云容器引擎

用户指南

文档版本 01

发布日期 2024-11-12

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

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，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 高危操作一览

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

集群/节点

表 1-1 集群及节点高危操作

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

分类	高危操作	导致后果	误操作后解决方案
	自行更换Master或etcd证书	可能导致集群不可用	不可恢复。
Node节点	修改集群内节点安全组 **说明** 安全组命名规则：集群名称-cce-node-随机数	可能导致节点无法使用	参照新建集群的安全组进行修复，放通安全组。详情请参见**集群安全组规则配置**。
	修改节点DNS配置（/etc/resolv.conf）	导致内部域名无法正常访问，可能出现插件异常、节点重置升级等基本功能异常 **说明** 如果业务需要使用自建DNS，可以在工作负载中配置DNS，请勿修改节点本身的DNS地址，详情请参见**工作负载DNS配置说明**。	参考新建节点中的DNS配置还原。
	节点被删除	该节点不可用	不可恢复。
	重装操作系统	节点组件被删除，节点不可用	重置节点，具体请参见**重置节点**。
	升级内核或容器平台依赖组件（如openvswitch/ipvlan/docker/containerd）	可能导致节点无法使用或网络异常 **说明** 节点运行依赖系统内核版本，如非必要，请不要使用**yum update**命令更新或重装节点的操作系统内核（使用原镜像或其它镜像重装均属高危操作）	EulerOS 2.2恢复方式请参见**如何解决yum update升级操作系统导致容器网络不可用问题?** 非EulerOS 2.2您可以重置节点，具体请参见**重置节点**。
	更改节点IP	节点不可用	改回原IP。
	自行修改核心组件（kubelet、kube-proxy等）参数	可能导致节点不可用、修改安全相关配置导致组件不安全等	按照推荐配置参数恢复，详情请参见**修改节点池配置**。
	修改操作系统配置	可能导致节点不可用	尝试还原配置项或重置节点，具体请参见**重置节点**。
	删除或修改/opt/cloud/cce、/var/paas目录，删除数据盘	节点不可用	重置节点，具体请参见**重置节点**。

分类	高危操作	导致后果	误操作后解决方案
	修改节点内目录权限、容器目录权限等	权限异常	不建议修改，请自行恢复。
	对节点进行磁盘格式化或分区，包括系统盘、Docker盘和kubelet盘	可能导致节点不可用	重置节点，具体请参见**重置节点**。
	在节点上安装自己的其他软件	导致安装在节点上的Kubernetes组件异常，节点状态变成不可用，无法部署工作负载到此节点	卸载已安装软件，尝试恢复或重置节点，具体请参见**重置节点**。
	修改NetworkManager的配置	节点不可用	重置节点，具体请参见**重置节点**。
	删除节点上的cce-pause等系统镜像	导致无法正常创建容器，且无法拉取系统镜像	请从其他正常节点复制该镜像恢复
	在ECS侧对节点池下的节点进行规格变更	节点的规格与节点池定义的规格不一致，导致在弹性扩缩容时出现非预期现象（多扩或者少扩）	重新将节点规格变更为节点池下定义的规格，或者删除该节点重新扩容。

网络

表 1-2 网络

高危操作	导致后果	误操作后解决方案
修改内核参数net.ipv4.ip_forward=0	网络不通	修改内核参数为net.ipv4.ip_forward=1
修改内核参数net.ipv4.tcp_tw_recycle=1	导致nat异常	修改内核参数net.ipv4.tcp_tw_recycle=0
修改内核参数net.ipv4.tcp_tw_reuse=1	导致网络异常	修改内核参数net.ipv4.tcp_tw_reuse=0
节点安全组配置未放通容器CIDR的53端口udp	集群内DNS无法正常工作	参照**新建集群**的安全组进行修复，放通安全组。
删除default-network的network-attachment-definitions的crd资源	容器网络不通，集群删除失败等	误删除该资源需要使用正确的配置重新创建default-network资源。

高危操作	导致后果	误操作后解决方案
启动iptables防火墙	CCE默认不开启iptables防火墙，开启后可能造成网络不通 **说明** 不建议开启iptables防火墙。如必须启动iptables防火墙，请在测试环境中确认/etc/sysconfig/iptables和/etc/sysconfig/ip6tables中配置的规则是否会对网络连通性造成影响。	关闭iptables防火墙，并检查/etc/sysconfig/iptables和/etc/sysconfig/ip6tables中配置的规则。

容器

表 1-3 容器

高危操作	导致后果	误操作后解决方案
将负载设置为特权容器，并直接操作主机的硬件设备，误操作节点系统文件等。 例如将启动命令配置为/usr/sbin/init，在容器内使用systemctl，影响节点/lib路径下的系统文件。	此操作可能会导致节点所有挂载点被unmount，导致节点异常，并且会影响节点上的Pod正常运行及存储插件功能。	禁止移除节点/lib路径下的挂载点，通过重置节点恢复，具体请参见**重置节点**。

负载均衡

表 1-4 Service ELB

高危操作	导致后果	误操作后解决方案
禁止通过ELB的控制台删除已绑定CCE集群的ELB实例	导致Service/Ingress访问不通。	不建议删除。
通过ELB的控制台停用已绑定CCE集群的ELB实例	导致Service/Ingress访问不通。	不建议停用，请自行恢复。
通过ELB的控制台修改ELB的IPv4私有IP	• 基于IPv4私有IP进行私网流量转发功能会出现中断 • Service/Ingress的YAML中status字段下的IP变化	不建议修改，请自行恢复。

高危操作	导致后果	误操作后解决方案
通过ELB的控制台解绑ELB的IPv4公网IP	解绑公网IP后，该弹性负载均衡器变更为私网类型，无法进行公网流量转发。	请自行恢复。
通过ELB的控制台在CCE管理的ELB创建自定义的监听器	若ELB是创建Service/Ingress时自动创建的，在Service/Ingress删除时无法删除ELB的自定义监听器，会导致无法自动删除ELB。	通过Service/Ingress自动创建监听器，否则需要手动删除ELB。
通过ELB的控制台删除CCE自动创建的监听器	● 导致Service/Ingress访问不通。 ● 在集群升级等需要重启控制节点的场景，所做修改会被CCE侧重置。	重新创建或更新Service/Ingress。
通过ELB的控制台修改CCE创建的监听器名称、访问控制、超时时间、描述等基本配置	如果监听器被删除，在集群升级等需要重启控制节点的场景，所做修改会被CCE侧重置。	不建议修改，请自行恢复。
通过ELB的控制台修改CCE创建的监听器后端服务器组，添加、删除后端服务器	● 导致Service/Ingress访问不通。 ● 在集群升级等需要重启控制节点的场景，所做修改会被CCE侧重置： – 用户删除的后端服务器会恢复 – 用户添加的后端服务器会被移除	重新创建或更新Service/Ingress。
通过ELB的控制台更换CCE创建的监听器后端服务器组	● 导致Service/Ingress访问不通。 ● 在集群升级等需要重启控制节点的场景，后端服务器组中的后端服务器会被CCE侧重置。	重新创建或更新Service/Ingress。
通过ELB的控制台修改CCE创建的监听器转发策略，添加、删除转发规则	● 导致Service/Ingress访问不通。 ● 如果该转发规则由Ingress添加，在集群升级等需要重启控制节点的场景，所做修改会被CCE侧重置。	不建议修改，请自行恢复。

高危操作	导致后果	误操作后解决方案
通过ELB的控制台修改CCE管理的ELB证书	在集群升级等需要重启控制节点的场景，后端服务器组中的后端服务器会被CCE侧重置。	通过Ingress的YAML来自动管理证书。

日志

表 1-5 日志

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

云硬盘

表 1-6 云硬盘

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

插件

表 1-7 插件

高危操作	导致后果	误操作后解决方案
后台修改插件相关资源	插件异常或引入其他非预期问题	通过插件配置页面或开放的插件管理API进行操作

2 集群

2.1 集群概述

2.1.1 集群基本信息

Kubernetes是一个开源的容器编排引擎，可用于容器化应用的自动化部署、 扩缩和管理。

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

集群的网络

集群的网络可以分成三个部分：

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

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

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

Master 节点数量与集群规模

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

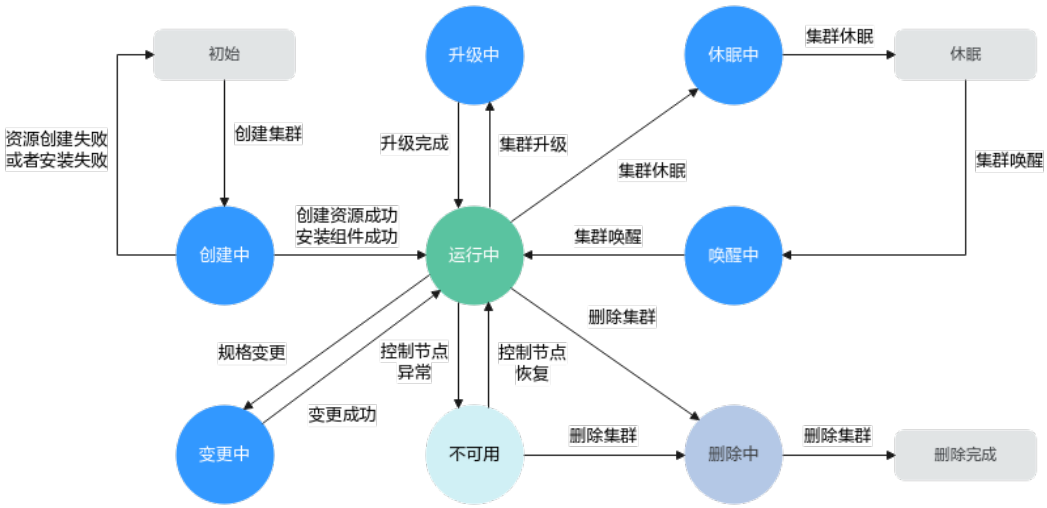
Master节点的规格决定集群管理Node节点的规模，创建集群时可以选择集群管理规模，这个规模就是指的集群可以有多少个Node节点，例如50节点、200节点等。

集群生命周期

表 2-1 集群状态说明

状态	说明
创建中	集群正在创建，正在申请云资源
运行中	集群正常运行
休眠中	集群正在休眠中
唤醒中	集群正在唤醒中
升级中	集群正在升级中
变更中	集群正处于规格变更中
不可用	当前集群不可用
删除中	集群正在删除中

图 2-1 集群状态流转



2.1.2 Kubernetes 版本发布记录

2.1.2.1 Kubernetes 1.30 版本说明

云容器引擎（CCE）严格遵循社区一致性认证，现已支持创建Kubernetes 1.30集群。本文介绍Kubernetes 1.30版本的变更说明。

索引

- 新增特性及特性增强
- API变更与弃用

- **CCE对Kubernetes 1.30版本的增强**
- **参考链接**

新增特性及特性增强

- Webhook匹配表达式（GA）

  在Kubernetes1.30版本中，Webhook匹配表达式特性进阶至GA。此特性允许对准入Webhook支持根据特定的条件进行匹配，更细粒度地控制Webhook的触发条件。详细使用方式请参考**动态准入控制**。

- Pod调度就绪态（GA）

  在Kubernetes1.30版本中，Pod调度就绪态特性进阶至GA。此特性允许对Pod添加自定义的schedulingGates，并由用户控制何时移除这些gate，当所有gates移除后，Pod才会被认为调度就绪。详细使用方式请参考**Pod调度就绪态**。

- 验证准入策略（GA）

  在Kubernetes1.30版本中，验证准入策略（ValidatingAdmissionPolicy）特性进阶至GA。该特性支持通过CEL表达式声明资源的验证准入策略。详细使用方式请参考**验证准入策略**。

- 基于ContainerResource指标的Pod水平自动扩缩容（GA）

  在Kubernetes1.30版本中，基于ContainerResource指标的Pod水平自动扩缩容特性进阶至GA。该特性允许HPA根据Pod中各个容器的资源使用情况来配置自动伸缩，而不仅是Pod的整体资源使用情况，便于为Pod中最重要的容器配置扩缩容阈值。详细使用方式请参考**容器资源指标**。

- 传统ServiceAccount令牌清理器（GA）

  在Kubernetes1.30版本中，传统ServiceAccount令牌清理器特性进阶至GA。其作为kube-controller-manager的一部分运行，每24小时检查一次，查看是否有任何自动生成的传统ServiceAccount令牌在特定时间段内（默认为一年，通过--legacy-service-account-token-clean-up-period指定）未被使用。如果有的话，清理器会将这些令牌标记为无效，并添加kubernetes.io/legacy-token-invalid-since标签，其值为当前日期。如果一个无效的令牌在特定时间段（默认为1年，通过--legacy-service-account-token-clean-up-period指定）内未被使用，清理器将会删除它。更多使用细节请参考**传统ServiceAccount令牌清理器**。

- Pod拓扑分布中的最小域（GA）

  在Kubernetes1.30版本中，Pod拓扑分布中的最小域特性进阶至GA。此特性允许通过Pod的minDomains字段配置符合条件的域的最小数量。负载拓扑约束匹配到的域的数量如果大于minDomains，则该字段没有影响；如果小于minDomains，则会将全局最小值（符合条件的域中匹配 Pod 的最小数量）设为0，该字段必须结合whenUnsatisfiable: DoNotSchedule一起使用，实现在不满足拓扑约束的情况下让Pod不进行调度。详细使用方式参考**Pod拓扑分布**。

API 变更与弃用

- 在Kubernetes1.30版本中，kubectl移除了apply命令的prune-whitelist参数，使用prune-allowlist替代。
- 在Kubernetes1.30版本中，移除了在1.27版本已废弃的准入插件SecurityContextDeny，使用**Pod安全性准入插件**（PodSecurity）替代。

CCE 对 Kubernetes 1.30 版本的增强

在版本维护周期中，CCE会对Kubernetes 1.30版本进行定期的更新，并提供功能增强。

关于CCE集群版本的更新说明，请参见**补丁版本发布说明**。

参考链接

关于Kubernetes 1.30与其他版本的性能对比和功能演进的更多信息，请参考：
Kubernetes v1.30 Release Notes

2.1.2.2 Kubernetes 1.29 版本说明

云容器引擎（CCE）严格遵循社区一致性认证，现已支持创建Kubernetes 1.29集群。本文介绍Kubernetes 1.29版本的变更说明。

索引

新增特性及特性增强

- Service的负载均衡IP模式（Alpha）

  在Kubernetes1.29版本，Service的负载均衡IP模式以Alpha版本正式发布。其在Service的status中新增字段ipMode，用于配置集群内Service到Pod的流量转发模式。当设置为VIP时，目的地址为负载均衡IP和端口的流量将由kube-proxy重定向到目标节点，当设置为Proxy时，流量将被发送到负载均衡器，然后由负载均衡器转发到目标节点。这项特性将有助于解决流量绕过负载均衡器缩导致的负载均衡器功能缺失问题。更多使用细节请参考**Service的负载均衡IP模式**。

- NFTables代理模式（Alpha）

  在Kubernetes1.29版本，NFTables代理模式以Alpha版本正式发布。该特性允许kube-proxy运行在NFTables模式，在该模式下，kube-proxy使用内核netfilters子系统的nftables API来配置数据包转发规则。更多使用细节请参考**NFTables代理模式**。

- 未使用容器镜像的垃圾收集（Alpha）

  在Kubernetes1.29版本，未使用容器镜像的垃圾收集以Alpha版本正式发布。该特性允许用户为每个节点配置本地镜像未被使用的最长时间，超过这个时间镜像将被垃圾回收。配置方法为使用kubelet配置文件中的ImageMaximumGCAge字段。更多使用细节请参考**未使用容器镜像的垃圾收集**。

- PodLifecycleSleepAction（Alpha）

  在Kubernetes1.29版本，PodLifecycleSleepAction以Alpha版本正式发布。该特性在容器生命周期回调中引入了Sleep回调程序，可以配置让容器在启动后和停止前暂停一段指定的时间。更多使用细节请参考**PodLifecycleSleepAction**。

- KubeletSeparateDiskGC（Alpha）

  在Kubernetes1.29版本，KubeletSeparateDiskGC以Alpha版本正式发布。该特性启用后，即使在容器镜像和容器位于独立文件系统的情况下，也能进行垃圾回收。

- matchLabelKeys/mismatchLabelKeys（Alpha）

  在Kubernetes1.29版本，matchLabelKeys/mismatchLabelKeys以Alpha版本正式发布。该特性启用后，在Pod的亲和/反亲和配置中新增了matchLabelKeys/

mismatchLabelKeys字段，可配置更丰富的Pod间亲和/反亲和策略。更多使用细节请参考**matchLabelKeys/mismatchLabelKeys**。

- clusterTrustBundle投射卷（Alpha）

  在Kubernetes1.29版本，clusterTrustBundle投射卷以Alpha版本正式发布。该特性启用后，支持将ClusterTrustBundle对象以自动更新的文件的形式注入卷。更多使用细节请参考**clusterTrustBundle投射卷**。

- 基于运行时类的镜像拉取（Alpha）

  在Kubernetes1.29版本中，基于运行时类的镜像拉取以Alpha版本正式发布。该特性启用后， kubelet 会通过一个元组（镜像名称，运行时处理程序）而不仅仅是镜像名称或镜像摘要来引用容器镜像。 您的容器运行时可能会根据选定的运行时处理程序调整其行为。基于运行时类来拉取镜像对于基于VM的容器会有帮助。更多使用细节请参考**基于运行时类的镜像拉取**。

- PodReadyToStartContainers状况达到Beta

  在Kubernetes1.29版本，PodReadyToStartContainers状况特性达到Beta版本。其在Pod的status中新增了一个名为PodReadyToStartContainers的Condition，该Condition为true表示Pod的沙箱已就绪，可以开始创建业务容器。该特性使得集群管理员可以更清晰和全面地查看 Pod 沙箱的创建完成和容器的就绪状态，增强了指标监控和故障排查能力。更多使用细节请参考**PodReadyToStartContainersCondition**。

- Job相关特性
  - Pod更换策略达到Beta

    在Kubernetes1.29版本，Pod更换策略特性达到Beta版本。该特性确保只有Pod达到Failed阶段（status.phase: Failed）才会被替换，而不是当删除时间戳不为空时，Pod仍处于删除过程中就重建Pod，以此避免出现2个Pod同时占用索引和节点资源。

  - 逐索引的回退限制达到Beta

    在Kubernetes1.29版本，逐索引的回退限制特性达到Beta版本。默认情况下，带索引的 Job（Indexed Job）的 Pod 失败情况会被统计下来，受 .spec.backoffLimit 字段所设置的全局重试次数限制。这意味着，如果存在某个索引值的 Pod 一直持续失败，则会 Pod 会被重新启动，直到重试次数达到限制值。 一旦达到限制值，整个 Job 将被标记为失败，并且对应某些索引的 Pod 甚至可能从不曾被启动。该特性可以在某些索引值的 Pod 失败的情况下，仍完成执行所有索引值的 Pod，并且通过避免对持续失败的、特定索引值的Pod进行不必要的重试，更好地利用计算资源。

- 原生边车容器达到Beta

  在Kubernetes1.29版本，原生边车容器特性达到Beta版本。其在initContainers中新增了restartPolicy字段，当配置为Always时表示启用边车容器。边车容器和业务容器部署在同一个Pod中，但并不会延长Pod的生命周期。边车容器常用于网络代理、日志收集等场景。更多使用细节请参考**边车容器**。

- 传统ServiceAccount令牌清理器达到Beta

  在Kubernetes1.29版本，传统ServiceAccount令牌清理器特性达到Beta版本。其作为kube-controller-manager的一部分运行，每24小时检查一次，查看是否有任何自动生成的传统ServiceAccount令牌在特定时间段内（默认为一年，通过--legacy-service-account-token-clean-up-period指定）未被使用。如果有的话，清理器会将这些令牌标记为无效，并添加kubernetes.io/legacy-token-invalid-since标签，其值为当前日期。如果一个无效的令牌在特定时间段（默认为1年，通过--legacy-service-account-token-clean-up-period指定）内未被使用，清理器将会删除它。更多使用细节请参考**传统ServiceAccount令牌清理器**。

- DevicePluginCDIDevices达到Beta

  在Kubernetes1.29版本，DevicePluginCDIDevices特性达到Beta版本。该特性在DeviceRunContainerOptions增加CDIDevices字段，使得设备插件开发者可以直接将CDI设备名称传递给支持CDI的容器运行时。

- PodHostIPs达到Beta

  在Kubernetes1.29版本中，PodHostIPs特性达到Beta版本。该特性在Pod和downward API的Status中增加hostIPs字段，用于将节点IP地址暴露给工作负载。该字段是hostIP的双栈协议版本，第一个IP始终与hostIP相同。

- API优先级和公平性达到GA

  在Kubernetes1.29版本，API优先级和公平性（APF）特性达到GA版本。APF以更细粒度的方式对请求进行分类和隔离，提升最大并发限制，并且它还引入了空间有限的排队机制，因此在非常短暂的突发情况下，API 服务器不会拒绝任何请求。 通过使用公平排队技术从队列中分发请求，这样，一个行为不佳的控制器就不会导致其他控制器异常（即使优先级相同）。更多使用细节请参考**API优先级和公平性**。

- APIListChunking达到GA

  在Kubernetes1.29版本，APIListChunking特性达到GA版本。该特性允许客户端在List请求中进行分页，避免一次性返回过多数据而导致的性能问题。

- ServiceNodePortStaticSubrange达到GA

  在Kubernetes1.29版本，ServiceNodePortStaticSubrange特性达到GA版本。该特性kubelet会根据Nodeport范围计算出预留地址大小，并将Nodeport划分为静态段和动态段。在Nodeport自动分配时，优先分配在动态段，这有助于减小指定静态段分配时的端口冲突。更多使用细节请参考**ServiceNodePortStaticSubrange**

- PersistentVolume的阶段转换时间戳达到Beta

  在Kubernetes1.29版本，PersistentVolume的阶段转换时间戳特性达到Beta版本。该特性在PV的status中添加了一个lastPhaseTransitionTime字段，表示PV上一次phase变化的时间。通过该字段，集群管理员可以跟踪PV上次转换到不同阶段的时间，从而实现更高效、更明智的资源管理。更多使用细节请参考**PersistentVolume的阶段转换时间戳**。

- ReadWriteOncePod达到GA

  在Kubernetes1.29版本中，ReadWriteOncePod特性达到GA版本。该特性允许用户在PVC中配置访问模式为ReadWriteOncePod，确保同时只有一个 Pod能够修改存储中的数据，以防止数据冲突或损坏。更多使用细节请参考**ReadWriteOncePod**。

- CSINodeExpandSecret达到GA

  在Kubernetes1.29版本中，CSINodeExpandSecret特性达到GA版本。该特性允许在添加节点时将Secret身份验证数据传递到CSI驱动以供后者使用。

- CRD验证表达式语言达到GA

  在Kubernetes1.29版本中，CRD验证表达式语言特性达到GA版本。该特性允许用户在CRD中使用通用表达式语言（CEL）定义校验规则，相比webhook更加高效。更多使用细节请参考**CRD校验规则**。

API 变更与弃用

- 在Kubernetes1.29版本中，新创建的CronJob不再支持在.spec.schedule中通过TZ或者CRON_TZ配置时区，请使用.spec.timeZone替代。已经创建的CronJob不受此影响。

- 在Kubernetes1.29版本中，移除了alpha API ClusterCIDR。
- 在Kubernetes1.29版本中，kube-apiserver新增启动参数--authentication-config，用于指定AuthenticationConfiguration文件地址，该启动参数与--oidc-*启动参数互斥。
- 在Kubernetes1.29版本中，移除了KubeSchedulerConfiguration的API版本kubescheduler.config.k8s.io/v1beta3，请迁移至kubescheduler.config.k8s.io/v1。
- 在Kubernetes1.29版本中，将CEL表达式添加到v1alpha1 AuthenticationConfiguration中。
- 在Kubernetes1.29版本中，新增对象ServiceCIDR，允许用户动态配置集群分配Service的ClusterIP时所使用的地址范围。
- 在Kubernetes1.29版本中，kube-proxy新增启动参数--conntrack-udp-timeout、--conntrack-udp-timeout-stream，可对nf_conntrack_udp_timeout和nf_conntrack_udp_timeout_stream内核参数进行设置 。
- 在Kubernetes1.29版本中，将CEL表达式的支持添加到v1alpha1 AuthenticationConfiguration的WebhookMatchCondition中。
- 在Kubernetes1.29版本中，PVC.spec.Resource的类型由原先的ResourceRequirements替换为VolumeResourceRequirements。
- 在Kubernetes1.29版本中，将PodFailurePolicyRule中的OnPodConditions转变为可选字段。
- 在Kubernetes1.29版本中，FlowSchema与PriorityLevelConfiguration的API版本flowcontrol.apiserver.k8s.io/v1beta3已升级至flowcontrol.apiserver.k8s.io/v1，并进行了以下更改
  - PriorityLevelConfiguration：.spec.limited.nominalConcurrencyShares字段在省略时自动设为默认值30，并且为了确保与1.28兼容，在1.29中v1版本该字段不允许显示指定为0。在1.30中，将允许v1版本该字段显示指定为0。flowcontrol.apiserver.k8s.io/v1beta3已废弃，并在1.32中不再支持。
- 在Kubernetes1.29版本中， 优化了kube-proxy的命令行文档，kube-proxy实际上不会将任何socket绑定到由--bind-address启动参数指定的IP。
- 在Kubernetes1.29版本中，移除了selectorSpread调度器插件，使用podTopologySpread替代。
- 在Kubernetes1.29版本中，当CSI-Node-Driver没有在运行时，NodeStageVolume操作会重试。
- 在Kubernetes1.29版本中，ValidatingAdmissionPolicy支持对CRD资源进行校验。使用该特性需开启特性门控ValidatingAdmissionPolicy。
- 在Kubernetes1.29版本中，kube-proxy新增启动参数--nf-conntrack-tcp-be-liberal，可对内核参数nf_conntrack_tcp_be_liberal进行配置。
- 在Kubernetes1.29版本中，kube-proxy新增启动参数--init-only，设置后使kube-proxy的init容器在特权模式下运行，进行一些初始化配置，然后退出。
- 在Kubernetes1.29版本中，CRI的返回体中新增容器的fileSystem字段，表示容器的fileSystem使用信息，而原先只包含镜像的fileSystem。
- 在Kubernetes1.29版本中，所有内置的CloudProvider全部默认设置为关闭，如果仍需使用，可通过配置DisableCloudProviders和DisableKubeletCloudCredentialProvider特性门控来选择性关闭或者打开。
- 在Kubernetes1.29版本中，kubelet可通过--node-ips来设置双栈IP（kubelet设置--cloud-provider=external，此时可以使用--node-ips参数来设置节点地址(IPv4/IPv6)），使用该特性需开启特性门控CloudDualStackNodeIPs。

CCE 对 Kubernetes 1.29 版本的增强

在版本维护周期中，CCE会对Kubernetes 1.29版本进行定期的更新，并提供功能增强。

关于CCE集群版本的更新说明，请参见**补丁版本发布说明**。

参考链接

关于Kubernetes 1.29与其他版本的性能对比和功能演进的更多信息，请参考：
Kubernetes v1.29 Release Notes

2.1.2.3 Kubernetes 1.28 版本说明

云容器引擎（CCE）严格遵循社区一致性认证，现已支持创建Kubernetes 1.28集群。本文介绍Kubernetes 1.28版本的变更说明。

索引

重要说明

- 在Kubernetes 1.28版本，调度框架发生变化，减少无用的重试，从而提高调度程序的整体性能。如果开发人员在集群中使用了自定义调度程序插件，请参见**调度框架变化**进行适配升级。
- 在Kubernetes 1.28 版本，Ceph FS 树内插件已在 v1.28 中弃用，并计划在 v1.31 中删除（社区没有计划进行 CSI 迁移）。 建议使用 **Ceph CSI** 第三方存储驱动程序作为替代方案。
- 在Kubernetes 1.28 版本，Ceph RBD 树内插件已在 v1.28 中弃用，并计划在 v1.31 中删除（社区没有计划进行 CSI 迁移）。建议使用 RBD 模式的 **Ceph CSI** 第三方存储驱动程序作为替代方案。

新增特性及特性增强

社区特性的Alpha阶段默认禁用、Beta阶段一般默认启用、GA阶段将一直默认启用，且不能禁用（会在后续版本中删除这个开关功能）。CCE对新特性的策略与社区保持一致。

- 版本偏差策略扩展至3个版本

  从1.28控制平面/1.25工作节点开始，Kubernetes版本偏差策略将支持的控制平面/工作节点偏差扩展到3个版本。 这使得节点的年度次要版本升级成为可能，同时保持受支持的次要版本。更多细节请参考**版本偏差策略**。
- 可追溯的默认StorageClass进阶至GA

在Kubernetes 1.28版本，可追溯默认StorageClass赋值现已进阶至GA。 这项增强特性极大地改进了默认的StorageClasses为PersistentVolumeClaim（ PVC ）赋值的方式。

PersistentVolume（ PV ）控制器已修改为：当未设置storageClassName时，自动向任何未绑定的PersistentVolumeClaim分配一个默认的StorageClass。此外，API服务器中的PersistentVolumeClaim准入验证机制也已调整为允许将值从未设置状态更改为实际的StorageClass名称。更多使用细节请参考**默认StorageClass赋值**。

- 原生边车容器（ Alpha ）

  在Kubernetes 1.28版本，原生边车容器以Alpha版本正式发布。其在Init容器中添加了一个新的restartPolicy字段， 该字段在SidecarContainers特性门控启用时可用。需要注意的是，原生边车容器目前仍有些问题需要解决，因此K8S社区建议仅在Alpha阶段的**短期测试集群**中使用边车功能。 更多使用细节请参考**原生边车容器**。

- 混合版本代理（ Alpha ）

  在Kubernetes 1.28版本，发布了用于改进集群安全升级的新机制（ 混合版本代理 ）。该特性为Alpha特性。当集群进行升级时，集群中不同版本的kube-apiserver为不同的内置资源集（ 组、版本、资源 ）提供服务。在这种情况下资源请求如果由任一可用的apiserver提供服务，请求可能会到达无法解析此请求资源的apiserver中，导致请求失败。该特性能解决该问题。（ 主要注意的是，CCE本身提供的升级能力即可做到无损升级，因此不存在该特性涉及的场景 ）。更多使用细节请参考**混合版本代理**。

- 节点非体面关闭特性达到GA

  在Kubernetes 1.28版本，节点非体面关闭特性达到GA阶段。当一个节点被关闭但没有被Kubelet的Node Shutdown Manager检测到时，StatefulSet的Pod将会停留在终止状态，并且不能移动到新运行的节点上。当用户确认该节点已经处于不可恢复的情况下，可以手动为Node打上out-of-service的污点，以使得该节点上的StatefulSet的Pod和VolumeAttachments被强制删除，并在健康的Node上创建相应的Pod。更多使用细节请参考**节点非体面关闭**。

- NodeSwap特性达到Beta

  在Kubernetes 1.28版本，NodeSwap能力进阶至Beta版本。目前仍然处于默认关闭状态，需要使用NodeSwap门控打开。该特性可以为Linux节点上运行的Kubernetes工作负载逐个节点地配置内存交换。需要注意的是，该特性虽然进阶至Beta特性，但仍然存在一些需要增强的问题和安全风险。更多使用细节请参考**NodeSwap特性**。

- Job相关特性

  在Kubernetes 1.28版本，增加了**Pod更换策略**和基于**带索引Job的回退限制**两个alpha特性。

  - Pod更换策略

    默认情况下，当Pod进入终止（ Terminating ）状态（ 例如由于抢占或驱逐机制 ）时，Kubernetes会立即创建一个替换的Pod，因此这时会有两个Pod同时运行。

    在Kubernetes 1.28版本中可以使用JobPodReplacementPolicy 来启用该特性。可以在Job的Spec中定义podReplacementPolicy，目前仅可设置为Failed。在设置为Failed之后，Pod仅在达到Failed阶段时才会被替换，而不是在它们处于终止过程中（ Terminating ）时被替换。此外，您可以检查Job的.status.termination字段。该字段的值表示终止过程中的Job所关联的Pod数量。

- 带索引Job的回退限制

  默认情况下，带索引的Job（Indexed Job）的 Pod 失败情况会被记录下来，受.spec.backoffLimit字段所设置的全局重试次数限制。这意味着，如果存在某个索引值的Pod一直持续失败，则Pod会被重新启动，直到重试次数达到限制值。一旦达到限制值，整个Job将被标记为失败，并且对应某些索引的Pod甚至可能从不曾被启动。

  在Kubernetes 1.28版本中，可以通过启用集群的JobBackoffLimitPerIndex特性门控来启用此特性。开启之后，允许在创建带索引的Job（Indexed Job）时指定.spec.backoffLimitPerIndex字段。当某个Job的失败次数超过设定的上限时，将不再进行重试。

- CEL相关特性

  在Kubernetes 1.28版本，CEL能力进行了相应的增强。

  - CRD使用CEL进行Validate的特性进阶至Beta

    该特性在v1.25版本就已经升级为Beta版本。通过将CEL表达式直接集成在CRD中，可以使开发者在不使用Webhook的情况下解决大部分对CR实例进行验证的用例。在未来的版本，将继续扩展CEL表达式的功能，以支持默认值和CRD转换。

  - 基于CEL的准入控制进阶至Beta

    基于通用表达式语言 (CEL) 的准入控制是可定制的，对于kube-apiserver接受到的请求，可以使用CEL表达式来决定是否接受或拒绝请求，可作为Webhook准入控制的一种替代方案。在 v1.28 中，CEL准入控制被升级为Beta，同时添加了一些新功能，包括但不限于：

    - ValidatingAdmissionPolicy类型检查现在可以正确处理CEL表达式中的“authorizer”变量。
    - ValidatingAdmissionPolicy支持对messageExpression字段进行类型检查。
    - kube-controller-manager组件新增ValidatingAdmissionPolicy控制器，用来对ValidatingAdmissionPolicy中的CEL表达式做类型检查，并将原因保存在状态字段中。
    - 支持变量组合，可以在ValidatingAdmissionPolicy中定义变量，然后在定义其他变量时使用它。
    - 新增CEL库函数支持对Kubernetes的resource.Quantity类型进行解析。

- 其它特性说明

  - 在Kubernetes 1.28版本，ServiceNodePortStaticSubrange 特性为beta，允许保留静态端口范围，避免与动态分配端口冲突。具体细节请参考**为NodePort Service分配端口时避免冲突**。
  - 在Kubernetes 1.28版本，增加了alpha特性ConsistentListFromCache，允许kube-apiserver从缓存中提供一致性列表，Get和List请求可以从缓存中读取数据，而不需要从etcd中获取。
  - 在Kubernetes 1.28版本，kubelet能够配置drop-in目录（alpha特性）。该特性允许向kubelet添加对“--config-dir”标志的支持，以允许用户指定一个插入目录，该目录将覆盖位于/etc/kubernetes/kubelet.conf位置的Kubelet的配置。
  - 在Kubernetes 1.28版本，ExpandedDNSConfig升级至GA，默认会被打开。该参数用于允许扩展DNS的配置。

  - 在Kubernetes 1.28版本，提供Alpha特性CRD Validation Ratcheting。该特性允许Patch或者Update请求没有更改任何不合法的字段，将允许CR验证失败。
  - 在Kubernetes 1.28版本，kube-controller-manager添加了--concurrent-cron-job-syncs flag用来设置cron job controller的workers数。

API 变更与弃用

- 在Kubernetes 1.28版本，移除特性NetworkPolicyStatus，因此Network Policy不再有status属性。
- 在Kubernetes 1.28版本，Job对象中增加了新的annotationbatch.kubernetes.io/cronJob-scheduled-timestamp，表示Job的创建时间。
- 在Kubernetes 1.28版本，Job API中添加podReplacementPolicy和terminating字段，当前一旦先前创建的pod终止，Job就会立即启动替换pod。添加字段允许用户指定是在先前的Pod终止后立即更换Pod（原行为），还是在现有的Pod完全终止后才替换Pod（新行为）。这是一项Alpha级别特性，您可以通过在集群中启用**JobPodReplacementPolicy** 来启用该特性。
- 在Kubernetes 1.28版本，Job支持BackoffLimitPerIndex字段。当前使用的运行Job的策略是Job中的整个Pod共享一个Backoff机制，当Job达到Backoff的限制时，整个Job都会被标记为失败，并清理资源，包括尚未运行的index。此字段允许对单个的index设置Backoff。更多信息请参见**带索引Job的Backoff限制**。
- 在Kubernetes 1.28版本，添加ServedVersions字段到 StorageVersion API中。该变化由混合代理版本特性引入。该增加字段ServedVersions用于表明API服务器可以提供的版本。
- 在Kubernetes 1.28版本，SelfSubjectReview 添加到authentication.k8s.io/v1中，并且kubectl auth whoami走向GA。
- 在Kubernetes 1.28版本，PersistentVolume有了一个新的字段LastPhaseTransitionTime，用来保存最近一次volume转变Phase的时间。
- 在Kubernetes 1.28版本，PVC.Status中移除resizeStatus，使用AllocatedResourceStatus替代。resizeStatus表示调整存储大小操作的状态，默认为空字符串。
- 在Kubernetes 1.28版本，设置了hostNetwork: true并且定义了ports的Pods，自动设置hostport字段。
- 在Kubernetes 1.28版本，StatefulSet的Pod索引设置为Pod的标签statefulset.kubernetes.io/pod-index。
- 在Kubernetes 1.28版本，Pod的Condition字段中的PodHasNetwork重命名为PodReadyToStartContainers，用来表明网络、卷等已成功创建，sandbox pod已经创建完成，可以启动容器。
- 在Kubernetes 1.28版本，在KubeSchedulerConfiguration中添加了新的配置选项delayCacheUntilActive， 该参数为true时，非master节点的kube-scheduler不会缓存调度信息。这为非主节点的内存减缓了压力，但会导致主节点发生故障时，减慢故障转移的速度。
- 在Kubernetes 1.28版本，在admissionregistration.k8s.io/v1alpha1.ValidatingAdmissionPolicy中添加namespaceParamRef字段。
- 在Kubernetes 1.28版本，在CRD validation rules中添加reason和fieldPath，允许用户指定验证失败的原因和字段路径。
- 在Kubernetes 1.28版本，ValidatingAdmissionPolicy的CEL表达式通过namespaceObject支持namespace访问。

- 在Kubernetes 1.28版本，将API groups ValidatingAdmissionPolicy 和 ValidatingAdmissionPolicyBinding提升到betav1。
- 在Kubernetes 1.28版本，ValidatingAdmissionPolicy 扩展了messageExpression字段，用来检查已解析类型。

特性门禁及命令行参数

- 在Kubernetes 1.28版本，kubelet移除了flag –short。因此kubectl version 默认输出与kubectl version –short相同。
- 在Kubernetes 1.28版本，kube-controller-manager废弃flag--volume-host-cidr-denylist和--volume-host-allow-local-loopback。--volume-host-cidr-denylist是用逗号分隔的一个CIDR范围列表，禁止使用这些地址上的卷插件。--volume-host-allow-local-loopback为false时，禁止本地回路IP地址和--volume-host-cidr-denylist中所指定的CIDR范围。
- 在Kubernetes 1.28版本，kubelet --azure-container-registry-config被弃用并在未来的版本中会被删除。 请使用--image-credential-provider-config和--image-credential-provider-bin-dir来设置。
- 在Kubernetes 1.28版本，kube-scheduler: 删除了--lock-object-namespace和--lock-object-name。 请使用--leader-elect-resource-namespace和--leader-elect-resource-name或ComponentConfig来配置这些参数。（ --lock-object-namespace用来定义锁对象的命名空间，--lock-object-name用来定义锁对象的名称 ）
- 在Kubernetes 1.28版本，KMSv1已弃用，以后只会接收安全更新。请改用KMSv2。在未来版本中，设置--feature-gates=KMSv1=true以使用已弃用的KMSv1功能。
- 在Kubernetes 1.28版本，移除了如下特性门禁：DelegateFSGroupToCSIDriver、DevicePlugins、KubeletCredentialProviders、MixedProtocolLBService、ServiceInternalTrafficPolicy、ServiceIPStaticSubrange、EndpointSliceTerminatingCondition。

CCE 对 Kubernetes 1.28 版本的增强

在版本维护周期中，CCE会对Kubernetes 1.28版本进行定期的更新，并提供功能增强。

关于CCE集群版本的更新说明，请参见**CCE集群版本发布说明**。

参考链接

关于Kubernetes 1.28与其他版本的性能对比和功能演进的更多信息，请参考：
Kubernetes v1.28 Release Notes

2.1.2.4 Kubernetes 1.27 版本说明

云容器引擎（CCE）严格遵循社区一致性认证，现已支持创建Kubernetes 1.27集群。本文介绍Kubernetes 1.27版本相对于1.25版本所做的变更说明。

索引

- **主要特性**
- **弃用和移除**

- **CCE对Kubernetes 1.27版本的增强**
- **参考链接**

主要特性

Kubernetes 1.27版本

- SeccompDefault特性已进入稳定阶段

  如需使用SeccompDefault特性，您需要为每个节点的kubelet启用--seccomp-default**命令行标志**。如果启用该特性，kubelet将为所有工作负载默认使用RuntimeDefault seccomp配置文件，该配置文件由容器运行时定义，而不是使用Unconfined（禁用seccomp）模式。

- Job可变调度指令

  该特性在Kubernetes 1.22版本中引入，当前已进入稳定阶段。在大多数情况下，并行作业Pod希望在一定的约束下运行，例如希望所有Pod在同一可用区。该特性允许在Job开始前修改调度指令。您可以使用suspend字段挂起Job，在Job挂起阶段，Pod模板中的调度部分（例如节点选择器、节点亲和性、反亲和性、容忍度）允许修改。详情请参见**可变调度指令**。

- Downward API HugePages已进入稳定阶段

  在Kubernetes 1.20版本中，**Downward API**引入了*requests.hugepages-<pagesize>*和*limits.hugepages-<pagesize>*，HugePage可以和其他资源一样设置资源配额。

- Pod调度就绪态进入Beta阶段

  Pod创建后，Kubernetes调度程序会负责选择合适的节点运行pending状态的Pod。在实际使用时，一些Pod可能会由于资源不足长时间处于pending状态。这些Pod可能会影响集群中的其他组件运行（如Cluster Autoscaler）。通过指定/删除Pod的.spec.schedulingGates，您可以控制Pod何时准备好进行调度。详情请参见**Pod调度就绪态**。

- 通过Kubernetes API访问节点日志

  此功能当前处于Alpha阶段。集群管理员可以直接查询节点上的服务日志，可以帮助调试节点上运行的服务问题。如需使用此功能，请确保在该节点上启用了NodeLogQuery**特性门控**，并且kubelet配置选项enableSystemLogHandler和enableSystemLogQuery都设置为true。

- ReadWriteOncePod访问模式进入Beta阶段

  在Kubernetes 1.22版本中，PV和PVC提供了一种新的访问模式ReadWriteOncePod，该功能当前进入Beta阶段。卷可以被单个Pod以读写方式挂载。如果您想确保整个集群中只有一个Pod可以读取或写入该PVC，请使用ReadWriteOncePod访问模式，详情请参见**访问模式**。

- Pod拓扑分布约束中matchLabelKeys字段进入Beta阶段

  matchLabelKeys是一个Pod标签键的列表，用于选择需要计算分布方式的Pod集合。使用matchLabelKeys字段，您无需在变更Pod修订版本时更新pod.spec。控制器或Operator只需要将不同修订版的标签键设为不同的值。调度器将根据matchLabelKeys自动确定取值。详情请参见**Pod拓扑分布约束**。

- 快速标记SELinux卷标签功能进入Beta阶段

  默认情况下，容器运行时递归地将SELinux标签赋予所有Pod卷上的所有文件。 为了加快该过程，Kubernetes使用挂载可选项*-o context=<label>*可以立即改变卷的SELinux标签。详情请参见**快速标记SELinux卷标签**。

- VolumeManager重构进入Beta阶段

  重构的VolumeManager后，如果启用NewVolumeManagerReconstruction**特性门控**，将会在kubelet启动期间使用更有效的方式来获取已挂载卷。

- 服务器端字段校验和OpenAPI V3已进入稳定阶段

  Kubernetes 1.23中添加了对OpenAPI v3的支持，1.24版本中已进入Beta阶段，1.27已进入稳定阶段。

- 控制StatefulSet启动序号

  Kubernetes 1.26为StatefulSet引入了一个新的Alpha级别特性，可以控制Pod副本的序号。 从Kubernetes 1.27开始，此特性进入Beta阶段，序数可以从任意非负数开始。详情请参见**Kubernetes 1.27: StatefulSet启动序号简化了迁移**。

- HorizontalPodAutoscaler ContainerResource类型指标进入Beta阶段

  Kubernetes 1.20在HorizontalPodAutoscaler (HPA) 中引入了**ContainerResource类型指标**。在Kubernetes 1.27中，此特性进阶至Beta，相应的特性门控 (HPAContainerMetrics) 默认被启用。

- StatefulSet PVC自动删除进入Beta阶段

  Kubernetes v1.27提供一种新的策略机制，用于控制StatefulSets的PersistentVolumeClaims（PVCs）的生命周期。这种新的PVC保留策略允许用户指定当删除StatefulSet或者缩减StatefulSet中的副本时，是自动删除还是保留从StatefulSet规约模板生成的PVC。详情请参见**PersistentVolumeClaim保留**。

- 磁盘卷组快照

  磁盘卷组快照在Kubernetes 1.27中作为Alpha特性被引入。此特性允许用户对多个卷进行快照，以保证在发生故障时数据的一致性。 它使用标签选择器来将多个PersistentVolumeClaims分组以进行快照。 这个新特性仅支持CSI卷驱动器。详情请参见**Kubernetes 1.27：介绍用于磁盘卷组快照的新API**。

- kubectl apply裁剪更安全、更高效

  在Kubernetes 1.5版本中，kubectl apply引入了--prune标志来删除不再需要的资源，允许kubectl apply自动清理从当前配置中删除的资源。然而，现有的--prune实现存在设计缺陷，会降低性能并导致意外行为。Kubernetes 1.27中，kubectl apply提供基于ApplySet的剪裁方式，当前处于Alpha阶段，详情请参见**使用配置文件对Kubernetes对象进行声明式管理**。

- 为NodePort Service分配端口时避免冲突

  在Kubernetes 1.27中，您可以启用新的**特性门控** ServiceNodePortStaticSubrange，为NodePort Service使用不同的端口分配策略，减少冲突的风险。当前该特性处于Alpha阶段。

- 原地调整Pod资源

  在Kubernetes 1.27中，允许用户调整分配给Pod的CPU和内存资源大小，而无需重新启动容器。当前该特性处于Alpha阶段，详情请参见**纵向弹性伸缩**。

- 加快Pod启动

  在Kubernetes 1.27中进行了一系列的参数调整，以提高Pod的启动速度，例如并行镜像拉取、提高Kubelet默认API每秒查询限值等。详情请参见**Kubernetes 1.27：关于加快Pod启动的进展**。

- KMS V2进入Beta阶段

  Kubernetes中的密钥管理KMS v2 API进入Beta阶段，对KMS加密提供程序的性能进行了重大改进。详情请参见**使用KMS驱动进行数据加密**。

Kubernetes 1.26版本

- 移除CRI v1alpha2

  Kubernetes 1.26版本不再支持CRI v1alpha2，请使用v1（要求containerd版本>=1.5.0）。这意味着Kubernetes 1.26将不支持containerd 1.5.x及更早的版本；需要升级到containerd 1.6.x或更高版本后，才能将该节点的kubelet升级到1.26。

  说明

  CCE目前使用的containerd版本为1.6.14，已满足要求。如存量的节点不满足containerd版本要求，请将节点重置为最新版本。

- 动态资源分配 Alpha API

  在Kubernetes 1.26版本，新增**动态资源分配**功能，用于Pod之间和Pod内部容器之间请求和共享资源，支持用户提供参数初始化资源。该功能尚处于alpha阶段，需要启用DynamicResourceAllocation特性门禁和resource.k8s.io/v1alpha1 API组，需要为要管理的特定资源安装驱动程序。更多信息，请参见**Kubernetes 1.26: 动态资源分配 Alpha API**。

- 节点非体面关闭进入Beta阶段

  在Kubernetes 1.26 中，节点非体面关闭特性是Beta版，默认被启用。当kubelet的节点关闭管理器可以检测到即将到来的节点关闭操作时，节点关闭才被认为是体面的。详情请参见**处理节点非体面关闭**。

- 支持在挂载时将Pod fsGroup传递给CSI驱动程序

  将fsGroup委托给CSI驱动程序管理首先在Kubernetes 1.22中作为Alpha特性引入， 并在Kubernetes 1.25中进阶至Beta状态。该特性在Kubernetes 1.26已进入正式发布阶段，详情请参见**将卷权限和所有权更改委派给CSI驱动程序**。

- Pod调度就绪态

  Kubernetes 1.26引入了一个新的Pod特性schedulingGates，可以让调度器感知到何时可以进行Pod调度。详情请参见**Pod调度就绪态**。

- CPU Manager正式发布

  CPU管理器是kubelet的一部分，从Kubernetes 1.10**进阶至 Beta**，能够将独占CPU分配给容器。该特性在Kubernetes 1.26已进入稳定阶段，详情请参见**控制节点上的CPU管理策略**。

- Kubernetes中流量工程的进步

  优化内部节点本地流量和**支持EndpointSlice终止状况**升级为正式发布版本，**ProxyTerminatingEndpoints**功能升级为Beta版本。

- 支持跨命名空间存储数据源

  Kubernetes 1.26允许在源数据属于不同的命名空间时为PersistentVolumeClaim指定数据源。当前该特性处于Alpha阶段，详情请参见**跨命名空间数据源**。

- 可追溯的默认StorageClass进入Beta阶段

  Kubernetes 1.25引入了一个Alpha特性来更改默认StorageClass被分配到PersistentVolumeClaim (PVC) 的方式。 启用此特性后，您不再需要先创建默认StorageClass，再创建PVC来分配类。 此外，任何未分配StorageClass的PVC都可以在后续被更新。此特性在Kubernetes 1.26 中已进入Beta阶段，详情请参见**可追溯的默认StorageClass赋值**。

- PodDisruptionBudget支持指定不健康Pod的驱逐策略

  Kubernetes 1.26允许针对**PodDisruptionBudget** (PDB) 指定不健康Pod驱逐策略，这有助于在节点执行管理操作期间保持可用性。当前该特性处于Beta阶段，详情请参见**不健康的Pod驱逐策略**。

- 支持设置水平伸缩Pod控制器的数量

kube-controller-manager支持flag --concurrent-horizontal-pod-autoscaler-syncs设置水平伸缩Pod控制器的worker数量。详情请参见**集群配置管理**。

弃用和移除

Kubernetes 1.27版本

- 在Kubernetes 1.27版本，针对卷扩展GA特性的以下特性门禁将被移除，且不得再在--feature-gates标志中引用。（**ExpandCSIVolumes，ExpandInUsePersistentVolumes，ExpandPersistentVolumes**）
- 在Kubernetes 1.27版本，移除--master-service-namespace 命令行参数。该参数支持指定在何处创建名为kubernetes的Service来表示API服务器。自v1.26版本已被弃用，1.27版本正式移除。
- 在Kubernetes 1.27版本，移除ControllerManagerLeaderMigration特性门禁。**Leader Migration**提供了一种机制，让HA集群在升级多副本的控制平面时通过在kube-controller-manager和cloud-controller-manager这两个组件之间共享的资源锁，安全地迁移“特定于云平台”的控制器。特性自v1.24正式发布，被无条件启用， 在v1.27版本中此特性门禁选项将被移除。
- 在Kubernetes 1.27版本，移除--enable-taint-manager命令行参数。该参数支持的特性基于污点的驱逐已被默认启用，且在标志被移除时也将继续被隐式启用。
- 在Kubernetes 1.27版本，移除--pod-eviction-timeout 命令行参数。弃用的命令行参数--pod-eviction-timeout将被从kube-controller-manager中移除。
- 在Kubernetes 1.27版本，移除CSI Migration特性门禁。**CSI migration**程序允许从树内卷插件移动到树外CSI驱动程序。CSI迁移自Kubernetes v1.16起正式发布，关联的CSIMigration特性门禁将在v1.27中被移除。
- 在Kubernetes 1.27版本，移除CSIInlineVolume特性门禁。**CSI Ephemeral Volume**特性允许在Pod规约中直接指定CSI卷作为临时使用场景。这些CSI卷可用于使用挂载的卷直接在Pod内注入任意状态，例如配置、Secret、身份、变量或类似信息。 此特性在v1.25中进阶至正式发布。因此，此特性门禁CSIInlineVolume将在v1.27版本中移除。
- 在Kubernetes 1.27版本，移除EphemeralContainers特性门禁。对于Kubernetes v1.27，临时容器的API支持被无条件启用；EphemeralContainers特性门禁将被移除。
- 在Kubernetes 1.27版本，移除LocalStorageCapacityIsolation特性门禁。**Local Ephemeral Storage Capacity Isolation**特性在 v1.25 中进阶至正式发布。此特性支持emptyDir卷这类Pod之间本地临时存储的容量隔离，因此可以硬性限制Pod对共享资源的消耗。如果本地临时存储的消耗超过了配置的限制，kubelet将驱逐Pod。特性门禁LocalStorageCapacityIsolation将在v1.27版本中被移除。
- 在Kubernetes 1.27版本，移除NetworkPolicyEndPort特性门禁。Kubernetes v1.25版本将NetworkPolicy中的endPort进阶至正式发布。支持endPort字段的NetworkPolicy提供程序可用于指定一系列端口以应用NetworkPolicy。
- 在Kubernetes 1.27版本，移除StatefulSetMinReadySeconds特性门禁。对于作为StatefulSet一部分的Pod，只有当Pod至少在**minReadySeconds**中指定的持续期内可用（并通过检查）时，Kubernetes才会将此Pod标记为只读。 该特性在Kubernetes v1.25中正式发布，StatefulSetMinReadySeconds特性门禁将锁定为true，并在v1.27版本中被移除。
- 在Kubernetes 1.27版本，移除IdentifyPodOS特性门禁。启用该特性门禁，您可以为Pod指定操作系统，此项特性支持自v1.25版本进入稳定。IdentifyPodOS特性门禁将在Kubernetes v1.27中被移除。

- 在Kubernetes 1.27版本，移除DaemonSetUpdateSurge特性门禁。Kubernetes v1.25版本还稳定了对DaemonSet Pod的浪涌支持，其实现是为了最大限度地减少部署期间DaemonSet的停机时间。DaemonSetUpdateSurge特性门禁将在Kubernetes v1.27中被移除。
- 在Kubernetes 1.27版本，移除--container-runtime 命令行参数。kubelet 接受一个已弃用的命令行参数--container-runtime， 并且在移除dockershim代码后，唯一有效的值将是remote。 Kubernetes v1.27将移除该参数，该参数自v1.24版本以来已被弃用。

Kubernetes 1.26版本

- 移除v2beta2版本的HorizontalPodAutoscaler API

  HorizontalPodAutoscaler的autoscaling/v2beta2 API版本将不再在1.26版本中提供，详情请参见**各发行版本中移除的API**。 用户应迁移至autoscaling/v2版本的API。
- 移除v1beta1版本的流量控制API组

  在Kubernetes 1.26版本后，开始不再提供flowcontrol.apiserver.k8s.io/v1beta1 API版本的FlowSchema和PriorityLevelConfiguration，详情请参见**各发行版本中移除的API**。但此API从Kubernetes 1.23版本开始，可以使用flowcontrol.apiserver.k8s.io/v1beta2；从Kubernetes 1.26版本开始，可以使用flowcontrol.apiserver.k8s.io/v1beta3。
- 存储驱动的弃用和移除，移除云服务厂商的in-tree卷驱动。
- 移除kube-proxy userspace模式

  在Kubernetes 1.26版本，Userspace代理模式已被移除，已弃用的Userspace代理模式不再受Linux或Windows支持。Linux用户应使用Iptables或IPVS，Windows用户应使用Kernelspace，现在使用--mode userspace会失败。

  – Windows winkernel kube-proxy不再支持Windows HNS v1 APIs。
- 弃用--prune-whitelist标志

  在Kubernetes 1.26版本，为了支持**Inclusive Naming Initiative**，--prune-whitelist标志将被**弃用**，并替换为--prune-allowlist，该标志在未来将彻底移除。
- 移除动态Kubelet配置

  DynamicKubeletConfig特性门控移除，通过API动态更新节点上的Kubelet配置。在Kubernetes 1.24版本中从Kubelet移除相关代码，在Kubernetes 1.26版本从APIServer移除相关代码，移除该逻辑有助于简化代码提升可靠性，推荐方式是修改Kubelet配置文件然后重启Kubelet。更多信息，请参见**在Kubernetes 1.26版本从APIServer移除相关代码**。
- 弃用kube-apiserver命令行参数

  在Kubernetes 1.26版本，正式标记**弃用 --master-service-namespace** 命令行参数，它对APIServer没有任何效果。
- 弃用kubectl run命令行参数

  在Kubernetes 1.26版本，kubectl run未使用的几个子命令将被标记为**弃用**，并在未来某个版本移除，包括--cascade、--filename、--force、--grace-period、--kustomize、--recursive、--timeout、--wait等这些子命令。
- 移除与日志相关的原有命令行参数

  在Kubernetes 1.26版本，将**移除**一些与日志相关的命令行参数，这些参数在之前的版本已被**弃用**。

CCE 对 Kubernetes 1.27 版本的增强

在版本维护周期中，CCE会对Kubernetes 1.27版本进行定期的更新，并提供功能增强。

关于CCE集群版本的更新说明，请参见**CCE集群版本发布说明**。

参考链接

关于Kubernetes 1.27与其他版本的性能对比和功能演进的更多信息，请参考：

- **Kubernetes v1.27 Release Notes**
- **Kubernetes v1.26 Release Notes**

2.1.2.5 Kubernetes 1.25 版本说明

云容器引擎（CCE）严格遵循社区一致性认证。本文介绍Kubernetes 1.25版本相对于1.23版本所做的变更说明。

索引

主要特性

Kubernetes 1.25版本

- Pod Security Admission进入稳定阶段，并移除PodSecurityPolicy

  PodSecurityPolicy被废弃，并提供Pod Security Admission取代，具体的迁移方法可参见**从PodSecurityPolicy迁移到内置的PodSecurity准入控制器**。

- Ephemeral Containers进入稳定阶段

  临时容器是在现有的Pod中存在有限时间的容器。它对故障排除特别有用，特别是当需要检查另一个容器，但因为该容器已经崩溃或其镜像缺乏调试工具不能使用kubectl exec时。

- 对cgroups v2的支持进入稳定阶段

  Kubernetes支持cgroups v2，与cgroups v1相比提供了一些改进，详情请参见**cgroups v2**。

- SeccompDefault提升到Beta状态

  如果要开启该特性，需要给kubelet增加启动参数为--seccomp-default=true，这样会默认开启seccomp为RuntimeDefault，提升整个系统的安全。1.25集群将不再支持使用注解“seccomp.security.alpha.kubernetes.io/pod”和“container.seccomp.security.alpha.kubernetes.io/annotation”来使用seccomp，请使用pod或container中“securityContext.seccompProfile”字段替代，详情请参见**为Pod或容器配置安全上下文**。

  说明

  特性开启后可能应用所需的系统调用会被runtime限制，所以开启后应确保在测试环境调试，不会对应用造成影响。

- 网络策略中的EndPort进入稳定阶段

  Network Policy中的EndPort已进入稳定状态，该特性于1.21版本合入。主要是在NetworkPolicy新增EndPort，可以指定一个Port范围，避免声明每一个Port。

- 本地临时容器存储容量隔离进入稳定阶段

  本地临时存储容量隔离功能提供了对Pod之间本地临时存储容量隔离的支持，如EmptyDir。因此，如果一个Pod对本地临时存储容量的消耗超过该限制，就可以通过驱逐Pod来硬性限制其对共享资源的消耗。

- CRD验证表达式语言升级为Beta阶段

  CRD验证表达式语言已升级为 beta 版本， 这使得声明如何使用**通用表达式语言（CEL）**验证自定义资源成为可能。 请参考**验证规则**指导。

- 引入KMS v2 API

  在Kubernetes 1.25版本，引入KMS v2 alpha1 API以提升性能，实现轮替与可观察性改进。 此API使用AES-GCM替代了AES-CBC，通过DEK实现静态数据加密（Kubernetes Secrets），此过程中无需您额外操作，且支持通过AES-GCM和AES-CBC进行读取。 更多信息，请参考**使用 KMS provider进行数据加密指南**。

- Pod新增网络就绪状况

  Kubernetes 1.25引入了对kubelet所管理的新的Pod状况PodHasNetwork的Alpha支持， 该状况位于Pod的status字段中 。详情请参见**Pod网络就绪**。

- 应用滚动上线所用的两个特性进入稳定阶段
  - 在Kubernetes 1.25版本，StatefulSet的minReadySeconds进入稳定阶段，允许每个Pod等待一段预期时间来减缓StatefulSet的滚动上线。更多信息，请参见**最短就绪秒数**。
  - 在Kubernetes 1.25版本，DaemonSet的maxSurge进入稳定阶段，允许DaemonSet工作负载在滚动上线期间在一个节点上运行同一 Pod的多个实例，有助于将DaemonSet的停机时间降到最低。DaemonSet不允许maxSurge和hostPort同时使用，因为两个活跃的Pod无法共享同一节点的相同端口。更多信息，请参见**DaemonSet工作负载滚动上线**。

- 对使用用户命名空间运行Pod提供Alpha支持

  对使用user namespace运行Pod提供alpha支持，将Pod内的root用户映射到容器外的非零ID，使得从容器角度看是root身份运行，而从主机角度看是常规的非特权用户。目前尚处于内测阶段，需要开启特性门控UserNamespacesStatelessPodsSupport， 且要求容器运行时必须能够支持此功能。更多信息，请参见**对使用user namespace运行Pod提供alpha支持**。

Kubernetes 1.24版本

- 从kubelet中删除 Dockershim

  Dockershim自1.20版本被标废弃以来，在1.24版本正式从Kubelet代码中移除。如果还想使用Docker作为容器运行时的话，需要切换到cri-dockerd，或者使用其他支持CRI的运行时比如Containerd/CRI-O等。

  从Docker Engine 切换到Containerd的流程请参见**将节点容器引擎从Docker迁移到Containerd**。

  说明

  您需要注意排查是否有agent或者应用强依赖Docker Engine的，比如在代码中使用docker ps，docker run， docker inspect等，需要注意兼容多种runtime，以及切换到标准cri接口。

- Beta APIs默认关闭

在社区移除一些长期Beta API的过程中发现，90%的集群管理员并没有关心Beta API默认开始，其实Beta特性是不推荐在生产环境中使用，但是因为默认的打开策略，导致这些API在生产环境中都被默认开启，这样会因为Beta特性的bug带来一些风险，以及升级的迁移的风险。所以在1.24版本开始，Beta API默认关闭，之前已经默认开启的Beta API会保持默认开启。

- 支持OpenAPI v3

  在Kubernetes 1.24版本后，OpenAPI V3默认开启。

- 存储容量跟踪特性进入稳定阶段

  在Kubernetes 1.24版本后，CSIStorageCapacity API支持显示当前可用的存储大小，确保Pod调度到足够存储容量的节点上，减少Volumes创建和挂载失败导致的Pod调度延迟，详细信息请参见**存储容量**。

- gRPC 探针升级到Beta阶段

  在Kubernetes 1.24版本后，gRPC探针进入Beta，默认可用特性门控参数GRPCContainerProbe，使用方式请参见**配置探针**。

- 特性门控LegacyServiceAccountTokenNoAutoGeneration默认启用

  LegacyServiceAccountTokenNoAutoGeneration特性门控进入beta状态，默认为开启状态，开启后将不再为Service Account自动生成Secret Token。如果需要使用永不过期的Token，需要自己新建Secrets并挂载，详情请参见**服务账号令牌Secret**。

- 避免 IP 分配给服务的冲突

  Kubernetes 1.24引入了一项新功能，允许**为服务的静态IP地址分配软保留范围**。通过手动启用此功能，集群将从服务IP地址池中自动分配IP，从而降低冲突风险。

- 基于Go 1.18编译

  在Kubernetes 1.24版本后，Kubernetes基于Go 1.18编译，默认不再支持SHA-1哈希算法验证证书签名，例如SHA1WithRSA、ECDSAWithSHA1算法，推荐使用SHA256算法生成的证书进行认证。

- StatefulSet支持设置最大不可用副本数

  在Kubernetes 1.24版本后，StatefulSets支持可配置maxUnavailable参数，使得滚动更新时可以更快地停止Pods。

- 节点非体面关闭进入Alpha阶段

  在Kubernetes 1.24中，节点非体面关闭特性是Alpha版。当kubelet的节点关闭管理器可以检测到即将到来的节点关闭操作时，节点关闭才被认为是体面的。详情请参见**处理节点非体面关闭**。

弃用和移除

Kubernetes 1.25版本

- 清理iptables链的所有权

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

  在Kubernetes 1.25版本后，Kubelet通过IPTablesCleanup特性门控分阶段完成迁移，是为了不在NAT表中创建iptables链，例如KUBE-MARK-DROP、KUBE-MARK-MASQ、KUBE-POSTROUTING。关于清理iptables链所有权的信息，请参见**清理IPTables链的所有权**。

- 存储驱动的弃用和移除，移除云服务厂商的in-tree卷驱动。

Kubernetes 1.24版本

- 在Kubernetes 1.24版本后，Service.Spec.LoadBalancerIP被弃用，因为它无法用于双栈协议。请使用自定义annotation。
- 在Kubernetes 1.24版本后，kube-apiserver移除参数--address、--insecure-bind-address、--port、--insecure-port=0。
- 在Kubernetes 1.24版本后，kube-controller-manager和kube-scheduler移除启动参数--port=0和--address。
- 在Kubernetes 1.24版本后，kube-apiserver --audit-log-version和--audit-webhook-version仅支持audit.k8s.io/v1，Kubernetes 1.24移除audit.k8s.io/v1[alpha|beta]1，只能使用audit.k8s.io/v1。
- 在Kubernetes 1.24版本后，kubelet移除启动参数--network-plugin，仅当容器运行环境设置为Docker时，此特定于Docker的参数才有效，并会随着Dockershim一起删除。
- 在Kubernetes 1.24版本后，动态日志清理功能已经被废弃，并在Kubernetes 1.24版本移除。该功能引入了一个日志过滤器，可以应用于所有Kubernetes系统组件的日志，以防止各种类型的敏感信息通过日志泄漏。此功能可能导致日志阻塞，所以废弃，更多信息请参见**Dynamic log sanitization**和 **KEP-1753**。
- VolumeSnapshot v1beta1 CRD在Kubernetes 1.20版本中被废弃，在Kubernetes 1.24版本中移除，需改用v1版本。
- 在Kubernetes 1.24版本后，移除自1.11版本就废弃的service annotation tolerate-unready-endpoints，使用Service.spec.publishNotReadyAddresses代替。
- 在Kubernetes 1.24版本后，废弃metadata.clusterName字段，并将在下一个版本中删除。
- Kubernetes 1.24及以后的版本，去除了kube-proxy监听NodePort的逻辑，在NodePort与内核net.ipv4.ip_local_port_range范围有冲突的情况下，可能会导致偶发的TCP无法连接的情况，导致健康检查失败、业务异常等问题。升级前，请确保集群没有NodePort端口与任意节点net.ipv4.ip_local_port_range范围存在冲突。更多信息，请参见**Kubernetes社区PR**。

CCE 对 Kubernetes 1.25 版本的增强

在版本维护周期中，CCE会对Kubernetes 1.25版本进行定期的更新，并提供功能增强。

关于CCE集群版本的更新说明，请参见**CCE集群版本发布说明**。

参考链接

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

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

2.1.2.6 Kubernetes 1.23 版本说明

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

资源变更与弃用

社区1.23 ReleaseNotes

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

社区1.22 ReleaseNotes

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

参考链接

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

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

2.1.2.7 （停止维护）Kubernetes 1.21 版本说明

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

资源变更与弃用

社区1.21 ReleaseNotes

- CronJob现在已达到稳定状态，版本号变为batch/v1。

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

社区1.20 ReleaseNotes

- API优先级和公平性已达到测试状态，默认启用。这允许kube-apiserver按优先级对传入请求进行分类。更多信息，请参见**API Priority and Fairness**。
- 修复 exec probe timeouts不生效的BUG，在此修复之前，exec 探测器不考虑 timeoutSeconds 字段。相反，探测将无限期运行，甚至超过其配置的截止日期，直到返回结果。 通过此更改，如果未指定值，将使用默认值，默认值为1秒。如果探测时间超过一秒，可能会导致应用健康检查失败。请在升级时确定使用该特性的应用更新timeoutSeconds字段。新引入的 ExecProbeTimeout 特性门控所提供的修复使集群操作员能够恢复到以前的行为，但这种行为将在后续版本中锁定并删除。
- RuntimeClass已达到稳定状态。RuntimeClass资源提供了一种机制，用于支持集群中的多个运行时，并将有关该容器运行时的信息公开到控制平面。
- kubectl调试已达到测试状态。kubectl调试直接从kubectl提供对常见调试工作流的支持。
- Dockershim在1.20被标记为废弃，目前您可以继续在集群中使用Docker。该变动与集群所使用的容器镜像（Image）无关。您依然可以使用Docker构建您的镜像。更多信息，请参见**Dockershim Deprecation FAQ**。

参考链接

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

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

2.1.2.8 （停止维护）Kubernetes 1.19 版本说明

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

资源变更与弃用

社区1.19 ReleaseNotes

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

社区1.18 ReleaseNotes

- kube-apiserver
  - apps/v1beta1 and apps/v1beta2下所有资源不再提供服务，使用apps/v1替代。
  - extensions/v1beta1下daemonsets，deployments，replicasets不再提供服务，使用apps/v1替代。
  - extensions/v1beta1下networkpolicies不再提供服务，使用networking.k8s.io/v1替代。
  - extensions/v1beta1下podsecuritypolicies不再提供服务，使用policy/v1beta1替代。
- kubelet
  - --redirect-container-streaming不推荐使用，v1.20会正式废弃。
  - 资源度量端点 /metrics/resource/v1alpha1以及此端点下的所有度量标准均已弃用。请转换为端点 /metrics/resource下的度量标准：
    - scrape_error --> scrape_error
    - node_cpu_usage_seconds_total --> node_cpu_usage_seconds
    - node_memory_working_set_bytes --> node_memory_working_set_bytes
    - container_cpu_usage_seconds_total --> container_cpu_usage_seconds
    - container_memory_working_set_bytes --> container_memory_working_set_bytes

    - scrape_error --> scrape_error
  - 在将来的发行版中，kubelet将不再根据CSI规范创建CSI NodePublishVolume目标目录。可能需要相应地更新CSI驱动程序，以正确创建和处理目标路径。
- kube-proxy
  - --healthz-port和--metrics-port参数不建议使用，请使用--healthz-bind-address和--metrics-bind-address。
  - 增加EndpointSliceProxying功能选项以控制kube-proxy中EndpointSlices的使用，默认情况下已禁用此功能。
- kubeadm
  - kubeadm upgrade node的--kubelet-version参数已弃用，将在后续版本中删除。
  - kubeadm alpha certs renew命令中--use-api参数已弃用。
  - kube-dns已弃用，在将来的版本中将不再受支持。
  - kubeadm-config ConfigMap中存在的ClusterStatus结构体已废弃，将在后续版本中删除。
- kubectl
  - --dry-run不建议使用boolean和unset values，新版本中server|client|none会被使用。
  - kubectl apply --server-dry-run已弃用，替换为--dry-run=server。
- add-ons

删除cluster-monitoring插件。

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

参考链接

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

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

2.1.2.9 （停止维护）Kubernetes 1.17 版本说明

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

资源变更与弃用

- apps/v1beta1和apps/v1beta2下所有资源不再提供服务，使用apps/v1替代。
- extensions/v1beta1下daemonsets、deployments、replicasets不再提供服务，使用apps/v1替代。
- extensions/v1beta1下networkpolicies不再提供服务，使用networking.k8s.io/v1替代。
- extensions/v1beta1下podsecuritypolicies不再提供服务，使用policy/v1beta1替代。
- extensions/v1beta1 ingress v1.20版本不再提供服务，当前可使用networking.k8s.io/v1beta1。
- scheduling.k8s.io/v1beta1 and scheduling.k8s.io/v1alpha1下的PriorityClass计划在1.17不再提供服务，迁移至scheduling.k8s.io/v1。
- events.k8s.io/v1beta1中event series.state字段已废弃，将在1.18版本中移除。
- apiextensions.k8s.io/v1beta1下CustomResourceDefinition已废弃，将在1.19不再提供服务，使用apiextensions.k8s.io/v1。
- admissionregistration.k8s.io/v1beta1 MutatingWebhookConfiguration和ValidatingWebhookConfiguration已废弃，将在1.19不再提供服务，使用admissionregistration.k8s.io/v1替换。
- rbac.authorization.k8s.io/v1alpha1 and rbac.authorization.k8s.io/v1beta1被废弃，使用rbac.authorization.k8s.io/v1替代，v1.20会正式停止服务。
- storage.k8s.io/v1beta1 CSINode object废弃并会在未来版本中移除。

其他废弃和移除

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

参考链接

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

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

2.1.2.10 （停止维护）Kubernetes 1.15 版本说明

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

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

版本说明

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

资源变更与弃用

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

参考链接

社区v1.13与v1.15版本之间的CHANGELOG

- v1.14到v1.15的变化：

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

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

2.1.2.11 （停止维护）Kubernetes 1.13 版本说明

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

表 2-2 v1.13 版本集群说明

Kubernetes版本（CCE增强版）	版本说明
v1.13.10-r0	**主要特性：** • CCE集群支持添加ARM节点 • 负载均衡支持设置名称 • 4层负载均衡支持健康检查，7层负载均衡支持健康检查/分配策略/会话保持 • CCE集群支持创建裸金属节点（容器隧道网络） • 支持AI加速型节点（搭载海思Ascend 310 AI处理器），适用于图像识别、视频处理、推理计算以及机器学习等场景 • 支持配置docker baseSize • 支持命名空间亲和调度 • 支持节点数据盘划分用户空间 • 支持集群cpu管理策略 • 支持集群下的节点跨子网（容器隧道网络）
v1.13.7-r0	**主要特性：** • Kubernetes同步社区1.13.7版本 • 支持网络平面（NetworkAttachmentDefinition）

参考链接

社区v1.11与v1.13版本之间的CHANGELOG

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

2.1.2.12 （停止维护）Kubernetes 1.11 版本说明

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

表 2-3 v1.11 版本集群说明

Kubernetes版本（CCE增强版）	版本说明
v1.11.7-r2	**主要特性：** • GPU支持V100类型 • 集群支持权限管理
v1.11.7-r0	**主要特性：** • Kubernetes同步社区1.11.7版本 • 支持创建节点池（nodepool），虚拟机/鲲鹏ARM集群均支持 • CCE集群支持创建裸金属节点（VPC网络），支持裸金属和虚机混合部署 • GPU支持V100类型 • 1.11集群对接AOM告警通知机制 • Service支持访问类型切换 • 支持服务网段 • 集群支持自定义每个节点分配的IP数（IP分配）
v1.11.3-r2	**主要特性：** • 集群支持IPv6双栈 • ELB负载均衡支持源IP跟后端服务会话保持
v1.11.3-r1	**主要特性：** • Ingress的URL匹配支持Perl语法的正则表达式
v1.11.3-r0	**主要特性：** • Kubernetes同步社区1.11.3版本 • 集群控制节点支持多可用区 • 容器存储支持对接SFS Turbo极速文件存储

参考链接

社区v1.9与v1.11版本之间的CHANGELOG

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

2.1.2.13 （停止维护）Kubernetes 1.9 及之前版本说明

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

表 2-4 v1.9 及之前版本集群说明

Kubernetes版本（CCE增强版）	版本说明
v1.9.10-r2	**主要特性：** ● ELB负载均衡支持源IP跟后端服务会话保持
v1.9.10-r1	**主要特性：** ● 支持对接SFS存储 ● 支持Service自动创建二代ELB ● 支持公网二代ELB透传源IP ● 支持设置节点最大实例数maxPods
v1.9.10-r0	**主要特性：** ● kubernetes对接ELB/Ingress，新增流控机制 ● Kubernetes同步社区1.9.10版本 ● 支持Kubernetes RBAC能力授权 **问题修复：** ● 修复操作系统cgroup内核BUG导致概率出现的节点内存泄漏问题
v1.9.7-r1	**主要特性：** ● 增强PVC和PV事件的上报机制，PVC详情页支持查看事件 ● 支持对接第三方认证系统 ● 集群支持纳管EulerOS2.3的物理机 ● 数据盘支持用户自定义分配比例 ● 裸金属场景支持对接EVS云硬盘存储 ● 裸金属场景下支持IB网卡 ● 裸金属场景支持通过CM-v3接口创建节点
v1.9.7-r0	**主要特性：** ● 新建集群的Docker版本升级到1706 ● 支持DNS级联 ● 支持插件化管理 ● Kubernetes同步社区1.9.7版本 ● 支持7层ingress的https功能 ● 有状态工作负载支持迁移调度更新升级

Kubernetes版本（CCE增强版）	版本说明
v1.9.2-r3	**主要特性：** ● 集群支持创建/纳管CentOS7.4操作系统的节点 ● kubernetes的Service支持对接DNAT网关服务 ● NetworkPolicy能力开放 ● 增强型ELB支持Service配置多个端口 **问题修复：** ● 修复kubernetes资源回收过程中连不上kube-apiserver导致pod残留的问题 ● 修复节点弹性扩容数据不准确的问题
v1.9.2-r2	**主要特性：** ● 经典型ELB支持自定义健康检查端口 ● 经典型ELB性能优化 ● ELB四层负载均衡支持修改Service的端口 **问题修复：** ● 修复网络插件防止健康检查概率死锁问题 ● 修复高可用集群haproxy连接数限制问题
v1.9.2-r1	**主要特性：** ● Kubernetes同步社区1.9.2版本 ● 集群节点支持CentOS 7.1操作系统 ● 支持GPU节点，支持GPU资源限制 ● 支持web-terminal插件
v1.7.3-r13	**主要特性：** ● 新建集群的Docker版本升级到1706 ● 支持DNS级联 ● 支持插件化管理 ● 增强PVC和PV事件的上报机制 ● 裸金属场景支持对接OBS对象存储

Kubernetes版本（CCE增强版）	版本说明
v1.7.3-r12	**主要特性：** ● 集群支持创建/纳管CentOS7.4操作系统的节点 ● kubernetes的Service支持对接DNAT网关服务 ● NetworkPolicy能力开放 ● 增强型ELB支持Service配置多个端口 **问题修复：** ● 修复kubernetes资源回收过程中连不上kube-apiserver导致pod残留的问题 ● 修复节点弹性扩容数据不准确的问题 ● 事件老化周期提示修正：集群老化周期为1小时
v1.7.3-r11	**主要特性：** ● 经典型ELB支持自定义健康检查端口 ● 经典型ELB性能优化 ● ELB四层负载均衡支持修改Service的端口 ● 支持删除命名空间 ● 支持EVS云硬盘存储解绑 ● 支持配置迁移策略 **问题修复：** ● 修复网络插件防止健康检查概率死锁问题 ● 修复高可用集群haproxy连接数限制问题
v1.7.3-r10	**主要特性：** ● 容器网络支持Overlay L2模式 ● 集群节点支持GPU类型虚机 ● 集群节点支持CentOS 7.1操作系统，支持操作系统选择 ● Windows集群支持对接二代ELB ● 支持弹性文件服务SFS导入 ● 裸金属场景支持对接SFS文件存储
v1.7.3-r9	**主要特性：** ● 工作负载支持跨AZ部署 ● 容器存储支持OBS对象存储服务 ● 支持ELB L7负载均衡 ● Windows集群支持EVS存储 ● 裸金属场景支持devicemapper direct-lvm模式

Kubernetes版本（CCE增强版）	版本说明
v1.7.3-r8	**主要特性：** ● 集群支持节点弹性扩容 ● 支持纳管ARM节点
v1.7.3-r7	**主要特性：** ● 容器隧道网络集群支持纳管SUSE 12sp2节点 ● docker支持direct-lvm模式挂载devicemapper ● 集群支持安装dashboard ● 支持创建Windows集群
v1.7.3-r6	**主要特性：** ● 集群存储对接原生EVS接口
v1.7.3-r5	**主要特性：** ● 支持创建HA高可靠集群 **问题修复：** ● 节点重启后容器网络不通
v1.7.3-r4	**主要特性：** ● 集群性能优化 ● 裸金属场景支持对接ELB
v1.7.3-r3	**主要特性：** ● 容器存储支持KVM虚拟机挂载
v1.7.3-r2	**主要特性：** ● 容器存储支持SFS文件存储 ● 工作负载支持自定义应用日志 ● 开放工作负载优雅缩容 **问题修复：** ● 修复容器存储AK/SK会过期的问题
v1.7.3-r1	**主要特性：** ● kube-dns支持外部域名解析
v1.7.3-r0	**主要特性：** ● Kubernetes同步社区1.7.3版本 ● 支持ELB负载均衡 ● 容器存储支持XEN虚拟机挂载 ● 容器存储支持EVS云硬盘存储

2.1.3 补丁版本发布记录

索引

v1.30 版本

表 2-5 v1.30 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.30.4-r0	v1.30.4	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。
v1.30.1-r2	v1.30.2	-	增强系统稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.30.1-r0	v1.30.2	首次发布CCE v1.30集群，有关更多信息请参见Kubernetes 1.30版本说明。 • CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。

v1.29 版本

表 2-6 v1.29 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.29.8-r0	v1.29.8	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。
v1.29.4-r2	v1.29.3	-	增强系统稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.29.4-r0	v1.29.3	• CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。
v1.29.3-r0	v1.29.3	• 支持RAM实现跨账号密钥共享。 • 自定义节点池支持迁移节点。 • 支持Flexus云服务器X实例。	• 节点池配置管理新增containerd默认镜像地址配置项。 • CCE节点池列表页面支持自定义排序。	修复部分安全问题。
v1.29.2-r4	v1.29.3	-	增强了早期版本集群在跨度较大的升级场景下负载均衡服务的稳定性。	修复部分安全问题。
v1.29.2-r2	v1.29.3	-	增强了集群升级场景的容器日志采集的可靠性。	修复部分安全问题。
v1.29.2-r0	v1.29.3	• CCE Ingress支持基于HTTP头部自定义Header进行流量分发。 • 支持为第三方工作负载应用配置扩缩容优先级策略。 • （仅CCE Turbo集群）支持使用annotation为Pod配置安全组。 • （仅CCE Turbo集群）为Pod绑定EIP时支持使用已有EIP。	• 节点排水过程支持取消。 • 创建节点池时不再区分按需计费和包年/包月计费。 • 更新节点池时支持修改委托及前后缀名称。 • 通过控制台重置节点将默认保留K8s标签和污点。 • 开放Kubernetes服务账号令牌卷投射配置和负载弹性伸缩控制器配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.29.1-r10	v1.29.1	• CCE Ingress转发策略支持优先级排序。 • StatefulSet配置volumeClaimTemplates时可以配置PV和底层存储的名称前缀（Everest版本要求为2.4.15及以上）。 • 包周期节点池支持选择多个节点规格。 • 更新节点池时支持删除默认规格。 • 自定义节点池支持纳管节点。	• 增加CoreDNS解析失败的告警规则。 • 节点退订前支持排水操作。 • 节点池更新后将展示各节点的配置差异点。	修复部分安全问题。
v1.29.1-r0	v1.29.1	首次发布CCE v1.29集群，有关更多信息请参见Kubernetes 1.29版本说明。	-	-

v1.28 版本

表 2-7 v1.28 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.28.13-r0	v1.28.13	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.28.8-r2	v1.28.8	-	增强系统稳定性。	修复部分安全问题。
v1.28.8-r0	v1.28.8	• CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。
v1.28.7-r2	v1.28.8	• 支持RAM实现跨账号密钥共享。 • 自定义节点池支持迁移节点。 • 支持Flexus云服务器X实例。	• 节点池配置管理新增containerd默认镜像地址配置项。 • CCE节点池列表页面支持自定义排序。	修复部分安全问题。
v1.28.6-r4	v1.28.8	-	增强了早期版本集群在跨度较大的升级场景下负载均衡服务的稳定性。	修复部分安全问题。
v1.28.6-r2	v1.28.8	-	增强了集群升级场景的容器日志采集的可靠性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.28.6-r0	v1.28.8	• CCE Ingress支持基于HTTP头部自定义Header进行流量分发。 • 支持为第三方工作负载应用配置扩缩容优先级策略。 • （仅CCE Turbo集群）支持使用annotation为Pod配置安全组。 • （仅CCE Turbo集群）为Pod绑定EIP时支持使用已有EIP。	• 节点排水过程支持取消。 • 创建节点池时不再区分按需计费和包年/包月计费。 • 更新节点池时支持修改委托及前后缀名称。 • 通过控制台重置节点将默认保留K8s标签和污点。 • 开放Kubernetes服务账号令牌卷投射配置和负载弹性伸缩控制器配置。	修复部分安全问题。
v1.28.5-r0	v1.28.5	• CCE Ingress转发策略支持优先级排序。 • StatefulSet配置volumeClaimTemplates时可以配置PV和底层存储的名称前缀（Everest版本要求为2.4.15及以上）。 • 包周期节点池支持选择多个节点规格。 • 更新节点池时支持删除默认规格。 • 自定义节点池支持纳管节点。	• 增加CoreDNS解析失败的告警规则。 • 节点退订前支持排水操作。 • 节点池更新后将展示各节点的配置差异点。	修复部分安全问题。
v1.28.4-r0	v1.28.5	• 创建节点支持选择Docker容器引擎。 • 支持使用通用型SSD v2类型的云硬盘。 • ELB Ingress支持配置灰度发布。 • ELB Ingress支持配置URL重定向、Rewrite重写、HTTP重定向到HTTPS能力。	开放高频使用的集群参数、节点池参数配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.28.3-r0	v1.28.3	负载均衡类型的Service和ELB Ingress能力新增： • 支持配置SNI。 • 支持开启HTTP/2。 • 支持配置空闲超时时间、请求超时时间、响应超时时间。 • 支持从HTTP报文的请求头中获取监听器端口号、客户端请求端口号、重写X-Forwarded-Host。	-	修复部分安全问题。
v1.28.2-r0	v1.28.3	• 创建Service或Ingress支持设置ELB黑/白名单访问控制。 • CCE的节点镜像支持安全加固（满足等保三级基线要求）。	-	修复部分安全问题。
v1.28.1-r4	v1.28.3	-	-	修复CVE-2024-21626安全漏洞。
v1.28.1-r2	v1.28.3	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.28.1-r0	v1.28.3	首次发布CCE v1.28集群，有关更多信息请参见Kubernetes 1.28版本说明。 • 节点池支持节点的自定义前缀和后缀命名 • CCE Turbo集群中，支持创建工作负载类型的容器网络配置，可指定Pod子网。 • ELB Ingress支持GRPC协议。 • 负载均衡类型的服务在通过YAML创建时支持指定ELB私有IP。	• 优化CCE Turbo集群中大批量创建安全容器的启动速度。 • 提升CCE Turbo集群中反复创建删除安全容器时的稳定性。	-

v1.27 版本

须知

Kubernetes在1.24版本中移除了Dockershim，并从此不再默认支持Docker容器引擎，建议您使用Containerd容器引擎。如果您需要将Docker节点迁移至Containerd节点，详情请参见**将节点容器引擎从Docker迁移到Containerd**。

