天翼云

云容器引擎

用户手册

天翼云科技有限公司

目 录

1 什么是云容器引擎

Kubernetes 是主流的开源容器编排平台。为了让用户可以方便地在云上使用 Kubernetes 管理容器应用，天翼云推出了基于原生 Kubernetes 的云容器引擎服务。

云容器引擎（Cloud Container Engine，简称 CCE）提供高度可扩展的、高性能的企业级 Kubernetes 集群，支持运行 Docker 容器。借助云容器引擎，您可以在云上轻松部署、管理和扩展容器化应用程序。

云容器引擎深度整合了高性能的计算（ECS/BMS）、网络（VPC/EIP/ELB）、存储（EVS/OBS/SFS）等服务，并支持 CPU、GPU 等异构计算架构，支持多可用区（Available zone，简称 AZ）、多区域（Region）容灾等技术构建高可用 Kubernetes 集群，并提供高性能可伸缩的容器应用管理能力，简化集群的搭建和扩容等工作，让您专注于容器化应用的开发与管理。

名词解释

使用云容器引擎服务，会涉及到以下基本概念：

- **集群：**是指容器运行所需云资源的集合，包含了若干台云主机、负载均衡器等云资源。
- **实例（Pod）：**由相关的一个或多个容器构成一个实例，这些容器共享相同的存储和网络空间。
- **工作负载：**Kubernetes 资源对象，用于管理 Pod 副本的创建、调度以及整个生命周期的自动控制。
- **Service：**由多个相同配置的实例（Pod）和访问这些实例（Pod）的规则组成的微服务。
- **Ingress：**Ingress 是用于将外部 HTTP（S）流量路由到服务（Service）的规则集合。
- **Helm 应用：**Helm 是管理 Kubernetes 应用程序的打包工具，提供了 Helm Chart 在指定集群内图形化的增删改查。
- **镜像仓库：**用于存放 Docker 镜像，Docker 镜像用于部署容器服务。

您在使用前可以了解更多 Kubernetes 相关知识，具体请参见 https://kubernetes.io/docs/concepts/。

主要功能

云容器引擎支持对容器应用的全生命周期管理，具有以下功能：

集群管理

- 通过控制台一键创建 Kubernetes 集群，支持跨可用区高可用。

一站式容器管理

- 容器应用全生命周期管理。
- 高性能容器隧道网络、VPC 网络等容器网络。
- 云硬盘 EBS、弹性文件存储 SFS、对象存储 OBS 等持久化存储支持。
- 资源、应用、容器多维度监控。
- 基于角色的权限管理和容器运行时安全。

应用市场内容

对接容器镜像服务 SWR，支持自定义镜像和共享镜像。

2 高危操作及解决方案

业务部署或运行过程中，用户可能会触发不同层面的高危操作，导致不同程度上的业务故障。为了能够更好地帮助用户预估及避免操作风险，本节将从集群/节点、网络与负载均衡、日志、云硬盘多个维度出发，为用户展示哪些高危操作会导致怎样的后果，以及为用户提供相应的误操作解决方案。

集群/节点

表2-1 集群及节点高危操作

分类	高危操作	导致后果	误操作后解决方案
Master 节点	修改集群内节点安全组	可能导致 master 节点无法使用 **说明** 命名规则：集群名称-cce-control-随机数	参照新建集群的安全组进行修复，放通安全组。
	节点到期或被销毁	该 master 节点不可用	不可恢复。
	重装操作系统	master 组件被删除	不可恢复。
	自行升级 master 或者 etcd 组件版本	可能导致集群无法使用	回退到原始版本。
	删除或格式化节点/etc/kubernetes 等核心目录数据	该 master 节点不可用	不可恢复。
	更改节点 IP	该 master 节点不可用	改回原 IP。
	自行修改核心组件（etcd、kube-apiserver、docker 等）参数	可能导致 master 节点不可用	按照推荐配置参数恢复，详情请参见配置管理。

分类	高危操作	导致后果	误操作后解决方案
	自行更换 master 或 etcd 证书	可能导致集群不可用	不可恢复。
Node 节点	修改集群内节点安全组	可能导致节点无法使用 **说明** 命名规则：集群名称-cce-node-随机数	参照新建集群的安全组进行修复，放通安全组。
	节点被删除	该节点不可用	不可恢复。
	重装操作系统	节点组件被删除，节点不可用	重置节点，具体请参见重置节点。
	升级节点内核	可能导致节点无法使用或网络异常 **说明** 节点运行依赖系统内核版本，如非必要，请不要使用 yum update 更新或重装节点的操作系统内核（使用原镜像或其它镜像重装均属高危操作）	重置节点，具体请参见重置节点。
	更改节点 IP	节点不可用	改回原 IP。
	自行修改核心组件（kubelet、kube-proxy 等）参数	可能导致节点不可用、修改安全相关配置导致组件不安全等	按照推荐配置参数恢复，详情请参见配置管理。
	修改操作系统配置	可能导致节点不可用	尝试还原配置项或重置节点，具体请参见重置节点。
	删除或修改 /opt/cloud/cce、/var/paas 目录，删除数据盘	节点不可用	重置节点，具体请参见重置节点。
	修改节点内目录权限、容器目录权限等	权限异常	不建议修改，请自行恢复。
	对节点进行磁盘格式化或分区，包括系统盘、docker 盘和 kubelet 盘	可能导致节点不可用	重置节点，具体请参见重置节点。
	在节点上安装自己的其他软件	导致安装在节点上的 Kubernetes 组件异常，节	卸载已安装软件，尝试恢复或重置节点，具体

分类	高危操作	导致后果	误操作后解决方案
		点状态变成不可用，无法部署工作负载到此节点	请参见重置节点。
	修改NetworkManager的配置	节点不可用	重置节点，具体请参见重置节点。
	删除节点上的cfe-pause等系统镜像	导致无法正常创建容器，且无法拉取系统镜像	请从其他正常节点拷贝该镜像恢复

网络与负载均衡

表2-2 网络与负载均衡

高危操作	导致后果	误操作后解决方案
修改内核参数net.ipv4.ip_forward=0	网络不通	修改内核参数为net.ipv4.ip_forward=1
修改内核参数net.ipv4.tcp_tw_recycle=1	导致nat异常	修改内核参数net.ipv4.tcp_tw_recycle=0
修改内核参数net.ipv4.tcp_tw_reuse=1	导致网络异常	修改内核参数net.ipv4.tcp_tw_reuse=0
节点安全组配置未放通容器CIDR的53端口udp	集群内DNS无法正常工作	参照新建集群的安全组进行修复，放通安全组。
通过ELB的控制台在CCE管理的ELB创建自定义的监听器	所做修改被CCE侧重置或Ingress故障	通过service的yaml来自动创建监听器。
通过ELB的控制台在CCE管理的ELB绑定自定义的后端		禁止手动绑定后端。
通过ELB的控制台修改CCE管理的ELB的证书		通过ingress的yaml来自动管理证书。
通过ELB的控制台修改CCE管理的ELB监听器名称		禁止修改CCE管理的ELB监听器名称。
通过ELB的控制台修改CCE管理的ELB实例、监听器、转发策略的描述		禁止修改CCE管理的ELB实例、监听器、转发策略的描述。
删除default-network的network-attachment-definitions的crd资源	容器网络不通，集群删除失败等	误删除该资源需要使用正确的配置创建default-network资源。

日志

表2-3 日志

高危操作	导致后果	误操作后解决方案
删除宿主机/tmp/ccs-log-collector/pos 目录	日志重复采集	无
删除宿主机/tmp/ccs-log-collector/buffer 目录	日志丢失	无

云硬盘

表2-4 云硬盘

高危操作	导致后果	误操作后解决方案	备注
控制台手动解挂 EBS	Pod 写入报 io error	删掉 node 上 mount 目录，重新调度 Pod	Pod 里面的文件记录了文件的采集位置
节点上 umount 磁盘挂载路径	Pod 写入本地磁盘	重新 mount 对应目录到 Pod 中	Buffer 里面是待消费的日志缓存文件
节点上直接操作 EVS	Pod 写入本地磁盘	无	无

3 快速入门

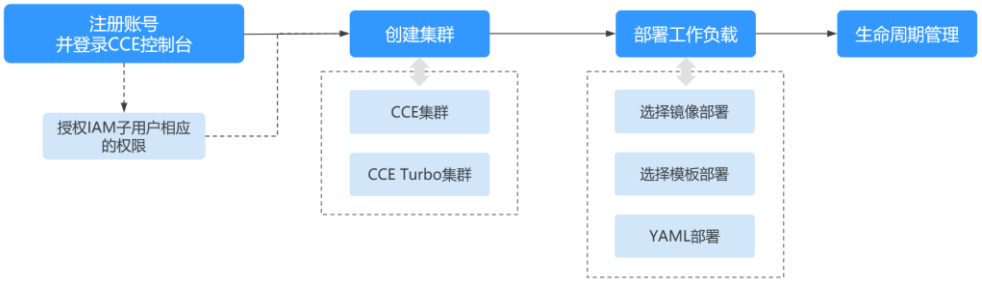
3.1 入门指引

本文旨在帮助您了解云容器引擎（Cloud Container Engine，简称 CCE）的基本使用流程以及相关的常见问题，帮助您快速上手容器服务。

使用步骤

完整的云容器引擎使用流程包含以下步骤：

图3-1 CCE 使用流程



步骤 1 **注册帐号，并授予 IAM 用户相应的权限。**

帐号无需授权即可拥有所有权限，由帐号创建的 IAM 子用户需要授予相应的权限才能使用 CCE。

步骤 2 **创建集群。**

如果您需要创建普通 Kubernetes 集群，请参见 集群。

步骤 3 **通过镜像或编排模板创建工作负载（应用）。**

步骤 4 **查看部署后工作负载的状态和日志信息，对工作负载进行相应的升级、伸缩和监控等。**

----结束

常见问题

1. **我不懂 kubernetes，是否可以使用 CCE？**

   可以使用，CCE 管理控制台操作简单，并提供新手入门指导文档，您可以快速了解并使用 CCE。

2. **我不会制作镜像，是否可以使用 CCE？**

   CCE 除了提供“我的镜像”功能用于存储您自行创建的镜像外，您还可以基于开源镜像创建容器应用。

3. **如何使用 CCE 创建工作负载？**

   创建工作负载非常简单，您只需要先创建一个集群，再创建工作负载即可。详细步骤请参考 ）。

4. **如何创建一个可以在公网访问的工作负载？**

   云容器引擎为满足多种复杂场景下工作负载间的互相访问，提供了不同的访问方式，从而满足不同场景提供不同访问通道。

5. **我有多个工作负载（在同个集群中），它们之间需要互相访问，应该怎么办？**

   集群内访问表示工作负载暴露给同一集群内其他工作负载访问的方式，可以通过“集群内部域名”访问。

   集群内部域名格式为“<自定义的服务名称>.<工作负载所在命名空间>.svc.cluster.local:<端口号>”，例如“nginx.default.svc.cluster.local:80”。

3.2 准备工作

在使用云容器引擎前，您需要完成本文中的准备工作。

- 创建 IAM 用户
- 获取资源权限
- （可选）创建虚拟私有云
- （可选）创建密钥对

创建 IAM 用户

如果您需要多用户协同操作管理您帐号下的资源，为了避免共享您的密码/访问密钥，您可以通过 IAM 创建用户，并授予用户对应权限。这些用户可以使用特别的登录链接和自己单独的用户帐号访问，帮助您高效的管理资源，您还可以设置帐号安全策略确保这些帐号的安全，从而降低您的企业信息安全风险。

注册帐号无需授权，由帐号创建的 IAM 用户需要授予相应的权限才能使用 CCE。

获取资源权限

由于 CCE 在运行中对计算、存储、网络以及监控等各类云服务资源都存在依赖关系，因此当您首次登录 CCE 控制台时，CCE 将自动请求获取当前区域下的云资源权限，从而更好地为您提供服务。服务权限包括：

- 计算类服务

CCE 集群创建节点时会关联创建云主机，因此需要获取访问云主机、物理机服务器的权限。

- 存储类服务

  CCE 支持为集群下节点和容器挂载存储，因此需要获取访问云硬盘、弹性文件、对象存储等服务的权限。

- 网络类服务

  CCE 支持集群下容器发布为对外访问的服务，因此需要获取访问虚拟私有云、弹性负载均衡等服务的权限。

- 容器与监控类服务

  CCE 集群下容器支持镜像拉取、监控和日志分析等功能，需要获取访问容器镜像、应用管理等服务的权限。

当您同意授权后，CCE 将在 IAM 中创建名为“cce_admin_trust”委托，统一使用系统帐号“op_svc_cce”对您的其他云服务资源进行操作，并且授予其 Tenant Administrator 权限。Tenant Administrator 拥有除 IAM 管理外的全部云服务管理员权限，用于对 CCE 所依赖的其他云服务资源进行调用，且该授权仅在当前区域生效。

如果您在多个区域中使用 CCE 服务，则需在每个区域中分别申请云资源权限。您可前往“IAM 控制台 > 委托”页签，单击“cce_admin_trust”查看各区域的授权记录。

说明

由于 CCE 对其他云服务有许多依赖，如果没有 Tenant Administrator 权限，可能会因为某个服务权限不足而影响 CCE 功能的正常使用。因此在使用 CCE 服务期间，请不要自行删除或者修改“cce_admin_trust”委托。

（可选）创建虚拟私有云

虚拟私有云为 CCE 集群提供一个隔离的、用户自主配置和管理的虚拟网络环境。

创建首个集群前，您必须先确保已存在虚拟私有云，否则无法创建集群。

若您已有虚拟私有云，可重复使用，无需重复创建。

步骤 1 登录管理控制台。

步骤 2 单击管理控制台左上角的，选择区域和项目。

步骤 3 选择“网络 > 虚拟私有云”。

步骤 4 单击“创建虚拟私有云”。

步骤 5 在“创建虚拟私有云”页面，根据界面提示配置虚拟私有云参数。

创建虚拟私有云时会同时创建一个默认子网，您还可以单击“添加子网”创建多个子网。

步骤 6 单击“立即创建”。

----结束

（可选）创建密钥对

云平台使用公共密钥密码术来保护您的云容器引擎节点的登录信息，密码或密钥对用于远程登录节点时的身份认证。

- 如果选择密钥登录方式，您需要在创建云容器引擎的集群节点时指定密钥对的名称，然后在 SSH 登录时提供私钥。
- 如果选择密码登录方式，可以跳过该任务。

说明

如果您计划在多个区域创建实例，则需要在每个区域中创建密钥对。

通过管理控制台创建密钥对

如果您尚未创建密钥对，可以通过管理控制台自行创建。步骤如下：

步骤 1 登录管理控制台。

步骤 2 单击管理控制台左上角的，选择区域和项目。

步骤 3 选择“计算 > 弹性云主机”。

步骤 4 在左侧导航树中，选择“密钥对”。

步骤 5 在“密钥对”页面，单击“创建密钥对”。

步骤 6 输入密钥名称，单击“确定”。

步骤 7 密钥名称由两部分组成：KeyPair-4 位随机数字，使用一个容易记住的名称，如 KeyPair-xxxx_ecs。

步骤 8 您的浏览器会提示您下载或自动下载私钥文件。文件名是您为密钥对指定的名称，文件扩展名为“.pem”。请将私钥文件保存在安全位置。然后在系统弹出的提示框中单击“确定”。

说明

这是您保存私钥文件的唯一机会，请妥善保管。当您创建弹性云主机时，您将需要提供密钥对的名称；每次 SSH 登录到弹性云主机时，您将需要提供相应的私钥。

----结束

3.3 快速创建 Kubernetes 集群

创建集群

步骤 1 登录 CCE 控制台。

- 如果您的帐号还未创建过集群，会看见一个引导页面，请在 CCE 集群下单击“创建”。
- 如果您的帐号已经创建过集群，请在左侧菜单栏选择集群管理，在右侧页面 CCE 集群下单击“购买”。

步骤 2 在购买 CCE 集群页面的“服务选型”步骤中配置集群参数：

本例中大多数配置保留默认值，仅解释必要参数，参照下表设置服务选型参数。

表3-1 创建集群参数配置

参数	参数说明
基础配置	
* 集群名称	新建集群的名称。集群名称长度范围为 4-128 个字符，以小写字母开头，由小写字母、数字、中划线（-）组成，且不能以中划线（-）结尾。
* 企业项目	该参数仅对开通企业项目的企业客户帐号显示，不显示时请忽略。
* 集群版本	建议选择最新的版本。
* 集群规模	当前集群可以管理的最大 **Node 节点**规模。若选择 50 节点，表示当前集群最多可管理 50 个 Node 节点。
* 高可用	默认选择“是”。
网络配置	
* 网络模型	默认即可。
* 虚拟私有云	新建集群所在的虚拟私有云。 若没有可选虚拟私有云，单击“新建虚拟私有云”进行创建，完成创建后单击刷新按钮。
* 控制节点子网	集群 Master 节点所在的子网。
* 容器网段	按默认配置即可。
* IPv4 服务网段	同一集群下容器互相访问时使用的 Service 资源的网段。决定了 Service 资源的上限。 创建后不可修改。

步骤 3 单击“下一步：插件配置”，配置默认插件即可。

步骤 4 单击“下一步：规格确认”，显示集群资源清单，确认无误后，勾选“我已阅读并知晓上述使用说明”，单击“提交”。

等待集群创建成功，创建集群预计需要 6-10 分钟左右，请耐心等待。

创建成功后在集群管理下会显示一个运行中的集群，且集群节点数量为 0。

----结束

创建节点

集群创建成功后，您还需要在集群中创建运行工作负载的节点。

步骤 1 登录 CCE 控制台。

步骤 2 单击创建的集群，进入集群控制台。

步骤 3 在左侧菜单栏选择节点管理，单击右上角“创建节点”，在弹出的页面中配置节点的参数。

本例中大多数配置保留默认值，仅解释必要参数。

计算配置

- 可用区：默认即可。
- 节点类型：选择“虚拟机节点”。
- 节点规格：请根据业务需求选择相应的节点规格。
- 容器引擎：请根据业务需要选择相应的容器引擎。
- 操作系统：请选择节点对应的操作系统。
- 节点名称：自定义节点名称。
- 登录方式：
  - 选择“密码”：用户名默认为“root”，请输入登录节点的密码，并确认密码。

    请妥善管理密码，登录节点时需要使用该密码，系统无法获取您设置的密码内容。
  - 选择“密钥对”：在选项框中选择用于登录本节点的密钥对，并单击勾选确认信息。

    密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。

存储配置

- 系统盘：按您的业务需求选择，缺省值为 50GB。
- 数据盘：按您的业务需求选择，缺省值为 100GB。

网络配置

- 虚拟私有云：使用默认，即创建集群时选择的子网。
- 节点子网：选择节点所在的子网。
- 节点 IP：支持指定节点 IP 地址。
- 弹性公网 IP：默认为“暂不使用”。支持“选择已有”和“自动创建”。

步骤 4 在页面最下方选择节点的数量，单击“下一步: 规格确认”。

步骤 5 查看节点规格无误后，阅读页面上的使用说明，勾选“我已阅读并知晓上述使用说明”，单击“提交”。

等待节点创建成功，添加节点预计需要 6-10 分钟左右，请耐心等待。

创建成功后在节点管理下会显示一个运行中的节点。

----结束

3.4 镜像创建无状态工作负载（Nginx）

您可以使用容器镜像快速创建一个可公网访问的单实例工作负载。本章节将指导您基于云容器引擎 CCE 快速部署 Nginx 容器应用，并管理该容器应用的全生命周期，以期让您具备将云容器引擎应用到实际项目中的能力。

前提条件

- 您需要创建一个至少包含一个 4 核 8G 节点的集群，且该节点已绑定弹性 IP，并确保其可联网下载容器镜像。
- 集群是运行工作负载的逻辑分组，包含一组云主机资源，每台云主机即集群中的一个节点。
- 创建集群的方法，请参见 集群。

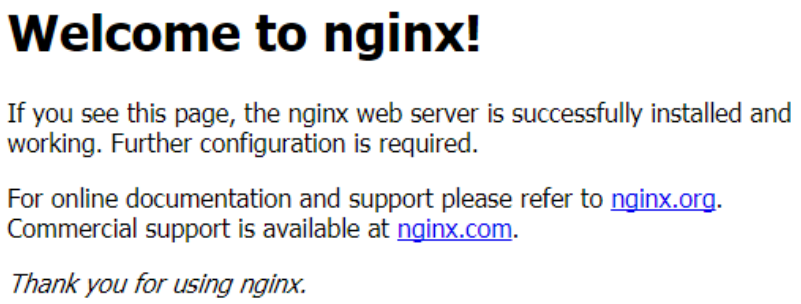
Nginx 应用概述

Nginx 是一款轻量级的 Web 服务器，您可通过 CCE 快速搭建 nginx web 服务器。

本章节将以创建 Nginx 应用为例，来创建一个工作负载，预计需要 5 分钟。

本章节执行完成后，可成功访问 Nginx 的网页，如下图。

图3-2 本例结果



操作步骤

步骤 1 登录 CCE 控制台。

步骤 2 单击集群进入集群控制台。

步骤 3 在左侧菜单栏选择“工作负载”，单击右上角“创建负载”。

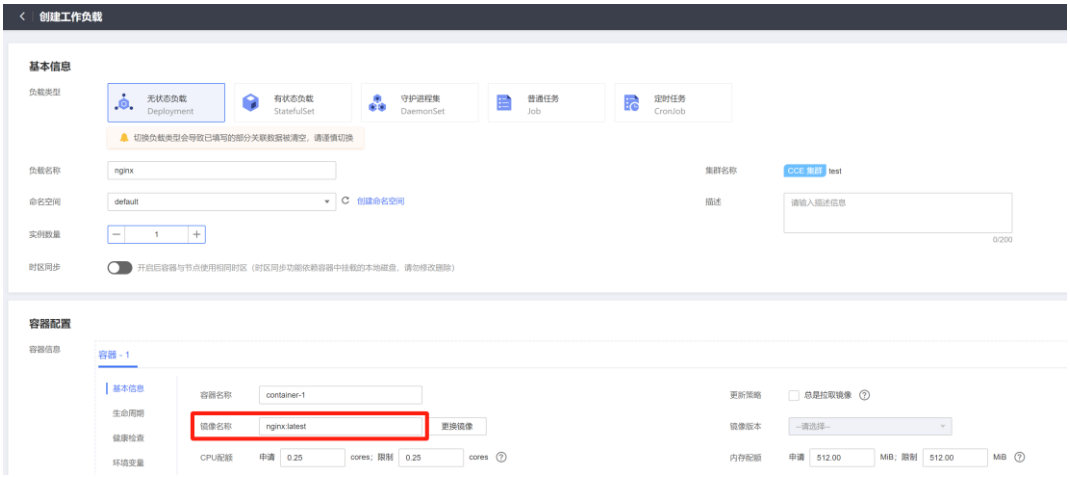
步骤 4 填写以下参数，其它保持默认。

基本信息

- 负载类型：选择无状态负载。
- 负载名称：nginx。
- 命名空间：default。
- 实例数量：请设置为 1。

容器配置

在基本信息中，“镜像名称”输入框中输入“nginx:latest”。



服务配置

单击服务配置下的加号，创建服务（Service），用于从外部访问负载。本例将创建一个负载均衡类型的 Service，请在右侧弹窗中配置如下参数。

- Service 名称：输入应用发布的可被外部访问的名称，设置为：nginx。
- 访问类型：选择“负载均衡（LoadBalancer）”。
- 服务亲和：保持默认。
- 负载均衡器：如果已有负载均衡（ELB）实例，可以选择已有 ELB，如果没有可选择“自动创建”，创建一个公网类型负载均衡器。
- 端口配置：
  - 对外协议：TCP。
  - 服务端口：设置为 9888，该端口号将映射到容器端口。
  - 容器端口：容器中应用启动监听的端口，nginx 镜像请设置为 80，其他应用容器端口和应用本身的端口一致。

步骤 5 单击右下角“创建工作负载”。

等待工作负载创建成功。

创建成功后在无状态负载下会显示一个运行中的工作负载。

----结束

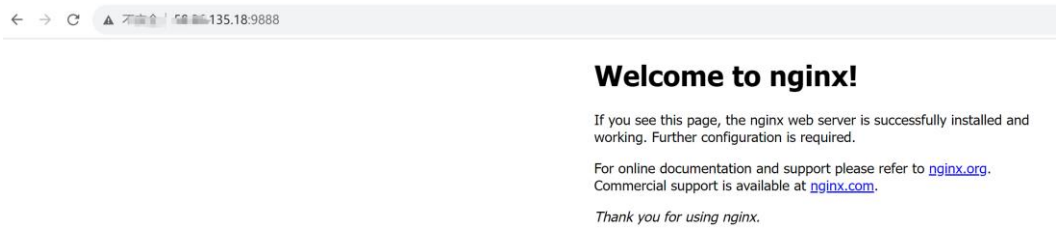
访问 Nginx

步骤 1 获取 Nginx 的外部访问地址。

单击 Nginx 工作负载名称，进入工作负载详情页。在访问方式页签下可以看到 nginx 的 IP 地址及端口，其中公网地址就是外部访问地址。

步骤 2 在浏览器中输入公网 IP 地址及端口号，即可成功访问应用，如下图所示。

图3-3 访问 nginx 应用



----结束

4 集群管理

4.1 集群概述

4.1.1 集群基本信息

Kubernetes 是一个很容易地部署和管理容器化的应用软件系统，使用 Kubernetes 能够方便对容器进行调度和编排。

对应用开发者而言，可以把 Kubernetes 看成一个集群操作系统。Kubernetes 提供服务发现、伸缩、负载均衡、自愈甚至选举等功能，让开发者从基础设施相关配置等解脱出来。

Kubernetes 可以把大量的服务器看做一台巨大的服务器，在一台大服务器上面运行应用程序。无论 Kubernetes 的集群有多少台服务器，在 Kubernetes 上部署应用程序的方法永远一样。

Kubernetes 集群架构

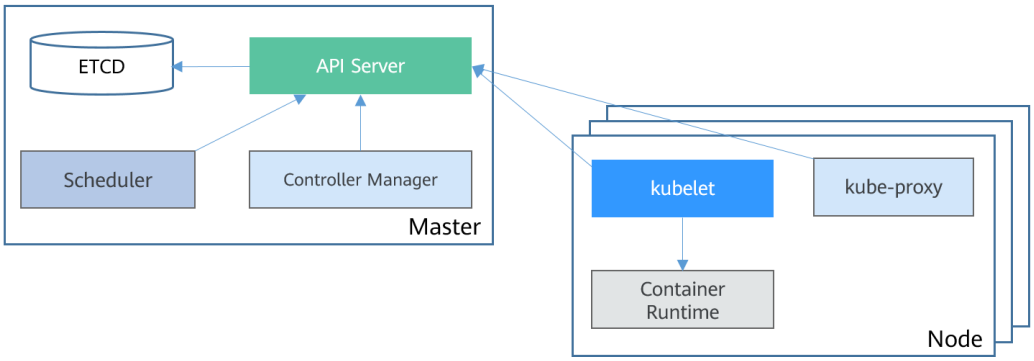
Kubernetes 集群包含 Master 节点（控制节点）和 Node 节点（计算节点/工作节点），应用部署在 Node 节点上，且可以通过配置选择应用部署在某些特定的节点上。

说明

CCE 集群的 Master 节点由云容器引擎服务创建并托管，您只需创建 Node 节点。

Kubernetes 集群的架构如下所示：

图4-1 Kubernetes 集群架构



Master 节点

Master 节点是集群的控制节点，由 API Server、Scheduler、Controller Manager 和 ETCD 四个组件构成。

- API Server：各组件互相通讯的中转站，接受外部请求，并将信息写到 ETCD 中。
- Controller Manager：执行集群级功能，例如复制组件，跟踪 Node 节点，处理节点故障等等。
- Scheduler：负责应用调度的组件，根据各种条件（如可用的资源、节点的亲和性等）将容器调度到 Node 上运行。
- ETCD：一个分布式数据存储组件，负责存储集群的配置信息。

在生产环境中，为了保障集群的高可用，通常会部署多个 Master，如 CCE 的集群高可用模式就是 3 个 Master 节点。

Node 节点

Node 节点是集群的计算节点，即运行容器化应用的节点。

- kubelet：kubelet 主要负责同 Container Runtime 打交道，并与 API Server 交互，管理节点上的容器。
- kube-proxy：应用组件间的访问代理，解决节点上应用的访问问题。
- Container Runtime：容器运行时，如 Docker，最主要的功能是下载镜像和运行容器。

Master 节点数量与集群规模

在 CCE 中创建集群，Master 节点可以是 1 个或 3 个，3 个 Master 节点会按高可用部署，确保集群的可靠性。

Master 节点的规格决定集群管理 Node 节点的规模，创建集群时可以选择集群管理规模，这个规模就是指的集群可以有多少个 Node 节点，例如 50 节点、200 节点等。

集群的网络

从网络的角度看，集群的节点都位于 VPC 之内，节点上又运行着容器，每个容器都需要访问，节点与节点、节点与容器、容器与容器都需要访问。

集群的网络可以分成三个网络来看。

- 节点网络：为集群内节点分配 IP 地址。
- 容器网络：为集群内容器分配 IP 地址，负责容器的通信，当前支持多种容器网络模型，不同模型有不同的工作机制。
- 服务网络：服务（Service）是用来解决访问容器的 Kubernetes 对象，每个 Service 都有一个固定的 IP 地址。

在创建集群时，您需要为各个网络选择合适的网段，确保各网段之间不存在冲突，每个网段下有足够的 IP 地址可用。**集群创建后不支持修改容器网络模型**，您需要在创建前做好规划和选择。

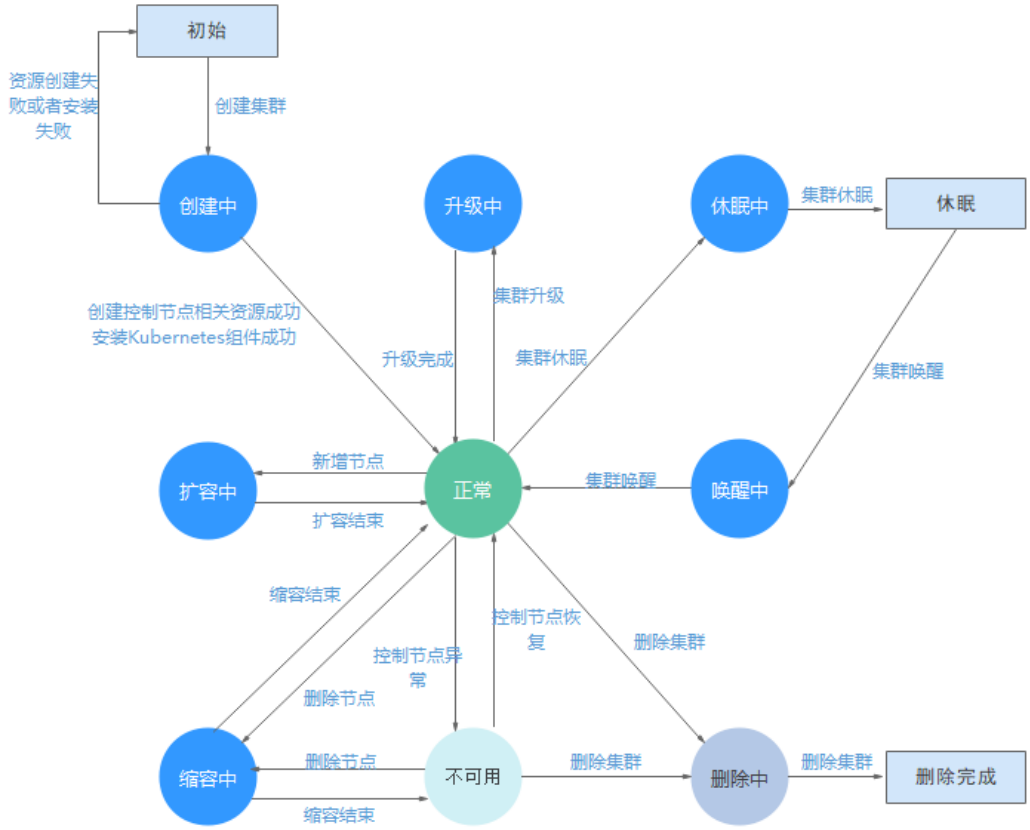
强烈建议您在创建集群前详细了解集群的网络以及容器网络模型，具体请参见容器网络模型。

集群生命周期

表4-1 集群状态说明

状态	说明
创建中	集群正在创建，正在申请云资源
运行中	集群正常运行
扩容中	集群正在增加节点
缩容中	集群正在删除节点
休眠中	集群正在休眠中
唤醒中	集群正在唤醒中
升级中	集群正在升级中
不可用	当前集群不可用
删除中	集群正在删除中

图4-2 集群状态流转



4.1.2 集群 Kubernetes 版本发布说明

4.1.2.1 CCE 发布 Kubernetes 1.25 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.25 版本所做的变更说明。

版本升级说明

CCE 针对 Kubernetes 1.25 版本提供了全链路的组件优化和升级。

表4-2 核心组件及说明

集群类型	核心组件	版本号	升级注意事项
CCE 集群	Kubernetes	1.25	无
	Docker	CentOS：18.9.0 Ubuntu：18.09.9	无
	Containerd	1.4.1	无
	操作系统	CentOS Linux release	无

集群类型	核心组件	版本号	升级注意事项
		7.6	
		Ubuntu 18.04 server 64bit	无

资源变更与弃用

CCE 1.25 变更说明

- CCE v1.25 集群的节点均默认采用 Containerd 容器引擎。

社区 1.25 ReleaseNotes

- 清理 iptables 链的所有权

  Kubernetes 通常创建 iptables 链来确保这些网络数据包到达，这些 iptables 链及其名称属于 Kubernetes 内部实现的细节，仅供内部使用场景，目前有些组件依赖于这些内部实现细节，Kubernetes 总体上不希望支持某些工具依赖这些内部实现细节。

  在 Kubernetes 1.25 版本后，Kubelet 通过 IPTablesCleanup 特性门控分阶段完成迁移，是为了不在 NAT 表中创建 iptables 链，例如 KUBE-MARK-DROP、KUBE-MARK-MASQ、KUBE-POSTROUTING。
- 存储驱动的弃用和移除，移除云服务厂商的 in-tree 卷驱动。

社区 1.24 ReleaseNotes

- 在 Kubernetes 1.24 版本后，Service.Spec.LoadBalancerIP 被弃用，因为它无法用于双栈协议。请使用自定义 annotation。
- 在 Kubernetes 1.24 版本后，kube-apiserver 移除参数--address、--insecure-bind-address、--port、--insecure-port=0。
- 在 Kubernetes 1.24 版本后，kube-controller-manager 和 kube-scheduler 移除启动参数--port=0 和--address。
- 在 Kubernetes 1.24 版本后，kube-apiserver --audit-log-version 和--audit-webhook-version 仅支持 audit.k8s.io/v1，Kubernetes 1.24 移除 audit.k8s.io/v1[alpha|beta]1，只能使用 audit.k8s.io/v1。
- 在 Kubernetes 1.24 版本后，kubelet 移除启动参数--network-plugin，仅当容器运行环境设置为 Docker 时，此特定于 Docker 的参数才有效，并会随着 Dockershim 一起删除。
- 在 Kubernetes 1.24 版本后，动态日志清理功能已经被废弃，并在 Kubernetes 1.24 版本移除。该功能引入了一个日志过滤器，可以应用于所有 Kubernetes 系统组件的日志，以防止各种类型的敏感信息通过日志泄漏。此功能可能导致日志阻塞，所以废弃。
- VolumeSnapshot v1beta1 CRD 在 Kubernetes 1.20 版本中被废弃，在 Kubernetes 1.24 版本中移除，需改用 v1 版本。
- 在 Kubernetes 1.24 版本后，移除自 1.11 版本就废弃的 service annotation tolerate-unready-endpoints，使用 Service.spec.publishNotReadyAddresses 代替。

- 在 Kubernetes 1.24 版本后，废弃 metadata.clusterName 字段，并将在下一个版本中删除。
- Kubernetes 1.24 及以后的版本，去除了 kube-proxy 监听 NodePort 的逻辑，在 NodePort 与内核 net.ipv4.ip_local_port_range 范围有冲突的情况下，可能会导致偶发的 TCP 无法连接的情况，导致健康检查失败、业务异常等问题。升级前，请确保集群没有 NodePort 端口与任意节点 net.ipv4.ip_local_port_range 范围存在冲突。

参考链接

关于 Kubernetes 1.25 与其他版本的性能对比和功能演进的更多信息，请参考：

- Kubernetes v1.25 Release Notes
- Kubernetes v1.24 Release Notes

4.1.2.2 CCE 发布 Kubernetes 1.23 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.23 版本所做的变更说明。

版本升级说明

CCE 针对 Kubernetes 1.23 版本提供了全链路的组件优化和升级。

表4-3 核心组件及说明

集群类型	核心组件	版本号	升级注意事项
CCE 集群	Kubernetes	1.23	无
	Docker	CentOS：18.9.0 Ubuntu：18.09.9	无
	Containerd	1.4.1	无
	操作系统	CentOS Linux release 7.6	无
		Ubuntu 18.04 server 64bit	无

资源变更与弃用

CCE 1.23 变更说明

- 不再支持 web-terminal 插件，使用 kubectl 对接代替。

社区 1.23 ReleaseNotes

- FlexVolume 废弃，建议使用 CSI。

- HorizontalPodAutoscaler v2 版本 GA，HorizontalPodAutoscaler API v2 在 1.23 版本中逐渐稳定。不建议使用 HorizontalPodAutoscaler v2beta2 API，建议使用新的 v2 版本 API。
- PodSecurity 支持 beta，PodSecurity 替代废弃的 PodSecurityPolicy，PodSecurity 是一个准入控制器，它根据设置实施级别的特定命名空间标签在命名空间中的 Pod 上实施 Pod 安全标准。在 1.23 中 PodSecurity 默认启用。

社区 1.22 ReleaseNotes

- Ingress 资源不再支持 networking.k8s.io/v1beta1 和 extensions/v1beta1 API。如果使用旧版本 API 管理 Ingress，会影响应用对外暴露服务，请尽快使用 networking.k8s.io/v1 替代。
- CustomResourceDefinition 资源不再支持 apiextensions.k8s.io/v1beta1 API。如果使用旧版本 API 创建自定义资源定义，会导致定义创建失败，进而影响调和（reconcile）该自定资源的控制器，请尽快使用 apiextensions.k8s.io/v1 替代
- ClusterRole、ClusterRoleBinding、Role 和 RoleBinding 资源不再支持 rbac.authorization.k8s.io/v1beta1 API。如果使用旧版本 API 管理 RBAC 资源，会影响应用的权限服务，甚至无法在集群内正常使用，请尽快使用 rbac.authorization.k8s.io/v1 替代。
- Kubernetes 版本发布周期由一年 4 个版本变为一年 3 个版本。
- StatefulSets 支持 minReadySeconds。
- 缩容时默认根据 Pod uid 排序随机选择删除 Pod（LogarithmicScaleDown）。基于该特性，可以增强 Pod 被缩容的随机性，缓解由于 Pod 拓扑分布约束带来的问题。更多信息，请参见 KEP-2185 和 issues 96748。
- BoundServiceAccountTokenVolume 特性已稳定，该特性能够提升服务账号（ServiceAccount）Token 的安全性，改变了 Pod 挂载 Token 的方式，Kubernetes 1.21 及以上版本的集群中会默认开启。

参考链接

关于 Kubernetes 1.23 与其他版本的性能对比和功能演进的更多信息，请参考：

- Kubernetes v1.23 Release Notes
- Kubernetes v1.22 Release Notes

4.1.2.3 CCE 发布 Kubernetes 1.21 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.21 版本所做的变更说明。

版本升级说明

CCE 针对 Kubernetes 1.21 版本提供了全链路的组件优化和升级。

表4-4 核心组件及说明

集群类型	核心组件	版本号	升级注意事项
CCE 集群	Kubernetes	1.21	无

集群类型	核心组件	版本号	升级注意事项
	Docker	CentOS：18.9.0 Ubuntu：18.09.9	无
	操作系统	CentOS Linux release 7.6	1.21 版本开始 CCE 集群 Centos7.6 节点 docker 存储模式使用 overlayfs
		Ubuntu 18.04 server 64bit	无

资源变更与弃用

社区 1.21 ReleaseNotes

- CronJob 现在已毕业到稳定状态，版本号变为 batch/v1。
- 不可变的 Secret 和 ConfigMap 现在已升级到稳定状态。向这些对象添加了一个新的不可变字段，以拒绝更改。此拒绝可保护集群免受可能无意中中断应用程序的更新。因为这些资源是不可变的，kubelet 不会监视或轮询更改。这减少了 kube-apiserver 的负载，提高了可扩展性和性能。更多信息，请参见 Immutable ConfigMaps。
- 优雅节点关闭现在已升级到测试状态。通过此更新，kubelet 可以感知节点关闭，并可以优雅地终止该节点的 Pod。在此更新之前，当节点关闭时，其 Pod 没有遵循预期的终止生命周期，这导致了工作负载问题。现在 kubelet 可以通过 systemd 检测即将关闭的系统，并通知正在运行的 Pod，使它们优雅地终止。
- 具有多个容器的 Pod 现在可以使用 kubectl.kubernetes.io/默认容器注释为 kubectl 命令预选容器。
- PodSecurityPolicy 废弃，详情请参见 https://kubernetes.io/blog/2021/04/06/podsecuritypolicy-deprecation-past-present-and-future/。
- BoundServiceAccountTokenVolume 特性进入 Beta，该特性能够提升服务账号（ServiceAccount）Token 的安全性，改变了 Pod 挂载 Token 的方式，Kubernetes 1.21 及以上版本的集群中会默认开启。

社区 1.20 ReleaseNotes

- API 优先级和公平性已达到测试状态，默认启用。这允许 kube-apiserver 按优先级对传入请求进行分类。更多信息，请参见 API Priority and Fairness。
- 修复 exec probe timeouts 不生效的 BUG，在此修复之前，exec 探测器不考虑 timeoutSeconds 字段。相反，探测将无限期运行，甚至超过其配置的截止日期，直到返回结果。 通过此更改，如果未指定值，将使用默认值，默认值为 1 秒。如果探测时间超过一秒，可能会导致应用健康检查失败。请再升级时确定使用该特性的应用更新 timeoutSeconds 字段。新引入的 ExecProbeTimeout 特性门控所提供的修复使集群操作员能够恢复到以前的行为，但这种行为将在后续版本中锁定并删除。

- RuntimeClass 已达到稳定状态。RuntimeClass 资源提供了一种机制，用于支持集群中的多个运行时，并将有关该容器运行时的信息公开到控制平面。
- kubectl 调试已达到测试状态。kubectl 调试直接从 kubectl 提供对常见调试工作流的支持。
- Dockershim 在 1.20 被标记为废弃，目前您可以继续在集群中使用 Docker。该变动与集群所使用的容器镜像（Image）无关。您依然可以使用 Docker 构建您的镜像。更多信息，请参见 Dockershim Deprecation FAQ。

参考链接

关于 Kubernetes 1.21 与其他版本的性能对比和功能演进的更多信息，请参考：

- Kubernetes v1.21 Release Notes
- Kubernetes v1.20 Release Notes

4.1.2.4 CCE 发布 Kubernetes 1.19 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.19 版本所做的变更说明。

版本升级说明

CCE 针对 Kubernetes 1.19 版本提供了全链路的组件优化和升级。

表4-5 核心组件及说明

集群类型	核心组件	版本号	升级注意事项
CCE 集群	Kubernetes	1.19.10	Kubernetes 1.19 版本弃用部分常用的 APIVersion。建议您在升级集群前对本文档中所列举的弃用 APIVersion 进行相应升级。
	Docker	CentOS：18.09.0.100 Ubuntu：18.09.9	无
	操作系统	CentOS Linux release 7.6	无
		Ubuntu 18.04 server 64bit	无

资源变更与弃用

社区 1.19 ReleaseNotes

- 增加对 vSphere in-tree 卷迁移至 vSphere CSI 驱动的支持。in-tree vSphere Volume 插件将不再使用，并在将来的版本中删除。

- apiextensions.k8s.io/v1beta1 已弃用，推荐使用 apiextensions.k8s.io/v1。
- apiregistration.k8s.io/v1beta1 已弃用，推荐使用 apiregistration.k8s.io/v1。
- authentication.k8s.io/v1beta1、authorization.k8s.io/v1beta1 已弃用，1.22 将移除，推荐使用 authentication.k8s.io/v1、authorization.k8s.io/v1。
- autoscaling/v2beta1 已弃用，推荐使用 autoscaling/v2beta2。
- coordination.k8s.io/v1beta1 在 1.19 中已弃用，1.22 将移除，推荐使用 v1。
- Kube-apiserver: componentstatus API 已弃用。
- Kubeadm：kubeadm config view 命令已被弃用，并将在未来版本中删除，请使用 kubectl get cm -o yaml -n kube-system kubeadm-config 来直接获取 kubeadm 配置。
- Kubeadm：弃用 kubeadm alpha kubelet config enable-dynamic 命令。
- Kubeadm：kubeadm alpha certs renew 命令--use-api 参数已弃用。
- Kubernetes 不再支持构建 hyperkube 镜像。
- Remove --export flag from kubectl get command - kubectl get 中移除 --export 参数。
- alpha 特性“ResourceLimitsPriorityFunction”已完全删除。
- storage.k8s.io/v1beta1 已弃用，推荐使用 storage.k8s.io/v1。

社区 1.18 ReleaseNotes

- kube-apiserver
  - apps/v1beta1 and apps/v1beta2 下所有资源不再提供服务，使用 apps/v1 替代。
  - extensions/v1beta1 下 daemonsets，deployments，replicasets 不再提供服务，使用 apps/v1 替代。
  - extensions/v1beta1 下 networkpolicies 不再提供服务，使用 networking.k8s.io/v1 替代。
  - extensions/v1beta1 下 podsecuritypolicies 不再提供服务，使用 policy/v1beta1 替代。
- kubelet
  - --redirect-container-streaming 不推荐使用，v1.20 会正式废弃。
  - 资源度量端点 /metrics/resource/v1alpha1 以及此端点下的所有度量标准均已弃用。请转换为端点 /metrics/resource 下的度量标准：
    - scrape_error --> scrape_error
    - node_cpu_usage_seconds_total --> node_cpu_usage_seconds
    - node_memory_working_set_bytes --> node_memory_working_set_bytes
    - container_cpu_usage_seconds_total --> container_cpu_usage_seconds
    - container_memory_working_set_bytes --> container_memory_working_set_bytes
    - scrape_error --> scrape_error
  - 在将来的发行版中，kubelet 将不再根据 CSI 规范创建 CSI NodePublishVolume 目标目录。可能需要相应地更新 CSI 驱动程序，以正确创建和处理目标路径。
- kube-proxy
  - --healthz-port 和--metrics-port 参数不建议使用，请使用--healthz-bind-address 和--metrics-bind-address。

  - 增加 EndpointSliceProxying 功能选项以控制 kube-proxy 中 EndpointSlices 的使用，默认情况下已禁用此功能。
- kubeadm
  - kubeadm upgrade node 的--kubelet-version 参数已弃用，将在后续版本中删除。
  - kubeadm alpha certs renew 命令中--use-api 参数已弃用。
  - kube-dns 已弃用，在将来的版本中将不再受支持。
  - kubeadm-config ConfigMap 中存在的 ClusterStatus 结构体已废弃，将在后续版本中删除。
- kubectl
  - --dry-run 不建议使用 boolean 和 unset values，新版本中 server|client|none 会被使用。
  - kubectl apply --server-dry-run 已弃用，替换为--dry-run=server。
- add-ons

删除 cluster-monitoring 插件。

- kube-scheduler
  - scheduling_duration_seconds 指标已弃用。
  - scheduling_algorithm_predicate_evaluation_seconds 和 scheduling_algorithm_priority_evaluation_seconds 指标已弃用，使用 framework_extension_point_duration_seconds[extension_point="Filter"]和 framework_extension_point_duration_seconds[extension_point="Score"]替代。
  - 调度器策略 AlwaysCheckAllPredicates 已弃用。
- 其他变化
  - k8s.io/node-api 组件不再更新。作为替代，可以使用位于 k8s.io/api 中的 RuntimeClass 类型和位于 k8s.io/client-go 中的 generated clients。
  - 已从 apiserver_request_total 中删除“client”标签。

参考链接

关于 Kubernetes 1.19 与其他版本的性能对比和功能演进的更多信息，请参考：

- Kubernetes v1.19.0 Release Notes
- Kubernetes v1.18.0 Release Notes

4.1.2.5 CCE 发布 Kubernetes 1.17 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.17 版本所做的变更说明。

资源变更与弃用

- apps/v1beta1 和 apps/v1beta2 下所有资源不再提供服务，使用 apps/v1 替代。
- extensions/v1beta1 下 daemonsets、deployments、replicasets 不再提供服务，使用 apps/v1 替代。

- extensions/v1beta1 下 networkpolicies 不再提供服务，使用 networking.k8s.io/v1 替代。
- extensions/v1beta1 下 podsecuritypolicies 不再提供服务，使用 policy/v1beta1 替代。
- extensions/v1beta1 ingress v1.20 版本不再提供服务，当前可使用 networking.k8s.io/v1beta1。
- scheduling.k8s.io/v1beta1 and scheduling.k8s.io/v1alpha1 下的 PriorityClass 计划在 1.17 不再提供服务，迁移至 scheduling.k8s.io/v1。
- events.k8s.io/v1beta1 中 event series.state 字段已废弃，将在 1.18 版本中移除。
- apiextensions.k8s.io/v1beta1 下 CustomResourceDefinition 已废弃，将再 1.19 不在提供服务，使用 apiextensions.k8s.io/v1。
- admissionregistration.k8s.io/v1beta1 MutatingWebhookConfiguration 和 ValidatingWebhookConfiguration 已废弃，将在 1.19 不在提供服务，使用 admissionregistration.k8s.io/v1 替换。
- rbac.authorization.k8s.io/v1alpha1 and rbac.authorization.k8s.io/v1beta1 被废弃，使用 rbac.authorization.k8s.io/v1 替代，v1.20 会正式停止服务。
- storage.k8s.io/v1beta1 CSINode object 废弃并会在未来版本中移除。

其他废弃和移除

- 移除 OutOfDisk node condition，改为使用 DiskPressure。
- scheduler.alpha.kubernetes.io/critical-pod annotation 已被移除，如需要改为设置 priorityClassName。
- beta.kubernetes.io/os 和 beta.kubernetes.io/arch 在 1.14 版本中已经废弃，计划在 1.18 版本中移除。
- 禁止通过--node-labels 设置 kubernetes.io 和 k8s.io 为前缀的标签，老版本中 kubernetes.io/availablezone 该 label 在 1.17 中移除，整改为 failure-domain.beta.kubernetes.io/zone 获取 AZ 信息。
- beta.kubernetes.io/instance-type 被废弃，使用 node.kubernetes.io/instance-type 替代。
- 移除{kubelet_root_dir}/plugins 路径。
- 移除内置集群角色 system:csi-external-provisioner 和 system:csi-external-attacher。

参考链接

关于 Kubernetes 1.17 与其他版本的性能对比和功能演进的更多信息，请参考：

- Kubernetes v1.17.0 Release Notes
- Kubernetes v1.16.0 Release Notes

4.1.2.6（停止维护）CCE 发布 Kubernetes 1.15 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.15 版本所做的变更说明。

为了能够更好地方便您使用容器服务，确保您使用稳定又可靠的 Kubernetes 版本，请您务必在维护周期结束之前升级您的 Kubernetes 集群。

版本说明

CCE 针对 Kubernetes v1.15 版本提供了全链路的组件优化和升级，v1.15 版本包含两个小版本，即 v1.15.11 和 v1.15.6-r1。

资源变更与弃用

- extensions/v1beta1 中 Ingress 已弃用，1.19 正式暂停使用，迁移到 networking.k8s.io/v1beta1
- extensions/v1beta1 中 NetworkPolicy 1.16 正式暂停使用，迁移到 networking.k8s.io/v1
- extensions/v1beta1 中 PodSecurityPolicy 1.16 正式暂停使用，迁移到 policy/v1beta1
- extensions/v1beta1、apps/v1beta1 或 apps/v1beta2 的 DaemonSet、Deployment、和 ReplicaSet，迁移至 apps/v1，1.16 版本暂停使用
- PriorityClass 升级到 scheduling.k8s.io/v1，scheduling.k8s.io/v1beta1 和 scheduling.k8s.io/v1alpha1 1.17 正式废弃
- events.k8s.io/v1beta1 Event API 中 series.state 字段废弃，将在 1.18 版本中移除

参考链接

社区 v1.13 与 v1.15 版本之间的 CHANGELOG

- v1.14 到 v1.15 的变化：

  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.15.md
- v1.13 到 v1.14 的变化：

  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.14.md

4.1.2.7（停止维护）CCE 发布 Kubernetes 1.13 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.13 版本所做的变更说明。

表4-6 v1.13 版本集群说明

Kubernetes 版本（CCE 增强版）	版本说明
v1.13.10-r0	**主要特性：** • 负载均衡支持设置名称 • 4 层负载均衡支持健康检查，7 层负载均衡支持健康检查/分配策略/会话保持 • CCE 集群支持创建物理机节点（容器隧道网络） • 支持 AI 加速型节点（搭载海思 Ascend 310 AI 处理器），适用于图像识别、视频处理、推理计算以及机器学习等场景

Kubernetes 版本（CCE 增强版）	版本说明
	• 支持配置 docker baseSize • 支持命名空间亲和调度 • 支持节点数据盘划分用户空间 • 支持集群 cpu 管理策略 • 支持集群下的节点跨子网（容器隧道网络）
v1.13.7-r0	**主要特性：** • Kubernetes 同步社区 1.13.7 版本 • 支持网络平面（NetworkAttachmentDefinition）

参考链接

社区 v1.11 与 v1.13 版本之间的 CHANGELOG

- v1.12 到 v1.13 的变化：
  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.13.md
- v1.11 到 v1.12 的变化：
  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.12.md

4.1.2.8（停止维护）CCE 发布 Kubernetes 1.11 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.11 版本所做的变更说明。

表4-7 v1.11 版本集群说明

Kubernetes 版本（CCE 增强版）	版本说明
v1.11.7-r2	**主要特性：** • GPU 支持 V100 类型 • 集群支持权限管理
v1.11.7-r0	**主要特性：** • Kubernetes 同步社区 1.11.7 版本 • 支持创建节点池（nodepool），虚拟机集群支持 • CCE 集群支持创建物理机节点（VPC 网络），支持物理机和虚机混合部署 • GPU 支持 V100 类型 • 1.11 集群对接 AOM 告警通知机制

Kubernetes 版本（CCE 增强版）	版本说明
	• Service 支持访问类型切换 • 支持服务网段 • 集群支持自定义每个节点分配的 IP 数（IP 分配）
v1.11.3-r2	**主要特性：** • 集群支持 IPv6 双栈 • ELB 负载均衡支持源 IP 跟后端服务会话保持
v1.11.3-r1	**主要特性：** • Ingress 的 URL 匹配支持 Perl 语法的正则表达式
v1.11.3-r0	**主要特性：** • Kubernetes 同步社区 1.11.3 版本 • 集群控制节点支持多可用区 • 容器存储支持对接 SFS Turbo 极速文件存储

参考链接

社区 v1.9 与 v1.11 版本之间的 CHANGELOG

- v1.10 到 v1.11 的变化：
  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.11.md
- v1.9 到 v1.10 的变化：
  https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.10.md

4.1.2.9（停止维护）CCE 发布 Kubernetes 1.9 及之前版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍 CCE 发布 Kubernetes 1.9 及之前版本所做的变更说明。

表4-8 v1.9 及之前版本集群说明

Kubernetes 版本（CCE 增强版）	版本说明
v1.9.10-r2	**主要特性：** • ELB 负载均衡支持源 IP 跟后端服务会话保持
v1.9.10-r1	**主要特性：** • 支持对接 SFS 存储 • 支持 Service 自动创建二代 ELB

Kubernetes 版本（CCE 增强版）	版本说明
	• 支持公网二代 ELB 透传源 IP • 支持设置节点最大实例数 maxPods
v1.9.10-r0	**主要特性：** • kubernetes 对接 ELB/Ingress，新增流控机制 • Kubernetes 同步社区 1.9.10 版本 • 支持 Kubernetes RBAC 能力授权 **问题修复：** • 修复操作系统 cgroup 内核 BUG 导致概率出现的节点内存泄漏问题
v1.9.7-r1	**主要特性：** • 增强 PVC 和 PV 事件的上报机制，PVC 详情页支持查看事件 • 支持对接第三方认证系统 • 集群支持纳管 EulerOS2.3 的物理机 • 数据盘支持用户自定义分配比例 • 物理机场景支持对接 EVS 云硬盘存储 • 物理机场景下支持 IB 网卡 • 物理机场景支持通过 CM-v3 接口创建节点
v1.9.7-r0	**主要特性：** • 新建集群的 Docker 版本升级到 1706 • 支持 DNS 级联 • 支持插件化管理 • Kubernetes 同步社区 1.9.7 版本 • 支持 7 层 ingress 的 https 功能 • 有状态工作负载支持迁移调度更新升级
v1.9.2-r3	**主要特性：** • 集群支持创建/纳管 CentOS7.4 操作系统的节点 • kubernetes 的 Service 支持对接 DNAT 网关服务 • NetworkPolicy 能力开放 • 增强型 ELB 支持 Service 配置多个端口 **问题修复：** • 修复 kubernetes 资源回收过程中连不上 kube-apiserver 导致 pod 残留的问题 • 修复节点弹性扩容数据不准确的问题
v1.9.2-r2	**主要特性：**

Kubernetes 版本 （CCE 增强版）	版本说明
	• 经典型 ELB 支持自定义健康检查端口 • 经典型 ELB 性能优化 • ELB 四层负载均衡支持修改 Service 的端口 **问题修复：** • 修复网络插件防止健康检查概率死锁问题 • 修复高可用集群 haproxy 连接数限制问题
v1.9.2-r1	**主要特性：** • Kubernetes 同步社区 1.9.2 版本 • 集群节点支持 CentOS 7.1 操作系统 • 支持 GPU 节点，支持 GPU 资源限制 • 支持 web-terminal 插件
v1.7.3-r13	**主要特性：** • 新建集群的 Docker 版本升级到 1706 • 支持 DNS 级联 • 支持插件化管理 • 增强 PVC 和 PV 事件的上报机制 • 物理机场景支持对接 OBS 对象存储
v1.7.3-r12	**主要特性：** • 集群支持创建/纳管 CentOS7.4 操作系统的节点 • kubernetes 的 Service 支持对接 DNAT 网关服务 • NetworkPolicy 能力开放 • 增强型 ELB 支持 Service 配置多个端口 **问题修复：** • 修复 kubernetes 资源回收过程中连不上 kube-apiserver 导致 pod 残留的问题 • 修复节点弹性扩容数据不准确的问题 • 事件老化周期提示修正：集群老化周期为 1 小时
v1.7.3-r11	**主要特性：** • 经典型 ELB 支持自定义健康检查端口 • 经典型 ELB 性能优化 • ELB 四层负载均衡支持修改 Service 的端口 • 支持删除命名空间 • 支持 EVS 云硬盘存储解绑 • 支持配置迁移策略 **问题修复：**

Kubernetes 版本 （CCE 增强版）	版本说明
	• 修复网络插件防止健康检查概率死锁问题 • 修复高可用集群 haproxy 连接数限制问题
v1.7.3-r10	**主要特性：** • 容器网络支持 Overlay L2 模式 • 集群节点支持 GPU 类型虚机 • 集群节点支持 CentOS 7.1 操作系统，支持操作系统选择 • Windows 集群支持对接二代 ELB • 支持弹性文件服务 SFS 导入 • 物理机场景支持对接 SFS 文件存储、OBS 对象存储
v1.7.3-r9	**主要特性：** • 工作负载支持跨 AZ 部署 • 容器存储支持 OBS 对象存储服务 • 支持 ELB L7 负载均衡 • Windows 集群支持 EVS 存储 • 物理机场景支持 devicemapper direct-lvm 模式
v1.7.3-r8	**主要特性：** • 集群支持节点弹性扩容
v1.7.3-r7	**主要特性：** • 容器隧道网络集群支持纳管 SUSE 12sp2 节点 • docker 支持 direct-lvm 模式挂载 devicemapper • 集群支持安装 dashboard • 支持创建 Windows 集群
v1.7.3-r6	**主要特性：** • 集群存储对接原生 EVS 接口
v1.7.3-r5	**主要特性：** • 支持创建 HA 高可靠集群 **问题修复：** • 节点重启后容器网络不通
v1.7.3-r4	**主要特性：** • 集群性能优化 • 物理机场景支持对接 ELB
v1.7.3-r3	**主要特性：** • 容器存储支持 KVM 虚拟机挂载

Kubernetes 版本（CCE 增强版）	版本说明
v1.7.3-r2	**主要特性：** • 容器存储支持 SFS 文件存储 • 工作负载支持自定义应用日志 • 开放工作负载优雅缩容 **问题修复：** • 修复容器存储 AK/SK 会过期的问题
v1.7.3-r1	**主要特性：** • kube-dns 支持外部域名解析
v1.7.3-r0	**主要特性：** • Kubernetes 同步社区 1.7.3 版本 • 支持 ELB 负载均衡 • 容器存储支持 XEN 虚拟机挂载 • 容器存储支持 EVS 云硬盘存储

4.2 购买集群

4.2.1 CCE Turbo 集群与 CCE 集群的区别

CCE 支持多种类型的集群创建，以满足您各种业务需求，如下为 CCE Turbo 集群与 CCE 集群区别：

表4-9 集群类型对比

维度	子维度	CCE Turbo 集群	CCE 集群
集群	定位	面向云原生 2.0 的新一代容器集群产品，计算、网络、调度全面加速	标准版本集群，提供商用级的容器集群服务
	节点形态	支持虚拟机和物理机混合	支持虚拟机和物理机混合
网络	网络模型	**云原生网络 2.0**：面向大规模和高性能的场景。 组网规模最大支持 2000 节点	**云原生网络 1.0**：面向性能和规模要求不高的场景。 • 容器隧道网络模式 • VPC 网络模式
	网络性能	VPC 网络和容器网络融合，性能无损耗	VPC 网络叠加容器网络，性能有一定损耗

维度	子维度	CCE Turbo 集群	CCE 集群
	容器网络隔离	Pod 可直接关联安全组，基于安全组的隔离策略，支持集群内外部统一的安全隔离。	• 隧道网络模式：集群内部网络隔离策略，支持 Networkpolicy。 • VPC 网络模式：不支持
安全	隔离性	• 物理机：安全容器，支持虚机级别的隔离 • 虚拟机：普通容器	普通容器，Cgroups 隔离

4.2.2 购买集群

您可以通过云容器引擎控制台非常方便快速的创建 Kubernetes 集群。Kubernetes 是大规模容器集群管理软件，一个集群可以管理一组节点资源。

CCE 集群支持虚拟机与物理机混合、支持 GPU 等异构节点的混合部署，基于高性能网络模型提供全方位、多场景、安全稳定的容器运行环境，您可以实现多种场景的混合部署。

约束与限制

- 创建节点过程中会使用域名方式从 OBS 下载软件包，需要能够使用云上内网 DNS 解析 OBS 域名，否则会导致创建不成功。为此，节点所在子网需要配置为内网 DNS 地址，从而使得节点使用内网 DNS。在创建子网时 DNS 默认配置为内网 DNS，如果您修改过子网的 DNS，请务必**确保子网下的 DNS 服务器可以解析 OBS 服务域名**，否则需要将 DNS 改成内网 DNS。
- 集群一旦创建以后，不支持变更以下项：
  - 变更集群类型。
  - 变更集群的控制节点数量。
  - 变更控制节点可用区。
  - 变更集群的网络配置，如所在的虚拟私有云 VPC、子网、容器网段、服务网段、IPv6、kubeproxy 代理（转发）模式。
  - 变更网络模型，例如“容器隧道网络”更换为“VPC 网络”。

操作步骤

步骤 1 登录 CCE 控制台。在“集群管理”页面选择需要创建的集群类型，单击“购买”。

说明

- 当前仅苏州、广州 4 支持 CCE turbo 集群类型。

步骤 2 填写集群参数。

基础配置

- 集群名称。

- 企业项目：

  该参数仅对开通企业项目的企业客户帐号显示。

  选择某企业项目（如：default）后，集群、集群下节点、集群安全组、节点安全组和自动创建的节点 EIP（弹性公网 IP）将创建到所选企业项目下。为方便管理资源，在集群创建成功后，建议不要修改集群下节点、集群安全组、节点安全组的企业项目。

  企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。了解更多企业项目相关信息，请查看企业管理。
- 集群版本：选择集群使用的 Kubernetes 版本。
- 集群规模：集群支持管理的最大节点数量，请根据业务场景选择。创建完成后支持扩容，不支持缩容。
- 高可用：控制节点分布方式，默认随机分配，控制节点尽可能随机分布在不同可用区以提高容灾能力 。

  您还可以展开高级配置自定义控制节点分布方式，支持如下 2 种方式。
  - 随机分配：通过把控制节点随机创建在不同的可用区中实现容灾。
  - 自定义：自定义选择每台控制节点的位置。
    - 主机：通过把控制节点创建在相同可用区下的不同主机中实现容灾。
    - 自定义：用户自行决定每台控制节点所在的位置。

网络配置

集群网络涉及节点、容器和服务，强烈建议您详细了解集群的网络以及容器网络模型，具体请参见网络概述。

- 网络模型：CCE 集群支持“VPC 网络”和“容器隧道网络”，详细介绍请参见 VPC 网络和 容器隧道网络。CCE Turbo 集群支持“云原生网络 2.0”。
- 虚拟私有云：选择集群所在的虚拟私有云 VPC，如没有可选项可以单击右侧“新建虚拟私有云”创建。创建后不可修改。

说明

- 1.15 及以上版本容器隧道网络的 CCE 集群支持开启 IPv4/IPv6 双栈。
- v1.23.8-r0 及以上版本的 CCE Turbo 集群支持开启 IPv4/IPv6 双栈。

- 控制节点子网：选择控制节点（即集群 Master 节点）所在子网，如没有可选项可以单击右侧“新建子网”创建。创建后控制节点子网不可修改。
- 容器网段：设置容器使用的网段。VPC 网络模型支持添加多个容器网段，且支持集群创建后添加容器网段。
- 默认容器子网（CCE Turbo 集群设置）：选择容器所在子网，如没有可选项可以单击右侧“新建子网”创建。容器子网决定了集群下容器的数量上限，创建后支持新增子网。
- 服务网段：同一集群下容器互相访问时使用的 Service 资源的网段。决定了 Service 资源的上限。 创建后不可修改。

高级配置

- 服务转发模式：支持 IPVS 和 iptables 两种转发模式，具体请参见 iptables 与 IPVS 如何选择。
- CPU 管理策略：具体请参见 CPU 绑核配置。
- 证书认证：
  - 系统默认：默认开启 X509 认证模式，X509 是一种非常通用的证书格式。
  - 自定义：您可以将自定义证书添加到集群中，用自定义证书进行认证。
    您需要分别上传自己的 **CA 根证书**、**客户端证书**和**客户端证书私钥**。

⚠ 注意

- 请上传**小于 1MB** 的文件，CA 根证书和客户端证书上传格式支持**.crt 或.cer** 格式，客户端证书私钥仅支持上传**未加密的证书私钥**。
- 客户端证书有效期需要 5 年以上。
- 上传的 CA 根证书既给认证代理使用，也用于配置 kube-apiserver 聚合层，**如不合法，集群将无法成功创建**。
- 从 1.25 版本集群开始，Kubernetes 不再支持使用 SHA1WithRSA、ECDSAWithSHA1 算法生成的证书认证，推荐使用 SHA256 算法生成的证书进行认证。

步骤 3 单击“下一步：插件配置”，配置插件。

默认安装 CoreDNS 和 Everest 两个插件。

业务日志

- ICAgent 日志采集：
  由应用运维服务 AOM 提供的日志采集器，配置采集规则后可同时将日志上报至 AOM 服务和云日志服务 LTS。
  默认选择使用 ICAgent 采集容器标准输出日志。

步骤 4 参数填写完成后，单击“下一步：规格确认”，显示集群资源清单，确认无误后，单击“提交”。

集群创建预计需要 6-10 分钟，您可以单击“返回集群管理”进行其他操作或单击“查看集群事件列表”后查看集群详情。

----结束

相关操作

- 添加节点：集群创建完成后，若您需要为集群添加节点，请参见创建节点。

4.2.3 iptables 与 IPVS 如何选择

kube-proxy 是 Kubernetes 集群的关键组件，负责 Service 和其后端容器 Pod 之间进行负载均衡转发。

CCE 当前支持 iptables 和 IPVS 两种转发模式，各有优缺点。

- IPVS: 吞吐更高，速度更快的转发模式。适用于集群规模较大或 Service 数量较多的场景。
- iptables: 社区传统的 kube-proxy 模式。适用于 Service 数量较少或客户端会出现大量并发短链接的场景。

约束与限制

IPVS 模式集群下，Ingress 和 Service 使用相同 ELB 实例时，无法在集群内的节点和容器中访问 Ingress，因为 kube-proxy 会在 ipvs-0 的网桥上挂载 LB 类型的 Service 地址，Ingress 对接的 ELB 的流量会被 ipvs-0 网桥劫持。建议 Ingress 和 Service 使用不同 ELB 实例。

iptables

iptables 是一个 Linux 内核功能，提供了大量的数据包处理和过滤方面的能力。它可以在核心数据包处理管线上用 Hook 挂接一系列的规则。iptables 模式中 kube-proxy 在 NAT pre-routing Hook 中实现它的 NAT 和负载均衡功能。

kube-proxy 的用法是一种 O(n) 算法，其中的 n 随集群规模同步增长，这里的集群规模，更明确的说就是服务和后端 Pod 的数量。

IPVS

IPVS(IP Virtual Server)是在 Netfilter 上层构建的，并作为 Linux 内核的一部分，实现传输层负载均衡。IPVS 可以将基于 TCP 和 UDP 服务的请求定向到真实服务器，并使真实服务器的服务在单个 IP 地址上显示为虚拟服务。

IPVS 模式下，kube-proxy 使用 IPVS 负载均衡代替了 iptables。这种模式同样有效，IPVS 的设计就是用来为大量服务进行负载均衡的，它有一套优化过的 API，使用优化的查找算法，而不是简单的从列表中查找规则。

kube-proxy 在 IPVS 模式下，其连接过程的复杂度为 O(1)。换句话说，多数情况下，他的连接处理效率是和集群规模无关的。

IPVS 包含了多种不同的负载均衡算法，例如轮询、最短期望延迟、最少连接以及各种哈希方法等。而 iptables 就只有一种随机平等的选择算法。

IPVS 相较于 iptables 的优势大致如下：

1. 为大型集群提供了更好的可扩展性和性能
2. 支持比 iptables 更好的负载均衡算法
3. 支持服务器健康检查和连接重试等功能

4.3 访问集群

4.3.1 通过 kubectl 连接集群

操作场景

本文将以 CCE 集群为例，介绍如何通过 kubectl 连接 CCE 集群。

权限说明

kubectl 访问 CCE 集群是通过集群上生成的配置文件（kubeconfig.json）进行认证，kubeconfig.json 文件内包含用户信息，CCE 根据用户信息的权限判断 kubectl 有权限访问哪些 Kubernetes 资源。即哪个用户获取的 kubeconfig.json 文件，kubeconfig.json 就拥有哪个用户的信息，这样使用 kubectl 访问时就拥有这个用户的权限。

使用 kubectl 连接集群

若您需要从客户端计算机连接到 kubernetes 集群，可使用 kubernetes 命令行客户端 kubectl，您可登录 CCE 控制台，单击待连接集群名称，在“集群信息”页面查看访问地址以及 kubectl 的连接步骤，如下图所示。

CCE 支持“VPC 网络内访问”和“互联网访问”两种方式访问集群。

- VPC 网络内访问：即通过内网地址访问，访问集群的客户端机器需要位于集群所在的同一 VPC 内。
- 互联网访问：即通过公网地址访问，访问集群的客户端机器需要具备访问公网的能力，并为集群绑定公网地址。

须知

通过“互联网访问”方式访问集群，您需要在集群信息页中的“连接信息”版块为集群绑定公网地址。绑定公网集群的 kube-apiserver 将会暴露到互联网，存在被攻击的风险，请谨慎开放该能力。

图4-3 集群连接信息

您需要先下载 kubectl 以及配置文件，拷贝到您的客户端机器，完成配置后，即可以访问 kubernetes 集群。使用 kubectl 连接集群的步骤如下：

步骤 1 下载 kubectl

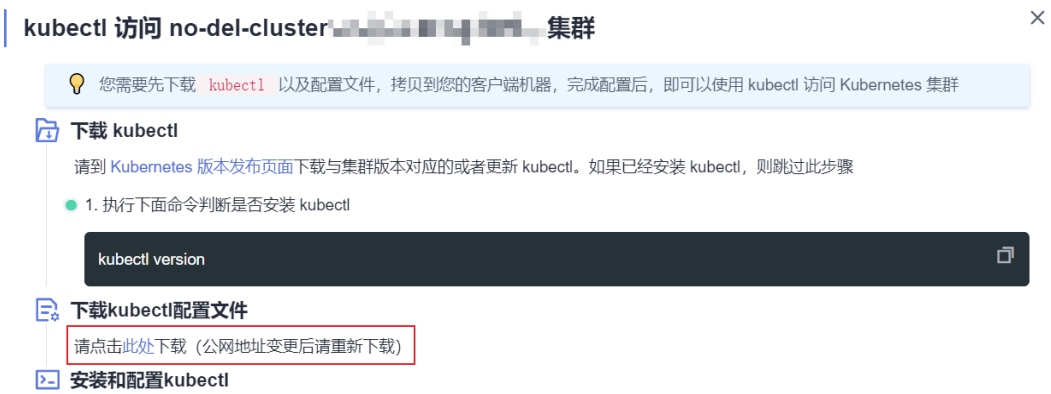
在 kubernetes 版本发布页面，根据集群版本单击对应链接，然后单击 **Client Binaries**，选择对应平台软件包下载即可。如需通过命令行方式安装 kubectl，请参考安装 kubectl。

图4-4 下载 kubectl

步骤 2 获取 kubectl 配置文件

在集群信息页中的“连接信息”版块，单击 kubectl 后的“点击查看”按钮，查看 kubectl 的连接信息，并在弹出页面中下载配置文件。

图4-5 下载配置文件



说明

- kubectl 配置文件（kubeconfig.json）用于对接认证集群，请您妥善保存该认证凭据，防止文件泄露后，集群有被攻击的风险。
- 当前集群默认不开启域名双向认证说明，可通过 kubectl config use-context externalTLSVerify 命令开启双向认证。对已经绑定了 EIP 的集群，如果在使用双向认证时出现认证不通过的情况（x509: certificate is valid），需要重新绑定 EIP 并重新下载 kubeconfig.json。
- IAM 用户下载的配置文件所拥有的 Kubernetes 权限与 CCE 控制台上 IAM 用户所拥有的权限一致。
- 如果 Linux 系统里面配置了 KUBECONFIG 环境变量，kubectl 会优先加载 KUBECONFIG 环境变量，而不是$home/.kube/config，使用时请注意。

步骤 3 配置 kubectl

以 Linux 环境为例安装和配置 kubectl。

1. 拷贝步骤 1 中下载的 kubectl 及步骤 2 中下载的配置文件到您客户端机器的/home 目录下。
2. 登录到您的客户端机器，配置 kubectl。如果已经安装 kubectl，则跳过此步骤。

```
cd /home
chmod +x kubectl
mv -f kubectl /usr/local/bin
```

3. 登录到您的客户端机器，配置 kubectl 配置文件。

```
cd /home
mkdir -p $HOME/.kube
mv -f kubeconfig.json $HOME/.kube/config
```

4. 根据使用场景，切换 kubectl 的访问模式。
   - VPC 内网接入访问请执行：

     ```
     kubectl config use-context internal
     ```

   - 互联网接入访问请执行（集群需绑定公网地址）：

     ```
     kubectl config use-context external
     ```

   - 互联网接入访问如需开启双向认证请执行（集群需绑定公网地址）：

     ```
     kubectl config use-context externalTLSVerify
     ```

     关于集群双向认证的说明请参见域名双向认证。

----结束

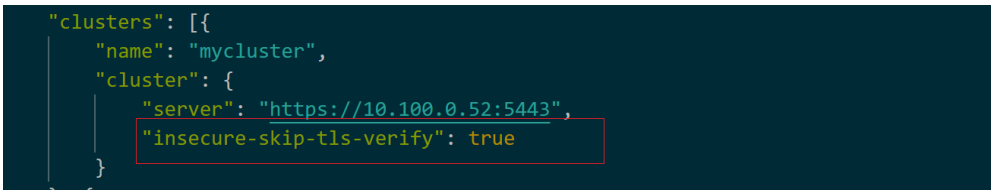
域名双向认证

CCE 当前支持域名双向认证。

- 域名双向认证默认不开启，可通过 **kubectl config use-context externalTLSVerify** 命令切换到 externalTLSVerify 这个 context 开启使用。
- 集群绑定或解绑弹性 IP、配置或更新自定义域名时，集群服务端证书将同步签入最新的集群访问地址（包括集群绑定的弹性 IP、集群配置的所有自定义域名）。
- 异步同步集群通常耗时约 5-10min，同步结果可以在操作记录中查看“同步证书”。

- 对已经绑定了 EIP 的集群，如果在使用双向认证时出现认证不通过的情况（x509: certificate is valid），需要重新绑定 EIP 并重新下载 kubeconfig.json。
- 早期未支持域名双向认证时，kubeconfig.json 中包含"insecure-skip-tls-verify": true 字段，下图所示。如果需要使用双向认证，您可以重新下载 kubeconfig.json 文件并配置开启域名双向认证。

图4-6 未开启域名双向认证



常见问题（Error from server Forbidden）

使用 kubectl 在创建或查询 Kubernetes 资源时，显示如下内容。

kubectl get deploy Error from server (Forbidden): deployments.apps is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "deployments" in API group "apps" in the namespace "default"

原因是用户没有操作该 Kubernetes 资源的权限，请参见命名空间权限（Kubernetes RBAC 授权）为用户授权。

4.3.2 通过 X509 证书连接集群

操作场景

通过控制台获取集群证书，使用该证书可以访问 Kubernetes 集群。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“集群信息”，在右边“连接信息”下证书认证一栏，单击“下载”。

步骤 3 在弹出的“证书获取”窗口中，根据系统提示选择证书的过期时间并下载集群 X509 证书。

须知

- 下载的证书包含 client.key、client.crt、ca.crt 三个文件，请妥善保管您的证书，不要泄露。
- 集群中容器之间互访不需要证书。

步骤 4 使用集群证书调用 Kubernetes 原生 API。

例如使用 curl 命令调用接口查看 Pod 信息，如下所示，其中 192.168.0.18:5443 为集群 API Server 地址。

```
curl --cert ./client.crt --key ./client.key
https://192.168.0.18:5443/api/v1/namespaces/default/pods/
```

更多集群接口请参见 Kubernetes API。

4.3.3 通过自定义域名访问集群

操作场景

集群服务端证书中签入的**主题备用名称（Subject Alternative Name，缩写 SAN）**。SAN 通常在 TLS 握手阶段被用于客户端校验服务端的合法性：服务端证书是否被客户端信任的 CA 所签发，且证书中的 SAN 是否与客户端实际访问的 IP 地址或 DNS 域名匹配。

当客户端无法直接访问集群内网私有 IP 地址或者公网弹性 IP 地址时，您可以将客户端可直接访问的 IP 地址或者 DNS 域名签入集群服务端证书，以支持客户端开启双向认证，提高安全性。典型场景例如 DNAT 访问、域名访问等特殊场景。

域名访问场景的典型使用方式如下：

- 客户端配置 Host 域名指定 DNS 域名地址，或者客户端主机配置/etc/hosts，添加域名映射。

- 云上内网使用，云解析服务 DNS 支持配置集群弹性 IP 与自定义域名的映射关系。后续更新弹性 IP 可以继续使用双向认证+自定义域名访问集群，无需重新下载 kubeconfig.json 配置文件。
- 自建 DNS 服务器，自行添加 A 记录。

约束与限制

仅支持 1.19 及以上版本集群。

添加自定义 SAN

步骤 1 登录 CCE 控制台。

步骤 2 在集群列表中单击集群，进入集群详情页。

步骤 3 在连接信息的自定义 SAN 处单击，在弹出的窗口中添加 IP 地址或域名，然后单击“保存”。

说明

1. 当前操作将会短暂重启 kube-apiserver 并更新 kubeConfig.json 文件，请避免在此期间操作集群。
2. 请输入域名或 ip，以英文逗号（,）分隔，最多 128 个。
3. 自定义域名如需绑定弹性公网，请确保已配置公网地址。

----结束

4.4 集群升级

4.4.1 集群升级概述

云容器引擎（CCE）严格遵循社区一致性认证。为了能够方便您使用稳定又可靠的 Kubernetes 版本，请您务必在维护周期结束之前升级您的 Kubernetes 集群。

CCE 在发布 Kubernetes 最新版本后，会对该版本所做的变更进行说明。

您可以通过云容器引擎管理控制台，可视化升级集群的 Kubernetes 版本。

升级前，您需要在集群列表页面确认集群的 Kubernetes 版本，以及当前是否有新的版本可供升级。

确认方法如下：

登录 CCE 控制台，查看待升级集群左下角是否存在“存在更新版本”提示，若存在则该集群支持升级，若不存在，则该集群不支持升级。

图4-7 集群-可升级

集群升级路径

如下表格详细介绍了各集群版本能够升级到的目标版本，以及升级方式和升级影响。

表4-10 集群升级路径和影响说明

源版本	目标版本	升级方式	升级影响
v1.19	v1.21	原地升级	需用户自行识别各版本差异，详情请参见大版本升级须知。
v1.17 v1.15	v1.19	原地升级	需用户自行识别各版本差异，详情请参见大版本升级须知。
v1.13	v1.15	滚动升级 重置升级	• coredns 配置中 proxy 配置项不支持，需要替换成 forward。 • 存储插件从原来的 storage-driver 替换成了 Everest。
v1.11 v1.9	Console 中可创建的最新版本	手动迁移	需用户自行识别各版本差异，详情请参见大版本升级须知。
v1.9.2 v1.7	Console 中可创建的最新版本	手动迁移	需用户自行识别各版本差异，详情请参见大版本升级须知。

升级方式

在集群升级过程中，Master 节点的升级流程都是一致的，而 Node 节点的升级流程存在区别，不同升级方式的优缺点如下：

表4-11 升级方式的区别和优缺点

升级名称	方式	优点	缺点
原地升级	节点上升级 Kubernetes 组件、网络组件和 CCE 管理组件，升级过程中业务 Pod 和网络均不受影响；升级过程中，存量节点将全部打上 SchedulingDisabled 标签，升级完成后，用户能够正常使用存量节点。	用户无需迁移业务，可以基本上保证业务不断。	原地升级不会升级节点的操作系统，如果希望升级操作系统，可以在节点升级完成后，清空对应的节点数据，并执行节点重置，升级到新版本的操作系统。
滚动升级	节点上只升级 Kubernetes 组件以及网络部分组件，存量节点将全部打上 SchedulingDisabled 标签，仅保证原本运行的应用不受影响。 须知 • **升级完成后，用户还需手动新建节点，并逐步释放老节点**，将应用逐步迁移到新节点上，用户控制升级步骤。	可以基本上保证业务不断。	• **升级完成后，用户需手动新建节点，并逐步释放老节点，将业务迁移至新节点。**该过程中，新建节点会存在额外的费用。待业务迁移至新节点后，老节点可以删除。 • 滚动升级完成后，如果需要继续向高版本升级，需要先完成老节点的重置，否则无法通过升级前检查。该过程可能会引起业务中断。
重置升级	对工作节点使用最新的 node 镜像进行操作系统重置。	时间最短，用户介入操作也较少。	如果节点上有数据或者配置，会丢失，同样会有一段时间业务受损。

大版本升级须知

大版本升级路径	版本差异	建议自检措施
v1.19 升级至	Kubernetes v1.21 集群版本修复 了 exec probe timeouts 不生效的 BUG，在此修复之前，exec 探测	升级前检查您使用了 exec probe 的应用的 probe timeouts 是否合理。

大版本升级路径	版本差异	建议自检措施
v1.21	器不考虑 timeoutSeconds 字段。相反，探测将无限期运行，甚至超过其配置的截止日期，直到返回结果。 若用户未配置，默认值为 1 秒。升级后此字段生效，如果探测时间超过 1 秒，可能会导致应用健康检查失败并频繁重启。	
	CCE 的 v1.19 及以上版本的 kube-apiserver 要求客户侧 webhook server 的证书必须配置 Subject Alternative Names (SAN)字段。否则升级后 kube-apiserver 调用 webhook server 失败，容器无法正常启动。 根因：Go 语言 v1.15 版本废弃了 X.509 CommonName，CCE 的 v1.19 版本的 kube-apiserver 编译的版本为 v1.15，若客户的 webhook 证书没有 Subject Alternative Names (SAN)，kube-apiserver 不再默认将 X509 证书的 CommonName 字段作为 hostname 处理，最终导致认证失败。	升级前检查您自建 webhook server 的证书是否配置了 SAN 字段。 • 若无自建 webhook server 则不涉及。 • 若未配置，建议您配置使用 SAN 字段指定证书支持的 IP 及域名。
v1.15 升级至 v1.19	CCE v1.19 版本的控制面与 v1.15 版本的 Kubelet 存在兼容性问题。若 Master 节点升级成功后，节点升级失败或待升级节点发生重启，则节点有极大概率为 NotReady 状态。 主要原因为升级失败的节点有大概率重启 kubelet 而触发节点注册流程，v1.15 kubelet 默认注册标签（failure-domain.beta.kubernetes.io/is-baremetal 和 kubernetes.io/availablezone）被 v1.19 版本 kube-apiserver 视为非法标签。 v1.19 版本中对应的合法标签为 node.kubernetes.io/baremetal 和 failure-domain.beta.kubernetes.io/zone。	1. 正常升级流程不会触发此场景。 2. 在 Master 升级完成后尽量避免使用暂停升级功能，快速升级完 Node 节点。 3. 若 Node 节点升级失败且无法修复，请尽快驱逐此节点上的应用，请联系技术支持人员，跳过此节点升级，在整体升级完毕后，重置该节点。

大版本升级路径	版本差异	建议自检措施
	CCE 的 v1.15 版本集群及 v1.19 版本集群将 docker 的存储驱动文件系统由 xfs 切换成 ext4，可能会导致升级后的 java 应用 Pod 内的 import 包顺序异常，既而导致 Pod 异常。	升级前查看节点上 docker 配置文件 /etc/docker/daemon.json。检查 dm.fs 配置项是否为 xfs。 • 若为 ext4 或存储驱动为 overlay 则不涉及。 • 若为 xfs 则建议您在新版本集群预先部署应用，以测试应用与新版本集群是否兼容。 `{ "storage-driver": "devicemapper", "storage-opts": [ "dm.thinpooldev=/dev/mapper/vgpaas-thinpool", "dm.use_deferred_removal=true", "dm.fs=xfs", "dm.use_deferred_deletion=true" ] }`
	CCE 的 v1.19 及以上版本的 kube-apiserver 要求客户侧 webhook server 的证书必须配置 Subject Alternative Names (SAN)字段。否则升级后 kube-apiserver 调用 webhook server 失败，容器无法正常启动。 根因：Go 语言 v1.15 版本废弃了 X.509 CommonName，CCE 的 v1.19 版本的 kube-apiserver 编译的版本为 v1.15。CommonName 字段作为 hostname 处理，最终导致认证失败。	升级前检查您自建 webhook server 的证书是否配置了 SAN 字段。 • 若无自建 webhook server 则不涉及。 • 若未配置，建议您配置使用 SAN 字段指定证书支持的 IP 及域名。 **须知** 为减弱此版本差异对集群升级的影响，v1.15 升级至 v1.19 时，CCE 会进行特殊处理，仍然会兼容支持证书不带 SAN。但后续升级不再特殊处理，请尽快整改证书，以避免影响后续升级。
	v1.17.17 版本及以后的集群 CCE 自动给用户创建了 PSP 规则，限制了不安全配置的 Pod 的创建，如 securityContext 配置了 sysctl 的 net.core.somaxconn 的 Pod。	升级后请按需开放非安全系统配置。
v1.13 升级至 v1.15	vpc 集群升级后，由于网络组件的升级，master 节点会额外占一个网段。在 Master 占用了网段后，无	一般集群内节点数量快占满容器网段场景下会出现该问题。例如，容器网段为 10.0.0.0/16，可用 IP 数量

大版本升级路径	版本差异	建议自检措施
	可用容器网段时，新建节点无法分配到网段，调度在该节点的 pod 会无法运行。	为 65536，VPC 网络 IP 分配是分配固定大小的网段（使用掩码实现，确定每个节点最多分配多少容器 IP），例如上限为 128，则此时集群最多支撑 65536/128=512 个节点，然后去掉 Master 节点数量为 509，此时是 1.13 集群支持的节点数。集群升级后，在此基础上 3 台 Master 节点会各占用 1 个网段，最终结果就是 506 台节点。

4.4.2 升级前须知

升级前，您可以在 CCE 控制台确认您的集群是否可以进行升级操作。确认方法请参见集群升级概述。

注意事项

- **集群升级操作不可回退，请务必慎重并选择合适的时间段进行升级，以减少升级对您的业务带来的影响。**
- 集群升级前请参考集群大版本发布说明了解每个集群版本发布的特性以及差异，否则可能因为应用不兼容新集群版本而导致升级后异常。
- 集群升级中**请勿关机、重启或删除节点**，否则会导致升级失败。
- 集群升级前**请关闭弹性扩缩容策略**，避免在升级过程中扩缩容节点，从而导致升级失败。
- 如果您本地修改了集群节点的配置，可能导致集群升级失败或升级后配置丢失，建议您通过集群的配置管理和节点池的配置管理修改配置，以便在升级时自动继承。
- 集群升级过程中，已运行工作负载业务不会中断，但 API Server 访问会短暂中断，如果业务需要访问 API Server 可能会受到影响。
- 请在升级集群前，请查看集群状态是否均为健康状态。若集群不正常，您可以自行修复，若仍有问题请提交工单联系我们协助您进行修复。
- 为了您的数据安全，强烈建议您先备份数据然后再升级，升级过程中不建议对集群进行任何操作。
- 集群升级过程中，将会给节点打上“node.kubernetes.io/upgrade”（效果为“NoSchedule”）的污点，并在集群升级完成后移除该污点。请您不要在节点上手动添加相同键名的污点，即使污点效果不同，也可能会在升级后被系统误删除。

约束与限制

- 当前仅支持虚拟机节点的 CCE 集群升级，暂不支持鲲鹏集群、CCE Turbo 集群、使用私有镜像的 CCE 集群升级。

- 集群升级后，如版本特性中修复了容器引擎 Containerd 相关漏洞，存量节点需要手动重启 Containerd 才可以生效，对于存量 Pod 同样需要通过重启 Pod 才能生效。
- 1.15 版本集群原地升级，如果业务中存在 initContainer 或使用 Istio 的场景，则需要注意以下约束：

  1.16 及以上的 kubelet 统计 QosClass 和之前版本存在差异，1.15 及以下版本仅统计 spec.containers 下的容器，1.16 及以上的 kubelet 版本会同时统计 spec.containers 和 spec.initContainers 下的容器，升级前后会造成 Pod 的 QosClass 变化，从而造成 Pod 中容器重启。建议参考下表在升级前修改业务容器的 QosClass 规避该问题。

表4-12 升级前后 QosClass 变化

init 容器（根据 spec.initContainers 计算）	业务容器（根据 spec.containers 计算）	Pod（根据 spec.containers 和 spec.initContainers 计算）	是否受影响
Guaranteed	Besteffort	Burstable	是
Guaranteed	Burstable	Burstable	否
Guaranteed	Guaranteed	Guaranteed	否
Besteffort	Besteffort	Besteffort	否
Besteffort	Burstable	Burstable	否
Besteffort	Guaranteed	Burstable	是
Burstable	Besteffort	Burstable	是
Burstable	Burstable	Burstable	否
Burstable	Guaranteed	Burstable	是

升级备份说明

目前集群升级备份方式有两种：

- **集群 ETCD 数据库备份**：CCE 服务会在集群升级流程中对 etcd 数据库进行备份，无需用户确认。
- **Master 节点整机备份**：在升级界面对集群的 Master 节点进行整机备份，**需要用户手动确认**，备份过程会使用云备份服务，备份通常耗时在 20 分钟左右，若当前局点云备份任务排队较多时，备份时间可能同步延长，推荐用户使用进行整机备份。

4.4.3 升级前检查

4.4.3.10 节点限制检查

检查项内容

当前检查项包括以下内容：

- 检查节点是否可用
- 检查节点操作系统是否支持升级
- 检查节点是否含有非预期的节点池标签
- 检查 K8S 节点名称是否与云主机保持一致

解决方案

- **问题场景一：节点不可用**

  若检查发现节点不可用，请登录 CCE 控制台，前往“集群信息->节点管理”处查看节点状态，请确保节点处于“运行中”状态。节点处于“安装中”、“删除中”状态时，均不支持升级。

  若节点状态异常，请修复节点后，重试检查任务。

- **问题场景二：节点操作系统升级支持受限**

  当前集群升级支持的节点操作系统范围如下表所示，若您的节点 OS 不在支持列表之内，暂时无法升级。您可将节点重置为列表中可用的操作系统。

表4-13 节点 OS 支持列表

操作系统	限制
CentOS	无限制
Ubuntu	部分局点受限，若检查结果不支持升级，请联系技术支持人员

- **问题场景三：节点含有非预期的节点池标签**

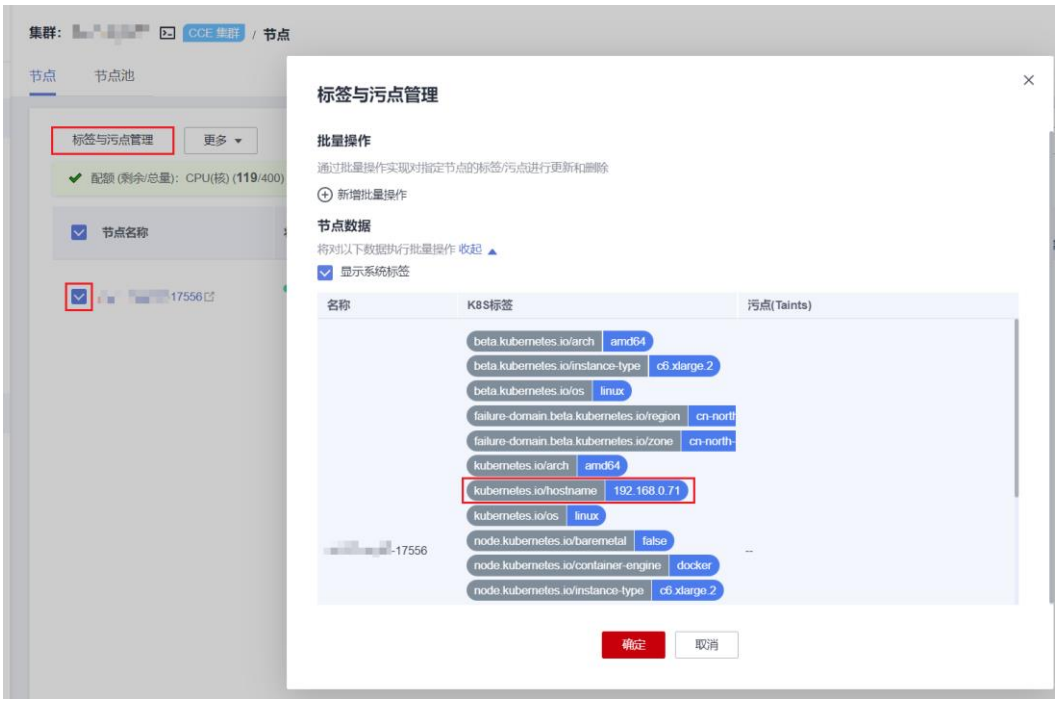
  由节点池迁移至默认节点池的节点，仍然会保留节点池标签"cce.cloud.com/cce-nodepool"，影响集群升级。请确认该节点上的负载调度是否依赖改标签：

  - 若无依赖，请删除该标签。
  - 若存在依赖，请修改负载调度策略，解除依赖后再删除该标签。

- **问题场景四：K8S 节点名称与云主机一致**

CCEK8S 节点名称默认“与节点私有 IP 保持一致”，若创建节点时选择“与云主机名称保持一致”，当前暂不支持升级。

请登录 CCE 控制台，前往“集群信息->节点管理”处查看节点标签，对比节点标签 kubernetes.io/hostname 的值是否与云主机名称是否一致。如果一致，则需在集群升级前移除该节点。



4.4.3.11 黑名单检查

检查项内容

当前检查用户是否处于升级黑名单列表中。

解决方案

CCE 基于以下几点原因暂时关闭该集群升级功能：

- 基于客户提供的信息，该集群被识别为核心重点保障生产集群
- 正在或即将进行提高集群稳定性的其他运维任务，例如 Master 节点 3AZ 改造等。

请联系技术支持人员了解限制原因并申请解除升级限制。

4.4.3.12 插件检查

检查项内容

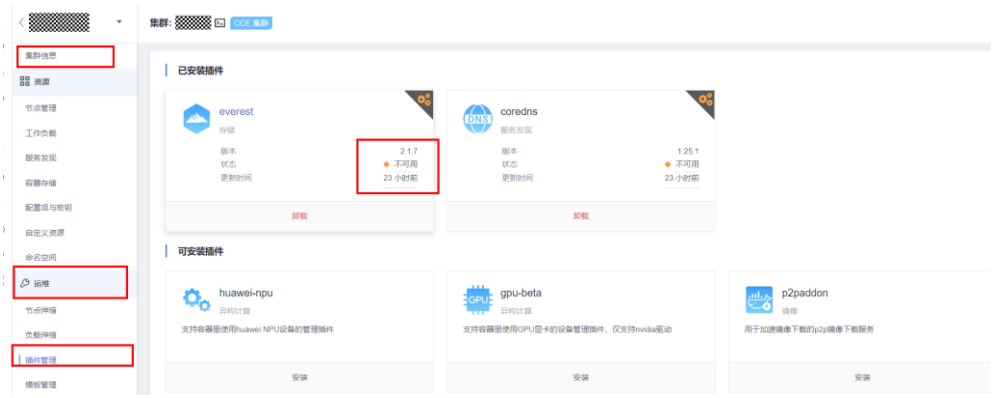
当前检查项包括以下内容：

- 检查插件状态是否正常
- 检查插件是否支持目标版本

解决方案

- **问题场景一：插件状态异常**

  请登录 CCE 控制台，前往“集群信息->运维->插件管理”处查看并处理处于异常状态的插件。



- **问题场景二：目标集群版本不支持当前插件版本**

  检查到该插件由于兼容性等问题无法随集群自动升级，请您登陆 CCE 控制台，在“集群信息->运维->插件管理”处进行手动升级。

4.4.3.13 Helm 模板检查

检查项内容

当前检查当前 HelmRelease 记录中是否含有目标集群版本不支持的 K8S 废弃 API，可能导致升级后 helm 模板不可用。

解决方案

将 HelmRelease 记录中 K8S 废弃 API 转换为源版本和目标版本均兼容的 API。

说明

该检查项解决方案已在升级流程中自动兼容处理，此检查不再限制。您无需关注并处理。

4.4.3.14 Master 节点 SSH 联通性检查

检查项内容

检查当前 CCE 是否能连接至您的 Master 节点。

解决方案

请通过工单方式，联系技术支持人员排查。

4.4.3.15 节点池检查

检查项内容

当前检查项包括以下内容：

- 检查节点池状态是否正常
- 检查节点池弹性伸缩能力是否关闭

解决方案

- **问题场景一：节点池状态异常**

  请登录 CCE 控制台，前往“集群信息->资源->节点管理->节点池”，查看问题节点池状态。若该节点池状态处于伸缩中，请等待节点池伸缩完毕，并参考问题场景二关闭节点池弹性伸缩能力。

- **问题场景二：节点池弹性伸缩能力未关闭**

  解决方案一（推荐）

  请登录 CCE 控制台，前往“集群信息->运维->插件管理->autoscaler 插件”处，卸载该插件。

说明

卸载 autoscaler 插件前，请单击“升级”按钮，备份当前插件配置，便于重装时还原插件配置。

卸载 autoscaler 插件前，请进入“运维->节点伸缩页面”，备份当前伸缩策略，便于重装时还原节点伸缩策略，这些策略会随 autoscaler 插件卸载而清除。

节点伸缩策略：通过编辑按钮获取并备份

解决方案二

若您并不希望卸载 autoscaler 插件，请登录 CCE 控制台，前往“集群信息->运维->节点管理->节点池”，单击对应节点池的“编辑”按钮，关闭弹性伸缩功能。

说明

关闭弹性伸缩时，请备份当前节点池弹性伸缩配置，便于开启时还原配置。

4.4.3.16 安全组检查

检查项内容

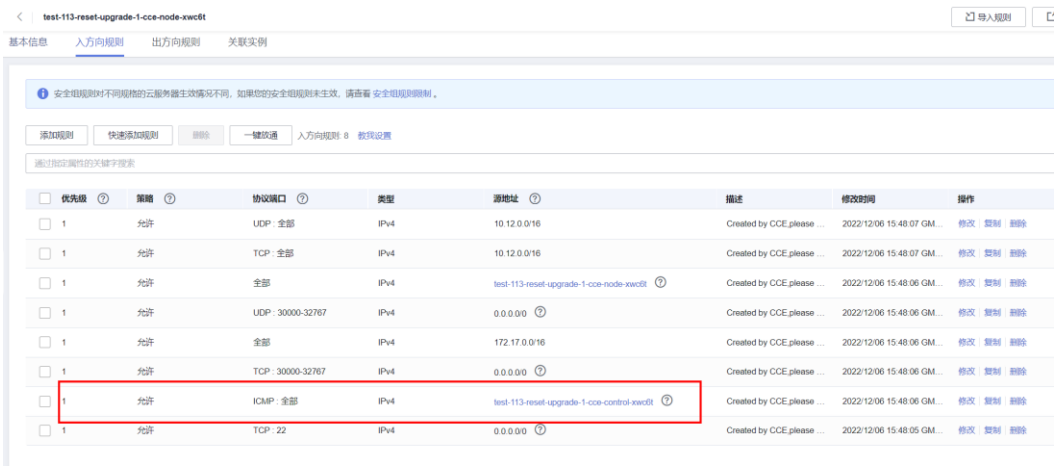
检查当前用户节点安全组是否允许 Master 节点使用 ICMP 协议访问节点。

解决方案

请登录 VPC 控制台，前往“访问控制->安全组”，在搜索框内输入集群名称，此时预期过滤出两个安全组：

- 安全组名称为“集群名称-node-xxx”，此安全组关联 CCE 用户节点。
- 安全组名称为“集群名称-control-xxx”，此安全组关联 CCE 控制节点。

单击 node 用户节点安全组，确保含有如下规则允许 Master 节点使用 **ICMP 协议**访问节点。



若不含有该规则请为 node 安全组添加该放通规则，源地址选择 control 安全组，描述信息为"Created by CCE,please don't modify! Used by the master node to access the worker node."。

4.4.3.17 ARM 节点限制检查

检查项内容

当前检查项包括以下内容

- 检查集群是否为鲲鹏集群或混合集群的 Master 节点为 ARM 架构。
- 检查集群是否包含 ARM 架构的节点。

解决方案

- **问题场景一： 集群为鲲鹏集群或混合集群的 Master 节点为 ARM 架构**

  当前集群升级支持鲲鹏集群升级到的最高版本为 v1.19，若您要升级的版本大于 v1.19，暂时无法升级。
- **问题场景二： 集群包含 ARM 架构的 Node 节点**

  删除 ARM 架构的节点。

4.4.3.18 残留待迁移节点检查

检查项内容

检查节点是否需要迁移。

解决方案

由 1.13 滚动升级而来的 1.15 集群，需要将所有节点迁移（重置或新建替换）后，才允许再次进行升级。

解决方案一

请登录 CCE 控制台，前往“集群信息->资源->节点管理”，单击对应节点的“更多->重置节点”，详情请参见重置节点。节点重置完毕后，重试检查任务。

说明

重置节点会重置所有节点标签，可能影响工作负载调度，请在重置节点前检查并保留您手动为该节点打上的标签。

解决方案二

新建节点后，删除问题节点。

4.4.3.19 K8S 废弃资源检查

检查项内容

检查集群是否存在对应版本已经废弃的资源。

解决方案

问题场景： 1.25 及以上集群废弃了 PodSecurityPolicy 资源对象

在集群中通过 **kubectl get psp -A** 命令获取当前集群中已存在的 PSP 对象。

如果您并未使用这两个对象，可以直接点击确定按钮跳过该检查，若您存在使用该对象的情况，请将 PSP 相应功能升级至 PodSecurity 安全准入能力中。

4.4.3.20 兼容性风险检查

检查项内容

请您阅读版本兼容性差异，并确认不受影响。

补丁升级不涉及版本兼容性差异。

版本兼容性差异

大版本升级路径	版本差异	建议自检措施
v1.19 升级至	Kubernetes v1.21 集群版本修复 了 exec probe timeouts 不生效的	升级前检查您使用了 exec probe 的应

大版本升级路径	版本差异	建议自检措施
v1.21 或 v1.23	BUG，在此修复之前，exec 探测器不考虑 timeoutSeconds 字段。相反，探测将无限期运行，甚至超过其配置的截止日期，直到返回结果。 若用户未配置，默认值为 1 秒。升级后此字段生效，如果探测时间超过 1 秒，可能会导致应用健康检查失败并频繁重启。	用的 probe timeouts 是否合理。
	CCE 的 v1.19 及以上版本的 kube-apiserver 要求客户侧 webhook server 的证书必须配置 Subject Alternative Names (SAN)字段。否则升级后 kube-apiserver 调用 webhook server 失败，容器无法正常启动。 根因：Go 语言 v1.15 版本废弃了 X.509 CommonName，CCE 的 v1.19 版本的 kube-apiserver 编译的版本为 v1.15，若客户的 webhook 证书没有 Subject Alternative Names (SAN)，kube-apiserver 不再默认将 X509 证书的 CommonName 字段作为 hostname 处理，最终导致认证失败。	升级前检查您自建 webhook server 的证书是否配置了 SAN 字段。 • 若无自建 webhook server 则不涉及。 • 若未配置，建议您配置使用 SAN 字段指定证书支持的 IP 及域名。
	1.21 及以上版本不支持管理 ARM 节点。	确认升级后无法管理 ARM 节点不影响您的使用场景。
v1.15 升级至 v1.19	CCE v1.19 版本的控制面与 v1.15 版本的 Kubelet 存在兼容性问题。若 Master 节点升级成功后，节点升级失败或待升级节点发生重启，则节点有极大概率为 NotReady 状态。 主要原因为升级失败的节点有大概率重启 kubelet 而触发节点注册流程，v1.15 kubelet 默认注册标签（failure-domain.beta.kubernetes.io/is-baremetal 和 kubernetes.io/availablezone）被 v1.19 版本 kube-apiserver 视为非法标签。	1. 正常升级流程不会触发此场景。 2. 在 Master 升级完成后尽量避免使用暂停升级功能，快速升级完 Node 节点。 3. 若 Node 节点升级失败且无法修复，请尽快驱逐此节点上的应用，请联系技术支持人员，跳过此节点升级，在整体升级完毕后，重置该节点。

大版本升级路径	版本差异	建议自检措施
	v1.19 版本中对应的合法标签为 node.kubernetes.io/baremetal 和 failure-domain.beta.kubernetes.io/zone。	
	CCE 的 v1.15 版本集群及 v1.19 版本集群将 docker 的存储驱动文件系统由 xfs 切换成 ext4，可能会导致升级后的 java 应用 Pod 内的 import 包顺序异常，既而导致 Pod 异常。	升级前查看节点上 docker 配置文件 /etc/docker/daemon.json。检查 dm.fs 配置项是否为 xfs。 • 若为 ext4 或存储驱动为 overlay 则不涉及。 • 若为 xfs 则建议您在新版本集群预先部署应用，以测试应用与新版本集群是否兼容。 `{ "storage-driver": "devicemapper", "storage-opts": [ "dm.thinpooldev=/dev/mapper/vgpaas-thinpool", "dm.use_deferred_removal=true", "dm.fs=xfs", "dm.use_deferred_deletion=true" ] }`
	CCE 的 v1.19 及以上版本的 kube-apiserver 要求客户侧 webhook server 的证书必须配置 Subject Alternative Names (SAN)字段。否则升级后 kube-apiserver 调用 webhook server 失败，容器无法正常启动。 根因：Go 语言 v1.15 版本废弃了 X.509 CommonName，CCE 的 v1.19 版本的 kube-apiserver 编译的版本为 v1.15。CommonName 字段作为 hostname 处理，最终导致认证失败。	升级前检查您自建 webhook server 的证书是否配置了 SAN 字段。 • 若无自建 webhook server 则不涉及。 • 若未配置，建议您配置使用 SAN 字段指定证书支持的 IP 及域名。 **须知** 为减弱此版本差异对集群升级的影响，v1.15 升级至 v1.19 时，CCE 会进行特殊处理，仍然会兼容支持证书不带 SAN。但后续升级不再特殊处理，请尽快整改证书，以避免影响后续升级。
	v1.17.17 版本及以后的集群 CCE 自动给用户创建了 PSP 规则，限制了不安全配置的 Pod 的创建，如 securityContext 配置了 sysctl 的	升级后请按需开放非安全系统配置。

大版本升级路径	版本差异	建议自检措施
	net.core.somaxconn 的 Pod。	
v1.13 升级至 v1.15	vpc 集群升级后，由于网络组件的升级，master 节点会额外占一个网段。在 Master 占用了网段后，无可用容器网段时，新建节点无法分配到网段，调度在该节点的 pod 会无法运行。	一般集群内节点数量快占满容器网段场景下会出现该问题。例如，容器网段为 10.0.0.0/16，可用 IP 数量为 65536，VPC 网络 IP 分配是分配固定大小的网段（使用掩码实现，确定每个节点最多分配多少容器 IP），例如上限为 128，则此时集群最多支撑 65536/128=512 个节点，然后去掉 Master 节点数量为 509，此时是 1.13 集群支持的节点数。集群升级后，在此基础上 3 台 Master 节点会各占用 1 个网段，最终结果就是 506 台节点。

4.4.3.21 节点 CCEAgent 版本检查

检查项内容

检测当前节点的 CCE 包管理组件 cce-agent 是否为最新版本。

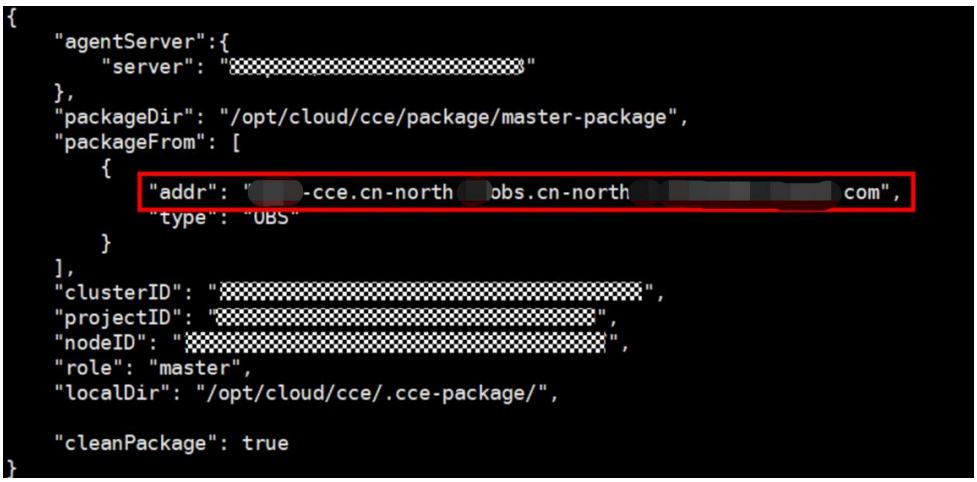
解决方案

cce-agent 非最新版本时，自动更新失败，该问题通常由 OBS 地址失效或组件版本过低引起。

步骤 1 登录检查通过的**正常节点**，获取 cce-agent 配置文件路径，查看有效 OBS 地址。

```
cat `ps aux | grep cce-agent | grep -v grep | awk -F '-f ''{print $2}'`
```

配置文件内的 OBS 配置地址字段为 packageFrom.addr



步骤 2 登陆检查失败的**异常节点**，参考上一步重新获取 OBS 地址，检查是否一致。若不一致，请将异常节点的 OBS 地址修改为正确地址。

步骤 3 通过以下命令下载最新的二进制文件。

- x86 系统

```
curl -k "https://{您获取的 obs 地址}/cluster-versions/base/cce-agent" > /tmp/cce-agent
```

- ARM 系统

```
curl -k "https://{您获取的 obs 地址}/cluster-versions/base/cce-agent-arm" > /tmp/cce-agent-arm
```

步骤 4 替换原有的 cce-agent 二进制文件。

- x86 系统

```
mv -f /tmp/cce-agent /usr/local/bin/cce-agent
chmod 750 /usr/local/bin/cce-agent
chown root:root /usr/local/bin/cce-agent
```

- ARM 系统

```
mv -f /tmp/cce-agent-arm /usr/local/bin/cce-agent-arm
chmod 750 /usr/local/bin/cce-agent-arm
chown root:root /usr/local/bin/cce-agent-arm
```

步骤 5 重启 cce-agent 服务。

```
systemctl restart cce-agent
```

若您对上述执行过程有疑问，请联系技术支持人员。

----结束

4.4.3.22 节点 CPU 使用率检查

检查项内容

检查节点 CPU 使用情况，是否超过 90%。

解决方案

- **请在业务低峰时进行集群升级。**
- 请检查该节点的 Pod 部署数量是否过多，适当驱逐该节点上 Pod 到其他空闲节点。

4.4.3.23 CRD 检查

检查项内容

当前检查项包括以下内容：

- 检查集群关键 CRD "packageversions.version.cce.io"是否被删除。
- 检查集群关键 CRD "network-attachment-definitions.k8s.cni.cncf.io"是否被删除。

解决方案

如出现该检查项异常，请联系技术支持人员。

4.4.3.24 节点磁盘检查

检查项内容

当前检查项包括以下内容：

- 检查节点关键数据盘使用率是否超过 95%
- 检查/tmp 目录是否存在 500MB 可用空间

解决方案

节点升级过程中需要使用磁盘存储升级组件包，使用/tmp 目录存储临时文件。

- **问题场景一：磁盘使用率超过 95%**

  请执行以下检查命令，检查当前各关键磁盘的空间使用情况，删除整理确保各空间使用率均小于 95%后，重试检查。

  - docker 容器运行时磁盘分区

    ```
    df -h /var/lib/docker
    ```

  - contianerd 容器运行时磁盘分区

    ```
    df -h /var/lib/containerd
    ```

  - kubelet 磁盘分区

    ```
    df -h /var/lib/docker
    ```

  - 系统盘

    ```
    df -h /
    ```

- **问题场景二：/tmp 目录空间不足**

  请执行以下检查命令，检查当前/tmp 目录所在文件系统的空间使用情况，删除整理确保空间大于 500MB 后，重试检查。

  ```
  df -h /tmp
  ```

4.4.3.25 节点 DNS 检查

检查项内容

当前检查项包括以下内容：

- 检查当前节点 DNS 配置是否能正常解析 OBS 地址
- 检查当前节点是否能访问存储升级组件包的 OBS 地址

解决方案

节点升级过程中，需要从 OBS 拉取升级组件包。此项检查失败，请联系技术人员支持。

4.4.3.26 节点关键目录文件权限检查

检查项内容

检查关键目录/var/paas 下是否有异常属主和属组的文件。

解决方案

CCE 使用/var/paas 目录进行基本的节点管理活动并存储属主和属组均为 paas 的文件数据。

当前集群升级流程会将/var/paas 路径下的文件的属主和属组均重置为 paas。

请您排查当前业务 Pod 中是否将文件数据存储在/var/paas 路径下，修改避免使用该路径，并移除该路径下的异常文件后重试检查，否则禁止升级。

4.4.3.27 节点 Kubelet 检查

检查项内容

检查节点 kubelet 服务是否运行正常。

解决方案

- **问题场景一：kubelet 状态异常**

  kubelet 异常时，节点显示不可用，请修复节点后，重试检查任务。

- **问题场景二：cce-pause 版本异常**

  检测到当前 kubelet 依赖的 pause 容器镜像版本非 cce-pause:3.1，继续升级将会导致批量 Pod 重启，当前暂不支持升级，请联系技术支持人员。

