL_NAME}" --namespace cpaas-syst
em \
  --type=merge --patch-file=registry-credential-patch.yaml

rm -f registry-credential-patch.yaml
```

补丁中的 `stringData` 会替换 `data` 中匹配的键。如果此处的值包含 YAML 元字符，请将其括起来 — 与创建时使用的 `key=value` 文件不同，此文件是 YAML。

故障排除

症状	可能原因	操作
平台集群记录中的 `cpaas.io/registry-address` 仍为空	Secret 不存在、位于 `cpaas-system` 之外，或缺少 `capi.cpaas.io/provider: registry-credential` 标签。	确认名称、命名空间和两个标签。
创建或更新时 Secret 被拒绝	`registry` 为空，或者 `username` 和 `password` 中只有一个被设置。	设置 `registry`，并同时设置凭证对，或同时省略两者。
Secret 因不支持的格式消息而被拒绝	`cpaas.io/registry-credential-format` 包含 `direct` 或 `encrypted` 之外的值，包括空值。	对于自行编写的凭证，请使用 `direct`；或者移除该标签以使用旧版加密格式。

症状	可能原因	操作
集群报告其控制平面已就绪，但平台 Pod 仍停留在 `ImagePullBackOff`	专用仓库不包含此版本所需的有效载荷，或节点无法向其进行身份验证。	从业务集群节点重复执行步骤 1，并确认 Secret 中的凭证。
删除凭证被拒绝	仍有一个或多个集群引用该凭证。	从错误信息中读取集群名称。在删除此凭证前，先删除这些集群，或使用其他凭证重新创建它们。
现有 `Cluster` 的新标签值未产生任何变化	引用只在创建集群时读取。	使用预期的引用重新创建集群。
已应用 `Cluster`，但平台集群记录始终未出现，且集群没有任何进展	引用的 Secret 不存在于 `cpaas-system` 中，因此平台无法完成集群注册。`Cluster` 资源本身不会报告此错误。	创建 Secret。平台会重试，因此 Secret 创建后集群即可继续运行。
`kubectl apply --dry-run=server` 接受了清单，但实际应用过程停滞	服务器端 dry run 会在平台解析仓库引用之前返回，因此无法检测缺失的 Secret。	不要依赖 dry run。先确认 Secret 存在：`kubectl -n cpaas-system get secret <registry-credential-name>`。
Secret 的值变更始终未传播到集群，或者即使集群仍在使用 Secret 也可以将其删除	Secret 缺少 `capi.cpaas.io/provider: registry-credential` 标签。	应用该标签。请参阅步骤 2。

裸金属

概览 安装 **Provider** 创建集群

在多磁盘裸金属主机上选择固定系统磁盘

裸金属 Provider

目录

概述

状态

主要功能

与基于 VM 的 Provider 的差异

概念和术语

对象层级

裸金属概念

`MachineRegistration`

`SeedImage`

`MachineInventory`

`MachineInventoryPool`

`BaremetalCluster`

`BaremetalMachineTemplate`

`BaremetalMachine`

镜像目录

主机生命周期计划

`alive`（控制平面 HA）

`MachineInventory.spec.observedNetwork`

API 组

支持的 Kubernetes 版本

要求

文档

概述

裸金属基础设施 Provider 可在物理服务器上创建和管理 Alauda OS 集群，无需虚拟化层。它由两个长期运行在 `global` 集群上的组件组成：

- **`elemental-operator`** — 注册物理主机、构建安装 ISO（`SeedImage`），并为每台主机维护长期存在的 `MachineInventory` 对象。`elemental-system-agent` 在每台主机上运行，并执行 `MachineInventory` 计划 Secret。
- **`cluster-api-provider-baremetal`** — Cluster API 基础设施 Provider。它将可用的 `MachineInventory` 对象分组到资源池中，将每个 `Machine` 绑定到一个 `MachineInventory`，并写入 `clean` / `reprovision` 计划，推动主机完成 Kubernetes 节点生命周期。

与 VM Provider（DCS、vSphere）不同，裸金属 Provider 不会创建或销毁机器。主机只需安装一次（通过 `elemental install` 将 Live ISO 安装到磁盘上的 OS），随后将自身注册为 `MachineInventory`，并在整个集群生命周期内保留在库存中。节点的“创建”和“删除”通过由计划驱动的 `elemental upgrade` 实现，随后执行重启并重新运行 cloud-init。

状态

该 Provider 目前采用纯 YAML 工作流。暂时还没有用于裸金属集群的 Fleet Essentials UI — 本页中的每个步骤都由 `kubectl apply` 驱动。

主要功能

- **`MachineInventoryPool`** 分配模型 — operator 预先声明哪些 `MachineInventory` 对象可用于支持 `KubeadmControlPlane` 或 `MachineDeployment`。创建 `Machine` 时，Provider 会从该资源池中选取一个 `Available` 库存；不会在声明范围之外配置资源。

- 由计划驱动的节点生命周期 — 节点接入使用 `reprovision` 计划（写入 cloud-init、`cloud-init clean`、`elemental upgrade`、重启、重新运行 cloud-init、`kubeadm init/join`）。节点移除使用 `clean` 计划（停止 kubelet、清除 CRI 工作负载、停止 containerd）。在扩缩容、升级或删除集群期间，Provider 绝不会删除 `MachineInventory`。
- **Cluster API** 原生对象树 — 使用上游的 `Cluster`、`KubeadmControlPlane`、`MachineDeployment`、`Machine`。Provider 仅拥有基础设施树（`BaremetalCluster`、`BaremetalMachine`、`BaremetalMachineTemplate`、`MachineInventoryPool`）。没有自定义控制平面 CRD。
- 由镜像目录驱动的 **Kubernetes** 版本 — Provider 维护一个集群作用域的 `elemental-image-catalog` ConfigMap，将 `Machine.spec.version` 映射到 `elemental upgrade` 镜像。升级时只需“修补版本，由控制器解析匹配的镜像，然后重新配置节点”。
- 两种控制平面端点模式 — `BaremetalCluster.spec.controlPlaneLoadBalancer.type: Internal` 会部署 Alive（keepalived + IPVS + `kube-lock` Lease 仲裁），并协调自建 VIP。`External` 会跳过 Alive，改用预先配置的第 4 层 TCP LoadBalancer，其后端在集群外部维护。请参阅规划控制平面端点。
- 保留网络身份 — `elemental-register` 会将启动 Live ISO 时观察到的网络信息报告为 `MachineInventory.spec.observedNetwork`。首次 `elemental install` 和之后每次 `reprovision` 计划都会将该快照作为 cloud-init `network-config v2` 重放，因此主机在整个生命周期内都能保留其 IP 地址、默认路由和 DNS。
- 由库存管理的数据磁盘 — operator 可在 `MachineInventory.spec.storage` 中声明最多 32 个 XFS 或 ext4 数据卷。Provider 会在分配前准备这些卷，在重新配置期间于 kubelet 启动前激活其挂载单元，在移除节点时停用这些单元，并保留其文件系统和数据，以便同一库存再次使用。请参阅管理裸金属主机上的数据磁盘。

与基于 VM 的 Provider 的差异

方面	**VM Provider**（**DCS**、**vSphere**）	裸金属 **Provider**
节点创建	从凭证作用域的平台克隆 VM 模板	通过 `elemental upgrade` 重新配置已安装的物理主机

方面	**VM Provider**（**DCS**、**vSphere**）	裸金属 **Provider**
节点删除	关闭并删除 VM	运行 `clean` 计划；主机保留在库存中并返回资源池
资源池模型	按从节点数配置的 IP 资源池 / 主机名资源池	`MachineInventoryPool` 列出具体的 `MachineInventory` 名称
节点上的数据	每次替换时都会重新创建系统盘和模板磁盘；声明的持久磁盘会保留（DCS）	每次重新配置时，`initramfs` 都会清除 Kubernetes 管理的系统状态。在长期存在的 `MachineInventory` 上显式声明的数据卷可以保留其文件系统和数据，但数据会留在对应库存中，不会自动跟随替换后的 `Machine`。
恢复时间	每次替换需数分钟	每个 `elemental upgrade` 需数分钟，另加重启时间
首次安装路径	由平台管理员一次性上传 VM 模板	由 `SeedImage` 构建 Live ISO，在主机上启动后将 `elemental install` 安装到磁盘
控制平面 LB	operator 提供外部 LB	由控制平面节点上的 `alive` 静态 Pod 管理内部 LB（通过 `type: External` 支持外部 LB）
UI 支持	YAML + Fleet Essentials UI（DCS）	仅支持 YAML

概念和术语

对象层级

层级	所拥有的资源	在工作流中的作用
`elemental.cattle.io`	`MachineRegistration`、`SeedImage`、`MachineInventory`	注册物理主机、构建可启动 ISO 存在的主机身份。

层级	所拥有的资源	在工作流中的作用
主机智能体	每台主机上的 `elemental-system-agent`	轮询 `MachineInventory.status.p` 引用的默认计划 secret，并将计 `MachineInventory.status.p`
`infrastructure.cluster.x-k8s.io`	`BaremetalCluster`、`BaremetalMachineTemplate`、`BaremetalMachine`、`MachineInventoryPool`	实现 Cluster API 基础设施 Prov 中选择库存，并写入 `clean` / 计划。

所有权边界经过有意设计：`elemental-operator` 负责主机注册和 `MachineInventory` 生命周期，而 `cluster-api-provider-baremetal` 负责 Cluster API 基础设施树。Provider 按名称引用 `MachineInventory` 对象，并通过库存现有的计划 secret 写入计划；它不会创建或删除库存。

裸金属概念

`MachineRegistration`

声明注册端点和首次安装时使用的 cloud-config，`elemental-register` 会在 Live ISO 上使用它。Operator 设置 `machineName`、`machineInventoryLabels` 和 `machineInventoryAnnotations`（使用 `${SMBIOS/...}` 模板化），以及 `elemental.install` 和 `elemental.registration` 块。裸金属 Provider 会查询 `MachineRegistration`，但不会修改它——它属于 elemental 层。

`SeedImage`

触发 `elemental-operator` 构建可启动 ISO，其中内置了注册 URL 和 Operator 的 TLS 材料。`spec.baseImage` 必须引用与目标 Kubernetes 版本匹配的 OS 镜像 **ISO** 变体（仓库名称带有 `-iso` 后缀；标签/摘要与计划安装版本对应的 `elemental-image-catalog` 条目匹配）。每台物理主机只需启动一次 ISO；启动后，主机先运行 `elemental-register`，再运行 `elemental install`，并创建一个 `MachineInventory`。

`MachineInventory`

长期存在的主机身份对象。Provider 依赖其契约中的以下部分：

- `status.plan.secretRef` — 由 `elemental-operator` 所有的唯一默认计划 secret。
- `status.plan.state` — 用于驱动 `BaremetalMachine` 状态机的 `Applied` / `Failed` 转换。
- `status.conditions` — 主机侧就绪信号。
- `spec.observedNetwork` — 仅 fork 中存在的字段，由 `elemental-register` 从 Live ISO 的 NIC 填充；在安装期间以及每个 `reprovision` 计划期间重放。
- `spec.storage` — 可选字段，由 Operator 编写，用于声明归库存所有的数据卷。
- `status.observedStorage` — 由主机上的 `elemental-storage-observer.service` 报告的客观设备、文件系统、挂载和多路径事实。
- `status.storage` — Provider 所有的 Prepare / Activate / Deactivate / Release 状态、操作日志、条件和已应用卷标识。

Provider 从不删除 `MachineInventory`，从不使用 `MachineInventorySelector`，也不会运行单独的 `MachineInventoryLifecycleController`——该生命周期仍由 `elemental-operator` 负责。

`MachineInventoryPool`

由 Operator 编写的 `MachineInventory` 名称集合，规定给定的 `BaremetalMachineTemplate` 可从中选取哪些资源。资源池的作用域限定为单个 `clusterName`，每个 `MachineInventory` 同一时间最多属于一个活动资源池。资源池协调器汇总整个资源池的容量计数器，供文档各处使用：

- `available` — 可供分配
- `allocated` — 已绑定到活动的 `BaremetalMachine`
- `preparing` — 正在运行 `clean` 计划
- `reprovisioning` — 正在运行 `reprovision` 计划
- `unavailable` — Ready=False、计划失败、缺少计划 secret，或在集群中不存在

`BaremetalCluster`

Cluster API 基础设施集群资源。拥有 `controlPlaneLoadBalancer`（VIP 和 `vrid`，由 `alive` chart 使用）以及 `controlPlaneEndpoint`（仅设置 VIP 时从 `controlPlaneLoadBalancer` 回填）。协调器会延迟部署集群附加组件（`alive`、`kube-ovn`），直到业务集群控制平面可访问。

`BaremetalMachineTemplate`

由 `KubeadmControlPlane.spec.machineTemplate.infrastructureRef` 和 `MachineDeployment.spec.template.spec.infrastructureRef` 引用的 Cluster API 基础设施模板。模板仅包含 `machineInventoryPoolRef`（此机器组从哪个资源池选取）。Provider 会按照资源池声明中的顺序选择第一个 `Available` 库存。

模板中有意不包含 `version`、`role` 或 `upgradeImage`。角色来自所属的 Cluster API 资源；版本来自 `Machine.spec.version`；升级镜像则在重新置备时从全局镜像目录中解析。

`BaremetalMachine`

Cluster API 基础设施机器。协调单个 `Machine` 与单个 `MachineInventory`：

1. 从所属模板引用的资源池中选择一个 `Available` 库存。
2. 读取所属的 `Machine.spec.bootstrap.dataSecretName`，并从镜像目录中解析适用于 `Machine.spec.version` 的 elemental 升级镜像。
3. 规范化引导用户数据（主机名、kubelet `provider-id`、`criSocket`），并将 `reprovision` 计划写入 `MachineInventory` 计划 secret。
4. 监视 `MachineInventory.status.plan.state`，直到计划报告 `Applied`，然后设置 `BaremetalMachine.spec.providerID = baremetal:///<inventory-name>`。
5. 删除时写入 `clean` 计划；计划应用后清除所有者注解，将库存归还资源池。

阶段转换：`Pending → Allocated → Reprovisioning → Running`；删除：`Running → Preparing → Deleted`；失败：`* → Failed`。

镜像目录

集群范围的 `ConfigMap`（在 `cpaas-system` 中的默认名称为 `elemental-image-catalog`），用于将 `Machine.spec.version` 映射到一个 `elemental upgrade` 镜像。裸金属 Provider chart 会根据 `provider.imageCatalog.images`（每个仓库前都会加上

`global.registry.address`）和 `provider.imageCatalog.data`（用于摘要固定镜像等完全限定的覆盖项）渲染此 ConfigMap。默认情况下，这两个值均为空，因此在添加条目之前，ConfigMap 不包含任何条目；请参阅导入 OS 镜像并填充镜像目录。协调器会热加载 ConfigMap；缺少键会导致终止性的 `Failed` 状态，不会回退到默认镜像。

请向 Alauda 技术支持获取匹配的 `base-image` 和 `base-image-iso` 引用。镜像目录在重新置备时使用 `base-image`，而 `SeedImage.spec.baseImage` 在 Live ISO 安装时使用 `base-image-iso`。不要根据其中一个引用推导另一个，也不要替换为独立构建的镜像。

主机生命周期计划

Provider 通过库存的唯一默认计划 secret 串行执行所有主机操作。该 secret 的 `baremetal.alauda.io/plan.type` 注解用于标识当前生命周期计划：

- **`storage-prepare`** — 在库存可用且尚未分配时运行。它会预检查所有已声明的卷，然后采用现有的匹配文件系统，或初始化明确授权的空白设备。不会挂载业务路径。
- **`reprovision`** — 在节点接入时运行。它会嵌入托管卷激活逻辑，写入 NoCloud `user-data` / `meta-data`（且当 `MachineInventory.spec.observedNetwork` 非空时写入 `network-config` v2），写入清理标记，运行 `cloud-init clean --logs --seed`，运行 `elemental upgrade --reboot=false --system <image>`，并触发延迟重启。重启后，`initramfs` 会清除 `/var/lib/kubelet`、`/var/lib/containerd`、`/var/lib/etcd` 和 `/etc/kubernetes`；托管挂载单元会在 kubelet 启动前启动，cloud-init 会执行 `kubeadm init` / `kubeadm join`。
- **`clean`** — 在节点脱离时运行。它会停止 kubelet、停用托管挂载、清理 CRI 工作负载并停止 containerd。它明确不会擦除托管文件系统、运行 `kubeadm reset`、运行 `cloud-init clean`、运行 `elemental upgrade` 或重启。OS 清理会推迟到下一个 `reprovision` 计划执行。
- **`storage-release`** — 仅在库存可用且所有托管卷均处于非活动状态时运行。它会移除 `spec.storage` 中已删除卷的 Provider 所有权和激活元数据，但不会擦除其文件系统或数据。

Provider 不使用 `MachineInventorySelector` 或 `FleetBundle`。存储生命周期详情、安全门控和示例请参阅管理裸金属主机上的数据磁盘。

`alive`（控制平面 HA）

`alive` 由一组静态 Pod（keepalived + IPVS）和一个 `kube-lock` Lease 仲裁器组成，用于维护 `BaremetalCluster.spec.controlPlaneLoadBalancer` 中描述的控制平面 VIP。第一个控制平面 Node Ready 后，Provider 会在业务集群上将 `alive` 部署为 `AppRelease`，并在控制平面成员发生变化时重新渲染它。在首次 `kubeadm init` 期间，Provider 会在引导过程中预先添加一次性的 `ip addr add <control-plane-vip>/32 dev eth0` 命令，使第一个节点持有 VIP，直到 `alive` 接管。

VIP 必须位于控制平面节点的二层广播域中；`vrid` 在该域内必须唯一。

`MachineInventory.spec.observedNetwork`

仅 fork 中存在的字段，由 `elemental-register` 根据 Live ISO 的 NIC 状态填充，并通过 `MsgObservedNetworkConfig` 注册消息报告回去。它在以下两个位置使用：

- 首次 `elemental install` 期间，如果 `MachineInventory.spec.network` 为空，注册服务器会回退到将 `observedNetwork` 转换为 `nmconnections` `NetworkConfig`，使磁盘上的 OS 保留与 Live ISO 相同的地址。
- 每次 `reprovision` 期间，Provider 会将 `observedNetwork` 转换为 cloud-init `network-config` v2（netplan 子集），并将其作为第三个 NoCloud 种子文件写入。两种情况下，显式设置的 `spec.network` 优先。

API 组

所有裸金属基础设施资源都属于 `infrastructure.cluster.x-k8s.io/v1beta1`。Elemental 资源属于 `elemental.cattle.io/v1beta1`。

资源	描述	文档
`BaremetalCluster`	集群级基础设施（控制平面 VIP、端点、网络类型）	BaremetalCluster
`BaremetalMachine`	绑定到一个 `MachineInventory` 的单个基础设施 Machine	BaremetalMachine
`BaremetalMachineTemplate`	将池绑定到 `KubeadmControlPlane` 或	BaremetalMachineTemplate

资源	描述	文档
	`MachineDeployment` 的模板	
`MachineInventoryPool`	集群允许使用的 `MachineInventory` 名称集合	MachineInventoryPool

支持的 **Kubernetes** 版本

请使用产品支持矩阵了解每个 ACP 版本支持的 Kubernetes 版本。有关发行版批准的 Kubernetes 版本及匹配的组件版本，请参阅 OS Support Matrix。

裸金属 provider 通过 `elemental-image-catalog` 强制执行该支持决策：集群使用的每个 `Machine.spec.version` 都必须作为该 ConfigMap 中的键存在。目录配置和带外覆盖操作步骤记录在安装和升级 Kubernetes 中。

要求

- 物理主机（或用于实验室环境、支持 PXE 启动的 VM），并可通过 BIOS 或 UEFI 访问以挂载 `SeedImage` ISO。
- 每台主机都可访问的平台 registry（`elemental install` 和 `elemental upgrade` 均使用该 registry）。设置 `global.registry.address`；当 registry 使用自签名证书时，保留 `global.registry.tlsVerify=false`（chart 默认值）。
- 一个控制平面 VIP、一个空闲端口（通常为 `6443`），以及一个在控制平面二层广播域内唯一的 `vrid`。
- 每个角色（控制平面、worker）使用一个 `MachineInventoryPool`，其规模至少应达到目标副本数，并为升级预留余量。
- 具有真实硬件 TPM（`/dev/tpm0`）的主机可以使用 `emulate-tpm: false` 注册。任何没有硬件 TPM 的主机——包括虚拟机和出厂时未配备 TPM 模块的物理服务器——都必须设置 `MachineRegistration.spec.config.elemental.registration.emulate-tpm: true`；物理机与虚拟机之间的区别不适用。
- 托管数据磁盘还要求使用启用了 `elemental-storage-observer.service` 的支持 observer 的主机 OS 镜像。仅升级管理端 provider 和 `elemental-operator`，不会将此服

务添加到使用旧镜像安装的主机。

文档

有关如何使用裸金属 provider 的详细说明，请参阅：

- 安装
- 创建集群
- 管理节点
- 管理数据磁盘
- 升级 Kubernetes
- API 参考

安装 Bare Metal Provider

本文介绍如何在 `global` 集群上安装 Bare Metal 基础设施 Provider 和集群引导 Provider，以及如何将安装在 Bare Metal 上的 `global` 集群准备为管理集群。

目录

适用场景
先决条件
安装 Provider 集群插件
下载软件包
上架软件包
安装集群插件
验证安装
将 global 集群准备为管理集群
让 provider 指向实际存在的平台证书
在 global 集群上安装 Cluster API providers
将基础镜像上传到 global Registry
创建 elemental 镜像目录
业务注册使用平台域名
后续步骤

适用场景

在 `global` 集群安装完成后、创建第一个 Bare Metal 业务集群之前，完成安装 Provider 集群插件。无论 `global` 集群本身运行在哪种 IaaS 平台上，都必须完成此操作。在 DR 对中，请在两个 `global` 集群上都完成此操作，以便发生故障转移后，备用集群能够协调相同的业务集群。

如果 `global` 集群安装在 Bare Metal 上，还要完成将 global 集群准备为管理集群。所有安装在 Bare Metal 上的 `global` 集群都需要执行此操作，无论您是否计划创建 Bare Metal 业务集群：

- 在停用引导集群之前完成此操作。
- 完成交接后，重新配置 `global` 节点时，会从 `global` Registry 拉取 `baremetal-base-image`；本节会将其上传到该 Registry。
- 在创建 Bare Metal 业务集群之前完成此操作。

这项任务与安装 global 集群中的 Provider 操作步骤是分开的。那些操作步骤会将 Provider 软件包上传到临时引导集群，以创建 `global` 集群本身。

先决条件

安装 Provider 之前，请确保您具备：

- 访问 灵雀云容器平台 `global` 集群的权限。
- 可访问 Customer Portal 以下载集群插件。

安装 **Provider** 集群插件

1 下载软件包

裸金属工作流需要两个集群插件。请从 Customer Portal 下载这两个插件。对于 ACP 4.5，它们列在 **Common ACP Extensions** 下；请参阅下载。

1. **Alauda Container Platform Cluster Bootstrap Provider**

2. **Alauda Container Platform Infrastructure Provider for Bare Metal**（一个 umbrella chart，会同时提供 `cluster-api-provider-baremetal-manager` 和 `elemental-operator` 子 chart）

2 上架软件包

使用 `violet push` 将这两个软件包上架到平台。有关命令及其参数，请参阅上架软件包。

3 安装集群插件

在 `global` 集群上安装这两个集群插件。有关操作步骤，请参阅集群插件。

如果 `global` 集群安装在 Bare Metal 上，请在安装 Infrastructure Provider for Bare Metal 时设置以下值。请在安装前确定这些值：

- `elemental.tls.caCertSecretKey`：在 `dex.tls` 中找到的密钥；请参阅将 Provider 指向实际存在的平台证书。
- 在 DR 对中，双方的 `elemental.systemAgent.splitAuthEnabled: true`，以及 active 集群上的 `elemental.systemAgent.sharedAuthReadOnly: false` 和 standby 集群上的 `true`。每个 `global` 集群都保留自己的值；DR 同步不会复制 Provider 的 `ModuleInfo`。

按照通过 YAML 安装中的说明，使用 YAML 安装 Infrastructure Provider for Bare Metal，并将这些值放入其 `ModuleInfo` 的 `spec.valuesOverride` 中；Web Console 安装表单没有用于设置这些值的字段：

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: cluster-api-provider-baremetal
    cpaas.io/module-type: plugin
  name: global-temporary-name
spec:
  config: {}
  version: <provider-plugin-version>
  valuesOverride:
    acp/chart-cluster-api-provider-baremetal:
      elemental:
        tls:
          caCertSecretKey: <key-found-in-dex.tls>
        systemAgent:                  # DR pair only
          splitAuthEnabled: true
          sharedAuthReadOnly: false   # true on the standby
```

如果插件已安装，请将相同的 `valuesOverride` 添加到其现有的 `ModuleInfo` 中：使用 `kubectl get moduleinfo -l cpaas.io/cluster-name=global,cpaas.io/module-name=cluster-api-provider-baremetal` 找到它，然后运行 `kubectl edit moduleinfo <name>`。不要编辑 Provider 的 `AppRelease`：平台会根据 `ModuleInfo` 重新生成它，因此所做的更改会丢失。

umbrella chart 会在一个操作步骤中同时安装裸金属 Provider manager 和 `elemental-operator`。它还会创建 `elemental-image-catalog` ConfigMap，但其中没有任何条目，并且软件包不包含 Alauda OS 镜像。创建集群前，请将 Bare Metal OS 镜像导入平台 Registry，并将您要部署的每个 Kubernetes 版本添加到 catalog 中；请参阅已导入 OS 镜像并填充镜像目录。

4 验证安装

安装插件后，验证 controller 和 CRD 是否存在：

```
# Provider manager + elemental-operator should both be Running
kubectl -n cpaas-system get pods | grep -E 'baremetal|elemental'

# Bare-metal CRDs
kubectl get crd | grep -E '^(baremetalclusters|baremetalmachines|baremetalmachinetemplates|machineinventorypools)\.infrastructure\.cluster\.x-k8s\.io$'

# Elemental CRDs (shipped by the elemental subchart)
kubectl get crd | grep -E '^(machineinventories|machineregistrations|seedimages)\.elemental\.cattle\.io$'

# Image catalog ConfigMap
kubectl -n cpaas-system get configmap elemental-image-catalog
```

预期输出包括：

- `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`

安装完成后，`elemental-image-catalog` 中没有 `data` 条目。添加要部署的 Kubernetes 版本后，请确认这些版本：

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

集群使用的每个 `Machine.spec.version`（包括创建时和升级时）都必须作为此 ConfigMap 中的键存在。缺少键会使 `BaremetalMachine` 停留在 `Failed`，并显示 `Reason=ImageCatalogMiss`。

将 **global** 集群准备为管理集群

> **仅限裸金属**
>
> 本节适用于安装在裸金属上的 `global` 集群。

安装 global 集群 本身会创建 `global` 集群。 之后创建裸金属业务集群是另一项工作：`global` 集群随后会作为 Cluster API 管理集群， 并且需要引导准备步骤未放置在其中的 provider 组件和制品。对于 DR 对中的两个 `global` 集群，都要完成本节操作—— 发生故障转移后，备用集群会成为活动管理集群，并且必须能够协调相同的业务集群。

安装期间导入 Cluster API 资源只会创建 CRD，不会在 `global` 集群上安装 Bare Metal provider 或 Cluster Bootstrap Provider。

设置本节使用的值。每个 `global` 集群运行一次，并将 `GLOBAL_KUBECONFIG` 指向要准备的集群。`OS_IMAGE_TAG` 和 `K8S_VERSION` 是 `global` 安装中第 3 步和 第 1 步使用的值。

```
export GLOBAL_KUBECONFIG="<path-to-this-global-kubeconfig>"
# Tag of the Bare Metal Alauda OS images imported in Step 3 of the global
installation.
export OS_IMAGE_TAG="<os-image-tag>"
# K8S_VERSION from Step 1 of the global installation.
export K8S_VERSION="<target-kubernetes-version>"
```

从要准备的集群自身的 `ProductBase` 和 `cpaas-system/registry-admin` Secret 中读取 Registry 地址和密码：

```
GLOBAL_REGISTRY_ADDRESS="$(kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" \
  get productbase.product.alauda.io base -o jsonpath='{.spec.registry.add
ress}')"
GLOBAL_REGISTRY_PASSWORD="$(kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -
n cpaas-system \
  get secret registry-admin -o jsonpath='{.data.password}' | base64 -d)"
```

让 **provider** 指向实际存在的平台证书

Bare Metal provider 从 `elemental.tls.caCertSecretName` / `caCertSecretKey` 挂载 `platformUrl` 的 CA bundle，这些值默认分别为 `dex.tls` 和 `ca.crt`。 这些默认值假定存

在由 cert-manager 签发的 `dex.tls` 。在 `global` 集群上， `dex.tls` 改为由安装程序的 `console.cert` 流程提供，而 `thirdParty` 证书生成的 Secret 仅包含 `tls.crt` 和 `tls.key` 。使用默认键时，operator 永远不会启动：

```
MountVolume.SetUp failed for volume "elemental-ca-cert":
  references non-existent secret key: ca.crt
```

检查 Secret 实际包含哪些键，并按照安装集群插件中的说明，在 provider 的 `ModuleInfo` 中相应设置 `caCertSecretKey` 。对于自签名 平台证书，证书本身就是其 CA，因此 `tls.crt` 是正确的键：

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  get secret dex.tls -o jsonpath='{.data}' | jq -r 'keys | join(",")'
```

在 global 集群上安装 Cluster API providers

将用于引导集群的相同 Bare Metal 和 Cluster Bootstrap Provider 软件包上传到此 `global` 集群，然后按照安装 Provider 插件中的说明安装两个插件。对于 DR 对中的备用 `global` 集群，按照上架软件包中的说明将软件包上传到备用集群。

安装 Bare Metal provider 时，将 `caCertSecretKey` 设置为上一步找到的键。在 DR 对中，还要将 DR 字段设置为与该侧引导版本相同的值： 两侧的 `elemental.systemAgent.splitAuthEnabled: true` ，活动集群上的 `sharedAuthReadOnly: false` ，以及备用集群上的 `true` 。按照安装集群插件中的说明，仅在 provider 的 `ModuleInfo` 的 `spec.valuesOverride` 中设置这些值。 此 `global` 集群不会继承引导版本的值；不要复制其他引导值，例如 `handoffHook.*` 。

继续操作前，确认 manager 和 `elemental-operator` 正在此 `global` 集群上运行：

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system get deploy \
  cluster-api-provider-baremetal-manager elemental-operator
```

将基础镜像上传到 global Registry

业务 `SeedImage` 构建在 `global` 集群上运行，并从 `ProductBase.spec.registry.address` 中记录的 Registry 拉取基础 ISO。交接后，重新置备 `global` 节点时也会 从该 Registry 拉取 `baremetal-base-image`，因为安装程序会将 `global` 集群的 `cpaas.io/registry-address` 注解重写为该 Registry。在第 3 步中导入的镜像仅存在于 引导 Registry 中，因此在将这两个镜像推送到 `global` Registry 之前，ISO 构建会因 `MANIFEST_UNKNOWN` 而失败。在引导节点上，从 `bare-metal-os` containerd 命名空间推送这些镜像。`--skip-verify` 需要 `--local`：

```
for repo in baremetal-base-image baremetal-base-image-iso; do
  ctr -n bare-metal-os images push --local --skip-verify \
    -u "admin:${GLOBAL_REGISTRY_PASSWORD}" \
    "${GLOBAL_REGISTRY_ADDRESS}/tkestack/${repo}:${OS_IMAGE_TAG}" "${rep
o}:${OS_IMAGE_TAG}"
  curl -sk -u "admin:${GLOBAL_REGISTRY_PASSWORD}" \
    "https://${GLOBAL_REGISTRY_ADDRESS}/v2/tkestack/${repo}/tags/list"
done
```

每个响应都必须列出 `${OS_IMAGE_TAG}`。完成后，使用 `ctr -n bare-metal-os images rm "baremetal-base-image:${OS_IMAGE_TAG}" "baremetal-base-image-iso:${OS_IMAGE_TAG}"` 删除本地副本。 如果引导节点上已不再有这些镜像，则按照已导入 OS 镜像并填充镜像目录中的说明，从 OS 存档导入。

创建 elemental 镜像目录

provider 通过由 `--image-catalog-namespace` 和 `--image-catalog-name` 定位的 ConfigMap，将 Kubernetes 版本映射到 elemental 升级镜像；这两个值默认分别为 `cpaas-system` 和 `elemental-image-catalog`。provider chart 创建此 ConfigMap 时不包含任何版本条目， 而且 `global` 集群不会继承你在引导版本中设置的条目。在添加条目之前，业务集群的首次 `BaremetalMachine` 会立即失败：

```
ImageResolved=False  ImageCatalogMiss:
  no elemental upgrade image mapped for Kubernetes version "<version>"
```

为此 `global` 集群管理的每个 Kubernetes 版本添加一个条目，包括其自身节点使用的 `${K8S_VERSION}`。请修改现有 ConfigMap，而不是替换它，以保留之前添加的条目：

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  patch configmap elemental-image-catalog --type merge \
  -p "{\"data\":{\"${K8S_VERSION}\":\"${GLOBAL_REGISTRY_ADDRESS}/tkestack/baremetal-base-image:${OS_IMAGE_TAG}\"}}"

kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  get configmap elemental-image-catalog -o jsonpath='{.data}'
```

ImageCatalogMiss 不会自行恢复

遇到 `ImageCatalogMiss` 的 `BaremetalMachine` 在修复目录后仍会保持 `Failed`。 重新协调对象和重启 provider 都不会改变其状态；协调器会刻意将缺少映射视为终止状态，而不是回退到默认镜像。 删除 `Machine`，使其所有者 `KubeadmControlPlane` 或 `MachineSet` 重新创建它。 在应用第一个业务集群清单之前填充目录，即可完全避免此问题。

业务注册使用平台域名

业务 `MachineRegistration` 不得带有 `baremetal.cluster.io/system-agent-auth-scope: global` 注解。其注册 URL 会解析为 `https://<platform-domain>/...`，生成的 `MachineInventory` 会带有 `system-agent-auth-scope: shared` 注解。该共享身份与共享 ServiceAccount 签名密钥共同使业务主机能够在故障转移后继续通过平台域名报告状态。

后续步骤

安装 provider 后，您可以继续执行以下操作：

- 创建集群
- 查看 Provider 概览，了解资源池模型和计划生命周期。

在裸金属上创建集群

本文介绍如何使用裸金属 provider 在物理服务器上创建 Kubernetes 集群。整个工作流仅使用 YAML——目前没有适用于裸金属集群的 Fleet Essentials UI。

目录

先决条件

1. 安装所需插件
2. 导入 OS 镜像并填充镜像目录
3. 网络连接
4. 主机时间同步
5. TPM 决策
6. 业务集群镜像 Registry
7. 主机磁盘和启动准备
8. 节点标签和污点

集群创建工作流

解析占位符值

步骤 1：构建 SeedImage 并注册主机

首次安装前调整 COS_STATE 大小

可选：准备托管数据磁盘

步骤 2：创建 `MachineInventoryPool` 资源

步骤 3：创建控制平面集群资源

步骤 4：部署工作节点

确认 worker pool 容量

配置 worker `BaremetalMachineTemplate`
配置 bootstrap template
配置 `MachineDeployment`
集群验证
使用 kubectl
验证控制平面端点
预期结果
常见故障模式
后续步骤
附录
完整的 `KubeadmControlPlane` 配置

先决条件

创建集群之前，请确保满足以下所有先决条件。

1. 安装所需插件

在裸金属上创建第一个业务集群之前，请在 `global` 集群上上传并安装以下插件。无论 `global` 集群本身运行在哪种 IaaS 平台上，都适用：

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Bare Metal**（umbrella chart，会同时安装裸金属 manager 和 `elemental-operator`）

操作步骤请参阅安装 Provider。如果 `global` 集群安装在裸金属上，还需完成将 global 集群准备为管理集群。

2. 导入 OS 镜像并填充镜像目录

创建任何集群资源之前，请将两种裸金属 OS 镜像都导入目标平台 registry，然后将目标 Kubernetes 版本添加到 `elemental-image-catalog`。这两个步骤都不会自动完成：provider

软件包不包含 OS 镜像，provider chart 创建的镜像目录也没有任何条目。

镜像	用途	说明
OS 镜像（`base-image`），仓库 `baremetal-base-image`	`elemental-image-catalog`	使用 Kubernetes 专用根文件系统重新配置已注册的主机。
OS ISO 镜像（`base-image-iso`），仓库 `baremetal-base-image-iso`	`SeedImage.spec.baseImage`	构建用于初始主机安装和注册的 live ISO。

只导入一种镜像是不够的。添加镜像目录条目或 `SeedImage` 引用，并不会导入对应的镜像。

导入镜像。 适用于你的 ACP 版本的 Bare Metal OS 下载文件是一个归档，其中包含一个 OCI 镜像布局，两个镜像使用同一个 tag；请参阅各 Provider 的镜像格式。在安装了 containerd 且能够访问平台 registry 的主机上以 root 身份运行导入操作，例如 `global` 控制平面节点；Alauda OS 自带 `ctr`，但不包含 `skopeo`。`/var/lib/containerd` 需要至少与归档大小相等的可用空间。将归档导入专用的 containerd namespace，并列出创建的镜像名称：

```
ctr -n bare-metal-os images import --all-platforms --no-unpack <bare-metal-os-archive>
ctr -n bare-metal-os images ls -q
```

输出会列出 `baremetal-base-image:<os-image-tag>` 和 `baremetal-base-image-iso:<os-image-tag>`。推送这两个镜像，并确认 registry 中各自都列出了该 tag。`--skip-verify` 需要 `--local`：

```
for repo in baremetal-base-image baremetal-base-image-iso; do
  ctr -n bare-metal-os images push --local --skip-verify \
    -u "<registry-username>:<registry-password>" \
    "<registry-address>/tkestack/${repo}:<os-image-tag>" "${repo}:<os-ima
ge-tag>"
  curl -sk -u "<registry-username>:<registry-password>" \
    "https://<registry-address>/v2/tkestack/${repo}/tags/list"
done

ctr -n bare-metal-os images rm \
  "baremetal-base-image:<os-image-tag>" "baremetal-base-image-iso:<os-ima
ge-tag>"
```

最后一条命令会在两个镜像均成功推送后删除本地副本。对于 `global` 集群的 registry，`admin` 密码存储在 `cpaas-system/registry-admin` Secret 中。在灾难恢复对中，请将镜像导入两个 `global` 集群的 registry；请参阅将 global 集群准备为管理集群。

导入后，`<base-image>` 为 `<registry-address>/tkestack/baremetal-base-image:<os-image-tag>`，而 `<base-image-iso>` 为 `<registry-address>/tkestack/baremetal-base-image-iso:<os-image-tag>`。两个镜像必须来自同一个归档。不要将集群资源指向构建用 registry，不要混用来自不同归档的镜像，也不要替换为独立构建的镜像。

填充镜像目录。`cpaas-system` 中的 `elemental-image-catalog` ConfigMap 将 `Machine.spec.version` 映射到用于（重新）配置节点的 `elemental upgrade` 镜像。provider chart 创建的该 ConfigMap 不包含任何条目，因此在添加目标版本之前，无法配置任何节点。请 patch 该 ConfigMap，而不是替换它，以保留现有条目，然后检查结果：

```
kubectl -n cpaas-system patch configmap elemental-image-catalog --type me
rge \
  -p '{"data":{"<kubernetes-version>":"<base-image>"}}'

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

控制平面和 worker `MachineDeployment` 资源中使用的每个 `Machine.spec.version` 值，都必须作为键出现在此 ConfigMap 中，并保留开头的 `v`，且解析到已导入的 `base-image`。provider 会在重新配置时，通过将条目中的 registry 部分替换为平台 registry 地址来解析镜像。如果缺少目标版本，则不会写入 `reprovision` plan，`BaremetalMachine` 最终会进入

`Failed / Reason=ImageCatalogMiss`。之后再添加条目也不会清除此状态；请删除受影响的 `Machine`，使其 owner 重新创建该对象。

3. 网络连接

裸金属主机的注册、安装和加入集群都完全通过网络进行，因此在首台主机启动 `SeedImage` ISO 之前，必须开放下表中的所有网络路径。

Alauda OS 节点上的主机防火墙已预先配置并由平台维护——你无需管理它，而且下表也不是节点防火墙规则集。请参阅管理防火墙端口。你需要规划的是主机与其访问端点之间的网络：主机子网、平台和 registry 之间的站点防火墙、安全组及路由器 ACL。

阶段	源	目标	协议 / 端口
首次启动	每台从 `SeedImage` ISO 启动的主机	站点 DNS 服务器	UDP 和 TCP `53`
首次启动	每台主机	站点 NTP 服务器	UDP `123`
首次启动	每台从 `SeedImage` ISO 启动的主机	平台访问地址（`global.platformUrl`）	TCP `443`，或自定义的
首次启动	每台主机	平台 registry（`global.registry.address`）	TCP `11443`
配置和稳定运行	每台业务集群主机	平台访问地址，路径 `/kubernetes/global`	TCP `443`，或自定义的

阶段	源	目标	协议 / 端口
加入集群	新集群的每个节点	此集群的控制平面端点	TCP `6443`，或 `BaremetalCluster.spe` 中设置的端口
加入集群	新集群的每个节点	根据 `cpaas.io/registry-address` annotation 解析出的平台 registry	TCP `11443`
集群管理	`global` 集群节点	此集群的控制平面端点	TCP `6443`，或配置的
集群运行时	新集群的每个节点	同一集群的所有其他节点	控制平面节点之间的 etc overlay 隧道，默认 Gen `kubelet`；以及不按端

WARNING

仅开放 `443` 不足以满足业务主机的要求

`elemental-register` 和 `elemental-system-agent` 都通过平台访问地址访问平台，因此注册和轮询 plan 只需要开放平台 HTTPS 端口。但这并非全部要求。同一主机仍需通过 DNS 解析名称、从 `11443` 上的 registry 拉取操作系统和平台镜像、通过控制平面端点端口加入 API server，并与其他节点交换集群流量。仅允许 `443` 的防火墙审批，会导致主机成功注册，却在安装或加入集群时失败，而且届时更难查明原因。

表格范围。 这些路径适用于业务集群主机，也就是本文创建的主机。另外两种相邻场景在其他文档中说明，未包含在上表中：

- **Bootstrap**。 在 `global` 本身仍在安装期间，其主机会向临时 bootstrap 端点注册，适用的端口也不同——请参阅Bootstrap 节点网络。
- 移交后的 **`global`** 主机。 `global` 主机的系统智能体使用 Global 控制平面端点，而不是平台入口路径；在 DR 对中，它会直接使用 `6443` 上的本地控制平面 VIP。请参阅端点与身份。

在 bootstrap 阶段创建的 `SeedImage` 指向 bootstrap 端点，移交后不得重复使用；请在活动的 `global` 集群上创建新的 `MachineRegistration` 和 `SeedImage` 对象。在 `global` 集群上创建的业务 `SeedImage` 在移交后仍然有效，因为其注册 URL 指向平台域名，而不是 bootstrap 地址。

控制平面端点。

- 对于 `Internal` Self-built VIP，该 VIP 必须与控制平面节点 IP 位于同一个二层广播域中。`vrid` 在该域中必须唯一，网络必须允许 VRRP 和 gratuitous ARP 更新，并且节点镜像必须公开 IPVS，且允许 Alive 设置 `net.ipv4.conf.all.arp_accept=1` 和 `net.ipv4.vs.conntrack=1`。
- 对于 `External` LoadBalancer，请在创建集群之前配置 listener 和控制平面后端。请遵循规划控制平面端点。

首次启动时的寻址。

如果目标 VM 或物理主机启动进入 live ISO 后未从 DHCP 获取地址，请先在主机控制台手动配置网络，然后再等待注册。先检查 NetworkManager 连接名称，再应用站点专用的地址、网关和 DNS 值：

```
nmcli connection show
nmcli connection modify "Wired connection 1" \
  ipv4.method manual \
  ipv4.addresses 192.168.254.73/24 \
  ipv4.gateway 192.168.254.1 \
  ipv4.dns 192.168.16.19 \
  ipv4.ignore-auto-dns yes
nmcli connection up "Wired connection 1"
```

4. 主机时间同步

每台将成为集群节点的主机——无论是控制平面节点还是 worker 节点——在启动 `SeedImage` ISO 时都必须已经具有正确的时间。平台对跨节点的要求是：每个集群节点使用统一的时区，且节点之间的时间同步误差不超过 10 秒；请参阅节点检查。该要求描述的是集群节点，因此同样适用于此处，但此路径不会替你验证这一点：该页面中的快速配置脚本和节点检查面向的是平台通过 SSH 加入的非 Alauda OS 机器，不会在通过 `elemental-operator` 安装的 Alauda OS 主机上运行。

将时间同步作为步骤 1 之前的必要条件：

- 确保主机子网可以访问站点 NTP 服务器，并按照网络连接中的说明开放 UDP `123`。生产集群不应依赖公共 NTP。
- 从固件设置时间，而不是从操作系统设置。此时安装磁盘已按照主机磁盘和启动准备中的说明被擦除，因此主机上没有可用于运行命令的操作系统。请在 BIOS/UEFI 设置中或通过 BMC 校正时间，并将硬件时钟保持为 UTC，以便安装 Alauda OS 后统一应用时区。
- 相互比较各主机，而不仅仅是与某个参考时间比较。对于所有计划加入同一集群的主机，在其中任何主机启动 ISO 之前，时间差都必须保持在 10 秒要求以内。
- 节点达到 `Ready` 后，在每个节点上使用 `timedatectl` 确认结果——该命令会同时报告时区以及时钟是否已同步——并使用 `chronyc tracking` 查看时间源和当前偏移量。

整个集群 PKI 都继承运行 `kubeadm init` 的主机时钟。kubeadm 根据该主机的当前时间设置证书颁发机构的 `NotBefore`，每个叶证书都会继承该 `NotBefore`，并据此计算每个证书的 `NotAfter`。因此，时钟超前的主机会生成一套 PKI，其他节点和任何客户端都会将其视为尚未生效，直到自身时钟追上；而时钟落后的主机会生成一套比配置的有效期所暗示的时间更早过期的 PKI。硬件时钟从未同步过的服务器通常会相差数分钟或数小时，而不是几秒，因此后果是控制平面完全无法启动——而且问题会在控制平面启动阶段才暴露，此时距离引入错误时钟的注册步骤已经很久。

> **WARNING**
>
> **Day-2 chrony `MachineConfig` 无法替代此要求**
>
> 配置 Chrony 时间服务会将 `/etc/chrony.conf` 写入已经运行并加入集群的节点。这是使集群在持续运行期间使用站点 NTP 服务器的正确方式，但它与此处的要求不同：它无法校正仍在安装 Alauda OS 或运行 `kubeadm` 的主机时钟。将时间同步推迟到该步骤，会使安装和加入集群的时间窗口得不到保护。

5. TPM 决策

根据主机是否提供真实硬件 **TPM**（`/dev/tpm0`）设置 `MachineRegistration.spec.config.elemental.registration.emulate-tpm`，而不是根据主机是物理机还是虚拟机来设置：

- `emulate-tpm: false`（或省略该字段）——仅适用于具有可用硬件 TPM 的主机，以便 `elemental-register` 使用它执行 `auth: tpm`。
- `emulate-tpm: true` 和 `emulated-tpm-seed: -1`——适用于任何没有硬件 TPM 的主机。这既包括虚拟机，也包括出厂时未配备 TPM 模块的物理服务器（例如 Dell R620）。在此类主机上，`auth: tpm` 与 `emulate-tpm: false` 结合使用会使 `elemental-register` 的 TPM 证明失败，并且永远不会发送注册请求，因此 `elemental-operator` 会记录 0 次注册 POST。

6. 业务集群镜像 Registry

启动集群不需要 Registry 凭证。裸金属主机仅使用平台 Registry 地址安装操作系统并加入集群，因此无需配置凭证即可访问一个所有节点均为 `Ready` 的集群。

下一步——在新集群上安装平台组件——才需要凭证。这些组件会从 Registry 拉取镜像，而要求身份验证的 Registry 需要集群可使用的凭证。

默认情况下，集群使用 `global` 集群的 Registry；当该 Registry 需要身份验证时，其凭证由 `public-registry-credential` Secret 提供。如果希望从专用 Registry 拉取平台组件镜像，请遵循为业务集群选择镜像 Registry。

> **WARNING**
>
> 凭证可以稍后配置，但 Registry 的选择不能推迟。集群创建时即会绑定其 Registry，现有集群无法迁移到其他 Registry。请在应用以下 manifest 之前完成决定，即使你要到之后才安装平台组件。

上述两种选择都不会影响前文所述的操作系统镜像。主机会在 `elemental install` 期间以及每次重新置备时，从平台 Registry 拉取 `base-image` 和 `base-image-iso`，此时尚不存在任何集群级绑定。

7. 主机磁盘和启动准备

在启动 `SeedImage` ISO 之前，先对每台主机上的所有磁盘和虚拟磁盘（VD）进行分类。记录哪个设备是 OS 安装目标，以及哪些设备（如有）必须保留应用数据：

- 擦除选定的 OS 安装磁盘和其他已废弃的启动磁盘，确保不再保留可启动的旧操作系统。主机必须启动 ISO，而不是磁盘上旧的操作系统。将 `MachineRegistration.spec.config.elemental.install.device` 设置为准确的 OS 磁盘。
- 如果计划使用 `Adopt` 策略管理数据磁盘，不要擦除该磁盘。请独立备份它，记录其稳定 ID 和文件系统 UUID，并确保不会将其选为安装设备。
- 用于 `InitializeIfBlank` 的磁盘必须可丢弃且确实为空。格式化仍需在主机注册后单独获得初始化批准；注册 manifest 不会授权执行该操作。
- 从所有磁盘中移除残留的 Elemental 分区标签——`COS_STATE`、`COS_PERSISTENT`、`COS_OEM`、`COS_RECOVERY`，而不仅是目标安装磁盘。Elemental 会根据标签（`blkid -L COS_STATE`）解析分区；第二块磁盘上的残留标签（例如来自之前的安装或遗留的 multipath 成员）会导致它解析到错误的设备，从而使重新置备 snapshotter 失败。
- 对于同一磁盘可能通过多个路径显示的主机（multipath），请确保这些路径中没有任何路径暴露仍带有 `COS_*` 标签的磁盘；任一路径上的残留标签都可能先于目标安装磁盘被解析。请清理磁盘，而不是禁用 multipath——该镜像会保持 multipath 启用，以支持网络连接存储启动。
- 对于具有多个磁盘的主机，不要将 `install.device` 留空，也不要将 `/dev/sda` 用作持久身份标识。请遵循在多磁盘裸金属主机上选择固定系统磁盘，以便重复使用同一个 ISO，同时通过 WWN 选择每台主机的系统磁盘。
- 设置启动顺序，使主机优先启动虚拟 CD / ISO。

只能使用已注册清单报告的稳定存储身份。`/dev/sdX`、`/dev/vdX`、`/dev/nvmeXnY` 和 `/dev/mapper/mpathX` 等 Linux 路径属于运行时路径，不得放入 `MachineInventory.spec.storage`。

8. 节点标签和污点

如果要为池中的节点设置专属角色标签和污点——例如为监控、日志记录或 ingress 预留专用池——请按照基础设施节点中的说明，在池的资源上声明这些配置。请在创建池之前完成规划。

集群创建工作流

使用 YAML 时，先导入配套的 OS 镜像，然后执行五个必需的资源步骤；在将节点加入池之前，还可选择执行存储准备步骤。每个 Kubernetes 资源都必须应用到 `cpaas-system` 命名空间中。

步骤	应用或执行	结果
0	将 `base-image` 和 `base-image-iso` 导入目标平台注册表，并将目标 Kubernetes 版本添加到 `elemental-image-catalog` 中——请参阅已导入 OS 镜像并填充镜像目录	两个 OS 镜像均可用，并且在开始构建 `SeedImage` 或重新置备主机之前，目录已解析每个 `Machine.spec.version`。
1	`MachineRegistration` 和 `SeedImage`	`elemental-operator` 会构建可启动 ISO。
2	在每台物理主机上启动 ISO	主机运行 `elemental install`，并注册为 `MachineInventory`。
2a（可选）	声明并准备 `MachineInventory.spec.storage`	清单未分配时，会验证并准备显式数据卷。
3	`MachineInventoryPool` 资源，每个角色一个	在 CAPI 创建 Machine 之前，声明允许使用的控制平面和工作节点清单。
4	`BaremetalCluster`、控制平面 `BaremetalMachineTemplate`、`KubeadmControlPlane` 和 `Cluster`	Cluster API 使用控制平面池启动控制平面推出。
5	工作节点 `KubeadmConfigTemplate`、工作节点 `BaremetalMachineTemplate` 和 `MachineDeployment`	Cluster API 使用工作节点池启动工作节点推出。

WARNING

重要的命名空间要求

所有裸金属资源都必须应用到 `cpaas-system` 命名空间中。provider 和 `elemental-operator` 只会协调该命名空间中的对象。

WARNING

业务集群命名

业务 `cluster-name` 不得命名为 `global`。该名称保留给 `global` 集群；重复使用会导致业务集群的资源与 `cpaas-system` 中 `global` 集群的资源发生冲突。按照惯例，CAPI `Cluster` 和 `BaremetalCluster` 应保持名称恰好为 `<cluster-name>`，而依赖资源（`KubeadmControlPlane`、`KubeadmConfigTemplate`、`MachineDeployment`、machine 模板、池、注册资源）则应以 `<cluster-name>-` 为前缀。

解析占位符值

以下示例清单使用 `<placeholder>` 语法表示特定环境的值：

占位符	真实来源	获取方式
`<cluster-name>`	operator 提供的业务集群名称。不得为 `global`。	一次确定，并在 `Cluster`、`Baremeta` 板和注册资源中保持一致。
`<kubernetes-version>`	导入的 OS 镜像的 Kubernetes 版本，作为 `elemental-image-catalog` ConfigMap 键添加（带前导 `v`，例如 `v1.33.7-2`）。	从 OS Support Matrix 获取值；使用 `kul system get cm elemental-image-cata` 键。
`<bare-metal-os-archive>`	目标 ACP 版本的 Bare Metal OS 归档文件。	OS Support Matrix 中列出的下载页面。
`<os-image-tag>`	归档文件中两个镜像共用的标签。	导入后执行 `ctr -n bare-metal-os im`

占位符	真实来源	获取方式
`<registry-username>` / `<registry-password>`	可向 `<registry-address>` 推送的账户。	对于 `global` 集群的 registry，使用用户 `kubectl -n cpaas-system get secre o jsonpath='{.data.password}' | ba`
`<base-image>`	image catalog 用于主机重新置备的导入 OS 镜像。	在 OS Images Imported and Image Cata 后执行 `<registry-address>/tkestack image:<os-image-tag>`。
`<base-image-iso>`	与 `<base-image>` 配对的导入 OS ISO 镜像。	执行 `<registry-address>/tkestack/b image-iso:<os-image-tag>`，从与 `<b` 归档文件导入。
`<registry-address>`	导入两个 OS 镜像的目标平台 registry（与安装 chart 上的 `global.registry.address` 取值相同）。	执行 `kubectl get cluster global -r jsonpath='{.metadata.annotations.c address}'`（如果存在）。
`<control-plane-load-balancer-type>`	operator 的决定。对于 provider 管理的 Alive，使用 `Internal`；对于现有负载均衡器，使用 `External`。	查看 Plan the Control Plane Endpoint。
`<control-plane-vip>` / `<control-plane-port>`	operator 提供的稳定端点。对于 `Internal`，使用控制平面 Layer-2 域中未使用的 IPv4 VIP。对于 `External`，使用负载均衡器前端地址。	n/a
`<vrid>`	仅供 `Internal` 使用的 operator 提供的 VRID。它在控制平面 Layer-2 域中必须唯一。对于 `External`，移除该字段。	n/a
`<install-device>`	`elemental install` 的整盘目标。仅当直接内核路径的标识稳定且明确时，才适合使用该路径。对于复用同一个 ISO 的多磁盘裸金属主机，请使用 `/dev/elemental-install-target`。	从主机控制台通过 WWN 确认磁盘。请参 System Disk on Multi-Disk Bare-Metal H

占位符	真实来源	获取方式
`<dns-image-tag>` / `<etcd-image-tag>`	为 `<kubernetes-version>` 构建到裸金属基础镜像中的组件版本。	请参阅 OS Support Matrix 中的 `coredn`
`<kube-ovn-version>`	与所选 ACP 和 Kubernetes 版本匹配的 Kube-OVN chart 版本。	使用 OS Support Matrix 中的 **kube-ovn** Kube-OVN 组件版本，并通过已安装的插 `get moduleplugins.cluster.alauda.i jsonpath='{.spec.mainChart}{" "} {.status.latestVersion}'` 。
`<dns-server>`	当首次启动注册路径需要显式 resolver 时由 operator 提供。	优先使用 `MachineRegistration.confi` 更高版本的 `KubeadmControlPlane` bo （ `format: cloud-config` ）。不要从 `config` 写入 `/etc/resolv.conf` 。
`<ssh-authorized-keys>`	operator 提供的 OpenSSH 公钥——节点上的任何交互式调试都需要该密钥。	n/a
`<inventory-name>`	主机启动 ISO 并注册后由 `elemental-operator` 分配。注册前无法预测。	`kubectl -n cpaas-system get machineinventories.elemental.cattl`
`<pool-name>`	声明 inventory 的 `MachineInventoryPool` 的名称。	在下面的示例中使用 `<cluster-name>- pool` 或 `<cluster-name>-worker-poo`
`<control-plane-inventory-1/2/3>` / `<worker-inventory-1/2/3>`	为每个角色选定的确切 `MachineInventory` 名称。	使用 `kubectl -n cpaas-system get machineinventories.elemental.cattl`
`<control-plane-host-1/2/3>` / `<worker-host-1>`	在重新置备期间应用的可选 Kubernetes 节点主机名。	由 operator 提供；省略 `hostname` 以回 称。

占位符	真实来源	获取方式
`<pods-cidr>` / `<services-cidr>` / `<kube-ovn-join-cidr>`	由 operator 提供。不得与主机网络、`global` 集群的 CIDR 或同一 `global` 上的任何其他 CAPI 集群重叠。	n/a
`<workload-kubeconfig>`	业务集群的 Kubeconfig。控制平面初始化后，Cluster API 会将其发布为 `Secret`。	`kubectl -n cpaas-system get secre kubeconfig -o jsonpath='{.data.val`
`<machine-name>`	用于定向移除或恢复的现有 CAPI `Machine` 名称。	`kubectl -n cpaas-system get machi k8s.io`
`<new-replica-count>`	扩缩容后的目标 `MachineDeployment.spec.replicas` 值。	operator 提供的目标工作节点数量。
`<new-template-name>`	用于更改后的 `BaremetalMachineTemplate` 或 `KubeadmConfigTemplate` 的新模板名称。	由 operator 提供；使用新名称。
`<new-kubernetes-version>` / `<new-tag>` / `<new-image>`	目标升级版本以及匹配的裸金属基础镜像标签或已解析镜像。	使用版本支持矩阵，并确认最终键存在于 `catalog` 中。

步骤 1：构建 SeedImage 并注册主机

创建一个描述注册 URL 和首次安装 cloud-config 的 `MachineRegistration`，以及一个将 `elemental-operator` 指向匹配 ISO 基础镜像的 `SeedImage`。

将 `SeedImage.spec.baseImage` 设置为导入的 `base-image-iso` 的最终目标平台 registry 引用。它必须与 image catalog 用于后续重新置备的导入 `base-image` 配对：两个镜像都必须来自同一个 OS 归档文件，绝不能来自不同的归档文件。

01-machineregistration-seedimage.yaml

```yaml
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: <cluster-name>-registration
  namespace: cpaas-system
  labels:
    app.kubernetes.io/part-of: cluster-api-provider-baremetal
spec:
  # ${...} placeholders are expanded by elemental-register from SMBIOS data.
  # System UUID is preferred over Serial Number because it is more reliable
  # across virtualised test environments.
  machineName: "<cluster-name>-${System Information/UUID}"
  machineInventoryLabels:
    app.kubernetes.io/part-of: cluster-api-provider-baremetal
    pool.baremetal.alauda.io/eligible: "true"
    elemental.cattle.io/serial-number: "${System Information/Serial Number}"
  machineInventoryAnnotations:
    elemental.cattle.io/machine-uuid: "${System Information/UUID}"
  config:
    elemental:
      install:
        device: <install-device>
        eject-cd: true
        reboot: true
        snapshotter:
          type: btrfs
          maxSnaps: 4
      registration:
        # false only when the host has a real hardware TPM (/dev/tpm0). Set true
        # (with emulated-tpm-seed: -1) on any host without one, including physical
        # servers that ship without a TPM module.
        emulate-tpm: false
---
apiVersion: elemental.cattle.io/v1beta1
kind: SeedImage
metadata:
  name: <cluster-name>-registration-iso
  namespace: cpaas-system
```

```
    labels:
      app.kubernetes.io/part-of: cluster-api-provider-baremetal
  spec:
    type: iso
    baseImage: <base-image-iso>          # Use the imported target-platform
  registry reference.
    registrationRef:
      apiVersion: elemental.cattle.io/v1beta1
      kind: MachineRegistration
      name: <cluster-name>-registration
      namespace: cpaas-system
    targetPlatform: linux/amd64
    size: 20Gi
    cleanupAfterMinutes: 120
    cloud-config:
      stages:
        boot:
          - name: "Size COS_STATE for reprovisioning"
            files:
              - path: /etc/elemental/config.d/partitions.yaml
                permissions: 0644
                content: |
                  install:
                    partitions:
                      state:
                        size: 20480
```

首次安装前调整 **COS_STATE** 大小

`state` 分区用于存放运行中的系统以及所有保留的快照。Elemental 默认的 8192 MiB 不够用：一个镜像的快照会共享 extent，但升级到*不同*镜像时，必须同时容纳两个大小约为 3.5 GiB 且互不相关的系统，并会在中途因 `no space left on device` 失败，导致每次重试都在其上留下一个不完整的快照。每次集群升级都会经过 `elemental upgrade`，而分区大小在安装时固定，因此该问题只会在之后才暴露——此时集群已经投入生产，若不重新安装就无法再重新分区。

在安装时将其大小设置为至少 20480 MiB（20 GiB）。

> 位置和阶段都很重要

该片段必须位于 `stages.boot` 下的 `SeedImage.spec.cloud-config` 中。将其放入 `MachineRegistration.spec.config.cloud-config` 或任何其他阶段都会留下 8 GiB 分区，并且不会报告错误。

在向其余主机群推出之前，先在第一台已安装的主机上验证结果：

```
# Expect ~20G, not 8G
lsblk -o NAME,SIZE,LABEL | grep COS_STATE
df -h /
```

不要将 `/etc/resolv.conf` 添加到 `SeedImage.spec.cloud-config`。保持 ISO 通用；仅当首次启动注册路径需要时，才在 `MachineRegistration.spec.config.cloud-config` 中配置站点特定的 resolver。在经过测试的 Global 部署流程中，ISO 不包含 resolver 文件；节点 DNS 随后由 kubeadm bootstrap 数据进行配置。

`install.device`、`install.eject-cd` 和 `install.reboot` 是有意设置的。如果省略目标磁盘，`elemental install` 可能会选择非预期设备。在多磁盘物理主机上，请使用固定系统磁盘操作流程，而不是使用会变化的 `/dev/sdX` 名称。如果 `eject-cd` 或 `reboot` 为 false，主机可能会在首次安装后仍停留在 live 环境中，永远无法成为 Cluster API 可用的 inventory。

应用 manifest，并等待 SeedImage 构建完成：

```
kubectl apply -f 01-machineregistration-seedimage.yaml
kubectl -n cpaas-system wait \
  --for=condition=SeedImageReady=True \
  seedimage/<cluster-name>-registration-iso \
  --timeout=30m

SEEDIMAGE_REASON="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.conditions[?(@.type=="SeedImageReady")].reaso
n}')"
SEEDIMAGE_DOWNLOAD_URL="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.downloadURL}')"
SEEDIMAGE_CHECKSUM_URL="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.checksumURL}')"

if [ "${SEEDIMAGE_REASON}" != "SeedImageBuildSuccess" ] || \
   [ -z "${SEEDIMAGE_DOWNLOAD_URL}" ] || \
   [ -z "${SEEDIMAGE_CHECKSUM_URL}" ]; then
  echo "SeedImage did not produce a downloadable ISO and checksum" >&2
  exit 1
fi

printf '%s\n%s\n' "${SEEDIMAGE_DOWNLOAD_URL}" "${SEEDIMAGE_CHECKSUM_URL}"
```

不要将 `status.state` 用作成功判定条件；当前 controller 不会填充该字段。只有当 `SeedImageReady=True` 的 reason 为 `SeedImageBuildSuccess`，且 `status.downloadURL` 和 `status.checksumURL` 均非空时，才能继续操作。下载生命周期已过期的 SeedImage 仍可能报告 `SeedImageReady=True`，但其 reason 会发生变化，且其 URL 会被清除。

从此 ISO 启动每台目标主机。`elemental-register` 首先运行（创建 `MachineInventory` 并上传 `observedNetwork`），随后 `elemental install` 将 OS 写入磁盘。安装完成后，主机仍可用于执行 plan。如果 live ISO 环境没有 DHCP 地址，请按照网络连接中的说明，在主机控制台上手动配置 NetworkManager，然后再等待 `MachineInventory`。

确认注册状态：

```
kubectl -n cpaas-system get machineinventories.elemental.cattle.io
```

每个计划使用的 inventory 都必须：

- 显示 `Ready=True`。
- 具有非空的 `status.plan.secretRef.name`。
- 具有与主机预期 NIC 匹配的 `spec.observedNetwork`（仅当希望安装时 IP 在重新置备期间保持不变时才需要）。
- 当需要托管数据磁盘时，运行支持 observer 的 OS image，报告新的 `status.observedStorage`，并在分配到 pool 之前达到 `status.storage.phase=Prepared`。

记录确切的 `MachineInventory` 名称——下一步会按名称引用它们。

可选：准备托管数据磁盘

托管存储属于长期存在的 `MachineInventory`，而不属于 CAPI `Machine` 或 `BaremetalMachineTemplate`。请在 inventory 尚未分配、且尚未将其添加到生产 pool 之前进行配置。

按照管理裸金属主机上的数据磁盘中的说明：

1. 验证 `elemental-storage-observer.service` 并检查 `status.observedStorage`。
2. 使用稳定 ID 选择每个设备，并选择 `Adopt` 或 `InitializeIfBlank`。
3. 使用匹配的 `storagectl` binary 进行验证，并以原子方式修补声明；在需要时，同时修补其初始化批准。
4. 等待 `StoragePrepared=True/AllRequiredVolumesPrepared` 和 `status.storage.phase=Prepared`。

如果不应管理任何磁盘，请省略 `spec.storage`，或将其设置为 `volumes: []`。存储 controller 会将该 inventory 收敛到 `Unmanaged`，并设置 `StoragePrepared=True/NoManagedVolumes`，且不会执行任何磁盘操作。

步骤 2：创建 `MachineInventoryPool` 资源

每个角色创建一个 pool。pool reconciler 会验证每个成员是否存在，计算容量计数器，并将 `baremetal.alauda.io/pool=<pool-name>` annotation 写入 inventory。

02-machineinventorypool.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: MachineInventoryPool
metadata:
  name: <cluster-name>-control-plane-pool
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  machineInventories:
    - name: <control-plane-inventory-1>
      hostname: <control-plane-host-1>    # Optional; falls back to the i
nventory name.
    - name: <control-plane-inventory-2>
      hostname: <control-plane-host-2>
    - name: <control-plane-inventory-3>
      hostname: <control-plane-host-3>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: MachineInventoryPool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  machineInventories:
    - name: <worker-inventory-1>
      hostname: <worker-host-1>
    - name: <worker-inventory-2>
    - name: <worker-inventory-3>
```

关键参数：

参数	类型	描述	必需
`.spec.clusterName`	string	此 pool 服务的业务集群名称。一个 `MachineInventory` 不能出现在两个活动 pool 中。	是

参数	类型	描述	必需
`.spec.machineInventories[].name`	string	已注册 `MachineInventory` 的确切名称。	是
`.spec.machineInventories[].hostname`	string	重新置备期间应用的主机名。省略时默认为 inventory 名称。	否

应用并验证：

```
kubectl apply -f 02-machineinventorypool.yaml
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.x-k8s.io
```

健康的 pool 会报告 `Ready=True`、`total = len(spec.machineInventories)` 和 `available = total - allocated - preparing - reprovisioning - unavailable`。`spec.machineInventories` 中列出的 inventory 如果未通过验证（缺失、plan secret 缺失、Ready=False 或存储未准备好），会增加 pool 的 `unavailable` 计数器，并显示在 `MembersValid` condition 中。非空的存储声明是分配门槛：只有在基于最新 observation 成功完成 Prepare 后，该 inventory 才会对 CAPI 处于 `Available` 状态。

将 control-plane pool 的大小设置为至少 `KubeadmControlPlane.spec.replicas`。将 worker pool 的大小设置为至少 `MachineDeployment.spec.replicas`。对于滚动升级，pool 必须容纳完整的 replica 数量——provider 会在同一个 pool 中采用先删除后添加的语义，绝不会同时执行两者。

步骤 3：创建控制平面集群资源

创建 `BaremetalCluster`（声明所选的控制平面 endpoint 模式）、控制平面 `BaremetalMachineTemplate`（指向控制平面 pool）、`KubeadmControlPlane`（replicas + kubeadm 配置）以及 CAPI `Cluster`。

Bare Metal API 在同一操作流程中支持两种 endpoint 模式：

- `Internal`：provider 部署 Alive，并协调 VIP 和控制平面后端成员。

- `External`：provider 跳过 Alive；负载均衡器所有者维护 listener、健康检查和后端成员。

在下面的 manifest 中使用 `<control-plane-load-balancer-type>` 选择模式。

> **WARNING**
>
> 使用 **OS image** 中内置的 **Kubernetes image**
>
> 受支持的裸金属 OS image 会在 `cloud.alauda.io/alauda` 下预加载 kubeadm 控制平面、CoreDNS 和 etcd image。在 KCP 配置中保留该 repository，并使用 OS Support Matrix 中的组件 tag。不要将其替换为 `<registry-address>/tkestack`：这会生成与 OS 中预加载的 image 不匹配的 image 引用。OS 会通过 containerd 配置其内置的 pause image；不要将 `pod-infra-container-image` 添加到 KCP kubelet 参数中。

> **TIP**
>
> 完整配置参考
>
> 下面的示例使用最小化的 `KubeadmControlPlane`。如需生产环境推荐的完整加固配置文件——包括 admission、audit、kubelet 补丁和 encryption provider——请参阅附录中的完整 KubeadmControlPlane 配置。

03-cluster.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalCluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    type: <control-plane-load-balancer-type>
    host: <control-plane-vip>
    port: <control-plane-port>
    # Required only for Internal. Remove this field for External.
    vrid: <vrid>
  networkType: kube-ovn      # Enables provider-managed Kube-OVN AppRelea
se reconciliation.
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalMachineTemplate
metadata:
  name: <cluster-name>-control-plane-template
  namespace: cpaas-system
spec:
  template:
    spec:
      machineInventoryPoolRef:
        name: <cluster-name>-control-plane-pool
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0            # Bare-metal does not over-provision; replac
e one at a time.
  machineTemplate:
    nodeDrainTimeout: 5m
    nodeVolumeDetachTimeout: 5m
    infrastructureRef:
```

```
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: BaremetalMachineTemplate
      name: <cluster-name>-control-plane-template
  kubeadmConfigSpec:
    users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        shell: /bin/bash
        sshAuthorizedKeys:
          - "<ssh-authorized-keys>"
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
      controllerManager:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
    initConfiguration:
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
    joinConfiguration:
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  annotations:
```

```
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: BaremetalCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
    cpaas.io/kube-ovn-version: <kube-ovn-version>
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
  labels:
    cluster-type: ProviderBaremetal
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - <pods-cidr>
    services:
      cidrBlocks:
        - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-control-plane
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: BaremetalCluster
    name: <cluster-name>
```

集群 **annotation**。 裸金属 provider 在 reconcile 期间依赖一组 `Cluster` annotation。 operator 必须设置的权威 annotation 如下：

Annotation	必需	值来源	用途
`capi.cpaas.io/resource-group-version`	是	字面值 `infrastructure.cluster.x-k8s.io/v1beta1`	CAPI 基础设施绑定。
`capi.cpaas.io/resource-kind`	是	字面值 `BaremetalCluster`	CAPI 基础设施绑定。
`cpaas.io/kube-ovn-join-cidr`	是	operator 选择的 `/16` CIDR，不得与 pod/service 或任何其他集群重叠。	Kube-OVN 节点间隧道。

Annotation	必需	值来源	用途
`cpaas.io/kube-ovn-version`	建议	OS Support Matrix 中的 **kube-ovn**（**chart**） 值。	固定 provider 创建 `cni-kube-ovn` 时使用的 chart 版本。省略时，provider 回退到 `kube-ovn` `ModulePlugin` 版本。
`cpaas.io/sentry-deploy-type`	是	字面值 `Baremetal`。	将集群标记为裸金属部署 profile。
`cpaas.io/alb-address-type`	是	字面值 `ClusterAddress`。	裸金属集群使用的 ALB 地址模式。

`BaremetalCluster` 参数：

参数	类型	描述
`.spec.controlPlaneLoadBalancer.type`	string（`Internal` / `External`）	`Internal` 会部署 Alive static pod 理 VIP。`External` 会跳过 Alive，求预先置备 Layer 4 TCP LoadBalancer。
`.spec.controlPlaneLoadBalancer.host`	string	控制平面 endpoint。对于 `Interna` 使用 Self-built VIP。对于 `Externa` 使用负载均衡器前端地址。当 endp 留空时，会回填到 `.spec.controlPlaneEndpoint.ho`
`.spec.controlPlaneLoadBalancer.port`	int（1–65535）	控制平面端口。通常为 `6443`。
`.spec.controlPlaneLoadBalancer.vrid`	int（0–255）	Keepalived `virtual_router_id`。制平面 Layer-2 域中必须唯一。对于 `External`，移除此字段。

参数	类型	描述
`.spec.controlPlaneEndpoint`	object	向 CAPI 暴露的 API server endpoin 置后不得更改。留空则由 reconciler `controlPlaneLoadBalancer` 回填
`.spec.networkType`	string	由 provider 管理的 CNI 实现。将其 为 `kube-ovn`；任何其他值都会导 provider 跳过 Kube-OVN `AppRele` reconcile 和节点移除集成。

应用并观察：

```
kubectl apply -f 03-cluster.yaml
kubectl -n cpaas-system get clusters.cluster.x-k8s.io
kubectl -n cpaas-system get kubeadmcontrolplanes.controlplane.cluster.x-k
8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io -w
```

每个新的控制平面 `BaremetalMachine` 都会推进 `Pending → Allocated → Reprovisioning → Running`。观察：

- `BaremetalMachine.status.machineInventoryRef.name` —— 被选中的 inventory。
- `BaremetalMachine.status.planSecretRef.name` —— 正在驱动的 plan secret。该 secret 携带 annotation `baremetal.alauda.io/plan.type=reprovision`。
- `MachineInventory.status.plan.state` —— 主机完成 `cloud-init clean`、`elemental upgrade`、重启以及 `kubeadm init` / `join` 后变为 `Applied`。
- `BaremetalCluster.status.conditions[EndpointReady]` —— 配置的控制平面 endpoint 可访问后变为 true。

> **WARNING**
>
> 裸金属 provider 不支持单节点控制平面。至少置备三个控制平面 replica（`KubeadmControlPlane.spec.replicas: 3`），以确保 etcd 保持仲裁。在 `Internal` 模式下，相同节点还会参与 Alive VIP 仲裁。

步骤 4：部署工作节点

控制平面 Ready 后，创建 worker `BaremetalMachineTemplate`、worker `KubeadmConfigTemplate` 和 `MachineDeployment`。

确认 worker pool 容量

```
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.x-k8s.io <cluster-name>-worker-pool
```

`status.available` 必须至少等于目标 replica 数量。如果需要额外容量，请注册更多主机（在这些主机上启动 `SeedImage` ISO——参见步骤 1），在每个 inventory 仍未分配时准备任何托管存储，然后将其名称添加到 `spec.machineInventories`。

以此方式添加的主机会经历与原始节点相同的首次启动流程，因此必须满足相同的先决条件：网络连接中所述的网络路径，以及主机时间同步中的时钟门槛。请为新主机确认这两项，不要假定交付时获得的批准涵盖其子网和硬件时钟。