表 2-8 v1.27 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.27.16-r0	v1.27.16	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。
v1.27.10-r2	v1.27.12	-	增强系统稳定性。	修复部分安全问题。
v1.27.10-r0	v1.27.12	• CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。
v1.27.9-r0	v1.27.12	• 支持RAM实现跨账号密钥共享。 • 自定义节点池支持迁移节点。 • 支持Flexus云服务器X实例。	• 节点池配置管理新增containerd默认镜像地址配置项。 • CCE节点池列表页面支持自定义排序。	修复部分安全问题。
v1.27.8-r4	v1.27.12	-	增强了早期版本集群在跨度较大的升级场景下负载均衡服务的稳定性。	修复部分安全问题。
v1.27.8-r2	v1.27.12	-	增强了集群升级场景的容器日志采集的可靠性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.27.8-r0	v1.27.12	• CCE Ingress支持基于HTTP头部自定义Header进行流量分发。 • 支持为第三方工作负载应用配置扩缩容优先级策略。 • （仅CCE Turbo集群）支持使用annotation为Pod配置安全组。 • （仅CCE Turbo集群）为Pod绑定EIP时支持使用已有EIP。	• 节点排水过程支持取消。 • 创建节点池时不再区分按需计费和包年/包月计费。 • 更新节点池时支持修改委托及前后缀名称。 • 通过控制台重置节点将默认保留K8s标签和污点。 • 开放Kubernetes服务账号令牌卷投射配置和负载弹性伸缩控制器配置。	修复部分安全问题。
v1.27.7-r0	v1.27.9	• CCE Ingress转发策略支持优先级排序。 • StatefulSet配置volumeClaimTemplates时可以配置PV和底层存储的名称前缀（Everest版本要求为2.4.15及以上）。 • 包周期节点池支持选择多个节点规格。 • 更新节点池时支持删除默认规格。 • 自定义节点池支持纳管节点。	• 增加CoreDNS解析失败的告警规则。 • 节点退订前支持排水操作。 • 节点池更新后将展示各节点的配置差异点。	修复部分安全问题。
v1.27.6-r0	v1.27.9	• 创建节点支持选择Docker容器引擎。 • 支持使用通用型SSD v2类型的云硬盘。 • ELB Ingress支持配置灰度发布。 • ELB Ingress支持配置URL重定向、Rewrite重写、HTTP重定向到HTTPS能力。	开放高频使用的集群参数、节点池参数配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.27.5-r0	v1.27.4	负载均衡类型的Service和ELB Ingress能力新增： • 支持配置SNI。 • 支持开启HTTP/2。 • 支持配置空闲超时时间、请求超时时间、响应超时时间。 • 支持从HTTP报文的请求头中获取监听器端口号、客户端请求端口号、重写X-Forwarded-Host。	-	修复部分安全问题。
v1.27.4-r0	v1.27.4	• 创建Service或Ingress支持设置ELB黑/白名单访问控制。 • CCE的节点镜像支持安全加固（满足等保三级基线要求）。	-	修复部分安全问题。
v1.27.3-r4	v1.27.4	-	-	修复CVE-2024-21626安全漏洞。
v1.27.3-r2	v1.27.4	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.27.3-r0	v1.27.4	• 节点池支持节点的自定义前缀和后缀命名 • CCE Turbo集群中，支持创建工作负载类型的容器网络配置，可指定Pod子网。 • ELB Ingress支持GRPC协议。 • 负载均衡类型的服务在通过YAML创建时支持指定ELB私有IP。	• 优化CCE Turbo集群中大批量创建安全容器的启动速度。 • 提升CCE Turbo集群中反复创建删除安全容器时的稳定性。 • 创建Ingress对象时增加配置证书校验，避免对ELB侧已存在的Ingress证书进行覆盖。 • 优化autoscaler扩容节点池时的事件上报逻辑，去除规格售罄的重复事件。 • 增加Service与Ingress端口占用的相互校验逻辑；增加同集群下Ingress的路径冲突的的校验逻辑。	修复部分安全问题。
v1.27.2-r0	v1.27.2	• Volcano支持节点池亲和调度。 • Volcano支持负载重调度能力。	-	修复部分安全问题。
v1.27.1-r10	v1.27.2	-	优化节点池伸缩时的事件信息。	修复部分安全问题。
v1.27.1-r0	v1.27.2	首次发布CCE v1.27集群，有关更多信息请参见**Kubernetes 1.27版本说明**。 • 节点池配置管理支持软驱逐和硬驱逐的设置。 • 支持为自动创建的EVS块存储添加TMS资源标签，以便于成本管理。	由于**社区安全加固**，v1.27及以上版本的集群中ClusterIP地址无法ping通。	-

v1.25 版本

须知

除EulerOS 2.5操作系统外，CCE v1.25集群的节点均默认采用Containerd容器引擎。

表 2-9 v1.25 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.25.16-r0	v1.25.16	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。
v1.25.13-r2	v1.25.16	-	增强系统稳定性。	修复部分安全问题。
v1.25.13-r0	v1.25.16	• CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。
v1.25.12-r0	v1.25.16	• 支持RAM实现跨账号密钥共享。 • 自定义节点池支持迁移节点。 • 支持Flexus云服务器X实例。	• 节点池配置管理新增containerd默认镜像地址配置项。 • CCE节点池列表页面支持自定义排序。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.25.11-r4	v1.25.16	-	增强了早期版本集群在跨度较大的升级场景下负载均衡服务的稳定性。	修复部分安全问题。
v1.25.11-r2	v1.25.16	-	增强了集群升级场景的容器日志采集的可靠性。	修复部分安全问题。
v1.25.11-r0	v1.25.16	• CCE Ingress支持基于HTTP头部自定义Header进行流量分发。 • 支持为第三方工作负载应用配置扩缩容优先级策略。 • （仅CCE Turbo集群）支持使用annotation为Pod配置安全组。 • （仅CCE Turbo集群）为Pod绑定EIP时支持使用已有EIP。	• 节点排水过程支持取消。 • 创建节点池时不再区分按需计费和包年/包月计费。 • 更新节点池时支持修改委托及前后缀名称。 • 通过控制台重置节点将默认保留K8s标签和污点。 • 开放Kubernetes服务账号令牌卷投射配置和负载弹性伸缩控制器配置。	修复部分安全问题。
v1.25.10-r0	v1.25.16	• CCE Ingress转发策略支持优先级排序。 • StatefulSet配置volumeClaimTemplates时可以配置PV和底层存储的名称前缀（Everest版本要求为2.4.15及以上）。 • 包周期节点池支持选择多个节点规格。 • 更新节点池时支持删除默认规格。 • 自定义节点池支持纳管节点。	• 增加CoreDNS解析失败的告警规则。 • 节点退订前支持排水操作。 • 节点池更新后将展示各节点的配置差异点。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.25.9-r0	v1.25.16	• 支持使用通用型SSD v2类型的云硬盘。 • ELB Ingress支持配置灰度发布。 • ELB Ingress支持配置URL重定向、Rewrite重写、HTTP重定向到HTTPS能力。	开放高频使用的集群参数、节点池参数配置。	修复部分安全问题。
v1.25.8-r0	v1.25.10	负载均衡类型的Service和ELB Ingress能力新增： • 支持配置SNI。 • 支持开启HTTP/2。 • 支持配置空闲超时时间、请求超时时间、响应超时时间。 • 支持从HTTP报文的请求头中获取监听器端口号、客户端请求端口号、重写X-Forwarded-Host。	-	修复部分安全问题。
v1.25.7-r0	v1.25.10	• 创建Service或Ingress支持设置ELB黑/白名单访问控制。 • CCE的节点镜像支持安全加固（满足等保三级基线要求）。	-	修复部分安全问题。
v1.25.6-r4	v1.25.10	-	-	修复CVE-2024-21626安全漏洞。
v1.25.6-r2	v1.25.10	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.25.6-r0	v1.25.10	• 节点池支持节点的自定义前缀和后缀命名 • CCE Turbo集群中，支持创建工作负载类型的容器网络配置，可指定Pod子网。 • ELB Ingress支持GRPC协议。 • 负载均衡类型的服务在通过YAML创建时支持指定ELB私有IP。	• 优化CCE Turbo集群中大批量创建安全容器的启动速度。 • 提升CCE Turbo集群中反复创建删除安全容器时的稳定性。 • 修复kubelet在特定场景下偶现启动卡死的问题。 • 优化autoscaler扩容节点池时的事件上报逻辑，去除规格售罄的重复事件。 • 修复特定场景下kubelet重启，出现Succeed状态的Pod变为Failed的问题。	修复部分安全问题。
v1.25.5-r0	v1.25.5	• Volcano支持节点池亲和调度。 • Volcano支持负载重调度能力。	-	修复部分安全问题。
v1.25.4-r10	v1.25.5	-	优化节点池伸缩时的事件信息。	修复部分安全问题。
v1.25.4-r0	v1.25.5	• 节点池配置管理支持软驱逐和硬驱逐的设置。 • 支持为自动创建的EVS块存储添加TMS资源标签，以便于成本管理。	-	修复部分安全问题。
v1.25.3-r10	v1.25.5	• CCE集群支持对接使用弹性规格的独享型ELB。 • 负载均衡支持设置超时时间。	kube-apiserver高频参数支持配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.25.3-r0	v1.25.5	• CCE Turbo容器网卡支持固定IP。详情请参见**为Pod配置固定IP**。 • CCE Turbo容器网卡支持自动创建和自动绑定EIP。详情请参见**为Pod配置固定EIP**。 • CCE Turbo集群在离线混部增强：支持Pod网络优先级限制。详情请参见**出口网络带宽保障**。 • CCE Turbo集群支持命名空间关联容器网段。详情请参见**网络配置（NetworkAttachmentDefinition）**。 • 集群支持CPU Burst特性，避免CPU限流影响时延敏感型容器业务。详情请参见**CPU Burst弹性限流**。	增强CCE Turbo集群在规格变更场景时网络的稳定性。	修复部分安全问题。
v1.25.1-r0	v1.25.5	首次发布CCE v1.25集群，有关更多信息请参见**Kubernetes 1.25版本说明**。	-	-

v1.23 版本

表 2-10 v1.23 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.18-r10	v1.23.18	• ELB Ingress支持根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数进行转发。	• 更新节点池时支持修改节点密码。 • 创建节点时支持只使用系统盘。 • 更新ELB Ingress时支持修改HTTP重定向到HTTPS的配置。 • 使用Docker容器引擎的节点池时支持自定义配置默认镜像地址。	修复部分安全问题。
v1.23.18-r2	v1.23.17	-	增强系统稳定性。	修复部分安全问题。
v1.23.18-r0	v1.23.17	• CCE删除集群，支持勾选删除日志组。 • 创建集群时支持选择用户自己管理的KMS实例来进行Secret加密落盘ETCD。 • 通过私有镜像创建节点，支持保留镜像密码选项。 • CCE支持GPU渲染场景。	CCE支持ELB全端口类型监听器。	修复部分安全问题。
v1.23.17-r0	v1.23.17	• 支持RAM实现跨账号密钥共享。 • 自定义节点池支持迁移节点。 • 支持Flexus云服务器X实例。	• 节点池配置管理新增containerd默认镜像地址配置项。 • CCE节点池列表页面支持自定义排序。	修复部分安全问题。
v1.23.16-r4	v1.23.17	-	增强了早期版本集群在跨度较大的升级场景下负载均衡服务的稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.16-r2	v1.23.17	-	增强了集群升级场景的容器日志采集的可靠性。	修复部分安全问题。
v1.23.16-r0	v1.23.17	• CCE Ingress支持基于HTTP头部自定义Header进行流量分发。 • 支持为第三方工作负载应用配置扩缩容优先级策略。 • （仅CCE Turbo集群）支持使用annotation为Pod配置安全组。 • （仅CCE Turbo集群）为Pod绑定EIP时支持使用已有EIP。	• 节点排水过程支持取消。 • 创建节点池时不再区分按需计费和包年/包月计费。 • 更新节点池时支持修改委托及前后缀名称。 • 通过控制台重置节点将默认保留K8s标签和污点。 • 开放Kubernetes服务账号令牌卷投射配置和负载弹性伸缩控制器配置。	修复部分安全问题。
v1.23.15-r0	v1.23.17	• CCE Ingress转发策略支持优先级排序。 • StatefulSet配置volumeClaimTemplates时可以配置PV和底层存储的名称前缀（Everest版本要求为2.4.15及以上）。 • 包周期节点池支持选择多个节点规格。 • 更新节点池时支持删除默认规格。 • 自定义节点池支持纳管节点。	• 增加CoreDNS解析失败的告警规则。 • 节点退订前支持排水操作。 • 节点池更新后将展示各节点的配置差异点。	修复部分安全问题。
v1.23.14-r0	v1.23.17	• 支持使用通用型SSD v2类型的云硬盘。 • ELB Ingress支持配置灰度发布。 • ELB Ingress支持配置URL重定向、Rewrite重写、HTTP重定向到HTTPS能力。	开放高频使用的集群参数、节点池参数配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.13-r0	v1.23.17	负载均衡类型的Service和ELB Ingress能力新增： • 支持配置SNI。 • 支持开启HTTP/2。 • 支持配置空闲超时时间、请求超时时间、响应超时时间。 • 支持从HTTP报文的请求头中获取监听器端口号、客户端请求端口号、重写X-Forwarded-Host。	-	修复部分安全问题。
v1.23.12-r0	v1.23.17	• 创建Service或Ingress支持设置ELB黑/白名单访问控制。 • CCE的节点镜像支持安全加固（满足等保三级基线要求）。	-	修复部分安全问题。
v1.23.11-r4	v1.23.17	-	-	修复CVE-2024-21626安全漏洞。
v1.23.11-r2	v1.23.17	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.11-r0	v1.23.17	• 节点池支持节点的自定义前缀和后缀命名 • CCE Turbo集群中，支持创建工作负载类型的容器网络配置，可指定Pod子网。 • ELB Ingress支持GRPC协议。 • 负载均衡类型的服务在通过YAML创建时支持指定ELB私有IP。	• 优化CCE Turbo集群中大批量创建安全容器的启动速度。 • 提升CCE Turbo集群中反复创建删除安全容器时的稳定性。 • 修复Docker在journald异常退出场景下导致容器无法被结束的问题。 • 修复Everest插件卸载时，调度器未能在创建Pod时拦截自动挂载SFS3.0存储卷的问题。 • 修复v1.23版本集群中，EulerOS 2.9系统的containerd节点上/var/lib/contained磁盘目录使用率虚高的问题。	修复部分安全问题。
v1.23.10-r0	v1.23.11	• Volcano支持节点池亲和调度。 • Volcano支持负载重调度能力。	-	修复部分安全问题。
v1.23.9-r10	v1.23.11	-	优化节点池伸缩时的事件信息。	修复部分安全问题。
v1.23.9-r0	v1.23.11	• 节点池配置管理支持软驱逐和硬驱逐的设置。 • 支持为自动创建的EVS块存储添加TMS资源标签，以便于成本管理。	-	修复部分安全问题。
v1.23.8-r10	v1.23.11	• CCE集群支持对接使用弹性规格的独享型ELB。 • 负载均衡支持设置超时时间。	kube-apiserver高频参数支持配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.8-r0	v1.23.11	• CCE Turbo容器网卡支持固定IP。详情请参见**为Pod配置固定IP**。 • CCE Turbo容器网卡支持自动创建和自动绑定EIP。详情请参见**为Pod配置固定EIP**。 • CCE Turbo集群在离线混部增强：支持Pod网络优先级限制。详情请参见**出口网络带宽保障**。 • CCE Turbo集群支持命名空间关联容器网段。详情请参见**网络配置（NetworkAttachmentDefinition）**。 • 集群支持CPU Burst特性，避免CPU限流影响时延敏感型容器业务。详情请参见**CPU Burst弹性限流**。	• 增强docker版本升级时的可靠性。 • 优化集群节点时间同步能力。	修复部分安全问题。
v1.23.7-r20	v1.23.11	-	• 增强Service/Ingress对接ELB特性稳定性。 • 增强节点挂载多块数据盘场景可靠性。	修复部分安全问题。
v1.23.7-r10	v1.23.11	-	• 增强docker版本升级时的可靠性。 • 增强containerd运行时在异常断链场景下的可靠性。 • 内核参数调优异常场景下加固。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.7-r0	v1.23.11	• Service和Ingress支持关联G-EIP的独享型ELB。	• 优化CCE Turbo集群在规格变更场景时网络的稳定性。 • 增强集群升级场景，nginx-ingress-controller的网络稳定性。 • 优化集群节点时间同步能力。	修复部分安全问题。
v1.23.6-r0	v1.23.11	• 支持LB类型的Service同时配置TCP/UDP端口 • 支持Pod readiness gate	• 增强底层网络异常时的流表可靠性。 • 增强高版本内核的OS异常掉电等重启场景的稳定性。 • cadvisor GPU/NPU相关指标优化。	修复部分安全问题。
v1.23.5-r0	v1.23.11	• 容器存储支持对接SFS 3.0文件存储服务。 • 支持GPU节点的设备故障检测和隔离能力。 • 支持配置集群维度的自定义安全组。 • CCE Turbo集群支持节点级别的网卡预热参数配置。 • 支持集群控制面组件的日志信息开放。 • 集群支持华为云自研的Huawei Cloud EulerOS 2.0操作系统。 • CCE集群支持选择Containerd容器运行时。 • CCE Turbo集群在离线混部增强：支持CPU潮汐亲和性。	• 优化升级控制节点ETCD版本至社区版本3.5.6。 • 优化EulerOS 2.8节点上Service的访问性能。 • 优化调度均衡性，工作负载实例数缩容时仍保持跨AZ分布均衡。 • 优化kube-apiserver在频繁更新CRD场景下的内存使用。	修复部分安全问题及以下CVE漏洞： • CVE-2022-3294 • CVE-2022-3162 • CVE-2022-3172 • CVE-2021-25749

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.23.4-r10	v1.23.4	-	优化kube-apiserver在频繁更新crd场景下的内存使用。	修复部分安全问题。
v1.23.4-r0	v1.23.4	支持ARM节点创建。	-	修复部分安全问题。
v1.23.3-r0	v1.23.4	• CCE集群支持对租户开放master节点组件监控指标。 • 通过预热机制优化CCE Turbo集群SubENI网卡启动速度。 • 支持ELB类型service配置后端服务器权重。 • CCE集群支持跨集群部署服务。 • CCE Turbo集群使用虚拟机节点场景下支持在离线混部功能。	增强安全容器在反复创删场景下的可靠性。	修复部分安全问题。
v1.23.1-r1	v1.23.4	优化节点的资源预留参数，支持资源耗尽检测，提升节点的稳定性。	提升节点的安装兼容性。	修复部分安全问题。
v1.23.1-r0	v1.23.4	首次发布CCE v1.23集群，有关更多信息请参见**Kubernetes 1.23版本说明**。	-	-

v1.21 版本

表 2-11 v1.21 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.21.15-r0	v1.21.14	• 支持使用通用型SSD v2类型的云硬盘。 • ELB Ingress支持配置灰度发布。 • ELB Ingress支持配置URL重定向、Rewrite重写、HTTP重定向到HTTPS能力。	开放高频使用的集群参数、节点池参数配置。	修复部分安全问题。
v1.21.14-r0	v1.21.14	支持使用PVC动态创建SFS Turbo子目录并挂载。	-	修复部分安全问题。
v1.21.13-r0	v1.21.14	• 创建Service或Ingress支持设置ELB黑/白名单访问控制。 • CCE的节点镜像支持安全加固（满足等保三级基线要求）。	-	修复部分安全问题。
v1.21.12-r4	v1.21.14	-	-	修复CVE-2024-21626安全漏洞。
v1.21.12-r2	v1.21.14	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.21.12-r0	v1.21.14	节点池支持节点的自定义前缀和后缀命名	• 优化健康检查配置，避免出现keepalived反复重启的问题。 • 优化securityPolicy缓存及Ingress重试风暴。 • 修复cloud-controller-manager组件主进程所在Master节点出现卡IO，组件切主失败的问题。 • 修复Service设置权重后，会错误计算terminating状态的Pod数，导致Pod权重不准确的问题。	修复部分安全问题。
v1.21.11-r20	v1.21.14	• Volcano支持节点池亲和调度。 • Volcano支持负载重调度能力。	-	修复部分安全问题。
v1.21.11-r10	v1.21.14	-	优化节点池伸缩时的事件信息。	修复部分安全问题。
v1.21.11-r0	v1.21.14	• 节点池配置管理支持软驱逐和硬驱逐的设置。 • 支持为自动创建的EVS块存储添加TMS资源标签，以便于成本管理。	-	修复部分安全问题。
v1.21.10-r10	v1.21.14	• CCE集群支持对接使用弹性规格的独享型ELB。 • 负载均衡支持设置超时时间。	kube-apiserver高频参数支持配置。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.21.10-r0	v1.21.14	• CCE Turbo容器网卡支持固定IP。详情请参见为Pod配置固定IP。 • CCE Turbo容器网卡支持自动创建和自动绑定EIP。详情请参见为Pod配置固定EIP。	• 增强docker版本升级时的可靠性。 • 优化集群节点时间同步能力。 • 优化节点重启后，docker运行时拉取镜像的稳定性。	修复部分安全问题。
v1.21.9-r0	v1.21.14	• Service和Ingress支持关联G-EIP的独享型ELB。	优化CCE Turbo集群在规格变更场景时网络的稳定性。	修复部分安全问题。
v1.21.8-r0	v1.21.14	• 支持LB类型的Service同时配置TCP/UDP端口 • 支持Pod readiness gate	• 增强底层网络异常时的流表可靠性。 • 增强高版本内核的OS异常掉电等重启场景的稳定性。 • cadvisor GPU/NPU相关指标优化。	修复部分安全问题。
v1.21.7-r0	v1.21.14	• 容器存储支持对接SFS 3.0文件存储服务。 • 支持GPU节点的设备故障检测和隔离能力。 • 支持配置集群维度的自定义安全组。 • CCE Turbo集群支持节点级别的网卡预热参数配置。 • 支持集群控制面组件的日志信息开放。	优化ELB Service/Ingress在大量连接场景下的稳定性。	修复部分安全问题及以下CVE漏洞： • CVE-2022-3294 • CVE-2022-3162 • CVE-2022-3172
v1.21.6-r0	v1.21.7	支持ARM节点创建。	-	修复部分安全问题。
v1.21.5-r10	v1.21.7	-	• 优化隧道网络在控制面节点间网络闪断场景下导网络分配的稳定性。 • ovs漏洞加固。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.21.5-r0	v1.21.7	-	• 增强集群升级下容器隧道网络模式的稳定性。 • 增强pod拓扑分布约束能力。 • 增强集群控制面节点掉电时链接释放的稳定性。 • 增强节点上pod并发挂载存储卷的稳定性。	修复部分安全问题。
v1.21.4-r10	v1.21.7	-	增强EulerOS 2.9操作系统NetworkManager的稳定性。	修复部分安全问题。
v1.21.4-r0	v1.21.7	-	• 增强安全容器场景下容器网卡驱动切换的稳定性。 • 增强升级工作负载且处于节点伸缩时，访问负载均衡服务的稳定性。	修复部分安全问题。
v1.21.3-r10	v1.21.7	-	增强安全容器场景下容器网卡驱动切换的稳定性。	修复部分安全问题。
v1.21.3-r0	v1.21.7	-	CCE Turbo集群容器内的SNAT网段支持可配置。	修复部分安全问题。
v1.21.2-r10	v1.21.7	-	增强Service对接负载均衡场景的稳定性。	修复部分安全问题。
v1.21.2-r0	v1.21.7	优化节点的资源预留参数，支持资源耗尽检测，提升节点的稳定性。	• 提升集群升级能力，增强升级可靠性。 • 优化容器本地存储功能，提升稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.21.1-r2	v1.21.7	• 容器存储支持本地持久卷。 • 支持管理EulerOS 2.9鲲鹏计算实例。 • 容器隧道网络模式和VPC网络模式支持OS内核版本宽匹配。	• 优化节点安装流程，增强节点创建的可靠性。 • 优化CentOS和EulerOS 2.5的内核参数，提升OS性能。	修复部分安全问题。
v1.21.1-r1	v1.21.7	-	容器网络支持内核宽匹配。	修复部分安全问题。
v1.21.1-r0	v1.21.7	首次发布CCE v1.21集群，有关更多信息请参见Kubernetes 1.21版本说明。	-	-

v1.19 版本

表 2-12 v1.19 补丁版本发布说明

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.19.16-r84	v1.19.16	-	-	修复CVE-2024-21626安全漏洞。
v1.19.16-r82	v1.19.16	-	修复Ingress配置SNI证书并同时开启HTTP/2的场景下偶现配置冲突的问题。	-

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.19.16-r80	v1.19.16	-	• 修复BMS节点重启后显示节点不可用的问题。 • 优化VIP路由清理逻辑，先清理残留VIP路由，避免出现NetworkManager重启后重新添加路由的问题。 • 修复CCE Turbo集群使用独享型ELB场景，Pod滚动升级复用IP可能导致ELB后端服务器无法添加的问题。 • 修复创建负载均衡类型Service时加入healthcheck队列后，如果Service被删除，会导致缓存残留的问题。 • 修复Master节点所在的物理机或者交换机网络设备掉线之后，内存持续上涨，docker占用内存持续上涨的问题。	修复部分安全问题。
v1.19.16-r60	v1.19.16	• Volcano支持节点池亲和调度。 • Volcano支持负载重调度能力。	-	修复部分安全问题。
v1.19.16-r50	v1.19.16	-	优化节点池伸缩时的事件信息。	修复部分安全问题。
v1.19.16-r40	v1.19.16	• 节点池配置管理支持软驱逐和硬驱逐的设置。 • 支持为自动创建的EVS块存储添加TMS资源标签，以便于成本管理。	-	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.19.16-r30	v1.19.16	• CCE集群支持对接使用弹性规格的独享型ELB。 • 负载均衡支持设置超时时间。	kube-apiserver高频参数支持配置。	修复部分安全问题。
v1.19.16-r20	v1.19.16	• CCE Turbo容器网卡支持固定IP。详情请参见**为Pod配置固定IP**。 • CCE Turbo容器网卡支持自动创建和自动绑定EIP。详情请参见**为Pod配置固定EIP**。	• 云原生2.0网络支持命名空间指定子网。 • 增强节点重启后，docker运行时拉取镜像的稳定性。 • 优化CCE Turbo集群在非全预热场景分配网卡的性能。	修复部分安全问题。
v1.19.16-r10	v1.19.16	-	提升Service/Ingress对接ELB特性稳定性。	修复部分安全问题。
v1.19.16-r7	v1.19.16	-	• 增强Docker版本升级时的可靠性。 • 优化集群节点时间同步能力。 • 增强CCE Turbo集群预热场景可靠性。	修复部分安全问题。
v1.19.16-r6	v1.19.16	• Service和Ingress支持关联G-EIP的独享型ELB。	• 增强配置了QoS的containerd容器运行的稳定性。 • Ingress支持对url重写策略配置和修改。 • 优化kube-controller-manager在频繁更新CRD场景下的内存使用。	修复部分安全问题。
v1.19.16-r5	v1.19.16	• 支持LB类型的Service同时配置TCP/UDP端口 • 支持Pod readiness gate	• 增强底层网络异常时的流表可靠性。 • 增强高版本内核的OS异常掉电等重启场景的稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.19.16-r4	v1.19.16	• 容器存储支持对接SFS 3.0文件存储服务。 • 支持GPU节点的设备故障检测和隔离能力。 • 支持配置集群维度的自定义安全组。 • CCE Turbo集群支持节点级别的网卡预热参数配置。	• 优化节点污点场景下负载调度的性能。 • 增强Containerd运行时绑核场景下长时间运行的稳定性。 • 优化ELB Service/Ingress在大量连接场景下的稳定性。 • 优化kube-apiserver在频繁更新crd场景下的内存使用。	修复部分安全问题及以下CVE漏洞： • CVE-2022-3294 • CVE-2022-3162 • CVE-2022-3172
v1.19.16-r3	v1.19.16	-	• 支持image-pull-progress-deadline启动参数升级保留。 • CCE Turbo支持节点自定义网卡动态预热配置。 • 解决隧道网络在master节点间网络闪断场景下导致的网络分配不一致问题。 • 优化容器隧道网络模式集群在控制面节点网络闪断场景下的稳定性。	修复部分安全问题。
v1.19.16-r2	v1.19.16	-	• 增强集群控制面节点掉电时链接释放的稳定性。 • 增强节点上Pod并发挂载存储卷的稳定性。	修复部分安全问题。
v1.19.16-r1	v1.19.16	-	增强EulerOS 2.9操作系统NetworkManager的稳定性。	修复部分安全问题。

CCE集群补丁版本号	Kubernetes社区版本	特性更新	优化增强	安全漏洞修复
v1.19.16-r0	v1.19.16	-	增强工作负载升级且节点伸缩状态下，负载均衡服务更新的稳定性。	修复部分安全问题及以下CVE漏洞： • CVE-2021-25741 • CVE-2021-25737
v1.19.10-r0	v1.19.10	首次发布CCE v1.19集群，有关更多信息请参见Kubernetes 1.19版本说明。	-	-

2.2 购买集群

2.2.1 集群类型对比

集群类型对比

CCE支持多种类型的集群创建，以满足您各种业务需求，如下为集群类型之间的区别，可帮助您选择合适的集群：

维度	子维度	CCE Standard	CCE Turbo	CCE Autopilot
产品定位	-	标准版本集群，提供高可靠、安全的商业级容器集群服务。	面向云原生2.0的新一代容器集群产品，计算、网络、调度全面加速。	无用户节点的Serverless版集群，无需对节点的部署、管理和安全性进行维护，并根据CPU和内存资源用量按需付费。

维度	子维度	CCE Standard	CCE Turbo	CCE Autopilot
使用场景	-	面向有云原生数字化转型诉求的用户，期望通过容器集群管理应用，获得灵活弹性的算力资源，简化对计算、网络、存储的资源管理复杂度。	适合对极致性能、资源利用率提升和全场景覆盖有更高诉求的客户。	适合具有明显的波峰波谷特征的业务负载，例如在线教育、电子商务等行业。
规格差异	网络模型	云原生网络1.0：面向性能和规模要求不高的场景。 ● 容器隧道网络模式 ● VPC网络模式	云原生网络2.0：面向大规模和高性能的场景。 组网规模最大支持2000节点	云原生网络2.0：面向大规模和高性能的场景。
	网络性能	VPC网络叠加容器网络，性能有一定损耗	VPC网络和容器网络融合，性能无损耗	VPC网络和容器网络融合，性能无损耗
	容器网络隔离	● 容器隧道网络模式：集群内部网络隔离策略，支持NetworkPolicy。 ● VPC网络模式：不支持	Pod可直接关联安全组，基于安全组的隔离策略，支持集群内外部统一的安全隔离。	Pod可直接关联安全组，基于安全组的隔离策略，支持集群内外部统一的安全隔离。
	安全隔离性	普通容器：Cgroups隔离	● 安全容器：当前仅物理机支持，提供虚机级别的隔离 ● 普通容器：Cgroups隔离	提供虚机级别的隔离
	边缘基础设施管理	不支持	支持管理智能边缘小站	不支持

2.2.2 购买 Standard/Turbo 集群

您可以通过云容器引擎控制台非常方便快速地创建Kubernetes集群。创建完成后，集群控制节点将由云容器引擎服务托管，您只需创建工作节点，帮助您降低集群运维成本，可实现简单高效的业务部署。

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。
- 变更控制节点可用区。
- 变更集群的网络配置，如所在的虚拟私有云VPC、子网、服务网段、IPv6、kube-proxy代理模式（即**服务转发模式**）。
- 变更网络模型，例如“容器隧道网络”更换为“VPC网络”。

步骤一：登录 CCE 控制台

步骤1 登录**CCE控制台**。

步骤2 在“集群管理”页面右上角单击“购买集群”。

----结束

步骤二：配置集群

在“购买集群”页面，填写集群配置参数。

基础配置

参数	说明
集群类型	根据需求选择“CCE Standard集群”或“CCE Turbo集群”。 • CCE Standard集群：标准版本集群，提供高可靠、安全的商业级容器集群服务。 • CCE Turbo集群：拥有更高性能的云原生网络，提供云原生混部调度能力，可实现更高的资源利用率和更广的全场景覆盖。 关于集群类型差异详情，请参见集群类型对比。
计费模式	根据需求选择集群的计费模式。 • 包年/包月：预付费模式，按订单的购买周期计费，适用于可预估资源使用周期的场景，价格比按需计费模式更优惠。 选择该计费模式时，需要设置“购买时长”并选择是否自动续费（按月购买时自动续费周期为1个月，按年购买时自动续费周期则为1年）。 • 按需计费：后付费模式，按资源的实际使用时长计费，可以随时开通/删除资源。
集群名称	请输入集群名称，同一账号下集群不可重名。
企业项目	该参数仅对开通企业项目的企业客户账号显示。 选择某个企业项目后，集群和集群安全组将会创建在该企业项目下。您可以通过企业项目服务（EPS）管理集群及其他资源（节点、ELB、以及节点的安全组等）。了解更多企业项目相关信息，请查看**企业管理**。
集群版本	选择集群使用的Kubernetes版本。
集群规模	集群支持管理的最大节点数量，请根据业务场景选择。创建完成后支持扩容，不支持缩容，详情请参见**变更集群规格**。

参数	说明
集群master实例数	选择集群控制平面的节点（master实例）数量。控制平面节点由系统自动托管，会部署Kubernetes集群的管控面组件，如 kube-apiserver，kube-controller-manager，kube-scheduler 等组件。 • 3实例（高可用）：创建3个控制平面节点，确保集群高可用。 • 单实例：您的集群只会创建一个控制平面节点。 **说明** • 在CCE集群的Master（控制节点）发生故障的数量超过一半情况下，集群将无法正常的活动。 您还可以指定集群 master 实例分布策略，默认为自动分配。 • 自动分配：即随机分配，尽可能将控制节点随机分布在不同可用区以提高容灾能力 。若某可用区资源不足，将分配至资源充足的可用区，优先保障集群创建成功，可能无法保障可用区级容灾。 • 自定义分配：自定义选择每台控制节点的位置。 单实例场景下，您可以选择一个可用区进行部署；多实例场景下，您可以选择多种分配场景： – 可用区：通过把控制节点创建在不同的可用区中实现容灾。 – 主机：通过把控制节点创建在相同可用区下的不同主机中实现容灾。 – 自定义：用户自行决定每台控制节点所在的位置。

网络配置

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

表 2-13 集群网络配置

参数	说明
虚拟私有云	选择集群所在的虚拟私有云VPC，如没有可选项可以单击右侧“新建虚拟私有云”创建。创建后不可修改。
节点子网	选择节点所在子网，如没有可选项可以单击右侧“新建子网”创建。创建后子网不可修改。
节点默认安全组	您可选择使用CCE自动生成的安全组，或选择已有安全组作为节点默认安全组。 **须知** 节点默认安全组必须放通指定端口来保证集群内部正常通信，否则将无法成功创建节点，安全组端口配置说明请参考**集群安全组规则配置**。

参数	说明
启用IPv6	开启后将支持通过IPv6地址段访问集群资源，包括节点，工作负载等。 ● v1.15及以上版本容器隧道网络的CCE Standard集群支持开启IPv4/IPv6双栈，并在v1.23版本中进入GA（Generally Available）阶段。 ● v1.23.8-r0、v1.25.3-r0及以上版本的CCE Turbo集群支持开启IPv4/IPv6双栈。 ● VPC网络模型的集群暂不支持开启IPv4/IPv6双栈。 具体请参见**如何通过CCE搭建IPv4/IPv6双栈集群?** 。

表 2-14 容器网络配置

参数	说明
容器网络模型	CCE Standard集群支持选择“VPC网络”和“容器隧道网络”。 CCE Turbo集群支持选择“云原生网络2.0”。 如需了解更多网络模型差异，请参见**容器网络模型对比**。
容器网段	CCE Standard集群支持该参数，设置容器使用的网段，决定了集群下容器的数量上限。VPC网络模型支持添加多个容器网段，且支持集群创建后添加容器网段，请参见**扩展集群容器网段**。
容器子网（Pod CIDR）	CCE Turbo集群支持该参数，选择容器所在子网，如没有可选项可以单击右侧“新建子网”创建。容器子网决定了集群下容器的数量上限，创建后支持新增子网。
容器默认安全组	CCE Turbo集群支持该参数，您可选择使用CCE自动生成的安全组，或选择已有安全组作为容器默认安全组。 **须知** 容器默认安全组必须放通指定端口来保证集群内部容器之间正常通信，安全组端口配置说明请参考**集群安全组规则配置**。

表 2-15 服务网络配置

参数	说明
服务网段（Service CIDR）	同一集群下容器互相访问时使用的Service资源的网段。决定了Service资源的上限。 创建后不可修改。

参数	说明
服务转发模式	支持IPVS和iptables两种转发模式，具体请参见**iptables与IPVS如何选择**。 • iptables：社区传统的kube-proxy模式。适用于Service数量较少或客户端会出现大量并发短链接的场景。IPv6集群不支持iptables模式。 • IPVS：吞吐更高，速度更快的转发模式。适用于集群规模较大或Service数量较多的场景。
IPv6服务网段	仅CCE Turbo集群开启IPv6双栈时需要设置，创建后不可修改。

高级配置（可选）

参数	说明
IAM认证	CCE集群支持通过IAM服务认证鉴权，您可以通过IAM认证调用接口连接到集群。
证书认证	• 系统生成：默认开启X509认证模式，X509是一种非常通用的证书格式。 • 自有证书：您可以将自定义证书添加到集群中，用自定义证书进行认证。 您需要分别上传自己的**CA根证书**、**客户端证书**和**客户端证书私钥**。 **注意** • 请上传**小于1MB**的文件，CA根证书和客户端证书上传格式支持**.crt**或**.cer**格式，客户端证书私钥仅支持上传**未加密的证书私钥**。 • 客户端证书有效期需要5年以上。 • 上传的CA根证书既给认证代理使用，也用于配置kube-apiserver聚合层，**如不合法，集群将无法成功创建**。 • 从1.25版本集群开始，Kubernetes不再支持使用SHA1WithRSA、ECDSAWithSHA1算法生成的证书认证，推荐使用SHA256算法生成的证书进行认证。
开启CPU管理策略	支持为工作负载实例设置独占CPU核的功能，详情请参见**CPU管理策略**。
开启过载控制	过载控制开启后，将根据控制节点的资源压力，动态调整请求并发量，维护控制节点和集群的可靠性。详情请参见**开启集群过载控制**。
开启对分布式云支持(homezone/cloudpond)	集群可以统一管理数据中心和边缘的计算资源，用户可以根据业务诉求将容器部署到合适的区域。 该功能仅CCE Turbo集群支持，且需要提前注册**智能边缘小站**，详情请参见**在CCE Turbo集群中使用分布式云资源**。
禁止集群删除	集群删除保护，防止通过控制台或API误删除集群，开启后将禁止用户从CCE侧删除或退订集群。集群创建后可前往集群配置中心修改。

参数	说明
集群时区	集群内的定时任务和节点时区将统一采用所选时区。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。最多可以添加20个资源标签。 **说明** 如您的账号归属某个**组织**，且该组织已经设定云容器引擎服务的相关**标签策略**，则需按照标签策略规则为集群添加标签。标签如果不符合标签策略的规则，则可能会导致集群创建失败，请联系组织管理员了解标签策略详情。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 ● 标签键只能包含中文、英文字母、数字、空格和特殊字符(-_.:=+@)，且首尾不能包含空格，不能以_sys_开头，长度不超过128个字符。资源标签键不可以为空。 ● 标签值只能包含中文，英文字母、数字、空格和特殊字符(-_.:/=+@)，长度不超过255个字符。资源标签值可以为空。
集群描述	支持200个字符。

步骤三：插件选择

单击“下一步：插件选择”，选择创建集群时需要安装的插件。

基础功能

参数	说明
CCE容器网络插件 (Yangtse CNI)	集群默认安装的基础插件，为集群内的Pod提供网络连通、公网访问、安全隔离等网络能力。
CCE 容器存储 (Everest)	默认安装**CCE容器存储（Everest）**，可为集群提供基于 CSI 的容器存储能力，支持对接云上云硬盘等存储服务。
CoreDNS 域名解析	默认安装**CoreDNS域名解析**插件，可为集群提供域名解析、连接云上 DNS 服务器等能力。
节点本地域名解析加速	可选插件。勾选后自动安装**节点本地域名解析加速**插件，通过在集群节点上运行 DNS 缓存代理来提高集群 DNS 性能。
Volcano调度器	可选插件。勾选后自动安装**Volcano调度器**插件，并将集群的默认调度器设置为Volcano，为您提供面向批量计算、高性能计算场景的高级调度能力。
CCE突发弹性引擎（对接CCI）	可选插件。勾选后自动安装**CCE突发弹性引擎（对接CCI）**插件，支持在短时高负载场景下，将部署在云容器引擎CCE上的容器实例（Pod），弹性创建到CCI。

可观测性

参数	说明
云原生监控插件	可选插件。勾选后自动安装**云原生监控插件**，为集群提供普罗指标采集能力，并将指标上报至指定的AOM实例。轻量化模式暂不支持基于自定义普罗语句的HPA，若需要相关功能，可在集群创建完成后手动安装全量的插件。 AOM采集的**基础指标**免费，自定义指标将由AOM服务进行收费，详情请参见**价格详情**。关于如何采集自定义指标，请参见**使用云原生监控插件监控自定义指标**。
云原生日志采集插件	可选插件。勾选后自动安装**云原生日志采集插件**，将日志上报至 LTS 的日志采集器。集群创建完成后可在 CCE 日志中心页面对采集规则进行查询与管理。 LTS创建日志组免费，并每月赠送每个账号一定量免费日志采集额度，超过免费额度部分将产生费用（**价格计算器**）。关于如何采集自定义指标，请参见**通过云原生日志采集插件采集容器日志**。
CCE 节点故障检测	可选插件。勾选后自动安装**CCE节点故障检测**插件，安装后可为集群提供节点故障检测、隔离能力，帮助您及时识别节点问题。

步骤四：插件配置

单击“下一步：插件配置”，配置插件。

基础功能

参数	说明
CCE容器网络插件 (Yangtse CNI)	不支持配置。
CCE 容器存储 (Everest)	不支持配置。集群创建完成后，可前往“插件中心”修改配置。
CoreDNS 域名解析	不支持配置。集群创建完成后，可前往“插件中心”修改配置。
节点本地域名解析加速	不支持配置。集群创建完成后，可前往“插件中心”修改配置。
Volcano调度器	不支持配置。集群创建完成后，可前往“插件中心”修改配置。
CCE突发弹性引擎（对接CCI）	选择CCI实例所在的子网，弹性到CCI创建的Pod实例会占用所选子网下的IP，请合理规划网段，避免IP资源不足。 工作负载实例（Pod）调度到CCI服务后，将按照CCI的收费标准进行计费，详情请参见**计费项**。

可观测性

参数	说明
云原生监控插件	选择指标上报的AOM实例。如果没有可用实例，您可以单击“新建实例”进行创建。 AOM采集的**基础指标**免费，自定义指标将由AOM服务进行收费，详情请参见**价格详情**。关于如何采集自定义指标，请参见**使用云原生监控插件监控自定义指标**。
云原生日志采集插件	选择需要采集的日志。开启后将自动创建一个名称为k8s-log-{clusterId}的日志组，并为每个勾选的日志类型创建一个日志流。 • 容器日志：采集容器标准输出日志，对应的日志流名称为stdout-{clusterId}。 • Kubernetes事件：采集Kubernetes日志，对应的日志流名称为event-{clusterId}。 • Kubernetes审计日志：采集Master控制平面审计日志，对应的日志流名称为audit-{clusterId}。 • 控制面组件日志：采集Master控制平面（包括 kube-apiserver、 kube-controller-manage 和 kube-scheduler）日志，对应的日志流名称分别为kube-apiserver-{clusterId}、kube-controller-manage-{clusterId}、kube-scheduler-{clusterId}。 如果不开启日志采集能力。集群创建后可以前往 CCE 日志中心页面重新开启。 LTS创建日志组免费，并每月赠送每个账号一定量免费日志采集额度，超过免费额度部分将产生费用（**价格计算器**）。关于如何采集自定义指标，请参见**通过云原生日志采集插件采集容器日志**。
CCE 节点故障检测	不支持配置。集群创建完成后，可前往“插件中心”修改配置。

步骤五：确认配置

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

集群创建预计需要5-10分钟，您可以单击“返回集群管理”进行其他操作或单击“查看集群事件列表”后查看集群详情。

相关操作

- 通过命令行工具连接集群：请参见**通过kubectl连接集群**。
- 添加节点：集群创建完成后，若您需要为集群添加节点，请参见**创建节点**。
- 创建IPv4/IPv6双栈集群：请参见**如何通过CCE搭建IPv4/IPv6双栈集群?** 。
- 通过kubectl对接多个集群：请参见**通过kubectl对接多个集群**

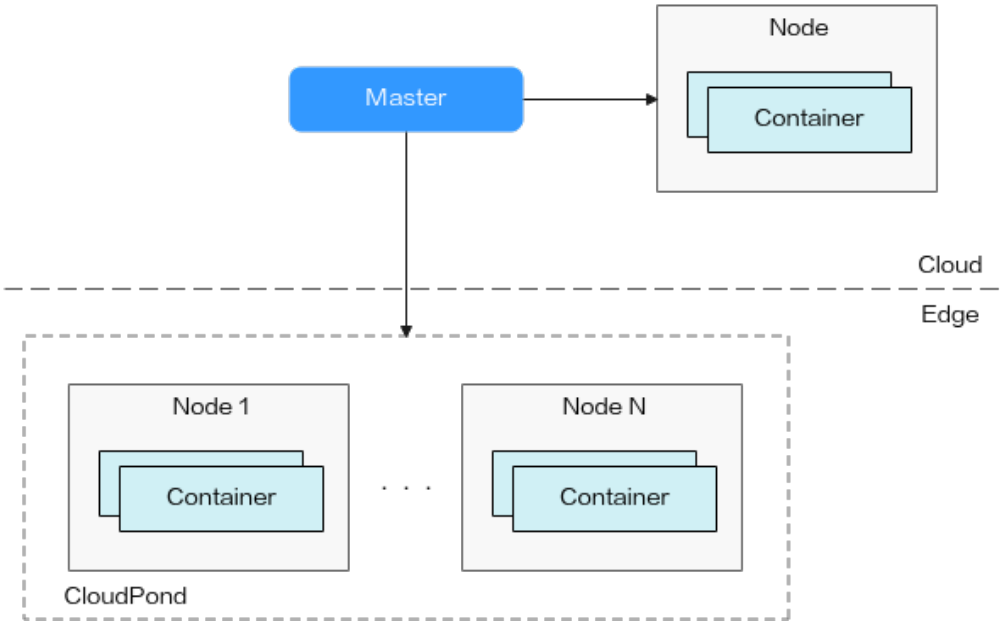
2.2.3 在 CCE Turbo 集群中使用分布式云资源

CCE Turbo集群支持**管理边缘基础设施**（智能边缘小站）的能力。启用分布式支持后，一个集群可以统一管理数据中心和边缘的计算资源，用户可以便捷地根据应用的诉求将其部署在对应的区域。

说明

CCE Turbo集群使用分布式云资源功能需要提前注册并部署智能边缘小站服务。

图 2-2 CCE Turbo 分布式管理



核心概念

为了区分云上资源和分布在不同边缘区域的资源，也为了方便用户使用和管理处于不同区域的资源，引入了**分区**概念，定义如下：

- 计算角度：一个分区是一组“风火水电”相互隔离、但网络位置靠近（通常互访时延小于2ms）的数据中心可用区（available zone）的集合。将应用分散在一个分区内不同可用区可以实现应用高可用性。
- 网络角度：一个分区内节点和容器需要使用创建在该分区内可用区下的虚拟私有云（VPC）子网。为方便配置和管理，分区创建时需设定默认子网，当创建节点不指定子网时，会填入分区默认子网。
- 其他属性：分区包含资源类别，即中心云、智能边缘小站。该属性的配置方便后续负载的调度。

约束与限制

- 节点：分布式集群当前仅支持普通x86虚拟机。暂不支持节点迁移功能。
- 节点池：节点池随机调度功能仅限于分区内。
- 存储：当前仅支持在边缘区域创建云硬盘（EVS），其他种类存储方式不推荐使用。

- 服务与路由：仅支持独享型ELB。
- 插件：分布式集群支持如下插件，且优先将插件部署在云上节点。
  - CCE容器存储（Everest）
  - CoreDNS域名解析
  - 节点本地域名解析加速
  - CCE集群弹性引擎
  - Kubernetes Metrics Server
  - CCE节点故障检测
  - Kubernetes Dashboard
  - CCE AI套件（Ascend NPU）
  - CCE AI套件（NVIDIA GPU）

开启对分布式云支持

创建CCE Turbo集群时，可在创建集群过程中，开启对分布式云(cloudpond)支持。

说明

开启分布式云支持后，在集群中创建的边缘节点，默认会添加以下污点和K8s标签：

- 污点：distribution.io/category=IES:NoSchedule
- K8s标签：
  - distribution.io/category=IES
  - distribution.io/partition=<AZ name>
  - distribution.io/publicbordergroup=<AZ name>

图 2-3 开启对分布式云(cloudpond)支持

2.2.4 iptables 与 IPVS 如何选择

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

CCE当前支持iptables和IPVS两种服务转发模式，各有优缺点。

特性差异	iptables	IPVS
定位	成熟稳定的kube-proxy代理模式，但性能一般，适用于Service数量较少（小于3000）或客户端会出现大量并发短连接的场景。更多说明请参见**iptables简介**。	高性能的kube-proxy代理模式，适用于集群规模较大或Service数量较多的场景。更多说明请参见**IPVS简介**。
吞吐量	较小	较大
复杂度	O(n)，其中n随集群中服务和后端Pod的数量同步增长	O(1)，多数情况下IPVS的连接处理效率和集群规模无关
负载均衡算法	随机平等的选择算法	包含多种不同的负载均衡算法，例如轮询、最短期望延迟、最少连接以及各种哈希方法等
ClusterIP连通性	集群内部ClusterIP地址无法ping通	集群内部ClusterIP地址可以正常ping通 **说明** 由于**社区安全加固**，v1.27及以上版本的集群中ClusterIP地址无法ping通。
额外限制	当集群中超过3000个Service时，可能会出现网络延迟的情况。	• Ingress和Service使用相同ELB实例时，无法在集群内的节点和容器中访问Ingress，因为kube-proxy会在ipvs-0的网桥上挂载LB类型的Service地址，Ingress对接的ELB的流量会被ipvs-0网桥劫持。建议Ingress和Service使用不同ELB实例。 • 不推荐使用EulerOS 2.5、CentOS 7.6、Ubuntu 18.04的操作系统，详情请参见**CCE集群IPVS转发模式下conn_reuse_mode问题说明**。

iptables 简介

iptables是一个Linux内核功能，提供了大量的数据包处理和过滤方面的能力。它可以在核心数据包处理管线上用Hook挂接一系列的规则。iptables模式中kube-proxy 在NAT pre-routing Hook中实现NAT和负载均衡功能。对于每个Service，kube-proxy都会添加一个iptables规则，这些规则捕获流向Service的ClusterIP和Port的流量，并将这些流量重定向到Service后端的其中之一。默认情况下，iptables模式下的kube-proxy会随机选择一个Service后端。详情请参见**iptables代理模式**。

IPVS 简介

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

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

2.3 连接集群

2.3.1 通过 kubectl 连接集群

操作场景

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

权限说明

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

用户拥有的权限请参见**集群权限（IAM授权）与命名空间权限（Kubernetes RBAC授权）的关系**。

使用 kubectl 连接集群

若您需要从客户端计算机连接到Kubernetes集群，可使用Kubernetes命令行客户端kubectl，您可登录CCE控制台，单击待连接集群名称，在集群总览页面查看访问地址以及kubectl的连接步骤，如**图2-4**所示。

CCE支持“内网访问”和“公网访问”两种方式访问集群。

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

须知

通过“公网访问”方式访问集群，您需要在总览页中的“连接信息”版块为集群绑定公网地址，如**图2-4**所示。绑定公网集群的kube-apiserver将会暴露到互联网，存在被攻击的风险，建议对kube-apiserver所在节点的EIP配置DDoS高防服务。

图 2-4 集群连接信息

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

步骤1 下载kubectl

您需要准备一台可访问公网的客户端计算机，并通过命令行方式安装kubectl。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。

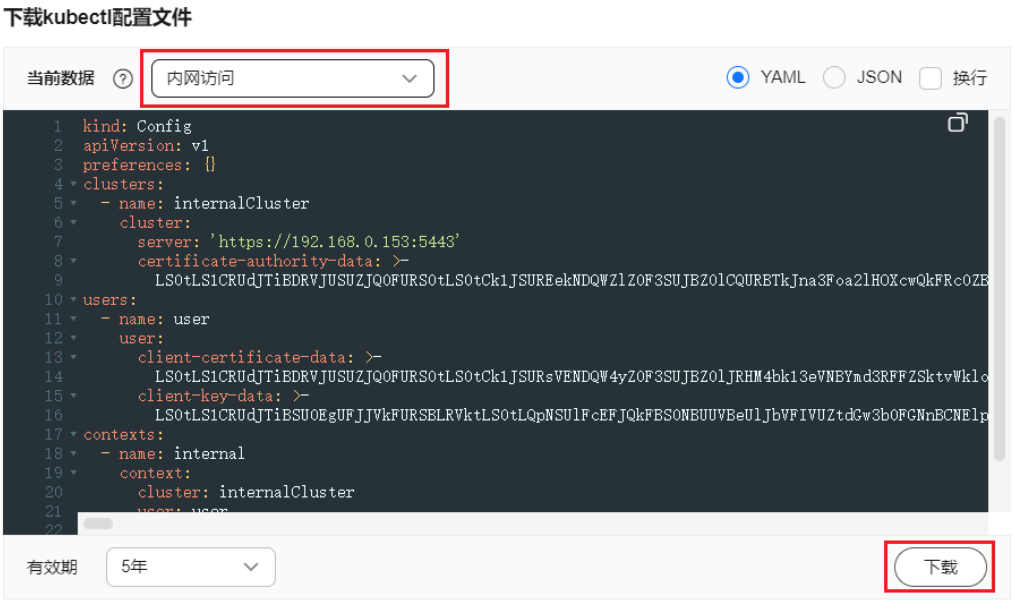
本文以Linux环境为例安装和配置kubectl，详情请参考**安装kubectl**。

1. 登录到您的客户端机器，下载kubectl。
   ```
   cd /home
   curl -LO https://dl.k8s.io/release/{v1.25.0}/bin/linux/amd64/kubectl
   ```
   其中*{v1.25.0}*为指定的版本号，请根据集群版本进行替换。
2. 安装kubectl。
   ```
   chmod +x kubectl
   mv -f kubectl /usr/local/bin
   ```

步骤2 获取kubectl配置文件

在集群总览页中的“连接信息”版块，单击kubectl后的“配置”按钮，查看kubectl的连接信息，并在弹出页面中选择“内网访问”或“公网访问”，然后下载对应的配置文件。

图 2-5 下载配置文件



说明

- kubectl配置文件（kubeconfig）用于对接认证集群，请您妥善保存该认证凭据，防止文件泄露后，集群有被攻击的风险。
- IAM用户下载的配置文件所拥有的Kubernetes权限与CCE控制台上IAM用户所拥有的权限一致。
- 如果Linux系统里面配置了KUBECONFIG环境变量，kubectl会优先加载KUBECONFIG环境变量，而不是$home/.kube/config，使用时请注意。

步骤3 配置kubectl

以Linux环境为例配置kubectl。

1. 登录到您的客户端机器，复制**步骤2**中下载的配置文件（以kubeconfig.yaml为例）到您客户端机器的/home目录下。
2. 配置kubectl认证文件。

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

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

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

   - 互联网接入访问请执行（集群需绑定公网地址）：

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

   - 互联网接入访问如需开启双向认证请执行（集群需绑定公网地址）：

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

     关于集群双向认证的说明请参见**域名双向认证**。

----结束

域名双向认证

CCE当前支持域名双向认证。

- 集群API Server绑定EIP后，使用kubectl连接集群时域名双向认证默认不开启，可通过 **kubectl config use-context externalTLSVerify** 命令切换到externalTLSVerify这个context开启使用。
- 集群绑定或解绑弹性IP、配置或更新自定义域名时，集群服务端证书将同步签入最新的集群访问地址（包括集群绑定的弹性IP、集群配置的所有自定义域名）。
- 异步同步集群通常耗时约5-10min，同步结果可以在操作记录中查看“同步证书”。
- 对于已绑定EIP的集群，如果在使用双向认证时出现认证不通过的情况（x509: certificate is valid），需要重新绑定EIP并重新下载kubeconfig.yaml。
- 早期未支持域名双向认证时，kubeconfig.yaml中包含"insecure-skip-tls-verify": true字段，如图2-6所示。如果需要使用双向认证，您可以重新下载kubeconfig.yaml文件并配置开启域名双向认证。

图 2-6 未开启域名双向认证

```
"clusters": [{
    "name": "mycluster",
    "cluster": {
        "server": "https://10.100.0.52:5443",
        "insecure-skip-tls-verify": true
    }
```

常见问题

- **Error from server Forbidden**

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

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

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

- **The connection to the server localhost:8080 was refused**

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

  ```
  The connection to the server localhost:8080 was refused - did you specify the right host or port?
  ```

  原因是由于该kubectl客户端未配置集群认证，请参见**步骤3**进行配置。

相关操作

- **通过kubectl对接多个集群**
- **通过配置kubeconfig文件实现集群权限精细化管理**

2.3.2 通过 CloudShell 连接集群

操作场景

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

权限说明

在CloudShell中使用kubectl时，kubectl的权限由登录用户的权限决定。

约束与限制

同一用户在使用CloudShell组件连接CCE集群或容器时，限制同时打开的实例上限数量为15个。

使用 CloudShell 连接集群

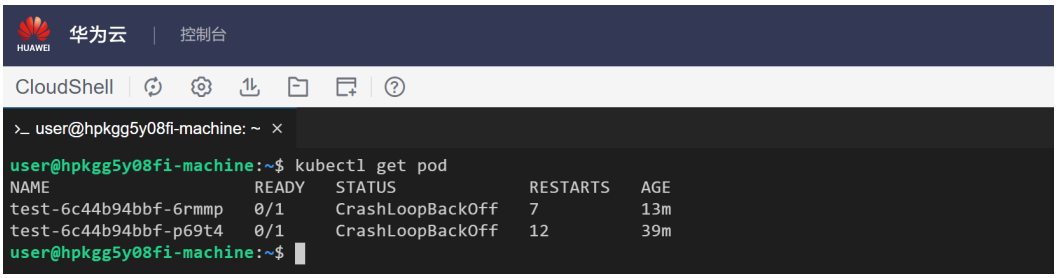
CloudShell是一款用于管理与运维云资源的网页版Shell工具，CCE支持使用CloudShell连接集群，如图2-7所示，单击“命令行工具”即可在CloudShell中使用kubectl访问集群。

说明

- CloudShell中kubectl证书有效期为1天，从云容器引擎重新跳转可以重置有效期。
- CloudShell基于VPCEP实现，在CloudShell中使用kubectl访问集群需要在集群控制节点的安全组（安全组名称：集群名称-cce-control-随机数）中放通如下网段访问5443端口。5443端口默认对所有网段放通，如果您对安全组做过加固，当出现在CloudShell中无法访问集群时，请检查5443端口是否放通了**198.19.0.0/16**网段。
- 集群必须安装CoreDNS才能使用CloudShell。
- 当前仅北京一、北京四、上海一、上海二、广州、贵阳一和乌兰察布一支持使用CloudShell登录容器。
- CloudShell暂不支持委托账号和子项目。

图 2-7 CloudShell

图 2-8 在 CloudShell 中使用 kubectl



2.3.3 通过 X509 证书连接集群

操作场景

通过控制台获取集群证书，使用该证书可以访问Kubernetes集群。

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 查看集群总览页，在右边“连接信息”下证书认证一栏，单击“下载”。

图 2-9 获取证书

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

须知

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

步骤4 使用集群证书调用Kubernetes原生API。

例如使用curl命令调用接口查看Pod信息，如下所示，其中*192.168.0.18:5443*为集群API Server的内网或公网地址。

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

更多集群接口请参见**Kubernetes API**。

----结束

2.3.4 通过自定义域名访问集群

操作场景

主题备用名称（Subject Alternative Name，缩写SAN）允许将多种值（包括IP地址、域名等）与证书关联。SAN通常在TLS握手阶段被用于客户端校验服务端的合法性：服务端证书是否被客户端信任的CA所签发，且证书中的SAN是否与客户端实际访问的IP地址或DNS域名匹配。

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

如果您有特殊的代理访问或跨域访问需求，可以通过自定义SAN实现。域名访问场景的典型使用方式如下：

- 客户端配置Host域名指定DNS域名地址，或者客户端主机配置/etc/hosts，添加域名映射。
- 云上内网使用，云解析服务DNS支持配置集群弹性IP与自定义域名的映射关系。后续更新弹性IP可以继续使用双向认证+自定义域名访问集群，无需重新下载kubeconfig.json配置文件。
- 自建DNS服务器，自行添加A记录。

前提条件

已创建一个v1.19及以上版本的集群。

添加自定义 SAN

步骤1 登录CCE控制台。

步骤2 在集群列表中单击集群名称，进入集群总览页。

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

图 2-10 自定义 SAN

说明

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

----结束

使用 SAN 连接集群

通过kubectl连接集群

步骤1 修改SAN后，需重新下载kubeconfig.json配置文件。

1. 登录CCE控制台，单击集群名称进入集群。
2. 在集群总览页中的“连接信息”版块，单击kubectl后的“配置”按钮，查看kubectl的连接信息，并在弹出页面中下载配置文件。

步骤2 配置kubectl。

1. 登录到您的客户端机器，复制**步骤1.2**中下载的配置文件（kubeconfig.json）到您客户端机器的/home目录下。
2. 配置kubectl认证文件。
   ```
   cd /home
   mkdir -p $HOME/.kube
   mv -f kubeconfig.json $HOME/.kube/config
   ```
3. 切换kubectl的访问模式，使用SAN连接集群。
   ```
   kubectl config use-context customSAN-0
   ```
   其中*customSAN-0*为自定义SAN对应的配置名称。如同时设置了多个SAN，每个SAN对应配置名称中的数字从0开始依次增大，例如*customSAN-0*、*customSAN-1*，以此类推。

----结束

通过X509证书连接集群

步骤1 修改SAN后，需重新下载X509证书。

1. 登录CCE控制台，单击集群名称进入集群。
2. 查看集群总览页，在右边“连接信息”下证书认证一栏，单击“下载”。
3. 在弹出的“证书获取”窗口中，根据系统提示选择证书的过期时间并下载集群X509证书。

步骤2 使用集群证书调用Kubernetes原生API。

例如使用curl命令调用接口查看Pod信息，如下所示，其中*example.com:5443*为自定义SAN。

```
curl --cacert ./ca.crt --cert ./client.crt --key ./client.key  https://example.com:5443/api/v1/namespaces/default/pods/
```

更多集群接口请参见**Kubernetes API**。

----结束

2.3.5 吊销集群访问凭证

在多租户场景下，CCE会为每个用户生成一个独立的集群访问凭证（kubeconfig或X509证书），该凭证包含了用户身份及授权信息，以便其可以连接到相应的集群并执行授权范围内的操作。这种方式可以确保不同用户之间的隔离和安全性，同时也方便了管理和授权。但该凭证的有效时间一般为固定值，当持有该凭证的员工离职或已授权的凭证疑似泄露等场景发生时，需要手动吊销集群访问凭证，以确保集群安全。

吊销集群访问凭证的存在两类场景：

- 子用户（非管理员）：支持吊销自身的集群访问凭证。

- 主账号或管理员用户（admin用户组用户）：支持吊销其他用户或委托账号的集群访问凭证。

须知

吊销集群访问凭证后，该凭证的持有人将无法访问集群，且无法恢复已吊销的凭证，请谨慎操作。

如需访问集群，需要为该用户重新生成集群访问凭证。

前提条件

您需要拥有一个v1.19.16-r50、v1.21.11-r10、v1.23.9-r10、v1.25.4-r10、v1.27.1-r10及以上版本的集群。

子用户（非管理员）操作

非管理员的子用户仅支持吊销自身的集群访问凭证，请参考如下操作。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“总览”，在右边“连接信息”版块中，单击“吊销”。

图 2-11 吊销集群访问凭证

步骤3 如果您确定要吊销自身的集群访问凭证，请在二次确认框中输入**REVOKE**后单击“确定”。

----结束

主账号或管理员用户（admin 用户组用户）操作

主账号或管理员用户（admin用户组用户）支持吊销其他用户或委托账号的集群访问凭证，请参考如下操作。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“总览”，在右边“连接信息”版块中，单击“吊销”。

图 2-12 吊销集群访问凭证

步骤3 选择证书申请人进行证书吊销。吊销操作无法恢复，请谨慎操作。

- 用户：选择一个IAM用户，吊销其集群凭证。
- 委托账号：选择一个委托账号，吊销其集群凭证。
- 指定用户ID/委托ID：若您的用户已经删除，或者您通过身份提供商方式登录，需要您手动填写用户 ID。若您的委托账号已经删除，需要您手动填写委托 ID。

  您可以通过以下方案获取 ID：

  - 方式一：如果您可以获取此证书申请人下载的证书，证书的通用名称 (CN - Common Name) 即所需 ID。
  - 方式二：如果您无法获取到此证书申请人下载的证书，您可以通过云审计服务查询删除用户 (deleteUser)、删除委托 (deleteAgency) 的事件，事件对应的资源 ID 分别是已删除用户、已删除委托账号的 ID。

  如果使用上述方式均无法获取到所需 ID，请提交工单联系运维人员处理。

步骤4 如果您确定吊销所选用户的集群访问凭证，请在二次确认框中输入**REVOKE**后单击“确定”。

----结束

2.4 管理集群

2.4.1 修改 CCE 集群配置

操作场景

CCE支持对集群配置参数进行管理，通过该功能您可以对核心组件进行深度配置。

操作步骤

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到目标集群，查看集群的更多操作，并选择“配置管理”。

图 2-13 配置管理

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

表 2-16 集群服务器配置（kube-apiserver）

名称	参数	详情	取值
容器迁移对节点不可用状态的容忍时间	default-not-ready-toleration-seconds	容器迁移对节点不可用状态的容忍时间，默认对所有的容器生效，用户也可以为指定Pod进行差异化容忍配置，此时将以Pod配置的容忍时长为准，详情请参见**设置容忍策略**。 如果容忍时间配置过小，在网络抖动等短时故障场景下，容器可能会频繁迁移而影响业务；如果容忍时间配置过大，在节点故障时，容器可能长时间无法迁移，导致业务受损。	默认：300s
容器迁移对节点无法访问状态的容忍时间	default-unreachable-toleration-seconds	容器迁移对节点无法访问状态的容忍时间，默认对所有的容器生效，用户也可以为指定Pod进行差异化容忍配置，此时将以Pod配置的容忍时长为准，详情请参见**设置容忍策略**。 如果容忍时间配置过小，在网络抖动等短时故障场景下，容器可能会频繁迁移而影响业务；如果容忍时间配置过大，在节点故障时，容器可能长时间无法迁移，导致业务受损。	默认：300s

名称	参数	详情	取值
修改类API请求最大并发数	max-mutating-requests-inflight	最大mutating并发请求数。 当服务器超过此值时，它会拒绝请求。 0表示无限制。该参数与集群规模相关，不建议修改。	从v1.21版本开始不再支持手动配置，根据集群规格自动配置如下： ● 50和200节点：200 ● 1000节点：500 ● 2000节点：1000
非修改类API请求最大并发数	max-requests-inflight	最大non-mutating并发请求数。 当服务器超过此值时，它会拒绝请求。 0表示无限制。该参数与集群规模相关，不建议修改。	从v1.21版本开始不再支持手动配置，根据集群规格自动配置如下： ● 50和200节点：400 ● 1000节点：1000 ● 2000节点：2000
Nodeport类型服务端口范围	service-node-port-range	NodePort端口范围，修改后需前往安全组页面同步修改节点安全组30000-32767的TCP/UDP端口范围，否则除默认端口外的其他端口将无法被外部访问。 端口号小于20106会和CCE组件的健康检查端口冲突，引发集群不可用；端口号高于32767会和net.ipv4.ip_local_port_range范围冲突，影响性能。	默认：30000-32767 取值范围： min>20105 max<32768
开启过载防护	support-overload	集群过载控制开关，开启后将根据控制节点的资源压力，动态调整请求并发量，维护控制节点和集群的可靠性。 该参数仅v1.23及以上版本集群支持。	● false：不启用过载控制 ● true：启用过载控制

名称	参数	详情	取值
节点限制插件	enable-admission-plugin-node-restriction	节点限制插件限制了节点的kubelet只能操作当前节点的对象，增强了在高安全要求或多租户场景下的隔离性。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。	默认：开启
Pod节点选择器插件	enable-admission-plugin-pod-node-selector	Pod节点选择器插件允许集群管理员通过命名空间注释设置默认节点选择器，帮助约束Pod可以运行的节点，并简化配置。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。	默认：开启
Pod容忍度限制插件	enable-admission-plugin-pod-toleration-restriction	Pod容忍度限制插件允许通过命名空间设置Pod的容忍度的默认值和限制，为集群管理者提供了对Pod调度的精细控制，以保护关键资源。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。	默认：关闭
API受众	api-audiences	为ServiceAccount令牌定义其受众。Kubernetes 用于服务账户令牌的身份验证组件，会验证API请求中使用的令牌是否指定了合法的受众。 配置建议：根据集群服务间通信的需求，精确配置受众列表。此举确保服务账户令牌仅在授权的服务间进行认证使用，提升安全性。 **说明** 不正确的配置可能导致服务间认证通信失败，或令牌的验证过程出现错误。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认值："https://kubernetes.default.svc.cluster.local" 支持配置多个值，用英文逗号隔开。

名称	参数	详情	取值
服务账户令牌发行者	service-account-issuer	指定发行服务账户令牌的实体标识符。这是在服务账户令牌的负载（Payload）中的 'iss' 字段所标识的值。 配置建议：请确保所配置的发行者（Issuer）URL在集群内部可被访问，并且可被集群内部的认证系统所信任。 **说明** 若设置了一个不可信或无法访问的发行者 URL，可能会导致基于服务账户的认证流程失败。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认值："https://kubernetes.default.svc.cluster.local" 支持配置多个值，用英文逗号隔开。

表 2-17 调度器配置

名称	参数	详情	取值
默认调度器	default-scheduler	• kube-scheduler调度器：提供社区原生调度器标准调度能力。 • volcano调度器：兼容kube-scheduler调度能力，并提供增量调度能力。详情请参见**Volcano调度**。	默认：kube-scheduler调度器
调度器访问kube-apiserver的QPS	kube-api-qps	与kube-apiserver通信的QPS	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200
调度器访问kube-apiserver的突发流量上限	kube-api-burst	与kube-apiserver通信的burst	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200

名称	参数	详情	取值
开启GPU共享	enable-gpu-share	是否开启GPU共享，该参数仅v1.23.7-r10、v1.25.3-r0及以上版本集群支持。 • 关闭GPU共享时，需保证集群中的Pod没有使用共享GPU能力（即Pod不存在cce.io/gpu-decision的annotation），并需保证关闭GPU虚拟化功能。 • 开启GPU共享时，需保证集群中已使用GPU资源的Pod均存在cce.io/gpu-decision的annotation。	默认：开启

表 2-18 集群控制器配置（kube-controller-manager）

名称	参数	详情	取值
Deployment	concurrent-deployment-syncs	Deployment的并发处理数	默认：5
Endpoint	concurrent-endpoint-syncs	Endpoint的并发处理数	默认：5
GC回收	concurrent-gc-syncs	Garbage Collector的并发数	默认：20
Job	concurrent-job-syncs	允许同时同步的作业对象的数量。	默认：5
CronJob	concurrent-cron-job-syncs	允许同时同步的定时任务对象的数量。	默认：5
Namespace	concurrent-namespace-syncs	Namespace的并发处理数	默认：10
ReplicaSet	concurrent-replicaset-syncs	ReplicaSet的并发处理数	默认：5
ResourceQuota	concurrent-resource-quota-syncs	Resource Quota的并发处理数	默认：5
Servicepace	concurrent-service-syncs	Service的并发处理数	默认：10

名称	参数	详情	取值
ServiceAccountToken	concurrent-serviceaccount-token-syncs	ServiceAccount Token的并发处理数	默认：5
TTLAfterFinished	concurrent-ttl-after-finished-syncs	ttl-after-finished的并发处理数	默认：5
RC	concurrent_rc_syncs（v1.19及以下版本集群中使用） concurrent-rc-syncs（v1.21至v1.25.3-r0版本集群中使用）	RC的并发处理数 **说明** v1.25.3-r0及以上版本中该参数弃用。	默认：5
Pod水平伸缩同步的周期	horizontal-pod-autoscaler-sync-period	水平Pod扩缩器对Pod进行弹性伸缩的周期。配置越小弹性伸缩器反应越及时，同时CPU负载也越高。 **说明** 请合理设置该参数，周期配置过长可能导致控制器处理响应慢；周期配置过短则会对集群管控面造成压力，产生过载风险。	默认：15s
Pod水平伸缩容忍度	horizontal-pod-autoscaler-tolerance	该配置影响控制器对伸缩策略相关指标反应的灵敏程度，当配置为0时，指标达到策略阈值时立即触发弹性。 配置建议：如业务资源占用随时间的“突刺”特征明显，建议保留一定的容忍度值，避免因业务短时资源占用飚高导致预期之外的弹性行为。	默认：0.1

名称	参数	详情	取值
HPA CPU初始化期间	horizontal-pod-autoscaler-cpu-initialization-period	这一时段定义了纳入HPA计算的CPU使用数据仅来源于已经达到就绪状态并完成了最近一次指标采集的Pods。它的目的是在Pod启动初期过滤掉不稳定的CPU使用数据，进而防止基于瞬时峰值做出错误的扩缩容决策。 配置建议：如果您观察到Pods在启动阶段的CPU使用率波动导致HPA做出错误的扩展决策，增大此值可以提供一个CPU使用率稳定化的缓冲期。 **说明** 请合理设置该参数，设置值过低可能导致基于CPU峰值做出过度反应的扩容；而设置得过高则可能在实际需要扩容时造成延迟反应。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认：5分钟
HPA 初始就绪状态延迟	horizontal-pod-autoscaler-initial-readiness-delay	在CPU初始化期之后，此时间段允许HPA以一个较宽松的标准筛选CPU度量数据。也就是说，这段时间内，即使Pods的就绪状态有所变化，HPA也会考虑它们的CPU使用数据进行扩缩容。这有助于在Pod状态频繁变化时，确保CPU使用数据被持续追踪。 配置建议：如果Pods在启动后的就绪状态发生波动，并且您需要避免此波动导致HPA的误判，适当增加此值可以使HPA得到更全面的CPU使用数据。 **说明** 请合理设置该参数，值设置过低可能会在Pod刚进入就绪状态时，因CPU数据波动导致不恰当的扩容行为；而设置过高则可能导致在需要快速反应时HPA无法立即做出决策。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认：30s

名称	参数	详情	取值
控制器访问kube-apiserver的QPS	kube-api-qps	与kube-apiserver通信的qps	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200
控制器访问kube-apiserver的突发流量上限	kube-api-burst	与kube-apiserver通信的burst	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200
终止状态pod触发回收的数量阈值	terminated-pod-gc-threshold	集群中可保留的终止状态Pod数量，终止状态Pod超出该数量时将会被删除。 **说明** 该参数设置为0时，表示保留所有终止状态的Pod。	默认：1000 取值范围为10-12500 集群版本为v1.21.11-r40、v1.23.8-r0、v1.25.6-r0、v1.27.3-r0及以上时，取值范围调整为0-100000
可用区亚健康阈值	unhealthy-zone-threshold	当可用区故障节点规模达到指定比例时被认定为不健康，针对不健康的区域，故障节点业务的迁移频率会降级，避免规模故障场景下大规模迁移操作产生更坏的影响。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。 **说明** 比例配置过大可能导致区域在规模故障场景下仍尝试执行大规模迁移动作，导致集群过载等风险。	默认：0.55 取值范围为0-1

名称	参数	详情	取值
节点迁移速率	node-eviction-rate	当某区域健康时，在节点故障的情况下每秒删除 Pods的节点数。该值默认设置为0.1，代表每10 秒钟内至多从一个节点驱逐Pods。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。 **说明** 迁移速率设置过大可能引入集群过载风险，同时每批迁移重调度的pod过多，大量pod无法及时调度，影响整体故障恢复时间。	默认：0.1
次级节点迁移速率	secondary-node-eviction-rate	当某区域不健康时，在节点故障的情况下每秒删除Pods的节点数。该值默认设置为0.01，代表每100秒钟内至多从一个节点驱逐Pods。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。 **说明** 区域亚健康场景迁移速率设置过大无实际意义，且可能引入集群过载风险。	默认：0.01 配合node-eviction-rate设置，一般建议设置为node-eviction-rate的十分之一。
大规模集群大小阈值	large-cluster-size-threshold	集群内节点数量大于此参数时，集群被判断为大规模集群。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。 **说明** 被视为大型集群时，kube-controller-manager 会进行特定配置调整。这些配置用来优化大规模集群性能。因此阈值如果过低，规模小的集群用上的大集群的配置，反而降低性能。	默认：50 在拥有大量节点的集群中，适当增加此阈值可以帮助提高控制器的性能和响应速度。对于规模较小的集群，保持默认值即可。在调整此参数时，建议先在测试环境中验证其对性能的影响，然后再在生产环境中应用。

表 2-19 网络组件配置（仅 CCE Turbo 集群支持）

名称	参数	详情	取值
集群级别的节点最少绑定容器网卡数	nic-minimum-target	保障节点最少有多少张容器网卡绑定在节点上。 参数值需为正整数。例如10，表示节点最少有10张容器网卡绑定在节点上。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	默认：10
集群级别的节点预热容器网卡上限检查值	nic-maximum-target	当节点绑定的容器网卡数超过节点预热容器网卡上限检查值(nic-maximum-target)，不再主动预热容器网卡。 当该参数大于等于节点最少绑定容器网卡数(nic-minimum-target)时，则开启预热容器网卡上限值检查；反之，则关闭预热容器网卡上限值检查。 参数值需为正整数。例如0，表示关闭预热容器网卡上限值检查。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	默认：0
集群级别的节点动态预热容器网卡数	nic-warm-target	当Pod使用完节点最少绑定容器网卡数(nic-minimum-target)后，会始终额外预热多少张容器网卡，只支持数值配置。 当 节点动态预热容器网卡数(nic-warm-target) + 节点当前绑定的容器网卡数 大于 节点预热容器网卡上限检查值(nic-maximum-target) 时，只会预热nic-maximum-target与节点当前绑定的容器网卡数的差值。	默认：2

名称	参数	详情	取值
集群级别的节点预热容器网卡回收阈值	nic-max-above-warm-target	只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数 (nic-warm-target) 大于此阈值时，才会触发预热容器网卡的解绑回收。只支持数值配置。 ● 调大此值会减慢空闲容器网卡的回收，加快Pod的启动速度，但会降低IP地址的利用率，特别是在IP地址紧张的场景，**请谨慎调大**。 ● 调小此值会加快空闲容器网卡的回收，提高IP地址的利用率，但在瞬时大量Pod激增的场景，部分Pod启动会稍微变慢。	默认：2
集群级别的节点绑定容器网卡数低水位	prebound-subeni-percentage	节点绑定容器网卡数高水位 **说明** 此参数配置废弃中，请采用其他4个容器网卡动态预热参数。	默认：0:0

表 2-20 网络组件配置（仅 VPC 网络模型的集群支持）

名称	参数	详情	取值
保留原有Pod IP的非伪装网段	nonMasqueradeCIDRs	在VPC网络集群中，集群内的容器如果想要访问集群外，则需要将源容器IP进行SNAT，转换为节点IP（伪装成节点与外部通信）。配置后，节点默认不会将该网段IP进行SNAT，即不进行这种伪装。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本支持该配置。 集群中的节点默认不会将目的IP为10.0.0.0/8,172.16.0.0/12,192.168.0.0/16 三个网段的报文进行SNAT，因为这三个网段CCE默认为私有网段，可以借由上层VPC直接将报文送达（即将这三个网段视为集群内的网络，默认三层可达）。	默认：10.0.0.0/8,172.16.0.0/12,192.168.0.0/16 **说明** 如果需要保证节点能正常访问跨节点的Pod，必须添加节点的子网网段。 同理，如果同VPC下的其他ECS节点需要能正常访问Pod IP，必须添加ECS所在子网网段。

表 2-21 扩展控制器配置（仅 v1.21 及以上版本集群支持）

名称	参数	详情	取值
启用资源配额管理	enable-resource-quota	创建Namespace时是否自动创建ResourceQuota对象。通过配额管理功能，用户可以对命名空间或相关维度下的各类负载数量以及资源上限进行控制。 • 关闭：不自动创建ResourceQuota对象。 • 开启：自动创建ResourceQuota对象。ResourceQuota的默认取值请参见**设置资源配额及限制**。 **说明** 在高并发场景下（如批量创建Pod），配额管理机制可能导致部分请求因冲突而失败，除非必要不建议启用该功能；如启用，请确保请求客户端具备重试机制。	默认：关闭

步骤4 单击“确定”，完成配置操作。

----结束

参考链接

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

2.4.2 开启集群过载控制

操作场景

过载控制开启后，将根据控制节点的资源压力，动态调整系统外LIST请求的并发限制，维护控制节点和集群的可靠性。

约束与限制

集群版本需为v1.23及以上。

开启集群过载控制

方式一：创建集群时开启

创建v1.23及以上集群时，可在创建集群过程中，开启过载控制选项。

图 2-14 创建集群时开启过载控制

方式二：已有集群中开启

步骤1 登录CCE控制台，进入一个已有的集群（集群版本为v1.23及以上）。

步骤2 在“总览”页面，查看控制节点信息，如集群未开启过载控制，此处将会出现相应提示。如需开启过载控制，您可单击“立即开启”。

图 2-15 已有集群开启过载控制

----结束

集群过载监控

方式一：CCE界面

步骤1 登录CCE控制台，进入一个已有的集群（集群版本为v1.23及以上）。

步骤2 在“总览”页面，查看控制节点信息，将会显示“过载等级”指标。

过载等级如下：

- 熔断：拒绝所有外部流量
- 重度过载：拒绝75%外部流量
- 中度过载：拒绝50%外部流量
- 轻度过载：拒绝25%外部流量
- 正常：不拒绝外部流量

----结束

方式二：应用运维管理界面

您可登录应用运维管理控制台，创建一个仪表盘，并在仪表盘中添加名为vein_overload_level的监控指标，详情请参见**仪表盘**。

监控指标对应的含义如下：

- 0：熔断，拒绝所有外部流量
- 1：重度过载，拒绝75%外部流量
- 2：中度过载，拒绝50%外部流量
- 3：轻度过载，拒绝25%外部流量
- 4：正常，不拒绝外部流量

此处以应用运维管理的操作步骤为例，供您参考。

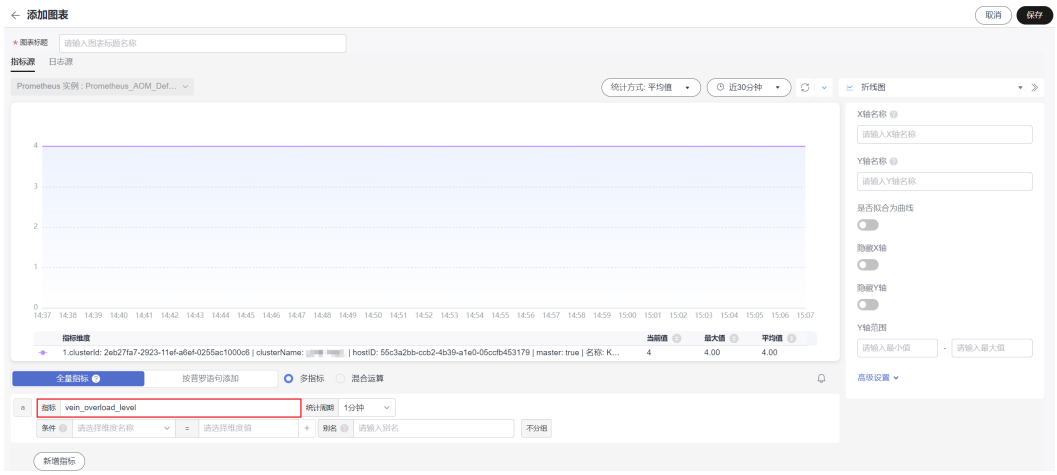
步骤1 登录应用运维管理控制台，在左侧导航栏中选择“仪表盘”，单击“创建仪表盘”。

图 2-16 创建仪表盘

步骤2 仪表盘创建完成后，添加指标图表。

- 图表标题：集群过载指标
- 添加方式：全量指标
- 指标名称：vein_overload_level
- 范围：可不选

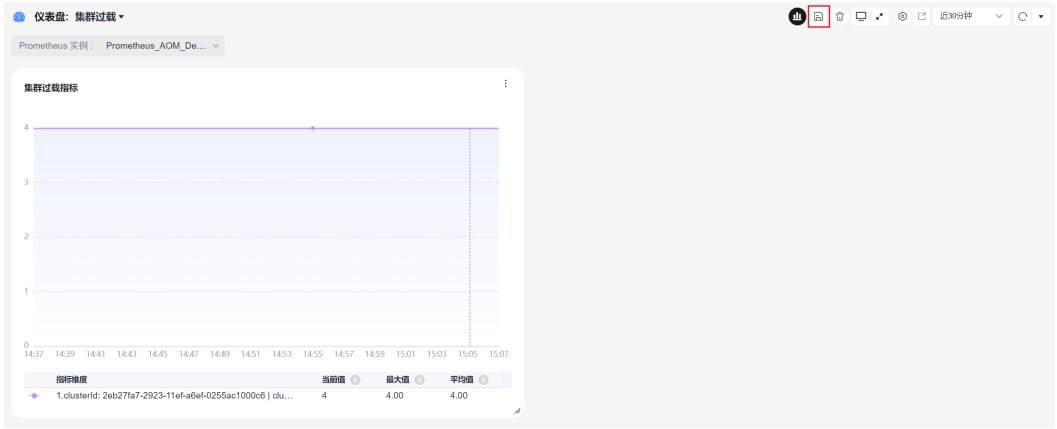
图 2-17 添加指标图表



步骤3 以上参数填写完成后，单击右上角“保存”，完成指标图表创建。

步骤4 单击右上角 按钮，保存仪表盘设计。

图 2-18 保存仪表盘



----结束

集群过载告警

此处以应用运维管理的操作步骤为例，供您参考。

步骤1 登录应用运维管理控制台，在左侧导航栏中选择“告警管理 > 告警规则”，单击“创建”。

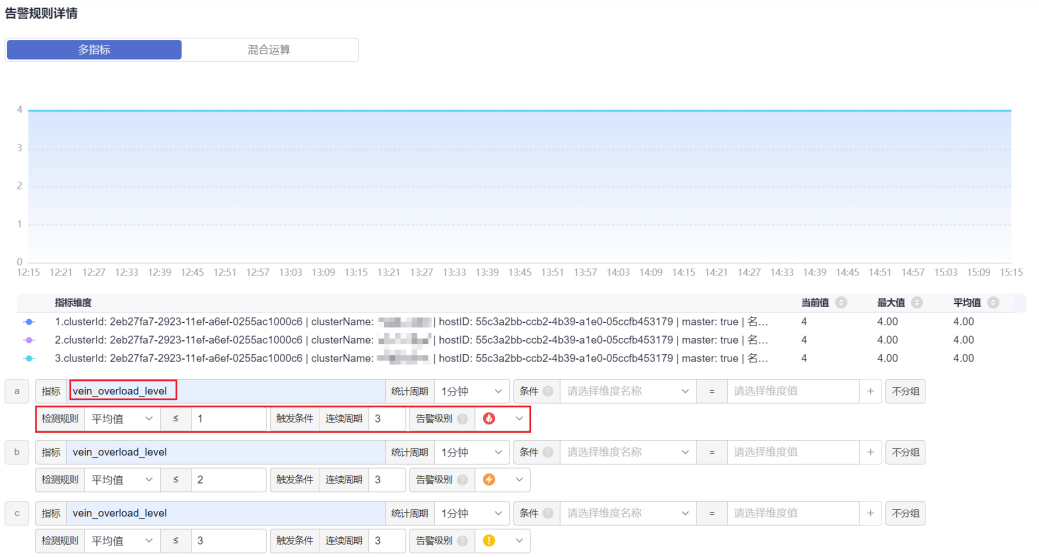
步骤2 填写以下参数：

- 规则名称：集群过载告警
- 企业项目：default
- 规则类型：指标告警规则
- 配置方式：按全量指标
- 告警条件：选择vein_overload_level，范围可选择对应集群或者选择全部。

- 当指标值小于等于1时，表示集群重度过载，推荐设置紧急告警。
- 当指标值小于等于2时，表示集群中度过载，推荐设置重要告警。
- 当指标值小于等于3时，表示集群轻度过载，推荐设置次要告警。

其余参数可按需求填写。

图 2-19 配置集群过载告警



步骤3 单击“立即创建”。

----结束

关闭集群过载控制

步骤1 登录CCE控制台，进入一个已有的集群（集群版本为v1.23及以上）。

步骤2 在左侧导航栏中选择“配置中心”。

步骤3 在“集群访问配置”中，将“开启过载防护（support-overload）”设置为关闭状态。

步骤4 单击“确定”保存修改。

----结束

2.4.3 变更集群规格

操作场景

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

约束限制

- 单控制节点的集群不允许变更到1000节点及以上。
- 变更集群规格不支持修改控制节点数量。
- 变更集群规格目前只支持扩容到更大规格，不支持降低集群规格。

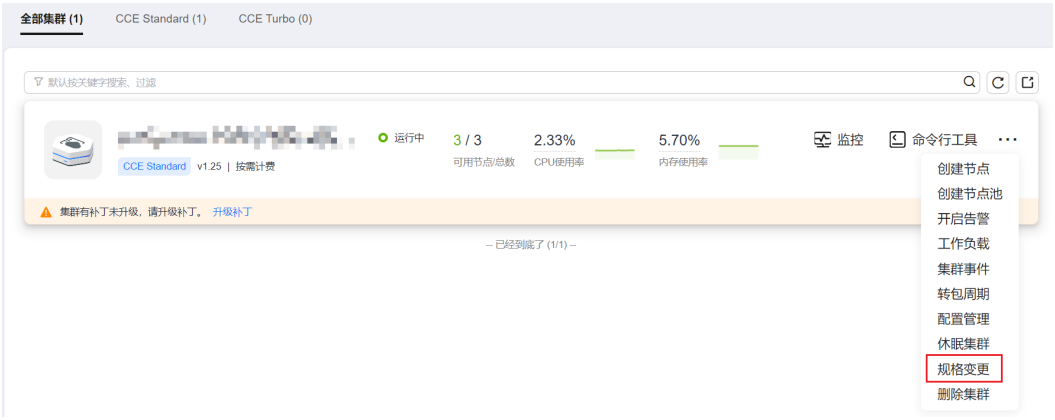
- 规格变更期间，控制节点存在开关机动作，集群将无法正常使用，规格变更期间请勿进行业务资源创建更新操作。

操作步骤

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要变更规格的集群，查看集群的更多操作，并选择“规格变更”。

图 2-20 变更规格



步骤3 在弹出的页面中，根据实际需求选择新的“集群规模”。

步骤4 单击“下一步”进行规格确认，并单击“确定”。

您可以在控制台右上角单击“操作记录”查看集群变更记录。状态从“执行中”变为“成功”，表示集群规格变更成功。

说明

当集群规格变更为1000节点及以上时，为了保证集群性能，集群部分参数值会根据集群的规格进行自动调整，详情请参见**修改CCE集群配置**。

图 2-21 操作记录

----结束

2.4.4 更改集群节点的默认安全组

操作场景

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

约束与限制

- 一个安全组关联的实例数量建议不超过1000个，否则可能引起安全组性能下降。更多关于安全组的限制请参考**安全组限制**。
- 请谨慎修改集群Master节点的安全组规则，详情请参见**集群安全组规则配置**。

操作步骤

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 单击集群名称，查看总览页面。

步骤3 在“网络信息”中单击“节点默认安全组”后的“编辑”按钮。

图 2-22 节点默认安全组

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

须知

- 请确认选择的安全组设置了正确的端口规则，否则将无法成功创建节点。安全组需要满足的端口规则根据集群类别存在差异，详情请参见**集群安全组规则配置**。
- 新安全组只对新创建或纳管的节点生效，存量节点需要手动修改节点安全组规则，即使对存量节点进行重置，也仍会使用原安全组。如需批量修改存量节点的安全组设置，请参考**如何批量修改集群node节点安全组？** 。

图 2-23 编辑节点默认安全组

----结束

2.4.5 删除集群

操作场景

- 按需计费的集群支持直接删除，详情请参见**删除按需计费的集群**。
- 包周期的集群不能直接删除，需进行集群退订（对于未超期集群）或释放（对于已超期未续费集群），详情请参见**退订/释放包周期的集群**。

注意事项

- 删除集群不会删除集群下包周期的资源，相关资源在集群删除后将会继续计费，请妥善处理。
- 删除集群会删除集群下工作负载与服务，相关业务将无法恢复。在执行操作前，请确保相关数据已完成备份或者迁移。
- 如果在删除集群时选择同步删除节点，将会同步删除节点挂载的系统盘和数据盘，请提前做好数据备份。
- 在集群非运行状态（例如冻结、不可用状态）时删除集群，会残留存储、网络等关联资源，请妥善处理。

删除按需计费的集群

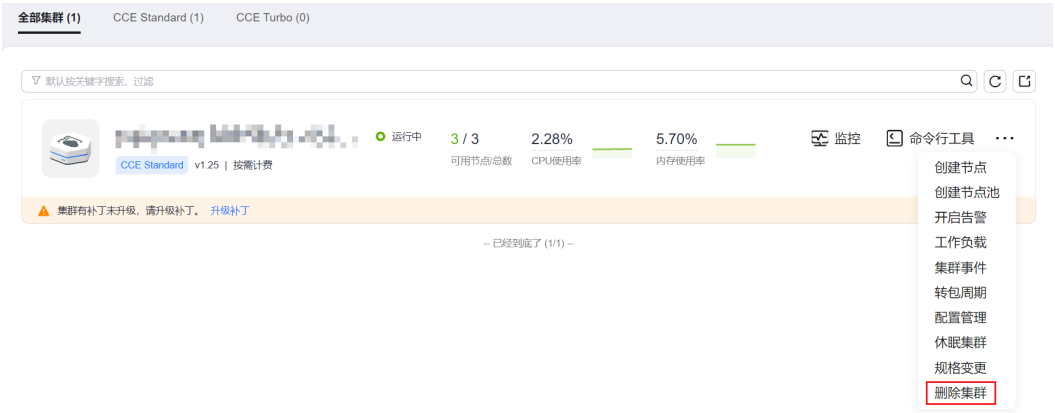
须知

处于休眠状态的集群无法直接删除，请将集群唤醒后重试。

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要删除的集群，查看集群的更多操作，并单击“删除集群”。

图 2-24 删除集群



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

- 删除集群节点，可支持以下操作选项：
  - 保留：保留服务器、系统盘和数据盘数据。
  - 删除：删除服务器（包周期节点不支持此选项，请进行手动退订）。
  - 重置：保留并重置服务器，系统盘和数据盘数据不保留。
- 删除集群下工作负载挂载的云存储。

  说明

  选择删除集群中存储卷绑定的底层云存储资源时，存在如下约束：
  - 底层存储依据存储卷指定的回收策略进行删除。例如，存储卷指定回收策略为Retain，则在删除集群后底层存储会保留。
  - 对象存储桶下存在大量文件（超过1000）时，请先手动清理桶内文件后再执行集群删除操作。

  勾选后，对于不同类型的PV删除逻辑如下：

PV类型	关联的底层存储资源	回收策略为Delete时，PV关联的底层存储资源是否删除
云硬盘	磁盘	是
专属存储	磁盘	是
文件存储	SFS容量型或通用文件系统（SFS 3.0）	是
对象存储	对象桶或并行文件系统	是
极速文件存储	SFS Turbo文件系统	是

PV类型	关联的底层存储资源	回收策略为Delete时，PV关联的底层存储资源是否删除
本地持久卷	节点上的逻辑卷	是否删除节点上的逻辑卷与删除集群时选择的节点操作策略有关。 如果选择保留节点，则逻辑卷不会删除；如果选择删除节点或重置节点，则逻辑卷会被删除。
通用文件系统子目录（SFS 3.0）	通用文件系统（SFS 3.0）中的一个目录	否
SFS Turbo子目录	SFS Turbo文件系统中的一个目录	否

- 删除集群下负载均衡ELB等网络资源（仅删除自动创建的ELB资源）。
- 删除云日志服务下该集群对应的日志组（删除日志组及组内所有日志流）。

  说明

  如未勾选日志将不会被删除，因此可能会产生LTS日志费用，详情请参考**价格计算器**。

步骤4 请输入“DELETE”，单击“是”，开始执行删除集群操作。

删除集群需要花费1~3分钟，请耐心等候。

----结束

退订/释放包周期的集群

须知

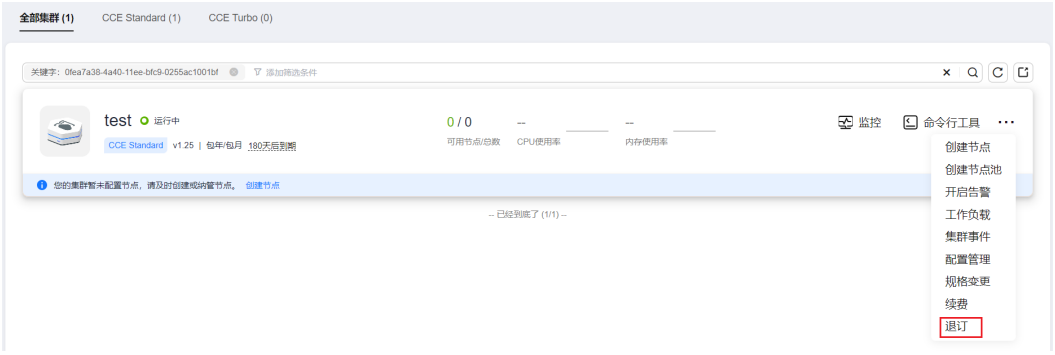
- 退订/释放集群仅退订订单关联的资源，非关联资源不会处理，相关资源会继续计费，请妥善处理。
- 对于包周期集群，如果过保留期会自动释放，集群下的节点如果同时到期会一并释放，节点如果未到期CCE不会对其做任何操作，相关数据会继续保留，相关资源会继续计费。请关注您账号下到期未续费集群，及时续费，防止节点被重装导致数据丢失。
- 若订单中存在主从关系的资源，需分别进行退订。
- 资源退订，相关注意事项请参见**退订规则说明**。
  - 如果您正在退订使用中的资源，请仔细确认资源信息和退款信息，资源退订后将无法恢复。若您要保留资源，仅退订未使用的续费周期，请**退订续费周期**。
  - 如果您退订的资源与其他包年/包月资源关联，退订操作可能会影响关联资源的正常使用，请谨慎操作。

    如果已与其他按需资源关联，退订操作完成后，关联的按需资源可以正常使用并计费。
  - 如果您本次退订属于非五天无理由退订，需收取手续费和已消费金额，已使用的代金券和折扣券不退还。

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要退订的集群，查看集群的更多操作，并单击“退订”或“释放”。

图 2-25 集群退订



步骤3 在弹出的“退订”或“释放”页面中，勾选要释放的资源。

- 删除集群节点，可支持以下操作选项：
  - 保留：保留服务器、系统盘和数据盘数据。
  - 删除：删除服务器（包周期节点不支持此选项，请进行手动退订）。
  - 重置：保留并重置服务器，系统盘和数据盘数据不保留。
- 删除集群下工作负载挂载的云存储。

  说明

  选择删除集群中存储卷绑定的底层云存储资源时，存在如下约束：
  - 底层存储依据存储卷指定的回收策略进行删除。例如，存储卷指定回收策略为Retain，则在删除集群后底层存储会保留。
  - 对象存储桶下存在大量文件（超过1000）时，请先手动清理桶内文件后再执行集群删除操作。

