节点

概览

概览

基础设施节点

基础设施节点

Alauda OS 节点

配置 **MachineHealthCheck**

管理裸金属节点

管理裸金属主

管理 **Huawei DCS** 上的节点

在 **Huawei Cloud Stack** 上管理节

在现有 **HCS**

在 **VMware vSphere** 上管理节点

非 **Alauda OS** 节点

添加和删除节点

平台管理员可以向节点运行非 Alauda OS 的

管理节点

在 Web 控制台中更新节点标签和污点、停.

为节点规划

规划并验证在非

概览

节点是集群的基本构建块。节点可以是虚拟机或物理服务器，并且每个节点都会运行托管 Pod 所需的组件，包括 kubelet、kube-proxy 和容器运行时。

具有平台管理权限的用户可以在集群下添加、管理和规划节点。

> **NOTE**
>
> 不支持向导入的集群添加节点，也不支持从导入的集群删除节点。

目录

- 节点类型
- 节点操作路径
  - Alauda OS 节点
  - 非 Alauda OS 节点
- Linux 节点可用性检查
- 支持的操作系统和 CPU 型号

节点类型

- 控制平面节点运行 kube-apiserver、kube-scheduler、kube-controller-manager、etcd、容器网络以及部分平台管理组件等集群组件。
- 工作节点托管在集群上运行的业务 Pod。请根据工作负载需求规划工作节点数量。

节点操作路径

以下任务适用于节点运行 Alauda OS 的集群，以及节点运行非 Alauda OS 的集群。

需求	继续阅读
规划基础设施节点或自定义角色节点。	基础设施节点
为性能敏感型工作负载调整 CPU、内存或拓扑管理器策略。	资源管理器策略

Alauda OS 节点

Alauda OS 集群上的节点通过 Cluster API 资源进行管理，并且会被替换，而不是在原位置修改。

需求	继续阅读
配置机器健康检查，或在特定基础设施提供商上扩展和维护节点。	Alauda OS 节点

非 Alauda OS 节点

需求	继续阅读
向集群添加节点或从集群删除节点。	添加和删除节点
管理标签、污点和节点调度状态。	管理节点
为符合条件的节点规划原地操作系统升级。	规划节点的原地操作系统升级

Linux 节点可用性检查

在非 Alauda OS 上构建业务集群之前，请完成节点检查，并确认每个节点均满足要求。未满足前置条件可能导致集群部署失败。

支持的操作系统和 CPU 型号

有关支持的操作系统和 CPU 型号，请参阅支持的操作系统和 CPU 型号。

基础设施节点

基础设施节点是专用于托管基础设施组件的工作节点。为工作节点添加基础设施角色标签和污点，即可创建基础设施节点，然后将选定的组件迁移到该节点。

目录

节点角色

在 Alauda OS 节点上创建基础设施节点

步骤 1：在节点池上声明基础设施标签和污点

步骤 2：验证

在非 Alauda OS 节点上创建基础设施节点

步骤 1：为节点资源添加基础设施角色标签

步骤 2：为节点资源添加污点

步骤 3：验证标签和污点

将工作负载移动到基础设施节点

自定义角色节点

节点角色

集群使用格式为 `node-role.kubernetes.io/<role>` 的 Kubernetes 节点角色标签为节点分配不同的角色。在本主题中，`role label` 指的是此类标签。

默认情况下，集群包含两种类型的节点：控制平面节点和工作节点，分别用于托管控制平面工作负载和应用工作负载。

在集群中，控制平面节点带有角色标签 `node-role.kubernetes.io/control-plane`。

> 注意：
> 在 Kubernetes v1.24 之前，Community 也使用标签 `node-role.kubernetes.io/master` 标记控制平面节点。为了向后兼容，这两个标签均可用于识别控制平面节点。

对于计划用于日志记录、监控和 ingress 等基础设施组件的节点或节点池，请使用角色标签 `node-role.kubernetes.io/infra`。要进一步划分这些节点，可以定义自己的角色，例如 `node-role.kubernetes.io/log` 或 `node-role.kubernetes.io/monitor`。

使用角色标签和污点创建基础设施节点或自定义角色节点，然后在工作负载隔离需要时，将选定的工作负载迁移到这些节点。

在 Alauda OS 节点上创建基础设施节点

在 Alauda OS 集群上，请勿手动将角色标签或污点应用于 `Node` 资源：此架构中的节点会被替换，而不是进行变更，因此应用于单个 `Node` 的任何设置都会在下一次节点替换时消失。请改为在节点池资源上声明标签和污点，这样节点池创建的每个节点（包括替换节点）都会携带这些设置。

以下步骤在节点池上声明基础设施角色。要创建节点池，请参阅提供商的“创建集群”页面：裸金属、Huawei DCS、Huawei Cloud Stack 或 VMware vSphere。

步骤 1：在节点池上声明基础设施标签和污点

在节点池的 `MachineDeployment` 中，将标签设置在 `.spec.template.metadata.labels` 上，并将污点设置在 `.spec.template.spec.taints` 上。对于控制平面节点池，请改为在其 `KubeadmControlPlane` 的 `.spec.machineTemplate.metadata.labels` 和 `.spec.machineTemplate.taints` 上进行设置。示例仅展示定义基础设施角色的字段；请保持 `MachineDeployment` 的其他字段不变。

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <infra-md-name>
  namespace: <namespace>
spec:
  clusterName: <cluster-name>
  template:
    metadata:
      labels:
        node-role.kubernetes.io/infra: ""
    spec:
      taints:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
```

在节点池模板中添加或更改的标签和污点会应用于该节点池的现有节点，而不会替换这些节点。

步骤 2：验证

节点池中的节点注册后，请检查 `Node` 是否具有基础设施角色标签和污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`，例如使用 `kubectl describe node <node-name>` 进行检查。

在单集群部署中，`global` 集群同时充当业务集群，这是一种受支持的部署形式；其节点规划遵循与其他相同规模集群相同的规则。

在非 Alauda OS 节点上创建基础设施节点

默认情况下，集群仅包含控制平面节点和 worker 节点。如果要将某些 worker 节点指定为专用于托管基础设施组件的基础设施节点，需要手动为这些节点添加相应的角色标签和污点。

> 注意：
> 本节中的操作仅适用于节点运行非 Alauda OS 的集群。请勿对 provider 管理的云集群、第三方集群或节点运行 Alauda OS 的集群使用这些操作。

步骤 1：为节点资源添加基础设施角色标签

```
kubectl label nodes 192.168.143.133 node-role.kubernetes.io/infra="" --ov
erwrite
```

此命令会为节点 192.168.143.133 添加基础设施角色标签：`node-role.kubernetes.io/infra: ""`，表示该节点是基础设施节点。

步骤 2：为节点资源添加污点

添加污点，以防止其他工作负载被调度到基础设施节点上。

```
kubectl taint nodes 192.168.143.133 node-role.kubernetes.io/infra=reserve
d:NoSchedule
```

此命令会为节点 192.168.143.133 添加污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`，表示只有容忍此污点的原生应用才能调度到该节点上。

步骤 3：验证标签和污点

检查节点是否已分配基础设施角色标签和污点：

```
# kubectl describe node 192.168.143.133
Name:               192.168.143.133
Roles:              infra
Labels:             node-role.kubernetes.io/infra=""
                    ...
Taints:             node-role.kubernetes.io/infra=reserved:NoSchedule
```

输出表明，节点 `192.168.143.133` 已配置为基础设施节点，并具有 `node-role.kubernetes.io/infra=reserved:NoSchedule` 污点。

如果要改为在 Web 控制台中为现有节点设置标签和污点，请参阅管理节点。

将工作负载移动到基础设施节点

如果要将特定 Pod 调度到基础设施节点上，需要进行以下配置：

- 以基础设施角色标签为目标的 nodeSelector。
- 与基础设施节点污点对应的 tolerations。

下面是一个配置为在基础设施节点上运行的 Pod 清单示例。

```
apiVersion: v1
kind: Pod
metadata:
  name: infra-pod-demo
  namespace: default
spec:
  ...
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
    operator: Equal
  ...
```

nodeSelector 确保 Pod 仅调度到具有标签 `node-role.kubernetes.io/infra: ""` 的节点上，而 toleration 允许 Pod 容忍污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`。

完成这些配置后，Pod 将被调度到基础设施节点上。

> 注意：
> 将通过 OLM Operators 或集群插件安装的 Pod 移动到基础设施节点上并不总是可行。能否移动这些 Pod 取决于每个 operator 或集群插件的配置。

对于以下组件，文档介绍了如何将其放置在基础设施节点上：

- Monitoring：为 Monitoring 规划基础设施节点。
- ALB：ALB 相关任务。

- Envoy Gateway：Envoy Gateway 相关任务。
- Operators：Operator 的“配置 Operator Pod 调度”部分。

自定义角色节点

要进一步划分基础设施节点，可以定义自定义角色，例如用于日志组件的日志节点，或用于监控组件的监控节点。角色名称可以自定义。

自定义角色节点的工作方式与基础设施节点完全相同，只是使用您的角色名称替代 `infra`：

- 使用角色标签 `node-role.kubernetes.io/<role>` 和污点 `node-role.kubernetes.io/<role>=<value>:NoSchedule`。`<value>` 是可选的，但可以帮助 operator 了解调度意图，例如 `reserved` 或 `dedicated`。
- 对于 Alauda OS 节点，每个角色使用一个节点池。每个节点池都有自己的 `node-role.kubernetes.io/<role>` 标签和污点，并按照在 Alauda OS 节点上创建基础设施节点中的方式在节点池上声明。
- 对于非 Alauda OS 节点，按照在非 Alauda OS 节点上创建基础设施节点 中的方式，将标签和污点添加到 `Node`。
- 要将 Pod 调度到这些节点上，请按照将工作负载移动到基础设施节点 中的方式配置 nodeSelector 和 tolerations。

管理 Alauda OS 集群上的节点

本节介绍节点运行 Alauda OS 的集群上的 day-2 节点操作。

INFO

`global` 集群的节点

在扩展节点、缩减节点、替换控制平面节点或工作节点，或在 `global` 集群本身上依赖 MachineHealthCheck 之前，请确保运行 `global` 的平台的基础设施提供商和 Cluster Bootstrap Provider 已作为集群插件安装在 `global` 上；安装 `global` 不会将它们保留在该集群上。请参阅 Bare Metal、Huawei DCS、Huawei Cloud Stack 或 VMware vSphere 的提供商安装页面。在 DR 对中，请在两个 `global` 集群上执行此操作。

目录

所有提供商的任务

本节页面

相关主题

所有提供商的任务

- 创建基础设施节点
- 配置 MachineHealthCheck

本节页面

配置 MachineHealthCheck

概述

前提条件

为 MachineDeployment 配置 MachineHea

为控制平面节点配置 MachineHealthCheck

参考

管理裸金属节点

先决条件

概述

节点管理操作

Pool 和 Inventory 可观测性

常见故障模式

后续步骤

管理裸金属主

可用性和升级边

存储生命周期

前提条件

步骤 1：验证 In

步骤 2：定义受

步骤 3：生成并

步骤 4：等待 P

步骤 5：分配并

在 Huawei Cloud Stack 上管理节点

先决条件

概述

使用 Web UI

使用 YAML

验证

故障排除

在 VMware vSphere 上管理节点

前提条件

步骤

更新 Bootstrap 模板

运行时拓扑更改

故障排查

后续步骤

相关主题

- 在 Alauda OS 上创建工作负载集群
- 升级 Alauda OS 工作负载集群上的 Kubernetes
- 机器配置

配置 MachineHealthCheck

目录

概述

前提条件

为 MachineDeployment 配置 MachineHealthCheck

为控制平面节点配置 MachineHealthCheck

参考

概述

MachineHealthCheck 是 Cluster API 中的一种资源，允许用户定义用于判断 Cluster 中 Machine 是否不健康的条件。MachineHealthCheck 定义在管理集群上，并限定作用于特定的业务集群。

定义 MachineHealthCheck 时，用户需要为所定义的每个条件指定超时时间，这些条件用于检查 Machine 的节点。如果任一条件在超时时间内持续满足，系统将对该 Machine 执行修复。默认情况下，修复 Machine 的操作应触发创建新的 Machine 来替换故障 Machine，但 provider 可以接入更复杂的外部修复方案。

WARNING

MachineHealthCheck 依赖 Cluster API 的滚动更新机制。在滚动更新期间，之前挂载的磁盘会被移除，并在新创建的 Machine 上替换为新磁盘。请确保没有任何集群功能或工作负载依赖存储在原始磁盘上的数据。

前提条件

在尝试配置 MachineHealthCheck 之前，您应拥有一个正常运行的管理集群，并且其中至少已部署一个 MachineDeployment 或 KubeadmControlPlane。若要在 `global` 集群本身上使用 MachineHealthCheck，必须先在 `global` 上安装基础设施 provider 和 Cluster Bootstrap Provider；请参阅在 Alauda OS 集群上管理节点。

为 MachineDeployment 配置 MachineHealthCheck

要为 MachineDeployment 配置 MachineHealthCheck，您需要在管理集群中创建 MachineHealthCheck 资源。

```
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: <machinehealthcheck-name>
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  nodeStartupTimeout: 30m
  selector:
    matchExpressions:
    - key: cluster.x-k8s.io/deployment-name
      operator: Exists
  unhealthyConditions:
  - type: Ready
    status: Unknown
    timeout: 600s
  - type: Ready
    status: "False"
    timeout: 600s
```

为控制平面节点配置 MachineHealthCheck