配置 worker `BaremetalMachineTemplate`

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      machineInventoryPoolRef:
        name: <cluster-name>-worker-pool
```

关键参数：

参数	类型	描述
`.spec.template.spec.machineInventoryPoolRef.name`	string	worker pool 名称（同一 namespace）。

参数	类型	描述
		如需将 worker 移动到其他 pool，请参阅将 worker 移动到其他 pool。

配置 **bootstrap template**

`imagePullCredentialsVerificationPolicy: NeverVerify` 仅从 Kubernetes 1.35 开始需要。使用 Kubernetes 1.34 或更早版本创建 worker 时，请省略此参数。

```yaml
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-bootstrap
  namespace: cpaas-system
spec:
  template:
    spec:
      users:
        - name: boot
          sudo: ALL=(ALL) NOPASSWD:ALL
          shell: /bin/bash
          sshAuthorizedKeys:
            - "<ssh-authorized-keys>"
      files:
        - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
son
          owner: "root:root"
          permissions: "0644"
          content: |
            {
              "apiVersion": "kubelet.config.k8s.io/v1beta1",
              "kind": "KubeletConfiguration",
              "imagePullCredentialsVerificationPolicy": "NeverVerify",
              "protectKernelDefaults": true,
              "staticPodPath": null,
              "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
              "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
              "streamingConnectionIdleTimeout": "5m",
              "clientCAFile": "/etc/kubernetes/pki/ca.crt"
            }
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
```

provider 会在将 bootstrap user-data 写入 `reprovision` plan 之前进行最小化规范化，因此请将以下字段从 template 中省略：

- **Hostname / FQDN** —— 自动根据 `MachineInventoryPool.spec.machineInventories[].hostname` 设置（省略时使用 inventory 名称）。

- **`kubeletExtraArgs.provider-id`** —— 自动设置为 `baremetal:///<inventory-name>`。
- **`nodeRegistration.criSocket`** —— 未设置时自动设置为 `unix:///var/run/containerd/containerd.sock`。

规范化后的 user-data 也会写回 bootstrap secret 中的 `data["resolved-value"]`，以便调试；原始 `data["value"]` 保持不变。

配置 `MachineDeployment`

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-workers
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0        # Bare-metal pools cannot over-provision.
      maxUnavailable: 1
  selector:
    matchLabels: {}
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster-name>
        pool.name: <cluster-name>-workers
    spec:
      clusterName: <cluster-name>
      version: <kubernetes-version>
      nodeDrainTimeout: 5m
      nodeVolumeDetachTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-bootstrap
          namespace: cpaas-system
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: BaremetalMachineTemplate
        name: <cluster-name>-worker-template
        namespace: cpaas-system
```

关键参数：

参数	类型	描述
`.spec.clusterName`	string	目标集群名称。
`.spec.replicas`	int	worker 节点数量。必须满 `MachineInventoryPool.s` `+ status.allocated`（ pool）。
`.spec.template.spec.version`	string	worker Kubernetes 版本。 `elemental-image-catal` 与控制平面版本不同，但必 kubelet 版本偏差策略。
`.spec.strategy.rollingUpdate.maxSurge`	int	裸金属不会超额配置—— 没有可用的备用物理主机来 点。
`.spec.strategy.rollingUpdate.maxUnavailable`	int	当 `maxSurge=0` 时，必须 会在此预算内逐个回收。

应用并观察：

```
kubectl apply -f workers.yaml
kubectl -n cpaas-system get machinedeployments.cluster.x-k8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io -w
kubectl get nodes -o wide                    # workload cluster
```

集群验证

使用 **kubectl**

```
# Cluster status
kubectl -n cpaas-system get cluster <cluster-name>

# Control plane progress
kubectl -n cpaas-system get kubeadmcontrolplane <cluster-name>-control-pl
ane

# Per-machine state
kubectl -n cpaas-system get machines.cluster.x-k8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io

# Pool counters
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.
x-k8s.io

# Storage lifecycle for inventories that declare managed volumes
kubectl -n cpaas-system get machineinventories.elemental.cattle.io \
  -o custom-columns='NAME:.metadata.name,PHASE:.status.storage.phase,PREP
ARED:.status.storage.conditions[?(@.type=="StoragePrepared")].status,ACTI
VE:.status.storage.conditions[?(@.type=="StorageActive")].status'

# Workload cluster Nodes (use the workload kubeconfig)
kubectl get nodes -o wide
```

验证控制平面端点

确认配置的模式和端点：

```
kubectl -n cpaas-system get baremetalcluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneLoadBalancer.type}{" "}{.spec.controlPl
aneEndpoint.host}{":"}{.spec.controlPlaneEndpoint.port}{"\n"}'
```

对于 `Internal`，验证 Alive 状态和节点前置条件：

```
kubectl --kubeconfig <workload-kubeconfig> -n kube-system get pods -l app
=alive -o wide
kubectl --kubeconfig <workload-kubeconfig> get --raw=/version
```

在每个控制平面节点上运行以下命令，而不是在管理集群上运行：

```
sysctl net.ipv4.conf.all.arp_accept net.ipv4.vs.conntrack
```

对于 `External`，根据External LoadBalancer contract 验证负载均衡器前端和每个后端。确认负载均衡器所有者已注册当前所有控制平面节点，并确认每个后端的 HTTPS `/healthz` 返回 HTTP `200`。

预期结果

成功创建的集群应显示：

- `Cluster.status.conditions[Ready]=True`。
- `KubeadmControlPlane` 个副本全部为 `Ready`。
- 每个 `BaremetalMachine.status.phase=Running` 和 `status.ready=true`。
- 每个使用的 `MachineInventory.status.plan.state=Applied` 都在其 plan secret 上带有注解 `baremetal.alauda.io/plan.type=reprovision`（这是注解而不是标签——`-o jsonpath='{.metadata.annotations}'`）。
- 每个使用的、`spec.storage.volumes[]` 非空的 inventory 都报告 `status.storage.phase=Active`、`StoragePrepared=True` 和 `StorageActive=True`；对应的 `BaremetalMachine` 报告 `StorageReady=True/AllVolumesReady`。
- `MachineInventoryPool.status` 满足 `available + allocated + preparing + reprovisioning + unavailable = total`。
- Kubernetes Nodes 均为 Ready。
- 对于 `Internal`，Alive Pods 均为 Ready，并且可以通过自建 VIP 访问 API。
- 对于 `External`，负载均衡器前端可访问，并且其后端列表与当前控制平面节点匹配。

常见故障模式

症状	可能原因
`BaremetalMachine` 停留在 `Pending`、`InventoryAllocated=False / Reason=PoolMissing`	`BaremetalMachineTemplate.spec.template.sp` 指向不存在的池。
`BaremetalMachine` 停留在 `Pending`、`InventoryAllocated=False / Reason=PoolExhausted`	池中没有剩余的 `Available` inventory。
`BaremetalMachine` 在分配后停滞，`BootstrapReady=False / Reason=BootstrapWaiting`	KubeadmConfig 尚未生成 bootstrap data secret。
`BaremetalMachine` 位于 `Failed`、`ImageResolved=False / Reason=ImageCatalogMiss`	目标 `Machine.spec.version` 不是 `elemental` 建的 catalog 不包含任何条目。
`ImageResolved=False / Reason=ImageRegistryMissing`	可选的 `cpaas.io/registry-address` 注解和平析 registry 地址。
inventory 已注册，但未计为 `Available`	其 managed storage 声明尚未达到 `StoragePrep` 或已过期。
`BaremetalMachine` 等待并显示 `StorageReady=False`	所需的 managed volume 未激活，或没有新的重启
重新配置始终无法完成；`MachineInventory.status.plan.state=Failed`	`elemental upgrade` 失败（registry 无法访问、
`BaremetalCluster` 使用 `Internal`，但无法访问 VIP	`vrid` 与另一个集群冲突；VIP 不在控制平面 L2 止；IPVS 或所需的 sysctl 不可用。
`BaremetalCluster` 使用 `External`，但无法访问端点	Listener、后端成员关系、TLS 透传、健康检查、

有关完整的 operator 侧状态机参考（每个 condition reason 和恢复操作），请参阅 Provider Overview → clean / reprovision plans。

后续步骤

创建集群后：

- 管理节点
- 管理数据磁盘
- 升级集群

附录

完整的 `KubeadmControlPlane` 配置

生产环境裸金属集群推荐使用的加固配置——包括准入控制、审计策略、kubelet 补丁、加密 provider 和 IPv6 绑定地址。请替换解析占位符值表中的占位符。

> **Kubernetes 1.35 kubelet** 设置
>
> 仅从 Kubernetes 1.35 开始才需要 `imagePullCredentialsVerificationPolicy: NeverVerify`。
>
> 使用 Kubernetes 1.34 或更早版本创建集群时，请省略此参数。

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  machineTemplate:
    nodeDrainTimeout: 5m
    nodeVolumeDetachTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: BaremetalMachineTemplate
      name: <cluster-name>-control-plane-template
  kubeadmConfigSpec:
    users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        shell: /bin/bash
        sshAuthorizedKeys:
          - "<ssh-authorized-keys>"
    files:
      - path: /etc/kubernetes/admission/psa-config.yaml
        owner: "root:root"
        permissions: "0644"
        content: |
          apiVersion: apiserver.config.k8s.io/v1
          kind: AdmissionConfiguration
          plugins:
          - name: PodSecurity
            configuration:
              apiVersion: pod-security.admission.config.k8s.io/v1
              kind: PodSecurityConfiguration
              defaults:
                enforce: "privileged"
                enforce-version: "latest"
                audit: "baseline"
                audit-version: "latest"
```

```
            warn: "baseline"
            warn-version: "latest"
          exemptions:
            usernames: []
            runtimeClasses: []
            namespaces:
            - kube-system
            - cpaas-system
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "protectKernelDefaults": true,
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
          "streamingConnectionIdleTimeout": "5m",
          "clientCAFile": "/etc/kubernetes/pki/ca.crt"
        }
    - path: /etc/kubernetes/audit/policy.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        omitStages:
        - "RequestReceived"
        rules:
        - level: None
          users:
          - system:kube-controller-manager
          - system:kube-scheduler
          - system:serviceaccount:kube-system:endpoint-controller
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
          - group: ""
            resources: ["endpoints"]
        - level: None
          nonResourceURLs:
```

```
            - /healthz*
            - /version
            - /swagger*
          - level: None
            resources:
            - group: ""
              resources: ["events"]
          - level: None
            verbs: ["get", "list", "watch"]
          - level: None
            resources:
            - group: "coordination.k8s.io"
              resources: ["leases"]
          - level: None
            resources:
            - group: "authorization.k8s.io"
              resources: ["subjectaccessreviews", "selfsubjectaccessrevie
ws"]
            - group: "authentication.k8s.io"
              resources: ["tokenreviews"]
          - level: Metadata
            resources:
            - group: ""
              resources: ["secrets", "configmaps"]
          - level: RequestResponse
            resources:
            - group: ""
            - group: "apps"
            - group: "rbac.authorization.k8s.io"
            - group: "storage.k8s.io"
            - group: "networking.k8s.io"
          - level: Metadata
    postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          audit-log-format: json
```

```
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          profiling: "false"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          tls-cipher-suites: "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS
_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY130
5,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          tls-min-version: VersionTLS12
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
        extraVolumes:
        - name: vol-dir-0
          hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          tls-min-version: VersionTLS12
          profiling: "false"
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
```

Worker bootstrap 采用对称配置——有关 worker `KubeadmConfigTemplate`，请参阅配置 Bootstrap 模板。

在多磁盘裸金属主机上选择固定系统磁盘

当一个 `SeedImage` ISO 必须在多个裸金属主机之间重复使用，且每台主机都拥有多个磁盘时，请使用本指南。本操作步骤可防止 Elemental 在每次启动时自动选择不同的磁盘，并允许 operator 根据稳定的 WWN 在每台主机上选择系统磁盘。

目录

为什么必须设置手动磁盘门控

先决条件

配置共享的 MachineRegistration

准备每台主机

1. 启动预期的 Live ISO 并阻止自动安装
2. 确认主机正在从 Live ISO 运行
3. DHCP 不可用时配置 Live ISO 网络
4. 识别所有磁盘和预期的系统磁盘
5. 重新安装主机时移除陈旧的 Elemental 标签
6. 创建每台主机的安装目标
7. 运行最终门控并开始安装

验证已安装的主机

故障排查

限制

为什么必须设置手动磁盘门控

在多磁盘主机上，不要将 `MachineRegistration.spec.config.elemental.install.device` 留空。空设备和空的 `device-selector` 并不会让 Elemental 等待 operator 选择磁盘。多个已发现的磁盘可能匹配，其中一个磁盘会在未进行唯一性检查的情况下被选中。

也不要将 `/dev/sda` 用作持久化磁盘标识。存储控制器的发现顺序可能会在 Live ISO 与已安装系统之间发生变化，因此同一个逻辑磁盘在一次启动期间可能显示为 `/dev/sda`，而在另一次启动期间显示为 `/dev/sdb`。

本指南使用一种失败即关闭的手动门控机制：

1. 每个可重复使用的 ISO 都包含相同的、初始不存在的安装路径：`/dev/elemental-install-target`。
2. 该路径不存在时，安装服务无法继续，并且不会回退到自动磁盘选择。
3. 在每台主机上，operator 根据 `/dev/disk/by-id/wwn-*` 识别目标整盘，并创建指向该磁盘的符号链接作为门控路径。
4. 只有在磁盘、网络、挂载和陈旧标签检查均通过后，operator 才会启动安装服务。

> **WARNING**
>
> 本指南中的清理操作会永久销毁数据。只能从 Live ISO 控制台运行该操作，只能在获准重新安装的磁盘上运行，并且必须先确认这些磁盘均未挂载。

先决条件

- 裸金属 provider 和 `elemental-operator` 已安装在 `global` 集群上。
- 你可以在 `cpaas-system` 中创建或编辑 `MachineRegistration` 和 `SeedImage` 资源。
- 你可以通过 BMC、iDRAC、iLO 或同等接口访问每台主机的控制台。
- 每个目标系统磁盘都有稳定的 `/dev/disk/by-id/wwn-*` 条目。
- 你知道哪些磁盘可以擦除。如果主机包含必须保留的数据磁盘，请在每个清理命令中排除这些磁盘。

- 如果 Live ISO 网络没有 DHCP 服务，你已准备好主机 IP 地址、前缀、网关、DNS 服务器和 Global 注册 URL。

有关完整的 provider 和集群创建工作流，请参阅在裸金属上创建集群。

配置共享的 MachineRegistration

将安装设备设置为 `/dev/elemental-install-target`。不要为此工作流配置 `device-selector`。

MachineRegistration install settings

```yaml
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: <cluster-name>-registration
  namespace: cpaas-system
spec:
  machineName: "<cluster-name>-${System Information/UUID}"
  config:
    elemental:
      install:
        device: /dev/elemental-install-target
        eject-cd: true
        reboot: true
        snapshotter:
          type: btrfs
          maxSnaps: 4
      registration:
        auth: tpm
        # Use hardware TPM when available. For hosts without TPM, enable
        # emulation and derive a unique identity for every host.
        emulate-tpm: true
        emulated-tpm-seed: -1
```

Live ISO 启动时，该路径会被有意保留为不存在。文件系统标签、GPT `PARTLABEL` 或 `/dev/disk/by-label/*` 路径都不能替代它：这些用于标识分区，而 Elemental 要求使用整盘安装目标。

创建引用此 MachineRegistration 的 `SeedImage`，或者在更改安装设备后重新构建现有的 SeedImage。该 `SeedImage` 还必须采用首次安装前调整 COS_STATE 大小中所述的 `state` 分区大小 — 主机升级到不同镜像后，Elemental 默认的 8 GiB 太小，并且必须重新安装主机才能更改该大小。

> **WARNING**
>
> MachineRegistration 设置会嵌入生成的 ISO 中。修改 MachineRegistration 不会更改已经构建并下载的 ISO。请重新构建 SeedImage，等待其就绪，下载新的 ISO，并在使用该 ISO 启动主机前验证其校验和。

准备每台主机

一次只对一台主机执行以下步骤。需要清理磁盘时，不要并行准备多台主机。

1. 启动预期的 Live ISO 并阻止自动安装

从共享 ISO 启动主机。Live ISO 控制台可用后，立即停止并屏蔽自动安装程序：

```
systemctl stop elemental-register-install.service
systemctl mask elemental-register-install.service
systemctl reset-failed elemental-register-install.service
```

确认 ISO 包含手动门控路径：

```
grep -n -C 3 '/dev/elemental-install-target' \
  /run/initramfs/live/livecd-cloud-config.yaml
```

如果路径不存在，说明主机未启动预期的 ISO。不要继续。

2. 确认主机正在从 Live ISO 运行

```
findmnt -n -o SOURCE,FSTYPE,TARGET /
findmnt /run/initramfs/live
lsblk -e7 -o NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,FSTYPE,LABEL,MOUNTPOINT
S
```

确认以下所有事项：

- `/` 是 Live 根目录，例如 `LiveOS_rootfs`。
- 虚拟 CD 设备（通常为 `/dev/sr0`）具有 `COS_LIVE` 标签。
- 你计划清理或用作安装目标的磁盘均未挂载。

检查物理磁盘、device-mapper、swap、LVM 和 multipath 的使用情况：

```
findmnt -rn -o SOURCE,TARGET | grep -E '^/dev/(sd|mapper/)' || \
  echo NO_PHYSICAL_DISK_MOUNTS
swapon --show
pvs 2>/dev/null || true
vgs 2>/dev/null || true
multipath -ll 2>/dev/null || true
```

如果目标磁盘已挂载，或者无法证明根文件系统属于 Live ISO，请停止操作。

3. DHCP 不可用时配置 Live ISO 网络

首先识别实际的 NetworkManager 连接和接口。不要假设连接始终名为 `Wired connection 1`。

```
nmcli device status
nmcli connection show
ip -br link
```

使用分配给此主机的值配置选定的连接：

```
nmcli connection modify "Wired connection 1" \
  ipv4.method manual \
  ipv4.addresses 192.168.254.76/24 \
  ipv4.gateway 192.168.254.1 \
  ipv4.dns 192.168.16.19 \
  ipv4.ignore-auto-dns yes

nmcli connection up "Wired connection 1"
```

验证地址、默认路由、DNS 解析以及完整的注册 URL：

```
ip -br addr
ip route
cat /etc/resolv.conf
ping -c 2 -W 2 192.168.254.1
getent hosts <global-domain>
curl -ksS --connect-timeout 10 \
  -o /dev/null -w 'REG_HTTP=%{http_code}\n' \
  'https://<global-domain>/<registration-path-and-token>'
```

使用特定于站点的网络值，而不要直接复制示例地址。不要将完整的注册 URL 和令牌存入源代码管理系统或共享日志。

此网络配置仅用于连接 Live ISO，以完成注册和初始安装。它不会替代 Cluster API 重新置备期间应用于已安装系统的持久网络配置。

4. 识别所有磁盘和预期的系统磁盘

列出稳定的磁盘属性：

```
lsblk -d -e7 -o NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,HCTL

for current_disk in /dev/sda /dev/sdb /dev/sdc /dev/sdd; do
  echo "=== ${current_disk} ==="
  udevadm info -q symlink -n "${current_disk}" | tr ' ' '\n' |
    grep -E 'disk/by-id/(wwn|scsi)-' || true
done
```

根据当前主机调整磁盘列表。记录并交叉核对：

- 当前内核设备名称。
- 大小、型号、序列号和控制器地址（`HCTL`）。
- `/dev/disk/by-id/wwn-*` 路径。
- 哪个整盘已获准作为系统磁盘。

使用 WWN 以及大小、型号和控制器信息做出决定。不要仅根据内核名称或容量选择磁盘。

5. 重新安装主机时移除陈旧的 Elemental 标签

对于必须保留数据的磁盘，跳过破坏性清理。完全重新安装主机时，请在清理获准的磁盘前移除旧映射：

```
swapoff -a
vgchange -an 2>/dev/null || true
systemctl stop multipathd.service multipathd.socket 2>/dev/null || true
systemctl mask multipathd.service multipathd.socket 2>/dev/null || true
multipath -F 2>/dev/null || true
udevadm settle
```

如果主机具有无关的 LVM、multipath 或共享存储设备，请不要使用全局停用命令。只能停用属于获准磁盘的映射。

对于每个获准的磁盘，请在磁盘开头和结尾清除其 GPT、已识别的签名和元数据。以下示例对 `/dev/sda` 执行操作；仅对你已独立识别并批准的每个磁盘重复该操作。

```
disk_to_wipe=/dev/sda

lsblk -d -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL "${disk_to_wipe}"
sgdisk --zap-all "${disk_to_wipe}"
wipefs -a -f "${disk_to_wipe}"
dd if=/dev/zero of="${disk_to_wipe}" \
  bs=1M count=64 conv=fsync status=progress

disk_sectors=$(blockdev --getsz "${disk_to_wipe}")
if [ "${disk_sectors}" -le 262144 ]; then
  echo 'REFUSE: disk is too small; stop and recheck the target'
else
  dd if=/dev/zero of="${disk_to_wipe}" bs=512 \
    seek=$((disk_sectors-131072)) count=131072 \
    conv=fsync status=progress
fi
```

清理所有获准的磁盘后，刷新内核和 udev：

```
partprobe /dev/sda /dev/sdb /dev/sdc /dev/sdd 2>/dev/null || true
udevadm trigger --subsystem-match=block
udevadm settle
```

调整 `partprobe` 列表，使其仅包含在此主机上清理的磁盘。

验证已清理的磁盘不具有签名、分区、文件系统类型、标签或分区标签：

```
wipefs /dev/sda /dev/sdb /dev/sdc /dev/sdd
lsblk -e7 -o \
NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,FSTYPE,LABEL,PARTLABEL,MOUNTPOINTS

if blkid /dev/sda /dev/sdb /dev/sdc /dev/sdd 2>/dev/null |
  grep -E 'LABEL="(OLD_)?COS_(GRUB|OEM|RECOVERY|STATE|PERSISTENT)"'; then
  echo COS_LABELS_REMAIN
else
  echo NO_COS_LABELS
fi
```

调整每个设备列表，使其与已清理的磁盘匹配。仅当以下条件均满足时继续：

- `wipefs` 对任何已清理的磁盘均不打印签名。
- `lsblk` 不显示旧的子分区、`FSTYPE`、`LABEL` 或 `PARTLABEL`。
- 检查会打印 `NO_COS_LABELS`。
- 唯一剩余的 `COS_LIVE` 标签属于虚拟 CD 设备，而不是系统磁盘。

6. 创建每台主机的安装目标

清理后再次列出稳定的 WWN 路径：

```
ls -l /dev/disk/by-id/wwn-*
lsblk -d -e7 -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL
```

选择并验证预期的整盘后，再创建门控路径。将示例 WWN 替换为记录的此主机 WWN：

```
test ! -e /dev/elemental-install-target
test ! -L /dev/elemental-install-target

ln -s \
  /dev/disk/by-id/wwn-0x0123456789abcdef \
  /dev/elemental-install-target

readlink -f /dev/elemental-install-target
test -b /dev/elemental-install-target && echo TARGET_IS_BLOCK
lsblk -d -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL \
  "$(readlink -f /dev/elemental-install-target)"
```

如果别名已存在或是悬空链接，请不要覆盖它。使用 `readlink` 检查该别名，确定其存在原因，并仅在确认其已陈旧后将其移除。

该链接只存在于当前 Live ISO 会话中。重启后它应当消失；已安装系统正常启动不需要该链接。

7. 运行最终门控并开始安装

最后一次验证 Live 根目录、挂载、目标链接、网络和 DNS：

```
findmnt -n -o SOURCE,FSTYPE,TARGET /
findmnt -rn -o SOURCE,TARGET | grep -E '^/dev/(sd|mapper/)' || \
  echo NO_PHYSICAL_DISK_MOUNTS
readlink -f /dev/elemental-install-target
test -b /dev/elemental-install-target
ip -br addr
ip route show default
getent hosts <global-domain>
```

仅在所有检查均通过后启动安装程序：

```
systemctl unmask elemental-register-install.service
systemctl reset-failed elemental-register-install.service
systemctl start elemental-register-install.service
journalctl -fu elemental-register-install.service
```

在日志中确认注册成功、`/dev/elemental-install-target` 解析到预期的 WWN 磁盘、分区和安装完成，并且配置的弹出/重启操作已执行。

验证已安装的主机

安装触发重启后：

1. 弹出 BMC 虚拟介质，并恢复正常的磁盘优先启动顺序。
2. 确认主机不会重新进入 Live ISO。
3. 在 Global 集群上验证新的 `MachineInventory`：

   ```
   kubectl -n cpaas-system get machineinventories.elemental.cattle.io -o w
   ide
   kubectl -n cpaas-system describe machineinventory <inventory-name>
   ```

4. 确认 inventory 为 `Ready=True`，报告了预期的网络，并且只有一个当前的 Elemental 磁盘布局。
5. 只有在 inventory 就绪后，才将其添加到 `MachineInventoryPool`。

6. Cluster API 分配主机后，按照在裸金属上创建集群中的说明，验证 `BaremetalMachine`、工作负载 Node、控制平面组件和 CNI。

故障排查

症状	原因和操作
在选择磁盘之前，安装服务就失败了。	这是 `/dev/elemental-install-target` 不存在时预期的安全失败行为。停止并屏蔽该服务，完成检查，创建链接，然后重置并启动服务。
`test -b /dev/elemental-install-target` 失败。	WWN 路径错误、不存在，或未解析为整个块设备。重新检查 `lsblk` 和 `/dev/disk/by-id`；不要启动安装。
仍有多个磁盘具有 `COS_STATE` 或 `COS_PERSISTENT`。	仍然存在之前的 Elemental 布局。停止安装，并清理每个获准用于重新安装的磁盘，直到 `wipefs`、`blkid` 和 `lsblk` 均通过。
同一物理磁盘在重启后显示不同的 `/dev/sdX` 名称。	某些存储控制器出现这种情况是正常的。请比较 WWN、序列号、容量和控制器地址，而不要比较内核名称。
MachineRegistration 已打补丁，但 ISO 仍使用旧目标。	重新构建 SeedImage 并下载新生成的 ISO。MachineRegistration 的更改不会追溯应用到现有 ISO。
Live ISO 中注册成功，但重新配置后已安装的主机没有网络。	请分别诊断已安装系统的 cloud-init 和 NetworkManager 激活情况。Live ISO 的 `nmcli` 配置文件是临时的，不能证明持久化的重新配置网络已成功。

限制

- 此工作流有意要求对每台主机执行一次控制台操作。它会阻止自动选择，但不会提供完全无人值守的异构磁盘选择。
- MachineRegistration 元数据标签不会插入到 `install.device` 中。
- 仅基于磁盘大小的 `device-selector` 只有在确切地只有一个磁盘可以匹配时才是安全的，而当前选择不会强制要求唯一性。对于具有多个相同大小磁盘的主机，首选 WWN gate。
- 目标必须是整个磁盘。不要将 gate 指向分区。

Huawei DCS

概览 安装 **Provider** 基础设施资源

创建集群 排查卡在 **Deleting** 状态的 **DCSMachine**

Huawei DCS Provider

目录

概述
主要功能
集群管理能力
基于 UI 的管理（Fleet Essentials）
基于 YAML 的管理
支持的 Kubernetes 版本
架构
概念和术语
DCS 平台对象层级
DCS 平台概念
DCS Site
DCS Cluster
DCS Host
DCS VM Folder
DCS Cluster Folder
DCS VM Template
DCS Datastore
DCS Datastore Cluster
DCS Distributed Virtual Switch (DVS)
DCS Port Group

DCS Volume

DCS Cloud Credential

Cluster API 资源映射

存储概念：Datastore、Datastore Cluster 和 Volume

网络概念：DVS、Port Group 和 IP 标识

字段与概念参考

`DCSCluster` 字段

`DCSMachineTemplate` / `DCSMachine` 字段

`DCSIpHostnamePool` 字段

要求

文档

概述

Huawei DCS Infrastructure Provider 可在 Huawei Datacenter Virtualization Solution (DCS) 平台上创建和管理 Alauda OS 集群。

DCS（Datacenter Virtualization Solution）是 Huawei 的虚拟化平台，提供企业级虚拟化能力。DCS Provider 与 DCS 集成，为 Kubernetes 集群管理虚拟机、网络和存储资源。

主要功能

- 虚拟机管理：在 DCS 平台上配置和管理 VM
- 网络配置：支持多 NIC 配置，并可自定义网络设置
- 存储管理：支持灵活的磁盘配置，可在多个数据存储上使用模板磁盘和基于 IP 槽位的持久磁盘
- 高可用性：内置对高可用控制平面部署的支持
- 自建 **VIP**：通过 `alive` 运行时和 ACP `v4.4+` 管理控制平面 VIP（`DCSCluster.spec.controlPlaneLoadBalancer.type: internal`）。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

- 滚动更新：自动执行滚动更新，实现零停机维护
- 升级数据保留：在滚动升级期间，可将声明的持久磁盘从旧 VM 分离，并重新连接到替换 VM，从而避免保留的节点本地数据随系统磁盘丢失

集群管理能力

DCS Provider 始终支持通过 Cluster API 清单进行基于 **YAML** 的集群管理。 当安装 **Fleet Essentials** 后，它还支持基于 **UI** 的集群管理。 `DCSIpHostnamePool` 上的持久磁盘声明仍仅支持 YAML。

基于 **UI** 的管理（**Fleet Essentials**）

> **INFO**
>
> **Fleet Essentials** 升级边界
>
> Fleet Essentials 1.0.4 及更高版本可以通过 CVO 请求 ACP 4.3 及更高版本的 Distribution Version 升级。它不会执行 DCS Kubernetes 和 Alauda OS 替换。有关第 2 阶段的 YAML 操作步骤，请参阅在 Huawei DCS 上升级 Kubernetes。Fleet Essentials 中的集群创建和常规节点池管理不受此边界影响。

Fleet Essentials 提供 UI 扩展锚点，使 Alauda Container Platform Infrastructure Provider for Huawei DCS 能够动态添加 DCS 专属页面。 安装 Fleet Essentials 后，您可以通过 Web UI 管理 DCS 集群。 `DCSIpHostnamePool` 上的 `persistentDisk` 声明仍必须通过 YAML 清单管理：

基础设施资源管理：

- Cloud Credentials - 存储 DCS 平台访问凭证
- IP Pools - 管理 IP 地址分配和多 NIC 设置。 `/var/cpaas` 等基于 IP 槽位的持久磁盘仍通过 YAML 管理
- Machine Templates - 定义控制平面和工作节点的 VM 规格，但不包括由池管理的持久磁盘

集群生命周期：

- 创建集群 - 用于引导式集群创建的 5 步向导工作流
- 通过 Overview、Nodes、Node Pools 和 Aligned Extensions 选项卡查看集群详细信息
- 升级集群 - 两阶段升级：Distribution Version → Kubernetes
- 通过确认对话框销毁集群

节点池管理：

- 查看控制平面和工作节点池
- 添加工作节点池，并配置副本数和发布策略
- 删除工作节点池
- 按节点池升级 Kubernetes 版本
- 查看控制平面节点池的详细 Conditions

基于 YAML 的管理

对于自动化和 GitOps 工作流，您可以使用 YAML 清单管理 DCS 集群。此工作流不依赖 Fleet Essentials：

- 定义基础设施资源（Secret、DCSIpHostnamePool、DCSMachineTemplate）
- 创建集群资源（KubeadmControlPlane、DCSCluster、Cluster）
- 管理工作节点（MachineDeployment、KubeadmConfigTemplate）
- 执行滚动更新和升级，并在节点替换期间保留已声明的持久磁盘

有关基于 YAML 的操作步骤，请参阅在 Huawei DCS 上创建集群。

支持的 Kubernetes 版本

DCS Provider 支持的 Kubernetes 版本以 OS 支持矩阵 中的定义为准。每个 Alauda OS 版本都包含特定的 Kubernetes 版本，必须使用对应的 Alauda OS 镜像创建 VM 模板。

架构

DCS Provider 实现了 Cluster API 基础设施提供程序规范：

- **DCSCluster**：表示 DCS 基础设施集群
- **DCSMachine**：表示单个虚拟机
- **DCSMachineTemplate**：定义用于创建机器的 VM 模板
- **DCSIpHostnamePool**：管理机器的 IP 和主机名分配

概念和术语

多个 Huawei DCS 平台概念的名称与 Cluster API 自定义资源名称十分相似。其中两个需要特别注意，因为它们的名称完全相同：`DCS Cluster`（DCS 平台上的物理主机池）与 `DCSCluster` 自定义资源（表示在 DCS 上运行的 Kubernetes 集群的 Cluster API 基础设施资源）并非同一对象。若将二者视为同一对象，会导致 `DCSMachineTemplate` 和 `DCSCluster` 配置错误。

本节列举了该 provider 引用的 DCS 平台概念，将每个概念映射到引用它的 Cluster API 资源或字段，并说明网络和存储的分层关系；这是最常见的配置错误来源。

DCS 平台对象层级

DCS 平台以层级结构组织对象。该 provider 使用以下层级（从上到下）：

- **DCS Site** — 单个 DCS 部署的顶层范围。通过站点 ID 标识，例如 `EC1C108C`。
- 在站点内：
  - **DCS Cluster** — 物理主机池。一个站点包含一个或多个 DCS Cluster。
  - **DCS VM Folder** 和 **DCS Cluster Folder** — 分别用于对虚拟机和集群进行逻辑分组的组织标签。
  - **DCS Datastore** 和 **DCS Datastore Cluster** — 存储容器及存储容器组。
  - **DCS Distributed Virtual Switch (DVS)** — 覆盖整个站点的虚拟交换机。
- 在 DCS Cluster 内：
  - **DCS Host** — 物理服务器（例如 `CNA-01`、`CNA-02`）。
- 在 DCS DVS 内：
  - **DCS Port Group** — 虚拟机 NIC 所连接的网络配置项。

- 与放置层级无关：
  - **DCS VM Template** — 标记为模板的虚拟机，用作克隆的源镜像。
  - **DCS Volume** — 在 DCS Datastore 上创建的持久磁盘。

该 provider 不拥有这些对象，而是通过自定义资源字段中的名称或 URN 引用它们。

DCS 平台概念

DCS Site

单个 DCS 部署的顶层容器。站点 ID 通过 `DCSCluster.spec.site` 提供，也存储在云凭证 `Secret` 中。该 provider 使用的所有 DCS API 路径都包含站点 ID（`/service/sites/{siteId}/...`）。

DCS Cluster

DCS 平台上的物理主机池。DCS 调度程序会将虚拟机放置到属于单个 DCS Cluster 的主机上。一个站点可以包含一个或多个 DCS Cluster；一个 DCS Cluster 包含一个或多个 DCS Host。DCS Cluster 名称在 DCS 门户中可见，并由 `GET /service/sites/{siteId}/clusters` API 返回。

名称 `DCS Cluster` 与 Cluster API 自定义资源 `DCSCluster` 冲突。请参阅下文的Cluster API 资源映射。

DCS Host

在 DCS Cluster 中注册的单台物理服务器。名称遵循 DCS 平台上的本地约定（例如 `CNA-01`）。虚拟机在主机上运行；DCS 调度程序会根据集群级策略（DRS、反亲和规则）和主机容量，选择承载新虚拟机的主机。该 provider 不会将虚拟机固定到特定的 DCS Host；`DCSMachineTemplate` 资源不提供主机绑定字段。

DCS VM Folder

DCS Site 内用于对虚拟机分组的组织标签。VM Folder 不影响计算资源放置；归入 VM Folder `foo` 的虚拟机仍会运行在属于某个 DCS Cluster 的 DCS Host 上。该 provider 通过 `GET /service/sites/{siteId}/folder?type=1` 查询 VM Folder。

必须先在 DCS 平台上创建 VM Folder，才能在 `DCSMachineTemplate.spec.template.spec.location`（使用 `type: folder`）中引用它们。

DCS Cluster Folder

用于对 DCS Cluster 本身分组的组织标签，由同一个 `/folder` 端点返回，但使用不同的 `type` 参数。该 provider 不会查询或引用 DCS Cluster Folder。此处列出这一概念，仅为避免与 DCS VM Folder 混淆。

DCS VM Template

DCS 平台上标记为模板的虚拟机（`isTemplate=true`）。该 provider 从此模板克隆以创建集群节点。VM Template 通过名称（`DCSMachine.spec.vmTemplateName` 的值）标识；该 provider 通过列出 `GET /service/sites/{siteId}/vms?isTemplate=true&name=<name>`，将名称解析为 URN。

VM Template 自带位于 DCS Datastore 上的磁盘及网络配置。克隆时，该 provider 会覆盖网络配置（NIC 端口组、自定义配置），但不会覆盖磁盘的存储类型。默认情况下，克隆出的虚拟机会继承模板所在的 DCS Cluster；有关跨集群使用，请参阅基础设施 → 高级：多集群部署。

DCS Datastore

DCS 平台上的存储容器，用于存放虚拟机磁盘。Datastore 可由 SAN、IPSAN、NAS、FusionStorage 或本地磁盘提供支持。存储后端决定一个 datastore 是仅对一个 DCS Cluster 可见，还是对多个 DCS Cluster 可见：SAN、IPSAN、NAS 和 FusionStorage 通常支持跨集群访问；本地磁盘则不支持。

磁盘规格通过 `datastoreName` 或 `datastoreUrn` 引用 Datastore；该 provider 会在发送克隆请求前将名称解析为 URN。

DCS Datastore Cluster

一组 DCS Datastore。磁盘规格通过 `datastoreClusterName` 引用 Datastore Cluster 时，DCS 调度程序会选择其中有可用容量的成员 datastore。如果 DCS 管理员已配置 datastore cluster，建议以这种方式声明存储，因为它能够应对单个 datastore 空间耗尽的情况。

DCS Datastore Cluster 不是 DCS Cluster。两者的名称都包含“cluster”，但底层概念不同：DCS Cluster 是主机池，DCS Datastore Cluster 是 datastore 池。阅读平台文档或与 DCS 管理员交流时，请注意这一区别。

DCS Distributed Virtual Switch (DVS)

DCS 平台上的虚拟交换机，覆盖一个或多个 DCS Host。虚拟机 NIC 连接到 DCS Port Group，而 DCS Port Group 又属于某个 DCS DVS。该 provider 通过 `GET /service/sites/{siteId}/dvswitchs` 查询 DVS。

DCS Port Group

DCS DVS 上的网络配置项。每个端口组都包含 VLAN 配置、安全策略及其他网络设置。该 provider 使用父 DVS URI，通过 `GET {dvSwitchUri}/portgroups` 查询端口组，并通过名称（`portGroupName`）或 URN（`portGroupUrn`）引用它们。

虚拟机 NIC 必须恰好连接到一个 DCS Port Group。其他 NIC 需要引用其他端口组。

DCS Volume

DCS 平台上的持久磁盘。Volume 位于 DCS Datastore 上，通过 URN 标识，例如 `urn:sites:EC1C108C:volumes:9841`。当 `DCSIpHostnamePool.spec.pool[].persistentDisk[]` 声明了在虚拟机替换后仍需保留的磁盘时，该 provider 会通过 `POST /service/sites/{siteId}/volumes` 创建 Volume，并通过 `attachvol` 操作将生成的 Volume 挂载到虚拟机。

DCS Volume 在概念上不同于 Kubernetes `PersistentVolume`。DCS Volume 是在 hypervisor 层挂载的 DCS 平台存储对象；Kubernetes `PersistentVolume` 是表示可供 pod 使用的存储的 Kubernetes API 对象。该 provider 仅管理 DCS Volume；集群内的 `PersistentVolume` 资源由工作负载的 CSI 驱动程序单独管理。

DCS Cloud Credential

该 provider 用于调用 DCS REST API 的身份验证材料。该 provider 会向 `POST /service/session` 发送 `X-Auth-User` 和 `X-Auth-Key` 标头以获取会话令牌，并在后续请求中使用返回的 `X-Auth-Token` 标头。凭证存储在 Kubernetes `Secret` 中，并由 `DCSCluster.spec.credentialSecretRef` 引用。Secret 包含 `authUser`、`authKey`、`endpoint` 和 `site` 键。

Cluster API 资源映射

下表用于区分名称与 Cluster API 自定义资源相似的 DCS 平台概念。

DCS 平台概念	名称相似的 **Cluster API** 资源	两者为何不同
DCS Cluster	`DCSCluster` (CRD)	DCS Cluster 是 DCS 平台上的物理 `DCSCluster` 是 Cluster API 自定义 DCS 平台之上的 Kubernetes 集群的 DCS Cluster 可以承载一个或多个 K 拟机。
DCS Host	Kubernetes Node	DCS Host 是在 DCS 平台上注册的物 Kubernetes Node 是运行 `kubelet` 内的工作负载主机。该 provider 不会 DCS Host；Kubernetes Node 与其 Host 之间的关系由 DCS 调度程序决 义资源中表示。
DCS VM Template	`DCSMachineTemplate` (CRD)	DCS VM Template 是由 DCS 管理员 的 OS 镜像模板。`DCSMachineTemp` 名称并添加 Kubernetes 专用配置（ 网络）的 Cluster API 资源。`DCSMa` 包含镜像数据；它通过 `vmTemplat` Template。
DCS Volume	Kubernetes `PersistentVolume`	DCS Volume 是在 hypervisor 层挂载 象。Kubernetes `PersistentVolum` 使用的存储的 Kubernetes API 对象。 DCS Volume（由 `DCSIpHostnamePool.spec.pool[]` 驱动）；Kubernetes `PersistentV` 载的 CSI 驱动程序独立管理。
DCS Cloud Credential	Kubernetes `Secret`（由 `DCSCluster.spec.credentialSecretRef` 引用）	DCS Cloud Credential 是由 DCS 平 密钥对。它以 Kubernetes `Secret` provider，其中包含 `authUser`、 `endpoint` 和 `site` 键。

以下概念与任何 Cluster API 资源都不冲突，但由于名称中包含“cluster”，在内部容易混淆：

DCS 平台概念	容易混淆的概念	两者为何不同
DCS Datastore Cluster	DCS Cluster	DCS Datastore Cluster 是一组 DCS Datastore（存储）。DCS Cluster 是一组 DCS Host（计算）。两者都包含“cluster”一词，但属于 DCS 平台的不同层级。

存储概念：**Datastore**、**Datastore Cluster** 和 **Volume**

DCS 平台提供了三个由该 provider 分别引用的存储相关概念：

- **DCS Datastore** 是单个存储容器。虚拟机磁盘固定到一个 DCS Datastore。
- **DCS Datastore Cluster** 是一组 DCS Datastore。引用 DCS Datastore Cluster 的虚拟机磁盘会放置在其中一个有可用容量的成员 datastore 上；成员由 DCS 调度程序选择。
- **DCS Volume** 是位于 DCS Datastore（或 DCS Datastore Cluster 的成员 datastore）上的持久磁盘对象。当 `DCSIpHostnamePool` 声明持久磁盘时，该 provider 会创建 Volume 并将其挂载到虚拟机；这些 Volume 在虚拟机替换后仍会保留。

三者之间的关系：

- `DCSMachineTemplate` 在 `vmConfig.dcsMachineDiskSpec[]` 下声明一个或多个磁盘。每个磁盘绑定到特定的 `datastoreName` 或 `datastoreClusterName`。这些模板级磁盘会随虚拟机一同创建和销毁。
- `DCSIpHostnamePool` 条目可以声明一个或多个 `persistentDisk[]` 项。每一项绑定到特定的 `datastoreName` 或 `datastoreClusterName`，该 provider 会创建相应的 DCS Volume，使其在虚拟机替换后仍然保留。

跨集群可见性由底层存储后端决定。SAN、IPSAN、NAS 和 FusionStorage datastore 通常对多个 DCS Cluster 可见；本地磁盘 datastore 仅对单个 DCS Cluster 可见。此可见性决定单个 DCS VM Template 是否可以克隆到多个 DCS Cluster。

网络概念：**DVS**、**Port Group** 和 **IP** 标识

基于 DCS 的 Kubernetes 集群的网络配置分为两个层级：

- **DCS** 侧网络配置（第 **2** 层）：虚拟机 NIC 必须连接到 DCS Port Group，而该 Port Group 属于 DCS Distributed Virtual Switch。DVS 和 Port Group 是 DCS 平台对象，由 DCS 侧创建和管理。provider 通过名称（`dvSwitchName`、`portGroupName`）或 URN（`portGroupUrn`）引用它们。
- 集群侧网络标识（第 **3** 层）：分配给虚拟机的 IP 地址、子网掩码、网关、DNS 服务器和主机名不是 DCS 平台对象。它们在 `DCSIpHostnamePool.spec.pool[]` 条目中声明（这是由集群 operator 所拥有的 Cluster API 资源），并在引导期间通过 cloud-init 或 Ignition 写入虚拟机。

这种划分有两个实际影响：

- DCS 平台团队负责管理 DVS 和 Port Group 定义。集群 operator 负责管理 IP 和主机名规划。两者通过 `DCSIpHostnamePool` 资源进行协调，该资源引用 DCS 侧的 Port Group，同时声明集群侧的 IP 标识。
- provider 不会调用任何 DCS API 来分配或预留 IP 地址。集群 operator 编写 `DCSIpHostnamePool` 条目时必须避免地址冲突。

Cluster API 资源 `DCSIpHostnamePool` 和 DCS 平台的 Port Group 是不同的对象。DCS 平台没有原生的“IP 池”概念；`DCSIpHostnamePool` 资源是记录每台虚拟机网络标识的唯一位置。

字段与概念参考

下表将最常用的自定义资源字段映射到它们引用的 DCS 平台概念，以及这些字段的值来源。

`DCSCluster` 字段

字段	**DCS** 平台概念	值来源
`spec.site`	DCS Site	DCS 平台管理员提供的 Site ID。
`spec.credentialSecretRef`	DCS Cloud Credential	同一命名空间中的 Kubernetes `Secret` 名称，其中保存 DCS 用户、密钥、端点和站点信息。

`DCSMachineTemplate` / `DCSMachine` 字段

字段	DCS 平台概念
`spec.template.spec.vmTemplateName`	DCS VM Template
`spec.template.spec.location`（与 `type: folder` 配合使用）	DCS VM Folder
`spec.template.spec.location`（与 `type: cluster` 配合使用）	DCS Cluster
`spec.template.spec.vmConfig.dvSwitchName`	DCS DVS
`spec.template.spec.vmConfig.portGroupName`	DCS Port Group
`spec.template.spec.vmConfig.portGroupUrn`	DCS Port Group
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreName`	DCS Datastore

字段	DCS 平台概念
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	DCS Datastore Cluster
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreUrn`	DCS Datastore
`spec.template.spec.vmConfig.cdRomDatastoreName`	DCS Datastore
`spec.template.spec.vmConfig.cdRomDatastoreClusterName`	DCS Datastore Cluster

`DCSIpHostnamePool` 字段

字段	DCS 平台概念	值来源
`spec.pool[].ip` / `mask` / `gateway` / `dns` / `hostname`	集群侧网络标识（非 DCS 平台概念）	由集群 operator 声或 Ignition 写入虚拟
`spec.pool[].dvSwitchName`	DCS DVS	用于额外 NIC 的 D 设置）。未指定时[DVS。
`spec.pool[].portGroupName` / `portGroupUrn`	DCS Port Group	用于额外 NIC 的 P （按 IP 设置）。

字段	DCS 平台概念	值来源
`spec.pool[].persistentDisk[].datastoreName`	DCS Datastore	用于存放持久磁盘的名称。
`spec.pool[].persistentDisk[].datastoreClusterName`	DCS Datastore Cluster	用于创建持久磁盘的 Cluster 名称。不能 `datastoreName`
`spec.pool[].persistentDisk[]`（条目本身）	创建 DCS Volume	provider 通过 `POS` `/service/sites/{` 创建 DCS Volume `attachvol` 操作机。

要求

- 具有 API 访问权限的 DCS 平台
- 具有 DCS administrator 角色的用户，可以是：
  - `Interface interconnection user`（默认），或
  - `Domain user`（由 LDAP/AD 支持；需要由 DCS 管理员配置 DCS 域身份验证）。

  有关选择标准和 DCS 端要求，请参阅凭证用户类型。
- 包含 Alauda OS 镜像的虚拟机模板。使用集群上安装的 ACP 版本所发布的 Alauda OS 镜像；请参阅下载 Alauda OS。
- 使用由池管理的持久磁盘时，需要使用 `4.2.1` 或更高版本的 DCS VM 模板，因为安全关机和磁盘分离依赖 guest tools
- 具备跨主机访问能力的共享存储

文档

有关使用 DCS Provider 的详细说明，请参阅：

- 安装

- 基础设施资源
- 创建集群
- 管理节点
- 升级 Kubernetes
- API 参考

安装 Huawei DCS Provider

本文档介绍如何在 `global` 集群上安装 Huawei DCS 的 Infrastructure Provider 和 Cluster Bootstrap Provider。

目录

适用场景
前提条件
操作步骤
下载软件包
上架软件包
安装集群插件
验证安装
后续步骤

适用场景

在安装 `global` 集群后、在 Huawei DCS 上创建第一个业务集群之前完成此页面中的操作。无论 `global` 集群自身运行在哪个 IaaS 平台上，都必须完成这些操作。在灾难恢复（DR）对中，需要在两个 `global` 集群上完成这些操作，以便在故障转移后，备用集群能够协调相同的业务集群。将软件包上架到每个 `global` 集群。

这与安装全局集群中的 Provider 操作步骤不同。那些操作步骤会将 Provider 软件包上架到临时引导集群，以创建 `global` 集群本身。

前提条件

安装 Provider 前，请确保已具备以下条件：

- 访问 灵雀云容器平台 `global` 集群的权限
- 访问 Customer Portal 以下载插件的权限

操作步骤

1 下载软件包

要在 Huawei DCS 上创建集群，需要两个集群插件。从 Customer Portal 下载这两个插件。对于 ACP 4.5，它们列在 **Common ACP Extensions** 下；请参阅下载。

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Huawei DCS**

2 上架软件包

使用 `violet push` 将两个软件包上架到平台。有关命令及其参数，请参阅上架软件包。

3 安装集群插件

在 `global` 集群上安装两个集群插件。有关操作步骤，请参阅集群插件。

4 验证安装

安装 Provider 后，验证其是否已正确安装：

```
# Check if the DCS provider pods are running
kubectl get pods -n cpaas-system | grep dcs

# Verify CRDs are registered
kubectl get crd | grep -E '^dcs.*\.infrastructure\.cluster\.x-k8s\.io
$'

# Expected output should include:
# dcsclusters.infrastructure.cluster.x-k8s.io
# dcsmachines.infrastructure.cluster.x-k8s.io
# dcsmachinetemplates.infrastructure.cluster.x-k8s.io
# dcsiphostnamepools.infrastructure.cluster.x-k8s.io
```

后续步骤

安装 Provider 后，您可以继续执行以下操作：

- 准备基础设施资源
- 创建集群

Huawei DCS 的基础设施资源

目录

概述
云凭证
使用 Web UI
先决条件
创建云凭证
管理云凭证
使用 CLI
凭证用户类型
IP Pools
使用 Web UI
先决条件
创建 IP Pool
管理 IP Pools
使用 YAML
IP Pool 中的额外 NIC
IP Pool 中的持久磁盘
DCS 平台容量与放置
运行诊断片段
选择 datastore
主机放置和高可用性

控制平面跨主机高可用性
故障排除：VM 卡在 `creating`
Machine Templates
使用 Web UI
先决条件
创建 Machine Template
管理 Machine Templates
使用 YAML
高级：使用共享存储进行多集群部署
资源关系
后续步骤

概述

在 Huawei DCS 上创建集群之前，需要配置基础设施资源，包括云凭证、IP 池和计算机模板。

您可以通过 Web UI 或 YAML 清单管理基础设施资源。Web UI 提供带验证功能的引导式界面，而 YAML 则支持自动化操作。

在 Huawei DCS 上，IP 池还包含必须在 VM 替换后保留的磁盘，其中包括平台所需的 `/var/cpaas` 磁盘。

INFO

命名空间要求 所有基础设施资源都必须部署在 `cpaas-system` 命名空间中，以确保与平台正确集成为业务集群。

INFO

DCS 平台概念 如果您不熟悉 `DCS Cluster`、`DCS Host`、`DCS VM Folder` 或 `DCS VM Template` 等术语，或者不清楚这些术语与 `DCSCluster` 和 `DCSMachineTemplate` 自定义资源之间的关系，请先阅读Huawei DCS 概念和术语。

云凭证

云凭证存储集群操作所需的 DCS 平台访问信息。

使用 **Web UI**

先决条件

创建云凭证之前，请确认满足以下 DCS 平台要求：

用户配置：

- 用户类型：以下类型之一：
  - `Interface interconnection user` — 默认类型；现有行为。如果 DCS 门户中已配置互联型账户，请使用此类型。
  - `Domain user` — 由目录服务管理的 LDAP/AD 用户。要求 DCS 已由您的 **DCS** 管理员配置域身份验证，并且已在 **DCS** 门户中预先创建目录用户，同时按下文映射角色。
- 角色：必须为 `administrator`

密码策略（仅适用于 `Interface interconnection user`）：

导航到系统管理 → 权限管理 → 权限管理策略并确认：

- 策略：`Whether to modify the password of an interface interconnection user upon password resetting and first login`
- 值：必须设为否

如果设为 `Yes`，用户首次登录时将被强制修改密码，导致身份验证失败并使集群创建失败。

对于 `Domain user`，密码生命周期由 LDAP/AD 目录服务管理，而非由 DCS 门户策略管理。

创建云凭证

导航：集群 → 云凭证 → 创建云凭证 → 选择 Huawei DCS

表单字段：

字段	类型	必填	说明
名称	text	是	凭证的唯一标识符（1-63 个字符，仅限小写字母、数字和连字符）
显示名称	text	否	便于识别的自定义描述
DCS Endpoint	URL	是	DCS 平台 API 地址（必须以 `http://` 或 `https://` 开头）
用户名	text	是	DCS 平台 API 用户登录名
密码	password	是	DCS 平台 API 用户登录密码
站点	text	是	VM 模板所在的站点（所有资源必须位于同一站点）

验证规则：

- 名称必须为 1-63 个字符，只能包含小写字母、数字和连字符，并且必须以字母或数字开头和结尾
- DCS Endpoint 必须是有效 URL 格式，并以 `http://` 或 `https://` 开头

管理云凭证

查看凭证：导航到集群 → 云凭证，可查看所有已配置凭证的类型、创建时间和创建者。

更新凭证：点击凭证上的更新可修改显示名称。当前版本不支持更新密码（计划在未来版本中提供）。

删除凭证：点击删除以移除凭证，并在对话框中确认删除。

> **INFO**
>
> 域用户支持：此版本仅支持 **YAML**
>
> 当前 Web UI 只能创建 `Interface interconnection user` 凭证。若要创建 `Domain user` 凭证，请使用下方的 CLI 操作方式，并将 `userType` 设为 `domain`。

使用 CLI

在 `cpaas-system` 中创建 Secret，以存储 DCS 身份验证信息。

使用编辑器编写凭证文件，而不要在 shell 提示符下直接输入值。Shell 会将整条命令（包括密码）记录在历史记录中；而使用编辑器编写文件则不会留下此类副本。

dcs-credential.env

```
authUser=<auth-user>
authKey=<auth-key>
endpoint=https://<dcs-api-host>:7443
site=<dcs-site-id>
userType=interconnect
```

限制文件权限，从文件创建 Secret，然后删除该文件：

```
chmod 600 dcs-credential.env

DCS_SECRET_NAME="<auth-secret-name>"

kubectl create secret generic "${DCS_SECRET_NAME}" \
  --namespace cpaas-system \
  --from-env-file=dcs-credential.env

rm -f dcs-credential.env
```

以纯文本形式填写值 — `kubectl` 会在创建 Secret 时对其进行 base64 编码，因此无需单独执行编码步骤。每行的格式为 `key=value`。不要给值加引号：`kubectl` 会将引号字符视为实际内容，因此 `authKey="s3cret"` 会存储带引号的 `"s3cret"`。值中可以包含 `=`；每行中只有第一个 `=` 用于分隔键和值。

对于接口互联用户（默认类型），请省略 `userType` 行。对于域用户，请将其设为 `domain` — 请参阅凭证用户类型。

WARNING

请勿使用 `kubectl apply` 创建凭证 Secret。客户端应用会将所应用的清单（包括 DCS 密码）存储在 Secret 本身的 `kubectl.kubernetes.io/last-applied-configuration` 注解中。`kubectl create` 不会写入此类注解。无论值是以 `stringData` 形式还是 base64 `data` 形式编写，均适用此规则；base64 是一种编码方式，并不提供保护。

之后如需更改凭证，请对其应用补丁，而不要重新应用清单。出于同样的原因，请使用编辑器编写补丁文件：

dcs-credential-patch.yaml

```yaml
stringData:
  authKey: <new-auth-key>
```

```
chmod 600 dcs-credential-patch.yaml

kubectl patch secret "${DCS_SECRET_NAME}" --namespace cpaas-system \
  --type=merge --patch-file=dcs-credential-patch.yaml

rm -f dcs-credential-patch.yaml
```

参数说明：

键	说明
`authUser`	DCS 平台 API 用户登录名。
`authKey`	DCS 平台 API 用户登录密码。
`endpoint`	DCS 平台 API 地址，包含 `http` 或 `https` 方案。注意：DCS 平台的默认 API 端口为 7443（而非常见的 8443）。如果您的环境使用自定义端口，请向管理员确认。
`site`	DCS 站点 ID。Provider 会从此 Secret 中读取该值，并在 `DCSCluster.spec.site` 字段为空时将其写入该字段，因此在此处设置即可。仅当需要为单个集群覆盖 Secret 中的值时，才显式设置 `DCSCluster.spec.site`。

键	说明
`userType`	可选。允许的值为 `interconnect`（默认值；未设置该键或其值为空时使用）或 `domain`。匹配时不区分大小写（`Domain`、`DOMAIN` 和 `domain` 都映射到同一值）。任何其他值，包括旧版 DCS 魔数 `1` 和 `2`，都会导致配置错误并被拒绝，以确保 `domain` 用户不会在不知情的情况下作为 `interconnect` 用户进行身份验证。

凭证用户类型

DCS Provider 支持两种凭证用户类型，通过可选 Secret 键 `userType` 进行选择：

`userType` 的值	映射到的 **DCS** 用户	适用场景	**DCS** 端要求	密码生命周期
`interconnect`（默认值；等同于未设置该键或该键为空）	`Interface interconnection user`	现有部署；如果 DCS 中已配置互联账户，这是最简单的使用方式。	DCS 门户必须将策略“重置密码和首次登录时是否修改接口互联用户的密码”设为否。	由 DCS 门户策略管理。
`domain`	`Domain user`（LDAP/AD 用户）	安全审计要求使用 LDAP/AD 管理的凭证，或部署已统一使用目录服务账户的客户。	DCS 必须已由您的 **DCS** 管理员配置域身份验证，并且必须已在 **DCS** 门户中预先创建目录用户并为其分配 `administrator` 角色。	由 LDAP/AD 目录服务管理，而非由 DCS 门户策略管理。

兼容性：不含 `userType` 键的现有 Secret 仍会像以前一样进行身份验证；继续使用互联型凭证无需迁移。

UI 覆盖范围：当前 Web UI 表单不提供 `userType` 字段。若要使用 `domain` 值，请通过 YAML 配置 Secret。

IP Pools

IP 池定义集群节点的网络配置（IP 地址、子网掩码、网关、DNS）。每个池可以包含多个节点条目，每个节点可以有多个网络接口配置。

对于 DCS 多 NIC 集群，每个 IP 条目可以声明 `additionNic` 项。这些额外 NIC 会绑定到 IP 槽位，并在 Provider 创建新 VM 时应用。更新现有池中的 `additionNic` 不会为已运行的 VM 热添加 NIC。

对于 DCS 持久磁盘，每个 IP 条目还可以声明 `persistentDisk` 项。这些磁盘绑定到 IP 槽位，而不是 VM 生命周期，因此在滚动升级期间，可以从旧 VM 分离并重新挂载到替换 VM。对于平台要求的 `/var/cpaas` 磁盘，以及任何必须在删除并重新创建操作后保留的其他节点本地数据，请使用此机制。此 `persistentDisk` 声明仅支持通过 YAML 配置。

使用 Web UI

先决条件

- 已创建 Cloud Credential

> **INFO**
>
> IP Pool Web UI 仅涵盖标准 IP、主机名和网络设置，不提供 `DCSIpHostnamePool.spec.pool[].persistentDisk`。请通过 YAML 清单配置持久磁盘。

创建 IP Pool

导航路径：Clusters → Virtual Machine → IP Pools → Create IP Pool → Select Credential

表单结构：

IP Pool 表单由 **Pools** 列表组成。每个 Pool 表示一个节点，包含：

1. **Node IP**（必填，每个 Pool 恰好一个）
2. **Additional NIC IPs**（可选，每个 Pool 可有多个）

Node IP 字段：

字段	类型	必填	说明
IP	IP 地址	是	Kubernetes 节点的 IP 地址
Subnet Mask	CIDR 前缀长度	是	网络的子网掩码，以前缀长度形式输入，例如 `24`
Gateway	IP 地址	是	网关 IP 地址
DNS	IP 地址	否	DNS 服务器地址（多个地址以分号分隔）
Hostname	文本	否	虚拟机的主机名
Machine Name	文本	否	DCS 平台中的虚拟机名称
dvSwitch Name	下拉菜单	否	虚拟交换机名称（来自 DCS 平台）
Port Group Name	下拉菜单	否	端口组名称（来自 DCS 平台）

Additional NIC IPs 字段：

字段	类型	必填	说明
IP	IP 地址	是	非节点 IP 地址（例如，存储网络）
Subnet Mask	CIDR 前缀长度	是	网络的子网掩码，以前缀长度形式输入，例如 `24`
Gateway	IP 地址	是	网关 IP 地址
DNS	IP 地址	否	DNS 服务器地址（多个地址以分号分隔）
dvSwitch Name	下拉菜单	是	虚拟交换机名称（来自 DCS 平台）
Port Group Name	下拉菜单	是	端口组名称（来自 DCS 平台）

验证规则：

- 同一个 IP Pool 中的 IP 地址必须唯一
- IP 地址必须是有效的 IPv4 格式
- 子网掩码必须以有效的 CIDR 前缀长度输入，例如 `24`。请勿包含 `/`，也不要使用点分十进制格式，例如 `255.255.255.0`。
- IP 地址必须位于已配置的子网范围内
- 网关必须是子网范围内有效的 IPv4 地址

提示：

- 至少需要一个节点条目
- 每个节点必须且只能配置一个 Node IP
- 多 NIC 场景下可选配置 `additionNic[]` 条目，例如用于分离存储网络

管理 **IP Pools**

查看 **Pools**：导航至 Clusters → Virtual Machine → IP Pools，可查看所有已配置的池、其节点 IP 和创建时间。

更新 **Pools**：点击 **Update**，即可添加或移除节点条目并修改网络配置。

删除 **Pools**：点击 **Delete** 删除池，并在对话框中确认删除。

使用 **YAML**

创建 `DCSIpHostnamePool` 资源：

dcs-ippool.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <iphostname-pool-name>
  namespace: cpaas-system
spec:
  pool:
  - ip: "<ip-1>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-1>"
    machineName: "<machine-name-1>"
    additionNic:
    - ip: "<additional-ip-1>"
      mask: "<additional-cidr-prefix-length>"
      gateway: "<additional-gateway>"
      dns: "<additional-dns>"
      dvSwitchName: "<additional-dvs>"
      portGroupName: "<additional-port-group>"
  - ip: "<ip-2>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-2>"
    machineName: "<machine-name-2>"
  - ip: "<ip-3>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-3>"
    machineName: "<machine-name-3>"
```

参数说明：

参数	类型	说明	必填
`.spec.pool[].ip`	string	要创建的虚拟机的 IP 地址	是

参数	类型	说明	必填
`.spec.pool[].mask`	string	CIDR 前缀长度格式的子网掩码，不含 `/`，例如 `24`	是
`.spec.pool[].gateway`	string	网关 IP 地址	是
`.spec.pool[].dns`	string	DNS 服务器 IP 地址（使用 `;` 分隔多个服务器）	否
`.spec.pool[].machineName`	string	DCS 平台中的虚拟机名称	否
`.spec.pool[].hostname`	string	虚拟机的主机名	否
`.spec.pool[].additionNic[]`	[]object	从此 IP 槽位创建的 VM 的额外 NIC。Provider 仅在创建新 VM 时应用这些配置。	否
`.spec.pool[].additionNic[].ip`	string	用于生成的客户机网络配置的额外 NIC IP 地址。此地址不是 Kubernetes 节点 IP。	建议填写
`.spec.pool[].additionNic[].mask`	string	CIDR 前缀长度格式的额外 NIC 子网掩码，不含 `/`，例如 `24`。	建议填写

参数	类型	说明	必填
`.spec.pool[].additionNic[].gateway`	string	额外 NIC 网络的网关。设置此值前请确认路由行为。	否
`.spec.pool[].additionNic[].dns`	string	额外 NIC 网络的 DNS 服务器 IP 地址；需要多个值时以分号分隔。	否
`.spec.pool[].additionNic[].dvSwitchName`	string	用于解析额外 NIC 的 Port Group 的 DCS 分布式虚拟交换机。	是*
`.spec.pool[].additionNic[].portGroupName`	string	用于连接额外 NIC 的 DCS Port Group。	是*

*标记为 `Yes*` 的字段是 Provider 解析目标网络并连接额外 NIC 所必需的。标记为 `Recommended*` 的字段是生成静态客户机网络配置所需的。CRD 不会强制要求这些字段。

> **WARNING**
>
> 您必须配置的机器信息数量大于或等于计划部署的节点数量。条目不足将导致节点无法部署。

IP Pool 中的额外 NIC

在 `DCSIpHostnamePool.spec.pool[].additionNic[]` 中声明额外 NIC。

当节点需要辅助网络（例如存储、管理或应用隔离网络）时，请使用此字段。除非您有意计划更改 Node IP，否则请将主要 `.spec.pool[].ip` 保留为 Kubernetes Node IP。

生效方式：

- 当新的 `DCSMachine` 声明某个槽位时，Provider 会将 Pool 槽位中的 `additionNic[]` 复制到 `DCSMachine.status.additionalNic`。
- Provider 克隆 DCS VM 时会添加这些 NIC。
- 客户机 OS 会在首次启动期间通过生成的 NetworkManager 配置来配置这些接口。
- 后续编辑 Pool 不会热更新现有 VM。

网络边界：

- DCS Provider 不会在 DCS 平台上预留额外 NIC 的 IP 地址。应用 Pool 前请避免 IP 冲突。
- 引用的 DVS 和 Port Group 必须已存在，且目标 DCS 主机必须能够访问它们。
- 每个额外 NIC 都要提供 `dvSwitchName` 和 `portGroupName`。Provider 会在克隆 VM 前使用这些值解析 Port Group URN。
- Provider 仅对 IP、掩码、网关、DNS、DVS 和 Port Group 进行建模。路由度量值、静态路由和 NIC 角色必须通过 MCP 或客户机 OS 配置处理。
- 如果额外 NIC 配置了网关，请验证它不会意外取代主要 NIC 的默认路由。

IP Pool 中的持久磁盘

在 `DCSIpHostnamePool.spec.pool[].persistentDisk` 中声明升级期间保留的磁盘，而不是在 `DCSMachineTemplate` 中声明。

这些持久磁盘仅支持通过 YAML 声明。

- 使用 IP 条目将每个持久磁盘绑定到固定的 `(ip, slot)` 标识。
- 对平台要求的 `/var/cpaas` 磁盘使用此模型。
- 将 `DCSMachineTemplate` 用于系统磁盘和任何可能随 VM 一同重新创建的模板本地磁盘。
- 替换 VM 启动时，客户机磁盘设置脚本会检查是否存在现有文件系统。如果已存在，则跳过格式化并直接挂载磁盘。
- Pool 管理的持久磁盘需要逐个替换。使用此功能时，请保留 `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge = 0` 和 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`。

如果计划使用持久磁盘，请确保 DCS VM 模板为 `4.2.1` 或更高版本，因为安全关机和磁盘分离依赖于客户机 OS 中的客户机工具。

示例：

dcs-ippool-with-persistent-disk.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <iphostname-pool-name>
  namespace: cpaas-system
spec:
  pool:
  - ip: "<ip-1>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-1>"
    machineName: "<machine-name-1>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
      mountOptions:
      - defaults
```

持久磁盘字段说明：

字段	说明
`slot`	IP 槽位中的磁盘位置。它决定挂载顺序、客户机设备命名，以及卷声明和重新挂载时使用的序号。
`quantityGB`	磁盘大小，单位为 GB。
`datastoreName` / `datastoreClusterName`	为持久磁盘指定且仅指定一个存储目标。
`isThin`	可选。控制新建持久卷的 DCS 精简置备。如果省略，Provider 不会发送 `isThin`，DCS 将使用平台默认值。

字段	说明
`path`	客户机 OS 中的挂载路径。`/var/cpaas` 是平台要求的路径。
`format`	磁盘首次初始化时使用的文件系统。如果替换 VM 检测到现有文件系统，则跳过格式化。
`options`	仅在首次格式化磁盘时应用的 `mkfs` 选项。
`mountOptions`	挂载磁盘时应用的挂载选项。
`pciType`	可选的 PCI 磁盘类型。如果省略，控制器将使用实现中的默认值。

更新规则：

- 可以向现有 IP 槽位追加新的 `persistentDisk` 条目。控制器会在 DCS 端将新添加的磁盘连接到正在运行的 VM，但不会在客户机 OS 中格式化或挂载该磁盘。只有在 VM 被替换且替换 VM 在引导期间运行生成的磁盘设置后，客户机中的格式化和挂载才会生效。
- 控制器会将新持久卷创建为独立的持久普通卷。复用现有卷时，只要求该卷为独立且持久，不要求特定的 DCS 卷类型。
- 请勿从 spec 中删除现有的 `persistentDisk` 条目。Webhook 会拒绝此类删除。
- 创建后，将 `format`、`options` 和 `pciType` 视为不可变。
- 将 `quantityGB` 和 datastore 更改视为可能影响滚动更新的更改。当池带有集群标签时，Webhook 会尽力进行平台验证。`isThin` 仅在创建时生效：更改它不会验证、协调或转换现有卷，只会影响之后创建的持久卷。
- 将 `path` 和 `mountOptions` 的更改视为客户机端更改，并在滚动更新期间通过替换机器生效。

DCS 平台容量与放置

在编写 `DCSMachineTemplate` 资源之前，请确认目标 DCS 站点、datastore 和计算集群有足够的可用容量来承载计划中的 VM，并确认 DCS 侧调度策略能够将新 VM 放置到可用主机上。

跳过这些检查是 VM 卡在 `creating` 状态且 DCS API 未返回任何错误的最常见原因。

运行诊断片段

以下容量和放置检查使用只读 DCS REST 调用。运行任何片段之前，请在 shell 中设置以下环境变量：

变量	含义	获取方式
`DCS_BASE`	DCS REST API 基础 URL，包含协议和端口。DCS REST API 发布在端口 7443 上（与 8443 Web 门户分开）。	向 DCS 平台管理员询问 API 端点。相同的值存储在云凭证 `Secret` 的 `endpoint` 键下。
`SITE`	DCS 站点 ID（大写字母数字，例如 `EC1C108C`）。	存储在云凭证 `Secret` 的 `site` 键下，也可通过 `DCSCluster.spec.site` 查看。
`TOKEN`	由 `POST /service/session` 返回的短期 `X-Auth-Token`。令牌在空闲约 30 分钟后失效。	运行下方的会话命令。DCS 用户名和密钥存储在云凭证 `Secret` 的 `authUser` 和 `authKey` 下。

在诊断会话开始时设置这三个变量：

```
# Replace the placeholders with values appropriate for your environment.
export DCS_BASE='https://<dcs-api-host>:7443'
export SITE='<site-id>'

# Read the user and key from the cloud-credential Secret without printing
them.
DCS_USER="$(kubectl -n cpaas-system get secret <credential-secret-name> -
o jsonpath='{.data.authUser}' | base64 -d)"
DCS_KEY="$(kubectl -n cpaas-system get secret <credential-secret-name> -o
jsonpath='{.data.authKey}' | base64 -d)"

# Acquire a session token. The response header X-AUTH-TOKEN carries the v
alue used by subsequent calls.
HEADERS="$(mktemp)"
curl -k -sS -D "$HEADERS" -o /dev/null -X POST "${DCS_BASE}/service/sessi
on" \
  -H "X-Auth-User: $DCS_USER" \
  -H "X-Auth-Key: $DCS_KEY" \
  -H 'X-ENCRYPT-ALGORITHM: 1' \
  -H 'X-Auth-UserType: 2' \
  -H 'version: 8.1'
export TOKEN="$(grep -i '^X-AUTH-TOKEN:' "$HEADERS" | tail -n1 | cut -d'
' -f2- | tr -d '\r\n ')"
rm -f "$HEADERS"
unset DCS_USER DCS_KEY
```

WARNING

令牌处理 请勿将令牌记录到文件或 shell 历史记录中。会话令牌拥有与底层 DCS 用户相同的权限，该用户通常是管理员。

在本页后续的每个诊断片段中复用 `$DCS_BASE`、`$SITE` 和 `$TOKEN`。当 API 开始返回 `errorCode: 10000002`（会话已过期或尚未登录）时，重新运行 `POST /service/session` 命令以刷新令牌。

选择 **datastore**

每个 `DCSMachineDiskSpec` 条目都会通过两个互斥字段之一将磁盘绑定到 DCS datastore：

- `datastoreName` — 将磁盘固定到特定 datastore。即使相邻的 datastore 有可用空间，DCS 调度器也不会检查它们。这是最严格的绑定方式。
- `datastoreClusterName` — 将磁盘固定到 datastore 集群（DCS 侧的一组 datastore）。DCS 调度器会从该集群中选择一个满足容量要求的特定 datastore。如果平台提供 datastore 集群，这是更安全的默认选项。

只要 DCS 管理员配置了 datastore 集群，就优先使用 `datastoreClusterName`，这样即使某个 datastore 即将用满，只要相邻 datastore 仍有可用空间，也不会阻塞部署。

容量预检：

应用新的 `DCSMachineTemplate` 之前，请列出已有的 DCS datastore，并确认目标 datastore 有足够的可用容量来满足 `system disk + ∑(spec disk size) × replica count` 的最坏情况需求。在可访问 DCS API 的环境中，只读列表查询如下所示：

```
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/datastores" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1' \
  | python3 -c "
import json,sys
for ds in json.load(sys.stdin).get('datastores',[]) or []:
  print(f\"{ds.get('name','?'):24} cap={ds.get('capacityGB','?')}GB used=
{ds.get('usedSizeGB','?')}GB free={ds.get('freeSizeGB','?')}GB state={ds.
get('status','?')}\")"
```

每对 `DCSMachineTemplate` 控制平面节点和工作节点通常需要系统磁盘（模板默认值，通常约为 80 GB）以及 spec 中声明的显式数据磁盘，因此单个带有 `etcd 10G + kubelet 100G + containerd 100G + /var/cpaas 100G` 的控制平面节点在计入模板的系统磁盘之前，大约会占用 390 GB。当 datastore 的使用率超过约 70% 时，这是应选择其他 datastore 的明确信号——即使绝对可用空间看起来很多，碎片和预留空间也可能导致新 VM 放置停滞。

主机放置和高可用性

DCS 平台上的主机放置由 DCS 调度器执行。DCS Provider 不提供通过 `DCSMachineTemplate` 将虚拟机固定到特定 DCS 主机的方式。因此，从同一模板克隆的虚拟机可能会落在同一台物理主机上。

由此会产生两个影响：

- 容量热点：如果 DCS 集群的主机负载不均（例如，一台主机的 CPU 超额分配严重，而另一台主机大部分处于空闲状态），DCS 调度器可能会拒绝在超额分配的主机上放置新 VM。部署任务可能会停滞，直到 DCS 侧重新平衡负载或启用 DRS。使用下方片段确认主机负载是否为原因。
- 控制平面分散取决于 **DRS**：机器模板不会将控制平面虚拟机固定到特定主机。对于高可用性部署，请按照下文控制平面跨主机高可用性中的说明启用由 Provider 管理的 DRS 互斥。

应用新模板之前，请检查当前主机负载：