  勾选后，对于不同类型的PV删除逻辑如下：

PV类型	关联的底层存储资源	回收策略为Delete时，PV关联的底层存储资源是否删除
云硬盘	磁盘	是
专属存储	磁盘	是
文件存储	SFS容量型或通用文件系统（SFS 3.0）	是
对象存储	对象桶或并行文件系统	是
极速文件存储	SFS Turbo文件系统	是
本地持久卷	节点上的逻辑卷	是否删除节点上的逻辑卷与删除集群时选择的节点操作策略有关。 如果选择保留节点，则逻辑卷不会删除；如果选择删除节点或重置节点，则逻辑卷会被删除。

PV类型	关联的底层存储资源	回收策略为Delete时，PV关联的底层存储资源是否删除
通用文件系统子目录（SFS 3.0）	通用文件系统（SFS 3.0）中的一个目录	否
SFS Turbo子目录	SFS Turbo文件系统中的一个目录	否

- 删除集群下负载均衡ELB等网络资源（仅删除自动创建的ELB资源）。
- 删除云日志服务下该集群对应的日志组（删除日志组及组内所有日志流）。

  说明

  如未勾选日志将不会被删除，因此可能会产生LTS日志费用，详情请参考**价格计算器**。

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

----结束

2.4.6 休眠/唤醒按需计费集群

操作场景

当按需计费的集群暂时不需要使用时，您可以将其设置为休眠状态，有助于节省成本并减少资源浪费。

集群休眠后，将无法在此集群上创建和管理工作负载等资源。

注意事项

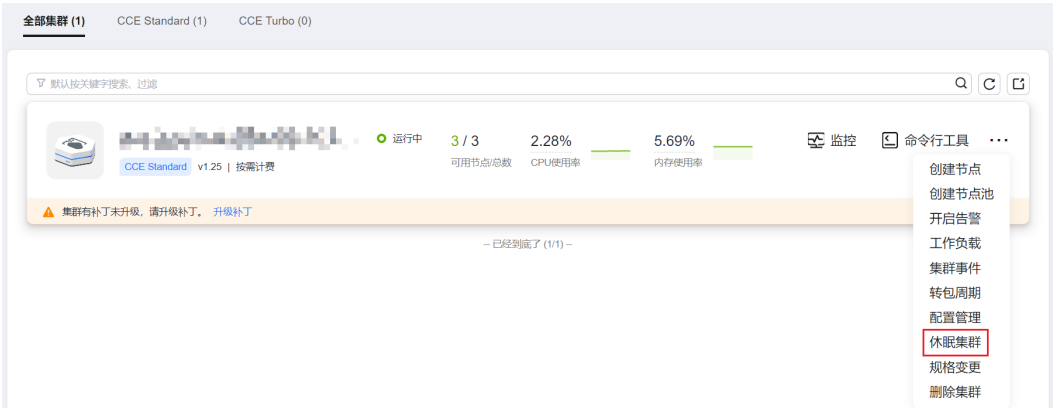
- 集群唤醒过程中，可能会由于资源不足导致Master节点启动失败，从而导致集群唤醒失败，请过一段时间再次唤醒。
- 集群唤醒后，需要3~5分钟进行数据初始化。建议您等待集群稳定运行后再进行业务下发。

集群休眠

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要休眠的集群，查看集群的更多操作，并单击“休眠集群”。

图 2-26 休眠集群



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

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

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

图 2-27 集群休眠提示

----结束

集群唤醒

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 单击待唤醒集群栏的“唤醒集群”。

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

说明

集群唤醒后，将继续收取集群管理费用。

----结束

2.4.7 续费包年/包月集群

客户购买包周期集群后，支持续费包周期资源。

操作步骤

本节以计费模式为“包年/包月”的集群为例，介绍如何为购买的集群续费。

须知

包周期的集群超期未续费将会被系统删除，删除后集群内的节点以及运行的业务都将销毁，请务必及时续费或**开通自动续费**。

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要续费的集群，查看集群的更多操作，并单击“续费”。

图 2-28 续费集群

步骤3 在弹出的“续费”页面中，根据系统提示进行续费操作。

说明

- 您已选择操作的资源（高亮显示）和其他资源有关联关系，请确认是否同时操作。
- 在资源续费周期生效前，若您变更了该资源的配置，您将不能退订未生效的续费周期。
- 生效的续费周期不能享受5天无理由退订。

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

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

----结束

2.4.8 按需计费集群转包周期

当前在CCE中购买集群时支持“按需计费”和“包年/包月”（按周期）两种计费方式。按需计费的购买的集群可以转成包年/包月计费的集群。

如果您需要将按需计费的节点转为包年/包月计费，请参见**按需节点转包年/包月**。

按需集群转包年/包月

如果您在购买按需计费的集群后，想更换为包年/包月计费，可按如下步骤进行操作：

步骤1 登录CCE控制台，在左侧导航栏中选择“集群管理”。

步骤2 找到需要转包年/包月的集群，查看集群的更多操作，并单击“转包年包月”。

步骤3 在转包年包月页面中，选择需要转包年/包月的集群，您也可以同时选择需要转包年/包月的节点。

图 2-29 按需集群转包年/包月

步骤4 单击“确定”，等待生成订单并完成支付即可。

----结束

2.5 升级集群

2.5.1 升级集群的流程和方法

云容器引擎（CCE）严格遵循社区一致性认证，每年发布3个Kubernetes版本，每个版本发布后提供至少24个月的维护周期，CCE保证维护周期内的Kubernetes版本的稳定运行。

为了保障您的服务权益，请您务必在维护周期结束之前升级您的Kubernetes集群，您可在集群列表页面确认集群的Kubernetes版本，以及当前是否有新的版本可供升级。主动升级集群有以下好处：

- 降低安全和稳定性风险：Kubernetes版本迭代过程中，会不断修复发现的安全及稳定性漏洞，长久使用EOS版本集群会给业务带来安全和稳定性风险。
- 支持新功能和新操作系统：Kubernetes版本的迭代过程中，会不断带来新的功能、优化。您可通过**CCE集群版本发布说明**查看最新版本的特性说明。
- 避免大跨度兼容风险：Kubernetes版本的迭代过程中，会不断带来API变更与功能废弃。长久未升级的集群，在需要升级时需要更大的运维保障投入。周期性的跟随升级能有效缓解版本差异累积导致的兼容性风险。建议用户每季度升级一次补丁版本，每年升级一次大版本至当前支持的最新版本。

- 更加有效的技术支持：对于EOS的Kubernetes版本集群，CCE不再提供安全补丁和问题修复，同样无法保证EOS版本集群的技术支持质量。

集群升级路径

CCE 集群基于社区Kubernetes版本迭代演进，版本号由社区Kubernetes版本和CCE补丁版本两部分构成，因此提供两类集群升级路径。

- Kubernetes版本升级：

Kubernetes版本号	支持升级到的Kubernetes版本号
v1.13及以下	不支持
v1.15	v1.19
v1.17	v1.19
v1.19	v1.21、v1.23
v1.21	v1.23、v1.25
v1.23	v1.25、v1.27、v1.28
v1.25	v1.27、v1.28
v1.27	v1.28
v1.28	v1.29
v1.29	/

说明

- 已停止维护的版本无法一步直接升级到最新版本，需要进行连续多次升级，例如v1.15 -> v1.19 -> v1.23 -> v1.27/v1.28。
- 补丁版本需要升级至最新补丁后方可进行Kubernetes版本升级，控制台将根据当前集群版本自动为您生成最佳升级路径。

- 补丁版本升级

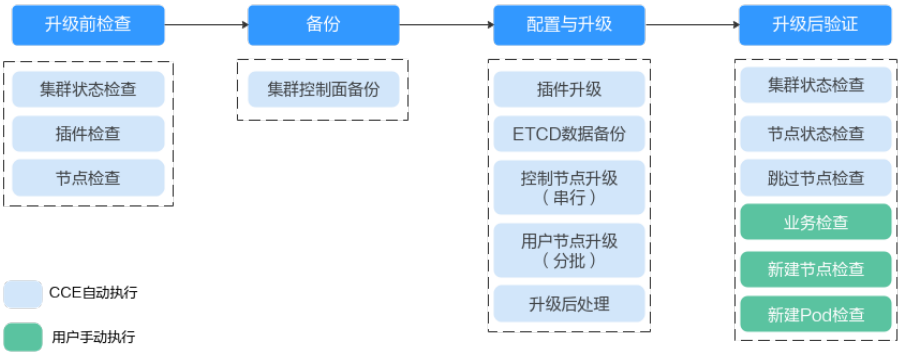
  1.19及以上版本的CCE集群采取了补丁版本管理策略，旨在不进行大版本升级的情况下，为在维集群提供新的特性、Bugfix和漏洞修复。

  新的补丁版本发布以后您可在任意补丁版本一次直升到最新补丁版本，补丁版本发布记录请参见**补丁版本发布记录**。

集群升级流程

集群升级流程包括升级前检查、备份、升级和升级后验证几个步骤，下面介绍集群升级过程中的相关流程。

图 2-30 集群升级流程



在确定集群的目标版本后，请您仔细阅读升级**注意事项**，避免升级时出现功能不兼容的问题。

1. **升级前检查**

   升级集群前，CCE会对您的集群进行必要的检查，包括集群状态、插件状态、节点状态、工作负载兼容性等多方面进行检查，确保集群满足升级的条件，检查项目请参考**升级前检查项**。如果出现检查异常项，请参考控制台中的提示进行修复。

2. **备份**

   通过硬盘快照的方式帮您备份集群控制节点，以保存CCE组件镜像、组件配置、Etcd数据等关键数据。建议您在升级前进行备份。如果在升级过程中出现不可预期的情况，可以基于备份为您快速恢复集群。

备份方式	备份对象	备份方式	备份时间	回滚时间	说明
etcd数据备份	etcd数据	升级流程中自动备份	1-5min	2h	必选备份，升级过程中自动进行，用户无需关注
CBR整机备份	Master节点磁盘，包括组件镜像、配置、日志以及etcd数据	通过页面一键备份（手动触发）	20min-2h（受当前区域云备份任务排队情况影响）	20min	该功能逐步由EVS快照备份替代
EVS快照备份	Master节点磁盘，包括组件镜像、配置、日志以及etcd数据	通过页面一键备份（手动触发）	1-5min	20min	该功能上线中 对于已上线的区域，EVS快照备份将替代CBR整机备份

3. **配置与升级**

   执行升级前，需要对升级参数进行配置，我们已为您提供了默认配置，您也可以根据需要进行配置，升级参数配置完成后，将进入正式升级流程，对插件、控制节点、用户节点依次进行升级。

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

     说明

     插件右侧如有⚠标记，表示当前插件不能同时兼容集群升级起始和目标版本，在集群版本升级完成后将为您升级该插件 ，该插件在集群升级过程中可能无法正常使用。

   - **节点升级配置**

     - **每批最大升级节点数**：您可以设置每批升级的最大节点数量。

       升级时节点池之间会依次进行升级。节点池内的节点分批升级，第一批升级1个节点，第二批升级2个节点，后续每批升级节点数以2的幂数增加，直到达到您设置的每批最大升级节点数，并会持续作用在下一个节点池中。默认每批最大升级节点数为20，最高可配置为60。

     - **节点优先级配置**：您可以自行定义节点升级的优先级顺序。如不设置该优先级，系统将根据默认策略生成优先级顺序执行升级。
       - 添加节点池优先级：自定义节点池升级的优先级顺序。如不设置，默认策略为节点数量少的节点池优先升级。
       - 添加节点优先级：自定义节点池内节点升级的优先级顺序。如不设置，默认策略为负载较轻（根据节点Pod数量、节点资源请求率、节点PV数量等维度计算负载情况）的节点优先升级。

     - **节点升级批次应用范围**：您可以自定义选择，默认为集群。
       - 节点升级批次配置应用到整个集群，整个升级过程不重置升级批次。
       - 节点升级批次配置应用范围为节点池，升级到每个节点池重置升级批次。

4. **升级后验证**

   升级完成后，会自动为集群执行集群状态检查、节点状态检查等，您需要手动进行业务验证、新建节点验证、新建Pod验证等，确保升级后集群功能正常。详情请参见**升级后验证**。

升级方式

表 2-22 升级方式介绍

升级方式	介绍	升级范围	优点	约束
原地升级	节点上升级Kubernetes组件、网络组件和CCE管理组件，升级过程中业务Pod和网络均不受影响。 升级过程中，节点分批进行升级，存量节点将不可调度，升级完成的批次支持调度新业务。	• 节点操作系统不升级 • 插件在目标版本集群不兼容时自动升级 • K8s组件自动升级	可一键式升级，用户无需迁移业务，可以基本上保证业务不断。	原地升级仅在v1.15及以上版本集群支持。
迁移	将老版本集群的业务迁移到新版本集群，适用于需要大幅度跨版本集群升级的需求。	集群内资源均重新部署。	可避免老版本连续升级导致的版本不兼容问题。	-

2.5.2 升级前须知

升级前，您可以在CCE控制台确认您的集群是否可以进行升级操作。确认方法请参见**升级集群的流程和方法**。

注意事项

升级集群前，您需要知晓以下事项：

- **请务必慎重并选择合适的时间段进行升级，以减少升级对您的业务带来的影响。**
- 集群升级前，请参考**Kubernetes版本发布说明**了解每个集群版本发布的特性以及差异，否则可能因为应用不兼容新集群版本而导致升级后异常。例如，您需要检查集群中是否使用了目标版本废弃的API，否则可能导致升级后调用接口失败，详情请参见**废弃API说明**。

集群升级时，以下几点注意事项可能会对您的业务存在影响，请您关注：

- 集群升级过程中，不建议对集群进行任何操作。尤其是**关机、重启或删除节点**等操作，都会导致升级失败。
- 集群升级前，**请确认集群中未执行高危操作**，否则可能导致集群升级失败或升级后配置丢失。例如，常见的高危操作有本地修改集群节点的配置、通过ELB控制台修改CCE管理的监听器配置等。建议您通过CCE控制台修改相关配置，以便在升级时自动继承。
- 集群升级过程中，已运行工作负载业务不会中断，但API Server访问会短暂中断，如果业务需要访问API Server可能会受到影响。
- 集群升级过程中默认不限制应用调度。但下列旧版本集群升级过程中，仍然会给节点打上“node.kubernetes.io/upgrade”（效果为“NoSchedule”）的污点，并在集群升级完成后移除该污点。

- 所有v1.15集群
- 所有v1.17集群
- 补丁版本小于等于v1.19.16-r4的1.19集群
- 补丁版本小于等于v1.21.7-r0的1.21集群
- 补丁版本小于等于v1.23.5-r0的1.23集群

约束与限制

- 集群升级出现异常时，集群可通过备份数据进行回滚。若您在升级成功之后对集群进行了其它操作（例如变更集群规格），将无法再通过备份数据回滚。
- 集群升级至v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及之后的版本时，集群中若安装以下插件，需升级至对应版本，详情请参见**容器引擎和Kubelet共享磁盘空间说明**。
  - npd插件：需升级至1.18.10及以上版本
  - log-agent插件：需升级至1.3.0及以上版本
- 容器隧道网络集群升级至1.19.16-r4、1.21.7-r0、1.23.5-r0、1.25.1-r0及之后的版本时，会移除匹配目的地址是容器网段且源地址非容器网段的SNAT规则；如果您之前通过配置VPC路由实现集群外直接访问所有的Pod IP，升级后只支持直接访问对应节点上的Pod IP。
- NGINX Ingress控制器插件新版本（参见**版本记录**）配置优雅退出和ELB删除后端控制器宽限时间，支持无损升级。以下插件版本升级过程可能会出现服务短暂不可用，若集群中已安装下列版本的插件，请选择业务低峰期进行升级，具体版本如下：

插件版本	小版本范围
2.1.x	x < 32
2.2.x	x < 41
2.4.x	x < 4

- 集群升级场景涉及NetworkManager重启，触发DHCP Client主动续租，默认情况下将根据子网DNS配置刷新/etc/resolv.conf，建议您通过VPC控制台修改DNS，详情请参见**怎样修改云服务器的DNS服务器地址？**。
- 更多版本升级约束请查看**版本差异说明**。

版本差异说明

版本升级路径	版本差异	建议自检措施
v1.23/v1.25 升级至v1.27	容器运行时Docker不再被推荐使用，建议您使用Containerd进行替换，详情请参见**容器引擎说明**。	已纳入升级前检查。

版本升级路径	版本差异	建议自检措施
v1.21/ v1.19 升级至 v1.23	社区较老版本的Nginx Ingress Controller来说（社区版本v0.49及以下，对应CCE插件版本v1.x.x），在创建Ingress时没有指定Ingress类别为nginx，即annotations中未添加kubernetes.io/ingress.class: nginx的情况，也可以被Nginx Ingress Controller纳管。但对于较新版本的Nginx Ingress Controller来说（社区版本v1.0.0及以上，对应CCE插件版本2.x.x），如果在创建Ingress时没有显示指定Ingress类别为nginx，该资源将被Nginx Ingress Controller忽略，Ingress规则失效，导致服务中断。	已纳入升级前检查，也可参照**nginx-ingress插件升级检查**进行自检。
v1.19升级至 v1.21	Kubernetes v1.21集群版本修复了exec probe timeouts不生效的BUG，在此修复之前，exec 探测器不考虑 timeoutSeconds 字段。相反，探测将无限期运行，甚至超过其配置的截止日期，直到返回结果。 若用户未配置，默认值为1秒。升级后此字段生效，如果探测时间超过1秒，可能会导致应用健康检查失败并频繁重启。	升级前检查您使用了exec probe的应用的probe timeouts是否合理。
	CCE的v1.19及以上版本的kube-apiserver要求客户侧webhook server的证书必须配置Subject Alternative Names (SAN)字段。否则升级后kube-apiserver调用webhook server失败，容器无法正常启动。 根因：Go语言v1.15版本废弃了X.509 **CommonName**，CCE的v1.19版本的kube-apiserver编译的版本为v1.15，若客户的webhook证书没有Subject Alternative Names (SAN)，kube-apiserver不再默认将X509证书的CommonName字段作为hostname处理，最终导致认证失败。	升级前检查您自建webhook server的证书是否配置了SAN字段。 • 若无自建webhook server则不涉及。 • 若未配置，建议您配置使用SAN字段指定证书支持的IP及域名。

版本升级路径	版本差异	建议自检措施
v1.15升级至v1.19	CCE v1.19版本的控制面与v1.15版本的Kubelet存在兼容性问题。若Master节点升级成功后，节点升级失败或待升级节点发生重启，则节点有极大概率为NotReady状态。 主要原因为升级失败的节点有大概率重启kubelet而触发节点注册流程，v1.15 kubelet默认注册标签（failure-domain.beta.kubernetes.io/is-baremetal和kubernetes.io/availablezone）被v1.19版本kube-apiserver视为非法标签。 v1.19版本中对应的合法标签为node.kubernetes.io/baremetal和failure-domain.beta.kubernetes.io/zone。	1. 正常升级流程不会触发此场景。 2. 在Master升级完成后尽量避免使用暂停升级功能，快速升级完Node节点。 3. 若Node节点升级失败且无法修复，请尽快驱逐此节点上的应用，请联系技术支持人员，跳过此节点升级，在整体升级完毕后，重置该节点。

	CCE的v1.15版本集群及v1.19版本集群将docker的存储驱动文件系统由 xfs切换成ext4，可能会导致升级后的java应用Pod内的import包顺序异常，继而导致Pod异常。	升级前查看节点上docker配置文件/etc/docker/daemon.json。检查dm.fs配置项是否为xfs。
		● 若为ext4或存储驱动为overlay则不涉及。
		● 若为xfs则建议您在新版本集群预先部署应用，以测试应用与新版本集群是否兼容。
		{
		"storage-driver": "devicemapper",
		"storage-opts": [
		"dm.thinpooldev=/dev/mapper/vgpaas-thinpool",
		"dm.use_deferred_removal=true",
		"dm.fs=xfs",
		"dm.use_deferred_deletion=true"
		]
		}

版本升级路径	版本差异	建议自检措施
	CCE的v1.19及以上版本的kube-apiserver要求客户侧webhook server的证书必须配置Subject Alternative Names (SAN)字段。否则升级后kube-apiserver调用webhook server失败，容器无法正常启动。 根因：Go语言v1.15版本废弃了X.509 **CommonName**，CCE的v1.19版本的kube-apiserver编译的版本为v1.15。CommonName字段作为hostname处理，最终导致认证失败。	升级前检查您自建webhook server的证书是否配置了SAN字段。 • 若无自建webhook server则不涉及。 • 若未配置，建议您配置使用SAN字段指定证书支持的IP及域名。 **须知** 为减弱此版本差异对集群升级的影响，v1.15升级至v1.19时，CCE会进行特殊处理，仍然会兼容支持证书不带SAN。但后续升级不再特殊处理，请尽快整改证书，以避免影响后续升级。
	v1.17.17版本及以后的集群CCE自动给用户创建了PSP规则，限制了不安全配置的Pod的创建，如securityContext配置了sysctl的net.core.somaxconn的Pod。	升级后请参考资料按需开放非安全系统配置，具体请参见**PodSecurityPolicy配置**。
	1.15版本集群原地升级，如果业务中存在initContainer或使用Istio的场景，则需要注意以下约束： 1.16及以上的kubelet统计**QosClass**和之前版本存在差异，1.15及以下版本仅统计spec.containers下的容器，1.16及以上的kubelet版本会同时统计spec.containers和spec.initContainers下的容器，升级前后会造成Pod的QosClass变化，从而造成Pod中容器重启。	建议参考**表2-23**在升级前修改业务容器的QosClass规避该问题。
v1.13升级至v1.15	vpc集群升级后，由于网络组件的升级，master节点会额外占一个网段。在Master占用了网段后，无可用容器网段时，新建节点无法分配到网段，调度在该节点的pod会无法运行。	一般集群内节点数量快占满容器网段场景下会出现该问题。例如，容器网段为10.0.0.0/16，可用IP数量为65536，VPC网络IP分配是分配固定大小的网段（使用掩码实现，确定每个节点最多分配多少容器IP），例如上限为128，则此时集群最多支持65536/128=512个节点，然后去掉Master节点数量为509，此时是1.13集群支持的节点数。集群升级后，在此基础上3台Master节点会各占用1个网段，最终结果就是506台节点。

表 2-23 1.15 版本升级前后 QosClass 变化

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

废弃 API 说明

随着Kubernetes API的演化，API会周期性地被重组或升级，部分API会被弃用并被最终删除。以下为各Kubernetes社区版本中被废弃的API，更多已废弃的API说明请参见**已弃用 API 的迁移指南**。

- **Kubernetes社区v1.29版本中废弃的API**
- Kubernetes社区v1.28版本中无废弃的API
- **Kubernetes社区v1.27版本中废弃的API**
- **Kubernetes社区v1.25版本中废弃的API**
- **Kubernetes社区v1.22版本中废弃的API**
- **Kubernetes社区v1.16版本中废弃的API**

说明

当某API被废弃时，已经创建的资源对象不受影响，但新建或编辑该资源时将出现API版本被拦截的情况。

表 2-24 Kubernetes 社区 v1.29 版本中废弃的 API

资源名称	废弃API版本	替代API版本	变更说明
FlowSchema和PriorityLevelConfiguration	flowcontrol.apiserver.k8s.io/v1beta2	flowcontrol.apiserver.k8s.io/v1 （该API从社区v1.29版本开始可用） flowcontrol.apiserver.k8s.io/v1beta3 （该API从社区v1.26版本开始可用）	• **flowcontrol.apiserver.k8s.io/v1**中的显着变化：PriorityLevelConfiguration的spec.limited.assuredConcurrencyShares字段已被重命名为spec.limited.nominalConcurrencyShares，仅在未指定时默认为30，并且显式值0不会更改为 30。 • **flowcontrol.apiserver.k8s.io/v1beta3**中需要额外注意的变更：PriorityLevelConfiguration的spec.limited.assuredConcurrencyShares字段已被更名为spec.limited.nominalConcurrencyShares。

表 2-25 Kubernetes 社区 v1.27 版本中废弃的 API

资源名称	废弃API版本	替代API版本	变更说明
CSIStorageCapacity	storage.k8s.io/v1beta1	storage.k8s.io/v1 （该API从社区v1.24版本开始可用）	-
FlowSchema和PriorityLevelConfiguration	flowcontrol.apiserver.k8s.io/v1beta1	flowcontrol.apiserver.k8s.io/v1beta3 （该API从社区v1.26版本开始可用）	-
HorizontalPodAutoscaler	autoscaling/v2beta2	autoscaling/v2 （该API从社区v1.23版本开始可用）	-

表 2-26 Kubernetes 社区 v1.25 版本中废弃的 API

资源名称	废弃API版本	替代API版本	变更说明
CronJob	batch/v1beta1	batch/v1 （该API从社区v1.21版本开始可用）	-
EndpointSlice	discovery.k8s.io/v1beta1	discovery.k8s.io/v1 （该API从社区v1.21版本开始可用）	此次更新需注意以下变更： ● 每个Endpoint中，**topology["kubernetes.io/hostname"]**字段已被弃用，请使用**nodeName**字段代替。 ● 每个Endpoint中，**topology["kubernetes.io/zone"]**字段已被弃用，请使用**zone**字段代替。 ● **topology**字段被替换为**deprecatedTopology**，并且在 v1 版本中不可写入。

资源名称	废弃API版本	替代API版本	变更说明
Event	events.k8s.io/v1beta1	events.k8s.io/v1 （该API从社区v1.19版本开始可用）	此次更新需注意以下变更： • **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

资源名称	废弃API版本	替代API版本	变更说明
			events.k8s.io/v1 Event 对象中）。
HorizontalPodAutoscaler	autoscaling/v2beta1	autoscaling/v2 （该API从社区v1.23版本开始可用）	-
PodDisruptionBudget	policy/v1beta1	policy/v1 （该API从社区v1.21版本开始可用）	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。
PodSecurityPolicy	policy/v1beta1	-	从社区v1.25版本开始，PodSecurityPolicy资源不再提供**policy/v1beta1**版本的API，并且PodSecurityPolicy准入控制器也会被删除。 请使用**Pod Security Admission配置**替代。
RuntimeClass	node.k8s.io/v1beta1	node.k8s.io/v1 （该API从社区v1.20版本开始可用）	-

表 2-27 Kubernetes 社区 v1.22 版本中废弃的 API

资源名称	废弃API版本	替代API版本	变更说明
MutatingWebhookConfiguration ValidatingWebhookConfiguration	admissionregistration.k8s.io/v1beta1	admissionregistration.k8s.io/v1 （该API从社区v1.16版本开始可用）	• **webhooks[*].failurePolicy** 在 v1 版本中默认值从 **Ignore** 改为 **Fail**。 • **webhooks[*].matchPolicy** 在 v1 版本中默认值从 **Exact** 改为 **Equivalent**。 • **webhooks[*].timeoutSeconds** 在 v1 版本中默认值从 **30s** 改为 **10s**。 • **webhooks[*].sideEffects** 的默认值被删除，并且该字段变为必须指定； 在 v1 版本中可选的值只能是 **None** 和 **NoneOnDryRun** 之一。 • **webhooks[*].admissionReviewVersions** 的默认值被删除，在 v1 版本中此字段变为必须指定（**AdmissionReview** 的被支持版本包括 v1 和 v1beta1）。 • **webhooks[*].name** 必须在通过 **admissionregistration.k8s.io/v1** 创建的对象列表中唯一。

资源名称	废弃API版本	替代API版本	变更说明
CustomResourceDefinition	apiextensions.k8s.io/v1beta1	apiextensions/v1 （该API从社区v1.16版本开始可用）	• **spec.scope** 的默认值不再是 **Namespaced**，该字段必须显式指定。 • **spec.version** 在 v1 版本中被删除，应改用 **spec.versions**。 • **spec.validation** 在 v1 版本中被删除，应改用 **spec.versions[*].schema**。 • **spec.subresources** 在 v1 版本中被删除，应改用 **spec.versions[*].subresources**。 • **spec.additionalPrinterColumns** 在 v1 版本中被删除，应改用 **spec.versions[*].additionalPrinterColumns**。 • **spec.conversion.webhookClientConfig** 在 v1 版本中被移动到 **spec.conversion.webhook.clientConfig** 中。 • **spec.conversion.conversionReviewVersions** 在 v1 版本中被移动到 **spec.conversion.webhook.conversionReviewVersions** • **spec.versions[*].schema.openAPIV3Schema** 在创建 v1 版本的 **CustomResourceDefinition** 对象时变成必需字段，并且其取值必须是一个**结构化的 Schema**。 • **spec.preserveUnknownFields: true** 在创建 v1 版本的 **CustomResourceDefinition** 对象时不允许指定；该配置必须在 Schema 定义中使用 **x-kubernetes-preserve-unknown-fields: true** 来设置。 • 在 v1 版本中，**additionalPrinterColumns** 的条目中的 **JSONPath** 字段

资源名称	废弃API版本	替代API版本	变更说明
			被更名为 **jsonPath**（补丁 **#66531**）。
APIService	apiregistration/v1beta1	apiregistration.k8s.io/v1 （该API从社区v1.10版本开始可用）	-
TokenReview	authentication.k8s.io/v1beta1	authentication.k8s.io/v1 （该API从社区v1.6版本开始可用）	-
LocalSubjectAccessReview SelfSubjectAccessReview SubjectAccessReview SelfSubjectRulesReview	authorization.k8s.io/v1beta1	authorization.k8s.io/v1 （该API从社区v1.16版本开始可用）	**spec.group** 在 v1 版本中被更名为 **spec.groups**（补丁 **#32709**）

资源名称	废弃API版本	替代API版本	变更说明
CertificateSigningRequest	certificates.k8s.io/v1beta1	certificates.k8s.io/v1 （该API从社区v1.19版本开始可用）	**certificates.k8s.io/v1** 中需要额外注意的变更： • 对于请求证书的 API 客户端而言： – **spec.signerName** 现在变成必需字段（参阅 **已知的 Kubernetes 签署者**）， 并且通过 certificates.k8s.io/v1 API 不可以创建签署者为 **kubernetes.io/legacy-unknown** 的请求。 – **spec.usages** 现在变成必需字段，其中不可以包含重复的字符串值， 并且只能包含已知的用法字符串。 • 对于要批准或者签署证书的 API 客户端而言： – **status.conditions** 中不可以包含重复的类型。 – **status.conditions[*].status** 字段现在变为必需字段。 – **status.certificate** 必须是 PEM 编码的，而且其中只能包含 CERTIFICATE 数据块。
Lease	coordination.k8s.io/v1beta1	coordination.k8s.io/v1 （该API从社区v1.14版本开始可用）	-

资源名称	废弃API版本	替代API版本	变更说明
Ingress	networking.k8s.io/v1beta1 extensions/v1beta1	networking.k8s.io/v1 （该API从社区v1.19版本开始可用）	• **spec.backend** 字段被更名为 **spec.defaultBackend**。 • 后端的 **serviceName** 字段被更名为 **service.name**。 • 数值表示的后端 **servicePort** 字段被更名为 **service.port.number**。 • 字符串表示的后端 **servicePort** 字段被更名为 **service.port.name**。 • 对所有要指定的路径，**pathType** 都成为必需字段。 可选项为 **Prefix**、**Exact** 和 **ImplementationSpecific**。要匹配 v1beta1 版本中未定义路径类型时的行为，可使用 **ImplementationSpecific**。
IngressClass	networking.k8s.io/v1beta1	networking.k8s.io/v1 （该API从社区v1.19版本开始可用）	-
ClusterRole ClusterRoleBinding Role RoleBinding	rbac.authorization.k8s.io/v1beta1	rbac.authorization.k8s.io/v1 （该API从社区v1.8版本开始可用）	-
PriorityClass	scheduling.k8s.io/v1beta1	scheduling.k8s.io/v1 （该API从社区v1.14版本开始可用）	-

资源名称	废弃API版本	替代API版本	变更说明
CSIDriver CSINode StorageClass VolumeAttachment	storage.k8s.io/v1beta1	storage.k8s.io/v1	• CSIDriver从社区v1.19版本开始在**storage.k8s.io/v1**中提供。 • CSINode从社区v1.17版本开始在**storage.k8s.io/v1**中提供。 • StorageClass从社区v1.6 版本开始在**storage.k8s.io/v1**中提供。 • VolumeAttachment从社区v1.13版本开始在**storage.k8s.io/v1**中提供。

表 2-28 Kubernetes 社区 v1.16 版本中废弃的 API

资源名称	废弃API版本	替代API版本	变更说明
NetworkPolicy	extensions/v1beta1	networking.k8s.io/v1 （该API从社区v1.8版本开始可用）	-
DaemonSet	extensions/v1beta1 apps/v1beta2	apps/v1 （该API从社区v1.9版本开始可用）	• **spec.templateGeneration**字段被删除。 • **spec.selector** 现在变成必需字段，并且在对象创建之后不可变更； 可以将现有模板的标签作为选择算符以实现无缝迁移。 • **spec.updateStrategy.type**的默认值变为**RollingUpdate**（**extensions/v1beta1** API版本中的默认值是**OnDelete**）。

资源名称	废弃API版本	替代API版本	变更说明
Deployment	extensions/v1beta1 apps/v1beta1 apps/v1beta2	apps/v1 （该API从社区v1.9版本开始可用）	• **spec.rollbackTo** 字段被删除。 • **spec.selector** 字段现在变为必需字段，并且在 Deployment 创建之后不可变更； 可以使用现有的模板的标签作为选择算符以实现无缝迁移。 • **spec.progressDeadlineSeconds** 的默认值变为 600 秒（**extensions/v1beta1** 中的默认值是没有期限）。 • **spec.revisionHistoryLimit** 的默认值变为 10（**apps/v1beta1** API 版本中此字段默认值为 2，在**extensions/v1beta1** API 版本中的默认行为是保留所有历史记录）。 • **maxSurge** 和 **maxUnavailable** 的默认值变为 25%（在 **extensions/v1beta1** API 版本中，这些字段的默认值是 1）。
StatefulSet	apps/v1beta1 apps/v1beta2	apps/v1 （该API从社区v1.9版本开始可用）	• **spec.selector** 字段现在变为必需字段，并且在 StatefulSet 创建之后不可变更； 可以使用现有的模板的标签作为选择算符以实现无缝迁移。 • **spec.updateStrategy.type** 的默认值变为 **RollingUpdate**（**apps/v1beta1** API 版本中的默认值是 **OnDelete**）。
ReplicaSet	extensions/v1beta1 apps/v1beta1 apps/v1beta2	apps/v1 （该API从社区v1.9版本开始可用）	**spec.selector** 现在变成必需字段，并且在对象创建之后不可变更； 可以将现有模板的标签作为选择算符以实现无缝迁移。
PodSecurityPolicy	extensions/v1beta1	policy/v1beta1 （该API从社区v1.10版本开始可用）	**policy/v1beta1** API 版本的 PodSecurityPolicy 会在 v1.25 版本中移除。

升级备份说明

目前集群升级备份方式如下：

备份方式	备份对象	备份方式	备份时间	回滚时间	说明
etcd数据备份	etcd数据	升级流程中自动备份	1-5min	2h	必选备份，升级过程中自动进行，用户无需关注
CBR整机备份	Master节点磁盘，包括组件镜像、配置、日志以及etcd数据	通过页面一键备份（手动触发）	20min-2h（受当前区域云备份任务排队情况影响）	20min	该功能逐步由EVS快照备份替代
EVS快照备份	Master节点磁盘，包括组件镜像、配置、日志以及etcd数据	通过页面一键备份（手动触发）	1-5min	20min	该功能上线中 对于已上线的区域，EVS快照备份将替代CBR整机备份

2.5.3 升级后验证

2.5.3.1 集群状态检查

检查项内容

集群升级后，需要检查集群状态是否为“运行中”状态。

检查步骤

系统会自动为您检查集群状态是否正常，您可以根据诊断结果前往集群列表页面进行确认。

解决方案

当集群状态异常时，请联系技术支持人员。

2.5.3.2 节点状态检查

检查项内容

集群升级后，需要检查节点状态是否为“运行中”状态。

检查步骤

系统会自动为您检查集群内节点的状态，您可以根据诊断结果前往节点列表页面进行确认。

解决方案

集群节点异常时，建议您通过**重置节点**来解决，若无法解决，请联系技术支持人员。

2.5.3.3 跳过节点检查

检查项内容

集群升级后，需要检测集群内是否有跳过升级的节点，这些节点可能会影响正常使用。

检查步骤

系统会为您检查集群内是否存在跳过升级的节点，您可以根据诊断结果前往节点列表页进行确认。跳过的节点含有标签upgrade.cce.io/skipped=true。

解决方案

对于升级详情页面中跳过的节点，请在升级完毕后**重置节点**。

说明

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

2.5.3.4 业务检查

检查项内容

集群升级完毕，由用户验证当前集群正在运行的业务是否正常。

检查步骤

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

常见的业务确认方式有：

- 业务界面可用
- 监控平台无异常告警与事件
- 关键应用进程无错误日志

- API拨测正常等

解决方案

若集群升级后您的在线业务有异常，请联系技术支持人员。

2.5.3.5 新建节点检查

检查内容

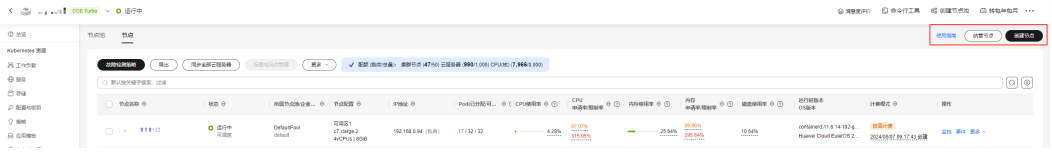
检查集群是否可以正常创建节点。

检查步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在导航栏中选择“节点管理”，并切换至“节点”页签，单击“创建节点”。节点配置详情请参见**创建节点**。

图 2-31 创建节点



----结束

解决方案

若集群升级后您的集群无法创建节点，请联系技术支持人员。

2.5.3.6 新建 Pod 检查

检查内容

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

检查步骤

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

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在导航栏中选择“工作负载”，单击右上角“创建工作负载”或“YAML创建”。创建DaemonSet的操作步骤详情请参见**创建守护进程集（DaemonSet）**。

图 2-32 创建守护进程集

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

说明

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

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: post-upgrade-check
  namespace: default
spec:
  selector:
    matchLabels:
      app: post-upgrade-check
      version: v1
  template:
    metadata:
      labels:
        app: post-upgrade-check
        version: v1
    spec:
      containers:
        - name: container-1
          image: nginx:perl
          imagePullPolicy: IfNotPresent
          resources:
            requests:
              cpu: 10m
              memory: 10Mi
            limits:
              cpu: 100m
              memory: 50Mi
```

步骤3 负载创建完毕后请检查该工作负载的Pod状态是否正常。

步骤4 检查完毕后，在导航栏中选择“工作负载”并切换至“守护进程集”，选择post-upgrade-check工作负载并单击“更多>删除”，删除该测试用工作负载。

----结束

解决方案

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

2.5.4 集群跨版本业务迁移

适用场景

本章介绍在CCE中如何将老版本集群的业务迁移到新版本集群。

适用于需要大幅度跨版本集群升级（如1.19.* 升级到1.28.*版本）的需求，可以接受新建新版本集群而进行业务迁移的升级方式。

前提条件

表 2-29 迁移前 Checklist

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

操作步骤

步骤1 创建新集群

创建与老版本集群同规格同配置的集群，创建方法请参见**购买Standard/Turbo集群**。

步骤2 添加节点

添加同规格节点，并且在节点上配置之前的手动配置项，创建方法请参见**创建节点**。

步骤3 创建存储

在新集群中使用已有存储创建PVC，PVC名称不变，方法请参见**通过静态存储卷使用已有对象存储**或**通过静态存储卷使用已有极速文件存储**。

说明

切流方案仅支持OBS、SFS Turbo等共享存储。非共享存储切流需要将老集群内的工作负载暂停，将会导致服务不可用。

步骤4 创建工作负载

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

步骤5 重新挂载存储

在工作负载中重新挂载已有的存储，方法请参见**通过静态存储卷使用已有对象存储**或**通过静态存储卷使用已有极速文件存储**。

步骤6 创建服务

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

步骤7 调测功能

全部创建完成后，请自行调测业务，调测无问题后切换流量。

步骤8 老集群退订或删除

新集群全部功能ready，退订或者删除老集群，删除集群方法请参见**删除集群**。

----结束

2.5.5 升级前检查异常问题排查

2.5.5.1 升级前检查项

集群升级前，系统将自动进行全面的升级前检查，当集群不满足升级前检查条件时将无法继续升级。为了能够更好地避免升级风险，本文提供全量的升级前检查问题及解决方案，帮助您对可能存在的升级故障进行预处理。

表 2-30 检查项列表

序号	检查项名称	检查项说明
1	**节点限制检查异常处理**	• 检查节点是否可用 • 检查节点操作系统是否支持升级 • 检查节点是否含有非预期的节点池标签 • 检查K8s节点名称是否与云服务器保持一致
2	**升级管控检查异常处理**	检查集群是否处于升级管控中。
3	**插件检查异常处理**	• 检查插件状态是否正常 • 检查插件是否支持目标版本
4	**Helm模板检查异常处理**	检查当前HelmRelease记录中是否含有目标集群版本不支持的K8s废弃API，可能导致升级后helm模板不可用。
5	**Master节点SSH连通性检查异常处理**	该检查通过尝试建立SSH连接，检查CCE是否能通过SSH方式连接至您的Master节点。
6	**安全组检查异常处理**	检查Node节点安全组规则中，协议端口为ICMP:全部，源地址为Master节点安全组的规则是否被删除。
7	**残留待迁移节点检查异常处理**	检查节点是否需要迁移。
8	**K8s废弃资源检查异常处理**	检查集群是否存在对应版本已经废弃的资源。
9	**兼容性风险检查异常处理**	请您阅读版本兼容性差异，并确认不受影响。补丁升级不涉及版本兼容性差异。

序号	检查项名称	检查项说明
10	节点CCE Agent版本检查异常处理	检测当前节点的CCE包管理组件cce-agent是否为最新版本。
11	节点CPU使用率检查异常处理	检查节点CPU使用量是否超过90%。
12	CRD检查异常处理	• 检查集群关键CRD "packageversions.version.cce.io"是否被删除。 • 检查集群关键CRD "network-attachment-definitions.k8s.cni.cncf.io"是否被删除。
13	节点磁盘检查异常处理	• 检查节点关键数据盘使用量是否满足升级要求 • 检查/tmp目录是否存在500MB可用空间
14	节点DNS检查异常处理	• 检查当前节点DNS配置是否能正常解析OBS地址 • 检查当前节点是否能访问存储升级组件包的OBS地址
15	节点关键目录文件权限检查异常处理	检查CCE使用的目录/var/paas内文件的属主和属组是否都为paas。
16	节点Kubelet检查异常处理	检查节点kubelet服务是否运行正常。
17	节点内存检查异常处理	检查节点内存使用量是否超过90%。
18	节点时钟同步服务器检查异常处理	检查节点时钟同步服务器ntpd或chronyd是否运行正常。
19	节点OS检查异常处理	检查节点操作系统内核版本是否为CCE支持的版本。
20	节点CPU数量检查异常处理	检查您的集群Master节点的CPU核心数量，要求Master节点的核心数量大于2核。
21	节点Python命令检查异常处理	检查Node节点中Python命令是否可用。
22	ASM网格版本检查异常处理	• 检查集群是否使用ASM网格服务 • 检查当前ASM版本是否支持目标集群版本
23	节点Ready检查异常处理	检查集群内节点是否Ready。
24	节点journald检查异常处理	检查节点上的journald状态是否正常。
25	节点干扰ContainerdSock检查异常处理	检查节点上是否存在干扰的Containerd.Sock文件。该文件影响Euler操作系统下的容器运行时启动。

序号	检查项名称	检查项说明
26	内部错误异常处理	该检查非常规检查项，表示升级前检查流程中出现了内部错误。
27	节点挂载点检查异常处理	检查节点上是否存在不可访问的挂载点。
28	K8s节点污点检查异常处理	检查节点上是否存在集群升级需要使用到的污点。
29	everest插件版本限制检查异常处理	检查集群当前everest插件版本是否存在兼容性限制。
30	cce-hpa-controller插件限制检查异常处理	检查cce-controller-hpa插件的目标版本是否存在兼容性限制。
31	增强型CPU管理策略检查异常处理	检查当前集群版本和要升级的目标版本是否支持**增强型CPU管理策略**。
32	用户节点组件健康检查异常处理	检查用户节点的容器运行时组件和网络组件等是否健康。
33	控制节点组件健康检查异常处理	检查集群中的Kubernetes组件、容器运行时组件、网络组件等组件，要求在升级前以上组件运行正常。
34	K8s组件内存资源限制检查异常处理	检查K8s组件例如etcd、kube-controller-manager等组件是否资源超出限制。
35	K8s废弃API检查异常处理	系统会扫描过去一天的审计日志，检查用户是否调用目标K8s版本已废弃的API。 **说明** 由于审计日志的时间范围有限，该检查项仅作为辅助手段，集群中可能已使用即将废弃的API，但未在过去一天的审计日志中体现，请您充分排查。
36	节点NetworkManager检查异常处理	检查节点上的NetworkManager状态是否正常。
37	节点ID文件检查异常处理	检查节点的ID文件内容是否符合格式。
38	节点配置一致性检查异常处理	在升级集群版本至v1.19及以上版本时，将对您的节点上的Kubenertes组件的配置进行检查，检查您是否后台修改过配置文件。
39	节点配置文件检查异常处理	检查节点上关键组件的配置文件是否存在。
40	CoreDNS配置一致性检查异常处理	检查当前CoreDNS关键配置Corefile是否同Helm Release记录存在差异，差异的部分可能在插件升级时被覆盖，**影响集群内部域名解析**。

序号	检查项名称	检查项说明
41	节点Sudo检查异常处理	检查当前节点sudo命令，sudo相关文件是否正常。
42	节点关键命令检查异常处理	检查节点升级依赖的一些关键命令是否能正常执行。
43	节点sock文件挂载检查异常处理	检查节点上的Pod是否直接挂载docker/containerd.sock文件。升级过程中Docker/Containerd将会重启，宿主机sock文件发生变化，但是容器内的sock文件不会随之变化，二者不匹配，导致您的业务无法访问Docker/Containerd。Pod重建后sock文件重新挂载，可恢复正常。
44	HTTPS类型负载均衡证书一致性检查异常处理	检查HTTPS类型负载均衡所使用的证书，是否在ELB服务侧被修改。
45	节点挂载检查异常处理	检查节点上默认挂载目录及软链接是否被手动挂载或修改。
46	节点paas用户登录权限检查异常处理	检查paas用户是否有登录权限。
47	ELB IPv4私网地址检查异常处理	检查集群内负载均衡类型的Service所关联的ELB实例是否包含IPv4私网IP。
48	检查历史升级记录是否满足升级条件	检查集群的历史升级记录，要求您的集群原始版本满足升级到目标集群版本的条件。
49	检查集群管理平面网段是否与主干配置一致	检查集群管理平面网段是否与主干配置一致。
50	GPU插件检查异常处理	检查到本次升级涉及GPU插件，可能影响新建GPU节点时GPU驱动的安装。
51	节点系统参数检查异常处理	检查您节点上默认系统参数是否被修改。
52	残留packageversion检查异常处理	检查当前集群中是否存在残留的packageversion。
53	节点命令行检查异常处理	检查节点中是否存在升级所必须的命令。
54	节点交换区检查异常处理	检查集群节点上是否开启交换区。
55	nginx-ingress插件升级检查异常处理	检查nginx-ingress插件升级路径是否涉及兼容问题。
56	ELB监听器访问控制配置项检查异常处理	若有配置访问控制则检查相关配置项是否正确。

序号	检查项名称	检查项说明
57	Master节点规格检查异常处理	检查本次升级集群的Master节点规格与实际的Master节点规格是否一致。
58	Master节点子网配额检查异常处理	检查本次升级集群子网剩余可用IP数量是否支持滚动升级。
59	节点运行时检查异常处理	该告警通常发生在低版本集群升级到v1.27及以上集群。CCE不建议您在1.27以上版本集群中继续使用docker，并计划在未来移除对docker的支持。
60	节点池运行时检查异常处理	该告警通常发生在低版本集群升级到v1.27及以上集群。CCE不建议您在1.27以上版本集群中继续使用docker，并计划在未来移除对docker的支持。
61	检查节点镜像数量异常处理	检查到您的节点上镜像数量过多（>1000个），可能导致docker启动过慢，影响docker标准输出，影响nginx等功能的正常使用。
62	OpenKruise插件兼容性检查异常处理	检查集群升级时，OpenKruise插件是否存在兼容性问题。
63	Secret落盘加密特性兼容性检查异常处理	检查本次升级的目标版本是否支持Secret落盘加密特性，若不支持则不允许开启Secret落盘加密特性的集群升级至该版本。
64	Ubuntu内核与GPU驱动兼容性提醒	检查到集群中同时使用GPU插件和Ubuntu节点，提醒客户存在可能的兼容性问题。当Ubuntu内核版本在5.15.0-113-generic上时，GPU插件必须使用535.161.08及以上的驱动版本。
65	排水任务检查异常处理	检查到集群中存在未完成的排水任务，此时升级可能会导致升级完成后触发排水动作，将运行中的Pod进行驱逐。
66	节点镜像层数量异常检查	检查到您的节点上镜像层数量过多（>5000层），可能导致docker/containerd启动过慢，影响docker/containerd标准输出。
67	检查集群是否满足滚动升级条件	检查到您的集群暂时不满足滚动升级条件。
68	轮转证书文件数量检查	检查您节点上的证书数量过多（>1000），由于升级过程中会批量处理证书文件，证书文件过多可能导致节点升级过慢，节点上Pod被驱逐等。

2.5.5.2 节点限制检查异常处理

检查项内容

当前检查项包括以下内容：

- 检查节点是否可用
- 检查节点操作系统是否支持升级
- 检查节点是否含有非预期的节点池标签
- 检查K8s节点名称是否与云服务器保持一致

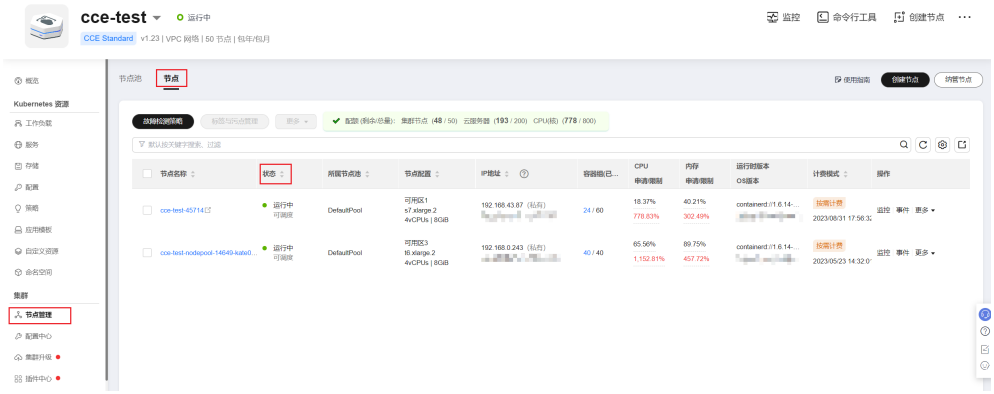
解决方案

1. **检查到节点状态异常，请优先恢复**

   若检查发现节点不可用，请登录CCE控制台，单击集群名称进入集群控制台，前往“节点管理”页面并切换至“节点”页签查看节点状态，请确保节点处于“运行中”状态。节点处于“安装中”、“删除中”状态时，均不支持升级。

   若节点状态异常，请参考 **集群可用，但节点状态为“不可用”**修复节点后，重试检查任务。

   图 2-33 查看节点状态

2. **检查到节点操作系统不支持升级**

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

   表 2-31 节点 OS 支持列表

操作系统	限制
EulerOS 2.x	目标版本为v1.27以下时，无限制 目标版本为v1.27及以上时，仅支持EulerOS 2.9、EulerOS 2.10
CentOS 7.x	无限制
Ubuntu	目标版本为v1.27及以上时，仅支持Ubuntu 22.04。
Huawei Cloud EulerOS	无限制

3. **检查到节点属于默认节点池，但是含有普通节点池标签，将影响升级流程**

   由节点池迁移至默认节点池的节点，"cce.cloud.com/cce-nodepool"该标签影响集群升级。请确认该节点上的负载调度是否依赖该标签：

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

4. **检查到节点含有CNIProblem污点，请优先恢复**

   检查到节点含有key为node.cloudprovider.kubernetes.io/cni-problem，效果为不可调度（NoSchedule）的污点。该污点由NPD插件检查添加，建议您优先升级NPD插件至最新版本，再重新检查，若仍然有问题，请联系支持人员。

5. **检查到节点对应的k8s node资源不存在，该节点可能正在删除中，请稍后重试**

   等待节点删除完全后，在升级前检查界面重新检查。

6. **检查控制节点操作系统为EulerOS 2.5，不支持升级到v1.27.5-r0版本**

   您可选择升级至1.25或1.28。如果您选择升级至1.28版本，升级过程中EulerOS 2.5将会被滚动升级成HCE 2.0。如果您有其他需求，请提交工单联系技术支持人员。

2.5.5.3 升级管控检查异常处理

检查项内容

检查集群是否处于升级管控中。

解决方案

CCE基于以下几点原因，可能会暂时限制该集群的升级功能：

- 基于用户提供的信息，该集群被识别为核心重点保障的生产集群。
- 正在或即将进行其他运维任务，例如Master节点3AZ改造等。

请根据界面日志联系技术支持人员了解限制原因并申请解除升级限制。

2.5.5.4 插件检查异常处理

检查项内容

当前检查项包括以下内容：

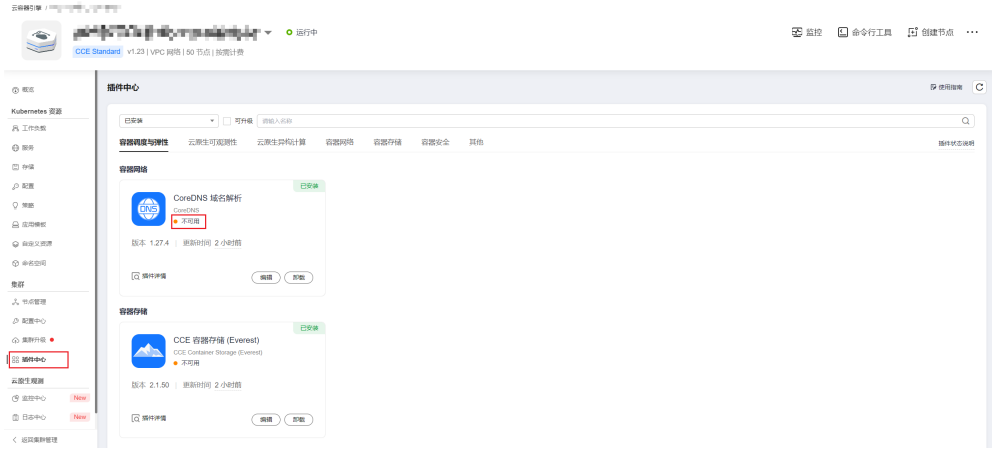
- 检查插件状态是否正常
- 检查插件是否支持目标版本

解决方案

- **问题场景一：插件状态异常**

  请登录CCE控制台，单击集群名称进入集群控制台，前往“插件中心”处查看并处理处于异常状态的插件。

图 2-34 查看插件状态



- **问题场景二：集群升级的目标版本已经不支持该插件**

  升级前检查出现以下报错：

  ```
  addon [***] does not support cluster target version, check and try again
  ```

  请您登录CCE控制台，单击集群名称进入集群控制台，在“插件中心”处进行手动卸载，具体插件支持版本以及替换方案可查看**帮助文档**。

- **问题场景三：插件配置不满足升级条件，请在插件升级页面升级插件之后重试**

  升级前检查出现以下报错：

  ```
  please upgrade addon [ ] in the page of addon managecheck and try again
  ```

  请您登录CCE控制台，在“插件中心”处手动升级插件。

2.5.5.5 Helm 模板检查异常处理

检查项内容

检查当前HelmRelease记录中是否含有目标集群版本不支持的K8s废弃API，可能导致升级后helm模板不可用。

解决方案

将HelmRelease记录中K8s废弃API转换为源版本和目标版本均兼容的API。

说明

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

2.5.5.6 Master 节点 SSH 连通性检查异常处理

检查项内容

该检查通过尝试建立SSH连接，检查CCE是否能通过SSH方式连接至您的Master节点。

解决方案

SSH连通性检查可能有较低概率因为网络波动检查失败，请您优先重试升级前检查；

若重试检查仍无法通过检查，请您提交工单，联系技术支持人员排查。

2.5.5.7 节点池检查异常处理

检查项内容

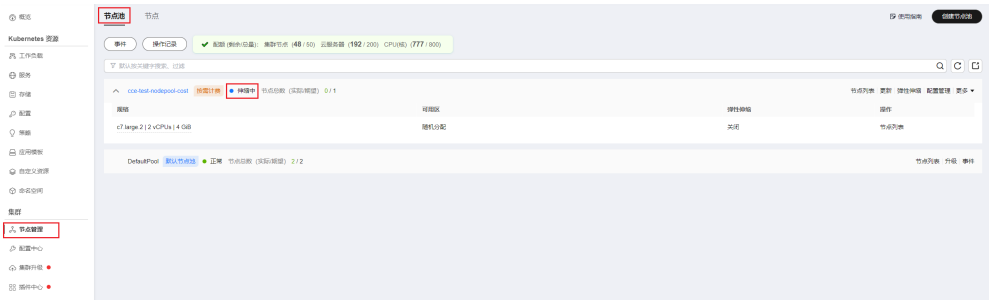
- 检查节点池状态是否正常。
- 检查升级后节点池操作系统或容器运行时是否支持。

解决方案

- **问题场景：节点池状态异常**

  请登录CCE控制台，单击集群名称进入集群控制台，前往“节点管理”页面查看问题节点池状态。若该节点池状态处于伸缩中，请等待节点池伸缩完毕。

  图 2-35 查看节点池状态

- **问题场景：节点池操作系统不支持**

  由于不同版本之间的运行时和OS存在差异，该异常通常发生在低版本集群升级到1.27及以上集群。当前CCE集群版本和OS的配套关系请参见**节点操作系统说明**。

  请登录CCE控制台，单击集群名称进入集群控制台，前往“节点管理“页面查看问题节点池，并单击节点池的“更新”。根据升级前检查的提示信息，修改支持的操作系统，并单击“确定”。

  如果节点池下存在节点，可以单击节点操作列的“更多 > 同步”选项，同步已有节点的操作系统，详情请参见**同步节点池**。

2.5.5.8 安全组检查异常处理

检查项内容

检查Node节点安全组规则中，协议端口为ICMP:全部，源地址为Master节点安全组的规则是否被删除。

说明

仅VPC网络模型的集群执行该检查项，非VPC网络模型的集群将跳过该检查项。

解决方案

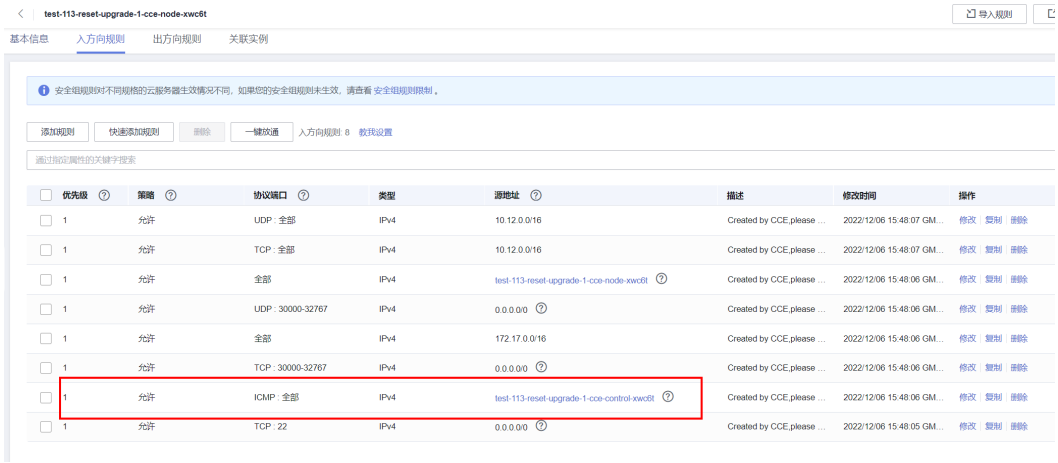
请登录VPC控制台，前往“访问控制 > 安全组”，在搜索框内输入集群名称，此时预期过滤出两个安全组：

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

图 2-36 查看集群安全组

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

图 2-37 Node 节点安全组



若不含有该规则请为Node安全组添加该放通规则，协议端口选择“基本协议/ICMP”，端口号为“全部”，源地址选择“安全组”并设置为Master安全组，描述信息为"Created by CCE,please don't modify! Used by the master node to access the worker node."。

图 2-38 对 Master 安全组放通 ICMP 协议

2.5.5.9 残留待迁移节点检查异常处理

检查项内容

检查节点是否需要迁移。

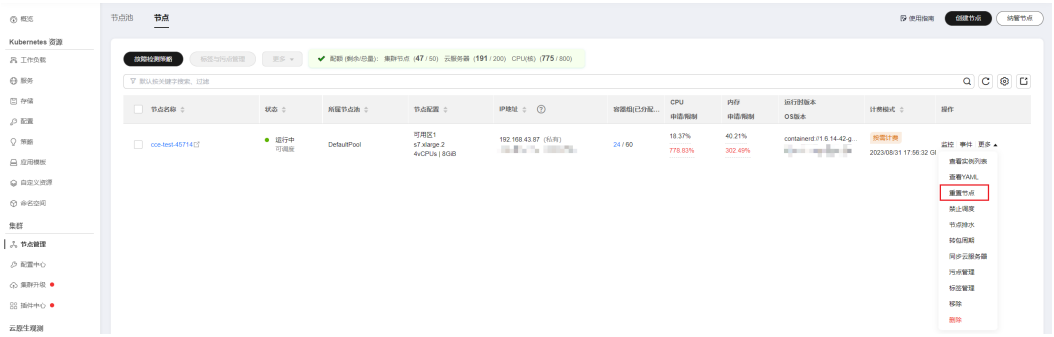
解决方案

该问题由于节点拉包组件异常或节点由比较老的版本升级而来，导致节点上缺少关键的系统组件导致。

解决方案一

请登录CCE控制台，单击集群名称进入集群控制台，前往“节点管理”页面，单击对应节点的“更多 > 重置节点”，详情请参见**重置节点**。节点重置完毕后，重试检查任务。

图 2-39 重置节点



说明

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

解决方案二

新建节点后，删除问题节点。

2.5.5.10 K8s 废弃资源检查异常处理

检查项内容

检查集群是否存在对应版本已经废弃的资源。

解决方案

- **问题场景一： 1.25及以上集群中的service存在废弃的annotation：tolerate-unready-endpoints**

  报错日志信息如下：

  ```
  some check failed in cluster upgrade: this cluster has deprecated service list: map[***] with deprecated annotation list [tolerate-unready-endpoints]
  ```


检查日志信息中所给出的service是否存在**"tolerate-unready-endpoints"**的annotation，如果存在则将其去掉，并在对应的service的spec中添加下列字段来替代该annotation：

```
publishNotReadyAddresses: true
```

- **问题场景二： 1.27及以上集群中的service存在废弃的annotation：service.kubernetes.io/topology-aware-hints**

  报错日志信息如下：

  ```
  some check failed in cluster upgrade: this cluster has deprecated service list: map[***] with deprecated annotation list [service.kubernetes.io/topology-aware-hints]
  ```

  检查日志信息中所给出的service是否存在**"service.kubernetes.io/topology-aware-hints"**的annotation，如果存在则将其去掉，并在对应的service中用**"service.kubernetes.io/topology-mode"**的annotation进行替换。

2.5.5.11 兼容性风险检查异常处理

检查项内容

请您阅读版本兼容性差异，并确认不受影响。补丁升级不涉及版本兼容性差异。

版本兼容性差异

版本升级路径	版本差异	建议自检措施
v1.23/ v1.25 升级至 v1.27	容器运行时Docker不再被推荐使用，建议您使用Containerd进行替换，详情请参见**容器引擎说明**。	已纳入升级前检查。
v1.21/ v1.19 升级至 v1.23	社区较老版本的Nginx Ingress Controller来说（社区版本v0.49及以下，对应CCE插件版本v1.x.x），在创建Ingress时没有指定Ingress类别为nginx，即annotations中未添加kubernetes.io/ingress.class: nginx的情况，也可以被Nginx Ingress Controller纳管。但对于较新版本的Nginx Ingress Controller来说（社区版本v1.0.0及以上，对应CCE插件版本2.x.x），如果在创建Ingress时没有显示指定Ingress类别为nginx，该资源将被Nginx Ingress Controller忽略，Ingress规则失效，导致服务中断。	已纳入升级前检查，也可参照**nginx-ingress插件升级检查**进行自检。

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

版本升级路径	版本差异	建议自检措施

	CCE的v1.15版本集群及v1.19版本集群将docker的存储驱动文件系统由 xfs切换成ext4，可能会导致升级后的java应用Pod内的import包顺序异常，继而导致Pod异常。	升级前查看节点上docker配置文件/etc/docker/daemon.json。检查dm.fs配置项是否为xfs。
		• 若为ext4或存储驱动为overlay则不涉及。
		• 若为xfs则建议您在新版本集群预先部署应用，以测试应用与新版本集群是否兼容。
		{
		"storage-driver": "devicemapper",
		"storage-opts": [
		"dm.thinpooldev=/dev/mapper/vgpaas-thinpool",
		"dm.use_deferred_removal=true",
		"dm.fs=xfs",
		"dm.use_deferred_deletion=true"
		]
		}

	CCE的v1.19及以上版本的kube-apiserver要求客户侧webhook server的证书必须配置Subject Alternative Names (SAN)字段。否则升级后kube-apiserver调用webhook server失败，容器无法正常启动。 根因：Go语言v1.15版本废弃了X.509 **CommonName**，CCE的v1.19版本的kube-apiserver编译的版本为v1.15。CommonName字段作为hostname处理，最终导致认证失败。	升级前检查您自建webhook server的证书是否配置了SAN字段。 • 若无自建webhook server则不涉及。 • 若未配置，建议您配置使用SAN字段指定证书支持的IP及域名。 **须知** 为减弱此版本差异对集群升级的影响，v1.15升级至v1.19时，CCE会进行特殊处理，仍然会兼容支持证书不带SAN。但后续升级不再特殊处理，请尽快整改证书，以避免影响后续升级。
	v1.17.17版本及以后的集群CCE自动给用户创建了PSP规则，限制了不安全配置的Pod的创建，如securityContext配置了sysctl的net.core.somaxconn的Pod。	升级后请参考资料按需开放非安全系统配置，具体请参见**PodSecurityPolicy配置**。
	1.15版本集群原地升级，如果业务中存在initContainer或使用Istio的场景，则需要注意以下约束： 1.16及以上的kubelet统计**QosClass**和之前版本存在差异，1.15及以下版本仅统计spec.containers下的容器，1.16及以上的kubelet版本会同时统计spec.containers和spec.initContainers下的容器，升级前后会造成Pod的QosClass变化，从而造成Pod中容器重启。	建议参考**表2-23**在升级前修改业务容器的QosClass规避该问题。

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

表 2-32 1.15 版本升级前后 QosClass 变化

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

2.5.5.12 节点 CCE Agent 版本检查异常处理

检查项内容

检测当前节点的CCE包管理组件cce-agent是否为最新版本。

解决方案

- **问题场景一： 错误信息为“you cce-agent no update, please restart it”。**

该问题为cce-agent无需更新，但是没有重启，需要登录节点手动重启cce-agent。

解决方式：登录节点执行：

```
systemctl restart cce-agent
```

执行完毕后，重新执行升级检查。

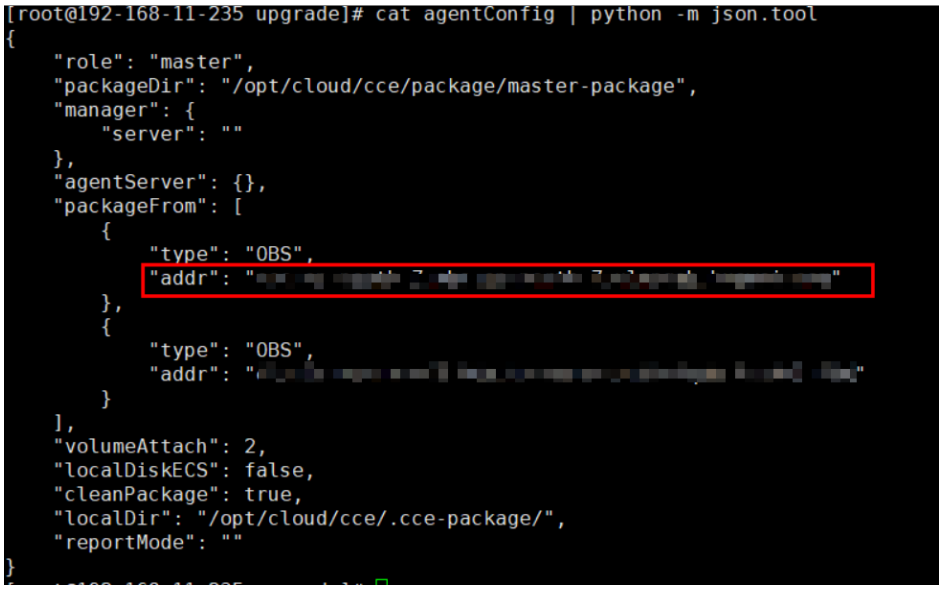
- **问题场景二： 错误信息为“your cce-agent is not the latest version”。**

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

  解决方式：

  a. 登录异常节点执行以下命令，获取有效的OBS地址，如图中addr地址为正确的OBS地址。

  ```
  cat /home/paas/upgrade/agentConfig | python -m json.tool
  ```

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

  c. 通过以下命令下载最新的二进制文件。

  - x86系统

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

  - ARM系统

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

  d. 替换原有的cce-agent二进制文件。

  - x86系统

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

  - ARM系统

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

  e. 重启cce-agent服务。

  ```
  systemctl restart cce-agent
  ```


若您对上述执行过程有疑问，请联系技术支持人员。

2.5.5.13 节点 CPU 使用率检查异常处理

检查项内容

检查节点CPU使用量是否超过90%。

解决方案

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

2.5.5.14 CRD 检查异常处理

检查项内容

当前检查项包括以下内容：

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

解决方案

如出现该检查项异常，请联系技术支持人员。

2.5.5.15 节点磁盘检查异常处理

检查项内容

当前检查项包括以下内容：

- 检查节点关键数据盘使用量是否满足升级要求
- 检查/tmp目录是否存在500MB可用空间

解决方案

节点升级过程中需要使用磁盘存储升级组件包，使用/tmp目录存储临时文件。

- **问题场景一：Master节点磁盘使用量不满足升级要求**

  请联系技术支持人员排查处理。
- **问题场景二：用户节点磁盘使用量不满足升级要求**

  请执行以下检查命令，检查当前各关键磁盘的空间使用情况，删除整理确保各可用空间满足要求后，重试检查。
  - docker容器运行时磁盘分区（可用空间需满足1G）
    ```
    df -h /var/lib/docker
    ```
  - contianerd容器运行时磁盘分区（可用空间需满足1G）
    ```
    df -h /var/lib/containerd
    ```
  - kubelet磁盘分区（可用空间需满足1G）

```
df -h /mnt/paas/kubernetes/kubelet
```

– 系统盘（可用空间需满足2G）

```
df -h /
```

- **问题场景三：用户节点/tmp目录空间不足**

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

```
df -h /tmp
```

2.5.5.16 节点 DNS 检查异常处理

检查项内容

当前检查项包括以下内容：

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

解决方案

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

2.5.5.17 节点关键目录文件权限检查异常处理

检查项内容

检查CCE使用的目录/var/paas内文件的属主和属组是否都为paas。

解决方案

- **问题场景一： 错误信息为“xx file permission has been changed!”。**

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

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

  请您参考下述命令排查当前业务Pod中是否将文件数据存储在/var/paas路径下，修改避免使用该路径，并移除该路径下的异常文件后重试检查，通过后可继续升级。

```
find /var/paas -not \( -user paas -o -user root \) -print
```

- **问题场景二： 错误信息为“user paas must have at least read and execute permissions on the root directory”。**

  解决方案：节点根目录权限被修改导致paas用户没有根目录的读权限，这会导致升级时组件重启失败，建议将根目录权限修正为默认权限555。

2.5.5.18 节点 Kubelet 检查异常处理

检查项内容

检查节点kubelet服务是否运行正常。

解决方案

- **问题场景一：kubelet状态异常**

  kubelet异常时，节点显示不可用，请参考**集群可用，但节点状态为“不可用”**修复节点后，重试检查任务。

- **问题场景二：cce-pause版本异常**

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

2.5.5.19 节点内存检查异常处理

检查项内容

检查节点内存使用量是否超过90%。

解决方案

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

2.5.5.20 节点时钟同步服务器检查异常处理

检查项内容

检查节点时钟同步服务器ntpd或chronyd是否运行正常。

解决方案

- **问题场景一：ntpd运行异常**

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

  以下为正常回显：

  图 2-40 ntpd 运行状态

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

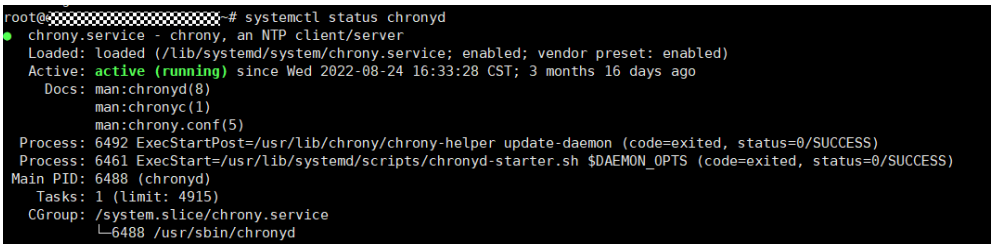
  若重启ntpd服务无法解决该问题，请联系技术支持人员。

- **问题场景二：chronyd运行异常**

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

  以下为正常回显：

图 2-41 chronyd 运行状态



若重启chronyd服务无法解决该问题，请联系技术支持人员。

2.5.5.21 节点 OS 检查异常处理

检查项内容

检查节点操作系统内核版本是否为CCE支持的版本。

解决方案

- **问题场景一：节点镜像非CCE标准镜像**

  CCE节点运行依赖创建时的初始标准内核版本，CCE基于该内核版本做了全面的兼容性测试，非标准的内核版本可能在节点升级中因兼容性问题导致节点升级失败，详情请参见**高危操作及解决方案**。

  当前CCE不建议该类节点进行升级，建议您在升级前**重置节点**至标准内核版本。

- **问题场景二：特殊版本镜像存在缺陷**

  检查到本次升级涉及1.17 欧拉2.8 Arm镜像，该版本镜像存在缺陷，其上docker重启后将影响"docker exec"命令，升级集群版本时将触发docker版本更新，触发docker重启，因此存在建议：

  a. 建议您提前排空、隔离该节点后进行集群升级。

  b. 建议您升级至1.19及更高版本后，通过重置节点操作更换更高版本镜像，例如欧拉2.9镜像。

2.5.5.22 节点 CPU 数量检查异常处理

检查项内容

检查您的集群Master节点的CPU核心数量，要求Master节点的核心数量大于2核。

解决方案

当前您的Master节点cpu数量为2，可能会导致集群升级失败；

请联系技术支持人员，将该集群Master节点扩容至4核及以上。

2.5.5.23 节点 Python 命令检查异常处理

检查项内容

检查Node节点中Python命令是否可用。

检查方式

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

如果回显值不为0证明检查失败。

解决方案

可优先重置节点或手动安装Python之后再进行升级。

2.5.5.24 ASM 网格版本检查异常处理

检查项内容

当前检查项包括以下内容：

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

解决方案

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

  表 2-33 ASM 网格版本与集群版本适配规则

ASM网格版本	集群版本
1.3	v1.13、v1.15、v1.17、v1.19
1.6	v1.15、v1.17
1.8	v1.15、v1.17、v1.19、v1.21
1.13	v1.21、v1.23
1.15	v1.21、v1.23、v1.25、v1.27
1.18	v1.25、v1.27、v1.28、v1.29、v1.30

- 若不需要使用ASM网格，可删除ASM网格后再进行升级，升级后集群不能绑定与表中不匹配的ASM网格版本。例如，使用v1.21版本集群与1.8版本ASM网格，若要升级至v1.25版本集群时，请先升级ASM网格至1.15版本后再进行v1.25版本集群升级。

2.5.5.25 节点 Ready 检查异常处理

检查项内容

检查集群内节点是否Ready。

解决方案

- **问题场景一：节点状态显示不可用**

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

- **问题场景二：节点状态与实际不符**

  节点状态与实际不符可能存在两种情况：

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

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

2.5.5.26 节点 journald 检查异常处理

检查项内容

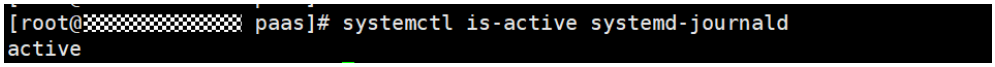
检查节点上的journald状态是否正常。

解决方案

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

以下为正常回显：

图 2-42 journald 服务运行状态



若重启journald服务无法解决该问题，请联系技术支持人员。

2.5.5.27 节点干扰 ContainerdSock 检查异常处理

检查项内容

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

解决方案

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

步骤1 登录相关节点。

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

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

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

----结束

2.5.5.28 内部错误异常处理

检查项内容

该检查非常规检查项，表示升级前检查流程中出现了内部错误。

解决方案

该问题出现后，请您优先重试升级前检查；

若重试升级前检查仍失败，请您提交工单，联系技术支持人员。

2.5.5.29 节点挂载点检查异常处理

检查项内容

检查节点上是否存在不可访问的挂载点。

解决方案

问题场景：节点上存在不可访问的挂载点

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

步骤1 登录节点。

步骤2 在节点上新建一个脚本文件（例如/tmp/check_hang_mount.sh），脚本文件内容如下：

```
for mount_path in `cat /proc/self/mountinfo | awk '{print $5}' | grep -v netns`
do
   timeout 10 sh -c "cd $mount_path"
   if [ $? == 124 ];then
      echo "$mount_path hang mount"
   fi
done
```

步骤3 执行保存好的脚本，查看输出。

```
[root@test-02-upgrade-v115-v119-vpc-06491 ~]# bash test.sh
/root/foo hang mount
/root/bar hang mount
[root@test-02-upgrade-v115-v119-vpc-06491 ~]#
```

如上图所示，则为/root/foo和/root/bar这两个文件夹的挂载点存在问题。

步骤4 执行以下命令，查看卡死的挂载点。

```
mount -n | grep /root/foo
```

```
/root/bar hang mount
[root@test-02-upgrade-v115-v119-vpc-06491 ~]# mount -n | grep /root/foo
localhost:/tmp/nfs on /root/foo type nfs (rw,relatime,vers=3,rsize=524288,wsize=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,mountaddr=127.0.0.1,mountvers=3,mountport=20048,mountproto=udp,local_lock=none,addr=127.0.0.1)
[root@test-02-upgrade-v115-v119-vpc-06491 ~]#
```

一般来说，此类卡死的挂载点表示已经没有业务使用，请您确认该挂载点确实废弃之后执行以下命令卸载掉对应卡死的挂载点，然后重新执行上述脚本。


```
umount -l -f localhost:/tmp/nfs
```

```
[root@test-02-upgrade-v115-v119-vpc-06491 ~]# umount -l -f localhost:/tmp/nfs
[root@test-02-upgrade-v115-v119-vpc-06491 ~]# bash test.sh
[root@test-02-upgrade-v115-v119-vpc-06491 ~]#
[root@test-02-upgrade-v115-v119-vpc-06491 ~]#
[root@test-02-upgrade-v115-v119-vpc-06491 ~]#
```

执行通过之后在升级前检查界面重新检查即可。

----结束

2.5.5.30 K8s 节点污点检查异常处理

检查项内容

检查节点上是否存在集群升级需要使用到的污点。

表 2-34 检查污点列表

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

解决方案

问题场景一：该节点为集群升级过程中跳过的节点。

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

步骤2 查看对应节点kubelet版本，以下为正常回显：

图 2-43 kubelet 版本

```
[root@10-3-120-59 paas]# kubectl get node
NAME          STATUS   ROLES    AGE   VERSION
10.3.[redacted]  Ready    <none>   28h   v1.19.16-r4-CCE22.11.1
10.3.[redacted]  Ready    <none>   28h   v1.19.16-r4-CCE22.11.1
```

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

说明

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

----结束

2.5.5.31 everest 插件版本限制检查异常处理

检查项内容

检查集群当前everest插件版本是否存在兼容性限制。

表 2-35 受限的 everest 插件版本

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

解决方案

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

2.5.5.32 cce-hpa-controller 插件限制检查异常处理

检查项内容

检查cce-controller-hpa插件的目标版本是否存在兼容性限制。

解决方案

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

请您在集群中安装相应metrics插件之后重试检查

2.5.5.33 增强型 CPU 管理策略检查异常处理

检查项内容

检查当前集群版本和要升级的目标版本是否支持**增强型CPU管理策略**。

解决方案

问题场景：当前集群版本使用增强型CPU管理策略功能，要升级的目标集群版本不支持增强型CPU管理策略功能。

升级到支持增强型CPU管理策略的集群版本，支持增强型CPU管理策略的集群版本如下表所示：

表 2-36 支持增强型 CPU 管理策略的集群版本列表

集群版本	是否支持增强型CPU管理策略功能
v1.17及以下版本	不支持
v1.19	不支持
v1.21	不支持
v1.23及以上版本	支持

2.5.5.34 用户节点组件健康检查异常处理

检查项内容

检查用户节点的容器运行时组件和网络组件等是否健康。

解决方案

- 问题场景一：CNI Agent is not active

  如果您的集群版本在1.17.17以下，或者1.17.17以上且是隧道网络，请登录该节点，执行**systemctl status canal**命令查询canal服务运行状态，若回显状态异常，请执行**systemctl restart canal**命令后重新查询状态。

  如果您的集群是1.17.17以上，且是VPC网络或云原生网络2.0，请登录该节点，执行**systemctl status yangtse**命令查询yangtse服务运行状态，若回显状态异常，请执行**systemctl restart yangtse**命令后重新查询状态。

- 问题场景二：kubeproxy is not active

  请登录该节点，执行**systemctl is-active kube-proxy**命令查询kube-proxy服务运行状态。若回显状态异常，请执行**systemctl restart kube-proxy**命令后重新查询状态。

  如果上述操作无法解决，建议您进行重置节点操作，参考**重置节点**。

2.5.5.35 控制节点组件健康检查异常处理

检查项内容

检查集群中的Kubernetes组件、容器运行时组件、网络组件等组件，要求在升级前以上组件运行正常。

解决方案

请您优先重试升级前检查；

若重试检查仍失败时，请您提交工单，联系技术支持人员进行处理。

2.5.5.36 K8s 组件内存资源限制检查异常处理

检查项内容

检查K8s组件例如etcd、kube-controller-manager等组件是否资源超出限制。

解决方案

- 方案一：适当减少K8s资源。
- 方案二：扩大集群规格，详情请参见**变更集群规格**。

2.5.5.37 K8s 废弃 API 检查异常处理

检查项内容

系统会扫描过去一天的审计日志，检查用户是否调用目标K8s版本已废弃的API。

说明

由于审计日志的时间范围有限，该检查项仅作为辅助手段，集群中可能已使用即将废弃的API，但未在过去一天的审计日志中体现，请您充分排查。

解决方案

检查说明

根据检查结果，检测到您的集群通过kubectl或其他应用调用了升级目标集群版本已废弃的API，您可在升级前进行整改，否则升级到目标版本后，该API将会被kube-apiserver拦截，影响您的使用。具体每个API废弃情况可参考**废弃API说明**。

案例介绍

社区v1.22版本集群废弃了extensions/v1beta1和networking.k8s.io/v1beta1 API 版本的 Ingress ，若您从v1.19或v1.21版本的集群升级到v1.23版本，原有已创建的资源不受影响，但新建与编辑场景将会遇到v1beta1 API 版本被拦截的情况。

具体yaml配置结构变更可参考文档**通过Kubectl命令行创建ELB Ingress**。

2.5.5.38 节点 NetworkManager 检查异常处理

检查项内容

检查节点上的NetworkManager状态是否正常。

解决方案

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

如果上述操作无法解决，建议您进行重置节点操作，参考**重置节点**。如果您不想重置节点，请联系技术支持人员恢复配置文件后进行升级。

2.5.5.39 节点 ID 文件检查异常处理

检查项内容

检查节点的ID文件内容是否符合格式。

解决方案

步骤1 在CCE控制台上的“节点管理”页面，单击异常节点名称进入ECS界面。

步骤2 复制节点ID，保存到本地。

图 2-44 复制节点 ID

步骤3 登录异常节点，备份文件 。

```
cp /var/lib/cloud/data/instance-id /tmp/instance-id
cp /var/paas/conf/server.conf /tmp/server.conf
```

步骤4 登录异常节点，将获取的节点ID写入文件。

```
echo "节点ID" > /var/lib/cloud/data/instance-id
echo "节点ID" > /var/paas/conf/server.conf
```

----结束

2.5.5.40 节点配置一致性检查异常处理

检查项内容

在升级集群版本至v1.19及以上版本时，将对您的节点上的Kubenertes组件的配置进行检查，检查您是否后台修改过配置文件。

- /opt/cloud/cce/kubernetes/kubelet/kubelet
- /opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml
- /opt/cloud/cce/kubernetes/kube-proxy/kube-proxy
- /etc/containerd/default_runtime_spec.json
- /etc/sysconfig/docker
- /etc/default/docker
- /etc/docker/daemon.json

如您对这些文件的某些参数进行修改，有可能会导致集群升级失败或升级之后业务出现异常。如您确认该修改对业务无影响，可单击确认后继续进行升级操作。

说明

CCE采用标准镜像的脚本进行节点配置一致性检查，如您使用其它自定义镜像有可能导致检查失败。

当前可预期的修改将不会进行拦截，可预期修改的参数列表如下：

表 2-37 可预期修改的参数列表

组件	配置文件	参数	升级版本
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	cpuManagerPolicy	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	maxPods	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	kubeAPIQPS	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	kubeAPIBurst	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	podPidsLimit	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	topologyManagerPolicy	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	resolvConf	v1.19以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	eventRecordQPS	v1.21以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	topologyManagerScope	v1.21以上
kubelet	/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	allowedUnsafeSysctls	v1.19以上
docker	/etc/docker/daemon.json	dm.basesize	v1.19以上

解决方案

如您对这些文件的某些参数进行修改，有可能导致升级之后出现异常情况。如果您不能确认自行修改的参数是否会影响到升级，请联系技术人员确认。

2.5.5.41 节点配置文件检查异常处理

检查项内容

检查节点上关键组件的配置文件是否存在。

当前检查文件列表如下：

文件名	文件内容	备注
/opt/cloud/cce/kubernetes/kubelet/kubelet	kubelet命令行启动参数	-
/opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml	kubelet启动参数配置	-
/opt/cloud/cce/kubernetes/kube-proxy/kube-proxy	kube-proxy命令行启动参数	-
/etc/sysconfig/docker	docker配置文件	containerd运行时或Debain-Group机器不检查
/etc/default/docker	docker配置文件	containerd运行时或Centos-Group机器不检查

解决方案

建议您进行重置节点操作，参考**重置节点**。如果您不想重置节点，请联系技术支持人员恢复配置文件后进行升级。

2.5.5.42 CoreDNS 配置一致性检查异常处理

检查项内容

检查当前CoreDNS关键配置Corefile是否同Helm Release记录存在差异，差异的部分可能在插件升级时被覆盖，**影响集群内部域名解析**。

解决方案

您可在明确差异配置后，单独升级CoreDNS插件。

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

步骤2 获取当前生效的Corefile。

```
kubectl get cm -nkube-system coredns -o jsonpath='{.data.Corefile}' > corefile_now.txt
cat corefile_now.txt
```

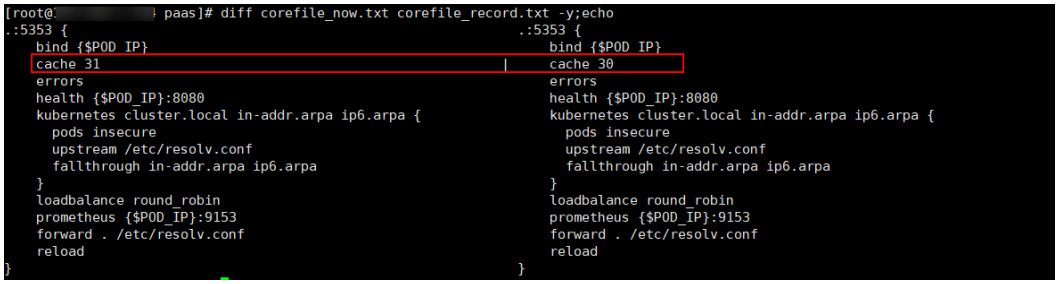
步骤3 获取Helm Release记录中的Corefile。

```
latest_release=`kubectl get secret -nkube-system -l owner=helm -l name=cceaddon-coredns --sort-by=.metadata.creationTimestamp | awk 'END{print $1}'`
kubectl get secret -nkube-system $latest_release -o jsonpath='{.data.release}' | base64 -d | base64 -d | gzip -d | python -m json.tool | python -c "
from __future__ import print_function
import json,sys,re,yaml;
manifests = json.load(sys.stdin)['manifest']
files = re.split('(?:^|\s*\n)---\s*',manifests)
for file in files:
    if 'coredns/templates/configmap.yaml' in file and 'Corefile' in file:
      corefile = yaml.safe_load(file)['data']['Corefile']
      print(corefile,end='')
      exit(0);
print('error')
exit(1);
" > corefile_record.txt
cat corefile_record.txt
```


步骤4 对比步骤2和步骤3的输出差异。

```
diff corefile_now.txt corefile_record.txt -y;
```

图 2-45 查看输出差异



步骤5 返回CCE控制台，单击集群名称进入集群控制台，前往“插件中心”，选择CoreDNS插件单击“升级”。

若要保留差异部分配置，您可采用以下方法之一：

- （推荐）将“parameterSyncStrategy”参数配置为“inherit”，差异配置将自动继承，由系统自动解析、识别与继承差异参数。
- 将“parameterSyncStrategy”参数配置为“force”，您需手动填写差异配置，详情请参考CoreDNS域名解析。

步骤6 单击“确定”，等待插件升级完毕，检查CoreDNS各实例均可用，且Corefile符合预期。

```
kubectl get cm -nkube-system coredns -o jsonpath='{.data.Corefile}'
```

步骤7 编辑CoreDNS插件配置的“parameterSyncStrategy”参数，重置为“ensureConsistent”，继续开启配置一致性校验。

同时建议您后续通过CCE插件管理的参数配置功能修改Corefile配置，避免产生差异。

----结束

2.5.5.43 节点 Sudo 检查异常处理

检查项内容

检查当前节点sudo命令，sudo相关文件是否正常。

解决方案

- 问题场景一：sudo命令执行失败

  集群原地升级过程中依赖sudo命令正常可用，请登录节点执行如下命令，排查sudo命令可用性。

  ```
  sudo echo hello
  ```

  如果sudo命令不存在，请您从其他节点复制sudo命令到该节点。

- 问题场景二：关键文件不可修改

  集群原地升级过程中会修改/etc/sudoers文件和/etc/sudoers.d/sudoerspaas文件，以获取sudo权限，更新节点上属主和属组为root的组件（例如docker、kubelet等）与相关配置文件。请登录节点执行如下命令，排查文件的可修改性。

  ```
  lsattr -l /etc/sudoers.d/sudoerspaas /etc/sudoers
  ```

  若回显中存在immutable，则说明文件被加了i锁不可修改，建议您去除i锁。

```
chattr -i /etc/sudoers.d/sudoerspaas /etc/sudoers
```

2.5.5.44 节点关键命令检查异常处理

检查项内容

检查节点升级依赖的一些关键命令是否能正常执行。

解决方案

- 问题场景一：包管理器命令执行失败

  检查到包管理器命令rpm或dpkg命令执行失败，请登录节点排查下列命令的可用性。

  ```
  rpm -qa
  ```

  如果上述命令不可用，可通过以下命令恢复：

  ```
  rpm --rebuilddb
  ```

- 问题场景二：systemctl status命令执行失败

  检查到节点systemctl status命令不可用，将影响众多检查项，请登录节点排查下列命令的可用性。

  ```
  systemctl status kubelet
  ```

  如果上述操作无法解决，建议您进行重置节点操作，参考**重置节点**。

2.5.5.45 节点 sock 文件挂载检查异常处理

检查项内容

检查节点上的Pod是否直接挂载docker/containerd.sock文件。升级过程中Docker/Containerd将会重启，宿主机sock文件发生变化，但是容器内的sock文件不会随之变化，二者不匹配，导致您的业务无法访问Docker/Containerd。Pod重建后sock文件重新挂载，可恢复正常。

通常K8S集群用户基于如下场景在容器中使用上述sock文件：

1. 监控类应用，以DaemonSet形式部署，通过sock文件连接Docker/Containerd，获取节点容器状态信息。
2. 编译平台类应用，通过sock文件连接Docker/Containerd，创建程序编译用容器。

解决方案

- 问题场景一：检查到应用存在该异常，进行整改。

  推荐您使用挂载目录的方式挂载sock文件。例如，若宿主机sock文件路径为/var/run/docker.sock，您可参考下述配置进行整改。注意，该整改生效时会触发Pod重建。

  ```
  kind: Deployment
  apiVersion: apps/v1
  metadata:
    name: test
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
  ```