```yaml
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: <control-plane-machinehealthcheck-name>
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  maxUnhealthy: 1
  selector:
    matchLabels:
      cluster.x-k8s.io/control-plane: ""
  unhealthyConditions:
    - type: Ready
      status: Unknown
      timeout: 600s
    - type: Ready
      status: "False"
      timeout: 600s
```

WARNING

控制平面节点的限制和风险

- **Etcd leader** 故障与 **Kubelet** 停机：如果不健康的节点当前是 etcd leader，且其 `kubelet` 服务已停止，自动修复可能会失败。Cluster API 会在删除节点前尝试迁移 etcd leader，但依赖 `kubelet` 将请求代理到本地 etcd 实例。如果 `kubelet` 无响应，则无法继续执行 leader 转移，从而阻止轮换。
- 丢失法定人数的风险：为控制平面节点配置 `MachineHealthCheck` 时请谨慎。过于激进的超时时间或错误配置可能导致移除多个控制平面节点，从而破坏 etcd 法定人数，使集群的控制平面无法使用。

WARNING

参考

配置 MachineHealthCheck↗

管理裸金属节点

本文档介绍如何在裸金属 provider 上扩缩工作节点、替换指定机器，以及恢复故障节点。节点管理使用 Cluster API `Machine` 资源，并通过 `MachineDeployment` 进行编排；provider 会将每个 `Machine` 绑定到池中的一个 `MachineInventory`，并通过 `clean` / `reprovision` 计划驱动主机。

> **INFO**
>
> 如需升级节点的 Kubernetes 版本，请参阅在裸金属上升级 Kubernetes。

目录

- 先决条件
- 概述
- 节点管理操作
  - 扩缩工作节点
    - 添加工作节点
    - 移除工作节点
  - 替换单个故障节点
  - 将工作节点移至其他池
  - 更新引导模板
- Pool 和 Inventory 可观测性
- 常见故障模式
- 后续步骤

先决条件

WARNING

重要先决条件

- 控制平面必须已在运行。请参阅创建集群。
- 工作节点 `MachineInventoryPool` 必须包含至少与目标副本数相同的 `Available` 库存项，此外还需满足滚动更新策略所需的容量余量。
- 目标 `Machine.spec.version` 必须是 `elemental-image-catalog` ConfigMap 中的一个键。
- 任何 `spec.storage.volumes[]` 非空的库存项，在计为 `Available` 之前，都必须已报告 `status.storage.phase=Prepared` 和 `StoragePrepared=True`。在将新注册的存储管理库存项添加到工作节点池之前，请遵循管理裸金属主机上的数据磁盘中的说明。
- 如需管理 `global` 集群本身的节点，必须先在 `global` 上安装 provider；请参阅管理 Alauda OS 集群上的节点。

INFO

配置指南

使用本文档中的配置时：

- 仅修改 `<>` 括号内的值。
- 将占位值替换为适用于您环境的设置。
- 除非有明确要求，否则保留所有其他默认配置。

概述

组成工作节点组的四种资源：

1. **`MachineInventoryPool`**（`<cluster-name>-worker-pool`）— 允许使用的 `MachineInventory` 名称集合。已在创建集群 → 第 2 步中创建。
2. **`BaremetalMachineTemplate`**（`<cluster-name>-worker-template`）— 引用工作节点池。如需将工作节点移至其他池，请参阅将工作节点移至其他池。
3. **`KubeadmConfigTemplate`**（`<cluster-name>-worker-bootstrap`）— `kubeadm join` 的 cloud-init `user-data`。裸金属 provider 会在重新配置时规范化此用户数据（主机名、`provider-id`、`criSocket`）；operator 不应预先填充这些字段。
4. **`MachineDeployment`** — 控制副本数、版本和滚动更新策略。

托管数据磁盘不属于 `BaremetalMachineTemplate`。其声明和保留的数据属于每个长期运行的 `MachineInventory`，因此，除非 operator 对存储声明进行标准化并验证，否则同一工作节点池中的两个成员可能会公开不同的主机本地数据集。托管存储需在未分配且处于非活动状态的 `MachineInventory` 上单独更改；请参阅管理裸金属主机上的数据磁盘。

如需使用 YAML 添加另一个工作节点池，请按照集群创建页面中的第 4 步：部署工作节点操作。

节点管理操作

扩缩工作节点

添加工作节点

用例：扩充集群容量。

先决条件：

- 工作节点池中有足够的 `Available` 库存项。如果没有，请注册其他主机（启动 SeedImage ISO 并确认新的 `MachineInventory` 对象），准备好任何托管存储，并将它们添加到 `MachineInventoryPool.spec.machineInventories[]`。

操作步骤：

1. 检查当前状态

```
kubectl -n cpaas-system get machines.cluster.x-k8s.io \
  -l cluster.x-k8s.io/deployment-name=<cluster-name>-workers
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluste
r.x-k8s.io \
  <cluster-name>-worker-pool
```

确认 `MachineInventoryPool.status.available` 足以满足计划增加的数量。

2. 需要更多库存项时扩展工作节点池

```
kubectl -n cpaas-system edit machineinventorypool <cluster-name>-worker
-pool
```

添加新的 `MachineInventory` 名称——这些名称对应的库存项必须已注册且处于 Ready 状态。具有托管卷的库存项还必须处于 Prepared 状态；否则，它们会增加 `status.total`，但仍无法分配。保存并退出，然后确认 `status.available` 按预期增加。

3. 扩展 **`MachineDeployment`**

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<n
ew-replica-count>}]'
```

4. 监控

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-
k8s.io -w
kubectl get nodes                # workload cluster
```

新的 `BaremetalMachine` 对象会推进 `Pending → Allocated → Reprovisioning → Running`；随着新节点绑定，池的 `available` 计数器会递减。

移除工作节点

支持两种策略，其形式与上游 Cluster API 合约相同：

策略	适用情况
随机移除	可移除任意节点——例如，临时缩减容量。
定向移除	需要释放特定物理主机（硬件维护、更换、IP 回收）。

INFO

库存项回收

`clean` 计划会停止 kubelet、停用已声明的存储挂载、清除 CRI 工作负载并停止 containerd。它不会擦除托管文件系统、Kubernetes 持久化目录或重置 OS——只有在之后为新的 `BaremetalMachine` 选择相同或不同的库存项并运行 `reprovision` 计划时，才会清理 OS。在此之前，库存项会返回 `Available`，并可再次分配。删除 Machine 执行的是 **Deactivate**，而不是存储 Release，因此该库存项上的存储声明、文件系统 UUID 和数据都会保留。

随机移除

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers \
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<new
-replica-count>}]'
```

CAPI 会按照其标准顺序选择要删除的机器。每个选中的 `BaremetalMachine` 都会进入 `Preparing`，provider 会写入一个 `clean` 计划；当计划报告 `Applied` 后，库存项会返回 `Available`。

定向移除

1. 识别机器

```
kubectl -n cpaas-system get machines.cluster.x-k8s.io \
  -l cluster.x-k8s.io/deployment-name=<cluster-name>-workers
```

2. 为目标机器添加注释

```
kubectl -n cpaas-system patch machine <machine-name> \
  --type='merge' -p='{"metadata":{"annotations":{"cluster.x-k8s.io/dele
te-machine":"true"}}}'
```

对要移除的每台机器重复此操作。

3. 按已添加注释的机器数量精确缩容

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<n
ew-replica-count>}]'
```

缩容数量少于目标数量会使部分已添加注释的机器继续保留；缩容数量多于目标数量，则会让其他随机机器也执行 `clean` 计划。

4. 验证清理结果

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-
k8s.io
kubectl -n cpaas-system get machineinventories.elemental.cattle.io <inv
entory-name> -o yaml
```

已释放的库存项必须显示 `baremetal.alauda.io/allocation-state=Available`，`baremetal.alauda.io/owner-*` 注释必须已清除，池注释则必须保留。`MachineInventory` 本身不会被删除。如果它声明了托管卷，则必须返回 `status.storage.phase=Prepared`，并带有 `StorageActive=False/Inactive`；其业务挂载路径必须处于非活动状态，同时文件系统 UUID 和数据保持不变。

替换单个故障节点

如果 `BaremetalMachine` 进入 `Failed`，最安全的恢复方法是删除故障的 `Machine`。CAPI 会立即创建替代的 `Machine`（因为 `replicas` 未更改），裸金属 provider 会从同一池中选取一个 `Available` 库存项。替代节点通常会使用不同的 `MachineInventory`；provider 不保证会重新选中刚释放的库存项。主机本地的托管数据不会随替代 Machine 一同迁移。如果应用必须重用特定库存项中的数据，请使用专用池或经过有意排序的池，并在依赖这些数据之前验证 `BaremetalMachine.status.machineInventoryRef`。

```
kubectl -n cpaas-system delete machine <machine-name>
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io -w
```

请单独调查原始故障：读取运行计划的主机上的 `BaremetalMachine.status.conditions`、`MachineInventory.status.plan.state`，以及失败计划 secret 中的 `failed-output` 键。常见根本原因记录在下文的常见故障模式部分。

将工作节点移至其他池

`BaremetalMachineTemplate` 仅包含池引用——模板中没有 CPU、内存或磁盘声明。

用例：新主机已注册并放入新的 `MachineInventoryPool`，且需要将节点迁移到这些主机上。创建一个指向新池的新 `BaremetalMachineTemplate`，然后将 `MachineDeployment` 指向它。节点将逐个替换。

开始之前：

- 目标池与模板位于同一命名空间中。
- 目标池中的 `Available` 库存项数量至少与 `MachineDeployment` 的副本数相同。每个迁移的节点会使用目标池中的一台主机；旧节点释放的主机会返回旧池，不会加入目标池。通过 `status.available` 查看数量：

  ```
  kubectl -n cpaas-system get machineinventorypools.infrastructure.cluste
  r.x-k8s.io <target-pool-name>
  ```

- 新主机满足与当前主机相同的先决条件：已注册且处于 Ready 状态；如果使用托管数据磁盘，则已准备好；镜像目录中包含目标 Kubernetes 版本和 CPU 架构；并且存在 CNI 网络设备。请参阅确认工作节点池容量。
- 在迁移完成之前，请保留旧模板和旧池。

操作步骤：

1. 创建新的 `BaremetalMachineTemplate`，并使用引用目标池的新 `metadata.name`。请使用配置工作节点 `BaremetalMachineTemplate` 中的格式。
2. 应用新模板。

3. 修补 `MachineDeployment`：

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='merge' \
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<new-te
mplate-name>"}}}}}'
```

4. 观察滚动替换（`maxSurge: 0`，每次一个节点）以及目标池的 `status.available`：

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-
k8s.io -w
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluste
r.x-k8s.io <target-pool-name>
```

预期情况：

- 旧节点会先被删除，然后才创建替代节点。如果目标池的主机耗尽，节点池会暂时少一个节点，直到添加主机为止。
- 节点名称和 IP 地址会发生变化。
- 节点本地数据不会迁移。托管数据磁盘上的数据仍保留在旧主机上，并绑定到其 `MachineInventory`，不会出现在新主机上。请在新主机上单独声明存储；请参阅管理裸金属主机上的数据磁盘。

如果迁移因没有空闲主机而停滞（`BaremetalMachine` 显示 `InventoryAllocated=False / Reason=PoolExhausted`），请向目标池添加 `Available` 主机；系统会自动选取这些主机。也可以通过将 `infrastructureRef.name` 修补回旧模板来回滚。已迁移的节点会逐个再次替换，因此旧池必须仍有足够的 `Available` 主机。

如需迁移控制平面节点，请使用相同的步骤，并改为修补 `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name`。节点会逐个替换。此外：

- `maxSurge: 0` 要求 `replicas` 至少为 3。
- 每次替换期间只有两个 etcd 成员运行。只有在控制平面运行正常时才开始，并确保目标池中有 3 台 `Available` 主机处于就绪状态。
- 新主机必须与控制平面 VIP 位于同一二层网络中。

迁移完成后，可以删除旧模板和旧池。旧节点释放的主机会留在旧池中。

更新引导模板

直接修改现有 `KubeadmConfigTemplate` 不会滚动更新现有机器；只有新创建的机器才会应用更改。

如需滚动应用引导配置更改：

1. 导出现有模板：

   ```
   kubectl -n cpaas-system get kubeadmconfigtemplate <cluster-name>-worker
   -bootstrap -o yaml \
     > new-worker-bootstrap.yaml
   ```

2. 更改 `metadata.name`，移除由服务器生成的字段（`resourceVersion`、`uid`、`creationTimestamp`、`managedFields`、`kubectl.kubernetes.io/last-applied-configuration`）以及整个 `status`，然后编辑所需字段。对于 Kubernetes 1.35 或更高版本，请将必需的 kubelet 修补设置添加到 `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`。

3. 应用新模板：

   ```
   kubectl apply -f new-worker-bootstrap.yaml
   ```

4. 修补 `MachineDeployment`，使其引用新模板：

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

   这会触发滚动替换。

Pool 和 Inventory 可观测性

Provider 在每个 `MachineInventory` 上维护的注解，是了解“此主机当前的用途”的权威来源：

注解	值	含义
`baremetal.alauda.io/pool`	Pool 名称	此 Inventory 所属的 Pool。
`baremetal.alauda.io/allocation-state`	`Available` / `Allocated` / `Preparing` / `Reprovisioning` / `Unavailable`	从 Provider 的角度看所处的生命周期阶段。
`baremetal.alauda.io/owner-cluster`	Cluster 名称	当前正在使用此 Inventory 的 Cluster（释放时会清除）。
`baremetal.alauda.io/owner-machine`	`Machine` 名称	所属的 Cluster API `Machine`。
`baremetal.alauda.io/owner-baremetalmachine`	`BaremetalMachine` 名称	所属的 Provider machine。

每个 Inventory 的默认 plan secret 也包含 `baremetal.alauda.io/plan.type`。正常的节点生命周期使用 `clean` 或 `reprovision`；Inventory 存储协调还会使用 `storage-prepare` 和 `storage-release`。诊断存储问题时，请结合计划类型检查 `MachineInventory.status.storage.conditions` 和 `.operation`。

常见故障模式

场景	预期状态	检查项
Pool 不存在	`BaremetalMachine` 保持为 `Pending`、`InventoryAllocated=False / Reason=PoolMissing`	`BaremetalMachineTemplate.s` 和命名空间

场景	预期状态	检查项
Pool 资源耗尽	`BaremetalMachine` 保持为 `Pending`、`InventoryAllocated=False / Reason=PoolExhausted`	`MachineInventoryPool.statu`
成员缺失	`MachineInventoryPool.MembersValid=False / Ready=False`	Pool 的 `status.unavailable`
Inventory 未 Ready	计入 `status.unavailable`	`MachineInventory.status.co`
plan secret 缺失	Inventory 不符合分配条件	`MachineInventory.status.pl`
存储未准备	Inventory 不计为 `Available`；`StoragePrepared=False`	`MachineInventory.status.st` 容，以及 `storage-prepare` p
Bootstrap secret 缺失	`BootstrapReady=False / Reason=BootstrapWaiting`	`Machine.spec.bootstrap.dat`
Catalog 未命中	`BaremetalMachine` `Failed`、`ImageResolved=False / Reason=ImageCatalogMiss`，未写入 plan	`elemental-image-catalog` k
Registry 注解缺失	`ImageResolved=False / Reason=ImageRegistryMissing`	`Cluster.metadata.annotatio`
重新置备 plan 失败	`BaremetalMachine` `Failed`，Inventory 标记为 `Unavailable`	plan secret `failed-output`；
Managed volume 激活失败	`BaremetalMachine` `StorageReady=False`；必需 volume 导致节点就绪状态受阻	稳定的设备标识、文件系统 UUI `failed-output`
Clean 或 Deactivate 失败	finalizer 阻止删除；Inventory 标记为 `Unavailable`	`clean` plan 输出、主机挂载，`MachineInventory.status.st`

有关完整的 operator 侧状态机参考，请参阅 Provider Overview → BaremetalMachine。

后续步骤

- 创建集群
- 管理数据磁盘
- 升级集群

管理裸金属主机上的数据磁盘

使用 `MachineInventory.spec.storage` 管理已连接到裸金属主机的数据磁盘。存储声明属于长期存在的 `MachineInventory`，而不属于临时的 Cluster API `Machine`，因此当节点被移除时，provider 可以保留文件系统和数据，并在之后重新使用相同的 inventory。

此功能不会创建、连接、分离、迁移、备份或复制物理磁盘或 SAN LUN。它只管理在 `spec.storage.volumes[]` 中明确列出的设备。所有未列出的设备仍会被观测，但不会受到管理。

DANGER

`InitializeIfBlank` 可以格式化磁盘

仅当你已确认目标 inventory 上准确的稳定设备 ID、确认该设备是可丢弃的空白数据磁盘，并且已独立备份任何重要数据后，才能使用 `InitializeIfBlank`。切勿从其他主机复制设备 ID，也不要将 `/dev/sdX`、`/dev/vdX`、`/dev/nvmeXnY` 或 `/dev/mapper/mpathX` 用作持久身份标识。

