百度智能云

CCE 文档

【版权声明】

版权所有©百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司。 未经本公司书面许可，任何单位和个人不得擅自摘抄、复制、传播本文档内容，否则本公司有权依法追究法律责任。

【商标声明】

和其他百度系商标，均为百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司的商标。 本文档涉及的第三方商标，依法由相关权利人所有。未经商标权利人书面许可，不得擅自对其商标进行使用、复制、修改、传播等行为。

【免责声明】

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导。 如您购买本文档介绍的产品、服务，您的权利与义务将依据百度智能云产品服务合同条款予以具体约定。本文档内容不作任何明示或暗示的保证。

目录

目录 2
功能发布记录 9
产品公告 24
Kubernetes 版本发布说明 24
CCE Kubernetes 版本更新说明 24
CCE发布Kubernetes 1.30版本说明 25
CCE发布Kubernetes 1.28版本说明 25
CCE发布Kubernetes 1.26版本说明 26
CCE发布Kubernetes 1.24版本说明 27
CCE发布Kubernetes 1.22版本说明 27
CCE发布Kubernetes 1.20版本说明 28
CCE发布Kubernetes 1.18版本说明 29
安全漏洞修复公告 29
漏洞CVE-2025-1097、CVE-2025-1098等修复公告 29
修复漏洞CVE-2021-30465公告 31
漏洞CVE-2020-14386修复公告 31
漏洞CVE-2019-5736修复公告 32
关于 runc 的安全问题（CVE-2024-21626）影响声明 33
产品描述 34
介绍 34
核心概念 34
镜像仓库 35
优势 35
特性 35
应用场景 36
使用限制 36
产品定价 39
产品定价 39
快速入门 39
使用CCE容器引擎流程概述 39
快速部署nginx应用 40
操作指南 41
使用须知 41
集群管理 42
创建集群 42
查看集群 47
操作集群 49
通过kubectl连接集群 51
CCE节点资源预留说明 64
CCE 节点 CDS 扩容 66

CCE安全组说明 69
集群快照 71
升级集群Kubernetes版本 71
托管集群使用说明 74
节点管理 75
添加节点 75
移出节点 79
设置节点封锁 80
设置GPU显存共享 81
管理节点标签 82
管理污点 84
自定义Kubelet参数 86
kubelet容器监控只读端口风险提示 89
节点排水 90
设置节点缩容保护 93
节点组管理 94
节点组介绍 94
节点组概述 94
核心概念 94
使用限制 94
节点组管理 95
添加备选机型 105
配置扩缩容策略 106
添加外部已有节点 108
节点组节点故障检测自愈 110
升级节点组 111
自定义节点组 Kubelet 配置 113
命名空间管理 117
命名空间基本操作 117
设置资源配额 118
设置资源限制 120
Helm管理 121
Helm模板 121
Helm实例 123
工作负载 124
Deployment管理 124
Daemonset管理 133
Statefulset管理 141
Job管理 150
CronJob管理 153
查看容器组 157

使用私有镜像创建工作负载 158
流量接入 160
通过 CCE 使用 K8S Service 160
通过 YAML 创建 LoadBalancer Service 161
Service复用已有负载均衡BLB 165
使用直连 Pod 模式 LoadBalancer Service 166
LoadBalancer Service Annotation说明 169
通过 CCE 使用 K8S Ingress 179
通过YAML创建CCE_Ingress 183
CCE 基于 nginx-ingress实现灰度发布 186
使用 NGINX Ingress 198
NGINX Ingress 使用配置参考 199
BLB Ingress Annotation说明 201
网络管理 211
创建VPC-ENI模式集群 211
容器网段空间耗尽如何继续扩容(VPC网络模式) 213
CCE 支持 IPv4 和 IPv6 双栈网络 218
对等连接场景下容器流量转发配置 226
容器网段空间耗尽如何继续扩容(VPC-ENI模式) 227
使用Network Policy网络策略 227
网络连通性 229
网络维护 234
网络编排 235
DNS 248
配置管理 255
Configmap管理 255
Secret管理 256
存储管理 259
概述 259
使用本地存储 260
使用云盘CDS 263
使用文件存储CFS 271
使用对象存储BOS 282
使用并行文件存储PFS 287
使用并行文件存储PFS L2 290
使用数据湖存储加速工具RapidFS 293
监控日志 296
日志中心 296
使用Prometheus监控集群 306
CCE 事件中心 308

CCE 集群异常事件报警 308
集群服务画像 310
集群审计 313
集群审计仪表盘 315
操作场景 315
前提条件 315
Java应用监控 318
权限管理 319
权限概述 319
配置IAM预置权限策略 320
配置IAM自定义权限策略 322
配置IAM标签权限策略 325
配置预置RBAC权限策略 327
配置集群OIDC认证 328
虚拟节点 329
管理虚拟节点 329
配置BCI Pod 333
配置bci-profile 335
弹性伸缩 337
容器水平伸缩（HPA） 337
容器定时水平伸缩（CronHPA） 340
CCE 集群节点自动伸缩 344
使用 cce-autoscaling-placeholder 实现秒级弹性伸缩 347
云原生AI 348
云原生AI概述 348
队列管理 348
任务管理 356
任务结束时，pod的清理策略，All表示所有pod，none表示不清理 369
完成策略，All表示所有pod完成即任务完成，Any表示任何pod完成即任务完成 369
失败策略，All表示所有pod失败即任务失败，Any表示任何pod完成即任务完成 369
支持horovod与paddle框架 369
弹性选项，true表示开启弹性，false不开启，开启时需开启trainer容器的容错选项 370
数据集管理 377
AI 加速套件 382
AI监控大盘 394
前提条件 397
前提条件 399
GPU虚拟化 413
组件管理 422
组件概述 422
CCE GPU Manager 说明 423

CCE Deep Learning Frameworks Operator 说明 427
CCE RDMA Device Plugin 说明 429
CCE AI Job Scheduler 说明 430
CCE 动态调度插件说明 436
CCE Fluid说明 441
CCE CSI PFS Plugin 说明 443
CCE CSI BOS Plugin 说明 445
CCE CSI CDS Plugin 说明 446
CCE NPU Manager 说明 448
CCE Image Accelerate 说明 449
CCE Ingress Nginx Controller 说明 450
CCE Ingress Controller 说明 454
CCE Log Operator 说明 455
CCE P2P Accelerator 说明 456
CCE Node Problem Detector 说明 459
CCE Node Remedier 说明 463
CCE Descheduler 说明 468
CCE QoS Agent 说明 474
CCE NodeLocal DNSCache 说明 484
CCE Ascend Mindx DL说明 490
CoreDNS 说明 491
CCE Network Plugin说明 492
CCE LB Controller 说明 497
CCE CronHPA Controller 说明 499
CCE CSI PFS L2 Plugin 503
Kube Scheduler 说明文档 505
Kube Controller Manager 说明 505
Kube ApiServer 说明 505
CCE Backup Controller 说明 505
CCE Credential Controller 说明 506
CCE Virtual Kubelet组件 508
CCE Onepilot 说明 510
备份中心 511
备份概述 511
备份仓库 511
备份管理 512
恢复管理 513
巡检与诊断 514
集群巡检 514
GPU运行环境检查 518
故障诊断 519

典型实践 524
CCE使用检查清单 524
CCE-访问公网实践 526
CCE集群中使用私有镜像实践 529
CCE典型实践之容器网络模式选择 531
通过 CCE Ingress 实现虚机和容器服务的统一接入 531
云原生AI 536
部署 TensorFlow Serving 推理服务 536
GPU虚拟化之隔离性最优型的最佳实践 539
Linux系统配置常用参数说明 553
使用 CNI 配置容器内网络参数 554
CCE容器运行时选择 555
添加CGroup V2节点 556
用户使用自定义 CNI 插件方法 557
CCE Resource Recommender 用户文档 560
Pod异常问题排查 567
利用 Velero 实现将应用跨云迁移到 百度CCE 572
1.简介 572
API参考（即将废弃） 578
概述 578
通用说明 579
公共头和错误返回 580
API_V2参考 585
概述 585
通用说明 586
服务域名 587
公共头和错误返回 588
Cluster相关接口 589
Autoscaler相关接口 601
InstanceGroup相关接口 605
Instance相关接口 616
Kubeconfig相关接口 632
Network相关接口 633
Task相关接口 639
组件管理相关接口 642
附录 651
RBAC相关接口 667
套餐适配相关接口 673
套餐适配相关接口 673
获取套餐列表 673

SDK 674
Go-SDK 674
概述 674
安装SDK工具包 674
初始化 674
集群管理 676
节点管理 680
节点组管理 682
常见问题 687
常见问题总览 687
通过kubectl创建简单的服务 688
业务应用如何使用负载均衡？ 689
自动扩缩容常见问题 692
集群管理常见问题 692
服务等级协议SLA 695
CCE服务等级协议SLA(V1.0) 695
常用工具 696
视图命令行场景示例 696

功能发布记录

发布时间	功能概述
2025-03	• 新增 故障诊断 功能，辅助您定位节点异常或Pod异常的根因。
2025-02	• 新增 节点组升级 功能，支持对节点的Kubelet和容器运行时版本进行升级，针对GPU节点支持升级nvidia-container-toolkit版本，提升节点运维效率。 • 新增节点组Kubelet配置功能，支持自定义节点组节点的Kubelet配置，提高用户运维效率。 • 节点支持挂载 PL2 类型 CDS 盘。 • 集群Containerd发布1.6.36和1.7.25版本，支持通过节点组升级至当前版本。 • 上线节点绑定 BCC IAM 角色，新建节点支持为服务器绑定IAM角色，实现在节点内部基于STSl临时身份凭证调用其他云产品的API。 • 新增节点数据盘加密功能，新建节点支持为数据盘开启加密功能，保障节点上磁盘数据不被窃取或泄露。 • 新增节点组同步更新存量节点同步更新节点标签、污点、注解信息功能，实现新旧节点配置一致，提高用户运维效率。 • 新增表单化创建管理CronHPA资源，支持按照类似Crontab的策略定时对Pod进行扩缩容，适用于应用资源使用率存在周期性变化的场景。
2025-01	• 上线Virtual Kubelet组件，通过安装CCE Virtual Kubelet组件，用户可将Pod快速调度到弹性容器实例BCI上运行，无需购买和管理BCC节点，更多信息请见 CCE Virtual Kubelet 组件 。 • 上线CCE监控日志就近体验功能，支持在节点、工作负载等资源列表页直接查看监控日志信息，缩短用户使用路径。 • 支持GPU运行环境检查，通过检查，用户可以确保包含加速芯片（如GPU）的节点在运行AI任务时具备必要的硬件和软件条件，从而提高任务的执行效率和成功率，更多信息请见 GPU运行环境检查 。
2024-12	• 节点列表增加加速芯片属性，节点列表增加加速芯片类型、空闲加速芯片卡数等属性，方便用户了解加速芯片使用情况。 • 容器组功能增强，容器组列表页面支持按节点IP和Pod IP地址搜索，并新增Pod Conditions信息，方便用户了解Pod健康情况。 •

2024-12	• 容器网络“VPC路由”网络模式白名单开放，需要 提交工单 申请白名单。 • 节点列表移出节点组节点支持同步调整期望节点数。 • 支持Rocky Linux 9.3操作系统，支持用户使用 Rocky Linnux 9.3 操作系统创建BCC、EBC 节点，提升用户体验。
2024-11	• 控制台支持创建保密字典，支持表单化创建保密字典Secret，无需base64编码，降低使用门槛。 • 上线集群管理左侧导航栏刷新。 • 节点列表支持筛选已封锁节点，优化用户体验。 • 支持为IAM角色进行RBAC授权，支持为IAM角色授予集群RBAC权限，实现用户通过角色访问集群资源，更多信息请见 配置预置RBAC权限策略 。 • 集群安全组优化，支持配置“自动创建默认安全组”和“使用已有安全组”，更多信息请见 CCE安全组 。 • 上线节点默认资源预留策略，支持根据节点规格大小自动调整系统预留资源，提升节点稳定性。更多信息请见 CCE 节点资源预留说明 。
2024-10	• 新增 集群巡检 功能，帮助您发现容器服务集群中可能存在的潜在风险。 • 新增VPC-ENI集群支持裸金属BBC节点功能，支持添加裸金属BBC作为节点，VPC-ENI模式下容器IP地址为分配的主网卡辅助IP。
2024-09	• 添加节点支持指定弹性高性能计算集群EHC，同一EHC集群中的节点之间可通过RDMA网络高速互联，提供高带宽、低时延的通信能力，更多信息请见弹性高性能集群概述 。 • 节点组支持绑定自愈规则，支持配置故障检测项及自愈操作实现节点发生故障时自动检测，自动执行自愈操作，更多信息请见节点组节点故障检测自愈。 • 组件管理新增 CCE Credential Controller 镜像免密拉取组件，实现部署工作负载时无需指定私有镜像访问凭证。 • 组件管理增加搜索功能，支持快速检索组件。

	• CCE Ingress Nginx Controller 组件支持 K8s 1.26及以上版本。
2024-08	• 发布 Kubernetes 1.30 版本，遵循 K8s 一致性原则，更多信息请见 CCE 发布 Kubernetes 1.30 版本说明，支持低版本集群升级至 1.30 版本。 • 上线备份中心模块，为容器化应用的备份、恢复提供了一体化解决方案，更多信息请见备份概述。 • 新增资源对象浏览器功能，支持通过 API 组、版本号管理 Kubernetes 资源，包括自定义资源。 • 节点新增排水操作，支持将节点上的 Pod 主动驱逐并封锁节点，提高节点运维能力。 • 节点列表增加虚拟节点，支持管理虚拟节点上调度的 BCI Pod。 • CCE CronHPA Controller 组件支持 Kubernetes 1.28 及以上版本，支持配置时区。
2024-07	• Prometheus 监控预置集群组件监控面板，针对网路、存储、监控、日志、镜像等组件提供基础指标展示。 • 托管集群中的 CCE Ingress Controller 组件新版本将托管在 CCE 侧，对用户免运维。 • 组件管理新增托管集群核心组件（ Kube APIServer、Kube Scheduler、Kube Controller Manager ），支持为核心组件快速开启日志采集和容器监控。 • CoreDNS 组件接入日志中心，支持一键开启日志采集，提供 CoreDNS 日志持久化存储与分析能力。 • 新建节点支持指定目录挂载本地磁盘，支持本地机型使用本地磁盘存储容器数据。 • VPC-ENI 容器网络模式下取消节点和容器不能使用相同私有网络子网的限制。

2024-06	• 发布 Kubernetes 1.28 版本，遵循 K8s 一致性原则，更多信息请见 CCE 发布 Kubernetes 1.28 版本说明。 • 容器运行时 Containerd 支持 1.7.13 版本。 • CCE CSI CDS Plugin 组件动态挂载云磁盘 CDS 可配置删除时是否进入回收站，更多信息请见 CCE CSI CDS Plugin 组件版本记录。 • 新建托管集群 API Server 调整为绑定用户的 BLB、EIP 资源，支持通过为 BLB 配置安全组规则实现集群访问控制。 • 事件中心提供事件总览大盘，提供事件总数及分布情况、节点异常、Pod OOM、重要事件趋势等仪表盘以及异常 TOP 事件列表。 • 集群审计提供审计日志可视化仪表盘的能力，将重点审计日志进行可视化统计分析，更多信息请见 集群审计仪表盘。
2024-05	• 支持纳管边缘计算节点构建标准的独立集群，提供边缘业务容器化部署能力。 • 上线 CCE CSI PFS L2 Plugin 组件，支持给容器挂载 L2 极速型并行文件存储。 • 节点组支持添加已有节点，可以将集群内游离的节点和集群外部的服务器添加到节点组中实现统一管理。
2024-04	• 上线 CCE CronHPA Controller 组件，提供工作负载定时弹性伸缩能力，解决固定周期变化业务对资源的弹性要求，降低资源使用成本。 • 集群审计支持投递到日志服务 BLS 进行持久化存储和查询。 • 添加已有节点支持针对服务器数据盘进行格式化，并指定数据目录进行挂载，实现使用数据盘存储容器、kubelet 数据。

2024-03	• 控制台上支持查看节点上运行的容器组，方便用户以节点的维度对 Pod 进行管理。 • 组件管理上线 CoreDNS 组件，支持手动升级组件版本。
2024-02	• LB Service 支持同端口多协议部署方式，满足DNS等业务使用诉求。 • 发布 Kubernetes 1.26 版本，遵循 K8s 一致性原则，更多信息请见 CCE 发布 Kubernetes 1.26 版本说明。 • 节点组新增复制操作，支持基于已有节点组配置快速创建新节点组。
2023-11	• 上线 NodeLocalDNSCache 组件，可提升集群服务发现的稳定性和性能。 • 集群支持通过cGroup v2实现容器的资源隔离。 • LB Service/Ingress支持通过 Annotation 指定EIP的线路类型及公网带宽范围。 • CCE集群融合监控服务CProm预置Prometheus监控面板，支持监控面板有：控制面组件监控（APIServer监控、ETCD监控、Kube Scheduler监控、Kube Controller Manager监控）、节点监控、工作负载监控、网络监控（CoreDNS监控）、集群Namespace监控。 • CCE集群预置云原生AI监控面板，支持监控面板有资源统计、节点资源、工作负载、调度器组件稳定性、GPUManager稳定性，通过预置的监控面板能够了解整个集群的GPU使用情况、工作负载性能、调度器组件稳定性等，从而实现对异常问题的快速诊断、优化GPU资源的分配、提升资源利用率。
2023-10	• 上线 CCE Qos Agent 组件，提供丰富的单机隔离能力，提升集群资源利用率和稳定性。 • 日志采集规则支持配置投递端 BES，将容器日志传输至 BES 中存储、分析。

	• 故障自愈组件 CCE Node Remedier 支持配置重启弹性裸金属服务器 EBC 自愈策略。
2023-09	• CCE CSI BOS Plugin 支持在 Containerd 容器运行时环境中挂在对象存储 BOS。 • 故障检测组件 CCE Node Problem Detector 支持检测CPU、内存等硬件故障。 • LB Service支持根据 label 选择挂载后端目标节点。
2023-08	• CCE 集群接入资源标签及标签鉴权，支持通过标签对集群进行管理和授权。 • 存储组件 CCE CSI PFS Plugin 发布 1.2.5 版本，支持使用多个 PFS 实例，更多信息请见使用并行文件存储 PFS。 • 新增负载感知调度插件，支持基于节点实际资源使用情况进行调度。 • LB Service/Ingress 支持通过 YAML 直接修改关联的 BLB、EIP 配置参数，具体配置信息请见 LoadBalancer Service Annotation说明 和 BLB Ingress Annotation说明。
2023-07	• 基于 VPC-ENI 容器网络模式实现 IPv6 双栈网络，支持部署 IPv4 和 IPv6 服务。 • 支持直接在控制台上使用自定义镜像创建弹性裸金属服务器 EBC 节点。
2023-06	• VPC-ENI 容器网络模式支持弹性裸金属服务器 EBC。 • 组件管理中心新增 CCE Descheduler 重调度组件，支持基于节点真实利用率重新调度 Pod。 • CCE Node Problem Detector 故障检测组件新增 GPU 等硬件故障检测，更多信息请见 GPU 故障检测介绍

	CCE Node Problem Detector 故障检测组件新增 GPU 等硬件故障检测，更多信息请见 GPU 故障检测介绍。 • 部署工作负载时支持在控制台上直接选择容器镜像服务 CCR 企业版和个人版中的容器镜像。
2023-05	• 托管集群控制面组件日志接入日志服务 BLS，更多信息请见查看集群控制面日志。 • 添加节点支持自定义 Kubelet 参数，更多信息请见自定义 Kubelet 参数。 • 支持更新集群访问凭证证书。 • 创建集群时支持开启 Network Policy，更多信息请见使用 Network Policy 网络策略。 • VPC 路由容器网络模式下支持弹性裸金属服务器 EBC。 • VPC-ENI 容器网络模式支持 Pod 独占弹性网卡，该功能白名单开放。
2023-04	• 发布 Kubernetes 1.24 版本，更多信息请见 CCE 发布Kubernetes 1.24版本说明。 • 组件管理中心新增 CCE Node Remedier 故障自愈组件。 • 控制台支持设置命名空间资源配额与限制，更多信息请见设置资源配额和设置资源限制。 • 支持使用公共镜像 BaiduLinux 3.0 创建集群节点。 • LB Service/Ingress 支持配置 EIP 线路类型。
	• 集群支持配置IAM自定义权限策略，更多信息请见配置 IAM 自定义权限策略

	集群支持配置IAM自定义权限策略，更多信息请见配置 IAM 自定义权限策略。
2023-03	• 组件管理中心新增 CCE Node Problem Detector 故障检测组件。 • LB Service（Local 模式）支持负载均衡配置，更多信息请见设置服务器组动态权重。 • 创建 GPU 节点支持勾选安装趋动。
2023-02	• 新增日志中心功能，将容器日志接入日志服务 BLS，更多信息请见日志概述。 • 集群、节点组支持创建预付费节点。 • 支持使用公共镜像 Rocky Linux 8.6 创建集群节点。 • LB Service支持扩展 Http、Https 等协议，更多信息请见扩展 Service 端口协议。 • 节点组支持多子网均匀分布扩缩容策略，更多信息请见配置扩缩容策略。 • 新增组件管理功能，支持管理集群扩展组件，更多信息请见组件概述。
2023-01	• 控制台支持查看容器子网。 • 【体验优化】全面升级产品页面风格，您可以去控制台上体验。
2022-06	• 节点组支持添加备选机型，可指定不同可用区下的不同机型（规格相同）。

2022-05	• 支持变更节点组配置。 • 支持 BLB Ingress Annotation，实现更丰富的负载均衡能力。
2022-04	• 混部大盘 增加混部收益计算与容器利用率统计等指标。 • 离线负载创建和资源配额设置流程优化。
2021-08	• 增加 Namespace 资源限制说明。集群管理员可以通过 Resource Quota 及 LimitRange 为每个租户设置资源限额。
2021-07	• 新增VPC-CNI模式下容器网段空间耗尽时的扩容方案。VPC-CNI 模式下，容器的 IP 来自 VPC 中的子网。随着集群规模扩大，ENI 子网中的 IP 可能耗尽，利用 CNI 提供的动态配置功能，可以给集群增加新的 ENI 子网。
2020-09	• 控制台应用管理相关功能（工作负载、流量接入、存储配置、监控日志等）进行菜单位置调整，移至单集群管理页面。变更详情请查看CCE 控制台升级 • Serverless Kubernetes集群开放公测，简化集群管理和运维，降低部署和维护成本 • 支持创建集群时，根据实际业务场景，选择不同类型的模板创建集群 • 支持使用密钥管理服务（Key Management Service，简称：KMS）对存储在etcd中的Kubernetes Secret数据进行加密
	• 集群管理功能升级：（1）支持自定义Master选择多副本规格、多可用区；（2）支持Master、Worker已有实例创建或扩容集群，同时提供自定义脚本注入功能；（3）支持自定义Master、LB Service、Ingress指定其关联负载均衡器BLB所在的VPC子网；（4）Kubelet升级为动态配置、KubeProxy使用容器化部署；（5）支持节点封锁（cordon）、集群OIDC认证、

	选择集群NodeName类型等功能
2020-08	• 容器网络功能增强：（1）支持自定义Service ClusterIP、Service NodePort；（2）支持VPC网络模式自定义网卡类型（kubenet、ipvlan、veth、auto） • 优化使用交互体验：（1）开放新版集群管理OpenAPI ；（2）控制台创建集群界面升级
2020-07	• CCE部署支持自定义K8S参数 • 容器网络VPC-CNI模式支持Pod固定IP • 支持容器引擎CCE的镜像向容器镜像服务CCR迁移
2020-06	• CCE控制台支持WebShell • 支持对Deployment、StatefulSet设置周期、定时伸缩任务，并与工作负载自动水平伸缩HPA结合使用，扩展弹性伸缩的应用场景
2020-05	• 集群自动伸缩支持配置竞价实例等待时间 • 控制台提示说明文案与文档链接展示优化
	•

2020-04	支持Kubernetes 1.16.8 版本 • 容器监控报警接收配置接入IAM，支持以邮件或短信将报警发送给指定用户或用户组 • 事件中心功能上线 • 节点配置支持选择BCC实例规格族 • CCE控制台支持配置工作负载自动水平伸缩 • CCE控制台支持创建、管理虚拟节点
2020-03	• 支持将创建工作负载、服务、配置存储的 Yaml 文件另存为自定义模板，并进行管理和编辑、删除等操作 • 支持通过表单使用Docker镜像部署工作负载 • 支持通过表单创建持久卷、持久卷声明
2020-02	• 支持使用昆仑云服务器（百度自研XPU芯片）创建集群 • 优化集群扩容策略和伸缩组节点数量展示

2020-01	• 支持使用伸缩组对集群竞价实例进行自动扩缩容
2019-12	• 容器网络支持VPC-CNI模式，将容器网络和集群节点网络设置为相同的网络地址空间 • 容器日志支持推送至百度智能云对象存储服务BOS
2019-11	支持使用伸缩组进行集群自动扩缩容
2019-10	• 支持为百度智能云子用户分配CCE集群中的kubernetes RBAC权限 • CCE支持并行处理集群扩容和缩容请求
2019-09	支持添加物理服务器BBC作为CCE集群节点
2019-08	CCE支持对拥有CCE只读权限的子账号配置RBAC权限，实现对CCE集群命名空间的权限控制
2019-07	CCE支持用户使用开源工具velero对云上的kubernetes集群进行备份、恢复。用于集群误操作、集群故障、集群迁移等场景
2019-06	• 支持K8S操作审计 • 集群列表优化，新增资源使用情况展示
2019-05	• 支持独立master部署 • 告警功能上线

	• 支持K8S版本升级
2019-04	• 支持Helm模板管理与Helm应用安装 • 支持已有节点移入移出 • 支持预付费工作节点 • 支持集群事件进度追踪
2019-03	支持使用容器镜像构建服务。百度智能云镜像构建服务支持在页面上配置基于代码仓库或Dockerfile的镜像构建规则，并由用户手动触发/代码仓库变更自动触发镜像构建，打通用户代码变更到容器部署的中间环节，方便用户更高效便捷地部署容器化服务
2019-02	• 支持Ingress流量管理 • UI界面升级，新增菜单：配置存储、监控日志、流量接入 • 镜像仓库支持镜像自动构建
	• 支持百度智能云多用户访问控制 • 支持容器日志采集，用户可以在CCE控制台，对Kubernetes中的容器日志和业务日志设置管理策略

2018-12	• 支持集群资源和Master组件监控 • 用户可以通过Kubernetes中的PV/PVC方式，将BOS存储挂载到容器中使用
2018-11	支持用户在CCE控制台为集群中部署的后端服务快速创建和管理基于应用型BLB的负载均衡
2018-10	• 支持创建集群时选择K8S版本 • 支持使用DCC创建集群 • 上线Fabric和超级链解决方案 • 上线应用管理，支持控制台管理容器、服务、部署等资源
2018-08	• 上线百度智能云华北-保定区域 • 限制最小节点配置为4G和8G • K8S版本升级至1.11 • 开放节点计费变更功能 • 集成Prometheus和alert监控报警 •

	在单个Pod中挂载多块CDS
2018-05	• 上线百度智能云香港区域 • 上线自动扩缩容功能 • 节点支持CentOS镜像 • 引入日志和事件机制 • 上线K8S Dashboard • 支持PV方式使用百度智能云CDS和CFS存储
2018-03	• 上线百度智能云广州、苏州区域 • CCE Serivce高可用 • K8S版本升级至1.8 • 支持NAT子网 • 获得CNCF “Kubernetes一致性认证”

2017-12	• 用户集群Master节点3副本高可用 • 开放百度智能云镜像仓库 • 支持CDS磁盘预挂载 • 支持GPU节点
2017-08	百度智能云容器引擎CCE正式发布

产品公告

Kubernetes 版本发布说明

CCE Kubernetes 版本更新说明

百度智能云容器引擎 CCE 提供高度可扩展的、高性能的企业级 Kubernetes 集群。由于社区定期发布 Kubernetes 版本，CCE 会随之发布相应的 Kubernetes 版本, 并严格遵循社区一致性认证。本文将介绍 CCE 的版本支持机制。

版本支持

集群创建

CCE 支持已发布的最新三个大版本的创建，例如当发布 Kubernetes 1.24 版本时，则支持创建 1.24、1.22、1.20 版本的集群 不再支持创建 1.18 版本的集群。

CCE 已发布的 Kubernetes 版本:

- 1.30 版本
- 1.28 版本
- 1.26 版本
- 1.24 版本
- 1.22 版本
- 1.20 版本
- 1.18 版本
- 1.16 版本
- 1.13 版本
- 1.11 版本

版本发布周期

- 从 Kubernetes 1.16 版本开始，CCE 仅发布 Kubernetes 次要版本为双数的版本，例如 1.24、1.26、1.28。
- 在 Kubernetes 社区发布新的双数次要版本后，CCE 会对该版本进行风险评估和一致性测试，在完成后开放新版本的创建和升级操作。

- 在 Kubernetes 社区针对双数次要版本发布新的补丁版本后，CCE 将会根据补丁所修复问题的风险等级判定是否发布该补丁版本的升级更新。对于涉及关键问题修复（如高危安全漏洞）的补丁版本，CCE 会进行评估验证，完成后开放新版本的创建和升级操作。

CCE发布Kubernetes 1.30版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证, 本文介绍 CCE 发布 Kubernetes 1.30 版本相对 1.28 所做重要变更说明。

资源变更和弃用

- v1.29 flowcontrol.apiserver.k8s.io/v1beta2 替换为 flowcontrol.apiserver.k8s.io/v1，包括 FlowSchema、PriorityLevelConfiguration。
- v1.29 移除 networking/v1alpha1 ClusterCIDR。
- v1.29 kubescheduler.config.k8s.io/v1beta3 替换为 kubescheduler.config.k8s.io/v1，包括 KubeSchedulerConfiguration。
- v1.29 v1 Node 弃用 .status.nodeInfo.kubeProxyVersion。
- v1.29 batch/v1 CronJob .spec.schedule 不再支持配置 CRON_TZ 或 TZ，使用 .spec.timeZone 代替。

其他变更和弃用

- v1.29 移除 selectorSpread 调度器插件，使用 podTopologySpread 调度器插件代替。
- v1.30 移除 SecurityContextDeny 准入插件，使用 PodSecurity 准入插件代替。
- v1.30 移除 kubectl apply --prune-whitelist 参数，使用 --prune-allowlist 参数代替。

参考链接

关于 Kubernetes 1.30 与其他版本详细对比可参考:

- kubernetes 1.29 changelog
- kubernetes 1.30 changelog

CCE发布Kubernetes 1.28版本说明

百度智能云容器引擎 CCE 严格遵循社区一致性认证, 本文介绍 CCE 发布 Kubernetes 1.28 版本相对 1.26 所做重要变更说明。

资源变更和弃用

- 从 v1.27 版本开始不再提供 storage.k8s.io/v1beta1 API 版本的 CSIStorageCapacity。
  - 自 v1.24 版本起，迁移清单和 API 客户端使用 storage.k8s.io/v1 API 版本
  - 所有现有的持久化对象都可以通过新的 API 访问
  - 没有需要额外注意的变更
- v1.27版本 resource.k8s.io/v1alpha1 替换为 resource.k8s.io/v1alpha2，包括PodScheduling、ResourceClaim、ResourceClaimTemplate、ResourceClass、Status，其中PodScheduling重命名为PodSchedulingContext
- v1.27 Pod 移除 seccomp annotations，包括seccomp.security.alpha.kubernetes.io/pod、container.seccomp.security.alpha.kubernetes.io，使用 .spec.securityContext.seccompProfile 代替
- v1.27 移除 service.kubernetes.io/topology-aware-hints annotations，使用 service.kubernetes.io/topology-mode 代替
- v1.27 resource/v1alpha1/DynamicResourceManagement 替换为 resource/v1alpha2/DynamicResourceManagement
- v1.28 PersistentVolumeClaim .status.resizeStatus 替换为 .status.allocatedResourceStatuses
- v1.28 NetworkPolicy 移除 .status
- v1.28 移除 kubescheduler.config.k8s.io/v1beta2，使用 kubescheduler.config.k8s.io/v1 代替

其他变更和弃用

- v1.27 kube-apiserver 移除 --master-service-namespace 选项
- v1.27 kube-controller-manager 移除 --enable-taint-manager 选项
- v1.27 kube-controller-manager 移除 --pod-eviction-timeout 选项

- v1.27 kubelet 移除 --container-runtime 选项
- v1.28 kube-controller-manager 弃用 --volume-host-cidr-denylist、 --volume-host-allow-local-loopback 选项
- v1.28 kube-scheduler 移除 --lock-object-namespace、--lock-object-name 选项，使用 --leader-elect-resource-namespace、--leader-elect-resource-name 或 ComponentConfig 代替
- v1.28 kubelet 弃用 --azure-container-registry-config 选项，使用 --image-credential-provider-config、--image-credential-provider-bin-dir 代替

参考链接

关于 Kubernetes 1.28 与其他版本详细对比可参考:

- kubernetes 1.27 changelog
- kubernetes 1.28 changelog

CCE发布Kubernetes 1.26版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证, 本文介绍 CCE 发布 Kubernetes 1.26 版本相对 1.24 所做重要变更说明。

资源变更和弃用

- flowcontrol.apiserver.k8s.io/v1beta1 被弃用，从 Kubernetes 1.26 版本开始，可以使用flowcontrol.apiserver.k8s.io/v1beta3
- autoscaling/v2beta1 和 autoscaling/v2beta2 分别在 Kubernetes 1.25 和 1.26 版本被弃用，从 Kubernetes 1.23 版本开始，可以使用 autoscaling/v2
- batch/v1beta1 被弃用，从 Kubernetes 1.21 版本开始，可以使用 batch/v1
- discovery.k8s.io/v1beta1 被弃用，从 Kubernetes 1.21 版本开始，可以使用 discovery.k8s.io/v1，discovery.k8s.io/v1 中需要额外注意的变更有：
  - 使用每个Endpoint的NodeName字段，而不是已被弃用的topology["kubernetes.io/hostname"]字段。
  - 使用每个Endpoint的Zone字段，而不是已被弃用的topology["kubernetes.io/zone"]字段。
  - Topology字段被替换为deprecatedTopology字段，并且在API的v1版本中不可写入。
- events.k8s.io/v1beta1 被弃用，从 Kubernetes 1.19 版本开始，可以使用 events.k8s.io/v1，events.k8s.io/v1 中需要额外注意的变更有：
  - type 字段只能设置为 Normal 和 Warning 之一；
  - involvedObject 字段被更名为 regarding；
  - action、reason、reportingController 和 reportingInstance 字段 在创建新的 events.k8s.io/v1 版本 Event 时都是必需的字段；
  - 使用 eventTime 而不是已被弃用的 firstTimestamp 字段 （该字段已被更名为 deprecatedFirstTimestamp，且不允许出现在新的 events.k8s.io/v1 Event 对象中）；
  - 使用 series.lastObservedTime 而不是已被弃用的 lastTimestamp 字段 （该字段已被更名为 deprecatedLastTimestamp，且不允许出现在新的 events.k8s.io/v1 Event 对象中）；
  - 使用 series.count 而不是已被弃用的 count 字段 （该字段已被更名为 deprecatedCount，且不允许出现在新的 events.k8s.io/v1 Event 对象中）；
  - 使用 reportingController 而不是已被弃用的 source.component 字段 （该字段已被更名为 deprecatedSource.component，且不允许出现在新的 events.k8s.io/v1 Event 对象中）；
  - 使用 reportingInstance 而不是已被弃用的 source.host 字段 （该字段已被更名为 deprecatedSource.host，且不允许出现在新的 events.k8s.io/v1 Event 对象中）。
- autoscaling/v2beta1 被弃用，从 Kubernetes 1.23 版本开始，可以使用 autoscaling/v2
- policy/v1beta1 被弃用，从 Kubernetes 1.21 版本开始，可以使用 policy/v1，policy/v1 中需要额外注意的变更有：
  - 在 policy/v1 版本的 PodDisruptionBudget 中将 spec.selector 设置为空（{}）时会选择名字空间中的所有 Pod（在 policy/v1beta1 版本中，空的 spec.selector 不会选择任何 Pod）。如果 spec.selector 未设置，则在两个 API 版本下都不会选择任何 Pod。
- node.k8s.io/v1beta1 被弃用，从 Kubernetes 1.20 版本开始，可以使用 node.k8s.io/v1

其他变更和弃用

- 弃用 PodSecurityPolicy：从 Kubernetes 1.25 版本开始不再提供 policy/v1beta1 API 版本中的 PodSecurityPolicy；
- 弃用 TopologyKeys: 废弃 Service 中的 topologyKeys 字段；
- 弃用 RuntimeClass: 从 v1.25 版本开始不再提供 node.k8s.io/v1beta1 API 版本中的 RuntimeClass。
- 弃用 PodSecurityPolicy: 在 1.21 版本进入废弃阶段，预计在 1.25 中被删除。

参考链接

关于 Kubernetes 1.26 与其他版本详细对比可参考:

- kubernetes 1.25 changelog
- kubernetes 1.26 changelog

CCE发布Kubernetes 1.24版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证，本文介绍 CCE 发布 Kubernetes 1.24 版本相对 1.22 所做重要变更说明。

资源变更和弃用

- autoscaling/v1 被弃用，推荐使用 autoscaling/v2
- flowcontrol.apiserver.k8s.io/v1beta1 被弃用，推荐使用 flowcontrol.apiserver.k8s.io/v1beta2
- csistoragecapacities 推荐使用storage.k8s.io/v1 ，弃用 storage.k8s.io/v1beta1

其他变更和弃用

- kube-scheduler、 kube-controller-manager配置--authorization-always-allow-paths新增加设置/metrics

CCE集群从1.22升级到1.24，如果需要采集metrics数据，需要注意采集端口号的变动并采用https协议访问

- kube-scheduler端口：10259
- kube-controller-manager端口：10257

新增运行时contianerd1.6.8，不再支持docker

- Kubernetes 1.24通过Dockershim对Docker的支持现已移除，但是Docker构建的镜像可以继续使用，新建的节点请使用Containerd作为容器运行时。更多信息，请参见Updated: Dockershim Removal FAQ。

参考链接

关于 Kubernetes 1.24 与其他版本详细对比可参考:

- kubernetes 1.23 changelog
- kubernetes 1.24 changelog
- Kubernetes Removals and Deprecations In 1.24

CCE发布Kubernetes 1.22版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证，本文介绍 CCE 发布 Kubernetes 1.22 版本相对 1.20 所做重要变更说明。

资源变更和弃用

- storage.k8s.io/v1beta1 被弃用，推荐试用 storage.k8s.io/v1；
- scheduling.k8s.io/v1beta1 被弃用，推荐使用 scheduling.k8s.io/v1；
- certificates.k8s.io/v1beta1 被弃用，推荐使用 certificates.k8s.io/v1；
- coordination.k8s.io/v1beta1 被弃用，推荐使用 coordination.k8s.io/v1；
- apiextensions.k8s.io/v1beta1 被废弃，推荐使用 apiextensions.k8s.io/v1；
- authorization.k8s.io/v1beta1 被废弃，推荐使用 authorization.k8s.io/v1；

- authentication.k8s.io/v1beta1 被弃用，推荐使用 authentication.k8s.io/v1；
- apiregistration.k8s.io/v1beta1 被废弃，推荐使用 apiregistration.k8s.io/v1；
- rbac.authorization.k8s.io/v1beta1 被弃用，推荐使用 rbac.authorization.k8s.io/v1；
- admissionregistration.k8s.io/v1beta1 被废弃，推荐使用 admissionregistration.k8s.io/v1；
- Ingress and IngressClass 推荐使用 networking.k8s.io/v1，Ingress 和 IngressClass 在 extensions/v1beta1 及 networking.k8s.io/v1beta1 被弃用，在 1.22 彻底废弃。

其他变更和弃用

- 弃用 TopologyKeys: 废弃 Service 中的 topologyKeys 字段；
- 弃用 PodSecurityPolicy: 在 1.21 版本进入废弃阶段，预计在 1.25 中被删除。

参考链接

关于 Kubernetes 1.22 与其他版本详细对比可参考:

- kubernetes 1.21 changelog
- kubernetes 1.22 changelog

CCE发布Kubernetes 1.20版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证，本文介绍CCE发布Kubernetes 1.20版本相对1.18版本所做重要变更说明。

资源变更和弃用

- 删除selfLink字段, 该字段在ObjectMeta中, 由系统自动生成且只读；
- autoscaling/v2beta1被废弃，推荐使用autoscaling/v2beta2；
- storage.k8s.io/v1beta1被废弃，推荐使用storage.k8s.io/v1；
- apiextensions.k8s.io/v1beta1被废弃，推荐使用apiextensions.k8s.io/v1；
- apiregistration.k8s.io/v1beta1被废弃，推荐使用apiregistration.k8s.io/v1；
- coordination.k8s.io/v1beta1在1.19版本被弃用, 1.22版本将移除，推荐使用coordination.k8s.io/v1；
- kubescheduler.config.k8s.io/v1alpha2被废弃，推荐使用kubescheduler.config.k8s.io/v1beta1；
- authentication.k8s.io/v1beta1和authorization.k8s.io/v1beta1被弃用，将在1.22版本被移除，推荐使用authentication.k8s.io/v1和authorization.k8s.io/v1；
- Ingress和IngressClass推荐使用networking.k8s.io/v1，Ingress和IngressClass在extensions/v1beta1及networking.k8s.io/v1beta1被弃用，将在1.22版本彻底废弃。

其他变更和弃用

- kube-apiserver配置--enable-admission-plugins禁止设置PodPreset；
- Docker运行时在1.20版本被标记为废弃，目前您可以继续在集群中使用Docker，社区计划在后续版本中移除。

部分CCE集群从1.16升级到1.20，kube-apiserver需要新增如下配置，若升级失败可提交工单联系管理员处理：

```
 --service-account-issuer=https://kubernetes.default.svc.cluster.local \
--service-account-signing-key-file=/etc/kubernetes/pki/ca-key.pem \
--api-audiences=kubernetes.default.svc \
```

参考链接

关于Kubernetes 1.20版本与其他版本详细对比可参考:

- kubernetes 1.19 changelog
- kubernetes 1.20 changelog

CCE发布Kubernetes 1.18版本说明

百度智能云容器引擎CCE严格遵循社区一致性认证，本文介绍CCE发布Kubernetes 1.18版本相对1.16版本所做重要变更说明。

资源变更和弃用

- 不再使用apps/v1beta1和apps/v1beta2资源版本，使用apps/v1替代;
- Networkpolicies资源的API extensions/v1beta1被废弃，使用networking.k8s.io/v1替代；
- Podsecuritypolicies资源的API extensions/v1beta1将被弃用，使用policy/v1beta1代替；
- Deployment/Daemonset/Replicaset资源的API extensions/v1beta1被废弃，使用apps/v1替代。

其他废弃和移除

- 废弃CSINodeInfo特性，该特性达到GA并默认开启；
- 移除内置的ClusterRole: system:csi-external-provisioner，system:csi-external-attacher；
- 新增IngressClass资源描述，用于设定Ingress控制器名称，替换废弃的kubernetes.io/ingress.class标注；
- API组rbac.authorization.k8s.io/v1alpha1和rbac.authorization.k8s.io/v1beta1计划在1.20版本中废弃，请迁移至rbac.authorization.k8s.io/v1。

cloud provider标签功能达到GA阶段，废弃标签及新标签如下表所示:

废弃标签	新标签
beta.kubernetes.io/instance-type	node.kubernetes.io/instance-type
failure-domain.beta.kubernetes.io/region	topology.kubernetes.io/region
failure-domain.beta.kubernetes.io/zone	topology.kubernetes.io/zone

参考链接

关于Kubernetes 1.18版本与其他版本详细对比可参考:

- kubernetes 1.17 changelog
- kubernetes 1.18 changelog

安全漏洞修复公告

漏洞CVE-2025-1097、CVE-2025-1098等修复公告

Kubernetes社区披露了Nginx Ingress相关的一系列安全漏洞，包括CVE-2025-1097、CVE-2025-1098、CVE-2025-1974、CVE-2025-24513、CVE-2025-24514等。

漏洞详情

漏洞编号	漏洞等级	漏洞描述
CVE-2025-1097	高危漏洞	有Ingress资源写权限的攻击者可以通过Nginx Ingress社区提供的auth-tls-match-cn Annotation向Nginx注入配置，从而在ingress-nginx controller上下文中执行任意代码，并进一步获取整个集群维度的Secrets。
CVE-2025-1098	高危漏洞	有Ingress资源写权限的攻击者可以通过Nginx Ingress社区提供的mirror-target和 mirror-host Annotation向Nginx注入配置，从而在ingress-nginx controller上下文中执行任意代码，并进一步获取整个集群维度的Secrets。
CVE-2025-1974	高危漏洞	在一定条件下，能访问集群内网的攻击者可以利用Nginx Ingress的验证准入控制器（ValidatingAdmissionWebhook）功能实现配置注入，从而在ingress-nginx controller的上下文中执行任意代码，并进一步获取整个集群维度的Secrets密钥。
CVE-2025-24513	中危漏洞	Nginx Ingress Controller不对Ingress资源写权限持有者提交的输入数据进行充分验证与过滤。攻击者可利用此漏洞构造恶意请求，将非法数据注入配置文件的生成路径中，从而触发容器内的目录遍历漏洞。该漏洞可能导致拒绝服务，或者与其他漏洞结合使用，导致集群内有限Secrets实例的泄露。
CVE-2025-24514	高危漏洞	有Ingress资源写权限的攻击者可以通过Nginx Ingress社区提供的auth-url Annotation向Nginx注入配置，从而在ingress-nginx controller上下文中执行任意代码，并进一步获取整个集群维度的Secrets。

影响范围

受漏洞影响的开源NGINX Ingress控制器版本范围如下：

- <v1.11.5
- v1.12.0

开源NGINX Ingress控制器在下列版本中修复了该问题：

- v1.11.5
- v1.12.1

判断方法

未安装CCE Ingress Nginx Controller组件的集群不在上述漏洞的影响范围内。您可以参见以下内容查询集群是否安装了该组件，组件版本发布记录请见 CCE Ingress NGINX Controller 说明。

> **说明：**
> 容器引擎CCE针对Nginx Ingress安全漏洞进行了修复，使用CCE Ingress Nginx Controller组件在2025年04月02日后创建的实例不受漏洞影响。

方法1:

1. 登录百度智能云控制台，在左侧导航栏选择集群列表。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择运维管理 > 组件管理。
3. 在组件管理页面，您可以搜索并定位CCE Ingress Nginx Controller组件，在组件卡片上查看组件是否已安装以及组件的当前版本。

方法2:

1. 登录百度智能云CCE控制台，在左侧导航栏选择Helm > Helm实例。
2. 在Helm实例页面，选择目标集群，查看是否有cce-ingress-nginx-controller模板部署的实例，其中模板版本为当前CCE Ingress Nginx Controller组件的版本。

修复方案

在升级到漏洞修复版本前，您可以手动关闭CCE Ingress Nginx Controller组件的Admission Webhook功能以降低风险。

1. 登录百度智能云CCE控制台，在左侧导航栏选择Helm > Helm实例。
2. 在Helm实例页面，选择cce-ingress-nginx-controller模板部署的目标实例，单击更新操作。

3. 在更新页面增加以下参数完成关闭Admission Webhook功能。

```
 controller:
admissionWebhooks:
 enabled: false //false表示关闭，true表示开启
```

CCE Ingress Nginx Controller在 v4.11.5 版本修复了相关漏洞，更多信息请见 CCE Ingress NGINX Controller 说明，请在业务低峰期将组件升级至 v4.11.5 及以上版本，若需升级请 提工单。

> **重要：**
>
> 升级完成后，请务必重新启用Admission Webhook功能。该功能作为Ingress配置的预验证机制，可有效提升服务可靠性和稳定性。在您创建或更新Ingress配置生效前，Admission Webhook能够及时提醒Ingress配置中存在的错误，避免不必要的麻烦。

修复漏洞CVE-2021-30465公告

漏洞详情

CVE 编号：CVE-2021-30465

涉及组件：runc

漏洞描述：CVE-2021-30465漏洞为高危漏洞，CVSS漏洞评分为7.6。漏洞详细信息参考官方公告。

修复策略：将runc升级至1.0.0-rc95及以上版本。

影响范围

- CCE集群容器运行时为Docker的节点均受影响。
- CCE集群容器运行时为Containerd的节点不受影响。

修复方案

- 2022年4月13日后新建的节点已修复该漏洞。
- 2022年4月13日前的存量节点，可手动升级替换runc版本至v1.0.0-rc95，参考runc v1.0.0-rc95。

漏洞CVE-2020-14386修复公告

漏洞描述

近日，百度云安全团队监测发现 Linux 发行版存在一个权限提升漏洞(CVE-2020-14386)，漏洞详情参考CVE-2020-14386。 攻击者可以利用该漏洞进行越界写操作，进而造成权限提升、容器逃逸以及内存访问错误导致系统崩溃的风险。

该漏洞出现在 net/packet/af_packet.c 文件中，触发需要本地低权限用户或可执行文件具有 CAP_NET_RAW 权限。 一般非 root 用户无 CAP_NET_RAW 权限，但在较高版本 Linux 中非 root 用户可以通过创建 user namespace 的方式成为新 user namespace 中的 root 用户，进而获得该权限。 此外，K8S 创建出的 Docker 容器被默认拥有 CAP_NET_RAW 权限，因此存在触发漏洞的可能性。

漏洞编号：CVE-2020-14386
漏洞等级：高危

影响范围

Linux kernel 4.6 至 5.9-rc4

受影响的发行版系统：

- Ubuntu 18.04 及后续版本
- Centos 8/RHEL 8
- Debian 9/10

相关安全建议

升级 Linux Kernel 版本 或 禁用容器 CAP_NET_RAW 权限。

升级 Linux Kernel 版本

- 升级内核至安全版本或应用修复补丁
  内核下载地址： https://github.com/torvalds/linux/releases
  补丁地址：https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=acf69c946233259ab4d64f8869d4037a198c7f06

禁用容器 CAP_NET_RAW 权限

- 在容器的 Spec 配置中删除容器的 CAP_NET_RAW 权限。

```
 spec:
containers:
   - name: your-container
     ...
     securityContext:
        capabilities:
           drop: ["NET_RAW"]
```

- 大多数情况下容器无需 CAP_NET_RAW 权限。K8S 在配置 PodSecurityPolicy（PSP）后，可通过PSP策略删除容器的 CAP_NET_RAW 权限。

```
 apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: mypsp
spec:
 ...
 requiredDropCapabilities:
  - 'NET_RAW'
```

漏洞CVE-2019-5736修复公告

runc容器逃逸漏洞修复方案（CVE-2019-5736）

解决方法

可以采用以下三种方法中选择合适的方式修复安全漏洞：

1. 新建 Kubernetes 集群，kubernetes 1.11.5 的 docker 版本已更新至 18.09.2，该版本已修复漏洞。
2. 节点替换，在集群中扩容入新的节点，容器漂移稳定后将旧节点缩容处理。扩容节点的 docker-runc 均已修复漏洞。该方案会造成容器漂移和业务流量风险。1.8 存量集群添加节点时，新节点的 docker 版本不变，runc 已打上漏洞补丁，1.11.1 及以上存量集群的新扩容节点，docker 已整体更新为 18.09.2 版本。
3. 单独升级 runc（1.11的 GPU 集群建议采用上面两种方案），从 docker-18 开始，docker-runc 更名为 runc，以下方案针对 docker-18 之前的版本。

- 定位 docker-runc 位置，并备份。

```
 $ which docker-runc # 正常为 /usr/bin/docker-runc
$ mv /usr/bin/docker-runc /usr/bin/docker-runc.orig.$(date -Iseconds)
```

- 执行以下执行，下载修复的 runc：

```
 $ curl -o /usr/bin/docker-runc https://baidu-container.cdn.bcebos.com/packages/runc/runc-v1.0.0-rc2
$ chmod +x /usr/bin/docker-runc
```

- 检查 runc 是否正常执行

```
 $ docker-runc -v
runc version 1.0.0-rc2
commit: b1adfd870be901735b1509e9560eec508bdfca1f
spec: 1.0.0-rc2-dev
```

- 检查 docker 是否正常运行（因版本和负载差异，显示会有区别）：

```
 $ docker version
Client:
 Version:      17.03.2-ce
 API version:  1.27
 Go version:   go1.7.5
 Git commit:   f5ec1e2
 Built:        Tue Jun 27 01:35:00 2017
 OS/Arch:      linux/amd64
Server:
 Version:      17.03.2-ce
 API version:  1.27 (minimum version 1.12)
 Go version:   go1.7.5
 Git commit:   f5ec1e2
 Built:        Tue Jun 27 01:35:00 2017
 OS/Arch:      linux/amd64
 Experimental: false
```

检查容器运行状态：

```
 $ docker ps
CONTAINER ID        IMAGE                                                                                                COMMAND                  CREATED
STATUS              PORTS               NAMES
52d0c793caf9        hub-readonly.baidubce.com/public/pause:2.0                                                           "/pause"                 3
hours ago           Up 3 hours                              k8s_POD_node-exporter-klfks_kube-system_c8560cf5-3014-11e9-88b7-fa163e8b0a7e_0
9b6d830d4ba3        hub-readonly.baidubce.com/public/pause:2.0                                                           "/pause"
3 hours ago         Up 3 hours                              k8s_POD_csi-bosplugin-g9ctz_kube-system_c855cf61-3014-11e9-88b7-fa163e8b0a7e_0
```

安全建议

由于该漏洞需要利用容器内的 UID 0 用户进行攻击，也可以通过修改容器内的默认用户为非 root 用户，或通过 Kubernetes 的安全限制来达到屏蔽漏洞的目的。

- 通过**securityContext** 限制 UID 为非 0

```
 apiVersion: v1
kind: Pod
metadata:
 name: run-as-uid-1000
spec:
 securityContext:
  runAsUser: 1000
```

- 利用 PodSecurityPolicy 限制容器的权限

```
 apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: non-root
spec:
 privileged: false
 allowPrivilegeEscalation: false
 runAsUser:
  # Require the container to run without root privileges.
  rule: 'MustRunAsNonRoot'
```

此外，也要确保所有的容器镜像都经过审查和且可信的。这可以通过自己构建所有镜像，或者通过审查镜像的内容然后固定到镜像的版本哈希来实现。

关于 runc 的安全问题（CVE-2024-21626）影响声明

漏洞描述：

2024年1月31号，runc官方发布安全通告（https://github.com/opencontainers/runc/security/advisories/GHSA-xr7r-f8xq-vfvv ）。在 runc 1.1.11 及之前的版本中，由于内部文件描述符泄露的问题，攻击者可以让新生成的容器进程在宿主文件系统命名空间中拥有工作目录，从而允许通过访问宿主文件系统实现容器逃逸等。

漏洞利用需要具体的环境，例如允许攻击者提供自定义镜像等。官方已于2024年2月1日发布安全更新，建议相关受影响用户升级。 **影响范围** 1.0.0-rc93 <= runc <= 1.1.11 **安全版本** runc 1.1.12

解决建议：

1、官方已于2024年2月1日发布安全更新（runc 1.1.12版本），建议相关受影响用户升级。
https://github.com/opencontainers/runc/releases/tag/v1.1.12

2、百度智能云 CCE 容器服务 影响情况及修复进展

- 对于2024年2月19日之后创建的新增 CCE 集群和节点不受该漏洞影响。
- 对于集群存量节点，可通过在机器上执行以下命令进行runc升级：

```
wget https://baidu-container.cdn.bcebos.com/runtime/containerd/cce-cve-2024-21626.tar.gz && tar -zxf cce-cve-2024-21626.tar.gz && cd cce-cve-2024-21626 && bash upgrade-runc-v1.1.12.sh
```

3、如无法升级到修复版本，官方缓解措施参考如下（实施前请评估对业务的影响）：

- 设置容器的WORKDIR为/
- 仅允许用户运行受信任的镜像
- 不要运行exec

参考链接：

https://github.com/opencontainers/runc/releases/tag/v1.1.12
https://github.com/opencontainers/runc/commit/02120488a4c0fc487d1ed2867e901eeed7ce8ecf
https://github.com/opencontainers/runc/security/advisories/GHSA-xr7r-f8xq-vfvv

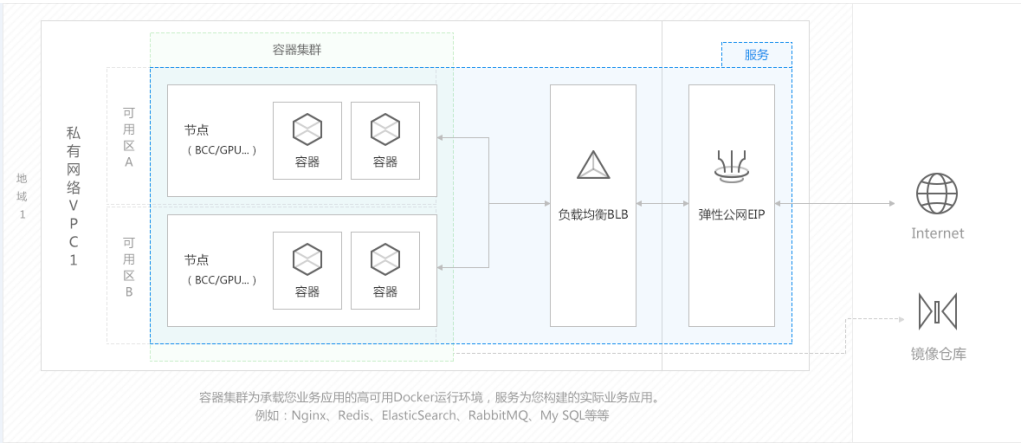
产品描述

介绍

百度智能云容器引擎(Cloud Container Engine，即CCE)是高度可扩展的高性能容器管理服务，您可以在托管的云服务器实例集群上轻松运行应用程序。使用CCE，您将无需安装、运维、扩展您的集群管理基础设施，只需进行简单的API调用，便可启动和停止Docker应用程序，查询集群的完整状态，以及使用各种云服务。您可以根据您的资源需求和可用性要求在您的集群中部署容器，满足业务或应用程序的特定要求。

产品架构

下图介绍百度智能云容器引擎服务的设计和实现过程，产品架构图如下所示：



在架构图中容器是实现操作系统虚拟化的方式，可以在资源隔离的条件下运行程序及其相关组件，所调用的镜像都储存在镜像仓库中。

服务指的是构建的实际业务应用，它是由集群、节点、容器、负载均衡BLB和弹性公网IP组合而成，因此您可以在托管的云服务器实例集群上轻松运行应用程序。使用该服务，您将无需安装、运维、扩展您的集群管理基础设施，只需进行简单的API调用，便可启动和停止Docker应用程序，查询集群的完整状态，以及使用各种云服务。您可以根据您的资源需求和可用性要求在您的集群中部署容器，满足业务或应用程序的特定要求

核心概念

集群

集群为一组百度智能云资源集合，这些云资源包含运行百度智能云容器引擎 CCE 的必要条件，通常由若干云服务器 BCC、GPU 服务器、FPGA 服务器、弹性裸金属服务器 BBC 组成。

节点

节点为注册到容器集群的单一服务器资源、同时包含了运行容器的一切必要条件。

镜像

Docker 镜像，用于部署容器服务，允许用户自定义镜像名。

镜像仓库

用户专有的 Docker 镜像存储空间，该存储空间运行在百度智能云容器服务上，由百度智能云负责保障性能和安全可靠。

容器

容器对应 Kubernetes 的 Pod 概念，容器拥有独立的存储、IP、端口、namespaces、配置，是百度智能云容器服务部署的最小单元。

服务

服务为一组配置相同的容器以及如何访问他们的规则，即微服务架构中的部署单元，即是用户业务应用的最小运行单元。

优势

简单易用

提供超大规模容器集群管理、资源调度、容器编排，屏蔽了底层基础构架的差异，简化了分布式应用的管理和运维。

安全可靠

相比较于自建集群，用户通过CCE集群可以独享计算资源，且集群在私有网络中运行，服务故障自动恢复快，迁移方便。

高效

使用CCE的用户镜像极速上传下载海量容器秒级启动,使用容器服务进行构建、测试、打包集成，而自建容器服务的网络无保证，因此无法保证使用镜像创建容器的效率。

低成本

相比独立集群或者自建集群，您只需花费少量费用即可获得高稳定、高可靠、可扩展的集群管理面，免去运维烦恼。

AI加速

支持多种深度学习框架容器化部署，为 AI 应用提供了极速部署和启动、资源隔离、动态调度、高可用、开发训练测试环境一致的平台服务。

特性

集群管理

通过百度智能云容器引擎可简单高效地管理用户的容器集群，安全可靠，无缝衔接百度智能云计算、存储、网络服务。

- 支持 BCC 云服务器、GPU 云服务器、BBC 裸金属服务器、DCC 专属云服务器、BEC 边缘计算节点、BCI 容器实例等计算资源。
- 支持 Master 托管或独立部署，选择多副本；支持 Master 和 Worker 节点跨可用区部署，实现高可用；支持新建节点或者添加已有节点创建集群。
- 支持多个地域，可以根据实际需求选择，可选择专有网络 VPC 环境。

服务管理

服务即是用户的业务应用，服务管理为用户提供高效的容器管理方案，支持服务的快速创建、快速扩缩容、负载均衡、服务发现、服务监控、健康检查等特性，您可以通过服务管理方便快捷的管理您的容器。

容器编排

- 解决容器集群数量大、拓扑关系复杂、缓存、调用、高可用设计交错的管理运维同步问题。

- 提供跨主机容器化应用的部署、高可用、弹性伸缩、监控、通信等功能。
- 提供了一套完整、易用的 RESTful API 来封装功能。
- 提供集群自恢复机制（自动重启、自动重调度、自动备份等），维护集群一直处于用户期望的状态。
- 用户只需要关心资源申请，Kubernetes 自动化完成调度、分配、高可用、底层适配等。

应用场景

微服务架构使用

使用百度智能云容器引擎 CCE 将服务模块进行拆分形成更小的服务模块，模块之间独立部署升级，便于应用者管理。

每个容器里可以使用完全不同环境的镜像服务，容器启动即产生了一个独立的微服务主机节点（独立的网络IP），上层服务与下层服务之间服务发现通过环境变量注入、配置文件挂载等多种方式灵活解决，而且还可以直接将云平台提供的各种云服务与自定义的微服务整合组成一个强大的服务集群。

更重要的是，拥有如此多服务的集群环境迁移、复制也非常轻松，只需选择好各服务对应的 Docker 服务镜像、配置好相互之间访问地址就能很快搭建出一份完全一样的新集群。

人工智能技术的输出

支持深度学习在云容器上的良好运行，使用容器引擎可以提供完备、稳定、高可用 K8S 集群服务和大规模计算存储能力，同时嫁接 GPU、NPU、FPGA 能力。便于差异化输出百度无人车等 AI 能力，而且还为百度 AI 能力在企业用户的落地解决好极速部署和启动、资源共享和隔离、任务自动调度、故障恢复、标准化模型训练到上线的工作流等场景。

提升资源利用率，降低成本

有些情况下用户只是需要执行一次性的任务，例如计算出某个结果即可，如果采用传统的服务模式，服务器需要一直运行，造成的极大的资源浪费。而使用 CCE 可以进行容器的快速创建、销毁能很灵活满足这种完全按需付费的场景。只要制作好一次性程序运行的 Docker 镜像，当需要运行的时候实时通过镜像创建出容器来执行任务，程序执行完成容器自动退出释放资源。

持续集成/持续交付

通过持续集成与持续交付提供优秀的 DevOps 环境，极大提高软件发布效率。

持续集成：持续集成强调开发人员提交了新代码之后，立刻进行构建、（单元）测试。根据测试结果，我们可以确定新代码和原有代码能否正确地集成在一起。

持续交付：持续交付在持续集成的基础上，将集成后的代码部署到运行环境。

通过在百度智能云容器引擎上部署服务，开发人员在 git 或其他代码平台提交新代码后，可立即进行构建、测试、打包集成。在持续集成的基础上，将集成的代码部署到预发布环境和线上环境。

使用限制

本文将介绍在使用容器引擎CCE集群过程中可能会涉及到云产品以及对应的限制项，方便您在使用前根据需要统一调整，避免因配额限制导致的使用问题。

CCE集群配额限制

限制项	默认配额	是否可提升配额
每个地域集群配额	20个	支持，您可以提工单提升配额
单集群节点规模	200个	
单集群快照配额	50个	

> **说明：** 您可以在容器引擎CCE控制台上查看当前配额和已使用量。

依赖底层云产品配额限制

限制大类	云产品	使用场景	限制项	默认配额	是否可提升配额
计算	云服务器	作为集群Node节点使用	每个地域后付费BCC实例	50个	支持，更多信息请见云

	BCC	• 作为集群Node节点使用的服务器资源	数量		服务器BCC使用限制
	弹性裸金属服务器BBC		每个地域下BBC实例数量	2个	支持
			每个账号下后付费BBC实例数量	1个	
网络	私有网络VPC	• 作为集群Node节点、容器实例Pod（VPC-ENI）、Service、Ingress使用的网段	每个地域下实例数量	10个，其中包括一个默认VPC	支持，更多信息请见私有网络VPC产品规格限制
	子网	• 作为Node节点使用的网段 • 作为容器实例Pod（VPC-ENI网络模式）使用的网段 • 作为LB Service使用的网段 • 作为LB Ingress使用的网段	每个VPC可创建的子网数量	10个	
	路由表	• VPC路由容器网络模式下实现Node和Pod互相连接	每个路由表可添加的路由规则数量	50条	
	安全组	• 绑定到Node节点服务器上控制节点进出流量 • 绑定到节点服务器挂载的弹性网卡ENI上控制Pod进出流量（VPC-ENI网络模式）	每个地域下企业安全组数量	100个	
			每个默认VPC可创建的安全组数量	100个	
			每个普通VPC可创建的安全组数量	20个	
			每个普通安全组关联实例数量	2000个	
	弹性网卡	• 作为VPC-ENI容器网络模式下为Pod分配弹性网卡上的辅助IP地址	每个VPC可创建的未挂载主机的弹性网卡数量	500个，已挂载主机的弹性网卡不占用配额	
			云主机可挂载的弹性网卡数量	min（主机核数，8）	
			云主机绑定的弹性网卡可配置的IP数量	与主机内存相关，如下图所示：	
				内存 / IP数量	
				1G / 2	
				(1-8]G / 8	
				(8-32]G / 16	
				(32-64]G / 30	
				大于64G / 40	
	弹性公网IP	• 绑定到Node节点服务器上为节点开启公网访问能力 • 绑定到LB Service对应的BLB上为Service开启公网访问能力	每个地域下标准型BGP EIP实例数量	50个（个人用户），130个（企业用户）	支持，更多信息请见弹性公网IP购买须知
			每个地域下增强型BGP EIP实例数量	20个	
			单个标准型BGP EIP最大带宽峰值	200Mbps（按使用流量计费），500Mbps（按固定带宽计费）	
			单个增强型BGP EIP最大	1000Mbps（按使用流量计	

		• 绑定到LB Ingress对应的BLB上为Ingress开启公网访问能力	带宽峰值	费），5000Mbps（按固定带宽计费）	
			单个地域下所有按流量计费标准型 BGP EIP累计带宽峰值总和	5000Mbps	
			单个地域下所有按带宽计费标准型 BGP EIP累计带宽峰值总和	10000Mbps	
			增强型BGP EIP起购带宽	100Mbps	
	负载均衡BLB	• 绑定到Service服务上向公网或内网暴露服务 • 绑定到Ingress路由上向公网或内网暴露服务	每个地域下BLB实例数量	60个	支持，更多信息请见负载均衡BLB使用限制
			每个BLB实例可添加的监听数量	50个	
			每个HTTPS监听器可添加的扩展域名数量	10个	
			每个应用型BLB实例服务器组数量	20个	
			每个应用型BLB实例服务器组内服务器数量	50个	
			每个应用型BLB实例规则条件数量	5个	
			每个应用型BLB实例可添加的规则数量	10个	
			每个应用型BLB实例IP组内成员数量	50个	
			每个应用型BLB实例IP组数量	10个	
			每个应用型BLB实例健康检查数量	20个	
存储	云磁盘CDS	• 挂载到Node节点上直接作为数据盘存储Kubelet、容器数据使用 • 通过PV方式挂载到Node节点上作为容器的持久化数据卷使用	每个地域云磁盘总量	160TB	支持，更多信息请见云磁盘CDS产品介绍
			每个地域CDS实例数量	n*5（n为单个地域下BCC个数）	
			每台云服务器BCC最多可挂载数据盘数量	5个	

> **说明：**您可以在配额管理控制台上查看相关产品的当前配额和已使用量。

典型场景所需云产品配额示例

场景1：VPC路由模式下创建1000个按量付费BCC节点所必须的产品及资源数量，其中每个节点挂载一块100G系统盘，2块200GB数据盘

示例：

限制大类	依赖云产品	限制项	默认配额	实际所需数量	是否调整默认配额
云原生	容器引擎CCE	单集群节点规模	200	1000	是
计算	云服务器BCC	每个地域后付费BCC实例数量	50	1000	是
网络	路由表	每个路由表可添加的路由规则数量	50	1000	是
	安全组	每个普通安全组关联实例数量	100000	1000	否
存储	云磁盘CDS	每个地域云磁盘总量	160TB	500TB	是
		每个地域CDS实例数量	5000	3000	否
		每台云服务器BCC最多可挂载数据盘数量	5	3	否

场景2：创建100个LB Service，每个Service后端关联1000 Pod（直联Pod模式），每个BLB绑定一个按流量计费，带宽大小为1000Mbps标准型EIP

示例：

限制大类	依赖云产品	限制项	默认配额	实际所需数量	是否调整默认配额
网络	弹性公网IP	每个地域下标准型BGP EIP实例数量	50个（个人用户），130个（企业用户）	100	是（个人用户）
		单个标准型BGP EIP最大带宽峰值	200Mbps	1000Mbps	是
		单个地域下所有按流量计费标准型 BGP EIP累计带宽峰值总和	5000Mbps	100000Mbps	是
	负载均衡BLB	每个地域下BLB实例数量	60	100	是
		每个BLB实例可添加的监听数量	50	1	否
		每个应用型BLB实例IP组内成员数量	50	1000	是
		每个应用型BLB实例IP组数量	10	1	否

产品定价

产品定价

百度智能云容器引擎 CCE 免费为用户提供服务，用户只需为使用的资源（如 BCC、BLB、EIP）来进行付费，具体计费标准可以参考 BCC 价格计算器、BLB 定价、EIP 定价。

快速入门

使用CCE容器引擎流程概述

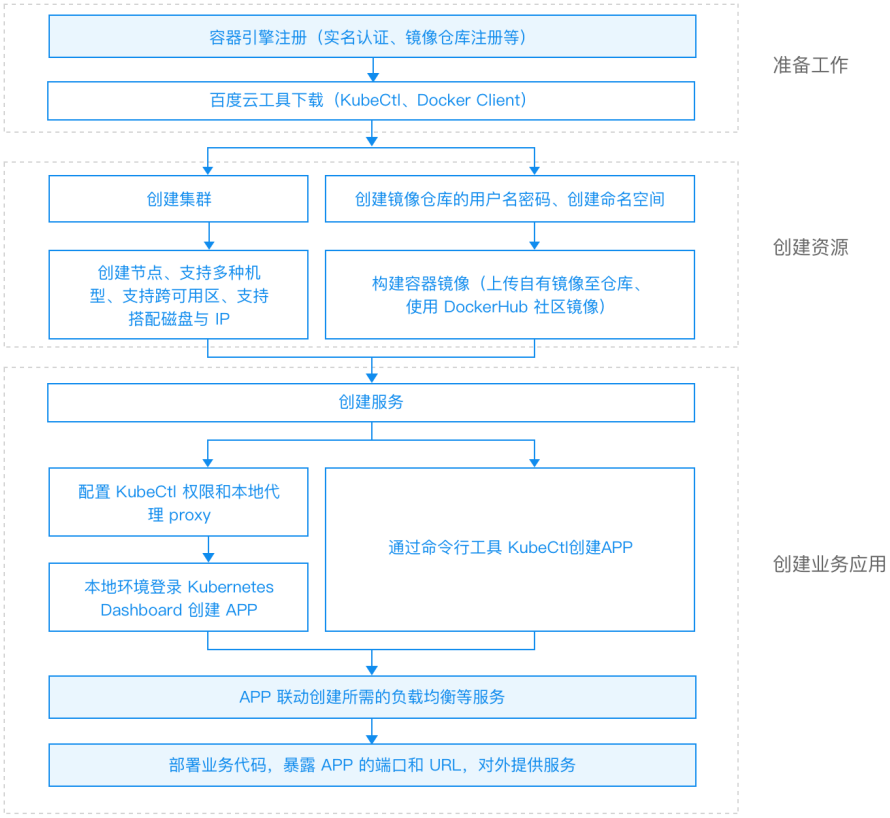
本文用简单、直观的图示，让您清晰地了解百度智能云容器引擎 CCE 的使用流程，助您快速上手使用。使用流程主要分为以下三步：

第一步：准备工作，包含注册百度云账号、注册镜像仓库（推荐使用 容器镜像服务CCR）、下载容器引擎服务的操作工具；

第二步：创建资源，包含创建集群、节点、镜像、命名空间；

第三步：创建业务应用，包含创建服务、配置负载均衡并对外提供服务。

各步骤的详细操作请查看“操作指南”，使用过程中可能遇到的问题和解决方法请查看“常见问题”。



快速部署nginx应用

本文将介绍如何使用百度智能云容器引擎CCE快速部署一个Nginx应用，并通过Load Balancer类型的Service暴露到公网。

操作步骤

创建Nginx工作负载

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品导览>容器>容器引擎**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**工作负载>无状态**"进入无状态列表页面。
6. 单击**新建无状态**按钮进入创建工作负载页面，根据以下信息，设置工作负载基本信息。

配置项	配置说明	示例
工作负载名称	填写工作负载的名称。	nginx
命名空间	选择工作负载部署的命名空间。	保持默认配置
工作负载类型	选择工作负载的类型。	保持默认配置
K8s标签（Label）	填写工作负载的标签（Label）信息。	保持默认配置（app = nginx）
注解（Annotation）	填写工作负载的注解（Annotation）信息。	保持默认配置（为空）
期望Pod数	输入无状态负载的 Pod 实例个数。	保持默认配置（2）

7. 设置容器配置。主要参数信息如下：

配置项	配置说明	示例
容器名称	输入容器名称。	nginx
镜像地址	输入不包含镜像版本（Tag）的镜像地址，或者单击**选择镜像**，选择已上传到容器镜像服务中的镜像。	registry.baidubce.com/cce/nginx-alpine-go
镜像版本（Tag）	选择镜像的版本，不选择则默认拉取**latest**版本。	latest
容器配额	支持按照业务需求配置 CPU 和内存的请求和上限。	保持默认配置

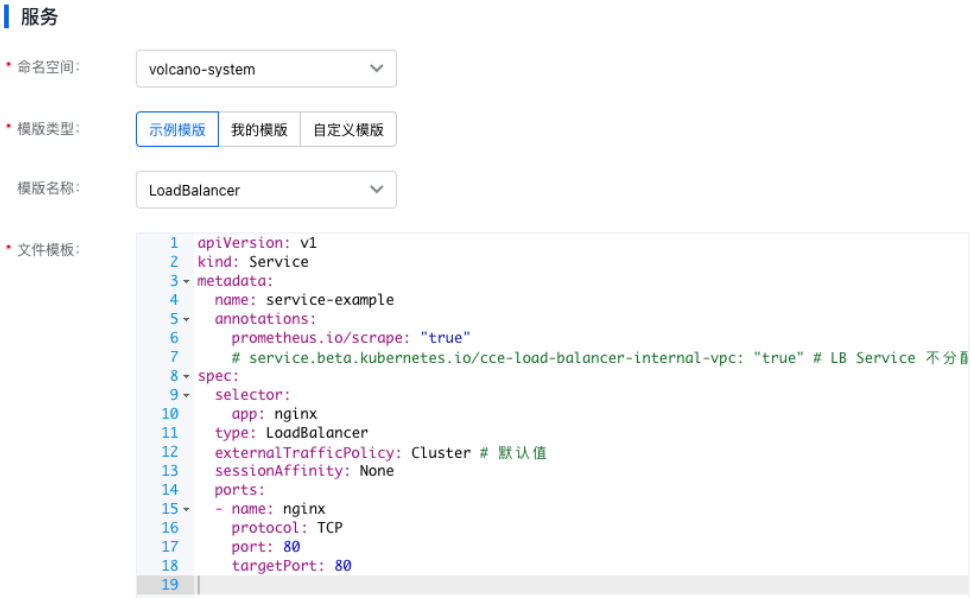
8. 单击**完成**按钮，完成工作负载创建。

创建Nginx服务

Service 定义了 Pod 的访问策略，通过 Service 的固定地址可以访问到后端 Pod。Service 支持多种访问类型，本例使用 Load Balancer 类型的 Service 将应用暴露到公网。

1. 在集群管理页面单击"**网络>服务**"进入服务列表页面。

2. 单击**新建服务**按钮进入创建服务页面，选择**示例模板>LoadBalancer**，将**selector**设置为**app: nginx**，其他配置保持不变。



3. 单击**确定**按钮，完成服务创建。

4. 创建成功后，可在服务列表中查看已创建的Nginx服务。

5. 可以通过服务列表中对应服务的外部端点访问该Nginx应用。

操作指南

使用须知

百度智能云容器引擎（CCE）提供标准托管类型集群，其中对容器相关的技术架构及核心组件提供托管服务，对于非托管组件及运行在CCE集群中的应用，不当的操作可能会导致组件或业务故障。为了您在云上业务的稳定可用，同时更好地预估和避免相关操作风险，在使用容器服务CCE前，请仔细阅读本文中的建议与注意事项。

控制面组件运维

- CCE托管集群的Master、Etcd及其他托管类型组件（如CCE LB Controller）由百度智能云技术团队负责管理和维护，同时CCE确保集群的高可用性，并根据需要进行自动或手动扩展。您只需要购置集群，运行负载所需的工作节点即可。
- Master、Etcd 不属于用户资源，您在该模式下无法自主修改 Master 和 Etcd 的部署规模和服务参数。

> **说明：**关于CCE独立集群，用户需要自行管理和运维Master和Etcd。

数据面组件运维

CCE托管集群的数据面组件运行在用户的节点（如BCC服务器）上，例如CoreDNS、CCE Network Plugin、Kube-Proxy、Kubelet等，因此数据面组件运行的稳定可用需要CCE与用户共同维护。

CCE对数据面组件提供以下支持：

- 提供组件的参数化设置、功能优化、BugFix、CVE 补丁等功能并给出指导文档。
- 提供监控、日志和报警等可观测能力，核心组件会提供组件日志，支持采集到日志服务BLS中。
- 若您在使用过程中出现其他问题，CCE原则上提供相关技术指导与支持。

您在使用数据面组件时，请遵循以下建议：

- 使用最新的组件版本。组件经常会发布新版本以修复BUG或提供新特性。CCE发布新的组件版本后，在保证业务稳定的前提下选择合适的时机，请您遵循组件升级指导文档中的说明进行升级操作。
- 部分数据面组件受上游开源社区版组件影响，可能存在Bug或漏洞，请注意及时升级组件，以避免开源组件Bug或漏洞导致您的业务受损。
- 请勿直接使用IaaS层产品的OpenAPI来变更集群组件的运行环境，包括但不限定于使用云服务器BCC的OpenAPI更改节点的运行状态、修改节点安全组配置、节点网络配置以及通过负载均衡器的OpenAPI修改集群关联的BLB配置等。擅自改动IaaS层资源可能会导致数据面组件异常。
- 在您管理数据面组件时，请通过容器引擎控制台集群管理页面**组件管理**模块的方式或者OpenAPI的方式配置组件的自定义参数。通过其他渠道修改组件配置可能会导致组件功能异常，并且参数可能在某些情况会被自动覆盖。
- 建议在集群中开启容器可观测，并配置相应基础资源和业务资源的告警，CCE会将重要的集群控制面和数据面指标发送到Prometheus监控服务中。更多信息请参见使用Prometheus监控集群。

集群管理

创建集群

本文为您详细介绍通过控制台创建托管集群的步骤和参数配置。

准备工作

创建集群前，建议您根据业务需求规划并设计集群配置，以确保集群能够稳定、高效且安全地运行。

- 地域：所选地域与用户和资源部署地域的距离越近，网络时延越低，访问速度越快。
- 网络地址规划：根据业务场景和集群规模规划 VPC 网段和 Kubernetes 网段（Pod地址段和Service地址段），定义整个集群的IP地址范围以及Pod和节点可用的IP地址数量。
- 公网访问：集群节点是否需要访问公网（拉取公共镜像时需开通公网）。

前提条件

在创建集群前，您需要完成以下工作：

- 注册百度账号，并完成实名认证。
- 集群创建过程中将使用或创建私有网络、子网、安全组等多种资源。资源所在地域具备一定的配额限制，更多信息请参见 使用限制。
- 如果想添加已有的云服务器（BCC）实例到集群中，请确保集群与云服务器实例在同一个私有网络（VPC）下。
- 创建集群过程中涉及负载均衡等按量资源的购买，请确保您的账户余额充足，以确保能够正常创建按量计费类型的资源。

注意事项

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。
- 变更集群所在地域。
- 变更集群的网络配置，如所在的私有网络VPC、子网、服务网段、IPv6、kube-proxy代理模式（即服务转发模式）。

费用说明

百度智能云容器引擎 CCE 免费为用户提供服务，但创建集群时可能会产生负载均衡、NAT 网关、公网 IP、云盘等云基础资源费用。请在创建集群页面下方 配置费用处，查看实际计费。更多计费说明及云基础资源计费，请参见 产品计费。

操作步骤

1. 登录容器引擎控制台，选择**产品服务 > 云原生 > 容器引擎 CCE**，在左侧导航栏选择**集群管理 > 集群列表**，进入集群列表页面。
2. 在集群列表页面，点击列表上侧的**创建集群**。
3. CCE为您预置了适合不同场景的集群创建模版，您可在**选择模版弹窗**中，根据模板介绍结合自身业务需求**选择模板**，点击**确认**进入创建集群页面。
4. 在创建集群页面中，根据集群创建导航并结合业务需求依次配置：集群配置、Master配置（仅创建标准独立集群需要配置）、Worker配置。

> **说明：**
>
> - **【推荐】标准托管集群：**集群Master由容器引擎CCE完全托管，您只需要购Worker节点来运行工作负载即可。
> - **标准独立集群：**集群Master将会部署在您购买的服务器实例上，需要自行规划、维护、升级服务器集群。

集群配置

基础配置

配置项	说明
集群名称	集群的名称。支持大小写字母、数字以及-_ /.特殊字符，必须以字母开头，长度1-65。
集群规模	若选择标准托管集群，需选择集群规模。集群支持管理的最大节点数量，请根据业务场景选择。可参见 如何选择集群规模。若需使用更大集群规模，请提工单联系。
Kubernetes 版本	选择当前支持的Kubernetes版本，推荐使用最新版本。可参见CCE Kubernetes版本更新说明了解CCE的版本支持情况。
容器运行时	Kubernetes集群的容器运行时组件，当前CCE支持的最新版本为 Containerd 1.6.36 和Containerd 1.7.25，可参见 如何选择 Kubernetes 集群的容器运行时组件。
节点命名模式	**内网 IP：**集群将以实例的内网IP作为节点名称（NodeName）。 • 注意：节点加入集群后，不支持变更 NodeName。
	主机名称：集群将以实例的主机名称（hostname）作为节点名称（NodeName），须保证加入集群的实例的主机名称不重复。 • 注意：节点加入集群后，不支持变更 NodeName。
付费方式	**包年包月：**先付费后使用，价格更低廉。关联创建的 BCC、BLB 和 EIP 资源，付费方式均为包年包月。 • 注意：若付费方式选择包年包月，还需设置购买时长，以及是否开启自动续费。
	按量付费：先使用后付费，按需开通。关联创建的 BCC、BLB 和 EIP资源，付费方式均为按量付费。
当前地域	指定集群所在的地域，处在不同地域的云产品内网不通，购买后不能更换。建议选择靠近您客户的地域，以降低访问延时、提高下载速度。

网络配置

容器网络为独立的地址空间，须与节点网络、节点子网、其他集群的容器网络相互独立。以下以创建标准托管集群时的网络配置项为例进行说明。

配置项	说明
IPv6 双栈网络	开启 IPv6 双栈网络，将筛选出已配置 IPv6 的 VPC 网络和子网，并自动分配 IPv6 的容器网段和服务网段。**注意：**选择后不支持变更，请谨慎选择，CCE 使用 IPv4/IPv6 双栈网络说明。
私有网络	设置集群的网络。私有网络为集群提供一个隔离的、自主配置和管理的虚拟网络环境。如需创建新的私有网络，可前往 创建私有网络。
	自动创建负载均衡器 BLB：系统将自动为集群 API Server 创建一个内网应用型负载均衡 BLB 实例，若删除该 BLB 实例会导致 API Server 无法访问。若选择自动创建负载均衡器 BLB，还需进行 API Server 子网和公网访问设置。

API Server 访问	Server 无法访问。若选择自动创建负载均衡器 BLB，还需进行 API Server 子网和公网访问设置。 • **API Server 子网：**Kubernetes API Server 将使用该子网创建负载均衡器，不能选择 NAT 子网，若无适合的子网，可前往 创建子网。 • **使用 EIP 暴露 API Server：**默认不开启，开启后将为集群 API Server 绑定公网 IP，否则无法通过外网访问 API Server。 • **注意：**系统默认创建按量付费的 BLB 和 EIP 实例，可手动转换计费方式，计费规则请参见 BLB 计费说明和EIP 计费说明。 **使用已有负载均衡器 BLB：**集群 API Server 将与已有的负载均衡 BLB 实例绑定，若删除该 BLB 实例会导致 API Server 无法访问。 • **注意：**仅支持应用型 BLB。若集群要开启公网访问，请确保选择的 BLB 实例已绑定弹性公网 IP，否则无法通过公网访问。
安全组	指定Worker节点的安全组，支持普通安全组和企业安全组： • **自动创建：**系统将自动为集群创建默认安全组（安全组名称为 {集群ID}-worker-{随机ID}）专用于绑定 Worker 节点，保障集群内节点之间正常通信。 • **使用已有：**您可以选择已有的安全组用于绑定Worker节点。 • **注意：**安全组必须放通指定端口来保证集群内节点之间正常通信，安全组端口配置说明请参见CCE默认安全组规则。 安全组创建之后不支持修改。

容器网络模式	支持 **VPC-ENI** 容器网络模式。**VPC-ENI：**基于弹性网卡实现的容器网络模式，容器网络与云主机网络在同一个 VPC 内。选择VPC-ENI模式，需选择设置弹性网卡模式、容器子网、ClusterIP 网段。CCE 提供**共享**和**独占**两种弹性网卡模式。
	• **弹性网卡模式：**不同实例规格的节点可绑定的弹性网卡数量以及弹性网卡可分配的辅助 IP 数量不同。更多内容请查看 弹性网卡配额。**注意：**集群创建之后不支持修改，
	  • **共享：**多个 Pod 共享一个弹性网卡，系统为弹性网卡申请多个辅助 IP 分配给不同的 Pod。若选择共享模式，还可选择是否开启 eBPF 增强、Network Policy 支持和 RDMA。
	  • **开启 eBPF 增强：**支持开启 eBPF 增强，开启后原生Kube-proxy 模式将被替换。**注意：**创建后不支持修改，镜像仅支持使用 BaiduLinux3.0 系统。
	  • **开启Network Policy 支持：**开启后支持为集群提供基于策略的网络控制，更多信息请见使用 Network Policy 网络策略。
	  • **独占：**每个 Pod 独占一个弹性网卡，基于 eBPF 技术提供更高网络性能，操作系统仅支持公共镜像 Ubuntu 20.04。独占弹性网卡模式目前为白名单开放，如需使用提工单申请。
	• **容器子网：** 集群创建的 Pod 将从容器子网中分配 IP 地址，请选择与节点同可用区的子网作为容器网络。如没有可选项，可以单击**去创建子网**。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。
	• **ClusterIP 网段：** 设置为集群中 Service 分配 IP 地址的专用网络地址范围。**注意：**
	  • ClusterIP 网段不能与节点网段冲突。
	  • 当前 ClusterIP 网段配置决定了Service资源的上限，创建后不支持修改。

LB Service 子网	设置创建 Service 及 Ingress 时关联创建BLB默认使用的子网，不能选择 NAT 子网。
弹性网卡安全组	**容器网络模式**选择 **VPC-ENI** 时，指定Worker节点挂载弹性网卡绑定的安全组，支持普通安全组和企业安全组： • **自动创建：**系统将自动为集群创建默认安全组（安全组名称为 {集群ID}-eni-{随机ID}）专用于绑定节点服务器上挂载的弹性网卡 ENI，保障集群内节点之间正常通信。 • **使用已有：**您可以选择已有的安全组用于绑定Worker节点，请确保不影响集群节点之间正常通信。了解更多CCE默认安全组规则。
Kube-proxy 模式	设置 Kube-proxy 代理模式，可选择 **IPTABLES** 或 **IPVS**。若集群**开启eBPF增强**时，则原生Kube-proxy模式将被替换，无需设置。 • **IPVS：**适用于集群存在大量Service，对负载均衡有高性能要求的场景。 • **IPTABLES：**成熟稳定的kube-proxy代理模式。Kubernetes Service的服务发现和负载均衡使用IPTABLES规则配置，但性能一般，受规模影响较大，适用于存在少量Service的集群。
Nodepor	可用于 NodePort 服务的端口号的范围。在 Kubernetes 集群中，默认 NodePort 范围是 30000-32767。

t 范围	

高级设置（可选）

配置项	说明
资源标签	可设置资源标签，标签支持按用途、所有者或项目等对资源进行分类。
集群删除保护	默认开启，开启后可以防止通过控制台或 API 误删除集群。
认证方式	默认使用常用的 X509 客户端证书认证，可选择开启 OIDC 认证，详情参见 集群认证说明。
自定义证书 SAN	在集群API Server服务端证书的SAN（Subject Alternative Name）字段中添加自定义的IP或域名，以实现对客户端的访问控制。集群创建后暂不支持更新。
集群备注	指定集群的描述信息。

Worker配置

若创建标准托管集群，需在创建集群页面进行 Work 配置。以下以标准托管集群为例，对 Work 配置的相关配置项进行说明。

节点配置

配置项	说明
节点来源	支持**新建节点**或**使用已有服务器**两种节点来源，若选择使用已有服务器则只支持选择当前集群私有网络下的可用实例。
Worker节点	点击**添加节点**进行 Worker 节点配置，节点配置相关参数请参见 添加节点。
资源标签	可设置资源标签，标签支持按用途、所有者或项目等对资源进行分类。可选择对其关联资源 CDS、EIP、快照、快照链统一加标签。

高级设置（可选）

配置项		说明
数据存储	kubelet数据目录	kubelet服务用来存储其运行时数据的文件系统目录，如 volume 文件、plugin 文件等数据存储目录。若已挂载数据盘，建议存储到数据盘。
	容器数据目录	容器、镜像等数据存储目录。若已挂载数据盘，建议存储到数据盘。
执行脚本	部署前执行脚本	在节点部署之前，系统会自动执行指定的脚本。请确保该脚本具备可重入性以及重试逻辑。脚本内容及其产生的日志将自动保存到节点的 **/usr/local/cce/scripts/** 目录中。
	部署后执行脚本	在节点部署完成后，系统将自动运行该脚本。您需要自行监控和验证脚本的执行状况。脚本内容及其产生的日志将自动保存到节点的 **/usr/local/cce/scripts/** 目录中。
节点参数	封锁节点（cordon）	默认不开启，若开启封锁节点，节点将处于不可调度状态，新创建的 Pod 将不会调度到该节点上。
	自定义 kubelet 参数	可根据特定需求或集群环境，手动设置 kubelet 的启动参数，更多信息请见自定义Kubelet参数。
	标签(Lables)	K8S 标签（Label）是管理和选择 K8S 对象的标识，将自动绑定创建的节点，每个 Label 包含键和值两部分。
	污点（Taints）	在 Kubernetes 集群中，污点（Taints） 是一种机制，用于阻止 Pod 被调度到特定的节点上。污点是添加到节点上的标记，它们可以指示节点应该避免被某些 Pod 访问。节点污点和 Pod 容忍共同作用。节点设置污点后，可避免 Pod 调度到节点上，或者将 Pod 从节点驱逐。除非 Pod 容忍能够与节点污点相匹配。污点由三部分组成： • **键（Key）**：污点的唯一标识符。 • **值（Value）**：与键相关联的字符串。 • **效应（Effect）**：污点应用的规则，有以下几种： 1. **NoSchedule**：没有容忍（Toleration）匹配该污点的 Pod 将不会被调度到该节点上。 2. **PreferNoSchedule**：推荐不调度没有容忍的 Pod 到该节点，但如果需要，仍然可以调度。 3. **NoExecute**：没有容忍的 Pod 将不会被调度到该节点上，如果已经在节点上运行，将被驱逐。

Master配置

若创建标准独立集群，需在创建集群页面进行 Master 配置。以下以标准独立集群为例，对 Master 配置的相关配置项进行说明。

节点配置

配置项	说明
节点来源	支持**新建节点**或**使用已有服务器**两种节点来源，若选择使用已有服务器则只支持选择当前集群私有网络下的可用实例。
Master节点	可选择 **3 副本**或 **5 副本**，若选择 3 副本则至少需要添加 3 个节点。点击**添加节点**进行 Master 节点配置，节点配置相关参数请参见添加节点。

服务器配置

配置项	说明
Master安全组	指定Master节点绑定的安全组，支持普通安全组和企业安全组。 • **自动创建**：系统将自动为集群创建默认安全组（安全组名称为 {集群ID}-worker-{随机ID}）专用于绑定 Worker 节点，保障集群内节点之间正常通信。 • **使用已有**：您可以选择已有的安全组用于绑定Master节点，请确保不影响集群节点之间正常通信。了解更多CCE默认安全组规则。
资源标签	可设置资源标签，标签支持按用途、所有者或项目等对资源进行分类。可选择对其关联资源 CDS、EIP、快照、快照链统一加标签。

高级设置（可选）

配置项		说明
数据存储	etcd 数据目录	在 etcd 服务器运行的机器上，用于存储 etcd 数据文件的文件系统目录。若已挂载数据盘，建议存储到数据盘。
	kubelet 数据目录	kubelet 服务用来存储其运行时数据的文件系统目录，如 volume 文件、plugin 文件等数据存储目录。若已挂载数据盘，建议存储到数据盘。
	容器数据目录	容器、镜像等数据存储目录。若已挂载数据盘，建议存储到数据盘。
执行脚本	部署前执行脚本	在节点部署之前，系统会自动执行指定的脚本。请确保该脚本具备可重入性以及重试逻辑。脚本内容及其产生的日志将自动保存到节点的 /usr/local/cce/scripts/ 目录中。
	部署后执行脚本	在节点部署完成后，系统将自动运行该脚本。您需要自行监控和验证脚本的执行状况。脚本内容及其产生的日志将自动保存到节点的/usr/local/cce/scripts/目录中。

确认配置

确认选择的配置信息和配置费用后，在**前置检查**模块通过前置检查后，点击**提交**，开始创建集群。

- 您可以在集群列表中查看创建集群的状态，创建成功后该集群状态将显示为“运行中”。

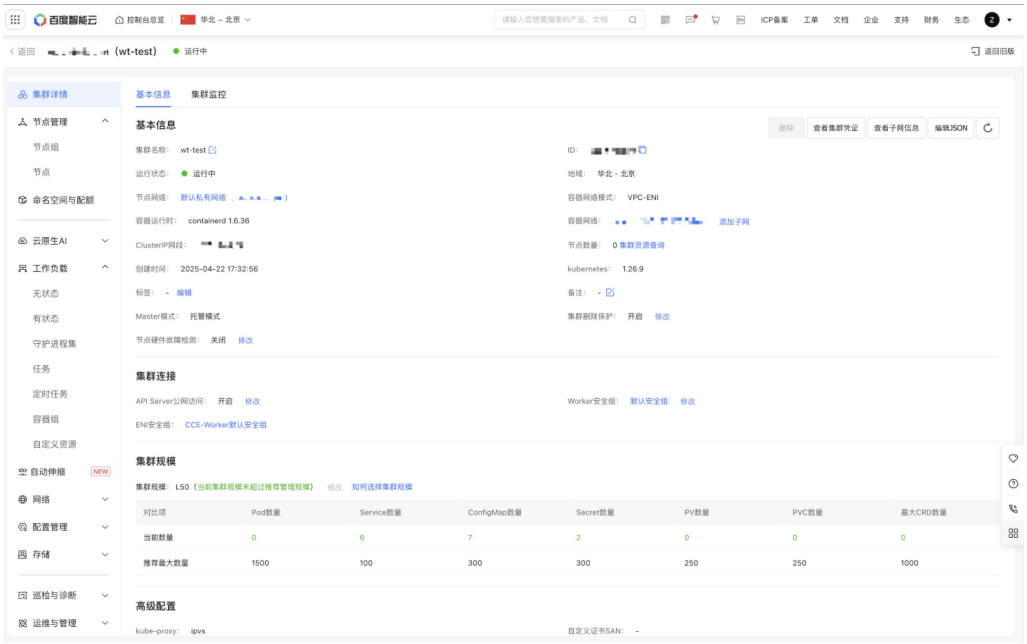
查看集群

创建集群后，支持通过集群的详情页面，查看和管理集群的集群信息，包括集群基本信息、集群连接、高级配置、集群监控。

查看集群基本信息

操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择集群列表。
3. 在“集群列表”页面单击目标集群，进入**集群详情**页面，选择**基本信息**页签，查看基本信息。



查看集群监控

前提条件

- 已成功创建CCE集群，操作方法参见 创建集群。
- 已开通**百度智能云 Prometheus 服务**，您可以登录Prometheus服务控制台确认是否开通。

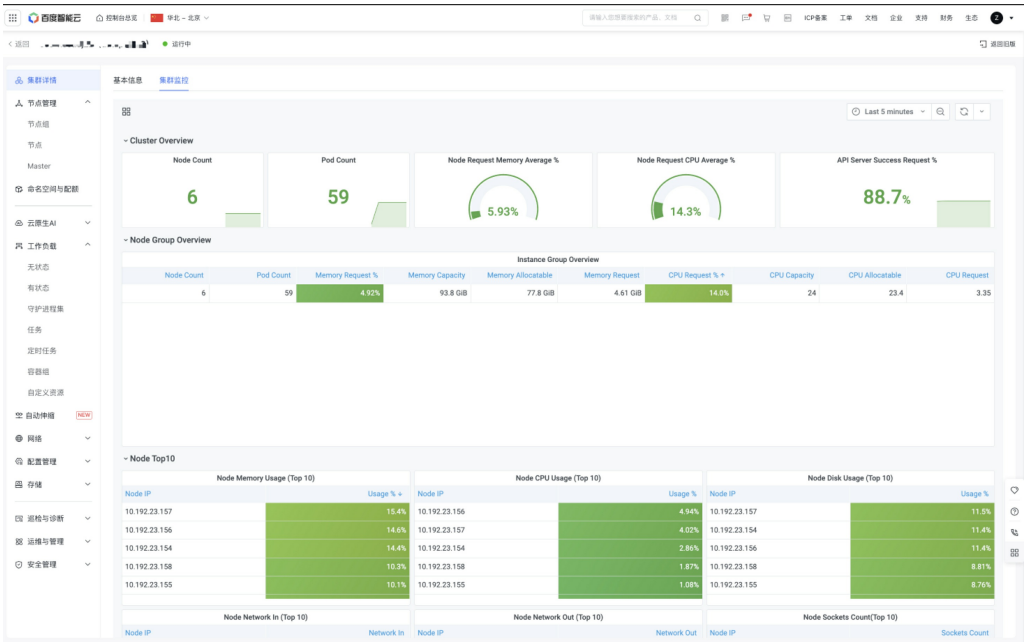
操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择集群列表。
3. 在“集群列表”页面单击目标集群，进入**集群详情**页面，选择**集群监控**页签。
4. 单击**立即开启**，弹出接入 CProm 监控弹窗，您可以选择接入与 CCE 集群相同地域中已有的监控实例，也可以选择新建监控实例，点击确认，完成配置。
5. 接入 CProm 监控实例后，控制台会自动安装组件、检查监控大盘。安装完成后，您可以单击各个页签查看相应监控数据。

说明

- 接入 CProm 监控实例，系统会默认在 CCE 集群安装监控组件。该组件用于采集 CCE 集群的各项监控指标，安装该组件会占用 CCE 集群一定资源，建议集群预留0.2核200Mi以上的资源，防止因资源不足导致实例接入失败。
- 您也可以选择登录Prometheus服务控制台选择 CProm 监控实例关联 CCE 集群，详情参考CProm监控实例关联集群。
- 接入 CProm 监控实例后会默认开启采集规则，如果您想要采集自定义业务指标，支持通过创建自定义采集任务，来满足您采集自定义指标需求，详情请参见采集配置。

查看监控 您可自定义时间段查看集群相关监控项的可视化监控图表，方便了解集群健康状态。



后续操作 若需要为集群配置告警策略，您可以在托管 Prometheus 的告警中心配置集群相关告警，详细信息请参考告警配置。

操作集群

扩容集群

在容器服务中，通过为集群新增 Worker 节点或将已有的云服务器（BCC）实例添加到集群的方式，达到为集群扩容的目的。

使用限制

- 通过新增节点或添加已有节点时，均支持添加 **按量计费** 和 **包年包月** 类型的节点。
- 请在集群的配额范围内进行扩容。若配额不足，可申请扩充配额。详细的操作和说明，请参见 使用限制。

操作步骤

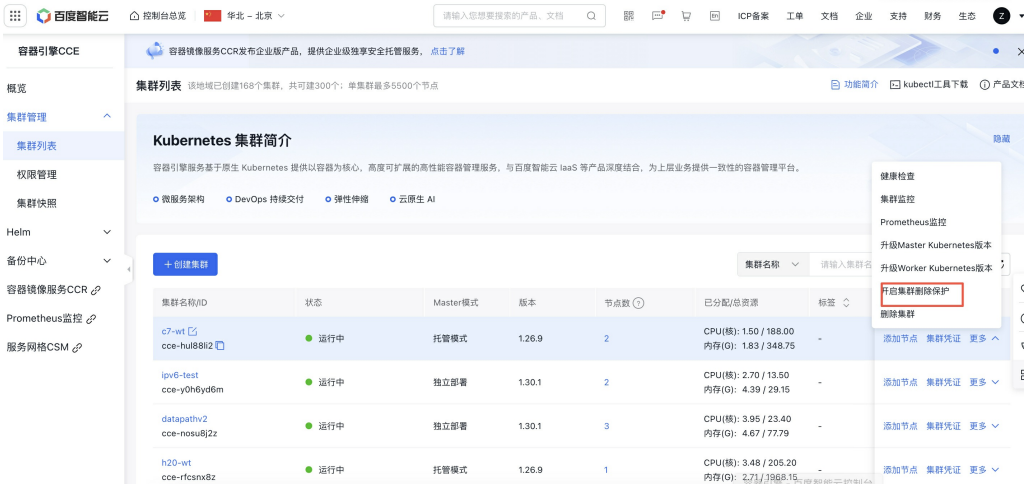
1. 登录百度智能云管理控制台，进入“产品服务>云原生>容器引擎 CCE”，点击“集群管理>集群列表”，进入集群列表页面。
2. 选择需要扩容的集群，在操作栏中单击添加节点，或点击集群名称，进入集群详情页，点击“节点列表>添加节点”：
3. 在“添加节点”页，设置新节点的规格，详细信息请参见添加节点。
4. 确认选择的配置信息和配置费用后，点击“完成”，付款后完成新节点的添加。

集群删除保护

在实际使用过程中，您可能会遇到许多误删除集群的场景。例如，如果您的账号为多人协作使用，其他用户可能会误删除不属于自己的集群。因此，您可以为重要的集群设置删除保护的措施，防止通过控制台或API误删除集群，避免集群中的重要数据丢失。

操作步骤

1. 登录百度智能云管理控制台，在左侧导航栏选择集群列表。
2. 在集群列表页面，选择所需的集群并单击右侧的**更多** > **开启删除保护**，或在集群详情页修改删除保护配置。



删除集群

删除集群将影响集群内创建的资源，高危操作，请确保当前集群不再使用时再执行此操作。

前提条件

1. 已经存在集群，且确保该集群不再使用。
2. 确保该集群未开启 **删除保护** 功能，如果集群开启了**删除保护**，在删除集群前，需要关闭集群的**删除保护**。

注意事项

删除集群将影响计费和集群内创建的资源，请谨慎操作。详细规则如下：

1. 集群删除时将根据您的以下选择，自动删除或保留节点、磁盘、公网IP等资源。
2. 若资源正在被其他集群或云产品使用，将自动为您保留。 保留资源将继续进行计费，请您自行到对应控制台进行查看和操作。
3. 集群删除后不可恢复，请谨慎操作！

操作步骤

1. 登录百度智能云管理控制台，在左侧导航栏选择集群列表。
2. 在集群列表页面，选择所需的集群并单击右侧的**更多** > **删除集群**。
3. 在删除集群面板，确认将自动删除的资源（例如节点、磁盘、公网IP等），按需选择待保留的资源，仔细阅读页面上关于资源删除的注意事项，并按照页面提示完成删除操作。

> **说明：**
>
> - 若集群删除失败，可根据控制台提示，处理故障后重新执行此操作。您也可以查看操作日志，处理故障；若多次尝试删除集群均失败，请提交工单获取技术支持。
> - 如果在删除集群提示页面，选中保留资源，则不会释放这些资源，您需要手动释放。
> - 删除集群需要花费5～10 分钟，请耐心等候。当集群列表中，对应集群消失，则说明删除成功。

通过kubectl连接集群

通过kubectl连接Kubernetes集群

如果用户需要从本地个人计算机连接到百度智能云的 Kubernetes 集群，需要使用 Kubernetes 命令行客户端 kubectl。kubectl 是 Kubernetes 提供的一种命令行工具，它用于与 Kubernetes 集群进行交互，帮助用户管理集群中的资源、查看集群状态、部署应用程序、进行调试等操作。通过 kubectl，您可以方便地在命令行界面上执行集群管理任务。可以选择两种访问方式：

- 内网访问：客户端通过内网IP地址与集群的API Server进行通信，数据流量不会经过互联网，安全性更强。
- 公网访问：集群的API Server会暴露一个公共接口，客户端可以通过互联网访问Kubernetes集群。注意：公网访问需要集群开启 API Server 公网访问。

前提条件

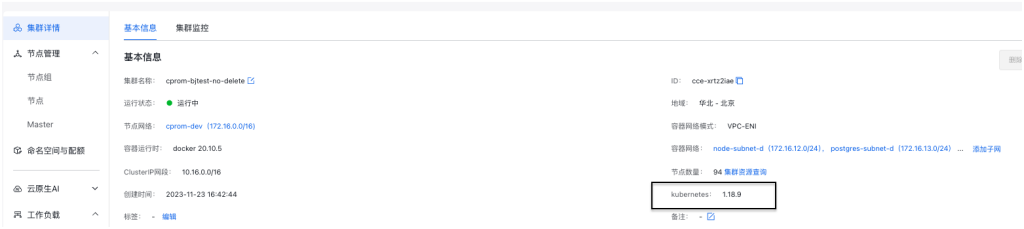
- 已创建集群。详细操作，请参见 创建集群。

操作步骤

使用 kubectl 连接集群时，您需要先在客户端中安装 kubectl。随后在集群中下载 kubectl 配置文件，并复制至客户端，在客户端中完成相关配置后即可访问对应的 CCE 集群。使用kubectl连接集群的具体步骤如下：

> **注意：**
>
> 本操作指南给出的操作步骤都是基于linux操作环境的。

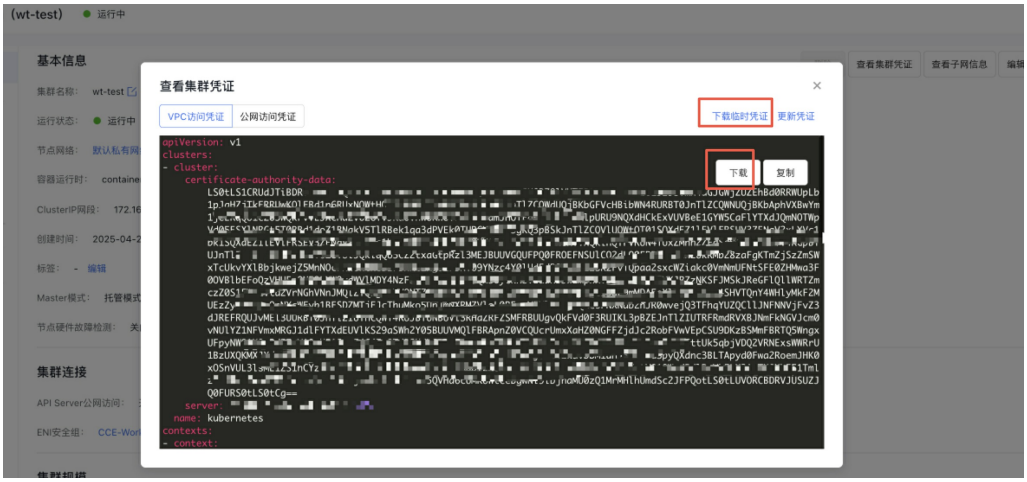
1. **下载kubectl**
   您需要准备一台可访问公网的客户端计算机。参考 安装工具 根据机器操作系统找到下载kubectl的对应步骤。如果已经安装kubectl，则跳过此步骤，您可执行kubectl version命令判断是否已安装kubectl。
   注意： 根据集群版本下载同版本的kubectl, 集群版本查看方式参考下图。



2. **下载集群凭证**：

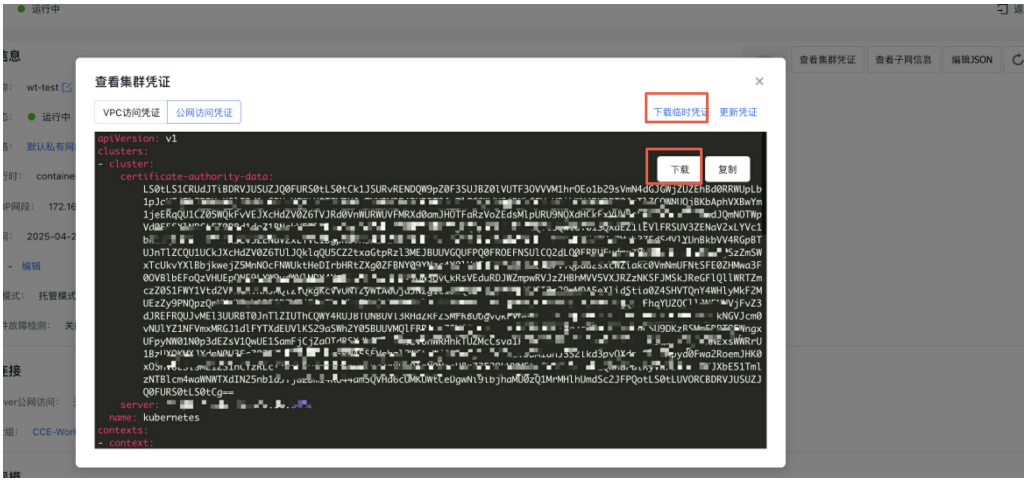
   可在集群列表查看集群凭证，也可在集群详情页查看集群凭证

a. VPC访问凭证：在查看集群凭证弹窗，选择VPC访问凭证，单击下载长期访问凭证，或下载临时访问凭证，获取集群私网访问凭证。



b.公网访问凭证：在集群基本信息页签下的集群连接模块，查看 API Server 公网访问 是否已开启。 若未打开 API Server 公网访问，请选择开启。

在查看集群凭证弹窗，选择公网访问凭证，单击下载长期访问凭证，或下载临时访问凭证下载集群公网访问凭证。



3. **配置kubectl**：将下载的集群凭证放在 kubectl 的默认配置路径。

   a.登录到您的客户端计算机，复制步骤3中下载的访问凭证文件（以kubeconfig.yaml为例）到您客户端机器的/home目录下。

   b.配置kubectl文件。

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

   c.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
```

4. 配置完成后，在客户端中执行以下命令，检查客户端能否通过kubectl连接集群。

```
kubectl cluster-info    # 查看集群信息
```

若回显结果如下，则说明客户端能够通过kubectl连接集群。

```
 Kubernetes control plane is running at https://xx.xx.xx.xx:5443
CoreDNS is running at https://xx.xx.xx.xx:5443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

通过kubectl创建简单的服务

创建一个nginx应用

这个命令假设kubectl命令行工具已经在PATH中，且有权限操作kubernetes集群（即已完成通过kubectl连接Kubernetes集群的操作）输入以下指令：

```
kubectl create deployment  my-nginx --image=registry.baidubce.com/cce/nginx-alpine-go:latest --replicas=3 --port=80
```

上述命令会创建3个nginx容器，每个容器都会暴露它的80端口。

查看创建nginx容器状态的指令如下：

```
 $ kubectl get pods
NAME                    READY    STATUS    RESTARTS   AGE
my-nginx-858393261-pfjdn  1/1     Running   0         14s
my-nginx-858393261-sn7g5  1/1     Running   0         14s
my-nginx-858393261-spv8w  1/1     Running   0         14s
```

> **注意：** 符号"$"代表输入的指令，其他行代表输入指令后显示的容器的信息。

暴露服务到internet

创建服务

在公有云环境下，可以通过输入命令，创建出一个类型为LoadBalancer的service，同时，还会联动公有云，创建一个负载均衡和一个公网IP，具体操作指令如下：

```
kubectl expose deployment my-nginx --port=80 --type=LoadBalancer
```

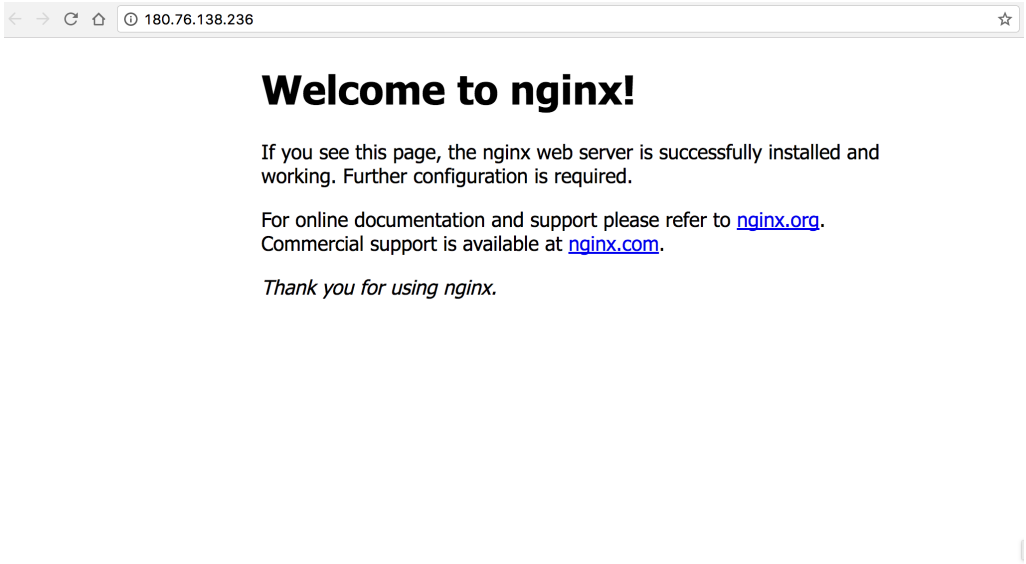
> **注意：**
>
> 负载均衡：通过将同一区域的多台百度智能云服务器虚拟成一个组，设置一个内网或外网的服务地址，将前端并发访问转发给后台多台云服务器，实现应用程序的流量均衡。您可以通过查询服务获取的公网IP,在百度智能云控制台中的负载均衡BLB页面找到您创建的BLB实例，其收费标准请参考负载均衡BLB定价。 公网IP：作为一个独立的商品为用户提供公网带宽服务。您可以通过查看服务来查询您创建的公网IP地址，其收费标准请参考弹性公网IP EIP的定价，且自动默认计费标准为后付费。

查看服务

1. 通过查看服务的命令，可以看到这个服务的情况和创建出的公网IP。

```
 $ kubectl get services
NAME         CLUSTER-IP    EXTERNAL-IP      PORT(S)       AGE
kubernetes   172.17.0.1    <none>           443/TCP       1h
my-nginx     172.17.44.5   180.76.139.247   80:30356/TCP   47s
```

2. 通过访问180.76.139.247这个链接可以看到创建的服务如图所示：



> **注意：** 符号"$"代表输入的指令，其他行代表输入指令后显示的服务信息，其中180.76.139.247 为发布到外网的服务网址。 服务使用负载均衡除了上述指令调用之外，还可以通过Dashboard调用具体操作参考 通过Kubernetes-Dashboard创建服务

删除服务

如果需要删除服务，不想将对公网暴露。可以直接删除服务

```
kubectl delete svc my-nginx
```

> **注意：**
>
> kubernetes会将刚才创建出的公网IP和负载均衡删除，然后也会删除创建的这个服务。

本地访问 Kubernetes Dashboard (Web UI)

前提条件

在本地使用 Kubernetes Dashboard 前，用户需要先配置好kubectl，能通过kubectl连接百度智能云的 Kubernetes 集群。

操作步骤

1. 创建Dashboard服务。在本地通过kubectl工具，可以在kubernetes集群里创建一个Dashboard应用。

```
 $ kubectl create -f dashboard.yaml
secret "kubernetes-dashboard-certs" created
serviceaccount "kubernetes-dashboard" created
clusterrolebinding "kubernetes-dashboard" created
deployment "kubernetes-dashboard" created
service "kubernetes-dashboard" created
```

对应的dashboard.yaml文件为：

```
 apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-certs
  namespace: kube-system
type: Opaque
```

```
---

apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system

---

apiVersion: v1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system

---

kind: Deployment
apiVersion: apps/v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: registry.baidubce.com/public/dashboard:v1.8.3-caas
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
        volumeMounts:
        - name: kubernetes-dashboard-certs
          mountPath: /certs
        - mountPath: /tmp
          name: tmp-volume
        livenessProbe:
          httpGet:
            scheme: HTTPS
            path: /
            port: 8443
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: kubernetes-dashboard-certs
        secret:
          secretName: kubernetes-dashboard-certs
```

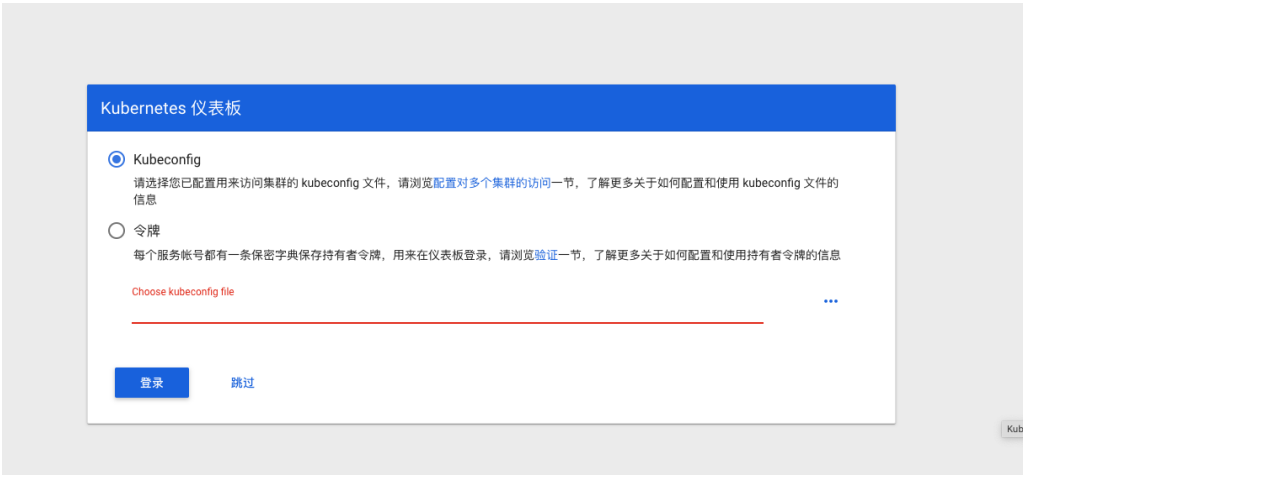
```
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule

---
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  ports:
    - port: 443
      targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
```

2. 在创建好dashboard之后，使用kubectl proxy命令开启一个代理，连接到Kubernetes API Server。

```
kubectl proxy
```

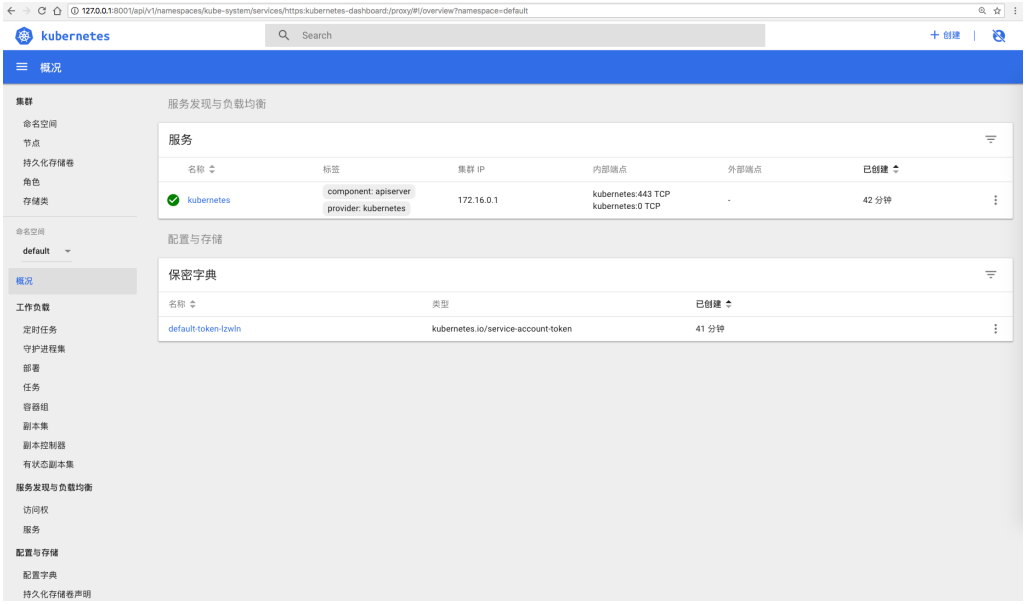
3. 成功开启代理后，可以在本地浏览器中访问http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login，登录dashboard 可以选择令牌登录，创建令牌的方式如下，使用返回结果作为令牌登录。

```
$ kubectl -n kube-system create token kubernetes-dashboard
eyJhbGciOiJSUzI1NiIsImtpZCI6ImREUzFpaHRJYnBLTV9jX1dmb05Bb2oxMXU0RE92QnhuejZ1WXJSYWo5MUEifQ.eyJhdWQiOlsia3ViZXJuZXRlcy5kZ
-ZkwSVT5A2gu0Qi2GjVKJPN00K4t19AXobzDmjIG-JEydVBN1n4SQ4GkW6_T_3IpwOV-P8JcAspTSi-0fHC5IR64TKVhS-
HXNRECD2Z11K2eQuoZR0hP887sCGmuX1Xz6kb12YkBxuZgDaGEHS7fiVi67xTey2B2kYAzibn4KZfMBqLOBzmZ_aM2O_iWjUJ8isgwvwce5DAf8KT8
-RWqmxUizXK3sFzvxoK4RLwGj1G1OSMpw6cFH0sLFuzR-gyIFLBOK9oyEsNnHhiqIpnBzVuamBpzwCLvvuwHVkKQ
```



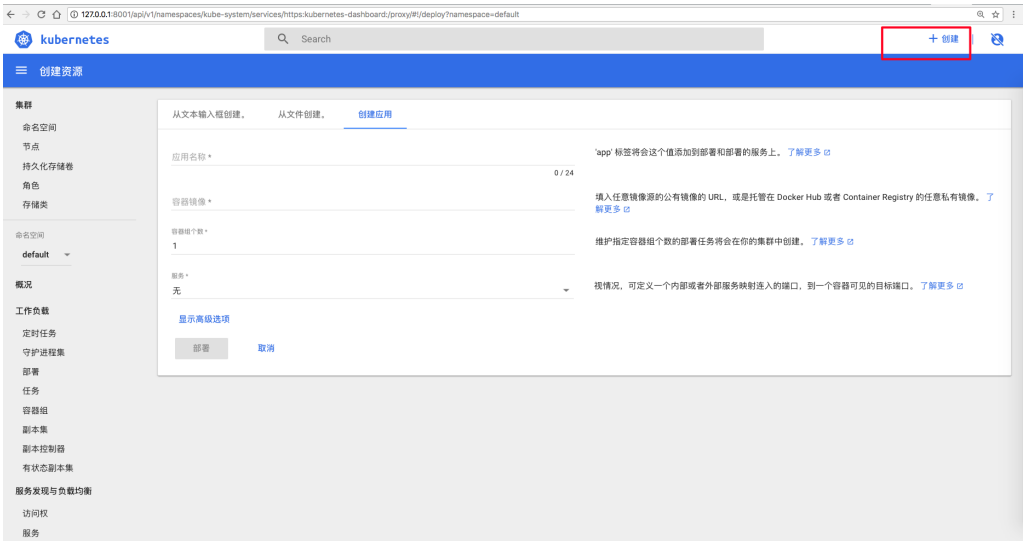
> **注意：**
>
> localhost 是指的用户使用的本地机器，而不是虚拟机，且对应的IP地址为：127.0.0.1

4. 登录成功后Dashboard 页面如下。



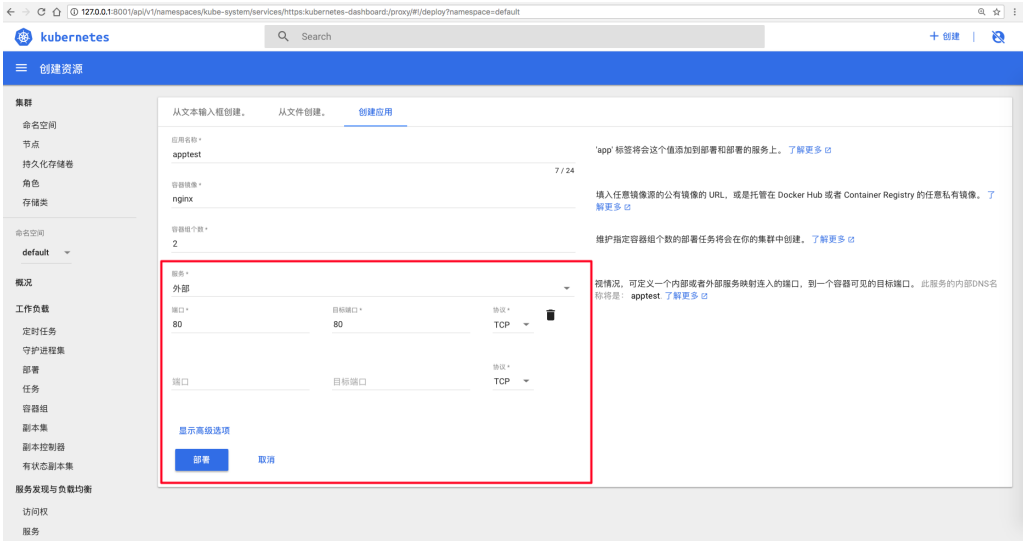
通过Kubernetes-Dashboard创建服务

1. 完成本地访问 Kubernetes Dashboard (Web UI)后，进入Dshboard页面，点击右上角创建，开始创建服务；

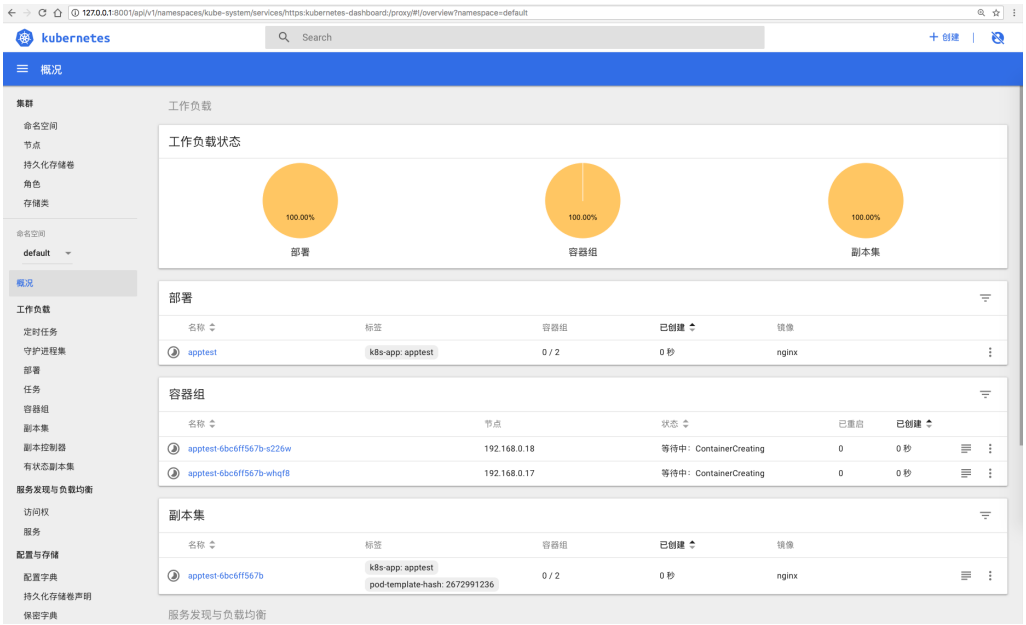


2. 进入创建应用页面，输入要创建的应用名称，容器镜像，容器组（Pod）数量；

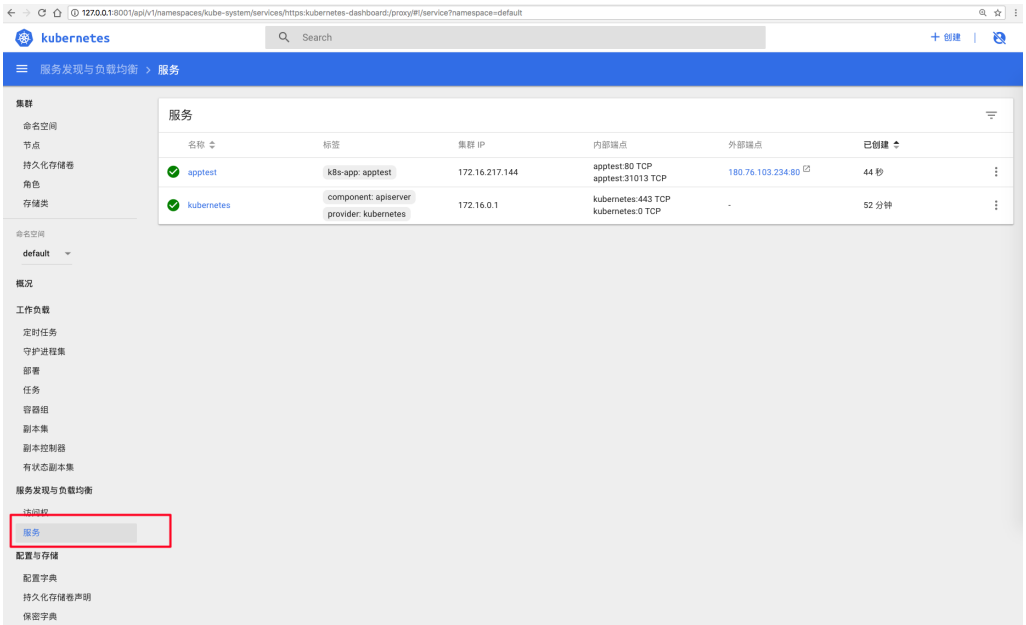
3. 服务中选择外部， 后台会完成EIP和BLB的创建，端口配置分别填写80，80，协议选择TCP，然后点击部署按钮开始服务创建；



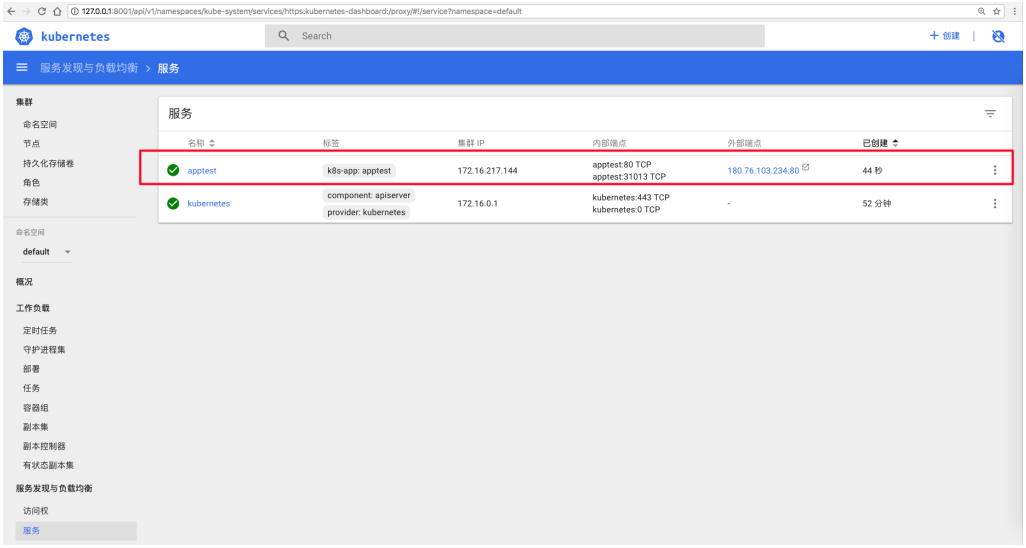
4. 等待服务创建成功，且页面需要手动刷新；



5. 在创建成功页面点击服务按钮，进入服务页面；



6. 在此页面中会显示服务的外网地址；



7. 在浏览器中输入外网地址，显示服务创建成功；

使用私有镜像创建服务

如果用户需要使用私有镜像，需要先配置ImagePullSecrets，同时在创建的资源中指明此ImagePullSecrets。下面简要介绍创建步骤，更加详细的关于ImagePullSecrets的信息，可以参考Kubernetes官方文档。

1. 通过kubectl创建ImagePullSecrets

   首先，用户需要有私有仓库空间的用户名密码，通过以下命令，可以创建出名为`myregistrykey`的ImagePullSecret

   ```
   $ kubectl create secret docker-registry myregistrykey --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL

   secret "myregistrykey" created.
   ```

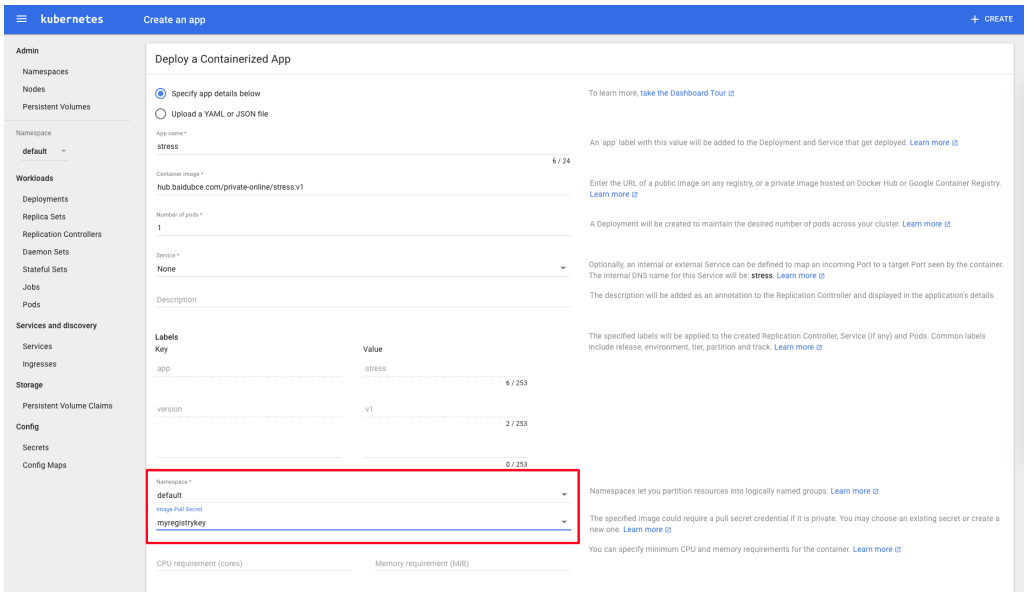
2. 在需要创建的资源中使用此ImagePullSecret

   例如，下面的pod yaml文件里指明此Pod使用私有镜像`registry.baidubce.com/private-online/awesomeapp:v1`，同时通过`myregistrykey`这个imagePullSecrets去镜像仓库拉取镜像。

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: foo
     namespace: awesomeapps
   spec:
     containers:
       - name: foo
         image: registry.baidubce.com/private-online/awesomeapp:v1
     imagePullSecrets:
       - name: myregistrykey
   ```

3. 在dashboard中指定创建服务的ImagePullSecrets

   在通过kubectl创建成功ImagePullSecrets后，在dashboard中创建应用时，也能选择使用ImagePullSecret。 例如下图中：



我们为即将创建的应用选择了前文创建的`myregistrykey`，这样kubernetes就有权限区拉取私有仓库的镜像了。

windows 下使用kubectl

1. 创建集群之后，下载windows版本的kubectl工具，例如64位的工具如下：

v1.8.6

Documentation & Examples

Downloads for v1.8.6

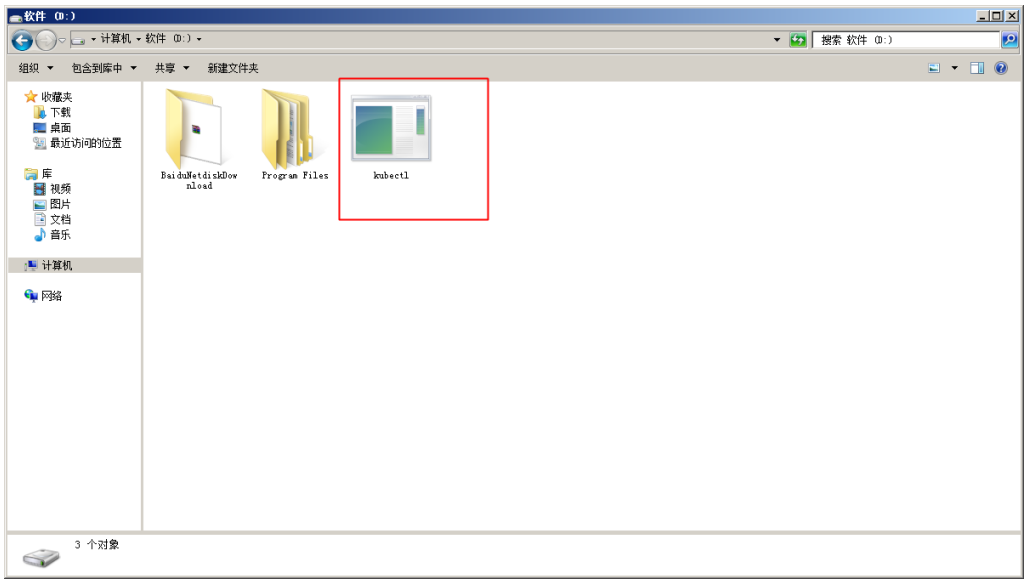
filename	sha256 hash
kubernetes.tar.gz	8289c42b5d6da1dbf910585fca3a9d909195e540cc81bace61ec1d06b2366c1b
kubernetes-src.tar.gz	8a9d5d890c44137527fe3976d71d4f7cb18db21ba34262ce587cd979a88bb2fe

Client Binaries

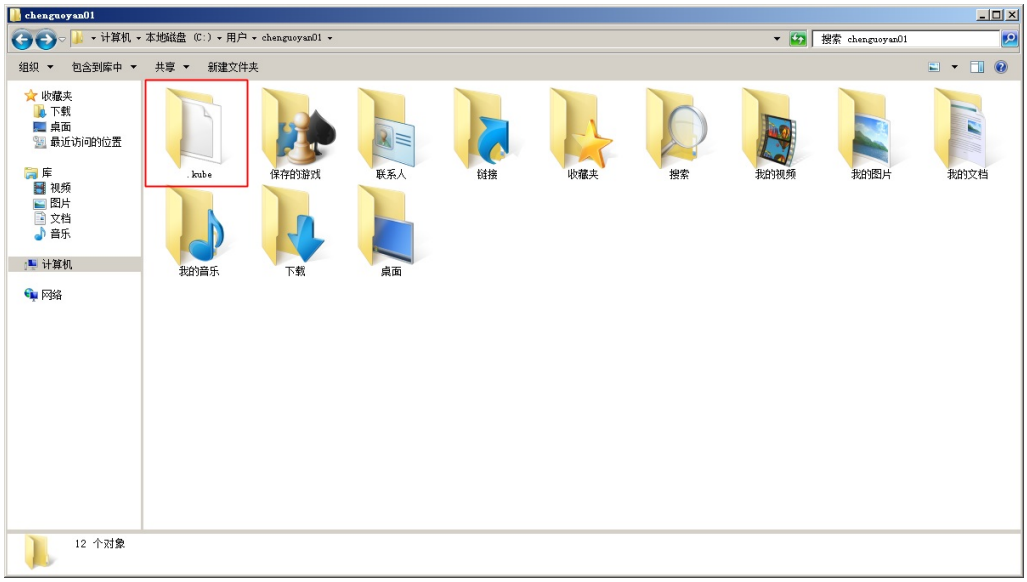
filename	sha256 hash
kubernetes-client-darwin-386.tar.gz	0e282477bfed6b534f2fbbd125e6e3e065bf72d15ac3532acef405e6717d8fb7
kubernetes-client-darwin-amd64.tar.gz	767c7bfbc6c1d01120e11726b9e33e184d32294e07c69a299b229329c5b98eba
kubernetes-client-linux-386.tar.gz	088b40c343fecb83b514bf9af0ad1c359c98ae7aa3b62d2a078c1363f50901c9
kubernetes-client-linux-amd64.tar.gz	47541706e4d27da55d32372344d7a4038ed389ba0be1e6fe15c651c574aac97a
kubernetes-client-linux-arm64.tar.gz	4be0b7a01c28c1f85d4f01f86def03dd3d49ef88cb43bf7be641d9d16b6aabc2
kubernetes-client-linux-arm.tar.gz	2d70384262cbdfb0958542bc5a71d926c49557fc8cc3000a2592571a945ad119
kubernetes-client-linux-ppc64le.tar.gz	c3be3a125ac77aa809da3495ad38456059a89cccfdfad0babaf95896fb958adc
kubernetes-client-linux-s390x.tar.gz	2b9831c2dd65c9669b335e3623e6a7001173b9ddf203f52f37b350659d9f1102
kubernetes-client-windows-386.tar.gz	9d14a96372cdcecbbb28717aff305fcd68beb540066a27f1b5e84e208a25405f
kubernetes-client-windows-amd64.tar.gz	0fbe358ff305188fe00793284e22c9c5b2ec0e0213882f0bfe0e4bf9685075f0

> **注意：** 下载kubectl之前，用户需要先登录自己的服务器使用指令`kubectl version`查看自己创建的集群的版本号，然后选择对应的kubectl版本进行下载。例如：用户的集群版本号为1.8，则需要下载的kubectl为1.8版。下载链接为https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/README.md,用户根据自己集群版本号，下载相应版本的工具。

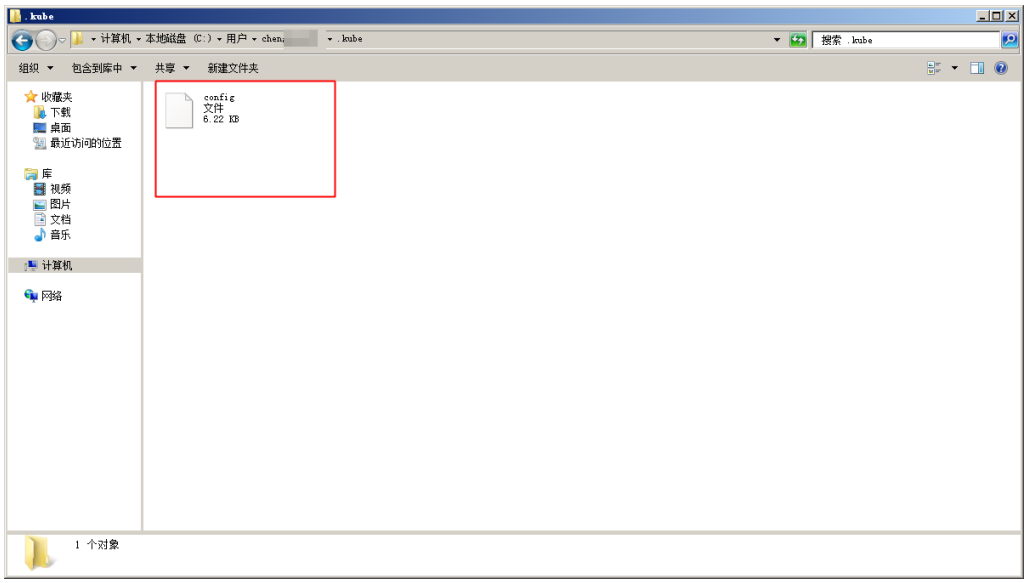
2. 下载之后，将其解压到任意目录，例如放在D盘下



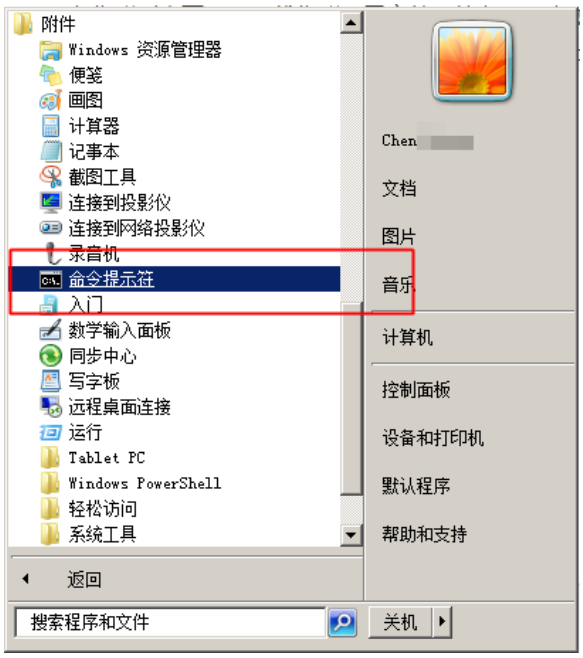
3. 在 目录C:\users\下进入用户的文件夹，例如用户名为foo,则文件夹应该为C:\users\foo，同时进入此目录，创建文件夹.kube



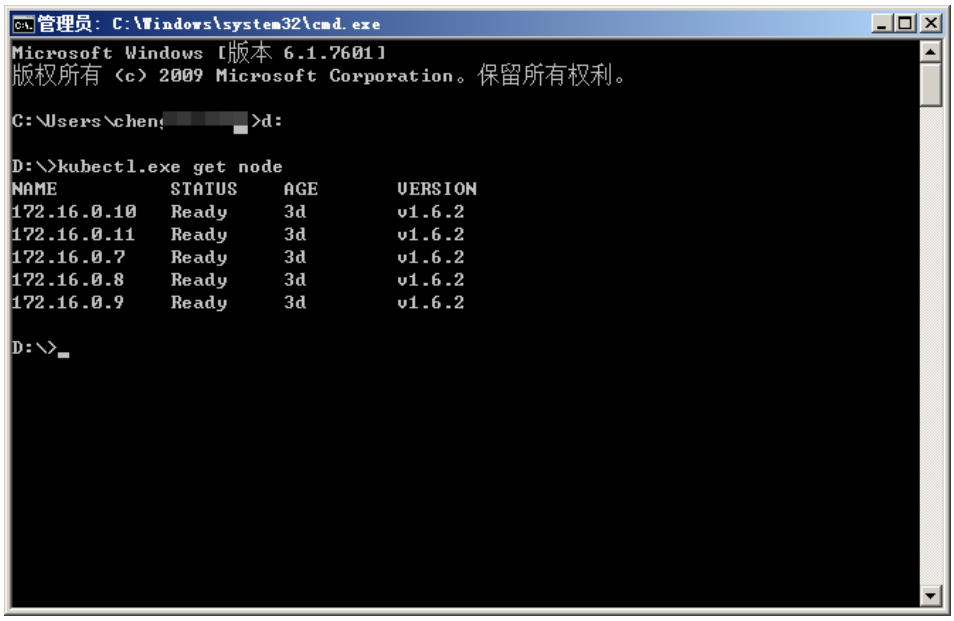
4. 在集群列表页面，下载集群配置文件，放在上一步创建的.kube文件夹中，并且重名名为config



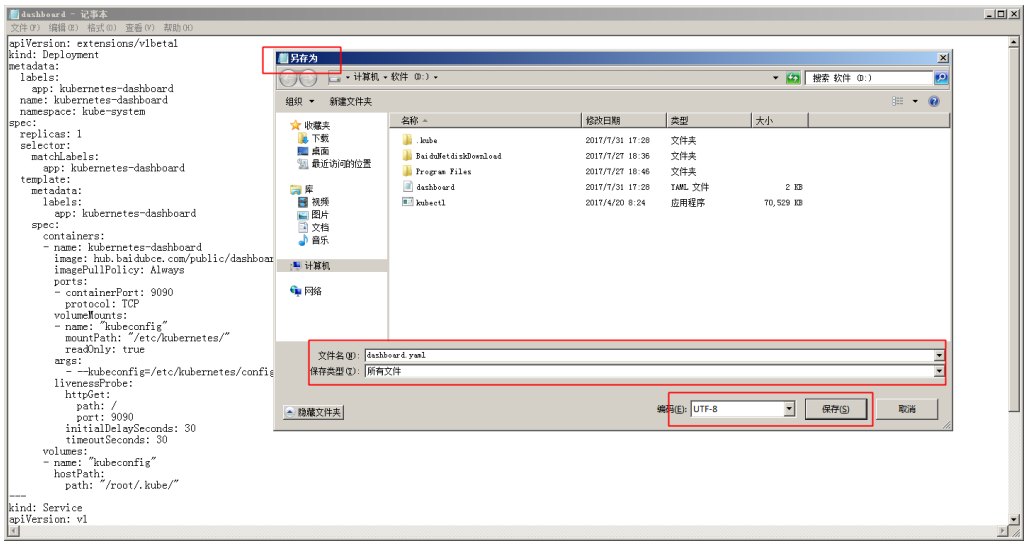
5. 打开windows命令行提示符



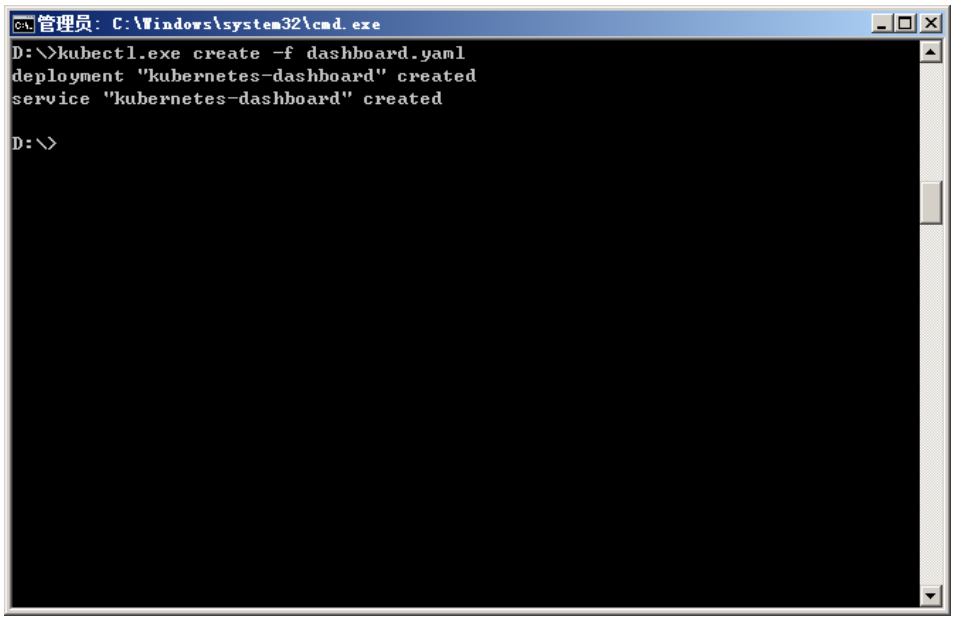
6. 进入kubectl文件存放的目录，运行`kubectl.exe get node`可以查看集群的节点。



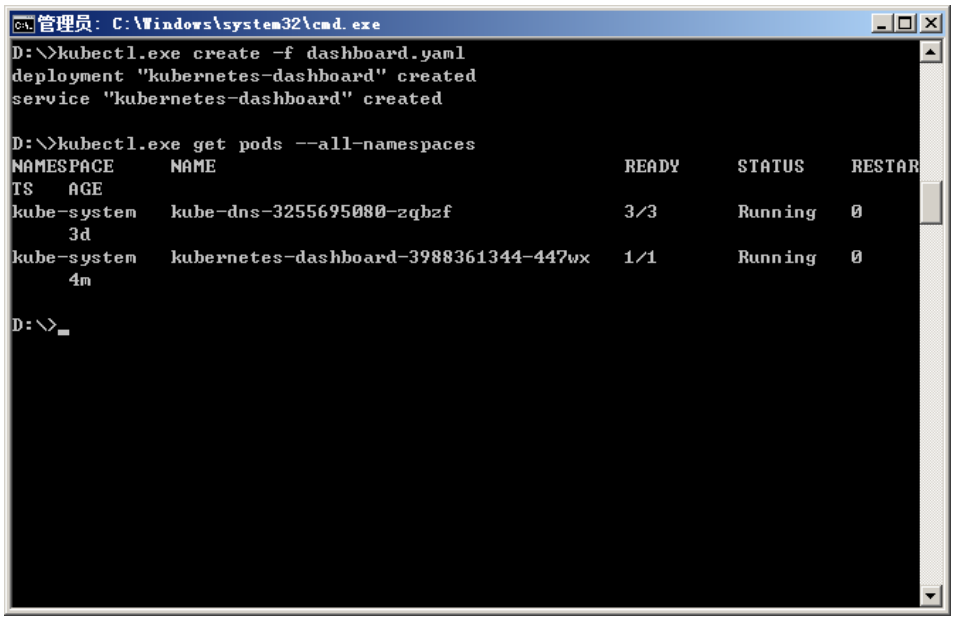
7. 在同一个目录下，创建dashboard.yaml文件，其内容是前文中的dashboard.yaml，例如可以用记事本创建：



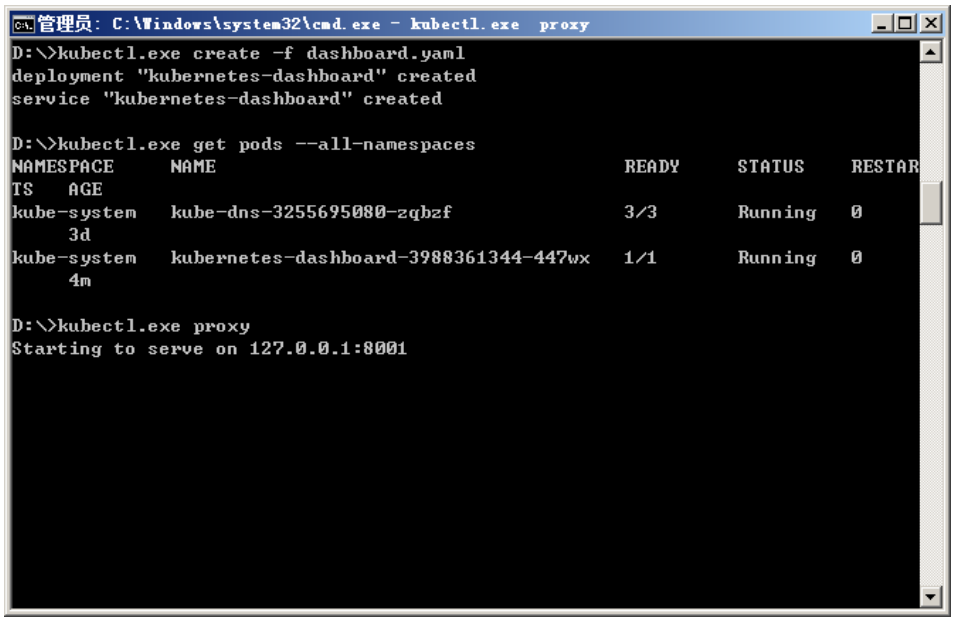
8. 然后输入`kubectl.exe create -f dashboard.yaml`创建dashboard

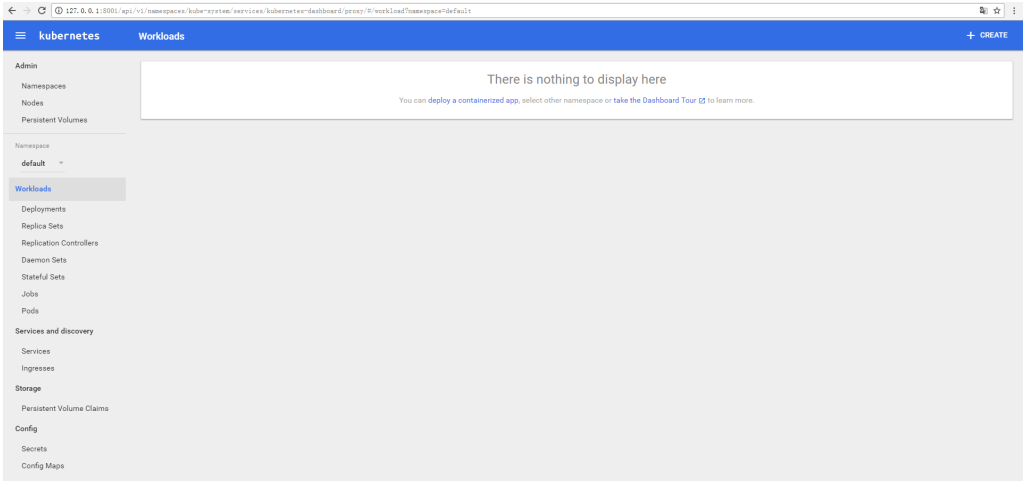


9. 通过`kubectl.exe get pod --all-namespaces`命令可以查看是否创建成功



10. 输入`kubectl.exe proxy`可以打开proxy，然后在浏览器中访问`127.0.0.1:8001/ui` 即可以访问dashboard





CCE节点资源预留说明

在 CCE 集群中，节点初始化时需要运行一些必要的系统组件和 Kubernetes 组件，使该节点能够成功纳入该集群并正常提供服务，因此需要为这些组件预留一部分系统资源，您会看到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，需要需预留的资源可能越多。

为了保证节点的稳定性，CCE 集群节点上会根据机器规格预留一部分资源给 Kubernetes 的相关组件（kubelet、kube-proxy、docker等）。CCE 存在默认的资源预留策略，也支持通过 kubelet 配置自定义资源预留。

节点最终可分配的资源计算公式如下：

可分配资源(Allocatable) = 总资源(Capacity)-预留资源(Reserved)-驱逐阈值(Eviction-Threshold)。其中内存资源的驱逐阈值，固定为100MiB。

> 说明：
> 此处总量 Capacity 为弹性云服务器除系统组件消耗外的可用内存，因此总量会略小于节点规格的内存值。详细信息请参考系统内看到的系统可用内存比标称值少。

当节点上所有 Pod 消耗的内存上涨时，可能存在下列两种行为：

- 当节点可用内存低于驱逐阈值时，将会触发 kubelet 驱逐 Pod。关于 Kubernetes 中驱逐阈值的相关信息，请参见节点压力驱逐。
- 如果节点在 kubelet 回收内存之前触发操作系统内存不足事件（OOM），系统会终止容器，但是与 Pod驱逐不同，kubelet 会根据 Pod 的 RestartPolicy 重新启动它。

影响范围

自定义资源预留及其影响范围 您可以参见自定义节点组 Kubelet 配置修改资源预留的取值。修改后，节点组中的已有节点将即时生效，新增节点（例如扩缩容的新节点、通过添加已有节点新增的 BCC 实例）也会使用该配置。

> **注意：**
>
> - 不支持通过黑屏手动修改 kubelet 配置文件，以避免配置冲突，导致后续节点组运维过程中的非预期结果。
> - 改变资源预留的值可能会造成节点的可分配资源变少。对于资源水位较高的节点，可能会触发节点驱逐。请合理配置取值。

默认资源预留及其影响范围 CCE 可能会迭代节点资源预留的默认取值。迭代后，如果您在节点组维度更新了节点配置，例如节点组升级、修改节点组自定义 kubelet 参数等，节点将自动使用新的资源预留策略。如果您没有相关运维操作，出于稳定性考量，节点组中的已有节点不会自动生效新的资源预留策略。

查询节点可分配资源

执行以下命令，查看节点的资源总量和可分配资源。

```
kubectl describe node 192.168.102.106 | grep Allocatable -B 9 -A 8
```

预期输出：

```
 Capacity:
  cce.baidubce.com/eni:  0
  cce.baidubce.com/ip:   7
  cpu:                   2
  ephemeral-storage:     103145380Ki
  hugepages-1Gi:         0
  hugepages-2Mi:         0
  memory:                8139584Ki
  pods:                  128
Allocatable:
  cce.baidubce.com/eni:  0
  cce.baidubce.com/ip:   7
  cpu:                   1900m
  ephemeral-storage:     98227408251
  hugepages-1Gi:         0
  hugepages-2Mi:         0
  memory:                7832384Ki
  pods:                  128
```

自定义资源预留

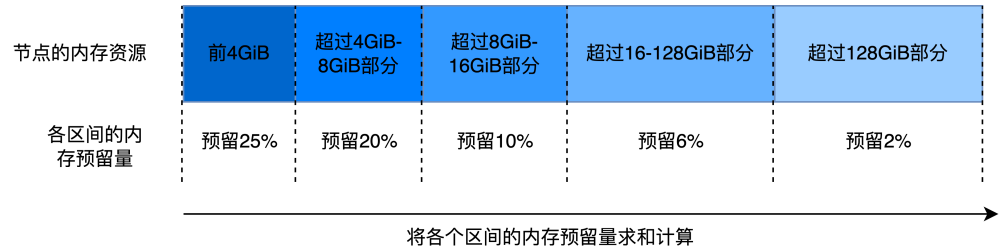
您可以参见自定义 kubelet 参数配置新增节点的资源预留；或修改节点组配置中的资源预留，新扩容节点将使用该配置。

> **重要**：如需修改自定义资源预留，请合理配置取值，不合理的设置会造成节点的可分配资源变少，对于资源水位较高的节点，可能会触发节点驱逐。

默认资源预留策略

如果没有指定 kubelet 资源预留参数，CCE 会按如下策略自动设置资源预留。

内存预留值：



举例说明：

- 256 GiB内存的节点为例，总的内存预留量计算如下：

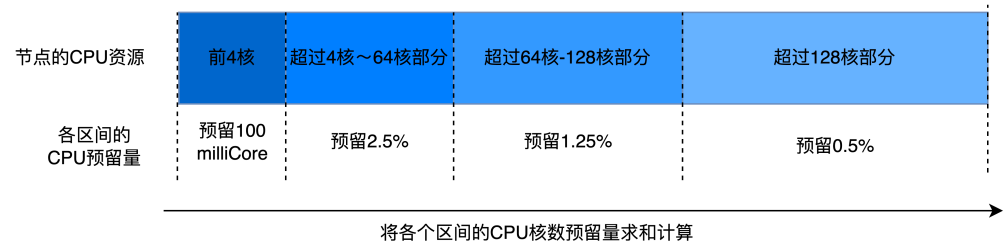
4×25%+(8-4)×20%+(16-8)×10%+(128-16)×6%+(256-128)×2% = 11.88 GiB

- 198 GiB内存的节点为例，总的内存预留量计算如下：

4×25%+(8-4)×20%+(16-8)×10%+(128-16)×6%+(198-128)×2% = 10.72 GiB

> 注意：
>
> - 节点内存资源：以系统实际可用的内存资源为准。单位为MiB。
> - 实例规格定义中的内存大小指所有可用内存，其中也包含了系统占用的部分。因此系统实时可用内存会小于实例规格定义，请参见系统内看到的系统可用内存比标称值少。

CPU 预留值：



举例说明：

- 32核的节点预留资源为：100+(32000-4000)×2.5% = 800 milliCore

CCE 节点默认资源预留示例

> 关于 BCC 实例规格的详细信息，请参见实例规格。
> 下面示例中 1GiB = 1024 MiB

节点总资源			预留资源	
示例实例规格	CPU（Core）	Memory（GiB）	CPU（millicore）	Memory（MiB）
BCC 通用型 g6	2	8	100	1843
	4	16	100	2662
	8	32	200	3645
	12	48	300	4628
	16	64	400	5611
	24	96	600	7577
	32	128	800	9543
	48	192	1200	10854
	64	256	1600	12165
	96	384	2000	14786
	128	512	2400	17408
	172	966	2620	26705

如何查看节点总 CPU 和内存？

CPU

执行如下命令，查询节点总 CPU。

```
cat /proc/cpuinfo | grep processor
```

预期输出：

```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
```

内存

执行如下命令，查询节点总内存。

```
cat /proc/meminfo | grep MemTotal
```

预期输出：

```
MemTotal:       7660952 kB
```

CCE 节点 CDS 扩容

CCE 节点容量不足时扩容节点 CDS 盘

当集群中的节点数据盘（CDS）空间不足，或因节点规格偏小等原因影响节点稳定运行时，您可以参见本文扩容节点数据盘(CDS)，以更好地满足

数据存储需求。

节点服务迁移

以节点 192.168.16.27 为例说明，执行 drain 驱逐节点服务：

```
##### kubectl drain 192.168.16.27  --ignore-daemonsets
```

检查节点状态为 SchedulingDisabled ：

```
##### kubectl get nodes
NAME           STATUS                     ROLES    AGE   VERSION
192.168.16.27   Ready,SchedulingDisabled   <none>   47m   v1.13.10
```

检查节点上业务 Pod 驱逐完成：

```
##### kubectl get pods --all-namespaces=true -o wide  |grep 192.168.16.27
```

停止节点服务

停止 kube-proxy

```
##### systemctl stop kube-proxy

##### 检查组件停止成功
##### systemctl status kube-proxy
```

停止 kubelet

```
##### systemctl stop kubelet

##### 检查组件停止成功
##### systemctl status kubelet
```

停止 docker

```
##### systemctl stop docker

##### 检查组件停止成功
##### systemctl status docker
```

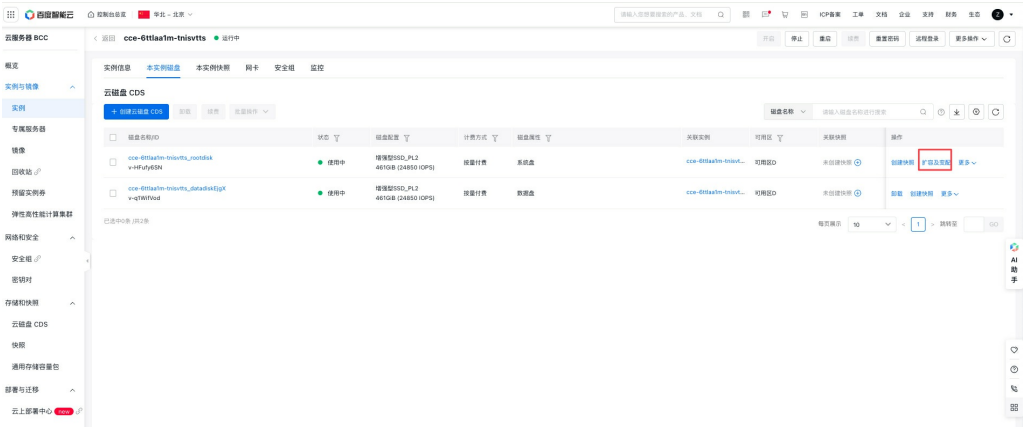
停止 containerd

```
##### systemctl stop conatinerd

##### 检查组件停止成功
##### systemctl status conatinerd
```

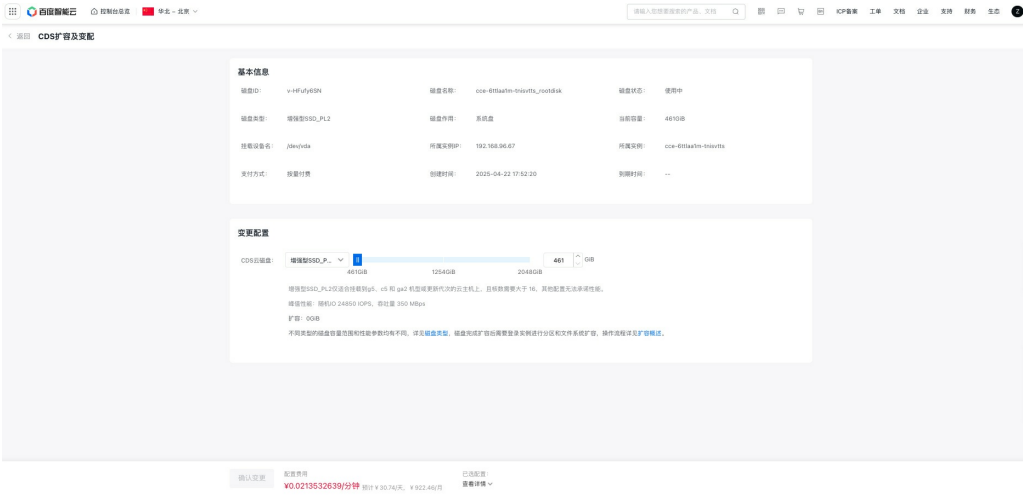
CDS 扩容

步骤一：确定当前磁盘大小和分区 执行数据盘扩容前，请登录节点，查看节点数据盘磁盘大小和所在分区。

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 集群管理 > 集群列表 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 左侧导航栏选择 节点管理 > 节点 ，进入节点列表页面，单击目标节点列表的实例名称。
5. 进入实例详情页面，选择本实例磁盘列表页签，选择需要扩容的CDS磁盘，单击扩容及变配。



6.在CDS扩容及变配页面，选择需要扩容的磁盘大小，点击**确认变更**，完成CDS的扩容，具体信息请参考CDS 扩容文档。



卸载 /data 目录

CCE 使用 /data 目录作为 docker 和 kubelet 数据盘，扩容分区前，先卸载磁盘挂载：

```
##### 卸载 docker 和 kubelet 挂载目录
##### for dir in $(df -h |grep -e docker -e kubelet); do umount $dir; done

##### 卸载 /data 挂载目录
##### umount /data
```

卸载完成后 df -h 看不到 /data 目录，表示成功。

扩容已有磁盘分区

参考 CDS 扩容步骤：

- CDS 扩容已有 MGR 磁盘分区
- CDS 扩容已有 GPT 磁盘分区

待扩容及文件系统挂载完成后，将 /dev/vdb1 挂载到 /data：

```
##### mount /dev/vdb1 /data
```

检查分区大小，大小一致表示符合预期：

```
##### df -h | grep data
/dev/vdb1       99G  1.3G   93G   2% /data
```

重启节点服务

启动 kube-proxy