```
    app: nginx
  spec:
    containers:
      - name: container-1
        image: 'nginx'
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - name: sock-dir
            mountPath: /var/run
    imagePullSecrets:
      - name: default-secret
    volumes:
      - name: sock-dir
        hostPath:
          path: /var/run
```

- 问题场景二：检查到应用存在该异常，明确应用使用场景后，接受sock短暂不可访问风险，继续升级。

  请选择跳过该检查项异常后重新检查，在集群升级完成后删除存量Pod，触发Pod重建，访问将恢复。

- 问题场景三：部分老版本的CCE插件存在该异常

  请将老版本的CCE插件升级至最新版本。例如1.2.2以下的dolphin插件存在该问题，需升级至1.2.2及以上版本。

- 问题场景四：日志分析里面出现“failed to execute docker ps -aq”错误。

  该报错出现一般是因为容器引擎功能异常导致，请您提工单联系运维人员处理。

2.5.5.46 HTTPS 类型负载均衡证书一致性检查异常处理

检查项内容

检查HTTPS类型负载均衡所使用的证书，是否在ELB服务侧被修改。

解决方案

该问题的出现，一般是由于用户在CCE中创建HTTPS类型Ingress后，直接在ELB证书管理功能中修改了Ingress引用的证书，导致CCE集群中存储的证书内容与ELB侧不一致，进而导致升级后ELB侧证书被覆盖。

步骤1 请登录ELB服务控制台，在“弹性负载均衡 > 证书管理”界面找到该证书，在证书描述字段中找到对应的secret_id。

图 2-46 查询证书

该secret_id即为集群中对应Secret的metadata.uid字段，可以根据该uid查询集群中Secret的名称。

您可以通过以下kubectl命令进行查询，其中*<secret_id>*请自行替换。

```
kubectl get secret --all-namespaces -o jsonpath='{range .items[*]}{"uid:"}{.metadata.uid}{" namespace:"}{.metadata.namespace}{" name:"}{.metadata.name}{"\n"}{end}' | grep <secret_id>
```

步骤2 仅v1.19.16-r2、v1.21.5-r0、v1.23.3-r0及以上版本的集群支持使用ELB服务中的证书，上述版本集群请参考**方案一**处理，其他版本集群请参考**方案二**处理。

- 方案一：您可以将Ingress使用的证书替换为ELB服务器证书，即可通过ELB控制台创建或编辑该证书。

  a. 请登录CCE控制台，前往“服务”页面并选择“路由”页签，找到使用该证书的路由，单击“更多 > 更新”。注意，这里可能有多个Ingess引用该证书，所涉及的Ingress都需要进行更新，可以根据Ingress的yaml文件的spec.tls中secretName字段判断是否引用该Secret中的证书。

  您可以通过以下kubectl命令进行查询引用该证书的Ingress，其中*<secret_name>*请自行替换。

  ```
  kubectl get ingress --all-namespaces -o jsonpath='{range .items[*]}{"namespace:"}{.metadata.namespace}{" name:"}{.metadata.name}{" tls:"}{.spec.tls[*]}{"\n"}{end}' | grep <secret_name>
  ```

  b. 在监听器配置中，选择服务器证书来源为“ELB服务器证书”，该证书可通过ELB控制台创建或编辑，单击“确定”。

  c. 在“配置与密钥”界面删除对应的Secret，删除前建议先备份。

- 方案二：您可以将Ingress使用的证书，覆写到集群对应的Secret资源中，避免在升级时出现ELB侧证书被更新。

  请登录CCE控制台，前往“配置与密钥”页面找到该Secret并编辑，填入您正在使用的证书并保存。

图 2-47 修改 Secret

----结束

2.5.5.47 节点挂载检查异常处理

检查项内容

检查节点上默认挂载目录及软链接是否被手动挂载或修改。

- 节点为非共享磁盘场景
  - CCE默认挂载/var/lib/docker或containerd、/mnt/paas/kubernetes/kubelet，检查/var、/var/lib、/mnt、/mnt/paas、/mnt/paas/kubernetes是否被用户挂载。
  - CCE默认创建链接/var/lib/kubelet -> /mnt/paas/kubernetes/kubelet，检查是否被用户修改。
- 节点为共享磁盘场景
  - CCE默认挂载/mnt/paas/，检查/mnt是否被用户挂载。
  - CCE默认创建软链接/var/lib/kubelet -> /mnt/paas/kubernetes/kubelet、/var/lib/docker或containerd-> /mnt/paas/runtime，检查是否被用户修改。

解决方案

如何确认是否共享磁盘

步骤1 根据检查信息，登录相应节点。

步骤2 执行**lsblk**命令，查看/mnt/paas挂载了vgpaas-share分区，若存在则是共享磁盘场景，若不存在，则是非共享磁盘场景。

图 2-48 查询是否为共享磁盘

```
[root@test-os-ugrade-35777 ~]# lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda            253:0    0   50G  0 disk
└─vda1         253:1    0   50G  0 part /
vdb            253:16   0  100G  0 disk
└─vgpaas-share 252:0    0  100G  0 lvm  /mnt/paas
```

----结束

节点挂载检查异常如何解决

1. 取消手动修改的挂载点。
2. 取消默认软链接修改。

2.5.5.48 节点 paas 用户登录权限检查异常处理

检查项内容

检查paas用户是否有登录权限。

解决方案

执行以下命令查看paas用户是否有登录权限：

```
sudo grep "paas" /etc/passwd
```

如果paas用户权限中带有"nologin"或者"false"，说明paas用户没有登录权限，需要先恢复paas用户的登录权限命令。

执行以下命令恢复paas用户权限之后重新检查：

```
usermod -s /bin/bash paas
```

2.5.5.49 ELB IPv4 私网地址检查异常处理

检查项内容

检查集群内负载均衡类型的Service所关联的ELB实例是否包含IPv4私网IP。

解决方案

解决方案一：删除关联无IPv4私网地址ELB的负载均衡型Service。

解决方案二：为无IPv4私网IP地址的ELB绑定一个私网IP。步骤如下：

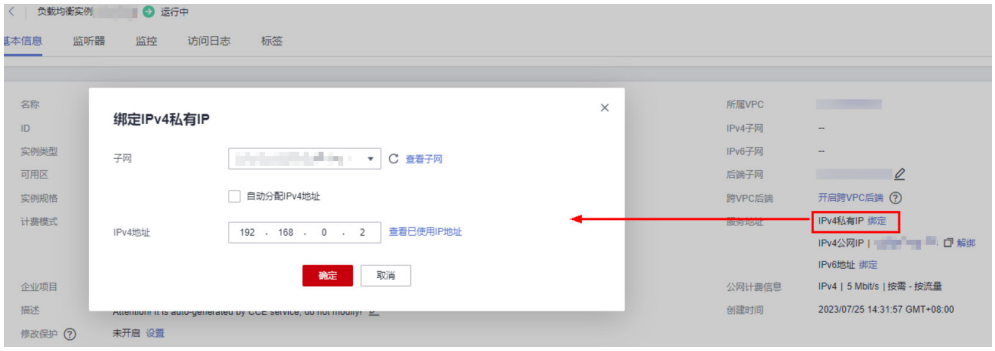
步骤1 查找负载均衡类型的Service所关联的ELB。

- 方法一：通过升级前检查的日志信息中，获取对应的ELB ID。然后前往ELB控制台通过ELB ID进行筛选。
  elbs (**ids: [*****]**) without ipv4 private ip, please bind private ip tothese elbs and try again
- 方法二：登录CCE控制台，前往“服务”页面查看服务，单击ELB名称，跳转到ELB界面。

步骤2 确认ELB实例是否包含IPv4私网IP。

步骤3 为无IPv4私网IP地址的ELB绑定一个私网IP。

1. 登录CCE控制台，单击目标ELB名称。
2. 在基本信息页面，单击“IPv4私有IP”旁的“绑定”。
3. 设置子网及IPv4地址，并单击“确定”。



----结束

2.5.5.50 检查历史升级记录是否满足升级条件

检查项内容

检查集群的历史升级记录，要求您的集群原始版本满足升级到目标集群版本的条件。

解决方案

该问题一般由于您的集群从比较老的版本升级而来，升级风险较大，建议您优先考虑集群迁移

若您仍然想要升级该集群，请您提交工单，联系技术支持人员进行评估。

2.5.5.51 检查集群管理平面网段是否与主干配置一致

检查项内容

检查集群管理平面网段是否与主干配置一致。

解决方案

该问题由于您的局点做过管理面网段配置修改，导致主干配置中的管理平面网段不一致；

请您提交工单，联系技术支持人员修改配置之后重启检查。

2.5.5.52 GPU 插件检查异常处理

检查项内容

检查到本次升级涉及GPU插件，可能影响新建GPU节点时GPU驱动的安装。

解决方案

由于当前GPU插件的驱动配置由您自行配置，需要您验证两者的兼容性。建议您在测试环境验证安装升级目标版本的GPU插件，并配置当前GPU驱动后，测试创建节点是否正常使用。

您可以执行以下步骤确认GPU插件的升级目标版本与当前驱动配置。

步骤1 登录CCE控制台，前往“插件中心”处查看**CCE AI套件（NVIDIA GPU）**插件。

步骤2 单击该插件的“升级”按钮，查看插件**目标版本**及**驱动版本**。

步骤3 在测试环境验证安装升级目标版本的GPU插件，并配置当前GPU驱动后，测试创建节点是否正常使用。

如果两者不兼容，您可以尝试替换高版本驱动。如有需要，请联系技术支持人员。

----结束

2.5.5.53 节点系统参数检查异常处理

检查项内容

检查您节点上默认系统参数是否被修改。

解决方案

如您的bms节点上bond0网络的mtu值非默认值1500，将出现该检查异常。

非默认参数可能导致业务丢包，请改回默认值。

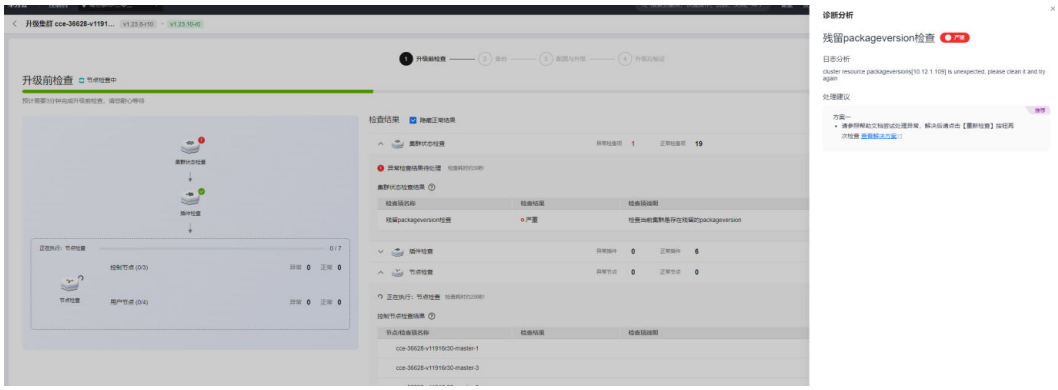
2.5.5.54 残留 packageversion 检查异常处理

检查项内容

检查当前集群中是否存在残留的packageversion。

解决方案

检查提示您的集群中存在残留的CRD资源10.12.1.109，该问题一般由于CCE早期版本节点删除后，对应的CRD资源未被清除导致。



您可以尝试手动执行以下步骤：

步骤1 备份残留的CRD资源。10.12.1.109 为示例资源，请根据报错中提示的资源进行替换。

```
kubectl get packageversion 10.12.1.109 -oyaml > /tmp/packageversion-109.bak
```

步骤2 清除残留的CRD资源。

```
kubectl delete packageversion 10.12.1.109
```

步骤3 上述步骤执行完成之后尝试重新检查。

----结束

2.5.5.55 节点命令行检查异常处理

检查项内容

检查节点中是否存在升级所必须的命令。

解决方案

该问题一般由于节点上缺少集群升级流程中使用到的关键命令，可能会导致集群升级失败。

报错信息如下：

```
__error_code#ErrorCommandNotExist#chage command is not exists#__
__error_code#ErrorCommandNotExist#chown command is not exists#__
__error_code#ErrorCommandNotExist#chmod command is not exists#__
__error_code#ErrorCommandNotExist#mkdir command is not exists#__
__error_code#ErrorCommandNotExist#in command is not exists#__
__error_code#ErrorCommandNotExist#touch command is not exists#__
__error_code#ErrorCommandNotExist#pidof command is not exists#__
```


以上报错代表您的节点上缺少了chage、chown、chmod、mkdir、in、touch、pidof等命令，请安装对应命令之后重新检查。

2.5.5.56 节点交换区检查异常处理

检查项内容

检查集群CCE节点的上是否开启了交换区。

解决方案

CCE节点默认关闭swap交换区，请您确认手动开启交换区的原因，并确定关闭影响；

若确定无影响后请执行**swapoff -a**命令关闭交换区之后重新检查。

2.5.5.57 nginx-ingress 插件升级检查异常处理

检查项内容

- 检查项一：检查集群中是否存在未指定Ingress类型（ annotations中未添加kubernetes.io/ingress.class: nginx ）的Nginx Ingress路由。
- 检查项二：检查Nginx Ingress Controller后端指定的DefaultBackend Service是否存在。

问题自检

检查项一自检

针对Nginx类型的Ingress资源，查看对应Ingress的YAML，如Ingress的YAML中未指定Ingress类型，并确认该Ingress由Nginx Ingress Controller管理，则说明该Ingress资源存在风险。关于该问题的触发原因详情，请参见**问题根因**。

步骤1 获取Ingress类别。

您可以通过如下命令获取Ingress类别：

```
kubectl get ingress <ingress-name> -oyaml | grep -E ' kubernetes.io/ingress.class: | ingressClassName:'
```

- 故障场景：如果上述命令输出为空，说明Ingress资源未指定类别。
- 正常场景：Ingress已通过annotations或ingressClassName指定其类别，即存在输出。

```
[root@ + + + + paas]# kubectl get ingress test -oyaml | grep -E ' kubernetes.io/ingress.class: | ingressClassName:' -B 1
Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
  annotations:
    kubernetes.io/ingress.class: nginx
--
spec:
  ingressClassName: nginx
```

步骤2 确认该Ingress被Nginx Ingress Controller纳管。如果使用ELB类型的Ingress则无此问题。

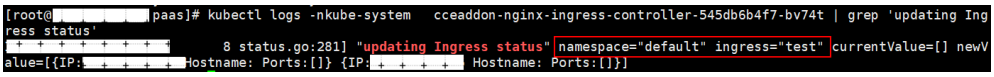
- 1.19集群可由通过managedFields机制确认。

  ```
  kubectl get ingress <ingress-name> -oyaml | grep 'manager: nginx-ingress-controller'
  ```

  ```
  [root@192-168-0-31 paas]# kubectl get ingress test -oyaml | grep 'manager: nginx-ingress-controller'
  Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
      manager: nginx-ingress-controller
  ```

- 其他版本集群可通过Nginx Ingress Controller Pod的日志确认。

  ```
  kubectl logs -nkube-system cceaddon-nginx-ingress-controller-545db6b4f7-bv74t | grep 'updating Ingress status'
  ```

若通过上述两种方式仍然无法确认，请联系技术支持人员。

----结束

检查项二自检

步骤1 在Nginx Ingress Controller部署的命名空间内查看对应的DefaultBackend Service。

```
kubectl get pod cceaddon-nginx-ingress-<controller-name>-controller-*** -n <namespace> -oyaml | grep 'default-backend'
```

其中cceaddon-nginx-ingress-<controller-name>-controller-***为Controller Pod名称，<controller-name>为安装插件时指定的控制器名称，<namespace>为Controller部署的命名空间。

回显如下：

```
- '--default-backend-service=<namespace>/<backend-svc-name>'
```

其中<backend-svc-name>为Controller对应的DefaultBackend Service名称。

步骤2 检查Nginx Ingress Controller对应的DefaultBackend Service是否存在。

```
kubectl get svc <backend-svc-name> -n <namespace>
```

如果无法查询到对应的Service，则无法通过该检查项。

----结束

解决方案

检查项一解决方案

为Nginx类型的Ingress添加注解，方式如下：

```
kubectl annotate ingress <ingress-name> kubernetes.io/ingress.class=nginx
```

须知

ELB类型的Ingress无需添加该注解，请**确认**该Ingress被Nginx Ingress Controller纳管。

检查项二解决方案

重新创建DefaultBackend Service。

- 如果安装插件时，在“默认404服务”配置项中指定了自定义的DefaultBackend Service，请您自行重新创建相同的Service。
- 如果安装插件时使用默认的DefaultBackend Service，则重新创建的YAML示例如下。

```
apiVersion: v1
kind: Service
metadata:
  name: cceaddon-nginx-ingress-<controller-name>-default-backend    # <controller-name>为控制器名称
  namespace: kube-system
  labels:
    app: nginx-ingress-<controller-name>
```



```
    app.kubernetes.io/managed-by: Helm
    chart: nginx-ingress-<version>    # <version>为插件版本
    component: default-backend
    heritage: Helm
    release: cceaddon-nginx-ingress-<controller-name>
  annotations:
    meta.helm.sh/release-name: cceaddon-nginx-ingress-<controller-name>
    meta.helm.sh/release-namespace: kube-system    # 插件安装的命名空间
spec:
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: http
  selector:
    app: nginx-ingress-<controller-name>
    component: default-backend
    release: cceaddon-nginx-ingress-<controller-name>
  type: ClusterIP
  sessionAffinity: None
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  internalTrafficPolicy: Cluster
```

2.5.5.58 云原生监控插件升级检查异常处理

检查项内容

在集群升级过程中，云原生监控插件从3.9.0之前的版本升级至3.9.0之后的版本升级时，存在兼容性问题，需检查该插件是否开启了grafana的开关。

解决方案

由于云原生监控插件在3.9.0之后的版本，不再聚合grafana的能力，因此升级前需要重新安装开源版本grafana插件。

说明

- 重新安装grafana不会影响已有的数据。
- 手动创建的grafana的服务（service）和路由（ingress）无法直接绑定至新的grafana插件，需要手动修改服务的选择器的配置，请及时修改对应的选择器。

方案一：如果当前插件能够升级至3.9.0及以上的版本，请前往“插件中心”页面，单击云原生监控插件的“升级”按钮。然后在新窗口中单击“安装Grafana”，等待请求下发完成后并单击“继续升级插件”，将插件升级至最新版本。

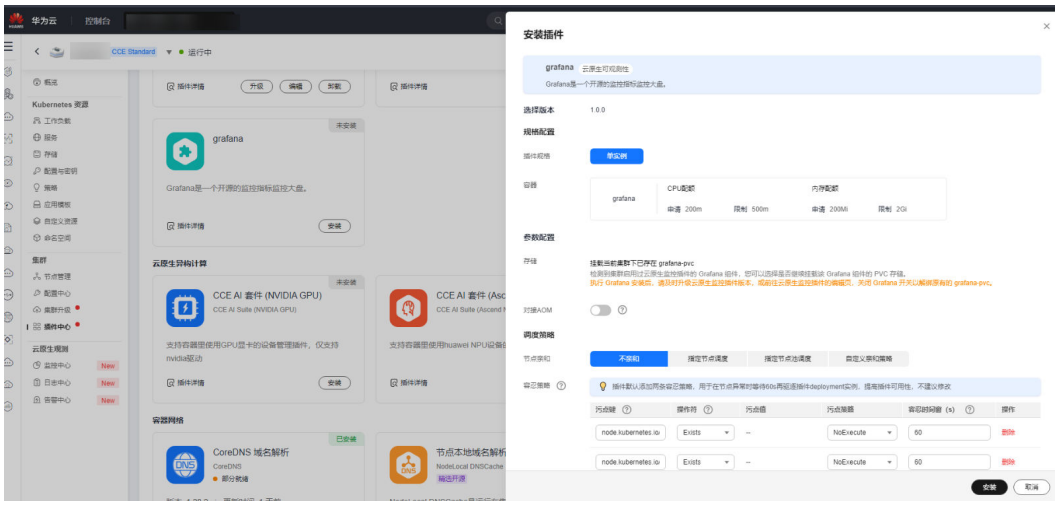
方案二：如果当前插件不支持升级至3.9.0及以上的版本，则按照以下方式，手动迁移grafana的插件。

步骤1 关闭云原生监控插件的grafana开关。

进入插件中心，在云原生监控插件的编辑页面，关闭grafana的开关。

步骤2 安装开源grafana插件。

在插件中心的页面中，安装grafana插件。



----结束

2.5.5.59 Containerd Pod 重启风险检查异常处理

检查项内容

检查当前集群内使用containerd的节点在升级containerd组件时，节点上运行的业务容器是否可能发生重启，造成业务影响。

解决方案

检测到您的节点上的containerd服务存在重启风险；请确保在业务影响可控的前提下（如业务低峰期）进行集群升级，以消减业务容器重启带来的影响；

如需帮助，请提交工单联系运维人员获取支持。

2.5.5.60 GPU 插件关键参数检查异常处理

检查项内容

检查CCE GPU插件中部分配置是否被侵入式修改，被侵入式修改的插件可能导致升级失败。

解决方案

步骤1 使用kubectl连接集群。

步骤2 执行以下命令获取插件实例详情。

```
kubectl get ds nvidia-driver-installer -nkube-system -oyaml
```

步骤3 请检查UpdateStrategy字段值是否被修改为OnDelete，应改回RollingUpdate。

步骤4 请检查NVIDIA_DRIVER_DOWNLOAD_URL字段是否与插件页面的GPU驱动版本一致，若不一致，请在页面上修改为正确的驱动版本。

----结束

2.5.5.61 GPU/NPU Pod 重建风险检查异常处理

检查项内容

检查当前集群升级重启kubelet时，节点上运行的GPU/NPU业务容器是否可能发生重建，造成业务影响。

解决方案

请确保在业务影响可控的前提下（如业务低峰期）进行集群升级，以消减业务容器重建带来的影响；

如需帮助，请您提交工单联系运维人员获取支持。

2.5.5.62 ELB 监听器访问控制配置项检查异常处理

检查项内容

检查当前集群Service是否通过annotation配置了ELB监听器的访问控制。

若有配置访问控制则检查相关配置项是否正确。

解决方案

如果配置项存在错误，请参考**为负载均衡类型的Service配置黑名单/白名单访问策略**进行重新配置。

2.5.5.63 Master 节点规格检查异常处理

检查项内容

检查本次升级集群的Master节点规格与实际的Master节点规格是否一致。

解决方案

该问题一般因为您进行过Master节点改造，此次升级可能会将您的Master节点重置为标准版本;

如您无法确认影响，请您提交工单联系运维人员支撑。

2.5.5.64 Master 节点子网配额检查异常处理

检查项内容

检查本次升级集群子网剩余可用IP数量是否支持滚动升级。

解决方案

该问题一般因为您选择的集群子网的IP数量不够，无法支持滚动升级；

请您迁移对应子网中的节点之后重试检查，若您无法确认迁移影响，请您提交工单联系运维人员支撑。

2.5.5.65 节点运行时检查异常处理

检查项内容

该告警通常发生在低版本集群升级到v1.27及以上集群。CCE不建议您在1.27以上版本集群中继续使用docker，并计划在未来移除对docker的支持。

解决方案

如果您仍想在1.27以上集群中创建并使用docker节点，可跳过该告警，但推荐您尽快切换至containerd，它提供了更出色的用户体验和更强大的功能。

2.5.5.66 节点池运行时检查异常处理

检查项内容

该告警通常发生在低版本集群升级到v1.27及以上集群。CCE不建议您在1.27以上版本集群中继续使用docker，并计划在未来移除对docker的支持。

解决方案

如果您仍想在1.27以上集群中创建并使用docker节点池，可跳过该告警，但推荐您尽快切换至containerd，它提供了更出色的用户体验和更强大的功能。

2.5.5.67 检查节点镜像数量异常处理

检查项内容

检查到您的节点上镜像数量过多（>1000个），可能导致docker启动过慢，影响docker标准输出，影响nginx等功能的正常使用。

解决方案

请手动删除残留的镜像，防止后续升级异常；

删除镜像之后请您重新进行升级前检查

2.5.5.68 OpenKruise 插件兼容性检查异常处理

检查项内容

检查集群升级时，OpenKruise插件是否存在兼容性问题。

解决方案

Kubernetes社区在1.24版本移除了对dockershim的支持。CCE为兼顾用户使用docker运行时的习惯，在CCE的v1.25及以上的集群版本引入了cri-dockerd用于替换原来的dockershim，但是OpenKruise社区当前并未实现对cri-dockerd的支持（参见**issue**）。

因此，在v1.25及以上版本的集群中安装1.0.3版本的OpenKruise插件时，kruise-daemon无法在使用docker容器引擎的节点上运行，请使用containerd容器引擎。

您可以选择以下方案之一进行解决：

- 方案一：关闭OpenKruise插件的kruise-daemon配置，然后重试集群升级。
- 方案二：将集群中运行时为docker的节点迁移至containerd，详情请参见**将节点容器引擎从Docker迁移到Containerd**。

2.5.5.69 Secret 落盘加密特性兼容性检查异常处理

检查项内容

检查本次升级的目标版本是否支持Secret落盘加密特性，若不支持则不允许开启Secret落盘加密特性的集群升级至该版本。

解决方案

CCE从v1.27版本开始支持Secret落盘加密特性，开放该特性的版本号如下：

- v1.27集群：v1.27.10-r0及以上
- v1.28集群：v1.28.8-r0及以上
- v1.29集群：v1.29.4-r0及以上
- 其他更高版本集群

如果升级前集群已开启Secret落盘加密特性，则目标集群的版本同样需要支持Secret落盘加密特性，您需要选择满足条件的版本进行升级。

2.5.5.70 Ubuntu 内核与 GPU 驱动兼容性提醒

检查项内容

检查到集群中同时使用GPU插件和Ubuntu节点，提醒客户存在可能的兼容性问题。当Ubuntu内核版本在5.15.0-113-generic上时，GPU插件必须使用535.161.08及以上的驱动版本。

解决方案

您在升级后新创或者重置Ubuntu节点时，可能遇到该问题，请编辑GPU插件中的驱动版本至535.161.08及以上，然后重启该节点。

2.5.5.71 排水任务检查异常处理

检查项内容

检查到集群中存在未完成的排水任务，此时升级可能会导致升级完成后触发排水动作，将运行中的Pod进行驱逐。

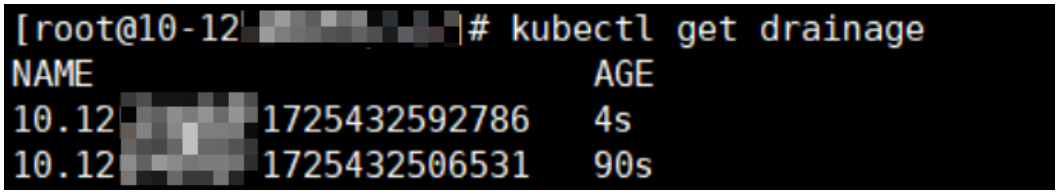
解决方案

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

步骤2 查看是否存在排水任务，以下为正常回显：

```
kubectl get drainage
```

图 2-49 排水任务，以下回显表示存在排水任务



请将drainage资源进行删除，删除之后再次触发升级前检查。

步骤3 执行以下命令删除排水任务。

```
kubectl delete drainage {排水任务名称}
```

----结束

2.5.5.72 节点镜像层数量异常检查

检查项内容

检查到您的节点上镜像层数量过多（>5000层），可能导致docker/containerd启动过慢，影响docker/containerd标准输出。

如果您集群中使用了nginx，可能会出现转发变慢等问题。

解决方案

请登录节点手动删除用不到的镜像，防止后续升级异常。

2.5.5.73 检查集群是否满足滚动升级条件

检查项内容

检查到您的集群暂时不满足滚动升级条件。

解决方案

该检查失败一般由于资源租户的资源配额不足引起，无法支持滚动升级；

请联系运维人员扩充资源之后重新检查。

2.5.5.74 轮转证书文件数量检查

检查项内容

检查您节点上的证书数量过多（>1000），由于升级过程中会批量处理证书文件，证书文件过多可能导致节点升级过慢，节点上Pod被驱逐等。

解决方案

方案一：优先建议您重置节点，详情请参考**重置节点**。

方案二：修复节点上证书轮转异常问题。

1. 进入节点/opt/cloud/cce/kubernetes/kubelet/pki/目录。
2. 备份节点上的证书文件kubelet-server-current.pem、kubelet-client-current.pem。
3. 删除节点上残留的kubelet-server-*证书文件。
```
link_target="$(basename $(readlink kubelet-server-current.pem))" && find -maxdepth 1 -type f -name 'kubelet-server-*.pem' ! -name "$link_target" -delete
```
4. 删除证书软连接文件。
```
find -maxdepth 1 -type f -name 'kubelet-server-current.pem'  -delete
```

2.5.5.75 Ingress 与 ELB 配置一致性检查

检查项内容

检查到您集群中Ingress配置与ELB配置不一致，请确认是否在ELB侧修改过Ingress自动创建的监听器、转发策略、转发规则、后端云服务器组、后端云服务器和证书配置。

升级后会覆盖您在ELB自行修改的内容，请整改后再进行集群升级。

解决方案

根据诊断分析中的日志排查哪些资源需要整改，常见场景是在Ingress对接的监听器下配置了其他的转发策略，导致监听器下转发策略与集群Ingress中配置的转发策略不一致，需要将您增加的转发策略迁移到其他监听器下，或者在Ingress中配置上该转发策略。

3 节点

3.1 节点概述

简介

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

说明

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

在云容器引擎CCE中，主要采用高性能的弹性云服务器ECS或裸金属服务器BMS作为节点来构建高可用的Kubernetes集群。

支持的节点规格

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

容器底层文件存储系统说明

Docker

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

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

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

Containerd

使用Containerd容器引擎的节点，均采用ext4格式的文件存储系统。

节点 paas 用户/用户组说明

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

须知

CCE组件正常运行依赖paas用户/用户组，您需要注意以下几点要求：

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

节点生命周期

生命周期是指节点从创建到删除（或释放）历经的各种状态。

表 3-1 节点生命周期状态说明

状态	状态属性	说明
运行中	稳定状态	节点正常运行状态，并且已连接上集群。 在这个状态的节点可以运行您的业务。
不可用	稳定状态	节点运行异常状态（包含关机状态）。 在这个状态下的实例，不能对外提供业务。
创建中	中间状态	创建节点实例后，在节点状态进入运行中之前的状态。
安装中	中间状态	节点处于安装Kubernetes软件的过程中。
升级中	中间状态	表示节点正处于升级过程中。
删除中	中间状态	节点处于正在被删除的状态。 如果长时间处于该状态，则说明出现异常。
错误	稳定状态	节点处于异常状态。 在这个状态下的实例，不能对外提供业务，需要**重置节点**。

3.2 容器引擎说明

容器引擎介绍

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

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

Kubernetes在v1.24版本中移除了Dockershim，并从此不再默认支持Docker容器引擎，详情请参见**Kubernetes即将移除Dockershim**。CCE计划未来移除对Docker容器引擎的支持，建议您将节点容器引擎从Docker迁移至Containerd，详情请参见**将节点容器引擎从Docker迁移到Containerd**。

表 3-2 容器引擎对比

对比	Containerd	Docker
调用链	kubelet --> CRI plugin（在containerd进程中） --> containerd	• Docker（Kubernetes 1.23及以下版本）：kubelet --> dockershim（在kubelet 进程中） --> docker --> containerd • Docker（Kubernetes 1.24及以上版本社区方案）：kubelet --> cri-dockerd（kubelet使用CRI接口对接cri-dockerd） --> docker --> containerd
命令	crictl/ctr	docker
Kubernetes CRI支持	原生支持	需通过dockershim或cri-dockerd提供CRI支持
Pod 启动延迟	低	高
kubelet CPU/内存占用	低	高
运行时CPU/内存占用	低	高

节点操作系统与容器引擎对应关系

说明

v1.23及以上的VPC网络集群都支持Containerd，容器隧道网络集群从v1.23.2-r0开始支持Containerd。

表 3-3 CCE 集群节点操作系统与容器引擎对应关系

操作系统	内核版本	容器引擎	容器存储Rootfs	容器运行时
CentOS 7.6	3.x	Docker 1.23起支持Containerd	1.19.16以下版本集群使用Device Mapper 1.19.16及以上版本集群使用OverlayFS	runC
EulerOS 2.3	3.x	Docker	Device Mapper	runC
EulerOS 2.5	3.x	Docker	Device Mapper	runC
EulerOS 2.9	4.x	Docker 1.23起支持Containerd	OverlayFS	runC
Ubuntu 18.04	4.x	Docker 1.23起支持Containerd	OverlayFS	runC
Ubuntu 22.04	4.x	Docker 1.23起支持Containerd	OverlayFS	runC
Huawei Cloud EulerOS 1.1	3.x	Docker Containerd	OverlayFS	runC
Huawei Cloud EulerOS 2.0	5.x	Docker Containerd	OverlayFS	runC

表 3-4 CCE Turbo 集群节点操作系统与容器引擎对应关系

节点类型	操作系统	内核版本	容器引擎	容器存储Rootfs	容器运行时
弹性云服务器-虚拟机	CentOS 7.6	3.x	Docker Containerd	OverlayFS	runC
	Ubuntu 18.04	4.x			
	EulerOS 2.9	4.x			

节点类型	操作系统	内核版本	容器引擎	容器存储Rootfs	容器运行时
	Huawei Cloud EulerOS 1.1	3.x			
	Huawei Cloud EulerOS 2.0	5.x			
弹性云服务器-物理机	EulerOS 2.9	4.x	Containerd	Device Mapper	Kata

表 3-5 鲲鹏节点操作系统与容器引擎对应关系

操作系统	内核版本	容器引擎	容器存储Rootfs	容器运行时
Huawei Cloud EulerOS 2.0	5.x	Docker Containerd	OverlayFS	runC
EulerOS 2.9	4.x	Docker Containerd	OverlayFS	runC
EulerOS 2.8	4.x	Docker	OverlayFS	runC

Containerd 和 Docker 组件常用命令对比

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

表 3-6 镜像相关功能

操作	Docker命令	Containerd命令	
	docker	crictl	ctr
列出本地镜像列表	docker images	crictl images	ctr -n k8s.io i ls
拉取镜像	docker pull	crictl pull	ctr -n k8s.io i pull
上传镜像	docker push	无	ctr -n k8s.io i push
删除本地镜像	docker rmi	crictl rmi	ctr -n k8s.io i rm
检查镜像	docker inspect	crictl inspecti	无

表 3-7 容器相关功能

操作	Docker命令	Containerd命令	
	docker	crictl	ctr
列出容器列表	docker ps	crictl ps	ctr -n k8s.io c ls
创建容器	docker create	crictl create	ctr -n k8s.io c create
启动容器	docker start	crictl start	ctr -n k8s.io run
停止容器	docker stop	crictl stop	无
删除容器	docker rm	crictl rm	ctr -n k8s.io c del
连接容器	docker attach	crictl attach	无
进入容器	docker exec	crictl exec	无
查看容器详情	docker inspect	crictl inspect	ctr -n k8s.io c info
查看容器日志	docker logs	crictl logs	无
查看容器的资源使用情况	docker stats	crictl stats	无
更新容器资源限制	docker update	crictl update	无

表 3-8 Pod 相关功能

操作	Docker命令	Containerd命令	
	docker	crictl	ctr
列出Pod列表	无	crictl pods	无
查看Pod详情	无	crictl inspectp	无
启动Pod	无	crictl start	无
运行Pod	无	crictl runp	无
停止Pod	无	crictl stopp	无
删除Pod	无	crictl rmp	无

说明

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

调用链区别

- Docker（Kubernetes 1.23及以下版本）：

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

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

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

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

容器引擎版本说明

- Docker
  - EulerOS/CentOS：docker-engine 18.9.0，CCE定制的Docker版本，会及时修复安全漏洞。
  - Ubuntu：docker-ce: 24.0.9，开源社区版本。Ubuntu节点建议使用containerd引擎。

    说明

    Ubuntu下开源docker-ce在并发exec（如配置了多个exec探针时）可能触发社区bug，建议使用http/tcp的探针。
- Containerd：1.6.14

  说明

  集群版本为v1.28.8-r0、v1.29.4-r0、v1.30.1-r0及以上时，Containerd版本升级至1.7.16。

3.3 节点操作系统说明

本文为您提供当前已经发布的集群版本与操作系统版本的对应关系。

弹性云服务器-虚拟机

表 3-9 弹性云服务器-虚拟机节点操作系统

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
Huawei Cloud EulerOS 2.0	v1.30	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.x86_64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.29	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.x86_64
	v1.28	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.x86_64
	v1.27	√	v1.27.3-r0及以上版本支持	√	5.10.0-182.0.0.95.r1941_123.hce2.x86_64
	v1.25	√	v1.25.6-r0及以上版本支持	√	• v1.25.3-r0及以上集群版本：5.10.0-182.0.0.95.r1941_123.hce2.x86_64 • v1.25.3-r0以下集群版本：5.10.0-60.18.0.50.r865_35.hce2.x86_64
	v1.23	√	v1.23.11-r0及以上版本支持	√	• v1.23.8-r0及以上集群版本：5.10.0-182.0.0.95.r1941_123.hce2.x86_64 • v1.23.8-r0以下集群版本：5.10.0-60.18.0.50.r865_35.hce2.x86_64
Huawei Cloud EulerOS 2.0 (ARM)	v1.30	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.aarch64
	v1.29	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.aarch64
	v1.28	√	√	√	5.10.0-182.0.0.95.r1941_123.hce2.aarch64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.27	√	v1.27.3-r0及以上版本支持	√	5.10.0-182.0.0.95.r1941_123.hce2.aarch64
	v1.25	√	v1.25.6-r0及以上版本支持	√	• v1.25.3-r0及以上集群版本：5.10.0-182.0.0.95.r1941_123.hce2.aarch64 • v1.25.3-r0以下集群版本：5.10.0-60.18.0.50.r865_35.hce2.aarch64
	v1.23	√	v1.23.11-r0及以上版本支持	√	• v1.23.8-r0及以上集群版本：5.10.0-182.0.0.95.r1941_123.hce2.aarch64 • v1.23.8-r0以下集群版本：5.10.0-60.18.0.50.r865_35.hce2.aarch64
Ubuntu 22.04	v1.30	√	×	√	5.15.0-113-generic
	v1.29	√	×	√	5.15.0-113-generic
	v1.28	√	×	√	5.15.0-113-generic
	v1.27	√	×	√	• v1.27.3-r0及以上集群版本：5.15.0-113-generic • v1.27.3-r0以下集群版本：5.15.0-86-generic

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.25	√	×	√	• v1.25.6-r0及以上集群版本：5.15.0-113-generic • v1.25.6-r0以下集群版本：5.15.0-86-generic
	v1.23	√	×	√	• v1.23.11-r0及以上集群版本：5.15.0-113-generic • v1.23.11-r0以下集群版本：5.15.0-86-generic
Huawei Cloud EulerOS 1.1	v1.30	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.29	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.28	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.27	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.25	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.23	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
	v1.21（停止维护）	√	√	√	3.10.0-1160.76.2.hce1c.x86_64
CentOS Linux release 7.6	v1.30	√	√	√	3.10.0-1160.119.1.el7.x86 64
	v1.29	√	√	√	3.10.0-1160.119.1.el7.x86 64
	v1.28	√	√	√	3.10.0-1160.119.1.el7.x86 64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.27	√	√	√	3.10.0-1160.119.1.el7.x86 64
	v1.25	√	√	√	3.10.0-1160.119.1.el7.x86 64
	v1.23	√	√	√	• v1.23.3-r0及以上集群版本：3.10.0-1160.119.1.el7.x86 64 • v1.23.3-r0以下集群版本：3.10.0-1160.66.1.el7.x86_64
	v1.21（停止维护）	√	√	√	• v1.21.5-r0及以上集群版本：3.10.0-1160.108.1.el7.x86_64 • v1.21.4-r0集群版本：3.10.0-1160.66.1.el7.x86_64 • v1.21.4-r0以下集群版本：3.10.0-1160.25.1.el7.x86_64
	v1.19（停止维护）	√	√	√	3.10.0-1160.108.1.el7.x86_64
	v1.17.17（停止维护）	√	√	√	3.10.0-1160.15.2.el7.x86_64
	v1.17.9（停止维护）	√	√	√	3.10.0-1062.12.1.el7.x86_64
	v1.15.11（停止维护）	√	√	√	3.10.0-1062.12.1.el7.x86_64
	v1.15.6-r1（停止维护）	√	√	√	3.10.0-1062.1.1.el7.x86_64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.13.10-r1（停止维护）	√	√	√	3.10.0-957.21.3.el7.x86_64
	v1.13.7-r0（停止维护）	√	√	√	3.10.0-957.21.3.el7.x86_64
EulerOS release 2.9	v1.30	√	√	√	4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64
	v1.29	√	√	√	4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64
	v1.28	√	√	√	4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64
	v1.27	√	√	√	4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64
	v1.25	√	√	√	4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64
	v1.23	√	√	√	• v1.23.5-r0及以上集群版本：4.18.0-147.5.1.6.h1305.eulerosv2r9.x86_64 • v1.23.5-r0以下集群版本：4.18.0-147.5.1.6.h1017.eulerosv2r9.x86_64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.21（停止维护）	√	√	√	• v1.21.7-r0及以上集群版本：4.18.0-147.5.1.6.h1152.eulerosv2r9.x86_64 • v1.21.7-r0以下集群版本：4.18.0-147.5.1.6.h1017.eulerosv2r9.x86_64
	v1.19（停止维护）	√	√	√	4.18.0-147.5.1.6.h1152.eulerosv2r9.x86_64
EulerOS release 2.9 (ARM)	v1.30	√	√	√	4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64
	v1.29	√	√	√	4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64
	v1.28	√	√	√	4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64
	v1.27	√	√	√	4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64
	v1.25	√	√	√	4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.23	√	√	√	• v1.23.5-r0及以上集群版本：4.19.90-vhulk2103.1.0.h1263.eulerosv2r9.aarch64 • v1.23.5-r0以下集群版本：4.19.90-vhulk2103.1.0.h990.eulerosv2r9.aarch64
	v1.21（停止维护）	√	√	√	• v1.21.7-r0及以上集群版本：4.19.90-vhulk2103.1.0.h1144.eulerosv2r9.aarch64 • v1.21.7-r0以下集群版本：4.19.90-vhulk2103.1.0.h990.eulerosv2r9.aarch64
	v1.19（停止维护）	√	√	√	4.19.90-vhulk2103.1.0.h1144.eulerosv2r9.aarch64
EulerOS release 2.8（ARM）（停止维护）	v1.27及以上	×	×	×	-
	v1.25（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h1350.eulerosv2r8.aarch64
	v1.23（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h1350.eulerosv2r8.aarch64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.21（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h1350.eulerosv2r8.aarch64
	v1.19.16（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h1350.eulerosv2r8.aarch64
	v1.19.10（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h962.eulerosv2r8.aarch64
	v1.17.17（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h962.eulerosv2r8.aarch64
	v1.15.11（停止维护）	√	√	√	4.19.36-vhulk1907.1.0.h702.eulerosv2r8.aarch64
EulerOS release 2.5（停止维护）	v1.27及以上	×	×	×	-
	v1.25（停止维护）	√	√	√	3.10.0-862.14.1.5.h687.eulerosv2r7.x86_64
	v1.23（停止维护）	√	√	√	3.10.0-862.14.1.5.h687.eulerosv2r7.x86_64
	v1.21（停止维护）	√	√	√	3.10.0-862.14.1.5.h687.eulerosv2r7.x86_64
	v1.19.16（停止维护）	√	√	√	3.10.0-862.14.1.5.h687.eulerosv2r7.x86_64
	v1.19.10（停止维护）	√	√	√	3.10.0-862.14.1.5.h520.eulerosv2r7.x86_64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.19.8（停止维护）	√	√	√	3.10.0-862.14.1.5.h520.eulerosv2r7.x86_64
	v1.17.17（停止维护）	√	√	√	3.10.0-862.14.1.5.h470.eulerosv2r7.x86_64
	v1.17.9（停止维护）	√	√	√	3.10.0-862.14.1.5.h428.eulerosv2r7.x86_64
	v1.15.11（停止维护）	√	√	√	3.10.0-862.14.1.5.h428.eulerosv2r7.x86_64
	v1.15.6-r1（停止维护）	√	√	√	3.10.0-862.14.1.5.h328.eulerosv2r7.x86_64
	v1.13.10-r1（停止维护）	√	√	√	3.10.0-862.14.1.2.h249.eulerosv2r7.x86_64
	v1.13.7-r0（停止维护）	√	√	√	3.10.0-862.14.1.0.h197.eulerosv2r7.x86_64
Ubuntu 18.04 server 64bit（停止维护）	v1.27及以上	×	×	×	-
	v1.25（停止维护）	√	×	√	4.15.0-171-generic
	v1.23（停止维护）	√	×	√	4.15.0-171-generic
	v1.21（停止维护）	√	×	√	4.15.0-171-generic
	v1.19.16（停止维护）	√	×	√	4.15.0-171-generic
	v1.19.8（停止维护）	√	×	√	4.15.0-136-generic

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.17.17（停止维护）	√	×	√	4.15.0-136-generic

弹性云服务器-物理机

表 3-10 弹性云服务器-物理机节点操作系统

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
EulerOS release 2.10	v1.30	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.29	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.28	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.27	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.25	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.23	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.21（停止维护）	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64
	v1.19.16（停止维护）	√	√	√	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64

裸金属服务器

该服务器的规格信息请参见**裸金属服务器实例家族**。

表 3-11 裸金属服务器节点操作系统

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
Huawei Cloud EulerOS 2.0（部分局点及机型支持，请以控制台呈现为准）	v1.30	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
	v1.29	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
	v1.28	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
	v1.27	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
	v1.25	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
	v1.23	√	√	×	5.10.0-60.18.0.50.r1083_58.hce2.x86_64
EulerOS release 2.9（受限使用，请提交工单申请）	v1.30	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.29	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.28	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.27	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.25	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64

操作系统	集群版本	CCE Standard集群		CCE Turbo集群	最新内核信息
		VPC网络模型	容器隧道网络模型	云原生网络2.0	
	v1.23	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.21（停止维护）	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
	v1.19（停止维护）	√	√	×	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64
EulerOS release 2.3（停止维护）	v1.27及以上	×	×	×	-
	v1.25（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64
	v1.23（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64
	v1.21（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64
	v1.19（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64
	v1.17（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64
	v1.15.11（停止维护）	√	√	×	3.10.0-514.41.4.28.h62.x86_64

3.4 节点规格说明

您可以通过本节快速浏览CCE支持的节点规格清单及相关特性，帮助您选择合适的机型规格。

节点类型	说明	节点规格
弹性云服务器-虚拟机	使用KVM/擎天虚拟化技术的弹性云服务器类型，针对不同的应用场景，可以选择多种规格类型，提供不同的计算能力和存储能力。	X86机型： • **通用计算增强型** • **通用计算型** • **内存优化型** • **通用入门型** • **磁盘增强型** • **超高I/O型** • **Flexus云服务器X** ARM（鲲鹏）机型： • **鲲鹏通用计算增强型** • **鲲鹏内存优化型** • **鲲鹏超高I/O型** 异构资源机型： • **GPU加速型** • **AI加速型**
弹性云服务器-物理机	基于擎天架构，使用裸金属虚拟化技术的弹性云服务器类型，该类型的物理机资源和虚拟机资源处于同一个资源池，可实现动态混合调度。	**通用计算增强型**
裸金属服务器	基于裸金属服务器部署容器服务，提供高性能和低延迟的计算能力。	裸金属服务器的规格信息请参见**裸金属服务器实例家族**。

说明

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

通用计算增强型

通用计算增强型弹性云服务器是CPU独享型实例，实例间无CPU资源争抢，性能强劲稳定，搭载全新网络加速引擎，提供更高的网络性能。

表 3-12 通用计算增强型实例特点

规格名称	计算	网络	支持集群类型
通用计算增强型 aC7	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-232 • 基频/睿频：2.45GHz/3.5GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：2000万PPS • 最大内网带宽：100Gbps	CCE Standard集群 CCE Turbo集群
通用计算增强型C7	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-128 • 处理器：第三代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.5GHz、2.8GHz/3.5GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1200万PPS • 最大内网带宽：42Gbps	CCE Standard集群 CCE Turbo集群
通用计算增强型 C6ne	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-64 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1000万PPS • 最大内网带宽：40Gbps	CCE Turbo集群
通用计算增强型C6s	• CPU/内存配比：1:2 • vCPU数量范围：2-64 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：850万PPS • 最大内网带宽：30Gbps	CCE Standard集群

规格名称	计算	网络	支持集群类型
通用计算增强型C6	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-88 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1200万PPS • 最大内网带宽：44Gbps	CCE Standard集群
通用计算增强型C3	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-60 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：500万PPS • 最大内网带宽：16Gbps	CCE Standard集群

表 3-13 aC7 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限	虚拟化类型
ac7.large.2	2	4	2/1	40	2	2	16	KVM
ac7.xlarge.2	4	8	3/1.5	60	2	3	32	
ac7.2xlarge.2	8	16	4/2.5	100	4	4	64	
ac7.3xlarge.2	12	24	6/4	150	4	6	96	
ac7.4xlarge.2	16	32	8/5	200	8	8	128	
ac7.6xlarge.2	24	48	12/6	250	8	8	192	
ac7.8xlarge.2	32	64	15/8	300	16	8	256	

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限	虚拟化类型
ac7.12xlarge.2	48	96	22/12	400	16	8	256	
ac7.16xlarge.2	64	128	28/16	550	24	12	256	
ac7.24xlarge.2	96	192	40/25	800	24	12	256	
ac7.29xlarge.2	116	216	50/30	950	32	16	256	
ac7.32xlarge.2	128	256	55/35	1000	32	16	256	
ac7.48xlarge.2	192	384	100/80	1600	32	16	256	
ac7.58xlarge.2	232	432	120/100	2000	32	16	256	
ac7.large.4	2	8	2/1	40	2	2	16	
ac7.xlarge.4	4	16	3/1.5	60	2	3	32	
ac7.2xlarge.4	8	32	4/2.5	100	4	4	64	
ac7.3xlarge.4	12	48	6/4	150	4	6	96	
ac7.4xlarge.4	16	64	8/5	200	8	8	128	
ac7.6xlarge.4	24	96	12/6	250	8	8	192	
ac7.8xlarge.4	32	128	15/8	300	16	8	256	
ac7.12xlarge.4	48	192	22/12	400	16	8	256	
ac7.16xlarge.4	64	256	28/16	550	24	12	256	
ac7.24xlarge.4	96	384	40/25	800	24	12	256	

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限	虚拟化类型
ac7.29xlarge.4	116	464	50/30	950	32	16	256	
ac7.32xlarge.4	128	512	55/35	1000	32	16	256	
ac7.48xlarge.4	192	768	100/80	1600	32	16	256	
ac7.58xlarge.4	232	928	120/100	2000	32	16	256	

表 3-14 C7 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	云硬盘基础带宽/突发带宽（Gbps）	虚拟化类型
c7.large.2	2	4	4/0.8	40	50	2	2	16	1.5/6	基于QingTian架构的自研极简虚拟化
c7.xlarge.2	4	8	8/1.6	80	50	2	3	32	2/6	
c7.2xlarge.2	8	16	15/3	150	100	4	4	64	3/6	
c7.3xlarge.2	12	24	17/5	200	150	4	6	96	4/6	
c7.4xlarge.2	16	32	20/6	280	150	8	8	128	5/6	
c7.6xlarge.2	24	48	25/9	400	200	8	8	192	6/无	
c7.8xlarge.2	32	64	30/12	550	300	16	8	256	8/无	
c7.12xlarge.2	48	96	35/18	750	400	16	8	256	10/无	

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	云硬盘基础带宽/突发带宽（Gbps）	虚拟化类型
c7.16xlarge.2	64	128	36/24	1000	500	28	8	256	16/无	
c7.24xlarge.2	96	192	40/36	1100	800	32	8	256	20/无	
c7.32xlarge.2	128	256	42/40	1200	1000	32	8	256	24/无	
c7.large.4	2	8	4/0.8	40	50	2	2	16	1.5/6	
c7.xlarge.4	4	16	8/1.6	80	50	2	3	32	2/6	
c7.2xlarge.4	8	32	15/3	150	100	4	4	64	3/6	
c7.3xlarge.4	12	48	17/5	200	150	4	6	96	4/6	
c7.4xlarge.4	16	64	20/6	280	150	8	8	128	5/6	
c7.6xlarge.4	24	96	25/9	400	200	8	8	192	6/无	
c7.8xlarge.4	32	128	30/12	550	300	16	8	256	8/无	
c7.12xlarge.4	48	192	35/18	750	400	16	8	256	10/无	
c7.16xlarge.4	64	256	36/24	1000	500	28	8	256	16/无	
c7.24xlarge.4	96	384	40/36	1100	800	32	8	256	20/无	
c7.32xlarge.4	128	512	42/40	1200	1000	32	8	256	24/无	

表 3-15 C6ne 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
c6ne.large.2	2	4	4/1.2	40	2	2	16
c6ne.large.4	2	8	4/1.2	40	2	2	16
c6ne.xlarge.2	4	8	8/2.4	80	2	3	32
c6ne.xlarge.4	4	16	8/2.4	80	2	3	32
c6ne.2xlarge.2	8	16	15/4.5	150	4	4	64
c6ne.2xlarge.4	8	32	15/4.5	150	4	4	64
c6ne.3xlarge.2	12	24	17/7	200	4	6	96
c6ne.3xlarge.4	12	48	17/7	200	4	6	96
c6ne.4xlarge.2	16	32	20/9	280	8	8	128
c6ne.4xlarge.4	16	64	20/9	280	8	8	128
c6ne.6xlarge.2	24	48	25/14	400	8	8	192
c6ne.6xlarge.4	24	96	25/14	400	8	8	192
c6ne.8xlarge.2	32	64	30/18	550	16	8	256
c6ne.8xlarge.4	32	128	30/18	550	16	8	256
c6ne.12xlarge.2	48	96	35/27	750	16	8	256
c6ne.12xlarge.4	48	192	35/27	750	16	8	256
c6ne.16xlarge.2	64	128	40/36	1,000	32	8	256

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
c6ne.16xlarge.4	64	256	40/36	1,000	32	8	256

表 3-16 C6s 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	虚拟化类型
c6s.large.2	2	4	1/1	30	50	2	2	KVM
c6s.xlarge.2	4	8	2/2	60	50	2	3	KVM
c6s.2xlarge.2	8	16	4/4	120	100	4	4	KVM
c6s.3xlarge.2	12	24	5.5/5.5	180	150	4	6	KVM
c6s.4xlarge.2	16	32	7.5/7.5	240	150	8	8	KVM
c6s.6xlarge.2	24	48	11/11	350	200	8	8	KVM
c6s.8xlarge.2	32	64	15/15	450	300	16	8	KVM
c6s.12xlarge.2	48	96	22/22	650	400	16	8	KVM
c6s.16xlarge.2	64	128	30/30	850	500	32	8	KVM
c6s.large.4	2	8	1/1	30	50	2	2	KVM
c6s.xlarge.4	4	16	2/2	60	50	2	3	KVM
c6s.2xlarge.4	8	32	4/4	120	100	4	4	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	虚拟化类型
c6s.3xlarge.4	12	48	5.5/5.5	180	150	4	6	KVM
c6s.4xlarge.4	16	64	7.5/7.5	240	150	8	8	KVM
c6s.6xlarge.4	24	96	11/11	350	200	8	8	KVM
c6s.8xlarge.4	32	128	15/15	450	300	16	8	KVM
c6s.12xlarge.4	48	192	22/22	650	400	16	8	KVM
c6s.16xlarge.4	64	256	30/30	850	500	32	8	KVM

表 3-17 C3 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	云硬盘基础带宽（Gbps）	虚拟化类型
c3.large.2	2	4	1.5/0.6	30	2	1	KVM
c3.xlarge.2	4	8	3/1	50	2	1.5	KVM
c3.2xlarge.2	8	16	5/2	90	4	2	KVM
c3.3xlarge.2	12	24	7/3	110	4	2.5	KVM
c3.4xlarge.2	16	32	10/4	130	4	3	KVM
c3.6xlarge.2	24	48	12/6	200	8	3.5	KVM
c3.8xlarge.2	32	64	15/8	260	8	4	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	云硬盘基础带宽（Gbps）	虚拟化类型
c3.15xlarge.2	60	128	16/16	500	16	8	KVM
c3.large.4	2	8	1.5/0.6	30	2	1	KVM
c3.xlarge.4	4	16	3/1	50	2	1.5	KVM
c3.2xlarge.4	8	32	5/2	90	4	2	KVM
c3.3xlarge.4	12	48	7/3	110	4	2.5	KVM
c3.4xlarge.4	16	64	10/4	130	4	3	KVM
c3.6xlarge.4	24	96	12/6	200	8	3.5	KVM
c3.8xlarge.4	32	128	15/8	260	8	4	KVM
c3.15xlarge.4	60	256	16/16	500	16	8	KVM

通用计算型

通用计算型弹性云服务器提供基本水平的vCPU性能、平衡的计算、内存和网络资源，同时可根据工作负载的需要实现性能的突增，具有短期发挥更高性能的能力。

表 3-18 通用计算型实例特点

规格名称	计算	网络	支持集群类型
通用计算型S7	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-8 • 处理器：第三代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.8GHz/3.5GHz	• 支持IPv6 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：50万PPS • 最大内网带宽：3Gbps	CCE Standard集群 CCE Turbo集群

规格名称	计算	网络	支持集群类型
通用计算型S6	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-8 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 支持IPv6 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：50万PPS • 最大内网带宽：3Gbps	CCE Standard集群
通用计算型Sn3	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-16 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：2.2GHz/3.0GHz	• 支持IPv6 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：50万PPS • 最大内网带宽：3Gbps	CCE Standard集群
通用计算型S3	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-16 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：2.2GHz/3.0GHz	• 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：30万PPS • 最大内网带宽：4Gbps	CCE Standard集群
通用计算型S2	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-32 • 处理器：英特尔® 至强® 处理器E5 v4家族 • 基频/睿频：2.4GHz/3.3GHz	• 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：50万PPS • 最大内网带宽：6Gbps	CCE Standard集群

表 3-19 S7 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限	虚拟化类型
s7.large.2	2	4	1.5/0.2	15	1	2	8	KVM
s7.xlarge.2	4	8	2/0.35	25	1	2	16	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限	虚拟化类型
s7.2xlarge.2	8	16	3/0.75	50	2	2	32	KVM
s7.large.4	2	8	1.5/0.2	15	1	2	8	KVM
s7.xlarge.4	4	16	2/0.35	25	1	2	16	KVM
s7.2xlarge.4	8	32	3/0.75	50	2	2	32	KVM

表 3-20 S6 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	虚拟化类型
s6.large.2	2	4	1.5/0.2	15	1	2	KVM
s6.xlarge.2	4	8	2/0.35	25	1	2	KVM
s6.2xlarge.2	8	16	3/0.75	50	2	2	KVM
s6.large.4	2	8	1.5/0.2	15	1	2	KVM
s6.xlarge.4	4	16	2/0.35	25	1	2	KVM
s6.2xlarge.4	8	32	3/0.75	50	2	2	KVM

表 3-21 Sn3 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	虚拟化类型
sn3.large.2	2	4	1.5/0.35	15	1	2	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	虚拟化类型
sn3.xlarge.2	4	8	2/0.7	25	1	2	KVM
sn3.2xlarge.2	8	16	3/1.3	50	2	2	KVM
sn3.4xlarge.2	16	32	6/2.5	100	4	2	KVM
sn3.large.4	2	8	1.5/0.35	15	1	2	KVM
sn3.xlarge.4	4	16	2/0.7	25	1	2	KVM
sn3.2xlarge.4	8	32	3/1.3	50	2	2	KVM
sn3.4xlarge.4	16	64	6/2.5	100	4	2	KVM

表 3-22 S3 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	虚拟化类型
s3.large.2	2	4	0.8/0.2	10	1	KVM
s3.xlarge.2	4	8	1.5/0.4	15	1	KVM
s3.2xlarge.2	8	16	3/0.8	20	2	KVM
s3.4xlarge.2	16	32	4/1.5	30	4	KVM
s3.large.4	2	8	0.8/0.2	10	1	KVM
s3.xlarge.4	4	16	1.5/0.4	15	1	KVM
s3.2xlarge.4	8	32	3/0.8	20	2	KVM
s3.4xlarge.4	16	64	4/1.5	30	4	KVM

表 3-23 S2 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	虚拟化类型
s2.large.2	2	4	0.8/0.2	10	1	KVM
s2.xlarge.2	4	8	1.5/0.4	15	1	KVM
s2.2xlarge.2	8	16	3/0.8	20	2	KVM
s2.4xlarge.2	16	32	4/1.5	30	4	KVM
s2.8xlarge.2	32	64	6/3	50	8	KVM
s2.large.4	2	8	0.8/0.2	10	1	KVM
s2.xlarge.4	4	16	1.5/0.4	15	1	KVM
s2.2xlarge.4	8	32	3/0.8	20	2	KVM
s2.4xlarge.4	16	64	4/1.5	30	4	KVM
s2.8xlarge.4	32	128	6/3	50	8	KVM

内存优化型

内存优化型弹性云服务器可应对大型内存数据集和高网络场景。适用于内存要求高，数据量大并且数据访问量大，同时要求快速的数据交换和处理。

表 3-24 内存优化型实例特点

规格名称	计算	网络	支持集群类型
内存优化型M7	• CPU/内存配比：1:8 • vCPU数量范围：2-128 • 处理器：第三代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.5GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1200万PPS • 最大内网带宽：42Gbps	CCE Standard集群 CCE Turbo集群

规格名称	计算	网络	支持集群类型
内存优化型M6ne	• CPU/内存配比：1:8 • vCPU数量范围：2-64 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1000万PPS • 最大内网带宽：40Gbps	CCE Turbo集群
内存优化型M6	• CPU/内存配比：1:8 • vCPU数量范围：2-64 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：1000万PPS • 最大内网带宽：40Gbps	CCE Standard集群
内存优化型M3	• CPU/内存配比：1:8 • vCPU数量范围：2-60 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：500万PPS • 最大内网带宽：17Gbps	CCE Standard集群

表 3-25 M6ne 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
m6ne.large.8	2	16	4/1.2	40	2	2	16
m6ne.xlarge.8	4	32	8/2.4	80	2	3	32
m6ne.2xlarge.8	8	64	15/4.5	150	4	4	64
m6ne.3xlarge.8	12	96	17/7	200	4	6	96

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
m6ne.4xlarge.8	16	128	20/9	280	8	8	128
m6ne.6xlarge.8	24	192	25/14	400	8	8	192
m6ne.8xlarge.8	32	256	30/18	550	16	8	256
m6ne.16xlarge.8	64	512	40/36	1000	32	8	256

表 3-26 M6 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	云硬盘基础带宽/突发带宽（Gbps）	虚拟化类型
m6.large.8	2	16	4/1.2	40	50	2	2	1/5	KVM
m6.xlarge.8	4	32	8/2.4	80	50	2	3	1.5/5	KVM
m6.2xlarge.8	8	64	15/4.5	150	100	4	4	2/5	KVM
m6.3xlarge.8	12	96	17/7	200	150	4	6	2.5/5	KVM
m6.4xlarge.8	16	128	20/9	280	150	8	8	3.5/5	KVM
m6.6xlarge.8	24	192	25/14	400	200	8	8	4/5	KVM
m6.8xlarge.8	32	256	30/18	550	300	16	8	7/10	KVM
m6.12xlarge.8	48	384	35/27	750	400	16	8	10/15	KVM
m6.16xlarge.8	64	512	40/36	1000	500	32	8	20/无	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	云硬盘基础带宽/突发带宽（Gbps）	虚拟化类型
m6.22xlarge.8.physical	88	768	40/40	1000	1000	16	33	20/无	裸金属

表 3-27 M3 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	云硬盘基础带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	云硬盘基础带宽（Gbps）	虚拟化类型
m3.large.8	2	16	1.5/0.6	1	30	2	1	KVM
m3.xlarge.8	4	32	3/1.1	1.5.	50	2	1.5.	KVM
m3.2xlarge.8	8	64	5/2	2	90	4	2	KVM
m3.3xlarge.8	12	96	8/3.5	2.5	110	4	2.5	KVM
m3.4xlarge.8	16	128	10/4.5	3	130	4	3	KVM
m3.6xlarge.8	24	192	12/6.5	3.5	200	8	3.5	KVM
m3.8xlarge.8	32	256	15/9	4	260	8	4	KVM
m3.15xlarge.8	60	512	17/17	8	500	16	8	KVM

通用入门型

通用入门型弹性云服务器性能受到基准性能和CPU积分的约束，适用于平时CPU都保持较低利用率而又需要瞬时冲高的场景，是成本最低的通用型实例。 使用积分机制不额外收取费用，详情请参见**CPU积分计算方法**。

表 3-28 通用入门型实例特点

规格名称	计算	网络	支持集群类型
通用入门型T6	• CPU/内存配比：1:1/1:2/1:4 • vCPU数量范围：2-16 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：2.2GHz/3.0GHz	• 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：60万PPS • 最大内网带宽：3Gbps	CCE Standard集群

表 3-29 T6 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	基准CPU计算性能（%）	平均基准CPU计算性能（%）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡个数上限	虚拟化类型
t6.xlarge.1	4	4	80	20	1/0.2	20	2	KVM
t6.2xlarge.1	8	8	120	15	2/0.4	40	2	KVM
t6.4xlarge.1	16	16	240	15	3/0.8	60	2	KVM
t6.large.2	2	4	40	20	0.5/0.1	10	1	KVM
t6.xlarge.2	4	8	80	20	1/0.2	20	2	KVM
t6.2xlarge.2	8	16	120	15	2/0.4	40	2	KVM
t6.4xlarge.2	16	32	240	15	3/0.8	60	2	KVM
t6.large.4	2	8	40	20	0.5/0.1	10	1	KVM

规格名称	vCPU	内存（GiB）	基准CPU计算性能（%）	平均基准CPU计算性能（%）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡个数上限	虚拟化类型
t6.xlarge.4	4	16	80	20	1/0.2	20	2	KVM
t6.2xlarge.4	8	32	120	15	2/0.4	40	2	KVM

GPU 加速型

GPU加速型云服务器（GPU Accelerated Cloud Server，GACS）能够提供强大的浮点计算能力，从容应对高实时、高并发的海量计算场景。

GPU加速型云服务器包括G系列和P系列两类。其中：

- G系列：图形加速型弹性云服务器，适合于3D动画渲染、CAD等。
- P系列：计算加速型或推理加速型弹性云服务器，适合于深度学习、科学计算、CAE等。

表 3-30 GPU 加速实例总览

类别	实例	GPU显卡	单卡Cuda Core数量	单卡GPU性能	使用场景	支持集群类型
图形加速型	G6	NVIDIA T4（GPU直通）	2560	• 8.1TFLOPS单精度浮点计算 • 130INT8 TOPS • 260INT4 TOPS	云桌面、图像渲染、3D可视化、重载图形设计。	CCE Standard集群
图形加速型	G5	NVIDIA V100（GPU直通）	5120	• 14TFLOPS 单精度浮点计算 • 7TFLOPS 双精度浮点计算 • 112TFLOPS Tensor Core深度学习加速	云桌面、图像渲染、3D可视化、重载图形设计。	CCE Standard集群

类别	实例	GPU显卡	单卡Cuda Core数量	单卡GPU性能	使用场景	支持集群类型
计算加速型	P2s	NVIDIA V100	5120	• 14TFLOPS 单精度浮点计算 • 7TFLOPS 双精度浮点计算 • 112TFLOPS Tensor Core 深度学习加速	AI深度学习训练、科学计算、计算流体动力学、计算金融、地震分析、分子建模、基因组学。	CCE Standard集群
计算加速型	P2v	NVIDIA V100 NVLink（GPU直通）	5120	• 15.7TFLOPS 单精度浮点计算 • 7.8TFLOPS 双精度浮点计算 • 125TFLOPS Tensor Core 深度学习加速 • 300GiB/s NVLINK	机器学习、深度学习、训练推理、科学计算、地震分析、计算金融学、渲染、多媒体编解码。	CCE Standard集群
推理加速型	Pi2	NVIDIA T4（GPU直通）	2560	• 8.1TFLOPS 单精度浮点计算 • 130INT8 TOPS • 260INT4 TOPS	机器学习、深度学习、训练推理、科学计算、地震分析、计算金融学、渲染、多媒体编解码。	CCE Standard集群
推理加速型	Pi1	NVIDIA P4（GPU直通）	2560	5.5TFLOPS 单精度浮点计算	机器学习、深度学习、训练推理、科学计算、地震分析、计算金融学、渲染、多媒体编解码。	CCE Standard集群

表 3-31 P2v 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	GPU	GPU连接技术	显存（GiB）	虚拟化类型	硬件配置
p2v.2xlarge.8	8	64	10/4	50	4	4	1 × V100	-	1 × 16GiB	KVM	CPU：Intel® Skylake-SP Xeon® Gold 6151 v5
p2v.4xlarge.8	16	128	15/8	100	8	8	2 × V100	NVLink	2 × 16GiB	KVM	
p2v.8xlarge.8	32	256	25/15	200	16	8	4 × V100	NVLink	4 × 16GiB	KVM	
p2v.16xlarge.8	64	512	30/30	400	32	8	8 × V100	NVLink	8 × 16GiB	KVM	

磁盘增强型

磁盘增强型弹性云服务器自带高存储带宽和IOPS的本地盘，具有高存储IOPS以及读写带宽的优势。同时，本地盘的价格更加低廉，在海量数据存储场景下，具备更高的性价比。

表 3-32 磁盘增强型实例特点

规格名称	计算	网络	支持集群类型
磁盘增强型D7	• CPU/内存配比：1:4 • vCPU数量范围：4-72 • 处理器：第三代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：850万PPS 最大内网带宽：42Gbps	CCE Standard集群 CCE Turbo集群

规格名称	计算	网络	支持集群类型
磁盘增强型D6	• CPU/内存配比：1:4 • vCPU数量范围：4-72 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：900万PPS • 最大内网带宽：44Gbps	CCE Standard集群 CCE Turbo集群

表 3-33 D7 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	本地盘（GiB）	虚拟化类型
d7.xlarge.4	4	16	5/1.7	60	50	2	3	32	2 × 3600	KVM
d7.2xlarge.4	8	32	10/3.5	120	100	4	4	64	4 × 3600	KVM
d7.4xlarge.4	16	64	20/6.7	240	150	4	6	96	8 × 3600	KVM
d7.6xlarge.4	24	96	25/10	350	200	8	8	128	12 × 3600	KVM
d7.8xlarge.4	32	128	30/13.5	450	300	8	8	192	16 × 3600	KVM
d7.12xlarge.4	48	192	40/20	650	400	16	8	256	24 × 3600	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	本地盘（GiB）	虚拟化类型
d7.16xlarge.4	64	256	42/27	850	500	16	8	256	32 × 3600	KVM

表 3-34 D6 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	本地盘（GiB）	虚拟化类型
d6.xlarge.4	4	16	5/2	60	50	2	3	2 × 3600	KVM
d6.2xlarge.4	8	32	10/4	120	100	4	4	4 × 3600	KVM
d6.4xlarge.4	16	64	20/7.5	240	150	8	8	8 × 3600	KVM
d6.6xlarge.4	24	96	25/11	350	200	8	8	12 × 3600	KVM
d6.8xlarge.4	32	128	30/15	450	300	16	8	16 × 3600	KVM
d6.12xlarge.4	48	192	40/22	650	400	16	8	24 × 3600	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	本地盘（GiB）	虚拟化类型
d6.16xlarge.4	64	256	42/30	850	500	32	8	32 × 3600	KVM
d6.18xlarge.4	72	288	44/34	900	700	32	8	36 × 3600	KVM

鲲鹏通用计算增强型

鲲鹏通用计算增强型云服务器搭载鲲鹏处理器，提供强劲的鲲鹏算力和高性能网络，更好地满足企业对云上业务性价比高、安全可靠等诉求。

表 3-35 鲲鹏通用计算增强型实例特点

规格名称	计算	网络	支持集群类型
鲲鹏通用计算增强型kC1	• CPU/内存配比：1:2/1:4 • vCPU数量范围：2-60 • 处理器：鲲鹏920处理器 • 基频：2.6GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：400万PPS • 最大内网带宽：30Gbps	CCE Standard集群
鲲鹏通用计算增强型kC1n	• CPU/内存配比：1:2/1:4 • vCPU数量范围：8-48 • 处理器：鲲鹏920处理器 • 基频：2.6GHz	• 支持IPv6 • 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：350万PPS • 最大内网带宽：25Gbps	CCE Standard集群 CCE Turbo集群

鲲鹏内存优化型

鲲鹏内存优化型弹性云服务器搭载鲲鹏920处理器及25GE智能高速网卡，提供最大480GiB基于DDR4的内存实例和高性能网络，擅长处理大型内存数据集和高网络场景。

表 3-36 鲲鹏内存优化型实例特点

规格名称	计算	网络	支持集群类型
鲲鹏内存优化型 kM1	• CPU/内存配比：1:8 • vCPU数量范围：2-60 • 处理器：鲲鹏920处理器 • 基频：2.6GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：400万PPS • 最大内网带宽：30Gbps	CCE Standard集群

表 3-37 kM1 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡多队列数	网卡个数上限	虚拟化类型
km1.large.8	2	16	3/0.8	30	2	2	KVM
km1.xlarge.8	4	32	5/1.5	50	2	3	KVM
km1.2xlarge.8	8	64	7/3	80	4	4	KVM
km1.3xlarge.8	12	96	9/4.5	110	4	5	KVM
km1.4xlarge.8	16	128	12/6	140	4	6	KVM
km1.6xlarge.8	24	192	15/8	200	8	6	KVM
km1.8xlarge.8	32	256	18/10	260	8	6	KVM
km1.12xlarge.8	48	384	25/16	350	16	8	KVM
km1.15xlarge.8	60	480	30/20	400	16	8	KVM

鲲鹏超高 I/O 型

鲲鹏超高I/O型弹性云服务器搭载鲲鹏920处理器及25GE智能高速网卡，提供最大480GiB基于DDR4的内存实例和高性能网络，擅长处理大型内存数据集和高网络场景。

表 3-38 鲲鹏超高 I/O 型实例特点

规格名称	计算	网络	支持集群类型
鲲鹏超高I/O型kI1	• CPU/内存配比：1:4 • vCPU数量范围：8-64 • 处理器：鲲鹏920处理器 • 基频：2.6GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：400万PPS • 最大内网带宽：30Gbps	CCE Standard集群

表 3-39 kI1 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网卡个数上限	网卡多队列数	本地盘	虚拟化类型
ki1.2xlarge.4	8	32	7/3	80	4	4	1 × 3200GiB	KVM
ki1.4xlarge.4	16	64	12/6	140	6	4	2 × 3200GiB	KVM
ki1.6xlarge.4	24	96	15/8.5	200	6	8	3 × 3200GiB	KVM
ki1.8xlarge.4	32	128	18/10	260	6	8	4 × 3200GiB	KVM
ki1.12xlarge.4	48	192	25/16	350	6	16	6 × 3200GiB	KVM
ki1.16xlarge.4	64	228	30/20	400	6	16	8 × 3200GiB	KVM

超高 I/O 型

超高I/O型弹性云服务器使用高性能NVMe SSD本地磁盘，提供高存储IOPS以及低读写时延。

表 3-40 超高 I/O 型实例特点

规格名称	计算	网络	支持集群类型
超高I/O型Ir7	• CPU/内存配比：1:4 • vCPU数量范围：2-64 • 处理器：第三代英特尔® 至强® 铂金® 可扩展处理器 • 基频/睿频：3.0GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：600万PPS • 最大内网带宽：40Gbps	CCE Standard集群 CCE Turbo集群
超高I/O型I7	• CPU/内存配比：1:4 • vCPU数量范围：8-96 • 处理器：第三代英特尔® 至强® 铂金® 可扩展处理器 • 基频/睿频：3.0GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：800万PPS • 最大内网带宽：44Gbps	CCE Standard集群 CCE Turbo集群
超高I/O型Ir3	• CPU/内存配比：1:4 • vCPU数量范围：2-32 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：550万PPS • 最大内网带宽：30Gbps	CCE Standard集群
超高I/O型I3	• CPU/内存配比：1:8 • vCPU数量范围：8-64 • 处理器：英特尔® 至强® 可扩展处理器 • 基频/睿频：3.0GHz/3.4GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：500万PPS • 最大内网带宽：25Gbps	CCE Standard集群

表 3-41 I7 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	本地盘（GiB）	虚拟化类型
i7.2xlarge.4	8	32	10/3	120	100	4	4	64	1 × 1600GiB NVMe	KVM
i7.4xlarge.4	16	64	15/6	200	150	4	6	96	2 × 1600GiB NVMe	KVM
i7.8xlarge.4	32	128	25/12	400	300	8	8	192	4 × 1600GiB NVMe	KVM
i7.12xlarge.4	48	192	30/18	500	400	16	8	256	6 × 1600GiB NVMe	KVM
i7.16xlarge.4	64	256	35/24	600	500	16	8	256	8 × 1600GiB NVMe	KVM
i7.24xlarge.4	96	384	44/36	800	800	32	8	256	12 × 1600GiB NVMe	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽（Gbps）	最大收发包能力（万PPS）	网络连接数（万）	网卡多队列数	网卡个数上限	辅助网卡个数上限	本地盘（GiB）	虚拟化类型
i7.2xlarge.8	8	64	10/3	120	100	4	4	64	1 × 1600GiB NVMe	KVM
i7.4xlarge.8	16	128	15/6	200	150	4	6	96	2 × 1600GiB NVMe	KVM
i7.8xlarge.8	32	256	25/12	400	300	8	8	192	4 × 1600GiB NVMe	KVM
i7.12xlarge.8	48	384	30/18	500	400	16	8	256	6 × 1600GiB NVMe	KVM
i7.16xlarge.8	64	512	35/24	600	500	16	8	256	8 × 1600GiB NVMe	KVM
i7.24xlarge.8	96	768	44/36	800	800	32	8	256	12 × 1600GiB NVMe	KVM

Flexus 云服务器 X

Flexus云服务器X实例是新一代面向中小企业和开发者打造的柔性算力云服务器，支持灵活自定义vCPU和内存配比，您可基于业务资源需要选择合适规格，节省资源开销。

表 3-42 Flexus 云服务器 X 实例特点

规格名称	计算	支持集群类型
Flexus云服务器X实例x1	• CPU/内存配比：X实例支持灵活自定义vCPU和内存配比，您可基于业务资源需要选择合适规格，节省资源开销。 • vCPU/内存范围：vCPU最大范围为1~16，内存最大范围为1GiB~128GiB。 • 处理器：第三代英特尔® 至强® 可扩展处理器。 • 基频/睿频：2.8GHz/3.5GHz。	CCE Standard集群 CCE Turbo集群
Flexus云服务器X实例x1e	• CPU/内存配比：X实例支持灵活自定义vCPU和内存配比，您可基于业务资源需要选择合适规格，节省资源开销。 • vCPU/内存范围：vCPU最大范围为2~32，内存最大范围为2GiB~256GiB。 • 基频/睿频：2.45GHz/3.5GHz。	CCE Standard集群 CCE Turbo集群

表 3-43 x1 性能规格

vCPU	内网基准/最大带宽（Gbit/s）	内网最大收发包（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
2	0.2/2	30	2	2	8
4	0.4/3	50	2	2	16
6	0.6/4	60	2	2	24
8	0.8/6	80	2	2	32
12	1.2/8	90	4	3	48
16	1.6/12	100	4	3	64

表 3-44 x1e 性能规格

vCPU	内网基准/最大带宽（Gbit/s）	内网最大收发包（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
2	1/2	40	2	2	16

vCPU	内网基准/最大带宽（Gbit/s）	内网最大收发包（万PPS）	网卡多队列数	网卡个数上限	辅助网卡个数上限
4	1.5/3	60	2	3	32
8	2.5/6	100	4	4	64
12	4/8	150	4	6	96
16	5/12	200	8	8	128
20	5/13	220	8	8	128
24	6/14	250	8	8	192
28	6/15	280	8	8	192
32	8/16	300	16	8	256

AI 加速型

AI加速型弹性云服务器是是专门为AI业务提供加速服务的云服务器。基于Ascend芯片低功耗、高算力特性，能效比大幅提升，适用于AI推理场景和视频编解码场景。

AI加速型云服务器包括kAi系列和Ai系列两类。其中：

- kAi系列：ARM架构，处理器为鲲鹏920系列。
- Ai系列：X86架构，处理器为Intel至强系列。

表 3-45 AI 加速型实例特点

规格名称	计算	网络	支持集群类型
AI加速型 kAi1s	• CPU/内存配比：1:1/1:2 • vCPU数量范围：4-48 • 处理器：鲲鹏920处理器 • 基频：2.6GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：400万PPS • 最大内网带宽：20Gbps	CCE Standard集群
AI加速型 Ai1s	• CPU/内存配比：1:4/1:2 • vCPU数量范围：2-32 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：200万PPS • 最大内网带宽：25Gbps	CCE Standard集群

规格名称	计算	网络	支持集群类型
AI加速型 Ai2	• CPU/内存配比：1:4 • vCPU数量范围：4-96 • 处理器：第三代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/ 3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：850万 PPS • 最大内网带宽：40Gbps	CCE Standard 集群 CCE Turbo集群
AI加速型 Ai1	• CPU/内存配比：1:4 • vCPU数量范围：2-32 • 处理器：第二代英特尔® 至强® 可扩展处理器 • 基频/睿频：2.6GHz/ 3.5GHz	• 超高网络收发包能力 • 实例网络性能与计算规格对应，规格越高网络性能越强 • 最大网络收发包：200万 PPS • 最大内网带宽：25Gbps	CCE Standard 集群

表 3-46 kAi1s 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	网卡多队列数	网卡个数上限	Ascend 310	虚拟化类型
kai1s.xlarge.1	4	4	3/0.8	20	2	2	1	KVM
kai1s.2xlarge.1	8	8	4/1.5	40	2	3	2	KVM
kai1s.4xlarge.1	16	16	6/3	80	4	4	4	KVM
kai1s.3xlarge.2	12	24	8/4	100	4	4	4	KVM
kai1s.4xlarge.2	16	32	10/6	140	4	5	6	KVM
kai1s.6xlarge.2	24	48	12/8	200	8	6	8	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	网卡多队列数	网卡个数上限	Ascend 310	虚拟化类型
kai1s.9xlarge.2	36	72	16/12	280	8	6	12	KVM
kai1s.12xlarge.2	48	96	20/16	400	16	6	12	KVM

表 3-47 Ai1s 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	Ascend 310	Ascend RAM（GiB）	网卡多队列数	网卡个数上限	虚拟化类型
ai1s.large.4	2	8	4/1.3	20	1	8	2	2	KVM
ai1s.xlarge.4	4	16	6/2	35	2	16	2	3	KVM
ai1s.2xlarge.4	8	32	10/4	50	4	32	4	4	KVM
ai1s.4xlarge.4	16	64	15/8	100	8	64	8	8	KVM
ai1s.8xlarge.4	32	128	25/15	200	16	128	8	8	KVM

表 3-48 Ai2 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	Ascend芯片	Ascend RAM（GiB）	网卡多队列数	网卡个数上限	虚拟化类型
ai2.xlarge.4	4	16	8/1.6	80	1	24	3	3	KVM
ai2.2xlarge.4	8	32	15/3	150	1	24	4	4	KVM
ai2.4xlarge.4	16	64	20/6	280	1	24	8	8	KVM
ai2.8xlarge.4	32	128	30/12	550	2	48	8	8	KVM
ai2.16xlarge.4	64	256	36/24	800	4	96	16	8	KVM
ai2.24xlarge.4	96	384	40/36	850	6	144	32	8	KVM

表 3-49 Ai1 型弹性云服务器的规格

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	Ascend 310	Ascend RAM（GiB）	网卡多队列数	网卡个数上限	虚拟化类型
ai1.large.4	2	8	4/1.3	20	1	8	2	2	KVM
ai1.xlarge.4	4	16	6/2	35	2	16	2	3	KVM
ai1.2xlarge.4	8	32	10/4	50	4	32	4	4	KVM

规格名称	vCPU	内存（GiB）	最大带宽/基准带宽	最大收发包能力（万/PPS）	Ascend 310	Ascend RAM（GiB）	网卡多队列数	网卡个数上限	虚拟化类型
ai1.4xlarge.4	16	64	15/8	100	8	64	8	8	KVM
ai1.8xlarge.4	32	128	25/15	200	16	128	8	8	KVM

3.5 创建节点

前提条件

- 已创建至少一个集群。
- 您需要新建一个密钥对，用于远程登录节点时的身份认证。
  若使用密码登录节点，请跳过此操作。创建方法请参见**创建密钥对**。

约束与限制

- 创建节点过程中依赖OBS等周边服务，因此节点所在子网的DNS配置不可修改。
- 集群开启IPv4/IPv6双栈时，所选节点子网不允许开启DHCP无限租约。

注意事项

- 为了保证节点的稳定性，CCE集群节点上会根据节点的规格预留一部分资源给Kubernetes的相关组件（kubelet、kube-proxy以及docker等）和Kubernetes系统资源，使该节点可作为您的集群的一部分。 因此，您的节点资源总量与节点在Kubernetes中的可分配资源之间会存在差异。节点的规格越大，在节点上部署的容器可能会越多，所以Kubernetes自身需预留更多的资源，详情请参见**节点预留资源策略说明**。
- 节点的网络（如虚机网络、容器网络等）均被CCE接管，请勿自行添加删除网卡、修改路由和IP地址。若自行修改可能导致服务不可用。例如，节点上名为的gw_11cbf51a@eth0网卡为容器网络网关，不可修改。
- 集群中通过“按需计费”模式购买的节点，在CCE“节点管理”中进行删除操作后将会直接被删除；通过“包年/包月”模式购买的节点不能直接删除，请在右上角单击“费用”，选择“订单管理 > 退订与退换货”执行资源退订操作。

操作步骤

集群创建完成后，您可以在集群中创建节点。

步骤1 登录**CCE控制台**。

步骤2 在左侧导航栏中选择“集群管理”，单击要创建节点的集群进入集群控制台。

步骤3 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签并单击右上角的“创建节点”，在节点配置步骤中参照如下表格设置节点参数。

节点配置：

配置节点云服务器的规格与操作系统，为节点上的容器应用提供基本运行环境。

表 3-50 节点配置参数

参数	参数说明
计费模式	支持以下计费方式： • 包年包月 包年包月需要选择购买时长，还可以勾选自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。 • 按需计费 按资源的实际使用时长计费，可以随时开通/删除资源。 • 竞价计费 竞价计费是后付费模式，相对于按需计费模式，以更低的折扣按实际使用时长计费。详情请参见**竞价计费型实例**。 **说明** – 如果创建竞价实例时同时购买了数据盘和弹性公网IP，数据盘和弹性公网IP会在竞价实例释放时随实例释放。如果给已经创建完成的竞价实例挂载数据盘和弹性公网IP，则需要在删除竞价实例后自行释放这些资源。 – 竞价实例不支持重置、纳管、移除、迁移、转包周期、集群重置升级。如果要对集群进行重置升级，需要先删除竞价节点，将竞价节点池实例数设为0。 – ECS可能会因为用户报价小于市场价、资源不足等原因主动释放竞价实例。建议在集群中安装最新版本的npd插件，npd插件会在竞价实例被ECS释放前5分钟收到通知，产生ReceivedReclaimNodeNotification事件，并给节点加污点**node-problem-controller.cce.io/SpotPriceNodeReclaimNotification: NoExecute**，驱逐节点上的Pod，使Pod能在节点被删除前迁移到其他节点。
可用区	节点云服务器所在的可用区，集群下节点创建在不同可用区下可以提高可靠性。创建后不可修改。 建议您选择“随机分配”，可根据选择的节点规格随机分配一个可以使用的可用区。 可用区是在同一区域下，电力、网络隔离的物理区域，可用区之间内网互通，不同可用区之间物理隔离。如果您需要提高工作负载的高可靠性，建议您将云服务器创建在不同的可用区。

参数	参数说明
节点类型	请根据不同的业务诉求选择节点类型，“节点规格”列表中将自动为您筛选该类型下可部署容器服务的规格，供您进一步选择。 CCE Standard集群支持以下类型： ● 弹性云服务器-虚拟机：使用虚拟化技术的弹性云服务器作为集群节点。 ● 弹性云服务器-物理机：使用擎天架构的裸金属服务器作为集群节点。 ● 裸金属服务器：使用传统裸金属服务器作为集群节点。选择数据盘时支持使用裸金属服务器自带的本地盘。 CCE Turbo集群支持以下类型： ● 弹性云服务器-虚拟机：使用虚拟化的弹性云服务器作为集群节点。CCE Turbo集群仅支持可添加多张弹性网卡的机型，请根据控制台页面展示规格进行选择。 ● 弹性云服务器-物理机：使用擎天架构的裸金属服务器作为集群节点。
节点规格	请根据业务需求选择相应的节点规格，节点规格要求CPU为2核及以上、内存为4GiB及以上。 不同的可用区可选择的节点规格类型可能不同，请根据实际情况选择。 **说明** CCE集群支持添加鲲鹏（ARM）节点，请以创建节点页面实际展示规格为准。
容器引擎	CCE支持Docker和Containerd容器引擎，不同的集群类型、集群版本、操作系统可能导致支持的容器引擎类型不同，请根据控制台呈现进行选择。具体场景请参见**节点操作系统与容器引擎对应关系**。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 ● 公共镜像：请选择节点对应的操作系统。 ● 私有镜像：支持使用私有镜像，私有镜像制作方法具体请参见**制作CCE节点自定义镜像**。 **说明** 由于业务容器运行时共享节点的内核及底层调用，为保证兼容性，建议节点的操作系统选择与最终业务容器镜像相同或接近的Linux发行版本。
节点名称	节点云服务器使用的名称，批量创建时将作为云服务器名称的前缀。 系统会默认生成名称，支持修改。 节点名称长度范围为1-56个字符，以小写字母开头，支持小写字母、数字、中划线(-)、点(.)，不能以中划线(-)或点(.)结尾，点(.)前后必须为小写字母或数字。

参数	参数说明
企业项目	该参数仅对开通企业项目的企业客户账号显示，且集群版本要求v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上。 选择某个企业项目后，节点将会创建在该企业项目下。您可以通过企业项目服务（EPS）管理集群及其他资源（节点、ELB、以及节点的安全组等）。了解更多企业项目相关信息，请查看**企业管理**。
登录方式	• **密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 • **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。 • **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持） 保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。

存储配置：

配置节点云服务器上的存储资源，方便节点上的容器软件与容器应用使用。请根据实际场景设置磁盘类型及大小。关于云硬盘类型的详细介绍请参见**磁盘类型及性能介绍**。

表 3-51 存储配置参数

参数	参数说明
系统盘	节点云服务器使用的系统盘，供操作系统使用。您可以设置系统盘的规格为40GiB-1024GiB之间的数值，缺省值为50GiB。 **说明** 通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。仅v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群中支持选择使用通用型SSD V2类型的云硬盘。 系统盘加密：系统盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**仅“弹性云服务器-虚拟机”类型支持系统盘加密，且仅部分Region支持此选项，具体请以界面为准。** ● 默认不加密。 ● 选择“加密-从密钥中选择”后，可选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 ● 选择“加密-输入KMS密钥ID”后，您需要输入的KMS密钥ID（包含他人共享的KMS密钥），且该密钥必须位于当前Region下。
系统组件存储	选择系统组件的存储位置： ● 数据盘：必须添加一块默认数据盘，供容器运行时和Kubelet组件使用，您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。该数据盘不能被删除卸载，否则会导致节点不可用。 ● 系统盘：CCE将下载的镜像、容器的临时存储、容器的stdout标准输出日志等资源都存储在系统盘中，过多占用系统盘可能影响节点运行稳定性。 **说明** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中支持选择系统组件的存储位置，且配套使用**CCE节点故障检测**插件时需安装1.19.2及以上版本的插件。

参数	参数说明
数据盘	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，**至少需要一块默认数据盘**，供容器运行时和Kubelet组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** ● 默认数据盘：供容器运行时和Kubelet组件使用。您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。 ● 其他普通数据盘，您可以设置数据盘的规格为10GiB-32768GiB之间的数值，缺省值为100GiB。 v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中，如果“系统组件存储”选择“系统盘”，则可以不添加默认数据盘。所有数据盘均为普通数据盘，您可以设置数据盘的规格为10GiB-32768GiB之间的数值，缺省值为100GiB。 **说明** ● 节点规格为“磁盘增强型”或“超高I/O型”时，有一块数据盘可以是本地盘。 ● 本地磁盘实例有宕机风险，不保证数据可靠性，建议您使用云硬盘存储您的业务数据。 ● 通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。仅v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群中支持选择使用通用型SSD V2类型的云硬盘。 **高级配置** 单击后方的“展开高级设置”可进行如下设置： ● 数据盘空间分配：对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 ● 数据盘加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**“裸金属服务器”类型节点不支持数据盘加密，且仅部分Region支持此选项，具体请以界面为准。** – 默认不加密。 – 选择“加密-从密钥中选择”后，可选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 – 选择“加密-输入KMS密钥ID”后，您需要输入的KMS密钥ID（包含他人共享的KMS密钥），且该密钥必须位于当前Region下。 **增加数据盘** 弹性云服务器最多可以添加16块，裸金属服务器最多可以添加10块。默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，选择如下配置。 ● 默认：默认情况直接创建为裸盘，不做任何处理。 ● 挂载到指定目录：将数据盘挂载到指定目录。

参数	参数说明
	• 作为持久存储卷：适用于对PV有性能要求的场景。持久存储卷的节点会添加上node.kubernetes.io/local-storage-persistent标签，取值为linear或striped。 • 作为临时存储卷：适用于对EmptyDir有性能要求的场景。 **说明** • 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。 • 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=1.2.29。 **本地持久卷**和**本地临时卷**支持如下两种写入模式。 • 线性（linear）：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。 • 条带化（striped）：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。多块存储卷才能选择条带化。

网络配置：

配置节点云服务器的网络资源，用于访问节点和容器应用。

表 3-52 网络配置参数

参数	参数说明
虚拟私有云/节点子网	节点子网默认使用创建集群时的子网配置，也可以选择其他子网。
节点IP	支持指定节点IP地址。默认随机分配。
弹性公网IP	未绑定弹性公网IP的云服务器无法直接访问外网，无法直接对外进行互相通信。 默认为“暂不使用”。支持“使用已有”和“自动创建”。

高级配置：

节点能力增强，可在此配置节点的标签、污点、启动命令等功能。

表 3-53 高级配置参数

参数	参数说明
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。最多可以添加8条资源标签。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动帮您创建CCE-Dynamic-Provisioning-Node=节点id的标签。
K8S标签（Labels）	设置附加到Kubernetes对象（比如Pod）上的键值对，填写键值对后，单击“确认添加”。最多可以添加20条标签。 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。详细请参见**Labels and Selectors**。
K8S污点（Taints）	默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点，每条污点包含以下3个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符；另外可以使用DNS子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符。 • Effect：只可选NoSchedule，PreferNoSchedule或NoExecute。 污点的使用请参见**管理节点污点**。 **说明** 对于1.19及以下版本集群，有可能会出现污点打上之前负载已经调度到节点上，如果需要避免这种情况，请选择1.19及以上集群。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。 节点最多能创建多少个Pod还受其他因素影响，具体请参见**节点可创建的最大Pod数量说明**。
云服务器组	云服务器组是对云服务器的一种逻辑划分，同一云服务器组中的云服务器遵从同一策略。 反亲和性策略：同一云服务器组中的云服务器分散地创建在不同主机上，提高业务的可靠性。 选择已创建的云服务器组，或单击“新建云服务器组”创建，创建完成后单击刷新按钮。
安装前执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装前执行，可能导致Kubernetes软件无法正常安装，需谨慎使用。

参数	参数说明
安装后执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。 **说明** 请不要在安装后执行脚本中使用**reboot**命令立即重启，如果需要重启，可以使用**shutdown -r 1**命令延迟1分钟重启。
委托	委托是由租户管理员在统一身份认证服务上创建的。通过委托，可以将云主机资源共享给其他账号，或委托更专业的人或团队来代为管理。 如果没有委托请单击右侧“新建委托”创建。
K8s节点名称	K8s节点名称，即节点YAML文件中的“metadata.labels.kubernetes.io/hostname”标签值，支持以下两种配置。 • 与节点私有IP保持一致：默认为节点私有IP。 • 与云服务器名称保持一致：将**节点配置**中配置的自定义云服务器名称作为K8s节点名称。由于云服务器名称可能存在重名，为避免K8s节点名称冲突，系统将在名称后自动添加五位随机字符后缀。 **须知** – 该功能在集群版本为v1.23.4-r0及以上时支持配置。 – 仅支持在创建（纳管）时将节点云服务器名称指定为K8s节点名称。创建（纳管）完成后，K8s节点名称无法修改，详情请参见**云服务器名称、节点名称与K8s节点名称说明**。 – 如您在创建（纳管）选择将云服务器名称指定为K8s节点名称，集群已有节点将仍使用私有IP作为K8s节点名称。 该场景下，存在部分K8s节点名称与私有IP不一致的情况，对于业务场景中将私有IP和K8s节点名称混用的场景，需做好适配。例如，在设置节点亲和调度时，不能将节点私有IP作为节点名称配置调度策略。 如您需要将已有节点的K8s节点名称统一为“与云服务器名称保持一致”，您可将已有节点从集群中移除，并重新纳管。执行节点移除、纳管操作前，请您充分了解节点**移除**及**纳管**可能带来的业务影响。

步骤4 完成以上配置后，您可以设置需要购买的节点数量，并单击“下一步：规格确认”，确认所设置的服务选型参数、规格和费用等信息，且确认已阅读并知晓华为云的**镜像免责声明**。

步骤5 单击“提交”，节点开始创建。

若选择购买“包年包月”的节点，请单击“去支付”，根据界面提示进行付款操作。

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

步骤6 单击“返回节点列表”，待状态为运行中，表示节点创建成功。

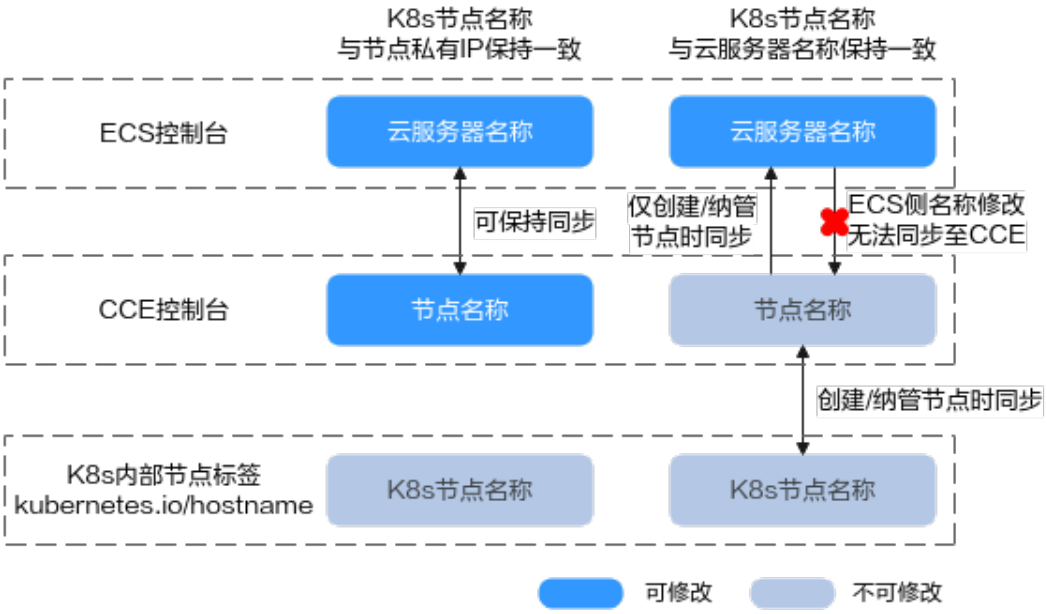
----结束

云服务器名称、节点名称与 K8s 节点名称说明

- 云服务器名称：ECS页面的云服务器名称，创建节点时支持自定义云服务器名称。
- 节点名称：CCE页面的节点名称，可与ECS页面的云服务器名称同步。但在指定K8s节点名称后，ECS云服务器名称的修改将无法被同步至节点名称。
- K8s节点名称：即节点YAML文件中的“metadata.labels.kubernetes.io/hostname”标签值。仅支持在创建（纳管）时将节点云服务器名称指定为K8s节点名称。创建（纳管）完成后，K8s节点名称无法修改。

云服务器名称、节点名称与K8s节点名称之间的同步关系如图3-1所示。

图 3-1 节点名称关系示意图



相关操作

创建节点注入脚本最佳实践

3.6 纳管节点

操作场景

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

须知

- 纳管时，如果您选择将所选弹性云服务器的操作系统重置为CCE提供的标准公共镜像，您需要重新设置密码或密钥，原有的密码或密钥将会失效。
- 所选弹性云服务器挂载的系统盘、数据盘都会在纳管时清理LVM信息，包括卷组（VG）、逻辑卷（LV）、物理卷（PV），请确保信息已备份。
- 纳管过程中，请勿在弹性云服务器控制台对所选虚拟机做任何操作。

约束与限制

- 纳管节点支持ECS（弹性云服务器）节点、BMS（裸金属服务器）节点、DeH（专属主机）节点。

前提条件

待纳管的云服务器需要满足以下前提条件：

- 待纳管节点必须状态为“运行中”，未被其他集群所使用，且不携带 CCE 专属节点标签CCE-Dynamic-Provisioning-Node。
- 待纳管节点需与集群在同一虚拟私有云内（若集群版本低于1.13.10，纳管节点还需要与CCE集群在同一子网内）。
- 待纳管节点需挂载数据盘，可使用本地盘（磁盘增强型实例）或至少挂载一块20GiB及以上的数据盘，且不存在10GiB以下的数据盘。关于节点挂载数据盘的操作说明，请参考**新增磁盘**。
- 待纳管节点规格要求：CPU必须2核及以上，内存必须4GiB及以上，网卡有且仅能有一个。
- 如果使用了企业项目，则待纳管节点需要和集群在同一企业项目下，不然在纳管时会识别不到资源，导致无法纳管。
- 批量纳管仅支持添加相同数据盘配置的云服务器。
- 集群开启IPv6后，只支持纳管所在的子网开启了IPv6功能的节点；集群未开启IPv6，只支持纳管所在的子网未开启IPv6功能的节点。
- CCE Turbo集群要求节点支持Sub-ENI或可以绑定至少16张ENI网卡，具体规格请参见创建节点时控制台上可以选择的节点规格。
- 纳管节点时已分区的数据盘会被忽略，您需要保证节点至少有一个未分区且符合规格的数据盘。

操作步骤

步骤1 登录CCE控制台，进入要纳管节点的集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签并单击右上角的“纳管节点”。

步骤3 配置节点参数。

节点配置

表 3-54 节点配置参数

参数	参数说明
选择添加节点池	• 默认节点池DefaultPool：您可以将满足纳管条件的节点添加至集群的默认节点池。 • 自定义节点池：您只需选择满足纳管条件的节点，该节点将使用自定义节点池的基础、网络、高级配置，无需继续填写其他参数，详情请参见**纳管节点至节点池**。
节点规格	单击添加已有云服务器，选择要纳管的服务器。 可以选择多台云服务器批量纳管，但批量纳管仅支持添加相同规格、相同可用区、相同数据盘配置的云服务器。 如果云服务器有多块数据盘，需要选择其中一块作为供容器运行时和Kubelet组件使用。
容器引擎	CCE支持Docker和Containerd容器引擎，不同的集群类型、集群版本、操作系统可能导致支持的容器引擎类型不同，请根据控制台呈现进行选择。具体场景请参见**节点操作系统与容器引擎对应关系**。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 • 公共镜像：请选择节点对应的操作系统。 • 私有镜像：支持使用私有镜像，私有镜像制作方法具体请参见**制作CCE节点自定义镜像**。 **说明** 由于业务容器运行时共享节点的内核及底层调用，为保证兼容性，建议节点的操作系统选择与最终业务容器镜像相同或接近的Linux发行版本。
登录方式	• **密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 • **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。 • **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持） 保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。

存储配置

配置节点云服务器上的存储资源，方便节点上的容器软件与容器应用使用。

表 3-55 存储配置参数

参数	参数说明
系统盘	直接使用云服务器的系统盘。
系统组件存储	选择系统组件的存储位置： • 数据盘：必须添加一块默认数据盘，供容器运行时和Kubelet组件使用，您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。该数据盘不能被删除卸载，否则会导致节点不可用。 • 系统盘：CCE将下载的镜像、容器的临时存储、容器的stdout标准输出日志等资源都存储在系统盘中，过多占用系统盘可能影响节点运行稳定性。 **说明** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中支持选择系统组件的存储位置，且配套使用**CCE节点故障检测**插件时需安装1.19.2及以上版本的插件。
数据盘	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，**至少需要一块数据盘**，供容器运行时和Kubelet组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中，如果“系统组件存储”选择“系统盘”，则可以不添加默认数据盘。 单击后方的“展开高级设置”可设置数据盘空间分配，对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 **其他数据盘**默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，将磁盘挂载到指定目录。另外还可以作为持久存储卷或临时存储卷，具体使用请参见**本地持久卷**和**本地临时卷**。

高级配置

表 3-56 高级配置参数

参数	参数说明
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。最多可以添加8条资源标签。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动帮您创建CCE-Dynamic-Provisioning-Node=节点id的标签。

参数	参数说明
K8S标签（Labels）	单击“添加标签”可以设置附加到Kubernetes 对象（比如Pods）上的键值对，最多可以添加20条标签。 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。详细请参见**Labels and Selectors**。
K8S污点（Taints）	默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点，每条污点包含以下3个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符；另外可以使用DNS子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符。 • Effect：只可选NoSchedule，PreferNoSchedule或NoExecute。 **须知** • 污点配置时需要配合Pod的toleration使用，否则可能导致扩容失败或者Pod无法调度到扩容节点。 • 节点池创建后可单击列表项的“编辑”修改配置，修改后将同步到节点池下的已有节点。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。
安装前执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装前执行，可能导致Kubernetes软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。

步骤4 单击“下一步：规格确认”，确认已阅读并知晓华为云的**镜像免责声明**，并单击“提交”。

----结束

3.7 登录节点

前提条件

- 使用SSH方式登录时，请确认节点安全组已放通SSH端口（默认为22）。详情请参见**配置安全组规则**。

- 通过公网使用SSH方式登录时要求该节点（弹性云服务器 ECS）已绑定弹性公网IP。
- 只有运行中的弹性云服务器才允许用户登录。
- Linux操作系统用户名为root。

登录方式

登录节点（弹性云服务器 ECS）的方式有如下两种：

- **管理控制台远程登录（VNC方式）**

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

  详细操作请参考：**Linux云服务器远程登录（VNC方式）**。

- **SSH方式登录**

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

  说明

  - SSH方式登录包括**SSH密钥**和**SSH密码**两种方式。详细操作请参考**SSH密钥方式登录**、**SSH密码方式登录**。
  - 本地使用Windows操作系统登录Linux节点时，输入的镜像用户名（Auto-login username）为：root。
  - CCE控制台不提供针对节点的操作系统升级，也不建议您通过yum方式进行升级，如果您在节点上通过yum update升级了操作系统，会导致容器网络的组件不可用。手动恢复方式请参见**如何解决yum update升级操作系统导致容器网络不可用问题？**。

表 3-57 Linux 云服务器登录方式一览

是否绑定EIP	本地设备操作系统	连接方法
是	Windows	使用PuTTY、Xshell等远程登录工具。 • SSH密码方式鉴权：**SSH密码方式登录** • SSH密钥方式鉴权：**SSH密钥方式登录**
是	Linux	使用命令连接。 • SSH密码方式鉴权：**SSH密码方式登录** • SSH密钥方式鉴权：**SSH密钥方式登录**
是/否	Windows或者Linux	使用管理控制台远程登录方式：**Linux云服务器远程登录（VNC方式）**。

3.8 管理节点

3.8.1 管理节点标签

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

节点标签使用场景

节点标签的主要使用场景有两类。

- 节点管理：通过节点标签管理节点，给节点分类。
- 工作负载与节点的亲和与反亲和：通过为节点添加标签，您可以使用节点亲和性将Pod调度到特定节点，或使用反亲和性避免将其调度到特定节点，具体操作详情请参见**设置节点亲和调度（nodeAffinity）**。

节点固有标签

节点创建出来会存在一些固有的标签，并且是无法删除的，这些标签的含义请参见**表3-58**。

说明

系统自动添加的节点固有标签不建议手动修改，如果手动修改值与系统值产生冲突，将以系统值为准。

表 3-58 节点固有标签

键	说明
新：topology.kubernetes.io/region 旧：failure-domain.beta.kubernetes.io/region	表示节点当前所在区域。
新：topology.kubernetes.io/zone 旧：failure-domain.beta.kubernetes.io/zone	表示节点所在区域的可用区。
新：node.kubernetes.io/baremetal 旧：failure-domain.beta.kubernetes.io/is-baremetal	表示是否为裸金属节点。 例如：false，表示非裸金属节点
node.kubernetes.io/container-engine	表示容器引擎。 例如：docker、containerd
node.kubernetes.io/instance-type	节点实例规格。
kubernetes.io/arch	节点处理器架构。

键	说明
kubernetes.io/hostname	节点名称。
kubernetes.io/os	节点操作系统类型。
node.kubernetes.io/subnetid	节点所在子网的ID。
os.architecture	表示节点处理器架构。 例如：amd64，表示AMD64位架构的处理器
os.name	节点的操作系统名称。
os.version	操作系统节点内核版本。
accelerator/huawei-npu	NPU节点标签。
accelerator	GPU节点标签。
cce.cloud.com/cce-nodepool	节点池节点专属标签。

添加/删除节点标签

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签，勾选目标节点，并单击左上方“标签与污点管理”。

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

填写需要增加/删除标签的“键”和“值”，单击“确定”。

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

图 3-2 添加节点标签

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

----结束

3.8.2 管理节点污点

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

通过控制台管理节点污点

在CCE控制台上同样可以管理节点的污点，且可以批量操作。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签，勾选目标节点，并单击左上方“标签与污点管理”。

步骤3 在弹出的窗口中，在“批量操作”下方单击“新增批量操作”，然后选择“添加/更新”或“删除”，选择“K8S 污点(Taints)”。

填写需要操作污点的“键”和“值”，选择污点的效果，单击“确定”。

图 3-3 添加污点

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

----结束

通过 kubectl 命令管理污点

节点污点是与“效果”相关联的键值对。以下是可用的效果：

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

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

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

通过describe命名和get命令可以查看到污点的配置。