目录

可用性和升级边界

存储生命周期

前提条件

步骤 1：验证 Inventory 和 Observer

步骤 2：定义受管卷

步骤 3：生成并应用原子补丁

步骤 4：等待 Prepare 完成

步骤 5：分配并验证激活

步骤 6：缩减并重新使用清单

步骤 7：释放存储所有权

失败的操作与重试

限制

相关文档

可用性和升级边界

受管存储需要以下全部三个组件：

- Bare Metal provider，包括其存储控制器和失败即关闭的 `MachineInventory` 准入 webhook。
- 匹配的 `elemental-operator` CRD 和 observer API。
- 安装并启用 `elemental-storage-observer.service` 以及早期启动存储集成的主机 OS 镜像。

> **WARNING**
>
> 在启用受管存储之前，请从 Alauda 技术支持获取匹配的 `base-image` 和 `base-image-iso`。将 `base-image` 用于主机重新置备，将 `base-image-iso` 用于 `SeedImage` / 实时安装。这些镜像是目标版本支持的配对；不要从一个镜像引用推导另一个镜像引用，也不要替换为独立构建的镜像。

仅在 `global` 集群上升级 provider 和 `elemental-operator`，不会在由旧版 OS 镜像创建的主机上安装 observer service。

- 当 `spec.storage` 保持缺失或具有 `volumes: []` 时，现有集群在管理侧升级后仍可继续使用非受管存储。此路径不会创建存储计划或修改磁盘。
- 要在旧主机上启用受管存储，请先使用经过批准且支持 observer 的镜像重新安装、重置或替换该主机。在添加任何非空存储声明之前，确认 observer service 和最新的 `status.observedStorage` 报告均已就绪。

- 不要将 `spec.storage.volumes[]` 添加到正在运行或已分配的 inventory 中。只有在 inventory 未分配且所有受管卷均处于非活动状态时，才能更改存储声明。

存储生命周期

阶段	触发条件	预期状态	主机
Observe	主机注册且 observer 报告设备。	`Unmanaged` 或 `Observed`	只读会自何设
Prepare	operator 在可用的 inventory 上保存非空存储声明。	`Prepared`、`StoragePrepared=True`	接管系统化明空白务路挂载
Activate	Cluster API 将已准备好的 inventory 分配给 `BaremetalMachine`。	`Active`、`StorageActive=True`	重新安装态；载单 kube 启动
Deactivate	删除所属的 Cluster API `Machine`。	`Prepared`、`StorageActive=False`	清理受管系统有权予以
Release	inventory 可用且处于非活动状态时，从 `spec.storage` 中移除卷。	`Unmanaged`、`StoragePrepared=True/NoManagedVolumes`	移除权和据，除文数据

删除 Machine 会执行 **Deactivate**，而不是 Release。因此，正常缩容会保留 inventory 的存储声明和数据，以便日后重用。

前提条件

1. 按照安装 Bare Metal Provider 中的说明，在 `global` 集群上安装 Bare Metal Infrastructure Provider，并等待其 manager 部署和准入 webhook 变为 Ready。
2. 从 Alauda 技术支持获取匹配的、支持 observer 的 `base-image` 和 `base-image-iso`。使用 ISO 镜像安装主机，并配置配套的基础镜像，以供后续重新置备使用。
3. 在 Prepare 成功之前，将目标 `MachineInventory` 保持在生产池之外。它必须为 `Ready=True`、未分配，并且具有默认 plan Secret。
4. 独立备份现有数据。保留存储不等同于应用一致性备份。
5. 获取由完全相同的 provider 修订版本构建的 `storagectl` 二进制文件。`storagectl` 会复用 provider 的规范化、验证和哈希代码；请勿手动计算初始化授权哈希，也不要使用其他版本的二进制文件。
6. 提交 `InitializeIfBlank` 的身份必须拥有虚拟 `machineinventories/storageinitialize` 子资源上的 `update` 权限。纯 `Adopt` 声明不需要此额外权限。

设置特定操作所需的变量：

```
export BM_NS=cpaas-system
export BM_INVENTORY='<machineinventory-name>'
export BM_STORAGECTL='<path-to-matching-storagectl>'
export BM_STORAGE_FILE=/tmp/baremetal-storage.yaml
export BM_INVENTORY_FILE=/tmp/baremetal-machineinventory.json
export BM_PATCH_FILE=/tmp/baremetal-storage.patch.json
```

验证管理组件和授权：

```
kubectl -n "$BM_NS" rollout status \
  deployment/cluster-api-provider-baremetal-manager --timeout=5m
kubectl get mutatingwebhookconfiguration \
  cluster-api-provider-baremetal-mutating-webhook-configuration
kubectl get validatingwebhookconfiguration \
  cluster-api-provider-baremetal-validating-webhook-configuration
kubectl auth can-i update machineinventories.elemental.cattle.io \
  --subresource=storageinitialize -n "$BM_NS"
```

身份在提交或更改初始化授权之前，最后一条命令必须返回 `yes`。仅向有权格式化磁盘的特定管理员用户或 ServiceAccount 授予此权限。

步骤 1：验证 Inventory 和 Observer

确认 inventory 可用且当前没有所有者：

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTORY" \
  -o jsonpath='uid={.metadata.uid}{"\n"}allocation={.metadata.annotations.baremetal\.alauda\.io/allocation-state}{"\n"}baremetalMachine={.metadata.annotations.baremetal\.alauda\.io/owner-baremetalmachine}{"\n"}machine={.metadata.annotations.baremetal\.alauda\.io/owner-machine}{"\n"}cluster={.metadata.annotations.baremetal\.alauda\.io/owner-cluster}{"\n"}plan={.status.plan.secretRef.name}{"\n"}'
```

分配值必须为空或为 `Available`；所有所有者值都必须为空。如果 inventory 已分配，请删除所属的 CAPI Machine，并等待正常的清理和 Deactivate 流程。请勿手动移除所有者注解或 finalizer。

在主机上验证 observer：

```
sudo systemctl is-enabled elemental-storage-observer.service
sudo systemctl is-active elemental-storage-observer.service
sudo systemctl cat elemental-storage-observer.service
```

两条状态命令都必须成功。在 `global` 集群上，检查观察水位标记和设备信息：

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTORY" \
  -o jsonpath='{.status.observedStorage.observedAt}{" epoch="}{.status.observedStorage.reportEpoch}{" sequence="}{.status.observedStorage.sequence}{" bootID="}{.status.observedStorage.bootID}{"\n"}'

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTORY" -o json \
  | jq -r '.status.observedStorage.devices[] |
      [.id,.kind,.systemRole,(.sizeBytes|tostring),(.filesystem.type//"-"),
       (.filesystem.uuid//"-"),([.mounts[].path]|join(",")),
       ((.health.activePaths//0)|tostring),((.health.totalPaths//0)|tostring)] | @tsv'
```

仅使用报告最新且 `systemRole` 为 `Data` 的设备。

设备类型	稳定 ID	支持的策略
直连磁盘	`wwn:`、`nvme-eui:`、`nvme-nguid:`，或 observer 批准的 `serial:` ID	`Adopt` 或 `InitializeIfBlank`
现有分区	`partuuid:`	仅限 `Adopt`
多路径聚合映射	`wwid:`	`Adopt`；若整个映射为空白，则也可使用 `InitializeIfBlank`

请勿选择系统设备或 `Unknown` 设备、多路径成员、只读或可移动设备，或具有意外分区表、文件系统、LVM、RAID、LUKS、swap、使用者或挂载的设备。请勿同时选择整块磁盘及其分区。

步骤 2：定义受管卷

创建存储声明。此示例有意使用占位符；请将每个设备 ID、UUID、容量和挂载路径替换为同一目标清单中的实际信息。移除不适用于该主机的卷。

baremetal-storage.yaml

```yaml
storage:
  retryNonce: 0
  volumes:
    - name: cpaas-data
      source:
        deviceID: <blank-direct-disk-id>
        minimumSize: 100Gi
      filesystem:
        policy: InitializeIfBlank
        type: xfs
      mount:
        path: /var/cpaas
        required: true
        options:
          - noatime

    - name: application-data
      source:
        deviceID: <existing-partition-id>
        minimumSize: 100Gi
      filesystem:
        policy: Adopt
        type: ext4
        expectedUUID: <existing-filesystem-uuid>
      mount:
        path: /data/application
        required: true

    - name: registry-data
      source:
        deviceID: <multipath-map-id>
        multipath:
          minimumActivePaths: 2
      filesystem:
        policy: Adopt
        type: xfs
        expectedUUID: <existing-filesystem-uuid>
      mount:
        path: /srv/registry
        required: true
        options:
          - noatime
```

Storage v2 每个清单最多支持 32 个卷。第一个版本支持整块设备上的 XFS 或 ext4、通过 `Adopt` 管理的现有 XFS/ext4 分区，以及整块多路径映射。它不会创建分区，也不管理 LVM、mdraid、LUKS、swap、共享集群文件系统或多主机读写访问。

`required` 默认为 `true`。可选卷仍会经过准备和激活；只有其失败时对 Ready 状态的影响会有所不同。`/var/cpaas` 必须始终为必需项。provider 会将 `nodev,nosuid` 添加到受管挂载中，并根据允许列表验证每个额外挂载选项。

在本地验证声明：

```
"$BM_STORAGECTL" hash --storage "$BM_STORAGE_FILE"
```

步骤 3：生成并应用原子补丁

始终从当前的实时对象开始，以确保补丁包含最新的 UID 和 `resourceVersion`：

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  >"$BM_INVENTORY_FILE"
```

对于包含 `InitializeIfBlank` 的声明，请选择一个大于 `status.storage.lastConsumedInitializationApprovalCounter` 的正计数器，并生成补丁：

```
jq '.status.storage.lastConsumedInitializationApprovalCounter // 0' \
  "$BM_INVENTORY_FILE"

"$BM_STORAGECTL" render-patch \
  --inventory "$BM_INVENTORY_FILE" \
  --storage "$BM_STORAGE_FILE" \
  --counter <next-initialization-counter> \
  >"$BM_PATCH_FILE"
```

生成的合并补丁会将批准绑定到当前 Inventory UID、规范化存储哈希、计数器和 `resourceVersion`。对于仅包含 `Adopt` 的声明，请省略 `--counter`。

运行服务器端 dry-run，再次检查生成的 `spec.storage`、设备 ID、路径和批准，然后再应用补丁：

```
kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE" --dry-run=server -o yaml

kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE"
```

如果 API 报告 `resourceVersion` 冲突，请重新获取清单、重新生成补丁、重复服务器端 dry-run，然后再应用。切勿从生成的补丁中移除 `resourceVersion`。

步骤 4：等待 Prepare 完成

监视存储状态：

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -w

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  | jq '.status.storage | {
      phase,appliedSpecHash,pendingSpecHash,
      lastConsumedInitializationApprovalCounter,conditions,appliedVolume
s,operation}'
```

在清单可以分配之前，预期结果如下：

```
phase=Prepared
StoragePrepared=True/AllRequiredVolumesPrepared
StorageActive=False/Inactive
operation=null
```

此时，已初始化的卷具有文件系统 UUID，而已接管的卷保留其原始 UUID，但业务挂载路径尚未激活。在 Prepare 完成之前，请勿将清单添加到生产 `MachineInventoryPool`。

步骤 5：分配并验证激活

将已准备的清单添加到目标 `MachineInventoryPool`，然后按照在裸金属上创建集群和在裸金属上管理节点中的说明创建或扩展 CAPI machine group。

重新置备完成后，验证 machine 和存储状态：

```
kubectl -n "$BM_NS" get baremetalmachines.infrastructure.cluster.x-k8s.io
-o json \
  | jq -r --arg inventory "$BM_INVENTORY" \
      '.items[] | select(.status.machineInventoryRef.name==$inventory) |
       [.metadata.name,.status.phase,.status.ready,
        ([.status.conditions[]|select(.type=="StorageReady")|(.status
+"/"+.reason)]|join(","))] | @tsv'

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  | jq '.status.storage | {phase,conditions,appliedVolumes,operation}'
```

预期结果为 `BaremetalMachine.status.phase=Running`、`StorageReady=True/AllVolumesReady`、`MachineInventory.status.storage.phase=Active` 和 `StorageActive=True/AllRequiredVolumesActive`。

在主机上，确认每个源 UUID、文件系统类型、挂载路径和挂载选项。所需的挂载单元必须在 kubelet 启动之前进入 active 状态：

```
sudo findmnt --mountpoint /var/cpaas
sudo systemctl is-active "$(systemd-escape --path --suffix=mount /var/cpa
as)"
sudo systemctl show "$(systemd-escape --path --suffix=mount /var/cpaas)"
\
  -p ActiveEnterTimestamp
sudo systemctl show kubelet.service -p ActiveEnterTimestamp
```

步骤 6：缩减并重新使用清单

通过正常的缩减或替换工作流删除拥有该资源的 CAPI `Machine`。不要删除 `MachineInventory`，也不要移除其存储 finalizer。

完成 drain、清理和 Deactivate 后，预期结果如下：

```
allocation-state=Available
status.storage.phase=Prepared
StoragePrepared=True
StorageActive=False/Inactive
business mount paths are inactive
filesystem UUIDs and data are retained
```

再次分配同一清单时，Activate 会重新使用现有的所有权记录和文件系统 UUID；不会再次格式化设备。通用池分配无法保证替换后的 Machine 获得相同的清单。如果应用程序必须返回到特定清单上的主机本地数据，请将该清单置于受控的有序池的首位，或使用专用池，并在依赖这些数据之前验证所选的 `BaremetalMachine.status.machineInventoryRef`。

步骤 7：释放存储所有权

只有在清单未分配且处于非活动状态时，才能释放卷。移除卷不会破坏数据：它会移除 Storage v2 所有权和激活元数据，但会保留文件系统 UUID 和数据。

要释放所有卷，请创建一个显式的空声明：

baremetal-storage-empty.yaml

```yaml
storage:
  volumes: []
```

获取实时清单，并渲染、执行 dry-run，然后应用不带初始化计数器的补丁：

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  >"$BM_INVENTORY_FILE"