```
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/hosts" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1' \
  | python3 -c "
import json,sys
for h in json.load(sys.stdin).get('hosts',[]) or []:
  print(f\"{h.get('name','?'):10} status={h.get('status','?'):10} cpuUse
d/total={h.get('cpuUsedCores','?')}/{h.get('cpuTotalCores','?')}c memUsed
={h.get('memUsedSizeMB','?')}MB cpuMuxRatio={h.get('cpuMuxRatio','?')}
\")"
```

如果集群中每台主机的 `cpuMuxRatio`（CPU 分配量 / 物理核心数）都高于约 3 倍，则表明集群已饱和，在 DCS 侧重新平衡负载之前，很可能会拒绝新的放置请求。

控制平面跨主机高可用性

DCS Provider 不实现主机固定或 Cluster API `FailureDomain` 分散。相反，启用 `DCSCluster.spec.controlPlaneHA` 后，它可以为控制平面虚拟机维护由 Provider 管理的 DCS DRS 规则。

控制平面 VM 创建完成且其 VM URN 已知后，Provider 会创建并更新一条 DRS `ruleType=2` 互斥规则。该规则会列出一个业务集群当前的控制平面 VM，并要求 DCS 调度器将它们放置在不同的物理主机上。放置决策的执行以及任何运行时迁移仍由 DCS 负责。

Provider 不会调用 DCS Run DRS API，也不会应用 DRS 建议。如果规则维护完成后，`ControlPlaneHAReady` 仍为 `False`，且原因是 `ControlPlaneHAPending`，则表示 DCS 调度器尚未完成放置调整。请等待 DCS 侧调度，或使用 DCS 平台操作流程运行 DRS 并应用生成的建议。

启用此功能之前，请确认满足以下 DCS 侧要求：

- 目标 DCS 计算集群已启用 DRS。
- 目标 DCS 计算集群有足够健康的主机和容量。三个副本的控制平面通常至少需要三台可用主机。
- 所选 datastore 对所有可能运行控制平面 VM 的主机均可见。
- CD-ROM 设备已挂载等 DCS 侧限制不会阻止 DRS 运行时迁移。

编写集群清单时，在 `DCSCluster.spec.controlPlaneHA` 中启用此功能。清单示例和需要关注的状态字段，请参阅配置 DCSCluster 和验证控制平面 HA。

创建集群后，检查 `DCSCluster` 上的条件和状态：

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHARead
y")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'

kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.status.controlPlaneHA}{"\n"}'
```

`ControlPlaneHAReady=True` 表示由 Provider 管理的 DRS 规则已存在，并且 Provider 已观察到当前控制平面 VM 分布在不同的 DCS 主机上。`ControlPlaneHAReady=False` 且原因为 `ControlPlaneHAPending`，通常表示控制器正在等待 VM URN、等待足够的成员，或等待 DCS 侧完成放置调整。`ControlPlaneHAReady=False` 且原因为 `ControlPlaneHAFailed`，表示 Provider 无法查询、验证、创建、更新、删除或检查 DRS 规则。查看条件消息后，再检查控制器日志。

故障排除：VM 卡在 `creating`

如果 `DCSMachine` 一直处于 `Provisioning`，且已设置 `status.cdRomFile`（说明 DCS API 身份验证和 ISO 上传均已成功），但 DCS 侧 VM 始终无法离开 `status=creating`，也始终未获得 `hostName`，则故障几乎总是出在 DCS 放置层，而不是控制器中。

按以下顺序进行诊断：

1. 确认这是放置停滞，而不是启动缓慢。查询 DCS 上的 VM 和与之匹配的部署任务：

```
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/vms?name=${VM_NAME}&limi
t=10" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1'
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/tasks?status=running&lim
it=50" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1'
```

放置停滞的症状包括：`status=creating`、`hostName=`（为空 / 未调度）、`disks` 数组为空或没有变化，以及部署任务在没有错误的情况下，超过约 5 分钟停留在相同的 `progress` 值。

2. 检查 **datastore** 可用空间和主机负载，使用选择 datastore和主机放置要求中的片段。datastore 使用率超过 70%，或所有主机的 `cpuMuxRatio` 都很高，是最常见的原因。
3. 在 **DCS** 侧重新平衡负载。如果单台主机已饱和，而其他主机仍有可用容量，请在重试前联系 DCS 平台团队重新平衡负载（手动 `Migrate` 或通过 DRS 重新平衡）。DCS Provider 不提供按模板固定主机的功能，因此此补救工作由 DCS 平台团队负责。
4. 如果 **DCS** 侧部署任务确实陷入死锁（可以取消，但对 ACP 驱动的 `DeleteVm` 无响应），请联系拥有门户管理员凭证的 DCS 管理员升级处理并取消任务。7443 REST API 不会向非管理员用户提供任务取消端点，因此此恢复流程由 DCS 平台团队负责，而非 ACP。
5. 注意序列化带来的副作用。在启用了全局 DeployVM 锁的环境中，单个卡住的任务可能导致所有新部署都停滞在相同的进度百分比。解决最初卡住的任务通常即可解除队列阻塞。

如果某个 `DCSMachine` 的状态条件显示 `InfrastructureReady=False reason=WaitingForInfrastructure` 已超过 10 分钟，并伴有上述症状，应将其视为 DCS 平台事件，而不是 Cluster API 或 DCS Provider 回归问题。

Machine Templates

Machine template 用于定义集群节点的虚拟机规格（VM 模板、CPU、内存、磁盘、网络）。每个 machine template 都有一个用于确定其用途的 **Type**：

- **Control Plane**：用于控制平面节点
- **Worker Node**：用于工作节点

使用 **Web UI**

先决条件

- 已创建 IP Pool
- 已在 DCS 平台上使用 Alauda OS 镜像创建 VM Template
- 已将 ConfigMap YAML 应用到 global 集群

VM Template 和 **ConfigMap**：

每个 Alauda OS 版本都包含一个将 VM 模板映射到 Kubernetes 版本的 ConfigMap YAML。创建 machine template 前，请先应用此 YAML：

```yaml
apiVersion: v1
data:
  corednsTag: 1.12.4-v4.2.3
  etcdTag: v3.5.21-251117
  kubernetesVersion: v1.33.6
  vmImageVersion: alaudaos-42.m55.202512081125-0.x86_64
kind: ConfigMap
metadata:
  labels:
    cpaas.io/dcs-vm-template: <vm-template-name>
    cpaas.io/distribution-version: v4.2.0
    cpaas.io/kubernetes-version: v1.33
  name: 420-dcs-vm-template
  namespace: cpaas-system
```

重要：`cpaas.io/dcs-vm-template` 标签值必须与 DCS 平台上的 VM 模板名称一致。

创建 **Machine Template**

导航路径：集群 → 虚拟机 → Machine Templates → 创建 Machine Template → 选择凭证

表单字段：

字段	类型	必填	描述
Name	文本	是	模板的唯一标识符（1-63 个字符，仅限小写字母、数字和连字符）
Type	下拉菜单	是	Control Plane 或 Worker Node
VM Template Name	下拉菜单	是	来自 ConfigMap，显示 OS 版本和 Kubernetes 版本
Location	下拉菜单	否	DCS VM Folder，用于在 DCS 平台上对克隆的虚拟机进行分组。VM Folder 的定义请参阅 Huawei DCS Concepts and Terminology。必须先在 DCS 平台上创建该文件夹。
Specs	-	是	CPU 和内存规格
Specs.CPU	数字	是	CPU 核数（整数）
Specs.Mem	数字	是	内存大小（MB，在列表视图中以 GB 显示）
Disk	-	是	磁盘配置（见下文）
IP Pool	下拉菜单	是	引用现有 IP Pool

磁盘配置：

磁盘配置因模板类型而异。

控制平面所需磁盘：

挂载路径	默认大小（**GB**）	可删除
系统卷	（模板默认值）	否
`/var/lib/kubelet`	100	否
`/var/lib/containerd`	100	否

挂载路径	默认大小（**GB**）	可删除
`/var/lib/etcd`	10	否

工作节点所需磁盘：

挂载路径	默认大小（**GB**）	可删除
系统卷	（模板默认值）	否
`/var/lib/kubelet`	100	否
`/var/lib/containerd`	100	否

可以添加其他磁盘，但必须保留上述所有必需磁盘。

INFO

平台必需的持久磁盘

平台仍然需要 `/var/cpaas`，但不再将其记录为 `DCSMachineTemplate` 磁盘。请在对应的 `DCSIpHostnamePool.spec.pool[].persistentDisk` 条目中配置它，使其在 VM 替换后仍能保留。

磁盘字段说明：

字段	类型	描述
Mount Path	文本	磁盘挂载目录路径
Disk Size	数字（GB）	磁盘大小
Datastore	下拉菜单	类型：ClusterName 或 Name，然后从 DCS 平台中选择

VM Template 选择提示：

TIP

如果多个 VM 模板具有相同的 Kubernetes 版本，请选择 OS 版本最新的模板，以便获得最新的安全更新和系统改进。

管理 **Machine Templates**

查看模板：导航至集群 → 虚拟机 → Machine Templates，即可查看所有模板及其 VM Template Name、Location、Specs 和 IP Pool。

更新模板：单击 更新 以修改规格。请注意，创建后无法更改 Name 字段。

删除模板：单击 删除 以移除模板。在对话框中确认删除。

使用 **YAML**

创建 `DCSMachineTemplate` 资源：

dcs-machinetemplate.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: <machine-template-name>
  namespace: cpaas-system
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      location:
        type: folder
        name: <folder-name>
      vmConfig:
        dvSwitchName: <dv-switch-name> # Optional
        portGroupName: <port-group-name> # Optional
        dcsMachineCpuSpec:
          quantity: <cpu-cores>
        dcsMachineMemorySpec: # MB
          quantity: <memory-mb>
        dcsMachineDiskSpec: # GB
        - quantity: 0
          datastoreClusterName: <datastore-cluster-name>
          systemVolume: true
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/kubelet
          format: xfs
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/containerd
          format: xfs
        - quantity: 10
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/etcd
          format: xfs
      ipHostPoolRef:
        name: <iphostname-pool-name>
```

参数说明：

参数	类型
`.spec.template.spec.vmTemplateName`	string
`.spec.template.spec.location`	object
`.spec.template.spec.location.type`	string
`.spec.template.spec.location.name`	string
`.spec.template.spec.vmConfig`	object
`.spec.template.spec.vmConfig.dvSwitchName`	string
`.spec.template.spec.vmConfig.portGroupName`	string
`.spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec`	[]object

参数	类型
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].systemVolume`	bool
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].path`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].format`	string
`.spec.template.spec.ipHostPoolRef.name`	string

*指定父对象时必填

高级：使用共享存储进行多集群部署

默认工作流假设 DCS VM Template（`vmTemplateName`）与克隆的虚拟机位于同一个 DCS Cluster。对于大多数部署，此假设是正确的，`spec.template.spec.location` 字段仅用于设置 VM Folder 以便进行组织分组。

跨多个 DCS Cluster 部署，并希望从单个共享 DCS VM Template 克隆到不同目标 DCS Cluster 的场景，需要满足以下条件：

- 存放 DCS VM Template 的 DCS Datastore 必须对目标 DCS Cluster 可见。配置为支持跨集群访问的 SAN、IPSAN、NAS 或 FusionStorage datastore 符合此条件；由主机本地磁盘提供支持的 datastore 则不符合。
- `vmConfig` 引用的 DCS 侧网络资源（分布式虚拟交换机和端口组）也必须对目标 DCS Cluster 可见。

满足这两个条件后，可以通过设置 `spec.template.spec.location.type: cluster` 和 `spec.template.spec.location.name: <target-cluster-name>`，而不是使用默认的 `type: folder` 形式，将克隆的虚拟机放置在目标 DCS Cluster 下。如果 datastore 未在多个 DCS Cluster 之间共享，克隆任务将在 DCS 平台侧失败。建议的替代方案是在每个目标 DCS Cluster 中分别准备一个 DCS VM Template，以便每个 `DCSMachineTemplate` 都引用一个已位于虚拟机运行所在集群中的模板。

使用此配置前，请向 DCS 平台管理员确认 datastore 在跨集群环境中的可见性。

> **WARNING**
>
> 存储要求
>
> **Datastore** 跨主机访问 datastore 集群（`datastoreClusterName`）必须支持在 DCS 平台的所有物理机之间进行跨主机访问。如果 datastore 仅在特定主机上可用，当 DCS 平台尝试将 VM 调度到其他主机时，VM 创建将失败。
>
> **Ignition** 的共享存储 如果 datastore 不支持直接上传文件（Ignition 配置所必需），则必须提供支持多主机挂载的共享存储解决方案（例如 NFS）。
>
> 磁盘配置规则 可以添加自定义模板磁盘，但必须根据节点角色保留必需磁盘：
>
> - 控制平面：`systemVolume`、`/var/lib/kubelet`、`/var/lib/containerd`、`/var/lib/etcd`
> - 工作节点：`systemVolume`、`/var/lib/kubelet`、`/var/lib/containerd` 在 IP Pool 中将 `/var/cpaas` 配置为持久磁盘。

资源关系

基础设施资源具有以下依赖关系：

```
Cloud Credential
    ↓
IP Pool
    (network settings + persistent disk declarations)
    ↓
Machine Template → references IP Pool
    ↓
Cluster Creation
```

资源复用性：

- 一个 Cloud Credential 可供多个集群使用
- 可以为不同的网络网段创建多个 IP Pool
- 可以为不同的节点类型和规格创建多个 Machine Template

后续步骤

配置基础设施资源后：

- 创建集群
- 管理节点池
- 升级集群

在 Huawei DCS 上创建集群

本文档介绍如何在 Huawei DCS 平台上创建 Kubernetes 集群。可以通过清单使用基于 YAML 的集群创建方式。如果已安装 Fleet Essentials，也可以通过 Web UI 创建集群。`persistentDisk` 在 `DCSIpHostnamePool` 上的声明仅支持通过 YAML 配置，在 Web UI 中不可用。对于 ACP `v4.4` 或更高版本，`Self-built VIP` 也支持通过 YAML 工作流进行配置。

> **INFO**
>
> Web UI 提供带验证功能的引导式工作流，而 YAML 则提供更灵活的自动化能力。

目录

- 先决条件
  1. 基础设施资源
  2. 安装所需插件
  3. 准备虚拟机模板
  4. 网络连通性
  5. LoadBalancer 配置
  6. 业务集群镜像仓库
  7. 节点标签和污点
- 使用 Web UI
  - 创建工作流
  - 步骤 1：基本信息

步骤 2：控制平面节点池
步骤 3：业务节点池
步骤 4：网络
步骤 5：审核
使用 YAML
集群创建工作流
配置工作流
解析占位符值
Magic-Token 占位符
网络规划和负载均衡器
配置 KubeadmControlPlane
配置 DCSCluster
自建 VIP 的行为和限制
验证自建 VIP
单控制平面（无外部 LB）布局
配置 Cluster
部署工作节点
步骤 1：配置 IP-Hostname Pool
步骤 2：配置 Machine Template
由 IP 池管理的持久磁盘
由 IP 池管理的其他 NIC
步骤 3：配置 Bootstrap Template
步骤 4：配置 Machine Deployment
集群验证
使用控制台
使用 kubectl
验证控制平面 HA
预期结果
附录
完整的 KubeadmControlPlane 配置
加密 Provider 配置的交付方式
后续步骤

故障排除

先决条件

创建集群前，请确保满足以下所有先决条件：

1. 基础设施资源

创建集群前，请配置以下基础设施资源：

- Cloud Credential - DCS 平台访问信息
- IP Pool - 集群节点、额外 NIC 以及任何 IP 槽位持久磁盘（例如 `/var/cpaas`）的网络配置
- Machine Template - 控制平面和工作节点的 VM 规格，不包括由池管理的持久磁盘

有关详细配置说明，请参阅Huawei DCS 基础设施资源。

2. 安装所需插件

在 Huawei DCS 上创建首个业务集群之前，请先将以下插件上传并安装到 `global` 集群上。无论 `global` 集群本身运行在哪个 IaaS 平台上，均适用此要求：

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Huawei DCS**

有关操作步骤，请参阅安装 Provider。

3. 准备虚拟机模板

安装 Kubernetes 时，必须：

- 将 Alauda OS 镜像上传到 DCS 平台。使用为集群上已安装的 ACP 版本发布的 Alauda OS 镜像；请参阅下载 Alauda OS。
- 基于此镜像创建虚拟机模板

- 确保模板包含所有必需的 Kubernetes 组件
- 如果计划使用持久磁盘，请使用 `4.2.1` 或更高版本的 DCS VM 模板，因为安全关机和磁盘卸载依赖客户机工具
- 对于依赖池管理的持久磁盘的集群，请使用逐个替换方式。在控制平面和工作节点池上保留 `maxSurge: 0`。

有关每个 VM 镜像中包含的 Kubernetes 组件的详细信息，请参阅 OS 支持矩阵。

4. 网络连通性

global 集群节点必须能够访问 DCS 平台上的两个不同目标：

来源	目标	端口	用途
global 集群节点	DCS VRM 虚拟 IP	TCP/7443	DCS REST API。涵盖集群生命周期调用和文件上传的第一步（`applyUpload`）。
global 集群节点	DCS 物理主机 MGMT IP（可能接收克隆的每台主机）	DCS 返回的端口；通常为 TCP/8443；请向 DCS 管理员确认	上传 Ignition ISO 文件流。Provider 会将文件流式传输到 `applyUpload` 响应返回的 URL。

文件上传分为两个步骤：Provider 先通过 VRM 虚拟 IP（TCP/7443）调用 `applyUpload`；DCS 平台随后返回一个 URL，该 URL 指向特定物理主机的管理 IP 和端口；然后 Provider 将文件流式传输到此 URL。在创建集群之前，必须确保从端到端都能访问这两个目标。

如果 global 集群节点采用多 NIC 布局（例如，一个 NIC 位于 ACP 集群网络，另一个 NIC 位于部署了 DCS 的客户管理网络），请确保这两个目标——VRM 虚拟 IP 和每台 DCS 物理主机的 MGMT IP——都能通过相应的 NIC 路由访问。

要求：创建和管理集群时，必须能够连接到这两个目标。

5. LoadBalancer 配置

创建集群前，请配置一个稳定的 Kubernetes API 端点。该端点会分发或转发控制平面访问所需的 API 服务器流量。

模式	版本	管理方	适用场景
外部 LoadBalancer	所有受支持的版本	在创建集群前预配并维护负载均衡器。	适用于环境中已有硬件或软件负载均衡器的情况。
`Self-built VIP`	ACP `v4.4+`	DCS Provider 创建并协调由 Provider 管理的 `alive` `ModuleInfo`。	适用于需要高可用控制平面端点但没有外部负载均衡器的情况。
单控制平面直连端点	所有受支持的版本	不会创建负载均衡器。API 端点为单个控制平面节点 IP。	仅适用于只有一个控制平面副本的开发或 PoC 集群。

有关外部 LoadBalancer 的要求，请参阅规划控制平面端点。有关 API 服务器前方没有负载均衡器的单控制平面部署，请参阅单控制平面（无外部 LB）布局。

6. 业务集群镜像仓库

确定此集群从哪个镜像仓库拉取其平台组件——Kube-OVN 以及 Alauda Container Platform 在集群上安装的平台模块。此选择不影响 kubeadm 管理的镜像，它们来自 `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageRepository`。

选项	配置内容	适用场景
`global` 集群的镜像仓库	无需配置。集群会继承该镜像仓库。	业务节点能够以可接受的吞吐量访问 `global` 集群使用的镜像仓库。
专用镜像仓库	`cpaas-system` 中的 RegistryCredential Secret，由 `cpaas.io/registry-reference` 标签从 `Cluster` 引用。	业务节点位于不同地域或网络网段，应从靠近其所在位置的镜像仓库拉取镜像。

无论选择哪种方式，镜像仓库都必须已包含目标版本的 Alauda Container Platform Core，以及计划在此集群上安装的所有平台插件对应的软件包；每个业务集群节点都必须能够访问该镜像仓库地址。

有关专用镜像仓库的操作步骤及其限制，请参阅为业务集群选择镜像仓库。

WARNING

仅在首次创建集群时读取镜像仓库绑定。在现有 `Cluster` 上更改或移除 `cpaas.io/registry-reference`，不会将该集群切换到其他镜像仓库。专用镜像仓库只能通过 YAML 配置；Web UI 工作流始终使用 `global` 集群的镜像仓库。

7. 节点标签和污点

如需为池中的节点设置专属角色标签和污点（例如，为监控、日志记录或入口流量预留专用节点池），请按照 Infra Nodes 中的说明，在池的资源上声明这些配置。创建节点池前请做好规划。此要求适用于所有节点池，无论集群通过 Web UI 还是 YAML 创建；Web UI 的节点池步骤不会设置标签或污点。

使用 Web UI

INFO

Fleet Essentials 升级边界

Fleet Essentials 1.0.4 及更高版本可以通过 CVO 请求 ACP 4.3 及更高版本的 Distribution Version 升级。但它不会执行 DCS Kubernetes 和 Alauda OS 的替换。有关该 Phase 2 YAML 操作步骤，请参阅在 Huawei DCS 上升级 Kubernetes。通过 Fleet Essentials 创建集群不受此边界影响。

要求：此工作流需要 Fleet Essentials。如果使用由池管理的持久磁盘，请通过 YAML 配置 `DCSIpHostnamePool.spec.pool[].persistentDisk`，因为 Web UI 不提供此字段。

如果新集群将依赖由池管理的持久磁盘，请先使用 YAML 创建或更新作为后端的 `DCSIpHostnamePool`，然后使用 Web UI 完成其余集群工作流。

创建工作流

集群创建遵循一个包含 5 个步骤的向导：

```
Step 1: Basic Info
  ↓
Step 2: Control Plane Node Pool
  ↓
Step 3: Worker Node Pools
  ↓
Step 4: Networking
  ↓
Step 5: Review
```

导航路径：集群 → 集群 → 创建集群 → 选择 Huawei DCS

步骤 1：基本信息

字段	类型	必填	描述
Infrastructure Credential	下拉列表	是	选择现有的 Cloud Credential
名称	文本	是	唯一的集群标识符（小写字母、数字、连字符）
显示名称	文本	否	便于识别的自定义描述
Distribution Version	只读	-	ACP 版本（与 `global` 集群匹配）
Kubernetes 版本	只读	-	由 Distribution Version 确定
集群 API 地址	文本	是	格式：`https://<load-balancer-address>:6443`。对于 `Self-built VIP`，请预留 VIP 并使用本文档中的 YAML 路径。

前置条件检查：

创建集群前，请确保：

- DCS 平台中存在 DCS VM Template，并且 Alauda OS 版本与 Kubernetes 版本匹配

- Kubernetes API 端点已就绪。对于外部 LoadBalancer，请在创建集群前创建它。对于 `Self-built VIP`，请预留 VIP 并使用 YAML 路径。

版本限制：只能创建平台支持的最新 Kubernetes 版本。

步骤 2：控制平面节点池

控制平面节点池固定为 3 个副本，以实现高可用性。

字段	类型	必填	描述
Machine Template	下拉列表	是	按类型筛选模板：Control Plane，以及兼容的 Kubernetes 版本
副本数	只读	-	固定为 3
SSH Authorized Keys	文本	否	添加多个用于访问节点的 SSH 公钥

验证：关联的 IP Pool 必须具有足够的可用 IP 地址（≥ 3）。

步骤 3：业务节点池

可以添加多个业务节点池。每个池具有以下配置：

字段	类型	必填	描述
池名称	文本	是	此节点池的唯一标识符
Machine Template	下拉列表	是	按类型筛选模板：Worker Node，以及兼容的 Kubernetes 版本
副本数	数字	是	默认值：3
Max Surge	数字	否	默认值：0，必须 ≥ 0。如果节点池将使用由池管理的持久磁盘，则保持此值为 `0`

字段	类型	必填	描述
Max Unavailable	数字	否	默认值：1，必须 ≥ 0。当 maxSurge = 0 时，maxUnavailable 必须 > 0 且 ≤ 副本数
SSH Authorized Keys	文本	否	添加多个 SSH 公钥

验证规则：

- 池名称在集群内必须唯一
- IP Pool 必须具有足够的可用 IP 地址（≥ 副本数）
- maxSurge 和 maxUnavailable 必须满足以下约束：如果 maxSurge = 0，则 maxUnavailable > 0
- 如果集群将依赖由池管理的持久磁盘，请保留 `maxSurge = 0`，以便在未来升级期间逐个替换节点

提示：在池名称前加上集群名称和连字符（例如 `mycluster-worker-1`），以避免不同集群之间的命名冲突。

步骤 4：网络

字段	类型	必填	描述
Pods CIDR	CIDR	是	Pod 网络地址范围
Services CIDR	CIDR	是	Service 网络地址范围
Join CIDR	CIDR	是	Kube-OVN Join CIDR 参数

验证：Pods CIDR 和 Services CIDR 不得重叠。

步骤 5：审核

创建集群前，审核所有配置设置：

基本信息：

- 名称、显示名称、Infrastructure Credential
- Distribution Version、Kubernetes 版本
- 集群 API 地址

控制平面节点池：

- Machine Template，包括 VM Template 名称、OS 版本、Kubernetes 版本
- CPU、内存、副本数、SSH 密钥

业务节点池（列表视图）：

- 池名称、Machine Template、副本数
- Max Surge、Max Unavailable、SSH 密钥

如果集群将依赖由池管理的持久磁盘，请将业务节点池的 **Max Surge** 保持为 `0`。

网络：

- Pods CIDR、Services CIDR、Join CIDR

点击 创建，开始集群创建过程。

使用 YAML

集群创建工作流

使用 YAML 时，请在 `global` 集群中创建 Cluster API 资源，以配置基础设施并引导功能完整的 Kubernetes 集群。

> **WARNING**
>
> 重要的命名空间要求

为确保正确集成为业务集群，所有资源都必须部署在 `cpaas-system` 命名空间中。在其他命名空间部署资源可能会导致集成问题。

WARNING

业务集群命名

业务 `cluster-name` 不得为 `global`。该名称专用于 `global` 集群，重复使用会导致业务集群的资源与 `cpaas-system` 中的 `global` 集群资源发生冲突。`global-` 前缀专用于 `global` 集群 DR 工作流拥有的资源；请参阅通用前提条件。请勿将 `global-` 用于业务集群资源，因为故障转移操作可能会将这些资源识别为属于 `global` 集群。

按照惯例，CAPI `Cluster` 和 provider 集群资源（`DCSCluster`）应使用名称 `<cluster-name>`，非根级 CAPI 和 provider 资源（`KubeadmControlPlane`、`KubeadmConfigTemplate`、`MachineDeployment`、machine template、IP/主机名池等）则应以 `<cluster-name>-` 为前缀，例如示例清单中使用的 `<cluster-name>-kcp`。这是一项建议，并非由控制器强制执行的规则，但可以避免多个业务集群位于 `cpaas-system` 中时发生同一命名空间内的冲突，并使操作期间的资源归属一目了然。

配置工作流

按以下步骤依次操作，以配置功能完整的集群（控制平面和业务节点）：

1. 配置 `KubeadmControlPlane`（控制平面 spec 和 kubeadm 引导配置）。
2. 配置 `DCSCluster`（基础设施绑定和负载均衡器引用）。
3. 创建 `Cluster` 资源（用于关联上述两个资源的顶层 CAPI 对象）。
4. 配置业务节点资源：`KubeadmConfigTemplate`（业务节点引导配置）、业务节点 `DCSMachineTemplate`、业务节点 `DCSIpHostnamePool` 和 `MachineDeployment`。仅有控制平面并不能构成可用的集群。 有关四个业务节点 YAML 资源，请参阅部署业务节点。

注意：基础设施资源（Secret、控制平面 `DCSIpHostnamePool`、控制平面 `DCSMachineTemplate`）应单独配置。相关说明请参阅 Huawei DCS 的基础设施资源。

如果需要在滚动替换期间保留任何磁盘，请在相应的 `DCSIpHostnamePool.spec.pool[].persistentDisk` 条目中声明。这包括平台要求的 `/var/cpaas` 磁盘。provider 会创建新的持久卷，作为独立的持久普通卷。当 DCS 环境要求

显式设置精简配置时，请设置 `persistentDisk[].isThin`；如果省略该项，provider 不会发送 `isThin`，DCS 将使用平台默认值。

如果集群需要额外的 NIC，请在创建相应的 Machines 之前，在 `DCSIpHostnamePool.spec.pool[].additionNic[]` 中声明。provider 仅会在创建新 VM 时添加额外的 NIC。之后编辑 Pool 时，现有 VM 不会热添加 NIC。

解析占位符值

以下示例清单使用 `<placeholder>` 语法表示与环境相关的值。其中有几个值具有规范数据来源，应查询这些来源，而不要自行选择：

占位符	规范数据来源
`<control-plane-kubernetes-version>` / `<worker-kubernetes-version>`	`cpaas-system` 命名空间中的 `cpa` `template` ConfigMap，每个发行版
`<dns-image-tag>`	同一 ConfigMap 中的 `data.coredn`
`<etcd-image-tag>`	同一 ConfigMap 中的 `data.etcdTa`
`<vm-template-name>`（位于 `DCSMachineTemplate.spec.template.spec.vmTemplateName` 中）	ConfigMap 的 `cpaas.io/dcs-vm-t` 须与 DCS 平台中注册的 VM templat
`<ssh-authorized-keys>`	由 operator 提供。每个条目都是单行 钥（`ssh-ed25519 AAAA…` / `ssh-r` 验证器要求此字段非空。对于不需要 PoC 集群，可提供任意语法有效的公 私钥）；生产环境请使用 operator 因
`<auth-secret-name>`	你在基础设施 → 云凭证中创建的 Se

占位符	规范数据来源
`<cluster-name>`	由 operator 选择；必须符合 DNS-11 `global`（该名称专用于 `global` `Cluster`、`DCSCluster` 以及每个 `cluster.x-k8s.io/cluster-name` `KubeadmControlPlane` 使用带前缀 `name>-kcp`。完整约定请参阅业务集
`<load-balancer-ip>`	由 operator 提供，供客户端用于访问 于 `type: external`，此值为外部 址 — 集群端点不支持域名）。对于 平面集群，此值为唯一控制平面节点 平面布局）。
`<self-built-vip>`	由 operator 提供的 `Self-built VI` 集群前预留此地址。此地址不得分配 `DCSIpHostnamePool.spec.pool[]`
`<pods-cidr>` / `<services-cidr>` / `<kube-ovn-join-cidr>`	由 operator 提供。不得与主机网络、 CIDR 或你计划互连的任何其他 CAP 如果留空，将回退到 `global` 集群 值；生产环境中不建议这样做：显式 一 `global` 集群上的多个业务集群

Magic-Token 占位符

示例清单中的少数值看起来像占位符，但它们实际上是由 DCS Provider 在机器加入时替换的字面令牌。请保持原样：

字面令牌	含义	替换方
`PROVIDER_ID`	每个 Machine 的 provider ID（例如 `dcs://<dcsmachine-name>`）。	DCS Provider — 在 `kubeadm init` / `kubeadm join` 运行前，覆盖生成的 kubelet 配置中的值。
`NODE_IP`	分配自附加到此 Machine 的 `DCSIpHostnamePool` 条目的节点 IP。	DCS Provider — 使用 `DCSIpHostnamePool.spec.pool[].ip` 的值覆盖。

手动替换或加引号会破坏加入流程，导致节点始终无法向控制平面注册。

网络规划和负载均衡器

创建控制平面资源前，请规划网络架构并选择控制平面端点模式。

要求：

- 网络分段：规划控制平面节点的 IP 地址范围
- 额外 **NIC**：如果节点需要存储、管理或隔离的应用网络，请为每个 IP 槽位规划 `DCSIpHostnamePool.spec.pool[].additionNic[]` 值，包括 DVS 和 Port Group 名称
- **API** 端点模式：使用外部 LoadBalancer，或使用配合 ACP `v4.4+` 的 `Self-built VIP`
- **API server** 地址：为 Kubernetes API server 准备稳定的 VIP 或负载均衡器地址
- 连通性：确保所有组件之间的网络连通

对于 `Self-built VIP`，请验证目标 DCS 端口组允许 VIP 所有权、GARP 或 ARP 更新，以及控制平面 VM 之间的 VRRP 流量。VIP 必须与承载它的 NIC 位于同一 Layer 2 网络中。 对于外部 LoadBalancer，请在规划控制平面端点中验证 Layer 4 监听器、TCP `6443` 后端、HTTPS `/healthz` 检查、后端所有权和端点可达性。

配置 KubeadmControlPlane

KubeadmControlPlane 资源定义控制平面配置，包括 Kubernetes 版本、节点规格和引导设置。

> **Kubernetes 1.35 kubelet 设置**
>
> 只有从 Kubernetes 1.35 开始才需要 `imagePullCredentialsVerificationPolicy: NeverVerify`。使用 Kubernetes 1.34 或更早版本创建集群时，请省略此参数。如果 `contentFrom.secret` 提供了补丁，请验证所引用的 Secret 键包含此 Kubernetes 1.35 参数。

> **TIP**
>
> 完整配置参考

为便于阅读，以下示例省略了较长配置文件中的部分内容。完整配置（包括默认审计策略、准入控制和文件内容）请参阅附录中的完整 KubeadmControlPlane 配置。

kubeadmcontrolplane.yaml

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-coredns: ""
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0 # Required when the cluster relies on pool-managed pers
istent disks
  kubeadmConfigSpec:
    users:
    - name: boot
      sshAuthorizedKeys:
      - "<ssh-authorized-keys>"
    format: ignition
    files:
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        # ... (Admission Configuration Content) ...
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.json
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "_comment": "... (Kubelet Configuration Content) ..."
        }
    # ... (other files) ...
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
```

```
          imageTag: <etcd-image-tag>
      # ... (apiServer, controllerManager, scheduler) ...
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-ip: NODE_IP
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: <cp-dcs-machine-template-name>
  replicas: 3
  version: <control-plane-kubernetes-version>
```

参数说明：

参数	类型	描述	必
`.spec.kubeadmConfigSpec`	object	kubeadm bootstrap provider 启动参数	是
`.spec.machineTemplate.infrastructureRef`	object	DCSMachineTemplate 引用	是

参数	类型	描述	必需
`.spec.replicas`	int	控制平面副本数。必须满足 `1 ≤ replicas ≤ IP Pool size`。对于开发/PoC 单控制平面部署，请设置为 `1`（请参阅单控制平面布局）；只有一个控制平面节点的集群无法升级。生产环境通常使用 `3` 以实现 HA。	是
`.spec.version`	string	Kubernetes 版本（必须与 VM 模板匹配 — 请参阅解析占位符值）	是

有关组件版本（例如 `<dns-image-tag>`、`<etcd-image-tag>`），请参阅 OS Support Matrix。

配置 DCSCluster

DCSCluster 是基础设施集群声明，用于引用负载均衡器和 DCS 平台凭证。

当您在集群外部管理负载均衡器时，请使用 `type: external`。

dcscluster.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSCluster
metadata:
  name: "<cluster-name>"
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    host: <load-balancer-ip>
    port: 6443
    type: external
  credentialSecretRef:
    name: <auth-secret-name>
  controlPlaneEndpoint:
    host: <load-balancer-ip>
    port: 6443
  # Optional. Enable only when the target DCS compute cluster has DRS ena
bled.
  controlPlaneHA:
    enabled: true
  networkType: kube-ovn
  site: <site>
```

对于 ACP `v4.4+`，使用 `type: internal` 启用 `Self-built VIP`：

dcscluster-self-built-vip.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSCluster
metadata:
  name: "<cluster-name>"
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    host: <self-built-vip>
    port: 6443
    type: internal
    vrid: 42
    interface: eth0
  credentialSecretRef:
    name: <auth-secret-name>
  controlPlaneEndpoint:
    host: <self-built-vip>
    port: 6443
  networkType: kube-ovn
  site: <site>
```

参数说明：

参数	类型	描述
`.spec.controlPlaneLoadBalancer`	object	控制平面 API server 暴露方式
`.spec.controlPlaneLoadBalancer.type`	string	`external` 使用由 operator 管理的负载均衡器。`internal` 启用 `Self-buil VIP`，并要求使用 ACP `v4.4+`。
`.spec.controlPlaneLoadBalancer.host`	string	对于 `external`，为外部负载 均衡器 VIP（IP 地址）。对于 `internal`，为预留的 IPv4 VIP。
`.spec.controlPlaneLoadBalancer.port`	int	Kubernetes API server 端口。 有效范围为 `1-65535`。

参数	类型	描述
`.spec.controlPlaneLoadBalancer.vrid`	int	`Self-built VIP` 使用的 Keepalived VRID。当 `type` 为 `internal` 时，请设置 `1` 到 `255` 范围内的值。当 `type` 为 `external` 时，将忽略此字段。
`.spec.controlPlaneLoadBalancer.interface`	string	bootstrap VIP 和 `alive` 使用的可选 NIC 名称。留空时，将根据节点 IP 检测 NIC。对于多 NIC 集群，如果 VIP 必须由特定接口持有，请显式设置此项。
`.spec.credentialSecretRef.name`	string	DCS 身份验证 Secret 名称。该 Secret 定义 DCS Provider 使用接口互联用户（默认）还是域用户进行身份验证 — 请参阅凭证用户类型。
`.spec.controlPlaneEndpoint`	object	CAPI 和 kubeadm 使用的 Kubernetes API endpoint。将其设置为与 `controlPlaneLoadBalancer` 相同的主机和端口。
`.spec.controlPlaneHA.enabled`	bool	启用由 provider 管理的 DCS DRS 控制平面 VM 互斥规则。仅当目标 DCS compute cluster 启用了 DRS，并且具有足够的主机和容量来进行跨主机放置时，才将其设置为 `true`。
`.spec.networkType`	string	目前仅支持 "kube-ovn"
`.spec.site`	string	DCS 平台站点 ID

`controlPlaneHA` 是可选的。启用后，provider 会为此业务集群当前的控制平面 VM 创建并维护一条 DRS `ruleType=2` 互斥规则。DCS 执行实际的放置和运行时迁移。provider 不会运行

DRS 或应用 DRS 建议。如果规则已维护但放置尚未收敛，请等待 DCS 调度机制完成，或从 DCS 平台侧触发/应用 DRS。有关基础设施前提条件，请参阅控制平面跨主机高可用。

自建 VIP 的行为和限制

`Self-built VIP` 需要 ACP `v4.4` 或更高版本。请在创建集群前规划此模式。

建议：仅对具有三个控制平面副本的新建高可用集群使用 `type: internal`。对于使用 `type: external` 的现有集群，请保留外部负载均衡器，除非您已有经过测试的迁移计划。`controlPlaneHA.enabled` 是独立的 DRS 放置功能，Self-built VIP 不需要此功能。

DCS provider 分两个阶段处理 `Self-built VIP`：

1. 在 bootstrap 期间，provider 仅在第一个 `kubeadm init` 控制平面节点上注入 `dcs-bootstrap-vip.service`。此服务会在 `kubeadm` 启动前添加 `<self-built-vip>/32`。
2. 控制平面节点注册后，provider 会创建或更新由 provider 管理的 `alive` `ModuleInfo`。平台插件流程会渲染业务 `AppRelease`，而 `alive` 会在控制平面节点上安装 VIP 运行时。

重要限制：

- 两种模式都只接受 IP 地址：`type: external` 接受外部负载均衡器 VIP，`type: internal` 接受预留的 IPv4 VIP。两种模式下，集群 endpoint 均不支持域名。
- VIP 必须从 `DCSIpHostnamePool` 的主 IP 和附加 NIC IP 中排除。
- 目标 DCS 端口组必须允许 VIP 所有权、GARP 或 ARP 更新以及 VRRP 流量。
- `interface` 必须是长度不超过 15 个字符的 Linux 接口名称。当 VIP 必须绑定到特定 NIC（例如 `eth0` 或 `eth1`）时，请使用此项。
- 在集群控制平面初始化后，不应更改 `type`、`host`、`port`、`vrid` 和 `interface`。
- DCS provider 会创建或修补 `alive` `ModuleInfo`；平台插件流程会创建业务 `AppRelease` 及运行时资源。
- DCS provider 不会注入或持久化 IPVS kernel modules 或 sysctl 参数。请使用随支持的 ACP 版本提供的 VM 模板和插件包。

验证自建 VIP

如果将 `DCSCluster.spec.controlPlaneLoadBalancer.type` 设置为 `internal`，请在集群创建后验证由 provider 管理的 VIP 路径。

```
kubectl -n cpaas-system get dcscluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneLoadBalancer.type}{" "}{.spec.controlPlaneEndpoint.host}{":"}{.spec.controlPlaneEndpoint.port}{"\n"}'
kubectl -n cpaas-system get cluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneEndpoint.host}{":"}{.spec.controlPlaneEndpoint.port}{"\n"}'
kubectl -n cpaas-system get moduleinfo \
  -l cpaas.io/cluster-name=<cluster-name>,cpaas.io/module-name=alive
kubectl --kubeconfig <workload-kubeconfig> -n cpaas-system get apprelease alive
kubectl --kubeconfig <workload-kubeconfig> -n kube-system get pods -l app=alive -o wide
kubectl --kubeconfig <vip-kubeconfig> get --raw=/version
```

预期结果包括：已配置的 VIP endpoint、由 provider 创建的 `alive` `ModuleInfo`、已同步且健康的 `alive` `AppRelease`、每个控制平面节点上均处于 Ready 状态的 `alive` pod，以及通过 VIP 成功执行的 `/version` 请求。

单控制平面（无外部 **LB**）布局

对于开发、PoC 或控制平面仅有 一个副本（`KubeadmControlPlane.spec.replicas: 1`）的任何部署，API server 前方都没有真正的负载均衡器。不过，仍有两个字段需要设置值：

- `.spec.controlPlaneLoadBalancer.host` 和 `.spec.controlPlaneEndpoint.host` — 将两者都设置为唯一控制平面节点的 **IP**（即为该节点在控制平面 `DCSIpHostnamePool` 中分配的同一 IP）。
- `.spec.controlPlaneLoadBalancer.type` — 保留 schema 兼容值 `external`；此布局不会创建负载均衡组件。

具体如下：

```
spec:
  controlPlaneLoadBalancer:
    host: 10.226.82.150     # same IP as the control plane node from the
IP pool
    port: 6443
    type: external
  controlPlaneEndpoint:
    host: 10.226.82.150     # same as above
    port: 6443
```

此布局不具备 HA — 丢失唯一的控制平面节点后，集群 API 将无法访问，直到该节点恢复。请勿在此布局中使用 `Self-built VIP`。生产环境请使用带外部 LoadBalancer 的 `replicas: 3` 或 `Self-built VIP`。

控制平面副本少于三个的集群无法升级。

配置 **Cluster**

Cluster 资源声明集群，并引用控制平面和基础设施资源。

cluster.yaml

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: DCSCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
  labels:
    cluster-type: DCS
    # Optional. Bind this cluster to a dedicated image registry instead of the
    # registry of the `global` cluster. Read only at cluster creation.
    # cpaas.io/registry-reference: <registry-credential-name>
  name: <cluster-name>
  namespace: cpaas-system
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - <pods-cidr>
    services:
      cidrBlocks:
      - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: DCSCluster
    name: <cluster-name>
```

参数说明：

参数	类型	描述	必需
`.spec.clusterNetwork.pods.cidrBlocks`	[]string	Pod CIDR。CAPI schema 中为可选项，但建议显式设置，以便多个	建议

参数	类型	描述	必需
		CAPI 集群可以共存而不发生重叠。如果未设置，kube-ovn 将回退到默认值，该默认值可能与同一 global 集群中的其他集群冲突。	
`.spec.clusterNetwork.services.cidrBlocks`	[]string	Service CIDR。建议与 Pod CIDR 相同 — 显式设置以避免集群之间发生冲突。	建议
`.spec.controlPlaneRef`	object	控制平面引用	是
`.spec.infrastructureRef`	object	基础设施集群引用	是

集群注解：

上面的示例显示了三个注解，但完整的 `Cluster` 资源还包含其他几个注解。下表列出了 **operator** 作者设置的注解（其他一些注解由 ACP controllers 写入，不应预先设置）：

注解	必需	值来源
`capi.cpaas.io/resource-group-version`	是	字面值 `infrastructure.cluster.x-k8s.io/v1`
`capi.cpaas.io/resource-kind`	是	字面值 `DCSCluster`
`capi.cpaas.io/kubernetes`	是	与 `KubeadmControlPlane.spec.version` 和 `MachineDeployment.spec.template.spec.ver` 的值。

注解	必需	值来源
`cpaas.io/kube-ovn-join-cidr`	是	operator 选择的 `/16` CIDR，不得与 Pod / Serv 任何其他集群的 join CIDR 重叠。
`cpaas.io/kube-ovn-version`	是	`global` 集群随附的 kube-ovn 版本。从 CAPI ` Cluster` (`kubectl get cluster global -n system -o jsonpath='{.metadata.annotations.cpaas\.i version}'`) 上的 `cpaas.io/kube-ovn-versio` 取。同一 `global` 集群上的每个健康业务集群都 值，因此如果存在，也可以从其中任意一个集群读
`cpaas.io/registry-address`	是	`global` 集群使用的镜像仓库（通常为 `<regis host>:11443` ）。从 `global` CAPI `Cluster` `get cluster global -n cpaas-system -o jsonpath='{.metadata.annotations.cpaas\.i address}'`) 上的 `cpaas.io/registry-addres` 取。
`cpaas.io/nodes-mode`	是	对于由 DCS Provider 置备的集群，使用字面值 ` managed` 。
`cpaas.io/os-family`	仅 KubeOS	当集群的 DCS VM 模板基于 KubeOS 镜像构建时 值 `kubeos` 。对于 Alauda OS，请省略此项：DC 会将未设置的值或任何不是 `kubeos` 的值处理为 `slemicro` 。

集群启动后，ACP 控制器可能会写入其他只读注解（`cpaas.io/cpu-cores-number`、`cpaas.io/memories`、`cpaas.io/nodes-number` 等）——这些注解由系统计算得出，不得

预先设置在要应用的 YAML 中。

Cluster 标签：

标签	必需	值来源	用途
`cluster-type`	是	字面值 `DCS`。	标记该集群由 DCS Provider 置备。
`cpaas.io/registry-reference`	否	`cpaas-system` 中 RegistryCredential Secret 的名称。	将此集群绑定到专用镜像仓库。省略此项则继承 `global` 集群的仓库。仅在集群创建时读取——请参阅为业务集群选择镜像仓库。

部署工作节点

仅有控制平面还不能构成可用集群。请按以下顺序创建工作节点资源：

1. **IP-Hostname Pool** 配置 - 工作节点的网络设置
2. **Machine Template** 设置 - VM 规格
3. **Bootstrap** 配置 - 节点初始化和加入设置
4. **Machine Deployment** - 节点创建和管理的编排

步骤 1：配置 IP-Hostname Pool

IP-Hostname Pool 定义工作节点虚拟机的网络配置。部署前，必须规划并配置 IP 地址、主机名、DNS 服务器、其他 NIC 以及其他网络参数。

在 Huawei DCS 上，IP 池还用于声明必须在 VM 替换后保留的持久磁盘。对于平台所需的 `/var/cpaas` 磁盘，以及删除并重新创建操作期间必须保留的任何其他工作节点磁盘，请使用 `persistentDisk`。

当节点模板在 `/var/lib/containerd` 中声明专用数据磁盘时，DCS provider 的引导配置会将 Pod 日志存储在 `/var/lib/containerd/pods` 下，并将 `/var/log/pods` 链接到该路径。如果需要在替换节点时保留 Pod 日志数据，请将磁盘继续挂载到 `/var/lib/containerd`，以便数据保留在 containerd 数据磁盘上。

WARNING

池大小要求

池中包含的条目数至少必须等于计划部署的工作节点数。条目不足将导致节点无法部署。

示例：

创建名为 `<cluster-name>-worker-pool` 的 `DCSIpHostnamePool`：

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
spec:
  pool:
  - ip: "<worker-ip-1>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-1>"
    machineName: "<worker-machine-name-1>"
    additionNic:
    - ip: "<worker-additional-ip-1>"
      mask: "<worker-additional-cidr-prefix-length>"
      gateway: "<worker-additional-gateway>"
      dns: "<worker-additional-dns>"
      dvSwitchName: "<additional-dvs>"
      portGroupName: "<additional-port-group>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
  - ip: "<worker-ip-2>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-2>"
    machineName: "<worker-machine-name-2>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
  - ip: "<worker-ip-3>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-3>"
```

```
    machineName: "<worker-machine-name-3>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
```

关键参数：

参数	类型	描述	必需
`.spec.pool[].ip`	string	工作节点虚拟机的 IP 地址	是
`.spec.pool[].mask`	string	CIDR 前缀长度格式的子网掩码，不带 `/`，例如 `24`	是
`.spec.pool[].gateway`	string	网关 IP 地址	是
`.spec.pool[].dns`	string	DNS 服务器 IP 地址（多个地址使用分号分隔）	否
`.spec.pool[].machineName`	string	DCS 平台中的虚拟机名称	否
`.spec.pool[].hostname`	string	虚拟机的主机名	否
`.spec.pool[].additionNic[]`	[]object	绑定到此 IP 槽位的其他 NIC。用于存储、管理或隔离的应用网络。	否
`.spec.pool[].additionNic[].ip`	string	为生成的客户机网络配置设	建议*

参数	类型	描述	必需
		置的其他 NIC IP 地址。	
`.spec.pool[].additionNic[].mask`	string	CIDR 前缀长度格式的其他 NIC 子网掩码，不带 `/`，例如 `24`。	建议*
`.spec.pool[].additionNic[].gateway`	string	其他 NIC 网络的网关。设置此值前请确认路由行为。	否
`.spec.pool[].additionNic[].dns`	string	其他 NIC 网络的 DNS 服务器 IP；需要多个值时使用分号分隔。	否
`.spec.pool[].additionNic[].dvSwitchName`	string	用于解析其他 NIC 的 Port Group 的 DCS 分布式虚拟交换机。	是*
`.spec.pool[].additionNic[].portGroupName`	string	用于附加其他 NIC 的 DCS Port Group。	是*
`.spec.pool[].persistentDisk[]`	[]object	绑定到此 IP 槽位的持久磁盘。用于 `/var/cpaas` 以及节点替换后必须保留的任何磁盘。	否

*标记为 `Yes*` 的字段是 provider 解析目标网络并附加其他 NIC 所必需的。标记为 `Recommended*` 的字段是生成静态客户机网络配置所需的。CRD 不会强制要求这些字段。

步骤 2：配置 Machine Template

DCSMachineTemplate 定义工作节点虚拟机的规格，包括 VM 模板、计算资源、存储配置和网络设置。

此资源中的字段引用了多个 Huawei DCS 平台概念（DCS VM Template、DCS VM Folder、DCS Datastore 及相关项）。有关这些概念的定义，以及它们如何映射到 Cluster API 资源，请参阅Huawei DCS 概念和术语。

> **WARNING**
>
> 必需的磁盘配置
>
> 以下磁盘挂载点是必需的。请勿删除：
>
> - 系统卷 (`systemVolume: true`)
> - `/var/lib/kubelet` - Kubelet 数据目录
> - `/var/lib/containerd` - 容器运行时数据
>
> 请在 IP 池中将 `/var/cpaas` 配置为持久磁盘，而不是在 `DCSMachineTemplate` 中配置。
>
> 可以添加其他模板磁盘，但必须保留这些必需的模板磁盘。

示例：

创建名为 `<cluster-name>-worker-template` 的 `DCSMachineTemplate`：

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      location:
        type: folder
        name: <folder-name>
      vmConfig:
        dvSwitchName: <dv-switch-name> # Optional
        portGroupName: <port-group-name> # Optional
        dcsMachineCpuSpec:
          quantity: <worker-cpu>
        dcsMachineMemorySpec: # MB
          quantity: <worker-memory>
        dcsMachineDiskSpec: # GB
        - quantity: 0
          datastoreClusterName: <datastore-cluster-name>
          systemVolume: true
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/kubelet
          format: xfs
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/containerd
          format: xfs
      ipHostPoolRef:
        name: <cluster-name>-worker-pool
```

关键参数：

参数	类型
`.spec.template.spec.vmTemplateName`	string

参数	类型
`.spec.template.spec.location`	object
`.spec.template.spec.location.type`	string
`.spec.template.spec.location.name`	string
`.spec.template.spec.vmConfig`	object
`.spec.template.spec.vmConfig.dvSwitchName`	string
`.spec.template.spec.vmConfig.portGroupName`	string
`.spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`	int

参数	类型
`.spec.template.spec.vmConfig.dcsMachineDiskSpec`	[]object
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].systemVolume`	bool
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].path`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].format`	string
`.spec.template.spec.ipHostPoolRef.name`	string

*父对象被指定时必需

由 IP 池管理的持久磁盘

在匹配的 `DCSIpHostnamePool.spec.pool[].persistentDisk` 条目中声明升级时保留的任何磁盘。

- 将其用于平台要求的 `/var/cpaas`。
- 为系统磁盘和可能随 VM 重新创建的模板本地磁盘保留 `DCSMachineTemplate`。
- 为每个 IP 条目选择唯一的 `slot`。控制器使用 `(ip, slot)` 作为持久磁盘标识。
- 在替换节点上，来宾磁盘设置逻辑会检查是否存在现有文件系统。如果磁盘已格式化，则跳过 `mkfs`，直接挂载磁盘。
- 持久磁盘工作流要求逐个替换，因此请保留 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`。

- 可以追加新的 `persistentDisk` 条目，但不支持删除现有条目。控制器会在 DCS 侧将新添加的磁盘挂载到运行中的 VM，但不会在来宾 OS 内格式化或挂载磁盘。只有在 VM 被替换且替换后的 VM 在引导期间运行生成的磁盘设置时，来宾格式化和挂载才会生效。
- 控制器会将新的持久卷创建为独立的持久普通卷。重新使用现有卷时，只要求该卷为独立且持久；不要求特定的 DCS 卷类型。
- 创建后，将 `format`、`options` 和 `pciType` 视为不可变字段。
- 仅当 DCS 环境要求为新创建的持久卷显式设置精简配置值时，才使用 `isThin`。如果省略，provider 不会发送 `isThin`，DCS 将使用平台默认值。
- 将 `quantityGB` 和 Datastore 变更视为对发布敏感的变更。当具有足够的集群上下文时，webhook 会尽力根据 DCS 平台执行验证。`isThin` 仅在创建时生效：更改它不会验证、调和或转换现有卷，只会影响之后创建的持久卷。

要在节点操作期间检查持久磁盘的运行时状态，请检查池上的 `status.persistentDiskStatus`：

```
kubectl get dcsiphostnamepool <cluster-name>-worker-pool -n cpaas-system
-o yaml
```

由 IP 池管理的其他 NIC

在 `DCSIpHostnamePool.spec.pool[].additionNic[]` 中声明 worker 的其他 NIC。

- provider 仅在创建新 VM 时应用 `additionNic[]`。
- 编辑 Pool 时，不会热更新现有 worker VM。
- 新的 worker Machine 使用其所占用池槽位中的 `additionNic[]` 值。
- 每个其他 NIC 都应包含 `dvSwitchName` 和 `portGroupName`，以便 provider 在克隆 VM 前解析目标 DCS Port Group。
- 如果同时使用 `persistentDisk[]`，请保留 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`，以便固定 IP 槽位和保留的磁盘逐个迁移节点。
- 如果其他 NIC 具有网关，请确认来宾 OS 路由表仍优先使用预期的主网络作为默认路由。

要检查其他 NIC 的运行时状态，请检查 `DCSMachine.status.additionalNic`：

```
kubectl -n cpaas-system get dcsmachine -l cluster.x-k8s.io/cluster-name=<
cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.networkCon
fig.ip}{"\t"}{.status.additionalNic}{"\n"}{end}'
```

步骤 3：配置 Bootstrap Template

KubeadmConfigTemplate 定义 worker 节点的引导配置，包括用户帐户、SSH 密钥、系统文件和 kubeadm join 设置。

从 Kubernetes 1.35 开始，才要求使用 `imagePullCredentialsVerificationPolicy: NeverVerify`。使用 Kubernetes 1.34 或更早版本创建 worker 时，请省略此参数。如果 `contentFrom.secret` 提供了补丁，请确认被引用的 `worker-kubelet-patch.json` 包含适用于 Kubernetes 1.35 的此参数。

> **INFO**
>
> 模板优化
>
> 该模板包含针对安全性和性能预先优化的配置。仅修改环境中需要自定义的参数。

示例：

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-kct
  namespace: cpaas-system
spec:
  template:
    spec:
      format: ignition
      users:
      - name: boot
        sshAuthorizedKeys:
        - "<ssh-authorized-keys>"
      files:
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
        owner: "root:root"
        permissions: "0644"
        content: |
          {
            "apiVersion": "kubelet.config.k8s.io/v1beta1",
            "kind": "KubeletConfiguration",
            "imagePullCredentialsVerificationPolicy": "NeverVerify",
            "protectKernelDefaults": true,
            "staticPodPath": null,
            "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
            "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
            "streamingConnectionIdleTimeout": "5m",
            "clientCAFile": "/etc/kubernetes/pki/ca.crt"
          }
      preKubeadmCommands:
      - while ! ip route | grep -q "default via"; do sleep 1; done; echo
"NetworkManager started"
      - mkdir -p /run/cluster-api && restorecon -Rv /run/cluster-api
      - if [ -f /etc/disk-setup.sh ]; then bash /etc/disk-setup.sh; fi
      postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
        nodeRegistration:
          kubeletExtraArgs:
            node-ip: NODE_IP
```

```
            provider-id: PROVIDER_ID
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/v
olume/exec/"
            protect-kernel-defaults: "true"
```

`PROVIDER_ID` 和 `NODE_IP` 是 DCS Provider 在 Machine 加入集群时替换的字面量 magic token，必须保持原样。完整列表请参阅 Magic-Token Placeholders。

TIP

替代方案：引用集中管理的 Secret 以提供 worker kubelet 补丁

DCS Provider 插件会在 `cpaas-system` 命名空间中提供名为 `dcs-kubernetes-<kubernetes-major-minor>-files` 的 Secret（例如，适用于 Kubernetes 1.33 的 `dcs-kubernetes-1.33-files`）。除了向 `KubeadmControlPlane` 提供控制平面文件之外（请参阅 Complete KubeadmControlPlane Configuration），此 Secret 还包含适用于 worker `KubeadmConfigTemplate` 的 `worker-kubelet-patch.json` 键。当该 Secret 存在时，可以将上方内联的 `files` 条目替换为 `contentFrom.secret` 引用，这样可以使 worker kubelet 补丁与已安装的插件版本保持同步，并避免在集群升级时手动更新。

```
files:
- contentFrom:
    secret:
      key: worker-kubelet-patch.json
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
  permissions: "0644"
```

内联形式与 Secret 引用形式在功能上等效；当目标集群中存在插件 Secret 时，建议使用 Secret 形式。

步骤 4：配置 Machine Deployment

MachineDeployment 通过引用前面配置的 DCSMachineTemplate 和 KubeadmConfigTemplate 资源，协调 worker 节点的创建和管理。它管理所需的节点数量并处理滚动更新。

示例：

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-md-0
  namespace: cpaas-system
spec:
  strategy:
    rollingUpdate:
      maxSurge: 0 # Required when this node pool relies on persistent dis
ks
      maxUnavailable: 1
    type: RollingUpdate
  clusterName: <cluster-name>
  replicas: 3
  selector:
    matchLabels: null
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster-name>
        pool.name: <cluster-name>-md-0
    spec:
      nodeDrainTimeout: 1m
      nodeDeletionTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-kct
          namespace: cpaas-system
      clusterName: <cluster-name>
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: DCSMachineTemplate
        name: <cluster-name>-worker-template
        namespace: cpaas-system
      version: <worker-kubernetes-version>
```

关键参数：

参数	类型	描述
`.spec.clusterName`	string	节点部署的目标集群名称
`.spec.replicas`	int	worker 节点数量（不得超过 IP 池大小）
`.spec.template.spec.bootstrap.configRef`	object	对 KubeadmConfigTemplate 的引用
`.spec.template.spec.infrastructureRef`	object	对 DCSMachineTemplate 的引用
`.spec.template.spec.version`	string	Kubernetes 版本（必须与 VM Template 匹配）
`.spec.strategy.rollingUpdate.maxSurge`	int	更新期间超过所需数量的最大节点数。当节点池依赖持久磁盘时，将其保持为 `0`
`.spec.strategy.rollingUpdate.maxUnavailable`	int	更新期间不可用节点的最大数量。当持久磁盘与 `maxSurge = 0` 一起使用时，将此值保持为大于 `0` 且不超过副本数

集群验证

部署所有集群资源后，验证集群是否已成功创建并正常运行。

使用控制台

1. 导航至 集群 → 集群
2. 在集群列表中找到新创建的集群
3. 验证集群状态是否显示为 运行中

4. 检查所有控制平面节点和 worker 节点是否为 就绪

使用 kubectl

或者，使用 kubectl 命令验证集群：

```
# Check cluster status
kubectl get cluster -n cpaas-system <cluster-name>

# Verify control plane
kubectl get kubeadmcontrolplane -n cpaas-system <cluster-name>-kcp

# Check machine status
kubectl get machines -n cpaas-system

# Verify cluster deployment
kubectl get clustermodule <cluster-name> -o jsonpath='{.status.base.deployStatus}'
```

对于使用额外 NIC 的集群，还要验证 provider 是否已在匹配的 `DCSMachine` 对象中记录这些 NIC：

```
kubectl -n cpaas-system get dcsmachine -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.networkConfig.ip}{"\t"}{.status.additionalNic}{"\n"}{end}'
```

验证控制平面 HA

如果启用了 `DCSCluster.spec.controlPlaneHA.enabled`，请先检查 `DCSCluster` 条件：

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHAReady")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'
```

按如下方式解读该条件：

状态和原因	含义	后续步骤
`True` `ControlPlaneHAReady`	provider 管理的 DRS 规则已存在，并且 provider 已观测到当前控制平面 VM 位于不同的 DCS 主机上。	无需执行任何操作。
`False` `ControlPlaneHAPending`	provider 正在等待可恢复状态，例如具有足够的 VM URN、足够的成员，或 DCS 侧完成放置。	阅读消息。如果提到放置，请检查 DCS 主机容量、DRS 状态，以及运行时迁移是否受阻。
`False` `ControlPlaneHAFailed`	provider 无法查询、验证、创建、更新、删除或检查 DRS 规则。	阅读消息，然后检查 DCS 凭证、DRS 设置和规则名称冲突；如有需要，再检查 controller 日志。

检查观测到的规则和成员快照：

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='rule={.status.controlPlaneHA.ruleName}{" index="}{.status.
controlPlaneHA.ruleIndex}{" cluster="}{.status.controlPlaneHA.clusterUrn}
{"\n"}{range .status.controlPlaneHA.members[*]}{.machineName}{" "}{.vmUr
n}{"\n"}{end}'
```

`members[]` 列表包含规则中每个控制平面 VM 的 CAPI Machine 名称和 DCS VM URN。当前 DCS 状态快照不包含 DCS 主机名称。如果需要确认确切的主机放置位置，请从 DCS 平台或 DCS API 查询 VM 详细信息，并比较 VM 的 `hostName` 或 `hostUrn` 值。

预期结果

成功创建的集群应显示：

- 集群状态：运行中 或 已配置
- 所有控制平面机器：运行中
- 所有 worker 节点（如果已部署）：运行中
- Kubernetes 节点：就绪

- 集群模块状态：已完成
- 对于多 NIC 集群，每个创建的 VM 都在 `DCSMachine.status.additionalNic` 中具有预期的额外 NIC，并且 guest OS 显示对应的 `ethN` 接口。

附录

完整的 **KubeadmControlPlane** 配置

下面是完整的 `KubeadmControlPlane` 配置，其中包括所有默认的审计策略、准入控制和文件内容。

嵌入式 kubelet 补丁包含 `imagePullCredentialsVerificationPolicy: NeverVerify`，仅从 Kubernetes 1.35 开始需要此参数。对于 Kubernetes 1.34 或更早版本，请移除此参数。

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-coredns: ""
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0 # Required when the cluster relies on pool-managed pers
istent disks
  kubeadmConfigSpec:
    users:
    - name: boot
      sshAuthorizedKeys:
      - "<ssh-authorized-keys>"
    format: ignition
    files:
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: AdmissionConfiguration
        plugins:
        - name: PodSecurity
          configuration:
            apiVersion: pod-security.admission.config.k8s.io/v1
            kind: PodSecurityConfiguration
            defaults:
              enforce: "privileged"
              enforce-version: "latest"
              audit: "baseline"
              audit-version: "latest"
              warn: "baseline"
              warn-version: "latest"
            exemptions:
              usernames: []
              runtimeClasses: []
              namespaces:
```

```
            - kube-system
            - cpaas-system
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.json
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "protectKernelDefaults": true,
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
          "streamingConnectionIdleTimeout": "5m",
          "clientCAFile": "/etc/kubernetes/pki/ca.crt"
        }
    - path: /etc/kubernetes/audit/policy.yaml
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        omitStages:
        - "RequestReceived"
        rules:
        - level: None
          users:
          - system:kube-controller-manager
          - system:kube-scheduler
          - system:serviceaccount:kube-system:endpoint-controller
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
          - group: ""
            resources: ["endpoints"]
        - level: None
          nonResourceURLs:
          - /healthz*
          - /version
          - /swagger*
        - level: None
          resources:
          - group: ""
```

```
      resources: ["events"]
    - level: None
      resources:
      - group: "devops.alauda.io"
    - level: None
      verbs: ["get", "list", "watch"]
    - level: None
      resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
    - level: None
      resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessreview
s"]
      - group: "authentication.k8s.io"
        resources: ["tokenreviews"]
    - level: None
      resources:
      - group: "app.alauda.io"
        resources: ["imagewhitelists"]
      - group: "k8s.io"
        resources: ["namespaceoverviews"]
    - level: Metadata
      resources:
      - group: ""
        resources: ["secrets", "configmaps"]
    - level: Metadata
      resources:
      - group: "operator.connectors.alauda.io"
        resources: ["installmanifests"]
      - group: "operators.katanomi.dev"
        resources: ["katanomis"]
    - level: RequestResponse
      resources:
      - group: ""
      - group: "aiops.alauda.io"
      - group: "apps"
      - group: "app.k8s.io"
      - group: "authentication.istio.io"
      - group: "auth.alauda.io"
      - group: "autoscaling"
      - group: "asm.alauda.io"
      - group: "clusterregistry.k8s.io"
```

```
          - group: "crd.alauda.io"
          - group: "infrastructure.alauda.io"
          - group: "monitoring.coreos.com"
          - group: "operators.coreos.com"
          - group: "networking.istio.io"
          - group: "extensions.istio.io"
          - group: "install.istio.io"
          - group: "security.istio.io"
          - group: "telemetry.istio.io"
          - group: "opentelemetry.io"
          - group: "networking.k8s.io"
          - group: "portal.alauda.io"
          - group: "rbac.authorization.k8s.io"
          - group: "storage.k8s.io"
          - group: "tke.cloud.tencent.com"
          - group: "devopsx.alauda.io"
          - group: "core.katanomi.dev"
          - group: "deliveries.katanomi.dev"
          - group: "integrations.katanomi.dev"
          - group: "artifacts.katanomi.dev"
          - group: "builds.katanomi.dev"
          - group: "versioning.katanomi.dev"
          - group: "sources.katanomi.dev"
          - group: "tekton.dev"
          - group: "operator.tekton.dev"
          - group: "eventing.knative.dev"
          - group: "flows.knative.dev"
          - group: "messaging.knative.dev"
          - group: "operator.knative.dev"
          - group: "sources.knative.dev"
          - group: "operator.devops.alauda.io"
          - group: "flagger.app"
          - group: "jaegertracing.io"
          - group: "velero.io"
            resources: ["deletebackuprequests"]
          - group: "connectors.alauda.io"
          - group: "operator.connectors.alauda.io"
            resources: ["connectorscores", "connectorsgits", "connectorso
cis"]
        - level: Metadata
    preKubeadmCommands:
    - while ! ip route | grep -q "default via"; do sleep 1; done; echo "N
etworkManager started"
    - mkdir -p /run/cluster-api && restorecon -Rv /run/cluster-api
```

```
    - if [ -f /etc/disk-setup.sh ]; then bash /etc/disk-setup.sh; fi
    postKubeadmCommands:
    - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          audit-log-format: json
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          profiling: "false"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          tls-cipher-suites: "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS
_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY130
5,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"
          encryption-provider-config: /etc/kubernetes/encryption-provide
r.conf
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          tls-min-version: VersionTLS12
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
        extraVolumes:
        - name: vol-dir-0
          hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
      scheduler:
        extraArgs:
```

```
          bind-address: "::"
          tls-min-version: VersionTLS12
          profiling: "false"
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-ip: NODE_IP
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: <cp-dcs-machine-template-name>
  replicas: 3
  version: <control-plane-kubernetes-version>
```

TIP

替代方案：引用集中管理的 **Secret**，而不是使用内联内容

Alauda Container Platform Infrastructure Provider for Huawei DCS 插件会在 `cpaas-system` 命名空间中提供一个名为 `dcs-kubernetes-<kubernetes-major-minor>-files` 的 Secret（例如，Kubernetes 1.33 对应的 `dcs-kubernetes-1.33-files`）。该 Secret 包含 `psa-config.yaml`、

`control-plane-kubelet-patch.json` 和 `audit-policy.yaml` 的规范内容，并会随每个版本一同更新。

当该 Secret 存在时，您可以将三个内联的 `files` 条目替换为 `contentFrom.secret` 引用。内联形式和 Secret 引用形式在功能上等效；使用 Secret 可以使文件内容与已安装的插件版本保持一致，并避免在集群升级时手动更新。

```
files:
- contentFrom:
    secret:
      key: psa-config.yaml
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/admission/psa-config.yaml
  permissions: "0644"
- contentFrom:
    secret:
      key: control-plane-kubelet-patch.json
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
  permissions: "0644"
- contentFrom:
    secret:
      key: audit-policy.yaml
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/audit/policy.yaml
  permissions: "0644"
```

`encryption-provider.conf` 不属于此 Secret，也完全不属于 `files` ——它由 DCS provider 提供。请参阅加密 provider 配置的交付方式。

在决定使用哪种形式之前，可以运行以下命令检查目标集群中是否存在该 Secret：

```
# Replace <kubernetes-major-minor> with the value matching this clu
ster
# (for example, 1.33 for Kubernetes v1.33.x).
K8S_MM=<kubernetes-major-minor>
kubectl -n cpaas-system get secret "dcs-kubernetes-${K8S_MM}-files"
>/dev/null 2>&1 \
  && echo "Secret present — contentFrom.secret form is supported" \
  || echo "Secret missing — use inline content form"
```

加密 Provider 配置的交付方式

`/etc/kubernetes/encryption-provider.conf` 由 DCS provider 管理，而不是由 `KubeadmControlPlane` 管理。provider 构建控制平面机器的 Ignition 配置时，会自行写入该文件，并移除从 `kubeadmConfigSpec.files` 传入的该路径的任何条目。您在其中添加的条目不会产生任何效果——它既不会被合并，也不会被应用，因此写入其中的密钥会被静默忽略。

provider 写入的内容取决于集群：

条件	文件内容
已设置 `DCSCluster.spec.encryptionProviderConfigRef`	所引用 Secret 的 `encryption-provider.conf` 密钥。用于灾难恢复，此时两端必须持有相同的密钥。该 Secret 必须存在于 provider 运行所在的集群中；请参阅可选的灾难恢复部署
未设置，且控制平面尚未初始化（第一台控制平面机器）	新生成的随机 AES-CBC 密钥
未设置，且控制平面已在运行（后续每台控制平面机器）	从正在运行的 `kube-apiserver` 中读回的文件，因此所有控制平面节点共享同一个密钥

worker 节点不会接收该文件。

`clusterConfiguration.apiServer.extraVolumes` 会将 `/etc/kubernetes` 挂载到 API server 容器中，因此 `apiServer.extraArgs.encryption-provider-config:`

`/etc/kubernetes/encryption-provider.conf` 在每个控制平面节点上都能解析。请保留该参数（附录中已包含），以使用启用 etcd Secret 静态加密的方式运行。移除该参数会关闭静态加密；provider 仍会写入该文件，但 API server 将不再使用它。

后续步骤

创建集群后：

- 管理节点
- 升级集群

故障排除

如果集群达到 `Provisioned`，但始终未变为 Ready——例如，由于 CNI 未部署，业务节点保持在 `NotReady`——请参阅排查卡在 Provisioned 状态的业务集群。

排查卡在 **Deleting** 状态的 **DCSMachine**

当 Huawei DCS 业务集群上的 `DCSMachine` 停留在 `Phase: Deleting` 状态超过预期的约 60 秒，导致在缩容、滚动升级或拆除整个集群期间节点无法更替时，请使用本指南。

目录

适用范围

症状

读取 VMStopPending Condition

诊断流程

VM 卡在 `stopping`

VM 卡在中间转换状态

controller 无法访问 DCS API

DCS 帐户锁定

FC 站点策略“仅能删除已停止虚拟机”（errorCode `102212808`）

将 `cpaas.io/retain-vm` 用作应急手段

验证恢复情况

另请参阅

适用范围

本指南涵盖 DCS 侧阻止 `cluster-api-provider-dcs` 在节点上完成 `reconcileDelete` 流程的故障模式。provider 的正常删除顺序如下：

```
running VM → safe StopVm → stopped → (detach persistent disks if any) → DeleteVm → VM gone in DCS
```

每个步骤都依赖 DCS 平台做出响应。当 DCS 无响应或拒绝执行某个步骤时，controller 会在 `DCSMachine.Status.Conditions.VMStopPending` 上显示等待原因。首先查看此状态，通常无需访问 controller 日志即可识别问题。

本指南不涵盖正常的删除前限制（CAPI `Machine` drain 超时、被其他 controller 保留的 finalizer）——这些问题属于上游 Cluster API 故障排查范围。

症状

查看位置	所见情况
`Machine.status.phase`（位于 `global` 集群上）	`Deleting` 超过 5 分钟
`DCSMachine.status.conditions[?(@.type=="VMStopPending")]`	`Status: False`，其中包含 `Reason` 和描述等待原因的 `Message`
DCS 门户 VM 视图	VM 仍在列表中（处于运行、停止中或其他非已停止状态）
`global` 集群上的 controller 日志	类似 `issued safe StopVm` 或 `DeleteVm rejected by FC site policy` 的 `cluster-api-provider-dcs-manager` 行

如果缺少 `VMStopPending`，且 `Machine.status.phase` 仍为 `Deleting`，则问题更可能发生在 `reconcileDelete` 之前的上游环节（例如 CAPI drain、kubeadm 删除前钩子）。这种情况下，请参阅 CAPI 故障排查文档↗。

读取 **VMStopPending Condition**

`VMStopPending` condition 是本指南的主要信号。

```
kubectl -n <ns> describe dcsmachine <name>
```

查找：

```
Conditions:
  Type:                VMStopPending
  Status:              False
  Severity:            Warning
  Reason:              VMMidTransition
  Message:             VM stuck in stopping; awaiting transition to stoppe
d
```

`Reason` 字段提供诊断信息：

Reason	Severity	含义	下一步
`WaitingForStop`	Info	已发出安全的 `StopVm`；controller 每 10 秒轮询一次 `stopped`	最多等待 ~60 秒。如果 VM 超过此时间仍处于 `stopping`，请跳转到 VM Stuck in `stopping`。
`VMMidTransition`	Warning	VM 处于除 `running` / `stopped` 之外的临时状态（例如 `stopping`、`starting`、`migrating`、`paused`、`hibernated`）	检查 `Message` 字段以了解具体状态。请参阅 VM Stuck in Mid-Transition。
(condition absent)	—	VM 要么已经处于 `stopped` 状态，要么已经消失——controller 已完成停止阶段。	检查 DCS 门户中 VM 是否存在，以及 `Machine` finalizer。

诊断流程

VM 卡在 `stopping`

controller 发出了优雅的 `StopVm`（调用 DCS API `/action/stop?mode=safe`），VM 已确认但始终未完成状态收敛。

常见原因：

1. **VM** `pvDriverStatus` 不是 `running`：优雅关机所需的 guest tools（vmtools / pvdriver）没有响应。在 DCS 门户中验证：打开 VM 详情页，检查 `pvDriverStatus` 字段。如果显示的不是 `running`，VM 就无法接受安全停止操作。
2. **VM** 模板早于 `4.2.1`：较旧的 DCS VM 模板缺少安全停止所需的 guest tools。确认 DCSMachine 中的 `vmTemplateName` 与 `4.2.1+` 模板匹配（`cpaas-system/<release>-dcs-vm-template` ConfigMap 上的标签 `cpaas.io/dcs-vm-template`，请参阅解析占位符值）。
3. **Guest OS** 挂起：VM 内核卡住，未处理 ACPI 关机请求。

解决路径：

- 对于 (1) 和 (3)：在 DCS 门户中手动发出 `force` 停止操作（VM 详情 → Operations → Stop → Force）。VM 达到 `stopped` 后，controller 的下一次 reconcile 会获取新状态，并继续执行 `DeleteVm`。
- 对于 (2)：此 provider 无法在该模板上安全删除 VM；先将业务负载从节点迁出，手动关闭 VM，然后使用 `cpaas.io/retain-vm` 跳过 controller 端删除。

VM 卡在中间转换状态

`Message` 显示的状态类似于 `migrating`、`starting`、`paused` 或 `hibernated`。controller 不会对处于这些状态的 VM 发出任何操作（DCS 会拒绝），而是无限期等待 VM 收敛到 `running` 或 `stopped`。