```
 ##### systemctl start kube-proxy

##### 检查组件启动成功
##### systemctl status kube-proxy
```

启动 kubelet

```
 ##### systemctl start kubelet

##### 检查组件启动成功
##### systemctl status kubelet
```

启动 docker

```
 ##### systemctl start docker

##### 检查组件启动成功
##### systemctl status docker
```

启动 containerd

```
 ##### systemctl start containerd

##### 检查组件启动成功
##### systemctl status containerd
```

恢复节点调度

以节点 192.168.16.27 为例说明，执行 kubectl uncordon 恢复节点调度：

```
 ##### kubectl uncordon 192.168.16.27
```

至此节点 CDS 扩容完成。

CCE安全组说明

安全组是一种虚拟防火墙，为同一个私有网络内具有相同安全保护需求并相互信任的云服务器提供访问策略，具备状态检测和数据包过滤能力，用于在云端划分安全域，是重要的网络安全隔离手段。

您可以通过配置安全组规则控制安全组内云服务器的入流量和出流量。更多安全组介绍，请参见安全组。

使用须知

容器引擎CCE对集群使用安全组进行了优化，使用集群时支持用户配置“自动创建默认安全组”和“使用已有安全组”，若无额外诉求，推荐使用默认安全组即可，无需关注规则细节，使用成本低；若有额外诉求，可参考默认安全组规则保障集群内互通，降低因安全组规则使用问题导致集群访问不同，影响集群正常运行。

CCE默认安全组说明

CCE作为通用的容器平台，安全组规则的设置适用于通用场景。集群在创建时将会自动为Master节点和Worker节点分别创建一个安全组，其中Master节点的安全组名称是：**{集群ID}-master-{随机ID}**；Worker节点的安全组名称是：**{集群ID}-worker-{随机ID}**。使用VPC-ENI容器网络模式的集群时会额外创建一个ENI的安全组，名为**{集群ID}-eni-{随机ID}**。

Worker默认安全组规则

为保证集群和节点本身功能的正常运作，确保集群和节点之间的网络正常连通，CCE自动创建的Worker默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	集群网络 CIDR （VPC网段）	1-65535	IPv4	允许	开通集群网络内全部节点之间通信
入站	ALL	容器网络 CIDR	1-65535	IPv4	允许	开通集群容器网络内全部 Pod 之间通信（容器网络模式为“VPC路由”）
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	开通集群与CCE控制面子网之间通信
入站	TCP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通集群对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作

ENI默认安全组规则

“VPC-ENI”容器网络模式下，为保证集群节点上的Pod之间的网络正常连通，CCE自动创建的ENI默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	集群网络 CIDR （VPC网段）	1-65535	IPv4	允许	开通集群网络内全部节点之间通信
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	开通集群与CCE控制面子网之间通信
入站	TCP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通集群对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作

Master默认安全组

为保证集群和节点本身功能的正常运作，确保集群和节点之间的网络正常连通，CCE自动创建的Master默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	集群网络 CIDR （VPC网段）	1-65535	IPv4	允许	开通集群网络内全部节点之间通信
入站	ALL	容器网络 CIDR	1-65535	IPv4	允许	开通集群容器网络内全部 Pod 之间通信（容器网络模式为“VPC路由”）
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	开通集群与CCE控制面子网之间通信
入站	TCP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	NodePort 范围	IPv4	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通集群对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作
入站	TCP	ALL	6443	IPv4	允许	开放 apiserver 公网访问

IPv6集群默认安全组规则

若集群开启IPv6双栈网络，则集群会额外创建以下安全组规则：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	集群网络 IPv6 CIDR （VPC网段）	1-65535	IPv6	允许	开通集群网络内全部节点之间基于 IPv6 地址通信
入站	TCP	ALL	NodePort 范围	IPv6	允许	开通集群内全部 NodePort 类型 Service 之间基于 TCP 协议及 IPv6 地址通信
入站	UDP	ALL	NodePort 范围	IPv6	允许	开通集群内全部 NodePort 类型 Service 之间基于 UDP 协议及 IPv6 地址通信
出站	ALL	ALL	1-65535	IPv6	允许	开通集群对外全部通信

集群快照

您可以为集群内的K8S资源创建一个集群快照。集群快照会将集群内所有K8S资源（不包括secret资源）保存为yaml、json两种格式。

前提条件

- 您已成功创建一个 Kubernetes 集群,详情请见创建集群。

限制说明

- 可使用集群快照对集群内 K8S 资源进行备份、下载，暂不支持使用快照恢复。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群快照** 。
4. 在集群快照页面单击 **创建快照**。

5. 创建完成后，可在快照列表页查看。

6. 可通过下载集群快照，查看集群内K8S资源信息。

升级集群Kubernetes版本

容器引擎CCE提供升级Kubernetes功能，您可以通过控制台可视化升级集群节点Kubernetes版本。升级过程包含升级前置检查、升级Master、升级Worker（集群会展示待升级节点数）。

升级须知

- 升级属于不可逆操作，请谨慎操作。
- 仅支持向上升级Kubernetes版本，不支持连续跨大版本升级（如1.16跳过1.18直接升级到1.20），且当集群内Worker版本和Master大版本一致时才可以升级到下一版本。
- 升级前强烈建议您先仔细阅读 CCE Kubernetes版本更新说明 了解待升级版本的版本解读、废弃API等，避免因高版本的功能变更导致业务功能

不兼容。

- 升级完成Master，请尽快完成全部Worker版本升级，且升级过程中不建议对集群进行任何操作。
- 升级过程中，可能会有升级失败的情况，为了您的数据安全，强烈建议您先打快照再升级。
- 升级过程中，集群上的应用不会中断。如果应用强依赖API Server可能会有短暂影响。

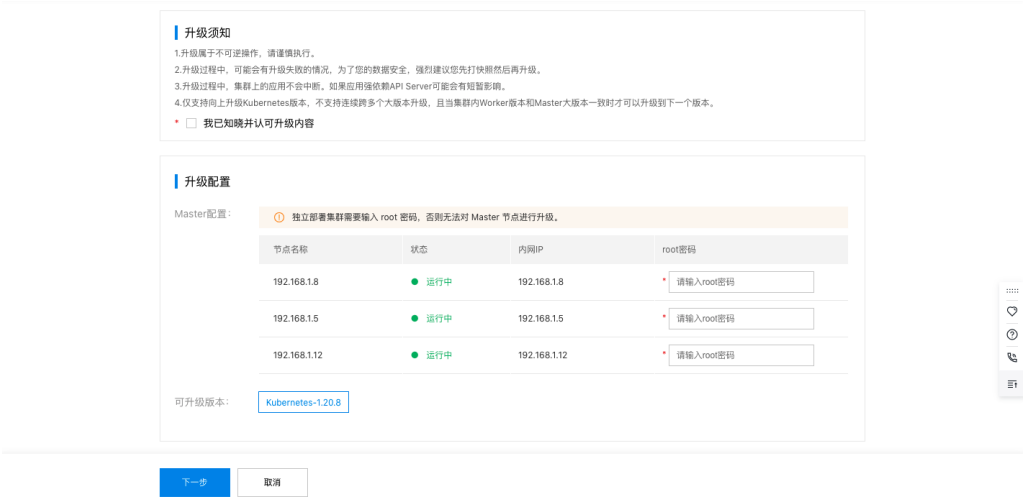
前提条件

- 集群类型为标准独立集群或托管集群。

操作步骤

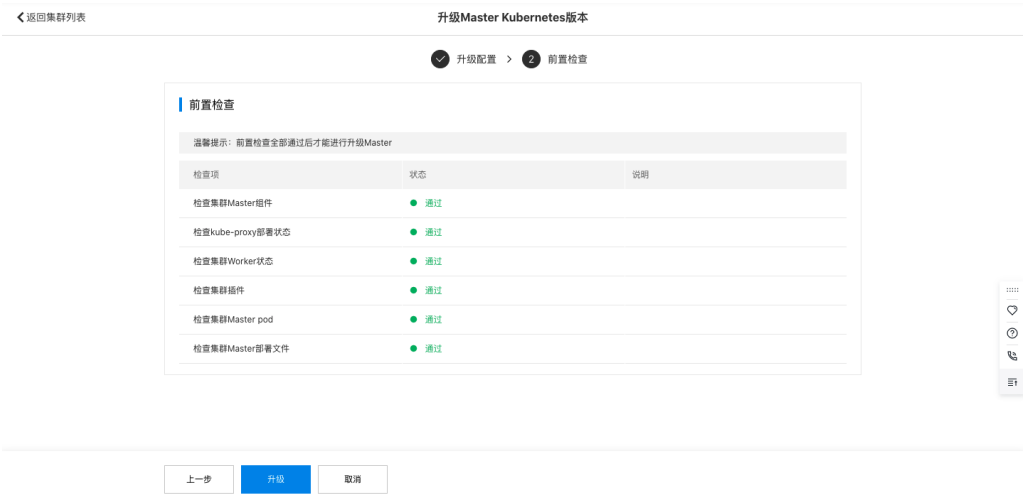
升级Master Kubernetes版本

1. 登录容器引擎CCE控制台，选择左侧导航栏中的**集群列表**。
2. 在“集群列表”页面单击目标集群左侧操作中的**更多>升级Master Kubernetes版本**进入升级页面并完成配置：

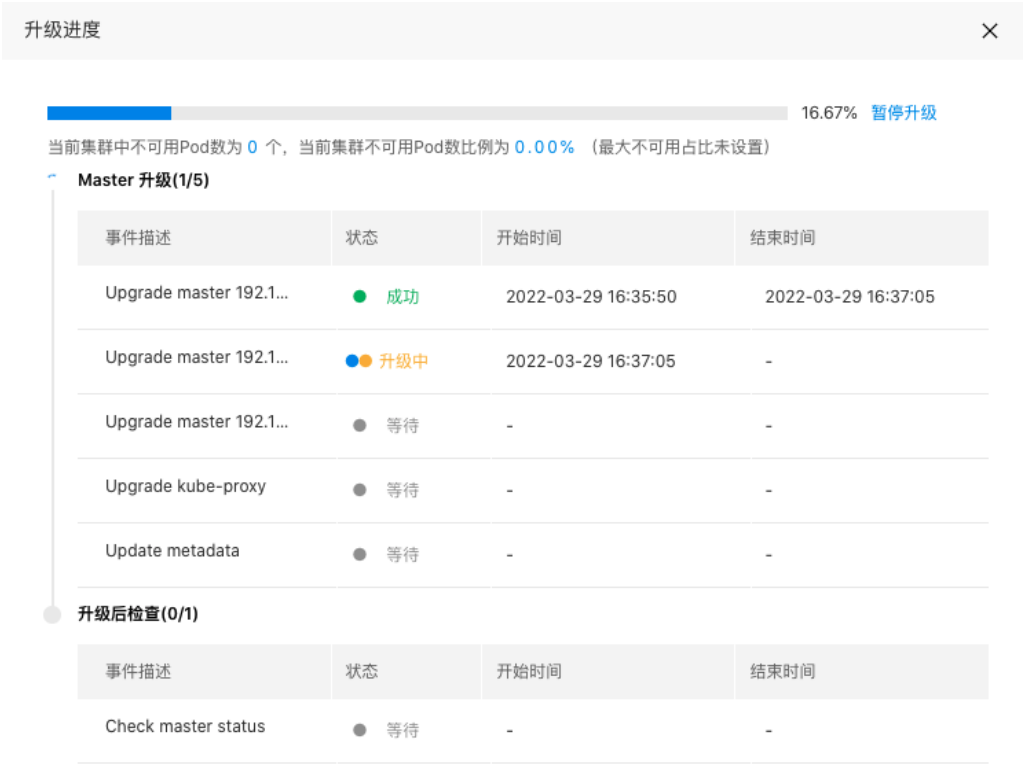


配置项	必选/可选	配置说明
升级须知	必选	升级前强烈建议您仔细阅读升级须知并且勾选后才能进行升级
Master配置	必选	请输入Master节点对应的root密码，托管集群不需要输入
可升级版本	必选	默认显示当前支持升级的大、小版本

3. 单击**下一步**进入前置检查页面，只有检查项全部通过后才能进行升级。



4. 单击**升级**开始升级Master Kubernetes版本。
5. 您可以在集群列表页面对应集群状态的位置查看升级进度，单击**查看进度**，系统弹出升级进度框如下所示：

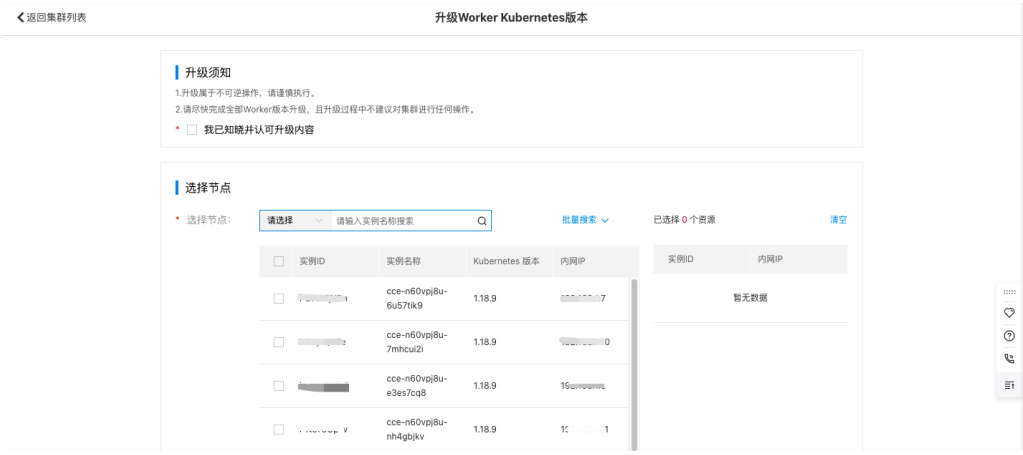


6. 您可以根据需要执行**暂停升级**、**取消升级**操作来暂停升级任务或取消升级任务，通常情况下不建议您暂停或取消。

> **说明**
>
> - **暂停升级**：您可以在集群升级任意阶段对升级任务进行暂停操作，暂停升级后，当前正在执行的任务会继续执行直至完成，还未执行的任务不会再执行。
> - **取消升级**：您可以在暂停升级之后取消本次升级，取消升级后，当前正在执行的任务会继续执行直至完成，还未执行的任务不会再执行。

升级Worker Kubernetes版本

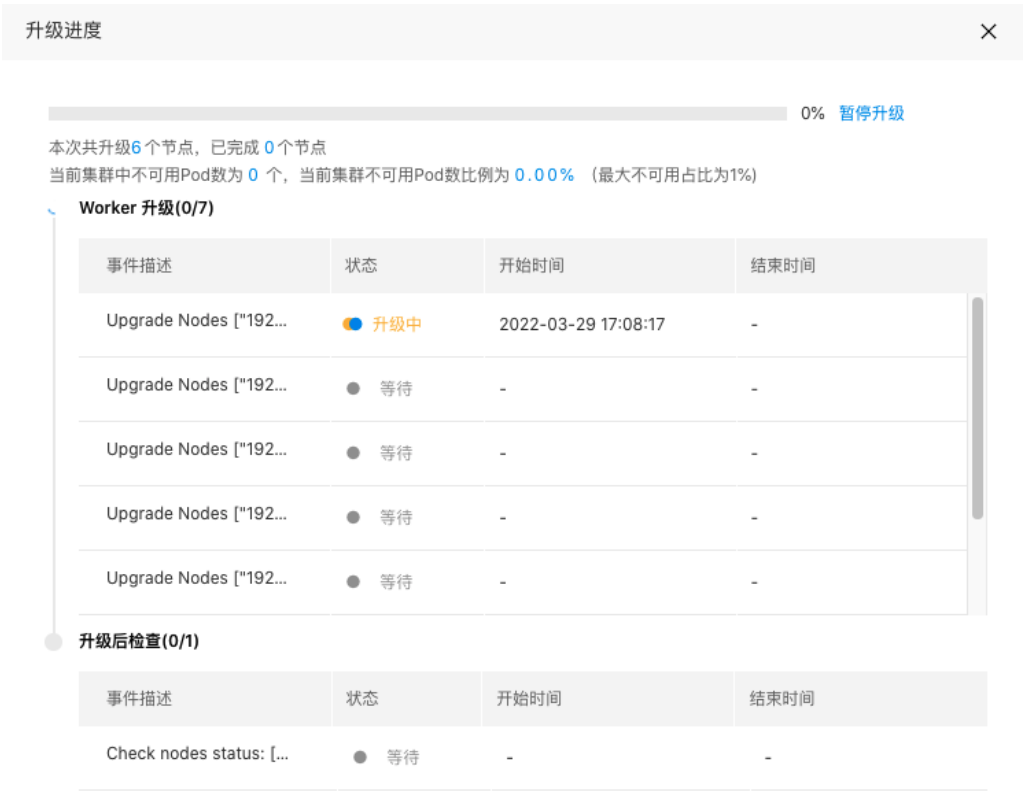
1. 登录容器引擎CCE控制台，选择左侧导航栏中的**集群列表**。
2. 在“集群列表”页面单击目标集群左侧操作中的**更多>升级Worker Kubernetes版本**进入升级页面并完成配置：



配置项	必选/可选	配置说明
升级须知	必选	升级前强烈建议您仔细阅读升级须知并且勾选后才能进行升级
选择节点	必选	选择将要升级的节点，支持分批多次升级
可升级版本	必选	默认显示当前支持升级的大、小版本
升级方式	必选	当前仅支持原地滚动升级，无需重装操作系统，直接升级节点
节点驱逐	必选	默认开启，开启后升级前会将节点上的Pod全部驱逐，建议开启
升级并发数	必选	输入范围为1～20，升级时会根据并发数多个节点同时升级，可提升升级速度
Pod不可用占比	必选	升级过程中Pod不可用数与集群总数占比超过设置的数值时将自动暂停升级任务，默认为20%

3. 单击“升级”开始升级Worker Kubernetes版本。

4. 您可以在集群列表页面对应集群状态的位置查看升级进度，单击**查看进度**，系统弹出升级进度框如下所示：



5. 您可以根据需要执行**暂停升级**、**取消升级**操作来暂停升级任务或取消升级任务，通常情况下不建议您暂停或取消。

> **说明**
>
> - **暂停升级**：您可以在集群升级任意阶段对升级任务进行暂停操作，暂停升级后，当前正在执行的任务会继续执行直至完成，还未执行的任务不会再执行。
> - **取消升级**：您可以在暂停升级之后取消本次升级，取消升级后，当前正在执行的任务会继续执行直至完成，还未执行的任务不会再执行。

托管集群使用说明

注意事项

CCE集群的可用性与集群中管理的Pod、ConfigMap、CRD等资源数量、以及各类资源的Get/List读操作和QPS、Patch/Delete/Create/Update等写操作QPS密切相关，应尽量避免对资源数量较多的集群发起类List操作，避免把集群当数据库使用，写入过多的ConfigMap/CRD/EndPoints 等，影响集群可用性。

常见的类List操作如下（以Pod为例）：

- 带标签查询

  `kubectl get pod -l app=nginx`

- 指定namespace查询

  `kubectl get pod -n default`

- 查询整个集群的Pod

  `kubectl get pod --all-namespaces`

- 通过client-go发起的List请求

  `k8sClient.CoreV1().Pods("").List(metav1.ListOptions{})`

如您有类似查询集群全量资源的需求，建议使用K8s的informer机制通过本地cache查询。对于一些简单的场景，可以通过在List请求中增加ResourceVersion参数，在kube-apiserver cache中查询，如 k8sClient.CoreV1().Pods("").List(metav1.ListOptions{ResourceVersion: "0"})。需注意，即使从kube-apiserver cache查询，如果对大量资源频繁发起List请求，仍会对kube-apiserver内存造成较大压力，仅建议在请求频率较低时使用该方式。

集群规模推荐配置

请在创建托管集群时参考如下推荐配置，根据业务实际情况选择合适的集群规模，以免集群控制面组件负载过大导致集群不可用。

> **说明：**
>
> 当前默认仅支持创建L50规模托管集群，若您想要创建更大规模集群，或者针对存量集群升级至所需的集群规模，您可以提工单联系我们。

示例：若您计划在一个集群中部署50个节点，但是需要部署2000个Pod，则应该选择推荐最大管理节点规模为200（而非50）的集群规模。

集群规模	最大管理节点数	推荐最大Pod数	推荐最大Service数	推荐最大ConfigMap数	推荐最大Secret数	推荐最大PV数	推荐最大PVC数	推荐最大CRD数
L50	50	1500	100	300	300	250	250	1000
L200	200	6000	400	1200	1200	1000	1000	4000
L500	500	15000	1000	3000	3000	2500	2500	10000
L1000	1000	30000	2000	6000	6000	5000	5000	20000
L3000	3000	90000	6000	18000	18000	15000	15000	60000
L5000	5000	150000	10000	30000	30000	25000	25000	100000

> **说明：**
>
> - Pod管理数量不包含集群系统组件产生的Pod（如CoreDNS）。
> - CRD管理数量不包含集群系统组件产生的CRD（如网络相关系统组件产生的CRD）。
> - 建议每种资源类型的所有对象总和不应超过800MiB，每个资源对象大小不超过100KB。

节点管理

添加节点

概述

您可以为已创建的 Kubernetes 集群手动添加工作节点（Worker 节点），以达到扩容集群资源的目的，不支持添加控制节点（Master 节点）。本文介绍如何使用 CCE 控制台添加工作节点。

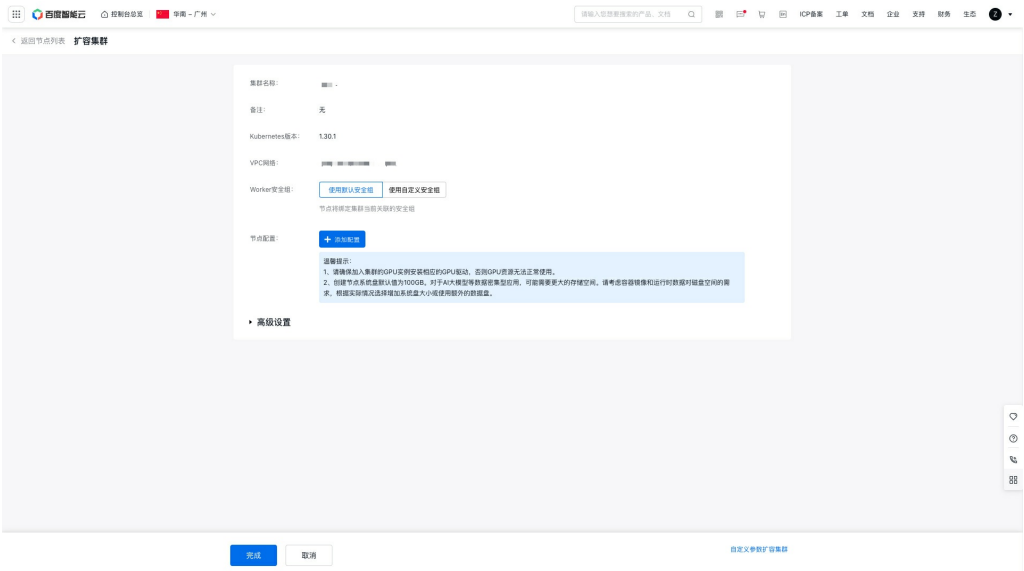
前提条件

已创建 CCE 集群，具体操作请参见 创建集群。

操作步骤

新建节点

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 左侧导航栏选择 **节点管理 > 节点** ，进入节点列表页面，单击 **添加节点** 。
5. 在扩容集群页面中，根据实际需求完成相关参数配置。
6. 完成安全组选项配置。



配置项	必选/可选	配置说明

安全组	必选	节点绑定的安全组，支持使用默认安全组和自定义安全组。
		• 默认安全组：节点将绑定集群当前关联的安全组。
		• 自定义安全组：支持选择普通安全组和企业安全组。
		  • 自动创建：系统将自动为集群创建默认安全组（安全组名称为 {集群ID}-worker-{随机ID}）专用于绑定 Worker 节点，保障集群内节点之间正常通信。
		  • 使用已有：您可以选择已有的安全组用于绑定Worker节点，请确保不影响集群节点之间正常通信。 了解更多 CCE 默认安全组规则。

7. 单击 **添加配置**，完成节点选项配置，相关配置项及说明如下：

配置项	配置说明
节点创建方式	选择 **新建节点**，当前介绍新建云服务器加入集群。
节点类型	按需选择，目前支持云服务器 BCC、弹性裸金属服务器 EBC、裸金属服务器 BBC。
付费方式	按需选择，目前支持后付费、预付费、抢占实例。
可用区	用于筛选可用区下可用的子网。
节点子网	选择为节点分配 IP 地址的子网。
实例配置	根据CPU与内存比值的不同，云服务器提供了多种不同类型的实例规格族。关于实例规格及适用的应用场景您可参考实例规格。
镜像类型及操作系统	您可根据实际需求，选择适合的镜像类型及操作系统。 • 公共镜像：由百度智能云官方提供，仅包含基础操作系统环境。 • 自定义镜像： 通过创建自定义镜像功能生成的镜像，包含系统盘基础操作系统、应用及软件环境的个性化配置。自定义镜像可以帮助您快速创建包含个性化配置的云服务器。 • 共享镜像： 支持用户将自定义镜像共享给其他用户，被共享用户可以通过管理控制台或API查询到其他用户共享的镜像。被共享用户可以使用其他用户共享的镜像创建实例和重装操作系统。
系统盘	用于安装操作系统。非异构计算实例且镜像os是linux，默认大小20GB；若镜像os是windows，默认大小40GB；异构计算实例不区分操作系统默认均为40GB。根据地域以及实例规格的不同，可选择的云盘类型也不同，以页面实际提供的云盘类型为准。
数据盘	即挂载的数据盘，用于提高云服务器的存储容量，默认不选择。云磁盘挂载数量有一定上限，若您想挂载超出上限的云磁盘，请提交工单联系我们。当前，云服务器的系统盘和数据盘均为CDS云磁盘，关于CDS的磁盘类型和使用限制请查看磁盘类型和使用限制。
绑定快照策略	默认关闭绑定快照策略。通过快照，您可以实现磁盘数据备份，磁盘数据恢复以及磁盘镜像的制作。更多关于快照的使用方法和使用限制，请查看快照使用说明。快照目前是收费服务，请查看快照计费说明。
公网IP	如需公网访问，请购买弹性公网IP，或在实例购买成功后绑定已有的弹性公网IP。公网带宽支持以下购买方式： • 包年包月计费：需要提前一次性支付所选时间段内的带宽费用，购买包年包月的云服务器时，费用将合并在实例中收取。 • 按使用流量计费：根据用户实际传输的数据量计费，流量使用没有上限，但可限定最大峰值带宽。 • 按使用带宽计费：根据用户选择的固定带宽值进行计费，最大可购买200Mbps的带宽。
实例名称	您可自定义设置实例名称或由系统随机生成实例名称。
Domain开关	用户打开后，主机名将带有domain后缀，支持DNS解析。
管理用户名	Windows系统的管理员账号为Administrator，Linux系统的管理员账号为root。
管理员密码	根据实例操作系统的不同，密码可供选择的设置方式也不同。 • 用户自定义：自定义设置登录实例的密码。 • 随机生成：购买成功后需登录控制台重置密码，可参考 重置密码。 • 密钥对：Linux操作系统可以选择使用密钥对的方式连接云服务器，SSH密钥对是一种比常规密码更安全的登录云服务器的方式，具体可参考密钥对设置。
数量	填写的购买节点数量。
部署集	指定部署集中创建云服务器实例时，会和处于同一部署集中的其他云服务器实例，按照物理服务器打散，保障发生硬件故障等异常情况下时的服务高可用性。具体可参考部署集进行设置。单个实例最多可加入2个部署集。

8. 单击 **高级设置**，完成高级选项配置，相关配置项及说明如下：

配置项	必选/可选	配置说明
节点显存共享	可选	默认不勾选。勾选该选项后，新添加的节点默认开启 GPU 共享能力。显存共享仅对带有 GPU 设备的节点生效，未带有 GPU 设备的节点将会被忽略。 详情请参考 GPU独占和共享 。 • 注意：开启节点显存共享前提是需要安装GPU Manager 和 AI Job Scheduler 组件。
Kubelet数据目录	可选	volume 文件、plugin 文件等数据存储目录；若已挂载数据盘，建议存储到数据盘。默认值/var/lib/kubelet。
容器数据目录	可选	docker 镜像、容器日志等数据存储目录；若已挂载数据盘，建议存储到数据盘。默认值/home/cce/containerd。
部署前执行脚本	可选	节点部署前将自动执行该脚本，您需要自行保证脚本的可重入及重试逻辑。脚本内容和产生日志将写入节点的/usr/local/cce/scripts/目录下。
部署后执行脚本	可选	节点部署后将自动执行该脚本，您需要自行确定脚本的执行情况。脚本内容和产生日志将写入节点的/usr/local/cce/scripts/目录下。
封锁节点	必选	默认关闭。选择开启封锁节点后，节点将处于不可调度状态，新创建的Pod将不会调度到该节点上。若需要取消封锁，请执行kubectl uncordon命令。封锁节点将减少该节点所属集群的剩余可用资源配额，若集群本身预留的可用资源不足，将会影响后续正常业务的调度和存量业务的运行。
自定义kubelet参数	可选	部署节点时，可自定义 kubelet 参数，具体信息请参见自定义 Kubelet 参数。
资源标签	可选	资源标签支持您按各种标准（如用途、所有者或项目）对云上资源进行分类，每个标签包含键和值两部分，具体可参考标签功能进行设置。 • 默认为实例关联资源 CDS、EIP 统一添加标签，可以选择关闭。
标签（Labels）	可选	K8S 标签是管理和选择 K8S 对象的标识，将自动绑定到添加的节点，具体信息请参见 K8S 标签说明。
污点（Taints）	可选	节点污点和 Pod 容忍共同作用，确保 Pod 不被调度到节点上，除非 Pod 能够容忍节点污点，具体信息请参见 污点和容忍说明。
注解（Annotations）	可选	Annotations（注解）是一种将非标识性元数据附加到对象上的机制，每个 Annotation 包含键和值两部分。具体信息参见 注解说明。

9. 单击 **完成**，完成添加节点，您可以在节点列表中查看添加的节点信息。

添加已有服务器

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点** ，进入 节点列表页面，单击 **添加节点** 。
5. 在扩容集群页面中，根据实际需求完成相关参数配置。
6. 完成安全组选项配置（略）。
7. 单击 **添加配置**，完成节点选项配置。

配置项	必选/可选	配置说明
节点创建方式	必选	选择 **使用已有服务器**，当前介绍添加已有云服务器加入集群，只支持选择当前集群私有网络下可用的实例。
节点类型	必选	目前支持通过云服务器 BCC、弹性裸金属服务器 EBC、裸金属服务器 BBC、高性能应用服务 HPAS 筛选可用实例。
重装操作系统	必选	重装系统开关，将对所选实例进行重装操作系统，会清除系统盘内数据且不可恢复（云磁盘数据不受影响），默认开启。 • 添加已有实例到集群中，无论是否重装操作系统，都会保留实例原有的自定义驱动（GPU Driver/CUDA/cuDNN等）或“用户数据注入”配置。 • 添加节点不重装操作系统，服务器实例上的旧数据可能会导致节点添加失败或出现其它问题，推荐选择重装操作系统

磁盘挂载	可选	可将容器和镜像数据存储在数据盘或本地盘中。
		• 勾选：需要填写设备名称、挂载目录。需要注意以下几点：
		  • 系统将会对所选的全部节点设备格式化为ext4，已经格式化的设备将直接挂目录。被格式化的数据盘内原有数据将会丢失，请提前备份重要数据。
		  • 您填写的挂载设置会对本批次添加节点全部生效，请确认填写的设备名称符合预期。
		  • 若输入的设备名称与所选节点的实际名称不匹配，系统会对不匹配的节点报错并终止节点初始化流程。若填写的挂载目录不存在，系统会自动创建对应目录。
		  • 若所选节点未挂载磁盘，则不会自动挂载新的磁盘。
		• 不勾选：不设置磁盘挂载选项，可手动或者使用脚本挂载。

镜像类型	必选	重装系统开启后，支持选择镜像类型，不同节点类型支持选择的镜像类型不同。
操作系统	必选	选择实例重装的操作系统。
管理员用户名	必选	登录实例的用户名，默认为 root。
登陆方式	必选	支持**密钥对**和**自定义密码**两种方式。
密钥对 / 密码	必填	• 若登陆方式选择密钥对，可选择已有密钥对或创建密钥对。 • 若登陆方式选择自定义密码，请输入登录实例的密码，请确保输入的密码与实例密码一致，否则节点将添加失败。若您忘记了待操作实例的密码，请参见 重置密码。

8. 单击 **高级设置**，完成高级选项配置（略）。

9. 单击 **完成**，完成添加节点，您可以在节点列表中查看添加的节点信息。

移出节点

若当您不再需要某 Work 节点继续工作时，可以在控制台将节点从节点组或集群中移出，不支持移出控制节点。请参见本文的注意事项，在业务低峰期执行标准化操作。

前提条件

- 您已成功创建一个 Kubernetes 集群,详情请参见创建集群。

注意事项

- 移出节点可能会导致集群无法为容器提供足够的 CPU 和内存资源，请谨慎操作。
- 移出节点会涉及到 Pod 迁移，可能会影响业务，建议您在业务低峰期操作。
- 移出过程中可能存在非预期风险，请提前做好相关数据备份。
- 移出节点时，请在控制台执行标准化操作。请勿使用 kubectl delete node 方式手动移除节点。
- 移出节点时，如果选择进行节点排水操作，请确保集群其他节点的资源充足，以避免业务 Pod 无法调度。
- 需检查待移出节点上的 Pod 的节点亲和性规则和调度策略，确保 Pod 不会因为节点移除而无法调度到其他节点。

- 未纳入节点组管理的节点，不支持配置缩容保护。如需配置节点缩容保护，请先将节点添加至节点组，请参见添加节点。

> **说明**：如移出后仍需将实例添加到节点组/集群中，请参见添加节点。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点** ，进入节点列表页面。
5. 在节点列表中，选择需要移出集群的节点，单击 **移出节点**。

> **注意：**
>
> - 移出节点默认不会删除节点对应的实例及关联资源。您可以勾选同时删除实例和关联公网 IP 和云磁盘，只支持同时删除后付费的资源，预付费的资源不删除。预付费的实例会在计费周期到期后自动释放，您可以在实例到期前申请退款，提前释放实例，或将包年包月实例转按量付费实例后释放。
> - 移出节点组的节点，默认会减少节点组期望节点数，您可以取消勾选同时减少节点组期望节点数，取消勾选节点组会通过继续扩容来维持当前期望实例数量，但不包含移出失败、删除失败的节点。
> - 如果节点上存在Pod需要驱逐，需要勾选"进行节点排水"。但是，节点上由 DaemonSet 管理的 Pod 不会被排水。
> - 移出异常状态的节点请不要勾选"进行节点排水"，否则可能导致移出失败。
> - 移出节点时，如果选择移出节点重装操作系统，会清除系统盘内数据且不可恢复（云磁盘数据不受影响）。

6. 单击 **确定**，完成移出节点，您可以在节点列表中查看正在移出的节点，直到从列表中消失。

设置节点封锁

概述

您可以通过设置节点封锁来控制节点的调度状态，从而控制节点是否能够调度上 Pod。节点开启封锁后，将不允许新的 Pod 调度到该节点上，若希望节点上能够正常调度上 Pod ，则需要关闭节点封锁。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点** ，进入节点列表页面。
5. 在节点列表中，选择需要设置封锁的节点，单击 **封锁设置**。

6. 根据您需要选择开启或关闭封锁节点，勾选我也了解上述说明，确认执行所选操作后，单击 **确定**，完成设置节点封锁。

设置GPU显存共享

您可以通过给 GPU 节点设置显存共享，开启显存共享后，节点上的 GPU 资源支持按照显存大小分配给多个容器，从而提高资源利用率。

使用说明

显存共享设置仅对 GPU 节点生效。

前提条件

集群已成功安装 CCE GPU Manager、 AI Job Scheduler 组件。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点** ，进入节点列表页面。
5. 在节点列表中，选择需要设置显存共享的 GPU 节点，单击 **显存共享设置**。

6. 根据您的需要选择开启或关闭显存共享，勾选我已了解上述说明，确认执行所选操作后，单击 **确定**，完成设置显存共享。

> **注意：**
>
> - 若所选节点属于节点组，当节点组开启了同步更新存量节点的标签、污点及注解能力后，节点组配置会覆盖节点配置。

管理节点标签

在Kubernetes中，标签（Labels）和污点（Taints）可用于管理和调度集群中的资源。标签支持对Kubernetes 对象（例如节点）进行标识和分类。建议已参见 Kubernetes 官方文档 Labels 和 Selectors 了解标签的构成与用法。 本文介绍如何在容器引擎CCE控制台上管理节点标签，包括批量新增标签和快速删除标签。

批量添加节点标签

前提条件

- 您已成功创建一个 Kubernetes 集群，详情请见创建集群。

操作步骤

1. 登录百度智能云官网，并进入容器引擎 CCE 管理控制台。
2. 单击左侧导航栏中的**集群管理** > **集群列表**，在集群列表页面中，单击目标集群名称进入集群管理页面。
3. 在集群管理页面单击**节点** > **标签与污点管理**进入标签与污点管理页面。

4. 在标签与污点管理页面批量选择节点，单击**编辑标签**。

> - 选择单个节点，可管理该节点所有标签，系统定义标签除外。
> - 节点创建出来会存在一些系统标签，并且是无法删除的。
> - 批量编辑标签只能添加标签，不能管理已有标签。

5. 在弹出的“编辑标签”框中设置标签键和标签值，节点标签使用说明请见 K8S 标签说明。

6. 单击“确定”按钮，完成批量添加节点标签。

删除节点标签

前提条件

- 您已成功创建一个Kubernetes集群，详情请见创建集群。

操作步骤

1. 登录百度智能云官网，并进入容器引擎CCE管理控制台。
2. 单击左侧导航栏中的**集群管理** > **集群列表**，在集群列表页面中，单击目标集群名称进入集群管理页面。
3. 在集群管理页面单击**Worker** > **标签与污点管理**进入标签与污点管理页面。

4. 在标签与污点管理页面选择某个节点，单击标签上的删除图标。
5. 在弹出的删除确认框中单击“确定”按钮，完成快速删除节点标签。

CCE 节点默认添加的标签

CCE 会为集群中的节点默认添加一些标签。

所有节点都拥有的标签：

标签	说明	示例
beta.kubernetes.io/cce-instance-type	节点实例规格	"bcc.g5.c2m8"
node.kubernetes.io/cce-instance-type	节点实例规格	"bcc.g5.c2m8"
beta.kubernetes.io/instance-type	节点机型（k8s 1.17后已过时）	"BCC"
node.kubernetes.io/instance-type	节点机型	"BCC"
beta.kubernetes.io/arch	节点架构	"amd64"
kubernetes.io/arch	节点架构	"amd64"
beta.kubernetes.io/os	节点操作系统类型	"linux"
kubernetes.io/os	节点操作系统类型	"linux"
cce.baidubce.com/gpu-share-device-plugin	GPU节点独占或共享	"disable/enable"
cce.baidubce.com/kubelet-dir	记录instance kubelet 根目录的 Hex 编码	"2f7661722f6c69622f6b7562656c6574"
cluster-id	集群id	"cce-xxx"
cluster-role	节点角色	"node/master"
kubernetes.io/hostname	节点名称	"192.168.0.60"
failure-domain.beta.kubernetes.io/region	节点所在地域（k8s 1.17后已过时）	"bj"
topology.kubernetes.io/region	节点所在地域	"bj"
failure-domain.beta.kubernetes.io/zone	节点所在可用区（k8s 1.17后已过时）	"zoneA"
topology.kubernetes.io/zone	节点所在可用区	"zoneA"
beta.kubernetes.io/instance-gpu	节点是否有GPU资源	"true"
beta.kubernetes.io/instance-npu	节点是否有NPU资源	"false"
kubernetes.io/tor	实例所属的物理交换机的ID	"sw-hzinnW1kCF96rcLUR664Nw"

仅云边集群节点拥有的标签：

标签	说明	示例
pools.apps.cce.io/cloud	节点是否是云端节点；仅true时该Label存在	"true"
pools.apps.cce.io/bcc	节点是否是BCC；仅true时该Label存在	"true"
pools.apps.cce.io/{BEC-CITY}-{Provider}	BEC节点地点城市和运营商；仅BEC节点存在	pools.apps.cce.io/guangzhou-u

管理污点

概述

污点是节点的一种属性，用于控制某些Pod不被调度到某节点上。如需允许调度器将Pod调度到该节点上，还需声明对应的容忍。常见场景如下：

- 专用节点：指定节点仅供特定用户或工作负载使用。
- 特殊硬件节点：确保只有需要特殊硬件（例如GPU）的Pod被调度到这些节点上。
- 基于污点的驱逐：通过为节点添加 NoExecute 的污点，并为待保留的Pod添加对应容忍，实现Pod的选择性驱逐。

本文介绍如何在容器引擎 CCE 控制台上管理污点，包括批量新增污点和快速删除污点。

批量添加污点

前提条件

- 您已成功创建一个 CCE 集群，详情请见创建集群。

操作步骤

1. 登录百度智能云官网，并进入容器引擎 CCE 管理控制台。
2. 单击左侧导航栏中的 **集群管理** > **集群列表** ，在集群列表页面中，单击目标集群名称进入集群管理页面。
3. 在集群管理页面单击 **节点管理** > **节点**，在节点列表右上方点击**标签与污点管理**，进入标签与污点管理页面。
4. 在标签与污点管理页面批量选择节点，单击 **编辑污点** 。

5. 在弹出的“编辑标签”框中设置污点键、值和 Effect，更多信息请见污点和容忍说明。

6. 单击“确定”按钮，完成批量添加污点。

> **说明**
>
> - 选择单个节点，可管理该节点所有污点。批量编辑污点时只能添加污点，不能管理已有污点。
> - 节点污点和 Pod 容忍共同作用。节点设置污点后，可避免 Pod 调度到节点上，或者将 Pod 从节点驱逐。除非 Pod 容忍能够节点污点相匹配。
> - 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。

删除污点

前提条件

- 您已成功创建一个 CCE 集群,详情请见创建集群。

操作步骤

1. 登录百度智能云官网，并进入容器引擎 CCE 管理控制台。
2. 单击左侧导航栏中的 **集群管理 > 集群列表** ，在集群列表页面中，单击目标集群名称进入集群管理页面。
3. 在集群管理页面单击**节点管理 > 节点**，在节点列表右上方点击**标签与污点管理**， 进入标签与污点管理页面。
4. 在标签与污点管理页面选择某个节点，单击污点上的删除图标。
5. 在弹出的删除确认框中单击“确定”按钮，完成快速删除污点。

系统污点说明

当节点出现某些问题时，Kubernetes会自动给节点添加一个污点，当前内置的污点包括：

- node.kubernetes.io/not-ready：节点未准备好。这相当于节点状况 Ready 的值为 "False"。
- node.kubernetes.io/unreachable：节点控制器访问不到节点。这相当于节点状况 Ready 的值为 "Unknown"。
- node.kubernetes.io/memory-pressure：节点存在内存压力。
- node.kubernetes.io/disk-pressure：节点存在磁盘压力。
- node.kubernetes.io/pid-pressure：节点存在 PID 压力。
- node.kubernetes.io/network-unavailable：节点网络不可用。
- node.kubernetes.io/unschedulable：节点不可调度。
- node.cloudprovider.kubernetes.io/uninitialized：如果 kubelet 启动时指定了一个“外部”云平台驱动， 它将给当前节点添加一个污点将其标志为不可用。在 cloud-controller-manager 的一个控制器初始化这个节点后，kubelet 将删除这个污点。

自定义Kubelet参数

Kubelet 提供了一系列参数用于调整节点行为。本文介绍如何在创建节点时自定义 Kubelet 参数。

前提条件

- 已创建 CCE 集群，具体操作请参见创建 CCE 集群。

操作步骤

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**节点管理>节点**。
2. 在节点列表中单击**添加节点**操作，在**高级设置**中设置节点的自定义 Kubelet 参数，具体参数说明请见下方**支持的 Kubelet 参数**。

3. 单击**确定**，完成节点 Kubelet 参数配置。

支持的 Kubelet 参数

目前支持配置如下表所示的 Kubelet 自定义参数。

> **注意**
>
> - 修改 topologyManagerPolicy、topologyManagerScope 和特性门控（featurGates）中的 QoSResourceManager 参数会重启 Kubelet，并且以更改后的策略重新计算容器实例的资源分配，因此可能会导致已经运行的容器实例重启，甚至无法进行资源分配。请谨慎操作。
> - 随着 Kubernetes 版本的演进，可能会有部分参数或 featureGates 被标记弃用或者从代码移除。如果受容器服务 Kubernetes 版管理的自定义参数在新的版本不再适用时，相关配置会在集群升级期间被移除弃用。

字段	说明	默认值	建议取值范围
kubeReserved	kubernetes 系统预留的资源配置。	根据节点规格自动设置。	详情请参见CCE资源预留说明。
systemReserved	系统预留的资源配置。	根据节点规格自动设置	详情请参见CCE资源预留说明。
allowedUnsafeSysctls	设置允许使用的非安全的sysctl或sysctl通配符（以*结尾的通配符模式）。是使用英文半角逗号（,）分隔的字符串序列。 • 使用此参数前，请仔细评估相关风险并自行保证可用性。	不涉及	支持下述前缀的 sysctl 配置： • kernel.shm • kernel.msg • kernel.sem • fs.mqueue. • net.

			• **注意**：这些sysctl 参数都支持容器化，但是默认容器内不允许配置
containerLogMaxFiles	容器的日志文件个数上限。此值必须大于等于2，且容器运行时需为containerd。	5	2~10
containerLogMaxSize	容器日志文件轮换生成新文件的最大阈值。容器运行时需为containerd。	10Mi	不涉及
cpuCFSQuota	允许为设置了 CPU 限制的容器实施CPU CFS 配额约束。	true	可选值 • true • false
cpuCFSQuotaPeriod	设置 CPU CFS 配额周期值。 需确保CustomCPUCFSQuotaPeriod特性门控已被启用。	100ms	介于1毫秒和1秒之间，包含1毫秒和1秒。
cpuManagerPolicy	指定节点的CPU 管理器策略。	none	可选值 • none • static
eventBurst	事件记录的个数的突发峰值上限。在遵从 event-qps 阈值约束的前提下临时允许事件记录达到此数目。仅在 event-qps 大于 0 时使用。	50	1~100
eventRecordQPS	每秒可生成的事件数量。	50	1~50
evictionHard	触发Pod驱逐操作的一组硬性门限。	"imagefs.available": "15%", "memory.available": "100Mi", "nodefs.inodesFree": "5%", "nodefs.available": "7%"	不涉及
evictionSoft	设置一组驱逐阈值。	无	不涉及
evictionSoftGracePeriod	设置一组驱逐宽限期。 • 需已设置evictionSoft。	"5m"	不涉及
featureGates	实验性特性的特性开关组。每个开关以key=value的形式表示。更多信息，请参见Feature Gates。 • 使用此参数前，请仔细评估相关风险并自行保证可用性。	不涉及	不涉及
ImageGCHighThresholdPercent	配置镜像的磁盘用量百分比阈值，一旦镜像用量超过此阈值，镜像垃圾收集会一直运行。百分比是用这里的值除以 100 得到的，所以此字段取值必须介于 0 和 100 之间，包括 0 和 100。配置时，此值需大于imageGCLowThresholdPercent取值。	85	60～95
ImageG	配置镜像的磁盘用量百分比阈值，镜像用量低于此阈值时不会执行镜像垃圾收集操作。垃圾收集操作也将此作为最		

CLowThresholdPercent	低磁盘用量边界。 百分比是用这里的值除以 100 得到的，所以此字段取值必须介于 0 和 100 之间，包括 0 和 100。配置时，此值需小于imageGCHighThresholdPercent取值。	80	30～90
kubeAPIBurst	每秒发送到APIServer的突发请求数量上限。	500	1~500
kubeAPIQPS	与APIServer通信的每秒查询个数。	500	1~500
maxPods	Kubelet 上可以运行的最大 Pod 数量。	128	不涉及，取决于机器规格和容器网络规划等物理资源配置。
memoryManagerPolicy	内存管理器需要使用的策略。 • 注意：仅Kubernetes 1.21及以上版本支持该参数。	None	可选值： • None • Static 使用此策略时，需已完成以下操作： 1. MemoryManager特性门控已被启用。 2. 已手动管理内存资源预留（systemReserved、kubeReserved、evictionHard、reservedMemory）并确保配置正确，即kube-reserved + system-reserved + eviction-hard = sum(reserved-memory)，以避免节点不可用。更多信息，请参见预留内存标志。
podPidsLimit	单pod最大进程数。	-1	取值根据用户实际情况来定，推荐默认值，-1表示无限制。
readOnlyPort	kubelet无鉴权只读端口。此端口上的服务不支持身份认证或鉴权。这一端口号必须介于 1 到 65535 之间， 包含 1 和 65535。将此字段设置为 0 会禁用只读服务。 **注意：**关于开放kubelet容器监控只读端口（10255）的风险。	0	0
registryPullQPS	镜像仓库的QPS上限。如果 --registry-qps 大于 0，用来限制镜像仓库的 QPS 上限。设置为 0，表示不受限制。	5	1~50
registryBurst	突发性镜像拉取的个数上限。在不超过 registration-qps 设置值的前提下暂时允许此参数所给的镜像拉取个数。仅在 --registry-qps 大于 0（默认值为 10）时使用。	10	1~100
reservedMemory	NUMA节点内存预留列表。 • 注意：仅Kubernetes 1.21及以上版本支持该参数。	不涉及	不涉及
resolvConf	容器内使用的DNS 解析配置文件。 • 注意：如果此值设置为空字符串，则会覆盖 DNS 解析的默认配置， 本质上相当于禁用了 DNS 查询。	/etc/resolv.conf	不涉及
runtimerequesttimeout	用来设置除长期运行的请求（pull、 logs、exec 和 attach）之外所有运行时请求的超时时长。	120s	不涉及
serializeImagePulls	串行拉取镜像。建议不要在 docker 守护进程版本低于 1.9 或启用了 Aufs 存储后端的节点上更改默认值。	true	可选值： • false：以支持并行拉取镜像，提高Pod启动速度。 • true：建议值，支持串行拉取镜像。
			可选值

topologyManagerPolicy	拓扑管理器策略。借助NUMA架构，数据可以被分配到相同的NUMA节点，以减少跨节点访问，提高系统性能。拓扑管理器可以作出与拓扑结构相对应的资源分配决定。更多信息，请参见控制节点上的拓扑管理策略。	none	可选值 • none（默认）：不执行拓扑管理策略。 • restricted：当同一 NUMA 节点的 Pod 无法满足资源请求时，kubelet 不允许跨 Pod 实现资源请求。 • best-effort：当同一 NUMA 节点的 Pod 无法满足资源请求时，kubelet 允许跨 Pod 实现资源请求。 • single-numa-node：kubelet仅允许在同一NUMA节点的Pod上分配CPU和设备资源。
topologyManagerScope	拓扑管理器作用范围。拓扑提示信息由提示提供者生成，提供给拓扑管理器。	container	可选值 • container • Pod

kubelet容器监控只读端口风险提示

由于社区版 Kubernetes 的只读容器监控透出存在信息泄露风险，百度智能云 CCE 容器服务集群不默认开放kubelet容器监控只读端口（10255）。

风险申明

在被攻破集群节点后，社区版 Kubernetes 默认开放的 kubelet 容器监控只读端口（10255）可能会被获取部分应用信息，存在信息泄露风险。因此，百度智能云 CCE 容器服务集群不默认开放 kubelet 容器监控只读端口（10255）。

若您正在使用或计划依赖 kubelet 只读端口（10255），请充分了解并知悉数据安全风险。了解后，再按需通过 kubelet 参数手动开启只读端口（10255）。请参见下文手动开启 kubelet 只读端口（10255）。

手动开启kubelet只读端口（10255）

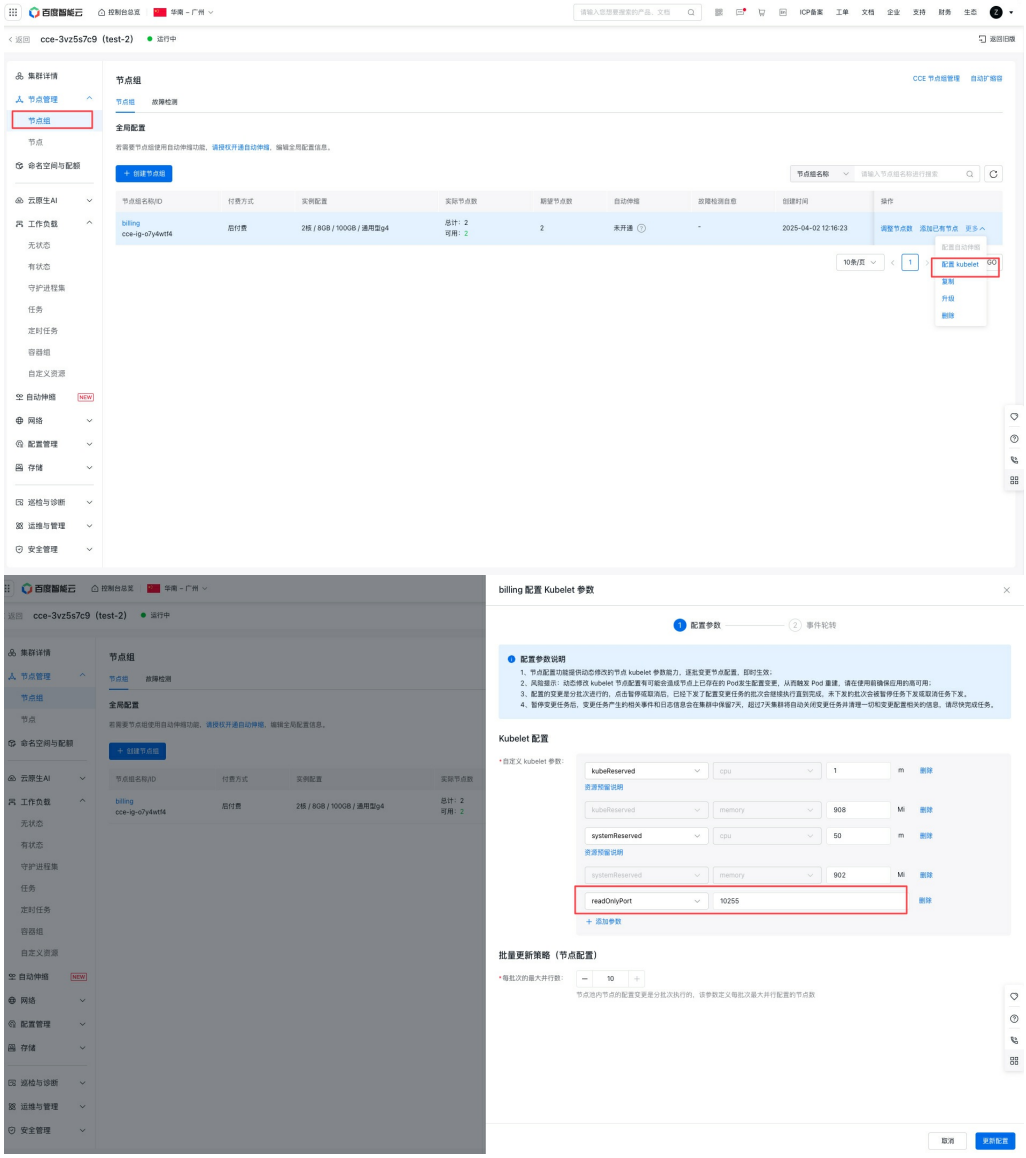
若您正在使用或计划依赖 kubelet 只读端口（10255），请充分了解并知悉数据安全风险。了解后，再按需通过 kubelet 参数手动开启只读端口（10255）。

方式一：添加节点开启 kubelet 只读端口

1. 登录百度智能云官网，并进入容器引擎 CCE 管理控制台。
2. 单击左侧导航栏中的 **集群管理** > **集群列表** ，在集群列表页面中，单击目标集群名称进入集群管理页面。
3. 左侧导航栏选择 **节点管理** > **节点** ，进入节点列表页面。
4. 在节点列表中单击**添加节点**操作，在**高级设置**中设置节点的自定义 kubelet 参数，修改参数**readOnlyPort**的值为10255。

5. 单击**确定**，完成节点10255端口的开启。 **方式二：节点组配置 kubelet 开启 kubelet 只读端口**
6. 登录百度智能云官网，并进入容器引擎 CCE 管理控制台。

7. 单击左侧导航栏中的 **集群管理 > 集群列表** ，在集群列表页面中，单击目标集群名称进入集群管理页面。
8. 左侧导航栏选择 **节点管理 > 节点组** ，进入节点组列表页面。
9. 在节点组列表选择需要的节点组，点击**更多**，选择**配置 kubelet**，在配置kubelet参数页面设置自定义 kubelet 参数，添加参数**readOnlyPort**的值为10255。



节点排水

概述

节点排水（Drain Node）是 Kubernetes 中的一个操作，它是指将一个节点（Node）上的所有 Pods 驱逐，以便可以安全地关闭或维护该节点。这个操作通常在节点需要维护或升级时执行。您可以通过 CCE 控制台使用节点排水功能，系统会将节点设置为封锁状态，然后安全地将节点上所有 Pod（不含守护进程 DaemonSet 管理的 Pod）从本节点驱逐。本文介绍如何使用 CCE 控制台进行节点排水。

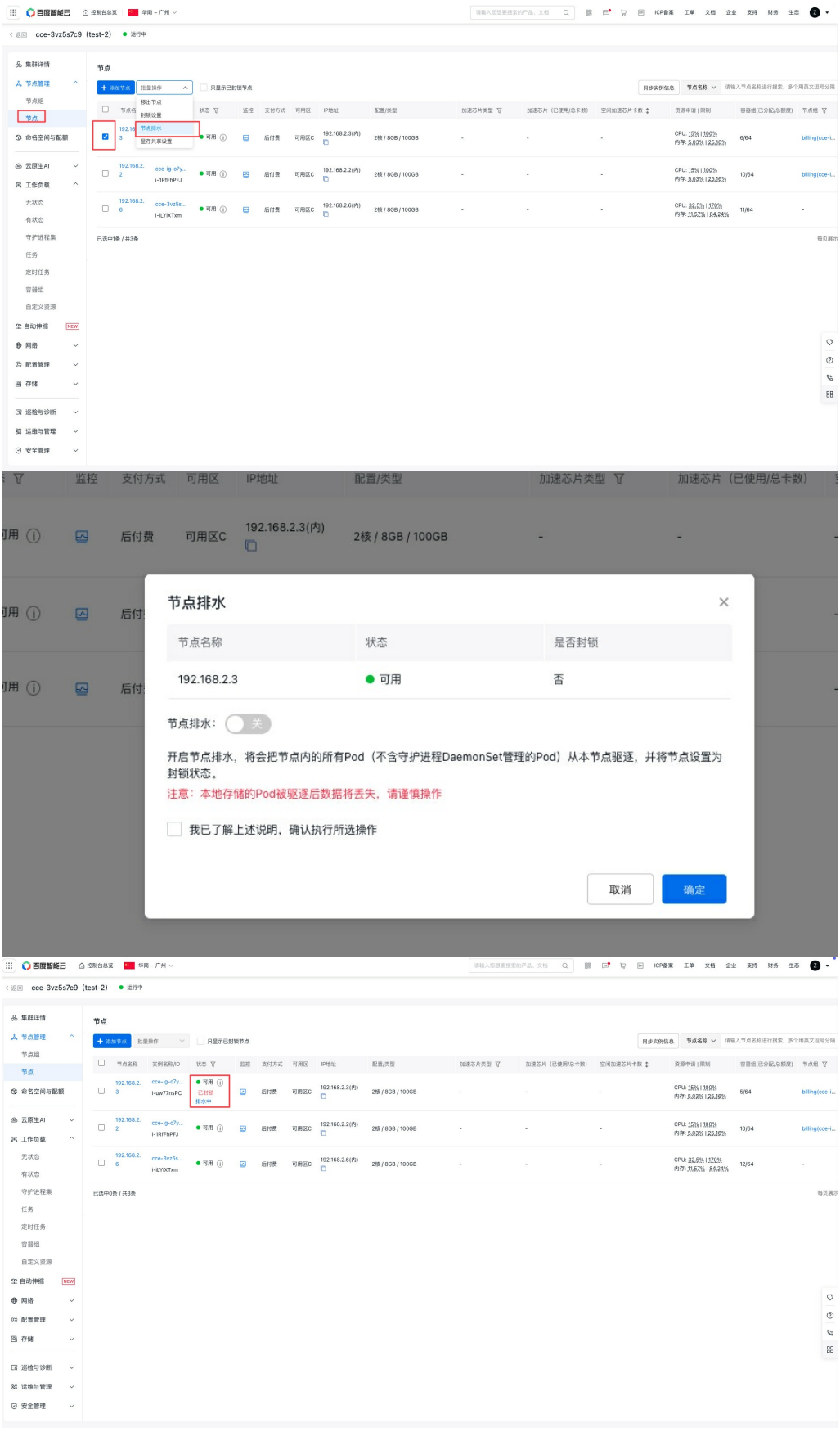
操作步骤

单节点排水

1. 登陆 容器引擎 CCE 控制台，选择左侧导航中的**集群管理 > 集群列表**。
2. 在集群列表页面中，单击目标集群名称/ID，进入集群详情页面。
3. 在左侧导航栏中选择**节点管理 > 节点**，切换至节点列表页面。
4. 找到目标节点，单击操作列中的**更多操作 > 节点排水**。
5. 在弹出的“节点排水”窗口中，确认节点名称和状态后，将**节点排水**的开关状态设为**开启**，并勾选**我已了解上述说明，确认执行所选操作**，点击**确定**即可进行节点排水操作。

注意：

- 开启节点排水，会将当前节点内的所有Pod（不含守护进程DaemonSet管理的Pod）驱逐，并将当前节点设置为封锁状态。
- 本地存储的 Pod 被驱逐后数据将丢失，请谨慎操作。



多节点排水

1. 登陆 容器引擎 CCE 控制台，选择左侧导航中的**集群管理 > 集群列表**。

2. 在集群列表页面中，单击目标集群名称/ID，进入集群详情页面。
3. 在左侧导航栏中选择**节点管理 > 节点**，切换至节点列表页面。
4. 勾选多个需要进行节点排水的节点，点击上侧**更多操作 > 节点排水**。
5. 在节点排水弹窗中，确认需要进行操作的节点，将**节点排水**的开关状态设为**开启**，并勾选**我已了解上述说明，确认执行所选操作**，点击**确定**即可进行节点排水操作。

> **注意：**
>
> - 开启节点排水，会将节点内的所有Pod（不含守护进程DaemonSet管理的Pod）从本节点驱逐，并将节点设置为封锁状态。
> - 本地存储的 Pod 被驱逐后数据将丢失，请谨慎操作。

移出节点时进行节点排水 移出节点时若节点上存在 Pod 需要驱逐，可同时勾选"进行节点排水"。

> **注意：**
>
> - 节点上由 DaemonSet 管理的 Pod 不会被排水。
> - 节点状态异常时不要勾选“节点排查”，否则会导致节点移出失败。

设置节点缩容保护

概述

您可以通过设置节点缩容保护来保护节点免于缩容。节点开启缩容保护后，不受节点组自动缩容或调整期望节点数影响。若希望节点上能够正常被自动缩容 ，则需要关闭节点缩容保护。

操作步骤

> **注意：**
>
> - 节点未纳入节点组管理时，不支持配置节点缩容保护。如需使用节点缩容保护，需要先将节点添加至节点组。

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点** ，进入节点列表页面。
5. 在节点列表中，选择需要设置节点缩容保护节点，在操作列单击**更多**>选择 **节点缩容保护**。
6. 根据您的需要选择开启或关闭节点缩容保护，勾选我已了解上述说明，确认执行所选操作后，单击 **确定**，完成设置节点缩容保护。

节点组管理

节点组介绍

为帮助您更好地管理 Kubernetes 集群内的节点，容器引擎 CCE 引入节点组概念。本文主要介绍节点组的概述、核心概念、使用限制等。

节点组概述

节点组是具有相同属性的一组节点的逻辑集合，允许对节点进行统一的管理和运维，例如节点升级、弹性伸缩等。通过创建多个节点组，用户可以进一步实现资源的隔离，例如为不同类型的工作负载部署独立的节点组，分组管理不同属性的节点等。一个集群中可以创建多个不同配置和类型的节点组。节点组的配置包含节点的属性，例如节点规格、可用区、标签、污点等。这些属性可以在创建节点组时指定，也可以在创建完成后进行编辑修改。

核心概念

首次使用节点组前，建议您了解节点组相关的概念。

概念	说明
节点组	具有相同配置的一组节点的集合，可以节点组为对象进行操作，完成一组节点的创建、管理、删除，以及节点数量调整和自动扩缩容
节点	集群的 Worker 节点，一个节点对应一个 BCC 实例，包含了运行容器的一切必要条件
节点配置	节点组创建节点的配置模版，包括所在可用区、节点子网、实例规格等，目前一个节点组对应一种节点配置，且暂不支持修改。节点组扩容即根据该配置创建节点
实际节点数	当前节点组中实际可用的节点数，即处于 Ready 状态的节点数量
期望节点数	期望节点组中保持的可用节点数
调整节点数	以节点组为操作对象，手动扩容或者缩容不用手动创建或删除具体的节点，而是直接设置节点组的期望节点数。节点组将自动对比当前实际节点数，根据期望节点数增加或减少节点数量（按节点配置自动创建节点、释放节点组中的节点），以达到扩容或缩容的目的
自动伸缩	K8S 提供的 Cluster Autoscaler（CA）功能。开启自动伸缩，集群将根据扩缩容策略，在资源不足时自动创建新节点，当资源富余时自动释放多余节点，以保障集群资源始终足以支撑业务负载，同时最大化节约成本
伸缩范围	启用自动伸缩，节点组将通过在该伸缩范围内自动调整期望节点数，以实现自动扩缩容
伸缩活动	节点组的每次扩缩容、添加节点、移除节点都会触发伸缩活动。触发伸缩活动后，所有扩容和缩容动作都交由系统自动完成，并留下相关记录。您可以通过节点组的伸缩活动查看节点组的历史伸缩活动记录
扩容优先级	自动扩容时，将以扩容优先级顺序，对已启用自动伸缩的节点组进行扩容。数字越小，优先级越高
缩容阈值	应用于自动伸缩，节点组内的节点资源（cpu、mem）分配率均低于设定阈值时，可能会触发自动缩容
缩容触发时延	应用于自动伸缩，在配置的缩容触发时延内，节点资源利用率持续低于缩容阈值，可能会触发自动缩容
最大并发缩容数	应用于自动伸缩，该值为整数，表示并发缩容资源利用率为0的节点数目
扩容后缩容启动间隔	应用于自动伸缩，该值为整数，单位分钟，扩容出来的节点经过此间隔后开始评估是否可以被缩容
扩缩容策略	应用于扩缩容时多种机型的扩容策略，支持机型配置顺序、多子网均匀分布 详见 配置扩缩容策略

使用限制

资源名称	默认配额 / 使用限制
节点组数量	不限制，只要该集群节点数总和不超过“单集群最大节点数”配额即可，需要提高配额请提交工单
单节点组最大节点数	不超过“单集群最大节点数”配额，该集群节点数总和也不可超过该配额，需要提高配额请提交工单
期望节点数	不超过“单集群最大节点数”配额，该集群节点数总和也不可超过该配额，需要提高配额请提交工单
单节点组节点配置数	一个节点组对应一种节点配置
自动伸缩范围	0 - 200，最小节点数不可大于最大节点数
扩容优先级	0 - 100

节点组管理

概述

本文介绍如何通过容器引擎 CCE 控制台在集群中创建节点组，以及节点组的查看、管理及删除等操作。如需了解相关概念和使用限制，请参见 节点组介绍。

创建节点组

1. 登录百度智能云容器引擎 CCE 控制台，在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
2. 在左侧导航栏，点击 "节点组"，进入节点组列表页。
3. 点击 "创建节点组" 按钮或者 "马上创建" 链接，进入节点组创建页面。
4. 填写节点组基本配置，配置项及说明如下：

- **节点组信息**

配置项	说明
节点组名称	自定义，支持大小写字母、数字、中文以及“-_ /.”；必须以字母开头，长度1-65。
VPC 网络	默认是集群的 VPC 网络，不支持更改。
Worker 安全组	支持配置“使用默认安全组”和“使用自定义安全组”。使用默认安全组会绑定集群当前关联的安全组，若现有的安全组不合适，可以自定义安全组。请在实例创建后，根据实际访问需求添加或修改访问规则 CCE默认/附加安全组说明。

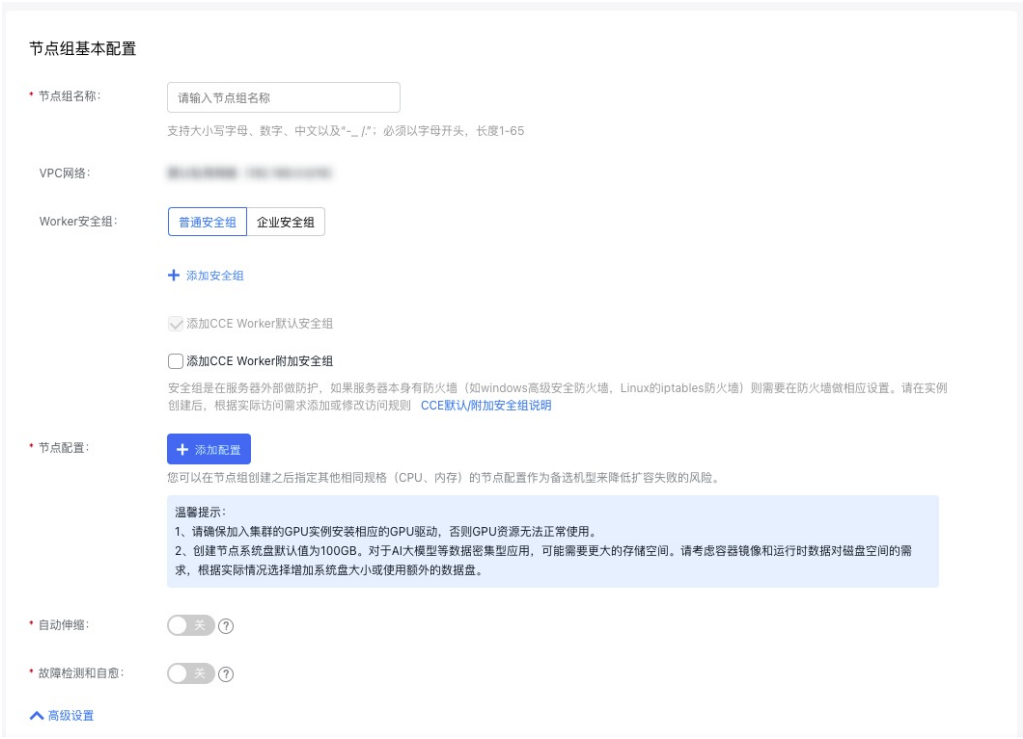
- **节点配置**：后续扩容节点组将以此配置作为创建节点的模版。

配置项	说明
节点类型	按需选择，目前支持云服务器 BCC、弹性裸金属服务器 EBC、裸金属服务器 BBC。
付费方式	按需选择，目前支持后付费、预付费、抢占实例。 • 包年包月：需要提前一次性支付所选时间段内的费用，这种模式价格相较于按量付费模式更加低廉。 • 按量付费：按实例的实际使用量收费，不需要提前支付费用，但需要保证账户余额大于或等于100元，这种模式比预付费模式价格高一些。 • 抢占实例：一种新运作模式，您在指定范围内出价，当指定实例规格的市场价格低于您的出价，且资源库存充足时，会成功创建抢占实例，并按照当前市场价计费。
可用区	可用区是指在同一地域下，电力和网络互相独立的物理区域，故障会被隔离在一个可用区内。用于筛选可用区下可用的子网。
节点子网	选择为节点分配 IP 地址的子网。不同可用区可用子网不同。
实例配置	根据CPU与内存比值的不同，云服务器提供了多种不同类型的实例规格族。关于实例规格及适用的应用场景您可参考实例规格。
镜像类型及操作系统	您可根据实际需求，选择适合的镜像类型及操作系统。 • 公共镜像：由百度智能云官方提供，仅包含基础操作系统环境。 • 自定义镜像： 通过创建自定义镜像功能生成的镜像，包含系统盘基础操作系统、应用及软件环境的个性化配置。自定义镜像可以帮助您快速创建包含个性化配置的云服务器。 • 共享镜像： 支持用户将自定义镜像共享给其他用户，被共享用户可以通过管理控制台或API查询到其他用户共享的镜像。被共享用

	户可以使用其他用户共享的镜像创建实例和重装操作系统。
系统盘	用于安装操作系统。非异构计算实例且镜像os是linux，默认大小20GB；若镜像os是windows，默认大小40GB；异构计算实例不区分操作系统默认均为40GB。根据地域以及实例规格的不同，可选择的云盘类型也不同，以页面实际提供的云盘类型为准。
数据盘	即挂载的数据盘，用于提高云服务器的存储容量，默认不选择。云磁盘挂载数量有一定上限，若您想挂载超出上限的云磁盘，请提交工单联系我们。当前，云服务器的系统盘和数据盘均为CDS云磁盘，关于CDS的磁盘类型和使用限制请查看磁盘类型和使用限制。
绑定快照策略	默认关闭绑定快照策略。通过快照，您可以实现磁盘数据备份，磁盘数据恢复以及磁盘镜像的制作。更多关于快照的使用方法和使用限制，请查看快照使用说明。快照目前是收费服务，请查看快照计费说明。
公网IP	如需公网访问，请购买弹性公网IP，或在实例购买成功后绑定已有的弹性公网IP。公网带宽支持以下购买方式： • 包年包月计费：需要提前一次性支付所选时间段内的带宽费用，购买包年包月的云服务器时，费用将合并在实例中收取。 • 按使用流量计费：根据用户实际传输的数据量计费，流量使用没有上限，但可限定最大峰值带宽。 • 按使用带宽计费：根据用户选择的固定带宽值进行计费，最大可购买200Mbps的带宽。
实例名称	您可自定义设置实例名称或由系统随机生成实例名称。
Domain开关	用户打开后，主机名将带有domain后缀，支持DNS解析。
管理用户名	Windows系统的管理员账号为Administrator，Linux系统的管理员账号为root。
管理员密码	根据实例操作系统的不同，密码可供选择的设置方式也不同。 • 用户自定义：自定义设置登录实例的密码。 • 随机生成：购买成功后需登录控制台重置密码，可参考 重置密码。 • 密钥对：Linux操作系统可以选择使用密钥对的方式连接云服务器，SSH密钥对是一种比常规密码更安全的登录云服务器的方式，具体可参考密钥对设置。
数量	填写的节点数量即初始设置的期望节点数。数量限制如下： • 单个节点组容量上限为1000。 • 每次操作最多200个节点。 • 扩容上限受节点组中各子网剩余IP数量总和以及机型库存限制。
部署集	指定部署集中创建云服务器实例时，会和处于同一部署集中的其他云服务器实例，按照物理服务器打散，保障发生硬件故障等异常情况下时的服务高可用性。具体可参考部署集进行设置。单个实例最多可加入2个部署集。
自动伸缩	启用自动伸缩，系统将根据节点配置和自动伸缩配置，在符合扩容条件时，自动进行扩容操作。并自动计算节点费用、生成订单。扩容操作完成后，可自行查看节点和订单信息。
故障检测和自愈	支持节点故障检测，可按需配置自愈规则。

- **高级配置**

配置项	说明
扩缩容策略	• 机型配置顺序：节点组会根据您配置的主备机型顺序进行扩缩容。若主机型无法扩容，则选择备选机型尝试扩容。 • 多子网均匀分布：在伸缩组指定的多可用区（即指定多个子网）之间尽最大努力均匀分配节点实例。只有配置了多个子网时该策略才能生效。
节点显存共享	默认不勾选。勾选该选项后，新添加的节点默认开启 GPU 共享能力。显存共享仅对带有 GPU 设备的节点生效，未带有 GPU 设备的节点将会被忽略。 详情请参考 GPU独占和共享 。 • **注意**：开启节点显存共享前提是需要安装GPU Manager 和 AI Job Scheduler 组件。
节点缩容保护	默认不开启。开启后，节点不受节点组自动缩容或调整期望节点数影响。 • 开启后，节点组中新增节点都默认开启缩容保护。 • 节点开启缩容保护，手动移出节点不受影响。
kubelet数据目录	volume文件、plugin文件等数据存储目录，例如 /var/lib/kubelet。 • 如果已挂载数据盘，建议存储到数据盘
容器数据目录	容器、镜像等数据存储目录。勾选即可设置容器和镜像存储目录，例如 /home/cce/containerd。 • 如果已挂载数据盘，建议存储到数据盘
部署前执行脚本	节点部署前将自动执行该脚本，您需要自行保证脚本的可重入及重试逻辑。脚本内容和产生日志将写入节点的/usr/local/cce/scripts/目录下。
部署后执行脚本	节点部署后将自动执行该脚本，您需要自行确定脚本的执行情况。脚本内容和产生日志将写入节点的/usr/local/cce/scripts/目录下。
自定义kubelet参数	支持自定义配置 Kubelet参数，详情请参见 自定义 kubelet 参数介绍。
封锁节点	默认关闭。选择开启封锁节点后，节点将处于不可调度状态，新创建的Pod将不会调度到该节点上。若需要取消封锁，请执行kubectl uncordon命令。封锁节点将减少该节点所属集群的剩余可用资源配额，若集群本身预留的可用资源不足，将会影响后续正常业务的调度和存量业务的运行。
资源标签	资源标签支持您按各种标准（如用途、所有者或项目）对云上资源进行分类，每个标签包含键和值两部分，具体可参考标签功能进行设置。 • 默认为实例关联资源 CDS、EIP 统一添加标签，可以选择关闭。
IAM 角色	为云服务器实例设置IAM角色。更多信息详见 设置IAM角色。
标签（Labels）	K8S标签（Label）是管理和选择K8S对象的标识， 将自动绑定到节点组创建的节点；每个Label包含键和值两部分。更多信息详见K8S标签说明。
污点（Taints）	节点污点和Pod容忍共同作用。节点设置污点后，可避免Pod调度到节点上，或者将Pod从节点驱逐。除非Pod容忍能够节点污点相匹配。更多信息详见污点和容忍说明。
注解（Annotations）	Annotations（注解）是一种将非标识性元数据附加到对象上的机制，每个Annotation包含键和值两部分。。更多信息详见注解说明。



5. 点击“完成”按钮，完成节点组的创建。

查看节点组

1. 创建完成后，返回节点组列表，即可查看节点组。
2. 节点组列表中展示可查看以下信息：

列名	说明
节点组名称/ID	节点组 ID 作为节点组的唯一标识，可用于在集群的节点列表中，搜索节点组中的具体节点。
付费方式	默认为后付费。
实例配置	创建节点组时所选节点配置中的实例规格、类型信息。
实际节点数	实际 Ready 节点数，节点的具体状态和扩缩容进度可在节点列表中查看。
期望节点数	创建节点组时设置的节点数量，即期望节点组中保持可用的节点数量。
自动伸缩范围	若启用自动弹性伸缩可查看自动伸缩范围，将在该伸缩范围内自动调节期望节点数。
故障检测自愈	若启用故障检测自愈，可查看故障自愈的规则。
创建时间	节点组的创建时间。

配置自动伸缩

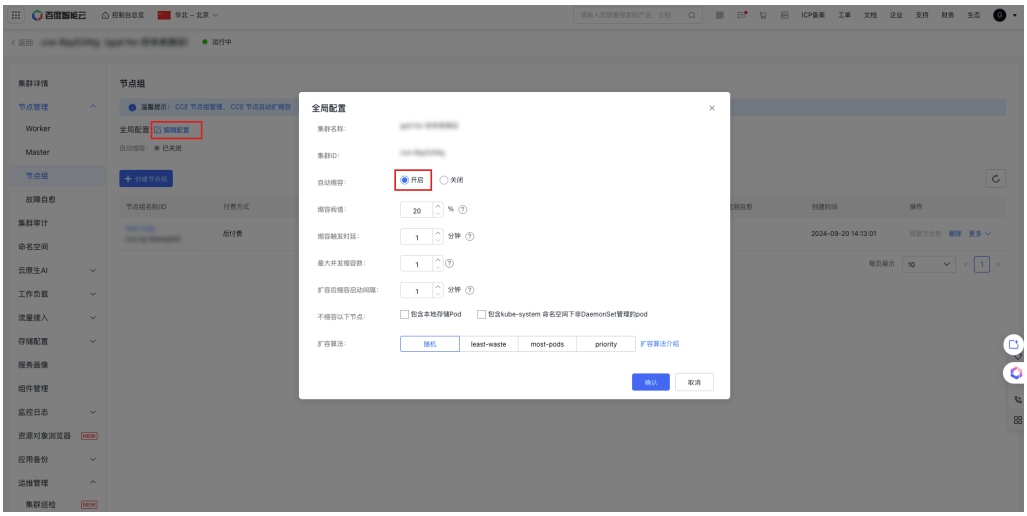
步骤一：开通自动伸缩

初次使用节点组功能，首次配置自动伸缩，需要先**授权开通自动伸缩**，才可以启用自动伸缩功能。可在节点组列表的**全局配置**模块中点击**授权开通自动伸缩**开通，也可以在**首次创建节点组**时开通。

步骤二：全局配置

授权开通自动伸缩后，可在节点组列表的**全局配置**模块中点击**编辑配置**，在全局配置弹窗中开启自动缩容并配置扩容算法，该配置将应用于集群中所有启用自动伸缩功能的节点组。相关配置项及说明如下：

配置项	可选项	说明
自动缩容	缩容阀值	伸缩组内节点资源（cpu、gpu、mem）利用率均低于设定阈值时，集群可能会触发自动缩容。默认输入范围：20 - 80。
	缩容触发时延	在配置的缩容触发时延内，节点资源利用率持续低于缩容阈值，集群可能会触发自动缩容。默认输入范围：1 - 60。
	最大并发缩容数	当节点的利用率为0时，并发缩容节点的数目。默认输入值范围：1 - 20。
	扩容后缩容启动间隔	扩容出来的节点经过此间隔后开始评估是否可以被缩容。默认输入值范围：1 - 60。
	不缩容以下节点	包含本地存储 Pod 包含 kube-system 命名空间下非 DaemonSet 管理的 pod。
扩容算法	随机	详情参见 扩容算法介绍。
	least-waste	
	most-pods	
	priority	



步骤三：自动伸缩配置

授权开通自动伸缩后，可在创建节点组时，或在节点组列表页的操作中，对节点组启用或停用自动伸缩功能，并配置相应的自动伸缩策略，以下介绍如何为已有节点组配置自动伸缩。

1. 在节点组列表页中，找到目标节点组，点击操作列中的**更多 > 自动伸缩配置**。

2. 在自动伸缩配置弹窗中启用自动伸缩，并配置伸缩范围和扩容优先级。

配置项	说明
伸缩范围	启用自动弹性伸缩后，将在该伸缩范围内自动调节期望节点数。您可输入期望的最小节点数和最大节点数。
扩容优先级	自动扩容时，将以扩容优先级顺序，对已启用自动伸缩的节点组进行扩容。数值越大，优先级越高。

调整节点数

调整节点数即通过手动调整节点组的期望节点数，来实现节点组扩容或缩容的目的。

1. 在节点组列表页中，找到需要调整节点数的目标节点组，点击操作列中的**调整节点数**。

2. 在调整节点组弹窗中输入期望节点数，点击**确定**即可调整节点组的期望节点数。

> **说明：**
>
> - 若已启用自动伸缩，在符合扩缩容条件时，不支持手动修改期望节点数，CCE 将在伸缩范围内为节点组自动调整期望节点数。
> - 若不希望自动调整期望节点数，请先在自动伸缩配置中关闭自动伸缩，再手动调整节点数。
> - 期望的节点数不是新增节点数量。例如：扩容前节点池中已存在 3 个节点。当您配置期望节点数为 5 时，系统会自动扩容 2 个节点，满足 5 个节点的期望，而不是新增 5 个节点。

- 单个节点组容量上限为1000。

- 每次操作最多500个节点。
- 扩容上限受节点组中各子网剩余IP数量总和以及机型库存限制。
- 节点组开启缩容保护后，期望节点数不能小于节点组开启缩容保护的节点。
- 调整期望节点数不会缩容开启缩容保护的节点。

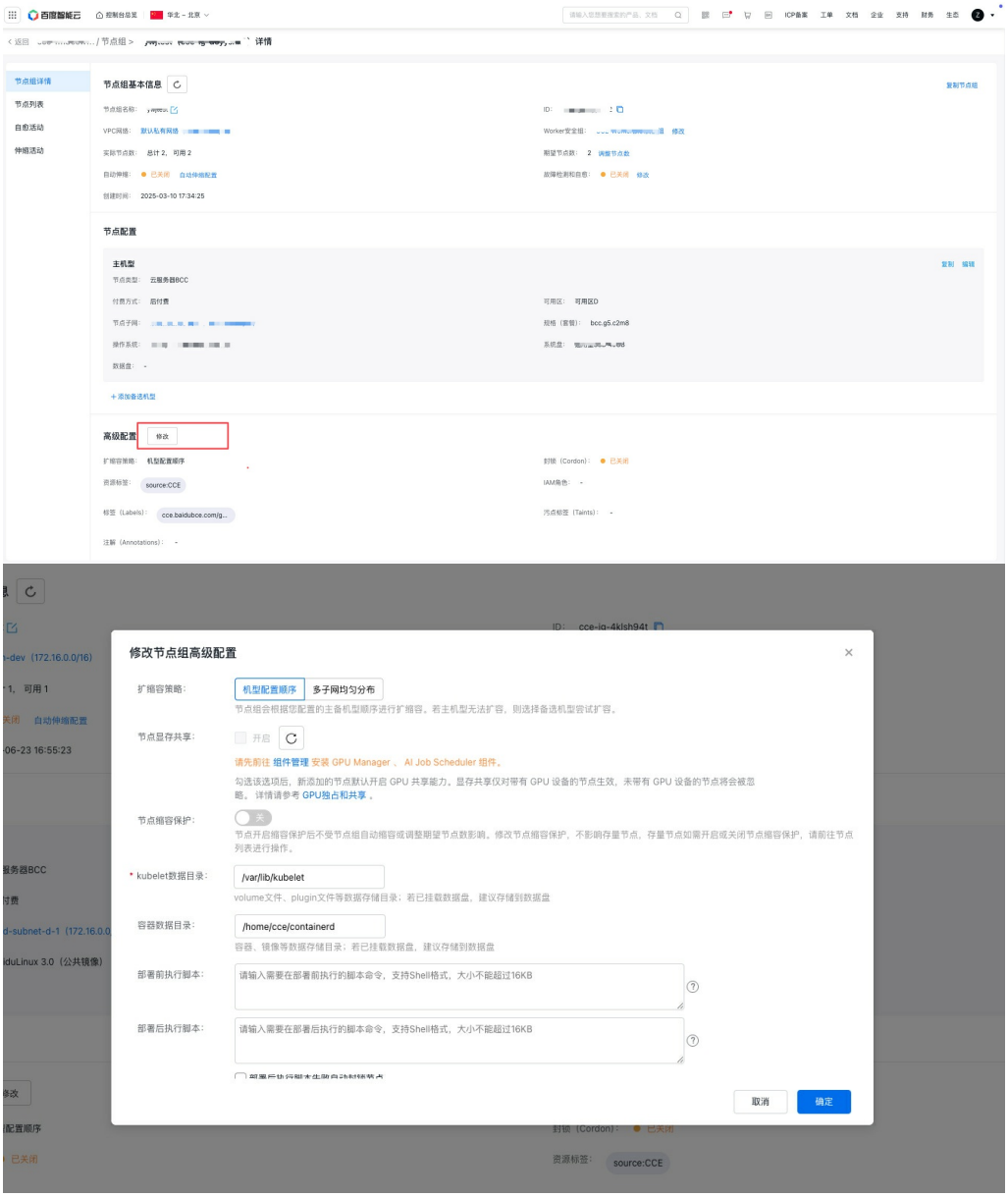
编辑节点组高级配置

创建节点组后，CCE集群支持通过控制台调整部分节点组配置。

说明：

- 节点组编辑高级配置过程中，不会影响节点组内已有节点和业务的运行。
- 节点组配置更新后，除特殊说明场景外（例如同步更新存量节点的标签（Labels）、污点（Taints）及注解（Annotations）），不会修改节点组已有节点的配置，仅作用于新增节点。
- 节点组配置更新后，后续节点组中新增的节点都将默认使用该配置。
- 更新节点组配置需参见本步骤来实现。如您通过其他方式对节点进行了更改，节点组升级时这些更改会被覆盖。
- 勾选同步更新存量节点的标签（Labels）、污点（Taints）及注解（Annotations）功能，在节点组中新增或修改标签、污点、注解，会自动更新到新建节点和存量节点。同时，修改存量节点上的标签和污点后，节点上的配置会被节点组的配置刷新。
- 关闭同步更新存量节点的标签（Labels）、污点（Taints）及注解（Annotations）功能，在节点组中新增或修改标签、污点、注解，仅会同步到新建节点，不会更新到存量节点。同时，修改存量节点上的标签和污点后，存量节点上的配置以修改后的配置为准，不会被节点组的配置刷新。
- 修改节点缩容保护，不影响存量节点，仅针对新增节点开启/关闭节点缩容保护。存量节点如需开启或关闭节点缩容保护，请前往节点列表进行操作。

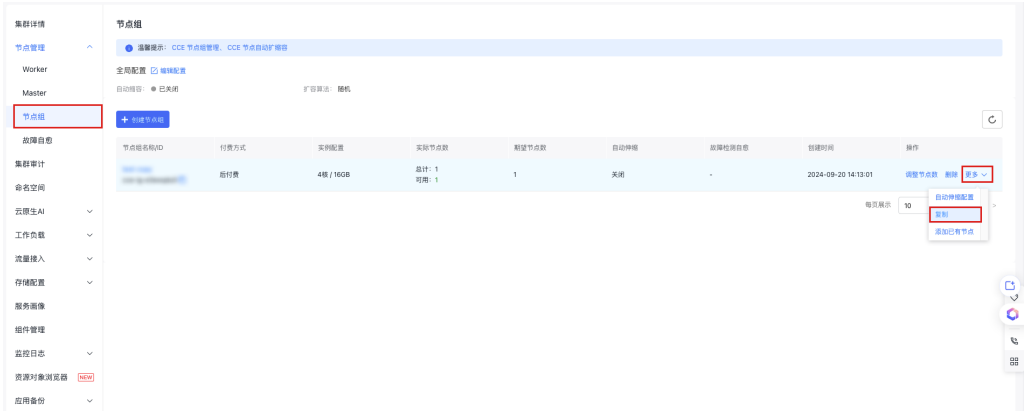
1. 在节点组列表页中，找到需要移除节点的目标节点组，点击节点组名称/ID进入节点组详情页。
2. 在节点组详情页高级配置点击**修改**，修改节点组高级配置页面，编辑节点组的配置项，并按照页面提示完成配置。



复制节点组

通过 CCE 控制台可以轻松地复制现有节点组的配置，并基于这些配置创建新的节点组。

1. 在节点组列表页中，找到需要复制的目标节点组，点击操作列中的**更多 > 复制**。



2. 在复制节点组页面中，可以看到复制的节点组配置，您可以根据需要进行修改。确定配置后单击**完成**即可进行节点组复制。

删除节点组

删除节点组，可根据实际需要选择是否保留/移出/释放节点，以及是否释放实例绑定的后付费公网 IP 和云磁盘。

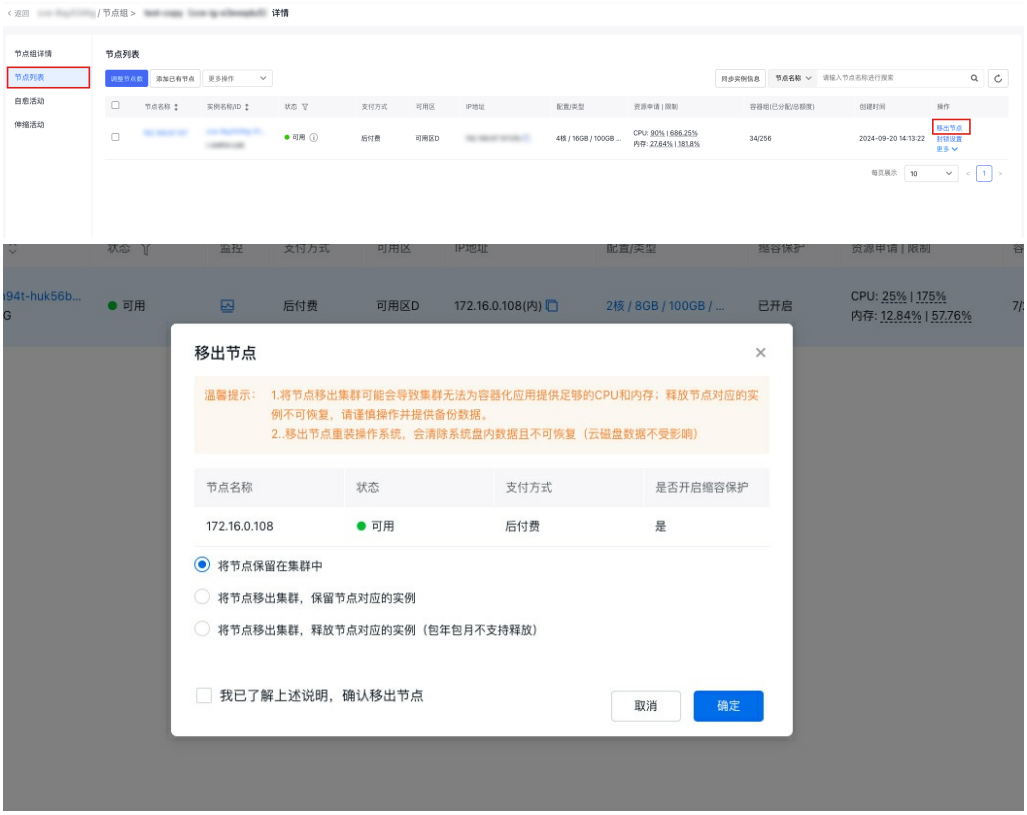
> **注意：** 节点组删除后无法恢复，请做好数据备份，谨慎操作。

1. 在节点组列表页中，找到需要删除的目标节点组，点击操作列中的**删除**。
2. 在删除节点组弹窗中选择是否保留/移出/释放该节点组中的节点，并选择是否释放实例绑定的后付费公网 IP 和云磁盘。在删除节点组时您可选择进行如下操作：
    - 将该节点组中的节点继续保留在集群中
    - 将该节点组中的节点移出集群，但保留虚机资源
    - 将该节点组中的节点移出集群，并释放虚机资源（预付费不会主动释放）
    - 释放实例绑定的后付费公网 IP 和云磁盘

移除节点

> **注意：**
>
> - 在使用节点组功能前已存在于集群中的节点，以及通过集群扩容、添加节点（非节点组方式）新建的节点，不归属于节点组，在节点管理 > 节点列表中单独管理。
> - 在集群的**节点组管理 > 节点列表**中移出节点，即以节点为操作对象，将完全按照当前节点的操作逻辑，不会影响节点组的期望节点数，节点组将根据当前期望节点数自动进行节点数量的调整。
> - 在**节点管理 > 节点组 > 节点列表**中移除节点会减少节点组对应的期望节点数量。

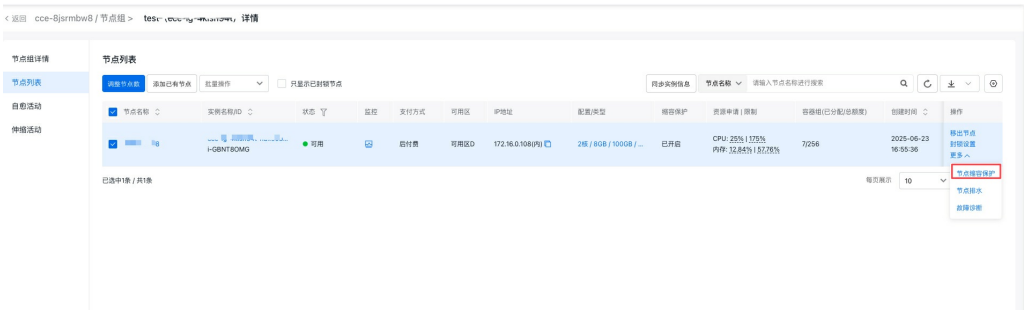
1. 在节点组列表页中，找到需要移除节点的目标节点组，点击**节点组名称/ID**进入节点组详情页。
2. 在左侧导航栏选择**节点列表**，查看当前节点组中的全部节点。
3. 找到需要移出节点组的节点，在操作列选择**移出节点**。若您需要同时移出多个节点，可在当前页面中勾选多个节点后，点击上侧**更多操作 > 移出节点**对多个节点进行移出操作。
4. 在移出节点弹窗中选择是否需要将该节点保留/移出集群，以及是否需要保留/释放节点对应的实例。
    - 将该节点组中的节点继续保留在集群中
    - 将该节点组中的节点移出集群，但保留虚机资源
    - 将该节点组中的节点移出集群，并释放虚机资源（预付费不会主动释放）
    - 释放实例绑定的后付费公网 IP 和云磁盘
5. 点击**确定**，即可将节点移出节点组。

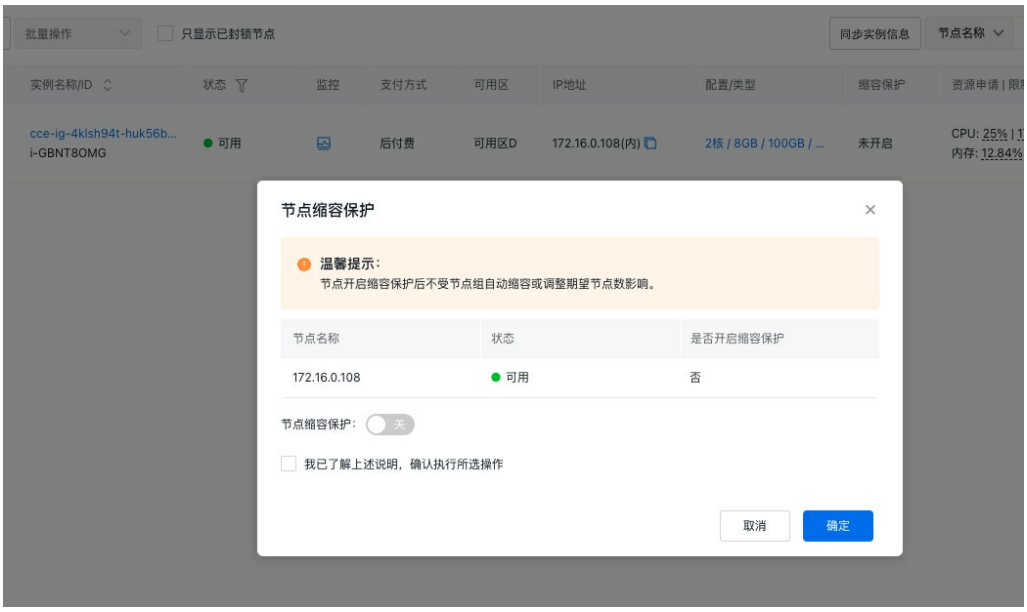


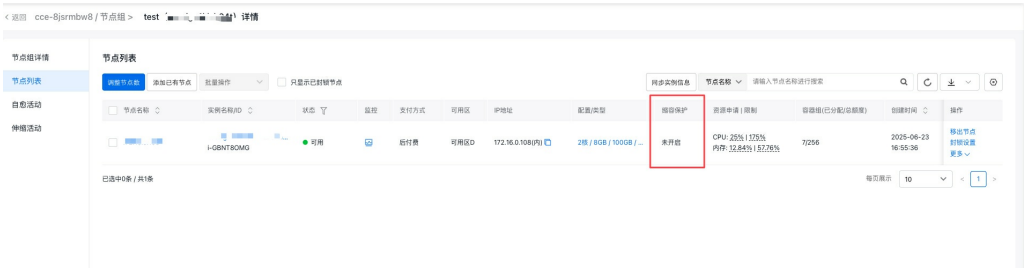
设置缩容保护

> **注意：**
>
> - 节点开启缩容保护后不受节点组自动缩容或调整期望节点数影响。
> - 手动将节点移出节点组/集群时，不受节点缩容保护影响。

1. 在节点组列表页中，找到需要移除节点的目标节点组，点击**节点组名称/ID**进入节点组详情页。
2. 在左侧导航栏选择**节点列表**，查看当前节点组中的全部节点。
3. 找到需要开启缩容保护的节点，在操作列点击更多，选择**节点缩容保护**。若您需要同时针对多个节点开启缩容保护，可在当前页面中勾选多个节点后，点击上侧**更多操作 > 节点缩容保护**对多个节点进行节点缩容保护配置。
4. 在节点缩容弹窗中选择是否需要开启节点缩容保护。
5. 开启成功之后，您可以在节点列表中查看节点缩容保护状态。







添加备选机型

容器引擎 CCE 节点组针对服务器 BCC 和 EBC 支持主备机型，即在主机型的基础之上支持配置多个备选机型，可指定不同可用区下的不同机型（规格相同），降低因资源售罄导致扩容失败的风险。

创建节点组时指定的节点配置默认为主机型，节点组创建成功之后您可以在主机型基础之上添加多个备选机型。本文将以 BCC 为例介绍如何配置备选机型。

前提条件

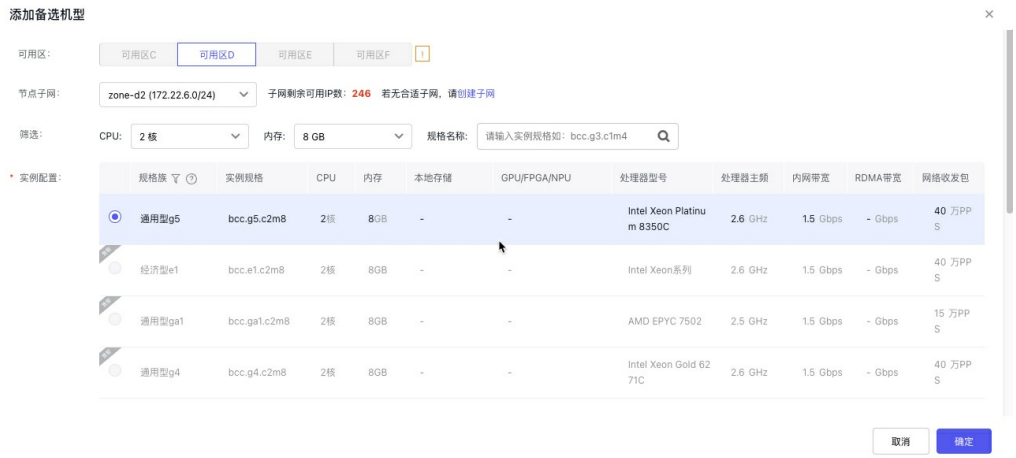
- 已成功 创建 CCE 集群。
- 已成功 创建节点组。

使用限制

- 备选机型必须与主机型规格（CPU、内存、架构、GPU卡数）相同。
- 同一节点组最多可以添加 10 个备选机型，请按需自行规划。
- 若修改主机型 CPU、内存、GPU 卡数需要重新添加备选机型，已有节点不受影响。
- 备选机型不支持选择售罄的实例规格。

操作步骤

1. 登录 容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理 > 节点组**。
5. 在“节点组”页面单击目标节点组进入“节点组详情”页面。
6. 在“节点配置”中单击**添加备选机型**。
7. 在“添加备选机型”弹出框中完成节点子网、实例规格等配置，如下图所示：



8. 单击**确定**按钮，即可完成添加备选机型。您可以在“节点组详情”页面查看配置的备选机型，如下图所示：

> **说明**
>
> - 编辑备选机型时，若备选机型实例规格配置已经售罄，需要重新选择与主机型实例规格（CPU、内存、架构、GPU卡数）相同的实例规格。

配置扩缩容策略

容器引擎 CCE 节点组支持配置多个备机型，同时结合扩缩容策略可满足不同场景下对资源的使用诉求。

扩缩容策略介绍

不同扩缩容策略对应的扩缩容方式不同，具体信息请见下方说明：

机型配置顺序

扩容原理：按照主备机型顺序依次扩容，优先选择主机型扩容，若主机型资源不足则依次尝试选择备选机型（按添加顺序排序）进行扩容

> 说明：例如，您配置了A主机型，B、C备选机型，系统会按照A、B、C依次尝试，若A售罄，会尝试B，若B也售罄才会尝试C。

缩容原理：按照扩容时指定的优先级（节点扩容顺序）倒序进行缩容。

多子网均匀分布

扩容原理：扩容时按照各个子网中的机型顺序依次尝试扩容，尽可能保证各个子网中的节点数量相同

缩容原理：缩容时先计算子网数量，按照缩容后子网下节点平均策略进行缩容，尽可能保证各个子网中的节点数量相同

前提条件

- 已成功 创建 CCE 集群。

操作步骤

创建节点组时指定扩缩容策略

1. 登录容器引擎控制台。

2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理 > 节点组**。
5. 在“节点组”页面单击**创建节点组**进入创建页面。
6. 在“高级设置”中配置**扩缩容策略**。
7. 单击“完成”按钮，完成创建节点组。

策略	说明
机型配置顺序	节点组会根据您配置的主备机型顺序进行扩缩容。若主机型无法扩容，则选择备选机型尝试扩容
多子网均匀分布	节点组会在您配置的多个子网之间尽可能均匀分配节点。只有配置了多个子网时该策略才会生效

> **注意：**
>
> - 仅配置了多个备选机型该策略才会生效（多个子网最好分布在不同的可用区中，提升集群跨可用区高可用性），详情请参考配置备选机型。
> - 节点开启缩容保护后不受节点组自动缩容的影响。

更新节点组扩缩容策略

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理 > 节点组**。
5. 在“节点组”页面单击目标节点组进入“节点组详情”页面。
6. 在“高级配置”中单击**修改**。
7. 在“修改节点组高级配置”弹出框中修改扩缩容策略配置。
8. 单击“确定”按钮，完成更新扩缩容策略。

> **注意：**
>
> - 修改节点组高级配置，仅对节点组新扩容节点生效，不会影响节点组内已有节点和业务的运行。

添加外部已有节点

如需将购买的实例添加到 CCE 集群中作为Worker节点，或将移除的 Worker 节点重新加入节点组，您可以在容器引擎 CCE 控制台手动添加节点。添加后，在节点组维度进行统一管理节点。添加前，请仔细阅读使用限制、注意事项等。

前提条件

- 已成功 创建 CCE 集群。
- 已成功 创建节点组。

使用限制

- 仅支持添加同一账号下的服务器 BCC、BBC、EBC 实例，且添加的 BCC、BBC、EBC 实例必须与集群在同一地域、同一 VPC 下。
- 不支持添加已属于其他 CCE 集群和节点组的服务器实例。
- 仅支持与当前节点组实例规格（CPU、内存、GPU卡数）相同的服务器，且付费方式需要与当前节点组一致。
- 仅支持添加与当前节点组同类型的服务器。
- 仅支持未开启自动伸缩的节点组添加已有节点，若节点组已开启自动伸缩需要先关闭才能添加已有节点，详情参见 节点组管理-配置自动伸缩。
- 需确保您的集群节点配额充足，关于 CCE 集群的配额限制，请参考 CCE 集群配额限制。

操作步骤

添加集群外已有的服务器实例

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理 > 节点组**。
5. 在目标节点组操作列选择**更多 > 添加已有节点**进入扩容节点页面；也可以点击目标节点组，进入节点列表页选择单击**添加已有节点**进入扩容节点页面。
6. 单击**完成**按钮，即完成添加已有节点。您可以在“节点组-节点列表”页面中查看添加的节点信息。

配置项	必选/可选	配置说明

节点来源	必选	当前示例选择“集群外”。
节点类型	必选	默认选择节点组配置的服务器类型，不支持修改。
节点选择	必选	选择要加入节点组的服务器实例，支持多选。 • 注意：支持添加的节点数量受集群规模限制。
重装操作系统	必选	重装操作系统，将对所选的实例进行操作系统重装，会清除系统盘内数据且不可恢复（云磁盘数据不受影响）。 • **注意**：添加节点不重装操作系统，服务器实例上的旧数据可能会导致节点添加失败或出现其它问题，推荐选择重装操作系统。
已选节点	必选	支持为已选择的节点配置重装操作系统或使用已有配置直接加入集群，当前示例选择“重装操作系统”。 重装操作系统，会清除系统盘内数据且不可恢复（云磁盘数据不受影响）。

磁盘挂载	可选	支持选择沿用节点组配置和不沿用节点组配置。默认不沿用节点组配置。
		• **不沿用节点组配置**：需要选择是否将容器和镜像数据存储在数据盘或本地盘中。
		  • 勾选：需要填写设备名称、挂载目录。需要注意以下几点：
		  • 系统将会对所选的全部节点设备格式化为ext4，已经格式化的设备将直接挂目录。被格式化的数据盘内原有数据将会丢失，请提前备份重要数据。
		  • 您填写的挂载设置会对本批次添加节点全部生效，请确认填写的设备名称符合预期。
		  • 若输入的设备名称与所选节点的实际名称不匹配，系统会对不匹配的节点报错并终止节点初始化流程。若填写的挂载目录不存在，系统会自动创建对应目录。
		  • 若所选节点未挂载磁盘，则不会自动挂载新的磁盘。
		  • 设备名称查询方式请参考查看磁盘信息。
		  • 不勾选：不设置磁盘挂载选项，可手动或者使用脚本挂载。
		• **沿用节点组配置**：无需手动填写磁盘挂载配置，系统会自动将节点组的磁盘挂载配置到已选节点。需要注意以下几点：
		  • 已选节点需要与节点组的实例规格、磁盘数量、磁盘类型、磁盘大小都匹配时，才支持沿用节点组的配置。如果已选节点和节点组不匹配时，已选节点保持原有磁盘配置，不会沿用节点组的配置。
		  • 当已选节点与节点组机型匹配时，磁盘将被系统格式化为ext4文件系统，已格式化的磁盘将直接进行挂载，系统从最后一块磁盘倒序进行挂载。格式化将会导致原有数据丢失，请注意备份。
		  • BBC类型的已有节点添加至节点组，暂不支持沿用节点组磁盘挂载配置。
		  • 当已选节点均与节点组不匹配时，则不支持沿用节点组配置。

操作系统配置	必选	当开启重装操作系统时，支持选择沿用节点组配置和不沿用节点组配置，默认不沿用节点组配置。 若关闭重装操作系统，则保留已选实例的操作系统配置。
		• **沿用节点组配置**：无需手动选择操作系统、驱动，系统会自动按照节点组的主机型操作系统、驱动版本重装已选节点。
		  • **注意:** 沿用节点组配置时，系统会清空已选实例自定义驱动（GPU Driver/CUDA/cuDNN等）或“用户数据注入”配置。
		• **不沿用节点组配置**：需要选择操作系统类型以及镜像版本。
		  • **注意：**如果是GPU节点，按需选择是否安装GPU驱动、CUDA、cuDNN。选择开启自定义安装驱动后，将由系统按照你选择的版本自动为您安装GPU驱动、CUDA、cuDNN。否则，保留实例原有的自定义驱动（GPU Driver/CUDA/cuDNN等）或“用户数据注入”配置。

管理员用户名	必选	登录服务器实例的用户名，Linux 操作系统默认为 root。
登录方式	必选	支持“密钥对”和“自定义密码”。
沿用节点组配	必选	默认勾选。添加节点的其他配置项，如Worker安全组、节点显存共享、节点缩容保护、kubelet数据目录、容器数据目录、部署

点组配置	必选	执行脚本、自定义kubelet参数、封锁节点、资源标签、IAM角色是否使用节点组当前的配置信息。若勾选，则添加节点时使用节点组的配置，若不勾选，您也可以自定义相关配置初始化节点。

注意事项：

- 安全组分为普通安全组和企业安全组，不同类型安全组不能同时绑定，更多信息请见 安全组概述。
- 若安全组类型一样，则会在已有安全组的基础之上加入节点组安全组配置，否则会自动解绑服务器原有安全组，使用节点组的安全组配置。
- K8s 标签、污点、Annotation 默认使用节点组当前的配置。
- 沿用节点组配置时会将云服务器实例绑定IAM角色变更为节点组当前配置，若节点组的IAM角色配置为空，不会对云服务器已绑定的IAM角色进行解绑。

添加集群内游离的节点

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理 > 节点组**。
5. 在目标节点组操作选择**更多 > 添加已有节点**进入扩容节点页面；也可以点击目标节点组，进入节点列表页选择单击“添加已有节点”进入扩容节点页面。
6. 单击**完成**按钮，即完成添加已有节点。您可以在“节点组-节点列表”页面中查看添加的节点信息。

配置项	必选/可选	配置说明
节点来源	必选	当前示例选择“集群内”。
节点类型	必选	默认选择节点组配置的服务器类型，不支持修改。
节点选择	必选	选择要加入节点组的集群中游离的节点（不属于任何节点组），支持多选。 • 注意：支持添加的节点数量受集群规模限制。

说明：

- 移入节点组后，节点原有的标签（Labels）、污点（Taints）、注解（Annotations）、节点显存共享配置信息、节点缩容保护、IAM角色配置会被更新为节点组当前的配置，若节点组的IAM角色配置为空，不会对云服务器已绑定的IAM角色进行解绑。其他配置保持不变。
- 添加已有实例到节点组中，会保留实例原有的自定义驱动（GPU Driver/CUDA/cuDNN等）或“用户数据注入”配置。

节点组节点故障检测自愈

节点作为集群的基础设施，其运行状态对业务来说至关重要。基础设施的不稳定性、环境的不确定性经常会引发不同维度的系统故障。为进一步降低用户运维成本，容器引擎CCE提供故障检测自愈功能，面向节点进行常见的故障检测和故障自愈。本文将为您介绍如何配置节点检查自愈功能。

功能概述

容器引擎CCE节点故障检测功能支持对接BCC维修平台实现节点服务器故障的自动检测和自愈，更多信息请参见维修平台概述。

检测项介绍

分类	检测项	描述	推荐自愈动作
节点服务器硬件故障	硬盘硬件故障	服务器硬盘发生硬件故障，需要维修恢复	节点封锁、节点排水、节点移出、维修授权
	其他硬件故障	服务器CPU、GPU等其他硬件发生故障， 需要维修恢复	节点封锁、节点排水、维修授权、恢复检测

说明

- 节点服务器硬件故障：影响业务正常运行的硬件故障，需要更换硬件部件维修。

自愈操作介绍

操作项	说明
节点封锁	节点将封锁，Pod无法调度上去
节点排水	节点上运行的 Pod将会被驱除
节点移出	节点将被移除集群。支持同步删除服务器实例或保留服务器实例
维修授权	服务器将授权维修，支持自动授权和人工授权，维修后会产生维修记录，可在维修平台查看维修进度
恢复检测	检测到故障修复成功后自动解除节点封锁

操作步骤

前提条件

- 已在集群组件管理模块安装CCE Node Problem Detector组件和CCE Node Remedier组件。

步骤一：创建检查自愈规则

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**节点管理 > 故障自愈**。
3. 单击**创建自愈规则**进入“创建自愈规则”页面完成自愈配置。

检测项	说明
规则名称	配置检查自愈规则的名称。
规则配置	在预置检查项中选择需要检查的项目，并配置异常时是否开启自愈。
自愈配置	自定义配置该检查项的自愈动作。

4. 配置完成后，勾选**我已知晓并安装了HAS-agent**，单击**确定**，完成配置。

说明

- 节点服务器硬件故障检测需要在服务器上安装HAS-agent，需确保已成功安装，否则无法检测，更多信息请见HAS-agent组件安装与升级。

步骤二：节点组绑定自愈规则

您可以在创建节点组时绑定检查自愈规则，也可以为存量节点组绑定检查自愈规则。当前以创建节点组时绑定检查自愈规则为例。

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**节点管理 > 节点组**。
3. 单击**创建节点组**，创建新的节点组，并在**故障检测和自愈**处绑定检查自愈规则。

检测项	说明
故障检测和自愈	配置是否开启节点检查自愈功能。
故障自愈规则	选择节点组绑定的自愈规则。

4. 单击**确定**，完成绑定。

后续操作

查看自愈任务详情

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**节点管理 > 节点组**。
3. 在节点组列表页面单击目标节点组进入管理页面。
4. 选择**自愈活动**页签，即可查看当前节点组中所有节点检测到故障而触发自愈操作的任务记录及详情。

升级节点组

节点组提供升级功能，支持对节点的Kubelet和容器运行时版本进行升级，针对GPU节点支持升级nvidia-container-toolkit版本。执行升级操作前，CCE集群会进行前置检查，节点组需要通过检查后才能进行升级。

注意事项

- 节点组升级过程中仅支持扩容操作，不支持缩容操作，升级过程中不会触发缩容节点。
- 节点组升级默认为原地升级，部分升级场景在升级节点前会先进行节点排水操作，如容器运行时由Docker升级为Containerd。
- 如果您的集群中存在游离节点，即未被节点组管理的Worker节点，可以参考添加外部已有节点将游离节点加入节点组中进行升级。
- 单账号最多支持10个节点组同时升级。

升级说明

节点组升级包含包含Kubelet、容器运行时和nvidia-container-toolkit版本升级。

- Kubelet升级：支持将节点组中的全部节点的Kubelet版本升级到与当前集群控制面相同的版本。
- 容器运行时升级：支持将节点组中的全部节点的容器运行时版本升级至集群发布支持的最新版本，同时支持从 Docker升级为Containerd。
- nvidia-container-toolkit升级：支持将节点组中的全部节点的nvidia-container-toolkit版本升级至所支持的最新版本。

操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理** > **节点组**。
5. 在目标节点组操作列选择**更多** > **升级**进入节点组升级页面并完成升级配置。

配置项	**必选/可选**	**配置说明**
升级对象	必选	选择节点需要升级的对象。
升级范围	必选	选择升级的节点范围。**全部节点**：默认选择节点组中全部符合升级条件的节点进行升级；**指定节点**：支持选择节点组中的一个或多个节点进行升级，升级后可再次选择升级。
节点排水	可选	选择节点升级前是否进行排水操作，驱逐节点上的Pod。
每批次执行最多节点数	必选	节点组中的节点升级分批次进行，设置每批次最大并行升级的节点数，默认为1，最大可设置10。
自动暂停策略	必选	设置升级过程中的暂停策略。**首批次暂停**：仅第一批次升级完成后升级任务将会自动暂停，需要手动触发继续升级；**每批次暂停**：每完成一批次升级完成后升级任务都会自动暂停，需要手动触发继续升级；**不暂停**：升级任务不会暂停，一批次升级完成之后继续升级下一批次。
每批次间隔时间	必选	当**自动暂停策略**选择**不暂停**时，需要设置节点过程中每批次的间隔时间。**不设置**：表示直接升级下一批次；**设置间隔时间**：升级将在间隔时间结束后自动进行下一批次节点升级。
间隔时间	必选	当**每次批间隔时间**选择**设置间隔时间**时，需要设置升级过程中执行下一次批次升级的间隔时间，默认为5分钟，设置范围为5～120分钟。

6. 单击**前置检查**，在升级检查全部通过后开始节点升级。
7. 在升级过程中，您可以进行如下操作。
   - 暂停：暂停升级。单击**暂停**，节点组将暂定升级，升级中的节点将继续升级直至完成。
   - 继续：继续升级。单击**继续**，节点组将继续执行剩余节点的升级。
   - 取消：取消升级，需要先暂停升级。单击**取消**，节点组将后续节点的升级。
8. 升级完成后，您可以在节点页面单击节点名称，在基本信息页查看节点的kubelet版本、容器运行时版本等信息是否符合预期。

> **注意：**
>
> 暂停状态为节点组升级的中间状态，建议您不要在此期间对集群进行操作，并尽快完成升级过程。集群会将处于中间状态的节点组在7日之后

> 关闭升级过程，并清理一切升级相关的事件和日志信息。

自定义节点组 Kubelet 配置

当 Kubelet 参数默认配置无法满足业务需求、您希望对集群节点进行个性化调整时，您可以在节点组维度自定义节点的 Kubelet 参数配置，调整节点行为。例如，您可以调整集群资源预留以调配资源用量、自定义节点压力驱逐阈值以缓解集群资源不足等。CCE 支持通过控制台配置节点的 Kubelet 参数。本文介绍如何配置节点的 Kubelet 参数。

前提条件

- 已成功 创建 CCE 集群。
- 已成功 创建节点组。

使用限制

- 已经放开自定义配置的 Kubelet 参数和集群版本相关，请参考支持的 Kubelet 参数。
- 升级集群后，已配置的 Kubelet 自定义参数不会变更。

注意事项

- 自定义 Kubelet 参数按批次变更节点配置，节点组中已有节点将即时生效，新增节点也会使用新配置。生效时，Kubelet 进程将重启，可能会对节点运行和工作负载运行产生一定影响。请在业务低峰期操作。
- 如果未配置 evictionHard、kubeReserved 或 systemReserved，系统则会使用默认值配置资源预留。请参见CCE节点资源预留策略 。
- 资源预留配置的改变，可能会造成节点的可分配资源变少。对于资源水位较高的节点，可能会触发节点驱逐。
- 更新 Kubelet 参数过程中不支持对节点组进行操作。

支持的 Kubelet 参数

目前支持配置如下表所示的 Kubelet 自定义参数。

> **注意**
>
> - 修改 topologyManagerPolicy、topologyManagerScope 和特性门控（featurGates）中的 QoSResourceManager 参数会重启 Kubelet，并且以更改后的策略重新计算容器实例的资源分配，因此可能会导致已经运行的容器实例重启，甚至无法进行资源分配。请谨慎操作。
> - 随着 Kubernetes 版本的演进，可能会有部分参数或 featureGates 被标记弃用或者从代码移除。如果受容器服务 Kubernetes 版管理的自定义参数在新的版本不再适用时，相关配置会在集群升级期间被移除弃用。

字段	说明	默认值	建议取值范围
kubeReserved	kubernetes 系统预留的资源配置。	根据节点规格自动设置。	详情请参见CCE资源预留说明。
systemReserved	系统预留的资源配置。	根据节点规格自动设置	详情请参见CCE资源预留说明。
allowedUnsafeSysctls	设置允许使用的非安全的sysctl或sysctl通配符（以*结尾的通配符模式）。是使用英文半角逗号（,）分隔的字符串序列。 • 使用此参数前，请仔细评估相关风险并自行保证可用性。	不涉及	支持下述前缀的 sysctl 配置： • kernel.shm • kernel.msg • kernel.sem • fs.mqueue. • net. • **注意**：这些sysctl 参数都支持容器化，但是默认容器内不允许配置
containerLogM	容器的日志文件个数上限。此值必须大于等于2，且容器运行时需为containerd。	5	2~10

axFiles			
containerLogMaxSize	容器日志文件轮换生成新文件的最大阈值。容器运行时需为containerd。	10Mi	不涉及
cpuCFSQuota	允许为设置了 CPU 限制的容器实施CPU CFS 配额约束。	true	可选值 • true • false
cpuCFSQuotaPeriod	设置 CPU CFS 配额周期值。 需确保CustomCPUCFSQuotaPeriod特性门控已被启用。	100ms	介于1毫秒和1秒之间，包含1毫秒和1秒。
cpuManagerPolicy	指定节点的CPU 管理器策略。	none	可选值 • none • static
eventBurst	事件记录的个数的突发峰值上限。在遵从 event-qps 阈值约束的前提下临时允许事件记录达到此数目。仅在 event-qps 大于 0 时使用。	50	1~100
eventRecordQPS	每秒可生成的事件数量。	50	1~50
evictionHard	触发Pod驱逐操作的一组硬性门限。	"imagefs.available": "15%", "memory.available": "100Mi", "nodefs.inodesFree": "5%", "nodefs.available": "7%"	不涉及
evictionSoft	设置一组驱逐阈值。	无	不涉及
evictionSoftGracePeriod	设置一组驱逐宽限期。 • 需已设置evictionSoft。	"5m"	不涉及
featureGates	实验性特性的特性开关组。每个开关以key=value的形式表示。更多信息，请参见Feature Gates。 • 使用此参数前，请仔细评估相关风险并自行保证可用性。	不涉及	不涉及
ImageGCHighThresholdPercent	配置镜像的磁盘用量百分比阈值，一旦镜像用量超过此阈值，镜像垃圾收集会一直运行。 百分比是用这里的值除以 100 得到的，所以此字段取值必须介于 0 和 100 之间，包括 0 和 100。配置时，此值需大于imageGCLowThresholdPercent取值。	85	60～95
ImageGCLowThresholdPercent	配置镜像的磁盘用量百分比阈值，镜像用量低于此阈值时不会执行镜像垃圾收集操作。垃圾收集操作也将此作为最低磁盘用量边界。 百分比是用这里的值除以 100 得到的，所以此字段取值必须介于 0 和 100 之间，包括 0 和 100。配置时，此值需小于imageGCHighThresholdPercent取值。	80	30～90
kubeAPIBurst	每秒发送到APIServer的突发请求数量上限。	500	1~500

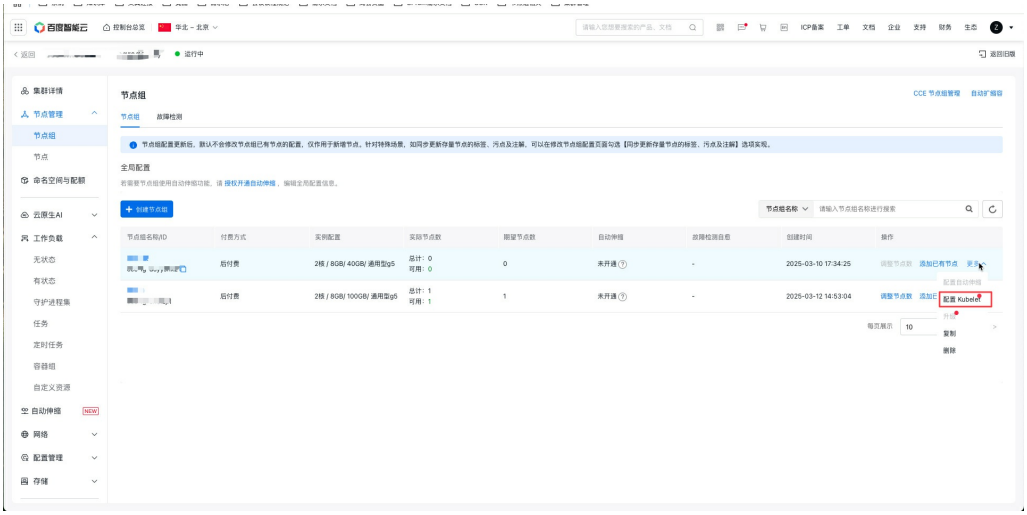
kubeAPIQPS	与APIServer通信的每秒查询个数。	500	1~500
maxPods	Kubelet 上可以运行的最大 Pod 数量。	128	不涉及，取决于机器规格和容器网络规划等物理资源配置。
memoryManagerPolicy	内存管理器需要使用的策略。 • 注意：仅Kubernetes 1.21及以上版本支持该参数。	None	可选值： • None • Static 使用此策略时，需已完成以下操作： 1. MemoryManager特性门控已被启用。 2. 已手动管理内存资源预留（systemReserved、kubeReserved、evictionHard、reservedMemory）并确保配置正确，即kube-reserved + system-reserved + eviction-hard = sum(reserved-memory)，以避免节点不可用。更多信息，请参见预留内存标志。
podPidsLimit	单pod最大进程数。	-1	取值根据用户实际情况来定，推荐默认值，-1表示无限制。
readOnlyPort	kubelet无鉴权只读端口。此端口上的服务不支持身份认证或鉴权。这一端口号必须介于 1 到 65535 之间， 包含 1 和 65535。将此字段设置为 0 会禁用只读服务。 **注意：**关于开放kubelet容器监控只读端口（10255）的风险。	0	0
registryPullQPS	镜像仓库的QPS上限。如果 --registry-qps 大于 0，用来限制镜像仓库的 QPS 上限。设置为 0，表示不受限制。	5	1~50
registryBurst	突发性镜像拉取的个数上限。在不超过 registration-qps 设置值的前提下暂时允许此参数所给的镜像拉取个数。仅在 --registry-qps 大于 0（默认值为 10）时使用。	10	1~100
reservedMemory	NUMA节点内存预留列表。 • 注意：仅Kubernetes 1.21及以上版本支持该参数。	不涉及	不涉及
resolvConf	容器内使用的DNS 解析配置文件。 • 注意：如果此值设置为空字符串，则会覆盖 DNS 解析的默认配置， 本质上相当于禁用了 DNS 查询。	/etc/resolv.conf	不涉及
runtimerequesttimeout	用来设置除长期运行的请求（pull、 logs、exec 和 attach）之外所有运行时请求的超时时长。	120s	不涉及
serializeImagePulls	串行拉取镜像。建议不要在 docker 守护进程版本低于 1.9 或启用了 Aufs 存储后端的节点上更改默认值。	true	可选值： • false：以支持并行拉取镜像，提高Pod启动速度。 • true：建议值，支持串行拉取镜像。
topologyManagerPolicy	拓扑管理器策略。借助NUMA架构，数据可以被分配到相同的NUMA节点，以减少跨节点访问，提高系统性能。拓扑管理器可以作出与拓扑结构相对应的资源分配决定。更多信息，请参见控制节点上的拓扑管理策略。	none	可选值 • none（默认）：不执行拓扑管理策略。 • restricted：当同一 NUMA 节点的 Pod 无法满足资源请求时，kubelet 不允许跨 Pod 实现资源请求。 • best-effort：当同一 NUMA 节点的 Pod 无法满足资源请求时，kubelet 允许跨 Pod 实现资源请求。

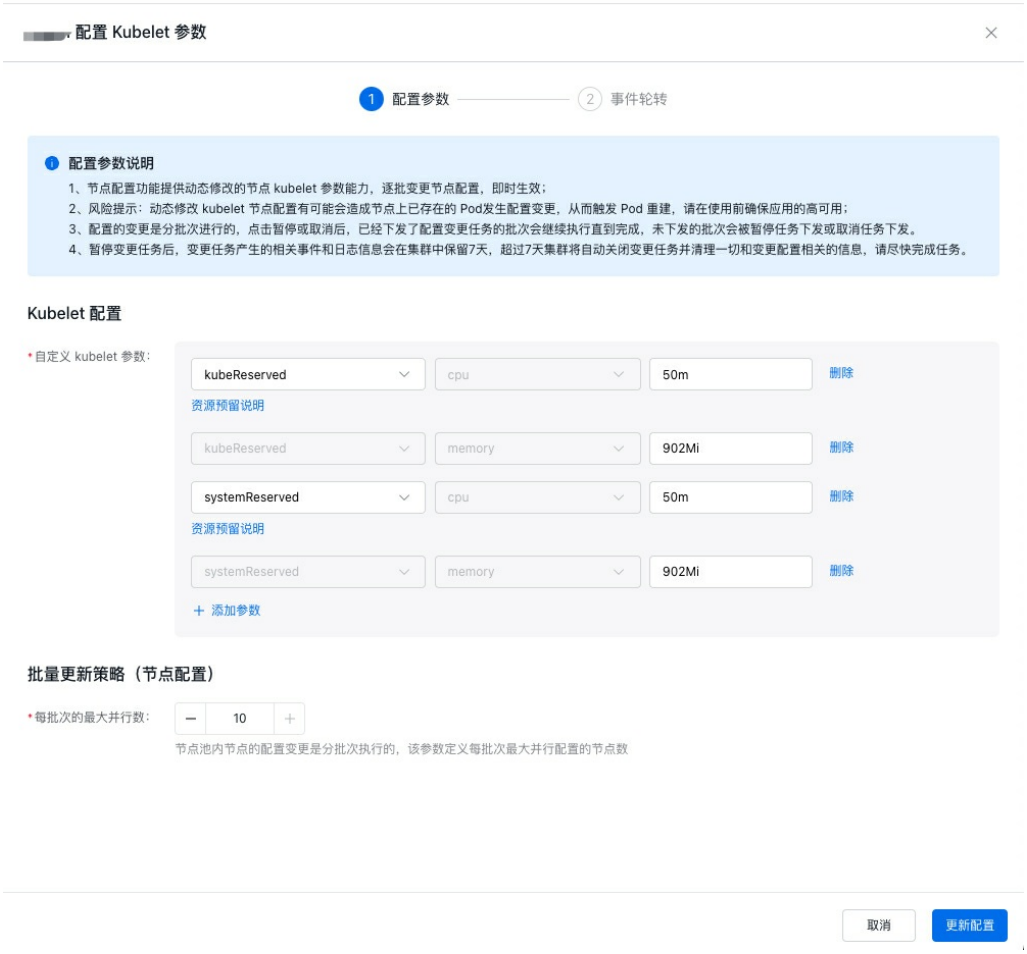
			• single-numa-node：kubelet仅允许在同一NUMA节点的Pod上分配CPU和设备资源。
topologyManagerScope	拓扑管理器作用范围。拓扑提示信息由提示提供者生成，提供给拓扑管理器。	container	可选值 • container • Pod

通过控制台自定义节点组 Kubelet 参数

自定义 Kubelet 参数生效时会重启 Kubelet 进程，可能会对业务产生一定影响。请在业务低峰期进行操作。

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在**集群列表**页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**节点管理** > **节点组**。
5. 在节点组列表页面，单击目标节点组操作列下的更多>**配置 Kubelet**
6. 仔细阅读当前页面上的注意事项，单击自定义参数选择需要配置的参数，设置每批次的最大并行数，然后单击提交，按照页面指引完成操作。
   配置**每批次的最大并行数**后（推荐设置为10），Kubelet 配置将按批次对节点生效。执行需要一定时间，您可以在事件列表区域查看执行进度，并控制执行过程，例如暂停、继续、取消等。
   您可以使用暂停功能对已升级的节点进行验证。执行暂停操作时，正在配置中的节点会继续完成执行；尚未执行的节点在任务继续前不会被执行自定义配置。





> **注意**
>
> 建议尽快完成自定义配置任务。处于暂停状态的任务将在7天后自动取消，相关的事件和日志信息也会被清理。

命名空间管理

命名空间基本操作

命名空间是对一组Kubernetes资源和对象的抽象集合，比如可以用来将系统内部的对象划分为不同的项目组或用户组。创建的 Pod、Service、Deployment、StatefulSet 等都是属于某一个命名空间的。

命名空间分类

按照命名空间创建的方式不同，可将命名空间分为两类：

- 集群默认创建：创建Kubernetes集群时，系统会自动创建一系列命名空间
  - default：集群默认命名空间，创建K8s资源时没有指名命名空间的都部署在该空间。
  - kube-system：Kubernetes系统部署的资源所使用的命名空间，通过组件管理部署的的资源也都部署在该空间。
  - kube-public：所有用户（包括未经过身份验证的用户）支持读取的命名空间，用于部署公共插件、容器模板等公共资源。
  - kube-node-lease：该空间用于存放和各节点相关的租约（Lease），租约由节点定期更新。租约期限内，控制面支持检测节点的心跳，以识别节点是否存在故障。

创建命名空间

1. 登录百度智能云管理控制台。
2. 进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**命名空间与配额**。
3. 单击**新建命名空间**，填写命名空间名称以及对应的标签（选填）。

配置项	必选/可选	配置说明
命名空间	必选	输入命名空间的名称。
标签	可选	设置命名空间的标签信息。

4. 单击**确定**，完成命名空间创建。

编辑命名空间

在命名空间列表页面，单击**编辑**按钮，可以对命名空间标签进行更新。

删除命名空间

在命名空间列表页面，选择要删除的空间，单击**删除**，确认信息后，单击"确认"删除该空间。

> **说明：**
>
> - default、kube-system、kube-public命名空间不支持删除

设置资源配额

命名空间的资源配额（ResourceQuota），用于多团队或多用户的共享集群资源场景下，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（容器、服务）的数量以及消耗计算资源（CPU、内存）的总量，更多信息请见ResourceQuota。

前提条件

- 已创建CCE集群，具体操作请参见创建CCE集群。
- 已创建命名空间，具体操作请参见命名空间基本操作。

操作步骤

1. 登录百度智能云管理控制台。
2. 进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**命名空间与配额**。
3. 在命名空间列表中单击目标命名空间的**更多>设置资源配额**操作
4. 配置该命名空间的计算资源限制、存储资源限制以及其他资源限制。

限制名称	限制说明
CPU Request（请求）	该命名空间下所有非终止状态的Pod的CPU请求总和的上限
CPU Limit（限制）	该命名空间下所有非终止状态的Pod的CPU总和的上限
内存 Request（请求）	该命名空间下所有非终止状态的Pod的内存请求总和的上限
内存 Limit（限制）	该命名空间下所有非终止状态的Pod的内存总和的上限
存储总量	该命名空间下所有PVC的存储资源需求总量的上限
PVC总量	该命名空间所允许的PVC总数的上限
本地临时存储Request（请求）	该命名空间下所有Pod的本地临时存储请求总和的上限
本地临时存储Limit（请求）	该命名空间下所有Pod的本地临时存储总和的上限
容器组总量	该命名空间下所有非终止状态的Pod的总数的上限
Service总量	该命名空间下所有Service的总数的上限
LoadBalance Service总量	该命名空间下所有LoadBalance类型Service的总数的上限
Service使用NodePort总量	该命名空间下所有Service使用NodePort总数的上限
Deployment总量	该命名空间下所有Deployment的总数的上限
StatefulSet总量	该命名空间下所有StatefulSet的总数的上限
Job总量	该命名空间下所有Job的总数的上限
CronJob总量	该命名空间下所有CronJob的总数的上限
Secret总量	该命名空间下所有Secret的总数的上限
ConfigMap总量	该命名空间下所有ConfigMap的总数的上限

> **说明：**
>
> - 不填表示对该资源无限制。
> - Service使用NodePort总量：限制的是该命名空间下NodePort资源的数量，不是NodePort类型Service的数量。

5. 单击**确定**，完成设置命名空间资源配额。

示例

限制 Namespace Pod 个数

本示例演示使用 ResourceQuota 限制命名空间中 PersistentVolumeClaim 的创建数量。

1、创建示例命名空间

```
kubectl create namespace quota-example
```

2、创建 ResourceQuota

将以下 yaml 保存为 quota-demo.yaml，并创建 ResourceQuota `kubectl create -f quota-demo.yaml`：

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: default-resource-quota
 namespace: quota-example
spec:
 hard:
  pods: "4"                    ## 最多 4 个 Pod
  configmaps: "10"             ## 最多 10 个 ConfigMap
  secrets: "10"                ## 最多 10 个 secret
  services: "10"               ## 最多 10 个 service
  services.loadbalancers: "2"    ## 最多 2 个 LoadbanIacer 模式的 service
  cpu: "1000"                  ## 该 Namespaces 下最多使用 1000 个 CPU 的资源
  memory: 200Gi                ## 该 Namespaces 下最多使用 200Gi 的内存
```

3、验证 ResourceQuota

使用以下 YAML 创建 5 个 Pod，但最终只会创建 4 个 Pod 出来：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-example
  namespace: quota-example
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      restartPolicy: Always
      containers:
        - name: nginx
          image: hub.baidubce.com/cce/nginx-alpine-go:latest
          imagePullPolicy: Always
          resources:
            limits:
              cpu: 100m
              memory: 128Mi
            requests:
              cpu: 100m
              memory: 128Mi
```

最多只会创建 4 个 Pod：

```
 $kubectl get deployment -n quota-example
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
deployment-example   4/5     4           4           59s
```

设置资源限制

命名空间中的资源使用限制（LimitRange）有利于资源的合理分配和管理，提高集群资源的利用率和可用性。本文介绍如何设置命名空间中资源的使用限制。，更多信息请见LimitRange。

前提条件

- 已创建CCE集群，具体操作请参见创建CCE集群。
- 已创建命名空间，具体操作请参见命名空间基本操作。

操作步骤

1. 登录百度智能云管理控制台。
2. 进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**命名空间与配额**。
3. 在命名空间列表中单击目标命名空间的**更多>设置资源限制**操作。
4. 配置该命名空间的CPU、内存资源限制。

限制名称	限制说明
CPU Request（请求）	容器没有声明CPU限制的情况下，该命名空间中每个容器执行请求占用CPU的最小值
CPU Limit（限制）	容器没有声明CPU限制的情况下，该命名空间中每个容器执行请求占用CPU的最大值
内存 Request（请求）	容器没有声明内存限制的情况下，该命名空间中每个容器执行请求占用内存的最小值
内存 Limit（限制）	容器没有声明内存限制的情况下，该命名空间中每个容器执行请求占用内存的最大值

> **说明：**
>
> - 不填表示对该资源无限制。
> - 仅在ResourceQuota设置了CPU、内存限制之后Rangelimit才会对Pod里的容器生效。

5. 单击**确定**，完成设置命名空间资源限制。

示例

使用 LimitRange 为 Container 设置默认 Resource

创建 LimitRange YAML，声明 Container 内存的默认限制量和默认请求量：

```
 apiVersion: v1
kind: LimitRange
metadata:
 name: mem-limit-range
 namespace: quota-example
spec:
 limits:
 - default:
    memory: 512Mi
  defaultRequest:
    memory: 256Mi
  type: Container
```

创建 Pod 如果没有声明 memory，会默认设置上该值：

```
  spec:
  resources:
    limits:
      memory: 512Mi
    requests:
      memory: 256Mi
```

更多内容可以参考：通过 LimitRange 限制范围

使用 LimitRange 为 Container 限制 Limits/Requests 的比例

> **说明：**
>
> 如果指定了 LimitRange 对象的 spec.limits.maxLimitRequestRatio 字段，名称空间中的 Pod/容器的 request 和 limit 都不能为 0，且 limit 除以 request 的结果必须小于或等于 LimitRange 的 spec.limits.maxLimitRequestRatio。

1、创建 LimitRange YAML，声明名称空间中限制 Limits/Requests 的比例：

```
 apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: quota-example
spec:
  limits:
  - maxLimitRequestRatio:
      memory: 2
    type: Container
```

2、使用以下 YAML 创建 Deployment, 会发现 Pod 的最大内存限定（limit）超过最小内存请求（request）的两倍而无法创建 Pod ：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-example
  namespace: quota-example
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      restartPolicy: Always
      containers:
        - name: nginx
          image: hub.baidubce.com/cce/nginx-alpine-go:latest
          imagePullPolicy: Always
          resources:
            limits:
              cpu: 100m
              memory: 1280Mi
            requests:
              cpu: 100m
              memory: 128Mi
```

Helm管理

Helm模板

Helm模板对应Helm社区的Helm Chart概念，是Helm定义的一种打包格式，用来描述一组相关的kubernetes资源。Helm 能够帮您管理自定义的 Kubernetes 资源集合，从而快速实现复杂应用的定义、安装、升级。

备注：Helm模板页面提供了模板托管服务，该服务不区分地域。

Helm客户端

helm客户端为命令行工具，可以在Helm Github页面进行下载。

模板市场

开放使用的Helm模板。可以点击模板名称进入模板详情，查看模板介绍、用法和参数。

我的模板

用户可以上传自己的Helm模板包，并在集群中部署。

用户自行上传的模板将在"Helm模板-我的模板"中展示，并可以通过helm命令行工具操作其中的模板。

模板包准备建议使用helm客户端操作。

准备模板包

您可以通过如下两种方法准备模板包：

1. 自定义模板包。模板包制作方式和文件结构可以参考Helm社区文档:

- https://helm.sh/docs/topics/charts/
- https://helm.sh/docs/chart_template_guide/

2. 使用Kubernetes官方模板包。访问https://hub.helm.sh/ ，可以获取当前社区所有模板包。

模板包命名规范

若要上传到百度智能云Helm模板仓库，模板包内Chart.yaml中的`name`和`version`字段须符合以下规则：

- `name`长度不能超过32个字符，必须以小写字母开头，可以包含小写字母、数字、-。
- `version`中主版本号、次版本号和修订号只能填写非负整数，语义参考语义化版本规则，不支持pre-release格式的修订版本号。

打包模板包

建议使用helm客户端操作。

在模板包文件夹所在目录执行`helm package ./{name}`，其中{name}为模板包名称，该操作会在目录下生成相应的模板包tgz文件。

```
 $ ls -l
drwxr-xr-x  6 root root     4096 Feb 15 23:07 mysql
$ helm package ./mysql
Successfully packaged chart and saved it to: /root/my-charts/mysql-0.15.0.tgz
$ ls
mysql  mysql-0.15.0.tgz
```

上传模板包

点击Helm模板-我的模板页面的上传模板按钮，选择刚刚打包完成的模板包，点击导入按钮即可上传模板包到我的模板中。

也可以通过helm客户端操作模板包的上传，参考使用helm客户端操作模板仓库。

使用helm客户端操作我的模板

可以通过helm客户端操作页面上"我的模板"对应的模板仓库，模板仓库地址见我的模板右上角-仓库地址按钮。

添加仓库时需要提供模板仓库用户名密码，可以通过右上角创建/修改模板仓库用户名密码。对于已经创建过镜像仓库用户名密码的用户，模板仓库用户名密码与镜像仓库用户名密码一致。

使用helm客户端添加模板仓库的命令如下：

```
$ helm repo add cce-private <模板仓库地址> --username=<username> --password=<password>
```

完成repo添加后，可以通过helm search repo命令查看模板仓库中的模板包。

如果需要通过helm客户端上传模板包，还需要安装helm-push plugin添加helm push命令，使用文档参考 https://github.com/chartmuseum/helm-push。

对于helm客户端支持的的其他模板仓库操作，可以参考Helm社区文档 https://helm.sh/docs/helm/helm_repo/。

安装模板

在列表页面和模板详情顶部的点击安装按钮，默认安装模板的最新版本。如果要安装历史版本，可以在模板详情-版本中选择历史版本进行安装。

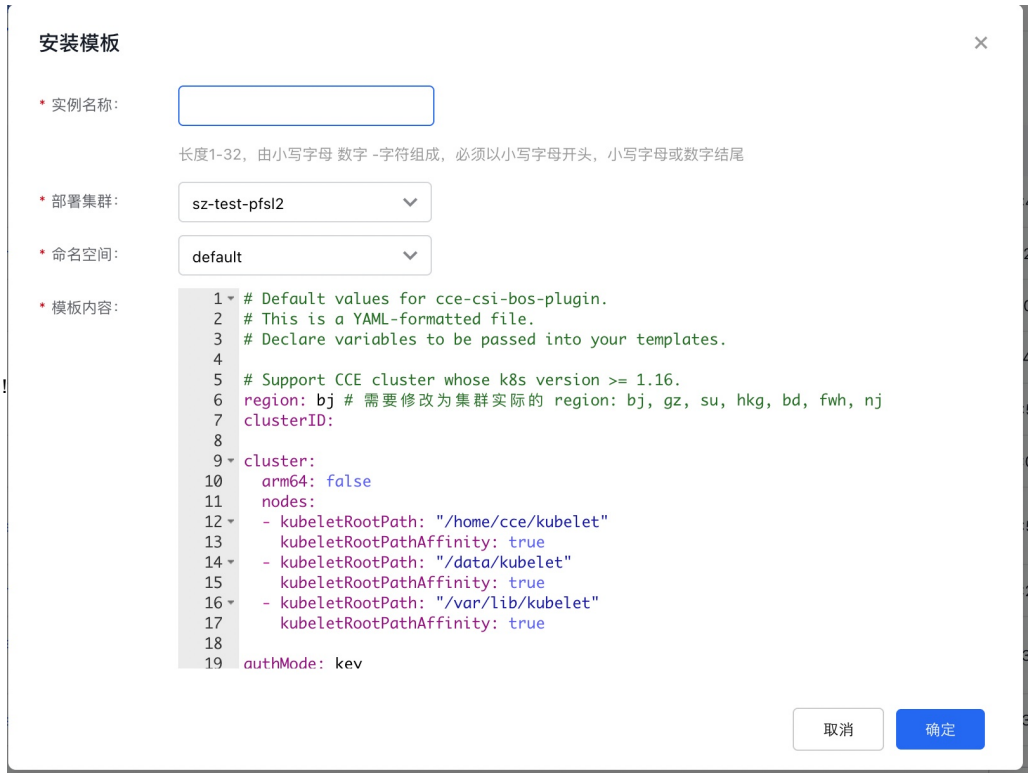
安装模板时，需要填写实例名称，目标CCE集群和集群命名空间，以及模板参数。

实例名称规范如下：

- 实例名称不能超过32个字符，必须以小写字母开头，小写字母或数字结尾，可以包含小写字母、数组、-。
- 实例名称不能与目标集群中已有的实例冲突。

模板参数文本框内填充模板默认参数值，可以在安装模板时进行修改。

安装完成后，会自动跳转到对应的Helm实例页面。



也可以通过helm客户端连接集群后，在对应集群中安装指定的helm模板，详情可以参考以下文档：

- 通过helm客户端连接集群
- helm Install

Helm实例

Helm实例对应Helm社区的Helm Release概念，安装Helm模板到集群中，即可产生一个实例。

Helm实例页面展示了用户选择的CCE集群和命名空间内的Helm实例信息。

在集群中安装helm组件

仅集群id以c-开头的老版本集群需要该步骤。集群id为cce-开头的新版CCE集群已经切换到helm v3，故不再需要该步骤。

对于未安装Helm服务端组件的CCE集群，在Helm实例列表页会提示 集群未安装helm。

点击安装按钮会在当前右上角选择的集群中安装Helm的服务端组件（不包含Helm客户端）。

安装完毕后，用户可以通过页面在该集群中使用Helm实例相关功能。

实例列表

当前选择的集群命名空间内没有实例时，显示暂无数据；否则会显示当前集群命名空间内的实例列表。

列表中展示了各个实例的部署状态，来源模板，最后更新时间等信息。

实例详情

点击列表中的实例名称，进入对应实例的详情页面。

详情页面分为以下四个部分：

- 基本信息：包含实例名称，所在集群/命名空间，版本，更新时间，描述等基本信息。
- 资源信息：该实例中包含的kubernetes资源信息。
- 版本：该实例的所有版本，包含历史版本和当前版本。可以查看每个版本的状态和参数，并选择回滚到特定版本。
- 参数：实例当前版本使用的实例参数。选中 查看全部 复选框后，会显示所有参数的值（包含默认值），否则仅显示用户定义的参数。

更新实例

可以在集群列表页面点击对应实例的更新按钮，或者在实例详情页面-参数文本框点击编辑按钮，会弹出参数编辑框，用户可以编辑参数完成后，点击更新按钮更新实例。更新实例操作会产生一个历史版本。

回滚实例

回滚实例，即是将实例参数设置为某个历史版本对应的值。回滚操作会产生一个新版本。

在实例详情页面，点击需要回滚的版本行尾的回滚按钮，确认参数无误后，点击确定即可回滚到指定版本。

通过helm客户端连接集群

由于新版本集群（集群id为cce-开头）和老版本集群（集群id以c-开头）分别对应helm的v3和v2版本，故连接集群使用的客户端版本和具体操作会有所区别。请根据需要连接的集群版本对照下述说明操作。

连接集群id为cce-开头的新版集群

在Helm Github页面下载最新helm客户端。

在访问集群apiserver的环境中，直接通过对应集群的kubeconfig即可连接。

如kubeconfig不在默认路径下(~/.kube/config)，通过`--kubeconfig`指定kubeconfig文件即可使用。

连接集群id以c-开头的老版本集群

注意: 由于helm v2版本未支持Kubernetes RBAC特性，因此该连接方式无法限制连接者权限，如需要控制集群内权限请勿使用helm客户端操作。

在Helm Github页面下载helm v2.12.3客户端。

由于CCE安装的helm tiller时开启了tls校验，需要指定集群kubeconfig和helm tiller使用的tls证书后方可连接。

tls证书在集群node的文件系统上有保存，可以登录集群node获取。

登录到在集群node后，下载helm客户端和集群kubeconfig，并指定以下env后可以连接集群中的helm tiller进行操作：

```
 export HELM_TLS_ENABLE=true
export HELM_TLS_CA_CERT=/etc/kubernetes/pki/ca.pem
export HELM_TLS_CERT=/etc/kubernetes/pki/admin.pem
export HELM_TLS_KEY=/etc/kubernetes/pki/admin-key.pem
```

也可以将上述tls证书文件和集群kubeconfig下载到本地后连接集群使用。

常见问题

删除了模板后，集群中以该模板创建的实例还能继续使用吗

删除模板后，从该模板创建的实例将无法进行升级操作。

安装/升级/回滚实例失败，如何进行排查

安装/升级/回滚操作失败时，实例详情页面-基本信息中的描述会展示具体的错误信息，用户可以通过其中的错误信息定位失败原因，进行修正。

我已经在集群自行安装了helm服务端组件，可以直接使用吗

由于CCE安装的Helm组件开启了tls校验，且Helm客户端和服务端间无法跨版本通信，如果用户已经在集群中自行安装了Helm服务端组件（如自行执行了`helm init`命令），可能与CCE支持的Helm服务端组件不兼容，需要用户卸载集群中的Helm组件后，重新通过实例列表页面进行安装。

工作负载

Deployment管理

概述

用户可以通过表单方式基于镜像创建部署，也可以通过百度智能云提供的 YAML 文件模板创建和修改部署，对已部署应用的资源进行实时监控，同时还可以便捷地对应用及其所关联的服务（Service）、容器组（Pod）的生命周期和部署方式进行控制管理。

前提条件

- 已成功创建 CCE 集群，操作说明参见 创建 CCE 集群。

创建无状态工作负载（Deployment）

用户可以选择通过 CCE 控制台使用表单形式或者通过 yaml 形式创建新的无状态工作负载，也可以选择通过 kubectl 命令创建。

方式一：通过控制台表单形式创建

1. 登录 容器引擎CCE控制台。
2. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
3. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
4. 点击无状态工作负载列表左上角**新建无状态**，进入创建工作负载页面。
5. 在基本信息模块完成相关配置，点击**下一步**进入容器配置。

参数	说明
工作负载名称	根据系统提示，配置工作负载的名称。在同一个命名空间下，名称必须唯一。
命名空间	选择工作负载部署的命名空间，默认为default。支持选择自定义创建的命名空间，自定义命名空间，请参见 命名空间操作。
工作负载类型	选择无状态工作负载Deployment。
期望 Pod 数	输入工作负载的 Pod 实例个数。
K8s标签	指定工作负载的标签，对应 Kubernetes 中的 Label，用于识别工作负载的键值对，配合选择器，帮助 Kubernetes 快速筛选目标工作负载。详细信息参考K8S 标签说明。
注释（Annotation）	指定工作负载的注解信息，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

6. 在容器配置模块完成相关配置，填写容器名称和镜像地址时，可点击**选择镜像**后在弹窗中选择 CCR 中的镜像地址。点击**下一步**进入高级配置模块。

> **说明**：
>
> - 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。
> - 若使用了私有镜像，需在当前页面的高级设置中配置仓库访问凭证。

参数	说明
容器名称	根据系统提示的命名规则，输入容器名称。同一个工作负载下容器名称必须唯一。
镜像地址	单击后方“选择镜像”，选择容器使用的镜像。支持选择以下两种种类型的镜像。 • 容器镜像 企业版 ：可以选择托管在容器镜像服务 CCR 中的企业版镜像。关于 CCR 的更多信息，详细信息请参考 CCR 企业版快速入门。 • 容器镜像 个人版：可以选择托管在容器镜像服务ACR中的个人版镜像。关于 CCR 的更多信息，详细信息请参考CCR 个人版说明。
镜像版本	选择需要部署的镜像版本。
镜像拉取策略	选择镜像拉取策略。CCE 支持以下三种镜像拉取策略（imagePullPolicy）： • 优先使用本地镜像（IfNotPresent）：如果节点上有本地镜像，则使用本地镜像，本地不存在时拉取镜像。 • 总是拉取镜像（Always）（默认）：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 • 仅使用本地镜像（Never）：仅使用本地镜像，本地无镜像时拉取失败。

容器配额	配置容器资源的resources.limits和容器资源的resources.requests。如不填写申请值和限制值，表示不限制配额。申请值和限制值的配置说明及建议，请参见Requests and Limits。
	• 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，防止占用过多资源。
	• 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。
	• 加速卡资源申请：仅具有 GPU/NPU 计算型节点的集群，可配置 GPU/NPU 资源。开启之后按需选择设置 GPU 和 NPU。
	  • 注意：若需要支持GPU感知调度，以及GPU算力和显存的共享与隔离，集群中需安装CCE GPU Manager组件。若需要 NPU 感知调度，以及按算力和显存分配NPU资源，请先安装CCE NPU Manager 组件。
	  • 不开启：表示不使用GPU。
	  • 独占GPU：单个容器独享GPU

	• 独占GPU：单个容器独享GPU。
容器端口	设置容器端口名称、协议、端口。 • 容器端口的名称，仅用于区分端口，没有实际功能。 • 协议：容器端口使用的4层（传输层）协议，支持TCP和UDP。 • 设置容器对外暴露的端口，必须介于1~65535之间。容器只有对外暴露端口，才能实现对Pod外暴露以及Pod中容器互相通信。一个Pod中的所有容器共享Pod的网络协议栈，因此在Pod中配置多个容器时，端口不能重复。
环境变量	单击 添加环境变量，配置环境变量。Kubernetes 在创建 Pod 时，会将环境变量的信息注入到容器中。 支持以下类型： • 自定义：使用env直接在工作负载中硬编码环境变量，支持输入变量名称和变量值，创建自定义的环境变量。 • 配置项(ConfigMap)：使用envFrom获取存储在ConfigMap中的非敏感配置数据。支持部分引用或全部引用配置项中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。 • 保密字典（Secret）：使用envFrom获取存储在ConfigMap中的敏感信息，如密码、API 密钥等。支持部分引用或全部引用保密字典中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。
容器启动项	为容器添加启动项参数，当前支持以下容器启动项参数： • stdin：为容器开启标准输入，允许从外部源（如终端或其他输入流）接收输入。 • tty：为容器分配一个虚拟终端，允许您像在本地终端中一样向容器发送命令。一般需要同时开启stdin，表示将终端（tty）绑定到容器的标准输入（stdin）上，可以支持类似于`kubectl exec -i -t`命令的交互式操作，但区别在于该参数在Pod启动时就已经配置完成。
特权容器	默认不开启。开启后，特权模式允许容器拥有与所在的Worker节点操作系统近似的权限，例如访问硬件设备、挂载文件系统等。
初始化容器	默认不开启。选择容器是否作为初始化（Init）容器。初始化（Init）容器不支持设置健康检查。Init容器是一种特殊容器，可以在Pod中的其他应用容器启动之前运行。每个Pod中可以包含多个容器，同时Pod中也可以有一个或多个先于应用容器启动的Init容器，当所有的Init 容器运行完成时，Pod中的应用容器才会启动并运行。详细说明请参见Init 容器。 • 注意：容器组中不能全为初始化容器，需要保证至少有一个非初始化容器。若容器组中有多个初始化容器，它们的运行顺序将与其创建顺序一致。

健康检查	支持配置**存活检查**、**就绪检查**和**启动检查**。允许基于HTTP请求检查、TCP端口检查、执行命令检查和GRCP检查的方式，探测容器中的负载是否正常。以下以配置存活检查为例，介绍配置参数。
	• HTTP请求检查：使用HTTP或HTTPS协议，向容器发送一个HTTP GET请求，通过检查响应确认容器状态。您需要进行如下检查配置：
	  • 检查协议：支持HTTP或HTTPS。
	  • 检查路径：访问HTTP Server的路径。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	  • Http请求头：HTTP请求中自定义的请求头（HTTP Headers）。单击添加请求头输入键值对的配置方式。
	• TCP端口检查：向容器发送一个TCP Socket，Kubelet将尝试在指定端口上打开容器的Socket。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。您需要进行如下检查配置：
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	• 执行命名检查：在容器内执行指定的命令，通过检查命令执行的结果确认容器状态。 您需要进行如下检查配置：
	  • 执行命令：检测容器健康情况的探测命令。多个命令换行分隔。
	• GRPC检查：使用gRPC协议，通过发送POST或GET请求检查服务器应用是否健康。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • GRPC服务：指定GPRC服务名称，不填为整体服务。
	• 公共参数说明：

	• 启动延时：指容器启动后，第一次执行探测时需要等待的时间。 • 间隔时间：相邻两次检查的时间间隔。 • 响应超时：指发送检查请求后，等待响应的超时时间。 • 健康阈值：检查请求发送后，表示系统正常的响应次数。 • 不健康阈值：检查请求发送后，表示系统异常的无响应次数。
生命周期	在容器的生命周期的特定阶段配置需要执行的操作，例如启动命令、启动后处理和停止前处理，详情请参见设置容器生命周期。 • 启动命令：为容器设置预启动命令和参数。启动命令与参数定义了容器启动时执行的操作，用于初始化应用服务。适用于需要特定环境变量、挂载点或端口映射的应用部署场景。 • 启动后处理：为容器设置启动后的命令。启动后处理命令用于容器启动后执行特定任务，如初始化配置、运行脚本等，适用于需要在主进程前完成准备工作的场景。 • 停止前处理：为容器设置预结束命令。停止前处理命令用于关闭容器内的应用进程，确保数据一致性与服务正常终止。适用于需安全停机避免数据丢失或服务异常的场景。
数据卷	支持为容器挂载不同类型的数据卷进行数据持久化，支持通过subPath方式挂载卷的子路径；支持通过subPathExpr方式挂载卷的扩展路径： • 主机目录：使用主机中的目录创建数据卷并挂载。 • 临时目录：使用临时目录创建数据卷。 • 配置项：使用已有配置项ConfigMap创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `cm1:path1;cm2:path2`。 • 保密字典：使用已有保密字典Secret创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `sc1:path1;sc2:path2`。 • 存储卷声明：使用已有存储卷声明PVC创建数据卷。
仓库访问凭证	若有容器使用了私有镜像，请务必添加对应的镜像仓库的访问凭证。支持新建访问凭证和选择已有的，详细信息参考配置访问凭证。

7. 在高级设置模块中，配置伸缩策略、更新策略和调度策略，点击**完成**进行工作负载的创建。

参数	说明
伸缩配置	支持两种伸缩配置：水平伸缩和定时水平伸缩。 • **水平伸缩**：通过监测容器的性能指标触发自动伸缩。指标伸缩可以帮助您在业务负载出现波动时自动调整工作负载使用的资源总量，在高负载时扩容缓解压力，低负载时缩容节省资源。更多信息，请参见使用容器水平伸缩（HPA）。 • **定时水平伸缩**：通过定时触发工作负载伸缩，适用于业务负载存在周期性变化的场景，例如社交媒体在午餐后和晚餐后周期性的流量高峰。更多信息，请参见使用容器定时水平伸缩（CronHPA）。 • 注意：使用定时水平伸缩 CronHPA 需要安装CCE CronHPA Controller组件，详细参考 CCE CronHPA Controller介绍。

更新策略	Pod配置发生变更时，工作负载使用新Pod替换旧Pod的机制，包含两种策略：
	• **滚动更新**：对该工作负载的Pod逐个进行更新，业务流量始终会负载均衡到旧版本的可用Pod上，避免更新过程中业务中断。
	  • 最大不可用Pod数(maxUnavailable)：指定 滚动更新 缩容过程中，工作负载中可用的实例 （Pod） 的数量的下限。选择 滚动更新 策略时，需要配置此配置项。支持基于百分比或具体数值配置 Pod 的最大不可用数量。使用百分比时，会将计算结果转换成绝对值并去除小数部分。最大不可用 和 最大超量 不能同时为 0。
	  • 最大超出期望Pod数：指定 滚动更新 扩容过程中，工作负载中可用的实例 （Pod） 的数量的上限。选择 滚动更新 策略时，需要配置此配置项。支持基于百分比或具体数值配置 Pod 的最大超量。使用百分比时，会将计算结果转换成绝对值并去除小数部分。最大不可用 和 最大超量 不能同时为 0。
	  • 最小准备时间（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）。

	• **替换更新**：对该工作负载的Pod逐个进行更新，业务流量始终会负载均衡到旧版本的可用Pod上，避免更新过程中业务中断。 • 最小准备时间（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）。

调度策略	通过配置亲和与反亲和规则，可实现灵活的工作负载调度，支持负载亲和与节点亲和。
	• **节点亲和性**：配置工作负载和节点的亲和性调度。包括：强制满足和尽量满足。
	  • 不配置：不设置节点亲和策略。
	  • 强制调度：通过匹配标签进行调度选择。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	  • 尽量满足：通过匹配标签进行调度选择，允许配置权重。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	• **Pod亲和性**：选择Pod将与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **Pod 反亲和性**：选择Pod将不与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod反亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **容忍设置** ：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，详细信息请参考污点和容忍说明。

Pod标签	为该工作负载所属的每个Pod添加标签，对应 Kubernetes 中的 Label，用于识别Pod的键值对，配合选择器，帮助 Kubernetes 快速筛选目标Pod。详细信息参考K8S 标签说明。
Pod注释	为该工作负载所属的每个Pod添加注解，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

方式二：通过 YAML 创建

1. 点击部署列表左上角**使用YAML创建**，进入新建工作负载页面。
2. 点击下拉列表选择命名空间，填写 yaml 文件，点击**确定**。

> **说明：**
>
> • 模板类型：可使用示例模板或“我的模板”创建。
>
> • 复制：复制当前 YAML 文件的内容。
>
> • 另存为：将当前文件模版另存为“我的模版”。
>
> • 取消：返回列表页。

工作负载YAML示例

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-example
 labels:
```

```
    app: nginx
spec:
  replicas: 2
  minReadySeconds: 0
  strategy:
    type: RollingUpdate # 策略：滚动更新
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      restartPolicy: Always
      # restartPolicy: OnFailure
      # restartPolicy: Never
      containers:
        - name: nginx
          image: registry.baidubce.com/cce/nginx-alpine-go:latest
          imagePullPolicy: Always
          # imagePullPolicy: IfNotPresent
          # imagePullPolicy: Never
          ports:
            - containerPort: 80
          resources: # 资源限制
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
          livenessProbe: # 健康检查：存活探针
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 20
            timeoutSeconds: 5
            periodSeconds: 5
          readinessProbe: # 健康检查：就绪探针
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 5
            timeoutSeconds: 1
            periodSeconds: 5
          # volumeMounts:
          #   - name: nginx-volume
          #     mountPath: "/usr/share/nginx"
      # volumes:
      #   - name: nginx-volume
      #     flexVolume:
      #       driver: "baidubce/cds"
      #       fsType: "ext4"
      #       options:
      #         volumeID: "{id}" # 填写cds的id，注意：cds必须和pod在同一可用区！！
      # nodeSelector:
      #   beta.kubernetes.io/os: linux
      # tolerations: # 容忍污点
      #   - effect: NoExecute
      #     operator: Exists
      #   - effect: NoSchedule
      #     operator: Exists
      # affinity: # 亲和性
      #   nodeAffinity:
      #     requiredDuringSchedulingIgnoredDuringExecution:
      #       nodeSelectorTerms:
```

```
      #       - matchExpressions:
      #           - key: failure-domain.beta.kubernetes.io/zone
      #             operator: In
      #             values:
      #               - zoneA
      #               - zoneB
      #               - zoneC
```

方式三：通过 kubectl 创建

1. 准备 Deployment yaml 文件。
2. 安装 kubectl，并连接集群，详情参见 通过 kubectl 连接集群。
3. 复制下方的YAML文件，并保存到 deployment.yaml中。下方的YAML中定义了一个Deployment以及用于ClusterIP类型Service。

```
 apiVersion: apps/v1
kind: Deployment    # 工作负载类型
metadata:
  name: nginx-test
  namespace: default  # 根据需要更改命名空间
  labels:
    app: nginx
spec:
  replicas: 2  # 指定Pod数量
  selector:
    matchLabels:
      app: nginx
  template: # Pod配置
    metadata:
      labels: # Pod标签
        app: nginx
    spec:
      containers:
      - name: nginx  # 容器名称
        image: registry.baidubce.com/cce/nginx-alpine-go:latest # 使用特定版本的 Nginx 镜像
        ports:
        - containerPort: 80  # 容器暴露的端口
          protocol: TCP  # 指定协议为 TCP/UDP，默认也是 TCP
---
##### service
apiVersion: v1
kind: Service
metadata:
  name: nginx-test-svc
  namespace: default  # 根据需要更改命名空间
  labels:
    app: nginx
spec:
  selector:
    app: nginx  # 匹配标签，确保服务指向正确的 Pods
  ports:
    - port: 80          # Service 在集群内提供的端口
      targetPort: 80    # 指向容器内部应用程序监听的端口 (containerPort)
      protocol: TCP     # 协议，默认是 TCP
  type: ClusterIP     # 服务类型，默认是 ClusterIP，内部访问
  sessionAffinity: None # 默认值，不使用客户端会话亲和性
```

4. 执行以下命令，创建Deployment及Service。

   ```
   kubectl apply -f deployment.yaml
   ```

   预期输出：

   ```
   deployment.apps/nginx-test created
   service/nginx-test-svc created
   ```

5. 执行以下命令，查看Service的公网IP地址

```
kubectl get svc
```

6. 执行以下命令，验证 Deployment 是否创建成功：

```
kubectl get deployments
```

查看工作负载详情

点击列表页任何一行的工作负载名称，即可进入工作负载详情页，详情页内容包括： 工作负载详情、关联容器组、关联服务、更新升级、伸缩策略、调度策略、监控、事件等信息。

更新 Pod 数

支持通过控制台快速更新无状态负载中的 Pod 实例个数。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 在目标无状态负载的列表，点击需要调整的无状态行中的“**伸缩**”，在伸缩的弹框页中，填写需要的容器组数量，点击“确认”。

更新无状态工作负载

支持通过控制台界面或 YAML 文件编辑已创建的无状态负载。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 工作负载 > 无状态负载。
2. 在目标无状态负载的操作列，单击 **更新升级** 或 **编辑 YAML**。

> 说明：
> 通过控制台界面更新和编辑 YAML 文件更新，均不支持更新无状态负载名称、命名空间。

a. 更新：在控制台更新配置信息。详细的参数说明，请参见 [通过控制台创建](https://cloud.baidu.com/doc/CCE/s/cjxpoxrgi)。
b. 编辑 YAML：编辑 YAML 文件更新配置信息。详细的 YAML 配置说明，请参见[通过 YAML 创建](https://cloud.baidu.com/doc/CCE/s/cjxpoxrgi)。

3. 更新完成后，单击**提交**。

重启工作负载

支持通过控制台重启无状态负载。重启后，将重启工作负载下的全部容器组（Pod）。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 选择“**无状态**”页签，单击工作负载后的“**更多** > **重启**”。
3. 在弹出的提示框中单击“确认”，即可完成工作负载的重启。

标签注解管理

标签是以key/value键值对的形式附加在无状态工作负载上的。添加标签后，可通过标签对无状态工作负载进行管理和选择。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。

> **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，详细 K8S标签说明。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 选择“**无状态**”页签，单击工作负载后的“**更多** > **标签注解**”。
3. 单击**添加标签**，输入键和值后单击**更新**。

查看监控

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。

> **说明**：
> 查看工作负载的监控信息时，依赖集群接入 Prometheus 监控服务。若集群未接入Prometheues 监控服务，请根据界面提示接入，详细信息请参见使用Prometheus监控集群。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 在目标无状态负载的列表，点击需要调整的无状态行中的“**监控**”。
3. 在监控页面查看无状态负载的监控数据。容器相关监控指标说明，请参见 监控指标说明。

查看日志

您可以通过“日志”功能查看无状态工作负载的日志信息。

> **说明**：
> 当前显示的日志内容为容器标准输出日志，不具备持久化和高阶运维能力，如需使用更完善的日志能力，可使用日志中心功能。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 在目标无状态负载的列表，单击工作负载后的“**更多** > **日志**”。
3. 在日志页面查看容器日志数据。

删除无状态负载

若工作负载无需再使用，您可以将工作负载删除。

> **说明**：
> 删除无状态负载可能导致服务不可用，请谨慎操作。

1. 登录无状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **无状态**。
2. 在目标无状态负载的列表，单击工作负载后的"**更多** > **删除**"。
3. 在系统弹出的二次确认对话框，确认信息后，点击**确认**。

Daemonset管理

概述

守护进程（DaemonSet）确保集群中符合条件的节点上有且仅有一个 Daemon Pod 运行。守护进程适合于系统层面的应用，例如日志收集、资源监控等，可以有效保障集群中的基础服务的可用性，关于DaemonSet的更多信息，请参见官方文档。用户可以通过表单方式基于镜像创建守护进程应用，也可以通过百度智能云提供的YAML文件模板创建和修改守护进程应用，对已守护进程关联的容器组（Pod）、事件等进行控制管理。

DaemonSet 介绍

DaemonSet保证在每个Node上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。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

- 已成功创建 CCE 集群，操作说明参见 创建 CCE 集群。

创建守护进程集 DaemonSet

用户可以选择通过 CCE 控制台使用表单形式或者通过 YAML 形式创建新的守护进程集，也可以选择通过 kubectl 命令创建。

通过控制台表单形式创建

1. 登录 容器引擎CCE控制台。
2. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
3. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **守护进程集**。
4. 点击无状态工作负载列表左上角**新建守护进程集**，进入创建工作负载页面。
5. 在基本信息模块完成相关配置，点击**下一步**进入容器配置。

参数	说明
工作负载名称	根据系统提示，配置工作负载的名称。在同一个命名空间下，名称必须唯一。
命名空间	选择工作负载部署的命名空间，默认为default。支持选择自定义创建的命名空间，自定义命名空间，请参见 命名空间操作。
工作负载类型	选择守护进程集DaemonSet。
期望 Pod 数	输入工作负载的 Pod 实例个数。
K8s标签	指定工作负载的标签，对应 Kubernetes 中的 Label，用于识别工作负载的键值对，配合选择器，帮助 Kubernetes 快速筛选目标工作负载。详细信息参考K8S 标签说明。
注释	指定工作负载的注解信息，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

6. 在容器配置模块，填写容器名称和镜像地址，可点击选择镜像后在弹窗中选择 CCR 中的镜像地址。点击下一步进入高级配置模块。

> **说明**
>
> - 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。
> - 若使用了 CCR 私有镜像，还需在当前页面的高级设置中配置仓库访问凭证。

参数	说明
容器名称	根据系统提示的命名规则，输入容器名称。同一个工作负载下容器名称必须唯一。
镜像地址	单击后方“选择镜像”，选择容器使用的镜像。支持选择以下两种种类型的镜像。 • 容器镜像企业版 ：可以选择托管在容器镜像服务 CCR 中的企业版镜像。关于 CCR 的更多信息，详细信息请参考 CCR 企业版快速入门。 • 容器镜像个人版：可以选择托管在容器镜像服务ACR中的个人版镜像。关于 CCR 的更多信息，详细信息请参考CCR 个人版说明。
镜像版本	选择需要部署的镜像版本。
镜像拉取策略	选择镜像拉取策略。CCE 支持以下三种镜像拉取策略（imagePullPolicy）： • 优先使用本地镜像（IfNotPresent）：如果节点上有本地镜像，则使用本地镜像，本地不存在时拉取镜像。 • 总是拉取镜像（Always）（默认）：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 • 仅使用本地镜像（Never）：仅使用本地镜像，本地无镜像时拉取失败。

容器配额	配置容器资源的resources.limits和容器资源的resources.requests。如不填写申请值和限制值，表示不限制配额。申请值和限制值的配置说明及建议，请参见Requests and Limits。
	• 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，防止占用过多资源。
	• 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。
	• 加速卡资源申请：仅具有 GPU/NPU 计算型节点的集群，可配置 GPU/NPU 资源。开启之后按需选择设置 GPU 和 NPU。
	  • 注意：若需要支持GPU感知调度，以及GPU算力和显存的共享与隔离，集群中需安装CCE GPU Manager组件。若需要 NPU 感知调度，以及按算力和显存分配NPU资源，请先安装CCE NPU Manager 组件。
	  • 不开启：表示不使用GPU。

	• 独占GPU：单个容器独享GPU。
容器端口	设置容器端口名称、协议、端口。 • 容器端口的名称：仅用于区分端口，没有实际功能。 • 协议：容器端口使用的4层（传输层）协议，支持TCP和UDP。 • 设置容器对外暴露的端口：必须介于1~65535之间。容器只有对外暴露端口，才能实现对Pod外暴露以及Pod中容器互相通信。一个Pod中的所有容器共享Pod的网络协议栈，因此在Pod中配置多个容器时，端口不能重复。
环境变量	单击 添加环境变量，配置环境变量。Kubernetes 在创建 Pod 时，会将环境变量的信息注入到容器中。 支持以下类型： • 自定义：使用env直接在工作负载中硬编码环境变量，支持输入变量名称和变量值，创建自定义的环境变量。 • 配置项(ConfigMap)：使用envFrom获取存储在ConfigMap中的非敏感配置数据。支持部分引用或全部引用配置项中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。 • 保密字典（Secret）：使用envFrom获取存储在ConfigMap中的敏感信息，如密码、API 密钥等。支持部分引用或全部引用保密字典中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。
容器启动项	为容器添加启动项参数，当前支持以下容器启动项参数： • stdin：为容器开启标准输入，允许从外部源（如终端或其他输入流）接收输入。 • tty：为容器分配一个虚拟终端，允许您像在本地终端中一样向容器发送命令。一般需要同时开启stdin，表示将终端（tty）绑定到容器的标准输入（stdin）上，可以支持类似于`kubectl exec -i -t`命令的交互式操作，但区别在于该参数在Pod启动时就已经配置完成。
特权容器	默认不开启。开启后，特权模式允许容器拥有与所在的Worker节点操作系统近似的权限，例如访问硬件设备、挂载文件系统等。
初始化容器	默认不开启。选择容器是否作为初始化（Init）容器。初始化（Init）容器不支持设置健康检查。Init容器是一种特殊容器，可以在Pod中的其他应用容器启动之前运行。每个Pod中可以包含多个容器，同时Pod中也可以有一个或多个先于应用容器启动的Init容器，当所有的Init 容器运行完成时，Pod中的应用容器才会启动并运行。详细说明请参见 Init 容器。 • 注意：容器组中不能全为初始化容器，需要保证至少有一个非初始化容器。若容器组中有多个初始化容器，它们的运行顺序将与其创建顺序一致。

健康检查	支持配置**存活检查**、**就绪检查**和**启动检查**。允许基于HTTP请求检查、TCP端口检查、执行命令检查和GRCP检查的方式，探测容器中的负载是否正常。以下以配置存活检查为例，介绍配置参数。
	• HTTP请求检查：使用HTTP或HTTPS协议，向容器发送一个HTTP GET请求，通过检查响应确认容器状态。您需要进行如下检查配置：
	  • 检查协议：支持HTTP或HTTPS。
	  • 检查路径：访问HTTP Server的路径。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	  • Http请求头：HTTP请求中自定义的请求头（HTTP Headers）。单击添加请求头输入键值对的配置方式。
	• TCP端口检查：向容器发送一个TCP Socket，Kubelet将尝试在指定端口上打开容器的Socket。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。您需要进行如下检查配置：
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	• 执行命名检查：在容器内执行指定的命令，通过检查命令执行的结果确认容器状态。 您需要进行如下检查配置：
	  • 执行命令：检测容器健康情况的探测命令。多个命令换行分隔。
	• GRPC检查：使用gRPC协议，通过发送POST或GET请求检查服务器应用是否健康。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • GRPC服务：指定GPRC服务名称，不填为整体服务。

	• 公共参数说明：
	  • 启动延时：指容器启动后，第一次执行探测时需要等待的时间。
	  • 间隔时间：相邻两次检查的时间间隔。
	  • 响应超时：指发送检查请求后，等待响应的超时时间。
	  • 健康阈值：检查请求发送后，表示系统正常的响应次数。
	  • 不健康阈值：检查请求发送后，表示系统异常的无响应次数。

生命周期	在容器的生命周期的特定阶段配置需要执行的操作，例如启动命令、启动后处理和停止前处理，详情请参见设置容器生命周期。 • 启动命令：为容器设置预启动命令和参数。启动命令与参数定义了容器启动时执行的操作，用于初始化应用服务。适用于需要特定环境变量、挂载点或端口映射的应用部署场景。 • 启动后处理：为容器设置启动后的命令。启动后处理命令用于容器启动后执行特定任务，如初始化配置、运行脚本等，适用于需要在主进程前完成准备工作的场景。 • 停止前处理：为容器设置预结束命令。停止前处理命令用于关闭容器内的应用进程，确保数据一致性与服务正常终止。适用于需安全停机避免数据丢失或服务异常的场景。
数据卷	支持为容器挂载不同类型的数据卷进行数据持久化，支持通过subPath方式挂载卷的子路径；支持通过subPathExpr方式挂载卷的扩展路径： • 主机目录：使用主机中的目录创建数据卷并挂载。 • 临时目录：使用临时目录创建数据卷。 • 配置项：使用已有配置项ConfigMap创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `cm1:path1;cm2:path2`。 • 保密字典：使用已有保密字典Secret创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `sc1:path1;sc2:path2`。 • 存储卷声明：使用已有存储卷声明PVC创建数据卷。
仓库访问凭证	若有容器使用了私有镜像，请务必添加对应的镜像仓库的访问凭证。支持新建访问凭证和选择已有的，详细信息参考配置访问凭证。

7. 在高级设置模块中，配置调度策略，点击完成进行工作负载的创建。

参数	说明

调度策略	通过配置亲和与反亲和规则，可实现灵活的工作负载调度，支持负载亲和与节点亲和。
	• **节点亲和性**：配置工作负载和节点的亲和性调度。包括：强制满足和尽量满足。
	  • 不配置：不设置节点亲和策略。
	  • 强制调度：通过匹配标签进行调度选择。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	  • 尽量满足：通过匹配标签进行调度选择，允许配置权重。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	• **Pod亲和性**：选择Pod将与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **Pod 反亲和性**：选择Pod将不与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod反亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **容忍设置** ：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，详细信息请参考污点和容忍说明。

Pod标签	为该工作负载所属的每个Pod添加标签，对应 Kubernetes 中的 Label，用于识别Pod的键值对，配合选择器，帮助 Kubernetes 快速筛选目标Pod。详细信息参考K8S 标签说明。
Pod注释	为该工作负载所属的每个Pod添加注解，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

方式二：通过 YAML 创建

1. 点击部署列表左上角使用YAML创建，进入新建工作负载页面。
2. 点击下拉列表选择命名空间，填写 yaml 文件，点击确定。

> **说明：**
>
> - 模板类型：可使用示例模板或“我的模板”创建。
> - 复制：复制当前 YAML 文件的内容。
> - 另存为：将当前文件模版另存为“我的模版”。
> - 取消：返回列表页。
> - 确定：创建守护进程。

工作负载YAML示例

```
  apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-example
spec:
  minReadySeconds: 0
  selector:
    matchLabels:
      app: daemonset-example
  revisionHistoryLimit: 10
  template:
    metadata:
      labels:
        app: daemonset-example
    spec:
      containers:
        - name: daemonset-example
          image: registry.baidubce.com/cce/busybox:latest
          command:
          - sleep
          - "3600"
  updateStrategy:
    type: RollingUpdate
```

方式三：通过 kubectl 创建

1. 准备 Daemonset yaml 文件。
2. 安装 kubectl，并连接集群，详情参见 通过 kubectl 连接集群。
3. 复制下方的YAML文件，并保存到 daemonset.yaml中。

> 注意：
>
> - 创建工作负载前，请确保已通过kubectl连接到集群。具体操作，请参见通过 kubectl 连接集群。

```
  apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-example
spec:
  minReadySeconds: 0
  selector:
    matchLabels:
      app: daemonset-example
  revisionHistoryLimit: 10
  template:
    metadata:
      labels:
        app: daemonset-example
    spec:
      containers:
        - name: daemonset-example
          image: registry.baidubce.com/cce/busybox:latest
          command:
          - sleep
          - "3600"
  updateStrategy:
    type: RollingUpdate
```

4. 执行以下命令创建 DaemonSet。

```
kubectl apply -f daemonset.yaml
```

预期输出

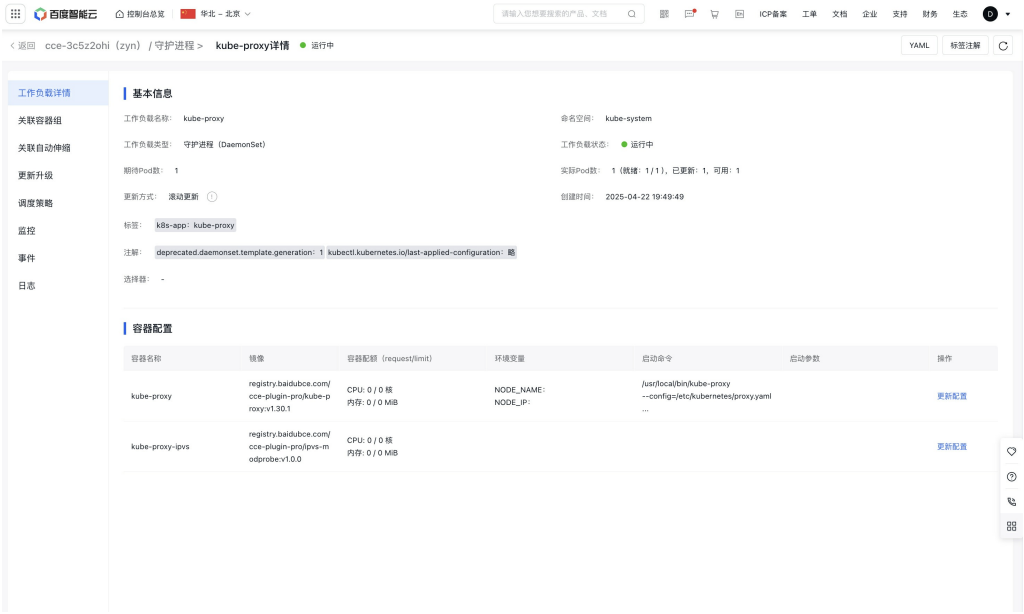
```
daemonset.apps/daemonset-example created
```

5. 执行以下命令，查看DaemonSet Pod的详细信息。

```
kubectl get pods --all-namespaces -o wide | grep daemonset-example
```

查看守护进程集详情

点击列表页任何一行的工作负载名称，即可进入工作负载详情页，详情页内容包括： 工作负载详情、关联容器组、自动伸缩、更新升级、调度策略、监控、事件等信息。



更新守护进程集

支持通过控制台界面或 YAML 文件编辑已创建的守护进程集。

1. 登录守护进程集负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 工作负载 > 守护进程集。
2. 在目标守护进程集的操作列，单击 **更新升级** 或 **编辑 YAML**。

> 说明：
> 通过控制台界面更新和编辑 YAML 文件更新，均不支持更新守护进程集名称、命名空间。

   a. 更新：在控制台更新配置信息。详细的参数说明，请参见 通过控制台创建。
   b. 编辑 YAML：编辑 YAML 文件更新配置信息。详细的 YAML 配置说明，请参见通过 YAML 创建。
3. 更新完成后，单击**更新**。

标签注解管理

标签是以key/value键值对的形式附加在守护进程集工作负载上的。添加标签后，可通过标签对守护进程集工作负载进行管理和选择。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。

> **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，详细 K8S标签说明。

1. 登录守护进程集管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。

c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **守护进程集**。

2. 选择“**守护进程集**”页签，单击工作负载后的“**更多** > **标签注解**”。

3. 单击**添加标签**，输入键和值后单击**更新**。

查看监控

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。

> **说明**：
> 查看工作负载的监控信息时，依赖集群接入 Prometheus 监控服务。若集群未接入Prometheues 监控服务，请根据界面提示接入，详细信息请参见使用Prometheus监控集群。

1. 登录守护进程集负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **守护进程集**。
2. 在目标守护进程集负载的列表，点击需要调整的守护进程集行中的“**监控**”。
3. 在监控页面查看守护进程集的监控数据。容器相关监控指标说明，请参见 监控指标说明。

查看日志

您可以通过“日志”功能查看守护进程集工作负载的日志信息。

> **说明**：
> 当前显示的日志内容为容器标准输出日志，不具备持久化和高阶运维能力，如需使用更完善的日志能力，可使用日志中心功能。

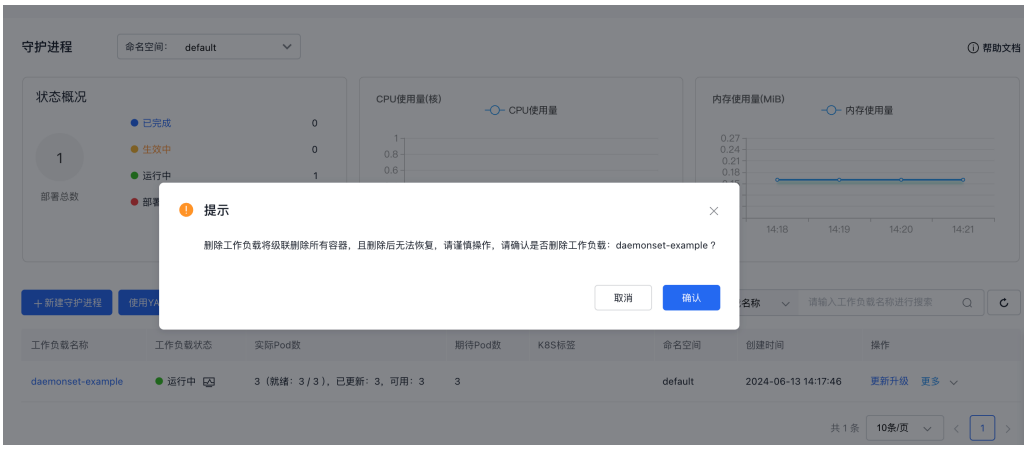
1. 登录守护进程集负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **守护进程集**。
2. 在目标守护进程集工作负载的列表，单击工作负载后的“**更多** > **日志**”。
3. 在日志页面查看容器日志数据。

删除守护进程集

若工作负载无需再使用，您可以将工作负载删除。

> **说明**：
> 删除守护进程集负载可能导致服务不可用，请谨慎操作。

1. 登录守护进程集负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **守护进程集**。
2. 在目标守护进程集负载的列表，单击工作负载后的“**更多** > **删除**”。
3. 在系统弹出的二次确认对话框，确认信息后，点击**确认**。



Statefulset管理

概述

在运行中会保存数据或状态的工作负载称为有状态负载 ，对应 Kubernetes 中的 StatefulSet，例如 MySQL，它在运行中会保存产生的数据。用户可以通过表单方式基于镜像创建有状态部署，也可以通过百度智能云提供的YAML文件模板创建和修改有状态部署，对有状态部署关联的容器组（Pod）、事件等进行控制管理。

StatefulSet 介绍

与Deployment相似，StatefulSet也会保证有既定数量的Pod在正常运行。但不同的是，StatefulSet会通过以下措施保留Pod的身份：

- 有序、固定的名称：StatefulSet的每个Pod会按顺序命名为<StatefulSet名称>-<序号>。例如，StatefulSet的名称为db-app，则Pod名称为db-app-0、db-app-1。删除并重建Pod时，新建Pod会继承原有的名称。
- 稳定的网络标识：StatefulSet通常需要关联一个Headless Service（通过spec.serviceName指定）。这个Headless Service并不像ClusterIP提供Pod的负载均衡，而仅用于为Pod提供固定的域名（对Headless Service进行DNS查询时，DNS会返回所有匹配的Pod的IP地址）。指定Headless Service后，Pod的域名格式为<Pod名称>.<Headless Service名称>..svc.，例如db-app-01.db-app.default.svc.cluster.local。Pod重建后，域名会自动被解析到新Pod。
- 稳定的持久化存储：在StatefulSet中，可以指定PVC模板（通过spec.volumeClaimTemplates指定）。StatefulSet会根据此模板为每个Pod生成独立的PersistentVolumeClaim（PVC），PVC名称为<PVC模板名称>-<Pod名称>。当Pod被删除时，PVC会被保留并自动关联到使用相同序号的新Pod。

以上的这些措施保证了StatefulSet的Pod在重建后能继承原有的网络和存储状态，使应用从持久化数据中恢复运行。关于StatefulSet的更多信息，请参见官方文档。

前提条件

- 已成功创建 CCE 集群，操作说明参见 创建 CCE 集群。

创建有状态工作负载 StatefulSet

用户可以选择通过 CCE 控制台使用表单形式或者通过 YAML 形式创建新的有状态工作负载，也可以选择通过 kubectl 命令创建。

通过控制台表单形式创建

1. 登录 容器引擎CCE控制台。
2. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
3. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
4. 点击无状态工作负载列表左上角**新建有状态**，进入创建工作负载页面。
5. 在基本信息模块完成相关配置，点击**下一步**进入容器配置。

参数	说明
工作负载名称	根据系统提示，配置工作负载的名称。在同一个命名空间下，名称必须唯一。
命名空间	选择工作负载部署的命名空间，默认为default。支持选择自定义创建的命名空间，自定义命名空间，请参见 命名空间操作。
工作负载类型	选择有状态StatefulSet。
期望 Pod 数	输入工作负载的 Pod 实例个数。
K8s标签	指定工作负载的标签，对应 Kubernetes 中的 Label，用于识别工作负载的键值对，配合选择器，帮助 Kubernetes 快速筛选目标工作负载。详细信息参考K8S 标签说明。
注释	指定工作负载的注解信息，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

6. 在容器配置模块，填写容器名称和镜像地址，可点击选择镜像后在弹窗中选择 CCR 中的镜像地址。点击下一步进入高级配置模块。

> **说明**
>
> - 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。
> - 若使用了 CCR 私有镜像,还需在当前页面的高级设置中配置仓库访问凭证。

参数	说明
容器名称	根据系统提示的命名规则，输入容器名称。同一个工作负载下容器名称必须唯一。
镜像地址	单击后方“选择镜像”，选择容器使用的镜像。支持选择以下两种种类型的镜像。 • 容器镜像企业版 ：可以选择托管在容器镜像服务 CCR 中的企业版镜像。关于 CCR 的更多信息，详细信息请参考 CCR 企业版快速入门。 • 容器镜像个人版：可以选择托管在容器镜像服务CCR中的个人版镜像。关于 CCR 的更多信息，详细信息请参考CCR 个人版说明。
镜像版本	选择需要部署的镜像版本。
镜像拉取策略	选择镜像拉取策略。CCE 支持以下三种镜像拉取策略（imagePullPolicy）： • 优先使用本地镜像（IfNotPresent）：如果节点上有本地镜像，则使用本地镜像，本地不存在时拉取镜像。 • 总是拉取镜像（Always）（默认）：表示每次部署或扩容都会从容器镜像服务重新拉取镜像，而不会从本地拉取镜像。 • 仅使用本地镜像（Never）：仅使用本地镜像，本地无镜像时拉取失败。

容器配额	配置容器资源的resources.limits和容器资源的resources.requests。如不填写申请值和限制值，表示不限制配额。申请值和限制值的配置说明及建议，请参见Requests and Limits。
	• 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，防止占用过多资源。
	• 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。
	• 加速卡资源申请：仅具有 GPU/NPU 计算型节点的集群，可配置 GPU/NPU 资源。开启之后按需选择设置 GPU 和 NPU。
	  • 注意：若需要支持GPU感知调度，以及GPU算力和显存的共享与隔离，集群中需安装CCE GPU Manager组件。若需要 NPU 感知调度，以及按算力和显存分配NPU资源，请先安装CCE NPU Manager 组件。
	  • 不开启：表示不使用GPU。

	• 独占GPU：单个容器独享GPU。
容器端口	设置容器端口名称、协议、端口。 • 容器端口的名称：仅用于区分端口，没有实际功能。 • 协议：容器端口使用的4层（传输层）协议，支持TCP和UDP。 • 设置容器对外暴露的端口：必须介于1~65535之间。容器只有对外暴露端口，才能实现对Pod外暴露以及Pod中容器互相通信。一个Pod中的所有容器共享Pod的网络协议栈，因此在Pod中配置多个容器时，端口不能重复。
环境变量	单击 添加环境变量，配置环境变量。Kubernetes 在创建 Pod 时，会将环境变量的信息注入到容器中。 支持以下类型： • 自定义：使用env直接在工作负载中硬编码环境变量，支持输入变量名称和变量值，创建自定义的环境变量。 • 配置项(ConfigMap)：使用envFrom获取存储在ConfigMap中的非敏感配置数据。支持部分引用或全部引用配置项中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。 • 保密字典（Secret）：使用envFrom获取存储在ConfigMap中的敏感信息，如密码、API 密钥等。支持部分引用或全部引用保密字典中的变量。部分引用时，一条规则仅支持为一个 Key 创建环境变量。
容器启动项	为容器添加启动项参数，当前支持以下容器启动项参数： • stdin：为容器开启标准输入，允许从外部源（如终端或其他输入流）接收输入。 • tty：为容器分配一个虚拟终端，允许您像在本地终端中一样向容器发送命令。一般需要同时开启stdin，表示将终端（tty）绑定到容器的标准输入（stdin）上，可以支持类似于`kubectl exec -i -t`命令的交互式操作，但区别在于该参数在Pod启动时就已经配置完成。
特权容器	默认不开启。开启后，特权模式允许容器拥有与所在的Worker节点操作系统近似的权限，例如访问硬件设备、挂载文件系统等。
初始化容器	默认不开启。选择容器是否作为初始化（Init）容器。初始化（Init）容器不支持设置健康检查。Init容器是一种特殊容器，可以在Pod中的其他应用容器启动之前运行。每个Pod中可以包含多个容器，同时Pod中也可以有一个或多个先于应用容器启动的Init容器，当所有的Init 容器运行完成时，Pod中的应用容器才会启动并运行。详细说明请参见 Init 容器。 • 注意：容器组中不能全为初始化容器，需要保证至少有一个非初始化容器。若容器组中有多个初始化容器，它们的运行顺序将与其创建顺序一致。

健康检查	支持配置**存活检查**、**就绪检查**和**启动检查**。允许基于HTTP请求检查、TCP端口检查、执行命令检查和GRCP检查的方式，探测容器中的负载是否正常。以下以配置存活检查为例，介绍配置参数。
	• HTTP请求检查：使用HTTP或HTTPS协议，向容器发送一个HTTP GET请求，通过检查响应确认容器状态。您需要进行如下检查配置：
	  • 检查协议：支持HTTP或HTTPS。
	  • 检查路径：访问HTTP Server的路径。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	  • Http请求头：HTTP请求中自定义的请求头（HTTP Headers）。单击添加请求头输入键值对的配置方式。
	• TCP端口检查：向容器发送一个TCP Socket，Kubelet将尝试在指定端口上打开容器的Socket。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。您需要进行如下检查配置：
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • 主机地址：请求的主机地址，不填时默认为Pod实例的IP地址。
	• 执行命名检查：在容器内执行指定的命令，通过检查命令执行的结果确认容器状态。 您需要进行如下检查配置：
	  • 执行命令：检测容器健康情况的探测命令。多个命令换行分隔。
	• GRPC检查：使用gRPC协议，通过发送POST或GET请求检查服务器应用是否健康。
	  • 检查端口：容器暴露的访问端口号或端口名称。
	  • GRPC服务：指定GPRC服务名称，不填为整体服务。

	• 公共参数说明：
	  • 启动延时：指容器启动后，第一次执行探测时需要等待的时间。
	  • 间隔时间：相邻两次检查的时间间隔。
	  • 响应超时：指发送检查请求后，等待响应的超时时间。
	  • 健康阈值：检查请求发送后，表示系统正常的响应次数。
	  • 不健康阈值：检查请求发送后，表示系统异常的无响应次数。

生命周期	在容器的生命周期的特定阶段配置需要执行的操作，例如启动命令、启动后处理和停止前处理，详情请参见设置容器生命周期。 • 启动命令：为容器设置预启动命令和参数。启动命令与参数定义了容器启动时执行的操作，用于初始化应用服务。适用于需要特定环境变量、挂载点或端口映射的应用部署场景。 • 启动后处理：为容器设置启动后的命令。启动后处理命令用于容器启动后执行特定任务，如初始化配置、运行脚本等，适用于需要在主进程前完成准备工作的场景。 • 停止前处理：为容器设置预结束命令。停止前处理命令用于关闭容器内的应用进程，确保数据一致性与服务正常终止。适用于需安全停机避免数据丢失或服务异常的场景。
数据卷	支持为容器挂载不同类型的数据卷进行数据持久化，支持通过subPath方式挂载卷的子路径；支持通过subPathExpr方式挂载卷的扩展路径： • 主机目录：使用主机中的目录创建数据卷并挂载。 • 临时目录：使用临时目录创建数据卷。 • 配置项：使用已有配置项ConfigMap创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `cm1:path1;cm2:path2`。 • 保密字典：使用已有保密字典Secret创建数据卷，支持全部挂载和部分挂载，指定部分数据挂载时，支持配置多数据item，用半角分号（;）分隔。例如 `sc1:path1;sc2:path2`。 • 存储卷声明：使用已有存储卷声明PVC创建数据卷。
仓库访问凭证	若有容器使用了私有镜像，请务必添加对应的镜像仓库的访问凭证。支持新建访问凭证和选择已有的，详细信息参考配置访问凭证。

7. 在高级设置模块中，配置调度策略，点击完成进行工作负载的创建。

参数	说明
伸缩配置	支持两种伸缩配置：水平伸缩和定时水平伸缩。 • **水平伸缩**：通过监测容器的性能指标触发自动伸缩。指标伸缩可以帮助您在业务负载出现波动时自动调整工作负载使用的资源总量，在高负载时扩容缓解压力，低负载时缩容节省资源。更多信息，请参见使用容器水平伸缩（HPA）。 • **定时水平伸缩**：通过定时触发工作负载伸缩，适用于业务负载存在周期性变化的场景，例如社交媒体在午餐后和晚餐后周期性的流量高峰。更多信息，请参见使用容器定时水平伸缩（CronHPA）。 • 注意：使用定时水平伸缩 CronHPA 需要安装CCE CronHPA Controller组件，详细参考 CCE CronHPA Controller介绍。

调度策略	通过配置亲和与反亲和规则，可实现灵活的工作负载调度，支持负载亲和与节点亲和。
	• **节点亲和性**：配置工作负载和节点的亲和性调度。包括：强制满足和尽量满足。
	  • 不配置：不设置节点亲和策略。
	  • 强制调度：通过匹配标签进行调度选择。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	  • 尽量满足：通过匹配标签进行调度选择，允许配置权重。可以设置 6 种匹配关系（In, NotIn, Exists, DoesNotExist, gt, lt）。
	• **Pod亲和性**：选择Pod将与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **Pod 反亲和性**：选择Pod将不与那些Pod部署在同一拓扑域，设置强制满足/尽量满足（Required/Preferred）条件，多条规则中的标签必须同时在一个工作负载中。包括：强制满足和尽量满足。
	  • 不配置：不设置Pod反亲和策略。
	  • 强制满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	  • 尽量满足：基于命名空间、拓扑域。支持配置选择器（MatchExpressions、MatchLabels）。通过匹配标签进行调度选择，允许配置权重。可以设置 4 种匹配关系（In, NotIn, Exists, DoesNotExist）。
	• **容忍设置** ：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，详细信息请参考污点和容忍说明。

Pod标签	为该工作负载所属的每个Pod添加标签，对应 Kubernetes 中的 Label，用于识别Pod的键值对，配合选择器，帮助 Kubernetes 快速筛选目标Pod。详细信息参考K8S 标签说明。
Pod注释	为该工作负载所属的每个Pod添加注解，对应 Kubernetes 中的 Annotation，主要记录用于应用部署、安全策略、调度策略的附加信息，不用于标识和区分资源。详细信息参考K8S 注释说明。

方式二：通过 YAML 创建

1. 点击部署列表左上角**使用YAML创建**，进入新建工作负载页面。
2. 点击下拉列表选择命名空间，填写 yaml 文件，点击**确定**。

> **说明：**
>
> • 模板类型：可使用示例模板或“我的模板”创建。
>
> • 复制：复制当前 YAML 文件的内容。
>
> • 另存为：将当前文件模版另存为“我的模版”。
>
> • 取消：返回列表页。

工作负载YAML示例

```
 apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: statefulset-example
spec:
 serviceName: "nginx"
 replicas: 2
 selector:
   matchLabels:
     app: nginx # has to match .spec.template.metadata.labels
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: registry.baidubce.com/cce/nginx-alpine-go:latest
       livenessProbe:
         httpGet:
           path: /
           port: 80
         initialDelaySeconds: 20
         timeoutSeconds: 5
         periodSeconds: 5
       readinessProbe:
         httpGet:
           path: /
           port: 80
         initialDelaySeconds: 5
         timeoutSeconds: 1
         periodSeconds: 5
       ports:
       - containerPort: 80
         name: web
     updateStrategy:
       type: RollingUpdate
```

方式三：通过 kubectl 创建

1. 准备 StatefulSet yaml 文件。
2. 安装 kubectl，并连接集群，详情参见 通过 kubectl 连接集群。
3. 复制下方的YAML文件，并保存到 statefulset.yaml中。

```
  apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-example
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: registry.baidubce.com/cce/nginx-alpine-go:latest
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 20
          timeoutSeconds: 5
          periodSeconds: 5
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 5
        ports:
        - containerPort: 80
          name: web
      updateStrategy:
        type: RollingUpdate
```

4. 执行以下命令，创建 StatefulSet。

   ```
   kubectl apply -f statefulset.yaml
   ```

   预期输出：

   ```
   service/nginx created
   statefulset.apps/statefulset-example created
   ```

5. 执行以下命令，查看Service的公网IP地址

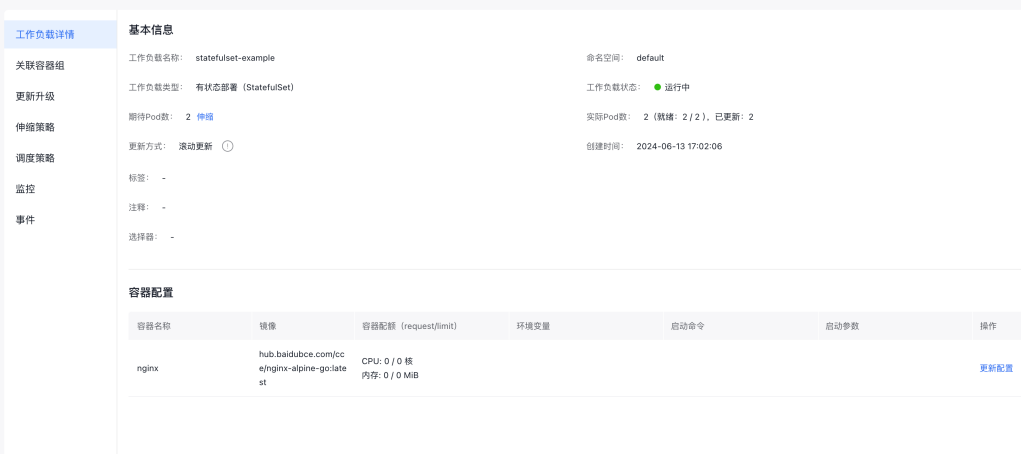
   ```
   kubectl get svc
   ```

6. 执行以下命令，验证 StatefulSet 是否创建成功：

   ```
   kubectl get statefulset
   ```

查看有状态部署详情

点击列表页任何一行的工作负载名称，即可进入工作负载详情页，详情页内容包括： 工作负载详情、关联容器组、更新升级、伸缩策略、调度策略、监控、事件等信息。



更新 Pod 数

支持通过控制台快速更新无状态负载中的 Pod 实例个数。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
2. 在目标有状态负载的列表，点击需要调整的有状态行中的“**伸缩**”，在伸缩的弹框页中，填写需要的容器组数量，点击“确认”。

更新有状态工作负载

支持通过控制台界面或 YAML 文件编辑已创建的有状态负载。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 工作负载 > 有状态负载。
2. 在目标有状态负载的操作列，单击 **更新升级** 或 **编辑 YAML**。

> 说明：
> 通过控制台界面更新和编辑 YAML 文件更新，均不支持更新有状态负载名称、命名空间。

a. 更新：在控制台更新配置信息。详细的参数说明，请参见 [通过控制台创建](https://cloud.baidu.com/doc/CCE/s/cjxpoxrgi)。
b. 编辑 YAML：编辑 YAML 文件更新配置信息。详细的 YAML 配置说明，请参见[通过 YAML 创建](https://cloud.baidu.com/doc/CCE/s/cjxpoxrgi)。

3. 更新完成后，单击**提交**。

标签注解管理

标签是以key/value键值对的形式附加在有状态工作负载上的。添加标签后，可通过标签对有状态工作负载进行管理和选择。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。

> **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，详细 K8S标签说明。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
2. 选择“**有状态**”页签，单击工作负载后的“**更多** > **标签注解**”。
3. 单击**添加标签**，输入键和值后单击**更新**。

查看监控

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。

> **说明**：
> 查看工作负载的监控信息时，依赖集群接入 Prometheus 监控服务。若集群未接入Prometheues 监控服务，请根据界面提示接入，详细信息请参见使用Prometheus监控集群。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
2. 在目标有状态负载的列表，点击需要调整的有状态行中的“**监控**”。
3. 在监控页面查看有状态负载的监控数据。容器相关监控指标说明，请参见 监控指标说明。

查看日志

您可以通过“日志”功能查看无状态工作负载的日志信息。

> **说明**：
> 当前显示的日志内容为容器标准输出日志，不具备持久化和高阶运维能力，如需使用更完善的日志能力，可使用日志中心功能。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
2. 在目标有状态负载的列表，单击工作负载后的“**更多** > **日志**”。
3. 在日志页面查看容器日志数据。

删除有状态负载

若工作负载无需再使用，您可以将工作负载删除。

> **说明**：
> 删除有状态负载可能导致服务不可用，请谨慎操作。

1. 登录有状态负载管理页面。
   a. 登录 容器引擎CCE控制台。
   b. 在左侧导航栏，点击 "集群管理->集群列表"，进入集群列表页，点击集群名称，进入集群管理页面。
   c. 在集群管理页面的左侧导航栏中，单击 **工作负载** > **有状态**。
2. 在目标有状态负载的列表，单击工作负载后的“**更多** > **删除**”。
3. 在系统弹出的二次确认对话框，确认信息后，点击**确认**。

Job管理

用户可以通过表单方式基于镜像创建普通任务，也可以通过百度智能云提供的YAML文件模板创建和修改普通任务，对已普通任务关联的容器组（Pod）、事件等进行控制管理。

Job 简介

Job负责批量处理短暂的一次性任务，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。 Kubernetes支持以下几种Job：

- 非并行 Job：通常创建一个 Pod 直至其成功结束
- 固定结束次数的 Job：设置 spec.completions，创建多个 Pod，直到 spec.completions 个 Pod 成功时结束运行
- 并行 Job：设置 spec.Parallelism 但不设置 spec.completions，创建 spec.Parallelism 个 Pod，当所有 Pod 结束且至少一个成功时，Job 就认为是成功的

典型使用场景

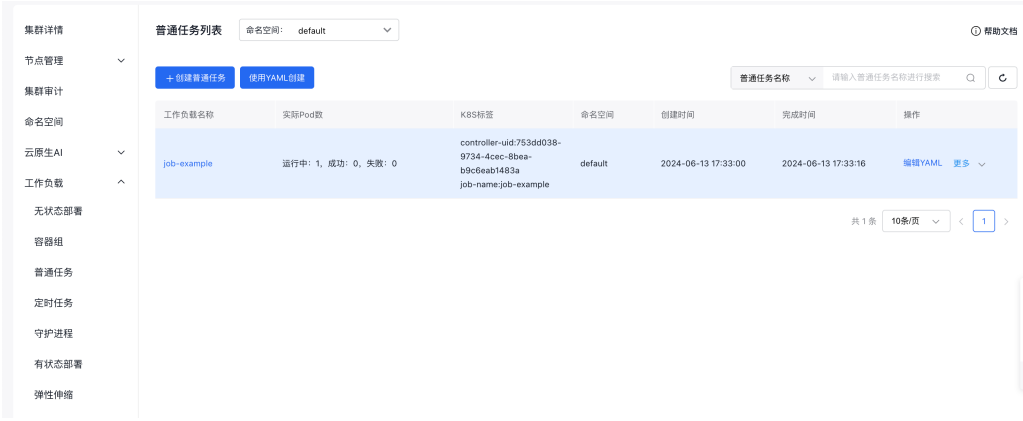
根据 spec.Parallelism 和 spec.completions 的设置，可以将 Job 划分为以下几种模式：

Job 类型	使用示例	行为	completions	Parallelism
一次性 Job	数据库迁移	创建一个 Pod 直至其成功	1	1
固定结束次数的 Job	处理工作队列的 Pod	一次创建一个 Pod 运行直至 completions 个成功时结束	2+	1
固定结束次数的并行 Job	多个 Pod 同时处理工作队列	依次创建多个 Pod 运行直至 completions 个成功时结束	2+	2+
并行 Job	多个 Pod 同时处理工作队列	创建一个或多个 Pod 直至有一个成功时结束	1	2+

查看普通任务列表

进入 "产品服务>容器引擎CCE"，点击左侧导航栏 "工作负载>普通任务"，进入普通任务页面。内容包括：

- 列表数据：显示所有普通任务的列表信息，及新建、删除、修改

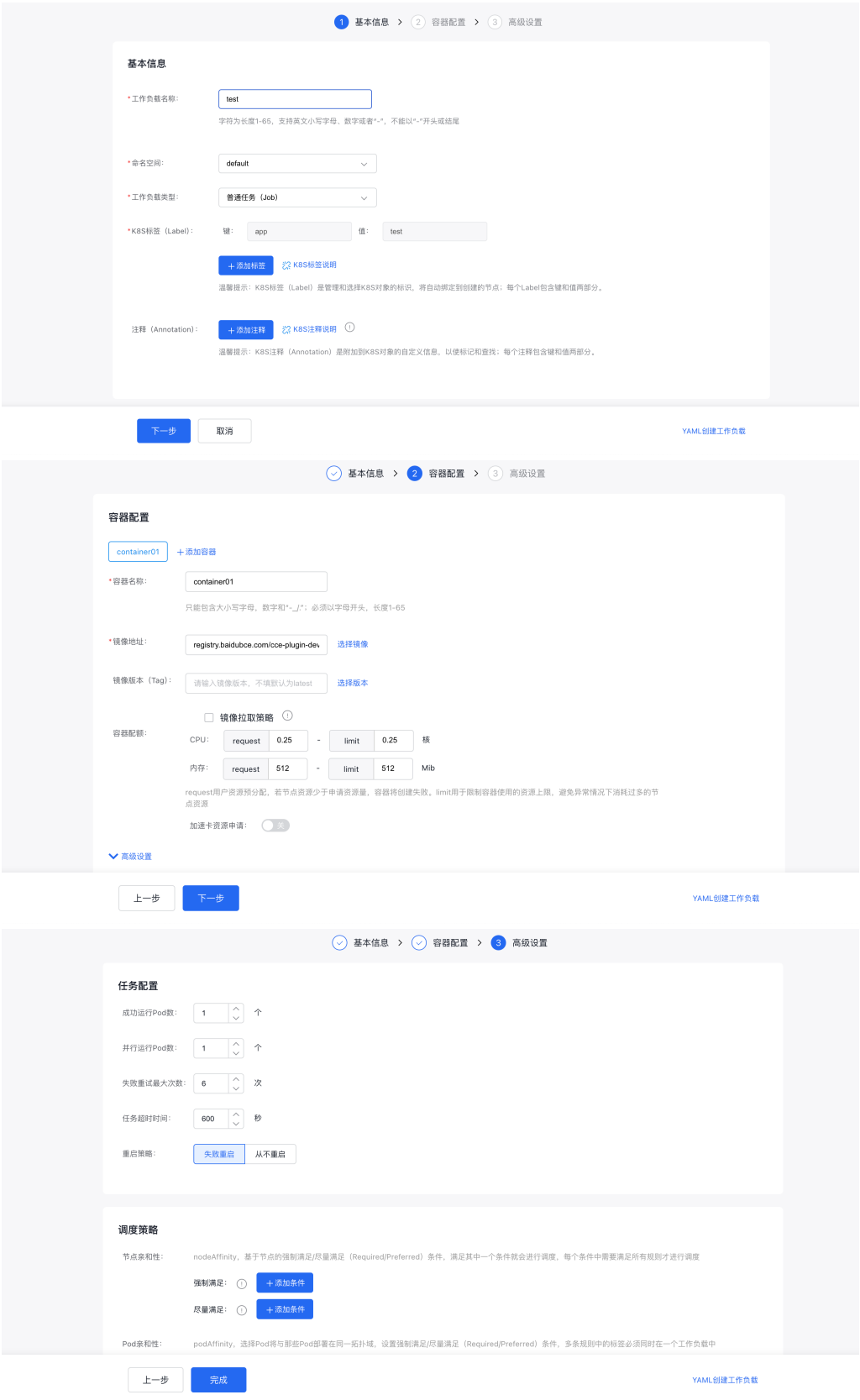


新建普通任务

点击 "新建普通任务"，用户可以选择通过表单形式或者通过 yaml 形式创建新的普通任务。

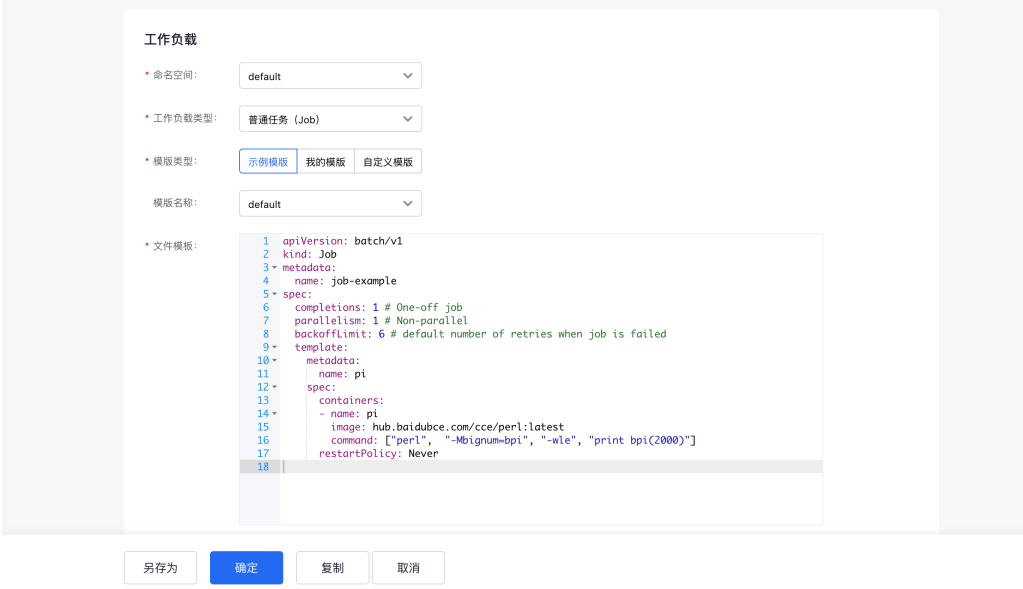
（1）通过表单形式创建：

- 填写普通任务名称、命名空间和标签；
- 在第二页容器配置页面，填入容器名称、镜像地址和容器配置信息；
- 在第三页高级设置页面，设置任务配置、调度策略，完成创建。



（2）通过 yaml 形式创建：

- 选择命名空间，填写 yaml 文件，点击确定。
- 模板类型：可使用示例模板或我的模板创建。
- 复制：复制当前 yaml 文件的内容
- 确定：进行Job的创建



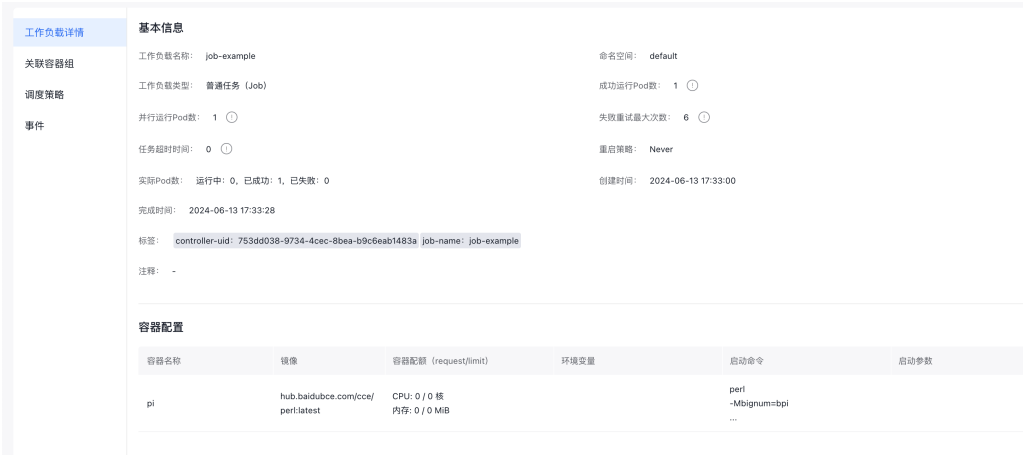
字段解释

- **completions**：标志 Job 结束时需要成功运行的 Pod 的个数，默认为 1
- **parallelism**：表示并行运行的 Pod 的个数，默认为 1
- **backoffLimit**：表示当 Job 失败时，重试的次数。默认为 6

查看普通任务详情

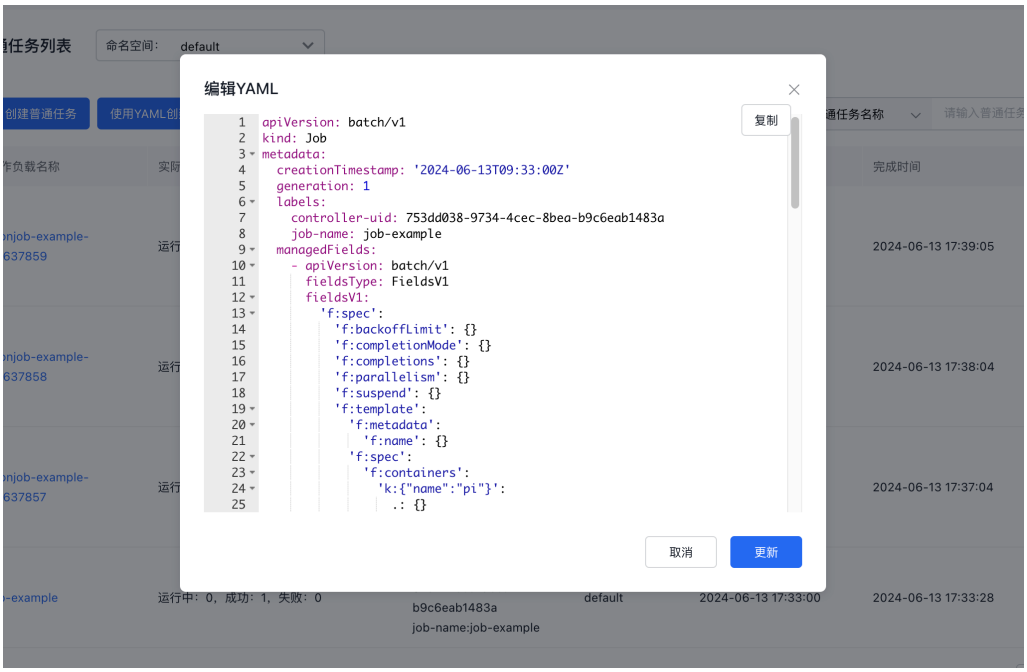
点击列表页任何一行的普通任务名称，即可进入普通任务详情页，详情页内容包括：

- 工作负载详情、关联容器组、调度策略、事件



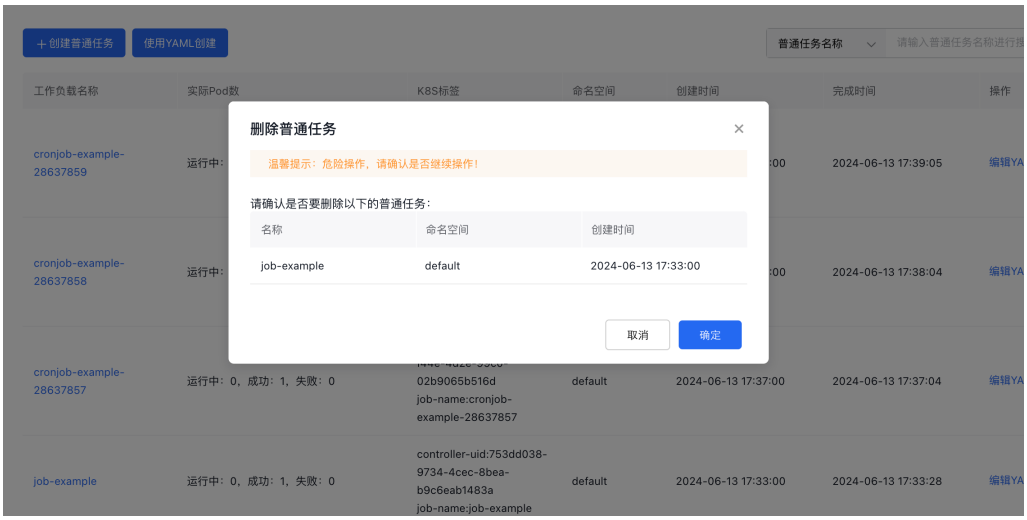
修改普通任务

在普通任务列表页面，点击需要修改的普通任务行中的“编辑YAML”按钮，完成修改后，点击“更新”提交。



删除普通任务

在普通任务列表页面，选择要删除的普通任务，点击“更多”按钮，选择”删除“，确认信息后，点击“确定”。



CronJob管理

用户可以通过表单方式基于镜像创建定时任务，也可以通过百度智能云提供的 YAML 文件模板创建和修改定时任务，对定时任务关联的容器组（Pod）、事件等进行控制管理。

CronJob 简介

Cron Job 类似于 Linux 系统中的 CronTab，在指定的时间周期运行指定的任务(job)，即：

- 在给定时间点运行一次
- 在给定时间点周期性地运行

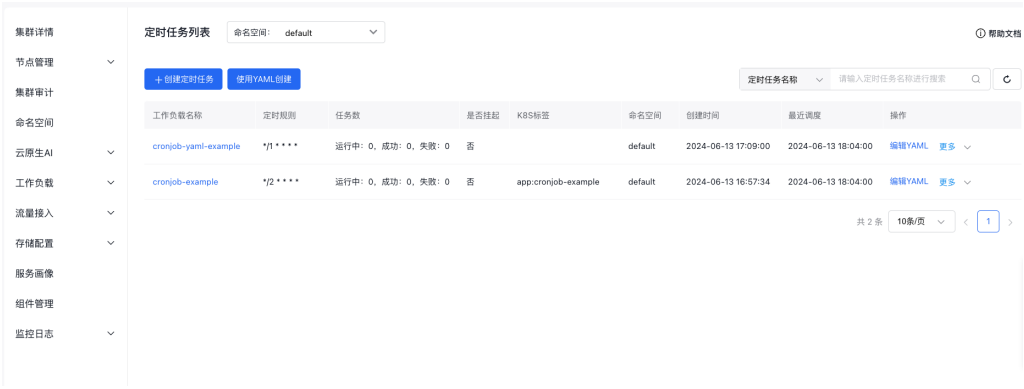
典型使用场景：

- 在给定的时间点调度 Job 运行，例如：5 min 后清理日志文件等
- 创建周期性运行的 Job，例如：每隔 1 h 进行数据库备份等

查看定时任务列表

进入 "产品服务>容器引擎CCE"，选择集群，点击 "工作负载>定时任务"，进入定时任务页面。 默认选择最新的CCE集群，然后选择该集群下的”default“命名空间，加载该命名空间下的所有定时任务，内容包括：

- 列表数据：显示所有定时任务的列表信息，及新建、删除、修改



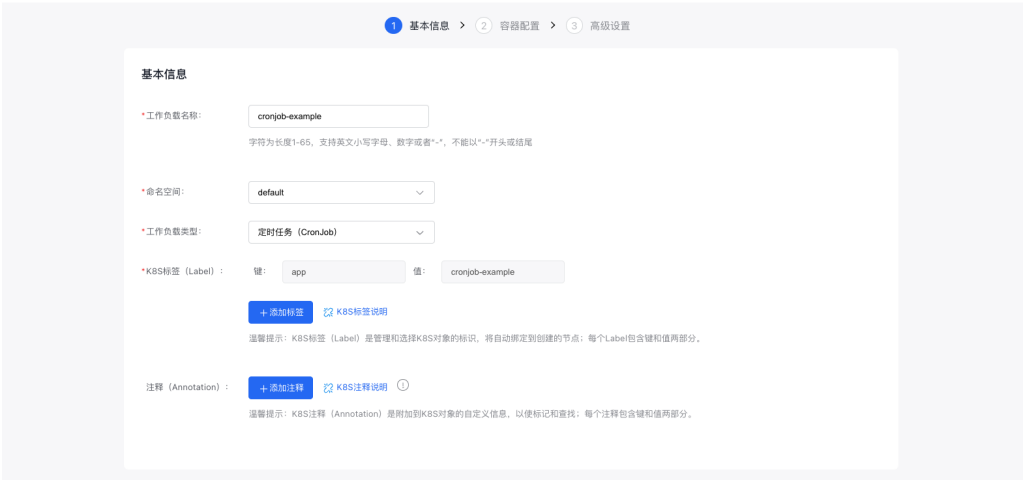
新建定时任务

用户可以选择通过表单形式或者通过 yaml 形式创建新的定时任务。

（1）通过表单形式创建：

- 点击 "+创建定时任务"；
- 基本信息：填写工作负载名称，选择命名空间、工作负载类型，按需添加K8S标签和注释；
- 容器配置：填写容器名称，选择镜像地址、版本，调整容器配额，申请加速卡资源点击加速卡资源申请开关并配置，新增容器配置点击“+添加容器”；
- 高级设置-任务配置：配置定时规则、定时任务并发策略、定时超时时间、重试最大次数、任务超时时间、重启策略等；
- 高级设置-调度策略：定制pod和节点的调度策略；
- 点击“完成”，返回创建成功信息，完成创建。

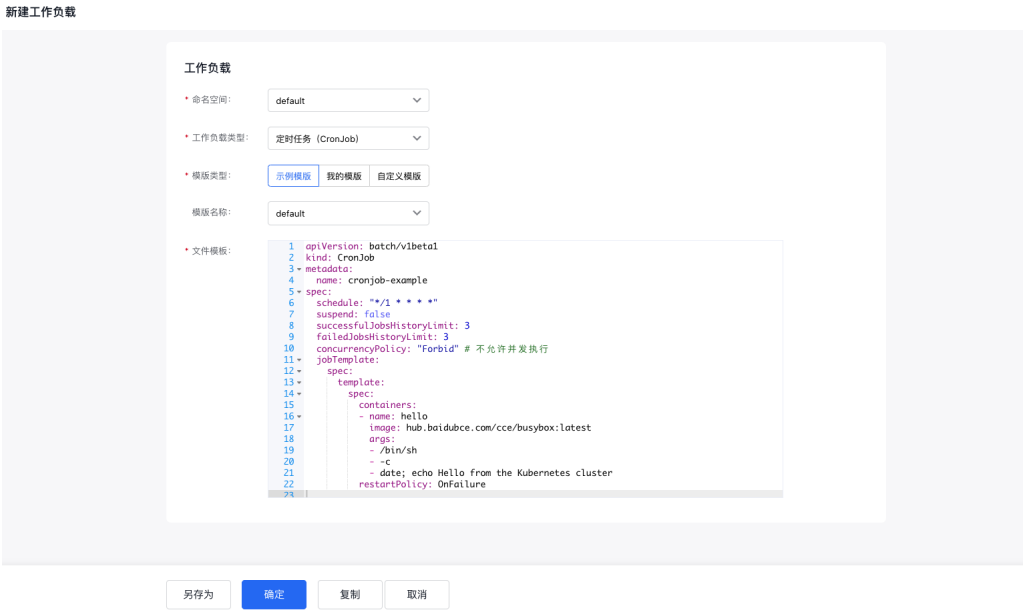
创建工作负载



创建工作负载

（2）通过 yaml 形式创建：

- 选择命名空间，填写 yaml 文件，点击确定。
- 模板类型：可使用示例模板或我的模板创建。
- 复制：复制当前 yaml 文件的内容。
- 取消：返回列表页。



字段解释 调度： spec.schedule 字段，是 cronjob 的必需的字段，值为 Cron 格式的字符串，例如：0 * * * *,或者 @hourly ，kubernetes 会按照该字段规定的调度时间创建并运行 Job

挂起： spec.suspend 字段，可选字段，默认值为 false。如果设置为 true，后续所有 Job 的执行将被挂起

并发策略： spec.concurrencyPolicy 字段：可选字段，默认值为 Allow。指定被 CronJob 创建的 Job 如何并发执行。只允许指定以下策略的一种：

- Allow：允许并发运行 Job，如果前一个还没有完成，到了当前 Job 该运行时，运行当前 Job
- Forbid：禁止并发运行，如果前一个还没有完成，到了当前 Job 该运行时，则直接跳过
- Replace：取消当前正在运行的 Job，用一个新的 Job 来替换

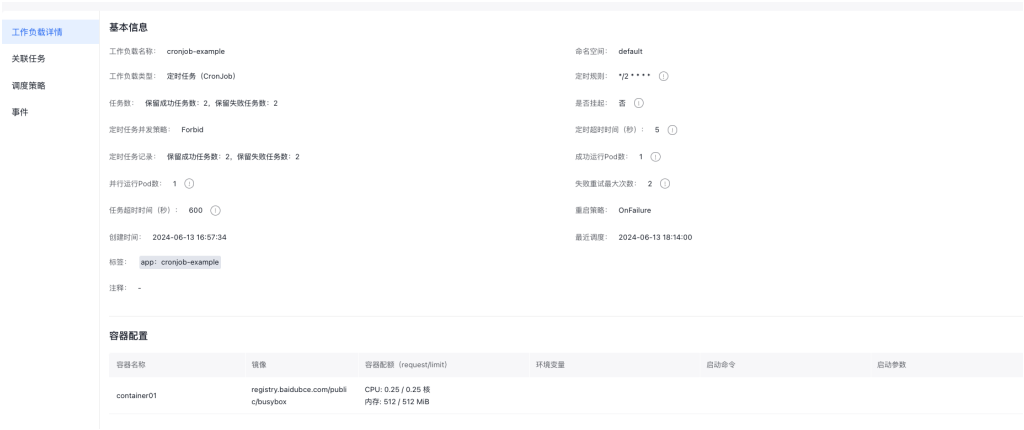
Job 历史限制 .spec.successfulJobsHistoryLimit 和 .spec.failedJobsHistoryLimit 这两个字段是可选的。它们指定了可以保留完成和失败 Job 数量的限制。

默认没有限制，所有成功和失败的 Job 都会被保留。然而，当运行一个 Cron Job 时，很快就会堆积很多 Job，推荐设置这两个字段的值。设置限制值为 0，相关类型的 Job 完成后将不会被保留。

查看定时任务详情

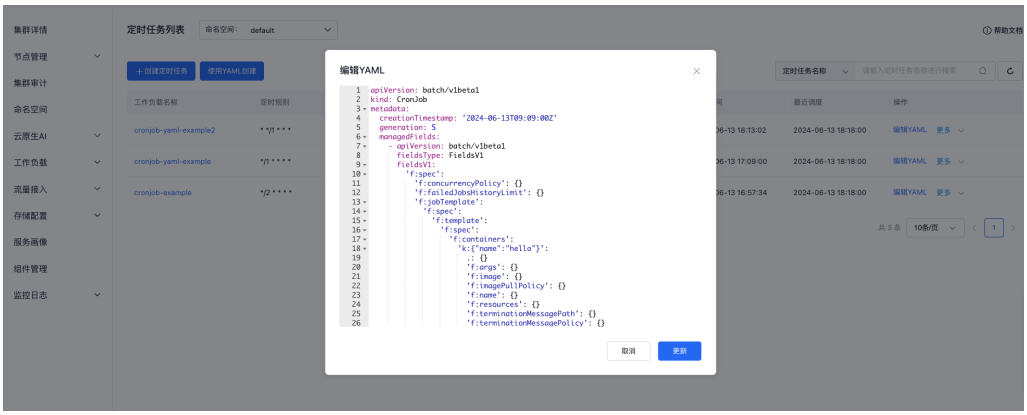
点击列表页任何一行的定时任务名称，即可进入定时任务详情页，详情页内容包括：

- 定时任务详情：包括基本信息、容器配置
- 关联任务
- 调度策略
- 事件



修改定时任务

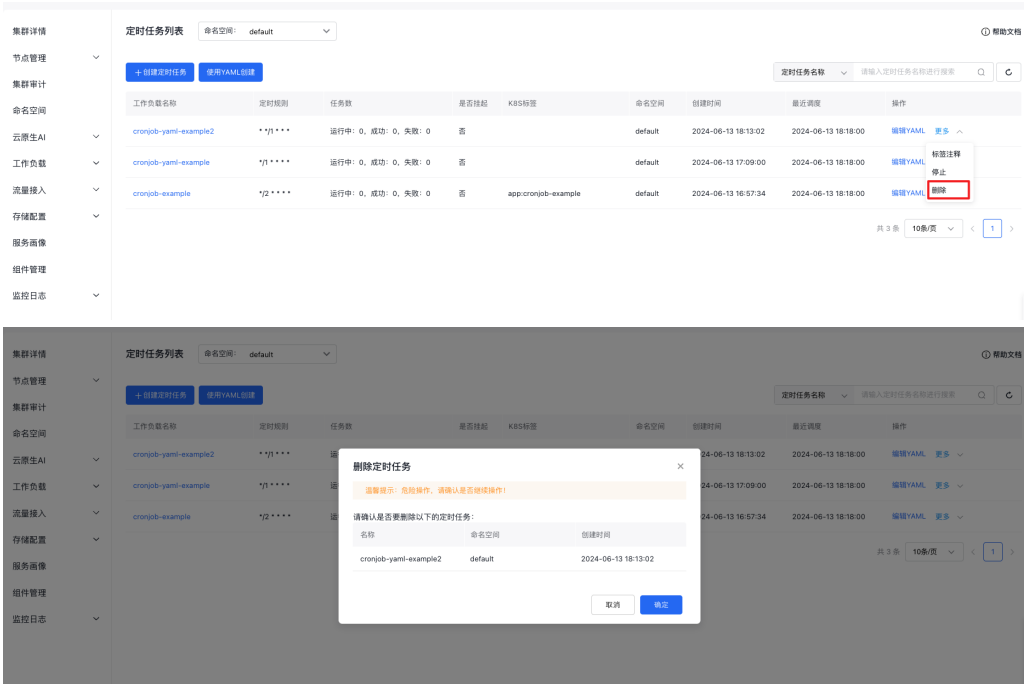
在定时任务列表页面，点击需要修改的定时任务行中的“编辑YAML”按钮，完成修改后，点击“更新”提交。



删除定时任务

在定时任务列表页面，找到要删除的定时任务，点击“更多”后再点击“删除”按钮，确认信息后，点击“确认”。

> **说明：** 目前仅支持单个删除。

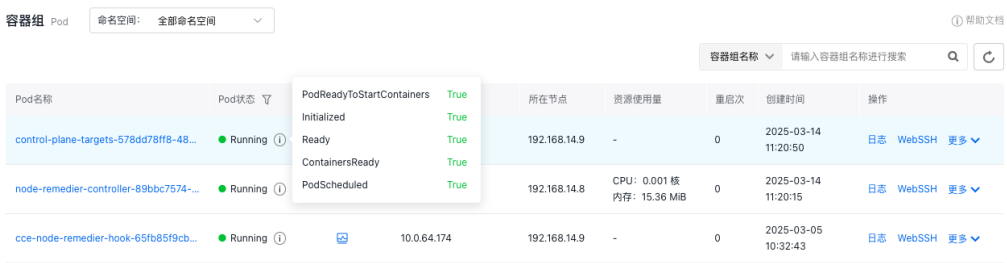


查看容器组

容器组（Pod）是Kubernetes中最小的可部署单元。Pod代表了Kubernetes中一个独立的应用程序运行实例，该实例可能由单个容器或者几个紧耦合在一起的容器组成。您可以在容器服务控制台上查看Pod、变更Pod配置以及手动伸缩容器应用。

查看容器组

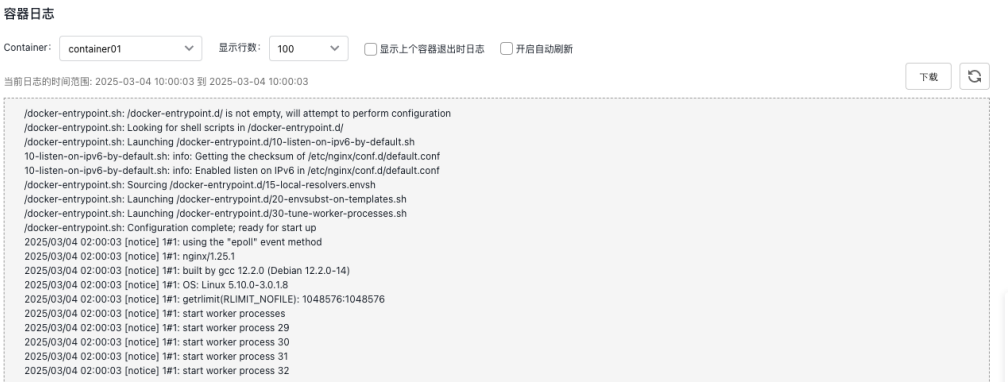
1. 进入**产品服务>容器引擎CCE**，选择集群，单击**工作负载>容器组**，进入容器组列表。



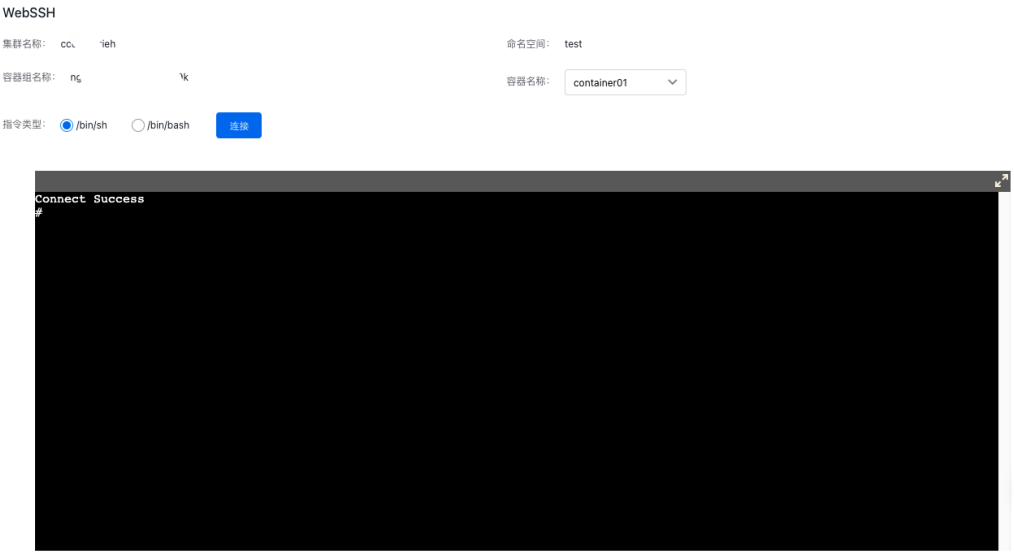
- Pod状态：展示容器组原生Phase状态及健康检查Conditions。
- CPU使用量：展示当前的CPU使用情况，以核数为单位。
- 内存使用量：展示当前的内存使用情况，以字节数为单位，如100MiB。
- 列表数据：展示所有的容器组列表信息，容器状态以及日志。

2. 单击容器组列表中的监控图标，系统弹出容器组监控页面，更多信息请参见使用Prometheus监控集群。

3. 单击容器组列表页面目标容器组名称，即可进入容器组详情页，详情页内容包括：
   - 基本信息：名称、命名空间、所属工作负载、状态、Pod ip、所在节点、标签、注释、创建时间。
   - 网络信息：节点IP、容器组IP。
   - 其他：容器列表、存储。
4. 单击左侧导航栏中的**日志**，进入容器日志页面，查看容器组详情。



5. 单击左侧导航栏中的**WebSSH**，即可登录容器，执行操作。



6. 单击左侧导航栏中的**事件**，进入容器事件页面，查看容器事件详情。

Pod故障诊断

在容器组列表页面，您可以单击目标容器组右侧的操作列的诊断，对该容器组进行故障诊断。更多信息请参见故障诊断。

使用私有镜像创建工作负载

私有镜像指位于私有仓库中的Docker镜像，用户需要使用仓库用户名密码登入(docker login)后才能拉取。在CCE集群中，需要为Pod配置imagePullSecrets，才能够在集群中使用私有镜像启动Pod中的容器。

1. 创建用于私有镜像拉取的secret (imagePullSecrets)

使用kubectl命令行创建，需要提供

- 镜像仓库登录用户名
- 镜像仓库登录密码

- 镜像仓库地址
- 邮件地址（可选）

  kubectl create secret docker-registry <secret名称> --docker-username=<镜像仓库用户名> --docker-password=<镜像仓库密码> --docker-server=<镜像仓库地址> --docker-email=<邮件地址（可选）>

对于存储在百度智能云CCR企业版镜像仓库（镜像地址为{实例ID}-vpc(或pub).cnc.{regrion}.baidubce.com/<namespace>/<image>:<tag>）的私有镜像，镜像仓库地址参数须填写为--docker-server='https://仓库地址'。

> **注意：**
>
> - secret需与引用该secret的pod在同一个k8s命名空间内。

创建完毕后，可以通过kubectl get secret查看到一个类型为 kubernetes.io/dockerconfigjson 的secret被创建。

```
 $ kubectl get secret
NAME                  TYPE                                  DATA      AGE
default-token-bx894   kubernetes.io/service-account-token   3         4d
private-repo-auth     kubernetes.io/dockerconfigjson        1         2m
```

2. 在pod spec中指定拉取镜像的secret

在 pod.spec.imagePullSecrets 中指定secret的名称，即可在pod中使用相应的私有镜像。

以创建一个Deployment为例，假设已经在default命名空间中通过上述方式创建了名为private-repo-auth的secret，对应的Deployment yaml文件如下：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-nginx
spec:
 replicas: 1
 selector:
   matchLabels:
     app: my-nginx
 template:
   metadata:
     name: my-nginx
     labels:
       app: my-nginx
   spec:
     imagePullSecrets:
     - name: private-repo-auth
     containers:
     - name: my-nginx
       image: nginx
```

> **注意：**
>
> - 可以在同一个pod中指定多个拉取镜像的secret，用于拉取不同私有仓库的私有镜像。

3. 使用仓库访问凭证

在创建工作负载时，在高级设置里可以添加仓库访问凭证以使用私有镜像。

点击“+添加仓库访问凭证”，使用已有凭证或新建凭证，新建凭证需要填写凭证名称、仓库地址、用户名、密码。

流量接入

通过 CCE 使用 K8S Service

概述

用户在Kubernetes中可以部署各种容器，其中一部分是通过HTTP、HTTPS协议对外提供七层网络服务，另一部分是通过TCP、UDP协议提供四层网络服务。而Kubernetes定义的Service资源就是用来管理集群中四层网络的服务访问。

Kubernetes的ServiceTypes允许指定Service类型，默认为ClusterIP类型。ServiceTypes的可取值以及行为描述如下：

类型	说明
ClusterIP	通过集群的内部 IP 暴露服务。当您的服务只需要在集群内部被访问时，请使用该类型。该类型为默认的 ServiceType。
NodePort	通过每个集群节点上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，该 ClusterIP 服务会自动创建。通过请求：，可从集群的外部访问该 NodePort 服务。 **注意：**除了测试以及非生产环境以外，不推荐在生产环境中直接通过集群节点对外甚至公网提供服务。从安全上考虑，使用该类型会直接暴露集群节点，容易受到攻击。
LoadBalancer	使用百度智能云的负载均衡器，可以向公网或者内网暴露服务。负载均衡器可以路由到 NodePort 服务，或直接转发到容器中。具体使用方式，请参考通过YAML创建LoadBalancer_Service。

创建服务

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**网络>服务**”进入服务列表页面。

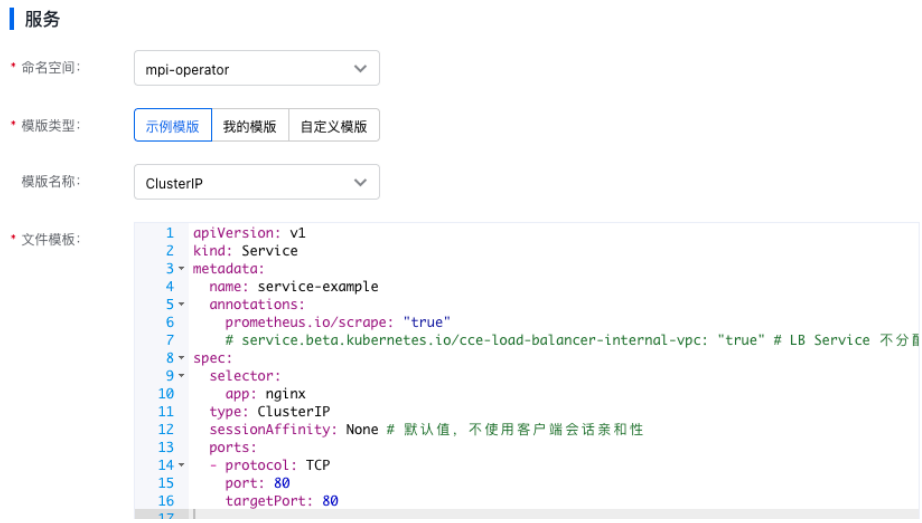
方式一：通过控制台服务

单击“**创建服务**”，进入创建服务页面，设置服务相关的参数。

配置项	描述	示例
服务名称	输入Service（服务）的名称	my-nginx-svc
服务类型	支持三种模式，分别对接不同来源和类型的客户端的访问：集群内访问（ClusterIP）、节点端口访问（NodePort）、负载均衡（LoadBalancer）	无
IP 双栈配置	单栈：单栈 Service，控制面使用第一个配置的服务集群 IP 范围为 Service 分配集群 IP； 双栈：启用双栈时，同时从 IPv4 和 IPv6 的地址范围中分配 Service 的 .spec.clusterIPs。 如果双栈未被启用或不被支持，则 Service API 对象创建失败; 偏好双栈：启用双栈时，为 Service 同时分配 IPv4 和 IPv6 集群 IP 地址。 如果双栈未被启用或不被支持，则会返回到单栈行为；	无
服务关联	选择服务要绑定的后端应用。若不进行关联部署，则不会创建相关的Endpoints对象。	名称：app、值：nginx
端口映射	添加服务端口和容器端口，容器端口需要与后端的Pod中暴露的容器端口一致	服务端口：80、容器端口：80、协议：TCP
注解	为该服务添加一个注解（Annotation），配置负载均衡的参数。	无

方式二：使用Yaml创建服务

单击“**Yaml创建资源**”进入新建服务页面指定服务类型并完成其他配置。



单击“**确定**”按钮完成服务创建。

修改服务

在服务列表页面，选择目标服务单击“修改”按钮，完成修改后，点击“更新”按钮完成修改。

删除服务

在服务列表页面，选择目标服务单击“删除”按钮，确认信息后，点击“确认”完成删除。

通过 YAML 创建 LoadBalancer Service

本文档会详细介绍如何在CCE下创建类型是LoadBalancer的Service。

注意：以下Annotation对1.16.3以下版本集群可能不生效，辛苦工单联系管理员处理

Kubernetes官方教程：Services

快速开始

当用户创建类型是LoadBalancer的Service，**默认情况下，CCE会联动的创建BLB，并为此BLB绑定弹性公网EIP。**

以创建一个简单的Nginx为例：

```
  kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  type: LoadBalancer
  ports:
  - name: nginx-port
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: registry.baidubce.com/public/nginx:latest
        ports:
        - containerPort: 80
```