"$BM_STORAGECTL" render-patch \
  --inventory "$BM_INVENTORY_FILE" \
  --storage /tmp/baremetal-storage-empty.yaml \
  >"$BM_PATCH_FILE"

kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE" --dry-run=server -o yaml
kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE"
```

等待 `phase=Unmanaged`、`StoragePrepared=True/NoManagedVolumes`、空的 `appliedVolumes[]` 和 `operation=null`。如有需要，使用只读挂载验证保留的数据。实时清单上可能仍会保留存储 finalizer，以便后续对象删除继续遵循正确的清理顺序；不要手动移除它。

失败的操作与重试

存储操作会采取故障关闭策略。缺失、过期、忙碌、容量过小、降级、从未知来源挂载，或不再匹配其声明 UUID 的必需设备，会阻止分配或 Machine 就绪。controller 永远不会自动选择替代磁盘。

在修正 `status.storage.operation.phase` 为 `Failed` 的操作的外部原因后，递增 `spec.storage.retryNonce`，同时保持 `volumes[]` 不变，并使用 `storagectl` 渲染新的补丁。`retryNonce` 会为同一逻辑操作创建另一次执行尝试。它不会授权首次格式化、绕过预检或允许使用不同的存储声明。

不要将重试用作未知或部分变更声明的恢复机制。当 `pendingSpecHash` 非空，或失败操作报告 `mutationState=DurableStateChanged` 或 `Unknown` 时，保留原始声明，并允许同一操作日志逐步收敛。在考虑任何紧急 finalizer 或所有权操作之前，请联系技术支持。

限制

- 当清单已分配或任何受管卷处于活动状态时，不能更改 `volumes[]`。
- 已准备卷的 `deviceID`、文件系统策略/类型或名称不能就地更改。先释放旧卷，等待操作完成，然后添加替代声明。
- Storage v2 不会在清单之间移动数据。清单所拥有的数据会保留在该清单标识的物理主机或 LUN 上。
- Multipath 支持仅管理聚合的 `wwid:` 映射。它不会配置 multipath、SAN zoning、LUN masking、fencing 或多主机读写文件系统。
- `/var/lib/kubelet`、`/var/lib/containerd`、`/var/lib/etcd`、`/etc`、`/usr`、`/boot` 以及其他系统路径均受保护。请使用不与受保护路径重叠的业务路径，例如 `/var/cpaas`、`/data/*`、`/srv/*` 或 `/mnt/*`。
- 不要同时通过 `MachineInventory.spec.storage` 和 kubeadm/cloud-init `disk_setup`、`fs_setup`、`mounts`、自定义 systemd 单元或主机脚本管理同一设备或路径。

相关文档

- 裸金属 Provider
- 在裸金属上创建集群
- 在裸金属上管理节点
- 在裸金属上升级 Kubernetes
- MachineInventory API

管理 Huawei DCS 上的节点

本文介绍如何使用 Cluster API `Machine` 资源管理工作节点。

> **INFO**
>
> 如需升级节点的 Kubernetes 版本，请参阅在 Huawei DCS 上升级 Kubernetes。

目录

- 前提条件
- 概述
- 节点管理操作
  - 扩缩工作节点
    - 添加工作节点
    - 移除工作节点
  - 更改 Machine 基础设施
  - 更新 Bootstrap 模板
- 使用 Web UI 管理节点池
  - 查看节点池
  - 添加工作节点池
  - 删除工作节点池
  - 查看 Conditions（仅限控制平面）
- 故障排查

新工作节点 `Machine` 一直处于 `Provisioning` 状态，且 DCS VM 一直处于 `creating` 状态

后续步骤

前提条件

> **WARNING**
>
> 重要前提条件
>
> - 执行节点操作前，必须先部署控制平面。有关设置说明，请参阅创建集群。
> - 确保您拥有 DCS 平台的适当访问权限和所需权限。
> - 要管理 `global` 集群本身的节点，必须先在 `global` 上安装 provider；请参阅管理 Alauda OS 集群上的节点。

> **INFO**
>
> 配置指南 使用本文档中的配置时：
>
> - 仅修改 `<>` 括号内的值
> - 将占位值替换为特定于您环境的设置
> - 除非明确要求，否则保留所有其他默认配置

概述

工作节点通过 Cluster API `Machine` 资源进行管理，从而以声明式和自动化的方式管理节点生命周期。一个工作节点池由以下四种资源构成：

1. **IP-**主机名池配置 - 工作节点的网络设置
2. **Machine** 模板设置 - VM 规格
3. **Bootstrap** 配置 - 节点初始化和加入设置

4. **MachineDeployment** - 节点创建和管理的编排

要使用 YAML 添加另一个工作节点池，请按照集群创建页面中的部署工作节点操作。

节点管理操作

本节介绍管理工作节点时的常见操作任务，包括扩缩容、Machine 基础设施变更以及 Bootstrap 模板更新。

> **INFO**
>
> **Cluster API** 框架 节点管理操作基于 Cluster API 框架。有关详细信息，请参阅 Cluster API 官方文档 ↗。

扩缩工作节点

工作节点扩缩容可根据工作负载需求调整集群容量。Cluster API 通过 MachineDeployment 资源自动管理节点生命周期。

添加工作节点

增加工作节点数量，以处理增加的工作负载或添加新容量。

使用场景：扩展集群以添加更多计算资源

前提条件：

- 确认 IP 池有足够的可用 IP 地址供新节点使用
- 确保 DCS 平台有足够的资源来创建新 VM

操作步骤：

1. 检查当前节点状态
   查看集群中的当前 Machine：

```
# List all machines in the cluster
kubectl get machines -n cpaas-system

# List machines for a specific MachineDeployment
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
e=<cluster-name>-md-0
```

2. 扩展 **IP** 池

   扩容前，将新的 IP 配置添加到池中，以供新增节点使用。

> **INFO**
>
> **IP** 池扩展 IP 池中的条目数必须至少等于所需的副本数。为计划部署的每个新增工作节点添加新的 IP 条目。

向池中添加 **IP** 条目：

首先，导出当前池配置以保留现有条目：

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

然后使用以下命令添加新的 IP 配置。`pool` 数组必须包含所有现有条目以及新条目：

```
kubectl patch dcsiphostnamepool <cluster-name>-worker-pool -n cpaas-sys
tem \
  --type='merge' -p='
{
  "spec": {
    "pool": [
      {
        "ip": "<existing-ip-1>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<existing-hostname-1>",
        "machineName": "<existing-machine-name-1>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<existing-ip-2>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<existing-hostname-2>",
        "machineName": "<existing-machine-name-2>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<new-worker-ip-1>",
        "mask": "<worker-cidr-prefix-length>",
```

```
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<new-worker-hostname-1>",
        "machineName": "<new-worker-machine-name-1>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<new-worker-ip-2>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<new-worker-hostname-2>",
        "machineName": "<new-worker-machine-name-2>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      }
    ]
  }'
```

WARNING

重要说明

- `pool` 数组必须包含所有现有条目以及要添加的新条目
- 从导出的 YAML 中复制现有条目，以避免数据丢失
- `kubectl patch --type='merge'` 会替换整个 `spec.pool` 数组，因此请原样复制每个现有的 `persistentDisk` 块，包括其中存在的 `isThin`，除非您确实要添加新磁盘

- 确保每个新条目具有唯一的 `ip`、`hostname` 和 `machineName` 值
- 如果新工作节点也需要平台要求的 `/var/cpaas` 磁盘，请在每个新条目的 `persistentDisk` 中声明该磁盘
- 网络参数（`mask`、`gateway`、`dns`）通常与现有条目一致

示例：向包含 3 个节点的现有池中添加 2 个新节点

```
# Current pool has 3 entries (10.0.1.11, 10.0.1.12, 10.0.1.13)
# Adding 2 more entries for nodes 4 and 5
kubectl patch dcsiphostnamepool worker-pool-1-ippool -n cpaas-system \
  --type='merge' -p='
{
  "spec": {
    "pool": [
      {
        "ip": "10.0.1.11",
        "mask": "24",
        "gateway": "10.0.1.1",
        "dns": "10.0.0.2",
        "hostname": "worker-node-1",
        "machineName": "worker-vm-1",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "shared-datastore-cluster",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "10.0.1.12",
        "mask": "24",
        "gateway": "10.0.1.1",
        "dns": "10.0.0.2",
        "hostname": "worker-node-2",
        "machineName": "worker-vm-2",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "shared-datastore-cluster",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "10.0.1.13",
```

```
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-3",
    "machineName": "worker-vm-3",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
        "path": "/var/cpaas",
        "format": "xfs"
      }
    ]
  },
  {
    "ip": "10.0.1.14",
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-4",
    "machineName": "worker-vm-4",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
        "path": "/var/cpaas",
        "format": "xfs"
      }
    ]
  },
  {
    "ip": "10.0.1.15",
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-5",
    "machineName": "worker-vm-5",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
```

```
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      }
    ]
  }'
```

3. 验证 **IP** 池容量

   扩展 IP 池后，验证其条目数足以满足所需的副本数：

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

   检查池中的条目数是否至少等于所需的副本数。

4. 扩展 **MachineDeployment**

   将 `replicas` 字段更新为所需的节点数：

   ```
   kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": <new-replica-count>}]'
   ```

   示例：从 3 个节点扩展到 5 个节点

   ```
   kubectl patch machinedeployment worker-pool-1 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": 5}]'
   ```

5. 监控扩容进度

   查看 Machine 创建过程：

   ```
   # Watch machines being created
   kubectl get machines -n cpaas-system -w

   # Check MachineDeployment status
   kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
   ```

   Cluster API 控制器将根据 MachineDeployment 模板自动创建新的 Machine。

6. 验证节点已加入集群

   切换到目标集群上下文并验证新节点：

```
# Switch to target cluster context
kubectl config use-context <target-cluster-context>

# Check all nodes are Ready
kubectl get nodes
```

   新节点应出现在列表中，并转换为 `Ready` 状态。

> **INFO**
>
> 滚动更新行为 扩容时，新节点会立即创建，不会影响现有节点，从而确保扩容期间零停机。

移除工作节点

减少工作节点数量，以降低集群容量或移除未充分利用的资源。Cluster API 支持两种移除策略：

1. 随机移除：减少副本数，由平台随机选择并删除 Machine
2. 定向移除：标记要删除的特定 Machine，然后减少副本数（推荐用于 IP 回收）

> **INFO**
>
> **IP** 回收场景 需要回收特定 Machine 的 IP（例如重新分配或管理 IP 池）时，请使用定向移除方法。删除注解可确保平台删除已标记的 Machine，而不是随机删除 Machine。

> **WARNING**
>
> 数据丢失警告 缩容会移除节点及其关联的 VM。请确保：
>
> - 工作负载通过适当的副本机制能够容忍节点丢失
> - 要移除的节点上没有仅存储一份的关键数据
> - 应用程序设计为支持水平扩展

仅仅因为替换 Machine，不会删除 `DCSIpHostnamePool.spec.pool[].persistentDisk` 中声明的持久化磁盘。在相应 IP 槽位仍保留在池中时，这些磁盘仍可供重复使用。从池中移除 IP 槽位、删除池或删除集群，都可能触发持久卷清理。

随机移除

使用场景：缩小集群，任意节点均可移除（无特定 IP 要求）

操作步骤：

1. 确定当前 **Machine** 状态
   查看 MachineDeployment 中的当前 Machine：

   ```
   kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
   e=<cluster-name>-md-0
   ```

2. 缩小 **MachineDeployment**
   更新 `replicas` 字段以减少节点数：

   ```
   kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": <new-replica-count>}]'
   ```

   示例：从 5 个节点缩小到 3 个节点

   ```
   kubectl patch machinedeployment worker-pool-1 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": 3}]'
   ```

   Cluster API 控制器将随机选择并删除 Machine，使其达到所需的副本数。

3. 监控移除进度
   查看 Machine 删除过程：

   ```
   kubectl get machines -n cpaas-system -w
   ```

   Cluster API 控制器将：

- 排空选定的节点（如果可能则驱逐 Pod）
- 从 DCS 平台删除底层 VM
- 移除 Machine 资源

4. 验证节点已移除
   切换到目标集群上下文：

```
kubectl config use-context <target-cluster-context>
kubectl get nodes
```

   已移除的节点不应再出现在列表中。

定向移除

使用场景：移除特定 Machine（例如回收 IP、替换运行状况不佳的节点）

操作步骤：

1. 确定要移除的 **Machine**
   查看当前 Machine：

```
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-name=<cluster-name>-md-0
```

   记录要移除的 Machine 的 `<machine-name>`。

2. 为 **Machine** 添加删除注解
   标记要删除的特定 Machine：

```
kubectl patch machine <machine-name> -n cpaas-system \
  --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/delete-machine": "true"}}}'
```

   对要移除的每个 Machine 重复此操作。
   示例：移除两个特定的 Machine

```
kubectl patch machine worker-pool-1-abc123 -n cpaas-system \
  --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/de
lete-machine": "true"}}}'

kubectl patch machine worker-pool-1-def456 -n cpaas-system \
  --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/de
lete-machine": "true"}}}'
```

3. 缩小 **MachineDeployment**

   为 Machine 添加注解后，减少副本数：

> **INFO**
>
> 副本数必须与已添加注解的 **Machine** 数量一致 副本数必须恰好减少已添加注解的 Machine 数量。
>
> - 如果减少的数量较少，则不会移除所有已添加注解的 Machine
> - 如果减少的数量较多，则会随机选择其他 Machine 进行删除

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
e": <new-replica-count>}]'
```

   示例：如果为 2 个 Machine 添加了注解，则副本数必须恰好减少 2（例如从 5 减少到 3）平台将删除已添加注解的 Machine，而不是随机选择的 Machine。

4. 监控移除进度

   查看 Machine 删除过程：

```
kubectl get machines -n cpaas-system -w
```

5. 验证节点已移除

   切换到目标集群上下文：

```
kubectl config use-context <target-cluster-context>
kubectl get nodes
```

已移除的节点不应再出现在列表中。

更改 Machine 基础设施

要更改工作节点的 Machine 规格（CPU、内存、磁盘、VM 模板），请创建新的 Machine 模板，然后将 `MachineDeployment` 指向该模板。工作节点会逐个替换。

1. 创建新的 **Machine** 模板

   将 `MachineDeployment` 引用的 `DCSMachineTemplate` 导出到新文件：

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

   在新文件中：

   - 将 `metadata.name` 设置为唯一名称
   - 修改所需的值（`spec.template.spec.vmTemplateName`、`spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`、`spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`、`spec.template.spec.vmConfig.dcsMachineDiskSpec`）
   - 移除服务器生成的元数据（`resourceVersion`、`uid`、`generation`、`creationTimestamp`、`managedFields`、`kubectl.kubernetes.io/last-applied-configuration` 注解）以及整个 `status` 字段
   - 不设置 `spec.template.spec.providerID`

   将池管理的持久化磁盘（包括 `/var/cpaas`）保留在 `DCSIpHostnamePool.spec.pool[].persistentDisk` 中，并将其他 NIC 保留在 `DCSIpHostnamePool.spec.pool[].additionNic[]` 中。仅将 `dcsMachineDiskSpec` 用于系统磁盘和模板本地磁盘。

   应用新模板：

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

2. 更新 **MachineDeployment**

   将 `MachineDeployment` 指向新模板：

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

3. 监控滚动更新

- 工作节点将使用新规格逐个替换
- `DCSIpHostnamePool.spec.pool[].persistentDisk` 中声明的任何磁盘都会从旧 VM 分离，并重新连接到替换 VM。如果池使用持久化磁盘，请在 `MachineDeployment` 上将 `spec.strategy.rollingUpdate.maxSurge` 保持为 `0`
- 监控进度：

```
kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
kubectl get machines -n cpaas-system -w
```

要更改控制平面节点的 Machine 规格，请使用相同的步骤，并改为修补 `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name`。

要将节点迁移到新 Kubernetes 版本的 Alauda OS 镜像，请改为按照在 Huawei DCS 上升级 Kubernetes 操作。

如果要将现有集群从旧的模板磁盘布局迁移到池管理的持久化磁盘，请在依赖升级期间的数据保留功能之前，按照将现有 Huawei DCS 集群迁移到池管理的持久化磁盘操作。

更新 Bootstrap 模板

Bootstrap 模板（KubeadmConfigTemplate）由 MachineDeployment 和 MachineSet 资源使用。更改现有模板不会自动触发现有 Machine 的滚动更新；只有新 Machine 会使用更新后的模板。

更新流程：

1. 导出现有模板

```
kubectl get KubeadmConfigTemplate <template-name> -o yaml > new-templat
e.yaml
```