```
$ kubectl describe node 192.168.10.240
Name:               192.168.10.240
...
Taints:             key1=value1:NoSchedule
...
$ kubectl get node 192.168.10.240 -oyaml
apiVersion: v1
...
spec:
  providerID: 06a5ea3a-0482-11ec-8e1a-0255ac101dc2
  taints:
  - effect: NoSchedule
    key: key1
    value: value1
...
```

去除污点可以在增加污点命令的基础上，在最后加一个“-”，如下所示。

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

一键设置节点调度策略

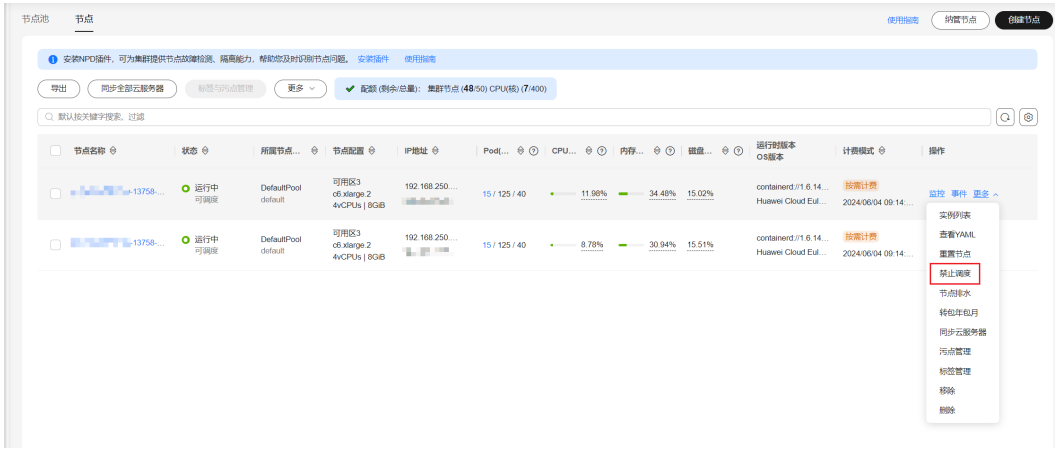
您可以通过控制台将节点设置为不可调度，系统会为该节点添加键为node.kubernetes.io/unschedulable，效果为NoSchedule的污点。节点设置为不可调度后，新的Pod将无法调度至该节点，节点上已运行的Pod则不受影响。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 在节点列表中找到目标节点，单击“更多 > 禁止调度”。

图 3-4 禁止调度



步骤4 在弹出的对话框中单击“是”，即可将节点设置为不可调度。

图 3-5 确认设置不可调度

这个操作会给节点打上污点，使用kubectl可以查看污点的内容。

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

步骤5 在CCE控制台相同位置再次设置，单击“更多 > 开启调度”。即可去除污点，将节点设置为可调度。

----结束

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。

相关操作：容忍度（Toleration）

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

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

在 Pod 中设置容忍度示例如下：

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

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

同样还可以按如下方式写，表示当节点有key1这个污点时，可以调度到节点。

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

3.8.3 重置节点

操作场景

您可以通过重置节点修改节点的配置，比如修改节点操作系统、登录方式等。

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

约束与限制

- v1.13及以上版本的CCE Standard集群、CCE Turbo集群支持重置节点。
- v1.15及以上版本的鲲鹏集群支持重置节点。

注意事项

- 重置节点功能不会重置控制节点，仅能对工作节点进行重置操作，如果重置后节点仍然不可用，请删除该节点重新创建。
- **重置节点将对节点操作系统进行重置安装，推荐您在重置节点之前为节点排水，将容器优雅驱逐至其他可用节点，同时建议在业务低峰期操作。**
- **节点重置后系统盘和数据盘将会被清空，如有重要数据请事先备份。**
- **工作节点如在ECS侧自行挂载了数据盘，重置完后会清除挂载信息，重置完成后请重新执行挂载行为，数据不会丢失。**
- 节点上的工作负载实例的IP会发生变化，但是不影响容器网络通信。
- 云硬盘必须有剩余配额。
- 操作过程中，后台会把当前节点设置为不可调度状态。

- 节点重置会清除用户单独添加的 K8S 标签和污点（通过节点池编辑功能添加的标签、污点不会丢失），可能导致与节点有绑定关系的资源（本地存储，指定调度节点的负载等）无法提供服务。
- 重置节点会导致与节点关联的**本地持久卷**类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。重置节点时使用了本地持久存储卷的Pod会从重置的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。节点重置完成后，Pod可能调度到重置好的节点上，此时Pod会一直处于creating状态，因为PVC对应的底层逻辑卷已经不存在了。

重置 DefaultPool 中的节点

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 在节点列表中选择一个或多个DefaultPool中的节点，单击“更多 > 重置节点”。

步骤4 在弹出的窗口中单击“下一步”，跳转到参数配置界面。

步骤5 配置节点参数。

计算配置

表 3-59 计算配置参数

参数	参数说明
节点规格	重置节点时不支持修改节点规格。
容器引擎	CCE支持Docker和Containerd容器引擎，不同的集群类型、集群版本、操作系统可能导致支持的容器引擎类型不同，请根据控制台呈现进行选择。具体场景请参见**节点操作系统与容器引擎对应关系**。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 • 公共镜像：请选择节点对应的操作系统。 • 私有镜像：支持使用私有镜像，私有镜像制作方法具体请参见**制作CCE节点自定义镜像**。 **说明** 由于业务容器运行时共享节点的内核及底层调用，为保证兼容性，建议节点的操作系统选择与最终业务容器镜像相同或接近的Linux发行版本。

参数	参数说明
登录方式	• **密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 • **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。 • **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持） 保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。

存储配置

配置节点云服务器上的存储资源，方便节点上的容器软件与容器应用使用。

表 3-60 存储配置参数

参数	参数说明
系统盘	直接使用云服务器的系统盘。
系统组件存储	选择系统组件的存储位置： • 数据盘：必须添加一块默认数据盘，供容器运行时和Kubelet组件使用，您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。该数据盘不能被删除卸载，否则会导致节点不可用。 • 系统盘：CCE将下载的镜像、容器的临时存储、容器的stdout标准输出日志等资源都存储在系统盘中，过多占用系统盘可能影响节点运行稳定性。 **说明** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中支持选择系统组件的存储位置，且配套使用**CCE节点故障检测**插件时需安装1.19.2及以上版本的插件。

参数	参数说明
数据盘	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，**至少需要一块数据盘**，供容器运行时和Kubelet组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中，如果“系统组件存储”选择“系统盘”，则可以不添加默认数据盘。 单击后方的“展开高级设置”可设置数据盘空间分配，对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 **其他数据盘**默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，将磁盘挂载到指定目录。另外还可以作为持久存储卷或临时存储卷，具体使用请参见**本地持久卷**和**本地临时卷**。

高级配置

表 3-61 高级配置参数

参数	参数说明
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。最多可以添加8条资源标签。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动帮您创建CCE-Dynamic-Provisioning-Node=节点id的标签。
K8S标签（Labels）	单击“添加标签”可以设置附加到Kubernetes 对象（比如Pods）上的键值对，最多可以添加20条标签。 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。详细请参见**Labels and Selectors**。

参数	参数说明
K8S污点（Taints）	默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点，每条污点包含以下3个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符；另外可以使用DNS子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符。 • Effect：只可选NoSchedule，PreferNoSchedule或NoExecute。 **须知** • 污点配置时需要配合Pod的toleration使用，否则可能导致扩容失败或者Pod无法调度到扩容节点。 • 节点池创建后可单击列表项的“编辑”修改配置，修改后将同步到节点池下的已有节点。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。
安装前执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装前执行，可能导致Kubernetes软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。

步骤6 单击“下一步：规格确认”，确认已阅读并知晓华为云的**镜像免责声明**。

步骤7 单击“提交”。

----结束

重置节点池中的节点

说明

- 重置节点池中的节点时，仅可修改节点的存储配置，其余配置将使用节点池参数。
- 重置节点会执行当前节点池中的安装前和安装后脚本，并将安全组更新为节点池的安全组配置。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 在节点列表中选择一个节点池中的节点，单击“更多 > 重置节点”。

步骤4 修改节点存储参数。

表 3-62 存储配置参数

参数	参数说明
系统盘	直接使用云服务器的系统盘。
选择默认数据盘	选择一块数据盘供容器运行时和Kubelet组件使用。
数据盘	对每块数据盘进行高级配置。 **默认数据盘**：单击后方的“展开高级设置”可设置数据盘空间分配，对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 **其他普通数据盘**：单击后方的“展开高级设置”，可选择挂载设置。 ● 默认：挂载为裸盘，不做任何设置。 ● 挂载到指定目录：将数据盘挂载到业务目录路径，不可设置为空或根目录等操作系统关键路径。 ● 作为持久存储卷：将数据盘作为持久存储卷供PVC使用，具体使用请参见**本地持久卷**。 ● 作为临时存储卷：将数据盘作为临时存储卷供PVC使用，具体使用请参见**本地临时卷**。

步骤5 单击“确定”。

----结束

批量重置节点说明

不同场景下，批量重置节点的情况不同，具体场景如下：

批量重置场景	是否支持批量重置	说明
批量重置DefaultPool中的节点	部分场景支持	当节点规格、AZ、磁盘配置相同时支持批量重置 当节点规格、AZ、磁盘配置不同时不支持批量重置
批量重置同一节点池中的节点	部分场景支持	当节点磁盘配置相同时支持批量重置 当节点磁盘配置不同时不支持批量重置
批量重置不同节点池中的节点	不支持	不同节点池的节点无法一起重置

3.8.4 移除节点

操作场景

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

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

节点移出集群后会继续开机运行，并继续产生费用。

约束限制

- 若节点在CCE集群移除后重装操作系统失败，请手动完成失败节点的操作系统重装，并在重装后登录节点执行清理脚本完成CCE组件清理，具体步骤参见**重装操作系统失败如何处理**。
- 移除节点会导致与节点关联的**本地持久卷**类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。移除节点时使用了本地持久存储卷的Pod会从移除的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。

注意事项

- 移除节点会涉及Pod迁移，可能会影响业务，建议您在移除节点之前为**节点排水**，将容器优雅驱逐至其他可用节点，同时建议在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 操作过程中，后台会把当前节点设置为不可调度状态。
- 移除节点重装操作系统后将清理原有的LVM分区，通过LVM管理的数据将会清空，请提前做好相关的数据备份。

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击节点后的“更多 > 移除”。

图 3-6 移除节点

您还可以选中多个节点一起移除，如下图所示。

图 3-7 一次移除多个节点

步骤4 在弹出的“移除节点”对话框中，配置重装操作系统需要的登录信息，单击“是”，等待完成节点移除。

移除节点后，原有节点上的工作负载实例会自动迁移至其他可用节点。

----结束

重装操作系统失败如何处理

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

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

步骤2 登录服务器，执行如下命令完成CCE组件和LVM数据的清理。

将如下脚本写入**clean.sh**文件。

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

执行如下命令。

bash clean.sh

----结束

3.8.5 同步云服务器

操作场景

集群中的每一个节点对应一台云服务器，集群节点创建成功后，您仍可以根据需求，修改云服务器的名称或变更规格。由于规格变更对业务有影响，建议一台成功完成后再对下一台进行规格变更。

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

ECS常见信息修改如下：

- 修改节点名称请参见**修改云服务器名称**。
- 当您购买的节点规格无法满足业务需要时，可参考**变更规格通用操作**变更节点规格，升级vCPU、内存。

约束与限制

- 支持同步数据：虚机状态、云服务器名称、CPU数量、Memory数量、云服务器规格、公网IP等。

  当用户节点指定了云服务器名称作为K8s节点名称时，该云服务器名称的修改将无法同步到CCE控制台。更多说明请参见**云服务器名称、节点名称与K8s节点名称说明**。
- 不支持同步数据：操作系统、镜像ID、磁盘配置。

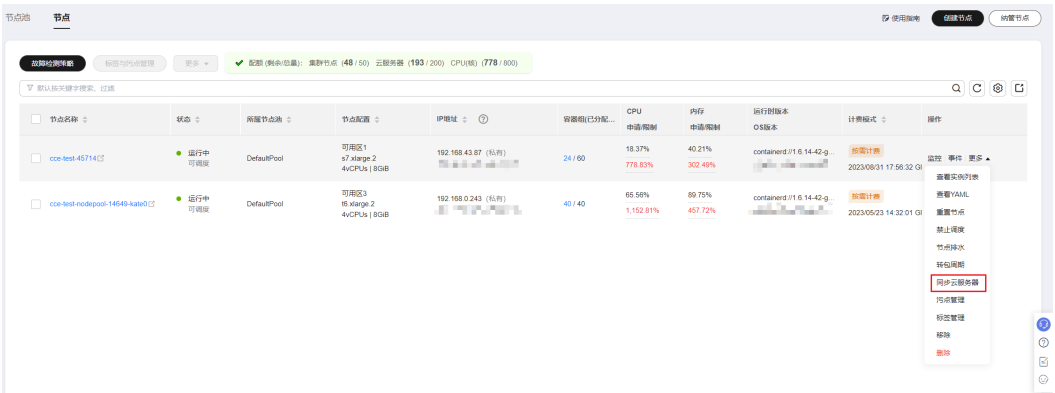
同步单个云服务器

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击节点后的“更多 > 同步云服务器”。

图 3-8 同步节点信息



完成后，页面右上角将会提示“同步云服务器任务下发成功”。

----结束

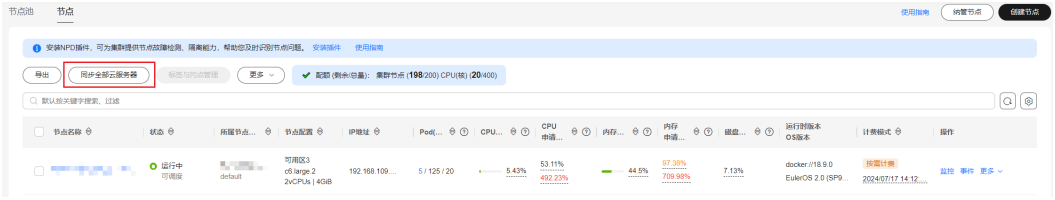
同步全部云服务器

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 单击“同步全部云服务器”，并单击“确认”。

图 3-9 同步全部云服务器



完成后，页面右上角将会提示“同步云服务器任务下发成功”。

----结束

3.8.6 节点排水

操作场景

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

在节点故障等场景下，该功能可帮助您快速排空节点，将故障节点进行隔离，原节点上被驱逐的Pod将会由工作负载controller转移到其他正常可调度的节点上。

须知

为保障排水期间业务可用性，建议为负载设置**干扰预算（Disruption Budget）**，否则Pod重新调度期间负载功能可能无法正常使用。

前提条件

- 您已创建一个集群，且集群版本满足以下要求：
  - v1.21集群：v1.21.10-r0及以上版本
  - v1.23集群：v1.23.8-r0及以上版本
  - v1.25集群：v1.25.3-r0及以上版本
  - v1.25以上版本集群
- 如果您通过IAM用户使用节点排水功能，至少需要具有以下一项权限，详情请参见**命名空间权限（Kubernetes RBAC授权）**。
  - cluster-admin（管理员权限）：对全部命名空间下所有资源的读写权限。
  - drainage-editor：节点排水操作权限，可执行节点排水。
  - drainage-viewer：节点排水只读权限，仅可查看节点排水状态，无法执行节点排水。

节点排水规则说明

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

Pod筛选条件	使用强制排水	不使用强制排水
Pod的**status.phase**字段为Succeeded或Failed	删除	删除
Pod不受工作负载controller管理	删除	放弃排水
Pod由DaemonSet管理	忽略	放弃排水 **说明** v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上集群版本，由DaemonSet管理的Pod，在不使用强制排水时的默认操作为忽略。
Pod中挂载了emptyDir类型的volume	驱逐	放弃排水
由kubelet直接管理的**静态Pod**	忽略	忽略

说明

节点排水过程中可能会对Pod执行的操作如下：

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

节点排水操作

您可以参考以下方式进行节点排水。

通过控制台操作

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击节点后的“更多 > 节点排水”。

步骤4 在弹出的“节点排水”窗口中，进行排水设置。

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

步骤5 单击“确定”，等待完成节点排水。

----结束

通过 kubectl 命令行操作

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 编辑Drainage资源的YAML。

Drainage-test.yaml示例如下：

```
apiVersion: node.cce.io/v1
kind: Drainage
metadata:
  name: 192.168.1.67-1721616409999   #Drainage资源名称
spec:
  nodeName: 192.168.1.67     #待排水节点的K8s名称，可以使用kubectl get node命令查询
  force: true
  timeout: 0
```

- nodeName：表示待排水的节点，参数值为Kubernetes中的节点名称，而不是控制台上的节点名称。

  Kubernetes中的节点名称可以使用**kubectl get node**命令查询。
- force：是否使用强制排水。true表示使用强制排水，false表示不使用强制排水。
- timeout：超时时间，单位为秒。超过设定的时间后排水任务会自动失败，0表示不设置超时时间。

步骤3 创建Drainage资源。

```
kubectl create -f Drainage-test.yaml
```

回显如下，表示Drainage资源创建成功：

```
drainage.node.cce.io/192.168.1.67-1721616409999 created
```

步骤4 查看结果。

```
kubectl get drainages 192.168.1.67-1721616409999 -o yaml
```

回显如下，如果phase参数为Successded则为成功。

```
apiVersion: node.cce.io/v1
kind: Drainage
metadata:
  creationTimestamp: "2024-07-22T03:12:56Z"
  generation: 1
  name: 192.168.1.67-1721616409999
  resourceVersion: "2683143"
  uid: 3ec131e4-0505-4c88-8255-ef9d0eb02712
spec:
  force: true
  nodeName: 192.168.1.67
  timeout: 0
status:
  conditions:
  - lastTransitionTime: "2024-07-22T03:12:56Z"
    message: start to drain node
    reason: Started
    status: "True"
    type: Started
  - lastTransitionTime: "2024-07-22T03:13:26Z"
    message: node has been drained
    reason: Succeeded
    status: "True"
    type: Finished
  phase: Succeeded
```

----结束

通过 API 操作

步骤1 获取集群所在区域的Token，获取方式请参见**获取Token**。

步骤2 根据接口格式确定节点排水接口URL。

节点排水接口的URL为：

```
https://{clusterid}.Endpoint/apis/node.cce.io/v1/drainages
```

- **{clusterid}**：集群ID，可通过CCE控制台的“总览”页面查询。
- **Endpoint**：集群所在区域的终端节点。

  对应的值请参见**地区和终端节点**。

  例如CCE服务在“亚太-新加坡”区域的Endpoint为“cce.ap-southeast-3.myhuaweicloud.com”

步骤3 使用**POST**请求方法，并设置请求Header参数。

```
curl --location --request POST 'https://{clusterid}.Endpoint/apis/node.cce.io/v1/drainages' \
--header 'Content-Type: application/json' \
--header 'X-Auth-Token: MIIWvw******' \
--data @Drainage.json
```

请求中包含的Header参数如下：

表 3-63 请求 Header 参数

参数	是否必选	参数类型	描述
Content-Type	是	String	消息体的类型（格式），例如application/json
X-Auth-Token	是	String	使用Token调用接口，Token的获取方式请参见**获取Token**。

其中Drainage.json为当前路径下的本地文件，内容如下：

```
{
   "apiVersion": "node.cce.io/v1",
   "kind": "Drainage",
   "metadata": {
      "name": "192.168.1.67-1721616404940"
   },
   "spec": {
      "nodeName": "192.168.1.67",
      "force": true,
      "timeout": 0
   }
}
```

- nodeName：表示待排水的节点，参数值为Kubernetes中的节点名称，而不是控制台上的节点名称。

  Kubernetes中的节点名称可以使用**kubectl get node**命令查询。
- force：是否使用强制排水。true表示使用强制排水，false表示不使用强制排水。
- timeout：超时时间，单位为秒。超过设定的时间后排水任务会自动失败，0表示不设置超时时间。

----结束

取消节点排水

如果在排水过程中需要取消排水，可参考以下方式。

说明

v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群中，节点排水支持取消。

该操作将中止节点上的排水流程，但已经从这些节点上迁移的工作负载不会自动迁移回来。

通过控制台操作

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到处于“排水中”状态的节点，单击“取消排水”。

步骤4 在确认框中单击“确定”，节点变成“已取消排水”状态，您可以单击“开启调度”，将节点恢复可调度状态。

----结束

通过 kubectl 命令行操作

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 查询Drainage资源。

```
kubectl get drainages
```

回显如下：

```
NAME                        AGE
192.168.1.67-1721616409999   13s
```

步骤3 取消排水。

```
kubectl annotate drainages 192.168.1.67-1721616409999 node.cce.io/drainage-disable=true
```

步骤4 查看结果。

```
kubectl get drainages 192.168.1.67-1721616409999 -o yaml
```

回显如下，此时phase参数为Cancelled。

```
apiVersion: node.cce.io/v1
kind: Drainage
metadata:
  annotations:
    node.cce.io/drainage-disable: "true"
  creationTimestamp: "2024-07-22T03:12:56Z"
  generation: 1
  name: 192.168.1.67-1721616409999
  resourceVersion: "2689858"
  uid: 3ec131e4-0505-4c88-8255-ef9d0eb02712
spec:
  force: true
  nodeName: 192.168.1.67
  timeout: 0
status:
  conditions:
  - lastTransitionTime: "2024-07-22T03:12:56Z"
    message: start to drain node
    reason: Started
    status: "True"
```

```
  type: Started
- lastTransitionTime: "2024-07-22T03:13:26Z"
  message: node has been drained
  reason: Succeeded
  status: "True"
  type: Finished
- lastTransitionTime: "2024-07-22T03:37:48Z"
  message: node drainage has been cancelled
  reason: Cancelled
  status: "True"
  type: Cancelled
phase: Cancelled
```

----结束

通过 API 操作

步骤1 获取集群所在区域的Token，获取方式请参见**获取Token**。

步骤2 根据接口格式确定节点排水接口URL。

取消节点排水接口的URL为：

```
https://{clusterid}.Endpoint/apis/node.cce.io/v1/drainages/{drainageName}
```

- **{clusterid}**：集群ID，可通过CCE控制台的“总览”页面查询。
- **Endpoint**：集群所在区域的终端节点。

  对应的值请参见**地区和终端节点**。

  例如CCE服务在“亚太-新加坡”区域的Endpoint为“cce.ap-southeast-3.myhuaweicloud.com”
- **{drainageName}**：Drainage资源的名称。Drainage资源名称可以使用**kubectl get drainages**命令查询。

步骤3 使用**PATCH**请求方法，并设置请求Header参数。

```
curl --location --request PATCH 'https://{clusterid}.Endpoint/apis/node.cce.io/v1/drainages/{drainageName}' \
--header 'Content-Type: application/merge-patch+json' \
--header 'X-Auth-Token: MIIWvw******' \
--data @Drainage-cancel.json
```

请求中包含的Header参数如下：

表 3-64 请求 Header 参数

参数	是否必选	参数类型	描述
Content-Type	是	String	消息体的类型（格式），使用PATCH方式时参数值为application/merge-patch+json。
X-Auth-Token	是	String	使用Token调用接口，Token的获取方式请参见**获取Token**。

其中Drainage-cancel.json为当前路径下的本地文件，内容如下：

```
{
    "metadata": {
        "annotations": {
            "node.cce.io/drainage-disable": "true"
```



```
    }
  }
}
```

----结束

3.8.7 删除/退订节点

操作场景

当您不再需要该节点继续工作时，请您在节点列表进行删除按需节点或退订包年/包月节点的标准化操作，以免带来不符合预期的效果。

在CCE集群中删除/退订节点会将该节点以及节点内运行的业务都销毁，请您在操作前提前进行排水和数据备份，确保正常业务运行不受影响。

注意事项

- 删除节点会涉及Pod迁移，可能会影响业务，请在业务低峰期操作，建议您提前进行**节点排水**。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 退订包年/包月节点时，订单处理需要一定时间，在此期间节点处于不可用状态，预计5-10分钟后自动清理该节点。
- 删除节点会导致与节点关联的**本地持久卷**类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。删除节点时使用了本地持久存储卷的Pod会从删除的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。

删除按需计费节点

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击节点后的“更多 > 删除”。

步骤4 在弹出的“删除节点”窗口中，输入“DELETE”，单击“是”，等待完成节点删除。

说明

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

----结束

退订包年/包月节点

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击节点后的“更多 > 退订”，可见“退订节点”弹窗，可选择排水操作。

选择排水节点后，系统会将节点设置为不可调度，然后安全地将节点上所有符合节点排水规则的Pod驱逐，后续新建的Pod都不会再调度到该节点，该排水时间取决于Pod情况，如时间过长，可返回节点列表查看事件。

步骤4 确认退订弹窗后将跳转到订单页面，您可确认退订类别和金额后完成退订操作。

说明

- 退订处理预计需要5-10分钟，在此期间，节点将处于不可用状态。
- 包周期节点在退订时，节点会自动被打上K8s标签node.cce.io/unsubscribe: true。

----结束

3.8.8 按需节点转包年/包月

当前在CCE中购买节点时支持“按需计费”和“包年/包月”（按周期）两种计费方式。按需计费的购买的节点可以转成按周期计费的节点。

约束与限制

- 按需节点池中的节点转成包年/包月时，需要将集群升级到v1.19.16-r40、v1.21.11-r0、v1.23.0-r0、v1.25.4-r0及以上版本。
- 当按需节点池中的节点转成包年/包月后，该节点不支持弹性缩容。

按需节点转包年/包月

须知

- 按需计费节点绑定的资源（云硬盘、弹性公网IP）可能不支持同步变更计费模式，详情请参见**弹性云服务器ECS按需转包年/包月说明**。
- 按需节点池中的节点转成包年/包月时，请在节点列表中找到目标节点并单击“更多>开启节点缩容保护”，然后再进行转包年/包月操作。

如果您在购买按需计费的节点后，想更换为包年/包月计费，可按如下步骤进行操作：

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧选择节点管理，在节点的操作列选择“更多 > 转包年包月”。

图 3-10 按需节点转包年/包月

步骤3 单击“确定”，等待生成订单并完成支付即可。

----结束

3.8.9 包年/包月节点修改自动续费配置

购买包年/包月计费模式的节点后，您可以根据需求为您的节点开通自动续费，或者修改已有的自动续费配置。

开通自动续费配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“节点管理”，并切换至“节点”页签。

步骤3 单击包年/包月节点操作栏中的“更多>开通自动续费”按钮。您也可以批量选择多个节点，单击顶部功能栏中的“更多>开通自动续费”按钮。然后在弹窗中对需要开通自动续费的节点进行确认。

步骤4 在跳转的“设为自动续费”页面中，设置自动续费的参数。

- 选择续费时长：支持设置每次续费的时长，每次续费最短为1个月，最长为1年。
- 自动续费次数：勾选后，可设置自定义的续费次数，超出该次数后将不再自动续费。不设置自动续费次数时，默认为不限次数。

图 3-11 设置自动续费

说明

如果节点上绑定了EIP等资源，您可以通过勾选该资源决定是否随节点开通自动续费。

步骤5 单击“开通”。

----结束

修改自动续费配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“节点管理”，并切换至“节点”页签。

步骤3 单击包年/包月节点操作栏中的“更多>修改自动续费”按钮。您也可以批量选择多个节点，单击顶部功能栏中的“更多>修改自动续费”按钮。然后在弹窗中对需要修改自动续费的节点进行确认。

步骤4 在跳转的“修改自动续费”页面中，修改自动续费的参数。

- 续费方式：如果节点开通自动续费的节点，默认设置为“到期自动续费”。如果需要关闭自动续费配置，请选择“手动续费”。

- 选择续费时长：支持修改每次续费的时长，每次续费最短为1个月，最长为1年。
- 自动续费次数：勾选后，可设置自定义的续费次数，超出该次数后将不再自动续费。不设置自动续费次数时，默认为不限次数。

图 3-12 修改自动续费

步骤5 单击“确定”。

----结束

3.8.10 节点关机

操作场景

集群中的节点关机后，该节点以及节点内的业务将停止运行，且该节点将被设置为不可调度状态。节点关机前，请先确认您的正常业务运行将不受影响，请谨慎操作。

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

注意事项

- 节点关机会涉及Pod迁移，可能会影响业务，请在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。

操作方法

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 找到目标节点，单击待关机节点的名称。

步骤4 页面跳转至弹性云服务器详情页中，单击右上角的“关机”，在弹出的关机窗口中单击“确定”，即可完成关机操作。

图 3-13 弹性云服务器详情页

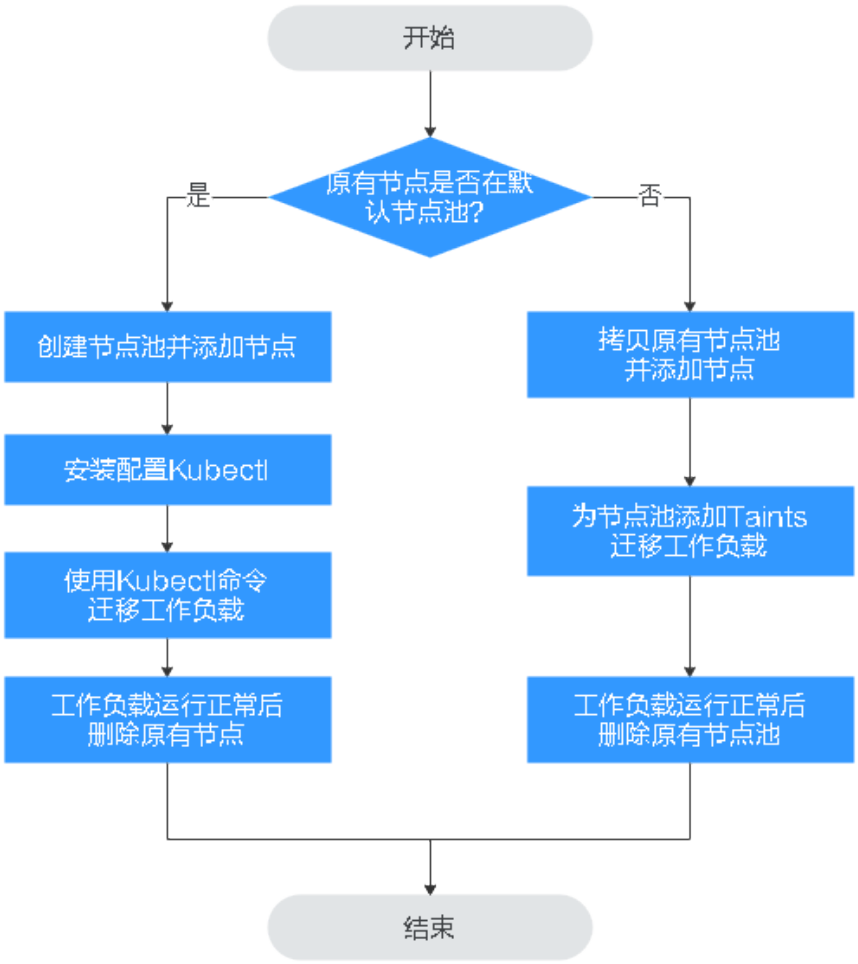
----结束

3.8.11 节点滚动升级

操作场景

节点滚动升级就是先创建新节点，然后将工作负载迁移到新的节点上，再删除旧节点。迁移流程如图3-14所示。

图 3-14 节点迁移流程



约束与限制

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

原有节点在默认节点池

步骤1 创建新的节点池。具体请参见**创建节点池**。

步骤2 单击节点池名称，单击“操作”区域的“节点列表”可查看新建节点的IP地址。

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

步骤4 迁移工作负载。

1. 给需要迁移工作负载的节点打上Taint（污点）。

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

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

   - NoSchedule：一定不能被调度。

- PreferNoSchedule：尽量不要调度。
- NoExecute：不仅不会调度，还会驱逐Node上已有的Pod。

说明

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

2. 安全驱逐节点上的工作负载。

   kubectl drain *[node]*

   其中，*[node]*为待转移工作负载所在节点的IP。

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

说明

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

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

步骤5 删除原有节点。

工作负载迁移成功且运行正常后，即可删除原有节点。

----结束

原有节点不在默认节点池

步骤1 复制节点池并添加节点。具体请参见**复制节点池**。

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

步骤3 迁移工作负载。

1. 单击原节点池后的“编辑”配置污点参数。
2. 输入“污点(Taints)”的Key和Value值，Effect选项有NoSchedule、PreferNoSchedule或NoExecute，此处必须选择“NoExecute”，单击“确认添加”。
   - NoSchedule：一定不能被调度。
   - PreferNoSchedule：尽量不要调度。
   - NoExecute：不仅不会调度，还会驱逐Node上已有的Pod。

   说明

   若需要重新设置污点，需删除已配置污点。
3. 单击“确定”。
4. 在左侧导航栏中选择“工作负载 > 无状态负载 Deployment”。在工作负载列表中，待迁移工作负载的状态由“运行中”变为“未就绪”。工作负载状态再次变为“运行中”，表示迁移成功。

说明

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

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

步骤4 删除原有节点。

工作负载迁移成功且运行正常后，即可删除原有节点。

----结束

3.9 节点运维

3.9.1 节点预留资源策略说明

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

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

CCE对用户节点可分配的资源计算法则如下：

Allocatable = Capacity - Reserved - Eviction Threshold

即，**节点资源可分配量=总量-预留值-驱逐阈值**。其中内存资源的驱逐阈值，固定为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。详情请参见**使用free命令查看弹性云服务器的内存，为什么与实际不符?**

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：

1. 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**。
2. 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。

CCE 对节点内存的预留规则 v1

说明

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

如果节点资源占用比较满，集群升级到**v1.21.4-r0**和**v1.23.3-r0**及以上版本之后可能会因为系统组件预留值变大而导致节点上的负载被驱逐。

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

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

表 3-65 系统组件预留规则

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

表 3-66 Kubelet 管理 Pod 所需预留规则

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

须知

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

CCE 对节点内存的预留规则 v2

对于**v1.21.4-r0**和**v1.23.3-r0**及以上版本集群，节点内存的预留规则优化为v2模型，且支持通过节点池配置管理参数（kube-reserved-mem和system-reserved-mem）动态调整，具体方法请参见**修改节点池配置**。

CCE节点内存v2模型的总预留值等于**OS侧预留值**与**CCE管理Pod所需预留值**之和。

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

表 3-67 节点内存预留规则 v2

预留类型	基础/浮动	预留公式	预留对象
OS侧预留	基础预留	固定400MiB	sshd、systemd-journald等操作系统服务组件占用
	浮动预留（随节点内存）	25MiB/GiB	内核占用
CCE侧预留	基础预留	固定500MiB	节点空载时， kubelet、kube-proxy等容器引擎组件占用。
	浮动预留（随节点Pod数量）	Docker：20MiB/Pod Containerd：5MiB/Pod	Pod数量增加时，容器引擎组件的额外占用。 **说明** v2模型在计算节点默认预留内存时，随内存估计节点默认最大实例数，请参见表3-71。

CCE 对节点 CPU 的预留规则

表 3-68 节点 CPU 预留规则

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

CCE 对节点数据盘的预留规则

CCE使用LVM（Logical Volume Manager）进行磁盘管理，LVM会在磁盘上创建一个metadata区域用于存储逻辑卷和物理卷的信息，导致磁盘存在4MiB的不可用空间。因此节点实际可用的磁盘空间 = 磁盘大小 - 4MiB。

3.9.2 默认数据盘空间分配说明

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

设置默认数据盘空间分配

说明

- v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，节点会添加一块默认数据盘供容器运行时和Kubelet组件使用，您可以自定义默认数据盘的空间分配。
- v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中，如果“系统组件存储”选择“数据盘”，节点才会添加一块默认数据盘供容器运行时和Kubelet组件单独使用，您可以自定义默认数据盘的空间分配。

在创建节点时，您可在存储配置中，单击“展开高级配置”，自定义默认数据盘的空间分配。

图 3-15 设置数据盘空间分配

- **容器引擎空间分配**：
  - 指定磁盘空间：CCE将数据盘空间默认划分为两块，一块用于存放容器引擎(Docker/Containerd) 工作目录、容器镜像的数据和镜像元数据；另一块用于Kubelet组件和EmptyDir临时存储等。容器引擎空间的剩余容量将会影响镜像下载和容器的启动及运行。
    - 容器引擎和容器镜像空间（默认占90%）：用于容器运行时工作目录、存储容器镜像数据以及镜像元数据。
    - Kubelet组件和EmptyDir临时存储（默认占10%）：用于存储Pod配置文件、密钥以及临时存储EmptyDir等挂载数据。

    说明

    当“容器引擎和容器镜像空间”和“Kubelet组件和EmptyDir临时存储空间”分配比例之和不满100%时，剩余空间将分配给用户数据使用，您可以将其挂载到指定路径下。挂载路径请填写业务目录路径，不可设置为空或根目录等操作系统关键路径。
  - 共享磁盘空间：v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及之后版本的集群中，CCE使用的数据盘支持采用**容器引擎和Kubelet共享磁盘空间**的方式，即不再划分容器引擎 (Docker/Containerd) 和Kubelet组件的空间。
- **Pod容器空间分配**：即容器的basesize设置，每个工作负载下的容器组 Pod 占用的磁盘空间设置上限（包含容器镜像占用的空间）。合理的配置可避免容器组无节制使用磁盘空间导致业务异常。建议此值不超过容器引擎空间的 80%。该参数与节点操作系统和容器存储Rootfs相关，部分场景下不支持设置。详情请参见**操作系统与容器存储Rootfs对应关系**。
- 写入模式：

- 线性：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。
- 条带化：有两块以上数据盘时才可支持选择条带化模式。创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。

容器引擎空间分配

对于容器引擎和Kubelet共享磁盘空间的节点，容器存储Rootfs为**OverlayFS类型**，数据盘空间分配详情请参见**容器引擎和Kubelet共享磁盘空间说明**。

对于容器引擎和Kubelet不共享磁盘空间的节点，数据盘根据容器存储Rootfs不同具有两种划分方式（以100G大小为例）：**Device Mapper类型**和**OverlayFS类型**。不同操作系统对应的容器存储Rootfs请参见**操作系统与容器存储Rootfs对应关系**。

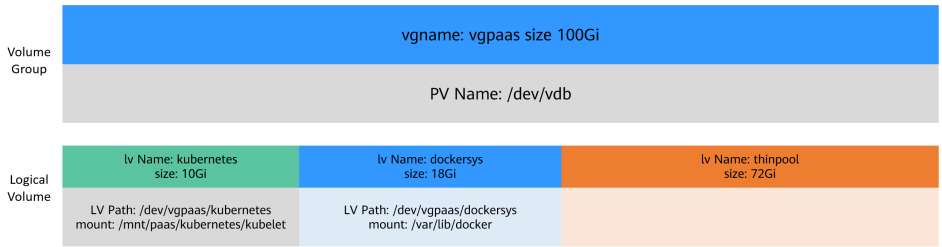
- **Device Mapper类型存储Rootfs**

  其中默认占90%的容器引擎和容器镜像空间又可分为以下两个部分：

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

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

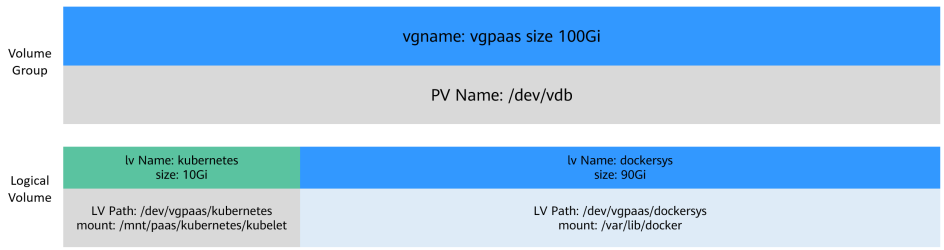
图 3-16 Device Mapper 类型容器引擎空间分配



- **OverlayFS类型存储Rootfs**

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

图 3-17 OverlayFS 类型容器引擎空间分配



Pod 容器空间分配

自定义Pod容器空间（basesize）设置与节点操作系统和容器存储Rootfs相关（容器存储Rootfs请参见**操作系统与容器存储Rootfs对应关系**）：

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

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

图 3-18 创建节点时的最大实例数配置

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

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

操作系统与容器存储 Rootfs 对应关系

表 3-69 CCE 集群节点操作系统与容器引擎对应关系

操作系统	容器存储Rootfs	自定义Pod容器空间（basesize）
CentOS 7.x	v1.19.16以下版本集群使用Device Mapper v1.19.16及以上版本集群使用OverlayFS	Rootfs为Device Mapper且容器引擎为Docker时支持自定义Pod容器空间，默认值为10G。 Rootfs为OverlayFS时不支持自定义Pod容器空间。
EulerOS 2.3	Device Mapper	仅容器引擎为Docker时支持自定义Pod容器空间，默认值为10G。
EulerOS 2.5	Device Mapper	仅容器引擎为Docker时支持自定义Pod容器空间，默认值为10G。

操作系统	容器存储Rootfs	自定义Pod容器空间（basesize）
EulerOS 2.8	v1.19.16-r2以下版本集群使用Device Mapper v1.19.16-r2及以上版本集群使用OverlayFS	Rootfs为Device Mapper且容器引擎为Docker时支持自定义Pod容器空间，默认值为10G。 Rootfs为OverlayFS且容器引擎为Docker时支持自定义Pod容器空间，默认值为不限制。
EulerOS 2.9	OverlayFS	v1.19.16-r0、v1.21.3-r0、v1.23.3-r0及以上的集群版本中，Docker支持自定义Pod容器空间，默认值为不限制。 集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，containerd支持自定义Pod容器空间，默认值为不限制。 v1.19.16-r0、v1.21.3-r0、v1.23.3-r0以前的集群版本不支持自定义Pod容器空间。
EulerOS 2.10	OverlayFS	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0以下时，仅容器引擎为Docker时支持自定义Pod容器空间，默认值为不限制。 集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，Docker和containerd均支持自定义Pod容器空间，默认值为不限制。
Ubuntu 18.04	OverlayFS	不支持自定义Pod容器空间。
Ubuntu 22.04	OverlayFS	不支持自定义Pod容器空间。
Huawei Cloud EulerOS 1.1	OverlayFS	不支持自定义Pod容器空间。
Huawei Cloud EulerOS 2.0	OverlayFS	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0以下时，仅容器引擎为Docker时支持自定义Pod容器空间，默认值为不限制。 集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，Docker和containerd均支持自定义Pod容器空间，默认值为不限制。

表 3-70 CCE Turbo 集群节点操作系统与容器引擎对应关系

操作系统	容器存储Rootfs	自定义Pod容器空间（basesize）
CentOS 7.x	OverlayFS	不支持自定义Pod容器空间。
Ubuntu 18.04	OverlayFS	不支持自定义Pod容器空间。
Ubuntu 22.04	OverlayFS	不支持自定义Pod容器空间。
EulerOS 2.9	OverlayFS	Rootfs为OverlayFS且仅容器引擎为Docker时支持自定义Pod容器空间，默认值为不限制。集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，Docker和containerd均支持自定义Pod容器空间。
EulerOS 2.10	弹性云服务器-物理机使用Device Mapper	Rootfs为Device Mapper且容器引擎为containerd时支持自定义Pod容器空间，默认值为10G。
Huawei Cloud EulerOS 1.1	OverlayFS	不支持自定义Pod容器空间。
Huawei Cloud EulerOS 2.0	OverlayFS	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0以下时，仅容器引擎为Docker时支持自定义Pod容器空间，默认值为不限制。 集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，Docker和containerd均支持自定义Pod容器空间，默认值为不限制。

镜像回收策略说明

当容器引擎空间不足时，会触发镜像垃圾回收。

镜像垃圾回收策略只考虑两个因素：HighThresholdPercent 和 LowThresholdPercent。 磁盘使用率超过上限阈值（HighThresholdPercent，默认值为80%）将触发垃圾回收。 垃圾回收将删除最近最少使用的镜像，直到磁盘使用率满足下限阈值（LowThresholdPercent，默认值为70%）。

容器引擎空间大小配置建议

- 容器引擎空间需要大于容器使用的磁盘总空间，即：**容器引擎空间** > **容器数量** * **Pod容器空间（basesize）**
- 容器业务的创删文件操作建议在容器挂载的本地存储（如emptyDir、hostPath）或云存储的目录中进行，这样不会占用thinpool空间。其中Emptydir使用的是kubelet空间，需要规划好kubelet空间的大小。
- 可将业务部署在使用OverlayFS存储模式的节点上（请参见**操作系统与容器存储Rootfs对应关系**），避免容器内创删文件后占用的磁盘空间不立即释放问题。

容器引擎和 Kubelet 共享磁盘空间说明

容器引擎和Kubelet共享磁盘空间即在节点上不再划分容器引擎 (Docker/Containerd)和Kubelet组件的空间，二者共用磁盘空间。

须知

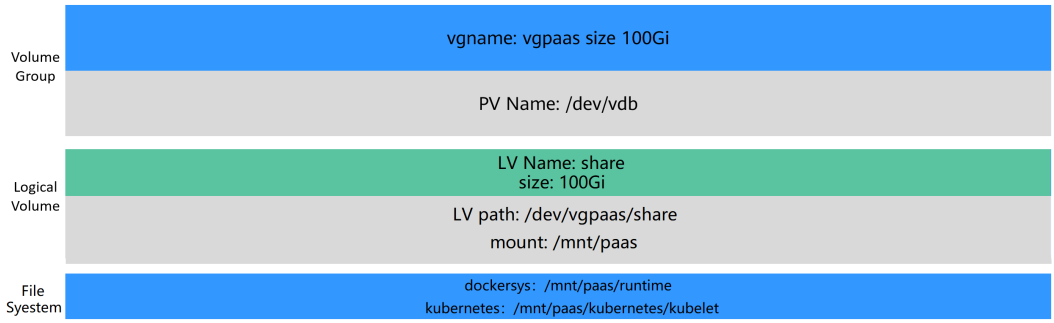
- 容器引擎和Kubelet共享磁盘空间仅v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及以上的集群支持。
- 容器存储Rootfs为OverlayFS类型时支持共享磁盘空间，Device Mapper类型不支持。
- 若您在集群中安装了npd插件，请将插件升级至1.18.10版本及以上，否则会产生误报警。
- 若您在集群中安装了log-agent插件，请将插件升级至1.3.0版本及以上，否则会影响日志采集。
- 若您在集群中安装了ICAgent，请将ICAgent升级至5.12.140版本及以上，否则会影响日志采集。查看或升级ICAgent版本请参见**CCE接入**。

图 3-19 共享磁盘空间配置

对于共享磁盘空间的节点，容器存储Rootfs为**OverlayFS类型**。节点创建完成后，数据盘空间（以100G大小为例）不再划分容器引擎和容器镜像空间和Kubelet组件空间，均在/mnt/paas目录下，并通过两个文件系统区分：

- dockersys：/mnt/paas/runtime
- kubernetes：/mnt/paas/kubernetes/kubelet

图 3-20 共享数据盘空间分配



常见问题

如何扩容容器的存储空间?

CCE集群中的节点磁盘扩容

3.9.3 节点可创建的最大 Pod 数量说明

节点最大 Pod 数量计算方式

根据集群类型不同，节点可创建的最大Pod数量计算方式如下：

网络模型	节点可创建的最大Pod数量计算方式	建议
“容器隧道网络”集群	仅取决于**节点最大实例数**	-
“VPC网络”集群	取决于**节点最大实例数**和**节点可分配容器IP数**中的最小值	建议节点最大实例数不要超过节点可分配容器IP数，否则当节点容器IP数不足时，新建Pod将无法在该节点上正常运行。
“云原生2.0网络”集群（CCE Turbo集群）	取决于**节点最大实例数**和**CCE Turbo集群节点网卡数量**中的最小值	建议节点最大实例数不要超过节点网卡数，否则当节点可分配网卡不足时，新建Pod将无法在该节点上正常运行。

节点可分配容器 IP 数说明

在创建CCE集群时，如果网络模型选择“VPC网络”，根据VPC网络模型的容器IP地址分配规则（详见**容器IP地址管理**），您需要选择每个节点可供分配的容器IP数量（即alpha.cce/fixPoolMask参数）。

该参数会影响节点上可以创建最大Pod的数量，因为使用**容器网络**时每个Pod会占用一个IP，如果节点预分配的容器IP数量不够的话，就无法创建Pod。当Pod直接使用**主机网络**（即YAML中配置hostNetwork: true）时，不占用可分配容器IP，详情请参见**容器网络与主机网络的Pod IP分配差异**。

图 3-21 VPC 网络模型节点可分配容器 IP 数配置

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

节点最大实例数说明

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

须知

对于默认节点池（DefaultPool）中的节点，节点创建完成后，最大实例数不支持修改。

对于自定义节点池中的节点，创建完成后可通过修改节点池配置中的max-pods参数，修改节点最大实例数。详情请参见**节点池配置管理**。

默认场景下，节点最大实例数最多可调整至256。如果您期望提升节点上的部署密度，您可以**提交工单**申请调整节点最大实例数，最大支持修改至512个实例。

图 3-22 创建节点时的最大实例数配置

根据节点规格不同，节点默认最大实例数如**表3-71**所示。

表 3-71 节点默认最大实例数

内存	节点默认最大实例数
4G	20

内存	节点默认最大实例数
8G	40
16G	60
32G	80
64G及以上	110

节点网卡数量说明（仅 CCE Turbo 集群）

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

图 3-23 节点网卡数

容器网络与主机网络的 Pod IP 分配差异

创建Pod时，可以选择Pod使用容器网络或是宿主机网络。

- 容器网络：默认使用容器网络，Pod的网络由集群网络插件负责分配，**每个Pod分配一个IP地址，会占用容器网络的IP**。
- 主机网络：Pod直接使用宿主机的网络，即在Pod中配置hostNetwork: true参数，详情请参见**在Pod中配置主机网络（hostNetwork）**。配置完成后的Pod会占用宿主机的端口，Pod的IP就是宿主机的IP，**不会占用容器网络的IP**。使用时需要考虑是否与宿主机上的端口冲突，因此一般情况下除非某个特定应用必须使用宿主机上的特定端口，否则不建议使用主机网络。

3.9.4 CCE 节点 kubelet 和 runtime 组件路径与社区原生配置差异说明

为保证节点的系统稳定性，CCE将Kubernetes和容器运行时的相关组件单独存储在数据盘中。其中Kubernetes使用“/mnt/paas/kubernetes”目录，容器运行时使用/“mnt/paas/runtime”目录，并使用软链接的方式与社区默认路径保持一致。

CCE 节点 kubelet 和 runtime 组件默认路径与社区原生的配置差异

kubelet和runtime组件路径名称	Kubernetes原生路径	CCE节点上的路径
kubelet的启动参数root-dir	/var/lib/kubelet	/mnt/paas/kubernetes/kubelet

kubelet和runtime组件路径名称	Kubernetes原生路径	CCE节点上的路径
kubelet的路径	/var/lib/kubelet	/mnt/paas/kubernetes/kubelet 同时创建了/var/lib/kubelet -> /mnt/paas/kubernetes/kubelet的软链接
容器运行时（docker）的路径	/var/lib/docker	• **数据盘空间分配**设置为“共享磁盘空间”：/mnt/paas/runtime 同时创建了/var/lib/docker -> /mnt/paas/runtime的软链接 • **数据盘空间分配**设置为“指定磁盘空间”：与Kubernetes原始路径保持一致，即/var/lib/docker
容器运行时（containerd）的路径	/var/lib/containerd	• **数据盘空间分配**设置为“共享磁盘空间”：/mnt/paas/runtime 同时创建了/var/lib/containerd -> /mnt/paas/runtime的软链接 • **数据盘空间分配**设置为“指定磁盘空间”：与Kubernetes原始路径保持一致，即/var/lib/containerd
containerd日志存储的路径	/var/log/pods	/var/lib/containerd/container_logs 同时创建了/var/log/pods -> /var/lib/containerd/container_logs的软链接

说明

CCE节点kubelet和runtime默认路径与社区原生的配置差异可能带来以下影响：

- 软链文件在容器挂载场景下，无法访问软链文件指向的真实路径。

  例如：将容器通过hostPath的方式将主机的/var/log路径挂载进容器/mnt/log路径，此时在容器内看到/mnt/log/pods是一个异常的软链文件，无法访问/var/log/pods下的真实文件内容。

  建议将真实的文件路径挂载进容器内，避免软链导致的文件读取失败。

- kubelet的root-dir启动参数（/mnt/paas/kubernetes/kubelet）与社区路径（/var/lib/kubelet）不一致，使用第三方CSI插件的容器挂载路径为社区路径，会导致文件挂载不生效。

  例如，vault开源三方插件在使用secrets-store-csi-driver挂载密钥时，如果插件的root-dir地址与CCE配置路径不一致（插件默认value值与社区地址一致：/var/lib/kubelet）会导致容器内无法获取到vault的密钥。

  这是因为CSI插件依赖挂载传播，将卷挂载到对应容器中。kubelet在挂卷的CSI请求中会带有与kubelet启动参数root-dir相关的挂载路径（/mnt/paas/kubernetes/kubelet/pods/{podID}/volume/…）。当CSI容器将卷挂载在kubelet CSI请求的挂载路径上，而此挂载路径与主机路径不相关时，挂载传播将失效。例如，CSI容器将主机的/var/lib/kubelet路径挂载进容器的/var/lib/kubelet路径，因此CSI容器内的/mnt/paas/kubernetes/kubelet/pods/{podID}/volume/…将与主机路径毫不相关。

  建议修改CSI插件中的root-dir地址，与CCE当前节点kubelet的root-dir地址保持一致。

CCE 节点 kubelet 和 runtime 组件默认路径由软链创建方式切换挂载绑定说明

为更好的兼容第三方开源软件，在v1.23.18-r0、v1.25.13-r0、v1.27-10-r0、v1.28.8-r0、v1.29.4-r0及以上集群版本中，支持通过集群或节点池配置将软链方式修改为挂载，且节点池配置优先级高于集群配置。

当前仅提供API方式创建：

- 创建集群API示例如下，其余参数配置说明请参见**创建集群**。

```
POST /api/v3/projects/{project_id}/clusters
{
    "kind": "Cluster",
    "apiVersion": "v3",
    "metadata": {
        "name": "xxxxxxxxxx",
        ...
    },
    "spec": {
        ...
        "configurationsOverride": [
            {
                "name": "node-config",
                "configurations": [
                    {
                        "name": "enable-mount-bind",
                        "value": true
                    }
                ]
            }
        ],
        ...
    }
}
```

- 节点池创建示例如下，其余参数配置说明请参见**创建节点池**。

```
{
    "kind": "NodePool",
    "apiVersion": "v3",
    "metadata": {
        "name": "xxxxxxxxx",
```

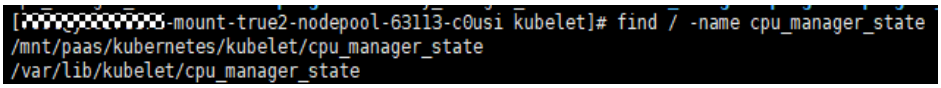
```
            ...
        },
        "spec": {
            ...
            "nodeTemplate": {
                "configurationsOverride": [{
                    "name": "node-config",
                    "configurations": [{
                        "name": "enable-mount-bind",
                        "value": true
                    }]
                }],
                ...
            }
        }
}
```

说明

CCE节点kubelet和runtime组件默认路径由软链创建方式切换挂载绑定的影响如下：

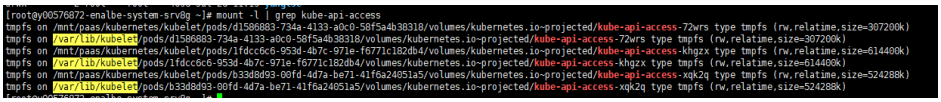
- 软链改挂载绑定后，源目录保持不变，目标软链文件将被替换为目录挂载。能够保证软链路径挂载的容器在改为挂载绑定后，也能够正常访问，无需额外适配。
- 链改挂载绑定后，在到源目录和目标目录会存在两个相同的路径结构。并且在公共父目录下查找文件时，会在不同路径下查询到同一个文件。

  例如：/var/lib/kubelet 和 /mnt/paas/kubernetes/kubelet

- 对应挂载绑定的源目的路径上，将会有两条相同的挂载信息。如业务代码中依赖挂载信息判断，请评估对您的影响。

  例如：

3.9.5 将节点容器引擎从 Docker 迁移到 Containerd

Kubernetes在1.24版本中移除了Dockershim，并从此不再默认支持Docker容器引擎。CCE计划未来移除对Docker容器引擎的支持，建议您将节点容器引擎从Docker迁移至Containerd。

前提条件

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

注意事项

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

默认节点池中的节点迁移步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签。

步骤3 在节点列表中选择一个或多个需要重置的节点，单击“更多 > 重置节点”。

步骤4 在容器引擎中选择Containerd，其余参数可根据需要进行调整，也可以和创建时保持一致。

步骤5 当节点状态显示为安装中时，即表示正在重置节点。

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

----结束

自定义节点池迁移步骤

您可使用**节点池复制**功能，复制原有的Docker节点池，并将新节点池的容器引擎选择为Containerd，其余配置和原Docker节点池保持一致。

步骤1 登录CCE控制台，单击集群名称进入集群。

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

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

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

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

说明

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

步骤5 迁移完成后，删除原有Docker节点池。

----结束

3.9.6 节点系统参数优化

3.9.6.1 可优化的节点系统参数列表

CCE提供默认的节点系统参数在某些用户场景下可能出现性能瓶颈，因此用户可对部分节点系统参数进行自定义优化，节点系统参数如**可优化的节点系统参数列表**所示。

须知

- 修改节点系统参数具有一定的风险，需要您对Linux命令和Linux系统知识具有较高程度的了解，避免误操作引起节点故障。
- **表3-72**中的参数已经过测试验证，**请勿自行修改其他参数**以免引起节点故障。
- 修改节点系统参数的命令仅在使用公共镜像时有效，使用私有镜像时本文中提供的命令仅供参考。
- 节点重启后需执行**sysctl -p**用于刷新参数值。

表 3-72 系统参数可优化列表

参数名称	参数位置	说明	参考文档
kernel.pid_max	/etc/sysctl.conf	节点进程 ID数量上限。 查看参数： `sysctl kernel.pid_max`	**修改节点进程 ID数量上限 kernel.pid_max**
RuntimeMaxUse	/etc/systemd/journald.conf	节点日志缓存内存占用量上限，若不配置长时间运行会占用较大内存。 查看参数： `cat /etc/systemd/journald.conf | grep RuntimeMaxUse`	**修改节点日志缓存内存占用量上限 RuntimeMaxUse**
Openfiles	/etc/security/limits.conf	节点单进程最大文件句柄数，可视业务情况调整。 查看参数： `ulimit -n`	**修改节点单进程最大文件句柄数**

（Openfiles容器内部）
LimitNOFILE
LimitNPROC
	• CentOS/EulerOS系统：
	– docker节点：/usr/lib/systemd/system/docker.service
	– containerd节点：/usr/lib/systemd/system/containerd.service
	• Ubuntu系统：
	– docker节点：/lib/systemd/system/docker.service
	– containerd节点：/lib/systemd/system/containerd.service
		容器内部单进程最大文件句柄数，可视业务情况调整。
		查看参数：
		docker节点：
		`cat /proc/`pidof dockerd`/limits | grep files`
		containerd节点：
		`cat /proc/`pidof containerd`/limits | grep files`
			修改容器单进程最大文件句柄数

参数名称	参数位置	说明	参考文档
file-max	/etc/sysctl.conf	系统整体最大文件句柄数，可视业务情况调整。 查看参数： `sysctl fs.file-max`	修改节点系统级最大文件句柄数

nf_conntrack_buckets
nf_conntrack_max
	/etc/sysctl.conf	连接跟踪表容量，可视业务场景调整。
		计算桶占用率= [nf_conntrack_count] / [nf_conntrack_buckets]。
		通过调整buckets值，将占用率调整至0.7以下。
		查看参数：
		`sysctl net.netfilter.nf_conntrack_count`
		`sysctl net.netfilter.nf_conntrack_buckets`
		`sysctl net.netfilter.nf_conntrack_max`
			修改节点内核参数

net.netfilter.nf_conntrack_tcp_timeout_close	/etc/sysctl.conf	连接跟踪表里处于close状态连接的表项的过期时间，缩短过期时间可加快回收。 查看参数： `sysctl net.netfilter.nf_conntrack_tcp_timeout_close`	
net.netfilter.nf_conntrack_tcp_be_liberal	/etc/sysctl.conf	参数值为0或1。 • 0：表示关闭，所有不在TCP窗口中的包都被标志为无效。 • 1：表示开启，只有不在TCP窗口内的包被标志为无效。容器场景下，开启这个参数可以避免NAT过的TCP连接带宽受限。 查看参数： `sysctl net.netfilter.nf_conntrack_tcp_be_liberal`	
tcp_keepalive_time	/etc/sysctl.conf	TCP超时时长，即发送keepalive探测消息的间隔时间。参数值过大可能导致TCP回收时间过长，长时间下造成大量连接卡在Close_wait阶段，耗尽系统资源。 查看参数： `sysctl net.ipv4.tcp_keepalive_time`	

参数名称	参数位置	说明	参考文档
tcp_max_syn_backlog	/etc/sysctl.conf	TCP最大半连接数，SYN_RECV 队列中的最大连接数。 查看参数： `sysctl net.ipv4.tcp_max_syn_backlog`	
tcp_max_tw_buckets	/etc/sysctl.conf	指定任何时候允许存在的处于“time-wait”状态的最大套接字数，参数值过大时易耗尽节点资源。 查看参数： `sysctl net.ipv4.tcp_max_tw_buckets`	
net.core.somaxconn	/etc/sysctl.conf	TCP最大连接数，该参数控制TCP连接队列的大小。参数值过小时极易不足，设置过大则可能会导致系统资源的浪费，因为连接队列中每个等待连接的客户端都需要占用一定的内存资源。 查看参数： `sysctl net.core.somaxconn`	
max_user_instances	/etc/sysctl.conf	每个用户允许的最大 inotify 实例数，参数值过小时容器场景下极易不足。 查看参数： `sysctl fs.inotify.max_user_instances`	
max_user_watches	/etc/sysctl.conf	所有监视实例的最大目录数，参数值过小时容器场景极易不足。 查看参数： `sysctl fs.inotify.max_user_watches`	
netdev_max_backlog	/etc/sysctl.conf	网络协议栈收包队列大小，参数值过小时极易不足。 查看参数： `sysctl net.core.netdev_max_backlog`	

net.core.wmem_max
net.core.rmem_max
	/etc/sysctl.conf	收发缓冲区内存大小（字节），参数值过小时大文件场景下易不足。
		查看参数：
		`sysctl net.core.wmem_max`
		`sysctl net.core.rmem_max`	

参数名称	参数位置	说明	参考文档

net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh3
	/etc/sysctl.conf	ARP表项的垃圾回收调优。
		• gc_thresh1：表示最小可保留的表项数量，如果表项数量小于此值GC（Garbage collector）不进行回收操作。**请勿修改。**
		• gc_thresh2：当表项数量超过此值时，GC将会清空大于5秒的表项。**请勿修改。**
		• gc_thresh3：最大可允许的非永久表项数量。如果系统拥有庞大的接口数量，或者直连了大量的设备，应增大此值。
		查看参数：
		`sysctl net.ipv4.neigh.default.gc_thresh1`
		`sysctl net.ipv4.neigh.default.gc_thresh2`
		`sysctl net.ipv4.neigh.default.gc_thresh3`	

vm.max_map_count	/etc/sysctl.conf	参数值过小时，安装elk时会提示不足。 查看参数： `sysctl vm.max_map_count`	

3.9.6.2 修改节点日志缓存内存占用量上限 RuntimeMaxUse

Journald是Linux中的日志系统，负责把日志信息写入二进制文件，并默认使用/run/log/journal目录作为日志缓存目录。Journald的配置文件位于节点/etc/systemd/journald.conf目录，其中RuntimeMaxUse参数表示日志缓存的最大内存占用量。若不配置RuntimeMaxUse，长时间运行会占用较大内存。

须知

修改节点系统参数的命令仅在使用公共镜像时有效，使用私有镜像时本文中提供的命令仅供参考。

修改节点 RuntimeMaxUse

步骤1 登录节点，查看/etc/systemd/journald.conf文件。

```
cat /etc/systemd/journald.conf
```

步骤2 修改RuntimeMaxUse参数，建议值为100M。

- 若查看journald.conf文件时，文件中已设置RuntimeMaxUse值，可通过以下命令对参数值进行修改。
  ```
  sed -i "s/RuntimeMaxUse=[0-9]*M/RuntimeMaxUse=100M/g" /etc/systemd/journald.conf && systemctl restart systemd-journald
  ```
- 若查看journald.conf文件时，文件中还未设置RuntimeMaxUse值，可通过以下命令添加。
  ```
  echo RuntimeMaxUse=100M >> /etc/systemd/journald.conf && systemctl restart systemd-journald
  ```

步骤3 修改完成后，可查看是否修改成功，当返回与修改值一致时说明修改正确。

```
cat /etc/systemd/journald.conf | grep RuntimeMaxUse
```

----结束

创建节点/节点池时自动配置 RuntimeMaxUse

您可以设置节点或节点池安装后执行脚本，在新建节点或节点池时通过脚本配置RuntimeMaxUse大小。

步骤1 首先您需要确认创建节点或节点池的操作系统，例如CentOS 7.6。

步骤2 在**同集群、同操作系统**的节点上进行脚本命令可行性的测试，在节点上手动执行命令，确认脚本命令可行。手动执行脚本命令请参考**修改节点RuntimeMaxUse**。

步骤3 **（以下命令需在手动执行命令验证成功后配置）**在创建节点或节点池时，在“高级配置 > 安装后执行脚本”中添加可执行的脚本命令。

- 登录节点查看/etc/systemd/journald.conf文件，若文件中已设置RuntimeMaxUse值，可通过以下命令对参数值进行修改。
  ```
  sed -i "s/RuntimeMaxUse=[0-9]*M/RuntimeMaxUse=100M/g" /etc/systemd/journald.conf && systemctl restart systemd-journald
  ```
- 登录节点查看/etc/systemd/journald.conf文件，若文件中还未设置RuntimeMaxUse值，可通过以下命令添加。
  ```
  echo RuntimeMaxUse=100M >> /etc/systemd/journald.conf && systemctl restart systemd-journald
  ```

下图中命令作为示例，请根据实际情况填写。

步骤4 节点创建完成后，登录节点查看是否修改成功。

```
cat /etc/systemd/journald.conf | grep RuntimeMaxUse
```

----结束

3.9.6.3 修改最大文件句柄数

最大文件句柄数即打开文件数的最大限制，Linux系统中包含两个文件句柄限制：一个是系统级的，即所有用户的进程同时打开文件数的上限；一种是用户级的，即单个用户进程打开文件数的上限。但是在容器中，还有另一个文件句柄限制，即容器内部单进程最大文件句柄数。

须知

修改节点系统参数的命令仅在使用公共镜像时有效，使用私有镜像时本文中提供的命令仅供参考。

修改节点系统级最大文件句柄数

步骤1 登录节点，查看/etc/sysctl.conf文件。

```
cat /etc/sysctl.conf
```

步骤2 修改fs.file-max参数，**fs.file-max=1048576**为内核参数名称及建议取值。

- 若查看sysctl.conf文件时，文件中已设置fs.file-max值，可通过以下命令进行修改。
  ```
  sed -i "s/fs.file-max=[0-9]*$/fs.file-max=1048576/g" /etc/sysctl.conf && sysctl -p
  ```
- 若查看sysctl.conf文件时，文件中还未设置fs.file-max值，可通过以下命令添加。
  ```
  echo fs.file-max=1048576 >> /etc/sysctl.conf && sysctl -p
  ```

步骤3 执行以下命令检查是否修改成功，当返回与修改值一致时说明修改正确。

```
# sysctl fs.file-max
fs.file-max = 1048576
```

----结束

修改节点单进程最大文件句柄数

步骤1 登录节点，查看/etc/security/limits.conf文件。

```
cat /etc/security/limits.conf
```

节点单进程最大文件句柄数通过以下参数设置：

```
...
root soft nofile 65535
root hard nofile 65535
* soft nofile 65535
* hard nofile 65535
```

步骤2 通过sed命令修改最大文件句柄数，其中65535为最大文件句柄数的建议取值。EulerOS 2.3节点/etc/security/limits.conf中没有nofile相关的默认配置，因此不能通过sed命令进行修改。

```
sed -i "s/nofile.[0-9]*$/nofile 65535/g" /etc/security/limits.conf
```

步骤3 **重新登录节点**，执行以下命令检查是否修改成功，当返回与修改值一致时说明修改正确。

```
# ulimit -n
65535
```

----结束

修改容器单进程最大文件句柄数

步骤1 登录节点，查看/usr/lib/systemd/system/docker.service文件。

- CentOS/EulerOS系统：
  - docker节点：
    ```
    cat /usr/lib/systemd/system/docker.service
    ```
  - containerd节点：
    ```
    cat /usr/lib/systemd/system/containerd.service
    ```
- Ubuntu系统：
  - docker节点：
    ```
    cat /lib/systemd/system/docker.service
    ```
  - containerd节点：
    ```
    cat /lib/systemd/system/containerd.service
    ```

说明

LimitNOFILE或LimitNPROC参数设置为infinity时，表示容器单进程最大文件句柄数为1048576。

容器单进程最大文件句柄数通过以下参数设置：

```
...
LimitNOFILE=1048576
LimitNPROC=1048576
...
```

步骤2 执行如下命令修改两个参数，其中1048576为最大文件句柄数的建议取值。

须知

修改容器最大文件句柄数将会重启docker/containerd进程，请知悉。

- CentOS/EulerOS系统：
  - docker节点：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /usr/lib/systemd/system/docker.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /usr/lib/systemd/system/docker.service && systemctl daemon-reload && systemctl restart docker
    ```
  - containerd节点：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /usr/lib/systemd/system/containerd.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /usr/lib/systemd/system/containerd.service && systemctl daemon-reload && systemctl restart containerd
    ```
- Ubuntu系统：
  - docker节点：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /lib/systemd/system/docker.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /lib/systemd/system/docker.service && systemctl daemon-reload && systemctl restart docker
    ```
  - containerd节点：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /usr/lib/systemd/system/containerd.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /usr/lib/systemd/system/containerd.service && systemctl daemon-reload && systemctl restart containerd
    ```

步骤3 查看容器单进程最大文件句柄数，当返回与修改值一致时说明修改正确。

- docker节点：
  ```
  # cat /proc/`pidof dockerd`/limits | grep files
  Max open files            1048576              1048576              files
  ```


- containerd节点：
  ```
  # cat /proc/`pidof containerd`/limits | grep files
  Max open files            1048576              1048576              files
  ```

----结束

创建节点/节点池时自动配置最大文件句柄数

您可以设置节点或节点池安装后执行脚本，在新建节点或节点池时通过脚本配置最大文件句柄数。

步骤1 首先您需要确认创建节点或节点池的操作系统，例如CentOS 7.6。

步骤2 在**同集群、同操作系统**的节点上，参考以下文档进行脚本命令可行性的测试，在节点上手动执行命令，确认脚本命令可行。

- **修改节点系统级最大文件句柄数**
- **修改节点单进程最大文件句柄数**
- **修改容器单进程最大文件句柄数**

步骤3 **（以下命令均需在手动执行命令验证成功后配置）**在创建节点或节点池时，在“高级配置 > 安装后执行脚本”中添加可执行的脚本命令。

- 修改节点系统级最大文件句柄数：
  - 登录节点查看/etc/sysctl.conf文件，若文件中已设置fs.file-max值，可通过以下命令进行修改。
    ```
    sed -i "s/fs.file-max=[0-9]*$/fs.file-max=1048576/g" /etc/sysctl.conf && sysctl -p
    ```
  - 登录节点查看/etc/sysctl.conf文件，若文件中还未设置fs.file-max值，可通过以下命令添加。
    ```
    echo fs.file-max=1048576 >> /etc/sysctl.conf && sysctl -p
    ```

  其中fs.file-max=1048576为内核参数名称及建议取值。
- 修改节点单进程最大文件句柄数：
  ```
  sed -i "s/nofile.[0-9]*$/nofile 65535/g" /etc/security/limits.conf
  ```

  其中65535为最大文件句柄数的建议取值。
- 修改容器单进程最大文件句柄数：
  - CentOS/EulerOS系统：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /usr/lib/systemd/system/docker.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /usr/lib/systemd/system/docker.service && systemctl daemon-reload && systemctl restart docker
    ```
  - Ubuntu系统：
    ```
    sed -i "s/LimitNOFILE=[0-9a-Z]*$/LimitNOFILE=1048576/g" /lib/systemd/system/docker.service;sed -i "s/LimitNPROC=[0-9a-Z]*$/LimitNPROC=1048576/g" /lib/systemd/system/docker.service && systemctl daemon-reload && systemctl restart docker
    ```

  其中1048576为最大文件句柄数的建议取值。

下图中命令仅做示例，请根据实际情况填写。

步骤4 节点创建完成后，登录节点查看参数是否修改成功。

----结束

3.9.6.4 修改节点内核参数

由于默认的Linux内核参数不一定符合所有用户场景，用户可通过修改节点上的/etc/sysctl.conf配置文件来更改内核参数。

须知

- 修改节点系统参数的命令仅在使用公共镜像时有效，使用私有镜像时本文中提供的命令仅供参考。
- 节点重启后需执行**sysctl -p**用于刷新参数值。

表 3-73 节点内核参数列表

参数名称	参数位置	说明	建议值
file-max	/etc/sysctl.conf	系统整体最大文件句柄数，可视业务情况调整。 查看参数： `sysctl fs.file-max`	fs.file-max=1048576

参数名称	参数位置	说明	建议值

nf_conntrack_buckets
nf_conntrack_max
	/etc/sysctl.conf	连接跟踪表容量，可视业务场景调整。
		计算桶占用率= [nf_conntrack_count] / [nf_conntrack_buckets]。
		当占用率持续超过0.7时，可以通过调大buckets值，将占用率调整至0.7以下。
		查看参数：
		`sysctl net.netfilter.nf_conntrack_count`
		`sysctl net.netfilter.nf_conntrack_buckets`
		`sysctl net.netfilter.nf_conntrack_max`
		说明
		EulerOS 2.3/EulerOS 2.5/CentOS 7.6上net.netfilter.nf_conntrack_buckets不能通过编辑/etc/sysctl.conf进行修改，而是通过修改/sys/module/nf_conntrack/parameters/hashsize达到修改buckets的目的。
			系统默认值已经根据节点内存大小浮动配置。如需修改，可参考以下公式：
			• net.netfilter.nf_conntrack_buckets = [nf_conntrack_count] / 0.7
			• net.netfilter.nf_conntrack_max = 4* [nf_conntrack_buckets]

net.netfilter.nf_conntrack_tcp_timeout_close	/etc/sysctl.conf	连接跟踪表里处于close状态连接的表项的过期时间，缩短过期时间可加快回收。 查看参数： `sysctl net.netfilter.nf_conntrack_tcp_timeout_close`	net.netfilter.nf_conntrack_tcp_timeout_close=3
net.netfilter.nf_conntrack_tcp_be_liberal	/etc/sysctl.conf	参数值为0或1。 • 0：表示关闭，所有不在TCP窗口中的包都被标志为无效。 • 1：表示开启，只有不在TCP窗口内的包被标志为无效。容器场景下，开启这个参数可以避免NAT过的TCP连接带宽受限。 查看参数： `sysctl net.netfilter.nf_conntrack_tcp_be_liberal`	net.netfilter.nf_conntrack_tcp_be_liberal=1

参数名称	参数位置	说明	建议值
tcp_keepalive_time	/etc/sysctl.conf	TCP发送keepalive探测消息的间隔时间。参数值过大可能导致TCP回收时间过长，长时间下造成大量连接卡在Close_wait阶段，耗尽系统资源。 查看参数： `sysctl net.ipv4.tcp_keepalive_time`	net.ipv4.tcp_keepalive_time=600
tcp_max_syn_backlog	/etc/sysctl.conf	TCP最大半连接数，SYN_RECV 队列中的最大连接数。 查看参数： `sysctl net.ipv4.tcp_max_syn_backlog`	net.ipv4.tcp_max_syn_backlog=8096
tcp_max_tw_buckets	/etc/sysctl.conf	指定任何时候允许存在的处于“time-wait”状态的最大套接字数，参数值过大时易耗尽节点资源。 查看参数： `sysctl net.ipv4.tcp_max_tw_buckets`	net.ipv4.tcp_max_tw_buckets=5000
net.core.somaxconn	/etc/sysctl.conf	TCP最大连接数，ESTABLISHED 队列的最大大小，参数值过小时极易不足。 查看参数： `sysctl net.core.somaxconn`	net.core.somaxconn=32768
max_user_instances	/etc/sysctl.conf	每个用户允许的最大 inotify 实例数，参数值过小时容器场景下极易不足。 查看参数： `sysctl fs.inotify.max_user_instances`	fs.inotify.max_user_instances=8192
max_user_watches	/etc/sysctl.conf	所有监视实例的最大目录数，参数值过小时容器场景极易不足。 查看参数： `sysctl fs.inotify.max_user_watches`	fs.inotify.max_user_watches=524288
netdev_max_backlog	/etc/sysctl.conf	网络协议栈收包队列大小，参数值过小时极易不足。 查看参数： `sysctl net.core.netdev_max_backlog`	net.core.netdev_max_backlog=16384

net.core.wmem_max
net.core.rmem_max
	/etc/sysctl.conf	收发缓冲区内存大小（字节），参数值过小时大文件场景下易不足。
		查看参数：
		`sysctl net.core.wmem_max`
		`sysctl net.core.rmem_max`
			net.core.wmem_max=16777216
			net.core.rmem_max=16777216

参数名称	参数位置	说明	建议值

net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh3
	/etc/sysctl.conf	ARP表项的垃圾回收调优。
		• gc_thresh1：表示最小可保留的表项数量，如果表项数量小于此值GC（Garbage collector）不进行回收操作。**请勿修改。**
		• gc_thresh2：当表项数量超过此值时，GC将会清空大于5秒的表项。**请勿修改。**
		• gc_thresh3：最大可允许的非永久表项数量。如果系统拥有庞大的接口数量，或者直连了大量的设备，应增大此值。
		查看参数：
		`sysctl net.ipv4.neigh.default.gc_thresh1`
		`sysctl net.ipv4.neigh.default.gc_thresh2`
		`sysctl net.ipv4.neigh.default.gc_thresh3`
			net.ipv4.neigh.default.gc_thresh1=0
			net.ipv4.neigh.default.gc_thresh2=4096
			net.ipv4.neigh.default.gc_thresh3=163790

vm.max_map_count	/etc/sysctl.conf	参数值过小时，安装elk时会提示不足。 查看参数： `sysctl vm.max_map_count`	vm.max_map_count=262144

修改节点内核参数

节点内核参数如表3-73所示，此处以修改TCP发送keepalive探测消息的间隔时间tcp_keepalive_time为例。

步骤1 登录节点，查看/etc/sysctl.conf文件。