4.4.3.28 节点内存检查

检查项内容

检查节点内存使用情况，是否超过 90%。

解决方案

- **请在业务低峰时进行集群升级。**
- 请检查该节点的 Pod 部署数量是否过多，适当驱逐该节点上 Pod 到其他空闲节点。

4.4.3.29 节点时钟同步服务器检查

检查项内容

检查节点时钟同步服务器 ntpd 或 chronyd 是否运行正常。

解决方案

- **问题场景一：ntpd 运行异常**

  请登陆该节点，执行 **systemctl status ntpd** 命令查询 ntpd 服务运行状态。若回显状态异常，请执行 **system restart ntpd** 命令后重新查询状态。

  以下为正常回显：

  ```
  [root@[redacted] paas]# systemctl status ntpd
  ● ntpd.service - Network Time Service
     Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor preset: disabled)
     Active: active (running) since Tue 2022-12-06 14:52:30 CST; 4 days ago
   Main PID: 8587 (ntpd)
      Tasks: 2
     Memory: 1.6M
     CGroup: /system.slice/ntpd.service
             └─8587 /usr/sbin/ntpd -u ntp:ntp -g -x
  ```

  若重启 ntpd 服务无法解决该问题，请联系技术支持人员。

- **问题场景二：chronyd 运行异常**

  请登陆该节点，执行 **systemctl status chronyd** 命令查询 chronyd 服务运行状态。若回显状态异常，请执行 **system restart chronyd** 命令后重新查询状态。

  以下为正常回显：

  ```
  root@[redacted] # systemctl status chronyd
  ● chrony.service - chrony, an NTP client/server
     Loaded: loaded (/lib/systemd/system/chrony.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-08-24 16:33:28 CST; 3 months 16 days ago
       Docs: man:chronyd(8)
             man:chronyc(1)
             man:chrony.conf(5)
    Process: 6492 ExecStartPost=/usr/lib/chrony/chrony-helper update-daemon (code=exited, status=0/SUCCESS)
    Process: 6461 ExecStart=/usr/lib/systemd/scripts/chronyd-starter.sh $DAEMON_OPTS (code=exited, status=0/SUCCESS)
   Main PID: 6488 (chronyd)
      Tasks: 1 (limit: 4915)
     CGroup: /system.slice/chrony.service
             └─6488 /usr/sbin/chronyd
  ```

  若重启 chronyd 服务无法解决该问题，请联系技术支持人员。

4.4.3.30 节点 OS 检查

检查项内容

检查节点操作系统内核版本为 CCE 支持的版本，未进行过内核升级。

解决方案

CCE 节点运行依赖创建时的初始标准内核版本，CCE 基于该内核版本做了全面的兼容性测试，非标准的内核版本可能在节点运行或升级中发生意外的兼容性问题，因此 CCE 将内核升级视为高危操作，详情请参见高危操作及解决方案。

当前 CCE 禁止该类节点进行升级，建议您在升级前重置节点至标准内核版本。

4.4.3.31 节点 CPU 数量检查

检查项内容

检查 Master 节点的 CPU 数量是否大于 2 核。

解决方案

Master 节点 CPU 数量为 2 核时，请联系技术支持人员，将该集群 Master 节点扩容至 4 核及以上。

4.4.3.32 节点 Python 命令检查

检查项内容

检查 Node 节点中 Python 命令是否可用。

检查方式

```
/usr/bin/python --version
echo $?
```

如果回显值不为 0 证明检查失败。

解决方案

安装 Python 之后再进行升级。

4.4.3.33 ASM 网格版本检查

检查项内容

当前检查项包括以下内容：

- 检查集群是否使用 ASM 网格服务
- 检查当前 ASM 版本是否支持目标集群版本

解决方案

- 先升级对应的 ASM 网格版本，再进行集群升级，ASM 网格版本与集群版本适配规则如下表。

表4-14 ASM 网格版本与集群版本适配规则

ASM 网格版本	集群版本
1.3	v1.13　v1.15　v1.17　v1.19
1.6	v1.15　v1.17
1.8	v1.15　v1.17　v1.19　v1.21
1.13	v1.21　v1.23

- 若不需要使用 ASM 网格，可删除 ASM 网格后再进行升级，升级后集群不能绑定与表中不匹配的 ASM 网格版本。例如，使用 v1.21 版本集群与 1.8 版本 ASM 网格，若要升级至 v1.23 版本集群时，请先升级 ASM 网格至 1.13 版本后再进行 v1.23 版本集群升级。

4.4.3.34 节点 Ready 检查

检查项内容

检查集群内节点是否 Ready。

解决方案

- **问题场景一：节点状态显示不可用**

  请登录 CCE 控制台，前往“节点管理”，筛选出状态不可用的节点后，请参照控制台提供的“修复建议”修复该节点后重试检查。

- **问题场景二：节点状态与实际不符**

  节点状态与实际不符可能存在两种情况：

  a. 控制台“节点管理”处显示正常，但检查结果仍然提示该节点 NotReady。请重试检查。

  b. 控制台“节点管理”处无该节点，但检查结果显示集群中仍然存在该节点。请联系技术人员支持。

4.4.3.35 节点 journald 检查

检查项内容

检查节点上的 journald 状态是否正常。

解决方案

请登陆该节点，执行 **systemctl is-active systemd-journald** 命令查询 journald 服务运行状态。若回显状态异常，请执行 **systemctl restart systemd-journald** 命令后重新查询状态。

以下为正常回显：

```
[root@[redacted] paas]# systemctl is-active systemd-journald
active
```

若重启 journald 服务无法解决该问题，请联系技术支持人员。

4.4.3.36 节点干扰 ContainerdSock 检查

检查项内容

检查节点上是否存在干扰的 Containerd.Sock 文件。该文件影响 euler 操作系统下的容器运行时启动。

解决方案

问题场景：节点使用的 docker 为定制的 Euler-dokcer 而非社区的 docker

步骤 1 登录相关节点。

步骤 2 执行 **rpm -qa | grep docker | grep euleros** 命令，如果结果不为空，说明节点上使用的 docker 为 Euler-docker

步骤 3 执行 **ls /run/containerd/containerd.sock** 命令，若发现存在该文件则会导致 docker 启动失败。

步骤 4 执行 **rm -rf /run/containerd/containerd.sock** 命令，然后重新进行集群升级检查。

----结束

4.4.3.37 内部错误

检查项内容

在升级前检查流程中是否出现内部错误。

解决方案

如遇该检查任务执行失败，请联系技术支持人员。

4.4.3.38 节点挂载点检查

检查项内容

检查节点上是否存在不可访问的挂载点。

解决方案

问题场景：节点上存在不可访问的挂载点

节点存在不可访问的挂载点，通常是由于该节点或节点上的 Pod 使用了网络存储 nfs（常见的 nfs 类型有 obsfs、sfs 等），且节点与远端 nfs 服务器断连，导致挂载点失效，所有访问该挂载点的进程均会 D 住卡死。

步骤 1 登录节点。

步骤 2 节点上依次执行如下命令：

```
- df -h
- for dir in `df -h | grep -v "Mounted on" | awk "{print \\$NF}"`;do cd $dir; done
&& echo "ok"
```

步骤 3 若返回 ok 则无问题。

否则，请另起一个终端执行如下命令，查询先前命令是否存在 D 状态：

```
- ps aux | grep "D "
```

步骤 4 若发现进程存在 D 状态，则确认为该问题，目前只可以通过重置节点解决。请选择一个合适的时间重置节点后，重试升级。

说明

重置节点会重置所有节点标签，可能影响工作负载调度，请在重置节点前检查并保留您手动为该节点打上的标签。

----结束

4.4.3.39 K8S 节点污点检查

检查项内容

检查节点上是否存在集群升级需要使用到的污点，如下表。

表4-15 检查污点列表

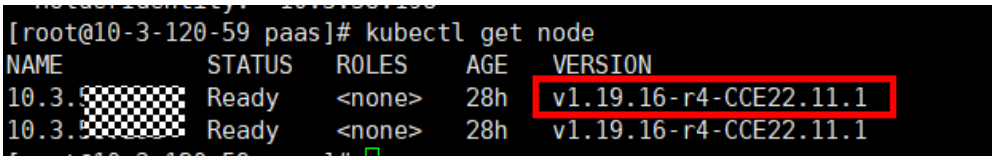
污点名称	污点影响
node.kubernetes.io/upgrade	NoSchedule

解决方案

问题场景一：该节点为集群升级过程中跳过的节点。

步骤 1 配置 Kubectl 命令，具体请参见通过 Kubectl 连接集群。

步骤 2 查看对应节点 kubelet 版本，以下为正常回显：



若该节点的 VERSION 与其他节点不同，则该节点为升级过程中跳过的节点，请在合适的时间重置节点后，重试检查。

说明

重置节点会重置所有节点标签，可能影响工作负载调度，请在重置节点前检查并保留您手动为该节点打上的标签。

----结束

4.4.3.40 everest 插件版本限制检查

检查项内容

检查集群当前 Everest 插件版本是否存在兼容性限制。

表4-16 受限的 Everest 插件版本

插件名称	涉及版本
Everest	v1.0.2-v1.0.7 v1.1.1-v1.1.5

解决方案

检测到当前 Everest 版本存在兼容性限制，无法随集群升级流程升级，请联系技术支持人员。

4.4.3.41 cce-hpa-controller 插件限制检查

检查项内容

检查到目标 cce-controller-hpa 插件版本是否存在兼容性限制。

解决方案

检测到目标 cce-controller-hap 插件版本存在兼容性限制，需要集群安装能提供 metric api 的插件，例如 metric-server。

4.4.3.42 动态绑核检查

检查项内容

检查当前集群版本和要升级的目标版本是否支持动态绑核。

解决方案

问题场景：当前集群版本使用动态绑核功能，要升级的目标集群版本不支持动态绑核功能。

升级到支持动态绑核的集群版本，支持动态绑核的集群版本如下表所示：

表4-17 支持动态绑核的集群版本列表

集群版本	是否支持动态绑核功能
v1.17 及以下版本	不支持
v1.19	支持
v1.21	不支持
v1.23 及以上版本	支持

4.4.4 升级后验证

4.4.4.1 业务验证

检查项内容

集群升级完毕，由用户验证当前集群正在运行的业务是否正常。

检查步骤

业务不同，验证的方式也有所不同，建议您在升级前确认适合您业务的验证方式，并在升级前后均执行一遍。

常见的业务确认方式有：

- 业务界面可用
- 监控平台无异常告警与事件
- 关键应用进程无错误日志
- API 拨测正常等

解决方案

若集群升级后您的在线业务有异常，请联系技术支持人员。

4.4.4.2 存量 Pod 检查

检查项内容

- 检查集群中是否的存在状态非预期的 Pod
- 检查集群中的 Pod 是否异常重启

检查步骤

请您登陆 CCE 控制台，在“资源->工作负载->容器组”处选择全部命名空间，点击“状态”，过滤并观察是否存在异常状态的 Pod。

查看“重启次数”栏目，观察是否存在异常重启的 Pod。

解决方案

若集群升级后您的集群有异常 Pod，请联系技术支持人员。

4.4.4.3 存量节点与容器网络检查

检查项内容

- 检查存量节点是否运行正常
- 检查存量节点的网络是否运行正常
- 检查存量容器的网络是否运行正常

检查步骤

节点组件异常或节点网络异常，均会反映在节点状态上。

请登录 CCE 控制台，前往“资源->节点管理”处查看节点状态，检查是否有处于异常状态的节点，可通过状态栏过滤。

容器网络异常会反映在业务上，请检查您的业务是否运行正常。

解决方案

若节点状态异常，请联系技术支持人员。

若容器网络异常，并影响了您的业务，请联系技术支持人员，并同步确认当前异常的网络访问路径。

源端	目的端	目的端类型	可能故障
• 集群内 Pod • 集群内节点 • 集群外同 VPC 下节点 • 天翼云外	Service ELB 公网 IP	集群流量负载均衡入口	未有记录
	Service ELB 私网 IP	集群流量负载均衡入口	未有记录
	Ingress ELB 公网 IP	集群流量负载均衡入口	未有记录
	Ingress ELB 私网 IP	集群流量负载均衡入口	未有记录

源端	目的端	目的端类型	可能故障
	Service Node Port 公网 IP	集群流量入口	kube proxy 配置覆盖，该故障已在升级流程适配
	Service Node Port 私网 IP	集群流量入口	未有记录
	Service Cluster IP	Service 网络平面	未有记录
	非 Service NodePort 节点单口	节点容器网络	未有记录
	跨节点 Pod	容器网络平面	未有记录
	同节点 Pod	容器网络平面	未有记录
	Service 域名、Pod 域名等，基于 CoreDNS 解析	域名解析	未有记录
	外部网站域名，基于 CoreDNS hosts 配置解析	域名解析	coredns 插件升级后配置被覆盖，该故障已在插件升级流程适配
	外部网站域名，基于 CoreDNS 上游服务器解析	域名解析	coredns 插件升级后配置被覆盖，该故障已在插件升级流程适配
	外部网站域名，不通过 CoreDNS 解析	域名解析	未有记录

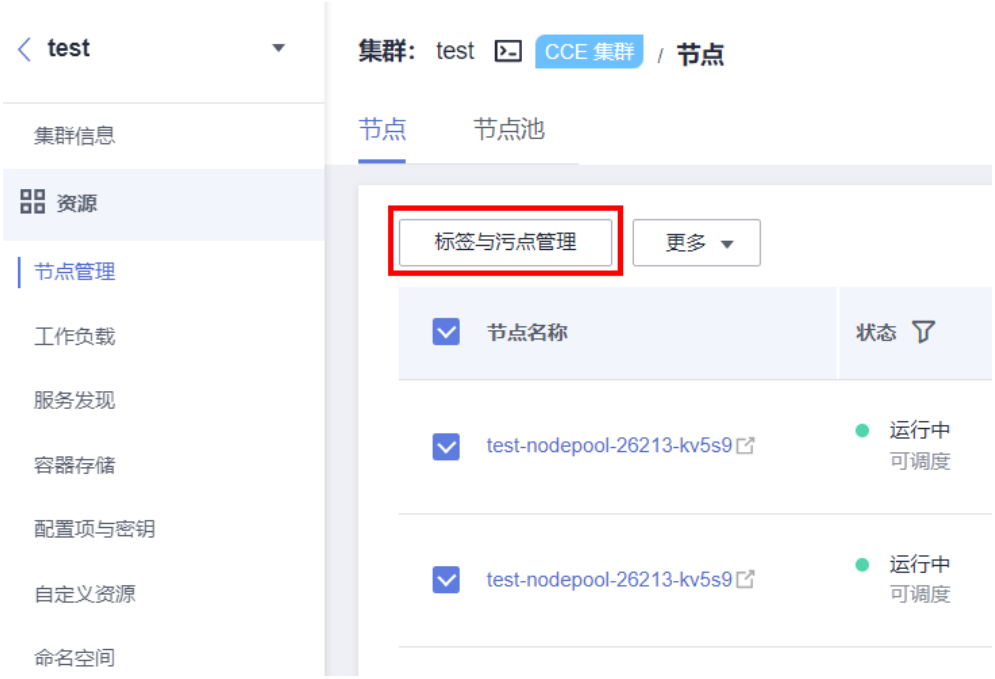
4.4.4.4 存量节点标签与污点检查

检查项内容

- 检查标签是否丢失
- 检查是否有异常新增的污点

检查步骤

请登录 CCE 控制台，前往“资源->节点管理->节点”，勾选所有节点后，单击“标签与污点管理”，查看目前节点的标签与污点。



解决方案

集群升级过程中不改变用户的标签，若您发现标签丢失或异常新增，请联系技术支持人员。

若您发现节点新增污点（node.kubernetes.io/upgrade），该节点可能为升级过程中跳过的节点，请参照重置跳过节点检查处理。

若您发现节点新增其他污点，请联系技术人员支持。

4.4.4.5 新建节点检查

检查内容

检查集群是否可以正常创建节点。

检查步骤

请登录 CCE 控制台，前往“资源->节点管理”，单击“创建节点”。

解决方案

若集群升级后您的集群无法创建节点，请联系技术支持人员。

4.4.4.6 新建 Pod 检查

检查内容

- 检查集群升级后，存量节点是否能新建 Pod。
- 检查集群升级后，新建节点是否能新建 Pod。

检查步骤

基于新建节点检查创建了新节点后，通过创建 daemonset 类型工作负载，在每个节点上创建 Pod。

请登录 CCE 控制台，前往“资源->工作负载->守护进程集”，单击右上角“创建负载”或“YAML 创建”。

建议您使用日常测试的镜像作为基础镜像。您可参照如下 yaml 部署最小应用 Pod。

说明

该测试 YAML 将 DaemonSet 部署在 default 命名空间下，使用 ngxin:perl 为基础镜像，申请 10m CPU，10Mi 内存，限制 100m CPU 50Mi 内存。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: post-upgrade-check
  namespace: default
spec:
  selector:
    matchLabels:
      app: post-upgrade-check
      version: v1
  template:
    metadata:
      labels:
        app: post-upgrade-check
        version: v1
    spec:
      containers:
        - name: container-1
          image: nginx:perl
          imagePullPolicy: IfNotPresent
          resources:
            requests:
              cpu: 10m
              memory: 10Mi
            limits:
```

```
            cpu: 100m
            memory: 50Mi
```

负载创建完毕后请检查该工作负载的 Pod 状态是否正常。

检查完毕后请登录 CCE 控制台，前往“资源->工作负载->守护进程集”，选择“post-upgrade-check”工作负载并单击“更多->删除”删除该测试用工作负载。

解决方案

若 Pod 无法新建，或状态异常，请联系技术支持人员，并说明异常发生的范围为新建节点还是存量节点。

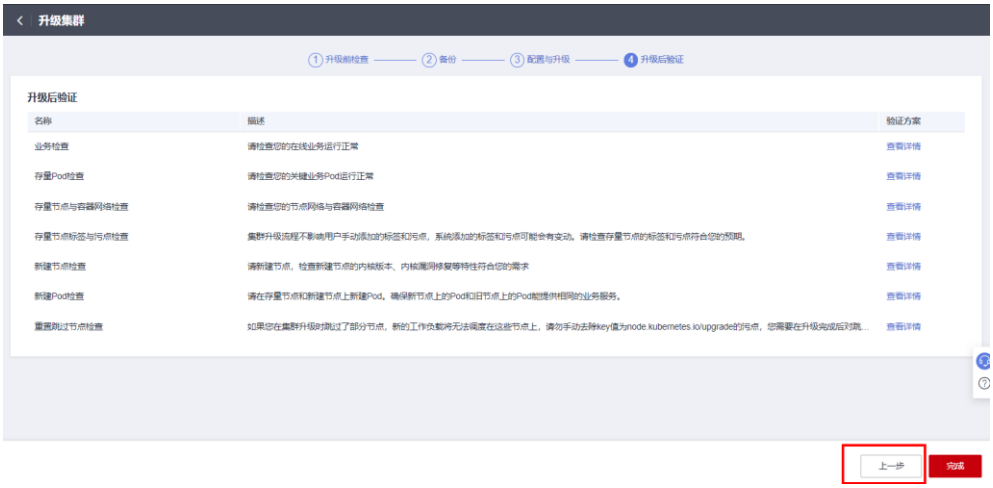
4.4.4.7 重置跳过节点检查

检查项内容

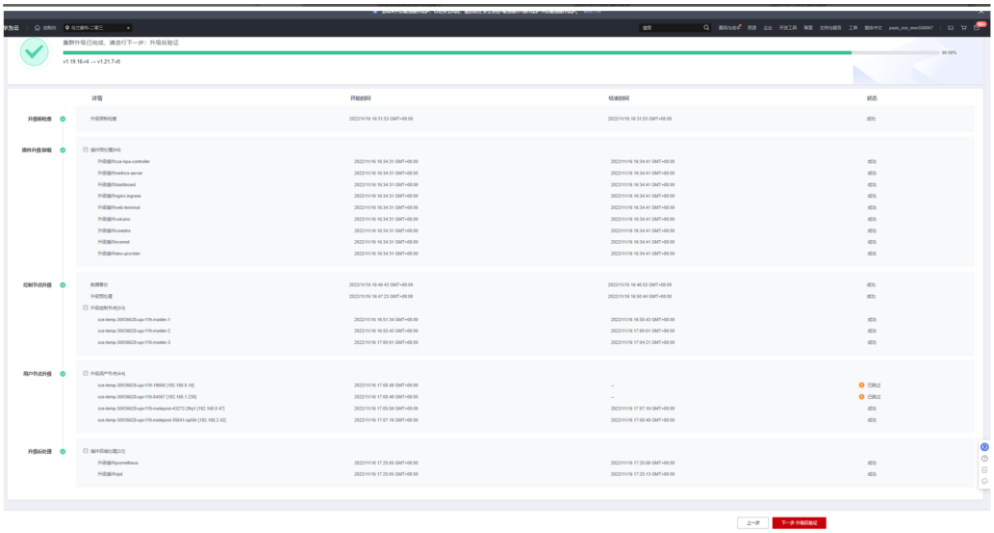
集群升级完毕之后，需要重置升级失败的节点。

检查步骤

返回上一步或通过升级历史页面查看进入升级详情页面，查看升级时跳过的节点。



对于升级详情页面中跳过的节点，请在升级完毕后重置节点。



说明

重置节点会重置所有节点标签，可能影响工作负载调度，请在重置节点前检查并保留您手动为该节点打上的标签。

4.4.5 重置升级/滚动升级（1.13 版本）

操作场景

您可以通过云容器引擎管理控制台，快速升级 Kubernetes 版本，以支持新特性的使用。

升级前，请先了解 CCE 各集群版本能够升级到的目标版本，以及升级方式和升级影响，详情请参见集群升级概述和升级前须知。

注意事项

- 集群升级时若需要升级 CoreDNS 插件，请确保节点数大于等于 CoreDNS 的实例数，且 CoreDNS 的所有实例都处于运行状态，否则将导致升级失败。1.13 版本的集群若需升级到更高版本，需先将 CoreDNS 插件升级当前集群可用的最新版本。
- 1.13 版本内升级时，集群上的应用只会在升级网络组件时有短暂中断。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

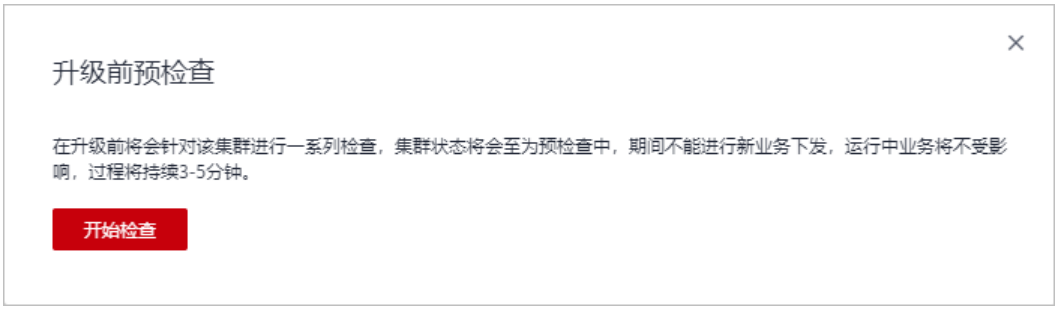
步骤 2 在左侧导航栏选择“集群升级”，在右侧可以看到可升级的版本，单击“版本升级”。

说明

- 若您的集群当前已是最新版本，则“版本升级”按钮为灰色不可用状态。
- 若您的集群状态异常或存在异常状态的插件，则“版本升级”按钮为灰色不可用状态，请参考升级前须知检查。

步骤 3 在弹出的“升级前预检查”对话框中，单击“开始检查”。

图4-8 升级前预检查



步骤 4 升级前预检查开始启动，在此过程中集群状态将显示为“预检查中”，期间该集群不能进行新业务下发，运行中的业务不受影响，该过程将持续 3-5 分钟。

图4-9 升级前预检查-进行中

步骤 5 待升级前预检查的状态显示为“已完成”时，单击“去升级”。

图4-10 升级前预检查-完成

步骤 6 进入集群升级页面，参照下表确认或配置基本信息。

表4-18 基本信息确认与配置

参数	参数说明
集群名称	请确认您要升级的集群名称。
当前版本	请确认待升级集群的版本。
升级后版本	请确认升级后的目标版本。
节点升级策略	**重置升级**：用户节点采用重置安装方式，节点操作系统将会被重装，系统盘和数据盘的数据均会被清空，请谨慎使用。 说明 • 本集群的节点及工作负载生命周期管理功能暂不可用。 • API 访问功能暂不可用。 • 由于升级过程节点进行重置安装，用户已运行的工作负载业务将会中断。 • 用户节点的系统盘和数据盘将会被清空，升级前请事先备份重要数据。 • 用户节点上挂载的非 LVM 管理的数据盘，升级后需要重新挂载，盘中数据不会丢失。 • 云硬盘的配额需大于 0。

参数	参数说明
	• 容器的 IP 地址会发生变化，但是不影响容器间的网络通信。 • 用户节点的自定义标签将不会保留。 • 集群升级时间约为 12 分钟。 **滚动升级**：用户节点采用节点池滚动升级，适用于集群下节点均采用节点池创建的场景。 **说明** • 本集群的节点及工作负载生命周期管理功能暂不可用。 • API 访问功能暂不可用。 • 用户已运行的工作负载业务不会中断。 • 集群升级时间约为 12 分钟。
重置节点镜像	仅支持物理机节点。 物理机节点支持在升级时替换操作系统镜像，可指定节点使用新的镜像，在升级时会使用新镜像重装操作系统。如不指定则默认使用原有镜像重装操作系统。
登录方式	**密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。
集群备份	对集群的 Master 节点进行整机备份，**需要用户手动确认**，备份过程会使用云备份服务，备份通常耗时在 20 分钟左右，若当前局点云备份任务排队较多时，备份时间可能同步延长，推荐用户使用进行整机备份。
节点升级优先级	可选择优先升级的节点。

步骤 7 完成后单击“下一步”，在弹出的“集群升级”对话框中单击“确定”。

根据您选择的“节点升级策略”，对话框中会有如下两种不同的提示：

- 重置升级：升级后的集群版本将使用更高版本操作系统，升级将会重启节点并升级操作系统版本，升级过程中运行的业务会暂时中断。
- 滚动升级：滚动升级后，需要用户重置节点（同时去除不可调度标签）或新建节点才能完成升级。

步骤 8 进入“升级插件”步骤，如有需要升级的插件会有红色圆点提示，请单击插件卡片左下角的“升级”按钮，完成后单击页面右下角的“升级”。

说明

- 集群将依次升级控制节点，然后并发升级用户节点。用户节点较多时，节点将分批升级。
- 请选择合适的时间段进行升级，以减少升级对业务的影响。
- 单击“升级”会立刻开始执行升级操作，并且无法撤销。升级过程中请勿对节点执行关机、重启等操作。

步骤 9 在弹出的“集群升级”对话框中阅读提示信息，确认后单击“确定”，注意集群升级后不可回退。

图4-11 集群升级确认

步骤 10 在集群列表页面中可以看到集群的状态为“升级中”，升级过程需要一定的时间，请耐心等待升级完成。

升级成功后，您可以在集群列表或集群详情页面查看升级后的集群状态和版本。

----结束

4.4.6 原地升级

操作场景

您可以通过云容器引擎管理控制台升级集群版本，以支持新特性的使用。

升级前，请先了解 CCE 各集群版本能够升级到的目标版本，以及升级方式和升级影响，详情请参见集群升级概述和升级前须知。

升级说明

- 集群的升级采用原地升级方式更新节点上的 Kubernetes 组件，升级后不会改变节点上的 OS 版本。
- 数据面节点升级时将采用分批升级的方式，默认会选择根据 CPU、内存、PDB（Pod Disruption Budget，即为应用程序设置干扰预算）等设置节点升级的优先级，您也可以根据您的业务需要自行设置优先级。

注意事项

- 集群升级过程中会自动升级插件到目标集群兼容的版本，升级过程中请不要卸载或者重装插件。
- 升级之前请确认所有的插件都处于运行状态，如果插件升级失败可以在插件问题修复后，重试升级。
- 升级时会检查插件运行状态，部分插件（如 CoreDNS）需要至少两个节点才能维持正常状态，那此时升级就至少需要两个节点。
- 若在集群升级过程中出现升级失败的提示，请参照提示信息修复问题后点击重试，若重试后仍未成功升级，请提交工单联系我们协助您进行修复。

更多注意事项请参见升级前须知。

操作步骤

集群升级步骤包括：升级前检查、备份、配置与升级、升级后处理。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏选择“集群升级”，在右侧可以看到推荐升级的版本。

在“集群升级”页面中，可以看到当前集群的版本信息（当前版本号、上次更新/升级时间）、可升级版本、升级须知以及升级历史。

步骤 3 选择可升级的集群版本，并单击“升级前检查”。

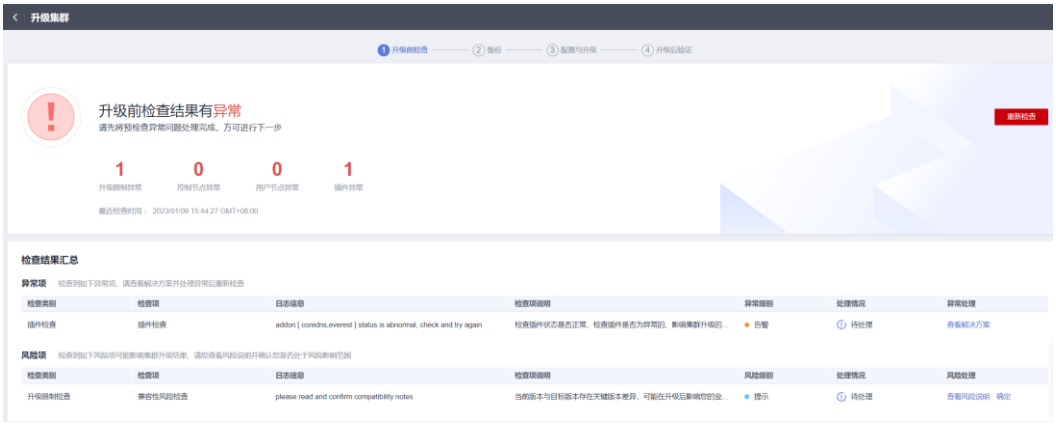
说明

- 若您的集群存在更新的小版本，此处无需选择集群版本，默认为最新的小版本。
- 若您的集群存在更新的大版本，您可根据需要选择升级的目标版本。
- 若您的集群当前已是最新版本，升级前检查入口将会隐藏。

步骤 4 单击“开始检查”并确认。如集群中存在异常项或风险项，请根据页面提示的检查结果进行处理，处理完成后需重新进行升级前检查。

- 异常项：请查看页面提示的解决方案并处理异常后，重新进行升级前检查。
- 风险项：表示该结果可能会影响集群升级结果，请您查看风险说明并确认您是否处于风险影响范围。如确认无风险，可单击该风险项后的“确认”按钮，手动跳过该风险项，然后重新进行升级前检查。

待升级前检查通过后，单击“下一步 备份”。



步骤 5 （可选）手动进行集群备份。集群升级时会提供默认的数据备份，如需手动备份可单击“备份”按钮执行备份，如无需手动备份可直接单击“下一步 配置与升级”。

手动单击“备份”按钮会对集群的 Master 节点进行整机备份，备份过程会使用云备份服务，备份通常耗时在 20 分钟左右，若当前局点云备份任务排队较多时，备份时间可能同步延长。备份过程中集群将不允许升级。

步骤 6 配置升级参数。

- **插件升级配置：**此处列出了您的集群中已安装的插件。在集群升级过程中系统会自动升级插件，以兼容升级后的集群版本，您可以单击插件右侧的“配置”重新定义插件参数。

  说明

  插件右侧如有红色标记，表明该插件将不能兼容升级后的集群版本，升级过程中会卸载并重装该插件，请您务必确认插件的配置参数。

- **节点升级配置：**您可以设置每批升级的最大节点数量。
- **节点优先级配置：**您可以自行定义节点升级的优先级顺序，若不选择，默认情况下系统会根据您节点的情况优选后分批升级。优先级设置时需要先选择节点池，再设置节点池中节点的升级批次，并按照您设置的节点池以及节点顺序进行升级。
  - 添加优先级：添加节点池的优先级，自行定义节点池升级的优先级顺序。
  - 添加节点优先级：添加节点池的优先级后，可以设置该节点池内节点升级的优先级顺序，升级时系统将按照您设置的顺序依次对节点进行升级，如不设置该优先级，系统将按照默认的策略执行。

步骤 7 配置完成后，单击“升级”按钮，并确认升级操作后集群开始升级。您可以在页面下方查看版本升级的进程。

升级过程中，您可以单击右侧的“暂停”按钮，暂停集群版本的升级，若想继续升级，可单击“继续”。当版本更新进度条显示 100%时，表示集群已完成升级。

说明

若在集群升级过程中出现升级失败的提示，请参照提示信息修复问题后重试。

步骤 8 升级完成后，单击“下一步 升级后验证”，请根据页面提示的检查项进行升级后验证。确认所有检查项均正常后，可单击“完成”按钮，并确认完成升级后检查。

您可以在集群列表页面查看集群当前的 Kubernetes 版本，确认升级成功。

----结束

4.4.7 集群跨版本业务迁移

适用场景

本章介绍在 CCE 中如何将老版本集群的业务迁移到新版本集群。

适用于需要大幅度跨版本集群升级（如 1.7.*或 1.9.* 升级到 1.17.*版本）的需求，可以接受新建新版本集群而进行业务迁移的升级方式。

前提条件

表4-19 迁移前 Checklist

类别	描述
集群相关	Nodeip 强相关：确认之前集群的节点 IP（包括 EIP），是否有作为其他的配置或者白名单之类的设置。
工作负载	记录工作负载数目，便于迁移后检查。
存储	1. 确认应用中存储，是否使用云，或者自己搭建存储。 2. 自动创建的存储需要在新集群中变成使用已有存储。
网络	1. 注意使用的负载均衡服务，以及 Ingress。 2. 老版本的集群只支持经典型负载均衡服务，迁移到新集群中需要改成共享型负载均衡服务，对应负载均衡服务将会重新建立。
运维	私有配置：确认在之前集群中，是否在节点上配置内核参数或者系统配置。

操作步骤

步骤 1 创建新集群

创建与老版本集群同规格同配置的集群，创建方法请参见购买 CCE 集群。

步骤 2 添加节点

添加同规格节点，并且在节点上配置之前的手动配置项，创建方法请参见创建节点。

步骤 3 创建存储

在新集群中使用已有存储创建 PVC，PVC 名称不变，方法请参见存储卷声明 PVC。

说明

切流方案仅支持 OBS、SFS、共享云硬盘类型的存储。非共享云硬盘存储切流需要将老集群内的工作负载暂停，会导致断服。

步骤 4 创建工作负载

在新集群中创建工作负载，名称和规格参数保持不变，创建方法请参见创建无状态负载(Deployment)或创建有状态负载(StatefulSet)。

步骤 5 创建服务

在新集群中创建 Service，名称和规格参数保持不变，创建方法请参见 Service。

步骤 6 调测功能

全部创建完成后，请自行调测业务，调测无问题后切换流量。

步骤 7 老集群退订或删除

新集群全部功能 ready，退订或者删除老集群，删除集群方法请参见删除集群（按需计费）。

----结束

4.5 管理集群

4.5.1 删除集群（按需计费）

操作场景

本章节以计费模式为“按需计费”的 CCE 集群为例，介绍如何删除集群。

包周期的集群支持退订操作，详情请参见退订/释放集群（包年/包月）。

注意事项

- 删除集群时，纳管以及包周期节点将会从集群中移除并重装系统，节点原有的登录密码将会失效，请重新设置节点密码。
- 删除集群不会删除集群下包周期的资源，相关资源在集群删除后将会继续计费，请妥善处理。
- 删除集群会删除集群下的节点（纳管节点不会被删除）、节点挂载的数据盘、工作负载与服务，相关业务将无法恢复。在执行操作前，请确保相关数据已完成备份或者迁移，删除完成后数据无法找回，请谨慎操作。

  部分不是在 CCE 中创建的资源不会删除：
  - 纳管的节点（仅删除在 CCE 中创建的节点）
  - Service 和 Ingress 关联的 ELB 实例（仅删除自动创建的 ELB 实例）
  - 手动创建 PV 关联的云存储/导入的云存储（仅删除 PVC 自动创建的云存储）
- 处于休眠状态的集群无法直接删除，请将集群唤醒后重试。
- 集群不可用时删除集群，存储会残留。
- 集群版本为 v1.13.10 及之前版本时，请勿在 ELB 服务手动修改监听器名称和后端服务器名称，否则删除集群会有资源残留。

操作步骤

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击待删除集群后的 。

图4-12 删除集群

步骤 3 在弹出的“删除集群”窗口中，根据系统提示，勾选删除集群时需要释放的资源。

- 删除集群下工作负载挂载的云存储

  说明

  删除集群中的存储卷声明和存储卷，存在如下约束：

  - 底层存储依据指定的回收策略进行删除。
  - 对象存储桶下存在大量文件（超过 1000）时，请先手动清理桶内文件后再执行集群删除操作。

- 删除集群下负载均衡 ELB 等网络资源（仅删除自动创建的 ELB 资源）

步骤 4 单击“是”，开始执行删除集群操作。

删除集群需要花费 1~3 分钟，请耐心等候。

----结束

4.5.2 退订/释放集群（包年/包月）

客户购买包周期集群后，支持退订/释放包周期资源。

按需计费的集群可以直接删除，详情请参见删除集群（按需计费）。

注意事项

- 退订/释放集群会删除集群下的节点（纳管节点不会被删除）、节点挂载的数据盘、工作负载与服务，相关业务将无法恢复。在执行操作前，请确保数据已完成备份或者迁移，退订/释放完成后数据无法找回，请谨慎操作。

  部分不是在 CCE 中创建的资源不会删除：

  - 纳管的节点（仅删除在 CCE 中创建的节点）
  - Service 和 Ingress 关联的 ELB 实例（仅删除自动创建的 ELB 实例）

  - 手动创建 PV 关联的云存储/导入的云存储（仅删除 PVC 自动创建的云存储）
- 退订/释放集群仅退订订单关联的资源，非关联资源不会处理，相关资源会继续计费，请妥善处理。
- 对于包周期集群，如果过保留期会自动释放，集群下的节点如果同时到期会一并释放，节点如果未到期 CCE 不会对其做任何操作，相关数据会继续保留，相关资源会继续计费。请关注您账号下到期未续费集群，及时续费，防止节点被重装导致数据丢失。
- 集群资源包含控制节点资源和工作节点所使用的的基础设施，详情请参见计费说明。
- 资源退订，相关注意事项：
  - 如果您正在退订使用中的资源，请仔细确认资源信息和退款信息，资源退订后将无法恢复。若您要保留资源，仅退订未使用的续费周期。
  - 如果您退订的资源与其他包年/包月资源关联，退订操作可能会影响关联资源的正常使用，请谨慎操作。

    如果已与其他按需资源关联，退订操作完成后，关联的按需资源可以正常使用并计费。

集群退订

本节以计费模式为“包年/包月”且未超期的 CCE 集群为例，介绍如何退订集群。

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击待退订集群后的。

图4-13 集群退订

步骤 3 在弹出的“退订”页面中，勾选要释放的资源。

- 删除集群下工作负载挂载的云存储

  说明

  删除集群中的存储卷声明和存储卷，存在如下约束：

  - 底层存储依据指定的回收策略进行删除。
  - 对象存储桶下存在大量文件（超过 1000）时，请先手动清理桶内文件后再执行集群删除操作。
- 删除集群下负载均衡 ELB 等网络资源（仅删除自动创建的 ELB 资源）

步骤 4 单击“是”，开始退订集群。退订集群需要花费 1~3 分钟，请耐心等候。

----结束

4.5.3 续费集群（包年/包月）

客户购买包周期集群后，支持续费包周期资源。

操作步骤

本节以计费模式为“包年/包月”的 CCE 集群为例，介绍如何为购买的集群续费。

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击待续费集群后的 。

图4-14 续费集群

步骤 3 在弹出的“续费”页面中，根据系统提示进行续费操作。

说明

- 您已选择操作的资源（高亮显示）和其他资源有关联关系，请确认是否同时操作。

步骤 4 单击“去支付”，在打开的支付页面中核对订单金额，并选择支付方式后单击“确认付款”。

步骤 5 完成付款后，可单击“返回我的订单”或“返回续费管理”页面查看和管理订单信息。

----结束

4.5.4 变更集群规格

操作场景

当前集群管理规模可支持管理的用户节点个数不能满足用户诉求，可通过“变更集群规格”功能来扩大使用的用户节点个数。

约束限制

- 集群 v1.15 及以上版本支持变更集群规格。

- 变更集群规格目前只支持扩容到更大规格，不支持降低集群规格。
- 规格变更期间，控制节点存在开关机动作，集群将无法正常使用，请尽量在业务平稳期执行变更操作。
- 集群规格变更不会影响集群中已运行业务，但变更过程中管理面（Master 节点）会有短暂中断，建议变更期间停止其他操作（如创建工作负载等）。
- 规格变更失败后集群将尽可能回退到正常状态，若回退异常可提交工单进行处理。

操作步骤

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击需要变更规格集群后的  。

图4-15 变更规格

步骤 3 在弹出的页面，在“可选规格”后根据实际需求选择新的集群规格。

步骤 4 单击“确定”。

您可以单击在左上角单击“操作记录”查看集群变更记录。状态从“执行中”变为“成功”，表示集群规格变更成功。

图4-16 操作记录

----结束

4.5.5 休眠与唤醒集群（按需计费）

操作场景

暂时不需要使用集群时，建议您将集群设置为休眠状态。

集群休眠后，将无法在此集群上创建和管理工作负载等资源。

休眠中的集群可以快速唤醒，正常使用。

约束与限制

包年包月的集群不支持休眠。

集群唤醒过程中，可能会由于资源不足导致 Master 节点启动失败，从而导致集群唤醒失败，请过一段时间再次唤醒。

集群休眠

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击待休眠集群后的。

图4-17 休眠集群

步骤 3 在弹出的集群休眠提示框中，查看风险提示，单击“是”，等待集群完成休眠。

集群休眠后，将暂停收取控制节点资源费用。

集群休眠后，集群所属的工作节点（ECS）、绑定的弹性 IP、带宽等资源仍将按各自的计费方式进行收费。如需关机节点，请在集群休眠提示框中勾选“关机集群下所有节点”或参见节点关机。

图4-18 集群休眠提示

----结束

集群唤醒

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击待唤醒集群后的。

图4-19 集群唤醒提示

步骤 3 当集群状态由“唤醒中”变为“运行中”时，即完成唤醒操作，唤醒集群预计需要 3-5 分钟。

说明

集群唤醒后，将继续收取控制节点资源费用。

----结束

4.5.6 更改集群节点的默认安全组

操作场景

集群在创建时可指定自定义节点安全组，方便统一管理节点的网络安全策略。对于已创建的集群，支持修改集群默认的节点安全组。

约束与限制

- 一个安全组关联的实例数量建议不超过 1000 个，否则可能引起安全组性能下降。
- 不支持指定 Master 节点的安全组，同时请谨慎修改集群 Master 节点的安全组规则。

操作步骤

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击集群名称，查看“集群信息”页面。

步骤 3 在“网络信息”中单击“节点默认安全组”后的按钮。

步骤 4 选择一个已有的安全组，并确认安全组规则满足集群要求后，单击“确定”。

须知

- 请确认选择的安全组设置了正确的端口规则，否则将无法成功创建节点。安全组需要满足的端口规则根据集群类别存在差异，详情请参见帮助中心 > 常见问题 > 网络规划。
- 新安全组只对新创建或纳管的节点生效，存量节点需要手动修改节点安全组规则，即使对存量节点进行重置，也仍会使用原安全组。

----结束

4.5.7 配置管理

操作场景

CCE 支持对集群配置参数进行管理，通过该功能您可以对核心组件进行深度配置。

约束与限制

本功能仅支持在 **v1.15 及以上版本**的集群中使用，V1.15 以下版本不显示该功能。

操作步骤

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击集群后的 。

图4-20 配置管理

步骤 3 在侧边栏滑出的“配置管理”窗口中，根据业务需求修改 Kubernetes 的参数值：

表4-20 扩展控制器组件参数

参数	详情	取值
enable-resource-quota	创建 namespace 时是否自动创建 resourcequota 对象。 • false：不自动创建 resourcequota 对象。 • true：自动创建 resourcequota 对象。resourcequota 的默认取值请参见设置资源配额及限制。	默认：false

表4-21 kube-apiserver 组件参数

参数	详情	取值
default-not-ready-toleration-seconds	notReady 容忍时间，NoExecute that is added by default to every pod that does not already have such a toleration.	默认：300s
default-unreachable-toleration-seconds	unreachable 容忍时间，NoExecute that is added by default to every pod	默认：300s

参数	详情	取值
	that does not already have such a toleration.	
max-mutating-requests-inflight	最大 mutating 并发请求数。 当服务器超过此值时，它会拒绝请求。 0 表示无限制。 从 1.21 版本开始不再支持手动配置，根据集群规格自动配置如下： • 50 和 200 节点：200 • 1000 节点：500 • 2000 节点：1000	默认：1000
max-requests-inflight	最大 non-mutating 并发请求数。 当服务器超过此值时，它会拒绝请求。 0 表示无限制。 从 1.21 版本开始不再支持手动配置，根据集群规格自动配置如下： • 50 和 200 节点：400 • 1000 节点：1000 • 2000 节点：2000	默认：2000
service-node-port-range	nodeport 端口范围	默认：30000-32767 取值范围： min>20105 max<32768

表4-22 kube-controller-manager 组件参数

参数	详情	取值
concurrent-deployment-syncs	deployment 的并发处理数	默认：5
concurrent-endpoint-syncs	endpoint 的并发处理数	默认：5
concurrent-gc-syncs	garbage collector 的并发数	默认：20
concurrent-job-syncs	允许同时同步的作业对象的数量。	默认：5
concurrent-namespace-syncs	namespace 的并发处理数	默认：10
concurrent-replicaset-syncs	replicaset 的并发处理数	默认：5
concurrent-resource-quota-	resource quota 的并发处理数	默认：5

参数	详情	取值
syncs		
concurrent-service-syncs	service 的并发处理数	默认：10
concurrent-serviceaccount-token-syncs	serviceaccount-token 的并发处理数	默认：5
concurrent-ttl-after-finished-syncs	ttl-after-finished 的并发处理数	默认：5
concurrent_rc_syncs	rc 的并发处理数	默认：5
horizontal-pod-autoscaler-sync-period	集群弹性计算的周期	默认：15s
kube-api-qps	与 kube-apiserver 通信的 qps	默认：100
kube-api-burst	与 kube-apiserver 通信的 burst	默认：100
terminated-pod-gc-threshold	在 Pod GC 开始删除终止 Pod 之前，可以存在的 terminated 状态 Pod 数量。 如果<=0，则禁用终止的 Pod GC。	默认：1000

表4-23 kube-scheduler 组件参数

参数	详情	取值
kube-api-qps	与 kube-apiserver 通信的 qps	默认：100
kube-api-burst	与 kube-apiserver 通信的 burst	默认：100

步骤 4 单击“确定”，完成配置操作。

----结束

参考链接

- kube-apiserver
- kube-controller-manager
- kube-scheduler

5 节点管理

5.1 节点概述

5.1.1 节点须知

简介

节点是容器集群组成的基本元素。节点取决于业务，既可以是虚拟机，也可以是物理机。每个节点都包含运行 Pod 所需要的基本组件，包括 Kubelet、Kube-proxy 、Container Runtime 等。

说明

CCE 创建的 Kubernetes 集群包含 master 节点和 node 节点，本章讲述的节点特指 **node 节点**，node 节点是集群的计算节点，即运行容器化应用的节点。

在云容器引擎 CCE 中，主要采用高性能的弹性云主机 ECS 或物理机 BMS 作为节点来构建高可用的 Kubernetes 集群。

支持的节点规格

不同区域支持的节点规格（flavor）不同，且节点规格存在新增、售罄下线等情况，建议您在使用前登录 CCE 控制台，在创建节点界面查看您需要的节点规格是否支持。

Docker 容器底层文件存储系统说明

- 1.15.6 及之前集群版本 docker 底层文件存储系统采用 xfs 格式。
- 1.15.11 及之后版本集群新建节点或重置后 docker 底层文件存储系统全部采用 ext4 格式。

对于之前使用 xfs 格式容器应用，需要注意底层文件存储格式变动影响（不同文件系统格式文件排序存在差异：如部分 java 应用引用某个 jar 包，但目录中存在多个版本该 jar 包，在不指定版本时实际引用包由系统文件排序决定）。

查看当前节点使用的 docker 底层存储文件格式可采用 docker info | grep "Backing Filesystem"确认。

节点 paas 用户/用户组说明

在 CCE 集群中创建节点时，默认会在节点上创建 paas 用户/用户组。节点上的 CCE 组件和 CCE 插件在非必要时会以非 root 用户（paas 用户/用户组）运行，以实现运行权限最小化，如果修改 paas 用户/用户组可能会影响节点上 CCE 组件和业务 Pod 正常运行。

须知

CCE 组件正常运行依赖 paas 用户/用户组，您需要注意以下几点要求：

- 请勿自行修改节点内目录权限、容器目录权限等。
- 请勿自行修改 paas 用户/用户组的 GID 和 UID。
- 请勿在业务中直接使用 paas 用户/用户组设置业务文件的所属用户和组。

节点生命周期

生命周期是指节点从创建到删除（或释放）历经的各种状态。

表5-1 节点生命周期状态说明

状态	状态属性	说明
运行中	稳定状态	节点正常运行状态，并连接上集群。 在这个状态的节点可以运行您的业务。
不可用	稳定状态	节点运行异常状态。 在这个状态下的实例，不能对外提供业务，需要重置节点。
创建中	中间状态	创建节点实例后，在节点状态进入运行中之前的状态。
安装中	中间状态	节点处于安装 Kubernetes 软件的过程中。
删除中	中间状态	节点处于正在被删除的状态。 如果长时间处于该状态，则说明出现异常。
关机	稳定状态	节点被正常停止。 在这个状态下的实例，不能对外提供业务，您可以在弹性云主机列表页对其进行开机操作。
错误	稳定状态	节点处于异常状态。 在这个状态下的实例，不能对外提供业务，需要重置节点。

5.1.2 容器引擎

容器引擎介绍

容器引擎是 Kubernetes 最重要的组件之一，负责管理镜像和容器的生命周期。Kubelet 通过 Container Runtime Interface (CRI) 与容器引擎交互，以管理镜像和容器。

CCE 当前支持用户选择 Containerd 和 Docker 容器引擎，**其中 Containerd 调用链更短，组件更少，更稳定，占用节点资源更少**。

Kubernetes 在 1.24 版本中移除了 Dockershim，并从此不再默认支持 Docker 容器引擎，详情请参见 Kubernetes 即将移除 Dockershim。为保障用户体验，CCE 1.25 版本中仍将继续维持 Docker 容器引擎，但在 1.27 版本中将不再支持 Docker 容器引擎，您需要将 Docker 节点迁移至 Containerd 节点。

节点操作系统与容器引擎对应关系

表5-2 CCE 集群节点操作系统与容器引擎对应关系

操作系统	内核版本	容器引擎	容器存储 Rootfs	容器运行时
CentOS 7.x	3.x	Docker 1.23 起支持 Containerd	1.19.16 以下版本集群使用 Device Mapper 1.19.16 及以上版本集群使用 OverlayFS	runC
EulerOS 2.9	4.x	Docker 1.23 起支持 Containerd	OverlayFS	runC
Ubuntu 18.04	4.x	Docker 1.23 起支持 Containerd	OverlayFS	runC

表5-3 CCE Turbo 集群节点操作系统与容器引擎对应关系

节点类型	操作系统	内核版本	容器引擎	容器存储 Rootfs	容器运行时
弹性云主机-虚拟机	CentOS 7.6	3.x	Docker Containerd	OverlayFS	runC
	Ubuntu 18.04	4.x			
	EulerOS 2.9	4.x			
弹性云主机-物理机	EulerOS 2.9	4.x	Containerd	Device Mapper	Kata

表5-4 鲲鹏节点操作系统与容器引擎对应关系

操作系统	内核版本	容器引擎	容器存储 Rootfs	容器运行时
EulerOS 2.8	4.x	Docker	OverlayFS	runC

Containerd 和 Docker 组件常用命令对比

Containerd 不支持 dockerAPI 和 dockerCLI，但是可以通过 cri-tool 命令实现类似的功能。

表5-5 镜像相关功能

序号	Docker 命令	Containerd 命令	备注
1	docker images [选项] [镜像名[:标签]]	crictl images [选项] [镜像名[:标签]]	列出本地镜像列表
2	docker pull [选项] 镜像名[:标签|@DIGEST]	crictl pull [选项] 镜像名[:标签|@DIGEST]	拉取镜像
3	docker push	无	上传镜像
4	docker rmi [选项] 镜像...	crictl rmi [选项] 镜像 ID…	删除本地镜像
5	docker inspect 镜像 ID	crictl inspect 镜像 ID	检查容器

表5-6 容器相关功能

序号	Docker 命令	Containerd 命令	备注
1	docker ps [选项]	crictl ps [选项]	列出容器列表
2	docker create [选项]	crictl create [选项]	创建容器
3	docker start [选项] 容器 ID...	crictl start [选项] 容器 ID...	启动容器
4	docker stop [选项] 容器 ID...	crictl stop [选项] 容器 ID...	停止容器
5	docker rm [选项] 容器 ID...	crictl rm [选项] 容器 ID...	删除容器
6	docker attach [选项] 容器 ID	crictl attach [选项] 容器 ID	连接容器
7	docker exec [选项] 容器 ID 启动命令 [参数...]	crictl exec [选项] 容器 ID 启动命令[参数...]	进入容器
8	docker inspect [选项] 容器 NAME|ID...	crictl inspect [选项] 容器 ID...	查看容器详情
9	docker logs [选项] 容器 ID	crictl logs [选项] 容器 ID	查看容器日志
10	docker stats [选项] [容器 ID...]	crictl stats [选项] [容器 ID]	查看容器的资源使用情况

序号	Docker 命令	Containerd 命令	备注
11	docker update [选项] 容器 ID...	crictl update [选项] 容器 ID...	更新容器资源限制

表5-7 Pod 相关功能

序号	Docker 命令	Containerd 命令	备注
1	无	crictl pods [选项]	列出 Pod 列表
2	无	crictl inspectp [选项] POD-ID...	查看 Pod 详情
3	无	crictl start [选项] POD-ID...	启动 Pod
4	无	crictl runp [选项] POD-ID...	运行 Pod
5	无	crictl stopp [选项] POD-ID...	停止 Pod
6	无	crictl rmp [选项] POD-ID...	删除 Pod

说明

Containerd 创建并启动的容器会被 kubelet 立即删除，不支持暂停、恢复、重启、重命名、等待容器，Containerd 不具备 docker 构建、导入、导出、比较、推送、查找、打标签镜像的能力，Containerd 不支持拷贝文件，可通过修改 containerd 的配置文件实现登录镜像仓库。

调用链区别

- Docker（Kubernetes 1.23 及以下版本）：

  kubelet --> docker shim （在 kubelet 进程中） --> docker --> containerd
- Docker（Kubernetes 1.24 及以上版本社区方案）：

  kubelet --> cri-dockerd （kubelet 使用 cri 接口对接 cri-dockerd） --> docker--> containerd
- Containerd：

  kubelet --> cri plugin（在 containerd 进程中） --> containerd

其中 Docker 虽增加了 swarm cluster、docker build、docker API 等功能，但也会引入一些 bug，并且与 Containerd 相比，多了一层调用，因此 **Containerd 被认为更加节省资源且更安全**。

容器引擎版本说明

- Docker
  - CentOS：docker-engine 18.9.0，CCE 定制的 Docker 版本，会及时修复安全漏洞。

- Ubuntu：docker-ce: 18.9.9，开源社区版本。Ubuntu 节点建议使用 containerd 引擎（当前 CCE Turbo 集群支持 containerd, CCE 集群后续也将提供 containerd）。

说明

Ubuntu 下开源 docker-ce 在并发 exec（如配置了多个 exec 探针时）可能触发社区 bug，建议使用 http/tcp 的探针。

- Containerd：1.4.1

5.1.3 节点操作系统

集群版本与操作系统对应关系

如下为当前已经发布的集群版本与操作系统版本的对应关系，请参考：

表5-8 虚拟机节点操作系统与集群版本对应表

操作系统	集群版本	内核信息
CentOS Linux release 7.6	v1.23	3.10.0-1160.66.1.el7.x86_64
	v1.21	3.10.0-1160.66.1.el7.x86_64
	v1.19.16	3.10.0-1160.66.1.el7.x86_64
	v1.19.10	3.10.0-1160.25.1.el7.x86_64
	v1.19.8	3.10.0-1160.15.2.el7.x86_64
	v1.17.17（停止维护）	3.10.0-1160.15.2.el7.x86_64
	v1.17.9（停止维护）	3.10.0-1062.12.1.el7.x86_64
	v1.15.11（停止维护）	3.10.0-1062.12.1.el7.x86_64
	v1.15.6-r1（停止维护）	3.10.0-1062.1.1.el7.x86_64
	v1.13.10-r1（停止维护）	3.10.0-957.21.3.el7.x86_64
	v1.13.7-r0（停止维护）	3.10.0-957.21.3.el7.x86_64
Ubuntu 18.04 server 64bit	v1.23	4.15.0-171-generic
	v1.21	4.15.0-171-generic
	v1.19.16	4.15.0-171-generic
	v1.19.8	4.15.0-136-generic

操作系统	集群版本	内核信息
	v1.17.17（停止维护）	4.15.0-136-generic

5.1.4 安全容器与普通容器

安全容器和普通容器相比，它最主要的区别是每个容器（准确地说是 pod）都运行在一个单独的微型虚拟机中，拥有独立的操作系统内核，以及虚拟化层的安全隔离。因为云容器引擎 CCE 的容器安全隔离比独立拥有私有 Kubernetes 集群有更严格的要求。通过安全容器，不同容器之间的内核、计算资源、网络都是隔离开的，保护了 Pod 的资源和数据不被其他 Pod 抢占和窃取。

CCE Turbo 集群下单节点支持普通容器和安全容器，您可以根据业务需求选择使用，两者的区别如下：

分类	安全容器	Docker 普通容器	Containerd 普通容器
容器所在节点类型	物理机	虚拟机	虚拟机
容器引擎	Containerd	Docker	Containerd
容器运行时	Kata	runC	runC
容器内核	独占内核	与宿主机共享内核	与宿主机共享内核
容器隔离方式	轻量虚拟机	Cgroups 和 Namespace	Cgroups 和 Namespace
容器引擎存储驱动	Device Mapper	OverlayFS2	OverlayFS
Pod Overhead	内存：100MiB CPU：0.1Core Pod Overhead 为安全容器本身资源占用。比如 Pod 申请的 limits.cpu = 0.5Core 和 limits.memory = 256MiB，那么该 Pod 最终会申请 0.6Core 的 CPU 和 356MiB 的内存。	无	无
最小规格	内存：256MiB CPU：0.25Core 安全容器的 CPU 核数（单位为 Core）与内存	无	无

分类	安全容器	Docker 普通容器	Containerd 普通容器
	（单位为 GiB）配比建议在 1:1 至 1:8 之间。例如 CPU 为 0.5Core，则内存范围建议在 512MiB-4GiB 间。		
容器引擎命令行	crictl	docker	crictl
Pod 的计算资源	CPU 和内存的 request 和 limit 必须一致	CPU 和内存的 request 和 limit 可以不一致	CPU 和内存的 request 和 limit 可以不一致
hostnetwork	不支持	支持	支持

5.1.5 节点可创建最大 pod 数量说明

节点可以创建最大 Pod 数量由如下参数决定。

- 节点可分配容器 IP 数（alpha.cce/fixPoolMask）：在创建 CCE 集群时配置，仅网络模型为“VPC 网络”需要配置。
- 节点最大实例数（maxPods）：在创建节点时配置，是 kubelet 的配置项。

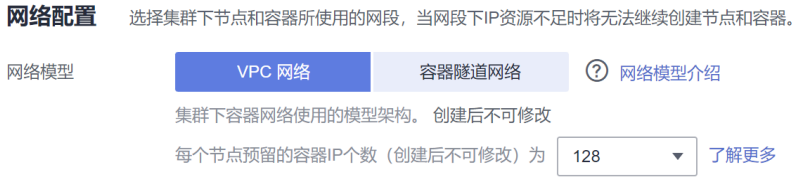
节点上最多能创建多少个 Pod，取决于这几个参数的最小值。

- 对于“容器隧道网络”的集群，仅取决于节点最大实例数。
- 对于“VPC 网络”的集群，取决于节点最大实例数和节点可分配容器 IP 数的最小值，即 **min(节点最大实例数，节点可分配容器 IP 数)**。
- 对于“云原生 2.0 网络”的集群（CCE Turbo 集群），取决于节点最大实例数和 CCE Turbo 集群节点网卡数量中的最小值。建议节点最大实例数不要超过节点网卡数，否则当节点规格可分配网卡不足时 Pod 实例可能无法正常调度。

节点可分配容器 IP 数

在创建 CCE 集群时，如果网络模型选择“VPC 网络”，会让您选择每个节点可供分配的容器 IP 数量。

该参数会影响节点上可以创建最大 Pod 的数量，因为每个 Pod 会占用一个 IP（使用容器网络），如果可用 IP 数量不够的话，就无法创建 Pod。



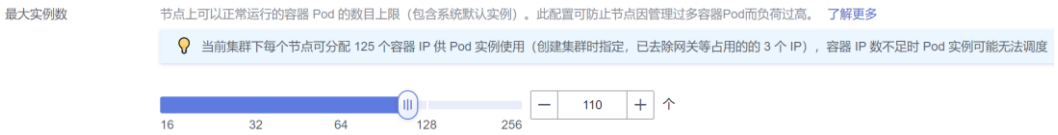
节点默认会占用掉 3 个容器 IP 地址（网络地址、网关地址、广播地址），因此节点上**可分配给容器使用的 IP 数量 = 您选择的容器 IP 数量 - 3**，例如上面图中可分配给容器使用的 IP 数量为 128-3=125。

节点网卡数量（仅 CCE Turbo 集群）

CCE Turbo 集群 ECS 节点使用弹性辅助网卡，物理机节点使用弹性网卡，节点可以创建最大 Pod 数量与节点可使用网卡数量相关。

节点最大实例数

在创建节点时，可以配置节点可以创建的最大实例数。该参数是 kubelet 的配置参数，决定 kubelet 最多可创建多少个 Pod。



5.1.6 节点预留资源计算公式

节点的部分资源需要运行一些必要的 Kubernetes 系统组件和 Kubernetes 系统资源，使该节点可作为您的集群的一部分。 因此，您的节点资源总量与节点在 Kubernetes 中的可分配资源之间会存在差异。节点的规格越大，在节点上部署的容器可能会越多，所以 kubernetes 自身需预留更多的资源。

为了保证节点的稳定性，CCE 集群节点上会根据节点的规格预留一部分资源给 Kubernetes 的相关组件（kubelet，kube-proxy 以及 docker 等）。

CCE 对用户节点可分配的资源计算法则如下：

Allocatable = Capacity - Reserved - Eviction Threshold

即，**节点资源可分配量=总量-预留值-驱逐阈值**。其中内存资源的驱逐阈值，固定为 100MB。

当节点上所有 Pod 消耗的内存上涨时，存在下列两种行为：

1. 内存大于等于节点可分配量，触发 kubelet 驱逐 Pod。
2. 内存节点接近可分配量与驱逐（总量-预留值），触发操作系统 OOM。

CCE 对节点内存的预留规则 v1

对于 **v1.21.4-r0** 和 **v1.23.3-r0** 以上版本集群，节点内存的预留规则优化为 V2 模型，请参见 CCE 对节点内存的预留规则 v2。

CCE 节点内存的总预留值等于**系统组件预留值**与 **Kubelet 管理 Pod 所需预留值**之和。

公式为：总预留值 = 系统组件预留值 + Kubelet 管理 Pod 所需预留值

表5-9 系统组件预留规则

内存总量范围	系统组件预留值
内存总量 <= 8GB	0MB
8GB < 内存总量 <= 16GB	((内存总量 - 8GB)*1024*10%)MB
16GB < 内存总量 <= 128GB	(8GB*1024*10% + (内存总量 - 16GB)*1024*6%)MB
内存总量 > 128GB	(8GB*1024*10% + 112GB*1024*6% + (内存总量 - 128GB)*1024*2%)MB

表5-10 Kubelet 管理 Pod 所需预留规则

内存总量范围	Pod 数量	Kubelet 管理 Pod 所需预留值
内存总量 <= 2GB	-	内存总量 *25%
内存总量 > 2GB	0 < 节点的最大实例数 <= 16	700 MB
	16 < 节点的最大实例数 <= 32	(700 + (节点的最大实例数 - 16)*18.75)MB
	32 < 节点的最大实例数 <= 64	(1024 + (节点的最大实例数 - 32)*6.25)MB
	64 < 节点的最大实例数 <= 128	(1230 + (节点的最大实例数 - 64)*7.80)MB
	节点的最大实例数 > 128	(1740 + (节点的最大实例数 - 128)*11.20)MB

须知

对于小规格节点，需根据实际使用情况调整节点的最大实例数。或者在 CCE 控制台创建节点时，需考虑根据节点规格自适应调整节点的最大实例数参数。

CCE 对节点内存的预留规则 v2

对于 **v1.21.4-r0** 和 **v1.23.3-r0** 以上版本集群，节点内存的预留规则优化为 V2 模型，且支持通过节点池配置管理参数（kube-reserved-mem 和 system-reserved-mem）动态调整，具体方法请参见管理节点池。

CCE 节点内存 V2 模型的总预留值等于 OS 侧预留与 CCE 管理 Pod 所需预留值之和。

其中 OS 侧预留包括基础预留和随节点内存规格变动的浮动预留；CCE 侧预留包括基础预留和随节点 Pod 数量的浮动预留。

表5-11 节点内存预留规则 v2

预留类型	基础/浮动	预留公式	预留对象
OS 侧预留	基础预留	固定 400MB	sshd、systemd-journald 等操作系统服务组件占用
	浮动预留（随节点内存）	25MB/GB	内核占用
CCE 侧预留	基础预留	固定 500MB	节点空载时， kubelet、kube-proxy 等容器引擎组件占用。
	浮动预留（随节点 Pod 数量）	Docker：20MB/Pod Containerd：5MB/Pod	Pod 数量增加时，容器引擎组件的额外占用。 说明 v2 模型在计算节点默认预留内存时，随内存估计节点默认最大实例数，参见节点默认最大实例数。

CCE 对节点 CPU 的预留规则

表5-12 节点 CPU 预留规则

CPU 总量范围	CPU 预留值
CPU 总量 <= 1core	CPU 总量 *6%
1core < CPU 总量 <= 2core	1core*6% + (CPU 总量 - 1core)*1 %
2core < CPU 总量 <= 4core	1core*6% + 1core*1% + (CPU 总量 - 2core)*0.5%
CPU 总量 > 4core	1core*6% + 1core*1% + 2core*0.5% + (CPU 总量 - 4core)*0.25%

节点默认最大实例数

表5-13 节点默认最大实例数

内存	节点默认最大实例数
4G	20
8G	40
16G	60
32G	80

内存	节点默认最大实例数
64G 及以上	110

5.1.7 数据盘空间分配说明

本章节将详细介绍节点数据盘空间分配的情况，以便您根据业务实际情况配置数据盘大小。

在创建节点时，您需要配置节点数据盘，且数据盘容量不小于 100G。您可“单击展开高级配置”，自定义节点数据盘的空间分配。

- 自定义容器引擎空间大小说明：CCE 将数据盘空间划分为两块：一块用于存放容器引擎 (Docker/Containerd) 工作目录、容器镜像的数据和镜像元数据；另一块用于 Kubelet 组件和 EmptyDir 临时存储等。容器引擎空间的剩余容量将会影响镜像下载和容器的启动及运行。
  - 容器引擎和容器镜像空间（默认占 90%）：用于容器运行时工作目录、存储容器镜像数据以及镜像元数据。
  - Kubelet 组件和 EmptyDir 临时存储（默认占 10%）：用于存储 Pod 配置文件、密钥以及临时存储 EmptyDir 等挂载数据。
- 自定义 Pod 容器空间大小：即容器的 basesize 设置，每个工作负载下的容器组 Pod 占用的磁盘空间设置上限（包含容器镜像占用的空间）。合理的配置可避免容器组无节制使用磁盘空间导致业务异常。建议此值不超过容器引擎空间的 80%。该参数与节点操作系统和容器存储 Rootfs 相关，部分场景下不支持设置。

自定义容器引擎空间大小

数据盘根据容器存储 Rootfs 不同具有两种划分方式（以 100G 大小为例）：**DeviceMapper 类型**和 **OverlayFS 类型**。

您可以登录到节点通过 docker info 命令查看存储引擎类型，如下所示。

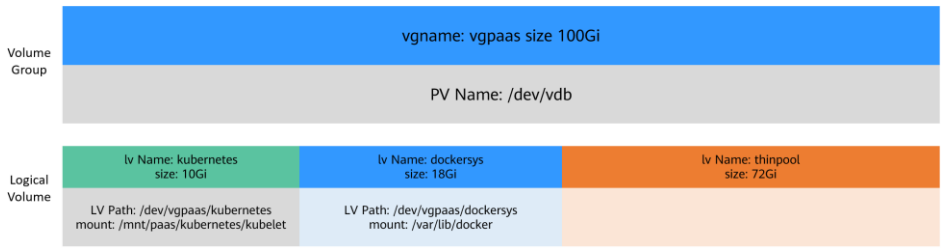
```
# docker info
Containers: 20
 Running: 17
 Paused: 0
 Stopped: 3
Images: 16
Server Version: 18.09.0
Storage Driver: devicemapper
```

- **Device Mapper 类型存储 Rootfs**

  其中默认占 90%的容器引擎和容器镜像空间又可分为以下两个部分：

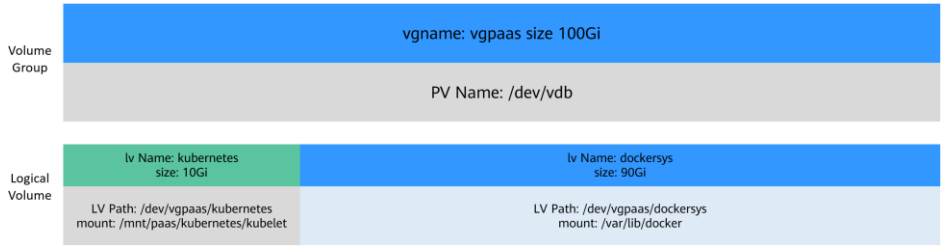
  - 其中/var/lib/docker 用于 Docker 工作目录，默认占比 20%，其空间大小 = **数据盘空间 * 90% * 20%**
  - thinpool 用于存储容器镜像数据、镜像元数据以及容器使用的磁盘空间，默认占比为 80%，其空间大小 = **数据盘空间 * 90% * 80%**

    thinpool 是动态挂载，在节点上使用 **df -h** 命令无法查看到，使用 **lsblk** 命令可以查看到。



- **OverlayFS 类型存储 Rootfs**

  相比 Device Mapper 存储引擎，没有单独划分 thinpool，容器引擎和容器镜像空间（默认占 90%）都在/var/lib/docker 目录下。



容器存储 Rootfs 情况如下：

- CCE 集群：EulerOS 2.5 操作系统使用 Device Mapper，Ubuntu 18.04 和 EulerOS 2.9 使用 OverlayFS。CentOS 7.6 在 1.19.16 以下版本集群中使用 Device Mapper，1.19.16 及以上版本集群使用 OverlayFS。EulerOS 2.8 系统在集群版本 v1.19.16-r2 前使用 Device Mapper, v1.19.16-r2 及之后版本使用 OverlayFS。

自定义 Pod 容器空间大小

自定义 Pod 容器空间（basesize）设置与节点操作系统和容器存储 Rootfs 相关（容器存储 Rootfs 可登录到节点通过 **docker info** 命令查看）：

- Device Mapper 模式下支持自定义 Pod 容器空间（basesize）配置，默认值为 10G。
- OverlayFS 模式默认不限制 Pod 容器空间大小。

配置 Pod 容器空间（basesize）时，需要同时考虑节点的最大实例数配置。理想情况下，容器引擎空间需要大于容器使用的磁盘总空间，即：**容器引擎和容器镜像空间（默认占 90%） > 容器数量 * Pod 容器空间（basesize）**。否则，可能会引起节点分配的容器引擎空间不足，从而导致容器启动失败。

对于支持配置 basesize 的节点，尽管可以限制单个容器的主目录大小（开启时默认为 10GB），但节点上的所有容器还是共用节点的 thinpool 磁盘空间，并不是完全隔离，当一些容器使用大量 thinpool 空间且总和达到节点 thinpool 空间上限时，也会影响其他容器正常运行。

另外，在容器的主目录中创删文件后，其占用的 thinpool 空间不会立即释放，因此即使 basesize 已经配置为 10GB，而容器中不断创删文件时，占用的 thinpool 空间会不断增加一直到 10GB 为止，后续才会复用这 10GB 空间。如果节点上的容器数量 *basesize > 节点 thinpool 空间大小，理论上有概率出现节点 thinpool 空间耗尽的场景。

镜像回收策略说明

当容器引擎空间不足时，会触发镜像垃圾回收。

镜像垃圾回收策略只考虑两个因素：HighThresholdPercent 和 LowThresholdPercent。磁盘使用率超过上限阈值（HighThresholdPercent，默认值为 85%）将触发垃圾回收。垃圾回收将删除最近最少使用的镜像，直到磁盘使用率满足下限阈值（LowThresholdPercent，默认值为 80%）。

容器引擎空间大小配置建议

- 容器引擎空间需要大于容器使用的磁盘总空间，即：**容器引擎空间 > 容器数量 * Pod 容器空间（basesize）**
- 容器业务的创删文件操作建议在容器挂载的本地存储（如 emptyDir、hostPath）或云存储的目录中进行，这样不会占用 thinpool 空间。其中 Emptydir 使用的是 kubelet 空间，需要规划好 kubelet 空间的大小。
- 容器运行时使用 OverlayFS 存储模式，当前 CCE 集群中 Ubuntu 18.04 节点容器已默认使用 OverlayFS 存储模式，1.19.16 版本及以上集群 CentOS 7.6 使用 OverlayFS 模式，可将业务部署在此类节点上，避免容器内创删文件后占用的磁盘空间不立即释放问题。

5.2 创建节点

前提条件

- 已创建至少一个集群。
- 您需要新建一个密钥对，用于远程登录节点时的身份认证。

  若使用密码登录节点，请跳过此操作。

约束与限制

- 为了保证节点的稳定性，CCE 集群节点上会根据节点的规格预留一部分资源给 Kubernetes 的相关组件（kubelet、kube-proxy 以及 docker 等）和 Kubernetes 系统资源，使该节点可作为您的集群的一部分。 因此，您的节点资源总量与节点在 Kubernetes 中的可分配资源之间会存在差异。节点的规格越大，在节点上部署的容器可能会越多，所以 Kubernetes 自身需预留更多的资源，详情请参见节点预留资源计算公式。
- 节点的网络（如虚机网络、容器网络等）均被 CCE 接管，不允许用户自行添加删除网卡或改变路由，若自行修改，CCE 不保证服务可用。例如，节点上名为的 gw_11cbf51a@eth0 网卡为容器网络网关，不可修改。
- 节点购买后如需对其进行升级或降低规格等操作，请将节点关机后进行变更，也可以重新购买节点后，将老节点删除。
- 创建节点过程中会使用域名方式从 OBS 下载软件包，需要能够使用云上内网 DNS 解析 OBS 域名，否则会导致创建不成功。为此，节点所在子网需要配置为内网 DNS 地址，从而使得节点使用内网 DNS。在创建子网时 DNS 默认配置为内网 DNS，如果您修改过子网的 DNS，请务必**确保子网下的 DNS 服务器可以解析 OBS 服务域名**，否则需要将 DNS 改成内网 DNS。
- 集群中的节点一旦创建后不可变更可用区。
- 集群开启 IPv4/IPv6 双栈时，所选节点子网不允许开启 DHCP 无限租约。

操作步骤

集群创建完成后，您可以在集群中创建节点。

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”，单击要创建节点的集群进入集群控制台。

步骤 2 在集群控制台左侧导航栏中选择“节点管理”，单击右侧“创建节点”，在节点配置步骤中参照如下表格设置节点参数。

计算配置：

配置节点云主机的规格与操作系统，为节点上的容器应用提供基本运行环境。

表5-14 计算配置参数

参数	参数说明
计费模式	支持如下两种计费方式。 • 包年包月 包年包月需要选择购买时长。 • 按需计费
可用区	节点云主机所在的可用区，集群下节点创建在不同可用区下可以提高可靠性。创建后不可修改。 建议您选择“随机分配”，可根据选择的节点规格随机分配一个可以使用的可用区。 可用区是在同一区域下，电力、网络隔离的物理区域，可用区之间

参数	参数说明
	内网互通，不同可用区之间物理隔离。如果您需要提高工作负载的高可靠性，建议您将云主机创建在不同的可用区。
节点类型	CCE 集群支持弹性云主机虚拟机和物理机。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 公共镜像：请选择节点对应的操作系统。
节点名称	节点云主机使用的名称，批量创建时将作为云主机名称的前缀。 系统会默认生成名称，支持修改。 节点名称以小写字母开头，支持小写字母、数字和中划线(-)，不能以中划线(-)结尾。
登录方式	**密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。

存储配置：

配置节点云主机上的存储资源，方便节点上的容器软件与容器应用使用。请根据实际场景设置磁盘大小。

表5-15 存储配置参数

参数	参数说明
系统盘	节点云主机使用的系统盘，供操作系统使用。您可以设置系统盘的规格为 40GB-1024GB 之间的数值，缺省值为 50GB。 加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**目前仅在部分 Region 显示此选项，具体以界面为准。** • 默认不加密。 • 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。
数据盘	节点云主机使用的数据盘，供容器运行时和 Kubelet 组件使用。您可以设置数据盘的规格为 100GB-32768GB 之间的数值，缺省值为 100GB。 **至少需要一块数据盘**，供容器运行时和 Kubelet 组件使用，**该数据**

参数	参数说明
	盘不能被删除卸载，否则会导致节点不可用。 单击后方的“展开高级设置”可进行如下设置： • 自定义空间分配：勾选后可定义容器运行时在数据盘上占用的空间比例，容器运行时的空间用于存放容器运行时工作目录、容器镜像数据以及镜像元数据。数据盘空间分配详细说明请参见数据盘空间分配说明。 • 加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**目前仅在部分 Region 显示此选项，具体以界面为准。** – 默认不加密。 – 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 **添加多个数据盘** 最多可以添加 4 个，默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，选择如下配置。 • 默认：默认情况直接创建为裸盘，不做任何处理。 • 挂载到指定目录：将数据盘挂载到指定目录。 • 作为持久存储卷：适用于对 PV 有性能要求的场景。持久存储卷的节点会添加上 node.kubernetes.io/local-storage-persistent 标签，取值为 linear 或 striped。 • 作为临时存储卷：适用于对 EmptyDir 有性能要求的场景。 说明 持久存储卷和临时存储卷仅在集群版本 >= v1.21.2-r0 时支持创建，且临时存储卷需要 Everest 插件版本>=1.2.29，持久存储卷需要 Everest 插件版本>=1.2.31。 持久存储卷和临时存储卷支持如下两种写入模式。 • 线性（linear）：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。 • 条带化（striped）：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。多块存储卷才能选择条带化。 **本地盘说明** 节点规格为“磁盘增强型”或“超高 I/O 型”时，有一块数据盘可以是本地盘。 本地磁盘实例有宕机风险，不保证数据可靠性，建议您使用云硬盘存储您的业务数据。

网络配置：

配置节点云主机的网络资源，用于访问节点和容器应用。

表5-16 网络配置参数

参数	参数说明
节点子网	节点子网默认使用创建集群时的子网配置，也可以选择其他子网。
节点 IP	支持指定节点 IP 地址。默认随机分配。

高级配置：

节点能力增强，可在此配置节点的标签、污点、启动命令等功能。

表5-17 高级配置参数

参数	参数说明
K8S 标签	单击“添加标签”可以设置附加到 Kubernetes 对象（比如 Pods）上的键值对，最多可以添加 10 条标签 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器 Pod 调度到指定节点的功能。详细请参见 Labels and Selectors。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 CCE 服务会自动帮您创建 CCE-Dynamic-Provisioning-Node=节点 id 的标签。
污点(Taints)	默认为空。支持给节点加 Taints 来设置反亲和性，每个节点最多配置 10 条 Taints，每条 Taints 包含以下 3 个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符；另外可以使用 DNS 子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符。 • Effect：只可选 NoSchedule，PreferNoSchedule 或 NoExecute。 Taints 的使用请参见管理节点污点（taint）。 **说明** 对于 1.19 及以下版本集群，有可能会出现污点打上之前负载已经调度到节点上，如果需要避免这种情况，请选择 1.19 及以上集群。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为 16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。 节点最多能创建多少个 Pod 还受其他因素影响，具体请参见节

参数	参数说明
	点最多可以创建多少个 Pod。
云主机组	云主机组是对云主机的一种逻辑划分，同一云主机组中的云主机遵从同一策略。 反亲和性策略：同一云主机组中的云主机分散地创建在不同主机上，提高业务的可靠性。 选择已创建的云主机组，或单击“新建云主机组”创建，创建完成后单击刷新按钮。
安装前执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装前执行，可能导致 Kubernetes 软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装后执行，不影响 Kubernetes 软件安装。
委托	委托是由租户管理员在统一身份认证服务上创建的。通过委托，可以将云主机资源共享给其他帐号，或委托更专业的人或团队来代为管理。 如果没有委托请单击右侧“新建委托”创建。

步骤 3 单击“下一步：规格确认”，确认所设置的服务选型参数、规格和费用等信息，且确认已阅读并知晓服务协议。

步骤 4 确认规格和费用后，单击“提交”，节点开始创建。

若选择购买“包年包月”的节点，请单击“去支付”，根据界面提示进行付款操作。

系统将自动跳转到节点列表页面，待节点状态为“运行中”，表示节点添加成功。添加节点预计需要 6-10 分钟左右，请耐心等待。

步骤 5 单击“返回节点列表”，待状态为运行中，表示节点创建成功。

----结束

5.3 纳管节点

操作场景

CCE 集群支持两种添加节点的方式：创建节点和纳管节点，纳管节点是指将“已有的 ECS/BMS 加入到 CCE 集群中”，所纳管节点的计费模式支持“按需计费”和“包年/包月”两种类型。

须知

- 纳管时，会将所选弹性云主机的操作系统重置为 CCE 提供的标准镜像，以确保节点的稳定性，请选择操作系统及重置后的登录方式。
- 所选弹性云主机挂载的系统盘、数据盘都会在纳管时被格式化，请确保信息已备份。
- 纳管过程中，请勿在弹性云主机控制台对所选虚拟机做任何操作。

约束与限制

- 集群版本需 1.15 及以上。
- 纳管节点支持 ECS（弹性云主机）节点、BMS（物理机）节点。
- 集群开启 IPv6 后，只支持纳管所在的子网开启了 IPv6 功能的节点；集群未开启 IPv6，只支持纳管所在的子网未开启 IPv6 功能的节点。
- 原虚拟机节点创建时若已设置密码或密钥，需等待虚拟机节点可用 10 分钟后方可纳管。纳管时您需要重新设置密码或密钥，原有的密码或密钥将会失效。

前提条件

支持纳管符合如下条件的云主机：

- 待纳管节点必须状态为“运行中”，未被其他集群所使用，且不携带 CCE 专属节点标签 CCE-Dynamic-Provisioning-Node。
- 待纳管节点需与集群在同一虚拟私有云内（若集群版本低于 1.13.10，纳管节点还需要与 CCE 集群在同一子网内）。
- 待纳管节点需挂载数据盘，数据盘需满足至少有 1 块，容量不少于 100GB。关于节点挂载数据盘的操作说明，请参考新增磁盘。
- 待纳管节点规格要求：CPU 必须 2 核及以上，内存必须 4GB 及以上，网卡有且仅能有一个。
- 如果使用了企业项目，则待纳管节点需要和集群在同一企业项目下，不然在纳管时会识别不到资源，导致无法纳管。
- 批量纳管仅支持添加相同规格、相同可用区、相同数据盘配置的云主机。

操作步骤

步骤 1 登录 CCE 控制台，进入要纳管节点的集群。

步骤 2 在左侧列表中选择节点管理，单击右上角纳管节点。

步骤 3 配置节点参数。

计算配置

表5-18 计算配置参数

参数	参数说明
节点规格	单击添加已有云主机，选择要纳管的服务器。

参数	参数说明
	可以选择多台云主机批量纳管，但批量纳管仅支持添加相同规格、相同可用区、相同数据盘配置的云主机。 如果云主机有多块数据盘，需要选择其中一块作为供容器运行时和Kubelet 组件使用。
容器引擎	CCE 集群支持 Docker。
操作系统	公共镜像：请选择节点对应的操作系统。
登录方式	**密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。

存储配置

配置节点云主机上的存储资源，方便节点上的容器软件与容器应用使用。

表5-19 存储配置参数

参数	参数说明
系统盘	直接使用云主机的系统盘。
数据盘	**至少需要一块数据盘**，供容器运行时和 Kubelet 组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** 单击后方的“展开高级设置”可设置自定义空间分配：勾选后可定义容器运行时在数据盘上占用的空间比例，容器运行时的空间用于存放容器运行时工作目录、容器镜像数据以及镜像元数据。数据盘空间分配详细说明请参见数据盘空间分配说明。 **其他数据盘**默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，将磁盘挂载到指定目录。另外还可以作为持久存储卷或临时存储卷，具体使用请参见本地持久存储卷和临时存储卷。

高级配置

表5-20 高级配置参数

参数	参数说明
K8S 标签	单击“添加标签”可以设置附加到 Kubernetes 对象（比如

参数	参数说明
	Pods）上的键值对，最多可以添加 10 条标签 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器 Pod 调度到指定节点的功能。详细请参见 Labels and Selectors。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 CCE 服务会自动帮您创建 CCE-Dynamic-Provisioning-Node=节点 id 的标签。
污点(Taints)	默认为空。支持给节点加 Taints 来设置反亲和性，每个节点最多配置 10 条 Taints，每条 Taints 包含以下 3 个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符；另外可以使用 DNS 子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符。 • Effect：只可选 NoSchedule，PreferNoSchedule 或 NoExecute。 须知 • Taints 配置时需要配合 Pod 的 toleration 使用，否则可能导致扩容失败或者 Pod 无法调度到扩容节点。 • 节点池创建后可单击列表项的“编辑”修改配置，修改后将同步到节点池下的已有节点。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为 16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。
安装前执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装前执行，可能导致 Kubernetes 软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装后执行，不影响 Kubernetes 软件安装。

步骤 4 单击“下一步：规格确认”，确认已阅读并知晓服务协议，并单击“提交”。

----结束

5.4 移除节点

操作场景

在集群中移除节点会将该节点移出集群，然后重装节点的操作系统，并清理节点上的 CCE 组件。

移除不会删除节点对应的服务器。移除前请确认您的正常业务运行不受影响，请谨慎操作。

节点移出集群后会继续开机运行，并继续产生费用。

约束限制

- 当且仅当 CCE 集群状态为运行中或不可用时允许移除节点。
- 当且仅当 CCE 节点状态为运行中、不可用或错误时允许被移除。
- 为使 CCE 节点正常移除，且移除后能正常重装操作系统清理 CCE 组件，请确保服务器处于正常运行中状态。
- 若节点在 CCE 集群移除后重装操作系统失败，请手动完成失败节点的操作系统重装，并在重装后登录节点执行清理脚本完成 CCE 组件清理，具体步骤参见重装操作系统失败如何处理。
- 移除节点会导致与节点关联的本地持久存储卷类型的 PVC/PV 数据丢失，无法恢复，且 PVC/PV 无法再正常使用。移除节点时使用了本地持久存储卷的 Pod 会从移除的节点上驱逐，并重新创建 Pod，Pod 会一直处于 pending 状态，因为 Pod 使用的 PVC 带有节点标签，由于冲突无法调度成功。

注意事项

- 移除节点会涉及 Pod 迁移，可能会影响业务，请在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 操作过程中，后台会把当前节点设置为不可调度状态。
- 移除节点重装操作系统后将清理原有的 LVM 分区，通过 LVM 管理的数据将会清空，请提前做好相关的数据备份。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧列表中选择节点管理，单击节点后的“更多 > 移除”。

图5-1 移除节点

您还可以选中多个节点一起移除，如下图所示。

图5-2 一次移除多个节点

步骤 3 在弹出的“移除节点”配置重装操作系统需要的登录信息，单击“是”，等待完成节点移除。

移除节点后，原有节点上的工作负载实例会自动迁移至其他可用节点。

----结束

重装操作系统失败如何处理

移除节点重装操作系统可能会失败，如果碰到这种情况，您可以执行如下步骤重装操作系统并清理节点上的 CCE 组件。

步骤 1 登录服务器的管理控制台，完成操作系统的重装，详细步骤请参见切换操作系统。

步骤 2 登录服务器，执行如下命令完成 CCE 组件和 LVM 数据的清理。

将如下脚本写入 **clean.sh** 文件。

```
lsblk
vgs --noheadings | awk '{print $1}' | xargs vgremove -f
pvs --noheadings | awk '{print $1}' | xargs pvremove -f
```

```
lvs --noheadings | awk '{print $1}' | xargs -i lvremove -f --select {}
function init_data_disk() {
   all_devices=$(lsblk -o KNAME,TYPE | grep disk | grep -v nvme | awk '{print $1}'
| awk '{ print "/dev/"$1}')
   for device in ${all_devices[@]}; do
      isRootDisk=$(lsblk -o KNAME,MOUNTPOINT $device 2>/dev/null| grep -E
'[[:space:]]/$' | wc -l )
      if [[ ${isRootDisk} != 0 ]]; then
         continue
      fi
      dd if=/dev/urandom of=${device} bs=512 count=64
      return
   done
   exit 1
}
init_data_disk
lsblk
```

执行如下命令。

bash clean.sh

----结束

5.5 重置节点

操作场景

您可以通过重置节点修改节点的配置，比如修改节点操作系统、登录方式等。

重置节点会重装节点操作系统，并重新安装节点上 Kubernetes 软件。如果您在使用过程中修改了节点上的配置等操作导致节点不可用，可以通过重置节点进行修复。

约束与限制

- v1.13 及以上版本的 CCE 集群支持重置节点。

注意事项

- 重置节点功能不会重置控制节点，仅能对工作节点进行重置操作，如果重置后节点仍然不可用，请删除该节点重新创建。
- **重置节点将对节点操作系统进行重置安装，节点上已运行的工作负载业务将会中断，请在业务低峰期操作。**
- **节点重置后系统盘和 docker 数据盘将会被清空，重置前请事先备份重要数据。**
- **用户节点如果有自行挂载了数据盘，重置完后会清除挂载信息，请事先备份重要数据，重置完成后请重新执行挂载行为，数据不会丢失。**
- 节点上的工作负载实例的 IP 会发生变化，但是不影响容器网络通信。
- 云硬盘必须有剩余配额。
- 操作过程中，后台会把当前节点设置为不可调度状态。

- 重置节点会导致与节点关联的本地持久存储卷类型的 PVC/PV 数据丢失，无法恢复，且 PVC/PV 无法再正常使用。重置节点时使用了本地持久存储卷的 Pod 会从重置的节点上驱逐，并重新创建 Pod，Pod 会一直处于 pending 状态，因为 Pod 使用的 PVC 带有节点标签，由于冲突无法调度成功。节点重置完成后，Pod 可能调度到重置好的节点上，此时 Pod 会一直处于 creating 状态，因为 PVC 对应的底层逻辑卷已经不存在了。

操作步骤

新 UI 支持批量重置节点。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧列表中选择要重置的节点，可以选多个。单击“更多 > 重置节点”。

步骤 3 在弹出的窗口中单击“是”。跳转到参数配置界面。

- 对于 DefaultPool 节点池下的节点，会跳转到配置参数界面，请参见步骤 4 进行配置。
- 对于用户创建的节点池下的节点，重置是不支持配置参数，直接使用节点池的配置镜像重置。

步骤 4 配置节点参数。

计算配置

表5-21 计算配置参数

参数	参数说明
节点规格	重置节点时不支持修改节点规格。
容器引擎	CCE 集群支持 Docker，1.23 起 CentOS、Ubuntu 支持 Containerd。
操作系统	公共镜像：请选择节点对应的操作系统。
登录方式	**密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。

存储配置

配置节点云主机上的存储资源，方便节点上的容器软件与容器应用使用。

表5-22 存储配置参数

参数	参数说明
系统盘	直接使用云主机的系统盘。
数据盘	**至少需要一块数据盘**，供容器运行时和 Kubelet 组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** 单击后方的“展开高级设置”可设置自定义空间分配：勾选后可定义容器运行时在数据盘上占用的空间比例，容器运行时的空间用于存放容器运行时工作目录、容器镜像数据以及镜像元数据。数据盘空间分配详细说明请参见数据盘空间分配说明。 **其他数据盘**默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，将磁盘挂载到指定目录。另外还可以作为持久存储卷或临时存储卷，具体使用请参见本地持久存储卷和临时存储卷。

高级配置

表5-23 高级配置参数

参数	参数说明
K8S 标签	单击“添加标签”可以设置附加到 Kubernetes 对象（比如 Pods）上的键值对，最多可以添加 10 条标签 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器 Pod 调度到指定节点的功能。详细请参见 Labels and Selectors。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 CCE 服务会自动帮您创建 CCE-Dynamic-Provisioning-Node=节点 id 的标签。
污点(Taints)	默认为空。支持给节点加 Taints 来设置反亲和性，每个节点最多配置 10 条 Taints，每条 Taints 包含以下 3 个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符；另外可以使用 DNS 子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符。 • Effect：只可选 NoSchedule，PreferNoSchedule 或 NoExecute。 须知 • Taints 配置时需要配合 Pod 的 toleration 使用，否则可能导致扩容失败或者 Pod 无法调度到扩容节点。 • 节点池创建后可单击列表项的“编辑”修改配置，修改后将同步到节点池下的已有节点。

参数	参数说明
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为 16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。
安装前执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装前执行，可能导致 Kubernetes 软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装后执行，不影响 Kubernetes 软件安装。

步骤 5 单击“下一步：规格确认”，确认已阅读并知晓服务协议。

步骤 6 单击“提交”。

----结束

5.6 登录节点

约束与限制

- SSH 方式登录时要求该节点（弹性云主机 ECS）已绑定弹性公网 IP。
- 只有运行中的弹性云主机才允许用户登录。
- Linux 操作系统用户名为 root。

登录方式概述

登录节点（弹性云主机 ECS）的方式有如下两种：

- **管理控制台远程登录（VNC 方式）**

  未绑定弹性公网 IP 的弹性云主机可通过管理控制台提供的远程登录方式直接登录。

  详细操作请参考：Linux 云主机远程登录（VNC 方式）。

- **SSH 方式登录**

  仅适用于 Linux 弹性云主机。您可以使用远程登录工具（例如 PuTTY、Xshell、SecureCRT 等）登录弹性云主机。如果普通远程连接软件无法使用，您可以使用云主机 ECS 管理控制台的管理终端连接实例，查看云主机操作界面当时的状态。

说明

- SSH 方式登录包括 **SSH 密钥**和 **SSH 密码**两种方式。
- 本地使用 Windows 操作系统登录 Linux 节点时，输入的镜像用户名（Auto-login username）为：root。

- CCE 控制台不提供针对节点的操作系统升级，也不建议您通过 yum 方式进行升级，如果您在节点上通过 yum update 升级了操作系统，会导致容器网络的组件不可用。

5.7 管理节点标签

节点标签可以给节点打上不同的标签，给节点定义不同的属性，通过这些标签可以快速的了解各个节点的特点。

节点标签使用场景

节点标签的主要使用场景有两类。

- 节点管理：通过节点标签管理节点，给节点分类。
- 工作负载与节点的亲和与反亲和：
  - 有的工作负载需要的 CPU 大，有的工作负载需要的内存大，有的工作负载需要 IO 大，可能会影响其他工作负载正常工作等等，此时建议给节点添加不同标签。在部署工作负载的时候，就可以选择相应标签的节点亲和部署，保证系统正常工作；反之，可以使用节点的反亲和部署。
  - 一个系统可以分为多个模块，每个模块由多个微服务组成，为保证后期运维的高效，可以将节点打上对应模块的标签，让各模块部署到各自的节点模块上，互不干扰，方便开发到各自节点上去维护。

节点固有标签

节点创建出来会存在一些固有的标签，并且是无法删除的，这些标签的含义请参见下表。

表5-24 节点固有标签

键	说明
新：topology.kubernetes.io/region 旧：failure-domain.beta.kubernetes.io/region	表示节点当前所在区域。
新：topology.kubernetes.io/zone 旧：failure-domain.beta.kubernetes.io/zone	表示节点所在区域的可用区。
新：node.kubernetes.io/baremetal 旧：failure-domain.beta.kubernetes.io/is-baremetal	表示是否为物理机节点。 例如：false，表示非物理机节点
node.kubernetes.io/container-engine	表示容器引擎。 例如：docker、containerd

键	说明
node.kubernetes.io/instance-type	节点实例规格。
kubernetes.io/arch	节点处理器架构。
kubernetes.io/hostname	节点名称。
kubernetes.io/os	节点操作系统类型。
node.kubernetes.io/subnetid	节点所在子网的 ID。
os.architecture	表示节点处理器架构。 例如：amd64，表示 AMD64 位架构的处理器
os.name	节点的操作系统名称。
os.version	操作系统节点内核版本。
node.kubernetes.io/container-engine	节点所用容器引擎。
accelerator	GPU 节点标签。
cce.cloud.com/cce-nodepool	节点池节点专属标签。

添加/删除节点标签

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，勾选节点，单击左上方“标签与污点管理”。

步骤 3 在弹出的窗口中，在“批量操作”下方单击“新增批量操作”，然后选择“添加/更新”或“删除”。

填写需要增加/删除标签的“键”和“值”，单击“确定”。

例如，填写的键为“deploy_qa”，值为“true”，就可以从逻辑概念表示该节点是用来部署 QA（测试）环境使用。

图5-3 添加节点标签

步骤 4 标签添加成功后，再次进入该界面，在节点数据下可查看到已经添加的标签。

----结束

5.8 管理节点污点（taint）

污点（taint）能够使节点排斥某些特定的 Pod，从而避免 Pod 调度到该节点上。

污点

节点污点是与“效果”相关联的键值对。以下是可用的效果：

- NoSchedule：不能容忍此污点的 Pod 不会被调度到节点上；现有 Pod 不会从节点中逐出。
- PreferNoSchedule：Kubernetes 会尽量避免将不能容忍此污点的 Pod 安排到节点上。
- NoExecute：如果 Pod 已在节点上运行，则会将该 Pod 从节点中逐出；如果尚未在节点上运行，则不会将其安排到节点上。

使用 **kubectl taint node** *nodename* 命令可以给节点增加污点，如下所示。

```
$ kubectl get node
NAME           STATUS   ROLES    AGE    VERSION
192.168.10.170   Ready    <none>   73d    v1.19.8-r1-CCE21.4.1.B003
192.168.10.240   Ready    <none>   4h8m   v1.19.8-r1-CCE21.6.1.2.B001
$ kubectl taint node 192.168.10.240 key1=value1:NoSchedule
node/192.168.10.240 tainted
```

通过 describe 命名和 get 命令可以查看到污点的配置。

```
$ kubectl describe node 192.168.10.240
Name:             192.168.10.240
...
Taints:           key1=value1:NoSchedule
...
$ kubectl get node 192.168.10.240 -oyaml
```

```
apiVersion: v1
...
spec:
  providerID: 06a5ea3a-0482-11ec-8e1a-0255ac101dc2
  taints:
  - effect: NoSchedule
    key: key1
    value: value1
...
```

去除污点可以使用如下命令，在 NoSchedule 后加一个“-”。

```
$ kubectl taint node 192.168.10.240 key1=value1:NoSchedule-
node/192.168.10.240 untainted
$ kubectl describe node 192.168.10.240
Name:            192.168.10.240
...
Taints:          <none>
...
```

在 CCE 控制台上同样可以管理节点的污点，且可以批量操作。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，勾选节点，单击左上方“标签与污点管理”。

步骤 3 在弹出的窗口中，在“批量操作”下方单击“新增批量操作”，单击“添加/更新”，选择“污点(Taints)”。

填写需要增加污点的“键”和“值”，选择污点的效果，单击“确定”。

图5-4 添加污点

步骤 4 污点添加成功后，再次进入该界面，在节点数据下可查看到已经添加的污点。

----结束

节点调度设置

在 CCE 控制台上可以配置调度设置，登录 CCE 控制台进入节点，选择“节点管理”，在节点列表右侧单击“更多 > 禁止调度”。

弹出如下对话框，单击“确认”，即可将节点设置为不可调度。

这个操作会给节点打上污点，使用 kubectl 可以查看污点的内容。

```
$ kubectl describe node 192.168.10.240
...
Taints:             node.kubernetes.io/unschedulable:NoSchedule
...
```

在 CCE 控制台相同位置再次设置，即可将去除污点，将节点设置为可调度。

容忍度（Toleration）

容忍度应用于 Pod 上，允许（但并不要求）Pod 调度到带有与之匹配的污点的节点上。

污点和容忍度相互配合，可以用来避免 Pod 被分配到不合适的节点上。 每个节点上都可以应用一个或多个污点，这表示对于那些不能容忍这些污点的 Pod，是不会被该节点接受的。

可以在 Pod 中容忍度。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
```

上面示例表示这个 Pod 容忍标签为 key1=value1，效果为 NoSchedule 的污点，所以这个 Pod 能够调度到对应的节点上。

同样还可以按如下方式写，表示当节点有 key1 这个污点时，可以调度到节点。

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

5.9 节点排水

操作场景

您可以通过控制台使用节点排水功能，系统会将节点设置为不可调度，然后安全地将节点上所有符合节点排水规则的 Pod 驱逐，后续新建的 Pod 都不会再调度到该节点。

在节点故障等场景下，该功能可帮助您快速隔离故障节点，被驱逐的 Pod 将会由工作负载 controller 转移到其他正常可调度的节点上。

约束与限制

仅以下指定版本的集群支持节点排水功能：

- v1.21 集群：v1.21.10-r0 及以上版本
- v1.23 集群：v1.23.8-r0 及以上版本
- v1.25 集群：v1.25.3-r0 及以上版本
- v1.25 以上版本集群

IAM 用户在使用节点排水功能时，至少需要具有以下一项权限，详情请参见命名空间权限（Kubernetes RBAC 授权）。

- cluster-admin（管理员权限）：对全部命名空间下所有资源的读写权限。
- drainage-editor：节点排水操作权限，可执行节点排水。
- drainage-viewer：节点排水只读权限，仅可查看节点排水状态，无法执行节点排水。

节点排水规格

节点排水功能会安全驱逐节点上的 Pod，但对于满足以下过滤规则的 Pod，系统会进行例外处理：

表5-25 节点排水规则

Pod 筛选条件	使用强制排水	不使用强制排水
Pod 的 status.phase 字段为 Succeeded 或 Failed	删除	删除
Pod 不受工作负载 controller 管理	删除	放弃排水
Pod 由 DaemonSet 管理	忽略	放弃排水
Pod 中挂载了 emptyDir 类型的 volume	驱逐	放弃排水
由 kubelet 直接管理的静态 Pod	忽略	忽略

说明

节点排水过程中可能会对 Pod 执行的操作如下：

- 删除：Pod 会从当前节点上删除，不会再重新调度至其他节点。
- 驱逐：Pod 会从当前节点上删除，且会重新调度至其他节点。
- 忽略：Pod 不会被驱逐或删除。
- 放弃排水：若节点上存在放弃排水的 Pod，节点排水过程会中止，不会驱逐或删除任何 Pod。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏选择“节点管理”，在右侧单击节点后的“更多 > 节点排水”。

步骤 3 在弹出的“节点排水”窗口中，进行排水设置。

- 超时时间（秒）：超过设定的时间后排水任务会自动失败，0 表示不设置超时时间。
- 强制排水：使用强制排水时，将忽略 DaemonSet 管理的 Pod，但会删除挂载了 emptyDir 卷的 Pod 和不受 controller 管理的 Pod。详情请参见节点排水规则。

步骤 4 单击“确定”，等待完成节点排水。

5.10 同步云服务器

操作场景

集群中的每一个节点对应一台云主机，集群节点创建成功后，您仍可以根据需求，修改云主机的名称或变更规格。

CCE 节点的部分信息是独立于弹性云主机 ECS 维护的，当您在 ECS 控制台修改云主机的名称、弹性公网 IP，以及变更计费方式或变更规格后，需要通过“同步节点信息”功能将信息同步到 CCE 控制台相应节点中，同步后信息将保持一致。

ECS 常见信息修改如下：

- 修改节点名称请参见修改云主机名称。
- 当您购买的节点规格无法满足业务需要时，可变更节点规格，升级 vCPU、内存。

约束与限制

当用户节点指定了云主机名称作为 K8s 节点名称时，该云主机名称的修改将无法同步到 CCE 控制台。

操作步骤

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”。

步骤 3 单击节点后的“更多 > 同步云主机”。

图5-5 同步节点信息

同步完成后，页面右上角将会提示“同步云主机任务下发成功”。

----结束

5.11 删除节点

操作场景

在 CCE 集群中删除节点会将该节点以及节点内运行的业务都销毁，删除前请确认您的正常业务运行不受影响，请谨慎操作。

约束与限制

- 删除 CCE 集群时，ECS 节点也将一起删除，暂不支持删除 CCE 集群而保留 ECS 节点的需求。
- 对于包周期（包年/包月）预付费的 ECS 节点不能直接删除，请通过页面右上角的“费用中心-我的订单”，执行资源退订操作。

须知

对于 1.17.11 及之后版本的集群，在 ECS 页面退订或删除虚机后，集群中对应的 node 节点也会自动删除。

- 删除节点会导致与节点关联的本地持久存储卷类型的 PVC/PV 数据丢失，无法恢复，且 PVC/PV 无法再正常使用。删除节点时使用了本地持久存储卷的 Pod 会从删除的节点上驱逐，并重新创建 Pod，Pod 会一直处于 pending 状态，因为 Pod 使用的 PVC 带有节点标签，由于冲突无法调度成功。

注意事项

- 删除节点会涉及 Pod 迁移，可能会影响业务，请在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 操作过程中，后台会把当前节点设置为不可调度状态。
- 删除节点仅能移除工作节点，不会移除控制节点。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏选择“节点管理”，在右侧单击节点后的“更多 > 删除”。

步骤 3 在弹出的“删除节点”窗口中，单击“是”，等待完成节点删除。

说明

- 删除节点后，原有节点上的工作负载实例会自动迁移至其他可用节点。
- 节点上绑定的磁盘和 EIP 如果属于重要资源请先解绑，否则会被级联删除。
- 释放 ECS 实例仅释放“按需计费”的 ECS 实例。

----结束

5.12 节点关机

操作场景

集群中的节点关机后，该节点以及节点内的业务将停止运行，节点关机前，请先确认您的正常业务运行将不受影响，请谨慎操作。

大部分节点关机后不再收费，特殊实例（包含本地硬盘，如磁盘增强型，超高 I/O 型等）关机后仍然正常收费，具体请参见 ECS 计费模式。

约束与限制

- 节点关机会涉及 Pod 迁移，可能会影响业务，请在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 操作过程中，后台会把当前节点设置为不可调度状态。
- 节点关机仅能对工作节点关机，不会对控制节点关机。

操作方法

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏选择“节点管理”，在右侧单击待关机节点的名称。

步骤 3 页面跳转至弹性云主机详情页中，单击右上角的“更多 > 实例状态 > 关机”，在弹出的关机窗口中单击“是”，即可完成关机操作。

图5-6 弹性云主机详情页

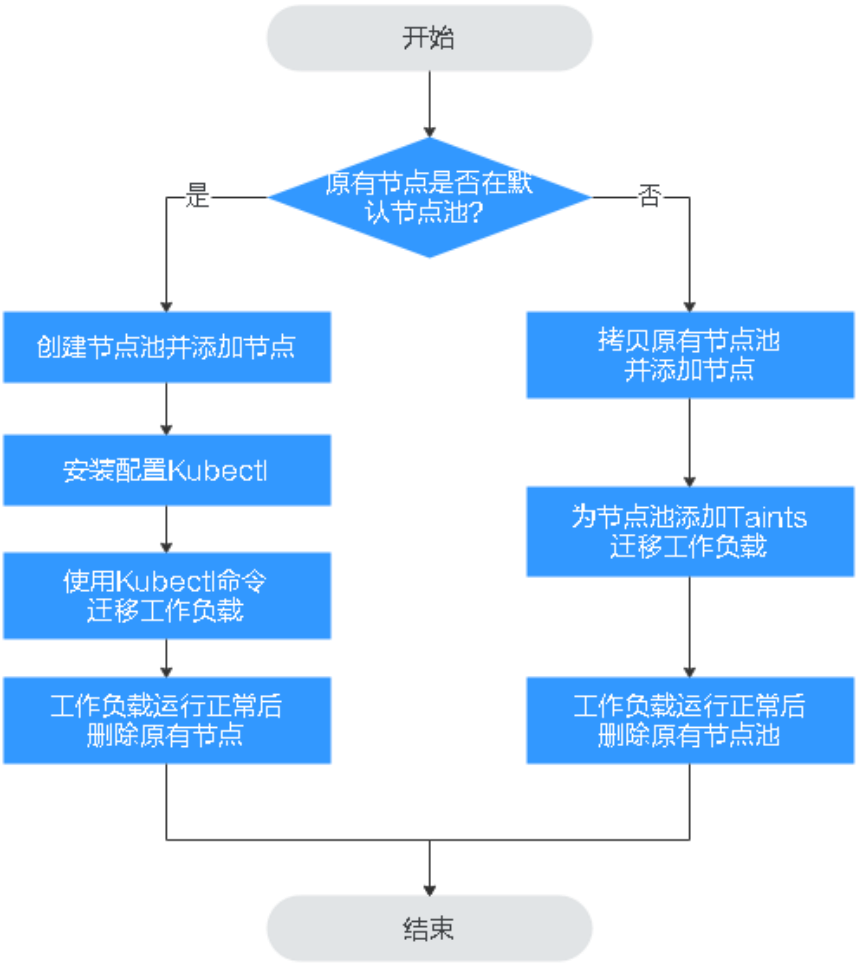
----结束

5.13 节点滚动升级

操作场景

节点滚动升级就是先创建新节点，然后将工作负载迁移到新的节点上，再删除老的节点。迁移流程如下图所示：

图5-7 节点迁移流程



约束与限制

- 现有节点和工作负载待迁移的节点必须在同一集群。
- 当前仅支持在 Kubernetes v1.13.10 及以后集群版本执行此操作。
- 默认节点池 DefaultPool 不支持修改配置。

原有节点在默认节点池

步骤 1 创建新的节点池。具体请参见创建节点池。

步骤 2 单击节点池名称，在节点列表中可查看到新建节点的 IP 地址。

步骤 3 安装配置 kubectl。具体请参见通过 kubectl 连接集群。

步骤 4 迁移工作负载。

1. 给需要迁移工作负载的节点打上 Taint（污点）。

   kubectl taint node *[node]* **key=value:***[effect]*

   其中，*[node]*为待迁移工作负载所在节点的 IP；*[effect]*取值为 NoSchedule、PreferNoSchedule 或 NoExecute，此处必须设置为 NoSchedule。

   - NoSchedule：一定不能被调度。
   - PreferNoSchedule：尽量不要调度。
   - NoExecute：不仅不会调度，还会驱逐 Node 上已有的 Pod。

   说明

   若需要重新设置污点时，可执行 **kubectl taint node** *[node]* **key:[***effect]*-命令去除污点。

2. 安全驱逐节点上的工作负载。

   kubectl drain *[node]*

   其中，*[node]*为待转移工作负载所在节点的 IP。

3. 在左侧导航栏中选择“工作负载 > 无状态负载 Deployment”。在工作负载列表中，待迁移工作负载的状态由“运行中”变为“未就绪”。工作负载状态再次变为“运行中”，表示迁移成功。

   说明

   迁移工作负载时，若工作负载配置了节点亲和性，则工作负载会一直提示“未就绪”等异常情况。请单击工作负载名称进入到负载详情页，在选择“调度策略”页签，删除原节点的亲和性配置，配置新的节点亲和性和反亲和性策略，详情请参见调度策略（亲和与反亲和）。

工作负载迁移成功后，在工作负载详情页的“实例列表”页签，可查看到工作负载状已迁移到步骤 1 中所创建的节点上。

步骤 5 删除原有节点。

工作负载迁移成功且运行正常后，即可删除原有节点。

----结束

原有节点不在默认节点池

步骤 1 拷贝节点池并添加节点。具体请参见拷贝节点池。

步骤 2 单击节点池名称操作列的“节点列表”，在节点列表中可查看到新建节点的 IP 地址。

步骤 3 迁移工作负载。

1. 单击原节点池后的“编辑”配置 Taints 参数。
2. 输入污点(Taints)的 Key 和 Value 值，Effect 选项有 NoSchedule、PreferNoSchedule 或 NoExecute，此处必须选择“NoExecute”，单击“添加”。
   - NoSchedule：一定不能被调度。
   - PreferNoSchedule：尽量不要调度。

- NoExecute：不仅不会调度，还会驱逐 Node 上已有的 Pod。

说明

若需要重新设置污点，需删除已配置污点。

3. 单击“确定”。
4. 在左侧导航栏中选择“工作负载 > 无状态负载 Deployment”。在工作负载列表中，待迁移工作负载的状态由“运行中”变为“未就绪”。工作负载状态再次变为“运行中”，表示迁移成功。

说明

迁移工作负载时，若工作负载配置了节点亲和性，则工作负载会一直提示“未就绪”等异常情况。请单击工作负载名称进入到负载详情页，在选择“调度策略”页签，删除原节点的亲和性配置，配置新的节点亲和性和反亲和性策略，详情请参见调度策略（亲和与反亲和）。

工作负载迁移成功后，在工作负载详情页的“Pods”页签，可查看到工作负载状已迁移到步骤 1 中所创建的节点上。

步骤 4 删除原有节点。

工作负载迁移成功且运行正常后，即可删除原有节点。

----结束

5.14 将节点容器引擎从 Docker 迁移到 Containerd

背景介绍

Kubernetes 在 1.24 版本中移除了 Dockershim，并从此不再默认支持 Docker 容器引擎。CCE 1.25 集群中仍将继续维护 Docker 容器引擎，并计划在 1.27 版本中移除对 Docker 容器引擎的支持。如果您需要将容器引擎为 Docker 的节点迁移至 Containerd 节点，请参考本文。

前提条件

- 已创建至少一个集群，并且该集群支持 Containerd 节点，详情请参见节点操作系统与容器引擎对应关系。
- 您的集群中存在容器引擎为 Docker 的节点或节点池。

注意事项

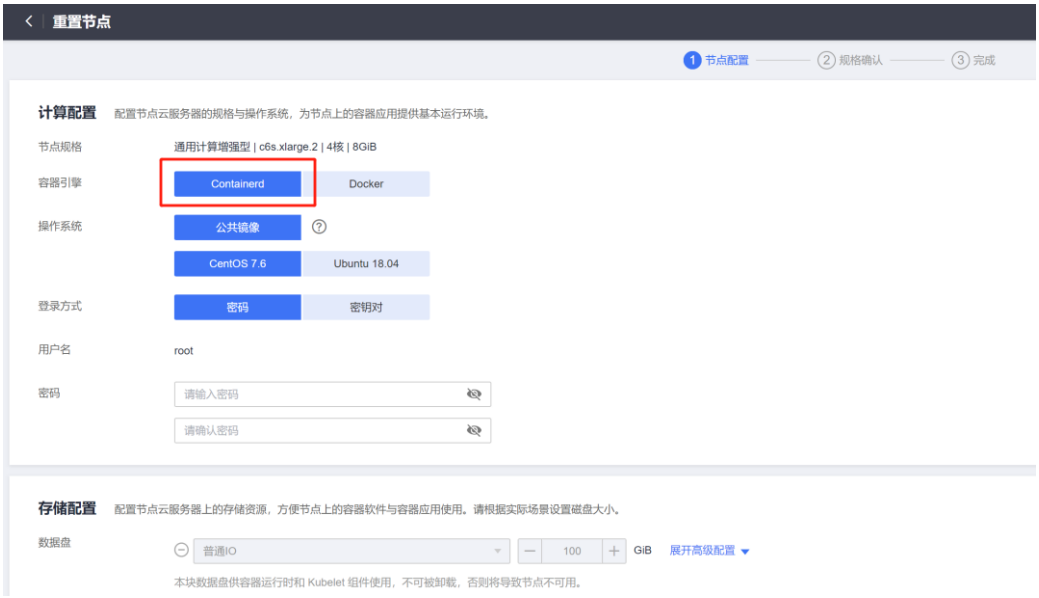
- 理论上节点容器运行时的迁移会导致业务短暂中断，因此强烈建议您迁移的业务保证多实例高可用部署，并且建议先在测试环境试验迁移的影响，以最大限度避免可能存在的风险。
- Containerd 不具备镜像构建功能，请勿在 Containerd 节点上使用 Docker Build 功能构建镜像。Docker 和 Containerd 其他差异请参考 容器引擎。