2. 修改配置

- 更新导出 YAML 中所需的字段
- 对于 Kubernetes 1.35 或更高版本，将所需的 kubelet 修补设置添加到 `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`
- 将 `metadata.name` 更改为新的唯一名称
- 移除多余的元数据字段（`resourceVersion`、`uid`、`creationTimestamp` 等）

3. 创建新模板

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

4. 更新 **MachineDeployment**

- 修改 MachineDeployment 资源
- 更新 `spec.template.spec.bootstrap.configRef.name` 以引用新模板
- 应用更改以触发滚动更新

> **INFO**
>
> 模板滚动更新行为 现有 Machine 继续使用旧的 Bootstrap 配置。只有新创建的 Machine（在扩缩容或滚动更新期间）会使用更新后的模板。

使用 **Web UI** 管理节点池

通过 Fleet Essentials 添加、删除和查看节点池不受《Fleet Essentials 限制》中所述升级限制的影响。

节点池提供了一种声明式方式，用于管理具有相同配置的节点组。您可以通过 Web UI 查看、添加和删除工作节点池。

要求：此操作流程需要 Fleet Essentials。`persistentDisk` 上的 `DCSIpHostnamePool` 声明仍仅支持 YAML，未在节点池 UI 中提供。

如果节点池依赖池管理的持久磁盘，请在使用此处的 Web UI 操作流程之前，使用 YAML 准备或更新相应的 `DCSIpHostnamePool` 条目。

> **INFO**
>
> 导航：集群 → 集群 → 选择集群 → 节点池选项卡

查看节点池

节点池选项卡显示集群中的所有节点池：

控制平面节点池：

- 固定为 3 个从节点，以实现高可用性
- 显示 Kubernetes 版本；如果有可用升级，则显示升级指示器
- 显示“Conditions”链接以查看详细状态

工作节点池：

- 可自定义从节点数量
- 可单独管理 Kubernetes 版本
- 支持扩缩容和升级操作

节点池卡片信息：

字段	描述
类型	控制平面 / 工作节点
名称	资源名称
状态	显示节点池运行状况的徽章
从节点数量	当前数量（工作节点显示 Max Surge/Max Unavailable）
SSH 授权密钥	SSH 公钥列表

字段	描述
Kubernetes 版本	当前版本（如果有可用升级，则显示升级指示器）
Machine 模板	关联的模板名称
Conditions	Conditions 列表链接（仅限控制平面）

添加工作节点池

导航：节点池选项卡 → 单击添加工作节点池

表单字段：

字段	类型	必填	描述
池名称	文本	是	节点池的唯一标识符
Machine 模板	下拉列表	是	按类型“工作节点”和兼容的 Kubernetes 版本进行筛选
从节点数量	数字	是	节点池中的节点数量
Max Surge	数字	否	默认值：0，必须 ≥ 0。如果节点池依赖持久磁盘，请将此值保持为 `0`
Max Unavailable	数字	否	默认值：1，必须 ≥ 0。当 maxSurge = 0 时，必须 > 0 且 ≤ 从节点数量
SSH 授权密钥	文本	否	添加多个 SSH 公钥

验证：

- 池名称在集群内必须唯一
- IP Pool 必须有足够的可用 IP 地址（≥ 从节点数量）
- 必须满足 maxSurge/maxUnavailable 约束
- 如果节点池依赖持久磁盘，请保留 `maxSurge = 0`，以便逐个替换 Machine

提示：将集群名称后接连字符作为池名称前缀（例如 `mycluster-worker-1`），以避免名称冲突。

创建后，新节点会显示在节点选项卡中。节点数量等于配置的从节点数量值。

删除工作节点池

操作步骤：

1. 单击工作节点池卡片上的删除图标
2. 在对话框中确认删除

> **WARNING**
>
> 删除工作节点池会永久移除所有关联的节点和 Machine。请通过适当的副本机制，确保工作负载能够承受这些节点的丢失。

查看 **Conditions**（仅限控制平面）

单击控制平面节点池卡片上的**Conditions**链接，查看详细状态信息。

Conditions 列表：

类型	状态	上次转换时间	原因	消息
Condition 类型	状态	时间戳	原因	人类可读的详细信息

故障排查

新工作节点 `Machine` 一直处于 `Provisioning` 状态，且 **DCS VM** 一直处于 `creating` 状态

控制平面和工作节点 `Machine` 对象的诊断流程相同：当 `DCSMachine` 达到 `Provisioning`，且已设置 `status.cdRomFile`（DCS API 身份验证和 ISO 上传均已成功），但底层 DCS VM 始终无法调度到主机时，故障出在 DCS 平台侧，而非控制器。

有关完整的诊断步骤，请参阅基础设施 → 故障排查：VM 卡在 `creating`，其中包括：

- 确认问题是放置停滞（而非启动缓慢）。
- 检查数据存储的可用空间。
- 检查主机 CPU / 内存利用率和超额分配比率。
- 当单个 DCS Host 负载饱和时，与 DCS 平台团队协调以重新平衡负载。
- 部署任务确实陷入死锁时的升级处理路径。

有关跨多个物理主机实现控制平面高可用性的信息，请参阅基础设施 → 控制平面跨主机高可用性。

无论停滞的节点是新集群的第一个控制平面节点、通过 `MachineDeployment` 扩容添加的新工作节点，还是滚动更新期间创建的替换工作节点，诊断方法都相同。

后续步骤

- 创建集群
- 升级集群

在 Huawei Cloud Stack 上管理节点

本文介绍如何使用 Cluster API `Machine` 资源在 Huawei Cloud Stack 平台上管理工作节点。

INFO

要升级节点的 Kubernetes 版本，请参阅在 Huawei Cloud Stack 上升级 Kubernetes。

目录

- 先决条件
- 概述
- 使用 Web UI
  - 查看节点池
  - 添加工作节点池
  - 管理节点 IP 地址
  - 删除工作节点池
- 使用 YAML
  - 扩缩工作节点
    - 添加工作节点
    - 移除工作节点
  - 更改机器基础设施
  - 更新引导模板
- 验证

故障排除

查看控制器日志

常见问题

先决条件

WARNING

重要先决条件

- 执行节点操作前，必须先部署控制平面。有关设置说明，请参阅创建集群。
- 确保您拥有 HCS 平台的适当访问权限和所需权限。
- 要管理 `global` 集群本身的节点，必须先在 `global` 上安装 provider；请参阅管理 Alauda OS 集群上的节点。

使用本文档中的 YAML 示例时，仅将 `<>` 中的值替换为特定于环境的值。除非集群策略要求使用其他值，否则请保留其余字段。

概述

工作节点通过 Cluster API `Machine` 资源进行管理，从而以声明式和自动化的方式管理节点生命周期。一个工作节点池由以下四个资源构成：

1. **Machine Configuration Pool** - 工作节点的网络设置
2. **Machine Template** - VM 规格
3. **Bootstrap Configuration** - 节点初始化设置
4. **Machine Deployment** - 节点创建和管理的编排

您可以通过 Web UI 或 YAML 清单管理这些资源。Web UI 提供引导式体验，并且需要 Fleet Essentials；无需 Fleet Essentials 即可使用 YAML。

使用 **Web UI**

要求：此工作流需要 Fleet Essentials `1.0.2` 或更高版本。没有该功能时，请使用使用 YAML 部分中的 YAML 工作流管理节点池。

Node Pools 选项卡可用于查看控制平面和工作节点池、添加或删除工作节点池，以及编辑现有节点池的预留 IP 地址。

> **INFO**
>
> 导航：Clusters → Clusters → 选择集群 → Node Pools 选项卡

查看节点池

Node Pools 选项卡以卡片形式显示 Control Plane Node Pool 和每个 Worker Node Pool。两种卡片类型包含相同的字段：

字段	描述
Name	资源名称。链接到基础 `KubeadmControlPlane`（控制平面）或 `MachineDeployment`（工作节点）。控制平面卡片带有 **Control Plane** 标签。
Status	显示节点池运行状况的徽章。
Conditions	状态条件的数量；单击可查看条件列表。
Ready Replicas	已就绪节点数与期望节点数之比（例如 `3 / 3`）。
Kubernetes Version	节点池当前的 Kubernetes 版本。
Machine Template	关联的 `HCSMachineTemplate`；单击可打开。
Config Pool	关联的 `HCSMachineConfigPool`；单击可打开。

字段	描述
SSH Authorized Keys	已配置公钥的数量。
Update Strategy	对于控制平面，节点会逐个替换，并且每个节点都会保留其节点 IP。对于工作节点，则根据节点池的 Max Surge 和 Max Unavailable 值执行滚动更新。

添加工作节点池

单击 **Add Worker Node Pool** 并完成对话框。字段与创建向导的第 5 步相同：节点池名称（带有 `<cluster-name>-` 前缀）、Flavor、Availability Zone、Machine Configs（主机名、网络、持久磁盘）、System Disk、Data Volumes、Replicas、Rollout Strategy 和 SSH Authorized Keys。节点池创建后，新节点会显示在 Nodes 选项卡中。

所选 Availability Zone 适用于新工作节点池中的每个节点。要将工作节点放置在其他区域，请创建单独的工作节点池并选择该区域；一个节点池不会自动将其副本分散到多个区域。下拉列表会禁用 HCS 报告为不可用的区域。有关自动选择、持久磁盘和控制平面放置限制，请参阅 Availability Zone 放置。

管理节点 IP 地址

HCS 节点池中的每个节点都使用其子网中的一个预留静态 IP 地址。要更改节点池预留的地址（例如，在扩容节点池之前增加容量），请打开节点池卡片的操作菜单，然后选择 **Update Config Pool** 以打开 **Manage IPs** 对话框。

此对话框用于编辑节点池的 `HCSMachineConfigPool` 的机器配置。对于每个节点，您可以设置主机名、网络（每个网络包含一个从其子网中选择的 IP 地址以及该子网）和持久磁盘。IP 选择遵循与创建向导相同的规则：先选择子网，然后从列表中选择可分配的地址；已在使用的地址不会显示。

> **WARNING**
>
> HCS 集群使用静态 IP 地址，不使用 DHCP。在扩容节点池之前，请在节点池中预留足够的地址——除非节点池为新节点提供地址，否则新节点无法启动。

删除工作节点池

打开 Worker Node Pool 卡片的操作菜单，选择 **Delete**，然后确认。

> **WARNING**
>
> 删除工作节点池会永久删除其中的所有节点以及底层 ECS 实例。请确保工作负载通过适当的副本机制能够承受这些节点的丢失。Control Plane Node Pool 无法删除。

使用 YAML

使用 Cluster API 清单管理工作节点。要使用 YAML 添加另一个工作节点池，请在集群创建页面按照部署工作节点中的步骤操作。

扩缩工作节点

工作节点扩缩允许您根据工作负载需求调整集群容量。

添加工作节点

增加工作节点数量以处理增加的工作负载。

操作步骤：

1. 检查当前节点状态

```
# List all machines in the cluster
kubectl get machines -n cpaas-system

# List machines for a specific MachineDeployment
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-name=<cluster-name>-md-0
```

2. 扩展配置池
   为其他节点向池中添加新的机器配置。如果新工作节点需要保留节点本地状态（例如 `/var/cpaas`），请在每个新配置中包含匹配的 `persistentDisks[]` 条目。

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

修改节点池以包含新的 IP 条目，然后应用更改：

```
kubectl apply -f <updated-pool-config.yaml>
```

编辑节点池时，请保持所有现有的 `configs[]` 条目及其已接受的 `persistentDisks[]` 条目不变，除非您有意追加新的磁盘插槽。

3. 扩容 **MachineDeployment**
   将 `replicas` 字段更新为所需的节点数：

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "value": <new-replica-count>}]'
```

4. 监控扩容进度

```
# Watch machines being created
kubectl get machines -n cpaas-system -w

# Check MachineDeployment status
kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
```

移除工作节点

减少工作节点数量以降低集群容量。

> **WARNING**
>
> 数据丢失警告
>
> 缩容会移除工作节点及其 ECS 实例。模板拥有的 `dataVolumes[]` 不会保留。在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中声明的、由节点池管理的持久磁盘仍由节点池跟踪；只要相应的主机名条目仍保留在节点池中，这些磁盘就可以重复使用。请确保：

- 工作负载能够通过适当的副本机制承受节点丢失
- 未将关键数据仅存储在要移除的节点上
- 应用程序按照水平扩展方式设计

操作步骤：

1. 缩容 **MachineDeployment**

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
e": <new-replica-count>}]'
```

2. 监控移除进度

```
kubectl get machines -n cpaas-system -w
```

Cluster API 控制器将：

- 排空选定的节点（如果可能则驱逐 Pod）
- 从 HCS 平台删除底层 VM
- 移除机器资源

更改机器基础设施

要更改工作节点机器规格（CPU、内存、磁盘和 VM 镜像），请创建新的机器模板，并将 `MachineDeployment` 指向该模板。随后，工作节点会逐个替换。请按照以下步骤操作：

注意：HCS `dataVolumes[]` 在替换期间不会保留。要保留节点本地状态（例如 `/var/cpaas`），请在发布更新前将其声明在 `HCSMachineConfigPool.spec.configs[].persistentDisks[]` 中，并保留 `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0`。如果您的集群仍在 `HCSMachineTemplate.spec.template.spec.dataVolumes[]` 中保留这些磁盘，请先完成将现有 Huawei Cloud Stack 集群迁移到由节点池管理的持久磁盘。

1. 创建新的机器模板

复制现有的 `HCSMachineTemplate` 并修改所需的值：

- `imageName` - VM 镜像
- `flavorName` - 实例类型
- `rootVolume.size` - 系统磁盘大小
- `dataVolumes` - 临时数据磁盘配置

如果需要添加新的、由节点池管理的持久磁盘，请先将其追加到工作节点的 `HCSMachineConfigPool` 中。Provider 会创建或声明 EVS 磁盘，但运行中的 ECS 不会挂载该磁盘，直到滚动替换创建出替代工作节点。

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

然后在应用前编辑 `new-template.yaml`：

- 将 `metadata.name` 更改为 `<new-template>`
- 不设置运行时身份字段，包括 `spec.template.spec.providerID` 和 `spec.template.spec.serverId`
- 移除服务器生成的字段，例如：
  - `metadata.resourceVersion`
  - `metadata.uid`
  - `metadata.creationTimestamp`
  - `metadata.managedFields`
  - `status`

2. 部署新模板

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

3. 更新 **MachineDeployment**
   修改 `MachineDeployment` 以引用新模板：

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

4. 监控滚动更新

```
kubectl get machines -n cpaas-system -w
```

要更改控制平面节点的机器规格，请执行相同的步骤，并改为修补 `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name`。

更新引导模板

要更改工作节点的引导配置（例如 kubelet 设置或节点加入时运行的命令），请创建新的 `KubeadmConfigTemplate`，并将 `MachineDeployment` 指向该模板。就地修改现有模板不会发布到现有机器；只有新创建的机器会应用此更改。

1. 导出现有模板

```
kubectl get kubeadmconfigtemplate <cluster-name>-worker-kct -n cpaas-sy
stem -o yaml > new-bootstrap-template.yaml
```