(1)创建

```
$ kubectl create -f nginx.yaml
```

(2)查询EIP

IP 8.8.8.8 即为此nginx-service的EIP。

```
 $ kubectl get svc
NAME            CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
nginx-service   1.1.1.1        8.8.8.8          80:30274/TCP   5m
```

(3)查询BLB

```
 $ kubectl get svc nginx-service -o jsonpath={.metadata.annotations}
map[service.beta.kubernetes.io/cce-load-balancer-id:lb-xxxxxx]
```

lb-xxxxxx即为此Service的BLB的id。

(4)访问测试

```
$ curl -i http://8.8.8.8
```

外部流量策略

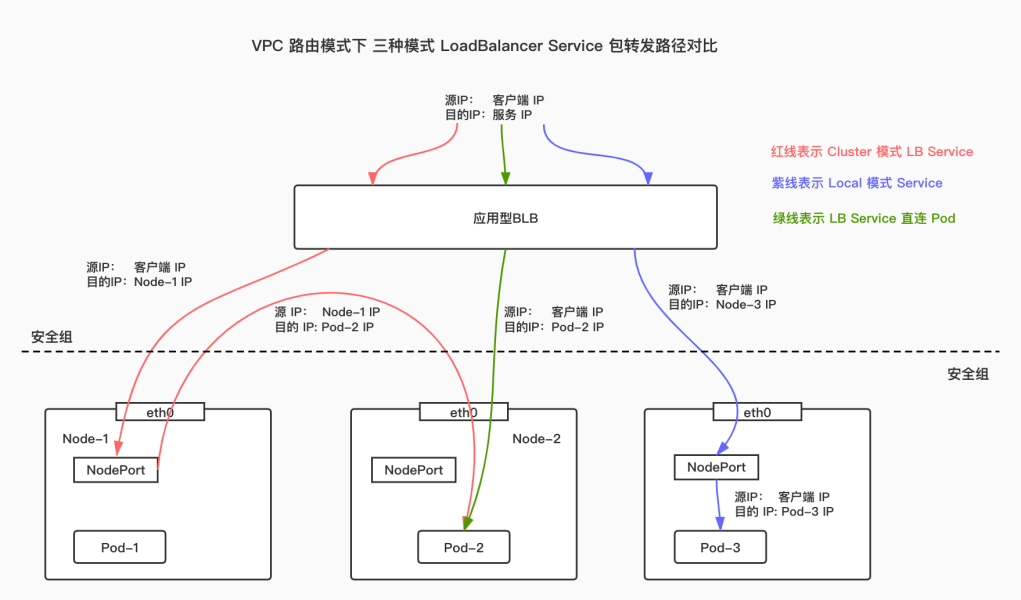
CCE提供三种外部流量策略的LoadBalancer Service，本文称其为Cluster模式、Local模式与LB直连Pod模式。

对于Cluster模式的LB Service，当其收到数据包后，负载均衡器将数据包发往集群的某个节点上，进而节点二次转发数据包到集群的某个Pod上。目标Pod的宿主节点和转发数据包的节点可能不是同一个；这种情况数据包的源IP地址**会丢失**。

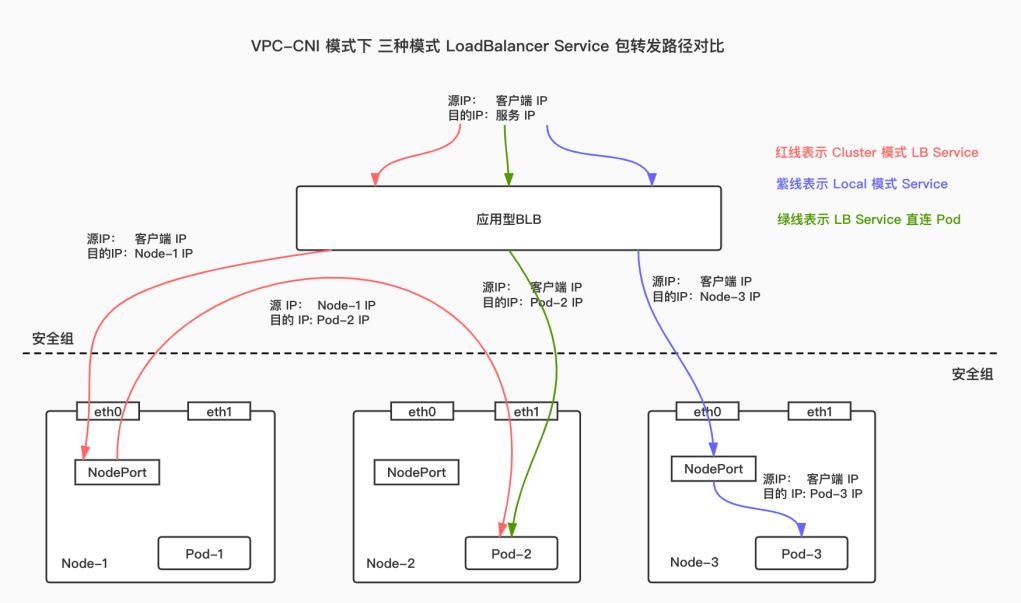
对于Local模式的LB Service，当其收到数据包后，负载均衡器将数据包发往部署有目标Pod的节点的上，再由节点转发到自身的Pod上；这种情况数据包的源IP地址**不会丢失**。

对于LB直连Pod模式的LB Service，当其收到数据包后，负载均衡器直接将数据包发往各个Pod。相比前两种模式，这种模式减少了一次节点转发操作。如果集群的容器网络模式是VPC-ENI模式，则源IP**不会丢失**；如果集群的容器网络模式是VPC路由模式，如果宿主机安装了**ttm模块**，源IP地址**不会丢失**，如果宿主机**没有安装ttm模块**，源IP地址则**会丢失**。

在VPC路由网络模式下，三种模式LoadBalancer Service的数据包转发路径如图所示：



在VPC-CNI网络模式下，三种模式LoadBalancer Service的数据包转发路径如图所示：



Cluster模式 如果希望使用Cluster模式的Service，在创建Service时，应指定`externalTrafficPolicy: Cluster`。如以下例子所示：

```
 apiVersion: v1
kind: Service
metadata:
  name: service-example-cluster
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: nginx
  type: LoadBalancer
  externalTrafficPolicy: Cluster
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

Local模式 如果希望使用Local模式的Service，在创建Service时，应指定`externalTrafficPolicy: Local`。如以下例子所示：

```
  apiVersion: v1
kind: Service
metadata:
  name: service-example-local
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: nginx
  type: LoadBalancer
  externalTrafficPolicy: Local
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

注意

当kube-proxy组件的代理模式为ipvs模式时，如果某个Service的externalTrafficPolicy为Local且某个节点上没有Service关联的后端Pod时，在该节点上访问该Service的BLB IP地址将会失败。

该问题已经在Kubernetes 1.24版本中修复。相关issue为：https://github.com/kubernetes/kubernetes/pull/97081

LB直连Pod模式 如果希望使用LB直连Pod模式的Service时，在创建Service时，应添加Annotationservice.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"。如以下例子所示：
更详细的使用说明请参考使用直连 Pod 模式 LoadBalancer Service.md。

```
  apiVersion: v1
kind: Service
metadata:
  name: service-example-direct
  annotations:
    prometheus.io/scrape: "true"
    service.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"
spec:
  selector:
    app: nginx
  type: LoadBalancer
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

使用UDP-Service

修改spec.ports.protocol为UDP即可使用UDP Service的功能，举例如下：

```
 ---
apiVersion: v1
kind: Service
metadata:
  name: udp-server-demo-svc
  labels:
    app: udp-server-demo
spec:
  type: LoadBalancer
  ports:
  - name: udp-server-demo-port
    port: 3005
    targetPort: 3005
    protocol: UDP
  selector:
    app: udp-server-demo
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: udp-server-demo
  labels:
    app: udp-server-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: udp-server-demo
  template:
    metadata:
      labels:
        app: udp-server-demo
    spec:
      containers:
      - name: udp-server-demo
        image: hub.baidubce.com/jpaas-public/udp-server-demo:latest
        ports:
        - containerPort: 3005
          protocol: UDP
```

（1）部署UDP测试服务

```
 $ kubectl apply -f udp.yaml
```

（2）UDP Service创建成功

```
 $ kubectl get svc
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes          ClusterIP     172.16.0.1      <none>           443/TCP          6h
udp-server-demo-svc   LoadBalancer   172.16.122.139   10.10.10.10      3005:31441/UDP   1m
```

（3）查看服务日志

```
 $ kubectl logs -f udp-server-demo-6fdf5d796f-h6595
Received: HealthCheck
Get Health Check, response OK
Received: HealthCheck
Get Health Check, response OK
Received: HealthCheck
Get Health Check, response OK
```

注意

当使用UDP服务时，应用型BLB会使用UDP健康检查字符串对后端进行健康检查，参考文档UDP健康检查介绍

用户需要在应用型BLB上自行配置UDP健康检查字符串，并在后端Pod上正确响应健康检查字符串。

Service复用已有负载均衡BLB

容器引擎CCE支持通过 service.beta.kubernetes.io/cce-load-balancer-id: <LoadBalanceId>注解实现使用已有负载均衡的功能，您可使用该注解指

定集群Service资源关联的负载均衡实例。当前还支持Service负载均衡复用功能，即指定多个Service使用同一个已有负载均衡，本文将介绍如何配置。

> **说明：**
>
> 仅CCE LB Controller组件版本为1.30.1及以上支持复用BLB，低版本不支持复用，您可以在集群的**组件管理 > 网络**中进行版本升级，更多信息请见 CCE LB Controller 说明。

使用负载均衡设置BLB标签行为

创建LoadBalance类型Service时新建BLB时，CCE会自动为对应BLB设置 `cce-created` 和 `cce-resource-name`标签，标签内容为：

- `cce-created：<ClusterID>/Service/UID/<Service UID>` 表示该BLB由CCE创建，Service会在销毁时默认删除对应资源。
- `cce-resource-name：<ClusterID./Service/UID/<Service UID>` 表示该BLB已被集群中对应的Service使用。

创建LoadBalance类型Service时使用已有BLB时，CCE会自动为对应BLB设置 `cce-resource-name` 标签，CCE LB Controller组件不同版本标签内容及作用不同：

- CCE LB Controller组件版本为1.30.1及以上，标签为 `cce-resource-name：<ClusterID>` ，该标签用于标识已被集群使用，且支持该集群下的其他Service复用，删除Service时仅会删除BLB中对应的监听器和目标组。
- CCE LB Controller组件版本为1.30.1以下，标签格为 `cce-resource-name：<ClusterID>/Service/UID/<Service UID>` ，该标签用于标识已被集群中对应的Service使用，不支持该集群下的其他Service复用。

> **注意：**
>
> 请不要修改和删除CCE设置的 `cce-created` 和 `cce-resource-name` 这两个标签，否则可能会导致资源泄漏，

使用限制

- 仅支持复用已有应用型负载均衡BLB。
- 复用的BLB需要和集群处于同一个VPC内。
- 不支持复用CCE自动创建的BLB实例。
- 多个Service复用BLB时，Service的端口不能重复。
- 不支持跨集群复用。

Service示例

```
 apiVersion: v1
kind: Service
metadata:
  name: service-example
  annotations:
    prometheus.io/scrape: "true"
    service.beta.kubernetes.io/cce-load-balancer-id: lb-xxxxxxxx #指定已有的应用型BLB实例
spec:
  selector:
    app: nginx
  type: LoadBalancer
  externalTrafficPolicy: Cluster # 默认值
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

使用直连 Pod 模式 LoadBalancer Service

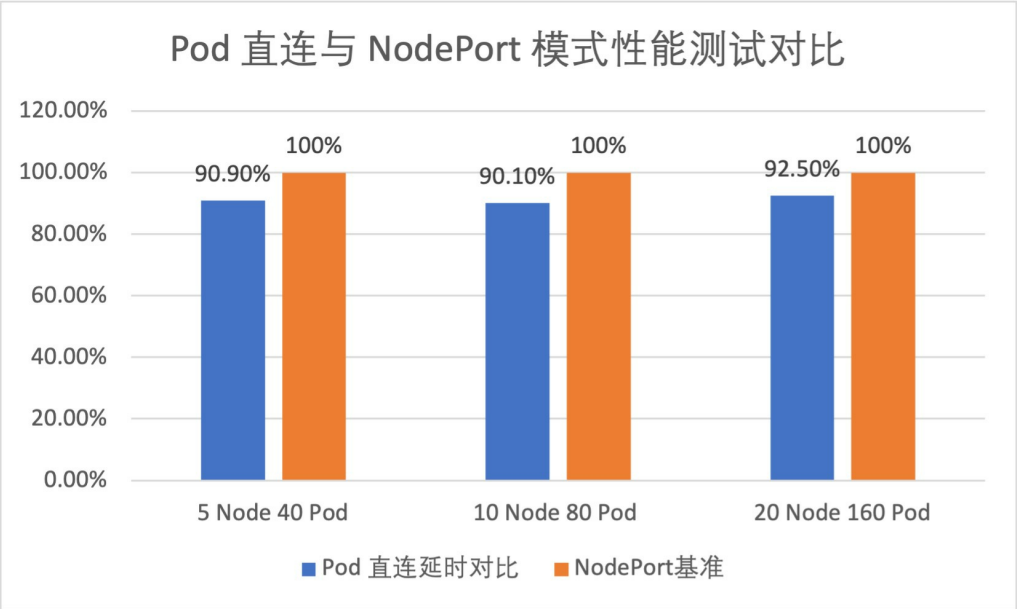
传统模式与直连模式对比

特性 传统模式 LoadBalancer Service 以集群节点的 IP 地址作为 BLB 的后端服务器。 当访问传统模式 LoadBalancer Service 时，请求会被百度智能云提供的负载均衡服务（BLB）转发到节点上，并在节点中进一步通过 iptables 或 IPVS 转发到具体的某一个 Pod 上。 在这个过程中请求经历

了 2 次负载均衡操作。

直连 Pod 模式 LoadBalancer Service 以 Pod 的 IP 地址作为其后端服务器。 当访问直连 Pod 模式 LoadBalancer Service 时，请求只需经过一次负载均衡操作即可被转发到具体的 Pod 上。 这种模式的 Service 具有以下优势：

- 保留请求的源 IP；
- 少一次请求转发，系统具有更高的性能。
- Pod 层面更均匀的负载均衡 **性能** 针对两种模式的 LoadBalancer Service，我们对其平均网络延时进行了测试。根据测试结果，直连 Pod 的 LoadBalancer Service 平均可以减少约 10% 的网络时延。



使用限制

使用直连 Pod 模式 LoadBalancer Service 的集群应同时满足集群版本、插件、容器网络类型 3 个方面的前提条件：

确认集群版本 新版 CCE 集群（集群 ID 以 `cce-` 为前缀）集群支持此功能。
旧版 CCE 集群（集群 ID 以 `c-` 为前缀）集群不支持此功能。

确认容器网卡类型

- 网卡类型为 veth 或 kubenet 时，满足要求。
- 网卡类型为 IPVLAN 时，需要设置 ConfigMap `cce-ip-masq-agent` 的 `masqOutBound` 和 `masqOutBoundIPv6` 属性为 false，操作流程如下：

  使用Kubectl连接集群，输入 `kubectl describe configmap cce-ip-masq-agent -n kube-system` 指令，您可能会得到如下输出，此时 `masqOutBound` 与 `masqOutBoundIPv6` 为 true：

```
 # kubectl describe configmap  cce-ip-masq-agent -n kube-system
Name:         cce-ip-masq-agent
Namespace:    kube-system
Labels:       addonmanager.kubernetes.io/mode=EnsureExists
Annotations:
Data
====
config:
----
nonMasqueradeCIDRs:
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16
  - fc00::/7
masqOutBound: true
masqOutBoundIPv6: true
masqLinkLocal: false
masqLinkLocalIPv6: false
resyncInterval: 60s

Events:  <none>
```

  输入 `kubectl edit configmap cce-ip-masq-agent -n kube-system` 指令，然后修改 `masqOutBound` 与 `masqOutBoundIPv6` 为 false 并保存，cce-ip-masq-agent 配置信息将会自动更新，此时再次输入 `kubectl describe configmap cce-ip-masq-agent -n kube-system` 指令，您可

能会得到如下输出，此时 masqOutBound 与 masqOutBoundIPv6 已经为 false。

```
 # kubectl describe configmap  cce-ip-masq-agent -n kube-system
Name:         cce-ip-masq-agent
Namespace:    kube-system
Labels:       addonmanager.kubernetes.io/mode=EnsureExists
Annotations:
Data
====
config:
----
nonMasqueradeCIDRs:
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16
  - fc00::/7
masqOutBound: false
masqOutBoundIPv6: false
masqLinkLocal: false
masqLinkLocalIPv6: false
resyncInterval: 60s

Events:  <none>
```

注意事项

检查配额限制

直连 Pod 模式 LoadBalancer Service 使用应用型 BLB，其后端默认最多关联 50 个 Pod。 如果无法满足需求，请提交工单申请开放后端数量限制。

开放安全组端口

当 CCE 集群的容器网络模式为 **VPC 网络** 时，需要为集群节点绑定的安全组中为目的端口 Service TargetPort 放通流量。否则数据包无法到达 Pod。

当 CCE 集群的容器网络模式为 **VPC-CNI** 时，需要在 ENI 弹性网卡绑定的安全组中为目的端口 Service TargetPort 放通流量。否则数据包无法到达 Pod。

操作步骤

创建直连 Pod 模式 LoadBalancer Service 时，需要为 annotations 属性添加 service.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"。

示例 yaml 如下：

```yaml
  apiVersion: v1
kind: Service
metadata:
  name: pod-direct-service-example
  annotations:
    prometheus.io/scrape: "true"
    service.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"
spec:
  selector:
    app: nginx
  type: LoadBalancer
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

常见问题

Pod 零中断滚动更新 使用直连 Pod 的 LB Service 在 Pod 滚动更新时可能服务中断问题。

原因

一个 Pod 在被终止时，会发生如下事件：

1. Pod 进入 Terminating 状态；
2. Pod 从 Endpoints 中被摘除；LB Controller 监听到 Endpoints 变化，并从应用型 BLB 中摘除该 Pod 后端；

3. Pod 销毁；

   其中2，3步是并行执行的。当 Pod 销毁后，LB Controller 仍未将 Pod 摘除时，就可能会有流量被转发到被销毁的 Pod，导致服务中断。

解决方法

为 Pod 设置 preStopHook，睡眠一段时间（一般为 40 秒），待到应用型 BLB 将 Pod 摘除后再销毁。

参考如下：

```
      lifecycle:
       preStop:
         exec:
           command: ["sleep", "40"]
```

LoadBalancer Service Annotation说明

LoadBalancer Service Annotation说明

在使用CCE提供的LoadBalancer类型的Service资源时（下文简称LB Service），可以通过在Annotation中添加注解的方式来使用一些高级配置。

Annotation位于Service完整YAML的以下位置：

```
  apiVersion: v1
kind: Service
metadata:
  name: service-example
  annotations:
    prometheus.io/scrape: "true"
    service.beta.kubernetes.io/cce-load-balancer-internal-vpc: "true" # LB Service 不分配 EIP
......
```

LoadBalancer Service可以使用以下高级配置项：

指定使用的应用型BLB

取值说明

若设置，CCE将在指定的应用型BLB上配置LB Service规则；

若缺失，CCE会新建一个应用型BLB，并自动设置该Annotation为新建应用型BLB的ID。

注意事项

- 只支持应用型BLB；
- 需要自行先创建好应用型BLB，BLB应在集群所在的VPC内；
- 修改或删除可能会导致BLB泄露，修改LB Service资源对象时不要覆盖或丢失该Annotation。

示例

`service.beta.kubernetes.io/cce-load-balancer-id: lb-12345678`

LB Service 根据 Label 选择挂载目标节点 取值说明

Service 可以通过注解 `service.beta.kubernetes.io/cce-load-balancer-backend-label: "k1=v1, k2=v2 ..."` ，选择 Label 匹配的节点进行绑定；

若未添加标签，Service 后端会排空，相应的 Service 描述中会上报事件。

注意事项

- 多个 Label 以逗号分隔，例如 `"k1=v1,k2=v2"` ，多个Label之间是 And 关系；
- 当注解为 `service.beta.kubernetes.io/cce-load-balancer-backend-label: "k1=v1, k2=v2 ..."` 的选择器没有选取到任何节点的时候，服务的后端将会被排空，会使得服务中断。使用此功能时，需要对集群节点的 Label 有一定的管理；
- 添加符合要求的节点或变更存量节点也会触发更新；
- Service Local 模式本身就具有节点选择能力，所以请不要在 Service Local 模式下，使用指定接入层后端的能力。

示例

```
  apiVersion: v1
kind: Service
metadata:
  name: service-example
  annotations:
    service.beta.kubernetes.io/cce-load-balancer-backend-label: k1=v1, k2=v2 # 多个Label之间是 And 关系
......
```

指定EIP地址/在删除时保留EIP资源 取值说明

该配置在Spec字段中，不在Annotation中。

在spec.loadBalancerIP配置一个EIP地址，CCE将会把该EIP绑定在LB Service对应的应用型BLB上；并且在删除LB Service资源时，也不会删除该EIP资源；

若缺失，CCE会默认新建一个EIP，但不会设置这里的字段，该字段将始终保持为空。

注意事项

- 可以指定预付费EIP资源；
- 修改或删除可能会导致EIP泄漏，修改LB Service资源对象时不要覆盖或丢失该配置；
- 如果在创建时忘记添加，可以在删除前补上这一条，在删除前添加依然有效；
- 如果LB Service上已经使用了某个EIP，更改该Annotation不会更换应用型BLB上绑定的EIP地址，需要用户自行换绑。

示例

```
  kind: Service
apiVersion: v1
metadata:
    name: test
spec:
    type: LoadBalancer
    loadBalancerIP: 8.8.8.8
......
```

设置LB Service关联的BLB名称

取值说明

用于设置该LB Service关联的BLB的名称，否则使用默认生成的名称格式（集群ID/svc/命名空间/Service名称）来创建BLB。

有时，默认名称会超出最长限制（64个字符），此时需要手动设置该Annotation的值来指定BLB的名称。

注意事项

- 不要和已有BLB的名称相同，否则可能发生混淆，导致误用BLB的情况发生。

示例

`service.beta.kubernetes.io/cce-load-balancer-lb-name: cce-test-name`

在删除时保留应用型BLB资源 取值说明

若值为`true`，则在该LB Service资源对象被删除时，关联的应用型BLB资源不会被删除；

若缺失，或值是`false`，则在该LB Service资源对象被删除时，关联的应用型BLB将会被删除。

注意事项

- 如果在创建时忘记添加该Annotation，可以在删除前补上这一条，在删除前添加依然有效。

示例

`service.beta.kubernetes.io/cce-load-balancer-reserve-lb: "true"`

指定新建BLB所在子网

取值说明

若值为集群所在VPC的有效子网ID，则CCE会在需要新建应用型BLB时，将应用型BLB创建在该子网中；

若缺失，CCE将会自行选择创建应用型BLB的子网。

注意事项

- 只针对新建应用型BLB有效。

示例

`service.beta.kubernetes.io/cce-load-balancer-subnet-id: "sbn-12345678"`

设置LB Service仅供VPC内网访问

取值说明

若值为`true`，CCE不会为其创建和设置EIP地址；

若缺失，或值是`false`，CCE会为应用型BLB分配和绑定EIP。

注意事项

- 如果应用型BLB上已经绑定了EIP，在添加该Annotation后，EIP不会被后台解绑，用户需要自行解绑。

示例

`service.beta.kubernetes.io/cce-load-balancer-internal-vpc: "true"`

自定义EIP配置 取值说明

LB Service在未指定EIP时且未配置仅供VPC内网访问时，会创建EIP资源。

可以在Annotation中设置新建EIP的配置：

- 指定计费方式：

支持`ByTraffic`（按流量计费）和`ByBandwidth`（按带宽计费）。

对应的Annotation为`service.beta.kubernetes.io/cce-elastic-ip-billing-method`。

- 指定线路类型：

支持增强型 BGP 与标准型 BGP 两种线路类型，CCE 默认支持对应的Annotation为`service.beta.kubernetes.io/cce-elastic-ip-route-type`，其中`BGP_S` 代表增强型 BGP，`BGP` 代表标准型 BGP，`ChinaTelcom` 代表电信单线，`ChinaUnicom` 代表联通单线，`ChinaMobile` 代表移动单线，使用示例如下：

示例

```
 ##### 增强型 BGP
service.beta.kubernetes.io/cce-elastic-ip-route-type:'BGP_S'
```

- 指定带宽：

取值范围为[1, 5000]。

对应的Annotation为`service.beta.kubernetes.io/cce-elastic-ip-bandwidth-in-mbps`。

注意事项

- 新建EIP都是后付费类型，暂不支持使用Annotation指定创建预付费EIP；
- 计费方式和线路类型只针对新建EIP有效；EIP带宽允许变更，但需要增加 全量配置更新（配置实时更新）Annotation `service.kubernetes.io/cce-update-in-time: "true"` 才能生效。
- 当带宽额度大小不为合法数字时，会上报警告事件；
- 当您没有添加带宽额度注解时，CCE 会为您默认创建并绑定带宽额度大小为100，计费方式为按流量计费的标准型 EIP；
- 境外部分区域暂不支持增强型 BGP 创建，详情请见；
- 带宽额度在不同线路类型和计费类型组合下，取值范围不同，如下表所示：

线路类型	标准型 BGP	增强型 BGP
流量计费	[1, 200]	[100, 1000]
带宽计费	[1, 500]	[100, 5000]

- 在不满足以下四种条件的情况下，会导致创建 EIP 失败：

线路类型	标准型 BGP 使用说明	增强型 BGP 使用说明
流量计费	需要添加如下带宽范围注解，且注解取值范围是[1, 200] service.beta.kubernetes.io/cce-elastic-ip-bandwidth-in-mbps: "99" service.beta.kubernetes.io/cce-elastic-ip-billing-method: ByTraffic service.beta.kubernetes.io/cce-elastic-ip-route-type: BGP	需同时添加如下带宽范围和线路类型注解，且注解取值范围是[100, 1000] service.beta.kubernetes.io/cce-elastic-ip-bandwidth-in-mbps: "200" service.beta.kubernetes.io/cce-elastic-ip-billing-method: ByTraffic service.beta.kubernetes.io/cce-elastic-ip-route-type: BGP_S
带宽计费	需同时添加如下带宽范围和计费类型注解，且注解取值范围是[1, 500] service.beta.kubernetes.io/cce-elastic-ip-bandwidth-in-mbps: "99" service.beta.kubernetes.io/cce-elastic-ip-billing-method: ByBandwidth service.beta.kubernetes.io/cce-elastic-ip-route-type: BGP	需同时添加如下带宽范围、线路类型和计费类型注解，且注解取值范围是[100, 5000] service.beta.kubernetes.io/cce-elastic-ip-bandwidth-in-mbps: "5000" service.beta.kubernetes.io/cce-elastic-ip-billing-method: ByBandwidth service.beta.kubernetes.io/cce-elastic-ip-route-type: BGP_S

隐藏应用型BLB内网IP地址 取值说明

若值为`true`，即可在Service的Status中不再展示应用型BLB内网IP字段；

若缺失，或值是`false`，则Service的Status中正常展示应用型BLB内网IP字段。

示例

`service.beta.kubernetes.io/cce-load-balancer-external: "true"`

设置BLB默认挂载的最大后端数

取值说明

该Annotation用于限制LB Service挂载的后端数目。

默认是200。

注意事项

- 如果Service的Spec.ExternalTrafficPolicy值为Local，或使用Pod直连模式，请设置该值确保比集群节点数大。否则未挂载到BLB后端的节点或Pod将无法接收到流量。

示例

`service.beta.kubernetes.io/cce-load-balancer-rs-max-num: 500`

使用Pod直连模式 取值说明

添加Annotation`service.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"`，将会使用Pod直连模式配置应用型BLB， 即流量直接从应用型BLB到达后端Pod。

注意事项

- 不可以在已有的Annotation上添加该Annotation来将其切换成Pod直连模式，如果有需要，请新建一个LB Service；反之亦然。

示例

`service.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"`

全量配置更新（配置实时更新） 取值说明

- CCE 默认不会同步所有的 BLB 配置：即使有修改，CCE 也不会主动同步BLB 监听器配置、健康检查规则、EIP 带宽配置、BLB 安全组规则。
- 若该Annotation值为true，CCE 会全量更新 BLB 配置：在修改 BLB 监听器配置、健康检查规则、EIP 带宽配置、BLB 安全组规则之后，立即更新 BLB 和 EIP 相关配置；

注意事项

- CCE不会自动添加该 Annotation，需要自行按需添加；

示例

`service.kubernetes.io/cce-update-in-time: "true"`

配置BLB安全组 取值说明

Service 可以通过注解 service.beta.kubernetes.io/cce.security.groups: "sgID1, sgID2 ..."，设置BLB实例所关联的安全组，支持传入多个安全组ID，以英文逗号隔开。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加，若未设置此注解，则CCE默认不会给BLB实例关联安全组；
- 需要同时搭配 service.kubernetes.io/cce-update-in-time: "true" 注解使用，用于触发安全组配置更新到BLB实例

示例

`service.beta.kubernetes.io/cce.security.groups: 'sgID1,sgID2'`

使用专属BLB集群 取值说明

当用户存在BLB专属集群时，后端会优先将应用型BLB创建在BLB专属集群上。

使用以下Annotation可以指定是否使用专属集群，以及使用的专属集群ID：

- 使用共享集群创建应用型BLB

使用该Annotation`service.beta.kubernetes.io/no-use-blb-cluster-l4: "true"`之后，应用型BLB将创建在共享集群上。

- 指定创建在L4专属集群上

使用该Annotation`service.beta.kubernetes.io/no-use-blb-cluster-l4: "test"`之后，应用型BLB将创建在指定的专属BLB集群上。

注意事项

- 有了专属集群之后，如果没有显式添加Annotation，将由后台自动选择所在的专属BLB集群。

示例

```
service.beta.kubernetes.io/no-use-blb-cluster-l4: "true"
```

扩展Service端口协议 取值说明

扩展协议配置用于补充Spec中规定的端口的协议配置，这些扩展配置将作用于应用型BLB的监听器上。

扩展协议中配置的端口，应首先存在于Spec配置中，否则扩展协议配置不会生效。

扩展协议内容在service.beta.kubernetes.io/cce.listener.customized-listener Annotation下，完整配置内容请参考示例。每种协议能配置的参数如下：

扩展TCP协议

参数名称	描述
protocol	TCP
timeout	非必填。TCP会话超时时间。
scheduler	负载均衡方法。支持"RoundRobin"、"LeastConnection"、"Hash"，默认"RoundRobin"。

扩展UDP协议

参数名称	描述
protocol	UDP
timeout	非必填。UDP会话超时时间。
scheduler	负载均衡方法。支持"RoundRobin"、"LeastConnection"、"Hash"，默认"RoundRobin"。

扩展TCP-SSL协议

参数名称	描述
protocol	TCP-SSL
timeout	非必填。TCP-SSL服务端超时时间。默认"30"。
scheduler	非必填。负载均衡方法。支持"RoundRobin"、"LeastConnection"、"Hash"，默认"RoundRobin"。
serverCertIDs	必填。服务端证书ID链，逗号分隔。
encryptionType	非必填。加密选项，默认tls_cipher_policy_default。支持：tls_cipher_policy_default/tls_cipher_policy_1_1/tls_cipher_policy_1_2/tls_cipher_policy_1_2_secure/userDefind。
encryptionProtocols	非必填。加密协议，当encryptionType值为userDefind时协议类型列表，是以"tlsv10"、"tlsv11"、"tlsv12"三种协议组合形成的字符串列表。
appliedCiphers	非必填。加密套件。不同加密套件用冒号":"隔开。
dualAuth	非必填。是否开启双向认证。默认"false"。
clientCertIDs	非必填。开启双向认证的客户端证书链。

扩展HTTP协议

参数名称	描述
protocol	HTTP
scheduler	非必填。负载均衡方法。支持"RoundRobin"、"LeastConnection"，默认"RoundRobin"。
timeout	非必填。HTTP服务端超时时间。默认"30"。
redirectPort	非必填。HTTPS重定向端口。
disableXForwardedForHeader	非必填。是否关闭XForwardedFor头。默认"false"。
enableXForwardedProtoHeader	非必填。是否开启XForwardedProto头。默认"false"。
enableKeepSession	非必填。是否开启会话保持。默认"false"。
keepSessionType	非必填。会话保持的Cookie处理方式，当且仅当开启会话保持时有效，支持"insert"/"rewrite"，默认"insert"。
keepSessionTimeout	非必填。会话保持超时时间。
KeepSessionCookieName	非必填。会话保持类型为"rewrite"时Cookie的名称。

扩展HTTPS协议

参数名称	描述
protocol	HTTPS
scheduler	非必填。负载均衡方法。支持"RoundRobin"、"LeastConnection"，默认 "RoundRobin"。
timeout	非必填。HTTP服务端超时时间。默认"30"。
serverCertIDs	必填。服务端证书ID链，逗号分隔。
encryptionType	非必填。加密选项，默认tls_cipher_policy_default。支持：tls_cipher_policy_default/tls_cipher_policy_1_1/tls_cipher_policy_1_2/tls_cipher_policy_1_2_secure/userDefind。
encryptionProtocols	非必填。加密协议，当encryptionType值为userDefind时协议类型列表，是以"tlsv10"、"tlsv11"、"tlsv12"三种协议组合形成的字符串列表。
appliedCiphers	非必填。加密套件。不同加密套件用冒号":"隔开。
dualAuth	非必填。是否开启双向认证。默认"false"。
clientCertIDs	非必填。开启双向认证的客户端证书链。
disableXForwardedForHeader	非必填。是否关闭XForwardedFor头。默认"false"。
enableXForwardedProtoHeader	非必填。是否开启XForwardedProto头。默认"false"。
enableKeepSession	非必填。是否开启会话保持。默认"false"。
keepSessionType	非必填。会话保持的Cookie处理方式，当且仅当开启会话保持时有效，支持"insert"/"rewrite"，默认"insert"。
keepSessionTimeout	非必填。会话保持超时时间。
KeepSessionCookieName	非必填。会话保持类型为"rewrite"时Cookie的名称。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 只有Spec中的端口协议为TCP时，才可以在为其添加TCP、TCP-SSL、HTTP、HTTPS扩展协议配置；如果端口在Spec中为UDP协议，则只能为其添加UDP扩展协议配置；
- 修改扩展协议类型，或是删除某个端口的扩展协议配置，将会导致应用型BLB对应端口的监听器重建，请谨慎操作。

示例

```
 service.beta.kubernetes.io/cce.listener.customized-listener: |-
    {
      "81":{
        "protocol":"TCP",
        "timeout":"31",
        "scheduler":"Hash"
      },
      "82":{
        "protocol":"UDP",
        "timeout":"31",
        "scheduler":"Hash"
      },
      "83":{
        "protocol":"HTTP",
        "redirectPort":"84",
        "timeout":"31",
        "scheduler":"LeastConnection",
        "enableKeepSession":"true",
        "keepSessionType":"rewrite",
        "disableXForwardedForHeader": "true",
        "enableXForwardedProtoHeader": "true"
      },
      "84":{
        "protocol":"HTTPS",
        "serverCertIDs":"cert-5bqpykggjzmi",
        "timeout":"31",
        "scheduler":"LeastConnection",
        "enableKeepSession":"true",
        "keepSessionType":"rewrite",
        "encryptionType": "userDefind",
        "encryptionProtocols": "tlsv10, tlsv11, tlsv12",
        "appliedCiphers": "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-
AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-SHA"
      },
      "85":{
        "protocol":"TCP-SSL",
        "serverCertIDs":"cert-5bqpykggjzmi",
        "timeout":"31",
        "scheduler":"LeastConnection",
        "encryptionType": "userDefind",
        "encryptionProtocols": "tlsv10, tlsv11, tlsv12",
        "appliedCiphers": "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-
AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-SHA"
      }
    }
```

示例 LoadBalancer Service

```
 apiVersion: v1
kind: Service
metadata:
 annotations:
   service.beta.kubernetes.io/cce.listener.customized-listener: |-
    {
      "81":{
        "protocol":"TCP",
        "timeout":"31",
        "scheduler":"Hash"
      },
      "82":{
        "protocol":"UDP",
        "timeout":"31",
        "scheduler":"Hash"
      },
      "83":{
        "protocol":"HTTP",
        "redirectPort":"84",
        "timeout":"31"
```

```
            "timeout": "31",
            "scheduler":"LeastConnection",
            "enableKeepSession":"true",
            "keepSessionType":"rewrite",
            "disableXForwardedForHeader": "true",
            "enableXForwardedProtoHeader": "true"
          },
          "84":{
            "protocol":"HTTPS",
            "serverCertIDs":"cert-5bqpykggjzmi",
            "timeout":"31",
            "scheduler":"LeastConnection",
            "enableKeepSession":"true",
            "keepSessionType":"rewrite",
            "encryptionType": "userDefind",
            "encryptionProtocols": "tlsv10, tlsv11, tlsv12",
            "appliedCiphers": "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-SHA"
          },
          "85":{
            "protocol":"TCP-SSL",
            "serverCertIDs":"cert-5bqpykggjzmi",
            "timeout":"31",
            "scheduler":"LeastConnection"
          }
        }
    name: service-example
    namespace: default
  spec:
    clusterIP: 172.16.27.138
    externalTrafficPolicy: Cluster
    ports:
      - name: nginx-1
        nodePort: 30710
        port: 81
        protocol: TCP
        targetPort: 80
      - name: nginx-2
        nodePort: 30792
        port: 82
        protocol: UDP
        targetPort: 80
      - name: nginx-3
        nodePort: 31906
        port: 83
        protocol: TCP
        targetPort: 80
      - name: nginx-4
        nodePort: 32514
        port: 84
        protocol: TCP
        targetPort: 80
      - name: nginx-5
        nodePort: 31749
        port: 85
        protocol: TCP
        targetPort: 80
      - name: nginx-6
        nodePort: 31569
        port: 86
        protocol: TCP
        targetPort: 80
    selector:
      app: nginx
    sessionAffinity: None
    type: LoadBalancer
```

设置服务器组动态权重 取值说明

添加该Annotation后，在LoadBalancer Service的ExternalTrafficPolicy为Local模式且未开启Pod直连时，CCE根据节点上的Pod数量来为BLB的后端服

务器设置权重；否则，CCE为每个后端节点设置均等权重。

这可以避免在ExternalTrafficPolicy为Local模式下产生Pod负载不均衡问题。

注意事项

- CCE不会自动添加该Annotation，用户需要按需自行添加。
- 在总体Pod数较少时且平均负载较高时，滚动更新Pod时可能会导致某些Pod负载短时过高；请避免在这种情况下使用该配置项。

示例

```
service.beta.kubernetes.io/cce-dynamic-backend-server-weight: 'true'
```

设置服务器组健康检查 取值说明

若设置，CCE将会给BLB对应的服务器组设置对应配置的健康检查；

若缺失，CCE将会给BLB对应的服务器组设置默认配置的健康检查。

注意事项

- 默认健康检查配置可能会导致检查失败，建议自定义健康检查配置，更多信息可参考负载均衡BLB健康检查介绍。

示例

```
 ##### LB Service 的各个端口可能使用不同协议，因此需要单独对每个端口来设置健康检查

###### 1、Service 监听端口为 80，健康检查协议为 TCP
service.beta.kubernetes.io/cce.listener.customized-health-check:  |-
    {
      "80":{
        "healthCheck":"TCP", #健康检查协议
        "healthCheckTimeoutInSecond":"3", #响应超时时间
        "healthCheckIntervalInSecond":"3", #健康检查间隔
        "healthCheckDownRetry":"3", #不健康阈值
        "healthCheckUpRetry": "3" #健康阈值
      }
    }

###### 2、Service 监听端口为 80，健康检查协议为 UDP
 service.beta.kubernetes.io/cce.listener.customized-health-check:  |-
    {
      "80":{
        "healthCheck":"UDP",
        "udpHealthCheckString":"\00\01\01\00\00\01\00\00\00\00\00\00\05baidu\03com\00\00\01\00\01", #检查字符串
        "healthCheckTimeoutInSecond":"3",
        "healthCheckIntervalInSecond":"3",
        "healthCheckDownRetry":"3",
        "healthCheckUpRetry": "3"
      }
    }

###### 3、Service 监听端口为 80，健康检查协议为 ICMP
 service.beta.kubernetes.io/cce.listener.customized-health-check:  |-
    {
      "80":{
        "healthCheck":"ICMP",
        "healthCheckTimeoutInSecond":"3",
        "healthCheckIntervalInSecond":"3",
        "healthCheckDownRetry":"3",
        "healthCheckUpRetry": "3"
      }
    }

###### 4、Service 监听端口为 80，健康检查协议为 HTTP
service.beta.kubernetes.io/cce.listener.customized-health-check:  |-
    {
      "80":{
        "healthCheck":"HTTP",
        "healthCheckHost": "baidu.com", #Host头域
        "healthCheckPort":"80", #检查端口
        "healthCheckUrlPath":"/", #检查路径
        "healthCheckTimeoutInSecond":"2",
        "healthCheckIntervalInSecond":"2",
        "healthCheckDownRetry":"3",
        "healthCheckUpRetry": "2",
        "healthCheckNormalStatus": "http_2xx|http_3xx" #正常状态码，可选项有“http_2xx”、“http_3xx”、“http_4xx”、“http_5xx”，支持多选
      }
    }
```

通过 CCE 使用 K8S Ingress

Ingress是Kubernetes提供的一种7层流量接入方式，它通过连接外部负载均衡与容器内部服务的方式对流量进行管理。相比LoadBalancer Service，Ingress可以简化主机对外开放端口的管理，同时利用外部负载的能力提供更完善的路由和安全规则，详情参考官网说明：Kubernetes Ingress。

Ingress主要由两部分组成：

- Ingress资源对象：提供Ingress Kubernetes对象，能够通过yaml进行创建和更新，将服务与域名对应起来。
- Ingress Controller：部署在集群中的公共组件，将Ingress资源对象的配置转化成外部负载均衡的配置，对负载均衡器进行管理和更新。

CCE提供了基于百度智能云负载均衡服务的CCE Ingress Controller的实现，用户在使用Ingress的同时将会联动创建和配置管理百度智能云负载均衡（应用型BLB）和弹性公网IP（EIP），从而实现对集群外部流量的接入和管理。Ingress的使用方式与原生的Kubernetes保持一致，用户可以通过

百度智能云控制台或者yaml创建和管理Ingress。

前提条件

- 集群已安装CCE Ingress Controller组件，更多信息请见CCE Ingress Controller 说明。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**网络>路由**”进入路由页面。
6. 单击“**新建Ingress**”按钮进入新建Ingress页面，进行基本信息配置：

配置项	必选/可选	配置说明
Ingress名称	必选	指定Ingress名称。
命名空间	必选	选择Ingress所在的命名空间。Ingress只能向所在命名空间的Service转发流量。
负载均衡器	必选	当前选择“**应用性BLB**”，系统将为Ingress创建或绑定应用性BLB实例。
访问类型	必选	指定Service的访问类型，支持**公网**和**内网**。
指定BLB	必选	可选新建或指定已有的应用型BLB。注意：一个BLB只能用于一个Ingress，不能复用。
指定EIP	必选	仅**访问类型**为**公网**时支持配置，可选新建或指定已有的EIP。注意：一个EIP只能用于一个Ingress，不能复用。
指定VPC子网	必选	选择当前Ingress关联创建的BLB所在的子网。如果不是新建BLB而是指定了已有BLB，该选项不会生效。

7. 下一步进行转发规则配置。

配置项	必选/可选	配置说明
更新策略	可选	指定开启BLB配置全量更新，更新Ingress时默认仅会更新对应BLB监听器的转发规则配置，开启后会同步更新BLB全部配置（如监听器、证书、健康检查、安全组）和绑定的EIP配置（如公网带宽）。
监听端口	必选	默认选择HTTP:80，支持勾选HTTPS:443。
证书	必选	仅**监听端口**勾选了**HTTPS:443**时支持配置，为HTTPS监听器绑定证书。
扩展域名	可选	为HTTPS监听器添加多条扩展域名。 • 域名：指定扩展的域名，支持精准域名和泛域名。 • 证书：选择百度智能云【安全认证>证书】管理中的证书信息。
转发规则	必选	支持配置多条转发规则。 • 监听协议：HTTP和HTTPS规则是互相独立的，如果需要在HTTP和HTTPS上都支持，需要分别在HTTP和HTTPS监听协议上都创建一次。 • 监听端口：跟随监听协议，HTTP为80，HTTPS为443。 • 域名：输入域名。 • URI：输入URI。 • 服务名称：目前仅支持NodePort或LoadBalancer类型的Service，必需和Ingress处于同一命名空间。 • 服务端口：选择Service暴露的端口。

8. 最后进行高级配置。

配置项	必选/可选	配置说明
超时时间	必选	配置Ingress等待后端服务器响应的超时时间。
HTTP重定向	必选	是否打开HTTP重定向，将HTTP请求转发到HTTPS端口上。该选项仅在勾选HTTPS后需要配置。
标签（Labels）	可选	为Ingress添加K8S标签。
注解（Annotations）	可选	为Ingress添加K8S Annotations。CCE定义的Annotation请参考BLB Ingress Annotation说明。

9. 单击“**完成**”按钮完成创建。

> **说明：**
>
> - 刚刚创建的Ingress，详情页和转发规则页都会显示“NotReady”，等待几分钟之后，Ingress完成配置，将会显示为“Ready”状态，此时Ingress已经可以使用。

Ingress使用示例

分别用以下yaml内容创建示例Deployment和Service资源：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-nginx-deployment
  labels:
    app: ingress-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ingress-nginx
  template:
    metadata:
      labels:
        app: ingress-nginx
    spec:
      containers:
      - name: nginx
        image: hub.baidubce.com/cce/nginx-ingress
        ports:
        - containerPort: 80

---
kind: Service
apiVersion: v1
metadata:
  name: hello-service
spec:
  selector:
    app: ingress-nginx
  type: NodePort
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 80

---
kind: Service
apiVersion: v1
metadata:
  name: world-service
spec:
  selector:
    app: ingress-nginx
  type: NodePort
  ports:
  - protocol: TCP
    port: 9000
    targetPort: 80
```

2.创建Ingress资源并配置以下转发规则：

- www.cce-ingress.com/hello/* -> hello-service:8000/hello/
- www.cce-ingress.com/world/* -> world-service:9000/world/

> **注意：**
>
> - Service对应后端服务必须支持转发策略的URI，如果要支持所有情况，需要设置为/*。

3.修改本地/etc/hosts，将www.cce-ingress.com指向EIP（例如：106.12.52.80）：

4.访问测试：

hello.index

通过YAML创建CCE_Ingress

本文档介绍如何通过YAML创建和管理CCE Ingress。

本文档主要包含以下内容：

1.部署CCE Ingress Controller

2.创建Deployment与Service

3.创建CCE Ingress

注意：一个CCE Ingress只能使用一个BLB，一个BLB也只能被一个CCE Ingress使用

部署CCE Ingress Controller

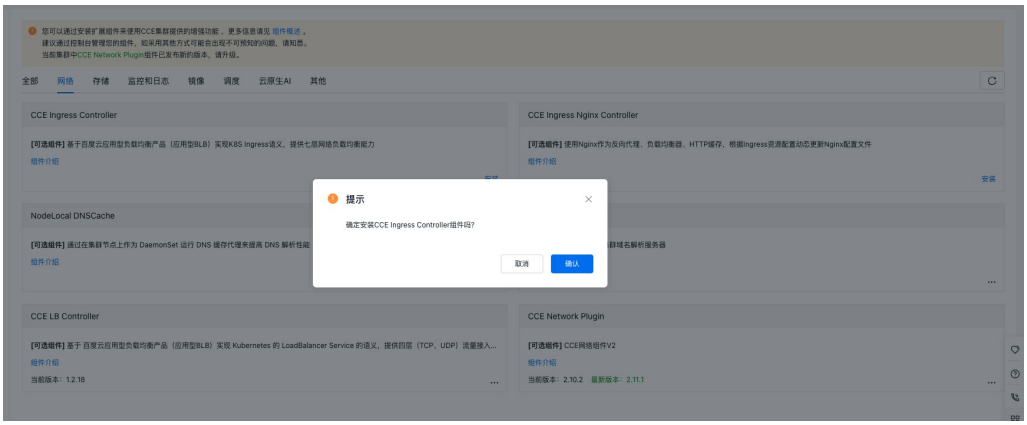
CCE Ingress使用百度智能云负载均衡产品（BLB）实现，亦称为BLB类型Ingress。

CCE Ingress的使用需要集群中部署CCE Ingress Controller组件。

CCE不会为集群默认安装Ingress Controller，用户可在组件管理中安装该组件。

1. 进入集群控制台“组件管理>网络”，找到CCE Ingress Controller组件。

2. 点击『安装』并确认。



3. 等待CCE Ingress Controller组件显示 "当前版本"，表示该组件安装成功。

创建示例Deployment和Service

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: ingress-nginx-deployment
  labels:
    app: ingress-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ingress-nginx
  template:
    metadata:
      labels:
        app: ingress-nginx
    spec:
      containers:
      - name: nginx
        image: hub.baidubce.com/cce/nginx-ingress
        ports:
        - containerPort: 80
---
kind: Service
apiVersion: v1
metadata:
  name: hello-service
spec:
  selector:
    app: ingress-nginx
  type: NodePort
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 80
---
kind: Service
apiVersion: v1
metadata:
  name: world-service
spec:
  selector:
    app: ingress-nginx
  type: NodePort
  ports:
  - protocol: TCP
    port: 9000
    targetPort: 80
```

创建Ingress对象

使用以下yaml内容创建名为helloworld的Ingress对象，并配置Ingress转发规则：

- www.cce-ingress.com/hello/ -> hello-service:8000/hello/
- www.cce-ingress.com/world/ -> world-service:9000/world/

可通过Ingress Annotations中的参数对Ingress进行配置，例如指定BLB或EIP。

Service对应后端服务必须支持转发策略的URI，如果要支持所有情况，建议设置为/*，这里对URI的路径要求很严格注意“/”。

```
  apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    cce.ingress.blb-backup-content: ""
    kubernetes.io/cce.ingress.blb-cert-id: ""
    kubernetes.io/cce.ingress.blb-id: ""
    kubernetes.io/cce.ingress.eip: ""
    kubernetes.io/cce.ingress.http-redirect: "false"
    kubernetes.io/cce.ingress.https: "false"
    kubernetes.io/cce.ingress.internal: "false"
    kubernetes.io/cce.ingress.timeout-in-seconds: "30"
    kubernetes.io/cce.ingress.vpc-subnet-id: ""
    kubernetes.io/ingress.class: cce
  name: helloworld
  namespace: default
spec:
  rules:
  - host: www.cce-ingress.com
    http:
      paths:
      - backend:
          service:
            name: hello-service
            port:
              number: 8000
        path: /hello/*
      - backend:
          service:
            name: world-service
            port:
              number: 9000
        path: /world/*
```

下文描述了创建CCE Ingress时，需要注意的事项。

Ingress Class

CCE Ingress Controller仅处理带有`kubernetes.io/ingress.class: cce`注释的Ingress，请确保Ingress带有该Annnotation。

HTTPS规则设置

CCE Ingress Controller默认将Ingress资源的规则创建为HTTP规则，不会为其配置HTTPS规则，如果要将某些规则配置为HTTPS规则，应该具有以下Annotation：

- 指定BLB使用的证书。

  ```
  kubernetes.io/cce.ingress.blb-cert-id: cert-id
  ```

- 显式声明开启HTTPS。

  ```
  kubernetes.io/cce.ingress.https: "true"
  ```

- 指明使用HTTPS的规则。如果Spec中的某些规则需要使用HTTPS，还需要添加到该Annotation中。

  ```
  kubernetes.io/cce.ingress.https-rules: >-
  [
     { "host":"ad.com",
       "http":{
          "paths":[
             {"path":"/*","backend":{"serviceName":"service-example","servicePort":80}}
          ]
       }
     }
  ]
  ```

完整的Annotation使用说明参考：BLB Ingress Annotation说明。

访问测试

修改本地/etc/hosts，将www.cce-ingress.com指向EIP（例如: 106.12.52.80），可在控制台或者集群内查看Ingress信息：

```
##### kubectl get ingress
NAME        HOSTS                 ADDRESS                  PORTS   AGE
helloworld  www.cce-ingress.com   10.0.3.251,106.12.52.80  80      6m34s
```

访问测试：

CCE 基于 nginx-ingress实现灰度发布

本文介绍如何利用百度云容器服务的Ingress功能，实现蓝绿发布。

背景信息：

灰度及蓝绿发布是为新版本创建一个与老版本完全一致的生产环境，在不影响老版本的前提下，按照一定的规则把部分流量切换到新版本，当新版本试运行一段时间没有问题后，将用户的全量流量从老版本迁移至新版本。

其中A/B测试就是一种灰度发布方式，一部分用户继续使用老版本的服务，将一部分用户的流量切换到新版本，如果新版本运行稳定，则逐步将所有用户迁移到新版本。

Ingress-Nginx Annotation 简介

CCE 基于 Nginx Ingress Controller 实现了项目的网关，作为项目对外的流量入口和项目中各个服务的反向代理。而 Ingress-Nginx 支持配置 Ingress Annotations 来实现不同场景下的灰度发布和测试，可以满足金丝雀发布、蓝绿部署与 A/B 测试等业务场景。

Nginx Annotations 支持以下 4 种 Canary 规则：

- **nginx.ingress.kubernetes.io/canary-by-header**：基于 Request Header 的流量切分，适用于灰度发布以及 A/B 测试。当 Request Header 设置为 always时，请求将会被一直发送到 Canary 版本；当 Request Header 设置为 never时，请求不会被发送到 Canary 入口；对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他金丝雀规则进行优先级的比较。
- **nginx.ingress.kubernetes.io/canary-by-header-value**：要匹配的 Request Header 的值，用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。当 Request Header 设置为此值时，它将被路由到 Canary 入口。该规则允许用户自定义 Request Header 的值，必须与上一个 annotation (即：canary-by-header）一起使用。
- **nginx.ingress.kubernetes.io/canary-weight**：基于服务权重的流量切分，适用于蓝绿部署，权重范围 0 - 100 按百分比将请求路由到 Canary Ingress 中指定的服务。权重为 0 意味着该金丝雀规则不会向 Canary 入口的服务发送任何请求。权重为 100 意味着所有请求都将被发送到 Canary 入口。
- **nginx.ingress.kubernetes.io/canary-by-cookie**：基于 Cookie 的流量切分，适用于灰度发布与 A/B 测试。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务的cookie。当 cookie 值设置为 always时，它将被路由到 Canary 入口；当 cookie 值设置为 never时，请求不会被发送到 Canary 入口；对于任何其他值，将忽略 cookie 并将请求与其他金丝雀规则进行优先级的比较。

canary规则优先级：canary-by-header - > canary-by-cookie - > canary-weight

安装nginx-ingress-controller

通过组件安装

进入“容器引擎”-“集群详情”，点击左侧导航栏“运维与管理 > 组件管理”，找到CCE Ingress Nginx Controller，点击安装；

配置Ingress Nginx Controller：填写IngressClass名称，选择命名空间，选择节点组，如果没有节点组点击“新建节点组”，配置污点设置、容器配额、容忍设置、服务访问；

点击确认，完成创建。

通过YAML安装

```
##### yaml文件内容见附录
kubectl apply -f ingress-nginx.yaml
kubectl apply -f ingress-nginx-service.yaml
```

部署production任务

1. 创建production应用资源

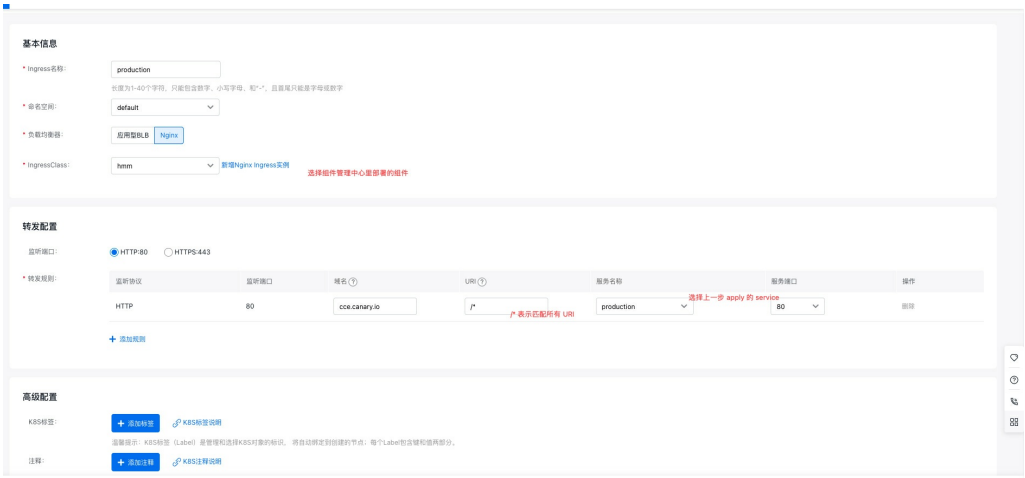
```
kubectl apply -f production.yaml
```

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: production
spec:
 replicas: 1
 selector:
  matchLabels:
   app: production
 template:
  metadata:
   labels:
    app: production
  spec:
   containers:
   - name: production
    image: registry.baidubce.com/jpaas-public/echoserver:1.10
    ports:
    - containerPort: 8080
    env:
     - name: NODE_NAME
      valueFrom:
       fieldRef:
        fieldPath: spec.nodeName
     - name: POD_NAME
      valueFrom:
       fieldRef:
        fieldPath: metadata.name
     - name: POD_NAMESPACE
      valueFrom:
       fieldRef:
        fieldPath: metadata.namespace
     - name: POD_IP
      valueFrom:
       fieldRef:
        fieldPath: status.podIP

---

apiVersion: v1
kind: Service
metadata:
 name: production
 labels:
  app: production
spec:
 ports:
 - port: 80
  targetPort: 8080
  protocol: TCP
  name: http
 selector:
  app: production
```

2. 创建 Production 版本的应用路由 (Ingress)

通过控制台创建 点击左侧导航栏“网络”后点击“路由”,界面点击“新建Ingress”；

如图完成Ingress的配置：



通过YAML创建

```
kubectl apply -f production.ingress.yaml
```

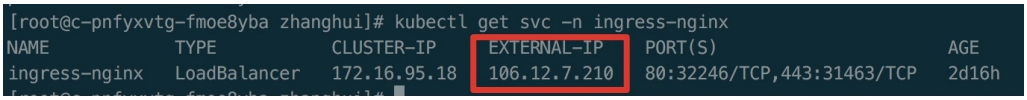
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: production
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: cce.canary.io
    http:
      paths:
      - backend:
          serviceName: production
          servicePort: 80
```

3、本机访问应用： 绑定hosts，集群内访问选择 clusterIP，集群外访问选择 externalIP （即 BLB IP）：

```
vi /etc/hosts
```

增加一行，如“106.12.7.210 cce.canary.io”

```
{ip} cce.canary.io
```



curl cce.canary.io 如下访问成功

```
~ curl cce.canary.io


Hostname: production-584f9c7fbc-57f9h

Pod Information:
        node name:      10.0.3.146
        pod name:       production-584f9c7fbc-57f9h
        pod namespace:  canary-demo
        pod IP: 172.23.0.17

Server values:
        server_version=nginx: 1.13.3 - lua: 10008

Request Information:
        client_address=172.23.0.9
        method=GET
        real path=/
        query=
        request_version=1.1
        request_scheme=http
        request_uri=http://cce.canary.io:8080/

Request Headers:
        accept=*/*
        host=cce.canary.io
        user-agent=curl/7.64.1
        x-forwarded-for=172.23.0.1
        x-forwarded-host=cce.canary.io
        x-forwarded-port=80
        x-forwarded-proto=http
        x-real-ip=172.23.0.1
```

创建canary版本任务

1. 创建canary版本应用资源

```
kubectl apply -f canary.yaml
```

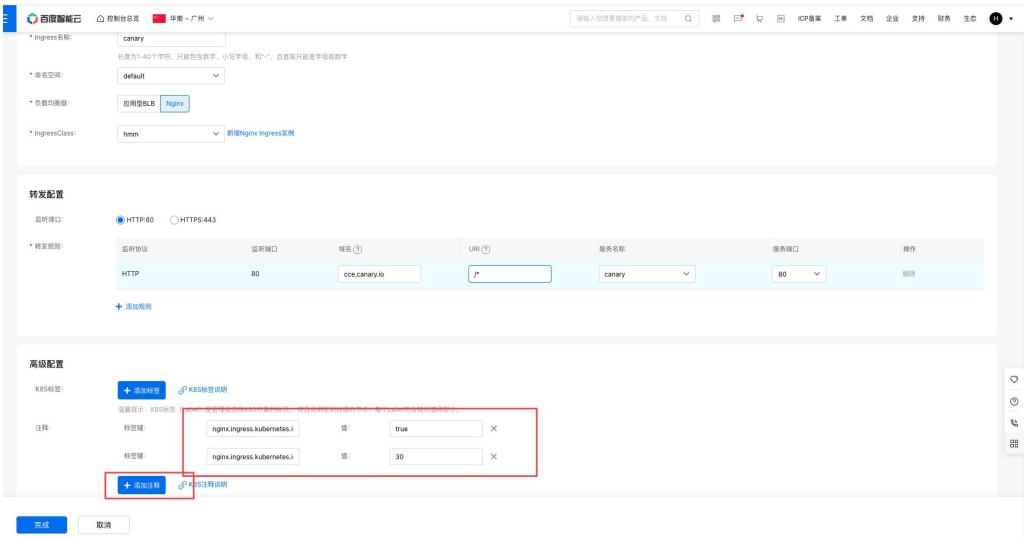
```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: canary
spec:
 replicas: 1
 selector:
  matchLabels:
   app: canary
 template:
  metadata:
   labels:
    app: canary
  spec:
   containers:
   - name: canary
    image: registry.baidubce.com/jpaas-public/echoserver:1.10
    ports:
    - containerPort: 8080
    env:
     - name: NODE_NAME
      valueFrom:
       fieldRef:
        fieldPath: spec.nodeName
     - name: POD_NAME
      valueFrom:
       fieldRef:
        fieldPath: metadata.name
     - name: POD_NAMESPACE
      valueFrom:
       fieldRef:
        fieldPath: metadata.namespace
     - name: POD_IP
      valueFrom:
       fieldRef:
        fieldPath: status.podIP

---

apiVersion: v1
kind: Service
metadata:
 name: canary
 labels:
  app: canary
spec:
 ports:
 - port: 80
  targetPort: 8080
  protocol: TCP
  name: http
 selector:
  app: canary
```

2. 基于权重创建 canary 版本的应用路由 (Ingress) 通过控制台创建

点击左侧导航栏“网络”后点击“路由”,界面点击“新建Ingress”；

在高级设置-注释中点击“+增加注释”, 增加以下标签键：

```
##### 高级配置中的标签设置如下：
nginx.ingress.kubernetes.io/canary: "true"
nginx.ingress.kubernetes.io/canary-weight: "30"
```



通过YAML创建

```
kubectl apply -f canary.ingress.yaml -n canary-demo
```

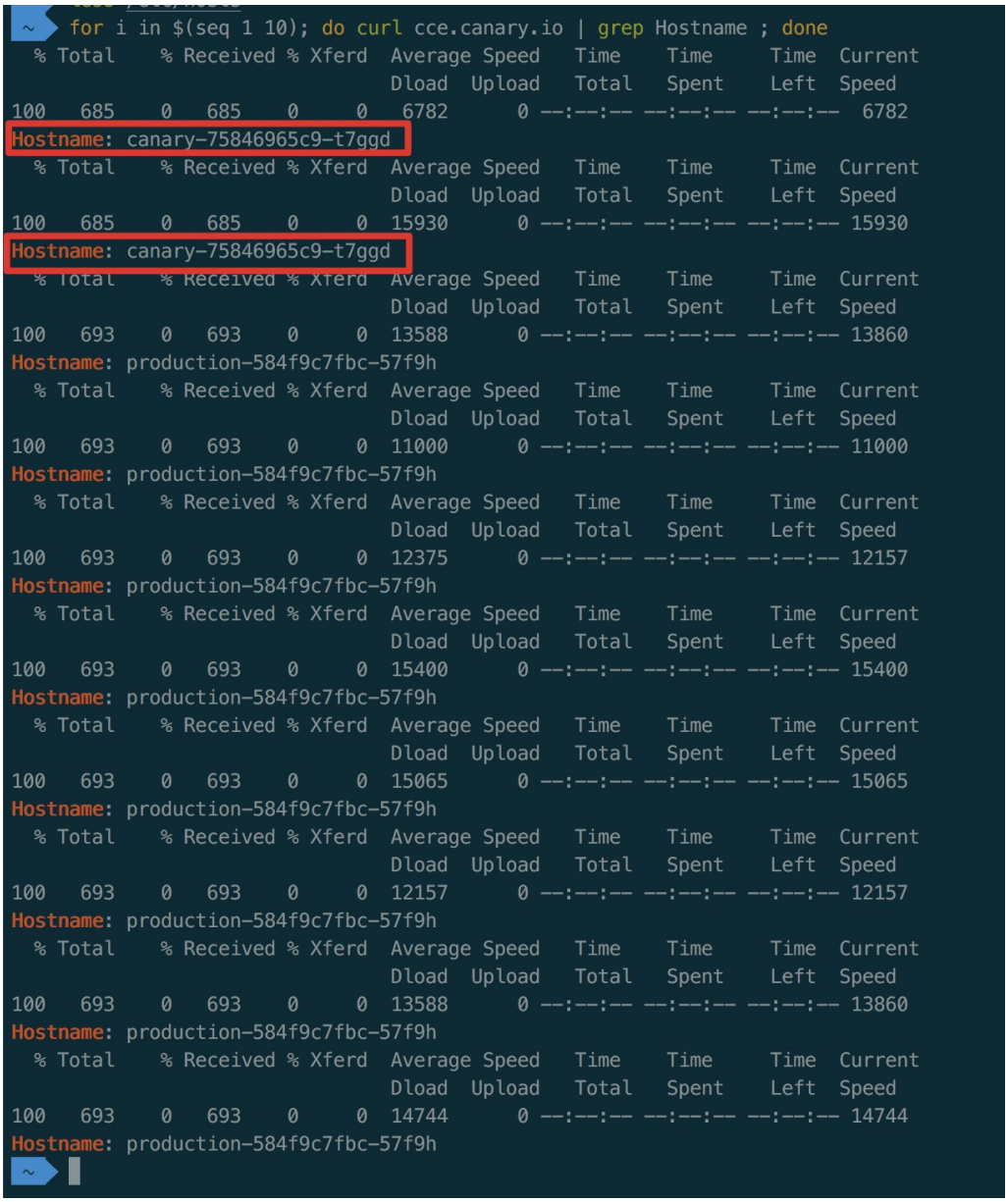
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: canary
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "30"
spec:
  rules:
  - host: cce.canary.io
    http:
      paths:
      - backend:
          serviceName: canary
          servicePort: 80
```

3. 访问应用域名验证

```
for i in $(seq 1 10); do curl cce.canary.io | grep Hostname ; done
```

如下图，流量部分打入canary

应用的 Canary 版本基于权重 (30%) 进行流量切分后，访问到 Canary 版本的概率接近 30%，流量比例可能会有小范围的浮动，属正常现象



附录：ingress-nginx相关yaml文件

- ingress-nginx.yaml

```
 apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: tcp-services
  namespace: ingress-nginx
```

```
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: udp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: nginx-ingress-clusterrole
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - endpoints
      - nodes
      - pods
      - secrets
    verbs:
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - "extensions"
      - "networking.k8s.io"
    resources:
      - ingresses
    verbs:
```

```
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "extensions"
      - "networking.k8s.io"
    resources:
      - ingresses/status
    verbs:
      - update

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: nginx-ingress-role
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - pods
      - secrets
      - namespaces
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      - configmaps
    resourceNames:
      # Defaults to "<election-id>-<ingress-class>"
      # Here: "<ingress-controller-leader>-<nginx>"
      # This has to be adapted if you change either parameter
      # when launching the nginx-ingress-controller.
      - "ingress-controller-leader-nginx"
    verbs:
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - endpoints
    verbs:
      - get

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: nginx-ingress-role-nisa-binding
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: nginx-ingress-role
subjects:
```

```
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: nginx-ingress-clusterrole-nisa-binding
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: nginx-ingress-clusterrole
subjects:
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/part-of: ingress-nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/part-of: ingress-nginx
      annotations:
        prometheus.io/port: "10254"
        prometheus.io/scrape: "true"
    spec:
      # wait up to five minutes for the drain of connections
      terminationGracePeriodSeconds: 300
      serviceAccountName: nginx-ingress-serviceaccount
      nodeSelector:
        kubernetes.io/os: linux
      containers:
        - name: nginx-ingress-controller
          image: registry.baidubce.com/jpaas-public/nginx-ingress-controller:0.30.0
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
            - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
            - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx
            - --annotations-prefix=nginx.ingress.kubernetes.io
          securityContext:
            allowPrivilegeEscalation: true
            capabilities:
              drop:
                - ALL
              add:
                - NET_BIND_SERVICE
            # www-data -> 101
            runAsUser: 101
```

```
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        ports:
          - name: http
            containerPort: 80
            protocol: TCP
          - name: https
            containerPort: 443
            protocol: TCP
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 10
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 10
        lifecycle:
          preStop:
            exec:
              command:
                - /wait-shutdown

---

apiVersion: v1
kind: LimitRange
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  limits:
  - min:
      memory: 90Mi
      cpu: 100m
    type: Container
```

- ingress-nginx-service.yaml

```
  kind: Service
apiVersion: v1
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  externalTrafficPolicy: Cluster
  type: LoadBalancer
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
```

使用 NGINX Ingress

本文介绍使用Nginx Ingress作为Ingress的实现方式。

介绍

Kubernetes Ingress除了CCE Ingress实现之外，也可以使用Kubernetes社区的Nginx Ingress实现。

Nginx支持反向代理与负载均衡器等功能。

Nginx Ingress Controller是Ingress控制器，使用Nginx实现反向代理功能，并通过解析集群中的Ingress来配置转发规则。

相比CCE Ingress，Nginx Ingress支持更多、更细粒度的规则配置，使用更加灵活。

名词解释

Nginx Ingress: 使用Nginx Ingress Controller来实现规则的K8s Ingress资源。

Nginx Ingress Controller组件：在CCE组件管理界面部署的Nginx Ingress控制器。

Nginx Ingress Controller实例：Nginx Ingress Controller组件既参与规则配置又参与数据转发，集群中可以部署多套组件来实现不同的流量入口配置。集群中部署的每套组件我们称之为Nginx Ingress Controller实例。

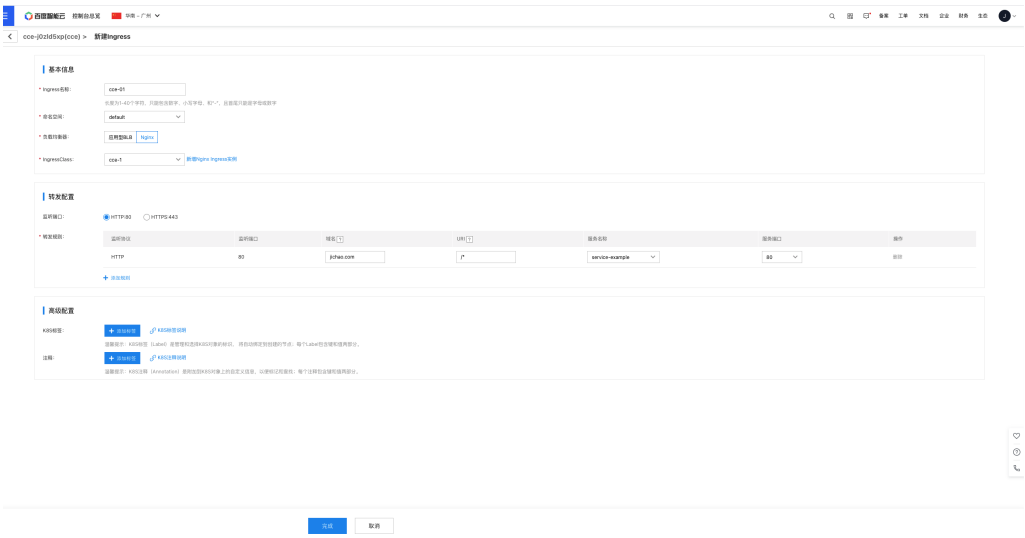
操作步骤

CCE提供了产品化能力，我们需要首先创建1个节点组以供后续Nginx Ingress Controller部署，完成Nginx Ingress Controller组件部署，然后再创建Ingress资源。

创建节点组 目前Nginx Ingress Controller采用DaemonSet方式部署，并部署于节点组中。
我们首先需要创建一个节点组。

安装Nginx Ingress Controller组件 节点组创建完成之后，我们即可向节点组中部署Nginx Ingress Controller组件。

创建 Nginx Ingress 资源 最后，我们创建Ingress资源并等待资源同步完成。

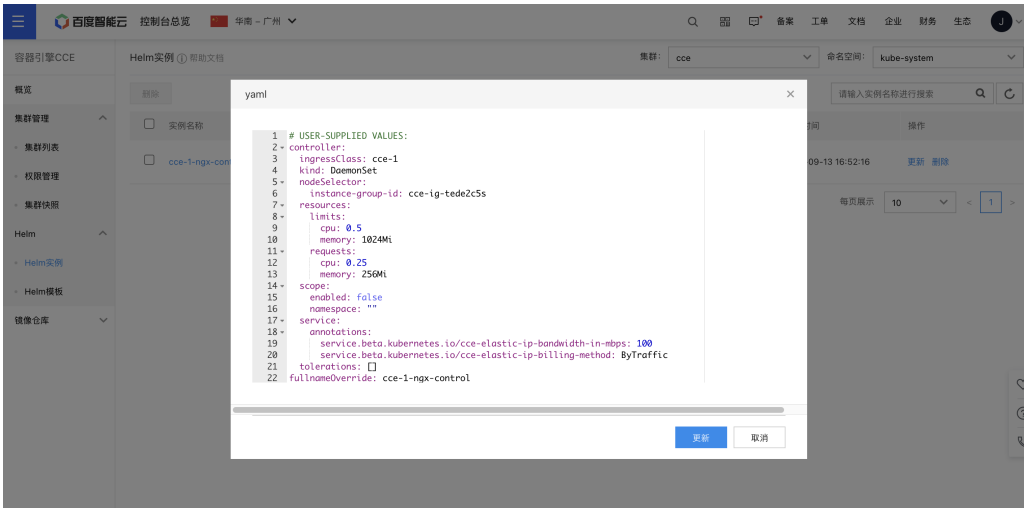


组件更多参数配置

CCE控制台仅支持一些最重要的组件部署参数。

如果希望对更多的参数进行配置，则在控制台“Helm”进行设置。

Nginx Ingress Controller对应的实例名称以部署组件时设置的ingress-class为前缀。



NGINX Ingress 使用配置参考

对于 cce-ingress-nginx-controller 组件有 2 处位置可以对组件参数或 Ingress 资源进行配置。

我们可以在组件的 ConfigMap 配置组件的全局运行参数，或是在 Ingress Annotation 中对某个 Ingress 或是 Host、Rule 的配置进行修改。

Ingress NGINX Controller ConfigMap

在集群中的每个 cce-ingress-nginx-controller 插件实例均配有对应的 ConfigMap，其配置项作用于所属插件整体。

组件会使用 ConfigMap 中的配置来覆盖组件的默认配置。

在集群中，该 ConfigMap 存在于 kube-system 命名空间并以 {ingressclass}-ngx-control-controller 命名。

ConfigMap 中配置项的内容均为 key-values 字符串：

- 对于数字和 Bool 类型 Value，需要对其添加引号，如："true"，"101"
- 对于数组 Value，使用逗号分隔的字符串

完整 ConfigMap 参数请参考 ConfigMap - NGINX Ingress Controller

示例 以下是一个集群中 cce-ingress-nginx-controller ConfigMap 以供参考

```
  apiVersion: v1
data:
  compute-full-forwarded-for: 'true'
  forwarded-for-header: X-Forwarded-For
  use-forwarded-headers: 'true'
  keep-alive-requests: '10000'
  log-format-upstream: >-
    {"time": "$time_iso8601", "remote_addr": "$proxy_protocol_addr",
    "x_forward_for": "$proxy_add_x_forwarded_for", "request_id": "$req_id",
    "remote_user": "$remote_user", "bytes_sent": $bytes_sent, "request_time":
    $request_time, "status": $status, "vhost": "$host", "request_proto":
    "$server_protocol", "path": "$uri", "request_query": "$args",
    "request_length": $request_length, "duration": $request_time,"method":
    "$request_method", "http_referrer": "$http_referer", "http_user_agent":
    "$http_user_agent" }
  max-worker-connections: '65536'
  upstream-keepalive-connections: '200'
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: cce-ngx-control
    meta.helm.sh/release-namespace: kube-system
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: cce-ngx-control
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: cce-ingress-nginx-controller
    app.kubernetes.io/version: 0.45.0
    helm.sh/chart: cce-ingress-nginx-controller-3.29.0
  name: cce-ngx-control-controller
  namespace: kube-system
```

NGINX Ingress Annotation

NGINX Ingress 中的 Annotation 中的配置可以对 Ingress、Host 或是某条 Rule 的默认行为进行修改。

完整的 Ingress Annotation 配置请参考 Annotations - NGINX Ingress Controller

示例 以下是一个集群中的 Ingress 资源内容以供参考：

```
  apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    cce.ingress.blb-backup-content: ''
    kubernetes.io/ingress.class: cce
    nginx.ingress.kubernetes.io/use-regex: 'true'
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
  name: test
  namespace: default
spec:
  rules:
    - host: a.com
      http:
        paths:
          - backend:
              service:
                name: service-example
                port:
                  number: 80
            path: /*
```

常见使用场景

以下说明常用的使用场景及其相关配置

使用 X-Forwarded-For Header 记录源 IP 当我们希望请求经过 Ingress 的时候，附加上请求的原始 ClientIP 时，可以按以下配置进行修改：

```
 ##### 配置位于 ConfigMap
compute-full-forwarded-for: 'true'    # 计算完整 X-Forwarded-For Header，将源 IP 添加在 Header 之后，而非直接覆盖原值
forwarded-for-header: X-Forwarded-For # 使用 X-Forwarded-For Header 来标识源 IP
use-forwarded-headers: 'true'         # 沿用上游请求的 X-Forwarded-* 相关 Header
```

Ingress 同时接收 HTTP 与 HTTPS 请求 NGINX Ingress 默认对配置 HTTPS 的域名的 HTTP 请求返回 308 响应，为 Ingress 资源以下 Annotation 之后 HTTPS 跳转会关闭，HTTP 可以直接访问：

```
 ##### 配置位于 Ingress Annotation
nginx.ingress.kubernetes.io/ssl-redirect: "false" # 关闭 SSL 重定向
```

URL 重写 Ingress 结合 Annotation 可以实现 URL 重写。

在正则匹配提取时，*$N* 代表匹配 URL 正则表达式模板中的第 N 个括号中匹配到的内容。

以下举例，假设我们希望将某个 Ingress 下的 ^/nginx/(.) URL 向后端服务转发时去除 /nginx 前缀，将某个 Ingress 下的 ^/tomcat/(.) URL 向后端服务转发时去除 /tomcat 前缀，那么可以在 Ingress 中添加如下 Annotation：

```
 ##### 配置位于 Ingress Annotation
nginx.ingress.kubernetes.io/configuration-snippet: |
    rewrite ^/nginx/(.*) /$1 break;
    rewrite ^/tomcat/(.*) /$1 break;
```

BLB Ingress Annotation说明

BLB Ingress Annotation说明

在使用CCE提供的**BLB类型**的Ingress资源时，可以通过在Annotation中添加注解的方式来使用Ingress的一些高级配置。

Annotation位于Ingress完整YAML的以下位置：

```
 apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 annotations:
   kubernetes.io/cce.ingress.blb-cert-id: cert-5bqpykggjzmi
   kubernetes.io/cce.ingress.blb-id: lb-13ddfb9e
   kubernetes.io/cce.ingress.eip: 106.13.169.172
   kubernetes.io/ingress.class: cce
……
```

Ingress可以使用以下高级配置项：

指定当前Ingress由CCE托管

取值说明

若值为`cce`，表示当前Ingress由CCE负责实现；

若缺失，或值不是`cce`，则CCE不会处理该Ingress。

注意事项

- 通过CCE控制台创建的BLB类型Ingress会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行添加该Annotation。

示例

`kubernetes.io/ingress.class: cce`

删除Ingress时保留应用型BLB/EIP资源

取值说明

若值为`true`，则在该Ingress资源对象被删除时，关联的应用型BLB和EIP资源不会被删除；

若缺失，或值是`false`，则在该Ingress资源对象被删除时，关联的应用型BLB和EIP资源将会被删除。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 如果在创建时忘记添加该Annotation，可以在删除前补上这一条，在删除前添加依然有效。

示例

`kubernetes.io/cce.ingress.keep-iaas-resource: "true"`

设置Ingress关联的BLB名称

取值说明

用于设置该Ingress关联的BLB的名称，否则名称使用生成的默认使用默认名称格式（集群ID/ingress/命名空间/Ingress名称）来创建BLB；

有时，默认名称会超出最长限制（64个字符），此时需要手动设置该Annotation的值来指定BLB的名称。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 不要和已有BLB的名称相同，否则容易混淆，导致误用BLB的情况发生。

示例

`kubernetes.io/cce.ingress.blb-name: cce-test-name`

为Ingress分配IPv6地址

取值说明

若值为`true`时，会为关联的应用型BLB分配IPv6地址；

若值是`false`，CCE不会为其分配IPv6地址，并且会摘除应用型BLB上绑定的IPv6地址。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加。

示例

`kubernetes.io/cce.ingress.enable-ipv6: "true"`

指定Ingress使用的应用型BLB

取值说明

若设置，CCE将在指定的应用型BLB上配置Ingress规则；

若缺失，CCE会新建一个应用型BLB，并自动设置该Annotation为新建应用型BLB的ID。

注意事项

- 只支持应用型BLB；
- 需要自行先创建好应用型BLB，BLB应在集群所在的VPC内；
- 通过CCE控制台创建的BLB类型Ingress，如果选择了某个已有应用型BLB，会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation；
- 修改或删除可能会导致BLB泄露，修改Ingress资源对象时不要覆盖或丢失该Annotation。

示例

`kubernetes.io/cce.ingress.blb-id: lb-12345678`

设置Ingress仅供VPC内网访问

取值说明

若值为`true`，CCE不会为其创建和设置EIP地址；

若缺失，或值是`false`，CCE会为应用型BLB分配和绑定EIP。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，当选择内网访问类型时，会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation；
- 如果应用型BLB上已经绑定了EIP，在添加该Annotation后，EIP不会被后台解绑，用户需要自行解绑。

示例

`kubernetes.io/cce.ingress.internal: "true"`

新建Ingress指定新建BLB所在子网

取值说明

若值为集群所在VPC的有效子网ID，则CCE会在需要新建应用型BLB时，将应用型BLB创建在该子网中；

若缺失，CCE将会自行选择创建应用型BLB的子网。

注意事项

- 通过CCE控制台创建的BLB类型Ingress会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation。

示例

`kubernetes.io/cce.ingress.vpc-subnet-id: sbn-adfEsDzs`

新建Ingress指定使用EIP

取值说明

若值为某EIP的地址，CCE将会把该EIP绑定在Ingress对应的应用型BLB上；

若缺失，CCE会默认新建一个EIP，并自动设置该Annotation。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，在指定公网IP后会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation；
- 修改或删除可能会导致EIP泄漏，修改Ingress资源对象时不要覆盖或丢失该Annotation；
- 如果Ingress上已经使用了某个EIP，更改该Annotation不会更换应用型BLB上绑定的EIP地址，需要用户自行换绑。

示例

`kubernetes.io/cce.ingress.eip: 1.2.3.4`

新建Ingress指定新建EIP带宽

取值说明

设置新建的Ingress使用的EIP带宽；

若该Annotation缺失，则默认新建EIP带宽为100Mbps;

取值范围为[1, 5000]。

示例

`kubernetes.io/cce.ingress.eip-bandwidth-in-mbps: "99"`

- 指定计费方式：

支持`ByTraffic`（按流量计费）和`ByBandwidth`（按带宽计费）。对应的Annotation为`kubernetes.io/cce.ingress.eip-billing-method`。

- 指定线路类型：

支持增强型 BGP 与标准型 BGP 两种线路类型，CCE 默认支持对应的Annotation为`kubernetes.io/cce.ingress.eip-route-type`，其中 `BGP_S` 代表增强型 BGP，`BGP` 代表标准型 BGP，`ChinaTelcom` 代表电信单线，`ChinaUnicom` 代表联通单线，`ChinaMobile` 代表移动单线，使用示例如下：

示例

```
 ##### 增强型 BGP
kubernetes.io/cce.ingress.eip-route-type:'BGP_S'
```

注意事项

- 新建EIP都是后付费类型，暂不支持使用Annotation指定创建预付费EIP；
- 只针对新建EIP有效；
- 当带宽额度大小不为合法数字时，会上报警告事件；
- 当您没有添加带宽额度注解时，CCE 会为您默认创建并绑定带宽额度大小为100，计费方式为按流量计费的标准型 EIP；
- 境外部分区域暂不支持增强型 BGP 创建，详情请见；
- 带宽额度在不同线路类型和计费类型组合下，取值范围不同，如下表所示：

线路类型	标准型 BGP	增强型 BGP
流量计费	[1, 200]	[100, 1000]
带宽计费	[1, 500]	[100, 5000]

- 在不满足以下四种条件的情况下，会导致创建 EIP 失败：

线路类型	标准型 BGP 使用说明	增强型 BGP 使用说明
流量计费	需要添加如下带宽范围注解，且注解取值范围是[1, 200] kubernetes.io/cce.ingress.eip-bandwidth-in-mbps: "99" kubernetes.io/cce.ingress.eip-billing-method: ByTraffic kubernetes.io/cce.ingress.eip-route-type: BGP	需同时添加如下带宽范围和线路类型注解，且注解取值范围是[100, 1000] kubernetes.io/cce.ingress.eip-bandwidth-in-mbps: "200" kubernetes.io/cce.ingress.eip-billing-method: ByTraffic kubernetes.io/cce.ingress.eip-route-type: BGP_S
带宽计费	需同时添加如下带宽范围和计费类型注解，且注解取值范围是[1, 500] kubernetes.io/cce.ingress.eip-bandwidth-in-mbps: "99" kubernetes.io/cce.ingress.eip-billing-method: ByBandwidth kubernetes.io/cce.ingress.eip-route-type: BGP	需同时添加如下带宽范围、线路类型和计费类型注解，且注解取值范围是[100, 5000] kubernetes.io/cce.ingress.eip-bandwidth-in-mbps: "5000" kubernetes.io/cce.ingress.eip-billing-method: ByBandwidth kubernetes.io/cce.ingress.eip-route-type: BGP_S

新建Ingress指定新建EIP计费方式

取值说明

若值为`ByBandwidth`，在新建EIP时按带宽计费；

若值为`ByTraffic`，在新建EIP时按流量计费；

若缺失，默认采用 `ByBandwidth` 计费方式。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 仅在新建EIP时有效，不会修改已有EIP的计费方式。

示例

`kubernetes.io/cce.ingress.eip-billing-method: ByBandwidth`

新建Ingress指定访问超时时间

取值说明

在新建Ingress时配置HTTP和HTTPS请求的超时时间。

注意事项

- 取值范围：[1, 3600]；
- 通过CCE控制台创建的BLB类型Ingress，在设置超时时间后会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation。

示例

`kubernetes.io/cce.ingress.timeout-in-seconds: "20"`

限制BLB默认挂载的后端数

取值说明

该Annotation用于限制Ingress挂载的后端数目。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加。

示例

`kubernetes.io/cce.ingress.max-backend-count: 50`

是否支持HTTPS

取值说明

若该Annotation值为`true`，CCE会为其创建HTTPS相关配置；

若缺失，或值为`false`，CCE不会管理关联应用型BLB上的HTTPS配置。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，在开启HTTPS后会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation；
- 若缺失，或值为`false`，CCE不会管理关联应用型BLB上的HTTPS配置，也就是说，如果应用型BLB上已经有HTTPS监听器和转发规则，此时控

制器不会删除这些HTTPS配置，需要用户手动处理。

示例

`kubernetes.io/cce.ingress.https: "true"`

新建Ingress支持HTTP重定向

取值说明

若值为`true`，CCE会配置将访问HTTP端口的流量转发到HTTPS端口上去。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，在开启HTTP重定向后会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation。

示例

`kubernetes.io/cce.ingress.http-redirect: "true"`

新建Ingress指定HTTPS证书

取值说明

若在新建Ingress时希望使用HTTPS，则同时需要设定HTTPS证书；

该Annotation的值应为百度云上已经申请到的证书ID。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，在选择HTTPS证书后会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation。

示例

`kubernetes.io/cce.ingress.blb-cert-id: xs-asdfDESz`

设置HTTP、HTTPS规则

取值说明

对于Ingress Spec中的规则，CCE默认将其创建为HTTP转发规则，而不是HTTPS转发规则。

使用`kubernetes.io/cce.ingress.http-rules`和`kubernetes.io/cce.ingress.https-rules`这两条Annotation可以显式指明某条规则是HTTP还是HTTPS规则。

注意事项

- 通过CCE控制台创建的BLB类型Ingress，会自带该Annotation；
- 如果在集群中使用YAML自行创建BLB类型Ingress，需要自行按需添加该Annotation。

示例

```
 kubernetes.io/cce.ingress.http-rules: >-
    [
       { "host":"ad.com",
         "http":{
            "paths":[
               {"path":"/*","backend":{"serviceName":"service-example","servicePort":80}}
            ]
         }
       }
    ]

kubernetes.io/cce.ingress.https-rules: >-
    [
       { "host":"ad.com",
         "http":{
            "paths":[
               {"path":"/*","backend":{"serviceName":"service-example","servicePort":80}}
            ]
         }
       }
    ]
```

使用专属BLB集群 取值说明

当用户存在BLB专属集群时，后端会优先将应用型BLB创建在BLB专属集群上。

使用以下Annotation可以指定是否使用专属集群，以及使用哪个专属集群：

- 使用共享集群创建应用型BLB

使用该Annotation`kubernetes.io/no-use-blb-cluster-l4: "true"`之后，应用型BLB将建在L4共享集群上；

使用该Annotation`kubernetes.io/no-use-blb-cluster-l7: "true"`之后，应用型BLB将建在L7共享集群上。

- 指定创建在专属集群上

使用该Annotation`kubernetes.io/blb-cluster-l4-id: "test"`之后，应用型BLB将创建在指定的专属L4 BLB集群上；

使用该Annotation`kubernetes.io/blb-cluster-l7-id: "test"`之后，应用型BLB将创建在指定的专属L7 BLB集群上。

注意事项

- 有了专属集群之后，如果没有显式添加Annotation，将由后台自动选择创建在哪个专属BLB集群上。

示例

```
 ##### 仅为 Ingress 指定 4 层 BLB 专属集群
kubernetes.io/blb-cluster-l4-id: blb-cluster-l4-id
kubernetes.io/no-use-blb-cluster-l7: true

##### 仅为 Ingress 指定 7 层 BLB 专属集群
kubernetes.io/no-use-blb-cluster-l4: true
kubernetes.io/blb-cluster-l7-id: blb-cluster-l7-id

##### 为 Ingress 指定 4 层和 7 层 BLB 专属集群
kubernetes.io/blb-cluster-l4-id: blb-cluster-l4-id
kubernetes.io/blb-cluster-l7-id: blb-cluster-l7-id

##### 显式声明 Ingress 不使用任何专属集群
kubernetes.io/no-use-blb-cluster-l4: true
kubernetes.io/no-use-blb-cluster-l7: true
```

设置请求调度方法

取值说明

若值为`RoundRobin`，则使用加权轮询法调度请求；

若值为`LeastConnection`，则使用最小连接数法法调度请求；

若缺失，默认采用`RoundRobin`调度方法。

设置HTTP调度方法使用`kubernetes.io/cce.ingress.blb.listener.http.scheduler`，设置HTTPS调度方法使用`kubernetes.io/cce.ingress.blb.listener.https.scheduler`。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 仅在新建应用型BLB监听器时有效。

示例

```
 ##### HTTP
kubernetes.io/cce.ingress.blb.listener.http.scheduler: "LeastConnection"

##### HTTPS
kubernetes.io/cce.ingress.blb.listener.https.scheduler: "LeastConnection"
```

取消添加X-Forwarded-For(XFF)头 取值说明

若值为`true`，应用型BLB自动在请求头中添加X-Forwarded-For请求头；

若值为`false`，应用型BLB不会自动在请求头中添加X-Forwarded-For请求头；

若缺失，行为和`true`一致，应用型BLB自动在请求头中添加X-Forwarded-For请求头。

设置HTTP请求使用`kubernetes.io/cce.ingress.blb.listener.http.enable-x-forwarded-for-header`，设置HTTPS请求使用`kubernetes.io/cce.ingress.blb.listener.https.enable-x-forwarded-for-header`。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；

- 仅在新建应用型BLB监听器时有效。

示例

```
 kubernetes.io/cce.ingress.blb.listener.http.enable-x-forwarded-for-header: "false",
kubernetes.io/cce.ingress.blb.listener.https.enable-x-forwarded-for-header: "false"
```

配置实时更新 取值说明

若值为true，CCE会在修改监听器配置、健康检查规则、EIP 带宽配置之后，立即更新 BLB 和 EIP 相关配置。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；

示例

```
kubernetes.io/cce.ingress.update-in-time: "true"
```

使用 Pod直连模式 取值说明

开启直连模式需要进行以下两步：

1. 修改 cce-ingress-controller Deployment，添加以下环境变量，如果已经有了该环境变量，就不必再添加了

设置HTTP请求使用`kubernetes.io/cce.ingress.blb.listener.http.enable-x-forwarded-for-header`，设置HTTPS请求使用`kubernetes.io/cce.ingress.blb.listener.https.enable-x-forwarded-for-header`。

2. 为新建 Ingress 添加以下 Annotation

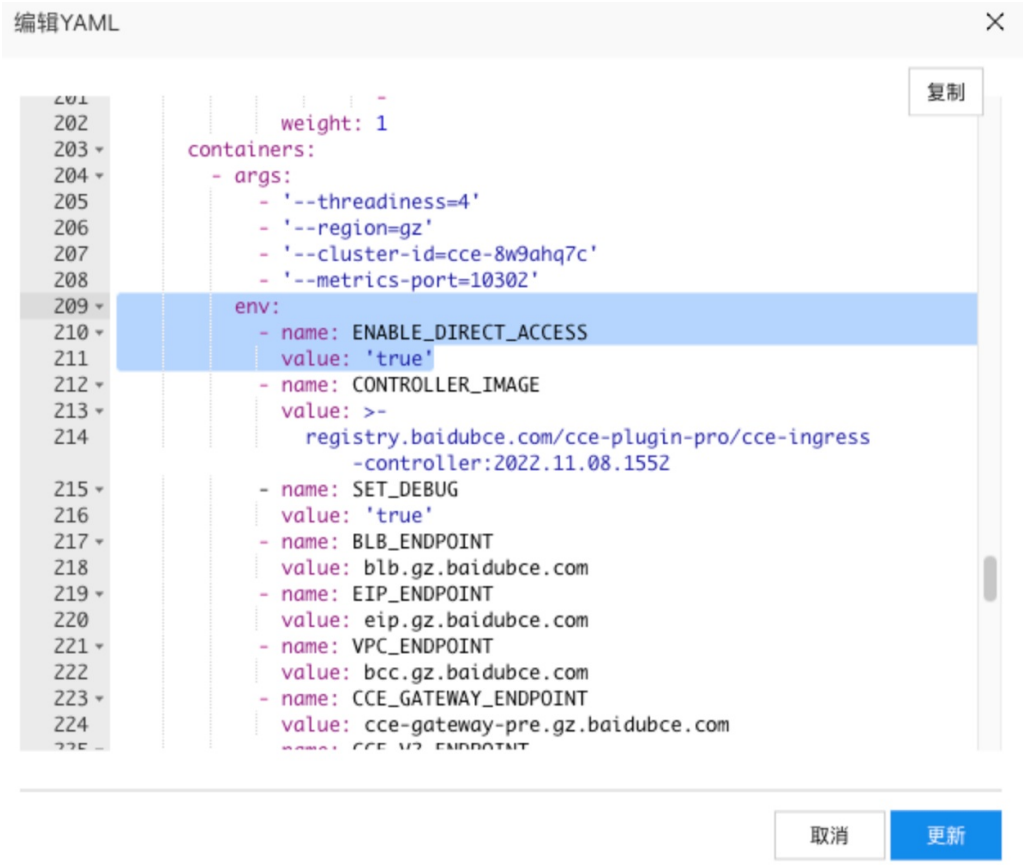
`kubernetes.io/cce.ingress.network-mode: direct-access`

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 仅在新建 Ingress 时有效；
- 使用 Pod 直连时，BLB 的后端地址出现在 IP 组中，IP组在服务器组标签页的右边。如果看不到，说明还没有开启IP组白名单，需要联系BLB负责同学处理。

示例

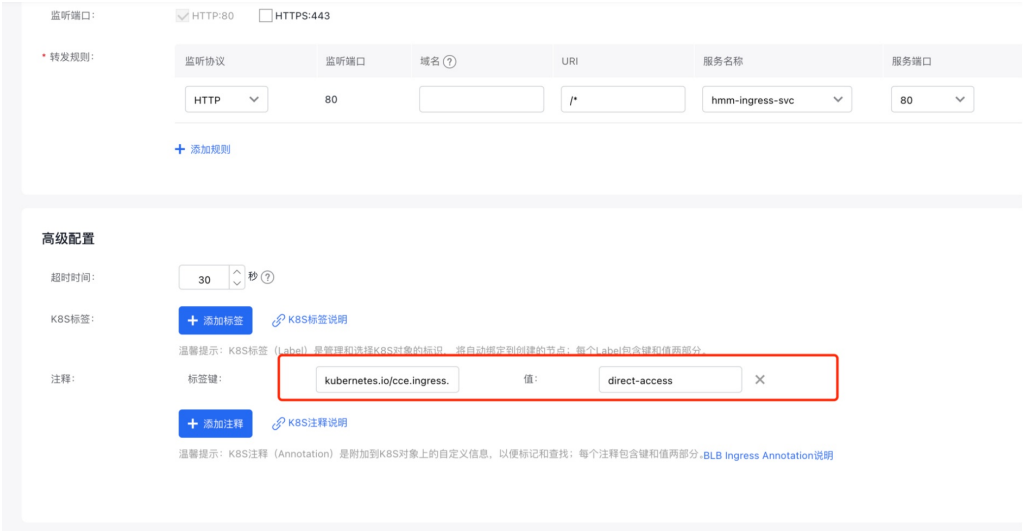
1.环境变量



2.添加 Annotation

`kubernetes.io/cce.ingress.network-mode: direct-access`

控制台添加 Annotation 的方式如下图所示



自定义监听器配置 取值说明

扩展协议配置用于补充 Spec 中规定的端口的协议配置，这些扩展配置将作用于应用型 BLB 的监听器上。

扩展协议内容在kubernetes.io/cce.ingress.listener.customized-listenerAnnotation 下。每种协议能配置的参数如下：

```
 ##### 对于 HTTP
protocol                HTTP
scheduler                非必填。负载均衡方法。支持"RoundRobin"、"LeastConnection"，默认 "RoundRobin"。
timeout                 非必填。HTTP服务端超时时间。默认30。
redirectPort             非必填。HTTPS重定向端口。
disableXForwardedForHeader  非必填。是否关闭 XForwardedFor 头。默认"false"。
enableXForwardedProtoHeader 非必填。是否关开启 XForwardedProto 头。默认"false"。
enableKeepSession          非必填。是否开启会话保持。默认"false"。
keepSessionType           非必填。会话保持的 Cookie 处理方式，当且仅当开启会话保持时有效，支持"insert"/"rewrite"，默认"insert"。
keepSessionTimeout         非必填。会话保持超时时间。
KeepSessionCookieName      非必填。会话保持类型为 rewrite 时 Cookie 的名称。

##### 对于 HTTPS
protocol                HTTPS
scheduler                非必填。负载均衡方法。支持"RoundRobin"、"LeastConnection"，默认"RoundRobin"。
timeout                 非必填。HTTPS服务端超时时间。默认30。
serverCertIDs             必填。服务端证书ID链，逗号分隔。
encryptionType            非必填。加密选项，默认tls_cipher_policy_default。支持：
tls_cipher_policy_default/tls_cipher_policy_1_1/tls_cipher_policy_1_2/tls_cipher_policy_1_2_secure/userDefind。
encryptionProtocols        非必填。加密协议，当encryptionType值为userDefind时协议类型列表，是以"tlsv10"、"tlsv11"、"tlsv12"三种协议
组合形成的字符串列表。
appliedCiphers            非必填。加密套件。不同加密套件用冒号":"隔开。
dualAuth                非必填。是否开启双向认证。默认"false"。
clientCertIDs             非必填。开启双向认证的客户端证书链。
disableXForwardedForHeader  非必填。是否关闭 XForwardedFor 头。默认"false"。
enableXForwardedProtoHeader 非必填。是否关开启 XForwardedProto 头。默认"false"。
enableKeepSession          非必填。是否开启会话保持。默认"false"。
keepSessionType           非必填。会话保持的 Cookie 处理方式，当且仅当开启会话保持时有效，支持"insert"/"rewrite"，默认"insert"。
keepSessionTimeout         非必填。会话保持超时时间。
KeepSessionCookieName      非必填。会话保持类型为"rewrite"时 Cookie 的名称。
```

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 只支持针对 80 端口自定义 HTTP 监听器配置，以及针对 443 端口自定义 HTTPS 监听器配置；

示例

```
 kubernetes.io/cce.ingress.listener.customized-listener: |-
    {
      "80":{
        "protocol":"HTTP",
        "redirectPort":"443",
        "timeout":"31",
        "scheduler":"LeastConnection",
        "enableKeepSession":"true",
        "keepSessionType":"rewrite",
        "disableXForwardedForHeader": "true",
        "enableXForwardedProtoHeader": "true",
        "extendedAttributes": {
            "proxy_set_header": ["X-Real-IP $cip"]
        }
      },
      "443":{
        "protocol":"HTTPS",
        "serverCertIDs":"cert-5bqpykggjzmi",
        "timeout":"31",
        "scheduler":"LeastConnection",
        "enableKeepSession":"true",
        "keepSessionType":"rewrite",
        "encryptionType": "userDefind",
        "encryptionProtocols": "tlsv10, tlsv11, tlsv12",
        "appliedCiphers": "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-SHA",
        "extendedAttributes": {
            "proxy_set_header": ["X-Real-IP $cip"]
        }
      }
    }
```

自定义健康检查规则 取值说明

扩展健康检查协议配置用于用户自定义 BLB Service 关联的 BLB 的服务器或 IP 组的健康检查配置。

在默认情况下，端口后端的服务器或 IP 组均使用 TCP 健康检查。

如果需要修改健康检查方式，或是健康检查相关的参数，可以使用此 Annotation。

扩展健康检查协议内容在kubernetes.io/cce.ingress.backend.customized-health-checkAnnotation 下，支持 TCP、HTTP 2 种健康检查协议。每种健康检查可配置的参数如下：

```
 ##### TCP 健康检查
healthCheck                必填，固定为TCP。
healthCheckTimeoutInSecond  非必填。健康检查超时时间。默认为3，需为1-60间的整数。
healthCheckIntervalInSecond 非必填。健康检查间隔时间。默认为3，需为1-60间的整数。
healthCheckDownRetry       非必填。不健康阈值，即连续多少次健康检查失败后，屏蔽该后端服务器。默认为3，需为2-5间的整数。
healthCheckUpRetry         非必填。健康阈值，即连续多少次健康检查成功后，重新将该后端服务器置为可用。默认为3，需为2-5间的整数。

##### HTTP 健康检查
healthCheck                必填，固定为HTTP。
healthCheckTimeoutInSecond  非必填。健康检查超时时间。默认为3，需为1-60间的整数。
healthCheckIntervalInSecond 非必填。健康检查间隔时间。默认为3，需为1-60间的整数。
healthCheckDownRetry       非必填。不健康阈值，即连续多少次健康检查失败后，屏蔽该后端服务器。默认为3，需为2-5间的整数。
healthCheckUpRetry         非必填。健康阈值，即连续多少次健康检查成功后，重新将该后端服务器置为可用。默认为3，需为2-5间的整数。
healthCheckUrlPath         非必填。健康检查URI，默认/。
healthCheckNormalStatus        非必填。健康检查正常时的HTTP状态码，支持5类状态码的组合，例如"http_1xx|http_2xx"，默认为"http_2xx|http_3xx"
healthCheckHost            非必填。7层健康检查请求的头部域会带指定的host字段，例如"localhost"，默认""。
```

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 只可以使用 TCP 或 HTTP 健康检查；

示例 以下是一个使用 HTTP 健康检查的 BLB Ingress 示例：

```
kubernetes.io/cce.ingress.backend.customized-health-check: |-
  {
        "healthCheck":"HTTP",
        "healthCheckUrlPath":"/",
        "healthCheckTimeoutInSecond":"2",
        "healthCheckIntervalInSecond":"2",
        "healthCheckDownRetry":"2",
        "healthCheckUpRetry": "2",
        "healthCheckNormalStatus": "http_2xx",
        "healthCheckHost": "baidu.com"
  }
```

配置会话保持 取值说明 会话保持功能涉及多条Annotation，解释如下：

- 打开或关闭会话保持

值为`true`为打开会话保持。默认值为`false`。

开关HTTP会话保持使用`kubernetes.io/cce.ingress.blb.listener.http.keep-session.enable`，开关HTTPS会话保持使用`kubernetes.io/cce.ingress.blb.listener.https.keep-session.enable`。

在打开会话保持后，后面的Annotation才会生效。

- 设置会话保持类型

值为`insert`表示植入Cookie，值为`rewrite`表示重写Cookie。默认值为`insert`。

设置HTTP会话保持类型使用`kubernetes.io/cce.ingress.blb.listener.http.keep-session.type`，设置HTTPS会话保持类型使用`kubernetes.io/cce.ingress.blb.listener.https.keep-session.type`。

- 设置会话超时时间（仅对insert类型有效）

取值为 [1,15552000]，默认值为3600，单位是秒。

设置HTTP会话保持超时时间用`kubernetes.io/cce.ingress.blb.listener.http.keep-session.timeout`，设置HTTPS会话保持超时时间使用`kubernetes.io/cce.ingress.blb.listener.https.keep-session.timeout`。

- 设置Cookie名称（仅对rewrite类型有效）

设置HTTP会话保持Cookie名称使用`kubernetes.io/cce.ingress.blb.listener.http.keep-session.cookie-name`，设置HTTP会话保持Cookie名称使用`kubernetes.io/cce.ingress.blb.listener.https.keep-session.cookie-name`。

注意事项

- CCE不会自动添加该Annotation，需要自行按需添加；
- 仅在新建应用型BLB监听器时有效。

示例

```
 ##### HTTP
kubernetes.io/cce.ingress.blb.listener.http.keep-session.enable: "true",
kubernetes.io/cce.ingress.blb.listener.http.keep-session.type: "insert"
kubernetes.io/cce.ingress.blb.listener.http.keep-session.timeout: 999

##### HTTPS
kubernetes.io/cce.ingress.blb.listener.https.keep-session.enable: "true",
kubernetes.io/cce.ingress.blb.listener.https.keep-session.type: "rewrite",
kubernetes.io/cce.ingress.blb.listener.https.keep-session.cookie-name: "cce-test"
```

网络管理

创建VPC-ENI模式集群

VPC-ENI 模式说明

VPC-ENI 模式是容器引擎 CCE 支持的扩展网络模式，基于百度云的弹性网卡产品，能够为集群内的 Pod 分配 VPC 内的 IP 地址。 由私有网络 VPC 产品负责路由，打通容器网络的连通性，可实现 Pod 和 Node 的控制面和数据面完全在同一网络层面，该模式下的 Pod 能够复用百度云 VPC 所有产品特性。

VPC-ENI 模式的集群一经创建，无法修改，请提前做好网络规划。

由于 VPC-ENI 模式存在使用限制，建议您提前考虑是否适配您的业务场景。

VPC-ENI 模式应用场景

采用 VPC-ENI 模式的集群，具有如下优势：

1. 原生的私有网络 VPC 网络性能；
2. 整合私有网络 VPC 的产品特性，例如您能够给 Pod 绑定单独安全组实现数据包过滤；

VPC-ENI 模式使用说明

1. VPC-ENI 模式支持 Node 分布在在不同可用区、不同子网；
2. VPC-ENI 模式支持弹性网卡分布在不同可用区、不同子网，但弹性网卡只能绑定在同可用区的 Node 上；
3. VPC-ENI 模式要求弹性网卡与 Node 分布在不同子网；
4. VPC-ENI 模式为每个 Node 动态绑定多个弹性网卡；

VPC-ENI 容器规模限制

VPC-ENI 模式下，每个 Node 上可分配给容器的 IP 数量上限由 Node 规格（CPU 核数与内存）决定。 Node 可挂载弹性网卡数量=min（主机核数，8），内存与单个弹性网卡网卡辅助 IP 数量对应关系见下表。

Node 内存	单个弹性网卡辅助 IP 数量上限
1G	1
(1-8]G	7
(8-32]G	15
(32-64]G	29
大于64G	39

单个 Node 上可分配给容器的 IP 数量 = 可挂载弹性网卡数量 * 单个弹性网卡辅助 IP 数量上限

如果使用 BBC 机器以及不支持弹性网卡的 BBC 机器，在主网卡辅助 IP 模式下，单网卡上IP数量最少1个，最多39个。

举例说明:

- Node1 的规格是 4C8G，那么 Node1 上最多可分配给容器的 IP 数量是 28；
- Node2 的规格是 12C32G，那么 Node2 上最多可分配给容器的 IP 数量是 120；

创建 VPC-ENI 模式集群

1. 在容器引擎 CCE 控制台单击**创建集群**进入集群创建页面；
2. 在**网络配置**选项中将**容器网络模式**配置选择为 **VPC-ENI** 如下图所示；
3. 配置完相应选项后，点击**下一步**继续完成集群其他配置。

配置项	必选/可选	配置说明
弹性网卡模式	必选	支持独占和共享两种模式。独占：多个 Pod 共享一个弹性网卡，系统为弹性网卡申请多个辅助 IP 分配给不同的 Pod。 • 开启eBPF增强：支持开启 eBPF 增强，开启后原生Kube-proxy 模式将被替换。 • 开启Network Policy 支持：开启后支持为集群提供基于策略的网络控制，更多信息请见使用 Network Policy 网络策略。 共享：每个 Pod 将独占一个弹性网卡，系统为弹性网卡申请一个辅助 IP 地址分配给 Pod。独占弹性网卡模式目前为白名单开放，若有需求请提工单。
容器子网	必选	指定私有网络 VPC 中的子网作为容器子网，集群创建的 Pod 将从容器子网中分配 IP 地址，请选择与节点同可用区的子网作为容器子网。
ClusterIP 网段	必选	指定集群中为 Service 分配 IP 地址的专用网络地址范围。此网段不能与 VPC 及 VPC 内已有集群使用的网段重复，且服务地址段也不能和容器地址段重复。
LB Service 子网	必选	指定创建负载均衡类型 Service 及 Ingress 时默认使用的子网，不能选择 NAT 子网。
弹性网卡安全组	必选	指定为弹性网卡绑定的安全组，支持自动创建和选择已有普通安全组及企业安全组，更多信息请见CCE安全组说明。
Kube-proxy模式	必选	支持IPVS和IPTABLES两种模式。 • IPVS：高性能的 Kube-proxy 代理模式，Kubernetes Service 的服务发现和负载均衡使用 Linux IPVS 模块进行配置，适用于集群存在大量 Service，对负载均衡有高性能要求的场景。 • IPTABLES：成熟稳定的 Kube-proxy 代理模式，Kubernetes Service 的服务发现和负载均衡使用 IPTABLES 模块进行配置，但性能一般，受规模影响较大，适用于集群存在少量 Service。
Nodeport 范围	必选	指定集群中Service的端口，默认为30000～32767。

注意事项：

- 当前配额每个 VPC 默认仅支持创建 500 个弹性网卡，如有提升配额需求请提交工单至 CCE；
- 客户在删除 k8s namespace 资源之前，需要先删除完该 namespace 下所有 Pod，否则可能造成辅助 IP 资源泄露；

容器网段空间耗尽如何继续扩容(VPC网络模式)

注意：以下内容仅针对使用 “VPC Route”模式的集群

概述

一个集群中最大的节点数量由容器网段的大小和每个节点上最大 Pod 数量所决定，例如：

- 容器网段选择 172.16.0.0/16，每个节点最大 Pod 数量为 256，则一个集群中最多只能有 256 个节点；
- 容器网段选择 192.168.0.0/22，每个节点最大 Pod 数量为 128，则一个集群中最多只能有 8 个节点；

在某些情况下，由于用户创建的集群时，选择的容器网段过小或者每个节点上最大的容器数量过大，在后续对集群进行扩容时，节点数量超过了集群中最大的节点数量。由于 kube-controller-manager 无法为这些扩容的节点分配Pod 的网段，会导致节点状态 notReady。

> 注意：在 2021-3-11 之前，controller manager 采用二进制方式部署，按照老版本解决办法。2021-3-11 之后的新建的集群，controller manager 采用静态 Pod 方式部署，按照新版本解决办法。

master 二进制部署集群解决方法

在 2021-3-11 之前，controller manager 采用二进制方式部署。

此时只有容器网络 V1 架构，不排除个人手动升级到 V2 架构的情况。请自行确认当前使用的容器网络版本。对应的方法为检查集群中是否有 cce-cni-node-agent 对象，如果有该对象，则使用的容器网络版本是 v1。

```
 $ kubectl -n kube-system get cm cce-cni-node-agent
NAME                 DATA   AGE
cce-cni-node-agent   1      125d
```

第一步

修改集群中 master 节点上 cce-network-operator 的配置，需要修改的字段是 cluster-pool-ipv4-mask-size。由于修改目的是让集群中可以容纳更多节点，也就是让节点上最大的 Pod 数量变小，我们需要修改 cluster-pool-ipv4-mask-size 值使其变大。

如果是多副本的 master，需要逐一修改各 master 节点上的配置。

> 注意: 不能修改 cluster-pool-ipv4-mask-size 比当前值更小，否则会造成网段冲突导致网络不通

第二步

从集群中移出节点再加入，包括的方法有两种：

- 从 CCE 产品 console 界面上选择“移出节点”或“删除节点”，之后再“移入节点”或“添加节点”。
- 执行 `kubectl delete node <nodeName>` 从 k8s 集群中删除节点。执行 `kubectl get pods --all-namespaces=true -o wide | grep <nodeName>` 确保该节点上没有 Pod 之后，在节点上重启 kubelet 将节点重新加入到 k8s 集群。

> 注意：不论哪种方式将节点从集群中删除，都会导致移出的节点上的所有 Pod 漂移。对于承载了线上服务的节点，需要谨慎操作。

实际案例

问题场景

目前有一个集群的容器网段是 172.26.0.0/22，kube-controller-manager 配置中 --node-cidr-mask-size=24。也就是说，集群中最多容纳 4 个节点，每个节点上最大 Pod 数量是 256。

集群中已经有 4 个节点了，如果再扩容节点将会导致新扩容的节点不可用。

```
 [root@instance-rhkiutp6-3 ~]# kubectl get node
NAME       STATUS   ROLES    AGE    VERSION
10.0.5.3   Ready    <none>   119m   v1.13.10
10.0.5.4   Ready    <none>   117m   v1.13.10
10.0.5.5   Ready    <none>   20m    v1.13.10
10.0.5.6   Ready    <none>   118m   v1.13.10
[root@instance-rhkiutp6-3 ~]# kubectl describe node | grep -i podcidr
PodCIDR:                     172.26.2.0/24
PodCIDR:                     172.26.1.0/24
PodCIDR:                     172.26.3.0/24
PodCIDR:                     172.26.0.0/24
```

修改步骤

第一步 在 master 上执行 `vim /etc/systemd/system/kube-controller.service` 查看 kube-controller-manager 配置：

```
 [Unit]
Description=Kubernetes Controller Manager
After=network.target
After=kube-apiserver.service

[Service]
ExecStart=/opt/kube/bin/kube-controller-manager \
--allocate-node-cidrs=true \
--cloud-config=/etc/kubernetes/cloud.config \
--cluster-cidr=172.26.0.0/22 \
--node-cidr-mask-size=24 \   #修改这里
.......
--kubeconfig=/etc/kubernetes/controller-manager.conf \
--leader-elect=true \
--logtostderr=true \
--master=https://100.64.230.195:6443 \
--v=6
Restart=always
Type=simple
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

修改 --node-cidr-mask-size 值从 24 变更为 26。这样修改后，集群中最多可以容纳 16 个节点，每个节点上最大 Pod 数量则减少为 64。

依次在每个 master 节点修改配置后执行以下命令重启 kube-controller-manager。

```
 systemctl daemon-reload
systemctl restart kube-controller.service
```

第二步

执行 `kubectl delete node 10.0.5.4`，此时集群状态中已经没有 10.0.5.4 节点:

```
 [root@instance-rhkiutp6-3 ~]# kubectl get node
NAME       STATUS   ROLES    AGE    VERSION
10.0.5.3   Ready    <none>   132m   v1.13.10
10.0.5.5   Ready    <none>   33m    v1.13.10
10.0.5.6   Ready    <none>   132m   v1.13.10
[root@instance-rhkiutp6-3 ~]# kubectl describe node | grep -i podcidr
PodCIDR:                     172.26.2.0/24
PodCIDR:                     172.26.3.0/24
PodCIDR:                     172.26.0.0/24
```

执行 `kubectl get pods --all-namespaces=true -o wide | grep <nodeName>` 确保 10.0.5.4 上没有 Pod。

在 10.0.5.4 节点上执行 `systemctl restart kubelet.service` 重启 kubelet，此时集群状态中表明 10.0.5.4 节点又被加入并且容器网段变为了 172.26.1.0/26:

```
 [root@instance-rhkiutp6-3 ~]# kubectl get node
NAME       STATUS   ROLES    AGE     VERSION
10.0.5.3   Ready    <none>   138m    v1.13.10
10.0.5.4   Ready    <none>   3m55s   v1.13.10
10.0.5.5   Ready    <none>   40m     v1.13.10
10.0.5.6   Ready    <none>   138m    v1.13.10
[root@instance-rhkiutp6-3 ~]# kubectl describe node | grep -i podcidr
PodCIDR:                     172.26.2.0/24
PodCIDR:                     172.26.1.0/26
PodCIDR:                     172.26.3.0/24
PodCIDR:                     172.26.0.0/24
```

每移入移出一个已有节点将会创造出 3 个节点的扩容空间。例如现在可以给集群继续扩容 3 台节点，分配的 PodCIDR 将会是`172.26.1.64/26`、`172.26.1.128/26` 以及 `172.26.1.192/26`。

用户继续执行以上步骤移入移出所有节点，以创造更多的扩容空间。

> 注意：PodCIDR 掩码不同的节点理论上可以存在于同一集群中，但还是建议用户将所有节点移入移出一遍，让节点拥有相同掩码的 PodCIDR。

master 采用静态Pod部署方式解决办法

在 2021-3-11 之后，controller manager 采用静态 Pod 方式部署，修改对应的配置文件即可完成 Pod 更新。

容器网络分为 V1 和 V2 架构。请自行确认当前使用的容器网络版本。对应的方法为检查集群中是否有 cce-cni-node-agent 对象，如果有该对象，则使用的容器网络版本是 v1。

```
 $ kubectl -n kube-system get cm cce-cni-node-agent
NAME                 DATA   AGE
cce-cni-node-agent   1      125d
```

第一步 修改配置 修改集群中 master 节点上 `kube-controller-manager` 的配置，需要修改的字段是 --node-cidr-mask-size。

由于修改目的是让集群中可以容纳更多节点，也就是让节点上最大的 Pod 数量变小，我们需要修改 --node-cidr-mask-size 值使其变大。

如果是多副本的 master，需要逐一修改各 master 节点上的配置。

| 注意: 不能修改 node-cidr-mask-size 比当前值更小，否则会造成网段冲突导致网络不通

第二步 修改网络组件 容器网络分 V1 和 V2 架构分别对应不同的修改方法。请自行确认当前使用的容器网络版本。对应的方法为检查集群中是否有 cce-cni-node-agent 对象，如果有该对象，则使用的容器网络版本是 v1。

```
 $ kubectl -n kube-system get cm cce-cni-node-agent
NAME                 DATA   AGE
cce-cni-node-agent   1      125d
```

对于容器网络 V1 架构，v1 直接用node spec里的podcidr，并不需要进行修改。

对于容器网络 V2 架构，需要修改 cce-network-v2-config，对应的字段是 cluster-pool-ipv4-mask-size。之后重启对应的组件 `cce-network-operator` 和 `cce-network-agent`，以确保对应的配置生效。

第三步 重新移出移入节点 从集群中移出节点再加入，包括的方法有两种：

- 从 CCE 产品 console 界面上选择“移出节点”或“删除节点”, 之后再“移入节点”或“添加节点”。
- 执行 `kubectl delete node <nodeName>` 从 k8s 集群中删除节点。执行 `kubectl get pods --all-namespaces=true -o wide | grep <nodeName>` 确保该节点上没有 Pod 之后，在节点上重启 `kubelet` 将节点重新加入到 k8s 集群。 注意：不论哪种方式将节点从集群中删除，都会导致移出的节点上的所有 Pod 漂移。对于承载了线上服务的节点，需要谨慎操作。

实际案例

问题场景

目前有一个集群的容器网段是 172.16.0.0/24 ，`kube-controller-manager` 配置中 `node-cidr-mask-size=24` 。也就是说，集群中最多容纳 256 个节点，每个节点上最大 Pod 数量是 256。现在需要将可容纳节点个数提升到 1024 个。

```
 [root@root ~]# kubectl get no
NAME          STATUS   ROLES    AGE   VERSION
192.168.1.4   Ready    <none>   42m   v1.24.4
192.168.1.5   Ready    <none>   35m   v1.24.4
root@root:~# kubectl describe node | grep -i podcidr
root@root:~# kubectl describe node | grep -i podcidr
PodCIDR:                      10.0.1.0/24
PodCIDRs:                     10.0.1.0/24
PodCIDR:                      10.0.0.0/24
PodCIDRs:                     10.0.0.0/24
```

修改步骤 第一步 修改配置 查看 cnode-cidr-mask-size 配置：

```
[root@root manifests]# kubectl get po kube-controller-manager-192.168.1.4 -n kube-system -o yaml | grep node-cidr-mask-size
  - --node-cidr-mask-size=24
```

修改 cluster-pool-ipv4-mask-size 值从 24 变更为 26。这样修改后，集群中最多可以容纳 1024 个节点，每个节点上最大 Pod 数量则减少为 64。

修改 `kube-controller-manager` 参数 `node-cidr-mask-size=26` ：

```
 vim /etc/kubernetes/manifests/kube-controller-manager.yaml

apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
    - command:
        - kube-controller-manager
        - --cluster-cidr=172.16.0.0/16
        - --feature-gates=MixedProtocolLBService=true
        - --master=https://192.168.1.4:6443
        - --node-cidr-mask-size=26
        ……
```

这里 `kube-controller-manager` 是通过静态方式部署的。对于静态Pod，kubelet 会监视定义文件的变化。当保存并关闭编辑器后，kubelet 将检测到文件已更改，并自动根据更新后的定义文件删除旧的 Pod 并启动新的 Pod 。

第二步 修改网络组件 这里使用的 V2 网络架构，需要修改 `cce-network-v2-config` 的 `cluster-pool-ipv4-mask-size` 字段。

```
kubectl edit cm cce-network-v2-config -n kube-system

apiVersion: v1
data:
  cced: |
    annotate-k8s-node: true
    api-rate-limit:
      bcecloud/apis/v1/AttachENI: rate-limit:5/1s,rate-burst:5,max-wait-duration:30s,parallel-requests:5,log:true
      bcecloud/apis/v1/BatchAddPrivateIP: rate-limit:5/1s,rate-burst:10,max-wait-duration:15s,parallel-requests:5,log:true
      bcecloud/apis/v1/BatchDeletePrivateIP: rate-limit:5/1s,rate-burst:10,max-wait-duration:15s,parallel-requests:5,log:true
      bcecloud/apis/v1/CreateENI: rate-limit:5/1s,rate-burst:5,max-wait-duration:30s,parallel-requests:5,log:true
      bcecloud/apis/v1/DescribeSubnet: rate-limit:5/1s,rate-burst:5,max-wait-duration:30s,parallel-requests:5
      bcecloud/apis/v1/StatENI: rate-limit:10/1s,rate-burst:15,max-wait-duration:30s,parallel-requests:10
    auto-create-network-resource-set-resource: true
    bbcEndpoint: bbc.gz.baidubce.com
    bccEndpoint: bcc.gz.baidubce.com
    bce-cloud-access-key: ""
    bce-cloud-country: cn
    bce-cloud-host: cce-gateway.gz.baidubce.com
    bce-cloud-region: gz
    bce-cloud-secure-key: ""
    bce-cloud-vpc-id: vpc-2f5wibbx4js7
    cce-cluster-id: cce-clboj6fa
    cce-endpoint-gc-interval: 30s
    cluster-pool-ipv4-cidr:
    - 172.16.0.0/16
    cluster-pool-ipv4-mask-size: 26
```

重启对应的组件`cce-network-operator` 和 `cce-network-agent` 以确保对应的配置生效。

```
kubectl rollout restart deployment cce-network-operator -n kube-system
kubectl rollout restart daemonset cce-network-agent -n kube-system
```

第三步 重新移出移入节点 执行 `kubectl delete` 指令，删除对应的节点。

```
kubectl delete node 192.168.1.4
```

查看集群状态，确保此时集群状态中已经没有 192.168.1.4 节点:

```
[root@root ~]# kubectl get no
NAME          STATUS   ROLES    AGE   VERSION
192.168.1.5   Ready    <none>   47m   v1.24.4
root@root:~# kubectl describe node | grep -i podcidr
PodCIDR:                      10.0.1.0/24
PodCIDRs:                     10.0.1.0/24
```

确保 192.168.1.4 上没有 Pod。如果有 Pod 残留会导致重新加入 Node 后 Pod CIDR 不变。执行以下命令进行查看。

```
kubectl get pods --all-namespaces=true -o wide | grep <nodeName>
```

在 192.168.1.2节点上执行 `systemctl restart kubelet.service` 重启 kubelet，此时集群状态中表明 192.168.1.2 节点又被加入并且容器网段变为了 172.26.1.0/26:

```
[root@root ~]# kubectl get node
NAME          STATUS   ROLES    AGE    VERSION
192.168.1.4   Ready    <none>   11m    v1.24.4
192.168.1.5   Ready    <none>   102m   v1.24.4
[root@root-3 ~]# kubectl describe node | grep -i podcidr
PodCIDR:                      10.0.0.0/26
PodCIDRs:                     10.0.0.0/26
PodCIDR:                      10.0.1.0/24
PodCIDRs:                     10.0.1.0/24
```

用户继续执行以上步骤移入移出所有节点，可以创造更多的扩容空间。

> 注意：PodCIDR 掩码不同的节点理论上可以存在于同一集群中，但还是建议用户将所有节点移入移出一遍，让节点拥有相同掩码的 PodCIDR。

CCE 支持 IPv4 和 IPv6 双栈网络

CCE 支持 IPv4 和 IPv6 双栈网络

互联网协议第六版（IPv6）是互联网升级演进的必然趋势、网络技术创新的重要方向、网络强国建设的基础支撑。CCE 目前支持 IPv4/IPv6 双栈集群，包括如下特点：

1. Node 同时支持 IPv4 和 IPv6 地址，集群内部支持使用两种类型地址通信；
2. Pod 双栈同时支持 IPv4 和 IPv6 地址，并支持两种类型 IP 访问；
3. Service 同时支持 IPv4 和 IPv6 地址，并支持通过 IPv6 对外暴露服务。

> IPv6 双栈网络参考：VPC 使用 IPv6

使用说明

仅支持在CCE标准集群中使用IPv4/IPv6双栈容器网络。 **前置条件**

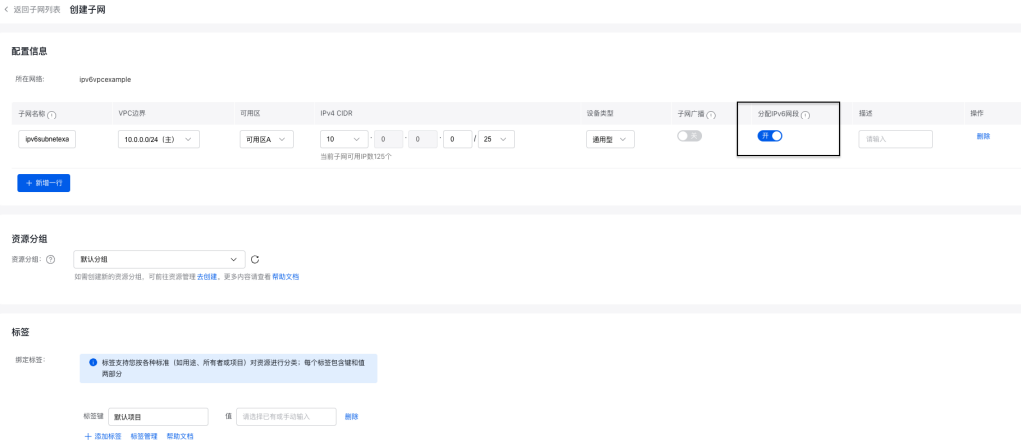
1. kubernetes 版本1.20以上
2. cce-network网络插件版本2.7.7以上

新建 IPv4/IPv6 双栈 VPC 及子网

CCE IPv4/IPv6 双栈集群需要运行在支持 IPv6 的 VPC（私有网络） 的子网中，因此在创建双栈集群前应首先创建一个支持 IPv6 的 VPC 及其子网。

创建 VPC 在 VPC 控制台内新建 VPC，并开启『分配IPv6网段』。

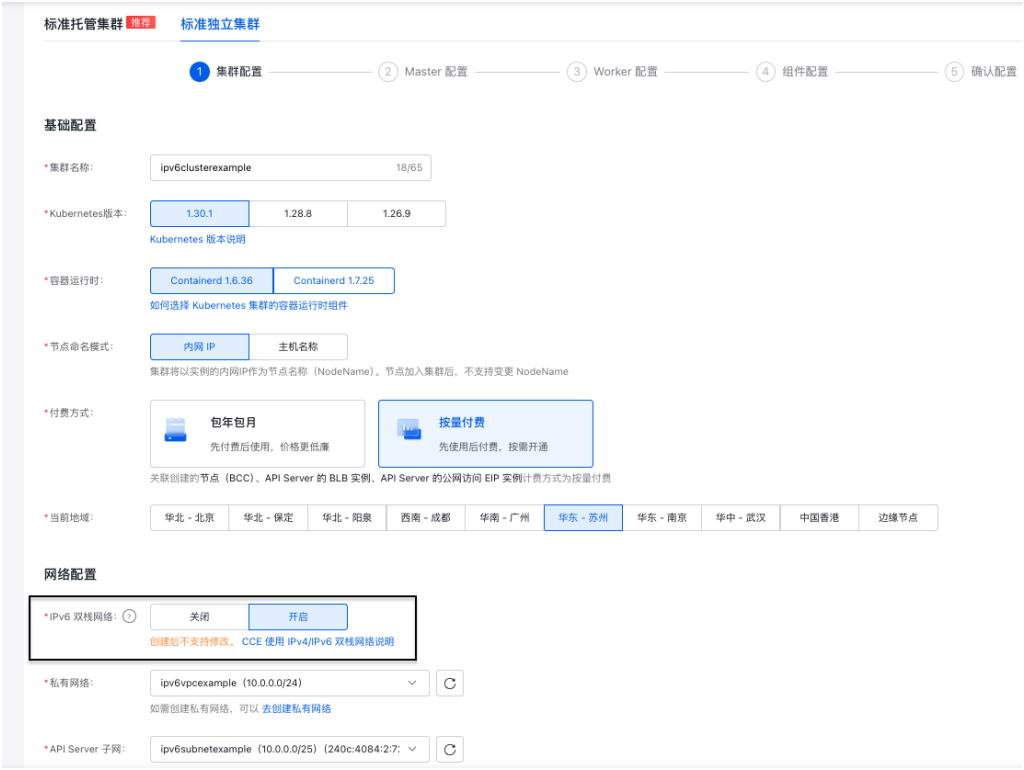
创建子网 在 VPC 中创建子网，并开启『分配IPv6网段』。



新建 IPv4/IPv6 双栈 K8S 集群

进入 CCE 控制台，创建双栈集群，如下所示：

1. 在『基础配置』页面
   选择开启『IPv6双栈网络』，并选择一个支持 IPv6 的 VPC 和其子网。



2. 在『网络配置』页面 系统会自动推荐合适的 IPv4/IPv6 ClusterIP 网段并显示在下方。 IPv6网段支持用户手动指定符合RFC 4193定义的唯一本地地址（Unique Local Addresses，ULA），地址范围是`fd00::/8` 的子集。

3. 后续按提示进行，完成集群的创建。

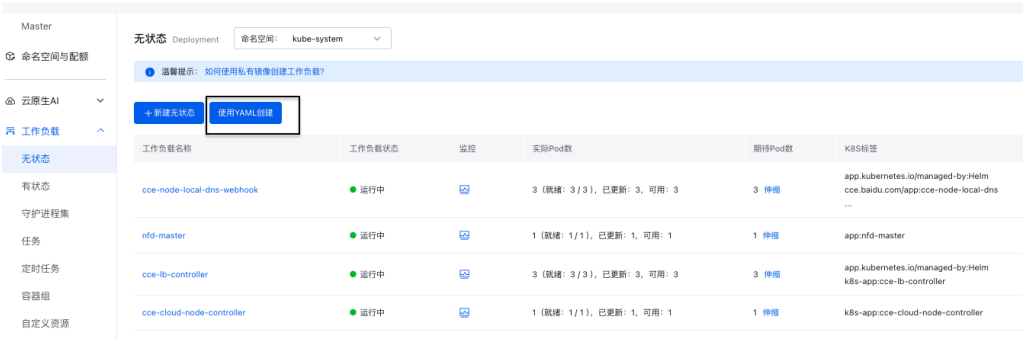
通过 IPv4 和 IPv6 访问 Pod

在 IPv4/IPv6 双栈集群中，可以为 Pod 同时分配 IPv4 地址和 IPv6 地址。

> 注意：后端 Pod 也需要监听 IPv6 协议栈下端口，否则会出现 Pod 被分配了 IPv6 地址但是连接被拒绝的情况

通过 CCE 控制台的『工作负载』页面，创建示例 Deployment，可以使用以下示例 Deployment，其中所用的镜像同时在监听 IPv4 和 IPv6 协议栈的 80 端口：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: deployment-example-ipv6
 labels:
   app: nginx
spec:
 replicas: 2
 minReadySeconds: 0
 strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
 selector:
  matchLabels:
    app: nginx
 template:
  metadata:
    labels:
      app: nginx
  spec:
    restartPolicy: Always
    containers:
      - name: nginx
        image: registry.baidubce.com/cce/nginx-alpine-go:ipv6
        imagePullPolicy: Always
        ports:
          - containerPort: 80
```



查看 Pod IPv4/IPv6 地址：

```
 ##### kubectl get pods deployment-example-64445974b7-8s6zc  -o go-template --template='{{range .status.podIPs}}{{printf "%s \n" .ip}}
{{end}}'
172.17.0.12
fc00::c
```

访问 Pod 的 IPv4 地址：

```
 ##### curl 172.17.0.12
```

访问 Pod 的 IPv6 地址：

```
 ##### curl -g [fc00::c]:80
```

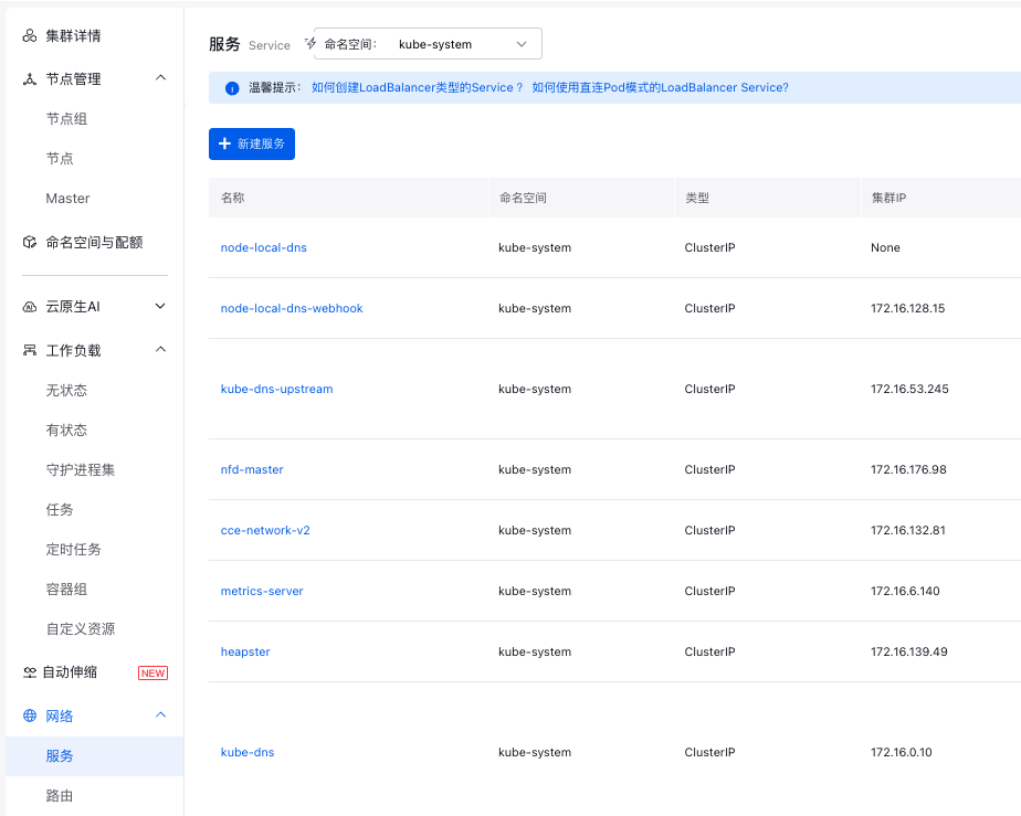
> **注意** 镜像 registry.baidubce.com/cce/nginx-alpine-go:latest 支持 IPv4/IPv6 双栈。

通过 IPv4 或 IPv6 访问 Service

在 IPv4/IPv6 双栈集群中，可以为 Service 分配 IPv4 地址或 IPv6 地址。 通过 Service 的`.spec.ipFamilies` 属性指定 Service 的 IP 地址类型，其值为 `IPv4` 或 `IPv6`。

> **注意**

1. 在未指定 .spec.ipFamilies 配置的情况下， Kubernetes 将使用 kube-controller-manager 中 --service-cluster-ip-range 参数的第一段 IP 地址为 Service 分配 IP 地址。因此为了避免混淆，我们建议在双栈集群中部署 Service 时始终配置 .spec.ipFamilies 属性。
2. .spec.ipFamilies 是一个可变更的字段：可以通过修改这个字段的值，来控制添加或删除service的辅助IP。 但不可以修改service的主IP。
3. 在使用 IPv6 版的 LoadBalancer Service 时，后端 Pod 无法从网络包中获取请求的原始 IP 地址。



创建 IPv4 单栈的 Service

指定 spec.ipFamily: IPv4 为 Service 分配 IPv4 地址：

```
 apiVersion: v1
kind: Service
metadata:
  name: service-example-ipv4
  annotations:
    prometheus.io/scrape: "true"
spec:
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  selector:
    app: nginx
  type: LoadBalancer
  externalTrafficPolicy: Cluster
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

查看 IPv4 Service 详情：

```
 ##### kubectl describe svc service-example-ipv4
Name:                     service-example-ipv4
Namespace:                default
Labels:                   <none>
Annotations:              prometheus.io/scrape: true
                          service.beta.kubernetes.io/cce-load-balancer-id: lb-bedc26c1
Selector:                 app=nginx
Type:                     LoadBalancer
IP:                       172.16.63.112
LoadBalancer Ingress:     106.13.103.101
Port:                     nginx  80/TCP
TargetPort:               80/TCP
NodePort:                 nginx  30009/TCP
Endpoints:                172.17.0.12:80,172.17.0.13:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason                Age    From                Message
  ----    ------                ----   ----                -------
  Normal  EnsuringLoadBalancer  6m56s  service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   6m37s  service-controller  Ensured load balancer
```

访问 IPv4 ClusterIP:

```
 ##### curl 172.16.63.112

<!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>
```

创建 IPv4/IPv6双栈 的 Service

指定 spec.ipFamily: IPv6 为 Service 分配 IPv6 地址：

```
  apiVersion: v1
kind: Service
metadata:
  name: service-example-ipv6
  annotations:
    prometheus.io/scrape: "true"
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
  - IPv6
  - IPv4
  selector:
    app: nginx
  type: LoadBalancer
  externalTrafficPolicy: Cluster
  sessionAffinity: None
  ports:
  - name: nginx
    protocol: TCP
    port: 80
    targetPort: 80
```

查看 IPv6 Service 详情：

```
  ##### kubectl describe svc service-example-ipv6
Name:                     service-example-ipv6
Namespace:                default
Labels:                   <none>
Annotations:              prometheus.io/scrape: true
                        service.beta.kubernetes.io/cce-load-balancer-id: lb-da9b6673
Selector:                 app=nginx
Type:                     LoadBalancer
IP:                       fdff:ffff:ffff:ffff:ffff:ffff:ffff:faa0
LoadBalancer Ingress:     240c:4082:1:4202::8
Port:                     nginx  80/TCP
TargetPort:               80/TCP
NodePort:                 nginx  30685/TCP
Endpoints:                [fc00::c]:80,[fc00::d]:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason                Age    From                Message
  ----    ------                ----   ----                -------
  Normal  EnsuringLoadBalancer  4m8s   service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   3m56s  service-controller  Ensured load balancer
```

访问 IPv6 ClusterIP:

```
##### curl -g [fdff:ffff:ffff:ffff:ffff:ffff:ffff:faa0]:80

<!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>
```

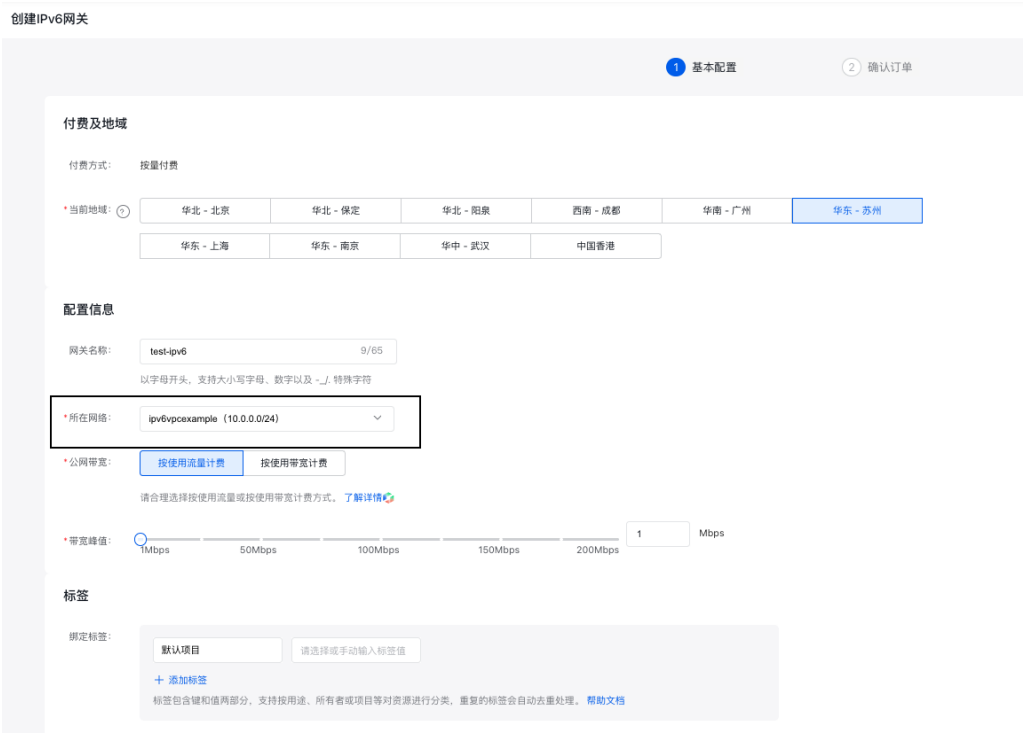
公网通过 IPv6 地址访问 Service

IPv6 网关为 VPC 内的 IPv6 地址提供公网连通性。若从公网通过 IPv6 地址访问 Service 或是使用 IPv6 地址的容器访问公网，需要 VPC 配置 IPv6 网关及相关路由。

创建 IPv6 网关

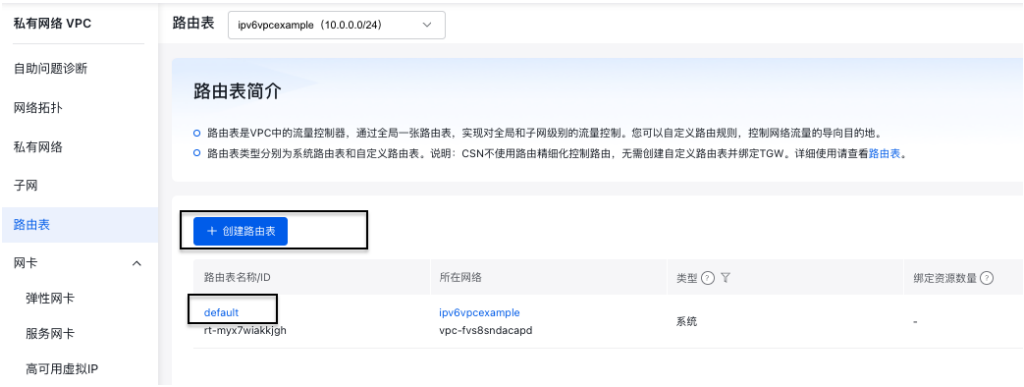
在 VPC 控制台的 『IPv6网关』标签页，点击『创建IPv6网关』

填写网关信息，其中『所在网络』选择集群所在的 VPC。



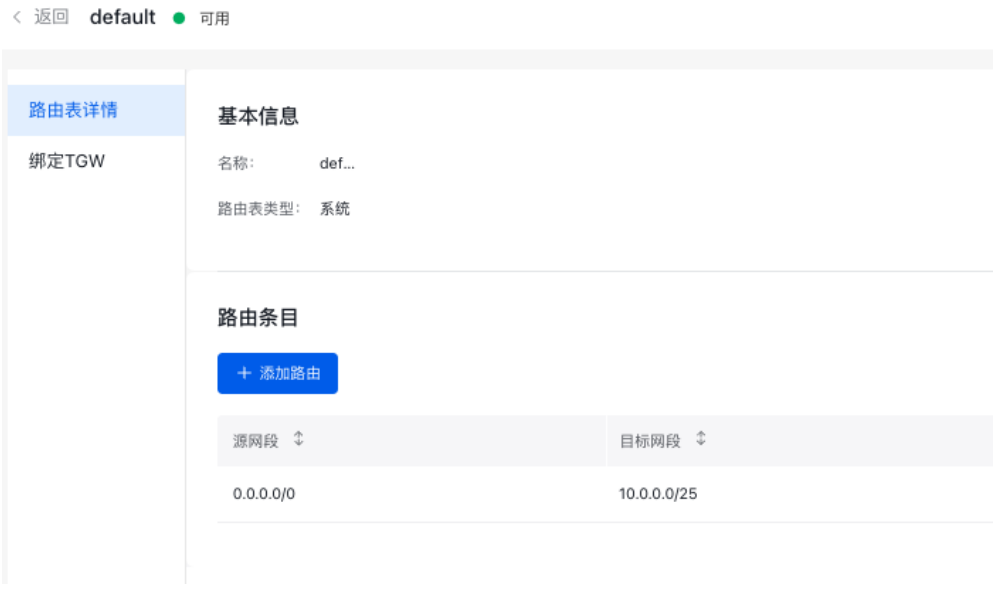
添加 IPv6 网关路由

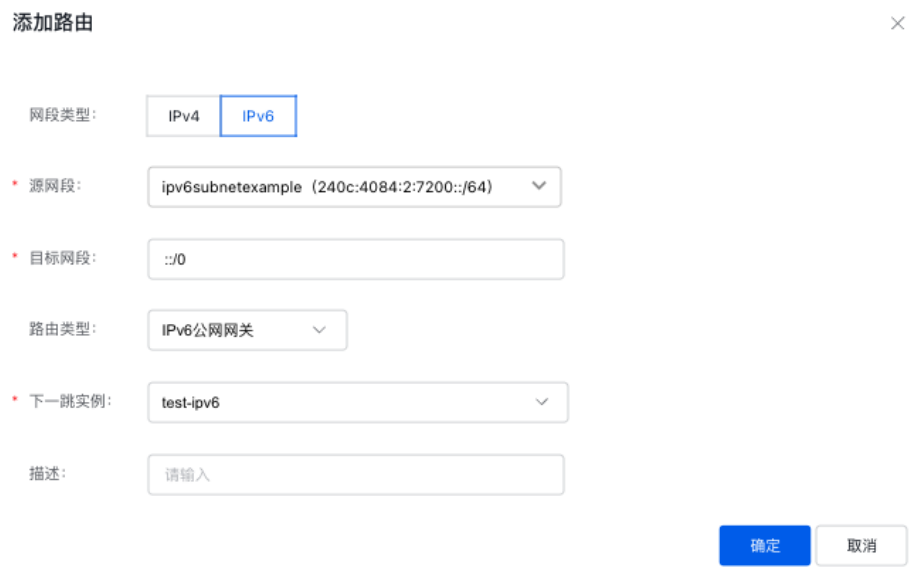
在 VPC 控制台的 VPC 实例页面中，点击『路由表』标签页，可以选择创建新路由表或者使用default 路由表。



选择需要配置的路由表进入，点击『添加路由』，填写路由表条目配置信息，其中『源网段』是 VPC 中的子网网段，目标网段为『::/0』，路由类型为『IPv6公网网关』，『下一跳实例』为上一步创建的网关。

> **注意** 若您的集群横跨一个 VPC 中的多个子网，您需要为每个子网网段创建路由条目。





访问上述 Service IPv6 EIP:

```
##### curl -g [240c:4082:1:4202::8]:80
```

参考文档

- IPv4/IPv6 双栈
- 验证 K8S IPv4/IPv6 双栈网络

对等连接场景下容器流量转发配置

本文档会描述在对等连接场景下，跨 VPC 的集群如何配置 VPC 路由，以便不同集群可以在节点、容器层面实现互相连通。

前置条件

- **对等连接已经创建完成**

  对等连接提供了VPC级别的网络互联服务，帮助用户实现在不同虚拟网络之间的流量互通。在本文档中，默认用户已经创建好对等连接以及 VPC 中的端点。具体创建流程请参考 对等连接

- **集群的网段互不冲突**

  k8s 集群中的三个网段分别是节点网段、容器网段以及 ClusterIP 网段。为了实现跨 VPC 的两个集群在节点、容器层面互相连通，需要保证不同 k8s 集群的中任意两个网段互不冲突。

- **容器流量出节点不进行 SNAT**

  由于 VPC 的网段均为私有 IP 网段，即 10.0.0.0/8、172.16.0.0/12、192.168.0.0/16。在 CCE 的默认配置下，容器流量出节点时不会对私有网段进行 SNAT。

配置案例

环境概述

以两个集群为例进行配置的说明，两个集群的名字分别为 A 和 B，集群具体的参数配置分别如下表所示。

集群 A 与 B 中的任意网段均不冲突，因此满足跨 VPC 集群互通的条件，可以进行下一步的配置。

集群	A	B
地域	gz	sz
节点网段	192.168.0.0/16	10.0.0.0/8
容器网段	172.16.0.0/16	172.17.0.0/16
ClusterIP 网段	172.30.0.0/16	172.31.0.0/16

配置 VPC 路由

在用户已经创建好对等连接以及 VPC 中的端点后，可以进行 VPC 路由的配置。用户需要分别在两个集群的 VPC 中配置路由。在本例中，用户需要在每个 VPC 中分别配置 4 条路由，分别是:

- 本集群节点通往对端集群节点的路由
- 本集群节点通往对端集群容器的路由

- 本集群容器通往对端集群节点的路由
- 本集群容器通往对端集群容器的路由

配置的结果如下：

集群 A 所在 VPC 的路由：

源网段	目标网段	下一跳	路由类型	状态	描述	操作
172.17.0.0/16	192.168.0.0/16	gz-to-sz	对等连接	可用	gz pod -> sz node	编辑 删除
172.17.0.0/16	172.16.0.0/16	gz-to-sz	对等连接	可用	gz pod -> sz pod	编辑 删除
test-middle(10.0.0.0/24)	172.16.0.0/16	gz-to-sz	对等连接	可用	gz node -> sz pod	编辑 删除
test-middle(10.0.0.0/24)	192.168.0.0/16	gz-to-sz	对等连接	可用	gz node -> sz node	编辑 删除

集群 B 所在 VPC 的路由：

源网段	目标网段	下一跳	路由类型	状态	描述	操作
172.16.0.0/16	172.17.0.0/16	sz-to-gz	对等连接	可用	sz pod -> gz pod	编辑 删除
172.16.0.0/16	10.0.0.0/8	sz-to-gz	对等连接	可用	sz pod -> gz node	编辑 删除
系统预定义子网(192.168.0....	172.17.0.0/16	sz-to-gz	对等连接	可用	sz node -> gz pod	编辑 删除
系统预定义子网(192.168.0....	10.0.0.0/8	sz-to-gz	对等连接	可用	sz node -> gz node	编辑 删除

容器网段空间耗尽如何继续扩容(VPC-ENI模式)

创建VPC-ENI容器网络模式的CCE集群时，需要选择VPC的子网作为容器子网，同时集群创建的Pod将从容器子网中分配IP地址。

随着集群规模的扩大，容器子网中的IP资源可能耗尽，本文将介绍如何给集群添加新的容器子网实现扩容容器网段。

使用限制

仅支持容器网络模式为**VPC-ENI**的集群。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的”**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**集群详情>基本信息**”进入集群详情页面。
6. 单击“添加子网”操作，在弹框中根据业务规划和子网剩余IP可用数勾选目标子网。
7. 单击“**确认**”按钮完成扩容容器子网。

使用Network Policy网络策略

Network Policy是Kubernetes提供的一种资源，用于定义基于Pod的网络隔离策略。它通过自定义的规则，描述了Pod间能否通信。本文将介绍如何在CCE集群开启Network Policy以及常见的配置示例。

前置说明

- 仅支持CCE标准独立集群和CCE托管集群，集群开启Network Policy后对集群网络性能有一定的影响。
- Network Policy网络策略依赖于网络插件实现，不同的网络配置行为不同。集群未开启**eBPF增强**时，网络策略插件由Calico-felix（仅支持veth网络模式）实现。集群开启**eBPF增强**时，网络策略插件由Cilium（仅支持veth网络模式）实现。网络策略插件的行为请参考社区文档。建议您充分理解、验证后启用网络策略。

操作步骤

创建集群时开启Network Policy

1. 登录容器引擎控制台，在左侧导航栏中选择**集群列表**。
2. 在“集群列表”页面单击**创建**，进入创建集群页面。
3. 在“创建集群”页面的“网络配置”中选中**开启 Network Policy 支持**配置。

> **注意**
>
> 勾选支持Network Policy集群将默认支持基于策略的网络控制（若未勾选eBPF增强，网络策略插件由Calico-felix实现，若勾选eBPF增强，网络插件由Cilium实现）

Network Policy配置示例

示例

- 禁止namespace=staging中Pod被访问。

```
 apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: staging
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

- 允许namespace=staging中Pod被访问。

```
 apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
  namespace: staging
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

- 禁止namespace=production中Pod对外访问。

```
 apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: production
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

- 允许namespace=production中Pod对外访问。

```
  apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
  namespace: production
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

- 禁止所有Pod的入和出的流量。

```
  apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

语义说明

参数	说明
PodSelector	选中需要隔离的Pod
policyTypes	策略类型，NetworkPolicy 将流量分为ingress和egress，即入方向和出方向。如果没有指定则表示不闲置
ingress	入方向，白名单，需要指定 from、ports，即来源、目的端口号，from有三种类型，ipBlock/namespaceSelector/podSelector
egress	出方向，白名单，类似ingress，egress需要指定to、ports，即目的、目的端口号

网络连通性

容器网络通过 NAT 网关访问公网

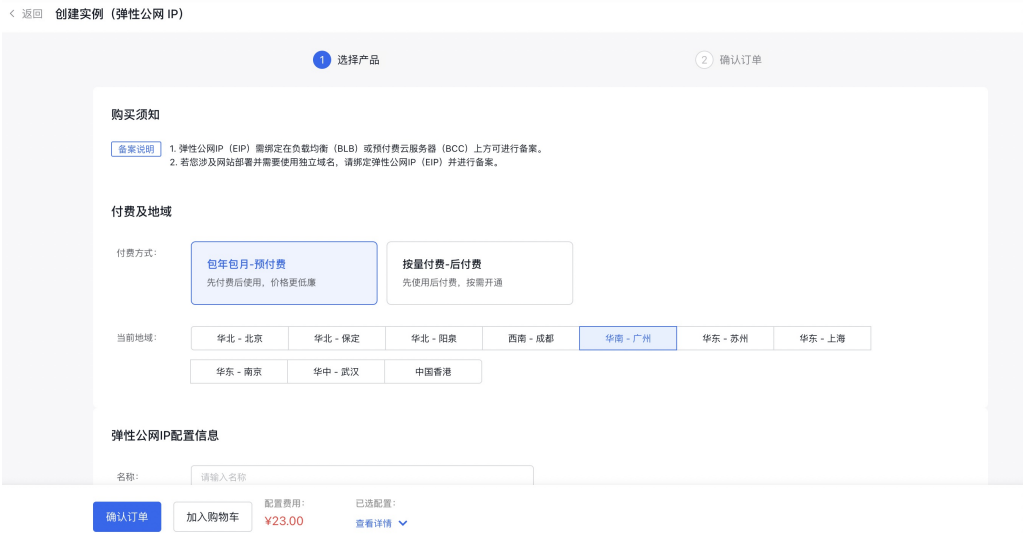
1. 概述

在一些场景中，用户需要让CCE集群中的节点和Pod具备访问公网的能力，如通过 `yum install` 等命令访问外部源地址下载安装软件、容器需要访问公网中的某些服务等等。 我们提供了以下两种方案，可以实现集群访问公网，用户可以根据具体需求自行选择。 本文档主要介绍了 `VPC-ENI` 和 `VPC 路由` 两种容器网络模式下，容器如何访问互联网。 **1.1 相关概念**

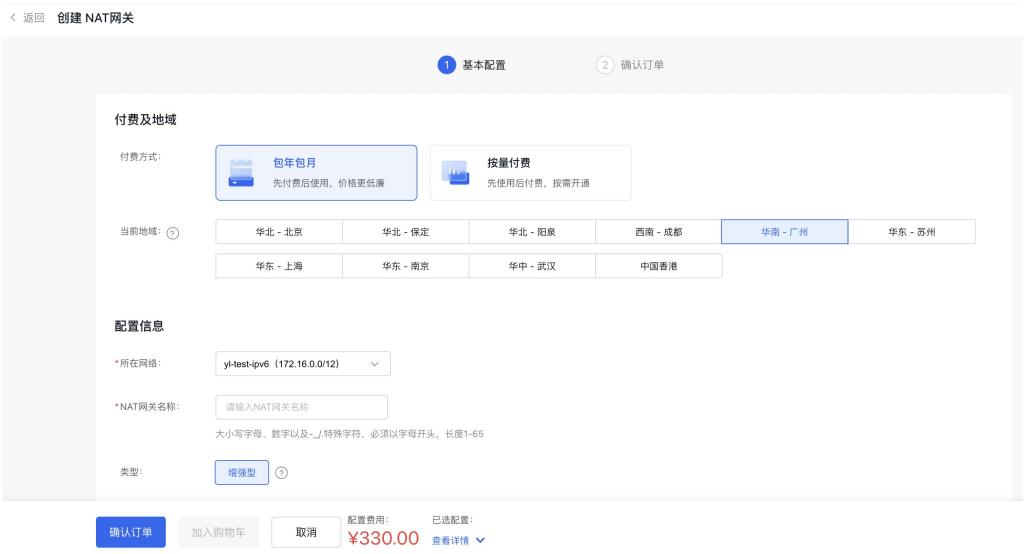
- EIP：EIP是百度智能云提供的弹性公网IP服务，EIP可以挂载到CCE节点、BLB、NAT网关等服务，提供可以访问公网的地址。
- NAT网关：百度智能云 NAT 网关支持用户VPC子网中的BCC、DCC等实例连接公网，通过NAT网关实现内网IP转换为公网IP地址。 节点子网：用户在创建CCE工作节点时，可以选择节点所在的VPC子网。子网类型为通用型子网，通用型子网中的节点可以挂载EIP直接访问公网，也可以通过NAT网关实现公网访问。

2. 前置准备

要完成 Pod 访问互联网的配置任务，需要您已经购买了和配置了以下产品。 **2.1 创建 EIP** 进入到弹性公网 IP 控制台，选择创建实例，购买新的EIP。

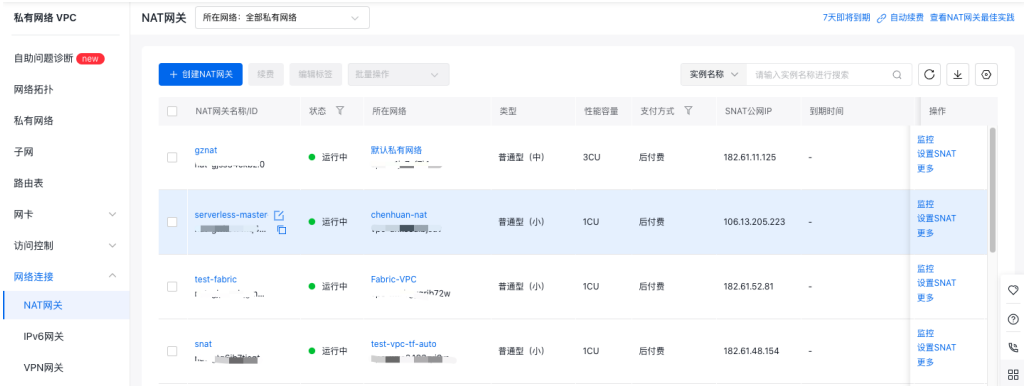


2.2 创建 NAT 网关 进入到私有网络 VPC 控制台，选定对应 VPC 后，创建 NAT 网关实例。并将刚创建的 EIP 实例，配置在 NAT 网关的 SNAT 公网 IP 的复选框。



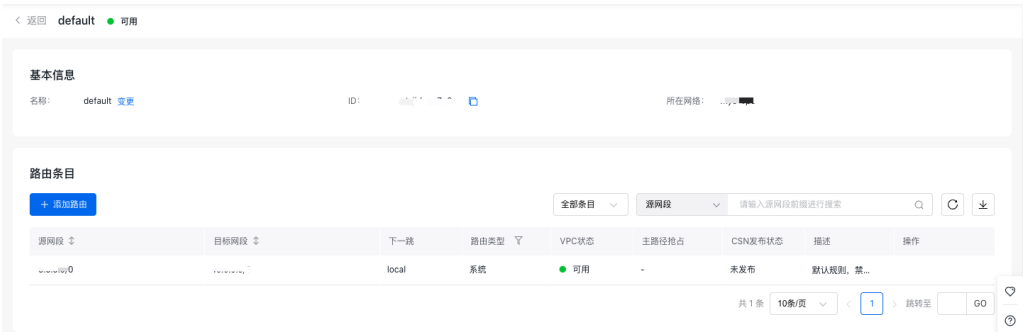
3. 开启容器访问公网能力

3.1 开启 Node 访问公网能力 如果需要 Node 节点有直接访问公网能力，需要进行两项配置：SNAT 规则和自定义路由配置。 **3.1.1 SNAT 规则配置** 在 私有网络 VPC -> NAT 网关控制台，选中 NAT 网关实例，操作设置 SNAT，并将需要访问公网的 Node 所在子网加入到新 SNAT 条目中。



- 源网段选择要允许访问公网的 Node 所在的子网
- 公网 IP 地址选择我们对应的 EIP 实例

3.1.2 自定义路由规则设置 在私有网络 VPC -> 路由表控制台，进入目标 VPC 的路由表，添加新路由条目。



- 源网段选择要允许访问公网的 Node 所在的子网。
- 目标网段选择 0.0.0.0/0或者你想访问的外网服务的 CIDR。
- 路由类型选择 NAT 网关
- 下一条实例选择已配置SNAT规则的 NAT 网关

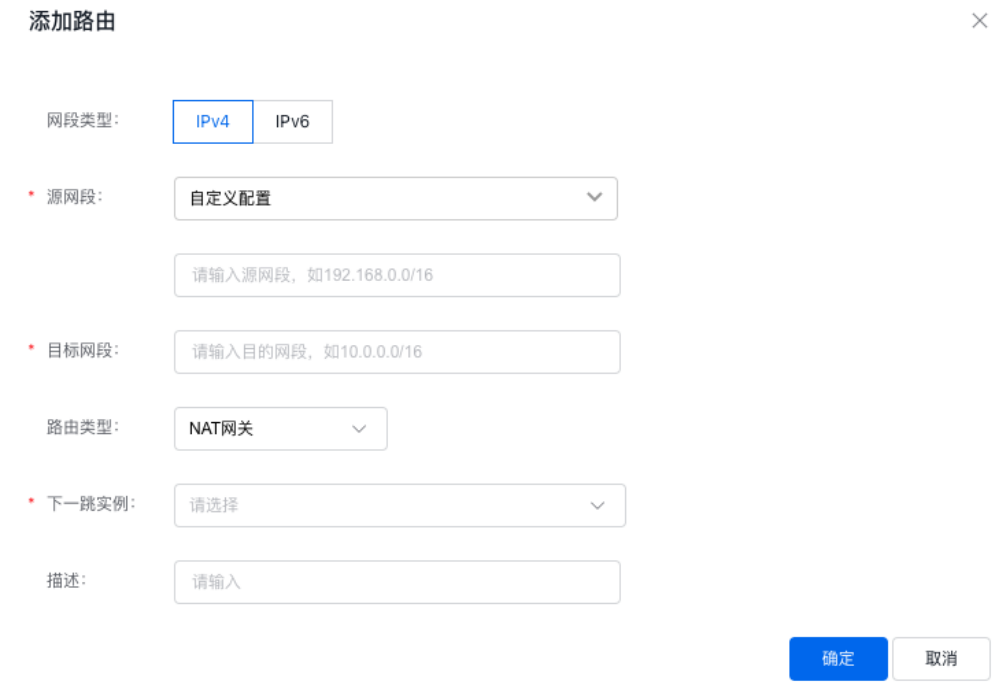
至此 Node 访问公网就配置完成，可以进入节点进行网络联通性验证。 **3.2 VPC-ENI 模式容器访问公网能力（可选） 3.2.1 确认容器网络模式信息** 容器网络模式由创建集群时选定，进入 CCE -> 选中已有集群可以进入集群详情页面，再次确认容器网络模式和Pod子网信息。案例集群使用 VPC-ENI容器网络，并配置了 2 个子网。

3.2.2 SNAT 规则配置 在 私有网络 VPC -> NAT 网关控制台，选中 NAT 网关实例，操作设置 SNAT，并将需要访问公网的容器所在子网加入到新 SNAT 条目中。

- 源网段选择要允许访问公网的容器所在的子网
- 公网 IP 地址选择我们对应的 EIP 实例

> **注意**
>
> 每个需要访问公网的容器所在的子网，都需要主动添加 SNAT 规则。

3.2.2 自定义路由规则设置 在私有网络 VPC -> 路由表控制台，进入目标 VPC 的路由表，添加新路由条目。把所有需要访问公网的容器子网添加一条指向 NAT 网关的路由。



- 源网段选择要允许访问公网的 Pod 所在的子网。
- 目标网段选择 0.0.0.0/0或者你想访问的外网服务的 CIDR。
- 路由类型选择 NAT 网关
- 下一条实例选择已配置SNAT规则的 NAT 网关

至此 VPC-ENIi 容器网络下 Pod 访问公网就配置完成，可以进入容器进行网络联通性验证。

3.3 VPC 路由模式容器访问公网（可选） 3.3.1 确认容器网络模式信息 容器网络模式由创建集群时选定，进入 CCE -> 选中已有集群可以进入集群详情页面，再次确认容器网络模式和Pod子网信息。案例集群使用 VPC路由容器网络，并配置容器网络的网段是 172.21.0.0/16。

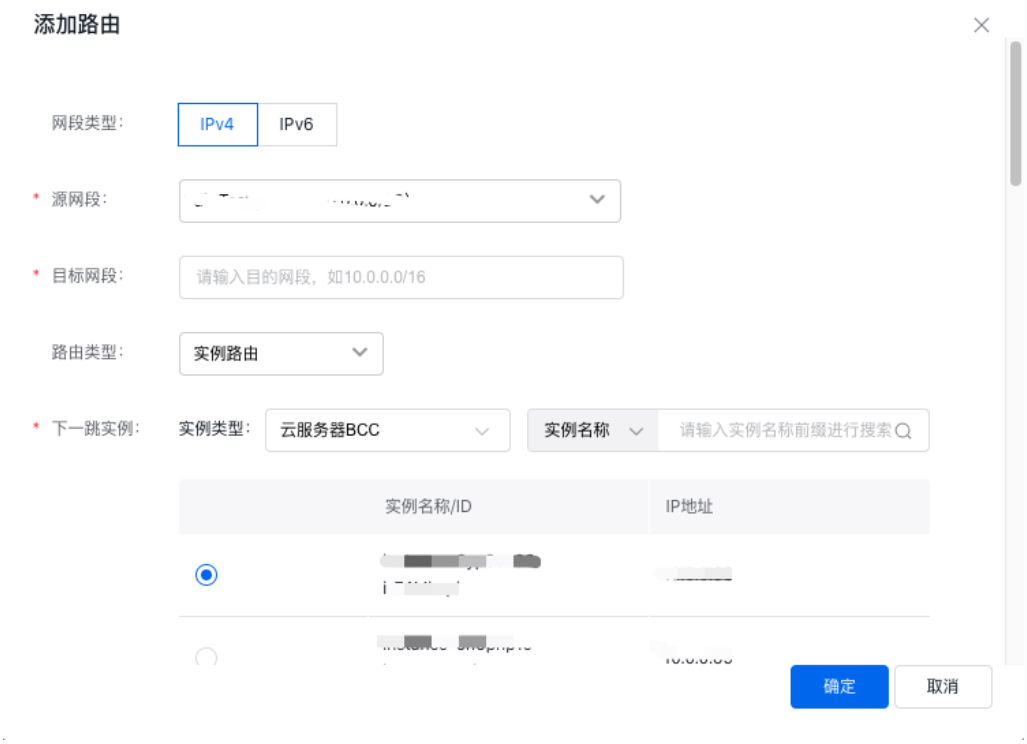
3.3.2 SNAT 规则配置 在 私有网络 VPC -> NAT 网关控制台，选中 NAT 网关实例，操作设置 SNAT，并将需要访问公网的容器所在子网加入到新 SNAT 条目中。

- 源网段选择自定义配置，并输入要允许访问公网的容器网段，例如 10.22.0.0/16。
- 公网 IP 地址选择我们对应的 EIP 实例。

3.3.3 开启路由中继 在私有网络 VPC 控制台，选择目标 VPC 实例打开路由中继开关。VPC 路由容器网络，Pod 的 IP 地址不在 VPC 地址段内，若要添加源网段是 VPC 外的地址的自定义路由，需要打开路由中继能力。

> 开启路由中继功能后，路由表可以转发非源自本VPC的流量。该选项默认关闭，即路由表默认仅会转发源自本VPC的流量。

3.3.4 自定义路由规则设置 在私有网络 VPC -> 路由表控制台，进入目标 VPC 的路由表，添加新路由条目。把所有需要访问公网的容器子网添加一条指向 NAT 网关的路由。



- 源网段选择自定义配置，手动输入 Pod 自定义的网段，例如10.22.0.0/16。
- 目标网段选择 0.0.0.0/0或者你想访问的外网服务的 CIDR。
- 路由类型选择 NAT 网关。
- 下一条实例选择已配置SNAT规则的 NAT 网关。

至此 VPC 路由容器网络下 Pod 访问公网就配置完成，可以进入容器进行网络联通性验证。

4. 常见问题

4.1 ACL 规则导致容器无法访问公网问题 4.1.1 安全组导致容器无法访问公网 在按照本文档给定的步骤如果仍无法访问外网，请逐个检查节点、ENI绑定的安全组、ACL 等网络策略规则，确保容器访问外网的流量在规则中都被放行。 部分互联网主机会禁止 ICMP 流量，当 ping 不通时，可以考虑使用 TCP/UDP 检测联通性。 **4.2 DNS导致的容器无法访问互联网** 在容器访问互联网前，请先检查目标域名是否能被解析。如域名无法被解析，可通过修改配置 DNS 完成域名解析的请求。例如增加配置 coredns 下游域名服务器，完成对域名解析请求。通过下面命令可以验证域名能否被成功解析：

```
dig a.example.com
```

网络维护

CCE容器网络常见错误码对应表

概述

本文档适用于CCE容器网络常见问题排查，文档解释了 CCE 容器网络组件中出现的所有的事件编码并给出解决方案。

1. Pod 错误事件

Pod 错误事件主要描述 Pod 无法正常启动的情况。Pod 的网络错误事件信息见于 Pod 对象上，可以通过 `kubectl describe pod {podName}` 查询到所有 Pod 相关的事件。 **1.1 CNI 错误** CNI 错误指的是在单机创建Pod时，容器网络组件无法正常创建 Pod 的网络资源。CCE 对创建 Pod 失败的原因进行了分类，并给出了解决方案。

1.1.1 VPC-ENI 模式 | 错误代码 | 解释 | 触发条件 | 解决方案 | | --- | --- | --- | --- | | ENIIPCapacityExceed | ENI 上挂载的辅助 IP 数量达到配额限制 | 容器网络组件创建的 ENI 上辅助 IP 数量超过配额限制。
点击查看ENI 配额 | 新增节点，并将 Pod 调度到新节点。 | | ENICapacityExceed | ENI 数量达到配额限制 | 节点上挂载的 ENI 已达到配额，且 ENI 已绑定的所有辅助 IP 已经达到配额限制。
点击查看ENI 配额 | 新增节点，并将 Pod 调度到新节点。 | | WaitCreateMoreENI | 等待创建新的 ENI | 节点上所有已经挂载的 ENI 的辅助 IP 耗尽，正在等待创建新的 ENI | 等待kubelet自动重试创建 Pod 网络资源 | | IPPoolExhausted | 缓存池中的 IP 耗尽 | 容器网络预申请的 IP 地址池中的 IP 全部分配完成，正在等待容器网络组件预申请新的 IP | 等待kubelet自动重试创建 Pod 网络资源 | | SubnetNoMoreIP | ENI 已绑定的子网 IP 耗尽 | ENI 已绑定的子网 IP 耗尽，无法为 ENI 绑定新的辅助IP。 | **方案1.** 新建与节点同可用区更大区间的子网，并添加新子网到 CCE 集群.然后删除当前节点，并购买新的同可用区节点加入到 CCE 集群。
方案2. 使用指定子网分配 IP 。 | | failed to set bandwidth | 容器网络组件无法为 ENI 设置带宽管理 | 容器网络组件无法为 Pod 设置带宽，如 内核

版本过低等。 | 联系客服解决 |

1.1.2 VPC-Route 模式 | 错误代码 | 解释 | 触发条件 | 解决方案 | | --- | --- | --- | --- | | NoMoreIP | 节点没有可用 IP | 节点上所有可用 IP 都分配完毕，无法为 Pod 分配新的 IP。 | VPC-Route 扩容容器网段|

2. 节点错误事件

CCE 容器网络中与节点相关的网络事件主要包含两类，Node 和网络资源错误事件。 **2.1 Node 错误事件** Node 错误事件主要描述节点无法正常启动的情况。Node 的网络错误事件信息见于k8s 的 Node 对象上，可以通过 `kubectl describe node {nodeName}` 查询到所有 Node 相关的事件。

错误代码	解释	触发条件	解决方案
MetaAPIError01	容器网络组件无法通过 MetaAPI 获取机器实例	在不受支持的机型上部署了 CCE 容器网络组件	联系客服解决
MetaAPIError02	容器网络组件无法通过 MetaAPI 获取机器地域等元数据信息	在不受支持的机型上部署了 CCE 容器网络组件	联系客服解决

2.2 网络资源错误事件 网络资源错误事件主要用于描述节点无法完成初始化，或者无法完成 ENI 等资源准备的情况。NetworkResourceSet 是 CCE 定义的网络资源管理对象，可以通过 `kubectl describe nrs {nodeName}` 查询到节点相关的所有网络相关事件.

错误代码	解释	触发条件	解决方案
VPCQuotaLimitExceeded	failed to create vpc route rule 由于VPC 资源配额受限，无法创建更多 VPC 路由规则。 点击查看VPC 配额	VPC 路由规则达到上限	在配额中心提出申请，联系客服解决
CreateRouteRuleFailed	容器网络组件无法通过 open API 创建路由	访问 open API 遇到了偶发问题	CCE 会在 2 分钟内自动恢复，如长时间未恢复，请联系客服
FailedCreateENI	容器网络组件无法通过 open API 创建 ENI	访问 open API 遇到了偶发问题	CCE 会在 2 分钟内自动恢复，如长时间未恢复，请联系客服
NoAvailableSubnet	节点所在的可用区没有可用的子网	节点所在的可用区没有可用的子网，节点无法创建新 ENI	新建与节点同可用区的子网，并添加新子网到 CCE 集群
FailedBorrowSubnet	集群配置的子网不存在	集群配置的子网子网不可用	联系客服解决

网络编排

容器网络 QoS 管理

容器网络 QoS

CCE 提供了两种对容器网络进行 QoS 管理的方式：带宽管理和出口数据包优先级管理。通过在 Pod 上设置 annnotation，可以控制容器网络 QoS。

1. 带宽管理 带宽管理配置如下：

annotation	描述	示例
`kubernetes.io/ingress-bandwidth`	容器 ingress 带宽	10M
`kubernetes.io/egress-bandwidth`	容器 egress 带宽	10M

1.1 带宽限制的单位 带宽管理支持的单位有：

- `B`: 字节
- `K`/`KB`/`KIB`：千字节
- `M`/`MB`/`MIB`：兆字节
- `G`/`GB`/`GIB`：吉字节
- `T`/`TB`/`TIB`：太字节

1.2 带宽管理原理 CCE 容器网络当前仅支持使用 tc 实现带宽管理，通过 tc 配置容器网卡的 ingress 和 egress 的 tbf qdisc，实现容器网络 ingress 和 egress 的限速。 关于 tbf可以在 linux man 手册中查看 。

2. 出口数据包优先级管理 出口数据包优先级管理配置如下：

annotation	描述	示例
`cce.baidubce.com/egress-priority`	容器 egress 数据包优先级	Burstable

2.1 出口数据包优先级取值

- `Guaranteed`: 最高优先级，用于需要极低延迟的场景

- `Burstable`: 普通优先级，用于需要高吞吐的场景
- `BestEffort`: 低优先级，用于延迟不敏感的场景

2.2 出口数据包优先级管理原理 CCE 容器网络当前仅支持使用 tc 实现出口数据包优先级管理，例如在 VPC-ENI 场景下，如果设置了数据包出口优先级，则 ENI 上的 tc 示例配置如下： cce-eni-0

- qdisc mq 1: dev cce-eni-0 root
- qdisc prio 8001: dev cce-eni-0 parent 1:2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

```
  filter protocol ip pref 1 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0 terminal flowid ??? not_in_hw  match ac16161e/ffffffff at 12
```

- qdisc prio 8002: dev cce-eni-0 parent 1:1 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

```
  filter protocol ip pref 1 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0 terminal flowid ??? not_in_hw  match ac16161e/ffffffff at 12
```

通过为 ENI 网卡设置 mq qdisc，再为每个 mq 队列设置 prio qdisc，再为每个 prio 队列设置 filter，实现容器网络出口数据包优先级管理。

使用限制

1. annotation 需要在创建 Pod 时配置，不支持动态修改。
2. 网络 QoS 需要配合 cce-network-v2 2.9.0 及以上版本使用。
3. QoS 当前仅支持 linux 操作系统，且要求内核版本 3.10 以上。

集群 Pod 子网拓扑分布（容器网络 v2）

概览

VPC-ENI容器网络模式支持指定子网为Pod分配IP，用户可以使用这个能力为不同的业务应用Pod规划分配不同子网的IP地址。本文档介绍了一种对`psts`的扩展，即集群级子网拓扑分布策略（`cpsts`）。

> 说明：使用该功能需要联系百度智能云客服开启白名单。

需求场景 `psts` 提供了 3 种分配 IP 的策略，可以用于解决动态 IP 地址分配、手动 IP 地址分配和固定 IP 的场景问题。但一个`psts`仅在一个 namespace 中生效，如果需要为不同业务应用有相同的 IP 地址分配策略，则需要创建多个`psts`，重用起来比较复杂。 `cpsts` 提供了集群级子网拓扑分布策略，只需要创建一个`cpsts`对象，可以在多个 namespace 下生效，解决多个 namespace 的 IP 地址分配策略问题。

方案概述

如下CRD数据结构所示，CCE提供了自定义CRD来指定Pod子网拓扑分布策略，用于在CCE集群中实施制定子网分配IP功能。

关键数据结构介绍

集群Pod子网拓扑分布 ClusterPodSubnetTopologySpread

集群子网拓扑分布对象(`cpsts`)在工作时会根据`namespaceSelector`在符合条件的 Namespace 下创建Pod子网拓扑分布(`psts`)对象。并依赖`psts` 影响 Pod 的 IP 地址分配。它的数据结构定义如下：

```
  apiVersion: cce.baidubce.com/v2alpha1
kind: ClusterPodSubnetTopologySpread
metadata:
  name: cpsts-1
spec:
  name: cpsts-1
  # 选择 cpsts 对象生效的 namespace
  namespaceSelector:
    matchLabels:
      zrq: test
  # 选择在每个 namespace 下 cpsts 对象生效的 Pod
  selector:
    matchLabels:
      app: redis
  subnets:
    # 当不填写指定一段子网内 IP 范围时，子网格式如下，注意中括号
    # sbn-6mrkdcsyzpaw: []
    sbn-6mrkdcsyzpaw:
    # 非必须，本子网内指定一段 IP 的范围，不填表示使用子网cidr IP 范围
      - family: 4
        range:
        - start: 10.0.0.2
          end: 10.0.0.254
  strategy:
    releaseStrategy: TTL
    ttl: 168h0m0s
    type: Elastic
    # 开启 IP 重用
    enableReuseIPAddress: true
  whenUnsatisfiable: DoNotSchedule
```

该对象的核心字段如下：

域	数据类型	必须	默认值	描述
name	string			拓扑分布对象名。
priority	int32	否	0	多个子网拓扑约束之间的优先级，数字越大，优先级越高。
namespaceSelector	object	否		使用该条件对 namespace 做标签匹配，符合条件的 namespace 中将会创建 psts 子对象。 如果不填写这个选项，则 cpsts 默认在所有的 namespace 下都创建一个名为 {namespace}-{cpstsName} 的psts对象。
selector	object	否		使用该条件对Pod做标签匹配，符合条件的Pod将在分配IP地址时使用该规则。如果selector为空，则同namespace下的所有Pod都匹配该规则。
subnets	object	是		策略要使用的子网。CCE会从这些子网中为Pod分配IP地址。 **注意：如果不指定下方字段，这里内容应写成 sbn-xxxxxxx: [] 的格式；如果指定下方字段应写成 sbn-xxxxxxx: 的格式**
subnets.[].family	string	否		IP 地址协议族。取值 "4" 或 "6"。 **注意：此处四个字段，应一同填写或不填**
subnets.[].range	array	否		IP 地址范围 **注意：此处四个字段，应一同填写或不填**
subnets.[].range[].start	string	否		起始 IP 地址 **注意：此处四个字段，应一同填写或不填**
subnets.[].range[].end	string	否		结束 IP 地址 **注意：此处四个字段，应一同填写或不填**
strategy	object	是		IP 地址使用和回收策略
strategy.type	string	是	Elastic	Elastic: 动态分配 IP 地址，可以使用任意工作负载； Fixed：永久固定 IP 地址，仅搭配 sts 工作负载使用；PrimaryENI：独占 ENI 专用 IP 地址
strategy.releaseStrategy	string	是	TTL	IP 地址释放策略。 TTL： Pod 被删除后，IP 地址随时间过期。其中在动态 IP 分配模式下，Pod 删除后，IP 立刻回收。 开启 enableReuseIPAddress 时，默认 7 天回收。 Never： 仅搭配 strategy.type: Fixed 使用，代表永不回收。
strategy.enableReuseIPAddress	bool	否	false	是否在 strategy.type: Elastic 的场景下开启 IP 重用。如开启 IP 重用，则在 IP 地址过期之前，如果有同名的 Pod 重复创建，则尽力去复用 IP，以达到类似固定 IP 的效果。 **注意，如果指定了range字段，需要这里置为true。**
strategy.ttl	string	否	168h0m0s	在开启 IP 地址重用时，Pod 删除后，保留 IP 的时间。默认值是 7 天(168h0m0s)

使用限制

1. 该功能需使用VPC的ENI跨子网分配IP功能，请提交工单申请使用ENI跨子网分配IP功能。
2. kube-system 命名空间的Pod无法使用指定子网分配IP功能。
3. 使用ipRange能力时，注意IP范围内不要包含等特殊IP地址（如IPv4网络地址、网关地址、广播地址、多播地址），否则可能产生无法正常分配IP。
4. 指定子网的Pod只能调度到子网所在可用区的节点，请确认可用区有Ready状态的节点。
5. 固定IP和IP复用场景下只能使用专属子网（指定的子网只能供单个CCE集群使用），专属子网不支持变更为普通子网、不支持从集群内删除，详情见专属子网说明。
6. 该功能仅适用于容器网络 v2 版本的集群。

> **专属子网**： 当用户需要为Pod分配指定子网下的某几个IP时，IP所属的子网将自动被标记为手动分配IP模式。手动分配IP模式下的子网有以下特征：

1. 专属子网应为当前CCE集群专属，CCE会自动为子网增加专属标签，其它的CCE集群拒绝使用该子网。（用户可以操作百度云其它产品使用

该子网）

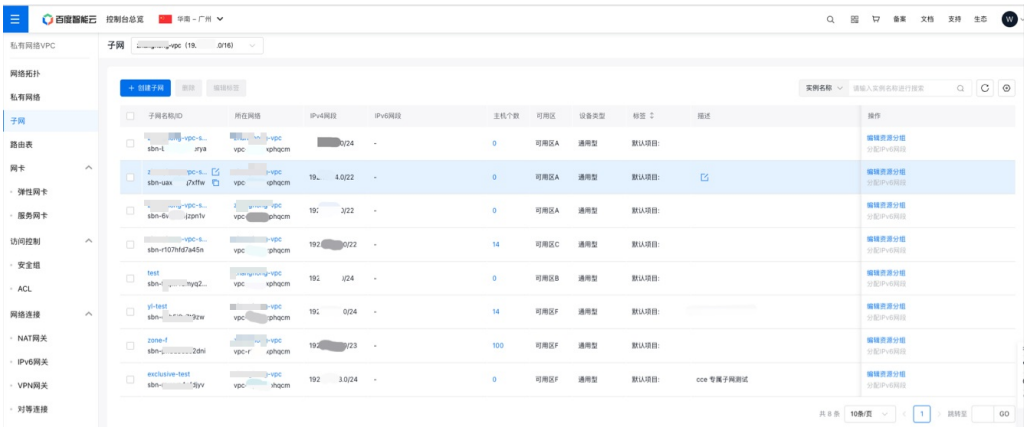
2. 对专属子网仅支持手动分配IP，不会自动分配IP。这需要用户自己管理IP地址规划
3. 专属子网下IP和Pod的关系可以有优先分配和固定绑定两种。固定绑定策略使用Pod名作为绑定标识，同一个Pod名的IP地址相同
4. 集群Pod的默认子网不能是专属子网，否则可能导致集群中的其它Pod无法正常分配IP
5. 专属子网不支持变更为普通子网、不支持从集群内删除

配置步骤

环境准备

创建私有子网 在百度智能云VPC控制台->子网选项卡，为自己的VPC新建一个子网，并保存子网ID （以sbn-xxx格式命名的是子网ID）。注意创建子网时应选择与CCE集群节点相关的可用区，否则可能导致调度失败。

> 说明： 如需使用ENI跨子网分配IP功能，请提交工单申请。



在CCE上操作指定子网分配IP

场景1： 全集群多应用固定 IP 从与Pod匹配的子网拓扑约束策略关联的子网中动态为Pod分配IP。匹配了子网拓扑约束策略（cpsts）的所有Pod只从策略包含的子网分配IP地址。 适用场景：

- 以子网维度对流量做统计
- 以子网维度做安全策略，如ACL规则控制
- 通过NAT网关为特定子网开启互联网访问 **1. 创建cpsts**

```
  apiVersion: cce.baidubce.com/v2alpha1
kind: ClusterPodSubnetTopologySpread
metadata:
  name: cpsts-1
spec:
  name: cpsts-1
  # 选择 cpsts 对象生效的 namespace
  namespaceSelector:
    matchLabels:
      zrq: test
  # 选择在每个 namespace 下 cpsts 对象生效的 Pod
  selector:
    matchLabels:
      app: fixedIPApp
  subnets:
    sbn-6mrkdcsyzpaw:
      # 非必须，固定 IP 的范围，不填表示使用子网默认 IP 范围
      - family: 4
        range:
        - start: 10.0.0.2
          end: 10.0.0.254
  strategy:
    releaseStrategy: TTL
    ttl: 168h0m0s
    type: Elastic
    # 开启 IP 重用
    enableReuseIPAddress: true
  whenUnsatisfiable: DoNotSchedule
```

2. 创建工作负载

```
  apiVersion: apps/v1
###### 必须使用有状态工作负载
kind: StatefulSet
metadata:
  name: foo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fixedIPApp
  serviceName: foo
  template:
    metadata:
      labels:
        workloadType: sts
        fixedIP: "true"
        app: fixedIPApp
    spec:
      containers:
        - image: nginx
          name: nginx
```

3. 验证IP分配结果

```
  ###### kubectl get pod {podName} -oyaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cce.baidubce.com/PodSubnetTopologySpread: example-subnet-topology
  labels:
    app: foo
  name: foo-0
  namespace: default
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - zoneF
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: nginx
    resources:
      limits:
        cce.baidubce.com/ip: "1"
      requests:
        cce.baidubce.com/ip: "1"
```

相关产品

- 私有网络VPC

VPC-ENI指定子网分配IP（容器网络 v2）

概览

VPC-ENI容器网络模式支持指定子网为Pod分配IP，用户可以使用这个能力为不同的业务应用Pod规划分配不同子网的IP地址。

> 说明：使用该功能需要联系百度智能云客服开启白名单。

需求场景 需求场景1：指定子网动态分配IP

从与Pod匹配的子网拓扑约束策略关联的子网中动态为Pod分配IP。匹配了子网拓扑约束策略（`psts`）的所有Pod只从策略包含的子网分配IP地址。

场景2：手动分配 IP 手动分配 IP 策略的含义是给定一个IP列表，CCE从这个IP列表中给 Pod 分配 IP 。Pod 删除后 IP 地址保留，在 IP 地址过期之前，Pod重建或者迁移后，IP地址仍然不变。默认情况下，Pod 删除后，IP 地址会保留 7 天。

场景3：固定IP 固定 IP 策略的含义是给定一个IP列表，CCE从这个IP列表中给有状态负载的Pod分配IP。Pod删除后IP地址保留，Pod重建或者迁移后，IP地址仍然不变。 无论 Pod 是否删除，已经分配的IP地址不会被释放，直到工作负载被删除后，IP 地址才会被回收。

方案概述

如下CRD数据结构所示，CCE提供了自定义CRD来指定Pod子网拓扑分布策略，用于在CCE集群中实施制定子网分配IP功能。

关键数据结构介绍

Pod子网拓扑分布 PodSubnetTopologySpread

子网拓扑分布对象是指定子网分配IP的核心工作对象，它的核心数据结构定义如下：

```
  apiVersion: cce.baidubce.com/v2
kind: PodSubnetTopologySpread
metadata:
  name: example-subnet-topology
  namespace: default
spec:
  # 拓扑分布对象名
  name: example-subnet-topology
  # 多个子网拓扑约束之间的优先级，数字越大，优先级越高。默认值0
  priority: 0
  subnets:
    # 必须是与当前集群同VPC的子网ID，格式为sbn-*例如sbn-ccfud13pwcqf
    # 使用专属子网，用户要确认该子网只有当前CCE集群使用
    sbn-ccfud13pwcqf: []
  strategy:
    releaseStrategy: TTL
    ttl: 168h0m0s
    type: Elastic
  # 选择要使用该子网拓扑分布的Pod
  selector:
    matchLabels:
      app: foo
```

该对象的核心字段如下：

域	数据类型	必须	默认值	描述
name	string			拓扑分布对象名，在通过 PodSubnetTopologySpread 创建子网拓扑分布时必填。
priority	int32	否	0	多个子网拓扑约束之间的优先级，数字越大，优先级越高。
selector	object	否		使用该条件对Pod做标签匹配，符合条件的Pod将在分配IP地址时使用该规则。如果selector为空，则同namespace下的所有Pod都匹配该规则。
subnets	object	是		策略要使用的子网。CCE会从这些子网中为Pod分配IP地址。
subnets.[].family	string	是		IP 地址协议族。取值 "4" 或 "6"
subnets.[].range	array	是		IP 地址范围
subnets.[].range[].start	string	是		起始 IP 地址
subnets.[].range[].end	string	是		结束 IP 地址
strategy	object	是		IP 地址使用和回收策略
strategy.type	string	是	Elastic	Elastic: 动态分配 IP 地址，可以使用任意工作负载； Fixed：永久固定 IP 地址，仅搭配 sts 工作负载使用；PrimaryENI：独占 ENI 专用 IP 地址
strategy.releaseStrategy	string	是	TTL	IP 地址释放策略。 TTL： Pod 被删除后，IP 地址随时间过期。其中在动态 IP 分配模式下，Pod 删除后，IP 立刻回收。 开启 enableReuseIPAddress 时，默认 7 天回收。 Never： 仅搭配 strategy.type: Fixed 使用，代表永不回收。
strategy.enableReuseIPAddress	bool	否	false	是否在 strategy.type: Elastic 的场景下开启 IP 重用。如开启 IP 重用，则在 IP 地址过期之前，如果有同名的 Pod 重复创建，则尽力去复用 IP，以达到类似固定 IP 的效果。
strategy.ttl	string	否	168h0m0s	在开启 IP 地址重用时，Pod 删除后，保留 IP 的时间。默认值是 7 天(168h0m0s)

使用限制

1. 该功能需使用VPC的ENI跨子网分配IP功能，请提交工单申请使用ENI跨子网分配IP功能。
2. kube-system 命名空间的Pod无法使用指定子网分配IP功能。
3. 在使用 ipRange 功能时，请确保指定的IP范围不包括特殊IP地址，如IPv4的网络地址、网关地址、广播地址和多播地址。包含这些特殊地址可能会导致IP分配出现问题。
4. 指定子网的Pod只能调度到子网所在可用区的节点，请确认可用区有Ready状态的节点。
5. 固定IP和IP复用场景下只能使用专属子网（指定的子网只能供单个CCE集群使用），专属子网不支持变更为普通子网、不支持从集群内删

除，详情见专属子网说明。

6. 该功能仅适用于容器网络 v2 版本的集群。

> **专属子网**： 当用户需要为Pod分配指定子网下的某几个IP时，IP所属的子网将自动被标记为手动分配IP模式。手动分配IP模式下的子网有以下特征：

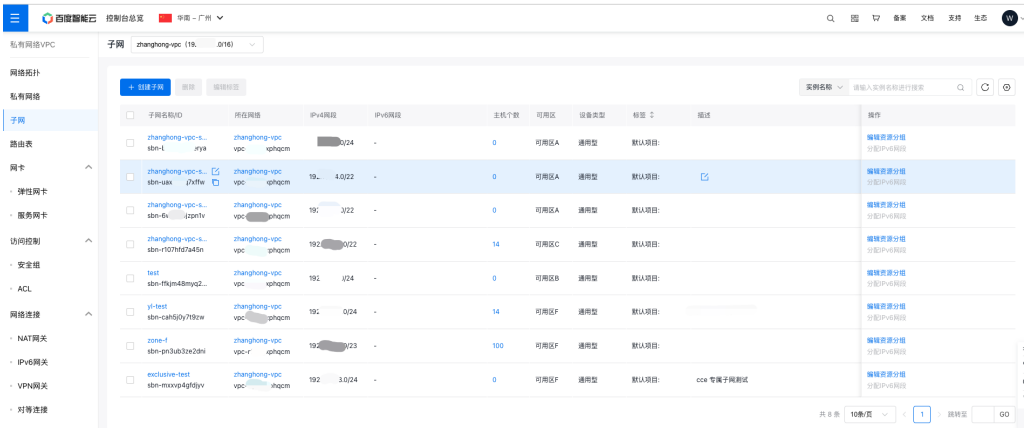
1. 专属子网应为当前CCE集群专属，CCE会自动为子网增加专属标签，其它的CCE集群拒绝使用该子网。（用户可以操作百度云其它产品使用该子网）
2. 对专属子网仅支持手动分配IP，不会自动分配IP。这需要用户自己管理IP地址规划
3. 专属子网下IP和Pod的关系可以有优先分配和固定绑定两种。固定绑定策略使用Pod名作为绑定标识，同一个Pod名的IP地址相同
4. 集群Pod的默认子网不能是专属子网，否则可能导致集群中的其它Pod无法正常分配IP
5. 专属子网不支持变更为普通子网、不支持从集群内删除

配置步骤

环境准备

创建私有子网 在百度智能云VPC控制台->子网选项卡，为自己的VPC新建一个子网，并保存子网ID （以`sbn-xxx`格式命名的是子网ID）。注意创建子网时应选择与CCE集群节点相关的可用区，否则可能导致调度失败。

> 说明： 如需使用ENI跨子网分配IP功能，请提交工单申请。



在CCE上操作指定子网分配IP

场景1： 指定子网动态分配IP 从与Pod匹配的子网拓扑约束策略关联的子网中动态为Pod分配IP。匹配了子网拓扑约束策略（`psts`）的所有Pod只从策略包含的子网分配IP地址。 适用场景：

- 以子网维度对流量做统计
- 以子网维度做安全策略，如ACL规则控制
- 通过NAT网关为特定子网开启互联网访问 **1. 创建psts**

```
  apiVersion: cce.baidubce.com/v2
###### pod 拓扑分布表
kind: PodSubnetTopologySpread
metadata:
  name: default
  namespace: default
spec:
  # 多个子网拓扑约束之间的优先级，排序越靠前，优先级越大
  priority: 0
  name: default-psts
  strategy:
    releaseStrategy: TTL
    type: Elastic
  subnets:
    # 必须是与当前集群同VPC的子网ID，格式为sbn-*例如sbn-ccfud13pwcqf
    sbn-ccfud13pwcqf: []
    sbn-e8rk4zxn2ys6: []
  # 选择要使用该子网拓扑分布的pod，如果为空则表示所有pod都使用该子网拓扑分布
    selector:
      matchLabels:
        app: foo
```

2. 创建工作负载

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: elastic-deploy
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: foo
  template:
    metadata:
      labels:
        app: foo
    spec:
      containers:
      - image: nginx
        name: nginx
```

3. 验证IP分配结果

```
  ###### kubectl get pod {podName} -oyaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cce.baidubce.com/PodSubnetTopologySpread: example-subnet-topology
  generateName: elastic-deploy-56dc49b486-
  labels:
    app: foo
  name: elastic-deploy-56dc49b486-d6z7b
  namespace: default
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - zoneF
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: nginx
    resources:
      limits:
        cce.baidubce.com/ip: "1"
      requests:
        cce.baidubce.com/ip: "1"
```

场景2：手动分配 IP 手动分配 IP 策略的含义是给定一个IP列表，CCE从这个IP列表中给 Pod 分配 IP 。Pod 删除后 IP 地址保留，在 IP 地址过期之前，Pod重建或者迁移后，IP地址仍然不变。默认情况下，Pod 删除后，IP 地址会保留 7 天。 **适用场景：**

- 固定Pod IP，要求Pod迁移后IP支持不变
- 使用专属子网，并完全手动管理该子网的IP
- Pod 多次重建后名字不变，例如使用有状态工作负载（`apps/v1 StatefulSet`）创建的 Pod

1. 创建 psts

```
  apiVersion: cce.baidubce.com/v2
kind: PodSubnetTopologySpread
metadata:
  name: example-subnet-topology
  namespace: default
spec:
  priority: 0
  subnets:
    sbn-6mrkdcsyzpaw:
    # 非必须，固定 IP 的范围，不填表示使用子网默认 IP 范围
    - family: 4
      range:
      - start: 10.0.0.2
        end: 10.0.0.254
  strategy:
    releaseStrategy: TTL
    ttl: 168h0m0s
    type: Elastic
    # 必须，开启 IP 重用
    enableReuseIPAddress: true
  selector:
    matchLabels:
      workloadType: sts
      fixedIP: "true"
      app: fixedIPApp
```

2. 创建工作负载

```
  apiVersion: apps/v1
###### 必须使用有状态工作负载
kind: StatefulSet
metadata:
  name: foo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fixedIPApp
  serviceName: foo
  template:
    metadata:
      labels:
        workloadType: sts
        fixedIP: "true"
        app: fixedIPApp
    spec:
      containers:
      - image: nginx
        name: nginx
```

3. 验证IP分配结果

```
  ###### kubectl get pod {podName} -oyaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cce.baidubce.com/PodSubnetTopologySpread: example-subnet-topology
  labels:
    app: foo
  name: foo-0
  namespace: default
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - zoneF
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: nginx
    resources:
      limits:
        cce.baidubce.com/ip: "1"
      requests:
        cce.baidubce.com/ip: "1"
```

场景3：固定IP 固定 IP 策略的含义是给定一个IP列表，CCE从这个IP列表中给有状态负载的Pod分配IP。Pod删除后IP地址保留，Pod重建或者迁移后，IP地址仍然不变。 无论 Pod 是否删除，已经分配的IP地址不会被释放，直到工作负载被删除后，IP 地址才会被回收。 **适用场景：**

- 指定的子网必须是单个CCE集群使用，且该子网不再用于动态分配IP地址（子网会被CCE标记为专属子网）。
- 以IP地址为维度做安全策略，如以子网维度制定ACL。
- 仅适用于有状态工作负载（`apps/v1 StatefulSet`）创建的 Pod。

1. 创建 psts

```
  apiVersion: cce.baidubce.com/v2
kind: PodSubnetTopologySpread
metadata:
  name: example-subnet-topology
  namespace: default
spec:
  subnets:
    sbn-6mrkdcsyzpaw:
    # 非必须，固定 IP 的范围，不填表示使用子网默认 IP 范围
    - family: 4
      range:
      - start: 10.0.0.2
        end: 10.0.0.254
  strategy:
    releaseStrategy: Never
    type: Fixed
    enableReuseIPAddress: true
  # 选择要使用该子网拓扑分布的Pod
  selector:
    matchLabels:
      app: foo
```

2. 创建工作负载

```
  apiVersion: apps/v1
###### 必须使用有状态工作负载
kind: StatefulSet
metadata:
  name: foo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fixedIPApp
  serviceName: foo
  template:
    metadata:
      labels:
        workloadType: sts
        fixedIP: "true"
        app: fixedIPApp
    spec:
      containers:
      - image: nginx
        name: nginx
```

3. 验证IP分配结果

```
###### kubectl get pod {podName} -oyaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cce.baidubce.com/PodSubnetTopologySpread: example-subnet-topology
  labels:
    app: foo
  name: foo-0
  namespace: default
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - zoneF
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: nginx
    resources:
      limits:
        cce.baidubce.com/ip: "1"
      requests:
        cce.baidubce.com/ip: "1"
```

相关产品

- 私有网络VPC

DNS

DNS 原理概述

在 CCE 集群中中，由 CoreDNS 组件为集群中工作负载提供 DNS 域名解析服务。

CoreDNS 为 CNCF 托管项目，拥有丰富的插件集，可以为集群中负载提供集群内、集群外公网域名、私有域名等域名解析。

DNS 原理说明 CCE 集群中的 CoreDNS 工作负载，该负载默认有 3 个 CoreDNS pod，并会通过名为 kube-dns 的服务名暴露 DNS 服务。 示意如下：

```
###### kubectl -n kube-system get svc kube-dns
NAME       TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   172.16.0.10   <none>        53/UDP,53/TCP,9153/TCP   13d

###### kubectl -n kube-system get pod | grep coredns
coredns-5bcb94c99b-2g5fq                  1/1     Running   0          12d
coredns-5bcb94c99b-x4xxk                  1/1     Running   0          12d
coredns-5bcb94c99b-x5kjg                  1/1     Running   0          12d
```

集群中 Pod 内的 DNS 域名解析配置文件 /etc/resolv.conf 内容如下：

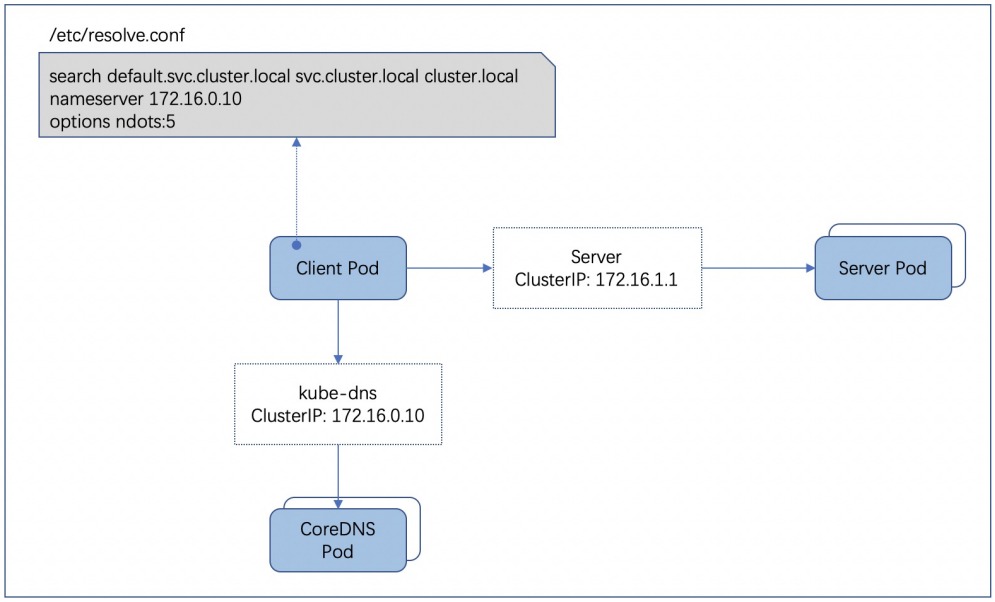
```
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver 172.16.0.10
options ndots:5
```

参数说明：

- search：域名的查找后缀规则规则
- nameserver：DNS 服务器的 IP 地址，这里为 kube-dns 的 ClusterIP
- options：定义域名解析配置文件选项；如果域名中点字符数量大于等于 ndots 值，则认为是完整域名，将被直接解析，如果小于 ndots 值，则追加 search 中指定的后缀依次查询

因此，pod 会通过 kube-dns 的 ClusterIP，如 172.16.0.10 请求 CoreDNS pod，拿到目标服务的 ClusterIP，然后对目标服务发起请求。

CCE 集群中 DNS 工作原理如下图所示：



通过 dnsPolicy 自定义 DNS 策略 在 CCE 集群中，DNS 策略可以逐个 Pod 来设定。目前支持以下特定 Pod 的 DNS 策略。 这些策略可以在 Pod Spec 中的 dnsPolicy 字段设置：

- "Default": Pod 从运行所在的节点继承名称解析配置，即使用云上 DNS 服务器进行域名解析服务
- "ClusterFirst": 使用 CoreDNS 进行域名解析服务，pod 内 /etc/resolv.conf 中 nameserver 将指向 kube-dns 服务的 ClusterIP
- "ClusterFirstWithHostNet": 对于以 hostNetwork 方式运行的 Pod，应将其 DNS 策略显式设置为 "ClusterFirstWithHostNet"。否则，以 hostNetwork 方式和 "ClusterFirst" 策略运行的 Pod 将会做出回退至 "Default" 策略的行为
- "None": 此设置允许 Pod 忽略 Kubernetes 环境中的 DNS 设置，Pod 会使用其 dnsConfig 字段所提供的 DNS 设置

说明："Default" 不是默认的 DNS 策略。如果未明确指定 dnsPolicy，则默认使用 "ClusterFirst"。

DNS 问题排查指南

本文档介绍 DNS 相关的常见问题和错误，以及对应的排查和解决方案。

DNS 问题排查流程 排查流程 当发生 DNS 解析失败时，可按照如下流程进行排查

1. 首先，我们可以根据解析失败报错，判断问题类型，判断方法参见 **客户端常见报错分类**
    1. 如果是因为网络不连通，参见 **问题排查思路** 的 **按解析异常的域名类型排查**
    2. 如果是因为域名解析失败，参见 **问题排查思路** 的 **按解析异常出现频次排查**
2. 如上述排查未解决问题，请按如下步骤排查
    1. 检查 Pod 配置的 DNSPolicy 字段，确认是否使用了 CoreDNS，具体参见 **Pod DNS 配置检查**
        1. 如果没有使用 CoreDNS，则默认会继承节点 DNS 配置，参见 **节点 DNS 问题排查**
        2. 如果使用了 CoreDNS，按如下步骤排查
            1. 排查 CoreDNS 负载运行情况，参见 **CoreDNS pod 运行状态排查**
            2. 排查 CoreDNS 日志，参见 **CoreDNS pod 日志排查**
    2. 如果使用的是 NodeLocal DNS，参见 **NodeLocal DNS 问题排查**
3. 如以上流程都未解决问题，可提交工单

客户端常见报错分类

分类	错误
网络连通性问题	network unreachable connection timeout connection reset by peer no route to host
域名解析失败	no such host could not resolve host name or service not known NXDOMAIN

注意：不同的 SKD 有不同的报错，上述报错关键字可能并不全面，可以结合报错信息进行判断。

DNS 问题排查思路 按解析异常的域名类型排查

异常类型	解决方案
集群内外域名都异常	**容器网络不通** **CoreDNS Pod 负载高** **节点 Conntrack 表满** **IPVS 缺陷导致解析异常**
仅集群外部域名异常	**集群外部域名解析异常**
仅 Headless 类型服务域名异常	**Headless 类型服务域名无法解析**

按解析异常出现频次排查

出现频次	解决方案
异常仅出现在业务高峰时期	**CoreDNS Pod 负载高** **节点 Conntrack 表满**
异常出现频次高	**IPVS 缺陷导致解析异常**
异常出现频次低	**A 记录和 AAAA 记录并发解析异常**
异常仅出现在节点扩缩容或CoreDNS缩容时	**IPVS 缺陷导致解析异常**

问题排查方法 Pod DNS 配置检查 查看业务 pod 的 dnsPolicy 字段，命令如下：