节点迁移步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧选择“节点管理”，并在节点列表中选择一个或多个需要重置的节点，单击“更多 > 重置节点”。

步骤 3 在容器引擎中选择 Containerd，其余参数可根据需要进行调整，也可以和创建时保持一致。



步骤 4 当节点状态显示为安装中时，即表示正在重置节点。

待节点状态显示为运行中时，您即可检查节点容器运行时是否切换成功，页面中可以看到节点运行时版本已经切换为 Containerd，并且登录节点可以执行 **crictl** 等 Containerd 相关命令查看节点上运行的容器信息。

----结束

节点池迁移步骤

您可使用节点池拷贝功能，拷贝原有的 Docker 节点池，并将新节点池的容器引擎选择为 Containerd，其余配置和原 Docker 节点池保持一致。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧选择“节点管理”，切换至“节点池”页签，并在需要拷贝的 Docker 节点池“操作”栏中，单击“更多 > 拷贝”。

步骤 3 在节点池配置页面中，选择容器引擎为 Containerd，其余参数可根据需要进行调整，并完成节点池创建。

步骤 4 将创建完的 Containerd 节点池扩容至原 Docker 节点池的数量，并逐个删除 Docker 节点池中的节点。

推荐使用滚动的方式迁移，即扩容部分 Containerd 节点，再删除部分 Docker 节点，直至新的 Containerd 节点池中节点数量和原 Docker 节点池中节点数量一致。

说明

若您在原有 Docker 节点或节点池上部署的负载设置了对应的节点亲和性，则需要将负载的节点亲和性策略配置为的新 Containerd 节点或节点池。

步骤 5 迁移完成后，删除原有 Docker 节点池。

----结束

6 节点池管理

6.1 节点池概述

简介

为帮助您更好地管理 Kubernetes 集群内的节点，云容器引擎 CCE 引入节点池概念。节点池是集群中具有相同配置的一组节点，一个节点池包含一个节点或多个节点。

您可以在 CCE 控制台创建新的自定义节点池，借助节点池基本功能方便快捷地创建、管理和销毁节点，而不会影响整个集群。新节点池中所有节点的参数和类型都彼此相同，您无法在节点池中配置单个节点，任何配置更改都会影响节点池中的所有节点。

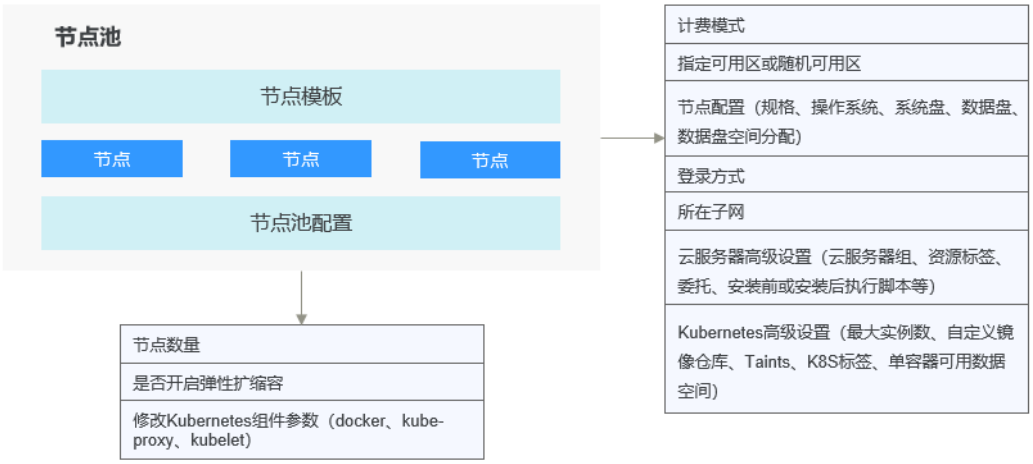
通过节点池功能您还可以实现节点的动态扩缩容（仅按需计费的节点池支持）：

- 当集群中出现因资源不足而无法调度的实例（Pod）时，自动触发扩容，为您减少人力成本。
- 当满足节点空闲等缩容条件时，自动触发缩容，为您节约资源成本。

本章节介绍节点池在云容器引擎（CCE）中的工作原理，以及如何创建和管理节点池。

节点池架构

图6-1 节点池整体架构图



通常情况下，节点池内的节点均具有如下相同属性：

- 节点操作系统。
- 节点规格。
- 节点登录方式。
- 节点容器运行时。
- 节点 Kubernetes 组件启动参数。
- 节点自定义启动脚本。
- 节点“K8S 标签”及“Taints”设置。

此外，CCE 将同时围绕节点池扩展以下属性：

- 节点池级别操作系统。
- 节点池级别每节点的 Pod 数上限。

默认节点池 DefaultPool 说明

DefaultPool 并非是一个真正的节点池，只是将非自定义节点池中的节点做一个**归类**，所有非自定义节点池中创建的节点（直接在控制台创建的节点或调用 API 创建的节点）都会归类在 DefaultPool 中。DefaultPool 不具备任何自定义节点池的功能，包括弹性伸缩、各项参数设置等，且不可编辑、删除或迁移，也不支持扩容、弹性伸缩。

应用场景

当业务需要使用大规模集群时，推荐您使用节点池进行节点管理，以提高大规模集群易用性。

下表介绍了多种大规模集群管理场景，并分别展示节点池在每种场景下发挥的作用：

表6-1 节点池场景及作用

场景	作用
集群存在较多异构节点（机型配置不同）	通过节点池可规范节点分组管理。
集群需要频繁扩缩容节点	通过节点池可降低操作成本。
集群内应用程序调度规则复杂	通过节点池标签可快速指定业务调度规则。

功能点及注意事项

功能点	功能说明	注意事项
创建节点池	新增节点池。	单个集群不建议超过 100 个节点池。
删除节点池	删除节点池时会先删除节点池中的节点，原有节点上的工作负载实例会自动迁移至其他节点池的可用节点。	如果工作负载实例具有特定的节点选择器，且如果集群中的其他节点均不符合标准，则工作负载实例可能仍处于无法安排的状态。
节点池开启弹性伸缩	开启弹性伸缩后，节点池将根据集群负载情况自动创建或删除节点池内的节点。	节点池中的节点建议不要放置重要数据，以防止节点被弹性缩容，数据无法恢复。
节点池关闭弹性伸缩	关闭弹性伸缩后，节点池内节点数量不随集群负载情况自动调整。	/
调整节点池大小	支持直接调整节点池内节点个数。若减小节点数量，将从现有节点池内随机缩容节点。	开启弹性伸缩后，不建议手动调整节点池大小。
调整节点池配置	可修改节点池名称、节点个数，删除或新增 K8S 标签、Taints 及资源标签。	删除或新增 K8S 标签和 Taints 会对节点池内节点全部生效，可能会引起 Pod 重新调度，请谨慎变更。
移出节点池内节点	可以将同一个集群下某个节点池中的节点迁移到默认节点池（defaultpool）中	暂不支持将默认节点池（defaultpool）中的节点迁移到其他节点池中，也不支持将自定义节点池中的节点迁移到其他自定义节点池。
拷贝节点池	可以方便的拷贝现有节点池的配置，从而创建新的节点池。	/
配置 kubernetes	通过该功能您可以对核心组件	• 本功能仅支持在 v1.15 及以上版本的集群中对节点池进

功能点	功能说明	注意事项
参数	进行深度配置。	行配置，V1.15 以下版本不显示该功能。 • 默认节点池 DefaultPool 不支持修改该类配置。

将工作负载部署到特定节点池

在定义工作负载时，您可以间接的控制将其部署在哪个节点池上。

例如，您可以通过 CCE 控制台工作负载页面的“调度策略”设置工作负载与节点的亲和性，强制将该工作负载部署到特定节点池上，从而实现该工作负载仅在该节点池中的节点上运行的目的。如果您需要更好地控制工作负载实例的调度位置，您可以使用调度策略（亲和与反亲和）章节中关于工作负载与节点的亲和或反亲和策略相关说明。

您也可以为容器指定资源请求，工作负载将仅在满足资源请求的节点上运行。

例如，如果工作负载定义了需要包含四个 CPU 的容器，则工作负载将不会选择在具有两个 CPU 的节点上运行。

6.2 创建节点池

操作场景

本章介绍了如何添加运行节点池以及对节点池执行操作。要了解节点池的工作原理，请参阅节点池概述。

约束与限制

- 节点弹性伸缩功能需要安装 autoscaler 插件。
- 仅按需计费支持节点弹性伸缩，包年/包月计费模式不支持弹性伸缩。
- 仅 1.19 及以上版本集群支持自定义安全组。

操作步骤

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。

步骤 3 单击右上角“创建节点池”。

基础配置

表6-2 基础配置

参数	参数说明
节点池名称	新建节点池的名称，默认按“集群名-nodepool-随机数”生成名称，可自定义。
节点数量	创建节点池时，创建节点的数量。
弹性伸缩	默认不开启。包年/包月不支持弹性伸缩，仅按需支持。 开启弹性扩缩容功能要求安装 AutoScaler 插件。 单击  开启后，节点池将根据集群负载情况自动创建或删除节点池内的节点，参数设置如下： • 节点数上限和节点数下限：您可设置节点数的上限和下限，保证节点数在合理的范围内伸缩。 • 节点池优先级：请根据业务需要设置相应数值，该数值表示节点池之间进行**扩容的优先级**，数值越大优先级越高，如设置为 4 的节点池比设置为 1 的节点池优先启动扩容。若多个节点池的值设置相同，如都设置为 2，表示这几个节点池之间不分优先级，系统将按最小资源浪费原则进行扩容。 **说明** 弹性扩容时 CCE 将按照如下策略来选择节点池进行扩容： 1. 通过预判算法判断节点池是否能满足让 Pending 的 Pod 正常调度的条件，包括节点资源大于 Pod 的 request 值、nodeSelect、nodeAffinity 和 taints 等是否满足 Pod 正常调度的条件；另外还会过滤掉扩容失败（因为资源不足等原因）还处于 15min 冷却时间的节点池。 2. 有多个节点池满足条件时，判断节点池设置的优先级（优先级默认值为 0，取值范围为 0-100，其中 100 为最高，0 为最低），选择优先级最高的节点池扩容。 3. 如果有多个节点池处于相同的优先级，或者都没有配置优先级时，通过最小浪费原则，根据节点池里设置的虚拟机规格，计算刚好能满足 Pending 的 Pod 正常调度，且浪费资源最少的节点池。 4. 如果还是有多个节点池的虚拟机规格都一样，只是 AZ 不同，那么会随机选择其中一个节点池触发扩容。 • 弹性缩容冷却时间：请设置时间，单位为分钟或小时。弹性缩容冷却时间是指当前节点池扩容出的节点多长时间不能被缩容。 节点池中配置的缩容冷却时间和 autoscaler 插件中配置的缩容冷却时间之间的影响和关系如下： **节点池配置的缩容冷却时间** 弹性缩容冷却时间：当前节点池扩容出的节点多长时间不能被缩容，作用范围为节点池级别。 **autoscaler 插件配置的缩容冷却时间** 扩容后缩容冷却时间：autoscaler 触发扩容后（不可调度、指标、周期策略等场景）整个集群多长时间内不能被缩容，作用范围为集群级别。

参数	参数说明
	节点删除后缩容冷却时间：autoscaler 触发缩容后整个集群多长时间内不能继续缩容，作用范围为集群级别。 缩容失败后缩容冷却时间：autoscaler 触发缩容失败后整个集群多长时间内不能继续缩容，作用范围为集群级别。 **说明** 节点池中的节点建议不要放置重要数据，以防止节点被弹性缩容，数据无法恢复。

计算配置：

配置节点云主机的规格与操作系统，为节点上的容器应用提供基本运行环境。

表6-3 计算配置参数

参数	参数说明
计费模式	支持如下两种计费方式。 • 包年包月 包年包月需要选择购买时长。 • 按需计费
可用区	节点云主机所在的可用区，集群下节点创建在不同可用区下可以提高可靠性。创建后不可修改。 建议您选择“随机分配”，可根据选择的节点规格随机分配一个可以使用的可用区。 可用区是在同一区域下，电力、网络隔离的物理区域，可用区之间内网互通，不同可用区之间物理隔离。如果您需要提高工作负载的高可靠性，建议您将云主机创建在不同的可用区。
节点类型	CCE 集群支持弹性云主机 ECS 和物理机。
容器引擎	CCE 集群支持 Docker，1.23 起 CentOS、Ubuntu 支持 Containerd。
节点规格	请根据业务需求选择相应的节点规格。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。具体请参见支持的节点规格。 公共镜像：请选择节点对应的操作系统。
登录方式	**密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。

参数	参数说明
	密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建。

存储配置：

配置节点云主机上的存储资源，方便节点上的容器软件与容器应用使用。请根据实际场景设置磁盘大小。

表6-4 存储配置参数

参数	参数说明
系统盘	节点云主机使用的系统盘，供操作系统使用。您可以设置系统盘的规格为 40GB-1024GB 之间的数值，缺省值为 50GB。 加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**目前仅在部分 Region 显示此选项，具体以界面为准。** • 默认不加密。 • 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。
数据盘	节点云主机使用的数据盘，供容器运行时和 Kubelet 组件使用。您可以设置数据盘的规格为 100GB-32768GB 之间的数值，缺省值为 100GB。 **至少需要一块数据盘**，供容器运行时和 Kubelet 组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** 单击后方的“展开高级设置”可进行如下设置： • 自定义空间分配：勾选后可定义容器运行时在数据盘上占用的空间比例，容器运行时的空间用于存放容器运行时工作目录、容器镜像数据以及镜像元数据。数据盘空间分配详细说明请参见数据盘空间分配说明。 • 加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**目前仅在部分 Region 显示此选项，具体以界面为准。** – 默认不加密。 – 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 **添加多个数据盘** 最多可以添加 4 个，默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，选择如下配置。 • 默认：默认情况直接创建为裸盘，不做任何处理。

参数	参数说明
	• 挂载到指定目录：将数据盘挂载到指定目录。 • 作为持久存储卷：适用于对 PV 有性能要求的场景。持久存储卷的节点会添加上 node.kubernetes.io/local-storage-persistent 标签，取值为 linear 或 striped。 • 作为临时存储卷：适用于对 EmptyDir 有性能要求的场景。 说明 持久存储卷和临时存储卷仅在集群版本 >= v1.21.2-r0 时支持创建，且临时存储卷需要 Everest 插件版本>=1.2.29，持久存储卷需要 Everest 插件版本>=1.2.31。 持久存储卷和临时存储卷支持如下两种写入模式。 • 线性（linear）：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。 • 条带化（striped）：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。多块存储卷才能选择条带化。 **本地盘说明** 节点规格为“磁盘增强型”或“超高 I/O 型”时，有一块数据盘可以是本地盘。 本地磁盘实例有宕机风险，不保证数据可靠性，建议您使用云硬盘存储您的业务数据。

网络配置：

配置节点云主机的网络资源，用于访问节点和容器应用。

表6-5 网络配置参数

参数	参数说明
节点子网	节点子网默认使用创建集群时的子网配置，也可以选择其他子网。
关联安全组	指定节点池创建出来的节点使用哪个安全组。最多选择 5 个安全组。 创建集群时会默认创建一个节点安全组，名称为**{集群名}-cce-node-{随机 ID}**，默认会使用该安全组。 节点安全组需要放通一些端口以保障节点通信，如选择其他安全组，需要放通这些端口。

高级配置：

节点能力增强，可在此配置节点的标签、污点、启动命令等功能。

表6-6 高级配置参数

参数	参数说明
K8S 标签	单击“添加标签”可以设置附加到 Kubernetes 对象（比如 Pods）上的键值对，最多可以添加 10 条标签 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器 Pod 调度到指定节点的功能。详细请参见 Labels and Selectors。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 CCE 服务会自动帮您创建 CCE-Dynamic-Provisioning-Node=节点 id 的标签。
污点(Taints)	默认为空。支持给节点加 Taints 来设置反亲和性，每个节点最多配置 10 条 Taints，每条 Taints 包含以下 3 个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符；另外可以使用 DNS 子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符。 • Effect：只可选 NoSchedule，PreferNoSchedule 或 NoExecute。 Taints 的使用请参见管理节点污点（taint）。 说明 对于 1.19 及以下版本集群，有可能会出现污点打上之前负载已经调度到节点上，如果需要避免这种情况，请选择 1.19 及以上集群。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为 16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。 节点最多能创建多少个 Pod 还受其他因素影响，具体请参见节点最多可以创建多少个 Pod。
云主机组	云主机组是对云主机的一种逻辑划分，同一云主机组中的云主机遵从同一策略。 反亲和性策略：同一云主机组中的云主机分散地创建在不同主机上，提高业务的可靠性。 选择已创建的云主机组，或单击“新建云主机组”创建，创建完成后单击刷新按钮。
安装前执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装前执行，可能导致 Kubernetes 软件无法正常安装，需谨慎使用。

参数	参数说明
安装后执行脚本	请输入脚本命令，大小限制为 0~1000 字符。 脚本将在 Kubernetes 软件安装后执行，不影响 Kubernetes 软件安装。
委托	委托是由租户管理员在统一身份认证服务上创建的。通过委托，可以将云主机资源共享给其他帐号，或委托更专业的人或团队来代为管理。 如果没有委托请单击右侧“新建委托”创建。

步骤 4 单击“下一步：规格确认”，确认已阅读并知晓服务协议。

步骤 5 单击“提交”。

----结束

6.3 管理节点池

约束与限制

默认节点池 DefaultPool 不支持如下管理操作。

配置管理

为方便对 CCE 集群中的 kubernetes 配置参数进行管理，CCE 提供了配置管理功能，通过该功能您可以对核心组件进行深度配置，更多信息请参见 kubelet。

仅支持在 **v1.15 及以上版本**的集群中对节点池进行配置，V1.15 以下版本不显示该功能。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。

步骤 3 单击节点池名称后的“更多 > 配置管理”。

步骤 4 在侧边栏滑出的“配置管理”窗口中，根据业务需求修改 Kubernetes 的参数值：

表6-7 配置管理参数

组件	参数	详情	默认	修改限制
容器引擎 Docker 配置	native-umask	`--exec-opt native.umask	normal	不支持修改
	docker-base-size	`--storage-opts dm.basesize	0	不支持修改

组件	参数	详情	默认	修改限制
	insecure-registry	不安全的镜像源地址	false	不支持修改
	limitcore	核数限制，节点池中创建的节点总核数不能超过该值	5368709120	-
	default-ulimit-nofile	容器内句柄数限制	{soft}:{hard}	该值大小不可超过节点内核参数 nr_open 的值，且不能是负数。 节点内核参数 nr_open 可通过以下命令获取： `sysctl -a | grep nr_open`
kube-proxy 组件配置	conntrack-min	sysctl -w net.nf_conntrack_max	131072	支持在节点池生命周期中修改
	conntrack-tcp-timeout-close-wait	sysctl -w net.netfilter.nf_conntrack_tcp_timeout_close_wait	1h0m0s	
kubelet 组件配置	cpu-manager-policy	`--cpu-manager-policy	none	支持在节点池生命周期中修改
	kube-api-qps	与 kube-apiserver 通信的 qps	100	
	kube-api-burst	与 kube-apiserver 通信的 burst	100	
	max-pods	kubelet 管理的 pod 上限	40 20	
	pod-pids-limit	k8s 限制进程数	-1	
	with-local-dns	是否使用本地 IP 作为该节点的 ClusterDNS	false	
	event-qps	事件创建 QPS 限制	5	
	allowed-unsafe-	允许使用的不安全系统	[]	

组件	参数	详情	默认	修改限制
	sysctls	配置。 CCE 从 **1.17.17** 集群版本开始，kube-apiserver 开启了 pod 安全策略，需要在 pod 安全策略的 allowedUnsafeSysctls 中增加相应的配置才能生效（1.17.17 以下版本的集群可不配置）。		
	kube-reserved-mem system-reserved-mem	节点内存预留。	随节点规格变动，具体请参见节点预留资源计算公式	kube-reserved-mem，system-reserved-mem 之和小于内存的一半
	topology-manager-policy	设置拓扑管理策略。 合法值包括： • restricted：kubelet 仅接受在所请求资源上实现最佳 NUMA 对齐的 Pod。 • best-effort：kubelet 会优先选择在 CPU 和设备资源上实现 NUMA 对齐的 Pod。 • none（默认）：不启用拓扑管理策略。 • single-numa-node：kubelet 仅允许在 CPU 和设备资源上对齐到同一 NUMA 节点的 Pod。	none	支持在节点池生命周期中修改 须知 修改 topology-manager-policy 和 topology-manager-scope 会重启 kubelet，并且以更改后的策略重新计算容器实例的资源分配，这有可能导致已经运行的容器实例重启甚至无法进行资源分配。
	topology-manager-scope	设置拓扑管理策略的资源对齐粒度。合法值包括： • container （默认)：对齐粒度为容器级 • pod：对齐粒度为 pod 级	container	

组件	参数	详情	默认	修改限制
	over-subscription-resource	节点超卖特性。 仅**弹性云主机-物理机**类型的节点池可见，设置为 true 表示开启节点超卖特性。	-	-
	nic-minimum-target	节点池级别的节点最少绑定容器网卡数	默认：10	
	nic-maximum-target	节点池级别的节点预热容器网卡上限检查值	默认：0	
	nic-warm-target	节点池级别的节点动态预热容器网卡数	默认：2	
	nic-max-above-warm-target	节点池级别的节点预热容器网卡回收阈值	默认：2	
容器引擎 Containerd 配置 （仅使用 Containerd 的节点池可见）	devmapper-base-size	单容器可用数据空间	-	不支持修改
	limitcore	核数限制	5368709120	-
	default-ulimit-nofile	容器内句柄数限制	1048576	该值大小不可超过节点内核参数 nr_open 的值，且不能是负数。 节点内核参数 nr_open 可通过以下命令获取： `sysctl -a | grep nr_open`

步骤 5 单击“确定”，完成配置操作。

----结束

编辑节点池

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。

步骤 3 单击节点池名称后的“编辑”，在弹出的“编辑节点池”中，配置以下参数：

表6-8 配置节点池参数

参数	参数说明
节点池名称	自定义节点池名称。
节点数量	请根据业务需求调整节点个数。
弹性伸缩	默认不开启。 单击  开启后，节点池将根据业务需求自动创建或删除节点池内的节点，参数设置如下： • 节点数上限和节点数下限：您可设置节点数的上限和下限，保证节点数在合理的范围内伸缩。 • 优先级：该数值表示节点池之间进行弹性扩容的优先级，数值越大优先级越高，如设置为 4 的节点池比设置为 1 的节点池优先启动扩容。若多个节点池的值设置相同，如都设置为 2，表示这几个节点池之间不分优先级，系统将按最小资源浪费原则进行扩容。 • 弹性缩容冷却时间：请设置时间，单位为分钟。弹性缩容冷却时间是指当前节点池扩容出的节点多长时间不能被缩容。 为保证功能的正常使用，节点池开启弹性扩缩容功能后，请务必安装 AutoScaler 插件。
K8S 标签	单击“添加标签”可以设置附加到 Kubernetes 对象（比如 Pods）上的键值对，最多可以添加 10 条标签 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器 Pod 调度到指定节点的功能。详细请参见 Labels and Selectors。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 CCE 服务会自动帮您创建 CCE-Dynamic-Provisioning-Node=节点 id 的标签。
污点(Taints)	默认为空。支持给节点加 Taints 来设置反亲和性，每个节点最多配置 10 条 Taints，每条 Taints 包含以下 3 个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长 63 个字符；另外可以使用 DNS 子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、

参数	参数说明
	连字符、下划线和点，最长 63 个字符。 • Effect：只可选 NoSchedule，PreferNoSchedule 或 NoExecute。 Taints 的使用请参见管理节点污点（taint）。

步骤 4 配置完成后，单击“确定”。

在节点池列表中可查看到节点池状态为“伸缩中”，待状态变为“完成”，节点池参数编辑成功，修改的配置后将同步到节点池下的已有节点。

----结束

删除节点池

删除节点池，会先删除节点池中的节点，节点删除后，原有节点上的工作负载实例会自动迁移至其他节点池的可用节点。如果工作负载实例具有特定的节点选择器，且如果集群中的其他节点均不符合标准，则工作负载实例可能仍处于无法安排的状态。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。

步骤 3 单击节点池名称后的“更多 > 删除”。

步骤 4 在弹出的“删除节点池”窗口中，请仔细阅读界面提示。

步骤 5 确定要对节点池进行删除操作后，请在弹窗中单击“是”，即可完成节点池的删除。

----结束

拷贝节点池

通过 CCE 控制台可以方便的拷贝现有节点池的配置，从而创建新的节点池。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。

步骤 3 单击节点池名称后的“更多 > 拷贝”。

步骤 4 在弹出的“拷贝节点池”窗口中，可以看到拷贝的节点池配置，您可以根据需要进行修改，确定配置后单击“下一步：规格确认”。

步骤 5 在“规格确认”步骤中再次确认规格并单击“提交”，即可完成节点池的拷贝并创建新的节点池。

----结束

迁移节点

您可以将同一个集群下某个节点池中的节点迁移到默认节点池（defaultpool）中，暂不支持将默认节点池（defaultpool）中的节点迁移到其他节点池中，也不支持将自定义节点池中的节点迁移到其他自定义节点池。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“节点管理”，并切换至“节点池”页签。

步骤 3 单击待迁移的节点池名称后的“节点列表”。

步骤 4 勾选需要迁移的节点，单击“更多 > 迁移”，将节点批量迁移至默认节点池。

您也可以在单个节点的“操作”栏中，单击“更多 > 迁移”，迁移单个节点。

步骤 5 在弹出的“迁移节点”窗口中进行确认。

说明

迁移完成后，节点原有资源标签、K8S 标签、Taints 不受影响。

----结束

7 工作负载

7.1 工作负载概述

工作负载是在 Kubernetes 上运行的应用程序。无论您的工作负载是单个组件还是协同工作的多个组件，您都可以在 Kubernetes 上的一组 Pod 中运行它。在 Kubernetes 中，工作负载是对一组 Pod 的抽象模型，用于描述业务的运行载体，包括 Deployment、Statefulset、Daemonset、Job、CronJob 等多种类型。

云容器引擎 CCE 提供基于 Kubernetes 原生类型的容器部署和管理能力，支持容器工作负载部署、配置、监控、扩容、升级、卸载、服务发现及负载均衡等生命周期管理。

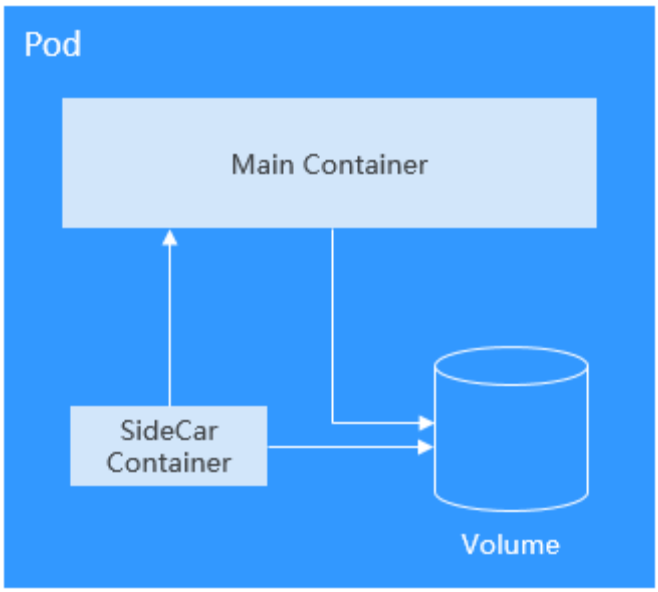
Pod

Pod 是 Kubernetes 创建或部署的最小单位。一个 Pod 封装一个或多个容器（container）、存储资源（volume）、一个独立的网络 IP 以及管理控制容器运行方式的策略选项。

Pod 使用主要分为两种方式：

- Pod 中运行一个容器。这是 Kubernetes 最常见的用法，您可以将 Pod 视为单个封装的容器，但是 Kubernetes 是直接管理 Pod 而不是容器。
- Pod 中运行多个需要耦合在一起工作、需要共享资源的容器。通常这种场景下应用包含一个主容器和几个辅助容器（SideCar Container），例如主容器为一个 web 服务器，从一个固定目录下对外提供文件服务，而辅助容器周期性的从外部下载文件存到这个固定目录下。

图7-1 Pod

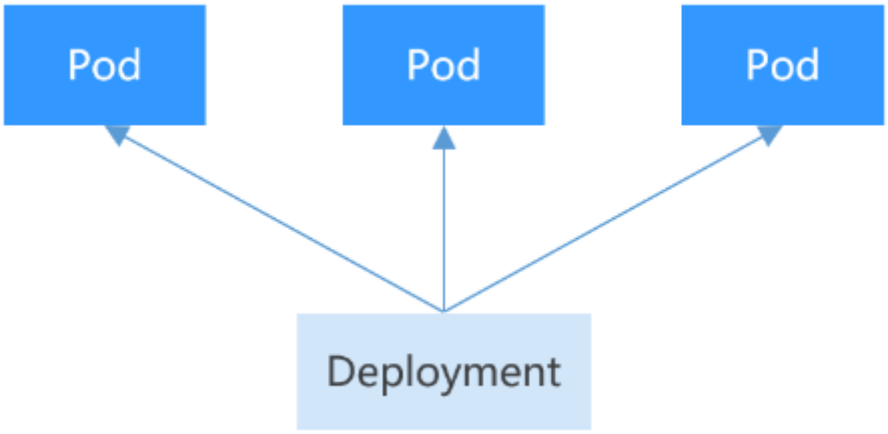


实际使用中很少直接创建 Pod，而是使用 Kubernetes 中称为 Controller 的抽象层来管理 Pod 实例，例如 Deployment 和 Job。Controller 可以创建和管理多个 Pod，提供副本管理、滚动升级和自愈能力。通常，Controller 会使用 Pod Template 来创建相应的 Pod。

Deployment

Pod 是 Kubernetes 创建或部署的最小单位，但是 Pod 是被设计为相对短暂的一次性实体，Pod 可以被驱逐（当节点资源不足时）、随着集群的节点崩溃而消失。Kubernetes 提供了 Controller（控制器）来管理 Pod，Controller 可以创建和管理多个 Pod，提供副本管理、滚动升级和自愈能力，其中最为常用的就是 Deployment。

图7-2 Deployment



一个 Deployment 可以包含一个或多个 Pod 副本，每个 Pod 副本的角色相同，所以系统会自动为 Deployment 的多个 Pod 副本分发请求。

Deployment 集成了上线部署、滚动升级、创建副本、恢复上线的功能，在某种程度上，Deployment 实现无人值守的上线，大大降低了上线过程的复杂性和操作风险。

StatefulSet

Deployment 控制器下的 Pod 都有个共同特点，那就是每个 Pod 除了名称和 IP 地址不同，其余完全相同。需要的时候，Deployment 可以通过 Pod 模板创建新的 Pod；不需要的时候，Deployment 就可以删除任意一个 Pod。

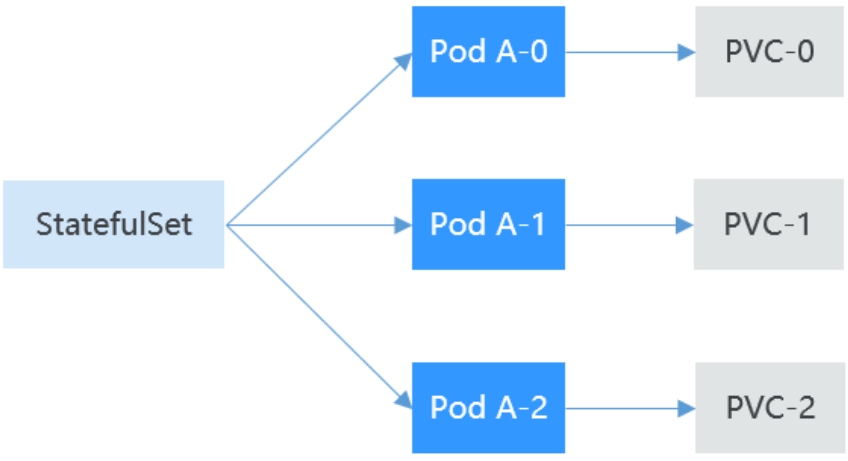
但是在某些场景下，这并不满足需求，比如有些分布式的场景，要求每个 Pod 都有自己单独的状态时，比如分布式数据库，每个 Pod 要求有单独的存储，这时 Deployment 就不能满足需求了。

详细分析下有状态应用的需求，分布式有状态的特点主要是应用中每个部分的角色不同（即分工不同），比如数据库有主备，Pod 之间有依赖，对应到 Kubernetes 中就是对 Pod 有如下要求：

- Pod 能够被别的 Pod 找到，这就要求 Pod 有固定的标识。
- 每个 Pod 有单独存储，Pod 被删除恢复后，读取的数据必须还是以前那份，否则状态就会不一致。

Kubernetes 提供了 StatefulSet 来解决这个问题，其具体如下：

1. StatefulSet 给每个 Pod 提供固定名称，Pod 名称增加从 0-N 的固定后缀，Pod 重新调度后 Pod 名称和 HostName 不变。
2. StatefulSet 通过 Headless Service 给每个 Pod 提供固定的访问域名，Service 的概念会在后面章节中详细介绍。
3. StatefulSet 通过创建固定标识的 PVC 保证 Pod 重新调度后还是能访问到相同的持久化数据。



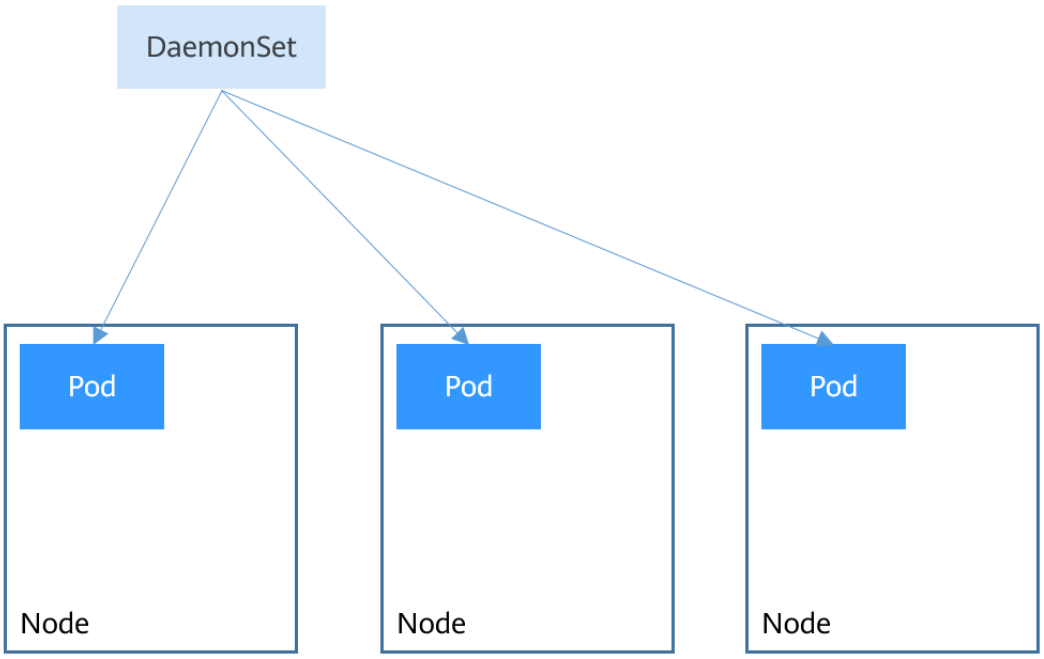
DaemonSet

DaemonSet 是这样一种对象（守护进程），它在集群的每个节点上运行一个 Pod，且保证只有一个 Pod，这非常适合一些系统层面的应用，例如日志收集、资源监控等，这

类应用需要每个节点都运行，且不需要太多实例，一个比较好的例子就是 Kubernetes 的 kube-proxy。

DaemonSet 跟节点相关，如果节点异常，也不会在其他节点重新创建。

图7-3 DaemonSet



Job 和 CronJob

Job 和 CronJob 是负责批量处理短暂的一次性任务（short lived one-off tasks），即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束。

- Job：是 Kubernetes 用来控制批处理型任务的资源对象。批处理业务与长期伺服业务（Deployment、Statefulset）的主要区别是批处理业务的运行有头有尾，而长期伺服业务在用户不停止的情况下永远运行。Job 管理的 Pod 根据用户的设置把任务成功完成就自动退出（Pod 自动删除）。
- CronJob：是基于时间的 Job，就类似于 Linux 系统的 crontab 文件中的一行，在指定的时间周期运行指定的 Job。

任务负载的这种用完即停止的特性特别适合一次性任务，比如持续集成。

工作负载生命周期说明

表7-1 状态说明

状态	说明
运行中	所有实例都处于运行中才是运行中。
未就绪	容器处于异常、实例数为 0 或 pending 状态时显示此状态。

状态	说明
升级/回滚中	触发升级或回滚动作后，工作负载会处于升级/回滚中。
可用	当多实例无状态工作负载运行过程中部分实例异常，可用实例不为 0，工作负载会处于可用状态。
执行完成	任务执行完成，仅普通任务存在该状态。
已停止	触发停止操作后，工作负载会处于停止状态，实例数变为 0。v1.13 之前的版本存在此状态。
删除中	触发删除操作后，工作负载会处于删除中状态。
暂停中	触发暂停操作后，工作负载会处于暂停中状态。

7.2 创建无状态负载(Deployment)

操作场景

在运行中始终不保存任何数据或状态的工作负载称为“无状态负载 Deployment”，例如 nginx。您可以通过控制台或 kubectl 命令行创建无状态负载。

前提条件

- 在创建容器工作负载前，您需要存在一个可用集群。若没有可用集群 ，请参照购买 CCE 集群中内容创建。
- 若工作负载需要被外网访问，请确保集群中至少有一个节点已绑定弹性 IP，或已创建负载均衡实例。

说明

单个实例（Pod）内如果有多个容器，请确保容器使用的端口不冲突 ，否则部署会失败。

通过控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 3 配置工作负载的信息。

基本信息

- 负载类型：选择无状态工作负载 Deployment。工作负载类型的介绍请参见工作负载概述。
- 负载名称：填写工作负载的名称。
- 命名空间：选择工作负载的命名空间，默认为 default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见创建命名空间。

- 实例数量：填写实例的数量，也就是 Pod 的数量。
- 容器运行时：CCE 集群默认使用普通运行时。
- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见时区同步。

容器配置

- 容器信息

  Pod 中可以配置多个容器，您可以单击右侧“添加容器”为 Pod 配置多个容器。

  - 基本信息：容器基本信息
  - 生命周期：设置容器生命周期
  - 健康检查：设置容器健康检查
  - 环境变量：设置环境变量
  - 数据存储：存储概述

  说明

  负载实例数大于 1 时，不支持挂载云硬盘类型的存储。

  - 安全设置：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户 ID，容器将以当前用户权限运行。
  - 容器日志：使用 ICAgent 采集容器日志
- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为 default-secret，使用 default-secret 可访问 SWR 镜像仓库的镜像。default-secret 详细说明请参见 default-secret。
- GPU 显卡：默认为不限制。当集群中存在 GPU 节点时，工作负载实例可以调度到指定 GPU 显卡类型的节点上。

服务配置

服务（Service）是用来解决 Pod 访问问题的。每个 Service 有一个固定 IP 地址，Service 将访问流量转发给 Pod，而且 Service 可以给这些 Pod 做负载均衡。

您也可以在创建完工作负载之后再创建 Service，Service 的概念和使用方法请参见 Service 概述。

高级配置

- 升级策略：工作负载升级配置
- 调度策略：调度策略（亲和与反亲和）
- 容忍策略：容忍策略与节点的污点能力配合使用，允许（不强制）负载调度到带有与之匹配的污点的节点上，也可用于控制负载所在的节点被标记污点后负载的驱逐策略，详情请参见容忍度（Toleration）。

步骤 4 单击右下角“创建工作负载”。

----结束

7.3 创建有状态负载(StatefulSet)

操作场景

在运行过程中会保存数据或状态的工作负载称为“有状态工作负载（statefulset）”。例如 Mysql，它需要存储产生的新数据。

因为容器可以在不同主机间迁移，所以在宿主机上并不会保存数据，这依赖于 CCE 提供的高可用存储卷，将存储卷挂载在容器上，从而实现有状态工作负载的数据持久化。

约束与限制

- 当您删除或扩缩有状态负载时，为保证数据安全，系统并不会删除它所关联的存储卷。
- 当您删除一个有状态负载时，为实现有状态负载中的 Pod 可以有序停止，请在删除之前将副本数缩容到 0。
- 您需要在创建有状态负载的同时，创建一个 Headless Service，用于解决有状态负载 Pod 互相访问的问题，详情请参见 Headless Service。
- 节点不可用时，Pod 状态变为“未就绪”，此时需要手工删除有状态工作负载的 Pod，Pod 实例才会迁移到正常节点上。

前提条件

- 在创建容器工作负载前，您需要存在一个可用集群。若没有请参照购买 CCE 集群中内容创建。
- 若工作负载需要被外网访问，请确保集群中至少有一个节点已绑定弹性 IP，或已创建负载均衡实例。

说明

单个实例（Pod）内如果有多个容器，请确保容器使用的端口不冲突 ，否则部署会失败。

通过控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 3 配置工作负载的信息。

基本信息

- 负载类型：选择有状态工作负载 StatefulSet。工作负载类型的介绍请参见工作负载概述。
- 负载名称：填写工作负载的名称。
- 命名空间：选择工作负载的命名空间，默认为 default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见创建命名空间。
- 实例数量：填写实例的数量，也就是 Pod 的数量。
- 容器运行时：CCE 集群默认使用普通运行时。

- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见时区同步。

容器配置

- 容器信息

  Pod 中可以配置多个容器，您可以单击右侧“添加容器”为 Pod 配置多个容器。

  - 基本信息：容器基本信息
  - 生命周期：设置容器生命周期
  - 健康检查：设置容器健康检查
  - 环境变量：设置环境变量
  - 数据存储：存储概述

  说明

  - 有状态负载支持“动态挂载”云硬盘。

    动态挂载通过 volumeClaimTemplates 字段实现，并依赖于 StorageClass 动态创建能力。有状态工作负载通过 volumeClaimTemplates 字段为每一个 Pod 关联了一个独有的 PVC，而这个 PVC 又会和对应的 PV 绑定。因此当 Pod 被重新调度后，仍然能够根据该 PVC 名称挂载原有的数据。
  - 负载创建完成后，动态存储不支持更新。
  - 安全设置：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户 ID，容器将以当前用户权限运行。
  - 容器日志：使用 ICAgent 采集容器日志

- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为 default-secret，使用 default-secret 可访问 SWR 镜像仓库的镜像。default-secret 详细说明请参见 default-secret。
- GPU 显卡：默认为不限制。当集群中存在 GPU 节点时，工作负载实例可以调度到指定 GPU 显卡类型的节点上。

实例间发现服务配置

Headless Service 用于解决 StatefulSet 内 Pod 互相访问的问题，Headless Service 给每个 Pod 提供固定的访问域名。具体请参见 Headless Service。

服务配置

服务（Service）是用来解决 Pod 访问问题的。每个 Service 有一个固定 IP 地址，Service 将访问流量转发给 Pod，而且 Service 可以给这些 Pod 做负载均衡。

您也可以在创建完工作负载之后再创建 Service，Service 的概念和使用方法请参见 Service 概述。

步骤 4 单击右下角“创建工作负载”。

----结束

7.4 创建守护进程集(DaemonSet)

操作场景

云容器引擎（CCE）提供多种类型的容器部署和管理能力，支持对容器工作负载的部署、配置、监控、扩容、升级、卸载、服务发现及负载均衡等特性。

其中守护进程集（DaemonSet）可以确保全部（或者某些）节点上仅运行一个 Pod 实例，当有节点加入集群时，也会为他们新增一个 Pod 。 当有节点从集群移除时，这些 Pod 也会被回收。删除 DaemonSet 将会删除它创建的所有 Pod。

使用 DaemonSet 的一些典型用法：

- 运行集群存储 daemon，例如在每个节点上运行 glusterd、ceph。
- 在每个节点上运行日志收集 daemon，例如 fluentd、logstash。
- 在每个节点上运行监控 daemon，例如 Prometheus Node Exporter、collectd、Datadog 代理、New Relic 代理，或 Ganglia gmond。

一种简单的用法是为每种类型的守护进程在所有的节点上都启动一个 DaemonSet。一个稍微复杂的用法是为同一种守护进程部署多个 DaemonSet；每个具有不同的标志，并且对不同硬件类型具有不同的内存、CPU 要求。

前提条件

在创建守护进程集前，您需要存在一个可用集群。若没有可用集群 ，请参照购买 CCE 集群中内容创建。

通过控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 3 配置工作负载的信息。

基本信息

- 负载类型：选择守护进程 DaemonSet。工作负载类型的介绍请参见工作负载概述。
- 负载名称：填写工作负载的名称。
- 命名空间：选择工作负载的命名空间，默认为 default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见创建命名空间。
- 容器运行时：CCE 集群默认使用普通运行时。
- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见时区同步。

容器配置

- 容器信息

Pod 中可以配置多个容器，您可以单击右侧“添加容器”为 Pod 配置多个容器。

- 基本信息：容器基本信息
- 生命周期：设置容器生命周期
- 健康检查：设置容器健康检查
- 环境变量：设置环境变量
- 数据存储：存储概述

说明

负载实例数大于 1 时，不支持挂载云硬盘类型的存储。

- 安全设置：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户 ID，容器将以当前用户权限运行。
- 容器日志：使用 ICAgent 采集容器日志

- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为 default-secret，使用 default-secret 可访问 SWR 镜像仓库的镜像。default-secret 详细说明请参见 default-secret。
- GPU 显卡：默认为不限制。当集群中存在 GPU 节点时，工作负载实例可以调度到指定 GPU 显卡类型的节点上。

服务配置

服务（Service）是用来解决 Pod 访问问题的。每个 Service 有一个固定 IP 地址，Service 将访问流量转发给 Pod，而且 Service 可以给这些 Pod 做负载均衡。

您也可以在创建完工作负载之后再创建 Service，Service 的概念和使用方法请参见 Service 概述。

步骤 4 单击右下角“创建工作负载”。

----结束

7.5 创建普通任务(Job)

操作场景

普通任务是一次性运行的短任务，部署完成后即可执行。正常退出（exit 0）后，任务即执行完成。

普通任务是用来控制批处理型任务的资源对象。批处理业务与长期伺服业务（Deployment、Statefulset）的主要区别是：

批处理业务的运行有头有尾，而长期伺服业务在用户不停止的情况下永远运行。Job 管理的 Pod 根据用户的设置把任务成功完成就自动退出了。成功完成的标志根据不同的 spec.completions 策略而不同，即：

- 单 Pod 型任务有一个 Pod 成功就标志完成。
- 定数成功型任务保证有 N 个任务全部成功。
- 工作队列型任务根据应用确认的全局成功而标志成功。

前提条件

已创建资源，具体操作请参见创建节点。若已有集群和节点资源，无需重复操作。

通过控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 3 配置工作负载的信息。

基本信息

- 负载类型：选择任务 Job。工作负载类型的介绍请参见工作负载概述。
- 负载名称：填写工作负载的名称。
- 命名空间：选择工作负载的命名空间，默认为 default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见创建命名空间。
- 实例数量：填写实例的数量，也就是 Pod 的数量。
- 容器运行时：CCE 集群默认使用普通运行时。

容器配置

- 容器信息

  Pod 中可以配置多个容器，您可以单击右侧“添加容器”为 Pod 配置多个容器。

  - 基本信息：容器基本信息
  - 生命周期：设置容器生命周期
  - 环境变量：设置环境变量
  - 数据存储：存储概述

  说明

  负载实例数大于 1 时，不支持挂载云硬盘类型的存储。

  - 容器日志：使用 ICAgent 采集容器日志
- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为 default-secret，使用 default-secret 可访问 SWR 镜像仓库的镜像。default-secret 详细说明请参见 default-secret。
- GPU 显卡：默认为不限制。当集群中存在 GPU 节点时，工作负载实例可以调度到指定 GPU 显卡类型的节点上。

步骤 4 单击右下角“创建工作负载”。

----结束

7.6 创建定时任务(CronJob)

操作场景

定时任务是按照指定时间周期运行的短任务。使用场景为在某个固定时间点，为所有运行中的节点做时间同步。

定时任务是基于时间的 Job，就类似于 Linux 系统的 crontab，在指定的时间周期运行指定的 Job，即：

- 在给定时间点只运行一次。
- 在给定时间点周期性地运行。

CronJob 的典型用法如下所示：

- 在给定的时间点调度 Job 运行。
- 创建周期性运行的 Job，例如数据库备份、发送邮件。

前提条件

已创建资源，具体操作请参见创建节点。

通过控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 3 配置工作负载的信息。

基本信息

- 负载类型：选择定时任务 CronJob。工作负载类型的介绍请参见工作负载概述。
- 负载名称：填写工作负载的名称。
- 命名空间：选择工作负载的命名空间，默认为 default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见创建命名空间。
- 容器运行时：CCE 集群默认使用普通运行时。

容器配置

- 容器信息

  Pod 中可以配置多个容器，您可以单击右侧“添加容器”为 Pod 配置多个容器。

  - 基本信息：容器基本信息
  - 生命周期：设置容器生命周期
  - 环境变量：设置环境变量
  - 容器日志：使用 ICAgent 采集容器日志
- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为 default-secret，使用 default-secret 可访问 SWR 镜像仓库的镜像。default-secret 详细说明请参见 default-secret。

- GPU 显卡：默认为不限制。当集群中存在 GPU 节点时，工作负载实例可以调度到指定 GPU 显卡类型的节点上。

定时规则

- 并发策略：支持如下三种模式。
  - Forbid：在前一个任务未完成时，不创建新任务。
  - Allow：定时任务不断新建 Job，会抢占集群资源。
  - Replace：已到新任务创建时间点，但前一个任务还未完成，新的任务会取代前一个任务。
- 定时规则：指定新建定时任务在何时执行，YAML 中的定时规则通过 CRON 表达式实现。
  - 以固定周期执行定时任务，支持的周期单位为分钟、小时、日、月。例如，每 30 分钟执行一次任务，对应的 CRON 表达式为“*/30 * * * *”，执行时间将从单位范围内的 0 值开始计算，如 **00:00:00**、**00:30:00**、**01:00:00**、**...**。
  - 以固定时间（按月）执行定时任务。例如，在每个月 1 日的 0 时 0 分执行任务，对应的 CRON 表达式为“0 0 1 */1 *”，执行时间为******-01-01 00:00:00**、******-02-01 00:00:00**、**...**。
  - 以固定时间（按周）执行定时任务。例如，在每周一的 0 时 0 分执行任务，对应的 CRON 表达式为“0 0 * * 1”，执行时间为******-**-01 周一 00:00:00**、******-**-08 周一 00:00:00**、**...**。
  - 自定义 CRON 表达式：关于 CRON 表达式的用法，可参考 CRON 官方文档。

说明

- 以固定时间（按月）执行定时任务时，在某月的天数不存在的情况下，任务将不会在该月执行。例如设置天数为 30，而 2 月份没有 30 号，任务将跳过该月份，在 3 月 30 号继续执行。
- 由于 CRON 表达式的定义，这里的固定周期并非严格意义的周期。将从 0 开始按周期对其时间单位范围（例如单位为分钟时，则范围为 0~59）进行划分，无法整除时最后一个周期会被重置。因此仅在周期能够平均划分其时间单位范围时，才能表示准确的周期。

  举个例子，周期单位为小时，因为“/2、/3、/4、/6、/8 和/12”可将 24 小时整除，所以可以表示准确的周期；而使用其他周期时，在新的一天开始时，最后一个周期将会被重置。比如 CRON 式为“* */12 * * *”时为准确的周期，每天的执行时间为 **00:00:00** 和 **12:00:00**；而 CRON 式为“* */13 * * *”时，每天的执行时间为 **00:00:00** 和 **13:00:00**，在第二天 0 时，虽然没到 13 个小时的周期还是会被刷新。

- 任务记录：可以设置保留执行成功或执行失败的任务个数，设置为 0 表示不保留。

步骤 4 单击右下角“创建工作负载”。

----结束

7.7 管理工作负载和任务

操作场景

工作负载创建后，您可以对其执行升级、编辑 YAML、日志、监控、回退、删除等操作。

表7-2 工作负载/任务管理

操作	描述
监控	可以通过 CCE 控制台查看工作负载和容器组的 CPU 和内存占用情况，以确定需要的资源规格。
日志	可查看工作负载的日志信息。
升级	可以通过更换镜像或镜像版本实现无状态工作负载、有状态工作负载、守护进程集的快速升级，业务无中断。
编辑 YAML	可通过在线 YAML 编辑窗对无状态工作负载、有状态工作负载、守护进程集和容器组的 YAML 文件进行修改和下载。普通任务和定时任务的 YAML 文件仅支持查看、复制和下载。
回退	无状态工作负载可以进行回退操作，仅无状态工作负载可用。
重新部署	工作负载可以进行重新部署操作，重新部署后将重启负载下的全部容器组 Pod。
关闭/开启升级	无状态工作负载可以进行关闭/开启升级操作，仅无状态工作负载可用。
删除	若工作负载无需再使用，您可以将工作负载或任务删除。工作负载或任务删除后，将无法恢复，请谨慎操作。
事件	查看具体实例的事件名称、事件类型、发生次数、Kubernetes 事件、首次和最近发生的时间。
停止/启动	停止/启动一个定时任务，该功能仅定时任务可用。

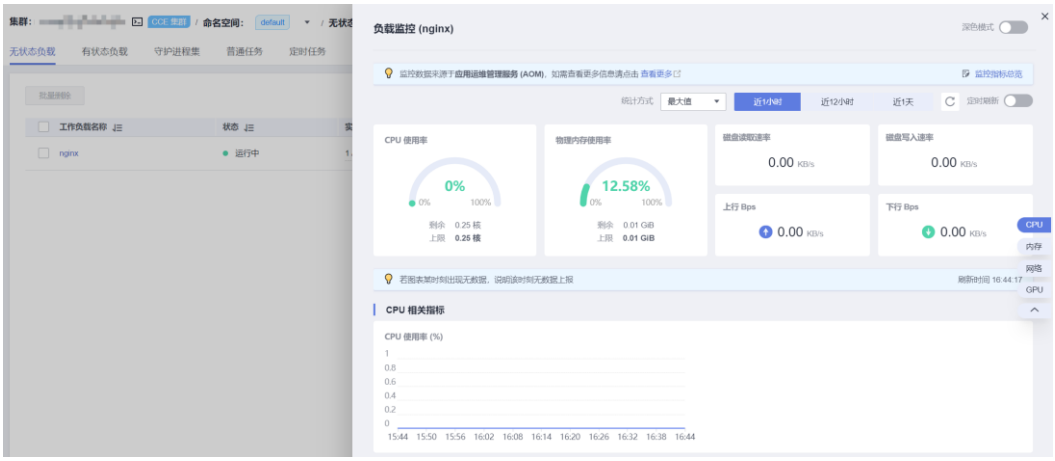
监控

您可以通过 CCE 控制台查看工作负载和容器组的 CPU 和内存占用情况，以确定需要的资源规格。本文以无状态工作负载为例说明如何使用监控功能。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击已创建工作负载后的“监控”。在监控页面，可查看工作负载的 CPU 利用率和物理内存使用率。

图7-4 查看无状态工作负载监控



步骤 3 单击工作负载名称，可在“实例列表”中单击某个实例的“监控”按钮，查看相应实例的 CPU 使用率、内存使用率。

----结束

日志

您可以通过“日志”功能查看无状态工作负载、有状态工作负载、守护进程集、普通任务的日志信息。本文以无状态工作负载为例说明如何查看日志。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载后的“日志”。

在弹出的“日志”窗口中可以根据时间查看日志信息。

图7-5 查看无状态工作负载日志

说明

云容器引擎服务对接了应用运维管理服务 AOM 提供日志查看、检索功能，默认存储时长为 7 天。目前 AOM 每月赠送 500M 免费日志采集额度，超过免费额度部分将产生费用。

----结束

升级

您可以通过 CCE 控制台实现无状态工作负载、有状态工作负载、守护进程集的快速升级。

本文以无状态工作负载为例说明如何进行升级。

若需要更换镜像或镜像版本，您需要提前将镜像上传到容器镜像服务。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击待升级工作负载后的“升级”。

说明

- 暂不支持批量升级多个工作负载。
- 有状态工作负载升级时，若升级类型为替换升级，需要用户手动删除实例后才能升级成功，否则界面会始终显示“升级中”。

步骤 3 请根据业务需求进行工作负载的升级，参数设置方法与创建工作负载时一致。

步骤 4 更新完成后，单击“升级工作负载”，并手动确认 YAML 文件差异后提交升级。

----结束

编辑 YAML

可通过在线 YAML 编辑窗对无状态工作负载、有状态工作负载、守护进程集和容器组的 YAML 文件进行修改和下载。普通任务和定时任务的 YAML 文件仅支持查看、复制和下载。本文以无状态工作负载为例说明如何在线编辑 YAML。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载后的“更多 > 编辑 YAML”，在弹出的“编辑 YAML”窗中可对当前工作负载的 YAML 文件进行修改。

步骤 3 单击“修改”，在弹出的提示框中单击“确定”，完成修改。

步骤 4（可选）在“编辑 YAML”窗中，单击“下载”，可下载该 YAML 文件。

----结束

回退（仅无状态工作负载可用）

所有无状态工作负载的发布历史记录都保留在系统中，您可以回退到指定的版本。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击待回退工作负载后的“更多 > 回退”。

步骤 3 切换至“版本记录”页签，并选择回退版本，单击“回退到此版本”，并手动确认 YAML 文件差异后单击“确定”。

----结束

重新部署

重新部署将重启负载下的全部容器组 Pod。本文以无状态工作负载为例说明如何重新部署工作负载。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载后的“更多 > 重新部署”。

步骤 3 在弹出的提示框中单击“是”，即可完成工作负载的重新部署。

----结束

关闭/开启升级（仅无状态工作负载可用）

无状态工作负载可以进行“关闭/开启升级”操作。

- 关闭升级后，对负载进行的升级操作可以正常下发，但不会被应用到实例。

  如果您正在滚动升级的过程中，滚动升级会在关闭升级命令下发后停止，出现新旧实例共存的状态。
- 开启升级后，负载可以正常升级和回退，负载下的实例会与负载当前的最新信息进行一次同步，如果有不一致的，则会自动按照负载的最新信息进行升级。

须知

工作负载状态在关闭升级时无法执行回退操作。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载后方操作栏中的“更多 > 关闭/开启升级”。

步骤 3 在弹出的信息提示框中，单击“是”。

----结束

标签管理

标签是以 key/value 键值对的形式附加在工作负载上的。添加标签后，可通过标签对工作负载进行管理和选择，主要用于设置亲和性与反亲和性调度。您可以给多个工作负载打标签，也可以给指定的某个工作负载打标签。

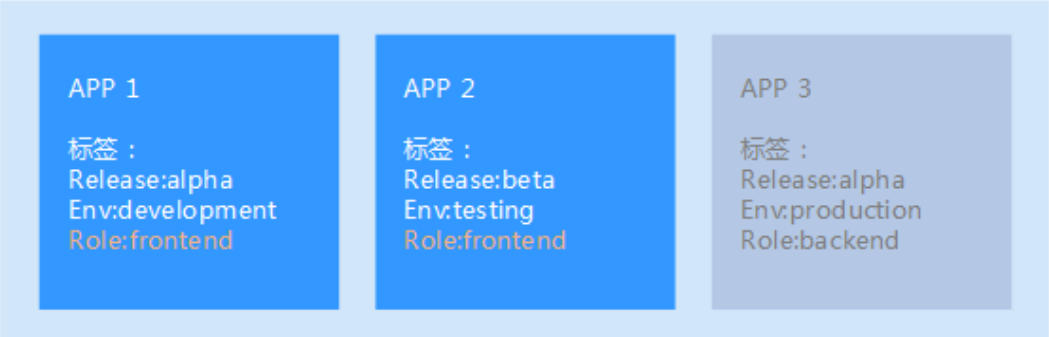
您可以根据业务需求对无状态工作负载、有状态工作负载和守护进程集的标签进行管理，本文以无状态工作负载为例说明如何使用标签管理功能。

如下图，假设为工作负载（例如名称为 APP1、APP2、APP3）定义了 3 个标签：release、env、role。不同工作负载定义了不同的取值，分别为：

- APP 1：[release:alpha;env:development;role:frontend]
- APP 2：[release:beta;env:testing;role:frontend]
- APP 3：[release:alpha;env:production;role:backend]

在使用调度或其他功能时，选择“key/value”值分别为“role/frontend”的工作负载，则会选择到“APP1 和 APP2”。

图7-6 标签案例



步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载后方操作栏中的“更多 > 标签管理”。

步骤 3 单击“添加”，输入键和值后单击“确定”。

图7-7 标签管理

说明

标签格式要求如下：以字母和数字开头或结尾，由字母、数字、连接符（-）、下划线（_）、点号（.）组成且 63 字符以内。

----结束

删除工作负载/任务

若工作负载无需再使用，您可以将工作负载或任务删除。工作负载或任务删除后，将无法恢复，请谨慎操作。本文以无状态工作负载为例说明如何使用删除功能。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 单击待删除工作负载后的“更多 > 删除”，删除工作负载。

请仔细阅读系统提示，删除操作无法恢复，请谨慎操作。

步骤 3 单击“是”。

说明

- 若 Pod 所在节点不可用或者关机，负载无法删除时可以在详情页面实例列表选择强制删除。
- 请确保要删除的存储没有被其他负载使用，导入和存在快照的存储只做解关联操作。

----结束

事件

本文以无状态工作负载为例说明如何使用事件功能。任务或定时任务中的事件功能可直接单击工作负载操作栏中的“事件”按钮查看。

步骤 1 登录 CCE 控制台，进入一个已有的集群，在左侧导航栏中选择“工作负载”。

步骤 2 选择“无状态负载”页签，单击工作负载名称，可在“实例列表”中单击某个实例的“事件”按钮，查看该工作负载或具体实例的事件名称、事件类型、发生次数、Kubernetes 事件、首次和最近发生的时间。

说明

事件保存时间为 1 小时，1 小时后自动清除数据。

----结束

7.8 容器设置

7.8.1 容器基本信息

工作负载是 Kubernetes 对一组 Pod 的抽象模型，用于描述业务的运行载体，一个 Pod 可以封装 1 个或多个容器，您可以单击右上方的“添加容器”，添加多个容器镜像并分别进行设置。

表7-3 镜像参数说明

参数	说明
容器名称	为容器命名。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见如何使用第三方镜像。
镜像版本	选择需要部署的镜像版本。
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
CPU 配额	• 申请：容器需要使用的最小 CPU 值，默认 0.25Core。 • 限制：允许容器使用的 CPU 最大值。建议设容器配额的最高限额，避免容器资源超额导致系统故障。
内存配额	• 申请：容器需要使用的内存最小值，默认 512MiB。 • 限制：允许容器使用的内存最大值。如果超过，容器会被终止。 申请和限制的具体请参见设置容器规格。
GPU 配额	当集群中包含 GPU 节点时，才能设置 GPU，无 GPU 节点不显示此选项。

参数	说明
	• 不限制：表示不使用 GPU。 • 独享：单个容器独享 GPU。 • 共享：容器需要使用的 GPU 百分比，例如设置为 10%，表示该容器需使用 GPU 资源的 10%。
NPU 配额	使用 NPU 芯片（昇腾 D310）的数量，必须为整数。 必须安装 NPU 插件后才能使用。
特权容器	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。
初始化容器	选择容器是否作为初始化容器。 Init 容器是一种特殊容器，在 Pod 内的应用容器启动之前运行。详细说明请参见 Init 容器。

7.8.2 如何使用第三方镜像

操作场景

CCE 支持拉取第三方镜像仓库的镜像来创建工作负载。

通常第三方镜像仓库必须经过认证（帐号密码）才能访问，而 CCE 中容器拉取镜像是使用密钥认证方式，这就要求在拉取镜像前先创建镜像仓库的密钥。

前提条件

使用第三方镜像时，请确保工作负载运行的节点可访问公网。

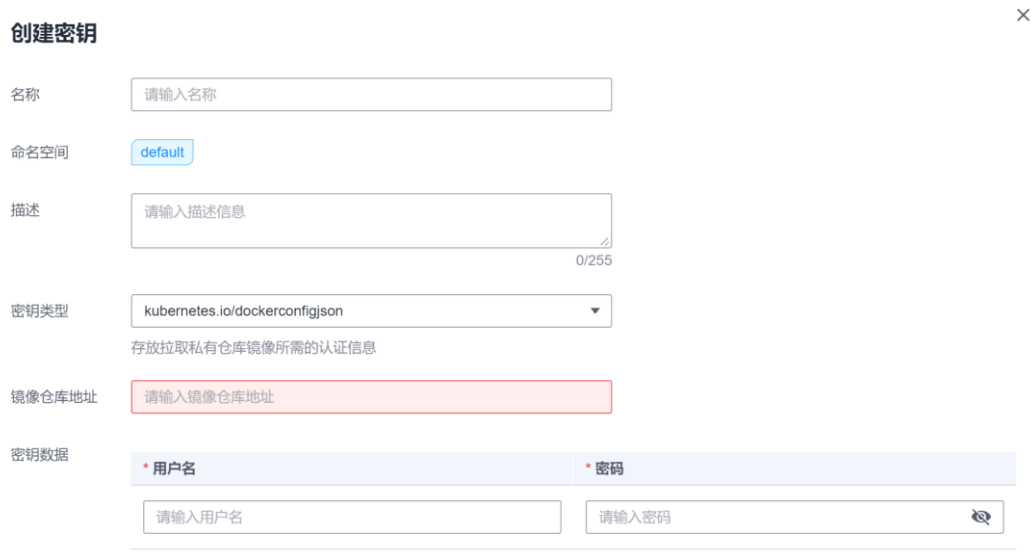
通过界面操作

步骤 1 创建第三方镜像仓库的密钥。

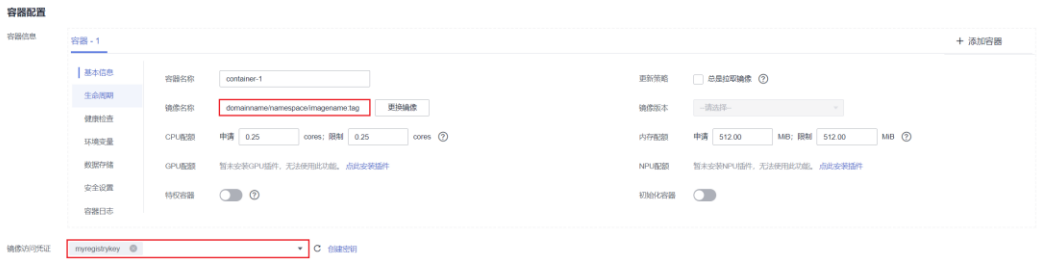
单击集群名称进入集群，在左侧导航栏选择“配置项与密钥”，在右侧选择“密钥”页签，单击右上角“创建密钥”，密钥类型必须选择为 kubernetes.io/dockerconfigjson，如下图所示。详细操作请参见创建密钥。

此处的“用户名”和“密码”请填写第三方镜像仓库的帐号密码。

图7-8 添加密钥



步骤 2 创建工作负载时，可以在“镜像名称”中直接填写私有镜像地址，填写的格式为domainname/namespace/imagename:tag，并选择步骤 1 中创建的密钥。



步骤 3 填写其他参数后，单击“创建工作负载”。

----结束

7.8.3 设置容器规格

操作场景

CCE 支持在创建工作负载时为添加的容器设置资源限制。可以对工作负载中每个实例所用的 CPU 配额、内存配额进行申请和限制。

配置含义

在 CPU 配额和内存配额设置中，**申请**与**限制**的含义如下：

- 申请：根据申请值调度该实例到满足条件的节点去部署工作负载。
- 限制：根据限制值限制工作负载使用的资源。

说明

创建工作负载时，建议设置 CPU 和内存的资源上下限。同一个节点上部署的工作负载，对于未设置资源上下限的工作负载，如果其异常资源泄露会导致其它工作负载分配不到资源而异常。未设置资源上下限的工作负载，工作负载监控信息也会不准确。

配置说明

在实际生产业务中，建议申请和限制比例为 1:1.5 左右，对于一些敏感业务建议设置成 1:1。如果申请值过小而限制值过大，容易导致节点超分严重。如果遇到业务高峰或流量高峰，容易把节点内存或者 CPU 耗尽，导致节点不可用的情况发生。

- CPU 配额：

表7-4 CPU 配额说明

参数	说明
CPU 申请	容器使用的最小 CPU 需求，作为容器调度时资源分配的判断依赖。只有当节点上可分配 CPU 总量 ≥ 容器 CPU 申请数时，才允许将容器调度到该节点。
CPU 限制	容器能使用的 CPU 最大值。

建议配置方法：

节点的实际可用分配 CPU 量 ≥ 当前实例所有容器 CPU 限制值之和 ≥ 当前实例所有容器 CPU 申请值之和，节点的实际可用分配 CPU 量请在“资源管理 > 节点管理”中对应节点的“可分配资源”列下查看“CPU: ** Core”。

- 内存配额：

表7-5 内存配额说明

参数	说明
内存申请	容器使用的最小内存需求，作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量 ≥ 容器内存申请数时，才允许将容器调度到该节点。
内存限制	容器能使用的内存最大值。当内存使用率超出设置的内存限制值时，该实例可能会被重启进而影响工作负载的正常使用。

建议配置方法：

节点的实际可用分配内存量 ≥ 当前节点所有容器内存限制值之和 ≥ 当前节点所有容器内存申请值之和，节点的实际可用分配内存量请在“资源管理 > 节点管理”中对应节点的“可分配资源”列下查看“内存: ** GiB”。

说明

可分配资源：可分配量按照实例请求值(request)计算，表示实例在该节点上可请求的资源上限，不代表节点实际可用资源。 计算公式为：

- 可分配 CPU = CPU 总量 - 所有实例的 CPU 请求值 - 其他资源 CPU 预留值
- 可分配内存 = 内存总量 - 所有实例的内存请求值 - 其他资源内存预留值

使用示例

以集群包含一个资源为 4Core 8GB 的节点为例，已经部署一个包含两个实例的工作负载到该集群上，并设置两个实例（实例 1，实例 2）的资源为{CPU 申请，CPU 限制，内存申请，内存限制}={1Core，2Core，2GB，2GB}。

那么节点上 CPU 和内存的资源使用情况如下：

- 节点 CPU 可分配量=4Core-（实例 1 申请的 1Core+实例 2 申请的 1Core）=2Core
- 节点内存可分配量=8GB-（实例 1 申请的 2GB+实例 2 申请的 2GB）=4GB

因此节点还剩余 2Core 4GB 的资源可供下一个新增的实例使用。

7.8.4 设置容器生命周期

操作场景

CCE 提供了回调函数，在容器的生命周期的特定阶段执行调用，比如容器在停止前希望执行某项操作，就可以注册相应的钩子函数。

目前提供的生命周期回调函数如下所示：

- **启动命令：**容器将会以该启动命令启动，请参见启动命令。
- **启动后处理：**容器启动后触发，请参见启动后处理。
- **停止前处理：**容器停止前触发。设置停止前处理，确保升级或实例删除时可提前将实例中运行的业务排水。详细请参见停止前处理。

启动命令

在默认情况下，镜像启动时会运行默认命令，如果想运行特定命令或重写镜像默认值，需要进行相应设置。

Docker 的镜像拥有存储镜像信息的相关元数据，如果不设置生命周期命令和参数，容器运行时将运行镜像制作时提供的默认的命令和参数，Docker 将这两个字段定义为 ENTRYPOINT 和 CMD。

如果在创建工作负载时填写了容器的运行命令和参数，将会覆盖镜像构建时的默认命令 ENTRYPOINT、CMD，规则如下：

表7-6 容器如何执行命令和参数

镜像 ENTRYPOINT	镜像 CMD	容器运行命令	容器运行参数	最终执行
[touch]	[/root/test]	未设置	未设置	[touch /root/test]
[touch]	[/root/test]	[mkdir]	未设置	[mkdir]

镜像 ENTRYPOINT	镜像 CMD	容器运行命令	容器运行参数	最终执行
[touch]	[/root/test]	未设置	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

步骤 1 登录 CCE 控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤 2 在“启动命令”页签，输入运行命令和运行参数。

表7-7 容器启动命令

命令方式	操作步骤
运行命令	输入可执行的命令，例如“/run/server”。 若运行命令有多个，多个命令之间用空格进行分隔。若命令本身带空格，则需要加引号（""）。 **说明** 多命令时，运行命令建议用/bin/sh 或其他的 shell，其他全部命令作为参数来传入。
运行参数	输入控制容器运行命令参数，例如--port=8080。 若参数有多个，多个参数以换行分隔。

----结束

启动后处理

步骤 1 登录 CCE 控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤 2 在“启动后处理”页签，设置启动后处理的参数。

表7-8 启动后处理-参数说明

参数	说明

命令行方式	在容器中执行指定的命令，配置为需要执行的命令。命令的格式为 Command Args[1] Args[2]…（Command 为系统命令或者用户自定义可执行程序，如果未指定路径则在默认路径下寻找可执行程序），如果需要执行多条命令，建议采用将命令写入脚本执行的方式。**不支持后台执行和异步执行的命令。**
	如需要执行的命令如下：
	`exec:`
	`  command:`
	`  - /install.sh`

参数	说明
	``` - install_agent ``` 请在执行脚本中填写: /install install_agent。这条命令表示容器创建成功后将执行 install.sh。
HTTP 请求方式	发起一个 HTTP 调用请求。配置参数如下： • 路径：请求的 URL 路径，可选项。 • 端口：请求的端口，必选项。 • 主机地址：请求的 IP 地址，可选项，默认是容器所在的节点 IP。

----结束

停止前处理

步骤 1 登录 CCE 控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤 2 在“停止前处理”页签，设置停止前处理的命令。

表7-9 停止前处理

参数	说明

命令行方式	在容器中执行指定的命令，配置为需要执行的命令。命令的格式为 Command Args[1] Args[2]…（Command 为系统命令或者用户自定义可执行程序，如果未指定路径则在默认路径下寻找可执行程序），如果需要执行多条命令，建议采用将命令写入脚本执行的方式。
	如需要执行的命令如下：
	``
	exec:
	  command:
	  - /uninstall.sh
	  - uninstall_agent
	``
	请在执行脚本中填写: /uninstall uninstall_agent。这条命令表示容器结束前将执行 uninstall.sh。

HTTP 请求方式	发起一个 HTTP 调用请求。配置参数如下： • 路径：请求的 URL 路径，可选项。 • 端口：请求的端口，必选项。 • 主机地址：请求的 IP 地址，可选项，默认是容器所在的节点 IP。

----结束

YAML 样例

本节以 nginx 为例，说明 kubectl 命令设置容器生命周期的方法。

在以下配置文件中，您可以看到 postStart 命令在容器目录/bin/bash 下写了个 **install.sh** 命令。 preStop 执行 **uninstall.sh** 命令。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        command:
        - sleep 3600                    #启动命令
        imagePullPolicy: Always
        lifecycle:
          postStart:
            exec:
              command:
              - /bin/bash
              - install.sh              #启动后命令
          preStop:
            exec:
              command:
              - /bin/bash
              - uninstall.sh             #停止前命令
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

7.8.5 设置容器健康检查

操作场景

健康检查是指容器运行过程中，根据用户需要，定时检查容器健康状况。若不配置健康检查，如果容器内应用程序异常，Pod 将无法感知，也不会自动重启去恢复。最终导致虽然 Pod 状态显示正常，但 Pod 中的应用程序异常的情况。

Kubernetes 提供了三种健康检查的探针：

- **存活探针**：livenessProbe，用于检测容器是否正常，类似于我们执行 ps 命令检查进程是否存在。如果容器的存活检查失败，集群会对该容器执行重启操作；若容器的存活检查成功则不执行任何操作。

- **就绪探针**：readinessProbe，用于检查用户业务是否就绪，如果未就绪，则不转发流量到当前实例。一些程序的启动时间可能很长，比如要加载磁盘数据或者要依赖外部的某个模块启动完成才能提供服务。这时候程序进程在，但是并不能对外提供服务。这种场景下该检查方式就非常有用。如果容器的就绪检查失败，集群会屏蔽请求访问该容器；若检查成功，则会开放对该容器的访问。
- **启动探针**：startupProbe，用于探测应用程序容器什么时候启动了。 如果配置了这类探测器，就可以控制容器在启动成功后再进行存活性和就绪检查， 确保这些存活、就绪探针不会影响应用程序的启动。 这可以用于对启动慢的容器进行存活性检测，避免它们在启动运行之前就被杀掉。

检查方式

- **HTTP 请求检查**

  HTTP 请求方式针对的是提供 HTTP/HTTPS 服务的容器，集群周期性地对该容器发起 HTTP/HTTPS GET 请求，如果 HTTP/HTTPS response 返回码属于 200~399 范围，则证明探测成功，否则探测失败。使用 HTTP 请求探测必须指定容器监听的端口和 HTTP/HTTPS 的请求路径。

  例如：提供 HTTP 服务的容器，HTTP 检查路径为：/health-check；端口为：80；主机地址可不填，默认为容器实例 IP，此处以 172.16.0.186 为例。那么集群会周期性地对容器发起如下请求：GET http://172.16.0.186:80/health-check。

  图7-9 HTTP 请求检查

- **TCP 端口检查**

  对于提供 TCP 通信服务的容器，集群周期性地对该容器建立 TCP 连接，如果连接成功，则证明探测成功，否则探测失败。选择 TCP 端口探测方式，必须指定容器监听的端口。

  例如：我们有一个 nginx 容器，它的服务端口是 80，我们对该容器配置了 TCP 端口探测，指定探测端口为 80，那么集群会周期性地对该容器的 80 端口发起 TCP 连接，如果连接成功则证明检查成功，否则检查失败。

图7-10 TCP 端口检查

- **执行命令检查**

  命令检查是一种强大的检查方式，该方式要求用户指定一个容器内的可执行命令，集群会周期性地在容器内执行该命令，如果命令的返回结果是 0 则检查成功，否则检查失败。

  对于上面提到的 TCP 端口检查和 HTTP 请求检查，都可以通过执行命令检查的方式来替代：

  - 对于 TCP 端口探测，我们可以写一个程序来对容器的端口进行 connect，如果 connect 成功，脚本返回 0，否则返回-1。
  - 对于 HTTP 请求探测，我们可以写一个脚本来对容器进行 wget。

    wget http://127.0.0.1:80/health-check

    并检查 response 的返回码，如果返回码在 200~399 的范围，脚本返回 0，否则返回-1。如下图：

    图7-11 执行命令检查

须知

- 必须把要执行的程序放在容器的镜像里面，否则会因找不到程序而执行失败。
- 如果执行的命令是一个 shell 脚本，由于集群在执行容器里的程序时，不在终端环境下，因此不能直接指定脚本为执行命令，需要加上脚本解析器。比如脚本是 **/data/scripts/health_check.sh**，那么我们使用执行命令检查时，指定的程序应该是 **sh /data/scripts/health_check.sh**。究其原因是集群在执行容器里的程序时，不在终端环境下。

公共参数说明

表7-10 公共参数说明

参数	参数说明
检测周期（periodSeconds）	探针检测周期，单位为秒。 例如，设置为 30，表示每 30 秒检测一次。
延迟时间（initialDelaySeconds）	延迟检查时间，单位为秒，此设置与业务程序正常启动时间相关。 例如，设置为 30，表明容器启动后 30 秒才开始健康检查，该时间是预留给业务程序启动的时间。
超时时间（timeoutSeconds）	超时时间，单位为秒。 例如，设置为 10，表明执行健康检查的超时等待时间为 10 秒，如果超过这个时间，本次健康检查就被视为失败。若设置为 0 或不设置，默认超时等待时间为 1 秒。
成功阈值（successThreshold）	探测失败后，将状态转变为成功所需要的最小连续成功次数。例如，设置为 1 时，表明健康检查失败后，健康检查需要连续成功 1 次，才认为工作负载状态正常。 默认值是 1，最小值是 1。 存活和启动探测的这个值必须是 1。
最大失败次数（failureThreshold）	当探测失败时重试的次数。 存活探测情况下的放弃就意味着重新启动容器。就绪探测情况下的放弃 Pod 会被打上未就绪的标签。 默认值是 3。最小值是 1。

YAML 示例

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
```

```
spec:
  containers:
  - name: liveness
    image: nginx:alpine
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 80
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:
      exec:
        command:
          - cat
          - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
    startupProbe:
      httpGet:
        path: /healthz
        port: 80
      failureThreshold: 30
      periodSeconds: 10
```

7.8.6 设置环境变量

操作场景

环境变量是指容器运行环境中设定的一个变量，环境变量可以在工作负载部署后修改，为工作负载提供极大的灵活性。

CCE 中设置的环境变量与 Dockerfile 中的“ENV”效果相同。

须知

容器启动后，容器中的内容不应修改。如果修改配置项（例如将容器应用的密码、证书、环境变量配置到容器中），当容器重启（例如节点异常重新调度 Pod）后，会导致配置丢失，业务异常。

配置信息应通过入参等方式导入容器中，以免重启后配置丢失。

环境变量支持如下几种方式设置。

- **自定义**
- **配置项导入**：将配置项中所有键值都导入为环境变量。
- **配置项键值导入**：将配置项中某个键的值导入作为某个环境变量的值。例如将 configmap-example 这个配置项中 configmap_key 的值 configmap_value 导入为环境

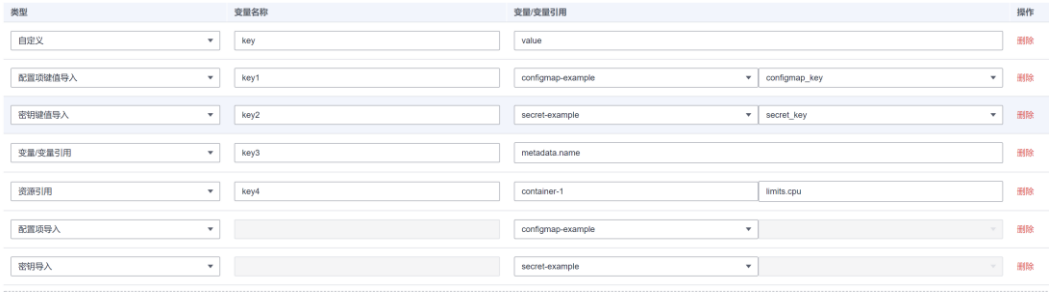
变量 key1 的值，导入后容器中有一个名为 key1 的环境变量，其值为 configmap_value。

- **密钥导入**：将密钥中所有键值都导入为环境变量。
- **密钥键值导入**：将密钥中某个键的值导入作为某个环境变量的值。例如将 secret-example 这个配置项中 secret_key 的值 secret_value 导入为环境变量 key2 的值，导入后容器中有一个名为 key2 的环境变量，其值为 secret_value。
- **变量/变量引用**：用 Pod 定义的字段作为环境变量的值，例如 Pod 的名称。
- **资源引用**：用 Container 定义的字段作为环境变量的值，例如容器的 CPU 限制。

添加环境变量

步骤 1 登录 CCE 控制台，在创建工作负载时，配置容器信息，选择“环境变量”。

步骤 2 设置环境变量。



----结束

YAML 样例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: env-example
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: env-example
  template:
    metadata:
      labels:
        app: env-example
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          imagePullPolicy: Always
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
```

```
            cpu: 250m
            memory: 512Mi
        env:
          - name: key                    # 自定义
            value: value
          - name: key1                   # 配置项键值导入
            valueFrom:
              configMapKeyRef:
                name: configmap-example
                key: key1
          - name: key2                   # 密钥键值导入
            valueFrom:
              secretKeyRef:
                name: secret-example
                key: key2
          - name: key3                   # 变量引用，用 Pod 定义的字段作为环境变量的值
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.name
          - name: key4                   # 资源引用，用 Container 定义的字段作为环境变量的值
            valueFrom:
              resourceFieldRef:
                containerName: container1
                resource: limits.cpu
                divisor: 1
        envFrom:
          - configMapRef:                # 配置项导入
              name: configmap-example
          - secretRef:                   # 密钥导入
              name: secret-example
    imagePullSecrets:
      - name: default-secret
```

环境变量查看

如果 configmap-example 和 secret-example 的内容如下。

```
$ kubectl get configmap configmap-example -oyaml
apiVersion: v1
data:
  configmap_key: configmap_value
kind: ConfigMap
...

$ kubectl get secret secret-example -oyaml
apiVersion: v1
data:
  secret_key: c2VjcmV0X3ZhbHVl            # c2VjcmV0X3ZhbHVl 为 secret_value 的 base64
编码
kind: Secret
...
```

则进入 Pod 中查看的环境变量结果如下。

```
$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
env-example-695b759569-lx9jp   1/1     Running   0          17m

$ kubectl exec env-example-695b759569-lx9jp  -- printenv
/ # env
key=value                           # 自定义环境变量
key1=configmap_value                  # 配置项键值导入
key2=secret_value                    # 密钥键值导入
key3=env-example-695b759569-lx9jp     # Pod 的 metadata.name
key4=1                              # container1 这个容器的 limits.cpu，单位为 Core，向上取整
configmap_key=configmap_value         # 配置项导入，原配置项中的键值直接会导入结果
secret_key=secret_value               # 密钥导入，原密钥中的键值直接会导入结果
```

7.8.7 健康检查 UDP 协议安全组规则说明

操作场景

当负载均衡协议为 UDP 时，健康检查也采用的 UDP 协议，您需要打开其后端服务器的 ICMP 协议安全组规则。

操作步骤

步骤 1 登录弹性云主机控制台，找到集群中的节点对应的节点，单击云主机名称，进入详情页面，记录安全组名称。

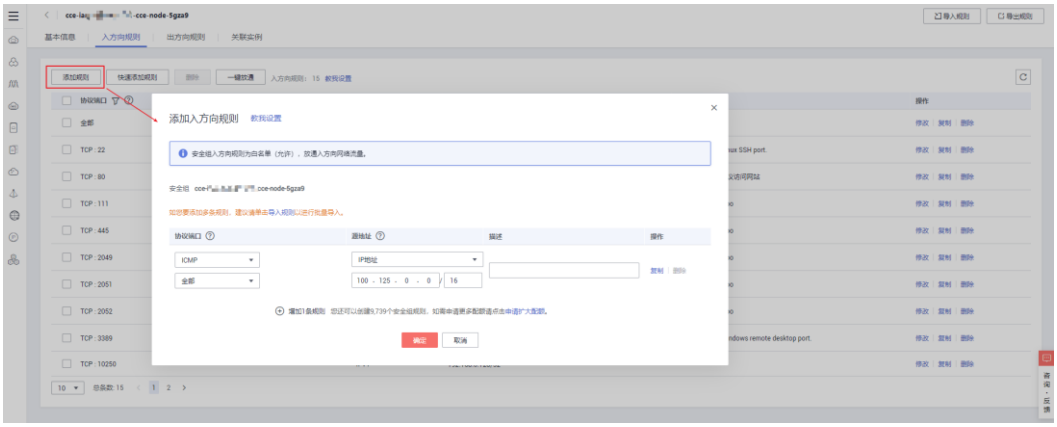
步骤 2 登录虚拟私有云控制台，在左侧导航栏单击“访问控制 > 安全组”，在界面右侧的安全组列表中单击步骤 1 获取的安全组名称。

步骤 3 在打开的页面中单击“入方向规则”页签，单击“添加规则”，为云主机添加入方向规则，单击“确定”。

说明

- 您只需为工作负载所在集群下的任意一个节点更改安全组规则，请添加规则即可，不要修改原有的安全组规则。
- 安全组需放通网段 100.125.0.0/16 流量。

图7-12 添加安全组规则



----结束

7.8.8 配置镜像拉取策略

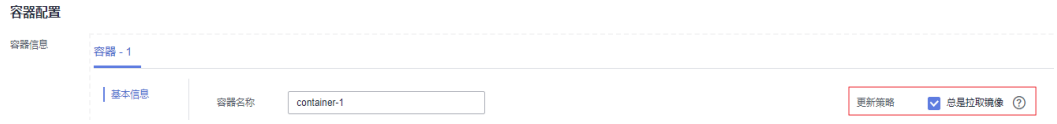
创建工作负载会从镜像仓库拉取容器镜像到节点上，当前 Pod 重启、升级时也会拉取镜像。

默认情况下容器镜像拉取策略 imagePullPolicy 是 **IfNotPresent**，表示如果节点上有这个镜像就直接使用节点已有镜像，如果没有这个镜像就会从镜像仓库拉取。

容器镜像拉取策略还可以设置为 **Always**，表示无论节点上是否有这个镜像，都会从镜像仓库拉取，并覆盖节点上的镜像。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
    resources:
      limits:
        cpu: 100m
        memory: 200Mi
      requests:
        cpu: 100m
        memory: 200Mi
    imagePullPolicy: Always
  imagePullSecrets:
  - name: default-secret
```

在 CCE 控制台也可以设置镜像拉取策略，在创建工作负载时设置。如下所示，勾选表示总是拉取镜像（Always），不勾选就是 IfNotPresent。



须知

建议您在制作镜像时，每次制作一个新的镜像都使用一个新的 Tag，如果不更新 Tag 只更新镜像，当拉取策略选择为 IfNotPresent 时，CCE 会认为当前节点已经存在这个 Tag 的镜像，不会重新拉取。

7.8.9 时区同步

创建工作负载时，支持设置容器使用节点相同的时区。您可以在创建工作负载时打开时区同步配置，如下所示。

时区同步功能依赖容器中挂载的本地磁盘（HostPath），如下所示，开启时区同步后，Pod 中会通过 HostPath 方式，将节点的“/etc/localtime”挂载到容器的“/etc/localtime”，从而使得节点和容器使用相同的时区配置文件。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      volumes:
        - name: vol-162979628557461404
          hostPath:
            path: /etc/localtime
            type: ''
      containers:
        - name: container-0
          image: 'nginx:alpine'
          volumeMounts:
            - name: vol-162979628557461404
              readOnly: true
              mountPath: /etc/localtime
          imagePullPolicy: IfNotPresent
```

```
    imagePullSecrets:
      - name: default-secret
```

7.8.10 工作负载升级配置

在实际应用中，升级是一个常见的场景，Deployment、StatefulSet 和 DaemonSet 都能够很方便的支撑应用升级。

设置不同的升级策略，有如下两种。

- RollingUpdate：滚动升级，即逐步创建新 Pod 再删除旧 Pod，为默认策略。
- Recreate：替换升级，即先把当前 Pod 删掉再重新创建 Pod。

升级参数说明

- 最大浪涌（maxSurge）

  与 spec.replicas 相比，可以有多少个 Pod 存在，默认值是 25%，比如 spec.replicas 为 4，那升级过程中就不能超过 5 个 Pod 存在，即按 1 个的步伐升级，实际升级过程中会换算成数字，且换算会向上取整。这个值也可以直接设置成数字。

  仅 Deployment 支持配置。

- 最大无效实例数（maxUnavailable）

  与 spec.replicas 相比，可以有多少个 Pod 失效，也就是删除的比例，默认值是 25%，比如 spec.replicas 为 4，那升级过程中就至少有 3 个 Pod 存在，即删除 Pod 的步伐是 1。同样这个值也可以设置成数字。

  仅 Deployment 支持配置。

- 实例可用最短时间（minReadySeconds）

  指定新创建的 Pod 在没有任意容器崩溃情况下的最小就绪时间， 只有超出这个时间 Pod 才被视为可用。默认值为 0（Pod 在准备就绪后立即将被视为可用）。

- 最大保留版本数（revisionHistoryLimit）

  用来设定出于回滚目的所要保留的旧 ReplicaSet 数量。 这些旧 ReplicaSet 会消耗 etcd 中的资源，并占用 kubectl get rs 的输出。 每个 Deployment 修订版本的配置都存储在其 ReplicaSets 中；因此，一旦删除了旧的 ReplicaSet， 将失去回滚到 Deployment 的对应修订版本的能力。 默认情况下，系统保留 10 个旧 ReplicaSet，但其理想值取决于新 Deployment 的频率和稳定性。

- 升级最大时长（progressDeadlineSeconds）

  指定系统在报告 Deployment 进展失败 之前等待 Deployment 取得进展的秒数。这类报告会在资源状态中体现为 Type=Progressing、Status=False、Reason=ProgressDeadlineExceeded。Deployment 控制器将持续重试 Deployment。

将来，一旦实现了自动回滚，Deployment 控制器将在探测到这样的条件时立即回滚 Deployment。

如果指定，则此字段值需要大于 .spec.minReadySeconds 取值。

- 缩容时间窗（terminationGracePeriodSeconds）：

  优雅删除时间，默认为 30 秒，删除 Pod 时发送 SIGTERM 终止信号，然后等待容器中的应用程序终止执行，如果在 terminationGracePeriodSeconds 时间内未能终止，则发送 SIGKILL 的系统信号强行终止。

升级示例

Deployment 的升级可以是声明式的，也就是说只需要修改 Deployment 的 YAML 定义即可，比如使用 kubectl edit 命令将上面 Deployment 中的镜像修改为 nginx:alpine。修改完成后再查询 ReplicaSet 和 Pod，发现创建了一个新的 ReplicaSet，Pod 也重新创建了。

```
$ kubectl edit deploy nginx

$ kubectl get rs
NAME             DESIRED   CURRENT   READY    AGE
nginx-6f9f58dffd   2         2         2        1m
nginx-7f98958cdf   0         0         0        48m

$ kubectl get pods
NAME                   READY     STATUS    RESTARTS   AGE
nginx-6f9f58dffd-tdmqk   1/1       Running   0          1m
nginx-6f9f58dffd-tesqr   1/1       Running   0          1m
```

Deployment 可以通过 maxSurge 和 maxUnavailable 两个参数控制升级过程中同时重新创建 Pod 的比例，这在很多时候是非常有用，配置如下所示。

```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
```

在前面的例子中，由于 spec.replicas 是 2，如果 maxSurge 和 maxUnavailable 都为默认值 25%，那实际升级过程中，maxSurge 允许最多 3 个 Pod 存在（向上取整，2*1.25=2.5，取整为 3），而 maxUnavailable 则不允许有 Pod Unavailable（向上取整，2*0.75=1.5，取整为 2），也就是说在升级过程中，一直会有 2 个 Pod 处于运行状态，每次新建一个 Pod，等这个 Pod 创建成功后再删掉一个旧 Pod，直至 Pod 全部为新 Pod。

回滚

回滚也称为回退，即当发现升级出现问题时，让应用回到老的版本。Deployment 可以非常方便的回滚到老版本。

例如上面升级的新版镜像有问题，可以执行 kubectl rollout undo 命令进行回滚。

```
$ kubectl rollout undo deployment nginx
deployment.apps/nginx rolled back
```

Deployment 之所以能如此容易的做到回滚，是因为 Deployment 是通过 ReplicaSet 控制 Pod 的，升级后之前 ReplicaSet 都一直存在，Deployment 回滚做的就是使用之前的 ReplicaSet 再次把 Pod 创建出来。Deployment 中保存 ReplicaSet 的数量可以使用 revisionHistoryLimit 参数限制，默认值为 10。

7.8.11 调度策略（亲和与反亲和）

在创建守护进程集(DaemonSet)中讲到使用 nodeSelector 选择 Pod 要部署的节点，其实 Kubernetes 还支持更精细、更灵活的调度机制，那就是亲和（affinity）与反亲和（anti-affinity）调度。

Kubernetes 支持节点和 Pod 两个层级的亲和与反亲和。通过配置亲和与反亲和规则，可以允许您指定硬性限制或者偏好，例如将前台 Pod 和后台 Pod 部署在一起、某类应用部署到某些特定的节点、不同应用部署到不同的节点等等。

节点亲和（nodeAffinity）

亲和性规则的基础是标签，先来看一下集群中节点上有些什么标签。

```
$ kubectl describe node 192.168.0.212
Name:              192.168.0.212
Roles:             <none>
Labels:            beta.kubernetes.io/arch=amd64
                   beta.kubernetes.io/os=linux
                   failure-domain.beta.kubernetes.io/is-baremetal=false
                   failure-domain.beta.kubernetes.io/region=******
                   failure-domain.beta.kubernetes.io/zone=******
                   kubernetes.io/arch=amd64
                   kubernetes.io/availablezone=******
                   kubernetes.io/eniquota=12
                   kubernetes.io/hostname=192.168.0.212
                   kubernetes.io/os=linux
                   node.kubernetes.io/subnetid=fd43acad-33e7-48b2-a85a-24833f362e0e
                   os.architecture=amd64
                   os.name=EulerOS 2.0 SP5
                   os.version=3.10.0-862.14.1.5.h328.eulerosv2r7.x86_64
```

这些标签都是在创建节点的时候 CCE 会自动添加上的，下面介绍几个在调度中会用到比较多的标签。

- failure-domain.beta.kubernetes.io/region：表示节点所在的区域。
- failure-domain.beta.kubernetes.io/zone：表示节点所在的可用区（availability zone）。
- kubernetes.io/hostname：节点的 hostname。

在 DaemonSet 中介绍了 nodeSelector，通过 nodeSelector 可以让 Pod 只部署在具有特定标签的节点上。如下所示，Pod 只会部署在拥有 gpu=true 这个标签的节点上。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeSelector:              # 节点选择，当节点拥有 gpu=true 标签时才在节点上创建 Pod
```

```
    gpu: true
...
```

通过节点亲和性规则配置，也可以做到同样的事情，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  gpu
  labels:
    app:  gpu
spec:
  selector:
    matchLabels:
      app: gpu
  replicas: 3
  template:
    metadata:
      labels:
        app:  gpu
    spec:
      containers:
      - image:  nginx:alpine
        name:  gpu
        resources:
          requests:
            cpu: 100m
            memory: 200Mi
          limits:
            cpu: 100m
            memory: 200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: gpu
                operator: In
                values:
                - "true"
```

看起来这要复杂很多，但这种方式可以得到更强的表达能力，后面会进一步介绍。

这里 affinity 表示亲和，nodeAffinity 表示节点亲和，requiredDuringSchedulingIgnoredDuringExecution 非常长，不过可以将这个分作两段来看：

- 前半段 requiredDuringScheduling 表示下面定义的规则必须强制满足（require）才会调度 Pod 到节点上。
- 后半段 IgnoredDuringExecution 表示已经在节点上运行的 Pod 不需要满足下面定义的规则，即去除节点上的某个标签，那些需要节点包含该标签的 Pod 不会被重新调度。

另外操作符 operator 的值为 In，表示标签值需要在 values 的列表中，其他 operator 取值如下。

- NotIn：标签的值不在某个列表中
- Exists：某个标签存在
- DoesNotExist：某个标签不存在
- Gt：标签的值大于某个值（字符串比较）
- Lt：标签的值小于某个值（字符串比较）

需要说明的是并没有 nodeAntiAffinity（节点反亲和），因为 NotIn 和 DoesNotExist 可以提供相同的功能。

下面来验证这段规则是否生效，假设某集群有如下三个节点。

```
$ kubectl get node
NAME          STATUS   ROLES    AGE   VERSION
192.168.0.212   Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2
192.168.0.94    Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2
192.168.0.97    Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2
```

首先给 192.168.0.212 这个节点打上 gpu=true 的标签。

```
$ kubectl label node 192.168.0.212 gpu=true
node/192.168.0.212 labeled

$ kubectl get node -L gpu
NAME          STATUS   ROLES    AGE   VERSION                        GPU
192.168.0.212   Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2   true
192.168.0.94    Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2
192.168.0.97    Ready    <none>   13m   v1.15.6-r1-20.3.0.2.B001-15.30.2
```

创建这个 Deployment，可以发现所有的 Pod 都部署在了 192.168.0.212 这个节点上。

```
$ kubectl create -f affinity.yaml
deployment.apps/gpu created

$ kubectl get pod -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP           NODE
gpu-6df65c44cf-42xw4    1/1     Running   0          15s   172.16.0.37   192.168.0.212
gpu-6df65c44cf-jzjvs    1/1     Running   0          15s   172.16.0.36   192.168.0.212
gpu-6df65c44cf-zv5cl    1/1     Running   0          15s   172.16.0.38   192.168.0.212
```

节点优先选择规则

上面讲的 requiredDuringSchedulingIgnoredDuringExecution 是一种**强制**选择的规则，节点亲和还有一种优先选择规则，即 preferredDuringSchedulingIgnoredDuringExecution，表示会根据规则**优先**选择哪些节点。

为演示这个效果，先为上面的集群添加一个 SAS 磁盘的节点，并打上 DISK=SAS 的标签，为另外三个节点打上 DISK=SSD 的标签。

```
$ kubectl get node -L DISK,gpu
NAME          STATUS   ROLES    AGE     VERSION                        DISK    GPU
192.168.0.100   Ready    <none>   7h23m   v1.15.6-r1-20.3.0.2.B001-15.30.2   SAS
192.168.0.212   Ready    <none>   8h      v1.15.6-r1-20.3.0.2.B001-15.30.2   SSD
```

```
true
192.168.0.94    Ready    <none>   8h     v1.15.6-r1-20.3.0.2.B001-15.30.2   SSD
192.168.0.97    Ready    <none>   8h     v1.15.6-r1-20.3.0.2.B001-15.30.2   SSD
```

下面定义一个 Deployment，要求 Pod 优先部署在 SSD 磁盘的节点上，可以像下面这样定义，使用 preferredDuringSchedulingIgnoredDuringExecution 规则，给 SSD 设置权重（weight）为 80，而 gpu=true 权重为 20，这样 Pod 就优先部署在 SSD 的节点上。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  gpu
  labels:
    app:  gpu
spec:
  selector:
    matchLabels:
      app: gpu
  replicas: 10
  template:
    metadata:
      labels:
        app:  gpu
    spec:
      containers:
      - image:  nginx:alpine
        name:  gpu
        resources:
          requests:
            cpu:  100m
            memory:  200Mi
          limits:
            cpu:  100m
            memory:  200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 80
            preference:
              matchExpressions:
              - key: DISK
                operator: In
                values:
                - SSD
          - weight: 20
            preference:
              matchExpressions:
              - key: gpu
                operator: In
                values:
                - "true"
```

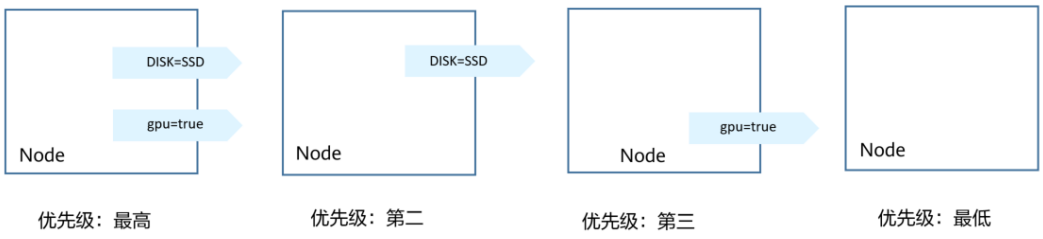
来看实际部署后的情况，可以看到部署到 192.168.0.212（标签为 DISK=SSD、gpu=true）这个节点上的 Pod 有 5 个，192.168.0.97（标签为 DISK=SSD）上有 3 个，而 192.168.0.100（标签为 DISK=SAS）上只有 2 个。

这里您看到 Pod 并没有调度到 192.168.0.94（标签为 DISK=SSD）这个节点上，这是因为这个节点上部署了很多其他 Pod，资源使用较多，所以并没有往这个节点上调度，这也侧面说明 preferredDuringSchedulingIgnoredDuringExecution 是优先规则，而不是强制规则。

```
$ kubectl create -f affinity2.yaml
deployment.apps/gpu created

$ kubectl get po -o wide
NAME                READY   STATUS    RESTARTS   AGE     IP           NODE
gpu-585455d466-5bmcz   1/1     Running   0          2m29s   172.16.0.44   192.168.0.212
gpu-585455d466-cg2l6   1/1     Running   0          2m29s   172.16.0.63   192.168.0.97
gpu-585455d466-f2bt2   1/1     Running   0          2m29s   172.16.0.79   192.168.0.100
gpu-585455d466-hdb5n   1/1     Running   0          2m29s   172.16.0.42   192.168.0.212
gpu-585455d466-hkgvz   1/1     Running   0          2m29s   172.16.0.43   192.168.0.212
gpu-585455d466-mngvn   1/1     Running   0          2m29s   172.16.0.48   192.168.0.97
gpu-585455d466-s26qs   1/1     Running   0          2m29s   172.16.0.62   192.168.0.97
gpu-585455d466-sxtzm   1/1     Running   0          2m29s   172.16.0.45   192.168.0.212
gpu-585455d466-t56cm   1/1     Running   0          2m29s   172.16.0.64   192.168.0.100
gpu-585455d466-t5w5x   1/1     Running   0          2m29s   172.16.0.41   192.168.0.212
```

上面这个例子中，对于节点排序优先级如下所示，有个两个标签的节点排序最高，只有 SSD 标签的节点排序第二（权重为 80），只有 gpu=true 的节点排序第三，没有的节点排序最低。

图7-13 优先级排序顺序



工作负载亲和（podAffinity）

节点亲和的规则只能影响 Pod 和节点之间的亲和，Kubernetes 还支持 Pod 和 Pod 之间的亲和，例如将应用的前端和后端部署在一起，从而减少访问延迟。Pod 亲和同样有 requiredDuringSchedulingIgnoredDuringExecution 和 preferredDuringSchedulingIgnoredDuringExecution 两种规则。

来看下面这个例子，假设有个应用的后端已经创建，且带有 app=backend 的标签。

```
$ kubectl get po -o wide
NAME                   READY   STATUS    RESTARTS   AGE     IP           NODE
backend-658f6cb858-dlrz8   1/1     Running   0          2m36s   172.16.0.67
192.168.0.100
```

将前端 frontend 的 pod 部署在 backend 一起时，可以做如下 Pod 亲和规则配置。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:   frontend
  labels:
    app:  frontend
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app:  frontend
    spec:
      containers:
      - image:  nginx:alpine
        name:  frontend
        resources:
          requests:
            cpu:  100m
            memory:  200Mi
          limits:
            cpu:  100m
            memory:  200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - backend
```

创建 frontend 然后查看，可以看到 frontend 都创建到跟 backend 一样的节点上了。

```
$ kubectl create -f affinity3.yaml
deployment.apps/frontend created

$ kubectl get po -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP           NODE
backend-658f6cb858-dlrz8    1/1     Running   0          5m38s   172.16.0.67
192.168.0.100
frontend-67ff9b7b97-dsqzn   1/1     Running   0          6s      172.16.0.70
192.168.0.100
frontend-67ff9b7b97-hxm5t   1/1     Running   0          6s      172.16.0.71
192.168.0.100
frontend-67ff9b7b97-z8pdb   1/1     Running   0          6s      172.16.0.72
192.168.0.100
```

这里有个 **topologyKey** 字段（拓扑域），意思是先圈定 topologyKey 指定的范围，然后再选择下面规则定义的内容。这里每个节点上都有 kubernetes.io/hostname，所以看不出 topologyKey 起到的作用。

如果 backend 有两个 Pod，分别在不同的节点上。

```
$ kubectl get po -o wide
NAME                   READY   STATUS    RESTARTS   AGE     IP           NODE
backend-658f6cb858-5bpd6   1/1     Running   0          23m     172.16.0.40
192.168.0.97
backend-658f6cb858-dlrz8   1/1     Running   0          2m36s   172.16.0.67
192.168.0.100
```

给 192.168.0.97 和 192.168.0.94 打一个 prefer=true 的标签。

```
$ kubectl label node 192.168.0.97 prefer=true
node/192.168.0.97 labeled
$ kubectl label node 192.168.0.94 prefer=true
node/192.168.0.94 labeled

$ kubectl get node -L prefer
NAME          STATUS   ROLES    AGE   VERSION                          PREFER
192.168.0.100   Ready    <none>   44m   v1.15.6-r1-20.3.0.2.B001-15.30.2
192.168.0.212   Ready    <none>   91m   v1.15.6-r1-20.3.0.2.B001-15.30.2
192.168.0.94    Ready    <none>   91m   v1.15.6-r1-20.3.0.2.B001-15.30.2   true
192.168.0.97    Ready    <none>   91m   v1.15.6-r1-20.3.0.2.B001-15.30.2   true
```

将 podAffinity 的 topologyKey 定义为 prefer。

```
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - topologyKey: prefer
          labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - backend
```

调度时，先圈定拥有 prefer 标签的节点，这里也就是 192.168.0.97 和 192.168.0.94，然后再匹配 app=backend 标签的 Pod，从而 frontend 就会全部部署在 192.168.0.97 上。

```
$ kubectl create -f affinity3.yaml
deployment.apps/frontend created

$ kubectl get po -o wide
NAME                    READY   STATUS    RESTARTS   AGE     IP           NODE
backend-658f6cb858-5bpd6    1/1     Running   0          26m     172.16.0.40
192.168.0.97
backend-658f6cb858-dlrz8    1/1     Running   0          5m38s   172.16.0.67
192.168.0.100
frontend-67ff9b7b97-dsqzn   1/1     Running   0          6s      172.16.0.70
192.168.0.97
frontend-67ff9b7b97-hxm5t   1/1     Running   0          6s      172.16.0.71
192.168.0.97
```

```
frontend-67ff9b7b97-z8pdb   1/1     Running   0          6s     172.16.0.72
192.168.0.97
```

工作负载反亲和（podAntiAffinity）

前面讲了 Pod 的亲和，通过亲和将 Pod 部署在一起，有时候需求却恰恰相反，需要将 Pod 分开部署，例如 Pod 之间部署在一起会影响性能的情况。

下面例子中定义了反亲和规则，这个规则表示 Pod 不能调度到拥有 app=frontend 标签 Pod 的节点上，也就是下面将 frontend 分别调度到不同的节点上（每个节点只有一个 Pod）。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:   frontend
  labels:
    app:  frontend
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 5
  template:
    metadata:
      labels:
        app:  frontend
    spec:
      containers:
      - image:  nginx:alpine
        name:  frontend
        resources:
          requests:
            cpu:  100m
            memory:  200Mi
          limits:
            cpu:  100m
            memory:  200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - frontend
```

创建并查看，可以看到每个节点上只有一个 frontend 的 Pod，还有一个在 Pending，因为在部署第 5 个时 4 个节点上都有了 app=frontend 的 Pod，所以第 5 个一直是 Pending。

```
$ kubectl create -f affinity4.yaml
deployment.apps/frontend created

$ kubectl get po -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP           NODE
frontend-6f686d8d87-8dlsc   1/1     Running   0          18s   172.16.0.76
192.168.0.100
frontend-6f686d8d87-d6l8p   0/1     Pending   0          18s   <none>       <none>
frontend-6f686d8d87-hgcq2   1/1     Running   0          18s   172.16.0.54
192.168.0.97
frontend-6f686d8d87-q7cfq   1/1     Running   0          18s   172.16.0.47
192.168.0.212
frontend-6f686d8d87-xl8hx   1/1     Running   0          18s   172.16.0.23
192.168.0.94
```

通过控制台配置调度策略

步骤 1 登录 CCE 控制台。

步骤 2 在创建工作负载时，在“高级设置”中找到“调度策略”。

表7-11 节点亲和性设置

参数名	参数描述
必须满足	即硬约束，设置必须要满足的条件，对应于requiredDuringSchedulingIgnoredDuringExecution，多条规则间是一种“或”的关系，即只需要满足一条规则即会进行调度。
尽量满足	即软约束，设置尽量满足的条件，对应于preferredDuringSchedulingIgnoredDuringExecution，无论是满足其中一条或者是都不满足都会进行调度。

步骤 3 在“节点亲和性”、“工作负载亲和性”、“工作负载反亲和性”下单击 添加调度策略。在弹出的窗口中可以直接添加策略、指定节点或指定可用区。

指定节点和指定可用区本质也是通过标签实现，只是通过控制台提供了更为便捷的操作。指定节点使用的是 kubernetes.io/hostname 标签，可用区使用的是 failure-domain.beta.kubernetes.io/zone 标签。

表7-12 调度策略设置

参数名	参数描述
标签名	对应节点的标签，可以使用默认的标签也可以用户自定义标签。
操作符	可以设置六种匹配关系（In, NotIn, Exists, DoesNotExist. Gt, and Lt）。 • In：是否在标签值的列表中

参数名	参数描述
	• NotIn：是否不在标签值的列表中 • Exists：某个标签存在 • DoesNotExist：某个标签不存在 • Gt：标签的值大于某个值（字符串比较） • Lt：标签的值小于某个值（字符串比较）
标签值	请填写标签值。
命名空间	仅支持在工作负载亲和/工作负载反亲和调度策略中使用。 指定调度策略生效的命名空间。
拓扑域	仅支持在工作负载亲和/工作负载反亲和调度策略中使用。 先圈定拓扑域（topologyKey）指定的范围，然后再选择策略定义的内容。
权重	仅支持在“尽量满足”策略中添加。

----结束

7.8.12 实例缩容优先级说明

缩容优先级

Pod 在缩容时，会按照如下优先级缩容：

1. 未被调度的 Pod
2. Pod Pending < Pod Unknown < Pod Running
3. Not ready < ready
4. 带有特定标签的 Pod，cce.io/priordeletion
5. Doubled up < not doubled up 优先缩容 Pod 多的节点
6. 缩容 Ready 时间较长的 Pod
7. 重启次数多的 Pod
8. createtimestamp 较前的 Pod

优先缩容老实例生效说明

- 优先缩容老实例**不生效**：

  按照上述优先级，如果节点数为 2（同规格），Deployment 实例数为 5 时（可以是任意奇数个），此时 A 节点分布 3 个 Pod，B 节点分布 2 个 Pod；如果此时将实例数缩容至 3 个，上述步骤中会先执行 5，此时根据 Pod 判断，会确定缩容的节点为 A；再执行 6，缩容一个老实例后发现目前已无老实例，且限定了缩容节点为 A，此时会缩容一个新实例。

- 优先缩容老实例**生效**：如果节点数为 2（同规格），Deployment 实例数为 6 时（可以是任意偶数个），此时 A、B 节点均分布 3 个 Pod；如果此时将实例数缩容至 4 个，此时代码会判定不符合 5 的条件，此时根据 Pod 判断，会确定缩容的节点为 A、B；再执行 6，回去将两个节点上的实例排序，成功缩容两个老的实例。

7.9 登录容器

操作场景

如果在使用容器的过程中遇到非预期的问题，您可登录容器进行调试。

使用 kubectl 命令登录容器

步骤 1 使用 kubectl 连接集群，详情请参见“通过 kubectl 连接集群”。

步骤 2 执行以下命令，查看已创建的 Pod。

```
kubectl get pod
```

示例输出如下：

```
NAME                          READY   STATUS    RESTARTS      AGE
nginx-59d89cb66f-mhljr        1/1     Running   0             11m
```

步骤 3 查询该 Pod 中的容器名称。

```
kubectl get po nginx-59d89cb66f-mhljr -o
jsonpath='{range .spec.containers[*]}{.name}{end}{"\n"}'
```

示例输出如下：

```
container-1
```

步骤 4 执行以下命令，登录到 nginx-59d89cb66f-mhljr 这个 Pod 中名为 container-1 的容器。

```
kubectl exec -it nginx-59d89cb66f-mhljr -c container-1 -- /bin/sh
```

步骤 5 如需退出容器，可执行 exit 命令。

7.10 Pod 标签与注解

Pod 注解

CCE 提供一些使用 Pod 的高级功能，这些功能使用时可以通过给 YAML 添加注解 Annotation 实现。具体的 Annotation 如下表所示。

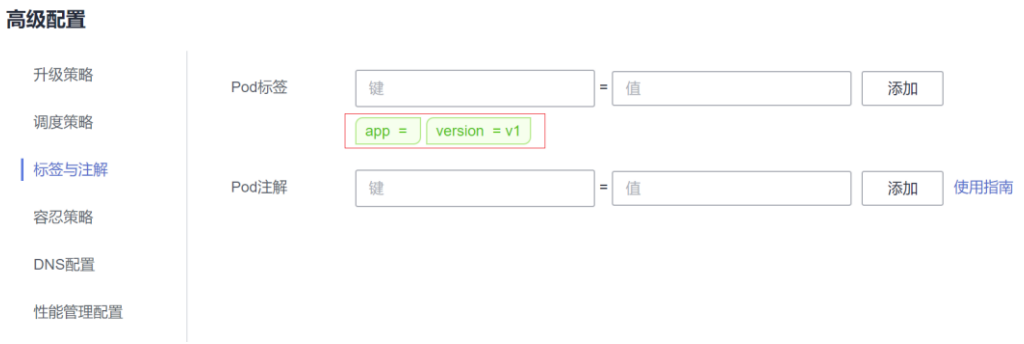
表7-13 Pod Annotation

注解	说明	默认值
kubernetes.AOM.log.std	容器标准输出采集参数，不配置默	-

注解	说明	默认值
out	认将全部容器的标准输出上报至AOM，可配置采集指定容器或全部不采集。 示例： • 全部不采集 kubernetes.AOM.log.stdout: '[]' • 采集 container-1 和 container-2 容器。 kubernetes.AOM.log.stdout: '["container-1","container-2"]'	
metrics.alpha.kubernetes.io/custom-endpoints	AOM 监控指标上报参数，可将指定指标上报是 AOM 服务。	-
prometheus.io/scrape	Prometheus 指标上报参数，值为 true 表示当前负载开启上报。	-
prometheus.io/path	Prometheus 采集的 url 路径。	/metrics
prometheus.io/port	Prometheus 采集的 endpoint 端口号。	-
prometheus.io/scheme	Prometheus 采集协议，值可以填写 http 或 https	-
kubernetes.io/ingress-bandwidth	Pod 的入口带宽	-
kubernetes.io/egress-bandwidth	Pod 的出口带宽	-

Pod 标签

在控制台创建工作负载时，会默认为 Pod 添加如下标签，其中 app 的值为工作负载名称。您也可以根据需要为 Pod 添加其他标签。



在此处添加的 Pod 标签，同时也会在工作负载中添加 selector.matchLabels，且一一对应，YAML 示例如下。

```
...
spec:
  selector:
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      ...
```

8 调度管理

8.1 调度概述

调度概述

CCE 支持不同类型的资源调度及任务调度等，可提升应用的性能和集群整体资源的利用率。本文介绍 CPU、GPU 等类型的异构资源调度功能。

CPU 调度

CCE 提供 CPU 管理策略为应用分配完整的 CPU 物理核，提升应用性能，减少应用的调度延迟。

功能	描述	文档参考
CPU 管理策略	当节点上运行了很多 CPU 密集的 Pod 时，工作负载可能会迁移到不同的 CPU 核。许多应用对这种迁移不敏感，因此无需任何干预即可正常工作。有些应用对 CPU 敏感，对于 CPU 敏感型应用，您可以利用 Kubernetes 中提供的 CPU 管理策略为应用分配独占核，提升应用性能，减少应用的调度延迟。	CPU 调度

GPU 调度

CCE 为集群中的 GPU 异构资源提供调度能力，支持在容器中使用 GPU 显卡。

功能	描述	文档参考
GPU 管理策略	Kubernetes 默认 GPU 调度可以指定 Pod 申请 GPU 的数量，支持申请设置为小于 1 的数量，实现多个 Pod 共享使用 GPU。	GPU 调度

8.2 CPU 调度

CPU 管理策略

默认情况下，kubelet 使用 CFS 配额 来执行 Pod 的 CPU 约束。 当节点上运行了很多 CPU 密集的 Pod 时，工作负载可能会迁移到不同的 CPU 核， 这取决于调度时 Pod 是否被扼制，以及哪些 CPU 核是可用的。许多应用对这种迁移不敏感，因此无需任何干预即可正常工作。有些应用对 CPU 敏感，CPU 敏感型应用有如下特点。

- 对 CPU throttling 敏感
- 对上下文切换敏感
- 对处理器缓存未命中敏感
- 对跨 socket 内存访问敏感
- 期望运行在同一物理 CPU 的超线程

如果您的应用有以上其中一个特点，可以利用 kubernetes 中提供的绑核策略去给应用绑核，提升应用性能，减少应用的调度延迟。cpu manager 会优先在一个 Socket 上分配资源，也会优先分配完整的物理核，避免一些干扰。

如何为 Pod 绑核

想要让 Pod 能够绑核，有如下几点要求：

- 节点上开启静态绑核策略。具体方法请参见开启 CPU 管理策略。
- Pod 的定义里都要设置 request 和 limits，request 和 limits 要一致。
- 对于要绑核的容器，request 值必须是整数。
- 如果有 init container 希望进行绑核的话，init container 的 request 最好与业务容器设置的 request 一致（避免业务容器未继承 init container 的 cpu 分配结果，导致 cpu manager 多预留一部分 cpu）。

在使用时您可以利用调度策略（亲和与反亲和）将如上配置的 Pod 调度到开启静态绑核策略的节点上，这样就能够达到绑核的效果。

开启 CPU 管理策略

CPU 管理策略通过 kubelet 参数 --cpu-manager-policy 来指定。支持两种策略：

- 关闭（none）：默认策略，显式地启用现有的默认 CPU 亲和方案，不提供操作系统调度器默认行为之外的亲和性策略。
- 开启（static）：针对具有整数型 CPU requests 的 Guaranteed Pod ，它允许该类 Pod 中的容器访问节点上的独占 CPU 资源（绑核）。

在创建集群时的高级配置中可以配置 CPU 管理策略，如下图所示。

另外在节点池中也可以配置 CPU 管理策略，配置后会自动修改节点的上 kubelet 参数 --cpu-manager-policy。登录 CCE 控制台，单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签，单击节点池名称后的“更多 > 配置管理”，将 cpu-manager-policy 的值修改为 static 即可。

为 Pod 设置独占 CPU

Pod 设置独占 CPU（即 CPU 绑核）有如下几点要求：

- 节点上开启静态（static）CPU 管理策略，具体方法请参见开启 CPU 管理策略。
- Pod 的定义里都要设置 requests 和 limits 参数，requests 和 limits 必须为整数，且数值一致。
- 如果有 init container 需要设置独占 CPU，init container 的 requests 参数建议与业务容器设置的 requests 参数一致（避免业务容器未继承 init container 的 CPU 分配结果，导致 CPU manager 多预留一部分 CPU）。更多信息请参见 App Containers can't inherit Init Containers CPUs - CPU Manager Static Policy。

在使用时您可以利用“调度策略（亲和与反亲和）”将如上配置的 Pod 调度到开启静态（static）CPU 管理策略的节点上，这样就能够达到独占 CPU 的效果。

设置独占 CPU 的 YAML 示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          resources:
            requests:
              cpu: 2          # 必须为整数，且需要与 limits 中一致
              memory: 2048Mi
            limits:
              cpu: 2          # 必须为整数，且需要与 requests 中一致
              memory: 2048Mi
      imagePullSecrets:
        - name: default-secret
```

8.3 GPU 调度

CCE 支持在容器中使用 GPU 资源。

前提条件

- 创建 GPU 类型节点，具体请参见创建节点。
- 安装 gpu-beta 插件，安装时注意要选择节点上 GPU 对应的驱动。
- gpu-beta 会把驱动的目录挂载到/usr/local/nvidia/lib64，在容器中使用 GPU 资源需要将/usr/local/nvidia/lib64 追加到 LD_LIBRARY_PATH 环境变量中。

  通常可以通过如下三种方式追加。

  a. 制作镜像的 Dockerfile 中配置 LD_LIBRARY_PATH。（推荐）

  ```
  ENV LD_LIBRARY_PATH /usr/local/nvidia/lib64:$LD_LIBRARY_PATH
  ```

  b. 镜像的启动命令中配置 LD_LIBRARY_PATH。

  ```
  /bin/bash -c "export
  LD_LIBRARY_PATH=/usr/local/nvidia/lib64:$LD_LIBRARY_PATH && ..."
  ```

  c. 创建工作负载时定义 LD_LIBRARY_PATH 环境变量（需确保容器内未配置该变量，不然会被覆盖）。

  ```
          env:
            - name: LD_LIBRARY_PATH
              value: /usr/local/nvidia/lib64
  ```

使用 GPU

创建工作负载申请 GPU 资源，可按如下方法配置，指定显卡的数量。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-test
  template:
    metadata:
      labels:
        app: gpu-test
    spec:
      containers:
      - image: nginx:perl
        name: container-0
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 1   # 申请 GPU 的数量
          limits:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 1   # GPU 数量的使用上限
      imagePullSecrets:
      - name: default-secret
```

通过 **nvidia.com/gpu** 指定申请 GPU 的数量，支持申请设置为小于 1 的数量，比如 **nvidia.com/gpu: 0.5**，这样可以多个 Pod 共享使用 GPU。GPU 数量小于 1 时，不支持跨 GPU 分配，如 0.5 GPU 只会分配到一张卡上。

指定 **nvidia.com/gpu** 后，在调度时不会将负载调度到没有 GPU 的节点。如果缺乏 GPU 资源，会报类似如下的 Kubernetes 事件。

- 0/2 nodes are available: 2 Insufficient nvidia.com/gpu.
- 0/4 nodes are available: 1 InsufficientResourceOnSingleGPU, 3 Insufficient nvidia.com/gpu.