2. 修改模板
   编辑 `new-bootstrap-template.yaml`：

   - 将 `metadata.name` 设置为新的唯一名称 `<new-bootstrap-template>`
   - 移除服务器生成的元数据（`resourceVersion`、`uid`、`creationTimestamp`、`managedFields`、`kubectl.kubernetes.io/last-applied-configuration` 注解）以及整个 `status` 字段
   - 编辑要更改的字段。保留 kubelet 修补文件 `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` 的现有条目（包括证书路径），仅更改所需内容
   - 对于 Kubernetes 1.35 或更高版本，将所需的 kubelet 修补设置添加到该文件中

3. 应用新模板

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

4. 更新 **MachineDeployment**

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

随后开始滚动替换工作节点：

```
kubectl get machines -n cpaas-system -w
```

验证

部署工作节点后，验证部署：

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

# Verify nodes are Ready
kubectl get nodes

# Check MachineDeployment status
kubectl get machinedeployment -n cpaas-system
```

故障排除

查看控制器日志

```
# View HCS controller logs
kubectl logs -n cpaas-system deployment/hcs-controller-manager

# View machine details
kubectl describe hcsmachine <machine-name> -n cpaas-system
```

常见问题

节点无法加入集群

- 验证 VM 模板与 Kubernetes 版本匹配
- 检查节点之间的网络连接
- 确保配置池中有可用条目

Machine 卡在预配阶段

- 检查 HCS 平台是否有可用资源
- 验证凭据和权限
- 查看控制器日志中的错误消息

在现有 HCS 集群上配置 FQDN 主机名

当现有 Huawei Cloud Stack (HCS) 集群需要其节点的 POSIX `hostname` 返回短标签，而 `hostname -f` 返回完整 FQDN，且该集群最初创建于 `cluster-api-provider-hcs` v1.0.1 之前时，请使用本指南。

依赖正常工作的 `hostname` / `hostname -f` 拆分的应用——例如 TLS 服务器证书 SAN 匹配、Kerberos SPN 组成、以短名称为键的监控标签，以及调用 `getfqdn()` 的任何代码——要求节点由 `cluster-api-provider-hcs` v1.0.1 或更高版本初始化。Cloud-init 仅在首次启动时运行，因此现有节点不会自动获取新行为；必须在新控制器就位后，通过标准的 Cluster API 滚动替换流程替换节点。

> **INFO**
>
> 版本
>
> 此版本随附的 Huawei Cloud Stack Infrastructure Provider 能够正确呈现带点的主机名。早于 `cluster-api-provider-hcs` v1.0.1 的控制器会将带点的 `HCSMachineConfigPool.spec.configs[].hostname` 值呈现为 POSIX 主机名中的完整 FQDN 字符串，这会破坏 `hostname -f`，还可能阻止 kubeadm init。

目录

概述

前提条件

操作步骤

1. 确认控制器已升级

2. 确定涉及的池和模板
3. 为控制平面池扩展一个扩容槽位
4. 为控制平面创建新的 `HCSMachineTemplate`
5. 修补 `KubeadmControlPlane` 以引用新模板
6. 对每个 `MachineDeployment` 重复上述操作（工作节点）
7. 验证每个新节点上的主机名行为

存储和数据注意事项

回滚

相关内容

概述

节点要呈现受支持的主机名 / FQDN 行为，以下两个条件必须同时满足：

1. 节点的 `HCSMachineConfigPool.spec.configs[].hostname` 条目包含点（FQDN 格式，例如 `master-1.example.org`）。
2. 节点首次由 `cluster-api-provider-hcs` v1.0.1 或更高版本启动。

在旧控制器下启动的现有节点会继续使用其首次启动时接收的 cloud-init 用户数据，因此编辑清单或重新应用用户数据不会追溯性地更改其主机名行为。受支持的迁移路径是通过 Cluster API 滚动替换来替换这些节点，并确保升级后的控制器已安装在管理集群上且处于 Ready 状态。

在节点上手动编辑 `/etc/hostname` 和 `/etc/hosts` 不会在 Alauda OS 节点上持久化。Cloud-init 会在每次启动时重新呈现这些文件，包括由 `transactional-update`、OS 修补或 `systemctl reboot` 触发的重启。

前提条件

- 管理集群上的 Huawei Cloud Stack Infrastructure Provider 为此版本随附的版本（`cluster-api-provider-hcs` v1.0.1 或更高版本），且 `cpaas-system` 中的控制器 Deployment 处于 Ready 状态。

- 现有业务集群运行正常：所有当前的 `Machine` 对象均报告 `phase=Running`，所有 `Node` 对象均报告 `Ready`，且当前没有正在进行的滚动替换。
- 已检查存储注意事项（参见存储和数据注意事项）。
- 现有 `HCSMachineTemplate` 使用的目标 VM 镜像仍存在于 HCS 环境中。（无需更改镜像；此次迁移仅重命名模板以触发替换。）
- 控制平面 `HCSMachineConfigPool` 至少有一个未使用的 `(hostname, IP)` 槽位，以便 Cluster API 在排空旧控制平面机器之前，可以先扩容出一个新的控制平面机器。如果现有池的大小恰好等于当前副本数，请在开始前为其再添加一个条目。

操作步骤

1. 确认控制器已升级

```
kubectl get deployment -n cpaas-system -l app=cluster-api-provider-hcs-manager -o wide
kubectl rollout status deployment -n cpaas-system cluster-api-provider-hcs-manager
```

Deployment 必须显示 v1.0.1 或更高版本的镜像标签，并报告所有副本均处于 Ready 状态。在控制器升级之前替换节点，会在新 VM 上重新呈现旧的用户数据，使集群保持原有状态。

2. 确定涉及的池和模板

仅包含带点条目的池需要迁移。列出现有的 `HCSMachineConfigPool` 资源，并检查每个 `spec.configs[].hostname`：

```
kubectl get hcsmachineconfigpool -n cpaas-system
kubectl get hcsmachineconfigpool <pool-name> -n cpaas-system \
  -o jsonpath='{range .spec.configs[*]}{.hostname}{"\n"}{end}'
```

条目全部为短标签（不含点）的池不受此次迁移影响，无需进行处理。仅继续处理包含带点条目的池。

3. 为控制平面池扩展一个扩容槽位

Cluster API 一次替换一个控制平面机器。默认滚动更新策略需要池中有一个空闲的 `(hostname, IP)` 槽位，以便在终止旧机器之前启动替换机器。如果现有控制平面池的条目数恰好等于 `KubeadmControlPlane.spec.replicas`，请立即再添加一个条目：

```
kubectl edit hcsmachineconfigpool <cp-pool-name> -n cpaas-system
```

在 `spec.configs` 下添加一个额外条目，为其设置新的主机名以及来自同一子网的空闲 IP，然后保存。新条目会作为备用项保留，仅在替换第一台机器时使用。

4. 为控制平面创建新的 `HCSMachineTemplate`

新模板可以是现有模板的逐字节副本，只需使用新的 `metadata.name`。仅重命名就会触发 Cluster API 滚动机器并获取新的用户数据；无需更改 `imageName`、`flavorName` 或任何其他字段。

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

编辑 `new-cp-template.yaml`：

- 将 `metadata.name` 设置为新值（例如追加 `-fqdn` 或日期后缀）。
- 删除服务器生成的元数据（`resourceVersion`、`uid`、`generation`、`creationTimestamp`、`managedFields`、`kubectl.kubernetes.io/last-applied-configuration` 注解）以及整个 `status` 字段。
- 保持运行时身份字段未设置，包括 `spec.template.spec.providerID` 和 `spec.template.spec.serverId`。HCS provider 会在创建新 VM 时分配这些值。

应用新模板：

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

在滚动更新确认正常之前，请保留旧模板；回滚时需要使用它。

5. 修补 `KubeadmControlPlane` 以引用新模板

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='merg
e' \
  -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<new-cp-te
mplate>"}}}}'
```

Cluster API 现在会开始滚动替换控制平面。监控进度：

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

等待每个控制平面 `Machine` 都变为新模板（旧的 `Machine` 名称消失），并且 `KubeadmControlPlane.status.readyReplicas` 等于 `spec.replicas`。

6. 对每个 `MachineDeployment` 重复上述操作（工作节点）

对于节点需要采用新主机名行为的每个 `MachineDeployment`，使用工作节点模板重复步骤 4 和 5：

```
kubectl get hcsmachinetemplate <current-worker-template> -n cpaas-system
-o yaml > new-worker-template.yaml
# Edit metadata.name, strip status/server fields, apply
kubectl apply -f new-worker-template.yaml -n cpaas-system

kubectl patch machinedeployment <md-name> -n cpaas-system --type='merge'
\
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<new-work
er-template>"}}}}}'
```

等待每个 `MachineDeployment` 报告 `status.updatedReplicas` 等于 `spec.replicas`，并且所有工作节点 `Machine` 对象均使用新模板后，再继续操作。

7. 验证每个新节点上的主机名行为

在每个新节点上：

```
hostname
hostname -f
cat /etc/hosts
```

对于池配置为 `hostname: master-1.example.org` 的节点，应满足以下预期：

- `hostname` 返回 `master-1`（短标签，去除点及其后的所有内容）。
- `hostname -f` 返回 `master-1.example.org`。
- `/etc/hosts` 包含形如 `<loopback-or-node-ip> master-1.example.org master-1` 的条目。

如果新节点报告来自 `hostname` 的 FQDN 风格字符串，或者 `hostname -f` 返回 `Name or service not known`，首先应检查节点上的 cloud-init 日志（`/var/log/cloud-init.log` 和 `/var/lib/cloud/instance/user-data.txt`），然后确认控制器确实为 v1.0.1 或更高版本。

存储和数据注意事项

滚动替换会销毁每个旧 VM，并根据新模板创建一个全新的替代 VM。在 HCS 上：

- 每次替换时，系统磁盘（根卷）都会根据 VM 镜像重新构建。
- `HCSMachineTemplate.spec.template.spec.dataVolumes` 中声明的数据卷不会先分离再重新挂载；旧 VM 上的卷可能会随旧 VM 一起被删除。
- 工作负载数据必须存储在外部持久化存储上（HCS EVS CSI 或同等存储），才能在迁移后保留。

开始操作前，请完成所有节点本地状态的备份和迁移。

回滚

如果新的部署不健康——新 VM 无法启动、节点未变为 `Ready`，或应用以意料之外的方式出现故障——请将每个受影响资源的模板名称改回之前的名称。Cluster API 会将回退视为新的规格偏差，并将新机器回滚到之前的模板：

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='merg
e' \
  -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<previous-
cp-template>"}}}}'

kubectl patch machinedeployment <md-name> -n cpaas-system --type='merge'
\
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<previous
-worker-template>"}}}}}'
```

在新部署运行正常之前，请勿删除之前的 `HCSMachineTemplate`。如果该模板已被删除，请先从版本控制或备份中重新创建，再执行回滚。

相关内容

- 在 Huawei Cloud Stack 上升级 Kubernetes — Kubernetes 版本升级使用相同的滚动替换机制。
- 排查 Huawei Cloud Stack 业务集群故障 — 诊断未修复的控制器启动使用带点主机名的节点时出现的 kubeadm-init-stalled 症状。
- 排查卡在 Provisioned 状态的业务集群 — 与云服务提供商无关的导入控制器诊断流程。
- 节点上的主机名行为 — 在新创建的集群上，池条目为短名称或带点名称时，`hostname` / `hostname -f` / `/etc/hosts` 的表现。

在 VMware vSphere 上管理节点

本文档介绍基准集群运行后如何在 VMware vSphere 上管理工作节点。节点生命周期操作通过 `VSphereMachineConfigPool`、`VSphereMachineTemplate`、`KubeadmConfigTemplate` 和 `MachineDeployment` 资源进行管理。

如需使用 YAML 添加另一个工作节点池，请参阅集群创建页面中的创建工作节点对象。

> **INFO**
>
> 如需升级节点的 Kubernetes 版本，请参阅在 VMware vSphere 上升级 Kubernetes。

目录

- 前提条件
- 步骤
  - 扩展工作节点
  - 发布更新后的工作节点配置
  - 验证工作节点状态
- 更新 Bootstrap 模板
- 运行时拓扑更改
- 故障排查
- 后续步骤

前提条件

开始之前，请确保满足以下条件：

- 业务集群已成功创建。请参阅在 VMware vSphere 上创建集群。
- 工作节点机器配置池有足够的可用槽位。
- 控制平面运行正常且可访问。
- 你知道当前用于定义工作节点的清单文件。
- 你了解Provider Requirements中的发布和池规则。
- 如需管理 `global` 集群本身的节点，必须先在 `global` 上安装 provider；请参阅在 Alauda OS 集群上管理节点。

WARNING

Day-2 `apply` 可能会被拒绝

发布和池规则适用于本页面执行的每次编辑：

- 重新应用归档的 `MachineDeployment` 清单会重新提交整个对象，包括唯一预期更改为 `replicas` 的应用操作。在应用之前，请确认文件设置了 `strategy.rollingUpdate.maxSurge: 0`；示例使用 `maxUnavailable: 1`。在默认的 `maxSurge` 为 `1` 时，发布操作需要一个完整池没有的额外槽位，因此会停滞。`strategy.type: OnDelete` 不需要 surge 设置。
- `KubeadmControlPlane` 必须将 `spec.rolloutStrategy.rollingUpdate.maxSurge: 0` 和 `spec.replicas` 设置为 `3` 或更高值，否则对其进行编辑会被拒绝。请参阅根据准入规则检查清单。
- `status.configStatuses[].state` 为 `InUse` 或 `Released` 的槽位不能移除，不能更改其 `network.primary.ip` 或 `.ipv6`，也不能从 `persistentDisks[]` 中移除磁盘，或更改其 `sizeGiB` 或 `mountPath`。添加槽位会被接受。请勿向此类槽位添加磁盘；请参阅已分配槽位限制。

请将拒绝消息与准入拒绝进行匹配。

步骤

1 扩展工作节点

添加更多工作节点时，请先更新工作节点机器配置池，再增加副本数。

工作节点扩展取决于 `MachineDeployment.spec.replicas` 与 `VSphereMachineConfigPool.spec.configs[]` 中可用节点槽位之间的关系。槽位数可以大于 `replicas`，空闲槽位不会影响正在运行的集群。如果 `replicas` 超过可用槽位数，CAPV 将无法正确分配新工作节点。

1. 向 `17-vspheremachineconfigpool-worker.yaml` 添加一个或多个新节点槽位。
2. 更新 `30-workers-md-0.yaml` 中的 `replicas`。
3. 应用更新后的清单。

以下示例添加了一个包含所需数据磁盘的工作节点槽位：

```
- hostname: "<worker_node_name_2>"
  datacenter: "<worker_02_datacenter>"
  network:
    primary:
      networkName: "<nic1_network_name>"
      ip: "<worker_02_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
```

此示例使用在 provider `v1.0.17` 上创建的集群的磁盘布局。在较早版本 provider 上创建的池没有 `ephemeralDisks[]`，而是在 `persistentDisks[]` 中声明 `/var/cpaas`、`/var/lib/containerd` 和 `/var/lib/etcd`；请匹配该池已有的槽位，而不要将 `ephemeralDisks[]` 引入其中。

若要附加额外的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
```

原始磁盘在来宾 OS 中可通过 `/dev/disk/by-capv/app-data` 使用。在滚动更新期间，同一个 VMDK 会重新附加到替换 VM，并重新创建符号链接。应用负责管理磁盘内容和运行时挂载。