```
cat /etc/sysctl.conf
```

步骤2 修改net.ipv4.tcp_keepalive_time参数，**net.ipv4.tcp_keepalive_time=600**为内核参数名称及建议值，建议值请参考表3-73。

如需修改其他内核参数，请参考表3-73，替换命令中的参数名称及参数值。

- 若查看sysctl.conf文件时，文件中已设置net.ipv4.tcp_keepalive_time值，可通过以下命令进行修改。

  ```
  sed -i "s/net.ipv4.tcp_keepalive_time=[0-9]*$/net.ipv4.tcp_keepalive_time=600/g" /etc/sysctl.conf && sysctl -p
  ```

- 若查看sysctl.conf文件时，文件中还未设置net.ipv4.tcp_keepalive_time值，可通过以下命令添加。

  ```
  echo net.ipv4.tcp_keepalive_time=600 >> /etc/sysctl.conf && sysctl -p
  ```

步骤3 执行表3-73中的查看参数命令检查是否修改成功，当返回与修改值一致时说明修改正确。

```
# sysctl net.ipv4.tcp_keepalive_time
net.ipv4.tcp_keepalive_time = 600
```

----结束

创建节点/节点池时自动配置内核参数

您可以设置节点或节点池安装后执行脚本，在新建节点或节点池时通过脚本配置内核参数。

此处以修改TCP发送keepalive探测消息的间隔时间tcp_keepalive_time为例，取值为**表3-73**中的建议值。

步骤1 首先您需要确认创建节点或节点池的操作系统，例如CentOS 7.6。

步骤2 在**同集群、同操作系统**的节点上进行脚本命令可行性的测试，在节点上手动执行命令，确认脚本命令可行。手动执行脚本命令请参考**修改节点内核参数**。

步骤3 **（以下命令需在手动执行命令验证成功后配置）**在创建节点或节点池时，在“高级配置 > 安装后执行脚本”中添加可执行的脚本命令。如需修改其他内核参数，请参考**表3-73**，替换命令中的参数名称及参数值。

- 登录节点查看/etc/sysctl.conf文件，若文件中已设置net.ipv4.tcp_keepalive_time值，可通过以下命令进行修改。
  ```
  sed -i "s/net.ipv4.tcp_keepalive_time=[0-9]*$/net.ipv4.tcp_keepalive_time=600/g" /etc/sysctl.conf && sysctl -p
  ```
- 登录节点查看/etc/sysctl.conf文件，若文件中还未设置net.ipv4.tcp_keepalive_time值，可通过以下命令添加。
  ```
  echo net.ipv4.tcp_keepalive_time=600 >> /etc/sysctl.conf && sysctl -p
  ```

下图中命令仅做示例，请根据实际情况填写。

步骤4 节点创建完成后，登录节点执行**表3-73**中的查看参数命令检查是否修改成功。

----结束

3.9.6.5 修改节点进程 ID 数量上限 kernel.pid_max

背景信息

进程 ID（PID）是节点上的一种基础资源，容易在尚未超出其它资源约束的时候触及进程ID数量上限，进而导致节点不稳定。

您可以根据实际业务需求调整进程ID数量上限。

默认 kernel.pid_max 说明

CCE在2022年1月底将1.17及以上集群的节点公共操作系统EulerOS 2.5、CentOS 7.6、Ubuntu 18.04镜像kernel.pid_max默认值调整为4194304，满足如下两个条件节点的kernel.pid_max值为4194304。

- 集群版本：1.17.17及以上版本
- 节点创建时间：2022年1月30日之后

如果不满足如上两个条件，EulerOS 2.5、CentOS 7.6、Ubuntu 18.04上kernel.pid_max默认值32768。

表 3-74 节点 kernel.pid_max 默认值

操作系统	1.17.9及以下版本集群	1.17.17及以上版本集群	
		2022年1月30日及之前创建的节点	2022年1月30日之后创建的节点
EulerOS 2.5	32768	32768	4194304
CentOS 7.6	32768	32768	4194304
Ubuntu 18.04	不涉及	32768	4194304
EulerOS 2.3	57344	57344	57344
EulerOS 2.9	不涉及	4194304	4194304

修改建议

- EulerOS 2.3：所有节点都涉及，建议您将kernel.pid_max取值修改为4194304，具体方法请参见**修改节点kernel.pid_max**。且后续创建节点和节点池时配置安装前脚本修改kernel.pid_max，具体方法请参见**配置节点池kernel.pid_max**和**创建节点时配置kernel.pid_max**
- EulerOS 2.5、CentOS 7.6、Ubuntu 18.04：
  - 对于1.17.17及以上版本集群2022年1月30日及之前创建的节点，建议您将kernel.pid_max取值修改为4194304，具体方法请参见**修改节点kernel.pid_max**。
  - 对于1.17.9及以下版本集群
    - 存量节点建议您将kernel.pid_max取值修改为4194304，具体方法请参见**修改节点kernel.pid_max**。
    - 如果新创建节点和节点池，建议配置安装前脚本修改kernel.pid_max，具体方法请参见**配置节点池kernel.pid_max**和**创建节点时配置kernel.pid_max**。

查看节点 kernel.pid_max

登录节点，执行如下命令查看节点kernel.pid_max。

sysctl kernel.pid_max

```
# sysctl kernel.pid_max
kernel.pid_max = 32768
```

如果节点kernel.pid_max大小不能满足您的业务诉求，您可以修改kernel.pid_max大小，具体方法请参见**修改节点kernel.pid_max**。

查看节点当前 pid 用量

登录节点，执行如下命令可查看当前pid用量。

ps -eflL | wc -l

```
# ps -eflL | wc -l
691
```

修改节点 kernel.pid_max

登录节点，执行如下命令修改，其中4194304为kernel.pid_max大小，可根据实际业务需求修改。

echo kernel.pid_max = *4194304* >> /etc/sysctl.conf && sysctl -p

echo *4194304* > /sys/fs/cgroup/pids/kubepods/pids.max

执行如下命令检查是否修改成功，当返回值与修改值一致时说明修改正确。

```
# sysctl kernel.pid_max
kernel.pid_max = 4194304
# cat /sys/fs/cgroup/pids/kubepods/pids.max
4194304
```

配置节点池 kernel.pid_max

EulerOS 2.3：建议配置。

EulerOS 2.5、CentOS 7.6、Ubuntu 18.04：1.17.9及以下版本集群建议配置；1.17.17及以上版本当前已在操作系统镜像中将kernel.pid_max调整为4194304，无需配置。

您可以设置节点池安装前执行脚本，在节点池中新创建节点时通过脚本配置kernel.pid_max大小。

在创建节点池时，在“高级配置 > 安装后执行脚本”中添加如下命令。

echo kernel.pid_max = *4194304* >> /etc/sysctl.conf && sysctl -p

创建节点时配置 kernel.pid_max

EulerOS 2.3：建议配置。

EulerOS 2.5、CentOS 7.6、Ubuntu 18.04：1.17.9及以下版本集群建议配置；1.17.17及以上版本当前已在操作系统镜像中将kernel.pid_max调整为4194304，无需配置。

您可以设置节点安装前执行脚本，在新创建节点时通过脚本配置kernel.pid_max大小。

在创建节点时，在“高级配置 > 安装后执行脚本”中添加如下命令。

echo kernel.pid_max = *4194304* >> /etc/sysctl.conf && sysctl -p

3.9.7 配置节点故障检测策略

节点故障检查功能依赖**node-problem-detector（简称：npd）**，npd是一款集群节点监控插件，插件实例会运行在每个节点上。本文介绍如何开启节点故障检测能力。

前提条件

集群中已安装**CCE节点故障检测**插件。

开启节点故障检测

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧选择“节点管理”，切换至“节点”页签，检查集群中是否已安装npd插件，或将其升级至最新版本。npd安装成功后，可正常使用故障检测策略功能。

步骤3 npd运行正常时，单击“故障检测策略”，可查看当前故障检测项。关于NPD检查项列表请参见**NPD检查项**。

步骤4 当前节点检查结果异常时，将在节点列表处提示“指标异常”。

步骤5 您可单击“指标异常”，按照修复建议提示修复。

----结束

自定义检查项配置

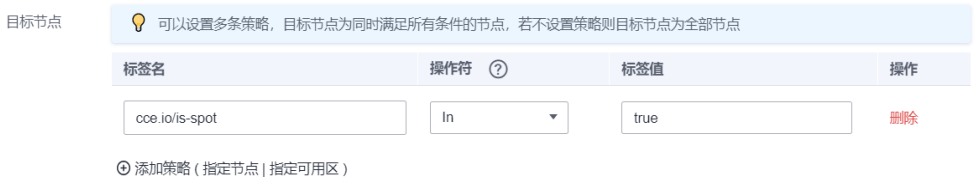
步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧选择“节点管理”，切换至“节点”页签，单击“故障检测策略”。

步骤3 在跳转的页面中查看当前检查项配置，单击检查项操作列的“编辑”，自定义检查项配置。

当前支持以下配置：

- 启用/停用：自定义某个检查项的开启或关闭。
- 目标节点配置：检查项默认运行在全部节点，用户可根据特殊场景需要自定义修改故障阈值。例如竞价实例中断回收检查只运行在竞价实例节点。



- 触发阈值配置：默认阈值匹配常见故障场景，用户可根据特殊场景需要自定义修改故障阈值。例如调整“连接跟踪表耗尽”触发阈值由90%调整至80%。

- 检查周期：默认检查周期为30秒，可根据用户场景需要自定义修改检查周期。

- 故障应对策略：故障产生后，可根据用户场景自定义修改故障应对策略，当前故障应对策略如下：

  表 3-75 故障应对策略

故障应对策略	效果
提示异常	上报Kuberentes事件。
禁止调度	上报Kuberentes事件，并为节点添加NoSchedule污点。
驱逐节点负载	上报Kuberentes事件，并为节点添加NoExecute污点。该操作会驱逐节点上的负载，可能导致业务不连续，请谨慎选择。

----结束

NPD 检查项

说明

当前检查项仅1.16.0及以上版本支持。

NPD的检查项主要分为事件类检查项和状态类检查项。

- 事件类检查项

  对于事件类检查项，当问题发生时，NPD会向APIServer上报一条事件，事件类型分为Normal（正常事件）和Warning（异常事件）

  表 3-76 事件类检查项

故障检查项	功能	说明
OOMKilling	监听内核日志，检查OOM事件发生并上报 典型场景：容器内进程使用的内存超过了Limt，触发OOM并终止该进程	Warning类事件 监听对象：/dev/kmsg 匹配规则："Killed process \\d+ (.+) total-vm:\\d+kB, anon-rss:\\d+kB, file-rss:\\d+kB.*"
TaskHung	监听内核日志，检查taskHung事件发生并上报 典型场景：磁盘卡IO导致进程卡住	Warning类事件 监听对象：/dev/kmsg 匹配规则："task \\S+:\\w+ blocked for more than \\w+ seconds\\."

故障检查项	功能	说明
ReadonlyFilesystem	监听内核日志，检查系统内核是否有Remount root filesystem read-only错误 典型场景：用户从ECS侧误操作卸载节点数据盘，且应用程序对该数据盘的对应挂载点仍有持续写操作，触发内核产生IO错误将磁盘重挂载为只读磁盘。 **说明** 节点容器存储Rootfs为Device Mapper类型时，数据盘卸载会导致thinpool异常，影响NPD运行，NPD将无法检测节点故障。	Warning类事件 监听对象：/dev/kmsg 匹配规则："Remounting filesystem read-only"

- 状态类检查项

  对于状态类检查项，当问题发生时，NPD会向APIServer上报一条事件，并同步修改节点状态，可配合**Node-problem-controller故障隔离**对节点进行隔离。

  下列检查项中若未明确指出检查周期，则默认周期为30秒。

  表 3-77 系统组件检查

故障检查项	功能	说明
容器网络组件异常 CNIProblem	检查CNI组件（容器网络组件）运行状态	无
容器运行时组件异常 CRIProblem	检查节点CRI组件（容器运行时组件）Docker和Containerd的运行状态	检查对象：Docker或Containerd
Kubelet频繁重启 FrequentKubeletRestart	通过定期回溯系统日志，检查关键组件Kubelet是否频繁重启	• 默认阈值：10分钟内重启10次 即在10分钟内组件重启10次表示频繁重启，将会产生故障告警。 • 监听对象：/run/log/journal目录下的日志 **说明** Ubuntu和HCE2.0操作系统由于日志格式不兼容，暂不支持上述检查项。
Docker频繁重启 FrequentDockerRestart	通过定期回溯系统日志，检查容器运行时Docker是否频繁重启	
Containerd频繁重启 FrequentContainerdRestart	通过定期回溯系统日志，检查容器运行时Containerd是否频繁重启	
Kubelet服务异常 KubeletProblem	检查关键组件Kubelet的运行状态	无
KubeProxy异常 KubeProxyProblem	检查关键组件KubeProxy的运行状态	无

表 3-78 系统指标

故障检查项	功能	说明
连接跟踪表耗尽 ConntrackFullProblem	检查连接跟踪表是否耗尽	• 默认阈值:90% • 使用量:nf_conntrack_count • 最大值:nf_conntrack_max
磁盘资源不足 DiskProblem	检查节点系统盘、CCE数据盘（包含CRI逻辑盘与Kubelet逻辑盘）的磁盘使用情况	• 默认阈值：90% • 数据来源: `df -h` 当前暂不支持额外的数据盘
文件句柄数不足 FDProblem	检查系统关键资源FD文件句柄数是否耗尽	• 默认阈值：90% • 使用量：/proc/sys/fs/file-nr中第1个值 • 最大值：/proc/sys/fs/file-nr中第3个值
节点内存资源不足 MemoryProblem	检查系统关键资源Memory内存资源是否耗尽	• 默认阈值：80% • 使用量：/proc/meminfo中MemTotal-MemAvailable • 最大值：/proc/meminfo中MemTotal
进程资源不足 PIDProblem	检查系统关键资源PID进程资源是否耗尽	• 默认阈值：90% • 使用量：/proc/loadavg中nr_threads • 最大值：/proc/sys/kernel/pid_max和/proc/sys/kernel/threads-max两者的较小值。

表 3-79 存储检查

故障检查项	功能	说明
磁盘只读 DiskReadonly	通过定期对节点系统盘、CCE数据盘（包含CRI逻辑盘与Kubelet逻辑盘）进行测试性写操作，检查关键磁盘的可用性	检测路径： • /mnt/paas/kubernetes/kubelet/ • /var/lib/docker/ • /var/lib/containerd/ • /var/paas/sys/log/cceaddon-npd/ 检测路径下会产生临时文件npd-disk-write-ping 当前暂不支持额外的数据盘
节点emptydir存储池异常 EmptyDirVolumeGroupStatusError	检查节点上临时卷存储池是否正常 故障影响：依赖存储池的Pod无法正常写对应临时卷。临时卷由于IO错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为临时卷存储池，用户误操作删除了部分数据盘导致存储池异常。	• 检测周期：30秒 • 数据来源：`vgs -o vg_name, vg_attr` • 检测原理：检查VG（存储池）是否存在p状态，该状态表征部分PV（数据盘）丢失。 • 节点持久卷存储池异常调度联动：调度器可自动识别此异常状态并避免依赖存储池的Pod调度到该节点上。 • 例外场景：NPD无法检测所有PV（数据盘）丢失，导致VG（存储池）丢失的场景；此时依赖kubelet自动隔离该节点，其检测到VG（存储池）丢失并更新nodestatus.allocatable中对应资源为0，避免依赖存储池的Pod调度到该节点上。无法检测单个PV损坏；此时依赖ReadonlyFilesystem检测异常。
节点持久卷存储池异常 LocalPvVolumeGroupStatusError	检查节点上持久卷存储池是否正常 故障影响：依赖存储池的Pod无法正常写对应持久卷。持久卷由于IO错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为持久卷存储池，用户误操作删除了部分数据盘。	

故障检查项	功能	说明
挂载点异常 MountPointProblem	检查节点上的挂载点是否异常 异常定义：该挂载点不可访问（cd） 典型场景：节点挂载了nfs（网络文件系统，常见有obsfs、s3fs等），当由于网络或对端nfs服务器异常等原因导致连接异常时，所有访问该挂载点的进程均卡死。例如集群升级场景kubelet重启时扫描所有挂载点，当扫描到此异常挂载点会卡死，导致升级失败。	等效检查命令： `for dir in `df -h | grep -v "Mounted on" | awk "{print \\$NF}"`;do cd $dir; done && echo "ok"`
磁盘卡IO DiskHung	检查节点上所有磁盘是否存在卡IO，即IO读写无响应 卡IO定义：系统对磁盘的IO请求下发后未有响应，部分进程卡在D状态 典型场景：操作系统硬盘驱动异常或底层网络严重故障导致磁盘无法响应	• 检查对象：所有数据盘 • 数据来源：/proc/diskstat 等效查询命令：`iostat -xmt 1` • 阈值（需同时满足）： – 平均利用率（ioutil）>= 0.99 – 平均IO队列长度（avgqu-sz）>=1 – 平均IO传输量 <= 1 平均IO传输量 = 每秒完成写次数（iops，单位为w/s）+每秒写数据量（ioth，单位为wMB/s） **说明** 部分操作系统卡IO时无数据变化，此时计算CPU IO时间占用率，iowait > 0.8。

故障检查项	功能	说明
磁盘慢IO DiskSlow	检查节点上所有磁盘是否存在慢IO，即IO读写有响应但响应缓慢 典型场景：云硬盘由于网络波动导致慢IO。	• 检查对象：所有数据盘 • 数据来源：/proc/diskstat 等效查询命令 `iostat -xmt 1` • 默认阈值：平均IO时延，await >= 5000ms **说明** 卡IO场景下该检查项失效，原因为IO请求未有响应，await数据不会刷新。

表 3-80 其他检查

故障检查项	功能	说明
NTP异常 NTPProblem	检查节点时钟同步服务ntpd或chronyd是否正常运行，系统时间是否漂移	默认时钟偏移阈值：8000ms
进程D异常 ProcessD	检查节点是否存在D进程	默认阈值：连续3次存在10个异常进程 数据来源： • /proc/{PID}/stat • 等效命令：ps aux 例外场景：ProcessD忽略BMS节点下的SDI卡驱动依赖的常驻D进程heartbeat、update
进程Z异常 ProcessZ	检查节点是否存在Z进程	
ResolvConf配置文件异常 ResolvConfFileProblem	检查ResolvConf配置文件是否丢失 检查ResolvConf配置文件是否异常 异常定义：不包含任何上游域名解析服务器（nameserver）。	检查对象：/etc/resolv.conf
存在计划事件 ScheduledEvent	检查节点是否存在热迁移计划事件。热迁移计划事件通常由硬件故障触发，是IaaS层的一种自动故障修复手段。 典型场景：底层宿主机异常，例如风扇损坏、磁盘坏道等，导致其上虚机触发热迁移。	数据来源： • http://169.254.169.254/meta-data/latest/events/scheduled 该检查项为Alpha特性，默认不开启。

故障检查项	功能	说明
竞价节点中断回收中 SpotPriceNodeReclaimNotification	检查竞价实例节点是否被抢占而处于中断回收状态	默认检查周期：120秒 默认故障应对策略：驱逐节点负载

另外kubelet组件内置如下检查项，但是存在不足，您可通过集群升级或安装NPD进行补足。

表 3-81 Kubelet 内置检查项

故障检查项	功能	说明
PID资源不足 PIDPressure	检查PID是否充足	• 周期：10秒 • 阈值：90% • 缺点：社区1.23.1及以前版本，该检查项在pid使用量大于65535时失效，详见**issue 107107**。社区1.24及以前版本，该检查项未考虑thread-max。
内存资源不足 MemoryPressure	检查容器可分配空间（allocable）内存是否充足	• 周期：10秒 • 阈值：最大值-100MiB • 最大值（Allocable）：节点总内存-节点预留内存 • 缺点：该检测项没有从节点整体内存维度检查内存耗尽情况，只关注了容器部分(Allocable)。
磁盘资源不足 DiskPressure	检查kubelet盘和docker盘的磁盘使用量及inodes使用量	• 周期：10秒 • 阈值：90%

3.9.8 创建节点时执行安装前/后脚本

应用现状

在创建节点时，对于需要在节点上安装一些工具或者进行安全加固等操作时，可以使用安装前/后脚本实现。本文为您提供正确使用安装前/后脚本的指导，帮助您了解和使用安装前/后脚本。如果有进阶的安装脚本使用需求，可以将脚本存放在OBS中，避免脚本字符数超限等问题，详情请参见**创建节点时使用OBS桶实现自定义脚本注入**。

注意事项

- 请避免使用执行耗时过长的安装前/后脚本。

  安装前脚本的时间限制为15min、安装后脚本的时间限制为30min，如果指定时间内节点没能到达可用状态，则会触发节点的回收操作。因此需要避免执行耗时过长的安装前/后脚本。

- 请避免在安装后脚本中直接使用reboot指令。

  当前CCE会在执行完节点必备组件的安装之后，再执行安装后脚本。当安装后脚本执行完之后才会将节点状态置为可用状态。如果直接使用reboot命令，可能会导致节点在上报状态之前就被重启，从而造成节点无法在30min内到达运行中状态，触发超时回滚。因此请尽量避免使用reboot指令。

  如果确实需要重启节点，可以选择：

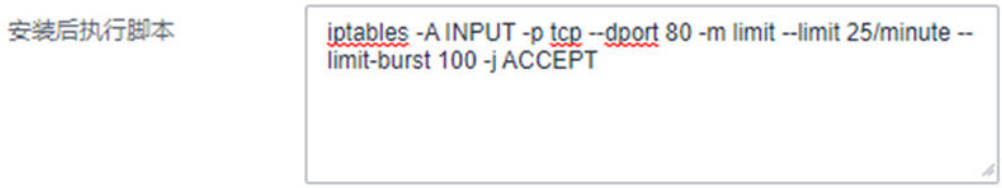
  - 在安装后脚本中使用**shutdown -r <时间>**命令，延迟重启。例如，使用**shutdown -r 1**命令可以延迟1分钟重启。
  - 在节点状态为可用状态之后，手动进行节点重启。

操作步骤

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

步骤2 在集群控制台左侧导航栏中选择“节点管理”，切换至“节点”页签，单击右侧“创建节点”，并设置节点参数。

步骤3 在“高级配置”中，填写安装前/后执行脚本。



例如，您可以通过安装后执行脚本创建iptables规则，限制每分钟最多只能有25个TCP协议的数据包通过端口80进入，并且在超过这个限制时，允许最多100个数据包通过，以防止DDoS攻击。

```
iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --limit-burst 100 -j ACCEPT
```

说明

此处的脚本示例仅供参考。

步骤4 完成以上配置后，您可以设置需要购买的节点数量，并单击“下一步：规格确认”。

步骤5 单击“提交”，开始创建节点。

----结束

4 节点池

4.1 节点池概述

简介

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

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

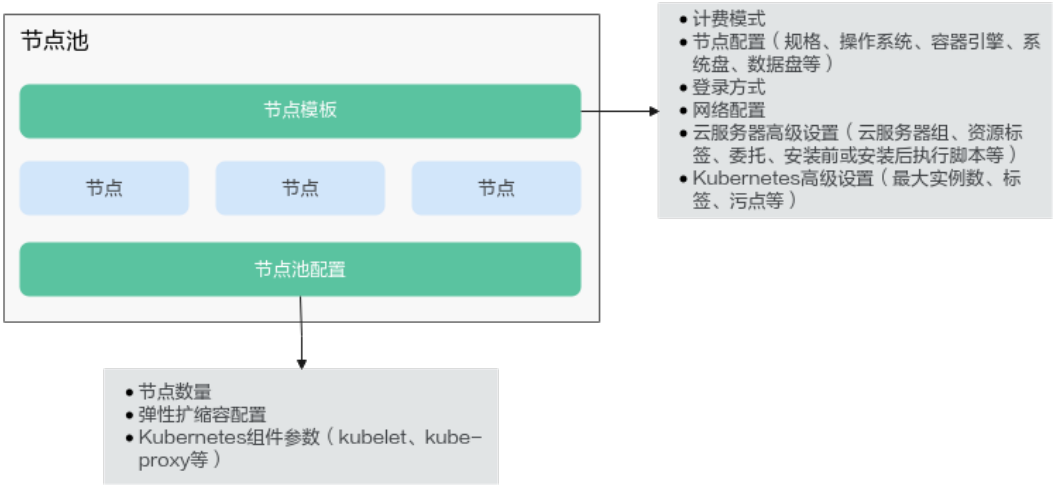
通过节点池功能您还可以实现节点的动态扩缩容（仅按需计费的节点池支持）：

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

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

节点池架构

图 4-1 节点池整体架构图



通常情况下，节点池内的节点均具有如下相同属性：

- 节点操作系统。
- 节点登录方式。
- 节点容器运行时。
- 节点所属企业项目。
- 节点Kubernetes组件启动参数。
- 节点自定义启动脚本。
- 节点“K8s标签”及“污点”设置。

此外，CCE将同时围绕节点池扩展以下属性：

- 节点池级别操作系统。
- 节点池级别每节点的Pod数上限。

节点池升级说明

对于v1.21.11-r0、v1.23.9-r0、v1.25.4-r0及以上版本的集群，新建的节点池不再区分计费模式，默认支持创建按需和包周期的节点，为您提供更优秀的资源管理体验。

新节点池的优势如下：

- 同一个节点池中支持创建按需节点和包周期节点，并且支持选择不同的包年包月周期。
- 支持纳管按需/包周期节点。
- 支持开启弹性伸缩，配置丰富的伸缩策略，为资源管理带来更高效、更灵活的体验。

对于已存在的节点池，升级为新节点池后将存在以下行为变更：

原节点池类型	新节点池变化
按需节点池	新节点池将自动继承按需节点池的全量能力。 此外，新节点池中创建的包周期节点不支持手动缩容，仅支持退订和移除。
包周期节点池	您可以将原有的包周期节点池无损切换到新节点池，并且不会对节点池下已有的节点产生任何影响。 此外，包周期节点池切换到新节点池后，行为变化如下： ● 默认创建的节点不再是包周期节点，而是按需节点。 ● 节点池中的包周期节点不再支持手动缩容，仅支持退订和移除。 ● 将支持开启弹性伸缩功能，且弹性伸缩默认扩缩容的节点为按需节点，不会扩缩容包周期节点。 ● 不再支持使用更新节点池接口创建包周期节点，如果您需要新建包周期节点，请使用扩缩容节点池接口。

默认节点池 DefaultPool 说明

DefaultPool并非一个真正的节点池，只是将非自定义节点池中的节点做一个**归类**，所有非自定义节点池中创建的节点（直接在控制台创建的节点或调用API创建的节点）都会归类在DefaultPool中。DefaultPool不具备任何自定义节点池的功能，包括弹性伸缩、各项参数设置等，且不可编辑、删除或迁移，也不支持扩容、弹性伸缩。

应用场景

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

下表介绍了多种大规模集群管理场景，并分别展示节点池在每种场景下发挥的作用：

表 4-1 节点池场景及作用

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

功能点及注意事项

功能点	功能说明	注意事项
创建节点池	新增节点池。	单个集群不建议超过100个节点池。
删除节点池	删除节点池时会先删除节点池中的节点，原有节点上的工作负载实例会自动迁移至其他节点池的可用节点。	如果工作负载实例具有特定的节点选择器，且如果集群中的其他节点均不符合标准，则工作负载实例可能仍处于无法安排的状态。
节点池开启弹性伸缩	开启弹性伸缩后，节点池将根据集群负载情况自动创建或删除节点池内的节点。	节点池中的节点建议不要放置重要数据，以防止节点被弹性缩容，数据无法恢复。
节点池关闭弹性伸缩	关闭弹性伸缩后，节点池内节点数量不随集群负载情况自动调整。	/
调整节点池大小	支持直接调整节点池内节点个数。若减小节点数量，将从现有节点池内随机缩容节点。	开启弹性伸缩后，不建议手动调整节点池大小。

功能点	功能说明	注意事项
调整节点池配置	可修改节点池名称、节点个数，删除或新增K8s标签、污点及资源标签，调整节点池磁盘配置、操作系统、容器引擎等配置。	删除或新增K8s标签和污点会对节点池内节点全部生效，可能会引起Pod重新调度，请谨慎变更。
移出节点池内节点	可以将同一个集群下某个节点池中的节点迁移到默认节点池（DefaultPool）中	暂不支持将默认节点池（DefaultPool）中的节点迁移到其他节点池中，也不支持将自定义节点池中的节点迁移到其他自定义节点池。
复制节点池	可以方便地复制现有节点池的配置，从而创建新的节点池。	/
配置Kubernetes参数	通过该功能您可以对核心组件进行深度配置。	• 本功能仅支持在v1.15及以上版本的集群中对节点池进行配置，v1.15以下版本不显示该功能。 • 默认节点池DefaultPool不支持修改该类配置。

将工作负载部署到特定节点池

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

您也可以为容器指定资源请求，工作负载将仅在满足资源请求的节点上运行。

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

相关操作

您可以登录CCE控制台并参考以下文档， 进行节点池对应的操作：

- **创建节点池**
- **管理节点池**
- **创建无状态负载（Deployment）**
- **设置节点亲和调度（nodeAffinity）**

4.2 新版节点池切换说明

升级后的节点池，不仅完美融合了按需和包年/包月节点的灵活性，更在原有的全量能力基础上进一步增强配置管理，为您的资源管理带来更高效、更灵活的体验。

为什么要切换新版节点池?

- **灵活的资源配置：**节点池提供更加灵活的节点类型，允许您根据即时需求创建按需节点，也可以选择成本效益更高的包年/包月节点。
- **多元的实例选择：**您可以基于业务需求，通过CPU和内存等参数筛选多种实例规格（如GPU实例），从而满足不同业务场景的需要。
- **高级的弹性伸缩：**节点池支持开启弹性伸缩功能，您可以配置多种伸缩策略来应对不同的业务场景，从而提高资源利用率。
- **增强的配置管理：**节点池进一步增强了Kubernetes参数的自定义配置能力，提供了更多的选项和指导，以满足对容器化应用管理的复杂需求。

新版节点池有什么变化?

- 新版节点池中的包周期节点不再支持手动缩容，仅支持退订和移除。
- 弹性伸缩默认扩缩容的节点为按需节点，不会缩容包周期节点。
- 不再支持通过更新节点池接口创建包周期节点，需要通过扩缩容新建包周期节点。
- 包周期节点池中默认创建的节点不再是包周期节点，而是按需节点。

表 4-2 节点池升级前后的行为对比

类型	升级前		升级后
	按需节点池	包年/包月节点池	
支持的节点类型	仅按需	仅包年/包月	同时支持按需和包年/包月
扩容节点方式	更新节点池配置中的节点数量	更新节点池配置中的节点数量	扩缩容节点池
缩容节点方式	• 更新节点池配置中的节点数量 • 删除节点/移除节点	• 更新节点池配置中的节点数量 • 退订节点/移除节点	按需节点：扩缩容 包周期节点：退订节点/移除节点
升级方式	集群升级到v1.21.11-r0、v1.23.9-r0、v1.25.4-r0及以上版本即可支持。 • 按需节点池升级至新节点池：无需操作，自动继承新节点池的全量功能。 • 包年/包月节点池升级至新节点池：在扩缩容节点数时触发，确认升级后将无损切换到新节点池，并且不会对节点池下已有的节点产生任何影响。 **说明** 节点池升级后，如果仍通过更新节点池API扩容节点，默认创建出来的节点为按需节点。		

如何轻松切换新版节点池?

您需要将集群升级到1.21.11-r0、1.23.9-r0、1.25.4-r0及以上版本，然后根据以下步骤触发新节点池升级流程。

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“扩缩容”，在弹出框中单击“立即升级”即可完成升级。

----结束

4.3 创建节点池

操作场景

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

操作步骤

步骤1 登录**CCE控制台**。

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

步骤3 单击右上角“创建节点池”。

基础配置

表 4-3 基础配置

参数	参数说明
节点池名称	新建节点池的名称，默认按“集群名-nodepool-随机数”生成名称，可自定义。
企业项目	该参数仅对开通企业项目的企业客户账号显示，且集群版本要求v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上。 选择某个企业项目后，节点池下的节点将会创建在该企业项目下。您可以通过企业项目服务（EPS）管理集群及其他资源（节点、ELB、以及节点的安全组等）。了解更多企业项目相关信息，请查看**企业管理**。

节点配置：

配置节点云服务器的规格与操作系统，为节点上的容器应用提供基本运行环境。

表 4-4 节点配置参数

参数	参数说明
节点类型	请根据不同的业务诉求选择节点类型，然后您可以在“节点规格”列表中进一步选择合适的规格。 CCE Standard集群支持以下类型： ● 弹性云服务器-虚拟机：使用虚拟化技术的弹性云服务器作为集群节点。 ● 弹性云服务器-物理机：使用擎天架构的裸金属服务器作为集群节点。 ● 裸金属服务器：使用传统裸金属服务器作为集群节点。选择数据盘时支持使用裸金属服务器自带的本地盘。 CCE Turbo集群支持以下类型： ● 弹性云服务器-虚拟机：使用虚拟化的弹性云服务器作为集群节点。CCE Turbo集群仅支持可添加多张弹性网卡的机型，请根据控制台页面展示规格进行选择。 ● 弹性云服务器-物理机：使用擎天架构的裸金属服务器作为集群节点。
节点规格	请根据业务需求选择相应的节点规格，不同区域/可用区支持的节点规格不同，请以CCE控制台呈现为准。 **说明** ● 节点池同时配置多个节点规格时，同一节点池仅支持同类型节点规格（可位于不同可用区）。例如，选择“通用计算增强型”的节点池只支持选择该类型下的规格，不支持选择“通用计算型”等其他类型的规格。 ● 一个节点池最多可添加 10 种节点规格配置（每个可用区为一条配置）。添加同一个节点规格时，支持选择不同可用区。添加节点规格时需要明确指定可用区，不支持随机可用区。 ● 新创建的节点池，仅按照默认规格创建节点，当默认规格资源不足时，会导致节点创建失败。 ● 节点池创建后，已存在节点的规格不可删除。
容器引擎	CCE支持Docker和Containerd容器引擎，不同的集群类型、集群版本、操作系统可能导致支持的容器引擎类型不同，请根据控制台呈现进行选择。具体场景请参见**节点操作系统与容器引擎对应关系**。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 ● 公共镜像：请选择节点对应的操作系统。 ● 私有镜像：支持使用私有镜像，私有镜像制作方法具体请参见**制作CCE节点自定义镜像**。 **说明** 由于业务容器运行时共享节点的内核及底层调用，为保证兼容性，建议节点的操作系统选择与最终业务容器镜像相同或接近的Linux发行版本。

参数	参数说明
登录方式	• **密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 • **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。 • **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持） 保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。

存储配置：

配置节点云服务器上的存储资源，方便节点上的容器软件与容器应用使用。请根据实际场景设置磁盘类型及大小。关于云硬盘类型的详细介绍请参见**磁盘类型及性能介绍**。

表 4-5 存储配置参数

参数	参数说明
系统盘	节点云服务器使用的系统盘，供操作系统使用。您可以设置系统盘的规格为40GiB-1024GiB之间的数值，缺省值为50GiB。 **说明** 通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。仅v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群中支持选择使用通用型SSD V2类型的云硬盘。 系统盘加密：系统盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**仅“弹性云服务器-虚拟机”类型支持系统盘加密，且仅部分Region支持此选项，具体请以界面为准。** • 默认不加密。 • 选择“加密-从密钥中选择”后，可选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 • 选择“加密-输入KMS密钥ID”后，您需要输入的KMS密钥ID（包含他人共享的KMS密钥），且该密钥必须位于当前Region下。

参数	参数说明
系统组件存储	选择系统组件的存储位置： • 数据盘：必须添加一块默认数据盘，供容器运行时和Kubelet组件使用，您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。该数据盘不能被删除卸载，否则会导致节点不可用。 • 系统盘：CCE将下载的镜像、容器的临时存储、容器的stdout标准输出日志等资源都存储在系统盘中，过多占用系统盘可能影响节点运行稳定性。 **说明** v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中支持选择系统组件的存储位置，且配套使用**CCE节点故障检测**插件时需安装1.19.2及以上版本的插件。

参数	参数说明
数据盘	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，**至少需要一块默认数据盘**，供容器运行时和Kubelet组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** ● 默认数据盘：供容器运行时和Kubelet组件使用。您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。 ● 其他普通数据盘，您可以设置数据盘的规格为10GiB-32768GiB之间的数值，缺省值为100GiB。 v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0及以上版本的集群中，如果“系统组件存储”选择“系统盘”，则可以不添加默认数据盘。所有数据盘均为普通数据盘，您可以设置数据盘的规格为10GiB-32768GiB之间的数值，缺省值为100GiB。 **说明** ● 节点规格为“磁盘增强型”或“超高I/O型”时，有一块数据盘可以是本地盘。 ● 本地磁盘实例有宕机风险，不保证数据可靠性，建议您使用云硬盘存储您的业务数据。 ● 通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。仅v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群中支持选择使用通用型SSD V2类型的云硬盘。 **高级配置** 单击后方的“展开高级设置”可进行如下设置： ● 数据盘空间分配：对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 ● 数据盘加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**“裸金属服务器”类型节点不支持数据盘加密，且仅部分Region支持此选项，具体请以界面为准。** – 默认不加密。 – 选择“加密-从密钥中选择”后，可选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 – 选择“加密-输入KMS密钥ID”后，您需要输入的KMS密钥ID（包含他人共享的KMS密钥），且该密钥必须位于当前Region下。 **增加数据盘** 弹性云服务器最多可以添加16块，裸金属服务器最多可以添加10块。默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，选择如下配置。 ● 默认：默认情况直接创建为裸盘，不做任何处理。 ● 挂载到指定目录：将数据盘挂载到指定目录。

参数	参数说明
	• 作为持久存储卷：适用于对PV有性能要求的场景。持久存储卷的节点会添加上node.kubernetes.io/local-storage-persistent标签，取值为linear或striped。 • 作为临时存储卷：适用于对EmptyDir有性能要求的场景。 **说明** • 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。 • 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=1.2.29。 **本地持久卷**和**本地临时卷**支持如下两种写入模式。 • 线性（linear）：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。 • 条带化（striped）：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。多块存储卷才能选择条带化。

网络配置：

配置节点云服务器的网络资源，用于访问节点和容器应用。

表 4-6 网络配置参数

参数	参数说明
虚拟私有云	默认为集群所在VPC，不可修改。
节点子网	节点子网默认使用创建集群时的子网配置，也可以选择其他子网。 • 多个子网：可选择同一VPC下的多个子网作为节点可用网段，扩容节点会优先消耗排序靠前的子网IP资源。 • 单个子网：当节点池关联的单个子网IP资源较为紧张时，推荐配置多个子网，否则可能会出现节点池扩容失败的问题。
节点IP	支持随机分配。
关联安全组	指定节点池创建出来的节点使用哪个安全组。最多选择5个安全组。 创建集群时会默认创建一个节点安全组，名称为**{集群名}-cce-node-{随机ID}**，默认会使用该安全组。 节点安全组需要放通一些端口以保障节点通信，如选择其他安全组，需要放通这些端口，具体请参见**集群安全组说明**。 **说明** 节点池创建完成后，关联安全组不可修改。

高级配置：

节点能力增强，可在此配置节点的标签、污点、启动命令等功能。

表 4-7 高级配置参数

参数	参数说明
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动帮您创建CCE-Dynamic-Provisioning-Node=节点id的标签。
K8S标签（Labels）	设置附加到Kubernetes对象（比如Pod）上的键值对，填写键值对后，单击“确认添加”。最多可以添加20条标签。 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。详细请参见**Labels and Selectors**。
K8S污点（Taints）	默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点，每条污点包含以下3个参数： ● Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符；另外可以使用DNS子域作为前缀。 ● Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符。 ● Effect：只可选NoSchedule，PreferNoSchedule或NoExecute。 污点的使用请参见**管理节点污点**。 **说明** 对于1.19及以下版本集群，有可能会出现污点打上之前负载已经调度到节点上，如果需要避免这种情况，请选择1.19及以上集群。
存量节点标签及污点	勾选后，更新节点池的资源标签、K8s标签及污点时，会将节点池配置中的资源标签、K8s标签或污点修改同步至节点池中已有的节点。

参数	参数说明
新增节点调度策略	节点池中新增节点的默认调度策略。勾选“设置为不可调度”之后，该节点池新增的节点在创建完成之后均会被打上不可调度的标签。以方便用户在工作负载被调度到节点上之前，对节点进行一些操作。 定时开启调度时间：若设置定时开启调度功能，在超过自定义时间后，节点将会自动开启调度。 • 不设置：默认情况下，将不设置超时时间，此时节点需要您前往“节点管理”界面，手动选择节点开启调度，详情请参见**一键设置节点调度策略**。 • 自定义：该参数可配置节点不可调度的默认超时时间，取值范围为0-99min。 **说明** • 如果需要同时使用节点池的自动扩缩容能力，定时开启调度时间应小于15min。因为通过autoscaler扩容的节点如果处于不可调度状态超过15min，autoscaler会认为本次扩容失败，触发再次扩容。同时，原节点处于不可调度状态超过20min，节点将被列入缩容备选节点，被autoscaler缩容。 • 开启该开关后，节点池的创建/更新过程中，节点会被打上node.cloudprovider.kubernetes.io/uninitialized的污点。
最大实例数	节点最大可以正常运行的实例数(Pod)，该数量包含系统默认实例，取值范围为16~256。 该设置的目的为防止节点因管理过多实例而负载过重，请根据您的业务需要进行设置。 节点最多能创建多少个Pod还受其他因素影响，具体请参见**节点可创建的最大Pod数量说明**。
云服务器组	云服务器组是对云服务器的一种逻辑划分，同一云服务器组中的云服务器遵从同一策略。 反亲和性策略：同一云服务器组中的云服务器分散地创建在不同主机上，提高业务的可靠性。 选择已创建的云服务器组，或单击“新建云服务器组”创建，创建完成后单击刷新按钮。
安装前执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装前执行，可能导致Kubernetes软件无法正常安装，需谨慎使用。
安装后执行脚本	请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。 脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。 **说明** 请不要在安装后执行脚本中使用**reboot**命令立即重启，如果需要重启，可以使用**shutdown -r 1**命令延迟1分钟重启。

参数	参数说明
委托	委托是由租户管理员在统一身份认证服务上创建的。通过委托，可以将云主机资源共享给其他账号，或委托更专业的人或团队来代为管理。 如果没有委托请单击右侧“新建委托”创建。
自定义节点名称前后缀	节点池下的节点名称自定义前后缀，支持配置前缀和后缀。配置完成之后，该节点池下的节点名称将带上配置的前后缀信息。例如前缀为prefix-，后缀为-suffix，那么最终该节点池下的节点名称为prefix-nodepoolName-五位随机数-suffix。 **须知** • 自定义前后缀名称前后缀仅支持创建节点池时指定，不支持修改。 • 前缀支持以特殊字符结尾，后缀支持以特殊字符开头。 • 节点名称由三部分组成：前缀+节点池名称-五位随机字符+后缀，总长度不超过56个字符。 • 节点名称中不支持“.”与特殊字符连用，例如“..”、“.-”、“-.”。 • 仅v1.28.1、v1.27.3、v1.25.6、v1.23.11、v1.21.12及以上集群版本支持该特性。
K8s节点名称	K8s节点名称，即节点YAML文件中的“metadata.labels.kubernetes.io/hostname”标签值，支持以下两种配置。 • 与节点私有IP保持一致：默认为节点私有IP。 • 与云服务器名称保持一致：将**节点配置**中配置的自定义云服务器名称作为K8s节点名称。由于云服务器名称可能存在重名，为避免K8s节点名称冲突，系统将在名称后自动添加五位随机字符后缀。 **须知** – 该功能在集群版本为v1.23.4-r0及以上时支持配置。 – 仅支持在创建（纳管）时将节点云服务器名称指定为K8s节点名称。创建（纳管）完成后，K8s节点名称无法修改，详情请参见**云服务器名称、节点名称与K8s节点名称说明**。 – 如您在创建（纳管）选择将云服务器名称指定为K8s节点名称，集群已有节点将仍使用私有IP作为K8s节点名称。 该场景下，存在部分K8s节点名称与私有IP不一致的情况，对于业务场景中将私有IP和K8s节点名称混用的场景，需做好适配。例如，在设置节点亲和调度时，不能将节点私有IP作为节点名称配置调度策略。 如您需要将已有节点的K8s节点名称统一为“与云服务器名称保持一致”，您可将已有节点从集群中移除，并重新纳管。执行节点移除、纳管操作前，请您充分了解节点**移除**及**纳管**可能带来的业务影响。

步骤4 单击“下一步：规格确认”，确认已阅读并知晓华为云的**镜像免责声明**。

步骤5 单击“提交”。

----结束

相关操作

节点池创建完成后节点总数默认为0，您需要手动选择规格扩容节点数，详情请参见**扩缩容节点池**。

4.4 扩缩容节点池

您可指定节点池中的某个规格进行扩缩容。

须知

默认节点池不支持扩缩容，请通过**创建节点**添加。

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“扩缩容”。

步骤4 在弹出的“节点池扩缩容”窗口中，设置扩缩容参数。

- 扩缩容：选择“扩容节点”或“缩容节点”。
- 扩容/缩容规格：使用选择的规格扩容或缩容节点。
- 计费模式：仅扩容节点时需选择。
  - 包年包月

    包年包月需要选择购买时长，还可以勾选自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。
  - 按需计费

    按资源的实际使用时长计费，可以随时开通/删除资源。
  - 竞价计费

    竞价计费是后付费模式，相对于按需计费模式，以更低的折扣按实际使用时长计费。详情请参见**竞价计费型实例**。

    说明

    - 如果创建竞价实例时同时购买了数据盘和弹性公网IP，数据盘和弹性公网IP会在竞价实例释放时随实例释放。如果给已经创建完成的竞价实例挂载数据盘和弹性公网IP，则需要在删除竞价实例后自行释放这些资源。
    - 竞价实例不支持重置、纳管、移除、迁移、转包周期、集群重置升级。如果要对集群进行重置升级，需要先删除竞价节点，将竞价节点池实例数设为0。
    - ECS可能会因为用户报价小于市场价、资源不足等原因主动释放竞价实例。建议在集群中安装最新版本的npd插件，npd插件会在竞价实例被ECS释放前5分钟收到通知，产生ReceivedReclaimNodeNotification事件，并给节点加污点**node-problem-controller.cce.io/SpotPriceNodeReclaimNotification: NoExecute**，驱逐节点上的Pod，使Pod能在节点被删除前迁移到其他节点。
- 本次扩容/缩容节点数：
  - 扩容时，本次需要扩容的节点数与已有节点数相加不可超过当前集群管理规模。

- 缩容时，本次需要缩容节点数不可超过已有节点数。

  缩容操作可能导致与节点有绑定关系的资源（本地存储，指定调度节点的负载等）无法正常使用。请谨慎操作，避免对运行中的业务造成影响。

步骤5 单击“确定”，即可完成节点池的扩缩容。

----结束

4.5 管理节点池

4.5.1 更新节点池

注意事项

- 仅v1.19及以上版本的集群支持修改容器引擎、操作系统、系统盘/数据盘大小、数据盘空间分配、安装前/后执行脚本配置。
- 修改节点池容器引擎、操作系统、安装前/后执行脚本时，修改后的配置仅对新增节点生效，存量节点如需同步配置，需要手动重置存量节点。
- 修改节点池系统盘/数据盘大小、数据盘空间分配则仅对新增节点生效，即使重置存量节点也无法同步配置。
- 修改资源标签、K8s标签和污点数据会根据“存量节点标签及污点”开关状态决定是否自动同步已有节点，无需重置节点。

更新节点池

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更新”，在弹出的“更新节点池”页面中配置参数。

基础配置

表 4-8 基础配置

参数	参数说明
节点池名称	自定义节点池名称。

节点配置

表 4-9 节点配置参数

参数	参数说明
节点规格	请根据业务需求选择相应的节点规格。 **说明** • 节点池同时配置多个节点规格时，同一节点池仅支持同类型节点规格（可位于不同可用区）。例如，选择“通用计算增强型”的节点池只支持选择该类型下的规格，不支持选择“通用计算型”等其他类型的规格。 • 一个节点池最多可添加 10 种节点规格配置（每个可用区为一条配置）。添加同一个节点规格时，支持选择不同可用区。添加节点规格时需要明确指定可用区，不支持随机可用区。 • 新创建的节点池，仅按照默认规格创建节点，当默认规格资源不足时，会导致节点创建失败。 • 节点池创建后，已存在节点的规格不可删除。
容器引擎	CCE支持Docker和Containerd容器引擎，不同的集群类型、集群版本、操作系统可能导致支持的容器引擎类型不同，请根据控制台呈现进行选择。具体场景请参见**节点操作系统与容器引擎对应关系**。 **说明** 修改“容器引擎”配置后，对新增的节点自动生效，存量节点需要手动重置节点后生效。
操作系统	选择操作系统类型，不同类型节点支持的操作系统有所不同。 • 公共镜像：请选择节点对应的操作系统。 • 私有镜像：支持使用私有镜像，私有镜像制作方法具体请参见**制作CCE节点自定义镜像**。 **说明** • 由于业务容器运行时共享节点的内核及底层调用，为保证兼容性，建议节点的操作系统选择与最终业务容器镜像相同或接近的Linux发行版本。 • 修改“操作系统”配置后，对新增的节点自动生效，存量节点需要手动重置节点后生效。

参数	参数说明
编辑登录方式	选择是否编辑登录方式，开启后支持修改节点登录方式。 • **密码** 用户名默认为“root”，请输入登录节点的密码，并确认密码。 登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。 • **密钥对** 选择用于登录本节点的密钥对，支持选择共享密钥。 密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。 • **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持） 保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。 **说明** 编辑“登录方式”后，对新增的节点自动生效，存量节点需要手动重置节点后生效。

存储配置

表 4-10 存储配置参数

参数	参数说明
系统盘	节点云服务器使用的系统盘，供操作系统使用。您可以设置系统盘的规格为40GiB-1024GiB之间的数值，缺省值为50GiB。 **说明** 修改系统盘“规格”配置时，仅对新增节点生效，存量节点即使重置也无法同步配置。 系统盘加密：系统盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**仅“弹性云服务器-虚拟机”类型支持系统盘加密，且仅部分Region支持此选项，具体请以界面为准。** • 默认不加密。 • 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 **说明** 修改“系统盘加密”配置时，对新增的节点自动生效，存量节点需要手动重置节点后生效。

参数	参数说明
数据盘	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0以下版本的集群中，**至少需要一块默认数据盘**，供容器运行时和Kubelet组件使用，**该数据盘不能被删除卸载，否则会导致节点不可用。** ● 默认数据盘：供容器运行时和Kubelet组件使用。您可以自行设置数据盘的规格为20GiB-32768GiB之间的数值，缺省值为100GiB。 ● 其他普通数据盘，您可以设置数据盘的规格为10GiB-32768GiB之间的数值，缺省值为100GiB。 **说明** 修改数据盘“规格”配置时，仅对新增节点生效，存量节点即使重置也无法同步配置。 **高级配置** 单击后方的“展开高级设置”可进行如下设置： ● 数据盘空间分配：对数据盘上存在的容器引擎、镜像、临时存储等进行空间划分，避免因磁盘空间不足导致业务异常。数据盘空间分配详细说明请参见**默认数据盘空间分配说明**。 **说明** 修改“数据盘空间分配”配置时，仅对新增节点生效，存量节点即使重置也无法同步配置。 ● 加密：数据盘加密功能可为您的数据提供强大的安全防护，加密磁盘生成的快照及通过这些快照创建的磁盘将自动继承加密功能。**“裸金属服务器”类型节点不支持数据盘加密，且仅部分Region支持此选项，具体请以界面为准。** – 默认不加密。 – 点选“加密”后，可在弹出的“加密设置”对话框中，选择已有的密钥，若没有可选的密钥，请单击后方的链接创建新密钥，完成创建后单击刷新按钮。 **说明** 修改“数据盘加密”配置时，仅对新增节点生效，存量节点即使重置也无法同步配置。 **增加数据盘** 弹性云服务器最多可以添加16块，裸金属服务器最多可以添加10块。默认情况直接创建为裸盘，不做任何处理。您也可以展开高级配置，选择如下配置。 ● 默认：默认情况直接创建为裸盘，不做任何处理。 ● 挂载到指定目录：将数据盘挂载到指定目录。 ● 作为持久存储卷：适用于对PV有性能要求的场景。持久存储卷的节点会添加上node.kubernetes.io/local-storage-persistent标签，取值为linear或striped。 ● 作为临时存储卷：适用于对EmptyDir有性能要求的场景。

参数	参数说明
	说明 • 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。 • 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=1.2.29。 本地持久卷和本地临时卷支持如下两种写入模式。 • 线性（linear）：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。 • 条带化（striped）：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。条带化模式的存储池不支持扩容。多块存储卷才能选择条带化。 **本地盘说明** 节点规格为“磁盘增强型”或“超高I/O型”时，有一块数据盘可以是本地盘。 本地磁盘实例有宕机风险，不保证数据可靠性，建议您使用云硬盘存储您的业务数据。

网络配置

表 4-11 网络配置参数

参数	参数说明
虚拟私有云	默认为集群所在VPC，不可修改。
节点子网	节点子网默认使用创建集群时的子网配置，也可以选择其他子网。 • 多个子网：可选择同一VPC下的多个子网作为节点可用网段，扩容节点会优先消耗排序靠前的子网IP资源。 • 单个子网：当节点池关联的单个子网IP资源较为紧张时，推荐配置多个子网，否则可能会出现节点池扩容失败的问题。

高级配置

表 4-12 高级配置

参数	参数说明
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动帮您创建CCE-Dynamic-Provisioning-Node=节点id的标签。 **说明** 修改“资源标签”后，对新增的节点自动生效，存量节点会根据“存量节点标签及污点”开关状态决定是否同步更新。
K8S标签（Labels）	设置附加到Kubernetes对象（比如Pod）上的键值对，填写键值对后，单击“确认添加”。最多可以添加20条标签。 使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。详细请参见**Labels and Selectors**。 **说明** 修改“K8s标签”后，对新增的节点自动生效，节点池下的存量节点会根据“存量节点标签及污点”开关状态决定是否同步更新。
K8S污点（Taints）	默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点，每条污点包含以下3个参数： • Key：必须以字母或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符；另外可以使用DNS子域作为前缀。 • Value：必须以字符或数字开头，可以包含字母、数字、连字符、下划线和点，最长63个字符。 • Effect：只可选NoSchedule，PreferNoSchedule或NoExecute。 污点的使用请参见**管理节点污点**。 **说明** 修改“污点（Taints）”后，对新增的节点自动生效，节点池下的存量节点会根据“存量节点标签及污点”开关状态决定是否同步更新。

参数	参数说明
存量节点标签及污点	勾选后，更新节点池的资源标签、K8s标签及污点时，会将节点池配置中的资源标签、K8s标签或污点修改同步至节点池中已有的节点。 **说明** 节点池更新时，如果修改“同步资源标签”开关状态，需要注意如下事项： ● 开关从“不同步”更改为“同步”时： – CCE会基于当前节点池的资源标签配置，对存量节点进行更新。如果用户已在ECS侧设置了与当前节点池的资源标签配置中同key值的资源标签，则该标签的value值将被刷新成节点池中的配置保持一致。 – 存量节点的资源标签同步需要一定的处理时间（通常为10分钟以内，与节点池的节点规模相关）。 – 请等待上一次资源标签同步完成之后，再下发同步资源标签的请求，否则可能会导致存量节点的资源标签不一致。 节点池更新时，如果修改“同步K8s标签”及“同步污点”开关状态，需要注意以下事项： ● 开关状态从“同步”更改为“不同步”时，可能会出现节点池中的新旧节点标签或污点不一致的情况。当业务调度依赖节点标签或污点时，可能会出现调度失败或节点池弹性扩容能力失效。 ● 开关状态从“不同步”更改为“同步”时： – 如果在未开启同步时用户**修改**或**新增**了节点池配置中的标签或污点，重新开启同步后，配置会在一定时间内（一般为10分钟内）自动同步到存量节点上。 – 如果在未开启同步时用户**删除**了节点池配置中的标签或污点，重新开启同步后，需要前往节点列表界面手动删除节点已有的标签或污点。 – 如果在未开启同步时对**已有节点的污点手动修改key或effect**，在开启同步后，会在已有节点上增加一个全新的污点（key不同但value和effect相同或effect不同但key和value相同）。这是由于K8s污点原生逻辑使用key和effect作为一组匹配键值，即key或effect不同的污点会被认为是两个污点。

参数	参数说明
新增节点调度策略	节点池中新增节点的默认调度策略。勾选“设置为不可调度”之后，该节点池新增的节点在创建完成之后均会被打上不可调度的标签。以方便用户在工作负载被调度到节点上之前，对节点进行一些操作。 定时开启调度时间：若设置定时开启调度功能，在超过自定义时间后，节点将会自动开启调度。 ● 不设置：默认情况下，将不设置超时时间，此时节点需要您前往“节点管理”界面，手动选择节点开启调度，详情请参见**一键设置节点调度策略**。 ● 自定义：该参数可配置节点不可调度的默认超时时间，取值范围为0-99min。 **说明** ● 如果需要同时使用节点池的自动扩缩容能力，定时开启调度时间应小于15min。因为通过autoscaler扩容的节点如果处于不可调度状态超过15min，autoscaler会认为本次扩容失败，触发再次扩容。同时，原节点处于不可调度状态超过20min，节点将被列入缩容备选节点，被autoscaler缩容。 ● 开启该开关后，节点池的创建/更新过程中，节点会被打上node.cloudprovider.kubernetes.io/uninitialized的污点。
委托	委托是由租户管理员在统一身份认证服务上创建的。通过委托，可以将云主机资源共享给其他账号，或委托更专业的人或团队来代为管理。 如果没有委托请单击右侧“新建委托”创建。 **说明** 修改“委托”后，对新增的节点自动生效，存量节点重置后也不会生效。
自定义节点名称前后缀	节点池下的节点名称自定义前后缀，支持配置前缀和后缀。配置完成之后，该节点池下的节点名称将带上配置的前后缀信息。例如前缀为prefix-，后缀为-suffix，那么最终该节点池下的节点名称为prefix-nodepoolName-五位随机数-suffix。 ● 自定义前后缀名称前后缀仅支持创建节点池时指定，不支持修改。 ● 前缀支持以特殊字符结尾，后缀支持以特殊字符开头。 ● 节点名称由三部分组成：前缀+节点池名称-五位随机字符+后缀，总长度不超过56个字符。 ● 节点名称中不支持“.”与特殊字符连用，例如“..”、“.-”、“-.”。 ● 仅v1.28.1、v1.27.3、v1.25.6、v1.23.11、v1.21.12及以上集群版本支持该特性。 **说明** 修改“自定义节点名称前后缀”后，对新增的节点自动生效，存量节点重置后也不会生效。

步骤4 配置完成后，单击“确定”。

节点池参数更新后，前往“节点管理”页面，可查看节点池所属节点存在更新，可通过重置节点同步节点配置，与节点池配置保持一致。

----结束

4.5.2 更新弹性伸缩配置

开启弹性伸缩功能可根据弹性伸缩策略自动伸缩，否则只能手动修改节点池下的节点数量。

约束与限制

为保证节点池弹性伸缩功能的正常使用，需要在集群中安装**CCE集群弹性引擎**。

更新弹性伸缩配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“节点管理”，在目标节点池所在行右上角单击“弹性伸缩”。

- 若未安装autoscaler插件，请根据业务需求配置插件参数后单击“安装”，并等待插件安装完成。插件配置详情请参见**CCE集群弹性引擎**。
- 若已安装autoscaler插件，则可直接配置弹性伸缩策略。

步骤3 配置节点池弹性伸缩策略。

伸缩配置

- 自定义扩容规则：单击“添加规则”，在弹出的添加规则窗口中设置参数。您可以设置多条节点弹性策略，最多可以添加1条CPU使用率指标规则、1条内存使用率指标规则，且规则总数小于等于10条。

  规则类型可选择“指标触发”或“周期触发”，两种类型区别如下：

表 4-13 自定义规则类型

规则类型	参数设置
指标触发	– 触发条件：请选择“CPU分配率”或“内存分配率”，输入百分比的值。该百分比应大于**配置集群弹性伸缩策略**时节点缩容的“节点资源条件”。 **说明** ■ 分配率 = 节点池容器组（Pod）资源申请量 / 节点池Pod可用资源量(Node Allocatable)。 ■ **如果多条规则同时满足条件，会有如下两种执行的情况：** 如果同时配置了“CPU分配率”和“内存分配率”的规则，两种或多种规则同时满足扩容条件时，执行扩容节点数更多的规则。 如果同时配置了“CPU分配率”和“周期触发”的规则，当达到“周期触发”的时间值时CPU也满足扩容条件时，较早执行的周期触发规则会将节点池状态置为伸缩中状态，导致指标触发规则无法正常执行。待周期触发规则执行完毕，节点池状态恢复正常后，指标触发规则也不会执行。反之，如果指标触发规则执行较早，则等指标规则执行完毕后周期规则仍会执行。 ■ 配置了“CPU分配率”和“内存分配率”的规则后，策略的检测周期会因autoscaler每次循环的处理逻辑而变动。只要一次检测出满足条件就会触发扩容（还需要满足冷却时间、节点池状态等约束条件）。 ■ 当节点数已到达集群规模上限、**所属节点池的节点数上限**或**该规格的节点数上限**时，将不会触发指标扩容。 ■ 当节点数量、CPU、内存达到autoscaler插件设置的**节点扩容资源上限**时，将不会触发指标扩容。 – 执行动作：达到触发条件后所要执行的动作。 ■ 自定义：为节点池增加指定数量的节点。 ■ 自动计算：当达到触发条件时，自动扩容节点，将分配率恢复到触发条件以下。计算公式如下： 扩容节点数 = 节点池容器组（Pod）资源申请值 /（单节点可用资源值 * 目标分配率） - 当前节点数 + 1
周期触发	– 触发时间：可选择每天、每周、每月或每年的具体时间点。 – 执行动作：达到触发时间值后所要执行的动作，为节点池增加指定数量的节点。

- 节点数范围：弹性伸缩时节点池下的节点数量会始终介于节点数范围内。
- 冷却时间：指当前节点池扩容出的节点多长时间不能被缩容。

伸缩对象

- 规格选择：对节点池中的节点规格单独设置是否开启弹性伸缩。

  说明

  当节点池中包含多个规格时，您可以对每个规格的节点数范围和优先级进行单独配置。

步骤4 配置完成后，单击“确定”。

----结束

4.5.3 修改节点池配置

约束与限制

默认节点池DefaultPool不支持如下管理操作。

配置管理

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

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

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“配置管理”。

步骤4 在侧边栏滑出的“配置管理”窗口中，根据业务需求修改节点池参数值。

节点池支持的配置参数如下：

- **kubelet组件配置**
- **kube-proxy组件配置**
- **容器引擎Docker配置（仅使用Docker的节点池可见）**
- **容器引擎Containerd配置（仅使用Containerd的节点池可见）**
- **网络组件配置（仅CCE Turbo集群可见）**
- **节点池Pod安全组配置（仅CCE Turbo集群可见）**

步骤5 单击“确定”，完成配置操作。

----结束

kubelet 组件配置

名称	参数	参数说明	取值	修改说明
CPU管理策略配置	cpu-manager-policy	CPU管理策略配置，详情请参见**CPU调度**。 • none：关闭工作负载实例独占CPU的功能，优点是CPU共享池的可分配核数较多。 • static：开启工作负载实例独占CPU，适用于对CPU缓存和调度延迟敏感的场景。 • enhanced-static：在支持CPU独占的基础上，增强支持给Burstable Pods配置CPU优先使用核，适用于波峰、波谷相差大且大部分时间处于波谷状态的工作负载。	默认：none	-
请求至kube-apiserver的QPS配置	kube-api-qps	与APIServer通信的每秒查询个数。	默认：100	-
请求至kube-apiserver的Burst配置	kube-api-burst	每秒发送到APIServer的突发请求数量上限。	默认：100	-
kubelet管理的Pod上限	max-pods	Node能运行的Pod最大数量。	• CCE Standard集群：由**节点最大实例数**设置决定。 • CCE Turbo集群：由**节点网卡数量**决定。	-

名称	参数	参数说明	取值	修改说明
限制Pod中的进程数	pod-pids-limit	每个Pod中可使用的PID个数上限。	默认：-1，表示不限制	-
是否使用本地IP作为该节点的ClusterDNS	with-local-dns	开启后，会自动在节点的kubelet配置中添加节点默认网卡IP作为首选DNS地址。	默认：关闭（参数值为false）	-
事件创建QPS限制	event-qps	每秒可生成的事件数量。	默认：5	-
事件创建的个数的突发峰值上限	event-burst	突发性事件创建的上限值，允许突发性事件创建临时上升到所指定数量。	默认：10	-
允许使用的不安全系统配置	allowed-unsafe-sysctls	允许使用的不安全系统配置。 CCE从**1.17.17**集群版本开始，kube-apiserver开启了pod安全策略，需要在pod安全策略的allowedUnsafeSysctls中增加相应的配置才能生效（1.17.17以下版本的集群可不配置）。详情请参见**Pod安全策略开放非安全系统配置示例**。	默认：[]	-

名称	参数	参数说明	取值	修改说明
节点超卖特性	over-subscription-resource	节点超卖特性。 设置为true表示开启节点超卖特性，节点超卖的特性请参见**动态资源超卖**。	• 集群版本为v1.23.9-r0、v1.25.4-r0以下：默认为开启（参数值为true） • 集群版本为v1.23.9-r0、v1.25.4-r0、v1.27-r0、v1.28.1-r0及以上：默认为关闭（参数值为false）	-
节点混部特性	colocation	节点混部特性。 设置为true表示开启节点混部特性，节点混部的特性请参见**动态资源超卖**。	• 集群版本为v1.23.9-r0、v1.25.4-r0以下：默认为开启（参数值为true） • 集群版本为v1.23.9-r0、v1.25.4-r0、v1.27-r0、v1.28.1-r0及以上：默认为关闭（参数值为false）	-

名称	参数	参数说明	取值	修改说明
拓扑管理策略	topology-manager-policy	设置拓扑管理策略。 合法值包括： ● restricted：kubelet 仅接受在所请求资源上实现最佳 NUMA对齐的Pod。 ● best-effort：kubelet会优先选择在 CPU 和设备资源上实现NUMA对齐的Pod。 ● none（默认）：不启用拓扑管理策略。 ● single-numa-node：kubelet仅允许在 CPU和设备资源上对齐到同一NUMA节点的Pod。	默认：none	**须知** 请谨慎修改，修改topology-manager-policy和topology-manager-scope会重启kubelet，并且以更改后的策略重新计算容器实例的资源分配，这有可能导致已经运行的容器实例重启甚至无法进行资源分配。
拓扑管理策略的资源对齐粒度	topology-manager-scope	设置拓扑管理策略的资源对齐粒度。合法值包括： ● container（默认)：对齐粒度为容器级 ● pod：对齐粒度为Pod级	默认：container	
容器指定DNS解析配置文件	resolv-conf	容器指定DNS解析配置文件	默认为空值	-
除长期运行的请求之外所有运行时请求的超时时长	runtime-request-timeout	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。	默认为2m0s	v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及以上版本的集群支持该参数。

名称	参数	参数说明	取值	修改说明
是否让kubelet每次仅拉取一个镜像	serialize-image-pulls	串行拉取镜像。 ● 关闭：建议值，以支持并行拉取镜像，提高Pod启动速度。 ● 开启：支持串行拉取镜像。	● 集群版本为v1.21.12-r0、v1.23.11-r0、v1.25.6-r0、v1.27.3-r0、v1.28.1-r0以下：默认为开启（参数值为true） ● 集群版本为v1.21.12-r0、v1.23.11-r0、v1.25.6-r0、v1.27.3-r0、v1.28.1-r0及以上：默认为关闭（参数值为false）	v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及以上版本的集群支持该参数。
每秒钟可以执行的镜像仓库拉取操作限值	registry-pull-qps	镜像仓库的QPS上限。	默认为5 取值范围为1~50	v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及以上版本的集群支持该参数。
突发性镜像拉取的上限值	registry-burst	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。	默认为10 取值范围为1~100，且取值必须大于等于registry-pull-qps的值。	v1.21.10-r0、v1.23.8-r0、v1.25.3-r0及以上版本的集群支持该参数。
容器的日志文件个数上限	container-log-max-files	每个容器日志文件的最大数量。当存在的日志文件数量超过这个值时，最旧的日志文件将被删除，以便为新的日志留出空间。	默认为10 取值范围为2~100	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。

名称	参数	参数说明	取值	修改说明
容器日志文件在轮换生成新文件时之前的最大值	container-log-max-size	每个容器的日志文件的最大大小。当日志文件达到这个大小时，将会触发日志轮换即关闭当前日志文件并创建新的日志文件以继续记录。	默认为50Mi 取值范围为1Mi~4096Mi	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
镜像垃圾回收上限百分比	image-gc-high-threshold	当kubelet磁盘达到多少时，kubelet开始回收镜像。	默认为80 取值范围为1~100	如果需要禁用镜像垃圾回收，请将该参数设置为100。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
镜像垃圾回收下限百分比	image-gc-low-threshold	回收镜像时当磁盘使用率减少至多少时停止回收。	默认为70 取值范围为1~100	该参数取值不得大于镜像垃圾回收上限百分比。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
节点内存预留	system-reserved-mem	系统内存预留，目的是为OS系统守护进程（如 sshd、udev 等）预留内存资源。	默认值：自动计算预留内存数，预留值随节点规格变动，具体请参见**节点预留资源策略说明**	kube-reserved-mem，system-reserved-mem之和小于节点池中节点最小内存规格的50%。
	kube-reserved-mem	Kubernetes组件内存预留，目的是为Kubernetes系统守护进程（如kubelet、container runtime 等）预留内存资源。		

kube-proxy 组件配置

名称	参数	参数说明	取值	修改说明
系统中最大的连接跟踪表项数目	conntrack-min	系统中最大的连接跟踪表项数目。 可通过以下命令查询： `sysctl -w net.nf_conntrack_max`	默认：131072	-
TCP连接在关闭状态下等待的时间	conntrack-tcp-timeout-close-wait	控制TCP连接在关闭状态下等待的时间。 可通过以下命令查询： `sysctl -w net.netfilter.nf_conntrack_tcp_timeout_close_wait`	默认：1h0m0s	-

容器引擎 Docker 配置（仅使用 Docker 的节点池可见）

名称	参数	参数说明	取值	修改说明
容器umask值	native-umask	默认值为normal，表示启动的容器umask值为0022。	默认：normal	不支持修改
单容器可用数据空间	docker-base-size	设置每个容器可使用的最大数据空间。	默认：0	不支持修改
不安全的镜像源地址	insecure-registry	是否允许使用不安全的镜像源地址。	false	不支持修改
容器core文件的大小限制	limitcore	容器core文件的大小限制，单位是Byte。 如果不设置大小限制，可设置为infinity。	默认：5368709120	-
容器内句柄数限制	default-ulimit-nofile	设置容器中可使用的句柄数上限。	默认：{soft}:{hard}	该值大小不可超过节点内核参数nr_open的值，且不能是负数。 节点内核参数nr_open可通过以下命令获取： `sysctl -a | grep nr_open`

名称	参数	参数说明	取值	修改说明
镜像拉取超时时间	image-pull-progress-timeout	如果超时之前镜像没有拉取成功，本次镜像拉取将会被取消。	默认：1m0s	该参数在v1.25.3-r0版本开始支持
单次拉取镜像层的最大并发数	max-concurrent-downloads	设置拉取镜像层的最大并发数。	默认：3 取值范围为1~20	该参数如果设置过大，可能导致节点其他业务的网络性能受影响或导致磁盘IO和CPU增高。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
容器日志文件轮换生成新文件的最大值	max-size	容器日志文件开始转储的最大大小。当日志文件达到这个大小时，将会触发日志轮换即关闭当前日志文件并创建新的日志文件以继续记录。	默认为50Mi 取值范围为1Mi~4096Mi	该参数如果设置过小，可能导致重要日志信息的丢失；如果设置过大，则可能占用过多的磁盘空间。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
容器的日志文件个数上限	max-file	容器可以保留的日志文件的最大数量。当存在的日志文件数量超过这个值时，最旧的日志文件将被删除，以便为新的日志留出空间。	默认：20 取值范围为2~100	该参数如果设置过小，可能导致重要日志信息的丢失；如果设置过大，则可能占用过多的磁盘空间。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。

名称	参数	参数说明	取值	修改说明
替代镜像仓库配置	registry-mirrors	配置一个或多个镜像仓库，以便在从容器运行时获取镜像时使用替代的镜像仓库。	默认：[] 替代的镜像仓库需要是http://或者https://开头的IP地址或域名。 例如，使用自建的镜像仓库地址"http://example.com"和"https://example.com"替代默认仓库，则参数值为["http://example.com,https://example.com"]。	• 如果需要提高镜像拉取速度可以将替代仓库配置为本地镜像仓库。 • 如果需要提高容错能力和可用性可以配置多个替代镜像仓库。 **须知** 配置错误的替代镜像仓库可能导致容器无法拉取所需镜像。 v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上版本的集群支持该参数。

容器引擎 Containerd 配置（仅使用 Containerd 的节点池可见）

名称	参数	参数说明	取值	修改说明
容器core文件的大小限制	limitcore	容器core文件的大小限制，单位是Byte。 如果不设置大小限制，可设置为infinity。	默认：5368709120	-
容器内句柄数限制	default-ulimit-nofile	设置容器中可使用的句柄数上限。	默认：1048576	该值大小不可超过节点内核参数nr_open的值，且不能是负数。 节点内核参数nr_open可通过以下命令获取： `sysctl -a | grep nr_open`
镜像拉取超时时间	image-pull-progress-timeout	如果超时之前镜像没有拉取成功，本次镜像拉取将会被取消。	默认：1m0s	该参数在v1.25.3-r0版本开始支持

名称	参数	参数说明	取值	修改说明
insecure_skip_verify	insecure_skip_verify	跳过仓库证书验证。	默认：false	不支持修改
单次拉取镜像层的最大并发数	max-concurrent-downloads	设置拉取镜像层的最大并发数。	默认：3 取值范围为1~20	该参数如果设置过大，可能导致节点其他业务的网络性能受影响或导致磁盘IO和CPU增高。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
容器的最大日志行大小	max-container-log-line-size	是容器的最大日志行大小（以字节为单位）。超过限制的日志行将被分成多行。	默认：16384 取值范围为1~2097152	配置增大会增加containerd内存消耗。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。
替代镜像仓库配置	registry-mirrors	配置一个或多个镜像仓库，以便在从容器运行时获取镜像时使用替代的镜像仓库。	不配置时镜像仓库默认为docker.io，且配置SWR镜像仓库为替代镜像仓库。 镜像仓库需要是IP地址或域名，替代的镜像仓库需要是http://或者https://开头的IP地址或域名。	• 建议添加本地镜像仓库以提高镜像拉取速度。 • 使用多个镜像仓库以提高容错能力和可用性。 v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本的集群支持该参数。 **须知** 配置错误的镜像仓库或者替代镜像仓库可能导致容器无法拉取所需镜像。

名称	参数	参数说明	取值	修改说明
跳过证书认证的镜像仓库	insecure-registries	控制指定的镜像仓库跳过对安全证书的验证，一般用于与不安全或自签名的镜像仓库建立连接。	默认为空 镜像仓库需要是IP地址或域名。	• 仅在开发或测试环境中使用，不建议在生产环境中启用。 • 如果使用自签名证书或无法获取有效证书的私有镜像仓库时，才考虑启用此选项。 v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本的集群支持该参数。

网络组件配置（仅 CCE Turbo 集群可见）

名称	参数	参数说明	取值	修改说明
节点池网卡预热参数配置开关	enable-node-nic-configuration	是否开启节点池级别的网卡预热。	默认：关闭	节点池网络组件配置开关关闭后，节点池容器网卡动态预热参数将与集群级别保持一致。
nic-threshold	nic-threshold	节点池级别的节点绑定容器网卡数低水位:节点绑定容器网卡数高水位	默认：0:0	**说明** 此参数配置废弃中，请采用其他4个容器网卡动态预热参数。
节点池级别的节点最少绑定容器网卡数	nic-minimum-target	保障节点最少有多少张容器网卡绑定在节点上。 参数值需为正整数。例如10，表示节点最少有10张容器网卡绑定在节点上。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	默认：10	建议配置为大部分节点平时日常运行的Pod数。

名称	参数	参数说明	取值	修改说明
节点池级别的节点预热容器网卡上限检查值	nic-maximum-target	当节点绑定的容器网卡数超过节点预热容器网卡上限检查值(nic-maximum-target)，不再主动预热容器网卡。 当该参数大于等于节点最少绑定容器网卡数(nic-minimum-target)时，则开启预热容器网卡上限值检查；反之，则关闭预热容器网卡上限值检查。 参数值需为正整数。例如0，表示关闭预热容器网卡上限值检查。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	默认：0	建议配置为大部分节点平时最多运行的Pod数。
节点池级别的节点动态预热容器网卡数	nic-warm-target	当Pod使用完节点最少绑定容器网卡数(nic-minimum-target)后，会始终额外预热多少张容器网卡，只支持数值配置。 当 节点动态预热容器网卡数(nic-warm-target) + 节点当前绑定的容器网卡数 大于节点预热容器网卡上限检查值(nic-maximum-target)时，只会预热nic-maximum-target与节点当前绑定的容器网卡数的差值。	默认：2	建议配置为大部分节点日常10s内会瞬时弹性扩容的Pod数。

名称	参数	参数说明	取值	修改说明
节点池级别的节点预热容器网卡回收阈值	nic-max-above-warm-target	只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值 时，才会触发预热容器网卡的解绑回收。只支持数值配置。 • 调大此值会减慢空闲容器网卡的回收，加快Pod的启动速度，但会降低IP地址的利用率，特别是在IP地址紧张的场景，**请谨慎调大**。 • 调小此值会加快空闲容器网卡的回收，提高IP地址的利用率，但在瞬时大量Pod激增的场景，部分Pod启动会稍微变慢。	默认：2	建议配置为大部分节点日常在分钟级时间范围内会频繁弹性扩容缩容的Pod数 - 大部分节点日常10s内会瞬时弹性扩容的Pod数。

节点池 Pod 安全组配置（仅 CCE Turbo 集群可见）

名称	参数	参数说明	取值	修改说明
节点池上Pod默认使用的安全组	security_groups_for_nodepool	可填写安全组 ID，不配置则使用集群容器网络的默认安全组，并且最多可同时指定5个安全组ID，中间以英文分号（;）分隔。 优先级低于**SecurityGroup**资源对象配置的安全组。	-	-

4.5.4 纳管节点至节点池

如果您需要在购买ECS云服务器后将其添加到集群中的某个节点池中，或者将节点池的某个节点从集群里移除后将其重新添加到节点池，您可以通过纳管节点实现以上诉求。

须知

- 纳管时，会将所选弹性云服务器的操作系统重置为CCE提供的标准镜像，以确保节点的稳定性。
- 所选弹性云服务器挂载的系统盘、数据盘都会在纳管时清理LVM信息，包括卷组（VG）、逻辑卷（LV）、物理卷（PV），请确保信息已备份。
- 纳管过程中，请勿在弹性云服务器控制台对所选虚拟机做任何操作。
- 节点纳管至节点池后，如果节点池触发弹性伸缩策略缩容节点，则该节点将会被删除。

操作步骤

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更多 > 纳管节点”。

步骤4 选择一个或多个满足条件的节点。支持纳管符合如下条件的云服务器至节点池：

- 待纳管节点需与节点池属于同一虚拟私有云和子网。
- 待纳管节点需与节点池属于相同的企业项目。
- 待纳管节点需与当前节点池相同的计费模式。例如，按需计费节点池只支持纳管按需计费的节点。
- 待纳管节点需与当前节点池相同的云服务器组。
- 待纳管节点必须状态为“运行中”，且不携带 CCE 专属节点标签CCE-Dynamic-Provisioning-Node。
- 待纳管节点的系统组件使用独立磁盘存储时需挂载数据盘，可使用本地盘（磁盘增强型实例）或至少挂载一块20GiB及以上的数据盘，且不存在10GiB以下的数据盘。关于节点挂载数据盘的操作说明，请参考**新增磁盘**。
- 待纳管节点规格要求至少 2 核 4 GiB，且只绑定了 1 张网卡。
- 批量纳管仅支持添加与节点池相同规格、可用区、资源预留、系统盘、数据盘配置的云服务器。
- 云服务器上已分区的磁盘不会被纳管为数据盘，请提前做好数据备份与磁盘清理。

步骤5 单击“确定”。

----结束

4.5.5 复制节点池

通过CCE控制台可以方便地复制现有节点池的配置，从而创建新的节点池。

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更多 > 复制”。

图 4-2 复制节点池

步骤4 在弹出的“复制节点池”窗口中，可以看到复制的节点池配置，您可以根据需要进行修改，配置项详情请参见**创建节点池**。确定配置后单击“下一步：规格确认”。

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

----结束

4.5.6 同步节点池

在节点池配置更新后，节点池中的已有节点无法自动同步部分配置，您可以手动同步节点配置。

须知

- 批量同步过程中请勿删除或重置节点，否则可能导致节点池配置同步失败。
- 该操作涉及重置节点，**节点上已运行的工作负载业务可能会由于单实例部署、可调度资源不足等原因产生中断**，请您合理评估升级风险，并挑选业务低峰期进行，或对关键业务应用设置PDB策略（Pod Disruption Budget，即**干扰预算**），升级过程中将严格根据PDB规则保障关键业务的可用性。
- 同步已有节点时，节点会被重置，系统盘和数据盘将会被清空，请在同步前备份重要数据。
- 仅部分节点池参数可通过重置节点同步，详细约束如下：
  - 仅v1.19及以上版本的集群支持修改容器引擎、操作系统、系统盘/数据盘大小、数据盘空间分配、安装前/后执行脚本配置。
  - 修改节点池容器引擎、操作系统、安装前/后执行脚本时，修改后的配置仅对新增节点生效，存量节点如需同步配置，需要手动重置存量节点。
  - 修改节点池系统盘/数据盘大小、数据盘空间分配则仅对新增节点生效，即使重置存量节点也无法同步配置。
  - 修改资源标签、K8s标签和污点数据会根据“存量节点标签及污点”开关状态决定是否自动同步已有节点，无需重置节点。

单个节点同步

步骤1 登录CCE控制台。

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

步骤3 节点池中的存量节点将提示“存在更新”。

步骤4 单击“存在更新”，在提示窗口中确认是否立即重置节点。

----结束

批量同步

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更多 > 同步”。

步骤4 在弹出的“批量同步”窗口中，设置同步参数。

- 操作系统：该项无需设置，用于展示目标版本的镜像信息。
- 同步方式：当前支持节点重置方式进行同步。
- 每批最大同步节点数：节点升级时，允许节点不可用的最大数量。节点重置方式进行同步时节点将不可用，请合理设置该参数，尽量避免出现集群节点不可用数量过多导致Pod无法调度的情况。
- 节点列表：选择需要同步节点池配置的节点。

步骤5 单击“确定”，即可开始节点池的同步。

----结束

4.5.7 升级操作系统

当CCE发布新版本的操作系统镜像时，已有节点无法自动升级，您可以手动进行批量升级。

注意事项

- 该操作会通过重置节点的方式升级操作系统，**节点上已运行的工作负载业务可能会由于单实例部署、可调度资源不足等原因产生中断**，请您合理评估升级风险，并挑选业务低峰期进行，或对关键业务应用设置PDB策略（Pod Disruption Budget，即**干扰预算**），升级过程中将严格根据PDB规则保障关键业务的可用性。
- 节点的系统盘和数据盘将会被清空，重置前请事先**备份重要数据**。

- 节点重置会清除用户单独添加的K8S标签和K8S污点，可能导致与节点有绑定关系的资源（本地存储，指定调度节点的负载等）无法正常使用。请谨慎操作，避免对运行中的业务造成影响。
- 升级操作完成后，节点将会自动开机。
- 为确保节点稳定性，系统会预留部分CPU和内存资源，用于运行必须的系统组件。

约束与限制

- 使用私有镜像的节点暂不支持升级操作。
- 老版本的节点升级操作系统时可能存在兼容性问题，请手动重置节点完成操作系统升级。

默认节点池

步骤1 登录CCE控制台。

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

步骤3 单击默认节点池名称后的“升级”。

步骤4 在弹出的“升级操作系统”窗口中，设置升级参数。

- 目标操作系统：该项无需设置，用于展示目标版本的镜像信息。
- 升级方式：当前支持节点重置方式进行升级。
- 每批最大升级节点数：节点升级时，允许节点不可用的最大数量。节点重置方式进行同步时节点将不可用，请合理设置该参数，尽量避免出现集群节点不可用数量过多导致Pod无法调度的情况。
- 节点列表：选择需要升级的节点。
- 登录方式：
  - **密码**

    用户名默认为“root”，请输入登录节点的密码，并确认密码。

    登录节点时需要使用该密码，请妥善管理密码，系统无法获取您设置的密码内容。
  - **密钥对**

    选择用于登录本节点的密钥对，支持选择共享密钥。

    密钥对用于远程登录节点时的身份认证。若没有密钥对，可单击选项框右侧的“创建密钥对”来新建，创建密钥对操作步骤请参见**创建密钥对**。
  - **使用镜像密码**（当节点类型为弹性云服务器虚拟机或物理机，且操作系统选择私有镜像时支持）

    保留所选择镜像的密码。为了保证您的正常使用，请确保所选择镜像中已经设置了密码。
- 安装前执行脚本：

  请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。

  脚本将在Kubernetes软件安装前执行，可能导致Kubernetes软件无法正常安装，需谨慎使用。
- 安装后执行脚本：

  请输入脚本命令，命令中不能包含中文字符。脚本命令会进行Base64转码。

脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。

步骤5 单击“确定”，即可开始操作系统滚动升级。

----结束

非默认节点池

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更多 > 同步”。

步骤4 在弹出的“批量同步”窗口中，设置同步参数。

- 操作系统：该项无需设置，用于展示目标版本的镜像信息。
- 同步方式：当前支持节点重置方式进行同步。
- 每批最大同步节点数：节点升级时，允许节点不可用的最大数量。节点重置方式进行同步时节点将不可用，请合理设置该参数，尽量避免出现集群节点不可用数量过多导致Pod无法调度的情况。
- 节点列表：选择需要同步节点池配置的节点。

步骤5 单击“确定”，即可开始节点池的同步。

----结束

4.5.8 迁移节点

您可以将同一个集群下节点在节点池间进行迁移，具体迁移场景如表4-14。

表 4-14 迁移场景

迁移场景		是否支持迁移	操作步骤
原节点池	待迁移的目标节点池		
自定义节点池	默认节点池（DefaultPool）	支持迁移	将自定义节点池中的节点迁移到默认节点池
默认节点池（DefaultPool）	自定义节点池	v1.23及以上版本的集群支持	将默认节点池中的节点迁移到自定义节点池
自定义节点池	自定义节点池	不支持迁移	-

将自定义节点池中的节点迁移到默认节点池

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，并切换至“节点池”页签。

步骤3 单击待迁移的节点池名称后的“节点列表”。

步骤4 在需要迁移的节点的“操作”栏中，单击“更多 > 迁移”，迁移单个节点。

图 4-3 迁移节点到默认节点池

步骤5 在弹出的“迁移节点”窗口中进行确认。

说明

- 迁移完成后，节点上用户自定义的资源标签、K8s标签、污点不受影响。
- 迁移完成后，节点上名为cce.cloud.com/cce-nodepool的系统标签会被删除。如果已有工作负载使用该标签进行亲和/反亲和调度，在Kubelet重启时会将该节点上已存在的Pod停止并重新调度。

----结束

将默认节点池中的节点迁移到自定义节点池

须知

竞价计费模式的节点池不支持迁入节点。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，并切换至“节点池”页签。

步骤3 找到待迁移的目标节点池，单击“更多 > 迁入节点”。

图 4-4 迁移节点到自定义节点池

步骤4 在弹出的“迁入节点”窗口中，勾选满足以下条件的节点。

- 待迁入节点与当前节点池属于相同的虚拟私有云和子网。
- 待迁入节点与当前节点池属于相同的企业项目。
- 待迁入节点与当前节点池属于相同的云服务器组。
- 待迁入节点的计费模式需要与当前节点池支持的计费模式相同。
- 待迁入节点需要属于DefaultPool节点池，且状态为“运行中”。
- 待迁入节点需要与节点池的规格、可用区、资源预留、容器引擎、操作系统配置相同。

步骤5 单击“确定”。

说明

- 迁入成功后，将同步节点池资源标签、K8S标签、K8S污点配置，与节点池配置冲突时将使用节点池配置覆盖。
- 迁入成功后，将采用节点池安全组替换节点原来的安全组。
- 迁入成功后，将采用节点池委托替换节点原来的委托。
- 迁入成功后，节点原有的登录方式会被保留。
- 纳管至集群的节点迁入到节点池后，节点的纳管标记将会被清除。如果节点池发生缩容，该节点也可能会被同步缩容。

----结束

4.5.9 删除节点池

删除节点池，会先删除节点池中的节点，节点删除后，原有节点上的工作负载实例会自动迁移至其他节点池的可用节点。

约束与限制

- 对于包周期（包年/包月）预付费的节点池不能直接删除，请先移除节点池下全部的节点。
- 删除节点会导致与节点关联的**本地持久存储卷**类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。删除节点时使用了本地持久存储卷的Pod会从删除的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。

注意事项

- 删除节点池会同时删除节点池下的全部节点，请及时备份数据，避免重要数据丢失。
- 删除节点会涉及Pod迁移，可能会影响业务，请在业务低峰期操作。如果Pod具有特定的节点选择器，且集群中的其他节点均不符合标准，则工作负载实例可能仍处于无法安排的状态。
- 删除过程中，系统会把当前节点池中的节点均设置为不可调度状态。

操作步骤

步骤1 登录CCE控制台。

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

步骤3 单击节点池名称后的“更多 > 删除”。

步骤4 在弹出的“删除节点池”窗口中，请仔细阅读界面提示。

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

----结束

5 工作负载

5.1 工作负载概述

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

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

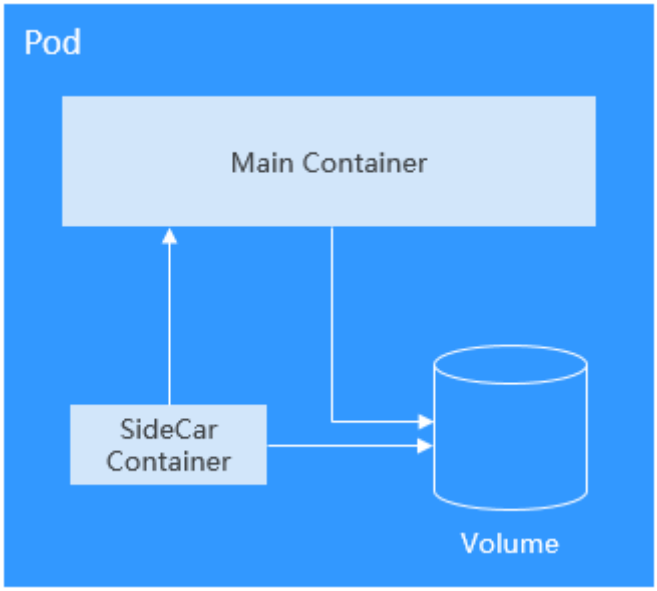
容器组（Pod）

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

Pod使用主要分为两种方式：

- Pod中运行一个容器。这是Kubernetes最常见的用法，您可以将Pod视为单个封装的容器，但是Kubernetes是直接管理Pod而不是容器。
- Pod中运行多个需要耦合在一起工作、需要共享资源的容器。通常这种场景下应用包含一个主容器和几个辅助容器（SideCar Container），如图5-1所示，例如主容器为一个web服务器，从一个固定目录下对外提供文件服务，而辅助容器周期性的从外部下载文件存到这个固定目录下。

图 5-1 Pod

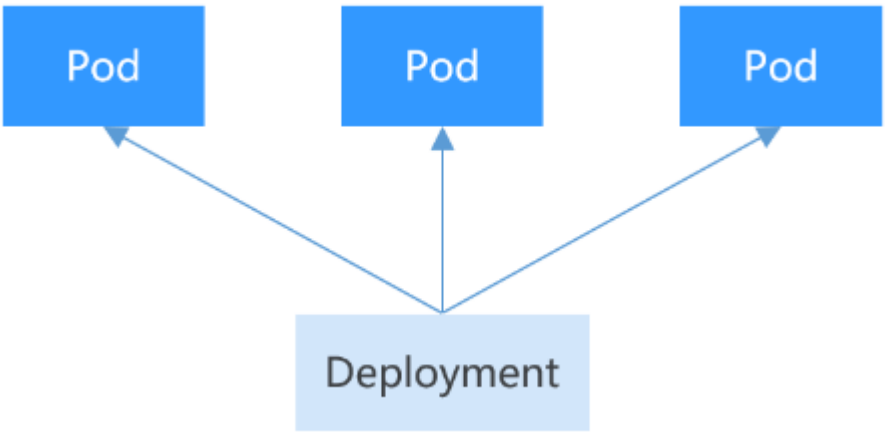


实际使用中很少直接创建Pod，而是使用Kubernetes中称为Controller的抽象层来管理Pod实例，例如Deployment和Job。Controller可以创建和管理多个Pod，提供副本管理、滚动升级和自愈能力。通常，Controller会使用Pod Template来创建相应的Pod。

无状态负载（Deployment）

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

图 5-2 Deployment



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

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

有状态负载（StatefulSet）

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

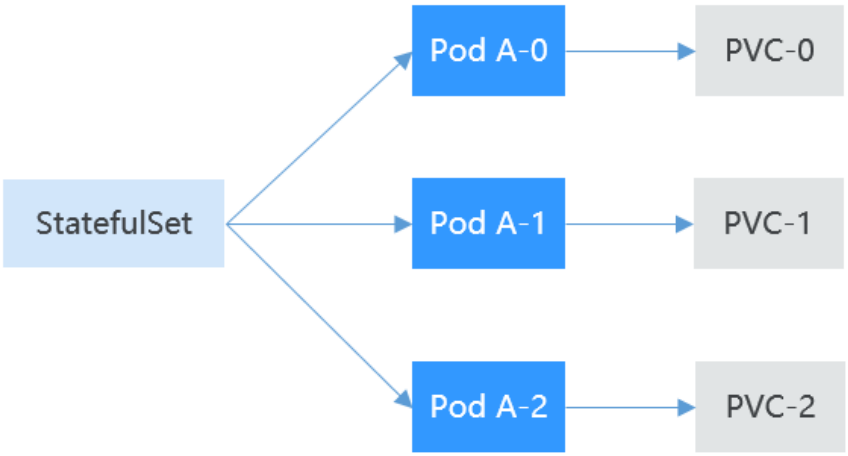
但是在某些场景下，这并不满足需求，比如有些分布式的场景，要求每个Pod都有自己单独的状态时，比如分布式数据库，每个Pod要求有单独的存储，这时Deployment无法满足业务需求。

分布式有状态应用的特点主要是应用中每个部分的角色不同（即分工不同），比如数据库有主备、Pod之间有依赖，在Kubernetes中部署有状态应用对Pod有如下要求：

- Pod能够被别的Pod找到，要求Pod有固定的标识。
- 每个Pod有单独存储，Pod被删除恢复后，必须读取原来的数据，否则状态就会不一致。

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

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

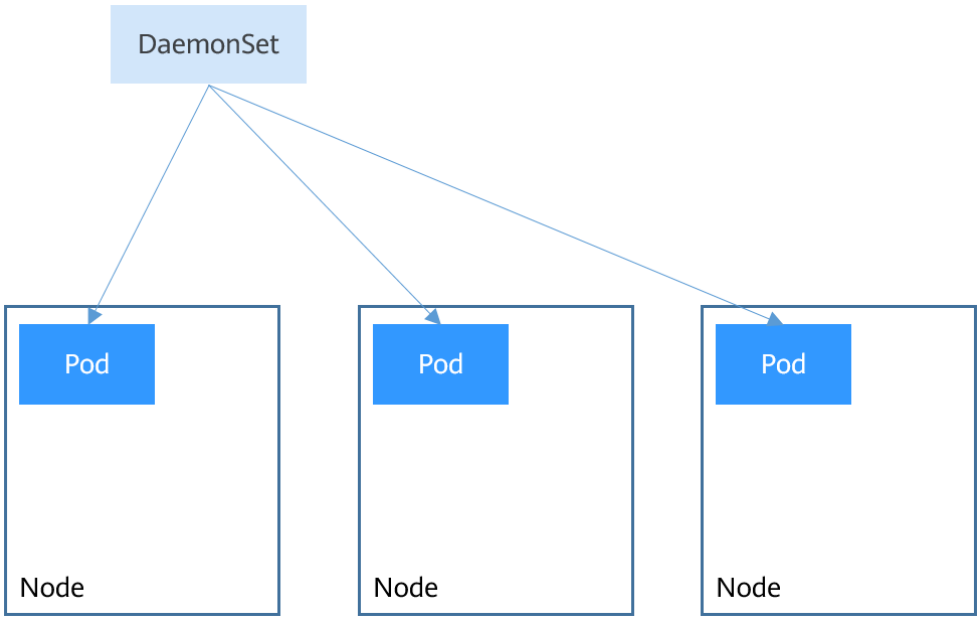


守护进程集（DaemonSet）

DaemonSet（守护进程集）在集群的每个节点上运行一个Pod，且保证只有一个Pod，非常适合一些系统层面的应用，例如日志收集、资源监控等，这类应用需要每个节点都运行，且不需要太多实例，一个比较好的例子就是Kubernetes的kube-proxy。

DaemonSet跟节点相关，如果节点异常，也不会在其他节点重新创建。

图 5-3 DaemonSet



普通任务（Job）和定时任务（CronJob）

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

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

任务负载的这种用完即停止的特性特别适合一次性任务，比如持续集成。

工作负载生命周期说明

表 5-1 状态说明

状态	说明
运行中	所有实例都处于运行中、或实例数为0时显示此状态。
未就绪	容器处于异常、负载下实例没有正常运行时显示此状态。
处理中	负载没有进入运行状态但也没有报错时显示此状态。
可用	当多实例无状态工作负载运行过程中部分实例异常，可用实例不为0，工作负载会处于可用状态。
执行完成	任务执行完成，仅普通任务存在该状态。
已停止	触发停止操作后，工作负载会处于停止状态，实例数变为0。v1.13之前的版本存在此状态。

状态	说明
删除中	触发删除操作后，工作负载会处于删除中状态。

5.2 创建工作负载

5.2.1 创建无状态负载（Deployment）

操作场景

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

前提条件

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

  说明

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

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载的信息。

基本信息

- 负载类型：选择无状态工作负载Deployment。工作负载类型的介绍请参见**工作负载概述**。
- 负载名称：填写工作负载的名称。请输入1到63个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。
- 命名空间：选择工作负载的命名空间，默认为default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见**创建命名空间**。
- 实例数量：填写实例的数量，即工作负载Pod的数量。
- 容器运行时：CCE Standard集群默认使用普通运行时，CCE Turbo集群可以使用普通运行时或安全运行时。具体区别请参见**安全运行时与普通运行时**。
- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见**设置时区同步**。

容器配置

- 容器信息

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

  - 基本信息：配置容器的基本信息。

参数	说明
容器名称	为容器命名。
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见**使用第三方镜像**。
镜像版本	选择需要部署的镜像版本。
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。 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**设置容器规格**。
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。 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**设置容器规格**。
GPU配额（可选）	当集群中包含GPU节点时，才能设置GPU配额，且集群中需安装**CCE AI套件（NVIDIA GPU）**插件。 ■ 不限制：表示不使用GPU。 ■ 独享：单个容器独享GPU。 ■ 共享：容器需要使用的GPU百分比，例如设置为10%，表示该容器需使用GPU资源的10%。 关于如何在集群中使用GPU，请参见**使用Kubernetes默认GPU调度**。
NPU配额（可选）	使用NPU芯片的数量，必须为整数，且必须安装**CCE AI套件（Ascend NPU）**插件后才能使用。 关于如何在集群中使用NPU，请参见**NPU调度**。
特权容器（可选）	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。

参数	说明
初始化容器（可选）	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**。

  - 生命周期（可选）：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**。
  - 健康检查（可选）：根据需求选择是否设置存活探针、就绪探针及启动探针，详情请参见**设置容器健康检查**。
  - 环境变量（可选）：支持通过键值对的形式为容器运行环境设置变量，可用于把外部信息传递给Pod中运行的容器，可以在应用部署后灵活修改，详情请参见**设置环境变量**。
  - 数据存储（可选）：在容器内挂载本地存储或云存储，不同类型的存储使用场景及挂载方式不同，详情请参见**存储**。

    说明

    负载实例数大于1时，不支持挂载云硬盘类型的存储。
  - 安全设置（可选）：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户ID，容器将以当前用户权限运行。
  - 容器日志（可选）：容器标准输出日志将默认上报至 AOM 服务，无需独立配置。您可以手动配置日志采集路径，详情请参见**通过ICAgent采集容器日志（不推荐）**。

    如需要关闭当前负载的标准输出，您可在**标签与注解**中添加键为kubernetes.AOM.log.stdout，值为[]的注解，即可关闭当前负载下全部容器的标准输出。该注解的使用方法请参见**表5-14**。

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

服务配置（可选）

服务（Service）可为Pod提供外部访问。每个Service有一个固定IP地址，Service将访问流量转发给Pod，而且Service可以为这些Pod自动实现负载均衡。

您也可以在创建完工作负载之后再创建Service，不同类型的Service概念和使用方法请参见**服务概述**。

高级配置（可选）

- 升级策略：指定工作负载的升级方式及升级参数，支持滚动升级和替换升级，详情请参见**设置工作负载升级策略**。
- 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。

  - 负载亲和：提供常用的负载亲和策略，快速实现负载亲和部署。
    - 不配置：不设置负载亲和策略。
    - 优先多可用区部署：通过设置Pod间反亲和（podAntiAffinity）实现，优先将工作负载的Pod调度到不同可用区的节点上。
    - 强制多可用区部署：通过设置Pod间反亲和（podAntiAffinity）实现，强制将工作负载的Pod调度到不同可用区的节点上，如集群下节点支持的可用区数量小于实例数，工作负载的Pod无法全部运行。
    - 自定义亲和策略：根据需求自定义设置亲和与反亲和规则，详情请参见**设置工作负载亲和/反亲和调度（podAffinity/podAntiAffinity）**。
  - 节点亲和：提供常用的负载亲和策略，快速实现负载亲和部署。
    - 不配置：不设置节点亲和策略。
    - 指定节点调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点，若不指定，将根据集群默认调度策略随机调度。
    - 指定节点池调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点池，若不指定，将根据集群默认调度策略随机调度。
    - 自定义亲和策略：根据需求自定义设置亲和与反亲和规则，详情请参见**设置节点亲和调度（nodeAffinity）**。
- 容忍策略：容忍策略与节点的污点能力配合使用，允许（不强制）负载调度到带有与之匹配的污点的节点上，也可用于控制负载所在的节点被标记污点后负载的驱逐策略，详情请参见**设置容忍策略**。
- 标签与注解：以键值对形式为工作负载Pod添加标签或注解，填写完成后需单击“确认添加”。关于标签与注解的作用及配置说明，请参见**设置标签与注解**。
- DNS配置：为工作负载单独配置DNS策略，详情请参见**工作负载DNS配置说明**。
- 性能管理配置：使用应用性能管理（APM）服务，为JAVA程序提供更精准的问题分析与定位，详情请参见**设置性能管理配置**。
- 网络配置：
  - Pod入/出口带宽限速：支持为Pod设置入/出口带宽限速，详情请参见**为Pod配置QoS**。
  - 是否开启指定容器网络配置：仅支持该功能的集群显示该选项，开启指定容器网络配置，工作负载使用指定的容器网络配置中的容器子网跟安全组来创建，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。
  - 指定容器网络配置名：只支持选择关联资源类型为工作负载类型的自定义容器网络配置。

步骤4 单击右下角“创建工作负载”。

----结束

通过 kubectl 命令行创建

本节以nginx工作负载为例，说明kubectl命令创建工作负载的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建一个名为nginx-deployment.yaml的描述文件。其中，nginx-deployment.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

描述文件内容如下。此处仅为示例，deployment的详细说明请参见**kubernetes官方文档**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx    #若使用“开源镜像中心”的镜像，可直接填写镜像名称；若使用“我的镜像”中的镜像，请在SWR中获取具体镜像地址。
        imagePullPolicy: Always
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

以上yaml字段解释如**表5-2**。

表 5-2 deployment 字段详解

字段名称	字段说明	必选/可选
apiVersion	表示API的版本号。 **说明** 请根据集群版本输入： • 1.17及以上版本的集群中无状态应用apiVersion格式为**apps/v1** • 1.15及以下版本的集群中无状态应用apiVersion格式为**extensions/v1beta1**	必选
kind	创建的对象类别。	必选
metadata	资源对象的元数据定义。	必选
name	deployment的名称。	必选
spec	用户对deployment的详细描述的主体部分都在spec中给出。	必选
replicas	实例数量。	必选
selector	定义Deployment可管理的容器实例。	必选

字段名称	字段说明	必选/可选
strategy	升级类型。当前支持两种升级方式，默认为滚动升级。 ● RollingUpdate：滚动升级。 ● ReplaceUpdate：替换升级。	可选
template	描述创建的容器实例详细信息。	必选
metadata	元数据。	必选
labels	metadata.labels定义容器标签。	可选
spec: containers	● image（必选）：容器镜像名称。 ● imagePullPolicy（可选）：获取镜像的策略，可选值包括Always（每次都尝试重新下载镜像）、Never（仅使用本地镜像）、IfNotPresent（如果本地有该镜像，则使用本地镜像，本地不存在时下载镜像），默认为Always。 ● name（必选）：容器名称。	必选
imagePullSecrets	Pull镜像时使用的secret名称。若使用私有镜像，该参数为必选。 ● 需要Pull SWR容器镜像仓库的镜像时，参数值固定为default-secret。 ● 当Pull第三方镜像仓库的镜像时，需设置为创建的secret名称。	可选

步骤3 创建deployment。

kubectl create -f nginx-deployment.yaml

回显如下表示已开始创建deployment。