在 CCE 控制台使用 GPU 资源，只需在创建负载时，勾选 GPU 配额，并指定使用的比例即可，如下图所示。

图8-1 使用 GPU

GPU 节点标签

创建 GPU 节点后，CCE 会给节点打上对应标签，如下所示，不同类型的 GPU 节点有不同标签。

```
$ kubectl get node -L accelerator
NAME          STATUS   ROLES    AGE    VERSION
ACCELERATOR
10.100.2.179   Ready    <none>   8m43s   v1.19.10-r0-CCE21.11.1.B006-21.11.1.B006
nvidia-t4
```

在使用 GPU 时，可以根据标签让 Pod 与节点亲和，从而让 Pod 选择正确的节点，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-test
  namespace: default
spec:
  replicas: 1
```

```
selector:
  matchLabels:
    app: gpu-test
template:
  metadata:
    labels:
      app: gpu-test
  spec:
    nodeSelector:
      accelerator: nvidia-t4
    containers:
    - image: nginx:perl
      name: container-0
      resources:
        requests:
          cpu: 250m
          memory: 512Mi
          nvidia.com/gpu: 1   # 申请 GPU 的数量
        limits:
          cpu: 250m
          memory: 512Mi
          nvidia.com/gpu: 1   # GPU 数量的使用上限
    imagePullSecrets:
    - name: default-secret
```

9 网络管理

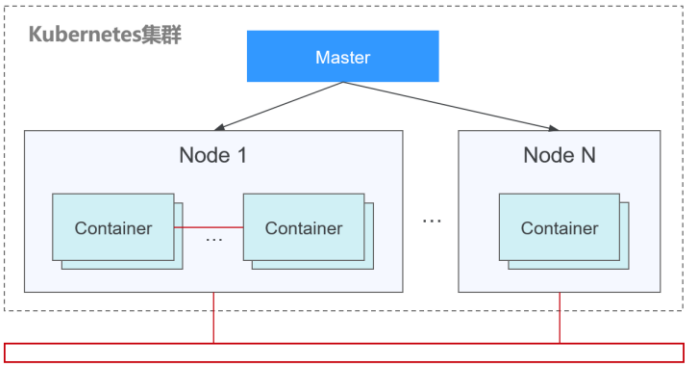
9.1 网络概述

关于集群的网络，可以从如下两个角度进行了解：

- 集群网络是什么样的：集群由多个节点构成，集群中又运行着 Pod（容器），每个 Pod 都需要访问，节点与节点、节点与 Pod、Pod 与 Pod 都需要访问。那集群中包含有哪些网络，各自的用处是什么，具体请参见集群网络构成。
- 集群中的 Pod 是如何访问的：访问 Pod 就是访问容器，也就是访问用户的业务，Kubernetes 提供 Service 和 Ingress 来解决 Pod 的访问问题。本章节根据用户使用场景总结了常见的网络访问场景，让您能够在不同使用场景下选择合适的使用方法。

集群网络构成

集群中节点都位于 VPC 中，节点使用 VPC 的网络，容器的网络是使用专门的网络插件来管理。



- **节点网络**

  节点网络为集群内主机（节点，图中的 Node）分配 IP 地址，您需要选择 VPC 中的子网用于 CCE 集群的节点网络。子网的可用 IP 数量决定了集群中可以创建节点数量的上限（包括 Master 节点和 Node 节点），集群中可创建节点数量还受容器网络的影响，在容器网络模型中会进一步说明。

- **容器网络**

为集群内容器分配 IP 地址。CCE 继承 Kubernetes 的 IP-Per-Pod-Per-Network 的容器网络模型，即每个 Pod 在每个网络平面下都拥有一个独立的 IP 地址，Pod 内所有容器共享同一个网络命名空间，集群内所有 Pod 都在一个直接连通的扁平网络中，无需 NAT 可直接通过 Pod 的 IP 地址访问。Kubernetes 只提供了如何为 Pod 提供网络的机制，并不直接负责配置 Pod 网络；Pod 网络的具体配置操作交由具体的容器网络插件实现。容器网络插件负责为 Pod 配置网络并管理容器 IP 地址。

当前 CCE 支持如下容器网络模型。

- 容器隧道网络：容器隧道网络在节点网络基础上通过隧道封装另构建的独立于节点网络平面的容器网络平面，CCE 集群容器隧道网络使用的封装协议为 VXLAN，后端虚拟交换机采用的是 openvswitch，VXLAN 是将以太网报文封装成 UDP 报文进行隧道传输。
- VPC 网络：VPC 网络采用 VPC 路由方式与底层网络深度整合，适用于高性能场景，节点数量受限于虚拟私有云 VPC 的路由配额。每个节点将会被分配固定大小的 IP 地址段。VPC 网络由于没有隧道封装的消耗，容器网络性能相对于容器隧道网络有一定优势。VPC 网络集群由于 VPC 路由中配置有容器网段与节点 IP 的路由，可以支持集群外直接访问容器实例等特殊场景。

不同容器网络模型，容器网络的性能、组网规模、适用场景各不相同，在容器网络模型对比章节，将会详细介绍不同容器网络模型的功能特性，了解这些有助于您选择容器网络模型。

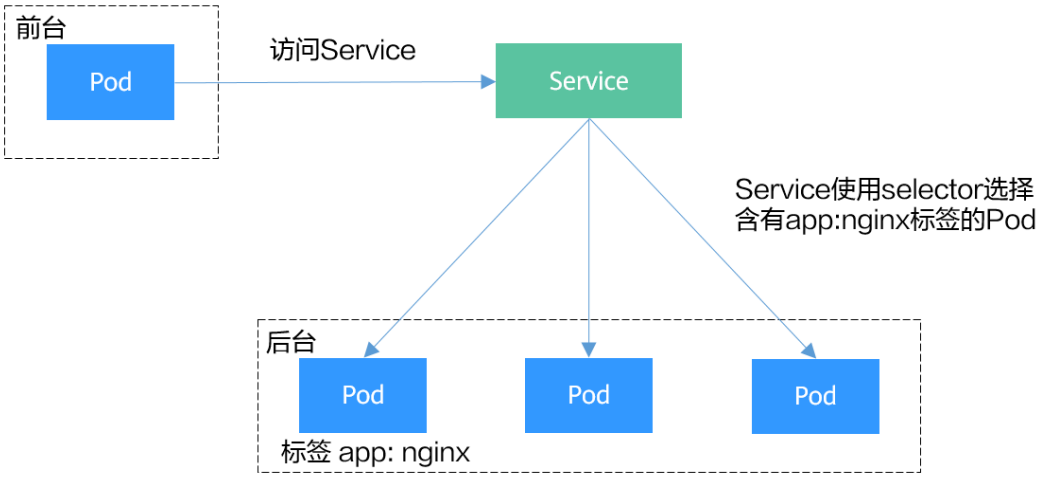
- **服务网络**

  服务（Service）是 Kubernetes 内的概念，每个 Service 都有一个固定的 IP 地址，在 CCE 上创建集群时，可以指定 Service 的地址段（即服务网段）。服务网段不能和节点网段、容器网段重叠。服务网段只在集群内使用，不能在集群外使用。

Service

Service 是用来解决 Pod 访问问题的。每个 Service 有一个固定 IP 地址，Service 将访问流量转发给 Pod，而且 Service 可以给这些 Pod 做负载均衡。

图9-1 通过 Service 访问 Pod



根据创建 Service 的类型不同，可分成如下模式：

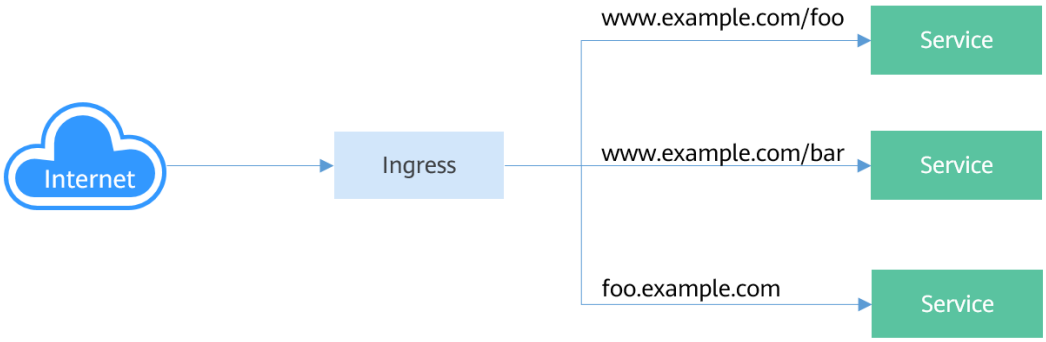
- ClusterIP：用于在集群内部互相访问的场景，通过 ClusterIP 访问 Service。
- NodePort：用于从集群外部访问的场景，通过节点上的端口访问 Service。
- LoadBalancer：用于从集群外部访问的场景，其实是 NodePort 的扩展，通过一个特定的 LoadBalancer 访问 Service，这个 LoadBalancer 将请求转发到节点的 NodePort，而外部只需要访问 LoadBalancer。

Service 的详细介绍请参见 Service 概述。

Ingress

Service 是基于四层 TCP 和 UDP 协议转发的，而 Ingress 可以基于七层的 HTTP 和 HTTPS 协议转发，可以通过域名和路径做到更细粒度的划分，如下图所示。

图9-2 Ingress-Service



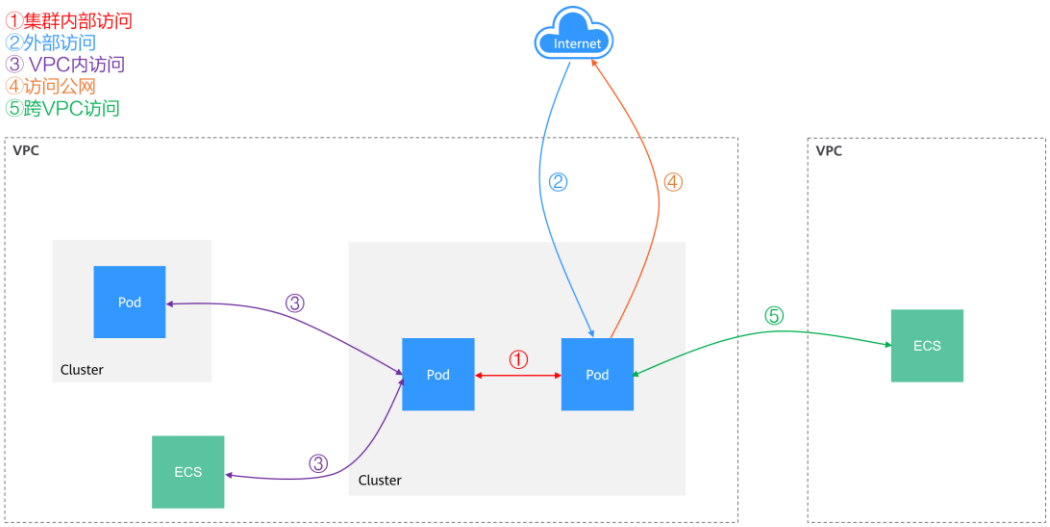
Ingress 的详细介绍请参见 Ingress 概述。

网络访问场景

负载网络访问可以分为如下几种场景。

- 集群内部访问：创建 ClusterIP 类型的 Service，通过 Service 访问负载。
- 外部访问负载：从外部访问负载推荐使用 Service（NodePort 类型或 LoadBalancer 类型）或 Ingress 访问。
  - 公网访问负载需要节点或 LoadBalancer 绑定公网 IP。
  - 内网访问负载通过节点或 LoadBalancer 内网 IP 即可。如果跨 VPC 需要通过对等连接等手段打通不同 VPC 网络。
- 负载访问外部：
  - 负载访问内网：负载访问内网地址，在不同容器网络模型下有不同的表现，需要注意在对端安全组放通容器网段，具体请参见容器如何访问 VPC 内部网络。
  - 负载访问公网：访问公网有几种方法可以实现，一是让容器所在节点绑定公网 IP（容器网络模型为 VPC 网络或容器隧道网络），另一个是通过 NAT 网关配置 SNAT 规则，具体请参见从容器访问公网。

图9-3 网络访问示意图



9.2 容器网络模型

9.2.1 容器网络模型对比

容器网络为集群内 Pod 分配 IP 地址并提供网络服务，CCE 支持如下几种网络模型，您可在创建集群时进行选择。

- 容器隧道网络
- VPC 网络
- 云原生网络 2.0

网络模型对比

主要介绍 CCE 所支持的网络模型，您可根据实际业务需求进行选择。

⚠ 注意

集群创建成功后，网络模型不可更改，请谨慎选择。

表9-1 网络模型对比

对比维度	容器隧道网络	VPC 网络	云原生网络 2.0
适用场景	• 一般容器业务场景。 • 对网络时延、带宽要求不是特别	• 对网络时延、带宽要求高。 • 容器与虚机 IP 互通，使用了微服务	• 对网络时延、带宽要求高，高性能场景。 • 容器与虚机 IP 互

对比维度	容器隧道网络	VPC 网络	云原生网络 2.0
	高的场景。	注册框架，如 Dubbo、CSE 等。	通，使用了微服务注册框架的，如 Dubbo、CSE 等。
核心技术	OVS	IPVlan，VPC 路由	VPC 弹性网卡/弹性辅助网卡
适用集群	CCE 集群	CCE 集群	CCE Turbo 集群
网络隔离	Pod 支持 Kubernetes 原生 NetworkPolicy	否	Pod 支持使用安全组隔离
ELB 直通容器	否	否	是
IP 地址管理	• 容器网段单独分配 • 节点维度划分地址段，动态分配（地址段分配后可动态增加）	• 容器网段单独分配 • 节点维度划分地址段，静态分配（节点创建完成后，地址段分配即固定，不可更改）	容器网段从 VPC 子网划分，无需单独分配
网络性能	基于 vxlan 隧道封装，有一定性能损耗。	无隧道封装，跨节点通过 VPC 路由器转发，性能好，媲美主机网络。	容器网络与 VPC 网络融合，性能无损耗
组网规模	最大可支持 2000 节点	默认支持 200 节点，受限于 VPC 路由表能力。 VPC 网络模式下，集群每添加一个节点，会在 VPC 的路由表中添加一条路由（包括默认路由表和自定义路由表），因此集群本身规模受 VPC 路由表上限限制。	最大可支持 2000 节点

须知

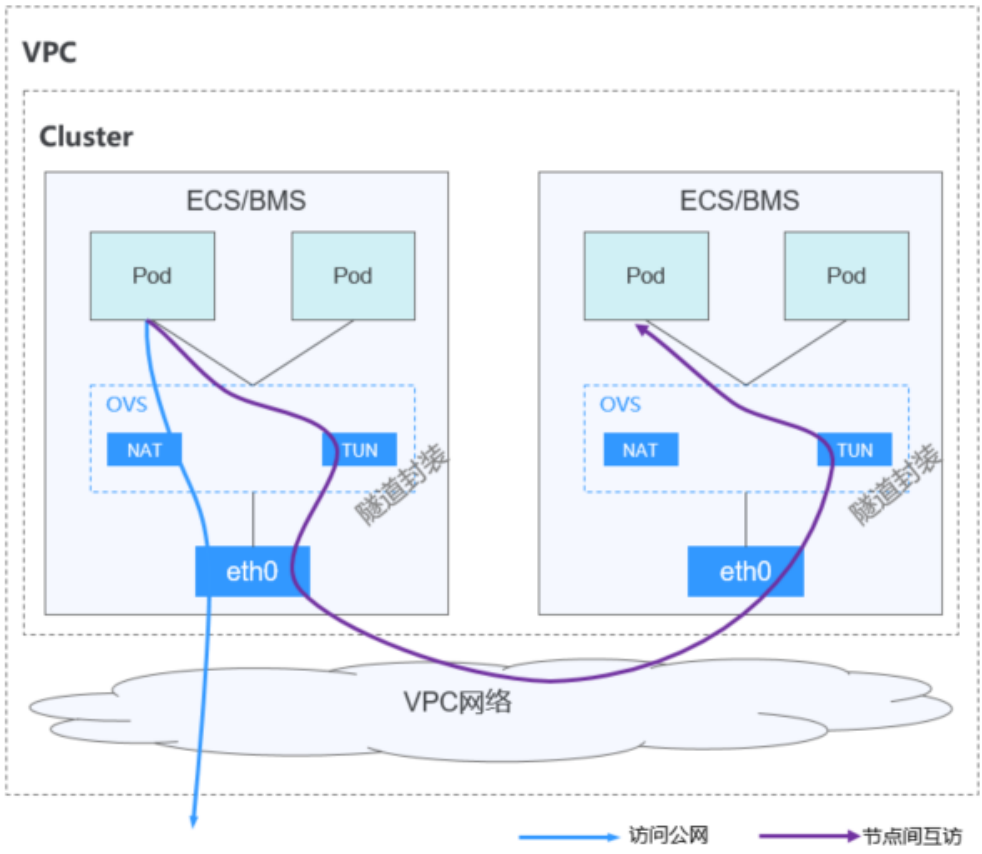
1. VPC 路由网络集群实际支持规模受限于 VPC 的路由表路由条目配额，创建前请提前评估集群规模。
2. 云原生网络 2.0 集群实际支持规模受限于网络平面选择的 VPC 子网网段大小，创建前请提前评估集群规模。
3. VPC 路由网络默认支持容器与同一 VPC 的虚拟机直接互访，与其他 VPC 的主机在配置对等连接策略后可以支持直接互访。此外，云专线/VPN 等混合组网场景在合理规划后可以支持对端直接与容器互访。
4. 请勿在创建集群后修改 VPC 侧的主网段掩码大小，若强行修改会导致 VPC 集群新建节点的部分网络功能异常。

9.2.2 容器隧道网络

容器隧道网络模型

容器隧道网络在节点网络基础上通过隧道封装构建的独立于节点网络平面的容器网络平面，CCE 集群容器隧道网络使用的封装协议为 VXLAN，后端虚拟交换机采用的是 openvswitch，VXLAN 是将以太网报文封装成 UDP 报文进行隧道传输。容器隧道网络具有付出少量隧道封装性能损耗，即可获得通用性强、互通性强、高级特性支持全面（例如 NetworkPolicy 网络隔离）的优势，可以满足大多数性能要求不高的场景。

图9-4 容器隧道网络



说明如下：

- 节点内 Pod 间通信：同节点的 Pod 间通信直接通过本节点的 ovs 网桥直接转发。
- 跨节点 Pod 间通信：所有跨节点 Pod 间的通信通过 ovs 隧道网桥进行封装后，转发到对端节点上。

优缺点

优点

- 容器网络和节点网络解耦，不受 VPC 配额规格、响应速度的限制（如 VPC 路由条目数、弹性网卡数、创建速度限制）
- 支持网络隔离，具体请参见网络策略（NetworkPolicy）
- 支持带宽限制
- 支持大规模组网

缺点

- 由于隧道封装，网络问题排查难度较大，整体性能较低
- 无法直接利用 VPC 提供的负载均衡、安全组等能力
- 不支持外部网络与容器 IP 直通

应用场景

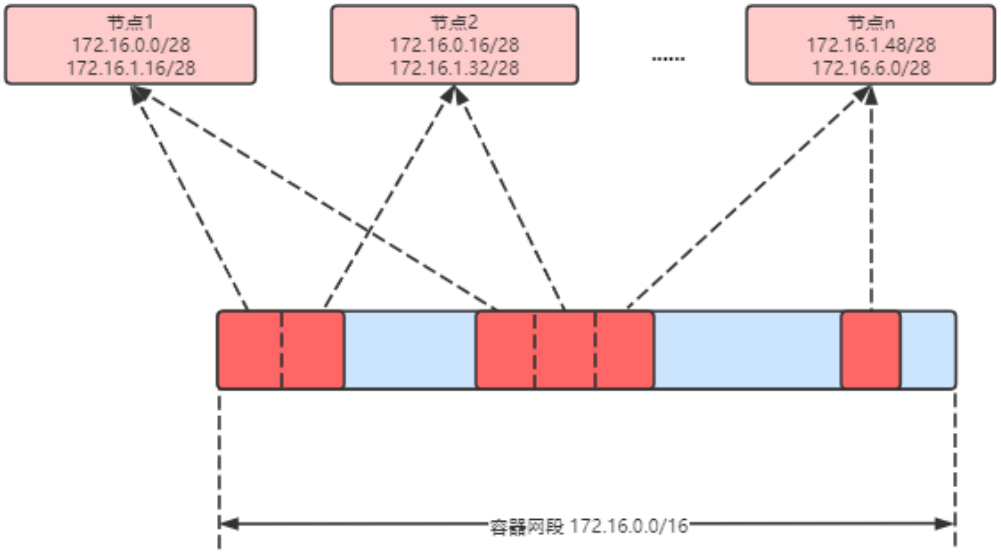
- 对性能要求不高：由于需要额外的 VXLAN 隧道封装，相对于另外两种容器网络模式，性能存在一定的损耗。大概有 5%-15%的性能损失。所以容器隧道网络适用于对性能要求不是特别高的业务场景，比如：web 应用、访问量不大的数据中台、后台服务等。
- 大规模组网：相比 VPC 路由网络受限于 VPC 路由条目配额的限制，容器隧道网络没有网络基础设施的任何限制；同时容器隧道网络把广播域控制到了节点级别，容器隧道网络最大可支持 2000 节点规模。

容器 IP 地址管理

容器隧道网络按如下规则分配容器 IP：

- 容器网段需单独分配，与节点网段无关
- 节点维度划分地址段，集群的所有节点从容器网段中分配一个或多个固定大小（默认 16）的 IP 网段
- 当节点上的 IP 地址使用完后，可再次申请分配一个新的 IP 网段
- 容器网段依次循环分配 IP 网段给新增节点或存量节点
- 调度到节点上的 Pod 依次循环从分配给节点的一个或多个 IP 网段内分配 IP 地址

图9-5 容器隧道网络 IP 地址分配



按如上 IP 分配，容器隧道网络的集群最多能创建节点数量 = 容器网段 IP 数量 ÷ 节点从容器网段中一次分配的 IP 网段大小（默认为 16）

比如容器网段为 172.16.0.0/16，则 IP 数量为 65536，一次分配 16，则最多可创建节点数量为 65536/16=4096。当然，这是一种极端情况，如果创建 4096 个节点，则每个节点最多只能创建 16 个 Pod，因为给每个节点只分配了 16 个 IP 的网段。另外集群能创建多少节点，还受节点网络和集群规模的影响。

图9-6 网络模型选择（创建集群时配置）

网段规划建议

在集群网络构成中介绍集群中网络地址可分为节点网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- **三个网段不能重叠**，否则会导致冲突。且集群所在 VPC 下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
- 保证**每个网段有足够的 IP 地址可用**。
  - 节点网段的 IP 地址要与集群规模相匹配，否则会因为 IP 地址不足导致无法创建节点。
  - 容器网段的 IP 地址要与业务规模相匹配，否则会因为 IP 地址不足导致无法创建 Pod。每个节点上可以创建多少 Pod 还与其他参数设置相关，具体请参见节点最多可以创建多少个 Pod。

容器隧道网络访问示例

创建一个容器隧道网络的集群。在集群中创建一个 Deployment。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
```

```
      - name: container-0
        image: 'nginx:perl'
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
    imagePullSecrets:
      - name: default-secret
```

创建后查看 Pod。

```
$ kubectl get pod -owide
NAME                   READY   STATUS    RESTARTS   AGE    IP         NODE
NOMINATED NODE   READINESS GATES
example-5bdc5699b7-5rvq4   1/1    Running   0        3m28s   10.0.0.20   192.168.0.42
<none>         <none>
example-5bdc5699b7-984j9   1/1    Running   0        3m28s   10.0.0.21   192.168.0.42
<none>         <none>
example-5bdc5699b7-lfxkm   1/1    Running   0        3m28s   10.0.0.22   192.168.0.42
<none>         <none>
example-5bdc5699b7-wjcmg   1/1    Running   0        3m28s   10.0.0.52   192.168.0.64
<none>         <none>
```

此时如果在集群同 VPC 下集群外部直接访问 Pod 的 IP，会发现访问不通，这就是容器隧道网络的特性，不支持外部网络与容器 IP 直通。

而在集群内部节点或 Pod 内，都能正常访问 Pod，如下进入到容器中直接访问 Pod 能够正常访问。

```
$ kubectl exec -it example-5bdc5699b7-5rvq4 -- curl 10.0.0.21
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
   body {
       width: 35em;
       margin: 0 auto;
       font-family: Tahoma, Verdana, Arial, sans-serif;
   }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
```

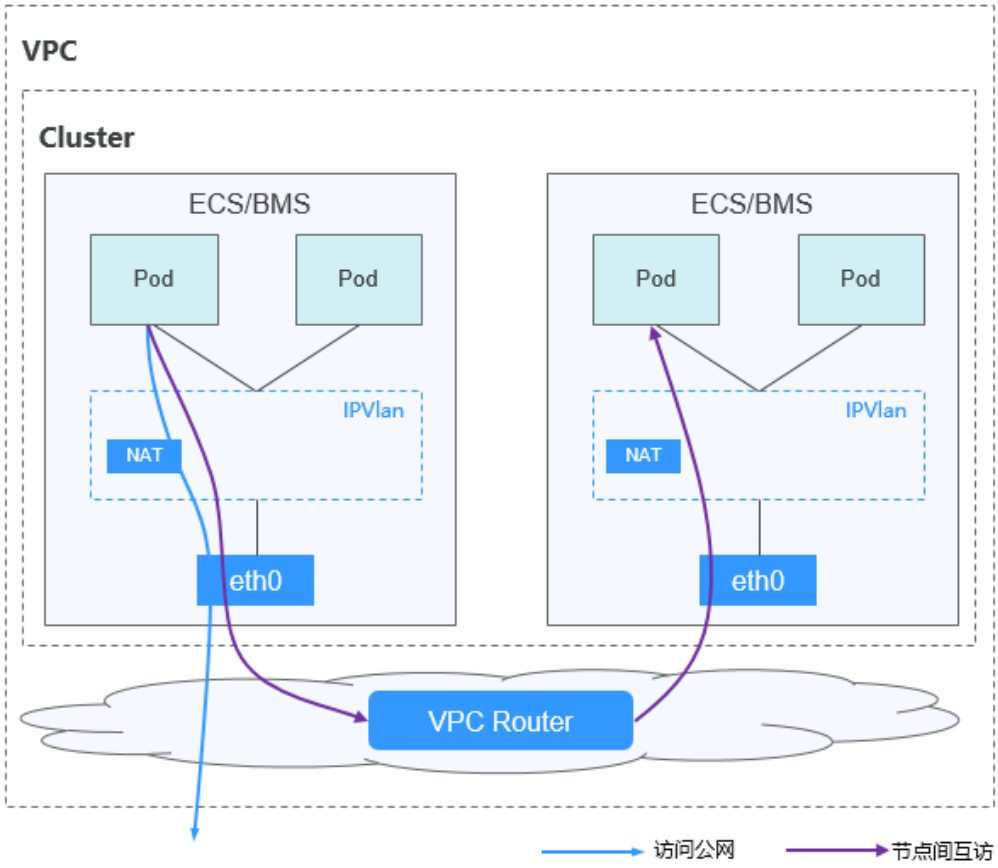
```
</body>
</html>
```

9.2.3 VPC 网络

VPC 网络模型

VPC 网络采用 VPC 路由方式与底层网络深度整合，适用于高性能场景，节点数量受限于虚拟私有云 VPC 的路由配额。每个节点将会被分配固定大小的 IP 地址段。VPC 网络由于没有隧道封装的消耗，容器网络性能相对于容器隧道网络有一定优势。VPC 网络集群由于 VPC 路由中配置有容器网段与节点 IP 的路由，可以支持集群外直接访问容器实例等特殊场景。

图9-7 VPC 网络



说明如下：

- 节点内 Pod 间通信：ipvlan 子接口分配给节点上的 Pod，同节点的 Pod 间通信直接通过 ipvlan 直接转发。
- 跨节点 Pod 间通信：所有跨节点 Pod 间的通信通过默认路由到默认网关，借助 VPC 的路由转发能力，转发到对端节点上。

优缺点

优点

- 由于没有隧道封装，网络问题易排查、性能较高
- 支持 VPC 内的外部网络与容器 IP 直通

缺点

- 节点数量受限于虚拟私有云 VPC 的路由配额
- 每个节点将会被分配固定大小的 IP 地址段，存在一定的容器网段 IP 地址浪费
- Pod 无法直接利用 EIP、安全组等能力

应用场景

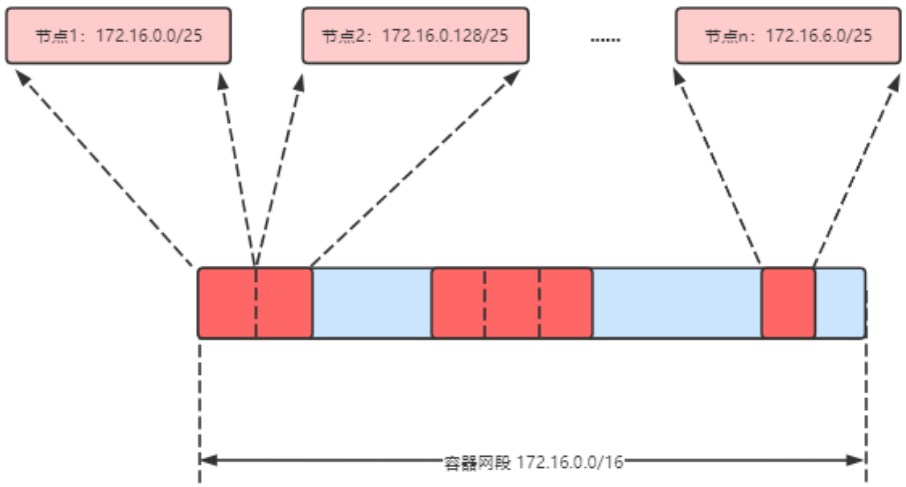
- 性能要求高：由于没有额外的隧道封装，相比于容器隧道网络模式，性能接近于 VPC 网络的性能，所以适用于对性能要求较高的业务场景，比如：AI 计算、大数据计算等。
- 中小规模组网：由于 VPC 路由网络受限于 VPC 路由表条目配额的限制，当前默认最大只支持 200 个节点，如果有更大规模的需求，可提升 VPC 路由表条目配额。

容器 IP 地址管理

VPC 网络按如下规则分配容器 IP：

- 容器网段需单独分配
- 节点维度划分地址段，集群的所有节点从容器网段中分配一个固定大小（用户自己配置）的 IP 网段
- 容器网段依次循环分配 IP 网段给新增节点
- 调度到节点上的 Pod 依次循环从分配给节点的 IP 网段内分配 IP 地址

图9-8 VPC 网络 IP 地址管理



按如上 IP 分配，VPC 网络的集群最多能创建节点数量 = 容器网段 IP 数量 ÷ 节点从容器网段中分配 IP 网段的 IP 数量

比如容器网段为 172.16.0.0/16，则 IP 数量为 65536，节点分配容器网段掩码为 25，也就是每个节点容器 IP 数量为 128，则最多可创建节点数量为 65536/128=512。当然，集群能创建多少节点，还受节点网络和集群规模的影响。

图9-9 容器网段配置（创建集群时配置）

网段规划建议

在集群网络构成中介绍集群中网络地址可分为节点网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- **三个网段不能重叠**，否则会导致冲突。且集群所在 VPC 下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
- 保证**每个网段有足够的 IP 地址可用**。
  - 节点网段的 IP 地址要与集群规模相匹配，否则会因为 IP 地址不足导致无法创建节点。
  - 容器网段的 IP 地址要与业务规模相匹配，否则会因为 IP 地址不足导致无法创建 Pod。每个节点上可以创建多少 Pod 还与其他参数设置相关，具体请参见节点最多可以创建多少个 Pod。

例如集群规模为 200 节点，容器网络模型为 VPC 网络。

则此时选择节点子网的可用 IP 数量需要超过 200，否则会因为 IP 地址不足导致无法创建节点。

容器网段为 10.0.0.0/16，可用 IP 数量为 65536，如容器 IP 地址管理中所述，VPC 网络 IP 分配是分配固定大小的网段（使用掩码实现，确定每个节点最多分配多少容器 IP），例如上限为 128，则此时集群最多支撑 65536/128=512 个节点，然后去掉 Master 节点数量，最终结果就是 509 个。

图9-10 容器网段配置（创建集群时配置）

VPC 网络访问示例

创建一个 VPC 网络的集群。集群有一个 Node 节点。

```
$ kubectl get node
NAME          STATUS   ROLES    AGE   VERSION
192.168.0.99   Ready    <none>   9d    v1.17.17-r0-CCE21.6.1.B004-17.37.5
```

查看 VPC 的路由表，会看到如下一条路由，目的地址 172.16.0.0/25 是分配给节点的容器网段，下一跳指向对应的节点，当访问容器 IP 时，VPC 路由就会转发给下一跳的节点。这印证了前面说的 VPC 网络模型使用 VPC 的路由。

图9-11 路由

在集群中创建一个 Deployment。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: container-0
          image: 'nginx:perl'
      imagePullSecrets:
        - name: default-secret
```

然后查看 Pod。

```
$ kubectl get pod -owide
NAME                   READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
example-86b9779494-l8qrw   1/1     Running   0          14s   172.16.0.6   192.168.0.99
<none>           <none>
example-86b9779494-svs8t   1/1     Running   0          14s   172.16.0.7   192.168.0.99
<none>           <none>
example-86b9779494-x8kl5   1/1     Running   0          14s   172.16.0.5   192.168.0.99
<none>           <none>
example-86b9779494-zt627   1/1     Running   0          14s   172.16.0.8   192.168.0.99
<none>           <none>
```

此时如果在集群同 VPC 下集群外部直接访问 Pod 的 IP，会发现可以访问，这就是 VPC 网络的特性，支持外部网络与通过 IP 地址直接访问容器。

而在集群内部节点或 Pod 内，都能正常访问 Pod，如下进入到容器中直接访问 Pod 能够正常访问。

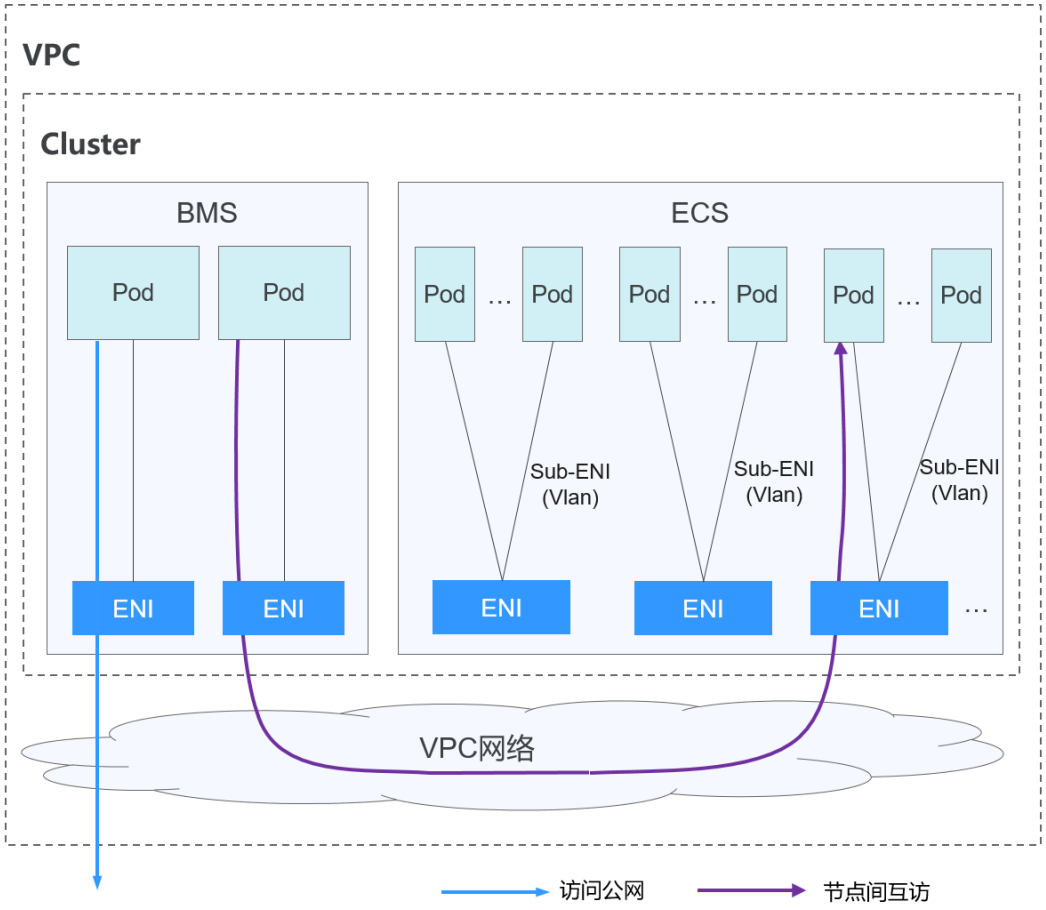
```
$ kubectl exec -it example-86b9779494-l8qrw -- curl 172.16.0.7
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
   body {
       width: 35em;
       margin: 0 auto;
       font-family: Tahoma, Verdana, Arial, sans-serif;
   }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

9.2.4 云原生网络 2.0

云原生网络 2.0 网络模型

云原生网络 2.0 是 CCE 的新一代容器网络模型，深度整合了虚拟私有云 VPC 的弹性网卡（Elastic Network Interface，简称 ENI）和辅助弹性网卡（Sub Network Interface，简称 Sub-ENI）的能力，直接从 VPC 网段内分配容器 IP 地址，支持 ELB 直通容器，绑定安全组，绑定弹性公网 IP，享有高性能。

图9-12 云原生网络 2.0



说明如下：

- 物理机节点上 Pod 使用 ENI 网卡；ECS 节点上 Pod 使用 Sub-ENI 网卡，Sub-ENI 网卡通过 VLAN 子接口挂载在 ENI 上。
- 节点内 Pod 间通信：直接通过 VPC 的弹性网卡/弹性辅助网卡进行流量转发。
- 跨节点 Pod 间通信：直接通过 VPC 的弹性网卡/弹性辅助网卡进行流量转发。

约束与限制

仅 CCE Turbo 集群支持使用云原生网络 2.0。

优缺点

优点

- 容器网络直接使用的 VPC，网络问题易排查、性能最高。
- 支持 VPC 内的外部网络与容器 IP 直通。
- 可直接利用 VPC 提供的负载均衡、安全组、弹性公网 IP 等能力。

缺点

由于容器网络直接使用的 VPC，消耗 VPC 的地址空间，创建集群前需要合理规划好容器网段。

适用场景

- 性能要求高，需要使用 VPC 其他网络能力的场景：由于云原生网络 2.0 直接使用的 VPC 网络，性能与 VPC 网络的性能几乎一致，所以适用于对带宽、时延要求极高的业务场景，比如：线上直播、电商秒杀等。
- 大规模组网：云原生网络 2.0 当前最大可支持 2000 个 ECS 节点，10 万个容器。

容器 IP 地址管理

云原生网络 2.0 下的 BMS 节点和 ECS 节点分别使用的是弹性网卡和辅助弹性网卡：

- Pod 的 IP 地址从配置给容器网络的 VPC 子网上直接分配，无需为节点分配一个单独的小网段。
- ECS 节点添加到集群中，先绑定用于承载辅助弹性网卡的弹性网卡，待弹性网卡绑定完成后，即可绑定辅助弹性网卡。
- ECS 节点上绑定的弹性网卡数：**该节点最多可绑定的辅助弹性网卡数/64**，向上取整。
- ECS 节点上绑定的总网卡数：**用于承载辅助弹性网卡的弹性网卡数+当前 Pod 使用的辅助弹性网卡数+预热的辅助弹性网卡数**。
- BMS 节点上绑定的网卡数：**当前 Pod 使用的弹性网卡数+预热的弹性网卡数**。
- Pod 创建时，优先从节点的预热网卡池中随机分配一个可用的网卡。
- Pod 删除时，网卡释放回节点的预热网卡池。
- 节点删除时，将释放节点上所有已绑定的网卡（弹性网卡释放回集群预申请的网卡池，辅助弹性网卡直接删除）。

云原生 2.0 网络目前支持两种网卡预热策略：**节点容器网卡动态预热策略**和**节点绑定容器网卡数总量高低水位策略（废弃中）**。使用场景如下表所示：

表9-2 容器网卡预热策略对比表

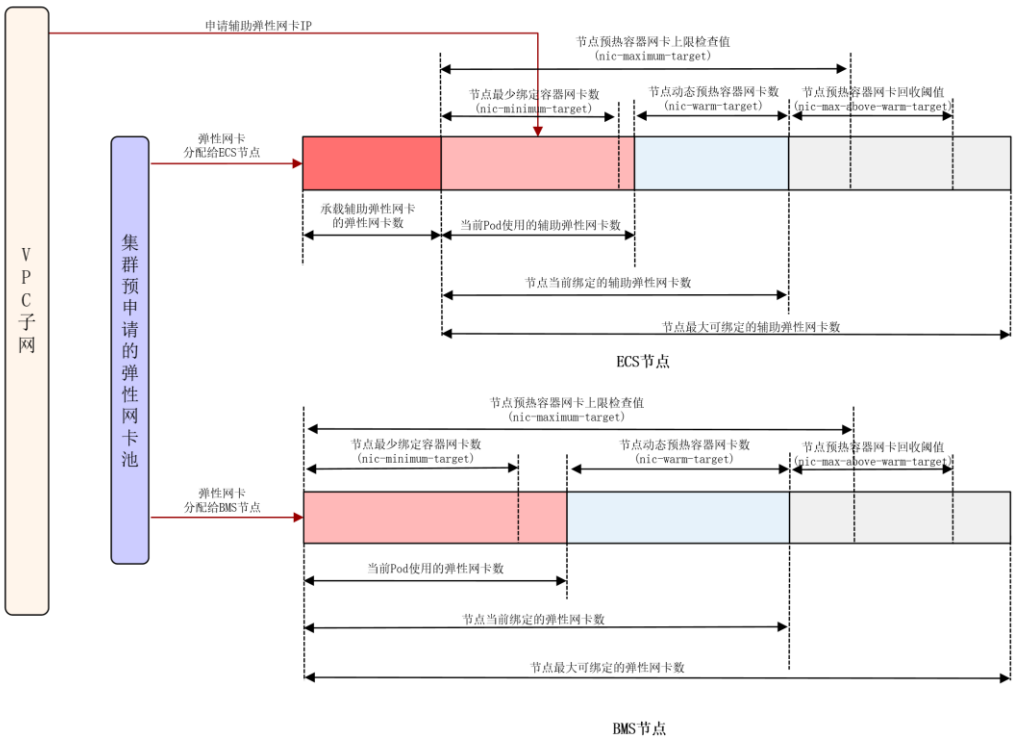
容器网卡预热策略	节点容器网卡动态预热策略（默认策略）	节点绑定容器网卡数总量高低水位策略（废弃中）
管理策略	节点最少绑定容器网卡数（nic-minimum-target）：保障节点最少有多少张容器网卡绑定在节点上（未被 Pod 使用+已被 Pod 使用） 节点预热容器网卡上限检查值（nic-maximum-target）：当节点绑定的容器网卡数超过该值，不再主动预热容器网卡 节点动态预热容器网卡数：当 Pod 使用完节点最少绑定容器网卡数（nic-minimum-	节点绑定容器网卡数低水位：保障节点至少会绑定多少张网卡（未被 Pod 使用+已被 Pod 使用） 节点绑定容器网卡数高水位：保障节点至多会绑定多少张网卡，超过该值会尝试解绑未被使用的空闲网卡

容器网卡预热策略	节点容器网卡动态预热策略（默认策略）	节点绑定容器网卡数总量高低水位策略（废弃中）
	target）后，会始终额外预热多少张容器网卡 节点预热容器网卡回收阈值(nic-max-above-warm-target)：只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值时，才会触发预热容器网卡的解绑回收	
适用场景	在尽可能提高 IP 资源利用率的前提下，尽可能加快 Pod 的启动速度，适用于容器网段 IP 地址数紧张的场景 通过合理配置上述四个参数，可适用于各种业务场景。	适用于容器网段 IP 地址数充足，且节点上 Pod 数变化剧烈，但固定在某个范围的场景

说明

- 1.19.16-r2、1.21.5-r0、1.23.3-r0 到 1.19.16-r4、1.21.7-r0、1.23.5-r0 之间的集群版本只支持 nic-minimum-target 和 nic-warm-target 两个容器网卡动态预热参数配置，绑定网卡数总量高低水位配置优先级高于容器网卡动态预热配置。
- 1.19.16-r4、1.21.7-r0、1.23.5-r0、1.25.1-r0 及以上集群版本支持全部四个容器网卡动态预热参数配置，容器网卡动态预热配置优先级高于绑定网卡数总量高低水位配置。

图9-13 节点容器网卡动态预热策略



针对节点容器网卡动态预热策略，CCE 提供了四个参数配置，您可以根据业务规划，集群规模以及节点上可绑定的网卡数，合理设置这四个参数。

表9-3 容器网卡动态预热参数

容器网卡动态预热参数	默认值	参数说明	配置建议
节点最少绑定容器网卡数(nic-minimum-target)	10	保障节点最少有多少张容器网卡绑定在节点上，支持数值跟百分比两种配置方式。 • 数值配置：参数值需为正整数。例如 10，表示节点最少有 10 张容器网卡绑定在节点上。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。 • 百分比配置：参数值范围为 1%-100%。例如 10%，如果节点容器网卡配额 128，表示节点最少有 12 张（向下取整）容器网卡绑定在节点上。 建议 nic-minimum-target 与 nic-maximum-target 为同类型的配置方式（同采用数值配置或同采用百分比配	建议配置为大部分节点平时日常运行的 Pod 数。

容器网卡动态预热参数	默认值	参数说明	配置建议
		置）。	
节点预热容器网卡上限检查值(nic-maximum-target)	0	当节点绑定的容器网卡数超过节点预热容器网卡上限检查值(nic-maximum-target)，不再主动预热容器网卡。 当该参数大于等于节点最少绑定容器网卡数(nic-minimum-target)时，则开启预热容器网卡上限值检查；反之，则关闭预热容器网卡上限值检查。支持数值跟百分比两种配置方式。 • 数值配置：参数值需为正整数。例如0，表示关闭预热容器网卡上限值检查。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。 • 百分比配置：参数值范围为 1%-100%。例如 50%，如果节点容器网卡配额 128，表示节点预热容器网卡上限检查值 64（向下取整）。 建议 nic-minimum-target 与 nic-maximum-target 为同类型的配置方式（同采用数值配置或同采用百分比配置）。	建议配置为大部分节点平时最多运行的 Pod 数。
节点动态预热容器网卡数(nic-warm-target)	2	当 Pod 使用完节点最少绑定容器网卡数(nic-minimum-target)后，会始终额外预热多少张容器网卡，只支持数值配置。 当 节点动态预热容器网卡数(nic-warm-target) + 节点当前绑定的容器网卡数 大于 节点预热容器网卡上限检查值(nic-maximum-target) 时，只会预热 nic-maximum-target 与节点当前绑定的容器网卡数的差值。	建议配置为大部分节点日常 10s 内会瞬时弹性扩容的 Pod 数。
节点预热容器网卡回收阈值(nic-max-above-warm-target)	2	只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值 时，才会触发预热容器网卡的解绑回收。只支持数值配置。 • 调大此值会减慢空闲容器网卡的回收，加快 Pod 的启动速度，但会降低 IP 地址的利用率，特别是在 IP 地址紧张的场景，**请谨慎调大**。 • 调小此值会加快空闲容器网卡的回收，提高 IP 地址的利用率，但在瞬时大量 Pod 激增的场景，部分 Pod 启动会稍微变慢。	建议配置为大部分节点日常在分钟级时间范围内会频繁弹性扩容缩容的 Pod 数 - 大部分节点日常 10s 内会瞬时弹性扩容的 Pod 数。

说明

上述容器网卡动态预热参数支持集群级别的全局配置和节点池级别的差异化配置，其中节点池级别的容器网卡动态预热配置优先级高于集群级别的容器网卡动态预热配置。

容器网络组件会为每个节点维护一个可弹性伸缩的预热容器网卡池 ，定时（约 10s 一次）检测并计算**需要绑定的预热容器网卡数**或**需要解绑的空闲容器网卡数** ：

- **需要绑定的预热容器网卡数 = min(nic-maximum-target - 当前绑定的容器网卡总数，max(nic-minimum-target- 当前绑定的容器网卡总数，nic-warm-target - 当前空闲的容器网卡数))**
- **需要解绑的空闲容器网卡数 = min(当前空闲的容器网卡数 - nic-warm-target- nic-max-above-warm-target，当前绑定的容器网卡总数 - nic-minimum-target)**

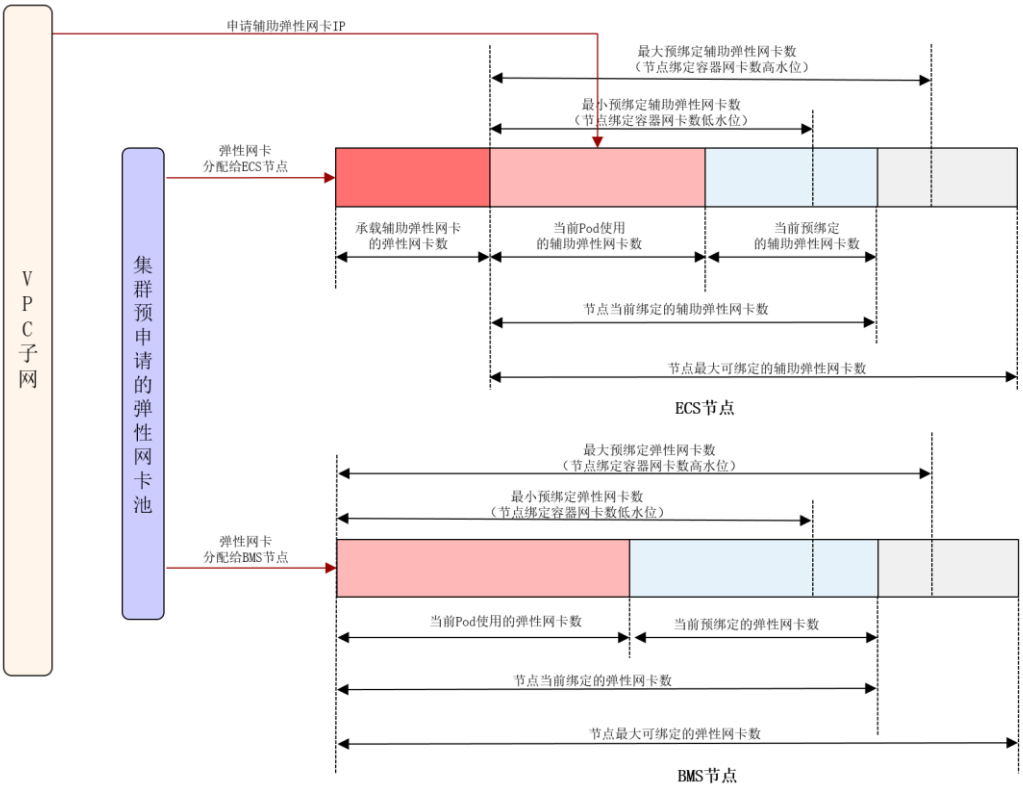
节点上当前预热的容器网卡数稳态后会维持在以下区间内：

- **当前预热的容器网卡数区间最小值 = min(max(nic-minimum-target- 当前绑定的容器网卡总数，nic-warm-target), nic-maximum-target - 当前绑定的容器网卡总数)**
- **当前预热的容器网卡数区间最大值 = max(nic-warm-target+ nic-max-above-warm-target, 当前绑定的容器网卡总数 - nic-minimum-target)**

Pod 创建时，优先从节点的预热容器网卡池中顺序分配（最早未被使用的）一张空闲的容器网卡，如没有可用的空闲网卡，会新创建一张网卡（辅助弹性网卡）或 新绑定一张网卡（弹性网卡）以分配给该 Pod。

Pod 删除时，对应的容器网卡先释放回节点的预热容器网卡池，2 分钟冷却时间内可供下一个 Pod 循环使用，超过 2 分钟冷却时间后且节点预热容器网卡池计算出需要释放该容器网卡，才会释放该容器网卡。

图9-14 节点绑定容器网卡数总量高低水位策略



针对总量高低水位算法，CCE 提供了一个配置参数，您可以根据业务规划，集群规模以及节点上可绑定的网卡数，合理设置这个参数：

- 节点绑定容器网卡数低水位：默认为 0，保障节点至少会绑定多少张网卡（未被 Pod 使用+已被 Pod 使用）。ECS 节点预绑定低水位网卡数=节点绑定网卡数低水位*节点总辅助弹性网卡数；BMS 节点预绑定低水位网卡数=节点绑定网卡数低水位*节点总弹性网卡数。
- 节点绑定容器网卡数高水位：默认为 0，保障节点至多会绑定多少张网卡，超过该值会尝试解绑未被使用的空闲网卡。ECS 节点预绑定高水位网卡数=节点绑定网卡数高水位*节点总辅助弹性网卡数；BMS 节点预绑定高水位网卡数=节点绑定网卡数高水位*节点总弹性网卡数。

容器网络组件会为每个节点维护一个可弹性伸缩的容器网卡池：

- 当**已绑定容器网卡数量（Pod 使用的容器网卡数+预绑定的容器网卡数）< 预绑定低水位容器网卡数**时，会绑定网卡直到节点上**已绑定容器网卡数量（Pod 使用的容器网卡数+预绑定的容器网卡数）=预绑定低水位容器网卡数**。
- 当**已绑定容器网卡数量（Pod 使用的容器网卡数+预绑定的容器网卡数）> 预绑定高水位容器网卡数**，且 **节点预绑定的容器网卡数>0** 时，会定时释放预绑定的容器网卡（超过 2 分钟未被使用的空闲网卡），直到 **Pod 使用的容器网卡数+预绑定的容器网卡数=节点预绑定高水位容器网卡数** 或 **Pod 使用的容器网卡数 > 节点预绑定高水位容器网卡数** 且 **节点预绑定的容器网卡数=0。**

网段规划建议

在集群网络构成中介绍集群中网络地址可分为节点网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- **三个网段不能重叠**，否则会导致冲突。且集群所在 VPC 下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
- 保证**每个网段有足够的 IP 地址可用**。
  - 节点网段的 IP 地址要与集群规模相匹配，否则会因为 IP 地址不足导致无法创建节点。
  - 容器网段的 IP 地址要与业务规模相匹配，否则会因为 IP 地址不足导致无法创建 Pod。

云原生网络 2.0 模型下，由于容器网段与节点网段共同使用 VPC 下的网络地址，建议容器子网与节点子网不要使用同一个子网，否则容易出现 IP 资源不足导致容器或节点创建失败的情况。

另外云原生网络 2.0 模型下容器网段支持在创建集群后增加子网，扩展可用 IP 数量，此时需要注意增加的子网不要与容器网段其他子网存在网络冲突。

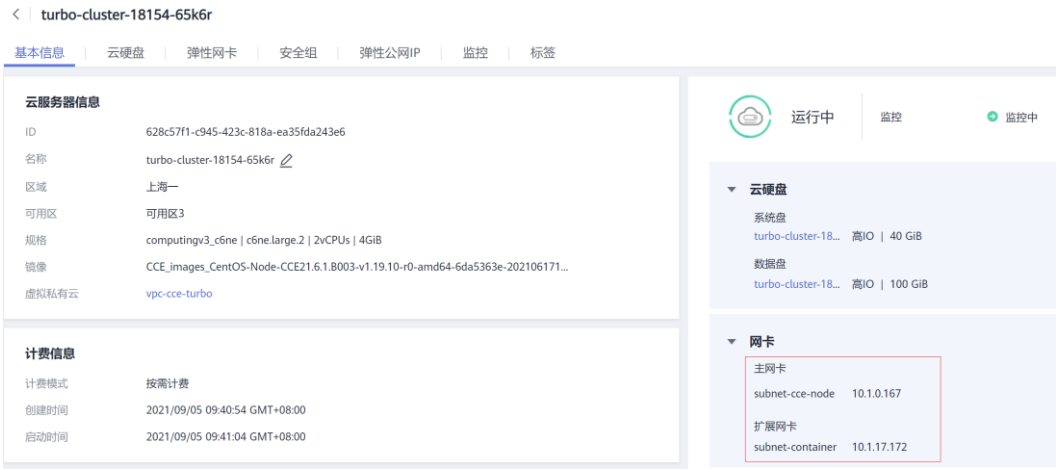
图9-15 网段配置（创建集群时配置）

云原生网络 2.0 访问示例

创建一个 CCE Turbo 集群，集群包含 3 个 ECS 节点。

进入其中一个节点，可以看到节点有一个主网卡和扩展网卡，这两个网卡都是弹性网卡，其中扩展网卡是属于容器网络网段，用于给 Pod 挂载辅助弹性网卡 Sub-ENI。

图9-16 节点网卡



在集群中创建一个 Deployment。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: default
spec:
  replicas: 6
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: container-0
          image: 'nginx:perl'
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      imagePullSecrets:
        - name: default-secret
```

创建后查询 Pod。

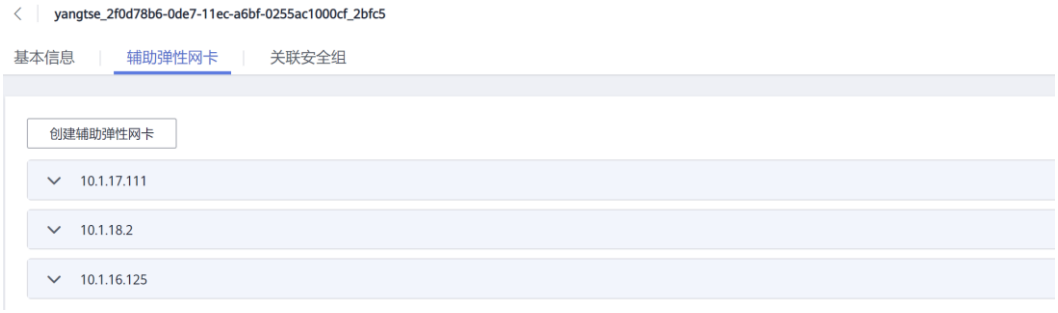
```
$ kubectl get pod -owide
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
example-5bdc5699b7-54v7g   1/1     Running   0          7s    10.1.18.2    10.1.0.167
<none>           <none>
example-5bdc5699b7-6dzx5   1/1     Running   0          7s    10.1.18.216  10.1.0.186
```

```
<none>          <none>
example-5bdc5699b7-gq7xs   1/1    Running   0         7s    10.1.16.63   10.1.0.144
<none>          <none>
example-5bdc5699b7-h9rvb   1/1    Running   0         7s    10.1.16.125  10.1.0.167
<none>          <none>
example-5bdc5699b7-s9fts   1/1    Running   0         7s    10.1.16.89   10.1.0.144
<none>          <none>
example-5bdc5699b7-swq6q   1/1    Running   0         7s    10.1.17.111  10.1.0.167
<none>          <none>
```

这里 Pod 的 IP 都是 Sub-ENI，挂载在节点的 ENI 上（扩展网卡）。

例如 10.1.0.167 节点对应的扩展网卡是 10.1.17.172。在弹性网卡控制台上可以看到 10.1.17.172 这块扩展网卡挂载 3 个 Sub-ENI，正是 Pod 的 IP。

图9-17 Pod 网卡



在 VPC 中直接访问 Pod 的 IP，能够正常访问。

9.3 Service

9.3.1 Service 概述

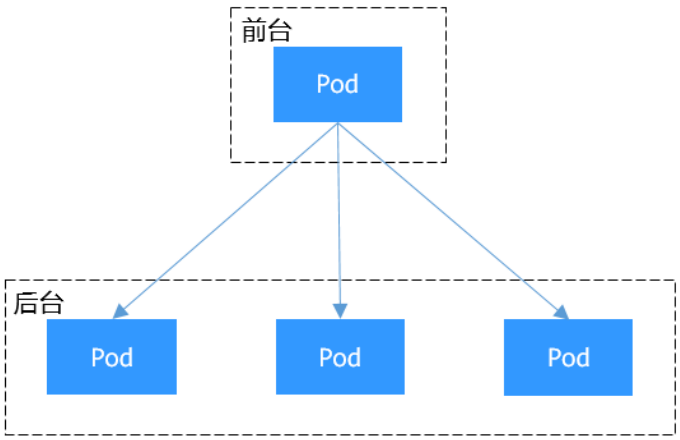
直接访问 Pod 的问题

Pod 创建完成后，如何访问 Pod 呢？直接访问 Pod 会有如下几个问题：

- Pod 会随时被 Deployment 这样的控制器删除重建，那访问 Pod 的结果就会变得不可预知。
- Pod 的 IP 地址是在 Pod 启动后才被分配，在启动前并不知道 Pod 的 IP 地址。
- 应用往往都是由多个运行相同镜像的一组 Pod 组成，逐个访问 Pod 也变得不现实。

举个例子，假设有这样一个应用程序，使用 Deployment 创建了前台和后台，前台会调用后台做一些计算处理。后台运行了 3 个 Pod，这些 Pod 是相互独立且可被替换的，当 Pod 出现状况被重建时，新建的 Pod 的 IP 地址是新 IP，前台的 Pod 无法直接感知。

图9-18 Pod 间访问

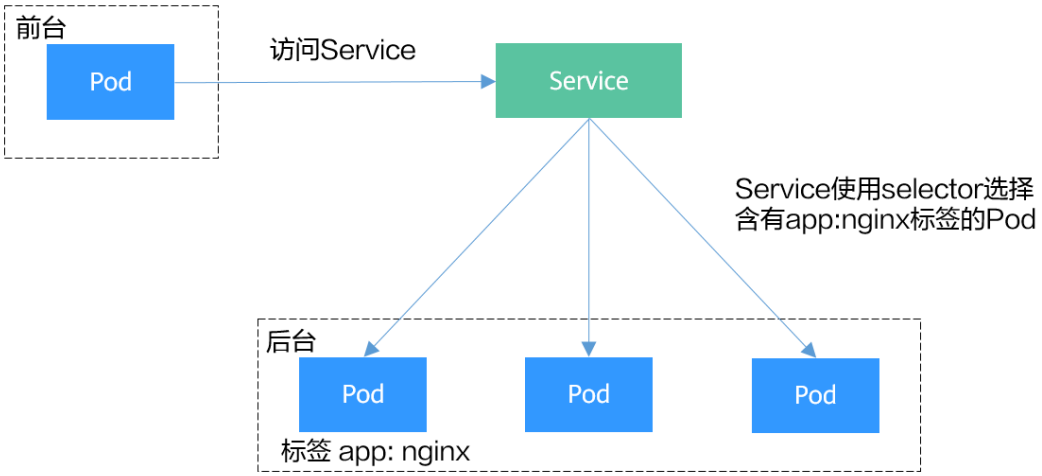


使用 Service 解决 Pod 的访问问题

Kubernetes 中的 Service 对象就是用来解决上述 Pod 访问问题的。Service 有一个固定 IP 地址（在创建 CCE 集群时有一个服务网段的设置，这个网段专门用于给 Service 分配 IP 地址），Service 将访问它的流量转发给 Pod，具体转发给哪些 Pod 通过 Label 来选择，而且 Service 可以给这些 Pod 做负载均衡。

那么对于上面的例子，为后台添加一个 Service，通过 Service 来访问 Pod，这样前台 Pod 就无需感知后台 Pod 的变化。

图9-19 通过 Service 访问 Pod



Service 的类型

Kubernetes 允许指定一个需要的类型的 Service，类型的取值以及行为如下：

- 集群内访问(ClusterIP)

集群内访问表示工作负载暴露给同一集群内其他工作负载访问的方式，可以通过“集群内部域名”访问。

- 节点访问(NodePort)

  节点访问（NodePort）是指在每个节点的 IP 上开放一个静态端口，通过静态端口对外暴露服务。节点访问（NodePort）会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求<NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务。

- 负载均衡(LoadBalancer)

  负载均衡（LoadBalancer）可以通过弹性负载均衡从公网访问到工作负载，与弹性 IP 方式相比提供了高可靠的保障，一般用于系统中需要暴露到公网的服务。

9.3.2 集群内访问(ClusterIP)

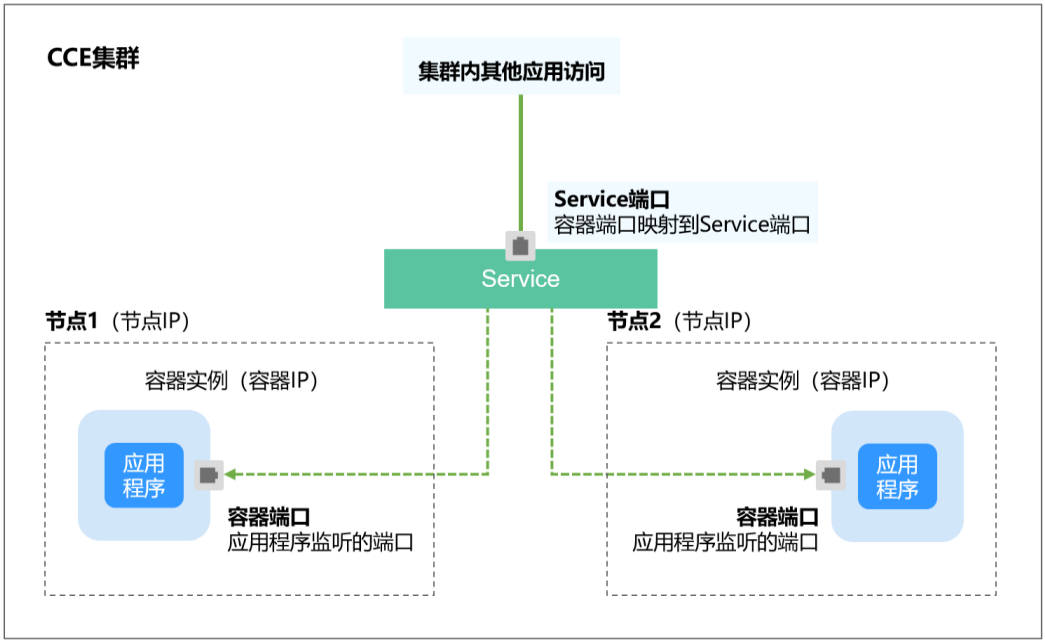
操作场景

集群内访问表示工作负载暴露给同一集群内其他工作负载访问的方式，可以通过“集群内部域名”访问。

集群内部域名格式为“<服务名称>.<工作负载所在命名空间>.svc.cluster.local:<端口号>”，例如“nginx.default.svc.cluster.local:80”。

访问通道、容器端口与访问端口映射如下图所示。

图9-20 集群内访问



创建 ClusterIP 类型 Service

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“服务发现”，在右上角单击“创建服务”。

步骤 3 设置集群内访问参数。

- **Service 名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型：**选择“集群内访问 ClusterIP”。
- **命名空间：**工作负载所在命名空间。
- **选择器：**添加标签，Service 根据标签选择 Pod，填写后单击“添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **IPv6：**默认不开启，开启后服务的集群内 IP 地址（ClusterIP）变为 IPv6 地址。**该功能仅在 1.15 及以上版本的集群创建时开启了 IPv6 功能才会显示。**
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service 使用的端口，端口范围为 1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如 nginx 默认使用 80 端口。

步骤 4 单击“确定”，创建 Service。

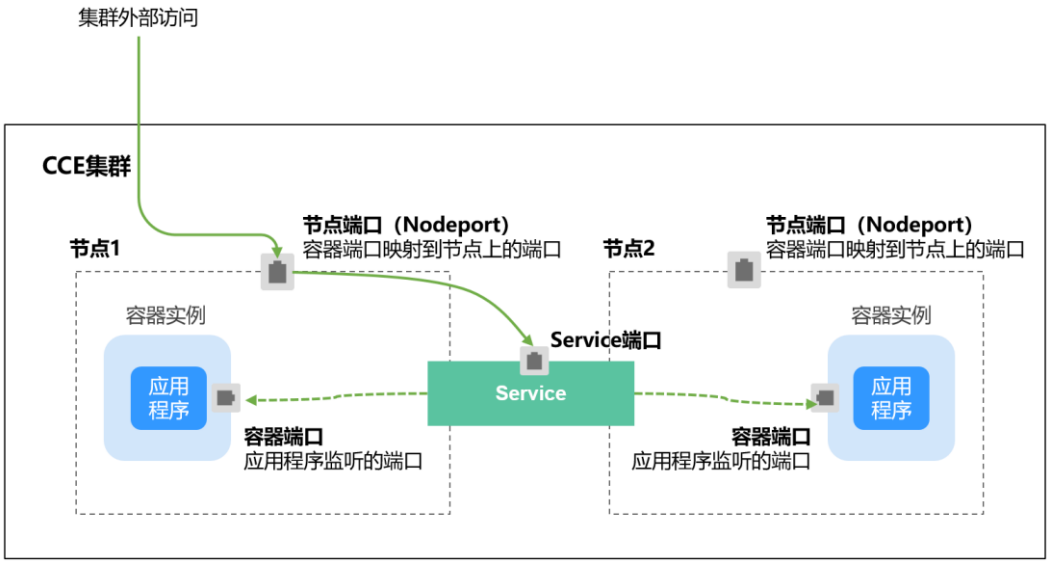
----结束

9.3.3 节点访问(NodePort)

操作场景

节点访问（NodePort）是指在每个节点的 IP 上开放一个静态端口，通过静态端口对外暴露服务。节点访问（NodePort）会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 <NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务。

图9-21 NodePort 访问



约束与限制

- “节点访问（NodePort）”默认为 VPC 内网访问，如果需要使用弹性 IP 通过公网访问该服务，请提前在集群的节点上绑定弹性 IP。
- 创建 service 后，如果服务亲和从集群级别切换为节点级别，连接跟踪表将不会被清理，建议用户创建 service 后不要修改服务亲和属性，如需修改请重新创建 service。
- VPC 网络模式下，当某容器 A 通过 NodePort 类型服务发布时，且服务亲和设置为节点级别（即 externalTrafficPolicy 为 local），部署在同节点的容器 B 将无法通过节点 IP+NodePort 访问容器 A。

创建 NodePort 类型 Service

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“服务发现”，在右上角单击“创建服务”。

步骤 3 设置集群内访问参数。

- **Service 名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型：**选择“节点访问 NodePort”。
- **命名空间：**工作负载所在命名空间。
- **服务亲和：**详情请参见 externalTrafficPolicy（服务亲和）。
  - 集群级别：集群下所有节点的 IP+访问端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源 IP。
  - 节点级别：只有通过负载所在节点的 IP+访问端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源 IP。

- **选择器：**添加标签，Service 根据标签选择 Pod，填写后单击“添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **IPv6：**默认不开启，开启后服务的集群内 IP 地址（ClusterIP）变为 IPv6 地址。**该功能仅在 1.15 及以上版本的集群创建时开启了 IPv6 功能才会显示。**
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service 使用的端口，端口范围为 1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如 nginx 默认使用 80 端口。
  - 节点端口：即 NodePort，建议选择“自动生成”；也可以指定端口，默认范围为 30000-32767。

步骤 4 单击“确定”，创建 Service。

----结束

externalTrafficPolicy（服务亲和）

NodePort 类型的 Service 接收请求时先从访问到节点，然后转到 Service，再由 Service 选择一个 Pod 转发到该 Pod，选择的 Pod 不一定在接收请求的节点上。默认情况下，从任意节点 IP+服务端口都能访问到后端工作负载，当 Pod 不在接收请求的节点上时，请求会在跳转到 Pod 所在的节点，带来一定性能损失。

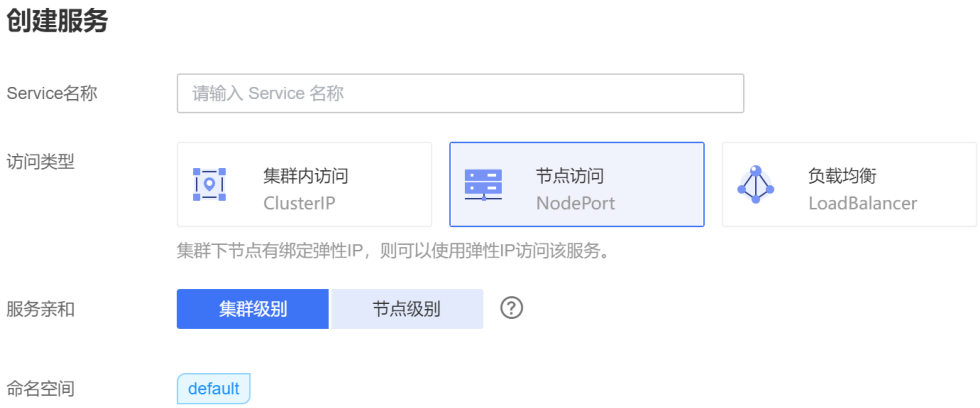
Service 有一个配置参数 externalTrafficPolicy，如下所示。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
spec:
  externalTrafficPolicy: local
  ports:
  - name: service
    nodePort: 30000
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: NodePort
```

当 externalTrafficPolicy 取值为 **local** 时，通过节点 IP:服务端口的请求只会转发给本节点上的 Pod，如果节点没有 Pod 的话请求会挂起。

externalTrafficPolicy 还有一个取值是 **cluster**，也是默认取值，就是前面说的请求会在集群内转发。

在 CCE 控制台创建 NodePort 类型 Service 时也可以配置该参数。



总结 externalTrafficPolicy 两个取值。

- cluster（集群级别）：集群下所有节点的 IP+访问端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源 IP。
- local（节点级别）：只有通过负载所在节点的 IP+访问端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源 IP。

9.3.4 负载均衡(LoadBalancer)

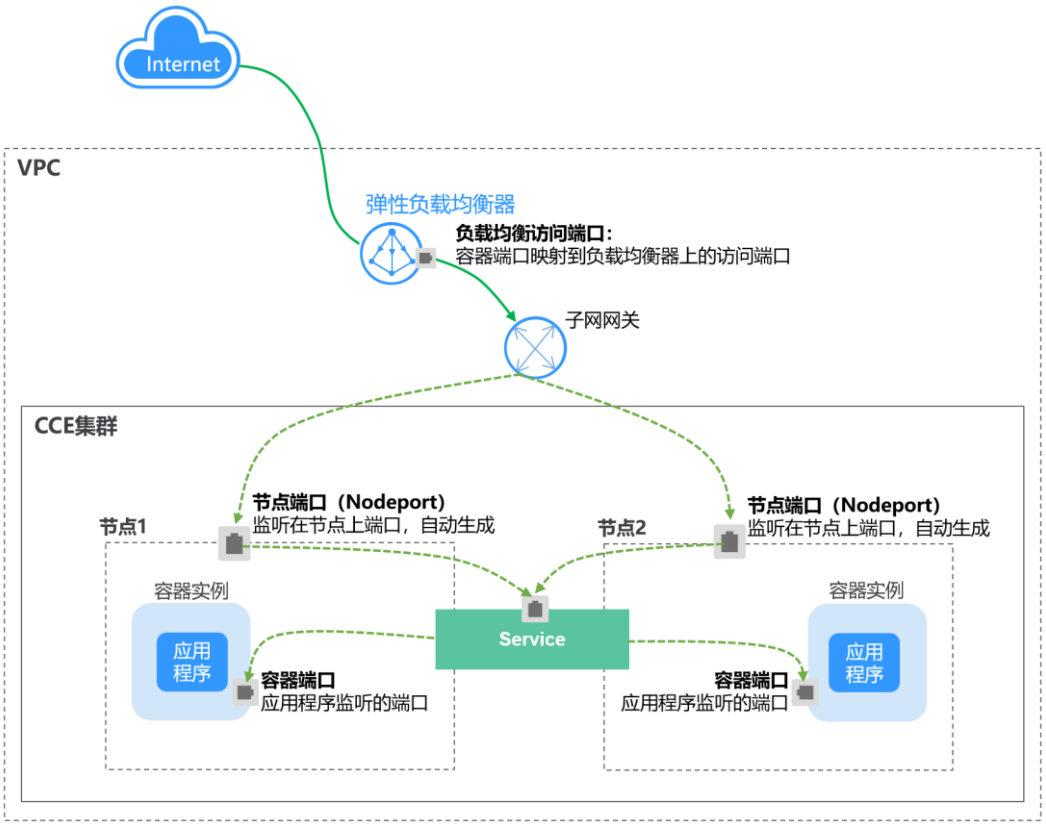
操作场景

负载均衡(LoadBalancer)可以通过弹性负载均衡从公网访问到工作负载，与弹性 IP 方式相比提供了高可靠的保障，一般用于系统中需要暴露到公网的服务。

负载均衡访问方式由公网弹性负载均衡服务地址以及设置的访问端口组成，例如“10.117.117.117:80”。

在访问时从 ELB 过来的流量会先访问到节点，然后通过 Service 转发到 Pod。

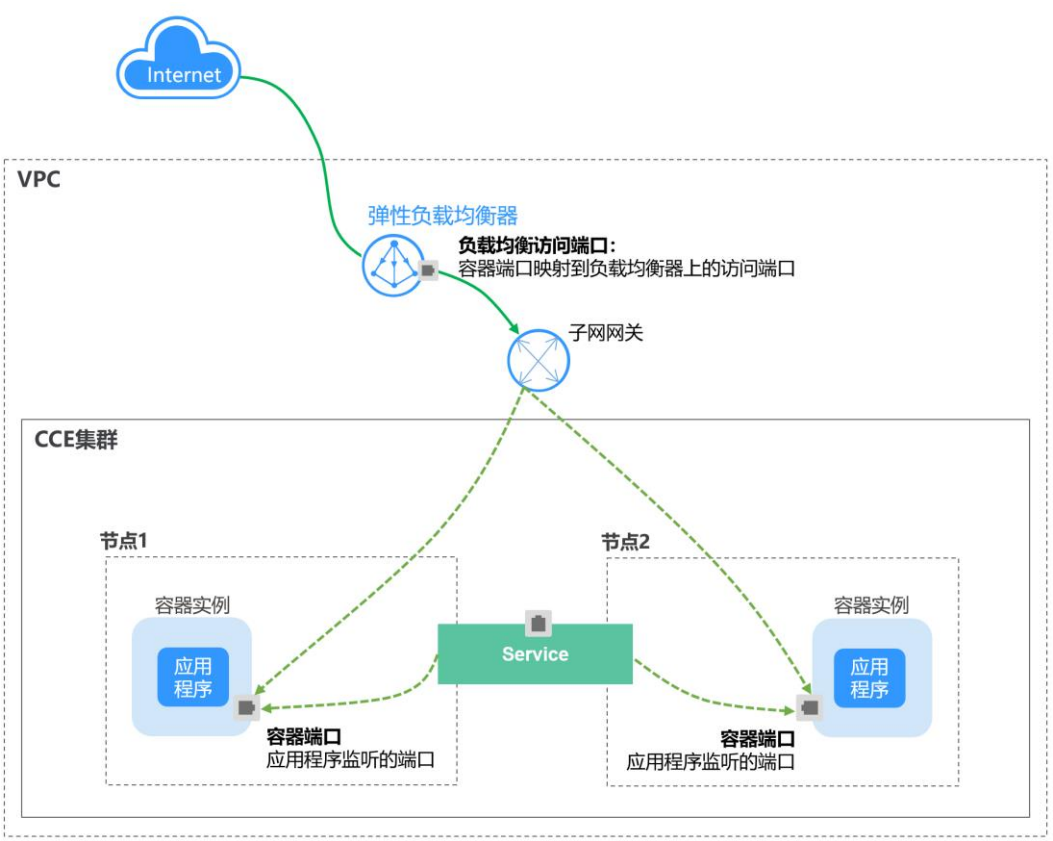
图9-22 负载均衡(LoadBalancer)



在使用 **CCE Turbo 集群 + 独享型 ELB 实例**时，支持 ELB 直通 Pod，使部署在容器中的业务时延降低、性能无损耗。

从集群外部访问时，从 ELB 直接转发到 Pod；集群内部访问可通过 Service 转发到 Pod。

图9-23 ELB 直通容器



约束与限制

- CCE 中的负载均衡（LoadBalancer）访问类型使用弹性负载均衡 ELB 提供网络访问，存在如下产品约束：
  - 自动创建的 ELB 实例建议不要被其他资源使用，否则会在删除时被占用，导致资源残留。
  - 1.15 及之前版本集群使用的 ELB 实例请不要修改监听器名称，否则可能导致无法正常访问。
- 创建 service 后，如果服务亲和从集群级别切换为节点级别，连接跟踪表将不会被清理，建议用户创建 service 后不要修改服务亲和属性，如需修改请重新创建 service。
- 当服务亲和设置为节点级别（即 externalTrafficPolicy 为 local）时，集群内部可能使用 ELB 地址访问不通，具体情况请参见集群内使用 ELB 地址无法访问 Service 说明。
- CCE Turbo 集群仅支持集群级别服务亲和。
- 独享型 ELB 仅支持 1.17 及以上集群。
- 独享型 ELB 规格必须支持网络型（TCP/UDP），且网络类型必须支持私网（有私有 IP 地址）。如果需要 Service 支持 HTTP，则独享型 ELB 规格需要为网络型（TCP/UDP）和应用型（HTTP/HTTPS）。

- 使用控制台创建 LoadBalancer 类型 Service 时会自动生成一个节点端口（nodeport），端口号随机。使用 kubectl 创建 LoadBalancer 类型 Service 时，如不指定节点端口，也会随机生成一个节点端口，端口号随机。
- 使用 CCE 集群时，如果 LoadBalancer 类型 Service 的服务亲和类型为集群级别（cluster），当请求进入到集群时，会使用 SNAT 分发到各个节点的节点端口（nodeport），不能超过节点可用的 nodeport 数量，而服务亲和为节点级别（local）则无此约束。使用 CCE Turbo 集群时，如果是使用共享型 ELB 依然有此约束，而独享型 ELB 无此约束，建议使用 CCE Turbo 时配合使用独享型 ELB。
- 集群服务转发模式为 IPVS 时，不支持配置节点的 IP 作为 Service 的 externalIP，会导致节点不可用。
- IPVS 模式集群下，Ingress 和 Service 使用相同 ELB 实例时，无法在集群内的节点和容器中访问 Ingress，因为 kube-proxy 会在 ipvs-0 的网桥上挂载 LB 类型的 Service 地址，Ingress 对接的 ELB 的流量会被 ipvs-0 网桥劫持。建议 Ingress 和 Service 使用不同 ELB 实例。

创建 LoadBalancer 类型 Service

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“服务发现”，在右上角单击“创建服务”。

步骤 3 设置参数。

- **Service 名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型：**选择“负载均衡 LoadBalancer”。
- **命名空间：**工作负载所在命名空间。
- **服务亲和：**详情请参见 externalTrafficPolicy（服务亲和）。
  - 集群级别：集群下所有节点的 IP+访问端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源 IP。
  - 节点级别：只有通过负载所在节点的 IP+访问端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源 IP。
- **选择器：**添加标签，Service 根据标签选择 Pod，填写后单击“添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **IPv6：**默认不开启，开启后服务的集群内 IP 地址（ClusterIP）变为 IPv6 地址。**该功能仅在 1.15 及以上版本的集群创建时开启了 IPv6 功能才会显示。**
- **负载均衡器：**

  选择对接的 ELB 实例，仅支持与集群在同一个 VPC 下的 ELB 实例。如果没有可选的 ELB 实例，请单击“创建负载均衡器”跳转到 ELB 控制台创建。

  CCE 控制台支持自动创建 ELB 实例，在下拉框选择自动创建，填写 ELB 实例名称、是否公网访问（将创建 5 Mbit/s 带宽的弹性公网 IP。默认按照流量计费），独享型 ELB 实例还需选择可用区、子网和规格。当前仅支持自动创建网络型（TCP/UDP）独享型 ELB 实例。

  您可以单击“编辑”配置 ELB 实例的参数，在弹出窗口中配置 ELB 实例的参数。
  - 分配策略：可选择加权轮询算法、加权最少连接或源 IP 算法。

说明

- 加权轮询算法：根据后端服务器的权重，按顺序依次将请求分发给不同的服务器。它用相应的权重表示服务器的处理性能，按照权重的高低以及轮询方式将请求分配给各服务器，相同权重的服务器处理相同数目的连接数。常用于短连接服务，例如 HTTP 等服务。
- 加权最少连接：最少连接是通过当前活跃的连接数来估计服务器负载情况的一种动态调度算法。加权最少连接就是在最少连接数的基础上，根据服务器的不同处理能力，给每个服务器分配不同的权重，使其能够接受相应权值数的服务请求。常用于长连接服务，例如数据库连接等服务。
- 源 IP 算法：将请求的源 IP 地址进行 Hash 运算，得到一个具体的数值，同时对后端服务器进行编号，按照运算结果将请求分发到对应编号的服务器上。这可以使得对不同源 IP 的访问进行负载分发，同时使得同一个客户端 IP 的请求始终被派发至某特定的服务器。该方式适合负载均衡无 cookie 功能的 TCP 协议。
  - 会话保持类型：默认不启用，可选择“源 IP 地址”。负载均衡监听是基于 IP 地址的会话保持，即来自同一 IP 地址的访问请求转发到同一台后端服务器上。
  - 健康检查：默认不启用。此处健康检查是设置负载均衡的健康检查配置。当端口配置协议为 TCP 时，支持 TCP 和 HTTP 协议，当端口配置协议为 UDP 时，支持 UDP 协议。健康检查默认使用业务端口（Service 的 NodePort 和容器端口）作为健康检查的端口；您也可以重新指定端口用于健康检查，重新制定端口会为服务增加一个名为 cce-healthz 的服务端口配置。

- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service 使用的端口，端口范围为 1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如 nginx 默认使用 80 端口。

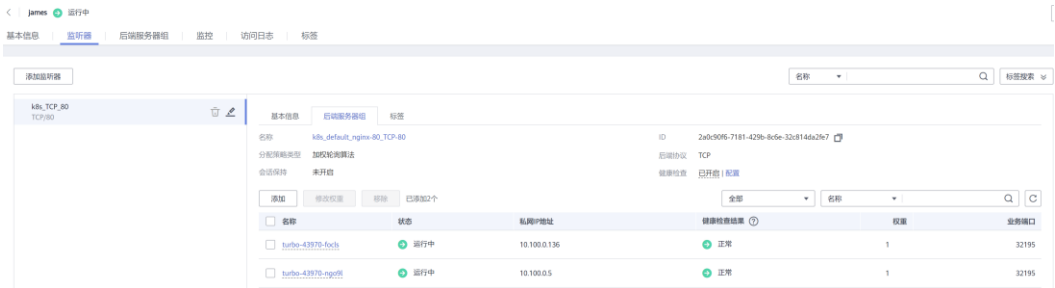
步骤 4 单击“确定”，创建 Service。

----结束

ELB 转发说明

LoadBalancer 类型 Service 创建完后，可以在 ELB 控制台查看 ELB 实例的监听器转发规则，如下所示。

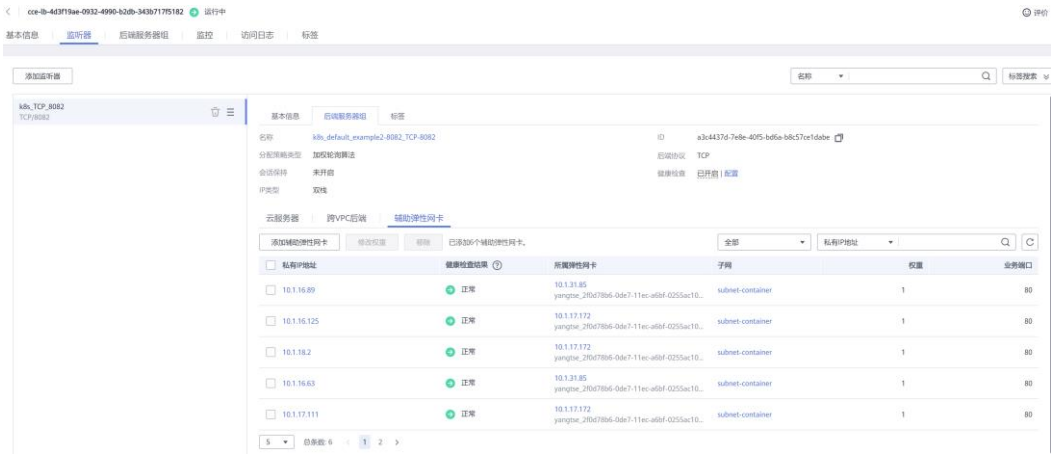
图9-24 ELB 转发说明



可以看到这个 ELB 实例创建了一个监听器，其后端服务器为 Pod 所在的节点，后端服务器端口为 Service 的 NodePort（节点端口）。当有流量通过 ELB 请求时，会转发给 Pod 所在节点 IP:节点端口，也就是访问到了 Service，从而访问到 Pod，这跟操作场景中所述是一致的。

ELB 直通容器场景（CCE Turbo + 独享型 ELB 实例）下，LoadBalancer 类型 Service 创建完后，可以在 ELB 控制台查看 ELB 实例的监听器转发规则，如下所示。

图9-25 ELB 转发说明



可以看到这个 ELB 实例创建了一个监听器，其后端服务器地址是 Pod 的 IP 地址，业务端口是容器端口。这是因为 Pod 使用了 ENI 或 Sub-ENI，ELB 会直通 Pod，当有流量通过 ELB 请求时，会直接转发给 Pod，从而访问到 Pod，这跟操作场景中所述是一致的。

Service 使用 HTTP

说明

- Service 使用 HTTP 仅 v1.19.16 及以上版本集群支持。
- CCE 控制台当前仅支持自动创建 4 层独享型 ELB 实例，此种情况下无法使用 HTTP 能力，在控制台创建 Service 使用 HTTP 时请选择对接已有独享型 ELB 实例。
- 请勿将 Ingress 与使用 HTTP 的 Service 对接同一个 ELB 下的同一个监听器，否则将产生端口冲突。

Service 支持使用 ELB 的 7 层能力，共享型和独享型 ELB 都支持对接。

独享型 ELB 实例有如下限制：

- 对接已有的独享型 ELB 实例，需要**独享型 ELB 实例同时支持 4 层和 7 层的 flavor**，否则会功能不可用。
- 使用自动创建的 ELB 实例，注意**同时使用独享型 ELB 实例的 4 层和 7 层能力**，需要在 kubernetes.io/elb.autocreate 的 annotation 中指定 4 层和 7 层 flavor。

使用 ELB 的 7 层能力时，需要添加如下 annotation

- **kubernetes.io/elb.protocol-port**: "https:443,http:80"

protocol-port 的取值需要和 service 的 spec.ports 字段中的端口对应，格式为 protocol:port，port 中的端口会匹配 service.spec.ports 中端口，并将该端口发布成对应的 protocol 协议。

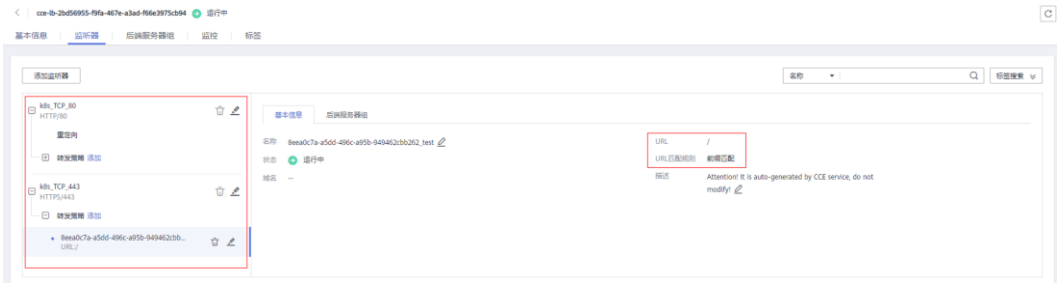
- **kubernetes.io/elb.cert-id**: "17e3b4f4bc40471c86741dc3aa211379"

  cert-id 内容为 ELB 证书管理的证书 ID，当 protocol-port 指定了 https 协议，ELB 监听器的证书会设置为 cert-id 证书，当发布多个 HTTPS 的服务，会使用同一份证书。

配置示例如下，其中 spec.ports 中两个端口与 kubernetes.io/elb.protocol-port 中对应，443 端口、80 端口分别发布成 HTTPS、HTTP 协议。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.autocreate: '
      {
          "type": "public",
          "bandwidth_name": "cce-bandwidth-1634816602057",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "available_zone": [
              "cn-north-4b"
          ],
          "l7 flavor name": "L7 flavor.elb.s2.small",
          "l4 flavor name": "L4 flavor.elb.s1.medium"
      }'
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.protocol-port: "https:443,http:80"
    kubernetes.io/elb.cert-id: "17e3b4f4bc40471c86741dc3aa211379"
  labels:
    app: nginx
    name: test
  name: test
  namespace: default
spec:
  ports:
  - name: cce-service-0
    port: 443
    protocol: TCP
    targetPort: 80
  - name: cce-service-1
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
    version: v1
  sessionAffinity: None
  type: LoadBalancer
```

使用上面的示例创建 Service，在新建的 ELB 实例中可以看到创建了 443 端口和 80 端口的监听器。



集群内使用 ELB 地址无法访问 Service 说明

当 LoadBalancer Service 设置了服务亲和为节点级别，即 externalTrafficPolicy 取值为 Local 时，在使用中可能会碰到从集群内部（节点上或容器中）使用 ELB 地址访问不通的情况，回显类似如下内容：

```
upstream connect error or disconnect/reset before headers. reset reason: connection failure
```

这是由于 Kubernetes 在创建 LoadBalancer Service 时，kube-proxy 会把 ELB 的访问地址作为 External-IP 添加到 iptables 或 IPVS 中。如果客户端从集群内部发起访问 ELB 地址的请求，该地址会被认为是服务的 External-IP，被 kube-proxy 直接转发，而不再经过集群外部的 ELB。

当 externalTrafficPolicy 的取值为 Local 时，在不同容器网络模型和服务转发模式下，情况会有所不同，详情如下：

Server	Client	容器隧道集群（IPVS）	VPC 集群（IPVS）	容器隧道集群（iptables）	VPC 集群（iptables）
节点访问类型 Service	同节点	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通
	跨节点	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通，访问本节点 IP+访问端口通；其他不通	OK，访问 Pod 容器所在节点通，访问本节点 IP+访问端口通；其他不通
	同节点容器	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点不通
	跨节点	OK，Pod 容器所在	OK，访问 Pod 容器所	OK，访问 Pod 容器所在	OK，访问 Pod 容器所在节点

	容器	节点通，其他不通	在节点通	节点通	通
独享型负载均衡类型 Service	同节点	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
	同节点容器	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
nginx-ingress 插件的 service 为 Local 级别独享型	同节点	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
	同节点容器	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通

解决这个问题通常有如下办法：

- （**推荐**）在集群内部访问使用 Service 的 ClusterIP 或服务域名访问。
- 将 Service 的 externalTrafficPolicy 设置为 Cluster，即集群级别服务亲和。不过需要注意这会影响源地址保持。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sha
retype":"PER","eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

- 使用 Service 的 pass-through 特性，使用 ELB 地址访问时绕过 kube-proxy，先访问 ELB，经过 ELB 再访问到负载。

说明

- 独享型负载均衡配置 pass-through 后，在工作负载同节点和同节点容器内无法通过 Service 访问。
- 1.15 及以下老版本集群暂不支持该能力。
- IPVS 网络模式下，对接同一个 ELB 的 Service 需保持 pass-through 设置情况一致。

```
apiVersion: v1
kind: Service
```

```
metadata:
  annotations:
    kubernetes.io/elb.pass-through: "true"
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sha
retype":"PER","eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Local
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

9.3.5 Headless Service

前面讲的 Service 解决了 Pod 的内外部访问问题，但还有下面这些问题没解决。

- 同时访问所有 Pod
- 一个 Service 内部的 Pod 互相访问

Headless Service 正是解决这个问题的，Headless Service 不会创建 ClusterIP，并且查询会返回所有 Pod 的 DNS 记录，这样就可查询到所有 Pod 的 IP 地址。有状态负载 StatefulSet 正是使用 Headless Service 解决 Pod 间互相访问的问题。

```
apiVersion: v1
kind: Service      # 对象类型为 Service
metadata:
  name: nginx-headless
  labels:
    app: nginx
spec:
  ports:
    - name: nginx     # Pod 间通信的端口名称
      port: 80        # Pod 间通信的端口号
  selector:
    app: nginx        # 选择标签为 app:nginx 的 Pod
  clusterIP: None     # 必须设置为 None，表示 Headless Service
```

执行如下命令创建 Headless Service。

```
# kubectl create -f headless.yaml
service/nginx-headless created
```

创建完成后可以查询 Service。

```
# kubectl get svc
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
nginx-headless  ClusterIP   None         <none>        80/TCP    5s
```

创建一个 Pod 来查询 DNS，可以看到能返回所有 Pod 的记录，这就解决了访问所有 Pod 的问题了。

```
$ kubectl run -i --tty --image tutum/dnsutils dnsutils --restart=Never --rm /bin/sh
If you don't see a command prompt, try pressing enter.
/ # nslookup nginx-0.nginx
Server:        10.247.3.10
Address:       10.247.3.10#53
Name:   nginx-0.nginx.default.svc.cluster.local
Address: 172.16.0.31

/ # nslookup nginx-1.nginx
Server:        10.247.3.10
Address:       10.247.3.10#53
Name:   nginx-1.nginx.default.svc.cluster.local
Address: 172.16.0.18

/ # nslookup nginx-2.nginx
Server:        10.247.3.10
Address:       10.247.3.10#53
Name:   nginx-2.nginx.default.svc.cluster.local
Address: 172.16.0.19
```

9.3.6 Service Annotations 说明

CCE 提供一些使用 Service 的高级功能，这些功能使用时可以通过给 YAML 添加注解 Annotation 实现。具体的 Annotation 如下表所示。

Service 的 Annotation 主要是 Service 对接 ELB 时需要指定的参数。

表9-4 Service Annotation

参数	类型	描述	控制台默认取值	支持的集群版本
kubernetes.io/elb.class	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： • union：共享型负载均衡。 • performance：独享型负载均衡，仅支持 1.17 及以上集群	performance	v1.9 及以上
kubernetes.io/elb.id	String	为负载均衡实例的 ID，取值范围：1-100 字符。 在关联已有 ELB 时：必填。 **获取方法：**	无	v1.9 及以上

参数	类型	描述	控制台默认取值	支持的集群版本
		在控制台的“服务列表”中，单击“网络 > 弹性负载均衡 ELB”，单击 ELB 的名称，在 ELB 详情页的“基本信息”页签下找到“ID”字段复制即可。		
kubernetes.io/elb.protocol-port	String	Service 使用 7 层能力配置端口。 详细使用请参见 Service 使用 HTTP。	无	v1.19.16 及以上
kubernetes.io/elb.cert-id	String	Service 使用 7 层能力配置 HTTPS 证书。 详细使用请参见 Service 使用 HTTP。	无	v1.19.16 及以上
kubernetes.io/elb.subnet-id	String	为集群所在子网的 ID，取值范围：1-100 字符。 • Kubernetes v1.11.7-r0 及以下版本的集群自动创建时：必填 • Kubernetes v1.11.7-r0 以上版本的集群：可不填。	无	v1.11.7-r0 以下必填 v1.11.7-r0 以上该字段废弃
kubernetes.io/elb.enterpriseID	String	**v1.15 及以上版本的集群支持此字段，v1.15 以下版本默认创建到 default 项目下。** 为 ELB 企业项目 ID，选择后可以直接创建在具体的 ELB 企业项目下。 该字段不传（或传为字符串'0'），则将资源绑定给默认企业项目。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“企业 > 项目管理”，在打开的企业项目列表中单击要加入的企业项目名称，进入企业项目详情页，找到“ID”字段复制即可。	无	v1.15 及以上

参数	类型	描述	控制台默认取值	支持的集群版本
kubernetes.io/elb.autocreate		自动创建 service 关联的 ELB **示例：** • 公网自动创建： 值为 '{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}' • 私网自动创建： 值为 '{"type":"inner", "name": "A-location-d-test"}'	无	v1.9 及以上
kubernetes.io/elb.adaptive-weight	String	根据 Pod 动态调整 ELB 后端云主机的权重。每个 Pod 收到的负载请求更加均衡。 • 开启：true • 关闭：false 该参数仅 1.21 及以上集群适用，且 ELB 直通 Pod 场景下无效。	无	v1.21 及以上
kubernetes.io/elb.lb-algorithm	String	后端云主机组的负载均衡算法。 取值范围： • ROUND_ROBIN：加权轮询算法。 • LEAST_CONNECTIONS：加权最少连接算法。 • SOURCE_IP：源 IP 算法。 当该字段的取值为 SOURCE_IP 时，后端云主机组绑定的后端云主机的 weight 字段无效。	ROUND_ROBIN	v1.9 及以上
kubernetes.io/elb.health-check-	String	是否开启 ELB 健康检查功能。	off	v1.9 及以上

参数	类型	描述	控制台默认取值	支持的集群版本
flag		• 开启：“（空值）”或“on” • 关闭：“off” 开启时需同时填写 kubernetes.io/elb.health-check-option 字段。		
kubernetes.io/elb.health-check-option		ELB 健康检查配置选项。	无	v1.9 及以上
kubernetes.io/elb.pass-through	String	集群内访问 Service 是否经过 ELB。	无	v1.19 及以上
kubernetes.io/elb.session-affinity-mode	String	负载均衡监听是基于 IP 地址的会话保持，即来自同一 IP 地址的访问请求转发到同一台后端服务器上。 • 不启用：不填写该参数。 • 开启会话保持：需增加该参数，取值“SOURCE_IP”，表示基于源 IP 地址。	无	v1.9 及以上
kubernetes.io/elb.acl-id	String	为 ELB 设置 IP 地址黑名单或白名单时需填写，参数值为 ELB 的 IP 地址组 ID。 **该参数仅独享型 ELB 生效，且仅在新建 Service 或指定新的服务端口（监听器）时生效。**	无	v1.19.16 v1.21.4
kubernetes.io/elb.acl-status	String	为 ELB 设置 IP 地址黑名单或白名单时需填写，取值为 'on'，表示开启访问控制。 **该参数仅独享型 ELB 生效，且仅在新建 Service 或指定新的服务端口（监听器）时生效。**	无	v1.19.16 v1.21.4
kubernetes.io/elb.acl-type	String	为 ELB 设置 IP 地址黑名单或白名单时需填写。 • black：表示黑名单，所选 IP 地址组无法访问 ELB 地址。	无	v1.19.16 v1.21.4

参数	类型	描述	控制台默认取值	支持的集群版本
		• white：表示白名单，仅所选 IP 地址组可以访问 ELB 地址。 **该参数仅独享型 ELB 生效，且仅在新建 Service 或指定新的服务端口（监听器）时生效。**		
kubernetes.io/elb.session-affinity-option		ELB 会话保持配置选项，可设置会话保持的超时时间。	无	v1.9 及以上
kubernetes.io/hws-hostNetwork	Boolean	为标记工作负载服务是否使用主机网络模式。如果 Pod 使用的主机网络，开启这个 annotation 会 ELB 转发到主机网络的方式对接。 取值范围：“true”或者“false” 默认是“false”，表示未使用主机网络。	无	v1.9 及以上

表9-5 elb.autocreate 字段数据结构说明

参数	是否必填	参数类型	描述
name	否	String	自动创建的负载均衡的名称。 取值范围：1-64 个字符，小写字母，数字，下划线，小写字母开头，小写字母或者数字结尾。 默认名称：cce-lb+service.UID
type	否	String	负载均衡实例网络类型，公网或者私网。 • public：公网型负载均衡 • inner：私网型负载均衡 默认类型：inner
bandwidth_name	公网型负载均衡必填	String	带宽的名称，默认值为：cce-bandwidth-******。 取值范围：1-64 个字符，小写字母，数字，下划线，小写字母开头，小写字母

参数	是否必填	参数类型	描述
			或者数字结尾。
bandwidth_chargemode	否	String	带宽付费模式。 • bandwidth：按带宽 • traffic：按流量 默认类型：bandwidth
bandwidth_size	公网型负载均衡必填	Integer	带宽大小，默认 1Mbit/s~2000Mbit/s，请根据 Region 带宽支持范围设置。
bandwidth_sharetype	公网型负载均衡必填	String	带宽共享方式。 • PER：独享带宽
eip_type	公网型负载均衡必填	String	弹性公网 IP 类型。
vip_subnet_cidr_id	否	String	指定 ELB 所在的子网。1.21 及以上版本支持。 如不指定，则 ELB 与集群在同一个子网。
available_zone	是	Array of strings	负载均衡所在可用区。 独享型负载均衡器独有字段。
l4_flavor_name	是	String	四层负载均衡实例规格名称。 独享型负载均衡器独有字段。
l7_flavor_name	否	String	七层负载均衡实例规格名称。 独享型负载均衡器独有字段。
elb_virsubnet_ids	否	Array of strings	负载均衡后端所在子网，不填默认集群子网。不同实例规格将占用不同数量子网 IP，不建议使用其他资源（如集群，节点等）的子网网段。 独享型负载均衡器独有字段。 示例： `"elb_virsubnet_ids": [ "14567f27-8ae4-42b8-ae47-9f847a4690dd" ]`

表9-6 elb.health-check-option 字段数据结构说明

参数	是否必填	参数类型	描述
delay	否	String	开始健康检查的初始等待时间（秒） 默认值：5，取值范围：1-50
timeout	否	String	健康检查的超时时间（秒） 默认值：10，取值范围 1-50
max_retries	否	String	健康检查的最大重试次数 默认值：3，取值范围 1-10
protocol	否	String	健康检查的协议 默认值：取关联服务的协议 取值范围：“TCP”、“UDP”或者“HTTP”
path	否	String	健康检查的 URL，协议是“HTTP”时配置 默认值：“/” 取值范围：1-10000 字符

表9-7 elb.session-affinity-option 字段数据结构说明

参数	是否必填	参数类型	描述
persistence_timeout	是	String	当 elb.session-affinity-mode 是“SOURCE_IP”时生效，设置会话保持的超时时间（分钟）。 默认值为：60，取值范围：1-60。

9.4 Ingress

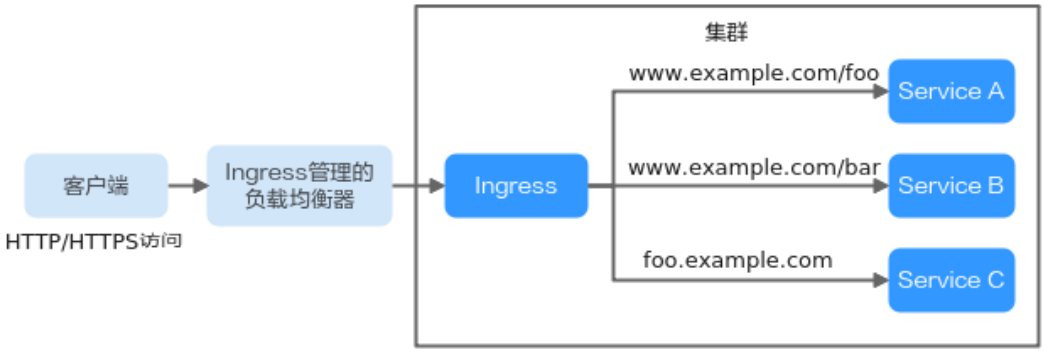
9.4.1 Ingress 概述

为什么需要 Ingress

Service 基于 TCP 和 UDP 协议进行访问转发，为集群提供了四层负载均衡的能力。但是在实际场景中，Service 无法满足应用层中存在着大量的 HTTP/HTTPS 访问需求。因此，Kubernetes 集群提供了另一种基于 HTTP 协议的访问方式——Ingress。