然后增加工作节点副本数：

```
replicas: <worker_replicas>
```

请按以下顺序执行：

```
kubectl apply -f 17-vspheremachineconfigpool-worker.yaml
kubectl apply -f 30-workers-md-0.yaml
```

第二条命令会替换整个 `MachineDeployment` 规范，而不仅仅是 `replicas`。应用归档文件之前，请确认其中包含 `strategy.rollingUpdate.maxSurge: 0`。

注意：当扩展或替换必须保留逻辑池时，请继续使用现有的 `VSphereMachineConfigPool.metadata.name`。不同名称的池对象之间不会重复使用槽位和 VMDK。`releaseDelayHours` 是未使用的 `Released` 槽位的回收宽限期，而不是同一池重新使用前的延迟时间。

2 发布更新后的工作节点配置

需要更改工作节点 VM 规格时，请创建新的 `VSphereMachineTemplate`，并更新 `MachineDeployment` 以引用它。这会触发滚动更新，以新配置替换工作节点。

常见更改包括：

- VM 模板名称（`spec.template.spec.template`）
- CPU 或内存大小（`numCPUs`、`memoryMiB`）
- 系统磁盘大小（`diskGiB`）
- VM 网络设备（`network.devices`）

如果要在 Kubernetes 版本保持不变的情况下将节点迁移到新的 Alauda OS 镜像，请改为按照不更改 Kubernetes 版本的 OS 镜像更新进行操作。如果 Kubernetes 版本也发生更改，请参阅在 VMware vSphere 上升级 Kubernetes。

持久磁盘在 `VSphereMachineConfigPool` 槽位中声明，而不是在 `VSphereMachineTemplate` 中声明，因此新模板不会更改这些磁盘。`InUse` 或

`Released` 槽位会拒绝更改现有磁盘的 `sizeGiB` 或 `mountPath`，因此现有磁盘需要使用不同的几何配置时，应创建新槽位，而不是编辑当前槽位。请参阅已分配槽位限制。

1. 导出现有模板

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

2. 修改模板

   编辑 `new-worker-template.yaml`：

   - 将 `metadata.name` 设置为新的唯一名称（例如 `<cluster_name>-worker-v2`）
   - 更新所需的规范字段
   - 移除服务器生成的字段：`metadata.resourceVersion`、`metadata.uid`、`metadata.generation`、`metadata.creationTimestamp`、`metadata.managedFields`、`metadata.annotations["kubectl.kubernetes.io/last-applied-configuration"]` 和 `status`

3. 应用新模板

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

4. 更新 **MachineDeployment** 引用

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

如果还需要更改引导设置，请参阅下方的更新引导模板。

5. 监控滚动更新

```
kubectl -n <namespace> get machinedeployment <cluster_name>-md-0 -
w
kubectl -n <namespace> get machine
```

若要更改控制平面节点的机器规格，请使用相同步骤，并改为修补 `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name`。

> **TIP**
>
> 回滚失败的更新
>
> 如果滚动更新失败（例如，新 VM 无法启动），请将 `MachineDeployment` 引用恢复为之前的模板名称。此操作仅在 Kubernetes 次要版本未发生更改时有效。旧模板仍然存在，Cluster API 将回滚到该模板。

3 验证工作节点状态

运行以下命令，验证管理集群和业务集群的状态：

```
kubectl -n <namespace> get machinedeployment,machine,vspheremachine,v
spherevm
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
```

确认以下结果：

- 已达到目标工作节点副本数。
- 每个新工作节点都已加入集群。
- 节点最终变为 `Ready`。

更新 **Bootstrap** 模板

要更新 Bootstrap 配置，请创建新的 `KubeadmConfigTemplate`，并更新 `MachineDeployment` 引用。

1. 导出现有模板

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

2. 修改模板

   编辑 `new-bootstrap-template.yaml`：

   - 将 `metadata.name` 设置为新的唯一名称（例如 `<cluster_name>-worker-bootstrap-v2`）
   - 更新所需的 Bootstrap 配置字段
   - 对于 Kubernetes 1.35 或更高版本，将必需的 kubelet 补丁设置添加到 `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`
   - 移除滚动发布更新的工作节点配置第 2 步中列出的相同服务器生成字段

3. 应用新模板

```
kubectl apply -f new-bootstrap-template.yaml
```

4. 更新 **MachineDeployment** 引用

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

Cluster API 控制器会触发滚动更新。在替换现有机器之前，它们会继续使用旧的 Bootstrap 配置。

运行时拓扑更改

网络、放置和磁盘拓扑属于机器模板和机器配置池的一部分。应将这些字段的更改视为替换机器的滚动发布，而不是就地编辑 VM：

1. 在增加副本数或开始替换之前，将所需的插槽定义添加到现有的 `VSphereMachineConfigPool`。已分配的插槽会拒绝某些更改；请参阅前提条件中的警告。
2. 当 VM、网络、放置或系统磁盘字段发生更改时，创建新的 `VSphereMachineTemplate`。
3. 当 Bootstrap 文件或节点配置发生更改时，创建新的 `KubeadmConfigTemplate`。
4. 仅在新的池容量和模板存在后，才更新 `MachineDeployment` 引用。
5. 监控替换的 Machine，并确认每个节点都变为 `Ready`，然后再移除旧模板。

有关确切的第二个 NIC、故障域和持久磁盘字段，请使用在 VMware vSphere 上创建集群中的创建时变体。在 Day-2 更改期间，保持相同的跨资源关系：

- 当 NIC 拓扑发生更改时，确保 `VSphereMachineConfigPool.spec.configs[].network.additional`、`VSphereMachineTemplate.spec.template.spec.network.devices` 和 `VSphereFailureDomain.spec.topology.networks` 保持一致。
- 启用故障域时，同时配置 `VSphereCluster.spec.failureDomainSelector` 和 CPI `[Labels]` 块。
- 不要移除必需的 `var-cpaas` 持久磁盘、必需的 `var-lib-kubelet` 和 `var-lib-containerd` 临时磁盘，或控制平面的 `var-lib-etcd` 临时磁盘。
- 保持新插槽的磁盘列表与该池中已有插槽的磁盘列表完全一致。`ephemeralDisks[]` 从 provider `v1.0.17` 开始存在，但在更早版本 provider 上创建的池会在 `persistentDisks[]` 中声明 `/var/cpaas`、`/var/lib/containerd` 和 `/var/lib/etcd`；不要转换这些插槽，也不要在同一个池中混用不同布局。
- 不要重塑已分配插槽的磁盘布局或向其中添加磁盘。处于 `InUse` 或 `Released` 状态的插槽会拒绝移除持久磁盘，或更改其 `sizeGiB` 或 `mountPath`。调整大小意味着规划新的插槽。

- 不要将 `machineNamingStrategy`、`wipeFilesystem` 或 `releaseDelayHours` 视为可在池之间移动磁盘或加快数据存储清理的开关。

从一个 NIC 扩展到两个 NIC 时，请同时更新以下所有字段：

1. 将第二个 NIC 条目追加到 `VSphereMachineConfigPool.spec.configs[].network.additional`；保持 `network.primary` 不变。
2. 将第二个设备添加到新模板中的 `VSphereMachineTemplate.spec.template.spec.network.devices`。
3. 启用故障域时，将第二个网络添加到 `VSphereFailureDomain.spec.topology.networks`。

从两个 NIC 恢复为一个 NIC 时，从这三个字段中移除相应的第二个条目。将 `network.additional` 留空或完全移除该键，并滚动发布新的 `VSphereMachineTemplate`；不要直接编辑现有 VM。

故障排查

工作节点管理失败时，请先执行以下检查：

- 检查 `VSphereMachine` 中 `MachineConfigPoolReady` 的状态。如果 `False`，其原因会说明插槽分配失败的原因：
  - `PoolBoundToOtherConsumer`：该池已绑定到其他 `KubeadmControlPlane` 或 `MachineDeployment`。
  - `NoAvailableSlots`：没有插槽符合所需的数据中心或故障域。
- 确认业务节点机器配置池仍有可用插槽。
- 确认业务节点 IP 地址、网关和 DNS 设置正确。
- 确认业务节点 VM 模板仍符合所需的 Kubernetes 版本和客户机工具要求。
- 当节点等待分配 IP 时，检查 `VSphereVM.status.addresses`。

后续步骤

有关基础设施准备和持久磁盘诊断，请参阅 VMware vSphere 基础设施准备。有关集群创建清单字段，请参阅在 VMware vSphere 上创建集群。

非 Alauda OS 节点

添加和删除节点

平台管理员可以向节点运行非 Alauda OS 的

管理节点

在 Web 控制台中更新节点标签和污点、停.

为节点规划

规划并验证在非

在非 Alauda OS 集群上添加和删除节点

当集群需要扩容，或需要使用新节点替换集群中的异常节点时，您可以向节点运行非 Alauda OS 的现有工作负载集群（集群类型为 **On-Premises**）添加控制平面节点和工作节点。您还可以从这些集群中删除节点，例如移除故障节点。

目录

约束和限制
前提条件
操作步骤
后续操作
查看执行进度
重新添加失败节点
删除节点
约束和限制
操作步骤

约束和限制

- 在将节点添加到集群之前准备好节点。使用节点可用性检查参考验证节点。未满足必需条件时，集群部署可能会失败。
- 要添加的节点的硬件架构必须与集群的硬件架构一致。

- 为避免出现不可预测的错误，要添加的节点的操作系统类型应与集群中的其他节点一致。
- 在同一个添加节点对话框中添加的节点，其 SSH 端口和身份验证信息必须统一。
- 集群规划指南：集群必须至少有 1 个控制平面节点。不支持将控制平面节点准确设置为 2 个。控制平面节点达到 3 个或更多时，集群将成为高可用集群（对于高可用集群，建议使用奇数个节点，最好是 3 个或 5 个）。注意：此要求仅适用于添加或更改控制平面容量；您可以安全地添加工作节点，而无需强制添加控制平面节点。
- 一个节点只能属于一个集群。已经属于其他集群的节点无法添加。

前提条件

- 当 global 集群无法通过 SSH 服务直接访问要添加到集群的节点，而需要通过代理访问时，请提前准备代理服务。目前仅支持 SOCKS5 代理。

操作步骤

1. 在左侧导航栏中，单击集群 > 集群。
2. 单击要添加节点的 **On-Premises** 类型的*集群名称*。
3. 在节点页签下，单击添加节点。
4. 配置相关参数。有关参数详情，请参阅节点配置参数。
5. 单击添加，对节点执行可用性检查。
   检查通过后，开始添加节点，节点处于添加中状态。

后续操作

查看执行进度

在节点列表页面，您可以查看已添加节点的列表信息。对于处于添加中状态的节点，您可以查看执行进度。

操作步骤

1. 单击处于添加中状态的节点右侧的查看执行进度。
2. 在弹出的执行进度对话框中，您可以查看节点执行进度（`status.conditions`）。
   提示：当某种类型正在执行或处于失败状态且存在原因时，将光标悬停在对应原因（以蓝色文本显示）上，可以查看原因的详细信息（`status.conditions.reason`）。

重新添加失败节点

添加节点后，如果部分节点添加失败，节点列表上方会显示提示。单击提示框中的重新添加按钮，重新添加失败的节点。

删除节点

您可以从 **On-Premises** 类型的集群中删除节点，例如移除故障节点。

约束和限制

- 不支持删除导入集群中的节点。
- 当集群中只有一个控制平面节点时，不支持删除该控制平面节点。

操作步骤

1. 在左侧导航栏中，单击集群管理 > 集群。
2. 单击要删除节点所在的 **On-Premises** 类型的*集群名称*。
3. 在节点页签下，单击要删除节点右侧的删除。
   提示：如果删除 Linux 节点后需要清理该节点下的资源，请单击对话框底部的下载清理脚本，将清理脚本下载到本地。节点成功删除后，登录该节点并执行清理脚本。
4. 输入节点名称，然后单击删除。

管理节点

如需创建 Infra 节点或自定义角色节点，请参阅 Infra Nodes。

目录

更新节点标签
操作步骤
停止/恢复节点调度
操作步骤
驱逐 Pod
操作步骤
设置污点
操作步骤
标签和污点管理
约束和限制
操作步骤
启用/禁用虚拟化开关

更新节点标签

标签↗ 是附加到节点的键值对，可用于定义节点属性。为节点设置标签后，您可以按标签轻松筛选或选择节点。例如：将 Pod 调度到指定节点。

支持更新处于正常状态的节点的节点标签，以及添加或删除自定义节点标签。

操作步骤

1. 在左侧导航栏中，点击 集群管理 > 集群。
2. 点击包含待更新标签节点的*集群名称*。
3. 在 节点 页签中，点击待更新标签节点右侧的 更新节点标签。
4. 添加、修改或删除节点标签。
5. 点击 确定。
   节点标签更新成功后，节点标签数量会发生变化。您可以在 节点 信息栏的 节点标签 项中查看节点的所有标签信息。

停止/恢复节点调度

通过设置节点的调度状态，可以控制集群中新创建的 Pod 是否允许调度到该节点。

- 停止调度：不允许新创建的 Pod 调度到该节点，但不影响节点上正在运行的现有 Pod。
- 恢复调度：允许新创建的 Pod 调度到该节点。

操作步骤

1. 在左侧导航栏中，点击 集群 > 集群。
2. 点击包含待停止/恢复调度节点的*集群名称*。
3. 在 节点 页签中，点击待设置调度状态的节点右侧的 停止调度/恢复调度。
4. 点击 确定。

驱逐 **Pod**

将处于正常状态节点上除由 DaemonSet（守护进程集）管理的 Pod 之外的所有 Pod 驱逐到集群中的其他节点，并将该节点设置为不可调度状态。

注意：驱逐操作会删除 Pod 中本地存储的数据。驱逐 Pod 前，请确认数据影响。

操作步骤

1. 在左侧导航栏中，点击 集群管理 > 集群。
2. 点击包含待驱逐 Pod 节点的*集群名称*。
3. 在 节点 页签中，点击要驱逐 Pod 的*节点名称*。
4. 在右上角点击 操作 > 驱逐 **Pod**。
5. 查看待驱逐 Pod 的信息，然后点击 驱逐。

设置污点

为处于正常状态的节点设置污点信息。

污点是节点的一种属性，允许节点拒绝运行某些类型的 Pod，甚至驱逐 Pod。污点与 Pod 上的容忍度配合使用，以防止 Pod 被分配到不适合的节点。每个节点可以应用一个或多个污点，无法容忍这些污点的 Pod 将不会被该节点接受。

例如：如果发现某个节点的内存利用率已达到 91%，则不建议继续向该节点调度新的 Pod。此时可以为该节点设置污点。设置污点后，Kubernetes 将不会向该节点调度 Pod。

有关 Kubernetes 污点和容忍度的行为，请参阅 污点和容忍度↗。

操作步骤

1. 在左侧导航栏中，点击 集群管理 > 集群。
2. 点击包含待设置污点节点的*集群名称*。
3. 在 节点 页签中，点击要设置污点的节点右侧的 设置污点。
4. 设置污点的键、值和效果。一个节点可以添加多个污点。
   污点属性由 `key=value [effect]` 组成。
   `key=value` 用于匹配 Pod 容忍度。污点表示该节点已受到 `key=value` 的污染，并且不允许向该节点调度 Pod 或应避免向该节点调度 Pod，除非 Pod 能够容忍（Tolerations）污点 `key=value`。
   effect 表示污点的效果，有以下三种选项：