```
deployment "nginx" created
```

步骤4 查看deployment状态。

kubectl get deployment

deployment状态显示为Running，表示deployment已创建成功。

```
NAME         READY    UP-TO-DATE   AVAILABLE   AGE
nginx        1/1      1            1           4m5s
```

参数解析：

- NAME：工作负载名称。
- READY：表示工作负载的可用状态，显示为“可用Pod个数/期望Pod个数”。
- UP-TO-DATE：指当前已经完成更新的副本数。
- AVAILABLE：可用的Pod个数。

- AGE：已经运行的时间。

步骤5 若工作负载（即deployment）需要被访问（集群内访问或节点访问），您需要设置访问方式，具体请参见**网络**创建对应服务。

----结束

相关文档

- **实现升级实例过程中的业务不中断**
- **CCE容器中域名解析的最佳实践**

5.2.2 创建有状态负载（StatefulSet）

操作场景

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

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

约束与限制

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

前提条件

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

  说明

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

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载的信息。

基本信息

- 负载类型：选择有状态工作负载StatefulSet。工作负载类型的介绍请参见**工作负载概述**。
- 负载名称：填写工作负载的名称。请输入1到63个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。
- 命名空间：选择工作负载的命名空间，默认为default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见**创建命名空间**。
- 实例数量：填写实例的数量，即工作负载Pod的数量。
- 容器运行时：CCE Standard集群默认使用普通运行时，CCE Turbo集群可以使用普通运行时或安全运行时。具体区别请参见**安全运行时与普通运行时**。
- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见**设置时区同步**。

容器配置

- 容器信息

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

  – 基本信息：配置容器的基本信息。

参数	说明
容器名称	为容器命名。
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见**使用第三方镜像**。
镜像版本	选择需要部署的镜像版本。
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。 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**设置容器规格**。
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。 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**设置容器规格**。

参数	说明
GPU配额（可选）	当集群中包含GPU节点时，才能设置GPU配额，且集群中需安装**CCE AI套件（NVIDIA GPU）**插件。 ■ 不限制：表示不使用GPU。 ■ 独享：单个容器独享GPU。 ■ 共享：容器需要使用的GPU百分比，例如设置为10%，表示该容器需使用GPU资源的10%。 关于如何在集群中使用GPU，请参见**使用Kubernetes默认GPU调度**。
NPU配额（可选）	使用NPU芯片的数量，必须为整数，且必须安装**CCE AI套件（Ascend NPU）**插件后才能使用。 关于如何在集群中使用NPU，请参见**NPU调度**。
特权容器（可选）	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。
初始化容器（可选）	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**。

- 生命周期（可选）：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**。
- 健康检查（可选）：根据需求选择是否设置存活探针、就绪探针及启动探针，详情请参见**设置容器健康检查**。
- 环境变量（可选）：支持通过键值对的形式为容器运行环境设置变量，可用于把外部信息传递给Pod中运行的容器，可以在应用部署后灵活修改，详情请参见**设置环境变量**。
- 数据存储（可选）：在容器内挂载本地存储或云存储，不同类型的存储使用场景及挂载方式不同，详情请参见**存储**。

  说明

  - 有状态负载支持“动态挂载”云硬盘，详情请参见**在有状态负载中动态挂载云硬盘存储**及**在有状态负载中动态挂载本地持久卷**。

    动态挂载通过**volumeClaimTemplates**字段实现，并依赖于StorageClass动态创建能力。有状态工作负载通过volumeClaimTemplates字段为每一个Pod关联了一个独有的PVC，而这个PVC又会和对应的PV绑定。因此当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。

  - 负载创建完成后，动态挂载的存储不支持更新。
- 安全设置（可选）：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户ID，容器将以当前用户权限运行。

  - 容器日志（可选）：容器标准输出日志将默认上报至 AOM 服务，无需独立配置。您可以手动配置日志采集路径，详情请参见**通过ICAgent采集容器日志（不推荐）**。

    如需要关闭当前负载的标准输出，您可在**标签与注解**中添加键为kubernetes.AOM.log.stdout，值为[]的注解，即可关闭当前负载下全部容器的标准输出。该注解的使用方法请参见**表5-14**。

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

实例间发现服务配置

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

服务配置（可选）

服务（Service）可为Pod提供外部访问。每个Service有一个固定IP地址，Service将访问流量转发给Pod，而且Service可以为这些Pod自动实现负载均衡。

您也可以在创建完工作负载之后再创建Service，不同类型的Service概念和使用方法请参见**服务概述**。

高级配置（可选）

- 升级策略：指定工作负载的升级方式及升级参数，支持滚动升级和替换升级，详情请参见**设置工作负载升级策略**。
- 实例管理策略（podManagementPolicy）：

  对于某些分布式系统来说，StatefulSet 的顺序性保证是不必要和/或者不应该的。这些系统仅要求唯一性和身份标志。

  - 有序策略：默认实例管理策略，有状态负载会逐个的、按顺序的进行部署、删除、伸缩实例, 只有前一个实例部署Ready或者删除完成后，有状态负载才会操作后一个实例。
  - 并行策略：支持有状态负载并行创建或者删除所有的实例，有状态负载发生变更时立刻在实例上生效。
- 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。
  - 负载亲和：提供常用的负载亲和策略，快速实现负载亲和部署。
    - 不配置：不设置负载亲和策略。
    - 优先多可用区部署：通过设置Pod间反亲和（podAntiAffinity）实现，优先将工作负载的Pod调度到不同可用区的节点上。
    - 强制多可用区部署：通过设置Pod间反亲和（podAntiAffinity）实现，强制将工作负载的Pod调度到不同可用区的节点上，如集群下节点支持的可用区数量小于实例数，工作负载的Pod无法全部运行。
    - 自定义亲和策略：根据需求自定义设置亲和与反亲和规则，详情请参见**设置工作负载亲和/反亲和调度（podAffinity/podAntiAffinity）**。
  - 节点亲和：提供常用的负载亲和策略，快速实现负载亲和部署。

    - 不配置：不设置节点亲和策略。
    - 指定节点调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点，若不指定，将根据集群默认调度策略随机调度。
    - 指定节点池调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点池，若不指定，将根据集群默认调度策略随机调度。
    - 自定义亲和策略：根据需求自定义设置亲和与反亲和规则，详情请参见**设置节点亲和调度（nodeAffinity）**。
- 容忍策略：容忍策略与节点的污点能力配合使用，允许（不强制）负载调度到带有与之匹配的污点的节点上，也可用于控制负载所在的节点被标记污点后负载的驱逐策略，详情请参见**设置容忍策略**。
- 标签与注解：以键值对形式为工作负载Pod添加标签或注解，填写完成后需单击“确认添加”。关于标签与注解的作用及配置说明，请参见**设置标签与注解**。
- DNS配置：为工作负载单独配置DNS策略，详情请参见**工作负载DNS配置说明**。
- 性能管理配置：使用应用性能管理（APM）服务，为JAVA程序提供更精准的问题分析与定位，详情请参见**设置性能管理配置**。
- 网络配置：
  - Pod入/出口带宽限速：支持为Pod设置入/出口带宽限速，详情请参见**为Pod配置QoS**。
  - 是否开启固定IP：仅支持该功能的集群显示该选项，开启后可设置Pod IP的回收时间间隔，详情请参见**为Pod配置固定IP**。
  - 是否开启指定容器网络配置：仅支持该功能的集群显示该选项，开启指定容器网络配置，工作负载使用指定的容器网络配置中的容器子网跟安全组来创建，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。
  - 指定容器网络配置名：只支持选择关联资源类型为工作负载类型的自定义容器网络配置。

步骤4 单击右下角“创建工作负载”。

----结束

通过 kubectl 命令行创建

本示例以nginx为例，并使用volumeClaimTemplates动态挂载云硬盘。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建一个名为nginx-statefulset.yaml的文件。

其中，nginx-statefulset.yaml为自定义名称，您可以随意命名。

vi nginx-statefulset.yaml

以下内容仅为示例，若需要了解statefulset的详细内容，请参考**kubernetes官方文档**。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
```



```
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          imagePullPolicy: IfNotPresent
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:
            - name: test
              readOnly: false
              mountPath: /usr/share/nginx/html
              subPath: ''
      imagePullSecrets:
        - name: default-secret
      dnsPolicy: ClusterFirst
      volumes: []
  serviceName: nginx-svc
  replicas: 2
  volumeClaimTemplates:  #动态挂载云硬盘示例
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: test
        namespace: default
        annotations:
          everest.io/disk-volume-type: SAS  # 云硬盘的类型
        labels:
          failure-domain.beta.kubernetes.io/region: ap-southeast-1  #云硬盘所在的区域
          failure-domain.beta.kubernetes.io/zone:    #云硬盘所在的可用区，必须和工作负载部署的节点可用区一
致
      spec:
        accessModes:
          - ReadWriteOnce  # 云硬盘必须为ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
        storageClassName: csi-disk #StorageClass的名称，云硬盘为csi-disk
  updateStrategy:
    type: RollingUpdate
```

vi nginx-headless.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  namespace: default
  labels:
    app: nginx
spec:
  selector:
    app: nginx
    version: v1
  clusterIP: None
  ports:
    - name: nginx
      targetPort: 80
      nodePort: 0
      port: 80
```

```
    protocol: TCP
  type: ClusterIP
```

步骤3 创建工作负载以及对应headless服务。

kubectl create -f nginx-statefulset.yaml

回显如下，表示有状态工作负载（stateful）已创建成功。

```
statefulset.apps/nginx created
```

kubectl create -f nginx-headless.yaml

回显如下，表示对应headless服务已创建成功。

```
service/nginx-svc created
```

步骤4 若工作负载需要被访问（集群内访问或节点访问），您需要设置访问方式，具体请参见网络创建对应服务。

----结束

5.2.3 创建守护进程集（DaemonSet）

操作场景

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

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

使用DaemonSet的一些典型用法：

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

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

前提条件

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

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载的信息。

基本信息

- 负载类型：选择守护进程DaemonSet。工作负载类型的介绍请参见**工作负载概述**。
- 负载名称：填写工作负载的名称。请输入1到63个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。
- 命名空间：选择工作负载的命名空间，默认为default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见**创建命名空间**。
- 容器运行时：CCE Standard集群默认使用普通运行时，CCE Turbo集群可以使用普通运行时或安全运行时。具体区别请参见**安全运行时与普通运行时**。
- 时区同步：选择是否开启时区同步。开启后容器与节点使用相同时区（时区同步功能依赖容器中挂载的本地磁盘，请勿修改删除），时区同步详细介绍请参见**设置时区同步**。

容器配置

- 容器信息

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

  – 基本信息：配置容器的基本信息。

参数	说明
容器名称	为容器命名。
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见**使用第三方镜像**。
镜像版本	选择需要部署的镜像版本。
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。 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**设置容器规格**。
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。 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**设置容器规格**。

参数	说明
GPU配额（可选）	当集群中包含GPU节点时，才能设置GPU配额，且集群中需安装**CCE AI套件（NVIDIA GPU）**插件。 ■ 不限制：表示不使用GPU。 ■ 独享：单个容器独享GPU。 ■ 共享：容器需要使用的GPU百分比，例如设置为10%，表示该容器需使用GPU资源的10%。 关于如何在集群中使用GPU，请参见**使用Kubernetes默认GPU调度**。
NPU配额（可选）	使用NPU芯片的数量，必须为整数，且必须安装**CCE AI套件（Ascend NPU）**插件后才能使用。 关于如何在集群中使用NPU，请参见**NPU调度**。
特权容器（可选）	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。
初始化容器（可选）	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**。

  - 生命周期（可选）：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**。
  - 健康检查（可选）：根据需求选择是否设置存活探针、就绪探针及启动探针，详情请参见**设置容器健康检查**。
  - 环境变量（可选）：支持通过键值对的形式为容器运行环境设置变量，可用于把外部信息传递给Pod中运行的容器，可以在应用部署后灵活修改，详情请参见**设置环境变量**。
  - 数据存储（可选）：在容器内挂载本地存储或云存储，不同类型的存储使用场景及挂载方式不同，详情请参见**存储**。
  - 安全设置（可选）：对容器权限进行设置，保护系统和其他容器不受其影响。请输入用户ID，容器将以当前用户权限运行。
  - 容器日志（可选）：容器标准输出日志将默认上报至 AOM 服务，无需独立配置。您可以手动配置日志采集路径，详情请参见**通过ICAgent采集容器日志（不推荐）**。

    如需要关闭当前负载的标准输出，您可在**标签与注解**中添加键为kubernetes.AOM.log.stdout，值为[]的注解，即可关闭当前负载下全部容器的标准输出。该注解的使用方法请参见**表5-14**。

- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为default-secret，使用default-secret可访问SWR镜像仓库的镜像。default-secret详细说明请参见**default-secret**。

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

服务配置（可选）

服务（Service）可为Pod提供外部访问。每个Service有一个固定IP地址，Service将访问流量转发给Pod，而且Service可以为这些Pod自动实现负载均衡。

您也可以在创建完工作负载之后再创建Service，不同类型的Service概念和使用方法请参见**服务概述**。

高级配置（可选）

- 升级策略：指定工作负载的升级方式及升级参数，支持滚动升级和替换升级，详情请参见**设置工作负载升级策略**。
- 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，支持节点亲和。
  - 节点亲和：提供常用的负载亲和策略，快速实现负载亲和部署。
    - 不配置：不设置节点亲和策略。
    - 指定节点调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点，若不指定，将根据集群默认调度策略随机调度。
    - 指定节点池调度：通过设置节点亲和（nodeAffinity）实现，指定工作负载的Pod部署的节点池，若不指定，将根据集群默认调度策略随机调度。
    - 自定义亲和策略：根据需求自定义设置亲和与反亲和规则，详情请参见**设置节点亲和调度（nodeAffinity）**。
- 容忍策略：容忍策略与节点的污点能力配合使用，允许（不强制）负载调度到带有与之匹配的污点的节点上，也可用于控制负载所在的节点被标记污点后负载的驱逐策略，详情请参见**设置容忍策略**。
- 标签与注解：以键值对形式为工作负载Pod添加标签或注解，填写完成后需单击“确认添加”。关于标签与注解的作用及配置说明，请参见**设置标签与注解**。
- DNS配置：为工作负载单独配置DNS策略，详情请参见**工作负载DNS配置说明**。
- 性能管理配置：使用应用性能管理（APM）服务，为JAVA程序提供更精准的问题分析与定位，详情请参见**设置性能管理配置**。
- 网络配置：
  - Pod入/出口带宽限速：支持为Pod设置入/出口带宽限速，详情请参见**为Pod配置QoS**。
  - 是否开启指定容器网络配置：仅支持该功能的集群显示该选项，开启指定容器网络配置，工作负载使用指定的容器网络配置中的容器子网跟安全组来创建，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。
  - 指定容器网络配置名：只支持选择关联资源类型为工作负载类型的自定义容器网络配置。

步骤4 单击右下角“创建工作负载”。

----结束

通过 kubectl 命令行创建

本节以nginx工作负载为例，说明kubectl命令创建工作负载的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建一个名为nginx-daemonset.yaml的描述文件。其中，nginx-daemonset.yaml为自定义名称，您可以随意命名。

vi nginx-daemonset.yaml

描述文件内容如下。此处仅为示例，daemonset的详细说明请参见**kubernetes官方文档**。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-daemonset
  labels:
    app: nginx-daemonset
spec:
  selector:
    matchLabels:
      app: nginx-daemonset
  template:
    metadata:
      labels:
        app: nginx-daemonset
    spec:
      nodeSelector:                # 节点选择，当节点拥有daemon=need时才在节点上创建Pod
        daemon: need
      containers:
      - name: nginx-daemonset
        image: nginx:alpine
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
      imagePullSecrets:
      - name: default-secret
```

这里可以看出没有Deployment或StatefulSet中的replicas参数，因为是每个节点固定一个。

Pod模板中有个nodeSelector，指定了只在有“daemon=need”的节点上才创建Pod，DaemonSet只在指定标签的节点上创建Pod。如果需要在每一个节点上创建Pod可以删除该标签。

步骤3 创建daemonset。

kubectl create -f nginx-daemonset.yaml

回显如下表示已开始创建daemonset。

```
daemonset.apps/nginx-daemonset created
```

步骤4 查看daemonset状态。

kubectl get ds

```
$ kubectl get ds
NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
nginx-daemonset   1         1         0       1            0           daemon=need     116s
```

步骤5 若工作负载需要被访问（集群内访问或节点访问），您需要设置访问方式，具体请参见**网络**创建对应服务。

----结束

5.2.4 创建普通任务（Job）

操作场景

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

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

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

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

前提条件

已创建资源，具体操作请参见**创建节点**。若已有集群和节点资源，无需重复操作。

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载的信息。

基本信息

- 负载类型：选择任务Job。工作负载类型的介绍请参见**工作负载概述**。
- 负载名称：填写工作负载的名称。请输入1到63个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。
- 命名空间：选择工作负载的命名空间，默认为default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见**创建命名空间**。
- 实例数量：填写实例的数量，即工作负载Pod的数量。
- 容器运行时：CCE Standard集群默认使用普通运行时，CCE Turbo集群可以使用普通运行时或安全运行时。具体区别请参见**安全运行时与普通运行时**。

容器配置

- 容器信息

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

  - 基本信息：配置容器的基本信息。

参数	说明
容器名称	为容器命名。

参数	说明
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见**使用第三方镜像**。
镜像版本	选择需要部署的镜像版本。
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。 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**设置容器规格**。
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。 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**设置容器规格**。
GPU配额（可选）	当集群中包含GPU节点时，才能设置GPU配额，且集群中需安装**CCE AI套件（NVIDIA GPU）**插件。 ■ 不限制：表示不使用GPU。 ■ 独享：单个容器独享GPU。 ■ 共享：容器需要使用的GPU百分比，例如设置为10%，表示该容器需使用GPU资源的10%。 关于如何在集群中使用GPU，请参见**使用Kubernetes默认GPU调度**。
NPU配额（可选）	使用NPU芯片的数量，必须为整数，且必须安装**CCE AI套件（Ascend NPU）**插件后才能使用。 关于如何在集群中使用NPU，请参见**NPU调度**。
特权容器（可选）	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。
初始化容器（可选）	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**。

  - 生命周期（可选）：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**。
  - 环境变量（可选）：支持通过键值对的形式为容器运行环境设置变量，可用于把外部信息传递给Pod中运行的容器，可以在应用部署后灵活修改，详情请参见**设置环境变量**。
  - 数据存储（可选）：在容器内挂载本地存储或云存储，不同类型的存储使用场景及挂载方式不同，详情请参见**存储**。

    说明

    负载实例数大于1时，不支持挂载云硬盘类型的存储。
  - 容器日志（可选）：容器标准输出日志将默认上报至 AOM 服务，无需独立配置。您可以手动配置日志采集路径，详情请参见**通过ICAgent采集容器日志（不推荐）**。

    如需要关闭当前负载的标准输出，您可在**标签与注解**中添加键为kubernetes.AOM.log.stdout，值为[]的注解，即可关闭当前负载下全部容器的标准输出。该注解的使用方法请参见**表5-14**。
- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为default-secret，使用default-secret可访问SWR镜像仓库的镜像。default-secret详细说明请参见**default-secret**。
- GPU显卡（可选）：默认为不限制。当集群中存在GPU节点时，工作负载实例可以调度到指定GPU显卡类型的节点上。

高级配置（可选）

- 标签与注解：以键值对形式为工作负载Pod添加标签或注解，填写完成后需单击“确认添加”。关于标签与注解的作用及配置说明，请参见**设置标签与注解**。
- 任务设置：
  - 并行数：任务负载执行过程中允许同时创建的最大实例数，并行数应不大于实例数。
  - 超时时间（秒）：当任务执行超出该时间时，任务将会被标识为执行失败，任务下的所有实例都会被删除。为空时表示不设置超时时间。
  - 完成模式：
    - 非索引：当执行成功的Pod数达到实例数时， Job执行成功。Job中每一个Pod都是同质的，Pod之间是独立无关。
    - 索引：系统会为每个Pod分配索引值，取值为 0 到实例数-1。每个分配了索引的Pod都执行成功，则Job执行成功。索引模式下，Job中的Pod命名遵循$(job-name)-$(index)模式。
  - 挂起任务：默认任务创建后被立即执行。选择挂起任务后，任务创建后处于挂起状态；将其关闭后，任务继续执行。
- 网络配置：
  - Pod入/出口带宽限速：支持为Pod设置入/出口带宽限速，详情请参见**为Pod配置QoS**。
  - 是否开启指定容器网络配置：仅支持该功能的集群显示该选项，开启指定容器网络配置，工作负载使用指定的容器网络配置中的容器子网跟安全组来创建，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。
  - 指定容器网络配置名：只支持选择关联资源类型为工作负载类型的自定义容器网络配置。

步骤4 单击右下角“创建工作负载”。

----结束

使用 kubectl 创建 Job

Job的关键配置参数如下所示：

- .spec.completions表示Job结束需要成功运行的Pod个数，默认为1。
- .spec.parallelism表示并行运行的Pod的个数，默认为1。
- .spec.backoffLimit表示失败Pod的最大重试次数，超过这个次数不会继续重试。
- .spec.activeDeadlineSeconds表示Pod运行时间，一旦达到这个时间，Job及其所有的Pod都会停止。且activeDeadlineSeconds优先级高于backoffLimit，即到达activeDeadlineSeconds的Job会忽略backoffLimit的设置。

根据.spec.completions和.spec.parallelism的设置，可以将Job划分为以下几种类型。

表 5-3 任务类型

Job类型	说明	.spec.completions	.spec.parallelism
一次性Job	创建一个Pod直至其成功结束。	1	1
固定结束次数的Job	依次创建一个Pod运行直至成功结束的Pod个数到达到.spec.completions的数值。	>1	1
固定结束次数的并行Job	依次创建多个Pod运行直至成功结束的Pod个数到达到.spec.completions的数值。	>1	>1
带工作队列的并行Job	创建一个或多个Pod，从工作队列中取走对应的任务并处理，完成后将任务从队列中删除后退出，详情请参见**使用工作队列进行精细的并行处理**。	不填写	>1或=1

以下是一个Job配置示例，保存在myjob.yaml中，其计算π到2000位并打印输出。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob
spec:
  completions: 50       # Job结束需要运行50个Pod，这个示例中就是打印π 50次
  parallelism: 5       # 并行5个Pod
  backoffLimit: 5       # 最多重试5次
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never     # 对于Job，只能设置为Never或者OnFailure。对于其他controller（比如Deployment）可以设置为Always
```



```
    imagePullSecrets:
      - name: default-secret
```

运行该任务，如下：

步骤1 启动这个Job。

```
[root@k8s-master k8s]# kubectl apply -f myjob.yaml
job.batch/myjob created
```

步骤2 查看这个Job。

kubectl get job

```
[root@k8s-master k8s]# kubectl get job
NAME    COMPLETIONS   DURATION   AGE
myjob   50/50         23s        3m45s
```

COMPLETIONS为 50/50 表示成功运行了这个Job。

步骤3 查看Pod的状态。

kubectl get pod

```
[root@k8s-master k8s]# kubectl get pod
NAME          READY   STATUS      RESTARTS   AGE
myjob-29qlw   0/1     Completed   0          4m5s
...
```

状态为Completed表示这个job已经运行完成。

步骤4 查看这个Pod的日志。

kubectl logs <pod_name>

```
# kubectl logs myjob-29qlw
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912983367336244065664308602139494639522473719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892354201995611212902196086403441815981362977477130996051870721134999999837297804995105973173281609631859502445945534690830264252230825334468503526193118817101000313783875288658753320838142061717766914730359825349042875546873115956286388235378759375195778185778053217122680661300192787661119590921642019893809525720106548586327886593615338182796823030195203530185296899577362259941389124972177528347913151557485724245415069595082953311686172785588907509838175463746493931925506040092770167113900984882401285836160356370766010471018194295559619894676783744944825537977472684710404753464620804668425906949129331367702898915210475216205696602405803815019351125338243003558764024749647326391419927260426992279678235478163600934172164121992458631503028618297455570674983850549458858692699569092721079750930295532116534498720275596023648066549911988183479775356636980742654252786255181841757467289097777279380008164706001614524919217321721477235014144197356854816136115735255213347574184946843852332390739414333454776241686251898356948556209921922218427255025425688767179049460165346680498862723279178608578438382796797668145410095388378636095068006422512520511739298489608412848862694560424196528502221066118630674427862203919494504712371378696095636437191728746776465757396241389086583264599581339047802759015
```

----结束

相关操作

普通任务创建完成后，您还可执行**表5-4**中操作。

表 5-4 其他操作

操作	操作说明
编辑YAML	单击任务名称后的“更多 > 编辑YAML”，可编辑当前任务对应的YAML文件。
删除普通任务	1. 选择待删除的任务，单击操作列的“更多 > 删除”。 2. 单击“是”。 任务删除后将无法恢复，请谨慎操作。

5.2.5 创建定时任务（CronJob）

操作场景

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

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

- 在给定时间点只运行一次。
- 在给定时间点周期性地运行。

CronJob的典型用法如下所示：

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

前提条件

已创建资源，具体操作请参见**创建节点**。

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载的信息。

基本信息

- 负载类型：选择定时任务CronJob。工作负载类型的介绍请参见**工作负载概述**。
- 负载名称：填写工作负载的名称。请输入1到63个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。
- 命名空间：选择工作负载的命名空间，默认为default。您可以单击后面的“创建命名空间”，命名空间的详细介绍请参见**创建命名空间**。
- 容器运行时：CCE Standard集群默认使用普通运行时，CCE Turbo集群可以使用普通运行时或安全运行时。具体区别请参见**安全运行时与普通运行时**。

容器配置

- 容器信息

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

  - 基本信息：配置容器的基本信息。

参数	说明
容器名称	为容器命名。
更新策略	镜像更新/拉取策略。可以勾选“总是拉取镜像”，表示每次都从镜像仓库拉取镜像；如不勾选则优使用节点已有的镜像，如果没有这个镜像再从镜像仓库拉取。
镜像名称	单击后方“选择镜像”，选择容器使用的镜像。 如果需要使用第三方镜像，请参见**使用第三方镜像**。
镜像版本	选择需要部署的镜像版本。
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。 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**设置容器规格**。
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。 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**设置容器规格**。
GPU配额（可选）	当集群中包含GPU节点时，才能设置GPU配额，且集群中需安装**CCE AI套件（NVIDIA GPU）**插件。 ■ 不限制：表示不使用GPU。 ■ 独享：单个容器独享GPU。 ■ 共享：容器需要使用的GPU百分比，例如设置为10%，表示该容器需使用GPU资源的10%。 关于如何在集群中使用GPU，请参见**使用Kubernetes默认GPU调度**。
NPU配额（可选）	使用NPU芯片的数量，必须为整数，且必须安装**CCE AI套件（Ascend NPU）**插件后才能使用。 关于如何在集群中使用NPU，请参见**NPU调度**。
特权容器（可选）	特权容器是指容器里面的程序具有一定的特权。 若选中，容器将获得超级权限，例如可以操作宿主机上面的网络设备、修改内核参数等。

参数	说明
初始化容器（可选）	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**。

  - 生命周期（可选）：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**。
  - 环境变量（可选）：支持通过键值对的形式为容器运行环境设置变量，可用于把外部信息传递给Pod中运行的容器，可以在应用部署后灵活修改，详情请参见**设置环境变量**。
- 镜像访问凭证：用于访问镜像仓库的凭证，默认取值为default-secret，使用default-secret可访问SWR镜像仓库的镜像。default-secret详细说明请参见**default-secret**。
- GPU显卡（可选）：默认为不限制。当集群中存在GPU节点时，工作负载实例可以调度到指定GPU显卡类型的节点上。

定时规则

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

说明

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

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

- 时区：可以指定时区。如果未指定，将默认为控制节点所在的时区。
- 任务记录：可以设置保留执行成功或执行失败的任务个数，设置为0表示不保留。

高级配置（可选）

- 标签与注解：以键值对形式为工作负载Pod添加标签或注解，填写完成后需单击“确认添加”。关于标签与注解的作用及配置说明，请参见**设置标签与注解**。
- 网络配置：
  - Pod入/出口带宽限速：支持为Pod设置入/出口带宽限速，详情请参见**为Pod配置QoS**。
  - 是否开启指定容器网络配置：仅支持该功能的集群显示该选项，开启指定容器网络配置，工作负载使用指定的容器网络配置中的容器子网跟安全组来创建，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。
  - 指定容器网络配置名：只支持选择关联资源类型为工作负载类型的自定义容器网络配置。

步骤4 单击右下角“创建工作负载”。

----结束

使用 kubectl 创建 CronJob

CronJob的配置参数如下所示：

- .spec.schedule指定任务运行时间与周期，参数格式请参见**Cron**，例如“0 * * * *”或“@hourly”。
- .spec.jobTemplate指定需要运行的任务，格式与**使用kubectl创建Job**相同。
- .spec.startingDeadlineSeconds指定任务开始的截止期限。
- .spec.concurrencyPolicy指定任务的并发策略，支持Allow、Forbid和Replace三个选项。
  - Allow（默认）：允许并发运行Job。
  - Forbid：禁止并发运行，如果前一个还没有完成，则直接跳过下一个。
  - Replace：取消当前正在运行的Job，用一个新的来替换。

下面是一个CronJob的示例，保存在cronjob.yaml文件中。

说明

在v1.21及以上集群中，CronJob的apiVersion为**batch/v1**。

在v1.21以下集群中，CronJob的apiVersion为**batch/v1beta1**。

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
          imagePullSecrets:
            - name: default-secret
```

运行该任务，如下：

步骤1 创建CronJob。

kubectl create -f cronjob.yaml

命令行终端显示如下信息：

```
cronjob.batch/hello created
```

步骤2 执行如下命令，查看执行情况。

kubectl get cronjob

```
NAME     SCHEDULE     SUSPEND   ACTIVE    LAST SCHEDULE   AGE
hello    */1 * * * *   False     0         <none>          9s
```

kubectl get jobs

```
NAME               COMPLETIONS   DURATION   AGE
hello-1597387980   1/1           27s        45s
```

kubectl get pod

```
NAME                           READY     STATUS      RESTARTS   AGE
hello-1597387980-tjv8f         0/1       Completed   0          114s
hello-1597388040-lckg9         0/1       Completed   0          39s
```

kubectl logs hello-1597387980-tjv8f

```
Fri Aug 14 06:56:31 UTC 2020
Hello from the Kubernetes cluster
```

kubectl delete cronjob hello

```
cronjob.batch "hello" deleted
```


须知

删除CronJob时，对应的普通任务及相关的Pod都会被删除。

----结束

相关操作

定时任务创建完成后，您还可执行表5-5中操作。

表 5-5 其他操作

操作	操作说明
编辑YAML	单击定时任务名称后的“更多 > 编辑YAML”，可修改当前任务对应的YAML文件。
停止定时任务	1. 选择待停止的任务，单击操作列的“停止”。 2. 单击“是”。
删除定时任务	1. 选择待删除的任务，单击操作列的“更多 > 删除”。 2. 单击“是”。 任务删除后将无法恢复，请谨慎操作。

5.3 配置工作负载

5.3.1 安全运行时与普通运行时

相比于普通运行时，安全运行时可以让您的每个容器（准确地说是Pod）都运行在一个单独的微型虚拟机中，拥有独立的操作系统内核，以及虚拟化层的安全隔离。通过使用安全运行时，不同容器之间的内核、计算资源、网络都是隔离开的，保护了Pod的资源和数据不被其他Pod抢占和窃取。

CCE Turbo集群支持使用普通运行时和安全运行时创建工作负载，您可以根据业务需求选择使用，两者的区别如下：

分类	安全运行时	普通运行时
容器所在节点类型	弹性云服务器-物理机	弹性云服务器-虚拟机 弹性云服务器-物理机
容器引擎	Containerd	Docker、Containerd
容器运行时	Kata	runC
容器内核	独占内核	与宿主机共享内核
容器隔离方式	轻量虚拟机	Cgroups和Namespace

分类	安全运行时	普通运行时
容器引擎存储驱动	Device Mapper	• Docker容器：OverlayFS2 • Containerd容器：OverlayFS
Pod Overhead	内存：100MiB CPU：0.1Core Pod Overhead为安全容器本身资源占用。比如Pod申请的limits.cpu = 0.5Core和limits.memory = 256MiB，那么该Pod最终会申请0.6Core的CPU和356MiB的内存。	无
最小规格	内存：256MiB CPU：0.25Core 安全容器的CPU核数（单位为Core）与内存（单位为GiB）配比建议在1:1至1:8之间。例如CPU为0.5Core，则内存范围建议在512MiB-4GiB间。	无
容器引擎命令行	crictl	• Docker容器：docker • Containerd容器：crictl
Pod的计算资源	CPU和内存的request和limit必须一致	CPU和内存的request和limit可以不一致
主机网络（hostNetwork）	不支持	支持

Containerd和Docker命令行的使用方式对比，请参见**容器引擎说明**。

5.3.2 设置时区同步

创建工作负载时，支持设置容器使用节点相同的时区。您可以在创建工作负载时打开时区同步配置。

时区同步功能依赖容器中挂载的本地磁盘（HostPath），如下所示，开启时区同步后，Pod中会通过HostPath方式，将节点的“/etc/localtime”挂载到容器的“/etc/localtime”，从而使得节点和容器使用相同的时区配置文件。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
  namespace: default
spec:
```

```
replicas: 2
selector:
  matchLabels:
    app: test
template:
  metadata:
    labels:
      app: test
  spec:
    volumes:
      - name: vol-162979628557461404
        hostPath:
          path: /etc/localtime
          type: ''
    containers:
      - name: container-0
        image: 'nginx:alpine'
        volumeMounts:
          - name: vol-162979628557461404
            readOnly: true
            mountPath: /etc/localtime
        imagePullPolicy: IfNotPresent
    imagePullSecrets:
      - name: default-secret
```

5.3.3 设置镜像拉取策略

创建工作负载会从镜像仓库拉取容器镜像到节点上，当前Pod重启、升级时也会拉取镜像。

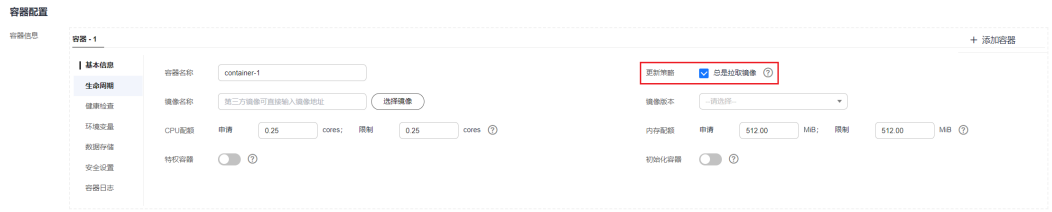
默认情况下容器镜像拉取策略imagePullPolicy是**IfNotPresent**，表示如果节点上有这个镜像就直接使用节点已有镜像，如果没有这个镜像就会从镜像仓库拉取。

容器镜像拉取策略还可以设置为**Always**，表示无论节点上是否有这个镜像，都会从镜像仓库拉取，并覆盖节点上的镜像。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
    resources:
      limits:
        cpu: 100m
        memory: 200Mi
      requests:
        cpu: 100m
        memory: 200Mi
    imagePullPolicy: Always
  imagePullSecrets:
  - name: default-secret
```

在CCE控制台也可以设置镜像拉取策略，在创建工作负载时设置“更新策略”：勾选表示总是拉取镜像（Always），不勾选则表示按需拉取镜像（IfNotPresent）。

图 5-4 设置更新策略



须知

建议您在制作镜像时，每次制作一个新的镜像都使用一个新的Tag，如果不更新Tag只更新镜像，当拉取策略选择为IfNotPresent时，CCE会认为当前节点已经存在这个Tag的镜像，不会重新拉取。

5.3.4 使用第三方镜像

操作场景

CCE支持拉取第三方镜像仓库的镜像来创建工作负载。

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

前提条件

使用第三方镜像时，请确保工作负载运行的节点可访问公网。

通过界面操作

步骤1 创建第三方镜像仓库的密钥。

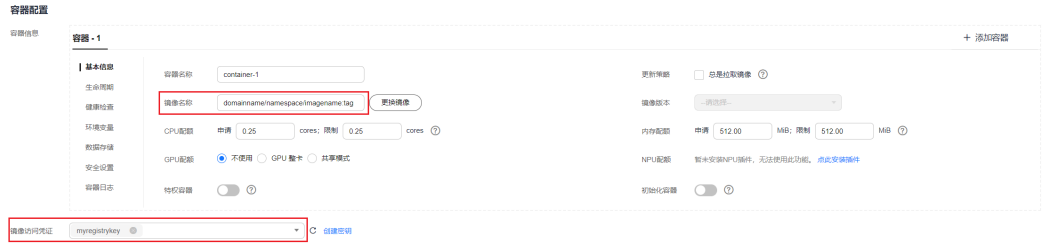
单击集群名称进入集群，在左侧导航栏选择“配置与密钥”，在右侧选择“密钥”页签，单击右上角“创建密钥”，密钥类型必须选择为kubernetes.io/dockerconfigjson。详细操作请参见**创建密钥**。

此处的“用户名”和“密码”请填写第三方镜像仓库的账号密码。

图 5-5 添加密钥

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

图 5-6 填写私有镜像地址



步骤3 填写其他参数后，单击“创建工作负载”。

----结束

使用 kubectl 创建第三方镜像仓库的密钥

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 通过kubectl创建认证密钥 ，该密钥类型为kubernetes.io/dockerconfigjson类型。

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

其中，*myregistrykey*为密钥名称，*default*为密钥所在的命名空间，其余参数如下所示。

- DOCKER_REGISTRY_SERVER：第三方镜像仓库的地址，如“www.3rdregistry.com”或“10.10.10.10:443”。
- DOCKER_USER：第三方镜像仓库的账号。
- DOCKER_PASSWORD：第三方镜像仓库的密码。

- DOCKER_EMAIL：第三方镜像仓库的邮箱。

步骤3 创建工作负载时使用第三方镜像，具体步骤请参见如下。

kubernetes.io/dockerconfigjson类型的密钥作为私有镜像获取的认证方式，以Pod为例，创建的myregistrykey作为镜像的认证方式。

```
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: default
spec:
  containers:
    - name: foo
      image: www.3rdregistry.com/janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey              #使用上面创建的密钥
```

----结束

5.3.5 设置容器规格

操作场景

CCE支持在创建工作负载时为添加的容器设置资源的需求量和限制，最常见的可设定资源是 CPU 和内存（ RAM ）大小。此外Kubernetes还支持其他类型的资源，可通过YAML设置。

申请与限制

在CPU配额和内存配额设置中，**申请**与**限制**的含义如下：

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

如果实例运行所在的节点具有足够的可用资源，实例可以使用超出申请的资源量，但不能超过限制的资源量。

例如，如果您将实例的内存申请值为1GiB、限制值为2GiB，而该实例被调度到一个具有8GiB CPU的节点上，且该节点上没有其他实例运行，那么该实例在负载压力较大的情况下可使用超过1GiB的内存，但内存使用量不得超过2GiB。若容器中的进程尝试使用超过2GiB的资源时，系统内核将会尝试将进程终止，出现内存不足（ OOM ）错误。

说明

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

配置说明

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

- CPU配额：CPU资源单位为核，可以通过数量或带单位后缀（ m ）的整数表达，例如数量表达式0.1核等价于表达式100m，但Kubernetes不允许设置精度小于1m的CPU资源。

表 5-6 CPU 配额说明

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

建议配置方法：

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

- 内存配额：内存资源默认单位为字节，或者也可以使用带单位后缀的整数来表达，例如100Mi。但需要注意单位大小写。

表 5-7 内存配额说明

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

建议配置方法：

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

说明

可分配资源：可分配量按照实例申请值(Request)计算，表示实例在该节点上可请求的资源上限，不代表节点实际可用资源（请参见**CPU和内存配额使用示例**）。 计算公式为：

- 可分配CPU = CPU总量 - 所有实例的CPU申请值 - 其他资源CPU预留值
- 可分配内存 = 内存总量 - 所有实例的内存申请值 - 其他资源内存预留值

CPU 和内存配额使用示例

假设集群中可调度的节点资源总量为4Core 8GiB，且已经在集群中部署了两个实例，其中实例1存在CPU和内存资源超分（即限制值>申请值），而实例2不存在资源超分。两个实例的规格设置如下：

实例	CPU申请	CPU限制	内存申请	内存限制
实例1	1Core	2Core	1GiB	4GiB

实例	CPU申请	CPU限制	内存申请	内存限制
实例2	2Core	2Core	2GiB	2GiB

那么节点上CPU和内存的资源使用情况如下：

- CPU可分配资源=4Core-（实例1申请的1Core+实例2申请的2Core）=1Core
- 内存可分配资源=8GiB-（实例1申请的1GiB+实例2申请的2GiB）=5GiB

此时节点还剩余1Core 5GiB的资源可供下一个新增的实例调度。

如果实例1处于业务高峰、负载压力较大时，会尝试在限制值范围内使用更多的CPU和内存，因此实际可用的资源将会小于1Core 5GiB。

其他资源配额

节点通常还可以具有本地的临时性存储（Ephemeral Storage），由本地挂载的可写入设备或者有时也用RAM来提供支持。临时性存储所存储的数据不提供长期可用性的保证，Pod通常可以使用本地临时性存储来实现缓冲区、保存日志等功能，也可以使用emptyDir类型的存储卷挂载到容器中。更多详情请参见**本地临时存储**。

Kubernetes支持在容器的定义中指定ephemeral-storage的申请值和限制值来管理本地临时性存储。 Pod中的每个容器可以设置以下属性：

- spec.containers[].resources.limits.ephemeral-storage
- spec.containers[].resources.requests.ephemeral-storage

以下示例中，Pod包含两个容器，每个容器的本地临时性存储申请值为2GiB，限制值为4GiB。 因此，整个Pod的本地临时性存储申请值是4GiB，限制值为8GiB，且emptyDir卷使用了500Mi的本地临时性存储。

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: container-1
    image: <example_app_image>
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  - name: container-2
    image: <example_log_aggregator_image>
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
```



```
    emptyDir:
      sizeLimit: 500Mi
```

5.3.6 设置容器生命周期

操作场景

CCE提供了回调函数，在容器的生命周期的特定阶段执行调用，比如容器在停止前希望执行某项操作，就可以注册相应的钩子函数。

目前提供的生命周期回调函数如下所示：

- **启动命令：**容器将会以该启动命令启动，请参见**启动命令**。
- **启动后处理：**容器启动后触发，请参见**启动后处理**。
- **停止前处理：**容器停止前触发。设置停止前处理，确保升级或实例删除时可提前将实例中运行的业务排水。详细请参见**停止前处理**。

启动命令

在默认情况下，镜像启动时会运行默认命令，如果想运行特定命令或重写镜像默认值，需要进行相应设置。

Docker的镜像拥有存储镜像信息的相关元数据，如果不设置生命周期命令和参数，容器运行时将运行镜像制作时提供的默认的命令和参数，Docker将这两个字段定义为ENTRYPOINT和 CMD。

如果在创建工作负载时填写了容器的运行命令和参数，将会覆盖镜像构建时的默认命令ENTRYPOINT、CMD，规则如下：

表 5-8 容器如何执行命令和参数

镜像ENTRYPOINT	镜像CMD	容器运行命令	容器运行参数	最终执行
[touch]	[/root/test]	未设置	未设置	[touch /root/test]
[touch]	[/root/test]	[mkdir]	未设置	[mkdir]
[touch]	[/root/test]	未设置	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

步骤1 登录CCE控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤2 在“启动命令”页签，输入运行命令和运行参数。

表 5-9 容器启动命令

命令方式	操作步骤
运行命令	输入可执行的命令，例如“/run/server”。 若运行命令有多个，需分行书写。 **说明** 多命令时，运行命令建议用/bin/sh或其他的shell，其他全部命令作为参数来传入。
运行参数	输入控制容器运行命令参数，例如--port=8080。 若参数有多个，多个参数以换行分隔。

----结束

启动后处理

步骤1 登录CCE控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤2 在“启动后处理”页签，设置启动后处理的参数。

表 5-10 启动后处理-参数说明

参数	说明

命令行脚本方式	在容器中执行指定的命令，配置为需要执行的命令。命令的格式为Command Args[1] Args[2]…（Command为系统命令或者用户自定义可执行程序，如果未指定路径则在默认路径下寻找可执行程序），如果需要执行多条命令，建议采用将命令写入脚本执行的方式。**不支持后台执行和异步执行的命令。**
	如需要执行的命令如下：
	`exec:`
	`  command:`
	`  - /install.sh`
	`  - install_agent`
	请在执行脚本中填写: /install install_agent。这条命令表示容器创建成功后将执行install.sh。

HTTP请求	发起一个HTTP调用请求。配置参数如下： ● 路径：请求的URL路径，可选项。 ● 端口：请求的端口，必选项。 ● 主机地址：请求的IP地址，可选项，默认为实例IP。

----结束

停止前处理

步骤1 登录CCE控制台，在创建工作负载时，配置容器信息，选择“生命周期”。

步骤2 在“停止前处理”页签，设置停止前处理的命令。

表 5-11 停止前处理

参数	说明

命令行脚本方式	在容器中执行指定的命令，配置为需要执行的命令。命令的格式为Command Args[1] Args[2]…（ Command为系统命令或者用户自定义可执行程序，如果未指定路径则在默认路径下寻找可执行程序），如果需要执行多条命令，建议采用将命令写入脚本执行的方式。
	如需要执行的命令如下：
	`exec:`
	`  command:`
	`  - /uninstall.sh`
	`  - uninstall_agent`
	请在执行脚本中填写: /uninstall uninstall_agent。这条命令表示容器结束前将执行uninstall.sh。

HTTP请求	发起一个HTTP调用请求。配置参数如下： ● 路径：请求的URL路径，可选项。 ● 端口：请求的端口，必选项。 ● 主机地址：请求的IP地址，可选项，默认为实例IP。

----结束

YAML 样例

本节以nginx为例，说明kubectl命令设置容器生命周期的方法。

在以下配置文件中，您可以看到postStart命令在容器目录/bin/bash下写了个**install.sh**命令。 preStop执行**uninstall.sh**命令。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        command:
        - sleep 3600                    #启动命令
        imagePullPolicy: Always
        lifecycle:
          postStart:
            exec:
              command:
              - /bin/bash
              - install.sh              #启动后命令
          preStop:
            exec:
              command:
              - /bin/bash
```



```
      - uninstall.sh          #停止前命令
  name: nginx
imagePullSecrets:
- name: default-secret
```

5.3.7 设置容器健康检查

操作场景

健康检查是指容器运行过程中，根据用户需要，定时检查容器健康状况。若不配置健康检查，如果容器内应用程序异常，Pod将无法感知，也不会自动重启去恢复。最终导致虽然Pod状态显示正常，但Pod中的应用程序异常的情况。

Kubernetes提供了三种健康检查的探针：

- **存活探针：**livenessProbe，用于检测容器是否正常，类似于执行ps命令检查进程是否存在。如果容器的存活检查失败，集群会对该容器执行重启操作；若容器的存活检查成功则不执行任何操作。
- **就绪探针：**readinessProbe，用于检查用户业务是否就绪，如果未就绪，则不转发流量到当前实例。一些程序的启动时间可能很长，比如要加载磁盘数据或者要依赖外部的某个模块启动完成才能提供服务。这时候程序进程已启动，但是并不能对外提供服务。这种场景下该检查方式就非常有用。如果容器的就绪检查失败，集群会屏蔽请求访问该容器；若检查成功，则会开放对该容器的访问。
- **启动探针**：startupProbe，用于探测应用程序容器什么时候启动了。 如果配置了这类探测器，就可以控制容器在启动成功后再进行存活性和就绪检查， 确保这些存活、就绪探针不会影响应用程序的启动。 这可以用于对启动慢的容器进行存活性检测，避免它们在启动运行之前就被终止。

检查方式

- **HTTP 请求检查**

  HTTP 请求方式针对的是提供HTTP/HTTPS服务的容器，集群周期性地对该容器发起HTTP/HTTPS GET请求，如果HTTP/HTTPS response返回码属于200~399范围，则证明探测成功，否则探测失败。使用HTTP请求探测必须指定容器监听的端口和HTTP/HTTPS的请求路径。

  例如：提供HTTP服务的容器，HTTP检查路径为：/health-check；端口为：80；主机地址可不填，默认为容器实例IP，此处以172.16.0.186为例。那么集群会周期性地对容器发起如下请求：GET http://172.16.0.186:80/health-check。您也可以为HTTP请求添加一个或多个请求头部，例如设置请求头名称为Custom-Header，对应的值为example。

  图 5-7 HTTP 请求检查

- **TCP 端口检查**

  对于提供TCP通信服务的容器，集群周期性地对该容器建立TCP连接，如果连接成功，则证明探测成功，否则探测失败。选择TCP端口探测方式，必须指定容器监听的端口。

  例如：有一个nginx容器，它的服务端口是80，对该容器配置了TCP端口探测，指定探测端口为80，那么集群会周期性地对该容器的80端口发起TCP连接，如果连接成功则证明检查成功，否则检查失败。

  图 5-8 TCP 端口检查

- **执行命令检查**

  命令检查是一种强大的检查方式，该方式要求用户指定一个容器内的可执行命令，集群会周期性地在容器内执行该命令，如果命令的返回结果是0则检查成功，否则检查失败。

  对于上面提到的TCP端口检查和HTTP请求检查，都可以通过执行命令检查的方式来替代：

  - 对于TCP端口探测，可以使用程序对容器的端口尝试connect，如果connect成功，脚本返回0，否则返回-1。
  - 对于HTTP请求探测，可以使用脚本命令来对容器尝试使用wget命令进行探测。

    wget http://127.0.0.1:80/health-check

    并检查response 的返回码，如果返回码在200~399 的范围，脚本返回0，否则返回-1。如下图：

    图 5-9 执行命令检查

须知

- 必须把要执行的程序放在容器的镜像里面，否则会因找不到程序而执行失败。
- 如果执行的命令是一个shell脚本，由于集群在执行容器里的程序时，不在终端环境下，因此不能直接指定脚本为执行命令，需要加上脚本解析器。比如脚本是**/data/scripts/health_check.sh**，那么使用执行命令检查时，指定的程序应该是**sh /data/scripts/health_check.sh**。

- **GRPC检查**

  GRPC检查可以为GRPC应用程序配置启动、活动和就绪探针，而无需暴露任何HTTP端点，也不需要可执行文件。Kubernetes可以通过GRPC 连接到工作负载并查询其状态。

须知

- GRPC检查仅在CCE v1.25及以上版本集群中支持。
- 使用GRPC检查时，您的应用需支持**GRPC健康检查协议**。
- 与HTTP和TCP探针类似，如果配置错误，都会被认作是探测失败，例如错误的端口、应用未实现健康检查协议等。

图 5-10 GRPC 检查

公共参数说明

表 5-12 公共参数说明

参数	参数说明
检测周期（periodSeconds）	探针检测周期，单位为秒。 例如，设置为30，表示每30秒检测一次。
延迟时间（initialDelaySeconds）	延迟检查时间，单位为秒，此设置与业务程序正常启动时间相关。 例如，设置为30，表明容器启动后30秒才开始健康检查，该时间是预留给业务程序启动的时间。

参数	参数说明
超时时间（timeoutSeconds）	超时时间，单位为秒。 例如，设置为10，表明执行健康检查的超时等待时间为10秒，如果超过这个时间，本次健康检查就被视为失败。若设置为0或不设置，默认超时等待时间为1秒。
成功阈值（successThreshold）	探测失败后，将状态转变为成功所需要的最小连续成功次数。例如，设置为1时，表明健康检查失败后，健康检查需要连续成功1次，才认为工作负载状态正常。 默认值是 1，最小值是 1。 存活和启动探测的这个值必须是 1。
最大失败次数（failureThreshold）	当探测失败时重试的次数。 存活探测情况下的放弃就意味着重新启动容器。就绪探测情况下的放弃 Pod 会被打上未就绪的标签。 默认值是 3，最小值是 1。

YAML 示例

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: <image_address>
    args:
    - /server
    livenessProbe:                # 存活探针
      httpGet:                    # 以HTTP请求检查为例
        path: /healthz            # HTTP检查路径为/healthz
        port: 80                  # 检查端口为80
        httpHeaders:              # 可选，请求头名称为Custom-Header，对应的值为Awesome
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:               # 就绪探针
      exec:                       # 以执行命令检查为例
        command:                  # 需要执行的命令
          - cat
          - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
    startupProbe:                 # 启动探针
      httpGet:                    # 以HTTP请求检查为例
        path: /healthz            # HTTP检查路径为/healthz
        port: 80                  # 检查端口为80
      failureThreshold: 30
      periodSeconds: 10
```

5.3.8 设置环境变量

操作场景

环境变量是指容器运行环境中设定的一个变量，环境变量可以在工作负载部署后修改，为工作负载提供极大的灵活性。

CCE中设置的环境变量与Dockerfile中的“ENV”效果相同。

须知

容器启动后，容器中的内容不应修改。如果修改配置项（例如将容器应用的密码、证书、环境变量配置到容器中），当容器重启（例如节点异常重新调度Pod）后，会导致配置丢失，业务异常。

配置信息应通过入参等方式导入容器中，以免重启后配置丢失。

环境变量支持如下几种方式设置。

- **自定义**：手动填写环境变量名称及对应的参数值。
- **配置项导入**：将配置项中所有键值都导入为环境变量。
- **配置项键值导入**：将配置项中某个键的值导入作为某个环境变量的值。例如图5-11中将configmap-example这个配置项中configmap_key的值configmap_value导入为环境变量key1的值，导入后容器中将会存在一个名为key1的环境变量，其值为configmap_value。
- **密钥导入**：将密钥中所有键值都导入为环境变量。
- **密钥键值导入**：将密钥中某个键的值导入作为某个环境变量的值。例如图5-11中将secret-example这个配置项中secret_key的值secret_value导入为环境变量key2的值，导入后容器中将会存在一个名为key2的环境变量，其值为secret_value。
- **变量/变量引用**：用Pod定义的字段作为环境变量的值。例如图5-11中将此Pod的名称导入为环境变量key3的值，导入后容器中将会存在一个名为key3的环境变量，其值为该Pod的名称。
- **资源引用**：用容器定义的资源申请值或限制值作为环境变量的值。例如图5-11中将容器container-1的CPU限制值导入为环境变量key4的值，导入后容器中将会存在一个名为key4的环境变量，其值为容器container-1的CPU限制值。

添加环境变量

步骤1 登录CCE控制台。

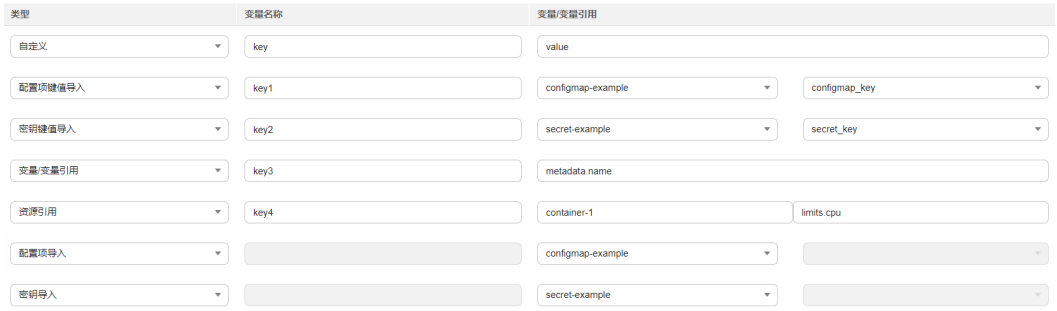
步骤2 单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建工作负载”。

步骤3 在创建工作负载时，在“容器配置”中修改容器信息，选择“环境变量”页签。

步骤4 设置环境变量。

- 单击“新增变量”，逐条增加环境变量，依次“配置类型”、“变量名称”和“变量/变量引用”。
- 单击“批量编辑自定义变量”，在编辑页面，按行输入自定义变量，格式为“变量名称=变量/变量引用”。

图 5-11 设置环境变量



----结束

YAML 样例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: env-example
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: env-example
  template:
    metadata:
      labels:
        app: env-example
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          imagePullPolicy: Always
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          env:
            - name: key                    # 自定义
              value: value
            - name: key1                   # 配置项键值导入
              valueFrom:
                configMapKeyRef:
                  name: configmap-example
                  key: configmap_key
            - name: key2                   # 密钥键值导入
              valueFrom:
                secretKeyRef:
                  name: secret-example
                  key: secret_key
            - name: key3                   # 变量引用，用Pod定义的字段作为环境变量的值
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: metadata.name
            - name: key4                   # 资源引用，用Container定义的字段作为环境变量的值
              valueFrom:
                resourceFieldRef:
                  containerName: container1
                  resource: limits.cpu
                  divisor: 1
```

```
      envFrom:
        - configMapRef:                # 配置项导入
            name: configmap-example
        - secretRef:                   # 密钥导入
            name: secret-example
    imagePullSecrets:
      - name: default-secret
```

环境变量查看

如果configmap-example和secret-example的内容如下。

```
$ kubectl get configmap configmap-example -oyaml
apiVersion: v1
data:
  configmap_key: configmap_value
kind: ConfigMap
...

$ kubectl get secret secret-example -oyaml
apiVersion: v1
data:
  secret_key: c2VjcmV0X3ZhbHVl              # c2VjcmV0X3ZhbHVl为secret_value的base64编码
kind: Secret
...
```

则进入Pod中查看的环境变量结果如下。

```
$ kubectl get pod
NAME                       READY   STATUS    RESTARTS   AGE
env-example-695b759569-lx9jp   1/1     Running   0          17m

$ kubectl exec env-example-695b759569-lx9jp  -- printenv
/ # env
key=value                           # 自定义环境变量
key1=configmap_value                    # 配置项键值导入
key2=secret_value                   # 密钥键值导入
key3=env-example-695b759569-lx9jp     # Pod的metadata.name
key4=1                              # container1这个容器的limits.cpu，单位为Core，向上取整
configmap_key=configmap_value         # 配置项导入，原配置项中的键值直接会导入结果
secret_key=secret_value               # 密钥导入，原密钥中的键值直接会导入结果
```

5.3.9 设置性能管理配置

操作场景

应用性能管理服务（APM）当前支持给JAVA类工作负载提供调用链、拓扑等监控能力。您可为JAVA类工作负载安装APM探针，以提供更精准的问题分析与定位，协助您高效解决应用难题。

工作负载创建时和创建后，均可以对JAVA类工作负载监控进行设置。

说明

- 如果您已经使用CCE部署了容器应用，您需要应用性能管理时，需要将容器服务上的JAVA应用通过Pinpoint探针接入到APM，详细介绍请参见**华为云容器应用接入APM**。

前提条件

若您还未开通APM服务，请前往APM控制台，并参照界面提示进行开通。

操作步骤

步骤1 登录CCE控制台。

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

步骤3 在创建工作负载时，在“高级设置”中找到“性能管理配置”。探针默认状态为“不启用”，您可根据需求选择“APM1.0探针”。开启后将会通过应用性能管理服务针对JAVA程序提供更精准的问题分析与定位。

说明

1. 启用APM探针后会自动新增一个名为init-pinpoint（APM1.0 探针）的初始化容器用来初始化探针, 并分配 0.25 Core CPU 和 250 MiB 内存供初始化容器使用。
2. 启用APM探针后会给所有业务容器自动添加环境变量：PAAS_MONITORING_GROUP、JAVA_TOOL_OPTIONS、PAAS_CLUSTER_ID。
3. 启用APM探针后会给所有业务容器自动挂载一个名为paas-apm（APM1.0 探针）的本地存储卷。

步骤4 填写探针相关参数。

APM1.0探针

- 监控组：输入监控组名称，如testapp。
- 探针版本：选择探针的版本。
- “探针升级策略”，默认为“重启自动升级”。
  - 重启自动升级：每次都尝试重新下载镜像。
  - 重启手动升级：如果本地有该镜像，则使用本地镜像，本地不存在时下载镜像。

步骤5 应用启动后，等待约3分钟，应用数据就会呈现在APM界面中，此时登录APM，您可以在APM上通过拓扑、调用链等进行应用性能优化，详细操作请参考**应用拓扑**。

----结束

修改性能管理配置

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“工作负载”，单击工作负载名称。

步骤3 在“性能管理配置”页签中，单击右下角“编辑”修改性能管理配置参数。

参数说明详情请参见**步骤4**。

----结束

5.3.10 设置工作负载升级策略

在实际应用中，升级是一个常见的场景，Deployment、StatefulSet和DaemonSet都能够很方便地支撑应用升级。

设置不同的升级策略，有如下两种。

- RollingUpdate：滚动升级，即逐步创建新Pod再删除旧Pod，为默认策略。
- Recreate：替换升级，即先把当前Pod删掉再重新创建Pod。

图 5-12 工作负载升级策略

升级参数说明

参数	说明	限制
最大浪涌（maxSurge）	与spec.replicas相比，可以有多少个Pod存在，默认值是25%。 比如spec.replicas为 4，那升级过程中就不能超过5个Pod存在，即按1个的步长升级，实际升级过程中会换算成数字，且换算会向上取整。这个值也可以直接设置成数字。	仅Deployment、DaemonSet支持配置。
最大无效实例数（maxUnavailable）	与spec.replicas相比，可以有多少个Pod失效，也就是删除的比例，默认值是25%。 比如spec.replicas为4，那升级过程中就至少有3个Pod存在，即删除Pod的步长是1。同样这个值也可以设置成数字。	仅Deployment、DaemonSet支持配置。
实例可用最短时间（minReadySeconds）	指定新创建的 Pod 在没有任意容器崩溃情况下的最小就绪时间， 只有超出这个时间 Pod 才被视为可用。默认值为 0（Pod 在准备就绪后立即将被视为可用）。	-
最大保留版本数（revisionHistoryLimit）	用来设定出于回滚目的所要保留的旧 ReplicaSet 数量。 这些旧 ReplicaSet 会消耗 etcd 中的资源，并占用 kubectl get rs 的输出。 每个 Deployment 修订版本的配置都存储在其 ReplicaSets 中；因此，一旦删除了旧的 ReplicaSet， 将失去回滚到 Deployment 的对应修订版本的能力。 默认情况下，系统保留 10 个旧 ReplicaSet，但其理想值取决于新 Deployment 的频率和稳定性。	-

参数	说明	限制
升级最大时长（progressDeadlineSeconds）	指定系统在报告 Deployment 进展失败 之前等待 Deployment 取得进展的秒数。 这类报告会在资源状态中体现为 Type=Progressing、Status=False、Reason=ProgressDeadlineExceeded。 Deployment 控制器将持续重试 Deployment。 将来，一旦实现了自动回滚，Deployment 控制器将在探测到这样的条件时立即回滚 Deployment。 如果指定，则此字段值需要大于 .spec.minReadySeconds 取值。	-
缩容时间窗（terminationGracePeriodSeconds）	优雅删除时间，默认为30秒，删除Pod时发送SIGTERM终止信号，然后等待容器中的应用程序终止执行，如果在terminationGracePeriodSeconds时间内未能终止，则发送SIGKILL的系统信号强行终止。	-

升级示例

Deployment的升级可以是声明式的，也就是说只需要修改Deployment的YAML定义即可，比如使用kubectl edit命令将上面Deployment中的镜像修改为nginx:alpine。修改完成后再查询ReplicaSet和Pod，发现创建了一个新的ReplicaSet，Pod也重新创建了。

```
$ kubectl edit deploy nginx

$ kubectl get rs
NAME               DESIRED   CURRENT   READY     AGE
nginx-6f9f58dffd   2         2         2         1m
nginx-7f98958cdf   0         0         0         48m

$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-6f9f58dffd-tdmqk   1/1       Running   0          1m
nginx-6f9f58dffd-tesqr   1/1       Running   0          1m
```

Deployment可以通过maxSurge和maxUnavailable两个参数控制升级过程中同时重新创建Pod的比例，这在很多时候是非常有用，配置如下所示。

```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
```

在前面的例子中，由于spec.replicas是2，如果maxSurge和maxUnavailable都为默认值25%，那实际升级过程中，maxSurge允许最多3个Pod存在（向上取整，2*1.25=2.5，取整为3），而maxUnavailable则不允许有Pod Unavailable（向上取整，2*0.75=1.5，取整为2），也就是说在升级过程中，一直会有2个Pod处于运行状态，每次新建一个Pod，等这个Pod创建成功后再删掉一个旧Pod，直至Pod全部为新Pod。

回滚

回滚也称为回退，即当发现升级出现问题时，让应用回到老版本。Deployment可以非常方便的回滚到老版本。

例如上面升级的新版镜像有问题，可以执行kubectl rollout undo命令进行回滚。

```
$ kubectl rollout undo deployment nginx
deployment.apps/nginx rolled back
```

Deployment之所以能如此容易的做到回滚，是因为Deployment是通过ReplicaSet控制Pod的，升级后之前ReplicaSet都一直存在，Deployment回滚做的就是使用之前的ReplicaSet再次把Pod创建出来。Deployment中保存ReplicaSet的数量可以使用revisionHistoryLimit参数限制，默认值为10。

5.3.11 设置容忍策略

容忍度（Toleration） 允许调度器将Pod调度至带有对应污点的节点上。 容忍度需要和**节点污点**相互配合，每个节点上都可以拥有一个或多个污点，对于未设置容忍度的Pod，调度器会根据节点上的污点效果进行选择性调度，可以用来避免Pod被分配到不合适的节点上。更多关于容忍度的使用示例请参见**污点和容忍度**。

污点可以指定多种效果，对应的容忍策略对Pod运行影响如下：

污点效果	Pod未设置对污点的容忍策略	Pod已设置对污点的容忍策略
NoExecute	• 已运行在该节点的Pod会立刻被驱逐。 • 未运行的Pod不会被调度到该节点。	• 未指定容忍时间窗（tolerationSeconds）：Pod可以在这个节点上一直运行。 • 已指定容忍时间窗（tolerationSeconds）：在容忍时间窗内，Pod还会在拥有污点的节点上运行，超出时间后会被驱逐。
PreferNoSchedule	• 已运行在该节点的Pod不会被驱逐。 • 未运行的Pod**尽量**不调度到该节点。	Pod可以在这个节点上一直运行。
NoSchedule	• 已运行在该节点的Pod不会被驱逐。 • 未运行的Pod不会被调度到该节点。	Pod可以在这个节点上一直运行。

通过控制台配置容忍策略

步骤1 登录CCE控制台。

步骤2 在创建工作负载时，在“高级设置”中找到“容忍策略”。

步骤3 添加污点容忍策略。

表 5-13 容忍策略设置参数说明

参数名	参数描述
污点键	节点的污点键。
操作符	• Equal：设置此操作符表示**准确匹配**指定污点键（必填）和污点值的节点。如果不填写污点值，则表示可以与所有污点键相同的污点匹配。 • Exists：设置此操作符表示**匹配存在**指定污点键的节点，此时容忍度不能指定污点值。若不填写污点键则可以容忍全部污点。
污点值	操作符为Equal时需要填写污点值。
污点策略	• 全部：表示匹配所有污点效果。 • NoSchedule：表示匹配污点效果为NoSchedule的污点。 • PreferNoSchedule：表示匹配污点效果为PreferNoSchedule的污点。 • NoExecute：表示匹配污点效果为NoExecute的污点。
容忍时间窗	即tolerationSeconds参数，当污点策略为NoExecute时支持配置。 在容忍时间窗内，Pod还会在拥有污点的节点上运行，超出时间后会被驱逐。

----结束

默认容忍策略说明

Kubernetes会自动给Pod添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的容忍度，且配置容忍时间窗（tolerationSeconds）为300s。这些默认容忍度策略表示当Pod运行的节点被打上这两个污点之一时，可以在5分钟内依旧保持运行在该节点上。

说明

DaemonSet中的Pod被创建时，针对以上污点自动添加的容忍度将不会指定容忍时间窗，即表示节点存在上述污点时，DaemonSet中的Pod一直不会被驱逐。

```
tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```


5.3.12 设置标签与注解

Pod 注解

CCE提供一些使用Pod的高级功能，这些功能使用时可以通过给YAML添加注解Annotation实现。具体的Annotation如下表所示。

表 5-14 Pod Annotation

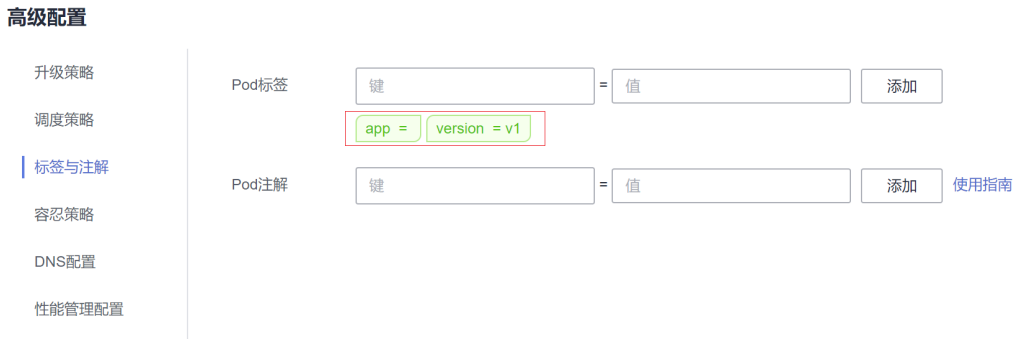
注解	说明	默认值

kubernetes.AOM.log.stdout	容器标准输出采集参数，不配置默认将全部容器的标准输出上报至AOM，可配置采集指定容器或全部不采集。
	示例：
	• 全部不采集：
	`kubernetes.AOM.log.stdout: '[]'`
	• 采集container-1和container-2容器：
	`kubernetes.AOM.log.stdout: '["container-1","container-2"]'`
		-

metrics.alpha.kubernetes.io/custom-endpoints	AOM监控指标上报参数，可将指定指标上报是AOM服务。 具体使用请参见**使用AOM监控自定义指标**。	-
prometheus.io/scrape	Prometheus指标上报参数，值为true表示当前负载开启上报。 具体使用请参见**使用云原生监控插件监控自定义指标**。	-
prometheus.io/path	Prometheus采集的url路径。 具体使用请参见**使用云原生监控插件监控自定义指标**。	/metrics
prometheus.io/port	Prometheus采集的endpoint端口号。 具体使用请参见**使用云原生监控插件监控自定义指标**。	-
prometheus.io/scheme	Prometheus采集协议，值可以填写http或https 具体使用请参见**使用云原生监控插件监控自定义指标**。	-
kubernetes.io/ingress-bandwidth	Pod的入口带宽 具体使用请参见**为Pod配置QoS**。	-
kubernetes.io/egress-bandwidth	Pod的出口带宽 具体使用请参见**为Pod配置QoS**。	-

Pod 标签

在控制台创建工作负载时，会默认为Pod添加如下标签，其中app的值为工作负载名称。



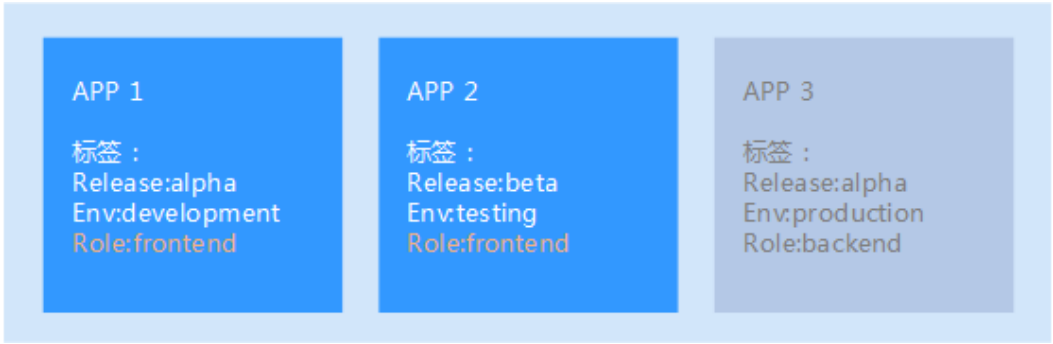
YAML示例如下：

```
...
spec:
  selector:
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      ...
```

您也可以根据需要为Pod添加其他标签，可用于设置工作负载亲和性与反亲和性调度。如下图，假设为工作负载（例如名称为APP1、APP2、APP3）定义了3个Pod标签：release、env、role。不同工作负载定义了不同的取值，分别为：

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

图 5-13 标签案例



例如，设置工作负载亲和性的“key/value”值为“role/backend”，则会选择APP3进行亲和性调度，详情请参见**设置工作负载亲和/反亲和调度（podAffinity/podAntiAffinity）**。

5.4 调度工作负载

5.4.1 工作负载调度策略概述

在Kubernetes中，工作负载调度的基本单位是Pod。创建工作负载时，调度器会自动对工作负载中的Pod进行合理分配，例如将Pod分散到资源充足的节点上。

虽然调度器的默认行为已经能够满足许多基本需求，但在一些特定场景下，用户可能需要更精细的控制Pod的部署位置。为了实现这一点，Kubernetes允许用户在工作负载定义中配置调度策略。例如：

- 将前端应用和后端应用部署在一起，有助于减少延迟，因为这两种类型的Pod可以共享相同的物理资源。
- 某类应用部署到某些特定的节点，确保关键应用总是运行在最优的硬件或配置上。
- 不同应用部署到不同的节点，有助于隔离应用，防止一个应用的问题影响到其他应用。

您可以使用以下方式来选择Kubernetes对Pod的调度策略：

表 5-15 工作负载调度策略

调度策略	YAML字段定义	说明	参考文档
节点选择	nodeSelector	最简单的调度形式，通过节点所具有的标签选择希望调度的目标节点，Kubernetes只会将Pod调度到拥有指定标签的节点上。	**设置指定节点调度（nodeSelector）**
节点亲和	nodeAffinity	节点亲和可以实现nodeSelector的能力，但其表达能力更强，您可以根据节点上的标签，使用**标签选择器**来筛选需要亲和的节点，支持**必须满足**和**尽量满足**的**亲和性规则**。 **说明** 如果同时指定nodeSelector和nodeAffinity，则两者必须都要满足，才能将Pod调度到候选节点上。	**设置节点亲和调度（nodeAffinity）**

调度策略	YAML字段定义	说明	参考文档
工作负载亲和/反亲和	podAffinity/ podAntiAffinity	您可以根据工作负载标签，使用**标签选择器**来筛选需要亲和/反亲和的Pod，并将新建的工作负载调度/不调度至目标Pod所在的节点（或节点组），同时支持**必须满足**和**尽量满足**的**亲和性规则**。 **说明** 工作负载亲和性和反亲和性需要一定的计算时间，因此在大规模集群中会显著降低调度的速度。在包含数百个节点的集群中，不建议使用这类设置。	**设置工作负载亲和/反亲和调度（podAffinity/ podAntiAffinity）**

亲和性规则

基于节点亲和或工作负载亲和/反亲和的调度策略还可以设置**必须满足**的硬约束和**尽量满足**的软约束，以满足更复杂的调度情况。

表 5-16 亲和性规则

规则类型	YAML字段定义	说明	配置示例
必须满足	requiredDuringSchedulingIgnoredDuringExecution	硬约束，即调度器只有在规则被满足的时候才能执行调度。	• **设置节点亲和调度（nodeAffinity）** • **设置工作负载亲和/反亲和调度（podAffinity/ podAntiAffinity）**
尽量满足	preferredDuringSchedulingIgnoredDuringExecution	软约束，即调度器会尝试寻找满足对应规则的目标对象。即使找不到匹配的目标，调度器仍然会调度该Pod。 在使用尽量满足的亲和性类型时，您可以为每个实例设置weight字段，其取值范围是1到100。 权重越高，调度的优先级越高。	

说明

在上述亲和规则中，YAML字段前半段requiredDuringScheduling或preferredDuringScheduling表示在调度时需要强制满足（require）或尽量满足（prefer）定义的标签规则。而后半段IgnoredDuringExecution表示如果节点标签在Kubernetes调度Pod后发生了变更，Pod仍将继续运行不会重新调度。但是如果该节点上的kubelet重启，kubelet会重新对节点亲和性规则进行校验，Pod仍会被调度至其他节点。

标签选择器

在创建调度策略时，您需要使用标签选择器的逻辑运算符来筛选标签值，最终确定需要亲和/反亲和的对象。

表 5-17 标签选择器

参数	说明	YAML示例
key	标签键名，满足筛选条件的对象需包含该键名的标签，且标签的取值满足标签值列表（values字段）和逻辑运算符的运算关系。	以下示例中，满足筛选条件的对象需**包含**键名为**topology.kubernetes.io/zone**的标签，并且该标签的取值为**az1**或**az2**。 matchExpressions: - key: **topology.kubernetes.io/zone** operator: **In** values: - **az1** - **az2**
operator	您可以使用逻辑运算符来确定标签值的筛选规则，所有逻辑运算符如下： • **In**：亲和/反亲和对象的标签**包含**在标签值列表（values字段）中。 • **NotIn**：亲和/反亲和对象的标签**不包含**在标签值列表（values字段）中。 • **Exists**：亲和/反亲和对象**存在**指定标签名，此时无需填写标签值列表（values字段）。 • **DoesNotExist**：亲和/反亲和对象**不存在**指定标签名，此时无需填写标签值列表（values字段）。 • **Gt**：仅在节点亲和性中设置，调度节点的标签值**大于**列表值（字符串比较）。 • **Lt**：仅在节点亲和性中设置，调度节点的标签值**小于**列表值（字符串比较）。	
values	标签值的列表。	

5.4.2 设置指定节点调度（nodeSelector）

在Kubernetes中，选择某个节点调度最简单的方式是在工作负载中配置nodeSelector字段，您可以通过nodeSelector字段设置希望调度的目标节点标签。Kubernetes只会将Pod调度到拥有指定标签的节点上。

前提条件

您需要为目标节点添加自定义标签，工作负载可根据该节点标签进行调度，操作步骤请参见**添加/删除节点标签**。

创建指定节点调度的工作负载

步骤1 使用kubectl连接集群，具体操作请参见**通过kubectl连接集群**。

步骤2 创建名为“nginx.yaml”的YAML文件，此处文件名可自定义。

为工作负载设置nodeSelector，例如，填写的键为“deploy_qa”，值为“true”，这表明该Pod将被调度到有deploy_qa=true标签的节点。示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:
        deploy_qa: "true"
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

步骤3 创建工作负载。