Ingress 是 Kubernetes 集群中一种独立的资源，制定了集群外部访问流量的转发规则。用户可根据域名和路径对转发规则进行自定义，完成对访问流量的细粒度划分。

图9-26 Ingress 示意图



下面对 Ingress 的相关定义进行介绍：

- Ingress 资源：一组基于域名或 URL 把请求转发到指定 Service 实例的访问规则，是 Kubernetes 的一种资源对象，通过接口服务实现增、删、改、查的操作。
- Ingress Controller：请求转发的执行器，用以实时监控资源对象 Ingress、Service、End-point、Secret（主要是 TLS 证书和 Key）、Node、ConfigMap 的变化，解析 Ingress 定义的规则并负责将请求转发到相应的后端 Service。

Ingress Controller 在不同厂商之间的实现方式不同，根据负载均衡器种类的不同，可以将其分成 ELB 型和 Nginx 型。CCE 支持上述两种 Ingress Controller 类型，其中 ELB Ingress Controller 基于弹性负载均衡服务（ELB）实现流量转发；而 Nginx Ingress Controller 使用 Kubernetes 社区维护的模板与镜像，通过 Nginx 组件完成流量转发。

Ingress 特性对比

表9-8 Ingress 特性对比

特性	ELB Ingress Controller	Nginx Ingress Controller
运维	免运维	自行安装、升级、维护
性能	一个 Ingress 支持一个 ELB 实例	多个 Ingress 只支持一个 ELB 实例
	使用企业级 LB，高性能高可用，升级、故障等场景不影响业务转发	性能依赖 pod 的资源配置
	支持配置动态加载	更新配置需 reload，可能会造成业务中断
组件部署	Master 节点，不占用工作节点	Worker 节点，需要 Nginx 组件运行成本

特性	ELB Ingress Controller	Nginx Ingress Controller
路由重定向	不支持	支持
SSL 配置	支持	支持

由于 ELB Ingress 和社区开源的 Nginx Ingress 在原理上存在本质区别，因此支持的 Service 类型不同。

ELB Ingress Controller 部署在 master 节点，所有策略配置和转发行为均在 ELB 侧完成。非 ELB 直通 Pod 场景下，集群外部的 ELB 只能通过 VPC 的 IP 对接集群内部节点，因此 ELB Ingress 只支持 NodePort 类型的 Service。ELB 直通 Pod 场景下，ELB 可直接将流量转发到集群内 Pod，此时 Ingress 仅支持对接 ClusterIP 类型的 Service。

Nginx Ingress Controller 运行在集群中，作为服务通过 NodePort 对外暴露，流量经过 Nginx-ingress 转发到集群内其他业务，流量转发行为及转发对象均在集群内部，因此支持 ClusterIP 和 NodePort 类型的 Service。ELB 直通 Pod 场景下，Nginx Ingress 仅支持 ClusterIP 类型的 Service。

综上，ELB Ingress 使用企业级 LB 进行流量转发，拥有高性能和高稳定性的优点，而 Nginx Ingress Controller 部署在集群节点上，牺牲了一定的集群资源但可配置性相对更好。

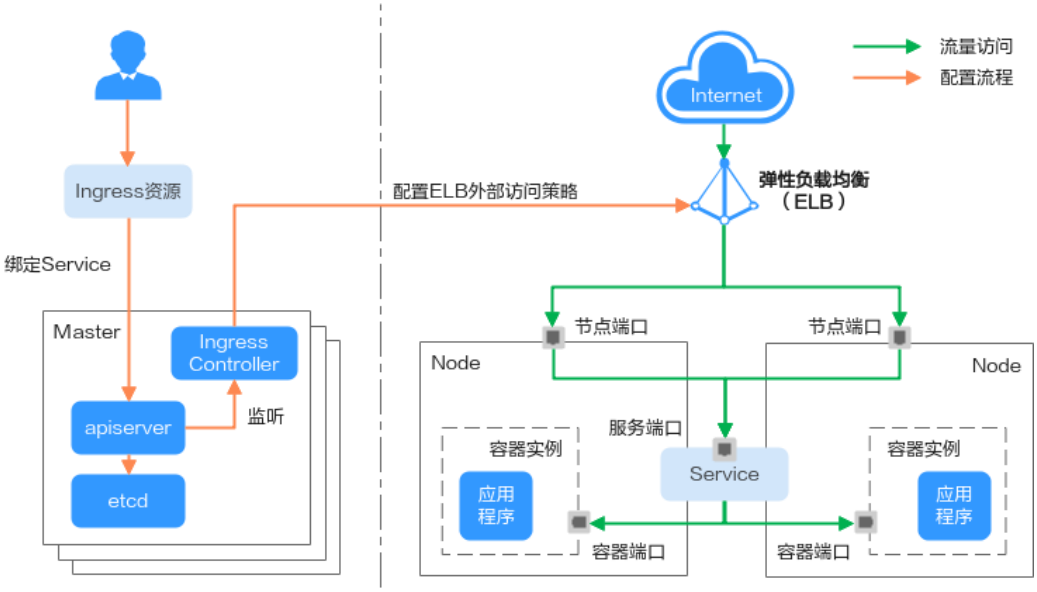
ELB Ingress Controller 工作原理

CCE 自研的 ELB Ingress Controller 基于弹性负载均衡服务 ELB 实现公网和内网（同一 VPC 内）的七层网络访问，通过不同的 URL 将访问流量分发到对应的服务。

ELB Ingress Controller 部署于 Master 节点上，与集群所在 VPC 下的弹性负载均衡器绑定，支持在同一个 ELB 实例（同一 IP）下进行不同域名、端口和转发策略的设置。ELB Ingress Controller 的工作原理如下图，实现步骤如下：

1. 用户创建 Ingress 资源，在 Ingress 中配置流量访问规则，包括负载均衡器、URL、SSL 以及访问的后端 Service 端口等。
2. Ingress Controller 监听到 Ingress 资源发生变化时，就会根据其中定义的流量访问规则，在 ELB 侧重新配置监听器以及后端服务器路由。
3. 当用户进行访问时，流量根据 ELB 中配置的转发策略转发到对应的后端 Service 端口，然后再经过 Service 二次转发访问到关联的各个工作负载。

图9-27 ELB Ingress Controller 工作原理



Nginx Ingress Controller 工作原理

Nginx 型的 Ingress 使用弹性负载均衡（ELB）作为流量入口，并在集群中部署 nginx-ingress 插件来对流量进行负载均衡及访问控制。

说明

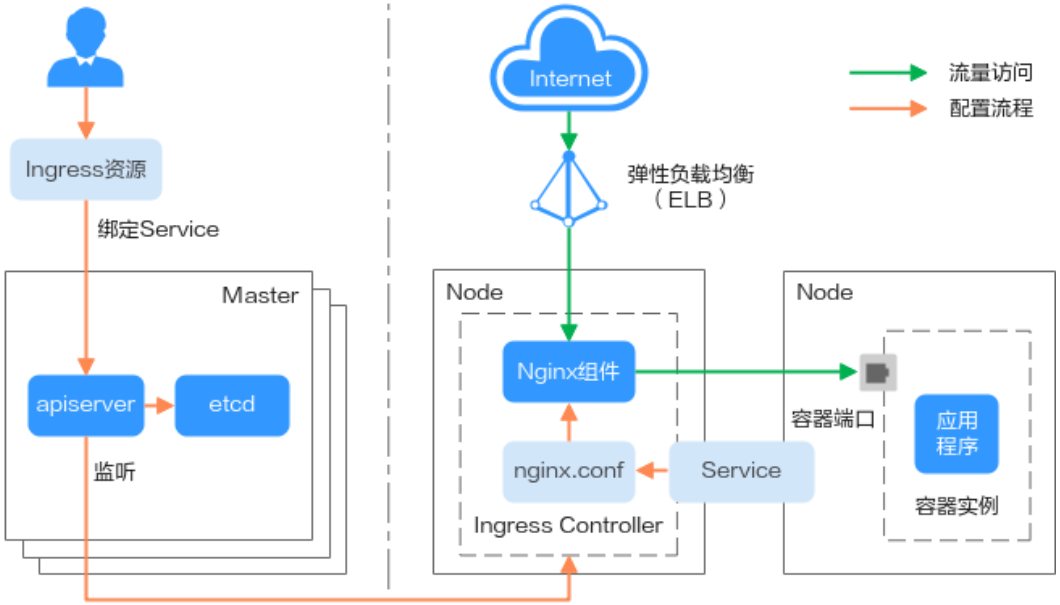
nginx-ingress 插件直接使用社区模板与镜像，CCE 不提供额外维护，不建议用于商用场景。

开源社区地址：https://github.com/kubernetes/ingress-nginx

Nginx 型的 Ingress Controller 通过 pod 部署在工作节点上，因此引入了相应的运维成本和 Nginx 组件运行成本，其工作原理如下图，实现步骤如下：

1. 当用户更新 Ingress 资源后，Ingress Controller 就会将其中定义的转发规则写入到 Nginx 的配置文件（nginx.conf）中。
2. 内置的 Nginx 组件进行 reload，加载更新后的配置文件，完成 Nginx 转发规则的修改和更新。
3. 在流量访问集群时，首先被已创建的负载均衡实例转发到集群内部的 Nginx 组件，然后 Nginx 组件再根据转发规则将其转发至对应的各个工作负载。

图9-28 Nginx Ingress Controller 工作原理



9.4.2 通过控制台使用 ELB Ingress

前提条件

- Ingress 为后端工作负载提供网络访问，因此集群中需提前部署可用的工作负载。若您无可用工作负载，可参考创建无状态负载(Deployment)、创建有状态负载(StatefulSet)或创建守护进程集(DaemonSet)部署工作负载。
- 独享型 ELB 规格必须支持应用型（HTTP/HTTPS），且网络类型必须支持私网（有私有 IP 地址）。

注意事项

- 建议其他资源不要使用 Ingress 自动创建的 ELB 实例，否则在删除 Ingress 时，ELB 实例会被占用，导致资源残留。
- 添加 Ingress 后请在 CCE 页面对所选 ELB 实例进行配置升级和维护，不可在 ELB 页面对配置进行更改，否则可能导致 Ingress 服务异常。
- Ingress 转发策略中注册的 URL 需与后端应用暴露的 URL 一致，否则将返回 404 错误。
- IPVS 模式集群下，Ingress 和 Service 使用相同 ELB 实例时，无法在集群内的节点和容器中访问 Ingress，因为 kube-proxy 会在 ipvs-0 的网桥上挂载 LB 类型的 Service 地址，Ingress 对接的 ELB 的流量会被 ipvs-0 网桥劫持。建议 Ingress 和 Service 使用不同 ELB 实例。
- 请勿将 Ingress 与使用 HTTP 的 Service 对接同一个 ELB 下的同一个监听器，否则将产生端口冲突。

添加 ELB Ingress

本节以 nginx 作为工作负载并添加 ELB Ingress 为例进行说明。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 选择左侧导航栏的“服务发现”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤 3 设置 Ingress 参数。

- **名称：**自定义 Ingress 名称，例如 ingress-demo。
- **对接 Nginx：**此选项只有在安装了 nginx-ingress 插件后才会显示。如显示了“对接 Nginx”，则说明您安装了 nginx-ingress 插件，创建 ELB Ingress 时不能打开该项开关，如果打开则是使用 Nginx Ingress Controller。
- **负载均衡器：**

  选择对接的 ELB 实例，仅支持与集群在同一个 VPC 下的 ELB 实例。如果没有可选的 ELB 实例，请单击“创建负载均衡器”调整到 ELB 控制台创建。

  独享型 ELB 规格需要支持应用型（HTTP），且网络类型必须支持私网。
- **监听器配置：**Ingress 为负载均衡器配置监听器，监听器对负载均衡器上的请求进行监听，并分发流量。配置完成后 ELB 实例侧将会创建对应的监听器，名称默认为 k8s_<协议类型>_<端口号>，例如“k8s_HTTP_80”。
  - 对外协议：支持 HTTP 和 HTTPS。
  - 对外端口：开放在负载均衡服务地址的端口，可任意指定。
  - 证书来源：支持 IngressTLS 密钥和 ELB 服务器证书。
  - 服务器证书：负载均衡器创建 HTTPS 协议监听时需要绑定证书，以支持 HTTPS 数据传输加密认证。
    - IngressTLS 密钥：创建密钥证书的方法请参见创建密钥。
    - ELB 服务器证书：使用在 ELB 服务中创建的证书。

说明

同一个 ELB 实例的同一个端口配置 HTTPS 时，一个监听器只支持配置一个密钥证书。若使用两个不同的密钥证书将两个 Ingress 添加到同一个 ELB 下的同一个监听器，ELB 侧实际只生效最先添加的证书。

  - SNI：单击  后开启 SNI 功能。SNI（Server Name Indication）是 TLS 的扩展协议，在该协议下允许同一个 IP 地址和端口号下对外提供多个基于 TLS 的访问域名，且不同的域名可以使用不同的安全证书。开启 SNI 后，允许客户端在发起 TLS 握手请求时就提交请求的域名信息。负载均衡收到 TLS 请求后，会根据请求的域名去查找证书：若找到域名对应的证书，则返回该证书认证鉴权；否则，返回缺省证书（服务器证书）认证鉴权。

说明

- 当选择 HTTPS 协议时，才支持配置“SNI”选项。
- 该功能仅支持 1.15.11 及以上版本的集群。
- 用于 SNI 的证书需要指定域名，每个证书只能指定一个域名。支持泛域名证书。

- **转发策略配置：**请求的访问地址与转发规则匹配时（转发规则由域名、URL 组成，例如：10.117.117.117:80/helloworld），此请求将被转发到对应的目标 Service 处理。单击“添加转发策略”按钮可添加多条转发策略。
  - 域名：实际访问的域名地址。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。
  - URL 匹配规则：
    - 前缀匹配：例如映射 URL 为/healthz，只要符合此前缀的 URL 均可访问。例如/healthz/v1，/healthz/v2。
    - 精确匹配：表示只有 URL 完全匹配时，访问才能生效。例如映射 URL 为/healthz，则必须为此 URL 才能访问。
    - 正则匹配：按正则表达式方式匹配 URL。例如正则表达式为**/[A-Za-z0-9_.-]+/test**。只要符合此规则的 URL 均可访问，例如/abcA9/test，/v1-Ab/test。正则匹配规则支持 POSIX 与 Perl 两种标准。
  - URL：需要注册的访问路径，例如：/healthz。

  说明

  此处添加的 URL 路径要求后端应用内存在相同的路径，否则转发无法生效。

  例如，Nginx 应用默认的 Web 访问路径为“/usr/share/nginx/html”，在为 Ingress 转发策略添加“/test”路径时，需要应用的 Web 访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回 404。

  - 目标服务名称：请选择已有 Service 或新建 Service。页面列表中的查询结果已自动过滤不符合要求的 Service。
  - 目标服务访问端口：可选择目标 Service 的访问端口。
  - 操作：可单击“删除”按钮删除该配置。

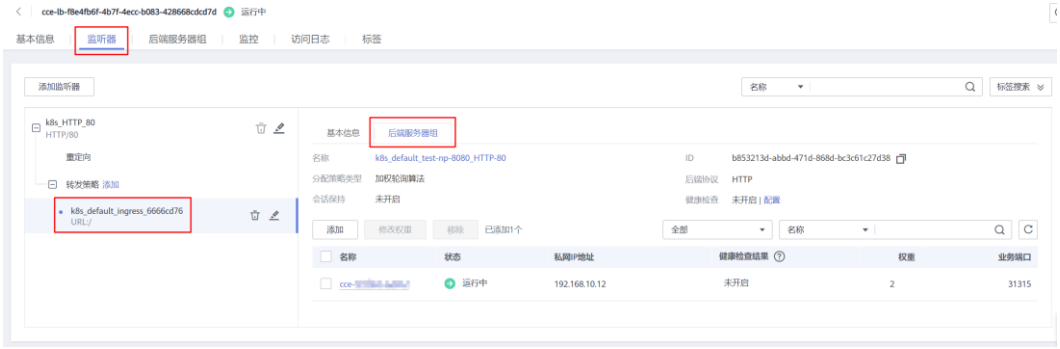
步骤 4 配置完成后，单击“确定”。创建完成后，在 Ingress 列表可查看到已添加的 Ingress。

在 ELB 控制台可查看通过 CCE 自动创建的 ELB，名称默认为“cce-lb-ingress.UID”。单击 ELB 名称进入详情页，在“监听器”页签下即可查看 Ingress 对应的路由设置，包括 URL、监听器端口以及对应的后端服务器组端口。

须知

Ingress 创建后请在 CCE 页面对所选 ELB 实例进行配置升级和维护，不要在 ELB 控制台对 ELB 实例进行维护，否则可能导致 Ingress 服务异常。

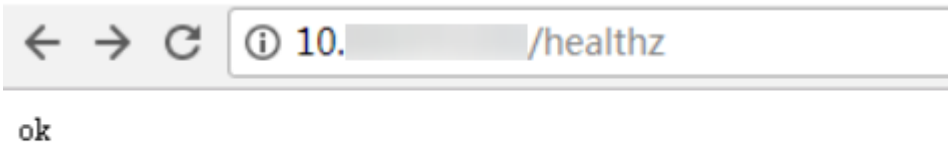
图9-29 ELB 路由设置



步骤 5 访问工作负载（例如名称为 defaultbackend）的“/healthz”接口。

1. 获取工作负载“/healthz”接口的访问地址。访问地址由负载均衡实例 IP、对外端口、映射 URL 组成，例如：10.**.**.**:80/healthz。
2. 在浏览器中输入“/healthz”接口的访问地址，如：http://10.**.**.**:80/healthz，即可成功访问工作负载。

图9-30 访问 defaultbackend“/healthz”接口



----结束

配置 HTTPS 证书

Ingress 支持配置 TLS 证书，以 HTTPS 协议的方式对外提供安全服务。

当前支持使用配置在集群中的 IngressTLS 密钥证书，以及 ELB 服务中的证书。

说明

同一个 ELB 实例的同一个端口配置 HTTPS 时，需要选择一样的证书。

使用 IngressTLS 密钥证书

步骤 1 请参见通过 kubectl 连接集群，使用 kubectl 连接集群。

步骤 2 执行如下命令，创建名为“**ingress-test-secret.yaml**”的 YAML 文件，此处文件名可自定义。

vi ingress-test-secret.yaml

YAML 文件配置如下：

```
apiVersion: v1
data:
  tls.crt: LS0******tLS0tCg==
  tls.key: LS0tL******0tLS0K
kind: Secret
metadata:
  annotations:
    description: test for ingressTLS secrets
  name: ingress-test-secret
  namespace: default
type: IngressTLS
```

说明

此处 tls.crt 和 tls.key 为示例，请获取真实密钥进行替换。tls.crt 和 tls.key 的值为 Base64 编码后的内容。

步骤 3 创建密钥。

kubectl create -f ingress-test-secret.yaml

回显如下，表明密钥已创建。

```
secret/ingress-test-secret created
```

查看已创建的密钥。

kubectl get secrets

回显如下，表明密钥创建成功。

```
NAME                  TYPE                                  DATA      AGE
ingress-test-secret      IngressTLS                            2         13s
```

步骤 4 创建名为“**ingress-test.yaml**”的 YAML 文件，此处文件名可自定义。

vi ingress-test.yaml

说明

默认安全策略选择（kubernetes.io/elb.tls-ciphers-policy）仅在 1.17.17 及以上版本的集群中支持。

自定义安全策略选择（kubernetes.io/elb.security_policy_id）仅在 1.17.17 及以上版本的集群中支持。

以自动创建关联 ELB 为例，YAML 文件配置如下：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type":"public",
          "bandwidth_name":"cce-bandwidth-15511633796**",
          "bandwidth_chargemode":"bandwidth",
          "bandwidth_size":5,
```

```
        "bandwidth_sharetype":"PER",
        "eip_type":"5_bgp"
       }'
    kubernetes.io/elb.security_policy_id: 99bec42b-0dd4-4583-98e9-b05ce628d157  #自定义安全策略优先级高于系统默认安全策略
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

表9-9 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.security_policy_id	否	String	ELB 中自定义安全组策略 ID，请前往 ELB 控制台获取。该字段仅在 HTTPS 协议下生效，且优先级高于默认安全策略。
kubernetes.io/elb.tls-ciphers-policy	否	String	默认值为“tls-1-2”，为监听器使用的默认安全策略，仅在 HTTPS 协议下生效。 取值范围： • tls-1-0 • tls-1-1 • tls-1-2 • tls-1-2-strict
tls	否	Array of strings	HTTPS 协议时，需添加此字段。该字段可添加多项独立的域名和证书，详见配置服务器名称指示（SNI）。
secretName	否	String	HTTPS 协议时添加，配置为创建的密钥证书名称。

表9-10 tls_ciphers_policy 取值说明

安全策略	支持的 TLS 版本类型	使用的加密套件列表

安全策略	支持的 TLS 版本类型	使用的加密套件列表
tls-1-0	TLS 1.2 TLS 1.1 TLS 1.0	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:AES128-SHA256:AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:AES128-SHA:AES256-SHA
tls-1-1	TLS 1.2 TLS 1.1	
tls-1-2	TLS 1.2	
tls-1-2-strict	TLS 1.2	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:AES128-SHA256:AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384

步骤 5 创建 Ingress。

kubectl create -f ingress-test.yaml

回显如下，表示 Ingress 服务已创建。

```
ingress/ingress-test created
```

查看已创建的 Ingress。

kubectl get ingress

回显如下，表示 Ingress 服务创建成功，工作负载可访问。

```
NAME           HOSTS    ADDRESS         PORTS   AGE
ingress-test     *       121.**.**.**     80      10s
```

步骤 6 访问工作负载（例如 Nginx 工作负载），在浏览器中输入安全访问地址 https://121.**.**.**:443 进行验证。

其中，121.**.**.**为统一负载均衡实例的 IP 地址。

----结束

使用 ELB 服务中的证书

使用 ELB 服务中的证书，可以通过指定 kubernetes.io/elb.tls-certificate-ids 这个 annotations 实现。

说明

1. 当同时指定 annotation 中已有证书和 IngressTLS 时，使用 ELB 服务中的证书。
2. CCE 不校验 ELB 服务中的证书是否有效，只校验证书是否存在。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: 0b9a6c4d-bd8b-45cc-bfc8-ff0f9da54e95
    kubernetes.io/elb.class: union
    kubernetes.io/elb.tls-certificate-ids:
058cc023690d48a3867ad69dbe9cd6e5,b98382b1f01c473286653afd1ed9ab63
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

使用 HTTP/2

Ingress 支持 HTTP/2 的方式暴露服务，在默认情况下，客户端与 LB 之间采用 HTTP1.X 协议，若需开启 HTTP/2 功能，可在 annotation 字段中加入如下配置：

`kubernetes.io/elb.http2-enable: 'true'`

以关联已有 ELB 为例，yaml 配置文件如下。

1.21 及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的 ELB ID
    kubernetes.io/elb.ip: <your_elb_ip>  #替换为您已有的 ELB IP
    kubernetes.io/elb.port: '443'
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.http2-enable: 'true' # 开启 HTTP/2 功能
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
```

```
      backend:
        serviceName: <your_service_name>  #替换为您的目标服务名称
        servicePort: 80                   #替换为您的目标服务端口
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

1.23 及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的 ELB ID
    kubernetes.io/elb.ip: <your_elb_ip>  #替换为您已有的 ELB IP
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.http2-enable: 'true' # 开启 HTTP/2 功能
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 8080           #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

表 6 HTTP/2 参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.http2-enable	否	Bool	表示 HTTP/2 功能的开启状态。开启后，可提升客户端与 LB 间的访问性能，但 LB 与后端服务器间仍采用 HTTP1.X 协议。**v1.19.16-r0、v1.21.3-r0 及以上版本的集群支持此字段**。 取值范围： • true：开启 HTTP/2 功能； • false：关闭 HTTP/2 功能（默认为关闭状态）。 注意：**只有当监听器的协议为 HTTPS 时，才支持开启或关闭 HTTP/2 功能**。当监听器的协议为 HTTP 时，该字段无效，默认将其设置为 false。

配置服务器名称指示（SNI）

SNI 允许同一个 IP 地址和端口号下对外提供多个基于 TLS 的访问域名，且不同的域名可以使用不同的安全证书。

说明

- 用于 SNI 的证书需要指定域名，每个证书只能指定一个域名。支持泛域名证书。
- 安全策略选择（kubernetes.io/elb.tls-ciphers-policy）仅在 1.17.11 及以上版本的集群中支持。

满足以上条件时可进行 SNI 配置，以自动创建关联 ELB 为例，yaml 文件配置如下，本例中 **sni-test-secret-1**、**sni-test-secret-2** 为 SNI 证书，该证书指定的域名必须与证书中的域名一致。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type":"public",
          "bandwidth_name":"cce-bandwidth-******",
          "bandwidth_chargemode":"bandwidth",
          "bandwidth_size":5,
          "bandwidth_sharetype":"PER",
          "eip_type":"5_bgp"
        }'
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  - hosts:
      - example.top  #签发证书时指定域名为 example.top
    secretName: sni-test-secret-1
  - hosts:
      - example.com  #签发证书时指定域名为 example.com
    secretName: sni-test-secret-2
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

路由到多个服务

Ingress 可通过不同的匹配策略同时路由到多个后端服务，YAML 文件中的 **spec** 字段设置如下。通过访问“www.example.com/foo”、“www.example.com/bar”、“foo.example.com/”即可分别路由到三个不同的后端 Service。

须知

Ingress 转发策略中注册的 URL 需与后端应用暴露的 URL 一致，否则将返回 404 错误。

```
spec:
  rules:
  - host: 'www.example.com'
    http:
      paths:
      - path: '/foo'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
      - path: '/bar'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
  - host: 'foo.example.com'
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

9.4.3 通过控制台使用 Nginx Ingress

前提条件

- Ingress 为后端工作负载提供网络访问，因此集群中需提前部署可用的工作负载。若您无可用工作负载，可参考创建无状态负载(Deployment)、创建有状态负载(StatefulSet)或创建守护进程集(DaemonSet)部署工作负载。
- 添加 Nginx Ingress 时，需在集群中提前安装 nginx-ingress 插件。

注意事项

- **不建议在 ELB 服务页面修改 ELB 实例的任何配置，否则将导致服务异常。**如果您已经误操作，请卸载 Nginx Ingress 插件后重装。

- Ingress 转发策略中注册的 URL 需与后端应用暴露的 URL 一致，否则将返回 404 错误。
- 负载均衡实例需与当前集群处于相同 VPC 且为相同公网或私网类型。
- 负载均衡实例需要拥有至少两个监听器配额，且端口 80 和 443 没有被监听器占用。

添加 Nginx Ingress

本节以 Nginx 作为工作负载并添加 Nginx Ingress 为例进行说明。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 选择左侧导航栏的“服务发现”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤 3 设置 Ingress 参数。

- **名称：**自定义 Ingress 名称，例如 nginx-ingress-demo。
- **命名空间：**选择需要添加 Ingress 的命名空间。
- **对接 Nginx：**集群中已安装 nginx-ingress 插件后显示此选项，未安装 nginx-ingress 模板时本选项不显示。

  单击  开启后将对接 nginx-ingress 提供 7 层访问，可配置如下参数。

  TLS 配置：nginx-ingress 支持 HTTP 和 HTTPS，安装 nginx-ingress 时预留的监听端口，默认 HTTP 为 80，HTTPS 为 443。使用 HTTPS 需要配置相关证书。

  - 服务器证书：创建 HTTPS 协议监听时需要绑定 IngressTLS 类型的密钥证书，以支持 HTTPS 数据传输加密认证，创建密钥的方法请参见创建密钥。
  - SNI：SNI（Server Name Indication）是 TLS 的扩展协议，在该协议下允许同一个 IP 地址和端口号下对外提供多个基于 TLS 的访问域名，且不同的域名可以使用不同的安全证书。开启 SNI 后，允许客户端在发起 TLS 握手请求时就提交请求的域名信息。负载均衡收到 TLS 请求后，会根据请求的域名去查找证书：若找到域名对应的证书，则返回该证书认证鉴权；否则，返回缺省证书（服务器证书）认证鉴权。
- **转发策略配置：**请求的访问地址与转发规则匹配时（转发规则由域名、URL 组成），此请求将被转发到对应的目标 Service 处理。单击“添加转发策略”按钮可添加多条转发策略。
  - 域名：实际访问的域名地址。请确保所填写的域名已注册并备案，在 Ingress 创建完成后，将域名与自动创建的负载均衡实例的 IP（即 Ingress 访问地址的 IP 部分）绑定。一旦配置了域名规则，则必须使用域名访问。
  - URL：需要注册的访问路径，例如：/healthz。社区 v1.2.0 版本（对应 CCE nginx-ingress 插件 2.1.0 版本）后修复 CVE-2021-25745(https://github.com/kubernetes/ingress-nginx/issues/8502)漏洞，新增规则（https://github.com/kubernetes/ingress-nginx/blame/main/internal/ingress/inspector/rules.go）禁用一些存在越权风险的访问路径。

说明

Nginx Ingress 的 URL 匹配规则是基于“/”符号分隔的路径前缀匹配，并区分大小写。只要访问路径以“/”符号分隔后的子路径匹配此前缀，均可正常访问，但如果该前缀仅是子路径中的部分字符串，则不会匹配。例如 URL 设置为/healthz，则匹配/healthz/v1，但不匹配/healthzv1。

- 目标服务名称：请选择已有 Service 或新建 Service。页面列表中的查询结果已自动过滤不符合要求的 Service。
- 目标服务访问端口：可选择目标 Service 的访问端口。
- 操作：可单击“删除”按钮删除该配置。

- **注解**：以“key: value”形式设置，可通过 Annotations 查询 nginx-ingress 支持的配置。社区 v1.2.0 版本（对应 CCE nginx-ingress 插件 2.1.0 版本）后修复 CVE-2021-25746(https://github.com/kubernetes/ingress-nginx/issues/8503)漏洞，新增规则（https://github.com/kubernetes/ingress-nginx/blame/main/internal/ingress/inspector/rules.go）禁用一些存在越权风险的 Anntotations 值。

步骤 4 配置完成后，单击“创建”。

创建完成后，在 Ingress 列表可查看到已添加的 Ingress。

----结束

9.5 DNS

9.5.1 DNS 概述

CoreDNS 介绍

创建集群时会安装 CoreDNS 插件，CoreDNS 是用来做集群内部域名解析。

在 kube-system 命名空间下可以查看到 CoreDNS 的 Pod。

```
$ kubectl get po --namespace=kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-7689f8bdf-295rk             1/1     Running   0          9m11s
coredns-7689f8bdf-h7n68             1/1     Running   0          11m
```

CoreDNS 安装成功后会成为 DNS 服务器，当创建 Service 后，CoreDNS 会将 Service 的名称与 IP 记录起来，这样 Pod 就可以通过向 CoreDNS 查询 Service 的名称获得 Service 的 IP 地址。

访问时通过 nginx.<namespace>.svc.cluster.local 访问，其中 nginx 为 Service 的名称，<namespace>为命名空间名称，svc.cluster.local 为域名后缀，在实际使用中，在同一个命名空间下可以省略<namespace>.svc.cluster.local，直接使用 ServiceName 即可。

使用 ServiceName 的方式有个主要的优点就是可以在开发应用程序时可以将 ServiceName 写在程序中，这样无需感知具体 Service 的 IP 地址。

CoreDNS 插件安装后也有一个 Service，在 kube-system 命名空间下，如下所示。

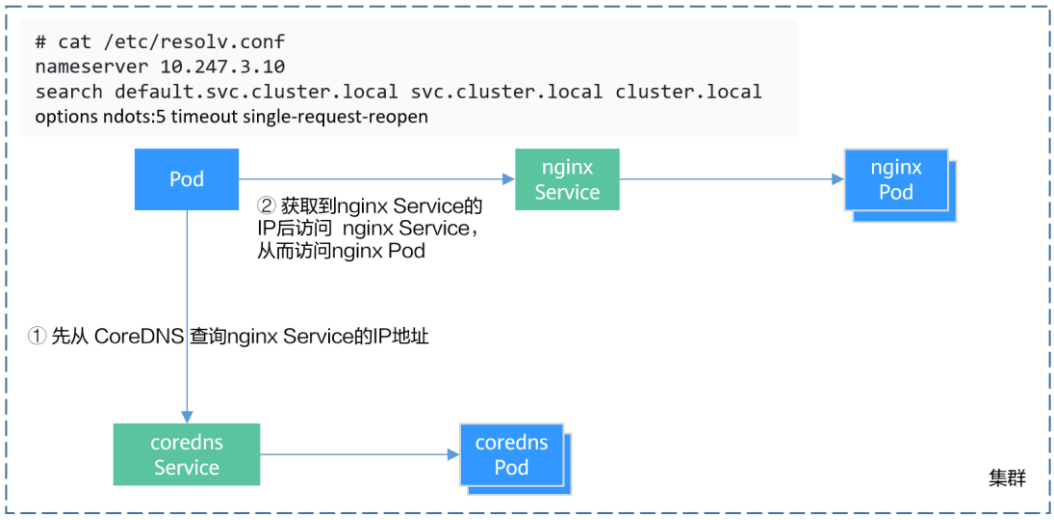
```
$ kubectl get svc -n kube-system
NAME             TYPE         CLUSTER-IP     EXTERNAL-IP                 PORT(S)
AGE
coredns          ClusterIP     10.247.3.10    <none>
53/UDP,53/TCP,8080/TCP      13d
```

默认情况下，其他 Pod 创建后，会将 coredns Service 的地址作为域名解析服务器的地址写在 Pod 的 /etc/resolv.conf 文件中，创建一个 Pod，查看/etc/resolv.conf 文件，如下所示。

```
$ kubectl exec test01-6cbbf97b78-krj6h -it -- /bin/sh
/ # cat /etc/resolv.conf
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5 timeout single-request-reopen
```

在 Pod 中访问 nginx Pod 的 ServiceName:Port，会先从 CoreDNS 中解析出 nginx Service 的 IP 地址，然后再访问 nginx Service 的 IP 地址，从而访问到 nginx Pod。

图9-31 集群内域名解析示例图



9.5.2 工作负载 DNS 配置说明

Kubernetes 集群内置 DNS 插件 Kube-DNS/CoreDNS，为集群内的工作负载提供域名解析服务。业务在高并发调用场景下，如果使用到域名解析服务，可能会触及到 Kube-DNS/CoreDNS 的性能瓶颈，导致 DNS 请求概率失败，影响用户业务正常运行。在 Kubernetes 使用的过程中，发现有些场景下工作负载的域名解析存在冗余的 DNS 查询，使得高并发场景更容易触及 DNS 的性能瓶颈。根据业务使用场景，对工作负载的 DNS 配置进行优化，能够在一定程度上减少 DNS 请求概率失败的问题。

更多 DNS 相关信息请参见 coredns（系统资源插件，必装）。

DNS 配置项说明

在 Linux 系统的节点或者容器里执行 cat /etc/resolv.conf 命令，能够查看到 DNS 配置，以 Kubernetes 集群的容器 DNS 配置为例：

```
nameserver 10.247.x.x
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

配置项说明：

- nameserver：容器解析域名时查询的 DNS 服务器的 IP 地址列表。如果设置为 10.247.x.x 说明 DNS 对接到 Kube-DNS/CoreDNS，如果是其他 IP 地址，则表示采用云上 DNS 或者用户自建的 DNS。
- search：定义域名的搜索域列表，当访问的域名不能被 DNS 解析时，会把该域名与搜索域列表中的域依次进行组合，并重新向 DNS 发起请求，直到域名被正确解析或者尝试完搜索域列表为止。对于 CCE 集群来说，容器的搜索域列表配置 3 个域，当解析一个不存在的域名时，会产生 8 次 DNS 查询，因为对于每个域名需要查询两次，分别是 IPv4 和 IPv6。
- options：定义域名解析配置文件的其他选项，常见的有 timeout、ndots 等等。

  Kubernetes 集群容器的域名解析文件设置为 options ndots:5，该参数的含义是当域名的“.”个数小于 ndots 的值，会先把域名与 search 搜索域列表进行组合后进行 DNS 查询，如果均没有被正确解析，再以域名本身去进行 DNS 查询。当域名的“.”个数大于或者等于 ndots 的值，会先对域名本身进行 DNS 查询，如果没有被正确解析，再把域名与 search 搜索域列表依次进行组合后进行 DNS 查询。

  如查询 www.***.com 域名时，由于该域名的“.”个数为 2，小于 ndots 的值，所以 DNS 查询请求的顺序依次为：www.***.default.svc.cluster.local、www.***.com.svc.cluster.local、 www.***.com.cluster.local 和 www.***.com，需要发起至少 7 次 DNS 查询请求才能解析出该域名的 IP。可以看出，这种配置在访问外部域名时，存在大量冗余的 DNS 查询，存在优化点。

说明

完整的 Linux 域名解析文件配置项说明可以参考文档：http://man7.org/linux/man-pages/man5/resolv.conf.5.html。

通过工作负载 YAML 进行 DNS 配置

您也可以通过 YAML 的方式创建工作负载，以 nginx 应用为例，其 YAML 文件中的 DNS 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-1
```

```
      image: nginx:latest
      imagePullPolicy: IfNotPresent
  imagePullSecrets:
    - name: default-secret
  dnsPolicy: None
  dnsConfig:
    options:
      - name: ndots
        value: '5'
      - name: timeout
        value: '3'
    nameservers:
      - 10.2.3.4
    searches:
      - my.dns.search.suffix
```

dnsPolicy 字段说明：

dnsPolicy 字段是应用设置的 DNS 策略，默认值为“ClusterFirst”。基于 dnsPolicy 策略生成的域名解析文件会与 dnsConfig 设置的 DNS 参数进行合并，合并规则将在下表中说明，dnsPolicy 当前支持四种参数值：

表9-11 dnsPolicy 字段说明

参数	说明
ClusterFirst（默认值）	应用对接 CoreDNS（CCE 集群的 CoreDNS 默认级联云上 DNS）。这种场景下，容器既能够解析 service 注册的集群内部域名，也能够解析发布到互联网上的外部域名。由于该配置下，域名解析文件设置了 search 搜索域列表和 ndots: 5，因此当访问外部域名和集群内部长域名（如 kubernetes.default.svc.cluster.local）时，大部分域名都会优先遍历 search 搜索域列表，导致至少有 6 次无效的 DNS 查询，只有访问集群内部短域名（如 kubernetes）时，才不存在无效的 DNS 查询。

ClusterFirstWithHostNet	对于配置主机网络的应用，即设置 hostNetwork 字段为 true 时，默认对接 kubelet 的“--resolv-conf”参数指向的域名解析文件（CCE 集群在该配置下对接云上 DNS）。如需对接集群的 Kube-DNS/CoreDNS，dnsPolicy 字段需设置为 ClusterFirstWithHostNet，此时容器的域名解析文件配置与“ClusterFirst”一致，也存在无效的 DNS 查询。
	`...`
	`spec:`
	`  containers:`
	`  - image: nginx:latest`
	`    imagePullPolicy: IfNotPresent`
	`    name: container-1`
	`  restartPolicy: Always`
	`  hostNetwork: true`
	`  dnsPolicy: ClusterFirstWithHostNet`

Default	容器的域名解析文件使用 kubelet 的“--resolv-conf”参数指向的域名解析文件（CCE 集群在该配置下对接云上 DNS），没有配置

参数	说明
	search 搜索域列表和 options。该配置只能解析注册到互联网上的外部域名，无法解析集群内部域名，且不存在无效的 DNS 查询。
None	设置为 None 之后，必须设置 dnsConfig 字段，此时容器的域名解析文件将完全通过 dnsConfig 的配置来生成。

说明

此处如果 dnsPolicy 字段未被指定，其默认值为 ClusterFirst，而不是 Default。

dnsConfig 字段说明：

dnsConfig 为应用设置 DNS 参数，设置的参数将合并到基于 dnsPolicy 策略生成的域名解析文件中。当 dnsPolicy 为“None”，应用的域名解析文件完全由 dnsConfig 指定；当 dnsPolicy 不为“None”时，会在基于 dnsPolicy 生成的域名解析文件的基础上，追加 dnsConfig 中配置的 dns 参数。

表9-12 dnsConfig 字段说明

参数	说明
options	DNS 的配置选项，其中每个对象可以具有 name 属性（必需）和 value 属性（可选）。该字段中的内容将合并到基于 dnsPolicy 生成的域名解析文件的 options 字段中，dnsConfig 的 options 的某些选项如果与基于 dnsPolicy 生成的域名解析文件的选项冲突，则会被 dnsConfig 所覆盖。
nameservers	DNS 的 IP 地址列表。当应用的 dnsPolicy 设置为“None”时，列表必须至少包含一个 IP 地址，否则此属性是可选的。列出的 DNS 的 IP 列表将合并到基于 dnsPolicy 生成的域名解析文件的 nameserver 字段中，并删除重复的地址。
searches	域名查询时的 DNS 搜索域列表，此属性是可选的。指定后，提供的搜索域列表将合并到基于 dnsPolicy 生成的域名解析文件的 search 字段中，并删除重复的域名。Kubernetes 最多允许 6 个搜索域。

通过控制台进行工作负载 DNS 配置

Kubernetes 为应用提供了与 DNS 相关的配置选项，通过对应用进行 DNS 配置，能够在某些场景下有效地减少冗余的 DNS 查询，提升业务并发量。以下步骤以 nginx 应用为例，介绍如何通过控制台为工作负载添加 DNS 配置。

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建负载”。

步骤 2 设置工作负载基本参数，详情请参见“创建无状态负载(Deployment)”。

步骤 3 在“高级配置”中，选择“DNS 配置”页签，并按需填写以下参数。

- DNS 策略：控制台中提供的 DNS 策略与 YAML 中的 dnsPolicy 字段对应。
  - 追加域名解析配置：即 dnsPolicy 字段设置为 ClusterFirst，此时容器中既能够解析 service 注册的集群内部域名，也能够解析发布到互联网上的外部域名。
  - 替换域名解析配置：即 dnsPolicy 字段设置为 None，此时必须填写“IP 地址”和“搜索域”参数。容器将仅使用自定义的 IP 地址和搜索域配置进行域名解析。
  - 继承 Pod 所在节点域名解析配置：即 dnsPolicy 字段设置为 Default，此时容器将使用 Pod 所在节点的域名解析配置，无法解析集群内部域名。
- 可选对象：即 dnsConfig 字段中的 options 参数。每个对象可以具有 name 属性（必需）和 value 属性（可选），填写完成后需单击“确认添加”。
  - timeout：超时时间 (s)。
  - ndots：域名中必须出现的"."的个数。如果域名中的"."的个数不小于 ndots，则该域名为一个全限定域名，操作系统会直接查询；如果域名中的"."的个数小于 ndots，操作系统会在搜索域中进行查询。
- IP 地址：即 dnsConfig 字段中的 nameservers 参数，您可对自定义的域名配置域名服务器，值为一个或一组 DNS IP 地址。
- 搜索域：即 dnsConfig 字段中的 searches 参数，表示域名查询时的 DNS 搜索域列表，此属性是可选的。指定后，提供的搜索域列表将合并到基于 dnsPolicy 生成的域名解析文件的 search 字段中，并删除重复的域名。

步骤 4 单击“创建工作负载”。

----结束

工作负载的 DNS 配置实践

前面介绍了 Linux 系统域名解析文件以及 Kubernetes 为应用提供的 DNS 相关配置项，下面将举例介绍应用如何进行 DNS 配置。

- **场景 1 对接 kubernetes 内置的 Kube-DNS/CoreDNS**

  场景说明：

  这种方式适用于应用中的域名解析只涉及集群内部域名，或者集群内部域名+外部域名两种方式，应用默认采用这种配置。

  示例：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
```

```
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: ClusterFirst
```

该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

- **场景 2 直接对接云 DNS**

  场景说明：

  这种方式适用于应用只访问注册到互联网的外部域名，该场景不能解析集群内部域名。

  示例：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: Default //使用 kubelet 的"--resolv-conf"参数指向的域名解析文件（CCE 集群在该配置下对接云 DNS）
```

  该配置下容器的域名解析文件将如下所示：

```
nameserver 100.125.x.x
```

- **场景 3 主机网络模式的应用对接 Kube-DNS/CoreDNS**

  场景说明：

  对于配置主机网络模式的应用，默认对接云 DNS，如果应用需要对接 Kube-DNS/CoreDNS，需将 dnsPolicy 设置为“ClusterFirstWithHostNet”。

  示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  containers:
  - name: nginx
    image: nginx:alpine
    ports:
    - containerPort: 80
```

  该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

- **场景 4 自定义应用的域名配置**

  场景说明：

  用户可以完全自定义配置应用的域名解析文件，这种方式非常灵活，dnsPolicy 和 dnsConfig 配合使用，几乎能够满足所有使用场景，如对接用户自建 DNS 的场景、串联多个 DNS 的场景以及优化 DNS 配置选项的场景等等。

  示例 1：对接用户自建 DNS

  该配置下，dnsPolicy 为“None”，应用的域名解析文件完全根据 dnsConfig 配置生成。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
    - 10.2.3.4 //用户自建 DNS 的 IP 地址
    searches:
    - ns1.svc.cluster.local
    - my.dns.search.suffix
    options:
    - name: ndots
      value: "2"
    - name: timeout
      value: "3"
```

  该配置下容器的域名解析文件将如下所示：

```
nameserver 10.2.3.4
search ns1.svc.cluster.local my.dns.search.suffix
options timeout:3 ndots:2
```

  示例 2：修改域名解析文件的 ndots 选项，减少无效的 DNS 查询

  该配置下，dnsPolicy 不为“None”，会在基于 dnsPolicy 生成的域名解析文件的基础上，追加 dnsConfig 中配置的 dns 参数。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: "ClusterFirst"
  dnsConfig:
```

```
    options:
    - name: ndots
      value: "2" //该配置会将基于 ClusterFirst 策略生成的域名解析文件的 ndots:5 参数改写为
ndots:2
```

该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:2
```

9.5.3 使用 CoreDNS 实现自定义域名解析

应用现状

在使用 CCE 时，可能会有解析自定义内部域名的需求，例如：

- 存量代码配置了用固定域名调用内部其他服务，如果要切换到 Kubernetes Service 方式，修改配置工作量大。
- 在集群外自建了一个其他服务，需要将集群中的数据通过固定域名发送到这个服务。

解决方案

使用 CoreDNS 有以下几种自定义域名解析的方案。

- 为 CoreDNS 配置存根域：可以直接在控制台添加，简单易操作。
- 使用 CoreDNS Hosts 插件配置任意域名解析：简单直观，可以添加任意解析记录，类似在本地/etc/hosts 中添加解析记录。
- 使用 CoreDNS Rewrite 插件指向域名到集群内服务：相当于给 Kubernetes 中的 Service 名称取了个别名，无需提前知道解析记录的 IP 地址。
- 使用 CoreDNS Forward 插件将自建 DNS 设为上游 DNS：自建 DNS 中，可以管理大量的解析记录，解析记录专门管理，增删记录无需修改 CoreDNS 配置。

注意事项

CoreDNS 修改配置需额外谨慎，因为 CoreDNS 负责集群的域名解析任务，修改不当可能会导致集群解析出现异常。请做好修改前后的测试验证。

为 CoreDNS 配置存根域

集群管理员可以修改 CoreDNS Corefile 的 ConfigMap 以更改服务发现的工作方式。

若集群管理员有一个位于 10.150.0.1 的 Consul 域名解析服务器，并且所有 Consul 的域名都带有.consul.local 的后缀。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“插件管理”，在“已安装插件”下，在 CoreDNS 下单击“编辑”，进入插件详情页。

步骤 3 在“参数配置”下添加存根域。

修改 stub_domains 参数，格式为一个键值对，键为 DNS 后缀域名，值为一个或一组 DNS IP 地址，如下所示。

```
{
    "stub_domains": {
             "consul.local": [
             "10.150.0.1"
         ]
    },
    "upstream_nameservers": []
}
```

步骤 4 单击“确定”。

----结束

也可以通过修改 ConfigMap，直接按如下方式添加。

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-
    .:5353 {
       bind {$POD_IP}
       cache 30
       errors
       health {$POD_IP}:8080
       kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
       }
       loadbalance round_robin
       prometheus {$POD_IP}:9153
       forward . /etc/resolv.conf {
          policy random
       }
       reload
    }

    consul.local:5353 {
       bind {$POD_IP}
       errors
       cache 30
       forward . 10.150.0.1
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2022-05-04T04:42:24Z"
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
```

```
  namespace: kube-system
  resourceVersion: "8663493"
  uid: bba87142-9f8d-4056-b8a6-94c3887e9e1d
```

修改 CoreDNS Hosts 配置

步骤 1 使用 kubectl 连接集群。

步骤 2 修改 CoreDNS 配置文件，将自定义域名添加到 hosts 中。

将 www.example.com 指向 192.168.1.1，通过 CoreDNS 解析 www.example.com 时，会返回 192.168.1.1。

须知

此处配置不能遗漏 fallthrough 字段，fallthrough 表示当在 hosts 找不到要解析的域名时，会将解析任务传递给 CoreDNS 的下一个插件。如果不写 fallthrough 的话，任务就此结束，不会继续解析，会导致集群内部域名解析失败的情况。

hosts 的详细配置请参见 https://coredns.io/plugins/hosts/。

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-
    .:5353 {
        bind {$POD IP}
        cache 30
        errors
        health {$POD IP}:8080
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        hosts {
          192.168.1.1 www.example.com
          fallthrough
        }
        loadbalance round_robin
        prometheus {$POD_IP}:9153
        forward . /etc/resolv.conf
        reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2021-08-23T13:27:28Z"
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
  namespace: kube-system
```

```
  resourceVersion: "460"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: be64aaad-1629-441f-8a40-a3efc0db9fa9
```

在 CoreDNS 中修改 hosts 后，就不用单独在每个 Pod 中配置 hosts 了，带来了一定的方便性。

----结束

添加 CoreDNS Rewrite 配置指向域名到集群内服务

使用 CoreDNS 的 Rewrite 插件，将指定域名解析到某个 Service 的域名，相当于给 Service 取了个别名。

步骤 1 使用 kubectl 连接集群。

步骤 2 修改 CoreDNS 配置文件，将 example.com 指向 default 命名空间下的 example 服务。

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-
    .:5353 {
        bind {$POD_IP}
        cache 30
        errors
        health {$POD_IP}:8080
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        rewrite name example.com example.default.svc.cluster.local
        loadbalance round_robin
        prometheus {$POD_IP}:9153
        forward . /etc/resolv.conf
        reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2021-08-23T13:27:28Z"
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
  namespace: kube-system
  resourceVersion: "460"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: be64aaad-1629-441f-8a40-a3efc0db9fa9
```

----结束

使用 CoreDNS 级联自建 DNS

步骤 1 使用 kubectl 连接集群。

步骤 2 修改 CoreDNS 配置文件，将 forward 后面的/etc/resolv.conf，改成外部 DNS 的地址，如下所示。

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-
    .:5353 {
        bind {$POD_IP}
        cache 30
        errors
        health {$POD_IP}:8080
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        loadbalance round_robin
        prometheus {$POD_IP}:9153
        forward . 192.168.1.1
        reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2021-08-23T13:27:28Z"
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
  namespace: kube-system
  resourceVersion: "460"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: be64aaad-1629-441f-8a40-a3efc0db9fa9
```

----结束

9.5.4 使用 NodeLocal DNSCache 提升 DNS 性能

应用现状

集群在做 DNS 解析时，如果请求量大，那 CoreDNS 将承受压力，会有如下影响。

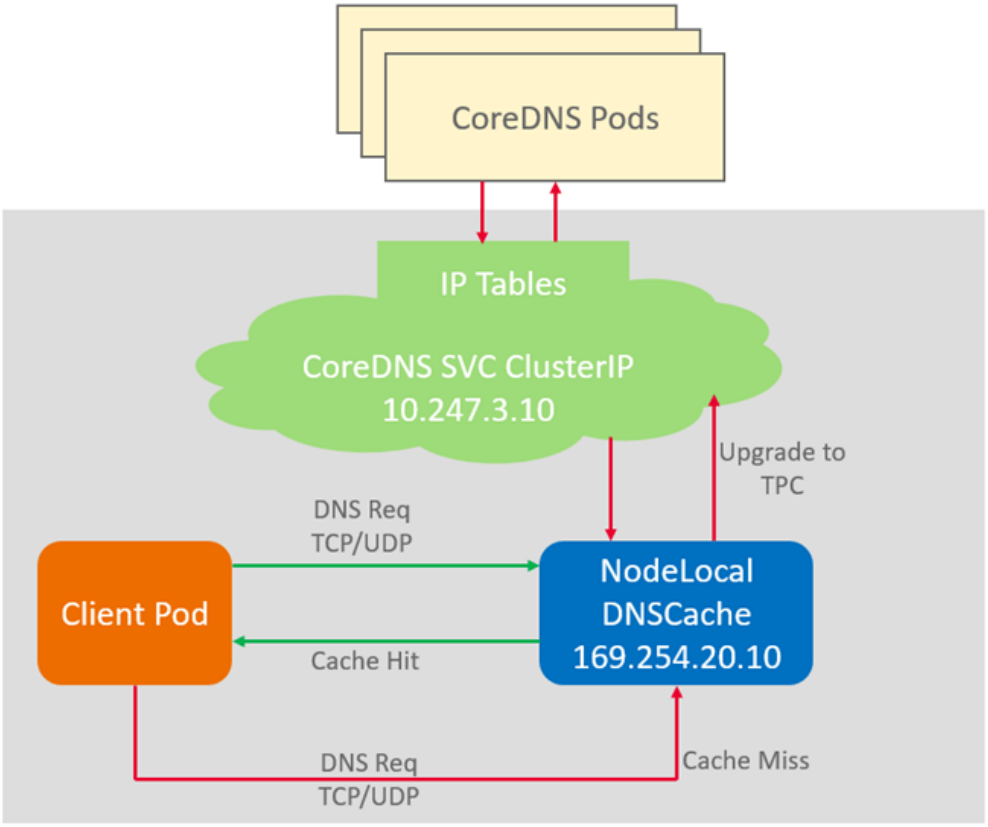
- 查询变慢，影响业务性能。
- 为保证性能，CoreDNS 需要更高规格的配置。

解决方案

NodeLocal DNSCache 通过在集群节点上运行 DNS 缓存代理来提高集群 DNS 性能。

启用 NodeLocal DNSCache 之后，DNS 查询所遵循的路径如下图所示。

图9-32 NodeLocal DNSCache 查询路径



插件安装

CCE 提供了 node-local-dns 插件，可以方便的安装 NodeLocal DNSCache。

说明

- node-local-dns 插件仅支持 1.19 及以上版本集群。
- NodeLocal DNSCache 不提供 Hosts、Rewrite 等插件能力，仅作为 CoreDNS 的透明缓存代理。如有需要，可在 CoreDNS 配置中修改。
- kube-system 命名空间下的 Pod 不支持自动注入。

步骤 1 （可选）修改 CoreDNS 配置，让 CoreDNS 优先采用 UDP 协议与上游 DNS 服务器通信。

NodeLocal DNSCache 采用 TCP 协议与 CoreDNS 进行通信，CoreDNS 会根据请求来源使用的协议与上游 DNS 服务器进行通信。当使用了 NodeLocal DNSCache 时，访问上游 DNS 服务器时会使用 TCP 协议，而云上 DNS 服务器对 TCP 协议支持有限，如果您

使用了 NodeLocal DNSCache，您需要修改 CoreDNS 配置，让其总是优先采用 UDP 协议与上游 DNS 服务器进行通信，避免解析异常。

执行如下命令修改。

kubectl edit configmap coredns -nkube-system

在 forward 插件中指定请求上游的协议为 perfer_udp，修改之后 CoreDNS 会优先使用 UDP 协议与上游通信。

```
forward . /etc/resolv.conf { prefer_udp }
```

步骤 2 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **node-local-dns**，单击“安装”。

步骤 3 在安装插件页面，选择插件规格，并配置相关参数。

enable_dnsconfig_admission：是否自动注入 DNSConfig 至新建的 Pod 中。默认为 **false**，设置为 **true** 表示支持自动注入，避免您手工配置 Pod YAML 进行注入。

步骤 4 完成以上配置后，单击“安装”。

----结束

使用 NodeLocal DNSCache

有两种方式可以使用 NodeLocal DNSCache：

- 自动注入：创建 Pod 时自动配置 Pod 的 dnsConfig 字段。（kube-system 命名空间下的 Pod 不支持自动注入）
- 手动配置：手动配置 Pod 的 dnsConfig 字段，从而使用 NodeLocal DNSCache。

自动注入

自动注入需要满足如下条件：

- 插件需要将 **enable_dnsconfig_admission** 参数配置为 **true**
- 命名空间添加 node-local-dns-injection=enabled 标签

  kubectl label namespace *default* node-local-dns-injection=enabled
- 新建 Pod 不位于 kube-system 和 kube-public 命名空间
- 新建 Pod 没有被打上禁用 DNS 注入 node-local-dns-injection=disabled 标签
- 新建 Pod 的网络为 hostNetwork 且 DNSPolicy 为 ClusterFirstWithHostNet，或 Pod 为非 hostNetwork 且 DNSPolicy 为 ClusterFirst

开启自动注入后，创建的 Pod 会自动添加如下 dnsConfig 字段，nameservers 中除了 NodeLocal DNSCache 的地址 169.254.20.10 外，还添加了 CoreDNS 的地址 10.247.3.10，保障了业务 DNS 请求高可用。

```
dnsConfig:
  nameservers:
    - 169.254.20.10
    - 10.247.3.10
  searches:
    - default.svc.cluster.local
```

```
    - svc.cluster.local
    - cluster.local
  options:
    - name: timeout
      value: ''
    - name: ndots
      value: '5'
    - name: single-request-reopen
```

手动配置

手动配置即自行给 Pod 加上 dnsConfig 配置。

创建一个 Pod，将 dnsconfig 配置为 169.254.20.10。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
  dnsConfig:
    nameservers:
    - 169.254.20.10
    searches:
    - default.svc.cluster.local
    - svc.cluster.local
    - cluster.local
    options:
    - name: ndots
      value: '2'
  imagePullSecrets:
  - name: default-secret
```

9.6 容器网络配置

9.6.1 主机网络 hostNetwork

背景信息

Kubenetes 支持 Pod 直接使用主机/节点的网络，对于需要直接访问主机网络的场景有一定的用途。

配置说明

Pod 使用主机网络只需要在配置中添加 hostNetwork: true 即可，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      hostNetwork: true
      containers:
      - image: nginx:alpine
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

部署后可以看到 Pod 的 IP 与节点的 IP 相同，说明 Pod 直接使用了主机网络。

```
$ kubectl get pod -owide
NAME                 READY   STATUS    RESTARTS   AGE     IP         NODE
NOMINATED NODE   READINESS GATES
nginx-6fdf99c8b-6wwft   1/1     Running   0          3m41s   10.1.0.55   10.1.0.55
<none>           <none>
```

hostNetwork 使用注意事项

Pod 直接使用主机的网络会占用宿主机的端口，Pod 的 IP 就是宿主机的 IP，使用时需要考虑是否与主机上的端口冲突，因此一般情况下除非您知道需要某个特定应用占用宿主机上的特定端口时，不建议使用主机网络。

由于使用主机网络，访问 Pod 就是访问节点，要**注意放通节点安全组端口**，否则会出现访问不通的情况。

另外由于占用主机端口，使用 Deployment 部署 hostNetwork 类型 Pod 时，要注意 **Pod 的副本数不要超过节点数量**，否则会导致一个节点上调度了多个 Pod，Pod 启动时端口冲突无法创建。例如上面例子中的 nginx，如果服务数为 2，并部署在只有 1 个节点的集群上，就会有一个 Pod 无法创建，查询 Pod 日志会发现是由于端口占用导致 nginx 无法启动。

⚠ 注意

请避免在同一个节点上调度多个使用主机网络的 Pod，否则在创建 ClusterIP 类型的 Service 访问 Pod 时，会出现访问 ClusterIP 不通的情况。

```
$ kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/2     2            1           67m
$ kubectl get pod
NAME                 READY   STATUS             RESTARTS   AGE
nginx-6fdf99c8b-6wwft   1/1     Running            0          67m
nginx-6fdf99c8b-rglm7   0/1     CrashLoopBackOff   13         44m
$ kubectl logs nginx-6fdf99c8b-rglm7
```

```
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: still could not bind()
nginx: [emerg] still could not bind()
```

9.6.2 Pod 互访 QoS 限速

操作场景

部署在同一节点上的不同业务容器之间存在带宽抢占，容易造成业务抖动。您可以通过对 Pod 间互访进行 QoS 限速来解决这个问题。

约束与限制

Pod 间互访设置 QoS 限速支持容器隧道网络模型、VPC 网络模型，其中 **VPC 网络模型**在使用 Pod 网络限速时需遵循以下约束：

- 仅支持 1.19.10 以上的集群版本。
- 仅支持普通容器（容器运行时为 runc），不支持安全容器（容器运行时为 kata）。
- 仅支持限制 Pod 访问 Pod 的场景限速，不对访问节点，对外访问产生影响。
- 带宽限速上限值为机型带宽上限和 4.3G 两者之间的最小值。
- 目前仅支持兆（M）级别以上的限速。

操作步骤

您可以通过对 Pod 添加 annotations 指定 Pod 出口带宽和入口带宽，如下所示。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/ingress-bandwidth: 100M
    kubernetes.io/egress-bandwidth: 100M
...
```

- kubernetes.io/ingress-bandwidth：Pod 的入口带宽
- kubernetes.io/egress-bandwidth：Pod 的出口带宽

如果不设置这两个参数，则表示不限制带宽。

9.6.3 容器隧道网络配置

9.6.3.1 网络策略（NetworkPolicy）

NetworkPolicy 是 Kubernetes 设计用来限制 Pod 访问的对象，通过设置 NetworkPolicy 策略，可以允许 Pod 被哪些地址访问（即入规则 Ingress）、或 Pod 访问哪些地址（即出规则 Egress）。这相当于从应用的层面构建了一道防火墙，进一步保证了网络安全。

NetworkPolicy 支持的能力取决于集群的网络插件的能力，如 CCE 的集群只支持设置 Pod 的入规则。

默认情况下，如果命名空间中不存在任何策略，则所有进出该命名空间中的 Pod 的流量都被允许。

NetworkPolicy 的规则可以选择如下 3 种：

- namespaceSelector：根据命名空间的标签选择，具有该标签的命名空间都可以访问。
- podSelector：根据 Pod 的标签选择，具有该标签的 Pod 都可以访问。
- ipBlock：根据网络选择，网段内的 IP 地址都可以访问。（CCE 当前不支持此种方式）

约束与限制

- 当前仅**容器隧道网络模型**的集群支持网络策略（NetworkPolicy）。
- 网络策略（NetworkPolicy）暂不支持设置出方向（egress）。
- 不支持对 IPv6 地址网络隔离。

使用 Ingress 规则

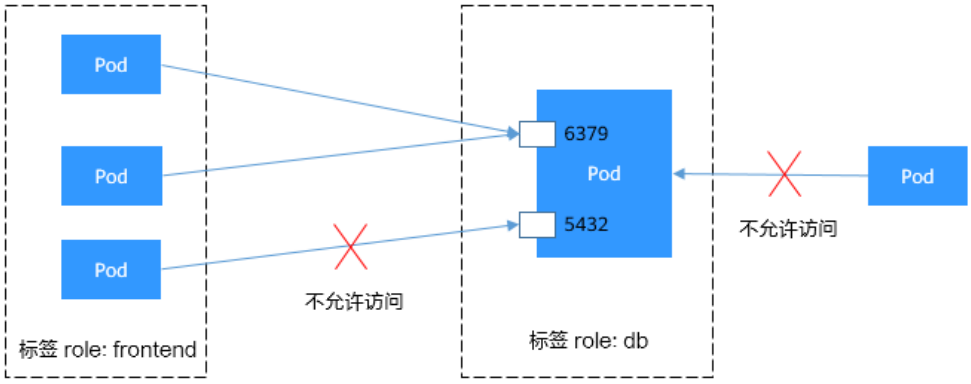
- **使用 podSelector 设置访问范围**

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
```

```
spec:
  podSelector:                 # 规则对具有 role=db 标签的 Pod 生效
    matchLabels:
      role: db
  ingress:                     # 表示入规则
  - from:
    - podSelector:             # 只允许具有 role=frontend 标签的 Pod 访问
        matchLabels:
          role: frontend
    ports:                     # 只能使用 TCP 协议访问 6379 端口
    - protocol: TCP
      port: 6379
```

示意图如下所示。

图9-33 podSelector

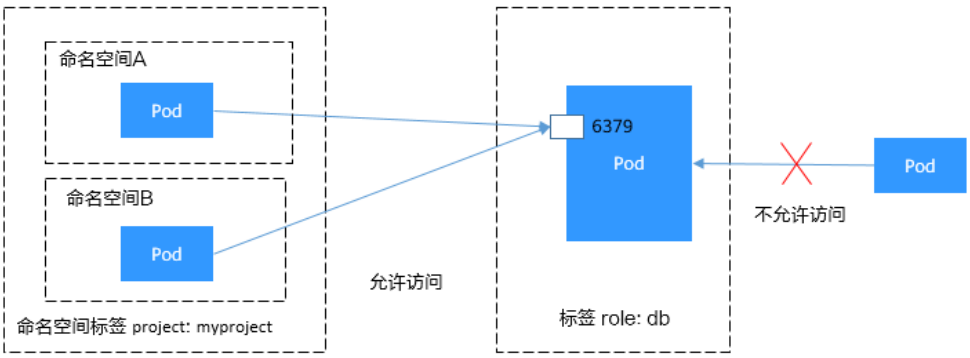


- **使用 namespaceSelector 设置访问范围**

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:                 # 规则对具有 role=db 标签的 Pod 生效
    matchLabels:
      role: db
  ingress:                     # 表示入规则
  - from:
    - namespaceSelector:         # 只允许具有 project=myproject 标签的命名空间中的 Pod 访问
        matchLabels:
          project: myproject
    ports:                     # 只能使用 TCP 协议访问 6379 端口
    - protocol: TCP
      port: 6379
```

示意图如下所示。

图9-34 namespaceSelector



在控制台创建网络策略

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“服务发现”，在右侧选择“网络策略”页签，单击右上角“创建网络策略”。

- 策略名称：自定义输入 NetworkPolicy 名称。
- 命名空间：选择网络策略所在命名空间。
- 选择器：输入标签选择要关联的 Pod，然后单击添加。您也可以单击“引用负载标签”直接引用已有负载的标签。
- 入方向规则：单击+添加入方向规则。

表9-13 添加规则

参数	参数说明
协议端口	请选择对应的协议类型和端口，目前支持 TCP 和 UDP 协议。
源对象命名空间	选择允许哪个命名空间的对象允许访问。
源对象 Pod 标签	选择允许带有这个标签的 Pod 访问。

步骤 3 设置完成后，单击“确定”。

----结束

9.7 容器如何访问 VPC 内部网络

前面章节介绍了使用 Service 和 Ingress 访问容器，本节将介绍如何从容器访问内部网络（**VPC 内集群外**），包括 VPC 内访问和跨 VPC 访问。

VPC 内访问

根据集群容器网络模型不同，从容器访问内部网络有不同表现。

- **容器隧道网络**

  容器隧道网络在节点网络基础上通过隧道封装网络数据包，容器访问同 VPC 下其他资源时，只要节点能访问通，容器就能访问通。如果访问不通，需要确认对端资源的安全组配置是否能够允许容器所在节点访问。

- **VPC 网络**

  VPC 网络使用了 VPC 路由功能来转发容器的流量，容器网段与节点 VPC 不在同一个网段，容器访问同 VPC 下其他资源时，**需要对端资源的安全组能够允许容器网段访问**。

  例如集群节点所在网段为 192.168.10.0/24，容器网段为 172.16.0.0/16。

  VPC 下（集群外）有一个地址为 192.168.10.52 的 ECS，其安全组规则仅允许集群节点的 IP 网段访问。

此时如果从容器中 ping 192.168.10.52，会发现无法 ping 通。

```
kubectl exec test01-6cbbf97b78-krj6h  -it -- /bin/sh
/ # ping 192.168.10.25
PING 192.168.10.25 (192.168.10.25): 56 data bytes
^C
--- 192.168.10.25 ping statistics ---
104 packets transmitted, 0 packets received, 100% packet loss
```

在安全组放通容器网段 172.16.0.0/16 访问。

此时再从容器中 ping 192.168.10.52，会发现可以 ping 通。

```
$ kubectl exec test01-6cbbf97b78-krj6h  -it -- /bin/sh
/ # ping 192.168.10.25
PING 192.168.10.25 (192.168.10.25): 56 data bytes
64 bytes from 192.168.10.25: seq=0 ttl=64 time=1.412 ms
64 bytes from 192.168.10.25: seq=1 ttl=64 time=1.400 ms
64 bytes from 192.168.10.25: seq=2 ttl=64 time=1.299 ms
64 bytes from 192.168.10.25: seq=3 ttl=64 time=1.283 ms
```

```
^C
--- 192.168.10.25 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
```

跨 VPC 访问

跨 VPC 访问通常采用对等连接等方法打通 VPC。

- 容器隧道网络只需将节点网络与对端 VPC 打通，容器自然就能访问对端 VPC。
- VPC 网络由于容器网段独立，除了要打通 VPC 网段，还要打通容器网段。

  例如有如下两个 VPC。

  - vpc-demo：网段为 192.168.0.0/16，集群在 vpc-demo 内，容器网段为 10.0.0.0/16。
  - vpc-demo2：网段为 10.1.0.0/16。

  创建一个名为 peering-demo 的对等连接（本端为 vpc-demo，对端为 vpc-demo2），注意对端 VPC 的路由添加容器网段，如下所示。

这样配置后，在 vpc-demo2 中就能够访问容器网段 10.0.0.0/16。具体访问时要关注安全组配置，打通端口配置。

访问其他云服务

与 CCE 进行内网通信的与服务常见服务有：RDS、DCS、Kafka、RabbitMQ、ModelArts 等。

访问其他云服务除了上面所说的 VPC 内访问和跨 VPC 访问的网络配置外，还**需要关注所访问的云服务是否允许外部访问**，如 DCS 的 Redis 实例，需要添加白名单才允许访问。通常这些云服务会允许同 VPC 下 IP 访问，但是 VPC 网络模型下容器网段与 VPC 网段不同，需要特殊处理，将容器网段加入到白名单中。

容器访问内网不通的定位方法

如前所述，从容器中访问内部网络不通的情况可以按如下路径排查：

1. 查看要访问的对端服务器安全组规则，确认是否允许容器访问。
   - 容器隧道网络模型需要放通容器所在节点的 IP 地址
   - VPC 网络模型需要放通容器网段
2. 查看要访问的对端服务器是否设置了白名单，如 DCS 的 Redis 实例，需要添加白名单才允许访问。添加容器和节点网段到白名单后可解决问题。

3. 查看要访问的对端服务器上是否安装了容器引擎，是否存在与 CCE 中容器网段冲突的情况。如果有网络冲突，会导致无法访问。

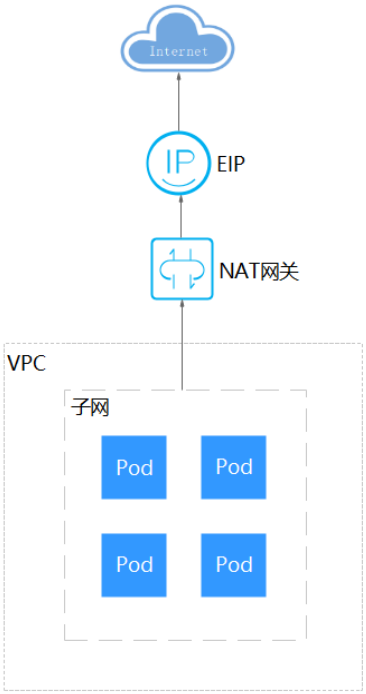
9.8 从容器访问公网

容器访问公网有如下方法可以实现。

- 给容器所在节点绑定公网 IP（容器网络模型为 VPC 网络或容器隧道网络）。
- 给 Pod IP 绑定公网 IP。
- 通过 NAT 网关配置 SNAT 规则，通过 NAT 网关访问公网。

下面将详细讲解通过 NAT 网关访问公网的方法，NAT 网关能够为 VPC 内的容器实例提供网络地址转换（Network Address Translation）服务，SNAT 功能通过绑定弹性公网 IP，实现私有 IP 向公有 IP 的转换，可实现 VPC 内的容器实例共享弹性公网 IP 访问 Internet。其原理如下图。通过 NAT 网关的 SNAT 功能，即使 VPC 内的容器实例不配置弹性公网 IP 也可以直接访问 Internet，提供超大并发数的连接服务，适用于请求量大、连接数多的服务。

图9-35 SNAT



您可以通过如下步骤实现容器实例访问 Internet。

步骤 1 创建弹性公网 IP。

1. 登录管理控制台。
2. 在管理控制台左上角单击 ，选择区域和项目。

3. 在控制台首页，单击左上角的，在展开的列表中单击“网络 > 弹性公网IP”。
4. 在“弹性公网 IP”界面，单击“购买弹性公网 IP”。
5. 根据界面提示配置参数。

说明

此处“区域”需选择容器实例所在区域。

步骤 2 创建 NAT 网关。

1. 登录管理控制台。
2. 在管理控制台左上角单击，选择区域和项目。
3. 在控制台首页，单击左上角的，在展开的列表中单击“网络 > NAT 网关”。
4. 在 NAT 网关页面，单击右上角的“购买公网 NAT 网关”。
5. 根据界面提示配置参数。

说明

此处需选择集群相同的 VPC。

步骤 3 配置 SNAT 规则，为子网绑定弹性公网 IP。

1. 登录管理控制台。
2. 在管理控制台左上角单击，选择区域和项目。
3. 在控制台首页，单击左上角的，在展开的列表中单击“网络 >NAT 网关”。
4. 在 NAT 网关页面，单击需要添加 SNAT 规则的 NAT 网关名称。
5. 在 SNAT 规则页签中，单击“添加 SNAT 规则”。
6. 根据界面提示配置参数。

说明

SNAT 规则是按网段生效，因为不同容器网络模型通信方式不同，此处子网需按如下规则选择。

- 容器隧道网络、VPC 网络：需要选择节点所在子网，即创建节点时选择的子网。

对于存在多个网段的情况，可以创建多个 SNAT 规则或选择自定义网段，只要网段能包含节点子网（容器隧道网络、VPC 网络）即可。

图9-36 配置 SNAT 规则

SNAT 规则配置完成后，您就可以从容器中访问公网了，从容器中能够 ping 通公网。

----结束

10 存储管理

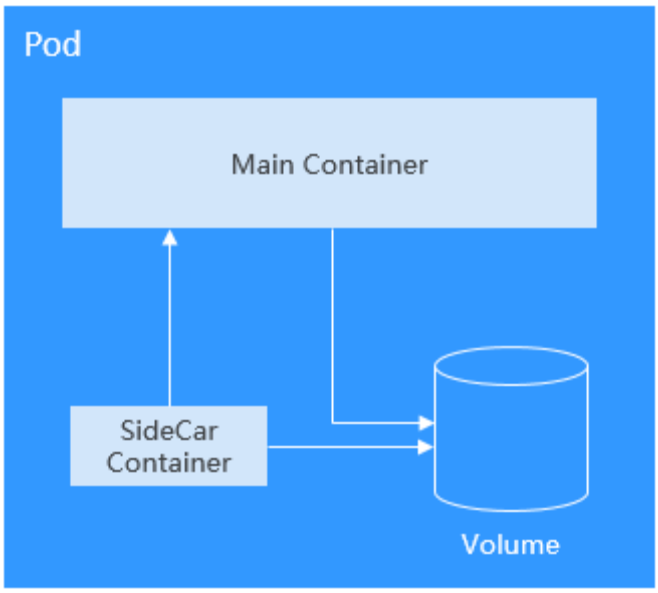
10.1 存储概述

Volume（存储卷）

容器中的文件在磁盘上是临时存放的，当容器重建时，容器中的文件将会丢失，另外当在一个 Pod 中同时运行多个容器时，常常需要在这些容器之间共享文件，这也是容器不好解决的问题。 Kubernetes 抽象出了 Volume 来解决这两个问题，也就是存储卷，Kubernetes 的 Volume 是 Pod 的一部分，Volume 不是单独的对象，不能独立创建，只能在 Pod 中定义。

Pod 中的所有容器都可以访问 Volume，但必须要挂载，且可以挂载到容器中任何目录。

实际中使用容器存储如下图所示，将容器的内容挂载到 Volume 中，通过 Volume 两个容器间实现了存储共享。



Volume 的生命周期与挂载它的 Pod 相同，但是 Volume 里面的文件可能在 Volume 消失后仍然存在，这取决于 Volume 的类型。

存储卷类型说明

Volume 可分为本地磁盘存储和云存储两大类。

- 本地磁盘存储

  本地磁盘存储可以使用如下几种类型，具体使用请参见本地磁盘存储。

  - emptyDir：一种简单的空目录，主要用于临时存储。
  - hostPath：将主机（节点）某个目录挂载到容器中，适用于读取主机上的数据。
  - ConfigMap：特殊类型，将 Kubernetes 特定的对象类型挂载到容器。
  - Secret：特殊类型，将 Kubernetes 特定的对象类型挂载到容器。
  - LocalPV：本地持久卷，直接使用节点的本地磁盘，持久化存储容器数据。
- 云存储：

  CCE 支持使用云存储有如下几种。

  - 云硬盘 EVS
  - 极速文件存储 SFS Turbo
  - 对象存储 OBS
  - 弹性文件存储 SFS

CSI

Kubernetes 提供了 CSI 接口（Container Storage Interface，容器存储接口），基于 CSI 这套接口，可以开发定制出 CSI 插件，从而支持特定的存储，达到解耦的目的。

CCE 开发了存储插件 Everest，在创建集群时必须安装，安装了插件就可以使用的 EVS、OBS 等云存储。

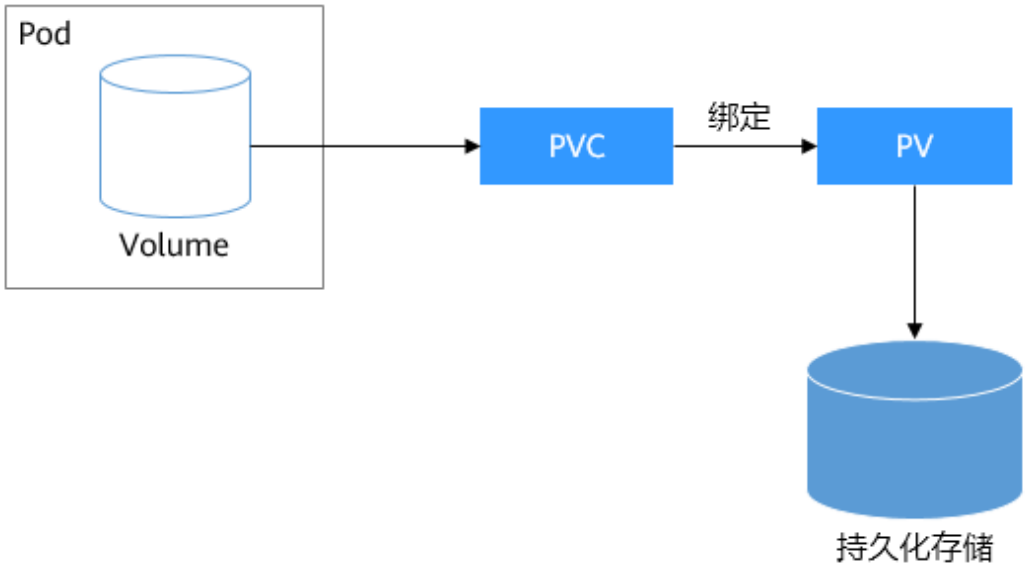
PV 与 PVC

Kubernetes 抽象了 PV（PersistentVolume）和 PVC（PersistentVolumeClaim）来定义和使用存储，从而让使用者不用关心具体的基础设施，当需要存储资源的时候，只要像 CPU 和内存一样，声明要多少即可。

- PV：PV 描述的是一个集群里的持久化存储卷，和节点一样，属于集群级别资源。在新版控制台（需要将集群升级到 1.19.10 并且 everest 存储插件升级到 1.2.10）PV 资源已经正式开放给用户管理；旧版控制台仍保持导入使用或者是通过动态创建方式进行创建，用户无法通过控制台对 PV 资源进行生命周期管理。
- PVC：PVC 描述的是负载对存储的申领，PVC 的申领会消耗集群中存量的 PV 资源，若集群中无存量 PV 资源，会动态创建底层存储及 PV 资源；创建 PVC 时，需描述请求的持久化存储的属性，比如，Volume 存储的大小、可读写权限等等。

在 Pod 中可以使用 Volume 关联 PVC，即可让 Pod 使用到存储资源，它们之间的关系如下图所示。

图10-1 PVC 绑定 PV



通常在使用时，可以使用 PV 描述已有的存储资源，然后创建 PVC 使用存储。具体使用可参见后续各章节中使用 kubectl 对接存储资源的内容。

而新创建存储资源时，可以使用一种更为方便的方法，可以跳过 PV 直接使用 PVC，这就是 StorageClass。

StorageClass

StorageClass 描述了集群中的存储类型“分类”，在创建 PVC/PV 均需要指定 StorageClass。目前 CCE 默认提供 csi-disk、csi-nas、csi-obs 等 StorageClass，在声明 PVC 时使用对应 StorageClassName，就可以创建对应类型 PV，并自动创建底层的存储资源。

执行如下命令即可查询 CCE 提供的默认 StorageClass。您可以使用 CCE 提供的 CSI 插件自定义创建 StorageClass，但从功能角度与 CCE 提供的默认 StorageClass 并无区别，这里不做过多描述。

```
# kubectl get sc
NAME              PROVISIONER                  AGE
csi-disk          everest-csi-provisioner      17d         # 云硬盘 StorageClass
csi-disk-topology   everest-csi-provisioner      17d         # 延迟绑定的云硬盘
StorageClass
csi-nas           everest-csi-provisioner      17d         # 文件存储 StorageClass
csi-obs           everest-csi-provisioner      17d         # 对象存储 StorageClass
csi-sfsturbo       everest-csi-provisioner      17d         # 极速文件存储
StorageClass
csi-local-topology  everest-csi-provisioner      17d         # 本地持久卷
```

定义了 StorageClass 后，就可以减少创建并维护 PV 的工作，PV 变成了自动创建，作为使用者，只需要在声明 PVC 时指定 StorageClassName 即可，这就大大减少工作量。

CCE 容器存储概览

CCE 支持工作负载 Pod 绑定存储卷类型说明中的多种本地磁盘存储和云存储，每种存储卷的主要特点及应用场景如下表：

图10-2 CCE 支持的存储类型

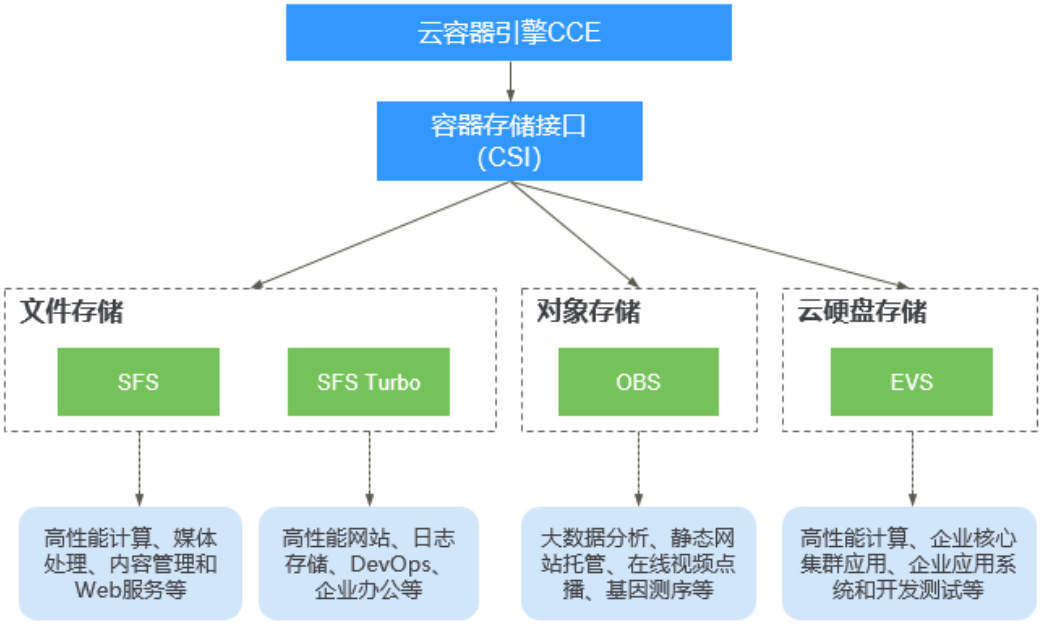


表10-1 网络存储对比

对比维度	云硬盘 EVS	弹性文件服务 SFS	对象存储 OBS	极速文件存储 SFS Turbo
概念	云硬盘（Elastic Volume Service）可以为云主机提供高可靠、高性能、规格丰富并且可弹性扩展的块存储服务，可满足不同场景的业务需求，适用于分布式文件系统、开发测试、数据仓库以及高性能计算等场景。	SFS 为用户提供一个完全托管的共享文件存储，能够弹性伸缩至 PB 规模，具备高可用性和持久性，为海量数据、高带宽型应用提供有力支持。适用于多种应用场景，包括 HPC、媒体处理、文件共享、内容管理和 Web 服务等。	对象存储服务（Object Storage Service，OBS）提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。适合企业备份/归档、视频点播、视频监控等多种数据存储场景。	SFS Turbo 为用户提供一个完全托管的共享文件存储，能够弹性伸缩至 320TB 规模，具备高可用性和持久性，为海量的小文件、低延迟高 IOPS 型应用提供有力支持。适用于多种应用场景，包括高性能网站、日志存储、压缩解压、DevOps、企业办公、容器应用等。

对比维度	云硬盘 EVS	弹性文件服务 SFS	对象存储 OBS	极速文件存储 SFS Turbo
存储数据的逻辑	存放的是二进制数据，无法直接存放文件，如果需要存放文件，需要先格式化文件系统后使用。	存放的是文件，会以文件和文件夹的层次结构来整理和呈现数据。	存放的是对象，可以直接存放文件，文件会自动产生对应的系统元数据，用户也可以自定义文件的元数据。	存放的是文件，会以文件和文件夹的层次结构来整理和呈现数据。
访问方式	只能在 ECS/BMS 中挂载使用，不能被操作系统应用直接访问，需要格式化成文件系统进行访问。	在 ECS/BMS 中通过网络协议挂载使用。需要指定网络地址进行访问，也可以将网络地址映射为本地目录后进行访问。	可以通过互联网或专线访问。需要指定桶地址进行访问，使用的是 HTTP 和 HTTPS 等传输协议。	提供标准的文件访问协议 NFS（仅支持 NFSv3），用户可以将现有应用和工具与 SFS Turbo 无缝集成。
静态数据卷	支持	支持	支持	支持
动态数据卷	支持	支持	支持	不支持
主要特点	非共享存储，每个云盘只能在单个节点挂载。	共享存储，可提供高性能、高吞吐存储服务。	共享存储，用户态文件系统。	高性能、高带宽、共享存储。
应用场景	HPC 高性能计算、企业核心集群应用、企业应用系统和开发测试等。 **说明** 高性能计算：主要是高速率、高 IOPS 的需求，用于作为高性能存储，比如工业设计、能源勘探等。	HPC 高性能计算、媒体处理、内容管理和 Web 服务、大数据和分析应用程序等。 **说明** 高性能计算：主要是高带宽的需求，用于共享文件存储，比如基因测序、图片渲染等。	大数据分析、静态网站托管、在线视频点播、基因测序、智能视频监控、备份归档、企业云盘（网盘）等。	高性能网站、日志存储、DevOps、企业办公等。
容量	TB 级别	PB 级别	EB 级别	TB 级别
时延	1~2ms	3~20ms	10ms	1~2ms
IOPS/TPS	单盘 33K	2K	千万级	100K
带宽	MB/s 级别	GB/s 级别	TB/s 级别	GB/s 级别

约束与限制

安全容器不支持使用对象存储卷。

- OBS 限制单用户创建 100 个桶，但是 CCE 使用 OBS 桶为单个工作负载挂载一个桶，当工作负载数量较多时，容易导致桶数量超过限制，OBS 桶无法创建。建议此种场景下直接通过 OBS 的 API 或 SDK 使用 OBS，不在工作负载中挂载 OBS 桶。
- 1.19.10 以下版本的集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，当新 Pod 被调度到另一个节点时，会导致之前 Pod 不能正常读写。

  1.19.10 及以上版本集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，新 Pod 会因为无法挂载云硬盘导致无法成功启动。
- 1.19 及以下版本的集群在卸载 subpath 时会遍历 subpath 下所有文件夹，若文件夹数量较多的情况下，遍历时间较长，卸卷时间对应较长。建议在 subpath 下不要建立太多文件夹，否则可能会出现 Pod 删除卸卷时间较长的情况。
- CCE 集群下挂载的 OBS 中单个文件大小限制远小于 obsfs 限制。

插件使用推荐

- 使用 CSI 插件（Everest）要求 Kubernetes 版本需为 **1.15 及以上**，v1.15 及以上版本的集群在创建时将默认安装本插件，v1.13 及以下版本集群创建时默认安装 Flexvolume 插件（storage-driver）。
- 集群版本由 v1.13 升级到 v1.15 后，v1.13 版本集群中的 Flexvolume 容器存储插件（storage-driver）能力将由 v1.15 的 CSI 插件（Everest，插件版本 v1.1.6 及以上）接管，接管后原有功能保持不变。
- 插件版本为 1.2.0 的 Everest 优化了使用 OBS 存储时的**密钥认证功能**，低于该版本的 Everest 插件在升级完成后，需要重启集群中使用 OBS 存储的全部工作负载，否则工作负载使用存储的能力将受影响。

CSI 和 Flexvolume 存储插件的区别

表10-2 CSI 与 Flexvolume

Kubernetes 插件方案	CCE 插件名称	插件特性	使用推荐
CSI	Everest	CSI 插件是 kubernetes 社区推荐的存储插件机制。CCE 发布的 kubernetes1.15 版本及以上版本默认安装 CSI 插件 Everest，并用于对接块存储、文件存储、对象存储、极速文件存储等 Iaas 存储服务。 Everest 插件包含两部分： • Everest-csi-controller：提供存储卷的创建、删除、扩	针对 **1.15 及以上**版本的集群，在创建时将默认安装 CSI 插件（Everest）。CCE 会跟随社区持续更新 CSI 插件的各种能力。

Kubernetes 插件方案	CCE 插件名称	插件特性	使用推荐
		容、云盘快照等功能； • Everest-csi-driver：提供存储卷在 node 上的挂载、卸载、格式化等功能。	
Flexvolume	storage-driver	Flexvolume 插件是 kubernetes 社区早期实现的存储卷插件机制。自 CCE 上线伊始，提供的就是 Flexvolume 数据卷服务。CCE 发布的 kubernetes 1.13 及以下版本安装的插件是“storage-driver”，并用于对接块存储、文件存储、对象存储、极速文件存储等 Iaas 存储服务。	针对已经创建的 **1.13 及以下**版本的集群，仍然使用已经安装的 Flexvolume 存储插件（storage-driver），CCE 已停止更新该插件。

说明

- 不支持 CSI 和 Flexvolume 插件在同一个集群中使用。
- 不支持将 v1.13 及以下版本集群的 Flexvolume 插件转变到 CSI 插件，v1.13 版本的集群可以通过升级集群版本切换为 CSI 插件。

如何判断集群的存储插件模式

步骤 1 登录 CCE 控制台。

步骤 2 在控制台左侧栏目树中，单击“插件管理”。

步骤 3 在右侧的插件管理列表中，单击“插件实例”页签。

步骤 4 在插件实例页面下，选择右上方的集群后，可以看到创建该集群时默认安装的存储插件。

----结束

10.2 本地磁盘存储

通过本地磁盘存储将容器所在宿主机的文件目录挂载到容器的指定路径中（对应 Kubernetes 的 HostPath），也可以不填写源路径（对应 Kubernetes 的 EmptyDir），不填写时将分配主机的临时目录挂载到容器的挂载点，指定源路径的本地硬盘数据卷适用于将数据持久化存储到容器所在宿主机，EmptyDir（不填写源路径）适用于容器的临时存储。

本地磁盘使用场景

您可以通过如下四种形式使用本地磁盘存储：

- **主机路径挂载**：将容器所在宿主机的文件目录挂载到容器指定的挂载点中，如容器需要访问/etc/hosts 则可以使用 HostPath 映射/etc/hosts 等场景。
- **临时路径挂载**：用于临时存储，生命周期与容器实例相同。容器实例消亡时，EmptyDir 会被删除，数据会永久丢失。
- **配置项挂载**：将 ConfigMap 配置项中的 key 映射到容器中，可以用于挂载配置文件到指定容器目录。配置项（ConfigMap）是一种用于存储工作负载所需配置信息的资源类型，内容由用户决定。ConfigMap 的创建请参见创建配置项，具体使用请参见使用配置项。
- **密钥挂载**：将密钥中的数据挂载到容器的某一路径中。密钥是一种用于存储工作负载所需要认证信息、密钥的敏感信息等的资源类型，内容由用户决定。Secret 的创建请参见创建密钥，具体使用请参见使用密钥。

下面分别介绍如何通过这四种形式进行挂载。

主机路径(HostPath)挂载

主机路径(HostPath)挂载表示将主机上的路径挂载到指定的容器路径。通常用于：“容器工作负载程序生成的日志文件需要永久保存”或者“需要访问宿主机上 Docker 引擎内部数据结构的容器工作负载”。

步骤 1 登录 CCE 控制台。

步骤 2 在创建工作负载时，在“容器配置”中找到“数据存储”，选择“本地存储”，单击 ＋。

步骤 3 设置添加本地磁盘参数。

表10-3 卷类型选择主机路径挂载

参数	参数说明
存储类型	主机路径(HostPath)。
主机路径	输入主机路径，如/etc/hosts。 **说明** 请注意“主机路径”不能设置为根目录“/”，否则将导致挂载失败。挂载路径一般设置为： • /opt/xxxx（但不能为/opt/cloud） • /mnt/xxxx（但不能为/mnt/paas） • /tmp/xxx • /var/xxx （但不能为/var/lib、/var/script、/var/paas 等关键目录） • /xxxx（但不能和系统目录冲突，例如 bin、lib、home、root、boot、dev、etc、lost+found、mnt、proc、sbin、srv、tmp、var、media、opt、selinux、sys、usr 等） 注意不能设置为/home/paas、/var/paas、/var/lib、/var/script、/mnt/paas、/opt/cloud，否则会导致系统或节点安装失败。
添加容器挂载	配置如下参数： 1. 子路径：请输入子路径，如：tmp。 使用子路径挂载本地磁盘，实现在单一 Pod 中重复使用同一

参数	参数说明
	个 Volume。不填写时默认为根。 2. 挂载路径：请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限帐号启动，否则可能会造成宿主机高危文件被破坏。 3. 权限： • 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。 单击“添加容器挂载”可增加多条设置，单击“确定”完成配置。

----结束

临时路径(EmptyDir)挂载

临时路径(EmptyDir)挂载适用于临时存储、灾难恢复、共享运行时数据等场景，工作负载实例的删除或者迁移会导致临时路径被删除。

步骤 1 登录 CCE 控制台。

步骤 2 在创建工作负载时，在“容器配置”中找到“数据存储”，选择“本地存储”，单击 +。

步骤 3 选择本地磁盘类型为“临时路径挂载”，设置添加本地磁盘参数。

表10-4 卷类型选择临时路径挂载

参数	参数说明
存储类型	临时路径(EmptyDir)。
磁盘介质	• 默认：存储在硬盘上，适用于数据量大，读写效率要求低的场景。 • 内存（Memory）：可以提高运行速度，但存储容量受内存大小限制。适用于数据量少，读写效率要求高的场景。 • 临时存储卷（LocalVolume）：存储在临时存储卷，相比内存类型可获得更大的存储容量，且支持扩容，相比默认类型性能要更好。要求节点存在临时存储卷，临时存储卷可以在创建节点时添加，也可以后期导入，详细说明请参见使用临

参数	参数说明
	时存储卷。 说明 • Memory 的 EmptyDir 使用的是内存，注意内存大小，用超过了容易 oom。 • Memory 的 EmptyDir 的大小为实例规格的 50%，暂时无法更改。 • 不使用 Memory 的 EmptyDir 不会占用系统内存。
添加容器挂载	配置如下参数： 1. 子路径：请输入子路径，如：tmp。 使用子路径挂载本地磁盘，实现在单一 Pod 中重复使用同一个 Volume。不填写时默认为根。 2. 挂载路径：请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 须知 挂载高危目录的情况下，建议使用低权限帐号启动，否则可能会造成宿主机高危文件被破坏。 3. 权限： • 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。 单击“添加容器挂载”可增加多条设置，单击“确定”完成配置。

----结束

配置项(ConfigMap)挂载

配置项(ConfigMap)挂载是将配置项中的数据挂载到指定的容器路径。平台提供工作负载代码和配置文件的分离，“配置项挂载”用于处理工作负载配置参数。用户需要提前创建工作负载配置，操作步骤请参见创建配置项。

步骤 1 登录 CCE 控制台。

步骤 2 在创建工作负载时，在“容器配置”中找到“数据存储”，选择“本地存储”，单击 +。

步骤 3 选择本地磁盘类型为“配置项挂载”，设置添加本地磁盘参数。

表10-5 卷类型选择配置项挂载

参数	参数说明
存储类型	配置项(ConfigMap)。
配置项	选择对应的配置项名称。 配置项需要提前创建，具体请参见创建配置项。
添加容器挂载	配置如下参数： 1. 子路径：请输入子路径，如：tmp。 • 使用子路径挂载本地磁盘，实现在单一 Pod 中重复使用同一个 Volume，不填写时默认为根。 • 子路径可以填写 ConfigMap/Secret 的键值，子路径若填写为不存在的键值则数据导入不会生效。 • 通过子路径导入的数据不会随 ConfigMap/Secret 的更新而动态更新。 2. 挂载路径：请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 须知 挂载高危目录的情况下，建议使用低权限帐号启动，否则可能会造成宿主机高危文件被破坏。 3. 设置权限：只读。只能读容器路径中的数据卷。 单击“添加容器挂载”可增加多条设置，单击“确定”完成配置。

----结束

密钥(Secret)挂载

密钥(Secret)挂载将密钥中的数据挂载到指定的容器路径，密钥内容由用户决定。用户需要提前创建密钥，操作步骤请参见创建密钥。

步骤 1 登录 CCE 控制台。

步骤 2 在创建工作负载时，在“容器配置”中找到“数据存储”，选择“本地存储”，单击 +。

步骤 3 选择本地磁盘类型为“密钥挂载”，设置添加本地磁盘参数。

表10-6 卷类型选择密钥挂载

参数	参数说明

参数	参数说明
存储类型	密钥(Secret)。
密钥	选择对应的密钥名称。 密钥需要提前创建，具体请参见创建密钥。
添加容器挂载	配置如下参数： 1. 子路径：请输入子路径，如：tmp。 • 使用子路径挂载本地磁盘，实现在单一 Pod 中重复使用同一个 Volume，不填写时默认为根。 • 子路径可以填写 ConfigMap/Secret 的键值，子路径若填写为不存在的键值则数据导入不会生效。 • 通过子路径导入的数据不会随 ConfigMap/Secret 的更新而动态更新。 2. 挂载路径：请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 须知 挂载高危目录的情况下 ，建议使用低权限帐号启动，否则可能会造成宿主机高危文件被破坏。 3. 设置权限：只读。只能读容器路径中的数据卷。 单击“添加容器挂载”可增加多条设置，单击“确定”完成配置。

----结束

10.3 存储卷 PV

PV 描述的是一个集群里的持久化存储卷，和节点一样，属于集群级别资源。

约束与限制

- 在新版控制台（需要将**集群升级到 1.19.10 及以上**并且 **Everest 存储插件升级到 1.2.10 及以上**）PV 资源已经正式开放给用户管理；旧版控制台仍保持导入使用或者是通过动态创建方式进行创建，用户无法通过控制台对 PV 资源进行生命周期管理。
- 支持多个 PV 挂载同一个 SFS 或 SFS Turbo，但有如下限制：
  - 多个不同的 PVC/PV 使用同一个底层 SFS 或 SFS Turbo 卷时，且被同一 Pod 使用会出现问题，需要避免这么使用。

  - PV 中 persistentVolumeReclaimPolicy 参数需设置为 Retain，否则可能存在一个 PV 删除时，级联删除底层卷，其他关联这个底层卷的 PV 会由于底层存储不在了，使用出现异常。
  - 重复用底层存储时，建议在应用层做好多读多写的隔离保护，防止产生的数据覆盖和丢失。
- 通过 YAML 创建 PV 时，存储资源如果为包周期资源，YAML 中的 persistentVolumeReclaimPolicy 字段请勿设置为 Delete，否则可能导致级联删除异常。关于 PV 回收策略详情请参见 PV 回收策略。

存储卷访问模式

PV 只能以底层存储资源所支持的方式挂载到宿主系统上。例如，文件存储可以支持多个节点读写，云硬盘只能被一个节点读写。

- ReadWriteOnce：卷可以被一个节点以读写方式挂载，云硬盘存储卷支持此类型。
- ReadWriteMany：卷可以被多个节点以读写方式挂载，文件存储、对象存储、极速文件存储支持此类型。

表10-7 云存储支持访问模式

存储类型	ReadWriteOnce	ReadWriteMany
云硬盘 EVS	√	×
文件存储 SFS	×	√
对象存储 OBS	×	√
极速文件存储 SFS Turbo	×	√

PV 回收策略

PV 回收策略用于指定删除 PVC 时，底层卷的回收策略，支持设定 Delete、Retain 回收策略。

- Delete：删除 PVC，PV 资源与底层存储资源均被删除。
- Retain：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。

Everest 还支持一种删除 PVC 时不删除底层存储资源的使用方法，当前仅支持使用 YAML 创建。PV 回收策略设置为 Delete，添加 annotations“everest.io/reclaim-policy: retain-volume-only”，这样删除 PVC，PV 会被删除，但底层存储资源会保留。

创建云硬盘 EVS 存储卷

说明

创建 EVS 存储卷有如下要求。

- 非系统盘，非专属盘，非共享盘。
- 云硬盘类型为支持类型（普通 IO，高 IO，超高 IO，通用 SSD），云硬盘模式为 SCSI。
- 云硬盘未冻结，云硬盘状态可用，云硬盘未被使用。
- 若云硬盘加密，所使用的密钥状态需可用。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷”页签。

步骤 3 单击右上角“创建存储卷”，在弹出的窗口中填写存储卷参数。

- 存储卷类型：选择“云硬盘”。
- 选择云硬盘。（仅支持选择集群所属企业项目和 default 企业项目下的云硬盘）。
- PV 名称：输入 PV 名称。
- 访问模式：ReadWriteOnce。
- 回收策略：Delete 或 Retain，具体解释请参见 PV 回收策略。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only        # 可选字段，删除 PV，保留底层存储卷
  name: cce-evs-test
  labels:
    failure-domain.beta.kubernetes.io/region: cn-north-4
    failure-domain.beta.kubernetes.io/zone: cn-north-4b
spec:
  accessModes:
    - ReadWriteOnce     # 访问模式，云硬盘必须为 ReadWriteOnce
  capacity:
    storage: 10Gi       # 云硬盘的容量，单位为 Gi，取值范围 1-32768
  csi:
    driver: disk.csi.everest.io     # 挂载依赖的存储驱动
    fsType: ext4
    volumeHandle: 459581af-e78c-4356-9e78-eaf9cd8525eb   #云硬盘的 volumeID
    volumeAttributes:
      everest.io/disk-mode: SCSI          # 云硬盘的磁盘模式，仅支持 SCSI
      everest.io/disk-volume-type: SAS     # 云硬盘的类型
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/crypt-key-id: 0992dbda-6340-470e-a74e-4f0db288ed82    # 可选字段，加密密钥 ID，使用加密盘的时候填写
      everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，企业项目 ID，如果指定企业项目，则创建 PVC 时也需要指定相同的企业项目，否则 PVC 无法绑定 PV。
```

```
persistentVolumeReclaimPolicy: Delete    # 回收策略
storageClassName: csi-disk             # 存储类名称，云硬盘必须为 csi-disk
```

表10-8 关键参数说明

参数	描述
everest.io/reclaim-policy: retain-volume-only	可选字段 目前仅支持配置“retain-volume-only” everest 插件版本需 >= 1.2.9 且回收策略为 Delete 时生效。如果回收策略是 Delete 且当前值设置为“retain-volume-only”删除 PVC 回收逻辑为：删除 PV，保留底层存储卷。
failure-domain.beta.kubernetes.io/region	集群所在的 region。
failure-domain.beta.kubernetes.io/zone	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
volumeHandle	云硬盘的 volumeID。 **获取方法：**在云主机控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下单击“ID”后的复制图标即可获取云硬盘的 volumeID。
everest.io/disk-volume-type	云硬盘类型，全大写。 • SAS：高 I/O • SSD：超高 I/O • GPSSD：通用型 SSD • ESSD：极速型 SSD
everest.io/crypt-key-id	卷为加密卷时为必填，填写创建加密秘钥的 ID。
everest.io/enterprise-project-id	可选字段 云硬盘的企业项目 ID。如果指定企业项目，则创建 PVC 时也需要指定相同的企业项目，否则 PVC 无法绑定 PV。 **获取方法：**在云主机控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下找到“管理信息”中的企业项目，单击并进入对应的企业项目控制台，复制对应的 ID 值即可获取云硬盘所属的企业项目的 ID。
persistentVolumeReclaimPolicy	集群版本号>=1.19.10 且 everest 插件版本>=1.2.9 时正式开放回收策略支持。

参数	描述
	支持 Delete、Retain 回收策略。 **Delete**: • Delete 且不设置 everest.io/reclaim-policy：删除 PVC，PV 资源与云硬盘均被删除。 • Delete 且设置 everest.io/reclaim-policy=retain-volume-only：删除 PVC，PV 资源被删除，云硬盘资源会保留。 **Retain**：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。 如果数据安全性要求较高，建议使用 **Retain** 以免误删数据。

创建文件存储 SFS 存储卷

说明

- SFS 必须与集群在同一个 VPC 内。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷”页签。

步骤 3 单击右上角“创建存储卷”，在弹出的窗口中填写存储卷参数。

- 存储卷类型：选择“文件存储”。
- 选择文件存储资源。
- PV 名称：输入 PV 名称。
- 访问模式：ReadWriteMany。
- 回收策略：Delete 或 Retain，具体解释请参见 PV 回收策略。
- 挂载参数：文件存储卷支持设置挂载参数。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除 PV，保留底层存储卷
```

```
  name: cce-sfs-test
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，文件存储必须为 ReadWriteMany
  capacity:
    storage: 1Gi       # 文件存储容量大小
  csi:
    driver: nas.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: 30b3d92a-0bc7-4610-b484-534660db81be   # 文件存储的 ID
    volumeAttributes:
      everest.io/share-export-location: sfs-nas01.cn-north-4.ctcloud.com:/share-
436304e8
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-nas                # 存储类名称
  mountOptions: []                         # 挂载参数
```

表10-9 关键参数说明

参数	描述
everest.io/reclaim-policy: retain-volume-only	可选字段 目前仅支持配置“retain-volume-only” everest 插件版本需 >= 1.2.9 且回收策略为 Delete 时生效。如果回收策略是 Delete 且当前值设置为“retain-volume-only”删除 PVC 回收逻辑为：删除 PV，保留底层存储卷。
volumeHandle	文件存储的 ID。 获取方法：在 CCE 控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，在弹性文件服务列表中单击对应的弹性文件服务名称，在详情页中复制“ID”后的内容即可。
everest.io/share-export-location	文件存储的共享路径。 获取方法：在 CCE 控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，在弹性文件服务列表中可以看到“挂载地址”列，即为文件存储的共享路径。

mountOptions	挂载参数。
	不设置时默认配置为如下配置。
	`mountOptions:`
	`- vers=3`
	`- timeo=600`
	`- nolock`
	`- hard`

everest.io/crypt-key-id	卷为加密卷时为必填，填写创建加密秘钥的 ID。
persistentVolumeReclaimPolicy	集群版本号>=1.19.10 且 everest 插件版本>=1.2.9 时正式开放回收策略支持。

参数	描述
	支持 Delete、Retain 回收策略。 **Delete**: • Delete 且不设置 everest.io/reclaim-policy：删除 PVC，PV 资源与文件存储均被删除。 • Delete 且设置 everest.io/reclaim-policy=retain-volume-only：删除 PVC，PV 资源被删除，文件存储资源会保留。 **Retain**：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。 如果数据安全性要求较高，建议使用 **Retain** 以免误删数据。
storageClassName	存储类名称，支持配置 csi-nas。 • csi-nas：表示使用 SFS 1.0 容量型文件存储。

创建对象存储 OBS 存储卷

说明

安全容器不支持使用对象存储卷。

OBS 限制单用户创建 100 个桶，但是 CCE 使用 OBS 桶为单个工作负载挂载一个桶，当工作负载数量较多时，容易导致桶数量超过限制，OBS 桶无法创建。建议此种场景下直接通过 OBS 的 API 或 SDK 使用 OBS，不在工作负载中挂载 OBS 桶。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷”页签。

步骤 3 单击右上角“创建存储卷”，在弹出的窗口中填写存储卷参数。

- 存储卷类型：选择“对象存储”。
- 选择对象存储资源。
- PV 名称：输入 PV 名称。
- 访问模式：ReadWriteMany。
- 回收策略：Delete 或 Retain，具体解释请参见 PV 回收策略。
- 密钥：对象存储卷挂载支持设置自定义访问密钥（AK/SK），您可以使用 AK/SK 创建一个 Secret，然后挂载到 PV。详细说明请参见对象存储卷挂载设置自定义访问密钥（AK/SK）。
- 挂载参数：对象存储卷支持设置挂载参数。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only        # 可选字段，删除 PV，保留底层存储卷
  name: cce-obs-test
spec:
  accessModes:
  - ReadWriteMany                    # 访问模式，对象存储必须为 ReadWriteMany
  capacity:
    storage: 1Gi      # 存储容量，此处填写仅因为 PV 的格式需要，取值大小无实际意义，可任意填写，不会因为填写的值而限制实际使用 OBS 空间大小。
  csi:
    driver: obs.csi.everest.io         # 挂载依赖的存储驱动
    fsType: obsfs                    # 对象存储文件类型
    volumeHandle: cce-obs-bucket       # 对象存储的桶名称
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: cn-north-4
      everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，企业项目 ID，如果指定企业项目，则创建 PVC 时也需要指定相同的企业项目，否则 PVC 无法绑定 PV。
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    nodePublishSecretRef:
      name: test-user
      namespace: default
  persistentVolumeReclaimPolicy: Retain       # 回收策略
  storageClassName: csi-obs                  # 存储类名称，对象存储必须为 csi-obs
  mountOptions: []                          # 挂载参数
```

表10-10 关键参数说明

参数	描述
everest.io/reclaim-policy: retain-volume-only	可选字段 目前仅支持配置“retain-volume-only” everest 插件版本需 >= 1.2.9 且回收策略为 Delete 时生效。如果回收策略是 Delete 且当前值设置为“retain-volume-only”删除 PVC 回收逻辑为：删除 PV，保留底层存储卷。
fsType	文件类型，支持“obsfs”与“s3fs”，取值为 s3fs 时创建是 obs 对象桶，配套使用 s3fs 挂载；取值为 obsfs 时创建的是 obs 并行文件系统，配套使用 obsfs 挂载，推荐使用。
volumeHandle	对象存储的桶名称。
everest.io/obs-volume-type	存储类型，包括 STANDARD（标准桶）、WARM（低频访问桶）。

参数	描述
everest.io/region	OBS 存储区域。
everest.io/enterprise-project-id	可选字段 对象存储的企业项目 ID。如果指定企业项目，则创建 PVC 时也需要指定相同的企业项目，否则 PVC 无法绑定 PV。 **获取方法**：在对象存储服务控制台，单击左侧栏目树中的“桶列表”或“并行文件系统”，单击要对接的对象存储名称进入详情页，在“基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的 ID 值即可获取对象存储所属的企业项目的 ID。
nodePublishSecretRef	对象存储卷挂载支持设置自定义访问密钥（AK/SK），您可以使用 AK/SK 创建一个 Secret，然后挂载到 PV。详细说明请参见对象存储卷挂载设置自定义访问密钥（AK/SK）。
mountOptions	挂载参数。
persistentVolumeReclaimPolicy	集群版本号>=1.19.10 且 everest 插件版本>=1.2.9 时正式开放回收策略支持。 支持 Delete、Retain 回收策略。 **Delete**: • Delete 且不设置 everest.io/reclaim-policy：删除 PVC，PV 资源与对象存储均被删除。 • Delete 且设置 everest.io/reclaim-policy=retain-volume-only：删除 PVC，PV 资源被删除，对象存储资源会保留。 **Retain**：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。 如果数据安全性要求较高，建议使用 **Retain** 以免误删数据。

创建极速文件存储 SFS Turbo 存储卷

说明

SFS Turbo 必须与集群在同一个 VPC 内。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷”页签。