```
kubectl -n <ns> get pod <pod-name> -o yaml | grep dnsPolicy
```

该字段取值如下 ：

- "Default": Pod 从运行所在的节点继承名称解析配置，即使用云上 DNS 服务器进行域名解析服务
- "ClusterFirst": 使用 CoreDNS 进行域名解析服务，pod 内 /etc/resolv.conf 中 nameserver 将指向 kube-dns 服务的 ClusterIP
- "ClusterFirstWithHostNet": 对于以 hostNetwork 方式运行的 Pod，应将其 DNS 策略显式设置为 "ClusterFirstWithHostNet"。否则，以 hostNetwork 方式和 "ClusterFirst" 策略运行的 Pod 将会做出回退至 "Default" 策略的行为
- "None": 此设置允许 Pod 忽略 Kubernetes 环境中的 DNS 设置，Pod 会使用其 dnsConfig 字段所提供的 DNS 设置

之后可以进一步进入业务 pod，查看业务 pod 内 dns 配置文件是否符合预期。

首先进入容器：

```
kubectl -n <ns> exec -it <pod-name> -- bash
```

如 `bash` 命令不存在，可尝试 `sh`

然后查看 dns 配置文件：

```
cat /etc/resolv.conf
```

如果 dnsPolicy 字段值为 ClusterFirst 或 ClusterFirstWithHostNet，则 resolv.conf 中 nameserver 应为集群中 kube-dns 的 ClusterIP。

如果 dnsPolicy 字段值为 Default，则 resolv.conf 中 nameserver 应和节点上 /etc/resolv.conf 相同。

如果 dnsPolicy 字段值为 None，则 resolv.conf 中配置应和用户自己再 pod yaml 中 dnsConfig 字段配置相同。

连通性异常排查 首先，我们可以检查 pod 到 CoreDNS 的 ClusterIP 的连通性。

可以通过如下命令进入 pod：

```
kubectl -n <ns> exec -it <pod-name> -- bash
```

如 `bash` 命令不存在，可尝试 `sh`

如 pod 无法通过 bash 或者 sh 登录，可以先登录节点，然后在节点上执行如下命令拿到 container-id：

```
docker ps | grep <容器名>
```

然后执行如下命令拿到 pid：

```
docker inspect <container-id> | grep -i pid
```

最后通过如下命令进入 pod 网络命名空间：

```
nsenter -t <pid> -n bash
```

注意如容器运行时为 containerd，可将上述 `docker` 命令换为 `crictl`。

先通过如下命令拿到 coredns 的 ClusterIP：

```
kubectl -n kube-system get svc kube-dns
```

然后通过如下命令依次测试到 ClusteIP 和 pod-ip 的连通性：

```
telnet <ClusterIP> 53
telnet <pod-ip> 53
```

如果发现到 ClusterIP 无法连通，但到 CoreDNS 的 pod ip 可以联通，这说明集群的 Service 连通性存在问题，可以去排查 kube-proxy 等提供 Service 负载均衡相关功能的模块。

如果检测发现 pod 到 CoreDNS pod ip 无法连通，那先确认集群 pod 网络是否联通，如果 pod 网络存在问题，需要先解决该问题，常见的问题包括节点网络不通、安全组配置错误等。

如果集群 pod 网络没有问题，那说明可能是 CoreDNS pod 存在问题。 可以通过如下命令测试解析：

```
dig <domain> @<ClusterIP>
```

根据解析结果，参考本文其他部分进行相关排查。

CoreDNS pod 运行状态排查 首先确认 coredns pod 运行状态，可以执行如下命令：

```
kubectl -n kube-system get pod -o wide | grep coredns
```

预期所有 pod 都应该处于 Running 状态，有有非 Running 的，可以执行如下命令查看原因：

```
kubectl -n kube-system describe pod <pod-name>
```

也可以使用如下命令查看 coredns pod 资源使用情况，看是否资源超限：

```
kubectl -n kube-system top pod -l k8s-app=kube-dns
```

如过发现 CoreDNS pod 负载过高，可以增加 CoreDNS 的副本数。

如过 coredns pod 状态正常，则可以进一步排查 coredns pod 的日志。

CoreDNS pod 日志排查 命令如下：

```
kubectl -n kube-system logs <pod-name>
```

这里，因为集群中服务通过 ClusterIP 负载均衡到多个 pod，可以抽查 coredns pod 日志。

如果想准确将 dns 请求打到某一个 coredns pod，可以通过 dig 命令指定 dns 服务器来实现，命令如下:

```
dig baidu.com @<pod-ip>
```

之后再查看对应 pod-ip 的 coredns pod 日志
coredns 日志示例如下：

```
[INFO] 192.168.2.90:30639 - 8870 "A IN nfd-master.kube-system.svc.cluster.local. udp 69 false 1232" NOERROR qr,aa,rd 114 0.00010245s
```

其中的关键字 NOERROR 是表示解析成功返回码，常见的返回码如下：

- NOERROR：解析成功
- NXDOMAIN：域名不存在
- SERVFAIL：从上游 DNS 服务器解析错误等

此外，还需要关注日志中是否存在其他错误，一些常见的错误有：

- 无法连接 API Server，请检查 API Server 的状态
- K8S API 兼容性错误，一般是由于 CoreDNS 和 K8S 版本不兼容，通常需要升级 CoreDNS 或 K8S 版本

域名解析失败排查 场景1：集群内 svc 解析成功，公网域名解析失败。这种情况，一般在 coredns pod 日志中会出现 NXDOMAIN 或 SERVFAIL 等返回码，此时可以联系云上 DNS 团队处理。

场景2：私有域名解析失败。需要确认私有域名是否已注册到云上 DNS 中，如果希望 CoreDNS 完成私有域名解析，需要在 CoreDNS 配置文件中添加私有域名解析服务器相关的配置。

场景3：Headless 类型 svc 无法解析。Headless 类型服务，解析结果会直接返回所有 pod ip，请检查服务对应的 pod 是否处于 Running 状态。

节点 DNS 问题排查 如用户负载 dnsPolicy 为 default 或使用 hostNetwork，则会使用节点 DNS 配置。
可在节点上使用 dig 命令进行复现和排查，示例如下：

```
dig baidu.com
```

检查节点内核日志，执行如下命令：

```
dmesg
```

检查其中是否有网络相关报错，如：

- queue failue
- conntract full

如节点内核日志未发现异常，因节点上 /etc/resolv.conf 中默认配置使用公有云 DNS 服务器，可进一步联系云上 DNS 团队进行处理。

NodeLocal DNS 问题排查 请先阅读 CCE Node Local DNS 说明 ，了解 NodeLocal DNS 的工作原理。
然后根据文档先检查 NodeLocal DNS 相关配置均已生效，其它各环节问题排查，可参考本文档。

常见 DNS 问题和解决方案 容器网络不通 问题现象：
业务 pod DNS 解析持续失败

问题原因：
业务 pod 到 CoreDNS pod 容器网络不通

解决方案：
排查并确保容器网络的连通性正常，可参见问题排查方法中的 **连通性异常排查**

集群外部域名解析异常 问题现象：
业务 pod 可以正常解析集群内部域名，但无法解析某些集群外部域名

问题原因：
上游服务器域名解析返回异常

解决方案：
查看 CoreDNS pod 的请求日志，确定错误码，日志示例如下：

```
[INFO] 192.168.2.90:30639 - 8870 "A x.y.com. udp 69 false 1232" NOERROR qr,aa,rd 114 0.00010245s
```

如返回码不为 NOERROR，说明上游服务器返回错误，常见错误如下：

- NXDOMAIN：域名不存在
- SERVFAIL：一般是上游服务器故障或无法连接到上游服务器

如确定是上游服务器问题，可提交工单处理

Headless 类型服务域名无法解析 问题现象： Headless 类型 svc 无法解析

问题原因：

Headless 类型服务，解析结果会直接返回所有 pod ip，如业务 pod 是没有处于 Running 状态，则无法解析

解决方案：

请检查并确保服务对应的 pod 处于 Running 状态

CoreDNS Pod 负载高 问题现象： 业务 Pod DNS 请求延迟大、概率失败或持续失败

问题原因：

CoreDNS pod 负载高，处理能力不足，导致请求延迟增加或失败

解决方案：

增加 CoreDNS 副本数，或调大 CoreDNS 资源

节点 Conntrack 表满 问题现象： 业务 Pod DNS 在请求高峰时，出现请求概率失败或持续失败 再节点上运行 dmesg -T，在对应时段的日志中，发现有 conntrack full 关键字的错误

问题原因：

业务高峰时，内核 Conntrack 表打满，导致无法进行新的 TCP 或者 UDP 请求

解决方案：

调大节点上的内核 Conntrack 表限制，可提交工单处理

A 记录和 AAAA 记录并发解析异常 问题现象： 业务 Pod 解析域名概率性失败

问题原因：

并发 A 和 AAAA 的 DNS 请求触发 Linux 内核 Conntrack 模块缺陷，导致 UDP 报文丢失

解决方案：

- 如果容器镜像是以Alpine制作的，建议更换基础镜像
- 考虑采用 NodeLocal DNS 缓存方案，提升 DNS 解析性能，降低 CoreDNS 负载
- CentOS、Ubuntu 等基础镜像，可以通过 options timeout:2 attempts:3 rotate single-request-reopen 等参数优化

IPVS 缺陷导致解析异常 问题现象： 集群节点扩缩容、CoreDNS 缩容时，业务出现概率性解析失败，通常时长在五分钟左右

问题原因：

若集群的 kube-proxy 负载均衡模式为 IPVS，在 CentOS 等内核版本小于 4.19 的节点上，摘除 IPVS UDP 类型后端后，一段时间内若新发起的 UDP 报文源端口冲突，该报文会被丢弃

解决方案：

可以考虑采用 NodeLocal DNS 方案，由于 NodeLocal DNS pod 和 CoreDNS pod 间使用 TCP，可以容忍该 IPVS 缺陷导致的丢包问题

CoreDNS 组件手动升级指南

CoreDNS 和 K8S 版本的兼容性

CoreDNS 镜像版本	支持的 K8S 版本
1.7.1	1.18
1.9.4-baidu	1.20，1.24，1.26，1.28

⚠ 注意：升级 CoreDNS 会导致全集群断网 5 分钟，请做好评估后再进行升级。

升级过程

1. 检查集群当前 CoreDNS 镜像版本，命令如下：
   `kubectl -n kube-system get deployment coredns -o yaml | grep "image:"`
   示例输出如下：
   `image: registry.baidubce.com/cce-plugin-pro/coredns:1.7.1`
2. 修改可能存在的插件名称变更。
   如果 CoreDNS 镜像版本很低（低于 1.6.2，forward 插件旧名称为 proxy），通过如下命令：
   `kubectl -n kube-system get cm coredns -o yaml`
   看到图中所示插件名称不为 forward，而是 proxy：

```
press kubectl -n kube-system get cm coredns -o yaml
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health
        log
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
            ttl 30
        }
        prometheus :9153
        forward . /etc/resolv.conf
        cache 30
        loop
        reload
        loadbalance
    }
```

则通过如下命令将其修改为 forward

`kubectl -n kube-system get cm coredns -o yaml`

3. 增加 endpointslices 权限

   通过如下命令：

   `kubectl -n kube-system get clusterrole system:coredns -o yaml`

   如果看结果中缺少 endpointslices 权限，则通过：

   `kubectl -n kube-system edit clusterrole system:coredns`

   增加权限，修改后的 clusterrole 内容如下：

```
 apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 labels:
   kubernetes.io/bootstrapping: rbac-defaults
 name: system:coredns
rules:
 - apiGroups:
   - ""
   resources:
   - endpoints
   - services
   - pods
   - namespaces
   verbs:
   - list
   - watch
 - apiGroups:
   - discovery.k8s.io
   resources:
   - endpointslices
   verbs:
   - list
   - watch
```

4. 修改镜像版本 将 coredns 镜像版本升级到希望的版本，如 1.9.4-baidu，命令如下：

   `kubectl -n kube-system edit deployment coredns` 修改后的示例如下：

```
              values:
              - kube-dns
          topologyKey: kubernetes.io/hostname
        weight: 100
containers:
- args:
  - -conf
  - /etc/coredns/Corefile
  image: registry.baidubce.com/cce-plugin-pro/coredns:1.9.4-baidu
  imagePullPolicy: IfNotPresent
  livenessProbe:
    failureThreshold: 5
    httpGet:
      path: /health
      port: 8080
      scheme: HTTP
    initialDelaySeconds: 60
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 5
  name: coredns
```

5. 检查升级结果 通过命令：

   `kubectl -n kube-system get pod -o wide | grep coredns`

   观察是否所有 pod 都启动成功 通过命令：

   `kubectl -n kube-system logs <coredns-pod-name> | less -N`

   检查 coredns pod 是否有错误日志

   如上述流程有异常，可发工单处理；如都正常，则升级完成。

配置管理

Configmap管理

配置项（ConfigMap）

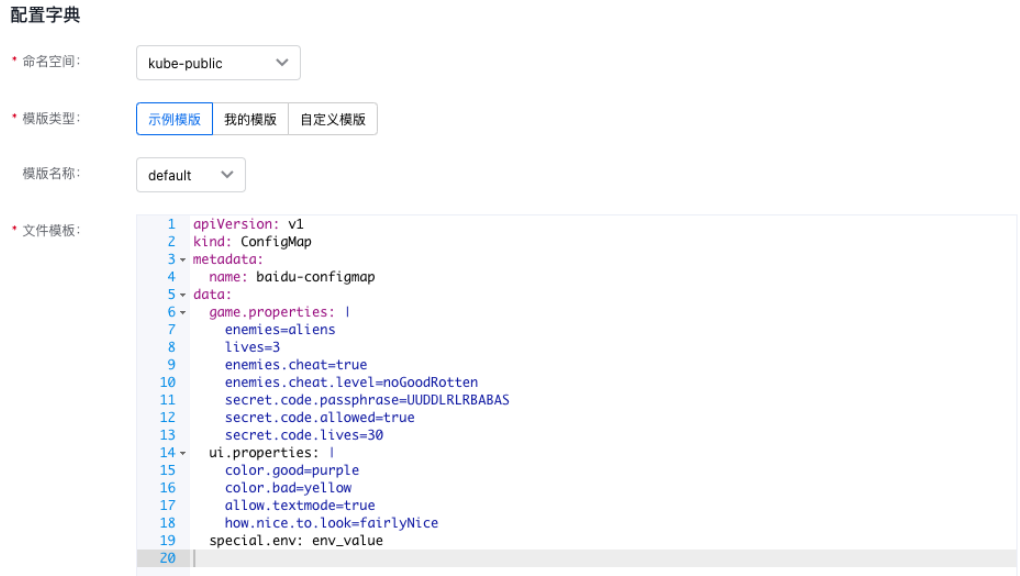
ConfigMap 是 K8S 的一种 API 对象，用来将非机密性的数据保存到健值对中，用户在 CCE 可以通过页面或 kubectl 方式管理 ConfigMap。

通过页面管理 ConfigMap

创建 ConfigMap

进入“产品服务>容器引擎CCE->集群详情”，点击“配置管理>配置项”，进入配置项列表页面：

点击“新建配置项”，用户可以通过 YAML 的形式创建新的配置项，选择命名空间，填写 YAML 文件，点击“确认”提交：



查看 ConfigMap

点击列表页任何一行的配置项名称，在右边栏弹出配置项详情页：

删除配置项

在配置项列表页面，勾选要删除的配置项，点击“删除”按钮，确认信息后，点击“确认”：

修改配置项

在配置项列表页面，点击“修改”按钮，完成修改后，点击“更新”提交，成功提交后返回列表页面：

通过 kubectl 管理 ConfigMap

示例：ConfigMap

```
 apiVersion: v1
kind: ConfigMap
metadata:
 name: app-config
data:
 config.json: |
  {
     "Listener": 9443,
     "MetricsAddress": "0",
     "EnableLeaderElection": true,
     "EnableClusterController": true,
     "EnableInstanceController": true
  }
```

Pod 将 ConfigMap 作为配置文件挂载：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: app-nginx
 labels:
   app: nginx
spec:
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
       - name: nginx
         image: registry.baidubce.com/cce/nginx-alpine-go:latest
         volumeMounts:
           - mountPath: /app/conf
             name: app-config
     volumes:
       - configMap:
           defaultMode: 420
           name: app-config
         name: app-config
```

上述 configmap app-config 对应的文件，会存在 Pod 的 /app/conf 目录下:

```
 ##### exec 进入容器:

$ cat /app/conf/config.json
{
   "Listener": 9443,
   "MetricsAddress": "0",
   "EnableLeaderElection": true,
   "EnableClusterController": true,
   "EnableInstanceController": true
}
```

更新 ConfigMap 后，Pod 中的文件会在 1min 内自动更新。

Secret管理

保密字典（Secret）

Secret是K8S提供一种API资源，用来保存敏感信息，例如密码、OAuth令牌和SSH密钥等，用户可在CCE上通过控制台及 kubectl方式管理Secret资

源。

前提条件

- 已创建CCE集群，具体操作请参见创建CCE集群。
- 已创建命名空间，具体操作请参见命名空间基本操作。

通过控制台管理Secret

创建 Secret

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**配置管理>保密字典**。
2. 在保密字典列表页面单击**创建保密字典**系统弹出保密字典弹框，在弹框中进行相关参数配置。

配置项	是否必填	说明
保密字典名称	必填	请输入自定义名称。
命名空间	必填	指定创建资源所在命名空间。
类型	必填	提供Opaque、TLS证书和私有镜像拉取访问凭证类型，请根据实际需求进行选择。 **Opaque:**适用于保存密钥证书和配置文件，Value 将以 base64 格式编码。 **TLS 证书：**适用于保存 TLS 证书和私钥。 **私有镜像拉取访问凭证：**适用于保存私有 Docker Registry 的认证信息。
内容	必填	当Secret类型为**Opaque**时，需要设置变量名和变量值，支持添加多个。
Cert	必填	当Secret类型为**TLS证书**时，需要添加证书或倒入证书文件。
Key	必填	当Secret类型为**TLS证书**时，需要添加私钥或倒入私钥文件。
镜像仓库地址	必填	当Secret类型为**私有镜像拉取访问凭证**时，需要输入镜像仓库访问地址。
用户名	必填	当Secret类型为**私有镜像拉取访问凭证**时，需要输入登录镜像仓库的用户名。
密码	必填	当Secret类型为**私有镜像拉取访问凭证**时，需要输入登录镜像仓库的密码。
确认密码	必填	当Secret类型为**私有镜像拉取访问凭证**时，需要再次输入登录镜像仓库的密码。

3. 点击**确认**按钮，即可完成创建。

查询Secret

单击列表页中目标保密字典名称，在右边栏弹出保密字典详情页，详情页内容包括：

- 基本信息：名称、命名空间、类型、创建时间、数据信息；
- 数据信息：该处有一个“隐藏”按钮，可以显示或者隐藏数据信息的详情。

编辑Secret

在保密字典列表页面，选择要编辑的保密字典，单击“编辑”按钮，完成修改后，单击**确认**按钮即可完成编辑。

删除Secret

在保密字典列表页面，选择要删除的保密字典，单击**删除**按钮，在弹框中单击**确定**按钮即可完成删除。

通过kubectl管理Secret

Secret YAML 示例:

```
  apiVersion: v1
kind: Secret
metadata:
  name: baidu-secret
data:
  username: dXNlcm5hbWU=  # echo -n "username" | base64
  password: cGFzc3dvcmQ=  # echo -n "password" | base64
```

Pod 中使用 Secret 作为文件:

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-nginx
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: hub.baidubce.com/cce/nginx-alpine-go:latest
          volumeMounts:
            - name: foo
              mountPath: "/etc/foo"
              readOnly: true
      volumes:
        - name: foo
          secret:
            secretName: baidu-secret
```

上述 secret baidu-secret 对应的文件，会存在 Pod 的 /etc/foo 下:

```
  ##### exec -it 进入容器

##### ls /etc/foo/
password  username
```

存储管理

概述

容器的特性决定了容器本身是非持久化的，容器被删除后容器内的数据也一并被删除，CCE 基于百度云存储产品，提供了容器数据共享和持久化解决方案，适用于大数据分析、媒体、游戏等很多场景。

PV（PersistentVolume）和 PVC（PersistentVolumeClaim）是 K8S 提供的用于抽象存储细节的 API 资源。在集群内创建 PV 和 PVC 资源，用户可以直接将百度云存储作为存储卷挂载到容器中，而无需关注底层的实现细节，从而更加便捷地为容器集群提供持久化存储方案。

- PV：持久卷，集群内的存储资源。PV 独立于 Pod 的生命周期，可根据不同的 StorageClass 类型创建不同类型的 PV。
- PVC：持久卷声明，集群内的存储请求，请求存储资源的声明。
- Volume：数据卷，是Pod与外部存储设备进行数据传递的通道，也是Pod内部容器间、Pod与Pod间、Pod与外部环境进行数据共享的方式。

> **说明**
>
> 数据卷（Volume）生命周期和Pod一致，即Pod被删除的时候，数据卷（Volume）也一起被删除（Volume中的数据是否丢失取决于Volume的具体类型）。

CCE 容器引擎支持多种类型的存储，并且支持 K8S 的静态存储卷与动态存储卷。

存储类型

CCE 基于百度云存储产品 CDS、BOS、CFS、PFS、RapidFS 对不同存储类型提供完整支持：

存储类型	说明
本地存储	包含 HostPath，EmptyDir，ConfigMap，Secret 等类型
云盘 CDS	通过 PV/PVC 使用，包含静态和动态两种方式
文件存储 CFS	通过 PV/PVC 使用，包含静态和动态两种方式
对象存储 BOS	通过 PV/PVC 使用，只可使用静态存储卷
并行文件存储 PFS	通过 PV/PVC 使用，包含静态和动态两种方式
数据湖存储加速工具 RapidFS	通过 PV/PVC 使用，包含静态和动态两种方式

> **说明**
>
> - 建议使用云存储产品，否则当节点异常无法恢复时，本地存储的数据同样不能恢复。
> - 并行文件存储 PFS 支持不同实例类型，CCE 针对不同类型实例的使用方式不同，更多信息请分别参见 使用并行文件存储PFS 和 使用并行文件存储PFS L2。

存储选型

有数据持久化需求的服务，建议直接使用云存储（CDS、CFS、BOS、PFS、RapidFS），本地存储当节点异常无法恢复时，本地磁盘中的数据也可能丢失，下面简单介绍不同云存储的使用场景：

- 云磁盘 CDS：**Pod 和 CDS 是一对一的关系**，不支持多个 Pod 挂载一块 CDS，CDS 提供数据块级别的持久性存储，通常用作需要频繁更新、细粒度更新的数据（如文件系统、数据库等）的主存储设备，具有高可用、高可靠和高性能的特点；
- 文件存储 CFS：**Pod 和 CFS 是多对一的关系，支持多读多写**，CFS 提供了标准的 NFS 文件系统访问协议，为多个 Pod 提供文件共享能力，支持弹性容量和性能的扩展，是一种高可用、高可靠的分布式文件系统，适合于大数据分析、媒体处理、内容管理等场景；
- 对象存储 BOS：**Pod 和 BOS 是多对一的关系，支持一写多读**，BOS 适用于数据分发、数据备份、大数据分析、音视频文件处理等各类场景。
- 并行文件存储 PFS：**Pod 和 PFS 是多对一的关系，支持多读多写**，PFS 提供了完全托管、简单可扩展的并行文件存储系统，针对高性能计算场景提供亚毫秒级的访问能力和高IOPS的数据读写请求能力。
- 数据湖存储加速工具 RapidFS：**Pod 和 RapidFS 是多对一的关系，支持多读多写**，RapidFS 提供了高可靠、高可用、弹性的数据湖存储加速工具，加速大数据、人工智能、机器学习等业务访问存储的性能。

使用本地存储

emptyDir

emptyDir 类型的 Volume 在 Pod 分配到 Node 上时被创建，Kubernetes 会在 Node 上自动分配一个目录，因此无需指定宿主机 Node 上对应的目录文件，这个目录的初始内容为空。

> **说明**
>
> 当 Pod 从 Node 上移除时，emptyDir 中的数据会被永久删除。

示例

创建 Pod，该 Pod 指定数据卷为 emptyDir：

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-ed
spec:
  containers:
  - image: registry.baidubce.com/cce/nginx-alpine-go:latest
    name: test-container
    volumeMounts:
    - mountPath: /test
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

查看创建的 Pod 详情：

```
$ kubectl describe pod test-ed
```

进入 Pod 内部查看挂载效果：

```
$ kubectl exec -it test-ed -c test-container -- /bin/sh
```

hostPath

HostPath 类型则是映射 Node 文件系统中的文件或者目录到 Pod 里。在使用 HostPath 类型的存储卷时，也可以设置 type 字段，支持的类型有文件、目录、File、Socket、CharDevice 和 BlockDevice。

示例

创建 Pod，该 Pod 数据卷指定 HostPath：

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-hp
spec:
  containers:
  - image: registry.baidubce.com/cce/nginx-alpine-go:latest
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      #directory location on host
      path: /data
```

Local Volume

Local Volume 允许将 Node 本地的磁盘、分区或者目录作为持久化存储使用。Local Volume 适合应用于数据缓存，应用可以就近访问数据，快速处理，同时还适用于分布式存储系统，如分布式数据库 Cassandra ，分布式文件系统 ceph/gluster。

示例

创建storage class

```
  kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-volume
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

> **说明**
>
> 其中WaitForFirstConsumer表示PV不要立即绑定PVC，而是直到有Pod需要用PVC的时候才绑定。

静态创建 PV

静态创建一个 5GiB 的 PV；该 PV 使用 node 的 /data/volume 目录（需要手动创建目录）：

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: example-local-pv
spec:
 capacity:
   storage: 5Gi
 accessModes:
 - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
 storageClassName: local-volume
 local:
   path: /data/volume
 nodeAffinity:
   required:
     nodeSelectorTerms:
     - matchExpressions:
       - key: kubernetes.io/hostname
         operator: In
         values:
         - HOSTNAME    # 替换对应的节点名称
```

使用 local volume PV

创建 PVC 绑定 local volume 的 PVC，并创建 Pod 使用该 PVC：

1. 创建 PVC

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mypvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 5Gi
 storageClassName: local-volume
```

2. 创建 Pod

```
 kind: Pod
apiVersion: v1
metadata:
 name: mypod
spec:
 containers:
 - name: myfrontend
   image: registry.baidubce.com/cce/nginx-alpine-go:latest
   volumeMounts:
   - mountPath: "/usr/share/nginx/html"
     name: mypd
 volumes:
 - name: mypd
   persistentVolumeClaim:
     claimName: mypvc
```

3. 查看创建的 PV、PVC、Pod。

```
 $ kubectl get pv
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS     REASON   AGE
example-local-pv    5Gi        RWO            Retain           Bound    default/mypvc           local-volume              5s
$ kubectl get pvc
NAME                STATUS   VOLUME                  CAPACITY   ACCESS MODES   STORAGECLASS     AGE
mypvc               Bound    example-local-pv        5Gi        RWO            local-volume     4m20s
$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mypod                       1/1     Running   0          4m28s
```

4. 向本地的/data/volume写入内容，使用curl查看是否写入成功

```
 $ echo "hello world" > /data/volume/index.html
$ curl 172.19.4.63    #这里替换自己的pod的ip
hello world
```

> **注意**
>
> 删除时注意首先删除创建的pod，随后删除pvc，pv。

使用云盘CDS

概述

容器引擎 CCE 支持通过创建 PV/PVC，并为工作负载挂载数据卷方式使用百度智能云磁盘CDS。本文将介绍如何在集群中动态和静态挂载云磁盘。

使用限制

- 集群 Kubernetes 版本需大于或等于 1.16。
- 挂载 CDS 的 Pod 只能在支持挂载云磁盘的节点上启动。
- 每个节点挂载的 CDS 数量由 CCE CSI CDS Plugin 组件配置参数及云服务器自身同时决定，更多信息请见创建服务器实例。
- 当前云盘只支持按量付费的方式进行挂载，不支持包年包月的付费方式创建。

前提条件

- 集群已安装块存储组件，更多信息请见 CCE CSI CDS Plugin 说明。

操作步骤

通过 PV/PVC 方式使用 CDS，具体分为两种方式：

方式	说明	注意事项
静态挂载	需要用户提前在百度智能云中创建好 CDS（操作方法参考 CDS 文档），然后通过 CDS 的 volume id 在集群中创建 PV 和 PVC 资源。	1. 用户提供的 CDS 磁盘**未被分区**且处于**未挂载状态**。 2. 确保填入正确的**磁盘 ID**。
动态挂载	用户在集群中声明 PVC 时，自动创建出按量付费 CDS 磁盘并且动态关联至 PV。	1. 动态挂载时将会自动创建按量付费 CDS 磁盘并且计费，详情参见计费说明。 2. 动态创建的 CDS 将会随着用户删除 PVC 时被联动删除（需要将对应 `storageClass` 的 `reclaimPolicy` 参数设置为 `delete`）。 3. 如果用户在没有删除 PVC 的情况下直接删除了 CCE 集群，则需要手动对这些 CDS 进行处理。

动态挂载块存储

方式一：通过 kubectl 命令行操作

1. 创建存储类 StorageClass

- 集群管理员可使用 StorageClass 为集群定义不同的存储类型。您可通过 StorageClass 配合 PVC 动态创建需要的存储资源。

- 本文介绍通过 Kubectl 方式创建云盘 CDS 类型的 StorageClass，自定义块存储使用所需的模板。

```
 apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: hp1                #StorageClass名称
provisioner: csi-cdsplugin
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
parameters:
 paymentTiming: "Postpaid"  #付费方式，Postpaid 表示按量付费，当前只支持按量付费
 storageType: "hp1"        #磁盘类型，支持cloud_hp1, hp1, hdd
 recycle: "on"            #云磁盘的删除回收策略，支持 on、off（on 表示删除时将云磁盘放入回收站，off 表示删除时云磁盘不放入回收站），仅当 reclaimPolicy 为 Delete 生效，默认值为 on
reclaimPolicy: Delete       #云磁盘的删除策略，支持 Delete、Retain（Delete 表示删除 PVC 时，PV 和 云磁盘会一起删除，删除云磁盘默认放入回收站；Retain 表示删除 PVC 时，PV和云磁盘不会被删除，需要您手动删除），默认为 Delete
```

> **说明：**
>
> - storageType 用于设置动态创建的 CDS 盘的类型，支持：cloud_hp1 （通用型 SSD）、hp1 （高性能云磁盘）、hdd（通用型 HDD），更多信息参见 创建CDS云盘。
> - 如果集群中存在使用动态方式挂载 CDS 的 Pod，且对应 StorageClass 中 reclaimPolicy 设置为 Delete，这部分 CDS 在集群删除集群时默认会保留，如果需要一同删除请在删除集群前手动停止相关 Pod 并删除对应 PVC。

2. 创建持久卷声明 PVC

注意 storageClassName 需要指定为上述部署 storageClass 时填写的 storageClass 名称。

```
 kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: csi-pvc-cds
spec:
 accessModes:
   - ReadWriteOnce
 storageClassName: hp1
 resources:
   requests:
     storage: 5Gi    # 指定PVC存储空间大小
```

> **说明：** 不同的磁盘类型，存储大小不同，如 hp1：5～32765GB，cloud_hp1：50～32765GB，hdd：5～32765GB

3. 检查 PVC 状态为 Bound

```
 $ kubectl get pvc csi-pvc-cds
NAME          STATUS   VOLUME              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-cds   Bound    pvc-1ab36e4d1d2711e9   50Gi       RWX            hp1        4s
```

> **说明：**
>
> 如果指定了 volumeBindingMode 为 `WaitForFirstConsumer`，则 PVC 此时处于 Pending 状态，会等待第一个挂载该 PVC 的 Pod 被创建后才进入 PV 创建和绑定流程。

4. 在 Pod 中挂载 PVC

Pod 和 PVC 需要在同一个 namespace 下。

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: cds-pvc
        mountPath: "/cds-volume"
  volumes:
    - name: cds-pvc
      persistentVolumeClaim:
        claimName: csi-pvc-cds
```

Pod 创建后，可以读写容器内的 /cds-volume 路径来访问相应的 cds 存储上的内容。

方式二：通过控制台操作

1. 登陆 CCE 控制台，点击集群名称进入集群详情页。
2. 创建存储类 StorageClass。

a. 在左侧导航栏中选择**存储配置 > 存储类**，进入存储类列表页面。

b. 在存储类列表上方点击**新建存储类**。在**文件模版**中自定义相关配置。storageType 用于设置动态创建的 CDS 盘的类型，支持：cloud_hp1 （通用型 SSD）、hp1 （高性能云磁盘）、hdd（通用型 HDD）。

b. 点击**确定**，将为您创建存储类。

3. 创建持久卷声明 PVC。

a. 在左侧导航栏选择**存储配置 > 持久卷声明**，进入持久卷声明列表。

b. 点击持久卷声明列表上方**新建持久卷声明**按钮，选择**表单创建**或 **Yaml 创建**。

c. 若选择表单创建则需填写相关参数，其中三种访问模式详情参见存储管理概述，点击**确定**后在**确认配置**弹窗中输入 `storageClassName` 的值，即存储类的名称。确认配置无误后点击**确定**。

d. 创建 PVC 后，在**持久卷声明列表 > 持久卷列**可以看见相应的 PV 自动创建，PVC 状态列显示为 Bound，即 PVC 已经与新创建的 PV 绑定。

4. 创建应用并挂载 PVC，在 Pod spec 内指定相应的 PVC 名称，且 Pod 和 PVC 需要在同一个 namespace 下。Pod 创建后，可以读写容器内的 /cds-volume 路径来访问相应的 CDS 存储上的内容。

静态挂载块存储

方式一：通过 kubectl 命令行操作

1. 创建持久卷 PV

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-cds
 namespace: "default"
spec:
 accessModes:
 - ReadWriteOnce
 capacity:
   storage: 5Gi
 csi:
   driver: "csi-cdsplugin"
   volumeHandle: "v-xxxx"    # 指定挂载的磁盘，格式为磁盘ID
   fsType: "ext4"
 nodeAffinity:
   required:
     nodeSelectorTerms:
     - matchExpressions:
       - key: failure-domain.beta.kubernetes.io/zone
         operator: In
         values:
         - zoneA
 persistentVolumeReclaimPolicy: Retain
```

说明：

- fsType 不填写时默认值为 ext4。Filesystem 模式下，若 fsType 与 cds 中实际文件系统类型不匹配，将导致挂载失败；若 cds 中尚无文件系统，则挂载时自动格式化 cds 为 fsType 指定的文件系统。
- nodeAffinity 中需要指定 cds 所在可用区，保证挂载该 cds 的 pod 只会被调度到与 cds 磁盘同可用区的 node 上。

2. 创建持久卷声明 PVC

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: csi-cds-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 5Gi
```

3. 检查 PVC 状态为 Bound，并且与对应的 PV 绑定

```
 $ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM                STORAGECLASS   REASON    AGE
pv-cds    5Gi        RWO            Retain           Bound     default/csi-cds-pvc                 4s
$ kubectl get pvc
NAME          STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-cds-pvc   Bound     pv-cds    5Gi        RWO                           36s
```

4. 在 Pod 中挂载 PVC

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-server-deployment
 labels:
  test: ws
spec:
 replicas: 1
 selector:
  matchLabels:
   se: ws
 template:
  metadata:
   labels:
    se: ws
   name: web-server
  spec:
   containers:
    - name: web-server
      image: nginx
      volumeMounts:
        - mountPath: /var/lib/www/html
          name: csi-cds-pvc
   volumes:
    - name: csi-cds-pvc
      persistentVolumeClaim:
        claimName: csi-cds-pvc
```

Pod 创建后，可以读写容器内的 /var/lib/www/html 路径来访问相应的 CDS 存储上的内容。

由于创建 PV 和 PVC 时只支持 accessModes 为 ReadWriteOnce，该 PVC 可以被同一节点上所有 Pod 挂载读写。

方式二：通过控制台操作

1. 登陆 CCE 控制台，单击集群名称进入集群详情页。
2. 创建持久卷 PV。

a. 在左侧导航栏选择**存储配置 > 持久卷**，进入持久卷列表。

b. 点击持久卷列表上方**新建持久卷**按钮，选择表单创建或 Yaml 创建。

c. 若选择表单创建则需填写相关参数，根据需求设置存储容量，存储类型选择**云盘 CDS**，并输入云盘 ID。点击**确定**在二次弹窗中确认配置后点击**确定**创建 PV。云盘 ID 可前往 CDS 控制台 查看。

3. 创建可以和持久卷 PV 绑定的持久卷声明 PVC。

a. 在左侧导航栏选择**存储配置 > 持久卷声明**，进入持久卷声明列表。

b. 点击持久卷列表上方**新建持久卷声明**按钮，选择表单创建或 Yaml 创建。

c. 根据之前创建的 PV 配置 PVC 的存储容量、访问模式、存储类（可选），点击**确定**创建 PVC。会查找现有的 PV 资源，寻找与 PVC 请求相匹配的 PV。

d. 绑定后，可以分别在持久卷列表和持久卷声明列表中看见 PV 和 PVC 的状态列变为 Bound。

4. 创建应用并挂载 PVC，在 Pod spec 内指定相应的 PVC 名称即可。

应用场景

在 StatefuleSet 里通过 claimTemple 批量使用 PVC

1. 创建 2 副本 StatefuleSet 并指定 volumeClaimTemplates，需要提前创建好相应 StorageClass：

```
 apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 serviceName: "nginx"
 replicas: 2
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: nginx
       ports:
       - containerPort: 80
         name: web
       volumeMounts:
       - name: www
         mountPath: /usr/share/nginx/html
 volumeClaimTemplates:
- metadata:
     name: www
   spec:
     accessModes: [ "ReadWriteOnce" ]
     resources:
       requests:
         storage: 5Gi
     storageClassName: hp1
```

2. 查看 Pod 以及自动创建的 PVC。

```
 $ kubectl get pod
NAME        READY     STATUS            RESTARTS   AGE
web-0       1/1       Running           0          3m
web-1       1/1       Running           0          2m
```

```
 $ kubectl get pvc
NAME          STATUS    VOLUME              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0     Bound     pvc-a1e885701d2f11e9   5Gi        RWO           hp1            6m
www-web-1     Bound     pvc-c91edb891d2f11e9   5Gi        RWO           hp1            5m
```

多可用区集群挂载 CDS

目前 CDS 不支持跨可用区，如果 CCE 集群中有多个可用区的机器，动态挂载的方式可以通过设置 StorageClass 的 Volume Binding Mode 为 WaitForFirstConsumer 来自动根据 Pod 被调度节点所处的可用区创建 CDS 磁盘，或者通过设置 StorageClass 中 allowedTopologies 字段或 Pod 的节点亲和属性来指定可用区；使用已有 CDS 时，须指定 PV 的 nodeAffinity 参数实现可用区亲和性。

CCE 集群默认会为集群节点打上可用区标签：

```
 $ kubectl get nodes --show-labels
NAME           STATUS    ROLES     AGE       VERSION   LABELS
192.168.80.15   Ready     <none>    13d       v1.8.12   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=BCC,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=bj,failure-domain.beta.kubernetes.io/zone=zoneC,kubernetes.io/hostname=192.168.80.15
```

其中 failure-domain.beta.kubernetes.io/zone=zoneC 标签说明该集群节点处于 C 可用区。

方式一：设置 StorageClass 的 Volume Binding Mode

通过指定 volumeBindingMode 为 WaitForFirstConsumer 来解决此问题。指定绑定模式为 WaitForFirstConsumer 时，先进行 Pod 的调度，随后根据调度结果对应的可用区创建 CDS。

使用示例

1. 创建 StorageClass 指定 `volumeBindingMode` 为 `WaitForFirstConsumer`。

```
 apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: csi-cds
provisioner: csi-cdsplugin
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
parameters:
 paymentTiming: "Postpaid"
 storageType: "hp1"
reclaimPolicy: Delete
```

2. 在 StatefulSet 中通过 claimTemple 使用 PVC：

```
 apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 serviceName: "nginx"
 replicas: 1
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: nginx
       ports:
       - containerPort: 80
         name: web
       volumeMounts:
       - name: www
         mountPath: /usr/share/nginx/html
 volumeClaimTemplates:
 - metadata:
     name: www
   spec:
     accessModes: [ "ReadWriteOnce" ]
     resources:
       requests:
         storage: 5Gi
     storageClassName: csi-cds
```

方式二：通过 StorageClass 的 allowedTopologies 字段强制指定 zone

1. 创建 StorageClass 并配置 `allowedTopologies` 为 参数。

```
  apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hp1-zonec
provisioner: csi-cdsplugin
parameters:
  paymentTiming: "Postpaid"
  storageType: "hp1"
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
- matchLabelExpressions:
  - key: failure-domain.beta.kubernetes.io/zone
    values:
    - zoneC
```

2. 在 StatefulSet 中通过 claimTemple 使用 PVC：

```
  apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-multi-zone
spec:
  serviceName: "nginx"
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 40Gi
      storageClassName: hp1-zonec
```

扩容 CDS PV

扩容 CDS PV 需要集群 Kubernetes 版本为 1.16 及以上，同时只针对动态创建的 CDS PV 生效。

1. 创建 storageClass 并添加 `allowVolumeExpansion: true` 配置

```
  apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hp1
allowVolumeExpansion: true    #允许磁盘扩容
...
```

2. 修改与 PV 绑定的 PVC 中 `spec.resources.requests` 中的磁盘容量为目标容量，触发扩容

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-pvc-cds
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: hp1
  resources:
    requests:
      storage: 8Gi #修改为目标磁盘大小
```

3. 默认情况下 CDS CSI 插件仅支持离线扩容，即需要在 PVC 未被 Pod 挂载时才会执行实际扩容动作。完成 PVC 容量 request 修改后，如果此时该 PVC 未被任何 Pod 挂载，则会触发 CDS 磁盘扩容操作，否则会等待挂载该 PVC 的 Pod 停止后才开始扩容。具体扩容进度可以通过 `kubectl describe pvc <pvcName>` 查看对应的 Events 了解。

说明：

- 如果需要在 PVC 被 Pod 挂载状态下进行在线扩容，可以通过 `kubectl -n kube-system edit deployment csi-cdsplugin-controller-server`，给 `csi-cdsplugin` 容器的添加启动参数`--enable-online-expansion`，来启用在线扩容功能。
- 但需要特别注意的是，高负载情况下在线扩容可能会出现IO性能下降或导致IO错误，建议选择低负载时操作并提前创建快照备份。

使用文件存储CFS

容器引擎 CCE 支持通过创建 PV/PVC，并为工作负载挂载数据卷方式使用百度智能云文件存储CFS。本文将介绍如何在集群中动态和静态挂载文件存储。

使用限制

- 创建的 CFS 实例和挂载点须和集群节点在同一 VPC 内。

前提条件

1. 创建 CFS 实例，具体操作请参见 创建文件系统。
2. 添加 CFS 挂载点，具体操作请参见 添加挂载点。
3. 获取 CFS 实例挂载地址，具体操作请参见 获取挂载地址。

操作步骤

本文以 CFS 挂载点地址为 cfs-test.baidubce.com 为例。

动态 PV/PVC 方式挂载 CFS

方式一：使用 kubectl 命令行操作

1. 创建 StorageClass 和 Provisioner

dynamic-cfs-template.yaml 是一个 yaml 文件模板，包含了需要创建的集群资源信息。

dynamic-cfs-template.yaml 文件内容如下：

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
```

```
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: kube-system
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: kube-system
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: nfs-client-provisioner
  namespace: kube-system
---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-cfs
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  mountOptions:
    - hard
    - nfsvers=4.1
    - nordirplus
  nfs:
    path: {{NFS_PATH}}
    server: {{NFS_SERVER}}
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-cfs
```

```
  namespace: kube-system
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-client-provisioner
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nfs-client-provisioner
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          image: registry.baidubce.com/cce-plugin-pro/nfs-client-provisioner:latest
          imagePullPolicy: Always
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: {{PROVISIONER_NAME}}
            - name: NFS_SERVER
              value: {{NFS_SERVER}}
            - name: NFS_PATH
              value: {{NFS_PATH}}
      volumes:
        - name: nfs-client-root
          persistentVolumeClaim:
            claimName: pvc-cfs
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{STORAGE_CLASS_NAME}}
provisioner: {{PROVISIONER_NAME}}
parameters:
  archiveOnDelete: "{{ARCHIVE_ON_DELETE}}"
  sharePath: "{{SHARE_PATH}}"
mountOptions:
  - hard
  - nfsvers=4.1
  - nordirplus
```

dynamic-cfs-template.yaml 模板文件中可自定义的选项及说明如下：

选项	说明
NFS_SERVER	CFS 挂载点地址。
NFS_PATH	CFS 远程挂载目录，注意该目录在使用前需要预先存在，如果目录不存在会导致 provisioner 插件启动失败。
SHARE_PATH	不同 PVC 的 CFS 挂载目录是否隔离，**true-不隔离，false-隔离**。若指定隔离，则会在 CFS 挂载目录下为每个 PVC 创建一个子目录，对应 PVC 使用该子目录作为挂载目录；否则所有 PVC 共享挂载目录。
ARCHIVE_ON_DELETE	删除 PVC 后是否保留对应数据，仅当 PVC 挂载目录隔离时生效，**true-保留，false-不保留**；PVC 挂载目录共享时，删除 PVC 不会删除任何数据。设置为不保留则直接删除对应 PVC 的子目录，否则仅将原子目录名加上 `archive-` 前缀后保留。
STORAGE_CLASS_NAME	创建的 StorageClass 名称。
PROVISIONER_NAME	Provisioner 名称。

支持 shell 的系统中，可以直接使用下面的 replace.sh 脚本进行 yaml 模板中模板变量的替换操作。

```
 # !/bin/sh
# user defined vars

NFS_SERVER="cfs-test.baidubce.com"
NFS_PATH="/cce/shared"
SHARE_PATH="true" # 不同PVC的挂载目录是否隔离，true-不隔离，false-隔离
ARCHIVE_ON_DELETE="false" # 删除PVC是否保留对应数据，仅当PVC挂载目录隔离时生效，true-保留，false-不保留
STORAGE_CLASS_NAME="sharedcfs" # StorageClass名称
PROVISIONER_NAME="baidubce/cfs-provisioner" # provisioner名称

YAML_FILE="./dynamic-cfs-template.yaml"

# replace template vars in yaml file

sed -i "s#{{SHARE_PATH}}#$SHARE_PATH#" $YAML_FILE
sed -i "s#{{ARCHIVE_ON_DELETE}}#$ARCHIVE_ON_DELETE#" $YAML_FILE
sed -i "s#{{STORAGE_CLASS_NAME}}#$STORAGE_CLASS_NAME#" $YAML_FILE
sed -i "s#{{PROVISIONER_NAME}}#$PROVISIONER_NAME#" $YAML_FILE
sed -i "s#{{NFS_SERVER}}#$NFS_SERVER#" $YAML_FILE
sed -i "s#{{NFS_PATH}}#$NFS_PATH#" $YAML_FILE
```

1. 将脚本中前半段中的 shell 变量替换为期望值，将 replace.sh 脚本和 dynamic-cfs-template.yaml 文件放置在同一个目录下，执行 `sh replace.sh` 即可。
2. 或者采用其他方式，将模板 yaml 文件中的模板变量替换为期望值。
3. 最后，使用 kubectl 工具，执行 `kubectl create -f dynamic-cfs-template.yaml` 完成 StorageClass 和 Provisioner 的创建。

```
 $ kubectl create -f dynamic-cfs-template.yaml
clusterrole "nfs-client-provisioner-runner" created
clusterrolebinding "run-nfs-client-provisioner" created
role "leader-locking-nfs-client-provisioner" created
rolebinding "leader-locking-nfs-client-provisioner" created
serviceaccount "nfs-client-provisioner" created
persistentvolume "pv-cfs" created
persistentvolumeclaim "pvc-cfs" created
deployment "nfs-client-provisioner" created
storageclass "sharedcfs" created
$ kubectl get pod --namespace kube-system  | grep provisioner
nfs-client-provisioner-c94494f6d-dlxmj   1/1       Running   0          26s
```

如果相应的 Pod 进入 Running 状态，则动态绑定 PV 所需的资源已经建立成功。

2. 创建 PVC 时动态生成 PV 并绑定

- 在 PVC Spec 中指定上面创建的 StorageClass 名称，则在创建 PVC 时，会自动调用相应 StorageClass 绑定的 Provisioner 生成相应的 PV 进行绑定。
- 使用 kubectl，执行 `kubectl create -f dynamic-pvc-cfs.yaml` 完成 PVC 的创建。

- 假设创建的 StorageClass 名称为 `sharedcfs`，对应的 dynamic-pvc-cfs.yaml 文件如下所示：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: dynamic-pvc-cfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: sharedcfs
  resources:
    requests:
      storage: 5Gi
```

- 创建 PVC 后，可以看见相应的 PV 自动创建，PVC 状态变为 `Bound`，即 PVC 已经与新创建的 PV 绑定。

```
$ kubectl create -f dynamic-pvc-cfs.yaml
persistentvolumeclaim "dynamic-pvc-cfs" created
$ kubectl get pvc
NAME              STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
dynamic-pvc-cfs   Bound     pvc-6dbf3265-bbe0-11e8-bc54-fa163e08135d   5Gi        RWX            sharedcfs      4s
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM                    STORAGECLASS   REASON
AGE
pv-cfs                                     5Gi        RWX            Retain           Bound     kube-system/pvc-cfs                             21m
pvc-6dbf3265-bbe0-11e8-bc54-fa163e08135d   5Gi        RWX            Delete           Bound     default/dynamic-pvc-cfs   sharedcfs
7s
```

3. 在 Pod 内挂载 PVC

- 在 Pod spec 内指定相应的 PVC 名称即可，使用 kubectl，执行 `kubectl create -f dynamic-cfs-pod.yaml` 完成资源的创建。
- 对应的 `dynamic-cfs-pod.yaml` 文件如下所示：

```
kind: Pod
apiVersion: v1
metadata:
  name: test-pvc-pod
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: cfs-pvc
        mountPath: "/cfs-volume"
  volumes:
    - name: cfs-pvc
      persistentVolumeClaim:
        claimName: dynamic-pvc-cfs
```

Pod 创建后，可以读写容器内的 `/cfs-volume` 路径来访问相应的 CFS 存储上的内容。

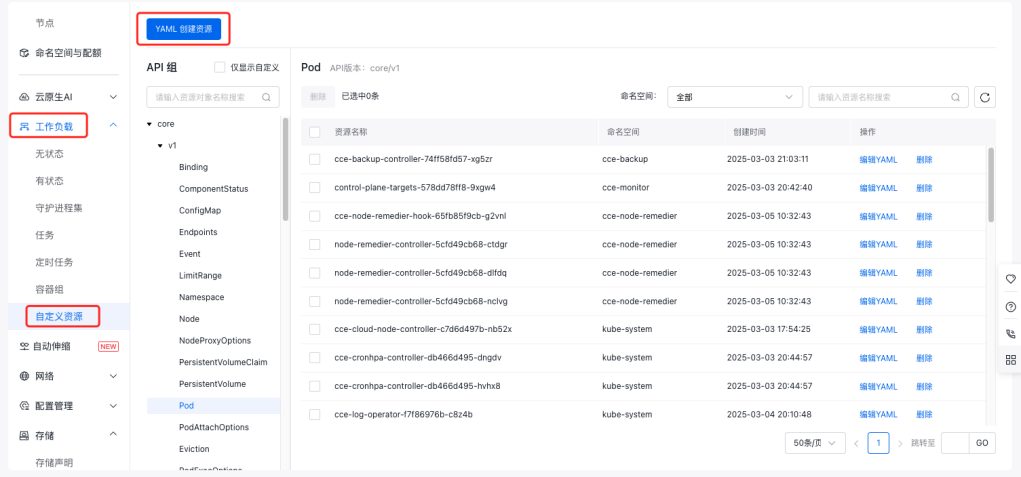
4. 释放 PVC 时动态销毁绑定 PV

删除 PVC 时，与之绑定的动态 PV 会被一同删除，其中的数据则根据用户定义的 `SHARE_PATH` 和 `ARCHIVE_ON_DELETE` 选项进行相应的保留或删除处理。

```
 $ kubectl get pvc
NAME              STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
dynamic-pvc-cfs   Bound     pvc-6dbf3265-bbe0-11e8-bc54-fa163e08135d   5Gi        RWX            sharedcfs      9m
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM                     STORAGECLASS   REASON
AGE
pv-cfs                                     5Gi        RWX            Retain           Bound     kube-system/pvc-cfs                      31m
pvc-6dbf3265-bbe0-11e8-bc54-fa163e08135d   5Gi        RWX            Delete           Bound     default/dynamic-pvc-cfs   sharedcfs
9m
$ kubectl delete -f dynamic-pvc-cfs.yaml
persistentvolumeclaim "dynamic-pvc-cfs" deleted
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM             STORAGECLASS   REASON   AGE
pv-cfs    5Gi        RWX            Retain           Bound     kube-system/pvc-cfs                      31m
$ kubectl get pvc
No resources found.
```

方式二：通过控制台操作

1. 登陆 CCE 控制台，单击集群名称进入集群详情页。
2. 创建Provisioner。
   a. 在左侧导航栏中选择**工作负载 > 自定义资源**，进入自定义资源管理页面。
   b. 点击页面上方**YAML创建资源**按钮，在弹框中分别创建以下资源。




```
  kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: kube-system
roleRef:
  kind: ClusterRole
```

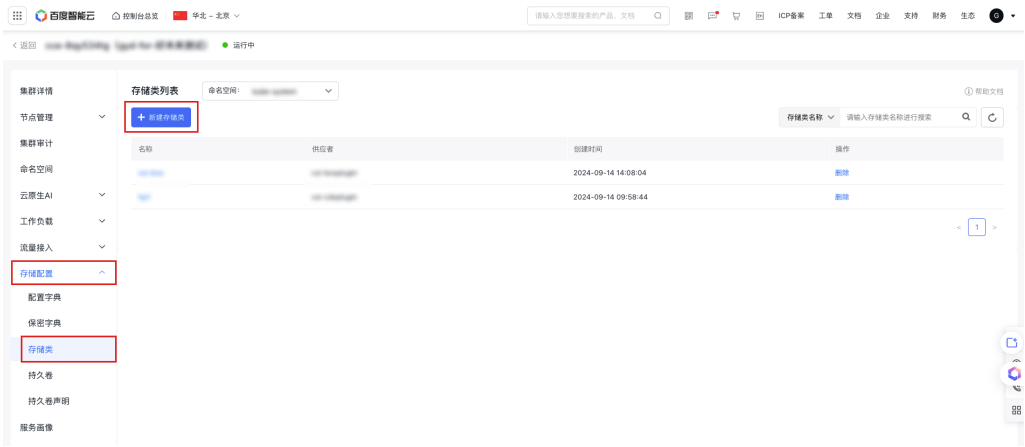
```
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: kube-system
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: kube-system
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
  namespace: kube-system
---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-cfs
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  mountOptions:
    - hard
    - nfsvers=4.1
    - nordirplus
  nfs:
    path: {{NFS_PATH}}
    server: {{NFS_SERVER}}
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-cfs
  namespace: kube-system
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-client-provisioner
  namespace: kube-system
spec:
```

```
  selector:
    matchLabels:
      app: nfs-client-provisioner
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          image: registry.baidubce.com/cce-plugin-pro/nfs-client-provisioner:latest
          imagePullPolicy: Always
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: {{PROVISIONER_NAME}} # 指定 Provisioner 名称
            - name: NFS_SERVER
              value: {{NFS_SERVER}} # 指定 CFS 挂载点地址
            - name: NFS_PATH
              value: {{NFS_PATH}} # 指定 CFS 远程挂载目录，注意该目录在使用前需要预先存在，如果目录不存在会导致 provisioner 插件
启动失败
      volumes:
        - name: nfs-client-root
          persistentVolumeClaim:
            claimName: pvc-cfs
```

3. 创建存储类 StorageClass。

   a. 在左侧导航栏中选择**存储配置 > 存储类**，进入存储类列表页面。

   b. 点击存储类列表上方**新建存储类**按钮，在文件模版中输入以下 yaml 内容。




```
  kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{STORAGE_CLASS_NAME}} # 指定 StorageClass 名称
provisioner: {{PROVISIONER_NAME}} # 指定创建的 Provisioner 名称
parameters:
  archiveOnDelete: "{{ARCHIVE_ON_DELETE}}" # 删除 PVC 后是否保留对应数据，仅当 PVC 挂载目录隔离时生效，true-保留，false-不
保留；PVC 挂载目录共享时，删除 PVC 不会删除任何数据。设置为不保留则直接删除对应 PVC 的子目录，否则仅将原子目录名加上
archive- 前缀后保留。
  sharePath: "{{SHARE_PATH}}" # 不同 PVC 的 CFS 挂载目录是否隔离，true-不隔离，false-隔离。若指定隔离，则会在 CFS 挂载目录下
为每个 PVC 创建一个子目录，对应 PVC 使用该子目录作为挂载目录；否则所有 PVC 共享挂载目录。
mountOptions:
  - hard
  - nfsvers=4.1
  - nordirplus
```

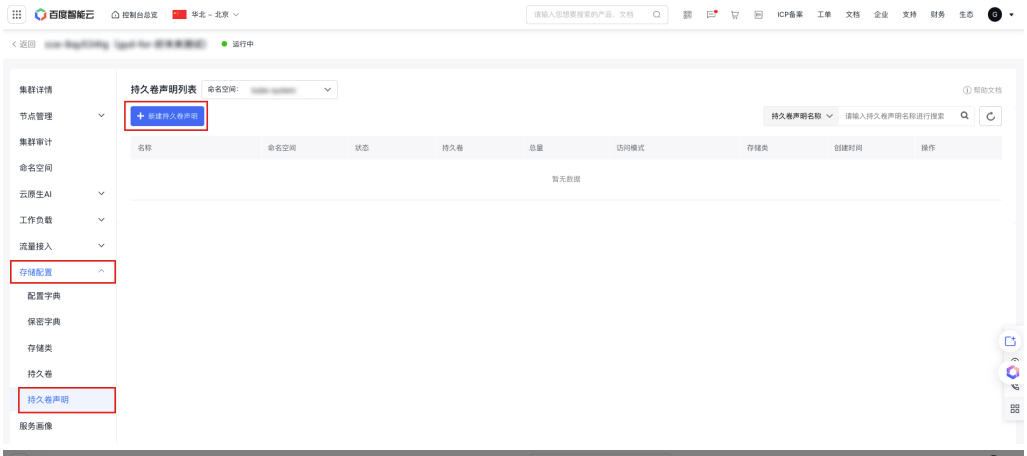
c. 点击**确定**，将为您创建存储类。

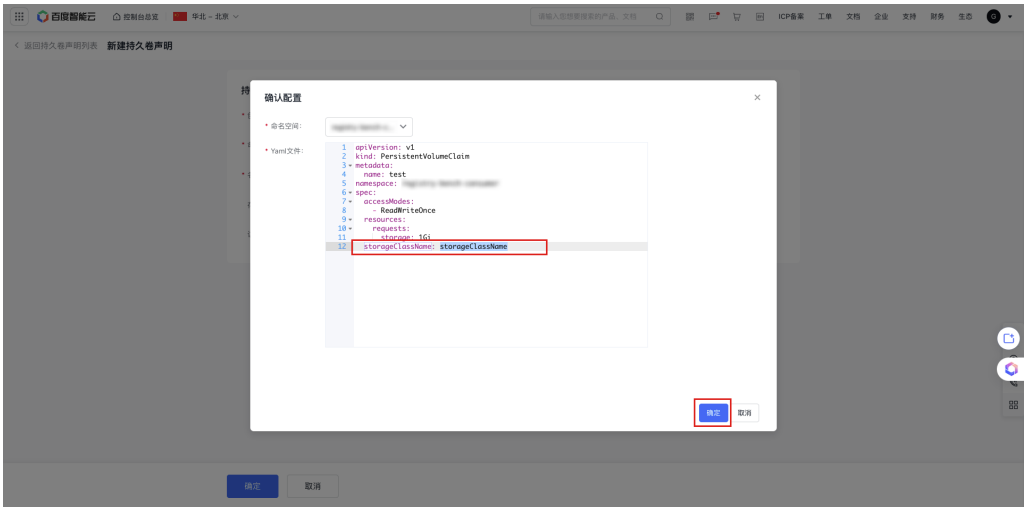
4. 创建持久卷声明 PVC 后动态生成 PV 并绑定。

a. 在左侧导航栏选择**存储配置 > 持久卷声明**，进入持久卷声明列表。

b. 点击持久卷声明列表上方**新建持久卷声明**按钮，选择**表单创建**或 **Yaml 创建**。

c. 若选择表单创建则需填写相关参数，其中三种访问模式详情参见存储管理概述，点击**确定**后在**确认配置**弹窗中输入 `storageClassName` 的值，即存储类的名称。确认配置无误后点击**确定**。

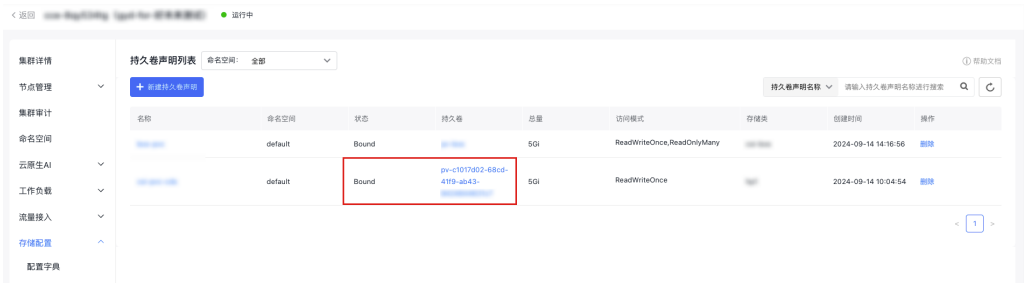




d. 若选择YAML创建，在文件模版中输入以下 yaml 内容。

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-cfs
  namespace: kube-system
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: {{STORAGE_CLASS_NAME}} # 指定创建的 StorageClass 名称
  resources:
    requests:
      storage: 5Gi
```

e. 创建 PVC 后，在**持久卷声明列表 > 持久卷列**可以看见相应的 PV 自动创建，PVC 状态列显示为 Bound，即 PVC 已经与新创建的 PV 绑定。



5. 创建应用并挂载 PVC，在 Pod spec 内指定相应的 PVC 名称即可。Pod 创建后，可以读写容器内的 /cfs-volume 路径来访问相应的 CFS 存

储上的内容。

静态 PV/PVC 方式挂载 CFS

方式一：通过 kubectl 命令行操作

1. 在集群中创建 PV 和 PVC 资源

- 使用kubectl，执行 `kubectl create -f pv-cfs.yaml` 完成 PV 的创建
- 对应的 `pv-cfs.yaml` 文件如下所示：

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-cfs
spec:
  capacity:
    storage: 8Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  mountOptions:
    - hard
    - nfsvers=4.1
    - nordirplus
  nfs:
    path: /
    server: cfs-test.baidubce.com
```

> **注意：**
>
> - yaml 中 server 字段对应的是 CFS 挂载点地址。
> - yaml 中 path 字段对应的是 CFS 挂载目录，该目录需要在挂载前预先存在。

创建 PV 后，输入`kubectl get pv` 可以看见一个 available 状态的 PV，如下所示：

```
$ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM    STORAGECLASS   REASON    AGE
pv-cfs   8Gi        RWX            Retain           Available                           3s
```

建立一个能够与该 PV 绑定的 PVC

使用 kubectl，执行 `kubectl create -f pvc-cfs.yaml` 完成 PVC 的创建

对应的 `pvc-cfs.yaml` 文件如下所示：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-cfs
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 8Gi
```

绑定前，PVC 为 pending 状态

```
$ kubectl get pvc
NAME     STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-cfs  Pending                                       2s                                  2s
```

绑定后，PV 和 PVC 状态变为 Bound

```
 $ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM           STORAGECLASS   REASON   AGE
pv-cfs   8Gi        RWX            Retain           Bound     default/pvc-cfs                          36s
$ kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-cfs   Bound    pv-cfs   8Gi        RWX                           1m
```

有关 PV 和 PVC 的更多设置和字段说明，参见k8s官方文档

2. 在 Pod 内挂载 PVC

在 Pod spec 内指定相应的 PVC 名称即可，使用 kubectl，执行 `kubectl create -f demo-cfs-pod.yaml` 完成 Pod 的创建

对应的 `demo-cfs-pod.yaml` 文件如下所示：

```
 kind: Pod
apiVersion: v1
metadata:
 name: demo-cfs-pod
 labels:
   app: demo-cfs-pod
spec:
 containers:
 - name: nginx
   image: nginx
   volumeMounts:
     - name: cfs-pvc
       mountPath: "/cfs-volume"
 volumes:
   - name: cfs-pvc
     persistentVolumeClaim:
       claimName: pvc-cfs
```

Pod 创建后，可以读写容器内的 `/cfs-volume` 路径来访问相应的 CFS 存储上的内容。

由于创建 PV 和 PVC 时指定了 `accessModes` 为 `ReadWriteMany`，该 PVC 可以被多节点上的 Pod 挂载读写。

3. 释放 PV 和 PVC 资源

完成存储资源的使用后，可以释放 PVC 和 PV 资源。在释放 PVC 和 PV 之前，需要先删除挂载了对应 PVC 的所有 Pod。

使用以下命令可以释放 PVC

```
 $ kubectl delete -f  pvc-cfs.yaml
```

释放 PVC 后，原来与之绑定的 PV 状态会变为 Release，如下所示：

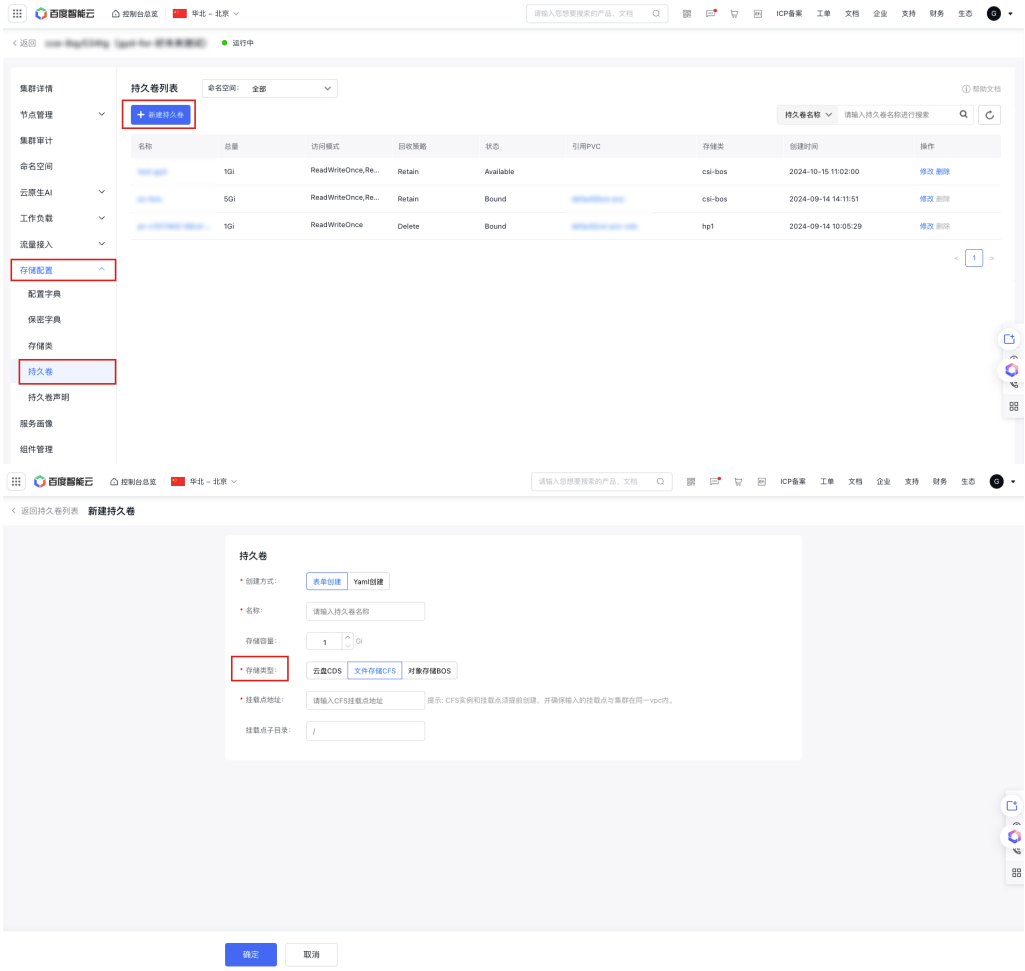
```
 NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM           STORAGECLASS   REASON   AGE
pv-cfs   8Gi        RWX            Retain           Released   default/pvc-cfs                          16m
```

输入以下指令释放 PV 资源

```
 $ kubectl delete -f  pv-cfs.yaml
```

方式二：通过控制台操作

1. 登陆 CCE 控制台，单击集群名称进入集群详情页。
2. 创建持久卷 PV。
   a. 在左侧导航栏选择**存储配置** > **持久卷**，进入持久卷列表。
   b. 点击持久卷列表上方**新建持久卷**按钮，选择表单创建或 Yaml 创建。
   c. 若选择表单创建则需填写相关参数，根据需求设置存储容量，存储类型选择**文件存储 CFS**。挂载点地址可参见获取挂载地址。点击**确定**在二次弹窗中确认配置后点击**确定**创建 PV。



3. 创建可以和持久卷 PV 绑定的持久卷声明 PVC。

   a. 在左侧导航栏选择**存储配置 > 持久卷声明**，进入持久卷声明列表。

   b. 点击持久卷列表上方**新建持久卷声明**按钮，选择表单创建或 Yaml 创建。

   c. 根据之前创建的 PV 配置 PVC 的存储容量、访问模式、存储类（可选），点击**确定**创建 PVC。会查找现有的 PV 资源，寻找与 PVC 请求相匹配的 PV。

   d. 绑定后，可以分别在持久卷列表和持久卷声明列表中看见 PV 和 PVC 的状态列变为 Bound。

4. 创建应用并挂载 PVC，在 Pod spec 内指定相应的 PVC 名称即可。

使用对象存储BOS

概述

容器引擎 CCE 支持通过创建 PV/PVC，并为工作负载挂载数据卷方式使用百度智能云对象存储BOS。本文将介绍如何在集群中静态挂载对象存储。

使用限制

- 集群 Kubernetes 版本需大于或等于 1.16。
- 请注意创建的 BOS Bucket 所处地域。如果创建的 BOS Bucket 与容器集群分属于不同地域，要求挂载点所在 BCC/BBC 开放公网访问，同地域则可以通过内网读写。

前提条件

- 集群已安装对象存储组件，更多信息请见 CCE CSI BOS Plugin 说明。

操作步骤

本文介绍通过 Kubectl 或者 CCE 控制台创建 PV/PVC 实现静态挂载 BOS。

方式一：通过 kubectl 命令行操作

1. 在集群中创建 AK/SK 的 Secret，用以访问 BOS 存储

```
kubectl create secret generic csi-bos-secret \
--from-literal=ak=<Your-AK> \
--from-literal=sk=<Your-SK>
```

> **说明：** 关于 AK/SK 的更多信息参考 如何获取 AK 和 SK

2. 创建持久卷 PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-bos
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  - ReadOnlyMany
  capacity:
    storage: 5Gi
  storageClassName: csi-bos
  csi:
    driver: csi-bosplugin
    volumeHandle: v-XXXXXX  # 对应的BOS的bucketName, 支持挂载BOS bucket子目录，如: bucketName/dirName
    nodePublishSecretRef:
      name: csi-bos-secret
      namespace: default
  mountOptions:
  - "-o meta_expires=0"
  persistentVolumeReclaimPolicy: Retain
```

> **说明：**
>
> - mountOptions：由于 BOS bucket 挂载依赖 Bosfs，因此支持通过 mountOptions 指定 Bosfs 的启动参数，支持的参数详情见 BOS参数说明，常用参数和说明如下:
>   - `meta_expires`：meta 缓存过期时间，单位为秒。不设置该参数代表永不过期。上述PV yaml示例中该参数被设置为0，代表不缓存。缓存过期时间过短可能影响写入性能，过长可能导致挂载点内看不到其他客户端写入同一bucket的数据。同一bucket一写多读场景建议设置为0，纯写入场景建议设置为3600，单点读写场景可以不设置。
>   - `allow_other`：是否允许非 root 用户读取挂载点内数据，默认只有 root 用户可读写，容器内非 root 进程如果要读取挂载点内数据需要设置该参数，更细粒度权限的管理可以配合 `mount_umask` 参数一同使用。
> - BOS 支持一写多读，对应的 accessMode 只支持 **ReadWriteOnce + ReadOnlyMany**

创建 PV 后，输入 `kubectl get pv` 可以看见一个 available 状态的 PV，如下所示：

```
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM     STORAGECLASS   REASON   AGE
bos-pv    5Gi        RWO,ROX        Retain           Available             csi-bos                3s
```

3. 创建持久卷声明 PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: bos-pvc
spec:
  accessModes:
  - ReadWriteOnce
  - ReadOnlyMany
  resources:
    requests:
      storage: 5Gi
  storageClassName: csi-bos
```

> **说明**：YAML 中 storageClassName 字段用于和 PV 关联，建议填写，如果集群中定义了多种存储类，通过指定 storageClassName，用户可以请求特定类存储资源 PV。

绑定前，PVC 为 pending 状态

```
$ kubectl get pvc
NAME      STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
bos-pvc   Pending                                       csi-bos        2s
```

绑定后，PV 和 PVC 状态变为 Bound

```
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM             STORAGECLASS   REASON    AGE
bos-pv    5Gi        RWX            Retain           Bound     default/bos-pvc                            36s
$ kubectl get pvc
NAME      STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
bos-pvc   Bound     bos-pv    5Gi        RWO,ROX        csi-bos        1m
```

有关 PV 和 PVC 的更多设置和字段说明，参见 Kubernetes官方文档。

4. 在 Pod 内挂载 PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx01
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx01
    volumeMounts:
    - mountPath: /var/lib/www/html
      name: bos-pvc
    - mountPath: /var/lib/www/html000
      name: bos-pvc
      readOnly: true
    livenessProbe:
      exec:
        command:
        - stat
        - /var/lib/www/html
  volumes:
  - name: bos-pvc
    persistentVolumeClaim:
      claimName: bos-pvc
      readOnly: false
```

Pod 创建后，可以读写容器内的 `/var/lib/www/html` 路径来访问相应的 BOS 存储上的内容，同时该路径支持读写，`/var/lib/www/html000` 支持只读。

> **注意：**
>
> 由于 CSI bosplugin 依赖 bosfs，在 Pod 中配置 livenessProbe 检查挂载点状态可以避免容器无法感知 bosfs 异常重启后的挂载点失效的问题，强烈建议添加该配置。

同时，支持在其余机器上挂载只读盘，如创建一个包含只读 BOS bucket 的 Pod

```
 apiVersion: v1
kind: Pod
metadata:
  name: nginx01-bbaa
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx01
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/lib/www/html000
      name: bos-pvc
      readOnly: true
    livenessProbe:
      exec:
        command:
        - stat
        - /var/lib/www/html000
  volumes:
  - name: bos-pvc
    persistentVolumeClaim:
      claimName: bos-pvc
      readOnly: true
```

5. 释放 PV 和 PVC 资源

完成存储资源的使用后，可以释放 PVC 和 PV 资源。

使用以下命令可以释放 PVC：

```
$ kubectl delete -f  bos-pvc.yaml
```

释放 PVC 后，原来与之绑定的 PV 状态会变为 Release，如下所示：

```
 NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM           STORAGECLASS   REASON    AGE
bos-pv    5Gi        RWO,ROX        Retain           Released   default/bos-pvc   csi-bos       16m
```

输入以下指令释放 PV 资源：

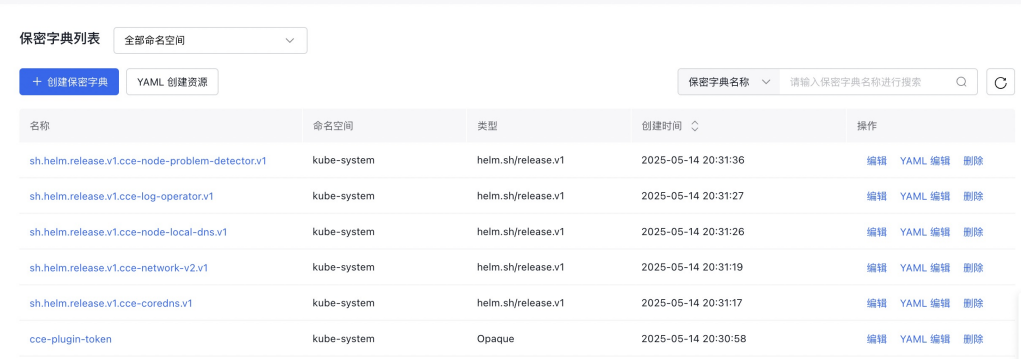
```
$ kubectl delete -f  bos-pv.yaml
```

方式二：通过控制台操作

1. 登陆 CCE 控制台，点击集群名称进入集群详情页面。
2. 在集群中创建 AK/SK 的 Secret，用以访问 BOS 存储。获取 AK/SK 参见 如何获取AK/SK。

a. 左侧导航栏选择**配置管理 > 保密字典**，点击保密字典列表上方**创建保密字典**。

b. 在新建保密字典页面中配置相关信息，补充 data，确认无误后点击**确定**。



3. 创建持久卷 PV。

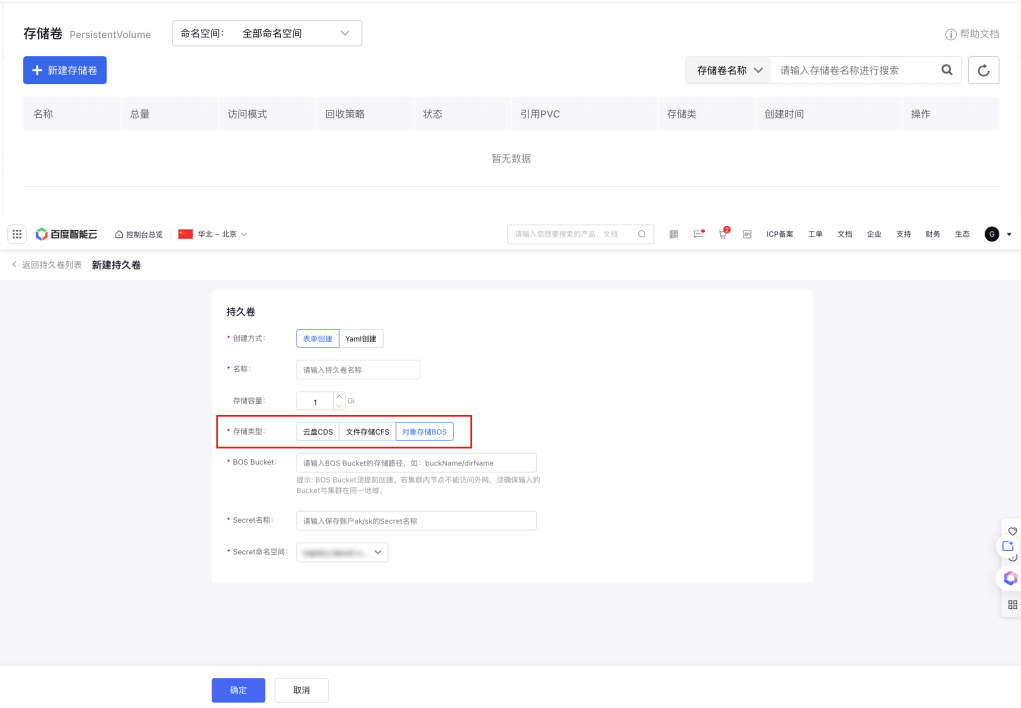
a. 左侧导航栏选择**存储 > 存储卷**，点击持久卷列表上方**新建存储卷**。

b. 可选择表单创建或 yaml 创建，若选择表单创建则需进行相关配置。

c. 根据需求配置存储容量，存储类型选择**对象存储 BOS**，输入 BOS Bucket 的存储路径，存储路径可前往 BOS 控制台 查看。

d. 输入保存 AK/SK 的 Secret 名称，并选择 Secret 所在命名空间。

e. 在确认配置弹窗中确认配置无误后，点击**确定**创建 PV。



4. 创建存储声明 PVC。

a. 左侧导航栏选择**存储 > 存储声明**，点击存储声明列表上方**新建存储声明**。

b. 根据需求配置存储容量、访问模式后点击**确定**，在二次弹窗中确认相关配置，其中`storageClassName`字段用于和 PV 关联，建议您填写。确认无误后点击**确定**创建 PVC。

c. 在存储声明列表中可以看见 PVC 的绑定状态，若绑定成功则为 Bound，并且在存储卷列中可以看见对应绑定的 PV 名称。

5. 创建工作负载并在 Pod 中挂载 PVC。

a. 左侧导航栏选择**工作负载 > 无状态**，选择通过表单 / YAML 创建并配置挂载参数。

b. 由于 CSI bosplugin 依赖 bosfs，强烈建议添加 `livenessProbe` ，通过 `livenessProbe` 可以确保在 bosfs 异常重启后，容器能够感知到挂载点是否失效，从而避免相关服务中断。

6. 完成存储资源的使用后，可以释放 PVC 和 PV 资源。可以通过对应列表的**操作列 > 删除**释放 PV 和 PVC 资源。

使用并行文件存储PFS

容器引擎CCE支持通过创建PV/PVC，并为工作负载挂载数据卷方式使用百度智能云并行文件存储PFS。本文将介绍如何在集群中动态和静态挂载并行文件存储。

使用限制

- 集群Kubernetes版本需大于或等于1.16。
- PFS实例需要和集群在同一VPC内。
- 当前PFS暂不支持arm架构的集群以及Ubuntu 18.04/16.04、baidulinux 3.0、RockyLinux 8.6操作系统的节点，如业务场景需要请提工单。

前提条件

- 集群已安装并行文件存储组件，更多信息请见CCE CSI PFS Plugin 说明。

操作步骤

动态挂载并行文件存储

1. 创建存储类StorageClass

集群管理员可使用StorageClass为集群定义不同的存储类型。您可通过StorageClass配合PVC动态创建需要的存储资源。

本文介绍通过Kubectl方式创建并行文件存储PFS类型的StorageClass，自定义并行文件存储使用所需的模板。

您需要先创建PFS实例，操作步骤请参考创建并行文件系统。

```
 kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: pfs-sc
provisioner: csi-clusterfileplugin
allowVolumeExpansion: true
parameters:
 clusterIP:     #必填，PFS实例连接地址
 clusterPort:    #必填，当前端口固定为"8888"
 parentDir:     #必填，自定义路径即可
```

2. 创建持久卷声明PVC

注意storageClassName需要指定为上述部署storageClass时填写的storageClass名称。

```
 kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: csi-pvc-pfs
 namespace: default
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: pfs-sc
 resources:
  requests:
   storage: 50Gi # 指定PVC存储空间大小
```

3. 检查PVC状态为Bound

```
 $ kubectl get pvc csi-pvc-pfs
NAME          STATUS   VOLUME              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-pfs   Bound    pvc-1ab36e4d1d2711e9   50Gi       RWX            pfs-sc         4s
```

4. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
 apiVersion: v1
kind: Pod
metadata:
 name: test-pvc-pod
 namespace: default
 labels:
   app: test-pvc-pod
spec:
 containers:
 - name: test-pvc-pod
   image: nginx
   volumeMounts:
     - name: pfs-pvc
       mountPath: "/pfs-volume"
 volumes:
   - name: pfs-pvc
     persistentVolumeClaim:
       claimName: csi-pvc-pfs
```

静态挂载并行文件存储

1. 创建持久卷PV

通过`pv.spec.csi.volumeAttributes.path`指定需要挂载的路径，该路径为填写的parentDir路径的相对路径。

本文以PV在PFS中的路径为`/exist/some/dir`为例。如果对应的路径不存在，在挂载PV时会自动创建该路径。

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: static-pv-pfs
spec:
 accessModes:
 - ReadWriteMany
 storageClassName:
 capacity:
   storage: 100Gi
 csi:
   driver: csi-clusterfileplugin
   volumeHandle: data-id
   volumeAttributes:
     parentDir: /          #必填，自定义路径
     path: /exist/some/dir  #必填，指定需要挂载的路径，为PFS实例相对parentDir的目录
     clusterIP: ""         #必填，PFS实例连接地址
     clusterPort: "8888"    #必填，当前端口固定为8888
```

> **说明**
>
> - storageClassName在静态PV为空，无需填写。
> - volumeHandle必须保证唯一性，可以使用当前时间（如2024-09-27-11-30-00-00）或者生成uuid来确保。
> - clusterPort当前固定为"8888"

2. 创建持久卷声明PVC

```
 kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: static-pvc-pfs
 namespace: default
spec:
 accessModes:
   - ReadWriteMany
 storageClassName:
 resources:
   requests:
     storage: 100Gi
```

2. 检查PVC状态为Bound，并且与对应的PV绑定

```
 $ kubectl get pvc
NAME             STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS    AGE
static-pvc-pfs   Bound    static-pv-pfs   100Gi      RWX            pfs-static-sc   10s
```

3. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
 apiVersion: v1
kind: Pod
metadata:
 name: test-pvc-pod
 namespace: default
 labels:
   app: test-pvc-pod
spec:
 containers:
 - name: test-pvc-pod
   image: nginx
   volumeMounts:
     - name: pfs-pvc
       mountPath: "/pfs-volume"
 volumes:
   - name: pfs-pvc
     persistentVolumeClaim:
       claimName: static-pvc-pfs
```

连接多个PFS实例

支持多个pfs只需要再部署一个storageclass，clusterIP，parentDir填入另一个pfs实例的信息即可。

1. 新增一个存储类StorageClass

```
 kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: pfs-sc-2
provisioner: csi-clusterfileplugin
allowVolumeExpansion: true
parameters:
 clusterIP:      #必填，PFS实例连接地址
 clusterPort:    #必填，当前端口固定为"8888"
 parentDir:      #必填，自定义路径即可
```

2.1 动态挂载在PVC指定storageclass为新创建的storageClassName

```
 kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: csi-pvc-pfs-2
 namespace: default
spec:
 accessModes:
   - ReadWriteMany
 storageClassName: pfs-sc-2
 resources:
   requests:
     storage: 50Gi # 用户指定PVC存储空间
```

2.2 静态挂载在PV参数中填入另一个PFS实例相关信息即可

```
  apiVersion: v1
kind: PersistentVolume
metadata:
  name: static-pv-pfs
spec:
  accessModes:
  - ReadWriteMany
  storageClassName:
  capacity:
    storage: 100Gi
  csi:
    driver: csi-clusterfileplugin
    volumeHandle: data-id
    volumeAttributes:
      parentDir: /          #必填，自定义路径
      path: /exist/some/dir  #必填，指定需要挂载的路径，为PFS实例相对parentDir的目录
      clusterIP: ""         #必填，PFS实例连接地址
      clusterPort: "8888"    #必填，当前端口固定为8888
```

PFS配置项示例

示例1

```
    parentDir:"/kubernetes"
```

如果部署storageClass或者静态pv时填写上述配置，则动态创建的PV在pfs中的路径为`/kubernetes/pvc-xxxx-xxxx-xxxx-xxxx`；静态PV在pfs中的path路径必须位于`/kubernetes`路径下。

示例2

```
    parentDir:"/"
```

如果安装时填写上述配置，则动态创建的PV在pfs中的路径为`/pvc-xxxx-xxxx-xxxx-xxxx`；静态PV在pfs中的path路径必须位于`/`路径下。

使用并行文件存储PFS L2

容器引擎CCE支持通过创建PV/PVC，并为工作负载挂载数据卷方式使用百度智能云并行文件存储PFS 极速型L2。本文将介绍如何在集群中动态和静态挂载并行文件存储。

前提条件

- 集群已安装并行文件存储组件，更多信息请见CCE CSI PFS L2 Plugin 说明。

使用限制

- 仅支持使用CCE CSI PFS L2 Plugin组件关联挂载服务绑定的PFS实例。
- 在添加CCE节点时，需要注意安装的操作系统版本，以下为CCE和PFS极速型L2均支持的操作系统版本。

操作系统类型	发行版	内核版本
CentOS	8	4.18.0-348.7.1.el8_5.x86_64
Ubuntu	20.04	5.4.0-135-generic
Ubuntu	20.04	5.4.0-139-generic
Ubuntu	22.04	5.15.0-72-generic

操作步骤

动态挂载并行文件存储

1. 创建存储类StorageClass

集群管理员可使用StorageClass为集群定义不同的存储类型。您可通过StorageClass配合PVC动态创建需要的存储资源。

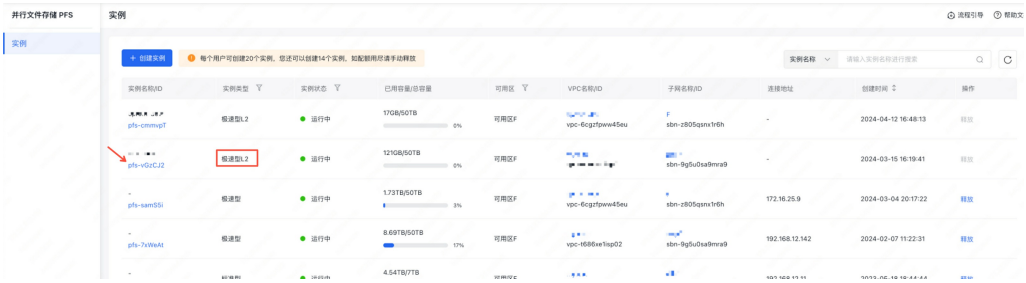
本文介绍通过Kubectl方式创建并行文件存储PFS L2类型的StorageClass，自定义并行文件存储使用所需的模板。

您需要先创建PFS实例，操作步骤请参考创建并行文件系统极速型L2。

```
 kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: pfsl2-sc
provisioner: spectrumscale.csi.ibm.com
parameters:
 mountTargetId: <mountTargetId>    #必填，<mountTargetId>为挂载服务ID
 pfsId: <pfsId>    #必填，<pfsID>为所需要挂载的并行文件存储PFS L2实例ID
 #volDirBasePath: <volDirBasePath> #可选，<volDirBasePath>为待挂载PFS系统内的子路径，动态方式创建的volume将基于该路径创建pvc，该路径需要在系统内已经存在，否则pvc会创建失败。
reclaimPolicy: Delete
```

- PFS 挂载服务ID获取

- PFS L2实例ID获取



2. 创建持久卷声明PVC

注意storageClassName需要指定为上述部署storageClass时填写的storageClass名称。

```
 kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: dynamic-pvc
spec:
 accessModes:
  - ReadWriteMany
 storageClassName: pfsl2-sc
 resources:
  requests:
   storage: 50Gi # 指定PVC存储空间大小
```

3. 检查PVC状态为Bound

```
 $ kubectl get pvc dynamic-pvc
NAME          STATUS   VOLUME                CAPACITY   ACCESS MODES   STORAGECLASS   AGE
dynamic-pvc   Bound    pvc-1ab36e4d1d2711e9   50Gi       RWX            pfsl2-sc       4s
```

4. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
 apiVersion: v1
kind: Pod
metadata:
 name: test-pvc-pod
 namespace: default
 labels:
   app: test-pvc-pod
spec:
 containers:
 - name: test-pvc-pod
   image: nginx
   volumeMounts:
     - name: pfs-pvc
       mountPath: "/pfs-volume"
 volumes:
   - name: pfs-pvc
     persistentVolumeClaim:
       claimName: dynamic-pvc
```

静态挂载并行文件存储

1. 创建持久卷PV

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: static-pv-pfsl2
spec:
 accessModes:
 - ReadWriteMany
 capacity:
   storage: 100Gi
 csi:
   driver: spectrumscale.csi.ibm.com #必填
   volumeHandle: <mountTargetId>;<pfsId>;<mountPath> #必填，分别对应挂载服务ID，PFS L2实例ID以及挂载地址，其中mountPath为待挂载的PFS目录在节点上的绝对路径。
```

- 获取绝对路径

进入PFS L2实例详情页，选择挂载信息，客户端本地挂载路径即为该绝对路径。

2. 创建持久卷声明PVC

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: static-pvc-pfsl2 # static PVC name
spec:
 volumeName: static-pv-pfsl2 # static PV name
 accessModes:
 - ReadWriteMany
 resources:
   requests:
     storage: 100Gi
```

2. 检查PVC状态为Bound，并且与对应的PV绑定

```
 $ kubectl get pvc
NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS    AGE
static-pvc-pfsl2  Bound   static-pv-pfsl2   1Gi      RWX                            10s
```

3. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
 apiVersion: v1
kind: Pod
metadata:
 name: test-pvc-pod
 namespace: default
 labels:
   app: test-pvc-pod
spec:
 containers:
 - name: test-pvc-pod
   image: nginx
   volumeMounts:
     - name: pfs-pvc
       mountPath: "/pfs-volume"
 volumes:
   - name: pfs-pvc
     persistentVolumeClaim:
       claimName: static-pvc-pfsl2
```

使用数据湖存储加速工具RapidFS

容器引擎CCE支持通过创建PV/PVC，并为工作负载挂载数据卷方式使用百度智能云数据湖存储加速工具RapidFS。本文将介绍如何在集群中动态和静态挂载数据湖存储加速工具RapidFS。

使用限制

- 集群Kubernetes版本需大于或等于1.18。
- RapidFS实例需要和集群在同一VPC内。

前提条件

- 在数据湖存储加速工具控制台创建RapidFS实例，具体操作请参见新建缓存实例。
- 在数据湖存储加速工具控制台新增导入数据源，具体操作请参见导入数据源。
- 在CCE控制台**Helm模板 > 百度智能云模板**中使用”cce-csi-rapidfs-plugin“模板在目标集群中安装RapidFS组件。
- 挂载RapidFS数据源的节点需要设置根路径为`shared`挂载模式，即需要在对应服务器上执行`mount --make-rshared /`命令。

操作步骤

动态挂载RapidFS

1. 创建存储类StorageClass

集群管理员可使用StorageClass为集群定义不同的存储类型。您可通过StorageClass配合PVC动态创建需要的存储资源。

本文介绍通过Kubectl方式创建RapidFS类型的StorageClass，自定义RapidFS存储使用所需的模板。

```
 kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: rapidfs-sc
provisioner: csi-rapidfsplugin
parameters:
 rapidfsAddress:  #必填，目标数据源的挂载地址，数据源关联的RapidFS实例必须和集群在同一个VPC下
 rapidfsID：  #必填，数据源关联RapidFS实例的ID
```

> **注意：**
>
> - 挂载地址对应的数据源关联的权限组需要为集群中的Pod配置访问的权限（只读或读写权限），否则容器无法进行读写操作，可以直接设置CCE集群的VPC网段或关联默认权限组。

为Pod挂载RapidFS类型PV时默认会通过创建fuse pod进行挂载，支持自定义Fuse Pod相关配置满足业务灵活使用；

如无特殊需求，建议以下参数使用默认值即可。如果配置的 CPU 或 Memory 过小，可能影响访问 rapidfs 存储的性能或可用性。

```
  parameters:
  memRequest: 4Gi #可选，指定fuse pod内存的请求值，默认为4Gi
  memLimit: 64Gi #可选，指定fuse pod内存的上限值，默认为64Gi
  cpuRequest: 4000m #可选，指定fuse pod cpu的请求值，默认为4000m
  cpuLimit: 16000m #可选，指定fuse pod cpu的请求值，默认为16000
  mountOptions: ""  #可选，fuse pod启动参数，多个使用英文逗号分隔，EnableNuma=1时候挂载进程绕开和绑定本机ip网卡相同的numa节点（仅支持设置为0或1，0表示不绑定），支持添加参数"EnableNuma=1,NumaNode=x"指定绑定的numa节点，x为需要绑定的服务器numa节点
  mountEnv: ""    #可选，fuse pod环境变量配置，默认为空，多个使用英文逗号分隔
```

说明：

- Fuse Pod：用于将数据源挂载到节点的本地路径上，并且用户的容器通过Fuse Pod的访问数据源。
- 上面为Fuse Pod的配置参数，选填，若不设置直接使用默认值。

2. 创建持久卷声明PVC

storageClassName需要指定为上述部署storageClass时填写的storageClass名称。

```
  kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-pvc-rapidfs
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: rapidfs-sc # 需要指定为上述部署storageClass时填写的名称
  resources:
    requests:
      storage: 50Gi # 用户指定PVC存储空间（说明：RapidFS不支持限制存储容量，容器实际可访问的为数据源绑定的BOS存储桶目录。）
```

3. 检查PVC状态为Bound

```
  $ kubectl get pvc csi-pvc-rapidfs
NAME              STATUS    VOLUME                CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-rapidfs       Bound     pvc-1ab36e4d1d2711e9   50Gi       RWX            rapidfs-sc         4s
```

4. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
  namespace: default  # Pod和PVC在同一个namespace下
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: rapidfs-pvc
        mountPath: "/rapidfs-volume"
  volumes:
    - name: rapidfs-pvc
      persistentVolumeClaim:
        claimName: csi-pvc-rapidfs
```

静态挂载RapidFS

1. 创建持久卷PV

```
  apiVersion: v1
kind: PersistentVolume
metadata:
  name: static-pv-rapidfs
spec:
  accessModes:
  - ReadWriteMany
  storageClassName: # storageClassName在静态PV为空，无需填写
  capacity:
    storage: 100Gi
  csi:
    driver: csi-rapidfsplugin
    volumeHandle:
    volumeAttributes:
      rapidfsAddress:  #必填，目标数据源的挂载地址，数据源关联的RapidFS实例必须和集群在同一个VPC下
      rapidfsID：  #必填，数据源关联RapidFS实例的ID
```

> **注意：**
>
> - 挂载地址对应的数据源关联的权限组需要为集群中的Pod配置访问的权限（只读或读写权限），否则容器无法进行读写操作，可以直接设置CCE集群的VPC网段或关联默认权限组。

为Pod挂载RapidFS类型PV时默认会通过创建fuse pod进行挂载，支持自定义Fuse Pod相关配置满足业务灵活使用；

如无特殊需求，建议以下参数使用默认值即可。如果配置的 CPU 或 Memory 过小，可能影响访问 rapidfs 存储的性能或可用性。

```
  volumeAttributes:
  memRequest: 4Gi #可选，指定fuse pod内存的请求值，默认为4Gi
  memLimit: 64Gi #可选，指定fuse pod内存的上限值，默认为64Gi
  cpuRequest: 4000m #可选，指定fuse pod cpu的请求值，默认为4000m
  cpuLimit: 16000m #可选，指定fuse pod cpu的请求值，默认为16000m
  mountOptions: ""  #可选，fuse pod启动参数，多个使用英文逗号分隔，EnableNuma=1时候挂载进程绕开和绑定本机ip网卡相同的numa节点（仅支持设置为0或1，0表示不绑定），支持添加参数"EnableNuma=1,NumaNode=x"指定绑定的numa节点，x为需要绑定的服务器numa节点
  mountEnv: ""    #可选，fuse pod环境变量配置，默认为空，多个使用英文逗号分隔
```

> **说明：**
>
> - Fuse Pod：用于将数据源挂载到节点的本地路径上，并且用户的容器通过Fuse Pod的访问数据源。
> - 上面为Fuse Pod的配置参数，选填，若不设置直接使用默认值。

2. 创建持久卷声明PVC

```
  apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: static-pvc-rapidfs # 指定PVC名称
spec:
  volumeName: static-pv-rapidfs # 指定PV的名称，与上述PV保持一致
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi # 用户指定PVC存储空间（说明：RapidFS不支持限制存储容量，容器实际可访问的为数据源绑定的BOS存储桶目录。）
```

2. 检查PVC状态为Bound，并且与对应的PV绑定

```
  $ kubectl get pvc static-pvc-rapidfs
NAME              STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS    AGE
static-pvc-rapidfs   Bound    static-pv-rapidfs   100Gi      RWX            rapidfs-static-sc   10s
```

3. 在Pod中挂载PVC

Pod和PVC需要在同一个namespace下。

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
  namespace: default  # Pod和PVC在同一个namespace下
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: rapidfs-pvc
        mountPath: "/rapidfs-volume"
  volumes:
    - name: rapidfs-pvc
      persistentVolumeClaim:
        claimName: csi-pvc-rapidfs
```

监控日志

日志中心

日志概述

日志中心是容器引擎CCE基于百度智能云 日志服务BLS 提供的Kubernetes集群内日志采集方案，可以将集群内的服务或集群节点路径文件的日志信息推送至BLS，进行持久化存储和日志查询分析。当集群发生故障后，您也可以从日志服务中查看过去的日志信息，进行故障排查。

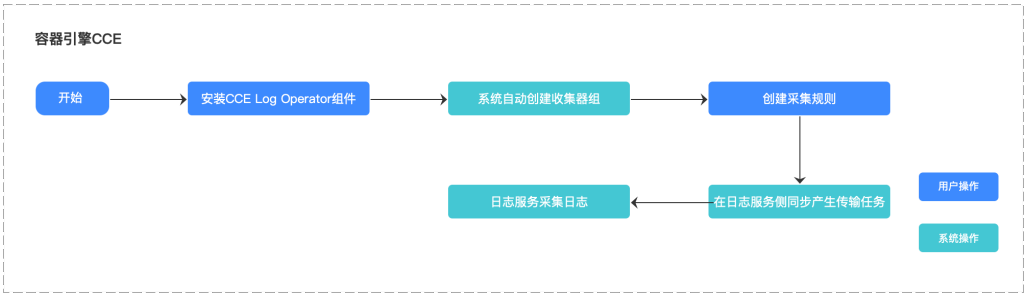
> 说明
>
> - 容器服务的日志功能基于百度智能云日志服务产品实现，日志服务产品的详细介绍，请参见 什么是日志服务。
> - 使用日志服务的功能，会产生相关费用。详细计费说明，请参见 日志服务计费。
> - 采集容器日志时，系统自动创建**日志集**。该操作可能会导致使用过程中超过日志服务的日志集配额，目前日志服务允许每个 accoutID 下最多可创建50个日志集，如需更多，提工单申请。

日志类型 容器引擎CCE支持多种资源的日志采集，包括：控制面组件日志、业务日志、组件日志等。方便用户通过采集、查询日志，了解集群中基础资源、关键组件和应用的运行状态，当出现故障时进行排障。支持的采集对象如下表所示。

采集对象	说明
控制面组件日志	控制面组件日志主要记录CCE集群控制面主要组件的日志，方便您了解集群控制面的状态和记录。包括：kube-apiserver、kube-scheduler、kube-controller-manager。
业务日志	在容器引擎CCE控制台创建日志采集规则，通过指定集群中需要采集日志的容器实现对业务日志的采集、存储和查询分析。
组件日志	支持采集 CoreDNS 等组件的日志信息，系统会采集 CoreDNS 组件的标准输出日志，并发送至日志服务进行保存和检索。

日志采集流程

安装采集容器日志所需的必要组件，并在CCE控制台配置采集规则。此时采集规则会在日志服务控制台同步生成 传输任务，最后由日志服务采集容器日志。



在容器引擎控制台配置采集规则

您可以通过在容器引擎CCE控制台创建日志采集规则，通过指定集群中需要采集日志的容器实现对业务日志的采集、存储和查询分析。

使用限制

目前仅支持将采集的日志推送至日志集、BES，暂不支持直接推送至BOS、Kafka；日志源支持采集主机和容器日志，容器日志类型支持标准输出

日志、容器内部日志。

前提条件

- 已成功创建CCE集群。
- 已开通日志服务产品，您可以登录日志服务控制台确认是否开通。
- 已安装 log-operator 组件，详情请参见组件安装。

> 说明
>
> 若未安装组件，日志中心页面会提示您安装该组件。请根据界面提示完成 log-operator 组件的安装。

操作步骤

安装日志采集组件

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**组件管理**。
5. 在“组件管理”页面的“日志”页签下，找到CCE Log Operator组件，单击**安装**。

创建采集规则

1. 在集群管理页面左侧导航栏中选择 **运维与管理** > **日志中心**。
2. 在日志中心管理页面的“采集规则”页签下，单击“使用YAML创建”。

3. 在“创建采集规则页面”配置日志采集参数。

> **说明**
>
> - 当前在容器引擎控制台上仅支持自定义资源（CustomResourceDefinitions，CRD）的方式配置日志采集规则。
> - 日志采集规则CRD配置说明请见使用CRD配置采集规则。

4. 单击**确认**按钮，完成配置。

> **说明**
>
> - 每条CRD资源都会在日志服务BLS中生成一条传输任务（名称格式为：集群ID+命名空间名称+规则名称），建议您不要在日志服务控制台直接编辑对应的传输任务，日志采集组件会对配置进行覆盖。

管理采集规则

1. 在集群管理页面左侧导航栏中选择监控日志 > 日志中心，选择**采集规则**页签，即可查看采集规则。
2. 单击采集规则列表中**操作**列的**修改**或**删除**，允许更新或删除采集规则。

> **说明**
>
> 删除已有的日志采集规则需要您勾选是否同步删除采集规则关联的BLS日志集资源，若不勾选，系统则不会同步删除关联的BLS日志资源。若勾选，则删除日志采集规则后，系统会同步删除关联的BLS日志资源，请谨慎操作。

使用CRD配置采集规则

操作场景 您不仅可以控制台配置日志采集，还可通过自定义资源（CustomResourceDefinitions，CRD）的方式配置日志采集。CRD支持采集容器标准输出、容器内部日志，日志源支持采集主机和容器日志，支持多种日志采集格式，支持投递到 BLS 和 BES等不同消费端。 **前提条件**

- 已成功CCE集群。
- 已开通日志服务产品，您可以登录日志服务控制台确认是否开通。

CRD 介绍 CRD 定义的各个字段的具体含义，如下

CRD 资源类型

- LogConfig

LogConfig 定义

字段	描述
apiVersion *string*	cce.baidubce.com/v1
kind *string*	LogConfig
metadata *ObjectMeta*	详情看 Kubernetes API metadata.
spec *LogConfigSpec*	详见下文
status *LogConfigStatus*	采集任务的状态和关联采集器的ID

LogConfigSpec 定义

字段	描述
srcConfig	详见下文
dstConfig	详见下文

SrcConfig 定义

字段	描述
srcType *SrcType*	srcType, 可选值 host, container，必填
logType *ContainerLogType*	logType 只有 srcType 字段为 container 时，logType 才有用 可选值 stdout，internal，分别表示标准输出日志和容器内部日志，必填
srcDir *string*	srcDir 日志采集目录，注意是目录，当 srcType=container,logType=stdout 时 不需要指定 srcDir，选填
matchPattern *string*	matchPattern srcDir 下日志文件匹配规则，选填
ignorePattern *string*	ignorePattern srcDir 下日志文件就忽略规则，选填
timeFormat *string*	timeFormat 用于投 BOS 时，原文件路径日期解析，选填
ttl *integer*	ttl agent 采集日志时间范围，单位为天，必填
useMultiline *boolean*	useMultiline 是否启用多行模式，选填
multilineRegex *string*	multilineRegex 多行模式首行模式，选填
recursiveDir *boolean*	recursiveDir 是否递归采集 srcDir 下满足 matchPattern 的所有文件，包括子目录，选填
logTime *LogTimeType*	logTime 日志时间，可选值 system, logTime, 分别表示使用系统时间和使用日志时间，选填
timestampKey *string*	timestampKey 指定解析后的字段作为日志时间，选填
dateFormat *string*	dateFormat 指定时间戳字段的时间解析格式，format格式参考https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'Z" ，选填
filterExpr *string*	filterExpr 日志匹配表达式，符合规则的日志，将被采集，选填
processType *ProcessType*	processType 解析类型，选填
processConfig *ProcessConfig*	processConfig 对应 processType 的 process 参数，详见下文，选填
matchLabels *KVPair array*	matchLabels 容器环境匹配 labels，这里的 label 对应 docker inspect 上的 label, 不是 pod 上的 label，选填
ignoreLabels *KVPair array*	ignoreLabels 容器环境忽略 labels，具体同上，选填
matchEnvs *KVPair array*	matchEnvs 容器环境匹配的 env，选填
ignoreEnvs *KVPair array*	matchEnvs 容器环境忽略匹配的 env，选填

> matchLabels/matchEnvs 两者必须要填一个

DstConfig 定义

设置日志数据投递目的端。产品当前提供日志集、BES两种目的端，各端对应的具体参数配置分别如下：

设置**日志集**作为目的端

字段	描述
dstType *DstType*	dstType 目的端存储类型，当前支持 BLS，必填
logStore *string*	logStore BLS 日志集名字，必填
retention *integer*	retention 日志存储时间，输入范围 1-90，单位为 天，必填
rateLimit *integer*	日志上传带宽限制，输入范围 1-100 ，单位为 MB，必填

设置BES（Elasticsearch）作为目的端

字段	描述
dstType *DstType*	dstType 目的端存储类型，当前支持 BES，必填
besClusterID *string*	BES 集群ID，需要在同一地域，必填
besUser *string*	BES 集群的登录用户名，必填
besPasswd *string*	BES 集群登录密码，必填
besIndexPrefix *string*	自定义 indes 前缀。index Rolling 开启状态下，BES 集群中的 index 名称由“index前缀+采集日期”组成，关闭状态下，index 名称由 index 前缀组成。采集日期是指数据写入 BES 时的日期，日期格式为：YYYY-MM-DD，必填
besIndexRolling *string*	设定 BES 集群自动生成新 index 的频率。可选值为 none（不滚动生成 index）；day（每天0时生成）；week（每周一0时生成）；month（每月1号0时生成），必填
besIsPwChange *boolean*	仅支持输入 true，必填

ProcessConfig 定义

字段	描述
regex *string*	regex 正则规则 ，选填
separator *string*	separator 分隔符，选填
quote *string*	quote 分隔符场景可指定引用符，可选值包括：空，双引号"，单引号'和自定义 ，选填
sampleLog *string*	sampleLog 解析日志样例, 解析后用于在 console 配置 keys 与 dataType ，选填
keys *string*	keys 解析结果的列名；@message 为系统保留字，不允许设置为 key，选填
dataType *string*	dataType 解析结果每列对应的数据类型，支持string/int/float/bool, 必须要与 keys 一一对应，选填
discardOnFailure *boolean*	discardOnFailure 日志解析失败是否丢弃 true: 丢弃 false: 返回原值，选填
KeepOriginal *boolean*	keepOriginal 是否保留原日志 true: 保留原日志到 kafka 中的 @message 字段，bls 日志集中的 @raw 字段 false: 解析成功则不保留原日志，选填

CRD 示例

- **注意：由于BLS的限制，当前默认单节点同时处理的日志文件数量上限为10**

配置容器标准输出 CRD 示例 采集所有 namespace 下所有容器

```
apiVersion: cce.baidubce.com/v1
kind: LogConfig
metadata:
  name: all-container-stdout-log
  namespace: kube-system
spec:
  srcConfig:
    srcType: container
    logType: stdout
    ttl: 3
    matchLabels:
      - key: io.kubernetes.container.name
        value: .*
      - key: io.kubernetes.pod.namespace
        value: .*
  dstConfig:
    dstType: BLS
    logStore: container-stdout-log
    retention: 10
    rateLimit: 10
```

采集指定 namespace 下所有容器

```
apiVersion: cce.baidubce.com/v1
kind: LogConfig
metadata:
  name: kube-system-container-stdout-log
  namespace: kube-system
spec:
  srcConfig:
    srcType: container
    logType: stdout
    ttl: 3
    matchLabels:
      - key: io.kubernetes.container.name
        value: .*
      - key: io.kubernetes.pod.namespace
        value: kube-system
  dstConfig:
    dstType: BLS
    logStore: kube-system-container-stdout-log
    retention: 10
    rateLimit: 10
```

采集指定 namespace 指定容器

```
apiVersion: cce.baidubce.com/v1
kind: LogConfig
metadata:
  name: kube-state-metrics-log
  namespace: kube-system
spec:
  srcConfig:
    srcType: container
    logType: stdout
    ttl: 3
    matchLabels:
      - key: io.kubernetes.container.name
        value: kube-state-metrics
      - key: io.kubernetes.pod.namespace
        value: kube-system
  dstConfig:
    dstType: BLS
    logStore: kube-state-metrics-log
    retention: 10
    rateLimit: 10
```

配置容器文件路径 CRD 示例

采集指定 namespace 指定容器内部文件路径 注意：容器内文件路径，如 /usr/local/tomcat/logs/ 需要 mount emptyDir

```
apiVersion: cce.baidubce.com/v1
kind: LogConfig
metadata:
  name: tomcat-log
  namespace: default
spec:
  srcConfig:
    srcType: container
    logType: internal
    srcDir: /usr/local/tomcat/logs/ # 容器内文件路径，注意 /usr/local/tomcat/logs/ 需要 mount emptyDir
    matchPattern: catalina.*.log # 容器内日志文件名（支持正则）
    ttl: 3
    matchLabels:
      - key: io.kubernetes.container.name
        value: tomcat
      - key: io.kubernetes.pod.namespace
        value: default
  dstConfig:
    dstType: BLS
    logStore: tomocat-log
    retention: 10
    rateLimit: 10
```

Tomcat deployment 参考示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: "tomcat:7.0"
        volumeMounts:
        - name: tomcat-log
          mountPath: /usr/local/tomcat/logs # 挂载 emptyDir
      volumes:
      - name: tomcat-log  # 日志
        emptyDir: {}
```

配置节点文件路径 CRD 示例 采集节点上指定目录下日志文件

```
apiVersion: cce.baidubce.com/v1
kind: LogConfig
metadata:
  name: host-messages
  namespace: kube-system
spec:
  srcConfig:
    srcType: host
    srcDir: /logbeat_host/var/log # 注意，host 日志需要添加 /logbeat_host 前缀
    matchPattern: messages # 采集 message 日志文件
    ttl: 3
  dstConfig:
    dstType: BLS
    logStore: host-messages
    retention: 10
    rateLimit: 10
```

查看集群控制面日志

容器引擎CCE支持通过日志服务BLS采集托管集群中控制面组件日志。本文将介绍如何通过CCE控制台开启并查看集群控制面组件日志。

独立集群控制面组件日志您可以通过配置采集规则获取日志信息。

> 说明
>
> 开启控制面组件日志，会产生相关费用。详细计费说明，请参见 日志服务产品计费。

控制面组件介绍

目前支持采集如下控制面组件：

组件	说明
kube-apiserver	kube-apiserver组件暴露Kubernetes API接口的控制面组件。任何资源的请求和调用都是通过该组件进行的。详情请见官方文档。
kube-controller-manager	kube-controller-manager组件是集群的管理和自动化控制中心，负责管理集群中众多类型的控制器，例如：Node Controller（节点控制器）、Replication Controller（副本控制器）等。详情请见官方文档。
kube-scheduler	kube-scheduler 组件主要负责整个集群资源的调度功能。详情请见官方文档。

前提条件

- 已安装日志采集组件CCE Log Operator，详情请参考CCE Log Operator组件介绍。

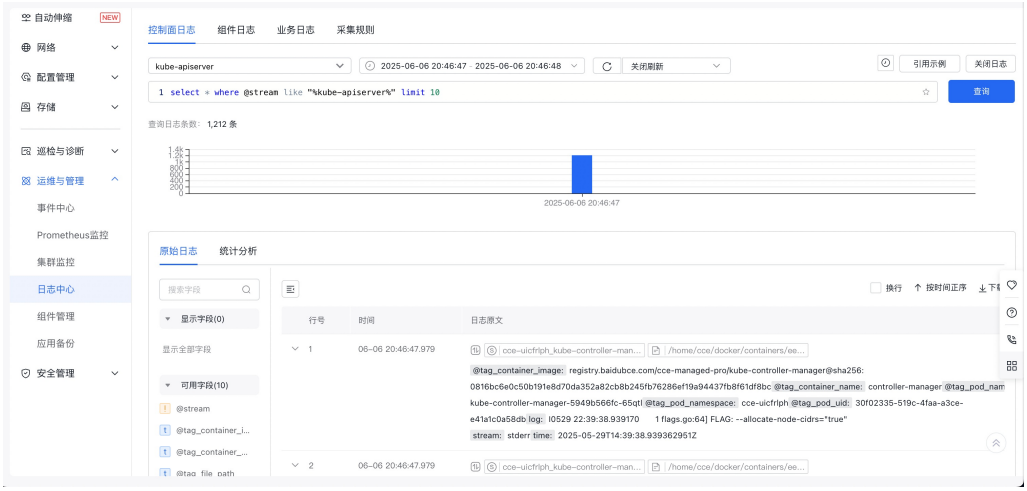
操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择**运维与管理** > **日志中心**。
5. 单击“控制面日志”进入控制面日志页面。

6. 单击**立即开启**，等待成功开通后即可。

> **说明**
>
> 开启采集控制面组件日志，系统会在您的账号下在BLS服务中自动创建{集群ID-k8s-log日志集。 **查看日志** 控制面组件日志采集成功开启后，可以在配置界面中查看采集到的日志信息。

7. 在**日志中心**页面中，选择**控制面日志**页签。
8. 在组件选择下拉菜单中，选择需要查看日志的控制面组件，即可查看到相关的日志信息。



> **说明**
>
> 查看【原始日志】需要为日志集**配置索引**，您可以在日志服务控制台进行索引配置，具体可参考索引配置。

关闭日志 您可以选择关闭收集控制面组件日志功能。

1. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**日志中心** > **控制面日志**。
2. 在控制平面组件日志页签，单击**关闭日志**。

> **说明**
>
> 关闭日志后，对应的日志集和采集的日志信息不会被删除。如需删除，请在日志服务控制台手动删除。

查看业务日志

容器引擎CCE支持通过日志服务BLS采集集群中容器日志，您可以在BLS控制台或CCE控制台上查询和分析日志信息。本文将介绍如何通过CCE控制台查询日志。

使用限制

目前仅支持将采集的日志推送至日志集BLS，暂不支持直接推送至BOS、Kafka、BES；日志源支持采集主机和容器日志，容器日志类型支持标准输出日志、容器内部日志。

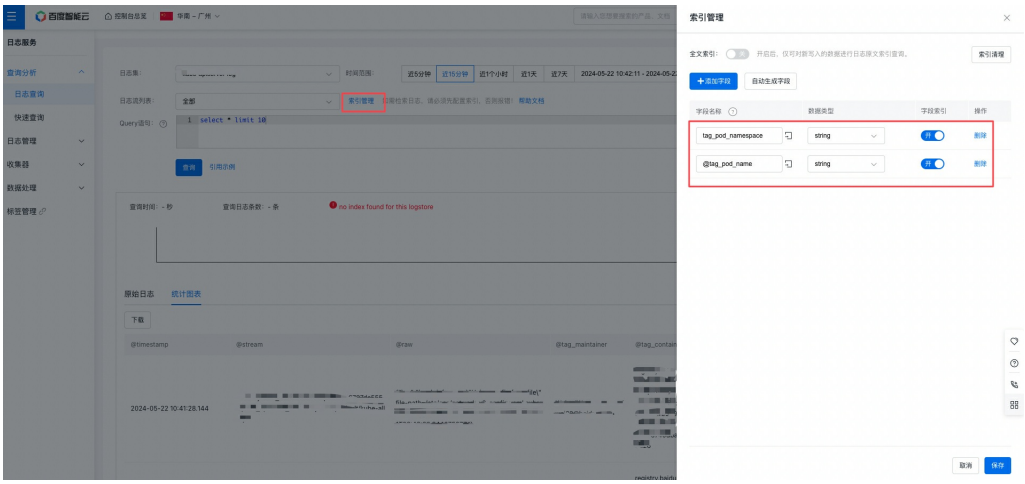
前提条件

- 已创建日志采集规则，具体操作请参见在容器引擎控制台配置采集规则。

操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择 **运维与管理** > **日志中心**。
5. 在日志中心管理页面的“业务日志”页签下，选择您要查看的日志集，输入查询内容，选择查询时间范围，单击**查询**，查看容器日志。

> **说明**
>
> - 容器业务支持通过命名空间、工作负载进行搜索查询，但需要为日志集**配置索引**，您可以在日志服务控制台进行索引配置。点击**索引管理**后，手工添加字段，并按照图中展示设置索引字段名称和数据类型。
> - 开启索引配置后，请不要轻易修改索引字段，若更换索引字段会导致无法按照命名空间、工作负载等关键字进行快速查询容器日志。



使用Prometheus监控集群

监控概述

CCE集群支持快速启用百度智能云 Prometheus 监控服务(CProm)，以实时监控集群和容器的健康状况，并查看可视化的 Grafana 监控数据大盘，您还可按需配置联系人接收监控报警、配置采集任务抓取自定义的监控指标等。

百度智能云 Prometheus 监控介绍

百度智能云 Prometheus 监控服务致力于提供轻量、稳定、高可用的服务。保留原生 Prometheus 的特性，支持采集自定义指标，支持千万级指标上报，提供基于 Grafana 的优秀可视化能力和预置默认面板，提供稳定的告警能力。 与自建 Prometheus 环境相比，您无需手工下载、配置 Prometheus 环境，无需考虑维护海量数据的存储成本。同时，Prometheus 监控兼容开源 Prometheus 的接口以及社区 Exporter 生态，提供免搭建、免运维的 Prometheus 集成环境，减少开发及运维成本。

> **说明**
>
> - 百度智能云 Prometheus 的详细介绍，请参见什么是Prometheus监控。
> - 启用 Prometheus 后，Prometheus 监控将采集集群中的容器监控指标。默认采集的指标均为百度智能云 Prometheus 基础指标，在默认情况下不会产生费用。但是，若您调整了指标存储时长，或上报了自定义指标，将会产生额外计费。关于百度智能云 Prometheus 的计费说明，请参见计费说明。

接入 Prometheus 监控实例

前提条件

- 已开通容器引擎CCE、百度智能云 Prometheus 服务，您可以登录Prometheus服务控制台确认是否开通。
- 已成功创建CCE集群，详情请参见创建CCE集群。

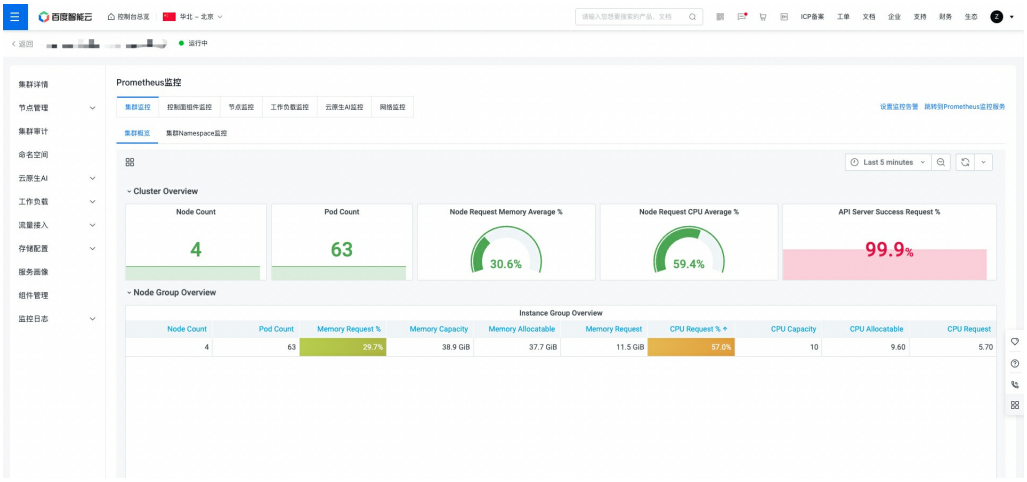
CCE集群接入 CProm 监控实例

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择**集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择 **运维与管理** > **Prometheus 监控**。
5. 单击**立即开启**，弹出接入 CProm 监控弹窗，您可以选择接入与 CCE 集群相同地域中已有的监控实例，也可以选择新建监控实例，点击**确认**，完成配置。
6. 接入 CProm 监控实例后，控制台会自动安装组件、检查监控大盘。安装完成后，您可以单击各个页签查看相应监控数据。

说明

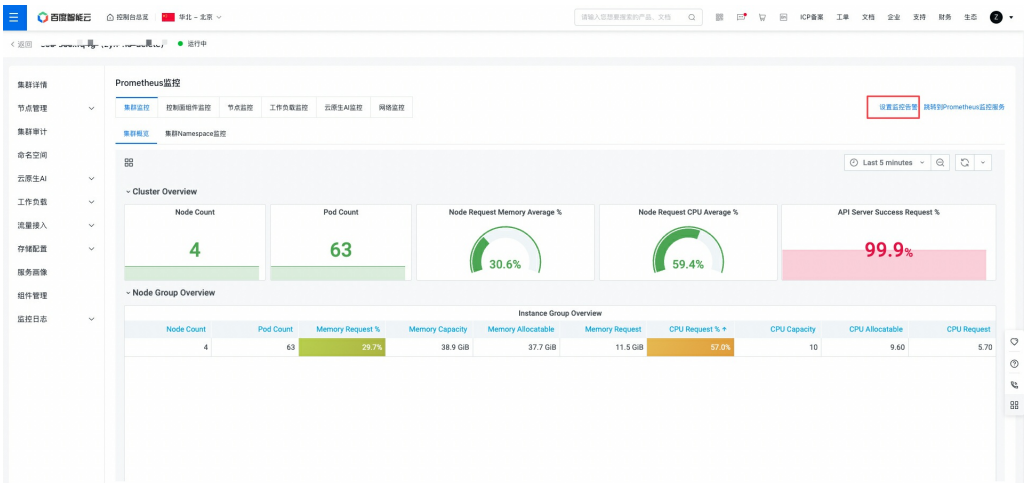
- 接入 CProm 监控实例，系统会默认在 CCE 集群安装监控组件。该组件用于采集 CCE 集群的各项监控指标，安装该组件会占用 CCE 集群一定资源，建议集群预留0.2核200Mi以上的资源，防止因资源不足导致实例接入失败。
- 您也可以选择登录Prometheus服务控制台选择 CProm 监控实例关联 CCE 集群，详情参考CProm监控实例关联集群。
- 接入 CProm 监控实例后会默认开启采集规则，如果您想要采集自定义业务指标，支持通过创建自定义采集任务，来满足您采集自定义指标需求，详情请参见采集配置。

查看监控面板 在容器引擎控制台的 Prometheus 监控页面，切换页签选择需要查看系统预置的 Grafana 监控大盘，获取相应的监控数据。监控面板支持设置查询的时间段，并指定刷新方式（手动刷新、自动刷新）。



配置 Promethues 监控告警 您可以为监控指标创建告警，在满足告警条件时通过邮件、短信、钉钉等渠道实时报警，主动帮助您发现异常。您可以在托管 Prometheus 的告警中心配置集群相关告警。

- 在监控大盘中点击设置监控告警，跳转至 Prometheus 监控服务为关键指标设置告警策略。
- 使用预置告警模版或者自定义，配置告警策略，详情请参见告警配置。



CCE 事件中心

概述

Kubernetes Events 包括了 Kubernetes 集群的运行和各类资源的调度情况，对维护人员日常观察资源的变更以及定位问题均有帮助。Kubernetes 默认只提供 1 小时的 Events 信息。CCE 支持为您的所有集群配置事件持久化功能，开启本功能后，会将您的集群事件实时导出到配置的存储端，支持查询最长 7 天之内的事件数据。本文介绍如何通过 CCE 控制台查看事件数据。

使用说明

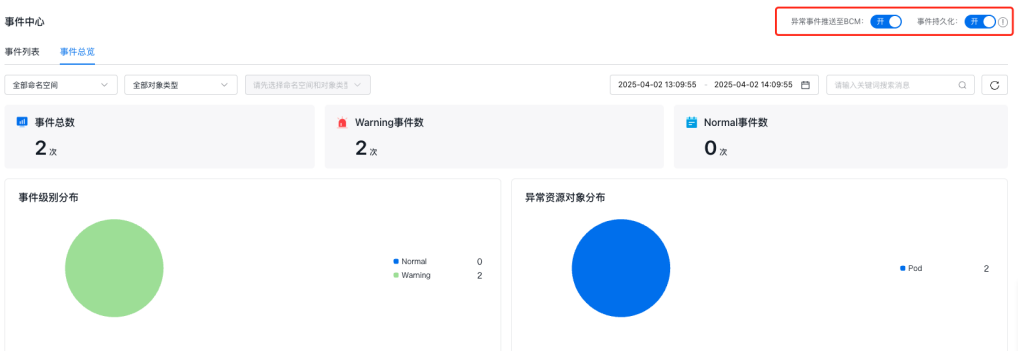
1. 登录百度智能云 容器引擎 CCE 控制台，在左侧导航栏选择**集群管理 > 集群列表**。
2. 在**集群列表**页面，单击目标集群名称进入集群详情页，然后在左侧导航栏，选择 **运维与管理 > 事件中心**。
3. 在事件中心的事件列表中，默认可查询最近 1 小时内产生的事件。可通过列表上方的下拉列表筛选与特定命名空间和资源对象相关联的事件。您可以通过事件列表查看事件详情，事件列表中每一列的含义如下：

列名	说明
事件类型	Normal 或者 Warning
对象	K8S 中的资源对象，如 Pod、Deployment 等
对象名称	对象名称
原因	原因概述，如镜像拉取失败、Mount 失败
消息	异常的具体原因
总数	重试次数
最早出现时间	最早时间
最近出现时间	最近时间

4. 开启右上角的**事件持久化**开关，可查询 7 天内的事件，此外，您也可以点击页面顶部导航栏中的**事件总览**，进入集群事件总览仪表盘，通过仪表盘可以直观地查看事件的总数、不同级别的事件分布情况以及异常资源对象的分布等。

> **注意：** 开启事件持久化后，生效可能稍有延迟，请耐心等待。

5. 为了进一步提升监控效率，您还可以选择将异常事件推送至云监控 BCM，通过开启右上角的**异常事件推送至 BCM** 即可实现。如需查看这些异常事件或进行报警配置的自定义，请前往 云监控 BCM 的相关页面进行操作。



CCE 集群异常事件报警

CCE 集群异常事件报警服务：当 CCE 集群中发生异常事件（如节点 NotReady）时，能够通过云上 BCM 服务 实时向用户报警。

前提条件

- 已成功创建CCE集群。
- 已开通云监控BCM服务，您可以登录云监控控制台确认是否开通。

操作步骤

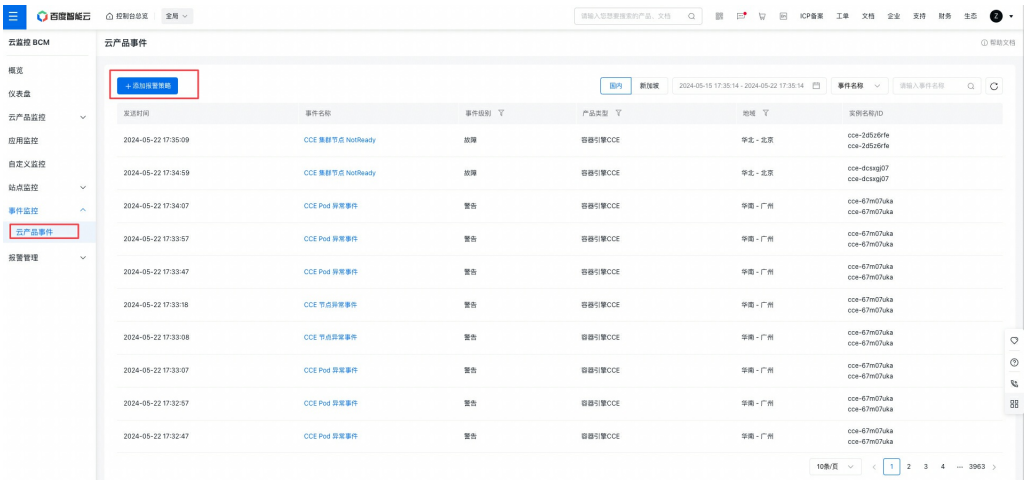
1. 登录容器引擎控制台。
2. 在左侧导航栏，选择集群列表。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。

4. 集群管理页面左侧导航栏中选择**运维与管理** >**事件中心**。

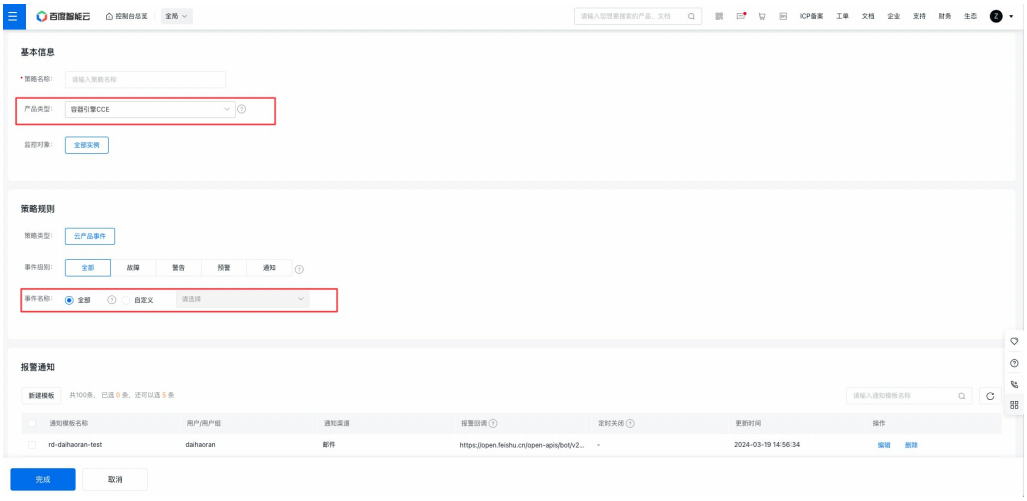
5. 单击右上角开启**异常事件推送至BCM**。

6. 点击**云监控服务**，跳转至云监控服务控制台配置CCE异常事件报警策略。

7. 在云监控BCM控制台左侧导航栏选择**事件监控** > **云产品事件**，点击左上角**添加报警策略**。

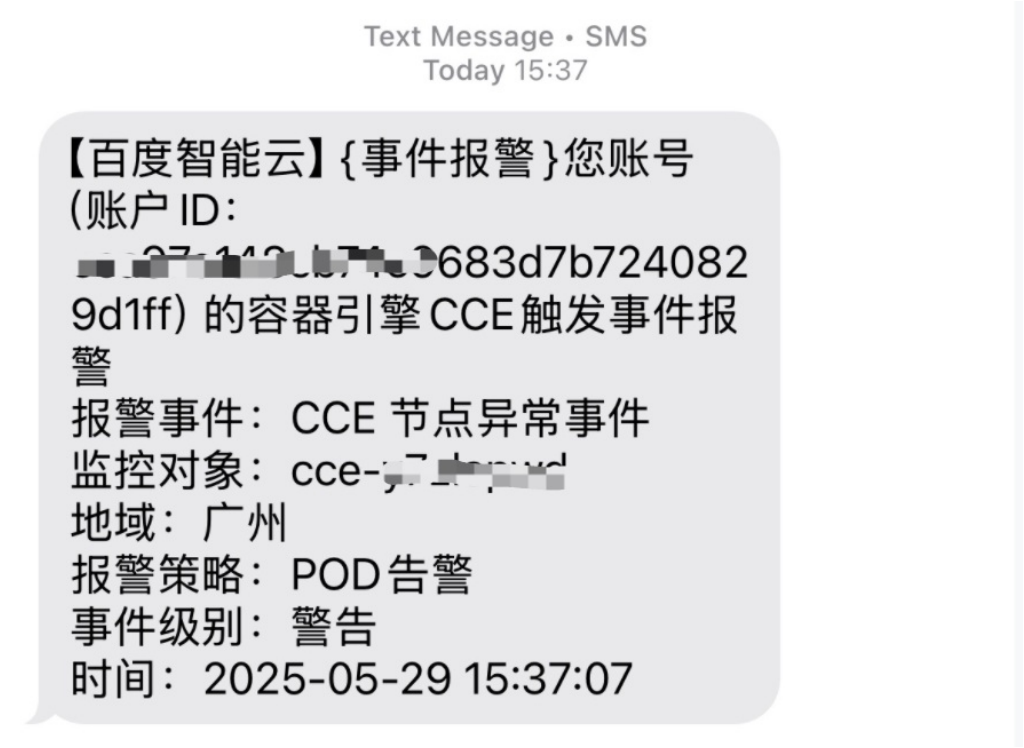


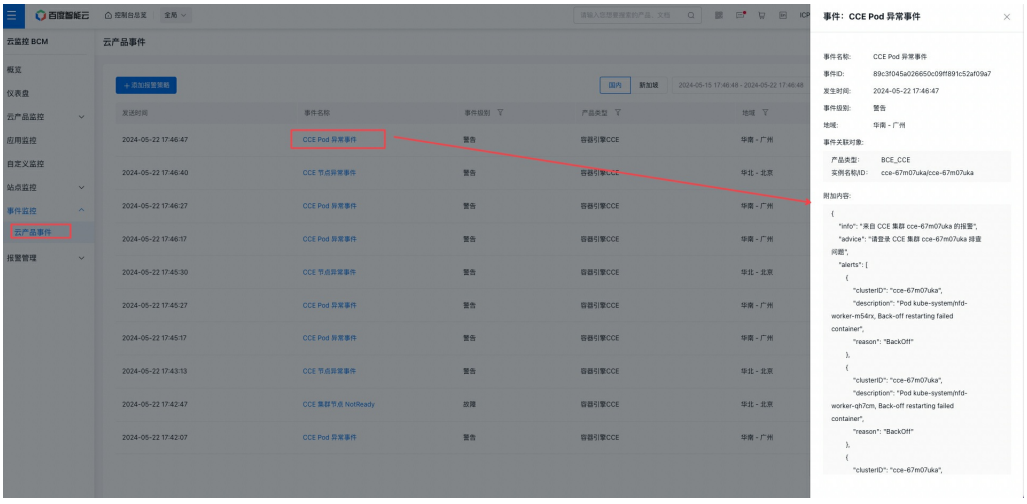
8. 创建报警策略，产品类型选择**容器引擎CCE**，按需配置告警信息、策略规则，并设置报警通知，详细操作请参考 报警管理。



报警通知

报警策略配置完成后，如果 CCE 集群出现对应的异常事件会收到报警（报警渠道取决于上面设置的报警动作，当前支持邮件、短信、电话报警）。BCM 事件监控页同样可以看到报警事件信息。





集群服务画像

概述

CCE 的服务画像可以查看集群内所有资源的特征和规范性，如镜像检查、应用检查、网络检查、安全检查，并给出对应的评分和详细内容，帮助

用户更好的了解集群内服务状况。

注意：服务画像的结果基于前一天的集群数据，且检查规则是一般判定标准，不一定适用于所有业务场景。

检查对象

检查的对象包括：

- 集群内的所有命名空间。
- 资源对象包括：Deployment、DaemonSet、StatefulSet、Service、CronJob、Job。

检查规则

一. 镜像检查

检查项	含义	影响级别
tagNotSpecified	镜像地址没有声明 tag或者 tag 为 latest	danger
pullPolicyNotAlways	镜像拉取策略不是 always	warning

解释：

- tagNotSpecified: 如果不声明镜像的版本(tag)，则默认为 latest。不指定 tag 可能会导致各种各样的问题，如同一时刻不同 pod 可能运行了不同的程序，尽管他们看起来镜像版本是相同的。另外，没有版本标识的镜像在发布时无法区分版本号，无法进行回滚操作。
- pullPolicyNotAlways: 依赖缓存的镜像可能会成为一个问题，尤其是配置了 latest 的 tag 时会直接使用本地缓存的镜像，不去拉取最新镜像。建议配置为 Always，不配置时默认就是 Always。

参考文档：https://vsupalov.com/docker-latest-tag/

错误示例：

```
containers:
- name: nginx
  image: hub.baidubce.com/cce/nginx-alpine-go:latest
  imagePullPolicy: Never
```

正确示例：

```
containers:
- name: nginx
  image: hub.baidubce.com/cce/nginx-alpine-go:v1
  imagePullPolicy: Always
```

二. 应用检查

检查项	含义	影响级别
cpuRequestsMissing	没有声明 cpu resource request 值	danger
cpuLimitsMissing	没有声明 cpu resource limit 值	danger
memoryRequestsMissing	没有声明 memory resource request 值	danger
memoryLimitsMissing	没有声明 memory resource limit 值	danger
readinessProbeMissing	没有声明 readinessProbe	danger
livenessProbeMissing	没有声明 livenessProbe	danger

解释：

- 为运行在 Kubernetes 中的容器配置资源请求(request)和限制(limit)是一个典型实践。设置适当的 request 将确保所有应用程序都有足够的计算资源，设置适当的 limit 将确保应用程序不会占用太多资源。
- readinessProbe 和 livenessProbe 可以帮助维护运行在 Kubernetes 中应用程序的健康状态。默认情况下 Kubernetes 只知道进程是否正在运行，而不知道它是否正常。正确配置 readiness 和 liveness 探针还可以确保应用程序的运行状况。

参考文档：

- https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
- https://cloud.google.com/blog/products/gcp/kubernetes-best-practices-resource-requests-and-limits
- https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

错误示例：

```
没有配置 resources、livenessProbe、readinessProbe
```

正确示例：

```
  resources: # 资源限制
   limits:
     cpu: 250m
     memory: 512Mi
   requests:
     cpu: 250m
     memory: 512Mi
livenessProbe: # 健康检查：存活探针
   httpGet:
     path: /
     port: 80
   initialDelaySeconds: 20
   timeoutSeconds: 5
   periodSeconds: 5
readinessProbe: # 健康检查：就绪探针
   httpGet:
     path: /
     port: 80
   initialDelaySeconds: 5
   timeoutSeconds: 1
   periodSeconds: 5
```

三. 网络检查

检查项	含义	影响级别
hostNetworkSet	设置了 hostNetwork	warning
hostPortSet	设置了 hostPort	warning

解释：

- hostNetworkSet: 尽管 Kubernetes 允许部署一个可以访问主机网络名称空间的 pod，但这并不是一个好的实践。在启用 hostNetwork 属性的情况下运行的 pod 将可以访问环回设备、监听本地主机的服务，并可用于窥探同一节点上其他 pod 的网络活动。一些特殊的系统组件可以将 hostNetwork 设置为 true，如 node-exporter 。
- hostPortSet: 在容器上设置 hostPort 属性将确保它可以在部署到的每个节点上的特定端口上访问。但如果指定了 hostPort 将限制在集群中调度 pod 的能力。

参考文档：https://kubernetes.io/docs/concepts/configuration/overview/#services

错误示例：

```
spec:
hostNetwork: true

containers:
  - name: nginx
    ports:
      - containerPort: 80
        hostPort: 80
```

正确示例：

```
spec:
hostNetwork: false

containers:
  - name: nginx
    ports:
      - containerPort: 80
```

四. 安全检查

检查项	含义	影响级别
hostIPCSet	设置了 hostPID	danger
hostPIDSet	设置了 hostPID	danger
notReadOnlyRootFileSystem	没有设置文件系统为只读	warning
privilegeEscalationAllowed	允许了特权升级	danger
runAsRootAllowed	作为 root 账户执行	danger
runAsPrivileged	特权模式执行	danger
dangerousCapabilities	capabilities 中有危险选项如 ALL/SYS_ADMIN/NET_ADMIN	danger

解释：

在 Kubernetes 中保护工作负载是整个集群安全的一个重要部分，总体目标应该是确保容器以尽可能少的特权运行。这包括避免权限提升、不使用 root 用户运行容器以及尽可能使用只读文件系统。

参考文档：https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

错误示例：

```
spec:
hostPID: true
hostIPC: true
containers:
  - name: nginx
    securityContext:
      capabilities:
        add: ["SYS_ADMIN"]
```

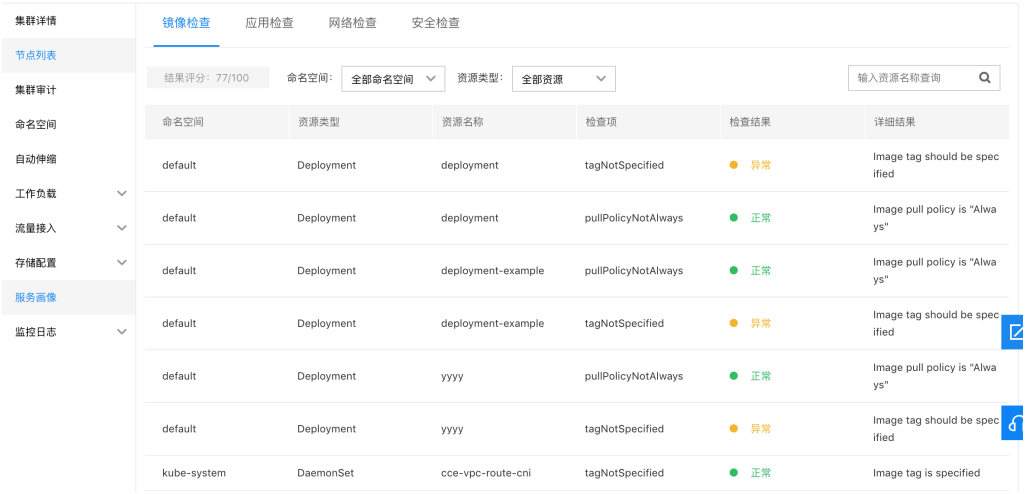
正确示例：

```
spec:
hostPID: false
hostIPC: false
containers:
  - name: nginx
```

计分规则

结果评分的计算方式为：检测通过数/检测总数 * 100

示例页面：



集群审计

概述

集群审计是基于 Kubernetes Audit 对 kube-apiserver 产生的可配置策略的 JSON 结构日志的记录存储及检索功能，支持通过 API Server 审计日志来记录用户在集群中的日常操作，帮助集群管理员对集群的操作行为进行安全审计，保证集群安全、稳定运行。本文为您介绍如何配置集群审计功能。

使用限制

- 本文仅适用于 CCE 托管集群、独立集群。

- 请确保您账号下 BLS 日志服务资源没有超出配额，否则会导致集群审计功能开启失败。
  - 目前BLS 日志服务允许每个 accoutID 下最多可创建50个日志集，如需更多，提工单申请。
- 当前新版集群审计由于受BLS 日志服务限制，不支持成都、香港、南京三个地域，因此这三个地域不支持集群审计功能升级，仍默认显示旧版集群审计页面。

前提条件

- 已成功 创建CCE集群。
- 已开通日志服务产品，您可以登录 日志服务控制台 确认是否开通。

计费说明

- 开启集群审计后，将采集集群审计日志投递到日志服务（BLS），日志服务会按照您的实际使用情况计费，计费标准请参考 计费概述。

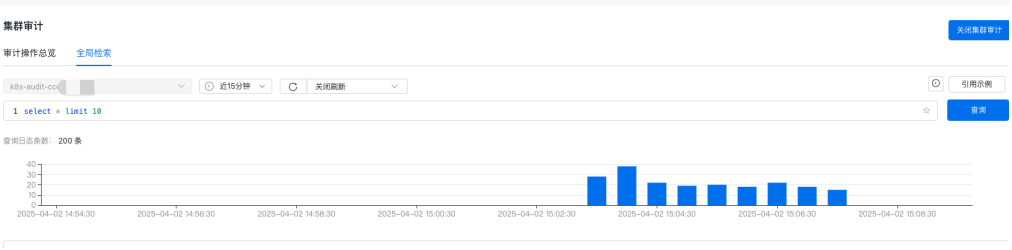
操作步骤

开启集群审计

1. 登录 容器引擎控制台。
2. 在左侧导航栏，选择 **集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择 **安全管理-集群审计**。
5. 在 **集群审计** 页面，针对未授权开启集群审计功能的集群，单击 **立即开启**；针对已授权开启过集群审计功能的集群，单击 **升级**，即可升级集群审计，将集群审计日志投递到 BLS 日志服务。

说明

- 开启集群审计功能，如果是独立集群，会重启集群的 API Server，导致 API Server 暂时无法访问，请勿频繁开启或关闭集群审计功能。
- 开启集群审计功能，容器服务会在您账号下自动创建的名称为k8s-audit-{集群ID}日志集中，帮助您更安全有效的运维集群自动创建日志集，并开启日志主题的索引配置。请勿修改索引，以免报表失效。 **查看审计日志** 集群审计功能配置完成后，您可以在**集群审计**页面中，查看集群审计日志信息。



说明

- 集群审计日志在BLS日志服务中对应的日志集中数据默认保存时间为180天。如需修改日志的默认保存时间，请参考文档 日志集。
- 集群审计日志在BLS默认开启ObjectRef、ResponseStatus、User、Verb、namespace、resource、userID作为索引字段，如果关闭默认索引字段，将影响审计日志查询。
- 如果您有自定义查询、分析审计日志的需求，可以进入日志服务控制台 查看详细的日志记录。

关闭集群审计 如果您不再需要集群API Server的审计功能的话，可以通过以下方法关闭审计功能。

1. 在集群审计页面中，单击右上角 **关闭集群审计**。
2. 弹出二次确认对话框，单击 **确认**，完成配置。

说明

- 关闭集群审计功能，会重启集群的 API Server，导致 API Server 暂时无法访问，请勿频繁开启或关闭集群审计功能。
- 关闭集群审计功能，不会删除 BLS 日志服务中的日志集，如需删除日志集，您可以登录 日志服务控制台 确认删除相应的日志集。

集群审计仪表盘

操作场景

容器引擎 CCE 为用户提供了开箱即用的审计仪表盘。在集群开启集群审计功能后，CCE 将自动为该集群配置审计操作总览仪表盘。同时内置 BLS 的全局检索，方便用户观测和检索各类集群操作，以便于及时发现和定位问题。CCE 集群审计操作总览仪表盘主要展示总审计记录数、操作用户数、活跃节点数、异常访问次数等。

前提条件

- 已成功 创建CCE集群。
- 已开通日志服务产品，您可以登录 日志服务控制台 确认是否开通。

说明

- 集群审计日志总览仪表盘，系统会自动为日志集配置索引，请不要修改索引字段，若更换索引字段会导致无法查看仪表盘数据。索引字段如下所示：

	命名空间	操作用户	状态码	操作类型	资源对象	资源类型
字段名	namespace	User.username	ResponseStatus.code	Verb	ObjectRef.Name	resource
索引	namespace	User	ResponseStatus	Verb	ObjectRef	resource

操作步骤

集群审计中配置了审计操作总览、全局检索。请按照以下步骤进入集群审计页面，开始使用对应功能：

1. 登录 容器引擎控制台。
2. 在左侧导航栏，选择 **集群列表**。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。

4. 在集群管理页面左侧导航栏中选择 **安全管理-集群审计**，开启集群审计功能，操作详情请参见 开启集群审计。

5. 进入审计操作总览页面，查看图表详情。

审计操作总览

当您想观测整个集群 APIServer 操作时，可在审计操作总览页面设置过滤条件，查看核心审计日志的汇总统计信息。例如，总审计日志的统计数、分布情况、重要操作趋势等。如下图所示：

您还可在该页面中查看更多统计信息，如下所示：

- 集群审计日志的统计数仪表盘：

- 分布情况仪表盘：

- 重要操作趋势仪表盘：

查询语句介绍

筛选条件说明 CCE集群审计操作总览仪表盘中的筛选条件说明如下所示：

- 命名空间，所关联的BLS的查询分析语句如下所示：

```
select distinct namespace
```

- 操作用户，所关联的查询分析语句如下所示：

```
select distinct User->"$.username"
```

- 状态码，所关联的查询分析语句如下所示：

```
select distinct `ResponseStatus`->"$.code"
```

- 操作类型，所关联的查询分析语句如下所示：

```
select distinct Verb
```

- 资源对象，所关联的查询分析语句如下所示：

```
select distinct ObjectRef->"$.Name"
```

- 资源类型，所关联的查询分析语句如下所示：

```
select distinct ObjectRef->"$.Resource"
```

重要图表说明

CCE日志审计中心仪表盘中重要图表说明如下所示：

- 总审计记录数，所关联的查询分析语句如下所示：

```
select count(1) as "记录数"
```

- 操作用户数，所关联的查询分析语句如下所示：

```
select count(DISTINCT User->"$.username") as "用户数"
```

- 活跃节点数，所关联的查询分析语句如下所示：

```
select count(DISTINCT User->"$.username") as "节点数" where (locate("system\:node\:", User->"$.username") = 1)
```

- 异常访问次数，所关联的查询分析语句如下所示：

```
select count(1) as "访问次数" where (User->"$.username" ="system:kube-scheduler") AND (ResponseStatus->"$.code" >= 400)
```

- 敏感操作次数，所关联的查询分析语句如下所示：

```
 select count(1) as "操作次数" from log where (Verb = 'create' AND (`ObjectRef`->'$.Subresource' = 'exec')) OR (Verb = 'create' AND (`ObjectRef`->'$.Subresource' = 'attach') AND (`ObjectRef`->'$.Resource' = 'pods')) OR (Verb = 'get' AND (`User`->"$.username" != 'apiserver') AND (locate('system:node', `User`->"$.username") != 1) AND (`ObjectRef`->'$.Resource' = 'secrets')) OR (Verb = 'delete' AND (locate('system:node', `User`->"$.username") != 1) AND ((locate('system:serviceaccount:kube-system', `User`->"$.username")) != 1) AND ((`User`->"$.username") != 'system:apiserver') AND (locate('system:kube-scheduler', (`User`->"$.username")) != 1) AND (locate('system:kube-controller-manager', `User`->"$.username") != 1))
```

- 创建操作次数，所关联的查询分析语句如下所示：

```
select count(1) as "操作次数" from log where Verb = 'create'
```

- 更新操作次数，所关联的查询分析语句如下所示：

```
select count(1) as "操作次数" where Verb in ('update','patch')
```

- 删除操作次数，所关联的查询分析语句如下所示：

```
select count(1) as "操作次数" where Verb = 'delete'
```

- 操作用户分布，所关联的查询分析语句如下所示：

```
select `User`->"$.username" as `用户名`,count(*) as cn group by `用户名` order by cn desc
```

- 命名空间分布，所关联的查询分析语句如下所示：

```
select namespace as `命名空间`,count(*) as cn group by namespace order by cn desc
```

- 资源类型分布，所关联的查询分析语句如下所示：

```
select resource as `资源类型`,count(*) as cn group by resource order by cn desc
```

- 操作类型分布，所关联的查询分析语句如下所示：

```
select Verb as `操作类型`,count(*) as cn group by Verb order by cn desc
```

- 状态码分布，所关联的查询分析语句如下所示：

```
select `ResponseStatus`->"$.code" as `状态码`,count(*) as cn group by `状态码` order by cn desc
```

- 节点操作分布，所关联的查询分析语句如下所示：

```
select Verb as `操作类型`, count(*) as cn where `resource` = 'nodes' group by `操作类型`, `resource` order by cn desc
```

- 工作负载操作分布，所关联的查询分析语句如下所示：

```
select Verb as `操作类型`, count(*) as cn where `操作类型` in ('create', 'delete') and `resource` in ('deployments','statefulsets','daemonsets','jobs','cronjobs') group by `操作类型`, `resource` order by cn desc
```

- Service/Ingress分布，所关联的查询分析语句如下所示：

```
select Verb as `操作类型`,count(*) as cn where `操作类型` in ('create', 'delete') and `resource` in ('ingressess','services') group by `操作类型`,`resource`order by cn desc
```

- 重要操作趋势，所关联的查询分析语句如下所示：

```
select histogram(cast(@timestamp as timestamp),interval 1 minute) as t, Verb, count(1) as "操作次数" where Verb in ('create','delete','update','patch')  group by t, Verb order by t limit 10000
```

- 非系统用户操作趋势，所关联的查询分析语句如下所示：

```
select histogram(cast(@timestamp as timestamp),interval 1 minute) as t, `User`->"$.username" as `用户名称`, count(1) as `count` where ((locate("system:", `User`->"$.username") = 0) AND ((`User`->"$.username") not like 'apiserver') AND ((`User`->"$.username") not like "kube-controller-manager") AND ((`User`->"$.username") not like "kube-apiserver-kubelet-client")) group by t, `用户名称` order by t limit 10000
```

Java应用监控

对于部署在CCE集群中的Java应用，通过安装APM应用监控组件CCE OnePilot，即可实现全链路智能监控能力。

背景信息

应用性能监控(Application Performance Management,APM)是一款应用性能管理平台，提供分布式应用拓扑查询和性能分析能力，可通过APM对应用的四个黄金指标（请求数，QPS，耗时，错误率)进行监控，并可通过Trace链路详情定位问题，全方位保障系统的可用性和稳定性。

前提条件

- 检查您的JDK版本。APM应用监控支持JDK1.8（含）以上版本
- 已开通APM

步骤一：安装 APM 应用监控组件

1. 登录容器引擎控制台。
2. 在左侧导航栏，选择集群列表。
3. 在“集群列表”页面单击目标集群，进入集群管理页面。
4. 在集群管理页面左侧导航栏中选择运维与管理-组件管理。
5. 在“组件管理”页面的“监控和日志日志”页签下，找到CCE Onepilot组件，单击安装。

安装完成后，您可以在组件管理页面升级或卸载CCE Onepilot组件。

步骤二：为 Java 应用开启 APM 应用监控

如需在创建新应用的同时开启ARMS应用监控，请完成以下操作。

1. 登录容器引擎控制台，在左侧导航栏选择集群列表。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择工作负载 > 无状态。

3. 在无状态页面单击使用YAML创建资源。

4. 选择自定义模板，并在模板（YAML格式）中将以下labels添加到spec.template.metadata层级下。

```
labels:
cceOnepilotEnable: "on"
apmServiceName: "<your_service_name>" # 请用您应用实例的名称替换 <your_service_name>，若不设置此标签，则会以工作负载的名称 metadata.name 上报指标至 APM。
```

创建一个无状态（Deployment）应用并开启APM应用监控的完整YAML示例模板如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-demo
  namespace: cce-onepilot
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-demo
  template:
    metadata:
      labels:
        app: test-demo
        cceOnepilotEnable: "on"
        apmServiceName: "test-demo"
    spec:
      containers:
      - name: test-demo
        image: registry.baidubce.com/cce-plugin-dev/tomcat:my-tomcat
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          hostPort: 8080
```

查看结果

在无状态页面上，目标应用的操作列将出现**APM控制台**按钮。

后续步骤

完成上述步骤后，您就为部署在容器服务Kubernetes版中的应用开启了APM应用监控。在目标应用的操作列中单击APM控制台，将进入APM控制台的应用监控页面。

权限管理

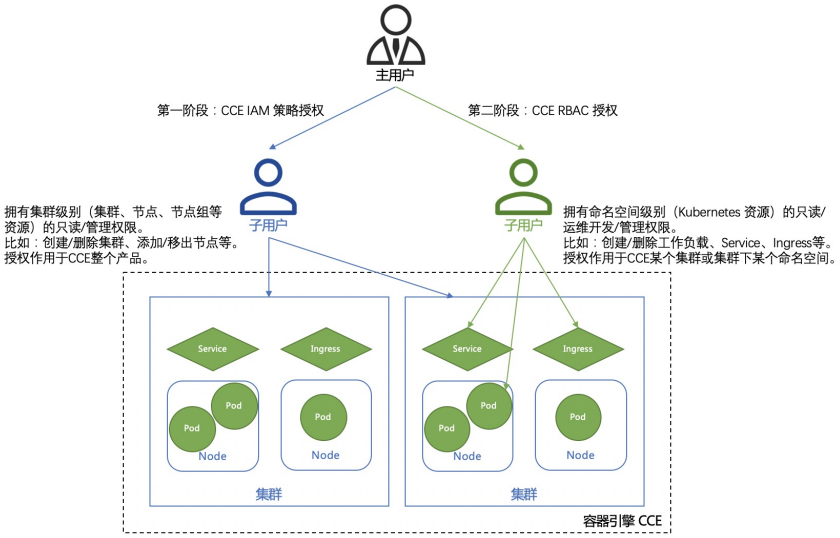
权限概述

注册百度智能云账号后，系统自动创建的超级管理员用户即为账号的主用户，默认拥有云账户的所有权限和管理所有云资源的权限。在多个人员共享资源和协作管理的场景下，为了保证账户和资源的安全，管理员可以创建子用户，实现不同子用户使用不同的用户名密码，以及拥有不同的云资源访问权限，更多信息请参见百度智能云账户身份体系。

默认情况下，子用户没有使用 CCE 的任何权限，因此，需要管理员授权允许子用户拥用相应的权限。

CCE 授权体系

CCE 权限管理，包括百度智能云多用户访问控制（IAM）策略授权和 Kubernetes 角色访问控制（RBAC）授权，支持从集群和命名空间级别对子用户/用户组进行细粒度的授权。CCE 的授权可分为两个使用阶段：



- 第一阶段是基于 IAM 策略的授权，管理员对 IAM 子用户/用户组授予 CCE 集群级别（集群、节点、节点组等资源）的只读/管理权限（比如创建/删除集群、添加/移出节点、创建/删除节点组等），授权将作用于 CCE 整个产品，更多信息请参见配置 IAM 预置权限策略。
- 第二阶段是基于 Kubernetes RBAC 的授权，管理员对 IAM 子用户授予 CCE 集群命名空间级别（Kubernetes 资源）的只读/运维开发/管理权限（比如创建/删除工作负载、Service、Ingress 等），授权作用于 CCE 某个集群或集群下某个命名空间，更多信息请参见配置预置 RBAC 权限策略。

配置IAM预置权限策略

CCE 基于 IAM 策略授权，指的是百度智能云账号的管理员（或拥有系统管理员权限的子用户）通过多用户访问控制（IAM），将预定义的 CCE 系统策略授予子用户/用户组，让其拥有集群级别（集群、节点、节点组等资源）的只读/管理权限，CCE 系统策略授权将作用于所有地域下的全部集群。

本文将介绍 CCE 基于 IAM 策略授权的流程及使用方法。

> **说明**
>
> 本节对于 CCE IAM 策略的授权，只包括 CCE 集群级别资源以及关联云产品/服务的权限，不包括集群命名空间级别 Kubernetes 资源的权限。对命名空间 Kubernetes 资源的操作授权，请参见配置预置RBAC权限策略 。

CCE预置策略

您可以使用以下预设策略为子用户授予相关权限：

策略名称	策略描述	权限范围
CCEDevelopPolicy	**CCE 开发管理策略：** 包括 CCE 集群、节点、节点组等集群级别资源的只读权限，以及 BCC、CDS、VPC、EIP、BLB、CCR 等相关云产品/服务的只读权限。不包括集群命名空间级别的权限。 对于需要对业务应用进行部署和变更的开发人员来说，授予该策略后，再通过 CCE RBAC 授权具体集群和命名空间的运维开发权限，即可进行创建工作负载等 Kubernetes 资源的操作。	**集群/节点资源操作：** 查看集群/节点/节点组的列表/详情 查看集群/节点健康检查 查看/下载集群kubeconfig 查看节点组伸缩配置/伸缩活动 查看集群快照列表 查看权限管理列表 查看组件介绍详情 查看集群/节点监控 查看集群/节点日志 查看集群/节点事件 查看Helm模板/Helm实例 查看/拉取镜像仓库镜像 查看镜像仓库命名空间 **关联云产品/服务操作：** 查看计算资源BCC、BBC，存储资源CDS、BOS，网络资源VPC、子网、安全组、BLB、EIP、ENI、ACL等，以及查看监控服务BCM、容器镜像服务CCR、ElasticSearch服务
CCEFullControlAccessPolicy	**CCE 完全控制管理策略：** 包括 CCE 集群、节点、节点组等集群级别资源的管理权限，以及 BCC、CDS、VPC、EIP、BLB、CCR 等相关云产品/服务的操作权限。不包括集群命名空间级别的权限。 对于需要创建、管理集群，扩缩容节点资源，对业务部署运行环境进行变更、升级、监控、维护的运维人员来说，授予该策略即可。	**集群/节点资源操作：** 创建/删除集群/节点组 添加/移出节点 升级集群版本 变更节点组配置 调整节点组节点数 创建/删除集群快照 安装/升级/卸载组件 配置容器监控/报警 创建/删除日志规则 开关事件持久化/异常事件推送 创建/更新/删除Helm实例 推送镜像到镜像仓库 创建/删除镜像仓库命名空间 **关联云产品/服务操作：** 运维操作计算资源BCC、BBC，存储资源CDS、BOS，网络资源VPC、子网、安全组、BLB、EIP、ENI、ACL等，以及操作容器镜像服务CCR、ElasticSearch服务

> **温馨提示**
>
> - CCE 的关联云产品/服务较多，为了让您的授权过程更加简单高效，CCE IAM 系统策略配置包含了常用关联云产品/服务的操作权限。
> - 您只需为子用户授权一次 CCE IAM 系统策略，即可让其拥有 CCE 集群、节点资源和关联云产品/服务的只读/管理权限，而无需理解多种云产品策略并进行多次授权。
> - 若系统策略不能满足您的授权需求，可以创建自定义策略允许或拒绝相关云产品/服务的操作。

操作步骤

直接给子用户授权

您可以直接为子用户关联策略以获取策略包含的权限。

1. 进入多用户访问控制控制台,选择左侧导航栏中的**用户管理>子用户**进入“子用户管理”列表页面。
2. 选择目标子用户单击**用户名**进入用户详情页面。

3. 单击“授权”操作，在“编辑权限”弹出框中勾选所需策略。
4. 单击“确定”即可。

通过用户组给子用户授权

您可以将子用户添加至用户组，该子用户将自动获取该用户组所关联策略的权限。如需解除随组关联策略，仅需将子用户移出相应用户组即可。

1. 进入多用户访问控制控制台,选择左侧导航栏中的**组管理**进入“组管理”列表页面。
2. 选择目标用户组单击**编辑**进入编辑组页面。
3. 在“编辑组”页面勾选所需策略以及添加对应子用户。
4. 单击“确认”即可。

配置IAM自定义权限策略

容器引擎CCE提供的系统授权策略的授权粒度比较粗，如果这种粗粒度授权策略不能满足您的需要，那么您可以创建自定义授权策略。例如，您想控制对某个具体的集群的操作权限，您可以使用自定义授权策略满足细粒度访问要求。

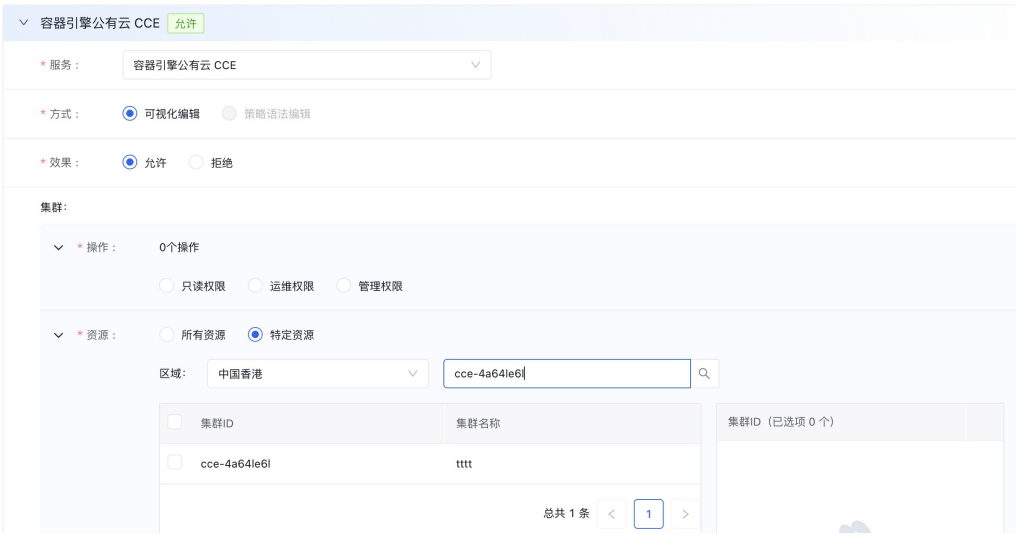
操作步骤

按策略生成器创建

1. 登录百度智能云官网，并进入管理控制台。
2. 在控制台右上角个人信息中单击**多用户访问控制**，进入“多用户访问控制”控制台页面。
3. 在左侧导航栏中选择“策略管理”进入“权限策略列表”页面。
4. 单击**创建策略**，在“选择创建策略的方式”弹出框中选择**按策略生成器创建**。

5. 在“在“创建权限策略”页面完成策略名称和权限配置。
6. 单击“添加权限”，在弹框中完成权限配置，这里以授予香港地域某个集群的只读权限为例：

权限配置



配置项	必选/可选	配置说明
选择服务	必选	选择生效的服务对象，这里选择“容器引擎CCE”。
权利效力	必选	选择允许操作或禁止操作，这里选择“允许”。
集群	必选	选择集群的操作权限，这里选择“只读”。
选择资源	必选	选择可操作的集群资源，这里选择“特定资源”，区域选择“中国香港”，然后勾选该地域下的集群。若想要操作全部集群，则选择“所有资源”。

> **说明**
>
> - 只读权限：拥有已选择标签关联全部集群的只读权限
> - 运维权限：拥有已选择标签关联全部集群的全部读写权限，但不能删除集群
> - 管理权限：拥有已选择标签关联全部集群的全部读写权限，可以删除集群
> - 若您有更高的配置要求，更多信息可参见管理IAM策略。

7. 单击**确认**按钮完成创建，您可以在“权限策略列表”页面查看到创建的策略。
8. 将自定义权限策略授权给目标IAM子用户，具体操作，请参见子用户授权。

按策略语法创建

在创建自定义授权策略时，您需要了解授权策略语言的基本结构和语法。更多信息，请参见策略语法说明。

1. 登录百度智能云官网，并进入管理控制台。
2. 在控制台右上角个人信息中单击**多用户访问控制**，进入“多用户访问控制”控制台页面。
3. 在左侧导航栏中选择“策略管理”进入“权限策略列表”页面。
4. 单击**创建策略**，在“选择创建策略的方式”弹出框中选择**按策略语法创建**。
5. 在“创建权限策略”页面编写自定义授权策略内容，当前以授予全地域所有集群的完全控制权限为例：

```
{
 "version": "v1",
 "accessControlList": [
    {
       "service": "bce:cce",
       "region": "*",
       "resource": [
          "*"
       ],
       "effect": "Allow",
       "permission": [
          "CLUSTER_CREATE",
          "CLUSTER_DELETE",
          "CLUSTER_MANAGE",
          "CLUSTER_READ",
          "CLUSTER_LIST"
       ]
    }
 ]
}
```

其中：

- region处为所要授权的地域，可以有以下配置：
  - 授予全部地域的权限

```
"region": "*"
```

  - 授予单个地域的权限，如：北京（bj）

```
"region": "bj"
```

> **说明**
>
> 一个accessControl只能指定一个地域，如需指定多个地域，则需创建多个。

- resource处为所要授权的资源，可以有以下配置：
  - 授予全部集群的权限

```
"resource": [
        "*"
    ]
```

  - 授予单个集群的权限

```
"resource": [
        "CCE_Cluster/集群ID"
    ]
```

  - 授予多个集群的权限

```
"resource": [
        "CCE_Cluster/集群ID",
        "CCE_Cluster/集群ID"
    ]
```

> **说明**
>
> 集群ID需要替换为所需授权的真实的集群ID。

- action处为所要授予的集群操作权限：
  - 您可以根据实际情况选择不同的权限组合

```
    "permission": [
            "CLUSTER_CREATE", # 授予创建集群的权限
            "CLUSTER_DELETE", # 授予删除集群的权限
            "CLUSTER_MANAGE", # 授予管理集群的权限
            "CLUSTER_READ",  # 授予只读集群的权限
            "CLUSTER_LIST"   # 授予集群列表的权限
        ]
```

6. 单击**完成**即可完成创建自定义策略。

7. 将自定义权限策略授权给目标IAM子用户，具体操作，请参见子用户授权。

自定义授权策略示例

以下策略中不包含集群关联使用云产品（如云服务器BCC、负载均衡BLB等）的权限，若有需要请单独为子用户授予相应产品的权限。

- 授权子用户单个集群的只读权限，如集群ID为cce-xxxx

```
{
  "version": "v1",
  "accessControlList": [
    {
      "service": "bce:cce",
      "region": "*",
      "resource": [
        "CCE_Cluster/cce-xxxx"
      ],
      "effect": "Allow",
      "permission": [
        "CLUSTER_READ",
        "CLUSTER_LIST"
      ]
    }
  ]
}
```

- 授权子用户单个集群的完全控制权限，如集群ID为cce-xxxx

```
{
  "version": "v1",
  "accessControlList": [
    {
      "service": "bce:cce",
      "region": "*",
      "resource": [
        "CCE_Cluster/cce-xxxx"
      ],
      "effect": "Allow",
      "permission": [
        "CLUSTER_DELETE",
        "CLUSTER_MANAGE",
        "CLUSTER_READ",
        "CLUSTER_LIST"
      ]
    }
  ]
}
```

配置IAM标签权限策略

CCE集群绑定云资源标签后，您可以使用标签为集群资源进行分配并控制访问权限。本文将介绍如何通过标签控制IAM子用户的权限，使得不同的子用户拥有不同CCE集群的访问权限。

背景信息

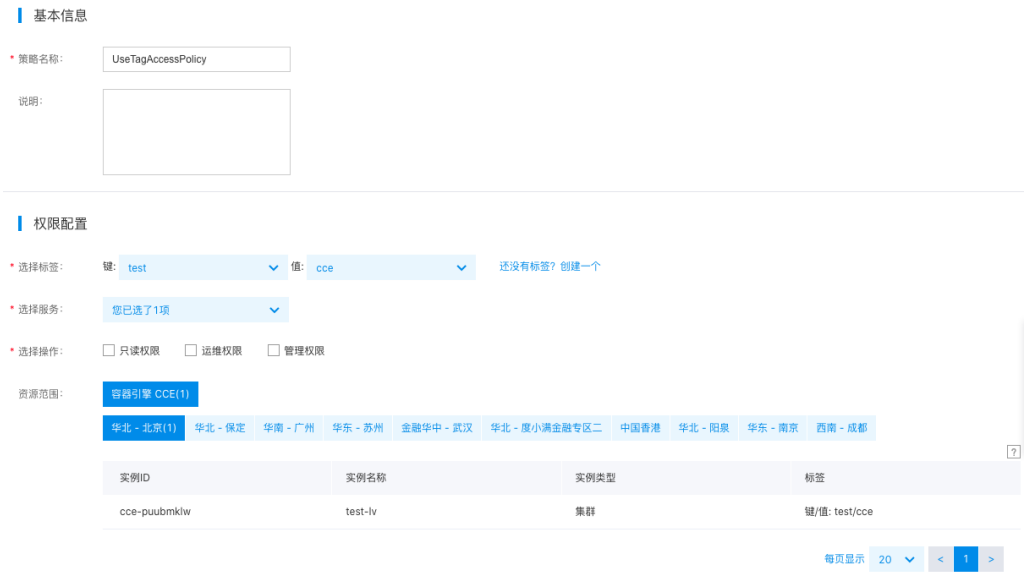
标签可用于标识云资源，实现资源的分类管理；多用户访问IAM可基于权限策略，管理用户身份，控制云资源的访问和操作权限。标签和IAM结合，将标签作为权限策略的匹配条件，可以实现云资源精细化权限管理。

操作步骤

本步骤将使用百度智能云主账号新建一个自定义权限策略UseTagAccessPolicy（规定了IAM子用户只能访问指定标签 test：cce 的CCE集群），并将自定义权限策略UseTagAccessPolicy授予IAM子用户。

1. 进入多用户访问控制控制台。
2. 单击左侧导航栏中**策略管理**进入“权限策略列表”页面。
3. 单击**创建策略**，在系统弹框中的选择**按标签创建**进入按标签创建权限策略页面。

4. 在标签创建权限策略页面完成相关配置：



配置项	必选/可选	配置说明
策略名称	必选	自定义策略名称，这里输入UseTagAccessPolicy。
选择标签	必选	选择目标资源所绑定的标签，这里选择 test：cce 。
选择服务	必选	选择目标资源对应的云服务，这里选择“容器引擎CCE”，系统将自动筛选出该标签所关联的全部CCE集群 。
选择操作	必选	选择子用户可操作目标资源的权限，这里选择“只读权限”，支持多选“运维权限”及“管理权限” 。

> **说明**
>
> - 只读权限：拥有已选择标签关联全部集群的只读权限
> - 运维权限：拥有已选择标签关联全部集群的全部读写权限，但不能删除集群
> - 管理权限：拥有已选择标签关联全部集群的全部读写权限，可以删除集群

5. 单击**确认**按钮完成创建，您可以在“权限策略列表”页面查看到创建的策略。

6. 将自定义权限策略授权给目标IAM子用户，具体操作，请参见子用户授权。

配置预置RBAC权限策略

CCE提供了对接Kubernetes RBAC的授权模式，便于对子用户进行细粒度的访问权限控制。本文介绍如何为子用户配置RBAC权限，实现对CCE集群和命名空间的权限控制。

公告

CCE集群服务对RBAC集群权限管理系统升级：

- 禁止未完成RBAC权限授予的子用户访问集群资源，请及时联系主账号完成RBAC授权，以免带来生产上的不便。
- 子用户将只拥有被指定授予的集群访问权限，原有的默认访问权限将被禁止。

授权说明

- 只有如下账号/用户/角色，才能配置集群的 RBAC 权限：
  - 百度智能云主账号。
  - 对指定集群拥有RBAC管理权限，且具有 IAMReadOnlyAccessPolicy 或 IAMFullControlAccessPolicy 权限的 IAM 用户、IAM 角色。
- IAM 用户、IAM 角色默认对自身创建的集群拥有管理员权限。

前提条件

- 已创建集群。详细操作，请参见创建集群。
- IAM 用户已完成 CCE 侧的相关权限配置。详细操作，请参见配置IAM预置权限策略和配置IAM自定义权限策略。
- 已创建 IAM 角色。详细操作，请参见创建角色。

权限说明

CCE内置了三种级别的RBAC权限，如下所示：

权限	权限说明
管理员	对集群的所有操作权限
运维开发	对命名空间下的资源有读写权限，对节点、命名空间、存储卷、存储类有只读权限
只读	仅对命名空间下的资源有只读权限

配置IAM用户的RBAC授权

1. 登录百度智能云管理控制台，进入"产品服务->容器引擎 CCE"，在左侧导航栏，点击 "集群管理->权限管理"，进入权限管理页面。
2. 在子用户列表中选择要授权的子用户，点击**RBAC授权**，进入RBAC授权页。
3. 点击RBAC授权列表左上方**添加授权**，弹出添加授权配置框，选择需要配置的权限，以及对应的集群和命名空间。

> **说明**
>
> - 如果子用户在IAM中没有被授予CCE任何权限，则授权不会生效。
> - 当集群范围选择"所有集群"时，仅包含当前已创建的全部集群，若后续新建集群还需再次授权。

4. 添加授权成功后，可以查看授权列表。

配置IAM角色的RBAC授权

1.登录百度智能云管理控制台，进入"产品服务->容器引擎 CCE"，在左侧导航栏，点击 "集群管理->权限管理"，进入IAM角色Tab页面。

2.输入需要授予 RBAC 权限的 IAM 角色名称，然后单击 管理角色权限。

3.在角色权利管理页面左上方添加授权，弹出添加授权配置框，选择需要配置的权限，以及对应的集群和命名空间。

4.单击 确定，配置完成 IAM 角色的 RBAC 权限。

FAQ

作为一个子用户，想对角色ggg 授予CCE集群A和CCE集群B的RBAC管理员权限，需要具备哪些前提条件？

1. 当前子用户具有目标集群A和目标集群B的IAM权限
2. 当前子用户拥有目标集群A和目标集群B的RBAC管理员权限
3. 当前子用户拥有IAMReadOnlyAccessPolicy 或 IAMFullControlAccessPolicy 权限

配置集群OIDC认证

本文介绍 CCE 集群 OIDC 认证、启用说明，以及如何与 RBAC 结合使用。

认证说明

- CCE 集群 OIDC 认证依赖百度云 IAM 的身份认证。支持 IAM 用户、IAM 角色认证。
- CCE 集群默认仅支持基于 X509 证书的认证方式。**仅支持新版 CCE 集群**在创建时选择开启 OIDC 认证。
- CCE 集群开启 OIDC 认证后，基于 X509 证书的认证能力依然保留。
- CCE 集群开启 OIDC 认证后，支持通过 AssumeRole 的方式操作集群资源。

操作步骤

开启OIDC认证

1. 登录百度智能云控制台，进入 "产品服务 > 容器引擎 CCE"；
2. 在集群列表页面单击“创建集群”进入集群创建页面：
3. 在**集群配置 > 高级设置**中，勾选“开启OIDC认证”。

4. 在目标集群列表详情页面，单击**查看集群凭证**下载基于 OIDC 的 KubeConfig 文件，基于该文件可以通过 OIDC 认证访问集群。
5. 通过 KubeConfig 或者直接在控制台操作集群资源，请先确保进行操作的子用户 (通过子用户访问时)，或者用户所切换的 IAM Role (通过 AssumeRole 访问时) 已经提前被赋予相应的 RBAC 权限。

配置RBAC权限

OIDC 仅仅解决集群认证问题，集群认证通过后，开始进行鉴权操作 (一般基于 RBAC)。需要事先为用户或者 IAM Role 绑定相应的 RBAC 权限，才能正常操作集群资源。

- 主用户:默认具备集群管理员权限，即可以操作集群全部资源，无需再绑定 RBAC 权限。
- 子用户：对于子用户的权限绑定，有以下两种方式可选：

通过预置RBAC授权

通过 CCE 控制台为子用户绑定 RBAC 权限，详情参考：配置预置RBAC权限策略

自定义RBAC策略授权

手动在集群创建 ClusterRole/Role 和 ClusterRoleBinding/RoleBinding，详情参考：K8s RBAC 官方文档。示例如下：

```
  kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: custom-cluster-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: default
  name: custom-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: custom-cluster-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: 294ae2c68794fb1ab58e71fbc8ef338
```

注意: ClusterRoleBinding/RoleBinding 中的 subjects.kind = User，subjects.name = 子用户 ID。

IAM Role

IAM Role 目前只能通过手动在集群中创建 ClusterRole/Role 和 ClusterRoleBinding/RoleBinding 方式绑定 RBAC 权限，示例如下：

```
  kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: custom-cluster-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: default
  name: custom-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: custom-cluster-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: 7e442e97bba54d9ea0d8da1fcacecd7a
```

> **注意：**
>
> ClusterRoleBinding/RoleBinding 中的 subjects.kind = Group，subjects.name = IAM Role ID；不要混淆 IAM Role 和 K8s Role，IAM Role 会被映射为集群中的 Group，Group name 等于 IAM Role ID。按照示例给 Group 绑定了 RBAC 权限后，那么扮演该 IAM Role 的用户就会获得相同的 RBAC 权限。

虚拟节点

管理虚拟节点

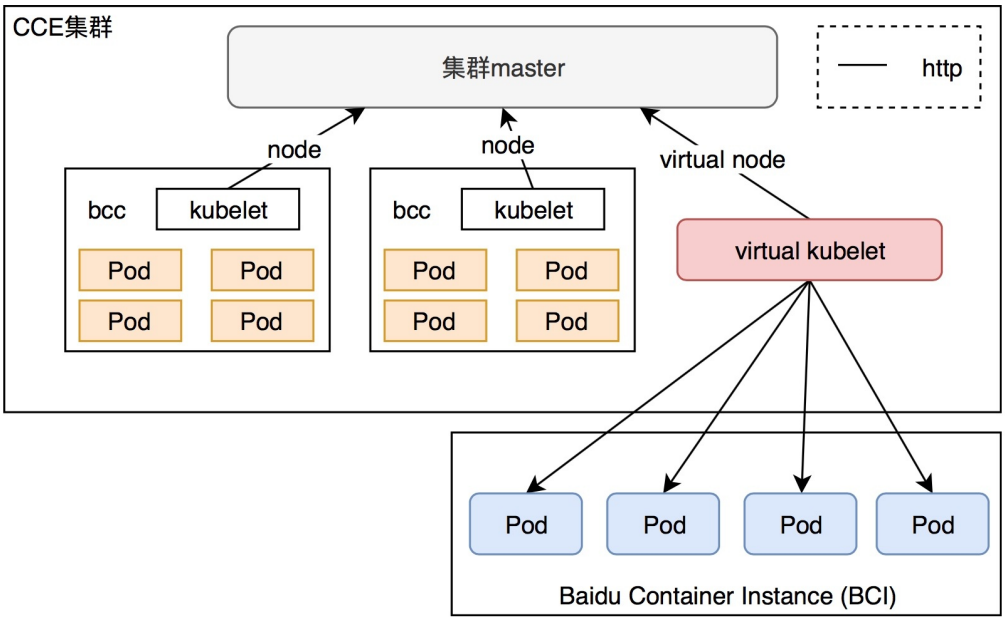
管理虚拟节点

本文档主要介绍虚拟节点和BCI，包含如何在CCE中创建虚拟节点和通过虚拟节点创建BCI Pod。

虚拟节点和百度智能云容器实例BCI

百度智能云容器实例BCI提供无服务器化的容器资源。您只需提供容器镜像及启动容器所需的配置参数，即可运行容器，而无需关心这些容器如何被调度部署到底层的物理服务器资源中。BCI服务将为您完成IaaS层资源的调度和运维工作，从而简化您对容器的使用流程，降低部署和维护成本。同时BCI只会对您创建容器时申请的资源计费，因此实现真正的按需付费。

虚拟节点Virtual Node来源于Kubernetes社区的virtual-kubelet技术，实现了在CCE集群中调度BCI实例，使用户无需购买物理集群节点即可快速启动和编排容器。启动容器的数量不受物理节点的容量限制，赋予了集群极大的弹性伸缩能力。



准备工作

- 注册百度账号，并完成实名认证。
- 开通百度智能云容器实例服务。登录容器实例控制台开通相应的服务。
- 创建一个容器集群，操作步骤参考创建集群。
  - 推荐：集群类型请选择标准Kubermetes托管集群；Master配置请选择托管Master
  - 推荐：k8s版本1.20及以上
- 集群创建成功后，请增加2个worker BCC节点，用来运行相关系统组件，BCC配置建议在2C8G及以上。

操作指南

新版本集群（集群id为cce-开头）和老版本集群（集群id以c-开头）操作有所不同，请根据需要添加虚拟节点的目标集群选择对应的操作。

- 新版本集群操作步骤
- 老版本集群操作步骤

新版本集群（集群id为cce-开头）

前提条件

- 已创建CCE标准托管集群或者CCE标准独立集群，且集群版本为1.18.9/1.20.8/1.22.5/1.24.4/1.26.9/1.28.8。如需创建集群则见创建集群。
- 已开通弹性容器实例服务并确认集群所在区域在BCI支持的地域列表内。可登录弹性容器实例控制台开通相应的服务，并查看当前支持的地域和可用区。

步骤一：部署CCE Virtual Kubelet组件 在CCE集群中，需要通过组件管理页面部署CCE Virtual Kubelet组件，根据两种不同集群类型，请按照如下不同步骤:

集群类型	安装步骤
CCE标准托管集群	1. 登录容器服务管理控制台，在左侧导航栏选择**集群**。 2. 在**集群列表**页面，单击目标集群名称，然后在左侧导航栏，选择**运维管理 > 组件管理**。 3. 在**组件管理**页面的**核心组件**区域，选择**CCE Virtual Kubelet**组件，然后单击**安装**，并按照页面提示完成操作。安装时，使用集群默认子网和安全组作为CCE Virtual Kubelet的初始BCI配置参数，组件默认被托管，不占用Worker节点资源。安装成功后组件状态为**运行中**。 a. 若容器网络模式VPC-ENI的CCE集群，默认子网为容器网络子网，默认安全组优先使用ENI安全组，若无则使用Worker安全组 b. 若容器网络模式VPC路由模式的CCE集群，默认子网为VPC内第一个页"通用"类型子网，默认安全组为Worker安全组 4. 如需修改BCI配置参数，请参见下文"修改CCE虚拟节点配置"进行 相关操作。
CCE标准独立集群	1. 登录容器服务管理控制台，在左侧导航栏选择**集群**。 2. 在**集群列表**页面，单击目标集群名称，然后在左侧导航栏，选择**运维管理 > 组件管理**。 3. 在**组件管理**页面的**核心组件**区域，选择**CCE Virtual Kubelet**组件，然后单击**安装**，并按照页面提示完成操作。安装时，使用集群默认子网和安全组作为CCE Virtual Kubelet的初始BCI配置参数，组件CCE Virtual Kubelet 组件将在kube-system命名空间下创建一个名为cce-virtual-kubelet的statefulset，运行在Worker节点上。若顺利安装完成，则组件显示状态为**运行中**。 a. 若容器网络模式VPC-ENI的CCE集群，默认子网为容器网络子网，默认安全组优先使用ENI安全组，若无则使用Worker安全组 b. 若容器网络模式VPC路由模式的CCE集群，默认子网为VPC内第一个页"通用"类型子网，默认安全组为Worker安全组 4. 如需修改BCI配置参数，请参见下文"修改CCE虚拟节点配置"进行 相关操作。

注意：若您已通过Helm模板安装cce-virtaul-kubelet，暂不支持在组件管理安装，若您想通过组件中心安装则可 提交工单 联系我们处理。

步骤二：将Pod调度到BCI上运行 在虚拟节点启动Pod

相较于常规Pod，需要在Pod Spec中添加如下的nodeSelector和tolerations字段，才能将Pod调度到虚拟节点上。

Pod Spec示例：

```
spec:
nodeSelector:
  type: "virtual-kubelet"
tolerations:
- key: "virtual-kubelet.io/provider"
  operator: "Equal"
  value: "baidu"
  effect: "NoSchedule"
```

调度到虚拟节点上的Pod会以BCI Pod的形式启动。 更多BCI Pod配置项说明请参考 配置BCI Pod

删除虚拟节点

1. 通过`kubectl drain <node name> --force`操作停止虚拟节点上的所有Pod。
2. 在 Helm-Helm实例页面，选择相应集群和namespace，删除虚拟节点对应的`cce-virtual-kubelet`实例。
3. 通过命令`kubectl delete node <node name>`删除虚拟节点。

> **注意**: 当集群中存在分配在虚拟节点上的BCI Pod时，卸载`cce-virtual-kubelet`组件会导致BCI实例的残留。

老版本集群（集群id以c-开头）

创建虚拟节点

登录CCE管理控制台，点击需要添加虚拟节点的集群，选择左侧导航栏中的虚拟节点标签，点击页面上的创建虚拟节点按钮。

> 注意: 如果用户尚未开通容器实例服务，创建按钮将置灰，点击创建按钮会提示需要开通容器实例服务。

点击创建按钮后，进入虚拟节点创建页面，相关参数说明如下：

基本信息定义了虚拟节点的属性：

参数	含义	备注
虚拟节点名称	虚拟节点名称，会在页面虚拟节点列表和kubernetes node列表中展示	同一个集群中节点名称不可重复
CPU使用限额	虚拟节点的CPU Quota，限制虚拟节点上可以调度的BCI实例CPU总量	需要显式声明BCI Pod的resource quota才会生效
内存使用限额	虚拟节点的Memory Quota，限制虚拟节点上可以调度的BCI实例内存总量	需要显式声明BCI Pod的resource quota才会生效

容器实例属性定义了在虚拟节点上启动的BCI实例的属性：

参数	含义	备注
VPC	BCI实例所在VPC	如果选择VPC与集群所在VPC不同，可能导致网络地址的冲突，详见网络冲突说明
可用区及子网	BCI实例所在的可用区及子网	只显示对应VPC下可以创建BCI实例的可用区及子网
安全组	应用于BCI实例的安全组	-

点击确认后跳转到虚拟节点列表页面，列表中会展示刚刚添加的虚拟节点及其状态。

管理虚拟节点

在虚拟节点列表中，可以查看虚拟节点当前的状态，以及虚拟节点的属性信息。

- 虚拟节点名称：虚拟节点名称。
- 虚拟节点状态：
  - 创建中：虚拟节点正在创建中，一般创建时间不超过一分钟。
  - 正常：虚拟节点Ready，可以将Pod调度到虚拟节点上启动对应的BCI实例。
  - 异常：虚拟节点NotReady，可以查看对应的virtual-kubelet排查异常原因。
  - 删除中：点击删除按钮后，虚拟节点状态会变为删除中，直至该虚拟节点被完全删除。
- 操作：
  - 查看virtual-kubelet：查看该虚拟节点对应的virtual-kubelet运行状态。
  - 删除：删除该虚拟节点。
  - 如何使用：提供将Pod调度到对应虚拟节点需要指定的Pod Spec字段。

在虚拟节点启动Pod

相较于常规Pod，需要在Pod Spec中添加如下的`nodeSelector`和`tolerations`字段，才能将Pod调度到虚拟节点上。

```
spec:
nodeSelector:
  type: "virtual-kubelet"
tolerations:
- key: "virtual-kubelet.io/provider"
  operator: "Equal"
  value: "baidu"
  effect: "NoSchedule"
```

调度到虚拟节点上的Pod会以BCI Pod的形式启动。

更多BCI Pod配置项说明请参考配置BCI Pod。

删除虚拟节点

在虚拟节点列表中点击对应虚拟节点的删除按钮，确认需要删除的虚拟节点后，点击确认，虚拟节点进入删除中状态，直至在列表中不可见即为删除成功。

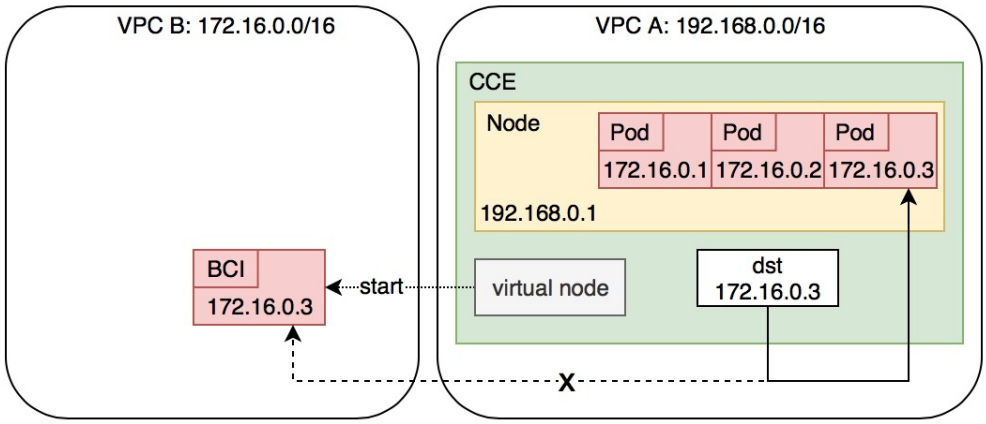
如果删除时提示禁止删除，请参考为什么无法删除虚拟节点。

为什么创建虚拟节点时提示有网络冲突

当创建虚拟节点时，选择集群所在VPC（称为VPC A）以外的其他VPC（称为VPC B）作为容器实例VPC后，虚拟节点上运行的BCI实例将运行在B VPC中。

VPC A与VPC B的地址空间可能存在重合，同时CCE集群容器网络也会占用A VPC地址空间以外的一段地址作为容器网段，这段地址也可能会和VPC B的地址空间存在重合。因此，BCI Pod的IP地址代表的是VPC B中对应的网络地址，在VPC A中可能存在另一个实体拥有相同的IP地址。此时如果在集群中访问BCI实例的IP地址，实际访问到的对象并不是该BCI实例，而是VPC A中的拥有相同IP地址的网络实体，此时就可能会出现非预期的访问结果。

即下图中在VPC A中访问172.16.0.3的流量并不会走虚线路径打到BCI实例上，而是会走实线路径打到集群中的另一个Pod。



如果在充分理解这种场景可能带来的问题的情况下，仍有业务场景需要在VPC B中调度容器实例，可以继续进行创建。

我们强烈建议仅让CCE集群调度处于与集群相同VPC的BCI实例。

为什么无法删除虚拟节点

当仍有运行中的BCI实例位于某个虚拟节点时，CCE不允许删除该虚拟节点，因为直接删除虚拟节点可能导致运行在该虚拟节点上的BCI实例泄漏。

通常情况下我们无需删除虚拟节点，虚拟节点不同于真实节点，不会占用集群计算资源。如果用户需要删除虚拟节点，我们建议手动停止运行在虚拟节点上的Pod，或者驱逐虚拟节点上的所有Pod。

驱逐虚拟节点上的Pod可以使用kubectl命令行工具执行以下命令进行操作，假设需要删除的虚拟节点名称为`bci-virtual-kubelet-0`:

```
$ kubectl drain bci-virtual-kubelet-0 --force
```

配置BCI Pod

配置CPU和Memory资源

BCI规格说明及创建见指定vCPU和内存创建Pod

BCI实例的运行费用与实例资源配置和实例运行时长有关，详情可以参考BCI计费规则。

BCI支持的规格如下表所示

CPU/核	内存区间（GiB）
0.25	0.5、1、2
0.5	1、2、4
1	1、2、4、8
2	4、8、16
4	8、16、32
8	16、32
12	24、48
16	32、64
32	64、128

如果配置了不合法的CPU和Memory规格，将无法成功创建BCI实例。

如果您有不在上述规格内的实例规格诉求，请提交工单。

镜像拉取

当前BCI仅支持拉取存储于以下仓库中的容器镜像：

- 百度智能云容器镜像服务CCR（镜像地址为registry.baidubce.com/<namespace>/<name>:<tag>）
- Docker Hub（镜像地址为 [<namespace>/]<name>:<tag>）
- 百度智能云CCE镜像仓库（镜像地址为hub.baidubce.com/<namespace>/<name>:<tag>）

如果镜像为私有镜像，可以为BCI Pod配置imagePullSecrets拉取私有镜像，示例可以参考CCE集群中使用私有镜像实践。

存储

当前BCI Pod支持挂载以下类型的volume：

- ConfigMap,详细说明见文档挂载EmptyDir数据卷,详细说明见文档挂载CFS文件存储
- Secret
- EmptyDir（暂不支持指定sizeLimit，单个Pod的emptyDir可用存储空间为10GB）
- PersistentVolumeClaim(PVC),PVC类型当前仅支持CFS/NFS/PFS PVC挂载.暂不支持挂载CDS云磁盘。
  - NFS(CFS), 详细说明见文档挂载CFS文件存储
  - PFS, 详细说明见文档挂载PFS并行文件存储
  - BOS, 详细说明见文档挂载BOS数据卷

暂不支持subPath/subPathExpr挂载。

网络

配置公网访问 一个BCI Pod将占用所属VPC下的交换机的一个弹性网卡资源，默认具备一个内网IP地址。如果需要连接公网，例如需要拉取公网镜像。则需要为BCI Pod绑定EIP，或者为所属VPC绑定NAT网关，更多信息详见连接公网

K8S service 在BCI Pod中访问集群内service, 参考文档配置BCI Pod访问集群内Service；

Serverless集群中Pod访问ClusterIP的方式根据集群创建时的参数不同有所区别，详情请参考在Serverless集群中使用Service。若Cluster Service模式选择为BLB则无需在BCI Pod中注入kube-proxy sidecar。

若您的BCI服务需发布被外部访问，则参考文档通过Service访问BCI服务

更多网络配置 配置BCI Pod所属安全组 支持自定义DNSConfig

配置日志推送

当前BCI支持将Pod容器内日志推送到BLS，进行日志的持久化存储和结构化查询。详细配置请参考 BLS日志采集方式

配置监控

查看实例监控指标 通过VK获取实例监控指标

错误排查

若创建BCI Pod后，Pod状态变为ProviderFailed，代表BCI实例创建失败。可以通过kubectl describe po <pod名称> --namespace <命名空间>查看具体失败原因。

若创建BCI Pod后，实例状态长时间处于Creating，一般为BCI订单异常导致，可以通过查看pod annotation中的bci.virtual-kubelet.io/order-id字段查询到BCI订单ID，查询对应的订单失败原因。

BCI自定义annotation列表

BCI支持的anntation列表详见BCI Pod Annotation

> **注意：**annotation需要配置在PodSpec中，而不是DeploymentSpec中。
>
> **注意：**配置网络，安全组和可用区相关annotation时，需要保证相互之间的一致性，如子网与安全组应该同属一个VPC，子网与可用区参数需要匹配，等等。没有在Pod中单独配置的情况下默认使用虚拟节点本身的配置。
>
> **注意：**若通过annotation指定了子网参数，必须同时通过annotation指定可用区参数，可用区参数的值为子网所属可用区。

BCI限制

由于公有云安全性及虚拟节点带来的限制，BCI目前还不支持Kubernetes中HostPath、DaemonSet等功能，如下表所示。

不支持的功能	说明	推荐替代方案
HostPath	挂载本地宿主机文件到容器中	使用emptyDir、云盘或者CFS文件系统
HostNetwork	将宿主机端口映射到容器上	BCI将忽略HostNetwork字段，推荐使用type=LoadBalancer的负载均衡
DaemonSet	在容器所在宿主机上部署Static Pod	通过sidecar形式在Pod中部署多个镜像
Privileged权限	容器拥有privileged权限	去除业务逻辑特权依赖
type=NodePort的Service	将宿主机端口映射到容器上	使用type=LoadBalancer的负载均衡

更多文档见： 对接概述

配置bci-profile

为了支持CCE Virtual Kubelet组件（VK）能动态热更新配置，BCI支持了bci-profile功能。

- 更新配置时无需重启VK。
- 对于新创建的BCI Pod，可以即时生效更新后的配置；对于存量BCI Pod，需要滚动发布后才能生效更新后的配置。

注意事项

使用该功能时，请确保集群中的VK（cce-virtual-kubelet组件）版本>=1.0.1。关于如何升级组件，请参见CCE Virtual Kubelet组件说明文档。

配置说明

创建Pod时，系统会读取kube-system命名空间下的bci-profile配置文件（名为bci-profile的ConfigMap），按其data配置来创建Pod。

您可以通过kubectl get cm -n kube-system bci-profile -o yaml命令查看bci-profile的YAML。

bci-profile的YAML模板如下：

```
  apiVersion: v1
kind: ConfigMap
metadata:
  name: bci-profile
  namespace: kube-system
data:
  config: |
    {
      "region": "gz",
      "clusterID": "example-cluster-id",
      "subnets": [
        {
          "subnetID": "subnet-12345",
          "logicalZone": "zone-a"
        },
        {
          "subnetID": "subnet-67890",
          "logicalZone": "zone-b"
        }
      ],
      "securityGroupID": "sg-abcdef",
      "enableReserveFailedPod": true,
      "dnsConfig": {
        "nameservers": [
          "169.254.20.10",
          "172.21.0.10"
        ],
        "searches": [
          "default.svc.cluster.local",
          "svc.cluster.local",
          "cluster.local"
        ],
        "options": [
          {
            "name": "ndots",
            "value": "3"
          },
          {
            "name": "attempts",
            "value": "2"
          },
          {
            "name": "timeout",
            "value": "1"
          }
        ]
      }
    }
```

修改bci-profile 方式一： 通过kubectl edit命令

kubectl edit configmap bci-profile -n kube-system

注意配置内容需要满足json语法

方式二：通过容器服务管理控制台

1. 登录容器服务管理控制台。
2. 在集群页面，找到目标集群，单击集群名称。
3. 在集群管理页的左侧导航栏，选择配置管理>配置项。
4. 选择命名空间为kube-system。
5. 找到bci-profile，单击YAML编辑。

更新配置项 data的config字段包含的子网、安全组等固定配置项，以json方式存储，您可以根据需要进行更新，更新后的配置可以即时生效（无需重启VK）。注意，如下配置为均为集群级别的配置项，即在创建BCI Pod时，如果没有额外配置，会采用bci-profile中的配置。 支持更新的配置项如下：

配置项	说明
subnets	BCI Pod运行的默认子网列表，需同时写出子网id和逻辑可用区；默认为VK组件安装时使用的子网参数
securityGroupID	BCI Pod所属安全组，默认为VK组件安装时参数,如 "securityGroupID": "sg-abcdef"
enableReserveFailedPod	是否自动保留失败的BCI Pod,默认为false
dnsConfig	BCI Pod配置的自定义DNS 配置，默认为null

子网示例：

```
"subnets": [
{
  "subnetID": "subnet-12345",
  "logicalZone": "zone-a"
},
{
  "subnetID": "subnet-67890",
  "logicalZone": "zone-b"
}
],
```

dnsConfig配置示例：

```
"dnsConfig": {
"nameservers": [
  "169.254.20.10",
  "172.21.0.10"
],
"searches": [
  "default.svc.cluster.local",
  "svc.cluster.local",
  "cluster.local"
],
"options": [
  {
    "name": "ndots",
    "value": "3"
  },
  {
    "name": "attempts",
    "value": "2"
  },
  {
    "name": "timeout",
    "value": "1"
  }
 ]
}
```

弹性伸缩

容器水平伸缩（HPA）

容器引擎 CCE 提供的 Pod 自动扩缩容功能（Horizontal Pod Autoscaler，HPA）可以根据CPU、内存度量指标自动伸缩服务的 Pod 数量，能更好的应对突发流量等场景。本文介绍如何通过百度智能云容器引擎控制台和 YAML 方式实现 Pod 自动扩缩容。

实现原理

- HPA 组件每隔 15 秒会获取 Pod 的监控指标，然后根据当前指标数据、当前副本数和该指标的目标值进行计算，计算所得结果作为工作负载的期望副本数。当期望副本数与当前副本数不一致时，HPA会触发工作负载进行Pod副本数调整，从而达到自动伸缩的目的。
- 以 CPU 使用率为例，假设工作负载有 2 个实例，平均 CPU 使用率（当前指标数据）为 90%，自动伸缩设置的目标值为 60% 时，则自动调整的副本数为：90% * 2 /60% = 3，即由 2 个实例自动扩容为 3 个实例。

> **说明：**
>
> 如果对一个工作负载设置了多个伸缩指标，HPA 会根据各个指标分别计算出目标副本数，取最大值进行扩缩容操作。

前提条件

- 已创建 CCE 集群，具体操作请参见 创建CCE集群。

操作步骤

通过控制台使用HPA

方式一：为已有工作负载开启HPA

1. 登录百度智能云管理控制台，进入**产品服务 > 云原生 > 容器引擎 CCE**，单击**集群管理 > 集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**自动伸缩 > 水平伸缩（HPA）**。
2. 在水平伸缩列表页面单击**新建规则**操作。
3. 在“新建弹性伸缩”页面配置伸缩配置。

配置项	必选/可选	配置说明
命名空间	必选	选择伸缩规则所在的命名空间。
规则名称	必选	弹性伸缩规则的名称，最长63个字符，只能包含小写字母，数字及分割符("-")，且必须以小写字母开头，数字或小写字母结尾。
工作负载类型	必选	支持无状态部署Deployment和有状态部署Statefulset。
工作负载名称	必选	选择已选命名空间下的无状态部署或有状态部署，支持选择多个。
伸缩指标	必选	提供CPU、内存指标，支持选择多个指标。
最小Pod数	必选	Pod数量不会低于设定范围。
最大Pod数	必选	Pod数量不会超过设定范围。

> **说明：**
>
> - 支持选择多个工作负载，若选择多个工作负载，则会按照工作负载数量生成对应的伸缩规则，实际生成的伸缩规则名称为“规则名称+工作负载名称”
> - Pod数量会在设定的范围内自动调节，不会超出该设定范围。

4. 单击**确认**，即可完成创建。

方式二：在创建工作负载过程中开启HPA

当前以创建无状态部署为例：

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**工作负载>无状态部署**。
2. 在"无状态部署"页面单击**新建无状态部署**操作。
3. 在“新建无状态部署-高级设置”页面勾选**水平伸缩**为该工作负载开启水平伸缩HPA。

配置项	必选/可选	配置说明
规则名称	必选	弹性伸缩规则的名称，最长63个字符，只能包含小写字母，数字及分割符("-")，且必须以小写字母开头，数字或小写字母结尾
伸缩指标	必选	提供CPU、内存指标，支持选择多个指标
最小Pod数	必选	Pod数量不会低于设定范围
最大Pod数	必选	Pod数量不会超过设定范围

4. 单击**完成**，一个支持HPA的无状态部署就已配置完成。

通过kubectl命令使用HPA

为演示 HPA，将使用一个基于 php-apache 镜像的定制 Docker 镜像。镜像中包括一个 index.php 页面，其中包含一些可以运行 CPU 密集计算任务的代码。 示例 Dockerfile 如下：

```
FROM php:5-apache

ADD index.php /var/www/html/index.php

RUN chmod a+rx index.php
```

示例 index.php 文件如下：

```
<?php
$x = 0.0001;
for ($i = 0; $i <= 1000000; $i++) {
 $x += sqrt($x);
}
echo "OK!";
?>
```

前提条件

- 已配置 kubectl，可以在本机访问 Kubernetes 集群，详情参见 通过kubectl连接集群。

步骤一：部署工作负载及服务

- 首先，部署一个 Deployment 运行上述 Docker 镜像并将其暴露成为一个 Kubernetes 服务（Service）

```
kubectl run php-apache --image=hpa-example --requests=cpu=200m --expose --port=80

service "php-apache" created
deployment "php-apache" created
```

步骤二：创建HPA

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
deployment "php-apache" autoscaled


##### 创建出来后，当前监控值为unknown，执行第3步后，等待1-2分钟，这里会变为正常的百分比
[root@instance-2tpjy37t ~]# kubectl get hpa
NAME         REFERENCE              TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   <unknown>/50%   1         10        1          5s
```

对应 YAML 如下：

```
apiVersion: autoscaling/v2alpha1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

字段解释：

- **scaleTargetRef： HPA 自动扩缩容的目标对象**
- **minReplicas：最小 Pod 数量**

- **maxReplicas：允许的最大 Pod 数量**
- **metrics：度量指标**
- **targetAverageUtilization：即设定的资源利用率，超过设定值时触发水平扩展**

HPA 目前支持 3 种类型的度量指标，详情参考 kubernetes Horizontal Pod Autoscaler：

- **预定义指标：** Pod 的 CPU 和内存使用率（内置支持）
- **自定义 Pod 指标：** 应用程序提供的监控指标（需部署监控、自定义 metric server）
- **自定义 Object 指标：** 和 Pod 同一个命名空间中的其它资源的监控指标（需部署监控、自定义 metric server）

步骤三：向服务增加负载，验证自动扩容 启动一个容器，并通过一个循环向php-apache服务器发送无限的查询请求（请在另一个终端中运行以下命令）

```
 kubectl run -i --tty load-generator --image=busybox /bin/sh

Hit enter for command prompt

$ while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done

##### 输出：
OK!OK!OK!
```

配置项	必选/可选	配置说明
名称	必选	指定定时水平伸缩的名称，长度为1-63个字符，只能包含数字、小写字母、和“-”，且必须以字母开头，数字或字母结尾。
命名空间	必选	选择定时水平伸缩所在的命名空间。
关联对象类型	必选	支持为Deployment、StatefulSet以及水平伸缩HPA设置定时水平伸缩。
关联对象	必选	选择需要开启的Deployment、StatefulSet以及水平伸缩HPA的资源。
定时任务	必选	配置定时执行伸缩规则的策略，支持添加多个定时任务。 **名称：**指定定时任务的名称，长度为1-63个字符，只能包含数字、小写字母、和“-”，且必须以字母开头，数字或字母结尾。 **定时规则：**输入定时规则，格式：秒 分 时 天 月 周（如：1 2 3 4 * * 表示每月4日3点2分1秒执行一次伸缩任务），更多信息请见go-cron。 **单词执行：**配置是否开启定时任务仅支持一次，若勾选则该任务仅执行一次。**目标副本数：**指定伸缩的Pod实例数。
任务跳过时间	可选	设置任务不执行时间规则，当匹配到该时间时不触发定时任务，格式为 秒 分 时 天 月 周，最小单位为天。

4. 单击**确定**，即可完成创建。

方式二：在创建工作负载过程中开启CronHPA

当前以创建无状态部署为例：

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**工作负载>无状态部署**。
2. 在"无状态部署"页面单击**新建无状态部署**操作。
3. 在“新建无状态部署-高级设置”页面勾选**定时水平伸缩**为该工作负载开启水平伸缩CronHPA。

配置项	必选/可选	配置说明
名称	必选	指定定时水平伸缩的名称，长度为1-63个字符，只能包含数字、小写字母、和“-”，且必须以字母开头，数字或字母结尾。
命名空间	必选	选择定时水平伸缩所在的命名空间。
关联对象类型	必选	支持为Deployment、StatefulSet以及水平伸缩HPA设置定时水平伸缩。
关联对象	必选	选择需要开启的Deployment、StatefulSet以及水平伸缩HPA的资源。
定时任务	必选	配置定时执行伸缩规则的策略，支持添加多个定时任务。 **名称：**指定定时任务的名称，长度为1-63个字符，只能包含数字、小写字母、和“-”，且必须以字母开头，数字或字母结尾。 **定时规则：**输入定时规则，格式：秒 分 时 天 月 周（如：1 2 3 4 * * 表示每月4日3点2分1秒执行一次伸缩任务），更多信息请见go-cron。 **单词执行：**配置是否开启定时任务仅支持一次，若勾选则该任务仅执行一次。**目标副本数：**指定伸缩的Pod实例数。
任务跳过时间	可选	设置任务不执行时间规则，当匹配到该时间时不触发定时任务，格式为 秒 分 时 天 月 周，最小单位为天。

4. 单击**完成**，一个支持CronHPA的无状态部署就已配置完成。

通过kubectl命令使用CronHPA

以下是一个具体的CronHPA示例:

```
  apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
   scaleTargetRef:
      apiVersion: apps/v1
      kind: Deployment
      name: nginx-deployment-basic
   excludeDates:
   - "* * * 15 11 *"
   - "* * * 1 11 *"
   crons:
   - name: "scale-down"
     schedule: "30 */1 * * * *"
     targetSize: 1
     runOnce: true
   - name: "scale-up"
     schedule: "01 */1 * * * *"
     targetSize: 3
```

部署参数介绍

名称	作用	是否必须
spec.scaleTargetRef	期望扩缩容对象的元数据	必须
crons.name	任务名称	必须
crons.schedule	任务触发的时间	必须
crons.runOnce	任务单次执行	可选
crons.targetSize	期望副本数	必须
spec.excludeDates	任务不触发的时间	可选

其中`schedule`同`excludeDates`需要按照如下格式配置:

字段名称	是否必填	允许值	允许的特殊字符
Seconds	是	0-59	*/,-
Minutes	是	0-59	*/,-
Hours	是	0-23	*/,-
Day of month	是	1-31	*/,-?
Month	是	1-12 或 JAN-DEC	*/,-
Day of week	是	0-6 或 SUN-SAT	*/,-?

特殊字符说明

- 星号 (*) 表示表达式能匹配该字段的所有值。如在第5个字段使用星号(month)，表示每个月。
- 斜线(/) 表示增长间隔，如第1个字段(minutes) 值是 3-59/15，表示每小时的第3分钟开始执行一次，之后每隔 15 分钟执行一次（即 3、18、33、48 这些时间点执行），这里也可以表示为：3/15
- 逗号(,) 用于枚举值，如第6个字段值是 MON,WED,FRI，表示 星期一、三、五 执行。
- 连字号(-) 表示一个范围，如第3个字段的值为 9-17 表示 9am 到 5pm 直接每个小时（包括9和17））
- 问号(?) 只用于日(Day of month)和星期(Day of week)，\表示不指定值，可以用于代替 *

cron举例说明

- 每隔5秒执行一次：*/5* * * ?
- 每隔1分钟执行一次：0 */1* * ?
- 每天23点执行一次：0 0 23 * * ?
- 每天凌晨1点执行一次：0 0 1 * * ?

- 每月1号凌晨1点执行一次：0 0 1 1 * ?
- 在26分、29分、33分执行一次：0 26,29,33 * * * ?
- 每天的0点、13点、18点、21点都执行一次：0 0 0,13,18,21 * * ?

CronHPA扩缩容Deployment示例 CronHPA要应用在普通工作负载只需要将`spec.scaleTargetRef`字段配置成该工作负载的信息，以下是一个具体的CronHPA扩缩容Deployment的示例:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-basic
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef: //配置成以上deployment的信息
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment-basic
  crons:
  - name: "scale-down"
    schedule: "30 */1 * * * *"
    targetSize: 1
  - name: "scale-up"
    schedule: "01 */1 * * * *"
    targetSize: 3
```

CronHPA兼容HPA示例 从CronHPA和HPA的模板定义中可以看出：

- CronHPA和HPA都是通过scaleTargetRef字段来获取伸缩对象。
- CronHPA通过jobs的crontab规则定时伸缩副本数。
- HPA通过资源利用率判断伸缩的情况。

如果同时设置CronHPA和HPA，那么会出现CronHPA和HPA同时操作一个scaleTargetRef的场景。CronHPA和HPA相互独立无法感知，后执行的操作会覆盖先执行的操作,例如刚刚定时扩容完成但是又由于资源利用率低被缩容回去了。 所以当期望扩缩容的对象被HPA引用的时候，为了避免HPA与CronHPA的冲突导致结果被覆盖，需要把`spec.scaleTargetRef`配置为对应的HPA对象，这样CronHPA能够感知HPA的状态并且通过调整HPA的上下限来调整副本数。

以下是一些兼容CronHPA与HPA来实现定时扩缩容的具体场景示例:

HPA(min/max)	CronHPA	deployment	result	说明
1/10	5	5	HPA(1/10) deployment 5	CronHPA期望副本数为固定一致，无需变更
1/10	4	5	HPA(1/10) deployment 5	当前副本数5高于CronHPA期望副本数4，并且期望副本数在区间[1,10]内，无需变更
1/10	6	5	HPA(6/10) deployment 6	CronHPA期望副本数6高于当前副本数5，调整HPA下限自动扩容副本数至6
5/10	4	5	HPA(4/10) deployment 5	CronHPA期望副本数低于当前副本数5，保留当前副本数 CronHPA期望副本数低于HPA下限，修改HPA下限(保证至少有4个副本)
5/10	11	5	HPA(11/11) deployment 11	CronHPA期望副本数高于HPA上限，修改HPA下限至CronHPA期望副本数，上限同下限

如下是CronHPA兼容HPA具体的yaml示例：

```
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-deployment-basic-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment-basic
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
---
apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef://配置为HPA对象
    apiVersion: autoscaling/v1
    kind: HorizontalPodAutoscaler
    name: nginx-deployment-basic-hpa
  crons:
    - name: "scale-down"
      schedule: "30 */1 * * * *"
      targetSize: 5
      runOnce: false
    - name: "scale-up"
      schedule: "01 */1 * * * *"
      targetSize: 10
      runOnce: false
```

CCE 集群节点自动伸缩

自动伸缩概述

CCE 的运行基于一组百度智能云服务器组成的集群，集群为用户的容器运行提供必要的基础资源，如CPU、内存和磁盘等。通常集群的规模在创建 CCE 服务的时候由用户定义，在使用 CCE 过程中也可以随时对集群进行扩容或者缩容。但是当用户的服务增长速度超过预期或者出现波动峰值时，集群提供的资源可能会不足以支撑服务所需，导致服务运行变慢。

通过开启 CCE 的自动伸缩功能，集群将在资源不足时自动创建新的节点，当资源富余时自动释放多余的节点，从而保障集群资源始终足以支撑业务负载，同时最大化节约成本。用户在开启自动伸缩功能时，还可以设置扩缩容的最大最小节点数，从而确保扩缩容在预期的范围内进行。

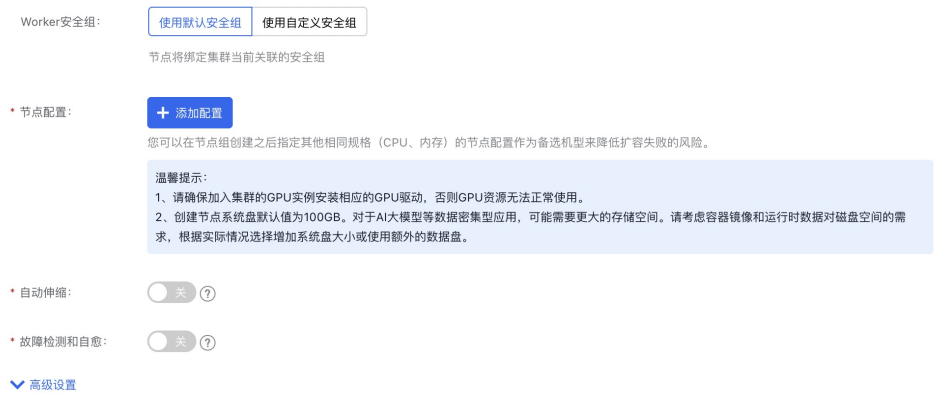
概念解释

概念	说明
伸缩组	指具有相同配置的节点的集合，基于该组的机器配置进行自动扩容、自动缩容
伸缩组 min	伸缩组符合缩容条件时，保证缩容后的伸缩组节点数目不低于该值
伸缩组 max	伸缩组符合扩容条件时，保证扩容后的伸缩组节点数目不高于该值
缩容阈值	伸缩组内节点资源（cpu、mem）的请求资源（Request）与资源容量（Capacity）的比值均低于设定阈值时，集群可能会触发自动缩容。
缩容触发时延	在配置的缩容触发时延内，节点资源利用率持续低于缩容阈值，集群可能会触发自动缩容。
最大并发缩容数	该值为整数，表示并发缩容资源利用率为0的节点数目。
扩容后缩容启动间隔	该值为整数，单位分钟，扩容出来的节点经过此间隔后开始评估是否可以被缩容。
使用本地存储的pod	缩容时可选择跳过包含本地存储 pod的节点
kube-system命名空间下的pod	缩容时可选择跳过kube-system命名空间下的非DaemonSet pod的节点
多伸缩组扩容选择策略	random：从满足扩容条件的伸缩组中随机选择一个伸缩组。least-waste：在满足pod的需求的同时，选择剩余最少资源的伸缩组。 most-pods：在扩容时选择能够调度最多pod的伸缩组

集群操作指导（集群 ID cce-开头）

新建节点组 登录百度智能云管理控制台，进入 "产品服务->云原生->容器引擎 CCE"，在左侧导航栏，点击 "节点管理->节点组"，进入节点组页面，新建节点组。

开启自动扩缩容 创建节点组页面中，点击"自动伸容"。



集群操作指导（集群 ID cce-开头）

配置集群的自动伸缩

1. 登录百度智能云管理控制台，进入 "产品服务->云原生->容器引擎 CCE"，在左侧导航栏，点击 "节点管理->节点组"，进入节点组页面进行

配置。

2. 进入节点组页面。 新集群首次配置，需要在节点组配置中“点击授权”，开通集群自动伸缩功能并进行配置。

上图中扩容算法各选项说明：

- 随机：当有多个伸缩组可选时，随机选择一个伸缩组进行扩容
- least-waste：当有多个伸缩组可选时，选择部署完 pending pods 后空闲 CPU 最少的伸缩组
- most-pods：当有多个伸缩组可选时，选择能够部署最多 pods 的伸缩组
- priority：当有多个伸缩组可选时，选择优先级最高的伸缩组进行扩容，优先级数值越大越优先

更详情说明可参考：https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-expanders

自动缩容常见问题

1. 什么情况下触发扩容？

- 集群中存在因为资源（cpu、内存）不足而处于pending的pod
- 伸缩组的节点未达到max值

2. 为什么有时候伸缩组内节点无法缩容至0？

- 每次修改配置，伸缩组件都会删除、重启，伸缩组组件被调度到伸缩组内的节点时，伸缩组内的该节点不会被缩容。（见问题5）

3. 为什么刚扩容出来的机器满足条件不会被缩容？

- 刚扩容出的机器会有10分钟的保护时间，过了十分钟后，才会考虑缩容。

4. 为什么修改配置后没有立刻生效？

- 修改配置（组配置、缩容配置）后伸缩组件会重启，伸缩组内已经存在的节点会被标识成刚扩容出来的节点，重新等待10分钟（时间可调节）后，节点才会匹配缩容条件。

5. 为什么符合缩容阈值与缩容时间机器没有被缩容？

- 首先判断组内的节点数是否达到设定的min

- 该机器上的pod是否可以调度到其他节点，不能，该机器也不会被缩容
- 是否设置了该节点为不可缩容（"cluster-autoscaler.kubernetes.io/scale-down-disabled": "true"）
- 这个节点是否是刚扩容出来的节点（-scale-down-delay-after-add，默认设置10分钟，意思是刚扩容出来的节点10分钟内不会被判定缩容）
- 这个组过去3分钟是否出现过扩容失败（--scale-down-delay-after-failure，该参数可设置）
- 是否设置了--scale-down-delay-after-delete（相邻两次缩容的间隔） 和 --scan-interval（扫描间隔）这两个参数

6. 如何在集群内查看伸缩组件的状态

- 查看configMap ->`kubectl get configmap cluster-autoscaler-status -n kube-system -o yaml`
- 查看自动伸缩log -> `kubectl logs $(kubectl get pods -n kube-system | grep cluster-autoscaler | awk '{print $1}') -n kube-system -f`

7. 为什么伸缩组件显示的使用率跟我计算的不一样？

- CCE会对机器资源进行一定的保留，伸缩组件计算的CPU核数与内存数是用户可分配的资源数

8. 如何调度pod到指定伸缩组？

- 创建伸缩组时指定伸缩组的label，该伸缩组内的所有节点都有该label，可以通过nodeSelector或者node Affinity调度pod

9. 如何使用GPU伸缩组？

- 创建GPU类型的伸缩组，在创建pod时指定requests、limits nvidia.com/gpu: 1 ，指定pod需要的GPU卡数。这里卡数只能是整数。

注意事项

- 如果服务无法容忍中断，则不建议使用自动扩缩。因为某些 Pod在缩容时可能在其他节点上重启，可能会导致短暂的中断。
- 不要直接修改属于伸缩组的节点，处于同一个伸缩组的节点应该具有相同的机器配置（cpu、内存..），相同的label，相同的系统pod。
- 判断您的账户支持机器实例数是否在符合设置伸缩组的min/max。
- 创建pod时，指定request字段。
- 限制伸缩组组件的使用MEM与CPU资源，伸缩组件默认需求内存300m，CPU 0.1 Core， 且未设置资源限制。为保护您的机器资源，如果您的集群规模较大可以带入如下公式，计算伸缩组件配额。 `MEM = job_num*10KB + pod_num*25KB + 22MB + node_num * 200KB ;CPU = 0.5 Core ~ 1 Core`这里计算的是理想状态下最小需求，当集群有大量pod pending、扩容、缩容时可能会额外需求更多CPU、MEM资源。

使用 cce-autoscaling-placeholder 实现秒级弹性伸缩

组件介绍

使用 CCE 节点组并开启自动扩缩容后，当出现 Pod 由于资源不足导致调度失败时，集群会自动的进行节点扩容，整个流程通常需要若干分钟才能完成，当出现流量突发时，分钟级的扩容速度不能快速满足业务要求。本文介绍通过 K8S 的 PriorityClass 实现 Pod 占位，从而能在 CCE 上实现秒级伸缩，用以应对流量突发的场景。

实现原理

cce-autoscaling-placeholder 利用低优先级的 Pod 对资源进行提前占位，预留部分资源作为缓冲，当需要扩容 Pod 时，高优先级的 Pod 可以快速抢占低优先级 Pod 资源进行调度，而低优先级的 cce-autoscaling-placeholder 的 Pod 则会被 "抢占"，状态变成 Pending，如果配置了节点组并启用弹性伸缩，就会触发节点的扩容，更多信息请见 节点组管理。

这样，由于有了一些资源作为缓冲，即使节点扩容慢，也能保证一些 Pod 能够快速扩容并调度上，实现秒级伸缩。要调整预留的缓冲资源多少，可根据实际需求调整 cce-autoscaling-placeholder 的 request 或副本数。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品导览>容器>容器引擎**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**Helm>Helm模板**"。
4. 在"百度智能云模板"中选择**cce-autoscaling-placeholder**模板，单击**安装**进行部署。
5. 在模板中完成以下配置。

参数名称	参数含义	说明
replicaCount	Pod 数	默认 3
imageID	镜像名称, 通用 pause	
cpu	单个 Pod 占用 cpu	-
mem	单个 Pod 占用 mem	-
nodeSelector	自定义 nodeSelector	推荐和 InstanceGroup 一致
tolerations	自定义 tolerations	-
affinity	自定义 affinity	-

6. 单击**确定**按钮完成部署。

7. 创建 Nginx Pod，可以发现 Nginx Pod 能快速抢占 Placeholder Pod 从而快速启动，Placeholder 会触发新的节点扩容，如下：

```
NAME                                          READY   STATUS    RESTARTS   AGE
cce-autoscaling-placeholder-cb75b46f4-252sl   0/1     Pending   0          16s
cce-autoscaling-placeholder-cb75b46f4-9l7bk   0/1     Pending   0          16s
cce-autoscaling-placeholder-cb75b46f4-pcgsx   0/1     Pending   0          16s
deployment-example-5b9d78df88-69zvs           1/1     Running   0          16s
deployment-example-5b9d78df88-gt6g6           0/1     Running   0          16s
deployment-example-5b9d78df88-lgmpx           1/1     Running   0          16s
```

参考

- Pod 优先级和抢占

云原生AI

云原生AI概述

云原生 AI 概述

云原生AI基于百度智能云容器引擎（CCE）支持 GPU 显存和算力的共享与隔离，同时集成PaddlePaddle、TensorFlow、Pytorch 等主流深度学习框架，通过对 AI 任务的编排、管理，提供低门槛、高效的深度学习训练服务，帮助企业客户提高 GPU 资源使用效率和提升 AI 训练速度，快速降本增效。

使用流程

第一步（必选）：新建 v1.18 或以上版本的集群，添加带有 GPU 设备的节点；

第二步（必选）：安装云原生AI组件，详情请见组件概述；

第三步（可选）：为 GPU 节点开启显存共享；

第四步（必选）：新建队列，指定资源配额和关联用户，详情请见新建队列；

第五步（必选）：新建任务，提交 AI 训练任务，详情请见新建任务。

GPU/NPU支持清单

目前包括但不限于以下GPU/NPU型号支持显存和算力的共享与隔离，您可以提交工单了解更多：

GPU/NPU卡型号
NVIDIA V100 16GB SXM2
NVIDIA V100 32GB SXM2
NVIDIA T4
NVIDIA A100 80GB SXM
NVIDIA A100 40GB SXM
NVIDIA A800 80GB
NVIDIA A30
NVIDIA A10
昆仑芯 R200

队列管理

新建队列

您可以新建一个队列，为 AI 任务指定运行所需的资源和操作人员。

前提条件

- 您已成功创建一个 Kubernetes 集群,详情请见创建集群。
- 您已成功安装 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件，否则云原生 AI 功能将无法使用。

限制说明

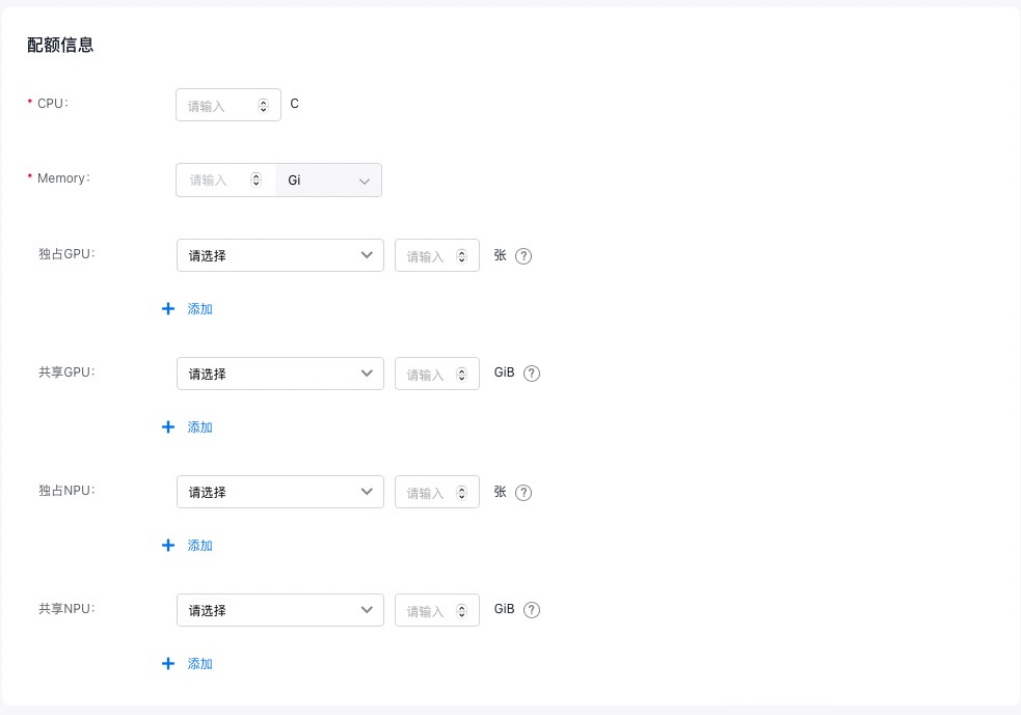
- 为队列分配 GPU 资源时，需要先安装 CCE GPU Manager 组件。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **云原生AI > 队列管理** 。
6. 在队列管理页面单击 **新建队列** 。
7. 在新建队列页面中，完成队列基本信息配置：

- 队列名称：自定义队名名称，长度为 1-63 个字符，只能包含数字、小写字母和“-”，且首尾只能是字母或数字。
- 资源超发：当队列资源不足时，可超发到集群中运行，当集群中有高优任务需要运行，会抢占超发任务的运行资源。
- 备注：自定义队列的备注信息，长度不超过 300 个字符。

8. 完成队列配额信息配置：



- CPU：为队列指定 CPU 资源大小，输入限制为大于 0 的数且最多保留两位小数。

- Memory：为队列指定内存资源大小，输入限制为大于 0 的数且最多保留两位小数，单位支持 Gi、Mi 切换。
- 独占 GPU：为队列指定按整张卡分配的 GPU 资源，输入限制为大于 0 的整数。
- 共享 GPU：为队列指定按显存大小分配的 GPU 资源，输入限制为大于 0 的整数。
- 独占 NPU：为队列指定按整张卡分配的 NPU 资源，输入限制为大于 0 的整数。
- 共享 NPU: 为队列指定按显存大小分配的 NPU 资源，输入限制为大于 0 的整数.