消息中的状态	可能原因	解决方法
`migrating`	平台发起的 vMotion / 实时迁移	等待。迁移会自行完成；状态达到 `running` 或 `stopped` 后，controller 会恢复处理。
`starting`	电源事件后平台发起 VM 启动	等待状态变为 `running`，然后继续删除。
`paused` / `hibernated`	平台暂停了 VM（不会自动恢复）	在 DCS 门户中手动恢复 VM（Operations → Resume），或使用 `cpaas.io/retain-vm` 绕过并稍后清理 VM。

controller 无法访问 DCS API

如果 `VMStopPending` condition 不存在，但 `Machine.status.phase` 长时间仍为 `Deleting`，则 controller 可能完全无法与 DCS 通信。检查 controller 日志：

```
kubectl -n cpaas-system logs deployment/cluster-api-provider-dcs-manager --tail=50
```

查找以下关键字之一：

日志关键字	原因
`connection refused` / `timeout`	从 `global` 集群无法访问 DCS API endpoint
`errorCode: 10100116` "帐户锁定中"	DCS 管理员帐户已锁定。请参阅 DCS 帐户锁定。
`401 Unauthorized`	Credential Secret 已过期；按照 Cloud Credentials 轮换。

DCS 帐户锁定

`errorCode: 10100116` 表示集群凭证 Secret 中的 DCS 门户帐户处于暴力破解锁定窗口。DCS 门户的锁定策略会在每次登录失败后重置，因此陷入重试循环的 controller 会无限期延长锁定时间。

要打破此循环：

```
# 1. Scale the manager to 0 so it stops retrying
kubectl -n cpaas-system scale deployment cluster-api-provider-dcs-manager
--replicas=0

# 2. Wait 5 minutes for the DCS lockout window to expire naturally
sleep 300

# 3. Restore the manager
kubectl -n cpaas-system scale deployment cluster-api-provider-dcs-manager
--replicas=1

# 4. Verify the controller is happy
kubectl -n cpaas-system logs deployment/cluster-api-provider-dcs-manager
--tail=20
```

请避免在生产环境中复用 **DCS** 门户的 `admin` 帐户作为 **provider** 的 **Secret**。 配置专用的 DCS 帐户（interconnect 或 domain user，请参阅凭证用户类型），并放宽其锁定策略，这样 controller 的暂时性故障就不会导致整个平台对所有人锁定。

FC 站点策略“仅能删除已停止虚拟机”（errorCode `102212808`）

如果业务集群运行在某个 DCS 站点上，而其底层 FusionCompute 平台启用了“仅能删除已停止虚拟机”安全策略，则当 VM 仍处于 `running` 状态时，**DeleteVm** 会被拒绝。provider 的正常删除流程会先发出安全的 `StopVm`，然后再执行 `DeleteVm`，从而自动处理此情况；但在少见的竞态条件下，外部操作者（DCS 门户中的人工操作员或其他自动化程序）可能在 controller 检查停止状态与调用 `DeleteVm` 之间重启 VM。controller 会将此竞态显示为：

```
Controller log:
  INFO  DeleteVm rejected by FC site policy, requeuing
        vmUrn=urn:sites:<site>:vms:<vm-id>
        errorCode=102212808
```

这不是错误——controller 会将 `errorCode 102212808` 视为可恢复情况并重新排队。下一次 reconcile 会重新检查 VM 状态；如果 VM 再次运行，controller 会重新发出 `StopVm(safe)` 并

继续处理。正常恢复时间约为 ~30 秒。

如果反复看到此 errorCode（超过 3 个周期），说明外部操作者正在与 controller 争抢操作。请暂停外部自动化程序，或应用 `cpaas.io/retain-vm`，以完全在 controller 之外接管 VM 生命周期。

将 `cpaas.io/retain-vm` 用作应急手段

当 controller 因任何原因无法完成 `reconcileDelete`，而你需要释放 `Machine` finalizer，以便业务集群继续执行操作（缩容、升级、移除）时，请为 `Machine` 或 `DCSMachine` 添加注解：

```
kubectl -n <ns> annotate machine <machine-name> cpaas.io/retain-vm=true
```

效果：

- controller 的 `reconcileDelete` 会识别此注解，跳过 `ensureVmStopped` 和 `DeleteVm`，并释放 `DCSMachine` finalizer。
- 随后，`Machine` 会退出 `Deleting` 状态，Cluster API 可以完成父级操作。

副作用——请务必注意：

- **VM** 不会从 **DCS** 中删除——它会保留在 DCS 平台上，继续占用计算和存储资源，不再受 **Cluster API** 生命周期管理。
- `DCSIpHostnamePool` 中的 IP / 主机名槽位也不会释放——在你手动释放之前，池条目会显示为仍在使用。

应用 `retain-vm` 后，请计划手动执行以下操作：

1. 在 DCS 门户中（或通过 DCS API）停止并删除 VM。
2. 从对应的 `DCSIpHostnamePool` 中释放 IP / 主机名条目。
3. （可选）清理已附加的任何持久磁盘。

验证恢复情况

应用上述任一解决方案后，监视 `DCSMachine`，直到条件清除：

```
kubectl -n <ns> describe dcsmachine <name>
# Look for: Conditions: VMStopPending  Status: True
#           (or the condition disappears as the resource is deleted)

kubectl -n <ns> get machines | grep <machine-name>
# Phase should transition: Deleting → (Machine gone within ~30s)
```

如果 `Machine` 已完全消失，且 DCS 门户中不再列出该 VM，则恢复完成。

另请参阅

- 在 Huawei DCS 上创建集群 — 完整的集群清单参考
- Huawei DCS 的云凭证 — Secret 格式和用户类型
- 排查卡在 Provisioned 状态的业务集群 — 与集群启动问题相关的指南
- 排查卡在 Deleting 状态的集群 — 每个子 Machine 都已消失后，顶级 `Cluster` CR 仍卡在 `Deleting` 状态

Huawei Cloud Stack

概览 安装 **Provider** 基础设施资源

创建集群 排查 **Huawei Cloud Stack** 业务集群故障

Huawei Cloud Stack Provider

安装 Provider

安装 Provider

适用场景

前提条件

操作步骤

后续步骤

基础设施资源

基础设施资源

概述

编写 YAML 之前

凭证 Secret 输入

Compute 值

版本和组件基线

网络清单

控制平面 ELB 地址规划

控制平面高可用放置规划

静态 IP 池规划

值到 YAML 的映射

后续步骤

创建集群

创建集群

前提条件

使用 Web UI

使用 YAML

集群验证

管理节点

升级集群

故障排除

排查 **Huawei Cloud Stack** 业务集群故障

排查 **Huawei Cloud Stack** 业务集群故障

模式：主机名错误导致 kubeadm init 始终无法完成

相关内容

Huawei Cloud Stack 的 Infrastructure Provider 可在 Huawei Cloud Stack (HCS) 平台上创建和管理 Alauda OS 集群。此 Provider 与 HCS 集成，通过 Cluster API 为 Kubernetes 集群配置和管理云资源。

目录

安装 Provider
基础设施资源
创建集群
排查 Huawei Cloud Stack 业务集群故障
概述
主要功能
集群管理功能
基于 Web UI 的管理（Fleet Essentials）
基于 YAML 的管理
支持的资源
文档

概述

Huawei Cloud Stack 是一种混合云解决方案，可将公有云功能扩展到私有数据中心。HCS Provider 利用 HCS API 管理基础设施资源，包括虚拟机、网络、存储和负载均衡器。

HCS Provider 支持由池管理的持久磁盘。必须在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中声明需要在节点替换后保留的磁盘，而不是在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中声明。

主要功能

- **Kube-OVN** 集成：原生支持 Kube-OVN 容器网络接口
- **ELB** 负载均衡器：内置弹性负载均衡器支持，用于控制平面高可用
- 静态 **IP** 配置：为企业网络环境分配静态 IP 地址
- 机器配置池：预定义主机名、静态 IP 地址和持久磁盘插槽，实现可预测的 VM 配置

- 由池管理的持久磁盘：在滚动升级期间，可将 EVS 磁盘从旧 ECS 分离并挂载到替换的 ECS
- 控制平面安全引导：支持通过 kubeadm 引导数据配置 kube-apiserver 审计、准入、加密 Provider 和 kubelet 证书
- 按节点池选择可用区：为控制平面或每个工作节点池选择一个 HCS 可用区；不同的工作节点池可以定位到不同的可用区。请参阅可用区放置

集群管理功能

HCS Provider 始终支持通过 Cluster API 清单进行基于 **YAML** 的集群管理。在安装 **Fleet Essentials** `1.0.2` 或更高版本后，还支持基于 **Web UI** 的管理。

基于 **Web UI** 的管理（**Fleet Essentials**）

安装 Fleet Essentials 后，您可以通过 Web UI 管理 HCS 集群：

- 通过 6 步引导式向导创建集群，该向导可按子网选择可分配的静态 IP 地址。
- 对于现有集群，可查看控制平面和工作节点池、添加或删除工作节点池，以及管理预留的节点 IP 地址。

基于 **YAML** 的管理

对于自动化和 GitOps 工作流，可使用 Cluster API 清单管理 HCS 集群。此工作流不依赖 Fleet Essentials。有关基于 YAML 的操作说明，请参阅在 Huawei Cloud Stack 上创建集群。

支持的资源

资源	描述
`HCSCluster`	表示 HCS 基础设施，包括 VPC、子网和负载均衡器配置
`HCSMachine`	表示 HCS 中的虚拟机实例

资源	描述
`HCSMachineTemplate`	用于创建 HCS 机器的模板，其中包含实例规格
`HCSMachineConfigPool`	预定义主机名、静态 IP 地址和持久磁盘插槽的池

文档

- 安装指南
- 创建集群
- 管理节点
- 升级 Kubernetes
- 将现有集群迁移到由池管理的持久磁盘
- Provider API

安装 Huawei Cloud Stack Provider

本文介绍如何在 `global` 集群上安装 Huawei Cloud Stack 的 Infrastructure Provider 和 Cluster Bootstrap Provider。

目录

- 适用场景
- 前提条件
- 操作步骤
  - 下载软件包
  - 上架软件包
  - 安装集群插件
  - 验证安装
- 后续步骤

适用场景

在安装 `global` 集群后、在 Huawei Cloud Stack 上创建第一个业务集群之前，完成本页面中的操作。无论 `global` 集群本身运行在哪个 IaaS 平台上，都必须执行此操作。在灾难恢复（DR）对中，请在两个 `global` 集群上完成此操作，以便在发生故障转移后，备用集群能够协调相同的业务集群。将软件包上架到每个 `global` 集群。

这与安装 global 集群中的 Provider 操作步骤不同。那些操作步骤会将 Provider 软件包上架到临时引导集群，以创建 `global` 集群本身。

前提条件

安装 Provider 前，请确保您具备：

- 访问 灵雀云容器平台 `global` 集群的权限
- 访问 Customer Portal 以下载插件的权限

操作步骤

1 下载软件包

要在 Huawei Cloud Stack 上创建集群，需要两个集群插件。从 Customer Portal 下载这两个插件。对于 ACP 4.5，它们列在 **Common ACP Extensions** 下；请参阅下载。

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Huawei Cloud Stack**

2 上架软件包

使用 `violet push` 将这两个软件包上架到平台。有关命令及其参数，请参阅上架软件包。

3 安装集群插件

在 `global` 集群上安装这两个集群插件。有关操作步骤，请参阅集群插件。

4 验证安装

安装 Provider 后，验证其是否已正确安装：

```
# Check if the HCS provider pods are running
kubectl get pods -n cpaas-system | grep hcs

# Verify HCS CRDs are registered
kubectl get crd | grep -E '^hcs.*\.infrastructure\.cluster\.x-k8s\.io
$'

# Expected output should include:
# hcsclusters.infrastructure.cluster.x-k8s.io
# hcsmachines.infrastructure.cluster.x-k8s.io
# hcsmachinetemplates.infrastructure.cluster.x-k8s.io
# hcsmachineconfigpools.infrastructure.cluster.x-k8s.io
```

后续步骤

安装 Provider 后，您可以继续执行以下操作：

- 准备基础设施资源
- 创建集群

Huawei Cloud Stack 的基础设施资源

目录

概述

编写 YAML 之前

凭证 Secret 输入

Compute 值

可用区放置

版本和组件基线

网络清单

多 NIC 约束

控制平面 ELB 地址规划

运行约束

控制平面高可用放置规划

静态 IP 池规划

池管理持久磁盘

值到 YAML 的映射

后续步骤

概述

在为 Huawei Cloud Stack (HCS) 集群编写 YAML 之前，请先准备好所有必需的 HCS 输入。本页列出了填写任何 `Secret`、`HCSMachineConfigPool`、`HCSMachineTemplate`、`KubeadmControlPlane` 或 `HCSCluster` 清单前必须准备好的值、来源和约束。

请将本页用作准备清单。完成准备后，请继续阅读在 Huawei Cloud Stack 上创建集群和在 Huawei Cloud Stack 上管理节点，了解清单工作流。

> **INFO**
>
> 命名空间要求
>
> 所有 HCS 基础设施资源都必须部署在 `cpaas-system` 命名空间中，以确保它们能够作为工作负载集群与平台正确集成。

编写 YAML 之前

创建或编辑任何集群清单之前，请准备以下输入：

输入项	用途	来源
集群名称	`Cluster`、`KubeadmControlPlane`、`HCSCluster`、模板、池	集群命名计划
Kubernetes 版本和组件基线	`Cluster`、`KubeadmControlPlane`	已批准的版本基线；OS 支持矩阵仅适用于其中列出的组件版本
`accessKey` 和 `secretKey`	HCS 凭证 `Secret`	HCS 我的设置 > 访问密钥
`projectID`	HCS 凭证 `Secret`	HCS 我的设置 > 资源空间
`externalGlobalDomain`	HCS 凭证 `Secret`	HCS 平台访问域名

输入项	用途	来源
`schema`	HCS 凭证 `Secret`	HCS 管理员
`region`	HCS 凭证 `Secret`	HCS 管理员
`imageName`	`HCSMachineTemplate`	HCS 镜像清单
`flavorName`	`HCSMachineTemplate`	HCS 管理员
`availabilityZone`	`HCSMachineTemplate`	HCS 管理员或 ACP Web UI
根磁盘、临时数据卷和持久磁盘布局	`HCSMachineTemplate`、`HCSMachineConfigPool`	集群存储计划
VPC 名称和安全组名称	`HCSCluster`	HCS 网络清单
集群子网清单	`HCSCluster`、`HCSMachineConfigPool`、控制平面 ELB	HCS 网络清单
控制平面和工作节点主机名及静态 IP	`HCSMachineConfigPool`	HCS 子网规划
`vipAddress` 和 `vipSubnetName`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS ELB 地址计划
`elbVirsubnetL4Ips` 和 `elbVirsubnetL7Ips`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS ELB 地址计划
`vipDomainName`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS Cloud DNS Private Zones

输入项	用途	来源
控制平面 HA 放置策略	`HCSCluster.spec.controlPlaneHA`	HCS 调度策略计划
`controlPlaneEndpoint`	`Cluster.status` / 派生的集群 endpoint	控制器管理

凭证 **Secret** 输入

收集所有必需值后，再创建 HCS 凭证 `Secret`。

Secret 密钥	含义	获取位置
`accessKey`	HCS 访问密钥 ID	HCS 我的设置 > 访问密钥
`secretKey`	HCS 秘密访问密钥	HCS 我的设置 > 访问密钥
`projectID`	资源空间 ID	HCS 我的设置 > 资源空间
`externalGlobalDomain`	HCS 平台访问域名	HCS 为 API 访问提供的平台域名
`region`	Provider 使用的 HCS region API 值	HCS 管理员
`schema`	HCS IAM endpoint 的可选 API scheme。若省略，Provider 将使用 `https`	HCS 管理员

在未设置 `schema` 的情况下创建的现有凭证 Secret 将继续正常工作。仅当 HCS IAM endpoint 使用 `http` 而不是默认值 `https` 时，才设置 `schema`。

注意：租户管理员无法从 HCS UI 获取 `region`。在对 Secret 进行编码之前，请从 HCS 管理员处获取 Provider 可识别的准确值。

Compute 值

在编写 `HCSMachineTemplate` 和 `HCSMachineConfigPool` 之前，准备好 VM 镜像、flavor、可用区、临时数据卷布局和持久磁盘布局。

输入	指导
`imageName`	对于要部署的 Alauda OS 镜像，使用经过验证的 HCS 镜像名称
`flavorName`	使用与 `Flavor.Name` 匹配、且 Provider 可识别的 HCS API 值。不要使用租户 UI 中显示的名称
`availabilityZone`	使用与 `ZoneName` 匹配、且 Provider 可识别的 HCS API 值。不要使用租户 UI 中显示的名称
根磁盘和临时数据卷	提前规划系统磁盘和临时数据磁盘。控制平面模板通常使用临时数据卷存储 `/var/lib/etcd`、`/var/lib/kubelet` 和 `/var/lib/containerd`。工作节点模板通常使用临时数据卷存储 `/var/lib/kubelet` 和 `/var/lib/containerd`。
池管理的持久磁盘	提前规划必须在 VM 替换后保留的磁盘。对于 `/var/cpaas` 以及滚动替换期间必须保留的其他节点本地状态，使用此模型。

注意：ACP Web UI 通过所选 HCS 凭证加载 flavor 和可用区。编写 YAML 时，请从 HCS 管理员处获取准确的、Provider 可识别的 `Flavor.Name` 和 `ZoneName` 值，因为原生 HCS 租户控制台可能不会显示这些值。

可用区放置

`HCSMachineTemplate` 会为从该模板创建的每台机器选择一个可用区。ACP Web UI 会为控制平面节点池创建一个模板，并为每个工作节点池创建一个模板。因此：

- 所有控制平面节点使用同一个可用区。
- 一个工作节点池中的所有从节点使用同一个可用区。
- 不同的工作节点池可以选择不同的可用区。
- 单个节点池不会自动将从节点分散到多个可用区。

为实现可预测的放置，请明确选择可用区。如果省略 `HCSMachineTemplate.spec.template.spec.availabilityZone`，Provider 会查询 HCS，并选择 HCS 报告为可用的第一个可用区。该列表的顺序由 HCS 控制。

使用 ACP Web UI 时，可用区下拉列表通过所选 HCS 凭证加载，使用 HCS API `ZoneName`，并禁用 HCS 报告为不可用的可用区。编写 YAML 时，请使用 Provider 可识别的 `ZoneName`，而不是原生 HCS 租户控制台中的显示名称。Provider 会拒绝 HCS 未返回的名称。对于已存在但当前不可用的指定可用区，ECS 配置仍可能访问 HCS 并在那里失败，因此在应用清单之前，请确认该可用区可用。

WARNING

池管理的持久磁盘和控制平面高可用会增加放置约束：

- 现有池管理的持久磁盘会将其替换后的 ECS 实例锁定到该磁盘所在的可用区。如果模板明确选择了不同的可用区，替换操作将因可用区冲突而失败。
- 控制平面高可用通过使用 HCS 服务器组，将控制平面 ECS 实例分散到物理主机上。它不会将控制平面分散到多个可用区；所有控制平面节点仍位于控制平面 `HCSMachineTemplate` 所选择的可用区中。

版本和组件基线

仅将 OS Support Matrix 用于其中明确列出的组件版本，例如受支持 Alauda OS 镜像的 Kubernetes、coredns、etcd 和 pause 版本。

OS Support Matrix 并不是所有 HCS 清单值的完整来源。在编写 YAML 之前，还应获取已批准的发布基线，用于确定容器镜像仓库、DNS 镜像仓库、Kube-OVN 版本、Kube-OVN 加入 CIDR、Pod CIDR 和 Service CIDR 等值。如果你的发布版本尚未提供完整的基线来源，请使用平台或发布负责人针对你的环境验证过的值。

通过 Web UI 而不是 YAML 创建集群时，无需手动输入这些组件版本。向导会从你为 OS 镜像注册的 OS-release ConfigMap 中读取这些版本。有关注册步骤和字段来源，请参阅 OS Release Registration。

网络清单

在编写 `HCSCluster` 或 `HCSMachineConfigPool` 资源之前，准备完整的集群网络清单。

网络规划必须包括：

- 目标 VPC 名称
- 目标安全组名称
- 集群将使用的每个子网名称
- 每个子网的子网 ID 和与 ELB 相关的子网元数据
- 每个子网的 CIDR
- 每个子网的网关和 DNS 值
- 控制平面节点、工作节点、ELB VIP 以及 ELB L4/L7 虚拟子网 IP 的计划可用 IP 范围

如果单个集群使用多个子网，这些子网必须属于同一个 VPC，并且必须允许集群节点彼此通信。

重要：`HCSCluster.spec.network.subnets` 是集群子网清单。`HCSMachineConfigPool`、`vipSubnetName`、`elbVirsubnetL4Ips[].subnetName` 和 `elbVirsubnetL7Ips[].subnetName` 引用的每个 `subnetName` 都必须已存在于该列表中。

对于初始集群创建流程，控制器可以在集群变为 Ready 之前按名称解析现有子网。对于现有的 Ready 集群，请勿仅追加子网名称。添加完整的子网对象，至少包括 `name`、`id`，以及在子网供控制平面 ELB 使用时包括 `neutronSubnetId`。同时保留清单中的 `cidr`、`gatewayIp`、`primaryDNS` 和 `secondaryDNS`，以确保集群子网列表保持完整。

多 NIC 约束

在 `HCSMachineConfigPool.spec.configs[].networks[]` 中声明多个 NIC。每个 `networks[]` 条目只为一个 NIC 选择一个子网和一个静态 IP。

当前 Provider 不允许声明：

- 每个 NIC 的角色或用途，例如管理、服务或存储流量
- 特定 NIC 的默认网关

- 静态路由或路由度量值
- 每个 NIC 的 DNS 设置

将当前的多 NIC 功能视为 NIC 附加，以及子网和静态 IP 的选择。

控制平面 **ELB** 地址规划

HCS Provider 会自动创建控制平面 Elastic Load Balance（ELB）。请在编写 `HCSCluster` 之前规划 ELB 输入。

本文档使用固定地址 ELB 工作流。请在编写 `HCSCluster` 之前准备所有与 ELB 相关的地址：

- `vipSubnetName`
- `vipAddress`
- `elbVirsubnetL4Ips`，恰好包含两个 L4 IP
- `elbVirsubnetL7Ips`，恰好包含两个 L7 IP

保持 `vipDomainName` 未设置：不支持通过域名访问集群（API server 证书只能由集群自身的 CA 签发），并且 Provider 仅使用该字段回填派生的 `controlPlaneEndpoint`。

运行约束

- Provider 会创建 ELB 并启用 Hybrid Load Balancing，使集群节点可以通过 ELB 地址访问 API server。
- 集群创建后，请勿在 HCS ELB 上禁用 Hybrid Load Balancing。
- 请勿在创建清单中填写 `spec.controlPlaneEndpoint`。ELB 就绪后，控制器会填充该字段。

控制平面高可用放置规划

对于多从节点控制平面，可以将 HCS Provider 配置为创建并维护一个由 Provider 管理的服务器组，用于控制平面 ECS 放置。在编写集群清单时，通过 `HCSCluster.spec.controlPlaneHA` 启用此功能。

启用此功能前，请规划放置策略：

策略	行为	运行影响
`anti-affinity`	强反亲和性。HCS 必须将控制平面 ECS 实例放置在不同的主机上。	需要严格主机隔离时使用此策略。如果 HCS 找不到满足规则的足够容量，ECS 创建可能会失败。
`soft-anti-affinity`	尽力反亲和性。HCS 尝试将控制平面 ECS 实例放置在不同的主机上。	如果倾向于分散放置，但不希望该规则阻止 ECS 创建，请使用此策略。最终放置仍可能共享同一主机。

此功能作用于集群范围，仅适用于控制平面计算机。它不会配置工作节点池。Provider 会创建服务器组，将控制平面 ECS 创建请求关联到该组，在 `HCSCluster.status.controlPlaneHA` 中记录服务器组，并通过 `ControlPlaneHAReady` 条件报告进度。

对于现有集群，启用 `HCSCluster.spec.controlPlaneHA` 会创建并记录由 Provider 管理的服务器组，但不会将已在运行的控制平面 ECS 实例追溯添加到该组。服务器组放置会在使用 `os:scheduler_hints.group` 创建控制平面 ECS 时应用。要使现有集群达到目标状态，请启用 `controlPlaneHA`，等待 `HCSCluster.status.controlPlaneHA.id`，然后触发控制平面滚动替换，以便使用该服务器组创建替换 ECS 实例。在当前控制平面 ECS 成员完成替换并在该组中被观测到之前，`ControlPlaneHAReady` 可能仍为 `False`，原因是 `ControlPlaneHAPending`。

有关清单示例和验证命令，请参阅配置 HCSCluster和验证控制平面高可用。

静态 IP 池规划

本页重点介绍规划的静态 IP 工作流。

创建 `HCSMachineConfigPool` 资源前，请准备以下内容：

- 控制平面主机名和静态 IP
- 工作节点主机名和静态 IP（如果创建工作节点）
- 足够覆盖初始从节点数的条目
- 必须绑定到每个固定主机名的池管理持久磁盘

对于至少包含三个从节点的静态 IP 控制平面，推荐使用 `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` 升级路径。这种先缩容再扩容的方法通常不需要额外的控制平面 IP。如果计划采用单控制平面、仅创建拓扑（`replicas: 1`），请勿将该滚动更新设置复制到创建清单中。对于使用持久磁盘的池，请保留 `maxSurge: 0`，因为每个磁盘都绑定到一个固定主机名和槽位。

池管理持久磁盘

在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中声明必须在 VM 删除并重新创建替换过程中保留的 HCS 磁盘。

对于 `/var/cpaas`、监控数据、日志数据或其他必须在滚动替换期间保留的节点本地状态，请使用此模型。对于可以随每个新 ECS 重新创建的临时磁盘，请保留 `HCSMachineTemplate.spec.template.spec.dataVolumes[]`。

hcs-machine-config-pool-with-persistent-disk.yaml

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

持久磁盘字段说明：

字段	说明
`slot`	单个机器配置中的磁盘位置。对于同一主机名，槽位必须从 `0` 开始保持唯一且连续。
`size`	EVS 磁盘大小，单位为 GB。对于新创建的 EVS 数据磁盘，请使用 `10` 到 `32768` GB。现有已声明磁盘必须与其当前大小匹配。
`type`	目标可用区中可用的 EVS 磁盘类型名称。
`mountPath`	客户机 OS 内的挂载路径。对于必须在 VM 替换后保留的平台状态，请使用 `/var/cpaas`。磁盘大小应足以存储平台监控、日志和插件数据。
`format`	初始化新磁盘时使用的文件系统。如果省略，Provider 将使用 `xfs`。重新挂载期间不会重新格式化现有文件系统。
`mountOptions`	替换 VM 挂载磁盘时应用的挂载选项。如果省略，Provider 将使用 `defaults,noatime`。

更新规则：

- 不要在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 和 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中同时声明相同的 `mountPath`。
- 可以向现有机器配置追加新的 `persistentDisks[]` 条目。Provider 会创建或声明 EVS 磁盘，但不会将磁盘热挂载到正在运行的 ECS 中。在期望新磁盘于客户机 OS 内完成格式化和挂载之前，请使用 `maxSurge: 0` 触发滚动替换。
- Provider 接受持久磁盘条目后，不要移除或更改 `slot`、`size`、`type`、`format` 或 `mountPath`。这些字段定义磁盘身份和声明契约。
- 可以更新 `mountOptions`。更改会在节点替换且替换 VM 渲染其磁盘挂载时生效。
- 对于使用持久磁盘的池，请保留 `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` 和 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0`。

值到 YAML 的映射

完成准备检查清单后，使用以下映射：

准备好的输入	YAML 字段
`accessKey`、`secretKey`、`projectID`、`externalGlobalDomain`、`region`、可选的 `schema`	`Secret.data.*`
`imageName`、`flavorName`、`availabilityZone`、根卷以及临时数据卷布局	`HCSMachineTemplate.spec.template.spec.*`
控制平面和工作节点主机名、静态 IP 以及池管理的持久磁盘	`HCSMachineConfigPool.spec.configs[]`
VPC 名称、子网清单、安全组名称	`HCSCluster.spec.network.*`
`vipAddress`、`vipSubnetName`、`vipDomainName`、`elbVirsubnetL4Ips`、`elbVirsubnetL7Ips`	`HCSCluster.spec.controlPlaneLoadBalancer.*`
控制平面 HA 放置策略	`HCSCluster.spec.controlPlaneHA.*`
Kubernetes 版本和组件基线	`KubeadmControlPlane.spec.version`、`Cluster.spec.clusterNetwork.*`、集群注解以及相关引导设置

后续步骤

完成准备检查清单后：

- 按照在 Huawei Cloud Stack 上创建集群编写控制平面和基础设施清单
- 按照在 Huawei Cloud Stack 上管理节点添加或修改工作节点

在 Huawei Cloud Stack 上创建集群

本文介绍如何在 Huawei Cloud Stack 平台上创建 Kubernetes 集群。可通过 Cluster API 清单使用基于 YAML 的方式创建集群。如果已安装 Fleet Essentials `1.0.2` 或更高版本，还可以通过引导式 Web UI 创建集群。

> **INFO**
>
> Web UI 提供内置验证的引导式工作流，而 YAML 则具有更高的自动化灵活性。

目录

前提条件

1. 安装所需插件
2. 准备 HCS 基础设施输入
3. 注册 OS 版本
4. 业务集群映像注册表
5. 节点标签和污点

使用 Web UI

创建流程

第 1 步：基本信息

第 2 步：网络

第 3 步：基础设施

第 4 步：控制平面节点池

Machine Config 条目

第 5 步：业务节点池
第 6 步：审阅
使用 YAML
集群创建概览
控制平面配置
配置 HCS 身份验证
配置 Machine Configuration Pool
配置 Machine Template
配置 KubeadmControlPlane
配置 HCSCluster
配置 Cluster
部署工作节点
步骤 1：配置 Machine Configuration Pool
工作节点池管理的持久磁盘
步骤 2：配置机器模板
步骤 3：配置引导模板
步骤 4：配置机器部署
集群验证
使用 kubectl
验证控制平面 HA
预期结果
管理节点
升级集群
故障排除

前提条件

创建集群前，请确保满足以下所有前提条件：

1. 安装所需插件

在 Huawei Cloud Stack 上创建第一个业务集群之前，请先将以下插件上传并安装到 灵雀云容器平台 的 `global` 集群上。无论 `global` 集群本身运行在哪个 IaaS 平台上，都需要执行此操作：

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Huawei Cloud Stack**

操作步骤请参阅安装 Provider。

2. 准备 HCS 基础设施输入

编写本文中的任何 YAML 之前，请准备好所有 HCS 特定输入：

- HCS 凭证 Secret 的值
- Provider 可识别的计算值，例如 `imageName`、`flavorName` 和 `availabilityZone`
- 集群网络清单，包括集群使用的子网和可用 IP 地址范围
- 控制平面 ELB 地址规划，包括 `vipAddress`、`vipSubnetName` 以及固定的 L4 和 L7 IP
- 控制平面和工作节点的静态 IP 池规划

完整的检查清单、来源信息和限制条件，请参阅华为云 Stack 的基础设施资源。

3. 注册 OS 版本

此先决条件仅适用于 **Web UI** 工作流。基于 YAML 的创建方式会直接设置 OS 映像和组件版本，因此无需执行此操作。

Web UI 不会要求你输入 OS 映像及其 Kubernetes 和组件版本，而是从 **OS-release ConfigMap** 中读取这些信息。每个 Alauda OS 映像都需要注册一个此类 ConfigMap。向导会从 `global` 集群解析平台 Distribution 版本，查找 `supportedACPVersions` 包含该 Distribution 的已注册 OS 映像，并据此确定节点映像和控制平面组件版本。

如果没有已注册的 OS 映像与当前 Distribution 匹配，向导会阻止创建，并显示类似以下错误：

> 当前平台 Distribution v4.3.2 未注册 OS。请先注册 OS-release ConfigMap。

打开向导之前，请先在 `global` 集群上注册 OS 映像：

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  # Any DNS-1123 name. The name is not a lookup key.
  # Do not copy the image version verbatim: it contains "_" (for example, x86_64), which is invalid in a ConfigMap name.
  name: alaudaos-43-m55-x86-64
  namespace: cpaas-system
  labels:
    cpaas.io/os-release: ""
data:
  osImageVersion: "alaudaos-43.m55.202606251459-0.x86_64"
  supportedACPVersions: "v4.3.2"
  osVariant: "m55"
  architecture: "x86_64"
  osBuild: "202606251459-0"
  kubernetesVersion: "v1.34.5-3"
  containerdVersion: "2.2.1-5"
  coreDNSImageTag: "1.14.2-v4.3.11"
  etcdImageTag: "v3.5.28-260625"
  pauseImageTag: "3.10"
```

使用 `kubectl apply -f os-release.yaml` 应用该配置。

字段来源：

字段	值来源
`cpaas.io/os-release` 标签	值为空的固定标记。向导通过此标签发现 ConfigMap。
`osImageVersion`	上传到 HCS 平台的 Alauda OS 映像的确切名称。Provider 使用此名称查找 ECS 映像，并将其设为节点池默认映像。对于标准版本，该名称与为相应 ACP 版本发布的 Alauda OS 映像上传时使用的名称一致。请参阅下载 Alauda OS。
`supportedACPVersions`	此 OS 映像适用的 ACP Distribution 版本，以逗号分隔（例如 `v4.3.2` 或 `v4.3.0,v4.3.1`）。向导会将 `global` 集群的 Distribution 与此列表进行匹配。

字段	值来源
`architecture`	`x86_64` 或 `aarch64`。必须与计划使用的节点规格的架构一致。
`osVariant`、`osBuild`	用于参考说明的字段；不用于匹配。
`kubernetesVersion`、`containerdVersion`、`coreDNSImageTag`、`etcdImageTag`、`pauseImageTag`	此 OS 映像的组件版本，取自 OS 支持矩阵中匹配的行。向导会将这些版本注入新集群的控制平面，因此它们必须存在于你的隔离网络注册表中。

INFO

请勿向此 ConfigMap 添加 Kube-OVN 版本。向导会从 `global` 集群的 Kube-OVN 配置中读取 Kube-OVN 版本，而不是从 OS 映像中读取。

`osImageVersion` 必须指定 HCS 平台上已存在的映像。若要读取已有集群的环境中使用的确切名称，请列出现有的机器模板：

```
kubectl -n cpaas-system get hcsmachinetemplate \
  -o custom-columns=NAME:.metadata.name,IMAGE:.spec.template.spec.imageName
```

4. 业务集群映像注册表

确定此集群从哪个注册表拉取平台组件——Kube-OVN 以及 Alauda Container Platform 安装在集群上的平台模块。此选择不会影响 kubeadm 管理的映像，这些映像来自 `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageRepository`。

选项	配置内容	适用场景
`global` 集群的注册表	无需配置。集群会继承该注册表，向导会自动填写其地址。	业务节点能够以可接受的吞吐量访问 `global` 集群使用的注册表。

选项	配置内容	适用场景
专用注册表	`cpaas-system` 中的 RegistryCredential Secret，由 `cpaas.io/registry-reference` 标签在 `Cluster` 中引用。	业务节点位于不同地域或网络段，应从距离较近的注册表拉取映像。

无论采用哪种方式，注册表都必须已包含目标版本的 Alauda Container Platform Core，以及计划安装到此集群的每个平台注册插件的软件包；并且每个业务集群节点都必须能够访问该注册表地址。

有关专用注册表的操作步骤和限制条件，请参阅为业务集群选择映像注册表。

WARNING

专用注册表只能通过 YAML 配置。Web UI 向导始终使用 `global` 集群的注册表：它会从 `global` 集群读取该地址，并自动将其写入新集群，且不提供供你选择的注册表凭证。如果此集群需要使用自己的注册表，请参阅使用 YAML。

此外，只有在首次创建集群时才会读取绑定。在现有 `Cluster` 上更改或移除 `cpaas.io/registry-reference`，不会使该集群切换到其他注册表。

5. 节点标签和污点

若要为某个池中的节点设置专属角色标签和污点，例如为监控、日志记录或入口预留专用池，请按照基础设施节点中的说明，在池的资源上声明标签和污点。请在创建节点池之前进行规划。此要求适用于所有节点池，无论集群是通过 Web UI 创建还是使用 YAML 创建；Web UI 的节点池步骤不会设置标签或污点。

使用 Web UI

要求：此操作流程要求 Fleet Essentials `1.0.2` 或更高版本。否则，请使用使用 YAML 中所述的 YAML manifests 创建集群。

Web UI 会使用你选择的 Infrastructure Credential，从 HCS 平台加载 VPC、子网、安全组、flavor 和可分配的 IP 地址。开始前，请确保满足上述前提条件，并且 `cpaas-system` 命名空间中存在 HCS credential Secret。

创建流程

集群创建遵循包含 6 个步骤的向导：

```
Step 1: Basic Info
  ↓
Step 2: Networking
  ↓
Step 3: Infrastructure
  ↓
Step 4: Control Plane Node Pool
  ↓
Step 5: Worker Node Pools
  ↓
Step 6: Review
```

导航路径：集群 → 集群 → 创建集群 → 选择 Huawei Cloud Stack

> **INFO**
>
> **IP** 地址选择方式
>
> 此向导中的每个 IP 字段——第 3 步中的控制平面 ELB 地址，以及第 4 步和第 5 步中的节点 IP 地址——都是选择器，而不是自由文本框。请先选择子网，选择器只会列出该子网中可分配的 IP 地址。HCS 集群使用静态 IP 地址，因此你必须从列表中选择地址；不支持手动输入和 DHCP。更改子网会清除已选择的地址。同一子网中已在其他位置选择的 IP 地址会从列表中移除，因此同一地址不会被分配两次。当子网有大量可用地址时，列表仅显示第一批地址；请输入内容以筛选特定地址。

第 1 步：基本信息

字段	类型	必填	描述
Infrastructure Credential	下拉列表	是	现有的 HCS credential Secret。用于加载 VPC、子网、安全组、flavor 和可分配的 IP 地址。如果下拉列表为空，请在 `cpaas-system` 命名空间中创建一个带有

字段	类型	必填	描述
			`capi.cpaas.io/provider=cluster-api-provider-hcs` label 的 Secret。
Name	文本	是	唯一的集群标识符（小写字母、数字和连字符），最多 63 个字符。不能为 `global`。
Display Name	文本	否	用于便于识别的自定义描述，最多 256 个字符。
Distribution Version	只读	-	平台 distribution 版本，继承自 `global` 集群。
Kubernetes Version	只读	-	由 Distribution Version 确定。
OS Version	只读	-	取自与 Distribution Version 匹配的已注册 OS-release ConfigMap 的 Alauda OS 镜像版本。

第 2 步：网络

字段	类型	必填	描述
Kube-OVN Version	只读	-	从 `global` 集群中检测。
Pod CIDR	CIDR	是	Pod 网络地址范围。
Service CIDR	CIDR	是	Service 网络地址范围。不能与 Pod CIDR 重叠。
Join CIDR	CIDR	是	Kube-OVN join 网络地址范围。

第 3 步：基础设施

此步骤分为两部分：HCS 基础设施资源和控制平面负载均衡器。

基础设施资源：

字段	类型	必填	描述
VPC Name	下拉列表	是	从所选 credential 加载的现有 VPC。
Security Group Name	下拉列表	是	现有安全组。
Subnets	下拉列表	是	一个或多个现有子网。负载均衡器或节点池引用的每个子网都必须添加到此处。

控制平面负载均衡器：HCS provider 会为 Kubernetes API server 创建 Elastic Load Balance（ELB），并负责管理其 VIP。Hybrid Load Balancing 要求固定的 L4 和 L7 虚拟子网 IP。

字段	类型	必填	描述
VIP Subnet Name	下拉列表	是	保存 ELB VIP 的子网。必须是上述已添加的子网之一。
VIP Address	IP 选择器	是	控制平面 ELB 的固定 VIP。
VIP Domain Name	文本	否	留空。系统不支持通过域访问集群——此字段仅用于回填派生的集群 endpoint，且 API server 证书由集群自己的 CA 签发。
L4 Virtual Subnet Name	下拉列表	是	L4 Hybrid Load Balancing IP 所使用的子网。
L4 Virtual Subnet IP 1 / IP 2	IP 选择器	是	两个固定的 L4 IP 地址。
L7 Virtual Subnet Name	下拉列表	是	L7 Hybrid Load Balancing IP 所使用的子网。
L7 Virtual Subnet IP 1 / IP 2	IP 选择器	是	两个固定的 L7 IP 地址。

第 4 步：控制平面节点池

为实现高可用，控制平面节点池固定为 3 个副本。

字段	类型	必填	描述
Control Plane Node Pool Name	文本	是	节点池名称，会自动添加 `<cluster-name>-` 前缀。
Flavor	下拉列表	是	HCS flavor；列表会显示每个选项的 vCPU 和内存。
Availability Zone	下拉列表	否	每个控制平面节点的目标可用区。报告为不可用的可用区会被禁用。请参阅可用区放置。
Machine Configs	列表	是	每个控制平面节点对应一个条目。至少需要 3 个条目。请参阅下方的 Machine Config 条目。
System Disk	数字	是	根磁盘大小，单位为 GB；类型固定为 SSD。
Data Volumes	列表	否	挂载到池中每个节点的固定数据磁盘；只能调整大小。控制平面会包含 etcd 数据卷。
Replicas	只读	-	固定为 3。
Control Plane High Availability	开关	否	默认启用。通过使用 provider 管理的 HCS server group，将控制平面节点分散到所选可用区内不同的物理主机上。
Placement Policy	单选按钮	有条件	启用 High Availability 时显示。**Preferred**（软反亲和性）（默认）会尽力将节点分散放置。**Required**（严格反亲和性）会强制将每个节点放置到不同主机上；如果主机数量不足，创建将失败。

字段	类型	必填	描述
SSH Authorized Keys	列表	否	用于访问节点的公钥。

Machine Config 条目

Machine Configs 中的每个条目描述一个节点：

- **Hostname**（必填）：小写字母、数字、连字符和点，最多 253 个字符（例如 `master-1` 或 `master-1.example.org`）。有关带点的 hostname 如何影响节点，请参阅节点上的 hostname 行为。
- **Networks**（必填）：每个 NIC 对应一行。每行包含一个 **IP Address** 选择器和一个 **Subnet Name**。对于多 NIC 节点，请添加多个行。
- **Persistent Disks**：平台要求的 `/var/cpaas` 磁盘会预先填充；可以调整其大小，但不能移除该行。仅为必须在节点重建后保留的额外主机绑定数据添加行。

第 5 步：业务节点池

点击 **Add Worker Node Pool** 打开节点池对话框。你可以添加多个节点池。

字段	类型	必填	描述
Worker Node Pool Name	文本	是	节点池名称，会自动添加 `<cluster-name>-` 前缀。在集群内必须唯一。
Flavor	下拉列表	是	HCS flavor；列表会显示每个选项的 vCPU 和内存。
Availability Zone	下拉列表	否	此业务节点池中每个节点的目标可用区。不同业务节点池可以选择不同的可用区；报告为不可用的可用区会被禁用。请参阅可用区放置。
Machine Configs	列表	是	每个业务节点对应一个条目。使用与控制平面相同的 Machine Config 条目布局。

字段	类型	必填	描述
System Disk	数字	是	根磁盘大小，单位为 GB；类型固定为 SSD。
Data Volumes	列表	否	挂载到池中每个节点的固定数据磁盘。
Replicas	数字	是	池中节点的数量。不能超过 machine configs 的数量。
Rollout Strategy	数字	否	**Max Surge** 和 **Max Unavailable**。当 Max Surge 为 `0` 时，Max Unavailable 必须 ≥ 1 且 ≤ Replicas。
SSH Authorized Keys	列表	否	用于访问节点的公钥。

第 6 步：审阅

审阅 **Basic Info**、**Infrastructure**、**Control Plane Load Balancer**、**Control Plane Node Pool**、**Worker Node Pools** 和 **Networking** 部分。点击 **View YAML** 查看将要创建的确切资源。点击 **Create** 开始创建集群。

使用 YAML

集群创建概览

总体而言，你将在 灵雀云容器平台 的 `global` 集群中创建以下 Cluster API 资源，以配置基础设施并引导一个正常运行的 Kubernetes 集群。

在本页编写任何 YAML 之前，请完成华为云 Stack 基础设施资源中的准备清单。此清单涵盖 provider 所需的值、这些值的获取位置，以及在填写清单文件之前必须规划的值。

WARNING

重要的命名空间要求

为确保与 灵雀云容器平台 作为业务集群正确集成，所有资源都必须部署在 `cpaas-system` 命名空间中。在其他命名空间中部署资源可能会导致集成问题。

WARNING

业务集群命名

业务 `cluster-name` 不能为 `global`。该名称已保留给 `global` 集群，重复使用会导致业务集群的资源与 `global` 集群在 `cpaas-system` 中的资源发生冲突。`global-` 前缀已保留给 `global` 集群的 DR 工作流所拥有的资源；请参阅通用前提条件。不要将 `global-` 用于业务集群资源，因为故障转移操作可能会将这些资源选作属于 `global` 集群的资源。

按照约定，将 CAPI `Cluster` 和 provider 集群资源（`HCSCluster`）均准确命名为 `<cluster-name>`，并为非根 CAPI 和 provider 资源（`KubeadmControlPlane`、`KubeadmConfigTemplate`、`MachineDeployment`、machine template、machine config pool 等）添加 `<cluster-name>-` 前缀——例如，示例清单使用 `<cluster-name>-kcp`。这是一项建议，而非控制器强制执行的规则，但它可以防止多个业务集群位于 `cpaas-system` 时发生同一命名空间内的冲突，并使操作期间的资源归属一目了然。

集群创建过程按以下顺序进行：

1. 配置 HCS 身份验证（Secret）
2. 创建 machine configuration pool（HCSMachineConfigPool）
3. 配置 machine template（HCSMachineTemplate）
4. 配置 KubeadmControlPlane
5. 配置 HCSCluster
6. 创建 Cluster
7. 部署工作节点（`HCSMachineConfigPool`、`HCSMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment`）。请参阅部署工作节点。

控制平面配置

控制平面管理集群状态、调度和 Kubernetes API。本节介绍如何配置高可用控制平面。

WARNING

配置参数指南

配置资源时，请谨慎修改参数：

- 仅替换 `<>` 中包含的值，替换为特定于环境的值
- 保留所有其他参数，因为这些参数代表经过优化或必需的配置
- 修改非占位符参数可能导致集群不稳定或集成问题

配置 HCS 身份验证

HCS 身份验证信息存储在 `cpaas-system` 中的 Secret 资源内。

请使用编辑器编写凭证文件，而不是在 shell 提示符下输入值。Shell 会将整条命令（包括 Secret 密钥）记录在历史记录中；使用编辑器写入的文件不会留下此类副本。

hcs-credential.env

```
accessKey=<access-key>
secretKey=<secret-key>
projectID=<project-id>
region=<region>
externalGlobalDomain=<domain>
schema=https
```

限制文件权限，从文件创建 Secret，然后删除该文件：

```
chmod 600 hcs-credential.env

HCS_SECRET_NAME="<credential-secret-name>"

kubectl create secret generic "${HCS_SECRET_NAME}" \
  --namespace cpaas-system \
  --from-env-file=hcs-credential.env

rm -f hcs-credential.env
```

以纯文本形式写入值——`kubectl` 在创建 Secret 时会对其进行 base64 编码，因此无需单独的编码步骤。每一行都是 `key=value`。不要为值加引号：`kubectl` 会将引号字符按字面值处理。值中可以包含 `=`；每行中只有第一个 `=` 用于分隔键和值。

> **不要使用 kubectl apply 创建凭证 Secret**
>
> 客户端 apply 会将其应用的清单（包括 HCS Secret 密钥）存储在 Secret 自身的 `kubectl.kubernetes.io/last-applied-configuration` 注解中。`kubectl create` 不会写入此类注解。无论值以 `stringData` 形式还是 base64 `data` 形式写入，这一点都适用；base64 是一种编码，而不是保护措施。

键	描述
`accessKey`	**My Settings** > **Access Keys** 中的 HCS access key ID
`secretKey`	**My Settings** > **Access Keys** 中的 HCS secret access key
`projectID`	**My Settings** > **Resource Spaces** 中的 Resource Space ID
`region`	provider 使用的 HCS region API 值。租户管理员无法从 HCS UI 获取此值；请向 HCS 管理员获取
`externalGlobalDomain`	provider 使用的 HCS platform access domain
`schema`	HCS IAM endpoint 的可选 API scheme，通常为 `https`。省略此行即可使用默认值

未设置 `schema` 创建的现有凭证 Secret 可以继续正常使用。只有当你的 HCS IAM endpoint 使用 `http` 而不是默认值 `https` 时，才设置 `schema`。

你可以重复使用现有的 HCS 凭证 Secret。其名称无需与集群名称匹配，但 `HCSCluster.spec.identityRef.name` 必须引用此 Secret。

配置 Machine Configuration Pool

`HCSMachineConfigPool` 定义预配置的主机名、静态 IP 地址，以及由 pool 管理的 VM 持久磁盘。

> **WARNING**
>
> **Pool** 大小要求
>
> 配置 pool 至少必须包含与你计划部署的控制平面节点数量相同的条目。

每个 `networks[]` 条目使用一个子网选择器。对于新清单，请设置 `subnetName` 或 `subnetId`，但不能同时设置两者。现有清单可以保留已弃用的 `subenetName` 字段；如果在更新该清单时同时添加 `subnetName`，其值必须与 `subenetName` 完全匹配。不要在 `subenetName`、`subnetName` 和 `subnetId` 之间提供相互冲突的值。

如果在 machine configuration pool 中使用 `subnetName`，请在 `HCSCluster.spec.network.subnets` 中包含相同的子网名称。

对于初始集群创建流程，只需按名称列出现有子网即可，因为控制器会在集群变为 Ready 前解析子网元数据。如果稍后向现有的 Ready `HCSCluster` 添加另一个子网，请勿仅追加 `name`。应使用完整的子网对象修补父级 `HCSCluster.spec.network.subnets` 条目，以便后续 machine 或 ELB 操作可以重复使用已解析的子网元数据。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <cluster-name>-cp-pool
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: master-1
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.11
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: master-2
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.12
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: master-3
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.13
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
```

参数	类型	必填	描
`.spec.configs[]`	array	是	非 co
`.spec.configs[].hostname`	string	是	V 写 符 （ 小 和 25 R D 制 点 符 会 在 (告 点 和 返 的 为
`.spec.configs[].networks[]`	array	是	V 列
`.spec.configs[].networks[].subnetName`	string	否*	新 网
`.spec.configs[].networks[].subnetId`	string	否*	子 不 段 s
`.spec.configs[].networks[].ipAddress`	string	是	V
`.spec.configs[].persistentDisks[]`	array	否	在 除

参数	类型	必填	描
			仍
`.spec.configs[].persistentDisks[].slot`	int	是*	一 co 盘 的 始
`.spec.configs[].persistentDisks[].size`	int	是*	E 大 的 请 3 声 当
`.spec.configs[].persistentDisks[].type`	string	是*	目 E
`.spec.configs[].persistentDisks[].mountPath`	string	否	G 于 保 使
`.spec.configs[].persistentDisks[].format`	string	否	文 省 用
`.spec.configs[].persistentDisks[].mountOptions`	array	否	挂 略 c

*对于新清单，请设置 `subnetName` 或 `subnetId` 中的一个。现有清单可以继续使用 `subenetName`，并且仅当两个字段使用相同值时才可以添加 `subnetName`。不要提供相互冲突的子网选择器值。

指定 **`persistentDisks`** 时，必须填写持久磁盘字段。

对于必须在 VM 替换后保留的节点本地状态，请使用 `persistentDisks[]`。不要在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中声明相同的挂载路径。

注意：CRD schema 将 `subnetName`、`subenetName` 和 `subnetId` 列为可选字段，但未表达其允许的组合。编写清单时，请遵循上述 provider 级别的规则。

注意：要将多个 NIC 挂载到一个节点，请添加多个 `networks[]` 条目。provider 仅使用这些条目挂载 NIC，并分配子网选择器和静态 IP。它不支持声明每个 NIC 的角色、默认网关、静态路由或每个 NIC 的 DNS 设置。

节点上的主机名行为

provider 根据 `hostname` 推导节点的 POSIX 主机名和 FQDN，具体如下：

pool `hostname` 值	节点 `hostname`	节点 `hostname -f`	写入的 `/etc/hosts` 条目
`master-1`（无点号）	`master-1`	`master-1`（解析器返回短名称）	provider 不添加任何条目
`master-1.example.org`（包含点号）	`master-1`（短标签，去除点号及其后的所有内容）	`master-1.example.org`	`<node-ip> master-1.example.org master-1`

包含点号的形式用于设置依赖 FQDN 解析的应用（`hostname -f`、包含 SAN 条目的证书和日志标签）。只有在提供包含点号的主机名时，provider 才会设置 `prefer_fqdn_over_hostname: false` 并启用 cloud-init `manage_etc_hosts`，因此 POSIX 工具仍会看到短名称，而 `hostname -f` 会返回完整的 FQDN。

无效的主机名（开头为点号、结尾为点号、全为点号的字符串、大写字母，或任何违反字段约束的值）会在 VM 启动前被拒绝。错误会设置在所属 `Machine.status` 上，并显示在 `Cluster.status.conditions` 中，因此会出现在 `kubectl describe cluster <name>` 和 `kubectl get machines -n cpaas-system -o wide` 下。

配置 **Machine Template**

`HCSMachineTemplate` 定义控制平面节点的 VM 规格。

> **WARNING**
>
> 存储要求

建议为控制平面节点配置以下数据磁盘挂载点：

- `/var/lib/etcd` - etcd 数据（10GB+）
- `/var/lib/kubelet` - kubelet 数据（100GB+）
- `/var/lib/containerd` - 容器运行时数据（100GB+）

`/var/cpaas` 路径存储平台状态和日志。当这些数据必须在 VM 替换后保留时，请在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中声明该路径。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  template:
    spec:
      imageName: <vm-image-name>
      flavorName: <instance-flavor>
      availabilityZone: <availability-zone>
      rootVolume:
        type: SSD
        size: 100
      configPoolRef:
        name: <cluster-name>-cp-pool
      dataVolumes:
        - size: 10
          type: SSD
          mountPath: /var/lib/etcd
          format: xfs
        - size: 100
          type: SSD
          mountPath: /var/lib/kubelet
          format: xfs
        - size: 100
          type: SSD
          mountPath: /var/lib/containerd
          format: xfs
```

参数	类型	必填	说明
`.spec.template.spec.imageName`	string	是	HCS 目标 Alauc 所用的 中以 `versi` 符。 Alauc 因此不 硬编码
`.spec.template.spec.flavorName`	string	是	Provi API 值 `Flav` 请勿使 的名称
`.spec.template.spec.availabilityZone`	string	否	Provi API 值 匹配。 的每个 选可用 Provi 告为可 参阅可
`.spec.template.spec.rootVolume.type`	string	是	卷类型 `SATA`
`.spec.template.spec.rootVolume.size`	int	是	系统磁 GB
`.spec.template.spec.configPoolRef.name`	string	是	引用的 HCSI 名称
`.spec.template.spec.dataVolumes[]`	array	否	数据卷
`.spec.template.spec.dataVolumes[].size`	int	是*	数据磁 GB

参数	类型	必填	说明
`.spec.template.spec.dataVolumes[].type`	string	是*	卷类型
`.spec.template.spec.dataVolumes[].mountPath`	string	是*	挂载路
`.spec.template.spec.dataVolumes[].format`	string	是*	文件系 或 `e`

*指定 dataVolumes 时必填。

`dataVolumes[]` 会随 ECS 一起重新创建。请勿将其用于 `/var/cpaas` 或任何其他必须在滚动替换期间保留的路径。请将这些路径放在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中。

注意：请勿在 `HCSMachineTemplate` 清单中设置 `providerID` 或 `serverId` 等运行时身份字段。Provider 会在创建 HCS 实例时分配这些值。

注意：ACP Web UI 会通过所选 HCS 凭证加载 flavor 和可用区。编写 YAML 时，请向 HCS 管理员获取 Provider 可识别的确切 `Flavor.Name` 和 `ZoneName` 值，因为原生 HCS 租户控制台可能不会显示这些值。

配置 KubeadmControlPlane

`KubeadmControlPlane` 定义 Kubernetes 控制平面配置。

> **Kubernetes 1.35 kubelet 设置**
>
> 只有从 Kubernetes 1.35 开始才需要 `imagePullCredentialsVerificationPolicy: NeverVerify`。使用 Kubernetes 1.34 或更早版本创建集群时，请省略此参数。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  kubeadmConfigSpec:
    files:
      - path: /etc/kubernetes/admission/psa-config.yaml
        owner: root:root
        permissions: "0644"
        content: |
          apiVersion: apiserver.config.k8s.io/v1
          kind: AdmissionConfiguration
          plugins:
          - name: PodSecurity
            configuration:
              apiVersion: pod-security.admission.config.k8s.io/v1
              kind: PodSecurityConfiguration
              defaults:
                enforce: "privileged"
                enforce-version: "latest"
                audit: "baseline"
                audit-version: "latest"
                warn: "baseline"
                warn-version: "latest"
              exemptions:
                usernames: []
                runtimeClasses: []
                namespaces:
                - kube-system
                - cpaas-system
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
        owner: root:root
        permissions: "0644"
        content: |
```

```
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "protectKernelDefaults": true,
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
          "streamingConnectionIdleTimeout": "5m",
          "clientCAFile": "/etc/kubernetes/pki/ca.crt"
        }
    - path: /etc/kubernetes/encryption-provider.conf
      owner: root:root
      permissions: "0600"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: EncryptionConfiguration
        resources:
        - resources:
          - secrets
          providers:
          - aescbc:
              keys:
              - name: key1
                secret: bootstrap-placeholder
    - path: /etc/kubernetes/audit/policy.yaml
      owner: root:root
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        rules:
        - level: Metadata
  clusterConfiguration:
    imageRepository: <image-repository>
    dns:
      imageTag: <dns-image-tag>
    etcd:
      local:
        imageTag: <etcd-image-tag>
    apiServer:
      extraArgs:
        audit-log-format: json
        audit-log-mode: batch
        audit-log-path: /etc/kubernetes/audit/audit.log
```

```
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          encryption-provider-config: /etc/kubernetes/encryption-provide
r.conf
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
          profiling: "false"
          tls-min-version: VersionTLS12
        extraVolumes:
          - name: vol-dir-0
            hostPath: /etc/kubernetes
            mountPath: /etc/kubernetes
            pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
    postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          protect-kernel-defaults: "true"
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
```

```
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: HCSMachineTemplate
      name: <cluster-name>-control-plane
```

解析 cloud-init 数据时，HCS 控制器还会注入文件。对于控制平面机器，它会写入 `/etc/kubernetes/pki/kubelet.crt`、`/etc/kubernetes/pki/kubelet.key` 和 `/etc/kubernetes/encryption-provider.conf`。对于第一个控制平面机器，控制器会生成加密提供程序配置。控制平面初始化后，控制器会尝试复用现有的 `kube-apiserver` 加密提供程序配置。如果在 `/etc/kubernetes/encryption-provider.conf` 添加了引导文件，请将其视为占位文件，因为控制器生成或同步的文件优先级更高。

注意：请同时配置 `apiServer.extraArgs` 和 `apiServer.extraVolumes`。如果未挂载该卷，`kube-apiserver` 就无法读取写入 `/etc/kubernetes` 下的文件。

注意：上方的 `rolloutStrategy.rollingUpdate.maxSurge: 0` 示例适用于高可用静态 IP 控制平面。对于至少有三个副本的固定大小控制平面池，请保留此设置，以确保替换按先缩容再扩容的顺序进行。如果要创建单控制平面 HCS 集群（`spec.replicas: 1`），请勿将 `rolloutStrategy` 区块复制到创建清单中。`KubeadmControlPlane` 校验会拒绝针对单个副本的这种缩容式滚动更新配置。

注意：HCS 还支持通过设置 `spec.replicas: 1`，并在引用的 `HCSMachineConfigPool` 中准备一个控制平面配置条目，来创建单控制平面集群。请将其视为仅用于创建的拓扑，并在创建清单中不设置滚动更新策略。单控制平面 HCS 集群无法升级。

仅当 OS 支持矩阵明确列出相关组件版本时，才使用其中的信息，例如受支持的 Alauda OS 镜像对应的 coredns 和 etcd 镜像标签。它并非所有 HCS 清单值的完整信息来源。应用此 YAML 前，还应根据已批准的版本基线设置 `imageRepository`、DNS 镜像仓库、Kube-OVN 版本、Kube-OVN join CIDR、Pod CIDR 和 Service CIDR 等值。

配置 **HCSCluster**

`HCSCluster` 资源定义 HCS 基础架构配置。

HCS Provider 会在 HCS 平台上为 Kubernetes API server 创建弹性负载均衡（ELB）。此 ELB 必须启用混合负载均衡，以便集群节点也能通过 ELB 地址访问 API server。

对于文档中介绍的 HCS 工作流，请提供 `vipAddress`、`elbVirsubnetL4Ips` 和 `elbVirsubnetL7Ips`。每个 `elbVirsubnetL4Ips[].ips` 和 `elbVirsubnetL7Ips[].ips` 条目都必须包含两个 IP。

请勿设置 `vipDomainName`——不支持通过域名访问集群；请参阅下方字段表。

请先在 `spec.network.subnets` 中列出集群的所有子网，再在其他位置引用这些子网。`vipSubnetName`、`elbVirsubnetL4Ips[].subnetName`、`elbVirsubnetL7Ips[].subnetName` 以及 `HCSMachineConfigPool` 使用的 `subnetName` 值，都必须存在于 `spec.network.subnets` 中。

在初始集群创建流程中，控制器可以从 `name` 解析现有子网元数据。对于已存在的 Ready 集群，请追加完整的子网对象，而不只是 `name`。请包含 `id`，并为控制平面 ELB 将使用的任何子网包含 `neutronSubnetId`。同时将 `cidr`、`gatewayIp`、`primaryDNS` 和 `secondaryDNS` 保留在子网清单中。

创建集群后，请勿禁用 Provider 创建的 ELB 上的混合负载均衡。集群依赖此 ELB 模式，以便节点能够通过 ELB 地址访问 API server。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSCluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
spec:
  # Optional. Include only when you want provider-managed control-plane ECS placement.
  controlPlaneHA:
    enabled: true
    policy: soft-anti-affinity
  controlPlaneLoadBalancer:
    vipAddress: <control-plane-vip-address>
    vipSubnetName: <vip-subnet-name>
    elbVirsubnetL4Ips:
      - subnetName: <subnet-name>
        ips:
          - <l4-ip-1>
          - <l4-ip-2>
    elbVirsubnetL7Ips:
      - subnetName: <subnet-name>
        ips:
          - <l7-ip-1>
          - <l7-ip-2>
  networkType: kube-ovn
  network:
    vpc:
      name: <vpc-name>
    subnets:
      - name: <subnet-name>
    securityGroup:
      name: <security-group-name>
  identityRef:
    name: <credential-secret-name>
```

参数	类型	必填
`.spec.networkType`	string	是
`.spec.network.vpc.name`	string	是

参数	类型	必填
`.spec.network.subnets[].name`	string	是
`.spec.network.subnets[].id`	string	建议
`.spec.network.subnets[].neutronSubnetId`	string	建议网
`.spec.network.subnets[].cidr`	string	建议
`.spec.network.subnets[].gatewayIp`	string	建议
`.spec.network.subnets[].primaryDNS`	string	建议
`.spec.network.subnets[].secondaryDNS`	string	建议
`.spec.network.securityGroup.name`	string	是
`.spec.identityRef.name`	string	是
`.spec.controlPlaneHA.enabled`	bool	设置 `con` 时必
`.spec.controlPlaneHA.policy`	string	设置 `con` 时必

参数	类型	必填
`.spec.controlPlaneLoadBalancer`	object	是
`.spec.controlPlaneLoadBalancer.vipAddress`	string	是
`.spec.controlPlaneLoadBalancer.vipSubnetName`	string	是
`.spec.controlPlaneLoadBalancer.vipDomainName`	string	否
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[]`	array	是
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[].subnetName`	string	是
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[].ips[]`	string	是
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[]`	array	是
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[].subnetName`	string	是
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[].ips[]`	string	是

请勿在创建清单中包含 `spec.controlPlaneEndpoint`。在 HCS 创建流程中，控制器会在创建 `HCSCluster` 后，从 `spec.controlPlaneLoadBalancer` 派生并填充此字段。请勿手动设

置 `controlPlaneEndpoint`，也不要添加空的 `controlPlaneEndpoint` 对象。如果清单中明确包含 `controlPlaneEndpoint`，则必须同时包含 `host` 和 `port`。

`controlPlaneHA` 为可选项。如果包含此项，则必须同时设置 `enabled` 和 `policy`。严格隔离主机时请使用 `anti-affinity`。如果希望将主机分散放置，但不希望仅因 HCS 无法满足严格放置规则而导致 ECS 创建失败，请使用 `soft-anti-affinity`。有关规划指南（包括在现有集群上启用此功能时的滚动替换要求），请参阅 控制平面高可用放置计划。

配置 **Cluster**

Cluster API 中的 `Cluster` 资源用于声明集群，并引用控制平面和基础设施资源。

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  annotations:
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: HCSCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
    cpaas.io/kube-ovn-version: <kube-ovn-version>
  labels:
    cluster-type: HCS
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - <pods-cidr>
    services:
      cidrBlocks:
        - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: HCSCluster
    name: <cluster-name>
```

注解	描述
`cpaas.io/sentry-deploy-type`	部署类型，设置为 `Baremetal`
`cpaas.io/alb-address-type`	ALB 地址类型，设置为 `ClusterAddress`
`capi.cpaas.io/resource-group-version`	资源组版本
`capi.cpaas.io/resource-kind`	资源类型，设置为 `HCSCluster`

注解	描述
`cpaas.io/kube-ovn-join-cidr`	Kube-OVN join 子网 CIDR
`cpaas.io/kube-ovn-version`	Kube-OVN 版本

部署工作节点

仅有控制平面还不能构成可用的集群。控制平面达到 Ready 状态后，请按以下顺序创建工作节点资源。

准备工作节点 YAML 之前，请完成 Huawei Cloud Stack 的基础设施资源中的 HCS 输入检查清单。具体来说，请在 `HCSCluster.spec.network.subnets` 中列出每个工作节点子网，从规划的可用 IP 地址范围中分配工作节点 IP，并收集 Provider 认可的 `flavorName` 和 `availabilityZone` API 值。如果要向现有且处于 Ready 状态的集群添加新的工作节点子网，请使用完整子网对象更新 `HCSCluster.spec.network.subnets`，而不只是添加子网名称。

步骤 1：配置 Machine Configuration Pool

`HCSMachineConfigPool` 定义工作节点 VM 的网络配置以及由池管理的持久磁盘。部署前必须规划并配置 IP 地址、主机名、持久磁盘槽位及其他参数。

> **WARNING**
>
> 池大小要求
>
> 池中条目的数量至少必须与计划部署的工作节点数量相同。条目不足将导致节点无法部署。

每个 `networks[]` 条目使用一个子网选择器。对于新清单，请设置 `subnetName` 或 `subnetId`，但不能同时设置。现有清单可以继续保留已弃用的 `subenetName` 字段；如果更新该清单时还添加了 `subnetName`，其值必须与 `subenetName` 完全一致。请勿在 `subenetName`、`subnetName` 和 `subnetId` 中提供冲突的值。

如果工作节点使用 `subnetName`，请在创建或扩容工作节点池之前，将相同的子网名称加入父级 `HCSCluster.spec.network.subnets` 列表。对于已存在且处于 Ready 状态的集群，请追加包含子网 ID 在内的完整子网对象，而不只是添加子网名称。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: <worker-1-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-1-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: <worker-2-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-2-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: <worker-3-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-3-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
```

参数	类型	必填	描
`.spec.configs[]`	array	是	非列
`.spec.configs[].hostname`	string	是	V写符（小和超
`.spec.configs[].networks[]`	array	是	V列
`.spec.configs[].networks[].subnetName`	string	否*	新网
`.spec.configs[].networks[].subnetId`	string	否*	子存此
`.spec.configs[].networks[].ipAddress`	string	是	工IP
`.spec.configs[].persistentDisks[]`	array	否	在除磐磁
`.spec.configs[].persistentDisks[].slot`	int	是*	单盘下且
`.spec.configs[].persistentDisks[].size`	int	是*	E大的请

参数	类型	必填	描
			的 当
`.spec.configs[].persistentDisks[].type`	string	是*	目 E
`.spec.configs[].persistentDisks[].mountPath`	string	否	来 V 的 /
`.spec.configs[].persistentDisks[].format`	string	否	文 时 x
`.spec.configs[].persistentDisks[].mountOptions`	array	否	挂 P c

*对于新清单，请设置 `subnetName` 或 `subnetId`。现有清单可以继续使用 `subenetName`；只有在两个字段的值相同时，才可以添加 `subnetName`。请勿提供相互冲突的子网选择器值。

指定 **`persistentDisks`** 时，必须填写持久磁盘字段。

对于工作节点替换后必须保留的节点本地状态，请使用 `persistentDisks[]`。请勿在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中声明相同的挂载路径。

注意： CRD 架构将 `subnetName`、`subenetName` 和 `subnetId` 列为可选字段，且未说明这些字段允许的组合。编写清单时，请遵循上述 Provider 级别的规则。

注意： 当工作节点需要多个 NIC 时，`networks[]` 可以包含多个条目。当前 Provider 仅使用每个条目，通过子网选择器和静态 IP 附加 NIC。它不支持按 NIC 声明角色、选择默认网关、配置静态路由、路由度量值或按 NIC 配置 DNS。

工作节点池管理的持久磁盘

在对应的 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 条目中声明必须在工作节点替换后保留的磁盘。对于 `/var/cpaas` 以及其他必须在滚动替换期间保留的节点本地

状态，请使用此模式。

- 将 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 用于临时磁盘，这些磁盘可以随每个 ECS 重新创建。
- 对每个主机名，槽位必须唯一，并从 `0` 起连续排列。Provider 使用 `(hostname, slot)` 作为持久磁盘标识。
- Provider 接受条目后，请将 `slot`、`size`、`type`、`format` 和 `mountPath` 视为不可变字段。
- 可以更新 `mountOptions`。更改将在工作节点替换后生效。
- 可以追加新的 `persistentDisks[]` 条目。Provider 会创建或认领 EVS 磁盘，但不会将磁盘热挂载到正在运行的 ECS 中。请使用 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` 触发滚动替换，然后再预期新磁盘已在来宾 OS 内完成格式化和挂载。

要在工作节点操作期间检查持久磁盘的运行时状态，请查看池状态：

```
kubectl get hcsmachineconfigpool <cluster-name>-worker-pool -n cpaas-syst
em -o yaml
```

步骤 2：配置机器模板

`HCSMachineTemplate` 定义工作节点的 VM 规格。

为工作节点配置系统卷和临时数据卷，用于存放可随每个 ECS 重新创建的路径，例如 `/var/lib/kubelet` 和 `/var/lib/containerd`。当平台状态必须在工作节点替换后保留时，请将 `/var/cpaas` 放在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中。

准备工作节点模板时，请使用 Provider 识别的 `flavorName` 和 `availabilityZone` API 值。这些值不是租户 UI 中显示的名称。

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      imageName: <vm-image-name>
      flavorName: <instance-flavor>
      availabilityZone: <availability-zone>
      rootVolume:
        type: SSD
        size: 100
      configPoolRef:
        name: <cluster-name>-worker-pool
      dataVolumes:
        - size: 20
          type: SSD
          mountPath: /var/lib/kubelet
          format: xfs
        - size: 20
          type: SSD
          mountPath: /var/lib/containerd
          format: xfs
```

参数	类型	必填	描述
`.spec.template.spec.imageName`	string	是	VM 镜
`.spec.template.spec.flavorName`	string	是	Provi API 位 `Flav`

参数	类型	必填	描述
`.spec.template.spec.availabilityZone`	string	否	Provi API 值 匹配。 的每个 所选可 略，F HCS 区。设
`.spec.template.spec.rootVolume.type`	string	是	卷类型
`.spec.template.spec.rootVolume.size`	int	是	系统磁 GB
`.spec.template.spec.configPoolRef.name`	string	是	引用的 HCSI 名称
`.spec.template.spec.dataVolumes[]`	array	否	数据卷
`.spec.template.spec.dataVolumes[].size`	int	是*	磁盘大
`.spec.template.spec.dataVolumes[].type`	string	是*	卷类型
`.spec.template.spec.dataVolumes[].mountPath`	string	是*	挂载路
`.spec.template.spec.dataVolumes[].format`	string	是*	文件系

*指定 dataVolumes 时必填。

`dataVolumes[]` 会随 ECS 重新创建。请勿将其用于 `/var/cpaas` 或任何其他必须在滚动替换后保留的路径。

注意： 请勿在 `HCSMachineTemplate` 清单中设置 `providerID` 或 `serverId` 等运行时身份字段。Provider 会在创建 HCS 实例时分配这些值。

注意： ACP Web UI 会通过所选 HCS 凭证加载规格和可用区。编写 YAML 时，请向 HCS 管理员获取 Provider 识别的确切 `Flavor.Name` 和 `ZoneName` 值，因为 HCS 原生租户控制台可能不会显示这些值。

步骤 3：配置引导模板

`KubeadmConfigTemplate` 定义工作节点的引导配置。

只有从 Kubernetes 1.35 开始才需要 `imagePullCredentialsVerificationPolicy: NeverVerify`。使用 Kubernetes 1.34 或更早版本创建工作节点时，请省略此参数。

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-kct
  namespace: cpaas-system
spec:
  template:
    spec:
      files:
        - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
son
          owner: root:root
          permissions: "0644"
          content: |
            {
              "apiVersion": "kubelet.config.k8s.io/v1beta1",
              "kind": "KubeletConfiguration",
              "imagePullCredentialsVerificationPolicy": "NeverVerify",
              "protectKernelDefaults": true,
              "staticPodPath": null,
              "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
              "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
              "streamingConnectionIdleTimeout": "5m",
              "clientCAFile": "/etc/kubernetes/pki/ca.crt"
            }
      postKubeadmCommands:
        - chmod 600 /var/lib/kubelet/config.yaml
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
        nodeRegistration:
          kubeletExtraArgs:
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/v
olume/exec/"
```

解析工作节点 cloud-init 数据时，HCS 控制器会注入 `/etc/kubernetes/pki/kubelet.crt` 和 `/etc/kubernetes/pki/kubelet.key`。上面的 kubelet 补丁将 kubelet 配置为使用这些由控制器提供的证书文件。

步骤 4：配置机器部署

`MachineDeployment` 负责协调工作节点的创建和管理。

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-md-0
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
  template:
    spec:
      clusterName: <cluster-name>
      version: <kubernetes-version>
      nodeDrainTimeout: 1m
      nodeDeletionTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-kct
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: HCSMachineTemplate
        name: <cluster-name>-worker-template
```

参数	类型	必填	描述
`.spec.clusterName`	string	是	目标集

参数	类型	必填	描述
`.spec.replicas`	int	是	工作
`.spec.template.spec.bootstrap.configRef`	object	是	对 Kube 的引
`.spec.template.spec.infrastructureRef`	object	是	对 HC 的引
`.spec.template.spec.version`	string	是	Kube
`.spec.strategy.rollingUpdate.maxSurge`	int	否	更新 最大
`.spec.strategy.rollingUpdate.maxUnavailable`	int	否	更新 大数

集群验证

部署所有集群资源后，验证集群是否已成功创建。

使用 **kubectl**

```
# Check cluster status
kubectl get cluster -n cpaas-system <cluster-name>

# Verify control plane nodes
kubectl get kubeadmcontrolplane -n cpaas-system <cluster-name>-kcp

# Check machine status
kubectl get machines -n cpaas-system

# Verify cluster deployment status
kubectl get clustermodule <cluster-name> -o jsonpath='{.status.base.deplo
yStatus}'
```

验证控制平面 HA

如果启用了 `HCSCluster.spec.controlPlaneHA`，请先检查 `HCSCluster` 条件：

```
kubectl get hcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHARead
y")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'
```

按以下方式解释该条件：

状态和原因	含义	后续步骤
`True` `ControlPlaneHAReady`	提供商管理的服务器组存在，且当前控制平面 ECS 成员已被跟踪。	无需执行任何操作。
`False` `ControlPlaneHAPending`	提供商正在等待可恢复状态，例如 ECS 创建完成或服务器组成员关系收敛。	阅读消息，并等待其中引用的控制平面机器或 ECS 操作完成。
`False` `ControlPlaneHAFailed`	提供商无法创建、验证、更新或清理服务器组，或者 HCS 拒绝按照所选策略进行 ECS 调度。	阅读消息，然后根据需要检查 HCS 容量、所选策略和控制器日志。