步骤 3 单击右上角“创建存储卷”，在弹出的窗口中填写存储卷参数。

- 存储卷类型：选择“极速文件存储”。
- 选择极速文件存储资源。
- PV 名称：输入 PV 名称。

- 访问模式：ReadWriteMany。
- 回收策略：Retain，具体解释请参见 PV 回收策略。
- 挂载参数：极速文件存储卷支持设置挂载参数。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
  name: cce-sfsturbo-test
spec:
  accessModes:
    - ReadWriteMany      # 访问模式，极速文件存储必须为 ReadWriteMany
  capacity:
    storage: 100.00Gi    # 极速文件存储容量大小
  csi:
    driver: sfsturbo.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: 6674bd0a-d760-49de-bb9e-805c7883f047      # 极速文件存储的 ID。
    volumeAttributes:
      everest.io/share-export-location: 192.168.0.85:/      # 极速文件存储的共享路径
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  persistentVolumeReclaimPolicy: Retain     # 回收策略
  storageClassName: csi-sfsturbo           # 存储类名称，极速文件存储必须为 csi-sfsturbo
  mountOptions: []                         # 挂载参数
```

表10-11 关键参数说明

参数	描述
volumeHandle	极速文件存储的 ID。 获取方法：在弹性文件服务控制台，单击 SFS Turbo 存储实例，在基本信息中可查看到 ID。
everest.io/share-export-location	极速文件存储的共享路径。

mountOptions	挂载参数。
	不设置时默认配置为如下配置。
	`mountOptions:`
	`- vers=3`
	`- timeo=600`
	`- nolock`
	`- hard`

persistentVolumeReclaimPolicy	集群版本号>=1.19.10 且 everest 插件版本>=1.2.9 时正式开放回收策略支持。

参数	描述
	支持 Delete、Retain 回收策略。 **Delete**: • Delete 且不设置 everest.io/reclaim-policy：删除 PVC，PV 资源与极速文件存储均被删除。 • Delete 且设置 everest.io/reclaim-policy=retain-volume-only：删除 PVC，PV 资源被删除，极速文件存储资源会保留。 **Retain**：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。 如果数据安全性要求较高，建议使用 **Retain** 以免误删数据。

10.4 存储卷声明 PVC

PVC 描述的是负载对存储卷的申领，PVC 的申领会消耗集群中存量的 PV 资源，若集群中无存量 PV 资源，会动态创建底层存储及 PV 资源；创建 PVC 时，需描述请求的持久化存储的属性，比如，Volume 存储的大小、可读写权限等等。

约束与限制

创建 PVC 会先匹配集群中是否有配置相同、且状态为可用状态的 PV，如果存在，PVC 将首先选择可匹配的，且为可用状态的 PV 进行绑定，在集群中无满足匹配条件的 PV 时，动态创建新的存储。

字段含义	pvc 字段	pv 字段	匹配逻辑
region	pvc.metadata.labels(failure-domain.beta.kubernetes.io/region 或者 topology.kubernetes.io/region)	pv.metadata.labels(failure-domain.beta.kubernetes.io/region 或者 topology.kubernetes.io/region)	同时定义/不被定义，若定义需要内容一致
zone	pvc.metadata.labels(failure-domain.beta.kubernetes.io/zone 或者 topology.kubernetes.io/zone)	pv.metadata.labels(failure-domain.beta.kubernetes.io/zone 或者 topology.kubernetes.io/zone)	同时定义/不被定义，若定义需要内容一致
云硬盘	pvc.metadata.annotations(everest.io/disk-	pv.spec.csi.volumeAttributes(everest.io/disk-volume-type)	同时定义/不被定义，若定义需要内

字段含义	pvc 字段	pv 字段	匹配逻辑
类型	volume-type)		容一致
秘钥 id	pvc.metadata.annotations(everest.io/crypt-key-id)	pv.spec.csi.volumeAttributes(everest.io/crypt-key-id)	同时定义/不被定义，若定义需要内容一致
企业项目 id	pvc.metadata.annotations(everest.io/enterprise-project-id)	pv.spec.csi.volumeAttributes(everest.io/enterprise-project-id)	同时定义/不被定义，若定义需要内容一致
accessMode	accessMode	accessMode	内容一致
存储类	storageclass	storageclass	内容一致

存储卷访问模式

PV 只能以底层存储资源所支持的方式挂载到宿主系统上。例如，文件存储可以支持多个节点读写，云硬盘只能被一个节点读写。

- ReadWriteOnce：卷可以被一个节点以读写方式挂载，云硬盘存储卷支持此类型。
- ReadWriteMany：卷可以被多个节点以读写方式挂载，文件存储、对象存储、极速文件存储支持此类型。

表10-12 支持访问模式

存储类型	ReadWriteOnce	ReadWriteMany
云硬盘 EVS	√	×
文件存储 SFS	×	√
对象存储 OBS	×	√
极速文件存储 SFS Turbo	×	√
本地持久卷 LocalPV	√	×

企业项目支持说明

说明

该功能需要 Everest 插件升级到 1.2.33 及以上版本。

- 使用存储类创建 PVC：

CCE 支持使用存储类创建云硬盘和对象存储类型 PVC 时指定企业项目，将创建的存储资源（云硬盘和对象存储）归属于指定的企业项目下，**企业项目可选为集群所属的企业项目或 default 企业项目**。

若不指定企业项目，则创建的存储资源默认使用存储类 StorageClass 中指定的企业项目。

- 对于自定义的 StorageClass，可以在 StorageClass 中指定企业项目，详见指定 StorageClass 的企业项目。StorageClass 中如不指定的企业项目，则默认为 default 企业项目。
- 对于 CCE 提供的 csi-disk 和 csi-obs 存储类，所创建的存储资源属于 default 企业项目。

- 使用存储卷 PV 创建 PVC：

  使用 PV 创建 PVC 时，因为存储资源在创建时已经指定了企业项目，如果 PVC 中指定企业项目，则务必确保在 PVC 和 PV 中指定的 everest.io/enterprise-project-id 保持一致，否则两者无法正常绑定。

使用存储类创建 PVC

说明

- 使用存储类创建的底层云硬盘、文件存储和对象存储均为按需计费模式。

存储类（StorageClass）描述了集群中的存储类型“分类”，在创建 PVC 需要可以指定 StorageClass，动态创建 PV 及底层存储资源。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷声明”页签。

步骤 3 单击右上角“创建存储卷声明”，在弹出的窗口中填写存储卷“声明”参数。

- 存储卷声明类型：请根据您的需求进行选择。
- PVC 名称：指定 PVC 的名称。
- 创建方式：选择“动态创建”。
- 存储类：选择需要的存储类型。当前支持如下几类存储动态创建。
  - csi-disk：云硬盘。
  - csi-local-topology：本地持久卷。
  - csi-obs：对象存储。
- 可用区（仅云硬盘支持）：选择云硬盘所在可用区。
- 云硬盘类型（仅云硬盘支持）：选择云硬盘的类型。云硬盘类型在不同区域会有所不同。
  - 高 I/O
  - 超高 I/O
  - 通用型 SSD
  - 极速型 SSD

- 访问模式：ReadWriteOnce 和 ReadWriteMany，具体请参见存储卷访问模式。
- 存储池（仅本地持久卷支持）：显示支持本地持久卷的节点，具体请参见本地持久存储卷和临时存储卷。
- 容量（仅云硬盘和文件存储支持）：存储的容量大小。仅云硬盘和文件存储需要配置，对象存储无需配置。
- 加密（仅云硬盘和文件存储支持）：勾选底层存储是否加密，勾选后需要选择使用的加密密钥。仅云硬盘和文件存储支持加密。
- 密钥（仅对象存储支持）：对象存储需要选择访问密钥，具体使用请参见对象存储卷挂载设置自定义访问密钥（AK/SK）。
- 企业项目（仅云硬盘和对象存储支持）：集群所属的企业项目或 default 企业项目。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

云硬盘 YAML 示例。

- failure-domain.beta.kubernetes.io/region：集群所在的 region。
- failure-domain.beta.kubernetes.io/zone：创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
- everest.io/enterprise-project-id：企业项目 ID。仅支持集群所属企业项目和 default 企业项目，"0"表示 default 企业项目。

  获取方法：在 CCE 控制台，单击左侧栏目树中的“集群管理”，选择集群，并进入指定的集群详情页，在“基本信息”页签下找到企业项目，点击并进入对应的企业项目控制台，复制对应的 ID 值即可获取集群所属的企业项目的 ID。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-auto-example
  namespace: default
  annotations:
    everest.io/disk-volume-type: SSD    # 云硬盘的类型
    everest.io/crypt-key-id: 0992dbda-6340-470e-a74e-4f0db288ed82  # 可选字段，密钥的id，使用该密钥加密云硬盘
    everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，企业项目 id，Everest 需升级到 1.2.33 及以上版本，仅支持集群所属企业项目和 default 企业项目，"0"表示 default 企业项目。
  labels:
    failure-domain.beta.kubernetes.io/region: cn-north-4
    failure-domain.beta.kubernetes.io/zone: cn-north-4b
spec:
  accessModes:
  - ReadWriteOnce             # 云硬盘必须为 ReadWriteOnce
  resources:
    requests:
      storage: 10Gi           # 云硬盘大小，取值范围 1-32768
  storageClassName: csi-disk    # StorageClass 类型为云硬盘
```

本地持久卷 YAML 示例，本地持久卷使用要求在节点上导入了本地持久卷，具体请参见本地持久存储卷和临时存储卷。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-local-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce              # 必须为 ReadWriteOnce
  resources:
    requests:
      storage: 10Gi            # 本地持久存储卷大小
  storageClassName: csi-local-topology    # StorageClass 类型为 csi-local-topology
```

对象存储 YAML 示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: obs-warm-provision-pvc
  namespace: default
  annotations:
    everest.io/obs-volume-type: STANDARD      # 桶类型，当前支持标准（STANDARD）和低频（WARM）两种桶。
    csi.storage.k8s.io/fstype: obsfs          # 文件类型，obsfs 表示创建并行文件系统，推荐使用；s3fs 表示创建对象桶
    everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，企业项目 id，Everest 需升级到 1.2.33 及以上版本，仅支持集群所属企业项目和 default 企业项目，"0"表示 default 企业项目。
spec:
  accessModes:
  - ReadWriteMany                # 对象存储必须为 ReadWriteMany
  resources:
    requests:
      storage: 1Gi               # 此处仅为校验需要（不能为空和 0），设置的大小不起作用，此处设定为固定值 1Gi
  storageClassName: csi-obs        # StorageClass 类型为对象存储
```

使用存储卷 PV 创建 PVC

如果已经创建了 PV，则可以创建 PVC 申请 PV 的资源。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“存储卷声明”页签。

步骤 3 单击右上角“创建存储声明”，在弹出的窗口中填写存储卷“声明”参数。

- 存储卷声明类型：请根据您的需求进行选择。
- PVC 名称：指定 PVC 的名称。
- 创建方式：选择“已有存储卷”。

- 关联存储卷：选择要关联的存储卷，即 PV。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

云硬盘 YAML 示例。

- failure-domain.beta.kubernetes.io/region：集群所在的 region。
- failure-domain.beta.kubernetes.io/zone：创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS                                # 云硬盘的类型
    everest.io/crypt-key-id: fe0757de-104c-4b32-99c5-ee832b3bcaa3   # 可选字段，密钥的id，使用该密钥加密云硬盘
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，如果指定企业项目，需保证与 PV 中指定的 everest.io/enterprise-project-id 一致，否则无法绑定
  labels:
    failure-domain.beta.kubernetes.io/region: cn-north-4
    failure-domain.beta.kubernetes.io/zone: cn-north-4b
spec:
  accessModes:
  - ReadWriteOnce              # 云硬盘必须为 ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk     # StorageClass 的名称，云硬盘为 csi-disk
  volumeName: cce-evs-test       # PV 的名称
```

文件存储示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs-test
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany            # 文件存储必须为 ReadWriteMany
  resources:
    requests:
      storage: 100Gi          # PVC 申请容量大小
  storageClassName: csi-nas    # StorageClass 的名称
  volumeName: cce-sfs-test     # PV 的名称
```

对象存储示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-obs-test
  namespace: default
  annotations:
    everest.io/obs-volume-type: STANDARD                        # 桶类型，当前支持标准（STANDARD）和低频（WARM）两种桶。
    csi.storage.k8s.io/fstype: obsfs                            # 文件类型，obsfs 表示创建并行文件系统，推荐使用；s3fs 表示创建对象桶
    csi.storage.k8s.io/node-publish-secret-name: test-user
    csi.storage.k8s.io/node-publish-secret-namespace: default
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 可选字段，如果指定企业项目，需保证与 PV 中指定的 everest.io/enterprise-project-id 一致，否则无法绑定
spec:
  accessModes:
  - ReadWriteMany           # 对象存储必须为 ReadWriteMany
  resources:
    requests:
      storage: 1Gi          # PVC 申请容量大小，此处仅为校验需要（不能为空和 0），设置的大小不起作用，此处设定为固定值 1Gi
  storageClassName: csi-obs   # StorageClass 的名称，对象存储为 csi-obs
  volumeName: cce-obs-test    # PV 的名称
```

极速文件存储示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
spec:
  accessModes:
    - ReadWriteMany             # 极速文件存储必须为 ReadWriteMany
  resources:
    requests:
      storage: 100Gi            # PVC 申请容量大小
  storageClassName: csi-sfsturbo  # StorageClass 的名称，极速文件存储为 csi-sfsturbo
  volumeName: pv-sfsturbo-test        # PV 的名称
```

使用快照创建 PVC

通过快照创建云硬盘 PVC 时，磁盘类型、磁盘模式、加密属性需和快照源云硬盘保持一致。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“快照与备份”页签。

步骤 3 找到需要创建 PVC 的快照，单击“创建存储卷声明”，并在弹出窗口中指定 PVC 的名称。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: default
  annotations:
    everest.io/disk-volume-type: SSD     # 云硬盘类型，需要与快照源云硬盘保持一致
  labels:
    failure-domain.beta.kubernetes.io/region: cn-north-4
    failure-domain.beta.kubernetes.io/zone: cn-north-4b
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: '10'
  storageClassName: csi-disk
  dataSource:
    name: cce-disksnap-test            # 快照的名称
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

10.5 存储类 StorageClass

StorageClass 描述了集群中的存储类型“分类”，在创建 PVC/PV 均需要指定 StorageClass。目前 CCE 默认提供 csi-disk、csi-nas、csi-obs 等 StorageClass，在声明 PVC 时使用对应 StorageClassName，就可以自动创建对应类型 PV，并自动创建底层的存储资源。

执行如下命令即可查询 CCE 提供的默认 StorageClass。您可以使用 CCE 提供的 CSI 插件自定义创建 StorageClass，但从功能角度与 CCE 提供的默认 StorageClass 并无区别，这里不做过多描述。

```
# kubectl get sc
NAME              PROVISIONER                  AGE
csi-disk          everest-csi-provisioner       17d         # 云硬盘 StorageClass
csi-nas           everest-csi-provisioner       17d         # 文件存储 1.0
StorageClass
csi-sfs           everest-csi-provisioner       17d         # 文件存储 3.0
StorageClass
csi-obs           everest-csi-provisioner       17d         # 对象存储 StorageClass
csi-sfsturbo       everest-csi-provisioner       17d         # 极速文件存储
StorageClass
csi-local-topology  everest-csi-provisioner       17d         # 本地持久卷
```

定义了 StorageClass 后，就可以减少创建并维护 PV 的工作，PV 变成了自动创建，作为使用者，只需要在声明 PVC 时指定 StorageClassName 即可，这就大大减少工作量。

除了使用 CCE 提供的 StorageClass 外，您还可以自定义 StorageClass，使用自定义 StorageClass 有时能为使用带来一定的方便。下面将详细介绍这些的应用现状、解决方案以及自定义 StorageClass 的方法等。

应用现状

CCE 中使用存储时，最常见的方法是创建 PVC 时通过指定 StorageClassName 定义要创建存储的类型，如下所示，使用 PVC 申请一个 SAS（高 I/O）类型云硬盘/块存储。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-example
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
```

可以看到在 CCE 中如果需要指定云硬盘的类型，是通过 everest.io/disk-volume-type: SAS 字段指定，这里 SAS 是云硬盘的类型，代表高 I/O，还有 SSD（超高 I/O）可以指定。

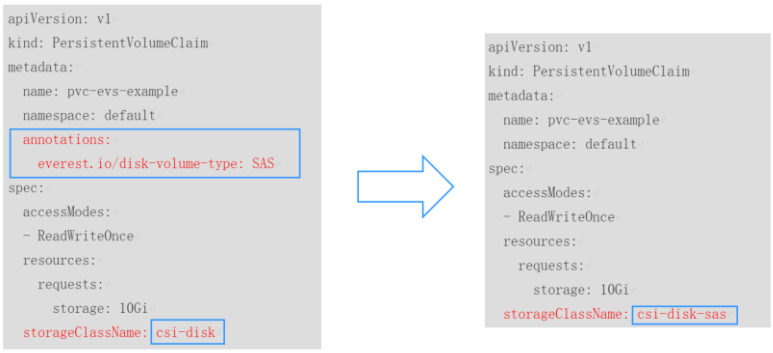
这种写法在如下几种场景下存在问题：

- 部分用户觉得使用 everest.io/disk-volume-type 指定云硬盘类型比繁琐，希望只通过 StorageClassName 指定。
- 部分用户是从自建 Kubernetes 或其他 Kubernetes 服务切换到 CCE，已经写了很多应用的 YAML 文件，这些 YAML 文件中通过不同 StorageClassName 指定不同类型存储，迁移到 CCE 上时，使用存储就需要修改大量 YAML 文件或 Helm Chart 包，这非常繁琐且容易出错。
- 部分用户希望能够设置默认的 StorageClassName，所有应用都使用默认存储类型，在 YAML 中不用指定 StorageClassName 也能按创建默认类型存储。

解决方案

本文介绍在 CCE 中自定义 StorageClass 的方法，并介绍设置默认 StorageClass 的方法，通过不同 StorageClassName 指定不同类型存储。

- 对于第一个问题：可以将 SAS、SSD 类型云硬盘分别定义一个 StorageClass，比如定义一个名为 csi-disk-sas 的 StorageClass，这个 StorageClass 创建 SAS 类型的存储，则前后使用的差异如下图所示，编写 YAML 时只需要指定 StorageClassName，符合特定用户的使用习惯。



未使用自定义StorageClass的写法　　使用自定义StorageClass的写法

- 对于第二个问题：可以定义与用户现有 YAML 中相同名称的 StorageClass，这样可以省去修改 YAML 中 StorageClassName 的工作。
- 对于第三个问题：可以设置默认的 StorageClass，则 YAML 中无需指定 StorageClassName 也能创建存储，按如下写法即可。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

自定义 StorageClass

自定义高 I/O 类型 StorageClass，使用 YAML 描述如下，这里取名为 csi-disk-sas，指定云硬盘类型为 SAS，即高 I/O。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-sas                          # 高 IO StorageClass 名字，用户可自定义
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS            # 云硬盘高 I/O 类型，用户不可自定义
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true                    # true 表示允许扩容
```

超高 I/O 类型 StorageClass，这里取名为 csi-disk-ssd，指定云硬盘类型为 SSD，即超高 I/O。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd                          # 超高 I/O StorageClass 名字，用户可自定义
```

```
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD        # 云硬盘超高 I/O 类型，用户不可自定义
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

reclaimPolicy：底层云存储的回收策略，支持 Delete、Retain 回收策略。

- **Delete**：删除 PVC，PV 资源与云硬盘均被删除。
- **Retain**：删除 PVC，PV 资源与底层存储资源均不会被删除，需要手动删除回收。PVC 删除后 PV 资源状态为“已释放（Released）”，不能直接再次被 PVC 绑定使用。

说明

此处设置的回收策略对 SFS Turbo 类型的存储无影响，因此删除集群或删除 PVC 时不会回收包周期的 SFS Turbo 资源。

如果数据安全性要求较高，建议使用 **Retain** 以免误删数据。

定义完之后，使用 kubectl create 命令创建。

```
# kubectl create -f sas.yaml
storageclass.storage.k8s.io/csi-disk-sas created
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
```

再次查询 StorageClass，回显如下，可以看到多了两个类型的 StorageClass。

```
# kubectl get sc
NAME              PROVISIONER                  AGE
csi-disk          everest-csi-provisioner      17d
csi-disk-sas      everest-csi-provisioner      2m28s
csi-disk-ssd      everest-csi-provisioner      16s
csi-disk-topology  everest-csi-provisioner     17d
csi-nas           everest-csi-provisioner      17d
csi-obs           everest-csi-provisioner      17d
csi-sfsturbo      everest-csi-provisioner      17d
```

其他类型存储自定义方法类似，可以使用 kubectl 获取 YAML，在 YAML 基础上根据需要修改。

- 文件存储

  ```
  # kubectl get sc csi-nas -oyaml
  kind: StorageClass
  apiVersion: storage.k8s.io/v1
  metadata:
    name: csi-nas
  provisioner: everest-csi-provisioner
  parameters:
    csi.storage.k8s.io/csi-driver-name: nas.csi.everest.io
    csi.storage.k8s.io/fstype: nfs
    everest.io/share-access-level: rw
  ```

```
  everest.io/share-access-to: 5e3864c6-e78d-4d00-b6fd-de09d432c632   # 集群所在
VPC ID
  everest.io/share-is-public: 'false'
  everest.io/zone: xxxxx         # 可用区
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

- 对象存储

```
# kubectl get sc csi-obs -oyaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-obs
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: obs.csi.everest.io
  csi.storage.k8s.io/fstype: s3fs          # 对象存储文件类型，s3fs 是对象桶，obsfs 是
并行文件系统
  everest.io/obs-volume-type: STANDARD      # OBS 桶的存储类别
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

指定 StorageClass 的企业项目

CCE 支持使用存储类创建云硬盘和对象存储类型 PVC 时指定企业项目，将创建的存储资源（云硬盘和对象存储）归属于指定的企业项目下，**企业项目可选为集群所属的企业项目或 default 企业项目**。

若不指定企业项目，则创建的存储资源默认使用存储类 StorageClass 中指定的企业项目，CCE 提供的 csi-disk 和 csi-obs 存储类，所创建的存储资源属于 default 企业项目。

如果您希望通过 StorageClass 创建的存储资源能与集群在同一个企业项目，则可以自定义 StorageClass，并指定企业项目 ID，如下所示。

说明

该功能需要 Everest 插件升级到 1.2.33 及以上版本。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-disk-epid      # 自定义名称
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 指定企业项
目 id
  everest.io/passthrough: 'true'
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

指定默认 StorageClass

您还可以指定某个 StorageClass 作为默认 StorageClass，这样在创建 PVC 时不指定 StorageClassName 就会使用默认 StorageClass 创建。

例如将 csi-disk-ssd 指定为默认 StorageClass，则可以按如下方式设置。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"   # 指定集群中默认的
StorageClass，一个集群中只能有一个默认的 StorageClass
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

先删除之前创建的 csi-disk-ssd，再使用 kubectl create 命令重新创建，然后再查询 StorageClass，显示如下。

```
# kubectl delete sc csi-disk-ssd
storageclass.storage.k8s.io "csi-disk-ssd" deleted
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
# kubectl get sc
NAME                  PROVISIONER                  AGE
csi-disk              everest-csi-provisioner       17d
csi-disk-sas          everest-csi-provisioner       114m
csi-disk-ssd (default)   everest-csi-provisioner       9s
csi-disk-topology      everest-csi-provisioner       17d
csi-nas               everest-csi-provisioner       17d
csi-obs               everest-csi-provisioner       17d
csi-sfsturbo          everest-csi-provisioner       17d
```

配置验证

- 使用 csi-disk-sas 创建 PVC。

  ```
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name:  sas-disk
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 10Gi
    storageClassName: csi-disk-sas
  ```

创建并查看详情，如下所示，可以发现能够创建，且 StorageClass 显示为 csi-disk-sas

```
# kubectl create -f sas-disk.yaml
persistentvolumeclaim/sas-disk created
# kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
sas-disk   Bound    pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c   10Gi       RWO
csi-disk-sas   24s
# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS     CLAIM                 STORAGECLASS   REASON   AGE
pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c   10Gi       RWO            Delete
Bound      default/sas-disk      csi-disk-sas            30s
```

在 CCE 控制台界面上查看 PVC 详情，在存储配置中可以看到磁盘类型是高 I/O。

- 不指定 StorageClassName，使用默认配置，如下所示，并未指定 storageClassName。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name:  ssd-disk
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

创建并查看，可以看到 PVC ssd-disk 的 StorageClass 为 csi-disk-ssd，说明默认使用了 csi-disk-ssd。

```
# kubectl create -f ssd-disk.yaml
persistentvolumeclaim/ssd-disk created
# kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
sas-disk   Bound    pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c   10Gi       RWO
csi-disk-sas   16m
ssd-disk   Bound    pvc-4d2b059c-0d6c-44af-9994-f74d01c78731   10Gi       RWO
csi-disk-ssd   10s
# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS     CLAIM                 STORAGECLASS   REASON   AGE
pvc-4d2b059c-0d6c-44af-9994-f74d01c78731   10Gi       RWO            Delete
Bound      default/ssd-disk      csi-disk-ssd            15s
pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c   10Gi       RWO            Delete
Bound      default/sas-disk      csi-disk-sas            17m
```

在 CCE 控制台界面上查看 PVC 详情，在存储配置中可以看到磁盘类型是超高 I/O。

10.6 快照与备份

CCE 通过云硬盘 EVS 服务为您提供快照功能，云硬盘快照简称快照，指云硬盘数据在某个时刻的完整拷贝或镜像，是一种重要的数据容灾手段，当数据丢失时，可通过快照将数据完整的恢复到快照时间点。

您可以创建快照，从而快速保存指定时刻云硬盘的数据。同时，您还可以通过快照创建新的云硬盘，这样云硬盘在初始状态就具有快照中的数据。

使用须知

- 快照功能**仅支持 v1.15 及以上版本**的集群，且需要安装基于 CSI 的 Everest 插件才可以使用。
- 基于快照创建的云硬盘，其子类型（普通 IO/高 IO/超高 IO）、是否加密、磁盘模式（VBD/SCSI）、共享性(非共享/共享)、容量等都要与快照关联母盘保持一致，这些属性查询和设置出来后不能够修改。
- 只有可用或正在使用状态，且存储格式为 CSI 的磁盘才能创建快照。快照免费试用期间，单个磁盘最大支持创建 7 个快照。
- 加密磁盘的快照数据以加密方式存放，非加密磁盘的快照数据以非加密方式存放。

使用场景

快照功能可以帮助您实现以下需求：

- **日常备份数据**

  通过对云硬盘定期创建快照，实现数据的日常备份，可以应对由于误操作、病毒以及黑客攻击等导致数据丢失或不一致的情况。

- **快速恢复数据**

  更换操作系统、应用软件升级或业务数据迁移等重大操作前，您可以创建一份或多份快照，一旦升级或迁移过程中出现问题，可以通过快照及时将业务恢复到快照创建点的数据状态。

  例如，当由于云主机 A 的系统盘 A 发生故障而无法正常开机时，由于系统盘 A 已经故障，因此也无法将快照数据回滚至系统盘 A。此时您可以使用系统盘 A 已有的快照新创建一块云硬盘 B 并挂载至正常运行的云主机 B 上，从而云主机 B 能够通过云硬盘 B 读取原系统盘 A 的数据。

  说明

  当前 CCE 提供的快照能力与 K8S 社区 CSI 快照功能一致：只支持基于快照创建新云硬盘，不支持将快照回滚到源云硬盘。

- **快速部署多个业务**

  通过同一个快照可以快速创建出多个具有相同数据的云硬盘，从而可以同时为多种业务提供数据资源。例如数据挖掘、报表查询和开发测试等业务。这种方式既保护了原始数据，又能通过快照创建的新云硬盘快速部署其他业务，满足企业对业务数据的多元化需求。

创建快照

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“快照与备份”页签。

步骤 3 单击右上角“创建快照”，在弹出的窗口中设置相关参数。

- 快照名称：填写快照的名称。
- 选择存储：选择要创建快照的 PVC，仅能创建云硬盘类型 PVC。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
kind: VolumeSnapshot
apiVersion: snapshot.storage.k8s.io/v1beta1
metadata:
  finalizers:
    - snapshot.storage.kubernetes.io/volumesnapshot-as-source-protection
    - snapshot.storage.kubernetes.io/volumesnapshot-bound-protection
  name: cce-disksnap-test
  namespace: default
spec:
  source:
    persistentVolumeClaimName: pvc-evs-test     # PVC 的名称，仅能创建云硬盘类型 PVC
  volumeSnapshotClassName: csi-disk-snapclass
```

使用快照创建 PVC

通过快照创建云硬盘 PVC 时，磁盘类型、磁盘模式、加密属性需和快照源云硬盘保持一致。

使用控制台创建

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“容器存储”，在右侧选择“快照与备份”页签。

步骤 3 找到需要创建 PVC 的快照，单击“创建存储卷声明”，并在弹出窗口中指定 PVC 的名称。

步骤 4 单击“创建”。

----结束

使用 YAML 创建

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: default
  annotations:
    everest.io/disk-volume-type: SSD     # 云硬盘类型，需要与快照源云硬盘保持一致
  labels:
    failure-domain.beta.kubernetes.io/region: cn-north-4
    failure-domain.beta.kubernetes.io/zone: cn-north-4b
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: '10'
  storageClassName: csi-disk
  dataSource:
    name: cce-disksnap-test            # 快照的名称
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

10.7 本地持久存储卷和临时存储卷

说明

持久存储卷和临时存储卷仅在集群版本 >= v1.21.2-r0 时支持创建，且临时存储卷需要 Everest 插件版本>=1.2.29，持久存储卷需要 Everest 插件版本>=1.2.31。

持久存储卷和临时存储卷

CCE 支持将节点上数据盘设置为持久存储卷和临时存储卷。

- 持久存储卷通过 LVM 组成存储池（VolumeGroup），然后划分 LV 给容器挂载使用。使用持久存储卷作为存储介质的 PV 的类型可称之为 Local PV。
- 临时存储卷可以作为 EmptyDir 的存储介质。临时存储卷通过 LVM 组成存储池（VolumeGroup），然后划分 LV 给容器挂载使用，相比原生 EmptyDir 默认的存储介质类型性能更好。

持久存储卷和临时存储卷支持如下两种写入模式。

- 线性：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。
- 条带化：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。多块卷才能选择条带化。

约束与限制

- 移除节点、删除节点、重置节点和缩容节点会导致与节点关联的本地持久存储卷类型的 PVC/PV 数据丢失，无法恢复，且 PVC/PV 无法再正常使用。移除节点、删除节点、重置节点和缩容节点时使用了本地持久存储卷的 Pod 会从待删除、重

置的节点上驱逐，并重新创建 Pod，Pod 会一直处于 pending 状态，因为 Pod 使用的 PVC 带有节点标签，由于冲突无法调度成功。节点重置完成后，Pod 可能调度到重置好的节点上，此时 Pod 会一直处于 creating 状态，因为 PVC 对应的底层逻辑卷已经不存在了。

- 持久存储卷或临时存储卷创建后，请勿在节点上手动删除对应的存储池或卸载数据盘，否则会导致数据丢失等异常情况。
- 如果要使用临时存储卷，请确保节点上 Pod 不要挂载/var/lib/kubelet/pods/目录，否则可能会导致使用了临时存储卷的 Pod 无法正常删除。

添加持久存储卷或临时存储卷

有两种方法可以添加持久存储卷或临时存储卷。

- 在创建节点时，可以为节点添加数据盘作为持久存储卷或临时存储卷。

- 如果创建节点时没有添加持久存储卷或临时存储卷，或当前存储卷容量不够，可以去 ECS 中为节点添加磁盘，然后在 CCE 的存储池中导入，导入时可以选择写入模式。

说明

- 条带化模式的存储池不支持扩容，条带化扩容后可能造成碎片空间，无法使用。
- 存储池不支持缩容和删除。
- 如果删除节点上存储池的磁盘，会导致存储池异常。

使用持久存储卷

本地持久存储卷支持使用 StorageClass 动态创建 PVC，StorageClass 名称为 csi-local-topology。csi-local-topology 的行为相比 csi-disk 等他类型 StorageClass 有较大差异，使用 csi-local-topology 行为如下。

添加了本地持久存储卷的节点会自动加上 node.kubernetes.io/local-storage-persistent 的标签。如果 Pod 使用 csi-local-topology 类型的 PVC，调度器会将 Pod 调度到拥有

node.kubernetes.io/local-storage-persistent 标签的节点上，也就是拥有本地持久存储卷的节点上。

- 单独创建 PVC，PVC 创建后，状态会一直为 Pending，不会立即创建 PV。等有 Pod 使用 PVC，调度器将 Pod 调度到节点后，everest 再创建 localpv 所需的逻辑卷，并返回 PV，PVC 完成与 PV 的绑定。待挂载成功后，Pod 启动。
- 创建应用时选择动态创建 PVC，此时动态创建 PVC 后，调度器将 Pod 调度到节点，everest 再创建逻辑卷，并返回 PV，PVC 完成与 PV 的绑定。待挂载成功后，Pod 启动。
- 删除应用时，可选择不删除使用的 PVC。这样在下一次创建应用时可以使用使用过的 PVC，这样 Pod 会被调度到 PVC 关联的节点上。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-local-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce             # 必须为 ReadWriteOnce
  resources:
    requests:
      storage: 10Gi           # 本地持久存储卷大小
  storageClassName: csi-local-topology    # StorageClass 类型为 csi-local-topology
```

使用临时存储卷

创建工作负载时，EmptyDir 的磁盘介质选择为 LocalVolume，表示使用临时存储卷。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
```

```
        volumeMounts:
          - name: vol-164284390917275733
            mountPath: /tmp
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: vol-164284390917275733
          emptyDir:
            medium: LocalVolume          # emptyDir 磁盘介质选择 LocalVolume，表示使用临时存储卷
            sizeLimit: 1Gi
```

临时存储卷异常处理说明

用户如果手动从 ECS 侧卸盘、手动执行 vgremove 两种误操作致临时卷存储池异常。可以先将节点设置为不可调度，具体方法请参见节点调度设置，然后通过重置节点进行恢复。

10.8 对象存储卷挂载设置自定义访问密钥（AK/SK）

背景信息

Everest 在 1.2.8 及以上版本提供了设置自定义访问密钥的能力，这样可以让 IAM 用户使用自己的访问密钥挂载对象存储卷，从而可以对 OBS 进行访问权限控制。

前提条件

- Everest 要求 1.2.8 及以上版本。
- 集群要求 1.15.11 及以上版本。

约束与限制

自定义访问密钥暂不支持安全容器。

关闭自动挂载访问密钥

老版本控制台会要求您上传 AK/SK，对象存储卷挂载时默认使用您上传的访问密钥，相当于所有 IAM 用户（即子用户）都使用的是同一个访问密钥挂载的对象桶，对桶的权限都是一样的，导致无法对 IAM 用户使用对象存储桶进行权限控制。

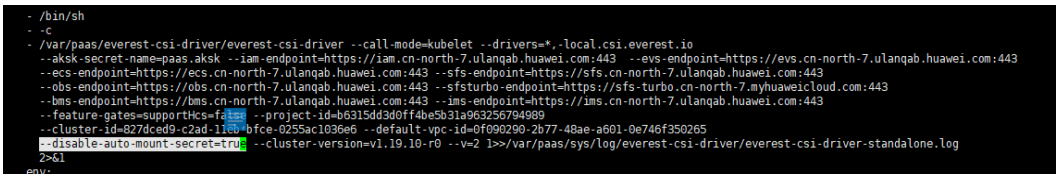
如果您之前上传过 AK/SK，为防止 IAM 用户越权，建议关闭自动挂载访问密钥，即需要在 Everest 插件中将 **disable_auto_mount_secret** 参数打开，这样使用对象存储时就不会自动使用在控制台上传的访问密钥。

说明

- 设置 disable-auto-mount-secret 时要求当前集群中无对象存储卷，否则挂载了该对象卷的工作负载扩容或重启的时候会由于必须指定访问密钥而导致挂卷失败。
- disable-auto-mount-secret 设置为 true 后，则创建 PV 和 PVC 时必须指定挂载访问密钥，否则会导致对象卷挂载失败。

kubectl edit ds everest-csi-driver -nkube-system

搜索 disable-auto-mount-secret，并将值设置为 true。



执行 **:wq** 保存退出，等待实例重启完毕即可。

使用访问密钥创建 Secret

步骤 1 获取访问密钥。

步骤 2 对访问密钥进行 base64 编码（假设上文获取到的 ak 为“xxx”，sk 为“yyy”）。

echo -n xxx|base64

echo -n yyy|base64

记录编码后的 AK 和 SK。

步骤 3 新建一个 secret 的 yaml，如 test-user.yaml。

```
apiVersion: v1
data:
  access.key: WE5WWVhVNU*****
  secret.key: Nnk4emJyZ0*****
kind: Secret
metadata:
  name: test-user
  namespace: default
  labels:
    secret.kubernetes.io/used-by: csi
type: cfe/secure-opaque
```

其中：

参数	描述
access.key	base64 编码后的 ak。
secret.key	base64 编码后的 sk。
name	secret 的名称
namespace	secret 的命名空间
secret.kubernetes.io/used-by: csi	带上这个标签才能在控制台上创建 OBS PV/PVC 时可见。
type	密钥类型，该值必须为 cfe/secure-opaque 使用该类型，用户输入的数据会自动加密。

步骤 4 创建 Secret。

kubectl create -f test-user.yaml

----结束

静态创建对象存储卷时指定挂载 Secret

使用访问密钥创建 Secret 后，在创建 PV 时只需要关联上 Secret，就可以使用 Secret 中的访问密钥（AK/SK）挂载对象存储卷。

步骤 1 登录 OBS 控制台，创建对象存储桶，记录桶名称和存储类型，以并行文件系统为例。

步骤 2 新建一个 pv 的 yaml 文件，如 pv-example.yaml。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  csi:
    nodePublishSecretRef:
      name: test-user
      namespace: default
    driver: obs.csi.everest.io
    fsType: obsfs
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: cn-north-4
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: obs-normal-static-pv
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-obs
```

参数	描述
nodePublishSecretRef	挂载时指定的密钥，其中 • name：指定 secret 的名字 • namespace：指定 secret 的命令空间
fsType	文件类型，支持“obsfs”与“s3fs”，取值为 s3fs 时创建是 obs 对象桶，配套使用 s3fs 挂载；取值为 obsfs 时创建的是 obs 并行文件系统，配套使用 obsfs 挂载，推荐使用。
volumeHandle	对象存储的桶名称。

步骤 3 创建 PV。

kubectl create -f pv-example.yaml

PV 创建完成后，就可以创建 PVC 关联 PV。

步骤 4 新建一个 PVC 的 yaml 文件，如 pvc-example.yaml。

PVC yaml 文件配置示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    csi.storage.k8s.io/node-publish-secret-name: test-user
    csi.storage.k8s.io/node-publish-secret-namespace: default
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: obsfs
  name: obs-secret
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
  volumeName: pv-obs-example
```

参数	描述
csi.storage.k8s.io/node-publish-secret-name	指定 secret 的名字
csi.storage.k8s.io/node-publish-secret-namespace	指定 secret 的命令空间

步骤 5 创建 PVC。

kubectl create -f pvc-example.yaml

PVC 创建后，就可以创建工作负载挂载 PVC 使用存储。

----结束

动态创建对象存储卷时指定挂载密钥

动态创建对象存储卷时，可通过如下方法指定挂载密钥。

步骤 1 新建一个 pvc 的 yaml 文件，如 pvc-example.yaml。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    csi.storage.k8s.io/node-publish-secret-name: test-user
    csi.storage.k8s.io/node-publish-secret-namespace: default
```

```
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: obsfs
  name: obs-secret
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
```

参数	描述
csi.storage.k8s.io/node-publish-secret-name	指定 secret 的名字
csi.storage.k8s.io/node-publish-secret-namespace	指定 secret 的命令空间

步骤 2 创建 PVC。

kubectl create -f pvc-example.yaml

PVC 创建后，就可以创建工作负载挂载 PVC 使用存储。

----结束

配置验证

根据上述步骤，使用 IAM 用户的密钥挂载对象存储卷。假设工作负载名称为 obs-secret，容器内挂载目录是/temp，IAM 用户权限为 CCE ReadOnlyAccess 和 Tenant Guest。

1. 查询工作负载实例名称。

   kubectl get po | grep obs-secret

   期望输出：

   ```
   obs-secret-5cd558f76f-vxslv         1/1     Running   0          3m22s
   ```

2. 查询挂载目录下对象，查询正常。

   kubectl exec obs-secret-5cd558f76f-vxslv -- ls -l /temp/

3. 尝试在挂在目录内写入数据，写入失败。

   kubectl exec obs-secret-5cd558f76f-vxslv -- touch /temp/test

   期望输出：

   ```
   touch: setting times of '/temp/test': No such file or directory
   command terminated with exit code 1
   ```

4. 在 OBS 控制台配置桶策略，给挂载桶的子用户设置读写权限。
5. 再次尝试在挂在目录内写入数据，写入成功。

   kubectl exec obs-secret-5cd558f76f-vxslv -- touch /temp/test

6. 查看容器内挂载目录，验证数据写入成功。

kubectl exec obs-secret-5cd558f76f-vxslv -- ls -l /temp/

期望输出：

```
-rwxrwxrwx 1 root root 0 Jun  7 01:52 test
```

10.9 设置挂载参数

背景信息

容器使用云存储的时候是将云存储挂载到容器上，挂载完成后就可以像使用本地目录一样使用云存储。

本节主要介绍在挂载云存储的时候如何设置挂载参数，包括文件系统存储卷和对象存储卷的挂载参数设置。您可以在 PV 中设置挂载参数，然后通过 PVC 绑定 PV；也可以在 StorageClass 中设置挂载参数，然后使用 StorageClass 创建 PVC，从而动态创建出的 PV 会默认带有 StorageClass 中设置挂载参数。

文件存储挂载参数

CCE 的存储插件 Everest 在挂载文件存储时默认设置了下表所示的参数。除了这些参数外，您还可以设置其他的文件存储挂载参数。

表10-13 文件存储挂载参数

参数	描述
keep-original-ownership	表示是否保留文件挂载点的 ownership，使用该参数时，要求 Everest 插件版本为 1.2.63 或 2.1.2 以上。 • 默认为不添加该参数，此时挂载文件存储时将会默认把挂载点的 ownership 修改为 root:root。 • 如添加该参数，挂载文件存储时将保持文件系统原有的 ownership。
vers=3	文件系统版本，目前只支持 NFSv3。取值：3
nolock	选择是否使用 NLM 协议在服务器上锁文件。当选择 nolock 选项时，锁对于同一主机的应用有效，对不同主机不受锁的影响。
timeo=600	NFS 客户端重传请求前的等待时间(单位为 0.1 秒)。建议值：600。
hard/soft	挂载方式类型。 • 取值为 hard，即使用硬连接方式，若 NFS 请求超时，则客户端一直重新请求直至成功。 • 取值为 soft，即软挂载方式挂载系统，若 NFS 请求超时，则客户端向调用程序返回错误。 默认为 hard。

对象存储挂载参数

CCE 的存储插件 Everest 在挂载文件存储时默认设置了下表参数。除了这些参数外，您还可以设置其他的对象存储挂载参数。

表10-14 默认使用且不可取消的挂载参数

参数	描述
use_ino	使用该选项，由 obsfs 分配 inode 编号。读写模式下自动开启。
big_writes	配置后可更改写缓存最大值大小
nonempty	允许挂载目录非空
allow_other	允许其他用户访问并行文件系统
no_check_certificate	不校验服务端证书
enable_noobj_cache	为不存在的对象启用缓存条目，可提高性能。对象桶读写模式下自动使用。 **从 Everest 1.2.40 版本开始不再默认设置 enable_noobj_cache 参数。**
sigv2	签名版本。对象桶自动使用。

表10-15 默认使用且可取消的挂载参数

参数	描述
max_write=131072	使用该选项，由 obsfs 分配 inode 编号。读写模式下自动开启。
ssl_verify_hostname=0	不根据主机名验证 SSL 证书。
max_background=100	可配置后台最大等待请求数。并行文件系统自动使用。
public_bucket=1	设置为 1 时匿名挂载公共桶。对象桶读写模式下自动使用。

对象存储卷挂载时使用的所有参数，可以登录到运行挂载对象存储卷的 Pod 所在节点上通过进程详情观察：

- 对象桶：ps -ef | grep s3fs

```
root     22142     1  0 Jun03 ?        00:00:00 /usr/bin/s3fs pvc-82fe2cbe-3838-
43a2-8afb-f994e402fb9d /mnt/paas/kubernetes/kubelet/pods/0b13ff68-4c8e-4a1c-
b15c-724fd4d64389/volumes/kubernetes.io~csi/pvc-82fe2cbe-3838-43a2-8afb-
```

```
f994e402fb9d/mount -o url=https://{{endpoint}}:443 -o endpoint=xxxxxx -o
passwd_file=/opt/everest-host-connector/1622707954357702943_obstmpcred/pvc-
82fe2cbe-3838-43a2-8afb-f994e402fb9d -o nonempty -o big_writes -o
enable_noobj_cache -o sigv2 -o allow_other -o no_check_certificate -o
ssl_verify_hostname=0 -o max_write=131072 -o multipart_size=20 -o umask=0
```

- 并行文件系统：ps -ef | grep obsfs

```
root     1355    1  0 Jun03 ?       00:03:16 /usr/bin/obsfs pvc-86720bb9-5aa8-
4cde-9231-5253994f8468 /mnt/paas/kubernetes/kubelet/pods/c959a91d-eced-4b41-
91c6-96cbd65324f9/volumes/kubernetes.io~csi/pvc-86720bb9-5aa8-4cde-9231-
5253994f8468/mount -o url=https://{{endpoint}}:443 -o endpoint=xxxxxx -o
passwd_file=/opt/everest-host-connector/1622714415305160399_obstmpcred/pvc-
86720bb9-5aa8-4cde-9231-5253994f8468 -o allow_other -o nonempty -o big_writes -
o use_ino -o no_check_certificate -o ssl_verify_hostname=0 -o umask=0027 -o
max_write=131072 -o max_background=100 -o uid=10000 -o gid=10000
```

前提条件

- Everest 插件版本要求 **1.2.8 及以上**版本。
- 插件主要负责将挂载参数识别并传递给底层存储，指定参数有否有效依赖于底层存储是否支持。

约束与限制

挂载参数暂不支持安全容器。

在 PV 中设置挂载参数

在 PV 中设置挂载参数可以通过 mountOptions 字段实现，如下所示，mountOptions 支持挂载的字段请参见文件存储挂载参数和对象存储挂载参数。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  mountOptions:
  - umask=0027
  - uid=10000
  - gid=10000
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc-obs-example
    namespace: default
  csi:
    driver: obs.csi.everest.io
    fsType: obsfs
```

```
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: cn-north-4
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: obs-normal-static-pv
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-obs
```

PV 创建后，可以创建 PVC 关联 PV，然后在工作负载的容器中挂载。

在 StorageClass 中设置挂载参数

在 StorageClass 中设置挂载参数同样可以通过 mountOptions 字段实现，如下所示，mountOptions 支持挂载的字段请参见文件存储挂载参数和对象存储挂载参数。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-obs-mount-option
mountOptions:
- umask=0027
- uid=10000
- gid=10000
parameters:
  csi.storage.k8s.io/csi-driver-name: obs.csi.everest.io
  csi.storage.k8s.io/fstype: s3fs
  everest.io/obs-volume-type: STANDARD
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

StorageClass 设置好后，就可以使用这个 StorageClass 创建 PVC，从而动态创建出的 PV 会默认带有 StorageClass 中设置挂载参数。

11 运维管理

11.1 监控管理

CCE 配合 AOM 对集群进行全方位的监控，在创建节点时会默认安装 AOM 的 ICAgent（在集群 kube-system 命名空间下名为 icagent 的 DaemonSet），ICAgent 默认采集集群底层资源以及运行在集群上负载的监控数据；另外，ICAgent 还能采集负载的自定义指标监控数据。

- 资源监控指标

  资源基础监控包含 CPU/内存/磁盘等，具体请参见资源监控指标。您可以在 CCE 控制台从集群、节点、工作负载等维度查看这些监控指标数据，也可以在 AOM 中查看。

- 自定义指标

  ICAgent 采集应用程序中的自定义指标并上传到 AOM，具体使用方法请参见自定义监控。

资源监控指标

在 CCE 控制台，可以查看如下指标。

表11-1 资源监控指标

监控指标	指标含义
CPU 分配率	分配给工作负载使用的 CPU 占比。
内存分配率	分配给工作负载使用的内存占比。
CPU 使用率	CPU 使用率。
内存使用率	内存使用率。
磁盘使用率	磁盘使用率。
下行速率	一般指从网络下载数据到节点的速度，单位 KB/s。
上行速率	一般指从节点上传网络的速度，单位 KB/s。

监控指标	指标含义
磁盘读取速率	每秒从磁盘读出的数据量，单位 KB/s。
磁盘写入速率	每秒写入磁盘的数据量，单位 KB/s。

在 AOM 控制台，可以查看主机指标和容器实例的指标。

查看集群监控数据

单击集群名称进入集群，在左侧导航栏单击集群信息，在右侧可看到集群所有节点（不含控制节点）近一小时的 CPU 指标和内存指标。

集群监控视图会展示集群资源的监控状态、集群所有节点的 CPU/内存/磁盘的使用率，以及 CPU 和内存的分配率。

监控名词解释：

- CPU 分配率 = 集群下运行的 Pod CPU 配额申请值（Request）之和 / 集群下所有节点（不含控制节点）的 CPU 可分配量之和
- 内存分配率 = 集群下运行的 Pod 内存配额申请值（Request）之和 / 集群下所有节点（不含控制节点）的内存可分配量之和
- CPU 使用率 = 集群下所有节点（不含控制节点）上实际使用的 CPU 使用率的平均值。
- 内存使用率 = 集群下所有节点（不含控制节点）上实际使用的内存使用率的平均值。

说明

节点资源（CPU 或内存）可分配量=总量-预留值-驱逐阈值。详情请参见节点预留资源计算公式。

CCE 提供了控制节点的状态、所在可用区、CPU 使用率和内存使用率。

查看节点监控数据

除了在集群监控界面查看所有节点监控数据外，您还可以查看单个节点的监控数据。单击集群名称进入集群，在左侧导航栏选择“节点管理”，在右侧节点所在行单击“监控”。

监控数据来源与 AOM，可查看节点的监控数据包括 CPU、内存、磁盘、网络、GPU 等。

查看工作负载的监控数据

工作负载的监控数据可以在工作负载详情的监控页面下查看。单击集群名称进入集群，在左侧导航栏选择“工作负载”，在右侧工作负载所在行单击“监控”。

监控数据来源与 AOM，可查看工作负载的监控数据包括 CPU、内存、网络、GPU 等。

监控名词解释：

- 工作负载 CPU 使用率 = 工作负载各个 Pod 中 CPU 使用率的最大值
- 工作负载内存使用率 = 工作负载各个 Pod 中内存使用率的最大值

您还可以单击“查看更多”直接跳转到 AOM 控制台查看工作负载的监控数据。

查看容器实例 Pod 的监控数据

在工作负载详情页面的实例列表页签中可以查看 Pod 的监控数据。

监控名词解释：

- Pod CPU 使用率 = Pod 实际使用的 CPU 核数 / 业务容器 CPU 核数限制值之和（未配置限制值时采用节点总量）

- Pod 内存使用率 = Pod 实际使用的物理内存 / 业务容器物理内存限制值之和（未配置限制值时采用节点总量）

11.2 日志管理

CCE 支持配置工作负载日志策略，便于日志的统一收集、管理和分析，以及按周期防爆处理。

- ICAgent 日志采集：

  默认情况下，ICAgent 会采集容器的标准输出，您无需做任何设置。

  您还可以在创建工作负载的时候设置容器日志存储路径，ICAgent 会采集该路径下日志。

  容器日志可以选择主机路径和容器路径两种模式。

  - 主机路径：HostPath 模式，将主机路径挂载到指定的容器路径（挂载路径）。用户可以在节点的主机路径中查看到容器输出在挂载路径中的日志信息。
  - 容器路径：EmptyDir 模式，将节点的临时路径挂载到指定的路径（挂载路径）。临时路径中存在的但暂未被采集器上报到 AOM 的日志数据在 Pod 实例删除后会消失。

11.3 使用 ICAgent 采集容器日志

CCE 配合 AOM 收集工作负载的日志，在创建节点时会默认安装 AOM 的 ICAgent（在集群 kube-system 命名空间下名为 icagent 的 DaemonSet），ICAgent 负责收集工作负载的日志并上报到 AOM，您可以在 CCE 控制台和 AOM 控制台查看工作负载的日志。

约束与限制

ICAgent 只采集*.log、*.trace 和*.out 类型的文本日志文件。

费用说明

AOM 每月赠送每个账号 500M 免费日志采集额度。

使用 ICAgent 采集日志

在工作负载中可以单独配置日志采集策略，此策略需要使用 ICAgent。

步骤 1 在 CCE 中创建工作负载时，在配置容器信息时可以设置容器日志。

步骤 2 单击+添加日志策略。

以 nginx 为例，不同工作负载根据实际情况配置。

图11-1 添加日志策略

步骤 3 存储类型有“主机路径”和“容器路径”两种类型可供选择：

表11-2 配置日志策略

参数	参数说明
存储类型	• 主机路径：HostPath 模式，将主机路径挂载到指定的容器路径（挂载路径）。用户可以在节点的主机路径中查看到容器输出在挂载路径中的日志信息。 • 容器路径：EmptyDir 模式，将节点的临时路径挂载到指定的路径（挂载路径）。临时路径中存在的但暂未被采集器上报到 AOM 的日志数据在 Pod 实例删除后会消失。
主机路径	请输入主机的路径，如：/var/paas/sys/log/nginx
挂载路径	请输入数据存储要挂载到容器上的路径，如：/tmp 须知 • 请不要挂载到系统目录下，如“/”、“**/var/run**”等，否则会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 • 挂载高危目录的情况下，建议使用低权限帐号启动，否则可能会造成宿主机高危文件被破坏。 • AOM 只采集最近修改过的前 20 个日志文件，且默认采集两级子目录。 • AOM 只采集挂载路径下的“.log”、“.trace”、“.out”文本日志文件。 • 容器中挂载点的权限设置方法，请参见为 Pod 或容器配置安全性上下文。
主机扩展路径	仅“主机路径”类型需要填写 通过实例的 ID 或者容器的名称扩展主机路径，实现同一个主机路径下区分来自不同容器的挂载。 会在原先的“卷目录/子目录”中增加一个三级目录。使用户更方便获取单个 Pod 输出的文件。

参数	参数说明
	• None：不配置拓展路径。 • PodUID：Pod 的 ID。 • PodName：Pod 的名称。 • PodUID/ContainerName：Pod 的 ID/容器名称。 • PodName/ContainerName：Pod 名称/容器名称。
采集路径	设置采集路径可以更精确的指定采集内容，当前支持以下设置方式： • 不设置则默认采集当前路径下.log .trace .out 文件 • 设置**表示递归采集 5 层目录下的.log .trace .out 文件 • 设置*表示模糊匹配 例子： 采集路径为/tmp/**/test*.log 表示采集/tmp 目录及其 1-5 层子目录下的全部以 test 开头的.log 文件。 **注意** 使用采集路径功能请确认您的采集器 ICAgent 版本为 5.12.22 或以上版本。
日志转储	此处日志转储是指日志的本地绕接。 • 设置：AOM 每分钟扫描一次日志文件，当某个日志文件超过 50MB 时，会立即对其转储（转储时会在该日志文件所在的目录下生成一个新的 zip 文件。对于一个日志文件，AOM 只保留最近生成的 20 个 zip 文件，当 zip 文件超过 20 个时，时间较早的 zip 文件会被删除），转储完成后 AOM 会将该日志文件清空。 • 不设置：若您在下拉列表框中选择“不设置”，则 AOM 不会对日志文件进行转储。 **说明** • AOM 的日志绕接能力是使用 copytruncate 方式实现的，如果选择了设置，请务必保证您写日志文件的方式是 append（追加模式），否则可能出现文件空洞问题。 • 当前主流的日志组件例如 Log4j、Logback 等均已经具备日志文件的绕接能力，如果您的日志文件已经实现了绕接能力，则无需设置。否则可能出现冲突。 • 建议您的业务自己实现绕接，可以更灵活的控制绕接文件的大小和个数。

步骤 4 单击“确定”，并完成创建工作负载。

----结束

YAML 示例（ICAgent）

您可以通过在 YAML 定义的方式设置容器日志存储路径。

如下所示，使用 EmptyDir 挂载到容器的“/var/log/nginx”路径下，这样 ICAgent 就会采集容器“/var/log/nginx”路径下的日志。其中 policy 字段是 CCE 自定义的字段，能够让 ICAgent 识别并采集日志。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testlog
  namespace: default
spec:
  selector:
    matchLabels:
      app: testlog
  template:
    replicas: 1
    metadata:
      labels:
        app: testlog
    spec:
      containers:
        - image: 'nginx:alpine'
          name: container-0
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:
            - name: vol-log
              mountPath: /var/log/nginx
              policy:
                logs:
                  rotate: ''
      volumes:
        - emptyDir: {}
          name: vol-log
      imagePullSecrets:
        - name: default-secret
```

使用 HostPath 方法如下所示，相比 EmptyDir 就是 volume 的类型变成 hostPath，且需要配置 hostPath 在主机上的路径。下面示例中将主机上“/tmp/log”挂载到容器的“/var/log/nginx”路径下，这样 ICAgent 就会采集容器“/var/log/nginx”路径下的日志，且日志还会在主机的“/tmp/log”路径下存储。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testlog
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testlog
```

```
template:
  metadata:
    labels:
      app: testlog
  spec:
    containers:
      - image: 'nginx:alpine'
        name: container-0
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
          limits:
            cpu: 250m
            memory: 512Mi
        volumeMounts:
          - name: vol-log
            mountPath: /var/log/nginx
            readOnly: false
            extendPathMode: PodUID
            policy:
              logs:
                rotate: Hourly
                annotations:
                  pathPattern: '**'
                  format: ''
    volumes:
      - hostPath:
          path: /tmp/log
        name: vol-log
    imagePullSecrets:
      - name: default-secret
```

表11-3 关键参数解释

参数	中文解释	说明
extendPathMode	主机扩展路径	通过实例的 ID 或者容器的名称扩展主机路径，实现同一个主机路径下区分来自不同容器的挂载。 会在原先的“卷目录/子目录”中增加一个三级目录。使用户更方便获取单个 Pod 输出的文件。 • None：不配置拓展路径。 • PodUID：Pod 的 ID。 • PodName：Pod 的名称。 • PodUID/ContainerName：Pod 的 ID/容器名称。 • PodName/ContainerName：Pod 名称/容器名称。
policy.logs.rotate	日志转储	此处日志转储是指日志的本地绕接。 • 设置：AOM 每分钟扫描一次日志文件，当某个日志文件超过 50MB 时，会立即对其转储（转储时会在该日志文件所在的目录下生成一个新的 zip

参数	中文解释	说明
		文件。对于一个日志文件，AOM 只保留最近生成的 20 个 zip 文件，当 zip 文件超过 20 个时，时间较早的 zip 文件会被删除），转储完成后 AOM 会将该日志文件清空。 • 不设置：若您在下拉列表框中选择“不设置”，则 AOM 不会对日志文件进行转储。 **说明** • AOM 的日志绕接能力是使用 copytruncate 方式实现的，如果选择了设置，请务必保证您写日志文件的方式是 append（追加模式），否则可能出现文件空洞问题。 • 当前主流的日志组件例如 Log4j、Logback 等均已经具备日志文件的绕接能力，如果您的日志文件已经实现了绕接能力，则无需设置。否则可能出现冲突。 • 建议您的业务自己实现绕接，可以更灵活的控制绕接文件的大小和个数。
policy.logs.annotations.pathPattern	采集路径	设置采集路径可以更精确的指定采集内容，当前支持以下设置方式： • 不设置则默认采集当前路径下.log .trace .out 文件 • 设置**表示递归采集 5 层目录下的.log .trace .out 文件 • 设置*表示模糊匹配 例子： 采集路径为/tmp/**/test*.log 表示采集/tmp 目录及其 1-5 层子目录下的全部以 test 开头的.log 文件。 **注意** 使用采集路径功能请确认您的采集器 ICAgent 版本为 5.12.22 或以上版本。
policy.logs.annotations.format	多行日志匹配	有些程序打印的日志存在一条完整的日志数据跨占多行（例如 Java 程序日志）情况，日志采集系统默认是按行采集。如果您想在日志采集系统中按整条显示日志，可以开启多行日志，采用时间或正则匹配的方式，当某行日志匹配上预先设置的时间格式或正则表达式，就认为是一条日志的开头，而下一个行首出现作为该条日志的结束标识符。 格式如下 `{ "multi": { "mode": "time", "value": "YYYY-MM-DD hh:mm:ss" } }` **multi 表示分行模式：** • time：日志时间，请输入时间通配符。时间通配符填写参考示例：如日志中的时间是 2017-01-01

参数	中文解释	说明
		23:59:59，按照规则应该填写：YYYY-MM-DD hh:mm:ss。 • regular：正则模式，请输入正则表达式。

查看日志

日志采集路径配置和工作负载创建完成后，若已配置的路径下存在日志文件，则ICAgent 会从已配置的路径中采集日志文件，采集大概需要 1 分钟，请您耐心等待。

待采集完成后，进入工作负载详情页，单击右上角的“日志”按钮查看日志详情。

您还可以在 AOM 控制台查看日志。

另外您还可以使用 kubectl logs 命令查看容器的标准输出，具体如下所示。

```
# 查看指定 pod 的日志
kubectl logs <pod_name>
kubectl logs -f <pod_name> #类似 tail -f 的方式查看

# 查看指定 pod 中指定容器的日志
kubectl logs <pod_name> -c <container_name>

kubectl logs pod_name -c container_name -n namespace (一次性查看)
kubectl logs -f <pod_name> -n namespace (tail -f 方式实时查看)
```

12 命名空间

12.1 创建命名空间

操作场景

命名空间（Namespace）是对一组资源和对象的抽象整合。在同一个集群内可创建不同的命名空间，不同命名空间中的数据彼此隔离。使得它们既可以共享同一个集群的服务，也能够互不干扰。

例如可以将开发环境、测试环境的业务分别放在不同的命名空间。

前提条件

至少已创建一个集群。

约束与限制

每个命名空间下，创建的服务数量不能超过 6000 个。此处的服务对应 kubernetes 的 service 资源，即工作负载所添加的服务。

命名空间类别

命名空间按创建类型分为两大类：集群默认创建的、用户创建的。

- 集群默认创建的：集群在启动时会默认创建 default、kube-public、kube-system、kube-node-lease 命名空间。
  - default：所有未指定 Namespace 的对象都会被分配在 default 命名空间。
  - kube-public：此命名空间下的资源可以被所有人访问（包括未认证用户），用来部署公共插件、容器模板等。
  - kube-system：所有由 Kubernetes 系统创建的资源都处于这个命名空间。
  - kube-node-lease：每个节点在该命名空间中都有一个关联的“Lease”对象，该对象由节点定期更新。NodeStatus 和 NodeLease 都被视为来自节点的心跳，在 v1.13 之前的版本中，节点的心跳只有 NodeStatus，NodeLease 特性从 v1.13 开始引入。NodeLease 比 NodeStatus 更轻量级，该特性在集群规模扩展性和性能上有明显提升。

- 用户创建的：用户可以按照需要创建命名空间，例如开发环境、联调环境和测试环境分别创建对应的命名空间。或者按照不同的业务创建对应的命名空间，例如系统若分为登录和游戏服务，可以分别创建对应命名空间。

创建命名空间

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“命名空间”，在右上角单击“创建命名空间”。

步骤 3 参照下表设置命名空间参数。

表12-1 命名空间基本信息

参数	参数说明
名称	新建命名空间的名称，命名必须唯一。
描述	输入对命名空间的描述信息。
配额管理	资源配额可以限制命名空间下的资源使用，进而支持以命名空间为粒度的资源划分。 须知 **建议根据需要在命名空间中设置资源配额，避免因资源过载导致集群或节点异常。** 例如：在集群中每个节点可以创建的实例（Pod）数默认为 110 个，如果您创建的是 50 节点规格的集群，则最多可以创建 5500 个实例。因此，您可以在命名空间中自行设置资源配额以确保所有命名空间内的实例总数不超过 5500 个，以避免资源过载。 请输入整型数值，不输入表示不限制该资源的使用。 若您需要限制 CPU 或内存的配额，则创建工作负载时必须指定 CPU 或内存请求值。

步骤 4 配置完成后，单击“确定”。

----结束

使用 kubectl 创建 Namespace

使用如下方式定义 Namespace。

```
apiVersion: v1
kind: Namespace
metadata:
  name: custom-namespace
```

使用 kubectl 命令创建。

```
$ kubectl create -f custom-namespace.yaml
namespace/custom-namespace created
```

您还可以使用 kubectl create namespace 命令创建。

```
$ kubectl create namespace custom-namespace
namespace/custom-namespace created
```

12.2 管理命名空间

使用命名空间

- 创建工作负载时，您可以选择对应的命名空间，实现资源或租户的隔离。
- 查询工作负载时，选择对应的命名空间，查看对应命名空间下的所有工作负载。

命名空间使用实践

- **按照不同环境划分命名空间**

  一般情况下，工作负载发布会经历开发环境、联调环境、测试环境，最后到生产环境的过程。这个过程中不同环境部署的工作负载相同，只是在逻辑上进行了定义。分为两种做法：

  - 分别创建不同集群。

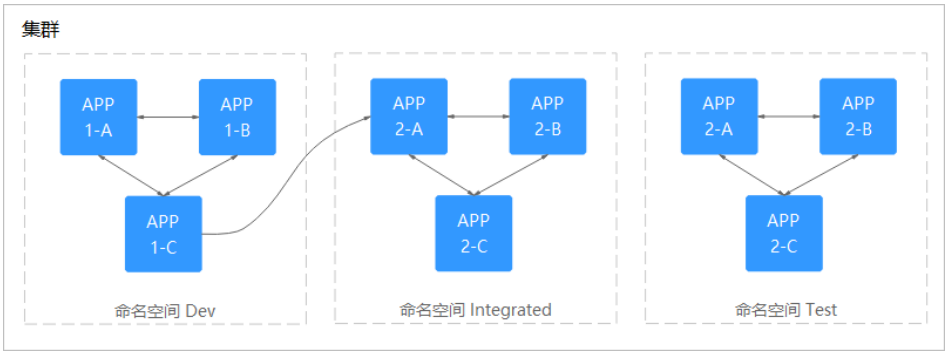
    不同集群之间，资源不能共享。同时，不同环境中的服务互访需要通过负载均衡才能实现。

  - 不同环境创建对应命名空间。

    同个命名空间下，通过服务名称（Service name）可直接访问。跨命名空间的可以通过服务名称、命名空间名称访问。

    例如下图，开发环境/联调环境/测试环境分别创建了命名空间。

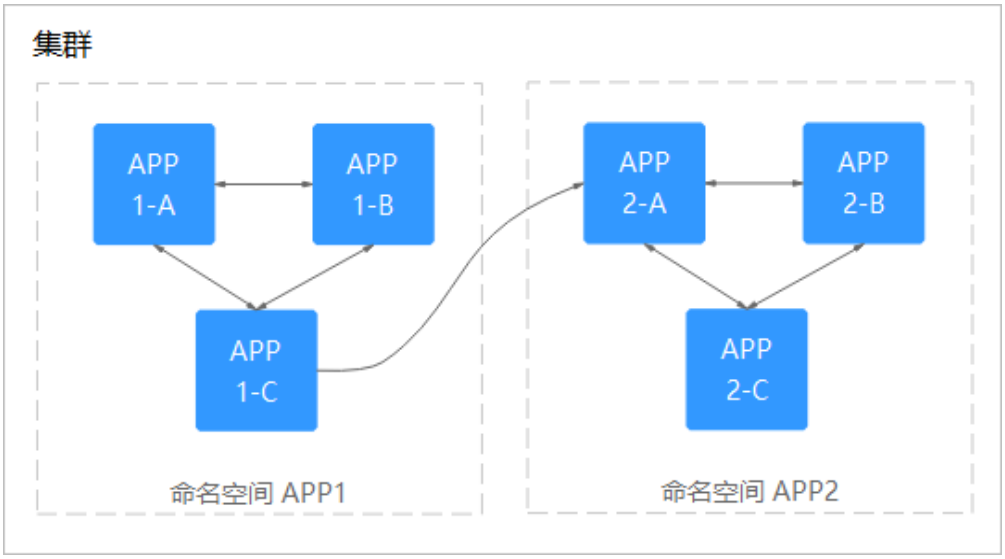
    图12-1 不同环境创建对应命名空间

- **按照应用划分命名空间**

  对于同个环境中，应用数量较多的情况，建议进一步按照工作负载类型划分命名空间。例如下图中，按照 APP1 和 APP2 划分不同命名空间，将不同工作负载在逻辑上当做一个工作负载组进行管理。且同一个命名空间内的工作负载可以通过服务名称访问，不同命名空间下的通过服务名称、命名空间名称访问。

图12-2 按照工作负载划分命名空间



删除命名空间

删除命名空间会删除该命名空间下所有的资源（如工作负载，短任务、配置项等），请谨慎操作。

步骤 1 登录 CCE 控制台，进入集群。

步骤 2 在左侧导航栏中选择“命名空间”，选中待删除的命名空间，单击“更多 > 删除”。

根据系统提示进行删除操作。系统内置的命名空间不支持删除。

----结束

12.3 设置命名空间级的网络策略

您可以通过网络隔离开关，设置命名空间层面的网络策略。

例如命名空间 default，网络隔离的默认状态为“隔离状态未开启”，表示“当前集群下的所有工作负载”都可以访问“命名空间 default 下的工作负载”。

若您需要设置其它工作负载不可以访问“命名空间 default 下的工作负载”，请参照以下步骤设置：

须知

- 当前仅**容器隧道网络模型**的集群支持网络策略（NetworkPolicy）。
- 网络策略（NetworkPolicy）暂不支持设置出方向（egress）。
- 不支持对 IPv6 地址网络隔离。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见购买 CCE 集群。
- 您已成功创建一个命名空间，参见创建命名空间。

操作步骤

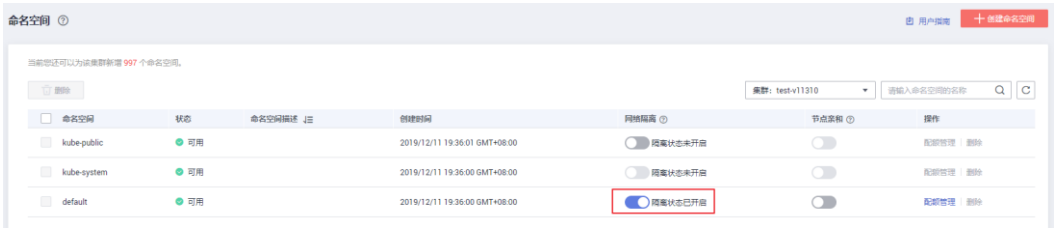
步骤 1 登录为 CCE 控制台，在左侧导航栏中选择“资源管理 > 命名空间”。

步骤 2 在“集群”下拉框中，选择命名空间所在的集群。

步骤 3 在待设置命名空间（例如 default）后，将网络隔离一栏的“隔离状态未开启”改为“隔离状态已开启”，开启后，当前集群内的其他工作负载都不能访问此命名空间下的工作负载。

设置完成后，当前集群内其它命名空间下的工作负载都不能访问 default 下的工作负载。

图12-3 命名空间网络策略



----结束

网络隔离说明

打开网络隔离其实是在这个命名空间下创建一个 NetworkPolicy，这个 NetworkPolicy 选择命名空间下所有的 Pod，然后不让其他命名空间的 Pod 访问。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
   name: deny-default
   namespace: default
spec:
   ingress:
      - from:
```

```
        - podSelector: {}
  podSelector: {}                    # {}表示选择所有 Pod
```

NetworkPolicy 还可以自定义，具体请参见网络策略（NetworkPolicy）。

12.4 设置资源配额及限制

通过设置命名空间级别的资源配额，实现多团队或多用户在共享集群资源的情况下限制团队、用户可以使用的资源总量，包括限制命名空间下创建某一类型对象的数量以及对象消耗计算资源（CPU、内存）的总量。

背景信息

默认情况下，运行中的 Pod 可以无限制的使用 Node 节点上的 CPU 和内存，这意味着任意一个 Pod 都可以无节制地使用集群的计算资源，某个命名空间的 Pod 可能会耗尽集群的所有资源。

kubernetes 在一个物理集群上提供了多个虚拟集群，这些虚拟集群被称为命名空间。命名空间可用于多种工作用途，满足多用户的使用需求，通过为每个命名空间配置资源额度可以有效限制资源滥用，从而保证集群的可靠性。

您可为命名空间配置包括 CPU、内存、Pod 数量等资源的额度，更多信息请参见 Resource Quotas。

其中，不同的集群规模对应的 Pod 数量推荐值如下：

集群规模	Pod 数量推荐值
50 节点	2500 Pod 实例
200 节点	1W Pod 实例
1000 节点	3W Pod 实例
2000 节点	5W Pod 实例

从 1.21 版本集群开始，如果在集群配置管理中开启了 enable-resource-quota 参数，则创建命名空间将会同时创建默认的资源配额 Resource Quotas，根据集群规格不同，各个资源的配额如下表所示。您可以根据实际需求修改。

表12-2 默认资源配额

集群规模	Pod	Deployment	Secret	ConfigMap	Service
50 节点	2000	1000	1000	1000	1000
200 节点	2000	1000	1000	1000	1000
1000 节点	5000	2000	2000	2000	2000

集群规模	Pod	Deployment	Secret	ConfigMap	Service
2000 节点	5000	2000	2000	2000	2000

约束与限制

在 Kubernetes 中，外部用户及内部组件频繁的数据更新操作采用乐观并行的控制方法。通过定义资源版本（resourceVersion）实现乐观锁，资源版本字段包含在对象的元数据（metadata）中。这个字段标识了对象的内部版本号，且对象被修改时，该字段将随之修改。kube-apiserver 可以通过该字段判断对象是否已经被修改。当包含 resourceVersion 的更新请求到达 apiserver，服务器端将对比请求数据与服务器中数据的资源版本号，如果不一致，则表明在本次更新提交时，服务端对象已被修改，此时 apiserver 将返回冲突错误（409）。客户端需重新获取服务端数据，重新修改后再次提交到服务器端；而资源配额对每个命名空间的资源消耗总量提供限制，并且会记录集群中的资源信息，因此开启资源配额后，在大规模并发场景下创建资源冲突概率会变高，会影响批创资源性能。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“命名空间”。

步骤 3 单击对应命名空间后的“管理配额”。

系统级别的命名空间 kube-system、kube-public 默认不支持设置资源配额。

步骤 4 设置资源配额，然后单击“确定”。

须知

- 命名空间设置了 CPU 或内存资源配额后，创建工作负载时，必须指定 CPU 或内存的请求值（request）和约束值（limit），否则 CCE 将拒绝创建实例。若设置资源配额值为 0，则不限制该资源的使用。
- 配额累计使用量包含 CCE 系统默认创建的资源，如 default 命名空间下系统默认创建的 kubernetes 服务（该服务可通过后端 kubectl 工具查看）等，故建议命名空间下的资源配额略大于实际期望值以去除系统默认创建资源的影响。

----结束

13 配置中心

13.1 创建配置项

操作场景

配置项（ConfigMap）是一种用于存储工作负载所需配置信息的资源类型，内容由用户决定。配置项创建完成后，可在容器工作负载中作为文件或者环境变量使用。

配置项允许您将配置文件从容器镜像中解耦，从而增强容器工作负载的可移植性。

配置项价值如下：

- 使用配置项功能可以帮您管理不同环境、不同业务的配置。
- 方便您部署相同工作负载的不同环境，配置文件支持多版本，方便您进行更新和回滚工作负载。
- 方便您快速将您的配置以文件的形式导入到容器中。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“配置项与密钥”，在右上角单击“创建配置项”。

步骤 3 填写参数。

表13-1 新建配置参数说明

参数	参数说明
名称	新建的配置项名称，同一个命名空间里命名必须唯一。
命名空间	新建配置项所在的命名空间。若不选择，默认为 default。
描述	配置项的描述信息。
配置数据	配置项的数据。 键值对形式，单击 ┼ 添加。其中值支持 String、JSON 和

参数	参数说明
	YAML 格式。
标签	配置项的标签。键值对形式，输入键值对后单击“添加”。

步骤 4 配置完成后，单击“确定”。

工作负载配置列表中会出现新创建的工作负载配置。

----结束

ConfigMap 配置项要求

ConfigMap 资源文件支持 YAML 文件格式，且文件大小不得超过 2MB。

文件名称为 configmap.yaml，配置示例如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap
data:
  data-1: value-1
  data-2: value-2
```

使用 kubectl 创建配置项

步骤 1 请参见通过 kubectl 连接集群配置 kubectl 命令。

步骤 2 创建并编辑 cce-configmap.yaml 文件。

vi cce-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cce-configmap
data:
  SPECIAL_LEVEL: Hello
  SPECIAL_TYPE: CCE
```

步骤 3 创建配置项。

kubectl create -f cce-configmap.yaml

kubectl get cm

```
NAME             DATA          AGE
cce-configmap      3             3h
cce-configmap1     3             7m
```

----结束

相关操作

配置项创建完成后，您还可以执行下表中的操作。

表13-2 其他操作

操作	说明
编辑 YAML	单击配置项名称后的“编辑 YAML”，可编辑当前配置项的 YAML 文件。
更新配置	1. 选择需要更新的配置项名称，单击“更新”。 2. 更改配置信息。 3. 单击“确定”。
删除配置	选择要删除的配置项，单击“删除”。 根据系统提示删除配置。

13.2 使用配置项

配置项创建后，可在工作负载环境变量、命令行参数和数据卷三个场景使用。

- 通过配置项设置工作负载环境变量
- 通过配置项设置命令行参数
- 使用配置项挂载到工作负载数据卷

本节以下面这个 ConfigMap 为例，具体介绍 ConfigMap 的用法。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cce-configmap
data:
  SPECIAL_LEVEL: Hello
  SPECIAL_TYPE: CCE
```

须知

在 Pod 里使用 ConfigMap 时，需要 Pod 和 ConfigMap 处于同一集群和命名空间中。

通过配置项设置工作负载环境变量

您可以在创建工作负载时将配置项设置为环境变量，使用 valueFrom 参数引用 ConfigMap 中的 Key/Value。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: configmap-pod-1
spec:
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:                          ##使用 valueFrom 来指定 env 引用配置项的 value 值
            configMapKeyRef:
              name: cce-configmap             ##引用的配置文件名称
              key: SPECIAL_LEVEL              ##引用的配置项 key
  restartPolicy: Never
```

如果您需要将多个配置项的 value 值定义为 pod 的环境变量值，您只需要在 pod 中添加多个环境变量参数即可。

```
env:
- name: SPECIAL_LEVEL_KEY
  valueFrom:
    configMapKeyRef:
         name: cce-configmap
         key: SPECIAL_LEVEL
- name: SPECIAL_TYPE_KEY
  valueFrom:
    configMapKeyRef:
         name: cce-configmap
         key: SPECIAL_TYPE
```

如果要将一个配置项中所有数据都添加到环境变量中，可以使用 envFrom 参数，配置项中的 key 会成为 Pod 中的环境变量名称。

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod-2
spec:
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "env" ]
      envFrom:
      - configMapRef:
          name: cce-configmap
  restartPolicy: Never
```

通过配置项设置命令行参数

您可以使用配置项设置容器中的命令或者参数值，使用环境变量替换语法$(VAR_NAME)来进行。如下面的编排示例所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod-3
```

```
spec:
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: cce-configmap
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: cce-configmap
              key: SPECIAL_TYPE
  restartPolicy: Never
```

这个 Pod 运行后，输出如下内容。

```
Hello CCE
```

使用配置项挂载到工作负载数据卷

配置项也可以在数据卷里面使用，只需在创建工作负载时将配置项挂载到工作负载中即可。挂载完成后，最终生成以 key 为文件名， value 为文件内容的配置文件。

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod-4
spec:
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]   ##列出该目录下的文件名称
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config                          ##挂载到/etc/config 目录下
  volumes:
    - name: config-volume
      configMap:
        name: cce-configmap
  restartPolicy: Never
```

这个 Pod 运行后，在/etc/config 目录下会生成 SPECIAL_LEVEL 和 SPECIAL_TYPE 两个文件，文件的内容为 Hello 和 CCE。且运行时会输出如下内容，即 cce-configmap 中 SPECIAL_LEVEL 和 SPECIAL_TYPE 的名称。

```
SPECIAL_TYPE
SPECIAL_LEVEL
```

挂载 ConfigMap 到数据卷还可以在界面上进行操作，在创建工作负载时，设置容器的高级设置，选择数据存储，添加本地磁盘，选择 ConfigMap 即可配置。具体操作请参见配置项(ConfigMap)挂载。

13.3 创建密钥

操作场景

密钥（Secret）是一种用于存储工作负载所需要认证信息、密钥的敏感信息等的资源类型，内容由用户决定。资源创建完成后，可在容器工作负载中作为文件或者环境变量使用。

操作步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中选择“配置项与密钥”，选择“密钥”页签，在右上角单击“创建密钥”。

步骤 3 填写参数。

表13-3 基本信息说明

参数	参数说明
名称	新建的密钥的名称，同一个命名空间内命名必须唯一。
命名空间	新建密钥所在的命名空间，默认为 default。
描述	密钥的描述信息。
密钥类型	新建的密钥类型。 • Opaque：一般密钥类型。 • kubernetes.io/dockerconfigjson：存放拉取私有仓库镜像所需的认证信息。 • IngressTLS：存放 7 层负载均衡服务所需的证书。 • 其他：若需要创建其他类型的密钥，请手动输入密钥类型。
密钥数据	工作负载密钥的数据可以在容器中使用。 • 当密钥为 Opaque 类型时，单击 ┼，在弹出的窗口中输入键值对，并且可以勾选“自动 Base64 转码”。 • 当密钥为 kubernetes.io/dockerconfigjson 类型时，输入私有镜像仓库的帐号和密码。 • 当密钥为 IngressTLS 类型时，上传证书文件和私钥文件。 **说明** • 证书是自签名或 CA 签名过的凭据，用来进行身份认证。 • 证书请求是对签名的请求，需要使用私钥进行签名。
密钥标签	密钥的标签。键值对形式，输入键值对后单击“添加”。

步骤 4 配置完成后，单击“确定”。

密钥列表中会出现新创建的密钥。

----结束

Secret 资源文件配置说明

本章节主要介绍 Secret 类型的资源描述文件的配置示例。

例如现在有一个工作负载需要获取帐号密码，可以通过 Secret 来实现：

- yaml 文件格式

  定义的 Secret 文件 secret.yaml 内容如下。其中 Value 需要用 Base64，Base64 编码方法请参见如何进行 Base64 编码。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret          # secret 的名称
  namespace: default      #命名空间，默认为 default
data:
  username: ******  #需要用 Base64 编码
  password: ******  #需要用 Base64 编码
type: Opaque     # type 建议不要做修改
```

如何进行 Base64 编码

对字符串进行 Base64 编码，可以直接使用“echo -n 要编码的内容 | base64”命令即可，示例如下：

```
root@ubuntu:~# echo -n "待编码内容" | base64
******
```

13.4 使用密钥

密钥创建后，可在工作负载环境变量和数据卷两个场景使用。

须知

如下密钥为 CCE 系统使用的，请勿对其做任何操作。

- 不要操作 kube-system 下的 secrets。
- 不要操作任何命名空间下的 default-secret、paas.elb。其中，default-secret 用于 SWR 的私有镜像拉取，paas.elb 用于该命名空间下的服务对接 ELB。

- 使用密钥配置 Pod 的数据卷
- 使用密钥设置 Pod 的环境变量

本节以下面这个所 Secret 为例，具体介绍 Secret 的用法。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: ****** #需要用 Base64 编码
  password: ******  #需要用 Base64 编码
```

须知

在 Pod 里使用密钥时，需要 Pod 和密钥处于同一集群和命名空间中。

使用密钥配置 Pod 的数据卷

密钥可以在 Pod 中作为文件使用。如下面的 Pod 示例所示，mysecret 密钥的 username 和 password 以文件方式保存在/etc/foo 目录下。

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

另外，还可以指定密钥的目录路径和权限，username 存放在容器中的/etc/foo/my-group/my-username 目录下。

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

```
    items:
    - key: username
      path: my-group/my-username
      mode: 511
```

挂载 Secret 到数据卷还可以在界面上进行操作，在创建工作负载时，设置容器的高级设置，选择数据存储，添加本地磁盘，选择 Secret 即可配置。具体请参见密钥(Secret)挂载。

使用密钥设置 Pod 的环境变量

密钥可以在 Pod 中设置为环境变量。如下面的 Pod 示例所示，mysecret 密钥的 username 和 password 配置为 Pod 的环境变量。

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: username
      - name: SECRET PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: password
  restartPolicy: Never
```

13.5 集群系统密钥说明

CCE 默认会在每个命名空间下创建如下密钥。

- default-secret
- paas.elb
- default-token-xxxxx（xxxxx 为随机数）

下面将详细介绍这个几个密钥的用途。

default-secret

default-secret 的类型为 kubernetes.io/dockerconfigjson，其 data 内容是登录 SWR 镜像仓库的凭据，用于从 SWR 拉取镜像。在 CCE 中创建工作负载时如果需要从 SWR 拉取镜像，需要配置 imagePullSecrets 的取值为 default-secret，如下所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
    resources:
      limits:
        cpu: 100m
        memory: 200Mi
      requests:
        cpu: 100m
        memory: 200Mi
  imagePullSecrets:
  - name: default-secret
```

default-secret 的 data 数据会定期更新，且当前的 data 内容会在一定时间后会过期失效。您可以使用 describe 命令在 default-secret 的中查看到具体的过期时间，如下所示。

须知

在使用时请直接使用 default-secret，而不要拷贝 secret 内容重新创建，因为 secret 里面的凭据会过期，从而导致无法拉取镜像。

```
$ kubectl describe secret default-secret
Name:        default-secret
Namespace:    default
Labels:      secret-generated-by=cce
Annotations:  temporary-ak-sk-expires-at: 2021-11-26 20:55:31.380909 +0000 UTC

Type:  kubernetes.io/dockerconfigjson

Data
====
.dockerconfigjson:  347 bytes
```

paas.elb

paas.elb 的 data 内容是临时 AK/SK 数据，用于创建 Service 和 Ingress 时创建 ELB，paas.elb 的 data 数据同样会定期更新，且在一定时间后会过期失效。

实际使用中您不会直接使用 paas.elb，但请不要删除 paas.elb，否则会导致创建 ELB 失败。

default-token-xxxxx

Kubernetes 为每个命名空间默认创建一个名为 default 的 ServiceAccount，default-token-xxxxx 为这个 ServiceAccount 的密钥，xxxxx 是随机数。

```
$ kubectl get sa
NAME     SECRETS   AGE
```

```
default  1        30d
$ kubectl describe sa default
Name:              default
Namespace:          default
Labels:            <none>
Annotations:        <none>
Image pull secrets:  <none>
Mountable secrets:   default-token-vssmw
Tokens:            default-token-vssmw
Events:            <none>
```

14 弹性伸缩

14.1 弹性伸缩概述

弹性伸缩是根据业务需求和策略，经济地自动调整弹性计算资源的管理服务。

背景介绍

随着 Kubernetes 已经成为云原生应用编排、管理的事实标准，越来越多的应用选择向 Kubernetes 迁移，用户也越来越关心在 Kubernetes 上应用如何快速扩容面对业务高峰，以及如何在业务低谷时快速缩容节约资源与成本。

在 Kubernetes 的集群中，“弹性伸缩”一般涉及到扩缩容 Pod 个数以及 Node 个数。Pod 代表应用的实例数（每个 Pod 包含一个或多个容器），当业务高峰的时候需要扩容应用的实例个数。所有的 Pod 都是运行在某一个节点（虚机或裸机）上，当集群中没有足够多的节点来调度新扩容的 Pod，那么就需要为集群增加节点，从而保证业务能够正常提供服务。

弹性伸缩在 CCE 上的使用场景非常广泛，典型的场景包含在线业务弹性、大规模计算训练、深度学习 GPU 或共享 GPU 的训练与推理、定时周期性负载变化等。

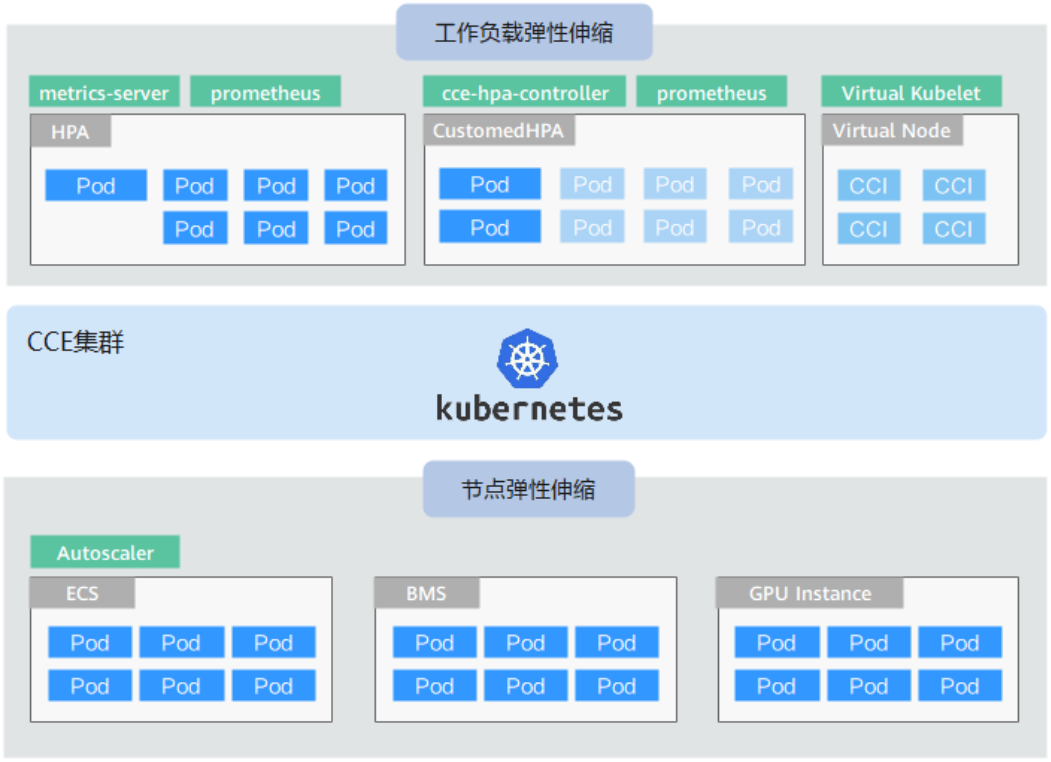
CCE 弹性伸缩

CCE 的弹性伸缩能力分为如下两个维度：

- **工作负载弹性伸缩**：即调度层弹性，主要是负责修改负载的调度容量变化。例如，HPA 是典型的调度层弹性组件，通过 HPA 可以调整应用的副本数，调整的副本数会改变当前负载占用的调度容量，从而实现调度层的伸缩。
- **节点弹性伸缩**：即资源层弹性，主要是集群的容量规划不能满足集群调度容量时，会通过弹出 ECS 等资源的方式进行调度容量的补充。

两个维度的弹性组件与能力可以分开使用，也可以结合在一起使用，并且两者之间可以通过调度层面的容量状态进行解耦。

组件介绍



工作负载弹性组件介绍

表14-1 工作负载弹性组件

类型	组件名称	组件介绍	参考文档
HPA	metrics-server	Kubernetes 内置组件，实现 Pod 水平自动伸缩的功能，即 Horizontal Pod Autoscaling。在 kubernetes 社区 HPA 功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。	创建工作负载弹性伸缩（HPA）
CustomedHPA	cce-hpa-controller	自研的弹性伸缩增强能力，主要面向无状态工作负载进行弹性扩缩容。能够基于指标（CPU 利用率、内存利用率）或周期（每天、每周、每月或每年的具体时间点）。	创建工作负载弹性伸缩（CustomedHPA）
	prometheus	一套开源的系统监控报警框架，负责采集 kubernetes 集群中 kubelet 的公开指标项（CPU 利用率、内存利用率）。	

节点弹性伸缩组件介绍

表14-2 节点弹性组件

组件名称	组件介绍	适用场景	参考文档
autoscaler	Kubernetes 社区开源组件，节点水平伸缩组件，提供了独有的调度、弹性优化、成本优化的功能。	全场景支持，适合在线业务、深度学习、大规模成本算力交付等。	节点自动伸缩

14.2 工作负载弹性伸缩

14.2.1 工作负载伸缩原理

CCE 支持 HPA 策略和 CustomedHPA 策略两种工作负载伸缩方式。两种策略的对比如下：

表14-3 HPA 和 CustomedHPA 策略对比

伸缩策略	HPA 策略	CustomedHPA 策略
实现方式	Kubernetes 中实现 POD 水平自动伸缩的功能，即 Horizontal Pod Autoscaling	自研的弹性伸缩增强能力
策略规则	基于**指标**（CPU 利用率、内存利用率），对无状态工作负载进行弹性扩缩容。	基于**指标**（CPU 利用率、内存利用率）或**周期**（每天、每周、每月或每年的具体时间点），对无状态工作负载进行弹性扩缩容。
主要功能	在 kubernetes 社区 HPA 功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。	**指标触发** • 支持按照当前实例数的百分比进行扩缩容。 • 支持设置一次扩缩容的最小步长，可分步分级扩缩容。 • 支持按照实际指标值执行不同的扩缩容动作。 **周期触发** 支持选择天、周、月或年的具体时间点或周期作为触发时间

HPA 工作原理

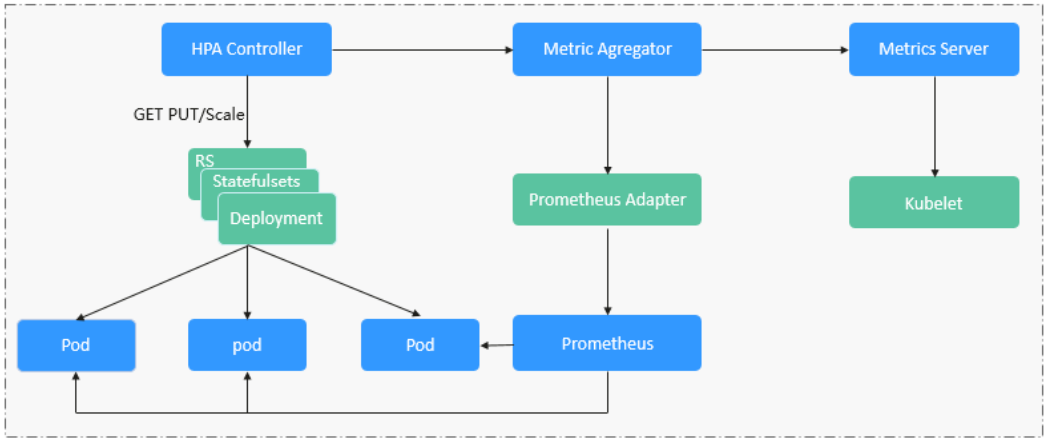
HPA（Horizontal Pod Autoscaler）是用来控制 Pod 水平伸缩的控制器，HPA 周期性检查 Pod 的度量数据，计算满足 HPA 资源所配置的目标数值所需的副本数量，进而调整目标资源（如 Deployment）的 replicas 字段。

想要做到自动弹性伸缩，先决条件就是能感知到各种运行数据，例如集群节点、Pod、容器的 CPU、内存使用率等等。而这些数据的监控能力 Kubernetes 也没有自己实现，而是通过其他项目来扩展 Kubernetes 的能力，CCE 提供 Prometheus 和 Metrics Server 插件来实现该能力：

- Prometheus 是一套开源的系统监控报警框架，能够采集丰富的 Metrics（度量数据），目前已经基本是 Kubernetes 的标准监控方案。
- Metrics Server 是 Kubernetes 集群范围资源使用数据的聚合器。Metrics Server 从 kubelet 公开的 Summary API 中采集度量数据，能够收集包括了 Pod、Node、容器、Service 等主要 Kubernetes 核心资源的度量数据，且对外提供一套标准的 API。

使用 HPA（Horizontal Pod Autoscaler）配合 Metrics Server 可以实现基于 CPU 和内存的自动弹性伸缩，再配合 Prometheus 还可以实现自定义监控指标的自动弹性伸缩。

图14-1 HPA 流程图



HPA 的核心有如下 2 个部分：

- 监控数据来源

  最早社区只提供基于 CPU 和 Mem 的 HPA，随着应用越来越多搬迁到 k8s 上以及 Prometheus 的发展，开发者已经不满足于 CPU 和 Memory，开发者需要应用自身的业务指标，或者是一些接入层的监控信息，例如：Load Balancer 的 QPS、网站的实时在线人数等。社区经过思考之后，定义了一套标准的 Metrics API，通过聚合 API 对外提供服务。

  - metrics.k8s.io： 主要提供 Pod 和 Node 的 CPU 和 Memory 相关的监控指标。
  - custom.metrics.k8s.io： 主要提供 Kubernetes Object 相关的自定义监控指标。
  - external.metrics.k8s.io：指标来源外部，与任何的 Kubernetes 资源的指标无关。

- 扩缩容决策算法

  HPA controller 根据当前指标和期望指标来计算缩放比例，计算公式如下：

  desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]

  例如当前的指标值是 200m，目标值是 100m，那么按照公式计算期望的实例数就会翻倍。那么在实际过程中，可能会遇到实例数值反复伸缩，导致集群震荡。为了保证稳定性，HPA controller 从以下几个方面进行优化：

  - 冷却时间：在 1.11 版本以及之前的版本，社区引入了 horizontal-pod-autoscaler-downscale-stabilization-window 和 horizontal-pod-autoScaler-upscale-stabilization-window 这两个启动参数代表缩容冷却时间和扩容冷却时间，这样保证在冷却时间内，跳过扩缩容。1.14 版本之后引入延迟队列，保存一段时间内每一次检测的决策建议，然后根据当前所有有效的决策建议来进行决策，从而保证期望的副本数尽量小的发生变更，保证稳定性。
  - 忍受度：可以看成一个缓冲区，当实例变化范围在忍受范围之内的话，保持原有的实例数不变。

    首先定义 ratio = currentMetricValue / desiredMetricValue

    当|ratio – 1.0| <= tolerance 时，则会忽略，跳过 scale。

    当|ratio – 1.0| > tolerance 时， 就会根据之前的公式计算期望值。

    当前社区版本中默认值为 0.1

HPA 是基于指标阈值进行伸缩的，常见的指标主要是 CPU、内存，当然也可以通过自定义指标，例如 QPS、连接数等进行伸缩。但是存在一个问题：基于指标的伸缩存在一定的时延，这个时延主要包含：采集时延(分钟级) + 判断时延(分钟级) + 伸缩时延(分钟级)。这个分钟级的时延，可能会导致应用 CPU 飚高，响应时间变慢。为了解决这个问题，CCE 提供了定时策略，对于一些有周期性变化的应用，提前扩容资源，而业务低谷时，定时回收资源。

14.2.2 创建工作负载弹性伸缩（HPA）

HPA 策略即 Horizontal Pod Autoscaling，是 Kubernetes 中实现 POD 水平自动伸缩的功能。该策略在 kubernetes 社区 HPA 功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。

前提条件

使用 HPA 需要安装能够提供 Metrics API 的插件，如 metrics-server 或 Prometheus。

约束与限制

- HPA 策略：仅支持 1.13 及以上版本的集群创建。
- 每个工作负载只能创建一个策略，即如果您创建了一个 HPA 策略，则不能再对其创建 CustomedHPA 策略或其他 HPA 策略，您可以删除该 HPA 策略后再创建。
- 1.19.10 以下版本的集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，当新 Pod 被调度到另一个节点时，会导致之前 Pod 不能正常读写。

  1.19.10 及以上版本集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，新 Pod 会因为无法挂载云硬盘导致无法成功启动。

操作步骤

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“负载伸缩”，在右上角单击“创建 HPA 策略”。

步骤 3 填写策略参数。

表14-4 HPA 策略参数配置

参数	参数说明
策略名称	新建策略的名称，请自定义。
命名空间	请选择工作负载所在的命名空间。
关联工作负载	请选择要设置 HPA 策略的工作负载。
实例范围	请输入最小实例数和最大实例数。 策略触发时，工作负载实例将在此范围内伸缩。
冷却时间	请输入缩容和扩容的冷却时间，单位为分钟，**缩容扩容冷却时间不能小于 1 分钟**。 **该设置仅在 1.15 及以上版本的集群中显示，1.13 版本的集群不支持该设置。** 策略成功触发后，在此缩容/扩容冷却时间内，不会再次触发缩容/扩容，目的是等待伸缩动作完成后在系统稳定且集群正常的情况下进行下一次策略匹配。
策略规则	策略规则可基于系统指标或自定义指标。 **系统策略** • 指标：可选择“CPU 利用率”或“内存利用率”。 **说明** 利用率 = 工作负载 Pod 的实际使用量 / 申请量。 • 期望值：请输入期望资源平均利用率。 期望值表示所选指标的期望值，通过向上取整(当前指标值 / 期望值 × 当前实例数)来计算需要伸缩的实例数。 • 阈值：请输入缩容和扩容阈值。 当指标值大于缩容阈值且小于扩容阈值时，不会触发扩容或缩容。**阈值仅在 1.15 及以上版本的集群中支持**。 **自定义策略（仅在 1.15 及以上版本的集群中支持）** • 自定义指标名称：自定义指标的名称，输入时可根据联想值进行选择。 • 指标来源：在下拉框中选择对象类型，可选择“Pod”。 • 期望值：Pod 支持指标为平均值。 • 伸缩范围：当指标值处于伸缩范围中时才会触发伸缩 **说明**

参数	参数说明
	HPA 在计算扩容、缩容实例数时，会选择最近 5 分钟内实例数的最大值。

步骤 4 设置完成后，单击“创建”，在“完成”步骤中若显示“创建工作负载策略***提交成功”，可单击“返回工作负载伸缩策略”。

步骤 5 在“工作负载伸缩”页签下，可以看到刚刚创建的 HPA 策略。

----结束

14.2.3 创建工作负载弹性伸缩（CustomedHPA）

CustomedHPA 策略是自研的弹性伸缩增强能力，能够基于指标（CPU 利用率、内存利用率）或周期（每天、每周、每月或每年的具体时间点），对无状态工作负载进行弹性扩缩容。

主要功能如下：

- 支持按照当前实例数的百分比进行扩缩容。
- 支持设置一次扩缩容的最小步长。
- 支持按照实际指标值执行不同的扩缩容动作。

前提条件

使用 CustomedHPA 策略必须安装 cce-hpa-controller。若 cce-hpa-controller 版本低于 1.2.11，则必须安装 prometheus 插件；若版本大于或等于 1.2.11，则需要安装能够提供 Metrics API 的插件，如 kube-prometheus-stack、metrics-server 或 Prometheus。

约束与限制

- CustomedHPA 策略：仅支持 1.15 及以上版本的集群创建。
- 每个工作负载只能创建一个策略，即如果您创建了一个 HPA 策略，则不能再对其创建 CustomedHPA 策略或其他 HPA 策略，您可以删除该 HPA 策略后再创建。
- 1.19.10 以下版本的集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，当新 Pod 被调度到另一个节点时，会导致之前 Pod 不能正常读写。

  1.19.10 及以上版本集群中，如果使用 HPA 策略对挂载了 EVS 卷的负载进行扩容，新 Pod 会因为无法挂载云硬盘导致无法成功启动。
- cce-hpa-controller 插件的资源使用量主要受集群中总容器数量和伸缩策略数量影响，通常场景下建议每 5000 容器配置 CPU 500m，内存 1000Mi 资源，每 1000 伸缩策略 CPU 100m，内存 500Mi。
- 若 cce-hpa-controller 插件版本低于 1.2.11，不支持使用 kube-prometheus-stack 插件提供 Metrics API 来实现工作负载弹性伸缩。

操作步骤

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“负载伸缩”，选择“CustomedHPA 策略”。

- 若插件名称后方显示“未安装”，请单击插件后方的“安装”按钮，根据业务需求配置插件参数后单击“安装”，等待插件安装完成。
- 若插件名称后方显示“已安装”，则说明插件已安装成功。

步骤 3 确认插件已安装成功后，单击右上角“创建 CustomedHPA 策略”。

步骤 4 填写策略参数。

表14-5 CustomedHPA 策略参数配置

参数	参数说明
策略名称	新建策略的名称，请自定义。
命名空间	请选择工作负载所在的命名空间。
关联工作负载	请选择要设置 CustomedHPA 策略的工作负载。
实例范围	请输入最小实例数和最大实例数。 策略触发时，工作负载实例将在此范围内伸缩。
冷却时间	请输入冷却时间值，单位为分钟。 策略成功触发后，在此冷却时间内，不会再次触发缩容/扩容，目的是等待伸缩动作完成后在系统稳定且集群正常的情况下进行下一次策略匹配。
策略规则	单击 ┼ 在弹出的窗口中设置伸缩策略参数： • 规则名称：请输入规则名称，可自定义。 • 类型：可选择“指标触发”或“周期触发”。 **指标触发** • 触发条件：请选择“CPU 利用率”或“内存利用率”，选择“>”或“<”，并输入百分比的值。如下图中所示，则表示 CPU 利用率瞬时值 > 50% 时，立即执行此规则。 **说明** 利用率 = 工作负载的实际使用量 / 申请量。 **图14-2 触发条件** 

参数	参数说明
	• 执行动作：与上述“触发条件”相对应，达到触发条件值后所要执行的动作，可添加多个执行动作。如下图中所示，当 CPU 利用率超过 50%时将伸缩至 5 个实例，当超过 70%时伸缩至 8 个实例，当超过 90%时在 8 个实例基础上再增加 10 个实例。反之，按此规则执行缩容。 图14-3 执行动作  • 是否启用：可单击启用或关闭该策略规则。 **周期触发** • 触发时间：可选择每天、每周、每月或每年的具体时间点，如设置为下图所示，则为每天 17:00 触发。 图14-4 周期触发-每天  • 执行动作：与上述“触发时间”相对应，达到触发时间值后所要执行的动作。如下图中所示，即每天 17:00 时将执行减少 4 个实例的动作。 图14-5 周期触发-执行动作

参数	参数说明
	 • 是否启用：可单击启用或关闭该策略规则。 单击确定后，您可以在“策略规则”列表中查看添加的规则，并可执行开启关闭、编辑、删除等操作。 单击“策略规则”列表下方的“添加策略规则”，可设置多条策略。

步骤 5 设置完成后，单击“创建”。

----结束

使用 kubectl 创建

CustomHPA 是一种 CRD 资源，可按如下 YAML 定义。

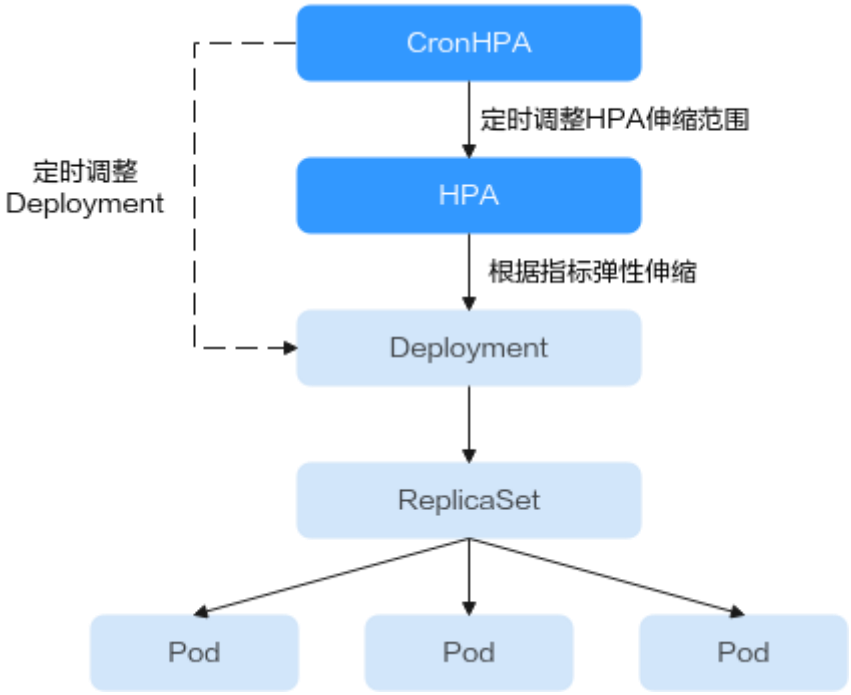
```
apiVersion: autoscaling.cce.io/v1alpha1
kind: CustomedHorizontalPodAutoscaler
metadata:
  name: customhpa-example
  namespace: default
spec:
  coolDownTime: 3m            # 冷却时间
  maxReplicas: 10             # 最大实例数
  minReplicas: 1              # 最小实例数
  rules:
    - actions:                          # 策略规则
        - metricRange: 0,0.1              # 指标范围，表示从 0 到 10%
          operationType: ScaleDown        # 伸缩类型，ScaleDown 表示减少
          operationUnit: Task             # 操作单位，Task 表示个数
          operationValue: 1               # 每次伸缩的数量
        - metricRange: 0.1,0.3            # 指标范围，表示从 10%到 30%
```

```
        operationType: ScaleDown
        operationUnit: Task
        operationValue: 2
    disable: false
    metricTrigger:
      hitThreshold: 1
      metricName: CPURatioToRequest   # 指标名称，CPURatioToRequest 为 CPU 利用率
      metricOperation: <            # 指标表达式操作符
      metricValue: 0.3              # 指标表达式右侧取值
      periodSeconds: 60             #
      statistic: instantaneous       #
    ruleName: low
    ruleType: Metric
  - actions:
      - metricRange: 0.7,0.9
        operationType: ScaleUp
        operationUnit: Task
        operationValue: 1
      - metricRange: 0.9,+Infinity
        operationType: ScaleUp
        operationUnit: Task
        operationValue: 2
    disable: false
    metricTrigger:
      hitThreshold: 1
      metricName: CPURatioToRequest
      metricOperation: '>'
      metricValue: 0.7
      periodSeconds: 60
      statistic: instantaneous
    ruleName: high
    ruleType: Metric
scaleTargetRef:             # 关联负载
  apiVersion: apps/v1
  kind: Deployment
  name: nginx
```

14.2.4 CronHPA 定时策略

概述

在一些复杂的业务场景下，可能有固定时间段高峰业务，又有日常突发高峰业务。此种情况下，用户既期望能定时弹性伸缩应对固定时间段高峰业务，又期望能根据指标弹性伸缩应对日常突发高峰业务。CCE 提供 CronHPA 的自定义资源，实现在固定时间段对集群进行扩缩容，并且可以和 HPA 策略共同作用，定时调整 HPA 伸缩范围，实现复杂场景下的工作负载伸缩。



CronHPA 支持定时调整 HPA 策略的最大和最小实例数，也可以直接定时调整 Deployment 的 Pod 实例数。

CronHPA 的 YAML 示例如下：

```
apiVersion: autoscaling.cce.io/v2alpha1
kind: CronHorizontalPodAutoscaler
metadata:
  name: ccetest
  namespace: default
spec:
   scaleTargetRef:                # 关联 HPA 策略或 Deployment
      apiVersion: autoscaling/v1
      kind: HorizontalPodAutoscaler
      name: hpa-test
   rules:
   - ruleName: "scale-down"
     schedule: "15 * * * *"        # 指定任务运行时间与周期，参数格式请参见 Cron，例如 0 * *
* * 或@hourly。
     targetReplicas: 1             # 目标实例数量
     disable: false
   - ruleName: "scale-up"
     schedule: "13 * * * *"
     targetReplicas: 6
     disable: false
```

表14-6 CronHPA 关键字段说明

字段	说明
apiVersion	API 版本，固定值“autoscaling.cce.io/v2alpha1”。
kind	API 类型，固定值“CronHorizontalPodAutoscaler”。

字段	说明
metadata.name	CronHPA 策略名称。
metadata.namespace	CronHPA 策略所在的命名空间。
spec.scaleTargetRef	指定 CronHPA 的扩缩容对象，可配置以下字段： • apiVersion：CronHPA 扩缩容对象的 API 版本。 • kind：CronHPA 扩缩容对象的 API 类型。 • name：CronHPA 扩缩容对象的名称。 CronHPA 支持 HPA 策略或 Deployment。
spec.rules	CronHPA 策略规则，可添加多个规则。每个规则可配置以下字段： • ruleName：CronHPA 规则名称，该名称需唯一。 • schedule：指定任务运行时间与周期，参数格式与 CronTab 类似，请参见 Cron，例如 0 * * * * 或@hourly。 • targetReplicas：扩缩容的 Pod 数目。 • disable：参数值为“true”或“false”。其中“false”表示该规则生效，“true”则表示该规则不生效。

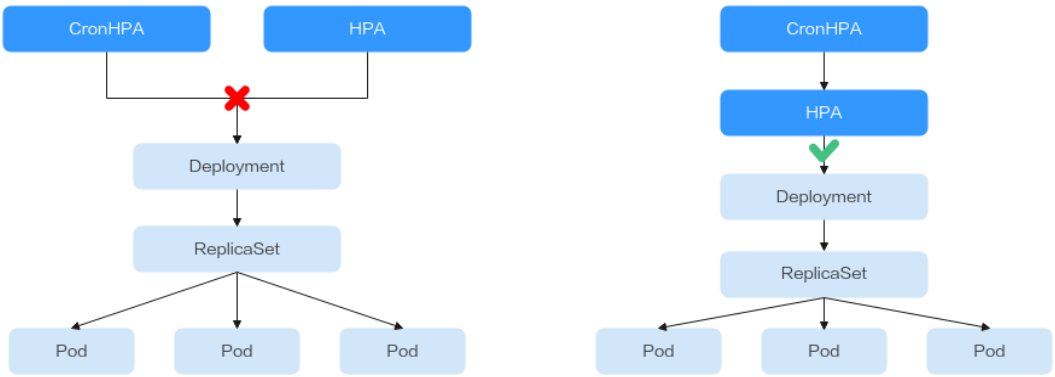
前提条件

已安装 1.2.13 及以上版本 cce-hpa-controller。

使用 CronHPA 调整 HPA 伸缩范围

CronHPA 支持定时调整 HPA 策略的最大和最小实例数，满足复杂场景下的工作负载伸缩。

由于 HPA 与 CronHPA 均通过 scaleTargetRef 字段来获取伸缩对象，如果 CronHPA 和 HPA 同时设置 Deployment 为伸缩对象，两个伸缩策略相互独立，后执行的操作会覆盖先执行的操作，导致伸缩效果不符合预期，因此需避免这种情况发生。



当 CronHPA 与 HPA 兼容使用时，需要将 CronHPA 中的 scaleTargetRef 字段设置为 HPA 策略，而 HPA 策略的 scaleTargetRef 字段设置为 Deployment，这样 CronHPA 策略

会在固定的时间调整 HPA 策略的实例数量上下限，即可实现工作负载定时伸缩和弹性伸缩的兼容。

步骤 1 为 Deployment 创建 HPA 策略。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-test
  namespace: default
spec:
  maxReplicas: 10              #  最大实例数
  minReplicas: 5              #  最小实例数
  scaleTargetRef:             #  关联 Deployment
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  targetCPUUtilizationPercentage: 50
```

步骤 2 创建 CronHPA 策略，并关联步骤 1 中创建的 HPA 策略。

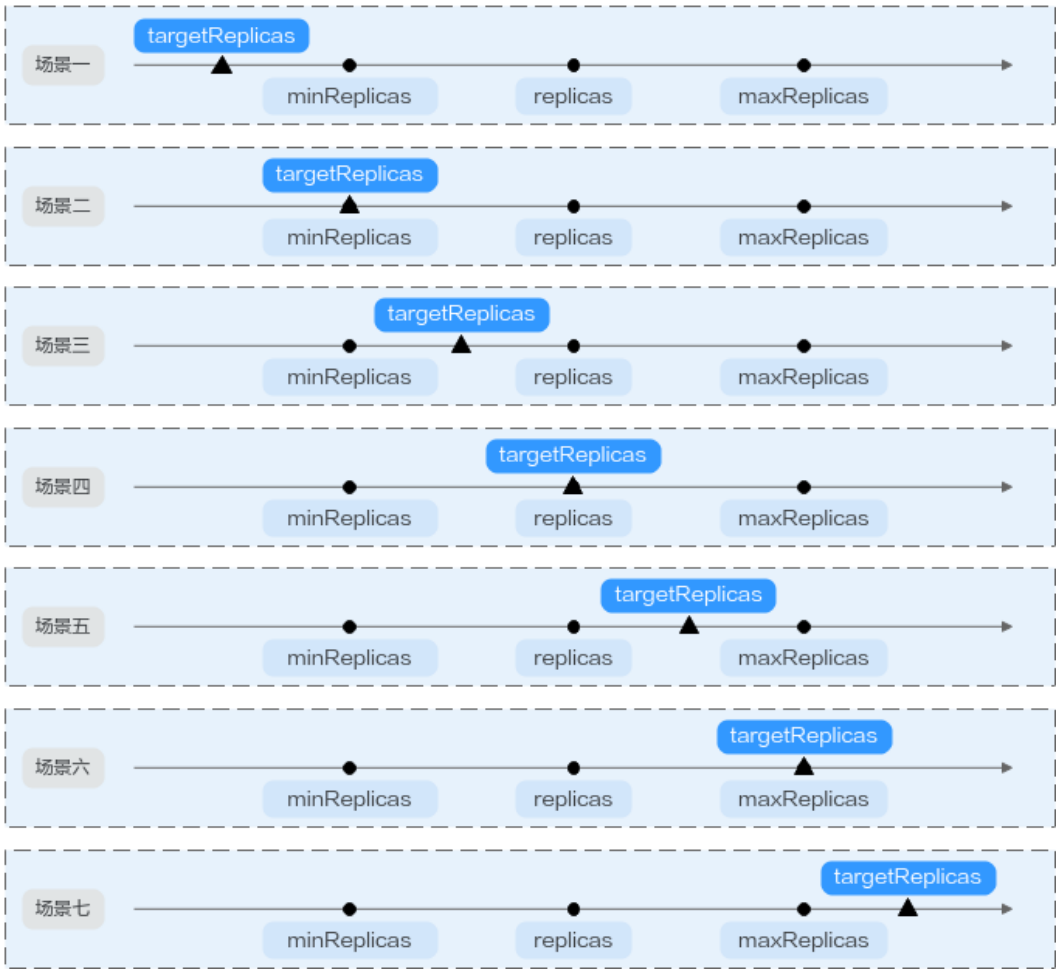
```
apiVersion: autoscaling.cce.io/v2alpha1
kind: CronHorizontalPodAutoscaler
metadata:
  name: ccetest
  namespace: default
spec:
   scaleTargetRef:                # 关联 HPA 策略
      apiVersion: autoscaling/v1
      kind: HorizontalPodAutoscaler
      name: hpa-test
   rules:
   - ruleName: "scale-down"
     schedule: "15 * * * *"         # 指定任务运行时间与周期，参数格式请参见 Cron，例如 0 * *
* * 或@hourly。
     targetReplicas: 1             # 目标实例数量
     disable: false
   - ruleName: "scale-up"
     schedule: "13 * * * *"
     targetReplicas: 11
     disable: false
```