> 注意：指定 GPU 和 NPU 资源时需要先安装 CCE GPU Manager 和 CCE NPU Manager 组件

9. 完成资源对象配置。

- 对象类型：为队列指定可使用资源对象的类型，目前仅支持“用户”。
- 对象选择：为队列指定可使用该队列资源的子用户，支持多选。

10. 点击“确定”按钮，完成队列的新建。

删除队列

您可以通过容器引擎管理控制台删除不再需要的队列。

前提条件

- 您已成功安装 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件，否则云原生 AI 功能将无法使用。
- 您已成功创建队列。

限制说明

- 系统创建的 default 队列不支持删除。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **云原生AI > 队列管理** 。
6. 在队列管理页面中，单击目标队列的 **删除** 。
7. 在确认删除弹出框中，单击“确定”按钮完成队列删除。

> 注意：删除队列并不会删除队列关联的任务。

修改队列

您可以根据新的需要修改已有的队列的备注信息、配额信息和资源对象。

前提条件

- 您已成功安装 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件，否则云原生 AI 功能将无法使用。
- 您已成功创建队列。

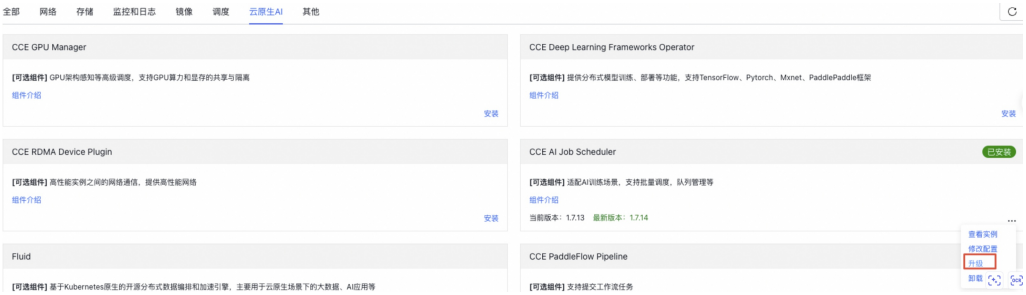
操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **云原生AI > 队列管理** 。
6. 在队列管理页面中，单击目标队列的 **修改** 。
7. 在修改队列页面中，更新队列配置。
8. 点击“确定”按钮，完成修改队列。

逻辑队列和物理队列使用说明

队列功能概述

队列是一个资源池中部分资源的集合，用于工作负载（训练任务、模型服务）的运行。用户可将资源池划分成若干个独立队列（逻辑队列、物理队列）来运行不同工作负载。资源池创建成功后会默认生成 default 队列。本文将介绍如何通过yaml模板创建、更新和使用队列。 **前提条件** 在 CCE 组件管理页面，选择云原生 AI 栏目，CCE AI Job Scheduler 组件版本按照下图升级到最新版本（1.7.14 以上），以使用相关队列功能。

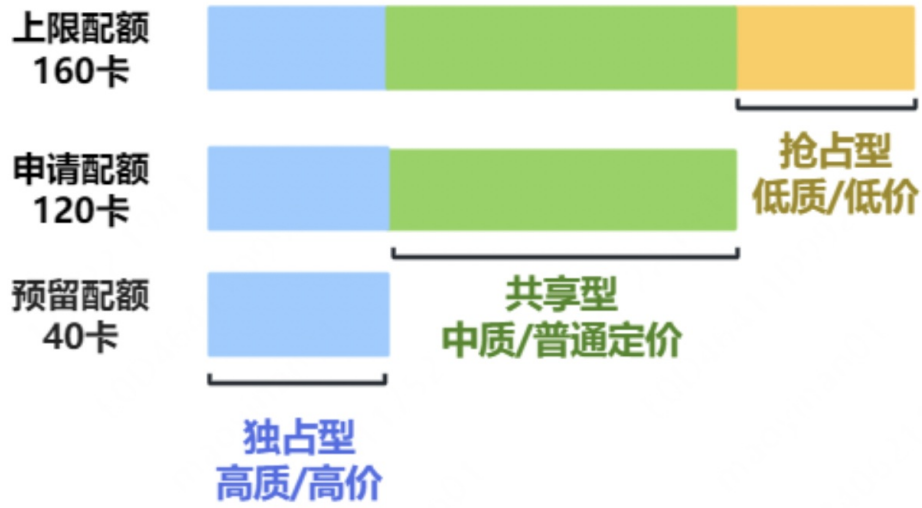


逻辑队列 普通队列 普通队列通常是为不同业务部门分配资源配额最常见的队列形式，是将固定数量的配额资源分配给不同部门使用，实现基础的资源分配能力。 **弹性队列** 弹性队列是帮助平台型客户更高效为不同业务部门分配资源配额的队列形式。在弹性队列中允许队列闲置配额资源借出给其他队列使用，实现资源出借回收、任务超发抢占等能力，以提高GPU/NPU资源的利用率。具体的任务超发、抢占资源的策略见附录详情。 **层级队列** 层级队列是面向中大型客户，组织架构复杂，存在多层的资源配额管理场景，提供多层级、多队列资源配额管理能力。 **物理队列** 通过物理队列对GPU/NPU资源进行物理隔离，支持客户保障重点业务的资源供给，避免资源抢占。同时在物理队列下可以创建多个子逻辑队列。

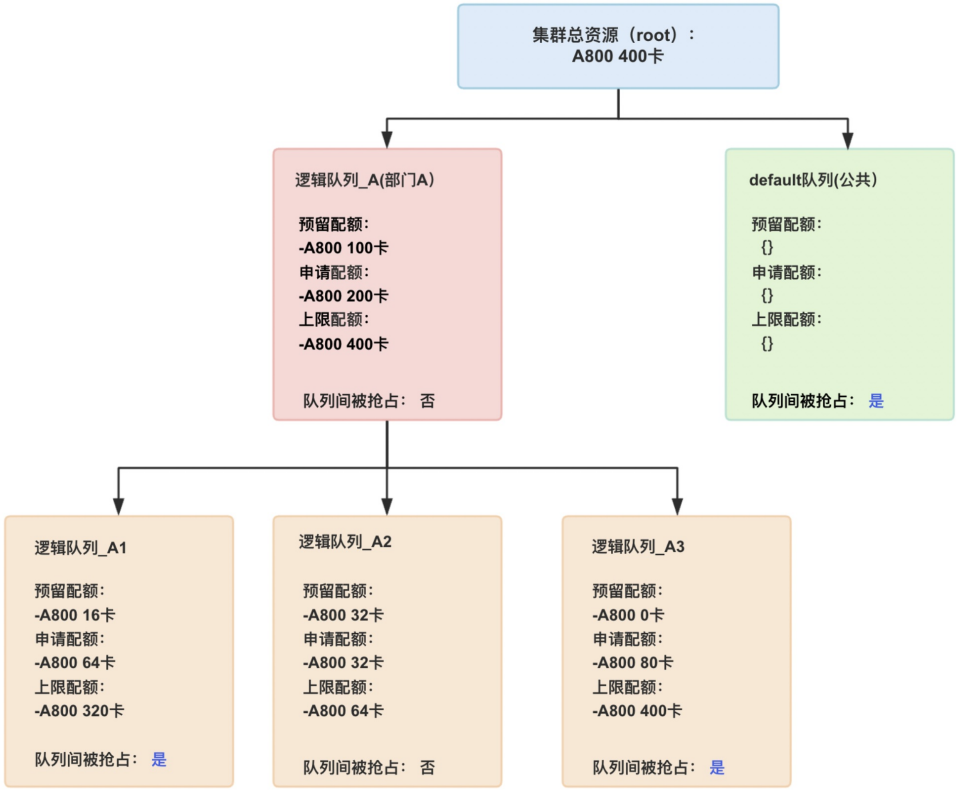
队列功能说明

配额设置说明 弹性队列需要设置三类配额：预留配额、申请配额、上限配额。将计算资源分成独占型、共享型和抢占型，以更好地优化成本并提高GPU/NPU资源的利用率。

- 预留配额表示单独为该队列预留的资源量，任何情况都不会被其他队列借用。
- 申请配额表示该队列申请的资源量，一般为经过确认的预估使用量。开启队列间抢占功能时，其他队列可使用该队列中空闲的共享型资源。
- 上限配额表示该队列最大的可用资源量。



针对以上配额的概念，可参考以下示例便于理解：



队列参数项说明

参数	参数类型	说明	
name	Metadata	队列名称，新队列名称不允许为root	
labels	Metadata	该队列所属的父队列名称，创建队列时禁止将 default队列 和 层级队列的第二层队列 配置为父队列	
Guarantee	Spec	预留资源，表示队列预留的资源量，即使未分配也不会借用，子队列预留资源之和<=父队列预留资源<=集群资源。默认不配置，可使用全集群空闲资源	
Deserved	Spec	申请资源，表示队列申请的资源量，子队列申请资源之和<=父队列申请资源<=集群资源。默认不配置，可使用全集群空闲资源	
Capability	Spec	最大可用资源，表示队列可用的最大资源量，子队列上限配额<=父队列上限配额<=集群资源。默认不配置	
Reclaimable	Spec	是否开启队列间抢占策略，借用的资源是否接受被抢占。默认开启	
State	Status	队列的状态，默认Open；支持配置Close关闭；队列关闭后，将拒绝新任务提交至队列	
baidu.com/elastic-queue-is-parent	Annotations	当前队列是否为父队列，默认为否，不支持直接向父队列提交任务，且当队列有任务运行时不允许创建子队列	
baidu.com/elastic-queue-parent	Annotations	可以指定父队列的名称，默认指向集群根队列 Root，不可指定为default队列、层级队列中的第二级队列以及正在运行任务的队列	

队列注意事项说明 创建队列

- 系统根队列未初始化，则不允许创建/修改队列
- 不允许创建同名队列
- 未指定父队列，默认将 Root 队列设置为父队列
- 队列配额遵循以下的限制：
  - 不允许申请负数配额
  - 队列需保障 capability >= deserved >= guarantee 配置
  - capability 申请配额不允许大于父队列 capability 配额
  - 集群内同一级队列deserved 或 guarantee 之和不允许大于父队列
- 层级队列遵循以下限制：
  - 指定父队列不允许不存在
  - 指定default 队列为父队列
  - 指定第二层队列为父队列
  - 不允许向已存在运行任务的队列添加自队列
- 物理队列遵循以下限制： * capability、deserved 和 guarantee 配置需相同 **删除队列**
- 层级队列中不允许直接删除父队列，需要先删除子队列后，再删除父队列；
- 有任务正在运行的队列不允许删除。 **更新队列**
- 物理队列不允许转换为普通队列
- 物理队列下的逻辑队列的物理队列绑定标签，不可修改
- capability 小于 allocated 已分配资源
- 修改父队列后
  - capability 申请配额小于子队列 capability 配额
  - 父队列小于子队列 deserved 或 guarantee 之和
- 违反了创建队列中的规则

yaml模板操作步骤

逻辑队列使用 创建弹性队列和层级队列

1. 准备创建队列 yaml，样例如下

```
 apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: demo-queue
  labels:
    baidu.com/queue-parent: xxxx
spec:
  capability:
    baidu.com/a800_80g_cgpu: "6"
  guarantee:
    resource:
      baidu.com/a800_80g_cgpu: "1"
  deserved:
    baidu.com/a800_80g_cgpu: "2"
  reclaimable: false
status:
  allocated:
    cpu: "0"
    memory: "0"
  reservation: {}
  state: Open
```

2. 提交yaml在集群中创建队列

`kubectl apply -f demo-queue.yaml` **更新逻辑队列** 运行如下命令修改配列配置：

`kubectl edit queue demo-queue`

提交任务到逻辑队列

通过YAML提交任务，验证队列效果。

```
 apiVersion: "kubeflow.org/v1"
kind: PyTorchJob
metadata:
  name: training-task
spec:
  runPolicy:
    schedulingPolicy:
      queue: "queue-name" //指定队列
      priorityClass: "normal"
  pytorchReplicaSpecs:
    Worker:
      replicas: 2
      template:
        spec:
          tolerations:
          - key: "kwok.x-k8s.io/node"
            operator: "Equal"
            value: "fake"
            effect: "NoSchedule"
          containers:
          - name: pytorch
            image: demo-image:1.0
            imagePullPolicy: IfNotPresent
            resources:
              limits:
                nvidia.com/gpu: 4
          schedulerName: volcano
```

参数	参数类型	说明
name	Meta	任务名称
runPolicy.schedulingPolicy.queue	Spec	向指定的队列提交任务
runPolicy.schedulingPolicy.priorityClass	Spec	运行任务的优先级，支持高、中、低三种
pytorchReplicaSpecs.Worker.replicas	Spec	运行任务的副本数

物理队列使用 划分节点 将目标节点纳入到物理队列中，建议提前排空相关节点的任务，对于未排空的节点也可直接加入物理队列，但节点GPU资源需要等待任务运行完成后才能释放。

需要注意，物理队列资源申请量需要与加入物理队列的节点资源准确匹配。比如物理队列申请资源 16 卡，可以纳入两台 8 卡节点，多划分会造成资源浪费，少划分可能会导致调度失败。

划分方式分成两步：

- 添加污点：只影响后续新任务的调度，不影响当前正在运行的任务。

`kubectl taint nodes targetNode aihc.baidu.com/dedicated-pool=queue-physical-name:NoSchedule`

- 添加标签

`kubectl label nodes targetNode aihc.baidu.com/dedicated-pool=queue-physical-name`

> 注意：添加标签和污点时将targetNode换成节点IP即可，多个IP中间用空格隔开。

创建物理队列 按照下面yaml模板描述一个物理队列

```
 apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
 labels:
   baidu.com/queue-is-physical-queue: "true"
 name: queue-physical-name
spec:
 capability:
   nvidia.com/gpu: "16"
 deserved:
   nvidia.com/gpu: "16"
 guarantee:
   resource:
     nvidia.com/gpu: "16"
 reclaimable: false
 weight: 1
```

> 注意：仅需修改name（队列名称）、capability、deserved、guarantee 4个参数值，且配额相关的capability、deserved、guarantee3个值需保持一致，对应物理队列申请的GPU卡数，其他参数保持不变。

队列下创建逻辑队列（可选） 如果想对物理队列资源进行进一步划分，可以在物理队列下创建逻辑子队列，但与物理队列不同的是，子队列不再绑定节点，只保证配额。

```
 apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
 labels:
   baidu.com/queue-parent: queue-physical-name
 name: queue-logical-name
spec:
 deserved:
   nvidia.com/gpu: "8"
 guarantee:{}
 reclaimable: false
 weight: 1
```

申请固定配额逻辑队列，可以参考上述例子，设置 deserved 即可，保证所有子队列 deserved 资源之和不大于物理队列的 deserved 值。

> 注意，以下两种行为是不合法的：
> * 向有任务运行的物理队列下添加子队列
> * 在物理队列有子队列的情况下，向物理队列提交任务

修改物理队列 （1）修改物理队列配额

使用 `kubectl edit queue queue-physical-name` 修改物理队列配额，修改配额为纳管新节点后的总配额数 （2）纳入新节点 类似划分节点步骤，给新节点执行下面两个操作

- 添加污点

`kubectl taint nodes targetNode aihc.baidu.com/dedicated-pool=queue-physical-name:NoSchedule`

- 添加标签

`kubectl label nodes targetNode aihc.baidu.com/dedicated-pool=queue-physical-name`

任务管理

新建TensorFlow任务

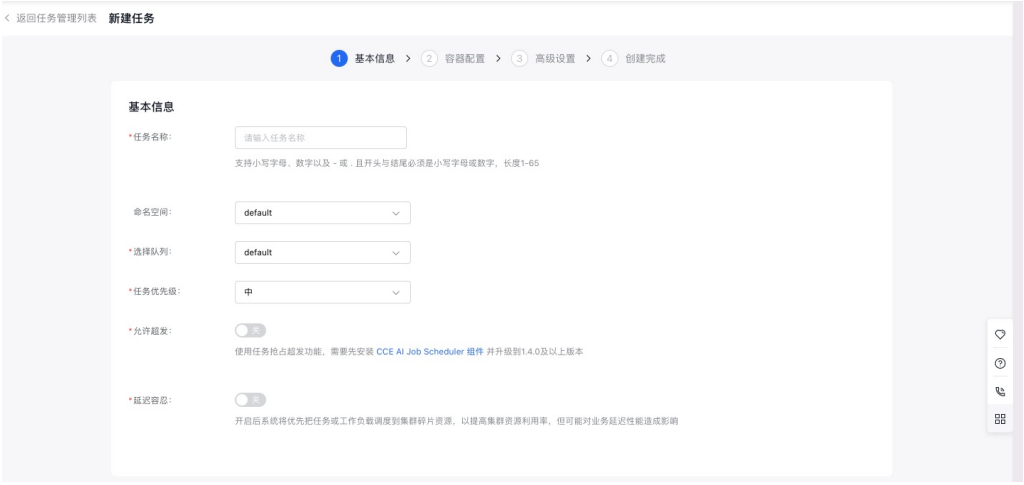
您可以新建一个TensorFlow类型的任务。

前提条件

- 您已成功安装CCE AI Job Scheduler和CCE Deep Learning Frameworks Operator组件，否则云原生AI功能将无法使用。
- 若您是子用户，队列关联的用户中有您才能使用该队列新建任务。
- 安装组件CCE Deep Learning Frameworks Operator时，系统安装了TensorFlow深度学习框架。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 任务管理**。
6. 在任务管理页面单击**新建任务**。
7. 在新建任务页面中，完成任务基本信息配置：



- 任务名称：自定义任务名称，支持小写字母、数字、以及-或.且开头与结尾必须是小写字母或者数字，长度 1-65。
- 命名空间：选择新建任务所在的命名空间。
- 队列：选择新建任务关联的队列。
- 任务优先级：选择任务对应的任务优先级。
- 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。
- 延迟容忍：系统将优先把任务或工作负载调度到集群碎片资源，以提高集群资源利用率，但可能对业务延迟行能造成影响。

8. 完成代码基本信息配置：

- 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”、“git代码仓库“与“暂不配置”。
- 执行命令：指定代码的执行命令。

9.完成数据相关信息配置：

- 设置数据源：当前支持数据集、持久卷声明、选择临时路径和选择主机路径。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

- **选择框架：选择“ TensorFlow ”**。
- 训练方式：指定训练方式为“单机”或“分布式”。
- 选择角色：训练方式为“单机”时，只能选择“Woker”；训练方式为“分布式”时，可额外选择“PS”、“Chief”、“Evaluator”。

12.完成容器组相关信息配置，可以根据需要同时进行高级设置。

- 期望Pod数：指定容器组的Pod数目。
- 重启策略：指定容器组的重启策略，可选择的策略有“失败重启”或“从不重启”。
- 镜像地址：指定容器的镜像拉取地址，也可以直接点击“选择镜像”，选择需要使用的镜像。
- 镜像版本：指定镜像的版本，若不指定默认拉取latest版。
- 容器配额：指定容器的CPU、内存、GPU/NPU资源相关信息。
- 环境变量：填写变量名和变量值。
- 生命周期：包含启动命令、启动参数、启动后执行和停止前执行，可根据需要添加。

13.完成任务高级信息相关配置。

- 私有仓库凭证：若需要使用私有镜像仓库，请在此处添加对应镜像仓库的访问凭证。
- Tensorboard：若需要任务可视化时，可开启Tensorboard功能，开启后需要指定“服务类型”与“ 训练日志读取路径”。
- K8S标签：指定任务对应的K8S Label。
- 注释：指定任务对应的Annotation。

14. 点击“完成”按钮，完成任务的新建。

Yaml创建任务示例

```
 apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
 name: "tfjob-dist-mnist-for-e2e-test"
spec:
 tfReplicaSpecs:
  PS:
   replicas: 2
   restartPolicy: Never
   template:
    metadata:
     annotations:
      sidecar.istio.io/inject: "false"
      # if your libcuda.so.1 is in custom path, set the correct path with the following annotation
      # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64
    spec:
     schedulerName: volcano
     containers:
      - name: tensorflow
        image: registry.baidubce.com/cce-public/kubeflow/tf-dist-mnist-test:1.0
        resources:
         requests:
          cpu: 1
          memory: 1Gi
         limits:
          baidu.com/v100_32g_cgpu: "1"
          # for gpu core/memory isolation
          baidu.com/v100_32g_cgpu_core: 10
          baidu.com/v100_32g_cgpu_memory: "2"
        # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown.
        # ${'`'}dist_mnist.py${'`'} needs to be replaced with the name of your gpu process.
        lifecycle:
         preStop:
          exec:
           command: [
            "/bin/sh", "-c",
            "kill -10 `ps -ef | grep dist_mnist.py | grep -v grep | awk '{print $2}'` && sleep 1"
           ]
  Worker:
   replicas: 4
   restartPolicy: Never
   template:
    metadata:
     annotations:
      sidecar.istio.io/inject: "false"
      # if your libcuda.so.1 is in custom path, set the correct path with the following annotation
      # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64
    spec:
     schedulerName: volcano
     containers:
      - name: tensorflow
        image: registry.baidubce.com/cce-public/kubeflow/tf-dist-mnist-test:1.0
        env:
         # for gpu memory over request, set 0 to disable
         - name: CGPU_MEM_ALLOCATOR_TYPE
           value: “1”
        resources:
         requests:
          cpu: 1
          memory: 1Gi
         limits:
          baidu.com/v100_32g_cgpu: "1"
          # for gpu core/memory isolation
          baidu.com/v100_32g_cgpu_core: 20
          baidu.com/v100_32g_cgpu_memory: "4"
        # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown.
```

```
            # ${'`'}dist_mnist.py${'`'} needs to be replaced with the name of your gpu process.
            lifecycle:
              preStop:
                exec:
                  command: [
                    "/bin/sh", "-c",
                    "kill -10 `ps -ef | grep dist_mnist.py | grep -v grep | awk '{print $2}'` && sleep 1"
                  ]
```

新建Pytorch任务

您可以新建一个Pytorch类型的任务。

前提条件

* 您已成功安装CCE AI Job Scheduler和CCE Deep Learning Frameworks Operator组件，否则云原生AI功能将无法使用。

* 若您是子用户，队列关联的用户中有您才能使用该队列新建任务。

* 安装组件CCE Deep Learning Frameworks Operator时，系统安装了Pytorch深度学习框架。

操作步骤

1. 登录[百度智能云官网](https://cloud.baidu.com/)，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。

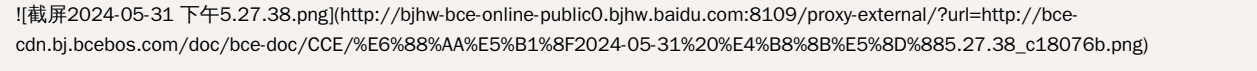
3. 单击左侧导航栏中的**集群管理 > 集群列表**。

4. 在集群列表页面中，单击目标集群名称进入集群管理页面。

5. 在集群管理页面单击**云原生AI > 任务管理**。

6. 在任务管理页面单击**新建任务**。

7. 在新建任务页面中，完成任务基本信息配置：



* 任务名称：自定义任务名称，支持小写字母、数字、以及-或.且开头与结尾必须是小写字母或者数字，长度 1-65。

* 命名空间：选择新建任务所在的命名空间。

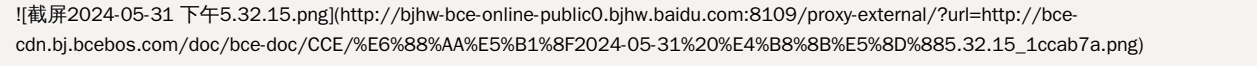
* 队列：选择新建任务关联的队列。

* 任务优先级：选择任务对应的任务优先级。

* 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。

* 延迟容忍：系统将优先把任务或工作负载调度到集群碎片资源，以提高集群资源利用率，但可能对业务延迟行能造成影响。

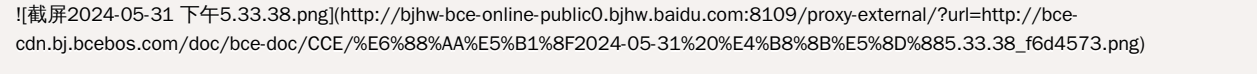
8. 完成代码基本信息配置：



* 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”、“git代码仓库“与“暂不配置”。

* 执行命令：指定代码的执行命令。

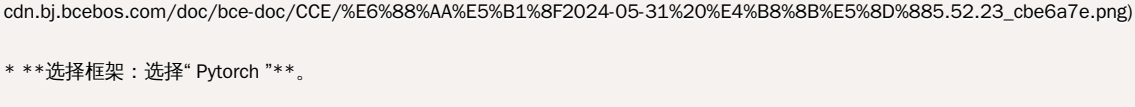
9.完成数据相关信息配置：



* 设置数据源：当前支持数据集、持久卷声明、选择临时路径和选择主机路径。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

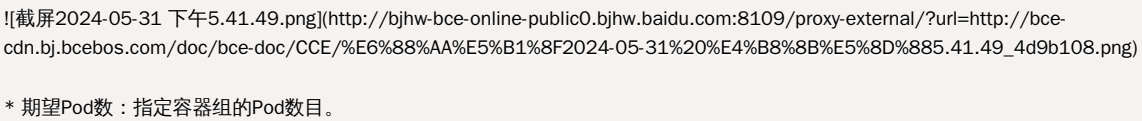


* **选择框架：选择“ Pytorch ”**。

* 训练方式：指定训练方式为“单机”或“分布式”。

* 选择角色：训练方式为“单机”时，只能选择“Woker”；训练方式为“分布式”时，可额外选择“PS”、“Chief”、“Evaluator”。

12.完成容器组相关信息配置，可以根据需要同时进行高级设置。



* 期望Pod数：指定容器组的Pod数目。

* 重启策略：指定容器组的重启策略，可选择的策略有“失败重启”或“从不重启”。

* 镜像地址：指定容器的镜像拉取地址，也可以直接点击“选择镜像”，选择需要使用的镜像。

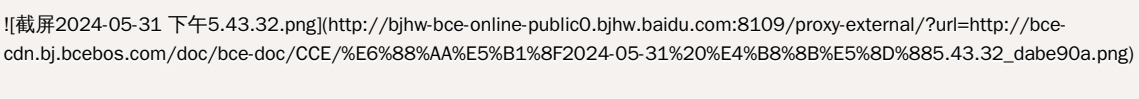
* 镜像版本：指定镜像的版本，若不指定默认拉取latest版。

* 容器配额：指定容器的CPU、内存、GPU/NPU资源相关信息。

* 环境变量：填写变量名和变量值。

* 生命周期：包含启动命令、启动参数、启动后执行和停止前执行，可根据需要添加。

13.完成任务高级信息相关配置。



* 私有仓库凭证：若需要使用私有镜像仓库，请在此处添加对应镜像仓库的访问凭证。

* Tensorboard：若需要任务可视化时，可开启Tensorboard功能，开启后需要指定“服务类型”与“
训练日志读取路径”。

* K8S标签：指定任务对应的K8S Label。

* 注释：指定任务对应的Annotation。

14. 点击“完成”按钮，完成任务的新建。

Yaml创建任务示例

```json
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
  name: "pytorch-dist-mnist-gloo"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
            # if your libcuda.so.1 is in custom path, set the correct path with the following annotation
            # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64
        spec:
          schedulerName: volcano
          containers:
```

```
          - name: pytorch
            image: registry.baidubce.com/cce-public/kubeflow/pytorch-dist-mnist-test-with-data:1.0
            args: ["--backend", "gloo"]
            # Comment out the below resources to use the CPU.
            resources:
              requests:
                cpu: 1
                memory: 1Gi
              limits:
                baidu.com/v100_32g_cgpu: "1"
                # for gpu core/memory isolation
                baidu.com/v100_32g_cgpu_core: 10
                baidu.com/v100_32g_cgpu_memory: "2"
            # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown.
            # `mnist.py` needs to be replaced with the name of your gpu process.
            lifecycle:
              preStop:
                exec:
                  command: [
                    "/bin/sh", "-c",
                    "kill -10 `ps -ef | grep mnist.py | grep -v grep | awk '{print $2}'` && sleep 1"
                  ]
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          schedulerName: volcano
          containers:
            - name: pytorch
              image: registry.baidubce.com/cce-public/kubeflow/pytorch-dist-mnist-test-with-data:1.0
              env:
                # for gpu memory over request, set 0 to disable
                - name: CGPU_MEM_ALLOCATOR_TYPE
                  value: “1”
              args: ["--backend", "gloo"]
              resources:
                requests:
                  cpu: 1
                  memory: 1Gi
                limits:
                  baidu.com/v100_32g_cgpu: "1"
                  # for gpu core/memory isolation
                  baidu.com/v100_32g_cgpu_core: 20
                  baidu.com/v100_32g_cgpu_memory: "4"
              # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown.
              # `mnist.py` needs to be replaced with the name of your gpu process.
              lifecycle:
                preStop:
                  exec:
                    command: [
                      "/bin/sh", "-c",
                      "kill -10 `ps -ef | grep mnist.py | grep -v grep | awk '{print $2}'` && sleep 1"
                    ]
```

新建Mxnet任务

您可以新建一个Mxnet类型的任务。

前提条件

* 您已成功安装CCE AI Job Scheduler和CCE Deep Learning Frameworks Operator组件，否则云原生AI功能将无法使用。

* 若您是子用户，队列关联的用户中有您才能使用该队列新建任务。

* 安装组件CCE Deep Learning Frameworks Operator时，系统安装了Mxnet深度学习框架。

操作步骤

1. 登录[百度智能云官网](https://cloud.baidu.com/)，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。

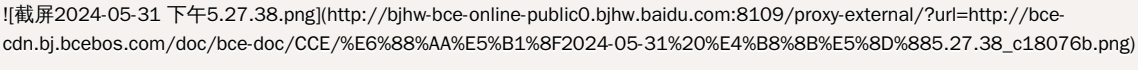
3. 单击左侧导航栏中的**集群管理 > 集群列表**。

4. 在集群列表页面中，单击目标集群名称进入集群管理页面。

5. 在集群管理页面单击**云原生AI > 任务管理**。

6. 在任务管理页面单击**新建任务**。

7. 在新建任务页面中，完成任务基本信息配置：



* 任务名称：自定义任务名称，支持小写字母、数字、以及-或.且开头与结尾必须是小写字母或者数字，长度 1-65。

* 命名空间：选择新建任务所在的命名空间。

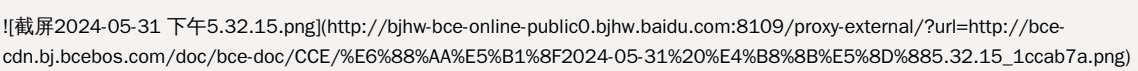
* 选择队列：选择新建任务关联的队列。

* 任务优先级：选择任务对应的任务优先级。

* 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。

* 延迟容忍：系统将优先把任务或工作负载调度到集群碎片资源，以提高集群资源利用率，但可能对业务延迟行能造成影响。

8. 完成代码基本信息配置：



* 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”、“git代码仓库“与“暂不配置”。

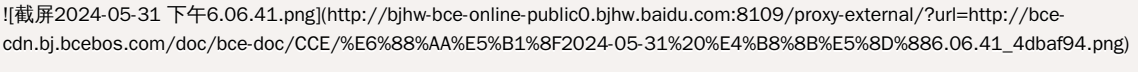
* 执行命令：指定代码的执行命令。

9.完成数据相关信息配置：

* 设置数据源：当前支持数据集、持久卷声明、选择临时路径和选择主机路径。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

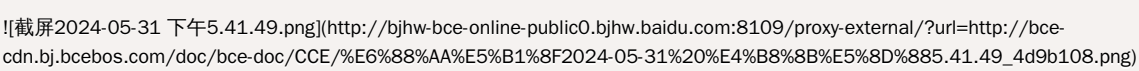


* **选择框架：选择“ Mxnet ”**。

* 训练方式：指定训练方式为“单机”或“分布式”。

* 选择角色：训练方式为“单机”时，只能选择“Woker”；训练方式为“分布式”时，可额外选择“PS”、“Chief”、“Evaluator”。

12.完成容器组相关信息配置，可以根据需要同时进行高级设置。



* 期望Pod数：指定容器组的Pod数目。

* 重启策略：指定容器组的重启策略，可选择的策略有“失败重启”或“从不重启”。

* 镜像地址：指定容器的镜像拉取地址，也可以直接点击“选择镜像”，选择需要使用的镜像。

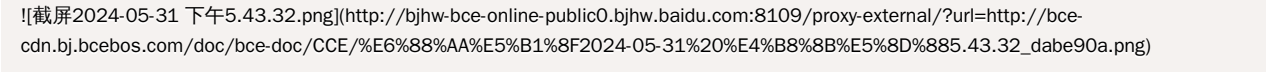
* 镜像版本：指定镜像的版本，若不指定默认拉取latest版。

* 容器配额：指定容器的CPU、内存、GPU/NPU资源相关信息。

* 环境变量：填写变量名和变量值。

* 生命周期：包含启动命令、启动参数、启动后执行和停止前执行，可根据需要添加。

13.完成任务高级信息相关配置。



* 私有仓库凭证：若需要使用私有镜像仓库，请在此处添加对应镜像仓库的访问凭证。

* Tensorboard：若需要任务可视化时，可开启Tensorboard功能，开启后需要指定“服务类型”与“训练日志读取路径”。

* K8S标签：指定任务对应的K8S Label。

* 注释：指定任务对应的Annotation。

14. 点击“完成”按钮，完成任务的新建。

Yaml创建任务示例

```
apiVersion: "kubeflow.org/v1" kind: "MXJob" metadata: name: "mxnet-job" spec: jobMode: MXTrain mxReplicaSpecs: Scheduler: replicas: 1 restartPolicy: Never template: metadata: annotations: sidecar.istio.io/inject: "false" # if your libcuda.so.1 is in custom path, set the correct path with the following annotation # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64 spec: schedulerName: volcano containers: - name: mxnet image: registry.baidubce.com/cce-public/kubeflow/cce-public/mxjob/mxnet:gpu resources: limits: baidu.com/v100_32g_cgpu: "1" # for gpu core/memory isolation baidu.com/v100_32g_cgpu_core: 5 baidu.com/v100_32g_cgpu_memory: "1" # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown. # ${''}train_mnist.py${''} needs to be replaced with the name of your gpu process. lifecycle: preStop: exec: command: [ "/bin/sh", "-c", "kill -10 ${''}ps -ef | grep train_mnist.py | grep -v grep | awk '{print $2}'${''} && sleep 1" ] Server: replicas: 1 restartPolicy: Never template: metadata: annotations: sidecar.istio.io/inject: "false" # if your libcuda.so.1 is in custom path, set the correct path with the following annotation # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64 spec: schedulerName: volcano containers: - name: mxnet image: registry.baidubce.com/cce-public/kubeflow/cce-public/mxjob/mxnet:gpu resources: limits: baidu.com/v100_32g_cgpu: "1" # for gpu core/memory isolation baidu.com/v100_32g_cgpu_core: 5 baidu.com/v100_32g_cgpu_memory: "1" # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown. # ${''}train_mnist.py${''} needs to be replaced with the name of your gpu process. lifecycle: preStop: exec: command: [ "/bin/sh", "-c", "kill -10 ${''}ps -ef | grep train_mnist.py | grep -v grep | awk '{print $2}'${''} && sleep 1" ] Worker: replicas: 1 restartPolicy: Never template: metadata: annotations: sidecar.istio.io/inject: "false" # if your libcuda.so.1 is in custom path, set the correct path with the following annotation # kubernetes.io/baidu-cgpu.nvidia-driver-lib: /usr/lib64 spec: schedulerName: volcano containers: - name: mxnet image: registry.baidubce.com/cce-public/kubeflow/cce-public/mxjob/mxnet:gpu env: # for gpu memory over request, set 0 to disable - name: CGPU_MEM_ALLOCATOR_TYPE value: “1” command: ["python"] args: [ "/incubator-mxnet/example/image-classification/train_mnist.py", "--num-epochs","10","--num-layers","2","--kv-store","dist_device_sync","--gpus","0" ] resources: requests: cpu: 1 memory: 1Gi limits: baidu.com/v100_32g_cgpu: "1" # for gpu core/memory isolation baidu.com/v100_32g_cgpu_core: 20 baidu.com/v100_32g_cgpu_memory: "4" # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown. # ${''}train_mnist.py${''} needs to be replaced with the name of your gpu process. lifecycle: preStop: exec: command: [ "/bin/sh", "-c", "kill -10 ${''}ps -ef | grep train_mnist.py | grep -v grep | awk '{print $2}'${''} && sleep 1" ]
```

新建PaddlePaddle任务

您可以新建一个PaddlePaddle类型的任务。

前提条件

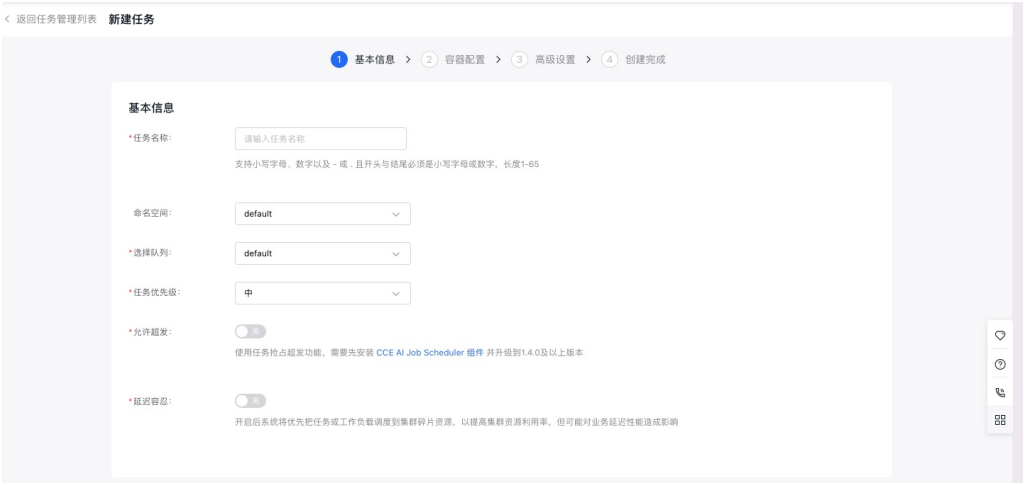
- 您已成功安装CCE AI Job Scheduler和CCE Deep Learning Frameworks Operator组件，否则云原生AI功能将无法使用。
- 若您是子用户，队列关联的用户中有您才能使用该队列新建任务。
- 安装组件CCE Deep Learning Frameworks Operator时，系统安装了PaddlePaddle深度学习框架。

限制说明

- 目前PaddlePaddle类型的任务不支持GPU显存共享。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 任务管理**。
6. 在任务管理页面单击**新建任务**。
7. 在新建任务页面中，完成任务基本信息配置：



- 任务名称：自定义任务名称，支持小写字母、数字、以及-或.且开头与结尾必须是小写字母或者数字，长度 1-65。
- 命名空间：选择新建任务所在的命名空间。
- 队列：选择新建任务关联的队列。
- 任务优先级：选择任务对应的任务优先级。
- 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。
- 延迟容忍：系统将优先把任务或工作负载调度到集群碎片资源，以提高集群资源利用率，但可能对业务延迟行能造成影响。

8. 完成代码基本信息配置：

- 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”与“暂不配置”。
- 执行命令：指定代码的执行命令。

9.完成数据相关信息配置：

- 设置数据源：当前同时支持数据集和持久卷声明。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

- **选择框架：选择“ PaddleJob ”**。
- 训练方式：指定训练方式为“单机”或“分布式”。
- 选择角色：训练方式为“单机”时，只能选择“Woker”；训练方式为“分布式”时，可额外选择“PS”。

12.完成容器组相关信息配置，可以根据需要同时进行高级设置。

- 期望Pod数：指定容器组的Pod数目。
- 重启策略：指定容器组的重启策略，可选择的策略有“失败重启”或“从不重启”。
- 镜像地址：指定容器的镜像拉取地址，也可以直接点击“选择镜像”，选择需要使用的镜像。
- 镜像版本：指定镜像的版本，若不指定默认拉取latest版。
- 容器配额：指定容器的CPU、内存、GPU资源相关信息。

- 环境变量：填写变量名和变量值。
- 生命周期：包含启动命令、启动参数、启动后执行和停止前执行，可根据需要添加。

13.完成任务高级信息相关配置。

- 最大训练时长：指定允许的最大训练时长，若不指定，则为不限制时间。
- 私有仓库凭证：若需要使用私有镜像仓库，请在此处添加对应镜像仓库的访问凭证。
- Tensorboard：若需要任务可视化时，可开启Tensorboard功能，开启后需要指定“服务类型”与“ 训练日志读取路径”。
- K8S标签：指定任务对应的K8S Label。
- 注释：指定任务对应的Annotation。

14. 点击“完成”按钮，完成任务的新建。

Yaml创建任务示例

```
 apiVersion: batch.paddlepaddle.org/v1
kind: PaddleJob
metadata:
 name: resnet
spec:
 cleanPodPolicy: Never
 worker:
  replicas: 2
  template:
   spec:
    schedulerName: volcano
    containers:
     - name: resnet
       image: registry.baidubce.com/cce-public/kubeflow/paddle-operator/demo-resnet:v1
       env:
         # for gpu memory over request, set 0 to disable
         - name: CGPU_MEM_ALLOCATOR_TYPE
           value: “1”
       command:
         - python
       args:
         - "-m"
         - "paddle.distributed.launch"
         - "train_fleet.py"
       volumeMounts:
         - mountPath: /dev/shm
           name: dshm
       resources:
         requests:
           cpu: 1
           memory: 2Gi
         limits:
           baidu.com/v100_16g_cgpu: "1"
    volumes:
     - name: dshm
       emptyDir:
```

```
    emptyDir:
      medium: Memory
```

新建AITraining任务

您可以新建一个AITraining类型的任务。AITrainingJob是一种优化的AI训练作业任务，在算力浪费、资源供给、容错与弹性调度上进行了优化改进。

前提条件

* 您已成功安装CCE AI Job Scheduler和CCE Deep Learning Frameworks Operator组件，否则云原生AI功能将无法使用。

* 若您是子用户，队列关联的用户中有您才能使用该队列新建任务。

* 安装组件CCE Deep Learning Frameworks Operator时，系统安装了Pytorch深度学习框架。

操作步骤

1. 登录[百度智能云官网](https://cloud.baidu.com/)，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。

3. 单击左侧导航栏中的**集群管理 > 集群列表**。

4. 在集群列表页面中，单击目标集群名称进入集群管理页面。

5. 在集群管理页面单击**云原生AI > 任务管理**。

6. 在任务管理页面单击**新建任务**。

7. 在新建任务页面中，完成任务基本信息配置：

* 任务名称：自定义任务名称，支持大小写字母、数字、以及-_ /.特殊字符，必须以中文或字母开头，长度 1-65。

* 命名空间：选择新建任务所在的命名空间。

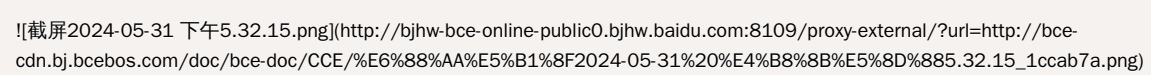
* 选择队列：选择新建任务关联的队列。

* 任务优先级：选择任务对应的任务优先级。

* 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。

* 延迟容忍：系统将优先把任务或工作负载调度到集群碎片资源，以提高集群资源利用率，但可能对业务延迟行能造成影响。

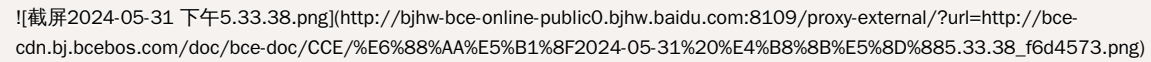
8. 完成代码基本信息配置：



* 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”与“暂不配置”。

* 执行命令：指定代码的执行命令。

9.完成数据相关信息配置：



* 设置数据源：当前支持数据集、持久卷声明、选择临时路径和选择主机路径。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

```
apiVersion: kongming.cce.baiudbce.com/v1 kind: AITrainingJob metadata: name: job-horovod-test namespace: default spec:
# 任务结束时，pod的清理策略，All表示所有pod，none表示不清理
cleanPodPolicy: All
# 完成策略，All表示所有pod完成即任务完成，Any表示任何pod完成即任务完成
completePolicy: Any
# 失败策略，All表示所有pod失败即任务失败，Any表示任何pod完成即任务完成
failPolicy: Any
# 支持horovod与paddle框架
frameworkType: horovod
```

弹性选项，true表示开启弹性，false不开启，开启时需开启trainer容器的容错选项

faultTolerant: true plugin: ssh: - "" discovery: - "" priority: normal replicaSpecs: launcher: completePolicy: Any failPolicy: Any maxReplicas: 1 minReplicas: 1 replicaType: master replicas: 1 restartLimit: 100 restartPolicy: OnNodeFailWithExitCode restartTimeLimit: 60 restartTimeout: 864000 template: metadata: creationTimestamp: null spec: initContainers: - args: - --barrier_roles=trainer - --incluster - --name=$(TRAININGJOB_NAME) - --namespace=$(TRAININGJOB_NAMESPACE) image: registry.baidubce.com/cce-plugin-dev/jobbarrier:v0.9 imagePullPolicy: IfNotPresent name: job-barrier resources: limits: cpu: "1" memory: 1Gi requests: cpu: "1" memory: 1Gi restartPolicy: Never schedulerName: volcano terminationMessagePath: /dev/termination-log terminationMessagePolicy: File securityContext: {} containers: - command: - /bin/bash - -c - horovodrun -np 3 --min-np=1 --max-np=5 --verbose --log-level=DEBUG --host-discovery-script /etc/edl/discover_hosts.sh python /horovod/examples/elastic/pytorch/pytorch_synthetic_benchmark_elastic.py env: image: registry.baidubce.com/cce-plugin-dev/horovod:v0.1.0 imagePullPolicy: Always name: aitj-0 resources: securityContext: capabilities: add: - SYS_ADMIN volumeMounts: - mountPath: /dev/shm name: cache-volume dnsPolicy: ClusterFirstWithHostNet terminationGracePeriodSeconds: 30 volumes: - emptyDir: medium: Memory sizeLimit: 1449Gi name: cache-volume trainer: completePolicy: None failPolicy: None # 容错配置，控制器将会以下面的配置作为容错判断条件进行容错 faultTolerantPolicy: # 程序退出码 - exitCodes: 129,10001,127,137,143,129 restartPolicy: ExitCode restartScope: Pod # 集群异常事件 - exceptionalEvent: "nodeNotReady,PodForceDeleted" restartPolicy: OnNodeFail restartScope: Pod # 开启弹性的最大副本数 maxReplicas: 5 # 开启弹性的最小副本数 minReplicas: 1 replicaType: worker replicas: 3 restartLimit: 100 restartPolicy: OnNodeFailWithExitCode restartTimeLimit: 60 restartTimeout: 864000 template: metadata: creationTimestamp: null spec: containers: - command: - /bin/bash - -c - /usr/sbin/sshd && sleep 40000 image: registry.baidubce.com/cce-plugin-dev/horovod:v0.1.0 imagePullPolicy: Always name: aitj-0 resources: # limit与request需保持一致 limits: baidu/gpu_p40_8: "1" requests: baidu/gpu_p40_8: "1" securityContext: capabilities: add: - SYS_ADMIN volumeMounts: - mountPath: /dev/shm name: cache-volume dnsPolicy: ClusterFirstWithHostNet terminationGracePeriodSeconds: 300 volumes: - emptyDir: medium: Memory sizeLimit: 1449Gi name: cache-volume schedulerName: volcano

查看任务信息

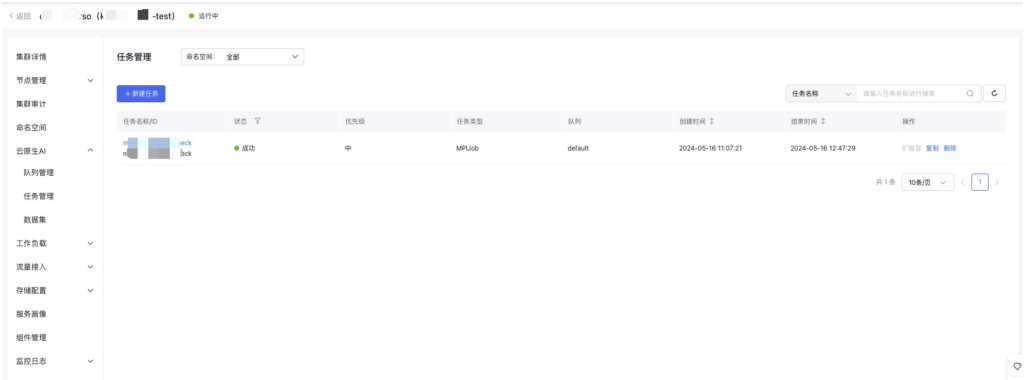
任务创建成功后，您可以在容器引擎控制台查看任务运行情况，包括基本信息、监控信息和日志信息。

前提条件

- 您已成功安装 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件，否则云原生 AI 功能将无法使用。
- 您成功创建任务。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **云原生AI > 任务管理** 。
6. 在任务管理页面您可以查看任务列表信息。



7. 单击目标任务的名称进入“任务详情”页面查看任务的基本信息、数据配置和容器组列表信息。
8. 单击 **任务监控** 您可查看任务和容器组的监控信息。
9. 单击 **任务日志** 您可以查看任务容器的日志信息。

删除任务

您可以通过容器引擎管理控制台删除不再需要的任务。

前提条件

- 您已成功安装 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件，否则云原生 AI 功能将无法使用。
- 您已成功创建任务。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **云原生AI > 任务管理** 。
6. 在任务管理页面中，单击目标任务的 **删除** 。
7. 在确认删除弹出框中，单击“确定”按钮完成任务删除。

基于 NCCL的RDMA分布式训练示例

概述

RDMA（Remote Direct Memory Access）是新一代的网络通信技术，它允许计算机之间直接进行内存对内存的数据传输，而不需要经过操作系统或中央处理器的处理。在大规模的分布式训练中，通过使用RDMA有效解决网络传输中服务器端数据处理的延迟问题，从而实现高吞吐、低延迟的网络通信，提升训练效率。

本文档用于介绍在云原生AI的环境下使用 RDMA 网络进行分布式训练。

> 说明：
> 1、由于 RDMA 网络的特殊性，以下示例在自建k8s集群中可能无法适用。
> 2、IB和RoCE在使用上几乎无区别，如无特别说明以下均以RoCE为例，IB在应用侧无需更改。
> 3、业务镜像中需要使用 nccl 依赖库，这里推荐使用 NVIDIA GPU Cloud（NGC）提供的基础镜像。NGC 提供的基础镜像通常会包含 nccl 依赖库，并且已经预先配置和优化了许多常用的深度学习框架和工具。使用 NGC 基础镜像可以简化您的设置和配置过程，并确保您能够顺利使用 nccl 进行 GPU 加速计算和深度学习任务。

使用前提

- 已经创建集群，且集群中至少有2台具有RDMA网络的GPU实例。
- GPU实例镜像中包含ofed和nvidia驱动，这里推荐使用百度智能云提供的GPU镜像，已包含OFED驱动，无需手动安装。
- 集群已安装 云原生AI CCE RDMA Device Plugin、 CCE GPU Manager 、 CCE AI Job Scheduler 和 CCE Deep Learning Frameworks Operator 组件。

环境验证 登录集群内具有 RDMA 网络的GPU节点，运行以下命令验证主机环境。

1. 验证 ofed 驱动

```
 $ ofed_info -s        #roce驱动版本
MLNX_OFED_LINUX-5.8-1.1.2.1:
```

2. 验证 Nvidia GPU 驱动

```
 $ nvidia-smi  #nvidia gpu驱动
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.141.03   Driver Version: 470.141.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:53:00.0 Off |                    0 |
| N/A   29C    P0    64W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM...  On   | 00000000:59:00.0 Off |                    0 |
| N/A   32C    P0    61W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM...  On   | 00000000:6E:00.0 Off |                    0 |
| N/A   33C    P0    67W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM...  On   | 00000000:73:00.0 Off |                    0 |
| N/A   29C    P0    60W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   4  NVIDIA A100-SXM...  On   | 00000000:8D:00.0 Off |                    0 |
| N/A   29C    P0    60W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   5  NVIDIA A100-SXM...  On   | 00000000:92:00.0 Off |                    0 |
| N/A   32C    P0    65W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   6  NVIDIA A100-SXM...  On   | 00000000:C9:00.0 Off |                    0 |
| N/A   33C    P0    64W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   7  NVIDIA A100-SXM...  On   | 00000000:CF:00.0 Off |                    0 |
| N/A   28C    P0    62W / 400W |      0MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

3. 查询 RDMA 网卡

```
 $ show_gids
DEV PORT INDEX GID     IPv4    VER DEV
--- ---- ----- ---     ------------   --- ---
mlx5_0 1 0 fe80:0000:0000:0000:f820:20ff:fe28:c769   v1 eth0
mlx5_0 1 1 fe80:0000:0000:0000:f820:20ff:fe28:c769   v2 eth0
mlx5_0 1 2 0000:0000:0000:0000:0000:ffff:0a00:3c03 10.0.60.3   v1 eth0
mlx5_0 1 3 0000:0000:0000:0000:0000:ffff:0a00:3c03 10.0.60.3   v2 eth0
mlx5_1 1 0 fe80:0000:0000:0000:eaeb:d3ff:fecc:c920   v1 eth1
mlx5_1 1 1 fe80:0000:0000:0000:eaeb:d3ff:fecc:c920   v2 eth1
mlx5_1 1 2 0000:0000:0000:0000:0000:ffff:190b:8002 25.11.128.2   v1 eth1
mlx5_1 1 3 0000:0000:0000:0000:0000:ffff:190b:8002 25.11.128.2   v2 eth1
mlx5_2 1 0 fe80:0000:0000:0000:eaeb:d3ff:fecc:c921   v1 eth2
mlx5_2 1 1 fe80:0000:0000:0000:eaeb:d3ff:fecc:c921   v2 eth2
mlx5_2 1 2 0000:0000:0000:0000:0000:ffff:190b:8022 25.11.128.34   v1 eth2
mlx5_2 1 3 0000:0000:0000:0000:0000:ffff:190b:8022 25.11.128.34   v2 eth2
mlx5_3 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe6c:51d2   v1 eth3
mlx5_3 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe6c:51d2   v2 eth3
mlx5_3 1 2 0000:0000:0000:0000:0000:ffff:190b:8042 25.11.128.66   v1 eth3
mlx5_3 1 3 0000:0000:0000:0000:0000:ffff:190b:8042 25.11.128.66   v2 eth3
mlx5_4 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe6c:51d3   v1 eth4
mlx5_4 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe6c:51d3   v2 eth4
mlx5_4 1 2 0000:0000:0000:0000:0000:ffff:190b:8062 25.11.128.98   v1 eth4
mlx5_4 1 3 0000:0000:0000:0000:0000:ffff:190b:8062 25.11.128.98   v2 eth4
mlx5_5 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe33:1366   v1 eth5
mlx5_5 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe33:1366   v2 eth5
mlx5_5 1 2 0000:0000:0000:0000:0000:ffff:190b:8082 25.11.128.130   v1 eth5
mlx5_5 1 3 0000:0000:0000:0000:0000:ffff:190b:8082 25.11.128.130   v2 eth5
mlx5_6 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe33:1367   v1 eth6
mlx5_6 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe33:1367   v2 eth6
mlx5_6 1 2 0000:0000:0000:0000:0000:ffff:190b:80a2 25.11.128.162   v1 eth6
mlx5_6 1 3 0000:0000:0000:0000:0000:ffff:190b:80a2 25.11.128.162   v2 eth6
mlx5_7 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe6c:68ae   v1 eth7
mlx5_7 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe6c:68ae   v2 eth7
mlx5_7 1 2 0000:0000:0000:0000:0000:ffff:190b:80c2 25.11.128.194   v1 eth7
mlx5_7 1 3 0000:0000:0000:0000:0000:ffff:190b:80c2 25.11.128.194   v2 eth7
mlx5_8 1 0 fe80:0000:0000:0000:eaeb:d3ff:fe6c:68af   v1 eth8
mlx5_8 1 1 fe80:0000:0000:0000:eaeb:d3ff:fe6c:68af   v2 eth8
mlx5_8 1 2 0000:0000:0000:0000:0000:ffff:190b:80e2 25.11.128.226   v1 eth8
mlx5_8 1 3 0000:0000:0000:0000:0000:ffff:190b:80e2 25.11.128.226   v2 eth8
```

提交任务

NCCL是NVIDIA的集合通信库，能实现Collective通信和点对点通信，NCCL内部已经实现了RDMA通信，同时NCCL可以根据环境中网卡类型和拓扑关系，自行选择一个最优的通信路径，目前主流的分布式训练框架都已支持NCCL。

下面介绍将介绍如何在云原生AI 上通过 YAML或者控制台的方式，提交基于NCCL的RDMA分布式训练任务。

YAML方式提交任务

准备 已通过kubectl连接kubernetes集群。具体操作，请参见通过Kubectl连接集群。

任务示例

以下是一个swin-transformer基于NCCL的Pytorch分布式训练任务的示例：

```
 apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
  name: "pytorch-swin-transformer-nccl"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              image: registry.baidubce.com/cce-ai-native/swin-transformer-torch:v2.0
              imagePullPolicy: IfNotPresent
```

```
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - python -m torch.distributed.launch --nproc_per_node 8 --nnodes $WORLD_SIZE --node_rank $RANK --master_addr $HOSTNAME --
master_port $MASTER_PORT main.py --cfg configs/swin/swin_large_patch4_window7_224_22k.yaml --pretrained
swin_large_patch4_window7_224_22k.pth --data-path /imagenet --batch-size 128 --accumulation-steps 2
            env:
              - name: NCCL_DEBUG
                value: "INFO"
              - name: NCCL_IB_DISABLE
                value: "0"
            securityContext:
              capabilities:
                add: [ "IPC_LOCK" ]
            resources:
              limits:
                baidu.com/a100_80g_cgpu: 8
                rdma/hca: 1
            volumeMounts:
              - mountPath: /imagenet
                name: dataset
              - mountPath: /dev/shm
                name: cache-volume
          schedulerName: volcano
          volumes:
            - name: dataset
              persistentVolumeClaim:
                claimName: imagenet-22k-pvc
            - emptyDir:
                medium: Memory
              name: cache-volume
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              image: registry.baidubce.com/cce-ai-native/swin-transformer-torch:v2.0
              imagePullPolicy: IfNotPresent
              command:
              - /bin/sh
              - -c
              - python -m torch.distributed.launch --nproc_per_node 8 --nnodes $WORLD_SIZE --node_rank $RANK --master_addr $MASTER_ADDR
--master_port $MASTER_PORT main.py --cfg configs/swin/swin_large_patch4_window7_224_22k.yaml --pretrained
swin_large_patch4_window7_224_22k.pth --data-path /imagenet --batch-size 128 --accumulation-steps 2
              env:
                - name: NCCL_DEBUG
                  value: "INFO"
                - name: NCCL_IB_DISABLE
                  value: "0"
              securityContext:
                capabilities:
                  add: [ "IPC_LOCK" ]
              resources:
                limits:
                  baidu.com/a100_80g_cgpu: 8
                  rdma/hca: 1
              volumeMounts:
                - mountPath: /imagenet
                  name: dataset
                - mountPath: /dev/shm
                  name: cache-volume
          volumes:
            - name: dataset
              persistentVolumeClaim:
                claimName: imagenet-22k-pvc
```

```
          - emptyDir:
              medium: Memory
            name: cache-volume
        schedulerName: volcano
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: imagenet-22k-pv
spec:
  accessModes:
  - ReadWriteMany
  storageClassName:
  capacity:
    storage: 100Gi
  csi:
    driver: csi-clusterfileplugin
    volumeHandle: data-id
    volumeAttributes:
      parentDir: /          #必填，自定义路径
      path: ""              #必填，pfs 挂载路径，这里需要填写相对于 parentDir 的路径
      clusterIP: ""         #必填，pfs实例的endpoint
      clusterPort: "8888"   #必填，当前端口固定为8888
      clientID: ""          #非必填，pfs实例的id

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: imagenet-22k-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  storageClassName:
  resources:
    requests:
      storage: 100Gi
```

关键参数说明

1. NCCL需要输入环境变量来使能RDMA特性，具体包括以下环境变量：

> 备注：带 * 号的环境变量，云原生AI会基于百度智能云内部大规模分布式训练经验，在任务运行时会自动注入推荐的值，无需手动填写

键	值	含义	备注
NCCL_IB_DISABLE	0,1	0代表NCCL使用的IB/RoCE传输； 1代表禁止NCCL使用的IB/RoCE传输。此时NCCL将退回到使用IP套接字	--
NCCL_IB_HCA*	mlx5_1~mlx5_8	NCCL_IB_HCA 变量指定使用哪些RDMA接口进行通信。 根据套餐类型填对应的值，例如：8卡的roce套餐填mlx5_1~mlx5_8，4卡为mlx5_1~mlx5_4，以此类推。 注：mlx5_0通常为tcp网络的主网卡	云原生AI在任务运行时，会自动检测容器的roce环境，并为容器PID为1的进程自动注入NCCL_IB_HCA 、NCCL_SOCKET_IFNAME 、 NCCL_IB_GID_INDEX、NCCL_IB_TIMEOUT 和 NCCL_IB_QPS_PER_CONNECTION 环境变量；通过PID1创建的子进程会继承这个环境变量，但是如果是通过sh启动的进程则无法继承。如果用户在yaml中声明了这几个环境变量，容器运行时则不再自动注入
NCCL_SOCKET_IFNAME*	eth0	NCCL_SOCKET_IFNAME变量指定使用哪个IP接口进行通信; 注：容器内默认是eth0	
NCCL_IB_GID_INDEX*	动态值	NCCL_IB_GID_INDEX变量定义了RoCE模式中使用的全局ID索引。设置方法请参见ib show_gids命令。 注：如果是通过容器网络使用RoCE网卡，CCE的CNI插件会为容器创建对应数量的子网卡，roce网卡的GID INDEX会动态生成。如果容器内有show_gids命令，可以看到对应的值。	
NCCL_IB_TIMEOUT*	22	网络断点重连超时时间。	
NCCL_IB_QPS_PER_CONNECTION*	8	每个IB qp连接的连接数。	
NCCL_DEBUG	INFO	NCCL_DEBUG变量控制从NCCL显示的调试信息。此变量通常用于调试。	

2. 为了在rank之间共享数据，NCCL需要为IPC和固定(页面锁定)系统内存资源共享系统内存，需要在YAML中配置如下参数

```
        securityContext:
        capabilities:
          add: [ "IPC_LOCK" ] #使能进程间通信
      resources:
        limits:
          baidu.com/a100_80g_cgpu: 8
          rdma/hca: 1  #使能RDMA资源
      volumeMounts:
        - mountPath: /imagenet
          name: dataset
        - mountPath: /dev/shm  #使用Emptydir，为IPC提供足够的共享内存
          name: cache-volume
  volumes:
    - name: dataset
      persistentVolumeClaim:
        claimName: imagenet-22k-pvc
    - emptyDir:  #使用Emptydir，为IPC提供足够的共享内存
        medium: Memory
      name: cache-volume
```

3. 此示例运行依赖开源数据集ImageNet-22K。这里使用百度云并行文件存储PFS作为共享存储，需要将上述开源数据集提前拷贝PFS挂载目录中，如何在CCE容器集群中使用并行文件存储PFS详见：这里。

控制台方式提交任务

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 任务管理**。
6. 在任务管理页面单击 **新建任务**，进入创建任务的流程
7. 在 **新建任务 > 容器配置** ，**任务类型**模块，训练方式选择 **分布式** 后，选择 **开启RDMA** 即可。

> 备注：云原生AI控制台会默认向任务中注入上述 YAML示例中的关键参数，无需手动填写

数据集管理

新建数据集

您可以新建一个数据集，数据集是逻辑上相关的一组数据的集合，可以被容器引擎使用。

创建数据集会同时创建Dataset与Runtime两种资源，Dataset为数据源配置，Runtime为后端的加速引擎，两者都成功创建后数据集进入到加速状态并创建一个加速PVC可被Pod挂载使用。

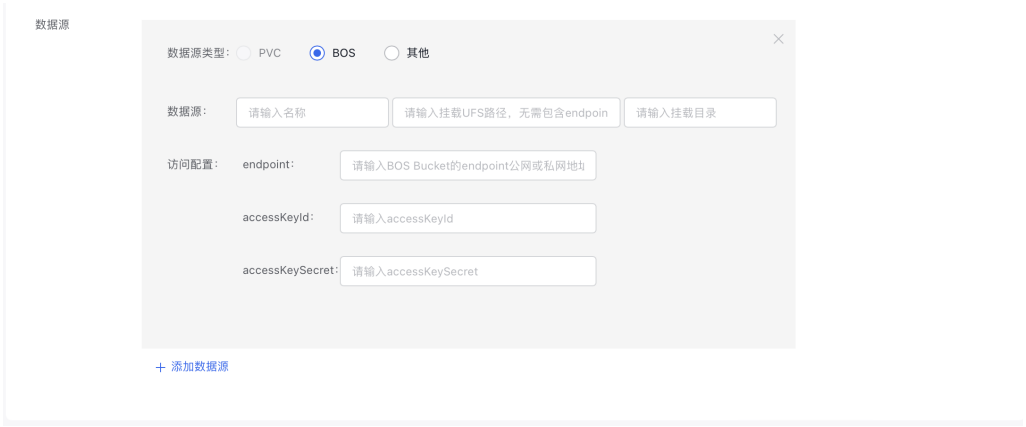
前提条件

- 您已成功创建一个Kubernetes集群,详情请见创建集群。
- 您已成功安装CCE Fluid组件，否则此功能将无法使用。
- 若在Runtime配置部分选择PFS类型，请先安装“CCE CSI PFS Plugin"或“CCE ROCE CSI PFS Plugin"组件。
- 组件支持集群版本：1.26.9,1.24.4,1.22.5,1.20.8,1.18.9,1.16.3。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 数据集**。
6. 在数据集管理单击**新建数据集**。
7. 在新建数据集页面中，完成数据集基本信息及数据源配置：

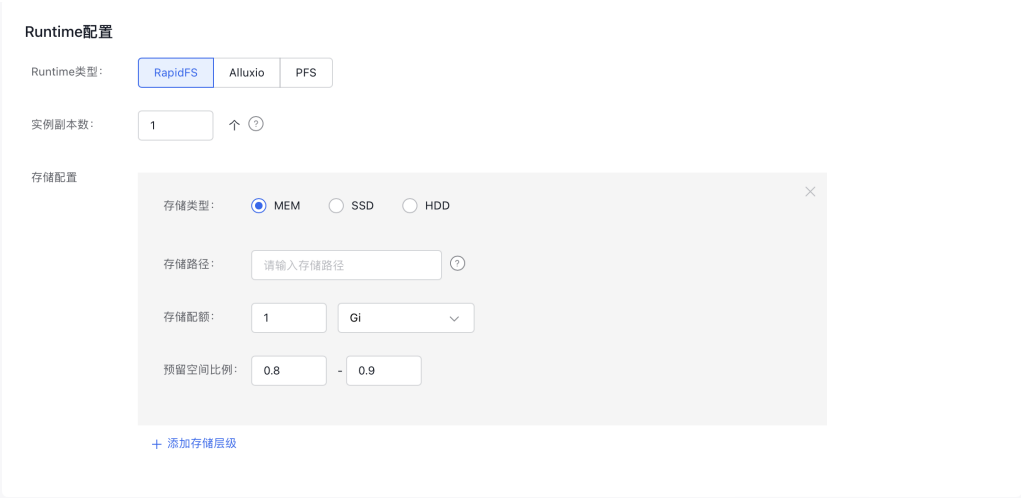
- 数据集名称：自定义队名名称，长度为1-65个字符，只能包含数字、英文小写字母和“-”，且不能以“-”开头或结尾。
- 命名空间：选择集群中已有的命名空间。
- 数据源：选择数据源类型、数据源位置，同时添加相关访问配置。其中数据源类型可选择“PVC”、“BOS”或“其他”。

> 注意：此处默认的Runtime配置类型为RapidFS，若需要Alluxio类型Runtime，请在下方Runtime类型中选择，同时在此处指定如下信息：
>
> 
>

8. 完成调度信息配置（非必填）：

- 节点亲和性：为数据集指定podAffinity。
- 容忍设置：为数据集指定podAntiAffinity。

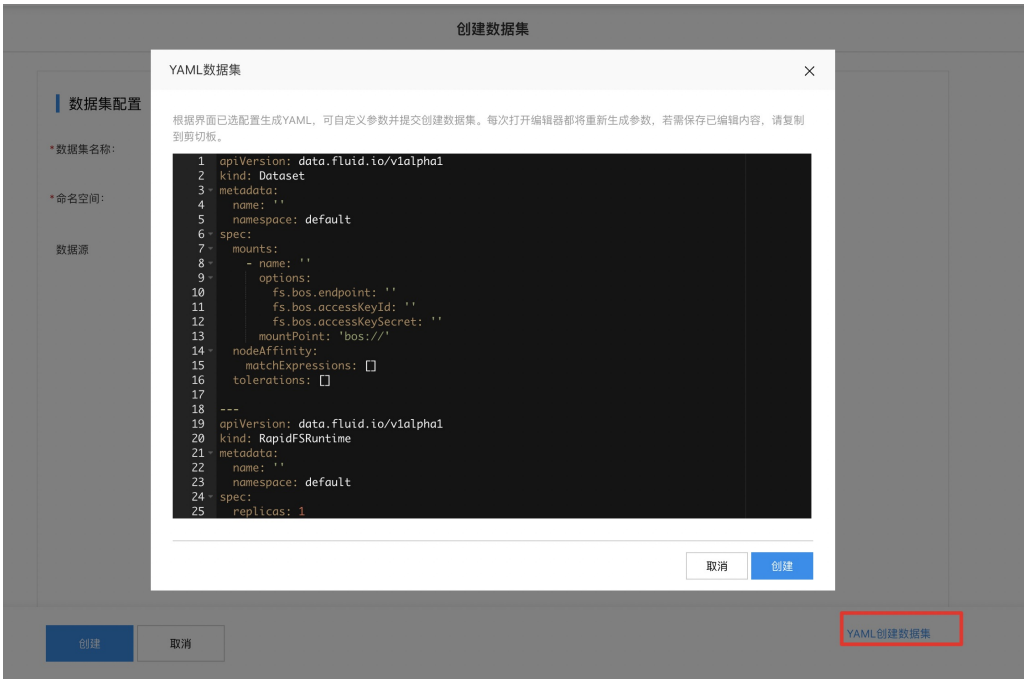
9. 完成Runtime配置。



- Runtime类型：为Runtime指定类型，目前支持“RapidFS”、“Alluxio”和”PFS“三种类型。若选择PFS类型，请先安装“CCE CSI PFS Plugin"或“CCE ROCE CSI PFS Plugin"组件。
- 实例副本数：为Runtime指定replicas。
- 存储配置：为Runtime指定tieredstore，同时可添加多层级。

10. 点击“创建”按钮，完成数据集的新建。
11. 若以上配置不足以满足您对资源的配置要求，您也可以选择使用YAML进行数据集创建。

> 注意：
>
> 若您已经在前端页面已添加了部分配置，将根据界面已选配置生成YAML，您可自定义参数并提交创建数据集。
>
> 每次打开编辑器都将重新生成参数，若需保存已编辑内容，请复制到剪切板。



操作数据集

您可以根据需要对数据集进行操作。

前提条件

- 您已成功安装CCE Fluid组件，否则此功能将无法使用。
- 您已成功创建数据集。

解除数据集加速

1. 登录百度智能云官网，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 数据集**。
6. 在数据集列表中确定需要操作的数据集，点击操作列的**“解除加速”**，确认信息后，点击“确认”。

> 注意：
>
> - 解除加速将删除对应的Runtime实例，并清除已缓存的数据，且不能恢复。
> - 后续可通过开启加速重新创建Runtime实例，以及通过缓存预热进行数据缓存。

缓存预热

1. 在CCE集群管理页面单击**云原生AI > 数据集**。
2. 在数据集列表中确定需要操作的数据集，点击操作列的**“缓存预热”**，添加需要预热的数据路径，点击“确认”。

> 注意：
>
> - 默认情况下，系统将预先缓存数据集的全部数据到本地。若指定数据集的一个或多个路径，则只缓存对应路径下的数据； 若指定为空则默认缓存所有路径。
> - RapidFS 仅支持对根路径即整个数据集进行预热，暂不支持子路径预热。

查看YAML

1. 在CCE集群管理页面单击**云原生AI > 数据集**。
2. 在数据集列表中确定需要操作的数据集，点击操作列的**“查看YAML”**。

查看数据集

您可以根据需要管理/查看数据集。

前提条件

- 您已成功安装CCE Fluid组件，否则此功能将无法使用。
- 您已成功创建数据集。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 数据集**。
6. 在数据集管理页面中，可以查看已创建数据集及其相关信息，如果数据集较多，可以通过左上方“数据集名称”进行快速搜索。

7.在数据集列表中“数据集名称”列点击某一数据集名称，进入“数据集详情”页面，查看详细信息。

删除数据集

您可以通过容器引擎管理控制台删除不再需要的数据集。

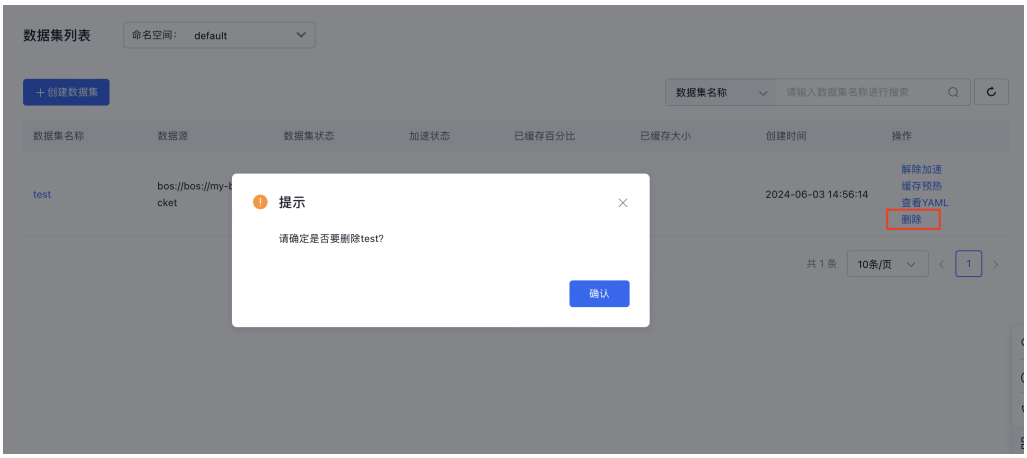
前提条件

- 您已成功安装CCE Fluid组件，否则此功能将无法使用。
- 您已成功创建数据集。

操作步骤

1. 登录百度智能云官网，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 数据集**。
6. 在数据集管理页面中，点击目标数据集操作列的**“删除”**。
7. 在确认删除弹出框中，单击“确定”按钮完成数据集删除。

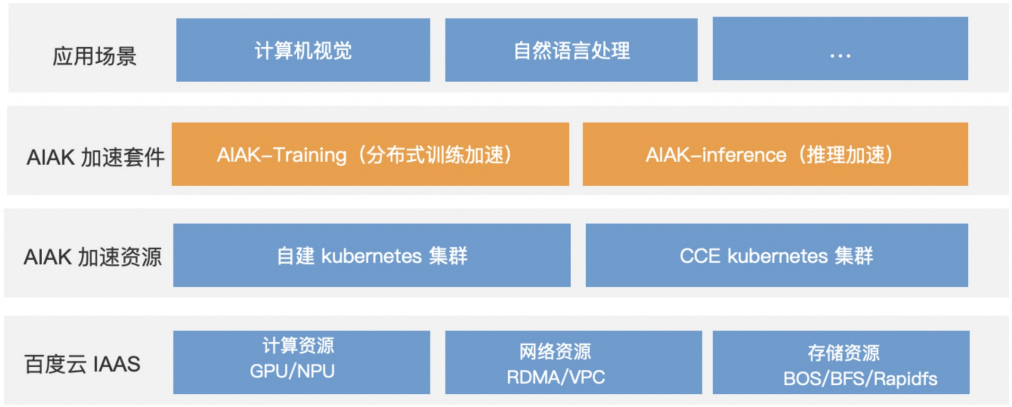


AI 加速套件

AIAK 简介

什么是AIAK？

AI加速套件AI Accelerate Kit（后文使用简称 AIAK）是基于百度云IAAS资源推出的AI加速能力，用来加速Pytorch、TensorFlow等深度学习框架的AI应用，能极大提升分布式训练和推理的性能。下图为AIAK的整体解决方案架构图。



应用场景

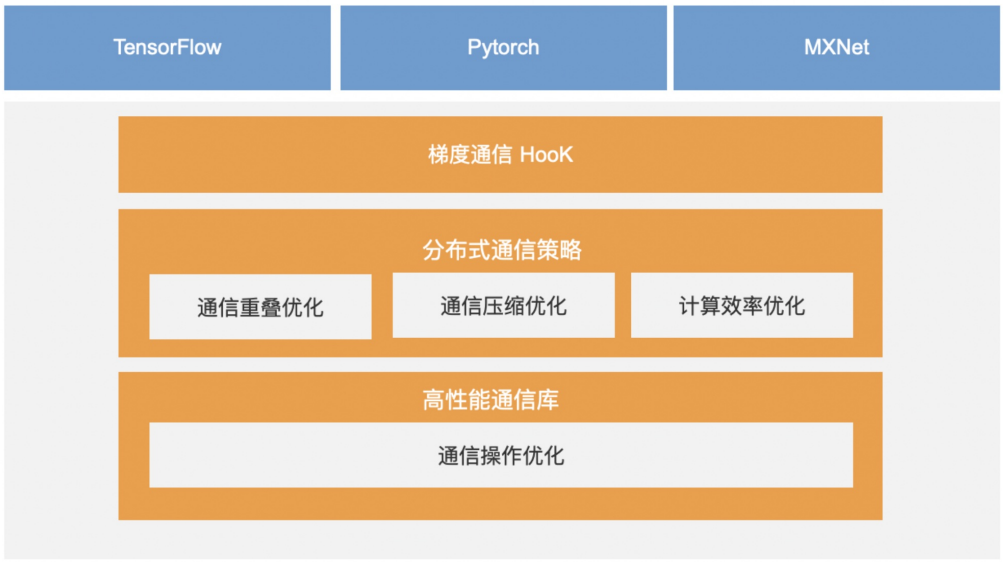
AIAK加速套件支持但不限于以下场景：

- 自然语言处理，例如BERT，Transformer等。
- 图像识别，例如ResNet等。

优势

- 多框架支持：提供对TensorFlow和PyTorch多种人工智能框架的加速优化。
- 轻量便捷：基于开源框架编写的算法代码或模型代码，只需少量修改。

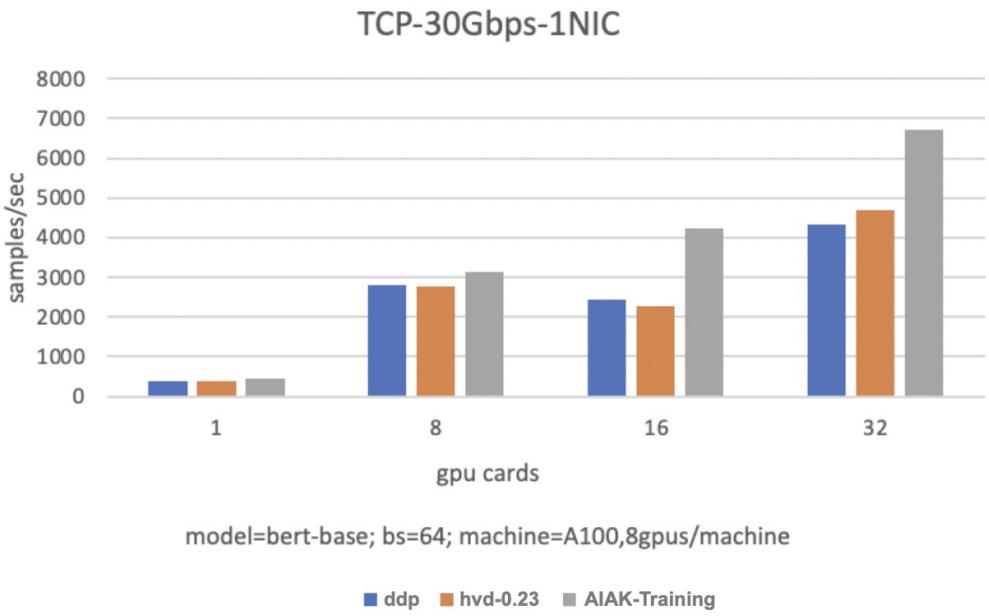
AIAK-Training AIAK-Training是基于Horovod深度定制优化的分布式训练框架，在保留Horovod已有功能特性的基础上，增加了新的通信优化特性，能够进一步提升分布式训练吞吐。AIAK-Training完全兼容Horovod 原有API，用户如果已经使用原生Horovod进行训练，已有的功能代码无需进行任何改变。下图为AIAK-Training架构图。



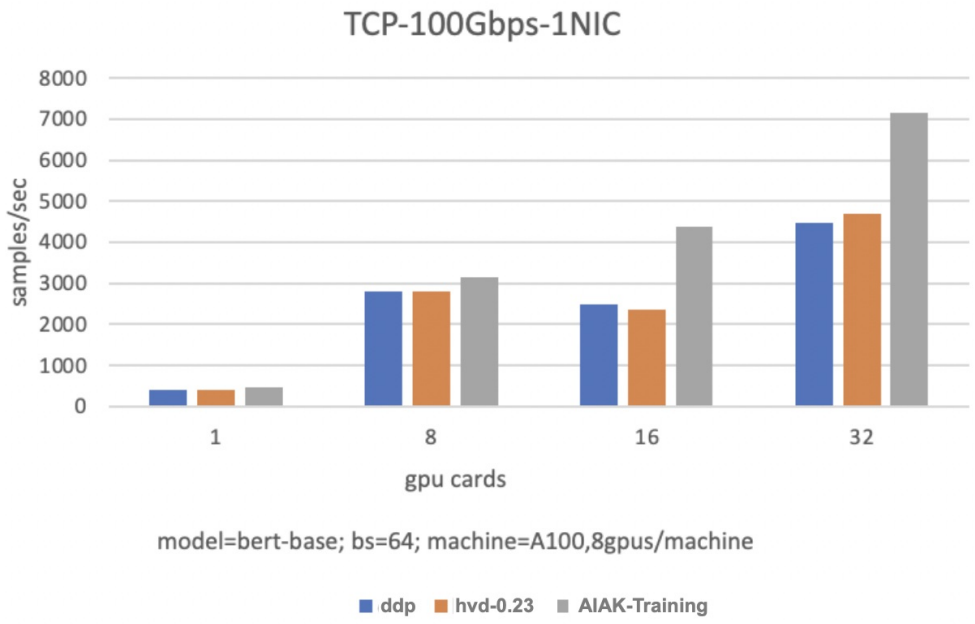
相比原生Horovod，AIAK-Training 新增了如下加速特性：

- 新增优化版分层Allreduce实现，相比原生Horovod分层Allreduce 实现，性能提升10%。
- 支持Deep Gradient Compression (DGC) 通信压缩技术，降低通信量，提升传输效率。暂仅支持PyTorch。
- 支持参数更新计算和梯度通信重叠，无需等待全局通信结束后统一进行参数更新，进一步隐藏通信时间。暂仅支持PyTorch。
- 支持fused optimizer，融合参数更新计算Kernel，减少访存、kernel launch等消耗。暂仅支持PyTorch。

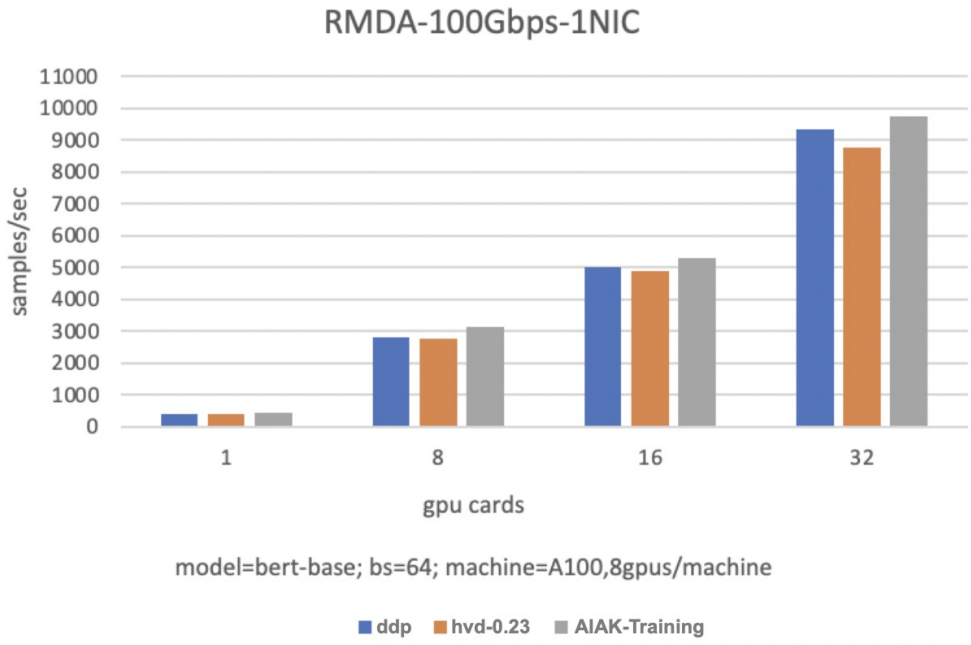
性能表现 在TCP-30Gbps-1NIC 场景下，2台单机8卡提升85.5%，4台单机8卡提升43.7%。



在TCP-100Gbps-1NIC 场景下，2台单机卡提升85.5%，4台单机8卡提升52.1%。

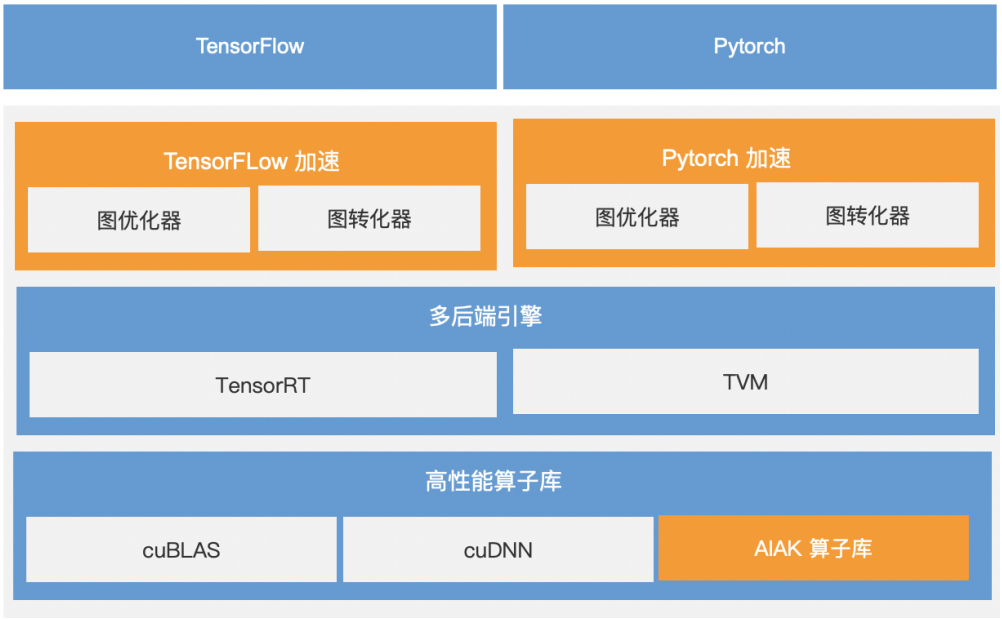


在RDMA-100Gbps-1NIC场景下，2台单机8卡提升8.5%，4台单机8卡提升11.4%。



AIAK-Inference

AIAK-Inference是通过优化AI框架（Tensorflow、PyTorch）产出的模型，降低在线推理延迟、提升服务吞吐，大幅增加异构资源使用效率。下图为AIAK-Inference 架构图。

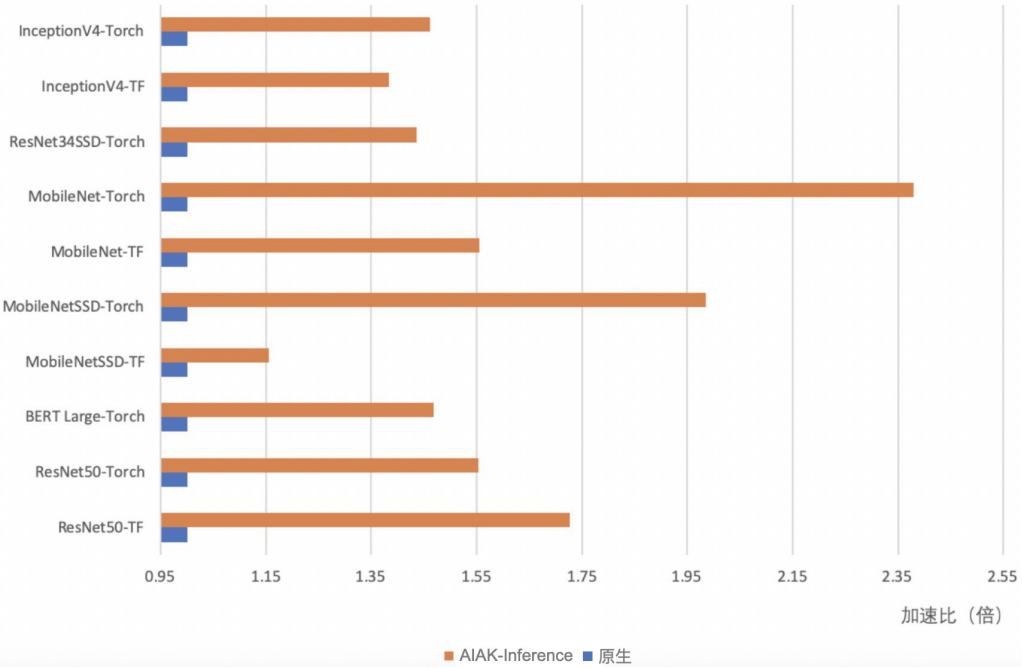


推理加速总体分为5大模块，分别的功能简介如下：

- 框架接入：对接Tensorflow/PyTorch等主流深度学习框架，直接结合原生框架能力对接，不需要框架间图转换，减少额外性能损耗。
- 图优化器：整体理解模型信息，结合硬件芯片特点，自动进行模型切分，使各个子图易于融合，减少冗余访存。
- 图转换器：将优化后的子图转换为多后端引擎支持的格式，进行后端实际推理。
- 多后端引擎：针对不同模型结构，在多类推理引擎间进行任务切分，同时支持常见模型与长尾模型优化，兼顾性能与灵活性。
- 高性能算子库：结合硬件厂商提供的基础算子库，针对常见模型中的典型结构（例如Transformer）和计算密集型算子（GEMM、Conv）进行深度定制，使模型推理过程中发挥硬件极致性能。

性能表现

通过图优化和算子融合提升加速效率，ResNet 等经典模型可以加速15%~135%。



使用AIAK-Inference 加速推理业务

前提条件

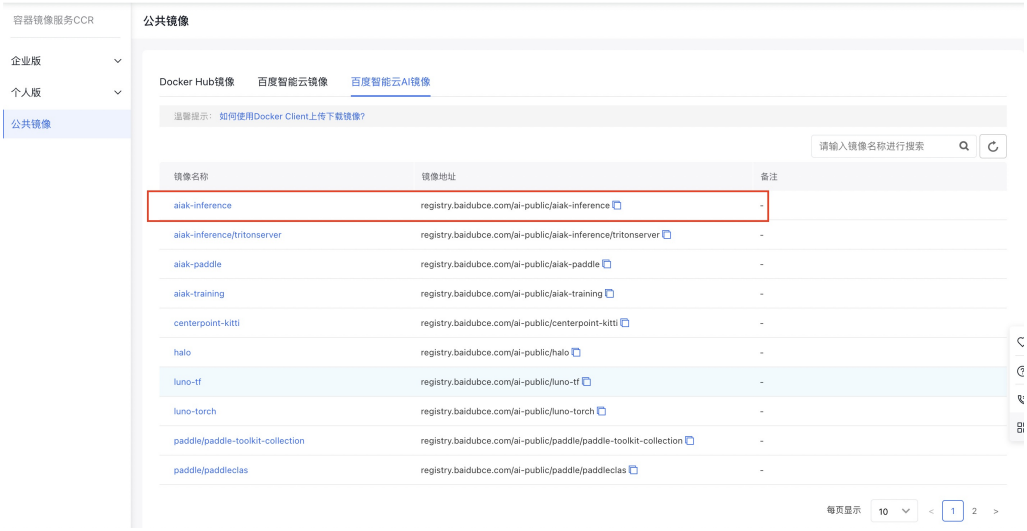
- 选择CCR中的AIAK-Inference推理加速镜像作为基础镜像。

操作流程

Tensorflow模型优化

在CCR公共镜像的“百度智能云AI镜像”中选择aiak-inference:ubuntu18.04-cu11.2-tf2.4.1-py3.6-aiak1.1-latest这个tag（或使用docker pull registry.baidubce.com/ai-public/aiak-inference:ubuntu18.04-cu11.2-tf2.4.1-py3.6-aiak1.1-latest拉取镜像），镜像内部已经安装好了CUDA、

Tensorflow 2.4.1等基础库。



在容器内准备ResNet50模型：

```
 import os
import numpy as np
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

def _wget_demo_tgz():
    # 下载一个公开的resnet50模型。
    url = 'http://url/to/your/model/YOUR_MODEL.tar.gz'
    local_tgz = os.path.basename(url)
    local_dir = local_tgz.split('.')[0]
    if not os.path.exists(local_dir):
        luno.util.wget(url, local_tgz)
        luno.util.unpack(local_tgz)
    model_path = os.path.abspath(os.path.join(local_dir, "frozen_inference_graph.pb"))
    graph_def = tf.GraphDef()
    with open(model_path, 'rb') as f:
        graph_def.ParseFromString(f.read())
    # 以随机数作为测试数据。
    test_data = np.random.rand(1, 800, 1000, 3)
    return graph_def, {'image_tensor:0': test_data}

graph_def, test_data = _wget_demo_tgz()

input_nodes=['image_tensor']
output_nodes = ['detection_boxes', 'detection_scores', 'detection_classes', 'num_detections', 'detection_masks']
```

然后尝试推理这个模型：

```
 import time

def benchmark(model):
    tf.reset_default_graph()
    with tf.Session() as sess:
        sess.graph.as_default()
        tf.import_graph_def(model, name="")
        # Warmup!
        for i in range(0, 1000):
            sess.run(['image_tensor:0'], test_data)
        # Benchmark!
        num_runs = 1000
        start = time.time()
        for i in range(0, num_runs):
            sess.run(['image_tensor:0'], test_data)
        elapsed = time.time() - start
        rt_ms = elapsed / num_runs * 1000.0
        # Show the result!
        print("Latency of model: {:.2f} ms.".format(rt_ms))

###### original graph
print("before compile:")
benchmark(graph_def)
```

接下来引入AIAK-Inference优化：

```
 import aiak_inference
optimized_model = aiak_inference.optimize(
graph_def,
'gpu',
outputs=['detection_boxes', 'detection_scores', 'detection_classes', 'num_detections', 'detection_masks']
)
```

优化后的模型仍然是一个GraphDef模型，可以使用同样的代码进行推理：

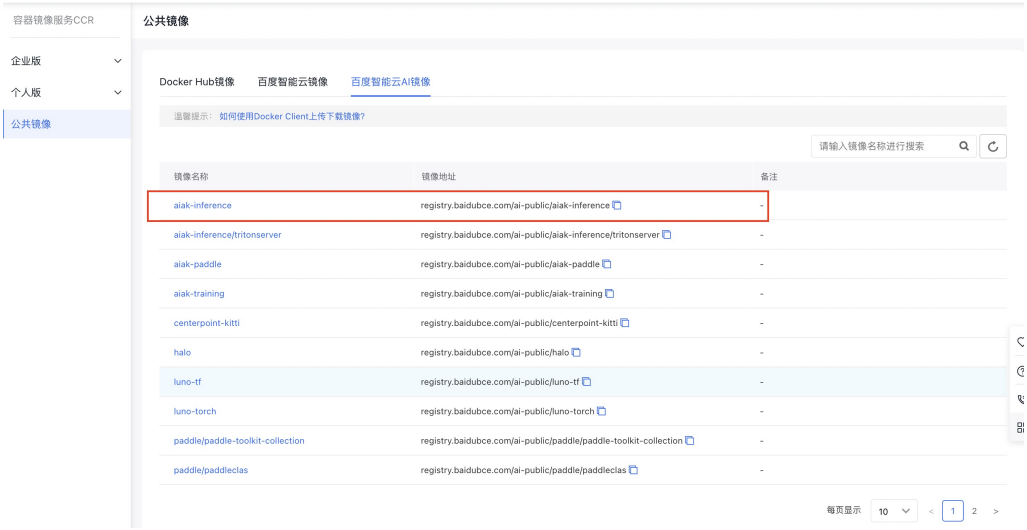
```
 ###### optimized graph
print("after compile:")
benchmark(optimized_model)
```

经过比较，可以看到性能有提升：

```
2022-04-27 11:03:19 =====Original Performance=====
2022-04-27 11:03:19 Latency of model: 7.14 ms.
2022-04-27 11:03:19 [array([112])]
2022-04-27 11:03:19 =====Optimized Performance=====
2022-04-27 11:03:19 Latency of model: 4.23 ms.
```

PyTorch模型优化

在CCR公共镜像的“AI加速镜像”中选择aiak-inference:cuda11.2_cudnn8_trt8.4_torch1.11-aiak_1.1.1_latest加速镜像（或使用docker pull registry.baidubce.com/ai-public/aiak-inference:cuda11.2_cudnn8_trt8.4_torch1.11-aiak_1.1.1_latest拉取镜像），镜像内部已经安装好了CUDA、PyTorch 1.11等基础库。



在容器中准备PyTorch相关模型，以ResNet50为例：

```
import os
import time
import torch
import torchvision.models as models

model = models.resnet50().float().cuda()
model = torch.jit.script(model).eval()  # 使用jit转为静态图
dummy = torch.rand(1, 3, 224, 224).cuda()
```

尝试进行推理：

```
@torch.no_grad()
def benchmark(model, inp):
    for i in range(100):
        model(inp)
    start = time.time()
    for i in range(200):
        model(inp)
    elapsed_ms = (time.time() - start) * 1000
    print("Latency: {:.2f}".format(elapsed_ms / 200))

###### benchmark before optimization
print("before optimization:")
benchmark(model, dummy)
```

接着使用AIAK-Inference进行模型优化，并得到优化后的模型：

```
import aiak_inference

optimized_model = aiak_inference.optimize(
model,
'gpu',
test_data=[dummy],
)
```

再次进行推理：

```
###### benchmark after optimization
print("after optimization:")
benchmark(optimized_model, dummy)
```

比较二者性能，可以看到单次推理延迟有大幅下降，证明AIAK-Inference加速能力：

```
2022-04-27 15:08:04 before optimization:
2022-04-27 15:08:04 Latency: 5.07
2022-04-27 15:08:04 after optimization:
2022-04-27 15:08:04 Latency: 3.26
```

使用 AIAK-Training 部署分布式训练任务

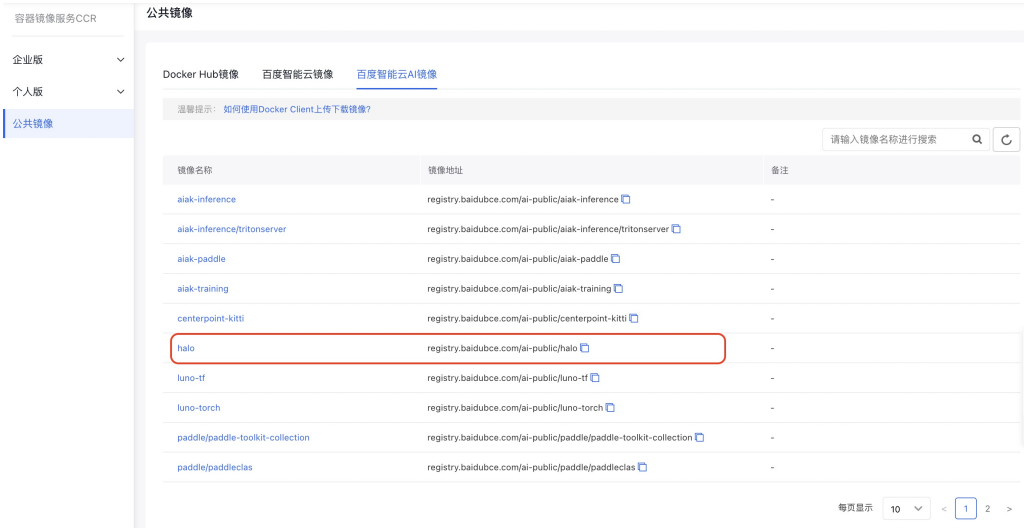
前提条件

- 已经开通CCE云原生AI服务。
- CUDA 11.2，Pytorch 1.8.1，TensorFlow 2.5.0，MxNet 1.8.0，如果AI应用是其他的版本，请提交工单。

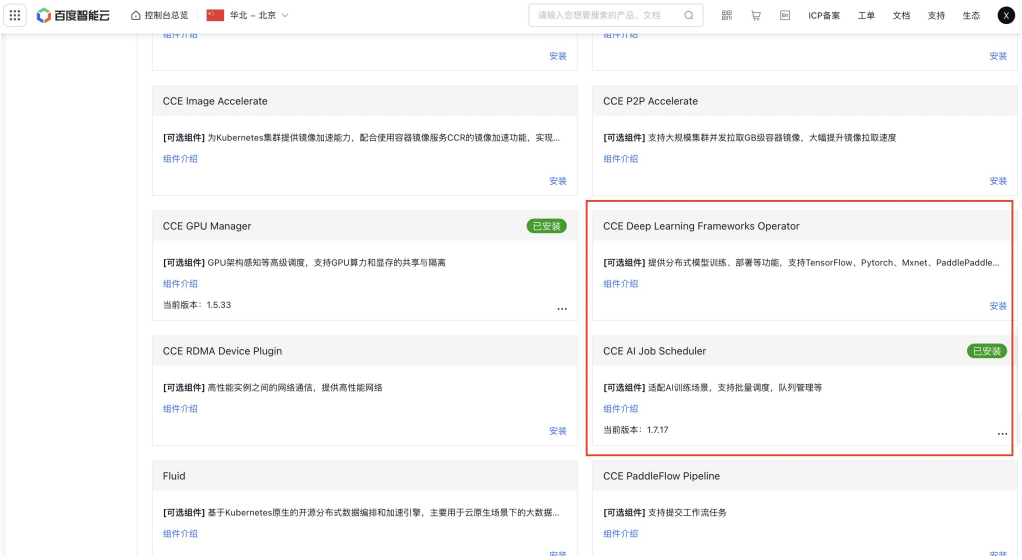
操作流程

1.选择训练镜像

在CCR公共镜像的“百度智能云AI镜像” 中选择“Halo”加速镜像，作为训练基础镜像，该镜像内已安装了 CUDA、Python 3、PyTorch、Tensorflow、MxNet和AIAK-Training加速软件等。

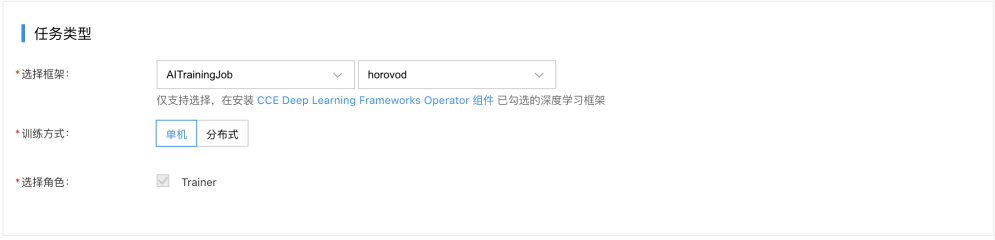


2.组件安装 点击集群名称进入，点击组件管理->云原生AI，点击安装CCE Deep Learning Frameworks Operator组件和CCE AI Job Scheduler，点击确定。



3.任务提交 云原生AI->任务管理->新建任务

- 选择框架AITrainingJob，horovod。
- 选择训练方式：分布式。
- 选择角色：选择“Launcher”，同时需要指定pod的弹性范围。



- 容器组配置->生命周期->启动命令中配置如下需要的内容。

启用优化版分层Allreduce

```
horovodrun  --nccl-hierarchical-allreduce [other-horovod-args] python [executable] [app-args]
```

horovodrun命令行通过传递 --nccl-hierarchical-allreduce 参数使能。

启用DGC稀疏化通信压缩算法

```
horovodrun --dgc \
        --compress-ratio 0.001 \
        --sample-ratio 0.01 \
        --clip-grad-func norm/value\
        --clip-grad-func-params 0.01 \
        ... ...
        [other-horovod-args] python [executable] [app-args]
```

其中，可配置DGC相关的参数有：

参数	说明	必填项
compress_ratio	压缩率，float，默认0.001，值越小，梯度的通信量越少；（推荐手动配置）	是
sample_ratio	样本采集比率，float，默认0.01	否
strided_sample：bool	默认True，是否按照步幅采集样本	否
compress_upper_bound	压缩上限，默认1.3	否
compress_lower_bound	压缩下限，默认0.8	否
max_adaptation_iters	最大自适应次数	否
resample	bool，默认True，是否重新采样	否
fp16_values	bool，默认False，梯度中的元素值是否转成fp16	否
int32_indices	bool，默认False，梯度中元素索引是否转成int32	否
warmup_epochs	预训练epoch数	否
warmup_coeff	模型预训练阶段的压缩率列表，若有多个值则命令行中以空格间隔，比如warmup_epochs=3，warmup_coeff=[0.5，0.6，0.7]，表示预热阶段的第一个step选择50%的梯度进行通信，第二个step选择60%的梯度进行通信，第三个step选择70%的梯度进行通信	否
clip_grad_func	梯度裁剪，可选norm或value，分别对应Pytorch中的 clip_grad_norm_和 clip_grad_value_	否
clip_grad_func_params	梯度裁剪方法的参数	否
momentum_masking	bool，默认True，动量隐藏，避免梯度过时更新问题	否

> 注：DGC压缩算法目前仅支持SGD优化器。

启用参数更新和梯度通信重叠

用户使能参数更新和梯度通信重叠功能，只需在horovodrun命令行里进行如下配置即可。

```
horovodrun --overlap-backward-and-step \
        --clip-grad-func norm/value\
        --clip-grad-func-params 0.01 \
        ... ...
        [other-horovod-args] python [executable] [app-args]
```

- 若用户在参数更新之前，需要进行梯度裁剪操作，可以通过设置clip_grad_func和 clip_grad_func_params两个参数来指定，此时用户在代码中的梯度裁剪调用需要移除。
- 若用户同时开启了DGC和参数更新和梯度通信重叠功能，并且指定了梯度裁剪方法，则梯度裁剪操作会在DGC中进行， 重叠过程中不进行梯度裁剪。
- 如果用户不开启参数更新和梯度通信重叠，无需指定梯度裁剪相关参数，指定也不会生效。

启用FusedOptimizer

用户使能优化器融合功能，只需在horovodrun命令行里进行如下配置即可。

```
horovodrun --fuse-optimizer
        ... ...
        [other-horovod-args] python [executable] [app-args]
```

> 注：目前fused optimizer与参数更新和梯度通信重叠两个功能暂不兼容。

使用AIAK-Training Pytorch版

前提条件

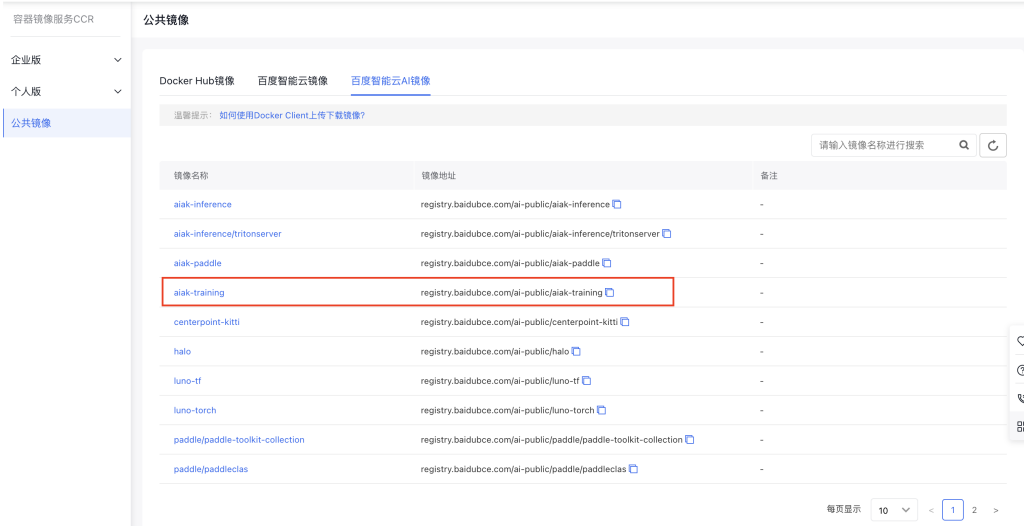
- 已经开通CCE云原生AI服务。 **支持的版本列表** AIAK-Training Pytorch版目前支持如下版本列表，用户如有其他版本需求，可提交工单获得支持。

CUDA版本	Pytorch版本	Python版本
11.7	1.12.0	3.8

加速特性

- 针对低带宽网络环境的通信优化，新增分层Allreduce算法，支持PowerSGD、FP16等通信压缩算法快速使能。
- 引入nvidia apex amp o2混合精度模式，并提供兼容torch amp原生用法的使用方式，方便更多计算转FP16来加速训练。
- 支持fused optimizer，融合计算Kernel，减少访存、kernel launch等开销，提升参数更新阶段效率。
- 支持lamb优化器算法，可用于解决超大batch训练中的收敛问题。 **实践流程**

1.获取训练镜像 在CCR公共镜像的“百度智能云AI镜像”中选择“aiak-training”加速镜像，作为训练基础镜像，该镜像内已安装了CUDA、Python、PyTorch 和AIAK-Training加速软件等。



2.任务提交 提交任务，可详细按照新建Pytorch任务文档进行。过程中，需要将镜像地址设置为上述aiak-training的镜像地址。 **3.应用加速能力 3.1 加速低配网络条件下的分布式训练** 在网络带宽较低的环境下，跨机梯度同步会成为分布式训练速度的主要瓶颈。为了优化该类问题，AIAK-Training新增了分层AllReduce算法实现，来减少跨机通信的影响。同时，我们简化了Pytorch官方提供的PowerSGD、FP16等通信压缩hook的使用方

式，用户无需修改代码，仅通过环境变量即可直接开启。注意：在单机多卡或RDMA环境等通信并非瓶颈情况下，上述功能可能不会带来加速收益。

功能	环境变量	默认	取值说明	建议说明
分层AllReduce配置	AIAK_HIERARCHICAL_ALLREDUCE	0	• 0 表示关闭该功能 • 1 表示开启该功能	• TCP网络环境下建议开启
通信压缩算法配置	AIAK_FP16_COMPRESSION	0	• 0 表示关闭该功能 • 1 表示采用FP16格式压缩梯度 • 2 表示采用BF16格式压缩梯度	• 优先设置为1，即采用FP16格式压缩；BF16目前实验性功能，需要NCCL版本在2.9.6以上 • 该功能开启后，梯度通信之前，会先将梯度转换成float16/bfloat16，通信结束之后转换回float32
	AIAK_POWERSGD_COMPRESSION	0	• 0 表示关闭该功能 • 1 开启powersgd压缩 • 2 开启batched powersgd压缩	• 优先设置为1，即采用powersgd压缩方式，batched powersgd性能更好，但是精度损失会多；如果在AIAK_MATRIX_APPROXIMATION_RANK=1的情况下，模型精度达到要求，可考虑使用batched powerSGD • 该功能可以单独使用，也可以和FP16压缩组合开启，此时压缩算法会叠加进行
	AIAK_MATRIX_APPROXIMATION_RANK	1	• 可以从1开始，按2的倍数来设置，比如1、2、4，...	• 针对powersgd算法的超参数，该值决定了压缩率，值越小，压缩越强 • 通常情况，设置为1-4即可，如果精度影响较多，可以尝试调大
	AIAK_START_COMPRESS_ITER	1000	• 表示从第几个step开始执行压缩算法，默认为1000	• 如果训练中有warmup阶段，该值通常不应少于warmup的step数量 • 过早使用梯度压缩可能会影响最终收敛效果，如要调整该值，建议不小于总训练step的10%

3.2 AMP O2混合精度模式，进一步加速计算 Pytorch官方提供了AMP自动混合精度功能，用户可以使用该技术加速模型的计算效率。AIAK-Training在该功能基础之上，引入了NVIDIA APEX AMP中的O2模式，并在接口层面与Torch AMP做了兼容，用户仅需增加一行代码，即可快速使能更为激进的纯FP16训练方案。具体用法如下：

1. 用户需要在模型和optimizer准备就绪后，增加一行aiak_amp_decorate函数调用的代码，另外当使用DDP时，aiak_amp_decorate需在DDP执行之前调用。（具体见下述示例第6行）
2. 当使用梯度裁剪时，clip_grad_norm(model.parameters(), max_norm)中 model.parameters() 需要替换为 torch.cuda.amp.aiak_amp_parameters(optimizer)。（具体见下述示例第 25行）

```
 ###### 定义 model 和优化器
model = Model().cuda()
optimizer = optim.SGD(model.parameters(), ...)

###### 新增 aiak_amp_decorate 封装调用，用于使能 Apex AMP O2模式
model, optimizer = torch.cuda.amp.aiak_amp_decorate(model, optimizer)

###### 在O2优化初始化之后构造DDP model
model = DDP(model)

###### torch 原生 amp 用法的相关代码，无需修改
scaler = GradScaler()

for input, target in data:
    optimizer.zero_grad()
    # 在autocast enable 区域运行forward
    with autocast():
        output = model(input)
        loss = loss_fn(output, target)
    # 用scaler，scale loss(FP16)，backward得到scaled的梯度(FP16)
    scaler.scale(loss).backward()
    # 当使用梯度裁剪时，遵守torch原生AMP调用规则， 调用unscale进行梯度恢复
    scaler.unscale_(optimizer)
    # 梯度裁剪时，需要替换model.parameters()为amp.amp_parameters(optimizer)
    torch.nn.utils.clip_grad_norm_(torch.cuda.amp.aiak_amp_parameters(optimizer), max_norm)
    # scaler 更新参数，会先自动unscale梯度, 如果有nan或inf，自动跳过
    scaler.step(optimizer)
    # scaler factor更新
    scaler.update()
```

aiak_amp_decorate 接口参数详细如下：

参数项	是否必填	默认值	说明
model	是	无	模型
optimizer	是	无	优化器
keep_batchnorm_fp32	否	TRUE	一般不需要调整该参数。该参数具体值是否保持batchNorm计算精度为FP32，通常设置为 True
loss_scale	否	None	• 一般不需要调整该参数。 • 如果设置为一个浮点数，则使用这个该静态值进行梯度缩放；如果设置为字符串"dynamic"，则会随时间推移调整损失比例，自动动态缩放；设置为None，会启动和"dynamic"相同的效果。
num_losses	否	1	• 一般不需要调整该参数。 • 表示有多少个loss/backward的pass使用
min_loss_scale	否	None	• 一般不需要调整该参数。 • 表示动态缩放时loss_scale的下限
max_loss_scale	否	2.**24	• 一般不需要调整该参数。 • 表示动态缩放时loss_scale的上限

其他注意点：在使用O2模式时，torch原生GradScaler类的backoff_factor、growth_factor、growth_interval等参数，会按照o2模式自动更新，用户无需手动调整，如手动调整也不会实际生效。

3.3 FusedOptimizer 加速参数更新效率 用户使能优化器融合功能，只需设置如下环境变量：

功能	环境变量	默认	取值说明	建议说明
优化器算子融合	AIAK_FUSED_OPTIMIZER	0	• 0 表示关闭该功能 • 1 表示开启该功能	加速优化器 step 阶段效率，建议开启

3.4 大 BatchSize 训练，使能 LAMB优化器 LAMB 优化器，和其他优化器使用方式类似，具体举例如下：

```
 import torch.optim as optim
...
optimizer = optim.LAMB(model.parameters(), lr=args.lr, weight_decay=args.weight_decay, eps=args.eps, betas=args.betas)
```

Lamb 优化器参数详细如下：

参数项	是否必填	默认值	说明
params	是	无	模型参数
lr	否	1e-3	初始学习率，默认1e-3，用户按需进行配置
weight_decay	否	0	权重衰减系数，用户按需进行配置
betas	否	(0.9, 0.999)	• 一般不需要调整该参数。 • betas = （beta1，beta2），用于计算梯度以及梯度平方的运行平均值的系数。 • beta1：一阶矩阵的指数衰减率（如 0.9）。 • beta2：二阶矩阵的指数衰减率（如 0.999）。
eps	否	1e-8	• 一般不需要调整该参数。 • 模糊因子，防止出现除0的情况。
adam_w_mode	否	TRUE	• 一般不需要调整该参数。 • 表示是否使用L2正则化，默认开启。
grad_averaging	否	TRUE	• 一般不需要调整该参数。 • 表示计算平均梯度时，是否使用系数(1-beta2)。
set_grad_none	否	TRUE	• 一般不需要调整该参数。 • 执行zero_grad时，是否将梯度设置为None。
max_grad_norm	否	1	• 一般不需要调整该参数。 • 表示梯度裁剪系数。

AI监控大盘

NVIDIA芯片资源观测

GPUManager组件

GPUManager组件，包含GPUManager组件异常pod 数量、GPUManager组件重启次数，以及GPUManager相关组件的cpu利用率和men使用量。

前提条件

- 已安装CCE GPU Manager组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

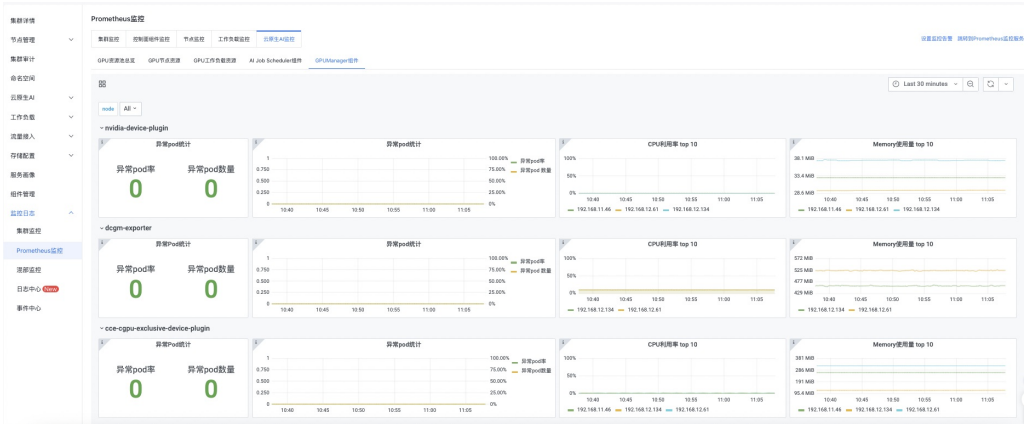
使用方法

1. 登录容器引擎CCE控制台。

2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**GPUManager组件**。

GPUManager组件如图所示



您可以点击右上角**按钮**，自行设定**监控时间**、**手动刷新**、**自动刷新**。

GPUManager组件具体说明

监控项	说明
GPUManager组件异常pod数量	当前集群GPUManager相关组件有异常的Pod数量。 指标异常：异常Pod数量大于0则为异常，需要查看异常原因。
GPUManager组件异常pod率	当前集群GPUManager相关组件有异Pod率。
CPU利用率top10	当前集群GPUManager相关各组件中cpu利用率前十的Pod。
Memory利用率top10	当前集群GPUManager相关各组件中Memory利用率前十的Pod。

AI Job Scheduler组件

AI Job Scheduler 组件，包含关键指标总览、队列排队总览、集群/队列超限总览、Pod 调度性能总览、任务调度性能总览、调度器调度阶段延迟总览。

前提条件

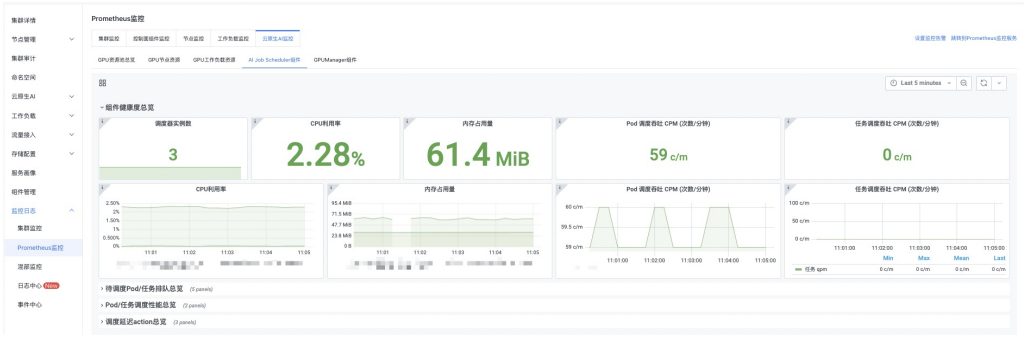
- 已安装 CCE AI Job Scheduler 组件且版本 >= 1.7.9
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**AI Job Scheduler组件**。

AI Job Scheduler组件如图所示



您可以点击右上角**按钮**，自行设定**监控时间**、**手动刷新**、**自动刷新**。

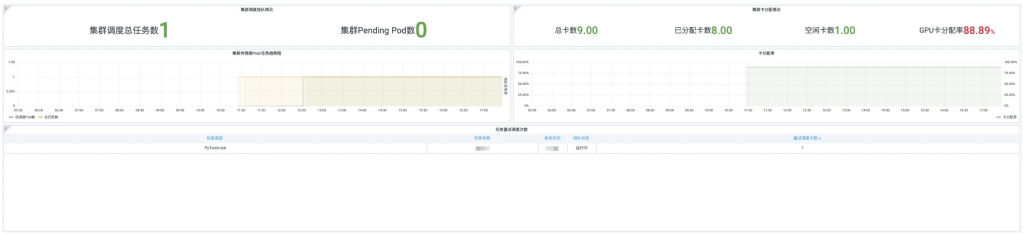
AI Job Scheduler组件具体说明

组件健康度总览

监控项	说明
调度器实例数	AI Job Scheduler Pod实例数，默认为3副本。 指标异常：调度器实例数为1-3为正常，为0为异常。
CPU利用率	AI Job Scheduler master实例的CPU利用率
内存占用量	AI Job Scheduler master实例的内存占用量
Pod 调度吞吐 CPM (次数/分钟)	调度器每分钟调度的Pod个数
任务调度吞吐 CPM (次数/分钟)	调度器每分钟调度的任务个数

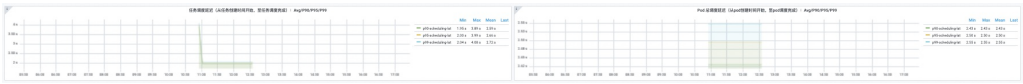
待调度Pod/任务排队总览

监控项	说明
集群调度排队情况	集群中调度任务排队情况，展示了集群待调度任务数、集群排队任务数、集群调度总任务数、集群Pending Pod数、集群任务排队率。
集群卡分配情况	集群中GPU卡分配情况，展示了GPU总卡数、已分配GPU卡数、空闲GPU卡数、不可用GPU卡数、GPU卡分配率。
集群待调度Pod/任务趋势图	详细的集群中待调度Pod/任务趋势，展示了待调度Pod数、待调度任务数、排队任务数、总任务数等数据的排队任务率。
卡分配率	详细的集群中GPU卡分配率。
任务重试调度次数	调度器持续调度任务的次数，还包含任务类型、任务名称、命名空间、排队状态。



Pod/任务调度性能总览

监控项	说明
任务调度延迟	从任务创建时间开始，至任务调度完成的时间
Pod 总调度延迟	从pod创建时间开始，至pod调度完成的时间



调度延迟action总览

监控项	说明
调度器各阶段延迟分布	调度器各阶段延迟分布范围
Pod 调度延迟	从pod调度算法+绑定阶段时间之和
调度器各阶段延迟	调度器各个调度阶段的调度时间

GPU工作负载资源

GPU 工作负载资源，包含任务属性、GPU 卡数、GPU 资源使用情况、GPU 节点使用概要。

前提条件

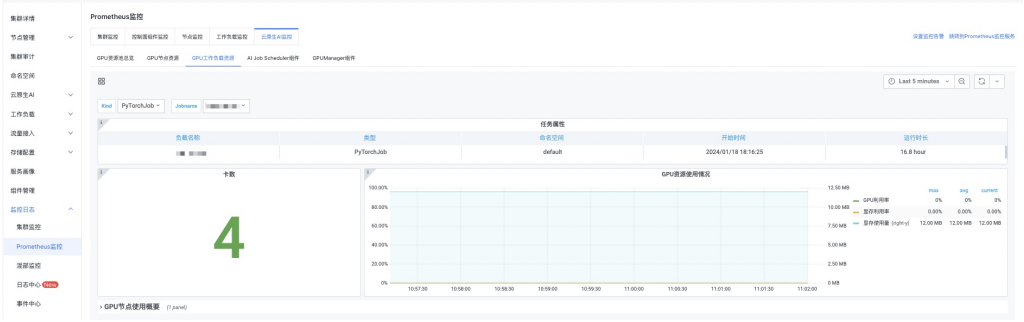
- 已安装 CCE AI Job Scheduler 组件且版本 >= 1.7.9
- 已安装 CCE GPU Manager 组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**GPU工作负载资源**。

GPU工作负载资源如图所示



您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

GPU工作负载资源具体说明

任务属性

监控项	说明
负载名称	当前工作负载的名称
类型	当前工作负载的类型
命名空间	当前工作负载所在命名空间
开始时间	当前工作负载的开始时间
运行时长	当前工作负载的运行时长

卡数&GPU资源使用情况

监控项	说明
卡数	当前工作负载的GPU卡数
GPU利用率	当前工作负载所有GPU的平均利用率实时值
显存利用率	当前工作负载所有显存的平均利用率实时值
显存使用量	当前工作负载所有显存的使用量实时值

GPU节点使用概要

监控项	说明
命名空间	当前工作负载中GPU节点所在命名空间
节点IP	当前工作负载中GPU节点IP
Pod名称	当前工作负载中GPU节点中运行Pod名称
GPU配给卡数	当前工作负载中GPU节点的GPU配给卡数
GPU平均利用率	当前工作负载中GPU节点的GPU平均利用率
显存使用量	当前工作负载中GPU节点的显存使用量
显存平均利用率	当前工作负载中GPU节点的显存平均利用率

GPU节点资源

GPU 节点资源，包含占用 GPU 的 Pod 数量、GPU 卡分配情况、GPU 卡使用情况、GPU & Memory 总量、GPU & Memory 利用率、GPU 卡平均利用率、GPU 卡利用率、卡平均显存利用率、卡显存利用率、GPU 利用率、Memory 利用率、运行中的占 GPU 的 Pod 列表。

前提条件

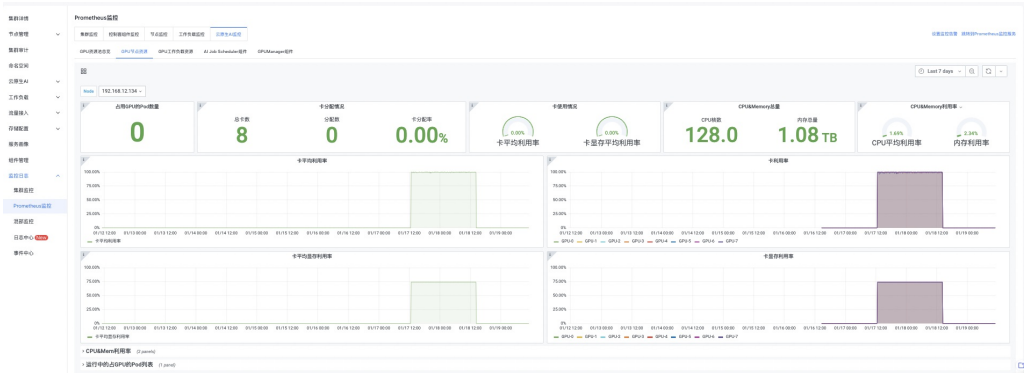
- 已安装 CCE AI Job Scheduler 组件且版本 >= 1.7.9
- 已安装 CCE GPU Manager 组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**GPU节点资源**。

GPU节点资源如图所示



您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

GPU节点资源具体说明

占用GPU的Pod数量

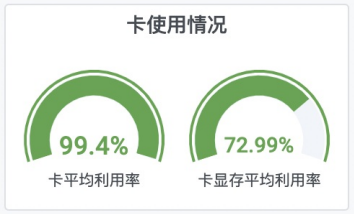
监控项	说明
占用GPU的Pod数量	当前节点内占用GPU资源的Pod数量

卡分配情况

监控项	说明
总卡数	当前节点内全部GPU卡数
分配数	当前节点内已分配的GPU卡数
卡分配率	分配率=已分配GPU卡数/总GPU卡数

卡使用情况

监控项	说明
卡平均利用率	当前节点内所有GPU卡平均利用率实时值，卡平均利用率=sum(所有GPU卡利用率)/所有GPU卡数
卡显存平均利用率	当前节点内所有GPU卡显存平均利用率实时值，显存平均利用率=sum(所有GPU卡显存利用率)/所有GPU卡数

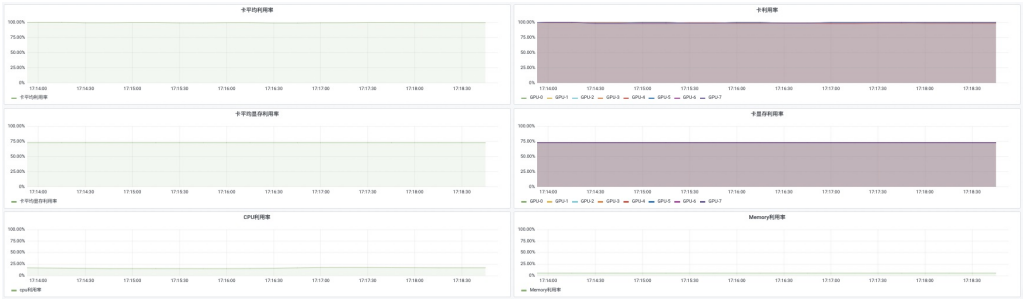


CPU&Memory总量/利用率

监控项	说明
CPU核数	当前节点内CPU总核数
CPU平均利用率	当前节点内所有CPU平均利用率实时值
内存总量	当前节点内存总量
内存利用率	当前节点内所有内存平均利用率实时值

利用率

监控项	说明
卡平均利用率	当前节点内所有GPU卡平均利用率实时值，卡平均利用率=sum(所有GPU卡利用率)/所有GPU卡数
卡利用率	当前节点内所有GPU卡利用率实时值
卡平均显存利用率	当前节点内所有GPU卡显存平均利用率实时值，显存平均利用率=sum(所有GPU卡显存利用率)/所有GPU卡数
卡显存利用率	当前节点内所有GPU卡显存利用率实时值
CPU利用率	当前节点内所有CPU利用率实时值
Memory利用率	当前节点内所有内存利用率实时值



运行中的占GPU的Pod列表

监控项	说明
负载名称	当前节点运行中的占GPU的Pod所在工作负载名称
类型	当前节点运行中的占GPU的Pod任务类型
命名空间	当前节点运行中的占GPU的Pod所在命名空间
Pod名称	当前节点运行中的占GPU的Pod名称
GPU配给卡数	当前节点运行中的占GPU的Pod的配给GPU卡数
GPU平均利用率	当前节点运行中的占GPU的Pod内GPU卡平均利用率实时值
GPU显存平均利用率	当前节点运行中的占GPU的Pod内GPU卡显存平均利用率实时值
内存使用量	当前节点运行中的占GPU的Pod的内存使用量
CPU核数	当前节点运行中的占GPU的Pod的CPU核数

GPU资源池总览

GPU资源池总览，包含节点使用情况、集群GPU卡分配情况、GPU卡使用情况、CPU&内存使用情况、GPU卡分配数、GPU卡分配率、GPU卡利用率、GPU卡显存利用率、节点信息、占用GPU的工作负载信息。 **前提条件**

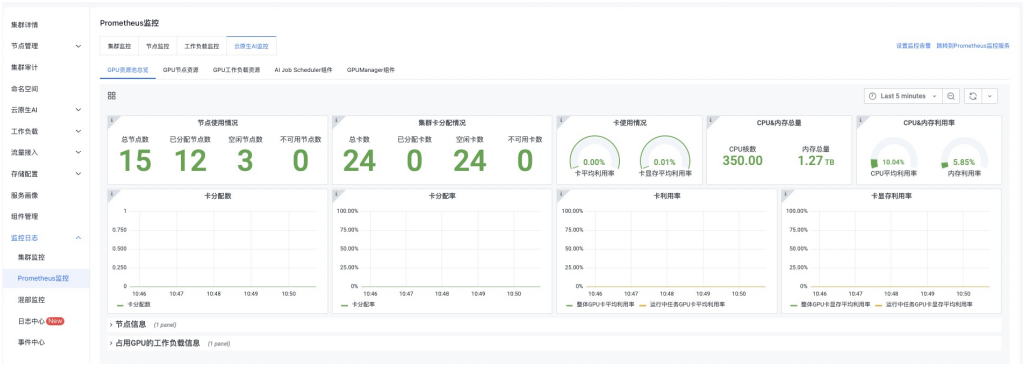
- 已安装CCE AI Job Scheduler组件且版本 >= 1.7.9
- 已安装CCE GPU Manager组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**GPU资源池总览**。

GPU资源池总览如图所示



您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

GPU资源池总览具体说明

节点使用情况

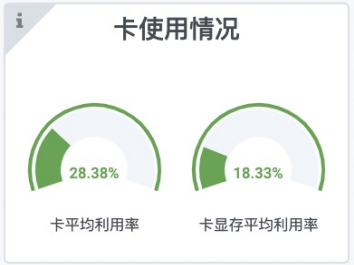
监控项	说明
总节点数	集群中所有节点
已分配节点数	可用GPU卡数为0的节点
空闲节点数	GPU卡数大于0的节点，含污点节点
不可用节点数	封锁或not ready节点

集群卡分配情况

监控项	说明
总卡数	集群中所有节点上的GPU卡数
已分配卡数	已分配使用中的GPU卡数
空闲卡数	GPU卡数大于0的节点，含污点节点空闲卡数
不可用卡数	集群中节点上的不可用卡数

卡使用情况

监控项	说明
卡平均利用率	当前集群内所有节点上GPU卡平均利用率实时值，卡平均利用率=sum(所有节点GPU卡利用率)/所有节点GPU卡数
卡显存平均利用率	当前集群内所有节点上GPU卡显存平均利用率实时值，显存平均利用率=sum（所有节点GPU卡显存利用率）/所有节点GPU卡数

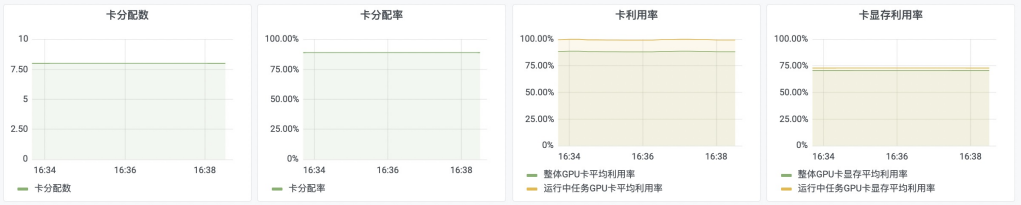


CPU&内存

监控项	说明
CPU核数	当前集群内CPU总核数
CPU平均利用率	当前集群内所有CPU平均利用率实时值
内存总量	当前集群内存总量
内存平均利用率	当前集群内所有内存平均利用率实时值

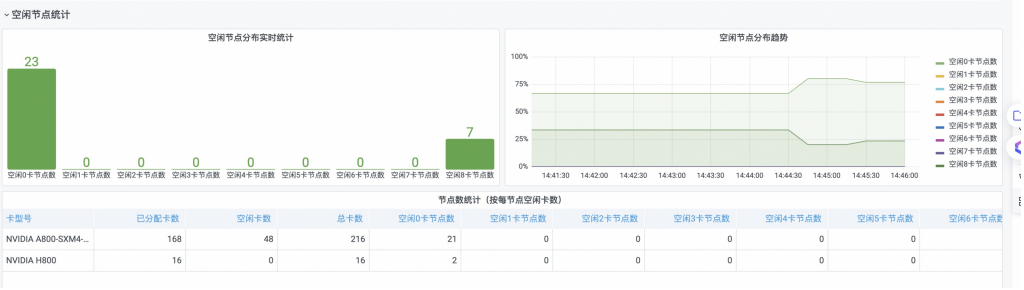
利用率 & 分配率

监控项	说明
卡分配数	已分配GPU卡数
卡分配率	分配率=已分配GPU卡数/总GPU卡数
整体GPU平均利用率	当前集群内所有节点上GPU卡平均利用率实时值，平均利用率=sum(所有节点GPU卡利用率)/所有节点GPU卡数
运行中任务GPU平均利用率	GPU平均利用率=sum(已分配GPU卡利用率)/已分配GPU卡数
整体GPU显存平均利用率	当前集群内所有节点上GPU卡显存平均利用率实时值，显存平均利用=sum(所有节点GPU卡显存利用率)/所有节点GPU卡数
运行中任务GPU显存平均利用率	GPU显存平均利用率=sum(已分配GPU卡显存利用率)/已分配GPU卡数



空闲节点统计

监控项	说明
空闲节点分布实时统计	当前集群内的空闲节点数量分布情况
空闲节点分布趋势	当前集群内的空闲节点分布历史趋势
卡型号	当前集群中已有的GPU卡型号
已分配卡数	集群中已分配使用的该型号GPU的数量
空闲卡数	集群中空闲未被分配的该型号GPU数量
总卡数	集群中该型号GPU卡总数量
空闲x卡节点数	集群中单节点空闲x卡的节点数量



节点信息

监控项	说明
节点名称	当前集群内的节点名称
分配卡数	当前集群内节点上分配的GPU卡数
GPU-Pod数	当前节点内占用GPU资源的Pod数量
CPU利用率	当前节点内所有CPU平均利用率实时值
内存利用率	当前节点内所有内存平均利用率实时值
节点状态	当前节点状态
CPU核数	当前节点内CPU总核数
内存总量	当前节点内存总量

节点信息

节点名称	节点状态	卡型号	卡总数	已分配卡数	GPU-Pod数	GPU平均利用率	GPU显存平均利用率	CPU核数
[redacted]	Ready		0.00	0.00	0	0.00%	0.00%	2
[redacted]	Ready	NVIDIA A800-SXM4-80GB	8.00	8.00	1	99.25%	72.99%	128
[redacted]	Ready	NVIDIA A10	1.00	0.00	0	0.00%	0.00%	16

占用GPU的工作负载信息

监控项	说明
负载名称	当前集群占用GPU的工作负载名称
类型	当前集群占用GPU的工作负载类型
命名空间	当前集群占用GPU的工作负载所在命名空间
开始时间	当前集群占用GPU的工作负载开始时间
运行时长	当前集群占用GPU的工作负载运行时长
GPU配给卡数	当前集群占用GPU的工作负载配给GPU卡数
显存平均利用率	当前集群占用GPU的工作负载内所有GPU卡显存平均利用率实时值
GPU平均利用率	当前集群占用GPU的工作负载内GPU卡平均利用率实时值
内存使用量	当前集群占用GPU的工作负载的内存使用量
CPU核数	当前集群占用GPU的工作负载的CPU核数

占用GPU的工作负载信息

负载名称	类型	命名空间	开始时间	运行时长	GPU配给卡数	显存平均利用率	GPU平均利用率	内存使用量
[redacted]	PyTorchJob	[redacted]	[redacted]	4.04 hour	8.00	72.99%	99.25%	30.12 GB

昇腾芯片资源观测

昇腾节点资源

前提条件

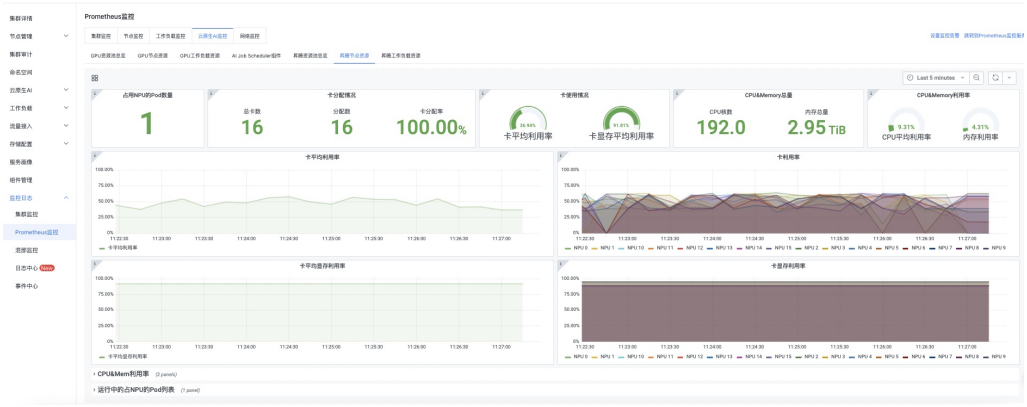
- 已安装CCE Ascend Mindx DL组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**昇腾节点资源**。

昇腾节点资源如图所示



您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

昇腾节点资源具体说明

占用NPU的Pod数量

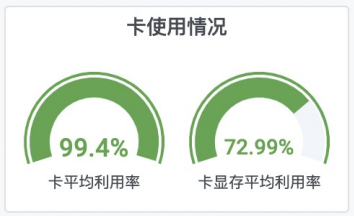
监控项	说明
占用NPU的Pod数量	当前节点内占用NPU资源的Pod数量

卡分配情况

监控项	说明
总卡数	当前节点内全部NPU卡数
分配数	当前节点内已分配的NPU卡数
卡分配率	分配率=已分配NPU卡数/总NPU卡数

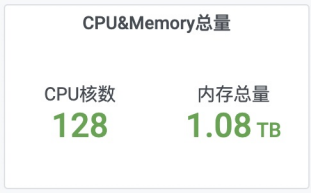
卡使用情况

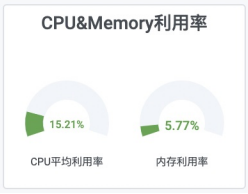
监控项	说明
卡平均利用率	当前节点内所有NPU卡平均利用率实时值，卡平均利用率=sum(所有NPU卡利用率)/所有NPU卡数
卡显存平均利用率	当前节点内所有NPU卡显存平均利用率实时值，显存平均利用率=sum(所有NPU卡显存利用率)/所有NPU卡数



CPU&Memory总量/利用率

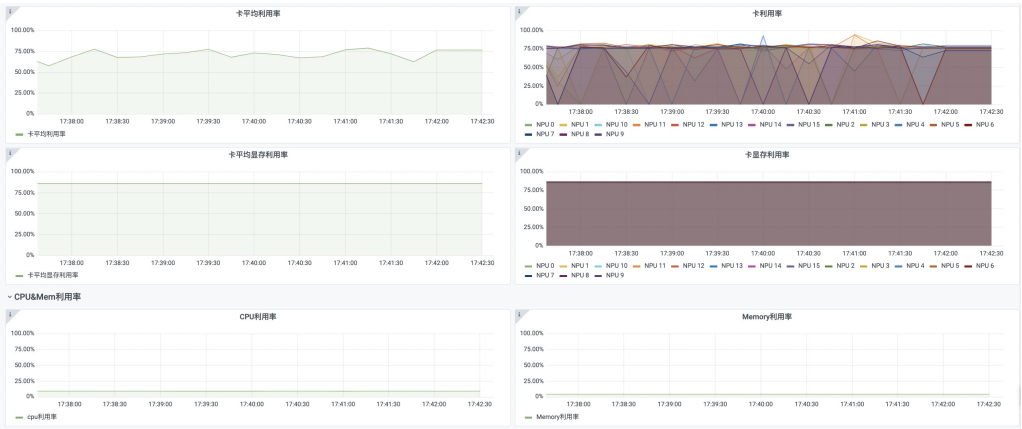
监控项	说明
CPU核数	当前节点内CPU总核数
CPU平均利用率	当前节点内所有CPU平均利用率实时值
内存总量	当前节点内存总量
内存利用率	当前节点内所有内存平均利用率实时值





利用率

监控项	说明
卡平均利用率	当前节点内所有NPU卡平均利用率实时值，卡平均利用率=sum(所有NPU卡利用率)/所有NPU卡数
卡利用率	当前节点内所有NPU卡利用率实时值
卡平均显存利用率	当前节点内所有NPU卡显存平均利用率实时值，显存平均利用率=sum(所有NPU卡显存利用率)/所有NPU卡数
卡显存利用率	当前节点内所有NPU卡显存利用率实时值
CPU利用率	当前节点内所有CPU利用率实时值
Memory利用率	当前节点内所有内存利用率实时值



运行中的占NPU的Pod列表

监控项	说明
命名空间	当前节点运行中的占NPU的Pod所在命名空间
节点名称	运行中的占NPU的Pod所在节点名称
Pod	当前节点运行中的占NPU的Pod名称
负载名称	当前节点运行中的占NPU的Pod所在工作负载名称
NPU配给卡数	当前节点运行中的占NPU的Pod的配给NPU卡数
NPU平均利用率	当前节点运行中的占NPU的Pod内NPU卡平均利用率实时值
NPU显存平均利用率	当前节点运行中的占NPU的Pod内NPU卡显存平均利用率实时值
内存使用量	当前节点运行中的占NPU的Pod的内存使用量
CPU核数	当前节点运行中的占NPU的Pod的CPU核数

昇腾工作负载资源

前提条件

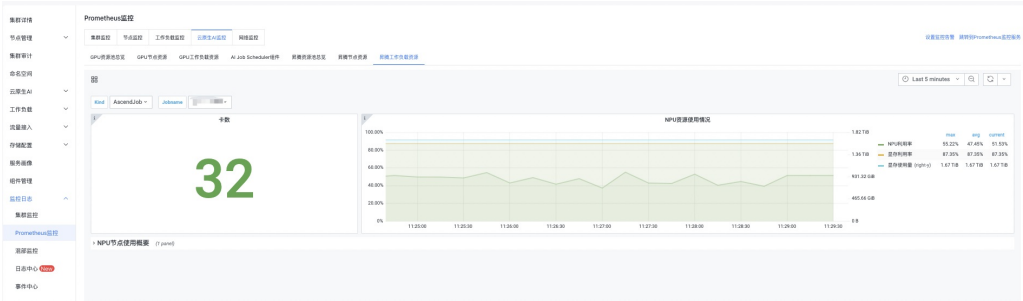
- 已安装CCE Ascend Mindx DL组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**昇腾工作负载资源**。

昇腾工作负载资源如图所示



您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

昇腾工作负载资源具体说明

卡数&NPU资源使用情况

监控项	说明
卡数	当前工作负载的NPU卡数
NPU利用率	当前工作负载所有NPU的平均利用率实时值
显存利用率	当前工作负载所有显存的平均利用率实时值
显存使用量	当前工作负载所有显存的使用量实时值

NPU节点使用概要

监控项	说明
命名空间	当前工作负载中NPU节点所在命名空间
节点名称	当前工作负载中NPU节点名称
Pod名称	当前工作负载中NPU节点中运行Pod名称
NPU配给卡数	当前工作负载中NPU节点的NPU配给卡数
NPU平均利用率	当前工作负载中NPU节点的NPU平均利用率
显存使用量	当前工作负载中NPU节点的显存使用量
显存平均利用率	当前工作负载中NPU节点的显存平均利用率

昇腾资源池总览

前提条件

- 已安装CCE Ascend Mindx DL组件
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。

3. 在**Prometheus监控页面**下方选项中选择**云原生AI监控**，随后选择**昇腾资源池总览**。

昇腾资源池总览如图所示

您可以点击右上角**按钮**，自行设定**监控时间、手动刷新、自动刷新**。

昇腾资源池总览具体说明

节点使用情况

监控项	说明
总节点数	集群中所有节点
已分配节点数	可用NPU卡数为0的节点
空闲节点数	NPU卡数大于0的节点，含污点节点
不可用节点数	封锁或not ready节点

集群卡分配情况

监控项	说明
总卡数	集群中所有节点上的NPU卡数
已分配卡数	已分配使用中的NPU卡数
空闲卡数	NPU卡数大于0的节点，含污点节点空闲卡数
不可用卡数	集群中节点上的不可用卡数

卡使用情况

监控项	说明
卡平均利用率	当前集群内所有节点上NPU卡平均利用率实时值，卡平均利用率=sum(所有节点NPU卡利用率)/所有节点NPU卡数
卡显存平均利用率	当前集群内所有节点上NPU卡显存平均利用率实时值，显存平均利用率=sum（所有节点NPU卡显存利用率）/所有节点NPU卡数

CPU&内存

监控项	说明
CPU核数	当前集群内CPU总核数
CPU平均利用率	当前集群内所有CPU平均利用率实时值
内存总量	当前集群内存总量
内存平均利用率	当前集群内所有内存平均利用率实时值

NPU节点信息

监控项	说明
节点名称	当前集群内的节点名称
卡类型	当前集群内的卡类型
CPU利用率	当前节点内所有CPU平均利用率实时值
内存利用率	当前节点内所有内存平均利用率实时值
CPU核数	当前节点内CPU总核数
内存总量	当前节点内存总量
卡总数	当前节点NPU卡总数
NPU平均利用率	当前集群内节点上NPU卡的平均利用率
NPU-Pod数	当前节点内占用NPU资源的Pod数量
已分配卡数	当前集群内节点上分配的NPU卡数
NPU显存平均利用率	当前集群内节点上NPU卡显存的平均利用率

占用NPU的工作负载信息

监控项	说明
负载类型	当前集群占用NPU的工作负载类型
负载名称	当前集群占用NPU的工作负载名称
命名空间	当前集群占用NPU的工作负载所在命名空间
NPU配给卡数	当前集群占用NPU的工作负载配给NPU卡数
显存平均利用率	当前集群占用NPU的工作负载内所有NPU卡显存平均利用率实时值
NPU平均利用率	当前集群占用NPU的工作负载内NPU卡平均利用率实时值
内存使用量	当前集群占用NPU的工作负载的内存使用量
CPU核数	当前集群占用NPU的工作负载的CPU核数

接入监控实例并开启采集任务

若您需要使用云原生AI资源观测功能，需进行以下步骤，分别为您的集群接入监控实例并开启采集任务。

操作步骤

1. 登录容器引擎CCE控制台。
2. 点击左侧**集群管理**，在**集群列表**中选择您需要的**集群名称**，在右侧**操作>更多**中点击**Prometheus监控**，即可跳转至**Prometheus监控服务**。
3. 接入监控实例：检查当前集群是否被**CProm实例**关联。

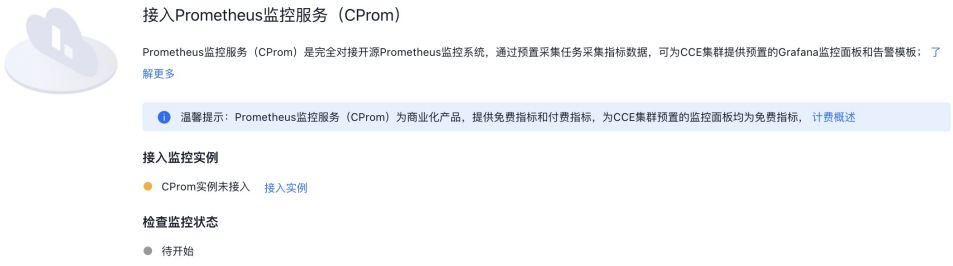
- **关联**：则进入下一步
- **未关联**：显示未关联状态，显示“接入实例”操作

检查监控状态：检查**CProm实例**是能正常监控集群以及数据能够正常采集及展示。

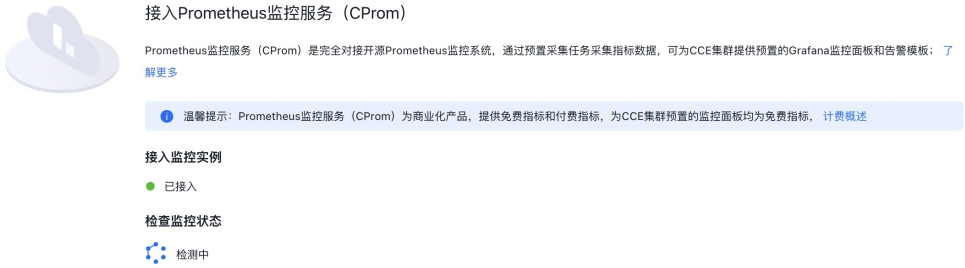
- **监控状态异常**：显示异常状态及信息
- **监控状态正常**：切换到预置监控面板页面

接入CProm监控：单击“**确定**”按钮系统需要前置校验CProm产品客户是否开通以及当前用户是否有相应操作权限，若**不满足**则不执行接入流程，并提示报错信息。

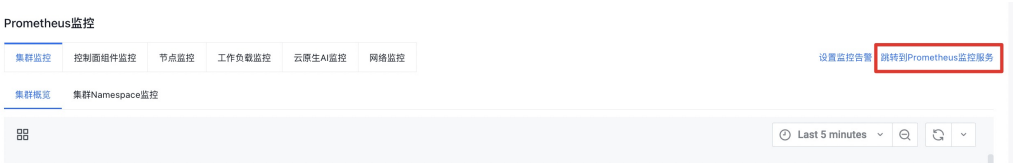
Prometheus监控



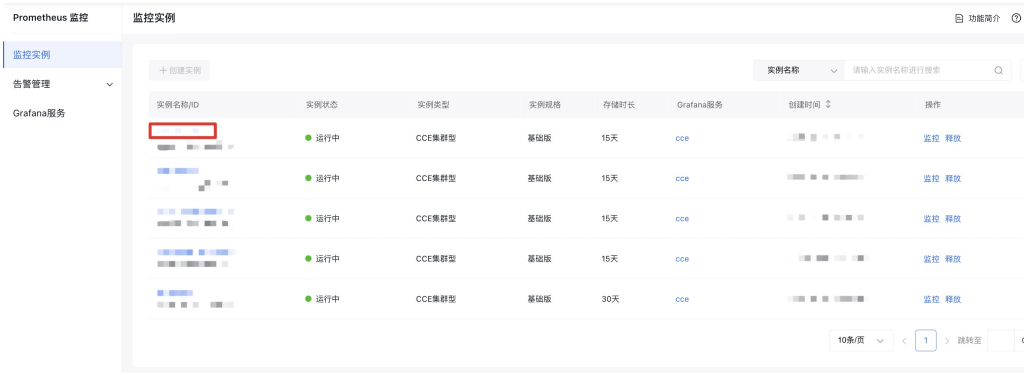
Prometheus监控



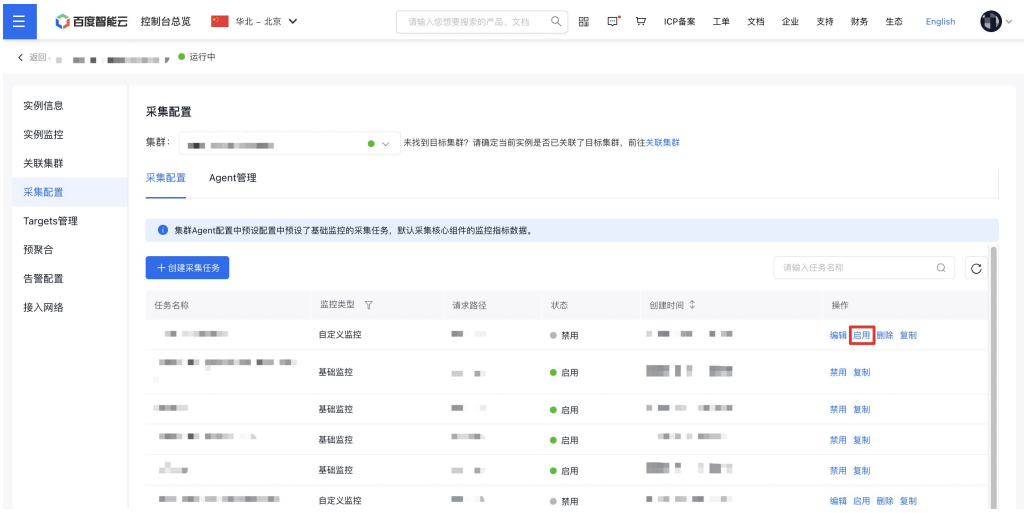
4. 接入成功后，在Prometheus监控页面右侧点击**跳转到Promethrus监控服务**



5. 选择您的实例，点击**实例名称**



6. 在左侧导航栏中选择**采集配置**，并在右侧选择**目标集群**，在下方**采集配置**列表中找到所需要的**任务名称**，在右侧**操作**中**点击启用**。任务状态由**禁用**转为**启用**。



GPU/NPU大盘需开启的采集任务

Nvidia GPU芯片采集项

大盘名称	采集任务					
	volcano	kubelet	gpu-dcgm	kubernetes-pods	cadvisor	kubernetes-pods-kube-state-metrics
GPU资源池总览	√	√	√	√	√	√
GPU节点资源	√	√	√	√	√	√
GPU工作负载资源	√	√	√	√	√	√
AI Job Scheduler组件	√	√	√	√	√	√
GPUManager组件	—	—	—	—	—	√

昇腾 AscendNPU芯片采集项

大盘名称	采集任务			
	npu-exporter	kubelet	cadvisor	kubernetes-pods-kube-state-metrics
昇腾资源池总览	√	√	√	√
昇腾节点资源	√	√	√	√
昇腾工作负载资源	√	√	√	√

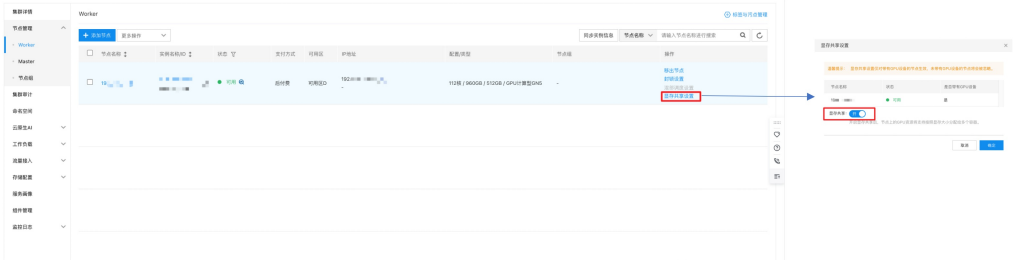
GPU虚拟化

GPU独占和共享说明

若集群支持 GPU 算力和显存的共享与隔离，您可以在新建任务或工作负载的时候，通过控制台操作或根据提交的 YAML 来决定独占还是共享 GPU 资源。

前提条件

- 您已成功安装CCE GPU Manager 和 CCE AI Job Scheduler 组件。您可在“集群 > 组件管理 > 云原生AI”中安装。
- 节点已开启显存共享，您可在“集群>节点管理>Worker>显存共享配置”中开启。



GPU卡型号对应资源名称

为保证正常使用GPU资源，请正确指定GPU卡型号对应的资源名称，本文以使用GPU卡数量为例，若想指定GPU卡算力资源或显存资源，则在资源名称后加上“_core”或“_memory”即可。

GPU卡型号	资源名称
NVIDIA V100 16GB	baidu.com/v100_16g_cgpu
NVIDIA V100 32GB	baidu.com/v100_32g_cgpu
NVIDIA T4	baidu.com/t4_16g_cgpu
NVIDIA A100 80GB	baidu.com/a100_80g_cgpu
NVIDIA A100 40GB	baidu.com/a100_40g_cgpu
NVIDIA A800 80GB	baidu.com/a800_80g_cgpu
NVIDIA A30	baidu.com/a30_24g_cgpu
NVIDIA A10	baidu.com/a10_24g_cgpu

资源描述

资源名称	类型	单位	说明
baidu.com/v100_32g_cgpu	int64	1	GPU 卡数量，共享场景下请填 1
baidu.com/v100_32g_cgpu_core	int64	5%	GPU 卡算力，eg.100=单卡总算力，10=单卡算力的10%，要求最低5%
baidu.com/v100_32g_cgpu_memory	int64	GiB	GPU 卡显存
baidu.com/v100_32g_cgpu_memory_percent	int64	1%	按百分比申请 GPU 卡显存，eg.100=单卡总显存 ，10=单卡显存的10%

控制台创建任务示例

若您通过CCE控制台创建任务（操作步骤可参考云原生AI任务管理），您可在任务的容器组配置中指定GPU类型为独占或共享。

单卡/多卡独占示例

- 当您指定GPU类型为独占GPU时，若希望任务使用整张GPU卡资源，操作步骤如下：

1. 选择GPU卡型号。
2. 输入GPU卡数量，GPU卡数量需输入正整数，范围为[1~当前集群中单节点所含GPU卡最大数量]。

- 若希望任务仅使用CPU和内存资源，无需使用GPU资源，操作步骤如下：

GPU卡型号保持为空，仅输入所需CPU和内存资源量。

单卡共享【不进行算力隔离，只有显存隔离】示例

当您指定GPU类型为共享GPU时，若希望任务只进行显存隔离，算力无需隔离，操作步骤如下：

1. 选择GPU卡型号。
2. 算力值保持为空。
3. 输入所需GPU显存，显存大小应为正整数，范围为[1~所选GPU卡显存大小]。

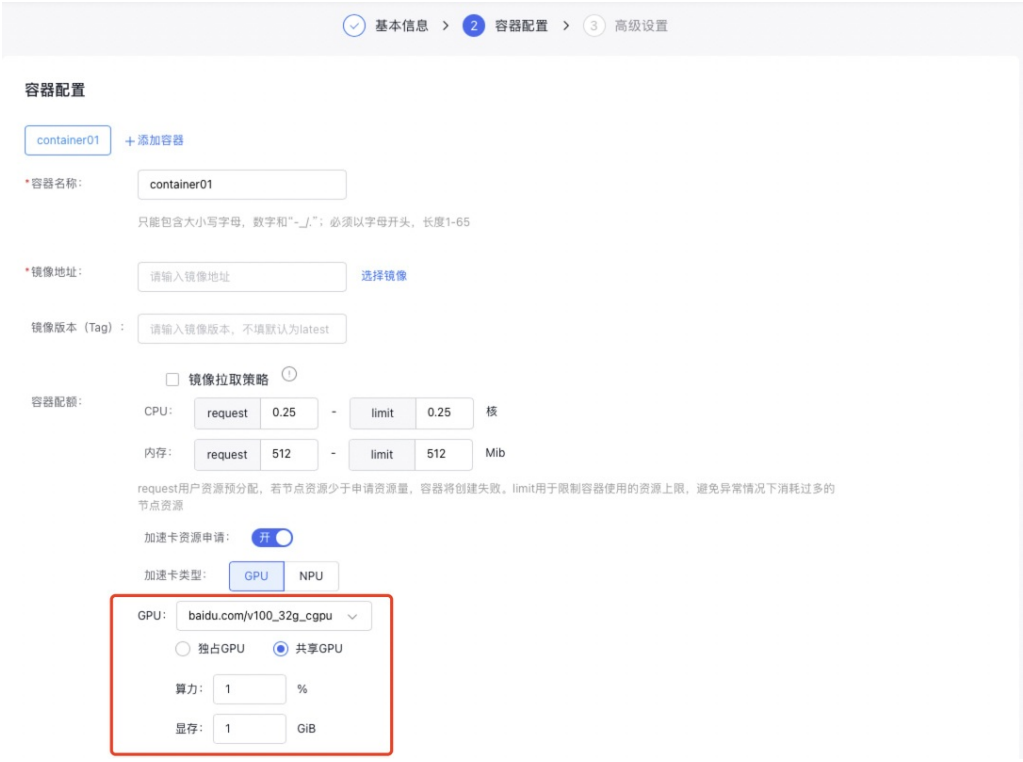
单卡共享【同时支持显存隔离和算力隔离】示例

当您指定GPU类型为共享GPU时，若希望任务同时进行显存隔离和算力隔离，操作步骤如下：

1. 选择GPU卡型号。
2. 输入所需算力值百分比，百分比大小应为[5~100正整数]。
3. 再输入所需GPU显存，显存大小应为正整数，范围为[1~所选GPU卡显存大小]。

控制台创建工作负载示例

若您通过CCE控制台创建工作负载（操作步骤可参考工作负载），您可在工作负载的容器配置中指定GPU类型为独占或共享。独占和共享的资源输入限制和上文中AI任务的创建一致。



Yaml创建任务/工作负载示例 若您通过Yaml创建任务或工作负载（详细配置可参考云原生AI任务管理、工作负载），您可在Yaml配置中指定所需GPU卡资源为独占或共享，具体示例如下：

> 备注：
> 1、如使用Yaml的方式创建任务，调度器必须指定 `schedulerName: volcano`
> 2、GPU显存资源必须申请，且 `baidu.com/v100_32g_cgpu_memory` 和 `baidu.com/v100_32g_cgpu_memory_percent`只能填写一个，不可同时填写

单卡独占示例：

```
resources:
requests:
  baidu.com/v100_32g_cgpu: 1 // 1卡
  cpu: "4"
  memory: 60Gi
limits:
  baidu.com/v100_32g_cgpu: 1 // limit与request必须一致
  cpu: "4"
  memory: 60Gi
```

多卡独占示例：

```
resources:
requests:
  baidu.com/v100_32g_cgpu: 2 // 2卡
  cpu: "4"
  memory: 60Gi
limits:
  baidu.com/v100_32g_cgpu: 2 // limit与request必须一致
  cpu: "4"
  memory: 60Gi
```

单卡共享【不进行算力隔离，只有显存隔离】示例：

```
resources:
 requests:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_memory: 10 // 10GB
   cpu: "4"
   memory: 60Gi
 limits:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_memory: 10
   cpu: "4"
   memory: 60Gi
```

单卡共享【同时支持显存隔离和算力隔离】示例：

```
resources:
 requests:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_core: 50 // 50%, 0.5卡算力
   baidu.com/v100_32g_cgpu_memory: 10 // 10GB
   cpu: "4"
   memory: 60Gi
 limits:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_core: 50 //
   baidu.com/v100_32g_cgpu_memory: 10
   cpu: "4"
   memory: 60Gi
```

编解码功能使用示例（仅支持性能最优型组件）：

```
resources:
 requests:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_core: "20" // 20%, 0.2卡算力
   baidu.com/v100_32g_cgpu_memory: "2" // 2
   baidu.com/v100_32g_cgpu_encode: 100 // 开启编码功能
   baidu.com/v100_32g_cgpu_decode: 100 // 开启解码功能
   memory: "2Gi"
   cpu: "1"
 limits:
   baidu.com/v100_32g_cgpu: 1
   baidu.com/v100_32g_cgpu_core: "20"
   baidu.com/v100_32g_cgpu_memory: "2"
   baidu.com/v100_32g_cgpu_encode: 100
   baidu.com/v100_32g_cgpu_decode: 100
```

单GPU容器共享多卡使用说明

本文介绍通过指定资源描述符使用sGPU的单容器多卡能力，分为GPU显存/算力同时隔离以及仅隔离显存的场景。 **1. GPU显存/算力同时隔离**

资源描述符	单位	含义
baidu.com/xxx_xxx_cgpu	个数	申请的GPU卡个数，该场景表示申请的GPU共享卡个数
baidu.com/xxx_xxx_cgpu_core	百分比	容器申请的总算力
baidu.com/xxx_xxx_cgpu_memory	GiB	容器申请的总显存

单张共享卡的资源：

- 每张 GPU卡的算力资源：baidu.com/xxx_xxx_cgpu_core/baidu.com/xxx_xxx_cgpu
- 每张 GPU卡的显存资源：baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu

资源描述示例，如下示例代表：该Pod共申请50%的算力，以及10GiB的显存，2张GPU共享卡。所以每张GPU共享卡的资源为 25%的算力以及5GiB显存。

```
resources:
 limits:
   baidu.com/a10_24g_cgpu: "2"
   baidu.com/a10_24g_cgpu_core: "50"
   baidu.com/a10_24g_cgpu_memory: "10"
```

2. GPU显存隔离，算力共享 | 资源描述符 | 单位 | 含义 | | --- | --- | --- | | baidu.com/xxx_xxx_cgpu | 个数 | 申请的GPU卡个数，该场景表示申请的GPU共享卡个数 | | baidu.com/xxx_xxx_cgpu_memory | GiB | 容器申请的总显存 | 单张共享卡的资源：

- 每张 GPU卡的算力资源：与其他容器共享100%的算力。
- 每张 GPU卡的显存资源：baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu

资源描述示例，如下示例代表：该Pod共申请10GiB的显存，2张GPU共享卡。所以每张GPU共享卡的资源为 共享100%的算力以及5GiB显存。

```
resources:
 limits:
   baidu.com/a10_24g_cgpu: "2"
   baidu.com/a10_24g_cgpu_memory: "10"
```

使用限制

1. 单卡的显存/算力需要为正整数， 即算力（baidu.com/xxx_xxx_cgpu_core/baidu.com/xxx_xxx_cgpu），显存（baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu ）需为正整数。
2. 单卡的显存/算力需要大于等于显存/算力的最小单位。
3. 如果没有申请_cgpu_memory或者_cgpu_memory_percent，是不允许申请_cgpu_core
4. 显存隔离最小单位：1GiB。
5. 算力隔离最小单位：5%。

GPU虚拟化适配表

注意事项

- 使用性能最优型时，镜像不推荐设置LD_LIBRARY_PATH环境变量，如果需要设置，需要把/usr/lib64目录按照LD_LIBRARY_PATH=/usr/lib64:$LD_LIBRARY_PATH 的形式添加进去

隔离最优型适配支持

参数	要求		
GPU卡类型	支持的显卡类型包括：NVIDIA A10、A30、A100、A800、V100、T4、P4		
操作系统	支持的操作系统包括:Centos、Ubuntu、BaiduLinux		
版本 & GPU Manager 版本&内核版本	操作系统版本	GPU Manager版本	内核版本
	CentOS	1.5.11	3.10.0-957.21.3.el7.x86_64、3.10.0-1160.42.2.el7.x86_64 3.10.0-1160.62.1.el7.x86_64、3.10.0-1160.80.1.el7.x86_64 3.10.0-1160.81.1.el7.x86_64、4.17.11-1.el7.elrepo.x86_64 5.4.123-1.el7.elrepo.x86_64
		1.5.12	3.10.0-1160.42.2.el7.x86_64、3.10.0-1160.62.1.el7.x86_64 3.10.0-1160.80.1.el7.x86_64、3.10.0-1160.81.1.el7.x86_64 3.10.0-1160.83.1.el7.x86_64、3.10.0-957.21.3.el7.x86_64 5.4.123-1.el7.elrepo.x86_64、4.17.11-1.el7.elrepo.x86_64
		1.5.14-1.5.24	3.10.0-1160.71.1.el7.x86_64、3.10.0-1160.76.1.el7.x86_64 3.10.0-1160.80.1.el7.x86_64、3.10.0-1160.81.1.el7.x86_64 3.10.0-1160.41.1.el7.x86_64、3.10.0-1160.42.2.el7.x86_64 3.10.0-1160.45.1.el7.x86_64、3.10.0-1160.62.1.el7.x86_64 3.10.0-1160.83.1.el7.x86_64、3.10.0-1160.88.1.el7.x86_64

			3.10.0-1160.90.1.el7.x86_64、3.10.0-957.21.3.el7.x86_64 5.4.123-1.el7.elrepo.x86_64
	1.5.25		3.10.0-1160.80.1.el7.x86_64、3.10.0-1160.81.1.el7.x86_64 3.10.0-1160.83.1.el7.x86_64、3.10.0-1160.88.1.el7.x86_64 3.10.0-1160.41.1.el7.x86_64、3.10.0-1160.42.2.el7.x86_64 3.10.0-1160.45.1.el7.x86_64、3.10.0-1160.62.1.el7.x86_64 3.10.0-1160.71.1.el7.x86_64、3.10.0-1160.76.1.el7.x86_64 4.18.0-348.7.1.el8_5.x86_64、4.18.0-348.el8.x86_64 3.10.0-1160.90.1.el7.x86_64、3.10.0-957.21.3.el7.x86_64 4.17.11-1.el7.elrepo.x86_64、4.18.0-348.2.1.el8_5.x86_64 4.18.0-348.2.1.el8_5.x86_64
	1.5.27		3.10.0-1160.11.1.el7.x86_64、3.10.0-1160.15.2.el7.x86_64 3.10.0-1160.2.1.el7.x86_64、3.10.0-1160.2.2.el7.x86_64 3.10.0-1160.21.1.el7.x86_64、3.10.0-1160.24.1.el7.x86_64 3.10.0-1160.25.1.el7.x86_64、3.10.0-1160.31.1.el7.x86_64 3.10.0-1160.36.2.el7.x86_64、3.10.0-1160.41.1.el7.x86_64 3.10.0-1160.42.2.el7.x86_64、3.10.0-1160.45.1.el7.x86_64 3.10.0-1160.49.1.el7.x86_64、3.10.0-1160.53.1.el7.x86_64 3.10.0-1160.59.1.el7.x86_64、3.10.0-1160.6.1.el7.x86_64 3.10.0-1160.62.1.el7.x86_64、3.10.0-1160.66.1.el7.x86_64 3.10.0-1160.71.1.el7.x86_64、3.10.0-1160.76.1.el7.x86_64 3.10.0-1160.80.1.el7.x86_64、3.10.0-1160.81.1.el7.x86_64 3.10.0-1160.83.1.el7.x86_64、3.10.0-1160.88.1.el7.x86_64 3.10.0-1160.90.1.el7.x86_64、3.10.0-1160.el7.x86_64 3.10.0-957.21.3.el7.x86_64、4.17.11-1.el7.elrepo.x86_64 4.18.0-348.2.1.el8_5.x86_64、4.18.0-348.7.1.el8_5.x86_64 4.18.0-348.el8.x86_64、5.4.123-1.el7.elrepo.x86_64
Ubuntu	1.5.11	4.4.0	150
	1.5.12	4.4.0	150
		4.15.0	140、204
		5.4.0	139
	1.5.14-1.5.24	4.4.0	150
		4.15.0	140、204
		5.4.0	139、150
	1.5.25	4.4.0	150
		4.15.0	140、204
		5.4.0	139、150
		5.15.0	72
	1.5.27	4.4.0	150
		4.15.0	22~24、29、30、32~34、36、38、39、42~48、50~52、54、55、58、60、62、64~66、69、70、72、74、76、88、91、96、99、101、106、108、109、111、112、115、117、118、121~124、128~130、134~137、139~144、147、151、153、154、156、158、159、161、162、163、166、167、169、171、173、175~177、180、184、187~189、191~194、196、197、211、212
		5.4.0	26、28、29、31、33、37、39、40、42、45、47、48、51~54、58、59、100、104、105、107、109、113、117、120~122、124~126、128、131、132、135~137、139、150、152、153、155、156、159、162~164
		5.15.0	72、73、75、78、79、82~84、86、87
BaiduLinux3.0	1.5.11	5.10.0-3.0.1.3	
	1.5.12	5.10.0-3.0.1.3	
	1.5.14-1.5.24	5.10.0-3.0.1.3	
	1.5.25	5.10.0-3.0.1.3	

	1.5.25	5.10.0-3.0.1.3
	1.5.27	5.10.0-3.0.1.3

CUDA & GPU Driver版本	CUDA	GPU Driver
	10.1	418.x
	10.2	440.x
	11.0	450.x
	11.2	460.x
	11.4	470.x
	11.7	515.x
	11.8	520.x
	12.1 12.0	525.x
	12.2	535.x

性能最优型适配支持

参数	要求	
GPU卡类型	支持的显卡类型包括：NVIDIA A10、A30、A100、A800、V100、T4、P4	
CUDA & GPU Driver版本	CUDA	GPU Driver
	10.2	440
	11.2	460
	11.4	470.x
	11.7	515.76.x
	12.1	525.105.17

MPS 最佳实践&注意事项

GPU Manager 选择性能最优型时，会使用 Nvidia MPS进行算力隔离，MPS 只具备有限的容错能力，请仔细阅读以下内容以确保您对MPS有基本了解。

MPS使用最佳实践

1. 适合MPS的使用场景：
   - MPS仅推荐用于运行单个应用程序的协作进程，例如相同MPI作业的多个ranks，以便内存保护和容错方案可用。
   - 当每个应用程序进程没有产生足够的工作负载使GPU饱和时，MPS是有用的。可以使用MPS在每个节点上运行多个进程，以实现更高的并发性。
   - 程序中因为每个grid 的线程较少导致 GPU 利用率较低时，适合使用 MPS技术提升性能。推荐的做法是：在每个grid中申请更少的block，在每次kernel调用中，每个block下启动更多的线程，以提高每个block的利用率。MPS技术允许 CUDA kernel 中其他的进程使用剩余的 GPU 容量。
2. MPS客户端优雅退出：使用性能最优型的容器进程退出前先执行发送SIGUSR1信号给MPS进程再等待2秒后最终退出，（此过程中MPS进程收到信号后会使mps客户端挂起，不再发送kernel）。
3. MPS尽可能不要使用多GPU：通过MPS使用多卡时，容易引发故障传导，导致多卡上的任务都引发异常。
4. 平台&业务层高可用：上层平台尽可能实现业务层高可用，并支持容错及迁移。

注意事项

- MPS容错能力：
  - 由MPS客户端进程生成的致命异常（ fatal exception）时，与此客户端共享GPU的所有客户端，将会收到这个异常
  - MPS发生fatal exception后，并未指示是哪个客户端生成了错误，受影响的客户端进程应该自行捕捉fatal exception并退出。
  - MPS使用单卡时，运行在其他GPU上的客户端进程不受影响
  - 一旦观察到致命异常，MPS服务器将等待与受影响gpu关联的所有客户端退出，禁止连接到这些gpu的新客户端加入。当与受影响的GPU关联的所有现有客户端都退出时，MPS服务器将在受影响的GPU上重新创建GPU上下文，并继续处理客户端对这些GPU的请求。

- MPS服务端发生异常会导致所有客户端进程异常

- MPS不支持多用户： mps server是用户级别的，不同用户（linux user）使用MPS时，可能会出现一个用户在等另一个用户mps server退出而导致进程假死的情况。

- 推荐使用 CUDA 11.7以上的版本 官方说明：https://docs.nvidia.com/deploy/mps/index.html

共享GPU场景下的镜像构建注意事项

1. 以下环境变量会由 云原生AI组件 CCE GPU Manager组件进行注入，请不要添加到镜像的环境变量里面：

环境变量	说明
NVIDIA_VISIBLE_DEVICES	可见 GPU设备列表，由调度器分配
NVIDIA_VISIBLE_GPUS_SLOT	可见 GPU设备插槽，由调度器分配
NVIDIA_VISIBLE_GPUS_UUID	UUID 形式的可见 GPU设备列表，由调度器分配
LD_LIBRARY_PATH	不推荐设置LD_LIBRARY_PATH环境变量，如果有特殊需求需要设置，需要把/usr/lib64目录也按 LD_LIBRARY_PATH=/usr/lib64:$LD_LIBRARY_PATH 形式添加进去
CUDA_MPS_ACTIVE_THREAD_PERCENTAGE	MPS 算力隔离设置，由调度器分配
CUDA_MPS_LOG_DIRECTORY	MPS 相关日志地址
CUDA_MPS_PIPE_DIRECTORY	和 MPS SERVER通信的地址
CGPUX_XXX	以 CGPU 开头的环境变量，用于运行显存&算力隔离功能，比如CGPU0_PRIORITY，CGPU0_SHAREMODE
CGPU_COUNT	设备数量
SGPU_DISABLE	使用隔离最优型时，表示是否使用虚拟化的方式使用 GPU

2. 构建镜像时，请不要直接将集群中运行的 GPU 容器保存为镜像使用，此种方式保存的容器镜像会包含CCE GPU Manager组件注入的环境变量，造成虚拟化功能不符合预期或者不可用。

关闭节点显存共享功能注意事项

1.关闭节点显存共享功能说明

开启显存共享功能的节点需要关闭显存共享功能时，系统会验证当前节点上是否有正在运行的显存共享任务，需等该节点所有显存共享任务都运行完成后才能够关闭显存共享功能，否则可能导致显存共享任务中断或任务结束时无法正常回收资源，影响节点上的后续任务。

您可以通过2、3提供的命令查看开启了显存共享的节点和查看该节点上显存共享任务的情况。

2.查询开启显存共享的节点

```
kubectl get nodes -l cce.baidubce.com/gpu-share-device-plugin=enable
```

该命令会列出开启了显存共享功能的所有节点，这些节点上都有标签cce.baidubce.com/gpu-share-device-plugin:enable。

3.查询运行显存共享任务的Pods