检查观测到的服务器组和成员关系快照：

```
kubectl get hcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='serverGroup={.status.controlPlaneHA.name}{" id="}{.status.
controlPlaneHA.id}{" policy="}{.status.controlPlaneHA.policy}{"\n"}{range
.status.controlPlaneHA.members[*]}{.machineName}{" "}{.serverID}{"\n"}{en
d}'
```

对于 `anti-affinity`，HCS 将服务器组策略视为硬调度约束。如果容量不足，ECS 创建可能会失败，此时应将条件消息作为首要诊断信号。对于 `soft-anti-affinity`，HCS 会尝试分散成员，但仍可能将多个 ECS 实例放置在同一主机上。

预期结果

成功创建的集群应显示：

- 集群状态：运行中 或 已置备
- 所有控制平面机器：运行中
- Kubernetes 节点：就绪
- 集群模块状态：已完成

管理节点

有关扩缩容、模板更新和 Web UI 节点池等日常节点操作，请参阅在 Huawei Cloud Stack 上管理节点。

升级集群

有关升级集群组件的说明，请参阅升级 Kubernetes。

故障排除

如果集群达到 `Provisioned`，但始终未变为就绪状态（例如，由于未部署 CNI，业务节点一直处于 `NotReady` 状态），请先参阅与提供商无关的排查卡在已置备状态的业务集群。

有关 HCS 特有的故障模式（例如，由于带点号的 `HCSMachineConfigPool` 主机名生成了包含点号的 POSIX 主机名，导致 `kubeadm init` 始终无法完成），请参阅排查 Huawei Cloud Stack 业务集群故障。

排查 Huawei Cloud Stack 业务集群故障

本指南介绍通过 Cluster API 管理的 Huawei Cloud Stack (HCS) 业务集群的特定故障排查模式。

如果集群已达到 `Cluster.status.phase = Provisioned`，但导入流程未完成，并且由于缺少 CNI，业务节点仍处于 `NotReady` 状态，请先参阅与提供商无关的指南排查卡在 Provisioned 状态的业务集群。该指南介绍适用于所有 Alauda OS 基础设施提供商的通用诊断流程（`global` 集群导入控制器、`sentry` ServiceAccount、`clusters.platform.tkestack.io` 不变量）。

如果你已完成通用流程，并且也出现以下症状，请参阅本页介绍的 HCS 特定模式。

目录

模式：主机名错误导致 kubeadm init 始终无法完成

诊断

缓解

预防

相关内容

模式：主机名错误导致 kubeadm init 始终无法完成

症状 — 以下情况同时成立：

- `Cluster.status.phase=Provisioned`、`HCSCluster.status.ready=true`，ELB 已启动。
- `HCSMachine.status.instanceState=ACTIVE` 已填充 `InternalIP`，但 `Machine.status.nodeRef` 始终未设置。
- `KubeadmControlPlane.status.initialized` 为空，并且 `status.readyReplicas=0` 超过通常约 5 到 10 分钟的初始化时长。
- `cluster-api-provider-hcs` 控制器日志（位于 `cpaas-system`）反复显示针对控制平面 ELB VIP 的 6443 端口打印 `connect: connection refused`。
- 为节点选择的 `HCSMachineConfigPool.spec.configs[].hostname` 值中包含点号（FQDN 格式，例如 `master-1.example.org`）。

在早于 v1.0.1 的 `cluster-api-provider-hcs` 版本中，带点号的 `HCSMachineConfigPool.spec.configs[].hostname` 会在 cloud-init 中以完整 FQDN 字符串的形式写入 `hostname` 字段，并附带 `prefer_fqdn_over_hostname: true`。生成的节点具有包含点号的 POSIX 主机名，而 kubeadm init 无法处理这种主机名，因此 kube-apiserver 始终无法启动。

诊断

如果可以访问节点（HCS 控制台、跳板机或集群内调试 Pod）：

```
# On the affected node
hostname
hostname -f
cat /etc/hosts
sudo cat /var/lib/cloud/instance/user-data.txt | head -40
sudo journalctl -u cloud-init -b 0 | grep -E "hostname|fqdn|update_etc_hosts"
```

表明你遇到此模式的迹象：

- `hostname` 返回完整的带点号字符串，而不只是短标签。
- `hostname -f` 返回 `Name or service not known` 或 `Temporary failure in name resolution`。
- `/etc/hosts` 中不包含形如 `<node-ip> <fqdn> <short>` 的行。

- 节点上的 cloud-init user-data 显示 `prefer_fqdn_over_hostname: true`，且未设置 `manage_etc_hosts`。

缓解

将 `global` 集群上的 `cluster-api-provider-hcs` 插件升级到此版本提供的版本，然后触发受影响的控制平面和工作节点机器滚动替换，以便在全新启动的 VM 上运行新的 cloud-init。分步操作步骤请参阅在现有 HCS 集群上配置 FQDN 主机名。

在现有 Alauda OS 节点上手动修改 `/etc/hostname` 和 `/etc/hosts` 不会持久生效：每次启动时，cloud-init 都会重新生成这些文件，包括由 `transactional-update`、OS 修补或 `systemctl reboot` 触发的重启。滚动替换是受支持的迁移路径。

预防

使用此版本提供的 `cluster-api-provider-hcs` 插件创建集群可完全避免此问题。为 HCS 创建新清单时，在 `HCSMachineConfigPool` 中选择带点号或短主机名，请遵循集群创建页面上的节点上的主机名行为。

相关内容

- 排查卡在“已预配”状态的业务集群 — 与 provider 无关的 import-controller 诊断流程，适用于本页面中任何 HCS 特定模式之前。
- 在现有 HCS 集群上配置 FQDN 主机名 — 适用于在 `cluster-api-provider-hcs` v1.0.1 之前创建的集群的滚动替换迁移操作步骤。
- 在 Huawei Cloud Stack 上创建集群 — 新集群的清单参考，包括节点上的主机名行为。

VMware vSphere

概览

安装 **Provider**

基础设施资源

创建集群

VMware vSphere Provider

安装 Provider

安装 Provider

适用场景

先决条件

步骤

升级 Provider

Standby Management Clusters

后续步骤

基础设施资源

基础设施资源

场景

先决条件

如何使用此检查清单

参数来源与安全收集

术语

管理集群先决条件

vCenter 和模板先决条件

负载均衡器前提条件
集群基线参数
最低单数据中心参数
创建时可选的拓扑参数
最终就绪检查
后续步骤

创建集群

创建集群
Provider 要求
场景
如何使用本文档
先决条件
控制平面端点模式
关键对象
本地文件布局
操作步骤
验证
故障排除
创建时拓扑变体
后续步骤

用于虚拟化平台的基础设施 Provider 可在 VMware vSphere 平台上创建和管理 Alauda OS 集群。此 Provider 构建于 Cluster API Provider vSphere (CAPV) 之上，并与 vCenter 集成，为 Kubernetes 集群置备和管理虚拟机、网络及存储。

目录

安装 Provider
基础设施资源
创建集群
概述
主要功能
支持的资源
文档

概述

VMware vSphere 是 VMware 推出的业界领先虚拟化平台。vSphere Provider 直接连接 vCenter 以管理虚拟机生命周期，并通过声明式节点槽位分配扩展 CAPV，使主机名、静态 IP 地址和持久磁盘能够在集群置备前预先定义。

INFO

端点模式和磁盘生命周期

该 Provider 支持两种控制平面端点模式：外部 LoadBalancer（默认模式）以及由 Provider 管理的自建 VIP（`VSphereCluster.spec.controlPlaneLoadBalancer.type: internal`，使用 keepalived 和 IPVS 运行的 Alive）。自建 VIP 要求 ACP `v4.4` 或更高版本，并且只能在创建时选择：仅由 Provider `v1.0.17` 或更高版本创建的集群可用；已经存在的集群（包括其 Provider 从早期版本升级而来的集群）会继续使用外部 LoadBalancer，无法采用该模式。Machine-config-pool 持久磁盘遵循由 Provider 管理的 `InUse → Released → reuse or reclaim` 生命周期；请参阅 VMware vSphere Infrastructure Preparation。

主要功能

- **vCenter** 集成：直接连接 vCenter 以克隆虚拟机模板、管理虚拟机生命周期，并将工作负载放置在目标数据中心、计算集群和数据存储上
- 两种控制平面端点模式：使用您自行管理的外部 LoadBalancer，或在控制平面节点上使用由 Provider 管理的自建 VIP（`type: internal`，使用 keepalived 和 IPVS 运行的 Alive，其中引导 VIP 会在 `kubeadm init` 后移交给 Alive）；该模式要求 ACP `v4.4` 或更高版本。集群创建后无法更改此模式
- 机器配置池：`VSphereMachineConfigPool` 为每个节点定义具有固定主机名、静态 IP 和按槽位配置的数据磁盘布局的槽位。同一现有池中的替换 Machine 可以重新使用已释放的槽位，但不同池对象之间不会共享槽位和 VMDK
- 静态 **IP** 分配：为每个节点槽位配置静态 IP、网关和 DNS，适用于无法使用 DHCP 的企业网络环境
- 多 **NIC** 支持：每个节点支持主 NIC 和附加 NIC，其中主 NIC 用于驱动 kubelet `node-ip`，附加 NIC 则按声明顺序连接到管理、存储或服务网络
- 多个数据中心和故障域：通过 `VSphereFailureDomain` 和 `VSphereDeploymentZone` 将控制平面节点和工作节点分布到多个 vCenter 数据中心或计算集群中，以实现高可用性
- 灵活的数据磁盘：按声明为每个槽位配置持久和临时数据磁盘，并支持可选的格式化/挂载、持久磁盘上的 `wipeFilesystem` 控制、在 `/dev/disk/by-capv/<name>` 暴露的原始设备模式，以及延迟释放，以便替换 Machine 在 Provider 管理的回收之前重新使用槽位和 VMDK
- 不可变模板滚动更新：控制平面和工作节点更新使用新的 `VSphereMachineTemplate` 和 `KubeadmConfigTemplate` 资源，从而支持可复现的替换和相同次版本模板恢复

支持的资源

下表列出了本文档工作流中使用的主要面向用户的资源。控制器协调的资源（例如 `VSphereVM`）以及文档工作流中未使用的资源（例如 `VSphereClusterIdentity`，它是此处使用的基于 `Secret` 的 `identityRef` 的替代方案）也在 Provider APIs 参考文档中完整介绍。

资源	描述
`VSphereCluster`	表示 vSphere 基础设施，包括 vCenter 端点、控制平面端点和故障域选择器

资源	描述
`VSphereMachine`	表示 vSphere 中的虚拟机实例
`VSphereMachineTemplate`	用于创建 vSphere Machine 的模板，包含 VM、CPU、内存、网络和磁盘规格
`VSphereMachineConfigPool`	预定义节点槽位池，包含主机名、静态 IP 和每个槽位的数据磁盘布局
`VSphereFailureDomain`	描述用作故障域的数据中心、计算集群、数据存储和网络拓扑
`VSphereDeploymentZone`	将 `VSphereFailureDomain` 绑定到 vCenter 服务器，并设置资源池等放置约束

文档

- 安装指南
- 基础设施准备
- 创建集群
- 管理节点
- 升级 Kubernetes
- Provider APIs

安装 VMware vSphere Provider

本文介绍如何安装 Virtualization Platform 的 Infrastructure Provider 和 Cluster Bootstrap Provider，以及如何验证所需的 Cluster API 组件在 `global` 集群上可用。

INFO

provider 软件包包含 CAPV controller、vSphere CPI image、provider chart，以及用于强制执行固定 IP rollout 规则的 admission webhooks。有关这些规则，请参阅 Provider Requirements。

目录

适用场景
先决条件
步骤
下载所需软件包
上架软件包
安装集群插件
验证安装
升级 Provider
Standby Management Clusters
后续步骤

适用场景

在安装 `global` 集群后、在 VMware vSphere 上创建第一个业务集群之前完成此页面中的操作。无论 `global` 集群自身运行在哪种 IaaS 平台上，都必须执行此操作。在灾难恢复 (DR) 对中，请在两个 `global` 集群上完成此操作，以便备用集群在故障转移后能够协调相同的业务集群。将软件包上传到每个 `global` 集群。

这是一项独立于安装 global 集群中 provider 操作的任务。后者会将 provider 软件包上传到临时引导集群，以创建 `global` 集群本身。

先决条件

安装 provider 前，请确保满足以下条件：

- 你可以访问 灵雀云容器平台 `global` 集群。
- 你可以访问 Customer Portal 以下载 provider 软件包。
- 集群将通过 `VSphereCluster.spec.identityRef` 引用的 vCenter 账户可以列出、创建和设置 VM 自定义属性。授予 **Global → Manage custom attributes** 和 **Global → Set custom attribute** 权限。每当 `VSphereCluster` 或其 CAPI `Cluster` 携带任何 `vsphere.cluster.x-k8s.io/*` annotation 时，provider 都需要这些权限；如果缺少权限，VM reconciliation 会失败，而不是跳过元数据写入。
- 要创建或升级业务集群，请针对 active management cluster 操作，而不是灾难恢复备用集群。请参阅备用管理集群。

步骤

1 下载所需软件包

要在 VMware vSphere 上创建集群，需要两个集群插件。从 Customer Portal 下载这两个插件。对于 ACP 4.5，它们列在 **Common ACP Extensions** 下；请参阅下载。

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Virtualization Platform**

2 上架软件包

使用 `violet push` 将这两个软件包上架到平台。有关命令及其参数，请参阅上架软件包。

3 安装集群插件

在 `global` 集群上安装这两个集群插件。有关操作步骤，请参阅集群插件。

4 验证安装

检查模块状态：

```
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vsphere
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubeadm
```

两个插件都必须存在，且处于 `Running` 状态。

检查已注册的 CRD：

```
kubectl get crd | grep -E '^vsphere[a-z]+\.infrastructure\.cluster\.x-k8s\.io\b'
```

预期输出包含全部九个 CRD：

1. `vphereclusters.infrastructure.cluster.x-k8s.io`
2. `vphereclusteridentities.infrastructure.cluster.x-k8s.io`
3. `vphereclustertemplates.infrastructure.cluster.x-k8s.io`
4. `vspheredeploymentzones.infrastructure.cluster.x-k8s.io`
5. `vspherefailuredomains.infrastructure.cluster.x-k8s.io`
6. `vspheremachines.infrastructure.cluster.x-k8s.io`
7. `vspheremachineconfigpools.infrastructure.cluster.x-k8s.io`
8. `vspheremachinetemplates.infrastructure.cluster.x-k8s.io`

9. `vspherevms.infrastructure.cluster.x-k8s.io`

检查已注册的 admission webhook。`ValidatingWebhookConfiguration` 对象名称因软件包而异，因此请改为匹配 webhook 名称：

```
kubectl get validatingwebhookconfiguration \
  -o jsonpath='{range .items[*]}{range .webhooks[*]}{.name}{"\n"}{end}{end}' \
  | grep -E 'vsphere|capv'
```

预期输出包括：

- `validation.vsphirecluster.infrastructure.cluster.x-k8s.io`
- `validation.vspheremachineconfigpool.infrastructure.cluster.x-k8s.io`
- `validation.vspheremachinetemplate.infrastructure.cluster.x-k8s.io`
- `validation.kubeadmcontrolplane.capv.cluster.x-k8s.io`

最后一个 webhook 会强制执行 Provider Requirements 中所述的 `KubeadmControlPlane` 固定 IP rollout 规则。`MachineDeployment` 的相同规则仍由 slot 模型要求——不符合要求的 rollout 会因没有可用 slot 而停滞，而不是在前置检查阶段被拒绝。`vspheremachineconfigpool` webhook 还会拦截 `DELETE`，从而将删除仍拥有 `InUse` slot 的 pool 立即拒绝，而不是将其保留为处于 `Terminating` 状态的资源。

如果平台限制 ClusterRole，请检查管理集群权限。除上游权限集之外，provider chart 还会向 `capv-manager-role` 授予以下权限：

- 在 `cluster.alauda.io` 中读取 `clustermodules`、`moduleconfigs` 和 `moduleplugins` 的权限。
- 写入 `moduleinfoes` 以及读取 `moduleinfoes/status` 的权限。
- 在 `clusterregistry.k8s.io` 和 `platform.tkestack.io` 中读取 `clusters` 的权限。
- 对 `controlplane.cluster.x-k8s.io/kubeadmcontrolplanes` 执行 `patch` 和 `update` 的权限。
- 访问 `vspheremachineconfigpools` 的权限。

> `kubeadmcontrolplanes` patch 权限使 provider 能够设置清单中由 provider 管理的字段中所述的 CoreDNS 和 kube-proxy skip annotation。

升级 Provider

Provider 是一个 Aligned 扩展，会随 `global` 集群的 Distribution Version 升级一同升级。按照升级前准备中的说明，使用 `violet` 发布目标版本的 provider 软件包；随后，CVO 会在集群升级过程中升级已安装的集群插件。请参阅升级 `global` 集群的 Distribution Version。

升级还会更新 provider CRD。有些字段会在不同版本间迁移，但现有资源会自动迁移——请勿手动编辑 `VSphereMachineConfigPool` 或任何其他 provider 对象以匹配新架构，也不要删除并重新创建对象。升级后，控制器在首次协调时会将每个对象更新为当前布局：

- 不会触及控制平面端点。现有集群的 `spec.controlPlaneLoadBalancer` 未设置，这表示使用外部 LoadBalancer，并且会保持不变：API server 地址不会更改，也不会向业务集群部署任何内容。
- 以前记录在 `spec.configs[].persistentDisks[].volumePath` 和 `.diskUUID` 上的每磁盘状态，以及 `status.configStatuses[].reclaimStatus` 下的每 slot 回收状态，都会合并到 `status.persistentDiskStatuses[]`。正在进行的回收会保留其阶段和任务引用并继续执行，随后会清除 `reclaimStatus`，使状态只有一个真实数据源。
- 已弃用的 `spec` 字段会保留在对象中。请将其视为历史记录，并改为读取 `status.persistentDiskStatuses[]`。
- 迁移绝不会覆盖已设置的值，因此可以安全地重复执行。磁盘部分仅更新状态记录，并且会在联系 vCenter 前运行，因此即使 vCenter 无法访问，也能完成迁移。
- 迁移仅涵盖 provider 自身的资源。Cluster API 对象上必需的 rollout 值——`maxSurge` 和 `replicas`——绝不会被重写，因此，不包含这些值的现有 `KubeadmControlPlane` 或 `MachineDeployment` 仍需手动修正。请参阅检查清单是否符合准入规则。

不会触及业务集群中的任何内容，也不会替换任何 VM。升级前需要了解以下三项影响：

- 在较早版本的 provider 上创建的集群会保留其外部 LoadBalancer，且无法采用 Self-built VIP。端点是在部署时确定的：集群运行期间无法调整，升级也不是更改端点的时机——在运行中的集群上填写 `spec.controlPlaneLoadBalancer` 会被拒绝。请参阅控制平面端点模式。

- pool 会保留创建时使用的磁盘列表。`spec.configs[].ephemeralDisks[]` 是 provider `v1.0.17` 中新增的，但升级不会将任何现有磁盘移入其中：在较早版本的 provider 上创建的集群会继续将 `/var/cpaas`、`/var/lib/containerd` 和 `/var/lib/etcd` 作为持久磁盘。新集群所记录的持久磁盘与临时磁盘拆分布局是在创建时确定的。请参阅标准数据磁盘。
- 通过 `guestinfo` user data 传递的来宾系统端更改（例如 pod 日志重定位）只有在节点被替换时才会应用。请参阅重定位 pod 日志目录。

请使用验证安装中的检查确认结果，然后读取一个 pool：

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

升级前已预配的每个磁盘都会显示其阶段，并且 `status.configStatuses[].reclaimStatus` 已被清除。尚无任何 Machine 分配过的 slot 上的磁盘还没有 VMDK，因此不会有记录；这是预期情况，并不表示迁移未完成。

Standby Management Clusters

当 `etcd-sync` ConfigMap 存在于 **`kube-public`** 命名空间中时，provider 会将管理集群视为灾难恢复备用集群：

- 仍会协调由保留的 `global` 集群拥有的对象。
- 每隔 30 秒跳过并重新排队其他所有对象。
- 如果无法确定对象所属的集群名称，也会跳过该对象。

请勿在备用管理集群上创建或升级业务集群。 协调器会静默地不执行任何操作——不会更改条件，也不会在资源上显示错误，因此工作流看起来像是停滞，而不是被阻止。

开始前请检查：

```
kubectl -n kube-public get configmap etcd-sync -o name 2>/dev/null || ech
o "not standby"
```

一种常见的错误是检查 `kube-system`。ConfigMap 位于 `kube-public` 中。

有关灾难恢复工作流本身，请参阅 Global Cluster Disaster Recovery。

后续步骤

安装 provider 后，请继续：

- VMware vSphere Infrastructure Preparation
- 在 VMware vSphere 上创建集群
- 在 VMware vSphere 上升级 Kubernetes
- VMware vSphere Provider

VMware vSphere 基础设施准备

本文档将帮助你准备基础设施，并收集创建 VMware vSphere 业务集群所需的值。在应用 Create a Cluster on VMware vSphere 中的清单之前，请先完成此检查清单。

目录

- 场景
- 先决条件
- 如何使用此检查清单
- 参数来源与安全收集
  - 使用 govc 可选地发现 vCenter 信息
- 术语
  - 机器配置池
  - 节点槽位
  - 槽位网络布局
  - `deviceName`
  - vCenter 资源池
  - 计算集群
  - 数据存储
  - VM 模板
  - 指纹
- 管理集群先决条件
- vCenter 和模板先决条件
  - vCenter 连接信息

VM 模板要求
负载均衡器前提条件
集群基线参数
Registry tag 验证
验证本地预加载的镜像
最低单数据中心参数
数据中心和资源放置
主 NIC 参数
控制平面 machine config pool
Worker machine config pool
计算资源规格
磁盘大小模型
标准数据磁盘
持久磁盘字段
临时磁盘字段
持久磁盘生命周期
控制平面节点（每个节点 4 个磁盘）
Worker 节点（每个节点 3 个磁盘）
大小参数
vSphere CPI 参数
创建时可选的拓扑参数
多个数据中心和多个故障域
第二个 NIC 参数
最终就绪检查
后续步骤

场景

在以下场景中使用此检查清单：

- 你正在准备新的 VMware vSphere 集群部署。

- 你希望在开始部署之前验证外部依赖项。
- 你计划启用可选的创建时拓扑变体，例如多个数据中心、多个 NIC 或额外的工作节点。

先决条件

开始之前，请确保满足以下条件：

- 你可以使用 `global` 访问 `kubectl` 集群。
- 业务集群对象必须存储在 `cpaas-system` 命名空间中。
- 你可以访问目标 vCenter 清单、网络、数据存储和模板。

如何使用此检查清单

请按以下顺序使用此检查清单：

1. 收集本文档中列出的部署参数。
2. 将清单模板中的每个占位符替换为此处收集的实际值。
3. 如果某个值在多个清单中以占位符形式出现，请始终使用相同的值。
4. 如果未启用某项可选功能，请按照 Creation-Time Topology Variants 中的说明，准确省略相应的 YAML 块。
5. 如果不需要某个可选字段（例如 `deviceName`），请从 YAML 清单中删除整行。

参数来源与安全收集

将参数收集分为以下三个来源。可以从 vCenter 发现某个值，并不意味着该值就是产品支持的版本；同样，产品提供的值也不能替代网络或容量方面的决策。

来源	示例	收集规则
可自动发现	数据中心、计算集群、资源池、网络、数据存储、VM 模板和 vCenter 证书指纹	使用只读工具查询 vCenter，然后让 vSphere 管理员确认所选对象。

来源	示例	收集规则
产品确定	Kubernetes、Alauda OS 镜像、Kube-OVN chart、CoreDNS、etcd、provider chart 和 CPI 版本	选择针对目标 ACP release 发布的确切版本组合。使用 OS Support Matrix 确认组件版本，并通过 OS 下载页面获取 OS 镜像，同时参考 provider release notes 和交付的软件包元数据；不要独立选择最新的 Registry 标签。
operator 决定	集群和资源名称、端点、CIDR、DNS、节点数量、CPU、内存、磁盘容量和凭证	记录明确的设计决策及验证负责人。仅凭清单信息无法安全推断这些值。

WARNING

本节中的命令仅用于发现和检查对象是否存在。请勿根据其输出自动生成最终清单，也不要将 vCenter 密码、Kubernetes Secret 数据或其他凭证重定向到检查清单中。

使用 govc 可选地发现 vCenter 信息

如果 `govc` 已通过受保护的本地环境完成身份验证，请使用如下只读清单命令：

```
govc find / -type d
govc find / -type c
govc find / -type p
govc find / -type n
govc find / -type s
govc find / -type m
```

请将结果视为候选项，而非自动选择的对象：

- `d`：数据中心
- `c`：计算集群
- `p`：资源池
- `n`：网络或分布式端口组
- `s`：数据存储

- `m`：VM 和模板；请确认所选对象是模板

使用 Thumbprint 中的 `openssl` 命令检索证书指纹，而不显示 vCenter 凭证。

术语

以下术语在 VMware vSphere 集群创建文档中保持一致。

机器配置池

机器配置池是 `VSphereMachineConfigPool` 自定义资源。它预定义节点槽位。每个槽位可以包含：

- 节点主机名
- 目标数据中心
- 每个 NIC 的静态 IP 配置
- 持久磁盘定义

> **WARNING**
>
> 每个 `VSphereMachineConfigPool` 只能由单个 `KubeadmControlPlane` 或单个 `MachineDeployment` 引用。不要在多个控制平面组或 worker 组之间共享同一个 `VSphereMachineConfigPool`。如果某个池已绑定到其他使用者，`VSphereMachine` 将报告一个原因是 `PoolBoundToOtherConsumer` 的 `MachineConfigPoolReady=False` 条件。

节点槽位

节点槽位是 `VSphereMachineConfigPool.spec.configs[]` 下的一项。单个槽位通常映射到一个节点，例如 `cp-01` 或 `worker-01`。槽位 `hostname` 会驱动 Kubernetes 节点名称、kubelet 服务证书 DNS SAN，以及（与解析后的主 NIC 地址结合）kubelet 的 `node-ip`；它必须是有效的 DNS-1123 子域名。

槽位网络布局

每个槽位都在 `network.primary` 和 `network.additional` 下声明其 NIC 布局：

- `network.primary` 为必需项。必须设置其 `networkName`，该值用作 kubelet 的 node-ip 来源。
- `network.additional` 是可选的额外 NIC 列表，会按照列出的顺序合并到主 NIC 之后。

`VSphereMachineConfigPool.spec.configs[].network` 中的 `dns` 值描述每个槽位的静态网络元数据。在受影响的 VMware 部署中，这些值可能无法可靠地更新来宾操作系统的 `/etc/resolv.conf`。因此，部署清单还会通过 `KubeadmControlPlane.spec.kubeadmConfigSpec.files` 和 `KubeadmConfigTemplate.spec.template.spec.files` 同时写入 `/etc/resolv.conf`。

`deviceName`

`deviceName` 是 `VSphereMachineConfigPool` 网络配置中的可选字段。它用于控制来宾操作系统内部看到的 NIC 名称，例如 `eth0` 或 `eth1`。

填充值时请使用以下区分：

- `networkName` 是 vCenter 网络或端口组名称。
- `deviceName` 是来宾操作系统内部的 NIC 名称。
- 如果省略 `deviceName`，CAPV 通常会根据 NIC 顺序分配类似 `eth0`、`eth1` 和 `eth2` 的名称。

vCenter 资源池

vCenter 资源池是 vCenter 原生清单对象，例如：

```
/Datacenter1/host/cluster1/Resources
```

启用故障域创建变体时，`VSphereDeploymentZone.spec.placementConstraint.resourcePool` 会使用此路径。

计算集群

计算集群是目标 vCenter 计算集群名称。在本文档中，当 `VSphereFailureDomain` 映射到特定部署目标时，主要使用此名称。

数据存储

数据存储是用于存储 VM 磁盘的 vSphere 存储位置。系统磁盘和数据磁盘都必须置于具体的数据存储上。

VM 模板

VM 模板是用于创建节点虚拟机的源模板。启用多个数据中心时，同一模板必须已存在于每个目标数据中心中，并且必须能够通过相同的模板名称解析。

指纹

指纹是 vCenter 服务器证书的 SHA-1 指纹。CAPV 使用它来验证目标 vCenter 服务器。

使用以下命令检索该指纹：

```
openssl s_client -connect <vsphere_server>:443 -servername <vsphere_serve
r> </dev/null 2>/dev/null | openssl x509 -noout -fingerprint -sha1
```

管理集群先决条件

使用下表记录所需值和验证结果。

参数	占位符	必需	验证或备注
`global` 集群 kubeconfig 路径	-	是	使用此 kubeconfig 执行 `kubectl get ns` 成功。
业务对象命名空间	`<namespace>`	是	存储业务集群对象的命名空间，必须为 `cpaas-system`。

参数	占位符	必需	验证或备注
已安装 `cluster-api-provider-vsphere`	-	是	`kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vsphere` 返回结果。
已安装 `cluster-api-provider-kubeadm`	-	是	`kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubeadm` 返回结果。
已启用 `ClusterResourceSet=true`	-	是	`capi-controller-manager` 参数中包含 `ClusterResourceSet=true`

vCenter 和模板先决条件

vCenter 连接信息

参数	占位符	必需	验证或备注	示例
vCenter 服务器	`<vsphere_server>`	是	使用 vCenter IP 地址或 FQDN。	`vc.example.`
vCenter 用户名	`<vsphere_username>`	是	CAPV 使用此用户名向 vCenter 进行身份验证。	`svc-capv@example`
vCenter 密码	`<vsphere_password>`	是	CAPV 使用此密码向 vCenter 进行身份验证。	`******`
指纹	`<thumbprint>`	是	使用前文所示的 `openssl`	`AA:BB:CC:..`

参数	占位符	必需	验证或备注	示例
			命令检索。	

注意：这些文档假设使用默认的 vCenter HTTPS 端口 `443`。

VM 模板要求

参数	占位符	必需	验证或备注	示
VM 模板名称	`<template_name>`	是	用于克隆控制平面节点和工作节点。	`a` `k8`
vCenter 凭证 Secret 名称	`<credentials_secret_name>`	是	使用稳定的名称，例如 `<cluster_name>-vsphere-credentials`。	`d` `cl` `vs` `cr`
克隆模式	`<clone_mode>`	是	允许的值：`linkedClone`、`fullClone`。默认值为 `linkedClone`。`linkedClone` 要求 VM 模板至少有一个快照；如果不存在快照，CAPV 将回退到 `fullClone`。使用 `linkedClone` 时，`diskGiB` 将被忽略；系统磁盘保持模板的大小。如果需要使 `diskGiB` 生效，请选择 `fullClone`。	`l`
关机模式	`<power_off_mode>`	是	允许的值：`hard`、`soft`、	`t`

参数	占位符	必需	验证或备注	示
			`trySoft`。默认值为 `hard`。`soft` 和 `trySoft` 要求模板中存在 VMware Tools / `open-vm-tools`，以执行优雅关机；在 guest-soft-power-off 超时后，`trySoft` 将回退到 `hard`。	
SSH 公钥	`<ssh_public_key>`	是	注入控制平面节点和工作节点。	`s` `AA`

模板还应满足以下要求：

- 使用平台镜像策略支持的操作系统。
- 包含 `cloud-init`。
- 包含 VMware Tools 或 `open-vm-tools`。
- 包含 `containerd`。
- 包含 kubeadm 引导所需的基线组件。
- 在 `/root/images/` 下包含预先导出的容器镜像 tar 文件。这些文件会在运行 kubeadm 之前由 `capv-load-local-images.sh` 导入 containerd，因此节点引导不依赖于从远程 Registry 拉取镜像。
- tar 文件包含随 OS 模板提供的 kubeadm 控制平面镜像，包括 etcd 和 kube-apiserver，并且这些镜像使用根据 `clusterConfiguration.imageRepository`、Kubernetes 版本和组件镜像标签生成的确切镜像引用。尽管 etcd 和 kube-apiserver 作为静态 Pod 运行，但仍需要容器镜像。缺失或不匹配的本地引用会导致运行时改为尝试从配置的 Registry 获取镜像。
- `/root/images/*.tar` 文件必须包含 sandbox（pause）镜像，并且其引用必须与 `/etc/containerd/config.toml` 中配置的 `sandbox_image` 值（containerd v1）或 `sandbox` 值（containerd v2）完全匹配。例如，如果 containerd 配置为 `sandbox_image = "registry.example.com/tkestack/pause:3.10"`，则其中一个 tar 文件必须包含该确切的镜像引用。不匹配会导致 containerd 从网络拉取 sandbox 镜像，从而违背本地预加载的目的，并在 air-gapped 环境中失败。

负载均衡器前提条件

本节介绍基线集群使用的外部负载均衡器端点。请在创建集群之前完成配置，并遵循规划控制平面端点。也可以使用 provider 管理的 Self-built VIP 创建集群，此方式无需外部负载均衡器，并且要求 ACP `v4.4` 或更高版本；请参阅控制平面端点模式。

参数	占位符	必需	验证或备注	示例
控制平面 VIP	`<vip>`	是	前端地址已分配、稳定，并且始终为 IP 地址——外部负载均衡器 VIP，或 Self-built VIP 使用的 IPv4 地址。不支持将域名用作集群端点。	`10.1`
API 服务器端口	`<api_server_port>`	是	默认端口为 `6443`。	`6443`
监听器模式	-	是	使用 TLS 透传的 Layer 4 TCP 转发。	`TCP`
后端	-	是	每个计划中的控制平面节点 IP 都注册在后端端口 `6443` 上；不包含工作节点 IP。	`3 co` `plane`
健康检查	-	是	优先使用 HTTPS `/healthz`，并结合 HTTP `200`。如果设备只能执行 TCP 连接检查，请记录下来。	`HTTP` `/heal`
端点连通性	-	是	`global` 集群和每个控制平面节点都可以访问 `<vip>:6443`。	`Yes`
后端维护责任	-	是	明确在控制平面扩容、替换或删除后负责更新后端的人员。	`Plat` `team`

参数	占位符	必需	验证或备注	示例
虚拟路由器 ID	`<vrid>`	集群使用 Self-built VIP 时必需	Keepalived `virtual_router_id`、`1`–`255`，在控制平面的 Layer-2 域中必须唯一。	`51`
VIP 接口	`<vip_interface>`	集群使用 Self-built VIP 时必需	承载 VIP 的主机接口，最多 15 个字符。留空可让 provider 匹配承载节点主 IP 的接口；对于多 NIC 节点，请显式设置，因为之后无法更改。	`ens1`

集群基线参数

参数	占位符	必需	验证或备注
集群名称	`<cluster_name>`	是	在所有 manifest 中使用
Kubernetes 版本	`<k8s_version>`	是	使用目标平台版本所需的 1.34 或更早版本，基线 `imagePullCredentia` 对于 Kubernetes 1.35 或 和工作节点 manifest 中 `imagePullCredentia` `NeverVerify` 添加到 k
控制平面副本数	`<cp_replicas>`	是	必须为 `3` 或更多。pro webhook 会拒绝副本数 `KubeadmControlPlane` `replicas: 1` 控制平 Requirements。
工作节点副本数	`<worker_replicas>`	是	基线拓扑使用 `1`。

参数	占位符	必需	验证或备注
Pod CIDR	`<pod_cidr>`	是	不得与现有网络重叠。
Service CIDR	`<service_cidr>`	是	不得与现有网络重叠。
镜像 Registry	`<image_registry>`	是	kubeadm `imageRepos` 址，不包含 URL schem `<image_registry>/t`
集群 Registry 地址	`<registry_address>`	是	可从业务节点访问的永 `cpaas.io/registry-` `<port>`；不要使用已退 Registry，也不要追加
`kube-ovn` 版本	`<kube_ovn_version>`	是	必须符合平台网络插件
`kube-ovn-join-cidr`	`<kube_ovn_join_cidr>`	是	不得与其他网络重叠。
CoreDNS 镜像 tag	`<dns_image_tag>`	是	使用为目标 ACP 版本发 `v<acp>` 后缀逐字复制 本中，使用暂存于 `acp` 低于 `v4.4` 的版本中， `tkestack/coredns` 下 后缀来选择该仓库，因 `tkestack/coredns`， 高版本上拉取失败。不 `dns.imageRepository` Provider 管理的字段。
etcd 镜像 tag	`<etcd_image_tag>`	是	使用 Kubernetes 版本推 Support Matrix 行中获取
加密 provider Secret	`<encryption_provider_secret>`	是	用于静态加密 Secret 的 使用 `head -c 32 /de` 成。不要重复使用示例

Registry tag 验证

首先从 `global` 集群读取永久 Registry 地址，或向平台管理员确认等效地址：

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

Secret 命令仅验证是否存在，不会打印凭证数据。导出已确认的集群 Registry 地址，以及 CPI 和 kubeadm manifest 使用的镜像 Registry。如果两个占位符指向同一个 Registry，则对两个变量使用相同的值。

```
export CLUSTER_REGISTRY_ADDRESS="<registry_address>"
export IMAGE_REGISTRY_ADDRESS="<image_registry>"
# Repository names depend on the target version. The values below are the
# ACP v4.4 and later ones; for an earlier target, change them to:
#   Kube-OVN chart -> acp/chart-cpaas-kube-ovn
#   CoreDNS        -> tkestack/coredns
export KUBE_OVN_CHART_REPO="acp/chart-kube-ovn"
export COREDNS_REPO="acp/coredns"

curl -sk "https://${CLUSTER_REGISTRY_ADDRESS}/v2/${KUBE_OVN_CHART_REPO}/t
ags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/${COREDNS_REPO}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/acp/cloud-provider-vspher
e/tags/list"
```

`NAME_UNKNOWN` 响应表示仓库名称与目标版本不匹配；请根据上面的映射修正 `KUBE_OVN_CHART_REPO` 或 `COREDNS_REPO` 并重试，而不要将其解读为缺少 tag。

权威的 chart-name 边界位于控制平面之前升级 Kube-OVN。

`/v2/` 片段仅属于这些 Registry HTTP API URL。镜像引用使用 `<registry>/<repository>:<tag>`，而 Cluster annotation 使用 `<host>:<port>`。

确认响应包含计划使用的以下值：

- `<kube_ovn_version>`
- `<dns_image_tag>`
- `<cpi_image_tag>`

当 VM template 包含 etcd 或 kube-apiserver 镜像的精确镜像引用，并且 `capv-load-local-images.sh` 在 kubeadm 运行前导入这些镜像时，Registry 无需提供这些镜像。仅部署 Static Pod 并不足够：每当 OS template、Kubernetes 版本、`imageRepository` 或组件 tag 发生变化时，都要验证本地镜像。

验证本地预加载的镜像

使用确切的候选 template 创建并启动一个临时 VM。在该 VM 中运行以下命令，以执行 kubeadm 之前使用的相同导入路径，并列出 containerd 实际可用的镜像引用：

```
sudo bash -euo pipefail <<'EOF'
shopt -s nullglob
image_files=(/root/images/*.tar)
if (( ${#image_files[@]} == 0 )); then
  echo "ERROR: no tar files found in /root/images" >&2
  exit 1
fi

systemctl restart containerd
for image_file in "${image_files[@]}"; do
  ctr -n k8s.io images import "${image_file}"
done
ctr -n k8s.io images list -q | sort -u
EOF

sudo grep -nE '^[[:space:]]*(sandbox_image|sandbox)[[:space:]]*=' \
  /etc/containerd/config.toml
```

将第一条命令的输出与所选 Kubernetes 版本和 manifest 值所需的完整引用进行比较。确认每个必需的 kubeadm 镜像都已存在，包括根据 `<image_registry>/tkestack`、`<k8s_version>` 和组件 tag 生成的精确 etcd 与 kube-apiserver 引用。将第二条命令的 sandbox 引用与导入的 pause 镜像引用进行比较；两者必须完全一致。

每当 template、Kubernetes 版本、`imageRepository`、etcd tag 或 containerd sandbox 设置发生变化时，都要重新执行此验证。之后销毁验证 VM；不要将其转换回源 template。

在业务 `Cluster` annotation 中将 `cpaas.io/registry-address` 设置为 `<registry_address>`。

最低单数据中心参数

数据中心和资源放置

参数	占位符	必需	验证或说明
默认数据中心	`<default_datacenter>`	是	由基线拓扑使用。
VM 文件夹	`<vm_folder>`	是	从 `VSphereMachineTemplate` 作 VM 的文件夹路径。使用 `/<datacenter>/vm/<cluster_n` 以便操作员可以在 vCenter 中识别 属的集群。对于 `global` DR 部 `<cluster_name>` 下添加一个子 夹，以区分主集群和备用集群。

主 NIC 参数

参数	占位符	必需	验证或说明	示例
vCenter 网络名称	`<nic1_network_name>`	是	vCenter 中第一个网络或端口组的名称。	`pg-busines`
Guest NIC 名称	`<nic1_device_name>`	否	仅当需要强制指定 Guest NIC 名称时设置。如果不需要，请从 YAML manifest 中删除 `deviceName` 行。	`eth0`
网关	`<nic1_gateway>`	是	主 NIC 的默认网关。	`10.10.10.1`
前缀长度	`<nic1_prefix>`	是	与每个节点 IP 地址一起使	`24`

参数	占位符	必需	验证或说明	示例
			用。	
DNS 服务器 1	`<nic1_dns_1>`	是	主 NIC 的 DNS 服务器。	`10.10.0.10`

控制平面 machine config pool

参数	占位符	必需	验证或说明	示例
控制平面 pool 名称	`<cluster_name>-cp-pool`	是	控制平面节点的 machine config pool 名称。在 Machine 替换期间保持使用相同的现有 pool 引用。不同的 pool 对象拥有独立的 slot 和 VMDK。	`demo-cluster-pool`
控制平面节点 1 主机名	`<cp_node_name_1>`	是	第一个控制平面节点的节点名称和 kubelet serving cert SAN。必须是有效的 DNS-1123 子域名。	`cp-01`
控制平面节点 1 数据中心	`<master_01_datacenter>`	是	通常与默认数据中心相同。	`dc-a`
控制平面节点 1 IP 地址	`<master_01_nic1_ip>`	是	仅限 IPv4 地址，不包含前缀长度。	`10.10.1`
控制平面节点 2 主机名	`<cp_node_name_2>`	是	第二个控制平面节点的节点名称和 kubelet	`cp-02`

参数	占位符	必需	验证或说明	示例
			serving cert SAN。必须是有效的 DNS-1123 子域名。	
控制平面节点 2 数据中心	`<master_02_datacenter>`	是	通常与默认数据中心相同。	`dc-a`
控制平面节点 2 IP 地址	`<master_02_nic1_ip>`	是	仅限 IPv4 地址，不包含前缀长度。	`10.10.1`
控制平面节点 3 主机名	`<cp_node_name_3>`	是	第三个控制平面节点的节点名称和 kubelet serving cert SAN。必须是有效的 DNS-1123 子域名。	`cp-03`
控制平面节点 3 数据中心	`<master_03_datacenter>`	是	通常与默认数据中心相同。	`dc-a`
控制平面节点 3 IP 地址	`<master_03_nic1_ip>`	是	仅限 IPv4 地址，不包含前缀长度。	`10.10.1`

Worker machine config pool

参数	占位符	必需	验证或说明	示
Worker pool 名称	`<cluster_name>-worker-pool`	是	Worker 节点的 machine config pool 名称。在 Machine 替换期间保持使用相同的现有 pool 引用。不同的 pool 对象	`d` `cl` `wo`

参数	占位符	必需	验证或说明	示
			拥有独立的 slot 和 VMDK。	
Worker 节点 1 主机名	`<worker_node_name_1>`	是	第一个 Worker 节点的节点名称和 kubelet serving cert SAN。必须是有效的 DNS-1123 子域名。	`w`
Worker 节点 1 数据中心	`<worker_01_datacenter>`	是	通常与默认数据中心相同。	`d`
Worker 节点 1 IP 地址	`<worker_01_nic1_ip>`	是	仅限 IPv4 地址，不包含前缀长度。	`1`
Worker 节点 2 主机名	`<worker_node_name_2>`	否	扩展 Worker pool 时使用。	`w`
Worker 节点 2 数据中心	`<worker_02_datacenter>`	否	扩展 Worker pool 时使用。	`d`
Worker 节点 2 IP 地址	`<worker_02_nic1_ip>`	否	扩展 Worker pool 时使用。	`1`

参数	占位符	必需	验证或说明	示
`releaseDelayHours`	`<release_delay_hours>`	是	provider 从未使用的 `Released` slot 回收持久 VMDK 前的延迟时间。默认值为 `24`。引用同一现有 pool 的替换 Machine 可以立即重新使用 `Released` slot；无需等待此延迟结束。	`2`

计算资源规格

参数	占位符	必需	验证或说明	示例
控制平面 CPU	`<cp_num_cpus>`	是	每个控制平面节点的 CPU 数量。	`4`
控制平面内存 MiB	`<cp_memory_mib>`	是	每个控制平面节点的内存。	`819`
控制平面系统 datastore	`<cp_system_datastore>`	是	控制平面系统磁盘使用的 datastore。	`dat` `cp`
控制平面系统磁盘大小 GiB	`<cp_system_disk_gib>`	是	基线示例值为 `300`。当 `<clone_mode>` 为 `linkedClone` 时忽略；此时系统磁盘保持模板中的大小。	`300`

参数	占位符	必需	验证或说明	示例
Worker CPU	`<worker_num_cpus>`	是	每个 Worker 节点的 CPU 数量。	`2`
Worker 内存 MiB	`<worker_memory_mib>`	是	每个 Worker 节点的内存。	`409`
Worker 系统 datastore	`<worker_system_datastore>`	是	Worker 系统磁盘使用的 datastore。	`dat` `work`
Worker 系统磁盘大小 GiB	`<worker_system_disk_gib>`	是	基线示例值为 `300`。当 `<clone_mode>` 为 `linkedClone` 时忽略；此时系统磁盘保持模板中的大小。	`300`

磁盘大小模型

`VSphereMachineTemplate.spec.template.spec.diskGiB` 和 `VSphereMachineConfigPool.spec.configs[]` 下的两个磁盘列表定义不同的磁盘。

字段	磁盘类型	含义
`VSphereMachineTemplate.spec.template.spec.diskGiB`	系统磁盘	在由 `ful` 或系统磁 像模板中 的总和。
`VSphereMachineConfigPool.spec.configs[].persistentDisks[]`	持久数据磁盘	绑定到节 VMDK， 开。
`VSphereMachineConfigPool.spec.configs[].ephemeralDisks[]`	临时数据磁盘	随 VM 一 VMDK，

字段	磁盘类型	含义
		`/var/li` 和 `/var/`

不要将数据磁盘大小加到 `diskGiB` 中。例如，如果 OS 镜像具有 100 GiB 系统磁盘，并且节点还需要一个 100 GiB 持久磁盘和三个临时磁盘，请使用至少为 100 GiB 的系统磁盘值，并在 `persistentDisks` 和 `ephemeralDisks` 中声明数据磁盘。将 `diskGiB` 设置为合并大小，会在数据磁盘之外创建一个过大的系统磁盘。

标准数据磁盘

每个节点角色都需要一组专用数据磁盘。下面列出的磁盘是所需的最少磁盘数。如果工作负载需要，您可以向任一列表追加其他磁盘。在第一个 Machine 占用 slot 之前，规划好磁盘、磁盘大小和挂载路径：处于 `InUse` 或 `Released` 状态的 slot 会拒绝移除持久磁盘，或更改其 `sizeGiB` 或 `mountPath`。

这种划分遵循 Huawei Cloud Stack 和 Huawei DCS provider 使用的相同模型：只有必须在 VM 替换后保留的平台状态（`/var/cpaas`）才使用持久磁盘，而可重建的运行时目录（`/var/lib/kubelet`、`/var/lib/containerd`、`/var/lib/etcd`）会随每个 VM 重新创建。

> **WARNING**
>
> 此划分仅适用于新集群
>
> `ephemeralDisks[]` 在 provider `v1.0.17` 中新增，因此此布局是在创建时确定的选项。在较早 provider 上创建的集群会将 `/var/cpaas`、`/var/lib/containerd` 和 `/var/lib/etcd` 声明为持久磁盘，并将 `wipeFilesystem: true` 放在 etcd 磁盘上，同时保留该布局。provider 升级不会迁移任何磁盘。请参阅升级 Provider。

持久磁盘字段

字段	必填	默认值	描述
`name`	是	—	slot 内唯一的磁盘名称。

字段	必填	默认值	描述
`sizeGiB`	是	—	磁盘大小，单位为 GiB。
`mountPath`	否	（空）	guest OS 内的挂载路径。如果为空，则磁盘会作为原始设备附加，并在 `/dev/disk/by-capv/<name>` 处创建符号链接，但不会进行格式化或挂载。当外部进程在运行时管理磁盘时，此设置很有用。
`mountOptions`	否	（空）	在设置 `mountPath` 时应用的挂载选项。
`fsFormat`	否	`ext4`（设置 `mountPath` 时）	文件系统格式。当 `mountPath` 为空时忽略。
`wipeFilesystem`	否	`false`	当 `true` 时，磁盘内容会在新 VM 首次启动时被擦除。重启和手动重启服务不受影响。标准磁盘集不使用此设置：唯一的持久磁盘 `/var/cpaas` 用于在替换过程中保留其内容，而所有必须以空磁盘启动的磁盘都在 `ephemeralDisks[]` 中声明。
`unitNumber`	否	（由 controller 分配）	用于在 VM 重新创建过程中保持 guest 设备顺序稳定的 SCSI 单元。取值范围为 `0` 到 `15`，不能为 `7`（该值保留给 SCSI controller），并且在 slot 内必须唯一。
`datastore`	否	（pool 默认值）	覆盖此磁盘使用的 datastore。
`storagePolicy`	否	（pool 默认值）	覆盖此磁盘使用的 vSphere 存储策略。

临时磁盘字段

`ephemeralDisks[]` 镜像持久磁盘面向 guest 的字段——`name`、`sizeGiB`、`mountPath`、`mountOptions`、`fsFormat`、`datastore` 和 `storagePolicy`——并省略持久性字段。不存在 `wipeFilesystem`，因为后端 VMDK 始终会新建，并从头开始格式化；也不存在 `unitNumber`，因为 controller 会在克隆时分配 SCSI 单元，并在 `status.ephemeralDiskStatuses[]` 中报告该单元。

在一个 slot 内，两个列表中的磁盘名称和挂载路径共享同一命名空间。有关逐字段比较以及应使用哪个列表的指导，请参阅选择持久磁盘或临时磁盘。

持久磁盘生命周期

持久磁盘归属于特定 `VSphereMachineConfigPool` 中的 slot，而不属于生命周期短暂的 VM。即使另一个 pool 声明了相同的主机名、IP 地址或磁盘名称，其中的 slot 仍属于不同的分配。预期的替换生命周期如下：

1. 当 Machine 使用该 slot 时，`status.configStatuses[].state` 为 `InUse`，VMDK 会附加到其 VM。
2. 当 Machine 和 VM 被删除时，slot 变为 `Released`。VMDK 仍可供替换使用，不会成为永久孤立的磁盘。
3. 引用同一现有 pool 的替换 Machine 可以立即重新使用 `Released` slot。provider 会优先选择已释放的 slot，而不是未使用的 slot，并将声明的持久 VMDK 重新附加到替换 VM。`releaseDelayHours` 不是重新使用前的等待时间。
4. 如果 slot 在 `releaseDelayHours` 到期前一直未使用，controller 可以使其重新可用，并且只有在确认 VMDK 不再附加后才回收持久 VMDK。异步进度会显示在 `status.persistentDiskStatuses[].phase` 中。

`wipeFilesystem` 控制持久磁盘附加到新 VM 后的数据处理；它不控制 slot 选择或 datastore 回收。`false` 保留现有文件系统内容。`true` 会在新 VM 首次启动期间删除已挂载的内容；它不会加快 VMDK 回收速度，也不得将其视为 datastore 容量设置。

重新部署和拆除容量

使用不同的 `metadata.name` 创建新的 `VSphereMachineConfigPool` 会创建一组独立的 slot。这些 slot 不会重新使用早期 pool 所拥有的 VMDK。因此，使用不同 pool 名称反复拆除和重新部署，可能会暂时消耗多组完整的持久磁盘，直到所有早期 pool 完成 provider 管理的回收。

重试或拆除部署时，请遵循以下规则：

- 对于同一集群内的 Machine 替换或滚动更新，请让每个 `VSphereMachineTemplate.spec.template.spec.machineConfigPoolRef.name` 始终指向同一现有 pool。更改 `KubeadmControlPlane.spec.machineNamingStrategy` 不会在 pool 之间转移磁盘，也不是 datastore 清理控制项。
- 如果新部署有意使用新的 pool 名称，请将新 pool 的完整持久磁盘分配纳入 datastore 容量规划。不要假设相匹配的主机名或磁盘名称可以支持跨 pool 重用。
- 在容量受限的 datastore 上开始另一次部署之前，请确认早期 pool 对象已完成删除，并且其 VMDK 已被回收。pool 删除是异步的，因为 finalizer 会等待 Machine 被移除并等待持久磁盘回收完成。
- 不要移除 pool finalizer，也不要手动删除后端 VMDK 来加速拆除。如果回收没有进展，请先收集 pool 状态、Events、provider-controller 日志和 vCenter 附加状态，再将该情况视为 provider 缺陷。
- 为两个列表都规划 datastore 容量。临时 VMDK 会在其 VM 存在期间占用 datastore 空间，但会随 VM 一起删除，因此不会跨 pool 累积，也不会阻止 slot 释放。只有持久 VMDK 会在拆除后保留，并等待 provider 管理的回收。

调查已释放的 slot 或 datastore 容量告警时，请使用以下字段：

状态字段	含义
`state`	Slot 分配状态：`Available`、`InUse` 或 `Released`。
`lastReleasedTime`	slot 进入 `Released` 的时间。将其与 `releaseDelayHours` 进行比较。
`reclaimStatus.*`	已弃用。provider 升级时会将其合并到 `status.persistentDiskStatuses[]`，然后清除该字段。

从 `status.persistentDiskStatuses[]` 读取每个磁盘的状态。其 `phase` 字段包含 `Creating`、`Attached`、`Available`、`Reclaiming`、`Reclaimed` 或 `Error`，并且每个条目还会报告 `lastError`、`retryAfter`、`taskRef` 以及所属 machine。`status.configStatuses[].reclaimStatus` 子树以及 `spec.configs[].persistentDisks[].volumePath` 和 `.diskUUID` 字段已弃用、冻结，并计划移除。现有 pool 会由 provider 自身迁移到 `persistentDiskStatuses[]`；请参阅升级 Provider。

检查 pool 和相关 Events，而无需读取凭证：

```
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> -o yaml
kubectl -n <namespace> describe vspheremachineconfigpool <pool_name>
kubectl -n <namespace> get events \
  --field-selector involvedObject.kind=VSphereMachineConfigPool,involvedO
bject.name=<pool_name>
```

WARNING

当 slot 处于 `Released` 状态时，或其 `persistentDiskStatuses[].phase` 为 `Reclaiming` 或 `Error` 时，不要手动删除 VMDK。替换 Machine 可能仍会重新使用该 VMDK，或者 provider 可能正在重试已验证的回收操作。在执行任何手动 datastore 清理之前，请确认 pool 状态、保留窗口、附加状态和 controller 状态。

控制平面节点（每个节点 4 个磁盘）

磁盘名称	挂载路径	磁盘列表	用途	建议最
`var-cpaas`	`/var/cpaas`	`persistentDisks[]`	存储日志、审计数据和其他平台数据	100 G
`var-lib-kubelet`	`/var/lib/kubelet`	`ephemeralDisks[]`	存储 kubelet 工作数据，包括 pod 卷和插件套接字	100 G
`var-lib-containerd`	`/var/lib/containerd`	`ephemeralDisks[]`	存储 containerd 运行时数据（镜像、层、快照）	100 G
`var-lib-etcd`	`/var/lib/etcd`	`ephemeralDisks[]`	存储 etcd 数据。每个新 VM 都会创建一个空磁盘，因此替换后的控制	20 GiB

磁盘名称	挂载路径	磁盘列表	用途	建议最
			平面节点不会将过时的成员数据带入 `kubeadm join`。	

Worker 节点（每个节点 3 个磁盘）

磁盘名称	挂载路径	磁盘列表	用途	建议最
`var-cpaas`	`/var/cpaas`	`persistentDisks[]`	存储日志、审计数据和其他平台数据	100 G
`var-lib-kubelet`	`/var/lib/kubelet`	`ephemeralDisks[]`	存储 kubelet 工作数据，包括 pod 卷和插件套接字	100 G
`var-lib-containerd`	`/var/lib/containerd`	`ephemeralDisks[]`	存储 containerd 运行时数据（镜像、层、快照）	100 G

大小参数

参数	占位符	必填	验证
CP `/var/cpaas` 大小（GiB）	`<cp_var_cpaas_size_gib>`	是	建议 100
CP `/var/lib/kubelet` 大小（GiB）	`<cp_var_lib_kubelet_size_gib>`	是	建议 100

参数	占位符	必填	验证
CP `/var/lib/containerd` 大小（GiB）	`<cp_var_lib_containerd_size_gib>`	是	建议 100
CP `/var/lib/etcd` 大小（GiB）	`<cp_var_lib_etcd_size_gib>`	是	建议 20 G
Worker `/var/cpaas` 大小（GiB）	`<worker_var_cpaas_size_gib>`	是	建议 100
Worker `/var/lib/kubelet` 大小（GiB）	`<worker_var_lib_kubelet_size_gib>`	是	建议 100
Worker `/var/lib/containerd` 大小（GiB）	`<worker_var_lib_containerd_size_gib>`	是	建议 100

vSphere CPI 参数

参数	占位符	必填	验证或备注
CPI 数据中心列表	`<cpi_datacenters>`	是	启用多个数据中心时，包含每个目标数据中心。
CPI 镜像标签	`<cpi_image_tag>`	是	vSphere CPI 组件的标签。完整引用为 `<image_registry>/acp/cloud-provider-vsphere:<cpi_image_tag>`。

创建时可选的拓扑参数

多个数据中心和多个故障域

参数	占位符	必填	验证或备注
数据中心 1 故障域名称	`<fd_name_1>`	否	仅在启用故障域时必填。
数据中心 1 部署区域名称	`<dz_name_1>`	否	仅在启用故障域时必填。
计算集群名称	`<compute_cluster_1>`	否	仅在启用故障域时必填。
默认数据存储	`<default_datastore_1>`	否	仅在启用故障域时必填。
vCenter 资源池路径	`<resource_pool_path_1>`	否	仅在启用故障域时必填。必须存在于 vCenter 清单中。
数据中心 2 名称	`<dc_name_2>`	否	仅用于多个数据中心或故障域。
数据中心 2 故障域名称	`<fd_name_2>`	否	仅用于多个数据中心或故障域。
数据中心 2 部署区域名称	`<dz_name_2>`	否	仅用于多个数据中心或故障域。
数据中心 2 计算集群	`<compute_cluster_2>`	否	仅用于多个数据中心或故障域。
数据中心 2 默认数据存储	`<default_datastore_2>`	否	仅用于多个数据中心或故障域。
数据中心 2 vCenter 资源池路径	`<resource_pool_path_2>`	否	仅用于多个数据中心或故障域。
Worker 部署区域	`<worker_failure_domain>`	否	使用 `VSphereDeploymentZone` 名称，而不是 `VSphereFailureDomain` 名称。

参数	占位符	必填	验证或备注
数据中心 3 名称	`<dc_name_3>`	否	仅用于其他数据中心或故障域场景。
数据中心 3 故障域名称	`<fd_name_3>`	否	仅用于其他数据中心或故障域场景。
数据中心 3 部署区域名称	`<dz_name_3>`	否	仅用于其他数据中心或故障域场景。
数据中心 3 计算集群	`<compute_cluster_3>`	否	仅用于其他数据中心或故障域场景。
数据中心 3 默认数据存储	`<default_datastore_3>`	否	仅用于其他数据中心或故障域场景。
数据中心 3 vCenter 资源池路径	`<resource_pool_path_3>`	否	仅用于其他数据中心或故障域场景。

第二个 NIC 参数

参数	占位符	必填	验证或备注	示例
次要网络名称	`<nic2_network_name>`	否	仅在节点需要第二个 NIC 时使用。	`pg-management`
次要 guest NIC 名称	`<nic2_device_name>`	否	仅在需要强制指定 guest NIC 名称时设置。	`eth1`
次要网关	`<nic2_gateway>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.10.1`
次要前缀长度	`<nic2_prefix>`	否	与每个次要 NIC IP 地址配合使用。	`24`

参数	占位符	必填	验证或备注	示例
次要 DNS 服务器 1	`<nic2_dns_1>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.0.10`
控制平面节点 1 次要 IP	`<master_01_nic2_ip>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.10.1`
控制平面节点 2 次要 IP	`<master_02_nic2_ip>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.10.1`
控制平面节点 3 次要 IP	`<master_03_nic2_ip>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.10.1`
Worker 节点 1 次要 IP	`<worker_01_nic2_ip>`	否	仅在节点需要第二个 NIC 时使用。	`10.20.10.2`
Worker 节点 2 次要 IP	`<worker_02_nic2_ip>`	否	在扩展 Worker 且需要第二个 NIC 时使用。	`10.20.10.2`

最终就绪检查

开始部署前，请确认以下所有项目：

1. `global` 集群可访问。
2. 两个集群插件均已安装：Alauda Container Platform Cluster Bootstrap Provider 和适用于 Virtualization Platform 的 Alauda Container Platform Infrastructure Provider。
3. `ClusterResourceSet=true` 已启用。
4. 已收集 vCenter 服务器、用户名、密码和 thumbprint。
5. 外部 LoadBalancer 满足 Layer 4 监听器、后端、运行状况检查、所有权和可访问性要求。如果集群改用 Self-built VIP，则已满足 Self-built VIP 前提条件，并已确定 `<vrid>` 和

`<vip_interface>`。

6. Pod CIDR、Service CIDR 和 `kube-ovn-join-cidr` 不与现有网络重叠。
7. VM 模板在每个所需的数据中心中均可用。
8. 所需的数据存储和 vCenter 资源池路径已确认。
9. 永久平台 Registry 包含所需的 Kube-OVN、CoreDNS 和 vSphere CPI 标签。
10. VM 模板包含所选 Kubernetes 和组件版本所需的、预加载到本地的 etcd 和 kube-apiserver 镜像的准确引用。
11. `cpaas.io/registry-address` 的值规划为不带 scheme 的 `<registry_address>`，或规划为 `/v2/` 路径。
12. 控制平面和 Worker 的 bootstrap manifest 均通过 kubeadm `files` 写入 `/etc/resolv.conf`。
13. 最小单数据中心拓扑的 machine config pool 值均已填写完整。
14. 基准系统磁盘和数据磁盘大小已确认，每个所需磁盘均已声明在正确的列表中（`/var/cpaas` 为持久磁盘；`/var/lib/kubelet`、`/var/lib/containerd` 和 `/var/lib/etcd` 为临时磁盘），且数据磁盘大小未计入 `diskGiB`。
15. 在分配数据存储容量前，已了解 `releaseDelayHours`、同池持久磁盘复用、新池容量分配以及由 provider 管理的拆除回收。
16. 每个所需参数均具有实际值。

后续步骤

完成此检查清单后，继续阅读在 VMware vSphere 上创建集群。

在 VMware vSphere 上创建集群

本文介绍如何通过向 ACP `global` 管理集群应用 Cluster API 清单，创建 VMware vSphere 业务集群。该操作步骤不会创建 `global` 集群本身。本文介绍一种受支持的最小拓扑，其中包含一个数据中心、每个节点一个 NIC，并通过 `VSphereMachineConfigPool` 分配静态 IP。

目录

- Provider 要求
- 场景
- 如何使用本文档
- 先决条件
- 控制平面端点模式
- 关键对象
  - `ClusterResourceSet`
  - vSphere CPI 组件
  - Provider 管理的清单字段
  - machine config pool
  - VM 模板要求
- 本地文件布局
- 操作步骤
  - 验证环境
  - 创建命名空间和 vCenter 凭证 Secret
  - 创建 `Cluster` 和 `VSphereCluster` 对象
  - 创建 vSphere CPI 交付资源

创建 machine config pool
应用可选的故障域对象
创建控制平面对象
创建 worker 对象
等待集群就绪
验证
故障排除
Admission 拒绝
创建时拓扑变体
使用 Self-built VIP
Self-built VIP 前提条件
`controlPlaneLoadBalancer` 字段
在 `10-cluster.yaml` 中启用 Self-built VIP
验证 Self-built VIP
添加第二个 NIC
多个数据中心和故障域
添加数据磁盘
选择持久磁盘或临时磁盘
固定 SCSI 单元号
重定位 pod 日志目录
验证创建时变体
后续步骤

Provider 要求

从管理集群上的 provider `AppRelease` 中读取已安装的 provider 版本；不要根据 ACP 版本推断。请参阅识别已安装的 Provider 版本。

当控制器引用的 `VSphereMachineTemplate` 设置 `machineConfigPoolRef` 时，以下规则适用——本页介绍的固定 IP 拓扑。Provider 的准入 webhook 会对 `CREATE` 和 `UPDATE` 强制执行 `KubeadmControlPlane` 和 pool 规则，并对 `DELETE` 强制执行最后一条规则。

- `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge` 是整数 `0`，且 `spec.replicas` 为 `3` 或更大。未设置 `maxSurge` 时，表示采用 Cluster API 默认值 `1`，该值不被接受。
- `MachineDeployment.spec.strategy.rollingUpdate.maxSurge` 是 `0`。采用默认的 `maxSurge` 值 `1` 时，滚动更新需要额外的槽位，而已满的 pool 不具备该槽位，因此更新会停滞。`strategy.type: OnDelete` 无需设置 surge。
- `VSphereMachineConfigPool.spec.configs` 至少声明一个槽位，且每个槽位都声明 `network`。绑定到同一个 `Cluster` 的所有 pool 中，主机名和主 IP 地址必须唯一；一个 pool 仅服务于一个控制器。
- 已处于 `InUse` 或 `Released` 状态的槽位会拒绝某些更改。请参阅已分配槽位的限制。
- 仍拥有 `InUse` 槽位的 pool 无法删除。

使用验证安装中的 webhook 检查，确认环境中已注册 webhook。如果 webhook 未注册，仍需满足相同的值要求：固定 IP 槽位模型没有可供 surge 机器使用的空闲槽位，因此不符合要求的滚动更新会停滞，而不是在开始前被拒绝。

场景

在以下场景中使用本文档：

- 您希望在环境中创建第一个基线 VMware vSphere 业务集群。
- 初始验证时，您使用一个数据中心，并为每个节点配置一个 NIC。
- 您希望在启用高级置放或网络功能之前，让首次部署尽可能简单。

本文档适用于以下部署模型：

- CAPV 直接连接到 vCenter。
- 控制平面节点和 worker 节点均使用 `VSphereMachineConfigPool` 分配静态 IP 和数据磁盘。
- `ClusterResourceSet` 自动交付 vSphere CPI 组件。
- 首次验证使用一个数据中心，并为每个节点配置一个 NIC。

本文档不适用于以下场景：

- 依赖 vSphere Supervisor 或 `vm-operator` 的部署。
- 不使用 `VSphereMachineConfigPool` 的部署。

本文档针对当前平台环境编写。`kube-ovn` 交付路径依赖于平台控制器读取 `Cluster` 资源上的 annotation，因此本文档介绍的操作流程并非面向平台环境之外的通用独立 CAPV 部署指南。

如何使用本文档

1. 完成基础设施和参数检查清单。
2. 按照下文的操作步骤准备基线清单文件。
3. 应用清单之前，仅添加所需的创建时拓扑变体。
4. 应用完整的清单集并完成验证检查。
5. 集群运行后，使用在 VMware vSphere 上管理节点进行扩容、机器模板更改和运行时拓扑更改。

基线是验证参考。如果目标集群需要多个可选拓扑功能，请一次引入并验证一项清单更改，再将它们组合使用。

先决条件

开始前，请确保满足以下条件：

1. Cluster Bootstrap Provider 和 Infrastructure Provider for Virtualization Platform 已上传并安装到 `global` 集群。无论 `global` 集群本身运行在哪种 IaaS 平台上，此要求都适用。请参阅安装 Provider。
2. 您已完成 VMware vSphere 基础设施准备。
3. `global` 集群可以访问 vCenter。
4. 目标模板、网络、数据存储和 vCenter 资源池均可用。
5. 已选择控制平面端点模式。对于外部 LoadBalancer，需满足端点契约。对于 Self-built VIP，需满足 Self-built VIP 先决条件。集群创建后无法更改此模式，因此请现在做出决定：请参阅控制平面端点模式。

6. 所有必需的静态 IP 地址均已分配、未被使用，并且未被同一 `Cluster` 绑定的其他 `VSphereMachineConfigPool` 中的 slot 声明。
7. 已启用 `ClusterResourceSet=true`。
8. 已确定此集群使用的镜像 registry。默认情况下，集群继承 `global` 集群的 registry；如果该 registry 需要身份验证，则必须提供有效的 `public-registry-credential` Secret。若要改为从专用 registry 拉取平台镜像，请在创建集群前按照为业务集群选择镜像 Registry 操作；之后无法更改此绑定。
9. 上述选定的 registry 可从业务节点访问，并包含所需的 Kube-OVN、CoreDNS 和 vSphere CPI 标签。
10. 平台能够处理安装网络插件所需的集群 annotation。
11. `VSphereCluster.spec.identityRef` 引用的 vCenter 帐户满足安装 VMware vSphere Provider 中的 Custom Fields 权限要求。
12. `global` 集群是活动管理集群，而不是灾难恢复备用集群。请参阅备用管理集群。
13. 已规划任何专用节点池的角色标签和污点。它们声明在节点池的资源上，具体说明请参阅基础设施节点。

控制平面端点模式

每个集群都通过稳定的端点访问其 API server。编写 `10-cluster.yaml` 前，请先选择模式，因为之后无法更改。此选择会记录在 `VSphereCluster.spec.controlPlaneLoadBalancer` 中；不填写该字段本身也是一种选择：字段缺省表示使用外部 LoadBalancer。

> **Self-built VIP 仅可在创建时选择**
>
> Self-built VIP 从 provider `v1.0.17` 起提供，需要 ACP `v4.4` 或更高版本，并且仅适用于创建时启用了该功能的集群。现有集群无法启用此功能；provider 从早期版本升级到 `v1.0.17` 的集群也无法启用：VIP 会在第一个控制平面节点引导时注入，随后由 Alive 接管，因此无法原地迁移。这些集群会继续使用外部 LoadBalancer；若要改用 Self-built VIP，必须创建新集群。请参阅升级 Provider。

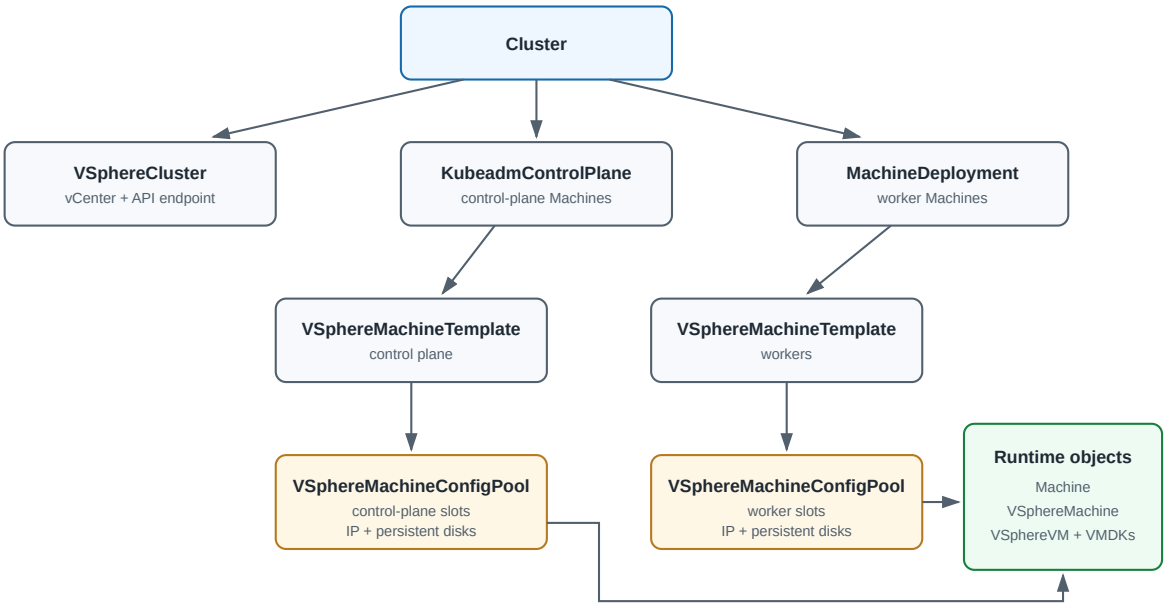
`spec.controlPlaneLoadBalancer`	端点由谁管理
缺省（本页的基线）	由您管理。配置外部 Layer 4 TCP LoadBalancer，并自行填写 `spec.controlPlaneEndpoint`。
`type: external`	由您管理。provider 会将该值与 `spec.controlPlaneEndpoint` 进行校验，且不会部署 VIP 组件。
`type: internal`	由 provider 管理。它会在第一个控制平面节点上注入引导 VIP，随后在业务集群中安装 Alive（keepalived 和 IPVS），并在控制平面节点上持有 VIP。

集群创建后，这三种模式都无法更改。该字段只能写入一次；请参阅 `controlPlaneLoadBalancer` 字段。

建议：对于外部 LoadBalancer，请勿填写该字段，而不是写入 `type: external`。字段缺省即表示使用外部 LoadBalancer；仅为记录此意图而填写该字段，会将端点固定为相同地址的冻结副本。

对于 `type: internal`，先决条件、字段参考、清单代码块和验证命令请参阅使用 Self-built VIP。

关键对象



顶层的 `Cluster` 引用 `VSphereCluster` 以及控制平面或 worker 控制器。这些控制器引用 `VSphereMachineTemplate` 对象。每个模板引用一个 `VSphereMachineConfigPool`，其插槽为运行时的 `Machine`、`VSphereMachine` 和 `VSphereVM` 对象提供主机名、静态网络配置和持久磁盘。

阶段	输入	应用	预期结果	首先诊
端点和 Registry	LoadBalancer、永久 Registry、版本集	验证外部依赖项。	端点和 Registry 可从所需网络访问。	LoadB DNS、口规则
集群根对象	网络 CIDR、vCenter 身份、端点	应用 `Cluster` 和 `VSphereCluster`。	两个资源均存在，并报告正在推进的 condition。	`Clust` `VSphe`
CPI 交付	vCenter CPI 配置和凭证	应用 `ClusterResourceSet` 资源。	workload cluster 的 API 可访问后，集群会接收 vSphere CPI。	`Clust` 和 CPI
节点插槽	主机名、静态 IP、数据存储、持久磁盘	应用两个 `VSphereMachineConfigPool` 对象。	每个池均为 Ready，并公开预期的插槽。	池 con `statu`
控制平面	VM 模板、kubeadm 配置、Kubernetes 版本	应用 `VSphereMachineTemplate` 和 `KubeadmControlPlane`。	三个控制平面 Machine 变为 Ready。	KCP、 `VSphe` `VSphe`
Worker	Worker 模板和引导配置	应用 `KubeadmConfigTemplate` 和 `MachineDeployment`。	Worker Machine 和 Node 变为 Ready。	Machi VM 状

`ClusterResourceSet`

`ClusterResourceSet` 是 `global` 集群中的 Cluster API 资源。workload API server 变为可访问后，它会将所引用的 `ConfigMap` 和 `Secret` 资源应用到 workload cluster。

在此工作流中，使用 `ClusterResourceSet` 自动交付 vSphere CPI 资源。

vSphere CPI 组件

vSphere CPI 组件通过 `ClusterResourceSet` 交付到 workload cluster。它将 workload 节点连接到 vSphere 基础设施，使集群能够报告基础设施身份并完成云 provider 初始化。

Provider 管理的清单字段

您所应用的对象中，有些字段在创建后由 provider 管理。了解这些字段可以避免升级期间发生仓库冲突。

- 您应用 `KubeadmControlPlane` 后，provider 会对其进行 patch。预期 CoreDNS 和 kube-proxy 跳过 annotation 会被设置为 `"true"`，并且 `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageRepository` 会由控制器设置。
- 请自行设置 **`dns.imageTag`**，并保持 **`dns.imageRepository`** 未设置。 提供目标 ACP 版本发布的 CoreDNS 标签：对于 ACP `v4.4` 及更高版本，使用暂存于 `acp/coredns` 下的标签；对于低于 `v4.4` 的版本，使用 `tkestack/coredns` 下的标签。provider 会读取标签中的 `-v<acp>` 后缀来选择该仓库，因此不带该后缀的标签会解析到 `tkestack/coredns`；在 ACP `v4.4` 或更高版本中，该仓库不包含此标签，CoreDNS Pod 会停留在 `ImagePullBackOff`。
- kube-proxy 仓库根据 Kubernetes 版本派生：在 `v1.35` 及更高版本中为 `acp/k8s`，低于该版本时为 `tkestack`。在 chart `v4.4` 及更高版本中，Kube-OVN chart 为 `acp/chart-kube-ovn`，低于该版本时为 `acp/chart-cpaas-kube-ovn`。
- 三者的 Registry 前缀均来自 `Cluster` annotation `cpaas.io/registry-address`。
- workload cluster 的 `kube-system/kube-proxy` DaemonSet 和 `kube-system/coredns` Deployment 镜像仓库也会进行 reconcile，但仅在控制平面 **rollout** 完成后进行。`KubeadmControlPlane` 达到目标版本与这些工作负载更改仓库之间存在延迟是正常现象。
- 这些受管理组件的镜像拉取凭证来自 workload cluster 的 `cpaas-system/sentry` ServiceAccount。