在 CronHPA 与 HPA 共同使用时，CronHPA 规则是在 HPA 策略的基础上生效的，CronHPA 不会直接调整 Deployment 的副本数目，而是通过 HPA 来操作 Deployment，因此了解以下参数可帮助您更好地理解其工作原理。

- CronHPA 的目标实例数（targetReplicas）：表示 CronHPA 设定的实例数，在 CronHPA 生效时用于调整 HPA 的最大/最小实例数，从而间接调整 Deployment 实例数。
- HPA 的最小实例数（minReplicas）：Deployment 的实例数下限。
- HPA 的最大实例数（maxReplicas）：Deployment 的实例数上限。
- Deployment 的实例数（replicas）：CronHPA 策略生效之前 Deployment 的 Pod 数量。

在 CronHPA 规则生效时，通过比较目标实例数（targetReplicas）与实际 Deployment 的实例数，并结合 HPA 的最小实例数或最小实例数的数值大小，来调整 Deployment 实例数的上下限值。

图14-6 CronHPA 扩缩容场景



上图为可能存在的扩缩容场景，如下表格以举例的形式说明了不同场景下 CronHPA 修改 HPA 的情况。

场景	场景说明	Cronhpa (targetReplicas)	Deployment (replicas)	HPA (minReplicas / maxReplicas)	最终结果	操作说明
场景一	**targetReplicas** < minReplicas ≤ replicas ≤ maxReplicas	4	5	5/10	HPA: 4/10 Deployment: 5	CronHPA 目标实例数低于 HPA 最小实例数（minReplicas）时：

场景	场景说明	Cronhpa (targetReplicas)	Deployment (replicas)	HPA (minReplicas / maxReplicas)	最终结果	操作说明
						• 修改 HPA 的最小实例数。 • Deployment 实例数无修改。
场景二	**targetReplicas** = minReplicas ≤ replicas ≤ maxReplicas	5	6	5/10	HPA: 5/10 Deployment: 6	CronHPA 目标实例数等于 HPA 最小实例数（minReplicas）时： • HPA 的最小实例数无修改。 • Deployment 实例数无修改。
场景三	minReplicas < **targetReplicas** < replicas ≤ maxReplicas	4	5	1/10	HPA: 4/10 Deployment: 5	CronHPA 目标实例数大于 HPA 最小实例数（minReplicas），小于 Deployment 实例数（replicas）时： • 修改 HPA 的最小实例数。 • Deployment 实例数无修改。
场景四	minReplicas < **targetReplicas** = replicas < maxReplicas	5	5	1/10	HPA: 5/10 Deployment: 5	CronHPA 目标实例数大于 HPA 最小实例数（minReplicas），等于 Deployment 实例数（replicas）时： • 修改 HPA 的最小实例数。 • Deployment 实例数无修改。
场景	minReplicas ≤ replicas < **targetReplicas** <	6	5	1/10	HPA: 6/10	CronHPA 目标实例数大于 Deployment

场景	场景说明	Cronhpa (targetReplicas)	Deployment (replicas)	HPA (minReplicas / maxReplicas)	最终结果	操作说明
五	maxReplicas				Deployment: 6	实例数（replicas），小于HPA 最大实例数（maxReplicas）时： • 修改 HPA 的最小实例数。 • 修改 Deployment 实例数。
场景六	minReplicas ≤ replicas < **targetReplicas** = maxReplicas	10	5	1/10	HPA: 10/10 Deployment: 10	CronHPA 目标实例数大于 Deployment 实例数（replicas），等于HPA 最大实例数（maxReplicas）时： • 修改 HPA 的最小实例数。 • 修改 Deployment 实例数。
场景七	minReplicas ≤ replicas ≤ maxReplicas < **targetReplicas**	11	5	5/10	HPA: 11/11 Deployment: 11	CronHPA 目标实例数大于 HPA 最大实例数（maxReplicas）时： • 修改 HPA 的最小实例数。 • 修改 HPA 的最大实例数。 • 修改 Deployment 实例数。

----结束

使用 CronHPA 直接调整 Deployment 实例数量

CronHPA 还可以单独调整关联 Deployment，定时调整 Deployment 的实例数，使用方法如下。

```
apiVersion: autoscaling.cce.io/v2alpha1
kind: CronHorizontalPodAutoscaler
metadata:
  name: ccetest
  namespace: default
spec:
   scaleTargetRef:           # 关联 Deployment
      apiVersion: apps/v1
      kind: Deployment
      name: nginx
   rules:
   - ruleName: "scale-down"
     schedule: "08 * * * *"    # 指定任务运行时间与周期，参数格式请参见 Cron，例如 0 * * * * 或@hourly。
     targetReplicas: 1
     disable: false
   - ruleName: "scale-up"
     schedule: "05 * * * *"
     targetReplicas: 3
     disable: false
```

14.2.5 管理工作负载伸缩策略

操作场景

HPA 或 CustomedHPA 策略创建完成后，可对创建的策略进行更新、克隆、编辑 YAML 以及删除等操作。

查看 HPA 策略

您可以查看 HPA 策略的规则、状态和事件，参照界面中的报错提示有针对性的解决异常事件。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，在“HPA 策略”页签下，单击要查看的 HPA 策略前方的 ⌄。

步骤 3 在展开的区域中，可以看到规则、状态和事件页签，若策略异常，请参照界面中的报错提示进行定位处理。

说明

您还可以在工作负载详情页中查看已创建的 HPA 策略，登录 CCE 控制台，在左侧导航栏中单击“工作负载 > 无状态负载 Deployment”，单击工作负载名称，在该工作负载详情页的“弹性伸缩”页签下可以看到为该工作负载配置的弹性伸缩-HPA 和 CustomedHPA 策略。

表14-7 事件类型及名称

事件类型	事件名称	描述
正常	SuccessfulRescale	扩缩容成功
异常	InvalidTargetRange	无效的 TargetRange
	InvalidSelector	无效选择器
	FailedGetObjectMetric	获取对象失败数
	FailedGetPodsMetric	获取 Pod 列表失败指标
	FailedGetResourceMetric	获取资源失败数
	FailedGetExternalMetric	获取外部指标失败
	InvalidMetricSourceType	无效的指标来源类型
	FailedConvertHPA	转换 HPA 失败
	FailedGetScale	获取比例尺失败
	FailedComputeMetricsReplicas	计算指标副本数失败
	FailedGetScaleWindow	获取 ScaleWindow 失败
	FailedRescale	扩缩容失败

----结束

查看 CustomedHPA 策略

您可以查看 CustomedHPA 策略的规则和最新状态，参照界面中的报错提示有针对性的解决异常事件。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，在“CustomHPA 策略”页签下，单击要查看的 CustomedHPA 策略前方的。

步骤 3 在展开的区域中，可以看到规则页签，若策略异常，请单击“最新状态”栏中的“详情”，参照展开的详细信息进行定位处理。

说明

您还可以在工作负载详情页中查看已创建的 CustomedHPA 策略，登录 CCE 控制台，在左侧导航栏中单击“工作负载 > 无状态负载 Deployment”，单击工作负载后方“操作”中的“更多 > 伸缩”，在该工作负载详情页的“伸缩”页签下可以看到“弹性伸缩 - HPA / CustomedHPA”，您在“弹性伸缩”页面配置的 CustomedHPA 策略也会在这里显示。

----结束

更新 HPA 或 CustomedHPA 策略

以 HPA 策略为例。

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，单击策略后方“操作”栏中的“更新”。

步骤 3 在打开的“更新工作负载 HPA 策略”页面中，并更新策略参数。

步骤 4 单击“更新”完成策略更新。

----结束

编辑 YAML（HPA 策略）

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，单击 HPA 策略后方“操作”栏中的“更多 > 编辑 YAML”。

步骤 3 在弹出的“编辑 YAML”窗口中，可以对 YAML 进行修改和下载。

步骤 4 在右上角关闭窗口完成操作。

----结束

查看 YAML（CustomedHPA 策略）

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，选择“CustomHPA 策略”页签，单击 CustomedHPA 策略后的“ 查看 YAML”。

步骤 3 在弹出的“查看 YAML”窗口中，可以对 YAML 进行复制和下载，不能对其修改。

步骤 4 在右上角关闭窗口完成操作。

----结束

删除 HPA 或 CustomedHPA 策略

步骤 1 登录 CCE 控制台，单击集群名称进入集群。

步骤 2 在左侧导航栏中单击“负载伸缩”，单击策略后方栏中的“删除”。

步骤 3 在弹出的窗口中，单击“是”完成删除操作。

----结束

14.3 集群/节点弹性伸缩

14.3.1 节点伸缩原理

HPA 是针对 Pod 级别的，但是如果集群的资源不够了，那就只能对节点进行扩容了。集群节点的弹性伸缩本来是一件非常麻烦的事情，但是好在现在的集群大多都是构建在云上，云上可以直接调用接口添加删除节点，这就使得集群节点弹性伸缩变得非常方便。

Autoscaler 是 Kubernetes 提供的集群节点弹性伸缩组件，根据 Pod 调度状态及资源使用情况对集群的节点进行自动扩容缩容。

前提条件

使用节点伸缩功能前，需要安装 autoscaler 插件，插件版本要求 1.13.8 及以上。

Autoscaler 工作原理

Autoscaler（简称 CA）的主要流程包括两部分：

- ScaleUp 流程： CA 会每隔 10s 检查一次所有不可调度的 Pod，根据用户设置的策略，选择出一个符合要求的节点组进行扩容。
- ScaleDown 流程：CA 每隔 10s 会扫描一次所有的 Node，如果该 Node 上所有的 Pod Requests 少于用户定义的缩容百分比时，CA 会模拟将该节点上的 Pod 是否能迁移到其他节点，如果可以的话，当满足不被需要的时间窗以后，该节点就会被移除。

如上所述，当集群节点处于一段时间空闲状态时（默认 10min），会触发集群缩容操作。

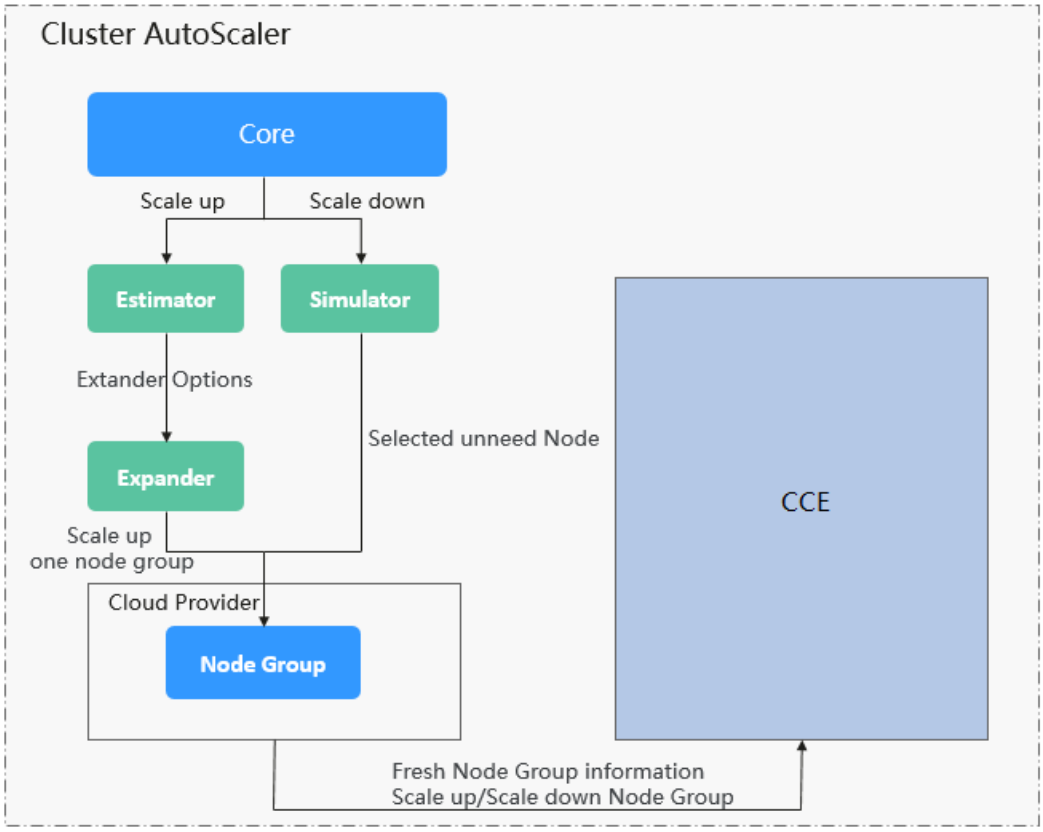
当节点存在以下几种状态的 pod 时，不可缩容：

1. 节点上的 pod 有设置 PodDisruptionBudget，当移除 pod 不满足对应条件时，节点不会缩容。
2. 节点上的 pod 由于一些限制，如亲和、反亲和或者 taints 等，无法调度到其他节点，节点不会缩容。
3. 节点上的 pod 拥有"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"这个 annotations 时，节点不缩容。
4. 节点上存在 kube-system namespace 下的 Pod（除 kube-system daemonset 创建的 Pod），节点不缩容。
5. 节点上有非 controller（deployment/replica set/job/stateful set）创建的 Pod，节点不缩容。

AutoScaler 架构

AutoScaler 架构如下图所示，主要由以下几个核心模块组成：

图14-7 AutoScaler 架构图



说明如下：

- Estimator： 负责扩容场景下，评估满足当前不可调度 Pod 时，每个节点池需要扩容的节点数量。
- Simulator： 负责缩容场景下，找到满足缩容条件的节点。
- Expander： 负责在扩容场景下，根据用户设置的不同的策略来，从 Estimator 选出的节点池中，选出一个最佳的选择。当前 Expander 有多种策略：
  - Random： 随机选择一个节点池，如果用户没有设置的话，默认是 Random。
  - most-Pods： 选择此次扩容后能满足调度最多 Pod 的节点池，如果有相同的，再随机选择一个。
  - least-waste： 选择此次扩容完成后，具有最小浪费的 CPU 或者 Mem 资源的节点池。
  - price： 选择此次扩容所需节点金额最小的节点池。
  - priority： 根据用户自定义的权重，选择权重最高的节点池。

当前 CCE 提供除 Price 以外的所有的策略，当前 CCE 插件默认使用 least-waste 策略。

14.3.2 创建节点伸缩策略

CCE 的自动伸缩能力是通过节点自动伸缩组件 autoscaler 实现的，可以按需弹出节点实例，支持多可用区、多实例规格、多种伸缩模式，满足不同的节点伸缩场景。

当节点伸缩中创建的策略和 autoscaler 插件中的配置同时生效时（比如不可调度和指标规则同时满足时），将优先执行不可调度扩容。

- 若不可调度执行成功，则会跳过指标规则逻辑，进入下一次循环。
- 若不可调度执行失败，将执行指标规则逻辑。

前提条件

使用节点伸缩功能前，需要安装 autoscaler 插件，插件版本要求 1.13.8 及以上。

约束限制

- 仅按需计费节点池支持弹性伸缩。
- 弹性伸缩的策略作用在节点池，当节点池中节点为 0 时，且按 CPU/内存弹性伸缩时，不会触发节点伸缩。
- 缩容节点会导致与节点关联的本地持久存储卷类型的 PVC/PV 数据丢失，无法恢复，且 PVC/PV 无法再正常使用。缩容节点时使用了本地持久存储卷的 Pod 会从缩容的节点上被驱逐，并重新创建 Pod，Pod 会一直处于 pending 状态，因为 Pod 使用的 PVC 带有节点标签，由于冲突无法调度成功。

操作步骤

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，进入创建节点伸缩策略页面。

- 若插件名称后方显示“未安装”，请单击插件后方的“安装”，根据业务需求配置插件参数后单击“安装”，等待插件安装完成。
- 若插件名称后方显示“已安装”，则说明插件已安装成功。

步骤 3 单击右上角“创建节点伸缩策略”，参照如下参数设置策略。

- 策略名称：新建策略的名称，请自定义。
- 关联节点池：选择要关联的节点池。您可以关联多个节点池，以使用相同的伸缩策略。

说明

节点池新增了优先级功能，弹性扩容时 CCE 将按照如下策略来选择节点池进行扩容：

1. 通过预判算法判断节点池是否能满足让 Pending 的 Pod 正常调度的条件，包括节点资源大于 Pod 的 request 值、nodeSelect、nodeAffinity 和 taints 等是否满足 Pod 正常调度的条件；另外还会过滤掉扩容失败（因为资源不足等原因）还处于 15min 冷却时间的节点池。
2. 有多个节点池满足条件时，判断节点池设置的优先级（优先级默认值为 0，取值范围为 0-100，其中 100 为最高，0 为最低），选择优先级最高的节点池扩容。
3. 如果有多个节点池处于相同的优先级，或者都没有配置优先级时，通过最小浪费原则，根据节点池里设置的虚拟机规格，计算刚好能满足 Pending 的 Pod 正常调度，且浪费资源最少的节点池。
4. 如果还是有多个节点池的虚拟机规格都一样，只是 AZ 不同，那么会随机选择其中一个节点池触发扩容。
5. 如果出现优先选择的节点池资源不足，会按照优先级顺序自动选择下一个节点池。

节点池优先级功能详情请参见创建节点池。

- 执行规则：单击“添加规则”，在弹出的添加规则窗口中设置如下参数：

  规则名称：请输入规则名称，可自定义。

  规则类型：可选择“指标触发”或“周期触发”，两种类型区别如下：

  - **指标触发：**

    触发条件：请选择“CPU 分配率”或“内存分配率”，输入百分比的值。该百分比应大于 autoscaler 插件中配置的缩容百分比。

说明

- 分配率 = 节点池容器组（Pod）资源申请量 / 节点池 Pod 可用资源量 (Node Allocatable) 。
- **如果多条规则同时满足条件，会有如下两种执行的情况：**

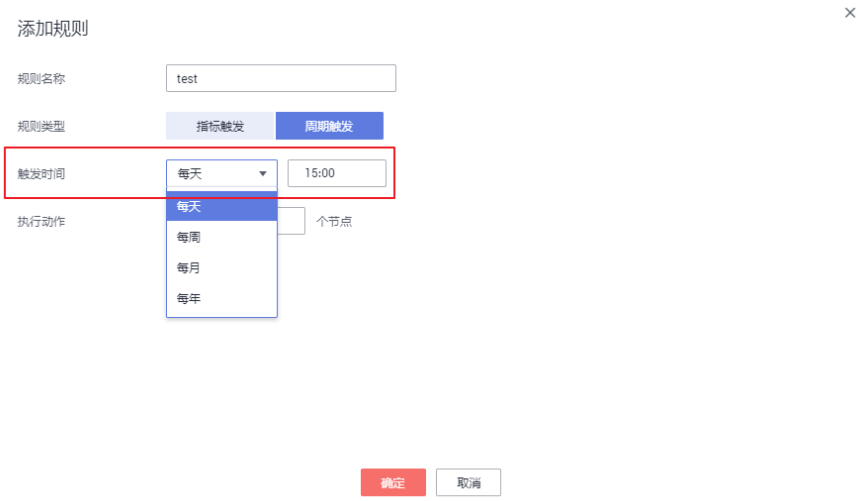
  如果同时配置了“CPU 分配率”和“内存分配率”的规则，两种或多种规则同时满足扩容条件时，执行扩容节点数更多的规则。

  如果同时配置了“CPU 分配率”和“周期触发”的规则，当达到“周期触发”的时间值时 CPU 也满足扩容条件时，较早执行的 A 规则会将节点池状态置为伸缩中状态，导致 B 规则无法正常执行。待 A 规则执行完毕，节点池状态恢复正常后，B 规则也不会执行。
- 配置了“CPU 分配率”和“内存分配率”的规则后，策略的检测周期会因 autoscaler 每次循环的处理逻辑而变动。只要一次检测出满足条件就会触发扩容（当然还要满足冷却时间、节点池状态等约束条件）。

  - **周期触发：**

    触发时间：可选择每天、每周、每月或每年的具体时间点，如下图所示，则为每天 15:00 触发。

图14-8 周期触发时间



执行动作：与上述“触发条件”或“触发时间”相对应，达到触发条件或触发时间值后所要执行的动作。

您可以单击“添加规则”，设置多条节点伸缩策略。您最多可以添加 1 条 CPU 使用率指标规则、1 条内存使用率指标规则，且规则总数小于等于 10 条。

步骤 4 设置完成后，单击“确定”。

----结束

缩容说明

节点伸缩策略中不能直接设置缩容策略，您可以在安装 autoscaler 插件时进行设置。

节点缩容仅支持资源分配率缩容机制，当集群下 CPU 和内存分配率低于您设置的门限（在安装/编辑 autoscaler 插件时设置）时将对节点池下节点启动缩容（该功能可以关闭）。

图14-9 自动缩容设置

YAML 样例

节点伸缩策略 Yaml 样例如下：

```
apiVersion: autoscaling.cce.io/v1alpha1
kind: HorizontalNodeAutoscaler
metadata:
  creationTimestamp: "2020-02-13T12:47:49Z"
  generation: 1
  name: xxxx
  namespace: kube-system
  resourceVersion: "11433270"
  selfLink: /apis/autoscaling.cce.io/v1alpha1/namespaces/kube-
system/horizontalnodeautoscalers/xxxx
  uid: c2bd1e1d-60aa-47b5-938c-6bf3fadbe91f
spec:
  disable: false
  rules:
  - action:
      type: ScaleUp
      unit: Node
      value: 1
    cronTrigger:
```

```
    schedule: 47 20 * * *
  disable: false
  ruleName: cronrule
  type: Cron
- action:
    type: ScaleUp
    unit: Node
    value: 2
  disable: false
  metricTrigger:
    metricName: Cpu
    metricOperation: '>'
    metricValue: "40"
    unit: Percent
  ruleName: metricrule
  type: Metric
targetNodepoolIds:
- 7d48eca7-3419-11ea-bc29-0255ac1001a8
```

表14-8 关键参数说明

参数	参数类型	描述
spec.disable	Bool	伸缩策略开关，会对策略中的所有规则生效
spec.rules	Array	伸缩策略中的所有规则
spec.rules[x].ruleName	String	规则名称
spec.rules[x].type	String	规则类型，当前支持“Cron”和“Metric”两种类型
spec.rules[x].disable	Bool	规则开关，当前仅支持“false”
spec.rules[x].action.type	String	规则操作类型，当前仅支持“ScaleUp”
spec.rules[x].action.unit	String	规则操作单位，当前仅支持“Node”
spec.rules[x].action.value	Integer	规则操作数值
spec.rules[x].cronTrigger	/	可选，仅在周期规则中有效
spec.rules[x].cronTrigger.schedule	String	周期规则的 cron 表达式
spec.rules[x].metricTrigger	/	可选，仅在指标规则中有效
spec.rules[x].metricTrigger.metricName	String	指标规则对应的指标，当前支持“Cpu”和“Memory”两种类型
spec.rules[x].metricTrigger.metricOperation	String	指标规则的比较符，当前仅支持“>”

参数	参数类型	描述
spec.rules[x].metricTrigger.metricValue	String	指标规则的阈值，支持 1-100 之间的所有整数，需以字符串表示
spec.rules[x].metricTrigger.Unit	String	指标规则阈值的单位，当前仅支持“%”
spec.targetNodepoolIds	Array	伸缩策略关联的所有节点池
spec.targetNodepoolIds[x]	String	伸缩策略关联节点池的 uid

14.3.3 管理节点伸缩策略

操作场景

节点伸缩策略创建完成后，可对创建的策略进行删除、编辑、停用、启用、克隆等操作。

查看节点伸缩策略

您可以查看节点伸缩策略的关联节点池、执行规则和伸缩历史，参照界面中的提示有针对性的解决异常问题。

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，单击要查看的策略前方的。

步骤 3 在展开的区域中，可以看到该策略的关联节点池、执行规则和伸缩历史页签，若策略异常，请参照界面中的报错提示进行定位处理。

说明

您还可以在节点池管理中关闭或开启弹性扩缩容，登录 CCE 控制台，在左侧导航栏中单击“资源管理 > 节点池管理”，单击要操作的节点池右上角的“编辑”，在弹出的“编辑节点池”窗口中的可以看到“弹性扩缩容”按钮，并可设置节点数上下限和弹性缩容冷却时间。

----结束

删除节点伸缩策略

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，单击要删除的策略后方的“更多 > 删除”。

步骤 3 在弹出的“删除节点伸缩策略”窗口中，确认是否删除。

步骤 4 单击“是”按钮即完成删除操作。

----结束

编辑节点伸缩策略

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，单击要编辑的策略后方的“编辑”。

步骤 3 在打开的“编辑节点伸缩策略”页面中，更新策略参数。

步骤 4 完成设置后，单击“确定”按钮完成编辑操作。

----结束

克隆节点伸缩策略

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，单击要克隆的策略后方的“更多 > 克隆”。

步骤 3 在打开的“拷贝节点伸缩策略”页面中，可以看到部分参数已经克隆过来，请按照业务需求补充或修改其他策略参数。

步骤 4 单击“确定”完成策略克隆。

----结束

停用/启用节点伸缩策略

步骤 1 在 CCE 控制台，单击集群名称进入集群。

步骤 2 单击左侧导航栏的“节点伸缩”，单击策略后方的“停用”，若策略为停用状态时，则单击策略后方的“启用”。

步骤 3 在弹出的“停用节点策略”或“启用节点策略”窗口中，确认是否进行停用或启用操作。

----结束

15 插件管理

15.1 插件概述

CCE 提供了多种类型的插件，用于管理集群的扩展功能，以支持选择性扩展满足特性需求的功能。

表15-1 插件列表

插件名称	插件简介
coredns（系统资源插件，必装）	CoreDNS 插件是一款通过链式插件的方式为 Kubernetes 提供域名解析服务的 DNS 服务器。
everest（系统资源插件，必装）	Everest 是一个云原生容器存储系统，基于 CSI 为 Kubernetes v1.15.6 及以上版本集群对接云存储服务的能力。
npd（系统资源插件）	node-problem-detector（简称：npd）是一款监控集群节点异常事件的插件，以及对接第三方监控平台功能的组件。它是一个在每个节点上运行的守护程序，可从不同的守护进程中搜集节点问题并将其报告给 apiserver。node-problem-detector 可以作为 DaemonSet 运行，也可以独立运行。
dashboard	Kubernetes Dashboard 是 Kubernetes 集群基于 Web 的通用 UI，集合了命令行可以操作的所有命令。它允许用户管理在集群中运行应用程序并对其进行故障排除，以及管理集群本身。
autoscaler	集群自动扩缩容插件 autoscaler，是根据 pod 调度状态及资源使用情况对集群的工作节点进行自动扩容缩容的插件。
metrics-server	Metrics-Server 是集群核心资源监控数据的聚合器。
cce-hpa-controller	cce-hpa-controller 插件是一款 CCE 自研的插件，能够基于 CPU 利用率、内存利用率等指标，对无状态工作负载进行弹性扩缩容。
prometheus	Prometheus 是一套开源的系统监控报警框架。在云容器引擎 CCE 中，支持以插件的方式快捷安装 Prometheus。
web-terminal	web-terminal 是一款支持在 Web 界面上使用 Kubectl 的插件。它

插件名称	插件简介
	支持使用 WebSocket 通过浏览器连接 Linux，提供灵活的接口便于集成到独立系统中，可直接作为一个服务连接，通过 cmdb 获取信息并登录服务器。
gpu-beta	gpu-beta 插件是支持在容器中使用 GPU 显卡的设备管理插件，仅支持 Nvidia 驱动。
volcano	Volcano 提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力，通过接入 AI、大数据、基因、渲染等诸多行业计算框架服务终端用户。
nginx-ingress	nginx-ingress 为 Service 提供了可直接被集群外部访问的虚拟主机、负载均衡、SSL 代理、HTTP 路由等应用层转发功能。
dolphin	dolphin 是一款容器网络流量监控管理插件，当前版本可支持从 CCE Turbo 集群安全容器以及运行时为 containerd 的普通容器的流量统计。 当前支持流量统计信息 ipv4 发送公网报文数和字节数、ipv4 接收报文数和字节数以及 ipv4 发送报文数和字节数，且支持通过 PodSelector 来对监控后端作选择，支持多监控任务、可选监控指标，且支持用户获取 Pod 的 label 标签信息。监控信息已适配 Prometheus 格式，可以通过调用 Prometheus 接口查看监控数据。
node-local-dns	NodeLocal DNSCache 通过在集群节点上作为守护程序集运行 DNS 缓存代理，提高集群 DNS 性能。

15.2 coredns（系统资源插件，必装）

插件简介

CoreDNS 插件是一款通过链式插件的方式为 Kubernetes 提供域名解析服务的 DNS 服务器。

CoreDNS 是由 CNCF 孵化的开源软件，用于 Cloud-Native 环境下的 DNS 服务器和服务发现解决方案。CoreDNS 实现了插件链式架构，能够按需组合插件，运行效率高、配置灵活。在 kubernetes 集群中使用 CoreDNS 能够自动发现集群内的服务，并为这些服务提供域名解析。同时，通过级联云上 DNS 服务器，还能够为集群内的工作负载提供外部域名的解析服务。

该插件为系统资源插件，kubernetes 1.11 及以上版本的集群在创建时默认安装。

目前 CoreDNS 已经成为社区 kubernetes 1.11 及以上版本集群推荐的 DNS 服务器解决方案。

CoreDNS 官网：https://coredns.io/

开源社区地址：https://github.com/coredns/coredns

说明

DNS 详细使用方法请参见 DNS。

约束与限制

CoreDNS 正常运行或升级时，请确保集群中的可用节点数大于等于 CoreDNS 的实例数，且 CoreDNS 的所有实例都处于运行状态，否则将导致插件异常或升级失败。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **coredns**，单击“安装”。

步骤 2 在安装插件页面，选择插件规格，并配置相关参数。

表15-2 coredns 插件参数配置

参数	参数说明
插件规格	并发域名解析能力，请根据业务需求选择插件规格。 如选择“自定义 qps”，CoreDNS 所能提供的域名解析 QPS 与 CPU 消耗成正相关，请根据业务需求，合理调整插件实例数和容器 CPU/内存配额。
实例数	选择上方插件规格后，显示插件中的实例数。
容器	选择插件规格后，显示插件容器的 CPU 和内存配额。
参数配置	• parameterSyncStrategy：插件升级时是否配置一致性检查。 – ensureConsistent：表示启用配置一致性检查，如果集群中记录的配置和实际生效配置不一致，插件将无法升级。 – force：表示升级时忽略配置一致性检查。请您自行确保当前生效配置和原配置一致。插件升级完毕后，需恢复 parameterSyncStrategy 值为 ensureConsistent，重新启用配置一致性检查。 • stub_domains：存根域，您可对自定义的域名配置域名服务器，格式为一个键值对，键为 DNS 后缀域名，值为一个或一组 DNS IP 地址。 • upstream_nameservers：上游域名服务器地址。 • servers: coredns 1.23.1 插件版本开始开放 servers 配置，用户可对 servers 做定制化配置，参见 dns-custom-nameservers，其中 plugins 为 coredns 中各组件配置（参考 https://coredns.io/manual/plugins/，一般场景建议保持默认配置，以免出现配置错误而导致 coredns 整体不可用），每个 plugin 组件可包含"name"、"parameters"(可选)、"configBlock"(可选)配置，对应生成的 Corefile 配置文件中格

参数	参数说明

	式如下:
	$name $parameters {
	$configBlock
	}
	示例：
	{
	"servers": [
	{
	"plugins": [
	{
	"name": "bind",
	"parameters": "{$POD_IP}"
	},
	{
	"name": "cache",
	"parameters": 30
	},
	{
	"name": "errors"
	},
	{
	"name": "health",
	"parameters": "{$POD_IP}:8080"
	},
	{
	"configBlock": "pods insecure\nfallthrough in-addr.arpa ip6.arpa",
	"name": "kubernetes",
	"parameters": "cluster.local in-addr.arpa ip6.arpa"
	},
	{
	"name": "loadbalance",
	"parameters": "round_robin"
	},
	{
	"name": "prometheus",
	"parameters": "{$POD_IP}:9153"
	},
	{
	"configBlock": "policy random",
	"name": "forward",
	"parameters": ". /etc/resolv.conf"
	},
	{
	"name": "reload"
	},
	{
	"name": "log"
	}
	],
	"port": 5353,

参数	参数说明

	<pre>"zones": [
	{
	"zone": "."
	}
	]
	}
	],
	"stub_domains": {
	"acme.local": [
	"1.2.3.4",
	"6.7.8.9"
	]
	},
	"upstream_nameservers": ["8.8.8.8", "8.8.4.4"]
	}</pre>

表15-3 coredns 主 zone 默认 plugin 配置说明

plugin 名称	描述
bind	coredns 侦听的 hostIP 配置，建议保持当前默认值{$POD_IP}。
cache	启用 DNS 缓存。
errors	错误信息到标准输出。
health	coredns 健康检查配置，当前侦听{$POD_IP}:8080，请保持此默认值，否则导致 coredns 健康检查失败而不断重启
kubernetes	CoreDNS Kubernetes 插件，提供集群内服务解析能力。
loadbalance	轮转式 DNS 负载均衡器， 在应答中随机分配 A、AAAA 和 MX 记录的顺序。
prometheus	CoreDNS 自身 metrics 数据接口，默认 zone 侦听{$POD_IP}:9153，请保持此默认值，否则普罗无法采集 coredns metrics 数据。
forward	不在 Kubernetes 集群域内的任何查询都将转发到默认的解析器 (/etc/resolv.conf)。
reload	允许自动重新加载已更改的 Corefile。 编辑 ConfigMap 配置后，请等待两分钟，以使更改生效。

步骤 3 完成以上配置后，单击“安装”。

----结束

kubernetes 中的域名解析逻辑

DNS 策略可以在每个 pod 基础上进行设置，目前，Kubernetes 支持 **Default、ClusterFirst、ClusterFirstWithHostNet 和 None** 四种 DNS 策略，具体请参见 https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/。这些策略在 pod-specific 的 **dnsPolicy** 字段中指定。

- **“Default”**：如果 dnsPolicy 被设置为“Default”，则名称解析配置将从 pod 运行的节点继承。 自定义上游域名服务器和存根域不能够与这个策略一起使用。
- **“ClusterFirst”**：如果 dnsPolicy 被设置为“ClusterFirst”，任何与配置的集群域后缀不匹配的 DNS 查询（例如，www.kubernetes.io）将转发到从该节点继承的上游名称服务器。集群管理员可能配置了额外的存根域和上游 DNS 服务器。
- **“ClusterFirstWithHostNet”**：对于使用 hostNetwork 运行的 Pod，您应该明确设置其 DNS 策略“ClusterFirstWithHostNet”。
- **“None”**：它允许 Pod 忽略 Kubernetes 环境中的 DNS 设置。应使用 dnsConfigPod 规范中的字段提供所有 DNS 设置 。

说明

- Kubernetes 1.10 及以上版本，支持 Default、ClusterFirst、ClusterFirstWithHostNet 和 None 四种策略；低于 Kubernetes 1.10 版本，仅支持 default、ClusterFirst 和 ClusterFirstWithHostNet 三种。
- “Default”不是默认的 DNS 策略。如果 dnsPolicy 的 Flag 没有特别指明，则默认使用“**ClusterFirst**”。

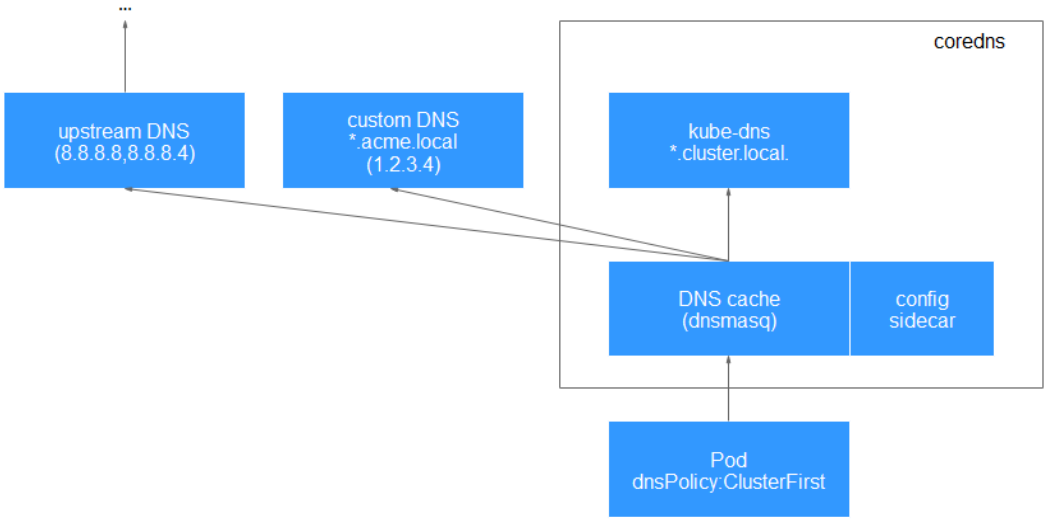
路由请求流程：

未配置存根域：没有匹配上配置的集群域名后缀的任何请求，例如“www.kubernetes.io”，将会被转发到继承自节点的上游域名服务器。

已配置存根域：如果配置了存根域和上游 DNS 服务器，DNS 查询将基于下面的流程对请求进行路由：

1. 查询首先被发送到 coredns 中的 DNS 缓存层。
2. 从缓存层，检查请求的后缀，并根据下面的情况转发到对应的 DNS 上：
   - 具有集群后缀的名字（例如“.cluster.local”）：请求被发送到 coredns。
   - 具有存根域后缀的名字（例如“.acme.local”）：请求被发送到配置的自定义 DNS 解析器（例如：监听在 1.2.3.4）。
   - 未能匹配上后缀的名字（例如“widget.com”）：请求被转发到上游 DNS。

图15-1 路由请求流程



版本记录

表15-4 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.23.2	/v1.(15|17|19|21|23).*/	1.8.4
1.23.1	/v1.(15|17|19|21|23).*/	1.8.4
1.17.15	/v1.(15|17|19|21).*/	1.8.4
1.17.9	/v1.(15|17|19).*/	1.8.4
1.17.7	/v1.(15|17|19).*/	1.8.4
1.17.4	/v1.(17|19).*/	1.6.5
1.17.3	/v1.17.*/	1.6.5
1.17.1	/v1.17.*/	1.6.5

15.3 storage-driver（系统资源插件，已废弃）

插件简介

storage-driver 是一款云存储驱动插件，北向遵循标准容器平台存储驱动接口。实现 Kubernetes Flex Volume 标准接口，提供容器使用 EVS 块存储、SFS 文件存储、OBS

对象存储、SFS Turbo 极速文件存储的能力。通过安装升级云存储插件可以实现云存储功能的快速安装和更新升级。

该插件为系统资源插件，kubernetes 1.13 及以下版本的集群在创建时默认安装。

约束与限制

- 在 CCE 所创的集群中，Kubernetes v1.15.11 版本是 Flexvolume 插件（storage-driver）被 CSI 插件（Everest）兼容接管的过渡版本，v1.17 及之后版本的集群中将彻底去除对 Flexvolume 插件（storage-driver）的支持，请您将 Flexvolume 插件的使用切换到 CSI 插件上。
- CSI 插件（Everest）兼容接管 Flexvolume 插件（storage-driver）后，原有功能保持不变，但请注意不要新建 Flexvolume 插件（storage-driver）的存储，否则将导致部分存储功能异常。
- 本插件仅支持在 **v1.13 及以下版本**的集群中安装，v1.15 及以上版本的集群在创建时默认安装 Everest 插件。

说明

在 **v1.13 及以下版本**的集群中，当存储功能有升级或者 BUG 修复时，用户无需升级集群或新建集群来升级存储功能，仅需安装或升级 storage-driver 插件。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

未安装 storage-driver 插件的集群，可参考如下步骤进行安装：

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **storage-driver**，单击“安装”。

步骤 2 云存储插件暂未开放可配置参数，直接单击“安装”。

----结束

15.4 everest（系统资源插件，必装）

插件简介

Everest 是一个云原生容器存储系统，基于 CSI（即 Container Storage Interface）为 Kubernetes v1.15.6 及以上版本集群对接云存储服务的能力。

该插件为系统资源插件，kubernetes 1.15 及以上版本的集群在创建时默认安装。

约束与限制

- 集群版本由 v1.13 升级到 v1.15 后，v1.13 版本集群中的 Flexvolume 容器存储插件（storage-driver）能力将由 v1.15 的 CSI 插件（Everest，插件版本 v1.1.6 及以上）接管，接管后原有功能保持不变。

- 插件版本为 1.2.0 的 Everest 优化了使用 OBS 存储时的**密钥认证功能**，低于该版本的 Everest 插件在升级完成后，需要重启集群中使用 OBS 存储的全部工作负载，否则工作负载使用存储的能力将受影响！
- **v1.15 及以上版本**的集群默认安装本插件，v1.13 及以下版本集群创建时默认安装 storage-driver 插件。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **everest**，单击“安装”。

步骤 2 该插件可配置“单实例”、“高可用”或自定义规格。

Everest 插件包含以下容器，您可根据需求自定义调整规格：

- everest-csi-controller：由 Deployment 形式部署，Pod 中含有一个 everest-csi-controller 容器，此容器负责存储卷的创建、删除、快照、扩容、attach/detach 等功能。若集群版本大于等于 1.19，everest-csi-driver 组件的 Pod 还会默认带有一个 everest-localvolume-manager 容器，此容器负责管理节点上的 lvm 存储池及 localpv 的创建。

说明

选择自定义规格时，everest-csi-controller 内存配置推荐如下。

- Pod 和 PVC 的数量均小于 2000 时，everest-csi-controller 的内存上限推荐配置为 600Mi。
- Pod 和 PVC 的数量均小于 5000 时，everest-csi-controller 的内存上限推荐配置为 1Gi。

- everest-csi-driver：由 DaemonSet 形式部署，Pod 中含有一个基本容器 everest-csi-driver 容器，负责 PV 的挂载、卸载、文件系统 resize 等功能。若集群所在区域支持 node-attacher，everest-csi-driver 组件的 Pod 还会带有一个 everest-node-attacher 的容器，此容器负责分布式 attach EVS，该配置项在部分 Region 开放。

说明

选择自定义规格时，everest-csi-driver 内存限制推荐配置不低于 300Mi。若该值太小可能导致插件实例容器启动异常，从而导致插件不可用的情况。

步骤 3 参数配置。

everest 1.2.26 以上版本针对大批量挂 EVS 卷的性能做了优化，提供了如下 3 个参数供用户配置。

- csi_attacher_worker_threads：everest 插件中同时处理挂 EVS 卷的 worker 数，默认值为“60”。
- csi_attacher_detach_worker_threads：everest 插件中同时处理卸载 EVS 卷的 worker 数，默认值均为“60”。
- volume_attaching_flow_ctrl：everest 插件在 1 分钟内可以挂载 EVS 卷的最大数量，此参数的默认值“0”表示 everest 插件不做挂卷限制，此时挂卷性能由底层存储资源决定。

上述三个参数由于存在关联性且与集群所在局点的底层存储资源限制有关，当您对大批量挂卷的性能有要求（大于 500EVS 卷/分钟）时，请联系后台工程师，在指导下进行配置，否则可能会因为参数配置不合理导致出现 everest 插件运行不正常的情况。

步骤 4 单击“安装”。

----结束

版本记录

表15-5 CCE 插件版本记录

插件版本	支持的集群版本
1.3.28	/v1.(19|21|23).*/
1.3.22	/v1.(19|21|23).*/
1.3.20	/v1.(19|21|23).*/
1.3.17	/v1.(19|21|23).*/
1.3.8	/v1.23.*/
1.3.6	/v1.23.*/
1.2.55	/v1.(15|17|19|21).*/
1.2.53	/v1.(15|17|19|21).*/
1.2.51	/v1.(15|17|19|21).*/
1.2.44	/v1.(15|17|19|21).*/
1.2.42	/v1.(15|17|19|21).*/
1.2.30	/v1.(15|17|19|21).*/
1.2.28	/v1.(15|17|19|21).*/
1.2.27	/v1.(15|17|19|21).*/
1.2.13	/v1.(15|17|19).*/
1.2.9	/v1.(15|17|19).*/
1.2.5	/v1.(15|17|19).*/
1.1.12	/v1.(15|17).*/
1.1.11	/v1.(15|17).*/
1.1.8	/v1.(15|17).*/
1.1.7	/v1.(15|17).*/

15.5 npd

插件简介

node-problem-detector（简称：npd）是一款监控集群节点异常事件的插件，以及对接第三方监控平台功能的组件。它是一个在每个节点上运行的守护程序，可从不同的守护进程中搜集节点问题并将其报告给 apiserver。node-problem-detector 可以作为 DaemonSet 运行， 也可以独立运行。

有关社区开源项目 node-problem-detector 的详细信息，请参见 node-problem-detector。

约束与限制

- 使用 NPD 插件时，不可对节点磁盘进行格式化或分区。
- 节点上每个 NPD 进程标准占用 30mCPU，100MB 内存。

权限说明

NPD 插件为监控内核日志，需要读取宿主机/dev/kmsg 设备，为此需要开启容器特权，详见 privileged。

同时 CCE 根据最小化权限原则进行了风险消减，NPD 运行限制只拥有以下特权：

- cap_dac_read_search，为访问/run/log/journal
- cap_sys_admin，为访问/dev/kmsg

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **npd**，单击“安装”。

步骤 2 根据下表配置参数，然后单击“安装”。

仅 v1.16.0 及以上版本支持配置。

npc.enable：是否启用 npc，默认值为 **false**，**true** 表示启用。

----结束

NPD 检查项

说明

当前检查项仅 1.16.0 及以上版本支持。

NPD 的检查项主要分为事件类检查项和状态类检查项。

- 事件类检查项

  对于事件类检查项，当问题发生时，NPD 会向 APIServer 上报一条事件，事件类型分为 Normal（正常事件）和 Warning（异常事件）

表15-6 事件类检查项

故障检查项	功能	说明
KubeletStart	检查 kubelet 启动并上报	Normal 类事件
DockerStart	检查 Docker 启动并上报	Normal 类事件
ContainerdStart	检查 Containerd 启动并上报	Normal 类事件
OOMKilling	检查 oom 事件发生并上报	Warning 类事件
TaskHung	检查 taskHung 事件发生并上报	Warning 类事件
KernelOops	检查内核 0 指针 panic 错误	Warning 类事件
ConntrackFull	检查连接跟踪表是否满	Warning 类事件 周期：30 秒 阈值:80%

- 状态类检查项

  对于状态类检查项，当问题发生时，NPD 会向 APIServer 上报一条事件，并同步修改节点状态，可配合 Node-problem-controller 故障隔离对节点进行隔离。

  下列检查项中若未明确指出检查周期，则默认周期为 30 秒。

表15-7 应用和 OS 类检查项

故障检查项	功能	说明
FrequentKubeletRestart	通过监听 journald 日志检测 kubelet 是否频繁重启	• 周期：5 分钟 • 回溯时间：10 分钟 • 阈值：10 次 即在回溯时间段内重启 10 次表示频繁重启，将会产生故障告警。 • 监听对象：/run/log/journal 目录下的日志 **说明** Ubuntu 操作系统由于日志格式不兼容，暂不支持上述检查项。
FrequentDockerRestart	通过监听 journald 日志检测 docker 是否频繁重启	
FrequentContainerdRestart	通过监听 journald 日志检测 containerd 是否频繁重启	
CRIProblem	检查容器 CRI 组件状态	检查对象：Docker 或 Containerd
KUBELETProblem	检查 Kubelet 状态	无
NTPProblem	检查 ntp、chrony 服务状态 检查节点时钟是否偏移	时钟偏移阈值：8000ms

故障检查项	功能	说明
PIDProblem	检查 Pid 是否充足	• 阈值：90% • 使用量：/proc/loadavg 中 nr_threads • 最大值：/proc/sys/kernel/pid_max 和/proc/sys/kernel/threads-max 两者的较小值。
FDProblem	检查文件句柄数是否充足	• 阈值：90% • 使用量：/proc/sys/fs/file-nr 中第 1 个值 • 最大值：/proc/sys/fs/file-nr 中第 3 个值
MemoryProblem	检查节点整体内存是否充足	• 阈值：90% • 使用量：/proc/meminfo 中 MemTotal-MemAvailable • 最大值：/proc/meminfo 中 MemTotal
ResolvConfFileProblem	检查 ResolvConf 配置文件是否丢失 检查 ResolvConf 配置文件是否异常 异常定义：不包含任何上游域名解析服务器（nameserver）。	检查对象：/etc/resolv.conf
ProcessD	检查节点是否存在 D 进程	数据来源： • /proc/{PID}/stat • 等效命令：ps aux 例外场景：ProcessD 忽略 BMS 节点下的 SDI 卡驱动依赖的常驻 D 进程 heartbeat、update
ProcessZ	检查节点是否存在 Z 进程	
ScheduledEvent	检查节点是否存在主机计划事件 典型场景：底层宿主机异常，例如风扇损坏、磁盘坏道等，导致其上虚机触发冷热迁移。	数据来源： • http://169.254.169.254/meta-data/latest/events/scheduled 该检查项为 Alpha 特性，默认不开启。

表15-8 网络连通类检查项

故障检查项	功能	说明
CNIProblem	检查容器 CNI 组件是否正常运行	无
KUBEPROXYProblem	检查 Kube-proxy 是否正常运行	无

表15-9 存储类检查项

故障检查项	功能	说明
ReadonlyFilesystem	监听内核日志，检查系统内核是否有 Remount root filesystem read-only 错误。 典型场景：用户从 ECS 侧误操作卸载节点数据盘，且应用程序对该数据盘的对应挂载点仍有持续写操作，触发内核产生 IO 错误将磁盘重挂载为只读磁盘。	监听对象：/dev/kmsg 匹配规则："Remounting filesystem read-only"
DiskReadonly	检查节点系统盘、CCE 数据盘（包含 docker 逻辑盘与 kubelet 逻辑盘）是否只读	检测路径： • /mnt/paas/kubernetes/kubelet/ • /var/lib/docker/ • /var/lib/containerd/ • /var/paas/sys/log/cceaddon-npd/ 检测路径下会产生临时文件 npd-disk-write-ping 当前暂不支持额外的数据盘
DiskProblem	检查节点系统盘、CCE 数据盘（包含 docker 逻辑盘与 kubelet 逻辑盘）的磁盘使用情况	• 阈值：80% • 数据来源： `df -h` 当前暂不支持额外的数据盘
EmptyDirVolumeGroupStatusError	检查节点上临时卷存储池是否正常 故障影响：依赖存储池的 Pod 无法正常写对应临时卷。临时卷由于 IO 错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为临时卷存	• 检测周期：60 秒 • 数据来源： `vgs -o vg_name, vg_attr` • 检测原理：检查 VG（存储池）是否存在 p 状态，该状态表征部分 PV（数据盘）丢失。

故障检查项	功能	说明
	储池，用户误操作删除了部分数据盘导致存储池异常。	• 调度联动：调度器可自动识别此异常状态并避免依赖存储池的 Pod 调度到该节点上。 • 例外场景：NPD 无法检测所有 PV（数据盘）丢失，导致 VG（存储池）丢失的场景；此时依赖 kubelet 自动隔离该节点，其检测到 VG（存储池）丢失并更新 nodestatus.allocatable 中对应资源为 0，避免依赖存储池的 Pod 调度到该节点上。无法检测单个 PV 损坏；此时依赖 ReadonlyFilesystem 检测异常。
LocalPvVolumeGroupStatusError	检查节点上持久卷存储池是否正常 故障影响：依赖存储池的 Pod 无法正常写对应持久卷。持久卷由于 IO 错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为持久卷存储池，用户误操作删除了部分数据盘。	
MountPointProblem	检查节点上的挂载点是否异常 异常定义：该挂载点不可访问（cd） 典型场景：节点挂载了 nfs（网络文件系统，常见有 obsfs、s3fs 等），当由于网络或对端 nfs 服务器异常等原因导致连接异常时，所有访问该挂载点的进程均卡死。例如集群升级场景 kubelet 重启时扫描所有挂载点，当扫描到此异常挂载点会卡死，导致升级失败。	等效检查命令： `for dir in `df -h | grep -v "Mounted on" | awk "{print \\$NF}"`;do cd $dir; done && echo "ok"`
DiskHung	检查节点磁盘是否存在卡 IO 故障 卡 IO 定义：系统对磁盘的 IO 请求下发后未有响应，部分进程卡在 D 状态 典型场景：操作系统硬盘驱动异常或底层网络严重故障导致磁盘无法响应	• 检查对象：所有数据盘 • 数据来源： /proc/diskstat 等效查询命令： `iostat -xmt 1` • 阈值： – 平均利用率，ioutil >= 0.99 – 平均 IO 队列长度，avgqu-sz >=1 – 平均 IO 传输量，iops(w/s) +ioth(wMB/s) <= 1

故障检查项	功能	说明
		说明 部分操作系统卡 IO 时无数据变化，此时计算 CPU IO 时间占用率，iowait > 0.8。
DiskSlow	检测节点磁盘是否存在慢 IO 故障 慢 IO 定义：系统对磁盘的 IO 请求下发后有响应，但平均响应时间超出阈值 典型场景：云硬盘由于网络波动导致慢 IO。	• 检查对象：所有数据盘 • 数据来源： /proc/diskstat 等效查询命令 `iostat -xmt 1` • 阈值： 平均 IO 时延，await >= 5000ms 说明 卡 IO 场景下该检查项失效，原因为 IO 请求未有响应，await 数据不会刷新。

另外 kubelet 组件内置如下检查项，但是存在不足，您可通过集群升级或安装 NPD 进行补足。

表15-10 Kubelet 内置检查项

故障检查项	功能	说明
PIDPressure	检查 Pid 是否充足	• 周期：10 秒 • 阈值：90% • 缺点：社区 1.23.1 及以前版本，该检查项在 pid 使用量大于 65535 时失效，详见 issue 107107。社区 1.24 及以前版本，该检查项未考虑 thread-max。
MemoryPressure	检查容器可分配空间（allocable）内存是否充足	• 周期：10 秒 • 阈值：最大值-100M • 最大值（Allocable）：节点总内存-节点预留内存 • 缺点：该检测项没有从节点整体内存维度检查内存耗尽情况，只关注了容器部分(Allocable)，在低。
DiskPressure	检查 kubelet 盘和 docker 盘的磁	周期：10 秒

故障检查项	功能	说明
	盘使用量及 inodes 使用量	阈值：90%

Node-problem-controller 故障隔离

说明

故障隔离仅 1.16.0 及以上版本支持。

Node-problem-controller（NPC）并不会随 NPD 插件默认安装。在安装 NPD 插件时，将参数`npc.enable`配置为 true 以部署双实例 NPC（注：也可配置单实例但不保证高可用）。

默认情况下，若多个节点发生故障，NPC 只会为 1 个节点添加污点，可通过参数 npc.maxTaintedNode 提高数量限制。故障恢复时，NPC 不在运行状态，污点会残留，需要手动清理或启动 NPC。

开源 NPD 插件提供了故障探测能力，但未提供基础故障隔离能力。对此，CCE 在开源 NPD 的基础上，增强了 Node-problem-controller（节点故障控制器组件），该组件参照 kubernetes 节点控制器实现，针对 NPD 探测上报的故障，自动为节点添加污点以进行基本的节点故障隔离。

可以按下表修改插件 **npc.customConditionToTaint** 参数，配置故障隔离规则。

表15-11 参数说明

参数	说明	默认值
npc.enable	是否启用 npc	false
npc.customCondtionToTaint	故障隔离规则列表	见下表
npc.customConditionToTaint[i]	故障隔离规则项	N/A
npc.customConditionToTaint[i]. condition.status	故障状态	true
npc.customConditionToTaint[i]. condition.type	故障类型	N/A
npc.customConditionToTaint[i]. enable	是否启用该故障隔离规则项	false
npc.customConditionToTaint[i]. .taint.effect	故障隔离效果 NoSchedule、PreferNoSchedule、NoExecute	NoSchedule 值域：NoSchedule、PreferNoSchedule、NoExecute
npc. maxTaintedNode	集群内多少节点允许被 npc	1

参数	说明	默认值
	添加污点 支持 int 格式和百分比格式	值域： • int 格式，数值范围为 1 到无穷大 • 百分比格式，数值范围为 1%到 100%，与集群节点数量乘积计算后最小值为 1。
Npc.affinity	Controller 的节点亲和性配置	N/A

表15-12 推荐故障隔离规则配置

故障	故障详情	污点
DiskReadonly	磁盘只读	NoSchedule，禁止新建 Pod
DiskProblem	磁盘空间不足，关键逻辑磁盘被卸载	NoSchedule，禁止新建 Pod
FrequentKubeletRestart	kubelet 频繁重启	NoSchedule，禁止新建 Pod
FrequentDockerRestart	Docker 频繁重启	NoSchedule，禁止新建 Pod
FrequentContainerdRestart	Containerd 频繁重启	NoSchedule，禁止新建 Pod
KUBEPROXYProblem	Kubeproxy 异常	NoSchedule，禁止新建 Pod
PIDProblem	Pid 不足	NoSchedule，禁止新建 Pod
FDProblem	FD 文件句柄数不足	NoSchedule，禁止新建 Pod
MemoryProblem	节点内存不足	NoSchedule，禁止新建 Pod

查看 NPD 事件

NPD 上报的事件可以在节点管理页面查询。

步骤 1 登录 CCE 控制台。

步骤 2 单击集群名称进入集群，在左侧选择“节点管理”。

步骤 3 在节点所在行，单击“事件”，可查看节点相关事件。

图15-2 查看节点事件

如下所示，当有事件上报可以查询。

----结束

AOM 告警配置

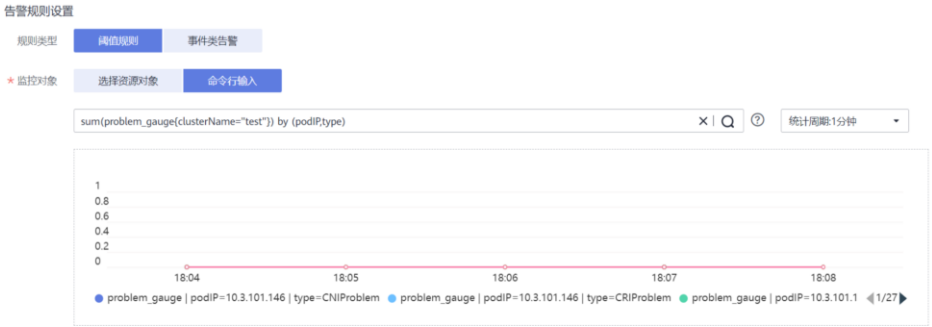
针对 NPD 状态类检查项，您可以通过配置 AOM（应用运维管理服务），以将异常状态转换为 AOM 告警，并通过短信、邮箱等方式通知到您。

步骤 1 登录 AOM 控制台。

步骤 2 在左侧导航栏选择“告警 > 告警规则”，在右上角单击“添加告警”。

步骤 3 设置告警规则。

- 规则类型：选择阈值类告警。
- 监控对象：选择命令行输入
- 命令行输入框：`sum(problem_gauge{clusterName="test"}) by (podIP,type)`



- 告警条件：选择触发条件在 1 个监控周期内，如果平均值>=1 达到连续 1 次时，产生重要告警。

- 告警通知（可选）：若需要将告警通过邮件、手机方式通知您，可在告警通知处，为此告警规则配置行动规则。若此处无行动规则，请新建告警行动规则。

----结束

Prometheus 指标采集

NPD 守护进程 POD 通过端口 19901 暴露 Prometheus metrics 指标，NPD Pod 默认被注释 metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"prometheus","path":"/metrics","port":"19901","names":""}]'。您可以自建 Prometheus 采集器识别并通过 http://{{NpdPodIP}}:{{NpdPodPort}}/metrics 路径获取 NPD 指标。

说明

NPD 插件为 1.16.5 版本以下时，Prometheus 指标的暴露端口为 20257。

目前指标信息包含异常状态计数 problem_counter 与异常状态 problem_gauge，如下所示

```
# HELP problem_counter Number of times a specific type of problem have occurred.
# TYPE problem_counter counter
problem_counter{reason="DockerHung"} 0
problem_counter{reason="DockerStart"} 0
problem_counter{reason="EmptyDirVolumeGroupStatusError"} 0
...
# HELP problem_gauge Whether a specific type of problem is affecting the node or
not.
# TYPE problem_gauge gauge
problem_gauge{reason="CNIIsDown",type="CNIProblem"} 0
problem_gauge{reason="CNIIsUp",type="CNIProblem"} 0
problem_gauge{reason="CRIIsDown",type="CRIProblem"} 0
problem_gauge{reason="CRIIsUp",type="CRIProblem"} 0
..
```

版本记录

表15-13 CCE 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本（仅 1.17 及以上版本集群支持）
1.16.4	/v1.(17|19|21|23).*/	• 新增 beta 检查项	0.8.10

插件版本	支持的集群版本	更新特性	社区版本（仅 1.17 及以上版本集群支持）
		ScheduledEvent，支持通过 metadata 接口检测宿主机异常导致虚拟机进行冷热迁移事件。该检查项默认不开启。	
1.16.3	/v1.(17|19|21|23).*/	• 新增 ResolvConf 配置文件检查。	0.8.10
1.16.1	/v1.(17|19|21|23).*/	• 新增 node-problem-controller。支持基本故障隔离能力。 • 新增 PID、FD、磁盘、内存、临时卷存储池、持久卷存储池检查项。	0.8.10
1.15.0	/v1.(17|19|21|23).*/	• 检测项全面加固，避免误报。 • 支持内核巡检。支持 OOMKilling 事件，TaskHung 事件上报。	0.8.10
1.14.11	/v1.(17|19|21).*/	• 适配 CCE 1.21 集群	0.7.1
1.14.5	/v1.(17|19).*/	• 修复监控指标无法被获取的问题	0.7.1
1.14.4	/v1.(17|19).*/	• 适配 ARM64 节点部署 • 适配 containerd 运行时节点	0.7.1
1.14.2	/v1.(17|19).*/	• 适配 Kubernetes 1.19 集群，新增支持 Ubuntu 操作系统和安全容器场景	0.7.1
1.13.8	/v1.15.11|v1.17.*/	• 修复容器隧道网络下 CNI 健康检查问题 • 调整资源配额	0.7.1
1.13.6	/v1.15.11|v1.17.*/	• 修复僵尸进程未被回收的问题	0.7.1
1.13.5	/v1.15.11|v1.17.*/	• 增加污点容忍配置	0.7.1
1.13.2	/v1.15.11|v1.17.*/	• 增加资源限制，增强 cni 插件的检测能力	0.7.1

15.6 dashboard

插件简介

Kubernetes Dashboard 是一个旨在为 Kubernetes 世界带来通用监控和操作 Web 界面的项目，集合了命令行可以操作的所有命令。

使用 Kubernetes Dashboard，您可以：

- 向 Kubernetes 集群部署容器化应用
- 诊断容器化应用的问题
- 管理集群的资源
- 查看集群上所运行的应用程序
- 创建、修改 Kubernetes 上的资源（例如 Deployment、Job、DaemonSet 等）
- 展示集群上发生的错误

例如：您可以伸缩一个 Deployment、执行滚动更新、重启一个 Pod 或部署一个新的应用程序。

开源社区地址：https://github.com/kubernetes/dashboard

安装步骤

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **dashboard**，单击“安装”。

步骤 2 在规格配置页面，配置以下参数。

- 证书配置：dashboard 服务端使用的证书。
  - 使用自定义证书。
    - 证书文件：单击查看证书文件样例参考。
    - 证书私钥：单击查看证书私钥样例参考。
  - 使用默认证书。

须知

dashboard 默认生成的证书不合法，将影响浏览器正常访问，建议您选择手动上传合法证书，以便通过浏览器校验，保证连接的安全性。

步骤 3 单击“安装”。

----结束

访问 dashboard

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，确认 dashboard 插件状态为“运行中”后，单击“访问”。

步骤 2 在 CCE 控制台弹出的窗口中复制 token。

步骤 3 在登录页面中选择“令牌”的登录方式，粘贴输入复制的 token，单击“登录”按钮。

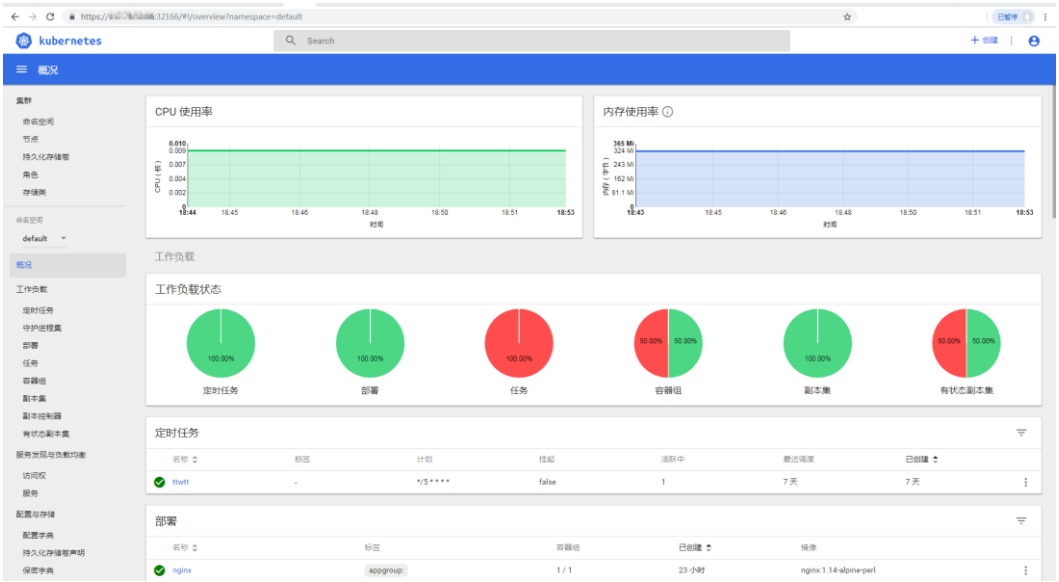
说明

本插件默认不支持使用证书认证的 kubeconfig 进行登录，推荐使用令牌方式登录。详细信息请参考：https://github.com/kubernetes/dashboard/issues/2474#issuecomment-348912376

图15-3 令牌方式登录

步骤 4 登录后效果，如下图。

图15-4 Dashboard 概览页



----结束

权限修改

安装 Dashboard 插件后初始角色仅拥有对大部分资源的只读权限，若想让 Dashboard 界面支持更多操作，需自行在后台对 RBAC 相关资源进行修改。

具体修改方式：

可对名为“kubernetes-dashboard-minimal”这个 ClusterRole 中的规则进行调整。

关于使用 RBAC 的具体细节可参看文档：https://kubernetes.io/docs/reference/access-authn-authz/rbac/。

附：访问报错解决方法

使用 Chrome 浏览器访问时，会出现如下“ERR_CERT_INVALID”的报错导致无法正常进入登录界面，原因是 dashboard 默认生成的证书未通过 Chrome 校验，当前有以下两种解决方式：

图15-5 Chrome 浏览器报错信息

- 方式一：使用火狐浏览器访问链接，为当前地址添加“例外”后即可进入登录页面。
- 方式二：通过启动 Chrome 时添加“--ignore-certificate-errors”开关忽略证书报错。

  Windows：保存链接地址，关闭所有已经打开的 Chrome 浏览器窗口，Windows 键+“R”弹出“运行”对话框，输入“chrome --ignore-certificate-errors”启动新的 chrome 窗口，输入地址进入登录界面。

版本记录

表15-14 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
2.1.1	/v1.(19|21|23).*/	2.5.0
2.0.10	/v1.(15|17|19|21).*/	2.0.0
2.0.4	/v1.(15|17|19).*/	2.0.0
2.0.3	/v1.(15|17|19).*/	2.0.0
2.0.2	/v1.(17|19).*/	2.0.0
2.0.1	/v1.(15|17).*/	2.0.0
2.0.0	/v1.(17).*/	2.0.0

15.7 autoscaler

插件简介

autoscaler 是 Kubernetes 中非常重要的一个 Controller，它提供了微服务的弹性能力，并且和 Serverless 密切相关。

弹性伸缩是很好理解的一个概念，当微服务负载高（CPU/内存使用率过高）时水平扩容，增加 pod 的数量以降低负载，当负载降低时减少 pod 的数量，减少资源的消耗，通过这种方式使得微服务始终稳定在一个理想的状态。

云容器引擎简化了 Kubernetes 集群的创建、升级和手动扩缩容，而集群中应用的负载本身是会随着时间动态变化的，为了更好的平衡资源使用率以及性能，Kubernetes 引入了 autoscaler 插件，它可以根据部署的应用所请求的资源量自动的动态的伸缩集群，详情请了解创建节点伸缩策略。

开源社区地址：https://github.com/kubernetes/autoscaler

插件说明

autoscaler 可分成扩容和缩容两个方面：

- **自动扩容**

  集群的自动扩容有以下两种方式实现：

  - 当集群中的 Pod 由于工作节点资源不足而无法调度时，会触发集群扩容，扩容节点与所在节点池资源配额一致。

    此时需要满足以下条件时才会执行自动扩容：

    - 节点上的资源不足。

- Pod 的调度配置中不能包含节点亲和的策略（即 Pod 若已经设置亲和某个节点，则不会自动扩容节点），节点亲和策略设置方法请参见调度策略（亲和与反亲和）。

- 当集群满足节点伸缩策略时，也会触发集群扩容。

说明

当前该插件使用的是最小浪费策略，即若 Pod 创建需要 3 核，此时有 4 核、8 核两种规格，优先创建规格为 4 核的节点。

- **自动缩容**

  当集群节点处于一段时间空闲状态时（默认 10min），会触发集群缩容操作（即节点会被自动删除）。当节点存在以下几种状态的 Pod 时，不可缩容：

  - Pod 有设置 PodDisruptionBudget，当移除 Pod 不满足对应条件时，节点不会缩容。
  - Pod 由于一些限制，如亲和、反亲和等，无法调度到其他节点，节点不会缩容。
  - Pod 拥有 cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'这个 annotations 时，节点不缩容。
  - 节点上存在 kube-system 命名空间下的 Pod（除 kube-system daemonset 创建的 Pod），节点不缩容。
  - 节点上如果有非 controller（deployment/replicaset/job/statefulset）创建的 Pod，节点不缩容。

约束与限制

- 集群为 1.9.7-r1 及以上版本时，才支持此功能。
- 安装时请确保有足够的资源安装本插件。
- 该插件功能仅支持通过**按需计费**方式购买的**虚拟机节点**，不支持“包年/包月”方式购买的节点和物理机节点。
- 默认节点池不支持弹性扩缩容。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“ 插件管理”，在右侧找到 **autoscaler**，单击“安装”。

步骤 2 配置插件安装参数。

表15-15 规格配置

参数	参数说明
插件规格	插件部署可选择如下几种规格。 **说明** autoscaler 插件高可用或自定义部署时，插件实例间存在反亲和策略，会分别部署在不同的节点上，因此集群中的可用节点必须大于等于插件实例数才可保证插件高可用。 • 单实例：以单实例部署插件。

参数	参数说明
	• 高可用 50：50 节点集群规模多实例部署，实例数为 2，具有高可用能力。 • 高可用 200：200 节点集群规模多实例部署，实例数为 2，具有高可用能力，每个实例使用资源比高可用 50 的实例更多。 • 自定义：根据需要自定义实例数量和实例规格。

表15-16 参数配置

参数	说明
扩缩容配置	可以勾选自动扩缩容。 • **当集群下负载实例无法调度时自动扩容（从节点池）** 即当出现 Pod 处于 Pending 状态无法调度时，集群会自动扩容节点。若 Pod 已经设置亲和某个节点，则不会自动扩容节点。该功能一般与 HPA 策略配合使用。 如不勾选，则只能通过节点伸缩策略进行扩缩容。 • **自动缩容节点** – 空置时间（min）：当集群节点处于一段时间的空闲状态时，会触发集群缩容操作，删除节点，默认 10min。 – 缩容阈值：当集群节点资源（Request 值）低于多少百分比时，进行集群缩容扫描（默认 0.5，即 50%，cpu 和 mem 都要满足的条件下才会缩容）。 – 冷却时间： 扩容执行后多久能再次判断是否缩容，默认 10min。 **说明** 集群中如果同时存在自动扩容和自动缩容的场景，建议配置“扩容执行后多久能再次判断是否缩容”为 0min，避免由于部分节点池持续扩容或者扩容失败重试而阻塞整体缩容节点行为，导致非预期的节点资源浪费。 节点删除后多久能再次判断是否缩容：删除节点后能再次启动缩容评估的时间间隔，默认 10min。 缩容失败后多久能再次判断是否缩容：缩容失败后能再次启动缩容评估的时间间隔，默认 3min。节点池中配置的缩容冷却时间和此处配置的缩容冷却时间之间的影响和关系请参见缩容冷却时间说明。 – 缩容并发数：最多支持多少个空闲节点同时缩容，默认 10。 缩容并发数只针对完全空闲节点，完全空闲节点可实现并发缩容。非完全空闲节点则只能逐个缩容。 **说明** 节点在缩容的时候，若节点上的 Pod 不需要驱逐（DaemonSet 的

参数	说明
	Pod 认为不需要驱逐），则认为该节点为完全空闲节点，否则认为该节点为非完全空闲。 – 检查间隔：节点被判定不可移除后能再次启动检查的时间间隔，默认 5min。
节点总数	集群可管理的节点数目的最大值，扩容时不会让集群下节点数超过此值。
CPU 总数（核）	集群中所有节点 CPU 核数之和的最大值，扩容时不会让集群下节点 CPU 核数之和超过此值。
内存总数（GB）	集群中所有节点内存之和的最大值，扩容时不会让集群下节点内存之和超过此值。

步骤 3 配置完成后，单击“安装”。

----结束

缩容冷却时间说明

节点池中配置的缩容冷却时间和 autoscaler 插件中配置的缩容冷却时间之间的影响和关系如下：

节点池配置的缩容冷却时间

弹性缩容冷却时间：当前节点池扩容出的节点多长时间不能被缩容，作用范围为节点池级别。

autoscaler 插件配置的缩容冷却时间

扩容后缩容冷却时间：autoscaler 触发扩容后（不可调度、指标、周期策略）整个集群多长时间内不能被缩容，作用范围为集群级别。

节点删除后缩容冷却时间：autoscaler 触发缩容后整个集群多长时间内不能继续缩容，作用范围为集群级别。

缩容失败后缩容冷却时间：autoscaler 触发缩容失败后整个集群多长时间内不能继续缩容，作用范围为集群级别。

版本记录

表15-17 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.23.9	/v1.23.*/	1.23.0
1.23.8	/v1.23.*/	1.23.0

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.23.7	/v1.23.*/	1.23.0
1.23.3	/v1.23.*/	1.23.0
1.21.8	/v1.21.*/	1.21.0
1.21.6	/v1.21.*/	1.21.0
1.21.4	/v1.21.*/	1.21.0
1.21.2	/v1.21.*/	1.21.0
1.21.1	/v1.21.*/	1.21.0
1.19.13	/v1.19.*/	1.21.0
1.19.12	/v1.19.*/	1.19.0
1.19.11	/v1.19.*/	1.19.0
1.19.9	/v1.19.*/	1.19.0
1.19.8	/v1.19.*/	1.19.0
1.19.7	/v1.19.*/	1.19.0
1.19.6	/v1.19.*/	1.19.0
1.19.3	/v1.19.*/	1.19.0
1.17.21	/v1.17.*/	1.19.0
1.17.19	/v1.17.*/	1.19.0
1.17.17	/v1.17.*/	1.17.0
1.17.16	/v1.17.*/	1.17.0
1.17.15	/v1.17.*/	1.17.0
1.17.14	/v1.17.*/	1.17.0
1.17.8	/v1.17.*/	1.17.0
1.17.7	/v1.17.*/	1.17.0
1.17.5	/v1.17.*/	1.17.0
1.17.2	/v1.17.*/	1.17.0

15.8 nginx-ingress

插件简介

Kubernetes 通过 kube-proxy 服务实现了 Service 的对外发布及负载均衡，它的各种方式都是基于传输层实现的。在实际的互联网应用场景中，不仅要实现单纯的转发，还有更加细致的策略需求，如果使用真正的负载均衡器更会增加操作的灵活性和转发性能。

基于以上需求，Kubernetes 引入了资源对象 Ingress，Ingress 为 Service 提供了可直接被集群外部访问的虚拟主机、负载均衡、SSL 代理、HTTP 路由等应用层转发功能。

Kubernetes 官方发布了基于 Nginx 的 Ingress 控制器，nginx-ingress 是一款使用 configmap 来存储 nginx 配置的插件，nginx ingress controller 会将 ingress 生成一段 nginx 的配置，将这个配置通过 Kubernetes API 写到 Nginx 的 Pod 中，然后 reload 完成 nginx 的配置修改和更新。

nginx-ingress 插件直接使用社区模板与镜像，CCE 不提供额外维护，不建议用于商用场景。

开源社区地址：https://github.com/kubernetes/ingress-nginx

说明

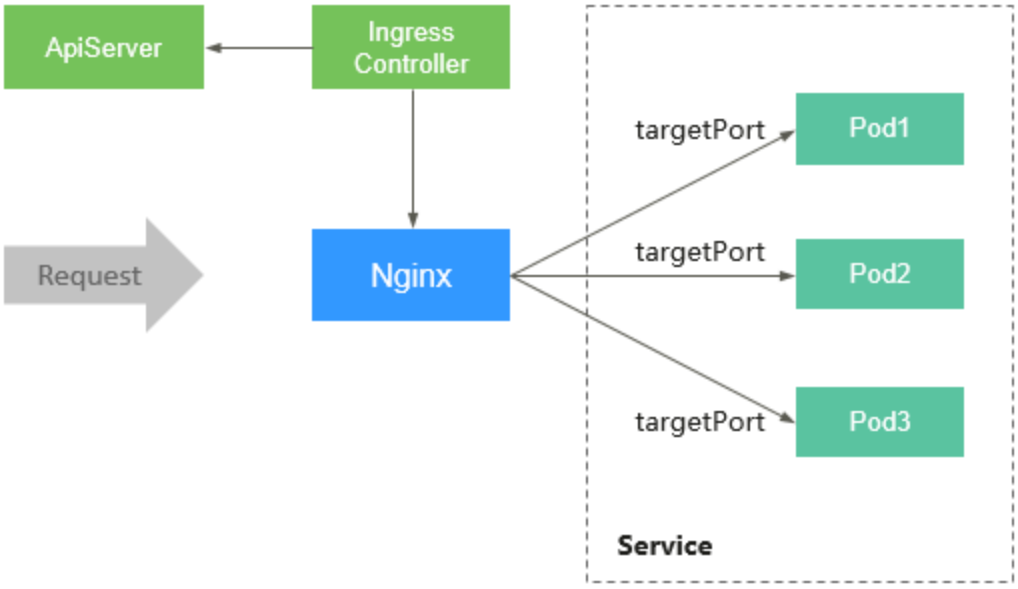
- 安装该插件时，您可以通过“定义 nginx 配置”添加配置，此处的设置将会全局生效，该参数直接通过配置 nginx.conf 生成，将影响管理的全部 Ingress，相关参数可通过 configmap 查找，如果您配置的参数不包含在 configmap 所列出的选项中将不会生效。
- 安装该插件后，您在 CCE 控制台创建 Ingress 时可以开启“对接 Nginx”按钮，并通过“Nginx 配置”下的“自定义配置”将 Kubernetes annotations 添加到特定的 Ingress 对象，以自定义其行为，Kubernetes annotations 字段详情请参见 Annotations。
- 请勿手动修改和删除 CCE 自动创建的 ELB 和监听器，否则将出现工作负载异常；若您已经误修改或删除，请卸载 Nginx Ingress 插件后重装。

工作原理

Nginx Ingress 由资源对象 Ingress、Ingress 控制器、Nginx 三部分组成，Ingress 控制器用以将 Ingress 资源实例组装成 Nginx 配置文件（nginx.conf），并重新加载 Nginx 使变更的配置生效。当它监听到 Service 中 Pod 变化时通过动态变更的方式实现 Nginx 上游服务器组配置的变更，无须重新加载 Nginx 进程。工作原理如下图所示。

- Ingress：一组基于域名或 URL 把请求转发到指定 Service 实例的访问规则，是 Kubernetes 的一种资源对象，Ingress 实例被存储在对象存储服务 etcd 中，通过接口服务被实现增、删、改、查的操作。
- Ingress 控制器（Ingress controller）：用以实时监控资源对象 Ingress、Service、Endpoint、Secret（主要是 TLS 证书和 Key）、Node、ConfigMap 的变化，自动对 Nginx 进行相应的操作。
- Nginx：实现具体的应用层负载均衡及访问控制。

图15-6 Nginx Ingress 工作原理



使用约束

- 仅支持在 1.15 及以上版本的 CCE 集群中安装此插件。
- 通过 api 调接口创建的 Ingress annotation 必须添加 kubernetes.io/ingress.class: "nginx"，如果是对接老的 Ingress，annotation 需添加为 kubernetes.io/ingress.class: "cce"。
- 独享型 ELB 规格必须支持网络型（TCP/UDP），且网络类型必须支持私网（有私有 IP 地址）。

前提条件

在创建容器工作负载前，您需要存在一个可用集群。若没有可用集群 ，请参照“购买 CCE 集群”中的步骤创建。

安装插件

新 UI 支持使用独享型负载均衡实例。

步骤 1 登录 CCE 控制台。

步骤 2 在左侧选择“插件市场”，找到“nginx-ingress”插件，单击“安装”。

步骤 3 填写配置参数。

- **选择集群**：选择要安装的集群
- **规格配置**：请根据业务需求选择插件规格，可自定义资源规格部署。
- **负载均衡器**：支持对接共享型或独享型负载均衡实例，如果可用实例，请先创建。负载均衡器需要拥有至少两个监听器配额，且端口 80 和 443 没有被监听器占用。

- **nginx 配置参数**：配置 nginx.conf 文件，将影响管理的全部 Ingress，相关参数可通过 configmap 查找，如果您配置的参数不包含在 configmap 所列出的选项中将不会生效。
- **默认 404 服务**：默认使用插件自带的 404 服务。支持自定义 404 服务，填写“命名空间/服务名称”，如果服务不存在，插件会安装失败。

步骤 4 单击“安装”。

----结束

版本记录

表15-18 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
2.1.1	/v1.(19|21|23).*/	1.2.1
2.1.0	/v1.(19|21|23).*/	1.2.0
2.0.1	/v1.(19|21|23).*/	1.1.1
1.3.2	/v1.(15|17|19|21).*/	0.49.3
1.2.6	/v1.(15|17|19).*/	0.46.0
1.2.5	/v1.(15|17|19).*/	0.46.0
1.2.3	/v1.(15|17|19).*/	0.43.0
1.2.2	/v1.(15|17).*/	0.43.0

15.9 metrics-server

从 Kubernetes 1.8 开始，Kubernetes 通过 Metrics API 提供资源使用指标，例如容器 CPU 和内存使用率。这些度量可以由用户直接访问（例如，通过使用 kubectl top 命令），或者由集群中的控制器（例如，Horizontal Pod Autoscaler）使用来进行决策，具体的组件为 Metrics-Server，用来替换之前的 heapster，heapster 从 1.11 开始逐渐被废弃。

Metrics Server 是集群核心资源监控数据的聚合器，您可以在 CCE 控制台快速安装本插件。

安装本插件后，可在“弹性伸缩”页面的“工作负载伸缩”页签下，创建 HPA 策略，具体请参见创建工作负载弹性伸缩（HPA）。

社区官方项目及文档：https://github.com/kubernetes-sigs/metrics-server。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **metrics-server**，单击“安装”。

步骤 2 该插件可配置“单实例”或“高可用”规格，选择后单击“安装”。

----结束

版本记录

表15-19 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.2.1	/v1.(19|21|23).*/	0.4.4
1.1.10	/v1.(15|17|19|21).*/	0.4.4
1.1.4	/v1.(15|17|19).*/	0.4.4
1.1.2	/v1.(15|17|19).*/	0.4.4
1.1.1	/v1.(13|15|17|19).*/	0.3.7
1.1.0	/v1.(13|15|17|19).*/	0.3.7
1.0.5	/v1.13.*|v1.15.*|v1.17.*/	0.3.7

15.10 cce-hpa-controller

cce-hpa-controller 插件是一款 CCE 自研的插件，能够基于 CPU 利用率、内存利用率等指标，对无状态工作负载进行弹性扩缩容。

安装本插件后，可在“弹性伸缩”页面的“工作负载伸缩”页签下，创建 CustomedHPA 策略，具体请参见创建工作负载弹性伸缩（CustomedHPA）。

主要功能

- 支持按照当前实例数的百分比进行扩缩容。
- 支持设置一次扩缩容的最小步长。
- 支持按照实际指标值执行不同的扩缩容动作。

约束与限制

- 仅支持在 v1.15 及以上版本的 CCE 集群中安装本插件。
- 若 cce-hpa-controller 版本低于 1.2.11，则必须安装 prometheus 插件，若版本大于或等于 1.2.11，则需要安装能够提供 Metrics API 的插件，如 metrics-server 和

Prometheus。若使用 Prometheus，需要将 Prometheus 注册为 Metrics API 的服务，详见提供资源指标。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **cce-hpa-controller**，单击“安装”。

步骤 2 该插件可配置“单实例”或“自定义”规格，选择后单击“安装”。

说明

单实例仅用于验证场景，商用场景请根据集群规格使用"自定义"资源配置，cce-hpa-controller 插件的规格大小主要受集群中总容器数量和伸缩策略数量影响，通常场景下建议每 5000 容器配置 CPU 500m，内存 1000Mi 资源，每 1000 伸缩策略 CPU 100m，内存 500Mi。

----结束

版本记录

表15-20 CCE 插件版本记录

插件版本	支持的集群版本
1.3.1	/v1.(19|21|23).*/
1.2.12	/v1.(15|17|19|21).*/
1.2.11	/v1.(15|17|19|21).*/
1.2.10	/v1.(15|17|19|21).*/
1.2.4	/v1.(15|17|19).*/
1.2.3	/v1.(15|17|19).*/
1.2.2	/v1.(15|17|19).*/
1.2.1	/v1.(15|17|19).*/
1.1.3	/v1.(15|17).*/

15.11 prometheus

插件简介

Prometheus 是一套开源的系统监控报警框架。它启发于 Google 的 borgmon 监控系统，由工作在 SoundCloud 的 Google 前员工在 2012 年创建，作为社区开源项目进行开发，并于 2015 年正式发布。2016 年，Prometheus 正式加入 Cloud Native Computing Foundation，成为受欢迎度仅次于 Kubernetes 的项目。

在云容器引擎 CCE 中，支持以插件的方式快捷安装 Prometheus。

插件官网：https://prometheus.io/

开源社区地址：https://github.com/prometheus/prometheus

约束与限制

CCE 提供的 Prometheus 插件仅支持 1.21 及以下版本的集群。

插件特点

作为新一代的监控框架，Prometheus 具有以下特点：

- 强大的多维度数据模型：
  a. 时间序列数据通过 metric 名和键值对来区分。
  b. 所有的 metrics 都可以设置任意的多维标签。
  c. 数据模型更随意，不需要刻意设置为以点分隔的字符串。
  d. 可以对数据模型进行聚合，切割和切片操作。
  e. 支持双精度浮点类型，标签可以设为全 unicode。
- 灵活而强大的查询语句（PromQL）：在同一个查询语句，可以对多个 metrics 进行乘法、加法、连接、取分数位等操作。
- 易于管理：Prometheus server 是一个单独的二进制文件，可直接在本地工作，不依赖于分布式存储。
- 高效：平均每个采样点仅占 3.5 bytes，且一个 Prometheus server 可以处理数百万的 metrics。
- 使用 pull 模式采集时间序列数据，这样不仅有利于本机测试而且可以避免有问题的服务器推送坏的 metrics。
- 可以采用 push gateway 的方式把时间序列数据推送至 Prometheus server 端。
- 可以通过服务发现或者静态配置去获取监控的 targets。
- 有多种可视化图形界面。
- 易于伸缩。

需要指出的是，由于数据采集可能会有丢失，所以 Prometheus 不适用对采集数据要 100%准确的情形。但如果用于记录时间序列数据，Prometheus 具有很大的查询优势，此外，Prometheus 适用于微服务的体系架构。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **Prometheus**，单击“安装”。

步骤 2 在“规格配置”步骤中，配置以下参数：

表15-21 Prometheus 配置参数说明

参数	参数说明
插件规格	根据业务需求，选择插件的规格，包含如下选项：

参数	参数说明
	• 演示规格（100 容器以内）：适用于体验和功能演示环境，该规模下 prometheus 占用资源较少，但处理能力有限。建议在集群内容器数目不超过 100 时使用。 • 小规格（2000 容器以内）：建议在集群中的容器数目不超过 2000 时使用。 • 中规格（5000 容器以内）：建议在集群中的容器数目不超过 5000 时使用。 • 大规格（超过 5000 容器）：建议集群中容器数目超过 5000 时使用此规格。
实例数	选择上方插件规格后，显示插件中的实例数，此处仅作显示。
容器	选择插件规格后，显示插件容器的 CPU 和内存配额，此处仅作显示。
数据保留期	自定义监控数据需要保留的天数，默认为 15 天。
存储	支持云硬盘作为存储，按照界面提示配置如下参数： • 可用区：请根据业务需要进行选择。可用区是在同一区域下，电力、网络隔离的物理区域，可用区之间内网互通，不同可用区之间物理隔离。 • 子类型：支持普通 IO、高 IO 和超高 IO 三种类型。 • 容量：请根据业务需要输入存储容量，默认 10G。 **说明** 若命名空间 monitoring 下已存在 pvc，将使用此存储作为存储源。

步骤 3 单击“安装”。安装完成后，插件会在集群中部署以下实例。

- prometheus-operator：根据自定义资源（Custom Resource Definition / CRDs）来部署和管理 Prometheus Server，同时监控这些自定义资源事件的变化来做相应的处理，是整个系统的控制中心。
- prometheus（Server）：Operator 根据自定义资源 Prometheus 类型中定义的内容而部署的 Prometheus Server 集群，这些自定义资源可以看作是用来管理 Prometheus Server 集群的 StatefulSets 资源。
- prometheus-kube-state-metrics：将 Prometheus 的 metrics 数据格式转换成 K8s API 接口能识别的格式。
- custom-metrics-apiserver：将自定义指标聚合到原生的 kubernetes apiserver。
- prometheus-node-exporter：每个节点上均有部署，收集 Node 级别的监控数据。
- grafana：可视化浏览普罗监控数据。

----结束

提供资源指标

容器和节点的资源指标，如 CPU、内存使用量，可通过 Kubernetes 的 Metrics API 获得。这些指标可以直接被用户访问，比如用 kubectl top 命令，也可以被 HPA 或者 CustomedHPA 使用，根据资源使用率使负载弹性伸缩。

Prometheus 插件可为 Kubernetes 提供 Metrics API，但默认未开启，若要将其开启，需要创建以下 APIService 对象：

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  labels:
    app: custom-metrics-apiserver
    release: cceaddon-prometheus
  name: v1beta1.metrics.k8s.io
spec:
  group: metrics.k8s.io
  groupPriorityMinimum: 100
  insecureSkipTLSVerify: true
  service:
    name: custom-metrics-apiserver
    namespace: monitoring
    port: 443
  version: v1beta1
  versionPriority: 100
```

可以将该对象保存为文件，命名为 metrics-apiservice.yaml，然后执行以下命令：

```
kubectl create -f metrics-apiservice.yaml
```

执行 kubectl top 命令，若显示如下，则表示 Metrics API 是否能正常访问：

```
# kubectl top pod -n monitoring
NAME                                                  CPU(cores)   MEMORY(bytes)
cceaddon-prometheus-kube-state-metrics-7b77694f48-zc9pl   4m           16Mi
cceaddon-prometheus-node-exporter-4jvwv                   1m           16Mi
cceaddon-prometheus-node-exporter-85zl4                   2m           39Mi
cceaddon-prometheus-node-exporter-qbrmb                   0m           15Mi
cceaddon-prometheus-operator-659547567d-j6484             0m           48Mi
custom-metrics-apiserver-d4f556ff9-l2j2m                  38m          44Mi
grafana-78f9966c99-xprkx                                  0m           25Mi
prometheus-0                                              18m          706Mi
```

参考资源

- Prometheus 概念及详细配置请参阅 Prometheus 官方文档
- Node exporter 安装请参考 node_exporter github 仓库
- Slack 信息发送请参考 Incoming Webhooks

版本记录

表15-22 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
2.23.32	/v1.(17|19|21).*/	2.10.0
2.23.31	/v1.15.*/	2.10.0
2.23.30	/v1.(17|19|21).*/	2.10.0
2.21.14	/v1.(17|19|21).*/	2.10.0
2.21.12	/v1.15.*/	2.10.0
2.21.11	/v1.(17|19).*/	2.10.0
1.15.1	/v1.(15|17).*/	2.10.0

15.12 web-terminal

web-terminal 是一款非常轻巧的终端服务器，支持在 Web 界面上使用 Kubectl 命令。它支持通过标准的 Web 浏览器和 HTTP 协议提供远程 CLI，提供灵活的接口便于集成到独立系统中，可直接作为一个服务连接，通过 cmdb 获取信息并登录服务器。

web-terminal 可以在 Node.js 支持的所有操作系统上运行，而不依赖于本机模块，快速且易于安装，支持多会话。

开源社区地址：https://github.com/rabchev/web-terminal

约束与限制

- 仅支持在 1.21 及以下版本的集群中安装此插件，暂不支持 ARM 集群。
- web-terminal 中 kubectl 具有高级别操作权限，限账号以及具有 CCE Administrator 权限的 IAM 用户安装此插件，如需收缩插件中 kubectl 权限，请参见收缩 web-terminal 权限操作。
- 集群必须安装 CoreDNS 才能使用 web-terminal。

注意事项

web-terminal 插件能够对 CCE 集群进行管理，请用户妥善保管好登录密码，避免密码泄漏造成损失。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“ 插件管理”，在右侧找到 **web-terminal**，单击“安装”。

步骤 2 配置以下参数。

- 访问类型：固定为节点访问，该插件默认以 NodePort 形式提供访问，需为集群任意一个节点绑定弹性 IP 才能使用。若集群没有绑定弹性 IP，需绑定弹性 IP。
- 用户名：默认为 root，不可修改。
- 密码：登录 web-terminal 的密码，请务必记住该密码。web-terminal 插件能够对 CCE 集群进行管理，请用户妥善保管好登录密码，避免密码泄漏造成损失。
- 确认密码：重新准确输入该密码。

步骤 3 单击“安装”。

----结束

使用 web-terminal 插件连接集群

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”。

步骤 2 在右侧找到 web-terminal，单击“访问”。

----结束

收缩 web-terminal 权限

web-terminal 插件安装后，其 kubectl 默认使用 cluster-admin ClusterRole 权限，能够操作该集群 k8s 资源，若需手动配置为其他类型权限，可通过 kubectl edit clusterrolebinding web-terminal 修改 web-terminal ServiceAccount 绑定其他权限的 ClusterRole。

ClusterRole、ClusterRoleBinding 相关配置说明请参见命名空间权限（Kubernetes RBAC 授权）。

须知

- 手动配置的 web-terminal 权限在 web-terminal 插件升级时存在被重置风险，需注意做好备份在插件升级后重新配置。
- 使用 kubectl 修改 clusterrolebinding 配置需确保当前 kubectl 有修改 clusterrolebinding 资源操作权限。

版本记录

表15-23 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.1.12	/v1.(15|17|19|21).*/	0.6.6
1.1.6	/v1.(15|17|19).*/	0.6.6

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.1.5	/v1.(15|17|19).*/	0.6.6
1.1.3	/v1.(17|19).*/	0.6.6
1.0.6	/v1.(15|17).*/	0.6.6
1.0.5	/v1.(9|11|13|15|17).*/	0.6.6

15.13 gpu-beta

插件简介

gpu-beta 插件是支持在容器中使用 GPU 显卡的设备管理插件，集群中使用 GPU 节点时必须安装 gpu-beta 插件。

约束与限制

- 下载的驱动必须是后缀为“.run”的文件。
- 仅支持 Nvidia Tesla 驱动，不支持 GRID 驱动。
- 安装或重装插件时，需要保证驱动下载链接正确且可正常访问，插件对链接有效性不做额外校验。
- gpu-beta 插件仅提供驱动的下载及安装脚本执行功能，插件的状态仅代表插件本身功能正常，与驱动是否安装成功无关。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **gpu-beta**，单击“安装”。

步骤 2 配置驱动链接地址。

须知

- 如果下载链接为公网地址，如 nvidia 官网地址 https://us.download.nvidia.com/tesla/396.37/NVIDIA-Linux-x86_64-396.37.run，各 GPU 节点均需要绑定 EIP。获取驱动链接方法请参考获取驱动链接-公网地址。
- 若下载链接为 OBS 上的链接，无需绑定 EIP 。获取驱动链接方法请参考获取驱动链接-OBS 地址。
- 请确保 Nvidia 驱动版本与 GPU 节点适配。
- 更改驱动版本后，需要重启节点才能生效。

步骤 3 单击“安装”，安装 gpu-beta 插件的任务即可提交成功。

----结束

验证插件

插件安装完成后，在 GPU 节点及调度了 GPU 资源的容器中执行 nvidia-smi 命令，验证 GPU 设备及驱动的可用性。

GPU 节点

```
cd /opt/cloud/cce/nvidia/bin && ./nvidia-smi
```

容器

```
cd /usr/local/nvidia/bin && ./nvidia-smi
```

能正常返回 GPU 信息，说明设备可用，插件安装成功。

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.118.02   Driver Version: 440.118.02   CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
| N/A   31C    P0    23W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

获取驱动链接-公网地址

步骤 1 登录 CCE 控制台。

步骤 2 创建节点，在节点规格处选择要创建的 GPU 节点，选中后下方显示的信息中可以看到节点的 GPU 显卡型号。

图15-7 查看显卡型号

步骤 3 登录到 *https://www.nvidia.com/Download/Find.aspx?lang=cn* 网站。

步骤 4 如下图所示，在“NVIDIA 驱动程序下载”框内选择对应的驱动信息。其中“操作系统”必须选 **Linux 64-bit**。

图15-8 参数选择

步骤 5 驱动信息确认完毕，单击“搜索”按钮，会跳转到驱动信息展示页面，该页面会显示驱动的版本信息，单击“下载”到下载页面。

图15-9 驱动信息

步骤 6 获取驱动软件链接方式分两种：

- 方式一：如上图，在浏览器的链接中找到路径为 *url=/tesla/396.37/NVIDIA-Linux-x86_64-396.37.run* 的路径，补齐全路径 https://us.download.nvidia.com/tesla/396.37/NVIDIA-Linux-x86_64-396.37.run 该方式节点需要绑定 EIP 。
- 方式二：如下图，单击“下载”按钮下载驱动，然后上传到 OBS，获取软件的链接，该方式节点不需要绑定 EIP。

图15-10 获取链接

----结束

获取驱动链接-OBS 地址

步骤 1 将驱动上传到对象存储服务 OBS 中，并将驱动文件设置为公共读。

说明

节点重启时会重新下载驱动进行安装，请保证驱动的 OBS 桶链接长期有效。

步骤 2 在 OBS 管理控制台左侧导航栏选择“对象存储”。

步骤 3 在桶列表单击待操作的桶，进入“概览”页面。

步骤 4 在左侧导航栏，单击“对象”。

步骤 5 选中目标对象，在对象详情页复制驱动链接。

----结束

版本记录

表15-24 CCE 插件版本记录

插件版本	支持的集群版本
1.2.15	/v1.(15|17|19|21|23).*/
1.2.11	/v1.(15|17|19|21).*/
1.2.10	/v1.(15|17|19|21).*/
1.2.9	/v1.(15|17|19|21).*/
1.2.2	/v1.(15|17|19).*/
1.2.1	/v1.(15|17|19).*/
1.1.13	/v1.(13|15|17).*/
1.1.11	/v1.(15|17).*/

15.14 volcano

插件简介

Volcano 是一个基于 Kubernetes 的批处理平台，提供了机器学习、深度学习、生物信息学、基因组学及其他大数据应用所需要而 Kubernetes 当前缺失的一系列特性。

Volcano 提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力，通过接入 AI、大数据、基因、渲染等诸多行业计算框架服务终端用户。(目前 Volcano 项目已经在 Github 开源)

Volcano 针对计算型应用提供了作业调度、作业管理、队列管理等多项功能，主要特性包括：

- 丰富的计算框架支持：通过 CRD 提供了批量计算任务的通用 API，通过提供丰富的插件及作业生命周期高级管理，支持 TensorFlow，MPI，Spark 等计算框架容器化运行在 Kubernetes 上。
- 高级调度：面向批量计算、高性能计算场景提供丰富的高级调度能力，包括成组调度，优先级抢占、装箱、资源预留、任务拓扑关系等。
- 队列管理：支持分队列调度，提供队列优先级、多级队列等复杂任务调度能力。

项目开源地址：https://github.com/volcano-sh/volcano

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **Volcano**，单击“安装”。

步骤 2 该插件可配置“单实例”、“高可用”或自定义规格。

选择自定义时，volcano-controller 和 volcano-scheduler 的建议值如下：

- 小于 100 个节点，可使用默认配置，即 CPU 的申请值为 500m，限制值为 2000m；内存的申请值为 500Mi，限制值为 2000Mi。
- 高于 100 个节点，每增加 100 个节点（10000 个 Pod），建议 CPU 的申请值增加 500m，内存的申请值增加 1000Mi；CPU 的限制值建议比申请值多 1500m，内存的限制值建议比申请值多 1000Mi。

表15-25 volcano-controller 和 volcano-scheduler 的建议值

节点/Pods 规模	CPU Request(m)	CPU Limit(m)	Memory Request(Mi)	Memory Limit(Mi)
50/5k	500	2000	500	2000
100/1w	1000	2500	1500	2500
200/2w	1500	3000	2500	3500
300/3w	2000	3500	3500	4500
400/4w	2500	4000	4500	5500

步骤 3 配置 volcano 默认调度器配置参数。

```
ca cert: ''
default_scheduler_conf:
  actions: 'allocate, backfill'
  tiers:
    - plugins:
        - name: 'priority'
        - name: 'gang'
        - name: 'conformance'
    - plugins:
        - name: 'drf'
        - name: 'predicates'
        - name: 'nodeorder'
    - plugins:
        - name: 'cce-gpu-topology-predicate'
        - name: 'cce-gpu-topology-priority'
        - name: 'cce-gpu'
    - plugins:
        - name: 'nodelocalvolume'
        - name: 'nodeemptydirvolume'
        - name: 'nodeCSIscheduling'
        - name: 'networkresource'
server_cert: ''
server_key: ''
```

表15-26 Volcano 插件配置参数说明

插件	功能	参数说明	用法演示
binpack	将 pod 调度到资	• binpack.weight：	`- plugins:`

插件	功能	参数说明	用法演示
	源使用较高的节点以减少资源碎片	binpack 插件本身在所有插件打分中的权重 • binpack.cpu：cpu 资源在资源比重的比例，默认是 1 • binpack.memory：memory 资源在所有资源中的比例，默认是 1 • binpack.resources：	- name: binpack arguments: binpack.weight: 10 binpack.cpu: 1 binpack.memory: 1 binpack.resources: nvidia.com/gpu, example.com/foo binpack.resources.nvidia.com/gpu: 2 binpack.resources.example.com/foo: 3
conformance	跳过关键 Pod，比如在 kube-system 命名空间的 Pod，防止这些 Pod 被驱逐	-	-
gang	将一组 pod 看做一个整体去分配资源	-	-
priority	使用用户自定义负载的优先级进行调度	-	-
overcommit	将集群的资源放到一定倍数后调度，提高负载入队效率。负载都是 deployment 的时候，建议去掉此插件或者设置扩大因子为 2.0。	overcommit-factor: 扩大因子，默认是 1.2	- plugins: - name: overcommit arguments: overcommit-factor: 2.0
drf	根据作业使用的主导资源份额进行调度，用的越少的优先	-	-
predicates	预选节点的常用算法，包括节点亲和，pod 亲和，污点容忍，node ports 重复，volume limits，volume zone 匹配	-	-

插件	功能	参数说明	用法演示
	等一系列基础算法		

nodeorder	优选节点的常用算法	• nodeaffinity.weight：节点亲和性优先调度，默认值是 1
		• podaffinity.weight：pod 亲和性优先调度，默认值是 1
		• leastrequested.weight：资源分配最少的的节点优先，默认值是 1
		• balancedresource.weight：node 上面的不同资源分配平衡的优先，默认值是 1
		• mostrequested.weight：资源分配最多的的节点优先，默认值是 0
		• tainttoleration.weight：污点容忍高的优先调度，默认值是 1
		• imagelocality.weight：node 上面有 pod 需要镜像的优先调度，默认值是 1
		• selectorspread.weight: 把 pod 均匀调度到不同的节点上，默认值是 0
		• volumebinding.weight: local pv 延迟绑定调度，默认值是 1
		• podtopologyspread.weight: pod 拓扑调度，默认值是 2
			`- plugins:`
			`  - name: nodeorder`
			`    arguments:`
			
			`leastrequested.weight: 1`
			
			`mostrequested.weight: 0`
			
			`nodeaffinity.weight: 1`
			
			`podaffinity.weight: 1`
			
			`balancedresource.weight: 1`
			
			`tainttoleration.weight: 1`
			
			`imagelocality.weight: 1`
			
			`volumebinding.weight: 1`
			
			`podtopologyspread.weight: 2`

cce-gpu-topology-predicate	GPU 拓扑调度预选算法	-	-
cce-gpu-topology-priority	GPU 拓扑调度优选算法	-	-
cce-gpu	结合 CCE 的 GPU 插件支持	-	-

插件	功能	参数说明	用法演示
	GPU 资源分配，支持小数 GPU 配置		
numaaware	numa 拓扑调度	weight: 插件的权重	
networkresource	支持预选过滤 ENI 需求节点，参数由 CCE 传递，不需要手动配置	NetworkType: 网络类型（eni 或者 vpc-router 类型）	-
nodelocalvolume	支持预选过滤不符合 local volume 需求节点	-	-
nodeemptydirvolume	支持预选过滤不符合 emptydir 需求节点	-	-
nodeCSIscheduling	支持预选过滤 everest 组件异常节点	-	-

步骤 4 单击“安装”。

----结束

在控制台中修改 volcano-scheduler 配置

Volcano 允许用户在安装，升级，编辑时，编写 Volcano 调度器配置信息，并将配置内容同步到 volcano-scheduler-configmap 里。

本节介绍如何使用自定义配置，以便用户让 volcano-scheduler 能更适合自己的场景。

说明

仅 Volcano 1.7.1 及以上版本支持该功能。在新版插件界面上合并了原 plugins.eas_service 和 resource_exporter_enable 等选项，以新选项 default_scheduler_conf 代替。

您可登录 CCE 控制台，单击集群名称进入集群，单击左侧导航栏的“插件管理”，在右侧找到 **Volcano**，单击“安装”或“升级”，并在“参数配置”中设置 Volcano 调度器配置参数。

- 使用 resource_exporter 配置，示例如下：

```
{
    "ca_cert": "",
    "default_scheduler_conf": {
        "actions": "allocate, backfill",
        "tiers": [
```

```
        {
            "plugins": [
                {
                    "name": "priority"
                },
                {
                    "name": "gang"
                },
                {
                    "name": "conformance"
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "drf"
                },
                {
                    "name": "predicates"
                },
                {
                    "name": "nodeorder"
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "cce-gpu-topology-predicate"
                },
                {
                    "name": "cce-gpu-topology-priority"
                },
                {
                    "name": "cce-gpu"
                },
                {
                    "name": "numa-aware" # add this also enable
resource_exporter
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "nodelocalvolume"
                },
                {
                    "name": "nodeemptydirvolume"
                },
                {
                    "name": "nodeCSIscheduling"
                },
                {
```

```
                    "name": "networkresource"
                }
            ]
        }
    ]
},
"server_cert": "",
"server_key": ""
}
```

开启后可以同时使用 volcano-scheduler 的 numa-aware 插件功能和 resource_exporter 功能。

- 使用 eas_service 配置，示例如下：

```
{
    "ca cert": "",
    "default_scheduler_conf": {
        "actions": "allocate, backfill",
        "tiers": [
            {
                "plugins": [
                    {
                        "name": "priority"
                    },
                    {
                        "name": "gang"
                    },
                    {
                        "name": "conformance"
                    }
                ]
            },
            {
                "plugins": [
                    {
                        "name": "drf"
                    },
                    {
                        "name": "predicates"
                    },
                    {
                        "name": "nodeorder"
                    }
                ]
            },
            {
                "plugins": [
                    {
                        "name": "cce-gpu-topology-predicate"
                    },
                    {
                        "name": "cce-gpu-topology-priority"
                    },
                    {
                        "name": "cce-gpu"
                    },
```

```
                {
                    "name": "eas",
                    "custom": {
                        "availability_zone_id": "",
                        "driver_id": "",
                        "endpoint": "",
                        "flavor_id": "",
                        "network_type": "",
                        "network_virtual_subnet_id": "",
                        "pool_id": "",
                        "project_id": "",
                        "secret_name": "eas-service-secret"
                    }
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "nodelocalvolume"
                },
                {
                    "name": "nodeemptydirvolume"
                },
                {
                    "name": "nodeCSIscheduling"
                },
                {
                    "name": "networkresource"
                }
            ]
        }
    ]
  },
  "server_cert": "",
  "server_key": ""
}
```

- 使用 ief 配置，示例如下：

```
{
  "ca_cert": "",
  "default_scheduler_conf": {
    "actions": "allocate, backfill",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
          },
          {
            "name": "gang"
          },
          {
            "name": "conformance"
          }
        ]
```

```
            },
            {
                "plugins": [
                    {
                        "name": "drf"
                    },
                    {
                        "name": "predicates"
                    },
                    {
                        "name": "nodeorder"
                    }
                ]
            },
            {
                "plugins": [
                    {
                        "name": "cce-gpu-topology-predicate"
                    },
                    {
                        "name": "cce-gpu-topology-priority"
                    },
                    {
                        "name": "cce-gpu"
                    },
                    {
                        "name": "ief",
                        "enableBestNode": true
                    }
                ]
            },
            {
                "plugins": [
                    {
                        "name": "nodelocalvolume"
                    },
                    {
                        "name": "nodeemptydirvolume"
                    },
                    {
                        "name": "nodeCSIscheduling"
                    },
                    {
                        "name": "networkresource"
                    }
                ]
            }
        ]
    },
    "server_cert": "",
    "server_key": ""
}
```

保留原 volcano-scheduler-configmap 配置

假如在某场景下希望插件升级后时沿用原配置，可参考以下步骤：

步骤 1 查看原 volcano-scheduler-configmap 配置，并备份。

示例如下：

```
# kubectl edit cm volcano-scheduler-configmap -n kube-system
apiVersion: v1
data:
  default-scheduler.conf: |-
    actions: "enqueue, allocate, backfill"
    tiers:
    - plugins:
      - name: priority
      - name: gang
      - name: conformance
    - plugins:
      - name: drf
      - name: predicates
      - name: nodeorder
      - name: binpack
        arguments:
          binpack.cpu: 100
          binpack.weight: 10
          binpack.resources: nvidia.com/gpu
          binpack.resources.nvidia.com/gpu: 10000
    - plugins:
      - name: cce-gpu-topology-predicate
      - name: cce-gpu-topology-priority
      - name: cce-gpu
    - plugins:
      - name: nodelocalvolume
      - name: nodeemptydirvolume
      - name: nodeCSIscheduling
      - name: networkresource
```

步骤 2 在控制台“参数配置”中填写自定义修改的内容：

```
{
    "ca_cert": "",
    "default_scheduler_conf": {
        "actions": "enqueue, allocate, backfill",
        "tiers": [
            {
                "plugins": [
                    {
                        "name": "priority"
                    },
                    {
                        "name": "gang"
                    },
                    {
                        "name": "conformance"
                    }
                ]
```

```
        },
        {
            "plugins": [
                {
                    "name": "drf"
                },
                {
                    "name": "predicates"
                },
                {
                    "name": "nodeorder"
                },
                {
                    "name": "binpack",
                    "arguments": {
                        "binpack.cpu": 100,
                        "binpack.weight": 10,
                        "binpack.resources": "nvidia.com/gpu",
                        "binpack.resources.nvidia.com/gpu": 10000
                    }
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "cce-gpu-topology-predicate"
                },
                {
                    "name": "cce-gpu-topology-priority"
                },
                {
                    "name": "cce-gpu"
                }
            ]
        },
        {
            "plugins": [
                {
                    "name": "nodelocalvolume"
                },
                {
                    "name": "nodeemptydirvolume"
                },
                {
                    "name": "nodeCSIscheduling"
                },
                {
                    "name": "networkresource"
                }
            ]
        }
    ]
},
"server_cert": "",
```

```
    "server_key": ""
}
```

说明

使用该功能时会覆盖原 volcano-scheduler-configmap 中内容，所以升级时务必检查是否在 volcano-scheduler-configmap 做过修改。如果是，需要把修改内容同步到升级界面里。

----结束

Volcano 1.0.0 版本升级说明

Volcano 1.0.0 版本与后续版本不兼容，不支持在控制台升级。如想使用新版本 Volcano 插件，需要先卸载 1.0.0 版本，然后再在控制台安装新版本。

执行如下命令可以卸载 Volcano。

kubectl delete crd jobs.batch.volcano.sh

kubectl delete crd commands.bus.volcano.sh

版本记录

表15-27 CCE 插件版本记录

插件版本	支持的集群版本
1.7.1	/v1.19.16.*|v1.21.*|v1.23.*|v1.25.*/
1.6.5	/v1.19.*|v1.21.*|v1.23.*/
1.4.2	/v1.15.*|v1.17.*|v1.19.*|v1.21.*/
1.3.3	/v1.15.*|v1.17.*|v1.19.*/
1.3.1	/v1.15.*|v1.17.*|v1.19.*/
1.2.5	/v1.15.*|v1.17.*|v1.19.*/
1.2.3	/v1.15.*|v1.17.*|v1.19.*/

15.15 dolphin

插件简介

dolphin 是一款容器网络流量监控管理插件，当前版本可支持从 CCE Turbo 集群安全容器以及运行时为 containerd 的普通容器的流量统计。

当前支持流量统计信息 ipv4 发送公网报文数和字节数、ipv4 接收报文数和字节数以及 ipv4 发送报文数和字节数，且支持通过 PodSelector 来对监控后端作选择，支持多监控任务、可选监控指标，且支持用户获取 Pod 的 label 标签信息。监控信息已适配 Prometheus 格式，可以通过调用 Prometheus 接口查看监控数据。

使用约束

- 仅支持在 1.19 及以上版本的 CCE Turbo 集群中安装此插件，且插件实例不支持部署在 ARM 节点上。
- 仅支持在容器引擎为 Containerd 且操作系统为 EulerOS 的节点上部署插件实例。
- 仅支持统计 CCE Turbo 集群**安全容器**（容器运行时为 kata）以及普通容器（容器运行时为 runc）的流量。
- 安装插件后默认将不进行流量监控，用户需通过创建 CR 来配置监控任务进行流量监控。
- 安装时请确保节点有足够的资源安装本插件。
- 监控标签及用户标签的来源必须为新创建 Pod 时就具有的。

安装插件

步骤 1 登录 CCE 控制台，单击左侧导航栏的“插件市场”，在“插件市场”中，单击 dolphin 插件下的“安装”。

步骤 2 在安装插件页面，在“选择集群”步骤中选择集群即可。

----结束

下发监控任务

当前用户可通过创建 CR 的方式下发监控任务，当前用户可以通过 API 或登录工作节点 kubectl apply 的方式创建 CR，后续将支持前端界面的创建。一个 CR 代表着一个监控任务，提供 selector、podLable、ip4Tx 等可选参数，具体参考以下 CR 的创建模板：

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task           #监控任务名
  namespace: kube-system       #必填，namespace 必须为 kube-system
spec:
  selector:                    #选填，配置 dolphin 插件监控的后端，形如 labelSelector 格式，默认将监控本节点所有容器
    matchLabels:
      app: nginx
    matchExpressions:
      - key: app
        operator: In
        values:
          - nginx
  podLable: [app]              #选填，用户标签
  ip4Tx:                       #选填，ipv4 发送报文数和发送字节数这两个指标的开关，默认不开
    enable: true
  ip4Rx:                       #选填，ipv4 接收报文数和发送字节数这两个指标的开关，默认不开
    enable: true
  ip4TxInternet:               #选填，ipv4 发送公网报文数和发送公网字节数这两个指标的开关，默认不开
    enable: true
```