```
kubectl get pods --all-namespaces -o json | jq -r '.items[] | select(.status.phase=="Running") | select(.spec.containers[].resources.limits // empty | keys[] // empty | test("baidu.com/.*(_core|_memory|_memory_percent)$")) | "\(.metadata.name) \(.spec.nodeName)"' | sort | uniq
```

该命令会列出所有在运行状态且使用了显存共享功能的 Pods，以及它们所在的节点。这些节点都无法马上关闭显存共享功能，需要等这些 Pods 结束以后才能关闭节点显存共享功能。

4.使用label命令修改节点标签的风险 除控制台方式外，可以使用kubectl label nodes命令来修改节点标签为cce.baidubce.com/gpu-share-device-plugin:disable，关闭节点的显存共享功能，修改前请关注以下风险：

- 业务中断：修改节点标签之后会触发非共享场景的环境安装，导致正在运行的显存共享任务中断或者结束时无法正常回收资源，影响节点上

的后续任务。

- 调度失败：修改标签之后，调度器可以向该节点调度非共享类型的任务，可能存在单机同时运行两种类型的任务，会有错卡，共享显存功能异常等问题。

为了避免这些风险，请始终确保集群中的节点标签正确地反映节点的能力，并且只有在确定节点上没有正在运行的显存共享任务时，再关闭节点显存共享功能。

组件管理

组件概述

容器引擎CCE支持多种不同类型的组件，包含网络、存储、监控、日志等，满足您多种业务场景需求。您可按需安装、升级或卸载组件。

组件类型

容器引擎CCE管理的集群组件类型包括系统组件和可选组件：

系统组件：创建CCE集群时，默认安装的组件。

可选组件：创建CCE集群时，可选择安装的组件，用于扩展集群功能。

组件介绍

核心组件

组件名称	类型	组件介绍
Kube APIServer	系统组件	Kubernetes 集群网关，是Kubernetes中与集群交互的主要接口。
Kube Scheduler	系统组件	Kubernetes 集群调度器，负责集群资源调度。
Kube Controller Manager	系统组件	Kubernetes 资源管理器，负责管理集群内部资源。
CCE Virtual Kubelet	可选组件	CCE 虚拟节点组件，提供极速接入BCI弹性容器能力。

网络

组件名称	类型	组件介绍
CCE Ingress Controller	可选组件	基于百度云应用型负载均衡产品（应用型BLB）实现K8S Ingress语义，提供七层网络负载均衡能力。
CCE Ingress Nginx Controller	可选组件	使用Nginx作为反向代理、负载均衡器、HTTP缓存，根据Ingress资源配置动态更新Nginx配置文件。
NodeLocal DNSCache	可选组件	通过在集群节点上作为 DaemonSet 运行 DNS 缓存代理来提高 DNS 解析性能。
CoreDNS	系统组件	Kubernetes集群域名解析服务器。
CCE LB Controller	系统组件	基于百度云应用型负载均衡产品（应用型BLB）实现Kubernetes的LoadBalancer Service的语义，提供四层（TCP、UDP）流量接入能力。
CCE Network Plugin	系统组件	CCE集群网络组件。

存储

组件名称	类型	组件介绍
CCE CSI BOS Plugin	可选组件	支持在Pod中以PV/PVC方式挂载百度云BOS存储。
CCE CSI CDS Plugin	可选组件	支持在Pod中以PV/PVC方式挂载百度云CDS存储，并支持动态创建PV。
CCE CSI PFS Plugin	可选组件	支持在Pod中以PV/PVC方式挂载百度云PFS存储，并支持动态创建PV。
CCE CSI PFS L2 Plugin	可选组件	支持在Pod中以PV/PVC方式挂载百度云L2类型PFS存储，并支持动态创建PV。

监控和日志

组件名称	类型	组件介绍
CCE Log Operator	可选组件	采集集群中的日志信息传输至日志服务（BLS）中进行持久化存储。
CCE Node Problem Detector	可选组件	实时检测节点上各种异常情况，并将检测结果上报至集群。
CCE Node Remedier	可选组件	节点自动维修系统，实现节点故障自愈。
CCE Onepilot	可选组件	用于接入多编程语言探针，在CCE集群中实现自动注入APM探针以及配置指标上报环境。

镜像

组件名称	类型	组件介绍
CCE Image Accelerate	可选组件	为Kubernetes集群提供镜像加速能力，配合使用容器镜像服务CCR的镜像加速功能，实现镜像数据免全量下载和在线解压，大幅减少容器启动时间。
CCE P2P Accelerate	可选组件	支持大规模集群并发拉取GB级容器镜像，大幅提升镜像拉取速度。
CCE Credential Controller	可选组件	支持在CCE集群中免密拉取CCR企业版实例的镜像。

调度

组件名称	类型	组件介绍
CCE Descheduler	可选组件	对集群中高负载节点上的 Pod 发起重调度。

云原生AI

组件名称	类型	组件介绍
CCE GPU Manager	可选组件	GPU架构感知等高级调度，支持GPU算力和显存的共享与隔离。
CCE Deep Learning Frameworks Operator	可选组件	提供分布式模型训练、部署等功能，支持TensorFlow、Pytorch、Mxnet、PaddlePaddle框架。
CCE RDMA Device Plugin	可选组件	在主机间通过高性能 RDMA 网络互联的场景下，提供容器间通过跨主机 RDMA 网络传输数据的能力。
CCE AI Job Scheduler	可选组件	适配AI训练场景，支持批量调度，队列管理等。
Fluid	可选组件	基于Kubernetes原生的开源分布式数据编排和加速引擎，主要用于云原生场景下的大数据、AI应用等。
CCE PaddleFlow Pipeline	可选组件	基于云原生架构的AI工作流组件，可实现编排AI作业，模板化AI作业的训练流程，提升实验效率。
CCE NPU Manager	可选组件	支持昆仑芯NPU算力和显存的共享与隔离。
CCE Ascend Mindx DL	可选组件	基于昇腾AI处理器的深度学习组件，提供昇腾AI处理器调度、集群性能测试等基础功能。

其他

组件名称	类型	组件介绍
CCE QoS Agent	可选组件	容器服务质量（QoS）增强组件，提供丰富的单机资源隔离能力，提升集群资源利用率，提供稳定性保障。
CCE CronHPA Controller	可选组件	支持集群工作负载定时扩缩容,可自主配置规则。
CCE BackUp Controller	可选组件	基于开源 Velero 实现的集群扩展组件，提供资源备份和恢复能力。

CCE GPU Manager 说明

组件介绍

一系列 GPU device plugin 的集合，结合配套的 scheduler 可以实现复杂场景下的 GPU 资源调度能力。CCE GPU Manager组件支持隔离最优型，可支持算力和显存的共享与隔离。

组件功能

- 拓扑分配：提供基于 GPU 拓扑分配功能，当用户分配超过1张 GPU 卡给 Pod 时，系统自动选择拓扑连接最快的方式分配 GPU 设备。
- GPU 共享：提供为节点上的 GPU 设备开启显存共享功能，支持将 GPU 卡按显存大小分配给多个 Pod。
- 显存和算力隔离：多 Pod 共享单张 GPU 卡时进行显存和算力级别的隔离。
- 精细化调度：开启精细化调度后，创建队列和任务时均支持选择具体的GPU型号。关闭精细化调度后创建队列和容器时仅支持输入配额，不支持选择具体的GPU型号。
- 编解码实例：提交编解码任务，使用GPU独立的编解码单元进行硬件编/解码。
- 组件详细使用说明请参照：GPU独占和共享说明

使用场景

在 CCE 集群中运行 GPU 应用时，可以解决 AI 训练等场景中独占整张卡造成资源浪费的情况，从而提高资源的使用率，降低成本。

限制说明

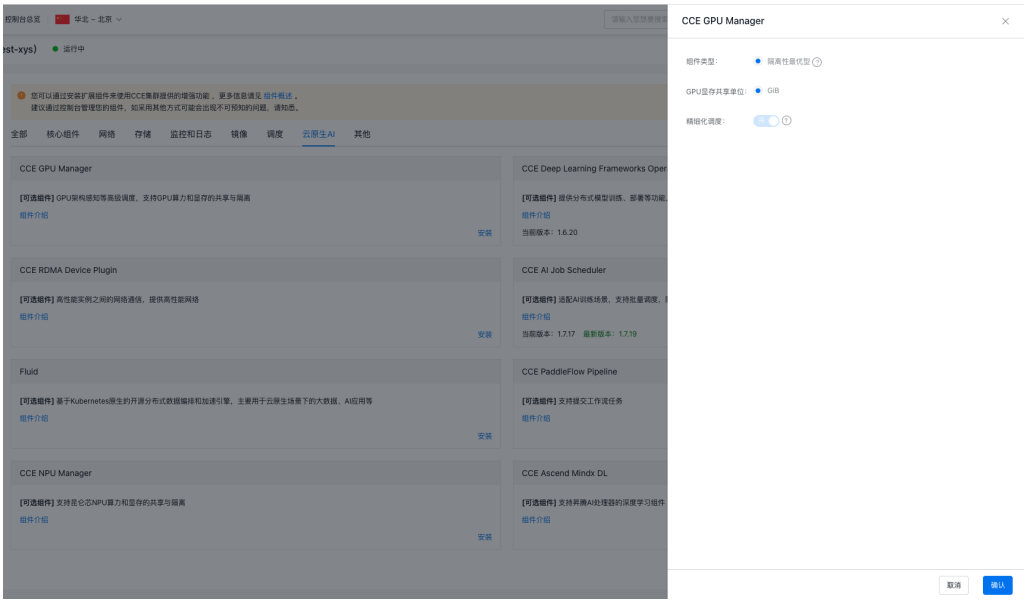
- 支持 v1.18 及以上版本的 Kubernetes 集群。
- 目前该组件依赖于 CCE AI Job Scheduler，若您需要请一同安装，否则可能导致组件功能不可用。
- GPU 共享虚拟化目前适配了以下主流的GPU CUDA和Driver版本，其中隔离最优型对操作系统内核版本等有额外的要求。如您有其他版本适配需求请提交工单。目前支持情况具体如下。

配置	版本
容器运行时	Docker、Containerd
GPU CUDA/Driver版本	• GPU Driver 470.X，515.X，525.X
操作系统内核版本（仅隔离最优型）	CentOS: • 3.10.0-957.21.3.el7.x86_64 • 3.10.0-1160.41.1.el7.x86_64 • 3.10.0-1160.42.2.el7.x86_64 • 3.10.0-1160.45.1.el7.x86_64 • 3.10.0-1160.62.1.el7.x86_64 • 3.10.0-1160.71.1.el7.x86_64 • 3.10.0-1160.76.1.el7.x86_64 • 3.10.0-1160.80.1.el7.x86_64 • 3.10.0-1160.81.1.el7.x86_64 • 3.10.0-1160.83.1.el7.x86_64 • 3.10.0-1160.88.1.el7.x86_64 • 3.10.0-1160.90.1.el7.x86_64 • 4.17.11-1.el7.elrepo.x86_64 • 5.4.123-1.el7.elrepo.x86_64 Ubuntu: • 4.4.0-150-generic • 4.15.0-140-generic • 5.4.0-72-generic • 5.4.0-139-generic

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **运维与管理 > 组件管理** 。

6. 在组件管理列表中选择 CCE GPU Manager 组件单击"安装"。
7. 在确认安装弹出框中默认选择隔离最优型。
8. GPU显存共享单位默认选择GiB。
9. 精细化调度默认开启。
10. 点击“确认”按钮完成组件的安装。



版本记录

版本号	适配集群版本	更新时间	变更内容	影响
1.5.35	CCE v1.18+	2024.07.05	新功能： • Pod申请虚拟化资源用法调整，支持仅申请虚拟化资源描述符，去除baidu.com/xx_xx_cgpu描述符约束 优化： • RDAM机型的BCC套餐适配，支持NCCL通信库自动感知单机GPU和网卡的拓扑关系 • 适配H芯片调度器分卡，获取设备信息的依赖由cuda改变为nvml 缺陷修复： • 【影响业务】修复PaddleJob的数据备份功能，修复备份概率失败问题	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.34	CCE v1.18+	2024.06.24	优化： • container-runtime 适配 VPC-ENI 网络模式下宿主机网络容器的 nccl 环境变量注入 缺陷修复： • 因影响通信性能 dcgm-exporter 默认不采集 FP16/FP32/FP64 指标；	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.33	CCE v1.18+	2024.05.31	新功能： • 新增多调度器GPU分卡信息识别服务，识别默认调度器等其他调度器分配的 GPU 卡号信息，避免节点被多种调度器混调； • IB环境默认开启自适应路由； 优化： • 虚拟化webhook支持高可用；	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.32	CCE	2024.0	新功能： • 支持集群内两种虚拟化模式共存; • 支持BCC RDMA拓扑文件的自动注入;	GPU内核态虚拟化业务不支持热升级，升级方式为

	v1.18+	5.15	优化： • GPU虚拟化的隔离性最优型中容器优化残留问题;	排空节点升级
1.5.31	CCE v1.18+	2024.05.06	新功能： • 新增GFD模块 • 支持GPU节点驱动&Cuda等环境信息上报至节点; • 隔离最优型GPU虚拟化支持L20芯片; • 支持 eks 模式的 hook 注入	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.30	CCE v1.18+	2024.03.26	新功能： • 适配bcc H800机型识别rdma拓扑 修复： • 一组镜像漏洞修复;	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.29	CCE v1.18+	2024.01.19	新功能： • dcgm-exporter 新增nvlink带宽、sm利用率、FP64/32/16计算利用率指标	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.28	CCE v1.18+	2023.12.15	新功能： • 支持NCCL环境变量适配不同的GPU卡类型，A100/A800使用NCCL_IB_QPS_PER_CONNECTION=8、NCCL_IB_ADAPTIVE_ROUTING=0，H800使用NCCL_IB_QPS_PER_CONNECTION=1、NCCL_IB_ADAPTIVE_ROUTING=1 优化： • dp健康检查端口设置为可修改的参数，且取消GPU虚拟化之隔离性最优型的dp健康检查	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.27	CCE v1.18+	2023.12.1	优化： • GPU虚拟化增加内核日志，打印GPU虚拟化显存OOM显存统计信息 • GPU虚拟化容器残留优化，提高GPU虚拟化的容器清理的效率，容器侧兼容容器创建/容器残留等多场景，内核模块增加GPU虚拟化清理 • GPU虚拟化的容器残留增加指标，体现GPU虚拟化残留	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.26	CCE v1.18+	2023.11.17	新功能： • 适配 Kubernetes 1.26 • 适配 Ubuntu 22.04 操作系统 优化： • 增加了 GPU虚拟化 支持内核版本的补全 • dp组件增加健康检查，兼容kubelet重启/访问apiserver失败等场景 缺陷修复： • 【影响业务】修复由于节点同时安装 docker 和 containerd 造成的 dcgm exporter 和 gpu exporter 无容器信息上报问题 • 【影响业务】修复由于 systemd 路径解析错误造成的 container-runtime 分配 GPU 卡和 RDMA 配置无效的错误 • 【影响业务】修复由于 systemd 路径解析错误造成的 gpu exporter 无法获取容器信息的错误	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
			新功能： • 支持GPU 虚拟化内核态 535 驱动	

1.5.25	CCE v1.18+	2023.11.03	• 支持4090芯片的独占卡/共享卡模式 优化： • GPU虚拟化之隔离性最优型 初始化优化: 增加对节点的 sgpu.ko 内核前置检查：增加残留模块版本校验以及残留无效模块删除重装 • 增加了 GPU虚拟化 容器的 GC 模块，实现了残留的残留的 GPU虚拟化 配置清理 • 优化 container-runtime-sgpu-hook prestart/poststop 的异常流程，修改流程为配置失败时，返回错误信息 缺陷修复： • 【影响业务】修复由于 container-runtime 没有区分 container 获取 pod 的 resource 导致单 Pod 多容器分配错卡的问题 • 【影响业务】修复由于组件升级过程中，install.sh 进程退到了内核态导致的 DCGM pod 一直处于 Terminating 状态，不能删除 • 【影响业务】修复由于 runtime 默认 lib-path 在操作系统Ubuntu 20中不生效造成的任务启动报错 • 【影响业务】修复 CUDA Driver 525 驱动环境下，性能最优型GPU虚拟化 共享卡的 libcuda.so 劫持无效的问题 限制： • 不支持创建使用 性能最优型GPU虚拟化 共享卡以 NCCL 进行通信的 DDP 训练任务	GPU内核态虚拟化业务不支持热升级，升级方式为排空节点升级
1.5.24	CCE v1.18+	2023.09.22	新功能： • 支持单容器使用多共享卡功能 缺陷修复： • 解决虚拟化场景监控指标获取失败问题 使用限制： • 当前版本为1.5.14及以下时，若开启性能最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件 • 当前版本为1.5.13及以下时，若开启隔离最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件	
1.5.23	CCE v1.18+	2023.08.29	优化： • 默认配置NCCL_DEBUG日志必要的子系统，NCCL_DEBUG_SUBSYS由ENV改为INIT,ENV,GRAPH 使用限制： • 当前版本为1.5.14及以下时，若开启性能最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件 • 当前版本为1.5.13及以下时，若开启隔离最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件	
1.5.22	CCE v1.18+	2023.08.10	缺陷修复： • 修正并发创建虚拟化容器时，偶发虚拟化显存和算力资源的分配错误问题 使用限制： • 当前版本为1.5.14及以下时，若开启性能最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件 • 当前版本为1.5.13及以下时，若开启隔离最优型GPU虚拟化能力，需要停止虚拟化任务，升级组件	

CCE Deep Learning Frameworks Operator 说明

组件介绍

主流深度学习框架 operator 组件，结合 CCE AI Job Scheduler，可实现直接在 CCE 上进行深度学习模型训练。

组件功能

集成主流深度学习框架，提供开箱即用的深度学习任务提交能力，目前支持以下深度学习框架：

1、TensorFlow（TFJob）

2、PyTorch（PyTorchJob）

3、MXNet（MXJob）

4、PaddlePaddle（PaddleJob）

使用场景

您可以直接在 CCE 集群上运行深度学习任务，提高 AI 工程效率。

限制说明

- 仅支持 v1.18 及以上版本的 Kubernetes 集群。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **运维与管理 > 组件管理** 。
6. 在组件管理列表中选择 CCE Deep Learning Frameworks Operator 组件单击“安装。

- 框架：目前支持 TensorFlow、Pytorch、Mxnet、PaddlePaddle 四种深度学习框架。

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.6.23	CCE/v1.18+	2024.09.11	**新功能**： 支持训练hang场景检测&告警 mpijob自动注入ssh免密登陆配置 **优化**： pytorchjob任务timeline字段优化 任务所在节点发生nic up/down故障场景，容错能力优化	此次升级不会对业务造成影响
1.6.22	CCE/v1.18+	2024.08.28	**新功能**： 支持 停止处于排队中/启动中状态的任务 **优化**： 支持展示更精确的任务状态 优化 MPIJob laucher 在 worker 未启动会重试几次的问题	此次升级不会对业务造成影响
1.6.21	CCE/v1.18+	2024.07.22	MPIJob支持通过任务标签配置RDMA亲和性策略;ftagent-exporter性能指标适配P800集群;job barrier优化,master和woker优化退出超时机制;增加容错触发后pod长时间处于terminating可强制删除开关;容错内部与k8s交互逻辑改为informer	
1.6.20	CCE/v1.18+	2024.05.29	修复 ftagent 会占用 8080 端口的问题	
			training-operator 默认开启 job-barrier，job-barrier 支持容错抢占；training-operator 支持异步清理终态	

1.6.19	CCE/v1.18+	2024.05.29	training-operator 默认开启 job-barrier; job-barrier 支持容错抢占；training-operator 支持异步清理终态任务的 service; training-operator 支持打上 stop condition && stop init container; 修复特殊情况下会重复创删 pod service	
1.6.18	CCE/v1.18+	2024.05.17	ftagent-exporter 适配AIAK2.0镜像&支持Loss指标&回溯时间更新为60s&针容错针对好未来nic up down 场景优化	
1.6.17	CCE/v1.18+	2024.04.12	ftagent-exporter 指标新增 pod_name, job_name 标签	
1.6.16	CCE/v1.18+	2024.03.11	支持训练任务性能指标采集和上报：吞吐性能指标、分阶段耗时指标	
1.6.15	CCE/v1.18+	2024.02.26	任务被抢占&触发容错重调度，pytorchjob置为Restarting状态，修复pytorchjob没有Created状态bug，修复pod failed pytorchjob为running bug；新增ftagent exporter，支持集合通信带宽指标暴露；	
1.6.14	CCE/v1.18+	2024.02.06	新增任务事件timeline&容错事件&tensorboard gc	
1.6.13	CCE/v1.18+	2024.01.17	新增master/worker node not ready 场景容错优化	
1.6.12	CCE/v1.18+	2023.12.18	新增 Training-Operator、Mpi-Operator 支持优先级抢占; bugfix: Mpi-Operator 修复同名任务频繁创删场景 创建 pod 卡住问题	
1.6.11	CCE/v1.18+	2023.12.04	新增自动容错覆盖节点not ready的场景;	
1.6.10	CCE/v1.18+	2023.11.22	新增容错支持 master-worker 模式	
1.6.9	CCE/v1.18+	2023.11.03	新增hang 检测解耦etcd；新增非法任务名校验；新增任务名长度校验不超过50；bugfix: ft-agent 支持重启策略为OnFailure Pod；修复c10d 作业概率性创建失败问题；新增容错开关；任务hang 支持容错；1.6.9 版本不部署etcd；新版本容错对接控制台；training-operator 默认关闭job barrier 功能；	
1.6.8	CCE/v1.18+	2023.10.10	容错功能重构，解决任务失败资源无法释放的问题	
1.6.6	CCE/v1.18+	2023.08.25	PytorchJob支持hang检测；MPIJob支持hostfile注入worker节点和任务停止；修复training-operator快速删除、创建任务时Pod无法被创建；PytorchJob initContainer添加最大重试次数；Training-Operator 任务失败 ftagent 退出时清理训练进程	
1.6.5	CCE/v1.18+	2023.07.07	升级 Pytorchjob支持宕机容错	
1.6.4	CCE/v1.18+	2023.07.05	MPI Operator 指定 gang-scheduling，改为由 operator 维护 pg	
1.6.3	CCE/v1.18+	2023.06.27	新增 MPI Operator 与 Paddle Operator；支持作业停止；TrainingOperator可将作业状态通过exporter对外暴露	
1.6.1	CCE/v1.18+	2023.05.30	升级 Pytorchjob支持硬件故障（GPU、NIC）容错	
0.3.0	CCE/v1.18+	2022.05.12	升级为 TrainingOperator 将 PyTorch/Tensorflow/MXNet Operator 合并	不支持一件升级，需先卸载旧版插件重新安装
0.2.1	CCE/v1.18+	2022.03.02	新增：AiTrainingJob Webhook	
0.2.0	CCE/v1.18+	2022.01.21	新增：AI Training Operator	
0.1.0	CCE/v1.18+	2021.05.31	首次上线	-

CCE RDMA Device Plugin 说明

组件介绍

RDMA 通信设备组件。

组件功能

在主机间通过高性能 RDMA 网络互联的场景下，提供容器间通过跨主机 RDMA 网络传输数据的能力。

使用场景

跨主机高性能数据传输，GPU Direct RDMA 等。

限制说明

- 支持 v1.18 版本以上的 Kubernetes 集群。
- 容器网络场景下使用RDMA通信的Worker节点OS内核版本 >= 4.9。
- 集群网络模式为 VPC 网络模式。关于 VPC 网络模式，请参考 这里。
- 集群 CNI 组件版本 >= 1.6.9。
  - 创建时间为 2022年2月28日 之后的集群，默认 CNI 插件版本 >= 1.6.9。
  - 如需要验证存量集群的 CNI 版本信息，可以通过如下指令获取。如集群 CNI 版本不满足要求，请提交工单申请升级。

```
kubectl -n kube-system get ds cce-cni-node-agent -o yaml | grep "image:"
```

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **组件管理** 。
6. 在组件管理列表中选择 CCE RDMA Device Plugin 组件单击“安装。
7. 在确认安装弹出框中点击“确定”按钮完成组件的安装。

版本记录

版本号	适配集群版本	变更时间	变更内容	影响
0.1.9	CCE v1.18+	2023.11.17	• rdma支持kubelet多路径配置	此次升级不会对业务造成影响。
0.1.8	CCE v1.18+	2023.06.21	**优化**： • 支持nccl环境变量配置文件注入 • rdma device plugins容忍污点	此次升级不会对业务造成影响。

CCE AI Job Scheduler 说明

组件介绍

任务调度组件，支持调度管理各种AI任务，结合 CCE Deep Leaning Frameworks Operator，可实现直接在 CCE 上进行深度学习模型训练。

组件功能

- 支持丰富的调度策略和增强型的 Job 管理能力。
- 调度策略支持 spread和binpack两种策略。binpack 表示多个 Pod 会优先集中共享使用同一 GPU 卡，适用于需要提高 GPU 资源利用率的场景，spread 表示多个 Pod 会尽量分散使用不同的 GPU 卡，使用于 GPU 高可用场景。
- 抢占模式支持队列内优先级抢占和队列间超发抢占。队列内优先级抢占指同一队列中，优先级高的任务可抢占优先级低任务的资源，保障高优先级任务的运行；队列间超发抢占是指A队列资源用满B队列有空闲资源时，此时若A队列上提交了新任务，将调度到B队列上运行，当B队列上有新任务提交发现资源不足时，将Kill超发任务保障B队列任务运行。
  抢占功能使用可参考队列管理和任务管理中相关说明。

使用场景

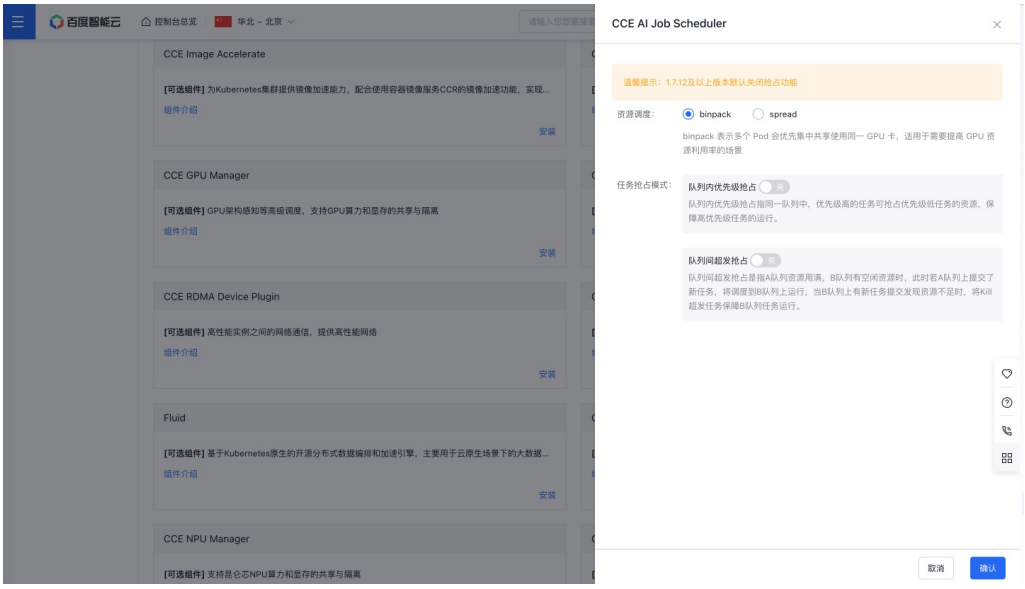
您可以直接在 CCE 集群上运行深度学习任务，提高 AI 工程效率。

限制说明

- 仅支持v1.18及以上版本的 Kubernetes 集群。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **组件管理** 。
6. 在组件管理列表中选择 CCE AI Job Scheduler 组件单击“安装。
7. 在组件配置页面中完成深度学习框架配置。



- 调度策略支持 spread 和 binpack 两种，binpack 表示多个 Pod 会优先集中共享使用同一 GPU 卡，适用于需要提高 GPU 资源利用率的场景，spread 表示多个 Pod 会尽量分散使用不同的 GPU 卡，使用于 GPU 高可用场景。
- 抢占模式支持队列内优先级抢占和队列间超发抢占。队列内优先级抢占指同一队列中，优先级高的任务可抢占优先级低任务的资源，保障高优先级任务的运行；队列间超发抢占是指A队列资源用满B队列有空闲资源时，此时若A队列上提交了新任务，将调度到B队列上运行，当B队列上有新任务提交发现资源不足时，将Kill超发任务保障B队列任务运行。

8. 点击“确认”按钮完成组件的安装。

版本记录

版本号	适配集群版本	变更时间	变更内容	影响
1.7.25	CCE v1.18+	2024.11.07	**新功能**： • 控制面模块支持指定节点部署，webhook类组件支持宿主机网络部署，增加对污点的容忍 • tor元信息同步优化，使用volcano-node-spec配置套餐rdma信息时，支持配置ehc字段 **优化**： • 整机申请资源比较优化，整机场景使用标量比较，减少数据拷贝，提升性能 • 增加myriator插件，支持大模型任务在一个tor下按照index排序调度，优化热点函数提升调度性能 **缺陷修复**： • 修复并发访问map导致的crash（绑定阶段写map与抢占阶段读map冲突）	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.24	CCE v1.18+	2024.09.30	**新功能**： • 队列支持配置调度策略，支持StrictFIFO的调度策略	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。

				v1.7.13以下版本请联系百度云协助升级。
1.7.23	CCE v1.18+	2024.09.27	**新功能**： • 支持队列独立配置优先级抢占开关，精细化控制优先级抢占能力 • 增加调度阶段可观测指标，支持调度阶段耗时可视化 • 开启队列内优先级抢占场景下，任务配额申请阶段考虑队列内不可抢占的资源 • NPU拓扑感知调度策略性能优化 **缺陷修复**： • 【非影响业务】修复偶发调度缓存并发访问导致的panic	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.22	CCE v1.18+	2024.09.03	**新功能**： • RDMA TOR拓扑感知调度适配EHC Cluster • 支持NPU和GPU统一调度器 **优化**： • 支持同队列任务抢占超发(lowest)任务 • 支持超发(lowest)任务延迟调度，优先调度高中低优先级的普通任务	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.21	CCE v1.18+	2024.08.14	**优化**： • 优化安装流程中的证书创建逻辑，解决集群无可用节点导致组件安装超时问题。 • RDMA信息同步组件适配 BCC/HPAS，支持通过外部配置指定RDMA 信息 • NPU插件支持抢占功能，支持 NPU 队列内/队列间抢占场景（for NPU） **缺陷修复**： • 【非影响业务】修复 Job内存在多种PodSpec时偶发无法调度的问题	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.20	CCE v1.18+	2024.07.22	**新功能**： • 支持NPU芯片资源视图大盘 **缺陷修复**： • 【非影响业务】同一任务存在多类不同的Pod配置，有概率部分pod调度失败影响其他pod的调度 • 【非影响业务】处理存量队列和 root 重名的情况，导致根队列更新失败 • 【非影响业务】volcano controller部分功能初始化失败导致队列信息不更新	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.19	CCE v1.18+	2024.07.05	**新功能**： • 支持集群配置申请GPU资源的 Pod统一至volcano调度器 **优化**： • 优化抢占场景下 RDMA亲和性入队检查策略，开启抢占则关闭 HPN 检查 • 优化单任务申请rdma资源的优选策略，尽量使binpack效果更明显 **缺陷修复**： • 解决 RDMA 资源视图不兼容Terminating释放中的资源，引发调度器panic的问题 • 修复了任务不指定队列时不设置默认队列的问题	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
			新功能： • 队列指标支持P800芯片，新增P800资源视图大盘； • 资源视图命令行工具适配 P800芯片，支持P800集群的任务诊断；	

1.7.18	CCE v1.18+	2024.06.26	• 物理队列支持自定义资源管理节点标签，兼容用户已有资源管理标签的场景； • RDMA亲和性调度策略支持扩展自定义资源描述符，例如baidu/gpu_hzz1o_8； **优化**： • MPIJob场景RDMA TOR策略优化：仅申请CPU的Pod去除一个Job内分布同一RDMA POD下的分布约束 • IB场景适配：针对无法获取 RDMA TOR 信息的IB实例，毋须关闭TOR亲和性调度策略 **缺陷修复**： • 修复推理服务不受物理队列管控的问题，支持了多种工作负载对物理队列的适配 • 修复了由于pod/node亲和性权重较低导致的反亲和部署策略效果不明显问题 • 解决volcano 视图工具 dump视图计算异常	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.17	CCE v1.18+	2024.06.02	**新功能**： • 新增队列资源视图大盘，队列指标丰富，支持弹性/层级队列功能，支持nvidia/昆仑等多种芯片； **优化**： • 集群多调度器混用场景加固，支持识别集群已有其他调度器分配的GPU卡，避免单节点混用多种调度器分配GPU； • 增加了单队列 Capability，Deserved 和 Guarantee 间的资源申请合法校验，避免创建无效队列；	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.16	CCE v1.18+	2024.05.23	**新功能**： • 增加GPU资源调度器强制拦截开关功能。 **优化**： • 修复队列无法忽略rdma资源问题 • 修复注入节点亲和性调度失效问题	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.15	CCE v1.18+	2024.05.17	**新功能**： • 支持昆仑新芯片以及拓扑感知调度功能。 **优化**： • 层级队列调度失败信息优化，非叶子队列调度失败时暴露事件	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.14	CCE v1.18+	2024.05.09	**新功能**： • 发布弹性队列能力，支持队列资源的预留、共享与回收。 • 发布物理队列能力，支持队列任务定向调度到指定资源池。 • 支持通过任务/服务标签配置工作负载所需的最小保障副本数。 **缺陷修复**： • 修复节点资源outOfSync不一致后，节点状态恢复但资源视图无法自恢复问题 • 优化抢占策略：抢占者任务预期抢占受害者后仍不满足调度时，不触发抢占	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该版本。 v1.7.13以下版本请联系百度云协助升级。
1.7.13	CCE v1.18+	2024.04.15	**新功能**： • 发布层级队列能力，支持层级的队列配额管理能力 **优化**： • 开启队列内抢占功能时，队列入队增加可抢占资源计算，预期抢占后满足调度条件则允许入队； • RDMA拓扑感知策略增加PodGroup事件	此次升级不会对业务造成影响。 不支持 1.5.8 以下版本升级至该

			缺陷修复： • 修复抢占场景下资源视图计算错误导致的调度器重启问题	版本。
1.7.12	CCE v1.18+	2024.03.28	**新功能** • RDMA亲和性策略，支持基于RDMA POD/TOR拓扑结构调度，提高多机训练性能 **优化** • 默认部署策略优化 a. 默认关闭在离线混部功能 b. 默认关闭队列内/队列间抢占功能 c. 默认关闭VPC TOR亲和性调度功能 d. 支持SLA 策略开关，支持特定客户的使用场景 **缺陷修复**： • 修复昆仑卡拓扑感知调度无法分配昆仑卡号的问题 • 修复推理服务申请配额计算错误导致的无法入队问题 • 修复webhook/controller中存在的并发访问内存导致的crash问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.11	CCE v1.18+	2024.01.31	**优化**： • 资源视图优化，工作负载指标增加pod_group_uid标签，节点资源指标增加节点类型标签 • 视图工具支持用户自定义的volcano命名空间 • 调度器内部分卡协议优化，避免分卡信息写入apiserver失败导致的错卡问题 **缺陷修复**： • 修复存在待释放资源节点(Terminating Pod占用)和空闲节点多个符合调度所需的场景下，调度器依然选择调度至待释放资源节点的问题 • 修复任务annotation不存在导致controller重启的问题 • 修复并发访问map未加锁导致调度器重启的问题 • 修复上报队列监控指标异常处理导致调度器重启的问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.10	CCE v1.18+	2023.12.21	**优化**： • 优先级调度策略支持跨namespace **缺陷修复**： • 修复tor调度在选不到tor且调度失败的情况下导致的panic • 修复device-affinity插件导致的panic，并且给device-affinity策略提供开关 • 修复volcano的webhook增加可以忽略带有kubernetes.io/mutate-pod-webhook: unavailable标签的namespace，并且在安装的时候，默认给kube-system和volcano-system增加这个label • 修复pod的owner reference为pod的管理方式	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.9	CCE v1.18+	2023.11.28	**新功能**： • 资源视图支持资源统计大盘和节点资源大盘 • 资源视图支持工作负载详情大盘 • volcano稳定性大盘指标 **优化**： • 支持任务通过preemptable标签指定任务不可被抢占 **缺陷修复**： - 修复调度器重启后，因视图同步延迟造成的视图错误 • 修复volcano增加对nvidia.com/gpu资源的binpack策略	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。

			• 修复抢占时需要保证卡类型相同，否则不发生抢占 • 修复tor策略的空指针异常 • 修复并发访问的device对应导致的panic • 修复collector采集指标时的并发访问导致的panic	
1.7.8	CCE v1.18+	2023.10.30	**新功能**： • 支持k8s标准工作负载Pod/Job/Deployement/Statefulset的PodGroup生命周期管理 • 新增命令行工具，支持集群节点/队列的资源视图查看，自主排查任务无法调度情况 **优化**： • 支持MPIJob查看被抢占事件。 **缺陷修复**： • 解决未支持的工作负载的队列/集群配额残留问题； • 解决队列未忽略弹性任务中申请的RDMA资源，造成队列配额超用误判的问题 • 解决GPU共享卡场景下，计算资源视图指标逻辑有误，引发调度器panic的问题 • 解决1.7.3以下版本升级时滚动策略机制不合理，有概率导致webhook证书不一致问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.7	CCE v1.18+	2023.10.11	**新功能**： • 新增昆仑r480型号的numa调度（依赖GPU-Manager版本 1.5.25） • 支持H800芯片的独占卡模式（依赖GPU-Manager版本 1.5.25） • 支持4090芯片的独占卡/共享卡模式（依赖GPU-Manager版本 1.5.25） • 资源视图支持(grafana监控大盘，展示集群资源总览与节点资源明细，和百舸页面保持一致) **优化**： • 支持Deployment的podgroup管理 • 命令行工具增加选项，支持根据job类型和podgroup状态筛选job列表，支持summary选项对选中的job列表占用资源量进行求和 • 命令行补充totalgpu字段，在nvidia和cgpu描述符混合使用的时候，统计实际的gpu卡数 **缺陷修复**： • 修复GPU选卡阶段选择到Terminating阶段的Pod • 修复notready节点，grafana监控无法显示 • 修复predicate阶段terminating导致的调度卡住问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.6	CCE v1.18+	2023.09.22	**新功能**： • 新增集群资源视图/调度问题诊断工具 • 支持单容器多共享卡 • TOR架构感知能力新增支持MPIJob类型任务，兼容Training-operator 1.5+/百度Deep Learning Framework组件1.6+。 **优化**： • 日志优化：支持日志等级动态调整；调整至json格式 **缺陷修复**： • 修复队列弹性资源计算错误引发调度器panic问题。针对1.7.x以上podgroup支持minResources功能，podgroup部分pod运行且不包含minResource中的全部资源，则会引发调度器panic。https://github.com/volcano-sh/volcano/issues/3105 • 修复Pod调度过程中，设备亲和性策略计算后，Pod备选节点为空，引发调度器panic问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。

			• 修复controller权限不足导致的job对应的podgroup标签获取失败问题	
1.7.4	CCE v1.18+	2023.06.14	**新功能**： • 支持volcano scheduler/admission/controller的高可用，默认3副本模式 **优化**： • 队列支持用量统计 • 优化admission的签发证书流程，使用secret保存访问证书 • scheduler/admission/controller增加资源配置参数 **缺陷修复**： • 修复并发读写node资源导致调度器panic问题	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.3	CCE v1.18+	2023.05.06	**新功能**： • 支持自定义抢占策略	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.2	CCE v1.18+	2023.04.24	**新功能**： • 支持a800芯片的独占卡/共享卡模式 • 支持自定义调度器名称及调度资源组	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。
1.7.0	CCE v1.18+	2023.04.14	**新功能**： • volcano升级至社区1.7版本	此次升级不会对业务造成影响。不支持 1.5.8 以下版本升级至该版本。

CCE 动态调度插件说明

组件介绍

CCE Dynamic Scheduler 是基于 Kubernetes 原生 Kube-scheduler Extender 机制实现的动态调度器插件，可基于节点 CPU/Memory 真实利用率进行 Pod 动态调度。在容器服务 CCE 集群中安装插件后，该插件利用 scheduler extender 机制向 Kube-Scheduler 注册 Filter、Prioritize 钩子，来干预默认调度器的调度行为，有效避免原生调度器基于 Request 和 Limit 调度机制带来的节点负载不均问题。该组件需要借助于 Prometheus 监控组件及相应的预聚合规则设置才能正常运行。因此，您可以参照本文的 `依赖部署` 部分进行操作，以免遇到插件不能正常工作的问题。

功能介绍

Kubernetes 自带的调度器 kube-scheduler 的作用是将新出现的 Pod 绑定到某一个最佳的节点。为了实现这一功能，调度器会需要进行一系列的筛选和打分（预选和优选策略），进行一次性的调度。然而，这种调度存在局限：不具备根据 Node 当前和过去一段时间的真实负载情况进行相关调度的决策，因此可能出现调度不合理的情况。 社区版本调度器中提供的节点利用率相关的策略（如 LowNodeUtilization、HighNodeUtilization）是利用 Pod 的 Reqeust 和 Limit 数据，而不是节点的真实利用率，如：集群内部分节点的剩余可调度资源较多（根据节点上运行的 Pod 的 Reqeust 和 Limit 计算出的值），但真实负载却比较高，而另外节点的剩余可调度资源比较少但真实负载却比较低，此时 Kube-scheduler 会优先将 Pod 调度到剩余资源比较多的节点上（根据 LeastRequestedPriority 策略）。

CCE Dynamic Scheduler 新增了 “基于节点真实负载” 的动态调度策略，用户在集群中安装该组件后，组件会根据从 Prometheus 采集的节点指标和用户设置的节点负载阈值对 Pod 实现动态调度。

使用场景

集群资源动态变化：

- 集群节点资源利用率时刻发生改变，需要 CCE Dynamic Scheduler 进行动态调度

限制说明

- 集群版本在 1.18.9 及以上, 仅支持独立集群
- 集群已经通过 Prometheus 实现指标采集
- 如果需要升级 Kubernetes master 版本，master 版本升级会重置 master 上所有组件的配置，从而影响到 Dynamic Scheduler 插件作为 Scheduler Extender 的配置，因此 Dynamic Scheduler 插件需要卸载后再重新安装

安装组件

依赖部署 CCE Dynamic Scheduler 组件依赖于 Node 当前和过去一段时间的真实负载情况来进行动态调度，需通过 Prometheus 等监控组件获取系统 Node 真实负载信息。在使用 CCE Dynamic Scheduler 组件之前，您需要完成 Prometheus 监控配置。 **配置 Prometheus 采集规则**

1. 使用自建 Prometheus 作为监控数据源的情况下，用户需要自行部署两个组件，可参考下面组件官方文档完成部署：

- NodeExporter
- Prometheus

2. 在完成组件部署后，用户需要在 Prometheus 添加对 cAdvisor 和 NodeExporter 的监控指标采任务以及指标聚合规则的配置，具体配置可参考：

```
crape_configs:
- job_name: "kubernetes-cadvisor"
  # Default to scraping over https. If required, just disable this or change to
  # `http`.
  scheme: https

  # Starting Kubernetes 1.7.3 the cAdvisor metrics are under /metrics/cadvisor.
  # Kubernetes CIS Benchmark recommends against enabling the insecure HTTP
  # servers of Kubernetes, therefore the cAdvisor metrics on the secure handler
  # are used.
  metrics_path: /metrics/cadvisor

  # This TLS & authorization config is used to connect to the actual scrape
  # endpoints for cluster components. This is separate to discovery auth
  # configuration because discovery & scraping are two separate concerns in
  # Prometheus. The discovery auth config is automatic if Prometheus runs inside
  # the cluster. Otherwise, more config options have to be provided within the
  # <kubernetes_sd_config>.
  tls_config:
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    # If your node certificates are self-signed or use a different CA to the
    # master CA, then disable certificate verification below. Note that
    # certificate verification is an integral part of a secure infrastructure
    # so this should only be disabled in a controlled environment. You can
    # disable certificate verification by uncommenting the line below.
    #
    insecure_skip_verify: true
  authorization:
    credentials_file: /var/run/secrets/kubernetes.io/serviceaccount/token

  kubernetes_sd_configs:
    - role: node

  relabel_configs:
    - action: labelmap
      regex: __meta_kubernetes_node_label_(.+)

- job_name: 'node-exporter'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
    - source_labels: [__meta_kubernetes_pod_name]
      regex: 'node-exporter-(.+)'
      action: keep
```

3. 在使用自建 Prometheus 的情况下，配置指标聚合规则和使用托管集群类似，区别在于规则中不需要考虑 clusterID 维度，该维度由托管 Prometheus 添加。因此，聚合规则可参考：

> 该规则实现了对 CCE Dynamic Scheduler 依赖的 machine_cpu_usage_5m和machine_memory_usage_5m等指标的自动汇聚计算。

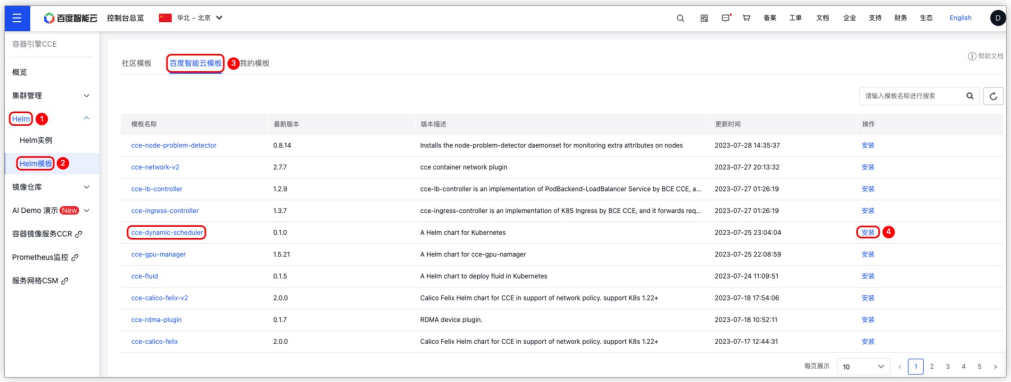
```
spec:
groups:
  - name: machine_cpu_mem_usage_active
    interval: 30s
    rules:
      - record: machine_memory_usage_active
        expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: machine_memory_usage_1m
    interval: 1m
    rules:
      - record: machine_memory_usage_5m
        expr: 'avg_over_time(machine_memory_usage_active[5m])'
  - name: machine_cpu_usage_1m
    interval: 1m
    rules:
      - record: machine_cpu_usage_5m
        expr: >-
          100 - (avg by (instance)
          (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

Helm 安装 CCE Dynamic Scheduler

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 Helm > Helm 模版。
4. 在 Helm 模版 页面中，单击 **百度智能云模版** 进入模版管理页面。
5. 在模版管理页面中选择模版名称 cce-dynamic-scheduler，然后单击 安装。
6. 在 安装模版 页面中完成相关配置填写，点击 确定 按钮完成组件的安装。
7. 登陆 Master Node，配置 Kube-Scheduler 启动项，使组件生效。

操作步骤

1. 用户需单击左侧导航栏中的 Helm > Helm 模版， 在 Helm 模版 页面中，单击 **百度智能云模版** 进入模版管理页面，找到组件 cce-dynamic-scheduler，然后单击 安装，流程如下图：



2. 在安装模版界面中，需填写【实例名称】、【部署集群】、【命名空间】、【Prometheus 地址】和【cpu/memory 阈值】，单机 确认 完成

组件部署。

3. 登陆 Master Node，配置 Kube-Scheduler 启动项

- 输入命令 `cd /etc/kubernetes/ && vim scheduler-extender.yaml` ，创建 `scheduler-extender.yaml` ，`scheduler-extender.yaml`内容如下：

```
apiVersion: kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfiguration
clientConnection:
  kubeconfig: "/etc/kubernetes/scheduler.conf"
extenders:
- urlPrefix: "http://dynamic-scheduler.svc/dynamic/extender" ## 实际 dynamic-scheduler的 svc 地址
  filterVerb: "filter"
  prioritizeVerb: "prioritize"
  weight: 1
  enableHTTPS: false
  nodeCacheCapable: true
  ignorable: true
```

- 输入命令 `cd manifests/ && vim kube-scheduler.yaml` ，修改 `Kube-Scheduler.yaml` 的配置文件，添加启动参数 `- --config=/etc/kubernetes/scheduler-extender.yaml` ，`Kube-Scheduler.yaml` 内容如下：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-scheduler
    tier: control-plane
  name: kube-scheduler
  namespace: kube-system
spec:
  containers:
    - command:
        ## ...
        - --config=/etc/kubernetes/scheduler-extender.yaml ## 修改kube-scheduler的配置文件，添加启动参数
```

使用案例

1. 热点规避的真实负载调度策略：

> 描述：当节点超过安全水位时，新部署的pod不会调度到超过阈值的 node 节点

1. 观察工作负载所在节点的负载情况

```
→  case01 kubectl top nodes
NAME            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
172.16.80.31    3083m        79%    2786Mi          17%
172.16.80.32    153m         3%     1709Mi          10%
172.16.80.33    73m          1%     2653Mi          16%
172.16.80.34    107m         2%     3863Mi          24%
172.16.80.35    150m         3%     1768Mi          11%
172.16.80.36    164m         4%     1698Mi          10%
```

2. 部署其他服务进行调度

3. 观察服务调度部署情况，发现部署的服务不会调度到 `node-172.16.80.31` 上

```
→  case01 kubectl get pods -owide
NAME                          READY   STATUS    RESTARTS   AGE     IP          NODE           NOMINATED NODE   READINESS GATES
busybox-56d9485878-2jqwx      1/1     Running   0          94s     10.4.3.82   172.16.80.33   <none>           <none>
busybox-56d9485878-4lqbk      1/1     Running   0          94s     10.4.3.95   172.16.80.33   <none>           <none>
busybox-56d9485878-5jrjp      1/1     Running   0          94s     10.4.3.122  172.16.80.33   <none>           <none>
busybox-56d9485878-5z744      1/1     Running   0          94s     10.4.3.231  172.16.80.33   <none>           <none>
busybox-56d9485878-6zf48      1/1     Running   0          94s     10.4.3.199  172.16.80.33   <none>           <none>
busybox-56d9485878-785g8      1/1     Running   0          94s     10.4.3.139  172.16.80.33   <none>           <none>
busybox-56d9485878-7vdgn      1/1     Running   0          94s     10.4.3.120  172.16.80.33   <none>           <none>
busybox-56d9485878-b8t8x      1/1     Running   0          94s     10.4.3.207  172.16.80.33   <none>           <none>
busybox-56d9485878-bqpv9      1/1     Running   0          94s     10.4.3.220  172.16.80.33   <none>           <none>
busybox-56d9485878-g7bhz      1/1     Running   0          94s     10.4.3.180  172.16.80.33   <none>           <none>
busybox-56d9485878-knvhc      1/1     Running   0          94s     10.4.3.32   172.16.80.33   <none>           <none>
busybox-56d9485878-kq7fr      1/1     Running   0          94s     10.4.3.242  172.16.80.33   <none>           <none>
busybox-56d9485878-nf2vl      1/1     Running   0          94s     10.4.3.71   172.16.80.33   <none>           <none>
busybox-56d9485878-qq9n4      1/1     Running   0          94s     10.4.3.169  172.16.80.33   <none>           <none>
busybox-56d9485878-t7t6l      1/1     Running   0          94s     10.4.3.151  172.16.80.33   <none>           <none>
busybox-56d9485878-tdvzb      1/1     Running   0          94s     10.4.3.13   172.16.80.33   <none>           <none>
busybox-56d9485878-vmsl6      1/1     Running   0          94s     10.4.3.225  172.16.80.33   <none>           <none>
busybox-56d9485878-vr42m      1/1     Running   0          94s     10.4.3.2    172.16.80.33   <none>           <none>
busybox-56d9485878-wtrwp      1/1     Running   0          94s     10.4.3.140  172.16.80.33   <none>           <none>
busybox-56d9485878-zdsds      1/1     Running   0          94s     10.4.3.174  172.16.80.33   <none>           <none>
cpu-stress-75dddc8656-xdh7d   1/1     Running   0          7h14m   10.4.5.93   172.16.80.31   <none>           <none>
```

4. 查看组件日志

```
I0726 12:48:53.393859       1 handler.go:22] scheduler call filter
I0726 12:48:53.394039       1 handler.go:29] scheduler call filter for default/busybox-56d9485878-kq7fr
I0726 12:48:53.394054       1 info.go:32] strategyData is Node 172.16.80.34 cpu usage 2.838333 mem usage 14.656576
I0726 12:48:53.394071       1 info.go:32] strategyData is Node 172.16.80.31 cpu usage 77.070000 mem usage 8.832973
I0726 12:48:53.394086       1 info.go:32] strategyData is Node 172.16.80.33 cpu usage 1.713333 mem usage 8.677264
I0726 12:48:53.394092       1 info.go:36] strategyDataList total count is 3
I0726 12:48:53.394102       1 extender.go:112] feasible nodes body: ["172.16.80.34","172.16.80.33"]
I0726 12:48:53.394110       1 extender.go:115] failed nodes body: {"172.16.80.31":"Real cpu ratio 77.070000 great than 50.000000"}
I0726 12:48:53.395700       1 handler.go:37] scheduler call prioritize
I0726 12:48:53.395865       1 handler.go:44] scheduler call prioritize for default/busybox-56d9485878-kq7fr
I0726 12:48:53.395880       1 info.go:32] strategyData is Node 172.16.80.34 cpu usage 2.838333 mem usage 14.656576
I0726 12:48:53.395892       1 info.go:32] strategyData is Node 172.16.80.33 cpu usage 1.713333 mem usage 8.677264
I0726 12:48:53.395899       1 info.go:36] strategyDataList total count is 2
I0726 12:48:53.395909       1 extender.go:72] prioritize body: [{"Host":"172.16.80.33","Score":10},{"Host":"172.16.80.34","Score":0}]
```

5. 预期结果

- 负载超过阈值的 node 不会再部署其他服务，规避真实超过阈值的调度，调度结果符合预期，调度日志符合预期

2. 基于真实负载的调度策略：

> 描述：当节点超过安全水位时，新部署的pod会按照当前所有节点的真实负载最优的进行调度

1. 观察工作负载所在节点的负载情况

```
→  case02 kubectl top nodes
NAME            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
172.16.80.31    3085m        79%    2782Mi          17%
172.16.80.32    167m         4%     1717Mi          10%
172.16.80.33    75m          1%     2634Mi          16%
172.16.80.34    113m         2%     3866Mi          24%
172.16.80.35    147m         3%     1781Mi          11%
172.16.80.36    155m         3%     1712Mi          10%
```

2. 部署其他服务进行调度，并观察调度日志，发现node-172.16.80.33真实负载最低，得分最高

```
→  case02 kubectl apply -f cpu-02-normal.yaml
deployment.apps/busybox created
→  case02
```

```
kl (kubectl)
I0726 12:58:17.128818       1 prometheus.go:107] query machine_cpu_usage_5m{clusterID='cce-c5easmhc'} success
I0726 12:58:17.128835       1 prometheus.go:109] query total count is 24
I0726 12:58:17.131012       1 prometheus.go:107] query machine_memory_usage_5m{clusterID='cce-c5easmhc'} success
I0726 12:58:17.131025       1 prometheus.go:109] query total count is 24
I0726 12:58:17.131086       1 cache.go:32] UpdateMetricStore resource total count is 18
I0726 12:59:17.143944       1 prometheus.go:107] query machine_memory_usage_5m{clusterID='cce-c5easmhc'} success
I0726 12:59:17.143965       1 prometheus.go:109] query total count is 24
I0726 12:59:17.144056       1 prometheus.go:107] query machine_cpu_usage_5m{clusterID='cce-c5easmhc'} success
I0726 12:59:17.144064       1 prometheus.go:109] query total count is 24
I0726 12:59:17.144127       1 cache.go:32] UpdateMetricStore resource total count is 18
I0726 12:59:20.347329       1 handler.go:22] scheduler call filter
I0726 12:59:20.347648       1 handler.go:29] scheduler call filter for default/busybox-56d9485878-sgmtw
I0726 12:59:20.347679       1 info.go:32] strategyData is Node 172.16.80.34 cpu usage 2.905000 mem usage 14.770407
I0726 12:59:20.347696       1 info.go:32] strategyData is Node 172.16.80.31 cpu usage 77.198333 mem usage 8.832479
I0726 12:59:20.347709       1 info.go:32] strategyData is Node 172.16.80.33 cpu usage 2.048333 mem usage 9.829918
I0726 12:59:20.347716       1 info.go:36] strategyDataList total count is 3
I0726 12:59:20.347733       1 extender.go:112] feasible nodes body: ["172.16.80.34","172.16.80.33"]
I0726 12:59:20.347742       1 extender.go:115] failed nodes body: {"172.16.80.31":"Real cpu ratio 77.198333 great than 50.000000"}
I0726 12:59:20.348744       1 handler.go:37] scheduler call prioritize
I0726 12:59:20.348974       1 handler.go:44] scheduler call prioritize for default/busybox-56d9485878-sgmtw
I0726 12:59:20.348995       1 info.go:32] strategyData is Node 172.16.80.34 cpu usage 2.905000 mem usage 14.770407
I0726 12:59:20.349009       1 info.go:32] strategyData is Node 172.16.80.33 cpu usage 2.048333 mem usage 9.829918
I0726 12:59:20.349016       1 info.go:36] strategyDataList total count is 2
I0726 12:59:20.349030       1 extender.go:72] prioritize body: [{"Host":"172.16.80.33","Score":10},{"Host":"172.16.80.34","Score":0}]
```

3. 观察服务调度部署情况，发现部署的服务调度到负载最低（得分最高）的节点node-172.16.80.33 上

```
→  case02 kubectl get pods -owide
NAME                          READY   STATUS    RESTARTS   AGE     IP           NODE           NOMINATED NODE   READINESS GATES
busybox-56d9485878-sgmtw      1/1     Running   0          2m17s   10.4.3.181   172.16.80.33   <none>           <none>
cpu-stress-75dddc8656-xdh7d   1/1     Running   0          7h26m   10.4.5.93    172.16.80.31   <none>           <none>
```

4. 预期结果

- 负载超过阈值的node不会再部署其他服务，部署的服务会调度到真实负载最低的node-172.16.80.33上，调度结果符合预期

CCE Fluid说明

组件介绍

CCE Fluid是一个开源的Kubernetes原生的分布式数据集编排和加速引擎，主要服务于云原生场景下的数据密集型应用，例如大数据应用、AI应用等。

组件功能

- 数据集抽象
- 数据预热与加速
- 数据应用协同编排
- 多命名空间支持
- 异构数据源管理

使用场景

通过在使用数据集来创建AI训练任务，以提升数据访问速度加速机器学习训练

限制说明

- 支持v1.16版本以上的Kubernetes集群。

安装组件

1. 登录百度智能云官网，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**组件管理**。
6. 在组件管理列表中选择Fluid组件，点击“安装”按钮完成组件的安装。

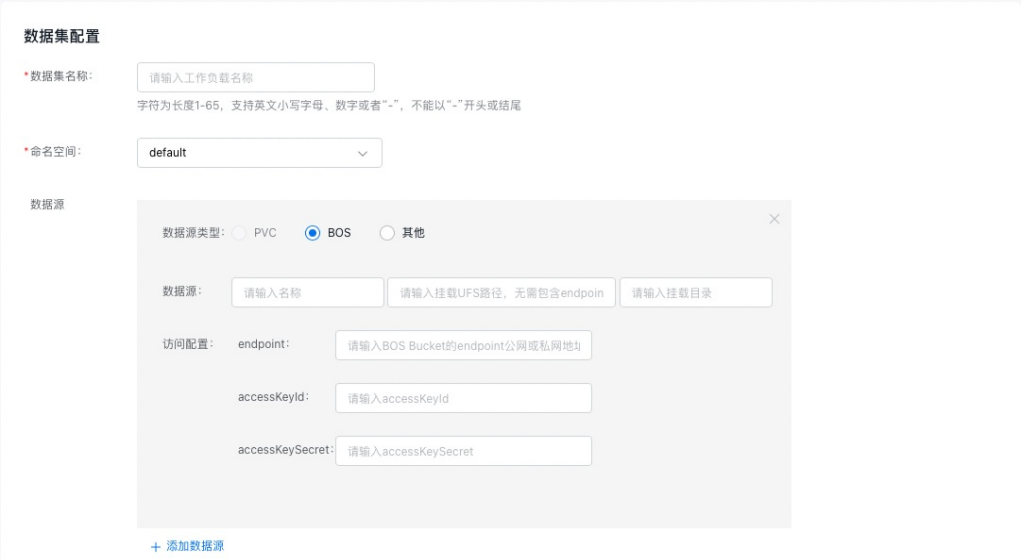
组件涉及基本概念

- Dataset：用于定义一组数据集，声明原始数据源的配置。
- Runtime：具体数据集存储引擎的实现，目前CCE支持以下两种Runtime：
  - RapidFSRuntime 百度自研存储引擎
  - AlluxioRuntime 开源存储引擎

组件使用-使用 RapidFS 加速 BOS 数据访问

1.创建BOS Bucket：需要创建一个存放原始数据的BOS Bucket并将训练数据上传到该Bucket中，建议将BOS创建在与计算节点相同地域以获得最佳访问速度。

2.创建数据集：数据源配置

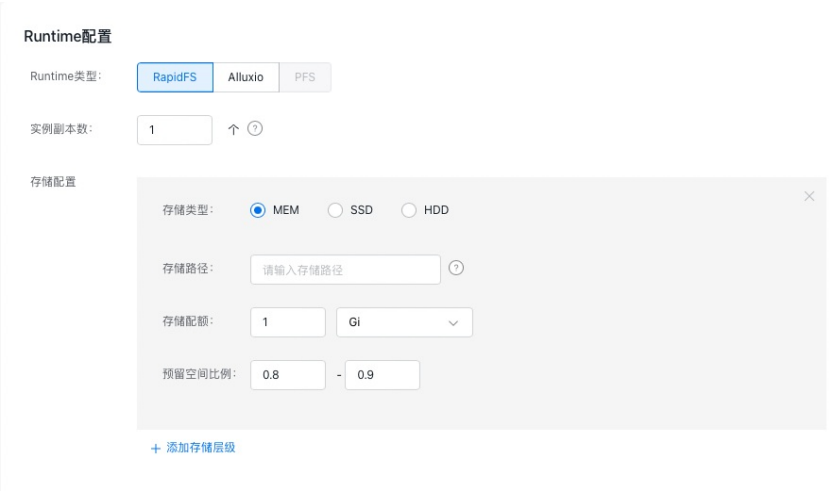


配置字段	说明
数据源名称	用于标识此数据源在这个数据集中的唯一名称，必填非空
数据源挂载 UFS 路径	数据源的BOS路径，格式为/<子路径>。例如 ： 1.mybucket/subdir ：使用mybucket下subdir子路径； 2.mybucket：使用 mybucket 整个根路径
数据源挂载路径	该数据源在此数据集下的子路径，如/subpath 。选填，如留空则使用数据源名称作为子路径
访问配置：endpoint	BOS访问Endpoint，如bj.bcebos.com， 可参考 获取访问域名
访问配置：accessKeyId	用于BOS访问accessKeyId
访问配置：accessKeySecret	用于BOS访问accessKeySeret

3.创建数据集：调度配置（可选）

创建数据集还可额外配置其容忍与亲和性策略，以将数据调度到指定的计算节点，当数据集与训练任务调度在相同节点时可获得最佳访问速度。

4.创建数据集：Runtime 配置



配置字段	说明
Runtime类型	存储引擎，目前支持自研RapidFS、开源存储引擎Alluxio和PFS
实例副本数	存储集群保存加速缓存的副本数量
存储类型	缓存介质类型，支持MEM/SSD/HDD，速度优先级依次递减
存储路径	存储引擎在节点上放置缓存的路径，选择内存缓存MEM时，可填入/dev/shm，选择 SSD/HDD时，可填入/mnt/diskx，具体路径取决于节点虚机的数据盘挂载路径用户可自行指定
存储配额	缓存的最大配额
预留空间比例	缓存驱逐的上下限比例，当使用缓存到达配额的上限比例时，存储引擎会执行数据驱逐操作，根据数据访问情况驱逐非热点数据

5.使用数据集：数据集创建成功后，可在“云原生AI> 任务管理>创建任务>数据配置”中选择数据集以在训练任务中挂载使用，数据集创建成功后也会自动在集群中创建一个同名PVC，也可在创建工作负载时直接使用该PVC挂载。

版本记录

版本号	适配集群版本	变更时间	变更内容	影响
v0.1.7	CCE v1.16+	2023.11.17	• fluid组件支持kubelet多路径配置 • 升级gcc12 • 新增rfsmount性能监控相关指标 • 修复rfsmount高并发rename发生目录死锁的问题 • 修复rfsmount sed -i丢失ACL权限的问题 • 修复rfsmount readdir无法显示正在写的文件最新长度的问题 • 修复rfsmount在master 0切主期间极小概率refresh master异常、出现挂载点不可用的问题 • fuse指定多个master endpoints来启动	此次升级不会对业务造成影响。

CCE CSI PFS Plugin 说明

组件介绍

CCE CSI PFS Plugin支持在Pod中以PV/PVC方式挂载百度云PFS存储，并支持动态创建PV。

组件功能

容器的特性决定了容器本身是非持久化的，容器被删除，其上的数据也一并删除。存储卷Volume可以解决容器的数据共享和持久化存储问题，PV（PersistentVolume）和PVC（PersistentVolumeClaim）是K8S提供的用于抽象存储细节的API资源。

利用百度云PFS服务在集群内创建PV和PVC资源，用户可以直接将PFS作为存储卷挂载到容器中，而无需关注底层的实现细节，从而更加便捷地为容器集群提供持久化存储方案。

使用限制

- PFS支持的操作系统和内核版本列表请查看 使用限制（标准型、增强型和极速型）。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品导览>容器>容器引擎**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**运维与管理>组件管理**"。
6. 在组件管理列表中选择"**CCE CSI PFS Plugin**"组件单击"**安装**"，在弹框中完成相关配置。

配置项	必选/可选	配置说明
KubeletRootPath	必选	指定节点上kubelet数据目录用于组件注册到节点Kubelet上，需要输入正确的节点kubelet数据目录，支持输入多个进行匹配。

7. 单击"**确认**"按钮完成组件的安装。

部署情况

安装CCE CSI PFS Plugin组件，将在Kubernetes集群中部署以下对象。某些资源对象根据部署参数可能会有所不同，这里展示组件默认内容。

对象名称	所属namespace	所占资源	类型
csi-clusterfileplugin	kube-system	-	CSIDriver
cce-csi-pfs-plugin-account	kube-system	-	ServiceAccount
cce-csi-pfs-plugin-runner	kube-system	-	ClusterRoleBinding
cce-csi-pfs-plugin-runner	kube-system	-	ClusterRole
cce-csi-pfs-plugin-resizer	kube-system	未限制	StatefulSet
cce-csi-pfs-plugin-resizer	kube-system	-	Service
cce-csi-pfs-plugin-provisioner	kube-system	未限制	StatefulSet
cce-csi-pfs-plugin-provisioner	kube-system	-	Service
cce-csi-pfs-plugin-nodeplugin-*	kube-system	未限制	DaemonSet
cce-csi-pfs-plugin-nodeplugin	kube-system	-	Service
cce-csi-pfs-plugin-attacher	kube-system	未限制	StatefulSet
cce-csi-pfs-plugin-attacher	kube-system	-	Service
cce-csi-pfs-plugin-clean-cce-reserved	kube-system	-	ConfigMap

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.2.7	CCE/1.18+	2024.10.12	优化csi node plugin	-
1.2.6	CCE/1.18+	2023.11.22	优化挂载脚本权限问题	-
1.2.5	CCE/1.18+	2023.08.10	去掉storageClass，k8s集群支持多个pfs实例	-
1.2.4	CCE/1.18+	2023.07.11	升级PFS client版本到7.0.0,去掉cluster参数	-
1.2.3	CCE/1.18+	2022.04.14	升级PFS client版本到6.8.1	-
1.2.2	CCE/1.18+	2022.06.28	兼容新版PFS内核模块	-
1.2.1	CCE/1.18+	2023.01.04	支持集群Kubernetes版本从1.20升级至1.22	-
1.2.0	CCE/1.18+	2022.12.13	优化pfs.configEndpoint参数格式，适配高版本集群	-
1.1.0	CCE/1.18+	2022.03.08	升级PFS client版本	-
1.0.0	CCE/1.18+	2022.01.13	Release版本	-

CCE CSI BOS Plugin 说明

组件介绍

CCE CSI BOS Plugin支持在Pod中以PV/PVC方式挂载百度云BOS存储。

组件功能

容器的特性决定了容器本身是非持久化的，容器被删除，其上的数据也一并删除。存储卷Volume可以解决容器的数据共享和持久化存储问题，PV（PersistentVolume）和PVC（PersistentVolumeClaim）是K8S提供的用于抽象存储细节的API资源。

利用百度智能云BOS服务在集群内创建PV和PVC资源，用户可以直接将BOS作为存储卷挂载到容器中，而无需关注底层的实现细节，从而更加便捷地为容器集群提供持久化存储方案。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品导览>容器>容器引擎**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**运维与管理>组件管理**"。
6. 在组件管理列表中选择"**CCE CSI BOS Plugin**"组件单击"**安装**"，在弹框中完成相关配置。

配置项	必选/可选	配置说明
KubeletRootPath	必选	指定节点上kubelet数据目录用于组件注册到节点Kubelet上，需要输入正确的节点kubelet数据目录，支持输入多个进行匹配。
maxVolumesPerNode	必选	指定集群中每个节点最多可挂载BOS类型PV数量。

7. 单击"**确认**"按钮完成组件的安装。

部署情况

安装CCE CSI BOS Plugin组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
csi-bos-external-runner	kube-system	-	ServiceAccount
csi-bos-external-runner	kube-system	-	ClusterRole
csi-bos-role-binding	kube-system	-	ClusterRoleBinding
csi-bos-external-runner-cfg	kube-system	-	Role
csi-bos-role-cfg	kube-system	-	RoleBinding
csi-bosplugin-node-server-*	kube-system	未限制	DaemonSet
csi-bosplugin	kube-system	-	CSIDriver
cce-csi-bos-plugin-cce-reserved	kube-system	-	ConfigMap

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.4.3	CCE/v1.22+	2024.02.20	修复NodeName为主机名称的问题	-
1.4.2	CCE/v1.22+	2024.01.04	优化bos csi存活探针脚本	-
1.4.1	CCE/v1.22+	2023.09.01	支持containerd容器运行时	-
1.4.0	CCE/v1.22+	2023.06.25	添加对CCE Kubernetes 1.22集群的支持	-
1.3.0	CCE/v1.18+	2023.08.10	支持arm64架构CCE集群	-
1.2.0	CCE/v1.18+	2022.03.25	拦截鉴权失败错误，优化报错信息	-
1.1.2	CCE/v1.18+	2022.02.10	默认 topologyMode 修改为 auto，兼容 DCC 使用场景	-
1.1.1	CCE/v1.18+	2021.11.17	修正错误的 Volume Type	-
1.1.0	CCE/v1.18+	2022.07.06	部署支持设置 Tolerations	-
1.0.0	CCE/v1.18+	2021.04.04	部署开源版本 v1.0.0 CSI driver	-
0.2.2	CCE/v1.18+	2021.02.16	支持自定义镜像和服务Endpoints	-
0.2.1	CCE/v1.18+	2020.12.10	支持通过节点亲和性支持kubelet数据目录异构的集群	-
0.2.0	CCE/v1.18+	2020.10.20	支持在CCE Kubernetes 1.18 集群中安装插件	-
0.1.0	CCE/v1.16+	2020.09.01	初始版本，支持在CCE Kubernetes 1.16 集群中安装插件	-

CCE CSI CDS Plugin 说明

组件介绍

CCE CSI CDS Plugin支持在Pod中以PV/PVC方式挂载百度云CDS存储，并支持动态创建PV。

组件功能

容器的特性决定了容器本身是非持久化的，容器被删除，其上的数据也一并删除。存储卷Volume可以解决容器的数据共享和持久化存储问题，PV（PersistentVolume）和PVC（PersistentVolumeClaim）是K8S提供的用于抽象存储细节的API资源。

利用百度智能云CDS磁盘在集群内创建PV和PVC资源，用户可以直接将CDS作为存储卷挂载到容器中，而无需关注底层的实现细节，从而更加便捷地为容器集群提供持久化存储方案。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择”**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的”**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击”**运维与管理>组件管理**”。
6. 在组件管理列表中选择“**CCE CSI CDS Plugin**”组件单击”**安装**”，在弹框中完成相关配置。

配置项	必选/可选	配置说明
KubeletRootPath	必选	指定节点上kubelet数据目录用于组件注册到节点Kubelet上，需要输入正确的节点kubelet数据目录，支持输入多个进行匹配。
maxVolumesPerNode	必选	指定集群中每个节点最多可挂载CDS类型PV数量。

7. 单击"**确认**"按钮完成组件的安装。

部署情况

安装CCE CSI CDS Plugin组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
csi-cds-external-runner	kube-system	-	ServiceAccount
csi-cds-external-runner	kube-system	-	ClusterRole
csi-cds-role-binding	kube-system	-	ClusterRoleBinding
csi-cds-external-runner-cfg	kube-system	-	Role
csi-cds-role-cfg	kube-system	-	RoleBinding
csi-cdsplugin-node-server-*	kube-system	未限制	DaemonSet
csi-cdsplugin-controller-server	kube-system	CPU: 100m Memory: 100Mi	Deployment
cce-csi-cds-plugin-cce-reserved	kube-system	-	ConfigMap

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.4.8	CCE/v1.22+	2025.01.25	• 支持高性能应用服务HPAS类型节点 • 增加节点对应实例规格可挂载CDS数量校验	-
1.4.7	CCE/v1.22+	2024.06.13	支持动态挂载可配置删除时是否进入回收站	-
1.4.6	CCE/v1.22+	2024.05.15	支持边缘集群挂载CDS	-
1.4.5	CCE/v1.22+	2024.03.12	优化gateway获取token方式，避免token过期导致挂载失败	-
1.4.4	CCE/v1.22+	2024.01.04	支持EBC机型挂载CDS	-
1.4.3	CCE/v1.22+	2023.10.20	修复对dev path字段依赖，修复可能造成的盘符残留	-
1.4.2	CCE/v1.22+	2023.06.28	设置默认fsType字段保证向前兼容	-
1.4.1	CCE/v1.22+	2023.01.04	支持集群Kubernetes版本从1.20升级至1.22	-
1.4.0	CCE/v1.22+	2022.10.10	添加对CCE Kubernetes 1.22集群的支持	-
1.3.0	CCE/v1.18+	2022.08.10	支持arm64架构CCE集群	-
1.2.0	CCE/v1.18+	2022.03.25	提升创建/挂载/漂移场景性能	-
1.1.2	CCE/v1.18+	2021.12.16	默认 topologyMode 修改为 auto，兼容 DCC 使用场景	-
1.1.1	CCE/v1.18+	2021.12.06	允许 GPU 节点运行 CSI CDS Node Server	-

	18+	0.26		
1.1.0	CCE/v1.18+	2021.06.28	CSI CDS Driver 支持 ExpandVolume 和 VolumeStats；部署支持设置 Tolerations	-
1.0.0	CCE/v1.18+	2021.08.11	部署开源版本 v1.0.0 CSI driver	-
0.5.3	CCE/v1.18+	2021.01.21	修改默认cdsplugin image	-
0.5.2	CCE/v1.18+	2020.12.29	支持自定义镜像和服务Endpoints	-
0.5.1	CCE/v1.18+	2020.12.10	支持通过节点亲和性支持kubelet数据目录异构的集群	-
0.5.0	CCE/v1.18+	2020.12.10	添加对CCE Kubernetes 1.18集群的支持 && 添加kubelet 数据目录配置	-
0.4.0	CCE/v1.16+	2020.11.29	添加对CCE Kubernetes 1.16集群的支持	-
0.3.4	CCE/v1.13+	2020.10.28	增加csi依赖的crds	-
0.3.3	CCE/v1.13+	2020.09.21	修复`external-runner-rolebinding`中的字段错误	-
0.3.2	CCE/v1.13+	2020.08.26	更新sidecar组件镜像，修复CVE-2019-11255	-
0.3.1	CCE/v1.13+	2020.07.15	更新`cdsplugin`镜像至`latest-1.13` / `latest-1.11`	-
0.3.0	CCE/v1.13+	2020.06.19	• 资源强制部署在kube-system下以支持priority class • 修改1.13集群中storageClass的`volumeBindingMode`为`WaitForFirstConsumer` • 1.11和1.13集群采用不同的指定cds所在可用区的方案，1.13集群中会自动根据cds可用区影响调度结果，1.11集群需要自行指定pod的节点可用区亲和性	-
0.2.2	CCE/v1.13+	2020.05.07	修复参数中`cdsSizeInGB`字段含义和默认值	-
0.2.1	CCE/v1.13+	2020.04.22	修复参数中`zone`字段为nil时插件创建无效storageClass的问题	-
0.2.0	CCE/v1.13+	2020.03.17	添加对CCE Kubernetes 1.13集群的支持	-
0.1.0	CCE/v1.11+	2020.01.20	初始版本，支持在CCE Kubernetes 1.11集群中安装插件	-

CCE NPU Manager 说明

组件介绍

一系列NPU Device Plugin、Exporter的集合，结合配套的Scheduler可以实现NPU资源调度能力。

> 目前该组件依赖于CCE AI Job Scheduler，若您需要请一同安装，否则可能导致组件功能不可用。

组件功能

支持对NPU资源管理、分配以及指标上报，支持使用RDMA网络。

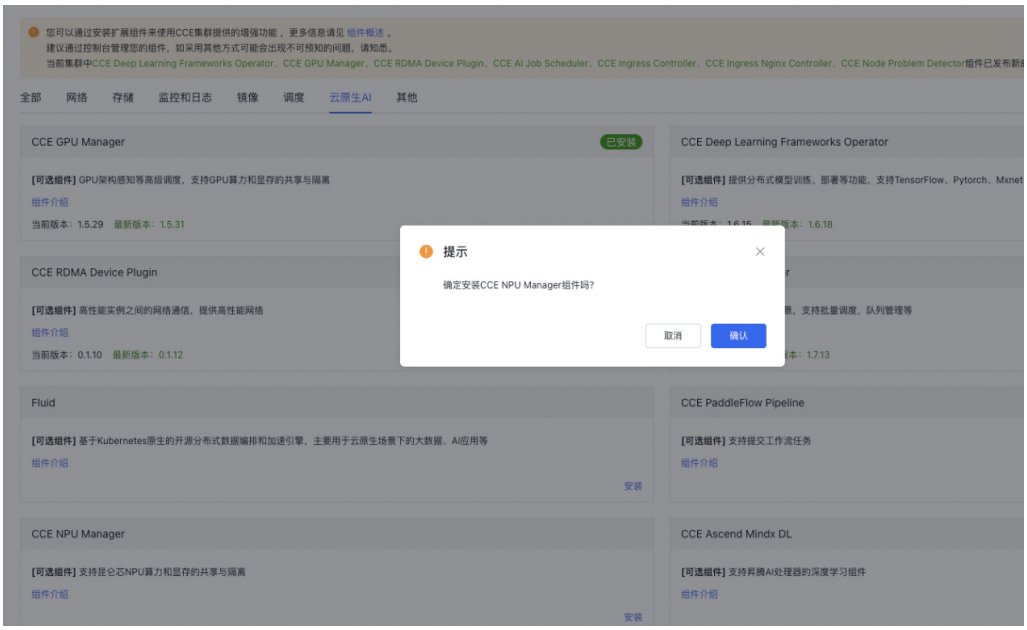
使用场景

在CCE集群中有NPU资源时，需要安装该组件后才能正常调度使用。

安装组件

1. 登录百度智能云官网，并进入管理控制台。

2. 选择"**产品服务>云原生>容器引擎CCE**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**运维与管理 > 组件管理**"。
6. 在组件管理列表中选择"**CCE NPU Manager**"组件单击"**安装**"。



7. 点击"**确认**"按钮完成组件的安装。

组件状态确认

通过如下命令查看K8s集群中跟CCE NPU Manager组件相关的Pod，需要满足以下Pod的STATUS都为Running，READY都为1/1，则该组件运行状态为正常。（该命令输出的pod数量跟集群节点数量有关，每个节点对应3个pod）

```
kubectl -n kube-system get po | grep xpu
```

```
 xpu-device-plugin-daemonset-v3-8pzxn              1/1     Running   0          55s
xpu-exporter-v3-bm6cd                      1/1     Running   0          55s
rdma-shared-dp-ds                          1/1     Running   0          55s
```

CCE Image Accelerate 说明

组件介绍

CCE Image Accelerate组件提供容器镜像按需加载能力，通过配合使用容器镜像服务CCR的镜像加速功能，实现镜像数据免全量下载和在线解压，大幅减少容器启动时间，提升应用分发效率。

组件功能

- cce-image-accelerate-controller: 镜像加速单机驱动的安装或卸载的控制器，可根据用户期望为某个node执行安装或卸载镜像加速驱动。
- cce-image-accelerate-installer：镜像加速驱动安装器，部署在每个需要安装镜像加速的节点上。
- cce-image-accelerate-uninstaller: 镜像加速卸载器，卸载并还原已经安装镜像加速驱动的节点。

使用场景

在容器业务部署组中免容器数据全量下载和在线解压，大幅减少容器启动时间，享受极致弹性体验。

使用限制

该组件存在以下使用限制：

配置	**版本**
容器运行时	Containerd 1.4.2以上，暂时不支持Docker
操作系统发行版	Centos 7.x、Ubuntu 16.x以上
Kubernetes集群版本	当前只支持1.18.9，1.20.8，1.22.5，1.24.4版本

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品服务>容器>容器引擎CCE**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**运维与管理>组件管理**"。
6. 在组件管理列表中选择"**CCE Image Accelerate**"组件单击"**安装**"。

7. 点击"**确认**"按钮完成组件的安装。

部署情况

安装CCE Image Accelerate组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
cce-image-accelerate-controller	kube-system	-	Deployment
cce-image-accelerate-installer	kube-system	-	DaemonSet
cce-image-accelerate-uninstaller	kube-system	-	DaemonSet
image-accelerate-manager-role	-	-	ClusterRole
image-accelerate-manager-role	-	-	ClusterRoleBinding
image-accelerate	kube-system	-	ServiceAccount
image-accelerate-service-mutating-webhook-configuration	-	-	MutatingWebhookConfiguration
image-accelerate-webhook	kube-system	-	Service

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.8.3	CCE/v1.18+	2022.06.28	首次上线	-

CCE Ingress Nginx Controller 说明

组件介绍

CCE Ingress Nginx Controller是Kubernetes集群中Ingress资源的控制器，该控制器在Kubernetes社区的Ingress Nginx Controller基础上优化了一些参数。

组件功能

该控制器使用Nginx作为反向代理和负载均衡器，实现了Kubernetes Ingress的语义和功能。

- 除了Kubernetes Ingress规定的配置之外，还提供诸多自定义配置项使得用户可以实现更灵活和精细的负载均衡策略。
- 组件中包含一个LoadBalancer类型的Service。部署组件后CCE会关联创建一个应用型BLB，以便将流量接入到集群中的该组件的Pod中去。

使用场景

当BLB类型Ingress无法满足需求时，可以使用**Nginx类型Ingress**来实现更灵活和精细的负载均衡策略。

使用限制

- 在使用Nginx Ingress时，数据流量会流经组件的Pod。因此该组件需要部署在单独节点组中，避免其他Pod挤占组件的资源。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击”**运维与管理>组件管理**”。
6. 在组件管理列表中选择“**CCE Ingress Nginx Controller**”组件单击“**安装**”，在弹框中完成相关配置。

配置项	必选/可选	配置说明
IngressClass名称	必选	指定IngressClass的名称，IngressClass名称是唯一的，用于区分不同的Ingress控制器。创建Ingress时，可通过指定控制器名称，声明该Ingress由此控制器进行管理。
命名空间	必选	指定IngressClass能够监听哪些命名空间下Ingress资源。
部署方式	必选	目前仅支持选择**节点组DaemonSet部署**，组件将以DaemonSet的形式部署在指定节点组中，后续您只要调整节点组的数量，即可扩缩容实例的副本数。
选择节点组	必选	指定组件部署的目标节点组。
容器配额	必选	指定组件部署Pod运行所分配的CPU、内存request值和limit值。 **CPU request**：容器需要使用的最小 CPU 值。 **CPU limit**：允许容器使用的 CPU 最大值。如果超过，容器会被终止。 **内存 request**：容器需要使用的最小内存值。 **内存 limit**：允许容器使用的内存最大值。如果超过，容器会被终止。
容忍设置	可选	为组件部署的Pod设置容忍信息。Pod能够调度到容忍能够匹配到污点的节点上。
服务访问	必选	为Ingress控制器创建LoadBalancer类型Service提供访问入口，分别支持： **公网**：系统将创建应用型负载均衡器BLB并绑定弹性公网EIP。 **内网**：系统只会创建应用型负载均衡器BLB。
公网带宽	必选	当**服务访问**为**公网**时，需要配置该参数。配置BLB绑定EIP的公网带宽分别支持： **按使用流量计费**：指定带宽峰值后，将按照实际使用的出公网流量计费，与使用时长无关。 **按使用带宽计费**：指定带宽峰值后，将按照使用时长计费，与实际流量无关。
带宽峰值	必选	当**服务访问**为**公网**时，需要配置该参数。配置BLB绑定EIP的带宽上限。

7. 单击“**确认**”按钮完成组件的安装。
8. 您可以再次单击”**新增实例**”创建多个不同配置的 Nginx Ingress Controller，以满足不同的服务访问场景。在创建 Ingress 资源对象时选择 IngressClass，即可为其指定相应的 Nginx Ingess Controller。

卸载组件

组件管理控制台暂不支持卸载已安装的Nginx Ingress Controller实例，若您想卸载可以在**Helm>Helm实例**中找到目标实例进行删除

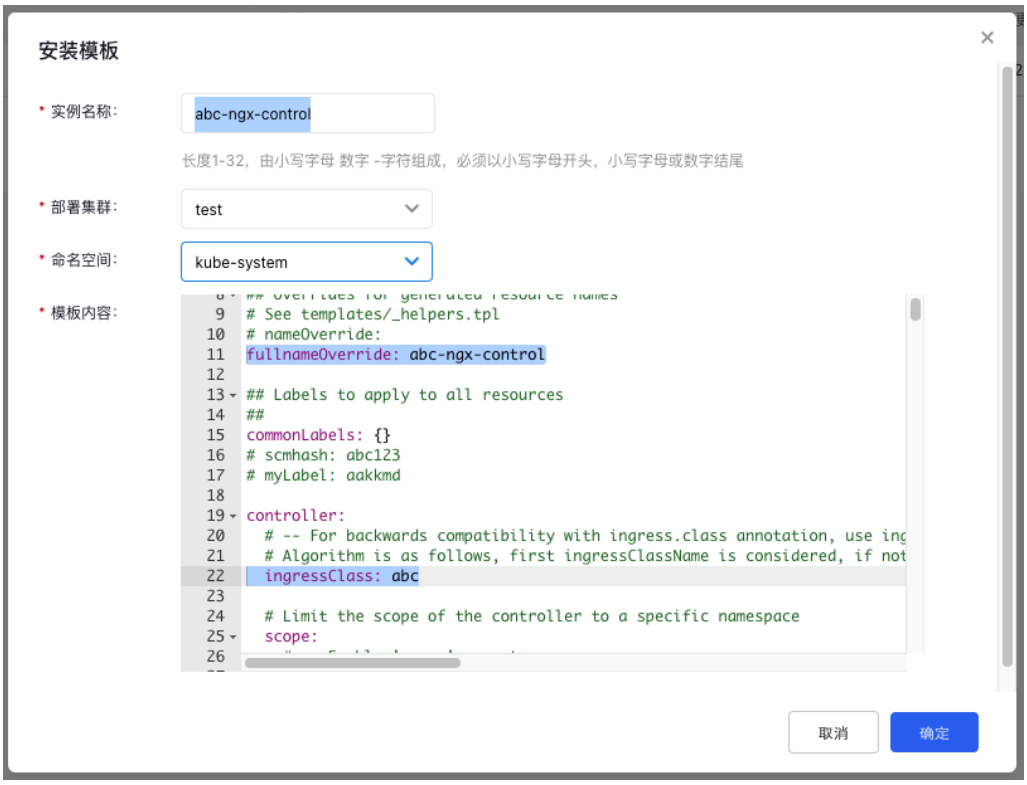
使用百度智能云模板部署组件

CCE Ingress NGINX Controller组件的配置项比较多，但“组件管理”功能中开放的配置是有限的，您也可以在**Helm>Helm模板**中找到**cce-ingress-nginx-controller**模板进行部署和修改配置。

如果希望自行在集群中部署该组件，并与“组件管理”实现联动效果，则在Helm部署时，需要同时满足以下条件：

1. Helm**实例名称**为{ingress-class}-ngx-control。
2. 部署参数**fullnameOverride**为{ingress-class}-ngx-control。

其中{ingress-class}取值和部署参数中的**controller.ingressClass**参数一致，如图所示：



部署情况

安装CCE Ingress NGINX Controller组件后，将在Kubernetes集群中部署以下资源对象。需要注意的是，在不同的部署配置下，部署内容会有一些差异。

- 某些对象在相关部署参数启用的情况下才会被部署到集群中。 部分对象的名称及命名空间取决于插件的部署配置。

对象名称	所属NameSpace	所占资源	类型
cce-reserved	kube-system	-	ConfigMap
clusterrole	-	-	ClusterRole
clusterrolebinding	-	-	ClusterRoleBinding
controller-configmap	取决于部署参数	-	ConfigMap
controller-configmap-addheaders	取决于部署参数	-	ConfigMap
controller-configmap-proxyheaders	取决于部署参数	-	ConfigMap
controller-configmap-tcp	取决于部署参数	-	ConfigMap
controller-configmap-udp	取决于部署参数	-	ConfigMap
controller-daemonset	取决于部署参数	取决于部署参数	DaemonSet
controller-deployment	取决于部署参数	取决于部署参数	Deployment
controller-hpa	取决于部署参数	-	HorizontalPodAutoscaler
controller-keda	取决于部署参数	-	ScaledObject
controller-poddisruptionbudget	取决于部署参数	-	PodDisruptionBudget
controller-prometheusrules	取决于部署参数	-	PrometheusRule
controller-psp	取决于部署参数	-	PodSecurityPolicy
controller-role	取决于部署参数	-	Role
controller-rolebinding	取决于部署参数	-	RoleBinding
controller-service	取决于部署参数	-	Service
controller-service-internal	取决于部署参数	-	Service
controller-service-metrics	取决于部署参数	-	Service
controller-service-webhook	取决于部署参数	-	Service
controller-serviceaccount	取决于部署参数	-	ServiceAccount
controller-servicemonitor	取决于部署参数	-	ServiceMonitor
default-backend-deployment	取决于部署参数	-	Deployment
default-backend-hpa	取决于部署参数	-	HorizontalPodAutoscaler
default-backend-poddisruptionbudget	取决于部署参数	-	PodDisruptionBudget
default-backend-psp	取决于部署参数	-	PodSecurityPolicy
default-backend-role	取决于部署参数	-	Role
default-backend-rolebinding	取决于部署参数	-	RoleBinding
default-backend-service	取决于部署参数	-	Service
default-backend-serviceaccount	取决于部署参数	-	ServiceAccount
dh-param-sercret	取决于部署参数	-	Secret
controller-ingressclass	取决于部署参数; 仅在 4.2.5 版本中存在	-	IngressClass
controller-wehbooks-networkpolicy	取决于部署参数; 仅在 4.2.5 版本中存在	-	NetworkPolicy

版本记录

版本号	适配集群版本	更新时间	更新内容	社区版本
4.11.5	CCE/v1.26/v1.28/v1.30	2025-03-31	更新到社区版本v1.11.5，修复CVE-2025-1097、CVE-2025-1098、CVE-2025-1974、CVE-2025-24513、CVE-2025-24514漏洞	v1.11.5
4.11.2	CCE/v1.26/v1.28/v1.30	2024-09-20	支持 Kubernetes v1.26 及以上集群使用	v1.11.2
4.2.5	CCE/v1.20/v1.22/v1.24	2022-11-01	支持 Kubernetes v1.22/v1.24 集群使用	v1.3.1
3.29.0	CCE/v1.16/v1.18/v1.20	2022-06-29	首次上线	v0.45.0

CCE Ingress Controller 说明

组件介绍

CCE Ingress Controller组件基于百度智能云应用型负载均衡BLB实现 Kubernetes Ingress 语义，提供7层（HTTPS、HTTPS）负载均衡能力。

组件功能

用户在CCE控制台创建BLB类型Ingress，或是使用YAML创建带有 `kubernetes.io/ingress.class: cce` Annotation 的Ingress资源后，CCE Ingress Controller组件根据该Ingress资源对象中定义的转发规则和Annotation中的配置信息，在应用型BLB和EIP上配置对应的参数，实现Ingress资源对象负载均衡的语义。

注意事项

- CCE Ingress Controller为控制面组件，数据流量不会经过该组件，请不要主动调节组件的副本数。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击”**运维与管理>组件管理**”。
6. 在组件管理列表中选择“**CCE Ingress Controller**”组件单击“**安装**”。
7. 在确认框中单击“**确认**”按钮完成组件的安装。

> **说明：**
>
> - CCE Ingress Controller组件在托管集群中为托管组件，部署在百度智能云CCE上，无需用户运维；在独立集群中部署在集群节点上，需要用户自行运维。
> - 仅带有【托管】标识的组件才表示该组件部署在百度智能云CCE上，为托管组件。

部署情况

安装 CCE Inress Controller组件，将在 Kubernetes 集群中部署以下对象：

对象名称	所属NameSpace	资源类型	所占资源
cce-ingress-controller-cce-reserved	kube-system	-	ConfigMap
cce-ingress-controller	kube-system	副本数：3，CPU: 200m，Memory: 500Mi	Deployment
cce-ingress-clusterrole	kube-system	-	ClusterRole
cce-ingress-binding	kube-system	-	ClusterRoleBinding
cce-ingress-serviceaccount	kube-system	-	ServiceAccount

版本记录

版本号	适配集群版本	更新时间	更新内容	限制和影响
1.3.16	CCE/v1.18+	2024-08-29	修复VPC内IP可能被置空的问题	无
1.3.15	CCE/v1.18+	2024-08-07	支持配置创建eip的接口回调等待时间参数	无
1.3.14	CCE/v1.18+	2024-07-01	• 支持托管部署在百度智能云CCE上 • 修复ExtendedAttributes字段解析失败问题	无
1.3.13	CCE/v1.18+	2024-05-28	• 支持边缘节集群 • 支持为组件指定资源配置	无
1.3.12	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2024.03.19	解决日志问题	无
1.3.11	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023.11.22	修复 BLB 健康检查端口问题	无
1.3.10	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023.10.28	支持健康检查配置	无
1.3.9	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023-08-18	为 Ingress 相关 BLB 和 EIP 绑定 Tag	无
1.3.8	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023-08-11	完善监听器配置 支持配置 EIP 带宽 支持默认为 BLB 开启"删除保护"功能	无
1.3.7	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023-07-24	内部优化	无
1.3.6	CCE/v1.18/v1.20/v1.22/v1.24/v1.26	2023-05-17	支持配置默认EIP计费类型 修复忽略 Cordon 节点问题 支持会话保持配置、调度算法、XFF	无
1.3.5	CCE/v1.18/v1.20/v1.22	2023-01-17	取消对 BLB Name 等字段的更新 增加 Pod 直连功能开关 修复 Service 端口匹配错误问题 兼容 1.22 版本集群	无
1.3.4	CCE/v1.18/v1.20	2022-06-22	支持 XFF 与 Real-IP 修复若干问题	无
1.3.3	CCE/v1.18/v1.20	2022-05-24	兼容旧版转发规则	无

CCE Log Operator 说明

组件介绍

CCE日志管理组件，支持用户通过CRD的方式创建日志规则，并在container上添加相关的环境变量从而实现容器日志的采集。

组件功能

- 对接日志服务BLS实现日志的采集与存储。
- 支持用户在container上添加`cce_log_internal`设置环境变量采集指定文件的log，支持`cce_log_stdout`环境变量采集容器终端输出的日志。

使用场景

采集集群控制面组件日志及业务日志，更多信息请见日志概述。

使用限制

集群Kubernetes版本为1.16及以上。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品服务>云原生>容器引擎CCE**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**运维与管理>组件管理**”。
6. 在组件管理列表中选择“**CCE Log Operator**”组件单击“**安装**”。

7. 点击“**确认**”按钮完成组件的安装。

部署情况

安装CCE Log Operator组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
cce-log-operator	kube-system	CPU:100m Memory:10Mi	Deployment
cce-log-agent-v2	kube-system	CPU:100m Memory:100Mi	DaemonSet
logrules.cce.io	-	-	CRD
logconfig.cce.baidubce.com	-	-	CRD

版本记录

版本号	适配集群版本	更新时间	更新内容
2.3.1	CCE/v1.18+	2022.12.05	支持BLS
2.2.3	CCE/v1.18+	2022.10.17	升级CRD版本v1beta1 => v1
2.2.2	CCE/v1.18+	2022.06.27	代码重构
2.2.1	CCE/v1.18+	2021.11.15	适配containerd
2.0.0	CCE/v1.18+	2021.07.05	首次上线

CCE P2P Accelerator 说明

使用前提

- 创建容器镜像服务CCR企业版实例，且创建的企业版实例规格为**标准版**或**高级版**，具体操作请参考 创建企业版实例。
- 创建CCE集群且容器集群需要具体以下配置，具体操作请参考 创建集群。
  - 集群Kubernetes版本在1.18及以上。
  - 不支持ARM架构集群。
- 容器镜像服务CCR企业版实例中配置了私有网络访问控制，具体操作请参考 配置私有网络访问控制。其中在配置CCR企业版私有网络控制时选择的私有网络对应着CCE集群所在的私有网络，CCE集群所在的私有网络查询请参考 查看集群。

适用场景

当大规模容器集群批量下载镜像时，容器镜像存储的网络带宽会成为性能瓶颈，导致镜像拉取缓慢。P2P加速功能利用集群中计算节点的带宽资源，进行节点之间镜像分发，减少对容器镜像存储的压力，可以大幅提升镜像拉取速度，减少应用部署时间。

组件介绍

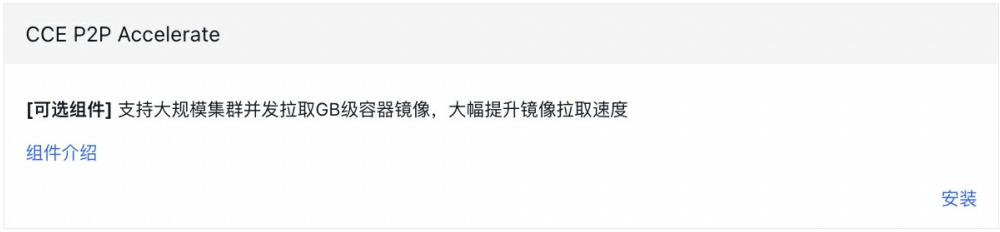
CCE P2P Accelerate 组件提供容器镜像P2P加速能力，用户在容器引擎CCE集群中部署P2P加速组件，部署或更新业务拉取镜像时，利用集群计算节点的带宽资源，进行节点之间镜像分发，提升镜像拉取速度。

限制说明

- 仅支持 v1.18 及以上版本的 Kubernetes 集群，且不支持ARM架构集群。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择”**产品服务>云原生>容器引擎CCE**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的”**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 选择“**运维与管理**”左侧栏，单击”**组件管理**”，切换Tab页到“**镜像**”。
6. 在组件管理列表中选择“**CCE P2P Accelerater**”组件单击”**安装**”。



7. 输入CCR实例ID，带宽限速和端口后，点击"**确认**"按钮完成组件的安装。

启用P2P加速

CCE集群通过给容器或命名空间打标签的方式开启P2P，您可以根据实际情况选择合适的方式：

> **说明：**标签的名称为`cce.baidubce.com/p2p-accelerate`，值为`true`。

- 单独为容器组添加镜像加速标签，例如Deployment、DaemonSet等。

  > **注意:** 容器的p2p标签需要打在工作负载的spec.template.metadata.labels下才可以有效的开启p2p功能。

- 也可以为CCE集群的命名空间设置标签，该命名空间下所有符合加速条件的工作负载都会启用按需加载容器镜像，无需再修改工作负载的YAML文件。

启用P2P加速后，加速组件会自动给Pod注入相关Annotation、P2P加速镜像地址以及对应的镜像拉取凭证（其中仅镜像仓库地址不一样，镜像拉取凭证与原先一致）。

P2P加速示例

1、在Deployment文件中的template中添加cce.baidubce.com/p2p-accelerate:true标签

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-example
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        cce.baidubce.com/p2p-accelerate: "true" #启用P2P加速
    spec:
      containers:
        - name: nginx
          image: ccr-1xgztest-vpc.cnc.gz.baidubce.com/eccr-qa-private/nginx:1.22.0
      imagePullSecrets：
        - name：test-registry
```

2、Deployment中Pod已注入P2P相关annotation、P2P加速镜像地址以及对应的镜像拉取凭证，说明启用P2P加速成功

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    cce.baidubce.com/p2p-accelerate: "true" #启用P2P加速
spec:
  containers:
  - image: ccr-1xgztest-vpc-p2p.cnc.gz.baidubce.com:60001/eccr-qa-private/nginx:1.22.0 #p2p代理以后的容器镜像地址
  imagePullSecrets:
  - name: test-registry
```

组件功能

- cce-p2p-accelerator：获取命名空间或者工作负载的上的标签cce.baidubce.com/p2p-accelerate，判断是否启用p2p镜像加速， 如果value为true，则修改镜像地址为p2p地址。
- cce-p2p-accelerator-dragonfly-dfdaemon：镜像运行守护进程，提供任务下载能力。

部署情况

安装 CCE P2P Accelerate 组件，将在 Kubernetes 集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
cce-p2p-accelerator	kube-system	-	Deployment
dragonfly-dfdaemon	kube-system	-	DaemonSet
cce-p2p-accelerator	kube-system	-	ServiceAccount
cce-p2p-accelerator	kube-system	-	Secret
cce-p2p-accelerator	kube-system	-	ConfigMap
dragonfly-dfdaemon	kube-system	-	ConfigMap
cce-p2p-accelerator	kube-system	-	ConfigMap
cce-p2p-accelerator	-	-	ClusterRole
cce-p2p-accelerator	-	-	ClusterRoleBinding
cce-p2p-accelerator	kube-system	-	ServiceAccount
cce-p2p-accelerator	kube-system	-	Service
cce-p2p-accelerator	kube-system	-	MutatingWebhookConfiguration

支持 Annotations

参数名	类型	说明	示例
cce.baidubce.com/p2p-accelerate	string	是否开启P2P镜像加速	"true"

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.1.1	CCE/v1.18+	2024.04.11	优化scheduler部署架构，提升p2p种子获取效率	-
1.0.1	CCE/v1.18+	2023.03.21	首次上线	-

CCE Node Problem Detector 说明

组件介绍

在 Kubernetes 集群的实际使用过程中，节点可能由于各种原因出现异常，导致影响服务的整体可用性。另外，由于 Kubernetes 的控制面在默认情况下，无法及时感知节点异常，导致在调度的过程中，将实例调度到不可用的节点上；而作为集群的使用者，用户往往也无法及时感知到集群中节点或实例的故障，导致不能及时进行止损处理。当遇到节点或实例故障时，用户期望平台能实现自动的故障恢复，例如自动迁移故障节点上的实例等。

Node-Problem-Detector 是为集群提供节点故障检测的扩展能力，用户在集群中安装该组件后，会以 DaemonSet 形式运行，来实时检测节点上的各种异常情况，并将检测结果报告给上游的 Kube-APIServer。

组件功能

- 提供节点故障检测能力
- 支持的故障上报方式包括
  - NodeCondition（节点状况）：可能造成 Pod 无法在这个节点运行
  - Event（事件）：影响节点的临时性问题，但是它对于系统诊断是有意义的
  - Metrics：以上两种情况，会同时生成 Prometheus 指标，可供采集和展示

限制说明

- 集群版本在 1.18.9 以上

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 集群管理 > 集群列表 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 组件管理 。
6. 在组件管理列表中选择 CCE Node Problem Detector 组件单击“安装”。
7. 点击“确定”按钮完成组件的安装。

查看集群健康检查状态

安装组件后，可以在节点健康检查界面查看故障情况。

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 集群管理 > 集群列表 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 节点管理 > Worker，进入节点管理界面。
6. 点击 健康检查 图标即可展示节点的健康检查状态：

7. 查看健康检查状态。目前，仅 NetworkUnavailable 和 Ready 会影响节点是否可用。

健康检查

最近检查结果： 可用

检查项	检查结果	具体信息
NetworkUnavailable	False	CCE RouteController created a
MemoryPressure	False	kubelet has sufficient memory
DiskPressure	False	kubelet has no disk pressure
PIDPressure	False	kubelet has sufficient PID avail
Ready	True	kubelet is posting ready status
InodesPressure	False	node has no inodes pressure

故障状态

Node Conditions

安装 CCE-Node-Problem-Detector 后，会在节点中添加以下 Conditions：

Condition Type	默认值	说明	影响
FrequentKubeletRestart	False	Kubelet是否在20Min内重启超过5次	影响容器创建
FrequentContainerRuntimeRestart	False	Docker或Containerd是否在20Min内重启超过5次	影响容器创建
ContainerRuntimeUnhealthy	False	节点容器运行时是否可用	影响容器创建
CorruptDockerOverlay2	False	Docker Overlay2文件系统错误	影响 Docker 容器运行时正常运行
InodesPressure	False	节点Inodes使用是否大于80%	影响目录/文件的创建
KernelDeadlock	False	内核是否存在死锁	可能影响容器的创建、删除和运行
ReadonlyFilesystem	False	文件系统是否只读	进程无法新建目录/文件，无法执行写操作
GPUUnhealthy	False	是否存在 GPU 故障（仅支持带有 GPU 设备的 EBC 机型）	1. GPU 发生故障不可用 2. 可能导致 GPU 训练/推理任务中断
NICUnhealthy	False	是否存在网卡故障（仅支持带有 GPU 设备的 EBC 机型）	1. 可能导致节点、Pod 间网络通信中断 2. 可能导致 GPU 训练/推理任务中断
MemoryUnhealthy	False	是否存在内存故障（仅支持EBC机型）	内存不可用，任务中断

EBC 机型故障检测介绍

针对于EBC弹性裸金属服务器，Node-Problem-Detector对接百度云硬件感知组件HAS-agent，新增对GPU/RDMA网卡/CPU/内存等硬件健康检测能力。

前提

1. 机器上已经安装百度云硬件感知组件HAS-agent。

GPU检测

检测维度	说明	处理方式
GPU掉卡	掉卡，无法识别GPU场景	更新 Node Condition， GPUUnhealthy：True
GPU内存	EccError、RemappedPending等场景	更新 Node Condition， GPUUnhealthy：True
GPU链路	Nvlink故障,带宽异常等场景	更新 Node Condition， GPUUnhealthy：True
XID	Nvidia GPU 的 Xid 故障	根据XID不同，处理方式不同，具体如下： 1. XID 48、62、64、74、79、95、109、122、123、124，处理方式：更新 Node Condition，GPUUnhealthy：True 2. 除上述列出的XID列表以外的XID，处理方式：仅打印事件
其他故障	温度过高、功耗异常、驱动异常等场景	更新 Node Condition， GPUUnhealthy：True

CPU/内存检测

检测维度	说明	处理方式
CPU	常见CPU Cache读写错误等场景	仅打印事件
内存	常见的不可恢复的ECC故障等场景	更新 Node Condition， MemoryUnhealthy：True
	可以恢复的ECC错误风暴、可隔离的故障场景	仅打印事件

网卡检测

检测维度	说明	处理方式
RDMA 网卡up/down	RDMA网卡频繁up/down	更新 Node Condition， NICUnhealthy：True
RDMA 网卡降速	RDMA网卡pcie降速，未达到设计速度等场景	更新 Node Condition， NICUnhealthy：True

利用监控指标实现故障报警

Node-Problem-Detector 指标介绍

Node-Problem-Detector 暴露 Prometheus 指标如下：

指标名	指标类型	指标说明
problem_counter	counter	当前时刻已经发生的故障数量
problem_gauge	gauge	当前时刻，是否存在某类型的故障

指标中包含的 LabelSet 如下：

1. instance: 故障节点
2. namespace: npd 实例部署的 namespace
3. type：故障类型，对应 Node Condition Type
4. reason：故障原因，对应 Node Condition Reason
5. region：节点所在地域

用户可根据需要基于以上指标配置 Prometheus 监控和告警；

在 CProm 中添加对 Node-Problem-Detector 的指标采集任务

在 CProm 中添加对 Node-Problem-Detector 的指标采集规则之前，需要先为 CCE 集群关联 CProm 实例，并添加指标采集任务。 操作参考：

1. CProm 实例关联 CCE 集群
2. CProm 实例中添加采集任务

添加如下指标采集任务：

```
 job_name: node-problem-detector
scrape_interval: 3s
kubernetes_sd_configs:
 - role: endpoints

relabel_configs:
 - source_labels: [ __meta_kubernetes_service_annotation_npd_prometheus_scrape ]
   action: keep
   regex: true
 - source_labels: [ __meta_kubernetes_service_annotation_npd_prometheus_scheme ]
   action: replace
   target_label: __scheme__
   regex: (https?)
 - source_labels: [ __meta_kubernetes_service_annotation_npd_prometheus_path ]
   action: replace
   target_label: __metrics_path__
   regex: (.+)
 - source_labels: [ __address__, __meta_kubernetes_service_annotation_npd_prometheus_port ]
   action: replace
   target_label: __address__
   regex: ([^:]+)(?::\d+)?;(\d+)
   replacement: $1:$2
 - source_labels: [ __meta_kubernetes_endpoint_node_name ]
   action: replace
   target_label: node
 - source_labels: [ __meta_kubernetes_namespace ]
   action: replace
   target_label: namespace
 - source_labels: [ __meta_kubernetes_service_name ]
   action: replace
   target_label: service
 - source_labels: [ __meta_kubernetes_pod_name ]
   action: replace
   target_label: pod
```

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.8.25	v1.18+	2024-03-21	组件新增对机器 gid 跳序的故障检测	--
0.8.24	v1.18+	2024-03-01	组件新增 Kubelet 故障检测 支持配置在 Metrics Labels 中禁用 bce-instance-id 优化组件配置	--
0.8.23	v1.18+	2024-01-08	适配 ubuntu22.04 操作系统 更新基础镜像，减少镜像体积, 2G -> 278M 去除对 systemctl 命令的依赖 优化对容器运行时（docker、containerd）的依赖	--
0.8.22	v1.18+	2023-12-28	MPS故障检测性能优化，支持根据mps-server启动用户执行探测 提供MPS故障检测开关，默认关闭，可按需开启 已经使用mps检测的用户升级后需要重新开启mps探测	--
0.8.21	v1.18+	2023-12-27	网卡故障检测信息源对接 has-agent 检测 GPU Xid 故障时，支持配置忽略期望的 Xid	--
0.8.20	v1.18+	2023-12-07	支持 MPS 故障检测和监控	--
0.8.19	v1.18+	2023-11-30	故障检测采集任务支持采集节点对应实例短ID	--
0.8.18	v1.18+	2023-11-23	优化 NPD 部署亲和性配置&监控配置	--
0.8.17	v1.18+	2023-11-20	优化内存故障监控时的超时时间 修正非EBC节点被部署NPD，但无HAS服务时，访问 has-agent 报错的问题	--
0.8.16	v1.18+	2023-10-17	GPU Xid 故障检测信息源对接 has-agent，取消检索内核日志 GPU 故障检测中，不支持自定义忽略 xid	--
0.8.15	v1.18+	2023-08-11	支持 CPU、内存故障检测，仅支持 EBC 机型。新增故障类型：CPUUnhealthy、MemoryUnhealthy 优化 GPU Xid 故障检测逻辑 优化 RDMA 网卡故障检测逻辑	--
0.8.14	v1.18+	2023-06-03	网卡故障检测逻辑 bugfix	--
0.8.13	v1.18+	2023-04-08	NPD 支持 GPU 故障检测、网卡故障检测，仅支持 EBC 机型。新增故障类型：GPUUnhealthy、NICUnhealthy 引入社区优化及 bugfix	--
0.8.12	v1.18+	2023-03-07	NPD 通过组件中心发布。支持以下故障检测： 1. FrequentKubeletRestart 2. FrequentContainerRuntimeRestart 3. ContainerRuntimeUnhealthy 4. CorruptDockerOverlay2 5. InodesPressure 6. KernelDeadlock 7. ReadonlyFilesystem	--

CCE Node Remedier 说明

组件介绍

CCE 节点故障自愈系统（CCE-Node-Remedier）是集群节点故障自愈组件，可以根据 CCE-Node-Problem-Detector 上报的故障自动进行节点故障维修。

当 CCE-Node-Problem-Detector（简称 NPD）组件完成节点故障检测后，会以 Condition 或者 Event 的方式上报给 Kubernetes 集群。用户在集群中安装 CCE-Node-Remedier 后，组件会以 Deployment 作为工作负载运行，实时监听每个节点的故障信息，并根据用户自定义的故障处理规则对节点发起相应的维修操作。

CCE-Node-Remedier 在发起对故障节点的维修后，会通过 CRD 向用户暴露对节点的维修进度，同时也会在节点事件中记录维修操作。

组件功能

- 提供根据节点故障自愈功能
- 支持的故障上报方式为：Node Condition
- 故障节点维修信息查询：
  - 利用 自定义对象（CRD）记录故障节点的维修进度
  - 利用 事件（Event）记录故障节点的维修操作

使用场景

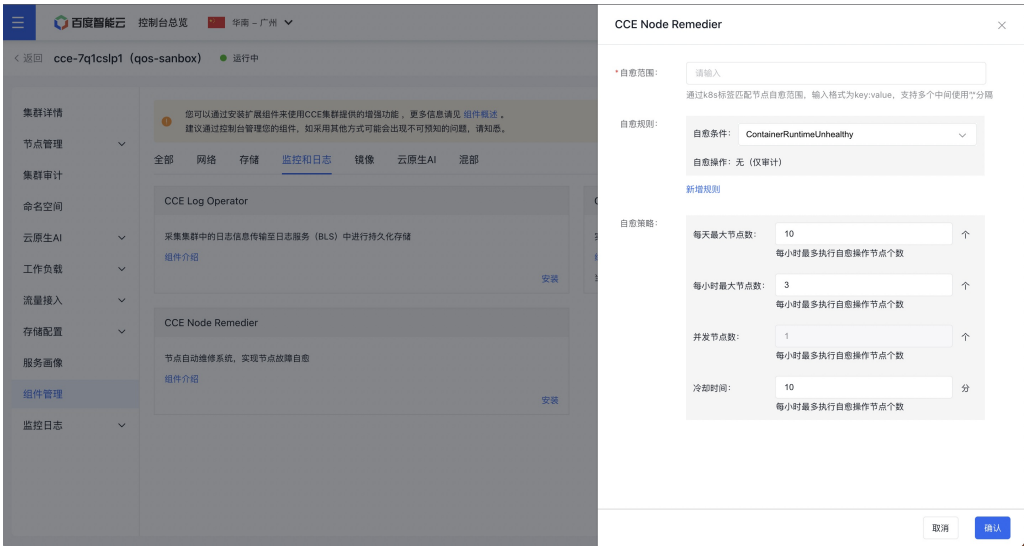
实时监听每个节点的故障信息，并根据用户自定义的故障处理规则对节点发起相应的维修操作。

使用限制

- 集群版本在 1.18.9 以上
- 已经安装了 CCE-Node-Problem-Detector。安装步骤见：CCE Node Problem Detector 说明
- 能够使用 kubectl 访问集群，操作步骤见：通过kubectl连接集群

安装组件

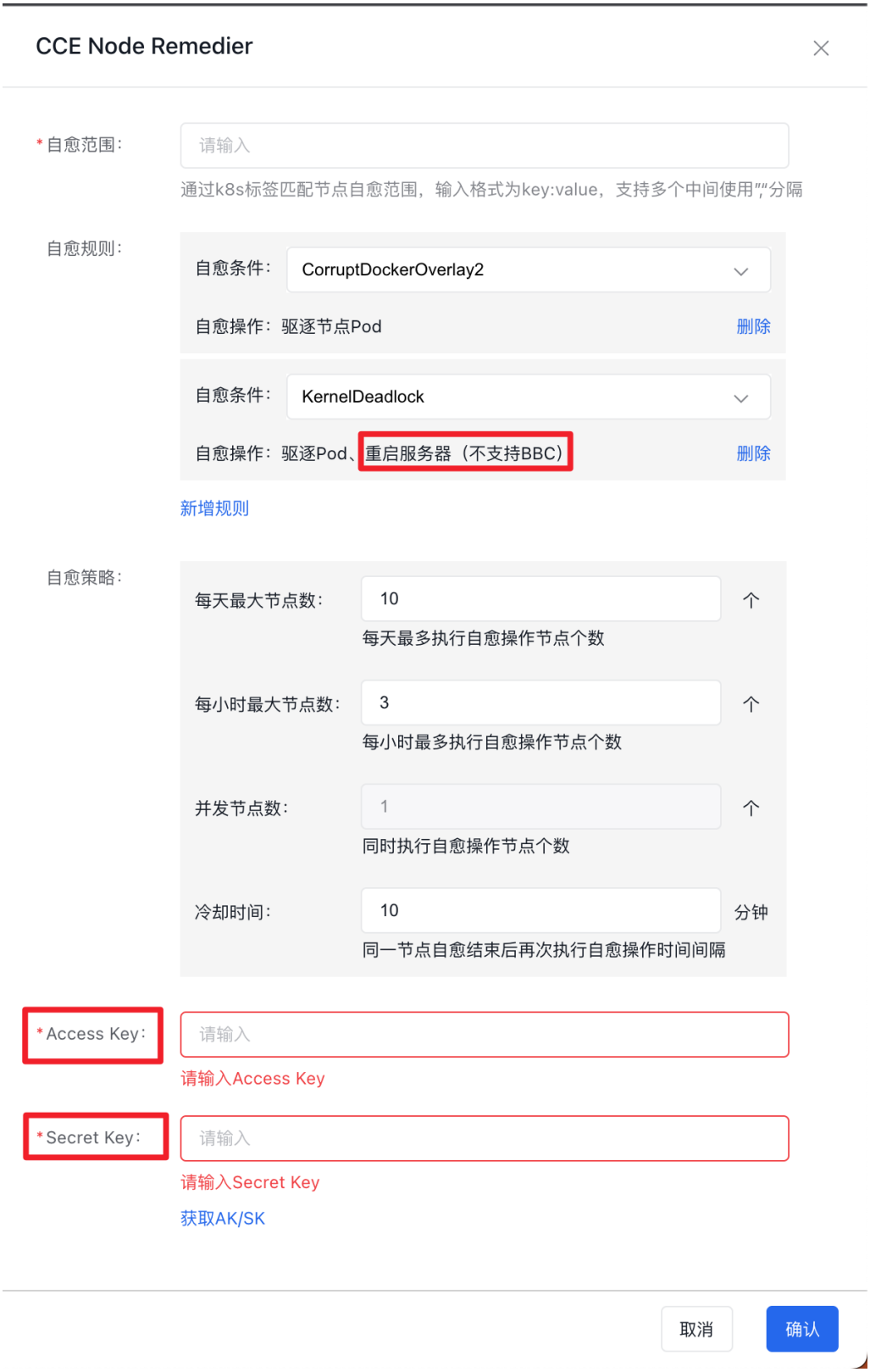
1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **组件管理** 。
6. 在组件管理列表中选择 CCE Node Remedier 组件单击“安装”。
7. 在组件配置页面中完成节点故障自愈策略配置。



8. 点击“确定”按钮完成组件的安装。

配置故障自愈策略

组件配置项如下图：



1. 自愈范围：通过k8s标签匹配节点自愈范围，输入格式为 key:value，支持多个中间使用","分隔。如：`app:nginx,component:backend`;
   - 用户需要提前在节点上添加期望的 label，或使用默认存在的 label。参考：管理节点标签
2. 自愈规则：可配置多组规则，用户可以通过点击"新增规则"添加自愈规则，每组规则包含：
   - 自愈条件：指由 CCE-Node-Problem-Detector 上报的，保存在 Node 对象 Condition 中的故障类型；
   - 自愈操作：指针对*自愈条件*，自愈系统所要执行的具体动作，目前暂不支持自定义*自愈操作*；

支持的自愈条件及自愈操作如下：

自愈条件	自愈规则
ContainerRuntimeUnhealthy	无（仅审计）
FrequentKubeletRestart	无（仅审计）
FrequentContainerRuntimeRestart	无（仅审计）
InodesPressure	无（仅审计）
CorruptDockerOverlay2	驱逐节点Pod
KernelDeadlock	驱逐Pod、重启服务器（不支持BBC）
ReadonlyFilesystem	驱逐Pod、重启服务器（不支持BBC）
GPUUnhealthy	仅冻结（Cordon）
NICUnhealthy	仅冻结（Cordon）

3. 自愈策略：
    - 每天最大节点数：每天最多执行自愈操作节点个数，数据类型为整数，默认为 10；
    - 每小时最大节点数：每小时最多执行自愈操作节点个数，数据类型为整数，默认为 3；
    - 并发节点数：同时执行自愈操作节点个数，数据类型为整数，默认为 1，**暂不支持修改**；
    - 冷却时间：同一节点自愈结束后再次执行自愈操作时间间隔，数据类型为整数，单位为 *分钟*，默认为 10；
4. Access/Secret Key：当用户选择包含 **重启服务器** 的自愈条件时，需要填写用户百度云账号的 AccessKey 和 SecretKey
    - 了解如何创建、查看和下载 Access Key(AK) 和 Secret Key(SK), 请参考 管理您的AK/SK.

查看节点故障状态及故障维修信息

查看节点故障状态

节点故障状态由 CCE-Node-Problem-Detector 组件上报，故障详情可参考：查看集群健康检查状态

查看节点故障维修信息

CCE-Node-Remedier 在集群中安装了三种自定义资源（CustomResourceDefinition，CRD），名称及作用如下：

1. `NodeRemedier`：简写为 `nr`，用于保存节点故障维修策略，对象名称为 `default-remedier`, 查询命令为：

```
kubectl -n cce-node-remedier get noderemediers default-remedier -o yaml

kubectl -n cce-node-remedier get nr default-remedier -o yaml
```

2. `MachineRemediationTemplate`: 简写为 `mrtpl`，保存节点故障维修的步骤模板，对象名称为 `default-remediation-template`, 查询命令为：

```
kubectl -n cce-node-remedier get machineremediationtemplates default-remediation-template -o yaml

kubectl -n cce-node-remedier get mrtpl default-remediation-template -o yaml
```

3. `RemedyMachine`: 简写为 `rm`，保存故障节点的具体维修故障和维修详情，每个对象对应一个故障节点，查询命令为：

```
kubectl -n cce-node-remedier get remedymachines

kubectl -n cce-node-remedier get rms
```

示例的 `RemedyMachine` 的 yaml 为：

```
  apiVersion: apps.baidu.com/v1
kind: RemedyMachine
metadata:
  creationTimestamp: "2023-03-30T01:35:00Z"
  generation: 1
  name: cce-node-remedier-default-remedier-cce-covdy1t9-192.168.96.52
  namespace: cce-node-remedier
  ownerReferences:
  - apiVersion: apps.baidu.com/v1
    blockOwnerDeletion: true
    controller: true
    kind: NodeRemedier
    name: default-remedier
    uid: 0fe02c17-b344-4c0b-8b4d-f7ed1d21e064
  resourceVersion: "5442036"
  uid: 9c15873d-1ffb-4a18-b063-4a2e55328d01
spec:
  clusterName: cce-covdy1t9
  nodeName: 192.168.96.52
  nodeProvider: baiducloud
  paused: false
  reconcileSteps:
  - CordonNode
  - DrainNode
  - RebootNode
status:
  healthy: true
  message: Check step time intervals, latest step is RebootNode, the time internal
    is left 885 second, please wait
  reconcileSteps:
    CordonNode:
      finish: true
      finishedTime: "2023-03-30T01:35:00Z"
      retryCount: 1
      startTime: "2023-03-30T01:35:00Z"
      successful: true
    DrainNode:
      finish: true
      finishedTime: "2023-03-30T01:35:45Z"
      retryCount: 1
      startTime: "2023-03-30T01:35:45Z"
      successful: true
    RebootNode:
      costSeconds: 27
      finish: true
      finishedTime: "2023-03-30T01:36:41Z"
      retryCount: 1
      startTime: "2023-03-30T01:36:14Z"
      successful: true
```

如上查询的到的 `RemedyMachine`，

1. `spec.reconcileSteps`：保存针对该节点具体的需要执行的自愈步骤，分别为：
   - `CordonNode`： 冻结节点，防止新的 Pod 被调度到该节点；
   - `DrainNode`：执行节点排水，驱逐该节点上 Pod，目前不会驱逐由 DaemonSet 管理的 Pod；
   - `RebootNode`：重启节点，重启该 Worker 节点所对应的 BCC 节点，暂不支持重启 BBC 节点；
2. `status`: 保存针对该节点的故障自愈状态
   - `healthy`：表示目前该节点是否健康；
   - `reconcileSteps`：针对该节点的各维修步骤的执行结果，每个元素包含：
     - `finish`： 该步骤是否执行结束
     - `successful`： 该步骤是否执行成功
     - `retryCount`：该步骤的重试次数

- `startTime`：该步骤的开始执行时间
- `finishedTime`：该步骤的结束时间

查看节点故障维修事件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **节点管理 > Worker**，进入节点管理界面。
6. 点击 **节点名称** 查看节点详情：

7. 点击 **事件**，可进入事件中心查看该节点相关的事件：

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.1.0	CCE v1.18 及以上	2023.04.12	首次上线	-
0.1.5	CCE v1.18 及以上	2024.01.18	新增三个自愈条件：CPUUnhealthyExt、MemoryUnhealthyExt 和 MainboardUnhealthyExt	-
0.1.6	CCE v1.18 及以上	2024.01.24	新增两个自愈操作：重启 Kubelet 和 重启容器运行时	-

CCE Descheduler 说明

组件介绍

CCE Descheduler 是容器引擎 CCE 在 社区版 Descheduler 的基础上，新增实现了 “基于节点 CPU/Memory 真实利用率” 进行 Pod 重调度的组件。“节点真实资源利用率”依赖 Prometheus 进行节点资源利用率信息的采集，支持 用户自建 Prometheus 和 百度云托管 Prometheus 两种数据源。

功能介绍

在 Kubernetes 中，kube-scheduler 负责对 Pending 状态的 Pod 进行调度，将 Pod 和具体的 Node 绑定。调度器对某个 Pod 是否可以被调度的决策，是由一组可配置的策略决定，被称为 predicates 和 priorities。调度器的决策是受到待调度的 Pod 出现时的集群资源视图影响的，这种一次性的调度存在局限：Node 节点数量、标签、污点、容忍等的变动可能会导致已经被调度过的 Pod 不是最优调度。我们可能需要将某些已经处于运行状态的 Pod 迁移到其他节点上。

社区版本的 descheduler 中提供的节点利用率相关的策略（如 LowNodeUtilization、HighNodeUtilization）是利用 Pod 的 Reqeust 和 Limit 数据，而不是节点的真实利用率，例如内存型和计算型任务分配的资源一致，但是在运行过程中的实际消耗量有巨大差异。

CCE Descheduler 新增了 “基于节点真实利用率” 的重调度策略。通过 Prometheus 获取各节点的负载统计信息，并根据用户设置的阈值对高负载节点上的 Pod 发起驱逐。

使用场景

基于节点 CPU/Memory 真实利用率进行 Pod 重调度。

注意事项

- 默认不驱逐 关键 Pod（Critical Pods），即 priorityClassName 被配置为 system-cluster-critical 或者 system-node-critical；
- 不驱逐不受工作负载（Deployment、StatefulSet、Job 等）管理的 Pod，因为这类 Pod 被驱逐后不会被重调度；
- 默认不驱逐 DaemonSet 管理的 Pod；
- 默认不驱逐使用本地存储的 Pod；

- 默认不驱逐挂载 PVC 的 Pod；
- 默认不驱逐系统组件的 Pod；
- 可以使用 PodDisruptionBudget 保护 Pod 不被驱逐，例如用户可以配置控制工作负载下不可用副本的数量或比例，例如：

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: zookeeper
```

- 驱逐属于高危操作，请注意节点亲和性、污点相关配置，以及 Pod 本身对节点的要求选择，防止出现驱逐后无节点可调度情况；
- 除以上限制条件外，默认全部 Pod 都可以被驱逐；

安装组件

依赖部署

CCE-Descheduler 组件依赖于 Node 当前和过去一段时间的真实负载情况来进行调度决策，需要通过 Prometheus 等监控组件获取系统 Node 真实负载信息。在使用 CCE-Descheduler 组件之前，您可以采用 百度云托管 Prometheus 或自建 Prometheus 监控。

使用百度云托管 Prometheus

使用百度云托管 Prometheus 作为监控数据源的情况下，需要开通 创建 Prometheus 实例、关联 CCE 集群、指标汇聚规则。相关操作可参考：

1. 创建 Prometheus 实例
2. 关联 CCE 集群
3. 配置指标采集任务
4. 配置指标汇聚规则

在完成托管 Prometheus 实例创建、关联 CCE 集群后，托管 Prometheus 会自动完成 “指标采集任务” 的配置，主要包含了对集群控制面组件（如 kube-apiserver、kube-scheduler 等）的监控，也包含了 Descheduler 依赖的对于 cAdvisor 和 NodeExporter 指标的采集。所以用户无需自行配置指标采集任务。

在配置指标汇聚规则阶段，用户要添加如下规则：

```
spec:
groups:
  - name: machine_cpu_mem_usage_active
    interval: 30s
    rules:
      - record: machine_memory_usage_active
        expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: machine_memory_usage_1m
    interval: 1m
    rules:
      - record: machine_memory_usage_5m
        expr: 'avg_over_time(machine_memory_usage_active[5m])'
  - name: machine_cpu_usage_1m
    interval: 1m
    rules:
      - record: machine_cpu_usage_5m
        expr: >-
          100 - (avg by (instance, clusterID)
          (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
  - name: container_cpu_usage_10m
    interval: 1m
    rules:
      - record: container_cpu_usage_total_5m
        expr: >-
          sum by(namespace, pod,
          clusterID)(rate(container_cpu_usage_seconds_total{pod!='',image=''}[5m]))
          * 1000
  - name: container_memory_working_set_bytes_pod
    interval: 1m
    rules:
      - record: container_memory_working_set_bytes_by_pod
        expr: 'container_memory_working_set_bytes{pod!='''',image=''''}'
```

该规则实现了对 CCE-Descheduler 依赖的 `machine_cpu_usage_5m`、`machine_memory_usage_5m`、`container_cpu_usage_total_5m`、`container_memory_working_set_bytes_by_pod` 等指标的自动汇聚计算。

使用自建 Prometheus

使用自建 Prometheus 作为监控数据源的情况下，用户需要自行部署两个组件：

1. NodeExporter
2. Prometheus

可以参考组件官方文档完成组件的部署。

在完成组件部署后，用户需要在 Prometheus 添加对 cAdvisor 和 NodeExporter 的监控指标采任务以及指标聚合规则的配置，具体配置可参考：

```
scrape_configs:
- job_name: "kubernetes-cadvisor"
  # Default to scraping over https. If required, just disable this or change to
  # `http`.
  scheme: https

  # Starting Kubernetes 1.7.3 the cAdvisor metrics are under /metrics/cadvisor.
  # Kubernetes CIS Benchmark recommends against enabling the insecure HTTP
  # servers of Kubernetes, therefore the cAdvisor metrics on the secure handler
  # are used.
  metrics_path: /metrics/cadvisor

  # This TLS & authorization config is used to connect to the actual scrape
  # endpoints for cluster components. This is separate to discovery auth
  # configuration because discovery & scraping are two separate concerns in
  # Prometheus. The discovery auth config is automatic if Prometheus runs inside
  # the cluster. Otherwise, more config options have to be provided within the
  # <kubernetes_sd_config>.
  tls_config:
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    # If your node certificates are self-signed or use a different CA to the
    # master CA, then disable certificate verification below. Note that
    # certificate verification is an integral part of a secure infrastructure
    # so this should only be disabled in a controlled environment. You can
    # disable certificate verification by uncommenting the line below.
    #
    insecure_skip_verify: true
  authorization:
    credentials_file: /var/run/secrets/kubernetes.io/serviceaccount/token

  kubernetes_sd_configs:
    - role: node

  relabel_configs:
    - action: labelmap
      regex: __meta_kubernetes_node_label_(.+)

- job_name: 'node-exporter'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
    - source_labels: [__meta_kubernetes_pod_name]
      regex: 'node-exporter-(.+)'
      action: keep
```

在使用自建 Prometheus 的情况下，配置指标聚合规则和使用托管集群类似，区别在于规则中不需要考虑 `clusterID` 维度，该维度由托管 Prometheus 添加。所以聚合规则可参考：

```
spec:
groups:
  - name: machine_cpu_mem_usage_active
    interval: 30s
    rules:
      - record: machine_memory_usage_active
        expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: machine_memory_usage_1m
    interval: 1m
    rules:
      - record: machine_memory_usage_5m
        expr: 'avg_over_time(machine_memory_usage_active[5m])'
  - name: machine_cpu_usage_1m
    interval: 1m
    rules:
      - record: machine_cpu_usage_5m
        expr: >-
          100 - (avg by (instance)
          (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
  - name: container_cpu_usage_10m
    interval: 1m
    rules:
      - record: container_cpu_usage_total_5m
        expr: >-
          sum by(namespace, pod)(rate(container_cpu_usage_seconds_total{pod!='',image=''}[5m])) * 1000
  - name: container_memory_working_set_bytes_pod
    interval: 1m
    rules:
      - record: container_memory_working_set_bytes_by_pod
        expr: 'container_memory_working_set_bytes{pod!='''',image=''''}'
```

安装 CCE-Descheduler

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品导览>容器>容器引擎**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**运维与管理>组件管理**”。
6. 在组件管理列表中选择 **CCE Descheduler** 组件单击“安装”。
7. 在组件配置页面中完成数据源选择及重调度策略配置,点击“确定”按钮完成组件的安装。

用户可以根据需求，选择 **托管 Prometheus** 或 **自建 Prometheus** 作为数据源。 当选择托管 Promehteus 时，用户可以在下拉框选择集群所关联的 Prometheus 实例：

当选择自建 Prometheus 时，用户需要手动输入**在集群内可访问**的地址：

8. 其他参数包括：
    1. 利用率阈值：当节点的 CPU 或 内存 利用率达到该阈值时，将发起对节点上 Pod 的驱逐；
    2. 目标利用率：当发起对节点上 Pod 驱逐后，当达到目标利用率后，停止对 Pod 的驱逐；
    3. 重调度挂载 PVC Pod：默认不会对挂载 PVC 的 Pod 发起驱逐
    4. 重调度挂载本地存储的 Pod：默认不会对挂载本地存储的 Pod 发起驱逐
9. 完成参数配置后，点击 “确认”，即可完成组件安装。

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.24.2	CCE/v1.16+	2023.06.21	首次上线	-
0.24.3	CCE/v1.16+	2023.12.15	修复内存泄漏问题	-

CCE QoS Agent 说明

组件介绍

`cce-qos-agent` 基于 BaiduLinux 的能力，为容器提供基于 Kubernetes 原生 QoS 的服务质量保障、容器隔离与加固能力，同时提升资源利用率。

> 注意：QoS 相关能力仅支持在使用 BaiduLinux3 作为操作系统的节点上使用，若您的节点使用其他操作系统，相关能力无法生效。

组件功能

- 配置节点资源水位，当整机资源用量超过配置的资源水位时，将触发对低优容器的压制或驱逐；

- 容器 QoS 增强，包括如下表所示能力

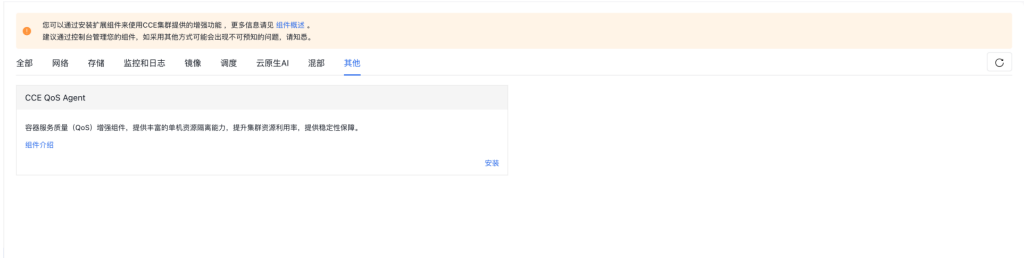
功能	说明
CPU QoS 增强	支持 CPU 优先级配置，可以通过对工作负载设置优先级，保证高优先级业务在发生资源竞争时的资源供给量，并压制低优先级业务。
内存 QoS 增强	支持内存背景回收、OOMKill 优先级等，提升整体内存性能。
网络 QoS 增强	支持网络带宽优先级配置，出入方向限速等，灵活限制容器对网络的使用。

限制说明

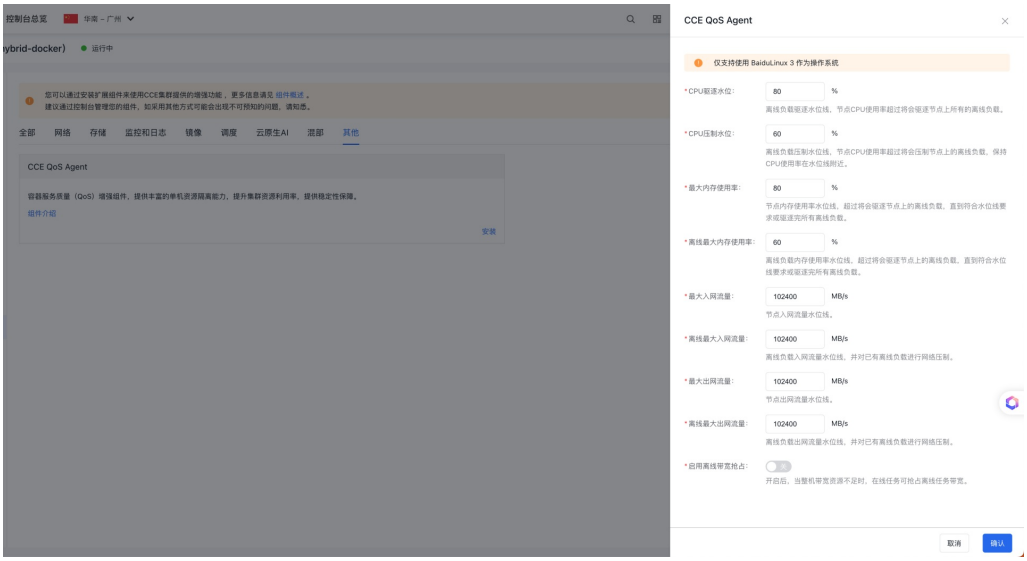
- 集群版本在 1.18.9 以上
- BaiduLinux3 作为节点操作系统
- 能够使用 kubectl 访问集群，操作步骤见：通过kubectl连接集群

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **组件管理** 。
6. 在组件管理列表中选择 CCE QoS Agent 组件单击“安装”。



7. 在组件配置页面中完成节点资源水位配置。



8. 点击“确定”按钮完成组件的安装。

配置节点资源水位

配置项含义如下：

- CPU驱逐水位线：当整机CPU使用量达到销毁水位线时，驱逐全部的低优服务保障高优服务，默认值 80%；
- CPU压制水位线：当整机CPU使用量达到压制水位线时，压制全部的低优服务保障高优服务，默认值 60%；

- 内存整机最大水位线：当整机内存超过最大内存使用量时，驱逐低优服务保障高优服务，默认值 80%；
- 内存低优最大水位线：当低优内存超过低优内存使用量时，驱逐低优服务保障高优服务，默认值 60%；
- 整机最大入网流量：当整机入网流量超过最大入网流量，禁止将离线应用调度到节点；
- 离线最大入网流量：当离线入网流量超过最大入网流量，将对已有的离线负载进行网络压制；
- 整机最大出网流量：当整机入网流量超过最大入网流量，禁止将离线应用调度到节点；
- 离线最大出网流量：当离线入网流量超过最大入网流量，将对已有的离线负载进行网络压制；
- 是否开启网络带宽抢占：当整机带宽不足时，在线任务将抢占离线任务带宽，默认关闭；

使资源水位对节点生效

完成 `cce-qos-agent` 组件的安装后，需要为节点添加标签（label）`cce.baidu.com/sla-pool="default"`，使资源水位对节点生效。

以节点 `10.0.3.13` 为例，在终端执行以下命令：

```
kubectl label node 10.0.3.13 cce.baidu.com/sla-pool="default"
```

配置 Pod QoS

用户在安装了单机引擎之后，可以通过 Pod 的 label 扩展Pod的增强隔离加固能力。 支持的 配置项如下表：

能力	能力描述	Pod Label	参数说明	单位	默认值
CPU 优先级	配置 CPU 优先级。	cpu.priority	支持将 Pod 配置为在线、离线任务	-	无
设置memcg背景回收能力	背景回收能力：控制Pod级别的异步回收cache动作，当Pod的cache触发到高水位线（memory.high_wmark_distance）时，会开始回收cache，直到cache达到低水位线（memory.low_wmark_distance）	memory.background_reclaim	是否开启内存背景回收	bool	0
		memory.low_wmark_distance	内存背景回收低水位线	byte	0
		memory.high_wmark_distance	内存背景回收高水位线	byte	0
设置OOM时，Pod被kill的优先级值	该功能在进行OOM操作时，会首先判定cgroup的优先级，选择低优先级的cgroup进行OOM操作，cgroup的优先级值（memory.oom_kill_prioirty）越大，优先级越低。	memory.oom_kill_priority	设置oom kill优先级，优先级越高优先杀死	int	5000
控制发生OOM时，cgroup内的进程组的kill模式	控制发生OOM时，cgroup内的进程组的kill模式：0: 不杀死进程组内所有的进程 1: 杀死进程组内所有的进程	memory.kill_mode	控制OOM kill模式，是否杀死进程组所有进程。	bool	0
设置网络优先级	设置 Pod 网络优先级	net.priority	可选值: besteffort	bool	0

配置 CPU QoS

支持的 CPU 优先级及与 Kubernetes QoS 的对应关系如下：

CPU 优先级	K8S QoS	说明
sensitive	Guaranteed	绑核，CPU 必须是 1000m 的整数倍
stable	Guaranteed/Burstable	实现更好的资源弹性和更灵活的资源调整能力，要求至少需要填写 requests
batch	BestEffort	离线任务；相对 BestEffort，CPU 使用比例是 BE 的两倍
besteffort	BestEffort	离线任务

场景一：离线任务支持多级 CPU 优先级，通过 CPU 比例划分，无抢占关系

场景描述：

1. 部署两个离线 CPU 任务，CPU 优先级分别为 batch 和 besteffort。
2. 优先级为 batch 的任务的 CPU 运行时间约为优先级为 besteffort 任务的两倍。

操作步骤：

1. 部署 离线 CPU 任务，以下yaml 包含两个 cpu stress 任务，分别配置优先级为 batch 和 besteffort

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: stress-batch
spec:
 selector:
   matchLabels:
     app: stress-batch
 template:
   metadata:
     labels:
       app: stress-batch
       cpu.priority: batch
   spec:
     nodeSelector:
       kubernetes.io/hostname: 10.0.3.13
     containers:
     - name: stress
       image: registry.baidubce.com/public-tools/stress-ng:latest
       command:
       - "stress-ng"
       args:
       - --cpu
       - "0"
       - --cpu-method
       - all
       - -t
       - 1h
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: stress-besteffort
spec:
 selector:
   matchLabels:
     app: stress-besteffort
 template:
   metadata:
     labels:
       app: stress-besteffort
       cpu.priority: besteffort
   spec:
     nodeSelector:
       kubernetes.io/hostname: 10.0.3.13
     containers:
     - name: stress
       image: registry.baidubce.com/public-tools/stress-ng:latest
       command:
       - "stress-ng"
       args:
       - --cpu
       - "0"
       - --cpu-method
       - all
       - -t
       - 1h
```

2. 部署后在 Pod 所在节点节点计算两个容器在过去 1s 的运行时间

```
##### !/bin/bash
batchPodID="eccb11ba-b875-4614-ac07-2e811397239a"
batchContainerID="6e677ceb30c28438bf32d1191d74e7fa0131124ea08fe8647950e07c7a34bef5"

bePodID="8339d083-5aae-48bd-a2cf-9c91a777dc53"
beContainerID="6c10640a79479f8d6606d3823a4666cffb225913990b6aab1bfdf8bcf3a6cf47"

batch_start=`cat /sys/fs/cgroup/cpu/kubepods/besteffort/pod${batchPodID}/${batchContainerID}/cpuacct.bt_usage`
be_start=`cat /sys/fs/cgroup/cpu/kubepods/besteffort/pod${bePodID}/${beContainerID}/cpuacct.bt_usage`
sleep 1
batch_end=`cat /sys/fs/cgroup/cpu/kubepods/besteffort/pod${batchPodID}/${batchContainerID}/cpuacct.bt_usage`
be_end=`cat /sys/fs/cgroup/cpu/kubepods/besteffort/pod${bePodID}/${beContainerID}/cpuacct.bt_usage`

expr "${batch_end}" - "${batch_start}"
expr "${be_end}" - "${be_start}"
```

得到计算结果：

```
[root@cce-ig-1puew4d5-ia799q1s-35782kzn-1 tmp]# bash cal.sh
2583203766
1464301248
```

可以看到 batch 优先级任务所占用的 CPU 时间约为 besteffort 优先级任务的两倍。

场景二：离线任务支持多级 CPU 优先级，通过 CPU 比例划分，无抢占关系

场景描述：

1. 在 case1 的部署离线任务的基础上，部署在线任务
2. 在线任务完成对离线任务的 CPU 抢占。

操作步骤：

1. 部署在线 CPU 任务，优先级配置为 stable

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stress-stable
spec:
  selector:
    matchLabels:
      app: stress-stable
  template:
    metadata:
      labels:
        app: stress-stable
        cpu.priority: stable
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.13
      containers:
      - name: stress
        image: registry.baidubce.com/public-tools/stress-ng:latest
        command:
        - "stress-ng"
        args:
        - --cpu
        - "0"
        - --cpu-method
        - all
        - -t
        - 1h
        resources:
          requests:
            cpu: 1
            memory: 2Gi
```

2. 再次执行 case 1 中的脚本，发现离线任务几乎未执行

```
[root@cce-ig-1puew4d5-ia799q1s-35782kzn-1 tmp]# bash cal.sh
0
0
```

可以看到，在线任务可以实现对离线任务的绝对抢占，当整机资源不足时，将保证在线任务的运行，而离线任务将无法执行。

场景三：部署在线 CPU 任务，sensitive 优先级实现 CPU 绑核

场景描述：

1. 部署 CPU 优先级为 sensitive 的任务，实现 cpu 绑核

操作步骤：

1. 部署任务，配置 CPU 优先级为 sensitive

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stress-sensitive
spec:
  selector:
    matchLabels:
      app: stress-sensitive
  template:
    metadata:
      labels:
        app: stress-sensitive
        cpu.priority: sensitive
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.13
      containers:
      - name: stress
        image: registry.baidubce.com/public-tools/stress-ng:latest
        command:
        - "stress-ng"
        args:
        - --cpu
        - "0"
        - --cpu-method
        - all
        - -t
        - 1h
        resources:
          requests:
            cpu: 1
            memory: 2Gi
          limits:
            cpu: 1
            memory: 2Gi
```

2. 通过 cgroups 查看绑核结果，在 pod 所在节点执行如下脚本

```
##### !/bin/bash
sensitivePodID="f203ffd5-8618-4568-8547-836db6a906f2"
sensitiveContainerID="2c551bfde9d0f5e9f46f049ac8acf856567bd9441dbdedac5fec1151f8a93721"

cat /sys/fs/cgroup/cpuset/kubepods/pod${sensitivePodID}/${sensitiveContainerID}
```

得到结果：

```
[root@cce-0l15wup2-5nsjbb48 2c551bfde9d0f5e9f46f049ac8acf856567bd9441dbdedac5fec1151f8a93721]# pwd
/sys/fs/cgroup/cpuset/kubepods/podf203ffd5-8618-4568-8547-836db6a906f2/2c551bfde9d0f5e9f46f049ac8acf856567bd9441dbdedac5fec1151f8a93721
[root@cce-0l15wup2-5nsjbb48 2c551bfde9d0f5e9f46f049ac8acf856567bd9441dbdedac5fec1151f8a93721]# cat cpuset.cpus
7
```

可以看到，该容器已经绑定了 7 号 cpu。

配置网络 QoS

开启网络优先级功能

在组件中心，修改单机隔离引擎配置，开启“网络优先级功能”

场景一：在线可以抢占离线带宽 场景描述：

1. 在同一节点，先后部署低网络优先级任务，和默认网络优先级任务，观察任务的带宽变化，预期低网络优先级的任务的带宽可以被抢占

操作步骤：

1. 部署 server 服务

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: net-server-1
  labels:
    app: net-server-1
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: net-server-1
  template:
    metadata:
      labels:
        app: net-server-1
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.14
      containers:
        - name: server
          image: registry.baidubce.com/public-tools/iperf3:latest
          args: [ "-s", "-p", "5201"]
          ports:
            - name: http
              containerPort: 5201
              hostPort: 5201
              protocol: TCP
          resources:
            limits:
              cpu: 4
              memory: 2Gi
            requests:
              cpu: 4
---
apiVersion: v1
kind: Service
metadata:
  name: net-server-1
spec:
  selector:
    app: net-server-1
  ports:
  - port: 5201
    targetPort: 5201

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: net-server-2
  labels:
    app: net-server-2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: net-server-2
  template:
    metadata:
      labels:
        app: net-server-2
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.15
      containers:
        - name: server
          image: registry.baidubce.com/public-tools/iperf3:latest
          args: [ "-s", "-p", "5201"]
          ports:
            - name: http
              containerPort: 5201
              hostPort: 5201
              protocol: TCP
          resources:
            limits:
              cpu: 4
              memory: 2Gi
            requests:
              cpu: 4
```

```
      cpu: 4
---
apiVersion: v1
kind: Service
metadata:
  name: net-server-2
spec:
  selector:
    app: net-server-2
  ports:
  - port: 5201
    targetPort: 5201
```

2. 部署任务，配置网络优先级为 besteffort

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: offline-net-client
  labels:
    app: offline-net-client
spec:
  replicas: 1
  selector:
    matchLabels:
      app: offline-net-client
  template:
    metadata:
      labels:
        app: offline-net-client
        net.priority: besteffort
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.13
      containers:
        - name: main
          image: registry.baidubce.com/public-tools/iperf3:latest
          args: [ "-c", "net-server-1.default.svc", "-p", "5201", "-b", "2000Mb" ,"-t","3600"]
```

观察任务带宽：

```
→  case1-在线可以抢占离线 kubectl logs -f offline-net-client-69ffd5d65c-2hf7r
Connecting to host net-server.default.svc, port 5201
[  5] local 172.16.5.45 port 49196 connected to 192.168.190.167 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   238 MBytes  2.00 Gbits/sec    0    479 KBytes
[  5]   1.00-2.00   sec   238 MBytes  2.00 Gbits/sec    0    510 KBytes
[  5]   2.00-3.00   sec   238 MBytes  2.00 Gbits/sec    0    510 KBytes
[  5]   3.00-4.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   4.00-5.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   5.00-6.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   6.00-7.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   7.00-8.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   8.00-9.00   sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]   9.00-10.00  sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]  10.00-11.00  sec   238 MBytes  2.00 Gbits/sec    0    549 KBytes
[  5]  11.00-12.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  12.00-13.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  13.00-14.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  14.00-15.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  15.00-16.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  16.00-17.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  17.00-18.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  18.00-19.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  19.00-20.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  20.00-21.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  21.00-22.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  22.00-23.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  23.00-24.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  24.00-25.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  25.00-26.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  26.00-27.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  27.00-28.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  28.00-29.00  sec   238 MBytes  2.00 Gbits/sec    0    935 KBytes
[  5]  29.00-30.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
[  5]  30.00-31.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
[  5]  31.00-32.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
[  5]  32.00-33.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
[  5]  33.00-34.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
[  5]  34.00-35.00  sec   238 MBytes  2.00 Gbits/sec    0    964 KBytes
```

3. 部署网络优先级为在线的任务，观察在线和离线任务带宽

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: online-net-client
  labels:
    app: online-net-client
spec:
  replicas: 1
  selector:
    matchLabels:
      app: online-net-client
  template:
    metadata:
      labels:
        app: online-net-client
    spec:
      nodeSelector:
        kubernetes.io/hostname: 10.0.3.13
      containers:
        - name: main
          image: registry.baidubce.com/public-tools/iperf3:latest
          args: [ "-c", "net-server-2.default.svc", "-p", "5201", "-b", "2000Mb" ,"-t","3600"]
```

观察在线任务带宽：

```
→  case1-在线可以抢占离线 kubectl logs -f online-net-client-54669d97f8-nw6jq
Connecting to host net-server-1.default.svc, port 5201
[  5] local 172.16.5.48 port 16246 connected to 192.168.63.223 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   238 MBytes  2.00 Gbits/sec    0    298 KBytes
[  5]   1.00-2.00   sec   238 MBytes  2.00 Gbits/sec    0    373 KBytes
[  5]   2.00-3.00   sec   238 MBytes  2.00 Gbits/sec  447   28.3 KBytes
[  5]   3.00-4.00   sec   238 MBytes  2.00 Gbits/sec  358    260 KBytes
[  5]   4.00-5.00   sec   238 MBytes  2.00 Gbits/sec  289    277 KBytes
[  5]   5.00-6.00   sec   238 MBytes  2.00 Gbits/sec  775    242 KBytes
[  5]   6.00-7.00   sec   238 MBytes  2.00 Gbits/sec  666    242 KBytes
[  5]   7.00-8.00   sec   238 MBytes  2.00 Gbits/sec  347    266 KBytes
[  5]   8.00-9.00   sec   233 MBytes  1.95 Gbits/sec  586    127 KBytes
[  5]   9.00-10.00  sec   244 MBytes  2.05 Gbits/sec  3371    274 KBytes
[  5]  10.00-11.00  sec   238 MBytes  2.00 Gbits/sec    0    274 KBytes
[  5]  11.00-12.00  sec   238 MBytes  2.00 Gbits/sec    0    296 KBytes
[  5]  12.00-13.00  sec   238 MBytes  2.00 Gbits/sec  385    276 KBytes
[  5]  13.00-14.00  sec   238 MBytes  2.00 Gbits/sec  644    269 KBytes
[  5]  14.00-15.00  sec   238 MBytes  2.00 Gbits/sec  845    365 KBytes
[  5]  15.00-16.00  sec   238 MBytes  2.00 Gbits/sec  884    269 KBytes
[  5]  16.00-17.00  sec   238 MBytes  2.00 Gbits/sec  3607    246 KBytes
[  5]  17.00-18.00  sec   238 MBytes  2.00 Gbits/sec    0    272 KBytes
[  5]  18.00-19.00  sec   238 MBytes  2.00 Gbits/sec    0    272 KBytes
[  5]  19.00-20.00  sec   238 MBytes  2.00 Gbits/sec    0    283 KBytes
[  5]  20.00-21.00  sec   238 MBytes  2.00 Gbits/sec    0    284 KBytes
[  5]  21.00-22.00  sec   238 MBytes  2.00 Gbits/sec  424    273 KBytes
[  5]  22.00-23.00  sec   238 MBytes  2.00 Gbits/sec  819    257 KBytes
[  5]  23.00-24.00  sec   238 MBytes  2.00 Gbits/sec  181    272 KBytes
[  5]  24.00-25.00  sec   238 MBytes  2.00 Gbits/sec  866    189 KBytes
[  5]  25.00-26.00  sec   238 MBytes  2.00 Gbits/sec  379    266 KBytes
[  5]  26.00-27.00  sec   238 MBytes  2.00 Gbits/sec  391    270 KBytes
[  5]  27.00-28.00  sec   238 MBytes  2.00 Gbits/sec  964    246 KBytes
[  5]  28.00-29.00  sec   238 MBytes  2.00 Gbits/sec  883    274 KBytes
[  5]  29.00-30.00  sec   238 MBytes  2.00 Gbits/sec  999    266 KBytes
[  5]  30.00-31.00  sec   238 MBytes  2.00 Gbits/sec  793    262 KBytes
```

观察离线任务带宽：

```
[  5] 391.00-392.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 392.00-393.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 393.00-394.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 394.00-395.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 395.00-396.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 396.00-397.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 397.00-398.00 sec   238 MBytes  2.00 Gbits/sec    0   2.04 MBytes
[  5] 398.00-399.00 sec   238 MBytes  2.00 Gbits/sec    0   3.10 MBytes
[  5] 399.00-400.00 sec   238 MBytes  2.00 Gbits/sec    0   3.10 MBytes
[  5] 400.00-401.00 sec   226 MBytes  1.90 Gbits/sec  3647   31.1 KBytes
[  5] 401.00-402.00 sec  31.0 MBytes   260 Mbits/sec  1645    245 KBytes
[  5] 402.00-403.00 sec   164 MBytes  1.38 Gbits/sec  664   2.92 MBytes
[  5] 403.00-404.00 sec   139 MBytes  1.17 Gbits/sec  5215    130 KBytes
[  5] 404.00-405.00 sec   113 MBytes   947 Mbits/sec  3399   79.2 KBytes
[  5] 405.00-406.00 sec  83.2 MBytes   698 Mbits/sec  1108   94.7 KBytes
[  5] 406.00-407.00 sec   202 MBytes  1.69 Gbits/sec  10922   60.8 KBytes
[  5] 407.00-408.00 sec  46.8 MBytes   392 Mbits/sec  805    170 KBytes
[  5] 408.00-409.00 sec  9.88 MBytes  82.8 Mbits/sec    0    170 KBytes
[  5] 409.00-410.00 sec   158 MBytes  1.33 Gbits/sec  126    441 KBytes
[  5] 410.00-411.00 sec   166 MBytes  1.39 Gbits/sec  850   1.56 MBytes
[  5] 411.00-412.00 sec   111 MBytes   932 Mbits/sec  3219    201 KBytes
[  5] 412.00-413.00 sec   192 MBytes  1.61 Gbits/sec  4578   1.41 KBytes
[  5] 413.00-414.00 sec   113 MBytes   947 Mbits/sec  3939   18.4 KBytes
[  5] 414.00-415.00 sec   104 MBytes   871 Mbits/sec  6002   76.4 KBytes
[  5] 415.00-416.00 sec  4.88 MBytes  40.9 Mbits/sec    0    119 KBytes
[  5] 416.00-417.00 sec  9.62 MBytes  80.7 Mbits/sec   15    139 KBytes
[  5] 417.00-418.00 sec  21.8 MBytes   182 Mbits/sec    0    202 KBytes
[  5] 418.00-419.00 sec   350 MBytes  2.93 Gbits/sec  119    765 KBytes
[  5] 419.00-420.00 sec   158 MBytes  1.33 Gbits/sec  1290   2.05 MBytes
[  5] 420.00-421.00 sec   147 MBytes  1.23 Gbits/sec  4702   2.39 MBytes
[  5] 421.00-422.00 sec   119 MBytes   995 Mbits/sec  2844   21.2 KBytes
[  5] 422.00-423.00 sec   112 MBytes   940 Mbits/sec  3316    318 KBytes
[  5] 423.00-424.00 sec  91.1 MBytes   764 Mbits/sec  2775    122 KBytes
[  5] 424.00-425.00 sec  85.1 MBytes   714 Mbits/sec  2382    126 KBytes
[  5] 425.00-426.00 sec   170 MBytes  1.43 Gbits/sec  4001   32.5 KBytes
[  5] 426.00-427.00 sec  94.4 MBytes   792 Mbits/sec  5553    339 KBytes
[  5] 427.00-428.00 sec   115 MBytes   964 Mbits/sec  6422    188 KBytes
[  5] 428.00-429.00 sec   115 MBytes   966 Mbits/sec  4965    117 KBytes
[  5] 429.00-430.00 sec   163 MBytes  1.37 Gbits/sec  1902   1.41 KBytes
[  5] 430.00-431.00 sec   112 MBytes   942 Mbits/sec  828   1.41 KBytes
[  5] 431.00-432.00 sec  96.6 MBytes   811 Mbits/sec  7110   62.2 KBytes
[  5] 432.00-433.00 sec  6.38 MBytes  53.5 Mbits/sec    0    119 KBytes
[  5] 433.00-434.00 sec   156 MBytes  1.31 Gbits/sec    0    482 KBytes
[  5] 434.00-435.00 sec   205 MBytes  1.72 Gbits/sec  2640   1.41 KBytes
[  5] 435.00-436.00 sec  87.8 MBytes   736 Mbits/sec  2377   43.8 KBytes
[  5] 436.00-437.00 sec  93.8 MBytes   786 Mbits/sec  3817   94.7 KBytes
[  5] 437.00-438.00 sec   187 MBytes  1.57 Gbits/sec  1924   2.59 MBytes
```

可以看到，当整机带宽不足时，将优先保证在线任务使用带宽，离线任务的带宽将被抢占。

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.1.0	v1.18+	2023.09.13	CPU QoS 增强：支持 CPU 优先级配置，可以通过对工作负载设置优先级，保证高优先级业务在发生资源竞争时的资源供给量，并压制低优先级业务。 内存 QoS 增强：支持内存背景回收、OOMKill 优先级等，提升整体内存性能. 网络 QoS 增强：支持网络带宽优先级配置，出入方向限速等，灵活限制容器对网络的使用。	无

CCE NodeLocal DNSCache 说明

组件介绍

CCE NodeLocal DNSCache 组件是基于社区开源项目 NodeLocal DNSCache 的一套 DNS 本地缓存解决方案。在 CCE 集群中部署 NodeLocalDNS 可以提升服务发现的稳定性和性能，NodeLocal DNSCache 通过在集群节点上作为 DaemonSet 运行 DNS 缓存代理来提高集群DNS性能。该组件由两部分组成，分别是：NodeLocal DNSCache 本地缓存的 DaemonSet 和 用于 DNSConfig 注入的动态注入控制器 Deployment。

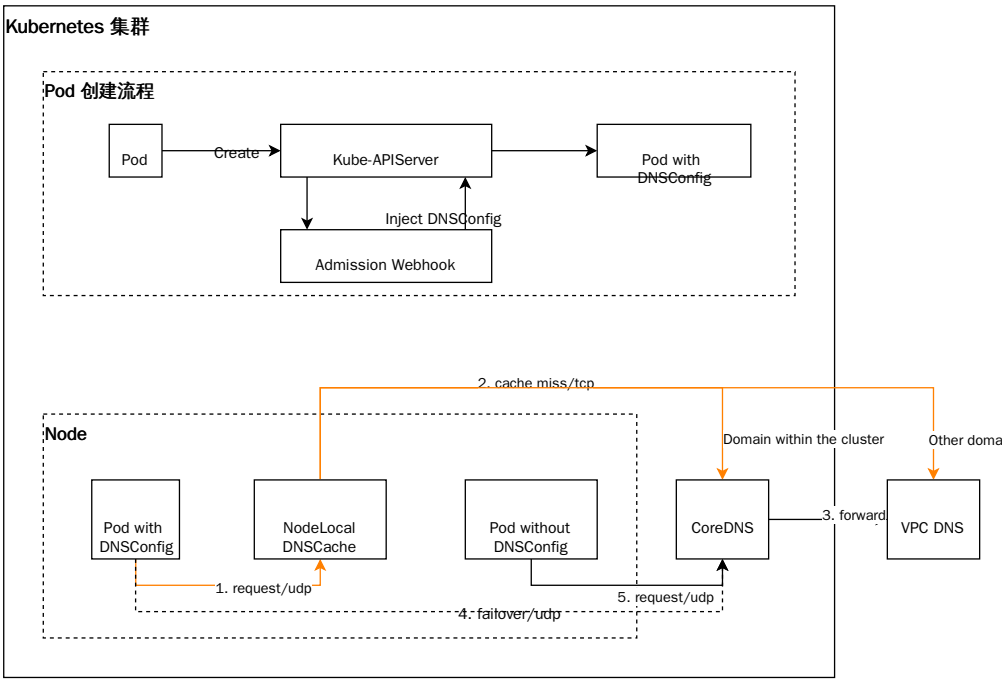
- NodeLocal DNSCache 本地缓存会在每个 Node 上创建一个 Pod 用于监听 Local IP（默认 IP 为 169.254.20.10），通过注入的 DNSConfig 配置， Pod 上的 DNS 请求会被 Local DNS 本地缓存所代理。

> Node Local DNS 服务是基于CoreDNS提供的服务，但其仅具有代理和缓存的功能。

- 用于 DNSConfig 注入的动态注入控制器是基于 Admission Webhook 机制拦截 Pod 创建的请求，自动为新建 Pod 注入 DNSConfig 配置信息。

> 如果没有启用该 Webhook，则需要您手动添加 DNSConfig 配置。

NodeLocal DNSCache 工作原理图



- 在 Pod 创建流程中，新增了 Admission Webhook 为 Pod 注入 DNSConfig。
- 在 Pod 的域名解析流程中，数据链路如下：

编号	描述
1	对于已经注入 DNSCofig 的 Pod，默认请求 NodeLocal DNSCache 在节点上监听的 IP 实现域名解析。
2	如果 NodeLocal DNSCache 无缓存应答解析请求，则会根据域名是否为集群内域名，转发对应请求到 CoreDNS 或 VPC DNS 进行解析。
3	CoreDNS对于解析失败的域名，会继续通过 VPC DNS 服务器进行解析。
4	对于已经注入 DNSCofig 的 Pod，默认优先使用 local DNS，在连接失败后请求 CoreDNS。但如果 Pod 本身使用的 DNS 请求库没有超时继续请求的设计，则只会使用localDNS。
5	未注入DNSConfig 的 Pod，会请求标准的 kube-dns 服务请求 CoreDNS 进行解析。

NodeLocal DNSCache 使用方式 为了使用 NodeLocal DNSCache，将请求 CoreDNS 的流量转到由本地 NodeLocal DNSCache 的 DaemonSet 代理，您需要在 Pod 内部注入 DNSConfig 配置，三种方式可供选择：

- 方式一：安装 CCE Node Local DNS 组件，通过组件中的 Webhook 实现 DNSConfig 注入，推荐使用该方式。
- 方式二：手动给 Pod 添加 DNSConfig
- 方式三：修改 kubelet 参数，并重启kubelet使其生效，不推荐该方式，业务存在中断风险。

方式一：DNSConfig 自动注入

> 前置条件：

- 集群安装 CCE NodeLocal DNSCache 组件

DNSConfig 动态注入控制器可实现在 Pod 新建时，自动注入 DNSConfig，无需手动添加配置。该 Webhook 默认会监听带有cce.baidu.com/node-local-dns-injection=enabled标签的命名空间中新建 Pod 的请求，您可以通过以下命令给命名空间打上Label标签:

```
kubectl label namespace default cce.baidu.com/node-local-dns-injection=enabled
```

> 请注意：

- 上述命令仅给 default 命名空间打上自动注入标签，这意味着只有在 default 命名空间下新建的 Pod 才会触发 Webhook 实现 DNSConfig 自动注入。如需修改命名空间，需要您将 default 替换成目标命名空间。
- 在满足上述条件下，如果您需要对部分 Pod 进行豁免（即不注入 DNSConfig），您可以给 Pod 添加标签：`cce.baidu.com/node-local-dns-injection=disabled` 。

DNSConfig自动注入生效后，您新建的 Pod 将会被添加以下字段：

```
dnsConfig:
 nameservers:
 - 169.254.20.10
 - 172.21.0.10
 options:
 - name: ndots
   value: "3"
 - name: attempts
   value: "2"
 - name: timeout
   value: "1"
 searches:
 - default.svc.cluster.local
 - svc.cluster.local
 - cluster.local
dnsPolicy: None
```

> 说明：

- nameservers：配置成169.254.20.10（默认值）和 kube-dns 的 ClusterIP 对应的IP地址，其中 ClusterIP 将作为备份的DNS服务器地址
- searches：设置搜索域，保证集群内部域名能够被正常解析，默认为 default.svc.cluster.local、svc.cluster.local 和 cluster.local
- ndots：默认为3，可以适当降低 ndots 以提升解析效率
- attempts：默认为2，尝试解析次数
- timeout：默认为1，超时时间

在 `方式一` 下，如果您新建的 Pod 未被注入 DNSConfig，请检查以下条件（只有当以下条件全部满足时，才会实现 Pod 自动注入）：

1. Pod 所在 Namespace 包含 label：cce.baidu.com/node-local-dns-injection=enabled
2. Pod 所在 Namespace 不为: kube-system、cce-monitor 或 cprom-system
3. Pod 不包含 label：cce.baidu.com/node-local-dns-injection=disabled
4. Pod 不包含要求调度到 VirtualKubelet 节点的调度配置（如节点选择、节点亲和性等配置）
5. Pod 的 hostNetwork 为 false 且 DNSPolicy 为 ClusterFirst 或 none，或 hostNetwork 为 true 且 DNSPolicy 为 ClusterFirstWithHostNet

方式二：手动添加DNSConfig 在不采用 `方式一` 的情况下，您需要手动添加 DNSConfig，参考案例如下：

```
  apiVersion: v1
kind: Pod
metadata:
  name: test01
  namespace: default
spec:
  containers:
  - image: registry.baidubce.com/cce-plugin-pro/cce-cni/netshoot:latest
    command:
      - sleep
      - "100000"
    imagePullPolicy: Always
    name: net-debug-container
  dnsPolicy: None
  dnsConfig:
    nameservers: ["169.254.20.10","172.21.0.10"]
    searches:
    - default.svc.cluster.local
    - svc.cluster.local
    - cluster.local
    options:
    - name: ndots
      value: "3"
    - name: attempts
      value: "2"
    - name: timeout
      value: "1"
```

> 请注意：
>
> - dnsPolicy必须为 None
> - 其他参数介绍请参考 `方式一` 说明

方式三：配置 kubelet 启动参数 您可以通过修改 kubelet 启动参数来全局控制Pod DNSConfig，详情请参考：Using NodeLocal DNSCache in Kubernetes Clusters。

组件功能

- 通过DNSConfig动态注入控制器Deployment，可以自动给新建 Pod 添加 DNSConfig 配置
- 通过在集群节点上以 Daemonset 的形式运行 NodeLocal DNS Cache，能够大幅提升集群内 DNS 解析性能

使用场景

通过在集群节点上作为 DaemonSet 运行 NodeLocal DNSCache 来提高集群DNS性能。

使用限制

- 集群版本在 1.18 以上
- CCE NodeLocal DNSCache 不支持 ARM 集群、边缘集群
- 能够使用 kubectl 访问集群，操作步骤见：通过kubectl连接集群

安装组件

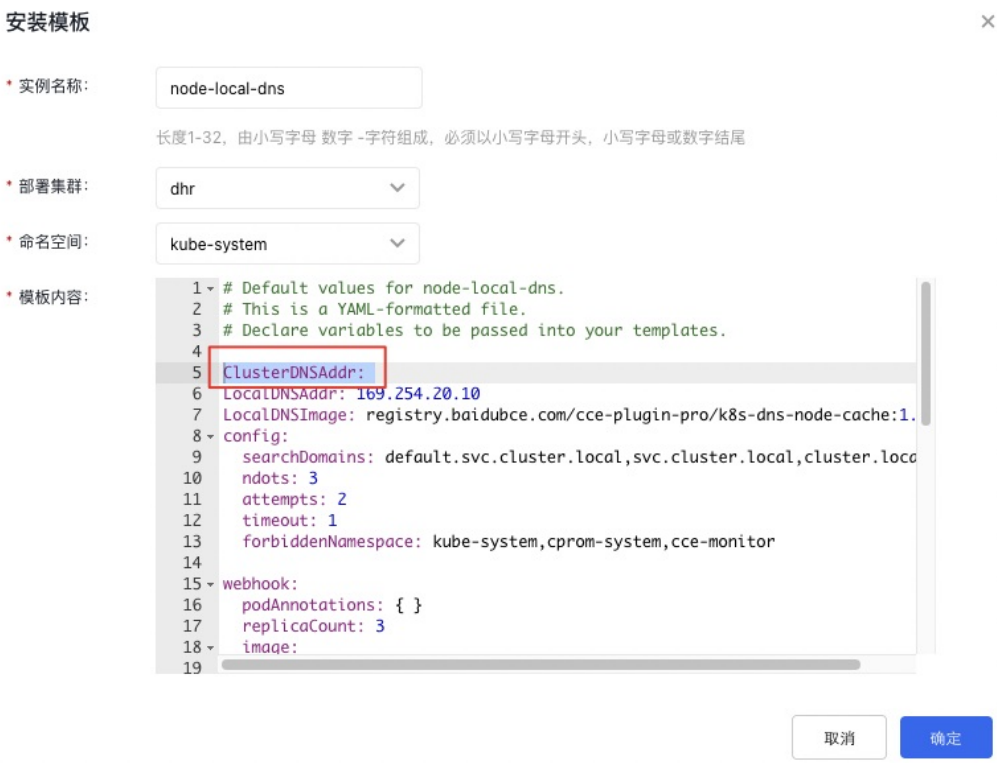
安装 CCE NodeLocal DNSCache 组件有两种方式，分别是：组件管理安装 和 Helm 模版安装，建议您通过组件管理安装该组件。 **方式一：通过组件管理安装 CCE Node Local DNS**

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **组件管理** 。

6. 在组件管理列表中选择 **NodeLocal DNSCache** 组件单击“**安装**”。
7. 点击“**确定**”按钮完成组件的安装。

方式二：通过 Helm 模版安装 CCE NodeLocal DNSCache

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品服务>云原生>容器引擎CCE**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 “**Helm > Helm 模版**”。
4. 在 Helm 模版页面中，单击 **百度智能云模版** 进入模版管理页面。
5. 在模版管理页面中选择模版名称 **cce-node-local-dns**，然后单击 “**安装**”。
6. 在 **安装模版** 页面中完成相关配置填写，点击 “**确定**” 按钮完成组件的安装。



相关配置填写说明如下表：

参数	是否必填	描述	获取路径
ClusterDNSAddr	是	kube-system命名空间下 是kube-dns的ClusterIP对应的IP地址。NodeLocal DNSCache通过该服务与CoreDNS通信来解析集群内域名。	通过kubectl连接集群， 输入 `kubectl get service kube-dns -n kube-system` 命令可以获取对应的 IP 地址信息。如果您需要指定不同的上游DNS服务器，可以修改此参数。无默认值，从集群中获取。
LocalDNSAddr	否	LocalDNSAddr 是 cce-node-local-dns 选择的本地侦听 IP 地址	默认为 169.254.20.10 ，可修改。
searchDomains	否	搜索域名，保证集群内部域名能够被正常解析	默认为 default.svc.cluster.local,svc.cluster.local,cluster.local，可修改。
ndots	否	可以适当降低ndots以提升解析效率	默认为 3，可修改。
attempts	否	解析尝试次数	默认为 2，可修改。
timeout	否	超时时间	默认为 1，可修改。
forbiddenNamespace	否	禁止 Pod 注入 DNSConfig 的 Namespace 列表	默认为 kube-system,cprom-system,cce-monitor，可修改。

部署情况

安装 CCE NodeLocal DNSCache 组件，将在 Kubernetes 集群中部署以下对象:

资源名称	类型
node-local-dns	ServiceAccount
cce-node-local-dns-cce-reserved	ConfigMap
node-local-dns	ConfigMap
cce-node-local-dns	ClusterRole
cce-node-local-dns	ClusterRoleBinding
kube-dns-upstream	Service
node-local-dns	Service
node-local-dns-webhook	Service
node-local-dns	DaemonSet
cce-node-local-dns-webhook	Deployment
pod-mutating.node-local-dns.cce.baidu.com	MutatingWebhookConfiguration

可观测性

您可以通过抓取 NodeLocal DNSCache 暴漏的指标接口，观测 NodeLocal DNSCache 组件的工作情况。

- 在集群中获取指标数据可以通过该方式：

```
curl http://<local ip>:9253/metrics # 默认为 169.254.20.10
```

下面是常见的 NodeLocal DNSCache 指标介绍：

指标	含义	类型	单位
coredns_build_info	CoreDNS的版本，修订和构建的 goversion	Gauge	/
coredns_cache_entries	缓存中的元素数量	Gauge	个
coredns_cache_hits_total	缓存命中的数量	Counter	次
coredns_cache_misses_total	缓存未命中的数量	Counter	次
coredns_cache_requests_total	缓存请求的数量	Counter	次
coredns_dns_request_duration_seconds	每个请求在每个区域所花费的时间的直方图	Histogram	秒
coredns_dns_request_size_bytes	EDNS0 UDP缓冲区的大小（对于TCP为64K）	Gauge	字节
coredns_dns_requests_total	DNS请求的数量	Counter	次
coredns_dns_response_size_bytes	返回的响应的大小	Gauge	字节
coredns_dns_responses_total	响应状态代码的数量	Counter	次
coredns_plugin_enabled	某插件在每个服务器和区域基础上是否启动	Gauge	/
coredns_reload_failed_total	重新加载失败的次数	Counter	次

在 CProm 中添加对 CCE NodeLocal DNSCache 的指标采集任务 在 CProm 中添加对 Node-Problem-Detector 的指标采集规则之前，需要先为 CCE 集群关联 CProm 实例，并添加指标采集任务。 操作参考：

- CProm 实例关联 CCE 集群
- CProm 实例中添加采集任务

添加如下指标采集任务：

```
 job_name: 'localdns'
scheme: http
static_configs:
- targets: [<local ip>:9253']
```

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.1.0	CCE/v1.18/v1.20	2023.11.20	首次上线	-
1.1.1	CCE/v1.18/v1.20	2023.12.21	缺陷修复：修正 DNSConfig 注入搜索域错误的问题	-
1.1.2	CCE/v1.18/v1.20	2023.12.22	缺陷修复：修正 Webhook 拦截规则	-

CCE Ascend Mindx DL说明

组件介绍

CCE Ascend Mindx DL组件是基于昇腾AI处理器的深度学习组件，提供昇腾AI处理器调度、集群性能测试等基础功能，为上层模型训练、模型部署、模型推理等应用提供底层软件支持。

组件功能

- 集群调度：基于Kubernetes增强了对昇腾AI处理器的调度，并支持查看昇腾节点、AI处理器状态。

使用场景

可以使您的CCE集群适配昇腾AI处理器，通过使用高效的集群调度组件，能够迅速部署和管理基于昇腾AI处理器的AI任务，为所有的AI作业提供昇腾AI处理器容器化支持。

注意事项

CCE Ascend Mindx DL组件安装前，请确保集群未安装冲突组件**CCE AI Job Scheduler**。

安装组件

1. 登录 百度智能云官网 ，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 **集群管理 > 集群列表** 。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击 **运维与管理 > 组件管理** 。
6. 在组件管理列表中选择 **CCE Ascend Mindx DL** 组件单击“安装”。

组件状态确认

通过如下命令查看K8s集群中跟CCE Ascend MindX DL组件相关的Pod，需要满足以下Pod的STATUS都为Running，READY都为1/1，则该组件运行状态为正常。

`kubectl get pods -n volcano-system`

```
 NAME                                  READY   STATUS    RESTARTS      AGE
volcano-controllers-86bxxxxxx1-xxxx1          1/1     Running   0           038m
volcano-controllers-86bxxxxxx2-xxxx2          1/1     Running   0           038m
volcano-controllers-86bxxxxxx3-xxxx3          1/1     Running   0           138m
volcano-node-controllers-6d8b9xxxxx-xxxxx       1/1     Running   0           09m10s
volcano-scheduler-5bbxxxxxx1-xxxx1            1/1     Running   0           038m
volcano-scheduler-5bbxxxxxx2-xxxx2            1/1     Running   0           138m
volcano-scheduler-5bbxxxxxx3-xxxx3            1/1     Running   0           038m
```

`kubectl get pods -n kube-system`

```
 NAME                                  READY   STATUS    RESTARTS      AGE
ascend-device-plugin-daemonset-xxxxx          1/1     Running   0           39m
npu-exporter-xxxxx                    1/1     Running   0           39m
```

如果当前集群中无昇腾对应的节点，则ascend-device-plugin-daemonset和npu-exporter副本数为0.

`kubectl get pods -n mindx-dl`

```
 NAME                                  READY   STATUS    RESTARTS      AGE
ascend-operator-manager-xxxxxxx-xxxxx          1/1     Running   0             038m
```

CoreDNS 说明

组件介绍

CoreDNS 提供了 Kubernetes 集群内部服务的域名解析能力。CoreDNS 版本与 Kubernetes 版本存在对应关系。在创建和升级您的集群时，将会遵循 CoreDNS version in Kubernetes 中版本对应关系为您安装兼容版本的 CoreDNS。关于版本对应关系的更多信息，请参见CoreDNS version in Kubernetes。

使用说明

CoreDNS 默认配置在 kube-system 的命名空间下，关于 CCE 集群中的 CoreDNS 配置项如何查看和修改，请参考CoreDNS组件手动升级指南。CoreDNS 会基于该配置项启用和配置插件。需要强调的是，不同 CoreDNS 版本的配置项有略微差异，修改配置前请仔细阅读CoreDNS官方文档和CoreDNS组件手动升级指南。以下是一个 1.20.3 版本CoreDNS 默认采用的配置文件：

```
.:53 {
    errors
    health {
       lameduck 10s
    }
    ready
    log
    kubernetes {{.ClusterDomain}} in-addr.arpa ip6.arpa {
       pods insecure
       fallthrough in-addr.arpa ip6.arpa
       ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
}
```

> 说明: 配置文件中ClusterDomain代指集群创建过程中填写的集群本地域名，默认值为cluster.local。

参数	描述
errors	错误信息到标准输出。
health	CoreDNS自身健康状态报告，默认监听端口8080，一般用来做健康检查。您可以通过http://localhost:8080/health 获取健康状态。
ready	CoreDNS插件状态报告，默认监听端口8181，一般用来做可读性检查。可以通过http://localhost:8181/ready 获取可读状态。当所有插件都运行后，ready状态为200。
kubernetes	CoreDNS Kubernetes插件，提供集群内服务解析能力。
prometheus	CoreDNS自身metrics数据接口。可以通过http://localhost:9153/metrics 获取prometheus格式的监控数据。
forward（或proxy）	将域名查询请求转到预定义的DNS服务器。默认配置中，当域名不在Kubernetes域时，将请求转发到预定义的解析器（/etc/resolv.conf）中。默认使用宿主机的/etc/resolv.conf配置。
cache	DNS缓存。
loop	环路检测，如果检测到环路，则停止CoreDNS。
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。
loadbalance	循环DNS负载均衡器，可以在答案中随机A、AAAA、MX记录的顺序。

限制说明

CoreDNS是系统组件，因为升级时可能会造成5分钟内全集群范围内的解析超时或失败的情况，所以暂不支持在控制台自动升级，请合理安排升级时间，进行手动升级。

安装组件

CoreDNS作为系统组件在创建集群时会默认安装，所以此处无需关注安装。

- CCE集群版本>=1.20，默认部署1.20.2版本，镜像版本为1.9.4-baidu
- CCE集群版本<1.20，默认部署1.0.0版本，镜像版本为1.7.1

版本记录

版本号	镜像版本	适配集群版本	更新时间	更新内容	影响
1.20.4	1.9.4-baidu-1	CCE/v1.20/v1.22/v1.22/v1.26	2024.07.02	取消通配符查询功能	--
1.20.3	1.9.4-baidu	CCE/v1.20/v1.22/v1.22/v1.26	2024.02.06	暴露configmap参数	--
1.20.2	1.9.4-baidu	CCE/v1.20/v1.22/v1.22/v1.26	2024.01.08	支持高版本集群	--
1.0.0	1.7.1	CCE/v1.18	2022.06.21	首次上线	--

CCE Network Plugin说明

组件介绍

CCE Network Plugin 是基于百度云弹性网卡实现的 CNI 插件。

组件功能

CCE Network Plugin 模式是百度云容器引擎 CCE 支持的扩展网络模式，基于百度云的弹性网卡产品，能够为集群内的 Pod 分配 VPC 内的 IP 地址。 由百度云 VPC 功能负责路由，打通容器网络的连通性，可实现 Pod 和 Node 的控制面和数据面完全在同一网络层面，该模式下的 Pod 能够复用百度云 VPC 所有产品特性。

使用限制

- CCE Network Plugin是系统组件，当前暂时不支持在控制台升级，如有升级需求请提工单

安装组件

安装组件 CCE Network Plugin作为系统组件，在创建集群时默认安装，此处您无需关注安装。

部署情况

安装 CCE Network Plugin组件，将在 Kubernetes 集群中部署以下对象:

对象名称	所属NameSpace	资源类型	所占资源
cceendpoints.cce.baidubce.com	-	CRD	-
clusterpodsubnettopologyspreads.cce.baidubce.com	-	CRD	-
enis.cce.baidubce.com	-	CRD	-
netresourcesets.cce.baidubce.com	-	CRD	-
podsubnettopologyspreads.cce.baidubce.com	-	CRD	-
subnets.cce.baidubce.com	-	CRD	-
cce-network-v2-reserved	kube-system	-	ConfigMap
cce-network-v2-config	kube-system	-	ConfigMap
cni-config-template	kube-system	-	ConfigMap
cce-network-operator	kube-system	-	Deployment
cce-network-agent	kube-system	-	DaemonSet
cce-network-v2	kube-system	-	Deployment
cce-cni-v2	kube-system	-	ServiceAccount
cce-network-v2	kube-system	-	Deployment
cce-cni-v2	kube-system	-	ClusterRole
cce-cni-v2	kube-system	-	ClusterRoleBinding
cce-network-v2-mutating-webhook	kube-system	-	MutatingWebhookConfiguration

主要版本变更

版本号	适配集群版本	更新时间	更新内容	限制和影响
v2.12	CCE/v1.18+	2024.06.28	新特性：支持 Burstable ENI 池，有效避免子网 IP 资源紧张时，节点 ENI 池资源不足的问题。 新特性：增加 eni 安全组同步功能， 保持CCE ENI 和节点安全组同步。 新特性：增加节点网络配置集功能 NetResourceConfigSet，支持指定节点独立配置网络资源。	无
v2.11	CCE/v1.18+	2024.05.27	新特性：容器内支持分配 RDMA 子网卡及 RDMA 辅助IP。	无
v2.10	CCE/v1.18+	2024.03.05	新特性：支持 VPC-ENI 模式下的 EBC 主网卡辅助 IP 模式。 新特性：重构 CNI 配置文件管理逻辑，支持保留自定义 CNI 插件配置。 新特性：增加对portmap 插件的支持，默认开启。 新特性：VPC-ENI 支持自动获取节点 ENI 配额信息，去掉了自定义 ENI 配额的参数。 新特性：支持通过在Node上添加 Annotation network.cce.baidubce.com/node-eni-max-ips-num 指定节点上 ENI 的最大辅助 IP 数量。	无
v2.9	CCE/v1.18+	2023.11.10	新的 CRD: 支持集群级 psts ClusterPodSubnetTopologyStrategy (cpsts)，单个cpsts 可以控制作用于整个集群的 psts 策略。 CRD 字段变更: NetworkResourceSet 资源池增加了节点上 ENI 的异常状态，报错单机 IP 容量状态，整机 ENI 网卡状态。 新特性: 支持ubuntu 22.04 操作系统，在容器网络环境下，定义 systemd-networkd 的 MacAddressPolicy 为 none。 新特性：支持 pod 级 Qos。	无

详细版本变更记录

2.12 新特性功能：

1. 支持 Burstable ENI 池，有效避免子网 IP 资源紧张时，节点 ENI 池资源不足的问题。
2. 增加 eni 安全组同步功能， 保持CCE ENI 和节点安全组同步。
3. 增加节点网络配置集功能 NetResourceConfigSet，支持指定节点独立配置网络资源。 **2.12.1 [2024/07/02]**
4. [Bug] 修复 bbc 机型开启 burstable ENI 时，初始化会导致空指针的问题
5. [Bug] 修复 bbc ENI 不返回实例 id 时，无法选中 ENI 的，影响节点就绪时间的问题 **2.12.0 [2024/06/28]**
6. [Feature] 支持 Burstable ENI 池，有效避免子网 IP 资源紧张时，节点 ENI 池资源不足的问题。
7. [Feature] 新增子网不足导致 ENI 创建失败 metrics 指标
8. [Feature] 增加 eni 安全组同步功能， 保持CCE ENI 和节点安全组同步
9. [Feature] 优化Pod调度算法，增加节点 ip capacity 自动适配，避免节点 IP 地址资源浪费
10. [Feature] 增加节点网络配置集功能 NetResourceConfigSet，支持指定节点独立配置网络资源
11. [Optimize] 修复 psts 对象在使用enableReuseIPAddress时可能更新 cep 时Addressing为空，不能记录错误信息的问题
12. [Optimize] 优化 operator 事件积压问题，避免事件长期超时积压
13. [Optimize] agent 优化 IP 地址 gc 算法，在达到 gc 周期后，支持按照 IP 地址清理已变更 cep 遗留地址的能力
14. [Optimize] 将动态 cep 与 nrs 生命周期绑定，减少 agent 被杀死的缩容时，遗留的 cep 对象数
15. [Optimize] 优化 rdma IP 申请流程，避免rdma 使用固定 IP 的 cep

2.11 (2024/5/27) 新特性功能：

1. 新特性：容器内支持分配 RDMA 子网卡及 RDMA 辅助IP。

2.11.3 [未发布]

1. [Feature] `--endpoint-gc-interval` 增加控制 agent 更新 nrs 的最小间隔时间
2. [Optimize] 优化对 eni 重启事件的处理逻辑，优化 agent 重启速度
3. [Optimize] 对 bce ENI 的 IP 地址做重排序，减少非必要 ENI 更新事件

4. [Bug] 优化手工删除subnet情况下StartSynchronizingSubnet可能遇到空指针

2.11.2 [20240616]

1. [Bug] 修复 nrs 删除后，仍会不断错误重试的问题，持续新建 eni 的问题
2. [Bug] 修复 agent 重启后，restore 失败的问题
3. [Feature] VPC-ENI 增加 ehc 机型支持
4. [Optimize] cce-network-operator 增加 alloc-worker ，可配置并发处理 nrs 对象协程数
5. [Optimize] 优化 rdma 默认预申请 13 个 IP，最大空闲 IP 数为 104 个，避免频繁申请与释放 IP。
6. [Bug] 修复 rdma 释放 IP 时可能遇见的空指针问题
7. [Optimize] 优化 ebc 主机新建子网逻辑，非主网卡辅助 IP 模式时，不再校验主网卡子网
8. [Optimize] 去除多余 operator 日志，减少 operator 占用资源
9. [Optimize] 重启 agent 时，更新最新的子网信息，避免重启后子网信息不一致
10. [Optimize] 缩小 operator 中非必要的锁范围，提升 operator 处理性能
11. [Optimize] 增加触发器强制结束时间，避免单个节点卡主影响整体同步
12. [Optimize] 增加 HPC eni OpenAPI接口限流
13. [Optimize] 合并 rdma 和以太网资源同步器，减少重复同步带来的资源开销
14. [Feature] 增加 IP 释放回收控制开关，默认不会回收 ENI IP
15. [Bug] 修复节点偶发 maintainIPPool 方法得不到调用，节点无法同步问题
16. [Bug] 修复bcesync map 并发访问问题
17. [Optimize] 子网IP不足时，Event增加对应Request ID
18. [Feature] 增加 trigger 触发器指标粒度，细化到节点

2.11.1 [20240611]

1. [Optimize] 更新RDMA IPPool MinAllocateIPs/PreAllocate/MaxAboveWatermark参数配置方式，与VPC-ENI保持一致方式
2. [Optimize] 保持RDMA网卡原始名称，不再rename RDMA网卡，避免Node上RDMA相关策略路由丢失
3. [Bug] 修复VPC路由下缺失ENISpec导致RDMA Discovery启动失败
4. [Bug] 修复ENI对象中RDMA网卡状态不同步问题
5. [Bug] 修复RDMA网卡最大IP数计算错误，优化报错信息
6. [Bug] 修复roce插件被误判为用户自定义插件

2.11.0 [20240527]

1. [Feature] 新特性：容器内支持分配 RDMA 网卡。 a. 支持单容器分配除VPC内以太网网卡之外，分配 RDMA 网卡的能力，分别支持 ERI 和 eRDMA 网卡。 b. 容器使用RDMA网卡为共享模式，单Node内的所有使用 RDMA 资源的容器共享 RDMA网卡，每个RDMA网卡创建子设备并在容器内有独立的 RDMA IP。

2.10 (2024/03/05) 新特性功能：

1. 新特性：支持 VPC-ENI 模式下的 EBC 主网卡辅助 IP 模式。
2. 新特性：重构 CNI 配置文件管理逻辑，支持保留自定义 CNI 插件配置。
3. 新特性：增加对portmap 插件的支持，默认开启。
4. 新特性：VPC-ENI 支持自动获取节点 ENI 配额信息，去掉了自定义 ENI 配额的参数。
5. 新特性：支持通过在Node上添加 Annotation `network.cce.baidubce.com/node-eni-max-ips-num` 指定节点上 ENI 的最大辅助 IP 数量。

2.10.4/2.10.5 [202405011]

1. [Bug] 修复 VPC-ENI 模式下，单机最大 IP 地址容量计算错误的问题

2.10.3 [20240425]

1. [Bug] 修复 ResyncController 已经添加EventHandler 时，informer 重复添加处理器, psts 会收到重复事件会导致 IP 地址冲突的问题

2.10.2 [20240403]

1. [Bug] 修复 vpc-route 模式下，重写 cni 文件错误问题

2.10.1 [20240325]

1. [Bug] 修复 vpc-route 模式下，重启 operator 可能导致多个节点的 cidr 重复的问题
2. [Bug] 修复调用 bce sdk 出错时，可能出现的stack overflow，导致operator重启的问题
3. [Optimize] vpc-eni 增加 mac 地址合法性校验，避免误操作其它网卡

2.10.0 （2024/03/05）

1. [Feature] VPC-ENI 支持自动获取节点 eni 配额信息，去掉了自定义 ENI 配额的参数。
2. [Feature] VPC-ENI 支持 ebc 主网卡辅助 IP 模式
3. [Feature] VPC-ENI bbc 升级为主网卡辅助 IP 模式
4. [Optimize] 增加 CNI 插件日志持久化
5. [Feature] 重构 CNI 配置文件管理逻辑，支持保留自定义 CNI 插件配置
6. [Feature] 增加对portmap 插件的支持，默认开启
7. [Feature] 支持通过在Node上添加 Annotation `network.cce.baidubce.com/node-eni-max-ips-num` 指定节点上 ENI 的最大辅助 IP 数量。
8. [Bug] 修复arm64架构下，cni 插件无法执行的问题
9. [Optimize] 增加 BCE SDK 日志持久化
10. [Optimize] 优化去掉 bce sdk 的退避重试策略，避免频繁重试
11. [Optimize] 支持使用`default-api-timeout` 自定义参数指定 bce openAPI 超时时间

2.9 (2023/11/10) 新特性功能：

1. 新的 CRD: 支持集群级 psts ClusterPodSubnetTopologyStrategy (cpsts)，单个cpsts 可以控制作用于整个集群的 psts 策略。
2. CRD 字段变更: NetworkResourceSet 资源池增加了节点上 ENI 的异常状态，报错单机 IP 容量状态，整机 ENI 网卡状态。
3. 新特性: 支持ubuntu 22.04 操作系统，在容器网络环境下，定义 systemd-networkd 的 MacAddressPolicy 为 none。
4. 新特性：支持 pod 级 Qos

2.9.5 [20240325]

1. [Bug] 修复 vpc-route 模式下，重启 operator 可能导致多个节点的 cidr 重复的问题
2. [Bug] 修复调用 bce sdk 出错时，可能出现的stack overflow，导致operator重启的问题

2.9.4 [20240305]

1. [Feature] 支持 BBC 实例通过 Node 上增加 `network.cce.baidubce.com/node-eni-subnet` Anotation 配置指定节点上 ENI 的子网。

2.9.3 [20240228]

1. [Feature] cce-network-agent 自动同步节Node的Annoation信息到CRD中。
2. [Feature] 支持 EBC/BCC 实例通过 Node 上增加 `network.cce.baidubce.com/node-eni-subnet` Anotation 配置指定节点上 ENI 的子网。
3. [Feature] 新增`enable-node-annotation-sync`参数，默认关闭。
4. [Bug] 修正预申请 IP 时，可新建 ENI 的数量计算错误。

2.9.2 [20240223]

1. [Bug] 修复arm64架构下，cni 插件无法执行的问题

2.9.1 [20240115]

1. [Optimize] 优化NetResourceManager在接收事件时处理的锁，消除事件处理过程中 6 分钟延迟
2. [Optimize] 优化ENI状态机同步错误时，增加 3 次重试机会，消除因 ENI 状态延迟导致的 10 分钟就绪延迟
3. [Bug]修复 cce-network-agent 识别操作系统信息错误的问题
4. [Bug]修复cce-network-agent pod 被删除后，小概率导致 operator 空指针退出问题
5. [Bug]修复创建 eni 无法向 nrs 对象上打印 event 的问题

2.9.0 [20240102]

1. [Optimize] 申请 IP 失败时,支持给出失败的原因.包括: a. 没有可用子网 b. IP 地址池已满 c. 节点 ENI 池已满 d. 子网没有可用 IP e. IP 缓存池超限
2. [Feature] 新增 CRD: ClusterPodSubnetTopologyStrategy (cpsts), 用于控制集群级别的 psts 策略。 a. 当前 crd 版本 cce.baidu.com/v2beta1 b. cpsts 支持为所有符合namespaceSelector的 namespace 配置 psts 策略，并作为自身的子对象管理生命周期和状态。
3. [Feature]支持ubuntu 22.04 操作系统，在容器网络环境下，定义 systemd-networkd 的 MacAddressPolicy 为 none。
4. [Feature]支持 pod 级别的带宽控制，通过在 Pod 上设置 annotation 控制 Pod 级别的带宽。 a. `kubernetes.io/ingress-bandwidth: 10M` 配置 Pod 的 ingress 带宽为 10M b. `kubernetes.io/egress-bandwidth: 10M` 配置 Pod 的 egress 带宽为 10M
5. [Feature]支持 pod 级别的 QoS，通过在 Pod 上设置 annotation 控制 Pod 的 QoS。 a. `cce.baidubce.com/egress-priority: Guaranteed` 配置 Pod 的流量为 Guaranteed （最低延迟）优先级 b. `cce.baidubce.com/egress-priority: Burstable` 配置 Pod 的流量为 Burstable （高优先级） c. `cce.baidubce.com/egress-priority: BestEffort` 配置 Pod 的 egress 流量为低优级
6. [Optimize] 修改 --bce-customer-max-eni 及 --bce-customer-max-ip 参数的逻辑，当参数非 0 时，强制生效
7. [Bug] 修复独占eni模式下容器网络命名空间挂载类型为tmpfs 时，无法读取netns的问题
8. [Feature] 增加override-cni-config开关，默认在 agent 启动时强制覆盖 cni 配置文件
9. [Feature] psts 复用 IP 时增加亲和性调度功能，保证同名 Pod 重复调度时，可以调度到同一可用区服用子网。
10. [Optimize] 优化并发创建 ENI 逻辑，避免在业务无需过多 IP 时并发创建过多 ENI
11. [Optimize] 优化 ENI 命名长度，限制为 64 字符
12. [Bug] 修复VPC-ENI 并发申请和释放IP 时，Pod 可能申请到过期的 IP 地址的问题

2.8 (2023/08/07)

1. 正式发布容器网络v2版本

2.8.8 [20231227]

1. [Bug] VPC-ENI 并发申请和释放IP 时，Pod 可能申请到过期的 IP 地址

2.8.7 [20231127]

1. [Bug] 修复 cce-network-v2-config 中 --bce-customer-max-eni 及 --bce-customer-max-ip 参数配置不生效；未限制并发创建 ENI ，并发下最大 ENI 数量可能超发

2.8.6 [20231110]

1. [Bug] 优化 EndpointManager 在更新 endpoint 对象时不会超时的逻辑，且由于资源过期等问题会出现死循环的问题
2. [Optimize] 优化 operator 工作队列，支持自定义 worker 数量，加速事件处理
3. [Optimize] EndpointManager 核心工作流日志，把关键流程日志修改为 info 级别
4. [Optimize] 优化EndpointManager gc工作流，动态 IP 分配的 gc 时间设置为一周

5. [Optimize] 增加 ENI VPC 状态机流转时没有触发状态变更时重新入队时间，加速 ENI 就绪时间
6. [Optimize] 增加增删 ENI 状态变更事件，增加 ENI 的 VPC 非终态日志记录
7. [Optimize] 缺少 metaapi 时，记录相关事件
8. [Optimize] 当VPC路由满，记录相关事件

2.8.5 [20241017]

1. [Optimize] 优化了 psts 分配 IP 时失败的回收机制，避免出现 IP 泄露
2. [Bug] 修复 vpc 路由模式下 nrs 标记 deleteTimeStamp 之后，由于 vpc 路由状态处于 released，nrs 的 finallizer 无法回收的问题
3. [Optimize] 优化创建 cep 的逻辑，当创建 cep 失败时，尝试主动删除并重新创建 cep

2.8.4 [20230914]

1. [Bug] vpc-eni，修复在 centos 8等使用 NetworkManager 的操作系统发行版，当 ENI 网卡被重命名后，DHCP 删除 IP 导致 ENI 无法就绪的问题

2.8.3 [20230904]

1. [Feature] 支持kubelet删除cni配置文件后，重新创建配置文件
2. [Feature] network-agent支持开启pprof，并获取mutex和block数据
3. [Optimize] 去掉network-agent在申请IP时的填充锁
4. [Bug] 修复network-agent的默认限流配置

2.8.2 [20230829]

1. [Optimize] 提升创建ENI的性能，缩短nrs任务管理时间
2. [Optimize] 增加并发预创建eni的逻辑，当单机未达到预加载eni数时，并发创建eni
3. [Bug] 修复创建ENI时，使用eni名字查询eni对象为空，使每个ENI最少需要1m创建时间的问题

CCE LB Controller 说明

组件介绍

CCE LB Controller基于百度智能云 应用型负载均衡BLB 实现K8s类型LoadBalancer Service, 提供 4 层负载均衡能力，用户在使用LoadBalancer Service的同时将会联动创建和管理百度智能云应用型BLB服务，从而实现流量的接入和管理。

LoadBalancer Service服务的使用方式与原生kubernetes Service保持一致，用户可以通过百度智能云控制台或者YAML创建和管理LoadBalancer Service。

安装组件

CCE LB Controller为系统组件，在创建集群时默认安装最新版本。在托管集群中为托管组件，部署在百度智能云CCE上，无需用户运维；在独立集群中部署在集群节点上，需要用户自行运维。

> **说明：**
>
> 仅带有【托管】标识的组件才表示该组件部署在百度智能云CCE上，为托管组件。

部署情况

安装CCE LB Controller组件，将在Kubernetes集群中部署以下对象:

对象名称	所属NameSpace	所占资源	资源类型
cce-lb-controller-cce-reserved	kube-system	-	ConfigMap
cce-lb-controller	kube-system	副本数：3 CPU：200m Memory：500Mi	Deployment
cce-lb-clusterrole	kube-system	-	ClusterRole
system:cce-lb-controller	kube-system	-	ClusterRoleBinding
cce-lb-controller-serviceaccount	kube-system	-	ServiceAccount

版本记录

版本号	适配集群版本	更新时间	更新内容	限制和影响
1.3.3	CCE/v1.18+	2025-04-11	修复删除资源时空指针问题	-
1.3.2	CCE/v1.18+	2025-04-01	修复IP Group并发冲突的问题	-
1.3.1	CCE/v1.18+	2025-03-21	支持多Service复用一个同一应用型BLB	-
1.2.21	CCE/v1.18+	2024-09-03	修复服务全量 Pod 变更过快时（30秒内）BLB 断流问题	-
1.2.20	CCE/v1.18+	2024-08-02	修复持续删除Finalizer导致频繁查询BLB的问题	-
1.2.19	CCE/v1.18+	2024-06-18	• 支持指定应用型BLB性能规格 • 修复BLB因余额不足创建失败后，无法删除LB Service的问题 • 修复集群节点增删时，仅有一个LB Service会同步BLB RS的问题 • 修复LB Service有多个相同前缀的端口值时，无法更新BLB RS的问题 • 修复LB Service Endpoints变更后，IPv6型BLB RS更新延迟问题 • 修复Pod未Ready已被添加到BLB RS的问题	-
1.2.18	CCE/v1.18+	2024-04-17	支持边缘节点集群	-
1.2.17	CCE/v1.18+	2024-04-17	Pod直连模式BLB RS同步容错优化	-
1.2.16	CCE/v1.18+	2024-03-05	支持托管模式	-
1.2.15	CCE/v1.18+	2024-01-24	支持同端口多协议	-
1.2.14	CCE/v1.18+	2024-01-24	增加对 LCC 场景的检查 修复健康检查端口设置问题	-
1.2.13	CCE/v1.18+	2023-09-18	增加按 Label 选择后端节点的机制	-
1.2.13	CCE/v1.18+	2023-09-18	增加按 Label 选择后端节点的机制	-
1.2.12	CCE/v1.18+	2023-09-18	健康检查配置与实时更新	-
1.2.11	CCE/v1.18+	2023-08-22	为 BLB 和 EIP 绑定 Tag	-
			支持 IPv6 双栈 支持设置最大 BLB 长度	

1.2.10	CCE/v1.18+	2023-08-14	支持默认为 BLB 开启"删除保护"功能 监听器配置全覆盖、EIP 带宽可配置 监听器配置与 EIP 带宽配置支持实时更新 修复 Pod 直连模式使用字符串类型 Port 的问题	-
1.2.9	CCE/v1.18+	2023-07-27	支持配置默认EIP计费类型	-
1.2.8	CCE/v1.13/v1.16/v1.18	2023-05-18	支持配置默认EIP计费类型	-
1.2.7	CCE/v1.13/v1.16/v1.18	2023-02-09	支持扩展协议配置 优化后台鉴权机制 支持后端服务器动态权重	-
1.2.6	CCE/v1.13/v1.16/v1.18	2022-12-30	部分 BLB 配置不符合预期直接报错 取消对 BLB Name 等字段的更新	-
1.2.5	CCE/v1.13/v1.16/v1.18	2022-09-14	修复服务器组名称问题 添加监控配置并部署于master节点	-
1.2.4	CCE/v1.13/v1.16/v1.18	2022-05-30	修复 EIP 与 Service 更新问题	-
1.2.3	CCE/v1.13/v1.16/v1.18	2022-04-22	修复 NodePort 未联动变更问题 修复 IP 分配配置 支持 EIP 计费方式可配置	-
1.2.5	CCE/v1.13/v1.16/v1.18	2022-05-30	修复 EIP 与 Service 更新问题	-
1.2.3	CCE/v1.13/v1.16/v1.18	2022-04-22	修复 NodePort 未联动变更问题 修复 IP 分配配置 支持 EIP 计费方式可配置	-
1.2.2	CCE/v1.13/v1.16/v1.18	2022-03-11	支持配置 BLB 最大后端数	-
1.2.1	CCE/v1.13/v1.16/v1.18	2022-02-14	允许在启动参数配置 Tag Endpoint	-
1.2.0	CCE/v1.13/v1.16/v1.18	2022-02-14	修复资源删除幂等性、预付费 EIP 删除等方面存在的问题 优化组件内部运行逻辑	-
1.1.4	CCE/v1.13/v1.16/v1.18	2021.09.07	优化 Event 优化应用型BLB名称 支持 TargetPort 使用字符串	-
1.1.3	CCE/v1.13/v1.16/v1.18	2021.08.10	在 Master 启动 Kubelet 时, LB Controller 不再将 Master 挂载为 LoadBalancer Service 的后端	-
1.1.2	CCE/v1.13/v1.16/v1.18	2021.07.29	调整 LB Service Status 中的 IP 顺序，将公网 IP 作为 IP 列表中第一个	-
1.1.1	CCE/v1.13/v1.16/v1.18	2021.07.29	修复环境变量读取 Bug	-
1.1.0	CCE/v1.13/v1.16/v1.18	2021.03.15	CCE LB Controller 负责处理所有类型的 LoadBalancer Service 增加 LoadBalancer Service Finalizer 的使用 修复已经发现的一些 Bug 优化 CCE LB Controller 日志输出和 Event 事件展示	-
1.0.0	CCE/v1.13/v1.16/v1.18	2021.03.15	首次上线	-

CCE CronHPA Controller 说明

组件介绍

CCE定时扩缩容管理组件(CCE-CronHPA-Controller)支持用户通过CRD的方式创建定时规则以对工作负载进行扩缩容。对于具有明显周期性负载的应用，可以通过容器定时水平伸缩的方式提供预弹的能力。

组件功能

- 对指定的工作负载副本数进行定时扩缩容

使用场景

- 根据预定的时间表来调整集群资源，通过设定一系列的任务，CronHPA可以在预设的时间点自动执行扩容或缩容操作。
- 保证在用户访问量激增时，系统资源已经按预期配置调整到位，避免因扩容延迟引发的服务中断。

使用限制

- 支持v1.16及以上版本的Kubernetes集群

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品服务>云原生>容器引擎CCE**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**组件管理**"。
6. 在组件管理列表中选择"**CCE CronHPA Controller**"组件单击"**安装**"。
7. 点击"确认"按钮完成组件的安装。

部署情况

安装CCE CronHPA Controller组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
cronhpas.cce.baidubce.com	kube-system	-	CRD
cce-cronhpa-controller	kube-system	CPU:100m Memory:100Mi	Deployment

使用

通过yaml的方式在集群中部署CronHPA,CCE CronHPA Controller监听到CronHPA的创建进行相应的逻辑处理。

CronHPA模版示例

```
  apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef:
      apiVersion: apps/v1
      kind: Deployment
      name: nginx-deployment-basic
  excludeDates:
  - "* * * 15 11 *"
  - "* * * 1 11 *"
  crons:
  - name: "scale-down"
    schedule: "30 */1 * * * *"
    targetSize: 1
    runOnce: true
  - name: "scale-up"
    schedule: "01 */1 * * * *"
    targetSize: 3
```

部署参数介绍

名称	作用	是否必须
spec.scaleTargetRef	期望扩缩容对象的元数据	必须
crons.name	任务名称	必须
crons.schedule	任务触发的时间	必须
crons.targetSize	期望副本数	必须
spec.excludeDates	任务不触发的时间	可选

其中schedule同excludeDates需要按照如下格式配置:

字段名称	是否必填	允许值	允许的特殊字符
Seconds	是	0-59	*/,-
Minutes	是	0-59	*/,-
Hours	是	0-23	*/,-
Day of month	是	1-31	*/,-?
Month	是	1-12 或 JAN-DEC	*/,-
Day of week	是	0-6 或 SUN-SAT	*/,-?

特殊字符说明

- 星号 (*) 表示表达式能匹配该字段的所有值。如在第5个字段使用星号(month)，表示每个月。
- 斜线(/) 表示增长间隔，如第1个字段(minutes) 值是 3-59/15，表示每小时的第3分钟开始执行一次，之后每隔 15 分钟执行一次（即 3、18、33、48 这些时间点执行），这里也可以表示为：3/15
- 逗号(,) 用于枚举值，如第6个字段值是 MON,WED,FRI，表示 星期一、三、五 执行。
- 连字号(-) 表示一个范围，如第3个字段的值为 9-17 表示 9am 到 5pm 直接每个小时（包括9和17））
- 问号(?) 只用于日(Day of month)和星期(Day of week)，\表示不指定值，可以用于代替 *

cron举例说明

- 每隔5秒执行一次：*/5 * * * ?
- 每隔1分钟执行一次：0 */1 * * ?
- 每天23点执行一次：0 0 23 * * ?
- 每天凌晨1点执行一次：0 0 1 * * ?
- 每月1号凌晨1点执行一次：0 0 1 1 * ?
- 在26分、29分、33分执行一次：0 26,29,33 * * * ?
- 每天的0点、13点、18点、21点都执行一次：0 0 0,13,18,21 * * ?