WARNING

请勿删除 CoreDNS 和 kube-proxy 跳过 annotation，也不要重新应用省略这些 annotation 的已保存 `KubeadmControlPlane` 清单。在 provider 同时管理 CoreDNS 或 kube-proxy 的情况下，将其交还给 kubeadm 管理会产生仓库冲突，导致镜像来回切换。

machine config pool

machine config pool 是 `VSphereMachineConfigPool` 自定义资源。在基线工作流中：

- 一个 machine config pool 用于控制平面节点。
- 一个 machine config pool 用于 worker 节点。

每个节点插槽都包含主机名、数据中心、静态 IP 分配以及可选的数据磁盘定义。

对于网络配置，请区分以下字段：

- `networkName` 是 vCenter 网络或端口组名称。
- `deviceName` 是 guest 操作系统内的 NIC 名称。

如果设置了 `deviceName`，CAPV 会将该值写入生成的 guest 网络元数据。如果未设置，当前实现通常会按 NIC 顺序使用 `eth0`、`eth1` 和 `eth2` 等 NIC 名称。

还请区分以下值格式：

- 节点 IP 地址与前缀长度搭配使用，例如 `10.10.10.11/24`。
- 网关字段仅包含网关 IP 地址，例如 `10.10.10.1`。

`spec.configs` 至少必须声明一个插槽，并且每个插槽都必须声明 `network`，且其中的 `network.primary.networkName` 不得为空。

每个插槽可以声明两个磁盘列表。`persistentDisks[]` 会在 VM 替换后保留，并重新挂载到替换之前机器的那台机器上；`ephemeralDisks[]` 会随 VM 一起删除，并以空磁盘重新创建。标准集合将 `/var/cpaas` 声明为持久磁盘，将 `/var/lib/kubelet`、`/var/lib/containerd` 和 `/var/lib/etcd` 声明为临时磁盘。在一个插槽内，两个列表中的磁盘名称和挂载路径共享同一个命名空间。`ephemeralDisks[]` 在

provider `v1.0.17` 中新增，因此此拆分仅适用于在 `v1.0.17` 或更高版本上创建的集群。请参阅选择持久磁盘或临时磁盘。

一个池只能服务一个控制器。 机器分配插槽时，provider 会在 `status.consumerRef` 中记录所属的 `KubeadmControlPlane` 或 `MachineDeployment`。引用同一池的第二个控制器不会被接受。控制平面使用一个池，并为每个 `MachineDeployment` 使用一个池。

VM 模板要求

此工作流使用的 VM 模板应满足以下最低要求：

1. 它使用目标平台环境所需的操作系统。
2. 它包含 `cloud-init`。
3. 它包含 VMware Tools 或 `open-vm-tools`。
4. 它包含 `containerd`。
5. 它包含 kubeadm 引导所需的基线组件。
6. 它在 `/root/images/` 下包含预先导出的容器镜像 tar 文件。这些文件会在 kubeadm 运行前由 `capv-load-local-images.sh` 导入 containerd，因此节点引导不依赖于从远程 Registry 拉取镜像。
7. 这些 tar 文件包含 OS 模板随附的 kubeadm 控制平面镜像，包括 etcd 和 kube-apiserver；镜像位于 `clusterConfiguration.imageRepository`、Kubernetes 版本和组件镜像标签生成的准确镜像引用下。尽管 etcd 和 kube-apiserver 作为静态 Pod 运行，但它们仍需要容器镜像。缺少本地引用或引用不匹配时，运行时会改为尝试从配置的 Registry 拉取镜像。
8. `/root/images/*.tar` 文件必须包含 sandbox（pause）镜像，且其引用必须与 `/etc/containerd/config.toml` 中配置的 `sandbox_image` 值（containerd v1）或 `sandbox` 值（containerd v2）完全匹配。例如，如果 containerd 配置为 `sandbox_image = "registry.example.com/tkestack/pause:3.10"`，则其中一个 tar 文件必须包含该准确的镜像引用。不匹配会导致 containerd 从网络拉取 sandbox 镜像，这会违背本地预加载的目的，并在 air-gapped 环境中失败。

静态 IP 配置、主机名注入和其他初始化设置取决于 `cloud-init`。节点 IP 报告取决于 guest tools。

本地文件布局

> **业务集群命名**
>
> 业务 `cluster_name` 不得为 `global`。该名称保留给 `global` 集群，重复使用会导致业务集群的资源与 `cpaas-system` 中 `global` 集群的资源发生冲突。`global-` 前缀保留给 `global` 集群 DR 工作流所拥有的资源；请参阅通用先决条件。请勿将 `global-` 用于业务集群资源，因为故障转移操作可能会将这些资源选作属于 `global` 集群的资源。
>
> 按照惯例，CAPI `Cluster` 和 provider 集群资源（`VSphereCluster`）应始终命名为 `<cluster_name>`；非根 CAPI 和 provider 资源（`KubeadmControlPlane`、`KubeadmConfigTemplate`、`MachineDeployment`、`VSphereMachineTemplate`、`VSphereMachineConfigPool` 等）应使用 `<cluster_name>-` 作为前缀——例如，示例清单使用 `<cluster_name>-kcp` 和 `<cluster_name>-md-0`。这是建议，而非控制器强制执行的规则，但这样可以避免多个业务集群位于 `cpaas-system` 时发生同一命名空间内的冲突，并使操作期间的资源归属一目了然。

创建本地工作目录，并按以下布局存放清单：

```
capv-cluster/
├── 00-namespace.yaml
├── 10-cluster.yaml
├── 15-vsphere-cpi-clusterresourceset.yaml
├── 16-vspheremachineconfigpool-control-plane.yaml
├── 17-vspheremachineconfigpool-worker.yaml
├── 18-failure-domains.yaml  # only when failure domains are enabled
├── 20-control-plane.yaml
└── 30-workers-md-0.yaml
```

使用以下命令创建目录：

```
mkdir -p ./capv-cluster
cd ./capv-cluster
```

操作步骤

1 验证环境

从 `global` 集群运行以下命令，以验证最低先决条件：

```
kubectl get ns
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vsphere
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubeadm
kubectl -n cpaas-system get deploy capi-controller-manager -o jsonpath='{.spec.template.spec.containers[0].args}'
kubectl -n cpaas-system get secret public-registry-credential -o name
kubectl -n cpaas-system get cluster global \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}{"\n"}'
kubectl -n kube-public get configmap etcd-sync -o name 2>/dev/null || echo "not standby"
```

导出 `Cluster` 注解返回的永久平台 Registry 地址，以及 CPI 和 kubeadm 清单使用的镜像 Registry。如果两个占位符指向同一个 Registry，则两个变量使用相同的值。

```
export CLUSTER_REGISTRY_ADDRESS="<registry_address>"
export IMAGE_REGISTRY_ADDRESS="<image_registry>"
# Repository names depend on the target version. The values below are the
# ACP v4.4 and later ones; for an earlier target, change them to:
#   Kube-OVN chart -> acp/chart-cpaas-kube-ovn
#   CoreDNS        -> tkestack/coredns
export KUBE_OVN_CHART_REPO="acp/chart-kube-ovn"
export COREDNS_REPO="acp/coredns"

curl -sk "https://${CLUSTER_REGISTRY_ADDRESS}/v2/${KUBE_OVN_CHART_REPO}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/${COREDNS_REPO}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/acp/cloud-provider-vsphere/tags/list"
```

如果响应为 `NAME_UNKNOWN`，表示仓库名称与目标版本不匹配；请根据上方映射更正 `KUBE_OVN_CHART_REPO` 或 `COREDNS_REPO`，然后重试，而不要将其视为缺少标签。

上面列出的仓库边界与 provider 在运行时应用的边界相同；有关选择规则，请参阅清单中由 provider 管理的字段；有关权威图表名称表，请参阅在控制平面之前升级 Kube-OVN。

`/v2/` 片段仅是 Registry HTTP API URL 的一部分。在业务集群 `Cluster` 清单中，将 Registry 注解设置为 `<registry_address>`，不带 scheme 或路径。

确认以下结果：

- `global` 集群可访问。
- Alauda Container Platform Cluster Bootstrap Provider 和 Alauda Container Platform Infrastructure Provider for Virtualization Platform 均在运行。
- 控制器参数包含 `ClusterResourceSet=true`。
- `public-registry-credential` Secret 存在；此操作步骤不会打印其内容。
- `global` 集群不是灾难恢复备用集群。`kube-public/etcd-sync` ConfigMap 不得存在。请参阅备用管理集群。
- 永久平台 Registry 包含所需的 Kube-OVN、CoreDNS 和 vSphere CPI 标签。
- VM 模板通过了可执行的本地镜像验证，其中包含准确的 etcd、kube-apiserver 和 sandbox 镜像引用。以静态 Pod 方式部署并不会免除镜像要求。

继续之前，还请验证以下事项：

- vCenter 服务器地址可访问。
- vCenter 用户名和密码有效。
- 指纹正确。
- 模板名称正确。
- 可在目标数据中心解析该模板。
- 如果 VM 克隆为 `fullClone`，模板系统磁盘不得大于清单稍后使用的 `diskGiB` 值。如果 CAPV 完成 `linkedClone`，系统磁盘大小将保持为模板的大小，并忽略 `diskGiB`。
- 模板中已安装 VMware Tools 或 `open-vm-tools`。

- 外部 LoadBalancer 符合规划控制平面端点中有关第 4 层侦听器、后端、运行状况检查、可达性和所有权的要求。
- 如果选择了 `type: internal`，则应满足自建 VIP 先决条件：VIP 尚未被占用；dvPortGroup 安全策略或 NSX-T 分布式防火墙规则未阻止 VRRP 和免费 ARP；模板能够加载 `ip_vs`；并且 `alive` 插件可部署到 `global` 集群。

2 创建命名空间和 vCenter 凭证 Secret

创建用于存储业务集群对象的命名空间。

此工作流将业务集群对象存储在 `cpaas-system` 命名空间中。在以下清单和命令中，将每个 `<namespace>` 占位符替换为 `cpaas-system`。

00-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
```

应用命名空间：

```
kubectl apply -f 00-namespace.yaml
```

创建 `VSphereCluster.spec.identityRef` 引用的 vCenter 凭证 Secret。

请使用编辑器编写凭证文件，而不要在 shell 提示符中输入这些值。shell 会将整个命令（包括密码）记录在历史记录中；使用编辑器写入文件则不会留下此类副本。请将文件保存在 `capv-cluster/` 之外，以免被误认为清单。

vsphere-credentials.env

```
username=<vsphere_username>
password=<vsphere_password>
```

每行的格式均为 `key=value`。请勿为值加引号——`kubectl` 会将引号字符按原样处理。值中可以包含 `=`；每行中只有第一个 `=` 用于分隔键和值。

限制文件权限，从文件创建 Secret，然后删除文件：

```
chmod 600 ../vsphere-credentials.env

VSPHERE_SECRET_NAME="<credentials_secret_name>"
VSPHERE_NAMESPACE="<namespace>"

kubectl create secret generic "${VSPHERE_SECRET_NAME}" \
  --namespace "${VSPHERE_NAMESPACE}" \
  --from-env-file=../vsphere-credentials.env

rm -f ../vsphere-credentials.env
```

> **请勿使用 kubectl apply 创建凭证 Secret**
>
> 客户端侧 apply 会将其应用的清单（包括 vCenter 密码）存储在 Secret 自身的 `kubectl.kubernetes.io/last-applied-configuration` 注解中。`kubectl create` 不会写入此类注解。无论值是以 `stringData` 还是 base64 `data` 形式写入，情况都一样；base64 是编码，不是保护。本文后续的 CPI 凭证也适用相同规则。

3 创建 `Cluster` 和 `VSphereCluster` 对象

创建基础集群清单，其中包含业务集群网络设置、控制平面端点和 vCenter 连接设置。将 `cpaas.io/registry-address` 设置为永久平台 Registry，格式为 `<registry_address>`（仅限 `<host>:<port>`）。

以下清单使用外部 LoadBalancer，这是本页的基准配置。对于自建 VIP，请将 在 `10-cluster.yaml` 中启用自建 VIP 中的 `spec.controlPlaneLoadBalancer` 块添加到 `VSphereCluster` 对象中。

10-cluster.yaml

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster_name>
  namespace: <namespace>
  labels:
    cluster.x-k8s.io/cluster-name: <cluster_name>
    cluster-type: VSphere
    addons.cluster.x-k8s.io/vsphere-cpi: "enabled"
    # Optional. Bind this cluster to a dedicated image registry instead of the
    # registry of the `global` cluster. Read only at cluster creation.
    # cpaas.io/registry-reference: <registry_credential_name>
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: VSphereCluster
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
    cpaas.io/network-type: kube-ovn
    cpaas.io/kube-ovn-version: <kube_ovn_version>
    cpaas.io/kube-ovn-join-cidr: <kube_ovn_join_cidr>
    cpaas.io/registry-address: <registry_address>
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - <pod_cidr>
    services:
      cidrBlocks:
      - <service_cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster_name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: VSphereCluster
    name: <cluster_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereCluster
```

```
metadata:
  name: <cluster_name>
  namespace: <namespace>
spec:
  controlPlaneEndpoint:
    host: "<vip>"
    port: <api_server_port>
  identityRef:
    kind: Secret
    name: <credentials_secret_name>
  server: "<vsphere_server>"
  thumbprint: "<thumbprint>"
```

应用清单：

```
kubectl apply -f 10-cluster.yaml
```

4 创建 vSphere CPI 交付资源

创建 `ClusterResourceSet`，以便业务集群 API 服务器变为可访问后，业务集群自动接收 vSphere CPI 配置和清单。

> **INFO**
>
> 在基准工作流中，`VSphereCluster.spec.failureDomainSelector` 有意不设置，且 CPI `vsphere.conf` 不包含 `[Labels]` 块。只有启用故障域后才需要同时配置这两项，具体说明请参阅多个数据中心和故障域。在 `vsphere.conf` 中添加 `[Labels]`，但没有与之匹配的 `VSphereFailureDomain` 对象，会导致 CPI 查找不存在的 zone 和 region 标签。

> **INFO**
>
> vSphere CPI 的 TLS 绕过选项为 `insecure-flag`。请在 `<cluster_name>-vsphere-cpi-config` 的 `[Global]` 部分中保留 `insecure-flag = "1"`，以便在 vCenter 证书为自签名证书或业务集群节点不信任该证书时，CPI 仍能连接。对于未设置自身 `insecure-flag` 的 vCenter 条目，CPI 会应用全局值。

WARNING

CPI 的 `ConfigMap`、`Secret` 和 `ClusterResourceSet` 资源必须与 `Cluster` 资源位于同一命名空间中。本指南使用的命名空间为 `cpaas-system`。`ClusterResourceSet` 只能匹配自身命名空间中的集群；将其部署到其他命名空间会导致资源交付被静默阻止。

INFO

`Cluster` 注解中的 `kube-ovn` 配置由平台控制器使用。本文档不会直接安装网络插件。

TIP

此清单较长，且在 `data` 字段中嵌套了 YAML。应用前请先验证清单：`kubectl apply --dry-run=client -f 15-vsphere-cpi-clusterresourceset.yaml`。

此清单声明两个 ConfigMap 和 `ClusterResourceSet`。它不会声明 `ClusterResourceSet` 也会引用的 CPI 凭证 Secret——该 Secret 包含 vCenter 密码，将在下一步单独创建。

15-vsphere-cpi-clusterresourceset.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <cluster_name>-vsphere-cpi-config
  namespace: <namespace>
data:
  data: |
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: cloud-config
      namespace: kube-system
    data:
      vsphere.conf: |
        [Global]
        secret-name = "vsphere-cloud-secret"
        secret-namespace = "kube-system"
        service-account = "cloud-controller-manager"
        port = "443"
        insecure-flag = "1"
        datacenters = "<cpi_datacenters>"

        [VirtualCenter "<vsphere_server>"]
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: <cluster_name>-vsphere-cpi-manifests
  namespace: <namespace>
data:
  data: |
    apiVersion: v1
    kind: ServiceAccount
    metadata:
      name: cloud-controller-manager
      namespace: kube-system
    automountServiceAccountToken: false
    ---
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: system:cloud-controller-manager
    rules:
```

```
    - apiGroups: [""]
      resources: ["events"]
      verbs: ["create", "patch", "update"]
    - apiGroups: [""]
      resources: ["nodes"]
      verbs: ["*"]
    - apiGroups: [""]
      resources: ["nodes/status"]
      verbs: ["patch"]
    - apiGroups: [""]
      resources: ["services"]
      verbs: ["list", "patch", "update", "watch"]
    - apiGroups: [""]
      resources: ["services/status"]
      verbs: ["patch"]
    - apiGroups: [""]
      resources: ["serviceaccounts"]
      verbs: ["create", "get", "list", "watch", "update"]
    - apiGroups: [""]
      resources: ["persistentvolumes"]
      verbs: ["get", "list", "update", "watch"]
    - apiGroups: [""]
      resources: ["endpoints"]
      verbs: ["create", "get", "list", "watch", "update"]
    - apiGroups: [""]
      resources: ["secrets"]
      verbs: ["get", "list", "watch"]
    - apiGroups: ["coordination.k8s.io"]
      resources: ["leases"]
      verbs: ["get", "list", "watch", "create", "update"]
    ---
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: servicecatalog.k8s.io:apiserver-authentication-reader
      namespace: kube-system
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: Role
      name: extension-apiserver-authentication-reader
    subjects:
    - apiGroup: ""
      kind: ServiceAccount
      name: cloud-controller-manager
```

```
    namespace: kube-system
- apiGroup: ""
    kind: User
    name: cloud-controller-manager
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
    name: system:cloud-controller-manager
roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: system:cloud-controller-manager
subjects:
- kind: ServiceAccount
    name: cloud-controller-manager
    namespace: kube-system
- kind: User
    name: cloud-controller-manager
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
    annotations:
      scheduler.alpha.kubernetes.io/critical-pod: ""
    labels:
      component: cloud-controller-manager
      tier: control-plane
      k8s-app: vsphere-cloud-controller-manager
    name: vsphere-cloud-controller-manager
    namespace: kube-system
spec:
    selector:
      matchLabels:
        k8s-app: vsphere-cloud-controller-manager
    updateStrategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          component: cloud-controller-manager
          k8s-app: vsphere-cloud-controller-manager
      spec:
        securityContext:
```

```
            runAsUser: 1001
          automountServiceAccountToken: true
          # Optional: required when the CPI image is stored in a priv
ate
          # registry that needs authentication. The platform automati
cally
          # syncs a dockerconfigjson secret named "global-registry-au
th"
          # into every namespace of the workload cluster when the
          # `global` cluster secret "public-registry-credential"
          # (data.content) is configured. If your environment does no
t
          # use a private registry, remove the imagePullSecrets bloc
k.
          imagePullSecrets:
          - name: global-registry-auth
          serviceAccountName: cloud-controller-manager
          hostNetwork: true
          tolerations:
          - operator: Exists
          - key: node.cloudprovider.kubernetes.io/uninitialized
            value: "true"
            effect: NoSchedule
          - key: node-role.kubernetes.io/master
            effect: NoSchedule
          - key: node.kubernetes.io/not-ready
            effect: NoSchedule
            operator: Exists
          containers:
          - name: vsphere-cloud-controller-manager
            image: <image_registry>/acp/cloud-provider-vsphere:<cpi_i
mage_tag>
            args:
            - --v=2
            - --cloud-provider=vsphere
            - --cloud-config=/etc/cloud/vsphere.conf
            volumeMounts:
            - mountPath: /etc/cloud
              name: vsphere-config-volume
              readOnly: true
            resources:
              requests:
                cpu: 200m
          volumes:
```

```
          - name: vsphere-config-volume
            configMap:
              name: cloud-config
    ---
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        component: cloud-controller-manager
      name: vsphere-cloud-controller-manager
      namespace: kube-system
    spec:
      type: NodePort
      ports:
      - port: 43001
        protocol: TCP
        targetPort: 43001
      selector:
        component: cloud-controller-manager
---
apiVersion: addons.cluster.x-k8s.io/v1beta1
kind: ClusterResourceSet
metadata:
  name: <cluster_name>-vsphere-cpi
  namespace: <namespace>
spec:
  strategy: Reconcile
  clusterSelector:
    matchLabels:
      addons.cluster.x-k8s.io/vsphere-cpi: "enabled"
  resources:
  - name: <cluster_name>-vsphere-cpi-config
    kind: ConfigMap
  - name: <cluster_name>-vsphere-cpi-secret
    kind: Secret
  - name: <cluster_name>-vsphere-cpi-manifests
    kind: ConfigMap
```

创建 CPI 凭证 Secret。它封装了 vSphere CPI 在业务集群上读取的 `vsphere-cloud-secret`，因此其负载中包含 vCenter 密码。出于本页前面所述的原因，请使用编辑器编写负载，并将其保存在 `capv-cluster/` 之外，以免被误认为清单：

vsphere-cpi-secret-payload.txt

```
apiVersion: v1
kind: Secret
metadata:
  name: vsphere-cloud-secret
  namespace: kube-system
type: Opaque
stringData:
  <vsphere_server>.username: <vsphere_username>
  <vsphere_server>.password: <vsphere_password>
```

限制文件权限，从文件创建 Secret，然后删除文件：

```
chmod 600 ../vsphere-cpi-secret-payload.txt

CLUSTER_NAME="<cluster_name>"

kubectl create secret generic "${CLUSTER_NAME}-vsphere-cpi-secret" \
  --namespace "${VSPHERE_NAMESPACE}" \
  --type=addons.cluster.x-k8s.io/resource-set \
  --from-file=data=../vsphere-cpi-secret-payload.txt

rm -f ../vsphere-cpi-secret-payload.txt
```

此处使用的 `<cluster_name>` 应与本页其他位置一致。创建 vCenter 凭证时已设置 `VSPHERE_NAMESPACE`；如果在新的 shell 中操作，请重新设置。Secret 的唯一键必须为 `data`，该键由 `--from-file=data=` 生成。

然后应用清单：

```
kubectl apply -f 15-vsphere-cpi-clusterresourceset.yaml
```

5 创建 machine config pool

创建控制平面 machine config pool。

INFO

每个节点槽位都在 `network.primary`（必需）和 `network.additional`（可选列表）下声明其 NIC 布局。主 NIC 的 `networkName` 为必需项，provider 会根据 `hostname` 和解析出的主 NIC 地址，推导出 Kubernetes 节点名称、kubelet 服务证书 DNS SAN 以及 kubelet `node-ip`。`hostname` 必须是有效的 DNS-1123 子域名。

INFO

`deviceName` 为可选项。如果不需要强制指定 guest NIC 名称，请从每个节点槽位中删除 `deviceName` 行。provider 会根据 NIC 顺序分配 `eth0`、`eth1` 等 NIC 名称。

WARNING

`VSphereMachineConfigPool` 中的 `dns` 条目会保留在静态网络配置中，但在受影响的 VMware 部署中，它们可能无法可靠地更新 guest 操作系统的 `/etc/resolv.conf`。因此，以下控制平面和 worker 引导清单也会通过 kubeadm `files` 显式写入 `/etc/resolv.conf`。

槽位验证规则

provider 会验证以下内容：

- `spec.configs` 至少声明一个槽位，且每个槽位都声明了 `network`，其中 `network.primary.networkName` 不为空。
- 每个磁盘声明的 `sizeGiB` 不小于 `1`。
- 如果设置了持久磁盘的 `unitNumber`，其值必须介于 `0` 和 `15` 之间，且绝不能为 `7`，因为该值保留给 SCSI 控制器。
- 在一个槽位内，`persistentDisks` 和 `ephemeralDisks` 中的磁盘名称和挂载路径必须唯一。
- 在池内以及同一命名空间中绑定到同一 `Cluster` 的所有其他池之间，主机名和主 IP 地址都必须唯一。

已分配槽位限制

槽位的 `status.configStatuses[].state` 一旦变为 `InUse` 或 `Released`，就不能移除该槽位、更改其 `network.primary.ip` 或 `.ipv6`、从 `persistentDisks[]` 中移除磁盘，也不能更改持久磁盘的 `sizeGiB` 或 `mountPath`。

请勿向状态为 `InUse` 或 `Released` 的槽位添加磁盘：provider 无法将磁盘挂接到正在运行的节点，并会报告该节点的错误；而添加的持久磁盘在节点被替换之前无法再次移除。请在第一个 Machine 接管槽位之前规划该槽位的所有磁盘。

请在第一个 Machine 分配槽位之前规划主机名和地址。有关扩容和运行时更改，请参阅管理 VMware vSphere 上的节点。

`16-vspheremachineconfigpool-control-plane.yaml`

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineConfigPool
metadata:
  name: <cluster_name>-cp-pool
  namespace: <namespace>
spec:
  clusterRef:
    apiVersion: cluster.x-k8s.io/v1beta1
    kind: Cluster
    name: <cluster_name>
  datacenter: "<default_datacenter>"
  releaseDelayHours: <release_delay_hours>
  configs:
  - hostname: "<cp_node_name_1>"
    datacenter: "<master_01_datacenter>"
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<master_01_nic1_ip>/<nic1_prefix>"
        gateway: "<nic1_gateway>"
        dns:
        - "<nic1_dns_1>"
    persistentDisks:
    - name: var-cpaas
      sizeGiB: <cp_var_cpaas_size_gib>
      mountPath: /var/cpaas
      fsFormat: ext4
    ephemeralDisks:
    - name: var-lib-kubelet
      sizeGiB: <cp_var_lib_kubelet_size_gib>
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <cp_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
    - name: var-lib-etcd
      sizeGiB: <cp_var_lib_etcd_size_gib>
      mountPath: /var/lib/etcd
      fsFormat: ext4
  - hostname: "<cp_node_name_2>"
    datacenter: "<master_02_datacenter>"
```

```
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<master_02_nic1_ip>/<nic1_prefix>"
        gateway: "<nic1_gateway>"
        dns:
        - "<nic1_dns_1>"
    persistentDisks:
    - name: var-cpaas
      sizeGiB: <cp_var_cpaas_size_gib>
      mountPath: /var/cpaas
      fsFormat: ext4
    ephemeralDisks:
    - name: var-lib-kubelet
      sizeGiB: <cp_var_lib_kubelet_size_gib>
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <cp_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
    - name: var-lib-etcd
      sizeGiB: <cp_var_lib_etcd_size_gib>
      mountPath: /var/lib/etcd
      fsFormat: ext4
  - hostname: "<cp_node_name_3>"
    datacenter: "<master_03_datacenter>"
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<master_03_nic1_ip>/<nic1_prefix>"
        gateway: "<nic1_gateway>"
        dns:
        - "<nic1_dns_1>"
    persistentDisks:
    - name: var-cpaas
      sizeGiB: <cp_var_cpaas_size_gib>
      mountPath: /var/cpaas
      fsFormat: ext4
    ephemeralDisks:
    - name: var-lib-kubelet
      sizeGiB: <cp_var_lib_kubelet_size_gib>
```

```
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <cp_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
    - name: var-lib-etcd
      sizeGiB: <cp_var_lib_etcd_size_gib>
      mountPath: /var/lib/etcd
      fsFormat: ext4
```

创建 worker machine config pool。

17-vspheremachineconfigpool-worker.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineConfigPool
metadata:
  name: <cluster_name>-worker-pool
  namespace: <namespace>
spec:
  clusterRef:
    apiVersion: cluster.x-k8s.io/v1beta1
    kind: Cluster
    name: <cluster_name>
  datacenter: "<default_datacenter>"
  releaseDelayHours: <release_delay_hours>
  configs:
  - hostname: "<worker_node_name_1>"
    datacenter: "<worker_01_datacenter>"
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<worker_01_nic1_ip>/<nic1_prefix>"
        gateway: "<nic1_gateway>"
        dns:
        - "<nic1_dns_1>"
    persistentDisks:
    - name: var-cpaas
      sizeGiB: <worker_var_cpaas_size_gib>
      mountPath: /var/cpaas
      fsFormat: ext4
    ephemeralDisks:
    - name: var-lib-kubelet
      sizeGiB: <worker_var_lib_kubelet_size_gib>
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <worker_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
```

应用两个清单：

```
kubectl apply -f 16-vspheremachineconfigpool-control-plane.yaml
kubectl apply -f 17-vspheremachineconfigpool-worker.yaml
```

创建 Machines 之前，请验证池绑定和槽位状态：

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

列表输出包含 `TOTAL`、`AVAILABLE` 和 `ALLOCATED` 列，分别取自 `status.total`、`status.available` 和 `status.allocated`。创建 Machines 之前，预期值为 `ClusterRefReady=True`、`VCenterAvailable=True`、`MembersValid=True`、`MembersUnique=True` 和 `Ready=True`。

`SlotAvailable` 有意不包含在 `Ready` 摘要中。即使池中的所有槽位都已分配，该池仍会报告 `Ready=True` 和 `SlotAvailable=False`；在 `maxSurge: 0` 部署过程中，这种组合是正常的，并不矛盾。

该池也是持久磁盘替换状态的权威来源。VM 被替换时，其槽位会从 `InUse` 变为 `Released`；引用同一现有池的替换 Machine 可以立即重用该槽位及其 VMDK。`releaseDelayHours` 控制槽位保持未使用时的回收行为；它不是重用延迟。

请勿将创建新的池名称作为 Machine 替换时的常规重试方式。不同的 `VSphereMachineConfigPool.metadata.name` 会创建相互独立的槽位和 VMDK，因此反复拆除和重新部署可能会暂时占用多套完整磁盘。在容量受限的数据存储上再次部署之前，请等待先前的池及其 VMDK 完成由 provider 管理的删除。有关拆除检查，请参阅持久磁盘生命周期。实时状态信号为 `status.persistentDiskStatuses[].phase`，其值包括 `Creating`、`Attached`、`Available`、`Reclaiming`、`Reclaimed` 和 `Error`。`status.configStatuses[].reclaimStatus` 子树以及 `spec.configs[].persistentDisks[].volumePath` 和 `.diskUUID` 字段已弃用、被冻结，并计划移除。

6 应用可选的故障域对象

基准单数据中心拓扑可跳过此步骤。

如果启用了故障域，请先完成多个数据中心和故障域，然后在创建控制平面之前应用故障域对象：

```
kubectl apply -f 18-failure-domains.yaml
kubectl get vspherefailuredomain,vspheredeploymentzone
```

确认集群引用的每个 `VSphereFailureDomain` 和 `VSphereDeploymentZone` 都存在。部署区域可用之前，请勿继续。以下内容应作为一组配置进行管理：

- `18-failure-domains.yaml` 中的 `VSphereFailureDomain` 和 `VSphereDeploymentZone` 对象
- `10-cluster.yaml` 中的 `VSphereCluster.spec.failureDomainSelector`
- CPI `15-vsphere-cpi-clusterresourceset.yaml` 中的 `[Labels]` 块

未启用故障域时，请勿将 `failureDomainSelector` 或 CPI `[Labels]` 块添加到基准清单中。

7 创建控制平面对象

创建 `VSphereMachineTemplate` 和 `KubeadmControlPlane` 对象。将以下完整模板中的占位符替换为检查清单文档中收集的值。

> **Kubernetes 1.35 kubelet 设置**
>
> 基准控制平面和 worker 清单省略了 `imagePullCredentialsVerificationPolicy`，适用于 Kubernetes 1.34 或更早版本。
>
> 对于 Kubernetes 1.35 或更高版本，在应用任一清单之前，请将以下字段添加到 `20-control-plane.yaml` 和 `30-workers-md-0.yaml` 中的 `KubeletConfiguration` JSON：
>
> ```
> "imagePullCredentialsVerificationPolicy": "NeverVerify",
> ```

`cloneMode` 和 `diskGiB` 在模板中都会保留，因为 CAPV 同时接受这两个字段。实际上，只有当实际克隆操作为 `fullClone` 时，`diskGiB` 才会影响系统磁盘。如果 `cloneMode` 为 `linkedClone` 且模板具有可用快照，CAPV 将完成链接克隆，系统磁盘大小保持与源模板相同。如果不存在可用快照，CAPV 将回退到 `fullClone`，并再次应用 `diskGiB`。

> **系统磁盘与持久化磁盘是分开的**
>
> `VSphereMachineTemplate.spec.template.spec.diskGiB` 仅设置 VM 系统磁盘的大小，而不是节点上所有磁盘的总容量。
>
> 数据磁盘需要分别在 `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` 和 `.ephemeralDisks[]` 下声明。不要将数据磁盘大小加到 `diskGiB` 中；否则，VM 可能会获得更大的系统磁盘以及单独的数据磁盘，导致实际容量翻倍。
>
> 对于 `fullClone`，`diskGiB` 必须大于或等于 OS 镜像模板中的系统磁盘大小。对于 `linkedClone`，系统磁盘保持模板大小，并忽略 `diskGiB`。

20-control-plane.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineTemplate
metadata:
  name: <cluster_name>-control-plane
  namespace: <namespace>
spec:
  template:
    spec:
      server: "<vsphere_server>"
      template: "<template_name>"
      cloneMode: <clone_mode>
      folder: "<vm_folder>"
      datastore: "<cp_system_datastore>"
      diskGiB: <cp_system_disk_gib>
      memoryMiB: <cp_memory_mib>
      numCPUs: <cp_num_cpus>
      os: Linux
      powerOffMode: <power_off_mode>
      network:
        devices:
        - networkName: "<nic1_network_name>"
      machineConfigPoolRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineConfigPool
        name: <cluster_name>-cp-pool
        namespace: <namespace>
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster_name>-kcp
  namespace: <namespace>
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  version: "<k8s_version>"
  replicas: <cp_replicas>
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    metadata:
```

```
      labels:
        node-role.kubernetes.io/control-plane: ""
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: VSphereMachineTemplate
      name: <cluster_name>-control-plane
  kubeadmConfigSpec:
    users:
    - name: boot
      sudo: ALL=(ALL) NOPASSWD:ALL
      sshAuthorizedKeys:
      - "<ssh_public_key>"
    files:
    - path: /etc/resolv.conf
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        nameserver <nic1_dns_1>
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: AdmissionConfiguration
        plugins:
        - name: PodSecurity
          configuration:
            apiVersion: pod-security.admission.config.k8s.io/v1
            kind: PodSecurityConfiguration
            defaults:
              enforce: "privileged"
              enforce-version: "latest"
              audit: "baseline"
              audit-version: "latest"
              warn: "baseline"
              warn-version: "latest"
            exemptions:
              usernames: []
              runtimeClasses: []
              namespaces:
              - kube-system
              - <namespace>
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
```

```
son
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "protectKernelDefaults": true,
          "streamingConnectionIdleTimeout": "5m",
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key"
        }
    # Generate the encryption key with: head -c 32 /dev/urandom | bas
e64
    - 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>
    - path: /etc/kubernetes/audit/policy.yaml
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        omitStages:
        - "RequestReceived"
        rules:
        - level: None
          users:
          - system:kube-controller-manager
          - system:kube-scheduler
          - system:serviceaccount:kube-system:endpoint-controller
```

```
      verbs: ["get", "update"]
      namespaces: ["kube-system"]
      resources:
      - group: ""
        resources: ["endpoints"]
    - level: None
      nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
    - level: None
      resources:
      - group: ""
        resources: ["events"]
    - level: None
      resources:
      - group: "devops.alauda.io"
    - level: None
      verbs: ["get", "list", "watch"]
    - level: None
      resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
    - level: None
      resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessrev
iews"]
      - group: "authentication.k8s.io"
        resources: ["tokenreviews"]
    - level: None
      resources:
      - group: "app.alauda.io"
        resources: ["imagewhitelists"]
      - group: "k8s.io"
        resources: ["namespaceoverviews"]
    - level: Metadata
      resources:
      - group: ""
        resources: ["secrets", "configmaps"]
    - level: Metadata
      resources:
      - group: "operator.connectors.alauda.io"
        resources: ["installmanifests"]
```

```
    - group: "operators.katanomi.dev"
      resources: ["katanomis"]
  - level: RequestResponse
    resources:
    - group: ""
    - group: "aiops.alauda.io"
    - group: "apps"
    - group: "app.k8s.io"
    - group: "authentication.istio.io"
    - group: "auth.alauda.io"
    - group: "autoscaling"
    - group: "asm.alauda.io"
    - group: "clusterregistry.k8s.io"
    - group: "crd.alauda.io"
    - group: "infrastructure.alauda.io"
    - group: "monitoring.coreos.com"
    - group: "operators.coreos.com"
    - group: "networking.istio.io"
    - group: "extensions.istio.io"
    - group: "install.istio.io"
    - group: "security.istio.io"
    - group: "telemetry.istio.io"
    - group: "opentelemetry.io"
    - group: "networking.k8s.io"
    - group: "portal.alauda.io"
    - group: "rbac.authorization.k8s.io"
    - group: "storage.k8s.io"
    - group: "tke.cloud.tencent.com"
    - group: "devopsx.alauda.io"
    - group: "core.katanomi.dev"
    - group: "deliveries.katanomi.dev"
    - group: "integrations.katanomi.dev"
    - group: "artifacts.katanomi.dev"
    - group: "builds.katanomi.dev"
    - group: "versioning.katanomi.dev"
    - group: "sources.katanomi.dev"
    - group: "tekton.dev"
    - group: "operator.tekton.dev"
    - group: "eventing.knative.dev"
    - group: "flows.knative.dev"
    - group: "messaging.knative.dev"
    - group: "operator.knative.dev"
    - group: "sources.knative.dev"
    - group: "operator.devops.alauda.io"
```

```
          - group: "flagger.app"
          - group: "jaegertracing.io"
          - group: "velero.io"
            resources: ["deletebackuprequests"]
          - group: "connectors.alauda.io"
          - group: "operator.connectors.alauda.io"
            resources: ["connectorscores", "connectorsgits", "connect
orsocis"]
        - level: Metadata
    - path: /usr/local/bin/capv-load-local-images.sh
      owner: "root:root"
      permissions: "0755"
      content: |
        #!/bin/bash
        set -euo pipefail
        until mountpoint -q /var/lib/containerd; do
          echo "waiting for /var/lib/containerd mount"
          sleep 1
        done
        systemctl restart containerd
        until systemctl is-active --quiet containerd; do
          echo "waiting for containerd"
          sleep 1
        done
        if [ ! -d "/root/images" ]; then
          echo "ERROR: /root/images directory not found" >&2
          exit 1
        fi
        image_count=0
        for image_file in /root/images/*.tar; do
          if [ -f "$image_file" ]; then
            echo "importing image: $image_file"
            ctr -n k8s.io images import "$image_file"
            image_count=$((image_count + 1))
          fi
        done
        if [ "$image_count" -eq 0 ]; then
          echo "ERROR: no tar files found in /root/images" >&2
          exit 1
        fi
        echo "imported $image_count images"
    preKubeadmCommands:
    - hostnamectl set-hostname "{{ ds.meta_data.hostname }}"
    - echo "::1         ipv6-localhost ipv6-loopback localhost6 local
```

```
      echo "::1         ipv6-localhost ipv6-loopback localhost6 local
host6.localdomain6" >/etc/hosts
    - echo "127.0.0.1   {{ ds.meta_data.hostname }} {{ local_hostname
}} localhost localhost.localdomain localhost4 localhost4.localdomain
4" >>/etc/hosts
    - while ! ip route | grep -q "default via"; do sleep 1; done; ech
o "NetworkManager started"
    - /usr/local/bin/capv-load-local-images.sh
    postKubeadmCommands:
    - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: <image_registry>/tkestack
      dns:
        imageTag: <dns_image_tag>
      etcd:
        local:
          imageTag: <etcd_image_tag>
      apiServer:
        extraArgs:
          admission-control-config-file: /etc/kubernetes/admission/ps
a-config.yaml
          audit-log-format: json
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          encryption-provider-config: /etc/kubernetes/encryption-prov
ider.conf
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
          profiling: "false"
          tls-cipher-suites: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_P
OLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHAC
HA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
          tls-min-version: VersionTLS12
        extraVolumes:
        - hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          name: vol-dir-0
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
```

```
          bind-address: ::
          cloud-provider: external
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-pl
ugins/volume/exec/"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
    initConfiguration:
      nodeRegistration:
        criSocket: /var/run/containerd/containerd.sock
        ignorePreflightErrors:
        - ImagePull
        kubeletExtraArgs:
          cloud-provider: external
          node-labels: kube-ovn/role=master
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
        name: '{{ local_hostname }}'
      patches:
        directory: /etc/kubernetes/patches
    joinConfiguration:
      nodeRegistration:
        criSocket: /var/run/containerd/containerd.sock
        ignorePreflightErrors:
        - ImagePull
        kubeletExtraArgs:
          cloud-provider: external
          node-labels: kube-ovn/role=master
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
        name: '{{ local_hostname }}'
      patches:
        directory: /etc/kubernetes/patches
```

`rolloutStrategy.rollingUpdate.maxSurge: 0` 以及 `3` 或更高的 `replicas` 值并非风格选择。基础设施模板设置了 `machineConfigPoolRef` 的 `KubeadmControlPlane` 需要同时配置这两项。请参阅 Provider Requirements。

应用清单：

```
kubectl apply -f 20-control-plane.yaml
```

8 创建 worker 对象

创建 worker machine 模板、bootstrap 模板和 `MachineDeployment`。

基线 worker kubelet patch 会省略仅适用于 Kubernetes 1.35 的字段。如果所选版本为 Kubernetes 1.35 或更高版本，请按照控制平面步骤中的说明，在此处以及 `20-control-plane.yaml` 中添加 `imagePullCredentialsVerificationPolicy: NeverVerify`。

30-workers-md-0.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineTemplate
metadata:
  name: <cluster_name>-worker
  namespace: <namespace>
spec:
  template:
    spec:
      server: "<vsphere_server>"
      template: "<template_name>"
      cloneMode: <clone_mode>
      folder: "<vm_folder>"
      datastore: "<worker_system_datastore>"
      diskGiB: <worker_system_disk_gib>
      memoryMiB: <worker_memory_mib>
      numCPUs: <worker_num_cpus>
      os: Linux
      powerOffMode: <power_off_mode>
      network:
        devices:
        - networkName: "<nic1_network_name>"
      machineConfigPoolRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineConfigPool
        name: <cluster_name>-worker-pool
        namespace: <namespace>
---
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster_name>-worker-bootstrap
  namespace: <namespace>
spec:
  template:
    spec:
      files:
      - path: /etc/resolv.conf
        owner: "root:root"
        append: false
        permissions: "0644"
        content: |
          nameserver <nic1_dns_1>
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategi
```

```
c.json
        owner: "root:root"
        permissions: "0644"
        content: |
          {
            "apiVersion": "kubelet.config.k8s.io/v1beta1",
            "kind": "KubeletConfiguration",
            "protectKernelDefaults": true,
            "staticPodPath": null,
            "streamingConnectionIdleTimeout": "5m",
            "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
            "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key"
          }
      - path: /usr/local/bin/capv-load-local-images.sh
        owner: "root:root"
        permissions: "0755"
        content: |
          #!/bin/bash
          set -euo pipefail
          until mountpoint -q /var/lib/containerd; do
            echo "waiting for /var/lib/containerd mount"
            sleep 1
          done
          systemctl restart containerd
          until systemctl is-active --quiet containerd; do
            echo "waiting for containerd"
            sleep 1
          done
          if [ ! -d "/root/images" ]; then
            echo "ERROR: /root/images directory not found" >&2
            exit 1
          fi
          image_count=0
          for image_file in /root/images/*.tar; do
            if [ -f "$image_file" ]; then
              echo "importing image: $image_file"
              ctr -n k8s.io images import "$image_file"
              image_count=$((image_count + 1))
            fi
          done
          if [ "$image_count" -eq 0 ]; then
            echo "ERROR: no tar files found in /root/images" >&2
            exit 1
          fi
```

```
          echo "imported $image_count images"
      joinConfiguration:
        nodeRegistration:
          criSocket: /var/run/containerd/containerd.sock
          ignorePreflightErrors:
          - ImagePull
          kubeletExtraArgs:
            cloud-provider: external
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugi
ns/volume/exec/"
          name: '{{ local_hostname }}'
        patches:
          directory: /etc/kubernetes/patches
      preKubeadmCommands:
      - hostnamectl set-hostname "{{ ds.meta_data.hostname }}"
      - echo "::1         ipv6-localhost ipv6-loopback localhost6 loc
alhost6.localdomain6" >/etc/hosts
      - echo "127.0.0.1   {{ ds.meta_data.hostname }} {{ local_hostna
me }} localhost localhost.localdomain localhost4 localhost4.localdoma
in4" >>/etc/hosts
      - while ! ip route | grep -q "default via"; do sleep 1; done; e
cho "NetworkManager started"
      - /usr/local/bin/capv-load-local-images.sh
      postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
      users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        sshAuthorizedKeys:
        - "<ssh_public_key>"
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster_name>-md-0
  namespace: <namespace>
spec:
  clusterName: <cluster_name>
  replicas: <worker_replicas>
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
```

```
  selector:
    matchLabels:
      nodepool: md-0
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster_name>
        nodepool: md-0
    spec:
      clusterName: <cluster_name>
      version: "<k8s_version>"
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster_name>-worker-bootstrap
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineTemplate
        name: <cluster_name>-worker
```

对于设置了 `machineConfigPoolRef` 的 `MachineDeployment`，同样需要 `strategy.rollingUpdate.maxSurge: 0`。在默认 `1` 的 `maxSurge` 下，滚动更新需要一个额外槽位，而完整的池没有该槽位，因此会停滞。示例使用 `maxUnavailable: 1`。

应用清单：

```
kubectl apply -f 30-workers-md-0.yaml
```

在基线工作流中，请注意以下 worker 特定规则：

- 主 worker 清单默认不会设置 `failureDomain`，因为基线工作流假定使用单个数据中心。如果需要让 worker `MachineDeployment` 运行在特定的 `VSphereDeploymentZone` 中，请按照多个数据中心和故障域中的说明添加 `failureDomain`。
- 某些环境会向 `KubeadmConfigTemplate` 添加额外的运行时镜像替换命令或服务重启命令。这些命令有意未包含在基线示例中。只有当环境中的平台要求明确需要这些命令时，才添加它们。

9 等待集群就绪

应用所有清单后，集群创建过程将异步执行。使用以下命令监控进度：

```
kubectl -n <namespace> get cluster,kubeadmcontrolplane,machinedeployment,machine -w
```

等待 `KubeadmControlPlane` 报告预期数量的就绪副本，并且所有 `Machine` 对象都进入 `Running` 阶段，然后再继续验证。

验证

使用以下命令验证集群创建工作流。

1. 检查 `global` 集群中的 CPI 交付资源：

```
kubectl -n <namespace> get clusterresourceset
kubectl -n <namespace> get clusterresourcesetbinding
```

2. 导出业务集群 kubeconfig：

```
kubectl -n <namespace> get secret <cluster_name>-kubeconfig -o jsonpath='{.data.value}' | base64 -d > /tmp/<cluster_name>.kubeconfig
```

3. 检查 vSphere CPI daemonset 是否已在业务集群中创建：

```
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system get daemonset
```

4. 检查 `global` 集群对象：

```
kubectl -n <namespace> get cluster,vspherecluster,kubeadmcontrolplane,machinedeployment,machine,vspheremachine,vspherevm
```

5. 检查业务集群节点：

```
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
```

6. 检查两个列表的池槽位计数器和每个磁盘的状态：

```
kubectl -n <namespace> get vspheremachineconfigpool
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp-p
ool \
  -o jsonpath='{range .status.persistentDiskStatuses[*]}{.hostname}{"
"}{.name}{" "}{.phase}{" unit="}{.unitNumber}{"\n"}{end}'
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp-p
ool \
  -o jsonpath='{range .status.ephemeralDiskStatuses[*]}{.hostname}{" "}
{.name}{" unit="}{.unitNumber}{"\n"}{end}'
```

确认以下结果：

- `vsphere-cloud-controller-manager` 出现在业务集群中。
- 控制平面和 worker 节点均已创建。
- 节点最终变为 `Ready`。
- 每个池均报告 `Ready=True`，且其 `ALLOCATED` 计数等于从该池分配的运行中 Machine 数量。
- 已分配槽位上的每个持久磁盘均报告 `phase: Attached`。
- 已分配槽位上的每个临时磁盘都显示在 `status.ephemeralDiskStatuses[]` 中，并分配有单元号。临时磁盘没有阶段；它们会随 VM 一起创建。

故障排除

当工作流失败时，首先使用以下命令：

```
kubectl -n <namespace> describe cluster <cluster_name>
kubectl -n <namespace> describe vsphere cluster <cluster_name>
kubectl -n <namespace> describe kubeadmcontrolplane <cluster_name>-kcp
kubectl -n <namespace> describe machinedeployment <cluster_name>-md-0
kubectl -n <namespace> get cluster,vspherecluster,kubeadmcontrolplane,mac
hinedeployment,machine,vspheremachine,vspherevm
kubectl -n cpaas-system logs deploy/capi-controller-manager
```

优先执行以下检查：

- 如果 CPI 资源未交付，请验证 `ClusterResourceSet=true`、`ClusterResourceSet` 和 `ClusterResourceSetBinding`。
- 如果存在 `ClusterResourceSet`，但未创建 `ClusterResourceSetBinding`，请检查控制器是否对所引用的 `ConfigMap` 和 `Secret` 资源具有所需的删除权限。
- 如果未安装网络插件，请验证所需的集群注解是否存在，以及平台控制器是否已处理这些注解。
- 如果缺少或错误设置了 `cpaas.io/registry-address` 注解，请验证其是否设置为不带 scheme 或 `/v2/` 的永久 `<registry_address>`。然后验证为其提供支持的凭证：当集群继承 `global` 集群的 registry 时，验证 `public-registry-credential`；当集群使用专用 registry 时，验证由 `cpaas.io/registry-reference` 指定名称的 Secret。
- 如果 Machine 卡在 `Provisioning`，请检查 `VSphereMachine` 中针对 `MachineConfigPoolReady` 的条件——它会显示槽位分配是否因池绑定或数据中心不匹配而失败。然后读取池自身的条件：`MembersValid`、`MembersUnique`、`PersistentDisksReady`、`ClusterRefReady` 和 `VCenterAvailable`。
- 如果 Machine 始终无法完成 bootstrap，请检查 `VSphereVM` 和 `VSphereMachine` 上的 `BootstrapReady` 条件。原因 `BootstrapSecretGetFailed` 表示 bootstrap Secret 缺失或无法读取；`BootstrapSecretContentInvalid` 表示该 Secret 存在但无法使用。在检查 vCenter 之前，请检查该 Machine 的 `KubeadmConfig` 及其 bootstrap 数据 Secret。
- 如果节点在 vCenter 中被关闭电源后一直保持关闭，这是预期行为。`VSphereVM` 会在首次成功开机后锁存 `InitialPowerOnCompleted=True`，并且永远不会清除；从此以后，已关闭电源的 VM 会被视为有意执行的带外操作，不会再次自动开机。`PoweredOn=False` 会传播到 `VSphereVM.Ready`，然后传播到 `VSphereMachine.Ready`，因此报告未就绪是正确的。在 vCenter 中启动该 VM 即可恢复；无需执行 rollout。rollout 创建的替代 VM 未设置该锁存值，因此其首次启动不受影响。

- 如果 `VSphereMachine` 在没有明显原因的情况下清除 `Ready`，请注意：当后端 `VSphereVM`、其 `providerID` 或其网络信息不可用时，`Ready` 会被清除，并且现在可以观测到带外电源变更。在将其视为控制器故障之前，请读取 `VSphereVM` 条件。
- 如果完全没有任何 reconcile 且条件也没有变化，请检查管理集群是否为灾难恢复备用集群。请参阅备用管理集群。
- 如果 VM 正在等待 IP 分配，请验证 VMware Tools、静态 IP 设置和 `VSphereVM.status.addresses`。
- 如果业务集群的 `Node` 对象仍然没有 `spec.providerID`，请先验证 CPI 交付资源，然后检查是否存在重复的 vCenter guest hostname。当同一数据中心中的旧 VM 仍报告与新节点相同的 guest hostname 时，`cloud-provider-vsphere` 可能回退到节点名称查找，缓存旧 VM，并因 VM IP 与 kubelet 节点 IP 不匹配而拒绝新节点。请检查 leader `vsphere-cloud-controller-manager` 日志、节点 `SystemUUID`、实际 VM UUID，以及 vCenter guest hostname/IP 值。修复或移除重复 hostname 或旧 VM 冲突后，重启业务集群的 `vsphere-cloud-controller-manager` Pod，以清除错误的内存缓存：

```
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system dele
te pod \
  -l k8s-app=vsphere-cloud-controller-manager
```

- 如果 datastore 空间耗尽，请先列出所有当前存在和正在终止的 `VSphereMachineConfigPool` 对象，并将其名称与每个 `VSphereMachineTemplate.spec.template.spec.machineConfigPoolRef.name` 进行比较。不同的池名称拥有相互独立的槽位和 VMDK；因此，重复部署可能会累积多组磁盘。然后检查 `status.configStatuses[]` 中的 `state` 和 `lastReleasedTime`，以及 `status.persistentDiskStatuses[]` 中的 `phase`、`lastError`、`retryAfter` 和 `taskRef`，同时检查相关 Events 和 vCenter 附加状态。`insufficient disk available` 可确认 datastore 容量故障，但其本身不能证明回收失败。在复用或 provider-managed 回收等待期间，不要手动删除 VMDK 或移除池 finalizer。
- 不要从 `status.persistentDiskStatuses[]` 中删除 `phase: Reclaimed` 条目。这是一个有意保留的 tombstone：后端 VMDK 已经不存在，而该记录会阻止控制器根据冻结的 `spec` `volumePath` 重新为磁盘播种。删除它会产生播种、删除、重新播种的循环，使池 finalizer 永远无法清除。
- 如果模板系统磁盘大小与清单值不匹配，请先检查实际的克隆模式。当 VM 创建为 `linkedClone` 时，系统磁盘会保持模板的大小，并忽略 `diskGiB`。只有 `fullClone` 使用 `diskGiB`；在这种情况下，`diskGiB` 不得小于模板磁盘大小。

- 如果控制平面端点未启动，请验证 TCP `6443` 透传、所有控制平面后端、HTTPS `/healthz` 结果、DNS 和证书 SAN，以及从 `global` 集群和控制平面节点的可达性。
- 如果与 vCenter 的 TLS 连接失败，请验证 thumbprint、vCenter 地址，以及代理设置是否干扰连接。

查看控制器日志时，请遵循以下规则：

- `deploy/capi-controller-manager` 在 `global` 集群的 `cpaas-system` namespace 中运行。
- 不要使用业务集群 kubeconfig 检查 `capi-controller-manager` 日志。
- 如果平台控制器处理集群网络注解，也请检查平台 network-controller 日志和平台 cluster-lifecycle-controller 日志。

Admission 拒绝

`apply` 或 `edit` 可能会在创建任何对象之前被拒绝。将拒绝消息中的片段与下表进行匹配。

拒绝消息包含	原因和修复
`maxSurge must be 0 when the infrastructure template references a machineConfigPoolRef`	在 `KubeadmControlPlane` 上将 `maxSurge` 设置为整数 `0`。不设置该值意味着使用 Cluster API 默认值 `1`。
`must have at least 3 replicas`	将 `KubeadmControlPlane.spec.replicas` 设置为 `3` 或更大值。固定 IP rollout 使用 `maxSurge: 0`，而 Cluster API 仅允许控制平面拥有三个或更多副本时使用该值。
`unit number 7 is reserved for the SCSI controller`	选择 `0` 到 `15` 之间的其他 `unitNumber`。
`also used by pool`	hostname 或主 IP 与此 namespace 中绑定到同一 `Cluster` 的另一个池中的槽位冲突。

拒绝消息包含	原因和修复
`allocated slot`	槽位为 `InUse` 或 `Released`。消息会指出被冻结的内容——主 IP 或 IPv6，或持久磁盘的 `sizeGiB` 或 `mountPath`——或者报告该槽位或其某个持久磁盘无法移除。请参阅已分配槽位限制。
`cannot delete pool while`	池仍拥有 `InUse` 个槽位。请先删除占用这些槽位的 Machine。
`is already referenced by` 或 `is bound to`	两个控制器引用同一个池。为每个 `KubeadmControlPlane` 和 `MachineDeployment` 分配各自的池。
`cannot change clusterRef while consumerRef is set`	该池已绑定到控制器。请创建新池，而不是重新指定此池的目标。
`VSphereMachineTemplate spec.template.spec field is immutable`	请在新名称下创建新模板，而不是编辑现有模板。请参阅发布更新的 worker 节点配置。
`is immutable once controlPlaneLoadBalancer is set`	`spec.controlPlaneLoadBalancer` 只能写入一次。请参阅 `controlPlaneLoadBalancer` 字段。
`cannot be set after the cluster is created`	创建集群时未设置 `spec.controlPlaneLoadBalancer`，这已经表示使用外部 LoadBalancer。端点在创建时固定，之后无法再填充。

创建时拓扑变体

基线清单有意从一个数据中心和一个 NIC 开始。当初始拓扑需要 Self-built 控制平面 VIP、额外 NIC、多个数据中心、故障域或额外数据磁盘时，请在创建集群前使用以下变体。每次应用一个变体，并在组合多个变体前验证完整的清单集合。

使用 **Self-built VIP**

当集群使用 `VSphereCluster.spec.controlPlaneLoadBalancer.type: internal` 而不是外部 LoadBalancer 创建时，请使用此变体。先在控制平面端点模式中确定模式；集群创建后无法更改。

Self-built VIP 前提条件

在选择 `type: internal` 前完成以下所有操作。

- 平台版本为 ACP `v4.4` 或更高版本。无论 provider 版本如何，较早的平台版本都不支持 Self-built VIP。
- VIP 必须为 IPv4、当前未被占用，并且与控制平面节点 IP 位于同一 Layer-2 广播域中。不接受 IPv6 和双栈端点。
- VIP 不能是此集群 `VSphereMachineConfigPool` 对象中声明的槽位地址（`network.primary.ip` 或 `network.additional[].ip`），也不能位于任何外部 IPAM 池范围内。provider 会检查槽位地址，但无法读取外部 IPAM 池，因此不会检测到该冲突。
- `vrid` 在该 Layer-2 域中必须唯一。
- 网络路径必须允许控制平面节点之间的 VRRP（IP 协议 `112`、多播 `224.0.0.18`）和免费 ARP。确认 dvPortGroup 安全策略以及任何 NSX-T 分布式防火墙规则未阻止这些流量。
- VM 模板必须公开 Linux IPVS 子系统（`ip_vs` 模块系列可加载），并允许 Alive 设置 `net.ipv4.conf.all.arp_accept=1` 和 `net.ipv4.vs.conntrack=1`。
- `alive` 集群插件必须存在，并且可部署到 `global` 集群。

provider 本身不携带 Alive。它会在 `global` 集群上创建一个 `ModuleInfo`，平台再据此渲染业务 `cpaas-system/alive` `AppRelease`。请在 `global` 集群上确认该插件：

```
kubectl get clustermodule global -o jsonpath='{.spec.version}{"\n"}'
kubectl get moduleplugin alive -o jsonpath='{.status.targetClusterVersion
s}{"\n"}'
kubectl get moduleplugin alive -o jsonpath='{.status.latestVersion}
{"\n"}'
kubectl get moduleconfig alive-<alive_version> -o jsonpath='{.status.read
yForDeploy}{"\n"}'
```

provider 按以下顺序解析 `<alive_version>`：如果已设置，则先使用 provider deployment 上的 `--plugin-alive-version` 覆盖值，然后使用 `ModulePlugin/alive` 条目 `status.targetClusterVersions[<ClusterModule spec.version>].version`，最后才使用

`status.latestVersion`。先读取与第一条命令打印的版本匹配的 `targetClusterVersions` 条目；仅当 map 中没有该版本的条目时，才使用 `latestVersion`。

业务集群自身的 `ClusterModule` 在集群创建前不存在，因此在规划时读取 `global` 集群的版本作为规划值。这只是最接近的可用代理值，并非保证：集群创建后，使用 `kubectl get clustermodule <cluster_name>` 读取其自身版本，并针对该版本解析到的 `alive` 版本重复执行 `readyForDeploy` 检查。请在控制平面完成引导前执行此操作，因为之后无法更改 `type: internal`。创建 `global` 集群本身时，请改为从安装包确认组件版本和 `alive` 插件是否存在。

最后一条命令必须针对解析选定的版本打印 `true`。如果缺少 `ModulePlugin/alive`，或者缺少匹配的 `ModuleConfig`，或其报告 `false`，则无法在此环境中安装 Self-built VIP。请改用外部 LoadBalancer 创建集群，并保持 `spec.controlPlaneLoadBalancer` 不存在。请联系 Alauda 技术支持获取与平台版本匹配的插件包。

`controlPlaneLoadBalancer` 字段

参数	类型	描述
`.spec.controlPlaneLoadBalancer.type`	string（`internal` / `external`）	`internal` 会安装 Alive 并管理 VIP。`external` 会跳过 Alive，并要求预先配置 Layer 4 TCP LoadBalancer。值均为小写；其他 provider 使用不同的拼写方式。
`.spec.controlPlaneLoadBalancer.host`	string	控制平面端点地址。对于 `internal`，即 Self-built VIP，该地址必须为 IPv4。对于 `external`，该地址为负载均衡器前端地址。
`.spec.controlPlaneLoadBalancer.port`	int (1–65535)	控制平面端口。通常为 `6443`。
`.spec.controlPlaneLoadBalancer.vrid`	int	Keepalived `virtual_router_id`，

参数	类型	描述
		范围为 `1` – `255`，且在控制平面 Layer-2 域中必须唯一。
`.spec.controlPlaneLoadBalancer.interface`	string（最多 15 个字符）	承载 VIP 的主机接口。留空可让 provider 匹配承载节点主 IP 的接口。

controlPlaneLoadBalancer 只能写入一次

provider 仅在创建集群时接受 `spec.controlPlaneLoadBalancer`。之后，`type`、`host`、`port`、`vrid` 和 `interface` 都会被冻结，该字段无法清除，创建时未设置该字段的集群也无法再添加。修正错误需要重建集群。

这会产生两个后果。在多 NIC 节点上，应在创建时显式设置 `interface`——错误的自动检测接口之后无法通过修补进行更改。此外，`host` 和 `port` 必须描述与 `spec.controlPlaneEndpoint` 相同的地址；不接受不匹配的配置。

`vrid` 范围由 provider 的 admission webhook 强制执行，而不是由 CRD schema 强制执行。客户端 dry run 不会联系任何服务器，因此既不会捕获此规则，也不会捕获 schema 规则；请使用 `kubectl apply --dry-run=server` 在不创建对象的情况下查看拒绝信息。

在 `10-cluster.yaml` 中启用 Self-built VIP

对于 `type: internal`，将以下代码块添加到创建 `Cluster` 和 `VSphereCluster` 对象中的 `VSphereCluster` 对象。基线清单中的其他内容保持不变。

```
spec:
  controlPlaneLoadBalancer:
    type: internal
    host: "<vip>"           # must equal spec.controlPlaneEndpoint.host
    port: <api_server_port> # must equal spec.controlPlaneEndpoint.port
    vrid: <vrid>
    interface: "<vip_interface>" # set explicitly on multi-NIC nodes
```

provider 还会为你协调业务集群的 `kube-system/kube-proxy` 配置，设置 `ipvs.strictARP: true` 并将 `<vip>/32` 添加到 `ipvs.excludeCIDRs`，然后滚动更新 kube-proxy DaemonSet。不要手动还原这些值；provider 会重新应用这些值，并再次滚动更新 DaemonSet。

验证 **Self-built VIP**

```
kubectl -n <namespace> get vsphere cluster <cluster_name> \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status} ({.reason})
{"\n"}{end}'
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system get po
ds -l app=alive -o wide
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n cpaas-system get a
pprelease alive
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get --raw='/version'
```

预期会看到 `SelfBuiltLoadBalancerReady=True`、每个控制平面节点上一个 Alive pod，以及通过 VIP 成功返回 API 响应，且无需禁用 TLS 验证。

Alive pod 运行在 `kube-system` 中，而不是 `cpaas-system` 中；`cpaas-system` 保存平台根据 provider 的 `ModuleInfo` 渲染的 `alive` `AppRelease`。`SelfBuiltLoadBalancerReady` 覆盖整个链路、`AppRelease` 和 pod，因此当其处于 `False` 时，请检查两个 namespace。

> **INFO**
>
> `SelfBuiltLoadBalancerReady` 仅报告 VIP 链路，不会阻止 `VSphereCluster.status.ready`，因此基础设施集群可能报告已就绪，而 VIP 仍在协调中。条件原因包括 `SelfBuiltLoadBalancerReady`、`SelfBuiltLoadBalancerReconciling`、`SelfBuiltLoadBalancerNotReady` 和 `InvalidSelfBuiltLoadBalancerConfiguration`。
>
> 在 VIP 尚未就绪时，provider 还会延迟[provider 管理的清单字段]中所述的业务集群 CoreDNS 和 kube-proxy 仓库协调。

添加第二个 **NIC**

当节点需要额外的管理、存储或服务网络时，请扩展以下资源中的清单：

- `16-vspheremachineconfigpool-control-plane.yaml`
- `17-vspheremachineconfigpool-worker.yaml`
- `20-control-plane.yaml`
- `30-workers-md-0.yaml`
- 启用故障域时使用 `18-failure-domains.yaml`

每个节点槽位都在 `network.primary` 和 `network.additional` 下声明其 NIC 布局。主 NIC 用于派生 kubelet `node-ip`，并仍作为节点的主要身份；额外 NIC 按列出的顺序合并到主 NIC 之后。

将第二个 NIC 添加到 machine config pool 中的每个控制平面节点槽位：

```
network:
  primary:
    networkName: "<nic1_network_name>"
    deviceName: "<nic1_device_name>"
    ip: "<master_01_nic1_ip>/<nic1_prefix>"
    gateway: "<nic1_gateway>"
    dns:
    - "<nic1_dns_1>"
  additional:
  - networkName: "<nic2_network_name>"
    deviceName: "<nic2_device_name>"
    ip: "<master_01_nic2_ip>/<nic2_prefix>"
    gateway: "<nic2_gateway>"
    dns:
    - "<nic2_dns_1>"
```

对 worker 节点槽位应用相同模式：

```
network:
  primary:
    networkName: "<nic1_network_name>"
    deviceName: "<nic1_device_name>"
    ip: "<worker_01_nic1_ip>/<nic1_prefix>"
    gateway: "<nic1_gateway>"
    dns:
    - "<nic1_dns_1>"
  additional:
  - networkName: "<nic2_network_name>"
    deviceName: "<nic2_device_name>"
    ip: "<worker_01_nic2_ip>/<nic2_prefix>"
    gateway: "<nic2_gateway>"
    dns:
    - "<nic2_dns_1>"
```

将第二个 NIC 添加到 machine template：

```
network:
  devices:
  - networkName: "<nic1_network_name>"
  - networkName: "<nic2_network_name>"
```

如果添加第二个 NIC 后节点使用的 DNS 服务器发生变化，请更新 `20-control-plane.yaml` 和 `30-workers-md-0.yaml` 中的 `/etc/resolv.conf` 文件条目。machine config pool 网络块中的 `dns` 值不会替代显式的 `/etc/resolv.conf` 引导文件条目。

如果启用了故障域，请更新 `VSphereFailureDomain.spec.topology.networks` 中的网络列表：

```
topology:
  networks:
  - <nic1_network_name>
  - <nic2_network_name>
```

定义第二个 NIC 的值时，请在基础设施检查清单和清单中准备以下占位符：

- `<master_01_nic2_ip>`

- `<master_02_nic2_ip>`
- `<master_03_nic2_ip>`
- `<worker_01_nic2_ip>`
- 同时扩展 worker 池时使用 `<worker_02_nic2_ip>`

对于运行中的集群，在替换节点时同时更改全部三个网络定义。请参阅运行时拓扑更改。

多个数据中心和故障域

当需要跨不同 vCenter 数据中心或计算集群放置节点时，请使用多个数据中心和故障域。

以下原则适用：

- 一个集群可以定义多个 `VSphereFailureDomain` 对象。
- 每个 `VSphereDeploymentZone` 引用一个 `VSphereFailureDomain`。
- 控制平面使用 `VSphereCluster.spec.failureDomainSelector`。
- 当 worker 必须面向特定部署区域时，worker `MachineDeployment` 使用 `spec.template.spec.failureDomain`。

为第一个数据中心准备以下占位符：

- `<compute_cluster_1>`
- `<default_datastore_1>`
- `<resource_pool_path_1>`
- `<fd_name_1>`
- `<dz_name_1>`

为第二个数据中心准备以下占位符：

- `<dc_name_2>`
- `<fd_name_2>`
- `<dz_name_2>`
- `<compute_cluster_2>`
- `<default_datastore_2>`

- `<resource_pool_path_2>`

如果添加第三个数据中心，请继续使用相同的占位符模式：

- `<dc_name_3>`
- `<fd_name_3>`
- `<dz_name_3>`
- `<compute_cluster_3>`
- `<default_datastore_3>`
- `<resource_pool_path_3>`

在 `18-failure-domains.yaml` 中创建故障域对象。启用故障域时，第一个数据中心还需要 `VSphereFailureDomain` 和 `VSphereDeploymentZone`：

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereFailureDomain
metadata:
  name: <fd_name_1>
spec:
  region:
    name: region-a
    type: Datacenter
    tagCategory: k8s-region
    autoConfigure: true
  zone:
    name: zone-1
    type: ComputeCluster
    tagCategory: k8s-zone
    autoConfigure: true
  topology:
    datacenter: <default_datacenter>
    computeCluster: <compute_cluster_1>
    datastore: <default_datastore_1>
    networks:
    - <nic1_network_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereDeploymentZone
metadata:
  name: <dz_name_1>
spec:
  server: <vsphere_server>
  failureDomain: <fd_name_1>
  controlPlane: true
  placementConstraint:
    resourcePool: <resource_pool_path_1>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereFailureDomain
metadata:
  name: <fd_name_2>
spec:
  region:
    name: region-a
    type: Datacenter
    tagCategory: k8s-region
    autoConfigure: true
```

```
  zone:
    name: zone-2
    type: ComputeCluster
    tagCategory: k8s-zone
    autoConfigure: true
  topology:
    datacenter: <dc_name_2>
    computeCluster: <compute_cluster_2>
    datastore: <default_datastore_2>
    networks:
    - <nic1_network_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereDeploymentZone
metadata:
  name: <dz_name_2>
spec:
  server: <vsphere_server>
  failureDomain: <fd_name_2>
  controlPlane: true
  placementConstraint:
    resourcePool: <resource_pool_path_2>
```

将 `failureDomainSelector` 添加到 `10-cluster.yaml` 中的 `VSphereCluster` spec，以启用跨可用故障域的控制平面选择：

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereCluster
metadata:
  name: <cluster_name>
  namespace: <namespace>
spec:
  controlPlaneEndpoint:
    host: "<vip>"
    port: <api_server_port>
  identityRef:
    kind: Secret
    name: <credentials_secret_name>
  server: "<vsphere_server>"
  thumbprint: "<thumbprint>"
  failureDomainSelector: {}
```

空选择器 `{}` 会匹配所有具有 `controlPlane: true` 的 `VSphereDeploymentZone`。使用匹配标签将控制平面限制到部分区域。

同时将 `[Labels]` 块添加到 CPI `ConfigMap`，以便 vSphere CPI 在业务节点上发布匹配的区域和地域标签。键必须与 `VSphereFailureDomain.spec.zone.tagCategory` 和 `VSphereFailureDomain.spec.region.tagCategory` 中使用的 `tagCategory` 值匹配。更新 `15-vsphere-cpi-clusterresourceset.yaml` 中的 `vsphere.conf` 数据：

```
    vsphere.conf: |
      [Global]
      secret-name = "vsphere-cloud-secret"
      secret-namespace = "kube-system"
      service-account = "cloud-controller-manager"
      port = "443"
      insecure-flag = "1"
      datacenters = "<cpi_datacenters>"

      [Labels]
      zone = "k8s-zone"
      region = "k8s-region"

      [VirtualCenter "<vsphere_server>"]
```

必须同时启用 `failureDomainSelector` 和 CPI 的 `[Labels]` 块。只添加其中一个会使集群处于不一致状态：节点会获得无法解析的区域或地域标签，或者控制平面无法选择部署目标。

当 worker `MachineDeployment` 必须固定到一个部署目标时，设置 worker 部署区域。将 `failureDomain` 添加到 `30-workers-md-0.yaml` 中的 `spec.template.spec`：

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster_name>-md-0
  namespace: <namespace>
spec:
  clusterName: <cluster_name>
  replicas: <worker_replicas>
  template:
    spec:
      clusterName: <cluster_name>
      failureDomain: <worker_failure_domain>
      version: "<k8s_version>"
      # ... rest of spec
```

为 `<worker_failure_domain>` 使用 `VSphereDeploymentZone` 名称，而不是 `VSphereFailureDomain` 名称。

启用多个数据中心前，确认满足以下所有前提条件：

1. 模板已同步到每个目标数据中心。
2. 每个目标数据中心都能解析网络名称。
3. 每个目标数据中心都能解析数据存储名称。
4. vSphere CPI 数据中心列表涵盖每个目标数据中心。

添加数据磁盘

基线部署包含以下必需数据磁盘：

- 控制平面节点（每个节点 4 个磁盘）：`persistentDisks[]` 中的 `var-cpaas`，以及 `ephemeralDisks[]` 中的 `var-lib-kubelet`、`var-lib-containerd` 和 `var-lib-etcd`。不要删除其中任何磁盘。
- **Worker** 节点（每个节点 3 个磁盘）：`persistentDisks[]` 中的 `var-cpaas`，以及 `ephemeralDisks[]` 中的 `var-lib-kubelet` 和 `var-lib-containerd`。不要删除其中任何磁盘。

`VSphereMachineTemplate.spec.template.spec.diskGiB` 是系统磁盘大小，而不是 VM 的总磁盘容量。将数据磁盘保留在 `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` 和 `.ephemeralDisks[]` 下。除非有意增大系统磁盘，否则不要将数据磁盘大小添加到 `diskGiB`。

如果节点需要超出必需集合的额外数据磁盘，请将更多条目追加到对应 `VSphereMachineConfigPool` 节点槽位中的匹配列表：数据必须在 VM 替换后保留时使用 `persistentDisks`，可重建时使用 `ephemeralDisks`。以下可选字段与此特别相关：

- **`mountPath`**：如果设置，磁盘会格式化并挂载到指定路径。如果省略，磁盘会作为原始设备附加，并在 `/dev/disk/by-capv/<name>` 处创建符号链接，以便外部进程在运行时管理该磁盘。
- **`wipeFilesystem`**：仅适用于持久磁盘。当 `true` 时，新 VM 首次启动会擦除磁盘内容。正常重启和手动服务重启不受影响。默认为 `false`。

要附加原始磁盘而不进行格式化或挂载，请省略 `mountPath` 和 `fsFormat`：

```
persistentDisks:
# ...standard persistent disks...
- name: app-data
  sizeGiB: 50
```

磁盘可在 guest OS 中通过 `/dev/disk/by-capv/app-data` 访问。在滚动更新期间，同一个 VMDK 会重新附加到新 VM，并重新创建符号链接。磁盘不会自动格式化或挂载；应用负责在运行时管理该磁盘。

选择持久磁盘或临时磁盘

一个槽位声明两个磁盘列表：`spec.configs[].persistentDisks[]` 和 `spec.configs[].ephemeralDisks[]`。

	`persistentDisks[]`	`ephemeralDisks[]`
Spec 字段	`name`、`sizeGiB`、`datastore`、`storagePolicy`、`unitNumber`、`mountPath`、`mountOptions`、`fsFormat`、`wipeFilesystem`	`name`、`sizeGiB`、`datastore`、`storagePolicy`、`mountPath`、`mountOptions`、`fsFormat`
VM 删除时	分离并保留，然后在同一槽位重新附加到替换 VM	随 VM 一起删除
VM 重新创建时	相同的 VMDK 和相同的内容，除非 `wipeFilesystem: true`	始终创建新的空磁盘，并从头开始格式化
回收和槽位释放门控	参与其中。在仍处于附加状态时阻止槽位释放和池回收	不参与
观测状态	`status.persistentDiskStatuses[]`，包含路径、UUID、单元号、阶段和所属 machine	`status.ephemeralDiskStatuses[]` 仅包含主机名、名称和单元号
SCSI 单元号	使用 `unitNumber` 固定，或由控制器分配后从 status 中读取	仅由控制器分配；没有对应的 spec 字
Guest 寻址	`/dev/disk/by-capv/<name>`。guest 脚本首先通过 VMDK `diskUUID` 定位设备，然后通过 SCSI 单元号定位	`/dev/disk/by-capv/<name>`。gues 脚本仅通过 SCSI 单元号定位设备