用户标签 PodLable：可输入多个 Pod 的 label 标签，以逗号分隔，如[app, version]。

标签需符合以下规则，对应正则表达式为(^[a-zA-Z_]$)|(^([a-zA-Z][a-zA-Z0-9_]|_[a-zA-Z0-9])([a-zA-Z0-9_]){0,254}$)：

- 支持输入最多 5 个标签，单个标签长度最长 256 个字符。
- 不能以数字和双下划线__开头。
- 单个标签格式需符合 A-Za-z_0-9。

示例 1：

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  podLable: [app]
  ip4Tx:
    enable: true
```

上述监控任务名为 example-task，将监控节点上所有 Pod，生成 ipv4 发送报文数和发送字节数这两个指标，若监控的容器携带 app 这个标签，将在监控指标上携带对应 label 的 key-value 信息，否则对应 label 的 value 为“not found”。

示例 2：

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLable: [test, app]
  ip4Tx:
    enable: true
  ip4Rx:
    enable: true
  ip4TxInternet:
    enable: true
```

上述监控任务名为 example-task，将监控节点上满足 app=nginx 的 labelselector 的所有 Pod，生成全部六个指标，若监控的容器携带 test 及 app 这两个标签，将在监控指标上携带对应 label 的 key-value 信息，否则对应 label 的 value 为“not found”。

用户可以按照上述格式对监控任务进行创建、修改、及删除，当前仅支持最多 10 个监控任务的创建，且多个监控任务匹配到同一个监控后端时，每一个监控后端将会产生监控任务数量的监控指标。

说明

- 修改或删除监控任务，都将导致丢失原有监控任务所采集的监控数据，请谨慎操作。
- 用户卸载插件后，用户之前配置的监控任务 CR 将随着插件卸载一并销毁。

查看流量统计

dolphin 插件的监控信息以普罗米修斯 exporter 格式输出，有两种方式获取 dolphin 插件的监控信息：

- 安装 prometheus 插件，prometheus 插件会自动对接 dolphin 插件并周期性采集监控信息。
- 直接访问 dophin 插件提供的服务端口 10001，形如 http://{POD_IP}:10001/metrics

注意，如果在节点上访问 dolphin 服务端口，需要放通节点和 pod 的安全组限制。

您可以通过安装 Prometheus 插件查看监控信息。

表15-28 当前支持的监控指标

监控指标	对应参数
ipv4 发送公网报文数	ip4_send_pkt_internet
ipv4 发送公网字节数	ip4_send_byte_internet
ipv4 接收报文数	ip4_rcv_pkt
ipv4 接收字节数	ip4_rcv_byte
ipv4 发送报文数	ip4_send_pkt
ipv4 发送字节数	ip4_send_byte

- 示例 1（ipv4 发送公网报文数）：

```
dolphin_ip4_send_pkt_internet{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 241
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的发送公网报文数为 241。

- 示例 2（ipv4 发送公网字节数）：

```
dolphin_ip4_send_byte_internet{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task" } 23618
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的发送公网字节数为 23618。

- 示例 3（ipv4 发送报文数）：

```
dolphin_ip4_send_pkt{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 379
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的发送报文数为 379。

- 示例 4（ipv4 发送字节数）：

```
dolphin_ip4_send_byte{app="nginx",pod="default/nginx-66c9c65dbf-
zjg24",task="kube-system/example-task "} 33129
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的发送字节数为 33129。

- 示例 5（ipv4 接收报文数）：

```
dolphin_ip4_rcv_pkt{app="nginx",pod="default/nginx-66c9c65dbf-
zjg24",task="kube-system/example-task "} 464
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的接收报文数为 464。

- 示例 6（ipv4 接收字节数）：

```
dolphin_ip4_rcv_byte{app="nginx",pod="default/nginx-66c9c65dbf-
zjg24",task="kube-system/example-task "} 34654
```

如上示例中，pod 所在命名空间为 default，pod 名称为 nginx-66c9c65dbf-zjg24，用户指定 label 为 app，其值对应为 nginx，该监控指标由名称为 example-task 的监控任务创建，该 pod 的接收字节数为 34654。

说明

若容器无用户指定的标签，返回体中标签值为 not found。形如：

dolphin_ip4_send_byte_internet{test="not found", pod="default/nginx-66c9c65dbf-zjg24",task="default" } 23618

版本记录

表15-29 CCE 插件版本记录

插件版本	支持的集群版本
1.2.2	/v1.(19|21|23|25).*/
1.1.6	/v1.(19|21|23).*/
1.1.5	/v1.(19|21|23).*/
1.1.2	/v1.(19|21|23).*/
1.0.1	/v1.(19|21).*/

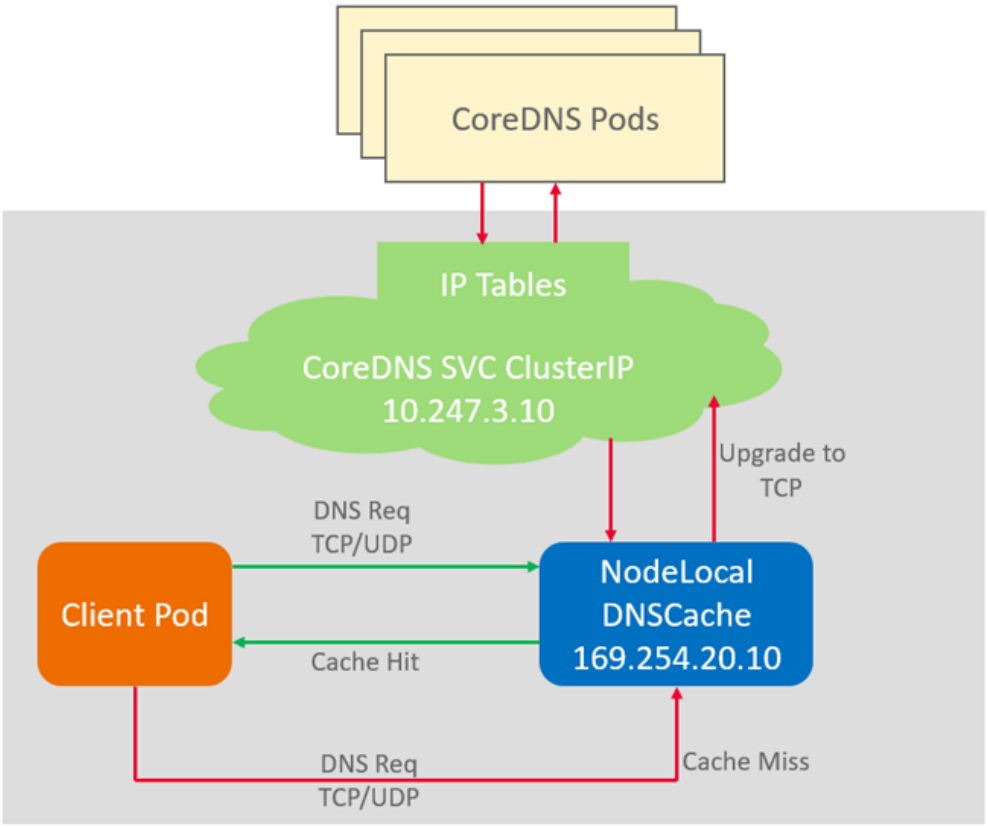
15.16 node-local-dns

插件简介

NodeLocal DNSCache 通过在集群节点上作为守护程序集运行 DNS 缓存代理，提高集群 DNS 性能。

开源社区地址：https://github.com/kubernetes/dns

图15-11 NodeLocal DNSCache 查询路径



约束与限制

- 仅支持 1.19 及以上版本集群。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **node-local-dns**，单击“安装”。

步骤 2 在安装插件页面，选择插件规格，并配置相关参数。

enable_dnsconfig_admission：是否自动注入。默认为 **false**，设置为 **true** 表示自动注入。

步骤 3 完成以上配置后，单击“安装”。

----结束

版本记录

表15-30 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
1.2.2	/v1.(19|21|23).*/	1.21.1

15.17 kube-prometheus-stack

插件简介

kube-prometheus-stack 通过使用 Prometheus-operator 和 Prometheus，提供简单易用的端到端 Kubernetes 集群监控能力。

使用 kube-prometheus-stack 可将监控数据与容器智能分析对接，在容器智能分析控制台查看监控数据，配置告警等。

开源社区地址：https://github.com/prometheus/prometheus

约束与限制

在默认配置下，插件中的 kube-state-metrics 组件不采集 Kubernetes 资源的所有的 labels 和 annotation。如需采集，您需要手动在启动参数中开启采集开关，并同时检查名称为 kube-state-metrics 的 ServiceMonitor 中采集白名单是否添加相应指标，详情请参见采集 Pod 所有 labels 和 annotations。

安装插件

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“插件管理”，在右侧找到 **kube-prometheus-stack**，单击“安装”。

步骤 2 在安装插件页面，选择插件规格，并配置相关参数。

- **开启智能分析**：默认不开启。开启后需选择智能分析工作区，在容器智能分析控制台查看监控数据，配置告警等。
- **对接第三方**：将普罗数据上报至第三方监控系统，需填写第三方监控系统的地址和 Token，并选择是否跳过证书认证。
- **普罗高可用**：高可用会在集群中将 Prometheus-server、Prometheus-operator、thanos-query、custom-metrics-apiserver、alertmanager 组件按多实例方式部署。
- **安装 grafana**：通过 grafana 可视化浏览普罗监控数据。grafana 会默认创建大小为 5 GiB 的存储卷，卸载插件时 grafana 的**存储卷不随插件被删除**。首次登录默认用户名与密码均为 admin，登录后会立即让您修改密码。
- **采集周期**：采集监控数据的周期。
- **数据保留期**：监控数据保留的时长。
- **存储**：选择用于存储监控数据的磁盘类型和大小。

- **调度策略**：可单独配置插件各个组件的节点亲和性和污点容忍能力。可以配置多个调度策略，不配置亲和节点键和容忍节点污点键则默认不开启对应的调度策略。
  - 作用范围：可选择调度策略生效的插件实例，默认对全部实例生效。当指定组件实例名称时，将覆盖全部实例所配置的调度策略。
  - 亲和节点标签键：填写节点标签键，为插件实例设置节点亲和性。
  - 亲和节点标签值：填写节点标签值，为插件实例设置节点亲和性。
  - 容忍节点污点键：目前仅支持污点键级别的污点容忍策略，组件可以调度到拥有该污点键的节点。

步骤 3 完成以上配置后，单击“安装”。

----结束

配置自定义指标

新版本的 kube-prometheus-stack 插件不再提供自定义配置的指标，即 user-adapter-config 配置项（历史版本插件中该配置项的名称为 adapter-config）中不再配置指标采集规则，请您自行添加。关于采集规则配置详情请参见 Metrics Discovery and Presentation Configuration。如您从老版本插件升级至新版，则原有的该配置会被继承，不会丢失。

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“配置项与密钥”。

步骤 2 切换至“monitoring”命名空间，在“配置项”页签找到 user-adapter-config 配置项（历史版本插件中该配置项的名称为 adapter-config），并单击“更新”。

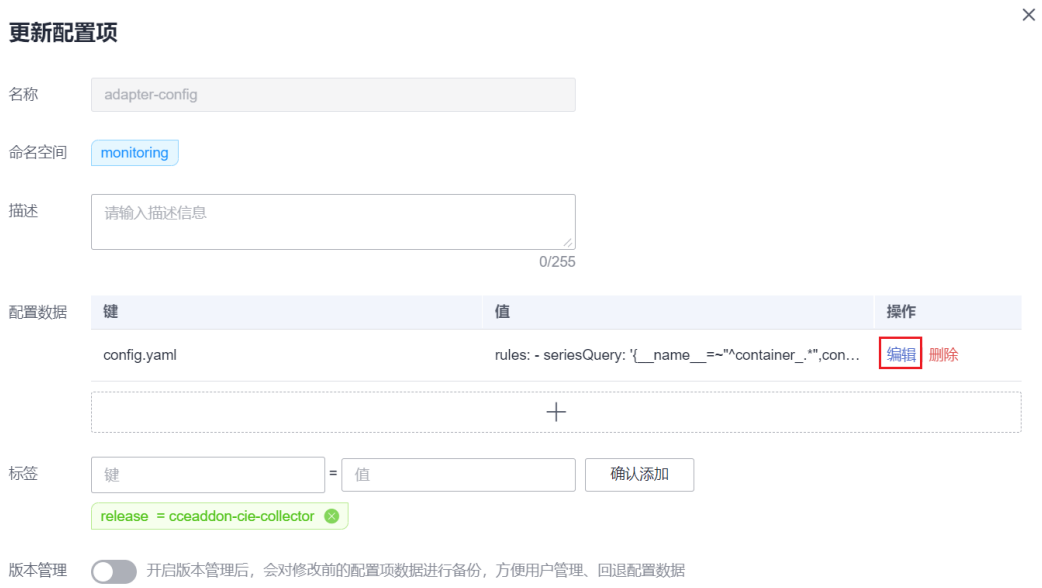
步骤 3 在“配置数据”中单击 config.yaml 对应的“编辑”按钮，在 rules 字段下添加自定义指标采集规则。修改完成后单击“确定”保存配置。

如果您需要增加多个采集规则，可在 rules 字段下添加多个配置，关于采集规则配置详情请参见 Metrics Discovery and Presentation Configuration。

自定义采集规则示例如下：

```
rules:
- seriesQuery: '{__name__=~"^container_.*",container!="POD",namespace!="",pod!=""}'
  resources:
    overrides:
      namespace:
```

```
      resource: namespace
    pod:
      resource: pod
  name:
    matches: "^container_(.*)_seconds_total$"
  metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>,container!="POD"}[2m])) by
(<<.GroupBy>>)'
```



步骤 4 重新部署 monitoring 命名空间下的 custom-metrics-apiserver 工作负载。

----结束

组件说明

安装 kube-prometheus-stack 插件创建的 Kubernetes 资源，全部都创建在 monitoring 命名空间下。

有状态负载：

- prometheus-server
- alertmanager-alertmanager

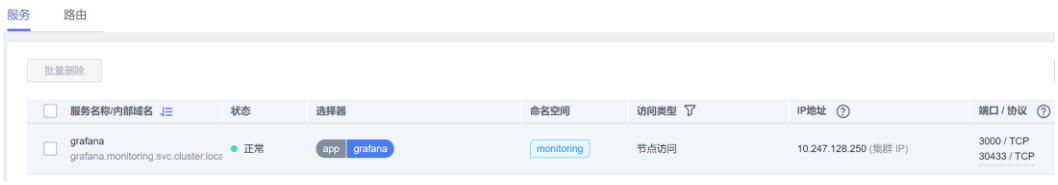
无状态负载：

- thanos-query
- event-exporter
- custom-metrics-apiserver
- grafana
- kube-state-metric
- prometheus-operator

访问 grafana

如果在安装插件时安装了 grafana，可以通过名为 grafana 的 Service 访问，grafana Service 类型为节点访问，如果是从外网访问，可以给节点绑定 EIP，通过节点端口访问。

如下图，访问地址为 **http://{{节点 IP}}:30433**



采集 Pod 所有 labels 和 annotations

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“工作负载”。

步骤 2 切换至“monitoring”命名空间，在“无状态负载”页签找到 kube-state-metrics 负载，并单击“升级”。

步骤 3 在容器配置的“生命周期”中，编辑启动命令。

在原有的 kube-state-metrics 的启动参数最后添加：

```
--metric-labels-allowlist=pods=[*],nodes=[node,failure-
domain.beta.kubernetes.io/zone,topology.kubernetes.io/zone]
```

如需采集 annotations 时，则在启动参数中以相同方法添加参数：

```
--metric-annotations-allowlist=pods=[*],nodes=[node,failure-
domain.beta.kubernetes.io/zone,topology.kubernetes.io/zone]
```

须知

编辑启动命令时，请勿修改其他原有的启动参数，否则可能导致组件异常。

步骤 4 kube-state-metrics 将开始采集 Pod 和 node 的 labels/annotations 指标，查询 kube_pod_labels/kube_pod_annotations 是否在普罗的采集任务中。

```
kubectl get servicemonitor kube-state-metrics -nmonitoring -oyaml | kube_pod_labels
```

----结束

更多 kube-state-metrics 的启动参数请参见 kube-state-metrics/cli-arguments。

版本记录

表15-31 CCE 插件版本记录

插件版本	支持的集群版本	社区版本（仅 1.17 及以上版本集群支持）
3.6.6	/v1.(19|21|23|25).*/	2.35.0
3.5.1	/v1.(19|21|23).*/	2.35.0
3.5.0	/v1.(19|21|23).*/	2.35.0

16 模板管理（helm）

16.1 概述

CCE 提供了管理 Helm Chart（模板）的控制台，能够帮助您方便的使用模板部署应用，并在控制台上管理应用。CCE 使用的 Helm 版本为 v3.8.2，支持上传 Helm v3 语法的模板包，具体请参见通过模板部署应用。

您也可以直接使用 Helm 客户端直接部署应用，使用 Helm 客户端部署应用不受版本控制，可以使用 Helm v2 或 v3，具体请参见通过 Helm v2 客户端部署应用及通过 Helm v3 客户端部署应用。

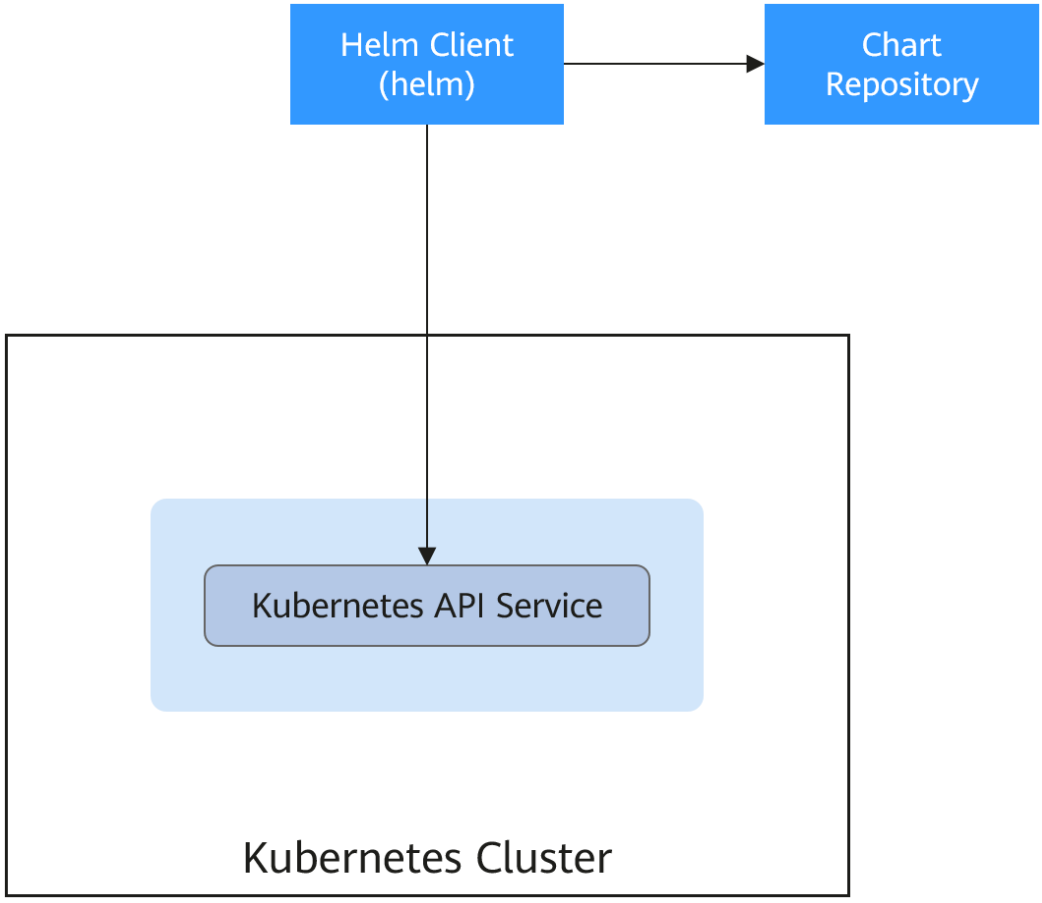
Helm

Helm 是 Kubernetes 的包管理器，主要用来管理 Charts。Helm Chart 是用来封装 Kubernetes 原生应用程序的一系列 YAML 文件。可以在您部署应用的时候自定义应用程序的一些 Metadata，以便于应用程序的分发。对于应用发布者而言，可以通过 Helm 打包应用、管理应用依赖关系、管理应用版本并发布应用到软件仓库。对于使用者而言，使用 Helm 后不用需要编写复杂的应用部署文件，可以以简单的方式在 Kubernetes 上查找、安装、升级、回滚、卸载应用程序。

Helm 和 Kubernetes 之间的关系可以如下类比：

- Helm < - > Kubernetes
- Apt < - > Ubuntu
- Yum < - > CentOS
- Pip < - > Python

Helm 的整体架构如下图：



Kubernetes 的应用编排存在着一些问题，Helm 可以用来解决这些问题，如下：

- 管理、编辑与更新大量的 Kubernetes 配置文件。
- 部署一个含有大量配置文件的复杂 Kubernetes 应用。
- 分享和复用 Kubernetes 配置和应用。
- 参数化配置模板支持多个环境。
- 管理应用的发布：回滚、diff 和查看发布历史。
- 控制一个部署周期中的某一些环节。
- 发布后的测试验证。

16.2 通过模板部署应用

在 CCE 控制台上，您可以上传 Helm 模板包，然后在控制台安装部署，并对部署的实例进行管理。

须知

云容器引擎各 region 将逐步切换至 Helm v3。模板管理未来将不再支持 Helm v2 版本的模板，若您在短期内不能切换至 Helm v3，可通过 Helm v2 客户端在后台管理 v2 版本的模板。

约束与限制

- 单个用户可以上传模板的个数有限制，请以各个 Region 控制台界面中提示的实际值为准。
- CCE 使用的 Helm 版本为 v3.8.2，支持上传 Helm v3 版本语法的模板包。
- 模板若存在多个版本，则消耗对应数量的模板配额。
- 由于模板的操作权限同时具有较高的集群操作权限，因此租户应当谨慎授予用户对于模板生命周期管理的权限，包括上传模板的权限，以及创建、删除和更新模板实例的权限。

模板包规范

以下以 redis 为例，在准备 redis 模板包时根据模板包规范制作模板包。

- **命名要求**

  模板包命名格式为：**{name}-{version}**.tgz，其中**{version}**为版本号，格式为“主版本号.次版本号.修订号”，如 redis-0.4.2.tgz。

  说明

  模板名称{name}的长度不能超过 64 个字符。

  版本号需遵循语义化版本规则。

  - 主版本号、次版本号为必选，修订号为可选。
  - 主版本号、次版本号、修订号的数值为整数，均需要≥0，且≤99。

- **目录结构**

  模板包的目录结构如下所示：

```
redis/
  templates/
  values.yaml
  README.md
  Chart.yaml
  .helmignore
```

  目录说明如下表所示，带*的为必选项：

表16-1 模板包目录说明

参数	参数说明
* templates	用于存放所有的 template（模板）文件。
* values.yaml	用于描述 template 文件所需的配置参数。 须知 定义 template 文件配置参数时，请注意此处定义的“镜像地址”务必和容器

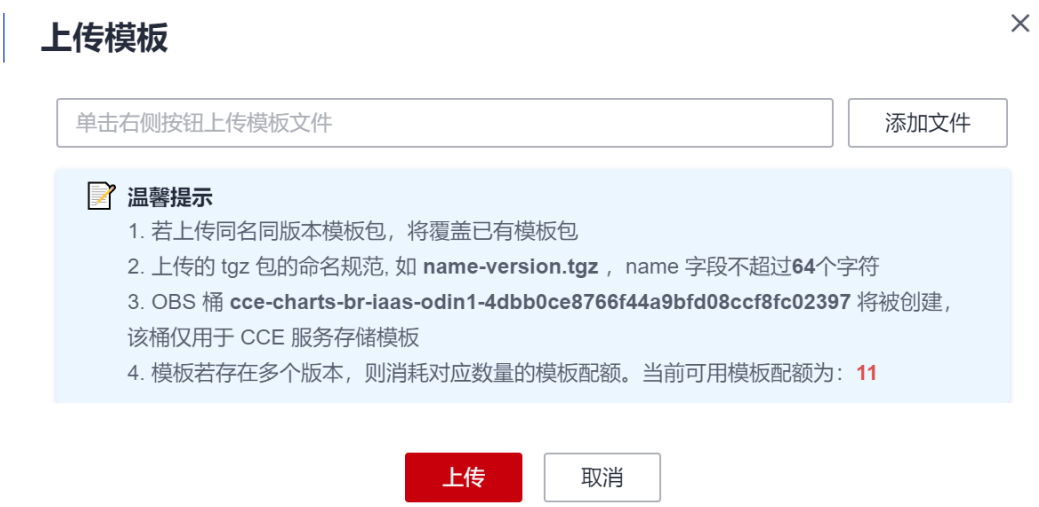
参数	参数说明
	镜像仓库中对应的镜像地址保持一致。否则创建工作负载会异常，提示镜像拉取失败。 镜像地址获取方法如下：在 CCE 控制台，单击左侧导航栏的“镜像仓库”，进入容器镜像服务控制台。在“我的镜像 > 自有镜像”中，单击已上传镜像的名称，在“镜像版本 ”页签的“下载指令”栏中即可获取镜像地址，单击 按钮即可复制该指令。
README.md	一个 markdown 文件，包括： • 描述 Chart 提供的工作负载或服务。 • 运行 Chart 的前提。 • 解释 values.yaml 文件中的配置。 • 安装和配置 Chart 的相关信息。
* Chart.yaml	模板的基本信息说明。 注：Helm v3 版本 apiVersion 从 v1 切换到了 v2。
.helmignore	设定在工作负载安装时不需要读取 templates 的某些文件或数据。

上传模板

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“模板管理”，在右上角单击“上传模板”。

步骤 2 单击“添加文件”，选中待上传的工作负载包后，单击“上传”。

图16-1 上传模板包



说明

由于上传模板时创建 OBS 桶的命名规则由 cce-charts-{region}-**{domain_name}**变为 cce-charts-{region}-**{domain_id}**，其中旧命名规则中的 domain_name 系统会做 base64 转化并取前 63 位，如果您在现有命名规则的 OBS 桶中找不到模板，请在旧命名规则的桶中进行查找。

----结束

创建模板实例

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“模板管理”。

步骤 2 在已上传的模板中，单击“安装”。

步骤 3 参照下表设置安装工作负载参数。

表16-2 安装工作负载参数说明

参数	参数说明
实例名称	新建模板实例名称，命名必须唯一。
命名空间	指定部署的命名空间。
选择版本	选择模板的版本。
配置文件	用户可以导入 values.yaml 文件，导入后可替换模板包中的 values.yaml 文件；也可直接在配置框中在线编辑模板参数。 **说明** 此处导入的 values.yaml 文件需符合 yaml 规范，即 KEY:VALUE 格式。对于文件中的字段不做任何限制。 导入的 value.yaml 的 key 值必须与所选的模板包的 values.yaml 保持一致，否则不会生效。即 key 不能修改。 1. 单击“添加文件”。 2. 选择对应的 values.yaml 文件，单击“打开”。

步骤 4 配置完成后，单击“安装”。

在“模板实例”页签下可以查看模板实例的安装情况。

----结束

升级模板工作负载

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“模板管理”，在右侧选择“模板实例”页签。

步骤 2 单击待升级工作负载后的“升级”，设置升级模板工作负载的参数。

步骤 3 选择对应的模板版本。

步骤 4 参照界面提示修改模板参数。单击“升级”，再单击“提交”。

步骤 5 单击“返回模板实例列表”，模板状态为“升级成功”时，表明工作负载升级成功。

----结束

回退模板工作负载

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“模板管理”，在右侧选择“模板实例”页签。

步骤 2 单击待回退工作负载后的“回退”，选择要回退的工作负载版本，单击“回退”。

模板工作负载列表中，状态为“回退成功”时，表明工作负载回退成功。

----结束

卸载模板工作负载

步骤 1 登录 CCE 控制台，单击集群名称进入集群，在左侧导航栏中选择“模板管理”，在右侧选择“模板实例”页签。

步骤 2 单击待卸载模板实例后的“更多 > 卸载”，确认待卸载模板实例后，单击“是”。模板实例卸载后不能恢复，请谨慎操作。

----结束

16.3 Helm v2 与 Helm v3 的差异及适配方案

随着 Helm v2 发布最终版本 Helm 2.17.0，Helm v3 现在已是 Helm 开发者社区支持的唯一标准。为便于管理，建议用户尽快将模板切换至 Helm v3 格式。

当前社区从 Helm v2 演进到 Helm v3，主要有以下变化：

1. **移除 tiller**

   Helm v3 使用更加简单和灵活的架构，移除了 tiller，直接通过 kubeconfig 连接 apiserver，简化安全模块，降低了用户的使用壁垒。

2. **改进了升级策略，采用三路策略合并补丁**

   Helm v2 使用双路策略合并补丁。在升级过程中，会对比最近一次发布的 chart manifest 和本次发布的 chart manifest 的差异，来决定哪些更改会应用到 Kubernetes 资源中。如果更改是集群外带的（比如通过 kubectl edit），则修改不会被 Helm 识别和考虑。结果就是资源不会回滚到之前的状态。

   Helm v3 使用三路策略来合并补丁，Helm 在生成一个补丁时，会考虑之前老的 manifest 的活动状态。因此，Helm 在使用老的 chart manifest 生成新补丁时会考虑当前活动状态，并将其与之前老的 manifest 进行比对，并再比对新的 manifest 是否有改动，并进行自动补全，以此来生成最终的更新补丁。

   详情及示例请见 Helm 官方文档：https://v3.helm.sh/docs/faq/changes_since_helm2

3. **默认存储驱动程序更改为 secrets**

   Helm v2 默认情况下使用 ConfigMaps 存储发行信息，而在 Helm v3 中默认使用 Secrets。

4. **发布名称限制在 namespace 范围内**

   Helm v2 只使用 tiller 的 namespace 作为 release 信息的存储，这样全集群的 release 名字都不能重复。Helm v3 只会在 release 安装的所在 namespace 记录对应的信息，这样 release 名称可在不同命名空间重用。应用和 release 命名空间一致。

5. **校验方式改变**

   Helm v3 对模板格式的校验更加严格，如 Helm v3 将 chart.yaml 的 apiVersion 从 v1 切换到 v2，针对 Helm v2 的 chart.yaml，强要求指定 apiVersion 为 v1。可安装 Helm v3 客户端后，通过执行 helm lint 命令校验模板格式是否符合 v3 规范。

   适配方案：根据 Helm 官方文档 https://helm.sh/docs/topics/charts/适配 Helm v3 模板，apiVersion 字段必填。

6. **废弃 crd-install**

   Helm v3 删除了 crd-install hook， 并用 chart 中的 crds 目录替换。需要注意的是，crds 目录中的资源只有在 release 安装时会部署，升级时不会更新，删除时不会卸载 crds 目录中的资源。若 crd 已存在，则重复安装不会报错。

   适配方案：根据 Helm 官方文档 https://helm.sh/docs/chart_best_practices/custom_resource_definitions/，当前可使用 crds 目录或者将 crd 定义单独放入 chart。考虑到目前不支持使用 Helm 升级或删除 CRD，推荐分隔 chart，将 CRD 定义放入 chart 中，然后将所有使用该 CRD 的资源放到另一个 chart 中进行管理。

7. **未通过 helm 创建的资源不强制 update，releaes 默认不强制升级**

   Helm v3 强制升级逻辑变化，不再是升级失败后走删除重建，而是直接走 put 更新逻辑。因此当前 CCE release 升级默认使用非强制更新逻辑，无法通过 Patch 更新的资源将导致 release 升级失败。若环境存在同名资源且无 Helm V3 的归属标记 app.kubernetes.io/managed-by: Helm，则会提示资源冲突。

   适配方案：删除相关资源，并通过 Helm 创建。

8. **Release history 数量限制更新**

   为避免 release 历史版本无限增加，当前 release 升级默认只保留最近 10 个历史版本。

更多变化和详细说明请参见 Helm 官方文档

- Helm v2 与 Helm v3 的区别：https://v3.helm.sh/docs/faq/changes_since_helm2
- Helm v2 如何迁移到 Helm v3：https://helm.sh/docs/topics/v2_v3_migration

16.4 通过 Helm v2 客户端部署应用

须知

云容器引擎各 region 将逐步切换至 Helm v3。模板管理不再支持 Helm v2 版本的模板，若您在短期内不能切换至 Helm v3，可通过 Helm v2 客户端在后台管理 v2 版本的模板。

前提条件

在 CCE 中创建的 Kubernetes 集群已对接 kubectl，具体请参见使用 kubectl 连接集群。

注意事项

CCE 当前会尝试转换 v2 模板实例到 v3 模板实例。若在后台操作 Helm v2 模板实例，删除实例后，发现 CCE 模板管理页面仍有实例信息，单击删除即可。

安装 Helm v2

本文以 Helm v2.17.0 为例进行演示。

如需选择其他合适的版本，请访问 https://github.com/helm/helm/releases。

步骤 1 在连接集群的虚拟机上下载 Helm 客户端。

```
wget https://get.helm.sh/helm-v2.17.0-linux-amd64.tar.gz
```

步骤 2 解压 Helm 包。

```
tar -xzvf helm-v2.17.0-linux-amd64.tar.gz
```

步骤 3 将 helm 拷贝到系统 path 路径下，以下为/usr/local/bin/helm。

```
mv linux-amd64/helm /usr/local/bin/helm
```

步骤 4 因为 Kubernetes APIServer 开启了 RBAC 访问控制，所以需创建 tiller 使用的 service account:tiller 并给其分配 cluster-admin 这个集群内置的 ClusterRole。按如下创建 tiller 的资源帐户。

vim tiller-rbac.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

步骤 5 部署 tiller 资源帐户。

```
kubectl apply -f tiller-rbac.yaml
```

步骤 6 初始化 Helm，部署 tiller 的 Pod。

```
helm init --service-account tiller --skip-refresh
```

步骤 7 查看状态。

```
kubectl get pod -n kube-system -l app=helm
```

回显如下

```
NAME                            READY   STATUS    RESTARTS   AGE
tiller-deploy-7b56c8dfb7-fxk5g   1/1    Running   1          23h
```

步骤 8 查看 helm 版本。

```
# helm version
Client: &version.Version{SemVer:"v2.17.0",
GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.17.0",
GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe", GitTreeState:"clean"}
```

----结束

安装 Helm 模板 chart 包

CCE 提供的模板不能满足要求时，可下载模板的 chart 包进行安装。

在 https://github.com/helm/charts 的 stable 目录中查找您需要的 chart 包，下载后将 chart 包上传至节点。

1. 下载并解压已获取的 chart 包，一般 chart 包格式为.zip。

   ```
   unzip chart.zip
   ```

2. 安装 Helm 模板。

   ```
   helm install aerospike/
   ```

3. 安装完成后，执行 **helm list** 查看已经安装的模板实例状态。

常见问题

- **执行 Helm version 时，提示如下错误信息：**

  ```
  Client:
  &version.Version{SemVer:"v2.17.0",
  GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe", GitTreeState:"clean"}
  E0718 11:46:10.132102    7023 portforward.go:332] an error occurred
  forwarding 41458 -> 44134: error forwarding port 44134 to pod
  d566b78f997eea6c4b1c0322b34ce8052c6c2001e8edff243647748464cd7919, uid : unable
  to do port forwarding: socat not found.
  Error: cannot connect to Tiller
  ```

  出现上述问题，说明未安装 socat，请执行如下命令安装 socat。

  ```
  yum install socat -y
  ```

- **socat 已安装，执行 Helm version 时，提示如下错误信息：**

  ```
  test@local:~/k8s/helm/test$ helm version
  Client: &version.Version{SemVer:"v3.3.0",
  GitCommit:"021cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
  Error: cannot connect to Tiller
  ```

Helm 模板从“.Kube/config”中读取配置证书和 kubernetes 进行通讯，出现上述错误信息说明 kubectl 配置有误，请重新对接 kubectl，具体请参见使用 kubectl 连接集群。

- **对接云存储后，存储未创建成功。**

  出现上述问题可能是创建的 pvc 中 annotation 字段导致的，请修改模板名称后再次进行安装。

- **如果 kubectl 没有配置好，helm install 时会出现如下报错：**

```
[root@prometheus-57046 ~]# helm install prometheus/ --generate-name
WARNING: This chart is deprecated
Error: Kubernetes cluster unreachable: Get
"http://localhost:8080/version?timeout=32s": dial tcp [::1]:8080: connect:
connection refused
```

  解决办法：给节点配置 kubeconfig，配置方法请参见使用 kubectl 连接集群。

16.5 通过 Helm v3 客户端部署应用

前提条件

在 CCE 中创建的 Kubernetes 集群已对接 kubectl，具体请参见使用 kubectl 连接集群。

安装 Helm v3

本文以 Helm v3.3.0 为例进行演示。

如需选择其他合适的版本，请访问 https://github.com/helm/helm/releases。

步骤 1 在连接集群的虚拟机上下载 Helm 客户端。

```
wget https://get.helm.sh/helm-v3.3.0-linux-amd64.tar.gz
```

步骤 2 解压 Helm 包。

```
tar -xzvf helm-v3.3.0-linux-amd64.tar.gz
```

步骤 3 将 Helm 拷贝到系统 path 路径下，以下为/usr/local/bin/helm。

```
mv linux-amd64/helm /usr/local/bin/helm
```

步骤 4 查看 Helm 版本。

```
helm version
version.BuildInfo{Version:"v3.3.0",
GitCommit:"e29ce2a54e96cd02ccfce88bee4f58bb6e2a28b6", GitTreeState:"clean",
GoVersion:"go1.13.4"}
```

----结束

安装 Helm 模板 chart 包

CCE 提供的模板不能满足要求时，可下载模板的 chart 包进行安装。

在 https://github.com/helm/charts 的 stable 目录中查找您需要的 chart 包，下载后将 chart 包上传至节点。

1. 下载并解压已获取的 chart 包，一般 chart 包格式为.zip。

```
unzip chart.zip
```

2. 安装 Helm 模板。

```
helm install aerospike/ --generate-name
```

3. 安装完成后，执行 **helm list** 查看已经安装的模板实例状态。

常见问题

- **执行 Helm version 时，提示如下错误信息：**

```
Client:
&version.Version{SemVer:"v3.3.0",
GitCommit:"012cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
E0718 11:46:10.132102    7023 portforward.go:332] an error occurred
forwarding 41458 -> 44134: error forwarding port 44134 to pod
d566b78f997eea6c4b1c0322b34ce8052c6c2001e8edff243647748464cd7919, uid : unable
to do port forwarding: socat not found.
Error: cannot connect to Tiller
```

出现上述问题，说明未安装 socat，请执行如下命令安装 socat。

yum install socat -y

- **socat 已安装，执行 Helm version 时，提示如下错误信息：**

```
$ helm version
Client: &version.Version{SemVer:"v3.3.0",
GitCommit:"021cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
Error: cannot connect to Tiller
```

Helm 模板从“.Kube/config”中读取配置证书和 kubernetes 进行通讯，出现上述错误信息说明 kubectl 配置有误，请重新对接 kubectl，具体请参见使用 kubectl 连接集群。

- **对接云存储后，存储未创建成功。**

出现上述问题可能是创建的 pvc 中 annotation 字段导致的，请修改模板名称后再次进行安装。

- **如果 kubectl 没有配置好，helm install 时会出现如下报错：**

```
# helm install prometheus/ --generate-name
WARNING: This chart is deprecated
Error: Kubernetes cluster unreachable: Get
"http://localhost:8080/version?timeout=32s": dial tcp [::1]:8080: connect:
connection refused
```

解决办法：给节点配置 kubeconfig，配置方法请参见使用 kubectl 连接集群。

16.6 Helm v2 Release 转换成 Helm v3 Release

背景介绍

当前 CCE 已全面支持 Helm v3 版本，用户可通过本指南将已创建的 v2 release 转换成 v3 release，从而更好地使用 v3 的特性。因 Helm v3 底层相对于 Helm v2 来说，一些功能已被弃用或重构，因此转换会有一定风险，需转换前进行模拟转换。

该指南参考社区文档：https://github.com/helm/helm-2to3

注意事项：

- Helm v2 release 信息存储在 configmap 中，Helm v3 release 信息存储在 secrets 中。
- 若用户通过前端 console 操作，在获取实例、更新实例等操作中 CCE 会自动尝试转换 v2 模板实例到 v3 模板实例。若用户仅在后台操作实例，需通过该指南进行转换操作。

转换流程（不使用 Helm v3 客户端）

步骤 1 在 CCE 节点上下载 helm 2to3 转换插件。

```
wget https://github.com/helm/helm-2to3/releases/download/v0.10.2/helm-
2to3_0.10.2_linux_amd64.tar.gz
```

步骤 2 解压插件包。

```
tar -xzvf helm-2to3_0.10.2_linux_amd64.tar.gz
```

步骤 3 模拟转换。

以 test-convert 实例为例，执行以下命令进行转换的模拟。若出现以下提示，说明模拟转换成功。

```
# ./2to3 convert --dry-run --tiller-out-cluster -s configmaps test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
```

步骤 4 执行正式转换。若出现以下提示，说明转换成功。

```
# ./2to3 convert --tiller-out-cluster -s configmaps test-convert
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" created.
[Helm 3] Release "test-convert" created.
Release "test-convert" was converted successfully from Helm v2 to Helm v3.
Note: The v2 release information still remains and should be removed to avoid
conflicts with the migrated v3 release.
v2 release information should only be removed using `helm 2to3` cleanup and when
all releases have been migrated over.
```

步骤 5 转换完成后进行 v2 release 资源的清理，同样先进行模拟清理，成功后正式清理 v2 release 资源。

模拟清理：

```
# ./2to3 cleanup --dry-run --tiller-out-cluster -s configmaps --name test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
```

正式清理：

```
# ./2to3 cleanup --tiller-out-cluster -s configmaps --name test-convert
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" d
```

----结束

转换流程（使用 Helm v3 客户端）

步骤 1 安装 Helm v3 客户端，参见安装 Helm v3。

步骤 2 安装转换插件。

```
# helm plugin install https://github.com/helm/helm-2to3
Downloading and installing helm-2to3 v0.10.2 ...
https://github.com/helm/helm-2to3/releases/download/v0.10.2/helm-
2to3_0.10.2_linux_amd64.tar.gz
Installed plugin: 2to3
```

步骤 3 查看已安装的插件，确认插件已安装。

```
# helm plugin list
NAME VERSION DESCRIPTION
2to3  0.10.2     migrate and cleanup Helm v2 configuration and releases in-place to
Helm v3
```

步骤 4 模拟转换。

以 test-convert 实例为例，执行以下命令进行转换的模拟。若出现以下相关提示，说明模拟转换成功。

```
# helm 2to3 convert --dry-run --tiller-out-cluster -s configmaps test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
Release "test-convert" will be converted from Helm v2 to Helm v3.
```

```
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
```

步骤 5 执行正式转换。若出现以下提示，说明转换成功。

```
# helm 2to3 convert --tiller-out-cluster -s configmaps test-convert
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" created.
[Helm 3] Release "test-convert" created.
Release "test-convert" was converted successfully from Helm v2 to Helm v3.
Note: The v2 release information still remains and should be removed to avoid
conflicts with the migrated v3 release.
v2 release information should only be removed using `helm 2to3` cleanup and when
all releases have been migrated over.
```

步骤 6 正式转换成功后，用户可通过 helm list 查看已转换成功的模板实例。

```
# helm list
NAME             NAMESPACE    REVISION UPDATED                      STATUS     CHART
APP VERSION
test-convert     default   1             2022-08-29 06:56:28.166918487 +0000 UTC
deployed   test-helmold-1
```

步骤 7 转换完成后进行 V2 release 资源的清理，同样先进行模拟清理，成功后正式清理 V2 release 资源。

模拟清理：

```
# helm 2to3 cleanup --dry-run --tiller-out-cluster -s configmaps --name test-
convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
```

正式清理：

```
# helm 2to3 cleanup --tiller-out-cluster -s configmaps --name test-convert
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" deleted.
[Helm 2] Release 'test-convert' deleted.
Helm v2 data was cleaned up successfully.
```

----结束

17 权限管理

17.1 CCE 权限概述

CCE 权限管理是在统一身份认证服务（IAM）与 Kubernetes 的角色访问控制（RBAC）的能力基础上，打造的细粒度权限管理功能，支持基于 IAM 的细粒度权限控制和 IAM Token 认证，支持集群级别、命名空间级别的权限控制，帮助用户便捷灵活的对租户下的 IAM 用户、用户组设定不同的操作权限。

如果您需要对 CCE 集群及相关资源进行精细的权限管理，例如限制不同部门的员工拥有部门内资源的细粒度权限，您可以使用 CCE 权限管理提供的增强能力进行多维度的权限管理。

本章节将介绍 CCE 权限管理机制及其涉及到的基本概念。如果当前帐号已经能满足您的要求，您可以跳过本章节，不影响您使用 CCE 服务的其它功能。

CCE 支持的权限管理能力

CCE 的权限管理包括“集群权限”和“命名空间权限”两种能力，能够从集群和命名空间层面对用户组或用户进行细粒度授权，具体解释如下：

- **集群权限：**是基于 IAM 系统策略的授权，可以通过用户组功能实现 IAM 用户的授权。用户组是用户的集合，通过集群权限设置可以让某些用户组操作集群（如创建/删除集群、节点、节点池、模板、插件等），而让某些用户组仅能查看集群。

  集群权限涉及 CCE 非 Kubernetes API，支持 IAM 细粒度策略、企业项目管理相关能力。

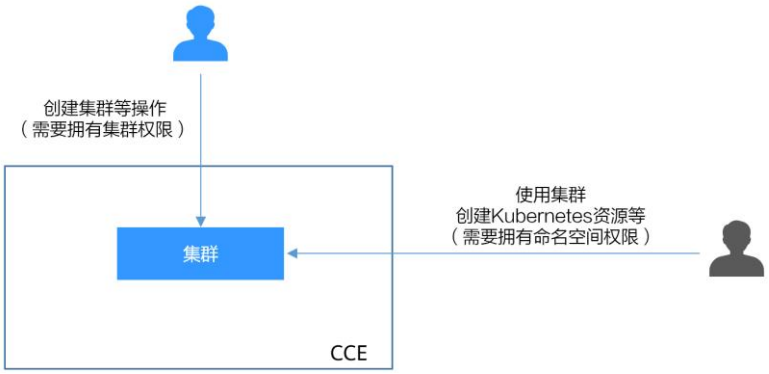
- **命名空间权限：**是基于 Kubernetes RBAC（Role-Based Access Control，基于角色的访问控制）能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同 Kubernetes 资源的权限。同时 CCE 基于开源能力进行了增强，可以支持基于 IAM 用户或用户组粒度进行 RBAC 授权、IAM token 直接访问 API 进行 RBAC 认证鉴权。

  命名空间权限涉及 CCE Kubernetes API，基于 Kubernetes RBAC 能力进行增强，支持对接 IAM 用户/用户组进行授权和认证鉴权，但与 IAM 细粒度策略独立。

  CCE 从 v1.11.7-r2 版本起的集群支持配置命名空间权限，1.11.7-r2 之前的版本默认拥有全部命名空间权限。

CCE 的权限可以从使用的阶段分为两个阶段来看，第一个阶段是创建和管理集群的权限，也就是拥有创建/删除集群、节点等资源的权限。第二个阶段是使用集群 Kubernetes 资源（如工作负载、Service 等）的权限。

图17-1 权限示例图



清楚了集群权限和命名空间权限后，您就可以通过这两步授权，做到精细化的权限控制。

集群权限（IAM 授权）与命名空间权限（Kubernetes RBAC 授权）的关系

拥有不同集群权限（IAM 授权）的用户，其拥有的命名空间权限（Kubernetes RBAC 授权）不同。下表给出了不同用户拥有的命名空间权限详情。

表17-1 不同用户拥有的命名空间权限

用户类型	1.13 及以上版本的集群
拥有 Tenant Administrator 权限的用户（例如账号）	全部命名空间权限
拥有 CCE Administrator 权限的 IAM 用户	全部命名空间权限
拥有 CCE FullAccess 或者 CCE ReadOnlyAccess 权限的 IAM 用户	按 Kubernetes RBAC 授权
拥有 Tenant Guest 权限的 IAM 用户	按 Kubernetes RBAC 授权

kubectl 权限说明

您可以通过 kubectl 访问集群的 Kubernetes 资源，那 kubectl 拥有哪些 Kubernetes 资源的权限呢？

kubectl 访问 CCE 集群是通过集群上生成的配置文件（kubeconfig.json）进行认证，kubeconfig.json 文件内包含用户信息，CCE 根据用户信息的权限判断 kubectl 有权限访问哪些 Kubernetes 资源。即哪个用户获取的 kubeconfig.json 文件，kubeconfig.json 就拥

有哪个用户的信息，这样使用 kubectl 访问时就拥有这个用户的权限。而用户拥有的权限就是上表所示的权限。

IAM 支持的授权项

策略包含系统策略和自定义策略，如果系统策略不满足授权要求，管理员可以创建自定义策略，并通过给用户组授予自定义策略来进行精细的访问控制。策略支持的操作与 API 相对应，授权项列表说明如下：

- 权限：允许或拒绝某项操作。
- 对应 API 接口：自定义策略实际调用的 API 接口。
- 授权项：自定义策略中支持的 Action，在自定义策略中的 Action 中写入授权项，可以实现授权项对应的权限功能。
- 依赖的授权项：部分 Action 存在对其他 Action 的依赖，需要将依赖的 Action 同时写入授权项，才能实现对应的权限功能。
- IAM 项目(Project)/企业项目(Enterprise Project)：自定义策略的授权范围，包括 IAM 项目与企业项目。授权范围如果同时支持 IAM 项目和企业项目，表示此授权项对应的自定义策略，可以在 IAM 和企业管理两个服务中给用户组授权并生效。如果仅支持 IAM 项目，不支持企业项目，表示仅能在 IAM 中给用户组授权并生效，如果在企业管理中授权，则该自定义策略不生效。

说明

“√”表示支持，“x”表示暂不支持。

云容器引擎（CCE）支持的自定义策略授权项如下所示：

表17-2 Cluster

权限	对应 API 接口	授权项（Action）	IAM 项目(Project)	企业项目(Enterprise Project)
获取指定项目下的集群	GET /api/v3/projects/{project_id}/clusters	cce:cluster:list	√	√
获取指定的集群	GET /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:get	√	√
创建集群	POST /api/v3/projects/{project_id}/clusters	cce:cluster:create	√	√
更新指定的集群	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:update	√	√
删除集群	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:delete	√	√
升级集群	POST	cce:cluster:up	√	√

权限	对应 API 接口	授权项（Action）	IAM 项目 (Project)	企业项目 (Enterprise Project)
	/api/v2/projects/:projectid/clusters/:clusterid/upgrade	grade		
唤醒集群	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/operation/awake	cce:cluster:start	√	√
休眠集群	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/operation/hibernate	cce:cluster:stop	√	√
变更集群规格	POST /api/v2/projects/{project_id}/clusters/:clusterid/resize	cce:cluster:resize	√	√
获取集群证书	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/clustercert	cce:cluster:get	√	√

表17-3 Node

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
获取集群下所有节点	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes	cce:node:list	√	√
获取指定的节点	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes/{node_id}	cce:node:get	√	√
创建节点	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes	cce:node:create	√	√
更新指定的节点	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes/{node_id}	cce:node:update	√	√
删除节点	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}/node	cce:node:delete	√	√

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
	s/{node_id}			

表17-4 Job

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
获取任务信息	GET /api/v3/projects/{project_id}/jobs/{job_id}	cce:job:get	√	√
列出所有任务	GET /api/v2/projects/{project_id}/jobs	cce:job:list	√	√
删除所有任务或删除单个任务	DELETE /api/v2/projects/{project_id}/jobs DELETE /api/v2/projects/{project_id}/jobs/{job_id}	cce:job:delete	√	√

表17-5 Nodepool

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
获取集群下所有节点池	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools	cce:nodepool:list	√	√
获取节点池	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools/{nodepool_id}	cce:nodepool:get	√	√
创建节点池	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools	cce:nodepool:create	√	√
更新节点池信息	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}/nod	cce:nodepool:update	√	√

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
	epools/{nodepool_id}			
删除节点池	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools/{nodepool_id}	cce:nodepool:delete	√	√

表17-6 Chart

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
更新模板	PUT /v2/charts/{id}	cce:chart:update	√	×
上传模板	POST /v2/charts	cce:chart:upload	√	×
列出所有模板	GET /v2/charts	cce:chart:list	√	×
获取模板信息	GET /v2/charts/{id}	cce:chart:get	√	×
删除模板	DELETE /v2/charts/{id}	cce:chart:delete	√	×

表17-7 Release

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
更新升级模板实例	PUT /v2/releases/{name}	cce:release:update	√	√
列出所有模板实例	GET /v2/releases	cce:release:list	√	√
创建模板实例	POST /v2/releases	cce:release:create	√	√
获取模板实例信息	GET /v2/releases/{name}	cce:release:get	√	√
删除模板实	DELETE	cce:release:d	√	√

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
例	/v2/releases/{name}	elete		

表17-8 Storage

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
创建 PersistentVolumeClaim	POST /api/v1/namespaces/{namespace}/cloudpersistentvolumeclaims	cce:storage:create	√	√
删除 PersistentVolumeClaim	DELETE /api/v1/namespaces/{namespace}/cloudpersistentvolumeclaims/{name}	cce:storage:delete	√	√
列出所有磁盘	GET /storage/api/v1/namespaces/{namespace}/listvolumes	cce:storage:list	√	√

表17-9 Addon

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
创建插件实例	POST /api/v3/addons	cce:addonInstance:create	√	√
获取插件实例	GET /api/v3/addons/{id}?cluster_id={cluster_id}	cce:addonInstance:get	√	√
列出所有插件实例	GET /api/v3/addons?cluster_id={cluster_id}	cce:addonInstance:list	√	√
删除插件实例	DELETE /api/v3/addons/{id}?cluster_id={cluster_id}	cce:addonInstance:delete	√	√
更新升级插件实例	PUT /api/v3/addons/{id}	cce:addonInstance:update	√	√

表17-10 Quota

权限	对应 API 接口	授权项	IAM 项目 (Project)	企业项目 (Enterprise Project)
查询配额详情	GET /api/v3/projects/{project_id}/quotas	cce:quota:get	√	√

17.2 集群权限（IAM 授权）

CCE 集群权限是基于 IAM **系统策略**和**自定义策略**的授权，可以通过用户组功能实现 IAM 用户的授权。

⚠ 注意

集群权限仅针对与集群相关的资源（如集群、节点等）有效，您必须确保同时配置了命名空间权限，才能有操作 Kubernetes 资源（如工作负载、Service 等）的权限。

前提条件

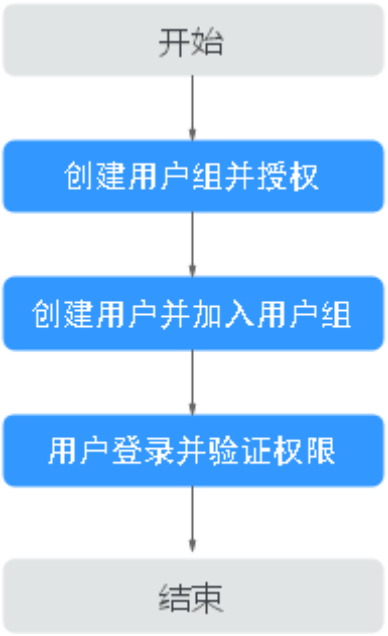
- 给用户组授权之前，请您了解用户组可以添加的 CCE 系统策略，并结合实际需求进行选择，CCE 支持的系统策略及策略间的对比。
- 拥有 Security Administrator（IAM 除切换角色外所有权限）权限的用户（如账号默认拥有此权限），才能看见 CCE 控制台权限管理页面当前用户组及用户组所拥有的权限。

配置说明

CCE 控制台“权限管理 > 集群权限”页面中创建用户组和具体权限设置均是跳转到 IAM 控制台进行具体操作，设置完后在集群权限页面能看到用户组所拥有的权限。本章节描述操作直接以 IAM 中操作为主，不重复介绍在 CCE 控制台如何跳转。

示例流程

图17-2 给用户授予 CCE 权限流程



1. 创建用户组并授权。

   在 IAM 控制台创建用户组，并授予 CCE 权限，例如 CCEReadOnlyAccess。

   说明

   CCE 服务按区域部署，在 IAM 控制台授予 CCE 权限时请选择“区域级项目”。

2. 创建用户并加入用户组。

   在 IAM 控制台创建用户，并将其加入 1 中创建的用户组。

3. 用户登录并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择云容器引擎，进入 CCE 主界面尝试购买集群，如果无法无法成功操作（假设当前权限仅包含 CCEReadOnlyAccess），表示“CCEReadOnlyAccess”已生效。
   - 在“服务列表”中选择除云容器引擎外（假设当前策略仅包含 CCEReadOnlyAccess）的任一服务，若提示权限不足，表示“CCEReadOnlyAccess”已生效。

系统角色

角色是 IAM 最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。

IAM 中预置的 CCE 系统角色为 **CCEAdministrator**，给用户组授予该系统角色权限时，必须同时勾选该角色依赖的其他策略才会生效，例如 Tenant Guest、Server

Administrator、ELB Administrator、OBS Administrator、SFS Administrator、SWR Admin、APM FullAccess。

系统策略

IAM 中预置的 CCE 系统策略当前包含 **CCEFullAccess** 和 **CCEReadOnlyAccess** 两种策略：

- **CCE FullAccess**：系统策略，CCE 服务集群相关资源的普通操作权限，不包括集群（启用 Kubernetes RBAC 鉴权）的命名空间权限，不包括委托授权、生成集群证书等管理员角色的特权操作。
- **CCE ReadOnlyAccess**：系统策略，CCE 服务集群相关资源的只读权限，不包括集群（启用 Kubernetes RBAC 鉴权）的命名空间权限。

表17-11 CCEFullAccess 策略主要权限

操作（Action）	Action 详情	说明
cce:*:*	cce:cluster:create	创建集群
	cce:cluster:delete	删除集群
	cce:cluster:update	更新集群，如后续允许集群支持 RBAC，调度参数更新等
	cce:cluster:upgrade	升级集群
	cce:cluster:start	唤醒集群
	cce:cluster:stop	休眠集群
	cce:cluster:list	查询集群列表
	cce:cluster:get	查询集群详情
	cce:node:create	添加节点
	cce:node:delete	删除节点/批量删除节点
	cce:node:update	更新节点，如更新节点名称
	cce:node:get	查询节点详情
	cce:node:list	查询节点列表
	cce:nodepool:create	创建节点池
	cce:nodepool:delete	删除节点池
	cce:nodepool:update	更新节点池信息
	cce:nodepool:get	获取节点池
	cce:nodepool:list	列出集群的所有节点池

操作（Action）	Action 详情	说明
	cce:release:create	创建模板实例
	cce:release:delete	删除模板实例
	cce:release:update	更新升级模板实例
	cce:job:list	查询任务列表（集群层面的 job）
	cce:job:delete	删除任务/批量删除任务（集群层面的 job）
	cce:job:get	查询任务详情（集群层面的 job）
	cce:storage:create	创建存储
	cce:storage:delete	删除存储
	cce:storage:list	列出所有磁盘
	cce:addonInstance:create	创建插件实例
	cce:addonInstance:delete	删除插件实例
	cce:addonInstance:update	更新升级插件实例
	cce:addonInstance:get	获取插件实例
	cce:addonTemplate:get	获取插件模板
	cce:addonInstance:list	列出所有插件实例
	cce:addonTemplate:list	列出所有插件模板
	cce:chart:list	列出所有模板
	cce:chart:delete	删除摸板
	cce:chart:update	更新模板
	cce:chart:upload	上传模板
	cce:chart:get	获取模板信息
	cce:release:get	获取模板实例信息
	cce:release:list	列出所有模板实例
	cce:userAuthorization:get	获取 CCE 用户授权
	cce:userAuthorization:create	创建 CCE 用户授权

操作（Action）	Action 详情	说明
ecs:*:*	-	ECS（弹性云主机）服务的所有权限。
evs:*:*	-	EVS（云硬盘）的所有权限。 可以将云硬盘挂载到云主机，并可以随时扩容云硬盘容量
vpc:*:*	-	VPC（虚拟私有云，包含二代 ELB）的所有权限。 创建的集群需要运行在虚拟私有云中，创建命名空间时，需要创建或关联 VPC，创建在命名空间的容器都运行在 VPC 之内。
sfs:*:get*	-	SFS（弹性文件存储服务）资源详情的查看权限。
sfs:shares:ShareAction	-	SFS（弹性文件存储服务）资源的扩容共享。
aom:*:get	-	AOM（应用运维管理）资源详情的查看权限。
aom:*:list	-	AOM（应用运维管理）资源列表的查看权限。
aom:autoScalingRule:*	-	AOM（应用运维管理）自动扩缩容规则的所有操作权限。
apm:icmgr:*	-	APM（应用性能管理服务）操作 ICAgent 权限。
lts:*:*	-	LTS（云日志服务）的所有权限。

表17-12 CCEReadOnlyAccess 策略主要权限

操作（Action）	操作（Action）	说明
cce:*:get	cce:cluster:get	查询集群详情
	cce:node:get	查询节点详情
	cce:job:get	查询任务详情（集群层面的 job）
	cce:addonInstance:get	获取插件实例
	cce:addonTemplate:get	获取插件模板
	cce:chart:get	获取模板信息

操作（Action）	操作（Action）	说明
	cce:nodepool:get	获取节点池
	cce:release:get	获取模板实例信息
	cce:userAuthorization:get	获取 CCE 用户授权
cce:*:list	cce:cluster:list	查询集群列表
	cce:node:list	查询节点列表
	cce:job:list	查询任务列表（集群层面的 job）
	cce:addonInstance:list	列出所有插件实例
	cce:addonTemplate:list	列出所有插件模板
	cce:chart:list	列出所有模板
	cce:nodepool:list	列出集群的所有节点池
	cce:release:list	列出所有模板实例
	cce:storage:list	列出所有磁盘
cce:kubernetes:*	-	操作所有 kubernetes 资源。
ecs:*:get	-	ECS（弹性云主机）所有资源详情的查看权限。 CCE 中的一个节点就是具有多个云硬盘的一台弹性云主机
ecs:*:list	-	ECS（弹性云主机）所有资源列表的查看权限。
bms:*:get*	-	BMS（物理机）所有资源详情的查看权限。
bms:*:list	-	BMS（物理机）所有资源列表的查看权限。
evs:*:get	-	EVS（云硬盘）所有资源详情的查看权限。可以将云硬盘挂载到云主机，并可以随时扩容云硬盘容量
evs:*:list	-	EVS（云硬盘）所有资源列表的查看权限。
evs:*:count	-	-
vpc:*:get	-	VPC（虚拟私有云，包含二代 ELB）所有资源详情的查看权限。

操作（Action）	操作（Action）	说明
		创建的集群需要运行在虚拟私有云中，创建命名空间时，需要创建或关联 VPC，创建在命名空间的容器都运行在 VPC 之内
vpc:*:list	-	VPC（虚拟私有云，包含二代 ELB）所有资源列表的查看权限。
sfs:*:get*	-	SFS（弹性文件服务）服务所有资源详情的查看权限。
sfs:shares:ShareAction	-	SFS（弹性文件服务）资源的扩容共享。
aom:*:get	-	AOM（应用运维管理）服务所有资源详情的查看权限。
aom:*:list	-	AOM（应用运维管理）服务所有资源列表的查看权限。
aom:autoScalingRule:*	-	AOM（应用运维管理）服务自动扩缩容规则的所有操作权限。
lts:*:get	-	LTS（云日志服务）的所有资源详情的查看权限。
lts:*:list	-	LTS（云日志服务）的所有资源列表的查看权限。

CCE 集群权限与企业项目

CCE 支持以集群为粒度，基于企业项目维度进行资源管理以及权限分配。

如下事项需特别注意：

- IAM 项目是基于资源的物理隔离进行管理，而企业项目则是提供资源的全局逻辑分组，更符合企业实际场景，并且支持基于企业项目维度的 IAM 策略管理，因此推荐您使用企业项目。
- IAM 项目与企业项目共存时，IAM 将优先匹配 IAM 项目策略、未决则匹配企业项目策略。
- CCE 集群基于已有基础资源（VPC）创建集群、节点时，请确保 IAM 用户在已有资源的企业项目下有相关权限，否则可能导致集群或者节点创建失败。

CCE 集群权限与 IAM RBAC

CCE 兼容 IAM 传统的系统角色进行权限管理，建议您切换使用 IAM 的细粒度策略，避免设置过于复杂或不必要的权限管理场景。

CCE 当前支持的角色如下：

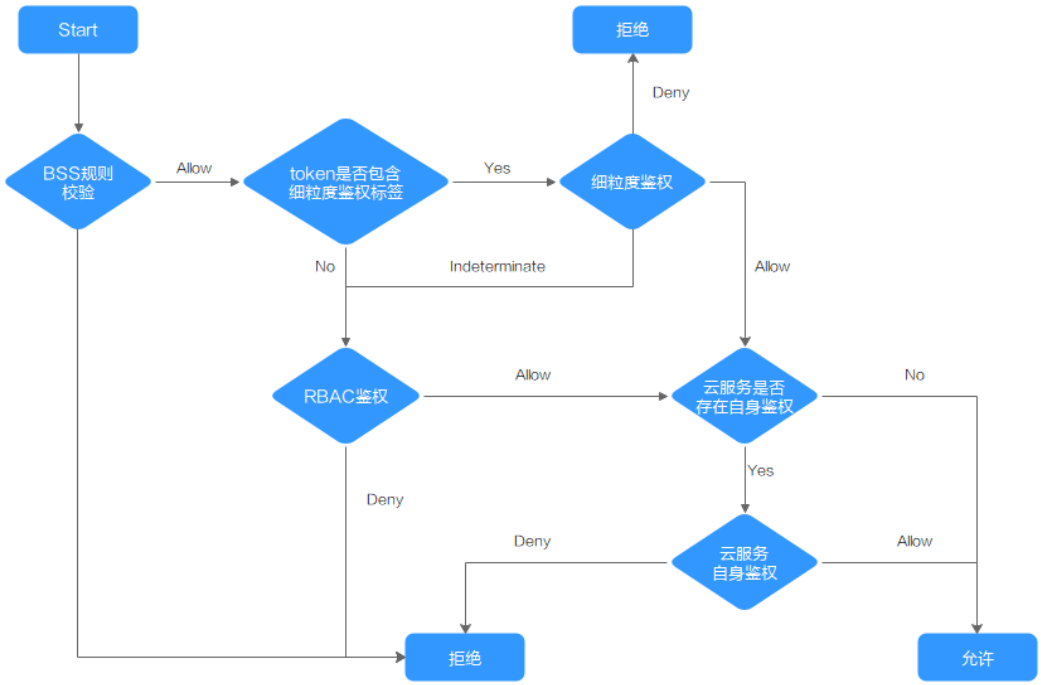
- IAM 的基础角色：
  - te_admin（Tenant Administrator）：可以调用除 IAM 外所有服务的所有 API。
  - readonly（Tenant Guest）：可以调用除 IAM 外所有服务的只读权限的 API。
- CCE 的自定义管理员角色：CCE Administrator。

说明

- Tenant Administrator、Tenant Guest 是特殊的 IAM 系统角色，当配置任意系统或自定义策略后，Tenant Administrator、Tenant Guest 将以系统策略形式生效，用于兼容 IAM RBAC 和 ABAC 场景。
- 如果用户有 Tenant Administrator 或者 CCE Administrator 的系统角色，则此用户拥有 Kubernetes RBAC 的 cluster-admin 权限，在集群创建后不可移除。

  如果用户为集群创建者，则默认被授权 Kubernetes RBAC 的 cluster-admin 权限，此项权限可以在集群创建后被手动移除：
- 方式 1：权限管理 - 命名空间权限 - 移除 cluster-creator。
- 方式 2：通过 API 或者 kubectl 删除资源，ClusterRoleBinding：cluster-creator。

RBAC 与 IAM 策略共存时，CCE 开放 API 或 Console 操作的后端鉴权逻辑如下：



注意

CCE 部分接口由于涉及命名空间权限或关键操作，需要特殊权限：

clusterCert 获取集群 k8s kubeconfig: cceadm/teadmin

17.3 命名空间权限（Kubernetes RBAC 授权）

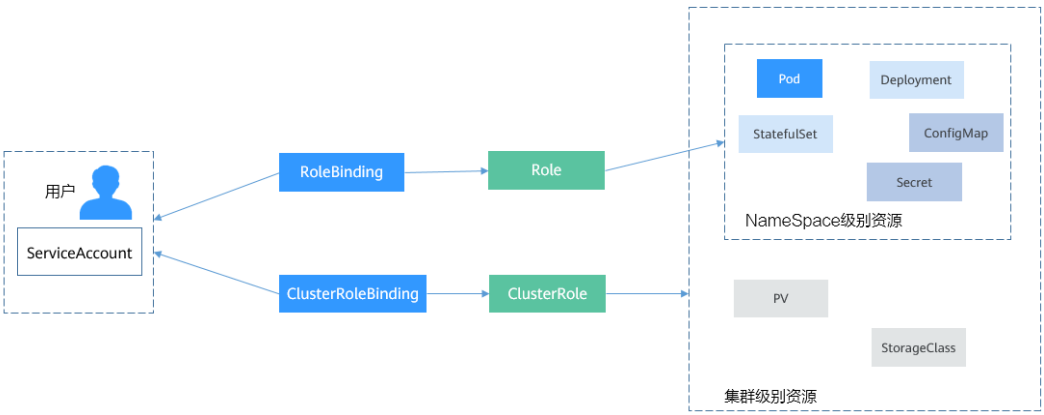
命名空间权限（kubernetes RBAC 授权）

命名空间权限是基于 Kubernetes RBAC 能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同 Kubernetes 资源的权限。Kubernetes RBAC API 定义了四种类型：Role、ClusterRole、RoleBinding 与 ClusterRoleBinding，这四种类型之间的关系和简要说明如下：

- Role：角色，其实是定义一组对 Kubernetes 资源（命名空间级别）的访问规则。
- RoleBinding：角色绑定，定义了用户和角色的关系。
- ClusterRole：集群角色，其实是定义一组对 Kubernetes 资源（集群级别，包含全部命名空间）的访问规则。
- ClusterRoleBinding：集群角色绑定，定义了用户和集群角色的关系。

Role 和 ClusterRole 指定了可以对哪些资源做哪些动作，RoleBinding 和 ClusterRoleBinding 将角色绑定到特定的用户、用户组或 ServiceAccount 上。如下图所示。

图17-3 角色绑定



在 CCE 控制台可以授予用户或用户组命名空间权限，可以对某一个命名空间或全部命名空间授权，CCE 控制台默认提供如下 ClusterRole。

- view（只读权限）：对全部或所选命名空间下大多数资源的只读权限。
- edit（开发权限）：对全部或所选命名空间下多数资源的读写权限。当配置在全部命名空间时能力与运维权限一致。
- admin（运维权限）：对全部命名空间下大多数资源的读写权限，对节点、存储卷，命名空间和配额管理的只读权限。
- cluster-admin（管理员权限）：对全部命名空间下所有资源的读写权限。

集群权限（IAM 授权）与命名空间权限（Kubernetes RBAC 授权）的关系

拥有不同集群权限（IAM 授权）的用户，其拥有的命名空间权限（Kubernetes RBAC 授权）不同。下表给出了不同用户拥有的命名空间权限详情。

表17-13 不同用户拥有的命名空间权限

用户类型	1.13 及以上版本的集群
拥有 Tenant Administrator 权限的用户（例如账号）	全部命名空间权限
拥有 CCE Administrator 权限的 IAM 用户	全部命名空间权限
拥有 CCE FullAccess 或者 CCE ReadOnlyAccess 权限的 IAM 用户	按 Kubernetes RBAC 授权
拥有 Tenant Guest 权限的 IAM 用户	按 Kubernetes RBAC 授权

注意事项

- Kubernetes RBAC 的授权能力支持 1.11.7-r2 及以上版本集群。若需使用 RBAC 功能请将集群升级至 1.11.7-r2 或以上版本。
- 任何用户创建 1.11.7-r2 或以上版本集群后，CCE 会自动为该用户添加该集群的所有命名空间的 cluster-admin 权限，也就是说该用户允许对集群以及所有命名空间中的全部资源进行完全控制。
- 拥有 Security Administrator（IAM 除切换角色外所有权限）权限的用户（如账号所在的 admin 用户组默认拥有此权限），才能在 CCE 控制台命名空间权限页面进行授权操作。

配置命名空间权限（控制台）

CCE 中的命名空间权限是基于 Kubernetes RBAC 能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同 Kubernetes 资源的权限。

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“权限管理”。

步骤 2 在右边下拉列表中选择要添加权限的集群。

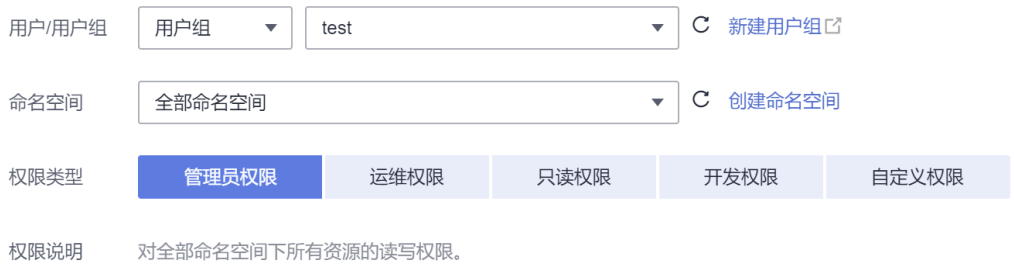
步骤 3 在右上角单击“添加权限”，进入添加授权页面。

步骤 4 在添加权限页面，确认集群名称，选择该集群下要授权使用的命名空间，例如选择“全部命名空间”，选择要授权的用户或用户组，再选择具体权限。

说明

对于没有 IAM 权限的用户，给其他用户和用户组配置权限时，无法选择用户和用户组，此时支持填写用户 ID 或用户组 ID 进行配置。

图17-4 配置命名空间权限



其中自定义权限可以根据需要自定义，选择自定义权限后，在自定义权限一行右侧单击新建自定义权限，在弹出的窗口中填写名称并选择规则。创建完成后，在添加权限的自定义权限下拉框中可以选择。

图17-5 自定义权限

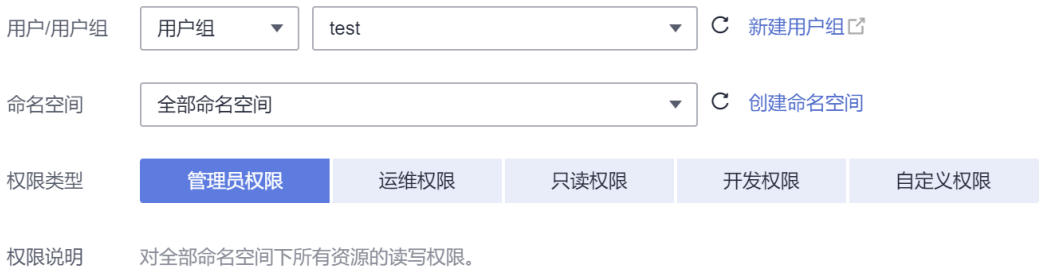
步骤 5 单击“确定”。

----结束

示例：授予集群全部权限（cluster-admin）

集群全部权限可以使用 cluster-admin 权限，cluster-admin 包含集群级别资源（PV、StorageClass 等）的权限。

图17-6 授予集群全部权限（cluster-admin）



如果使用 kubectl 查看可以看到创建了一个 ClusterRoleBinding，将 cluster-admin 和 cce-role-group 这个用户组绑定了起来。

```
# kubectl get clusterrolebinding
NAME                                                    ROLE
AGE
clusterrole_cluster-admin_group0c96fad22880f32a3f84c009862af6f7
ClusterRole/cluster-admin     61s

# kubectl get clusterrolebinding clusterrole_cluster-
admin_group0c96fad22880f32a3f84c009862af6f7 -oyaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    CCE.com/IAM: "true"
  creationTimestamp: "2021-06-23T09:15:22Z"
  name: clusterrole cluster-admin group0c96fad22880f32a3f84c009862af6f7
  resourceVersion: "36659058"
  selfLink:
/apis/rbac.authorization.k8s.io/v1/clusterrolebindings/clusterrole cluster-
admin group0c96fad22880f32a3f84c009862af6f7
  uid: d6cd43e9-b4ca-4b56-bc52-e36346fc1320
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7
```

使用被授予用户连接用户连接集群，如果能正常查询 PV、StorageClass 的信息，则说明权限配置正常。

```
# kubectl get pv
No resources found
# kubectl get sc
NAME              PROVISIONER                  RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
csi-disk          everest-csi-provisioner        Delete         Immediate
true                 75d
csi-disk-topology   everest-csi-provisioner        Delete
```

```
WaitForFirstConsumer   true                 75d
csi-nas           everest-csi-provisioner       Delete        Immediate
true                75d
csi-obs           everest-csi-provisioner       Delete        Immediate
false               75d
csi-sfsturbo      everest-csi-provisioner       Delete        Immediate
true                75d
```

17.4 示例：某部门权限设计及配置

概述

随着容器技术的快速发展，原有的分布式任务调度模式正在被基于 Kubernetes 的技术架构所取代。云容器引擎（Cloud Container Engine，简称 CCE）是高度可扩展的、高性能的企业级 Kubernetes 集群，支持社区原生应用和工具。借助云容器引擎，您可以在云上轻松部署、管理和扩展容器化应用程序，快速高效的将微服务部署在云端。

为方便企业中的管理人员对集群中的资源权限进行管理，CCE 后台提供了多种维度的细粒度权限策略和管理方式。CCE 的权限管理包括“集群权限”和“命名空间权限”两种能力，分别从集群和命名空间两个层面对用户组或用户进行细粒度授权，具体解释如下：

- **集群权限**：是基于 IAM 系统策略的授权，可以让用户组拥有“集群管理”、“节点管理”、“节点池管理”、“模板市场”、“插件管理”权限。
- **命名空间权限**：是基于 Kubernetes RBAC 能力的授权，可以让用户或用户组拥有 Kubernetes 资源的权限，如“工作负载”、“网络管理”、“存储管理”、“命名空间”等的权限。

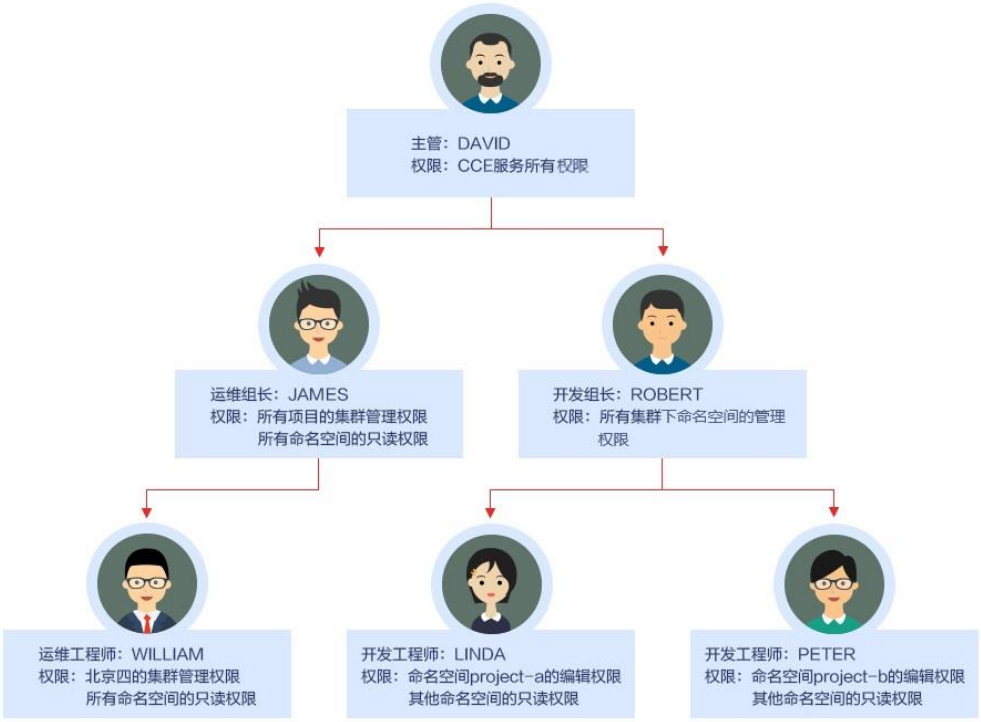
基于 IAM 系统策略的“集群权限”与基于 Kubernetes RBAC 能力的“命名空间权限”，两者是完全独立的，互不影响，但要配合使用。同时，为用户组设置的权限将作用于用户组下的全部用户。当给用户或用户组添加多个权限时，多个权限会同时生效（取并集）。

权限设计

下面我们以一个公司为例进行介绍。

通常一个公司中有多个部门或项目，每个部门又有多个成员，所以在配置权限前需要先进行详细设计，并在设置权限之前提前为每个成员创建用户名，便于后续对用户进行用户组归属和权限设置。

下图为某公司某部门的组织架构图和相关人员的权限设计，我们将按照该设计对每个角色的权限设置进行演示：



主管：DAVID

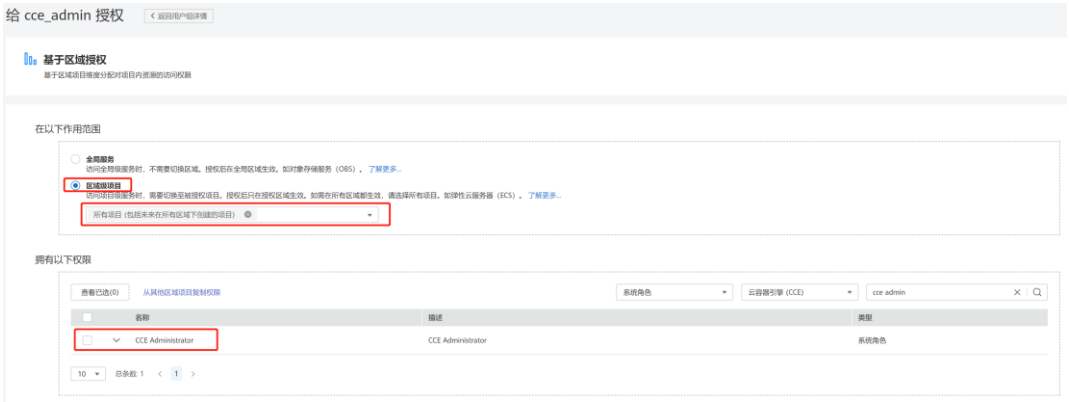
用户“DAVID”为该公司某部门的主管，根据权限设计我们需要为其配置 CCE 服务的所有权限（包括集群权限和命名空间权限），因此需要在统一身份认证服务 IAM 中单独为 DAVID 创建用户组“cce-admin”，并配置所有项目的权限：“CCE Administrator”，这样主管 DAVID 的权限就配置好了。

说明

CCE Administrator：CCE 的管理员权限，拥有该服务的所有权限，不需要再赋予其他权限。

CCE FullAccess、CCE ReadOnlyAccess：CCE 的集群管理权限，仅针对与集群相关的资源（如集群、节点）有效，您必须确保同时配置了“命名空间权限”，才能有操作 Kubernetes 资源（如工作负载、Service 等）的权限。

图17-7 为主管 DAVID 所在的用户组授权



运维组长：JAMES

用户“JAMES”为该部门的运维组长，需要设置所有项目的集群权限和所有命名空间的只读权限。

我们需要在统一身份认证服务 IAM 中先为用户“JAMES”单独创建用户组“cce-sre”并将其添加到用户组“cce-sre”中，然后为用户组“cce-sre”配置所有项目的集群权限：“CCE FullAccess”，用户组“cce-sre”便拥有了所有项目的集群管理权限，接下来还需要为其设置命名空间的只读权限。

图17-8 为运维组长 JAMES 所在的用户组授权

为所有组长和工程师添加所有集群和命名空间的只读权限

我们可以在统一身份认证服务 IAM 中再创建一个只读用户组“read_only”，然后将相关用户都添加到此用户组中。

- 两个开发工程师虽然他们不需要配置集群的管理权限，但也需要查看 CCE 控制台，因此需要有集群的只读权限才能满足需求。
- 运维工程师需要特定资源池集群的管理权限，为方便管理，这里我们先为其赋予集群的只读权限。
- 运维组长已经拥有了所有集群的管理权限，为方便管理，我们也可以将其添加到“read_only”用户组中，为其赋予集群的只读权限。

我们将 JAMES、ROBERT、WILLIAM、LINDA、PETER 五个用户都添加到用户组“read_only”中。

接下来为用户组“read_only”赋予集群的只读权限。

图17-9 为用户组赋予集群的只读权限

然后返回 CCE 控制台，为这五个用户所在的用户组“read_only”增加命名空间的只读权限，单击左侧栏目树中的“权限管理 > 命名空间权限”，为用户组“read_only”逐个赋予所有集群的只读权限。

图17-10 为用户组赋予命名空间的只读权限

设置完成后，运维组长“JAMES”就拥有了所有项目的集群管理权限和所有命名空间的只读权限，而开发组长“ROBERT”、运维工程师“WILLIAM”以及两位开发工程师“LINDA”和“PRTER”则拥有了所有集群和命名空间的只读权限。

开发组长：ROBERT

用户“ROBERT”作为开发组的组长，虽然在上一步中已经为其设置了所有集群和命名空间的只读权限，但显然还不够，还需要为其设置所有命名空间的管理权限。

因此需要再单独为其赋予所有集群下全部命名空间的管理员权限。

图17-11 为用户 ROBERT 赋予命名空间的管理权限

运维工程师：WILLIAM

运维工程师“WILLIAM”虽然也有了所有集群和命名空间的只读权限，但还需要在统一身份认证服务 IAM 中为其设置区域的集群管理权限，因此我们单独为其创建一个用户组“cce-sre-b4”，然后配置区域项目的“CCE FullAccess”。

图17-12 为用户 WILLIAM 所在的用户组配置资源池的集群管理权限

由于之前已经为其设置过所有命名空间的只读权限，所以运维工程师“WILLIAM”现在就拥有了区域的集群管理权限和所有命名空间的只读权限。

开发工程师：LINDA、PETER

“LINDA”和“PETER”是开发工程师，由于前面已经在用户组“read-only”中为两位工程师配置了集群和命名空间的只读权限，这里只需要再另外配置相应命名空间的编辑权限即可。

图17-13 为开发工程师分别配置命名空间的编辑权限

至此，该部门的所有权限就设置完成了。

17.5 CCE 控制台的权限依赖

CCE 对其他云服务有诸多依赖关系，因此在您开启 IAM 系统策略授权后，在 CCE Console 控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，详细说明如下：

- 依赖服务的权限配置均基于您已设置了 IAM 系统策略授权的 CCE FullAccess 或 CCE ReadOnlyAccess 策略权限，详细设置方法请参见集群权限（IAM 授权）。
- 1.11.7-r2 及以上版本的集群，显示情况依赖于命名空间权限的设置情况，如果没有设置命名空间权限，则无法查看集群下的资源。
  - 如果您设置了全部命名空间的 view 权限，则可以查看到对应集群的全部命名空间下的资源，但密钥（Secret）除外，密钥（Secret）需要在命名空间权限下设置 admin 或者 edit 权限才能查看。
  - CustomedHPA 与 HPA 策略需要配置命名空间 cluster-admin 权限下才能生效。
  - 如果您设置的是单一命名空间的 view 权限，则看到的只能是指定命名空间下的资源。

依赖服务的权限设置

如果 IAM 用户需要在 CCE Console 控制台拥有相应功能的查看或使用权限，请确认已经对该用户所在的用户组设置了 CCE Administrator、CCE FullAccess 或 CCE ReadOnlyAccess 策略的集群权限，再按下表增加依赖服务的角色或策略。

说明

企业项目能够实现企业不同项目间资源的分组和管理，重在资源隔离，而 IAM 可以实现细粒度授权，因此强烈推荐您使用 IAM 实现权限管理。

若您使用企业项目设置子用户权限，会有如下功能限制：

- 在 CCE 控制台，集群监控获取 AOM 监控的接口暂不支持企业项目，因此企业项目子用户将无法查看监控相关数据。

- 在 CCE 控制台，由于创建节点时的密钥对查询接口不支持企业项目，因此企业项目子用户将无法使用“密钥对”登录方式，您可以选择使用“密码”登录方式。
- 为企业项目设置 **CCE FullAccess** 权限后 ，需要在 IAM 控制台界面中再配置 **ecs:availabilityZones:list** 权限，企业项目子用户创建节点才可以正常显示并创建，否则将提示没有 ecs:availabilityZones:list 权限。

CCE 支持细粒度的权限设置，但有如下限制说明：

- AOM 不支持资源级别细粒度：当通过 IAM 集群资源细粒度设置特定资源操作权限之后，IAM 用户在 CCE 控制台的总览界面查看集群监控时，将显示非细粒度关联集群的监控信息。
- 在 IAM 页面设置 **CCE FullAccess** 或者 **CCE ReadOnlyAccess** 权限后，需要配置 **sfsturbo:*:*** 权限才能使用极速文件存储卷，否则 IAM 用户在集群下查询极速文件存储卷将失败。

表17-14 CCE Console 中依赖服务的角色或策略

Console 控制台功能	依赖服务	需配置角色/策略
总览	应用运维管理 AOM	• IAM 用户设置了 CCE Administrator 权限后，需要增加 AOM FullAccess 权限后才能访问总览中的数据图表。 • 支持设置了 IAM ReadOnlyAccess 和 CCE FullAccess 或 CCE ReadOnlyAccess 权限的 IAM 用户直接访问总览中的数据图表。
工作负载	弹性负载均衡 ELB 应用运维管理 AOM NAT 网关 NAT 对象存储服务 OBS 弹性文件服务 SFS	正常创建工作负载时不依赖其他服务的权限。 • 如果需要创建 ELB 类型的服务，需要设置 ELB FullAccess 或者 ELB Administrator 权限，以及 VPC Administrator 权限。 • 如果需要使用 Java 探针，需要设置 AOM FullAccess 权限。 • 如果需要 NAT 网关类型的服务，需要设置 NAT Gateway Administrator 权限。 • 如果使用对象存储，需要全局设置 OBS Administrator 权限。 **说明** 由于缓存的存在，对用户、用户组以及企业项目授予 OBS 相关的 RBAC 策略后，大概需要等待 13 分钟 RBAC 策略才能生效；授予 OBS 相关的系统策略后，大概需要等待 5 分钟系统策略能生效。 • 如果使用文件存储，需要设置 SFS FullAccess 权限。
集群管理	应用运维管理 AOM	• 如果需要弹性扩容权限，需要设置 AOM FullAccess 权限。

Console 控制台功能	依赖服务	需配置角色/策略
节点管理	弹性云主机 ECS	当 IAM 用户权限为 CCE Administrator 时，如果创建和删除节点，需要配置 ECS FullAccess 或 ECS Administrator 权限，以及 VPC Administrator 权限。
网络管理	弹性负载均衡 ELB NAT 网关 NAT	正常创建时不依赖其他服务的权限。 • 如果需要创建 ELB 类型的服务，需要设置 ELB FullAccess 或者 ELB Administrator 权限，以及 VPC Administrator 权限。 • 如果需要 NAT 网关类型的服务，需要设置 NAT Administrator 权限。
存储管理	对象存储服务 OBS 弹性文件服务 SFS	• 如果使用对象存储，需要全局设置 OBS Administrator 权限。 说明 由于缓存的存在，对用户、用户组以及企业项目授予 OBS 相关的 RBAC 策略后，大概需要等待 13 分钟 RBAC 策略才能生效；授予 OBS 相关的系统策略后，大概需要等待 5 分钟系统策略能生效。 • 如果使用文件存储，需要设置 SFS FullAccess 权限。 • 如果使用极速文件存储，需要设置 SFS Turbo Admin 权限 导入存储的功能需要设置 CCE Administrator 权限。
命名空间	/	无需其他依赖权限。
模板市场	/	当前仅支持帐号、设置了 CCE Administrator 权限的 IAM 用户访问。
插件管理	/	支持帐号、设置了 CCE Administrator、CCE FullAccess 或 CCE ReadOnlyAccess 等权限的 IAM 用户访问本功能。
权限管理	/	• 支持帐号访问。 • 支持设置了 CCE Administrator 和 Security Administrator（全局级策略）权限的 IAM 用户访问。 • 支持设置了 CCE FullAccess 或 CCE ReadOnlyAccess 权限的 IAM 用户访问。
配置中心	/	• 配置项 (ConfigMap)无需其他依赖权

Console 控制台功能	依赖服务	需配置角色/策略
		限。 • 密钥（Secret）需要在命名空间权限下设置 cluster-admin、admin 或者 edit 权限才能查看，依赖服务需要添加 DEW KeypairFullAccess 或者 DEW KeypairReadOnlyAccess 权限。
帮助中心	/	无需其他依赖权限。
其他服务跳转	容器镜像服务 SWR 应用运维管理 AOM	为便于您快速进入 CCE 相关服务的控制台，在 CCE 控制台增加了其他服务的跳转链接，CCE 默认没有这些服务的全部权限，如果 IAM 用户需要查看或使用其功能，请按照该服务的权限策略说明设置相应的权限策略。

17.6 Pod 安全配置

17.6.1 PodSecurityPolicy 配置

Pod 安全策略（Pod Security Policy） 是集群级别的资源，它能够控制 Pod 规约中与安全性相关的各个方面。 PodSecurityPolicy 对象定义了一组 Pod 运行时必须遵循的条件及相关字段的默认值，只有 Pod 满足这些条件才会被系统接受。

v1.17.17 版本的集群默认启用 Pod 安全策略准入控制组件，并创建名为 **psp-global** 的全局默认安全策略，您可根据自身业务需要修改全局策略（请勿直接删除默认策略），也可新建自己的 Pod 安全策略并绑定 RBAC 配置。

说明

- 除全局默认安全策略外，系统为 kube-system 命名空间下的系统组件配置了独立的 Pod 安全策略，修改 psp-global 配置不影响 kube-system 下 Pod 创建。
- 在 Kubernetes 1.25 版本中， PodSecurityPolicy 已被移除，并提供 Pod 安全性准入控制器（Pod Security Admission）作为 PodSecurityPolicy 的替代，详情请参见 Pod Security Admission 配置。

修改全局默认 Pod 安全策略

修改全局默认 Pod 安全策略前，请确保已创建 CCE 集群，并且通过 kubectl 连接集群成功。

步骤 1 执行如下命令：

kubectl edit psp psp-global

步骤 2 修改所需的参数，请参考 PodSecurityPolicy。

----结束

Pod 安全策略开放非安全系统配置示例

节点池管理中可以为相应的节点池配置 allowed-unsafe-sysctls，CCE 从 **1.17.17** 集群版本开始，需要在 pod 安全策略的 allowedUnsafeSysctls 中增加相应的配置才能生效，详情请参见 PodSecurityPolicy。

除修改全局 Pod 安全策略外，也可增加新的 Pod 安全策略，如开放 net.core.somaxconn 非安全系统配置，新增 Pod 安全策略示例参考如下：

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  name: sysctl-psp
spec:
  allowedUnsafeSysctls:
  - net.core.somaxconn
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  fsGroup:
    rule: RunAsAny
  hostIPC: true
  hostNetwork: true
  hostPID: true
  hostPorts:
  - max: 65535
    min: 0
  privileged: true
  runAsGroup:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - '*'
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sysctl-psp
rules:
  - apiGroups:
      - "*"
    resources:
      - podsecuritypolicies
    resourceNames:
      - sysctl-psp
    verbs:
      - use
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sysctl-psp
roleRef:
  kind: ClusterRole
  name: sysctl-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

恢复原始 Pod 安全策略

如果您已经修改默认 Pod 安全策略后，想恢复原始 Pod 安全策略，请执行以下操作。

步骤 1 创建一个名为 policy.yaml 的描述文件。其中，policy.yaml 为自定义名称，您可以随意命名。

vi policy.yaml

描述文件内容如下。

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: psp-global
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
    - '*'
  volumes:
    - '*'
  hostNetwork: true
  hostPorts:
    - min: 0
      max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: psp-global
rules:
  - apiGroups:
      - "*"
    resources:
      - podsecuritypolicies
    resourceNames:
      - psp-global
    verbs:
      - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp-global
roleRef:
  kind: ClusterRole
  name: psp-global
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

步骤 2 执行如下命令：

kubectl apply -f policy.yaml

----结束

17.6.2 Pod Security Admission 配置

在使用 Pod Security Admission 前，需要先了解 Kubernetes 的 Pod 安全性标准（Security Standards）。Pod 安全性标准（Security Standards） 为 Pod 定义了不同的安全性策略级别。这些标准能够让你以一种清晰、一致的方式定义如何限制 Pod 行为。而 Pod Security Admission 则是这些安全性标准的控制器，用于在创建 Pod 时执行定义好的安全限制。

Pod 安全性标准定义了三种安全性策略级别：

表17-15 Pod 安全性策略级别

策略级别（level）	描述
privileged	不受限制，通常适用于特权较高、受信任的用户所管理的系统级或基础设施级负载，例如 CNI、存储驱动等。
baseline	限制较弱但防止已知的特权提升（Privilege Escalation），通常适用于部署常用的非关键性应用负载，该策略将禁止使用 hostNetwork、hostPID 等能力。

策略级别（level）	描述
restricted	严格限制，遵循 Pod 防护的最佳实践。

Pod Security Admission 配置是命名空间级别的，控制器将会对该命名空间下 Pod 或容器中的安全上下文（Security Context）以及其他参数进行限制。其中，privileged 策略将不会对 Pod 和 Container 配置中的 securityContext 字段有任何校验，而 Baseline 和 Restricted 则会对 securityContext 字段有不同的取值要求，具体规范请参见 Pod 安全性标准（Security Standards）。

关于如何在 Pod 或容器中设置 Security Context，请参见为 Pod 或容器配置 Security Context。

Pod Security Admission 标签

Kubernetes 为 Pod Security Admission 定义了三种标签，您可以在某个命名空间中设置这些标签来定义需要使用的 Pod 安全性标准级别，但请勿在 kube-system 等系统命名空间修改 Pod 安全性标准级别，否则可能导致系统命名空间下 Pod 故障。

表17-16 Pod Security Admission 标签

隔离模式（mode）	生效对象	描述
enforce	Pod	违反指定策略会导致 Pod 无法创建。
audit	工作负载（例如 Deployment、Job 等）	违反指定策略会在审计日志（audit log）中添加新的审计事件，Pod 可以被创建。
warn	工作负载（例如 Deployment、Job 等）	违反指定策略会返回用户可见的告警信息，Pod 可以被创建。

说明

Pod 通常是通过创建 Deployment 或 Job 这类工作负载对象来间接创建的。在使用 Pod Security Admission 时，audit 或 warn 模式的隔离都将在工作负载级别生效，而 enforce 模式并不会应用到工作负载，仅在 Pod 上生效。

使用命名空间标签进行 Pod Security Admission 配置

您可以在不同的隔离模式中应用不同的策略，由于 Pod 安全性准入能力是在命名空间（Namespace）级别实现的，因此假设某个 Namespace 配置如下：

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-baseline-namespace
```

```
labels:
  pod-security.kubernetes.io/enforce: privileged
  pod-security.kubernetes.io/enforce-version: v1.25
  pod-security.kubernetes.io/audit: baseline
  pod-security.kubernetes.io/audit-version: v1.25
  pod-security.kubernetes.io/warn: restricted
  pod-security.kubernetes.io/warn-version: v1.25

  # 标签有以下两种格式：
  # pod-security.kubernetes.io/<MODE>: <LEVEL>
  # pod-security.kubernetes.io/<MODE>-version: <VERSION>
  # audit 和 warn 模式的作用主要在于提供相应信息供用户排查负载违反了哪些安全行为
```

命名空间的标签用来表示不同的模式所应用的安全策略级别，存在以下两种格式：

- pod-security.kubernetes.io/<MODE>: <LEVEL>
  - <MODE>：隔离模式必须是 enforce、audit 或 warn 之一。
  - <LEVEL>：安全性策略级别必须是 privileged、baseline 或 restricted。
- pod-security.kubernetes.io/<MODE>-version: <VERSION>
  该标签为可选，可以将安全性策略锁定到 Kubernetes 版本号。
  - <MODE>：隔离模式必须是 enforce、audit 或 warn 之一。
  - <VERSION>：Kubernetes 版本号。例如 v1.25，也可以使用 latest。

若在上述 Namespace 中部署 Pod，则会有以下安全性限制：

1. 设置了 enforce 隔离模式对应的策略为 privileged，将会跳过 enforce 阶段的校验。
2. 设置了 audit 隔离模式对应的策略为 baseline，将会校验 baseline 策略相关限制，即如果 Pod 或 Container 违反了该策略，审计日志中将添加相应事件。
3. 设置了 warn 隔离模式对应的策略为 restricted，将会校验 restricted 策略相关限制，即如果 Pod 或 Container 违反了该策略，用户将会在创建 pod 时收到告警信息。

从 PodSecurityPolicy 迁移到 Pod Security Admission

如您在 1.25 之前版本的集群中使用了 PodSecurityPolicy，且需要在 1.25 及以后版本集群中继续使用 Pod Security Admission 来替代 PodSecurityPolicy 的用户，请参见从 PodSecurityPolicy 迁移到内置的 Pod Security Admission。

须知

1. 由于 Pod Security Admission 仅支持三种隔离模式，因此灵活性相比于 PodSecurityPolicy 较差，部分场景下需要用户自行定义验证准入 Webhook 来实施更精准的策略。
2. 由于 PodSecurityPolicy 具有变更能力，而 Pod Security Admission 并不具备该能力，因此之前依赖该能力的用户需要自行定义变更准入 Webhook 或修改 Pod 中的 securityContext 字段。
3. PodSecurityPolicy 允许为不同的服务账号（Service Account）绑定不同策略（Kubernetes 社区不建议使用该能力）。如果您有使用该能力的诉求，在迁移至 Pod Security Admission 后，需要自行定义第三方 Webhook。
4. 请勿将 Pod Security Admission 能力应用于 kube-system、kube-public 和 kube-node-lease 等一些 CCE 组件部署的 Namespace 中，否则会导致 CCE 组件、插件功能异常。

17.7 ServiceAccount Token 安全性提升说明

Kubernetes 1.21 以前版本的集群中，Pod 中获取 Token 的形式是通过挂载 ServiceAccount 的 Secret 来获取 Token，这种方式获得的 Token 是永久的。该方式在 1.21 及以上的版本中不再推荐使用，并且根据社区版本迭代策略，在 1.25 及以上版本的集群中，ServiceAccount 将不会自动创建对应的 Secret。

Kubernetes 1.21 及以上版本的集群中，直接使用 TokenRequest API 获得 Token，并使用投射卷（Projected Volume）挂载到 Pod 中。使用这种方法获得的 Token 具有固定的生命周期（默认有效期为 1 小时），在到达有效期之前，Kubelet 会刷新该 Token，保证 Pod 始终拥有有效的 Token，并且当挂载的 Pod 被删除时这些 Token 将自动失效。该方式通过 BoundServiceAccountTokenVolume 特性实现，能够提升服务账号（ServiceAccount）Token 的安全性，Kubernetes 1.21 及以上版本的集群中会默认开启。

为了帮助用户平滑过渡，社区默认将 Token 有效时间延长为 1 年，1 年后 Token 失效，不具备证书 reload 能力的 client 将无法访问 APIServer，建议使用低版本 Client 的用户尽快升级至高版本，否则业务将存在故障风险。

如果用户使用版本过低的 Kubernetes 客户端（Client），由于低版本 Client 并不具备证书轮转能力，会存在证书轮转失效的风险。K8s 社区默认具有证书轮转能力的 Client 版本如下：

- Go: >= v0.15.7
- Python: >= v12.0.0
- Java: >= v9.0.0
- Javascript: >= v0.10.3
- Ruby: master branch
- Haskell: v0.3.0.0
- C#: >= 7.0.5

官方说明请参见：https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/1205-bound-service-account-tokens

说明

如果您在业务中需要一个永不过期的 Token，您也可以选择手动管理 ServiceAccount 的 Secret。尽管存在手动创建永久 ServiceAccount Token 的机制，但还是推荐使用 TokenRequest 的方式使用短期的 Token，以提高安全性。

排查方案

CCE 提供以下排查方式供用户参考（CCE 1.21 及以上版本的集群均涉及）：

4. 排查集群中使用的插件版本。
   - 若用户集群中有使用 2.23.34 及以下版本 Prometheus 插件，则需升级至 2.23.34 以上版本。
   - 若用户集群中有使用 1.15.0 及以下版本 npd 插件，则需升级至最新版本。
5. 通过 kubectl 连接集群，并通过 **kubectl get --raw "/metrics" | grep stale** 查询，可以看到一个名为 serviceaccount_stale_tokens_total 的指标。

   如果该值大于 0，则表示当前集群可能存在某些负载正在使用过低的 client-go 版本情况，此时请您排查自己部署的应用中是否有该情况出现。如果存在，则尽快将 client-go 版本升级至社区指定的版本之上（至少不低于 CCE 集群的两个大版本，如部署在 1.23 集群上的应用需要使用 1.19 版本以上的 Kubernetes 依赖库）。

```
[root@ ]# kubectl get --raw "/metrics" | grep stale
# HELP serviceaccount_stale_tokens_total [ALPHA] Cumulative stale projected service account tokens used
# TYPE serviceaccount_stale_tokens_total counter
serviceaccount_stale_tokens_total 52
```

17.8 系统委托说明

由于 CCE 在运行中对计算、存储、网络以及监控等各类云服务资源都存在依赖关系，因此当您首次登录 CCE 控制台时，CCE 将自动请求获取当前区域下的云资源权限，从而更好地为您提供服务。服务权限包括：

- 计算类服务

  CCE 集群创建节点时会关联创建云主机，因此需要获取访问弹性云主机、物理机的权限。
- 存储类服务

  CCE 支持为集群下节点和容器挂载存储，因此需要获取访问云硬盘、弹性文件、对象存储等服务的权限。
- 网络类服务

  CCE 支持集群下容器发布为对外访问的服务，因此需要获取访问虚拟私有云、弹性负载均衡等服务的权限。
- 容器与监控类服务

  CCE 集群下容器支持镜像拉取、监控和日志分析等功能，需要获取访问容器镜像等服务的权限。

当您同意授权后，CCE 将在 IAM 中自动创建账号委托，将帐号内的其他资源操作权限委托给 CCE 服务进行操作。

CCE 自动创建的委托如下：

- cce_admin_trust
- cce_cluster_agency

cce_admin_trust 委托说明

cce_admin_trust 委托具有 Tenant Administrator 权限。Tenant Administrator 拥有除 IAM 管理外的全部云服务管理员权限，用于对 CCE 所依赖的其他云服务资源进行调用，且该授权仅在当前区域生效。

如果您在多个区域中使用 CCE 服务，则需在每个区域中分别申请云资源权限。您可前往“IAM 控制台 > 委托”页签，单击“cce_admin_trust”查看各区域的授权记录。

说明

由于 CCE 对其他云服务有许多依赖，如果没有 Tenant Administrator 权限，可能会因为某个服务权限不足而影响 CCE 功能的正常使用。因此在使用 CCE 服务期间，请不要自行删除或者修改“cce_admin_trust”委托。

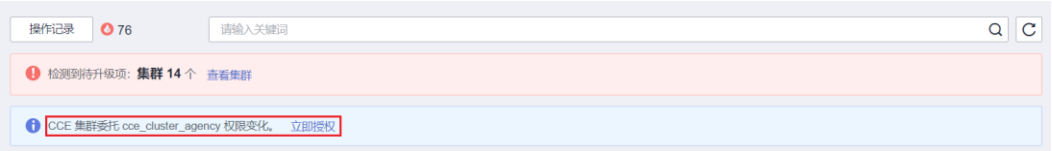
cce_cluster_agency 委托说明

cce_cluster_agency 委托没有 Tenant Administrator 系统角色，只包含 CCE 组件需要的云服务资源操作权限，用于生成 CCE 集群中组件使用的临时访问凭证。

说明

- cce_cluster_agency 委托仅支持 1.21 及以上版本新建的集群。
- 创建 cce_cluster_agency 委托时将会自动创建名为“CCE cluster policies”的自定义策略，请勿删除该策略。

若当前 cce_cluster_agency 委托的权限与 CCE 期望的权限不同时，控制台会提示权限变化，需要您重新授权。



以下场景中，可能会出现 cce_cluster_agency 委托重新授权：

- CCE 组件依赖的权限可能会随版本变动而发生变化。例如新增组件需要依赖新的权限，CCE 将会更新期望的权限列表，此时需要您重新为 cce_cluster_agency 委托授权。

当您手动修改了 cce_cluster_agency 委托的权限时，该委托中拥有的权限与 CCE 期望的权限不相同，此时也会出现重新授权的提示。若您重新进行授权，该委托中手动修改的权限可能会失效。

18 云审计

18.1 云审计服务支持的 CCE 操作列表

CCE 通过云审计服务（Cloud Trace Service，简称 CTS）为您提供云服务资源的操作记录，记录内容包括您从云管理控制台或者开放 API 发起的云服务资源操作请求以及每次请求的结果，供您查询、审计和回溯使用。

表18-1 云审计服务支持的 CCE 操作列表

操作名称	资源类型	事件名称
创建用户委托	集群	createUserAgencies
创建集群	集群	createCluster
创建包周期集群	集群	createCluster/createPeriodicCluster
更新集群描述	集群	updateCluster
升级集群	集群	clusterUpgrade
删除集群	集群	claimCluster/deleteCluster
下载集群证书	集群	getClusterCertByUID
绑定、解绑 eip	集群	operateMasterEIP
集群休眠唤醒、节点纳管重置（V2）	集群	operateCluster
集群休眠（V3）	集群	hibernateCluster
集群唤醒（V3）	集群	awakeCluster
按需集群规格变更	集群	resizeCluster
包周期集群规格变更	集群	resizePeriodCluster
修改集群配置	集群	updateConfiguration
创建节点池	节点池	createNodePool

操作名称	资源类型	事件名称
更新节点池	节点池	updateNodePool
删除节点池	节点池	claimNodePool
迁移节点池	节点池	migrateNodepool
修改节点池配置	节点池	updateConfiguration
创建节点	节点	createNode
创建包周期节点	节点	createPeriodNode
删除集群下所有节点	节点	deleteAllHosts
删除单个节点	节点	deleteOneHost/claimOneHost
更新节点描述	节点	updateNode
创建插件实例	插件实例	createAddonInstance
删除插件实例	插件实例	deleteAddonInstance
上传模板	模板	uploadChart
更新模板	模板	updateChart
删除模板	模板	deleteChart
创建模板实例	模板实例	createRelease
升级模板实例	模板实例	updateRelease
删除模板实例	模板实例	deleteRelease
创建 ConfigMap	K8S 资源	createConfigmaps
创建 DaemonSet	K8S 资源	createDaemonsets
创建 Deployment	K8S 资源	createDeployments
创建 Event	K8S 资源	createEvents
创建 Ingress	K8S 资源	createIngresses
创建 Job	K8S 资源	createJobs
创建 namespace	K8S 资源	createNamespaces
创建 Node	K8S 资源	createNodes
创建 PersistentVolumeClaim	K8S 资源	createPersistentvolumeclaims
创建 Pod	K8S 资源	createPods
创建 ReplicaSet	K8S 资源	createReplicasets

操作名称	资源类型	事件名称
创建 ResourceQuota	K8S 资源	createResourcequotas
创建密钥	K8S 资源	createSecrets
创建服务	K8S 资源	createServices
创建 StatefulSet	K8S 资源	createStatefulsets
创建卷	K8S 资源	createVolumes
删除 ConfigMap	K8S 资源	deleteConfigmaps
删除 DaemonSet	K8S 资源	deleteDaemonsets
删除 Deployment	K8S 资源	deleteDeployments
删除 Event	K8S 资源	deleteEvents
删除 Ingress	K8S 资源	deleteIngresses
删除 Job	K8S 资源	deleteJobs
删除 Namespace	K8S 资源	deleteNamespaces
删除 Node	K8S 资源	deleteNodes
删除 Pod	K8S 资源	deletePods
删除 ReplicaSet	K8S 资源	deleteReplicasets
删除 ResourceQuota	K8S 资源	deleteResourcequotas
删除 Secret	K8S 资源	deleteSecrets
删除 Service	K8S 资源	deleteServices
删除 StatefulSet	K8S 资源	deleteStatefulsets
删除卷	K8S 资源	deleteVolumes
替换指定的 ConfigMap	K8S 资源	updateConfigmaps
替换指定的 DaemonSet	K8S 资源	updateDaemonsets
替换指定的 Deployment	K8S 资源	updateDeployments
替换指定的 Event	K8S 资源	updateEvents
替换指定的 Ingress	K8S 资源	updateIngresses
替换指定的 Job	K8S 资源	updateJobs
替换指定的	K8S 资源	updateNamespaces

操作名称	资源类型	事件名称
Namespace		
替换指定的 Node	K8S 资源	updateNodes
替换指定的 PersistentVolumeClaim	K8S 资源	updatePersistentvolumeclaims
替换指定的 Pod	K8S 资源	updatePods
替换指定的 Replicaset	K8S 资源	updateReplicasets
替换指定的 ResourceQuota	K8S 资源	updateResourcequotas
替换指定的 Secret	K8S 资源	updateSecrets
替换指定的 Service	K8S 资源	updateServices
替换指定的 Statefulset	K8S 资源	updateStatefulsets
替换指定的状态	K8S 资源	updateStatus
上传组件模板	K8S 资源	uploadChart
更新组件模板	K8S 资源	updateChart
删除组件模板	K8S 资源	deleteChart
创建模板应用	K8S 资源	createRelease
更新模板应用	K8S 资源	updateRelease
删除模板应用	K8S 资源	deleteRelease

18.2 查看云审计日志

操作场景

开启了云审计服务后，系统开始记录 CCE 资源的操作。云审计服务管理控制台保存最近 7 天的操作记录。

操作步骤

步骤 1 登录管理控制台。

步骤 2 在管理控制台左上角单击 图标，选择区域。

步骤 3 单击页面上方的“服务列表”，选择“管理与部署 > 云审计服务”，进入云审计服务信息页面。

步骤 4 单击左侧导航树的“事件列表”，进入事件列表信息页面。

步骤 5 事件列表支持通过筛选来查询对应的操作事件。当前事件列表支持四个维度的组合查询，详细信息如下：

- 事件来源、资源类型和筛选类型。

  在下拉框中选择查询条件。其中，事件来源选择“CCE”。

  当筛选类型选择事件名称时，还需选择某个具体的事件名称。

  选择资源 ID 时，还需选择或者手动输入某个具体的资源 ID。

  选择资源名称时，还需选择或手动输入某个具体的资源名称。
- 操作用户：在下拉框中选择某一具体的操作用户，此操作用户指用户级别，而非租户级别。
- 事件级别：可选项为“所有事件级别”、“normal”、“warning”、“incident”，只可选择其中一项。
- 时间范围：可选择查询最近七天内任意时间段的操作事件。

步骤 6 在需要查看的记录左侧，单击 展开该记录的详细信息，展开记录如下图所示。

图18-1 展开记录

步骤 7 在需要查看的记录右侧，单击“查看事件”，将会弹出一个窗口显示该操作事件结构的详细信息。

图18-2 查看事件


```
{
    "code": "200",
    "source_ip": "10.2[illegible].21",
    "trace_type": "ApiCall",
    "event_type": "system",
    "project_id": "14fc2d[illegible]240e407",
    "trace_id": "02fe9743-3502-11ee-ad8a-65a9e1eb2069",
    "trace_name": "updateConfigmaps",
    "resource_type": "configmaps",
    "trace_rating": "normal",
    "message": "UserName: [illegible]telecom.cn/asm_admin_trust; Operation : PUT /api/v1/namespaces/istio-system/
    "service_type": "CCE",
    "resource_id": "5d660d24-eaf8-11ed-ad7a-0255ac100019",
    "tracker_name": "system",
    "time": "2023-08-07 17:08:45 GMT+08:00",
    "resource_name": "istio-sidecar-quota",
    "record_time": "2023-08-07 17:08:45 GMT+08:00",
    "user": {
        "domain": {
            "name": "[illegible]inatelecom.cn",
            "id": "f18fae26ab1e491da2cbe9d61d5ff30a"
        },
        "id": "11fe006d7182a0004fd9c0019d4eb2da",
        "name": "[illegible]atelecom.cn/asm_admin_trust"
    }
}
```


----结束