CronHPA扩缩容普通对象示例 CronHPA要应用在普通工作负载只需要将spec.scaleTargetRef字段配置成该工作负载的信息，以下是一个具体的CronHPA扩缩容Deployment的示例:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-basic
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef: //配置成以上deployment的信息
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment-basic
  crons:
  - name: "scale-down"
    schedule: "30 */1 * * * *"
    targetSize: 1
  - name: "scale-up"
    schedule: "01 */1 * * * *"
    targetSize: 3
```

CronHPA兼容HPA 从CronHPA和HPA的模板定义中可以看出：

- CronHPA和HPA都是通过scaleTargetRef字段来获取伸缩对象。
- CronHPA通过jobs的crontab规则定时伸缩副本数。
- HPA通过资源利用率判断伸缩的情况。

如果同时设置CronHPA和HPA，那么会出现CronHPA和HPA同时操作一个scaleTargetRef的场景。CronHPA和HPA相互独立无法感知，后执行的操作会覆盖先执行的操作,例如刚刚定时扩容完成但是又由于资源利用率低被缩容回去了。 所以当期望扩缩容的对象被HPA引用的时候，为了避免HPA与CronHPA的冲突导致结果被覆盖，需要把spec.scaleTargetRef配置为对应的HPA对象，这样CronHPA能够感知HPA的状态并且通过调整HPA的上下限来调整副本数。

以下是一些兼容CronHPA与HPA来实现定时扩缩容的具体场景示例:

HPA(min/max)	CronHPA	deployment	result	说明
1/10	5	5	HPA(1/10) deployment 5	CronHPA期望副本数为固定一致，无需变更
1/10	4	5	HPA(1/10) deployment 5	当前副本数5高于CronHPA期望副本数4，并且期望副本数在区间[1,10]内，无需变更
1/10	6	5	HPA(6/10) deployment 6	CronHPA期望副本数6高于当前副本数5，调整HPA下限自动扩容副本数至6
5/10	4	5	HPA(4/10) deployment 5	CronHPA期望副本数低于当前副本数5，保留当前副本数 CronHPA期望副本数低于HPA下限，修改HPA下限(保证至少有4个副本)
5/10	11	5	HPA(11/11) deployment 11	CronHPA期望副本数高于HPA上限，修改HPA下限至CronHPA期望副本数，上限同下限

如下是CronHPA兼容HPA具体的yaml示例：

```
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-deployment-basic-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment-basic
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
---
apiVersion: cce.baidubce.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef://配置为HPA对象
    apiVersion: autoscaling/v1
    kind: HorizontalPodAutoscaler
    name: nginx-deployment-basic-hpa
  crons:
    - name: "scale-down"
      schedule: "30 */1 * * * *"
      targetSize: 5
      runOnce: false
    - name: "scale-up"
      schedule: "01 */1 * * * *"
      targetSize: 10
      runOnce: false
```

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.1.0	CCE/v1.16+	2024.03.15	首次上线	-

CCE CSI PFS L2 Plugin

组件介绍

CCE CSI PFS L2 Plugin支持在Pod中以PV/PVC方式挂载百度云L2类型PFS存储，并支持动态创建PV。

组件功能

容器的特性决定了容器本身是非持久化的，容器被删除，其上的数据也一并删除。存储卷Volume可以解决容器的数据共享和持久化存储问题，PV（PersistentVolume）和PVC（PersistentVolumeClaim）是K8S提供的用于抽象存储细节的API资源。

利用百度智能云PFS L2服务在集群内创建PV和PVC资源，用户可以直接将PFS L2作为存储卷挂载到容器中，而无需关注底层的实现细节，从而更加便捷地为容器集群提供持久化存储方案。

前提条件

- 已创建PFS挂载服务，更多信息请见 PFS挂载服务

使用限制

- PFS L2 支持的操作系统和内核版本列表请查看 使用限制（极速型L2）

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择"**产品导览>容器>容器引擎**"，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的"**集群管理>集群列表**"。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击"**运维与管理>组件管理**"。
6. 在组件管理列表中选择"**CCE CSI PFS L2 Plugin**"组件单击"**安装**"，在弹框中完成相关配置。

配置项	必选/可选	配置说明
关联挂载服务	必选	指定组件关联的PFS挂载服务，挂载服务需要与集群在同一VPC中，挂载后集群可以使用挂载服务绑定的PFS实例。

7. 单击"**确认**"按钮完成组件的安装。

部署情况

安装CCE CSI PFS L2 Plugin组件，将在Kubernetes集群中部署以下对象:

对象名称	所属namespace	所占资源	类型
cce-csi-pfsl2-plugin-attacher	kube-system	-	Deployment
cce-csi-pfsl2-plugin-operator	kube-system	-	Deployment
cce-csi-pfsl2-plugin-provisioner	kube-system	-	Deployment
cce-csi-pfsl2-plugin-resizer	kube-system	-	Deployment
cce-csi-pfsl2-plugin-snapshotter	kube-system	-	Deployment
cce-csi-pfsl2-plugin	kube-system	CPU:20m;Memory:20Mi	DaemonSet
cce-csi-pfsl2-plugin-cce-reserved	kube-system	-	ConfigMap

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
1.0.8	CCE/1.18+	2025.04.15	支持集群挂载多个PFS实例	-
1.0.7	CCE/1.18+	2025.03.14	支持PFS成都地域	-
1.0.6	CCE/1.18+	2024.12.13	支持hostname作为nodeName进行挂载	-
1.0.5	CCE/1.18+	2024.10.11	修复kubelet root dir软链偶发清空数据目录问题	-
1.0.4	CCE/1.18+	2024.08.28	**新增功能**： • 支持pv参数mountPath中子路径自动创建 • 支持移出节点时自动卸载 **优化功能** • 优化kubelet root path软链问题 • 优化组件污点容忍	-
1.0.3	CCE/1.18+	2024.07.08	优化和PFS交互的地域参数	-
1.0.2	CCE/1.18+	2024.05.16	优化插件的交互流程,上线到组件中心	-
1.0.1	CCE/1.18+	2023.12.06	增加StorageClass参数	-
1.0.0	CCE/1.18+	2023.11.30	首次发布PFS L2 CSI版本	-

Kube Scheduler 说明文档

Kube Scheduler 说明

组件介绍 Kube Scheduler 基于 Pod 申明的 Request 和节点的 Allocatable 属性，Kube Scheduler 可以为调度队列中每个 Pod 确定其可放置的节点，并保证节点的合法性。此外，Kube Scheduler 还可以将所有合法的节点排序，将 Pod 绑定到一个合适的节点。Kube Scheduler 默认根据 Request 值对 Pod 进行均匀放置。更多信息，请参见Kubernetes官方文档kube-scheduler。

使用说明 Kube Scheduler 组件默认安装，无需配置即可使用。

变更记录 Kube Scheduler 组件跟随 Kubernetes 版本升级，详情请参见Kubernetes版本发布概览。

Kube Controller Manager 说明

Kube Controller Manager 说明

组件介绍 Kube Controller Manager是一个守护进程，内嵌随 Kubernetes一起发布的核心控制回路。Kube Controller Manager通过 API 服务器监控集群的状态，确保集群处于预期的工作状态。Kube Controller Manager由负责不同资源的多个控制器构成。目前，Kubernetes自带的控制器包括副本控制器、节点控制器、命名空间控制器和服务账号控制器等。

使用说明 Kube Controller Manager组件默认安装，无需配置即可使用。

变更记录 Kube Controller Manager组件跟随Kubernetes版本升级，详情请参见Kubernetes版本发布概览。

Kube ApiServer 说明

Kube API Server 说明

组件介绍

Kube API Server 用于验证并配置 API 对象的数据，这些对象包括 Pods、Services、Deployments等。 Kube API Server 为 REST 操作提供服务，并为集群的共享状态提供接口，所有其他组件都通过该接口进行交互。

使用说明 Kube API Server组件默认安装，无需配置即可使用。

变更记录

Kube API Server 组件跟随 Kubernetes 版本升级，详情请参见Kubernetes版本发布概览。

CCE Backup Controller 说明

组件介绍

基于开源 Velero 实现的集群扩展组件，提供资源备份和恢复能力。

组件功能

基于Velero实现集群中应用（资源YAML）的备份和恢复，并将备份数据安全地存储在指定的对象存储 BOS Bucket 中。

使用场景

- 备份恢复：当集群或命名空间下的K8s资源被误删除时，可以通过备份数据快速恢复业务。
- 业务迁移：可以将源集群备份的业务快速迁移到目标集群。（暂不支持，敬请期待）
- 业务合规：配合安全部门定期拉取备份数据进行业务审计。

限制说明

- 仅支持 v1.20 及以上版本的 Kubernetes 集群。
- 备份应用时，正在被删除的资源不会被备份。

安装组件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“**产品服务>云原生>容器引擎CCE**”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的“**集群管理>集群列表**”。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击“**运维语管理>组件管理**”。
6. 在组件管理列表中选择“**CCE Backup Controller**”组件单击“**安装**”。
7. 点击“**确认**”按钮完成组件的安装。

使用说明

请配合 CCE 备份中心使用该组件。

1. 登录容器引擎控制台,创建备份仓库，具体请参见备份仓库。
2. 在源集群中创建备份或定时备份策略备份指定的资源对象，具体请参见备份管理。
3. 在目标集群中根据备份数据创建恢复任务恢复指定的资源对象，具体请参见恢复管理。

版本记录

版本号	适配集群版本	更新时间	更新内容
1.0.0	CCE/v1.20+	2024.08.30	组件中心新增备份组件

CCE Credential Controller 说明

使用前提

- 创建容器镜像服务CCR企业版实例，具体操作请参考 创建企业版实例 。
- CCE集群所在的私有网络配置到CCR企业版实例。CCE集群所在的私有网络查询请参考 查看集群，CCR企业版实例配置私有网络访问请参考 配置私有网络访问。

组件介绍

CCE Credential Controller 组件提供在CCE集群中免密拉取CCR企业版容器镜像的能力。

- 在CCE集群中安装免密组件后，组件会自动创建CCR企业版实例的镜像访问凭据，并关联到免密组件生效的命名空间和ServiceAccount下。
- 在免密组件生效的命名空间下使用ServiceAccount创建工作负载时，默认会自动配置ImagePullSecret拉取CCR企业版实例的镜像，做到免配置仓库秘钥使用CCR镜像的效果。

使用场景

- 按照K8S规范，使用私有镜像部署工作负载时必须为工作负载配置私有镜像的访问凭据，关于K8S集群中使用私有镜像的更多介绍请参考 从私有仓库拉取镜像 。
- 在CCE集群中安装免密组件后，会自动创建CCR企业版实例的镜像访问凭据，用户创建工作负载时不再需要显式配置ImagePullSecret即可通过内网拉取CCR企业版实例中的私有镜像。简化镜像配置流程，提高了CCE集群使用CCR镜像部署服务的效率。

限制说明

- **仅支持使用CCR实例的私有网络域名进行免密拉取，公网域名不支持免密拉取。**
- 仅支持 v1.18 及以上版本的Kubernetes 集群，且不支持ARM架构集群。
- CCR容器镜像服务在南京未开服，不支持在南京地域的CCE集群中使用免密组件。

注意事项

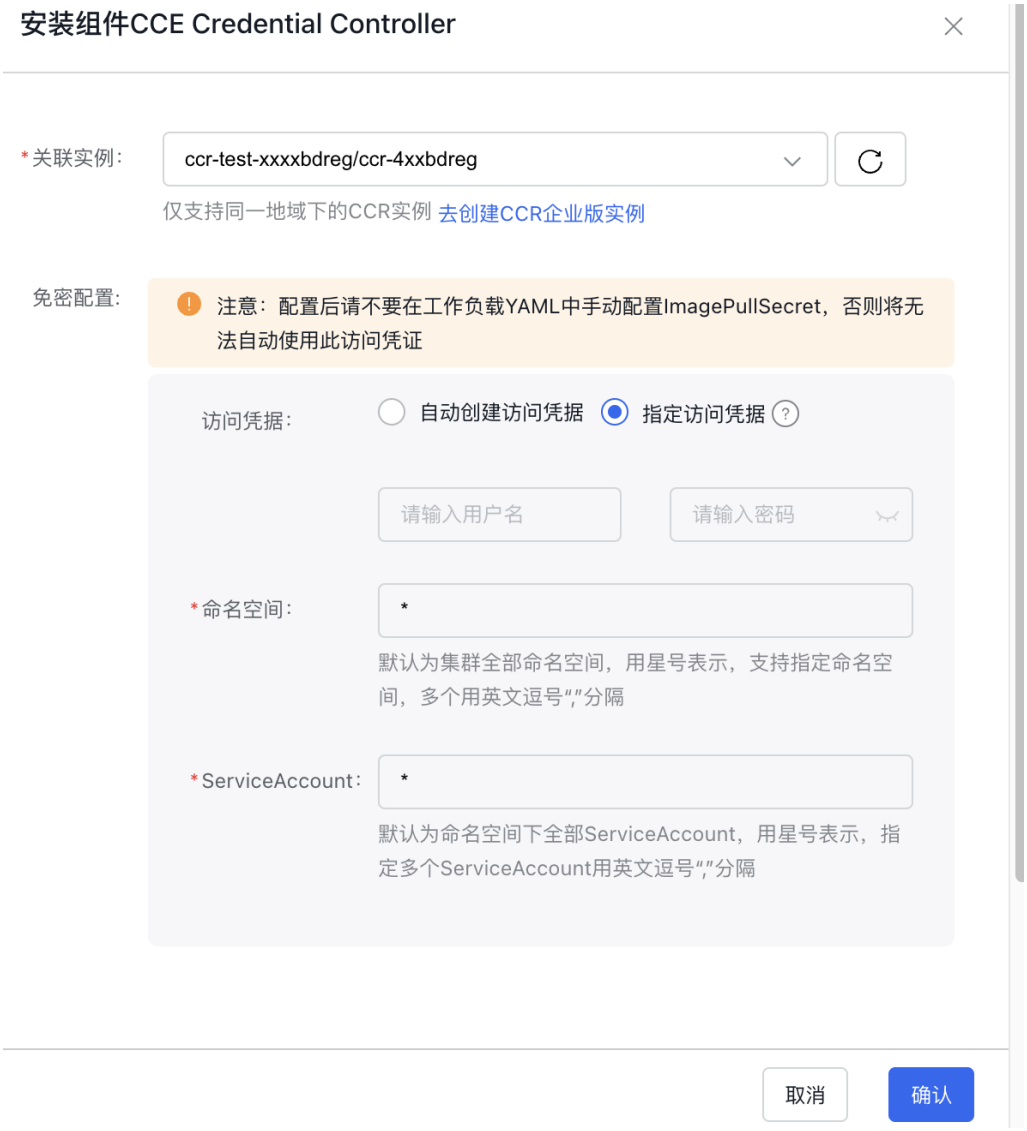
- 组件安装：免密组件安装过程会在CCE集群中创建一系列资源资源对象，请使用具有CCE集群管理权限的用户安装组件。
- 组件配置：在集群的系统命名空间kube-system下，免密组件默认只给名称为"default"的ServiceAccount关联ImagePullSecret，如果需要对kube-system命名空间下非"default" ServiceAccount配置免密，请使用指定ServiceAccount的方式。
- 组件工作：配置免密组件以后，在提交工作负载时请不要再手动配置ImagePullSecret，**手动配置后会覆盖默认的ImagePullSecret，将导致无法使用自动访问凭证拉取CCR实例镜像。**
- 组件卸载：**免密组件卸载后，原有自动配置的ImagePullSecret将会在24小时内过期**，已有关联的工作负载将无法免密从CCR拉取镜像。

安装组件

1. 登录百度智能云官网进入管理控制台，进入容器引擎CCE>集群管理>集群列表>组件管理,切换到镜像Tab页，在组件管理列表中选择CCE Credential Controller组件单击“安装”。

2. 填写组件安装参数：

- 关联实例：需要关联的CCR实例，仅支持同一地域下的CCR实例，如果尚未创建CCR实例点击 去创建CCR企业版实例。
- 访问凭据
  - 自动创建访问凭据：免密组件自动获取CCR实例的临时密码并创建创建ImagePullSecret，同时支持在临时密码过期前自动更新临时密码，保证ImagePullSecret有效。
  - 指定访问凭据：需要提供访问关联CCR实例的固定用户名密码（**注：该用户名密码是指在CCR控制台配置后用于docker login的用户名密码，非百度智能云账户的登录用户**），免密组件将使用该用户名密码区配置镜像访问凭据。由于固定密码无法校验校验有效性及自动更新，请确保用户密码及访问凭据持续有效，否则无法拉取镜像。
- 命名空间：免密组件生效的命名空间，默认*表示所有的命名空间都会生效，指定多个命名空间用逗号分隔。
- ServiceAccount：免密组件生效的ServiceAccount，默认*表示所有的都会生效，指定多个ServiceAccount用逗号分隔。



3. 点击"安装"按钮以后，等待组件安装成功。

使用免密组件提交工作负载

免密组件安装成功以后，在免密组件生效的命名空间下使用ServiceAccount创建工作负载时，会自动注入CCR实例的镜像访问凭据。

- 备注：在用户创建的工作负载当中没有明确指定ImagePullSecret和ServiceAccount的情况下，K8S 会尝试从当前命名空间下名为"default"的ServiceAccount资源中查找匹配合适的ImagePullSecret。

部署情况

安装 CCE Credential Controller 组件，将在CCE集群中部署以下对象

对象名称	所属namespace	所占资源	类型
cce-credential-controller	kube-system	cpu:500m memory: 500Mi	Deployment
cce-credential-controller	kube-system	-	ServiceAccount
cce-credential-controller	kube-system	-	Secret
cce-credential-controller	kube-system	-	ConfigMap
f0ec2fe2.baidubce.com	kube-system	-	ConfigMap
cce-credential-controller	kube-system	-	Service
cce-credential-controller	kube-system	-	MutatingWebhookConfiguration
cce-credential-controller	-	-	ClusterRole
cce-credential-controller	-	-	ClusterRoleBinding
cce-credential-controller	-	-	CCEImageCredential

版本记录

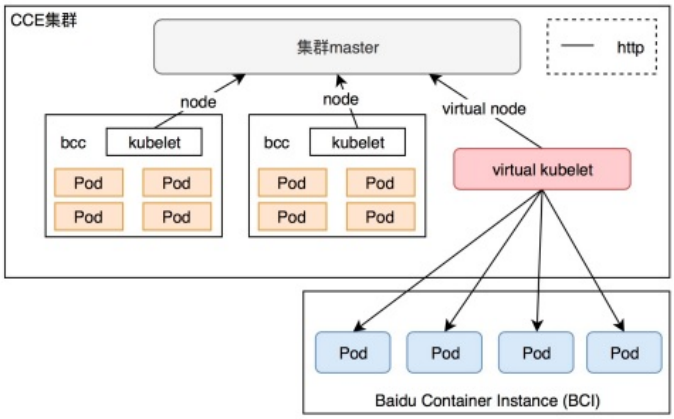
版本号	适配集群版本	更新时间	更新内容	影响
1.0.1	CCE/v1.18+	2024.09.10	首次上线	-

CCE Virtual Kubelet组件

当您需要在短时间内快速创建大量Pod时，BCC节点扩容速度可能无法满足要求，而预留额外的BCC节点又会产生资源浪费。可通过安装CCE Virtual Kubelet组件，该组件创建CCE虚拟节点，将Pod快速通过CCE虚拟节点调度到弹性容器实例BCI上运行，且无需购买和管理BCC节点。
本文介绍如何在CCE集群（CCE标准托管集群和CCE标准独立集群）中将Pod调度到BCI上运行。

组件介绍

CCE Virtual Kubelet 组件是基于社区开源项目Virtual Kubelet，实现Kubernetes与弹性容器实例BCI的无缝连接。Kubernetes集群可以轻松获得极大的弹性能力，而不必受限于集群的节点计算容量。您可以灵活地按需创建BCI Pod，无需自行规划集群容量。
CCE Virtual Kubelet 将应用Pod以Serverless容器（BCI）方式运行，提供弹性、免容量规划、按需使用、按需计费的能力。在Job类任务、CI/CD、Spark大数据计算、在线应用弹性等场景中可以显著提升应用部署的弹性效率，以及降低应用的计算成本。



适用场景

在BCI上运行Pod适合应对突发流量，也能降低计算成本。典型场景包括：

- 有明显波峰波谷的在线业务：例如在线教育、电商等业务的流量通常有明显的波峰波谷，使用BCI Pod可以更快速地应对突发流量，降低计算成本。
- 非持续运行的计算任务，使用BCI Pod运行计算任务，无需保留固定节点，仅需为任务执行期间的计算资源付费，降低计算成本，例如：
  - 数据计算：Spark大数据计算
  - CI/CD Pipeline
  - Job任务：例如定时任务、AI任务等。

组件功能

集群类型	组件类型	组件功能
CCE标准托管集群	提供托管免运维的CCE Virtual Kubelet组件	• 组件将创建虚拟节点，可实现与 BCI容器实例的无缝对接。 • CCE Virtual Kubelet 组件该组件默认被托管，不占用Worker节点资源。 • 如有托管组件相关问题，可提交工单 联系我们处理。
CCE标准独立集群	提供CCE Virtual Kubelet组件	• 组件将创建虚拟节点，可实现与 BCI容器实例的无缝对接。 • CCE Virtual Kubelet 组件将在kube-system命名空间下创建一个名为cce-virtual-kubelet的statefulset，运行在Worker节点上 • 如有相关问题，可 提交工单 联系我们处理。

使用说明

前提条件

- 已创建CCE标准托管集群或者CCE标准独立集群，且集群版本为1.18.9/1.20.8/1.22.5/1.24.4/1.26.9/1.28.8。如需创建集群则见创建集群。
- 已开通弹性容器实例服务并确认集群所在区域在BCI支持的地域列表内。可登录弹性容器实例控制台开通相应的服务，并查看当前支持的地域和可用区。 **步骤一：部署CCE Virtual Kubelet组件** 在CCE集群中，需要通过组件管理页面部署CCE Virtual Kubelet组件，根据两种不同集群类型，请按照如下不同步骤

集群类型	安装步骤
CCE标准托管集群	1. 登录容器服务管理控制台，在左侧导航栏选择**集群**。 2. 在**集群列表**页面，单击目标集群名称，然后在左侧导航栏，选择**运维管理 > 组件管理**。 3. 在**组件管理**页面的**核心组件**区域，选择**CCE Virtual Kubelet**组件，然后单击**安装**，并按照页面提示完成操作。安装时，使用集群默认子网和安全组作为CCE Virtual Kubelet的初始BCI配置参数，组件默认被托管，不占用Worker节点资源。安装成功后组件状态为**运行中**。 a. 若容器网络模式VPC-ENI的CCE集群，默认子网为容器网络子网，默认安全组优先使用ENI安全组，若无则使用Worker安全组 b. 若容器网络模式VPC路由模式的CCE集群，默认子网为VPC内第一个页"通用"类型子网，默认安全组为Worker安全组 4. 如需修改BCI配置参数，请参见下文"修改CCE虚拟节点配置"进行 相关操作。
CCE标准独立集群	1. 登录容器服务管理控制台，在左侧导航栏选择**集群**。 2. 在**集群列表**页面，单击目标集群名称，然后在左侧导航栏，选择**运维管理 > 组件管理**。 3. 在**组件管理**页面的**核心组件**区域，选择**CCE Virtual Kubelet**组件，然后单击**安装**，并按照页面提示完成操作。安装时，使用集群默认子网和安全组作为CCE Virtual Kubelet的初始BCI配置参数，组件CCE Virtual Kubelet 组件将在kube-system命名空间下创建一个名为cce-virtual-kubelet的statefulset，运行在Worker节点上。若顺利安装完成，则组件显示状态为**运行中**。 a. 若容器网络模式VPC-ENI的CCE集群，默认子网为容器网络子网，默认安全组优先使用ENI安全组，若无则使用Worker安全组 b. 若容器网络模式VPC路由模式的CCE集群，默认子网为VPC内第一个页"通用"类型子网，默认安全组为Worker安全组 4. 如需修改BCI配置参数，请参见下文"修改CCE虚拟节点配置"进行 相关操作。

注意：若您已通过Helm模板安装cce-virtaul-kubelet，暂不支持在组件管理安装，若您想通过组件中心安装则可 提交工单 联系我们处理。

步骤二：将Pod调度到BCI上运行 在虚拟节点启动Pod 相较于常规Pod，需要在Pod Spec中添加如下的nodeSelector和tolerations字段，才能将Pod调度到虚拟节点上。

Pod Spec示例：

```
spec:
nodeSelector:
  type: "virtual-kubelet"
tolerations:
- key: "virtual-kubelet.io/provider"
  operator: "Equal"
  value: "baidu"
  effect: "NoSchedule"
```

调度到虚拟节点上的Pod会以BCI Pod的形式启动。 更多BCI Pod配置项说明请参考 配置BCI Pod

相关操作 为了使用虚拟节点的一些更高级的特性，后续您可能需要升级CCE Virtual Kubelet组件或者修改CCE虚拟节点配置。当您不再需要使用虚拟节点时，也可以卸载CCE Virtual Kubelet组件。

升级CCE Virtual Kubelet组件

升级耗时约1分钟。升级过程中无法新建Pod，但对存量Pod不造成影响。升级步骤如下：

1. 登录容器服务管理控制台，在左侧导航栏选择集群。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择运维管理 > 组件管理。
3. 在组件管理页面的核心组件区域，选择CCE Virtual Kubelet组件，然后单击升级，并按照页面提示完成操作。

> **请注意：**若曾使用Helm模板安装组件的，暂不支持在组件中心升级，可通过Helm模板中心升级；若您想通过组件中心升级则可 提交工单 联系我们处理。

修改CCE虚拟节点配置 关于修改CCE虚拟节点配置的操作，请参见配置bci-profile。

卸载CCE Virtual Kubelet组件

1. 登录容器服务管理控制台，在左侧导航栏选择集群。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择运维管理 > 组件管理。
3. 在组件管理页面的核心组件区域，选择CCE Virtual Kubelet组件，然后单击卸载，并按照页面提示完成操作。若您的虚拟节点上仍存在BCI Pod,将提示无法卸载。

版本记录

版本号	适配集群版本	更新时间	更新内容	影响
0.3.16	CCE/1.18.9/1.20.8/1.22.5/1.24.4/1.26.9/1.28.8	2025.01.13	首次上线	-

CCE Onepilot 说明

使用前提

- 开通 APM 服务 APM 产品介绍

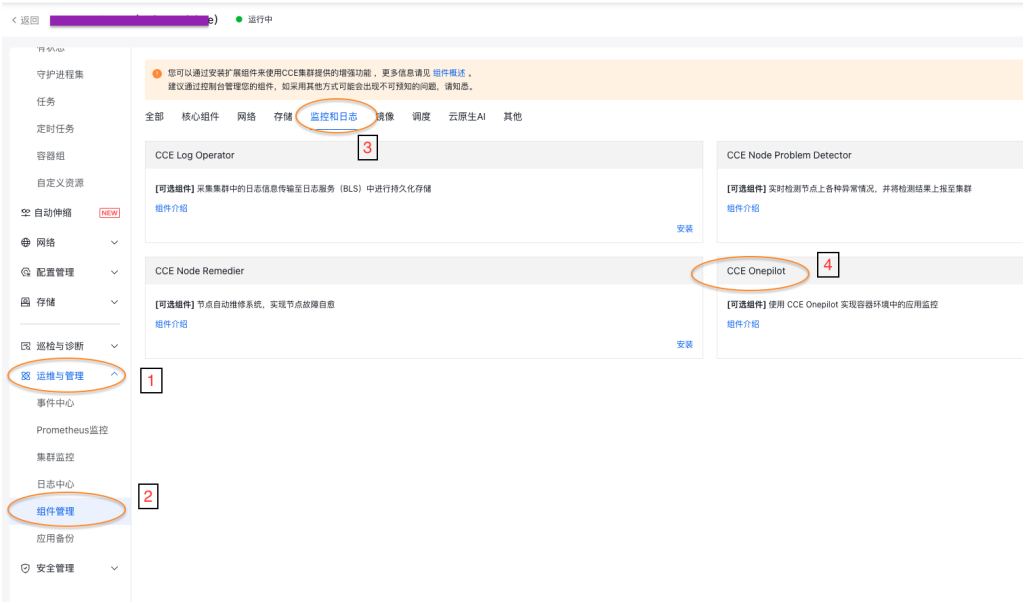
组件介绍 CCE Onepilot（应用性能监控接入）用于接入多编程语言探针，在 CCE 集群中实现自动注入 APM 探针以及配置指标上报环境。用户创建带有指定标签（Labels）的工作负载后，在 CCE 控制台可以方便跳转 APM，并查看对应应用实例的实时指标数据。

使用限制

- 当前 CCE Onepilot 支持针对 JAVA 应用的探针注入与指标上传
- 组件默认支持1000个Pod规模，集群Pod每超过1000个，组件资源对应的CPU请增加0.5核、内存请增加512MB。
- 使用依赖
  - 集群版本：1.16及以上版本（CCE支持的版本参照）
  - java 版本：JDK1.8（含）以上

安装组件

1. 进入集群详情页面（具体步骤见 查看集群）。
2. 集群详情页CCE Onepilot的查找路径为 运维管理->组件管理->监控和日志->CCE Onepilot 组件，具体参考下图。
3. 在组件管理监控和日志tab中选择 CCE Onepilot 组件单击“安装”。
4. 点击“确定”按钮完成组件的安装。



开启监控

- 在目标工作负载yaml文件的 spec.template.metadata 层级下，添加如下 labels：

```
labels:
 cceOnepilotEnable: "on"
 apmServiceName: "<your_service_name>" # 请用您应用实例的名称替换 <your_service_name>，若不设置此标签，则会以工作负载的名称 metadata.name 上报指标至 APM。
```

> CCE Onepilot 会识别以上标签，并对该 Pod 自动注入探针（agent）以及 APM-Token（进行用户认证，因此需要先开通 APM 服务）

- 添加后进入工作负载列表，找到目标工作负载，操作列点击【跳转APM】，您可以在APM中查看jvm等指标信息。

部署情况 | 名称 | namespace | 所占资源 | 类型 | |----------|----------|----------|----------| | cce-onepilot | cce-onepilot | - | Namespace | | cce-onepilot-server | cce-onepilot |LimitCPU: 250m LimitMemory: 512Mi| Deployment | | cce-onepilot-configuration | cce-onepilot | - | MutatingWebhookConfiguration| | cce-onepilot-service | cce-onepilot | - | Service | | cce-onepilot-tls | cce-onepilot | - | Secret | | cce-onepilot-cluster-role | cce-onepilot | - | Clusterrole | | cce-onepilot-role-binding | cce-onepilot | - | Binding | | cce-onepilot-serviceaccount | cce-onepilot | - | ServiceAccount |

版本记录 |版本号 | 适配集群版本 | 更新时间 | 更新内容 | 影响| |----- |----------|------|------|----| | 1.0.0 | CCE/v1.16 及以上 | 2025.04.15 | 首次上线| - |

备份中心

备份概述

容器服务CCE备份中心为容器化应用的备份、恢复与迁移提供了一站式解决方案，目前支持CCE标准集群资源对象的备份与恢复。本文将介绍备份中心的工作原理、计费说明、使用场景及使用流程。

工作原理

备份中心基于Velero实现集群中应用（资源YAML）的备份和恢复，并将备份数据安全地存储在指定的对象存储BOS Bucket中。

计费说明

备份中心功能本身不收费，使用备份中心时，会关联使用对象存储BOS服务，您需要支付相关的存储费用。

> **说明：**备份的集群资源（即相关YAML）将存储在BOS Bucket中。BOS相关计费请参见计费概述。

使用场景

- **备份恢复：**当集群或命名空间下的K8s资源被误删除时，可以通过备份数据快速恢复业务。
- **业务迁移：**可以将源集群备份的业务快速迁移到目标集群。
- **业务合规：**配合安全部门定期拉取备份数据进行业务审计。

使用流程

1. 登录容器引擎控制台,创建备份仓库，具体请参见备份仓库。
2. 在源集群中创建备份或定时备份策略备份指定的资源对象，具体请参见备份管理。
3. 在目标集群中根据备份数据创建恢复任务恢复指定的资源对象，具体请参见恢复管理。

备份仓库

备份中心使用对象存储BOS对备份数据进行存储。本文介绍如何创建备份仓库来存储备份数据。

前提条件

- 您已成功创建对象存储BOS Bucket，Bucket命名必须以“**cce-bos-**”开头，具体请参见创建Bucket，。

使用说明

- 不同地域的CCE集群可以使用相同的备份仓库存储数据，**推荐使用与集群相同地域的BOS Bucket存储备份数据，不同地域之间访问默认不通。**
- 删除仓库时，关联了该仓库的备份对象将无法正常执行恢复操作，请谨慎操作。
- 删除仓库时，关联的BOS Bucket中的底层存储资源不会被删除，需要您在BOS控制台进行操作。

操作步骤

1. 登录容器引擎控制台。
2. 在左侧导航栏选择**备份中心>备份仓库**进入备份仓库管理页面。
3. 单击**创建备份仓库**，完成相关配置。

配置项	必选/可选	配置说明
仓库名称	必选	自定义备份仓库的名称。
BOS Bucket地域	必选	选择BOS Bucket所在地域。
BOS Bucket名称	必选	选择备份仓库关联的BOS Bucket用于存储备份数据，BOS Bucket名称必须以“**cce-bos-**”开头
BOS Bucket子目录	可选	指定BOS Bucket的子目录，默认为“/”

4. 单击**确定**完成创建。

备份管理

您可以为集群中的Kubernetes资源进行备份，本文介绍如何针对目标集群创建备份任务和定时备份策略。

前提条件

- 在目标集群中已成功安装**CCE Backup Controller**备份组件，具体信息请参见CCE Backup Controller说明。

操作步骤

创建定时备份策略

定时备份策略将按照设置的定时规则，周期性地创建备份任务，直到该定时备份策略被删除。创建时间由定时规则决定，例如，每隔一段时间，或每天、每周、每月等特定的某个时间。

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**运维与管理 > 应用备份 > 备份管理**。
3. 开启后，在定时备份策略页面单击**创建定时备份策略**，在创建定时备份策略页面中完成相关配置。

配置项	必选/可选	配置说明
任务名称	必选	自定义定时备份任务的名称。
备份仓库	必选	选择备份任务关联的备份仓库用于存储备份数据。
定时规则	必选	指定自动创建备份任务的定时规则，支持Crontab表达式，格式为分 时 天 月 周，具体信息请参见如何使用Crontab。
备份范围	必选	**全部命名空间：**将备份集群中所有命名空间中的资源，其中kube-system、kube-publish、kube-node-lease、cce-back命名空间与集群强依赖，不建议直接进行备份恢复。因此，不支持备份这四个命名空间。 **指定命名空间：**将备份指定一个或多个命名空间下的资源
备份命名空间	必选	仅“备份范围”为**指定命名空间**时需要指定，支持选中一个或者多个备份命名空间。表示备份选中的一个或多个命名空间中的资源。
备份有效期	必选	备份的数据存储有效期，过期之后数据将被删除且无法恢复。取值范围为1~65536，单位为天。
排出命名空间	可选	仅“备份范围”为**全部命名空间**时需要指定，用于过滤出不需要备份的命名空间。
指定备份对象	可选	仅备份指定的Kubernetes资源对象，“全部资源对象”则表示备份命名空间中所有的资源对象。
排除备份对象	可选	仅“指定备份对象”为**全部资源对象**时需要指定，用于过滤出不需要备份的资源对象。
指定标签备份	可选	根据指定的标签进一步筛选资源对象，仅备份目标命名空间下带有该标签的资源。

4. 单击**确定**完成创建。

创建备份任务

立即备份将直接创建单个备份任务。备份任务创建时间为当前时间。

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**运维与管理 > 应用备份 > 备份管理**。
3. 在备份任务页面单击**创建备份任务**，完成相关配置。

配置项	必选/可选	配置说明
任务名称	必选	自定义备份任务的名称。
备份仓库	必选	选择备份任务关联的备份仓库用于存储备份数据。
备份范围	必选	**全部命名空间：**将备份集群中所有命名空间中的资源，其中kube-system、kube-publish、kube-node-lease、cce-back命名空间与集群强依赖，不建议直接进行备份恢复。因此，不支持备份这四个命名空间。 **指定命名空间：**将备份指定一个或多个命名空间下的资源。
备份命名空间	必选	仅“备份范围”为**指定命名空间**时需要指定，支持选中一个或者多个备份命名空间。表示备份选中的一个或多个命名空间中的资源。
备份有效期	必选	备份的数据存储有效期，过期之后数据将被删除且无法恢复。取值范围为1~65536，单位为天。
排除命名空间	可选	仅“备份范围”为**全部命名空间**时需要指定，用于过滤出不需要备份的命名空间。
指定备份对象	可选	仅备份指定的Kubernetes资源对象，“全部资源对象”则表示备份命名空间中所有的资源对象。
排除备份对象	可选	仅“指定备份对象”为**全部资源对象**时需要指定，用于过滤出不需要备份的资源对象。
指定标签备份	可选	根据指定的标签进一步筛选资源对象，仅备份目标命名空间下带有该标签的资源。

4. 单击**确定**完成创建。

查看备份信息

定时备份策略和立即备份都将在集群中下发备份任务，您可在**备份管理**页面查看**备份任务列表**和**定时备份策略**列表。

查看备份状态

您可以通过**备份任务列表**中的**状态**属性查看备份状态，状态说明如下：

状态	说明
初始化中	创建备份任务资源对象
执行中	正在执行备份任务。
成功	备份任务执行完成。
部分成功	备份出现部分资源对象成功，部分失败情况。可在控制台通过查看YAML中的status字段获取成功的对象数量，失败的原因等。
失败	备份任务执行失败。可在控制台或通过YAML的status字段查看失败原因。

恢复管理

您可以将备份的Kubernetes资源恢复到目标集群中，本文介绍如何针对已经创建了备份任务的集群进行恢复操作。

前提条件

- 在目标集群中已成功安装**CCE Backup Controller**备份组件，具体信息请参见CCE Backup Controller说明。
- 源集群中存在成功状态的备份任务。

使用限制

- 仅支持 Kubernetes 资源对象的恢复，不支持块存储CDS、负载均衡器BLB等云资源的恢复。

操作步骤

1. 登录容器引擎控制台。
2. 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择**运维与管理 > 应用备份 > 恢复管理**。
3. 在恢复任务页面单击**创建恢复任务**，完成相关配置。

配置项	必选/可选	配置说明
任务名称	必选	自定义恢复任务的名称。
备份仓库	必选	选择备份数据所在的备份仓库，需要通过备份仓库过滤出对应的备份任务。
选择备份	必选	选择需要恢复的备份任务。
恢复范围	必选	**全部命名空间：**恢复在备份任务中找到的所有命名空间下的资源对象。 **指定命名空间：**从备份任务中选择指定命名空间恢复资源。
排除命名空间	必选	仅"恢复范围"为**全部命名空间**时需要指定，用于过滤出不需要恢复的命名空间。
冲突处理	必选	**不覆盖（推荐）：**若目标集群被恢复的命名空间存在同名的备份资源时，则恢复任务不会覆盖已有资源。 **更新：**若目标集群被恢复的命名空间存在同名的备份资源时，则恢复任务会尝试对已有资源更新。

4. 单击**确定**完成创建。

> **说明：**
>
> - 恢复任务无法保证100%成功。

查看恢复状态

您可以通过**恢复任务列表**中的**状态**属性查看恢复状态，状态说明如下：

状态	说明
初始化中	创建恢复任务资源对象
执行中	正在执行恢复任务。
成功	恢复任务执行完成。
部分成功	恢复出现部分资源对象成功，部分失败情况。可在控制台通过查看YAML中的status字段获取成功的对象数量，失败的原因等。
失败	恢复任务执行失败。可在控制台或通过YAML的status字段查看失败原因。

备份恢复注意事项

- 恢复操作时，备份集群和恢复集群的容器网络模式（如VPC路由、VPC-ENI）请保持一致。
- 不支持恢复集群中的云存储资源，如CDS/BOS/DPFS等，涉及存储相关的Pod可能会由于找不到存储资源而Pending。
- 为提高恢复成功率，建议您在恢复时优先选择"指定命名空间"进行少量多次操作。
- Kubernetes不同版本之间会存在不兼容的变更（如参数废弃、APIServer 版本变更），为提高恢复成功率，恢复集群的Kubernetes版本与源集群尽量保持相同或相邻大版本，如备份集群1.18可恢复至目标集群1.18/1.20。
- 建议在执行恢复动作前预先检查恢复集群的资源情况，资源不足时可能会造成 Pod Pending。
- 备份集群和目标集群不在相同地域下需确保网络联通性，否则会造成镜像拉取失败造成Pod Pending。

巡检与诊断

集群巡检

容器引擎CCE提供集群巡检能力，可以帮助您发现容器服务集群中可能存在的潜在风险，包括：资源配额、集群风险、资源状态等（持续更新中），并针对异常巡检结果提供解决建议，提升运维效率。本文介绍如何使用集群巡检功能排查集群的潜在风险。

前提条件

- 已创建CCE集群。具体操作，请参见创建集群
- 已确保Kubernetes集群处于正常运行状态。访问CCE控制台，在集群列表页面，查看目标集群的集群状态。若集群状态为运行中，表示集群处于正常运行状态。

开启集群巡检

重要：使用集群巡检功能时，部分检查项将在您的容器集群内启动容器并采集检查项，采集的信息包括系统版本、负载、Docker、kubelet等运行

状态及系统日志中关键错误信息。数据采集程序不会采集您的业务信息及敏感数据。

1. 登录百度智能云管理控制台，进入“产品服务>云原生>容器引擎 CCE”，点击“集群管理>集群列表”，进入集群列表页面。
2. 单击目标集群名称，然后在左侧导航栏，选择“**巡检与诊断 > 集群巡检**”。
3. 在集群巡检页面右上侧，点击自动巡检&订阅配置，配置执行自动巡检的时间以及订阅报告的发送时间和接收方式。
4. 您也可以在集群巡检页面，单击开始巡检，手动巡检集群。巡检完成后，将在报告列表区域显示相关信息。

查看巡检结果

1. 登录百度智能云管理控制台，进入“产品服务>云原生>容器引擎 CCE”，点击“集群管理>集群列表”，进入集群列表页面。
2. 单击目标集群名称，然后在左侧导航栏，选择 **巡检与诊断 > 集群巡检**。
3. 在集群巡检页面的巡检报告列表区域右侧的操作列，单击目标巡检报告ID。

- 集群巡检会按照触发风险的程度分为低危、中危和高危。如果集群某些巡检项因为未知原因没有执行成功，则会显示“未知”，您可按需提交工单。
- 集群巡检详细内容包含风险级别、风险项名称、异常影响及解决方案。关于集群巡检的常见风险预警及修复方案的更多信息，请参见集群巡检项及解决方案。

4. 在检查报告页面，查看风险项、异常影响以及推荐的解决方案。

集群巡检项及解决方案

类型	巡检项	异常影响	修复建议
资源配额	VPC路由规则配额紧张	检查VPC内剩余路由表条目配额是否少于5条。 VPC路由模式下，集群每个节点都会消耗一条路由表规则，当路由表规则耗尽后，集群内无法添加新节点。（VPC-ENI模式下，集群不使用VPC路由表）	前往配额中心申请增加VPC路由规则配额。
	EIP实例配额紧张	检查集群所在地域可创建的个人/企业EIP实例数量是否小于5。 EIP配额不足可能影响集群、节点、服务开启公网访问。	前往配额中心申请增加EIP实例配额。
	ENI实例配额紧张	检查每个VPC可创建的未挂载主机的弹性网卡数量是否小于5。 ENI配额不足可能导致无法创建和加入节点。	前往配额中心申请增加ENI实例配额。
	BLB实例配额紧张	检查集群所在地域可创建的BLB实例数量是否小于5。 BLB配额不足可能影响service、ingress的创建。	前往配额中心申请增加BLB实例配额。
	BCC按需付费实例配额紧张	检查集群所在地域的按需付费BCC实例数是否大于95%。 配额不足影响节点创建。	前往配额中心申请增加按需付费BCC实例配额。
	CDS容量紧张	检查集群所在地域的CDS磁盘使用量占总容量（TB）是否大于95%。 配额不足影响节点及持久卷创建。	前往配额中心申请增加CDS容量配额。
	节点组实例规格可用库存紧张	检查节点组实例规格可用库存数量是否小于15。可用库存数量不足可能影响节点组伸缩。	BCC提工单增加实例规格可用库存 或 使用其他BCC实例规格
资源水位	集群可分配CPU是否充足	检查节点已分配 CPU 是否 > 80%。 当可分配 CPU 小于 Pod Request 值时，将无法创建 Pod。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	集群可分配内存是否充足	检查节点已分配内存是否 > 80%。 当可分配内存小于 Pod Request 值时，将无法创建 Pod。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	节点实时CPU使用率	检查节点 CPU 使用率是否 > 80%。 使用率过高可能导致 CPU 资源抢占，影响业务的正常	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，

	过高	运行。	找到resources字段，检查 Pod 资源配额（Request、Limit）。
	节点实时内存水位过高	检查节点内存使用率是否 > 80%。 使用率过高可能导致 OOM（Out of Memory），影响业务的正常运行。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	Pod CPU使用率过高	检查 Workload 的 CPU 负载是否 > 95%。 负载过高可能导致 CPU 资源争抢，影响业务的正常运行。	通过工作负载页面或使用kubectl编辑工作负载YAML，找到resources字段，调整资源配额（Request、Limit）。 通过工作负载页面点击伸缩增加期待Pod数，或使用kubectl编辑工作负载YAML增加实例数。 配置弹性伸缩策略（HPA）。
	Pod内存水位过高	检查 Workload 内存负载是否 > 95%。 负载过高可能导致 OOM（Out of Memory），影响业务的正常运行。	通过工作负载页面或使用kubectl编辑工作负载YAML，找到resources字段，调整资源配额（Request、Limit）。 通过工作负载页面点击伸缩增加期待Pod数，或使用kubectl编辑工作负载YAML增加实例数。 配置弹性伸缩策略（HPA）。
	节点磁盘使用率过高	检查节点磁盘使用率是否 > 80%。 使用率过高可能导致Pod被驱逐，影响业务的正常运行。	清理临时文件。 增加磁盘容量。
	节点根目录容量使用率过高	检查节点根目录容量使用率是否 > 80%。 使用率过高可能影响业务的正常运行。	清理临时文件。 增加磁盘容量。
	集群PFS使用率过高	检查PFS使用率是否 > 80%。 PFS使用率达到100%时，将无法向该文件系统写入增量数据，影响业务的正常运行。	PFS提工单扩容。
	VPC路由模式剩余Pod网段数不足	检查VPC路由模式下，集群剩余可用PodCIDR网段是否少于5个。每个节点消耗一个PodCIDR网段，集群可添加的节点少于5个。Pod网段耗尽后，新添加的节点将无法正常工作。	CCE提工单扩容。
	VPC-ENI模式子网剩余IP数不足	检查VPC-ENI模式下，集群内配置的子网剩余IP是否小于10个，每个Pod占用一个IP。当可用IP耗尽后，新创建的Pod分配不到IP，所以无法正常启动。	查看CCE集群详情，找到容器网络，添加子网。
	节点周CPU使用率过高	检查节点过去一周内CPU使用率是否 > 80%。 使用率过高可能导致 CPU 资源抢占，影响业务的正常运行。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	节点周内存水位过高	检查节点过去一周内平均内存使用率是否 > 80%。 使用率过高可能导致 OOM（Out of Memory），影响业务的正常运行。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	节点日CPU使用率过高	检查节点过去一天内CPU使用率是否 > 80%。 使用率过高可能导致 CPU 资源抢占，影响业务的正常运行。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
	节点日内存水位过高	检查节点过去一天内平均内存使用率是否 > 80%。 使用率过高可能导致 OOM（Out of Memory），影响业务的正常运行。	1.增加节点数量。 2.通过工作负载-容器组或使用kubectl获取Pod YAML，找到resources字段，检查 Pod 资源配额（Request、Limit）。
集群风险	集群Kubernetes版本过低	检查集群Kubernetes版本是否即将或者已经超出支持期限。 CCE仅保障支持最近三个Kubernetes双数版本的稳定运行。过期版本的集群存在运行不稳定和集群升级失败的风险。	参考 升级集群Kubernetes版本

	Node数量是否超过规格	检查集群Node数量是否超过集群规格限制。 Node数量超过规格限制可能导致控制面资源耗尽和节点组扩缩容失败。	CCE 提工单升级集群规格。
	集群删除保护是否开启	检查集群是否开启删除保护功能。 如未开启，集群有可能被控制台或 API 误删除，导致业务故障。	开启集群删除保护。（点击集群详情-基本信息-集群删除保护）
	审计日志是否开启	检查检查审计日志是否开启。 开启集群审计日志方便日常排查问题。	开启集群审计。
	集群worker node (ready)数量	检查集群中 Worker 节点的数量是否少于2 个。 单个节点的集群存在单点失效问题。	添加节点。
	CoreDNS 组件状态	检查 CoreDNS 组件是否为 非 Running 状态。 该组件异常会导致集群内 DNS 解析错误，无法通过 Service 名称进行访问。	检查 CoreDNS 组件状态，排除异常原因。
	CoreDNS 是否有新版本	检查CoreDNS是否有最新版本。 集群内CoreDNS组件版本过低，会导致业务出现DNS解析问题。最新版的CoreDNS提供了更好的稳定性配置和新的功能。	升级CoreDNS（集群左侧导航点击运维与管理-组件管理-网络-CoreDNS组件，在组件右下角找到升级按钮）。手动升级具体参考：https://cloud.baidu.com/doc/CCE/s/glto9zt0l
	CoreDNS 高可用	检查 CoreDNS 组件的副本数是否大于2个，且不同副本部署在不同的节点中。 如未达到预期要求，则 CoreDNS 不具备高可用性，存在单点失效风险。当节点宕机或重启的时，CoreDNS 将无法提供服务，影响业务正常运行。	检查 CoreDNS 副本状态，保持 2 个以上副本，并将副本打散到不同的节点上。
	DNS Service是否正常	检查集群DNS服务的Cluster IP是否正常分。 集群DNS服务异常会造成集群功能异常，影响业务。	检查CoreDNS Pod运行状态和运行日志，排查DNS问题。
	APIServer BLB 6443端口监听配置是否正常	检查集群API Server BLB 6443端口监听配置。 若配置异常，将导致集群无法访问。	1.前往BLB应用型实例页面找到集群关联的BLB实例，检查BLB实例监听设置。 2.如果找不到BLB实例，请提CCE工单。
	APIServer BLB 实例是否存在	检查集群API Server负载均衡实例是否存在。 若集群API Server负载均衡实例不存在，会造成集群不可用。	1.前往BLB应用型实例页面检查集群关联的BLB实例是否存在。 2.如果找不到BLB实例，请提CCE工单。
	APIServer BLB 实例状态是否正常	检查集群API Server BLB实例状态。 若实例状态异常，将会影响集群可用性。	1.前往BLB应用型实例页面找到集群关联的BLB实例，在实例详情里检查BLB实例状态。 2.如果找不到BLB实例，请提CCE工单。
	节点Kubelet组件版本低于控制面版本	检查节点Kubelet组件版本是否低于控制面版本。 Kubelet版本低于控制面版本可能导致兼容性问题和安全性问题。	升级Kubelet版本。
	安全组规则	检查节点安全组出/入方向规则是否满足集群访问权限需求。 安全组规则不满足集群访问权限需求可能影响容器网络连通性。	前往VPC访问控制-安全组页面，调整安全组规则。
	节点是否关联CCE安全组	检查集群节点是否关联CCE安全组。 未关联CCE安全组可能影响容器网络连通性。	找到目标BCC实例并查看实例详情，在实例安全组界面选择网卡并绑定CCE默认安全组。
	APIServer BLB 6443端口目标组配置是否正常	检查集群API Server BLB 6443端口对应目标组是否配置正常。 若配置异常，可能导致集群无法访问。	1.前往BLB应用型实例页面找到集群关联的BLB实例，检查BLB实例目标组的配置。 2.如果找不到BLB实例，请提CCE工单。
	APIServer Loopback证书过期	检查API Server Loopback证书是否过期。 如果过期，可能影响API Server内部通信。	重启APIServer。
组件风险	集群组件状态是否正常	检查组件管理的组件（已经安装的组件）是否符合预期状态。 组件状态不正常可能无法提供对应服务，影响业务正常运行。	检查组件状态。
	集群组件是否有最新	检查集群中关键组件是否需要更新版本。	

	版本	新版本的组件提供了新功能和更好的稳定性。	升级组件。
资源状态	节点状态	检查集群中是否存在 NotReady 的节点。 如果节点状态异常，会导致 Pod 无法被调度到该节点上。	查看节点状态，必要时扩容或缩容节点。
	Workload 副本数	检查工作负载的期望副本数和实际副本数是否一致。 如不一致，则不满足高可靠性要求。	检查副本数异常工作负载，排除异常原因，更新副本数。
	DaemonSets 状态检测（检查 DaemonSets 数量是否和节点数量一致）	检查 DaemonSets 数量是否和节点数量一致。 如未达到预期副本数，则可能导致相关功能异常。	检查副本数异常原因，排除异常原因，更新副本数。

GPU运行环境检查

概述

本文档旨在指导如何对包含加速芯片（如GPU）的节点运行环境的集群检查。通过此检查，您可以确保这些节点在运行AI任务时具备必要的硬件和软件条件，从而提高任务的执行效率和成功率。

前置条件

- 已创建CCE集群。具体操作，请参见创建集群
- 包含加速芯片（如GPU）的节点已处于可用状态

注意事项

- 检查过程可能需要较长时间，请耐心等待。
- 检查期间，不建议进行其他对集群的操作，以免影响结果。

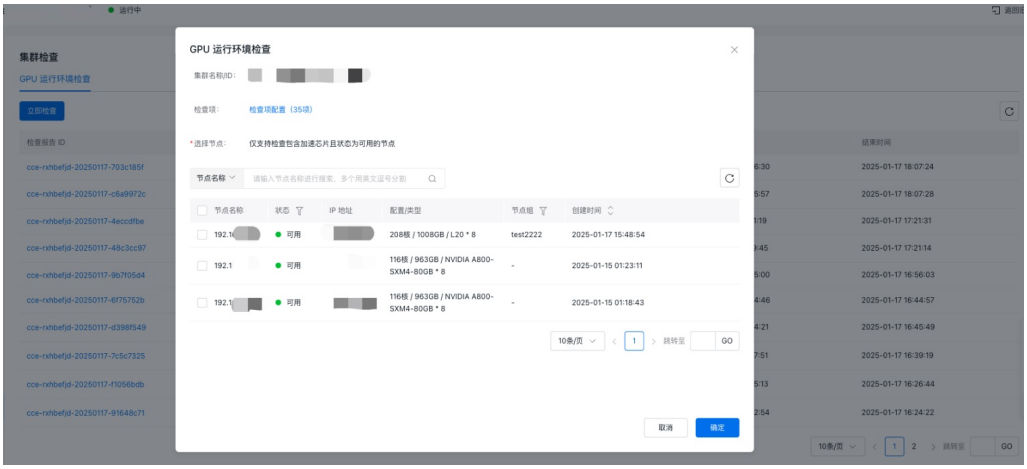
操作步骤

1.登录百度智能云管理控制台，进入“产品服务>云原生>容器引擎 CCE”，点击“集群管理>集群列表”，进入集群列表页面。

2.单击目标集群名称，然后在左侧导航栏“巡检与检查”，选择“集群检查”。

3.在集群检查>GPU运行环境检查页面，点击“立即检查”

4.在弹出的对话框中，配置检查项范围，然后选择需要检查的节点。



5.点击“开始检查”以启动检查流程。

6.检查完成后，系统将生成检查报告，用户可查看检查结果和建议。

GPU运行检查项说明

一级分类	二级分类	检查项描述
计算机软件故障	Has agent健康性检查	检查Has agent的运行情况，Has正常运行才能提供硬件故障上报能力
	Has agent版本检查	检查节点安装的Has agent版本是否过低，版本过低会影响故障上报最新功能的使用和准确性
	Has agent 加速芯片检测开启检查	检查Has agent加速芯片故障检测能力是否开启
	网卡固件版本检查	检查多机间网卡固件版本是否一致
	加速芯片初始化状态检查	检查是否存在未正确初始化或存在问题的加速芯片节点。修订版本可能导致设备存在问题，不能正常工作
	加速芯片预期数量检查	检查节点中感知的加速芯片数量，并与预期数量进行比较确保设备正常工作
	加速芯片固件版本检查	检查多机间加速芯片固件版本是否一致
	持久化模式开启检查	检查加速芯片是否开启了持久化模式，未开启持久化会导致加速芯片功耗增加、性能下降
OS	OS & Kernel版本检查	检查节点间的OS和内核版本是否一致
	内核参数检查	检查PFS-L1设置的内核参数是否正确，阈值设置错误会影响存储设备性能
	sGPU内核依赖检查	检查节点内核版本是否支持开启GPU虚拟化功能，较低的内核版本会导致虚拟化失败
	图形界面关闭检查	检查图形界面是否关闭，图形界面未关闭会导致节点死机问题
	Kubelet资源预留检查	检查Kubelet资源预留情况，如果预留资源过少会导致节点负载过高时节点假死
	home盘挂载检查	检查home盘挂载情况，home盘未正确挂载会导致数据盘不识别等存储问题
	镜像目录、根目录资源预留检查	检查镜像目录、根目录资源大小，如果过小，会导致镜像拉取失败，集群无法创建任务
驱动	nouveau驱动禁用检查	检查nouveau驱动禁用情况，该驱动不禁用会导致与加速芯片驱动冲突，影响节点运行和创建任务
	网卡驱动检查	检查网卡驱动版本，如果存在异常版本会导致集群通信出现故障
	fabric-manager安装检查	检查fabric-manager组件安装情况，该组件异常会导致无法正常提交训练任务
	link_status健康状态检查	检查link_status健康状态，该状态异常会导致多机间通信失败和多机任务无法运行
	peermem配置检查	检查加速芯片peermem包安装情况，该配置未安装会导致显存管理异常，引起任务性能下降
	hALT配置检查	检查hALT配置情况，hALT未配置会导致物理机发生重启掉电
	网卡抖动参数配置	检查网卡抖动参数是否配置，抖动参数未配置会导致网络性能下降
	FW版本检查	检查多机间FW版本检查是否一致
	BMC版本检查	检查多机间BMC版本检查是否一致
	ECC Correctable计数检查	检查ECC Correctable计数是否大于阈值，该数值异常说明存储可能存在硬件错误
网卡	MTU配置检查	检查MTU参数配置，该配置不正确会导致网络降速
	RDMA网卡mac地址小写检查	检查RDMA网卡mac地址是否为小写
	GID index一致性检查	检查所有ROCE网卡v2 ipv4的GID index是否一致
	多网卡IP配置检查	检查所有网卡的IP配置，该配置不正确会导致RDMA网络不通
	多网卡IP rule配置检查	检查所有网卡的IP rule配置，该配置不正确会导致RDMA网络不通
	加速芯片NV Link状态检查	检查节点加速芯片的 NVLink 状态，包括活跃状态、数量和带宽
	加速芯片NV Link连接拓扑检查	检查多个节点加速芯片间是否有正确的NV Link连接
集群组件&状态	节点状态	检查集群节点的运行情况，正常运行才能有效提交训练任务和部署推理服务
	节点污点、封锁信息	检查节点的污点和封锁情况，保障信息正确以便更好使用训练任务的容忍能力
	组件部署状态	检查组件Pod的存活情况，正常运行才能有效提交训练任务和部署推理服务

故障诊断

概述

百度云容器引擎（CCE）故障诊断功能提供自动化异常检测与根因定位能力，支持对集群核心组件的健康状态进行系统性检查，帮您快速识别常见配置错误、资源瓶颈及组件故障。

前提条件

- 已创建CCE集群。具体操作，请参见创建集群
- 已确保Kubernetes集群处于正常运行状态。

诊断功能介绍

当前版本重点覆盖 **节点级** 与 **Pod级** 资源异常诊断，详细诊断项参考文档下方说明。

开启故障诊断

重要：故障诊断功能采集的信息包括系统版本、负载、docker、kubelet等组件运行状态，系统日志中关键错误信息。诊断全程遵循数据安全规范，不涉及任何业务及敏感数据。

节点诊断和Pod诊断操作类似，下文以开启节点诊断为例，介绍如何使用故障诊断功能。

方式一：

1. 登录百度智能云管理控制台，进入“产品服务>云原生>容器引擎 CCE”，点击“集群管理>集群列表”，进入集群列表页面。
2. 单击目标集群名称，然后在左侧导航栏【巡检与诊断】中，选择“故障诊断”。
3. 在集群巡检选择【节点诊断】Tab，单击【立即诊断】
4. 在节点诊断弹窗选择节点名称，仔细阅读注意事项后选中我已知晓并同意，然后单确认发起诊断。

   方式二：
5. 在目标集群的节点列表页，操作列选择【故障诊断】
6. 跳转到节点诊断弹窗后，仔细阅读注意事项后选中我已知晓并同意，然后单确认发起诊断。

发起诊断后，您可以根据任务诊断中的状态查看诊断进展。

查看诊断结果

在故障诊断页面诊断列表页，单击目标诊断报告对应的诊断详情，在诊断详情页面查看详细诊断结果。

节点诊断项及说明

Node 节点

诊断项名称	诊断项说明	修复方案
节点状态	确认节点状态为 Ready。	请尝试重启节点
节点调度状态	确认节点未被标记为不可调度。	节点不可调度，请检查节点封锁设置。具体操作，请参见设置节点封锁。
BCC 实例是否存在	检查节点对应的 BCC 实例存在。	检查BCC实例状态。
BCC 实例健康状态	检查节点对应的 BCC 实例运行正常。	检查BCC实例状态。
节点 CPU 使用率	检查节点当前 CPU 使用率处于正常范围。	无
节点内存使用率	检查节点当前内存使用率处于正常范围。	无
节点周 CPU 水位	检查节点过去一周 CPU 使用率未持续过高，避免资源争抢影响业务。	为避免业务受到影响，请设置合理的Pod request和limit，避免一个节点上运行的Pod过多。
节点周内存水位	检查节点过去一周内存使用率未持续过高，避免 OOM 影响业务。	为避免业务受到影响，请设置合理的Pod request和limit，避免一个节点上运行的Pod过多。
节点 OOM 事件	检查节点未发生 OOM (内存溢出) 事件。	登陆节点，查看Pod所在节点的内核日志/var/log/messages
Kubelet 状态	检查节点 Kubelet 运行正常。	请检查节点Kubelet日志。
Containerd 状态	检查节点 Containerd 服务运行正常。	登陆节点，查看节点的内核日志/var/log/messages
Docker 状态	检查节点 Docker 服务运行正常。	登陆节点，查看节点的内核日志/var/log/messages
Docker hang检测	检查节点未发生 Docker hang情况。	请尝试登录节点，通过命令systemctl restart docker重启Docker。
API Server 连接	检查节点可正常连接集群 API Server。	请检查集群相关配置。
节点 DNS 服务	检查节点可正常使用主机 DNS 服务。	请检查主机DNS服务是否正常。更多信息，请参见DNS问题排查指南。
集群 DNS 服务	检查节点访问集群kube-dns服务的Cluster IP，正常使用集群的DNS服务	请检查CoreDNS Pod运行状态和运行日志。更多信息，请参见DNS问题排查指南。
集群CoreDNS Pod 可用性	检查节点可正常访问集群 CoreDNS的Pod IP地址。	请检查节点能否正常访问CoreDNS的Pod IP地址。
Containerd 镜像拉取	检查节点 Containerd 可正常拉取镜像。	请检查节点网络及镜像配置。
Docker 镜像拉取状态	检查节点 Docker 可正常拉取镜像。	请检查节点网络及镜像配置。

NodeComponent 节点核心组件

诊断项名称	诊断项说明	修复建议
CNI 组件状态	检查节点 CNI组件运行正常。	请提交工单处理。
CSI 组件状态	确认节点 CSI 组件运行正常。	请前往CCE集群-运维与管理-组件管理-存储，检查集群件存储组件状态。
Network Agent 状态	确认节点 Network Agent 服务运行正常。	请提交工单处理。
Network Operator 状态	确认集群 Network Operator 服务运行正常。	请提交工单处理。

ClusterComponent 集群组件

诊断项名称	诊断项说明	修复建议
Pod网段余量	确认 VPC路由模式下集群剩余可用 PodCIDR 网段大于 5 个，避免 Pod 网段耗尽导致新节点无法正常工作。	请提交工单处理。
DNS 服务 ClusterIP	确认集群 DNS 服务的 Cluster IP 已正常分配，DNS 服务异常将导致集群功能异常，影响业务	请检查CoreDNS Pod运行状态和运行日志。更多信息，请参见DNS问题排查指南
API Server BLB 实例状态	确认 API Server BLB 实例运行正常。	前往BLB应用型实例页面找到集群关联的BLB实例，在实例详情里检查BLB实例状态。
API Server BLB 实例是否存在	确认 API Server BLB 实例存在。	前往BLB应用型实例页面检查集群关联的BLB实例是否存在。
API Server BLB 6443 端口监听	确认 API Server BLB 6443 端口监听配置正确。	前往BLB应用型实例页面找到集群关联的BLB实例，检查BLB实例目标组的配置。
节点与容器子网可用区一致性	确认 VPC-ENI 模式下节点和容器子网位于同一可用区。	查看CCE集群详情，找到容器网络，添加子网并确保节点和子网位于同一可用区。
子网 IP 余量	确认 VPC-ENI 模式下子网剩余 IP 数量充足。	查看CCE集群详情，找到容器网络，添加子网。

GPUNode GPU 节点

诊断项名称	诊断项说明	修复建议
GPU节点状态	检查节点GPU综合状态是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU节点可分配GPU数	检查节点可分配GPU数是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID48Error	检查 NVIDIA GPU Double Bit ECC Error。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID62Error	检查 NVIDIA GPU Internal micro-controller halt (newer drivers)。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID64Error	检查 NVIDIA GPU ECC page retirement or row remapper recording failure。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID74Error	检查 NVIDIA GPU NVLINK Error。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID79Error	检查 NVIDIA GPU 掉卡：GPU has fallen off the bus。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID95Error	检查 NVIDIA GPU Uncontained ECC error。	请尝试重启GPU节点，重启未恢复请提交工单处理
NVIDIA XID109Error	检查 NVIDIA GPU Context Switch Timeout Error。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XID140Error	检查 NVIDIA GPU Unrecovered ECC Error。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA XIDError	检查 NVIDIA GPU XID错误。	请尝试重启GPU节点，重启未恢复请提交工单处理
NVIDIA SXIDError	检查 NVIDIA GPU SXID错误。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA 存在行重映射失败	检查 NVIDIA GPU 是否存在行重映射失败。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA Device Plugin掉卡	检查 NVIDIA Device Plugin 是否上报掉卡。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA InfoROM情况	检查 NVIDIA GPU infoROM 是否已损坏。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA ECC错误	检查 NVIDIA GPU ECC错误。	请尝试重启GPU节点，重启未恢复请提交工单处理。
		请尝试重启GPU节点，重启未恢复请提交工

NVIDIA GPU高温告警	检查 NVIDIA GPU 温度是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA GPU运行模式	检查 NVIDIA GPU 运行模式是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA-SMI状态码	检查 nvidia-smi 状态码。	请尝试重启GPU节点，重启未恢复请提交工单处理。
PCI配置读写	检查 PCI 配置是否读写失败。	请尝试重启GPU节点，重启未恢复请提交工单处理。
PCI地址访问	检查 lspci 是否能够读取 GPU 配置空间。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU带宽	检查 GPU 带宽是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU功耗告警	检查 GPU 功耗是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU驱动无法访问	检查 GPU 驱动是否能够正常访问。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA GPU识别情况	检查总线的 NVIDIA GPU 是否能够被驱动和nvidia-smi识别。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA-Container-Toolkit版本	检查 NVIDIA-Container-Toolkit 版本是否符合集群版本。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA-Container-Toolkit配置	检查 NVIDIA-Container-Toolkit 在容器运行时配置是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA-Container-Toolkit状态	检查 NVIDIA-Container-Toolkit 状态是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU节点上的异常进程	检查 GPU 节点上是否存在异常进程。	请尝试重启GPU节点，重启未恢复请提交工单处理。
HAS状态	检查 HAS 状态是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
HAS版本	检查 HAS 版本是否支持。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA 掉卡情况	检查 NVIDIA GPU 是否已从总线上掉落或不可访问。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA ECC错误超过限制	检查 NVIDIA GPU ECC 内存校验错误是否超过限制。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA GPU互联链路方式	检查 NVIDIA GPU 互联通信链路方式是否正常（SYS或NODE方式链接将导致降速 ）。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA GPU互联链路告警	检查 NVLink & NVSwitch错误	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA GPU互联服务错误	检查 GPU 互联服务 FabricManager 是否正常。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVLink状态	检查 NVLink 是否中断或失活。	请尝试重启GPU节点，重启未恢复请提交工单处理。
CUDA版本	检查 CUDA 版本是否支持。	请尝试重启GPU节点，重启未恢复请提交工单处理。
GPU驱动版本	检查 GPU 驱动版本是否支持。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVIDIA设备电源线连接	检查NVIDIA GPU 设备电源线是否正确连接。	请尝试重启GPU节点，重启未恢复请提交工单处理。
NVLink数量	检查 NVLink的links数量是否减少。	请尝试重启GPU节点，重启未恢复请提交工单处理。
		请尝试重启GPU节点，重启未恢复请提交工

GPU与网卡连接类型	检查网卡位置是否插错。	单处理。
节点网卡状态	检查网卡综合状态。	请尝试重启GPU节点，重启未恢复请提交工单处理。
网卡PCI地址不可用	检查网卡 PCI 地址是否不可用。	请尝试重启GPU节点，重启未恢复请提交工单处理。
网卡通道数量	检查网卡通道数量是否达到支持的最大值。	请尝试重启GPU节点，重启未恢复请提交工单处理。
网卡带宽	检查网卡带宽是否达到支持的最大值。	请尝试重启GPU节点，重启未恢复请提交工单处理。

Pod诊断项及说明

诊断项名称	诊断项说明	修复建议
Pod 容器重启次数	统计 Pod 内容器的重启次数，识别异常重启情况。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod 容器镜像下载	检查 Pod 所在节点是否存在其他 Pod 的镜像下载阻塞，避免资源竞争。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod 容器镜像拉取 Secrets 有效性	检查 Pod 拉取镜像所需的 Secrets 有效，避免镜像拉取失败。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod 内存使用情况	检查 Pod 内存使用率 ≤ 95%，避免内存过高导致 OOM 影响业务	1.通过工作负载页面或使用kubectl编辑工作负载YAML，找到resources字段，调整资源配额（Request、Limit）。 2.通过工作负载页面点击伸缩增加期待Pod数，或使用kubectl编辑工作负载YAML增加实例数。 3.配置弹性伸缩（HPA）。
Pod CPU使用情况	检查 Pod CPU 使用率 ≤ 95%，避免 CPU 过高导致资源争抢影响业务。	1.通过工作负载页面或使用kubectl编辑工作负载YAML，找到resources字段，调整资源配额（Request、Limit）。 2.通过工作负载页面点击伸缩增加期待Pod数，或使用kubectl编辑工作负载YAML增加实例数。 3.配置弹性伸缩（HPA）。
Pod到CoreDNS Pods的连通性	检查 Pod 可正常访问 CoreDNS Pods	检查Pod到CoreDNS Pods的连通性。
Pod到CoreDNS Service的连通性	检查 Pod 可正常访问 CoreDNS Service	检查Pod到CoreDNS Pods的连通性。
Pod到主机网络DNS服务器的连通性	检查 Pod 可正常访问主机网络 DNS 服务器	检查Pod到主机网络DNS服务器的连通性。
Pod初始化状态	检查 Pod 已完成初始化，进入正常运行阶段。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod调度状态	检查 Pod 已被成功调度到目标节点。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod是否可调度	检查 Pod 满足调度条件，可被调度到合适节点。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。
Pod状态	检查 Pod 当前状态符合预期，例如 Running、Pending 等。	请检查Pod状态及日志。更多信息，请参见Pod异常问题排查。

典型实践

CCE使用检查清单

概览

CCE 提供基于原生 Kubernetes 的容器管理服务，为方便用户更好的使用 CCE，我们从集群、应用、问题排查三个方面，总结出来一些典型实践的 checklist，强烈建议 CCE 用户在开始使用或服务上线前，能对照 checklist 过一遍，以帮忙您顺利的将服务迁移到 CCE 上，降低因为使用不当导致应用异常或需重建集群的风险。

集群检查项

类型	项目	建议	参考文档
集群	节点数	不管服务规模多小，对于线上服务，强烈建议集群节点数至少大于1，且需要预留一定的资源buffer，避免单点故障导致业务受损	
	节点密码	节点的 root 密码请务必设置为强密码	
	节点网络	如果集群有对外访问需求，不建议节点直接绑定 EIP，节点对外暴露有安全风险，可以通过为集群VPC网络绑定NAT网关实现访问公网的能力。	容器网络通过 NAT 网关访问公网
	VPC 路由	CCE 容器网络实现依赖 VPC 路由，CCE 创建的路由规则描述为 auto generated by cce ，新建路由请不要和 CCE 的路由规则冲突，如无法避免，可以发工单进行咨询。	
	安全组	如果有设置安全组需求，安全组需要放通节点网络、容器网络、以及网段100.64.230.0/24和端口22、6443、30000-32768，否则可能出现容器引擎网络不通的问题。	
	磁盘容量	创建集群时，强烈建议为节点挂载至少 100GB 的 CDS（CCE 已默认勾选）。	
	节点升降配	考虑到升降配需要重启机器，可能导致集群容量不足，CCE 未直接支持虚机的升降配，用户可在 BCC 页面操作，但还是建议用户通过先扩容再缩容的方式实现，减少业务影响。	
	虚机监控	虚机 CPU、MEM、磁盘使用率等过高，会影响集群稳定性，CCE 有 Evicted 机制，在节点负载过高时，会将一部分实例迁移走，所以强烈建议用户在 BCM 给节点添加监控报警。	BCM 添加报警
	百度智能云三方资源	强烈建议 CCE 用户不要直接在 BCC、DCC、VPC、BLB、EIP 等产品页，修改 CCE 创建资源的相关配置，包括名字等，可能导致非预期结果。	

应用检查项

类型	项目	建议	参考文档
应用	镜像	建议用户构建 Docker 镜像时，可以在镜像中安装一些常用的 debug 工具，比如：ping、telnet、curl、vim 等，可自行定制。	
	私有镜像	容器使用私有镜像时，需要设置 secret。	CCE集群中使用私有镜像实践
	实例副本数	如果是无状态服务，且不存在冲突情况，建议实例副本 > 2，避免单点故障造成实例迁移，服务短暂不可用。	
	资源限制	强烈建议所有上线的服务，必须设置 resource.limits 。	kubernetes 资源限制
	健康检查	建议所有上线的服务，设置 livenesss 和 readiness probe 健康检查方式，保证服务自动 Failover。	kubernetes 健康检查
	服务暴露方式	集群内访问：ClusterIP Service； 集群外访问：LB Service； 集群外访问（HTTP/HTTPS）：Ingress	LoadBalancer 接入网络流量 Ingress 接入网络流量
	服务数据持久化	服务有数据持久化需求，建议通过 PV 和 PVC 方式使用，目前 CCE 已经支持通过 PV/PVC 方式使用文件存储(CFS)、块存储(CDS)、对象存储(BOS)。	通过 PV/PVC 方式使用 CFS 通过 PV/PVC 方式使用 CDS 通过 PV/PVC 方式使用 BOS

常见问题排查

1. 容器启动失败 ？

通常可通过下面两种方式查看错误信息：

1. kubectl describe podName
2. kubectl logs podName

如果通过上述方式看不到明显错误，可以在 YAML 中修改容器的启动命令，比如设置为 sleep 3600：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: hub.baidubce.com/cce/nginx-alpine-go:latest
        command: ["/bin/sh", "-c", "sleep 3600"]
```

启动服务后，通过 kubectl exec -it podName /bin/sh 进入容器，手动执行启动命令，查看服务报错信息。

2. LoadBalancer Service 创建失败 ？

可以通过 kubectl describe service serviceName，查看Events排查失败原因，一般是 EIP、BLB 的配额超过限制，可以发工单申请加大配额。

注意：用户可以购买的EIP实例数 <= 当前已存在的BCC实例数 + 当前已存在的BLB实例数 + 2

3. 容器网络访问失败？

容器网络访问失败具体表现分很多情况：

- Serivce EIP 无法访问；
- ServiceName 在容器中无法访问；
- Service ClusterIP 集群中无法访问；
- PodIP 在集群中无法访问；
- ...

容器网络问题，一般最终都是 PodIP 不通，导致 Service 的各种访问问题。首选分别在节点和 Pod 中检查 PodIP 是否可以 ping 通，如果不行，需要检查两个地方：

- 查看 VPC 路由表，确认是否有路由规则和 CCE 冲突；
- 查看 VPC 安全组，是否有策略将请求拦截。

如果确定没有，可以发工单联系管理员进行排查。

注意：Service clusterIP 不能直接 ping 通，需要通过 ip:port 访问；PodIP 可以 ping 通。

CCE-访问公网实践

概述

在一些场景中，用户需要让CCE集群中的节点和Pod具备访问公网的能力，如通过yum install等命令访问外部源地址下载安装软件、容器需要访问公网中的某些服务等等。 我们提供了以下两种方案，可以实现集群访问公网，用户可以根据具体需求自行选择。

相关概念

EIP：EIP是百度智能云提供的弹性公网IP服务，EIP可以挂载到CCE节点、BLB、NAT网关等服务，提供可以访问公网的地址。

NAT网关：百度智能云 NAT 网关支持用户VPC子网中的BCC、DCC等实例连接公网，通过NAT网关实现内网IP转换为公网IP地址。

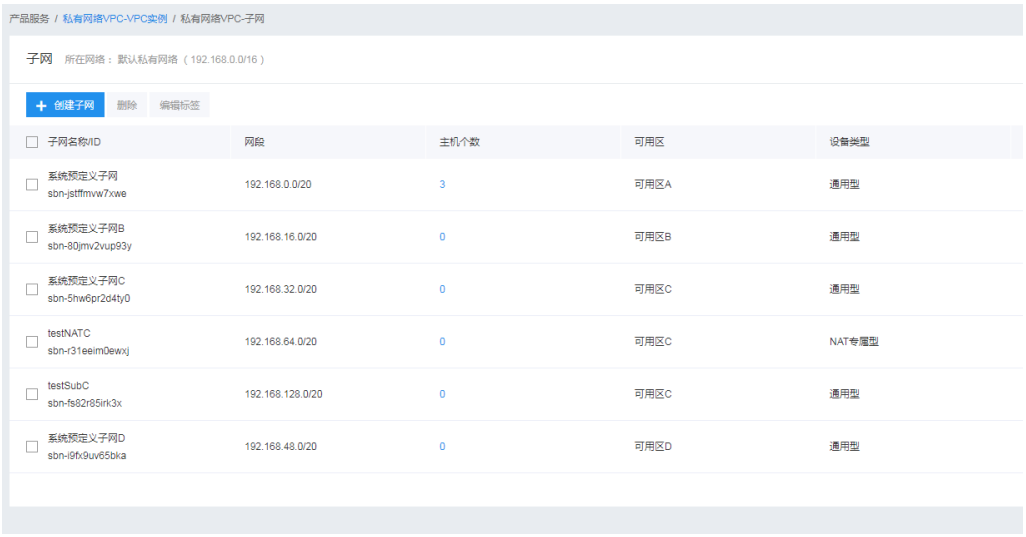
节点子网：用户在创建CCE工作节点时，可以选择节点所在的VPC子网。子网类型为通用型子网，通用型子网中的节点可以挂载EIP直接访问公网，也可以通过NAT网关实现公网访问。

（注：百度云私有网络VPC不再支持创建NAT专属型子网，若使用方案二进行访问公网实践，请选择通用型子网作为节点子网。之前创建的NAT专属型子网将不受影响，可正常使用。）

方案一：通用型子网 + EIP

在创建和添加CCE节点时，为节点选择通用型的子网：

如果无法确认子网列表中哪些子网是通用型，可以在VPC控制台查看子网的设备类型：



如果VPC中没有合适的通用型子网，也可以创建新的通用型子网，将设备类型选为通用型即可：

如果节点在通用型子网中，那么可以直接为节点绑定EIP，可以在创建和添加节点时联动购买EIP，也可以在节点加入集群后再进行EIP的绑定：

节点在绑定EIP之后，即可以直接访问公网。

方案二：NAT专属型子网 + NAT 网关

（注：百度云私有网络VPC不再支持创建NAT专属型子网，若使用方案二进行访问公网实践，请选择通用型子网作为节点子网。之前创建的NAT专属型子网将不受影响，可正常使用。）

如果不希望节点在访问公网时暴露自己的公网IP，可以通过NAT网关为私有网络提供访公网访问服务。

首先在VPC中创建子网时，设备类型选择NAT专属型。

进入VPC控制台，选择VPC实例--> NAT网关--> 创建NAT网关，如下图：

完成NAT网关后，在VPC中配置路由表，路由的源网段为节点所在的子网，目标网段为全0，路由类型为NAT网关，下一跳实例为创建好的NAT网关。如下图：

配置完成后，创建集群时选择该NAT专属型子网，创建出的集群节点都可以访问公网。

注意：NAT专属型子网内的节点将无法绑定EIP，必须创建NAT网关并且建立路由表规则后，才能访问公网。

优缺点分析

方案	优点	缺点
通用型子网 + EIP	操作简单 无需为NAT网关付费	需要暴露节点EIP，存在安全风险
NAT专属子网 + NAT网关	节点完全没有EIP，安全性高	节点无法绑定EIP，无法被公网访问

CCE集群中使用私有镜像实践

私有镜像指位于私有仓库中的Docker镜像，用户需要使用仓库用户名密码登入(docker login)后才能拉取。在CCE集群中，需要为Pod配置imagePullSecrets，才能够在集群中使用私有镜像启动Pod中的容器。

1. 创建用于私有镜像拉取的secret (imagePullSecrets)

使用kubectl命令行创建，需要先获取

- 镜像仓库登录用户名
- 镜像仓库登录密码
- 邮箱地址
- 镜像仓库服务器地址

创建命令为

```
kubectl create secret docker-registry <secret名称> \
--docker-server=<镜像仓库服务器地址> \
--docker-username=<镜像仓库用户名> \
--docker-password=<镜像仓库密码> \
--docker-email=<邮箱地址>
```

关于镜像仓库服务器地址(--docker-server参数)：

- 对于存储在百度智能云容器镜像服务CCR的私有镜像（镜像地址为registry.baidubce.com/<namespace>/<name>:<tag>），镜像仓库服务器地址参数须填写为 --docker-server='https://registry.baidubce.com'。
- 对于存储在百度智能云镜像仓库的私有镜像（镜像地址为hub.baidubce.com/<namespace>/<name>:<tag>），镜像仓库服务器地址参数须填写为 --docker-server='https://hub.baidubce.com'。
- 其他第三方镜像仓库请按照实际服务器地址进行配置，可参考 社区说明文档 或咨询镜像服务提供商。

> **注意：**
>
> - secret需与引用该secret的pod在同一个k8s命名空间内。

后续推荐使用 CCR，创建 CCR 镜像仓库 Secret 命令如下：

```
kubectl create secret docker-registry private-repo-auth \
--docker-server=registry.baidubce.com \
--docker-username=<镜像仓库用户名> \
--docker-password=<镜像仓库密码> \
--docker-email=<邮箱地址>
```

创建完毕后，可以通过kubectl get secret查看到一个类型为 kubernetes.io/dockerconfigjson 的secret被创建。

```
$ kubectl get secret
NAME                  TYPE                                  DATA      AGE
default-token-bx894   kubernetes.io/service-account-token   3         4d
private-repo-auth     kubernetes.io/dockerconfigjson        1         2m
```

2. 在pod spec中指定拉取镜像的secret

在 pod.spec.imagePullSecrets 中指定secret的名称，即可在pod中使用相应的私有镜像。

以创建一个Deployment为例，假设已经在default命名空间中通过上述方式创建了名为private-repo-auth的secret，对应的Deployment yaml文件如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-nginx
  template:
    metadata:
      name: my-nginx
      labels:
        app: my-nginx
    spec:
      imagePullSecrets:
      - name: private-repo-auth
      containers:
      - name: my-nginx
        image: nginx
```

> **注意：**

- 可以在同一个pod中指定多个拉取镜像的secret，用于拉取Pod中声明的来自多个来源的私有镜像。

CCE典型实践之容器网络模式选择

通过 CCE Ingress 实现虚机和容器服务的统一接入

使用场景

很多情况下，我们的集群中的虚拟机会同时存在非容器化部署的服务。如果希望使用一个应用型 BLB 可以同时向容器化部署的服务和虚拟机上非容器化部署的服务转发请求，可以按照本文档进行配置。

方案概述

概念

- **应用型 BLB**：适用于用户复杂应用部署架构的 BLB，允许用户配置不同的转发规则（4层或7层）。
- **监听器转发规则**：对进入 BLB 数据包按照设定的规则进行匹配，并转发到不同的目标组上。
- **目标组 - 服务器组**：数据包转发的目标，内容是一组百度智能云服务器。
- **目标组 - IP 组**：数据包转发的目标，内容是一组 IP 地址，IP 地址可以是 VPC 内的 IP 地址（如节点地址、Ingress 地址等）。

原理

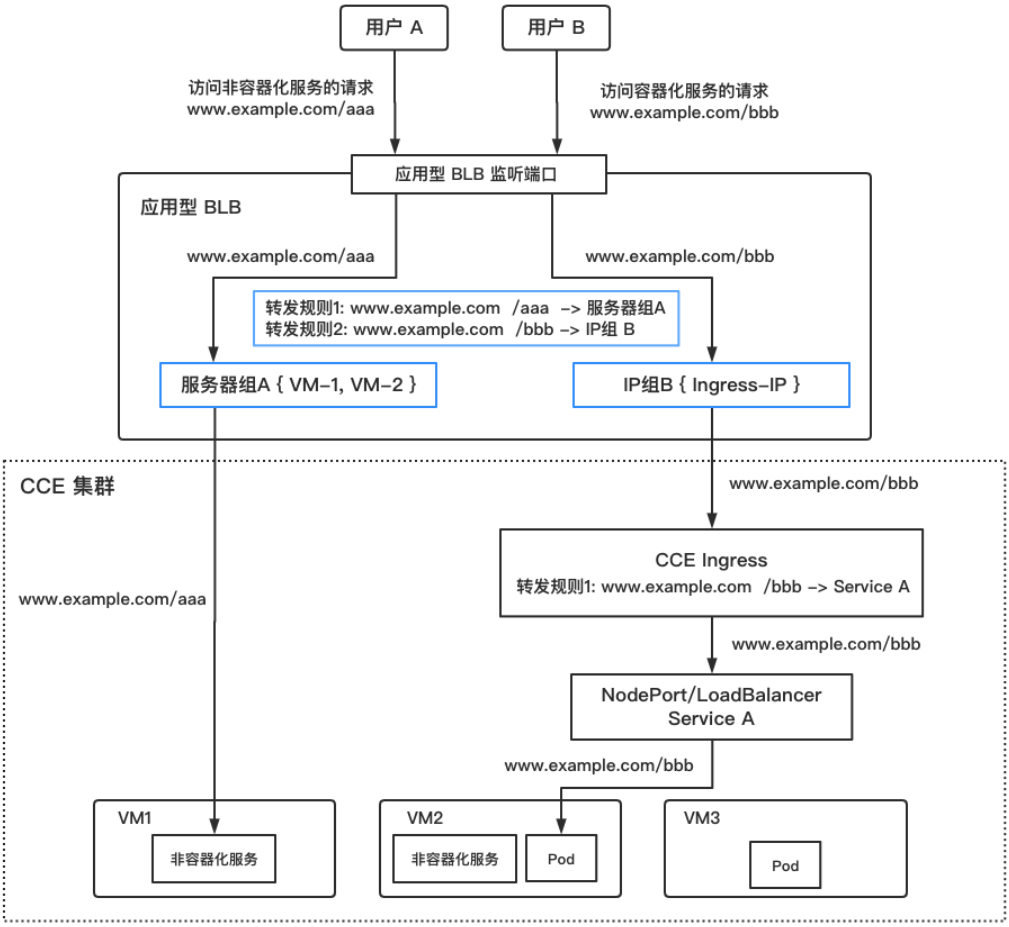
在 CCE 集群中，容器化部署的一个或多个服务（以下简称容器化服务）一般通过 Ingress 接入外部流量；而非容器化部署的服务（以下简称虚机服务）一般通过自行配置应用型 BLB 来接入外部流量。

我们可以为应用型 BLB 配置 HTTP 转发规则，并将 Ingress 的 VPC 内 IP 地址添加到用户自行配置的应用型 BLB 的 IP 组中，将访问容器化服务的流量导入 Ingress。

网络拓扑示例

配置完成后的网络拓扑如图所示。 访问非容器化服务的请求被应用型 BLB 直接转发到各台后端虚拟机上；访问容器化服务的请求先被转发到 Ingress，再被转发到各个 Service 上。

本指南所涉及的内容如图中蓝色方框所示，包括服务器组配置、IP组配置与监听器转发规则的设置。



操作步骤

1. 准备工作

在开始操作前首先确认下述工作已经完成：

1. 虚机服务已经部署并正常运行
2. 容器化服务已经部署并正常运行，相关 Ingress 已经创建完成
3. 对外暴露的应用型 BLB 及其 HTTP/HTTPS 监听器已经创建完成

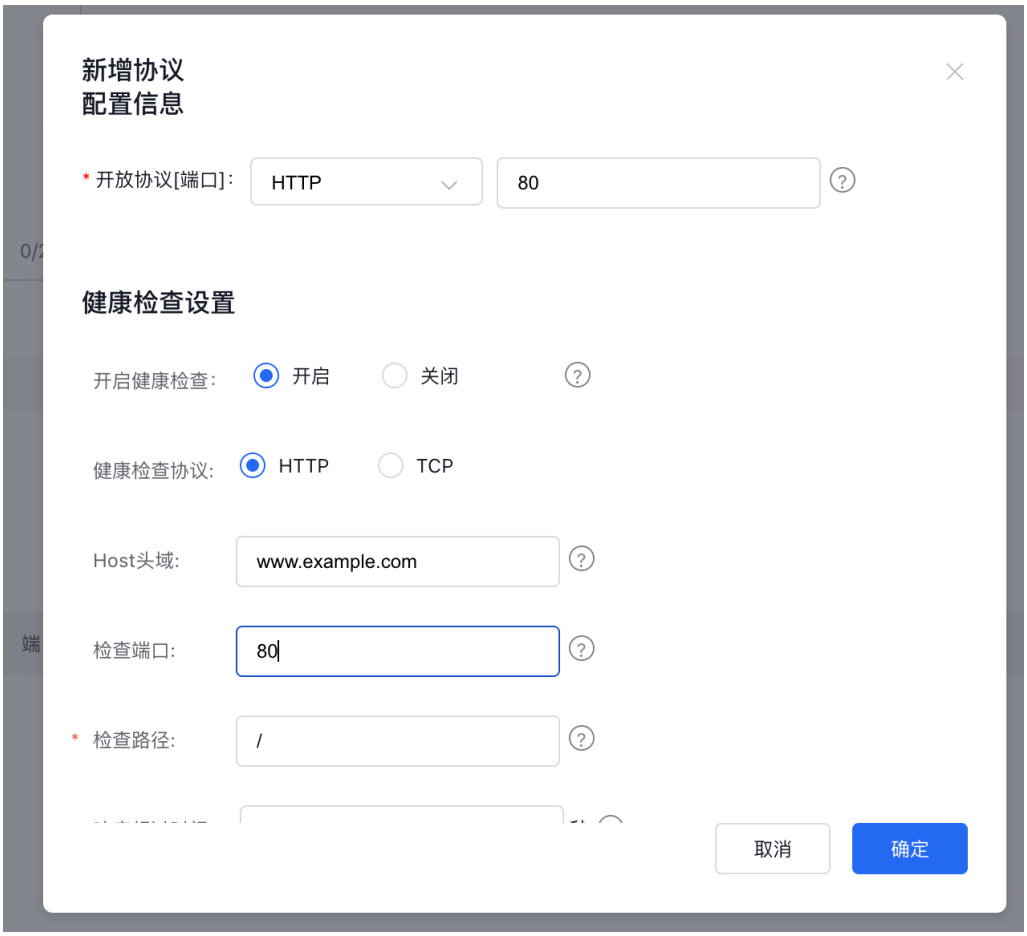
2. 设置服务器组（对于虚机服务）

此步目的是将虚机服务所在的虚机加入应用型 BLB 的后端服务器组中。

1. 进入应用型 BLB 控制台，点击『目标组-添加服务器组』

2. 按需设置『服务器组名称』与『描述』

3. 设置『开放端口』。这里端口指的是虚机服务所示使用的端口，本例为 HTTP:80。并按实际情况配置健康检查规则。



4. 添加后端服务器。这里需要添加的是正在运行虚机服务的服务器。

5. 点击『确认』完成创建。创建后注意查看服务器组和服务器组中后端服务器的健康检查状态。如果状态异常，需要检查配置是否正确，后端服务是否正常运行。

3. 设置IP组（对于容器化服务）

此步目的是将容器化服务相关的 Ingress 加入应用型 BLB 的后端 IP 组中。

1. 确认 Ingress 的内网 IP 地址。可以在『CCE控制台-选择集群-网络-路由』查询到。本例为 172.16.2.2

2. 进入应用型 BLB 控制台，点击『目标组-添加IP组』

3. 按需设置『IP组名称』与『描述』

4. 设置『开放协议』，本指南IP组内容是Ingress，而 Ingress 使用 HTTP 协议，所以此处选择 HTTP 即可。并按实际情况配置健康检查规则。

5. 加入IP。这里将第1步记录的 Ingress 内网地址与端口填入即可。

6. 点击确认完成创建。

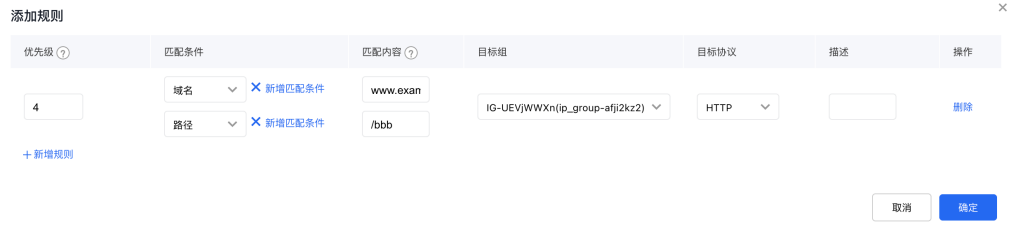
4. 设置监听器转发规则

此步是设置转发规则，以将不同的请求正确地分流到容器化服务或虚机服务中。

1. 进入 BLB 控制台，进入应用型 BLB 实例地页面，选择『添加监听设置』。

2. 设置 BLB 向虚机服务转发请求的规则。点击『添加规则』，设置匹配条件以及对应的『目标组-服务器组』，即之前设定的服务器组。本例为 www.example.com /aaa。

3. 设置 BLB 向容器化服务转发请求的规则。点击『添加规则』，设置匹配条件以及对应的『目标组-IP组』，即之前设定的服务器组。本例为 www.example.com /bbb。



4. 所有配置完成。请测试功能是否符合预期。

云原生AI

部署 TensorFlow Serving 推理服务

本文介绍如何部署 TensorFlow Serving 推理服务，并指定队列、GPU资源。

前提条件

- 您已成功安装 CCE GPU Manager 和 CCE AI Job Scheduler 组件，否则云原生 AI 功能将无法使用。

操作步骤示例

这里用 TensorFlow Serving 作为示例，演示如何通过 deployment 部署推理服务。

1. 部署 TensorFlow Serving 推理服务
    - 指定使用 default 队列：scheduling.volcano.sh/queue-name: default
    - 申请 1张GPU卡的50%的算力，10Gi显存
    - 调度器指定为 volcano （必须）

参考 yaml 如下:

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: gpu-demo
 namespace: default
spec:
 replicas: 1
 selector:
   matchLabels:
     app: gpu-demo
 template:
   metadata:
     annotations:
       scheduling.volcano.sh/queue-name: default
     labels:
       app: gpu-demo
   spec:
     containers:
       - image: registry.baidubce.com/cce-public/tensorflow-serving:demo-gpu
         imagePullPolicy: Always
         name: gpu-demo
         env:
           - name: MODEL_NAME
             value: half_plus_two
         ports:
         - containerPort: 8501
         resources:
           limits:
             cpu: "2"
             memory: 2Gi
             baidu.com/v100_32g_cgpu: "1"
             baidu.com/v100_32g_cgpu_core: "50"
             baidu.com/v100_32g_cgpu_memory: "10"
           requests:
             cpu: "2"
             memory: 2Gi
             baidu.com/v100_32g_cgpu: "1"
             baidu.com/v100_32g_cgpu_core: "50"
             baidu.com/v100_32g_cgpu_memory: "10"
         # if gpu core isolation is enabled, set the following preStop hook for graceful shutdown.
         # `tf_serving_entrypoint.sh` needs to be replaced with the name of your gpu process.
         lifecycle:
             preStop:
               exec:
                 command: ["/bin/sh", "-c", "kill -10 `ps -ef | grep tf_serving_entrypoint.sh | grep -v grep | awk '{print $2}'` && sleep 1"]
     dnsPolicy: ClusterFirst
     restartPolicy: Always
     schedulerName: volcano
```

2. 执行以下命令，查看任务运行状态

```
 kubectl get deployments
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
gpu-demo   1/1     1            1           30s

kubectl get pod -o wide
NAME                          READY   STATUS      RESTARTS   AGE    IP           NODE          NOMINATED NODE   READINESS GATES
gpu-demo-65767d67cc-xhdgg     1/1     Running     0          63s    172.23.1.86   192.168.48.8   <none>           <none>
```

3. 验证 Tensorflow 推理服务是否可用

```
 ##### 需替换 <172.23.1.86> 为实际 pod ip
curl -d '{"instances": [1.0, 2.0, 5.0]}'   -X POST http://172.23.1.86:8501/v1/models/half_plus_two:predict

##### 输出类似如下结果:
{
    "predictions": [2.5, 3.0, 4.5]
}
```

队列使用说明

可通过 annotations 指定队列

```
 annotations:
  scheduling.volcano.sh/queue-name: <队列名称>
```

资源申请说明

单卡独占示例

```
resources:
    requests:
      baidu.com/v100_32g_cgpu: 1 // 1卡
      cpu: "4"
      memory: 6Gi
    limits:
      baidu.com/v100_32g_cgpu: 1 // limit与request必须一致
      cpu: "4"
      memory: 6Gi
```

多卡独占示例：

```
resources:
    requests:
      baidu.com/v100_32g_cgpu: 2 // 2卡
      cpu: "4"
      memory: 6Gi
    limits:
      baidu.com/v100_32g_cgpu: 2 // limit与request必须一致
      cpu: "4"
      memory: 6Gi
```

单卡共享【不进行算力隔离，只有显存隔离】示例：

```
resources:
    requests:
      baidu.com/v100_32g_cgpu: 1 // 1卡
      baidu.com/v100_32g_cgpu_memory: 10 // 10GB
      cpu: "4"
      memory: 6Gi
    limits:
      baidu.com/v100_32g_cgpu: 1 // limit与request必须一致
      baidu.com/v100_32g_cgpu_memory: 10
      cpu: "4"
      memory: 6Gi
```

单卡共享【同时支持显存隔离和算力隔离】示例：

```
resources:
    requests:
      baidu.com/v100_32g_cgpu: 1 // 1卡
      baidu.com/v100_32g_cgpu_core: 50 // 50%, 0.5卡算力
      baidu.com/v100_32g_cgpu_memory: 10 // 10GB
      cpu: "4"
      memory: 6Gi
    limits:
      baidu.com/v100_32g_cgpu: 1 // limit与request必须一致
      baidu.com/v100_32g_cgpu_core: 50
      baidu.com/v100_32g_cgpu_memory: 10
      cpu: "4"
      memory: 6Gi
```

GPU卡类型和资源名称对比关系

目前以下类型的GPU支持显存和算力的共享与隔离：

GPU卡型号	资源名称
Tesla V100-SXM2-16GB	baidu.com/v100_16g_cgpu
Tesla V100-SXM2-32GB	baidu.com/v100_32g_cgpu
Tesla T4	baidu.com/t4_16g_cgpu

GPU虚拟化之隔离性最优型的最佳实践

1. 概览

本指南旨在帮助用户高效地使用百度云的 GPU虚拟化的隔离性最优型 服务。通过本文档，您将了解如何选择和配置 GPU 资源，实现 GPU 独占和共享模式，并确保任务的稳定性和安全性。

2. 业务流程

3. 前提条件

在开始使用 GPU虚拟化的隔离性最优型 服务前，您需要完成以下操作，才可以进行进一步配置。

3.1 集群创建

（可参考集群创建相关使用文档）

1. 登录百度智能云平台：

- 若没有用户名，请先完成注册，操作请参考注册百度账号。
- 若有用户名，登录操作请参考登录。

2. 登录成功后，选择“产品服务>云原生>容器引擎 CCE”，进入“集群管理>集群列表”页面。
3. （可选）“集群列表”页面会显示所有已创建的 CCE 集群名称/ID、状态、版本等信息，并且用户可以通过集群名称进行搜索集群的操作。
4. （可选）选择区域，请根据实际需求进行选择切换。容器引擎 CCE 服务目前支持华北—北京、华北—保定、华南-广州、华东-苏州、中国香港等区域。
5. 点击“创建集群”，进入“选择模板”界面，根据模板介绍结合业务需求选择模板。

- 默认单一区域下集群配额为20个，每个集群内节点配额为200个。
- 如有需求，可以提工单提升配额。

6. 点击“确认”，根据集群创建导航并结合业务需求配置。
7. 付费及地域

8. 基础配置

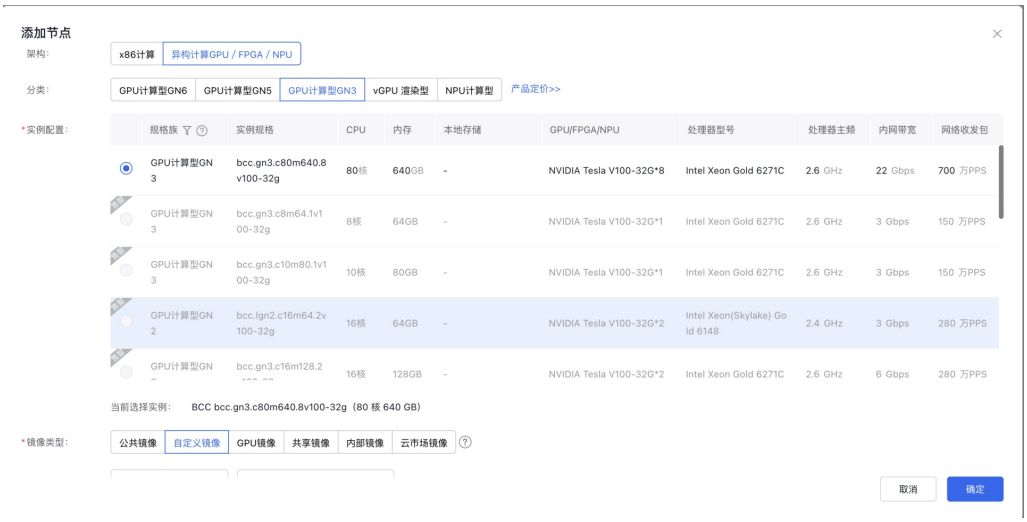
9. 网络配置

容器网络为独立的地址空间，须与节点网络、节点子网、其他集群的容器网络相互独立。

10. 节点配置

- 用户可以选择“新建节点”。

- 选择后点击添加节点，弹出“添加节点”对话框。



注意：在实例配置部分，若该节点想使用GPU虚拟化功能，配置时请参考隔离性最优型对应的GPU虚拟化适配表。为确保任务能够顺利创建，请根据适配表选择合适的实例，否则创建任务时识别不到不适配的GPU。

11. 服务器配置

3.2 安装必要组件：

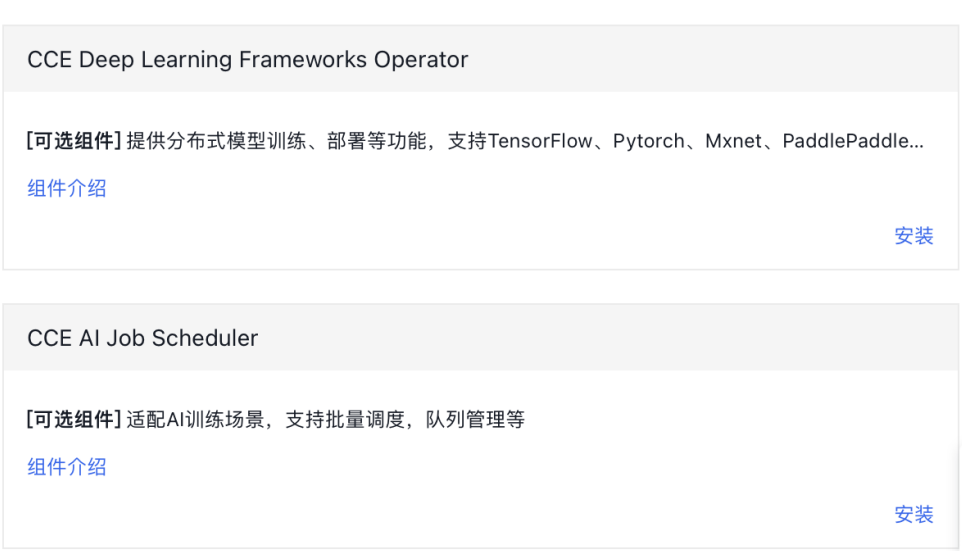
您需要在“集群 > 组件管理 > 云原生AI”中安装 **CCE GPU Manager**组件、**CCE Deep Learning Frameworks Operator**和**CCE AI Job Scheduler** 组件。

3.2.1 CCE GPU Manager组件： 组件参数说明：

参数	可选值	说明
组件类型	隔离性最优型	请注意对应的GPU虚拟化适配表。
GPU显存共享单位	GiB	GPU显存切分的最小单位，目前仅支持GiB
精细化调度	开启/关闭	关闭精细化调度后，资源上报不区分具体的GPU型号。 开启精细化调度后，创建队列和任务时均支持选择具体的GPU型号。

3.2.2 CCE Deep Learning Frameworks Operator和CCE AI Job Scheduler 组件

进入组件管理部分安装即可



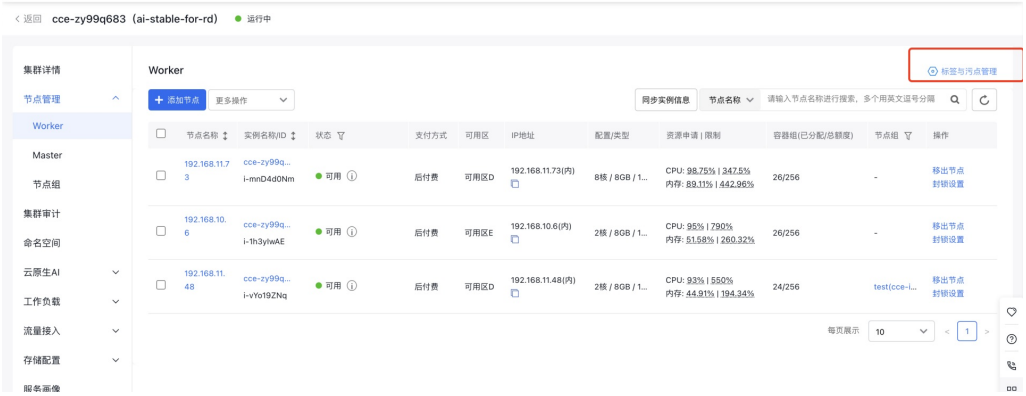
3.3 节点开启显存共享步骤：

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > Worker** ，进入节点列表页面。
5. 在节点列表中，选择需要设置显存共享的 GPU 节点，单击 **开启显存共享**。
6. 打开显存共享开关后单击 **确定**，完成设置显存共享。

3.4 批量打开节点的GPU虚拟化功能 方法一：批量针对**已有节点**添加虚拟化节点标签

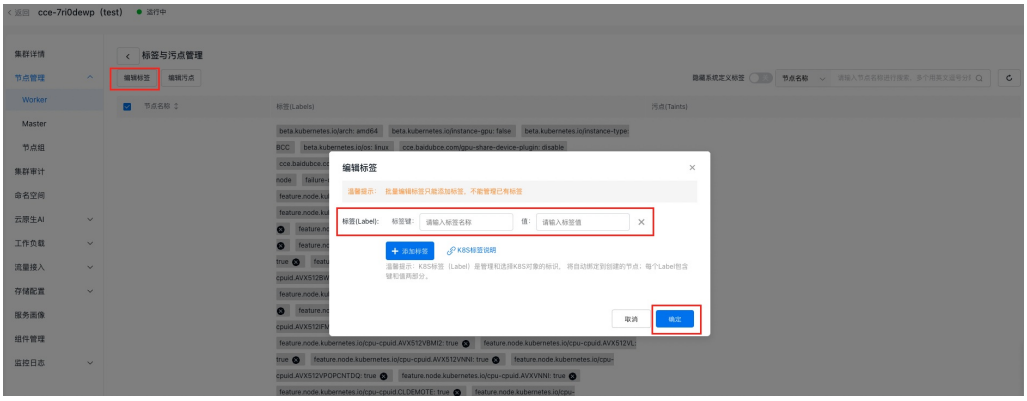
1. 登录百度智能云官网，并进入管理控制台。

2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > Worker** ，进入节点列表页面。
5. 点击**标签与污点管理**进入



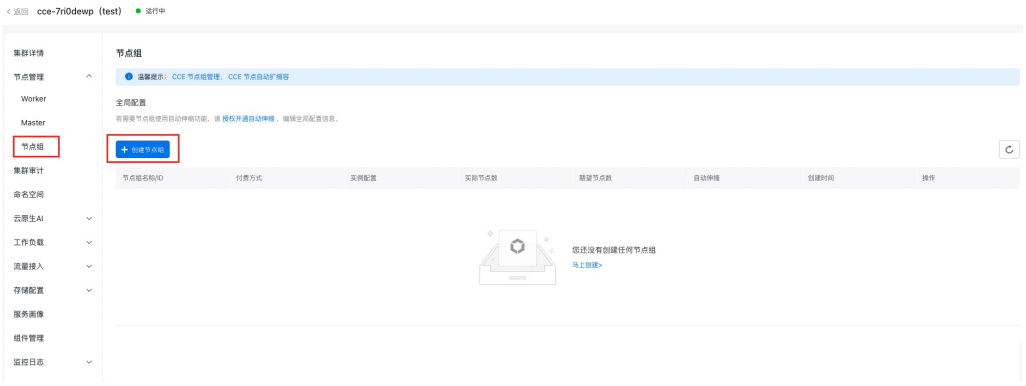
6. 点击**编辑标签**，输入标签键和值，点击**确定**

标签键：cce.baidubce.com/gpu-share-device-plugin
值：enable



方法二：使用节点组默认添加节点标签

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，进入 **集群管理 > 集群列表** 。
3. 在集群列表页面中，单击目标集群名称进入集群管理页面。
4. 选择页面左侧 **节点管理 > 节点组**，进入节点组页面。
5. 点击**创建节点**组进入



6. 填写节点组基本配置：

- 节点组名称：自定义，支持大小写字母、数字、中文以及“-_ /.”；必须以字母开头，长度1-65

- VPC网络：集群的VPC网络
- 节点配置：节点组中节点的配置，包括可用区、节点子网、实例规格等。后续扩容节点组将以此配置作为创建节点的模版。创建节点时填写的节点数量即初始设置的期望节点数
- 自动伸缩：启用自动伸缩，系统将根据节点配置和自动伸缩配置，在符合扩容条件时，自动进行扩容操作。并自动计算节点费用、生成订单。扩容操作完成后，可自行查看节点和订单信息
- 高级设置：支持对扩缩容策略、kubelet数据目录、容器数据目录、部署前执行脚本、部署后执行脚本、自定义kubelet参数、封锁节点（cordon）、资源标签、K8S标签、污点设置（Taints）、注释（Annotations）等参数的设置。 **配置K8S标签**

```
标签键：cce.baidubce.com/gpu-share-device-plugin
值：enable
```

3.5 获得RBAC授权

注意：若您是**子用户**，您需要拥有主账号或具有管理员权限的子用户的RBAC授权，您才能新建任务。若不是，则不需要此步骤。

1. 获得IAM授权。子用户需要先在IAM中至少被授予CCE的只读权限，详细的授权流程请参考配置IAM预置权限策略
2. 获得RBAC授权。详细的授权流程请参考配置预置RBAC权限策略

4. GPU 资源描述

4.1 GPU卡型号和资源名称

正确指定 GPU 卡型号对应的资源名称是使用 GPU虚拟化的隔离性最优型 的关键，在创建任务前，了解GPU资源配置和适配才可以正确的创建任务。若需指定 GPU 卡算力资源或显存资源，请在资源名称后加上“_core”或“_memory”。

以下是常见 GPU 卡型号及其对应的资源名称：

GPU卡型号	资源名称
NVIDIA V100 16GB	baidu.com/v100_16g_cgpu
NVIDIA V100 32GB	baidu.com/v100_32g_cgpu
NVIDIA T4	baidu.com/t4_16g_cgpu
NVIDIA A100 80GB	baidu.com/a100_80g_cgpu
NVIDIA A100 40GB	baidu.com/a100_40g_cgpu
NVIDIA A800 80GB	baidu.com/a800_80g_cgpu
NVIDIA A30	baidu.com/a30_24g_cgpu
NVIDIA A10	baidu.com/a10_24g_cgpu
NVIDIA RTX3090	baidu.com/rtx_3090_cgpu
NVIDIA RTX4090	baidu.com/rtx_4090_cgpu
NVIDIA H800	baidu.com/h800_80g_cgpu
NVIDIA L20	baidu.com/l20_cgpu
NVIDIA H20 96GB	baidu.com/h20_96g_cgpu
NVIDIA H20 141GB	baidu.com/h20_141g_cgpu
NVIDIA H20z	baidu.com/h20z_141g_cgpu

4.2 资源描述

资源名称	类型	单位	说明
baidu.com/xxx_xxx_cgpu	int64	1	GPU 卡数量，共享场景下请填 1。 单容器多卡情况下表示申请的GPU共享卡个数
baidu.com/xxx_xxx_cgpu_core	int64	5%	GPU 卡算力，最小单位 5%
baidu.com/xxx_xxx_cgpu_memory	int64	GiB	GPU 卡显存
baidu.com/xxx_xxx_cgpu_memory_percent	int64	1%	按百分比申请 GPU 卡显存，最小单位 1%

5. 创建任务

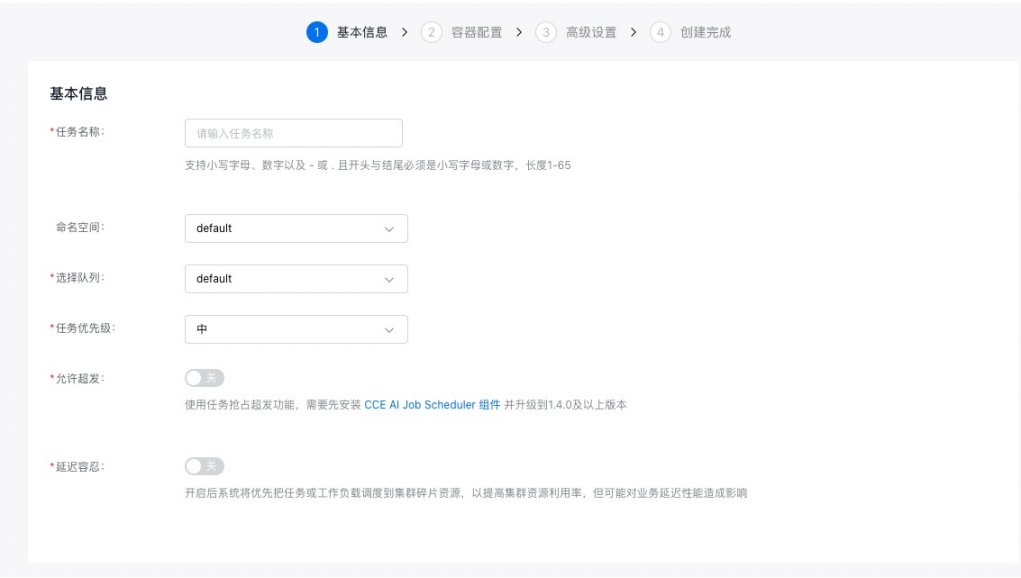
用户可通过**CCE控制台**创建任务/工作负载或者**YAML** 创建任务/工作负载，可根据您的需求选择不同的方式创建。创建的细节如下文所示。

注意：共享场景下的镜像构建注意事项在第七部分有详细说明。

5.1. 在控制台创建任务

操作步骤(可参考云原生AI任务管理)

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**云原生AI > 任务管理**。
6. 在任务管理页面单击**新建任务**。
7. 在新建任务页面中，完成任务基本信息配置：



- 任务名称：自定义任务名称，支持大小写字母、数字、以及-_ /.特殊字符，必须以中文或字母开头，长度 1-65。
- 命名空间：选择新建任务所在的命名空间。
- 选择队列：选择新建任务关联的队列。
- 任务优先级：选择任务对应的任务优先级。
- 允许超发：允许超发将使用任务抢占超发功能，需要先安装CCE AI Job Scheduler组件并升级到1.4.0及以上版本。
- 延迟容忍：开启后系统将优先把该任务或工作负载调度到集群碎片资源。

8. 完成代码基本信息配置：

- 代码配置类型：指定代码配置方式，目前支持“BOS文件”、“本地文件上传”与“暂不配置”。
- 执行命令：指定代码的执行命令。

9.完成数据相关信息配置：

- 设置数据源：当前同时支持数据集和持久卷声明。选择数据集时列出所有可用的数据集，选择后会同时选择与数据集同名的持久卷声明；使用持久卷声明时直接选择即可。

10.点击“下一步”，进入容器相关配置。

11.完成任务类型相关信息配置：

- 选择框架：选择“ TensorFlow ”。
- 训练方式：指定训练方式为“单机”或“分布式”。

- 选择角色：训练方式为“单机”时，只能选择“Woker”；训练方式为“分布式”时，可额外选择“PS”、“Chief”、“Evaluator”。 12.完成容器组相关信息配置，可以根据需要同时进行高级设置。
- 期望Pod数：指定容器组的Pod数目。
- 重启策略：指定容器组的重启策略，可选择的策略有“失败重启”或“从不重启”。
- 镜像地址：指定容器的镜像拉取地址，也可以直接点击“选择镜像”，选择需要使用的镜像。共享GPU场景下镜像构建注意事项在第六部分。
- 镜像版本：指定镜像的版本，若不指定默认拉取latest版。
- **容器配额**：指定容器的CPU、内存、GPU资源相关信息。 您可在任务的容器组配置中指定GPU类型为独占或共享，不同类型如下。

5.1.1 单卡/多卡独占

- 当您指定GPU类型为独占GPU时，若希望任务使用整张GPU卡资源，操作步骤如下：

1. 选择 GPU 卡型号。
2. 输入 GPU 卡数量，范围为 [1~当前集群中单节点所含 GPU 卡最大数量]。

- 若希望任务仅使用CPU和内存资源，无需使用GPU资源，操作步骤如下：

GPU卡型号保持为空，仅输入所需CPU和内存资源量。

5.1.2 单卡共享（不进行算力隔离，仅显存隔离）

当您指定GPU类型为共享GPU时，若希望任务只进行显存隔离，算力无需隔离，操作步骤如下：

1. 选择 GPU 卡型号。
2. 算力开关关闭。
3. 输入所需 GPU 显存，范围为 [1~所选 GPU 卡显存大小]。

5.1.3 单卡共享（显存隔离和算力隔离）

当您指定GPU类型为共享GPU时，若希望任务同时进行显存隔离和算力隔离，操作步骤如下：

1. 选择GPU卡型号。
2. 打开算力开关，输入所需算力值百分比，百分比大小应为[5~100正整数]。
3. 再输入所需GPU显存，显存大小应为正整数，范围为[1~所选GPU卡显存大小]。

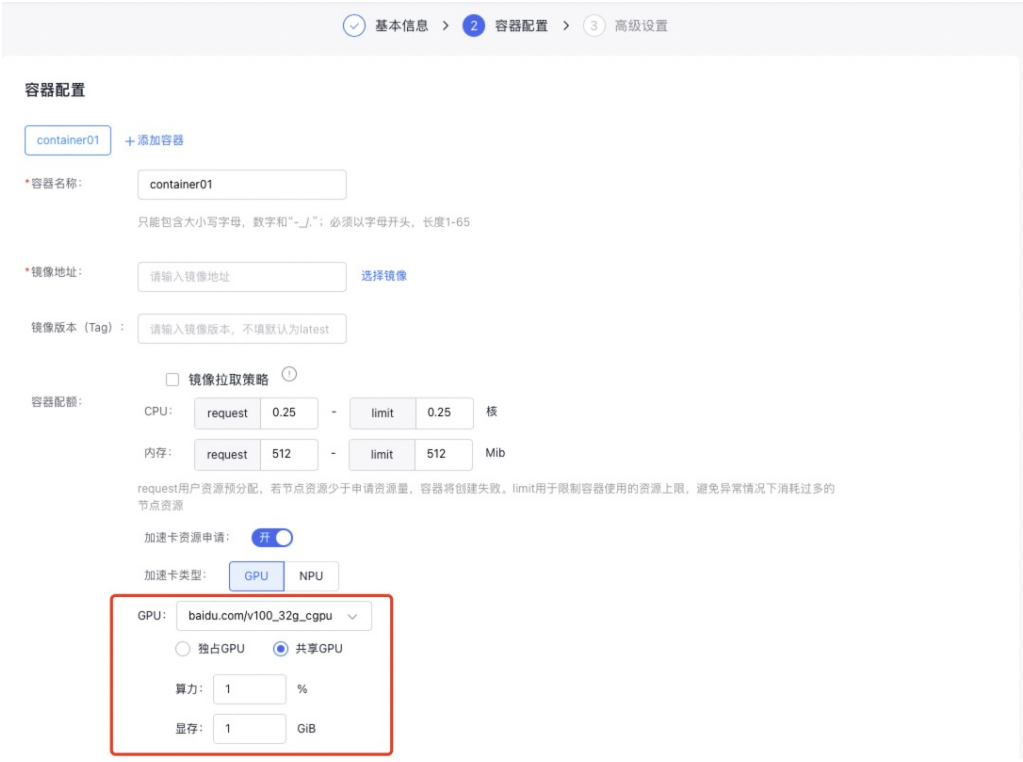
13.完成任务高级信息相关配置。

- 最大训练时长：指定允许的最大训练时长，若不指定，则为不限制时间。
- 私有仓库凭证：若需要使用私有镜像仓库，请在此处添加对应镜像仓库的访问凭证。
- Tensorboard：若需要任务可视化时，可开启Tensorboard功能，开启后需要指定“服务类型”与“ 训练日志读取路径”。
- K8S标签：指定任务对应的K8S Label。
- 注释：指定任务对应的Annotation。

14. 点击“**完成**”按钮，完成任务的新建。

5.2. 在控制台创建工作负载

若您通过CCE控制台创建工作负载（操作步骤可参考工作负载），您可在工作负载的容器配置中指定GPU类型为独占或共享。独占和共享的资源输入限制和上文中AI任务的创建一致。



5.3. 使用 YAML 创建任务/工作负载

若您通过Yaml创建任务或工作负载（详细配置可参考云原生AI任务管理、工作负载），您可在Yaml配置中指定所需GPU卡资源为独占或共享，具体示例如下：

备注：

1. 如使用Yaml的方式创建任务，调度器必须指定 schedulerName: volcano
2. GPU显存资源必须申请，且 baidu.com/v100_32g_cgpu_memory 和 baidu.com/v100_32g_cgpu_memory_percent只能填写一个，不可同时填写
3. 卡的资源名称根据第四部分资源描述填写，不同卡型号对应不同的资源名称。 **5.3.1 单卡独占**

```
resources:
requests:
  baidu.com/v100_32g_cgpu: 1 // 1卡
  cpu: "4"
  memory: 60Gi
limits:
  baidu.com/v100_32g_cgpu: 1 // limit与request必须一致
  cpu: "4"
  memory: 60Gi
```

5.3.2 多卡独占

```
resources:
requests:
  baidu.com/v100_32g_cgpu: 2 // 2卡
  cpu: "4"
  memory: 60Gi
limits:
  baidu.com/v100_32g_cgpu: 2 // limit与request必须一致
  cpu: "4"
  memory: 60Gi
```

5.3.3 单卡共享（不进行算力隔离，仅显存隔离）

```
resources:
 requests:
   baidu.com/v100_32g_cgpu_memory: 10 // 10GB，用户可根据自己的需求填写
   cpu: "4"
   memory: 60Gi
 limits:
   baidu.com/v100_32g_cgpu_memory: 10
   cpu: "4"
   memory: 60Gi
```

5.3.4 单卡共享（显存隔离和算力隔离）

```
resources:
 requests:
   baidu.com/v100_32g_cgpu_core: 50 // 50%, 0.5卡算力，用户可根据自己的需求填写
   baidu.com/v100_32g_cgpu_memory: 10 // 10GB，用户可根据自己的需求填写
   cpu: "4"
   memory: 60Gi
 limits:
   baidu.com/v100_32g_cgpu_core: 50 //
   baidu.com/v100_32g_cgpu_memory: 10
   cpu: "4"
   memory: 60Gi
```

5.3.6 单容器多卡能力（GPU显存/算力同时隔离&仅显存隔离）

通过指定资源描述符使用sGPU的单容器多卡能力，分为GPU显存/算力同时隔离以及仅隔离显存的场景。

1. GPU显存/算力同时隔离

单张共享卡的资源：

- 每张 GPU卡的算力资源：baidu.com/xxx_xxx_cgpu_core/baidu.com/xxx_xxx_cgpu
- 每张 GPU卡的显存资源：baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu

资源描述示例，如下示例代表：该Pod共申请50%的算力，以及10GiB的显存，2张GPU共享卡。所以每张GPU共享卡的资源为 25%的算力以及5GiB显存。

```
 resources:
limits:
baidu.com/a10_24g_cgpu: "2"
baidu.com/a10_24g_cgpu_core: "50"
baidu.com/a10_24g_cgpu_memory: "10"
```

2. GPU显存隔离，算力共享

单张共享卡的资源：

- 每张 GPU卡的算力资源：与其他容器共享100%的算力。
- 每张 GPU卡的显存资源：baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu

资源描述示例，如下示例代表：该Pod共申请10GiB的显存，2张GPU共享卡。所以每张GPU共享卡的资源为 共享100%的算力以及5GiB显存。

```
resources:
      limits:
        baidu.com/a10_24g_cgpu: "2"
        baidu.com/a10_24g_cgpu_memory: "10"
```

3. 使用限制

- 单卡的显存/算力需要为正整数， 即算力（baidu.com/xxx_xxx_cgpu_core/baidu.com/xxx_xxx_cgpu），显存（baidu.com/xxx_xxx_cgpu_memory/baidu.com/xxx_xxx_cgpu ）需为正整数。
- 单卡的显存/算力需要大于等于显存/算力的最小单位。
- 如果没有申请_cgpu_memory或者_cgpu_memory_percent，是不允许申请_cgpu_core

- 显存隔离最小单位：1GiB。

6. 共享GPU场景下的镜像构建注意事项

1. 在共享GPU场景下，您在创建任务选择镜像时，需要注意该镜像的环境变量。

以下环境变量会由 GPU Manager的组件进行注入，请不要添加到镜像的环境变量里面：

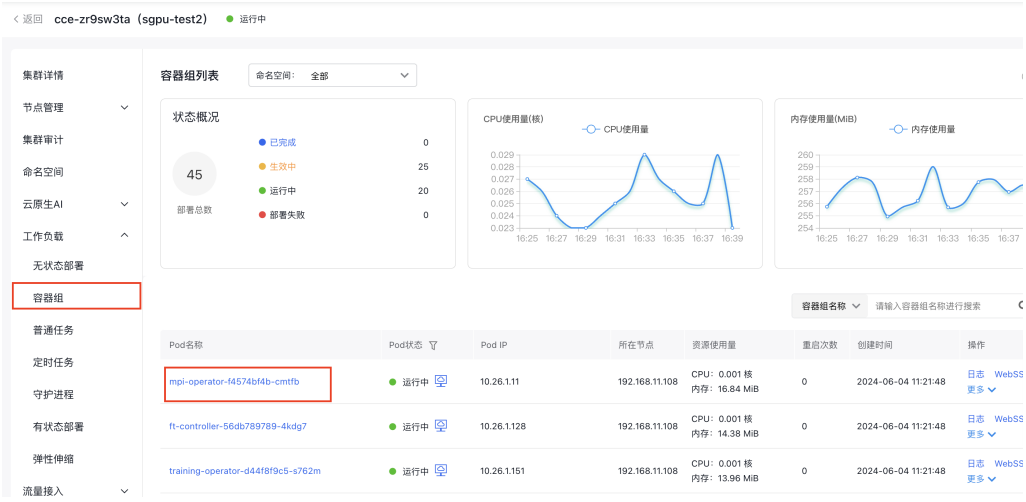
环境变量	说明
NVIDIA_VISIBLE_DEVICES	可见 GPU 设备列表，由调度器分配
NVIDIA_VISIBLE_GPUS_SLOT	可见 GPU 设备插槽，由调度器分配
NVIDIA_VISIBLE_GPUS_UUID	UUID 形式的可见 GPU 设备列表，由调度器分配
LD_LIBRARY_PATH	不推荐设置 LD_LIBRARY_PATH 环境变量，如果非要设置，需要把 /usr/lib64 目录也按 `LD_LIBRARY_PATH=/usr/lib64:$LD_LIBRARY_PATH` 形式添加进去
CUDA_MPS_ACTIVE_THREAD_PERCENTAGE	MPS 算力隔离设置，由调度器分配
CUDA_MPS_LOG_DIRECTORY	MPS 相关日志地址
CUDA_MPS_PIPE_DIRECTORY	和 MPS SERVER 通信的地址
CGPUX_XXX	以 CGPU 开头的环境变量，用于运行显存&算力隔离功能，比如 CGPU0_PRIORITY，CGPU0_SHAREMODE
CGPU_COUNT	设备数量
SGPU_DISABLE	使用隔离最优型时，表示是否使用虚拟化的方式使用 GPU

2. 构建镜像时，请不要直接将集群中运行的 GPU 容器保存为镜像使用，此种方式保存的镜像会包含GPU Manager组件注入的环境变量，造成虚拟化功能不符合预期或者不可用。

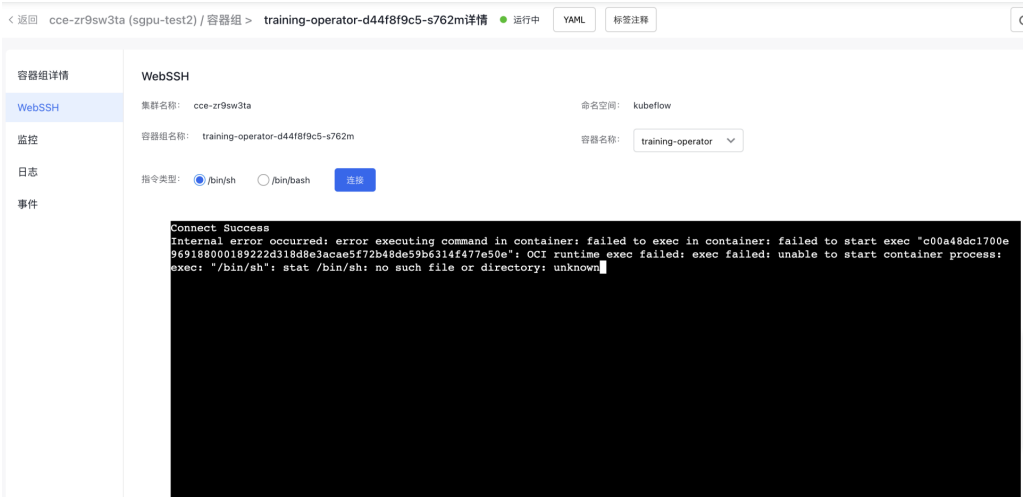
7. (可选）验证GPU虚拟化的隔离效果

您在完成以上操作后，若您想验证被共享的GPU是否按照所设置的隔离参数运行，可通过以下步骤查看。

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 云原生 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的**集群管理 > 集群列表**。
4. 在集群列表页面中，单击目标集群名称进入集群管理页面。
5. 在集群管理页面单击**工作负载>容器组**。
6. 单击您要查看的**Pod名称**



7. 点击**WebSSH**,指令类型选择**/bin/sh**,点击**连接**



8. 输入命令行：nvidia-smi，按下回车键

参数	说明
GPU Name	列出了服务器上的所有GPU。
Memory-Usage	显示了每个GPU的内存使用情况。
GPU-Util	显示了每个GPU的利用率。
Processes	显示了正在使用GPU的进程。

8. 关闭显存共享

1.关闭显存共享功能的限制条件

已开启显存共享功能的节点需要关闭显存共享功能时，系统会验证当前节点上是否有正在运行的显存共享任务，需等该节点所有显存共享任务都运行完成后才能够关闭显存共享功能，否则可能导致显存共享任务中断或任务结束时无法正常回收资源，影响节点上的后续任务。

您可以通过2、3提供的命令查看开启了显存共享的节点和查看该节点上显存共享任务的情况。

2.查询开启显存共享的节点

```
kubectl get nodes -l cce.baidubce.com/gpu-share-device-plugin=enable
```

该命令会列出开启了显存共享功能的所有节点，这些节点上都有标签cce.baidubce.com/gpu-share-device-plugin:enable。

3.查询运行显存共享任务的Pods

```
kubectl get pods --all-namespaces -o json | jq -r '.items[] | select(.status.phase=="Running") | select(.spec.containers[].resources.limits // empty | keys[] // empty | test("baidu.com/.*(_core|_memory|_memory_percent)$")) | "\(.metadata.name) \(.spec.nodeName)"' | sort | uniq
```

该命令会列出所有在运行状态且使用了显存共享功能的 Pods，以及它们所在的节点。这些节点都无法马上关闭显存共享功能，需要等这些Pods结束以后才能关闭节点显存共享功能。

4.使用label命令修改节点标签的风险

除控制台方式外，可以使用kubectl label nodes命令来修改节点标签为cce.baidubce.com/gpu-share-device-plugin:disable，关闭节点的显存共享功能，修改前请关注以下风险：

- 业务中断：修改节点标签之后会触发非共享场景的环境安装，导致正在运行的显存共享任务中断或者结束时无法正常回收资源，影响节点上的后续任务。
- 调度失败：修改标签之后，调度器可以向该节点调度非共享类型的任务，可能存在单机同时运行两种类型的任务，会有错卡，共享显存功能异常等问题。

为了避免这些风险，请始终确保集群中的节点标签正确地反映节点的能力，并且只有在确定节点上没有正在运行的显存共享任务时，再关闭节点

显存共享功能。

- 算力隔离最小值：5%。

Linux系统配置常用参数说明

本文档将列出 Linux 操作系统常用的参数配置以及相应说明。

文件句柄限制

在大流量的业务场景下，可以调整以下内核设置和内置限制，以占用部分系统内存的代价来处理更大的量。

设置	说明
fs.file-max	Linux 内核将分配的最大文件句柄数。通过增加该值，可以增加允许打开的最大文件数。
fs.inotify.max_user_watches	系统允许的最大文件监视数。 每个监视在 32 位内核上约为 90 字节，在 64 位内核上约为 160 字节。
fs.aio-max-nr	显示当前系统范围内的异步 IO 请求数。
fs.nr_open	进程可分配的最大文件句柄数。

套接字和网络优化

设置	说明
net.core.somaxconn	可为任何给定侦听套接字排队的最大连接请求数传递给 listen(2) 函数的积压工作（backlog）参数值的上限。如果积压工作参数大于 somaxconn ，则它会以静默方式截断到此限制。
net.core.netdev_max_backlog	接口接收数据包的速度快于内核的处理速度时，在 INPUT 端排队的数据包的最大数量。
net.core.rmem_max	接收套接字缓冲区最大大小（以字节为单位）。
net.core.wmem_max	发送套接字缓冲区最大大小（以字节为单位）。
net.core.optmem_max	每个套接字允许的辅助缓冲区（选项内存缓冲区）最大大小。 在少数情况下，套接字选项内存用于存储与套接字使用相关的附加结构。
net.ipv4.tcp_max_syn_backlog	尚未收到连接客户端确认的最大排队连接请求数。如果超出此数量，内核将开始删除请求。
net.ipv4.tcp_max_tw_buckets	系统同时容纳timewait套接字的最大数量。 如果超过此数量，则会立即销毁time-wait套接字并显示警告消息。
net.ipv4.tcp_fin_timeout	孤立连接（任何应用程序均不再引用）在本地终止之前保持 FIN_WAIT_2 状态的时间长度。
net.ipv4.tcp_keepalive_time	TCP keepalive 在启用时发送keepalive消息的频率。
net.ipv4.tcp_keepalive_probes	TCP 在确定连接已中断前，发出 keepalive 探测的数量。
net.ipv4.tcp_keepalive_intvl	探测发出的频率。再乘以 tcp_keepalive_probes ， 等于在启动探测后终止未响应连接所需的时间。
net.ipv4.tcp_tw_reuse	从协议角度看安全时，允许将TIME-WAIT套接字用于新连接。
net.ipv4.ip_local_port_range	TCP 和 UDP 流量用来选择本地端口的本地端口范围。 由两个数字组成：第一个数字是代理节点上的 TCP 和 UDP 流量允许的第一个本地端口，第二个是最后一个本地端口号。
net.ipv4.neigh.default.gc_thresh1	ARP 缓存中可能的最小项数。 如果条目数低于此设置，则不会触发垃圾回收。
net.ipv4.neigh.default.gc_thresh2	ARP 缓存中可能的最大项数（软）。 此设置可能是最重要的，因为达到这一软最大值大约 5 秒后会触发 ARP 垃圾回收。
net.ipv4.neigh.default.gc_thresh3	ARP 缓存中的最大项数（硬）。
net.netfilter.nf_conntrack_max	nf_conntrack 是在 Linux 中跟踪 NAT 连接项的模块。 nf_conntrack模块使用哈希表记录 TCP 协议的已 *建立连接* 记录。 nf_conntrack_max 是哈希表中的最大节点数，即，nf_conntrack模块支持的最大连接数或连接跟踪表的大小。
net.netfilter.nf_conntrack_buckets	nf_conntrack 是在 Linux 中跟踪 NAT 连接项的模块。 nf_conntrack模块使用哈希表记录 TCP 协议的已建立连接的记录。 nf_conntrack_buckets 是哈希表的大小。

工作线程限制

与文件描述符限制一样，进程可以创建线程的数量受内核设置所限制。

设置	说明
kernel.threads-max	进程可以启动工作线程。 可创建的所有线程的最大数量均使用 kernel.threads-max 进行设置。

虚拟内存

以下设置可用于调整 Linux 内核虚拟内存子系统的操作以及磁盘脏数据的writeout。

设置	说明
vm.max_map_count	此文件包含进程可能具有的最大内存映射区域数。
vm.vfs_cache_pressure	此百分比值控制内核回收内存（用于缓存目录和 Inode 对象）的趋势。
vm.swappiness	此控制用于定义内核交换内存页的积极程度。 增加值会提高积极程度，降低值会减少交换量。 值 0 指示内核不发起交换，直到可用且文件支持的页数小于区域中的高水位。

使用 CNI 配置容器内网络参数

背景

部分容器化应用需要对容器网络命名空间的 sysctl 参数进行优化，尽管可以使用带有特权的 initContainer 实现初始化配置，但更推荐的方式是利用 CNI 原生支持。

CCE CNI 提供了 sysctl 插件，基于 CNI Chain 能够进行灵活的配置。

本文描述了如何使用和配置 CCE CNI，从而实现集群粒度和应用粒度的容器 sysctl 配置。

使用说明

1 确认容器网络模式

执行 `kubectl get cm -n kube-system cce-cni-node-agent -o yaml`，查看 `cniMode` 字段。

- `vpc-route-` 开头表示基于 VPC 实例路由的网络方案
- `vpc-secondary-ip-`, 开头表示基于弹性网卡的网络方案

2 修改 CNI 配置文件模板

根据步骤 1 中获取的网络模式，执行 `kubectl edit cm -n kube-system cce-cni-config-template`, 修改对应模式的 CNI 配置文模板。

手动编辑 `plugins` 列表, 在最后加上 sysctl 的配置文件，等待 1 分钟后所有节点的 CNI 配置将会更新。

修改完的样例如下:

```
cce-cni-secondary-ip-veth: |
{
    "name":"{{ .NetworkName }}",
    "cniVersion":"0.3.1",
    "plugins":[
        {
            "type":"ptp",
            "enableARPProxy":true,
            "vethPrefix":"veth",
            "mtu": {{ .VethMTU }},
            "ipam":{
                "type":"eni-ipam",
                "endpoint":"{{ .IPAMEndPoint }}",
                "instanceType":"{{ .InstanceType }}",
                "deleteENIScopeLinkRoute":true
            }
        },
        {
          "type":"sysctl",
          "kubeconfig":"/etc/cni/net.d/cce-cni.d/cce-cni.kubeconfig"
        }
    ]
}
```

3 集群粒度的配置（可选）

在步骤 2 中的配置里，可以继续添加 `sysctl` 字段，指明需要配置的参数。

如下的配置生效后，集群内所有新建的容器都会将 `/proc/sys/net/core/somaxconn` 设置为 500。

```
{
 "type":"sysctl",
 "kubeconfig":"/etc/cni/net.d/cce-cni.d/cce-cni.kubeconfig",
 "sysctl":{
   "net.core.somaxconn":"8192"
 }
}
```

4 应用粒度的配置（可选）

在完成步骤 2 后，用户可以通过指定 Pod 的 annotations，传递 sysctl 参数的配置。

采用如下的 yaml 的创建的 nginx 容器，将会设置 `/proc/sys/net/core/somaxconn` 为 8192 以及 `/proc/sys/net/ipv4/tcp_tw_reuse` 为 1。

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-example
  template:
    metadata:
      labels:
        app: nginx-example
      annotations:
        net.sysctl.cce.io/net.core.somaxconn: "8192"
        net.sysctl.cce.io/net.ipv4.tcp_tw_reuse: "1"
    spec:
      containers:
        - name: nginx-example
          image: nginx:alpine
          imagePullPolicy: IfNotPresent
```

注意事项

- 确保 CNI 配置是一个合法的 JSON 格式文件
- 不同版本内核中 netns 可配置的 sysctl 参数不同，请小心配置否则容器可能无法启动

参考文档

- Documentation for /proc/sys/net/*
- /proc/sys/net/ipv4/* Variables
- Kernel doc for networking

CCE容器运行时选择

容器运行时（Container Runtime）是 Kubernetes 重要的组件之一，负责管理镜像和容器的生命周期。本文通过对比两种运行时的使用场景、调用链，并通过对比 docker 和 containerd 两种容器引擎常用命令，帮助您根据需求场景选择合适的容器运行时。

containerd 作为一种常见的容器运行时，相比较于 docker，它的调用链更短，组件更少，更稳定，占用节点的资源更少。CCE 同时支持 containerd 和 docker 两种容器运行时，通常情况下建议使用 containerd，若存在以下情况时，请选择 docker 容器运行时：

- 如需使用 docker in docker。
- 如需在 CCE 节点上执行 docker build/push 等命令。
- 如需调用 docker API。
- 如需 docker compose 或 docker swarm。

两种运行时调用链对比

运行时	调用链
docker	kubelet > dockershim > dockerd > containerd
containerd	kubelet > CRI plugin > containerd

Docker 和 Containerd 两种容器引擎常用命令对比

镜像相关功能

功能	Docker	Containerd
查看镜像列表	docker images	crictl images
查看镜像详情	docker inspect	crictl inspecti
推送镜像	docker push	无
拉取镜像	docker pull	crictl pull
删除镜像	docker rmi	crictl rmi

容器相关功能

功能	Docker	Containerd
查看容器列表	docker ps	crictl ps
查看容器详情	docker inspect	crictl inspect
查看容器日志	docker logs	crictl logs
容器内执行命令	docker exec	crictl exec
挂载容器	docker attach	crictl attach
显示容器资源使用情况	docker stats	crictl stats
创建容器	docker create	crictl create
启动容器	docker start	crictl start
停止容器	docker stop	crictl stop
删除容器	docker rm	crictl rm

Pod 相关功能

功能	Docker	Containerd
查看 Pod 列表	无	crictl pods
查看 Pod 详情	无	crictl inspectp
启动 Pod	无	crictl runp
停止 Pod	无	crictl stopp

添加CGroup V2节点

Kubernetes采用 cgroup 实现容器的资源隔离。cgroup v2 统一了访问资源的路径，支持Pod级别的资源监控、网络重定向，同时在跨多资源协调上具备更强的资源隔离能力。本文将介绍如何在CCE控制台添加 cgroup v2 节点。

背景信息

Kubernetes采用 cgroup 实现容器的资源隔离。cgroup 分为 v1 和 v2 版本，v2 版本提供了更一致的体验和更丰富的功能。主要特性如下：

- 统一了访问资源的路径，各种资源处于统一路径下。
- 新增PSI等功能特性。
- 支持 cgroup 级别的eBPF挂载，可以实现Pod级别的资源监控、网络重定向等。
- 在跨多资源协调上具备更强的资源隔离能力。
  - 统一地管理各种类型的内存分配，例如网络内存、Kernel内存等。
  - 支持异步资源变化的统计，例如通过Page Cache的Write-back统计实现异步IO的限制。

Kubernetes在V1.18中alpha支持 cgroup v2，在V1.22中beta支持 cgroup v2，在V1.25中GA支持cgroup v2。具体信息，请参见About cgroup v2。

使用限制

- 目前仅支持ubuntu 22.04和BaiduLinux 3.0，使用方式如下：
  - ubuntu 22.04默认为CGroup V2，可直接在控制台选择使用；
  - BaiduLinux 3.0默认为CGroup V1，可以使用BaiduLinux 3.0作为基础镜像来构建支持CGroup V2的自定义镜像。
- Kubernetes版本大于或等于1.24。

操作步骤

手动添加cgroup v2节点

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**节点管理>Worker**。
2. 在节点列表中单击**添加节点**操作，在**节点配置**选择**公共镜像 ubuntu 22.04**或选择目标自定义镜像。
3. 完成添加节点其他配置。

通过节点组扩容cgroup v2节点

1. 登录百度智能云管理控制台，进入**产品服务>云原生>容器引擎 CCE**，单击**集群管理>集群列表**，单击目标集群名称，进入"集群详情"页，在侧边栏单击**节点管理>节点组**。
2. 在节点组列表中单击**创建节点组**操作，在**节点配置**选择**公共镜像 ubuntu 22.04**或选择目标自定义镜像。
3. 完成创建节点组其他配置。

验证cgroup v2

登录集群节点，执行如下命令查看CGroup类型。验证集群中的节点是否采用了CGroup V2。

```
df -T /sys/fs/cgroup
```

预期结果：

```
Filesystem     Type    1K-blocks  Used Available Use% Mounted on
cgroup2        cgroup2         0     0         0    - /sys/fs/cgroup
```

表示集群中的节点采用了cgroup v2。

用户使用自定义 CNI 插件方法

背景

为了更好地满足用户对于高度定制化网络配置的需求，cce-network 不仅允许用户使用默认的 CNI 插件，更提供了一种机制，使用户能够自定义 CNI 插件。这种自定义能力极大地增强了网络配置的灵活性，从而使得用户可以根据自己的具体需求，进行详细的网络设计和配置。

自定义 CNI 插件的方法包括了对官方支持的插件的选择配置，以及通过修改配置文件直接部署用户自定义的插件。

本文描述了如何使用和配置 CCE CNI，从而实现用户自定义 CNI 插件的配置。

V2版本使用说明

前提条件：

- 已安装 cce-network-v2 组件，版本不小于 2.10.0。
- 确保 CNI 插件使用的规范版本 >= 0.4.0。

1 确认容器网络模式

执行 `kubectl get cm -n kube-system cce-network-v2-config -o yaml`，查看 `ipam` 字段。

- `vpc-route` 开头表示基于 VPC 路由的网络方案
- `vpc-eni`, 开头表示基于弹性网卡的网络方案

2 自定义 CCE 官方支持的插件

下面提供 CCE 官方支持的插件列表，用户可以通过安装 cce-network-v2 时通过 `ccedConfig.ext-cni-plugins` 配置文件进行选择。

插件名称	提供者	插件描述	插件版本
portmap	社区	社区的 CNI 插件，支持Pod 上配置端口直通能力。当开启 ebpf 加速时，该插件会自动失效。	v1.0.0 及以上版本
cilium-cni	CCE	Cilium CNI 插件，支持网络策略、service加速等。	v1.12.5-baidu 及以上版本
endpoint-probe	CCE	CCE 提供的 CNI 插件，用于支持 Pod Qos 等能力。	v2.9.0 及以上版本
cptp	CCE	CCE 提供的默认 CNI 插件，支持Pod基础网络通信能力。	v1.0.0 及以上版本
exclusive-device	CCE	CCE 提供的 CNI 插件，支持Pod独占网卡能力。	v2.6.0 及以上版本
sbr-eip	CCE	CCE 提供的 CNI 插件，支持Pod 直通 EIP 功能。	v2.6.0 及以上版本

例如需要开启 portmap 插件，则可以通过如下配置文件进行重新部署。

1. 修改cni插件模板

```
kubectl edit cm cce-network-v2-config -n kube-system -o yaml

apiVersion: v1
data:
  cced: |
    ......
    ext-cni-plugins:
    - portmap
    - endpoint-probe
    ......
```

2. 清理cni配置文件 需要在修改后重启daemonset cce-network-agent

```
kubectl rollout restart daemonset -n kube-system cce-network-agent
```

3. 验证

进入任意机器，执行 cat /etc/cni/net.d/00-cce-cni.conflist 验证是否包含用户自定义的cni配置文件。

```
cat /etc/cni/net.d/00-cce-cni.conflist
{
  "cniVersion": "0.4.0",
  "name": "generic-veth",
  "ipam": {},
  "dns": {},
  "plugins": [
    {
      "ipam": {
        "type": "cipam"
      },
      "mtu": 1500,
      "type": "cptp"
    },
    {
      "capabilities": {
        "portMappings": true
      },
      "externalSetMarkChain": "KUBE-MARK-MASQ",
      "type": "portmap"
    },
    {
      "type": "endpoint-probe"
    }
  ]
```

3 自定义 CNI 插件

CCE 也支持保留用户自定义的 CNI 插件，用户可以通过修改主机配置文件 /etc/cni/net.d/00-cce-cni.conflist 自行进行部署CNI 配置文件。CCE 在更新时，会使用用户自定义的 CNI 插件。

```
{
  "name":"generic-veth",
  "cniVersion":"0.4.0",
  "plugins":[
    {
      "type":"cptp",
      "ipam":{
         "type":"cipam",
      },
      "mtu": 1500
    }
    ,{
      "type": "endpoint-probe"
    },
    {
      "type": "user-cni"
    }
  ]
}
```

CCE 容器网络在启动时会自动读取该配置文件，如果发现了用户的自定义 CNI 插件，则在更新 CNI 配置时，会把用户自定义 CNI 插件自动插入到插件链的尾端。

V1 版本使用说明

1 确认容器网络模式

执行 `kubectl get cm -n kube-system cce-cni-node-agent -o yaml`，查看 `cniMode` 字段。

- `vpc-route-` 开头表示基于 VPC 实例路由的网络方案
- `vpc-secondary-ip-`, 开头表示基于弹性网卡的网络方案

2 修改 CNI 配置文件模板

根据步骤 1 中获取的网络模式，执行 `kubectl edit cm -n kube-system cce-cni-config-template`, 修改对应模式的 CNI 配置文模板。

手动编辑 `plugins` 列表, 在最后加上 sysctl 的配置文件，等待 1 分钟后所有节点的 CNI 配置将会更新。

修改完的样例如下:

```
cce-cni-secondary-ip-veth: |
{
   "name":"{{ .NetworkName }}",
   "cniVersion":"0.3.1",
   "plugins":[
      {
         "type":"ptp",
         "enableARPProxy":true,
         "vethPrefix":"veth",
         "mtu": {{ .VethMTU }},
         "ipam":{
            "type":"eni-ipam",
            "endpoint":"{{ .IPAMEndPoint }}",
            "instanceType":"{{ .InstanceType }}",
            "deleteENIScopeLinkRoute":true
         }
      },
      {
        "type":"sysctl",
        "kubeconfig":"/etc/cni/net.d/cce-cni.d/cce-cni.kubeconfig"
      }
   ]
}
```

3 集群粒度的配置（可选）

在步骤 2 中的配置里，可以继续添加 `sysctl` 字段，指明需要配置的参数。

如下的配置生效后，集群内所有新建的容器都会将 `/proc/sys/net/core/somaxconn` 设置为 500。

```
 {
  "type":"sysctl",
  "kubeconfig":"/etc/cni/net.d/cce-cni.d/cce-cni.kubeconfig",
  "sysctl":{
    "net.core.somaxconn":"8192"
  }
}
```

4 应用粒度的配置（可选）

在完成步骤 2 后，用户可以通过指定 Pod 的 annotations，传递 sysctl 参数的配置。

采用如下的 yaml 的创建的 nginx 容器，将会设置 `/proc/sys/net/core/somaxconn` 为 8192 以及 `/proc/sys/net/ipv4/tcp_tw_reuse` 为 1。

```
 kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-example
  template:
    metadata:
      labels:
        app: nginx-example
      annotations:
        net.sysctl.cce.io/net.core.somaxconn: "8192"
        net.sysctl.cce.io/net.ipv4.tcp_tw_reuse: "1"
    spec:
      containers:
        - name: nginx-example
          image: nginx:alpine
          imagePullPolicy: IfNotPresent
```

注意事项

用户自定义的 CNI 插件，必须满足以下条件：

- 确保 CNI 插件的 type 字段不为空且不要与已有插件重复；
- 确保 CNI 插件的可执行程序/opt/cni/bin/{user-cni} 存在且可执行，否则 kubelet 无法正常创建 Pod；
- 确保 CNI 插件完整的遵循了 CNI 规范，确保 CNI DEL 操作可以正常执行。防止 CNI 插件异常导致 Pod 无法删除重建，持续在故障中无法恢复。
- 确保 CNI 配置是一个合法的 JSON 格式文件
- 不同版本内核中 netns 可配置的 sysctl 参数不同，请小心配置否则容器可能无法启动

CCE Resource Recommender 用户文档

组件介绍

- Kubernetes 提升业务编排和资源利用率
  - Kubernetes 可以有效的提升业务编排能力和资源利用率，但如果没有额外的能力支撑，提升的能力十分有限。
  - Kubernetes 集群的资源利用率不高的主要原因是根据 Kubernetes 的资源调度逻辑，在创建 Kubernetes 工作负载时，通常需要为工作负载配置合适的资源 Request，表示对资源的占用和限制，其中对利用率影响最大的是 Request。
  - 为防止自己的工作负载所用的资源被别的工作负载所占用，或者是为了应对高峰流量时的资源消耗诉求，用户习惯于为 Request 设置较大的数值。
  - Request 和实际使用资源之间的差值，是不能被其它工作负载所使用的，因此造成了浪费。
  - Request 数值设置不合理，造成了 Kubernetes 集群资源利用率低。
- 百度智能云 CCE 支持在集群中安装 CCE Resource Recommender 组件。CCE Resource Recommender 组件 可以为 Kubernetes 的 Workload 推荐容器级别资源的 Request 数值，减少资源浪费

部署在集群内的资源对象

安装 CCE Resource Recommender 组件，将在集群内部署以下 Kubernetes 对象：

Kubernetes 对象名称	类型	所属 Namespaces
analytics.analysis.baidubce.com	CustomResourceDefinition	-
recommendations.analysis.baidubce.com	CustomResourceDefinition	-
analysis-default	Analytics	kube-system
recommendation-configuration	ConfigMap	kube-system
recommenderd	ClusterRole	-
recommenderd	ClusterRoleBinding	-
recommenderd	Service	kube-system
recommenderd	ClusterRoleBinding	kube-system
recommenderd	ServiceAccount	kube-system
recommenderd	Deployment	kube-system

功能说明

支持为 Deployment、StatefulSet、DaemonSet 中的每一个 Container 智能推荐合适的资源 Request。 支持维持 Request 比例：推荐的 Request 会维持初始 Workload 中 Container 设置的 Reqeust 之间的比例。

CCE Resource Recommender 推荐原理

组件在 kube-system 命名空间下创建 Analytics CR 对象，覆盖所有集群中的所有 Kubernetes 原生工作负载（Deployment、DaemonSet、StatefulSet），会分析工作负载最长 14 天的监控数据数据，12 小时更新一次推荐值。 然后根据 Analytics 生成集群内每个工作负载的 Recommendation CR 对象，用于存储推荐的数据。 Recommendation CR 如果产生了推荐数据，就会把推荐数据写入到对应工作负载的 Annotation 里。

注意事项

环境要求

- Kubernetes 版本：1.18+
- 接入 Prometheus 监控服务（CProm 或 自建 Prometheus 数据源）

被控资源要求

- 支持 Deployment、StatefulSet、DaemonSet。
- 不支持 Job、CronJob，不支持不是由 Workload 管理的 Pod。

推荐计算

- 推荐最小值：单个容器推荐的 CPU 最小值是 0.125 核，即 125m；内存的最小值是 125Mi。
- 该组件会自动分析工作负载历史的监控数据，推荐合适的 Request 数值。
- 安装该组件非立即生效，为准确计算推荐值，需要分析该 Workload 的历史资源使用数据。
- 不同的 Workload 的计算时间长度可能不一致，集群中不同的 Workload 之间互相可能会有影响。
- 安装该组件后，对至少运行一天的 Workload 产生推荐数据。
- 对于安装组件后新建的 Workload，一般情况下，也需要一天的时间才会产生 Workload 的推荐数据。
- 建议工作负载稳定运行一段时间之后，再使用推荐值更新 Workload。

使用说明

安装组件

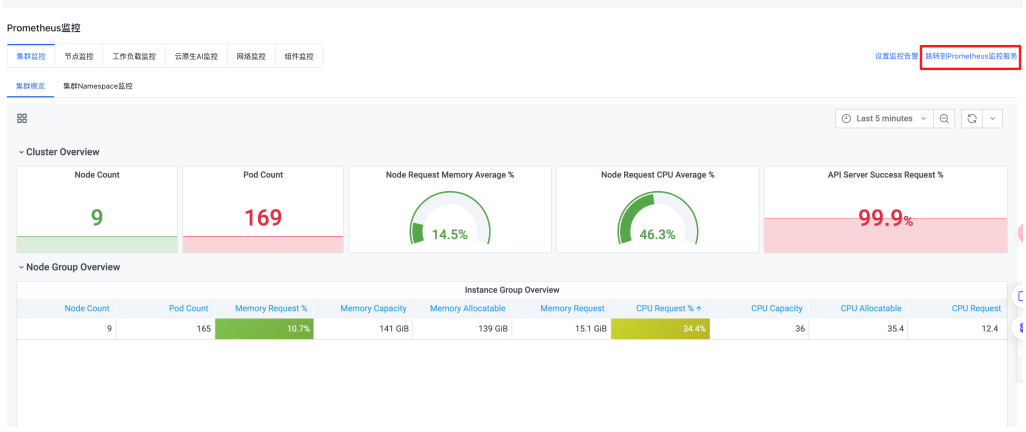
百度智能云官方 helm 模版安装。

安装流程

1、开启 CProm

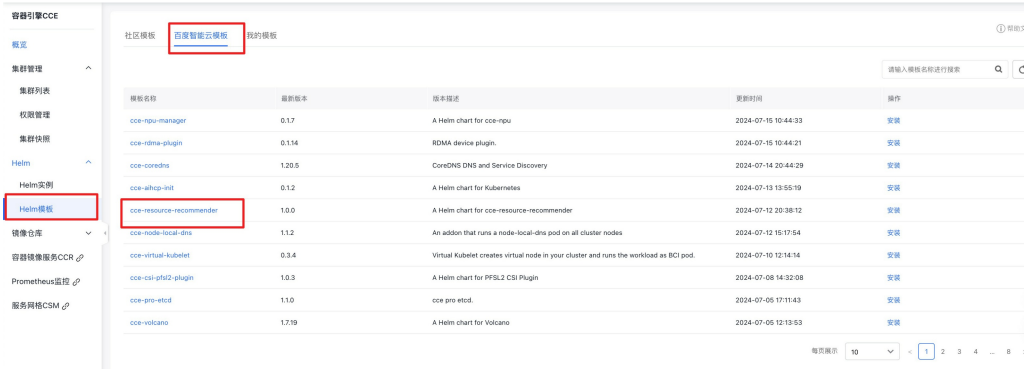
- 登录CCE管理控制台，选择“监控日志 > Prometheus监控”，点击“接入实例”。

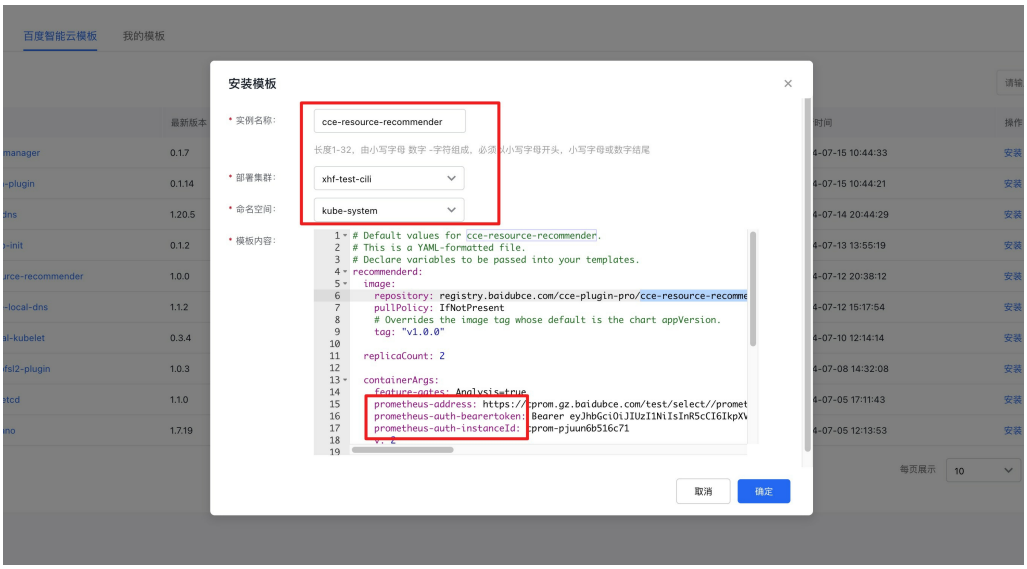
- 等待接入实例成功，单击"跳转到Prometheus监控服务"。



- 在监控实例页面复制 实例id 和 Remote Read 地址，且单击生成Token并复制。这些信息将用于下一步安装。

2、 安装 CCE Resource Recommender





参数	描述	默认值	是否必填
recommenderd.containerArgs.prometheus-address	recommenderd 的 Prometheus 地址	空（Cprom的地址后面需加 /prometheus）	是
recommenderd.containerArgs.prometheus-auth-instanceId	recommenderd 的 CProm 实例ID	空	是
recommenderd.containerArgs.prometheus-auth-bearertoken	recommenderd 的 CProm Token信息	空	是
analysisDefault.enable	是否开启全局资源推荐默认配置		
默认配置会	true	否	

- 添加参数示例

```
#### Default values for cce-resource-recommender.
#### This is a YAML-formatted file.
#### Declare variables to be passed into your templates.
recommenderd:
  image:
    repository: registry.baidubce.com/cce-plugin-pro/cce-resource-recommender
    pullPolicy: IfNotPresent
    # Overrides the image tag whose default is the chart appVersion.
    tag: "v1.0.0"

  replicaCount: 2

  containerArgs:
    feature-gates: Analysis=true
    v: 2
  prometheus-address: https://cprom.gz.baidubce.com/test/select/prometheus
    prometheus-auth-instanceId: cprom-pjuun6b516c71
    prometheus-auth-bearertoken: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lc3BhY2UiOiJjcHJvbS1wanV1bjZiNTE2YzcxIiwic2VjcmV0TmFtZSI6ImVjYTk3ZTE0OGNiNzRlOTY4MA

  podAnnotations: {}

  resources: {}

  nodeSelector: {}

  tolerations: []

  affinity: {}

analysisDefault:
  enable: true
```

3、 验证安装是否成功

使用如下命令检查安装的 Deployment 是否正常：

```
kubectl get deploy recommenderd -n kube-system
```

结果类似如下：

```
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
recommenderd   2/2     2            2           37s
```

卸载 CCE Resource Recommender

```
helm --kubeconfig {$kubeconfig} uninstall cce-resource-recommender -n kube-system
```

后台获取推荐数值

工作负载

CCE Resource Recommender 组件会将推荐值保存至对应工作负载的YAML文件中，您可以通过标准的 Kuberentes API 获取每个工作负载的推荐值，然后集成到业务的发布系统中。如下所示查看工作负载下每个容器的 Request 推荐量：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    analysis.baidubce.com/resource-recommendation: |
      containers:
      # 若一个 Pod 里有多个容器，每个容器都有 CPU 和 Memory 的 Request 的推荐值
      - containerName: nginx
        target:
          cpu: 125m
          memory: 125Mi #若这里缺少单位，显示的是字符串"58243235"，省略的单位是byte
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2024-06-11T03:15:57Z"
  generation: 1
  labels:
    app: nginx
```

```
  name: deployment-example
  namespace: default
  resourceVersion: "1118119"
  uid: 8b6d54d9-c683-4e76-a95e-658e14a954b1
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: hub.baidubce.com/cce/nginx-alpine-go:latest
        imagePullPolicy: Always
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /
            port: 80
            scheme: HTTP
          initialDelaySeconds: 20
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 5
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /
            port: 80
            scheme: HTTP
          initialDelaySeconds: 5
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 1
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

Recommendation CR

CCE Resource Recommender 组件会根据分析结果会生成集群内每个工作负载的 Recommendation CR 对象，用于存储推荐数据，同时也会将推荐值保存到工作负载的 YAML 中

```yaml
apiVersion: analysis.baidubce.com/v1alpha1
kind: Recommendation
metadata:
  annotations:
    analysis.baidubce.com/run-number: "1"
  creationTimestamp: "2024-06-11T04:54:27Z"
  generateName: analysis-default-resource-
  generation: 2
  labels:
    analysis.baidubce.com/analytics-uid: 83cccfd5-b3c5-45aa-a92a-d9dd607dc75f
    analysis.baidubce.com/recommendation-rule-name: analysis-default
    analysis.baidubce.com/recommendation-rule-recommender: Resource
    analysis.baidubce.com/recommendation-rule-uid: bce27929-64d6-4b5f-89e4-001cbed5ed64
    analysis.baidubce.com/recommendation-target-kind: StatefulSet
    analysis.baidubce.com/recommendation-target-name: agent-q7vl19h81
    analysis.baidubce.com/recommendation-target-version: v1
    app.kubernetes.io/component: vmagent
    app.kubernetes.io/instance: agent-q7vl19h81
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: monitor-agent
    app.kubernetes.io/version: 0.2.0
    helm.sh/chart: monitor-agent-0.3.6
  name: analysis-default-resource-2gzjt
  namespace: default
  ownerReferences:
  - apiVersion: analysis.baidubce.com/v1alpha1
    blockOwnerDeletion: false
    controller: false
    kind: RecommendationRule
    name: analysis-default
    uid: bce27929-64d6-4b5f-89e4-001cbed5ed64
  resourceVersion: "1118082"
  uid: 4793159d-83b5-45db-8c34-c3443a7c45cd
spec:
  adoptionType: StatusAndAnnotation
  completionStrategy:
    completionStrategyType: Once
  targetRef:
    apiVersion: apps/v1
    kind: StatefulSet
    name: agent-q7vl19h81
    namespace: cprom-system
  type: Resource
status:
  action: Patch
  conditions:
  - lastTransitionTime: "2024-06-11T04:54:28Z"
    message: Recommendation is ready
    reason: RecommendationReady
    status: "True"
    type: Ready
  currentInfo: '{"spec":{"template":{"spec":{"containers":[{"name":"sidecar","resources":{"requests":{"cpu":"100m","memory":"100Mi"}}},
{"name":"vmagent","resources":{"requests":{"cpu":"100m","memory":"100Mi"}}}]}}}}'
  lastUpdateTime: "2024-06-11T04:54:28Z"
  recommendedInfo: '{"spec":{"template":{"spec":{"containers":[{"name":"sidecar","resources":{"requests":
{"cpu":"125m","memory":"125Mi"}}},{"name":"vmagent","resources":{"requests":{"cpu":"125m","memory":"125Mi"}}}]}}}}'
  recommendedValue: |
    resourceRequest:
      containers:
      - containerName: sidecar
        target:
          cpu: 125m
          memory: 125Mi
      - containerName: vmagent
        target:
          cpu: 125m
          memory: 125Mi
  targetRef: {}
```

在该示例中：

- 推荐的 TargetRef 指向 cprom-system 的 StatefulSet：agent-q7vl19h81
- 推荐类型为资源推荐
- adoptionType 是 StatusAndAnnotation，表示将推荐结果展示在 recommendation.status 和 Deployment 的 Annotation
- recommendedInfo 显示了推荐的资源配置，currentInfo 显示了当前的资源配置，格式是 Json ，可以通过 Kubectl Patch 将推荐结果更新到 TargetRef

Pod异常问题排查

1.调度问题

Pod未调度到节点

如果Pod长时间处于“未调度到节点”（Pending）状态，没有被安排到任何节点上运行，可能是出于以下原因。

报错信息	说明	推荐的解决方案
no nodes available to schedule pods.	当前集群中无可用节点可供调度。	1. 查看集群节点状态是否为NotReady。如果存在节点处于NotReady状态，则对问题节点进行检查和修复。 2. 检查Pod中是否声明了nodeSelector、nodeAffinity或污点容忍。若不存在亲和性策略，可以增加节点。
0/x nodes are available: x Insufficient cpu. 0/x nodes are available: x Insufficient memory.	集群中没有可用节点能够满足Pod所需的CPU或内存资源。	在节点页面查看容器组、CPU、内存的使用情况，确定集群的资源使用率。 **说明** 当某个节点的CPU和内存利用率维持在较低水平时，即便引入一个新Pod不会立即达到资源使用的上限，调度器也会谨慎考虑，以防该Pod的加入导致未来高峰时段节点资源紧张，避免因资源分配不当而引发的节点资源短缺问题。 若集群中的CPU或内存已经耗尽，可参考如下方法处理。 1.删除或减少不必要的Pod 2.根据自身业务情况，调整Pod的资源配置，重新配置容器Request和Limit。 3.根据Pod声明的亲和性策略在集群中添加新的节点。
x node(s) didn't match node selector. x node(s) didn't match pod affinity/anti-affinity.	集群现有节点没有匹配Pod声明的节点亲和性（nodeSelector）要求或Pod亲和性（podAffinity和podAnitiAffinity）要求。	1. 检查并调整Pod的节点亲和性策略，包括节点标签、nodeSelector、nodeAffinity、污点和容忍等。 2. 检查并调整Pod的Pod亲和性策略，评估节点是否满足Pod的亲和性要求：如果Pod配置了podAffinity，则需要检查目标节点上是否有匹配的Pod存在；如果配置了podAntiAffinity，则需确认目标节点上没有不应共存的Pod。
0/x nodes are available: x node(s) had volume node affinity conflict.	Pod所使用的存储卷与待调度的节点之间存在亲和性冲突，云盘无法跨可用区挂载，导致调度失败。	检查并调整Pod和存储卷的节点亲和性策略，包括节点标签、nodeSelector、nodeAffinity、污点和容忍等。
0/x nodes are available: x node(s) had taints that the pod didn't tolerate.	需要调度的节点打上了污点，不允许Pod调度到该节点上。	1.如果污点是由您手动添加，您可以删除非预期的污点。 2.如果无法删除污点，可以为Pod配置相应的容忍。 3.如果污点为系统自动添加，根据污点内容排查和解决问题，并等待Pod重新调度。
0/x nodes are available: pod has unbound immediate PersistentVolumeClaims.	Pod绑定PVC失败。	检查Pod所指定的PVC或PV是否已经创建，通过kubectl describe pvc 或 kubectl describe pv 命令查看PVC、PV的Event信息，进一步进行判断。

Pod已调度到节点

如果Pod已经被调度到某个节点上但仍处于Pending状态，请参见下文解决。

1. 判断Pod是否配置了hostPort：如果Pod配置了hostPort，那么每个节点上只能运行一个使用该hostPort的Pod实例。因此，Deployment或ReplicationController中Replicas值不能超过集群中的节点数。如果该端口被其他应用占用，将导致Pod调度失败。hostPort会带来一些管理和调度上的复杂性，推荐您使用Service来访问Pod。
2. 如果Pod没有配置hostPort，请参见下方步骤排查。
   a. 通过kubectl describe pod 命令查看Pod的Event信息，并解决对应的问题。Event可能会解释Pod启动失败的原因，例如镜像拉取失败、资源不足、安全策略限制、配置错误等。

b. Event中没有有效信息时，进一步查看该节点kubelet的日志，进一步排查Pod启动过程中存在的问题。您可以通过grep -i /var/log/messages* | less命令搜索系统日志文件（/var/log/messages*）中包含指定Pod名称的日志条目。

2.镜像拉取问题

报错信息	说明	推荐的解决方案
Failed to pull image "xxx": rpc error: code = Unknown desc = Error response from daemon: Get xxx: denied:	请求访问镜像仓库时被拒绝，创建Pod时未指定imagePullSecret。	检查Pod YAML中spec.template.imagePullSecrets指定的Secret是否存在。 使用CCR企业版时，可以使用免密组件拉取镜像，请参见https://cloud.baidu.com/doc/CCE/s/4m0kru8g5。
Failed to pull image "xxxx:xxx": rpc error: code = Unknown desc = Error response from daemon: Get https://xxxxxx/xxxxx/: dial tcp: lookup xxxxxxx.xxxxx: no such host	通过HTTPS协议从指定的镜像仓库地址拉取镜像时，镜像地址解析失败。	1. 检查Pod YAML中spec.containers.image配置的镜像仓库地址是否正确。如有误，需修改为正确地址。 2. 如地址无误，进一步验证从Pod所在节点到镜像仓库的网络连接是否通畅。
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "xxxxxxxxx": Error response from daemon: mkdir xxxxx: no space left on device	节点磁盘空间不足。	登录到Pod所在节点，运行df -h查看磁盘空间状态。如磁盘已满，请扩容磁盘
Failed to pull image "XXX": rpc error: code = Unknown desc = context canceled	操作取消，可能是由于镜像文件过大。Kubernetes默认存在拉取镜像超时时间，如果一定时间内镜像下载没有任何进度更新，Kubernetes会认为此操作异常或处于无响应状态，主动取消该任务。	1. 检查Pod YAML中配置的imagePullPolicy是否为IfNotPresent。 2. 登录到Pod所在节点，运行cmd命令docker pull或ctr images pull判断镜像是否可以被拉取。
Failed to pull image "xxxxx": rpc error: code = Unknown desc = Error response from daemon: Get https://xxxxxxx: xxxxx/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)	无法连接镜像仓库，网络不通。	1. 登录到Pod所在节点，运行cmd命令curl https://xxxxxx/xxxxx/ 判断地址是否可以访问。如有报错，进一步判断是否存在网络配置、防火墙规则、DNS解析等网络访问问题。 2. 判断节点的公网策略是否正常，例如负载均衡BLB、安全组、绑定的弹性公网IP等配置。
一直显示Pulling image	可能触发了kubelet的镜像拉取限流机制。	通过自定义Kubelet参数功能调整registryPullQPS（镜像仓库的QPS上限）和registryBurst（突发性镜像拉取的个数上限）。

3.启动问题

Pod处于init状态

错误信息	说明	推荐的解决方案
停留在Init:N/M状态	该Pod包含M个Init容器，其中N个已经启动完成，但仍有M-N个Init容器未启动成功。	1. 通过kubectl describe pod -n 命令查看Pod的事件，确认当前Pod中未启动的Init容器是否存在异常。 2. 通过kubectl logs -n -c 命令查看Pod中未启动的Init容器的日志，通过日志内容排查问题。 3. 查看Pod的配置，例如检查健康检查配置，进一步确认未启动的Init容器配置是否正常。 关于Init容器的更多信息，请参见调试Init容器。
停留在Init:Error状态	Pod中的Init容器启动失败。	同上
停留在Init:CrashLoopBackOff状态	Pod中的Init容器启动失败并处于反复重启状态。	同上

Pod创建中（Creating）

错误信息	说明	推荐的解决方案
Network plugin returns error: cni plugin not initialized	CNI网络组件未初始化导致pod无法创建	查看pod所在节点cce-network-agent和cce-network-operator状态，收集日志根据日志内容排查问题或提工单解决。
ce-network-agent is not running or ready on this node	cce-network-agent异常导致pod无法创建。	查看pod所在节点cce-network-agent状态，收集日志根据日志内容排查问题或提工单解决。

Pod启动失败（CrashLoopBackOff）

错误信息	说明	推荐的解决方案
日志中存在exit(0)。	日志中存在exit(0)。	1. 登录异常工作负载所在的节点。 2. 执行docker ps -a
Event信息中存在Liveness probe failed: Get http…。	健康检查失败。	核查Pod中所配置的容器健康检查（Liveness Probe）策略是否符合预期，能有效地反映出容器内应用程序的实际运行状况。
Pod日志中存在no left space。	磁盘空间不足。	1.扩容磁盘。 2.清理不必要的镜像，释放磁盘空间，并按需配置imageGCHighThresholdPercent，控制节点上触发镜像垃圾回收（Image GC）的阈值。
启动失败，无Event信息。	Pod中声明的Limit资源少于实际所需资源时，会导致启动容器失败。	检查Pod的资源配置是否正确。重新配置容器Request和Limit。
Pod日志中出现Address already in use。	同一Pod中的Container端口存在冲突。	1. 检查Pod是否配置了hostNetwork: true，这意味着Pod内的容器会直接与宿主机共享网络接口和端口空间。如果无需使用，请改为hostNetwork: false。 2. 如果Pod需要使用hostNetwork: true，请配置Pod的反亲和性，确保同一副本集中的Pod被调度到不通节点。 3. 检查并确保不存在也不会有两个或多个具有相同端口需求的Pod运行在同一台节点上。
Pod日志中出现container init caused "setenv: invalid argument"": unknown。	工作负载中挂载了Secret，但Secret对应的值没有进行Base64加密。	1.通过控制台创建Secret，Secret对应的值会自动进行Base64加密。 2.通过YAML创建Secret，并执行echo -n "xxxxx"
自身业务问题。	-	查看Pod日志，通过日志内容排查问题。

4.Pod运行问题

OOM

当集群中的容器使用超过其限制的内存，容器可能会被终止，触发OOM（Out Of Memory）事件，导致容器异常退出。关于OOM事件，请参见为容器和Pod分配内存资源。

- 若被终止的进程为容器的阻塞进程，可能会导致容器异常重启。
- 若集群配置了集群容器副本异常报警，OOM事件出现时会收到相关报警信息。

Terminating

可能原因	说明	推荐的解决方案
节点存在异常，处于NotReady状态。	-	处于NotReady状态的节点恢复正常后会被自动删除。
Pod配置了Finalizers。	如果Pod配置了Finalizers，Kubernetes会在删除Pod之前执行Finalizers指定的清理操作。如果相关的清理操作没有正常响应，Pod将保持在Terminating状态。	通过kubectl get pod -n -o yaml查看Pod的Finalizers配置，进一步排查异常原因。
Pod的preStop配置异常。	如果Pod配置了preStop，Kubernetes会在容器被终止之前执行preStop指定的操作。Pod正处于终止流程的preStop阶段时，Pod将处于Terminating状态。	通过kubectl get pod -n -o yaml查看Pod的preStop配置，进一步排查异常原因。
Pod配置了优雅退出时间。	如果Pod配置了优雅退出时间（terminationGracePeriodSeconds），Pod收到终止命令后（例如kubectl delete pod <pod_name>命令）会进入Terminating状态。等待terminationGracePeriodSeconds设定的时间后，或容器提前退出后，Kubernetes才认为Pod已经成功关闭。	等待容器优雅退出后，Kubernetes将自动删除Pod。
容器无响应。	发起停止或删除Pod的请求后，Kubernetes会向Pod内的容器发送SIGTERM信号。如果容器在终止过程中没有正确响应SIGTERM信号，Pod可能会停留在Terminating状态。	1. 使用kubectl delete pod --grace-period=0 --force强制删除，释放Pod资源。 2. 检查Pod所在节点的containerd或Docker日志，进一步进行排查。

Evicted

可能原因	说明	推荐的解决方案
节点存在资源压力，包括内存不足、磁盘空间不足等，引发kubelet主动驱逐节点上的一个或者多个Pod，以回收节点资源。	可能存在内存压力、磁盘压力、Pid压力等。可以通过kubectl describe node | grep Taints命令查询。 1.内存压力：带有污点node.kubernetes.io/memory-pressure。 2.磁盘压力：带有污点node.kubernetes.io/disk-pressure。 3.Pid压力：带有污点node.kubernetes.io/pid-pressure。	1.内存压力： 根据自身业务情况，调整Pod的资源配置。或增加节点。 2.磁盘压力：定时清理节点上的业务Pod日志，防止磁盘空间被耗尽。或为节点进行磁盘扩容。 3.Pid压力：根据自身业务情况，调整Pod的资源配置，请参见进程ID约束与预留。
发生了非预期的驱逐行为。	待运行Pod的节点被手动打上了NoExecute的污点，导致出现非预期的驱逐行为。	通过kubectl describe node | grep Taints命令检查节点是否被打上了NoExecute污点。如是，请删除。
未按照预期流程执行驱逐。	--pod-eviction-timeout（before 1.26）：当节点宕机时间超过设置时间后，开始驱逐宕机节点上的Pod，默认为5min。 --node-eviction-rate：每秒从节点上驱逐的Pod数量。默认为0.1，即每10s至多从一个节点上驱逐Pod。 --secondary-node-eviction-rate：第二档的节点驱逐速率。当集群中宕机节点过多时，节点驱逐速率会降低至第二档，默认值为0.01。 --unhealthy-zone-threshold：可用区的不健康阈值，默认为0.55，即当宕机的节点数量超过总节点数的55%时，该可用区被判定为不健康。 --large-cluster-size-threshold：集群的大规模阈值，默认为50，即当集群节点数量超过50时判定集群为大规模集群。	在小规格的集群（集群节点数小于等于50个节点）中，如果故障的节点大于总节点数的55%，实例的驱逐会被停止，更多信息请参见节点驱逐速率限制。 在大规模集群中（集群节点数大于50），如果集群中不健康的节点数量占总节点数的比例超过了预设的阈值--unhealthy-zone-threshold（默认为0.55），驱逐速率由--secondary-node-eviction-rate控制（代表每分钟驱逐节点上Pod的最大比例），默认值为0.01。更多信息，请参见节点驱逐速率限制。
容器被驱逐后仍然频繁调度到原节点。	节点驱逐容器时会根据节点的资源使用率进行判断，而容器的调度规则是根据节点上的“资源分配量”进行判断，被驱逐的Pod有可能被再次调度到这个节点，从而出现频繁调度到原节点的现象。	根据集群节点的可分配资源检查Pod的资源Request请求配置是否合理

Completed

Completed状态下，Pod中容器的启动命令已执行完毕，容器中的所有进程均已成功退出。Completed状态通常适用于Job、Init容器等。

5.其他常见问题

Pod状态为Running但没正常工作

如果您的业务YAML存在问题，Pod可能会处于Running状态但没有正常工作。您可以参见以下流程解决。

1. 查看Pod的配置，确定Pod中容器的配置是否符合预期。
2. 使用以下方法，排查环境变量中的某一个Key是否存在拼写错误。创建Pod时，如果环境变量中的某个Key拼写错误，集群会忽略该错误并使用该YAML成功创建资源。但在容器运行过程中，系统无法执行YAML文件中指定的命令。
   a. 在执行kubectl apply -f命令前为其添加--validate，然后执行kubectl apply --validate -f XXX.yaml 命令。如拼写存在错误，会提示报错。
   b. 执行以下命令，将输出结果的pod.yaml与您创建Pod使用的YAML进行对比。

```
kubectl get pods [$Pod] -o yaml > pod.yaml
```

1.pod.yaml文件比您创建Pod所使用的文件行数更多，表明已创建的Pod符合预期。
2.如果您创建Pod的YAML代码行不存在于pod.yaml文件中，表明YAML中存在拼写问题。

3. 查看Pod的日志，通过日志内容排查问题。
4. 通过终端进入容器，查看容器内的本地文件是否符合预期。

利用 Velero 实现将应用跨云迁移到 百度CCE

1.简介

CCE 支持用户使用开源工具 Velero 对云上及私有环境的 Kubernetes 集群进行备份、恢复。用于集群误操作、集群故障、集群迁移等场景。

velero的作用：

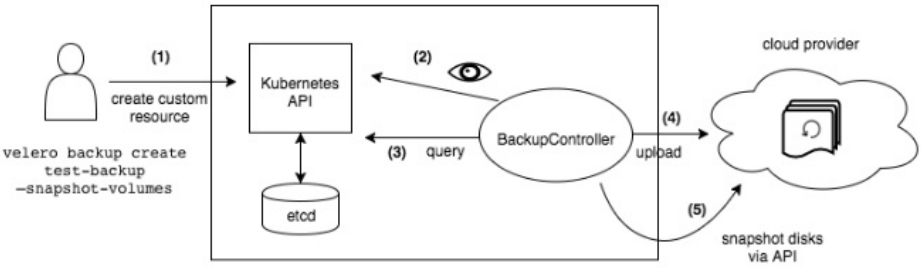
- 灾备能力：提供备份恢复k8s集群的能力
- 迁移能力：提供拷贝集群资源到其他集群的能力

和 etcd 备份的区别：

- etcd 的备份必须拥有 etcd 运维权限，CCE上托管型集群用户无法操作 etcd
- etcd 更适合单集群内数据备份，不太适合集群迁移
- etcd 是当前状态备份，velero 可以做到只备份集群内的一部分资源

Velero 工作原理图如下图所示（来源于 Velero 官网），当用户执行备份命令时，备份过程说明如下：

1. 调用自定义资源 API 创建备份对象（1）。
2. BackupController 控制器检测到生成的备份对象时（2）执行备份操作（3）。
3. 将备份的集群资源和存储卷快照上传到 Velero 的后端存储（4）和（5）。



另外当执行还原操作时，Velero 会将指定备份对象的数据从后端存储同步到 Kubernetes 集群完成还原工作。 更多关于 Velero 介绍，请参见Velero 官网文档。

本文将介绍如何使用 百度云对象存储 BOS 作为 Velero 后端存储实现从阿里云ACK到百度智能云CCE的集群备份。

2.安装及配置

如果要将 ACK 上的应用迁移到 CCE 上，需要安装并配置 velero 和 velero-baiducloud-plugin。

- 在百度云上创建你的 BOS bucket
- 在百度云上获取你的 AccessKey 和 SecretKey
- 获取百度云 CCE 发布的迁移组件 CCE-Backup-Controller，在阿里云配置容器镜像示例，并推送 CCE-Backup-Controller 相关镜像
- 安装 Velero 命令工具及服务端
- 根据实际情况修改Helm Values
- 阿里云 ACK 集群安装部署 CCE-Backup-Controller

2.1 在百度云对象存储创建 BOS Bucket

Velero 需要对象存储来存放备份数据，在 百度云上创建你的BOS bucket。

创建 Bucket 的过程参考：对象存储 BOS - 创建 Bucket

示例：

- bucket 名称：cce-bos-test

2.2 获取百度云用户的 AccessKey 和 SecretKey

关于 AK SK 的更多信息参考： 如何获取 AK 和 SK

2.3 确认环境

- 源集群：
  - 地域：北京

- 阿里 ACK 集群 ID

- 备份到 BOS：
  - bucket：cce-bos-test（示例）
- 目标集群：
  - 地域：北京
  - 百度 CCE 集群 ID

2.4 安装 Velero 命令工具及服务端

2.4.1 安装 Velero 命令行工具 方法一：MacOS - Homebrew

在 macOS，您可以在终端使用Homebrew 完成 velero 客户端的安装：

`brew install velero`

方法二：Github release

- 从 Velero release 页面 下载和使用平台匹配的压缩包
- 解压压缩包

  `tar -xvf <RELEASE-TARBALL-NAME>.tar.gz`

  3.将解压得到的 velero 二进制移动到您的 $PATH 环境变量包含的路径下（如 /usr/local/bin）

方法三：Windows - Chocolatey

在 Windows 平台，您可以在终端使用 Chocolatey 完成 velero 客户端的安装。

`choco install velero` **2.4.2 在阿里ACK安装 CCE-Backup-Controller** 我们主要利用 Helm 完成 Velero 服务端的安装。

步骤一：下载 CCE-Backup-Controller Helm Chart，并解压

步骤二：将 CCE-Backup-Controller Helm Chart 中的镜像推送到 阿里云镜像服务，下文统一以阿里云镜像作为示例。

镜像列表：

```
 registry.baidubce.com/cce-plugin-pro/velero:v1.13.5
registry.baidubce.com/cce-plugin-pro/velero-plugin-baiducloud:0.6
registry.baidubce.com/cce-plugin-pro/kubectl:1.24.9
```

推送到阿里云镜像服务后，镜像列表为：

```
 registry-vpc.cn-beijing.aliyuncs.com/velero-test/velero:v1.13.5
registry-vpc.cn-beijing.aliyuncs.com/velero-test/velero-plugin-baiducloud:0.6
registry-vpc.cn-beijing.aliyuncs.com/velero-test/kubectl:1.24.9
```

步骤三：配置 CCE-Backup-Controller Chart

安装执行部署 Velero 服务端之前，需要对 Velero 的 Helm Chart 进行正确的配置，即需要正确编辑 Chart 中的 values.yaml 文件。

1. 替换核心镜像。为了在未开通外网的 CCE 环境中顺利拉取 Velero Helm Chart 中的核心镜像，需要修改 values.yaml 中的镜像地址

- registry.baidubce.com/cce-plugin-pro/velero:v1.13.2 替换为 registry-vpc.cn-beijing.aliyuncs.com/velero-test/velero:v1.13.2
- registry.baidubce.com/cce-plugin-pro/velero-plugin-baiducloud:0.6 替换为 registry-vpc.cn-beijing.aliyuncs.com/velero-test/velero-plugin-baiducloud:0.6
- registry-vpc.cn-beijing.aliyuncs.com/velero-test/kubectl:1.24.9 替换为 registry-vpc.cn-beijing.aliyuncs.com/velero-test/kubectl:1.24.9 修改后，示例 yaml 如下：

```
  #### Details of the container image to use in the Velero deployment & daemonset (if
 #### enabling node-agent). Required.
 image:
  repository: registry-vpc.cn-beijing.aliyuncs.com/velero-test/velero
  tag: v1.13.2
  # Digest value example: sha256:d238835e151cec91c6a811fe3a89a66d3231d9f64d09e5f3c49552672d271f38.
  # If used, it will take precedence over the image.tag.
  # digest:
  pullPolicy: IfNotPresent
  # One or more secrets to be used when pulling images
  imagePullSecrets: []
  # - registrySecretName

 kubectl:
  image:
    repository: registry-vpc.cn-beijing.aliyuncs.com/velero-test/kubectl
    tag: 1.24.9
    # Digest value example: sha256:d238835e151cec91c6a811fe3a89a66d3231d9f64d09e5f3c49552672d271f38.
    # If used, it will take precedence over the kubectl.image.tag.
    # digest:
    # kubectl image tag. If used, it will take precedence over the cluster Kubernetes version.
    # tag: 1.16.15
```

2.添加 百度云 Velero 插件。为了能够顺利将集群备份文件保存到 百度云 BOS，CCE 提供了 Velero 插件。该配置需要添加到 initContainer 配置中。修改后示例 Yaml 如下：

```
 initContainers:
 - name: velero-plugin-baiducloud
   image: hub.baidubce.com/jpaas-public/velero-plugin-baiducloud:v0.3
   imagePullPolicy: IfNotPresent
   env:
   - name: BAIDU_CLOUD_CREDENTIALS_FILE
     value: "/credentials/cloud"
   volumeMounts:
     - mountPath: /target
       name: plugins
     - mountPath: /credentials
       name: cloud-credentials
```

3. 配置 百度云 BOS 访问凭证。注意：

- 将 <BAIDU_CLOUD_ACCESS_KEY_ID> 和 <BAIDU_CLOUD_SECRET_ACCESS_KEY> 替换成您在 2.2 步骤获取的实际百度云上的 AccessKey 及 SecretAccessKey；
- 将 <BAIDU_CLOUD_BOS_ENDPOINT> 替换成您所在地域的 BOS 服务域名，BOS 服务在各地域的域名见：对象存储 BOS - 获取访问域名

```
  credentials:
  # Whether a secret should be used. Set to false if, for examples:
  # - using kube2iam or kiam to provide AWS IAM credentials instead of providing the key file. (AWS only)
  # - using workload identity instead of providing the key file. (Azure/GCP only)
  useSecret: true
  # Name of the secret to create if `useSecret` is true and `existingSecret` is empty
  name: baidu-bos-credentials
  # Name of a pre-existing secret (if any) in the Velero namespace
  # that should be used to get IAM account credentials. Optional.
  existingSecret:
  # Data to be stored in the Velero secret, if `useSecret` is true and `existingSecret` is empty.
  # As of the current Velero release, Velero only uses one secret key/value at a time.
  # The key must be named `cloud`, and the value corresponds to the entire content of your IAM credentials file.
  # Note that the format will be different for different providers, please check their documentation.
  # Here is a list of documentation for plugins maintained by the Velero team:
  # [AWS] https://github.com/vmware-tanzu/velero-plugin-for-aws/blob/main/README.md
  # [GCP] https://github.com/vmware-tanzu/velero-plugin-for-gcp/blob/main/README.md
  # [Azure] https://github.com/vmware-tanzu/velero-plugin-for-microsoft-azure/blob/main/README.md
  secretContents:
   cloud: |
      BAIDU_CLOUD_BOS_ENDPOINT=<BAIDU_CLOUD_BOS_ENDPOINT>
      BAIDU_CLOUD_ACCESS_KEY_ID=<BAIDU_CLOUD_ACCESS_KEY_ID>
      BAIDU_CLOUD_SECRET_ACCESS_KEY=<BAIDU_CLOUD_SECRET_ACCESS_KEY>
```

4. 配置备份存储位置。该部分主要配置备份文件的存储位置，在使用 BOS 的场景下，实现将集群文件备份到 BOS 的指定 Bucket 及目录。注意：

- 将 <BUCKET_NAME> 替换为您在 BOS 创建的 Bucket 名称
- 将 <CLUSTER_ID> 替换为您的 ACK 集群 ID

```
  #### Source: cce-backup-controller/templates/backupstoragelocation.yaml
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  name: default
  namespace: cce-backup
  labels:
    app.kubernetes.io/name: cce-backup-controller
    app.kubernetes.io/instance: test-dhr
    app.kubernetes.io/managed-by: Helm
    helm.sh/chart: cce-backup-controller-1.1.3
spec:
  credential:
    name: baidu-bos-credentials
    key: cloud
  provider: baiducloud
  accessMode: ReadWrite
  objectStorage:
    bucket: "cce-bos-test"
    prefix: "ceb93762c98b9496889487187c6423196"
  config:
    checksumAlgorithm: "SHA256"
    region: "bj"
    s3ForcePathStyle: "true"
    s3Url: "https://s3.bj.bcebos.com"
    signatureVersion: "AWS4-HMAC-SHA256"
```

执行命令：

```
  kubectl --kubeconfig={kubeconfig} apply -f bkp-location.yaml
```

其他 Velero 相关配置请参考官方文档及 Helm Chart 中配置注释。

- https://velero.io/docs/v1.14/backup-reference/
- https://github.com/vmware-tanzu/helm-charts/blob/main/charts/velero/README.md

步骤三：部署及更新 CCE-Backup-Controller

使用 helm 命令完成 Velert 服务端的部署

```
cd cce-backup-controller\nhelm --kubeconfig={kubeconfig} install cce-backup-controller . -n cce-backup --create-namespace
```

每次修改 values.yaml 文件后，执行以下命令完成更新：

```
helm upgrade cce-backup-controller . -n velero
```

确认 velero 服务端正常运行：

```
kubectl -n cce-backup get pods
#### 结果
NAME                  READY   STATUS    RESTARTS   AGE\nvelero-76b4d755f5-fzztr   1/1     Running   0          49m
```

步骤四：创建备份和恢复任务

- 备份任务创建（命令行指令）

```
velero --kubeconfig {{kubeconfig-path}} backup create nginx-backup --include-namespaces default -n cce-backup
```

```
→  Desktop velero --kubeconfig ~/Desktop/ali.kubeconfig backup create nginx-backup --include-namespaces default -n cce-backup
Backup request "nginx-backup" submitted successfully.
Run `velero backup describe nginx-backup` or `velero backup logs nginx-backup` for more details.
```

- 查看备份任务是否成功

```
velero --kubeconfig {{kubeconfig-path}} -n cce-backup backup get
```

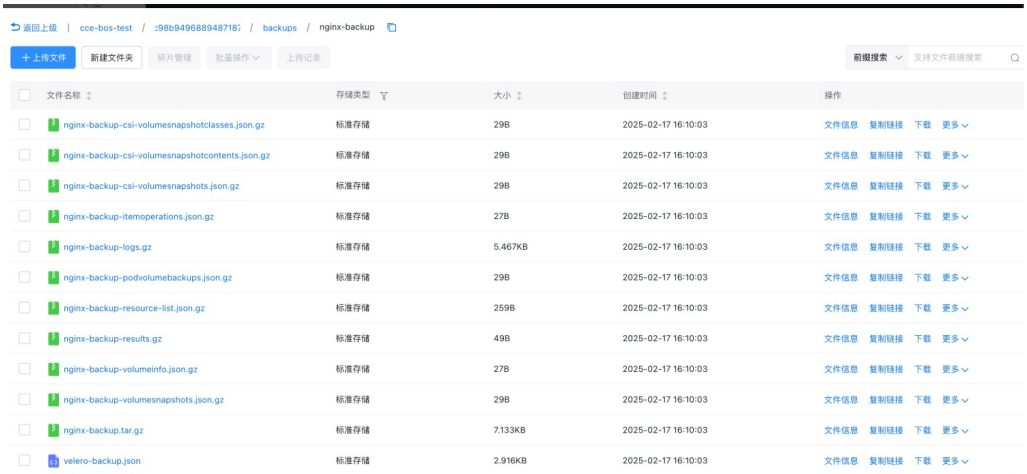
```
→  Desktop velero --kubeconfig ~/Desktop/ali.kubeconfig -n cce-backup backup get
NAME           STATUS      ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
nginx-backup   Completed   0        0          2025-02-17 16:10:00 +0800 CST   29d       default            <none>
```

2.5 查看备份数据

登陆 BOS 控制台，并选择目标 bucket。示例的 bucket 名称为 cce-bos-test:

- 查看 bucket 文件列表，其中每个集群的备份数据保存在单独的文件夹，如 cce-bos-test/ceb93762c98b9496889487187c6423196

- 查看集群的备份信息：



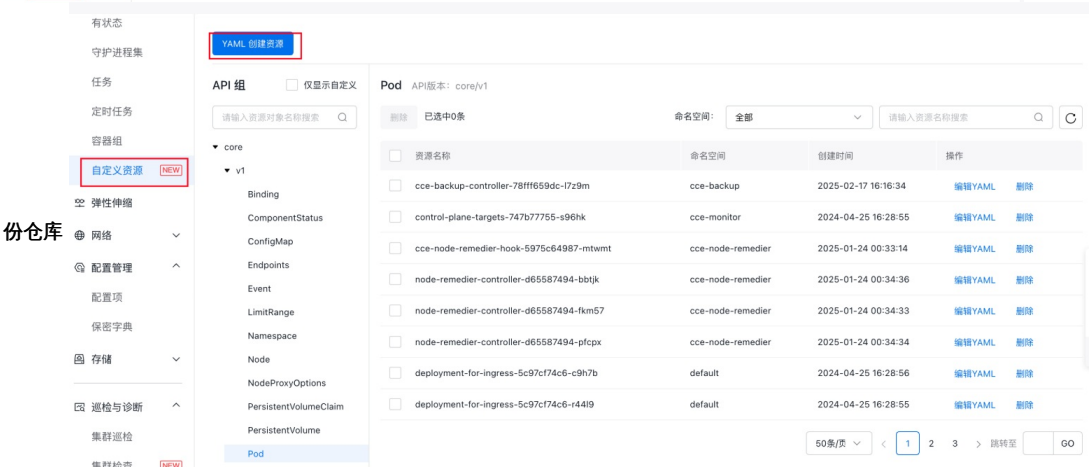
2.6 备份恢复

步骤一：迁移镜像，保证应用镜像在 CCE 可拉取

- 方法一：使用 CCR 镜像迁移功能，将 ACK 镜像迁移至 百度云 CCR
  - 恢复迁移应用到 CCE 后，需要手工修改镜像为 CCR 地址；
- 方法二：将 ACK 应用镜像修改为公网可访问，CCE 目标集群节点开通访问公网访问

步骤二：在目标恢复的 CCE 集群，通过组件中心安装 CCE-Backup-Controller

步骤三：通过命令行创建备份仓库



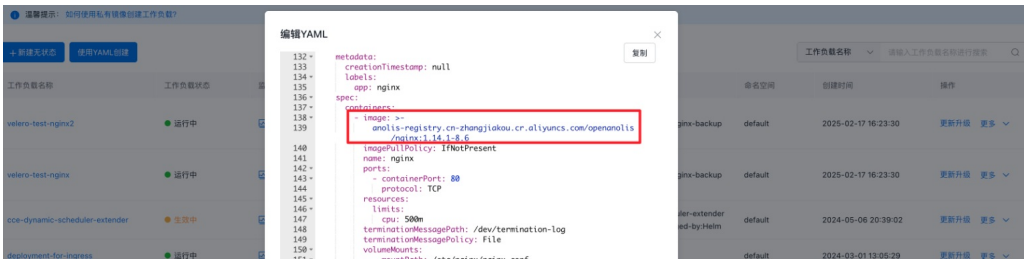

```
  #### Source: cce-backup-controller/templates/backupstoragelocation.yaml
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  name: default
  namespace: cce-backup
  labels:
    app.kubernetes.io/name: cce-backup-controller
    app.kubernetes.io/instance: test-dhr
    app.kubernetes.io/managed-by: Helm
    helm.sh/chart: cce-backup-controller-1.1.3
spec:
  credential:
    name: baidu-bos-credentials-bj
    key: cloud
  provider: baiducloud
  accessMode: ReadWrite
  objectStorage:
    bucket: "cce-bos-test"
    prefix: "ceb93762c98b9496889487187c6423196"
  config:
    checksumAlgorithm: "SHA256"
    region: "bj"
    s3ForcePathStyle: "true"
    s3Url: "https://s3.bj.bcebos.com"
    signatureVersion: "AWS4-HMAC-SHA256"
```

步骤四：执行备份恢复命令

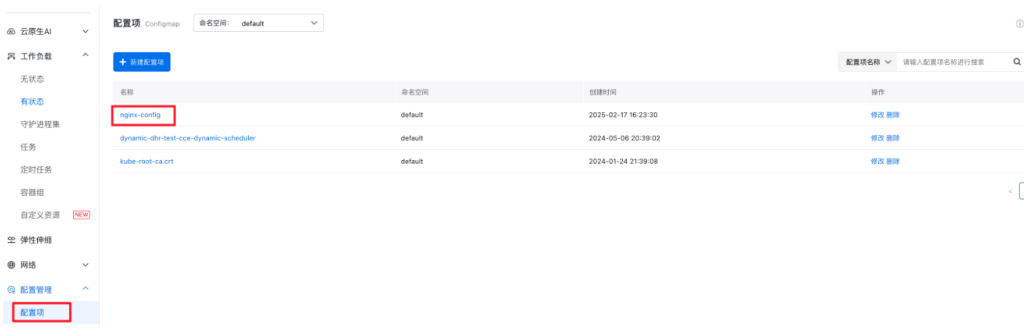
```
velero --kubeconfig {{restore-cluster-kubeconfig-path}} restore create --from-backup nginx-backup -n cce-backup
```

步骤五：确认应用正常运行，相关配置正确迁移

应用镜像：



配置项：



API参考（即将废弃）

概述

欢迎使用百度智能云的核心产品——百度智能云容器引擎 CCE（Cloud Container Engine）。您可以使用本文档介绍的 API 对 CCE 服务进行灵活的操作。

如果您是初次调用百度智能云产品的 API，可以观看 API 入门视频指南，快速掌握调用 API 的能力。

与新版 API 的区别

若您是使用旧版 API 或是 Web 页面中的『 创建集群（旧版） 』选项创建的集群，则应该继续使用旧版API进行操作。 若您是使用新版 API 或是 Web 页面中的『 创建集群 』选项创建的集群，则应该继续使用新版 API 进行操作。

两个版本的 API 互不兼容，旧版的 API 目前已经停止维护。 我们强烈建议您使用新版集群。

cluster 接口

接口类型	请求方式	描述
/v1/cluster	POST	创建一个 K8s 集群
/v1/cluster?status={status}	GET	返回对应状态的 K8s 集群信息列表
/v1/cluster/{clusterUuid}	GET	根据 clusterUuid 获取对应的 K8s 集群信息
/v1/cluster/{clusterUuid}	DELETE	根据 clusterUuid 删除一个 K8s 集群
/v1/cluster?scalingUp	POST	集群扩容（为集群添加节点）
/v1/cluster?scalingDown	POST	集群缩容（为集群删除节点）
/v1/node?clusterUuid={clusterUuid}	GET	集群所含节点列表

image 接口

接口	请求方式	说明
/v1/image/user	POST	创建镜像仓库用户
/v1/image/user	GET	获取镜像仓库用户
/v1/image/user	PUT	更新镜像仓库用户密码
/v1/image/namespace	POST	创建命名空间
/v1/image/namespace?user={user}	GET	根据用户名获取命名空间列表
/v1/image/namespace/{namespace}	GET	检查一个命名空间是否存在
/v1/image/namespace/{namespace}	DELETE	删除一个命名空间
/v1/image?user={user}	GET	根据用户名获取镜像列表
/v1/image/meta	POST	新增一个镜像元数据
/v1/image/meta	DELETE	删除一个镜像元数据
/v1/image/meta	GET	检查一个镜像元数据是否存在

通用说明

API认证机制

所有 API 的安全认证一律采用 Access Key 与请求签名机制。 Access Key 由 Access Key ID 和 Secret Access Key 组成，均为字符串。 对于每个 HTTP 请求，使用下面所描述的算法生成一个认证字符串，提交认证字符串放在 Authorization 头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version 是正整数。
- timestamp 是生成签名时的UTC时间。
- expirationPeriodInSeconds 表示签名有效期限。
- signedHeaders 是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如 `host;x-bce-date`。列表按照字典序排列。（本 API 签名仅使用 `host` 和 `x-bce-date` 两个 header）
- signature 是 256 位签名的十六进制表示，由 64 个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。鉴权认证机制的详细内容请参见鉴权认证机制。

密码加密传输规范定义

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用 AES 128 位加密算法进行加密，用 SK 的前 16 位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

- byte[] bCiphertext= AES(明文,SK)
- String strHex = HexStr(bCiphertext)

幂等性

当调用创建接口时如果遇到了请求超时或服务器内部错误，用户可能会尝试重发请求，这时用户通过 clientToken 参数避免创建出比预期要多的资源，即保证请求的幂等性。

幂等性基于 clientToken，clientToken 是一个长度不超过 64 位的 ASCII 字符串，通常放在query string 里，如

`http://cce.bj.baidubce.com/v1/instance?clientToken=be31b98c-5e41-4838-9830-9be700de5a20`

如果用户使用同一个 clientToken 值调用创建接口，则服务端会返回相同的请求结果。因此用户在遇到错误进行重试的时候，可以通过提供相同的 clientToken 值，来确保只创建一个资源；如果用户提供了一个已经使用过的 clientToken，但其他请求参数（包括 queryString 和 requestBody ）不同甚至url Path不同，则会返回 `IdempotentParameterMismatch` 的错误代码。

clientToken 的有效期为 24 小时，以服务端最后一次收到该 clientToken 为准。也就是说，如果客户端不断发送同一个 clientToken，那么该 clientToken 将长期有效。

排版约定

排版格式	含义
< >	变量
[ ]	可选项
{ }	必选项
|	互斥关系
等宽字体 Courier New	屏幕输出

请求参数

数据交换格式为 JSON，所有 request/response body 内容均采用 UTF-8 编码。

请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体，如 PUT /v1/cluster/{clusterUuid}
Query参数	URL 中携带的请求参数
HEADER	通过 HTTP 头域传入，如 x-bce-date
RequestBody	通过 JSON 格式组织的请求数据体

返回值说明

返回值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如 200, 400, 403, 404 等
ResponseBody	JSON 格式组织的响应数据体

API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL 参数	API 版本号，当前 API 版本 v1	是

公共头和错误返回

公共头规格

公共请求头

公共头部	描述
Authorization	包含 Access Key 与请求签名
Content-Type	application/json; charset=utf-8
x-bce-date	表示日期的字符串，符合 API 规范

HTTP 协议的标准头域不再这里列出。公共头域将在每个 API 中出现，是必需的头域。POST、PUT、DELETE 等请求数据放在 request body 中。

公共响应头

公共头部	描述
Content-Type	application/json; charset=utf-8
x-bce-request-id	后端服务生成，并自动设置到响应头域中

错误返回

当用户访问 API 出现错误时，会返回给用户相应的错误码和错误信息，便于用户定位问题，并做出适当的处理。请求发生错误时通过 Response Body 返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	错误码
message	String	错误描述
requestId	String	本次请求的 requestID

示例：

```
 {
   "code": "NoSuchObject",
   "message": "account not exist",
   "requestId": "c4abab14-ebc4-4eb0-85d2-03c4d51eed18"
}
```

BCE公共错误码

错误返回码	错误消息	状态码	说明
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源。
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate forthis operation.	400 Bad Request	请求中的 JSON 格式正确，但语义上不符合要求。如缺少某个必需项，或值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided doesnot exist in our records.	403 Forbidden	Access Key ID 不存在。
InvalidHTTPAuthHeader	The Access Key ID you provided does notexist in our records.	400 BadRequest	Authorization 头域格式错误。
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body 格式错误。例如不符合指定的 Encoding 等。
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI 形式不正确。例如一些服务定义的关键词不匹配等。对于 ID 不匹配的问题，应定义更加具体的错误码，如 NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 BadRequest	JSON 格式不合法。
InvalidVersion	The API version specified was invalid.	404 NotFound	URI 的版本号不合法。
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务。
PreconditionFailed	The specified If-Match header doesn’tmatch the ETag header.	412 PreconditionFailed	详见 Etag。
RequestExpired	Request has expired. Timestamp date is <Data>.	400 BadRequest	请求超时。要改成 x-bce-date。若请求中只有 Date，需将 Date 转成 datetime。
IdempotentParameterMismatch	The request uses the same client token asa previous, but non-identical request.	403 Forbidden	clientToken 对应的 API 参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check yourSecret Access Key and signing method. Consultthe service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致。

错误返回码

错误码	错误描述	HTTP状态码	中文解释
RegistryAlreadyExist	Namespace Already exist.	400 BadRequest	域名已存在
RegistryNoAuth	Authorization failed.	404 NotFound	auth 授权失败
RegistryInitFail	Init Registry failed.	404 NotFound	registry 初始化失败
RegistryUpdateFail	Update user failed.	404 NotFound	registry 更新失败
RegistryUserNotFound	Cannot found user.	404 NotFound	registry 所属用户不存在
RegistryLackParams	Bad request, lack param: user, namespace or public.	404 NotFound	缺少参数
RegistryOutOfLimit	Create failed, because number of namespaces out of limit.	404 NotFound	namespace 数超额
RegistryNamespaceNotFound	Cannot find Namespace.	404 NotFound	找不到 namespace
RegistryRepositoryNotFound	Cannot find Repository.	404 NotFound	registry 不存在
RegistryTagNotFound	Cannot find tag.	404 NotFound	tag 不存在
RegistryDelFail	Delete manifest from registry failed.	404 NotFound	删除 manifest 失败
RegistryInvalidParams	Invalid params to create image.	404 NotFound	registry 更新失败
RegistryNamespaceNotExist	Namespace is not exist.	404 NotFound	namespace 不存在
RegistryImageNotExist	Image is not exists.	404 NotFound	镜像不存在
InternalServerError	Unknown error, please try again later.	500 Internal Server	服务器内部错误
Cce.warning.UserNotExist	User not exist.	404 NotFound	用户名不存在
Cce.ClusterDeleted	Cluster is deleted already.	500 Internal Server	cluster 已删除
Cce.ClusterCreating	Cluster is creating.	500 Internal Server	集群正在创建中
Cce.ClusterDeleting	Cluster is deleting.	500 Internal Server	集群正在删除中
Cce.ClusterNotExit	Cluster is not exist.	500 Internal Server	集群不存在
DuplicateCLusterUuid	Generate cluster ID failed, please try again.	500 Internal Server	集群创建失败，请重试

Cce.DeleteIDToken	BlbId or clientToken is null.	500 Internal Server	BldID 或 clientToken为空
Cce.UpdateBlbId	BlbId is null.	500 Internal Server	BlbId 为空
Cce.CreateTokenNULL	ClientToken is null.	500 Internal Server	clientToken 为空
Cce.DeleteIdNULL	RouteRuled or clientToken is null.	500 Internal Server	RouteRuled或clientToken为空
Cce.RouteInfoIdNULL	RouteTableId or vpcId is null.	500 Internal Server	RouteTableId 或vpcId为空
Cce.ClusterNotExit	Cluster is not exist.	500 Internal Server	集群不存在
Cce.EncryptFail	Encrypt passwd wrong.	500 Internal Server	密码加密有误
Cce.DecryptFail	Decrypt passwd wrong.	500 Internal Server	密码解密有误
Cce.MoreThanOneUserPerAccount	One can create only one user per account.	500 Internal Server	账号需唯一
Cce.ImageMetaNull	ImageMeta is null.	500 Internal Server	镜像元数据为空
Cce.PasswdNotChange	New passwd should not as old.	500 Internal Server	新旧密码需不一致
Cce.UserExist	Username already exist.	500 Internal Server	用户名已存在
Cce.ConfirmFail	Passwd confirm fail.	500 Internal Server	密码确认失败
Cce.UserNoAuth	User authorization failed.	500 Internal Server	用户 auth 授权失败
Cce.NamespaceRepositoryOrTagNull	Namespace Repository or Tag is null.	500 Internal Server	镜像的仓库需非空
Cce.ImageTagFormatError	image tag cannot be empty, should be 1-65 characters long, and can only contain numbers, letters and '- _ /.' and can only start with letter or number	500 Internal Server	镜像格式有误败
Cce.ImageRepositoryFormatError	image repository cannot be empty, should be 1-65 characters long, and can only contain numbers, letters and '- _ /.' and can only start with letter	500 Internal Server	imageRepository 的格式有误
Cce.NamespaceNul	Namespace is null	500 Internal	namespace 为

l	Namespace is null.	Internal Server	空
Cce.UserNameNull	Username cannot be empty.	500 Internal Server	Username 为空
Cce.NamespaceRequestFormatError	Username or namespace cannot be empty,namespace should be 1-65 characters long, and can only contain numbers and letters and '- _ /.' and only can start with letters.	500 Internal Server	namespace请求格式有误
Cce.PasswordFormatError	Password format error or password conformation not same as password.	500 Internal Server	密码格式有误
Cce.UserNameFormatError	Username format error,your username must be between 4 and 30 characters long, and can only contain numbers and lowercase letters.	500 Internal Server	用户名格式有误
Cce.WrongMainZone	Main zone in order is illegal.	500 Internal Server	主 zone 非法
Cce.namespaceOverQuota	Over the namespace quota.	500 Internal Server	域名配额超限
Cce.slaveOverQuota	Over the slave quota.	500 Internal Server	slave 配额超限
Cce.ClusterOverQuota	Over the cluster quota.	500 Internal Server	集群配额超限
Cce.ContainerNetConflict	Container net conflict with vpc or other container nets in the same vpc.	500 Internal Server	容器网络有冲突
Cce.IllegalArguments	Illegal arguments, check your input.	500 Internal Server	非法参数
Cce.UnsupportedPayType	Unsupported payment method.	500 Internal Server	支付方式不支持
Cce.OrderError	Cluster order creation failed.	500 Internal Server	订单有误

API_V2参考

概述

欢迎使用百度智能云的核心产品——百度智能云容器引擎服务 CCE（Cloud Container Engine）。您可以使用本文档介绍的 API 对 CCE 服务进行灵活的操作。

如果您是初次调用百度智能云产品的API，可以观看 API 入门视频指南，快速掌握调用API的能力。

与旧版API的区别

若您是使用旧版 API 或是 Web 页面中的『 创建集群（旧版） 』选项创建的集群，则应该继续使用旧版API进行操作。 若您是使用新版 API 或是 Web 页面中的『 创建集群 』选项创建的集群，则应该继续使用新版 API 进行操作。

两个版本的 API 互不兼容，旧版的 API 目前已经停止维护。 我们强烈建议您使用新版集群。

Cluster 接口

接口	请求方式	描述
/v2/cluster	POST	创建 CCE K8S 集群
/v2/clusters	GET	查询 CCE K8S 集群列表
/v2/cluster/{clusterID}	GET	查询指定集群详情
/v2/cluster/{clusterID}	DELETE	删除指定集群

InstanceGroup 接口

接口	请求方式	说明
/v2/cluster/{clusterID}/instancegroup	POST	创建节点组
/v2/cluster/{clusterID}/instancegroups	GET	获取集群的节点组列表
/v2/cluster/{clusterID}/instancegroup/{instanceGroupID}	GET	获取节点组详情
/v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/replicas	PUT	修改节点组节点副本数
/v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/autoscaler	PUT	修改节点组节点自动扩缩容配置
/v2/cluster/{clusterID}/instancegroup/{instanceGroupID}	DELETE	删除节点组

Instance 接口

接口	请求方式	说明
/v2/cluster/{clusterID}/instances	POST	创建节点（集群扩容）
/v2/cluster/{clusterID}/instance/{instanceID}	GET	获取节点详情
/v2/cluster/{clusterID}/instances	GET	获取集群节点列表
/v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/instances	GET	获取节点组节点列表
/v2/cluster/{clusterID}/instance/{instanceID}	PUT	更新节点属性
/v2/cluster/{clusterID}/instances	PUT	删除节点（集群缩容）

Network 接口

接口	请求方式	说明
/v2/net/check_clusterip_cidr	POST	检查集群 ClusterIP 网段
/v2/net/check_container_network_cidr	POST	检查集群容器网络网段
/v2/net/recommend_clusterip_cidr	POST	推荐集群 ClusterIP 网段
/v2/net/recommend_container_cidr	POST	推荐集群容器网络网段

Autoscaler 接口

接口	请求方式	说明
/v2/autoscaler/{clusterID}	POST	创建集群 Autoscaler
/v2/autoscaler/{clusterID}	GET	查询集群 Autoscaler 配置
/v2/autoscaler/{clusterID}	PUT	更新集群 Autoscaler 配置

Kubeconfig 接口

接口	请求方式	说明
/v2/kubeconfig/{clusterID}/admin/{type}	GET	查询集群 Admin KubeConfig

通用说明

API调用遵循HTTP协议，各Region采用不同的域名，具体域名为cce.{region}.baidubce.com。 例：cce.bj.baidubce.com。数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

实名认证

使用容器引擎CCE API的用户需要实名认证，没有通过实名认证的可以前往百度开放云官网控制台中的安全认证下的实名认证中进行认证。

API认证机制

所有 API 的安全认证一律采用 Access Key 与请求签名机制。 Access Key 由 Access Key ID 和 Secret Access Key 组成，均为字符串。 对于每个 HTTP 请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在 Authorization 头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version 是正整数。

- timestamp 是生成签名时的 UTC 时间。
- expirationPeriodInSeconds 表示签名有效期限。
- signedHeaders 是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如 host;x-bce-date。列表按照字典序排列。（本 API 签名仅使用 host 和 x-bce-date 两个 header）
- signature 是 256 位签名的十六进制表示，由 64 个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的 SK 和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行对比。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。鉴权认证机制的详细内容请参见鉴权认证机制。

通信协议

CCE API支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。
请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体,如:POST /v{version}/instance/{instanceId}
Query参数	URL中携带的请求参数
HEADER	通过HTTP头域传入,如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

响应结构说明

响应值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL 参数	API 版本号，当前 API 版本 v2	是

密码加密传输规范定义

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用 AES 128 位加密算法进行加密，用 SK 的前 16 位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

- byte[] bCiphertext= AES(明文,SK)
- String strHex = HexStr(bCiphertext)

排版约定

排版格式	含义
< >	变量
[ ]	可选项
{ }	必选项
|	互斥关系
等宽字体 Courier New	屏幕输出

服务域名

API 的服务域名为：

Region	Endpoint	Protocol
北京	cce.bj.baidubce.com	HTTP and HTTPS
广州	cce.gz.baidubce.com	HTTP and HTTPS
苏州	cce.su.baidubce.com	HTTP and HTTPS
保定	cce.bd.baidubce.com	HTTP and HTTPS
武汉	cce.fwh.baidubce.com	HTTP and HTTPS
香港	cce.hkg.baidubce.com	HTTP and HTTPS
阳泉	cce.yq.baidubce.com	HTTP and HTTPS
成都	cce.cd.baidubce.com	HTTP and HTTPS
南京	cce.nj.baidubce.com	HTTP and HTTPS

说明： API 支持 HTTP 和 HTTPS 两种调用方式。为了提升数据的安全性，建议通过 HTTPS 调用。

公共头和错误返回

公共头规格

公共请求头

公共头部	描述
Authorization	包含 Access Key 与请求签名
Content-Type	application/json; charset=utf-8
x-bce-date	表示时间的字符串，符合 API 规范

HTTP 协议的标准头域不再这里列出。公共头域将在每个 API 中出现，是必需的头域。接口请求参数可能在path, query 和body 中，以具体接口说明为准。

公共响应头

公共头部	描述
Content-Type	application/json; charset=utf-8
X-Bce-Request-Id	服务端生成，并自动设置到响应头域中

错误返回

当用户访问 API 出现错误时，会返回给用户相应的错误码和错误信息，便于用户定位问题，并做出适当的处理。请求发生错误时通过 Response Body 返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	错误码
message	String	错误描述
requestId	String	本次请求的 requestID

示例：