对于不应位于系统磁盘上的可重建节点本地数据，请使用 `ephemeralDisks`，包括标准的 `/var/lib/kubelet`、`/var/lib/containerd` 和 `/var/lib/etcd` 磁盘，以及任何临时空间。对于任何丢失即意味着数据丢失的内容，请使用 `persistentDisks`。

> **将标准磁盘保留在其声明的列表中**
>
> `/var/cpaas` 必须保留在 `persistentDisks[]` 中：它包含必须在 VM 替换后保留的平台状态。`/var/lib/kubelet`、`/var/lib/containerd` 和 `/var/lib/etcd` 属于 `ephemeralDisks[]`；它们会在每个新 VM 上重建，将其声明为持久磁盘会把过期的 kubelet、运行时和 etcd 成员数据带到替换节点上。
>
> 此拆分是在创建时选择的。`ephemeralDisks[]` 在 provider `v1.0.17` 中新增，因此，在较早 provider 上创建的集群完全无法使用标准临时磁盘集：它将

`/var/cpaas`、`/var/lib/containerd` 和 `/var/lib/etcd` 声明为持久磁盘，并在 etcd 磁盘上设置 `wipeFilesystem: true`。该集群会保留其布局：升级 provider 不会迁移任何磁盘。请参阅升级 Provider。

以下示例在标准 Worker 磁盘集旁添加一个临时 scratch 磁盘：

```
persistentDisks:
- name: var-cpaas
  sizeGiB: <worker_var_cpaas_size_gib>
  mountPath: /var/cpaas
  fsFormat: ext4
ephemeralDisks:
- name: var-lib-kubelet
  sizeGiB: <worker_var_lib_kubelet_size_gib>
  mountPath: /var/lib/kubelet
  fsFormat: ext4
- name: var-lib-containerd
  sizeGiB: <worker_var_lib_containerd_size_gib>
  mountPath: /var/lib/containerd
  fsFormat: ext4
- name: scratch
  sizeGiB: 50
  mountPath: /var/scratch
  fsFormat: ext4
```

同一槽位的两个列表中的磁盘名称和挂载路径必须唯一。

固定 SCSI 单元号

持久磁盘上的 `unitNumber` 可在 VM 重新创建期间保持 guest 设备顺序稳定。有效值为从 `0` 到 `15`，但不包括为 SCSI 控制器保留的 `7`；每个值在槽位内也必须唯一。

如果省略 `unitNumber`，控制器会在克隆时分配一个值，将其记录在 `status.persistentDiskStatuses[].unitNumber` 中，并在 VM 重新创建时复用该值。

临时磁盘完全没有 `unitNumber` 字段。控制器会在每次克隆时分配其单元号，并在 `status.ephemeralDiskStatuses[].unitNumber` 中报告该单元号，因此无法固定标准的 `/var/lib/kubelet`、`/var/lib/containerd` 和 `/var/lib/etcd` 磁盘。

重定位 pod 日志目录

当槽位中的任何磁盘挂载到 `/var/lib/containerd` 时，guest 磁盘协调脚本会将 `/var/log/pods` 指向该磁盘上的目录，使 pod 日志写入数据磁盘，而不是根文件系统。在标准布局中，该磁盘是临时磁盘，因此重定位的 pod 日志会随其所属 VM 一起销毁；需要保留的内容请发送到平台日志存储中。

> **WARNING**
>
> 此重定位仅通过 `guestinfo` 用户数据写入 guest，且仅在首次启动前进行。升级 provider 不会将其补写到现有 machine 上；节点滚动并替换时才会应用此更改。请参阅不更改 Kubernetes 版本的 OS 映像更新。要就地修复正在运行的节点，需要使用带外脚本，这不属于此操作流程。

验证创建时变体

应用拓扑变体后，使用以下命令验证集群状态：

```
kubectl -n <namespace> get cluster,vspherecluster,kubeadmcontrolplane,machinedeployment,machine,vspheremachine,vspherevm
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> \
  -o jsonpath='{range .status.ephemeralDiskStatuses[*]}{.hostname}{" "}{.name}{" unit="}{.unitNumber}{"\n"}{end}'
```

确认以下结果：

- 目标资源中反映了 placement、NIC 或磁盘定义。
- 新节点进入 `Ready` 状态。
- 更改后现有节点仍保持正常。
- 每个声明的临时磁盘都已分配单元号，并可在 guest 中通过 `/dev/disk/by-capv/<name>` 进行寻址。

后续步骤

有关 worker 扩容以及后续模板或运行时拓扑变更，请参阅 在 VMware vSphere 上管理节点。

每次仅应用一项变更，并在同一集群中合并多项变更之前验证结果。

故障排除

排查卡在 **Provisioned** 状态的业务集群 排查卡在删除中的集群

排查卡在 **Provisioned** 状态的业务集群

当 Alauda OS 上的业务集群已达到 `Cluster.status.phase = Provisioned`，且 Kubernetes 控制平面报告就绪，但集群始终无法完全投入使用时，请使用本指南。

本诊断流程假设业务集群是通过将 provider Cluster API 清单应用到 `global` 集群创建的。

目录

范围
症状
健康的已导入集群的不变量
诊断步骤
步骤 1 — 确认 Cluster API 接口正常
步骤 2 — 检查 Machine 和 Node 状况
步骤 3 — 检查基础设施提供商协调器日志
步骤 4 — 验证 global 集群已导入业务集群
步骤 5 — 验证 ClusterCredential 数量不变量
表示导入失败的模式
此模式无法通过以下方式解决
后续步骤
另请参阅

范围

本指南中的诊断流程针对 `global` 集群的导入控制器，该控制器由所有基础设施提供商共享。健康的已导入集群的不变量 中的四项不变量——`clusters.platform.tkestack.io` 是否存在、`capi.cpaas.io/imported` 标签、`ClusterCredential` 数量，以及业务集群的 `sentry` ServiceAccount——适用于所有提供商。

本指南中的日志字符串使用占位符表示协调器对象名称，因为每个基础设施提供商都使用自己的对象类型（例如，提供商自有的 `Cluster` 和 `Machine` 基础设施 CRD）。查看提供商的日志时，请将占位符替换为提供商协调器输出中显示的实际对象名称。诊断模式本身不依赖于提供商，只有日志字符串中的对象名称会有所不同。

如果您的环境是 Huawei Cloud Stack，且下述症状还包括某个节点上的 `kubeadm init` 停滞，而该节点的池配置中带有带点号的主机名，请在完成本页的通用诊断流程后，参阅排查 HCS 业务集群，了解提供商特有的模式。

症状

集群达到部分成功状态后便停滞不前。以下指标会同时出现：

查看位置	观察结果
`Cluster.status.phase`（位于 `global` 集群上）	`Provisioned`
`KubeadmControlPlane.status.ready`	`true`
`Machine.status.phase`	`Running`
`Machine.status.conditions[?(@.type=="NodeHealthy")]`	`status: False`、`reason: NodeConditionsFailed`、`message: Node condition Ready is False`
`MachineDeployment.status.conditions[?(@.type=="Available")]`	`status: False`、`reason: NotAvailable`、`message: WorkersAvailable: 0 available replicas`

查看位置	观察结果
业务集群内部：`kubectl get nodes`	一个或多个节点报告 `STATUS=NotReady`，因为 CNI 未运行

基础设施提供商协调器日志中通常会在 `global` 集群上出现以下次要特征：

```
ERROR  failed to reconcile <CNI-AppRelease-object>: ServiceAccount "sentr
y" not found
ERROR  failed to reconcile <Infra-Cluster-object>: ServiceAccount "sentr
y" not found
```

`<CNI-AppRelease-object>` 和 `<Infra-Cluster-object>` 是占位符，代表提供商使用的实际对象类型。例如，CNI AppRelease 对象通常以正在部署的 CNI 命名；基础设施集群对象名称则与提供商的 CRD 相匹配（即提供商自有的 `*Cluster` CRD）。

协调器会尝试通过 `AppRelease` 资源应用 CNI，该资源由业务集群自己的 `sentry` ServiceAccount 执行。如果该 ServiceAccount 不存在，CNI 将无法部署，业务节点会一直处于 `NotReady` 状态。

健康的已导入集群的不变量

业务集群达到健康的稳定状态后，必须满足以下所有条件。除非另有说明，请从 `global` 集群上下文中验证每项不变量。

1. 存在一个 `clusters.platform.tkestack.io/<cluster-name>` 对象。

   ```
   kubectl get clusters.platform.tkestack.io <cluster-name>
   ```

2. Cluster API `Cluster` 资源带有标签 `capi.cpaas.io/imported: ""`。
   健康导入时，标签值为空字符串 `""`，因此仅使用 `jsonpath` 查询无法区分“缺少标签”和“标签值为空”。请使用 `jq` 显式检查标签是否存在：

```
kubectl get cluster <cluster-name> -n <namespace> -o json | \
  jq -r 'if (.metadata.labels | has("capi.cpaas.io/imported"))
         then "present (value=\"\(.metadata.labels["capi.cpaas.io/impor
ted"])\")"
         else "missing"
         end'
```

预期输出：`present (value="")`。`missing` 表示导入尚未完成。

3. 恰好有一个 `clustercredentials.platform.tkestack.io` 对象带有该集群的标签。

```
kubectl get clustercredentials.platform.tkestack.io -o json | \
  jq -r --arg c <cluster-name> \
    '.items[] | select(.metadata.labels["cluster.x-k8s.io/cluster-nam
e"]==$c) | .metadata.name'
```

应只打印一个名称。

4. 在业务集群内部，`sentry` ServiceAccount 存在于 `cpaas-system` 中。

```
kubectl --kubeconfig <workload-kubeconfig> -n cpaas-system get servicea
ccount sentry
```

如果任何不变量不成立，则该业务集群尚未被 `global` 集群的导入控制器导入，因此预期会出现上述症状模式。

诊断步骤

步骤 1 — 确认 **Cluster API** 接口正常

```
kubectl get cluster <cluster-name> -n <namespace> -o wide
kubectl get kubeadmcontrolplane -n <namespace>
kubectl get machinedeployment -n <namespace>
```

如果此处失败，则与导入模式无关；请直接调查基础设施提供商。

步骤 2 — 检查 Machine 和 Node 状况

```
kubectl get machine -n <namespace> \
  -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{" phase="}{.status.phase}{" "}{range .status.conditions[*]}{.type}={.status} {end}{"\n"}{end}'
```

如果 `phase=Running`，但 `NodeHealthy` 状况为 `False`，则底层业务节点处于 `NotReady` 状态。继续执行步骤 3。

步骤 3 — 检查基础设施提供商协调器日志

```
kubectl logs -n cpaas-system <infra-provider-manager-pod> --tail=200 | \
  grep -E 'ServiceAccount|sentry|<cluster-name>'
```

如果看到重复的 `ServiceAccount "sentry" not found` 错误，同时 CNI AppRelease 对象协调失败，则说明由于导入流程尚未完成，业务集群的 CNI 未能部署。继续执行步骤 4。

步骤 4 — 验证 global 集群已导入业务集群

运行上一节中的不变量检查。导入失败的综合诊断特征如下：

- `clusters.platform.tkestack.io/<cluster-name>` 返回 `NotFound`。
- `Cluster.metadata.labels` 不包含 `capi.cpaas.io/imported`。

步骤 5 — 验证 ClusterCredential 数量不变量

```
kubectl get clustercredentials.platform.tkestack.io -o json | \
  jq -r --arg c <cluster-name> \
    '[.items[] | select(.metadata.labels["cluster.x-k8s.io/cluster-name"]==$c)] | length'
```

健康的集群会打印 `1`。大于 `1` 的值（例如 `8`）表示导入流程反复尝试但均失败，留下了孤立的凭据。

表示导入失败的模式

当以下所有条件同时满足时，诊断结果表明导入失败：

- Cluster API 表面正常（`Cluster.status.phase=Provisioned`、`KubeadmControlPlane.status.ready=true`）。
- `Machine.condition.NodeHealthy=False` 以及至少一个业务节点处于 `NotReady` 状态。
- 基础设施 provider 日志反复报告 `ServiceAccount "sentry" not found`。
- `clusters.platform.tkestack.io/<cluster-name>` 不存在。
- Cluster 资源不包含 `capi.cpaas.io/imported` 标签。
- 多个 `clustercredentials.platform.tkestack.io` 对象带有该集群的标签。

当出现此模式时，业务集群在 `global` 集群上的初始导入失败，导入控制器当前不会从头开始重试。

此模式无法通过以下方式解决

> **WARNING**
>
> 在 `global` 集群上重启 cluster-transformer pod，不会重新导入其初始导入已按此模式失败的业务集群。手动重启 pod 后，日志中不会再出现受影响集群的条目，并且不会创建缺失的 `clusters.platform.tkestack.io` 条目。

当同一个集群存在多个孤立的 `ClusterCredential` 时，不要假定控制器最终会自行恢复。

后续步骤

请向平台支持团队提供以下信息：

- 集群名称和命名空间。
- 步骤 1 至 5 中诊断命令的输出。

- 与受影响集群共享同一 `cluster.x-k8s.io/cluster-name` 标签的每个 `clustercredentials.platform.tkestack.io` 对象的名称和创建时间戳。

当前的恢复方式要求先清理孤立的 `ClusterCredential` 条目，然后删除并重新创建业务集群。未经平台支持团队指导，请勿在生产集群上执行此操作：重新创建流程会触发 IaaS 平台上底层虚拟机的删除，并且在执行重新创建前，必须审查特定于 provider 的详细信息（例如 DCS IP 池和主机名规划，或 HCS `HCSMachineConfigPool` 重新分配）。

另请参阅

- 排查卡在 Deleting 状态的集群 — 介绍由 ACP 构建、卡在 `Deleting` 而非导入失败的集群。
- 排查卡在 Deleting 状态的 DCSMachine — 节点级删除停滞；范围不同。

排查卡在删除中的集群

目录

范围
症状
诊断
原因
临时解决方法
为什么不能只是等待？
永久修复
另请参阅

范围

本页面介绍以下情况：一个由 **ACP** 构建的业务集群正在被删除，所有子资源（特定于 provider 的 `Machine` CR 以及它们所代表的 IaaS VM）均已删除，但顶层 `Cluster` CR 仍无限期保留在 `Phase: Deleting` 中。

本页面适用于任何由 ACP 构建的业务集群，与底层 IaaS 无关：

- Huawei DCS（DCS Provider）
- Huawei Cloud Stack（HCS Provider）

- VMware vSphere（vSphere Provider）
- 裸金属（Bare Metal Provider）
- 通过 SSH UPI 路径创建并由 `cluster-manager` 管理的集群

直接观察到的证据来自 DCS；其他 provider 共享相同的 `global`-集群控制平面路径。

不在范围内：

- 特定于 provider 的 `Machine` CR（例如 `DCSMachine`）卡在 `Deleting` 中，而 IaaS VM 仍然存在 — 请参阅排查卡在删除中的 DCSMachine。
- 已导入（作为 Third-party Cluster 接入）但从未达到就绪状态的业务集群 — 请参阅排查卡在 Provisioned 中的业务集群。

症状

Cluster CR 显示以下模式。

查看位置	看到的内容
`global` 集群上的 `Cluster.status.phase`	`Deleting`，持续时间远超预期 — 通常为一小时到数小时。
`Cluster.metadata.deletionTimestamp`	已设置。
`Cluster.metadata.finalizers`	仍包含 `capi.cpaas.io/imported`。
`Cluster.spec.infrastructureRef` 和 `Cluster.spec.controlPlaneRef`	两者均已设置（例如 `infrastructureRef.kind: DCSCluster`、`controlPlaneRef.kind: KubeadmControlPlane`）。这标志着该集群由 ACP 构建 — Third-party Cluster 不包含这两个引用中的任何一个。
同一命名空间中的特定于 provider 的 `Machine` CR（`DCSMachine` / `HCSMachine` / `VSphereMachine` / 等），以及匹配的基础设施 `Cluster`、`KubeadmControlPlane`、`MachineDeployment` 和 `Machine` 资源	均不存在 — 级联删除已完成。

查看位置	看到的内容
IaaS 平台本身（DCS 门户、vCenter、HCS 门户、裸金属清单）	集群的 VM 均不存在。
Provider controller 日志（例如 `cluster-api-provider-dcs-manager`）	显示成功删除机器的条目，例如 `removed DCSMachine finalizer`。

`capi.cpaas.io/imported` finalizer（以及 `capi.cpaas.io/alauda-cluster: imported` 标签，如果存在）本身并不意味着该集群已被导入 — 由 ACP 构建的集群也会携带这些内容。请使用上方的 `infrastructureRef` / `controlPlaneRef` 行来区分两者。

诊断

在 `global` 集群上运行以下两项检查，以确认该模式。

确认 `capi.cpaas.io/imported` finalizer 仍然存在：

```
kubectl get cluster <name> -n <ns> -o yaml | grep -A 2 finalizers
```

确认该集群由 ACP 构建。由 ACP 构建的集群会同时设置 `spec.infrastructureRef` 和 `spec.controlPlaneRef`；Third-party Cluster（通过导入接入）则两者均未设置：

```
kubectl get cluster <name> -n <ns> \
  -o jsonpath='infrastructureRef={.spec.infrastructureRef.kind} controlPlaneRef={.spec.controlPlaneRef.kind}{"\n"}'
```

如果两个值均非空（例如 `infrastructureRef=DCSCluster controlPlaneRef=KubeadmControlPlane`），则该集群由 ACP 构建，适用本页中的临时解决方法。如果两个值均为空，则该集群是作为 Third-party Cluster 接入的；本页不适用，且不得使用下方的临时解决方法。

原因

`Cluster` CR 上的 `capi.cpaas.io/imported` finalizer 由 `global` 集群上的平台 controller 管理。对于作为 Third-party Cluster 接入的集群，移除集群时，该 finalizer 会作为平台侧清理的一部分被释放。

由 ACP 构建的集群也带有相同的 finalizer。删除由 ACP 构建的集群时，其子资源和 IaaS VM 都会被正确移除，但顶层 `Cluster` CR 上的 `capi.cpaas.io/imported` finalizer 不会被释放，因此该 CR 会一直处于 `Deleting` 状态。

这是平台级行为，并非任何 IaaS provider 的故障，也不会影响已导入的（Third-party）集群。平台团队正在调查具体的处理方式。

临时解决方法

清除 finalizer 之前，请确认没有任何内容依赖 `Cluster` CR 的持续存在 — 每个子资源和每个 IaaS VM 都必须已经删除。

步骤 **1** — 确认命名空间中的所有特定于 **provider** 的 `Machine` **CR** 均已删除：

```
# Replace dcsmachines with the resource for your provider (hcsmachines, v
spheremachines, ...).
kubectl get dcsmachines,hcsmachines,vspheremachines -n <ns>

# Also confirm the higher-level CAPI resources are gone:
kubectl get machines,machinedeployments,kubeadmcontrolplanes,dcsclusters,
hcsclusters,vsphereclusters -n <ns>
```

每条命令对于此集群都必须返回无资源。

步骤 **2** — 确认集群的 **VM** 已从 **IaaS** 平台中删除。 使用特定于 provider 的验证方法：

- DCS：通过 DCS 门户或 DCS API 列出 VM，并确认集群的 VM 均已不存在。相关集群字段请参阅在 Huawei DCS 上创建集群。
- HCS：在 HCS 门户中列出 VM，并确认均已不存在。
- vSphere：在 vCenter 中集群文件夹下列出 VM，并确认均已不存在。
- 裸金属：确认裸金属清单中的机器取消配置已完成。

步骤 **3** — 只有在步骤 **1** 和步骤 **2** 均确认没有剩余资源后，才能清除 Cluster CR 的 finalizer：

```
kubectl patch cluster <name> -n <ns> --type=merge \
  -p '{"metadata":{"finalizers":null}}'
```

Cluster CR 会立即消失。由于每个子资源和每个 IaaS VM 在此步骤之前都已删除，移除顶层 finalizer 不会遗留孤立资源 — 它只会清除一个已没有任何愿意释放它的 reconciler 的标记。

为什么不能只是等待？

finalizer 不会在计时器到期后释放；对于由 ACP 构建的集群，正常的平台清理路径也不会释放它。等待不会改变状态 — 已记录有持续九小时或更长时间仍无进展的情况。确认步骤 1 和步骤 2 完成后，请立即应用上述变通方法。

永久修复

修复应位于 `global`-cluster 控制平面中：对于由 ACP 构建的集群，一旦其子资源已删除，就必须释放 `capi.cpaas.io/imported` finalizer。此问题的跟踪在本文档集之外；此处不承诺发布日期。在修复发布之前，上述变通方法是受支持的恢复路径。

另请参阅

- 排查删除卡住的 DCSMachine — 节点级删除停滞；范围不同。
- 排查卡在 Provisioned 状态的业务集群 — 涵盖真正的 Third-party Cluster 启动问题；范围不同。

在非 Alauda OS 上创建集群

目录

前提条件
节点要求
负载均衡
连接 `global` 集群和业务集群
镜像仓库
容器网络
创建操作步骤
基本信息
容器网络
节点设置
扩展参数
创建后的操作步骤
查看创建进度
关联项目

前提条件

节点要求

1. 如果你从下载安装包下载了单架构安装包，请确保节点机器的架构与该安装包相同。否则，节点会因缺少特定架构的镜像而无法启动。
2. 确认节点操作系统和内核受支持。详情请参阅支持的操作系统和内核。
3. 对节点机器执行可用性检查。具体检查项请参阅节点预处理 > 节点检查。
4. 如果无法通过 SSH 直接访问节点机器的 IP，请为节点提供 SOCKS5 代理。`global` 集群将通过此代理服务访问节点。

负载均衡

对于生产环境，需要为集群控制平面节点配置负载均衡器，以确保高可用性。

如果环境中有可用的硬件 LoadBalancer，建议使用该负载均衡器。 如果没有，可以启用 `Self-built VIP`，通过 keepalived 提供软件负载均衡。

建议：使用硬件 LoadBalancer 时，建议使用域名配置 Cluster Endpoint。如果集群安装后硬件 LoadBalancer 的 VIP 发生变化，可以更新 DNS 记录，而无需重新配置集群。

注意：`Self-built VIP` 不支持域名配置。

负载均衡器必须能够将流量正确转发到集群中所有控制平面节点上的端口 `6443` 和 `11780`。

如果集群只有一个控制平面节点，并将该节点的 IP 用作 Cluster Endpoint 参数，则之后无法将集群从单节点扩展为高可用多节点集群。因此，即使是单节点集群，也建议配置负载均衡器。

启用 `Self-built VIP` 时，需要准备：

1. 可用的 VRID
2. 支持 VRRP 协议的主机网络
3. 所有控制平面节点和 VIP 必须位于同一子网中，且 VIP 必须与任何节点 IP 不同。

连接 `global` 集群和业务集群

平台要求 `global` 集群与业务集群之间可以相互访问。如果它们不在同一网络中，则需要：

1. 为业务集群提供 `External Access`，确保 `global` 集群可以访问该业务集群。网络必须满足以下要求：`global` 可以访问所有控制平面节点上的端口 `6443` 和 `11780`。

2. 为 `global` 添加一个业务集群可以访问的额外地址。创建业务集群时，将此地址添加到集群的注解中，键为 `cpaas.io/platform-url`，值设置为 `global` 的公共访问地址。

镜像仓库

集群镜像支持平台内置、私有仓库和公共仓库选项。这些选项用于配置业务集群使用的平台镜像源。

- 平台内置：使用为 `global` 集群配置的平台镜像源。此镜像源可以是通过 `global` 部署的 Registry，也可以是安装期间选择的外部平台镜像仓库。如果镜像源是内置 Registry，且业务集群无法访问 `global`，请参阅为内置 Registry 添加外部地址。
- 私有仓库：为此业务集群使用另一个由客户管理的镜像仓库。选择此选项前，请上传所选 Kubernetes 版本和架构所需的完整负载，并满足准备外部平台镜像仓库中所述的功能要求和访问条件。用于生产环境时，还应遵循有关 HTTPS、DNS、可用性、监控、备份、恢复和保留的生产建议。
- 公共仓库：使用平台的公共镜像仓库。使用前，请完成更新公共仓库凭据。

容器网络

如果计划在集群中使用 Kube-OVN 的 Underlay，请准备好 Kube-OVN Underlay 物理网络。

创建操作步骤

1. 进入管理员视图，点击左侧导航栏中的集群/集群。
2. 点击创建集群。
3. 按照以下说明配置各部分：基本信息、容器网络、节点设置和扩展参数。

基本信息

参数	描述
Kubernetes 版本	所有可选版本均经过严格测试，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。
容器运行时	默认提供 Containerd 作为容器运行时。
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，请确保所有节点均已正确配置 IPv6 地址；设置后无法更改网络协议。
集群端点	`IP 地址 / 域名`：输入预先准备的域名；如果没有可用的域名，则输入 VIP。 `自建 VIP`：默认禁用。仅在未提供 LoadBalancer 时启用。启用后，安装程序将自动部署 `keepalived`，以支持软件负载均衡。 `外部访问`：当集群与 `global` 集群不在同一网络环境中时，输入为集群准备的可从外部访问的地址。

容器网络

Kube-OVN

Kube-OVN 是由 灵雀云 开发的云原生 Kubernetes 容器网络编排系统。它为跨云网络管理、传统网络架构集成、基础设施互联和边缘集群部署场景提供 Kubernetes 网络能力。

参数	描述

子网	也称为 Cluster CIDR，表示默认子网段。创建集群后，可以添加其他子网。
传输模式	**Overlay**：在基础设施之上抽象出的虚拟网络，不会占用物理网络资源。创建 Overlay 默认子网时，集群中的所有 Overlay 子网都使用相同的集群网卡和节点网卡配置。 **Underlay**：此传输方式依赖物理网络设备。它可以直接为 Pod 分配物理网络地址，从而确保更高的性能以及与物理网络的更好连接。Underlay 子网中的节点必须具有多个网卡，并且用于网桥网络的网卡必须专供 Underlay 使用，不得承载 SSH 等其他流量。创建 Underlay 默认子网时，集群网卡实际上是用于网桥网络的默认网卡，节点网卡则是网桥网络中的节点网卡配置。 • 默认网关：物理网络网关地址，即 Cluster CIDR 网段的网关地址（必须位于 Cluster CIDR 地址范围内）。 • **VLAN ID**：虚拟局域网标识符（VLAN 编号），例如 `0`。 • 保留 **IP**：设置不会自动分配的保留 IP，例如子网中已被其他设备使用的 IP。
Service CIDR	Kubernetes ClusterIP 类型 Service 使用的 IP 地址范围。不能与默认子网范围重叠。
Join CIDR	在 Overlay 传输模式下，这是用于节点与 Pod 之间通信的 IP 地址范围。不能与默认子网或 Service CIDR 重叠。

Custom

如果需要安装其他网络插件，请选择自定义模式。集群成功创建后，可以手动安装网络插件。

参数	描述
Cluster CIDR	集群启动时创建的 Pod 使用的 IP 地址范围。
Service CIDR	Kubernetes ClusterIP 类型 Service 使用的 IP 地址范围。不能与容器子网范围重叠。

节点设置

参数	描述
网络接口卡	集群网络插件使用的主机网络接口设备名称。 注意： • 为 Kube-OVN 默认子网选择 Underlay 传输模式时，必须指定网络接口名称，该接口将作为用于网桥网络的默认网卡。 - 平台默认通过接口名称 `eth.|en.|wl.|ww.` 识别网络接口流量。如果使用其他命名规则的接口，请在集群接入后参阅从自定义名称的网络接口收集网络数据，修改相关资源，以确保平台能够正确监控网络接口流量。
节点名称	可以选择使用节点 IP 或主机名作为平台中的节点名称。 注意：选择使用主机名作为节点名称时，请确保添加到集群的节点主机名唯一。

节点	向集群中添加节点，或从草稿恢复临时保存的节点信息。有关添加节点的详细参数说明，请参阅下文。
监控类型	支持 **Prometheus** 和 **VictoriaMetrics**。 选择 **VictoriaMetrics** 作为监控组件时，必须配置部署类型： **-** 部署 **VictoriaMetrics**：部署所有相关组件，包括 **VMStorage**、**VMAlert**、**VMAgent** 等。 **-** 部署 **VictoriaMetrics Agent**：仅部署日志采集组件 **VMAgent**。使用此部署方式时，需要关联平台中已在其他集群上部署的 VictoriaMetrics 实例，为该集群提供监控服务。
监控节点	选择用于部署集群监控组件的节点。支持选择允许部署原生应用的工作节点和控制平面节点。 为避免影响集群性能，建议优先选择工作节点。集群成功创建后，存储类型为 **Local Volume** 的监控组件将部署在所选节点上。

节点添加参数

参数	描述

类型	控制平面节点：负责在集群中运行 kube-apiserver、kube-scheduler、kube-controller-manager、etcd、容器网络和部分平台管理组件等。当 允许部署原生应用启用时，控制平面节点也可以用作工作节点。 工作节点：负责承载在集群上运行的业务 Pod。
IPv4 地址	节点的 IPv4 地址。对于在内部网络模式下创建的集群，请输入节点的私有 **IP**。
IPv6 地址	在集群启用 IPv4/IPv6 双栈时有效。节点的 IPv6 地址。
允许部署原生应用	当节点类型为控制平面节点时有效。是否允许在此控制平面节点上部署业务应用，将业务相关 Pod 调度到此节点。
显示名称	节点的显示名称。
SSH 连接 **IP**	通过 SSH 服务访问节点时可用于连接节点的 IP 地址。 如果可以使用 `ssh <username>@<node's IPv4 address>` 登录节点，则无需设置此参数；否则请输入节点的公共 IP 或 NAT 外部 IP，以确保 `global` 集群和代理可以通过此 IP 连接到节点。
网络接口卡	输入节点使用的网络接口名称。网络接口配置的生效优先级如下（从左到右依次降低）： **Kube-OVN Underlay**：节点网卡 > 集群网卡

	Kube-OVN Overlay：节点网卡 > 集群网卡 > 与节点默认路由对应的网卡 **Calico**：集群网卡 > 与节点默认路由对应的网卡 **Flannel**：集群网卡 > 与节点默认路由对应的网卡
关联的网桥网络	注意：创建集群时不支持配置网桥网络；此选项仅在向已创建 Underlay 子网的集群中添加节点时可用。 选择现有的添加网桥网络。 如果不想使用网桥网络的默认网卡，可以单独配置节点网卡。
SSH 端口	SSH 服务端口号，例如 `22`。
SSH 用户名	SSH 用户名，需要使用具有 root 权限的用户，例如 `root`。
代理	是否通过代理访问节点的 SSH 端口。当 `global` 集群无法通过 SSH 直接访问待添加节点时（例如，`global` 集群与业务集群不在同一子网中；节点 IP 是 `global` 集群无法直接访问的内部 IP），需要开启此开关并配置代理相关参数。配置代理后，可以通过代理访问节点并进行部署。 注意：当前仅支持 SOCKS5 代理。

	访问 **URL**：代理服务器地址，例如 `192.168.1.1:1080` 。 用户名：访问代理服务器的用户名。 密码：访问代理服务器的密码。
SSH 身份验证	登录所添加节点的身份验证方式及相应的身份验证信息。选项包括： 密码：需要具有 root 权限的用户名及对应的**SSH** 密码。 密钥：需要具有 root 权限的私钥及 私钥密码。
保存草稿	将对话框中当前配置的数据保存为草稿，并关闭添加节点对话框。 无需离开创建集群页面，即可选择 从草稿恢复，打开添加节点对话框并恢复保存为草稿的配置数据。 注意：从草稿恢复的数据是最近保存的草稿数据。

扩展参数

注意：

- 除必需配置外，不建议设置扩展参数，因为错误的设置可能导致集群不可用，并且在集群创建后无法修改。
- 如果输入的键与默认参数的键重复，则会覆盖默认配置。

操作步骤

1. 点击扩展参数，展开扩展参数配置区域。可以为集群选择性设置以下扩展参数：

参数	描述
Kubelet 参数	`kubeletExtraArgs`，Kubelet 的其他配置参数。 注意：输入容器网络的节点 **IP** 数量 参数后，系统会自动生成默认的**Kubelet** 参数配置，其键为 `max-pods`，值为节点 **IP** 数量。此配置用于设置集群中任意节点上可以运行的 Pod 最大数量。此配置不会显示在界面中。 在**Kubelet** 参数区域中添加新的 `max-pods: 可运行的 Pod 最大数量` 键值对将覆盖默认值。允许输入任意正整数，但建议使用默认值（节点 IP 数量），或输入不超过 `256` 的值。
Controller Manager 参数	`controllerManagerExtraArgs`，Controller Manager 的其他配置参数。
Scheduler 参数	`schedulerExtraArgs`，Scheduler 的其他配置参数。
APIServer 参数	`apiServerExtraArgs`，APIServer 的其他配置参数。
APIServer URL	`publicAlternativeNames`，证书中签发的 APIServer 访问地址。只能输入 IP 或域名，最多 253 个字符。
集群注解	集群注解信息，以键值对的形式在元数据中标记集群特征，供平台组件或业务组件获取相关信息。

4. 点击创建。系统将返回集群列表页面，此时集群状态为创建中。

创建后的操作步骤

查看创建进度

在集群列表页面中，可以查看已创建的集群列表。对于状态为创建中的集群，可以检查执行进度。

操作步骤

1. 点击集群状态右侧的查看执行进度小图标。
2. 在显示的执行进度对话框中，可以查看集群的执行进度（status.conditions）。
   提示：当某一类型正在执行或处于失败状态且有失败原因时，将光标悬停在对应的原因（以蓝色文本显示）上，即可查看该原因的详细信息（status.conditions.reason）。

关联项目

集群创建完成后，可以在项目管理视图中将其添加到项目中。

托管集群

托管集群

托管集群

导入第三方集群

导入第三方集群

导入标准 **Kubernetes** 集群

导入 **OpenS**

导入 **Amazon EKS** 集群

导入 **GKE** 集群

导入 **Huawe**

将现有 CCE 集

导入 **Azure AKS** 集群

导入 **Alibaba Cloud ACK** 集群

导入 **Tencer**

注册集群

注册集群

公有云集群初始化

网络初始化

公有云集群网络初始化。

存储初始化

公有云集群存储初始化。

操作指南

导入集群的网络配置

获取导入集群信息

信任不安全的

从自定义命名的网卡采集网络数据

从自定义命名的网卡采集网络数据

为导入的标准 **Kubernetes** 集群配置审计收集

在导入的标准 Kubernetes 集群上启用 Kubernetes 审计日志记

第三方集群接入

第三方集群是指其 Kubernetes 发行版、节点生命周期或提供商基础设施由 灵雀云 外部管理的现有 Kubernetes 环境。在控制台中，这些环境显示在托管集群下。

当连接、凭据、组件前置条件、提供商要求和 Extension 兼容性均得到满足时，ACP 可以为第三方集群提供集中式治理、资源可见性、Extension 安装、应用运维和可观测性集成。接入并不意味着 ACP 成为外部集群 Kubernetes 生命周期、节点生命周期或提供商基础设施生命周期的所有者。

目录

选择接入方式

选择接入方式

方法	连接模型	适用场景
导入集群	`global` 集群使用提供的地址、CA 和凭据连接到目标集群 API 服务器。	平台可以访问目标 API 服务器，并且管理员可以提供所需的集群信息。
注册集群	目标集群中的反向代理服务发起注册，并与平台建立隧道。	应由目标集群发起连接，或者从 `global` 集群直接访问目标 API 服务器受到限制。

接入后，高层次的 day-2 操作在集群列表中保持一致。节点操作、审计数据、控制平面指标、证书、入口、存储和 Extension 兼容性仍可能受到提供商特定限制。

有关导入流程，请参阅导入第三方集群。有关反向连接接入，请参阅注册集群。

导入第三方集群

导入第三方集群

导入标准 **Kubernetes** 集群

导入 **OpenS**

导入 **Amazon EKS** 集群

导入 **GKE** 集群

导入 **Huawe**

将现有 CCE 集

导入 **Azure AKS** 集群

导入 **Alibaba Cloud ACK** 集群

导入 **Tencer**

导入第三方集群

当 `global` 集群可以访问目标集群 API 服务器，且管理员能够提供所需的集群地址、CA 和凭据时，请使用导入方式。

目标环境	从以下内容开始
现有标准 Kubernetes 环境	导入标准 Kubernetes 集群
外部 Kubernetes 发行版	导入 OpenShift 集群
公有云 Kubernetes 服务	导入 Amazon EKS 集群、导入 GKE 集群、导入 Azure AKS 集群、导入阿里云 ACK 集群、导入腾讯云 TKE 集群或导入华为云 CCE 集群

特定于提供商的操作步骤会在必要时包含提供商控制台或 CLI 中的操作。提供商名称用于标识操作步骤路径；它不是单独的 ACP 集群模型。

导入标准 Kubernetes 集群

使用此操作步骤将现有的标准 Kubernetes 集群（例如使用 **kubeadm** 部署的集群）作为第三方集群导入。

目录

术语
前提条件
注意事项
获取 Registry 地址
检查是否需要额外的 Registry 配置
获取集群信息
集成集群
网络配置
导入后配置
常见问题
为什么“添加节点”按钮处于禁用状态？
支持哪些证书？
哪些功能不受支持？
如何修复 Containerd 运行时导致的分布式存储部署失败？

术语

术语	描述
Provider-managed Kubernetes service	由提供商管理控制平面节点和控制平面组件的 Kubernetes 服务。用户通常无法登录控制平面节点，也无法直接维护这些节点。
Self-managed Kubernetes cluster	由集群所有者维护控制平面节点和集群组件的 Kubernetes 集群。

前提条件

- 集群中的 Kubernetes 及相关组件必须满足版本和参数要求。
- 如果运行时为 Containerd，请在集成前更新 Containerd 配置，以确保能够成功部署分布式存储。

注意事项

默认情况下，平台会监控与 `eth.*|en.*|wl.*|ww.*` 匹配的 NIC 流量。如果您的 NIC 使用不同的命名约定，请按照从自定义命名的网卡收集网络数据中的说明，在集成后更新配置。

获取 **Registry** 地址

- 若要使用平台在安装 **global** 集群期间部署的 Registry，请在 global 控制平面节点上运行以下命令：

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.preferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.platformURL}' | awk -F // '{print $NF}')
fi
echo "Registry address: $REGISTRY"
```

- 若要使用外部 **Registry**，请手动设置 **REGISTRY**：

```
REGISTRY=<external-registry-address>  # e.g., registry.example.cn:60080 or 192.168.134.43
echo "Registry address: $REGISTRY"
```

检查是否需要额外的 **Registry** 配置

1. 运行以下命令，检查 Registry 是否支持使用受信任 CA 证书的 HTTPS：

```
REGISTRY=<registry-address-from-previous-step>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Pass: Registry uses a trusted CA certificate. No extra config needed.'
else
    echo 'Fail: Registry does not support HTTPS or uses an untrusted certificate. Follow "Trust Insecure Registry".'
fi
```

2. 如果检查失败，请参阅如何信任不安全的 Registry？

获取集群信息

请参阅如何获取集群信息？。

集成集群

1. 在左侧导航栏中，转到 集群管理 > 集群。
2. 单击 导入集群。
3. 按如下所示配置参数：

参数	描述
Registry	存储平台所需组件镜像的 Registry。选项包括：平台默认（在 global 设置期间配置）、私有 **Registry**（需要地址、端口、用户名和密码）以及公共 **Registry**（需要更新云凭证）。
集群信息	可以手动输入，也可以从 KubeConfig 文件解析。必填字段包括：集群地址、**CA** 证书（手动输入时需进行 Base64 解码）和身份验证（具有 cluster-admin 权限的令牌或客户端证书）。

4. 单击 检查连接。平台会验证网络访问并自动检测集群类型。
5. 如果成功，请单击 导入 完成操作。
   可以通过执行进度对话框（status.conditions）查看进度。集成后，该集群会在列表中显示为健康状态。

网络配置

确保 global 集群与导入的集群之间可以连通。

导入后配置

如果需要平台从导入的标准 Kubernetes 集群收集审计数据，请在导入后为该集群配置 Kubernetes API server 审计日志记录。请参阅如何为导入的标准 Kubernetes 集群配置审计数据收集？。

常见问题

为什么“添加节点”按钮处于禁用状态？

平台 UI 不支持为导入的标准 Kubernetes 集群添加节点。请直接在目标集群中或通过集群提供商添加节点。

支持哪些证书？

1. **Kubernetes** 证书：只能查看 API Server 证书；不支持其他证书，也不会自动轮换。
2. 平台组件证书：可以查看并自动轮换。

哪些功能不受支持？

- 由提供商管理的 **Kubernetes** 服务：无法获取审计日志。
- 由提供商管理的 **Kubernetes** 服务：不支持监控 ETCD、Scheduler 和 Controller Manager。仅提供 API Server 指标。
- 所有集群：不支持 API Server 以外的证书。

如何修复 **Containerd** 运行时导致的分布式存储部署失败？

使用 Containerd 时，除非调整所有节点上的 Containerd 设置，否则分布式存储部署会失败：

1. 编辑 `/etc/systemd/system/containerd.service`，设置 `LimitNOFILE=1048576`。
2. 运行 `systemctl daemon-reload`。
3. 重启 Containerd：`systemctl restart containerd`。
4. 在控制平面节点上，重启分布式存储 Pod：

```
kubectl delete pod --all -n rook-ceph
```

导入 OpenShift 集群

使用此 provider-specific 操作步骤将现有 OpenShift 集群作为第三方集群导入。除非有文档说明的 provider-specific 功能提供相应的生命周期管理，否则外部发行版所有者仍负责 Kubernetes 发行版、节点生命周期和 provider 基础设施生命周期。

目录

前提条件
获取 Registry 地址
检查是否需要额外的 Registry 配置
信任不安全的 Registry
为集群配置 DNS
获取集群信息
方法 1（推荐）：获取 KubeConfig 文件
方法 2：使用令牌、API Server 地址和 CA 证书
导入集群
网络配置
部署附加组件
更新审计策略
常见问题
为什么“添加节点”按钮处于禁用状态？
支持哪些证书？
此提供程序专用导入路径的限制

前提条件

- 集群的 Kubernetes 版本和参数必须满足 Standard Kubernetes Cluster Requirements。
- 集成期间需要使用 `kubectl` 命令。在能够访问集群的 bastion 主机上安装 CLI 工具。
- 若要启用对节点、工作负载（Deployment、StatefulSet、DaemonSet）、Pod 和容器等指标的实时监控，请确保目标集群中已部署 **Prometheus**。

获取 **Registry** 地址

- 若要使用平台在 **global** 集群安装期间部署的 registry，请在 global 控制平面节点上运行以下命令：

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.prefe
rPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.platformURL}' | awk -F // '{print $NF}')
fi
echo "Registry address is: $REGISTRY"
```

- 若要使用外部 **registry**，请手动设置 **REGISTRY** 变量：

```
REGISTRY=<external-registry-address>  # e.g., registry.example.cn:60080 o
r 192.168.134.43
echo "Registry address is: $REGISTRY"
```

检查是否需要额外的 **Registry** 配置

1. 运行以下命令，检查 registry 是否支持 HTTPS，以及是否使用受信任的 CA 证书：

```
REGISTRY=<registry-address-from-previous-step>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Pass: Registry uses a trusted CA certificate. No extra config needed.'
else
    echo 'Fail: Registry does not support HTTPS or uses an untrusted certificate. Follow "Trust Insecure Registry".'
fi
```

2. 如果检查失败，请按照以下步骤操作。

信任不安全的 **Registry**

1. 登录所有 **OCP** 集群节点。
2. 在每个节点上配置 registry 设置：

```
sudo -i
sudo chattr -i /

sudo mkdir -p /etc/systemd/system/crio.service.d/
cat | sudo tee /etc/systemd/system/crio.service.d/99-registry-cpaas-system.conf << 'EOF'
[Service]
ExecStart=
ExecStart=/usr/bin/crio \
          --insecure-registry='<registry-address>' \  # e.g., registry.example.cn:60080 or 192.168.134.43
          $CRIO_CONFIG_OPTIONS \
          $CRIO_RUNTIME_OPTIONS \
          $CRIO_STORAGE_OPTIONS \
          $CRIO_NETWORK_OPTIONS \
          $CRIO_METRICS_OPTIONS
EOF
```

3. 重启 `crio` ：

```
sudo systemctl daemon-reload && sudo systemctl restart crio
```

为集群配置 DNS

修改 global 集群中的 CoreDNS `ConfigMap` 以配置 DNS。

1. 在 bastion 主机上获取 OCP 集群基础域：

   ```
   oc get dns cluster -o jsonpath='{.spec.baseDomain}'
   ```

   示例输出：

   ```
   ocp.example.com
   ```

2. 登录平台管理控制台，切换到 **global** 集群，然后转到 集群管理 > 资源管理。

3. 在 `kube-system` 命名空间中编辑 `cpaas-coredns` `ConfigMap`。
   使用 OCP 基础域和 DNS 服务器地址（从集群节点上的 `/etc/resolv.conf` 获取）添加一个新块。
   示例：

   ```
   Corefile: |
   ocp.example.com:1053 {
       log
       forward . 192.168.31.220
   }
   .:1053 {
       log
       forward . 192.168.31.220
   }
   ```

获取集群信息

选择以下方法之一：

方法 1（推荐）：获取 KubeConfig 文件

1. 在 bastion 主机上搜索 `kubeconfig` 文件，并确认其中包含 admin 上下文。
2. 将 kubeconfig 文件从 bastion 主机复制到本地计算机：

```
scp root@<bastion-ip>:</path/to/kubeconfig> <local-path>
```

方法 2：使用令牌、API Server 地址和 CA 证书

请参阅如何获取集群信息？。

导入集群

1. 在左侧导航栏中，转到集群管理 > 集群。
2. 单击导入集群。
3. 配置参数：

参数	描述
Registry	存储平台组件镜像的 Registry。平台默认：全局设置期间配置的 Registry。私有 **Registry**：需要提供 Registry 地址、端口、用户名和密码。公有 **Registry**：需要更新云凭据。
集群信息	上传 KubeConfig 文件或手动输入。集群地址：API Server 地址。**CA** 证书：经过 Base64 解码的 CA 证书。身份验证：具有 cluster-admin 权限的令牌或客户端证书。

4. 单击检查连接。
5. 如果检查成功，单击导入。可以在执行日志中查看进度。导入后，集群会在列表中显示为运行正常。

网络配置

确保 global 集群与导入的集群之间的网络连通性。请参阅导入集群的网络配置。

部署附加组件

集成成功后，前往 **Marketplace** 部署所需的附加组件，例如监控、日志收集和日志存储。

部署日志收集之前，请确保 `/var/cpaas/` *有超过 50GB 的可用空间：*

```
df -h /var/cpaas
```

更新审计策略

您可以修改集群的审计策略（`spec.audit.profile`）：

- **Default**：记录读写请求的元数据（OAuth 访问令牌创建日志会记录请求正文）。
- **WriteRequestBodies**：记录所有请求的元数据，以及写入请求的正文。
- **AllRequestBodies**：记录所有请求的元数据和正文。

敏感资源（例如 Secrets、Routes、OAuthClient）仅记录元数据。

使用以下命令更新：

```
oc edit apiserver cluster
```

常见问题

为什么“添加节点”按钮处于禁用状态？

平台 UI 不支持添加节点。请使用发行版提供程序的方法。

支持哪些证书？

1. **Kubernetes** 证书：仅显示 API Server 证书，不会自动轮换。
2. 平台组件证书：可见且会自动轮换。

此提供程序专用导入路径的限制

- 审计数据收集。
- ETCD、Scheduler、Controller Manager 监控（仅提供 API Server 指标）。
- API Server 以外的证书。

导入 Amazon EKS 集群

使用此提供商专用操作步骤，将现有 Amazon EKS 集群作为第三方集群导入。除非有文档说明的提供商专用功能支持相应生命周期管理，否则外部提供商仍负责提供商基础设施，以及由提供商管理的 Kubernetes 和节点生命周期。

目录

前提条件
准备环境
获取集群信息
获取导入令牌
导入集群
网络配置
后续步骤
初始化 Ingress 和存储
常见问题
导入后“添加节点”按钮被禁用。如何添加节点？
导入的集群的证书管理支持哪些证书？
此提供商专属导入路径的限制

前提条件

- 集群的 Kubernetes 版本和设置符合导入标准 Kubernetes 集群的版本兼容性要求。
- 镜像仓库必须支持 HTTPS，并提供由公共 CA 签发的有效 TLS 证书。

准备环境

为遵循 AWS EKS 安全实践，请在 AWS CloudShell 中执行以下操作步骤。

1. 确保可以连接到 AWS Management Console。
2. 搜索 `cloudshell`，然后打开 CloudShell↗。
3. 确认所选地域与目标集群所在地域一致；如有需要，请切换地域。
4. CloudShell 准备就绪后，清空终端并运行：

```
# List clusters in the current region and verify your permissions
aws eks list-clusters

# <region-code> is the region of the cluster, e.g., us-west-1
# <my-cluster> is the cluster name from the previous output
aws eks update-kubeconfig --region <region-code> --name <my-cluster>

# The kubeconfig file is saved to "${HOME}/.kube/config"
# Save its content to a file, then upload it to the platform for parsin
g
cat "${HOME}/.kube/config"
```

5. 现在环境已准备就绪。后续步骤（例如获取集群信息和导入集群）中，请在 CloudShell 内运行针对目标集群的命令。

获取集群信息

获取导入令牌

公有云集群的 KubeConfig 不能直接用于导入。

如需获取集群导入令牌，请参阅如何获取集群信息？。

导入集群

1. 在左侧导航中，前往集群管理 > 集群。
2. 点击导入集群。
3. 按如下方式配置参数。

参数	说明
镜像仓库	存储集群所需平台组件镜像的仓库。- 平台默认：部署 global 集群时配置的仓库。- 私有仓库：托管所需镜像的预置仓库。请提供私有仓库地址、端口、用户名和密码。- 公共仓库：公共互联网仓库。使用前，请按照更新公共仓库云凭据中的说明获取凭据。
集群信息	提示：上传 kubeconfig 文件，让平台自动解析。集群端点：目标集群对外公开的外部 API 服务器地址。**CA** 证书：集群的 CA 证书。身份验证：使用上一步创建的、具有集群管理员权限的令牌。

4. 点击检查连通性，验证网络连接并自动检测集群类型。检测到的类型会以徽标形式显示在表单右上角。
5. 连通性检查通过后，点击导入，然后确认。
   提示：

   - 对于处于导入中状态的集群，点击详情图标，可在执行进度对话框（`status.conditions`）中查看进度。
   - 导入成功后，集群列表会显示关键信息。集群状态为正常，并且可以执行集群操作。

网络配置

确保 global 集群与导入的集群之间网络连通。请参阅导入集群的网络配置。

后续步骤

初始化 **Ingress** 和存储

如果需要 Ingress 和存储功能，请参阅为 AWS EKS 初始化 Ingress和为 AWS EKS 初始化存储。

常见问题

导入后“添加节点”按钮被禁用。如何添加节点？

平台 UI 不支持添加节点。请通过集群提供商添加节点。

导入的集群的证书管理支持哪些证书？

1. **Kubernetes** 证书：只能查看 API 服务器证书。其他 Kubernetes 证书不可见，也不会自动轮换。
2. 平台组件证书：可在平台中查看，并支持自动轮换。

此提供商专属导入路径的限制

- 审计数据不可用。
- 不支持 ETCD、Scheduler 和 Controller Manager 指标；可使用部分 API 服务器图表。
- 除 Kubernetes API 服务器证书外，其他证书详细信息不可用。

导入 GKE 集群

使用此提供商特定的操作步骤，将现有 Google GKE 集群作为第三方集群导入。除非提供商特定功能有明确文档说明支持相关生命周期管理，否则外部提供商仍负责提供商基础设施，以及由提供商管理的 Kubernetes 和节点生命周期。

目录

前提条件
准备操作环境
获取集群信息
获取目标集群的 API Server 地址和 CA 证书
获取目标集群令牌
导入集群
网络配置
导入后的操作
Ingress 和存储初始化
常见问题
导入集群后，“添加节点”按钮显示为灰色时，如何添加节点？
导入的集群的证书管理功能支持哪些证书？

前提条件

- 集群上的 Kubernetes 版本和组件满足导入公有云集群的版本要求。
- 确保集群类型为标准集群，并且帐户具有维护控制平面的权限。目前不支持 Autopilot 集群。
- 镜像仓库必须支持 HTTPS 访问，并提供由公共证书颁发机构认证的有效 TLS 证书。

准备操作环境

为符合 GKE 安全标准，必须使用 Cloud Shell 执行以下步骤。

1. 确保与 Google 的网络连接正常。
2. 在 Kubernetes Engine 功能中访问 集群页面↗；找到要导入的集群，单击集群详细信息，然后选择 连接 按钮。
3. 在弹出对话框中，复制用于配置 kubectl 命令行访问权限的命令，然后单击 在 **Cloud Shell** 中运行 按钮。
4. 等待 Cloud Shell 准备就绪，清空命令行，粘贴上一步复制的内容，然后执行该命令。
5. 现在环境已准备就绪。后续在导入集群环境中执行的所有命令，例如 获取集群信息 和 导入集群 等操作步骤中的命令，都应在 Cloud Shell 中执行。

获取集群信息

获取目标集群的 **API Server** 地址和 **CA** 证书

1. 在 Kubernetes Engine 功能中访问 集群页面↗，然后单击进入目标集群的详细信息页面。
2. 可以在 外部端点 部分找到 API Server 地址。
3. 在 Cloud Shell 中使用以下任一方法获取 CA 证书：
   方法 **A**：从 kubeconfig 获取 CA 证书：

   ```
   gcloud container clusters get-credentials <cluster-name> --zone <zone>
   kubectl config view --raw -o jsonpath='{.clusters[0].cluster.certificat
   e-authority-data}' | base64 -d
   ```

   方法 **B**：直接从集群获取 CA 证书：

```
gcloud container clusters describe <cluster-name> --zone <zone> --forma
t='get(masterAuth.clusterCaCertificate)' | base64 -d
```

注意：将证书粘贴到导入表单之前，必须先对其进行 Base64 解码。

获取目标集群令牌

公有云集群的 KubeConfig 文件无法直接用于导入集群。

请参阅如何获取集群信息？以获取目标集群令牌。

导入集群

1. 在左侧导航栏中，点击 集群 > 集群。
2. 点击 管理集群 > 导入集群。
3. 根据以下说明配置相关参数。

参数	描述
镜像仓库	用于存储集群所需平台组件镜像的仓库。- 平台默认：全局部署期间配置的镜像仓库。- 私有仓库：存储平台所需组件的预构建仓库。请输入用于访问镜像仓库的 私有镜像仓库地址、端口、用户名和密码。- 公有仓库：使用互联网上的公有镜像仓库服务。使用前，请更新公有仓库云凭证以获取仓库身份验证权限。
集群信息	集群信息：包括目标集群令牌、目标集群的 API Server 地址和 CA 证书。集群地址：目标集群向平台公开 API Server 的访问地址，用于平台访问集群的 API Server。**CA** 证书：目标集群的 CA 证书。注意：手动输入时，需要输入经过 Base64 解码的证书。身份验证方式：目标集群的身份验证方式，需要使用上一步创建的、具有集群管理权限的 **token**（令牌）进行身份验证。

4. 点击 检查连通性，验证与目标集群的网络连通性，并自动识别集群类型。集群类型将以徽章形式显示在表单右上角。
5. 连通性检查通过后，点击 导入并确认。

TIP

- 在处于导入中状态的集群右侧，点击详情图标，在弹出的执行进度对话框中查看集群执行进度（status.conditions）。
- 集群成功导入后，可以在集群列表中查看集群关键信息，集群状态显示为正常，并可以执行集群相关操作。

网络配置

确保 global 集群与导入的集群之间的网络连通性。请参阅导入集群的网络配置。

导入后的操作

Ingress 和存储初始化

导入集群后，如果需要使用 Ingress 和存储相关功能，请参阅 Google GKE Ingress Controller 配置和 Google GKE 存储配置。

常见问题

导入集群后，“添加节点”按钮显示为灰色时，如何添加节点？

不支持通过平台界面添加节点。请联系集群提供商添加节点。

导入的集群的证书管理功能支持哪些证书？

1. **Kubernetes** 证书：所有导入的集群仅支持在平台证书管理界面中查看 APIServer 证书信息。不支持查看其他 Kubernetes 证书，也不支持自动轮换。
2. 平台组件证书：所有导入的集群均可在平台证书管理界面中查看平台组件证书信息，并支持自动轮换。

导入 Huawei Cloud CCE 集群（公有云）

使用此提供商专用操作步骤，将现有 Huawei Cloud CCE 集群作为第三方集群导入。外部提供商仍负责提供商基础设施，以及由提供商管理的 Kubernetes 和节点生命周期，除非有文档说明的提供商专用功能可以提供该生命周期管理。

目录

前提条件
获取镜像仓库地址
确定镜像仓库是否需要额外配置
获取集群信息
  获取导入集群令牌
导入集群
网络配置
后续操作
  Ingress（入站规则）和存储初始化
FAQ
  导入集群后，添加节点按钮显示为灰色。如何添加节点？
  证书管理功能支持导入集群的哪些证书？
  此提供商特定导入路径的限制

前提条件

- 集群上的 Kubernetes 版本和参数符合标准 Kubernetes 集群组件版本和参数要求。
- 确保集群类型为 Huawei Cloud CCE 集群，并且账号具有维护控制平面的权限。目前不支持 Turbo 集群。
- Huawei Cloud CCE 集群创建后默认无法访问外部网络资源。导入集群前，请确保待导入集群可以访问平台访问地址。

获取镜像仓库地址

- 若要使用 global 集群部署中的平台部署的镜像仓库，请在 **global** 集群的控制平面节点上执行以下命令以获取地址：

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.prefe
rPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- 若要使用外部镜像仓库，请手动设置 **REGISTRY** 变量。

```
REGISTRY=<external image registry address>  # Valid examples: registry.e
xample.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

确定镜像仓库是否需要额外配置

1. 执行以下命令，确定指定的镜像仓库是否支持 HTTPS 访问，以及是否使用受信任 CA 机构签发的证书：

```
REGISTRY=<image registry address obtained from the "Obtain Image Registry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Test passed: The image registry uses certificates issued by trusted CA authorities. You do not need to execute the content in the "Trust Insecure Image Registry" section.'
else
    echo 'Test failed: The image registry does not support HTTPS or the certificate is not trusted. See the "Trust Insecure Image Registry" section for configuration.'
fi
```

2. 如果测试失败，请参阅如何信任不安全的镜像仓库？。

获取集群信息

1. 确保与 Huawei Cloud 控制台的网络连通。
2. 访问 `Cloud Container Engine CCE` 功能的集群管理页面↗；找到要导入的集群，然后点击集群名称进入详情页面。
3. 如下图所示，按照导航找到下载 KubeConfig 文件按钮：`Cluster Information - Connection Information - kubectl - Configuration`，然后下载 KubeConfig 文件。

获取导入集群令牌

公有云集群的 KubeConfig 文件无法直接用于集群导入。

请参阅如何获取集群信息？以获取导入集群令牌。

导入集群

1. 在左侧导航栏中，点击 集群管理 > 集群。
2. 点击 导入集群。

3. 根据以下说明配置与 `Image Registry` 相关的参数。

参数	描述
镜像仓库	用于存储集群所需平台组件镜像的仓库。 - 平台默认：部署 global 集群时配置的镜像仓库。 - 私有仓库：用于存储平台所需组件的预构建仓库。请输入用于访问镜像仓库的私有镜像仓库地址、端口、用户名和密码。 - 公有仓库：使用位于公有网络中的镜像仓库服务。使用前，请更新公有镜像仓库云凭证以获取仓库认证权限。
集群信息	提示：上传 KubeConfig 文件，平台将自动解析并填充信息。 集群地址：导入集群对外暴露的 API Server 访问地址，用于平台访问导入集群的 API Server。 **CA** 证书：导入集群的 CA 证书。 认证方式：导入集群的认证方式，需要使用上一步创建的具有集群管理权限的令牌进行认证。

4. 点击 `Parse KubeConfig File` 按钮，并提交上一步下载的 KubeConfig 文件。平台将自动解析并填充与 `Cluster Information` 相关的参数。
5. 点击 检查连通性，检查与导入集群的网络连通性，并自动识别导入集群的类型。集群类型将以徽章形式显示在表单右上角。
6. 连通性检查通过后，点击 导入 并确认。
   提示：

   - 点击处于导入中状态的集群右侧的
     
     图标，可在弹出的执行进度对话框中查看集群的执行进度（status.conditions）。
   - 集群成功导入后，可以在集群列表中查看集群的关键信息。集群状态显示为正常，并且可以执行集群相关操作。

网络配置

为确保 global 集群与导入集群之间的网络连通性，请参阅导入集群网络配置。

后续操作

Ingress（入站规则）和存储初始化

导入集群后，如果需要使用 Ingress（入站规则）和存储相关功能，请参阅Huawei Cloud CCE 集群 Ingress 初始化配置和Huawei Cloud CCE 集群存储初始化配置。

FAQ

导入集群后，添加节点按钮显示为灰色。如何添加节点？

不支持通过平台界面添加节点。请联系集群提供商添加节点。

证书管理功能支持导入集群的哪些证书？

1. **Kubernetes** 证书：所有导入集群仅支持在平台证书管理界面查看 APIServer 证书信息。不支持查看其他 Kubernetes 证书，也不支持自动轮换。
2. 平台组件证书：所有导入集群均可在平台证书管理界面查看平台组件证书信息，并支持自动轮换。

此提供商特定导入路径的限制

- 不支持检索审计数据。
- 不支持 ETCD、Scheduler 和 Controller Manager 相关的监控信息。支持 APIServer 部分监控图表。
- 无法检索 Kubernetes APIServer 证书以外的集群证书相关信息。

导入 Azure AKS 集群

使用此提供商专用操作步骤，将现有 Azure AKS 集群作为第三方集群导入。除非有文档说明的提供商专用功能提供相应生命周期管理，否则外部提供商仍负责提供商基础设施，以及由提供商管理的 Kubernetes 和节点生命周期。

目录

前提条件
准备操作环境
获取集群信息
  获取导入令牌
导入集群
网络配置
导入后操作
  Ingress（入站规则）和存储初始化
常见问题
  如何配置 AKS 节点外部 IP 安全组规则
  如何访问 AKS 节点
  Azure ALB 使用内部负载均衡器
  Azure ALB 使用外部负载均衡器
  导入集群后添加节点按钮显示为灰色。如何添加节点？
  导入集群的证书管理功能支持哪些证书？
  此提供商特定导入路径的限制

前提条件

- 集群上的 Kubernetes 版本和参数必须符合标准 Kubernetes 集群组件版本和参数要求。

> **TIP**
>
> - 如果 AKS 节点无法访问 global 集群，请参阅如何配置 AKS 节点外部 IP 安全组规则。

- 镜像仓库必须支持 HTTPS 访问，并提供由公共认证机构验证的有效 TLS 证书。

准备操作环境

为符合 Azure AKS 安全标准，必须使用 Cloud Shell 执行以下步骤。

1. 确保与 Azure Console 的网络连接正常。
2. 打开Kubernetes Services 页面↗，找到要导入的集群，然后单击进入集群概览页面。
3. 单击 `Connect` 按钮，这将打开标题为 `Connect to <import cluster name>` 的浮动窗口。按照说明打开 Cloud Shell 并配置操作环境。

获取集群信息

获取导入令牌

公共云集群的 KubeConfig 文件无法直接用于集群导入。

请参阅如何获取集群信息？以获取导入集群令牌。

导入集群

1. 在左侧导航栏中，单击集群管理 > 集群。
2. 单击导入集群。
3. 根据以下说明配置相关参数。

参数	描述
镜像仓库	存储集群所需平台组件镜像的仓库。- 平台默认：部署 global 集群时配置的镜像仓库。- 私有仓库：预先构建的、用于存储平台所需组件镜像的仓库。输入用于访问镜像仓库的私有镜像仓库地址、端口、用户名和密码。- 公共仓库：使用互联网上的公共镜像仓库服务。使用前，请更新公共镜像仓库云凭据以获取仓库身份验证权限。
集群信息	提示：上传 KubeConfig 文件，平台将自动解析并填充信息。集群地址：导入集群公开的 API Server 访问地址，平台通过该地址访问导入集群的 API Server。**CA** 证书：导入集群的 CA 证书。身份验证方式：导入集群的身份验证方式，需要使用上一步创建的、具有集群管理权限的 **Token** 进行身份验证。

4. 单击检查连接，验证与导入集群的网络连接，并自动识别导入集群类型。集群类型将以徽章形式显示在表单右上角。
5. 连接检查通过后，单击导入并确认。

> **TIP**
>
> - 在处于导入中状态的集群右侧单击详情图标，可在弹出的执行进度对话框中查看集群的执行进度（status.conditions）。
> - 集群成功导入后，可以在集群列表中查看集群的关键信息。集群状态将显示为正常，并且可以执行与集群相关的操作。

网络配置

确保 global 集群与导入集群之间的网络连通。请参见导入集群的网络配置。

导入后操作

Ingress（入站规则）和存储初始化

导入集群后，如果需要使用 Ingress（入站规则）和存储相关功能，请参见 Azure AKS 集群 Ingress 初始化配置 和 Azure AKS 集群存储初始化配置。

常见问题

如何配置 AKS 节点外部 IP 安全组规则

节点默认仅具有内部 IP。外部 IP 配置在前端负载均衡器（LB）上，默认用于出站流量。此 LB 由 AKS 主体控制。直接手动修改此配置可能会导致问题。您可以通过**Kubernetes** > 属性 > 基础结构资源组 > 网络安全组 > 添加出站/入站全部规则允许流量通过。

如何访问 AKS 节点

若要查看 Kubelet、CNI 和内核等系统组件的日志，您需要先通过 SSH 登录节点。建议使用 `kubectl-node-shell` 插件，而不是为每个节点分配公共 IP 地址。

选项 **1**：使用 **kubectl node-shell**

请参见 kubectl-node-shell 项目↗。

选项 **2**：使用 **debug**

请参见提供商关于连接到 AKS 节点↗的指导。

> **NOTE**
>
> 此示例要求使用 1.25 或更高版本的 kubectl，其中包含 GA `kubectl debug` 命令。

```
kubectl debug node/aks-newadd-41368356-vmss000002 -it --image=mcr.microsoft.com/dotnet/runtime-deps:6.0
chroot /host
```

Azure ALB 使用内部负载均衡器

有关内部负载均衡器的行为，请参见提供商指导↗。

```
apiVersion: v1
kind: Service
metadata:
  name: internal-app
  namespace: cpaas-system
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  ports:
  - name: http-port
    port: 80
    protocol: TCP
  - name: https-port
    port: 443
    protocol: TCP
  selector:
    service.cpaas.io/name: deployment-aks-alb
    service_name: alb2-aks-alb
```

Azure ALB 使用外部负载均衡器

部署高可用 ALB，并将访问地址配置为外部 LB。

```
apiVersion: v1
kind: Service
metadata:
  name: azure-alb
  namespace: cpaas-system
spec:
  type: LoadBalancer
  ports:
  - name: http-port
    port: 80
    protocol: TCP
  - name: https-port
    port: 443
    protocol: TCP
  - name: prom-port
    port: 11780
    protocol: TCP
  - name: prom2-port
    port: 11781
    protocol: TCP
  - name: prom3-port
    port: 15012
    protocol: TCP
  selector:
    service_name: alb2-cpaas-system
```

如果已提前部署，可以使用以下命令进行修改。

```
kubectl edit helmrequest -n cpaas-system uat-cluster-aks-alb
```

导入集群后添加节点按钮显示为灰色。如何添加节点？

不支持通过平台界面添加节点。请联系集群提供商添加节点。

导入集群的证书管理功能支持哪些证书？

1. **Kubernetes** 证书：所有导入集群仅支持在平台证书管理界面查看 APIServer 证书信息。不支持查看其他 Kubernetes 证书，也不支持自动轮换。

2. 平台组件证书：所有导入集群均可在平台证书管理界面查看平台组件证书信息，并支持自动轮换。

此提供商特定导入路径的限制

- 不支持获取审计数据。
- 不支持 ETCD、Scheduler 和 Controller Manager 相关的监控信息。支持 APIServer 部分监控图表。
- 无法获取 Kubernetes APIServer 证书以外的集群证书相关信息。

导入 Alibaba Cloud ACK 集群

使用此提供商专用操作步骤，将现有的 Alibaba Cloud ACK 托管集群或专有集群作为第三方集群导入。除非有记录在案的提供商专用功能提供相应的生命周期管理，否则外部提供商仍负责提供商基础设施，以及由提供商管理的 Kubernetes 和节点生命周期。

TIP

有关提供商集群类型的详细信息，请参阅提供商指南↗。

目录

前提条件
获取镜像仓库地址
确定镜像仓库是否需要其他配置
获取 KubeConfig
导入集群
网络配置
常见问题
如何处理 Alibaba Cloud 监控与平台监控组件之间的端口冲突？
如何使用 Alibaba Cloud 集群的公网访问？
导入集群后，添加节点按钮显示为灰色。如何添加节点？
导入集群的证书管理功能支持哪些证书？
此提供商特定导入路径的限制

前提条件

- 集群上的 Kubernetes 版本和参数满足导入标准 Kubernetes 集群的组件版本和参数要求。

获取镜像仓库地址

- 若要使用 global 集群部署中的平台部署的镜像仓库，请在 global 集群的控制平面节点上执行以下命令以获取地址：

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.preferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- 若要使用外部镜像仓库，请手动设置 **REGISTRY** 变量。

```
REGISTRY=<external image registry address>  # Valid examples: registry.example.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

确定镜像仓库是否需要其他配置

1. 执行以下命令，确定指定的镜像仓库是否支持 HTTPS 访问，以及是否使用由受信任 CA 颁发机构签发的证书：

```
REGISTRY=<image registry address obtained from the "Get Image Registry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Test passed: The image registry uses certificates issued by trusted CA authorities. You do not need to execute the content in the "Trust Insecure Image Registry" section.'
else
    echo 'Test failed: The image registry does not support HTTPS or the certificate is not trusted. See the "Trust Insecure Image Registry" section for configuration.'
fi
```

2. 如果测试失败，请参阅如何信任不安全的镜像仓库？。

获取 KubeConfig

1. 登录 Alibaba Cloud 容器服务管理平台。
2. 在控制台左侧导航栏中，单击集群。
3. 在集群列表页面中，单击目标集群名称，或单击目标集群右侧操作列中的详情。
4. 在集群信息页面中，单击连接信息选项卡，然后单击生成临时 **KubeConfig**。
5. 在临时 **KubeConfig** 对话框中，设置临时凭证的有效期以及访问集群的方式（包括公网访问和内网访问）。
6. 单击生成临时 **KubeConfig**，然后单击复制，将内容复制并保存到本地计算机上的 **KubeConfig** 文件中。
7. 集群成功导入后，您可以撤销临时凭证。

导入集群

1. 在左侧导航栏中，单击集群管理 > 集群。
2. 单击导入集群。
3. 根据以下说明配置相关参数。

参数	描述
镜像仓库	用于存储集群所需平台组件镜像的仓库。- 平台默认：global 集群部署期间配置的镜像仓库。- 私有仓库：存储平台所需组件镜像的预构建仓库。您需要输入用于访问镜像仓库的私有镜像仓库地址、端口、用户名和密码。- 公共仓库：使用互联网中的公共镜像仓库服务。使用前，请更新公共仓库云凭证以获取仓库身份验证权限。
集群信息	提示：可以手动填写，也可以通过上传 KubeConfig 文件，由平台自动解析并填写。解析 **KubeConfig** 文件：上传获取的 KubeConfig 文件后，平台将自动解析并填写集群信息。您可以修改自动填写的信息。集群地址：集群对外暴露的 API Server 访问地址，平台通过该地址访问集群的 API Server。**CA** 证书：集群的 CA 证书。注意：手动输入时，需要输入经过 Base64 解码的证书。身份验证方式：访问集群的身份验证方式。您需要使用具有集群管理权限的 **token** 或证书身份验证（客户端证书和密钥） 进行身份验证。

4. 单击检查连通性，检查与待导入集群的网络连通性，并自动识别待导入集群的类型。集群类型将以徽章形式显示在表单右上角。
5. 连通性检查通过后，单击导入并确认。

> **TIP**
>
> - 单击处于导入中状态的集群右侧的详情图标，在弹出的执行进度对话框中查看集群的执行进度（status.conditions）。
> - 集群成功导入后，您可以在集群列表中查看集群的关键信息。集群状态显示为正常，您可以执行与集群相关的操作。

网络配置

确保 global 集群与待导入集群之间的网络连通性。请参阅导入集群的网络配置。

常见问题

如何处理 **Alibaba Cloud** 监控与平台监控组件之间的端口冲突？

当 Alibaba Cloud 内置监控与平台监控组件共存时，会发生端口冲突。建议卸载 Alibaba Cloud 监控，仅保留平台监控。

如何使用 **Alibaba Cloud** 集群的公网访问？

如果使用 Alibaba Cloud 集群的公网访问，可以在 Alibaba Cloud 上绑定公网 IP。

导入集群后，添加节点按钮显示为灰色。如何添加节点？

Alibaba Cloud ACK 托管集群和 **ACK** 专有集群均不支持通过平台界面添加节点。请在提供商后端添加节点，或联系集群提供商。

导入集群的证书管理功能支持哪些证书？

1. **Kubernetes** 证书：所有导入的集群仅支持在平台证书管理界面查看 APIServer 证书信息。不支持查看其他 Kubernetes 证书，也不支持自动轮换。
2. 平台组件证书：所有导入的集群均可在平台证书管理界面查看平台组件证书信息，并支持自动轮换。

此提供商特定导入路径的限制

- **Alibaba Cloud ACK** 托管集群不支持获取审计数据。
- **Alibaba Cloud ACK** 托管集群不支持获取 ETCD、Scheduler、Controller Manager 相关的监控信息，但支持部分 APIServer 监控图表。
- **Alibaba Cloud ACK** 托管集群和 **ACK** 专有集群均不支持获取集群证书相关信息，但 Kubernetes APIServer 证书除外。

导入 Tencent Cloud TKE 集群

使用此提供商专用操作步骤，将现有的 Tencent Cloud TKE 专用集群或托管集群作为第三方集群导入。除非有记录在案的提供商专用功能支持相应的生命周期管理，否则外部提供商仍负责提供商基础设施以及提供商管理的 Kubernetes 和节点生命周期。

TIP

有关提供商集群类型的详细信息，请参阅提供商指南↗。

目录

前提条件
获取镜像仓库地址
确定镜像仓库是否需要额外配置
获取 KubeConfig
导入集群
网络配置
常见问题
导入集群后，“添加节点”按钮置灰。如何添加节点？
导入集群的证书管理功能支持哪些证书？
此提供商特定导入路径的限制

前提条件

- 集群上的 Kubernetes 版本和参数符合导入标准 Kubernetes 集群的组件版本和参数要求。
- 镜像仓库必须支持 HTTPS 访问，并提供由公共证书颁发机构签发的有效 TLS 证书。

获取镜像仓库地址

- 要使用在 global 集群部署期间配置的平台部署镜像仓库，请在 global 集群的控制平面节点上执行以下命令以获取地址：

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.prefe
rPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- 要使用外部镜像仓库，请手动设置 **REGISTRY** 变量。

```
REGISTRY=<external image registry address>  # Valid examples: registry.
example.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

确定镜像仓库是否需要额外配置

1. 执行以下命令，确定指定的镜像仓库是否支持 HTTPS 访问，以及是否使用受信任 CA 签发的证书：