   - **NoSchedule**：表示不允许调度，已调度的资源不受影响。

- **PreferNoSchedule**：表示尽量不进行调度。
- **NoExecute**：表示不允许调度，已调度的资源将在 `tolerationSeconds` 后被删除。

5. 点击 确定。

标签和污点管理

平台支持批量为节点设置标签和污点。

约束和限制

- 设置设备标签前，需要先在集群中部署设备插件，例如 NVIDIA GPU MPS 设备插件、NVIDIA GPU 设备插件、GPU Manager 设备插件等。
  提示：设备标签实际上是节点标签。为方便操作，平台将设备插件所依赖的节点标签归类为设备标签，以便快速配置。

操作步骤

1. 在左侧导航栏中，点击 集群 > 集群。
2. 点击要管理标签和污点的*集群名称*。
3. 在 节点 页签中，选择多个要管理的节点，然后点击 标签和污点管理按钮。
   提示：您可以在节点列表页的搜索框中输入关注的节点标签，以快速筛选出要管理标签和污点的节点列表。
4. 在 批量操作中，添加并填写要执行的操作，然后点击确定，将批量操作提交到集群。

   - 节点标签：您可以为选定节点添加/更新指定标签，也可以删除指定标签。选择删除时，平台将筛选出选定节点上的所有标签列表。当值设置为 **Any** 时，表示删除所有包含指定标签键的节点上的标签。
   - 污点：您可以为选定节点添加/更新指定污点，也可以删除指定污点。选择删除时，平台将筛选出选定节点上的所有污点列表。当值设置为 **Any** 时，表示删除所有包含指定污点键的节点上的污点。
   - 设备标签：您可以为选定节点设置要使用的设备，设备列表来自已在此集群中部署的设备插件。

启用/禁用虚拟化开关

当本地集群中的节点为物理机时，您可以通过启用/禁用节点虚拟化开关，控制 Kubernetes 是否允许将虚拟机（VMI，VirtualMachineInstance）调度到该节点。

启用开关后，允许将新创建的虚拟机调度到物理机节点；禁用开关后，禁止将新创建的虚拟机调度到物理机节点，但不会影响已在该节点上运行的虚拟机。

提示：有关相关操作和注意事项，请参阅准备虚拟化环境。

为节点规划原地 OS 升级

集群管理员可以使用此检查清单，为在非 Alauda OS 上由 ACP 创建的业务集群节点规划原地操作系统升级（集群类型为 **On-Premises**）。请在更改操作系统之前和之后完成 ACP 检查。

> **适用范围和支持边界**
>
> 请在维护时段内，将此检查清单与操作系统提供商支持的升级操作步骤配合使用。升级任何节点之前，请根据 ACP 支持矩阵确认目标操作系统和内核。此检查清单不保证支持所有操作系统升级路径。

目录

- 适用范围和限制
- 升级前
  - 步骤 1：确认目标操作系统和内核
  - 步骤 2：检查冲突软件包
  - 步骤 3：记录当前节点基线
  - 步骤 4：验证集群容量并排空节点
- 控制平面节点的其他检查
- 执行操作系统升级
- 升级后
  - 步骤 1：确认操作系统和内核
  - 步骤 2：验证基本节点配置
  - 步骤 3：验证运行时和节点组件版本

步骤 4：验证时间同步和 DNS

步骤 5：重启 kubelet 并验证节点恢复

步骤 6：验证工作负载恢复

已知高风险场景

操作检查清单

适用范围和限制

仅在满足以下所有条件时使用此操作步骤：

- 集群是由 ACP 在非 Alauda OS 上创建的业务集群。
- 你可以登录目标节点并管理节点操作系统。
- 你拥有平台管理员权限、`kubectl` 管理员权限，以及每个目标节点的 SSH 或控制台访问权限。
- 你可以在操作系统升级之前排空目标节点上的工作负载 Pod。

以下集群类型不适用此操作步骤：

- 节点运行 Alauda OS 的集群。请通过使用新镜像替换节点来应用操作系统更改。
- 云提供商管理节点或控制平面操作系统的托管云 Kubernetes 集群。
- 无法登录节点或控制平面节点的导入集群。

升级前

在更改任何节点的操作系统之前，完成以下检查。

1 步骤 1：确认目标操作系统和内核

确认目标操作系统版本、内核版本、内核来源和 CPU 架构均在支持范围内。有关当前支持矩阵和已知限制，请参阅支持的操作系统和内核版本。

在维护时段之前应用以下规则：

- 操作系统的主版本和次版本必须与支持矩阵匹配。
- 核心内核版本必须与支持矩阵匹配。只有构建后缀可以不同。
- 内核必须是操作系统供应商提供的官方内核。
- 不支持 Ubuntu HWE 内核以及第三方或自定义编译的内核。
- 如果目标版本不在支持矩阵中，请在升级之前联系技术支持确认兼容性。

2 步骤 2：检查冲突软件包

在原地操作系统升级期间，操作系统软件包管理器可能会安装或覆盖容器运行时、Kubernetes 或容器网络二进制文件。升级之前，请检查并解决与 ACP 组件冲突的软件包。有关软件包列表和命令，请参阅移除冲突的软件包。

如果发现冲突的软件包，请在卸载这些软件包之前准备应用迁移计划并备份受影响的数据。

3 步骤 3：记录当前节点基线

记录操作系统升级前的节点状态。节点升级后，使用这些记录进行比较。

现在在节点上运行步骤 2：验证基础节点配置中的 `Base node configuration check` 代码块，并保留其输出。该代码块会捕获 SELinux 模式、AppArmor 和主机防火墙状态，以及正在使用的防火墙规则。升级后的检查会与此输出进行精确比较，因此没有此输出的节点无法完成验证。

同时记录运行时和节点组件版本：

```
containerd --version
runc --version
crictl --version
kubelet --version
```

另外，记录操作系统升级过程可能会更新的关键节点配置文件，例如 `/etc/resolv.conf`、`/etc/fstab`、systemd 配置文件和容器运行时配置文件。

4 步骤 4：验证集群容量并排空节点

> 确认其余节点具有足够的容量来运行从目标节点驱逐的 Pod。然后在操作系统升级之前排空节点。
>
> 你可以使用 ACP 控制台从节点驱逐 Pod。有关控制台操作，请参阅管理节点。
>
> 如果使用 `kubectl`，请在运行之前与运维团队确认命令选项。例如：
>
> ```
> kubectl cordon <node-name>
> kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
> ```
>
> 排空操作不会驱逐由 DaemonSet 管理的 Pod。使用本地存储的工作负载在驱逐后可能会丢失本地数据。继续操作之前，请确认对工作负载的影响。

控制平面节点的其他检查

控制平面节点运行 `kube-apiserver`、`etcd`、`kube-scheduler` 和 `kube-controller-manager` 等组件。一次升级一个控制平面节点，并在每个节点升级完成后验证集群运行状况。

升级控制平面节点之前：

- 备份 etcd 数据。有关支持的备份机制和还原注意事项，请参阅 etcd 备份和还原。
- 确认 etcd 集群运行状况正常，并且在一个控制平面节点不可用期间仍可维持仲裁。
- 在对控制平面节点执行滚动维护时，确认集群至少有三个控制平面节点。
- 如果可能，先在工作节点上验证操作步骤，然后再继续处理控制平面节点。

检查控制平面运行状况时，可以参考以下命令：

```
kubectl get nodes
kubectl get pods -n kube-system | grep -E "etcd|apiserver|scheduler|contr
oller"
```

如果在控制平面节点上使用 `etcdctl`，请使用环境中的证书路径：

```
ETCDCTL_API=3 etcdctl member list \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --write-out=table
```

执行操作系统升级

按照操作系统供应商针对目标版本提供的支持升级操作步骤执行升级。具体的软件包命令、软件仓库配置和重启要求由操作系统供应商以及组织的操作系统维护策略决定。

在操作系统升级期间：

- 不要有意安装与 ACP 组件冲突的容器运行时、Kubernetes 或容器网络软件包。
- 保留节点网络配置、DNS 配置、时间同步配置和 systemd 服务配置，除非供应商操作步骤要求进行更改。
- 根据供应商操作步骤的要求重启节点。
- 在所有升级后检查通过之前，保持节点不可调度。

升级后

操作系统升级和重启完成后，在每个节点上执行以下检查。

1 步骤 1：确认操作系统和内核

验证节点运行的操作系统和内核版本是否符合预期。

```
cat /etc/os-release
uname -r
```

将输出结果与受支持的矩阵和已批准的维护计划进行比较。

2 步骤 2：验证基本节点配置

验证操作系统升级未还原所需的节点设置。

```
echo "=== Base node configuration check ==="

# SELinux must be disabled. Record the value so that you can confirm
the
# operating system upgrade did not turn it back on.
if command -v getenforce >/dev/null; then
  echo "SELinux: $(getenforce)"
else
  echo "SELinux: getenforce not available on this system"
fi

# AppArmor must be disabled. A missing unit means the system has no A
ppArmor at
# all, which is not the same as AppArmor being present but turned of
f.
if systemctl list-unit-files --no-legend apparmor.service 2>/dev/null
| grep -q .; then
  echo "AppArmor: enabled=$(systemctl is-enabled apparmor.service 2>&
1) active=$(systemctl is-active apparmor.service 2>&1)"
else
  echo "AppArmor: unit not present on this system"
fi

# firewalld. Same distinction as AppArmor.
if systemctl list-unit-files --no-legend firewalld.service 2>/dev/nul
l | grep -q .; then
  echo "firewalld: enabled=$(systemctl is-enabled firewalld.service 2
>&1) active=$(systemctl is-active firewalld.service 2>&1)"
  firewall-cmd --list-all 2>/dev/null || true
else
  echo "firewalld: unit not present on this system"
fi

# ufw. The unit reports active (exited) once it has ever run, so the
unit state
# does not show whether the ruleset is enabled. Only ufw status does.
if command -v ufw >/dev/null; then
  ufw status verbose
else
  echo "ufw: not installed"
fi

# Swap must be disabled.
```

```
swapon --show
free -h | grep Swap

# /tmp must not be mounted with noexec.
mount | grep " /tmp "

# DefaultTasksMax must be infinity.
systemctl show --property=DefaultTasksMax
```

如果操作系统升级后启用了 `swap`，请根据操作系统策略将其禁用，并从 `/etc/fstab` 中移除 swap 条目。

必须禁用 SELinux 和 AppArmor。如果操作系统升级重新启用了其中任何一个，请根据操作系统策略再次将其禁用。请参阅必须禁用 SELinux 和 AppArmor 的原因。

主机防火墙必须恢复到集群安装时的状态。如果集群安装时禁用了主机防火墙，而操作系统升级重新启用了它，请再次将其禁用。如果集群运行时启用了主机防火墙，请确认升级后防火墙仍处于启用状态且规则完整。有关所需配置和端口列表，请参阅强化节点安全配置和通信矩阵。

3 步骤 3：验证运行时和节点组件版本

将运行时和节点组件版本与升级前的记录进行比较。

```
containerd --version
runc --version
crictl --version
kubelet --version
```

然后验证 `containerd` 正在运行：

```
systemctl status containerd
```

如果任何二进制文件被操作系统软件包覆盖，请停止维护操作，并在继续处理其他节点之前联系技术支持。

4 步骤 4：验证时间同步和 DNS

验证时间同步和 DNS 配置未因操作系统升级而发生更改。

```
date
timedatectl
cat /etc/resolv.conf
```

节点之间的时间偏差不得超过 10 秒。如果偏差大于 10 秒，请在节点上重启工作负载之前同步时间。

5 步骤 5：重启 kubelet 并验证节点恢复

操作系统升级完成且基本节点检查通过后，重启 `kubelet`。

```
systemctl daemon-reload
systemctl restart kubelet
systemctl status kubelet
```

等待节点返回 `Ready` 状态。

```
kubectl get node <node-name> --watch
```

节点运行状况正常后，恢复调度。

```
kubectl uncordon <node-name>
```

你也可以从 ACP 控制台恢复调度。有关控制台操作，请参阅管理节点。

6 步骤 6：验证工作负载恢复

恢复调度后，验证节点状态和工作负载放置情况。

```
kubectl get nodes
kubectl get pods -A -o wide | grep <node-name>
```

然后验证受节点排空影响的业务服务。只有在当前节点及相关服务运行状况正常后，才能继续处理下一个节点。

已知高风险场景

症状	可能原因	检查方法	建议操作
`containerd` 无法启动	操作系统升级覆盖了 ACP 运行时二进制文件	将 `containerd --version` 与升级前的记录进行比较	停止维护并联系技术支持
节点持续处于 `NotReady` 状态	`kubelet`、容器运行时、CNI、DNS 或节点网络配置发生了变化	检查 `systemctl status kubelet`、`systemctl status containerd` 和节点事件	恢复已更改的配置，或联系技术支持
集群组件报告 TLS 错误	升级期间或升级后节点时间发生偏差	检查 `timedatectl`，并比较各节点上的 `date`	同步时间后再继续
DNS 解析失败	`/etc/resolv.conf` 被覆盖	检查 `cat /etc/resolv.conf`	恢复已批准的 DNS 配置
`kubelet` 因安全策略错误而失败	操作系统升级重新启用了 SELinux 或 AppArmor	运行 `Base node configuration check` 块，并将其与升级前的记录进行比较	根据操作系统策略再次禁用该机制；如果操作系统要求，请重启节点。如果错误仍然存在，请联系技术支持
取消封锁后无法调度工作负载	节点仍处于运行状况异常、带有污点或资源受限的状态	检查 `kubectl describe node <node-name>`	解决节点状况问题后再继续

操作检查清单

在维护窗口期间使用此检查清单，并在升级后保留已完成的记录。

阶段	项目	负责人	状态
升级前	确认目标操作系统和内核位于支持矩阵中		
升级前	确认内核为供应商提供的官方内核		
升级前	检查并解决冲突的软件包		
升级前	记录 `containerd`、`runc`、`crictl` 和 `kubelet` 版本		
升级前	记录 DNS、时间同步、`/etc/fstab` 和运行时配置		
升级前	记录 SELinux、AppArmor 和主机防火墙是否已启用，并记录当前使用的防火墙规则		
升级前	确认集群有足够的容量来承载被驱逐的工作负载		
升级前	排空目标节点并确认对工作负载的影响		
仅控制平面节点	备份 etcd 数据		
仅控制平面节点	确认 etcd 健康状态和仲裁状态		
升级	执行操作系统供应商支持的升级操作步骤		
升级	如果供应商操作步骤要求，重启节点		
升级后	确认操作系统和内核版本		
升级后	确认 SELinux 和 AppArmor 已禁用		
升级后	确认交换分区已禁用		

阶段	项目	负责人	状态
升级后	确认主机防火墙与集群安装配置一致		
升级后	确认 `/tmp` 未使用 `noexec` 挂载		
升级后	确认 `DefaultTasksMax` 为 `infinity`		
升级后	确认运行时和节点组件版本未被覆盖		
升级后	确认 `containerd` 和 `kubelet` 状态正常		
升级后	确认节点之间的时间偏差不超过 10 秒		
升级后	确认 DNS 配置正确		
升级后	确认节点处于 `Ready` 状态		
升级后	恢复节点上的调度		
升级后	确认工作负载和业务服务运行状况正常		