```
 {
   "code":"cce.warning.ClusterNotFound",
   "message":"ClusterID [xxx] belongs to account [xxx] not exists",
   "requestId": "c4abab14-ebc4-4eb0-85d2-03c4d51eed18"
}
```

CCE 集群管理 错误返回码

当错误码前缀为 **cce.error.** 时用户一般需要提交工单进行处理，当错误码前缀为 **cce.warning.** 或 **cce.info.** 时用户可根据错误信息自行处理。

错误码	错误描述	HTTP状态码	中文解释
cce.error.InternalError	We encountered an internal error Please try again.	500 Internal Server Error	未定义的服务器内部错误。
cce.warning.AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源。
cce.warning.MalformedJSON	The JSON you provided was not well-formed.	400 BadRequest	JSON 格式不合法。
cce.warning.InvalidParam	Some params are invalid.	400 BadRequest	输入参数不合法。
cce.warning.NoSuchObject	Target object not found.	404 NotFound	找不到目标资源。
cce.warning.ClusterNotFound	Cluster not found	404 NotFound	无法找到该集群。
cce.warning.ClusterNotExposedToPublic	Cluster not exposed to public	404 NotFound	集群未开启公网访问。
cce.warning.InstanceAlreadyExists	Instance already exists	400 Bad Request	节点已经存在。
cce.warning.ExistedInstanceIDDuplicated	Existed instance ID duplicated	400 Bad Request	与已有的 InstanceID 重复。
cce.warning.InstanceNotFound	Instance not Found	404 NotFound	无法找到该节点。
cce.warning.VPCIDNotExists	VPC ID not exists	404 NotFound	VPC ID 不存在。
cce.warning.ClusterHAInvalid	ClusterHA invalid	400 Bad Request	集群的 Master 节点数不合法。
cce.warning.InstanceGroupNotFound	Instance group not found	404 NotFound	无法找到该节点组。
cce.info.RegionNotAvailable	Region not available	412 Status Precondition Failed	指定的地域不可用。
cce.info.BLBNotAvailable	BLB not available. You must have at least one bcc, dcc or bbc to create BLB for cluster.	412 Status Precondition Failed	BLB 不可用。VPC 内至少存在一台 BCC DCC 或 BBC 才能建 BLB。

Cluster相关接口

创建集群

描述

创建 CCE K8S 集群

请求结构

```
POST /v2/cluster HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
cluster	ClusterSpec	是	RequestBody 参数	集群配置
masters	List<InstanceSet>	否	RequestBody 参数	集群 Master 配置，支持异构组合. 当集群类型为托管型时无需设置该参数
nodes	List<InstanceSet>	否	RequestBody 参数	集群 Worker 配置，支持异构组合. 不设置此参数时将会创建仅有master的集群
options	CreateClusterOptions	是	RequestBody 参数	集群创建选项. 用户可以设置强制跳过网段冲突检查

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
clusterID	String	是	集群 ID
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例: 托管 Master 集群

- 设置 `cluster.masterConfig` 的 `masterType` 设为 `managed`
- 无需设置 `masters` 参数
- 其它参数参考 API 文档按需设置

```
 POST /v2/cluster  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "cluster":{
        "clusterName":"test-open-types",
        "k8sVersion":"1.16.8",
        "runtimeType":"docker",
        "vpcID":"vpc-mwbgygrjb72w",
        "masterConfig":{
            "masterType":"managed",
            "clusterHA":1,
            "exposedPublic":false,
            "clusterBLBVPCSubnetID":"sbn-mnbvhnuupv1u",
            "managedClusterMasterOption":{
                "masterVPCSubnetZone":"zoneA"
            }
        },
  "containerNetworkConfig": {
   "mode": "kubenet",
   "lbServiceVPCSubnetID": "sbn-mnbvhnuupv1u",
   "clusterPodCIDR": "172.28.0.0/16",
   "clusterIPServiceCIDR": "172.31.0.0/16"
  },
  "k8sCustomConfig": {
   "kubeAPIQPS": 1000,
   "kubeAPIBurst": 2000
  }
    },
    "nodes":[
        {
            "instanceSpec":{
                "instanceName":"instance-name",
                "clusterRole":"node",
                "existed":false,
                "machineType":"BCC",
                "instanceType":"N3",
                "vpcConfig":{
                    "vpcID":"vpc-mwbgygrjb72w",
                    "vpcSubnetID":"sbn-mnbvhnuupv1u",
                    "availableZone":"zoneA",
                    "securityGroup": {
                      "customSecurityGroups": [],
                      "enableCCERequiredSecurityGroup": true,
                      "enableCCEOptionalSecurityGroup": true
                    }
                },
                "instanceResource":{
                    "cpu":4,
                    "mem":8,
                    "rootDiskSize":40,
                    "localDiskSize":0,
                    "cdsList":[]
                }
```

```
            },
            "imageID":"m-gTpZ1k6n",
            "instanceOS":{
                "imageType":"System",
                "osType":"linux",
                "osName":"CentOS",
                "osVersion":"7.5",
                "osArch":"x86_64 (64bit)"
            },
            "needEIP":false,
            "adminPassword":"test123!T",
            "instanceChargingType":"Postpaid",
            "runtimeType":"docker"
        },
        "count":1
    }
  ]
}
```

请求示例: 自定义 Master 集群

- 设置 `cluster.masterConfig` 的 `masterType` 设为 `custom`
- 设置 `masters` 参数
- 其它参数参考 API 文档按需设置

```
{
  "cluster":{
    "clusterName":"create-custom-cluster-reg",
    "description":"集群描述",
    "k8sVersion":"1.16.8",
    "runtimeType":"docker",
    "vpcID":"vpc-43zsdm46t9rp",
    "masterConfig":{
      "masterType":"custom",
      "exposedPublic":true,
      "clusterBLBVPCSubnetID":"sbn-vvqsb9b57f24"
    },
    "containerNetworkConfig":{
      "mode":"kubenet",
      "lbServiceVPCSubnetID":"sbn-vvqsb9b57f24",
      "nodePortRangeMin":50000,
      "nodePortRangeMax":51000,
      "clusterPodCIDR":"10.2.0.0/16",
      "clusterIPServiceCIDR":"172.16.0.0/16",
      "maxPodsPerNode":64,
      "kubeProxyMode":"ipvs"
    }
  },
  "masters":[
    {
      "instanceSpec":{
        "machineType":"BCC",
        "instanceType":"N3",
        "vpcConfig":{
          "vpcSubnetID":"sbn-vvqsb9b57f24",
          "securityGroup": {
            "customSecurityGroups": [],
            "enableCCERequiredSecurityGroup": true,
            "enableCCEOptionalSecurityGroup": true
          }
        },
        "instanceResource":{
          "CPU":4,
          "MEM":8
        },
        "instanceOS": {
          "imageName": "7.5 x86_64 (64bit)",
          "imageType": "System",
          "osType": "linux",
```

```
                "osName": "CentOS",
                "osVersion": "7.5",
                "osArch": "x86_64 (64bit)"
            },
            "adminPassword":"test123!T"
        },
        "count":1
      }
    ],
    "nodes":[
      {
        "instanceSpec":{
            "machineType":"BCC",
            "instanceType":"N3",
            "vpcConfig":{
                "vpcSubnetID":"sbn-vvqsb9b57f24",
                "securityGroupID":"g-k4tsm0id2g1n"
            },
            "instanceResource":{
                "CPU":4,
                "MEM":8,
                "cdsList":[
                  {
                    "diskPath":"/data",
                    "storageType":"cloud_hp1",
                    "cdsSize":200
                  }
                ]
            },
            "instanceOS": {
                "imageName": "7.5 x86_64 (64bit)",
                "imageType": "System",
                "osType": "linux",
                "osName": "CentOS",
                "osVersion": "7.5",
                "osArch": "x86_64 (64bit)"
            },
            "adminPassword":"test123!T"
        },
        "count":1
      }
    ]
}
```

请求示例: 已有实例

- 如果 Master 需要使用已有实例，设置 `cluster.masterConfig` 的 `masterType` 设为 `custom`
- 添加 Master 或 Node 机器配置时，设置 `instanceSpec.existed` 为 `true` 并设置 `instanceSpec.existedOption.existedInstanceID` 为希望使用的已有节点 ID
- 如果不希望重装系统，设置 `instanceSpec.existedOption.rebuild` 为 `false` 并务必保证机器密码正确，否则节点会因无法部署相关服务而创建失败
- 如果不希望重装系统，无需设置 `instanceSpec.instanceOS` 与 `instanceSpec.machineType`
- 其它参数参考 API 文档按需设置

```
{
  "cluster":{
     "clusterName":"create-existed-bcc-cluster-reg",
     "description":"集群描述",
     "k8sVersion":"1.16.8",
     "runtimeType":"docker",
     "vpcID":"vpc-43zsdm46t9rp",
     "masterConfig":{
        "masterType":"custom",
        "exposedPublic":true,
        "clusterBLBVPCSubnetID":"sbn-vvqsb9b57f24"
     },
     "containerNetworkConfig":{
        "mode":"kubenet",
        "lbServiceVPCSubnetID":"sbn-vvqsb9b57f24",
        "nodePortRangeMin":30000,
        "nodePortRangeMax":32768,
        "clusterPodCIDR":"10.0.0.0/16",
        "clusterIPServiceCIDR":"172.16.0.0/16",
        "maxPodsPerNode":256,
        "kubeProxyMode":"ipvs"
     }
  },
  "masters":[
     {
        "instanceSpec":{
           "existed":true,
           "existedOption":{
              "existedInstanceID":"i-SxeBLkcN",
              "rebuild":true
           },
           "machineType":"BCC",
           "instanceOS": {
              "imageName": "7.5 x86_64 (64bit)",
              "imageType": "System",
              "osType": "linux",
              "osName": "CentOS",
              "osVersion": "7.5",
              "osArch": "x86_64 (64bit)"
           },
           "adminPassword":"test123!T"
        }
     }
  ],
  "nodes":[
     {
        "instanceSpec":{
           "existed":true,
           "existedOption":{
              "existedInstanceID":"i-M56Un1DO",
              "rebuild":true
           },
           "machineType":"BCC",
           "instanceOS": {
              "imageName": "7.5 x86_64 (64bit)",
              "imageType": "System",
              "osType": "linux",
              "osName": "CentOS",
              "osVersion": "7.5",
              "osArch": "x86_64 (64bit)"
           },
           "adminPassword":"test123!T"
        }
     }
  ]
}
```

请求示例：挂载CDS

有时我们希望在**新建节点**的同时为节点挂载 1 到多个 CDS，此时在创建集群时，为节点设置 CDS 参数和相关挂载路径即可。示例如下。

需要注意的是，每个路径下只能挂载一个 CDS，但一个路径的子路径下可以挂载另一个 CDS，例如 /a 目录下仅可挂载一个 CDS，但 /a/b 目录下可以挂载另一个 CDS。
已有节点仅可挂载 CDS 到指定路径，不会新建 CDS。因此当已有节点试图挂载 CDS 到指定 Path 但找不到满足条件的 CDS 时，将会忽略相关 CDS 配置。

```
{
  ......

  "masters":[
    {
      "instanceSpec":{
        ......
        "instanceResource": {
          "cdsList": [
            {diskPath: "/home/cce", storageType: "cloud_hp1", cdsSize: 50}
          ]
        },
        ......
      }
    }
  ],
  "nodes":[
    {
      "instanceSpec":{
        ......
        "instanceResource": {
          "cdsList": [
            {diskPath: "/home/cce", storageType: "cloud_hp1", cdsSize: 60}
          ]
        },
        ......
      }
    }
  ]
}
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: d2ce8f50-529a-4663-9265-ad08c94633c8
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
{
   "clusterID": "cce-NqYwWEhu",
   "requestID": "d2ce8f50-529a-4663-9265-ad08c94633c8"
}
```

集群列表

描述

查询用户 CCE K8S 集群列表

请求结构

```
 GET /v2/clusters  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
keywordType	String	否	Query 参数	集群模糊查询字段，可选 [clusterName, clusterID]，默认值为 clusterName
keyword	String	否	Query 参数	查询关键词，默认值为空字符串
orderBy	String	否	Query 参数	集群查询排序字段，可选 [clusterName, clusterID, createdAt]，默认值为 clusterName
order	String	否	Query 参数	排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC
pageNo	Integer	否	Query 参数	页码，默认值为1
pageSize	Integer	否	Query 参数	单页结果数，默认值为10

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述	描述
clusterPage	ClusterPage	是	集群分页查询返回结果
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /v2/clusters?keywordType=clusterName&keyword=&orderBy=clusterID&order=ASC&pageNo=1&pageSize=10  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 97342dc7-29a1-4ed9-a75d-904bb293d295
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "clusterPage": {
  "keywordType": "clusterName",
  "keyword": "",
  "orderBy": "clusterID",
  "order": "ASC",
  "pageNo": 1,
  "pageSize": 10,
  "totalCount": 1,
  "clusterList": [
   {
    "spec": {
     "clusterID": "cce-shpdaa9l",
     "clusterName": "sdk-ccev2-test",
     "clusterType": "normal",
     "description": "",
     "k8sVersion": "1.16.8",
     "vpcID": "vpc-aj2rcjm084y5",
     "vpcCIDR": "192.168.0.0/16",
     "plugins": [
      "ip-masq-agent",
      "core-dns",
      "kube-proxy",
      "metrics-server",
      "nvidia-gpu"
     ],
     "masterConfig": {
      "masterType": "managed",
      "clusterHA": 1,
      "clusterBLBVPCSubnetID": "sbn-0dizryuc81c0",
      "managedClusterMasterOption": {
       "masterVPCSubnetZone": "zoneA"
      }
     },
     "containerNetworkConfig": {
      "mode": "kubenet",
      "ipVersion": "ipv4",
      "lbServiceVPCSubnetID": "sbn-0dizryuc81c0",
      "nodePortRangeMin": 30000,
      "nodePortRangeMax": 32767,
      "clusterPodCIDR": "172.28.0.0/16",
      "clusterIPServiceCIDR": "172.31.0.0/16",
      "maxPodsPerNode": 128,
      "kubeProxyMode": "ipvs"
     }
    },
    "status": {
     "clusterBLB": {
      "id": "lb-1454d5c4",
      "vpcIP": "100.64.230.44",
      "eip": ""
     },
     "clusterPhase": "running",
     "nodeNum": 0
    },
    "createdAt": "2020-09-04T01:58:12Z",
    "updatedAt": "2020-09-04T02:00:37Z"
   }
  ]
 },
 "requestID": "97342dc7-29a1-4ed9-a75d-904bb293d295"
}
```

集群详情

描述

查询指定集群详情

请求结构

```
 GET /v2/cluster/{clusterID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
cluster	Cluster	是	集群详情查询结果
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /v2/cluster/cce-zyt88sqy  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 928a21b5-d117-4a83-a274-fd7d6f413524
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "cluster": {
  "spec": {
   "clusterID": "cce-shpdaa9l",
   "clusterName": "sdk-ccev2-test",
   "clusterType": "normal",
   "description": "",
   "k8sVersion": "1.16.8",
   "vpcID": "vpc-aj2rcjm084y5",
   "vpcCIDR": "192.168.0.0/16",
   "plugins": [
    "ip-masq-agent",
    "core-dns",
    "kube-proxy",
    "metrics-server",
    "nvidia-gpu"
   ],
   "masterConfig": {
    "masterType": "managed",
    "clusterHA": 1,
    "clusterBLBVPCSubnetID": "sbn-0dizryuc81c0",
    "managedClusterMasterOption": {
     "masterVPCSubnetZone": "zoneA"
    }
   },
   "containerNetworkConfig": {
    "mode": "kubenet",
    "ipVersion": "ipv4",
    "lbServiceVPCSubnetID": "sbn-0dizryuc81c0",
    "nodePortRangeMin": 30000,
    "nodePortRangeMax": 32767,
    "clusterPodCIDR": "172.28.0.0/16",
    "clusterIPServiceCIDR": "172.31.0.0/16",
    "maxPodsPerNode": 128,
    "kubeProxyMode": "ipvs"
   }
  },
  "status": {
   "clusterBLB": {
    "id": "lb-1454d5c4",
    "vpcIP": "100.64.230.44",
    "eip": ""
   },
   "clusterPhase": "running",
   "nodeNum": 0
  },
  "createdAt": "2020-09-04T01:58:12Z",
  "updatedAt": "2020-09-04T02:00:37Z"
 },
 "requestID": "928a21b5-d117-4a83-a274-fd7d6f413524"
}
```

删除集群

描述

删除指定集群

请求结构

```
 DELETE /v2/cluster/{clusterID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
deleteResource	Boolean	否	Query 参数	是否删除相关资源（后付费公网IP和云磁盘），默认值为 false
deleteCDSSnapshot	Boolean	否	Query 参数	是否删除云磁盘快照，默认值为 false
moveOut	Boolean	否	Query 参数	集群删除是否保留节点，默认值为 false

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 DELETE /v2/cluster/cce-uqc4lju1?deleteResource=true&deleteCDSSnapshot=true  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 105ce04b-1a42-4f77-9d22-ab6f413f9d69
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "requestID": "105ce04b-1a42-4f77-9d22-ab6f413f9d69"
}
```

更新集群删除保护开关

描述

更新集群删除保护开关

请求结构

```
 PUT /v2/cluster/{clusterID}/forbiddelete HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
forbidDelete	Boolean	是	RequestBody 参数	是否打开或关闭集群删除保护，true为打开，false为关闭

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
success	Boolean	是	请求是否成功
forbidDelete	Boolean	是	集群删除保护状态，true为打开，false为关闭

请求示例

```
 PUT v2/cluster/cce-5gw06fjs/forbiddelete HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "forbidDelete": false
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 3d61a898-b804-4d6f-8c7c-b741d2ffbe45
Date: Wed, 11 Dec 2024 02:59:18 GMT
Content-Type: application/json;charset=UTF-8

{
    "success": true,
    "forbidDelete": true
}
```

获取集群事件步骤

描述

获取创建或删除过程中集群所处的事件步骤。

请求结构

```
 GET /v2/event/cluster/{clusterID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
clusterID	String	是	URL 参数	集群 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必选	描述
status	String	是	事件类型
steps	List<Step>	是	集群操作步骤
requestID	String	是	请求ID

请求示例

```
 GET v2/event/cluster/cce-j8pb3dm0  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 08eb305a-87fb-4360-8a27-42b9561edf4a
Date: Fri, 12 Aug 2022 03:10:30 GMT
Content-Type: application/json;charset=UTF-8

{
    "status": "created",
    "steps": [
        {
            "stepName": "创建基础证书"
```

```
            "stepName": "创建基础组件",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:23:07Z",
            "finishedTime": "2022-07-12T09:23:07Z",
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "创建 BLB",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:23:07Z",
            "finishedTime": "2022-07-12T09:23:16Z",
            "costSeconds": 9,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "创建 EIP",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:23:16Z",
            "finishedTime": "2022-07-12T09:23:25Z",
            "costSeconds": 9,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "创建 Master",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:23:25Z",
            "finishedTime": "2022-07-12T09:23:54Z",
            "costSeconds": 29,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "连通 APIServer",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:23:54Z",
            "finishedTime": "2022-07-12T09:25:52Z",
            "costSeconds": 118,
            "retryCount": 2,
            "errInfo": {}
        },
        {
            "stepName": "部署 K8S 插件",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-07-12T09:25:53Z",
            "finishedTime": "2022-07-12T09:26:34Z",
            "costSeconds": 41,
            "retryCount": 1,
            "errInfo": {}
        }
    ],
    "requestID": "562c76ee-c65f-4673-9574-89222f1cd210"
}
```


Autoscaler相关接口

查询Autoscaler配置

描述

查询集群 Autoscaler 配置

请求结构

```
 GET /v2/autoscaler/{clusterID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
autoscaler	Autoscaler	是	查询到的 Autoscaler 信息
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /v2/autoscaler/cce-uqc4lju1 HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6836119a-99e1-4448-b469-1199c3fb1b07
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "autoscaler": {
        "clusterID": "cce-uqc4lju1",
        "clusterName": "sdk-ccev2-test2",
        "caConfig": {
            "replicaCount": 2,
            "scaleDownEnabled": true,
            "scaleDownUtilizationThreshold": 50,
            "scaleDownGPUUtilizationThreshold": 50,
            "scaleDownUnneededTime": 10,
            "scaleDownDelayAfterAdd": 10,
            "maxEmptyBulkDelete": 10,
            "skipNodesWithLocalStorage": true,
            "skipNodesWithSystemPods": false,
            "expander": "random",
            "customConfigs": {
                "--node-deletion-delay-timeout": "3m"
            }
        }
    },
    "requestID": "41d2b661-fe2f-4454-a017-5060795a0ac6"
}
```

创建Autoscaler

描述

创建集群 Autoscaler

请求结构

```
 POST /v2/autoscaler/{clusterID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/autoscaler/cce-uqc4lju1 HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: b339ba19-1a26-49f5-9cec-74b5a7a080c6
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "b339ba19-1a26-49f5-9cec-74b5a7a080c6"
}
```

更新 Autoscaler 配置

描述

更新集群 Autoscaler 配置

请求结构

```
 PUT /v2/autoscaler/{clusterID}  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
expander	String	是	RequestBody 参数	自动扩缩容选组的策略. 可选 [random, most-pods, least-waste, priority], 默认值为 random.
instanceGroups	List<ClusterAutoscalerInstanceGroup>	否	RequestBody 参数	节点组的 Autoscaler 配置. 用户无需输入此项内容.
kubeVersion	String	否	RequestBody 参数	K8S 版本. 为空时后台会自动查询集群K8S版本号.
maxEmptyBulkDelete	Integer	否	RequestBody 参数	最大并发缩容数
scaleDownDelayAfterAdd	Integer	否	RequestBody 参数	扩容后缩容启动时延, 单位为分钟
scaleDownEnabled	Boolean	否	RequestBody 参数	是否启动缩容. 默认值为false
scaleDownGPUUtilizationThreshold	Integer	否	RequestBody 参数	GPU缩容阈值百分比, 取值范围（0, 100）.
scaleDownUnneededTime	Integer	否	RequestBody 参数	缩容触发时延，单位为分钟.
scaleDownUtilizationThreshold	Integer	否	RequestBody 参数	缩容阈值百分比, 取值范围（0, 100）.
skipNodesWithLocalStorage	Boolean	否	RequestBody 参数	是否跳过使用本地存储的节点, 默认值为 true.
skipNodesWithSystemPods	Boolean	否	RequestBody 参数	是否跳过有部署系统 Pod 的节点, 默认值为 true.
customConfigs	Map<String,String>	否	RequestBody 参数	用户自定义配置.

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 PUT /v2/autoscaler/cce-br0i4kl5 HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "scaleDownEnabled": true,
    "scaleDownUtilizationThreshold": 50,
    "scaleDownGPUUtilizationThreshold": 50,
    "scaleDownUnneededTime": 10,
    "scaleDownDelayAfterAdd": 10,
    "maxEmptyBulkDelete": 10,
    "skipNodesWithLocalStorage": true,
    "skipNodesWithSystemPods": false,
    "expander": "random",
    "customConfigs": {
        "--node-deletion-delay-timeout": "3m"
    }
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 3d61a898-b804-4d6f-8c7c-b741d2ffbe45
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "3d61a898-b804-4d6f-8c7c-b741d2ffbe45"
}
```

InstanceGroup相关接口

创建节点组

描述

创建节点组

请求结构

```
 POST /v2/cluster/{clusterID}/instancegroup HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupName	String	是	Request Body 参数	节点组名称，不可为空
clusterRole	String	否	Request Body 参数	节点在集群中的角色. 目前仅支持Node类型阶段组, 默认值为node
shrinkPolicy	String	否	Request Body 参数	节点组收缩规则. 可选 [Priority, Uniform]. 默认为 Priority. Priority 优先收缩掉节点优先值低的节点, Uniform 多子网平均缩容
updatePolicy	String	否	Request Body 参数	节点组更新规则. 可选 [Rolling, Concurrency]. 默认为 Concurrency. Concurrency 并发更新, Rolling 滚动更新. 该参数暂未启用
cleanPolicy	String	否	Request Body 参数	节点清理规则. 可选 [Remain, Delete]. 默认为 Delete.
instanceTemplate	InstanceTemplate	是	Request Body 参数	节点组的节点配置
replicas	Integer	是	Request Body 参数	节点组节点要求的副本数. 取值范围是自然数集
clusterAutoscalerSpec	ClusterAutoscalerSpec	否	Request Body 参数	集群自动伸缩配置

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
instanceGroupID	String	是	节点组 ID
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/cluster/cce-f7zeyx1u/instancegroup  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "instanceGroupName": "sdk-testcase",
 "selector": null,
 "cleanPolicy": "Delete",
 "instanceTemplate": {
  "instanceName": "",
  "runtimeType": "docker",
  "clusterRole": "node",
  "existedOption": {},
  "machineType": "BCC",
  "instanceType": "N3",
  "vpcConfig": {
   "vpcID": "vpc-pi9fghaxcpnf",
   "vpcSubnetID": "sbn-ww1xf6a5fi88",
   "securityGroupID": "g-4mnvpnrfscm1",
   "availableZone": "zoneA"
  },
  "instanceResource": {
   "cpu": 1,
   "mem": 4,
   "rootDiskSize": 40
  },
  "imageID": "305eeb25-a693-4cc0-832a-57bccf855771",
  "instanceOS": {
   "imageType": "System"
  },
  "instanceChargingType": "Postpaid",
  "instancePreChargingOption": {},
  "deployCustomConfig": {
   "dockerConfig": {},
   "preUserScript": "ls",
   "postUserScript": "ls"
  }
 },
 "replicas": 3
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: aef503ab-66e2-4b7f-9044-e922389ed03f
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "aef503ab-66e2-4b7f-9044-e922389ed03f",
 "instanceGroupID": "cce-ig-dvej1d3y"
}
```

获取节点组详情

描述

获取节点组详情

请求结构

```
 GET /v2/cluster/{clusterID}/instancegroup/{instanceGroupID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID
instanceGroup	InstanceGroup	是	查询到的节点组详情

请求示例

```
 GET /v2/cluster/cce-47bqnhmj/instancegroup/cce-ig-796lmt7a  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 9ccdfbcf-f989-49e6-9701-6996dee804b1
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "9ccdfbcf-f989-49e6-9701-6996dee804b1",
 "instanceGroup": {
  "spec": {
   "cceInstanceGroupID": "cce-ig-dvej1d3y",
   "instanceGroupName": "sdk-testcase",
   "clusterID": "cce-z6qjgcq7",
   "clusterRole": "node",
   "shrinkPolicy": "Priority",
   "updatePolicy": "Concurrency",
   "cleanPolicy": "Delete",
   "instanceTemplate": {
    "instanceName": "",
    "runtimeType": "docker",
    "runtimeVersion": "18.9.2",
    "clusterID": "cce-z6qjgcq7",
    "clusterRole": "node",
    "instanceGroupID": "cce-ig-dvej1d3y",
    "instanceGroupName": "sdk-testcase",
    "existedOption": {},
    "machineType": "BCC",
    "instanceType": "N3",
    "bbcOption": {},
    "vpcConfig": {
     "vpcID": "vpc-pi9fghaxcpnf",
     "vpcSubnetID": "sbn-ww1xf6a5fi88",
     "securityGroupID": "g-4mnvpnrfscm1",
     "vpcSubnetType": "BCC",
     "vpcSubnetCIDR": "192.168.16.0/24",
     "availableZone": "zoneA"
    },
    "instanceResource": {
     "cpu": 1,
     "mem": 4,
     "rootDiskType": "hp1",
     "rootDiskSize": 40
    },
    "imageID": "m-4Umtt2i5",
    "instanceOS": {
     "imageType": "System",
     "imageName": "centos-8u0-x86_64-20200601205040",
     "osType": "linux"
```

```
    "osType": "linux",
    "osName": "CentOS",
    "osVersion": "8.0",
    "osArch": "x86_64 (64bit)",
    "osBuild": "2020060100"
   },
   "eipOption": {},
   "instanceChargingType": "Postpaid",
   "instancePreChargingOption": {},
   "deleteOption": {
    "deleteResource": true,
    "deleteCDSSnapshot": true
   },
   "deployCustomConfig": {
    "dockerConfig": {},
    "preUserScript": "bHM=",
    "postUserScript": "bHM="
   },
   "labels": {
    "cluster-id": "cce-z6qjgcq7",
    "cluster-role": "node",
    "instance-group-id": "cce-ig-dvej1d3y"
   },
   "cceInstancePriority": 5
  },
  "replicas": 3,
  "clusterAutoscalerSpec": {
   "enabled": false,
   "minReplicas": 0,
   "maxReplicas": 0,
   "scalingGroupPriority": 0
  }
 },
 "status": {
  "readyReplicas": 3,
  "pause": {
   "paused": false,
   "reason": ""
  }
 },
 "createdAt": "2020-09-27T06:34:51Z"
}
}
```


删除节点组

描述

删除节点组

请求结构

```
 DELETE /v2/cluster/{clusterID}/instancegroup/{instanceGroupID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
deleteInstances	Boolean	否	Query 参数	是否删除节点组内节点. 默认为false
releaseAllResource	Boolean	否	Query 参数	是否将该节点组中的节点移出集群，并释放虚机资源、后付费公网IP和云磁盘. 默认为false

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 DELETE /v2/cluster/cce-f7zeyx1u/instancegroup/cce-ig-dvej1d3y?deleteInstances=true  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: b1a9e426-c0ca-4668-a1d8-624c5000d365
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "b1a9e426-c0ca-4668-a1d8-624c5000d365"
}
```

修改节点组节点自动扩缩容配置

描述

修改节点组节点自动扩缩容配置

请求结构

```
 PUT /v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/autoscaler HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
enabled	Boolean	是	Request Body 参数	是否启用Autoscaler
minReplicas	Integer	是	Request Body 参数	最小副本数. 取值范围是自然数集.
maxReplicas	Integer	是	Request Body 参数	最大副本数. 取值范围是自然数集, 需大于minReplicas.
scalingGroupPriority	Integer	是	Request Body 参数	伸缩组优先级. 取值范围是自然数集.

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 PUT /v2/cluster/cce-f7zeyx1u/instancegroup/cce-ig-dvej1d3y/autoscaler  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "enabled": true,
 "minReplicas": 2,
 "maxReplicas": 5,
 "scalingGroupPriority": 1
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 27f91a44-7257-48e7-a8d8-849f45a32da4
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "27f91a44-7257-48e7-a8d8-849f45a32da4"
}
```

修改节点组节点副本数

描述

修改节点组节点副本数

请求结构

```
 PUT /v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/replicas HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
replicas	Integer	是	Request Body 参数	期望的节点组节点的副本数. 取值范围是自然数集.
instanceIDs	List<String>	否	Request Body 参数	指定被添加或是优先被删除的节点 ID 集合
deleteInstance	Boolean	否	Request Body 参数	是否删除节点组收缩时被剔除的节点. 默认为 false. 此值的优先级高于节点组CleanPolicy, 被缩容节点使用该配置.
deleteOption	DeleteOption	否	Request Body 参数	修改指定节点删除选项.

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 PUT /v2/cluster/cce-f7zeyx1u/instancegroup/cce-ig-dvej1d3y/replicas  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "replicas": 1,
 "instanceIDs": null,
 "deleteInstance": true,
 "deleteOption": {
  "deleteResource": true,
  "deleteCDSSnapshot": true
 }
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6f593304-6787-45ea-8e0f-426ee331cc8b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "6f593304-6787-45ea-8e0f-426ee331cc8b"
}
```

获取集群节点组列表

描述

获取集群的节点组列表

请求结构

```
 GET /v2/cluster/{clusterID}/instancegroups HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
pageNo	Integer	否	Query 参数	查询页码序号. pageNo 或 pageSize 为0时会返回集群全部节点组
pageSize	Integer	否	Query 参数	查询结果每页条目数. pageNo 或 pageSize 为0时会返回集群全部节点组

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
page	ListInstanceGroupPage	是	节点组查询结果页
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /v2/cluster/cce-f7zeyx1u/instancegroups?pageNo=1&pageSize=10  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 1a65cc49-f3da-4d66-b994-7d7cd6797af5
```

```
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "1a65cc49-f3da-4d66-b994-7d7cd6797af5",
 "page": {
  "pageNo": 0,
  "pageSize": 0,
  "totalCount": 1,
  "list": [
   {
    "spec": {
     "cceInstanceGroupID": "cce-ig-dvej1d3y",
     "instanceGroupName": "sdk-testcase",
     "clusterID": "cce-z6qjgcq7",
     "clusterRole": "node",
     "shrinkPolicy": "Priority",
     "updatePolicy": "Concurrency",
     "cleanPolicy": "Delete",
     "instanceTemplate": {
      "instanceName": "",
      "runtimeType": "docker",
      "runtimeVersion": "18.9.2",
      "clusterID": "cce-z6qjgcq7",
      "clusterRole": "node",
      "instanceGroupID": "cce-ig-dvej1d3y",
      "instanceGroupName": "sdk-testcase",
      "existedOption": {},
      "machineType": "BCC",
      "instanceType": "N3",
      "bbcOption": {},
      "vpcConfig": {
       "vpcID": "vpc-pi9fghaxcpnf",
       "vpcSubnetID": "sbn-ww1xf6a5fi88",
       "securityGroupID": "g-4mnvpnrfscm1",
       "vpcSubnetType": "BCC",
       "vpcSubnetCIDR": "192.168.16.0/24",
       "availableZone": "zoneA"
      },
      "instanceResource": {
       "cpu": 1,
       "mem": 4,
       "rootDiskType": "hp1",
       "rootDiskSize": 40
      },
      "imageID": "m-4Umtt2i5",
      "instanceOS": {
       "imageType": "System",
       "imageName": "centos-8u0-x86_64-20200601205040",
       "osType": "linux",
       "osName": "CentOS",
       "osVersion": "8.0",
       "osArch": "x86_64 (64bit)",
       "osBuild": "2020060100"
      },
      "eipOption": {},
      "instanceChargingType": "Postpaid",
      "instancePreChargingOption": {},
      "deleteOption": {
       "deleteResource": true,
       "deleteCDSSnapshot": true
      },
      "deployCustomConfig": {
       "dockerConfig": {},
       "preUserScript": "bHM=",
       "postUserScript": "bHM="
      },
      "labels": {
       "cluster-id": "cce-z6qjgcq7",
       "cluster-role": "node",
```

```
        "instance-group-id": "cce-ig-dvej1d3y"
       },
       "cceInstancePriority": 5
      },
      "replicas": 3,
      "clusterAutoscalerSpec": {
       "enabled": false,
       "minReplicas": 0,
       "maxReplicas": 0,
       "scalingGroupPriority": 0
      }
     },
     "status": {
      "readyReplicas": 3,
      "pause": {
       "paused": false,
       "reason": ""
      }
     },
     "createdAt": "2020-09-27T06:34:51Z"
    }
   ]
  }
 }
```


创建扩容节点组任务

描述

创建扩容节点组任务。

请求结构

```
 PUT /v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/scaleup HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
upToReplicas	Integer	是（upToReplicas和upReplicas两者填其一）	Query 参数	扩容节点组的目标副本数，如果选择该参数则必须大于节点组的当前副本数
upReplicas	Integer	是（upToReplicas和upReplicas两者填其一）	Query 参数	节点组扩容的副本数，如果选择该参数则必须大于0

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述
taskID	String	节点组扩容任务 ID
requestID	String	请求 ID, 问题定位提供该 ID

请求示例

```
 PUT /api/cce/service/v2/cluster/cce-jvyvb9al/instancegroup/cce-ig-glw5jb7x/scaleup?upToReplicas=4 HTTP/1.1
Host: cce.bj.baidubce.com
Content-Type: application/json
Authorization: authorization string
```

返回示例

```
{
    "requestID": "deb8d7b9-37b2-4fc2-b956-0db512c37b5e",
    "taskID": "task-cce-ig-glw5jb7x-scaleup-zibi4ixw"
}
```

创建缩容节点组任务

描述

创建缩容节点组任务。

请求结构

```
PUT /v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/scaledown HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
instancesToBeRemoved	List<String>	是	Request Body 参数	缩容节点组时计划从节点组移除的节点 ID 列表
cleanPolicy	String	是	Request Body 参数	缩容节点组时是否保留节点对应的实例, 可选 [Remain,Delete]
deleteOption	DeleteOption	否	Request Body 参数	修改指定节点删除选项。cleanPolicy为Delete时，需要设置该参数。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述
taskID	String	节点组扩容任务 ID
requestID	String	请求 ID, 问题定位提供该 ID

请求示例

```
PUT /api/cce/service/v2/cluster/cce-91xd2ojb/instancegroup/cce-ig-l68ajx08/scaledown?= HTTP/1.1
Host: cce.bj.baidubce.com
Content-Type: application/json
Authorization: authorization string

{
    "instancesToBeRemoved": [
        "cce-91xd2ojb-6y5feg8v",
        "cce-91xd2ojb-6y5feg8v"
    ],
    "cleanPolicy":"Delete",
    "deleteOption":{
        "moveOut":true,
        "deleteResource":false,
        "deleteCDSSnapshot":false
    }
}
```

返回示例

```
{
    "requestID": "deb8d7b9-37b2-4fc2-b956-0db512c37b5e",
    "taskID": "task-cce-ig-l68ajx08-scaledown-zibi4ixw"
}
```

移入已有节点

描述

移入已有节点到节点组中。

返回头域

除公共头域，无其它特殊头域。

请求结构

```
 PUT  /v2/cluster/[clusterID]/instancegroup/[instanceGroupID]/attachInstances HTTP/1.1
Host: cce.bd.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	string	是	URL 参数	集群 ID，用于标识特定的集群。
instanceGroupID	string	是	URL 参数	节点组 ID，用于标识特定的节点组。
inCluster	bool	是	Request Body 参数	是否是集群内节点。
useInstanceGroupConfig	bool	是	Request Body 参数	针对集群外节点生效，设置为 true 将使用节点组配置。
existedInstances	List<InstanceSet>	否	Request Body 参数	配置集群外节点的详细信息。
existedInstancesInCluster	List<ExistedInstanceInCluster>	否	Request Body 参数	配置集群内节点的详细信息。

返回示例

```
 {
   "requestID": "deb8d7b9-37b2-4fc2-b956-0db512c37b5e",
   "taskID": "task-cce-ig-l68ajx08-scaledown-zibi4ixw"
}
```

请求示例 移入集群外节点

```
 PUT /api/cce/service/v2/cluster/cce-qfdlj5g9/instancegroup/cce-ig-iw245bl0/attachInstances HTTP/1.1
Host: cce.bd.baidubce.com
Authorization: authorization string

{
   "inCluster": false,
   "useInstanceGroupConfig": true,
   "existedInstances": [
      {
         "instanceSpec": {
            "sshKeyID": "k-FH1hIJ0h",
            "existed": true,
            "existedOption": {
               "existedInstanceID": "i-oCnp4BG5",
               "rebuild": true
            },
            "machineType": "BCC",
            "clusterRole": "node",
            "instanceOS": {
               "imageType": "System"
            },
            "imageID": "m-PljC8e8k",
            "instanceResource": {
               "cdsList": []
            }
         }
      }
   ]
}
```

移入集群内节点

```
 PUT /api/cce/service/v2/cluster/cce-qfdlj5g9/instancegroup/cce-ig-iw245bl0/attachInstances HTTP/1.1
Host: cce.bd.baidubce.com
Authorization: authorization string

{
    "inCluster": true,
    "existedInstancesInCluster": [
        {
            "existedInstanceID": "i-oCnp4BG5"
        }
    ]
}
```

返回示例

```
 {
    "requestID": "deb8d7b9-37b2-4fc2-b956-0db512c37b5e",
    "taskID": "task-cce-ig-l68ajx08-scaledown-zibi4ixw"
}
```

Instance相关接口

CCE Instance OpenAPI

创建节点（集群扩容）

描述

为集群添加节点

请求结构

```
 POST /v2/cluster/{clusterID}/instances  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群ID
无（RequestBody 为数组）	List<InstanceSet>	是	RequestBody 参数	为集群增加的节点列表

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
cceInstanceIDs	List<String>	是	新增节点的 ID 列表
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例：新建 BCC

```
 POST /v2/cluster/cce-f7zeyx1u/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

[ {
  "instanceSpec" : {
    "instanceName" : "",
    "clusterRole" : "node",
    "existed" : false,
    "machineType" : "BCC",
    "instanceType" : "N3",
    "vpcConfig" : {
      "vpcID" : "vpc-mwbgygrjb72w",
      "vpcSubnetID" : "sbn-mnbvhnuupv1u",
      "availableZone" : "zoneA",
      "securityGroup": {
        "customSecurityGroups": [],
        "enableCCERequiredSecurityGroup": true,
        "enableCCEOptionalSecurityGroup": true
      }
    },
    "instanceResource" : {
      "cpu" : 1,
      "mem" : 4,
      "rootDiskSize" : 40,
      "localDiskSize" : 0,
      "cdsList" : [ ]
    },
    "imageID" : "m-gTpZ1k6n",
    "instanceOS" : {
      "imageType" : "System"
    },
    "needEIP" : false,
    "bid":false,
    "adminPassword" : "test123!T",
    "sshKeyName" : "k-3uvrdvVq",
    "instanceChargingType" : "Postpaid",
    "runtimeType" : "docker"
  },
  "count" : 1
} ]
```

请求示例：使用已有节点

- 添加 Node 机器配置时，设置 `instanceSpec.existed` 为 `true` 并设置 `instanceSpec.existedOption.existedInstanceID` 为希望使用的已有节点 ID
- 如果不希望重装系统，设置 `instanceSpec.existedOption.rebuild` 为 `false` 并务必保证机器密码正确，否则节点会因无法部署相关服务而创建失败
- 如果不希望重装系统，无需设置 `instanceSpec.instanceOS` 与 `instanceSpec.machineType`
- 其它参数参考 API 文档按需设置

```
[
  {
     "instanceSpec":{
        "existed":true,
        "existedOption":{
           "existedInstanceID":"i-M56Un1DO",
           "rebuild":true
        },
        "machineType":"BCC",
        "instanceOS": {
           "imageName": "7.5 x86_64 (64bit)",
           "imageType": "System",
           "osType": "linux",
           "osName": "CentOS",
           "osVersion": "7.5",
           "osArch": "x86_64 (64bit)"
        },
        "adminPassword":"test123!T"
   }
]
```

请求示例：挂载CDS

有时我们希望在**新建节点**的同时为节点挂载 1 到多个 CDS，此时在创建集群时，为节点设置 CDS 参数和相关挂载路径即可。示例如下。

需要注意的是，每个路径下只能挂载一个 CDS，但一个路径的子路径下可以挂载另一个 CDS，例如 /a 目录下仅可挂载一个 CDS，但 /a/b 目录下可以挂载另一个 CDS。

已有节点仅可挂载 CDS 到指定路径，不会新建 CDS。因此当已有节点试图挂载 CDS 到指定 Path 但找不到满足条件的 CDS 时，将会忽略相关 CDS 配置。

```
[
    {
       "instanceSpec":{
          ......
          "instanceResource": {
             "cdsList": [
                {diskPath: "/home/cce", storageType: "cloud_hp1", cdsSize: 60}
              ]
          },
          ......
       }
    }
]
```

请求示例：新建 BBC

> 智能卡 BBC 需要使用普通子网，需要安全组 ID；非智能卡 BBC 需要使用 BBC 型子网，不能传安全组ID。

```
 POST /v2/cluster/cce-f7zeyx1u/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

[ {
  "instanceSpec" : {
    "instanceName" : "",
    "clusterRole" : "node",
    "existed" : false,
    "machineType" : "BBC",
    "vpcConfig" : {
      "vpcSubnetID" : "sbn-mnbvhnuupv1u",
      "securityGroupID" : "g-xh04bcdkq5n6"
    },
    "instanceResource" : {
      "rootDiskSize" : 100,
      "cdsList" : [ ]
    },
    "bbcOption": {
   "flavor": "BBC-G4-02S",
   "diskInfo": "NoRaid"
 },
    "imageID" : "m-zGnIzI5v",
    "instanceOS" : {
      "imageType" : "BbcSystem"
    },
    "adminPassword" : "test123!T",
    "instanceChargingType" : "Postpaid",
    "runtimeType" : "docker"
  },
  "count" : 1
} ]
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 58868d06-2e79-4d68-9f62-389e70f54996
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "cceInstanceIDs": [
        "cce-f7zeyx1u-7ombtu3j"
    ],
    "requestID": "58868d06-2e79-4d68-9f62-389e70f54996"
}
```

删除节点（集群缩容）

描述

为集群删除节点

请求结构

```
 PUT /v2/cluster/{clusterID}/instances  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID
deleteOption	DeleteOption	否	RequestBody 参数	删除选项. 为空时使用节点自身的 DeleteOption 配置
instanceIDs	List<String>	是	RequestBody 参数	要删除的节点 ID 列表
scaleDown	Boolean	否	RequestBody 参数	是否同时减少被删除节点所在节点组的期望节点数

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	响应的请求的ID

请求示例：将节点从集群中移出后同时删除相关资源

```
 PUT /v2/cluster/cce-f7zeyx1u/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "instanceIDs": [
  "cce-f7zeyx1u-7ombtu3j"
 ],

 "deleteOption": {
  "moveOut": false,
  "deleteResource": true,
  "deleteCDSSnapshot": true
 }
}
```

请求示例：仅将节点从集群中移出

```
 PUT /v2/cluster/cce-f7zeyx1u/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "instanceIDs": [
  "cce-f7zeyx1u-7ombtu3j"
 ],

 "deleteOption": {
  "moveOut": true
 }
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 23a82d7e-954f-4539-b828-ce620eaa97b3
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "requestID": "23a82d7e-954f-4539-b828-ce620eaa97b3"
}
```

更新节点属性

描述

更新一个节点的某些配置信息。目前仅支持部分属性的更新。

请求结构

```
PUT /v2/cluster/{clusterID}/instance/{instanceID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceID	String	是	URL 参数	节点 ID
labels	Map<String,String>	是	RequestBody 参数	节点的标签
annotations	Map<String,String>	否	RequestBody 参数	节点的注解
taints	List<Taint>	是	RequestBody 参数	节点的污点
cceInstancePriority	Integer	是	RequestBody 参数	节点的优先级

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
instance	Instance	是	节点的查询结果
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
PUT /v2/cluster/cce-oi0ihu53/instance/i-bn71n2eK HTTP/1.1
Host: cce.hkg.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "labels": {
        "cluster-id": "cce-b3gzxpwg",
        "cluster-role": "node",
        "new-label-key": "new-label-value"
    },
    "taints": [
        {
            "key": "testKey",
            "value": "testValue",
            "effect": "NoSchedule"
        }
    ],
    "cceInstancePriority": 5
}
```

返回示例

```
HTTP/1.1 200 OK
X-Bce-Request-Id: 60dae0e8-8fa0-443c-a5d5-45b1e7beea68
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "instance": {
        "spec": {
            "cceInstanceID": "cce-oi0ihu53-pfmx3lu6",
            "instanceName": "cce-oi0ihu53-pfmx3lu6",
            "runtimeType": "docker",
            "runtimeVersion": "18.9.2",
            "clusterID": "cce-oi0ihu53",
            "clusterRole": "node",
            "userID": "eca97e148cb74e9683d7b7240829d1ff",
```

```
            "instanceGroupID": "",
            "instanceGroupName": "",
            "machineType": "BCC",
            "instanceType": "N3",
            "bbcOption": {},
            "vpcConfig": {
                "vpcID": "vpc-epyxw9mwjc18",
                "vpcSubnetID": "sbn-11gfqpsax4jn",
                "securityGroupID": "g-urmir67c35x3",
                "vpcSubnetType": "BCC",
                "VPCSubnetCIDR": "192.168.0.0/16",
                "VPCSubnetCIDRIPv6": "",
                "availableZone": "zoneA"
            },
            "instanceResource": {
                "cpu": 4,
                "mem": 12,
                "rootDiskType": "hp1",
                "rootDiskSize": 40
            },
            "deployCustomConfig": {
                "dockerConfig": {}
            },
            "imageID": "m-Yp443gTZ",
            "instanceOS": {
                "imageType": "System",
                "imageName": "centos-7u5-x86_64-20200601203742",
                "osType": "linux",
                "osName": "CentOS",
                "osVersion": "7.5",
                "osArch": "x86_64 (64bit)",
                "osBuild": "2020060100"
            },
            "needEIP": false,
            "eipOption": {},
            "sshKeyID": "",
            "instanceChargingType": "Postpaid",
            "deleteOption": {
                "deleteResource": true,
                "deleteCDSSnapshot": true
            },
            "labels": {
                "cluster-id": "cce-oi0ihu53",
                "cluster-role": "node"
            },
            "cceInstancePriority": 5
        },
        "status": {
            "machine": {
                "instanceID": "i-bn71n2eK",
                "instanceUUID": "60c466f2-929a-4cf9-af7c-52af00fa4476",
                "orderID": "0fe2237296b644da8ef2b215b4ff19c8",
                "vpcIP": "192.168.0.31"
            },
            "instancePhase": "running",
            "machineStatus": "ACTIVE"
        },
        "createdAt": "2020-09-15T06:43:20Z",
        "updatedAt": "2020-09-25T06:03:40Z"
    },
    "requestID": "60dae0e8-8fa0-443c-a5d5-45b1e7beea68"
}
```


获取节点详情

描述

获取一个节点的详细信息

请求结构

```
GET /v2/cluster/{clusterID}/instance/{instanceID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceID	String	是	URL 参数	节点 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
instance	Instance	是	节点的查询结果
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
GET /v2/cluster/cce-oi0ihu53/instance/i-bn71n2eK HTTP/1.1
Host: cce.hkg.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
HTTP/1.1 200 OK
X-Bce-Request-Id: a13e8b5f-6878-4f1f-8746-3e986ee49a96
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "instance": {
        "spec": {
            "cceInstanceID": "cce-oi0ihu53-pfmx3lu6",
            "instanceName": "cce-oi0ihu53-pfmx3lu6",
            "runtimeType": "docker",
            "runtimeVersion": "18.9.2",
            "clusterID": "cce-oi0ihu53",
            "clusterRole": "node",
            "userID": "eca97e148cb74e9683d7b7240829d1ff",
            "instanceGroupID": "",
            "instanceGroupName": "",
            "machineType": "BCC",
            "instanceType": "N3",
            "bbcOption": {},
            "vpcConfig": {
                "vpcID": "vpc-epyxw9mwjc18",
                "vpcSubnetID": "sbn-11gfqpsax4jn",
                "securityGroupID": "g-urmir67c35x3",
                "vpcSubnetType": "BCC",
                "VPCSubnetCIDR": "192.168.0.0/16",
                "VPCSubnetCIDRIPv6": "",
                "availableZone": "zoneA"
            },
            "instanceResource": {
                "cpu": 4,
                "mem": 12,
                "rootDiskType": "hp1",
                "rootDiskSize": 40
            },
            "deployCustomConfig": {
                "dockerConfig": {}
            }
```

```
            },
            "imageID": "m-Yp443gTZ",
            "instanceOS": {
                "imageType": "System",
                "imageName": "centos-7u5-x86_64-20200601203742",
                "osType": "linux",
                "osName": "CentOS",
                "osVersion": "7.5",
                "osArch": "x86_64 (64bit)",
                "osBuild": "2020060100"
            },
            "needEIP": false,
            "eipOption": {},
            "sshKeyID": "",
            "instanceChargingType": "Postpaid",
            "deleteOption": {
                "deleteResource": true,
                "deleteCDSSnapshot": true
            },
            "labels": {
                "cluster-id": "cce-oi0ihu53",
                "cluster-role": "node"
            },
            "cceInstancePriority": 5
        },
        "status": {
            "machine": {
                "instanceID": "i-bn71n2eK",
                "instanceUUID": "60c466f2-929a-4cf9-af7c-52af00fa4476",
                "orderID": "0fe2237296b644da8ef2b215b4ff19c8",
                "vpcIP": "192.168.0.31"
            },
            "instancePhase": "running",
            "machineStatus": "ACTIVE"
        },
        "createdAt": "2020-09-15T06:43:20Z",
        "updatedAt": "2020-09-24T08:27:21Z"
    },
    "requestID": "74a74b2e-f439-4147-b27a-f473e3576338"
}
```


获取集群节点列表

描述

请求集群节点列表

请求结构

```
 GET /v2/cluster/{clusterID}/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群ID
keywordType	String	否	Query 参数	集群模糊查询字段，可选 [instanceName, instanceID, clusterRole]，默认值为 instanceName
keyword	String	否	Query 参数	查询关键词
orderBy	String	否	Query 参数	集群查询排序字段，可选 [instanceName, instanceID, createdAt]，默认值为 instance_name
order	String	否	Query 参数	排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC
pageNo	Integer	否	Query 参数	页码，默认值为1
pageSize	Integer	否	Query 参数	单页结果数，默认值为10

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
instancePage	InstancePage	是	集群的节点查询结果
requestID	String	是	响应的请求的ID

请求示例

```
 GET /v2/cluster/cce-f7zeyx1u/instances?keywordType=instanceName&keyword=&orderBy=createdAt&order=asc&pageNo=1&pageSize=10
HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: a13e8b5f-6878-4f1f-8746-3e986ee49a96
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "instancePage": {
        "clusterID": "cce-f7zeyx1u",
        "keywordType": "instanceName",
        "keyword": "",
        "orderBy": "createdAt",
        "order": "ASC",
        "pageNo": 1,
        "pageSize": 10,
        "totalCount": 1,
        "instanceList": [
            {
                "spec": {
                    "cceInstanceID": "cce-f7zeyx1u-vbsry5uc",
                    "instanceName": "cce-f7zeyx1u-vbsry5uc",
                    "runtimeType": "docker",
                    "runtimeVersion": "18.9.2",
                    "clusterID": "cce-f7zeyx1u",
                    "clusterRole": "node",
                    "userID": "eca97e148cb74e9683d7b7240829d1ff",
                    "instanceGroupID": "",
                    "instanceGroupName": "",
                    "machineType": "BCC",
                    "instanceType": "N3",
                    "bbcOption": {},
                    "vpcConfig": {
                        "vpcID": "vpc-mwbgygrjb72w",
                        "vpcSubnetID": "sbn-mnbvhnuupv1u",
                        "securityGroupID": "g-xh04bcdkq5n6",
                        "vpcSubnetType": "BCC",
                        "VPCSubnetCIDR": "192.168.0.0/20",
                        "VPCSubnetCIDRIPv6": "",
                        "availableZone": "zoneA"
                    },
                    "instanceResource": {
                        "cpu": 1,
                        "mem": 4,
                        "rootDiskType": "hp1",
                        "rootDiskSize": 40
                    },
                    "deployCustomConfig": {
                        "dockerConfig": {}
                    },
                    "imageID": "m-gTpZ1k6n",
                    "instanceOS": {
                        "imageType": "System",
                        "imageName": "centos-7u3-x86_64-20191105104647",
```

```
                    "osType": "linux",
                    "osName": "CentOS",
                    "osVersion": "7.3",
                    "osArch": "x86_64 (64bit)",
                    "osBuild": "2019110600"
                },
                "needEIP": false,
                "eipOption": {},
                "sshKeyID": "",
                "instanceChargingType": "Postpaid",
                "deleteOption": {},
                "labels": {
                    "cluster-id": "",
                    "cluster-role": "node"
                }
            },
            "status": {
                "machine": {
                    "instanceID": "i-HJitrtYn",
                    "orderID": "b49e015b819c4fd1887b8848c7a38025",
                    "vpcIP": "192.168.4.25"
                },
                "instancePhase": "running",
                "machineStatus": "ACTIVE"
            },
            "createdAt": "2020-07-21T12:12:54Z",
            "updatedAt": "2020-07-21T12:15:00Z"
        }
    ]
  },
  "requestID": "a13e8b5f-6878-4f1f-8746-3e986ee49a96"
}
```


获取节点组节点列表

描述

获取节点组节点列表

请求结构

```
 GET /v2/cluster/{clusterID}/instancegroup/{instanceGroupID}/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
instanceGroupID	String	是	URL 参数	节点组 ID
pageNo	Integer	否	Query 参数	页码，默认值为 0. 取值为 0 时返回节点组下的所有节点
pageSize	Integer	否	Query 参数	单页结果数，默认值为 0. 取值为 0 时返回节点组下的所有节点

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
page	ListInstancesByInstanceGroupIDPage	是	集群的节点查询结果
requestID	String	是	响应的请求的 ID

请求示例

```
 GET /v2/cluster/cce-47bqnhmj/instancegroup/cce-ig-796lmt7a/instances?pageNo=1&pageSize=10  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: d52c646c-207d-4630-95ba-0539124fd42b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "d52c646c-207d-4630-95ba-0539124fd42b",
 "page": {
  "pageNo": 0,
  "pageSize": 0,
  "totalCount": 3,
  "list": [
   {
    "spec": {
     "cceInstanceID": "cce-z6qjgcq7-7dd7z1fc",
     "instanceName": "cce-z6qjgcq7-7dd7z1fc",
     "runtimeType": "docker",
     "runtimeVersion": "18.9.2",
     "clusterID": "cce-z6qjgcq7",
     "clusterRole": "node",
     "instanceGroupID": "cce-ig-dvej1d3y",
     "instanceGroupName": "sdk-testcase",
     "existedOption": {},
     "machineType": "BCC",
     "instanceType": "N3",
     "bbcOption": {},
     "vpcConfig": {
      "vpcID": "vpc-pi9fghaxcpnf",
      "vpcSubnetID": "sbn-ww1xf6a5fi88",
      "vpcSubnetType": "BCC",
      "availableZone": "zoneA"
     },
     "instanceResource": {
      "cpu": 1,
      "mem": 4,
      "rootDiskType": "hp1",
      "rootDiskSize": 40
     },
     "imageID": "m-4Umtt2i5",
     "instanceOS": {
      "imageType": "System",
      "imageName": "centos-8u0-x86_64-20200601205040",
      "osType": "linux",
      "osName": "CentOS",
      "osVersion": "8.0",
      "osArch": "x86_64 (64bit)",
      "osBuild": "2020060100"
     },
     "eipOption": {},
     "instanceChargingType": "Postpaid",
     "instancePreChargingOption": {},
     "deleteOption": {
      "deleteResource": true,
      "deleteCDSSnapshot": true
     },
     "deployCustomConfig": {
      "dockerConfig": {}
     },
     "labels": {
      "cluster-id": "cce-z6qjgcq7",
      "cluster-role": "node",
      "instance-group-id": "cce-ig-dvej1d3y"
     }
```

```
    },
    "cceInstancePriority": 5
   },
   "status": {
    "machine": {
     "instanceID": "i-HVq1GwKr",
     "orderID": "71f6d4249aed4a08b2bc1470fefd6b2a",
     "vpcIP": "192.168.16.33"
    },
    "instancePhase": "running",
    "machineStatus": "ACTIVE"
   },
   "createdAt": "2020-09-27T06:35:04Z",
   "updatedAt": "2020-09-27T06:36:53Z"
  },
  {
   "spec": {
    "cceInstanceID": "cce-z6qjgcq7-asf0p09i",
    "instanceName": "cce-z6qjgcq7-asf0p09i",
    "runtimeType": "docker",
    "runtimeVersion": "18.9.2",
    "clusterID": "cce-z6qjgcq7",
    "clusterRole": "node",
    "instanceGroupID": "cce-ig-dvej1d3y",
    "instanceGroupName": "sdk-testcase",
    "existedOption": {},
    "machineType": "BCC",
    "instanceType": "N3",
    "bbcOption": {},
    "vpcConfig": {
     "vpcID": "vpc-pi9fghaxcpnf",
     "vpcSubnetID": "sbn-ww1xf6a5fi88",
     "vpcSubnetType": "BCC",
     "availableZone": "zoneA"
    },
    "instanceResource": {
     "cpu": 1,
     "mem": 4,
     "rootDiskType": "hp1",
     "rootDiskSize": 40
    },
    "imageID": "m-4Umtt2i5",
    "instanceOS": {
     "imageType": "System",
     "imageName": "centos-8u0-x86_64-20200601205040",
     "osType": "linux",
     "osName": "CentOS",
     "osVersion": "8.0",
     "osArch": "x86_64 (64bit)",
     "osBuild": "2020060100"
    },
    "eipOption": {},
    "instanceChargingType": "Postpaid",
    "instancePreChargingOption": {},
    "deleteOption": {
     "deleteResource": true,
     "deleteCDSSnapshot": true
    },
    "deployCustomConfig": {
     "dockerConfig": {}
    },
    "labels": {
     "cluster-id": "cce-z6qjgcq7",
     "cluster-role": "node",
     "instance-group-id": "cce-ig-dvej1d3y"
    },
    "cceInstancePriority": 5
   },
   "status": {
    "machine": {
     "instanceID": "i-RndUYIWF",
```

```
    "orderID": "71f6d4249aed4a08b2bc1470fefd6b2a",
    "vpcIP": "192.168.16.34"
   },
   "instancePhase": "running",
   "machineStatus": "ACTIVE"
  },
  "createdAt": "2020-09-27T06:35:06Z",
  "updatedAt": "2020-09-27T06:36:58Z"
 },
 {
  "spec": {
   "cceInstanceID": "cce-z6qjgcq7-ibtt74zc",
   "instanceName": "cce-z6qjgcq7-ibtt74zc",
   "runtimeType": "docker",
   "runtimeVersion": "18.9.2",
   "clusterID": "cce-z6qjgcq7",
   "clusterRole": "node",
   "instanceGroupID": "cce-ig-dvej1d3y",
   "instanceGroupName": "sdk-testcase",
   "existedOption": {},
   "machineType": "BCC",
   "instanceType": "N3",
   "bbcOption": {},
   "vpcConfig": {
    "vpcID": "vpc-pi9fghaxcpnf",
    "vpcSubnetID": "sbn-ww1xf6a5fi88",
    "vpcSubnetType": "BCC",
    "availableZone": "zoneA"
   },
   "instanceResource": {
    "cpu": 1,
    "mem": 4,
    "rootDiskType": "hp1",
    "rootDiskSize": 40
   },
   "imageID": "m-4Umtt2i5",
   "instanceOS": {
    "imageType": "System",
    "imageName": "centos-8u0-x86_64-20200601205040",
    "osType": "linux",
    "osName": "CentOS",
    "osVersion": "8.0",
    "osArch": "x86_64 (64bit)",
    "osBuild": "2020060100"
   },
   "eipOption": {},
   "instanceChargingType": "Postpaid",
   "instancePreChargingOption": {},
   "deleteOption": {
    "deleteResource": true,
    "deleteCDSSnapshot": true
   },
   "deployCustomConfig": {
    "dockerConfig": {}
   },
   "labels": {
    "cluster-id": "cce-z6qjgcq7",
    "cluster-role": "node",
    "instance-group-id": "cce-ig-dvej1d3y"
   },
   "cceInstancePriority": 5
  },
  "status": {
   "machine": {
    "instanceID": "i-aRGKcLtv",
    "orderID": "71f6d4249aed4a08b2bc1470fefd6b2a",
    "vpcIP": "192.168.16.35"
   },
   "instancePhase": "running",
   "machineStatus": "ACTIVE"
  }
```

```
    },
    "createdAt": "2020-09-27T06:35:07Z",
    "updatedAt": "2020-09-27T06:37:02Z"
   }
  ]
 }
}
```

获取节点事件步骤

描述

获取创建或删除过程中节点所处的事件步骤,请求参数同时兼容cceInstanceID和instanceID

请求结构

```
 GET /v2/event/instance/{instanceID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
instanceID	String	是	URL 参数	节点 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必选	描述
status	String	是	事件类型
steps	List<Step>	是	集群操作步骤
requestID	String	是	请求ID

请求示例

```
 GET /v2/event/instance/i-ezs9bmjY  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 08eb305a-87fb-4360-8a27-42b9561edf4a
Date: Fri, 12 Aug 2022 03:10:30 GMT
Content-Type: application/json;charset=UTF-8

{
    "status": "created",
    "steps": [
        {
            "stepName": "准备机器",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-08-04T02:45:28Z",
            "finishedTime": "2022-08-04T02:45:56Z",
            "costSeconds": 28,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "绑定安全组",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-08-04T02:45:56Z",
            "finishedTime": "2022-08-04T02:45:57Z",
            "costSeconds": 1,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "前置检查",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-08-04T02:45:57Z",
            "finishedTime": "2022-08-04T02:48:38Z",
            "costSeconds": 161,
            "retryCount": 2,
            "errInfo": {}
        },
        {
            "stepName": "部署 K8S",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-08-04T02:48:38Z",
            "finishedTime": "2022-08-04T02:49:20Z",
            "costSeconds": 42,
            "retryCount": 1,
            "errInfo": {}
        },
        {
            "stepName": "等待就绪",
            "stepStatus": "done",
            "ready": true,
            "startTime": "2022-08-04T03:40:41Z",
            "finishedTime": "2022-08-04T03:40:41Z",
            "retryCount": 1,
            "errInfo": {}
        }
    ],
    "requestID": "08eb305a-87fb-4360-8a27-42b9561edf4a"
}
```

同步节点元信息

描述

同步并更新集群下所有节点的IAAS配置信息。例如：用户变更BCC付费方式，可通过接口进行数据同步。

请求结构

```
 POST /v2/sync/cluster/{clusterID}/instances  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
clusterID	String	是	URL 参数	集群 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必选	描述
clusterID	String	是	请求的集群ID
requestID	String	是	请求ID

请求示例

```
 POST /v2/sync/cluster/cce-f7zeyx1u/instances HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 23a82d7e-954f-4539-b828-ce620eaa97b3
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "requestID": "23a82d7e-954f-4539-b828-ce620eaa97b3",
    "clusterID": "cce-f7zeyx1u"
}
```

Kubeconfig相关接口

查询集群KubeConfig

描述

查询集群 KubeConfig

请求结构

```
 GET /v2/kubeconfig/{clusterID}/{type}  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL 参数	集群 ID
type	String	是	URL 参数	Kubeconfig 类型, 可选 [vpc, public], 分别表示使用 BLB VPCIP, 使用 BLB EIP.

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
kubeConfig	String	是	kubeconfig 内容
kubeConfigType	String	是	Kubeconfig 类型, 可选 [vpc, public], 分别表示使用 BLB VPCIP, 使用 BLB EIP.
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /v2/kubeconfig/cce-br0i4kl5/vpc HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: f3da66d2-4c9f-40c9-aa5c-ab5b2d827361
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8


{
 "kubeConfigType": "vpc",
 "kubeConfig": "apiVersion: v1\nclusters:\n- cluster:\n    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ3SUJBZ0lVQit0NWJteWcydjVHMHRFcnVEdTdMVWxQWnpNd0RRWUpLb1pJaHZjTkFRRUwKQlFBd1p6RUxNQWtHQTF
n    server: https://100.64.230.48:6443\n  name: kubernetes\ncontexts:\n- context:\n    cluster: kubernetes\n    user:
eca97e148cb74e9683d7b7240829d1\n  name: eca97e148cb74e9683d7b7240829d1@kubernetes\ncurrent-context:
eca97e148cb74e9683d7b7240829d1@kubernetes\nkind: Config\npreferences: {}\nusers:\n- name:
eca97e148cb74e9683d7b7240829d1\n  user:\n    client-certificate-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCkSdjNYbDM0SmlxZ1dwUnF0alhWVE1zd0RRWUpLb1pJaHZjTkFRRUwKQlFBd1p6RUxNQWtHQTFVRU
n    client-key-data:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBclZ4UUl3TzZIdE80NUZqL1RHd1FYSFJCS2pGcTU0NGRxZGMwUU
,
 "requestID": "f3da66d2-4c9f-40c9-aa5c-ab5b2d827361"
}
```

Network相关接口

检查集群ClusterIP网段

描述

检查集群 ClusterIP 网段

请求结构

```
 POST /v2/net/check_clusterip_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
ipVersion	String	是	RequestBody 参数	容器网络IP地址版本，可选[ipv4, ipv6, dualStack]
clusterIPCIDR	String	否	RequestBody 参数	Cluster IPv4 网段. IP版本为IPv4时必填
clusterIPCIDRIPv6	String	否	RequestBody 参数	Cluster IPv6 网段. IP版本为IPv6时必填
vpcID	String	是	RequestBody 参数	VPC ID
vpcCIDR	String	否	RequestBody 参数	VPC IPv4 网段. IP版本为IPv4时必填
vpcCIDRIPv6	String	否	RequestBody 参数	VPC IPv6 网段. IP版本为IPv6时必填

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
errMsg	String	否	错误信息
isConflict	String	是	是否冲突
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/net/check_clusterip_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "vpcID": "vpc-pi9fghaxcpnf",
 "vpcCIDR": "192.168.0.0/16",
 "vpcCIDRIPv6": "",
 "clusterIPCIDR": "172.31.0.0/16",
 "clusterIPCIDRIPv6": "",
 "ipVersion": "ipv4"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: d1b66c3d-b16f-4ff2-bedf-af21f6bcd827
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "isConflict": false,
 "errMsg": "",
 "requestID": "40165eb4-0fc7-405e-bcdf-10ba2ffc1ee3"
}
```

检查集群容器网络网段

描述

检查集群容器网络网段

请求结构

```
 POST /v2/net/check_container_network_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
ipVersion	String	否	RequestBody 参数	容器网络IP地址版本，默认值为 ipv4，可选 [ipv4, ipv6, dualStack]
clusterIPCIDR	String	否	RequestBody 参数	Cluster IPv4 网段. IP 版本为 IPv4 时必填
clusterIPCIDRIPv6	String	否	RequestBody 参数	Cluster IPv6 网段. IP 版本为 IPv6 时必填
containerCIDR	String	否	RequestBody 参数	容器网络 IPv4 网段. IP 版本为 IPv4 时必填
containerCIDRIPv6	String	否	RequestBody 参数	容器网络 IPv6 网段. IP 版本为 IPv6 时必填
maxPodsPerNode	Integer	是	RequestBody 参数	单节点最大容器组数量
vpcID	String	是	RequestBody 参数	VPC ID
vpcCIDR	String	否	RequestBody 参数	VPC IPv4 网段. IP版本为IPv4时必填
vpcCIDRIPv6	String	否	RequestBody 参数	VPC IPv6 网段. IP版本为IPv6时必填

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
clusterIPCIDRConflict	ClusterIPCIDRConflict	否	ClusterIP 网段冲突信息
containerCIDRConflict	ContainerCIDRConflict	否	容器网段冲突信息
errMsg	String	否	错误信息
isConflict	String	是	是否冲突
maxNodeNum	String	是	最大节点数量
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/net/check_container_network_cidr HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "vpcID": "vpc-pi9fghaxcpnf",
 "vpcCIDR": "192.168.0.0/16",
 "vpcCIDRIPv6": "",
 "containerCIDR": "172.28.0.0/16",
 "containerCIDRIPv6": "",
 "clusterIPCIDR": "172.31.0.0/16",
 "clusterIPCIDRIPv6": "",
 "maxPodsPerNode": 256,
 "ipVersion": "ipv4"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: a2c5b12b-6005-4266-a8e8-f5b3d903a5c7
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "maxNodeNum": 256,
 "isConflict": false,
 "errMsg": "",
 "containerCIDRConflict": null,
 "clusterIPCIDRConflict": null,
 "requestID": "a2c5b12b-6005-4266-a8e8-f5b3d903a5c7"
}
```

推荐ClusterIP网络网段

描述

推荐 ClusterIP 网络网段

请求结构

```
 POST /v2/net/recommend_clusterip_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterMaxServiceNum	Integer	是	RequestBody 参数	集群最大 Service 数量
ipVersion	String	否	RequestBody 参数	容器网络IP地址版本，默认值为 ipv4，可选 [ipv4, ipv6, dualStack]
containerCIDR	String	否	RequestBody 参数	容器网络 IPv4 网段. IP 版本为 IPv4 时必填
containerCIDRIPv6	String	否	RequestBody 参数	容器网络 IPv6 网段. IP 版本为 IPv6 时必填
privateNetCIDRs	List<String>	否	RequestBody 参数	IPv4私有网络地址段，可选 [10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16]
privateNetCIDRIPv6s	List<String>	否	RequestBody 参数	IPv6私有网络地址段，目前仅支持 [fc00::/7]
vpcCIDR	String	否	RequestBody 参数	VPC IPv4 网段. IP版本为IPv4时必填
vpcCIDRIPv6	String	否	RequestBody 参数	VPC IPv6 网段. IP版本为IPv6时必填

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
errMsg	String	否	错误信息
isSuccess	Boolean	是	请求是否成功
recommendedClusterIPCIDRs	List<String>	否	推荐 Cluster IP 网段
recommendedClusterIPCIDRIPv6s	List<String>	否	推荐 Cluster IP 网段 IPv6
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/net/recommend_clusterip_cidr HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "vpcCIDR": "192.168.0.0/16",
 "vpcCIDRIPv6": "",
 "containerCIDR": "172.28.0.0/16",
 "containerCIDRIPv6": "",
 "clusterMaxServiceNum": 8,
 "privateNetCIDRs": [
  "172.16.0.0/12"
 ],
 "privateNetCIDRIPv6s": null,
 "ipVersion": "ipv4"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 79f1993e-fc76-41c3-8f20-a22bd1010324
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "recommendedClusterIPCIDRs": [
  "172.31.255.248/29",
  "172.31.255.240/29",
  "172.31.255.232/29",
  "172.31.255.224/29",
  "172.31.255.216/29"
 ],
 "recommendedClusterIPCIDRIPv6s": null,
 "isSuccess": true,
 "errMsg": "",
 "requestID": "79f1993e-fc76-41c3-8f20-a22bd1010324"
}
```

推荐容器网络网段

描述

推荐容器网络网段

请求结构

```
 POST /v2/net/recommend_container_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterMaxNodeNum	Integer	是	RequestBody 参数	集群最大节点数
ipVersion	String	否	RequestBody 参数	容器网络IP地址版本，默认值为 ipv4，可选 [ipv4, ipv6, dualStack]
k8sVersion	String	是	RequestBody 参数	集群的K8S版本
maxPodsPerNode	Integer	是	RequestBody 参数	单节点最大容器组数量
privateNetCIDRs	List<String>	否	RequestBody 参数	IPv4私有网络地址段，可选 [10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16]
privateNetCIDRIPv6s	List<String>	否	RequestBody 参数	IPv6私有网络地址段，目前仅支持 [fc00::/7]
vpcID	String	是	RequestBody 参数	VPC ID
vpcCIDR	String	否	RequestBody 参数	VPC IPv4 网段. IP版本为IPv4时必填
vpcCIDRIPv6	String	否	RequestBody 参数	VPC IPv6 网段. IP版本为IPv6时必填

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
errMsg	String	否	错误信息
isSuccess	Boolean	是	请求是否成功
recommendedContainerCIDRs	List<String>	否	推荐容器网段
recommendedContainerCIDRIPv6s	List<String>	否	推荐容器网段 IPv6
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 POST /v2/net/recommend_container_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
 "vpcID": "vpc-pi9fghaxcpnf",
 "vpcCIDR": "192.168.0.0/16",
 "vpcCIDRIPv6": "",
 "clusterMaxNodeNum": 2,
 "maxPodsPerNode": 32,
 "privateNetCIDRs": [
  "172.16.0.0/12"
 ],
 "privateNetCIDRIPv6s": null,
 "k8sVersion": "1.16.8",
 "ipVersion": "ipv4"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 162b630e-38f9-4b31-addc-f89339058e70
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "recommendedContainerCIDRs": [
  "172.16.0.0/26",
  "172.16.0.64/26",
  "172.16.0.128/26",
  "172.16.0.192/26",
  "172.16.1.0/26"
 ],
 "recommendedContainerCIDRIPv6s": null,
 "isSuccess": true,
 "errMsg": "",
 "requestID": "162b630e-38f9-4b31-addc-f89339058e70"
}
```

辅助网段冲突检测

描述

检查辅助网段是否与当前vpc内已有路由冲突

请求结构

```
 POST /v2/net/check_auxiliary_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
ipVersion	String	是	RequestBody 参数	容器网络IP地址版本，可选[ipv4, ipv6, dualStack]
ContainerCIDR	String	否	RequestBody 参数	Container IPv4 网段. IP版本为IPv4时必填
ContainerCIDRIPv6	String	否	RequestBody 参数	Container IPv6 网段. IP版本为IPv6时必填
vpcID	String	是	RequestBody 参数	VPC ID
vpcCIDR	String	否	RequestBody 参数	VPC IPv4 网段. IP版本为IPv4时必填
vpcCIDRIPv6	String	否	RequestBody 参数	VPC IPv6 网段. IP版本为IPv6时必填
isEBPF	Bool	是	RequestBody 参数	集群是否启用了ebpf加速

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
errMsg	String	否	错误信息
isConflict	Bool	是	是否冲突
requestID	String	是	请求 ID, 问题定位提供该 ID
isAuxiliary	Bool	是	在启用ebpf时使用，用于判断是否为辅助网段

请求示例

```
 POST /v2/net/check_auxiliaryip_cidr  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "vpcID": "vpc-pi9fghaxcpnf",
    "vpcCIDR": "192.168.0.0/16",
    "vpcCIDRIPv6": "",
    "ContainerCIDR": "172.31.0.0/16",
    "ContainerCIDRIPv6": "",
    "ipVersion": "ipv4"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: d1b66c3d-b16f-4ff2-bedf-af21f6bcd827
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "isConflict": false,
    "errMsg": "",
    "requestID": "d1b66c3d-b16f-4ff2-bedf-af21f6bcd827",
    "isAuxiliary": true
}
```

Task相关接口

获取任务列表

描述

获取任务列表

请求结构

```
 GET /v2/tasks/{taskType} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
taskType	String	是	URL 参数	任务类型，目前仅支持 [InstanceGroupReplicas]
targetID	String	是	Query 参数	任务目标对象 ID，任务类型为 InstanceGroupReplicas 时，targetID 是目标节点组的 ID
operationType	String	否	Query 参数	操作类型，目前支持[ScalingUp, ScalingDown, Repair, ExistedScaleup]，ScalingUp表示扩容，ScalingDown表示缩容，Repair 表示修改期望节点数的伸缩处理，ExistedScaleup表示添加已有节点到节点组的处理
phase	String	否	Query 参数	任务状态，目前支持[Pending, Processing, Done, Aborted, Collecting]
order	String	否	Query 参数	排序方式，目前支持[ASC,DESC], 默认为DESC(降序)，如果传ASC需要保证orderBy非空才有效。
orderBy	String	否	Query 参数	排序字段，目前仅支持[startTime], 默认为startTime（创建时间）
pageNo	Integer	否	Query 参数	查询页码序号。 为0时不分页
pageSize	Integer	否	Query 参数	查询结果每页条目数。为0时不分页

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述
page	ListTaskPage	任务查询结果页
requestID	String	请求 ID, 问题定位提供该 ID

请求示例

```
GET /api/cce/service/v2/tasks/InstanceGroupReplicas?targetID=cce-ig-h5zc8dqu&pageNo=1&pageSize=2 HTTP/1.1
Host: cce.bj.baidubce.com
Content-Type: application/json
Authorization: authorization string
```

返回示例

```
{
  "requestID": "dc172e92-67b1-4697-97f7-ded9cccfa455",
  "Page": {
    "pageNo": 1,
    "pageSize": 2,
    "totalCount": 5,
    "items": [
      {
        "id": "task-cce-ig-h5zc8dqu-scaleupexist-2w2snkxf",
        "type": "InstanceGroupReplicas",
        "description": "attach 4 instances to instancegroup",
        "startTime": "2025-04-11T11:39:40Z",
        "finishTime": "2025-04-11T11:47:09Z",
        "phase": "Done",
        "processes": [
          {
            "name": "MoveExistedInstanceIntoInstanceGroup",
            "phase": "Done",
            "startTime": "2025-04-11T19:39:41+08:00",
            "finishTime": "2025-04-11T19:47:09+08:00",
            "metrics": {
              "cce-fm8eewfh-5v7gw5g2": "create_failed",
              "cce-fm8eewfh-iw5kw5pu": "running",
              "cce-fm8eewfh-l3f4lsbd": "running",
              "cce-fm8eewfh-obgcevn2": "running"
            }
          }
        ]
      },
      {
        "id": "task-cce-ig-h5zc8dqu-scaledown-9xejb2fj",
        "type": "InstanceGroupReplicas",
        "description": "remove specified instances from instance group",
        "startTime": "2025-04-11T11:32:56Z",
        "finishTime": "2025-04-11T11:37:10Z",
        "phase": "Done",
        "processes": [
          {
            "name": "RemoveInstancesFromInstanceGroup",
            "phase": "Done",
            "startTime": "2025-04-11T19:32:56+08:00",
            "finishTime": "2025-04-11T19:37:10+08:00",
            "metrics": {
              "cce-fm8eewfh-60cyxs3v": "Succeed",
              "cce-fm8eewfh-djzjnidb": "Succeed",
              "cce-fm8eewfh-nom6v0on": "Succeed",
              "cce-fm8eewfh-qbdqj6aj": "Succeed"
            }
          }
        ]
      }
    ]
  }
}
```


查看任务详情

描述

查看任务详情

请求结构

```
GET /v2/task/{taskType}/{taskID} HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
taskType	String	是	URL 参数	任务类型，目前仅支持 [InstanceGroupReplicas]
taskID	String	是	URL 参数	任务 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述
task	Task	任务详情
requestID	String	请求 ID, 问题定位提供该 ID

请求示例

```
 GET /api/cce/service/v2/task/InstanceGroupReplicas/task-cce-ig-h5zc8dqu-scaleupexist-2w2snkxf~~~~ HTTP/1.1
Host: cce.bj.baidubce.com
Content-Type: application/json
Authorization: authorization string
```

返回示例

```
{
  "requestID": "58731113-1d37-49e8-8644-afbc55cee5ef",
  "task": {
    "id": "task-cce-ig-h5zc8dqu-scaleupexist-2w2snkxf",
    "type": "InstanceGroupReplicas",
    "description": "attach 4 instances to instancegroup",
    "startTime": "2025-04-11T11:39:40Z",
    "finishTime": "2025-04-11T11:47:09Z",
    "phase": "Done",
    "processes": [
      {
        "name": "MoveExistedInstanceIntoInstanceGroup",
        "phase": "Done",
        "startTime": "2025-04-11T19:39:41+08:00",
        "finishTime": "2025-04-11T19:47:09+08:00",
        "metrics": {
          "cce-fm8eewfh-5v7gw5g2": "create_failed",
          "cce-fm8eewfh-iw5kw5pu": "running",
          "cce-fm8eewfh-l3f4lsbd": "running",
          "cce-fm8eewfh-obgcevn2": "running"
        }
      }
    ]
  }
}
```

组件管理相关接口

组件管理相关接口

获取组件状态

获取组件的基本信息与状态。

如果已经安装，还会返回部署参数等信息。

请求结构

```
 GET /v2/cluster/{ClusterID}/addon
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID。
addons	String	否	Query参数	为空时查询所有组件的信息。查询多个组件时按逗号分隔，如cce-ingress-controller,cce-gpu-controller。取值参考文档末尾附录。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求ID，问题定位提供该ID。
items	List<AddOnInfo>	否	组件查询结果。每个数组元素是一个组件的查询结果。
code	String	否	错误代码，仅在请求失败时存在。
message	String	否	错误信息，仅在请求失败时存在。

请求示例

```
GET /v2/cluster/cce-5e130ugt/addon?addons=cce-ingress-controller HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: aef503ab-66e2-4b7f-9044-e922389ed03f
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "items": [
        {
            "meta": {
                "name": "cce-ingress-controller",
                "type": "Networking",
                "latestVersion": "1.3.5",
                "shortIntroduction": "基于百度云应用型负载均衡产品（应用型BLB）实现K8S Ingress语义，提供七层网络负载均衡能力",
                "defaultParams": "\n# Default values for cce-ingress-controller\n# This is a YAML-formatted file.\n# Declare variables to be
passed into your templates.\n\nClusterVersion: v2\nImageID: registry.baidubce.com/cce-plugin-dev/cce-ingress-
controller:rc853b4b_20230117\n\nOpenCCEGatewayEndpoint:\nBLBOpenAPIEndpoint:\nEIPOpenAPIEndpoint:\nVPCEndpoint:\nCCEV2Endpoir
:  # 集群地域\nClusterID: # 集群id\nConcurrentIngressSyncs: # 最多同时处理的 Service 的数量\nDefaultMaxRSCount: # 组件默认为 BLB
挂载的最大后端数\n",
                "installInfo": {
                    "allowInstall": true
                }
            },
            "instance": {
                "name": "cce-ingress-controller",
                "installedVersion": "2022.11.08.1552",
                "params": "# No Params",
                "status": {
                    "phase": "Running"
                },
                "uninstallInfo": {
                    "allowUninstall": true
                },
                "upgradeInfo": {
                    "allowUpgrade": false,
                    "nextVersion": "",
                    "message": "暂不支持升级版本"
                },
                "updateInfo": {
                    "allowUpdate": false,
                    "message": "不支持更新参数"
                }
            }
        }
    ],
    "requestID": "62709f56-3f0b-4fcc-86e0-c18547ba1ce7"
}
```

安装组件

向集群中安装组件。

每个组件需要的安装参数不同，部署参数均以字符串的形式传入。

请求结构

```
 POST /v2/cluster/{ClusterID}/addon
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID。
name	String	是	Request Body参数	要安装的组件名称 。
params	String	否	Request Body参数	组件安装参数。每个组件需要的参数不同，请参照【附录】部分。部分组件不需要设置安装参数。
version	String	否	Request Body参数	要指定安装的组件版本。通常不需要设置该参数。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求ID，问题定位提供该ID。
code	String	否	错误代码，仅在请求失败时存在。
message	String	否	错误信息，仅在请求失败时存在。

请求示例

```
 POST /v2/cluster/cce-5e130ugt/addon HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "name": "cce-gpu-manager",
    "params": "EnableHook: true\nEnableSGPU: true\n\n"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6f593304-6787-45ea-8e0f-426ee331cc8b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "6f593304-6787-45ea-8e0f-426ee331cc8b"
}
```

卸载组件

卸载集群中已经安装的组件。

请求结构

```
 DELETE /v2/cluster/{ClusterID}/addon
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID。
name	String	是	Request Body参数	要卸载的组件名称。
instanceName	String	否	Request Body参数	要卸载的组件实例名称。仅在允许多实例部署的组件中，用于指定要卸载的组件实例。通常不会使用到该字段。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求ID，问题定位提供该ID。
code	String	否	错误代码，仅在请求失败时存在。
message	String	否	错误信息，仅在请求失败时存在。

请求示例

```
 DELETE /v2/cluster/cce-5e130ugt/addon HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "name": "cce-ingress-controller"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6f593304-6787-45ea-8e0f-426ee331cc8b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "6f593304-6787-45ea-8e0f-426ee331cc8b"
}
```

更新组件部署参数

更新集群中已经安装的组件的部署参数。

请求结构

```
 PUT /v2/cluster/{ClusterID}/addon
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID。
name	String	是	Request Body参数	要更新的组件名称。
instanceName	String	否	Request Body参数	要更新的组件实例名称。仅在允许多实例部署的组件中，用于指定要更新的组件实例。通常不会使用到该字段。
params	String	是	Request Body参数	要更新的部署参数。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求ID，问题定位提供该ID。
code	String	否	错误代码，仅在请求失败时存在。
message	String	否	错误信息，仅在请求失败时存在。

请求示例

```
 PUT /v2/cluster/cce-mehgoi9r/addon HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "name": "cce-hybrid-manager",
    "params": "A: a"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6f593304-6787-45ea-8e0f-426ee331cc8b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "6f593304-6787-45ea-8e0f-426ee331cc8b"
}
```

升级组件

升级集群中已经安装的组件的版本。

请求结构

```
 POST /v2/cluster/{ClusterID}/addon/upgrade
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	URL参数	集群ID。
name	String	是	Request Body参数	要升级的组件名称。
targetVersion	String	否	Request Body参数	升级的目标版本。通常不会使用到该字段，目标版本由后台自动决定。
instanceName	String	否	Request Body参数	要升级的组件实例名称。仅在允许多实例部署的组件中，用于指定要升级的组件实例。通常不会使用到该字段。
params	String	否	Request Body参数	要在升级时同时更新的参数。通常不会使用到该字段。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求ID，问题定位提供该ID。
code	String	否	错误代码，仅在请求失败时存在。
message	String	否	错误信息，仅在请求失败时存在。

请求示例

```
 POST /v2/cluster/cce-jx4l7afz/addon/upgrade HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "name": "cce-npu-manager"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 6f593304-6787-45ea-8e0f-426ee331cc8b
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "6f593304-6787-45ea-8e0f-426ee331cc8b"
}
```

组件名称列表

组件名称	参数名称
CCE GPU Manager	cce-gpu-manager
CCE AI Job Scheduler	cce-volcano
CCE RDMA Device Plugin	cce-rdma-plugin
CCE PaddleFlow	cce-paddleflow
CCE Fluid	cce-fluid
CCE Deep Learning Frameworks Operator	cce-aibox
CCE Image Accelerate	cce-image-accelerate
CCE Hybrid Manager	cce-hybrid-manager
CCE Ingress NGINX Controller	cce-ingress-nginx-controller
CCE Ingress Controller	cce-ingress-controller
CCE CSI CDS Plugin	cce-csi-cds-plugin
CCE CSI BOS Plugin	cce-csi-bos-plugin
CCE CSI PFS Plugin	cce-csi-pfs-plugin
CCE NPU Manager	cce-npu-manager
CCE Log Operator	cce-log-operator

组件示例参数

CCE GPU Manager

```
 EnableSGPU: false # true 使用内核态隔离，false 使用用户态隔离
GPUShareMemoryUnit: GiB # 显存共享资源上报单位 GiB/MiB
IgnoreDeviceType: false # 是否为不区分卡类型，默认 false 为上报资源中带有卡型号
```

CCE AI Job Scheduler

```
 Binpack: true # Binpack: false 表示启用 spread
Reclaimgang: true # 是否开启队列内抢占
Preemptgang: true # 是否开启队列间抢占
```

CCE RDMA Device Plugin

该组件不需要用户设置参数

CCE PaddleFlow

```
global:
  PF_SELFED_DB_ENABLED: &selfd_mysql true
  PF_DB_DATABASE: &pf_db_database paddleflow
  PF_DB_HOST: &pf_db_host mysql-standalone
  PF_DB_PASSWORD: &mysqlpwd Paddle@2022
  PF_DB_USER: &pf_db_user root
  PF_DB_PORT: &pf_db_port 3306
  PF_SVC_TYPE: &pf_svc_type NodePort
  PF_SERVER_NODE_PORT: &pf_server_node_port 30999
  PF_LB_IP: &eip "0.0.0.0"
  CCE_LB_ID: "lb-12345678"
  CCE_LB_SBN_ID: "sbn-123456789123"
  CCE_LB_INTERNAL: true
  CCE_LB_POD_DERICT: false
  IMAGE_REPOSITORY: registry.baidubce.com/cce-plugin-dev/paddleflow

paddleflow-server:
  out_depend_msg:
    PF_DB_DATABASE: *pf_db_database
    PF_DB_HOST: *pf_db_host
    PF_DB_PASSWORD: *mysqlpwd
    PF_DB_PORT: *pf_db_port
    PF_DB_USER: *pf_db_user
  paddleflow_server:
    service:
      extra_usr_define_services:
        paddleflow-server:
          ports:
            port-0:
              nodePort: *pf_server_node_port
          loadBalancerIP: *eip
          type: *pf_svc_type


paddleflow-db-init:
  out_depend_msg:
    PF_DB_DATABASE: *pf_db_database
    PF_DB_HOST: *pf_db_host
    PF_DB_PASSWORD: *mysqlpwd
    PF_DB_PORT: *pf_db_port
    PF_DB_USER: *pf_db_user


mysql-replication:
  enabled: *selfd_mysql
  fullnameOverride: *pf_db_host
  nameOverride: *pf_db_host
  auth:
    rootPassword: *mysqlpwd
    database: *pf_db_database
    username: *pf_db_user
    password: *mysqlpwd
  primary:
    service:
      port: *pf_db_port
```

CCE Fluid

该组件不需要用户设置参数

CCE Deep Learning Frameworks Operator

```
  TFOperatorEnable: false
MPIOperatorEnable: false
PyTorchOperatorEnable: false
PaddleOperatorEnable: false
MXNetOperatorEnable: false
AITrainingOperatorEnable: false
```

CCE Image Accelerate

该组件不需要用户设置参数

CCE Hybrid Manager

```
 version:
 scheduler: 1.2.8
 hybridlet: 1.2.8

globalSLA:
 cpu:
   highPercent: 65
   bestEffortMaxCores: 60
 memory:
   highPercent: 80
   bestEffortMax: 60
 net:
   inHigh: 10000
   bestEffortInHigh: 10000
   outHigh: 10000
   bestEffortOutHigh: 10000
 coolDownSec: 10
 expulsionDelaySec: 60
 maxInstances: 256
```

CCE Ingress NGINX Controller

```
 controller:
 ingressClass: ads
 kind: DaemonSet
 nodeSelector:
   instance-group-id: cce-ig-ubn1700b
 resources:
   limits:
     cpu: 0.5
     memory: 1024Mi
   requests:
     cpu: 0.25
     memory: 256Mi
 scope:
   enabled: false
   namespace: ""
 service:
   annotations:
     service.beta.kubernetes.io/cce-load-balancer-internal-vpc: true
 tolerations: []
fullnameOverride: ads-ngx-control
isArchArm64: false
```

CCE Ingress Controller

该组件不需要用户设置参数

CCE CSI CDS Plugin

```
  maxVolumesPerNode: 5 # maxVolumesPerNode 集群中每个节点最大可以挂载的 CDS PV 数量
cluster:            # kubeletRootPath 用户节点 Kubelet 数据目录。要将集群所有节点出现过的数据目录都列在这里。如果节点没有特别修改过 kubelet 数据目录，这里可以不传。
  nodes:
    - kubeletRootPath: "/home/cce/kubelet"
      kubeletRootPathAffinity: true
    - kubeletRootPath: "/data/kubelet"
      kubeletRootPathAffinity: true
    - kubeletRootPath: "/var/lib/kubelet"
      kubeletRootPathAffinity: true
```

CCE CSI BOS Plugin

```
  maxVolumesPerNode: 5 # maxVolumesPerNode 集群中每个节点最大可以挂载的 BOS PV 数量
cluster:            # kubeletRootPath 用户节点 Kubelet 数据目录。要将集群所有节点出现过的数据目录都列在这里。如果节点没有特别修改过 kubelet 数据目录，这里可以不传。
  nodes:
  - kubeletRootPath: "/home/cce/kubelet"
    kubeletRootPathAffinity: true
  - kubeletRootPath: "/data/kubelet"
    kubeletRootPathAffinity: true
  - kubeletRootPath: "/var/lib/kubelet"
    kubeletRootPathAffinity: true
```

CCE CSI PFS Plugin

```
  pfs:
  configEndpoint: ""     # configEndpoint PFS 连接地址和端口
  parentDir: /kubernetes # parentDir 代表CSI有权限读写的子路径，需要使用静态PV挂载的路径必须在这个路径之内。最大可以填 /，默认为 /kubernetes。挂载前需确认该路径已经对CCE集群机器授权。
nodes:                  # kubeletRootPath 用户节点 Kubelet 数据目录。要将集群所有节点出现过的数据目录都列在这里。如果节点没有特别修改过 kubelet 数据目录，这里可以不传。
  - kubeletRootPath: "/var/lib/kubelet"
    kubeletRootPathAffinity: true
  - kubeletRootPath: "/home/cce/kubelet"
    kubeletRootPathAffinity: true
  - kubeletRootPath: "/data/kubelet"
    kubeletRootPathAffinity: true
```

CCE NPU Manager

```
  XPUUseSriov: false # 开启则使用硬件隔离切分，不支持虚机
XPUSriovNumVfs: 3  # 指定昆仑硬件切分的份数
```

CCE Log Operator

该组件不需要用户设置参数

附录

ClusterSpec

参数名称	类型	是否必须	描述
clusterID	String	否	集群ID. 创建集群时不需要传递此字段
clusterName	String	是	集群名称. 集群名称只能包含英文大小写字母、数字、-、.、和_ 名称长度不超过65个字符，不可为空
clusterType	String	否	集群类型，目前仅支持 normal. 默认值 normal
description	String	否	集群描述
k8sVersion	String	是	K8S版本号，可选 [1.18.9, 1.20.8, 1.21.14, 1.22.5, 1.24.4, 1.26.9]
runtimeType	String	否	容器运行时类型，可选 [docker, containerd, bci] 一般集群默认值 docker, Serverless集群默认为BCI
runtimeVersion	String	否	容器运行时的版本，目前仅支持18.9.2. 默认值18.9.2
vpcID	String	是	VPC ID
vpcCIDR	String	否	VPC 网段 创建集群时无需设置此值
vpcCIDRIPv6	String	否	VPC IPv6 网段 创建集群时无需设置此值
plugins	List<String>	否	插件列表 支持的插件包括 [cce-ingress-controller，cluster-autoscaler，core-dns，core-dns-for-serverless，cronhpa，ip-masq-agent，kongming-nvidia，kube-proxy，kunlun-nvidia，metrics-adapter，metrics-server，network-inspector，nvidia-gpu，vpc-cni，vpc-route] 其中core-dns、kube-proxy, metrics-server会在所有集群默认部署；容器网络模式为kubenet时会默认部署ip-masq-agent；GPU 共享型集群会默认部署kongming-nvidia，否则会部署nvidia-gpu； VPC路由模式CNI时会部署vpc-route，VPC辅助IP模式会部署vpc-cni；
masterConfig	MasterConfig	是	Master节点配置
containerNetworkConfig	ContainerNetworkConfig	是	容器网络配置
k8sCustomConfig	K8SCustomConfig	否	K8S自定义配置
PluginsConfig	map[string]PluginHelmConfig	否	插件 Helm 安装配置
ForbidDelete	bool	否	集群删除保护标识，true 表示开启删除保护不允许删除集群；false 表示关闭删除保护允许删除集群
ResourceChargingOption	ResourceChargingOption	否	IaaS资源付费选项
AuthenticateMode	string	否	APIServer 认证模式，可选x509、oidc
Tags	List<Tag>	否	标签

CreateClusterOptions

参数名称	类型	是否必须	描述
skipNetworkCheck	Boolean	否	是否强行跳过容器网络的检查

MasterConfig

参数名称	类型	是否必须	描述
masterType	String	是	Master 部署类型，新建可选 [managedPro, containerizedCustom, serverless] 。接口返回的可能值有：[managed, managedPro, custom, containerizedCustom, serverless] 。
clusterHA	Integer	否	Master 副本数，可选 [1, 3, 5, 2]. 对于托管型集群其值可选[1, 3],默认值为3. 对于Serverless集群其值仅可为2. 自定义集群无需设置此值.
exposedPublic	Boolean	否	是否向公网暴露
clusterBLBVPCSubnetID	String	否	集群的 BLB VPC 子网 ID. 托管型集群无需设置此值, 自定义集群必须设置此值.
managedClusterMasterOption	ManagedClusterMasterOption	否	托管型集群的 Master 节点选项. 仅在集群类型是托管型时需要设置.
serverlessMasterOption	ServerlessMasterOption	否	Serverless Master 节点选项. 仅在集群类型是Serverless时需要设置

ManagedClusterMasterOption

参数名称	类型	是否必须	描述
masterVPCSubnetZone	String	否	Master 所在的 VPC 子网区域，可选 [zoneA, zoneB, zoneC, zoneD, zoneE, zoneF]. 默认值为zoneA.
masterFlavor	String	否	托管集群规格，可选 [L50, L200, L500, L1000, L3000, L5000]. 默认值为L50.
clusterBLBSource	String	否	集群Apiserver使用的BLB归属，默认值为USER.

ServerlessMasterOption

参数名称	类型	是否必须	描述
masterSecurityGroupID	String	否	集群master安全组，后台自动覆盖，用户无需手动填写
vkSecurityGroupID	String	是	集群中启动的bci实例的安全组
vkSubnets	List<VKSubnetType>	否	集群中启动的bci实例所在的子网列表，如果不传将会使用 clusterBLBVPCSubnetID

ContainerNetworkConfig

参数名称	类型	是否必须	描述
mode	String	是	容器的网络模式，可选 [kubenet, vpc-cni, vpc-route-veth, vpc-route-ipvlan, vpc-route-auto-detect, vpc-secondary-ip-veth, vpc-secondary-ip-ipvlan, vpc-secondary-ip-auto-detect]
eniVPCSubnetIDs	Map<String,List<String>>	否	ENI VPC 子网 ID
eniSecurityGroupID	String	否	ENI 安全组ID
ipVersion	String	否	容器IP类型，可选 [ipv4, ipv6, dualStack]，默认值ipv4
lbServiceVPCSubnetID	String	是	关联 BLB 所在子网 ID
nodePortRangeMax	Integer	否	指定 NodePort Service 的端口范围，默认值32767, 最大值65536
nodePortRangeMin	Integer	否	指定 NodePort Service 的端口范围，默认值30000, 最大值65536
clusterPodCIDR	String	否	集群 Pod IP 网段, 在 kubenet 网络模式下有效. 网络类型是VPC-CNI时自动使用VPC的CIDR
clusterPodCIDRIPv6	String	否	集群 Pod IPv6 网段, 在 kubenet 网络模式下有效.网络类型是VPC-CNI时自动使用VPC的CIDR
clusterIPServiceCIDR	String	否	Service ClusterIP 的网段. ipv4时设置
clusterIPServiceCIDRIPv6	String	否	Service ClusterIP 的 IPv6 网段. ipv6时设置
maxPodsPerNode	Integer	否	每个 Node 上最大的 Pod 数，默认值128
kubeProxyMode	String	否	kube-proxy 代理模式，可选 [ipvs, iptables]，默认值为 ipvs

K8SCustomConfig

参数名称	类型	是否必须	描述
masterFeatureGates	Map<String,Boolean>	否	自定义 MasterFeatureGates
nodeFeatureGates	Map<String,Boolean>	否	自定义 NodeFeatureGates
admissionPlugins	List<String>	否	自定义 AdmissionPlugins
pauseImage	String	否	自定义 PauseImage
kubeAPIQPS	Integer	否	自定义 KubeAPIQPS
kubeAPIBurst	Integer	否	自定义 KubeAPIBurst
schedulerPredicates	List<String>	否	自定义 SchedulerPredicates
schedulerPriorities	Map<String,Integer>	否	自定义 SchedulerPrioritiess
etcdDataPath	String	否	自定义 etcd 数据目录

InstanceSet

参数名称	类型	是否必须	描述
instanceSpec	InstanceSpec	是	节点配置信息
count	Integer	否	使用上述配置的节点数量. 当节点配置是已有节点时无需设置此值

InstanceSpec

参数名称	类型	是否必须	描述
cceInstanceID	String	否	用于 CCE 唯一标识 Instance 如果用户不指定: CCE 默认生成；如果用户指定: CCE 按照规则生成
instanceN			

ame	String	否	节点名称
runtimeType	String	否	容器运行时类型，可选 [docker, bci] 一般集群默认值 docker, Serverless集群默认为BCI
runtimeVersion	String	否	容器运行时的版本，目前仅支持18.9.2. 默认值18.9.2
clusterID	String	否	集群 ID. 在创建集群时无需填写
clusterRole	String	否	节点在集群中的角色，可选 [master, node]. 创建集群时无需填写.
instanceGroupID	String	否	节点所属节点组 ID
instanceGroupName	String	否	节点所属节点组名称
masterType	String	否	Master 机器来源。可选 [managedPro, containerizedCustom, serverless] 。接口返回的可能值有：[managed, managedPro, custom, containerizedCustom, serverless]。
existed	Boolean	否	是否为已有节点. 仅在节点类型为已有节点时需要设置
existedOption	ExistedOption	否	已有实例相关配置. 仅在节点类型为已有节点时需要设置.
machineType	String	否	机器类型，可选 [BCC, BBC, Metal, BCI]. 用户无需设置此值. 对于Serverless自动设为BCI. 对于托管型集群的Master自动设为BCC. 其他新建节点自动设为BCC. 已有节点会根据其节点类型自动设为BCC或BBC
instanceType	String	否	机器规格，可选 [N1, N2, N3, N4, N5, C1, C2, S1, G1, F1, ServerlessMaster]. 详情参考：实例规格 仅自定义新建节点需要设置. 对于Serverless集群的Master自动设为ServerlessMaster. 对于托管型集群的Master使用DefaultMasterConfig中配置. 已有节点使用本节点的节点类型
deploySetID	String	否	部署集 ID
autoSnapshotID	String	否	自动快照策略ID
bbcOption	BBCOption	否	BBC 选项. 仅在节点类型为BBC类型已有节点时需要设置
vpcConfig	VPCConfig	否	VPC 选项. 新建BCC节点时和新建Serverless Master时需要设置. 托管型Master节点组自动使用Master Config配置. 已有节点自动使用自身VPC配置.
instanceResource	InstanceResource	否	集群规格相关配置. 新建BCC节点时必须设置. 托管型Master节点组自动使用后台默认配置. 已有节点自动使用自身资源配置.
imageID	String	否	新建BCC节点和已有节点需要重装系统时时需要设置 imageID 和 InstanceOS 二者中的其中一个. 优先使用ImageID, 如果用户传入 InstanceOS 信息, 由后台计算 ImageID.
instanceOS	InstanceOS	否	新建BCC节点和已有节点需要重装系统时时需要设置 imageID 和 InstanceOS 二者中的其中一个. 优先使用ImageID, 如果用户传入 InstanceOS 信息, 由后台计算 ImageID.
needEIP	Boolean	否	机器是否需要EIP
relationTag	Boolean	否	是否将该节点上的tag应用到与该节点绑定的其他资源上，例如cds盘，默认为false
userData	String	否	节点自定义数据, 支持安装驱动。因为传输API请求时，不会加密所设置的UserData，建议不要以明文方式传入机密的信息，例如密码和私钥等。如果必须传入，建议加密后，然后以Base64的方式编码后再传入，在节点内部以同样的方式反解密
eipOption	EIPOption	否	EIP 选项. needEIP为True时必须设置.
adminPassword	String	否	管理员密码. 不设置时将由系统自动生成. 已有BCC节点在不重装系统时必须设置. 密码要求8～32位字符, 仅限且必须包含字母、数字和指定符号 !@#$%^*()
sshKeyID	String	否	SSH Key ID
instanceChargingType	String	否	节点计费方式，可选 [Prepaid, Postpaid, bidding]. 新建节点、托管型集群Master节点、Serverless节点默认且仅限为后付费. 已有节点支持预付费或后付费
instancePr	InstancePr		

eCharging Option	InstanceChargingOption	否	节点预付费选项. 预付费节点需要设置.
deleteOption	DeleteOption	否	删除节点选项.
deployCustomConfig	DeployCustomConfig	否	自定义部署选项
tags	List<Tag>	否	节点 Tag 列表.
labels	Map<String,String>	否	节点 Label 列表. 后台会节点自动添加cluster-id和cluster-role两个label
taints	List<Taint>	否	节点 Taint 列表
annotations	Map<String,String>	否	节点 annotations 列表
bid	Boolean	否	是否开启竞价，默认为 false
bidOption	BidOption	否	竞价实例选项，bid=true 时为必填项
isOpenHostnameDomain	Boolean	否	是否自动生成hostname domain
ehcClusterId	String	否	ehc集群ID

ExistedOption

参数名称	类型	是否必须	描述
existedInstanceID	String	是	现有节点 ID
rebuild	Boolean	否	是否重装系统，默认为true，即重装系统

BBCOption

参数名称	类型	是否必须	描述
reserveData	Boolean	是	是否保留数据
raidID	String	否	磁盘阵列类型 ID；reserveData=false 时必填, reserveData=true 时不生效
sysDiskSize	Integer	否	系统盘分配大小，单位 GB；reserveData=false 时必填, reserveData=true 时不生效
flavor	String	否	套餐ID，例：BBC-G4-02S
diskInfo	String	否	磁盘阵列类型，例：Raid5, 不支持 raid 的 bbc，设置为 NoRaid。

VPCConfig

参数名称	类型	是否必须	描述
vpcID	String	否	VPC ID. 为空时使用集群的VPC ID
vpcSubnetID	String	否	VPC 子网 ID. 新建节点必须设置此值. 已有节点无需设置.
securityGroupID	String	是	安全组 ID. 新建节点如果没有设置 securityGroup 必须设置此值. 已有节点无需设置.
securityGroupType	String	否	安全组类型，可选[normal，enterprise]，默认为 normal
securityGroup	SecurityGroup	否	安全组信息. 新建 BCC 推荐设置. 已有节点无需设置.
vpcSubnetType	String	否	VPC 子网类型，可选 [BBC, BCC]. 创建集群时无需设置，后台根据子网ID自动设置.
vpcSubnetCIDR	String	否	VPC 子网网段. 创建集群时无需设置，后台根据子网ID自动设置.
vpcSubnetCIDRIPv6	String	否	VPC IPv6 子网网段. 创建集群时无需设置，后台根据子网ID自动设置.
availableZone	String	否	可用区，可选 [zoneA, zoneB, zoneC, zoneD, zoneE, zoneF]. 创建集群时无需设置，后台根据子网ID自动设置.

SecurityGroup

参数名称	类型	是否必须	描述
customSecurityGroups	String 数组	否	用户指定的安全组，将被绑定到节点上
enableCCERequiredSecurityGroup	bool	否	是否绑定 CCE 默认安全组到节点上
enableCCEOptionalSecurityGroup	bool	否	是否绑定 CCE 可选安全组到节点上

InstanceResource

参数名称	类型	是否必须	描述
cpu	Integer	否	CPU 核数. 新建节点必须设置此字段
mem	Integer	否	内存大小，单位GB. 新建节点必须设置此字段
rootDiskType	String	否	根磁盘类型，可选 [hp1, cloud_hp1, hdd, local, sata, ssd]. 新建节点默认为hp1 已有节点和其本身属性一致. 更多详情参考：CDS磁盘性能 与 CDS磁盘类型参数
rootDiskSize	Integer	否	根磁盘大小，单位GB. 默认值为40
localDiskSize	Integer	否	本地磁盘大小，GPU 机器必须指定，单位 GB
cdsList	List<CDSConfig>	否	CDS 列表，默认第一块盘作为 docker 和 kubelet 数据盘
gpuType	String	否	GPU 类型，可选 [V100-32, V100-16, P40, P4, K40, DLCard]. 详情参考: GPU卡详情 节点类型为G1时必须设定
gpuCount	Integer	否	GPU 数量. 节点类型为G1时必须设定
machineSpec	String	是	机器规格，必填参数
cpuThreadConfig	String	否	调整每物理核的线程数（vCPU），本质上对应各处理器的超线程能力是否启用。 取值范围：1、2 注意： 1. 默认情况下，百度智能云实例按照默认值 2 配置。 2. 仅Intel第七代以上、AMD第三代以上的裸金属实例规格支持设置 CPU 线程数。 3. 实例创建后该参数配置不可修改。
numaConfig	String	否	调整CPU的NUMA配置，对于不同处理器平台取值有不同含义。 1. Intel平台：0代表关闭NUMA特性，1代表开启NUMA特性。 2. AMD平台：主要影响NPS(Nodes Per Socket)的配置，可取值0、1、2、4、auto，分别对应NPS0、NPS1、NPS2、NPS4以及自动。 注意： 1. 默认情况下，百度智能云实例按照开启NUMA优化（Intel实例）、NPS1（AMD实例），也即默认值 1 配置。 2. 仅Intel第七代以上、AMD第三代以上的裸金属实例规格支持设置NUMA选项。 3. 实例创建后该参数配置不可修改。

CDSConfig

参数名称	类型	是否必须	描述
diskPath	String	是	磁盘路径
storageType	String	是	存储类型，可选 [hp1, cloud_hp1, hdd, local, sata, ssd] 更多详情参考：CDS磁盘性能 与 CDS磁盘类型参数
cdsSize	Integer	是	磁盘空间大小
snapshotID	String	否	快照ID，支持从快照创建磁盘

InstanceOS

参数名称	类型	是否必须	描述
imageType	String	是	镜像类型。取值范围包括 [All, System, Custom, Integration, Sharing, GpuBccSystem, GpuBccCustom, BbcSystem, BbcCustom]
imageName	String	是	镜像名字。例如ubuntu-14.04.1-server-amd64-201506171832
osType	String	是	操作系统类型，可选 [linux, windows]
osName	String	是	操作系统名字，可选 [CentOS, Ubuntu, Windows Server, Debian, opensuse]
osVersion	String	是	操作系统版本，例如14.04.1 LTS
osArch	String	是	操作系统架构。例如x86_64 (64bit)
osBuild	String	否	镜像创建时间信息，例如2015061700

EIPOption

参数名称	类型	是否必须	描述
eipName	String	是	EIP 名称
eipChargeType	String	是	EIP的计费方式，可选 [ByTraffic, ByBandwidth]
eipPurchaseType	String	否	EIP线路类型，可选[BGP(标准型BGP),BGP_S(增强型BGP),ChinaTelcom,ChinaUnicom,ChinaMobile]，默认标准BGP。
eipBandwidth	Integer	是	EIP 带宽. 按带宽计费取值范围是1-200. 按流量计费取值范围是1-1000

InstancePreChargingOption

参数名称	类型	是否必须	描述
purchaseTime	Integer	是	购买时间
purchaseTimeUnit	String	是	购买时间单位
autoRenew	Boolean	是	是
autoRenewTimeUnit	String	是	续费单位
autoRenewTime	Integer	是	续费时间

DeleteOption

参数名称	类型	是否必须	描述
moveOut	Boolean	否	是否移出节点，true表示仅将节点移出集群，false表示将节点删除。创建集群时新建节点默认为false， 加入已有节点默认为true。
deleteResource	Boolean	否	是否删除相关资源。创建集群时新建节点默认为true，加入已有节点默认为false。
deleteCDSSnapshot	Boolean	否	是否删除CDS快照。创建集群时新建节点默认为true，加入已有节点默认为false。
drainNode	Boolean	否	是否进行节点排水。

DeployCustomConfig

参数名称	类型	是否必须	描述
dockerConfig	List<DockerConfig>	否	Docker 相关配置
containerdConfig	List<ContainerdConfig>	否	Containerd 相关配置
kubeletRootDir	String	否	kubelet 数据目录
registryPullQPS	Integer	否	每秒钟可以执行的镜像仓库拉取操作限值。 此值必须为非负整数，将其设置为 0 表示没有限值
registryBurst	Integer	否	突发性镜像拉取的上限值，允许镜像拉取临时上升到所指定数量，不能超过 registryPullQPS 所设置的约束。此值必须是非负整数。只有 registryPullQPS 参数值大于 0 时才会使用此设置
podPidsLimit	Integer	否	每个 Pod 中可使用的 PID 个数上限
eventRecordQPS	Integer	否	每秒钟可创建的事件个数上限。此值必须为非负整数，将其设置为 0 表示没有限值
eventBurst	Integer	否	突发性事件创建的上限值，允许事件创建临时上升到所指定数量，不过仍然不超过 eventRecordQPS 所设置的约束。此值必须为非负整数，只有 eventRecordQPS > 0 时才会使用此设置
kubeAPIQPS	Integer	否	与 Kubernetes API 服务器通信时的 QPS 限制（每秒查询数）
kubeAPIBurst	Integer	否	与 Kubernetes API 服务器通信时突发的流量限制，此值必须为非负整数
maxPods	Integer	否	节点 kubelet 上运行的 Pod 个数上限。此值必须为非负整数
cpuManagerPolicy	string	否	要使用的cpuManagerPolicy策略名称，可选[none，static]，默认值为none
topologyManagerScope	string	否	拓扑管理器作用域，可选[pod，container],默认值为container
topologyManagerPolicy	string	否	拓扑管理器策略名称，可选[none，best-effort，restricted，single-numa-node]，默认值为none
cpuCFSQuota	Boolean	否	是否为设置了CPU限制的容器实施CPU CFS配额约束
postUserScriptFailedAutoCordon	Boolean	否	部署时执行脚本失败后是否自动封锁节点
kubeletBindAddressType	string	否	kubelet绑定地址类型，可选[all，local，hostip]，默认值为hostip
EnableResourceReserved	Boolean	否	是否开启资源预留
kubeReserved	Map<String, String>	否	资源预留配额，例如 { cpu: 100m, memory: 1000Mi }
enableCordon	Boolean	否	是否封锁节点
preUserScript	String	否	部署前执行脚本, 前端 base64编码后传参
postUserScript	String	否	部署后执行脚本, 前端 base64编码后传参

DockerConfig

参数名称	类型	是否必须	描述
dockerDataRoot	String	否	自定义 docker 数据目录
registryMirrors	List<String>	否	自定义 RegistryMirrors
insecureRegistries	List<String>	否	自定义 InsecureRegistries
dockerLogMaxSize	String	否	docker日志大小，默认值为 20m
dockerLogMaxFile	String	否	docker日志保留数，默认值为 10
dockerBIP	String	否	docker0网桥网段，默认值为 169.254.30.1/28

ContainerdConfig

参数名称	类型	是否必须	描述
dataRoot	String	否	自定义 containerd 数据目录
registryMirrors	List<String>	否	自定义 RegistryMirrors
insecureRegistries	List<String>	否	自定义 InsecureRegistries

Tag

参数名称	类型	是否必须	描述
tagKey	String	是	Tag Key
tagValue	String	是	Tag Value

Taint

参数名称	类型	是否必须	描述
effect	String	是	当Pod不容忍 Taint 时的行为，可选 [NoSchedule, PreferNoSchedule, NoExecute]
key	String	是	Taint Key
timeAdded	Date-time	否	添加污点的时间点，只有effect为NoExecute时使用
value	String	是	Taint Value

BidOption

参数名称	类型	是否必须	描述
bidMode	String	是	竞价模式，可取值：MARKET_PRICE_BID (随市场价出价)，(用户主动出价)
bidPrice	String	否	竞价实例出价，bidMode=MARKET_PRICE_BID为必填项
bidTimeout	Integer	是	竞价超时（单位：分钟），当超过此时间仍未竞价成功将自动取消订单并清理实例
bidReleaseEIP	Boolean	否	竞价实例被抢占释放时是否级联删除 EIP，默认为 false
bidReleaseCDS	Boolean	否	竞价实例被抢占释放时是否级联删除 CDS，默认为 false

ClusterPage

参数名称	类型	描述
keywordType	String	集群模糊查询字段，可选 [clusterName, clusterID]
keyword	String	查询关键词
orderBy	String	集群查询排序字段，可选 [clusterName, clusterID, createdAt]
order	String	排序方式，可选 [ASC, DESC]
pageNo	Integer	页码
pageSize	Integer	单页结果数
totalCount	Integer	集群总数量
clusterList	List<Cluster>	查询到的集群列表

Cluster

参数名称	类型	描述
spec	ClusterSpec	集群属性
status	ClusterStatus	集群状态
createdAt	String	创建时间
updatedAt	String	更新时间

ClusterStatus

参数名称	类型	描述
clusterBLB	BLB	集群的BLB
clusterPhase	String	集群状态，可选 [pending, provisioning, provisioned, running, create_failed, deleting, deleted, delete_failed]
nodeNum	Integer	节点数量

BLB

参数名称	类型	描述
id	String	BLB ID
vpcIP	String	VPC IP 地址
eip	String	EIP 地址

VKSubnetType

参数名称	类型	是否必须	描述
availableZone	String	是	可用区名称
subnetID	String	是	子网 ID

InstancePage

参数名称	类型	描述
clusterID	String	集群的ID
keywordType	String	集群模糊查询字段，可选 [clusterName, clusterID]
keyword	String	查询关键词
orderBy	String	集群查询排序字段，可选 [clusterName, clusterID, createdAt]
order	String	排序方式，可选 [ASC, DESC]
pageNo	Integer	页码
pageSize	Integer	单页结果数
totalCount	Integer	节点总数
instanceList	List<Instance>	节点列表

Instance

参数名称	类型	描述
createdAt	String	节点创建时间
spec	InstanceSpec	节点的配置
status	InstanceStatus	节点的状态
updatedAt	String	节点更新时间

InstanceStatus

参数名称	类型	描述
instancePhase	String	节点的状态，可选 [pending, provisioning, provisioned, running, create_failed, deleting, deleted, delete_failed] 。pending, provisioning, provisioned 均表示创建中状态。
machine	Machine	虚拟机信息
machineStatus	String	BBC虚机状态，可选 [ACTIVE, BUILD, REBUILD, DELETED, SNAPSHOT, DELETE_SNAPSHOT, VOLUME_RESIZE, ERROR, EXPIRED, REBOOT, RECHARGE, SHUTOFF, STOPPED, UNKNOWN]

Machine

参数名称	类型	描述
eip	String	EIP IP地址
instanceID	String	对应节点ID
instanceName	String	对应节点名称
mountList	List<MountConfig>	磁盘挂载信息列表
orderID	String	订单号
vpcIP	String	VPC IP 地址
vpcIPIPv6	String	VPC IPv6地址
k8sNodeName	String	K8S NodeName, 使用 IP 或 Hostname

MountConfig

参数名称	类型	描述
cdsID	String	CDS磁盘ID
cdsSize	Integer	CDS磁盘大小
device	String	设备路径. 如/dev/vdb
diskPath	String	磁盘路径. 如/data
storageType	String	磁盘存储类型，可选 [hp1, cloud_hp1, hdd, local, sata, ssd]. 更多详情参考：CDS磁盘性能 与 CDS磁盘类型参数

ListInstancesByInstanceGroupIDPage

参数名称	类型	描述
pageNo	Integer	页码
pageSize	Integer	单页结果数
totalCount	Integer	节点总数
list	List<Instance>	节点列表

InstanceGroup

参数名称	类型	描述
spec	InstanceGroupSpec	节点组的配置
status	InstanceGroupStatus	节点组的状态
createdAt	String	节点组的创建时间

InstanceGroupSpec

参数名称	类型	描述
cceInstanceGroupID	String	节点组 ID
instanceGroupName	String	节点组名称
clusterID	String	集群ID
clusterRole	String	节点在集群中的角色，目前节点组仅支持nod. 默认值为 node
shrinkPolicy	String	节点组收缩规则. 可选 [Priority, Priority].
updatePolicy	String	节点组更新规则. 可选 [Rolling, Concurrency].
cleanPolicy	String	节点清理规则. 可选 [Remain, Delete].
instanceTemplate	InstanceTemplate	节点配置
replicas	Integer	节点副本数
clusterAutoscalerSpec	ClusterAutoscalerSpec	集群的自动伸缩配置

InstanceTemplate

此结构等同于InstanceSpec，参见InstanceSpec

ClusterAutoscalerSpec

参数名称	类型	描述
enabled	Boolean	是否启用Autoscaler
minReplicas	Integer	最小副本数. 取值范围是自然数集.
maxReplicas	Integer	最大副本数. 取值范围是自然数集, 需大于minReplicas.
scalingGroupPriority	Integer	伸缩组优先级. 取值范围是自然数集.

InstanceGroupStatus

参数名称	类型	描述
readyReplicas	Integer	节点组中处于 Ready 状态的节点数
pause	Pause	节点组的暂停状态

Pause

参数名称	类型	描述
paused	Boolean	节点组是否处于暂停状态
reason	String	节点组处于暂停状态的原因

ListInstanceGroupPage

参数名称	类型	描述
pageNo	Integer	页码
pageSize	Integer	单页结果数
totalCount	Integer	节点组总数
list	List<InstanceGroup>	节点组列表

Autoscaler

参数名称	类型	描述
clusterID	String	集群 ID
clusterName	String	集群名称
caConfig	ClusterAutoscalerConfig	节点组列表

ClusterAutoscalerConfig

参数名称	类型	描述
expander	String	自动扩缩容选组的策略. 可选 [random, most-pods, least-waste, priority], 默认值为 random.
instanceGroups	List<ClusterAutoscalerInstanceGroup>	节点组的 Autoscaler 配置. 用户无需输入此项内容.
kubeVersion	String	K8S 版本. 为空时后台会自动查询集群K8S版本号.
maxEmptyBulkDelete	Integer	最大并发缩容数
replicaCount	Integer	预期副本数量
scaleDownDelayAfterAdd	Integer	扩容后缩容启动时延, 单位为分钟
scaleDownEnabled	Boolean	是否启动缩容. 默认值为false
scaleDownGPUUtilizationThreshold	Integer	GPU缩容阈值百分比, 取值范围（0, 100）.
scaleDownUnneededTime	Integer	缩容触发时延，单位为分钟.
scaleDownUtilizationThreshold	Integer	缩容阈值百分比, 取值范围（0, 100）.
skipNodesWithLocalStorage	Boolean	是否跳过使用本地存储的节点, 默认值为 true.
skipNodesWithSystemPods	Boolean	是否跳过有部署系统 Pod 的节点, 默认值为 true.

ClusterAutoscalerInstanceGroup

参数名称	类型	描述
instanceGroupID	String	节点组 ID
minReplicas	String	最小副本数
maxReplicas	String	最大副本数
priority	String	优先级

ContainerCIDRConflict

参数名称	类型	描述
conflictCluster	ConflictCluster	与容器网段冲突的VPC内集群，当且仅当 NetworkConflictType 为 ContainerCIDRAndExistedClusterContainerCIDRConflict 不为 nil
conflictNodeCIDR	ConflictNodeCIDR	与容器网段冲突的节点网段，当且仅当 NetworkConflictType 为 ContainerCIDRAndNodeCIDRConflict 不为 nil
conflictType	String	网络冲突类型，可选 [ContainerCIDRAndNodeCIDR, ContainerCIDRAndExistedClusterContainerCIDR, ContainerCIDRAndVPCRoute, ClusterIPCIDRAndNodeCIDR, ClusterIPCIDRAndContainerCIDR]
conflictVPCRoute	ConflictVPCRoute	与容器网段冲突的VPC路由，当且仅当 NetworkConflictType 为 ContainerCIDRAndVPCRouteConflict 不为 nil

ClusterIPCIDRConflict

参数名称	类型	描述
conflictContainerCIDR	ConflictContainerCIDR	容器网段冲突信息
conflictNodeCIDR	ConflictNodeCIDR	节点网段冲突信息
conflictType	String	网络冲突类型，可选 [ContainerCIDRAndNodeCIDR, ContainerCIDRAndExistedClusterContainerCIDR, ContainerCIDRAndVPCRoute, ClusterIPCIDRAndNodeCIDR, ClusterIPCIDRAndContainerCIDR]

ConflictCluster

参数名称	类型	描述
clusterID	String	集群ID
containerCIDR	String	冲突的容器网段

ConflictContainerCIDR

参数名称	类型	描述
containerCIDR	String	冲突的容器网段

ConflictNodeCIDR

参数名称	类型	描述
nodeCIDR	String	冲突的节点网段

ConflictVPCRoute

参数名称	类型	描述
routeRule	RouteRule	冲突的 VPC 路由

RouteRule

参数名称	类型	描述
routeRuleId	String	路由规则 ID
routeTableId	String	路由表 ID
sourceAddress	String	源地址
destinationAddress	String	目的地址
nexthopId	String	下一跳 ID
nexthopType	String	下一跳类型
description	String	描述

ListTaskPage

参数名称	类型	描述
pageNo	Integer	页码
pageSize	Integer	单页结果数
totalCount	Integer	任务总数
items	List<Task>	任务列表

Task

参数名称	类型	描述
id	String	任务 ID
type	String	任务类型
description	String	任务描述
startTime	String	任务开始时间
finishTime	String	任务结束时间
phase	String	任务所处阶段，可能取值[Pending, Processing, Done, Aborted, Collecting]
processes	List<TaskProcess>	任务进展列表
errMessage	String	任务错误信息

TaskProcess

参数名称	类型	描述
name	String	任务进展名称
phase	String	任务进展所处阶段，可能取值[Pending, Processing, Done, Aborted]
startTime	String	任务进展开始时间
finishTime	String	任务进展结束时间
metrics	Map<String, String>	任务进展采集项，根据不同任务有不同的采集项
subProcesses	List<TaskProcess>	任务进展的子进展列表
errMessage	String	错误信息

Step

参数名称	类型	描述
stepName	String	步骤名称
stepStatus	String	步骤状态，可选 [todo, doing, paused, done, failed]
ready	Boolean	是否准备就绪
startTime	String	步骤开始时间
finishedTime	String	步骤结束时间
costSeconds	integer	步骤花费时间
retryCount	integer	重试次数
errorInfo	ReconcileResponse	失败详情

ReconcileResponse

参数名称	类型	描述
code	string	响应码
message	String	响应信息
traceID	String	请求 ID, 问题定位提供该 ID
suggestion	integer	建议信息

AddOnInfo

参数名称	类型	描述
meta	Meta	组件基础信息
instance	AddOnInstance	组件安装信息。如果组件未安装，该字段为空值。
multiInstances	List<AddOnInstance>	如果组件允许多实例部署，则为该组件全部部署实例。通常不会使用到该字段。

Meta

参数名称	类型	描述
name	String	组件名称
type	String	组件类型。包括CloudNativeAI、Networking、HybridSchedule、Image、Storage、Observability
latestVersion	String	组件的最新版本
shortIntroduction	String	组件简介
defaultParams	String	组件默认部署参数
installInfo	InstallInfo	组件是否可以安装

AddOnInstance

参数名称	类型	描述
name	String	组件名称
installedVersion	String	已安装组件的版本
params	String	组件的部署参数
status	AddonInstanceStatus	组件状态
uninstallInfo	UninstallInfo	组件是否允许卸载
upgradeInfo	UpgradeInfo	组件是否允许升级
updateInfo	UpdateInfo	组件是否允许更新部署参数

AddonInstanceStatus

参数名称	类型	描述
phase	String	组件当前的状态
code	String	组件状态异常时，其错误码内容
message	String	组件状态异常时，其错误详情
traceID	String	组件状态异常时，其故障ID

InstallInfo

参数名称	类型	描述
allowInstall	String	组件是否允许安装
message	String	如果不允许安装，该字段为原因

UninstallInfo

参数名称	类型	描述
allowUninstall	String	组件是否允许卸载
message	String	如果不允许卸载，该字段为原因

UpgradeInfo

参数名称	类型	描述
allowUpgrade	String	组件是否允许升级
nextVersion	String	如果允许升级，其目标升级版本
message	String	如果不允许升级，该字段为原因

ResourceChargingOption

参数名称	类型	描述
ChargingType	String	后付费或预付费
PurchaseTime	int	预付费才生效：单位月，12 = 12 月
PurchaseTimeUnit	String	预付费时间单位
AutoRenew	bool	是否自动续费
AutoRenewTime	String	自动续费时长，12 = 12 个月
AutoRenewTimeUnit	String	续费单位：月

PluginHelmConfig

参数名称	类型	描述
PluginType	String	插件类型(插件名称)，非必要
PluginName	String	插件别名，非必要
ChartVersion	String	使用的Chart版本，非必要
ChartName	string	插件在云端的ChartName，非必要
Namespaces	String	插件部署到哪个命名空间，非必要
Description	String	描述，非必要
Values	String	取决于插件，系统插件传空值即可

MachineSpecStatus

参数名称	类型	描述
machineSpec	String	规格套餐
status	String	套餐状态，Adapted(已适配), Unadapted(暂未适配), NotSupported(不支持)

ExistedInstanceInCluster

参数名称	类型	描述
existedInstanceID	String	bcc 实例 id

RBAC相关接口

授权

描述

给指定用户授权集群的 RBAC 权限

请求结构

```
 POST /v2/rbac HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	RequestBody 参数	待授权的集群 ID。特殊值 `all`，表示授权所有已有集群。
userID	String	是	RequestBody 参数	待授权的用户 ID。
namespace	String	否	RequestBody 参数	待授权的命名空间。仅当 clusterID 不为空或 `all` 时有用。特殊值 `all` 和 ""，表示授权所有命名空间。
role	String	是	RequestBody 参数	授予的 RBAC 角色，取值： • `cce:admin` 管理员 • `cce:devops` 运维开发 • `cce:readonly` 只读

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
data	List<CreateRBACMessage>	是	每个集群的授权结果
requestID	String	是	请求 ID，问题定位提供该 ID

CreateRBACMessage

参数名称	类型	是否必须	描述
success	Boolean	是	是否成功
clusterID	String	是	集群 ID
message	String	否	失败时的提示信息

请求示例: 授权所有已有集群

- `clusterID` 设为 `all`
- 无需设置 `namespace` 参数
- 其它参数参考 API 文档按需设置

```
 POST /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "clusterID": "all",
    "userID": "60a039xxxxxxxxxxxxxxx",
    "role": "cce:readonly"
}
```

请求示例: 授权指定集群所有命名空间

- `clusterID` 设为指定集群 ID
- `namespace` 设为 `all`
- 其它参数参考 API 文档按需设置

```
 POST /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "clusterID": "cce-xxxxxxx",
    "userID": "60a039xxxxxxxxxxxxxxx",
    "namespace": "all",
    "role": "cce:devops"
}
```

请求示例: 授权指定集群指定命名空间

- `clusterID` 设为指定集群 ID
- `namespace` 设为指定命名空间
- 其它参数参考 API 文档按需设置

```
 POST /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "clusterID": "cce-xxxxxxx",
    "userID": "60a039xxxxxxxxxxxxxxx",
    "namespace": "default",
    "role": "cce:admin"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: d2ce8f50-529a-4663-9265-ad08c94633c8
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
{
    "requestID": "d2ce8f50-529a-4663-9265-ad08c94633c8",
    "data": [
        {
            "success": true,
            "clusterID": "cce-6m8tij7z"
        }
    ]
}
```

RBAC 列表

描述

查询用户的 RBAC 列表

请求结构

```
 GET /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
userID	String	是	Query 参数	用户 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	描述	描述
data	List<GetRBACMessage>	是	RBAC 查询结果
requestID	String	是	请求 ID, 问题定位提供该 ID

GetRBACMessage

参数名称	类型	是否必须	描述
role	String	是	RBAC 角色
clusterID	String	是	集群 ID
namespace	String	是	命名空间，特殊取值 `all`
clusterName	String	否	集群名字

请求示例

```
 GET /v2/rbac?userID=60a039xxxxxxxxxxxxxxx HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 97342dc7-29a1-4ed9-a75d-904bb293d295
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "97342dc7-29a1-4ed9-a75d-904bb293d295",
 "data": [{
   "role": "cce:readonly",
   "clusterID": "cce-aaaaaa",
   "namespace": "all",
   "clusterName": "test-resource"
  },
  {
   "role": "cce:readonly",
   "clusterID": "cce-bbbbbb",
   "namespace": "test-ns",
   "clusterName": "test04"
  }
 ]
}
```

删除 RBAC 权限

描述

删除用户的 RBAC 权限

请求结构

```
 DELETE /v2/rbac HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	RequestBody 参数	集群 ID，必须是具体的集群 ID，不能是 all。
userID	String	是	RequestBody 参数	用户 ID
namespace	String	是	RequestBody 参数	命名空间，需跟授权时保持一致，如果授权时是 all，则必须也是 all。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 DELETE /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "clusterID": "cce-xxxxxxx",
    "userID": "60a039xxxxxxxxxxxxxxx",
    "namespace": "default"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 928a21b5-d117-4a83-a274-fd7d6f413524
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
 "requestID": "928a21b5-d117-4a83-a274-fd7d6f413524"
}
```

重置 RBAC Kubeconfig

描述

失效用户旧的 RBAC Kubeconfig，生成新的 Kubeconfig。注意：执行该方法后，旧的 RBAC Kubeconfig 不可用。

请求结构

```
 PUT /v2/rbac HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	RequestBody 参数	集群 ID，必须是具体的集群 ID，不能是 all。
userID	String	否	RequestBody 参数	用户 ID，未填写时表示当前用户
namespace	String	是	RequestBody 参数	命名空间，需跟授权时保持一致，如果授权时是 all，则必须也是 all。
role	String	是	RequestBody 参数	角色，需跟授权时保持一致。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
requestID	String	是	请求 ID, 问题定位提供该 ID

请求示例

```
 PUT /v2/rbac HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: 105ce04b-1a42-4f77-9d22-ab6f413f9d69
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8

{
    "requestID": "105ce04b-1a42-4f77-9d22-ab6f413f9d69"
}
```

申请临时访问凭证

描述

申请指定集群的临时凭证，可指定凭证生效时间

请求结构

```
 POST /v2/rbac HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
clusterID	String	是	RequestBody 参数	待授权的集群 ID。特殊值 `all`，表示授权所有已有集群。
temp	Boolean	是	RequestBody 参数	必须是 `true`，说明需要申请临时访问凭证。
expireHours	Integer	是	RequestBody 参数	临时凭证生效时间，单位是`小时`
kubeConfigType	String	是	RequestBody 参数	临时凭证的类型，取值： • `vpc` vpc 访问凭证 • `public` 公网访问凭证

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
temporaryKubeConfig	String	是	临时凭证 Kubeconfig
requestID	String	是	请求 ID，问题定位提供该 ID

请求示例: 申请 7天的 vpc 临时访问凭证

```
 POST /v2/rbac  HTTP/1.1
Host: cce.bj.baidubce.com
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2019-03-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "clusterID": "cce-xxxx",
    "temp": true,
    "expireHours": 168,
    "kubeConfigType": "vpc"
}
```

返回示例

```
 HTTP/1.1 200 OK
X-Bce-Request-Id: d2ce8f50-529a-4663-9265-ad08c94633c8
Date: Thu, 16 Mar 2020 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
{
    "requestID": "d2ce8f50-529a-4663-9265-ad08c94633c8",
    "temporaryKubeConfig": "apiVersion: v1\nclusters:\n- cluster:\n    certificate-authority-data: xxxx....."
}
```

套餐适配相关接口

套餐适配相关接口

获取套餐列表

获取 cce 保存的套餐列表，及适配状态。

请求结构

```
 POST /api/cce/artifact-service/v1/machine-specs
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
type	String	否	Query 参数	套餐类型，common(普通套餐) 或 gpu(异构套餐)
machineSpecList	List<String>	否	RequestBody 参数	查询指定套餐

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	是否必须	描述
machineSpecList	List<MachineSpecStatus>	是	套餐列表

请求示例：查询套餐列表

```
 POST /api/cce/artifact-service/v1/machine-specs
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求示例：查询异构套餐列表

```
 POST /api/cce/artifact-service/v1/machine-specs?type=gpu
Host: cce.bj.baidubce.com
Authorization: authorization string
```

请求示例：查询普通套餐列表

```
 POST /api/cce/artifact-service/v1/machine-specs?type=common
Host: cce.bj.baidubce.com
Authorization: authorization string

{
    "machineSpecList": [
        "bcc.c3ne.c1m2"
    ]
}
```

请求示例：查询指定套餐

```
POST /api/cce/artifact-service/v1/machine-specs?type=common
Host: cce.bj.baidubce.com
Authorization: authorization string
```

返回示例

```
{
  "machineSpecList": [
    {
      "machineSpec": "bcc.c3ne.c1m2",
      "status": "Adapted"
    },
    {
      "machineSpec": "bcc.c3ne.c2m4",
      "status": "Adapted"
    }
    ...
}
```

SDK

Go-SDK

概述

本文档主要介绍CCE GO SDK的安装和使用。在使用本文档前，您需要先了解CCE的一些基本知识，并已经开通了CCE服务。若您还不了解CCE，可以参考产品描述。

安装SDK工具包

安装SDK工具包

运行环境

GO SDK可以在go1.3及以上环境下运行。

安装SDK

直接从github下载

使用go get工具从github进行下载：

```
go get github.com/baidubce/bce-sdk-go
```

SDK目录结构

```
bce-sdk-go
    ├─── auth                       //BCE签名相关类
    ├─── http                       //BCE的Http通信相关类
    ├─── services
    │      └─── cce                  //CCE 服务相关类
    │           ├─── v2              //CCE V2 接口
    ├─── util                       //BCE公用工具类
    ├─── examples
          └─── cce                  //CCE 各种对象 SDK 使用示例
```

卸载SDK

卸载SDK时，删除下载的源码即可。

初始化

使用步骤

确认Endpoint

在使用SDK之前，需确认您将接入的百度云产品的Endpoint（服务域名）。可阅读CCE 访问域名，理解Endpoint相关的概念。

创建Client对象

每种具体的服务都有一个Client对象，为开发者与对应的服务进行交互封装了一系列易用的方法。

调用功能接口

开发者基于创建的对应服务的Client对象，即可调用相应的功能接口，使用百度云产品的功能。

示例

下面给出一个基本的使用示例。

```
 package main

import (
    "fmt"

 cceapi "github.com/baidubce/bce-sdk-go/services/cce/v2"
)

func main() {
 // 用户的Access Key ID和Secret Access Key
 accessKeyID, secretAccessKey := "<your-access-key-id>", "<your-secret-access-key>"

 // CCE服务的Endpoint
 endpoint := "cce.bj.baidubce.com"

 // 创建CCE服务的Client
 cceClient, err := cceapi.NewClient(accessKeyID, secretAccessKey, endpoint)

 // 获取集群列表
    args := &cceapi.ListClustersArgs{
  KeywordType: "",
  Keyword:     "",
  OrderBy:     "",
  Order:       "",
  PageSize:    10,
  PageNum:     1,
 }
 result, err := cceClient.ListClusters(args)
 if err != nil {
  fmt.Println("list cluster failed:", err)
 }
 fmt.Println("requestID: ", result.RequestID)
}
```

配置

使用HTTPS协议

该SDK支持使用HTTPS协议访问百度云的服务产品。要使用HTTPS协议，只需在您创建对应服务的Client对象时指定的Endpoint中指明使用HTTPS协议的域名即可，SDK会自动识别并使用HTTPS协议访问。

详细配置

开发者使用GO SDK时，创建的对应服务的Client对象，其导出字段Config提供如下参数以便支持详细配置：

配置项名称	类型	含义
Endpoint	string	请求服务的域名
ProxyUrl	string	客户端请求的代理地址
Region	string	请求资源的区域
UserAgent	string	用户名称，HTTP请求的User-Agent头
Credentials	*auth.BceCredentials	请求的鉴权对象，分为普通AK/SK与STS两种
SignOption	*auth.SignOptions	认证字符串签名选项
Retry	RetryPolicy	连接重试策略
ConnectionTimeoutInMillis	int	连接超时时间，单位毫秒，默认20分钟

开发者可据此进行详细参数的配置，下面给出部分配置示例：

```
// client为某一种具体服务的`Client`对象

// 配置请求代理地址
client.Config.ProxyUrl = "127.0.0.1:8080"

// 配置不进行重试，默认为Back Off重试
client.Config.Retry = bce.NewNoRetryPolicy()

// 配置连接超时时间为30秒
client.Config.ConnectionTimeoutInMillis = 30 * 1000

// 配置签名使用的HTTP请求头为`Host`
client.Config.SignOption.HeadersToSign = map[string]struct{}{"Host": struct{}{}}

// 配置签名的有效期为30秒
client.Config.SignOption.ExpireSeconds = 30
```

说明：

1. Credentials字段使用auth.NewBceCredentials与auth.NewSessionBceCredentials函数创建，默认使用前者，后者为使用STS鉴权时使用。
2. SignOption字段为生成签名字符串时的选项，详见下表说明：

名称	类型	含义
HeadersToSign	map[string]struct{}	生成签名字符串时使用的HTTP头
Timestamp	int64	生成的签名字符串中使用的时间戳，默认使用请求发送时的值
ExpireSeconds	int	签名字符串的有效期

> 其中，HeadersToSign默认为Host，Content-Type，Content-Length，Content-MD5；TimeStamp一般为零值，表示使用调用生成认证字符串时的时间戳，单位为秒，用户一般不应该明确指定该字段的值；ExpireSeconds默认为1800秒即30分钟。

3. Retry字段指定重试策略，目前支持两种：NoRetryPolicy和BackOffRetryPolicy。默认使用后者，该重试策略是指定最大重试次数、最长重试时间和重试基数，按照重试基数乘以2的指数级增长的方式进行重试，直到达到最大重试次数或者最长重试时间为止。

集群管理

创建集群

```
func CreateCluster() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
  panic(err)
}
args := &v2.CreateClusterArgs{
  CreateClusterRequest: &v2.CreateClusterRequest{
    ClusterSpec: &types.ClusterSpec{
      ClusterName: "your-cluster-name",
      K8SVersion:  "1.21.14",
      RuntimeType: types.RuntimeTypeDocker,
      VPCID:       "vpc-id",
      MasterConfig: types.MasterConfig{
        MasterType:           types.MasterTypeManaged,
        ClusterHA:            1,
        ExposedPublic:        false,
        ClusterBLBVPCSubnetID: "cluster-blb-vpc-subnet-id",
        ManagedClusterMasterOption: types.ManagedClusterMasterOption{
          MasterVPCSubnetZone: types.AvailableZoneA,
        },
      },
```

```
        },
        ContainerNetworkConfig: types.ContainerNetworkConfig{
          Mode:                 types.ContainerNetworkModeKubenet,
          LBServiceVPCSubnetID: "lb-service-vpc-subnet-id",
          ClusterPodCIDR:       "172.28.0.0/16",
          ClusterIPServiceCIDR: "172.31.0.0/16",
        },
        Tags: types.TagList{
          {
            TagKey:   "tagkey",
            TagValue: "tagvalue",
          },
        },
      },
      NodeSpecs: []*v2.InstanceSet{
        {
          Count: 1,
          InstanceSpec: types.InstanceSpec{
            InstanceName: "",
            ClusterRole:  types.ClusterRoleNode,
            Existed:      false,
            MachineType:  types.MachineTypeBCC,
            InstanceType: api.InstanceTypeN3,
            VPCConfig: types.VPCConfig{
              VPCID:           "",
              VPCSubnetID:     "vpc-subnet-id",
              SecurityGroupID: "security-group-id",
              AvailableZone:   types.AvailableZoneA,
            },
            InstanceResource: types.InstanceResource{
              CPU:           4,
              MEM:           8,
              RootDiskSize:  40,
              LocalDiskSize: 0,
              CDSList:       []types.CDSConfig{},
            },
            ImageID: "image-id",
            InstanceOS: types.InstanceOS{
              ImageType: api.ImageTypeSystem,
            },
            NeedEIP:              false,
            AdminPassword:        "admin-password",
            SSHKeyID:             "ssh-key-id",
            InstanceChargingType: api.PaymentTimingPostPaid,
            RuntimeType:          types.RuntimeTypeDocker,
            RelationTag:          true,
          },
        },
      },
    },
  }

  resp, err := ccev2Client.CreateCluster(args)
  if err != nil {
    fmt.Println(err.Error())
    return
  }
  s, _ := json.MarshalIndent(resp, "", "\t")
  fmt.Println("Response:" + string(s))
}
```

创建集群并关联标签

```
  args := &v2.CreateClusterArgs{
 CreateClusterRequest: &v2.CreateClusterRequest{
  ClusterSpec: &types.ClusterSpec{
   // 创建集群的其他配置

   // 集群关联的标签
   Tags: types.TagList{
    {
     TagKey:   "tagkey",
     TagValue: "tagvalue",
    },
   },
  },
  NodeSpecs: []*v2.InstanceSet{
   {
    // 创建节点实例的其他配置

    // 节点同时关联到上述标签
    InstanceSpec: types.InstanceSpec{
     RelationTag: true,
    },
   },
  },
 },
}
```

查询集群

```
 func GetCluster() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
clusterID := ""
resp, err := ccev2Client.GetCluster(clusterID)
if err != nil {
 fmt.Println(err.Error())
 return
}
s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

查询集群标签

```
    resp, err := ccev2Client.GetCluster(clusterID)
if err != nil {
 fmt.Println(err.Error())
 return
}

tags, _ := json.MarshalIndent(resp.Cluster.Spec.Tags, "", "\t")
fmt.Println("Response:" + string(tags))
```

查询集群列表

```
func ListClusters() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := ccev2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
args := &ccev2.ListClustersArgs{
 KeywordType: "clusterName",
 Keyword:     "",
 OrderBy:     "clusterID",
 Order:       ccev2.OrderASC,
 PageSize:    10,
 PageNum:     1,
}
resp, err := ccev2Client.ListClusters(args)
if err != nil {
 fmt.Println(err.Error())
 return
}
s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

删除集群

```
func DeleteCluster() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := ccev2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
clusterID := ""
args := &ccev2.DeleteClusterArgs{
 ClusterID:          clusterID,
 DeleteResource:     true,
 DeleteCDSSnapshot: true,
}

resp, err := ccev2Client.DeleteCluster(args)
if err != nil {
 fmt.Println(err.Error())
 return
}

s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

更新集群删除保护

```
func UpdateClusterForbidDelete() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := ccev2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
clusterID := ""
args := &ccev2.UpdateClusterForbidDeleteArgs{
 ClusterID: clusterID,
 UpdateClusterForbidDeleteRequest: ccev2.UpdateClusterForbidDeleteRequest{
  ForbidDelete: true,
 },
}

resp, err := ccev2Client.UpdateClusterForbidDelete(args)
if err != nil {
 fmt.Println(err.Error())
 return
}

s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

获取集群事件步骤

```
func GetClusterEventSteps() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
clusterID := "cluster-id"
resp, err := ccev2Client.GetClusterEventSteps(clusterID)
if err != nil {
 fmt.Println(err.Error())
 return
}
s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

节点管理

创建节点

```
func CreateInstance() {
// 用户的Access Key ID 和 Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
```

```
 }
 resp, err := ccev2Client.CreateInstances(&v2.CreateInstancesArgs{
  ClusterID: "",
  Instances: []*v2.InstanceSet{
   {
    InstanceSpec: types.InstanceSpec{
     MachineType:  types.MachineTypeBCC,
     InstanceType: api.InstanceType34,
     InstanceName: "",
     VPCConfig: types.VPCConfig{
      VPCSubnetID: "sbn-xxx",
      SecurityGroup: types.SecurityGroup{
       EnableCCERequiredSecurityGroup: true,
       EnableCCEOptionalSecurityGroup: false,
      },
     },
     InstanceResource: types.InstanceResource{
      CPU:           2,
      MEM:           8,
      RootDiskType:  api.StorageTypeHP1,
      RootDiskSize:  100,
      LocalDiskSize: 0,
      CDSList:       []types.CDSConfig{},
      MachineSpec:   "bcc.g5.c2m8",
     },
     ImageID: "7183d3d0-3e24-464d-9c02-xxxx",
     InstanceOS: types.InstanceOS{
      ImageType: api.ImageTypeSystem,
     },
     SSHKeyID: "k-xxx",
     DeployCustomConfig: types.DeployCustomConfig{
      KubeletRootDir: "/var/lib/kubelet",
      KubeReserved: map[string]string{
       "cpu":    "50m",
       "memory": "100Mi",
      },
      SystemReserved: map[string]string{
       "cpu":    "50m",
       "memory": "100Mi",
      },
      ContainerdConfig: types.ContainerdConfig{
       DataRoot: "/home/cce/containerd",
      },
     },
     RelationTag:          true,
     InstanceChargingType: api.PaymentTimingPostPaid,
     Tags: types.TagList{
      {
       TagKey:   "tagkey",
       TagValue: "tagvalue",
      },
     },
    },
    Count: 1,
   },
  },
 })
 if err != nil {
  fmt.Println(err.Error())
  return
 }
 s, _ := json.MarshalIndent(resp, "", "\t")
 fmt.Println("Response:" + string(s))
}
```

创建节点并关联标签

```
    resp, err := ccev2Client.CreateInstances(&v2.CreateInstancesArgs{
 ClusterID: "",
 Instances: []*v2.InstanceSet{
  {
   InstanceSpec: types.InstanceSpec{
                // 创建节点所需的其他配置

                // 节点关联的标签
    Tags: types.TagList{
     {
      TagKey:   "tagkey",
      TagValue: "tagvalue",
     },
    },
   },
   Count: 1,
  },
 },
})
```

查询节点

```
 func GetInstance(){
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}

resp, err := ccev2Client.GetInstance(&v2.GetInstanceArgs{
 ClusterID:  "",
 InstanceID: "",
})
if err != nil {
 fmt.Println(err.Error())
 return
}

s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

查询节点标签

```
    instance, err := ccev2Client.GetInstance(&v2.GetInstanceArgs{
 ClusterID:  "",
 InstanceID: "",
})
if err != nil {
 fmt.Println(err.Error())
 return
}

tags, _ := json.MarshalIndent(instance.Instance.Spec.Tags, "", "\t")
fmt.Println("Response:" + string(tags))
```

节点组管理

创建节点组

```
 func CreateInstanceGroup() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""
```

```
// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}
resp, err := ccev2Client.CreateInstanceGroup(&v2.CreateInstanceGroupArgs{
 ClusterID: "",
 Request: &v2.CreateInstanceGroupRequest{
  InstanceGroupSpec: types.InstanceGroupSpec{
   InstanceGroupName: "",
   ClusterID:         "",

   InstanceTemplate: types.InstanceTemplate{
    InstanceSpec: types.InstanceSpec{
     MachineType:  types.MachineTypeBCC,
     InstanceType: api.InstanceType34,
     InstanceName: "",
     VPCConfig: types.VPCConfig{
      VPCSubnetID: "sbn-xxx",
      SecurityGroup: types.SecurityGroup{
       EnableCCERequiredSecurityGroup: true,
       EnableCCEOptionalSecurityGroup: false,
      },
     },
     InstanceResource: types.InstanceResource{
      CPU:           2,
      MEM:           8,
      RootDiskType:  api.StorageTypeHP1,
      RootDiskSize:  100,
      LocalDiskSize: 0,
      CDSList:       []types.CDSConfig{},
      MachineSpec:   "bcc.g5.c2m8",
     },
     ImageID: "7183d3d0-3e24-464d-9c02-xxxxxxx",
     InstanceOS: types.InstanceOS{
      ImageType: api.ImageTypeSystem,
     },
     SSHKeyID: "k-xxxx",
     DeployCustomConfig: types.DeployCustomConfig{
      KubeletRootDir: "/var/lib/kubelet",
      KubeReserved: map[string]string{
       "cpu":    "50m",
       "memory": "100Mi",
      },
      SystemReserved: map[string]string{
       "cpu":    "50m",
       "memory": "100Mi",
      },
      ContainerdConfig: types.ContainerdConfig{
       DataRoot: "/home/cce/containerd",
      },
     },
     RelationTag:          true,
     InstanceChargingType: api.PaymentTimingPostPaid,
     Tags: types.TagList{
      {
       TagKey:   "tagkey",
       TagValue: "tagvalue",
      },
     },
    },
   },
   Replicas: 1,
  },
 },
})
if err != nil {
```

```
 fmt.Println(err.Error())
 return
}
s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
}
```

创建节点组并关联标签

```
    resp, err := ccev2Client.CreateInstanceGroup(&v2.CreateInstanceGroupArgs{
 ClusterID: "",
 Request: &v2.CreateInstanceGroupRequest{
  InstanceGroupSpec: types.InstanceGroupSpec{
   InstanceGroupName: "",
   ClusterID:         "",

   InstanceTemplate: types.InstanceTemplate{
    InstanceSpec: types.InstanceSpec{
                    // 节点组中的实例配置

                    //
     RelationTag: true,
     Tags: types.TagList{
      {
       TagKey:   "tagkey",
       TagValue: "tagvalue",
      },
     },
    },
   },
   Replicas: 1,
  },
 },
})
if err != nil {
 fmt.Println(err.Error())
 return
}
s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))
```

查询节点组

```
func GetInstanceGroup() {
// 用户的Access Key ID和Secret Access Key
AK, SK := "", ""

// 用户指定的endpoint
ENDPOINT := ""

// 初始化一个CCEClient
ccev2Client, err := v2.NewClient(AK, SK, ENDPOINT)
if err != nil {
 panic(err)
}

resp, err := ccev2Client.GetInstanceGroup(&v2.GetInstanceGroupArgs{
 ClusterID:       "",
 InstanceGroupID: "",
})
if err != nil {
 fmt.Println(err.Error())
 return
}

s, _ := json.MarshalIndent(resp, "", "\t")
fmt.Println("Response:" + string(s))

// 节点组标签
tags, _ := json.MarshalIndent(resp.InstanceGroup.Spec.InstanceTemplate.Tags, "", "\t")
fmt.Println("Response:" + string(tags))
}
```

添加已有节点至节点组

```
// TestClient_AttachInstancesToInstanceGroup 添加已有节点至节点组
func TestClient_AttachInstancesToInstanceGroup(t *testing.T) {
var (
 ak              = ""
 sk              = ""
 endpoint        = ""
 clusterID       = ""
 instanceGroupID = ""
 existInstanceID = ""
 AdminPassword   = ""
 // rebuild 是否重装操作系统
 rebuild = true
 // useInstanceGroupConfig 是否使用节点组配置
 useInstanceGroupConfig = true
 // useLocalDiskForContainer 是否使用本地盘保存数据
 useLocalDiskForContainer = false
 imageID                  = ""
)

cceClient, err := NewClient(ak, sk, endpoint)
if err != nil {
 fmt.Printf("NewClient error: %s", err.Error())
 return
}

attachInstanceToInstanceGroupArgs := func() *AttachInstancesToInstanceGroupArgs {
 args := AttachInstancesToInstanceGroupArgs{}
 args.ClusterID = clusterID
 args.InstanceGroupID = instanceGroupID
 args.Request = &AttachInstancesToInstanceGroupRequest{
  ExistedInstances:       make([]*InstanceSet, 0),
  UseInstanceGroupConfig: useInstanceGroupConfig,
 }
 existInstance := &InstanceSet{
  InstanceSpec: types.InstanceSpec{
   Existed: true,
   ExistedOption: types.ExistedOption{
    // bcc 实例 id
```

```
    // bcc 实例 id
    ExistedInstanceID: existInstanceID,
    Rebuild:           &rebuild,
   },
   // 看具体的类型，bcc，bbc，ebc
   MachineType: types.MachineTypeBCC,
   ClusterRole: types.ClusterRoleNode,

   // 二选一
   AdminPassword: AdminPassword,
   SSHKeyID:      "",
  },
 }

 // 如果需要重装操作系统，需要配置这里
 if rebuild {
  existInstance.InstanceSpec.InstanceOS = types.InstanceOS{
   ImageType: bccapi.ImageTypeSystem,
  }
  existInstance.InstanceSpec.ImageID = imageID
 }

 if useLocalDiskForContainer {
  // 将容器数据存储在数据盘或本地盘中
  existInstance.InstanceSpec.InstanceResource = types.InstanceResource{
   CDSList: types.CDSConfigList{
    {
     DataDevice: "/dev/xxx",
     Path:       "/home/cce",
    },
   },
  }
 }
 if !useInstanceGroupConfig {
  existInstance.InstanceSpec.VPCConfig = types.VPCConfig{
   SecurityGroups: []types.SecurityGroupV2{
    {
     ID:   "",
     Name: "",
     Type: types.SecurityGroupTypeNormal,
    },
   },
  }
  existInstance.InstanceSpec.DeployCustomConfig = types.DeployCustomConfig{
   KubeletRootDir: "",
   PreUserScript:  "",
   PostUserScript: "",
  }
  // 标签配置
  existInstance.InstanceSpec.Tags = types.TagList{
   {
    TagKey:   "",
    TagValue: "",
   },
  }
 }
 args.Request.ExistedInstances = append(args.Request.ExistedInstances, existInstance)
 return &args
}()
commonResp, err := cceClient.AttachInstancesToInstanceGroup(attachInstanceToInstanceGroupArgs)
if err != nil {
 fmt.Printf("attach instance to instance group failed, errir: %v", err)
 return
}
fmt.Printf("Request ID: %s", commonResp.RequestID)
}
```

节点组移出已有节点

```
func TestClient_CreateScaleDownInstanceGroupTask(t *testing.T) {
type fields struct {
 ak, sk, endpoint string
}
type args struct {
 args *CreateScaleDownInstanceGroupTaskArgs
}
tests := []struct {
 name   string
 fields fields
 args   args
}{
 {
  name: "移出已有节点",
  fields: fields{
   ak:       "",
   sk:       "",
   endpoint: "",
  },
  args: args{
   args: &CreateScaleDownInstanceGroupTaskArgs{
    InstancesToBeRemoved: []string{""},
    ClusterID:             "",
    InstanceGroupID:       "",
    CleanPolicy:           CleanPolicyDelete,
    DeleteOption: &types.DeleteOption{
     DeleteCDSSnapshot: false,
     DeleteResource:    false,
     DrainNode:         false,
     MoveOut:           true,
    },
   },
  },
 },
}
for _, tt := range tests {
 t.Run(tt.name, func(t *testing.T) {
  c, err := NewClient(tt.fields.ak, tt.fields.sk, tt.fields.endpoint)
  if err != nil {
   t.Errorf("failed init client, error = %v", err)
   return
  }
  if resp, err := c.CreateScaleDownInstanceGroupTask(tt.args.args); err != nil {
   t.Errorf("CreateScaleDownInstanceGroupTask() error = %v", err)

  } else {
   t.Logf("request id is: %s", resp.RequestID)
  }

 })
 }
}
```

常见问题

常见问题总览

通过 kubectl 创建简单的服务

- 通过 kubectl 连接 Kubernetes 集群
- 创建一个 nginx 应用

业务应用如何使用负载均衡？

- kubectl 在部署业务应用过程中使用负载均衡
- 通过 Kubernetes Dashboard 部署业务应用过程中使用负载均衡

- 在百度智能云控制台查看负载均衡和 EIP

自动扩缩容常见问题

- 集群在什么条件下会扩容或缩容？
- 自动扩缩容的扫描频率是多少？
- 扩缩容最大最小节点数的设置范围为多大？
- 自动扩容时创建节点的配置如何决定？
- 如何防止特定节点被缩容释放？

集群管理常见问题

- 常见问题
- 工单注意事项

通过kubectl创建简单的服务

通过 kubectl 连接 Kubernetes 集群

如果需要从本地个人计算机连接到百度智能云的 Kubernetes 集群，需要使用 Kubernetes 命令行客户端 kubectl，kubectl 的安装步骤如下所示

操作步骤

kubectl 的安装可以参见 kubernetes 官方安装和设置 kubectl文档。

> **注意：**
>
> 本操作指南给出的操作步骤都是基于 linux 操作环境的。

1.解压下载后的文件，为 kubectl 添加执行权限，并放在 PATH 下

```
 chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

2.配置 kubectl，下载集群凭证。在集群界面下载集群配置文件，放在 kubectl 的默认配置路径

```
 mv kubectl.conf  ~/.kube/config
```

3.配置完成后，您即可以使用 kubectl 从本地计算机访问 Kubernetes 集群

```
 kubectl get node
```

创建一个 nginx 应用

完成通过 kubectl 连接 Kubernetes 集群的操作后，输入以下指令：

```
 kubectl run my-nginx --image=nginx --replicas=3 --port=80
```

上述命令会创建3个 nginx 容器，每个容器都会暴露它的80端口。

常看创建 nginx 容器状态的指令如下：

```
 $ kubectl get pods
NAME                    READY    STATUS    RESTARTS   AGE
my-nginx-858393261-pfjdn   1/1      Running   0          14s
my-nginx-858393261-sn7g5   1/1      Running   0          14s
my-nginx-858393261-spv8w   1/1      Running   0          14s
```

> **注意：** 符号"$"代表输入的指令，其他行代表输入指令后显示的容器的信息。

暴露服务到 internet

创建服务

在公有云环境下，可以通过输入命令，创建出一个类型为 LoadBalancer 的 service，同时，还会联动公有云，创建一个负载均衡和一个公网 IP，

具体操作指令如下：

```
kubectl expose deployment my-nginx --port=80 --type=LoadBalancer
```

查看服务

通过查看服务的命令，可以看到这个服务的情况和创建出的公网 IP。

```
 $ kubectl get services
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP                PORT(S)       AGE
kubernetes   ClusterIP      172.16.0.1     <none>                     443/TCP       21d
my-nginx     LoadBalancer   172.16.72.193  106.13.134.33,192.168.0.75  80:31787/TCP  61s
```

通过访问180.76.139.247这个链接可以看到创建的服务如图所示：

完成服务创建之后，用户可以登录百度智能云平台，选择“产品服务>负载均衡 BLB”，进入 BLB 控制台查看负载均衡 BLB 的详细信息,具体操作步骤可以参考负载均衡操作指南

> **注意：** 符号"$"代表输入的指令，其他行代表输入指令后显示的服务的信息其中106.13.134.33为发布到外网的服务网址。 服务使用负载均衡除了上述指令调用之外，还可以通过Dashboard调用。

删除服务

如果需要删除服务，不想对公网暴露。可以直接删除服务

```
kubectl delete svc my-nginx
```

> **注意：**
>
> kubernetes会将刚才创建出的公网IP和负载均衡删除，然后也会删除创建的这个服务。

业务应用如何使用负载均衡？

业务应用过程中，可能会遇到高并发的场景，通过负载均衡 BLB 的流量均衡和分发功能，可以为应用提供多台 WebServer 并发服务的能力，从而达到业务水平扩展的效果。

目前创建业务应用的方式有两种: 一是通过 kubectl 创建业务应用; 二是通过 Kubernetes Dashboard 创建业务应用，因此创建和使用负载均衡的方式也分为以下两种：

kubectl 在部署业务应用过程中使用负载均衡

创建负载均衡和弹性公网IP EIP

在公有云环境下，可以通过输入命令，创建出一个类型为 LoadBalancer 的 service，同时，还会联动公有云，创建一个负载均衡和一个公网 IP，具体操作指令如下：

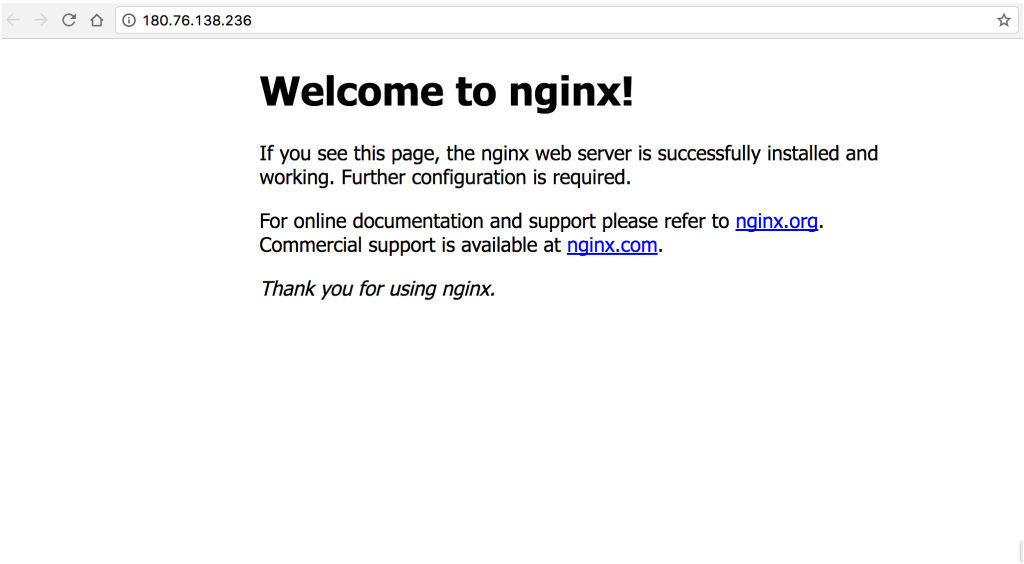
```
kubectl expose deployment my-nginx --port=80 --type=LoadBalancer
```

查看负载均衡和弹性公网IP EIP

通过查看服务的命令，可以看到这个服务的情况和创建出的公网 IP。

```
 $ kubectl get services
NAME         CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP     172.16.0.1       <none>                  443/TCP        21d
my-nginx     LoadBalancer  172.16.72.193    106.13.134.33,192.168.0.75   80:31787/TCP   3m16s
```

通过访问180.76.139.247这个链接可以看到创建的服务如图所示：

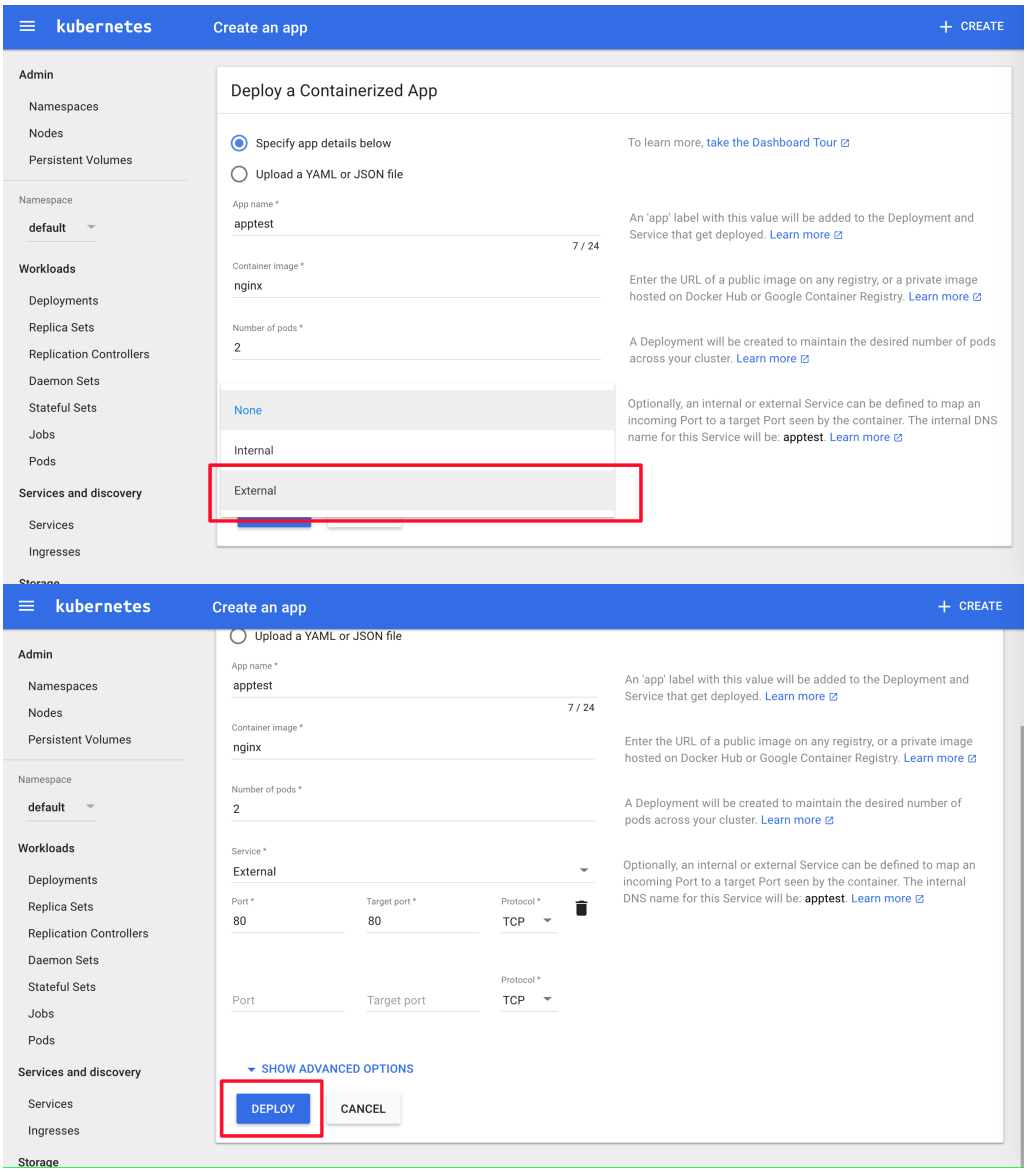


> **注意：** 符号"$"代表输入的指令，其他行代表输入指令后显示的服务的信息，其中180.76.139.247 为发布到外网的服务网址。

通过 Kubernetes Dashboard 部署业务应用过程中使用负载均衡

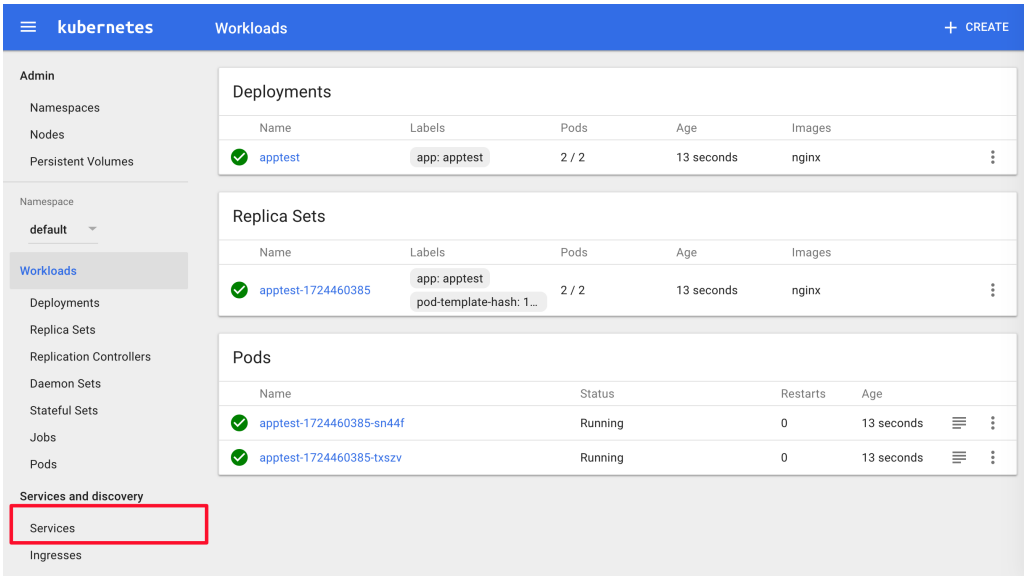
创建负载均衡和弹性公网IP EIP

Service 中选择 External 完成 EIP 和 BLB 的配置，然后会暴漏出端口页面分别填写80 80，然后点击 DEPLOY 按钮开始服务创建；



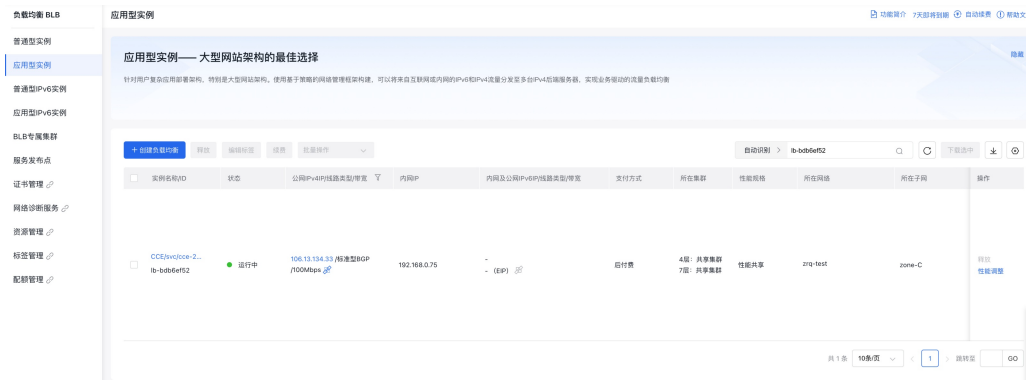
查看负载均衡和弹性公网IP EIP

在创建成功页面点击 Services 按钮，进入 services 页面；



在百度智能云控制台查看负载均衡和 EIP

完成服务创建之后，用户可以登录百度智能云平台，选择“产品服务>负载均衡 BLB”，进入 BLB 控制台查看负载均衡 BLB 的详细信息，具体操作步骤可以参考负载均衡操作指南



选择产品服务>弹性公网 IP EIP，进入控制台的实例列表，在此可以查看节点（实例）公网 IP 信息

自动扩缩容常见问题

集群在什么条件下会扩容或缩容？

- 扩容：由于资源不足，pod 调度失败，即有 pod 一直处于 Pending 状态。
- 缩容：node 的资源利用率较低时，且此 node 上存在的 pod 都能被重新调度到其他 node 上运行。

自动扩缩容的扫描频率是多少？

30秒

扩缩容最大最小节点数的设置范围为多大？

最小节点数需要大于等于0，最大节点数需要小于等于当前集群节点数限额（该限额可以通过提交工单提升）。最大最小节点数将会限制自动扩缩容的范围，但是不会限制用户手动发起的扩缩容行为。

自动扩容时创建节点的配置如何决定？

自动扩容时，将会按照当前集群中最主流的节点配置类型创建节点，即当前集群中节点数最多的配置类型。

如何防止特定节点被缩容释放？

以下类型的节点不会被缩容释放：

- 节点上有 pod 被 PodDisruptionBudget 控制器限制。
- 节点上的 pod 不是被控制器创建，例如不是被 deployment, replica set, job, stateful set 创建。
- 节点上有 pod 使用了本地存储
- 节点上 pod 驱逐后无处可去，即没有其他 node 能调度这个 pod
- 节点有注解："cluster-autoscaler.kubernetes.io/scale-down-disabled": "true"

集群管理常见问题

1. 常见问题

集群

错误 Code	错误说明	解决方案
MASTER_CREATE_FAILED	Master 节点创建/部署失败	1）如果是**独立部署**集群，请先至节点列表页 (集群列表->点击集群名->节点列表) 查看失败的 Master 节点进度展示，根据失败信息提示参考本文档解决问题，然后删除集群重新创建。 2) 如果是**托管模式**集群，请直接提工单解决，问题所属产品请选择：**【容器引擎CCE】**，工单所需提供信息请参考文末。
MASTER_DELETE_FAILED	Master 节点删除失败	1）如果是**独立部署**集群，请先至节点列表页 (集群列表->点击集群名->节点列表) 查看失败的 Master 节点进度展示，根据失败信息提示参考本文档解决问题，然后重新删除集群。 2) 如果是**托管模式**集群，请直接提工单解决，问题所属产品请选择：**【容器引擎CCE】**，工单所需提供信息请参考文末。
NODE_DELETE_FAILED	Worker 节点删除失败	请至节点列表页查看失败的 Worker 节点进度展示，根据失败信息提示参考本文档解决问题，然后重新删除集群。
ACCOUNT_INSUFFICIENT_BALANCE	账户余额不足	请先充值 (余额100元以上)，然后清理失败的资源，并重新创建。
FACE_AUTHENTICATION_NEEDED	未进行高级人脸认证	下载百度智能云 APP，在 "我的-实名认证" 中完成人脸认证，然后重试。
UNDEFINED_ERROR	未定义的错误	请直接提工单解决，问题所属产品请选择：**【容器引擎CCE】**，工单所需提供信息请参考文末。

节点

错误 Code	错误说明	解决方案
BCC_INSUFFICIENT	已经成功发起虚机创建，但是创建失败，大概率是该地域该可用区没有对应类型和配置的虚机资源了	1) 删除失败的节点并更换可用区或机器配置重新创建。 2) 提工单解决，问题所属产品请选择：**【云服务器BCC】**，工单所需提供信息请参考文末，其中**问题描述**填：机器创建失败。
INSTANCE_NO_STOCK	可用区内指定的机器没有库存	1) 删除失败的节点并更换可用区或机器配置重新创建。 2) 提工单解决，问题所属产品请选择：**【云服务器BCC】**，工单所需提供信息请参考文末，其中**问题描述**填：指定 instance 没有库存 (**务必**列出指定的机器实例类型和配置信息)
INSTANCE_QUOTA_EXCEEDED	用户账户下机器数量已经达到配额	提工单提升配额，问题所属产品请选择：**【云服务器BCC】**，工单所需提供信息请参考文末，其中**问题描述**填：提升 instance 配额 (可指定一个期望的数值)。
HOSTNAME_DUPLICATE_IN_VPC	同 VPC 内出现相同的 hostname	实例 hostname 是根据用户指定的实例名称按照一定的规则自动生成的 (如将实例名称改成全小写作为 hostname)，可重新新建实例并另取实例名称或者不指定实例名称 (随机生成) 。
NODE_SSH_TIMEOUT	节点 ssh 登录超时	请直接提工单解决，问题所属产品请选择：**【容器引擎CCE】**，工单所需提供信息请参考文末。
PREPAID_BCC_CANNOT_DELETED	预付费的机器不允许删除	1) 如果是删除集群之后报的错，可先在节点列表页将失败节点**移出**集群，然后再执行删除集群。 2) 如果是删除节点之后报的错，可直接在节点列表页将失败节点**移出**集群。
ACCOUNT_INSUFFICIENT_BALANCE	账户余额不足	请先充值 (余额100元以上)，然后清理失败的资源，并重新创建。
INSUFFICIENT_IP_IN_SUBNET	子网 IP 不足	清理失败的资源，更换子网重新创建。
VOLUME_NOT_AVAILABLE_IN_ZONE	可用区内指定的 CDS 资源不足	1) 清理失败的资源，更换可用区或者 CDS 类型重新创建。 2) 如果对 CDS 不是强需，可清理失败的资源，然后不选择 CDS 重新创建 3) 提工单解决，问题所属产品请选择：**【云服务器BCC】**，工单所需提供信息请参考文末，其中**问题描述**填：机器创建失败，Volume.ZoneResourceNotAvailable.
EIP_BY_TRAFFIC_BANDWIDTH_QUOTA_EXCEEDED	按流量计费的 EIP 总带宽已经达到账户配额	1) 自行清理账户下不用的按流量计费的 EIP，释放被占用的带宽额度，然后删除失败的节点重新创建。 2) 提工单提升按流量计费的 EIP 总带宽配额，问题所属产品请选择：**【弹性公网IP EIP】**，工单所需提供信息请参考文末，其中**问题描述**填：提升按流量计费的 EIP 总带宽配额。
DELETE_INSTANCE_WITH_USER_CUSTOM_ROUTE	待删除节点存在用户自定义的路由	在百度智能云控制台，进去 "私有网络VPC-路由表" 找到节点所在路由，确认无用后删除，然后再重试删除节点。
K8S_NODE_NOT_READY	部署完成，但 K8S Node NotReady	可使用 kubectl describe node 查看具体失败原因, 或 journalctl -u kubelet 查看具体日志。
UNDEFINED_ERROR	未定义的错误	请直接提工单解决，问题所属产品请选择：**【容器引擎CCE】**，工单所需提供信息请参考文末。

更多集群管理常见问题持续补充中.....

2. 工单注意事项

为了提高问题解决效率，节约您的时间，请严格按照以下约定提工单！

提工单链接：https://ticket.bce.baidu.com/?_=1603264688209/#/ticket/create

工单中需提供的信息：

- 地域名：如北京、广州、苏州、保定、香港、武汉等
- 集群 ID (如有)：为失败信息中的 clusterID，如 c-xxxxxxx、cce-xxxxxxxx，**问题所属产品非【容器引擎 CCE】时请勿提供**
- CCE 节点 ID (如有)：为失败信息中的 cceInstanceID，如 cce-xxxxxxxx-xxxxxxxx，**问题所属产品非【容器引擎 CCE】时请勿提供**

- 节点 ID (如有)：为失败信息中的 instanceID，如 i-xxxxxxxx
- 订单 ID (如有)：为失败信息中的 orderID
- 失败页面截图：**问题所属产品非【容器引擎 CCE】时请勿提供**
- 问题描述

服务等级协议SLA

CCE服务等级协议SLA(V1.0)

版本生效时间：2021年11月15日

本服务等级协议（Service Level Agreement，以下简称“SLA”）规定了百度智能云向客户提供容器引擎（Cloud Container Engine，以下简称“CCE”）的服务可用性等级指标。目前容器引擎CCE为免费服务，暂不承诺赔偿标准，但仍致力于为客户提供稳定可靠的服务。除非另有约定，本协议不适用于容器引擎CCE内测、公测的集群类型、功能、版本。

1. 定义

1.1 容器引擎CCE：指百度智能云提供的Kubernetes容器集群管理服务，客户可以在托管的Kubernetes集群上轻松运行应用程序和弹性扩展资源等，具体以客户创建的CCE集群类型及其关联云产品/服务提供的功能和服务内容为准。

1.2 服务周期：一个服务周期为一个自然月，不满一个自然月按照一个自然月计算。

1.3 服务周期内费用：CCE提供免费服务，不收取任何管理费用或软件费用。客户只需对使用CCE过程中，关联使用的其他付费云产品或服务按其付费规则进行付费。

1.4 服务周期总分钟数：按照服务周期内的总天数 × 24（小时）× 60（分钟）计算。

1.5 服务周期内不可用分钟数：在某一分钟内，客户通过CCE控制台或OpenAPI，持续尝试与指定的CCE托管Master集群/独立部署Master集群的Kubernetes API Server建立连接并使用服务，且均失败（无法连通集群的API Server或者API Server无法响应客户请求）的情况下，则视为该分钟内该CCE集群服务无法使用（若操作尝试全部或部分成功，则视为该分钟内的本服务可用）。服务无法使用持续时间达5分钟及5分钟以上，则计入不可用分钟数；服务无法使用时间不足5分钟，则不计入不可用分钟数。一个服务周期内计入不可用分钟数之和，即服务周期内不可用分钟数。

1.6 服务不可用：指客户通过CCE控制台或OpenAPI，持续尝试与指定的CCE托管Master集群/独立部署Master集群的Kubernetes API Server建立连接并使用服务，但无法连通集群的API Server或者API Server无法响应客户请求。 但不包括以下场景：

(1) 预先通知客户后进行的系统维护引起的不可用，如：割接、维修以及服务升级、变更、模拟故障演练等。

(2) 预先获得客户授权后进行的变更行为引起的不可用，如：集群版本升级、集群配置变更等。

(3) 由于客户主动操作导致的不可用，如：客户手动重启、删除Kubernetes核心系统组件；客户自行通过一些手段修改或攻击API Server服务，如安全攻击、压力测试等。

(4) 由于客户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的不可用。

(5) 由于客户的疏忽授权、错误操作引起的不可用，如：未通过CCE修改关联BLB的配置、误删除关联资源等。

(6) 由于客户未遵循百度智能云产品使用文档或使用建议引起的不可用。

(7) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的不可用。

(8) 不可抗力引起、客户自有设备或第三方软件/设备引起等非百度智能云原因造成的不可用.

(9) CCE关联的其他云产品或服务（如：VPC、BCC、BBC、EIP、BLB、CDS等）故障导致的不可用。

(10) 其他属于相关法律法规、相关协议、相关规则或百度云单独发布的相关规则、说明等中所述的百度云可以免责、免除赔偿责任等的情况。

2. 服务可用性

2.1 服务可用性计算方式

服务可用性 =（1 - 服务周期内不可用分钟数 / 服务周期总分钟数）× 100%

2.2 服务可用性标准

CCE集群的API Server服务可用性为 99.9%。

注：通过CCE级联创建或手动加入等方式，关联使用的VPC、BCC、BBC、BLB、EIP、CDS、BOS等云产品的可用性，以及通过CCE系统策略授权关联使用的BCM、CCR等服务的可用性，以各自云产品或服务提供的服务等级协议为准。

3. 赔偿方案

目前容器引擎CCE为免费服务，暂不对服务可用性承诺赔偿标准。

常用的关联云产品/服务的服务等级协议可参考以下文档（更多可在百度智能云文档中心查看）：

- BCC：https://cloud.baidu.com/doc/BCC/s/Ikxk15ii2
- BLB: https://cloud.baidu.com/doc/BLB/s/Mjwvxnu7o

- EIP: https://cloud.baidu.com/doc/EIP/s/Ljwvz2voz
- CDS: https://cloud.baidu.com/doc/CDS/s/Yjwvyakvv
- BOS: https://cloud.baidu.com/doc/BOS/s/Hjwvys7xf

4. 其他说明

4.1 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。

4.2 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将以网站公示或发送邮件等方式通知客户。如客户不同意百度智能云对SLA所做的修改，客户有权停止使用本服务，如客户继续使用本服务，则视为客户接受修改后的SLA。

4.3 本协议项下百度智能云对于客户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该等通知于发送之日视为已送达收件人。因客户自身未及时了解百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担赔偿责任。

4.4 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。

4.5 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。

4.6 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。

常用工具

视图命令行场景示例

一般的问题定位流程可以分为五个阶段：**问题发现 => 数据导出 => 关键错误提取 => 视图分析 => 问题处理**

场景1:资源超配额

1. **问题发现**：pytorchjob 没有进入running状态，对应的pod长时间pending

```
 % kubectl get pytorchjob
NAME                    STATE     AGE
pytorchjob-bert-2-bert   Created   58m

% kubectl get pod
pytorchjob-bert-2-bert-master-0        0/2     Pending          0        58m
pytorchjob-bert-2-bert-worker-0        0/2     Pending          0        58m
```

2. 调整日志级别，获取视图文件与日志文件

```
 #### 调整日志等级到4
% cce-volcano-cli log -v 4
4
#### dump日志与视图文件
cce-volcano-cli dump -l 5000
copy file volcano-scheduler-7848fb7487-44fpq:/dump/volcano.1695461246.snapshot to ./volcano.1695461246.snapshot success!
start dump volcano logfile volcano.1695461246.log
end dump volcano logfile volcano.1695461246.log

#### 调整日志等级到3
% cce-volcano-cli log -v 3 -k kubeconfig
3
```

3. 关键错误提取：通过job日志，发现**ExceedQueueResource**，因此需要对队列配额分配情况进行分析

```
 % cce-volcano-cli log job -f volcano.1695461246.log -j pytorchjob-bert-2-bert
```

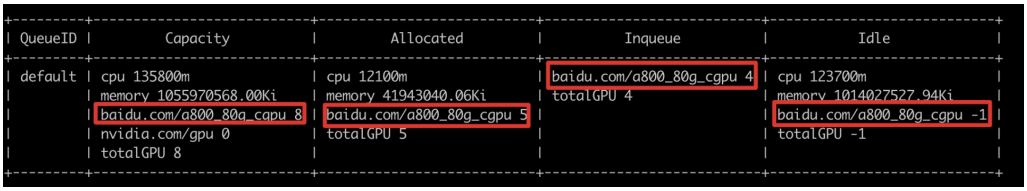
```
{
    "EventName": "ResouceCapcityOverUsed",
    "JobName": "pytorchjob-bert-2-bert",
    "Namespace": "default",
    "PodGroupName": "pytorchjob-bert-2-bert",
    "Queue": "default",
    "Reason": "ExceedQueueResource",
    "Session": "41bf2518-d0c3-4111-88ca-9a7e76018019",
    "caller": "cceproportion/proportion.go:479",
    "msg": "Reject job enqueue",
    "ts": 1695461821912.9575,
    "v": 3
}
```


得到配额不足的错误信息以后，可以使用 **cce-volcano-cli queue** 命令对队列配额进行分析，常见的几种情况如下

case1 队列配额不足

a800资源已经用完，其中总卡数为8，已分配5卡，排队中4卡

```
% cce-volcano-cli queue -f volcano.1695461246.snapshot
```

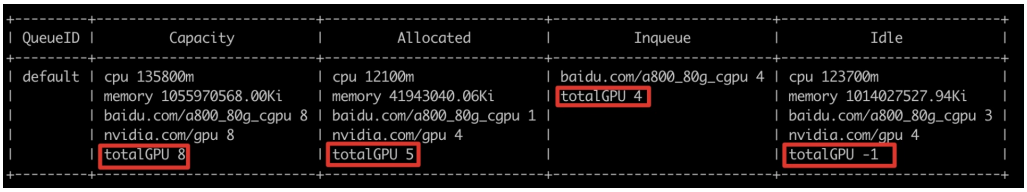


QueueID	Capacity	Allocated	Inqueue	Idle
default	cpu 135800m	cpu 12100m	baidu.com/a800_80g_cgpu 4	cpu 123700m
	memory 1055970568.00Ki	memory 41943040.06Ki	totalGPU 4	memory 1014027527.94Ki
	baidu.com/a800_80g_cgpu 8	baidu.com/a800_80g_cgpu 5		baidu.com/a800_80g_cgpu -1
	nvidia.com/gpu 0	totalGPU 5		totalGPU -1
	totalGPU 8			

case2 混合卡调度

对于混合申请的场景，工具提供了**totalGPU字段**,用于统计实际剩余的卡数，可以看到总卡数为8，已分配5卡，排队中4卡，剩余卡为-1。
已分配的卡中，1个通过baidu.com/a800_80g_cgpu的方式申请，4个通过nvidia.com/gpu的方式申请。排队中的4卡使用了baidu.com/a800_80g_cgpu的方式申请

```
% cce-volcano-cli queue -f volcano.1695461246.snapshot
```

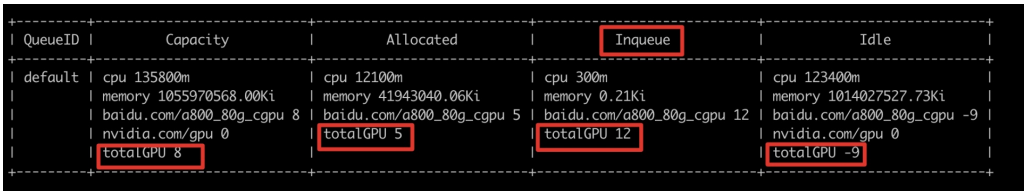


QueueID	Capacity	Allocated	Inqueue	Idle
default	cpu 135800m	cpu 12100m	baidu.com/a800_80g_cgpu 4	cpu 123700m
	memory 1055970568.00Ki	memory 41943040.06Ki	totalGPU 4	memory 1014027527.94Ki
	baidu.com/a800_80g_cgpu 8	baidu.com/a800_80g_cgpu 1		baidu.com/a800_80g_cgpu 3
	nvidia.com/gpu 8	nvidia.com/gpu 4		nvidia.com/gpu 4
	totalGPU 8	totalGPU 5		totalGPU -1

case3 podgroup残留

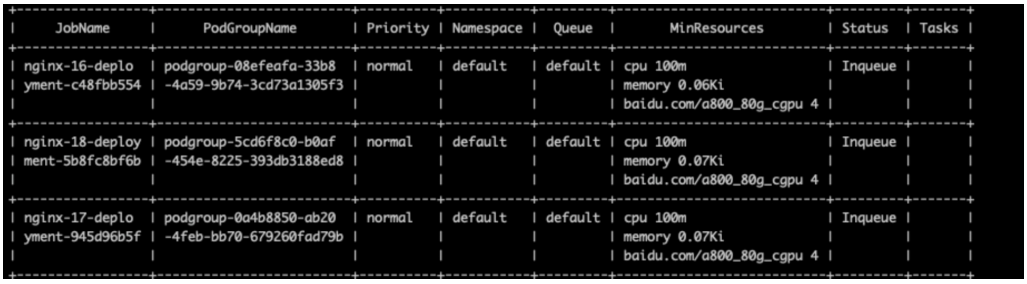
inqueue状态资源较多，实际还剩3卡没有用起来

```
% cce-volcano-cli queue -f volcano.1695461246.snapshot
```



QueueID	Capacity	Allocated	Inqueue	Idle
default	cpu 135800m	cpu 12100m	cpu 300m	cpu 123400m
	memory 1055970568.00Ki	memory 41943040.06Ki	memory 0.21Ki	memory 1014027527.73Ki
	baidu.com/a800_80g_cgpu 8	baidu.com/a800_80g_cgpu 5	baidu.com/a800_80g_cgpu 12	baidu.com/a800_80g_cgpu -9
	nvidia.com/gpu 0	totalGPU 5	totalGPU 12	nvidia.com/gpu 0
	totalGPU 8			totalGPU -9

使用cce-volcano-cli job命令进行job视图分析，查看inqueue状态job列表，并对其中的异常job做清理
% cce-volcano-cli job -f volcano.1695461246.snapshot -a
工具提供了-t选项对这种异常的job列表进行筛选
% cce-volcano-cli job -f volcano.1695461246.snapshot -t



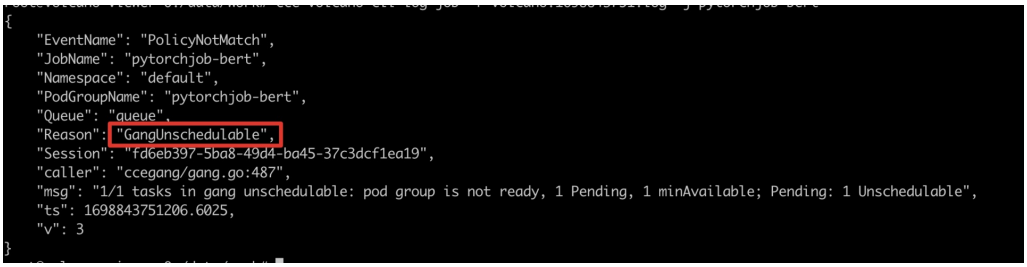
场景2 亲和性、污点、GPU资源不足

1. **问题发现：job**一直是Created状态，pod pending

```
 % kubectl get PyTorchJob
NAME            STATE     AGE
pytorchjob-bert   Created   36s
% kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
pytorchjob-bert-master-0   0/2     Pending   0          39s
```

2. **关键错误提取：**通过job日志，发现**GangUnschedulable**，不满足gang调度，由此可以知道是pod调度失败导致pending，继续查pod失败原因

```
 #### 收集CCE AI Job Scheduler日志
% cce-volcano-cli dump -l 5000
#### 通过命令行查看job级别日志./c
% cce-volcano-cli log job -f volcano.1698844162.log -j pytorchjob-bert
```



3. 通过命令行查看pod级别日志

```
#### 通过命令行查看pod级别日志
% cce-volcano-cli log pod -f volcano.1698844162.log -p pytorchjob-bert-master-0
```

```
failure reason : SchedulingDisabled ,involved Nodes : [10.0.0.5]
failure reason : Insufficient CPU/Memory Resources ,involved Nodes : [10.0.0.181]
failure reason : Node Has Taint ,involved Nodes : [10.0.0.142 10.0.0.3 10.0.0.4]
```

192.168.14.5 是封锁节点

192.168.11.181 cpu或者memory资源不足

192.168.14.142 10.0.3 10.0.4 节点有污点

case 1 亲和性匹配

通过node视图分析：10.0.0.181 cpu资源不足(job申请1000m cpu，node剩余50m)。

10.0.0.3 10.0.0.4 10.0.0.142 节点有gpu资源但是有污点，无法调度，需要确认是否使用该节点。

10.0.0.5 是封锁节点(status:Unschedulable)。

```
% cce-volcano-cli node -f volcano.1698844162.snapshot
```

```
|    Name     |         Capacity          |        Allocatable        |      Used       |          Idle             |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+
| 10.0.0.5    | cpu 80.00                 | cpu 79.90                 | cpu 10.75       | cpu 69.15                 |
|             | memory 314.889Gi          | memory 314.596Gi          | memory 11.154Gi | memory 303.441Gi          |
|             | baidu.com/rtx_3090_cgpu 8 | baidu.com/rtx_3090_cgpu 8 | totalGPU 0      | baidu.com/rtx_3090_cgpu 8 |
|             | nvidia.com/gpu 8          | nvidia.com/gpu 8          |                 | nvidia.com/gpu 8          |
|             | totalGPU 8                | totalGPU 8                |                 | totalGPU 8                |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+
| 10.0.0.3    | cpu 2.00                  | cpu 2.00                  | cpu 0.55        | cpu 1.45                  |
|             | memory 7.795Gi            | memory 7.697Gi            | memory 0.439Gi  | memory 7.258Gi            |
|             | totalGPU 0                | totalGPU 0                | totalGPU 0      | totalGPU 0                |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+
| 10.0.0.142  | cpu 8.00                  | cpu 7.90                  | cpu 3.80        | cpu 4.10                  |
|             | memory 31.351Gi           | memory 31.058Gi           | memory 6.602Gi  | memory 24.456Gi           |
|             | totalGPU 0                | totalGPU 0                | totalGPU 0      | totalGPU 0                |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+
| 10.0.0.4    | cpu 80.00                 | cpu 79.90                 | cpu 5.80        | cpu 74.10                 |
|             | memory 314.882Gi          | memory 314.589Gi          | memory 7.254Gi  | memory 307.336Gi          |
|             | baidu.com/rtx_3090_cgpu 8 | baidu.com/rtx_3090_cgpu 8 | totalGPU 0      | baidu.com/rtx_3090_cgpu 8 |
|             | nvidia.com/gpu 8          | nvidia.com/gpu 8          |                 | nvidia.com/gpu 8          |
|             | totalGPU 8                | totalGPU 8                |                 | totalGPU 8                |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+
| 10.0.0.181  | cpu 2.00                  | cpu 1.90                  | cpu 1.40        | cpu 0.50                  |
|             | memory 7.763Gi            | memory 7.470Gi            | memory 4.906Gi  | memory 2.563Gi            |
|             | totalGPU 0                | totalGPU 0                | totalGPU 0      | totalGPU 0                |
+-------------+---------------------------+---------------------------+-----------------+---------------------------+

+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
|    Name     | Releasing  |                 Status                  |                 Taints                  | Tasks |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
| 10.0.0.5    | totalGPU 0 | {"Status":"Ready","Reason":"Unschedulable"} | [                                       |       |
|             |            |                                         |  {                                      |       |
|             |            |                                         |   "key": "node.kubernetes.io/unschedulable", |       |
|             |            |                                         |   "effect": "NoSchedule",               |       |
|             |            |                                         |   "timeAdded": "2023-11-01T11:56:27Z"   |       |
|             |            |                                         |  }                                      |       |
|             |            |                                         | ]                                       |       |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
| 10.0.0.3    | totalGPU 0 | {"Status":"Ready","Reason":""}          | [                                       |       |
|             |            |                                         |  {                                      |       |
|             |            |                                         |   "key": "node-role.kubernetes.io/master", |       |
|             |            |                                         |   "effect": "NoSchedule"                |       |
|             |            |                                         |  }                                      |       |
|             |            |                                         | ]                                       |       |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
| 10.0.0.142  | totalGPU 0 | {"Status":"Ready","Reason":""}          | [                                       |       |
|             |            |                                         |  {                                      |       |
|             |            |                                         |   "key": "key1",                        |       |
|             |            |                                         |   "value": "value1",                    |       |
|             |            |                                         |   "effect": "NoSchedule"                |       |
|             |            |                                         |  }                                      |       |
|             |            |                                         | ]                                       |       |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
| 10.0.0.4    | totalGPU 0 | {"Status":"Ready","Reason":""}          | [                                       |       |
|             |            |                                         |  {                                      |       |
|             |            |                                         |   "key": "key1",                        |       |
|             |            |                                         |   "value": "value1",                    |       |
|             |            |                                         |   "effect": "NoSchedule"                |       |
|             |            |                                         |  }                                      |       |
|             |            |                                         | ]                                       |       |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
| 10.0.0.181  | totalGPU 0 | {"Status":"Ready","Reason":""}          |                                         |       |
+-------------+------------+-----------------------------------------+-----------------------------------------+-------+
```

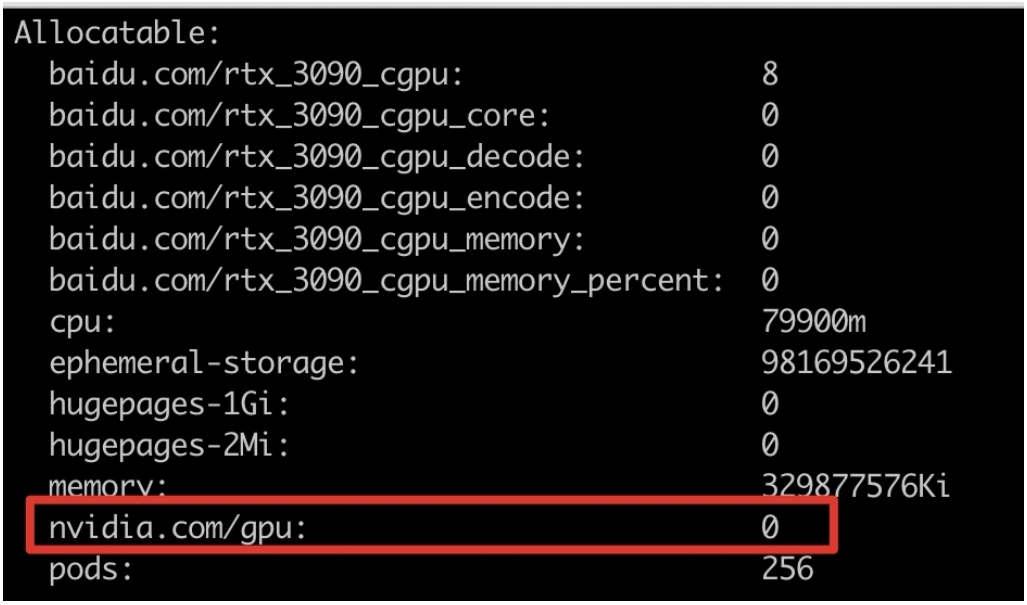
case 2 gpu掉卡

如果节点有gpu资源，并且任务需要卡数<=node节点卡数，仍然无法调度。可查看node视图，判断是否掉卡。

如下图所示，发现节点的allocatable的nvidia.com/gpu<节点capacity ，初步判断节点掉卡。

```
+----------+--------------------------+--------------------------+--------------------------+--------------------------+
|   Name   |         Capacity         |        Allocatable       |           Used           |           Idle           |
+----------+--------------------------+--------------------------+--------------------------+--------------------------+
| 10.0.0.5 | cpu 80.00                | cpu 79.90                | cpu 11.75                | cpu 68.15                |
|          | memory 314.889Gi         | memory 314.596Gi         | memory 12.154Gi          | memory 302.441Gi         |
|          | baidu.com/rtx_3090_cgpu 8 | baidu.com/rtx_3090_cgpu 0 | baidu.com/rtx_3090_cgpu 0 | baidu.com/rtx_3090_cgpu 0 |
|          | nvidia.com/gpu 8         | nvidia.com/gpu 0         | totalGPU 0               | nvidia.com/gpu 0         |
|          | totalGPU 8               | totalGPU 0               |                          | totalGPU 0               |
+----------+--------------------------+--------------------------+--------------------------+--------------------------+
+----------+--------------------------+------------------------------+--------+-------+
|   Name   |         Releasing        |            Status            | Taints | Tasks |
+----------+--------------------------+------------------------------+--------+-------+
| 10.0.0.5 | cpu 1.00                 | {"Status":"Ready","Reason":""} |        |       |
|          | memory 1.000Gi           |                              |        |       |
|          | baidu.com/rtx_3090_cgpu 1 |                              |        |       |
|          | totalGPU 1               |                              |        |       |
+----------+--------------------------+------------------------------+--------+-------+
```

describe node发现，nvidia.com/gpu为0，确认掉卡。



```
Allocatable:
  baidu.com/rtx_3090_cgpu:                8
  baidu.com/rtx_3090_cgpu_core:           0
  baidu.com/rtx_3090_cgpu_decode:         0
  baidu.com/rtx_3090_cgpu_encode:         0
  baidu.com/rtx_3090_cgpu_memory:         0
  baidu.com/rtx_3090_cgpu_memory_percent: 0
  cpu:                                    79900m
  ephemeral-storage:                      98169526241
  hugepages-1Gi:                          0
  hugepages-2Mi:                          0
  memory:                                 329877576Ki
  nvidia.com/gpu:                         0
  pods:                                   256
```