```
REGISTRY=<image registry address obtained from the "Obtain Image Registry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Verification passed: The image registry uses a certificate issued by a trusted CA. It is not necessary to execute the content in the "Trust Unsafe Image Registry" section.'
else
    echo 'Verification failed: The image registry does not support HTTPS or the certificate is not trusted. See the "Trust Unsafe Image Registry" section for configuration.'
fi
```

2. 如果验证失败，请参阅如何信任不安全的镜像仓库？。

获取 KubeConfig

1. 登录 Tencent Cloud 容器服务管理平台。
2. 在集群详情 > 基本信息中，查看集群 **APIServer** 信息。
3. 根据实际客户网络选择公网访问或内网访问，然后下载 **Kubeconfig** 并将其保存到本地计算机。

导入集群

1. 在左侧导航栏中，单击集群管理 > 集群。
2. 单击导入集群。
3. 按照以下说明配置相关参数。

参数	描述
镜像仓库	用于存储集群所需平台组件镜像的仓库。- 平台默认：在 global 集群部署期间配置的镜像仓库。- 私有仓库：用于存储平台所需组件镜像的预构建仓库。请输入用于访问镜像仓库的私有镜像仓库地址、端口、用户名和密码。- 公共仓

参数	描述
	库：使用位于公网的镜像仓库服务。使用前，请更新公共仓库云凭证以获取仓库身份验证权限。
集群信息	提示：可以手动填写，也可以通过 KubeConfig 文件上传，由平台自动解析并填写。解析 **KubeConfig** 文件：上传获取的 KubeConfig 文件后，平台将自动解析并填写集群信息，您可以修改自动填写的信息。集群地址：集群对外暴露的 API Server 访问地址，平台使用此地址访问集群的 API Server。**CA** 证书：集群的 CA 证书。注意：手动输入时，需要输入经过 Base64 解码的证书。身份验证方式：访问集群的身份验证方式，需要使用具有集群管理权限的 **token**（Token）或证书身份验证（客户端证书和密钥）。

4. 单击检查连通性，验证与待导入集群的网络连通性，并自动识别待导入集群的类型。集群类型将以徽章形式显示在表单右上角。
5. 连通性检查通过后，单击导入并确认。
   提示：

   - 单击处于导入中状态的集群右侧的详情图标，在弹出的执行进度对话框中查看集群的执行进度（status.conditions）。
   - 集群成功导入后，您可以在集群列表中查看集群的关键信息。集群状态显示为正常，并且可以执行与集群相关的操作。

网络配置

确保 global 集群与待导入集群之间的网络连通性。请参阅导入集群的网络配置。

常见问题

导入集群后，“添加节点”按钮置灰。如何添加节点？

TKE Dedicated 集群和 **TKE Managed** 集群均不支持通过平台界面添加节点。请在提供商后端添加节点，或联系集群提供商。

导入集群的证书管理功能支持哪些证书？

1. **Kubernetes** 证书：所有导入的集群仅支持在平台的证书管理界面查看 APIServer 证书信息，不支持查看其他 Kubernetes 证书，也不支持自动轮换。
2. 平台组件证书：所有导入的集群均可在平台的证书管理界面查看平台组件证书信息，并支持自动轮换。

此提供商特定导入路径的限制

- **TKE Managed** 集群不支持获取审计数据。
- **TKE Managed** 集群不支持 ETCD、Scheduler、Controller Manager 相关监控信息，但支持部分 APIServer 监控图表。
- **TKE Managed** 集群和 **TKE Dedicated** 集群均不支持获取集群证书相关信息，但 Kubernetes APIServer 证书除外。

注册集群

当目标第三方集群需要通过反向代理隧道主动连接平台时，请使用注册方式。

目录

先决条件
重要说明
注册集群
查看注册命令
常见问题
当已连接集群的运行时组件为 Containerd 时，如何解决分布式存储部署失败问题？

先决条件

- 根据目标环境，目标集群中的 Kubernetes 版本和组件参数必须满足版本和参数要求。
- 镜像仓库必须支持 HTTPS 访问，并提供由公共证书颁发机构认证的有效 TLS 证书。如果无法满足此要求，请参阅如何信任不安全的镜像仓库？
  注意：平台在公网提供的 公共仓库 已满足 HTTPS 访问要求。您只需确认 平台默认仓库 和 私有仓库 是否支持 HTTPS 访问。
- 如果待连接集群的运行时组件为 Containerd，则需在连接集群前修改 Containerd 配置，以确保分布式存储成功部署。

重要说明

网络接口流量监控默认识别名称与 `eth\.\|en\.\|wl\.*\|ww\.*` 匹配的网络接口。如果目标集群使用其他命名规则，请参阅从自定义命名的网卡采集网络数据，并在注册后更新相应资源。

注册集群

1. 在左侧导航栏中，单击集群 > 集群。
2. 单击托管集群 > 注册集群。
3. 按照以下说明配置用于存储注册集群所需平台组件镜像的镜像仓库参数。

参数	描述
平台默认仓库	部署全局组件时使用的镜像仓库。
私有仓库	您预先设置的外部镜像仓库。您需要输入用于访问镜像仓库的私有镜像仓库地址、端口、用户名和密码。
公共仓库	通过平台提供的公共镜像仓库拉取所需镜像。请确保目标集群可以访问公网。使用前，请参阅更新公共网络镜像仓库云凭证以获取仓库认证权限。

4. 单击创建，在注册命令页面获取注册命令，并在待注册集群中运行该命令。
   注意：注册命令的有效期为 24 小时。命令过期后，请获取新命令。

查看注册命令

在集群列表中找到等待注册的集群，然后单击查看注册命令。请在注册命令过期前运行该命令。

常见问题

当已连接集群的运行时组件为 **Containerd** 时，如何解决分布式存储部署失败问题？

当已连接集群的运行时组件为 Containerd 时，分布式存储部署将失败。要解决此问题，您需要手动修改集群所有节点上的 Containerd 配置信息，然后重启 Containerd。

注意：如果您在部署分布式存储前按照以下步骤修改 Containerd 配置，则无需执行第四步。

1. 登录集群节点并编辑 `/etc/systemd/system/containerd.service` 文件，将 `LimitNOFILE` 参数值修改为 `1048576`。
2. 执行命令 `systemctl daemon-reload` 重新加载配置。
3. 执行命令 `systemctl restart containerd` 重启 Containerd。
4. 在集群控制平面节点上执行命令 `kubectl delete pod --all -n rook-ceph`，重启 rook-ceph 命名空间中的所有 Pod，使配置生效。

公有云集群初始化

网络初始化

AWS EKS 集群网络初始化配置

目录

支持概览
前提条件
配置步骤
部署 AWS Load Balancer Controller
创建 Ingress 和 LoadBalancer 服务
相关操作
测试 AWS CLI 和 eksctl 安装
获取 ACCOUNT_ID
Kubeconfig 配置文件
为子网添加标签
创建证书

支持概览

功能	支持状态	要求
LoadBalancer 服务	支持	可选部署 AWS Load Balancer Controller。如果没有此控制器，LoadBalancer 功能将受到限制。

功能	支持状态	要求
Ingress	支持	可选部署 AWS Load Balancer Controller。可选启用 **Ingress Class** 功能（启用后，您可以在通过表单界面创建 ingress 时手动选择 Ingress Class）。

前提条件

- 准备两个带有 **kubernetes.io/role/elb** 标签的子网。对于共享子网，请添加 **kubernetes.io/cluster/<cluster-name>: shared** 标签。请参阅为子网添加标签。
- 如果您已创建 EKS 集群，请导入 Amazon EKS 集群。
- 在部署 AWS Load Balancer Controller 之前，请确保 kubectl、Helm、AWS CLI 和 eksctl 工具均可用。
  注意：安装工具后，请使用创建集群的用户通过 AWS CLI 配置登录信息，并测试 AWS CLI 和 eksctl 工具是否已正确安装。
- 提前获取 ACCOUNT_ID、**REGION** 和 **CLUSTER_NAME**，并将 `<ACCOUNT_ID>`、`<REGION>` 和 `<CLUSTER_NAME>` 替换为实际值。
  注意：**ACCOUNT_ID** 是创建集群的用户的账户 ID，**REGION** 是集群所在地域，**CLUSTER_NAME** 是集群名称。
- 更新并验证 Kubeconfig 配置文件。

配置步骤

部署 **AWS Load Balancer Controller**

注意：有关部署 AWS Load Balancer Controller 的详细信息，请参阅提供商指南↗。

配置 **OIDC Provider**

Kubernetes 集群使用 OpenID Connect (OIDC) 进行身份管理，并与 OIDC issuer URL 关联。要在集群中启用 AWS Identity 并允许 Service Account 使用 IAM 角色，请创建与集群 OIDC issuer URL 关联的 IAM OIDC Provider。

在 eksctl 中执行以下命令以配置 OIDC Provider：

```
eksctl utils associate-iam-oidc-provider --region=<REGION> --cluster=<CLU
STER_NAME> --approve
```

配置 **Service Account**

执行以下命令以创建 IAM policy，并创建名为 `aws-load-balancer-controller` 的 Service Account，将其与 IAM role 关联：

```
curl -o aws-load-balancer-controller-iam-policy.json https://raw.githubus
ercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.7/docs/in
stall/iam_policy.json
aws iam create-policy \
    --policy-name <CLUSTER_NAME>-AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://aws-load-balancer-controller-iam-policy.json

eksctl create iamserviceaccount \
  --cluster=<CLUSTER_NAME> \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::<ACCOUNT_ID>:policy/<CLUSTER_NAME>-AWS
LoadBalancerControllerIAMPolicy \
  --approve
```

将 **AWS Load Balancer Controller** 部署到集群

在 eksctl 中执行以下命令，将 AWS Load Balancer Controller 部署到集群：

1. 添加 eks-charts repository：

   ```
   helm repo add eks https://aws.github.io/eks-charts
   ```

2. 更新本地 repository：

   ```
   helm repo update eks
   ```

3. 将 AWS Load Balancer Controller Helm Chart 部署到集群：

注意：`aws-load-balancer-controller` 是在配置 Service Account中创建的 **Service Account**。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-control
ler \
  -n kube-system \
  --version=v2.4.7 \
  --set ingressClassConfig.default=true \
  --set clusterName=<CLUSTER_NAME> \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

创建 Ingress 和 LoadBalancer 服务

您可以同时创建 ingress 和 LoadBalancer 服务，也可以根据需要选择其中一种。

创建 **Ingress**

1. 在容器平台中，点击左侧导航栏中的网络 > **Ingress**。
2. 点击创建 **Ingress**，并在 **Ingress Class** 中选择 **EKS Ingress Class**。
3. 选择协议。默认为 **HTTP**。如需使用 **HTTPS**，请先创建证书，然后选择该证书。
4. 切换到 **YAML**，并添加以下 annotations。详情请参阅 annotation 参考↗：

```
alb.ingress.kubernetes.io/scheme: internet-facing ## Specify public acc
ess
alb.ingress.kubernetes.io/target-type: ip ## Route traffic directly to
pods
```

5. 点击创建。

创建 **LoadBalancer** 服务

1. 在容器平台中，点击左侧导航栏中的网络 > 服务。
2. 点击创建服务，并在外部访问中选择 **LoadBalancer**。
3. 展开 **annotations**，根据需要填写 LoadBalancer 服务 annotations。
4. 点击创建。

相关操作

测试 AWS CLI 和 eksctl 安装

- 执行以下命令。如果返回集群列表，则表示 AWS CLI 已正确安装：

```
aws eks list-clusters
```

- 执行以下命令。如果返回集群列表，则表示 eksctl 已正确安装：

```
eksctl get clusters
```

获取 ACCOUNT_ID

执行 `aws sts get-caller-identity` 获取 **ACCOUNT_ID**。响应中的 `651168850570` 即为 **ACCOUNT_ID**：

```
{
  "ARN": "arn:aws:iam::651168850570:user/jwshi"
}
```

Kubeconfig 配置文件

1. 执行以下命令，更新指定地域的 Kubeconfig 文件：

```
aws eks --region <REGION>  update-kubeconfig --name <CLUSTER_NAME>
```

2. 执行以下命令，验证 Kubeconfig 文件。如果正常返回信息，则表示配置正确：

```
kubectl get svc -n cpaas-system
```

为子网添加标签

1. 执行以下命令，获取集群子网：

```
eksctl get cluster --name <CLUSTER_NAME>
```

2. 执行以下命令，获取子网详情：

```
aws ec2 describe-subnets
```

3. 执行以下命令为子网添加标签。将 `<subnet-id>` 替换为实际值。详情请参阅子网自动发现 ↗：

   - 为子网添加 `kubernetes.io/role/elb` 标签：

   ```
   aws ec2 create-tags --resources <subnet-id> --tags Key=kubernetes.io/role/elb,Value="1"
   ```

   - 为共享子网添加 `kubernetes.io/cluster/<CLUSTER_NAME>: shared` 标签：

   ```
   aws ec2 create-tags --resources <subnet-id> --tags Key=kubernetes.io/cluster/<CLUSTER_NAME>,Value="shared"
   ```

创建证书

使用 HTTPS 协议时，请提前将 HTTPS 证书凭证以 Secret（TLS 类型）的形式保存。

1. 在容器平台中，点击左侧导航栏中的配置 > 密钥。
2. 点击创建密钥。
3. 选择 **TLS** 类型，并根据需要导入或填写证书和私钥。
4. 点击创建。

AWS EKS 补充信息

目录

术语

重要说明

EKS 使用 aws-lb 为容器网络负载均衡器提供外部访问

Service Annotation 配置说明

获取访问地址的方法

术语

缩写	全称	描述
eks-clb	Classic Load Balancer	AWS 默认的负载均衡器。在某些情况下存在问题，不建议使用。
eks-nlb	Network Load Balancer	AWS Layer 4 负载均衡器，在 TCP/UDP 层执行负载均衡，适用于需要更高层级网络控制的场景。
eks-alb	Application Load Balancer	AWS Layer 7 负载均衡器。与 eks-nlb 相比，eks-alb 可以解析 HTTP/HTTPS 协议并更智能地分发请求，适用于 Web 应用程序。

缩写	全称	描述
aws-lb	AWS Load Balancer	安装在 Kubernetes 上的负载均衡器，可以根据 Kubernetes 中的 LoadBalancer Services 和 Ingress 自动创建 eks-nlb 和 eks-alb，以满足应用程序的负载均衡需求。
Platform Load Balancer	-	平台专有的 Layer 7 负载均衡器。
Service Annotations	-	以键值对形式附加到对象的元数据。这些附加信息本身可以被识别和利用，以增强并简化 Kubernetes 资源各个方面的管理。Annotations 可以是没有特定功能的说明性文本，也可以指定云提供商的配置或行为，或者指定配置参数和工具。功能非常强大。

重要说明

创建负载均衡器时，建议手动配置 Service Annotations，以确保平台负载均衡器正确使用 aws-lb。如果未正确配置适用的 Service Annotations，平台将默认使用 eks-clb，而 eks-clb 存在与 UDP 相关的问题，可能导致异常情况。

EKS 使用 aws-lb 为容器网络负载均衡器提供外部访问

Service Annotation 配置说明

1. 在对应集群中，使用 kubectl 执行以下命令，查找 kube-system 命名空间中名称包含 "aws-load" 的所有 Pod：

   ```
   kubectl get pod -n kube-system |grep aws-load
   ```

2. 创建负载均衡器；有关详细的创建步骤和参数，请参阅 AWS EKS Service Annotation 说明 中的负载均衡器创建部分。

- 如果上述命令未返回相关 Pod，则表示集群未安装 AWS Load Balancer Controller。无需配置 Service Annotations，直接创建负载均衡器即可。
- 如果上述命令返回相关 Pod，则表示集群已安装 AWS Load Balancer Controller。在对应集群中创建负载均衡器时，添加以下 Service Annotations。有关 Annotation 的详细信息，请参阅 AWS EKS Service Annotation 说明：
  - `service.beta.kubernetes.io/aws-load-balancer-type: external //Required`
  - `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip //Required`
  - `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing //Optional. Add this annotation if public network support is needed.`

获取访问地址的方法

- 创建容器网络类型负载均衡器时，填写的 Service Annotations 将设置到与平台负载均衡器对应的 LoadBalancer Service 上。
- 在公有云中，配置了适用 Service Annotations 的 LoadBalancer Service 将被公有云识别并分配地址。平台负载均衡器将读取此地址，并将其设置为自身的访问地址。

Huawei Cloud CCE 集群网络初始化配置

目录

支持概览
前提条件
配置步骤
创建 Ingress
创建 LoadBalancer Service
相关操作
创建证书

支持概览

功能	支持状态	要求
LoadBalancer Service	默认支持	无需额外部署。
Ingress	默认支持	可选择启用 **Ingress Class** 功能（启用后，通过表单界面创建 ingress 时可以手动选择 ingress class）。无需额外部署。

前提条件

如果您已创建 CCE 集群，请导入 CCE 集群（Public Cloud）。

配置步骤

您可以同时创建 ingress 和 LoadBalancer 服务，也可以根据需要选择其中一项。

创建 Ingress

创建 ingress 有两种方法。推荐使用方法 **1**：手动选择 **Ingress Class**。

注意：避免创建两个具有相同 **path** 的 ingress 资源。

（推荐）方法 **1**：手动选择 **Ingress Class**

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Ingress**。
2. 点击 **Create Ingress**，并在 **Ingress Class** 中选择 **CCE Ingress Class**。
3. 选择 **Protocol**。默认为 **HTTP**。对于 **HTTPS**，请先创建证书并选择该证书。
4. 切换到 **YAML**，并根据默认 Ingress Controller 类型添加以下注解。有关注解的详细信息，请参阅使用注解配置负载均衡器↗：
   注意：将以下注解中的 **values** 替换为实际环境值。

默认 **Ingress Controller** 类型	注解
Shared (Auto-create)	`kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"{random}","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandw` `kubernetes.io/elb.class: union`

Shared (Reuse)	`kubernetes.io/elb.class: union`
	`kubernetes.io/elb.id: <Load Balancer Instance ID>`
	`kubernetes.io/elb.port: '80'`

默认 **Ingress Controller** 类型	注解

Dedicated (Auto-create)	`kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"<E`
	`Name>","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandwidtl`
	`["<AZ A>","<AZ B>","<AZ C>"],"elb_virsubnet_ids":["<ELB Virtual Subne`
	`ID>"],"l7_flavor_name":"L7_flavor.elb.s1.small","l4_flavor_name":"L4_`
	`kubernetes.io/elb.class: performance`
	`kubernetes.io/elb.port: "80"`

Dedicated (Reuse)	`kubernetes.io/elb.class: performance`
	`kubernetes.io/elb.id: <Load Balancer Instance ID>`
	`kubernetes.io/elb.port: "80"`

5. 点击 **Create**。创建完成后，您可以通过 ELB 访问集群服务。

方法 **2**：使用默认 **Ingress Class**

1. 使用以下内容创建 IngressClass YAML 文件。有关详细信息，请参阅默认 Ingress Class ↗ ：

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
  name: cce
spec:
  controller: alauda/cce
```

2. 保存文件并将其应用到已导入的集群。将 `<filename.yaml>` 替换为实际的 YAML 文件名：

```
kubectl apply -f <filename.yaml>
```

3. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Ingress**。

4. 选择 **Protocol**。默认为 **HTTP**。对于 **HTTPS**，请先创建证书并选择该证书。
5. 点击 **Create**。创建完成后，您可以通过 ELB 访问集群服务。

创建 LoadBalancer Service

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Services**。
2. 点击 **Create Service**，并在 **External Access** 中选择 **LoadBalancer**。
3. 展开 **annotations**，并根据需要填写 LoadBalancer 服务注解。
4. 点击 **Create**。

相关操作

创建证书

使用 HTTPS 协议时，请提前将 HTTPS 证书凭据以 Secret（TLS 类型）形式保存。

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Configuration** > **Secrets**。
2. 点击 **Create Secret**。
3. 选择 **TLS** 类型，并根据需要导入或填写 **Certificate** 和 **Private Key**。
4. 点击 **Create**。

Azure AKS 集群网络初始化配置

目录

支持概览
前提条件
配置步骤
部署 Ingress Controller
创建 Ingress 和 LoadBalancer Services
相关操作
创建证书

支持概览

功能	支持状态	要求
LoadBalancer Service	默认支持	无需额外部署。
Ingress	支持	可选择部署 Ingress Controller。也可选择启用 **Ingress Class** 功能（启用后，可在通过表单界面创建 ingress 时手动选择 ingress class）。

前提条件

如果已创建 AKS 集群，请导入 Azure AKS 集群。

配置步骤

部署 Ingress Controller

AKS 使用容器网络模式，并利用 **Nginx Ingress Controller** 管理负载均衡器，同时通过 **LoadBalancer** 类型的 **Services** 为容器内部网络中的虚拟 IP 地址（VIP）提供外部访问地址。

1. 登录 Microsoft Azure 并访问已创建的 AKS 集群。
2. 在左侧导航栏中，点击 **Kubernetes Resources** > **Services and Ingresses**。
3. 点击 创建，从下拉菜单中选择 **Ingress (Preview)**，系统将提示并自动创建 Ingress Controller。
4. 点击 启用并等待完成。

创建 Ingress 和 LoadBalancer Services

您可以同时创建 Ingress 和 LoadBalancer Services，也可以根据需要选择创建其中一种。

创建 **Ingress**

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Ingress**。
2. 点击 创建 **Ingress**，并为 **Ingress Class** 选择 **webapprouting.kubernetes.azure.com**。
3. 选择 **Protocol**。默认值为 **HTTP**。如需使用 **HTTPS**，请先创建证书并选择该证书。
4. 点击 创建。

创建 **LoadBalancer Service**

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Services**。
2. 点击 创建 **Service**，并为 **External Access** 选择 **LoadBalancer**。
3. 展开 **annotations**，并根据需要填写 LoadBalancer Service 注解。

4. 点击 创建。

相关操作

创建证书

使用 HTTPS 协议时，请提前将 HTTPS 证书凭据保存为 Secret（TLS 类型）。

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Configuration** > **Secrets**。
2. 点击 **Create Secret**。
3. 选择 **TLS** 类型，并根据需要导入或填写 **Certificate** 和 **Private Key**。
4. 点击 **Create**。

Google GKE 集群网络初始化配置

目录

支持概述
先决条件
配置步骤
部署 Ingress Controller
创建 Ingress 和 LoadBalancer Services
相关操作
在 Google Cloud 中查看 Ingress 资源
创建证书

支持概述

功能	支持状态	要求
LoadBalancer Service	默认支持	无需额外部署。
Ingress	默认支持	可选择启用 **Ingress Class** 功能（启用后，可通过表单界面创建 Ingress 时手动选择 Ingress 类）。无需额外部署。

先决条件

如果已创建 GKE 集群，请导入 GKE 集群。

配置步骤

部署 Ingress Controller

无需手动部署。GKE 提供名为 GKE Ingress 的托管内置 Ingress Controller。此 Controller 会将 Ingress 资源映射到 Google Cloud Load Balancer，以处理 GKE 中的 HTTP(S) 工作负载，使配置更加简单且自动化。

创建 Ingress 和 LoadBalancer Services

您可以同时创建 Ingress 和 LoadBalancer Services，也可以根据需要选择其中一种。

创建 **Ingress**

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Ingress**。
2. 点击 **Create Ingress**，并为 **Ingress Class** 选择 **GKE Ingress Class**。
3. 选择 **Protocol**。默认值为 **HTTP**。若要使用 **HTTPS**，请先创建证书并选择该证书。
4. 点击 **Create**。等待约 5 分钟，GKE 平台会自动为 Ingress 分配一个公共 IP 地址。
   注意：不同的 Ingress 资源会分配到不同的公共 IP 地址。

创建 **LoadBalancer Service**

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Network** > **Services**。
2. 点击 **Create Service**，并为 **External Access** 选择 **LoadBalancer**。
3. 展开 **annotations**，并根据需要填写 LoadBalancer Service 注解。
4. 点击 **Create**。

相关操作

在 **Google Cloud** 中查看 **Ingress** 资源

1. 前往 **Google Cloud > Kubernetes Engine**，并点击左侧导航栏中的 **Services and Ingress**。
2. 点击 **INGRESS**。
3. 在列表中查看相应 Ingress 资源的信息。

创建证书

使用 HTTPS 协议时，请预先将 HTTPS 证书凭据保存为 Secret（TLS 类型）。

1. 在 **Container Platform** 中，点击左侧导航栏中的 **Configuration** > **Secrets**。
2. 点击 **Create Secret**。
3. 选择 **TLS** 类型，并根据需要导入或填写 **Certificate** 和 **Private Key**。
4. 点击 **Create**。

存储初始化

公有云存储初始化概述

导入受支持的公有云 Kubernetes 服务后，请按照存储初始化流程进行操作，然后再在工作负载中使用持久化存储功能。外部提供商仍负责提供商基础设施和由提供商管理的存储后端；ACP 负责配置和使用导入集群公开的存储类。

下表总结了各提供商特定初始化路径所记录的默认存储类和操作。

目录

存储类支持

AWS EKS 集群

Huawei Cloud CCE 集群

Azure AKS 集群

Google GKE 集群

存储类支持

AWS EKS 集群

存储类型	默认存储类	使用 **RWO** 访问模式创建 **PVC**	使用 **RWX** 访问模式创建 **PVC**	**PVC** 扩容	**P**
文件存储	efs-sc	支持	支持	不支持	不
块存储	ebs-sc	支持	不支持	支持	不

Huawei Cloud CCE 集群

存储类型	默认存储类	使用 **RWO** 访问模式创建 **PVC**	使用 **RWX** 访问模式创建 **PVC**	**PVC** 扩容	**P**
文件存储	csi-nas	不支持	支持	支持	不
块存储	csi-disk	支持	不支持	支持	不

Azure AKS 集群

存储类型	默认存储类	使用 **RWO** 访问模式创建 **PVC**	使用 **RWX** 访问模式创建 **PVC**	**PVC** 扩容	**P**
文件存储	azurefile	支持	支持	支持	不
块存储	default	支持	不支持	支持	不

Google GKE 集群

存储类型	默认存储类	使用 **RWO** 访问模式创建 **PVC**	使用 **RWX** 访问模式创建 **PVC**	**PVC** 扩容	**P**
文件存储	standard-rwx	支持	支持	支持	不

存储类型	默认存储类	使用 **RWO** 访问模式创建 **PVC**	使用 **RWX** 访问模式创建 **PVC**	**PVC** 扩容	**P**
块存储	standard-rwo	支持	不支持	支持	不

AWS EKS 集群存储初始化配置

平台与 AWS EKS 集成及存储初始化配置。

目录

约束和限制
先决条件
配置步骤
创建存储类
修改存储类项目分配
相关操作
配置可用存储类参数

约束和限制

- 默认的 efs-sc 文件存储类在挂载后可能不支持修改权限，这可能导致 PostgreSQL 和 Jenkins 等某些应用无法正常运行。
- A1 系列实例不受 AL2023 AMI 支持，因此 EBS 块存储插件（Amazon EBS CSI Driver）无法正常部署。EBS CSI driver 已正式提供多架构/ARM 支持，因此限制在于 AMI/实例支持，而非驱动程序本身。如果需要使用 EBS 块存储类，请避免使用以下实例类型，并考虑改用 Graviton2/3 实例：
  - a1.medium

- a1.large
- a1.xlarge
- a1.2xlarge
- a1.4xlarge

推荐替代方案：使用 m6g、c6g、r6g、t4g 等 Graviton2/3 实例系列，它们性能更佳，并且完全支持 EBS CSI driver。

先决条件

- 确保已安装 kubectl↗ 和 AWS CLI 工具。
- 如果已创建 EKS 集群，请导入 Amazon EKS 集群；如果尚未创建，请创建 AWS EKS 集群。
- 在 EKS 集群中部署 EFS 文件存储插件 **Amazon EFS CSI Driver** 和 EBS 块存储插件 **Amazon EBS CSI Driver**。
  注意：如果使用 EFS 文件存储，请在 EKS 地域中创建文件存储，并记录 **File System** 中的 **File System ID**。

配置步骤

创建存储类

1. 前往平台管理，然后在左侧导航中点击存储管理 > 存储类。
2. 点击创建存储类旁边的下拉菜单 > 从 **YAML** 创建。
3. 根据需要，将以下内容添加到 YAML 文件中以创建默认存储类。文件存储↗的默认存储类名称为 **efs-sc**，块存储↗的默认存储类名称为 **ebs-sc**。

   - EFS 文件存储
     注意：将 `<File System ID>` 替换为实际的 **File System ID**，例如 `fileSystemId: fs-05aef9e1edd309f2b`。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: <File System ID>
  directoryPerms: "755"
```

- EBS 块存储

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

4. 点击创建。
   注意：如果默认存储类不满足要求，请按照上述步骤创建新的存储类，并根据需要修改参数。请参阅可用存储类参数。

修改存储类项目分配

1. 在左侧导航中，点击存储管理 > 存储类。
2. 点击名为 **efs-sc** 或 **ebs-sc** 的存储类旁边的三个点 > 更新项目。
3. 根据需要选择项目分配方式，然后点击更新，将存储类分配给项目。

相关操作

配置可用存储类参数

- EFS 文件存储可用参数

参数	可选值	默认值	可选	描述
az		""	true	用于跨账户 则使用与 az 行跨账户挂 则随机选择 跨账户挂载
basePath			true	用于创建动 径。如果未 统根目录下
directoryPerms			false	创建访问点 目录权限。
uid			true	创建访问点 POSIX 用户
gid			true	创建访问点 POSIX 组 II
gidRangeStart		50000	true	创建访问点 POSIX 组 II 已设置 uid/g
gidRangeEnd		7000000	true	POSIX 组 II 果已设置 ui 置。
subPathPattern			true	用于构造子 通过动态预 位于此路径 串和有限变 nfs-subdir-e chart 中的 " 量。可选参

参数	可选值	默认值	可选	描述
				.PVC.name 和 .PV.nam
ensureUniqueDirectory		true	true	在启用动态 true 时，会 subPathPat 式，以确保 向同一目录 定这是所需 为 false。
provisioningMode	efs-ap		false	EFS 卷类型 点。
fileSystemId			false	已创建访问

- EBS 块存储可用参数
  注意：有关不同卷类型的性能参数，请参阅 Amazon EBS Volume Types↗。

参数	可选值	默认值	描述
"allowAutoIOPSPerGBIncrease"	true, false	false	设为 "true" 时，如果 iopsPerGB * <volume size> 过低，无法满足 AWS 支持的 IOPS 范围，则 CSI 驱动程序会提高卷的 IOPS。这样可确保即使用户指定的 PVC 容量或 iopsPerGB 值过小，动态预置仍始终成功；但由于此类卷的 IOPS 高

参数	可选值	默认值	描述
			于 iopsPerGB 所需值，可能会产生额外费用。
"blockExpress"	true, false	false	通过将 io2 卷的 IOPS 限制提高到 256000 来创建 io2 Block Express 卷，但在不支持 io2 Block Express 的实例上无法挂载 IOPS 超过 64000 的卷。
"blockSize"			格式化底层文件系统时使用的块大小。仅适用于使用 ext2、ext3、ext4 或 xfs 文件系统类型的 Linux 节点。
"bytesPerINode"			格式化底层文件系统时使用的每个 inode 的字节数。仅适用于使用 ext2、ext3 或 ext4 文件系统类型的 Linux 节点。
"csi.storage.k8s.io/fstype"	xfs, ext2, ext3, ext4	ext4	创建卷时要格式化的文件系统类型。区分大小写。
"encrypted"	true, false	false	是否需要加密卷。

参数	可选值	默认值	描述
"inodeSize"			格式化底层文件系统时使用的 inode 大小。仅适用于使用 ext2、ext3、ext4 或 xfs 文件系统类型的 Linux 节点。inode 是文件系统中的数据结构，用于存储文件和目录的元数据。
"iops"			每秒 I/O 操作数，适用于 IO1、IO2 和 GP3 卷。
"iopsPerGB"			每 GiB 每秒的 I/O 操作数，适用于 IO1、IO2 和 GP3 卷。
"kmsKeyId"			用于加密卷的密钥完整 ARN。如果未指定，AWS 会使用卷所在地域的默认 KMS 密钥，并自动生成名为 /aws/ebs 的密钥。

参数	可选值	默认值	描述
"numberOfINodes"			格式化底层文件系统时指定的 inode 数量。仅适用于使用 ext2、ext3 或 ext4 文件系统类型的 Linux 节点。
"throughput"		125	吞吐量，单位为 MiB/s。仅在指定 gp3 卷类型时有效。如果为空，则默认为 125 MiB/s。请参阅 Amazon EBS Volume Types↗ 。
"type"	io1, io2, gp2, gp3, sc1, st1, standard, sbp1, sbg1	gp3	EBS 卷类型。

Huawei Cloud CCE 集群存储初始化配置

与 Huawei Cloud CCE 集成并配置存储初始化。

目录

约束和限制
前提条件
配置步骤
默认存储类说明
常见问题
PVC 创建失败
账户欠费

约束和限制

集群 PVC 数量有限制，账户存储容量也有配额。您可以通过提交支持工单申请提高配额。

前提条件

- 如果您已创建 CCE 集群，请导入 CCE 集群（Public Cloud）。

配置步骤

1. 前往平台管理，然后在左侧导航中点击存储管理 > 存储类。
2. 点击名为 **csi-nas** 或 **csi-disk** 的存储类旁边的三个点 > 更新项目。
   注意：导入 CCE 集群后，系统会生成默认存储类。块存储推荐使用 **csi-disk**，文件存储推荐使用 **csi-nas**。请参阅默认存储类说明。
3. 根据需要选择项目分配方式，然后点击更新，将 **csi-nas** 或 **csi-disk** 存储类分配给项目。

默认存储类说明

存储类名称	存储类类型	描述
（推荐）**csi-disk**	块存储	推荐使用。
（推荐）**csi-nas**	文件存储	推荐使用。仅在支持 csi-nas 服务的区域可用。
csi-disk-topology	块存储	当节点跨越 AZ 时自动配置云磁盘拓扑。
csi-local	本地存储	
csi-local-topology	本地存储	
csi-obs	对象存储	
csi-sfsturbo	超高速文件存储	超高速文件存储无法动态创建持久卷。

常见问题

PVC 创建失败

出现以下错误是因为已达到 PVC 数量限制。您可以通过提交工单申请提高限制：

```
message: "ShareLimitExceeded: Maximum number of shares allowed (10) excee
ded."
```

账户欠费

由于账户欠费，PVC 创建失败并出现以下错误。请先解决付款问题，然后重试：

```
message: "Your account is suspended and resources cannot be used"
```

Azure AKS 集群存储初始化配置

平台与 Azure AKS 集成以及存储初始化配置。

目录

约束和限制
前提条件
配置步骤
相关信息
默认存储类说明
可用的存储类参数

约束和限制

默认的 azurefile 文件存储类在挂载后可能不支持修改权限，这可能导致 PostgreSQL 和 Jenkins 等某些应用无法正常运行。

前提条件

- 如果您已创建 AKS 集群，请导入 Azure AKS 集群。

配置步骤

1. 前往平台管理，然后在左侧导航中点击存储管理 > 存储类。
2. 点击名为 **default** 或 **azurefile** 的存储类旁边的三个点 > 更新项目。
   注意：导入 AKS 集群后，将生成以下默认存储类。**default** 推荐用于块存储，**azurefile** 推荐用于文件存储。请参阅默认存储类说明。
3. 根据需要选择项目分配方式，然后点击更新，将 **default** 或 **azurefile** 存储类分配给项目。
   注意：如果默认存储类不符合要求，请按照上述步骤创建新的存储类，并根据需要修改参数。请参阅可用的存储类参数。

相关信息

默认存储类说明

存储类名称	存储类类型	描述
（推荐）**azurefile**	文件存储	使用 Azure 标准存储创建 Azure 文件共享。
（推荐）**default**	块存储	使用 Azure StandardSSD 存储创建托管磁盘。
azurefile-csi	文件存储	使用 Azure 标准存储创建 Azure 文件共享。
azurefile-csi-nfs	文件存储	使用 Azure 标准存储和 NFS 协议创建 Azure 文件共享。
azurefile-csi-premium	文件存储	使用 Azure 高级存储创建 Azure 文件共享。
azurefile-premium	文件存储	使用 Azure 高级存储创建 Azure 文件共享。
managed	块存储	使用 Azure StandardSSD 存储创建托管磁盘。
managed-csi	块存储	使用 Azure StandardSSD 本地冗余存储（LRS）创建托管磁盘。

存储类名称	存储类类型	描述
managed-csi-premium	块存储	使用 Azure 高级本地冗余存储（LRS）创建托管磁盘。
managed-premium	块存储	使用 Azure 高级存储创建托管磁盘。

可用的存储类参数

- 有关块存储的可选参数及其含义，请参阅在 Azure Kubernetes Service (AKS) 中使用 Azure 磁盘创建和使用卷↗。
- 有关文件存储的可选参数及其含义，请参阅在 Azure Kubernetes Service (AKS) 中使用 Azure Files 创建和使用卷↗。

Google GKE 集群存储初始化配置

平台与 Google GKE 集成以及存储初始化配置。

目录

约束和限制
前提条件
配置步骤
相关信息
默认存储类说明
可用的存储类参数
常见问题
文件存储类型存储类 PVC 创建失败
块存储类型存储类 PVC 无法正常绑定

约束和限制

- 默认文件存储类型 PVC 的最小容量为 1T。如果创建时将容量设置为小于 1T，系统会自动扩展到 1T。
- 默认文件存储存在容量限制。您可以通过支持工单申请扩容。

- 默认文件存储的创建和删除操作需要较长时间。如果资源长时间处于 `Creating` 状态，请继续监控该操作。

前提条件

- 创建集群时，在 Google Cloud Platform 的 **Cluster > Features** 页面中，在 **Other** 部分选中 **Enable Compute Engine Persistent Disk CSI Driver** 和 **Enable Filestore CSI Driver** 选项。
- 在 Google Cloud Platform 上启用 **Cloud Filestore API** 和 **Google Kubernetes Engine API**。请参阅使用 Filestore CSI 驱动程序访问 Filestore 实例↗。
- 在 Google Cloud Platform 上调整地域文件存储配额。请参阅资源配额和限制↗。
- 如果您已创建 GKE 集群，请导入 GKE 集群。

配置步骤

1. 前往平台管理，然后在左侧导航中单击存储管理 > 存储类。
2. 单击名为 **standard-rwx** 或 **standard-rwo** 的存储类旁边的三个点 > 更新项目。
   注意：导入 GKE 集群后，系统会生成默认存储类。文件存储推荐使用 **standard-rwx**，块存储推荐使用 **standard-rwo**。请参阅默认存储类说明。
3. 根据需要选择项目分配方式，然后单击更新，将 **standard-rwx** 或 **standard-rwo** 存储类分配给项目。
   注意：如果默认存储类无法满足要求，请按照上述步骤创建新的存储类，并根据需要修改参数。请参阅可用的存储类参数。

相关信息

默认存储类说明

存储类名称	存储类类型	描述
（推荐）**standard-rwx**	文件存储	使用 Basic HDD Filestore service tier ↗。
（推荐）**standard-rwo**	块存储	使用均衡型持久磁盘。
premium-rwx	文件存储	使用 Basic SSD Filestore service tier ↗。
premium-rwo	块存储	SSD 持久磁盘。
enterprise-rwx	文件存储	使用 Enterprise Filestore tier ↗。
enterprise-multishare-rwx	文件存储	使用 Enterprise Filestore tier ↗。请参阅 Filestore multishares for Google Kubernetes Engine ↗。

可用的存储类参数

- 有关块存储可选参数及其含义，请参阅存储选项 ↗。
- 有关文件存储可选参数及其含义，请参阅服务层级 ↗。

常见问题

文件存储类型存储类 PVC 创建失败

- 以下错误是由于项目中未启用 Cloud Filestore API 或缺少相应权限导致的。请参阅前提条件进行解决：

```
failed to provision volume with StorageClass "standard-rwx": rpc error:
code = PermissionDenied desc = googlespi: Error 403: Cloud Filestore AP
I has not been used in project alauda-proj-1234 before or it is disable
d.
...
resion: SERVICE_DISABLED
```

- 以下错误是由于超出存储配额导致的。请参阅前提条件进行解决：

```
failed to provision volume with StorageClass "standard-rwx": rpc error:
code = ResourceExhausted desc = googlespi: Error 429: Quora limit 'Stan
dardStorageGbPerRegion' has been exceeded. Limit 2048 in region asia-ea
st1.
 rateLimitExceeded
```

块存储类型存储类 **PVC** 无法正常绑定

以下错误是由于节点的 CSINode 缺少 pd.csi.storage.gke.io 驱动程序的配置导致的。您可以通过重启 **Compute Engine Persistent Disk CSI Driver** 解决此问题。

注意：更新此插件会导致集群不可用。更新过程大约需要 5-10 分钟。

```
Warning ProvisioningFailed 18m (x14 over 39m) pd.csi.storage.gke.io_gke-5
cb9bddae4d1430eb8ad-01f4-2084-vm_4b4e70bd-e2db-4779-9102-fee83a657ced fai
led to provision volume with StorageClass "standard": error generating ac
cessibility requirements: no available topology found
Normal ExternalProvisioning 4m35s (x143 over 39m) persistentvolume-contro
ller waiting for a volume to be created, either by external provisioner
"pd.csi.storage.gke.io" or manually created by system administrator
Normal Provisioning 3m19s (x18 over 39m) pd.csi.storage.gke.io_gke-5cb9bd
dae4d1430eb8ad-01f4-2084-vm_4b4e70bd-e2db-4779-9102-fee83a657ced External
provisioner is provisioning volume for claim "acp-gke-test/standard"
```

操作指南

导入集群的网络配置

获取导入集群信息

信任不安全的

从自定义命名的网卡采集网络数据

从自定义命名的网卡采集网络数据

为导入的标准 **Kubernetes** 集群配置审计收集

在导入的标准 Kubernetes 集群上启用 Kubernetes 审计日志记

导入集群的网络配置

导入前，所需的连接路径可让 `global` 集群访问目标集群。导入后，导入集群中的平台组件可能也需要访问 `global` 集群。如果需要告警和日志收集等组件进行双向连接，请添加平台 URL 注解。

目录

前提条件

添加平台 URL 注解

前提条件

准备域名、该域名指向的 **IP** 地址以及相应的有效证书，确保导入集群可以访问它们。

注意：

- 此域名不得与平台访问地址相同。
- 确保域名的端口（HTTPS 端口，与平台访问地址使用的端口相同）可以将流量转发到 global 集群的所有控制平面节点。

添加平台 **URL** 注解

将注解添加到导入集群资源中。

1. 在左侧导航栏中，单击集群管理 > 集群。
2. 单击**global**。
3. 单击操作 > **CLI** 工具，然后使用以下命令替换相关参数：

```
kubectl annotate --overwrite -n cpaas-system clusters.cluster.x-k8s.io
<imported cluster name> \
        cpaas.io/platform-url=<prepared domain address, e.g.: https://w
ww.domain.cn>
```

代码示例：

```
kubectl annotate --overwrite -n cpaas-system clusters.cluster.x-k8s.io
cluster-imported \
        cpaas.io/platform-url=https://www.domain.cn
```

如何获取导入集群信息？

目录

问题描述
先决条件
获取集群信息
获取集群令牌
获取导入集群的 API server 地址和 CA 证书

问题描述

获取连接到导入集群所需的配置，以便平台获授权访问和管理该集群。

先决条件

- 可正常使用的 `kubectl` 环境。对于公有云集群，强烈建议使用提供商的 **CloudShell**。 如果无法使用 CloudShell，建议使用 Linux 或 macOS。
- 已获取导入集群的 KubeConfig 文件，并将其复制到安装了 `kubectl` 的环境中。为避免在错误的环境中操作，强烈建议采用以下任一种非破坏性方式：
  - 备份方式：先将现有 kubeconfig 复制到安全位置：`cp "${HOME}/.kube/config" "${HOME}/.kube/config.backup"`

- 隔离方式：设置 `KUBECONFIG` 环境变量，使其指向导入的 kubeconfig：`export KUBECONFIG="/path/to/imported/kubeconfig"`
- 合并方式：使用 kubectl 合并/扁平化配置，同时保留现有上下文：
  1. `export KUBECONFIG="/path/to/imported/kubeconfig:${HOME}/.kube/config"`
  2. `kubectl config view --flatten > "${HOME}/.kube/merged.kubeconfig"`
  3. `export KUBECONFIG="${HOME}/.kube/merged.kubeconfig"`

获取集群信息

获取集群令牌

1. 运行以下命令：

```
# [Important] The following operations support bash only

# Manually create a secret, bind a service account, and generate a non-expiring token
kubectl get ns cpaas-system > /dev/null 2>&1 || kubectl create namespace cpaas-system
kubectl create serviceaccount k8sadmin -n cpaas-system
kubectl create clusterrolebinding k8sadmin --clusterrole=cluster-admin --serviceaccount=cpaas-system:k8sadmin

cat | kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: k8sadmin
  namespace: cpaas-system
  annotations:
    kubernetes.io/service-account.name: "k8sadmin"
type: kubernetes.io/service-account-token
EOF

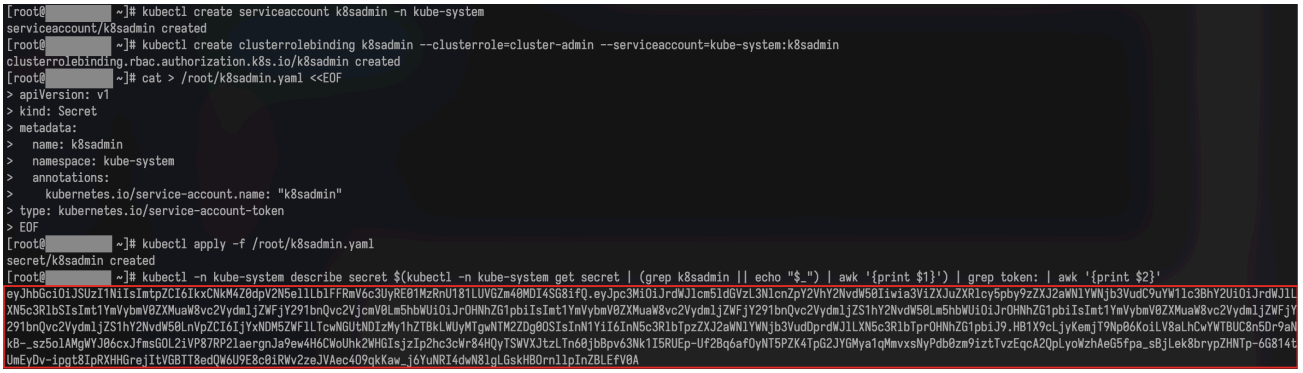
kubectl -n cpaas-system describe secret \
    $(kubectl -n cpaas-system get secret | (grep k8sadmin || echo "$_") | awk '{print $1}') \
    | grep -F 'token:' | awk '{print $2}'
```

WARNING

此操作步骤会创建一个不会过期的 cluster-admin 凭证。

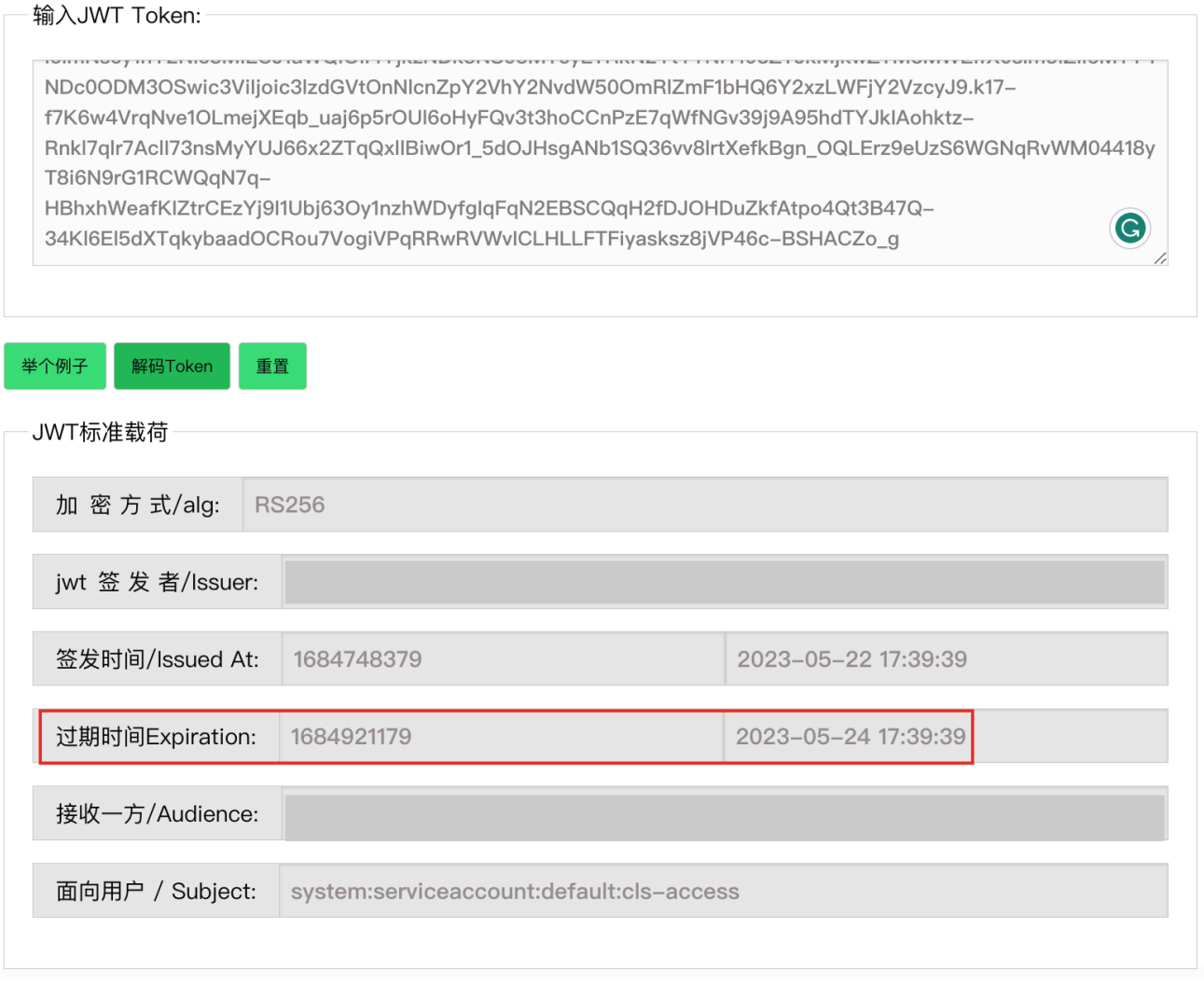
- 尽可能使用仅对所需资源具有权限的最小权限 RBAC。
- 安全存储令牌；不再需要时应轮换或撤销令牌。
- 限制可以读取 `Secret` 中 `cpaas-system` 对象的人员。

2. 以下是上一步获取的令牌示例。



3. 验证令牌是否过期。

   使用支持解析 JWT 令牌的任意工具分析令牌，并确认其过期时间。 如果在解析结果中发现过期字段（包含 "exp" 的键，如下所示），平台将在该时间之后无法管理导入集群。 在这种情况下，请停止操作并联系技术支持。



TIP

过期时间在原始 JWT 负载中记录为 `"exp": 1684486916,`。该值是 UNIX 时间戳，可转换为 UTC 时间。

完成后清理（撤销访问权限）：

```
kubectl delete clusterrolebinding k8sadmin
kubectl -n cpaas-system delete secret k8sadmin
kubectl -n cpaas-system delete serviceaccount k8sadmin
```

获取导入集群的 **API server** 地址和 **CA** 证书

> **TIP**
>
> 如果已在导入集群页面使用平台的 `Parse KubeConfig File` 功能获取 API server 地址和 CA 证书，请跳过此步骤。

1. 运行以下命令：

```
# View the import cluster API server addresses. There may be multiple a
ddresses; choose the one that fits your environment.
kubectl --kubeconfig "${KUBECONFIG:-${HOME}/.kube/config}" config view
--show-managed-fields=false --flatten --raw -ojsonpath='{$.clusters..cl
uster.server}'
addr_apiserver='<Selected API server address>'

# Get the CA certificate for the API server specified above
kubectl --kubeconfig "${KUBECONFIG:-${HOME}/.kube/config}" config view
--show-managed-fields=false --flatten --raw \
    -ojsonpath="{$.clusters[?(@.cluster.server == '${addr_apiserve
r}')].cluster.certificate-authority-data}" \
    | { base64 -d 2>/dev/null || base64 -D; }
```

如何信任不安全的镜像仓库？

目录

问题描述

配置对不安全镜像仓库的信任

问题描述

托管平台组件镜像的镜像仓库可能不提供 HTTPS 服务，或者没有由公共证书颁发机构签发的有效 TLS 证书。如果您信任此仓库，请按照以下步骤配置容器运行时。

配置对不安全镜像仓库的信任

注意：

- 所有需要使用镜像的节点（包括新添加的节点）都必须完成配置，并重启 Containerd。
- Containerd v1.4/v1.5 与 v1.6 之间的配置略有不同。请根据您的版本执行相应步骤。

1. 在导入集群中的每个节点上运行以下命令：

    - 备份配置文件

```
mkdir -p '/var/backup-containerd-confs/'
if ! [ -f /etc/containerd/config.toml ]; then
    echo 'Containerd config not found. Please check if containerd is
correctly installed. If you still cannot resolve the issue, contact t
echnical support.'
    exit 1
else
    cp /etc/containerd/config.toml /var/backup-containerd-confs/confi
g.toml_$(date +%F_%T)
fi
```

- 获取 Containerd 运行时版本

```
# Get the containerd version
# Compare this version to v1.6. Choose steps accordingly
ctr --version | grep -Eo 'v[0-9]+\.[0-9]+\.[0-9]+'
```

Containerd v1.4 v1.5 不安全仓库配置

2. 在导入集群中的每个节点上运行以下命令：

- 编辑 `/etc/containerd/config.toml`

```
# Example content to add to the config file
# Lines in brackets are sections. If the file already has sections wi
th the same name, merge their contents.
[plugins."io.containerd.grpc.v1.cri".registry]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<regis
try-address>"]
            endpoint = ["https://<registry-address>", "http://<regist
ry-address>"]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."192.16
8.134.43"]
            endpoint = ["https://192.168.134.43", "http://192.168.13
4.43"]
    [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."<regis
try-address>".tls]
            insecure_skip_verify = true
        [plugins."io.containerd.grpc.v1.cri".registry.configs."192.16
8.134.43".tls]
            insecure_skip_verify = true
```

- 重启 Containerd。

```
systemctl daemon-reload && systemctl restart containerd
```

Containerd v1.6 不安全仓库配置

2. 在导入集群中的每个节点上运行以下命令：

- 检查配置中是否存在 `config_path`。

```
if ! grep -qF 'config_path' /etc/containerd/config.toml; then
    if grep -qE '\[plugins."io.containerd.grpc.v1.cri".registry.(mirrors|configs)(\.|\])' /etc/containerd/config.toml; then
        echo 'Follow the steps in "Containerd v1.4 v1.5 configuration for insecure registries".'
    else
        cat >> /etc/containerd/config.toml << 'EOF'
[plugins."io.containerd.grpc.v1.cri".registry]
    config_path = "/etc/containerd/certs.d/"
EOF
    fi
fi

config_path_var=$(grep -F '/etc/containerd/certs.d' /etc/containerd/config.toml)
if [ -z "$config_path_var" ]; then
   echo 'The value of config_path in the file is unexpected. Please check!'
   exit 1
fi
```

- 创建 `hosts.toml` 文件。
  如果上一条命令输出了 `Follow the steps in "Containerd v1.4 v1.5 configuration for insecure registries".` ，请参阅 *Containerd v1.4 v1.5 不安全仓库配置*。

```
REGISTRY='<registry address obtained in the "Get the registry address" section>'

mkdir -p "/etc/containerd/certs.d/$REGISTRY/"
cat > "/etc/containerd/certs.d/$REGISTRY/hosts.toml" << EOF
server = "$REGISTRY"
[host."http://$REGISTRY"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
[host."https://$REGISTRY"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
EOF
```

- 重启 Containerd。

```
systemctl daemon-reload && systemctl restart containerd
```

从自定义命名的网卡采集网络数据

目录

场景说明

操作步骤

场景说明

创建业务集群后，平台监控默认只能识别网卡名称与 `eth.*|en.*|wl.*|ww.*` 等模式匹配的网卡。对于用户自定义命名的网卡，无法在监控页面查看网络流量数据。为解决此问题，平台支持修改相关资源参数，以手动采集网卡流量数据。

操作步骤

1. 登录 global 集群的控制平面节点，并使用 kubectl 执行以下命令。
2. 首先，查找 global 集群中与业务集群对应的 moduleinfo 资源名称：

```
kubectl get moduleinfo | grep -E 'prometheus|victoriametrics'
```

示例输出：

```
global-6448ef7f7e5e3924c1629fad826372e7     global     prometheus
prometheus                          Running   v3.15.0-zz231204040711-9d
1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
11-9d1fc12474c2
ovn-0954f21f0359720e8c115804376b3e7e        ovn        prometheus
prometheus                          Running   v3.15.0-zz231204040711-9d
1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
11-9d1fc12474c2
```

3. 编辑业务集群的 moduleinfo 资源，将 `ovn-0954f21f0359720e8c115804376b3e7e` 替换为上一步获取的业务集群 moduleinfo 资源名称：

```
kubectl edit moduleinfo ovn-0954f21f0359720e8c115804376b3e7e
```

4. 添加 valuesOverride 字段，并根据注释信息修改字段和正则表达式：

```
spec:
  valuesOverride: # If this field does not exist, you need to add the v
aluesOverride field and the following parameters under spec
    ait/chart-cpaas-monitor:
      ovn: # Replace with the workload cluster name
        indicator:
          networkDevice: eth.*|em.*|en.*|wl.*|ww.*|[A-Z].*i|custom_inte
rface # Replace custom_interface with a custom regular expression to en
sure correct network card name matching
```

5. 等待 10 分钟，然后检查节点监控页面上的网络相关图表，确保更改生效。

如何为导入的标准 Kubernetes 集群配置审计收集？

目录

场景描述

前提条件

操作步骤

场景描述

将标准 Kubernetes 集群导入平台后，必须先在集群上启用 Kubernetes API server 审计日志记录，平台才能从该集群收集审计数据。

对于控制平面节点由您管理的标准 Kubernetes 集群，请使用此操作步骤，例如基于 kubeadm 的集群。此操作步骤不适用于由云服务提供商管理的 Kubernetes 服务，因为您无法登录或修改其控制平面节点。

前提条件

- 标准 Kubernetes 集群已导入平台。
- 您可以登录集群中的每个控制平面节点。

- 集群使用标准 kubeadm 风格的 API server 静态 Pod 清单路径：`/etc/kubernetes/manifests/kube-apiserver.yaml`。

操作步骤

1. 创建一个本地 `policy.yaml` 文件作为审计策略。
   根据 Kubernetes 版本设置 `apiVersion`：

   - 早于 Kubernetes 1.24：`audit.k8s.io/v1beta1`
   - Kubernetes 1.24 及更高版本：`audit.k8s.io/v1`

   使用以下内容：

```
apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - "RequestReceived"
rules:
  - level: None
    users:
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
    resources:
      - group: ""
        resources: ["endpoints"]
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
  - level: None
    resources:
      - group: ""
        resources: ["events"]
  - level: None
    resources:
      - group: "devops.alauda.io"
  - level: None
    verbs: ["get", "list", "watch"]
  - level: None
    namespaces:
      - kube-system
      - cpaas-system
      - alauda-system
      - istio-system
      - kube-node-lease
    resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
  - level: None
    resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessreviews"]
```

```
    - group: "authentication.k8s.io"
      resources: ["tokenreviews"]
  - level: Metadata
    resources:
    - group: ""
      resources: ["secrets", "configmaps"]
  - level: RequestResponse
    resources:
    - group: ""
    - group: "aiops.alauda.io"
    - group: "apps"
    - group: "app.k8s.io"
    - group: "authentication.istio.io"
    - group: "auth.alauda.io"
    - group: "autoscaling"
    - group: "asm.alauda.io"
    - group: "clusterregistry.k8s.io"
    - group: "crd.alauda.io"
    - group: "infrastructure.alauda.io"
    - group: "monitoring.coreos.com"
    - group: "networking.istio.io"
    - group: "networking.k8s.io"
    - group: "portal.alauda.io"
    - group: "rbac.authorization.k8s.io"
    - group: "storage.k8s.io"
    - group: "tke.cloud.tencent.com"
    - group: "devopsx.alauda.io"
    - group: "core.katanomi.dev"
    - group: "deliveries.katanomi.dev"
    - group: "integrations.katanomi.dev"
    - group: "builds.katanomi.dev"
    - group: "operators.katanomi.dev"
    - group: "tekton.dev"
    - group: "operator.tekton.dev"
    - group: "eventing.knative.dev"
    - group: "flows.knative.dev"
    - group: "messaging.knative.dev"
    - group: "operator.knative.dev"
    - group: "sources.knative.dev"
    - group: "operator.devops.alauda.io"
  - level: Metadata
```

TIP

如果集群版本早于 1.24，则仅将 `apiVersion` 字段更改为 `audit.k8s.io/v1beta1`。策略内容的其余部分保持不变。

2. 将 `policy.yaml` 上传到每个控制平面节点上的 `/etc/kubernetes/audit/`。

> **WARNING**
>
> - 如果集群包含多个控制平面节点，请将文件上传到每个节点。
> - 如果目录不存在，请手动创建：`/etc/kubernetes/audit/`

3. 在每个控制平面节点上更新 `/etc/kubernetes/manifests/kube-apiserver.yaml`。
   在 `spec.containers[].command` 中添加或更新以下与审计相关的标志：

标志	必需	描述
`--audit-policy-file`	是	必须设置为 `/etc/kubernetes/audit/policy.yaml`。
`--audit-log-format`	是	必须设置为 `json`。
`--audit-log-path`	是	必须设置为 `/etc/kubernetes/audit/audit.log`。
`--audit-log-mode`	否	建议值：`batch`。
`--audit-log-maxsize`	否	审计日志文件的最大大小，单位为 MiB。建议值：`200`。
`--audit-log-maxbackup`	否	保留的审计日志文件数量。建议值：`2`。

示例：

```
- --audit-log-format=json
- --audit-log-maxbackup=2
- --audit-log-maxsize=200
- --audit-log-mode=batch
- --audit-log-path=/etc/kubernetes/audit/audit.log
- --audit-policy-file=/etc/kubernetes/audit/policy.yaml
```

4. 将审计目录挂载配置添加到同一个 `kube-apiserver.yaml` 文件中。
   在 `spec.containers[].volumeMounts` 下添加以下项目：

```
- mountPath: /etc/kubernetes/audit
  name: k8s-audit
```

   在 `spec.volumes` 下添加以下项目：

```
- hostPath:
    path: /etc/kubernetes/audit
    type: DirectoryOrCreate
  name: k8s-audit
```

> **WARNING**
>
> - 如果集群包含多个控制平面节点，请在每个控制平面节点上更新清单。
> - `volumeMounts[].name` 值必须与对应的 `volumes[].name` 值一致。
> - 请勿更改挂载路径 `/etc/kubernetes/audit`。

5. 保存文件并验证配置是否生效。
   检查每个控制平面节点上是否生成了 `/etc/kubernetes/audit/audit.log`。如果文件存在且包含审计记录，则表示配置已生效。

```
ls -l /etc/kubernetes/audit/audit.log
tail -n 20 /etc/kubernetes/audit/audit.log
```

操作指南

使用 **KubeConfig** 访问集群

为内置 **Registry** 添加外部地址

更新公共仓

使用 KubeConfig 访问集群

目录

简介
场景
选择访问路径
先决条件
操作步骤
下载并使用平台提供的 KubeConfig
下载 KubeConfig 文件
选择 KubeConfig 文件
验证集群访问
使用您自己的 ACP API 令牌构建基于令牌的 KubeConfig
创建 ACP API 令牌
准备 KubeConfig 模板
在 KubeConfig 文件中设置 ACP API 令牌
验证有效权限
使用专用账户进行 Pipeline 或 Connector 访问
生命周期和撤销
验证
后续步骤

简介

如果需要从本地使用 ACP CLI（`ac`）、`kubectl`、`helm`、`client-go` 或类似的 Kubernetes 客户端访问集群，请使用 KubeConfig。本页面中的操作步骤适用于 `global` 集群以及工作负载集群。

KubeConfig 文件是 YAML 客户端配置文件，不是 Kubernetes 资源。KubeConfig 文件通常会定义目标集群端点、客户端凭证以及将两者关联起来的上下文。

ACP CLI（`ac`）使用标准 kubeconfig，提供类似 kubectl 的使用体验。它与 `kubectl` 工作流兼容，同时还提供特定于平台的上下文、集群和会话操作。如果您已在使用 `ac login`，ACP CLI 可以自动配置 kubeconfig 并创建上下文。以下访问路径包括下载 KubeConfig 文件和手动组装基于令牌的 KubeConfig 文件。

场景

以下场景可使用这些访问路径：

- 集群管理员需要一个可直接使用的 KubeConfig，用于集群管理、故障排查或检查。
- 开发人员或项目用户需要一个与其自身 ACP 身份和 RBAC 权限相符的 KubeConfig。
- 流水线或连接器需要独立于个人日常账户的凭证。

选择访问路径

使用下表选择合适的访问路径：

访问路径	凭证来源	有效权限	生命周期控制	最适用场景
从平台下载的 KubeConfig	嵌入从平台下载的 KubeConfig 文件中的凭证	由下载文件中嵌入的凭证决定	由平台当前对下载文件的处理方式控制	集群管理员和模板准备
基于令牌的 KubeConfig	为用户账户创建的 ACP API 令牌	受限于令牌所有者的身份和 RBAC 范围	由令牌过期或撤销状态控制	开发人员、项目用户和自动化账户

> **当前行为**
>
> 目前，从 ACP 下载的 KubeConfig 文件默认有效期为 10 年。ACP 目前不支持在此访问路径中替换 CA，也不支持为此访问路径指定用户定义的集群 CA。

如果需要更严格的身份隔离或更短的生命周期控制，请使用基于令牌的 KubeConfig，而不要长期重复使用下载的文件。

先决条件

开始之前，请确保满足以下条件：

- 您可以登录 ACP 并访问目标集群。
- 您已在要使用 KubeConfig 文件的本地计算机上安装 ACP CLI（`ac`）或 `kubectl`。
- 如果要使用基于令牌的 KubeConfig，您可以从个人资料 > **API** 令牌创建 ACP API 令牌。有关详细信息，请参阅简介。
- 如果要将 KubeConfig 用于流水线或连接器，请准备一个专用账户，并在创建其令牌之前仅授予所需的权限。

操作步骤

下载并使用平台提供的 KubeConfig

如果需要通过平台提供的 KubeConfig 文件直接访问集群，请使用此路径。

1 下载 KubeConfig 文件

在管理员视图中，转到集群 > 集群，选择目标集群，打开集群详情页，点击操作，然后选择下载 **Kubeconfig**。

将下载的文件保存到本地路径，例如 `~/.kube/<cluster-name>.yaml`。

2 选择 KubeConfig 文件

将 ACP CLI（`ac`）或 `kubectl` 指向下载的文件：

```
export KUBECONFIG="$HOME/.kube/<cluster-name>.yaml"
kubectl config get-contexts
kubectl config current-context
ac config current-context
```

如果您管理多个集群，请使用 `kubectl config use-context <context-name>` 或 `ac config use-context <context-name>` 切换到正确的上下文，然后再运行命令。如果使用 ACP CLI 会话，也可以使用 `ac config use-cluster <cluster-name>` 在当前平台会话中切换集群。

3 验证集群访问

运行只读命令，确认 KubeConfig 文件按预期工作：

```
kubectl get nodes
kubectl get pods -A
ac get nodes
ac get pods -A
```

如果命令返回预期的集群资源，则 KubeConfig 文件可直接使用。

需要进行集群管理时，直接使用此下载文件。您也可以将其用作基于令牌的 KubeConfig 模板，使其同时适用于 `ac` 和 `kubectl`。

使用您自己的 ACP API 令牌构建基于令牌的 KubeConfig

如果您希望 KubeConfig 权限与特定 ACP 用户或自动化账户匹配，请使用此路径。

此路径不会提升权限。KubeConfig 文件仅携带连接和身份验证设置。实际访问边界仍取决于令牌所有者及该身份的 RBAC 分配。

1 创建 ACP API 令牌

打开个人资料 > **API** 令牌，创建新令牌，并设置与您的使用场景相匹配的过期时间。

安全存储令牌。如果计划将其用于自动化，请避免使用个人日常账户。

2 准备 KubeConfig 模板

使用以下方法之一：

- 下载平台提供的 KubeConfig 文件，并另存一份副本用于基于令牌的访问。
- 手动创建标准 KubeConfig 文件。

如果手动创建文件，请使用以下格式的目标集群端点：

```
https://<platform-access-address>/kubernetes/<cluster-name>
```

如果不想从头构建集群部分，请先下载平台提供的 KubeConfig，并将其中的集群和 CA 字段作为模板重复使用。

3 在 KubeConfig 文件中设置 ACP API 令牌

将 `users[].user.token` 值替换为您为目标用户或自动化账户创建的 ACP API 令牌。

以下示例展示了基于令牌的 KubeConfig：

KubeConfig.yaml

```
apiVersion: v1
kind: Config
clusters:
  - name: <cluster-name>
    cluster:
      server: https://<platform-access-address>/kubernetes/<cluster-n
ame>
      certificate-authority-data: <base64-encoded-ca-data>
users:
  - name: <user-name>
    user:
      token: <platform-api-token>
contexts:
  - name: <context-name>
    context:
      cluster: <cluster-name>
      user: <user-name>
      namespace: <namespace>
current-context: <context-name>
```

`namespace` 是可选的。它用于设置命令的默认命名空间，但不会扩大令牌的权限范围。

4 验证有效权限

使用新的 KubeConfig 文件，并验证返回的权限是否与预期角色匹配：

```
export KUBECONFIG="$HOME/.kube/<user-or-automation>.yaml"
kubectl auth can-i get pods -n <namespace>
kubectl get pods -n <namespace>
ac config current-context
ac get pods -n <namespace>
```

如果命令针对 `kubectl auth can-i` 返回 `no`，请授予所需角色或使用其他账户。不要将 KubeConfig 更改视为 RBAC 更改的替代方案。

使用专用账户进行 **Pipeline** 或 **Connector** 访问

对于 Pipeline 或 Connector 访问，请使用专用账户，而不是个人用户账户：

1. 为 Pipeline 或 Connector 创建专用账户。
2. 仅向该账户授予所需的项目级或租户级权限。
3. 为该账户创建一个 ACP API 令牌。
4. 使用该令牌构建基于令牌的 KubeConfig。
5. 将令牌或 KubeConfig 文件存储在自动化平台的密钥管理系统中。
6. 通过创建新令牌、更新密钥并撤销旧令牌来轮换凭证。

此模式为自动化提供独立凭证，但它不是 ACP 中原生的项目级 KubeConfig 对象。

生命周期和撤销

生命周期行为取决于您使用的访问路径：

访问路径	有效性控制因素	当前行为
平台下载的 KubeConfig	当前平台对下载文件的行为	下载的 KubeConfig 当前默认有效期为 10 年。ACP 当前不为此路径提供 CA 替换或用户定义的集群 CA。
基于令牌的 KubeConfig	`users[].user.token` 中的 ACP API 令牌	KubeConfig 仅在令牌有效期间保持可用。令牌过期或被撤销后，KubeConfig 也会停止工作。

如果您需要更短的有效期或更便捷的撤销方式，请优先使用基于令牌的路径。

验证

准备好 KubeConfig 文件后，执行以下检查：

1. 运行 `kubectl config get-contexts`，确认存在预期的上下文。
2. 运行 `kubectl get pods -A` 或 `kubectl get pods -n <namespace>` 等只读命令，确认响应与预期的访问范围一致。
3. 对于基于令牌的 KubeConfig，运行 `kubectl auth can-i <verb> <resource> -n <namespace>`，确认结果与令牌所有者的角色一致。

4. 如果使用 ACP CLI，运行 `ac config current-context` 或 `ac namespace`，确认活动上下文与预期的集群和命名空间一致。
5. 如果使用 ACP CLI，运行 `ac get pods -n <namespace>` 或 `ac get nodes`，确认响应与预期的访问范围一致。

后续步骤

- 有关 ACP API 令牌管理，请参阅简介。
- 有关 ACP CLI 基础知识，请参阅 CLI 入门。
- 有关 ACP CLI 上下文和会话管理，请参阅 管理 CLI 配置文件。
- 有关 ACP CLI 与 kubectl 的兼容性，请参阅 ac 和 kubectl 命令的使用。

为内置 **Registry** 添加外部地址

目录

概述

前提条件

操作步骤

为平台 Registry 配置证书和路由规则

概述

当 `global` 集群使用 `Platform Built-in` Registry 时，业务集群通常也使用此 Registry 拉取镜像。该 Registry 不仅为 `global` 集群内的组件提供服务，还必须能够被业务集群节点访问。

在某些场景下，业务集群节点无法直接访问 `global` 集群的 Registry 地址，例如，`global` 集群位于私有数据中心，而业务集群位于公有云或边缘环境中。

为平台的默认 Registry 配置一个可从外部访问的地址，以便业务集群拉取镜像。

前提条件

开始前，请准备以下内容：

- 业务集群节点可访问的域名
- 域名指向的 IP 地址
- 域名的有效 SSL 证书

> **WARNING**
>
> - 域名必须与平台访问地址不同
> - 确保域名的 IP 地址可以将流量转发到 `global` 集群的所有控制平面节点

操作步骤

为平台 **Registry** 配置证书和路由规则

1. 将域名的有效证书复制到 `global` 集群的任意控制平面节点
2. 创建包含域名证书的 TLS secret：

   ```
   kubectl create secret tls registry-address.tls --cert=<certificate-file
   name> --key=<key-filename> -n kube-system
   ```

   示例：

   ```
   kubectl create secret tls registry-address.tls --cert=custom.crt --key=
   custom.key -n kube-system
   ```

   注意：创建证书后，监控 `global` 集群 `kube-system` 命名空间中 **registry-address.tls** secret 的过期日期。在证书过期前将其替换。
3. 在 `global` 集群的任意控制平面节点上创建 ingress 规则：

```
REGISTRY_DOMAIN_NAME=<www.registry.com> # Replace with your accessible
domain name
cat << EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
  name: registry-address
  namespace: kube-system
  labels:
    service_name: registry
spec:
  rules:
    - host: $REGISTRY_DOMAIN_NAME
      http:
        paths:
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/_catalog
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/tags/list
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
```

```
            path: /v2/.+/manifests/[A-Za-z0-9_+.-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobs/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobs/uploads/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /auth/token
            pathType: ImplementationSpecific
  tls:
    - secretName: registry-address.tls
      hosts:
        - $REGISTRY_DOMAIN_NAME
EOF
```

如果返回类似于 `... created` 的响应，则表示 ingress 创建成功。

4. 检查是否存在 Registry Service 资源：

```
kubectl -n kube-system get svc | grep registry
```

如果 Service 不存在，请使用以下命令创建：

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Service
metadata:
  labels:
    name: registry
    service_name: registry
  name: registry
  namespace: kube-system
spec:
  ports:
    - protocol: TCP
      port: 443
      targetPort: 60080
  selector:
    component: registry
  type: ClusterIP
EOF
```

5. 使用域名从 Registry 拉取镜像，测试配置：

```
crictl pull <registry-domain-name>/automation/qaimages:helloworld
```

或

```
podman pull <registry-domain-name>/automation/qaimages:helloworld
```

更新公共仓库凭证

目录

概述

操作步骤

概述

`Public Repository` 是由平台提供、可通过公共互联网访问的镜像注册表服务。如果您希望集群使用 `Public Repository` 作为镜像注册表，则需要更新内置的 `public-registry-credential` Cloud Credentials。这样可确保平台有权限从公共注册表拉取镜像。

`public-registry-credential` 是平台内置的注册表凭证，其作用不限于配置为使用 `Public Repository` 的集群。当平台注册表需要身份验证时，通过基础设施提供商置备的 Alauda OS 集群会使用此凭证解析其平台镜像源，除非该集群引用了自己的凭证。因此，更新此凭证也会影响这些集群。（如果平台注册表允许匿名访问，则不会填充此凭证，这些集群会根据安装时记录的地址解析其注册表，因此在此处更新不会影响它们。）

更改内容前，请列出可能受影响的集群——这些集群没有自己的注册表引用：

```
kubectl get cluster.platform.tkestack.io \
  -o custom-columns='NAME:.metadata.name,REGISTRY_REFERENCE:.metadata.lab
els.cpaas\.io/registry-reference'
```

第二列显示 `<none>` 的任何集群都会继承此凭证。

> **为集群配置自己的注册表凭证**
>
> 集群可以绑定自己的注册表凭证，而不是继承此凭证，以便从距离更近的注册表拉取平台镜像。绑定在创建集群时选择，之后无法更改。请参阅为业务集群选择镜像注册表。

操作步骤

1. 登录 灵雀云 **Customer Portal**，并从右上角 **User Information** 下拉菜单中的 **Enterprise Management** 部分下载组织的身份验证文件。
2. 在 **Administrator** 控制台左侧导航栏中，依次进入 **Clusters** > **Cloud Credential**。
3. 找到名为 `public-registry-credential` 的云凭证，然后点击右侧下拉菜单中的 **Update**。
4. 在 **Upload Public Repository Address** 部分，上传从 灵雀云 **Customer Portal** 下载的身份验证文件。
5. 点击 **Update** 以应用更改。