```
kubectl apply -f nginx.yaml
```

步骤4 验证Pod全部运行在目标节点上。

```
kubectl get pod -o wide
```

回显如下，节点192.168.0.103为包含deploy_qa=true标签的节点。

```
NAME                     READY   STATUS    RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
nginx-66859f4f48-xgp2g   1/1     Running   0          6h57m   172.16.3.0    192.168.0.103   <none>
<none>
nginx-66859f4f48-t9gqj   1/1     Running   0          6h57m   172.16.3.1    192.168.0.103   <none>
<none>
nginx-66859f4f48-2grhq   1/1     Running   0          6h57m   172.16.3.2    192.168.0.103   <none>
<none>
```

----结束

5.4.3 设置节点亲和调度（nodeAffinity）

Kubernetes在调度工作负载时支持将节点作为亲和对象，将工作负载调度至具有指定标签和标签值的节点上。例如，某些节点支持使用GPU算力，则可以使用节点亲和调度，确保高性能计算的Pod最终运行在GPU节点上。

配置节点亲和调度策略

您可以通过不同的方式配置节点亲和性调度策略，将Pod调度到满足条件的节点。

通过控制台配置

本文示例中，集群内已创建GPU节点，并设置标签为gpu=true，您可以通过该标签将Pod调度到GPU节点上。

步骤1 登录CCE控制台。

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

步骤3 在创建工作负载时，在“高级设置”中找到“调度策略”，选择节点亲和调度的策略类型，本示例中选择**自定义亲和策略**。创建工作负载的其余步骤详情请参见**创建工作负载**。

表 5-18 调度策略类型

参数	参数说明	示例
节点亲和	• **不配置**：不设置节点亲和策略。 • **指定节点调度**：指定工作负载Pod部署的节点。若不指定，将根据集群默认调度策略随机调度。 • **指定节点池调度**：指定工作负载Pod部署的节点池。若不指定，将根据集群默认调度策略随机调度。 • **自定义亲和策略**：根据节点标签实现灵活的调度策略，支持的亲和性规则请参见**表5-19**。选择合适的策略类型后可以添加对应的调度策略，参数详情请参见**表5-20**。	**自定义亲和策略**

步骤4 选择合适的节点亲和性规则，并单击+，添加相应的调度策略。本示例中在**必须满足**的类别下添加调度策略，表示节点必须拥有指定节点才可以调度该工作负载。

表 5-19 亲和性规则

参数	参数说明	示例
节点亲和性	• **必须满足**：即硬约束，设置必须要满足的条件，对应requiredDuringSchedulingIgnoredDuringExecution。添加多条“必须满足”规则时，只需要满足一条规则就会进行调度。 • **尽量满足**：即软约束，设置尽量满足的条件，对应preferredDuringSchedulingIgnoredDuringExecution。 添加多条“尽量满足”规则时，满足其中一条或者都不满足也会进行调度。	**必须满足**

步骤5 在右侧弹出窗口中单击“添加策略”，设置节点标签筛选规则。

您也可以单击“指定节点”或“指定可用区”通过控制台快速选择需要调度的节点或可用区。

“指定节点”和“指定可用区”本质也是通过标签实现，只是通过控制台提供了更为便捷的操作，无需手动填写节点标签和标签值。指定节点使用的是 kubernetes.io/

hostname 标签，指定可用区使用的是 failure-domain.beta.kubernetes.io/zone 标签。

表 5-20 节点亲和性调度策略设置参数说明

参数	参数说明	示例
权重	仅支持在“尽量满足”策略中添加。权重的取值范围为1-100，调度器在进行调度时会将该权重视为一个附加的评分项，并将其与节点的其他优先级函数评分相加。最终，调度器会将Pod调度到总分最大的节点上。	-
标签名	设置节点亲和性时，填写需要匹配的节点标签。 该标签可以使用系统默认的标签，也可以使用自定义标签。	**gpu**
操作符	可以设置六种匹配关系（In、NotIn、Exists、DoesNotExist、Gt、Lt）。 • **In**：亲和/反亲和对象的标签在标签值列表（values字段）中。 • **NotIn**：亲和/反亲和对象的标签不在标签值列表（values字段）中。 • **Exists**：亲和/反亲和对象存在指定标签名。 • **DoesNotExist**：亲和/反亲和对象不存在指定标签名。 • **Gt**：调度节点的标签值大于列表值（字符串比较）。 • **Lt**：调度节点的标签值小于列表值（字符串比较）。	**In**
标签值	设置节点亲和性时，填写节点标签对应的标签值。	**true**

步骤6 调度策略添加完成后，单击“创建工作负载”。

步骤7 验证Pod全部运行在目标节点上。

1. 在集群控制台左侧导航栏中选择“工作负载”。
2. 单击工作负载名称，进入详情页面，查看实例列表，验证Pod全部运行在目标节点上，即节点包含gpu=true标签。

----结束

通过 YAML 配置

工作负载节点亲和性规则通过节点标签实现。CCE集群中节点在创建时会自动添加一些标签，常用的节点标签如下（更多标签请参见**节点固有标签**）：

- topology.kubernetes.io/zone：表示节点所在的可用区（availability zone），可在指定可用区调度时使用。
- kubernetes.io/hostname：节点的hostname，可在指定节点调度时使用。
- cce.cloud.com/cce-nodepool：节点所属的节点池，可在指定节点池调度时使用。

本示例中，**必须满足**的规则表示调度的节点必须包含一个键名为gpu的标签，且标签值为true。而**尽量满足**规则表示根据节点可用区的标签topology.kubernetes.io/zone进行优先级排序，尽量将Pod调度至可用区az1的节点上。设置节点亲和性示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  gpu
  labels:
    app:  gpu
spec:
  selector:
    matchLabels:
      app: gpu
  replicas: 3
  template:
    metadata:
      labels:
        app:  gpu
    spec:
      containers:
      - image:  nginx:alpine
        name:  gpu
        resources:
          requests:
            cpu: 100m
            memory: 200Mi
          limits:
            cpu: 100m
            memory: 200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:  # 设置调度策略
        nodeAffinity:  # 表示节点亲和性调度
          requiredDuringSchedulingIgnoredDuringExecution:  # 表示必须满足的调度策略
            nodeSelectorTerms:     # 根据节点标签选择满足条件的节点
              - matchExpressions:    # 节点标签匹配规则
```



```
          - key: gpu   # 节点标签的键为gpu
            operator: In  # 表示存在values列表中的值即满足规则
            values:   # 节点标签的值为true
            - "true"
      preferredDuringSchedulingIgnoredDuringExecution:  # 表示尽量满足的调度策略
        - weight: 100  # 使用尽量满足策略时可设置优先级，取值为1-100，数值越大优先级越高
          preference:  # 使用尽量满足策略时，设置优先选择的节点标签匹配规则
            matchExpressions:   # 节点标签匹配规则
              - key: topology.kubernetes.io/zone   # 节点可用区的标签
                operator: In  # 表示存在values列表中的值即满足规则
                values:   # 节点标签的值为az1
                - "az1"
```

说明

节点亲和性调度不存在反亲和策略，如果需要实现节点反亲和，您可使用NotIn和DoesNotExist操作符，反向筛选节点标签值即可。

5.4.4 设置工作负载亲和/反亲和调度（podAffinity/podAntiAffinity）

工作负载亲和/反亲和调度是Kubernetes提供的任务调度方式，可以使用工作负载作为亲和对象，灵活地将新建的工作负载调度到与其相关或无关的节点上，可以有效地提高集群的利用率。

例如，通信频繁的前端应用Pod和后端应用Pod可优先调度到同一个节点或同一个可用区，减少网络延迟。工作负载亲和/反亲和的示意如下：

1. 首先，拓扑域（根据topologyKey划分）通过节点的标签和标签值划分节点范围，将节点分为不同的拓扑域。

   例如，topologyKey为prefer，表示可以通过节点标签prefer划分拓扑域。拓扑域1的范围为带有prefer=true标签的节点，拓扑域2的范围为带有prefer=false标签的节点，拓扑域3的范围为不带prefer标签的节点。

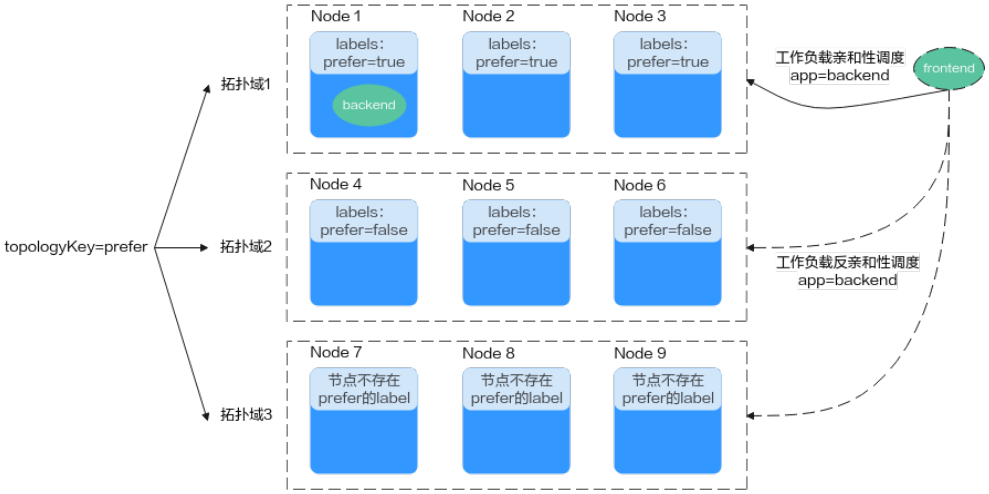
2. 然后，根据工作负载标签名、操作符、标签值确定需要亲和/反亲和的工作负载对象。

   例如，标签选择器筛选出需要亲和/反亲和的对象为带有app=backend的工作负载。

3. 最后，调度器选择目标工作负载所处的拓扑域进行亲和性调度，或者选择不存在目标工作负载的拓扑域进行反亲和性调度。

   示例中，带有app=backend的工作负载在拓扑域1中，因此，亲和app=backend工作负载在调度时，可以调度到拓扑域1中。同理，反亲和app=backend工作负载在调度时，只能调度到拓扑域2或3中。

图 5-14 工作负载亲和/反亲和示意图



配置负载亲和/反亲和调度策略

您可以通过不同的方式配置负载亲和/反亲和调度策略，将Pod调度到满足条件的节点。

通过控制台配置

本文示例中，集群内已创建后端应用的工作负载，且带有app=backend的标签，您可以通过该标签进行工作负载亲和/反亲和调度，将新创建的前端应用（标签为app=frontend）和后端应用（标签为app=backend）部署在同一节点上，即拓扑域为kubernetes.io/hostname。由于每个节点的hostname不同，因此使用kubernetes.io/hostname划分拓扑域时，每个拓扑域中仅1个节点，在工作负载亲和时可实现调度到同一节点。

步骤1 登录CCE控制台。

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

步骤3 在创建工作负载时，在“高级设置”中找到“调度策略”，选择负载亲和调度的策略类型，本示例中选择**自定义亲和策略**。创建工作负载的其余步骤详情请参见**创建工作负载**。

表 5-21 调度策略类型

参数	参数说明	示例
负载亲和	• **不配置**：不设置负载亲和策略。 • **优先多可用区部署**：该策略通过Pod自身反亲和实现，并以可用区作为拓扑域，可**优先**将工作负载的Pod调度到不同可用区的节点上。 • **强制多可用区部署**：该策略通过Pod自身反亲和实现，并以可用区作为拓扑域，可**强制**将工作负载的Pod调度到不同可用区的节点上。使用该调度策略时，如果节点数小于实例数或节点资源不足，Pod将无法全部运行。 • **自定义亲和策略**：根据Pod标签实现灵活的调度策略，支持的调度策略类型请参见表5-22。选择合适的策略类型后，可以添加相应的调度策略，参数详情请参见表5-23。	**自定义亲和策略**

步骤4 选择合适的负载亲和亲和性规则，并单击 ，添加相应的调度策略。本示例中在**工作负载亲和性 > 必须满足**的类别下添加调度策略，表示节点上必须已经运行了指定标签的工作负载才可以调度本次创建的工作负载。

表 5-22 负载亲和策略类型

策略	规则类型	说明	示例
工作负载亲和性	必须满足	即硬约束，设置必须满足的条件，对应YAML定义中的requiredDuringSchedulingIgnoredDuringExecution字段。 通过标签筛选需要亲和的Pod，如果满足筛选条件的Pod已经运行在拓扑域中的某个节点上，调度器会将本次创建的Pod**强制**调度到该拓扑域。 **说明** 添加多条亲和性规则时，即设置多个标签筛选需要亲和的Pod，则本次创建的Pod必须要同时亲和所有满足标签筛选的Pod，即所有满足标签筛选的Pod要处于同一拓扑域中才可以调度。	**必须满足**
	尽量满足	即软约束，设置尽量满足的条件，对应YAML定义中的preferredDuringSchedulingIgnoredDuringExecution字段。 通过标签筛选需要亲和的Pod，如果满足筛选条件的Pod已经运行在拓扑域中的某个节点上，调度器会将本次创建的Pod**优先**调度到该拓扑域。 **说明** 添加多条亲和性规则时，即设置多个标签筛选需要亲和的Pod，则本次创建的Pod会尽量同时亲和多个满足标签筛选的Pod。但即使所有Pod都不满足标签筛选条件，也会选择一个拓扑域进行调度。	

策略	规则类型	说明	示例
工作负载反亲和性	必须满足	即硬约束，设置必须满足的条件，对应YAML定义中的requiredDuringSchedulingIgnoredDuringExecution字段。 通过标签筛选需要反亲和的一个或多个Pod，如果满足筛选条件的Pod已经运行在拓扑域中的某个节点上，调度器**不会**将本次创建的Pod调度到该拓扑域。 **说明** 添加多条反亲和性规则时，即设置多个标签筛选需要反亲和的Pod，则本次创建的Pod必须要同时反亲和所有满足标签筛选的Pod，即所有满足标签筛选的Pod所处的拓扑域都不会被调度。	-
	尽量满足	即软约束，设置尽量满足的条件，对应YAML定义中的preferredDuringSchedulingIgnoredDuringExecution字段。 通过标签筛选需要反亲和的一个或多个Pod，如果满足筛选条件的Pod已经运行在拓扑域中的某个节点上，调度器会将本次创建的Pod**优先**调度到其他拓扑域。 **说明** 添加多条反亲和性规则时，即设置多个标签筛选需要反亲和的Pod，则本次创建的Pod会尽量同时反亲和多个满足标签筛选的Pod。但即使每个拓扑域都存在需要反亲和的Pod，也会选择一个拓扑域进行调度。	

步骤5 在右侧弹出窗口中单击“添加策略”，设置节点标签筛选规则。

表 5-23 负载亲和/反亲和调度策略设置参数说明

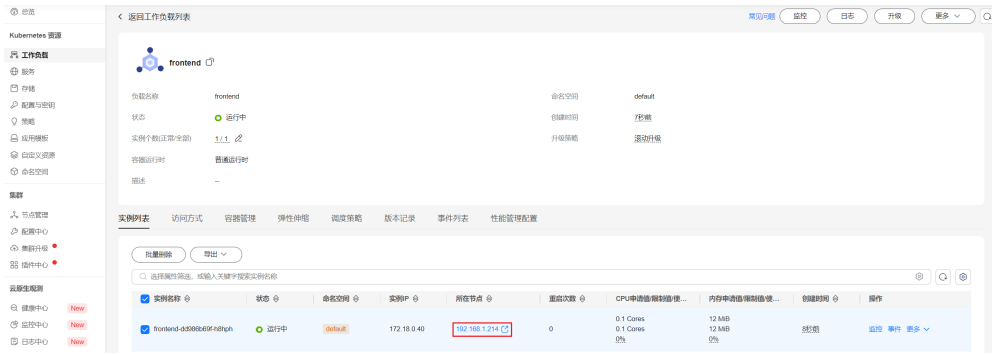
参数	参数说明	示例
权重	仅支持在“尽量满足”策略中添加。权重的取值范围为1-100，调度器在进行调度时会将该权重视为一个附加的评分项，并将其与节点的其他优先级函数评分相加。最终，调度器会将Pod调度到总分最大的节点上。	-
命名空间	指定调度策略生效的命名空间。	**default**

参数	参数说明	示例
拓扑域	拓扑域（topologyKey）通过节点的标签和标签值先圈定调度的节点范围，然后再通过标签名、操作符、标签值确定亲和/反亲和的对象，最后根据目标对象所处的拓扑域进行调度。 ● 如果标签指定为kubernetes.io/hostname，此时标签值为节点名称，则将不同名称的节点划分为不同的拓扑域，由于节点名称不可重复，此时一个拓扑域中仅包含一个节点，因此可以实现单个节点级别的负载亲和性调度。 ● 如果指定标签为kubernetes.io/os，此时标签值为节点的操作系统类型，则将不同操作系统的节点划分为不同的拓扑域，此时一个拓扑域中可能包含多个节点，因此可以将多个节点作为一个整体进行负载亲和性调度。 例如，某个拓扑域中的一个节点上运行着满足负载亲和性规则的Pod，则该拓扑域中的节点均可以被调度。	**kubernetes.io/hostname**
标签名	设置工作负载亲和/反亲和性时，填写需要匹配的工作负载标签。 该标签可以使用系统默认的标签，也可以使用自定义标签。	**app**
操作符	可以设置四种匹配关系（In、NotIn、Exists、DoesNotExist）。 ● In：亲和/反亲和对象的标签在标签值列表（values字段）中。 ● NotIn：亲和/反亲和对象的标签不在标签值列表（values字段）中。 ● Exists：亲和/反亲和对象存在指定标签名。 ● DoesNotExist：亲和/反亲和对象不存在指定标签名。	**In**
标签值	设置工作负载亲和/反亲和性时，填写工作负载标签对应的标签值。	**backend**

步骤6 调度策略添加完成后，单击“创建工作负载”。

步骤7 验证Pod全部运行在目标节点上。

1. 在集群控制台左侧导航栏中选择“工作负载”。
2. 单击工作负载名称，进入详情页面，查看实例列表，验证新建的Pod和已有的backend Pod运行在同一节点上。



----结束

通过 YAML 配置

- 工作负载亲和性

  Kubernetes支持Pod和Pod之间的亲和，例如将应用的前端和后端部署在一起，从而减少访问延迟。

  假设有个应用的后端已经创建，且带有app=backend的标签。您可以使用.spec.affinity.podAffinity字段来设置工作负载亲和性，将前端Pod（标签为app=frontend）和后端Pod（标签为app=backend）部署在一起。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - image: nginx:alpine
        name: frontend
        resources:
          requests:
            cpu: 100m
            memory: 200Mi
          limits:
            cpu: 100m
            memory: 200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:  # 设置调度策略
        podAffinity:  # 工作负载亲和性调度规则
          requiredDuringSchedulingIgnoredDuringExecution:   # 表示必须满足的调度策略
          - topologyKey: prefer    # 根据节点标签划分拓扑域，示例中prefer为自定义标签
            labelSelector:  # 根据工作负载标签选择满足条件的工作负载
              matchExpressions: # 工作负载标签匹配规则
              - key: app # 工作负载标签的键为app
                operator: In # 表示存在values列表中的值即满足规则
                values: # 工作负载标签的值列表
                - backend
          preferredDuringSchedulingIgnoredDuringExecution:    # 表示尽量满足的调度策略
          - weight: 100  # 使用尽量满足策略时可设置优先级，取值为1-100，数值越大优先级越高
            podAffinityTerm:  # 使用尽量满足策略时的亲和项
```



```
          topologyKey: topology.kubernetes.io/zone   # 根据节点标签划分拓扑域，以节点的可用区为粒
度
          labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - backend
```

上述示例中的工作负载调度时，**必须满足**规则会根据prefer标签划分节点拓扑域，如果当拓扑域中某个节点运行着后端Pod（标签为app=backend），即使该拓扑域中并非所有节点均运行了后端Pod，前端Pod（标签为app=frontend）同样会部署在此拓扑域中。而**尽量满足**规则根据topology.kubernetes.io/zone划分拓扑域，以节点的可用区为粒度进行调度，表示尽量将前后端部署至同一可用区的节点。

说明

对于工作负载亲和来说，使用requiredDuringSchedulingIgnoredDuringExecution和preferredDuringSchedulingIgnoredDuringExecution规则时， topologyKey字段不允许为空。

topologyKey字段用于划分拓扑域，当节点上存在topologyKey字段指定的标签，且键、值均相同时，这些节点会被认为属于同一拓扑域，然后调度器会根据工作负载的标签选择需要调度的拓扑域。一个拓扑域中可能包含多个节点，当拓扑域中的一个节点上运行了满足标签选择规则的工作负载时，则该拓扑域中的节点均可以被调度。

例如，当topologyKey的标签为topology.kubernetes.io/zone时，表示以节点的可用区作为拓扑域，工作负载在部署时会以可用区为粒度进行调度。

- 工作负载反亲和性

  在某些情况下，需要将Pod分开部署，例如Pod之间部署在一起会影响性能的情况。

  假设有个应用的前端已经创建，且带有app=frontend的标签。您可以使用.spec.affinity.podAntiAffinity字段来设置工作负载反亲和性，将各个Pod部署在不同的节点，且优先多可用区。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:   frontend
  labels:
    app:  frontend
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 5
  template:
    metadata:
      labels:
        app:  frontend
    spec:
      containers:
      - image:  nginx:alpine
        name:  frontend
        resources:
          requests:
            cpu:  100m
            memory:  200Mi
          limits:
            cpu:  100m
            memory:  200Mi
      imagePullSecrets:
      - name: default-secret
      affinity:
        podAntiAffinity:  # 工作负载反亲和性调度规则
```



```
      requiredDuringSchedulingIgnoredDuringExecution:   # 表示必须满足的调度策略
      - topologyKey: kubernetes.io/hostname    # 根据节点标签划分拓扑域
        labelSelector:    # Pod标签匹配规则
          matchExpressions:  # 工作负载标签的键为app
          - key: app  # 工作负载标签的键为app
            operator: In  # 表示存在values列表中的值即满足规则
            values:  # 工作负载标签的值列表
            - frontend
      preferredDuringSchedulingIgnoredDuringExecution:    # 表示尽量满足的调度策略
      - weight: 100  # 使用尽量满足策略时可设置优先级，取值为1-100，数值越大优先级越高
        podAffinityTerm:  # 使用尽量满足策略时的亲和项
          topologyKey: topology.kubernetes.io/zone   # 根据节点标签划分拓扑域
          labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - frontend
```

以上示例中定义了反亲和规则，**必须满足**的规则表示根据kubernetes.io/hostname标签划分节点拓扑域。由于拥有kubernetes.io/hostname标签的节点中，每个节点的标签值均不同，因此一个拓扑域中只有一个节点。当一个拓扑域中（此处为一个节点）已经存在frontend标签的Pod时，该拓扑域不会被继续调度具有相同标签的Pod。而**尽量满足**规则根据topology.kubernetes.io/zone划分拓扑域，以节点的可用区为粒度进行调度，表示尽量将Pod分布至不同可用区的节点。

说明

对于工作负载反亲和来说，使用requiredDuringSchedulingIgnoredDuringExecution规则时， Kubernetes默认的准入控制器 LimitPodHardAntiAffinityTopology要求topologyKey字段只能是kubernetes.io/hostname。如果您希望使用其他定制拓扑逻辑，可以更改或者禁用该准入控制器。

5.5 登录容器实例

操作场景

如果在使用容器的过程中遇到非预期的问题，您可登录容器进行调试。

约束与限制

同一用户在使用CloudShell组件连接CCE集群或容器时，限制同时打开的实例上限数量为15个。

使用 CloudShell 登录容器

须知

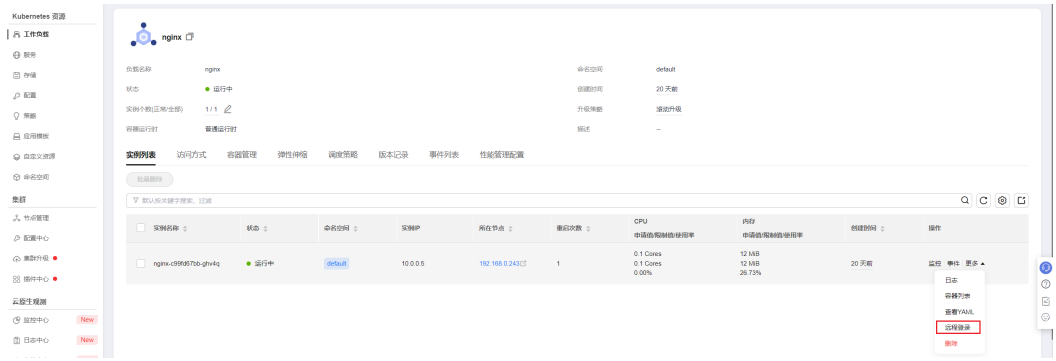
- CloudShell基于VPCEP实现，在CloudShell中使用kubectl访问集群需要在集群控制节点的安全组（安全组名称：集群名称-cce-control-随机数）中放通5443端口。5443端口默认对所有网段放通，如果您对安全组做过加固，当出现在CloudShell中无法访问集群时，请检查5443端口是否放通了网段。
- 当前仅北京一、北京四、上海一、上海二、广州、贵阳一和乌兰察布一支持使用CloudShell登录容器。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧选择“工作负载”，单击目标工作负载名称，查看工作负载的实例列表。

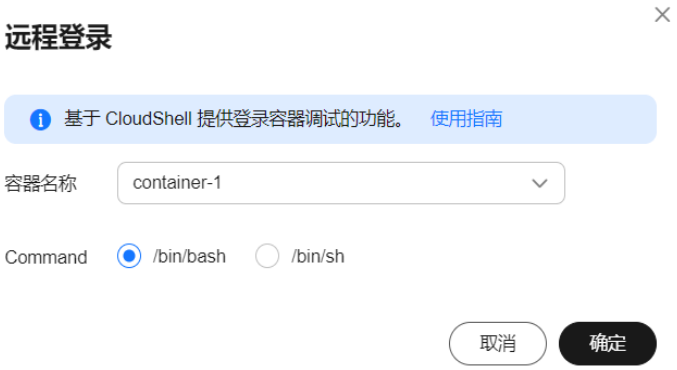
步骤3 单击目标实例操作列中的“更多 > 远程登录”。

图 5-15 登录容器



步骤4 在弹出窗口中选择要登录的容器以及命令，然后单击“确定”。

图 5-16 选择登录的容器与命令

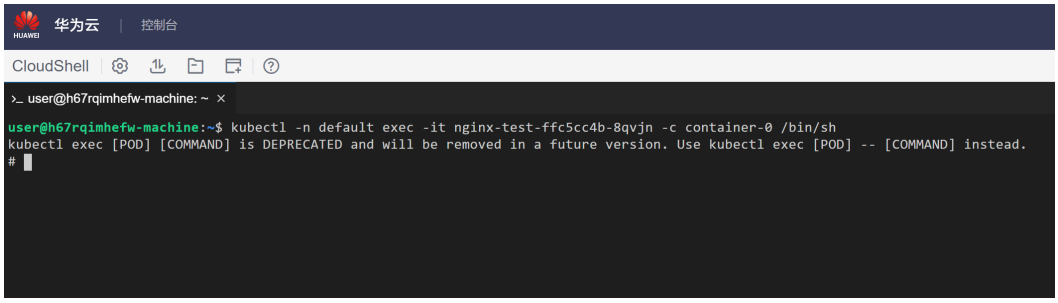


步骤5 页面会自动跳转到CloudShell，并初始化启动kubectl，然后自动执行kubectl exec命令登录到容器。

说明

请等待kubectl exec 命令自动执行后再操作，此命令出现需要一段时间 5-10秒。

图 5-17 CloudShell 页面



----结束

使用 kubectl 命令登录容器

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 执行以下命令，查看已创建的Pod。

```
kubectl get pod
```

示例输出如下：

```
NAME                     READY   STATUS    RESTARTS      AGE
nginx-59d89cb66f-mhljr   1/1     Running   0             11m
```

步骤3 查询该Pod中的容器名称。

```
kubectl get po nginx-59d89cb66f-mhljr -o jsonpath='{range .spec.containers[*]}{.name}{end}{"\n"}'
```

示例输出如下：

```
container-1
```

步骤4 执行以下命令，登录到nginx-59d89cb66f-mhljr这个Pod中名为container-1的容器。

```
kubectl exec -it nginx-59d89cb66f-mhljr -c container-1 -- /bin/sh
```

步骤5 如需退出容器，可执行**exit**命令。

----结束

5.6 管理工作负载

操作场景

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

表 5-24 工作负载/任务管理

操作	描述
监控	可以通过CCE控制台查看工作负载和容器组的CPU和内存占用情况，以确定需要的资源规格。
日志	可查看工作负载的日志信息。
升级	可以通过更换镜像或镜像版本实现无状态工作负载、有状态工作负载、守护进程集的快速升级，业务无中断。
编辑YAML	可通过在线YAML编辑窗对无状态工作负载、有状态工作负载、守护进程集、定时任务和容器组的YAML文件进行修改和下载。普通任务的YAML文件仅支持查看、复制和下载。 **说明** 如果对已有的定时任务（CronJob）进行修改，修改之后运行的新Pod将使用新的配置，而已经运行的Pod将继续运行不会发生任何变化。
回退	无状态工作负载可以进行回退操作，仅无状态工作负载可用。
重新部署	工作负载可以进行重新部署操作，重新部署后将重启负载下的全部容器组Pod。
关闭/开启升级	无状态工作负载可以进行关闭/开启升级操作，仅无状态工作负载可用。

操作	描述
标签管理	标签是以key/value键值对的形式附加在工作负载上的。添加标签后，可通过标签对工作负载进行管理和选择。任务或定时任务无法使用标签管理功能。
删除	若工作负载无需再使用，您可以将工作负载或任务删除。工作负载或任务删除后，将无法恢复，请谨慎操作。
事件	查看具体实例的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间。
停止/启动	停止/启动一个定时任务，该功能仅定时任务可用。

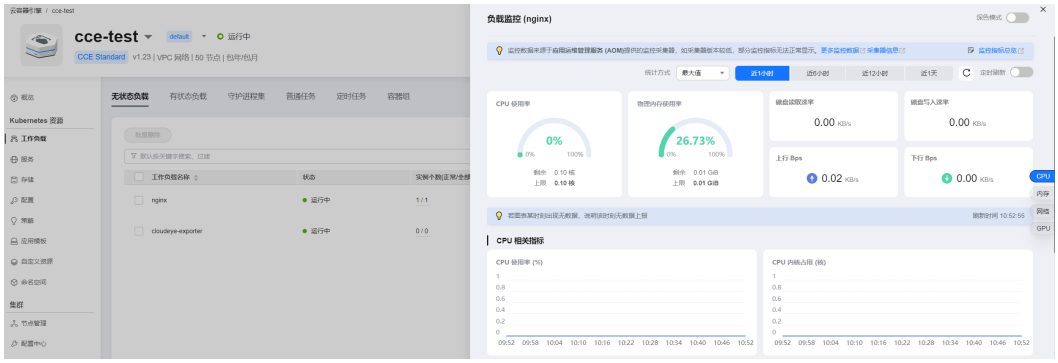
监控

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

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

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

图 5-18 查看无状态工作负载监控



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

----结束

日志

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

须知

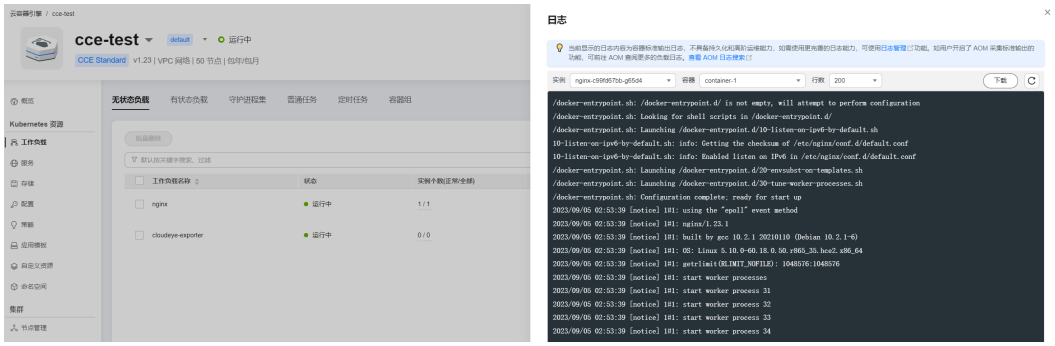
查看日志前请将浏览器与后端服务器时间调成一致。

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

步骤2 选择“无状态负载”页签，单击工作负载后的“日志”。

在弹出的“日志”窗口中可以查看容器日志信息。

图 5-19 查看无状态工作负载日志



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**日志管理**功能。如工作负载开启了AOM采集标准输出的功能（默认开启），可前往AOM查阅更多的负载日志，详情请参见**通过ICAgent采集容器日志（不推荐）**。

----结束

升级

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

本文以无状态工作负载为例说明如何进行升级。

若需要更换镜像或镜像版本，您需要提前将镜像上传到容器镜像服务，上传方法请参见**通过Docker客户端上传镜像**。

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

步骤2 选择“无状态负载”页签，单击待升级工作负载后的“升级”。

说明

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

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

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

----结束

编辑 YAML

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

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

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

步骤3 单击“确定”，完成修改。

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

----结束

回退（仅无状态工作负载可用）

所有无状态工作负载的发布历史记录都保留在系统中，您可以回退到指定的版本。

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

步骤2 选择“无状态负载”页签，单击待回退工作负载后的“更多 > 回退”。

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

图 5-20 回退工作负载版本

----结束

重新部署

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

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

步骤2 选择“无状态负载”页签，单击工作负载后的“更多 > 重新部署”。

步骤3 在弹出的提示框中单击“是”，即可完成工作负载的重新部署。

----结束

关闭/开启升级（仅无状态工作负载可用）

无状态工作负载可以进行“关闭/开启升级”操作。

- 关闭升级后，对负载进行的升级操作可以正常下发，但不会被应用到实例。
  如果您正在滚动升级的过程中，滚动升级会在关闭升级命令下发后停止，出现新旧实例共存的状态。

- 开启升级后，负载可以正常升级和回退，负载下的实例会与负载当前的最新信息进行一次同步，如果有不一致的，则会自动按照负载的最新信息进行升级。

须知

工作负载状态在关闭升级时无法执行回退操作。

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

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

步骤3 在弹出的信息提示框中，单击“是”。

----结束

标签管理

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

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

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

步骤3 单击+，输入键和值后单击“确定”。

图 5-21 标签管理

说明

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

----结束

删除工作负载/任务

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

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

步骤2 单击待删除工作负载后的“更多 > 删除”，删除工作负载。

请仔细阅读系统提示，删除操作无法恢复，请谨慎操作。

步骤3 单击“是”。

说明

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

----结束

事件

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

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

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

说明

事件保存时间为1小时，1小时后自动清除数据。

----结束

5.7 Pod 安全配置

5.7.1 PodSecurityPolicy 配置

Pod安全策略（Pod Security Policy）是集群级别的资源，它能够控制Pod规约中与安全性相关的各个方面。 **PodSecurityPolicy**对象定义了一组Pod运行时必须遵循的条件及相关字段的默认值，只有Pod满足这些条件才会被系统接受。

v1.17.17版本的集群默认启用Pod安全策略准入控制组件，并创建名为**psp-global**的全局默认安全策略，您可根据自身业务需要修改全局策略（请勿直接删除默认策略），也可新建自己的Pod安全策略并绑定RBAC配置。

说明

- 除全局默认安全策略外，系统为kube-system命名空间下的系统组件配置了独立的Pod安全策略，修改psp-global配置不影响kube-system下Pod创建。
- PodSecurityPolicy在Kubernetes v1.21版本中被弃用，并在Kubernetes v1.25中被移除。您可以Pod安全性准入控制器（Pod Security Admission）作为PodSecurityPolicy的替代，详情请参见**Pod Security Admission配置**。

修改全局默认 Pod 安全策略

修改全局默认Pod安全策略前，请确保已创建CCE集群，并且通过kubectl连接集群成功。

步骤1 执行如下命令：

kubectl edit psp psp-global

步骤2 修改所需的参数，如表5-25。

表 5-25 Pod 安全策略配置

配置项	描述
privileged	启动特权容器。
hostPID hostIPC	使用主机命名空间。
hostNetwork hostPorts	使用主机网络和端口。
volumes	允许使用的挂载卷类型。
allowedHostPaths	允许hostPath类型挂载卷在主机上挂载的路径，通过pathPrefix字段声明允许挂载的主机路径前缀组。
allowedFlexVolumes	允许使用的指定FlexVolume驱动。
fsGroup	配置Pod中挂载卷使用的辅组ID。
readOnlyRootFilesystem	约束启动Pod使用只读的root文件系统。
runAsUser runAsGroup supplementalGroups	指定Pod中容器启动的用户ID以及主组和辅组ID。
allowPrivilegeEscalation defaultAllowPrivilegeEscalation	约束Pod中是否允许配置allowPrivilegeEscalation=true，该配置会控制Setuid的使用，同时控制程序是否可以使用额外的特权系统调用。
defaultAddCapabilities requiredDropCapabilities allowedCapabilities	控制Pod中使用的Linux Capabilities。
seLinux	控制Pod使用seLinux配置。
allowedProcMountTypes	控制Pod允许使用的ProcMountTypes。
annotations	配置Pod中容器使用的AppArmor或Seccomp。
forbiddenSysctls allowedUnsafeSysctls	控制Pod中容器使用的Sysctl配置。

----结束

Pod 安全策略开放非安全系统配置示例

节点池管理中可以为相应的节点池配置allowed-unsafe-sysctls，CCE从**1.17.17**集群版本开始，需要在Pod安全策略的allowedUnsafeSysctls字段中增加相应的配置才能生效，配置详情请参考**表5-25**。

除修改全局Pod安全策略外，也可增加新的Pod安全策略，如开放net.core.somaxconn非安全系统配置，新增Pod安全策略示例参考如下：

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  name: sysctl-psp
spec:
  allowedUnsafeSysctls:
  - net.core.somaxconn
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  fsGroup:
    rule: RunAsAny
  hostIPC: true
  hostNetwork: true
  hostPID: true
  hostPorts:
  - max: 65535
    min: 0
  privileged: true
  runAsGroup:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - '*'
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sysctl-psp
rules:
  - apiGroups:
      - "*"
    resources:
      - podsecuritypolicies
    resourceNames:
      - sysctl-psp
    verbs:
      - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sysctl-psp
roleRef:
  kind: ClusterRole
  name: sysctl-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

恢复原始 Pod 安全策略

如果您已经修改默认Pod安全策略后，想恢复原始Pod安全策略，请执行以下操作。

步骤1 创建一个名为policy.yaml的描述文件。其中，policy.yaml为自定义名称，您可以随意命名。

vi policy.yaml

描述文件内容如下。

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: psp-global
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
    - '*'
  volumes:
    - '*'
  hostNetwork: true
  hostPorts:
    - min: 0
      max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-global
rules:
  - apiGroups:
      - "*"
    resources:
      - podsecuritypolicies
    resourceNames:
      - psp-global
    verbs:
      - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp-global
roleRef:
  kind: ClusterRole
  name: psp-global
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```


步骤2 执行如下命令：

kubectl apply -f policy.yaml

----结束

5.7.2 Pod Security Admission 配置

在使用**Pod Security Admission**前，需要先了解Kubernetes的**Pod安全性标准（Security Standards）**。Pod安全性标准（Security Standards）为 Pod 定义了不同的安全性策略级别。这些标准能够让你以一种清晰、一致的方式定义如何限制Pod行为。而Pod Security Admission则是这些安全性标准的控制器，用于在创建Pod时执行定义好的安全限制。

Pod安全性标准定义了三种安全性策略级别：

表 5-26 Pod 安全性策略级别

策略级别（level）	描述
privileged	不受限制，通常适用于特权较高、受信任的用户所管理的系统级或基础设施级负载，例如CNI、存储驱动等。
baseline	限制较弱但防止已知的特权提升（Privilege Escalation），通常适用于部署常用的非关键性应用负载，该策略将禁止使用hostNetwork、hostPID等能力。
restricted	严格限制，遵循Pod防护的最佳实践。

Pod Security Admission配置是命名空间级别的，控制器将会对该命名空间下Pod或容器中的安全上下文（Security Context）以及其他参数进行限制。其中，privileged策略将不会对Pod和Container配置中的securityContext字段有任何校验，而Baseline和Restricted则会对securityContext字段有不同的取值要求，具体规范请参见**Pod安全性标准（Security Standards）**。

关于如何在Pod或容器中设置Security Context，请参见**为Pod或容器配置Security Context**。

Pod Security Admission 标签

Kubernetes为Pod Security Admission定义了三种标签，如**表5-27**，您可以在某个命名空间中设置这些标签来定义需要使用的Pod安全性标准级别，但请勿在kube-system等系统命名空间修改Pod安全性标准级别，否则可能导致系统命名空间下Pod故障。

表 5-27 Pod Security Admission 标签

隔离模式（mode）	生效对象	描述
enforce	Pod	违反指定策略会导致Pod无法创建。

隔离模式（mode）	生效对象	描述
audit	工作负载（例如Deployment、Job等）	违反指定策略会在审计日志（audit log）中添加新的审计事件，Pod可以被创建。
warn	工作负载（例如Deployment、Job等）	违反指定策略会返回用户可见的告警信息，Pod可以被创建。

说明

Pod通常是通过创建Deployment或Job这类工作负载对象来间接创建的。在使用Pod Security Admission时，audit或warn模式的隔离都将在工作负载级别生效，而enforce模式并不会应用到工作负载，仅在Pod上生效。

使用命名空间标签进行 Pod Security Admission 配置

您可以在不同的隔离模式中应用不同的策略，由于Pod安全性准入能力是在命名空间（Namespace）级别实现的，因此假设某个Namespace配置如下：

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-baseline-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.25
    pod-security.kubernetes.io/audit: baseline
    pod-security.kubernetes.io/audit-version: v1.25
    pod-security.kubernetes.io/warn: restricted
    pod-security.kubernetes.io/warn-version: v1.25

    # 标签有以下两种格式：
    # pod-security.kubernetes.io/<MODE>: <LEVEL>
    # pod-security.kubernetes.io/<MODE>-version: <VERSION>
    # audit和warn模式的作用主要在于提供相应信息供用户排查负载违反了哪些安全行为
```

命名空间的标签用来表示不同的模式所应用的安全策略级别，存在以下两种格式：

- pod-security.kubernetes.io/<MODE>: <LEVEL>
  - <MODE>：必须是enforce、audit或warn之一，关于标签详情请参见**表5-27**。
  - <LEVEL>：必须是privileged、baseline或restricted之一，关于安全性策略级别详情请参见**表5-26**。
- pod-security.kubernetes.io/<MODE>-version: <VERSION>

  该标签为可选，可以将安全性策略锁定到Kubernetes版本号。
  - <MODE>：必须是enforce、audit或warn之一，关于标签详情请参见**表5-27**。
  - <VERSION>：Kubernetes版本号。例如 v1.25，也可以使用latest。

若在上述Namespace中部署Pod，则会有以下安全性限制：

1. 设置了enforce隔离模式对应的策略为privileged，将会跳过enforce阶段的校验。

2. 设置了audit隔离模式对应的策略为baseline，将会校验baseline策略相关限制，即如果Pod或Container违反了该策略，审计日志中将添加相应事件。
3. 设置了warn隔离模式对应的策略为restricted，将会校验restricted策略相关限制，即如果Pod或Container违反了该策略，用户将会在创建Pod时收到告警信息。

从 PodSecurityPolicy 迁移到 Pod Security Admission

如您在1.25之前版本的集群中使用了PodSecurityPolicy，且需要在1.25及以后版本集群中继续使用Pod Security Admission来替代PodSecurityPolicy的用户，请参见**从PodSecurityPolicy迁移到内置的Pod Security Admission**。

须知

1. 由于Pod Security Admission仅支持三种隔离模式，因此灵活性相比于PodSecurityPolicy较差，部分场景下需要用户自行定义验证准入Webhook来实施更精准的策略。
2. 由于PodSecurityPolicy具有变更能力，而Pod Security Admission并不具备该能力，因此之前依赖该能力的用户需要自行定义变更准入Webhook或修改Pod中的securityContext字段。
3. PodSecurityPolicy允许为不同的服务账号（Service Account）绑定不同策略（Kubernetes社区不建议使用该能力）。如果您有使用该能力的诉求，在迁移至Pod Security Admission后，需要自行定义第三方Webhook。
4. 请勿将Pod Security Admission能力应用于kube-system、kube-public和kube-node-lease等一些CCE组件部署的Namespace中，否则会导致CCE组件、插件功能异常。

参考文档

- **Pod安全性准入**
- **从PodSecurityPolicy映射到Pod安全性标准**
- **使用命名空间标签来实施Pod安全性标准**
- **通过配置内置准入控制器实施Pod安全标准**

6 调度

6.1 调度概述

CCE支持不同类型的资源调度及任务调度等，可提升应用的性能和集群整体资源的利用率。本文介绍CPU资源调度、GPU/NPU异构资源调度、Volcano调度的主要功能。

CPU 调度

CCE提供CPU管理策略为应用分配完整的CPU物理核，提升应用性能，减少应用的调度延迟。

功能	描述	参考文档
CPU管理策略	当节点上运行了很多 CPU 密集的 Pod 时，工作负载可能会迁移到不同的 CPU 核。许多应用对这种迁移不敏感，因此无需任何干预即可正常工作。有些应用对CPU敏感，对于CPU敏感型应用，您可以利用Kubernetes中提供的CPU管理策略为应用分配独占核，提升应用性能，减少应用的调度延迟。	CPU管理策略
增强型CPU管理策略	增强型CPU管理策略（enhanced-static），是在兼容静态绑核CPU管理策略的基础上，新增一种符合某些资源特征的Burstable Pod（CPU的Request和Limit值都是正整数）优先使用某些CPU的能力，以减少应用在多个CPU间频繁切换带来的影响。	增强型CPU管理策略

GPU 调度

CCE为集群中的GPU异构资源提供调度能力，支持在容器中使用GPU显卡。

功能	描述	参考文档
Kubernetes默认GPU调度	Kubernetes默认GPU调度可以指定Pod申请GPU的数量，支持申请设置为小于1的数量，实现多个Pod共享使用GPU。	使用Kubernetes默认GPU调度
GPU虚拟化	GPU虚拟化能够动态对GPU设备显存与算力进行划分，单个GPU卡最多虚拟化成20个GPU虚拟设备。相对于静态分配来说，虚拟化的方案更加灵活，最大程度保证业务稳定的前提下，可以完全由用户自己定义使用的GPU量，提高GPU利用率。	GPU虚拟化

NPU 调度

CCE为集群中的NPU异构资源提供调度能力，实现快速高效地处理推理和图像识别等工作。

功能	描述	参考文档
NPU调度	NPU调度可以指定Pod申请NPU的数量，为工作负载提供NPU资源。	NPU调度

Volcano 调度

Volcano是一个基于Kubernetes的批处理平台，提供了机器学习、深度学习、生物信息学、基因组学及其他大数据应用所需要而Kubernetes当前缺失的一系列特性，提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力。

功能	描述	参考文档
资源利用率优化调度	针对计算资源进行优化的调度策略，可以有效减少各节点资源碎片，最大化地提高计算资源的利用率。	资源利用率优化调度
业务优先级保障调度	根据业务的重要性和优先级，设置自定义的策略对业务占用的资源进行调度，确保关键业务的资源优先级得到保障。	业务优先级保障调度
AI任务性能增强调度	根据AI任务的工作性质、资源的使用情况，设置对应的调度策略，可以增强集群业务的吞吐量，提高业务运行性能。	AI任务性能增强调度
NUMA亲和性调度	Volcano可解决调度程序NUMA拓扑感知的限制，实现以下目标： • 避免将Pod调度到NUMA拓扑不匹配的节点。 • 将Pod调度到NUMA拓扑的最佳节点。	NUMA亲和性调度

云原生混部

云原生混部解决方案围绕Volcano和Kubernetes生态，帮助用户提升资源利用率，实现降本增效。

功能	描述	参考文档
动态资源超卖	根据在线作业和离线作业类型，通过Volcano调度将集群中申请而未使用的资源（即申请量与使用量的差值）利用起来，实现资源超卖和混合部署，提升集群资源利用率。	动态资源超卖
CPU Burst弹性限流	提供一种可以短暂突破CPU Limit值的弹性限流机制，以降低业务长尾响应时间，可以有效提升时延敏感型业务的服务质量。	CPU Burst弹性限流
出口网络带宽保障	平衡在线业务与离线业务对出口网络带宽的使用，保证在线业务有足够的网络带宽。	出口网络带宽保障

6.2 CPU 调度

6.2.1 CPU 管理策略

使用场景

默认情况下，kubelet使用**CFS 配额**来执行Pod的CPU约束。 当节点上运行了很多CPU密集的Pod时，工作负载可能会迁移到不同的CPU核， 这取决于调度时Pod是否被扼制，以及哪些CPU核是可用的。许多应用对这种迁移不敏感，因此无需任何干预即可正常工作。有些应用对CPU敏感，CPU敏感型应用有如下特点。

- 对CPU throttling 敏感
- 对上下文切换敏感
- 对处理器缓存未命中敏感
- 对跨Socket内存访问敏感
- 期望运行在同一物理CPU的超线程

如果您的应用有以上其中一个特点，可以利用Kubernetes中提供的**CPU管理策略**为应用分配独占的CPU核（即CPU绑核），提升应用性能，减少应用的调度延迟。CPU manager会优先在一个Socket上分配资源，也会优先分配完整的物理核，避免一些干扰。

CPU管理策略通过kubelet参数*--cpu-manager-policy*来指定。Kubernetes默认支持两种策略：

- none：默认策略，显式地启用现有的默认CPU亲和方案，不提供操作系统调度器默认行为之外的亲和性策略。
- static：针对CPU申请值设置为整数的**Guaranteed Pods**，它允许该类Pod中的容器访问节点上的独占CPU资源（绑核）。

约束与限制

弹性云服务器-物理机节点不支持使用CPU管理策略。

为集群开启 CPU 管理策略（DefaultPool 中的节点）

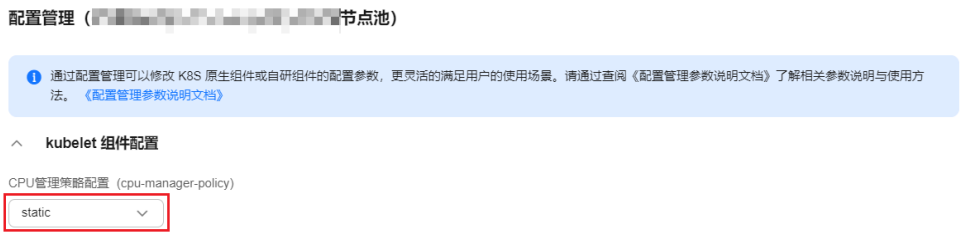
在创建集群时的“高级配置”中可以选择开启CPU管理策略。

- 开启：对应Kubernetes策略中的static，即使用CPU绑核。
- 关闭：对应Kubernetes策略中的none，即不使用CPU绑核。

为自定义节点池开启 CPU 管理策略

您可以在自定义节点池中单独配置CPU管理策略，配置后会自动修改节点池中节点的上kubelet参数 --cpu-manager-policy。

1. 登录CCE控制台，单击集群名称进入集群。
2. 在左侧选择“节点管理”，在右侧选择“节点池”页签，单击节点池名称后的“更多 > 配置管理”。
3. 在侧边栏滑出的“配置管理”窗口中，修改kubelet组件的CPU管理策略配置（cpu-manager-policy）参数值，选择**static**。



4. 单击“确定”，完成配置操作。

为 Pod 设置独占 CPU

Pod设置独占CPU（即CPU绑核）有如下几点要求：

- 节点上开启静态（static）CPU管理策略，具体方法请参见**为集群开启CPU管理策略（DefaultPool中的节点）**。
- Pod的定义里都要设置requests和limits参数，requests和limits必须为整数，且数值一致。
- 如果有init container需要设置独占CPU，init container的requests参数建议与业务容器设置的requests参数一致（避免业务容器未继承init container的CPU分配结果，导致CPU manager多预留一部分CPU）。更多信息请参见**App Containers can't inherit Init Containers CPUs - CPU Manager Static Policy**。

在使用时您可以利用**节点亲和调度**将如上配置的Pod调度到开启静态（static）CPU管理策略的节点上，这样就能够达到独占CPU的效果。

设置独占CPU的YAML示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          resources:
            requests:
              cpu: 2          # 必须为整数，且需要与limits中一致
              memory: 2048Mi
            limits:
              cpu: 2          # 必须为整数，且需要与requests中一致
              memory: 2048Mi
      imagePullSecrets:
        - name: default-secret
```

验证

以8U16G节点为例，并提前在节点上部署一个CPU request为2，limit为2的工作负载。

登录到工作负载运行的节点，查看/var/lib/kubelet/cpu_manager_state输出内容。

```
cat /var/lib/kubelet/cpu_manager_state
```

回显如下：

```
{"policyName":"static","defaultCpuSet":"0-1,4-7","entries":{"de14506d-0408-411f-bbb9-822866b58ae2":{"container-1":"2-3"}},"checksum":3744493798}
```

- policyName字段值为static代表策略设置成功。
- 2-3代表该Pod中容器可以使用的CPU集合。

6.2.2 增强型 CPU 管理策略

在Kubernetes默认提供的**CPU管理策略**中有none和static两种：

- none： 默认不开启CPU管理策略，表示现有的调度行为。
- static：开启静态绑核的CPU管理策略，允许为节点上具有某些资源特征的 Pod（Guaranteed pod）赋予增强的 CPU 亲和性和独占性。

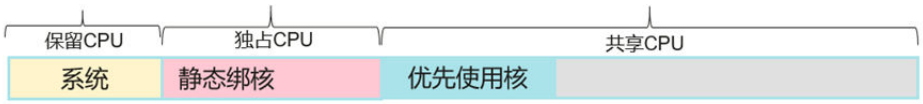
增强型CPU管理策略（enhanced-static），是在兼容静态绑核CPU管理策略的基础上，新增一种符合某些资源特征的Burstable Pod（要求CPU的requests和limits参数值都是正整数）优先使用某些CPU的能力，以减少应用在多个CPU间频繁切换带来的影响。能够使用优先使用CPU的Burstable Pod举例如下：

```
...
spec:
  containers:
```



```
- name: nginx
  image: nginx
  resources:
    limits:
      memory: "300Mi"
      cpu: "2"
    requests:
      memory: "200Mi"
      cpu: "1"
```

该特性是基于Huawei Cloud EulerOS 2.0内核中优化了CPU调度能力实现的。在Pod容器优先使用的CPU利用率超过85%时，会自动分配到其他利用率较低的CPU上，进而保障了应用的响应能力。



说明

- 开启增强型CPU管理策略时，应用性能优于不开启CPU管理策略（none），但弱于静态CPU管理策略（static）。
- 应用分配的优先使用的CPU并不会被独占，仍处于共享的CPU池中。因此在该Pod处于业务波谷时，节点上其他Pod可使用该部分CPU资源。

约束与限制

使用该特性，需同时满足以下条件：

- 集群版本为v1.23及以上。
- 节点操作系统为Huawei Cloud EulerOS 2.0。
- 弹性云服务器-物理机节点不支持使用CPU管理策略。

操作步骤

步骤1 登录CCE控制台。

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

步骤3 选择一个操作系统为Huawei Cloud EulerOS 2.0的节点池，单击节点池名称后的“配置管理”。

步骤4 在侧边栏滑出的“配置管理”窗口中，修改kubelet组件的CPU管理策略配置（cpu-manager-policy）参数值，选择**enhanced-static**。

图 6-1 CPU 管理策略配置

步骤5 单击“确定”，完成配置操作。

----结束

验证

以8U32G节点为例，并提前在集群中部署一个CPU request为1，limit为2的工作负载。

步骤1 登录到节点池中的一个节点，查看/var/lib/kubelet/cpu_manager_state输出内容。

```
cat /var/lib/kubelet/cpu_manager_state
```

回显如下：

```
{"policyName":"enhanced-static","defaultCpuSet":"0,2-7","entries":{"6739f6f2-ebe5-48ae-945a-986d5d8919b9":{"container-1":"0-7,10001"}},"checksum":1638128523}
```

- policyName字段值为enhanced-static代表策略设置成功。
- 优先使用CPU号将10000作为基数，本例中10001即代表容器使用的亲和CPU号为1，0-7代表该Pod中容器可以使用的CPU集合。

步骤2 查看容器的cpuset.preferred_cpus的cgroup设置，输出内容即为优先使用的CPU号。

```
cat /sys/fs/cgroup/cpuset/kubepods/burstable/pod{pod uid}/{容器id}/cpuset.preferred_cpus
```

- *{pod uid}*为Pod UID，可在已通过kubectl连接集群的机器上使用以下命令获取：

  ```
  kubectl get po {pod name} -n {namespace} -ojsonpath='{.metadata.uid}{"\n"}'
  ```

  命令中的*{pod name}*和*{namespace}*是Pod名称及其所在的命名空间。
- *{容器id}*需要是完整的容器ID，可在容器运行的节点上通过以下命令获取：

  docker节点池：命令中的*{pod name}*是Pod名称。

  ```
  docker ps --no-trunc | grep {pod name} | grep -v cce-pause | awk '{print $1}'
  ```

  containerd节点池：命令中的*{pod name}*是Pod名称，*{pod id}*是Pod的ID，*{container name}*是容器名称。

  ```
  # 获取Pod ID
  crictl pods | grep {pod name} | awk '{print $1}'
  # 获取完整容器ID
  crictl ps --no-trunc | grep {pod id} | grep {container name} | awk '{print $1}'
  ```

完整示例如下：

```
cat /sys/fs/cgroup/cpuset/kubepods/burstable/pod6739f6f2-ebe5-48ae-945a-986d5d8919b9/5ba5603434b95fd22d36fba6a5f1c44eba83c18c2e1de9b52ac9b52e93547a13/cpuset.preferred_cpus
```


回显如下，表示优先使用1号CPU。

```
1
```

----结束

6.3 GPU 调度

6.3.1 使用 Kubernetes 默认 GPU 调度

CCE支持在容器中使用GPU资源。

前提条件

- 创建GPU类型节点，具体请参见**创建节点**。
- 集群中需要安装GPU插件，且安装时注意要选择节点上GPU型号对应的驱动，具体请参见**CCE AI套件（NVIDIA GPU）**。
- 在v1.27及以下的集群中使用默认GPU调度能力时，GPU插件会把驱动的目录挂载到/usr/local/nvidia/lib64，在容器中使用GPU资源需要将/usr/local/nvidia/lib64追加到LD_LIBRARY_PATH环境变量中。v1.28及以上的集群中则无需执行此步骤。

  通常可以通过如下三种方式追加环境变量。

  - 制作镜像的Dockerfile中配置LD_LIBRARY_PATH。（推荐）
    ```
    ENV LD_LIBRARY_PATH /usr/local/nvidia/lib64:$LD_LIBRARY_PATH
    ```
  - 镜像的启动命令中配置LD_LIBRARY_PATH。
    ```
    /bin/bash -c "export LD_LIBRARY_PATH=/usr/local/nvidia/lib64:$LD_LIBRARY_PATH && ..."
    ```
  - 创建工作负载时定义LD_LIBRARY_PATH环境变量（需确保容器内未配置该变量，不然会被覆盖）。
    ```
    ...
            env:
              - name: LD_LIBRARY_PATH
                value: /usr/local/nvidia/lib64
    ...
    ```

使用 GPU

创建工作负载申请GPU资源，可按如下方法配置，指定显卡的数量。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-test
  template:
    metadata:
      labels:
        app: gpu-test
    spec:
      containers:
      - image: nginx:perl
        name: container-0
        resources:
```

```
      requests:
        cpu: 250m
        memory: 512Mi
        nvidia.com/gpu: 1   # 申请GPU的数量
      limits:
        cpu: 250m
        memory: 512Mi
        nvidia.com/gpu: 1   # GPU数量的使用上限
  imagePullSecrets:
  - name: default-secret
```

通过**nvidia.com/gpu**指定申请GPU的数量，支持申请设置为小于1的数量，比如**nvidia.com/gpu: 0.5**，这样可以多个Pod共享使用GPU。GPU数量小于1时，不支持跨GPU分配，如0.5 GPU只会分配到一张卡上。

说明

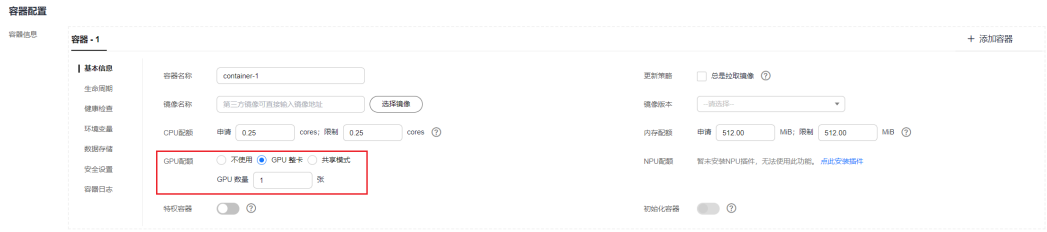
使用**nvidia.com/gpu**参数指定GPU数量时，requests和limits值需要保持一致。

指定**nvidia.com/gpu**后，在调度时不会将负载调度到没有GPU的节点。如果缺乏GPU资源，会报类似如下的Kubernetes事件。

- 0/2 nodes are available: 2 Insufficient nvidia.com/gpu.
- 0/4 nodes are available: 1 InsufficientResourceOnSingleGPU, 3 Insufficient nvidia.com/gpu.

在CCE控制台使用GPU资源，只需在创建工作负载时，选择使用的GPU配额即可。

图 6-2 使用 GPU



GPU 节点标签

创建GPU节点后，CCE会给节点打上对应标签，如下所示，不同类型的GPU节点有不同标签。

```
$ kubectl get node -L accelerator
NAME          STATUS   ROLES    AGE     VERSION                           ACCELERATOR
10.100.2.179  Ready    <none>   8m43s   v1.19.10-r0-CCE21.11.1.B006-21.11.1.B006   nvidia-t4
```

在使用GPU时，可以根据标签让Pod与节点亲和，从而让Pod选择正确的节点，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-test
  template:
    metadata:
```

```
    labels:
      app: gpu-test
  spec:
    nodeSelector:
      accelerator: nvidia-t4
    containers:
    - image: nginx:perl
      name: container-0
      resources:
        requests:
          cpu: 250m
          memory: 512Mi
          nvidia.com/gpu: 1   # 申请GPU的数量
        limits:
          cpu: 250m
          memory: 512Mi
          nvidia.com/gpu: 1   # GPU数量的使用上限
    imagePullSecrets:
    - name: default-secret
```

6.3.2 GPU 虚拟化

6.3.2.1 GPU 虚拟化概述

CCE GPU虚拟化采用xGPU虚拟化技术，能够动态对GPU设备显存与算力进行划分，单个GPU卡最多虚拟化成20个GPU虚拟设备。相对于静态分配来说，虚拟化的方案更加灵活，最大程度保证业务稳定的前提下，可以完全由用户自己定义使用的GPU量，提高GPU利用率。

GPU 虚拟化的优势

CCE提供的GPU虚拟化功能优势如下：

- **灵活：**精细配置GPU算力占比及显存大小，算力分配粒度为5%GPU，显存分配粒度达MiB级别。
- **隔离：**支持显存和算力的严格隔离，支持单显存隔离，算力与显存同时隔离两类场景。
- **兼容：**业务无需重新编译，无需进行CUDA库替换，对业务无感。

前提条件

配置	支持版本
集群版本	v1.23.8-r0、v1.25.3-r0及以上
操作系统	Huawei Cloud EulerOS 2.0操作系统
GPU类型	支持T4、V100类型的GPU
驱动版本	GPU虚拟化功能仅支持470.57.02、510.47.03、535.54.03版本的GPU驱动。
运行时	仅支持containerd

配置	支持版本
插件	集群中需要同时安装以下插件： • Volcano调度器插件：1.10.5及以上版本 • CCE AI套件（NVIDIA GPU）插件：2.0.5及以上版本

约束与限制

- 单个GPU卡最多虚拟化成20个GPU虚拟设备。
- init容器不支持使用GPU虚拟化资源。
- GPU虚拟化支持显存隔离、显存与算力隔离两种隔离模式。单个GPU卡仅支持调度同一种隔离模式的工作负载。
- 使用GPU虚拟化后，不支持使用Autoscaler插件自动扩缩容GPU虚拟化节点。
- XGPU服务的隔离功能不支持以UVM的方式申请显存，即调用CUDA API cudaMallocManaged()，更多信息，请参见NVIDIA官方文档。请使用其他方式申请显存，例如调用cudaMalloc()等。
- 受GPU虚拟化技术的限制，容器内应用程序初始化时，通过nvidia-smi监测工具监测到的实时算力可能超过容器可用的算力上限。

6.3.2.2 准备 GPU 虚拟化资源

CCE GPU虚拟化采用xGPU虚拟化技术，能够动态对GPU设备显存与算力进行划分，单个GPU卡最多虚拟化成20个GPU虚拟设备。本文介绍如何在GPU节点上实现GPU的调度和隔离能力。

前提条件

配置	支持版本
集群版本	v1.23.8-r0、v1.25.3-r0及以上
操作系统	Huawei Cloud EulerOS 2.0操作系统
GPU类型	支持T4、V100类型的GPU
驱动版本	GPU虚拟化功能仅支持470.57.02、510.47.03、535.54.03版本的GPU驱动。
运行时	仅支持containerd
插件	集群中需要同时安装以下插件： • Volcano调度器插件：1.10.5及以上版本 • CCE AI套件（NVIDIA GPU）插件：2.0.5及以上版本

步骤一：开启 GPU 虚拟化

集群中需要同时安装**CCE AI套件（NVIDIA GPU）**插件和**Volcano调度器**插件。

步骤二：创建 GPU 节点

您需要在集群中创建支持GPU虚拟化的节点以使用GPU虚拟化功能，具体操作步骤请参见**创建节点**或**创建节点池**。

说明

如果您的集群中已有符合**前提条件**的GPU节点，您可以跳过此步骤。

步骤三（可选）：修改 Volcano 调度策略

Volcano针对GPU节点的调度策略默认为Spread，即如果节点配置相同，会选择一个正在运行的容器数量最少的节点，可以尽量将容器平均分配到各个节点。而Binpack调度策略与之相反，它会尽可能地把所有的容器调度到一台节点上运行，尽量少用节点，避免资源碎片化。

如果在使用GPU虚拟化特性时需要使用Binpack调度策略，可以在Volcano插件的高级配置中进行修改，具体操作步骤如下。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧选择“插件中心”。

步骤2 在右侧找到**Volcano调度器**插件，单击“编辑”。

步骤3 在编辑插件页面，修改插件的“高级配置”。

1. 在**nodeorder**插件中，添加**arguments**参数，配置leastrequested.weight为0，即资源分配最少的节点优先级设置为0。
2. 新增**binpack**插件，并指定xGPU自定义资源（**volcano.sh/gpu-core.percentage**和**volcano.sh/gpu-mem.128Mi**）的权重。

完整示例如下：

```
{
  "colocation_enable": "",
  "default_scheduler_conf": {
    "actions": "allocate, backfill, preempt",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
```

```
            },
            {
               "enablePreemptable": false,
               "name": "gang"
            },
            {
               "name": "conformance"
            }
         ]
      },
      {
         "plugins": [
            {
               "enablePreemptable": false,
               "name": "drf"
            },
            {
               "name": "predicates"
            },
            {
               "name": "nodeorder",
               //将资源分配最少的节点优先级设置为0
               "arguments": {
                  "leastrequested.weight": 0
               }
            }
         ]
      },
      {
         "plugins": [
            {
               "name": "cce-gpu-topology-predicate"
            },
            {
               "name": "cce-gpu-topology-priority"
            },
            {
               "name": "xgpu"
            },
            //添加binpack插件，指定xGPU资源的权重。
            {
               "name": "binpack",
               "arguments": {
                  "binpack.resources": "volcano.sh/gpu-core.percentage,volcano.sh/gpu-mem.128Mi",
                  "binpack.resources.volcano.sh/gpu-mem.128Mi": 10,
                  "binpack.resources.volcano.sh/gpu-core.percentage": 10
               }
            }
         ]
      },
      {
         "plugins": [
            {
               "name": "nodelocalvolume"
            },
            {
               "name": "nodeemptydirvolume"
            },
            {
               "name": "nodeCSIscheduling"
            },
            {
               "name": "networkresource"
            }
         ]
      }
   ]
},
"tolerations": [
```



```
    {
      "effect": "NoExecute",
      "key": "node.kubernetes.io/not-ready",
      "operator": "Exists",
      "tolerationSeconds": 60
    },
    {
      "effect": "NoExecute",
      "key": "node.kubernetes.io/unreachable",
      "operator": "Exists",
      "tolerationSeconds": 60
    }
  ]
}
```

----结束

6.3.2.3 使用 GPU 虚拟化

本文介绍如何使用GPU虚拟化能力实现算力和显存隔离，高效利用GPU设备资源。

前提条件

- 已完成**GPU虚拟化资源准备**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 单个GPU卡最多虚拟化成20个GPU虚拟设备。
- init容器不支持使用GPU虚拟化资源。
- GPU虚拟化支持显存隔离、显存与算力隔离两种隔离模式。单个GPU卡仅支持调度同一种隔离模式的工作负载。
- 使用GPU虚拟化后，不支持使用Autoscaler插件自动扩缩容GPU虚拟化节点。
- XGPU服务的隔离功能不支持以UVM的方式申请显存，即调用CUDA API cudaMallocManaged()，更多信息，请参见**NVIDIA官方文档**。请使用其他方式申请显存，例如调用cudaMalloc()等。
- 受GPU虚拟化技术的限制，容器内应用程序初始化时，通过nvidia-smi监测工具监测到的实时算力可能超过容器可用的算力上限。

创建 GPU 虚拟化应用

通过控制台创建

步骤1 登录CCE控制台。

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

步骤3 配置工作负载信息。

在“容器配置>基本信息”中设置xGPU配额：

- 显存：显存值单位为MiB，需为正整数，且为128的倍数。若配置的显存超过单张GPU卡的显存，将会出现无法调度状况。
- 算力：算力值单位为%，需为5的倍数，且最大不超过100。

说明

- 当显存设置为单张GPU卡的容量上限或算力设置为100%时，将会使用整张GPU卡。
- 使用GPU虚拟化时，工作负载调度器将默认指定为Volcano且不可更改。

图 6-3 设置 xGPU 配额

本文主要为您介绍GPU虚拟化的使用，其他参数详情请参见**工作负载**。

其余信息都配置完成后，单击“创建”。

步骤4 工作负载创建成功后，您可以尝试验证GPU虚拟化的隔离能力。

1. 登录容器查看容器被分配显存总量。

```
kubectl exec -it gpu-app -- nvidia-smi
```

预期输出：

```
Wed Apr 12 07:54:59 2023
+-----------------------------------------------------------------------------+
| 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  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
| N/A   27C    P0    37W / 300W |   4912MiB /  5120MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

预期输出表明，该容器被分配显存总量为5120 MiB，实际使用了4912MiB。

2. 查看所在节点的GPU显存隔离情况（在节点上执行）。

```
nvidia-smi
```

预期输出：

```
Wed Apr 12 09:31:10 2023
+-----------------------------------------------------------------------------+
| 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  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
| N/A   27C    P0    37W / 300W |   4957MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
```



```
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    760445      C   python                           4835MiB |
+-----------------------------------------------------------------------------+
```

预期输出表明，GPU节点上的显存总量为16160 MiB，其中示例Pod使用了4957MiB。

----结束

通过kubectl命令行创建

步骤1 使用kubectl连接集群。

步骤2 创建使用GPU虚拟化的应用。

说明

当前支持隔离显存或同时隔离显存与算力，暂不支持设置为仅隔离算力，即不支持单独设置volcano.sh/gpu-core.percentage。

创建gpu-app.yaml文件，内容如下：

- 仅隔离显存：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-app
  labels:
    app: gpu-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-app
  template:
    metadata:
      labels:
        app: gpu-app
    spec:
      containers:
      - name: container-1
        image: <your_image_address>     # 请替换为您的镜像地址
        resources:
          limits:
            volcano.sh/gpu-mem.128Mi: 40  # 该Pod分配的显存大小，该数值表示128Mi的倍数，即40*128=5120Mi
      imagePullSecrets:
        - name: default-secret
      schedulerName: volcano
```

- 同时隔离显存与算力：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-app
  labels:
    app: gpu-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-app
  template:
```



```
  metadata:
    labels:
      app: gpu-app
  spec:
    containers:
    - name: container-1
      image: <your_image_address>     # 请替换为您的镜像地址
      resources:
        limits:
          volcano.sh/gpu-mem.128Mi: 40  # 该Pod分配的显存大小，该数值表示128Mi的倍数，即40*128=5120Mi
          volcano.sh/gpu-core.percentage: 25    # 该Pod分配的算力大小
    imagePullSecrets:
      - name: default-secret
    schedulerName: volcano
```

表 6-1 关键参数说明

参数	是否必选	描述
volcano.sh/gpu-mem.128Mi	否	该数值表示128Mi的倍数，需为正整数，显存值单位为MiB。若配置的显存超过单张GPU卡的显存，将会出现无法调度状况。
volcano.sh/gpu-core.percentage	否	算力值单位为%，需为5的倍数，且最大不超过100。

说明

- 当显存设置为单张GPU卡的容量上限或算力设置为100%时，将会使用整张GPU卡。
- 使用GPU虚拟化时，工作负载调度器将默认指定为Volcano且不可更改。

步骤3 执行以下命令，创建应用。

```
kubectl apply -f gpu-app.yaml
```

步骤4 验证GPU虚拟化的隔离能力。

1. 登录容器查看容器被分配显存总量。

   ```
   kubectl exec -it gpu-app -- nvidia-smi
   ```

   预期输出：

   ```
   Wed Apr 12 07:54:59 2023
   +-----------------------------------------------------------------------------+
   | 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  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
   | N/A   27C    P0    37W / 300W |   4912MiB /  5120MiB |      0%      Default |
   |                               |                      |                  N/A |
   +-------------------------------+----------------------+----------------------+

   +-----------------------------------------------------------------------------+
   | Processes:                                                                  |
   |  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
   |        ID   ID                                                   Usage      |
   |=============================================================================|
   +-----------------------------------------------------------------------------+
   ```

   预期输出表明，该容器被分配显存总量为5120 MiB，实际使用了4912MiB。

2. 查看所在节点的GPU显存隔离情况（在节点上执行）。

```
nvidia-smi
```

预期输出：

```
Wed Apr 12 09:31:10 2023
+-----------------------------------------------------------------------------+
| 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  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
| N/A   27C    P0    37W / 300W |   4957MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    760445      C   python                           4835MiB |
+-----------------------------------------------------------------------------+
```

预期输出表明，GPU节点上的显存总量为16160 MiB，其中示例Pod使用了4957MiB。

----结束

6.3.2.4 兼容 Kubernetes 默认 GPU 调度模式

开启GPU虚拟化后，默认该GPU节点不再支持使用**Kubernetes默认GPU调度模式**的工作负载，即不再支持使用nvidia.com/gpu资源的工作负载。如果您在集群中已使用nvidia.com/gpu资源的工作负载，可在gpu-device-plugin插件配置中选择“虚拟化节点兼容GPU共享模式”选项，即可兼容Kubernetes默认GPU调度能力。

- 开启该兼容能力后，在工作负载中声明nvidia.com/gpu配额（即配置nvidia.com/gpu为小数，例如0.5）时将通过虚拟化GPU提供，实现GPU显存隔离，按照设定值的百分比为容器分配GPU显存（例如分配0.5×16GiB=8GiB的GPU显存，该数值需为128MiB的整数倍否则会自动向下取整）。如果在开启兼容能力前工作负载中已经使用nvidia.com/gpu资源，则不会转成虚拟化GPU，依然使用整卡资源。
- 开启该兼容能力后，使用nvidia.com/gpu配额时等价于开启虚拟化GPU显存隔离，可以和显存隔离模式的工作负载共用一张GPU卡，但不支持和算显隔离模式负载共用一张GPU卡。同时，还需遵循GPU虚拟化的其他**约束与限制**。
- 未开启该兼容能力时，在工作负载中声明nvidia.com/gpu配额仅影响调度结果，并不会有显存隔离的限制。即虽然配置nvidia.com/gpu为0.5，依然可以在容器中看到完整的GPU显存资源。且使用nvidia.com/gpu资源的工作负载无法和使用虚拟化显存的工作负载共同调度到同一节点。
- 编辑插件配置时，修改“虚拟化节点兼容GPU共享模式”选项，不会影响已运行的工作负载。修改该配置可能工作负载导致调度失败。例如，兼容能力从开启到关闭，已使用nvidia.com/gpu资源的工作负载仍存在虚拟化GPU显存隔离，会导致该GPU卡无法调度算显隔离模式的工作负载，您需要将使用nvidia.com/gpu资源的工作负载删除才可重新调度。

约束与限制

使用GPU虚拟化兼容Kubernetes默认GPU调度模式，要求配套的CCE AI 套件 (NVIDIA GPU)插件版本为2.0.10及以上、Volcano调度器插件版本为1.10.5及以上。

开启 Kubernetes 默认 GPU 调度模式兼容

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧选择“插件中心”。

步骤2 在右侧找到**CCE AI 套件 (NVIDIA GPU)**插件，单击“安装”。

如已安装该插件，单击“编辑”。

步骤3 填写插件配置，详情请参见**安装插件**。

开启GPU虚拟化后，可选择是否兼容nvidia.com/gpu字段，实现Kubernetes默认GPU调度能力的兼容。

步骤4 单击“安装”。

----结束

兼容 Kubernetes 默认 GPU 调度模式示例

步骤1 使用kubectl连接集群。

步骤2 创建一个使用nvidia.com/gpu资源的工作负载。

创建gpu-app.yaml文件，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-app
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpu-app
  template:
    metadata:
      labels:
        app: gpu-app
    spec:
      schedulerName: volcano
      containers:
      - image: <your_image_address>     # 请替换为您的镜像地址
        name: container-0
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 0.1   # 申请GPU的数量
          limits:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 0.1   # GPU数量的使用上限
      imagePullSecrets:
      - name: default-secret
```

步骤3 执行以下命令，创建应用。

```
kubectl apply -f gpu-app.yaml
```


步骤4 登录容器查看容器被分配显存总量。

```
kubectl exec -it gpu-app -- nvidia-smi
```

预期输出：

```
Thu Jul 27 07:53:49 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.57.02    Driver Version: 470.57.02    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 A30          Off  | 00000000:00:0D.0 Off |                    0 |
| N/A   47C    P0    34W / 165W |      0MiB /  2304MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

预期输出表明，Pod可使用的显存总量为2304MiB。

本示例中，GPU节点上的显存总量为24258MiB，而24258MiB * 0.1 = 2425.8MiB并非128MiB的整数倍，因此进行向下取整至18倍，即18 * 128MiB=2304MiB。

----结束

6.3.3 监控 GPU 资源指标

通过Prometheus和Grafana，可以实现对GPU资源指标的观测。本文以实际示例介绍如何通过Prometheus查看集群的GPU显存的使用。

本文将通过一个示例应用演示如何监控GPU资源指标，具体步骤如下：

1. **访问Prometheus**

   （可选）为Prometheus绑定LoadBalancer类型的Service，支持从外部访问Prometheus。

2. **监控GPU指标**

   在集群中部署使用GPU能力的工作负载，将自动上报GPU监控指标。

3. **访问Grafana**

   从Grafana可视化面板中查看Prometheus的监控数据。

前提条件

- 集群中已安装**云原生监控插件**插件。
- 集群中已安装**CCE AI套件（NVIDIA GPU）**插件，且插件版本不低于2.0.10。
- 如果需要监控GPU虚拟化监控指标，集群中需要已安装**Volcano调度器**插件，且插件版本不低于1.10.5。

访问 Prometheus

Prometheus插件安装完成后会在集群中部署一系列工作负载和Service。其中Prometheus的Server端会在monitoring命名空间下以有状态工作负载进行部署。

您可以创建一个公网**LoadBalancer类型Service**，这样就可以从外部访问Prometheus。

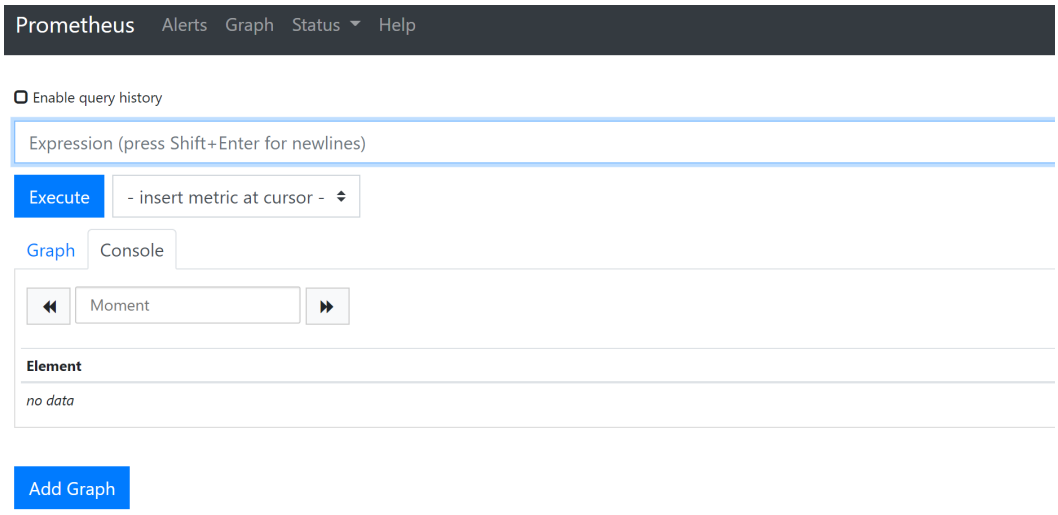
步骤1 登录CCE控制台，选择一个已安装Prometheus的集群，单击集群名称进入集群，在左侧导航栏中选择“服务”。

步骤2 单击右上角“YAML创建”，创建一个公网LoadBalancer类型的Service。

```
apiVersion: v1
kind: Service
metadata:
  name: prom-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: prometheus
    component: server
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 88            #服务端口号，可自定义
      targetPort: 9090     #Prometheus的默认端口号，无需更改
  selector:               #标签选择器可根据Prometheus Server实例的标签进行调整
    app.kubernetes.io/name: prometheus
    prometheus: server
  type: LoadBalancer
```

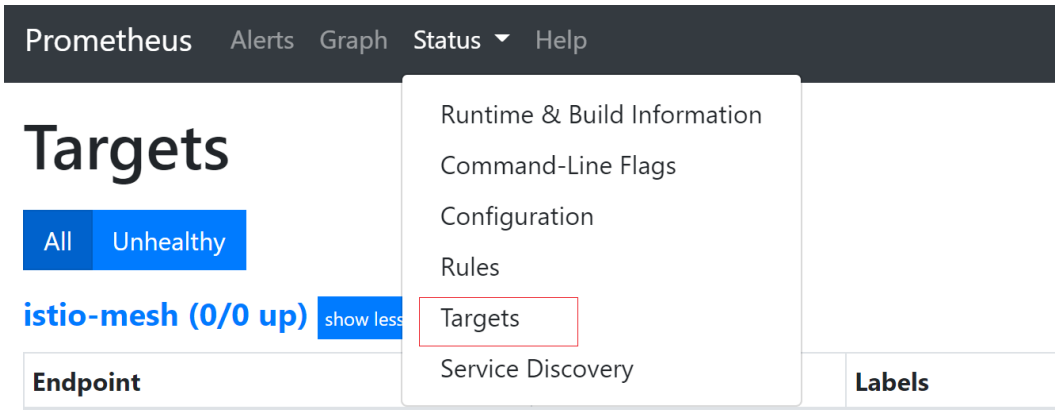
步骤3 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Prometheus。

图 6-4 访问 Prometheus



步骤4 单击“Status > Targets”，可以查看到Prometheus监控了哪些目标。

图 6-5 查看监控目标



----结束

监控 GPU 指标

创建一个使用GPU的工作负载，等工作负载正常运行后，访问Prometheus，在“Graph”页面中，查看GPU指标。

图 6-6 查看 GPU 监控指标

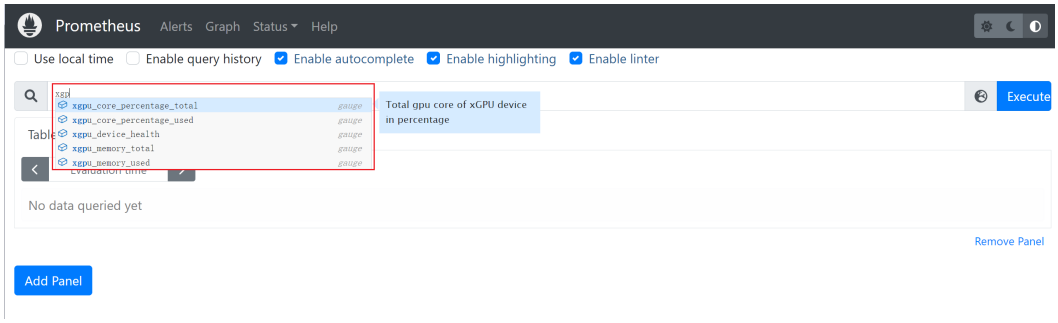


表 6-2 GPU 基础监控指标

类型	指标	监控级别	说明
利用率指标	cce_gpu_utilization	GPU卡	GPU卡算力使用率
	cce_gpu_memory_utilization	GPU卡	GPU卡显存使用率
	cce_gpu_encoder_utilization	GPU卡	GPU卡编码使用率
	cce_gpu_decoder_utilization	GPU卡	GPU卡解码使用率
	cce_gpu_utilization_process	GPU进程	GPU各进程算力使用率

类型	指标	监控级别	说明
	cce_gpu_memory_utilization_process	GPU进程	GPU各进程显存使用率
	cce_gpu_encoder_utilization_process	GPU进程	GPU各进程编码使用率
	cce_gpu_decoder_utilization_process	GPU进程	GPU各进程解码使用率
内存指标	cce_gpu_memory_used	GPU卡	GPU显存使用量
	cce_gpu_memory_total	GPU卡	GPU显存总量
	cce_gpu_memory_free	GPU卡	GPU显存空闲量
	cce_gpu_bar1_memory_used	GPU卡	GPU bar1 内存使用量
	cce_gpu_bar1_memory_total	GPU卡	GPU bar1 内存总量
频率	cce_gpu_clock	GPU卡	GPU时钟频率
	cce_gpu_memory_clock	GPU卡	GPU显存频率
	cce_gpu_graphics_clock	GPU卡	GPU图形处理器频率
	cce_gpu_video_clock	GPU卡	GPU视频处理器频率
物理状态数据	cce_gpu_temperature	GPU卡	GPU温度
	cce_gpu_power_usage	GPU卡	GPU功率
	cce_gpu_total_energy_consumption	GPU卡	GPU总能耗
带宽数据	cce_gpu_pcie_link_bandwidth	GPU卡	GPU PCIE 带宽
	cce_gpu_nvlink_bandwidth	GPU卡	GPU nvlink 带宽

类型	指标	监控级别	说明
	cce_gpu_pcie_throughput_rx	GPU卡	GPU PCIE 接收带宽
	cce_gpu_pcie_throughput_tx	GPU卡	GPU PCIE 发送带宽
	cce_gpu_nvlink_utilization_counter_rx	GPU卡	GPU nvlink 接收带宽
	cce_gpu_nvlink_utilization_counter_tx	GPU卡	GPU nvlink 发送带宽
隔离内存页面	cce_gpu_retired_pages_sbe	GPU卡	GPU单比特错误隔离页数量
	cce_gpu_retired_pages_dbe	GPU卡	GPU双比特错误隔离页数量

表 6-3 GPU 虚拟化监控指标

指标	监控级别	说明
xgpu_memory_total	GPU进程	GPU虚拟化显存总量。
xgpu_memory_used	GPU进程	GPU虚拟化显存使用量。
xgpu_core_percentage_total	GPU进程	GPU虚拟化算力总量。
xgpu_core_percentage_used	GPU进程	GPU虚拟化算力使用量。
gpu_schedule_policy	GPU卡	GPU虚拟化分三种模式： ● 0：显存隔离算力共享模式 ● 1：显存算力隔离模式 ● 2：默认模式，表示当前卡还没被用于GPU虚拟化设备分配。
xgpu_device_health	GPU卡	GPU虚拟化设备的健康情况。 ● 0：表示GPU虚拟化设备为健康状态。 ● 1：表示GPU虚拟化设备为非健康状态。

访问 Grafana

Prometheus插件同时安装了Grafana（一款开源可视化工具），并且与Prometheus进行了对接。您可以创建一个公网LoadBalancer类型Service，这样就可以从公网访问Grafana，从Grafana中看到Prometheus的监控数据。

单击访问地址，访问Grafana，选择合适的DashBoard，即可以查到相应的聚合内容。

步骤1 登录CCE控制台，选择一个已安装Prometheus插件的集群，单击集群名称进入集群，在左侧导航栏中选择“服务”。

步骤2 单击右上角“YAML创建”，为Grafana创建一个公网LoadBalancer类型Service。

```
apiVersion: v1
kind: Service
metadata:
  name: grafana-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: grafana
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 80     #服务端口号，可自定义
      targetPort: 3000     #Grafana的默认端口号，无需更改
  selector:
    app: grafana
  type: LoadBalancer
```

步骤3 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Grafana并选择合适的DashBoard，即可查看GPU资源状态。

图 6-7 查看 GPU 虚拟化资源

----结束

6.3.4 基于 GPU 监控指标的工作负载弹性伸缩配置

集群中包含GPU节点时，可通过GPU指标查看节点GPU资源的使用情况，例如GPU利用率、显存使用量等。在获取GPU监控指标后，用户可根据应用的GPU指标配置弹性伸缩策略，在业务波动时自适应调整应用的副本数量。

前提条件

- 目标集群已创建，且集群中包含GPU节点，并已运行GPU相关业务。
- 在集群中安装**CCE AI套件（NVIDIA GPU）**，且插件的metrics API正常工作。您可以登录GPU节点，执行以下命令进行检查：
  ```
  curl {Pod IP}:2112/metrics
  ```
  其中*{Pod IP}*是GPU插件的Pod IP，返回指标结果则为正常。
- 在集群中安装3.9.5及以上版本的**云原生监控插件**，且部署模式需选择“本地数据存储”。

采集 GPU 指标

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“配置项与密钥”。

步骤2 切换至“monitoring”命名空间，在“配置项”页签找到user-adapter-config配置项，并单击“更新”。

图 6-8 更新配置项

步骤3 在“配置数据”中单击config.yaml对应的“编辑”按钮，在rules字段下添加自定义指标采集规则。修改完成后单击“确定”保存配置。

如果您需要增加多个采集规则，可在rules字段下添加多个配置，关于采集规则配置详情请参见**Metrics Discovery and Presentation Configuration**。

针对cce_gpu_memory_utilization指标的自定义采集规则示例如下，更多GPU指标请参见**监控GPU指标**。

```
rules:
- seriesQuery: '{__name__=~"cce_gpu_memory_utilization",container!="",namespace!="",pod!=""}'
  seriesFilters: []
  resources:
    overrides:
      namespace:
        resource: namespace
      pod:
        resource: pod
  metricsQuery: sum(last_over_time(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)
```


图 6-9 设置自定义采集规则

步骤4 重新部署monitoring命名空间下的custom-metrics-apiserver工作负载。

图 6-10 重新部署 custom-metrics-apiserver

步骤5 重启后，可以通过以下指令查看对应的Pod的指标是否正常（注意替换命名空间和业务Pod名）。

```
# 查询指标
$ kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"

{"kind":"APIResourceList","apiVersion":"v1","groupVersion":"custom.metrics.k8s.io/v1beta1","resources":
[{"name":"pods/
cce_gpu_memory_utilization","singularName":"","namespaced":true,"kind":"MetricValueList","verbs":["get"]},
{"name":"namespaces/
cce_gpu_memory_utilization","singularName":"","namespaced":false,"kind":"MetricValueList","verbs":
["get"]}]}

# 查询负载的对应指标值
$ kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/test-gpu-
hpa-68667fdd94-grmd2/cce_gpu_memory_utilization"

{"kind":"MetricValueList","apiVersion":"custom.metrics.k8s.io/v1beta1","metadata":{},"items":
[{"describedObject":{"kind":"Pod","namespace":"default","name":"test-gpu-hpa-68667fdd94-
grmd2","apiVersion":"/
v1"},"metricName":"cce_gpu_memory_utilization","timestamp":"2024-01-10T08:36:44Z","value":"20","selecto
r":null}]}
```

----结束

创建弹性伸缩策略

步骤1 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“弹性伸缩”。

步骤2 策略类型选择“HPA+CronHPA策略”，并启用HPA策略。

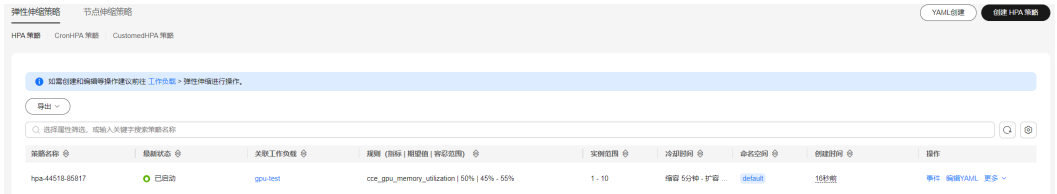
您可在“自定义策略”中选择GPU监控参数创建弹性伸缩策略，示例如下。

图 6-11 选择自定义指标

示例中以cce_gpu_memory_utilization（GPU显存使用率）作为伸缩指标，其余HPA参数的设置请根据实际需求进行设置，详情请参见**创建HPA策略**。

步骤3 返回“策略”页面，查看HPA策略已创建成功。

图 6-12 HPA 策略创建成功



----结束

6.3.5 GPU 故障处理

前提条件

如需将GPU事件同步上报至AOM，集群中需安装**云原生日志采集插件**，您可前往AOM服务查看GPU插件隔离事件。

GPU 插件隔离事件

当GPU显卡出现异常时，系统会将出现问题的GPU设备进行隔离，详细事件如表6-4所示。

表 6-4 GPU 插件隔离事件

事件原因	详细信息	描述	隔离结果
GPUMemoryError	Device=%s, UUID=%s, SN=%s has failed remapped rows; The device will go unhealthy.	NVML显存重映射行数查询异常	隔离单点故障的GPU设备
GPUMemoryError	Device=%s, UUID=%s, SN=%s has more than 60 retired pages caused by both multiple single bit ecc error and double bit ecc error, DBE error number: %d, SBE error number: %d; The device will go unhealthy.	GPU设备DBE错误与SBE错误总数过高（>60）	隔离单点故障的GPU设备
GPUMemoryError	Device=%s, UUID=%s, SN=%s has more than 4 SRAM uncorrectable ecc errors count; The device will go unhealthy.	GPU设备Uncorrectable ECC错误计数大于4	隔离单点故障的GPU设备
GPUXidError	Failed to determine gpu device uuid for Xid=%d; Marking all devices as unhealthy.	NVML获取设备UUID异常	隔离故障GPU节点的GPU设备
GPUXidError	Xid=%d on Device=%s, UUID=%s, SN=%s, the device will go unhealthy.	GPU设备存在Xid错误，Xid捕获范围为74和79	隔离单点故障的GPU设备
GPUHealthWarning	Device=%s, UUID=%s, SN=%s failed to get fan state.	GPU设备存在风扇异常	不隔离
GPUHealthWarning	Device=%s, UUID=%s, SN=%s failed to get power state.	GPU设备存在功率查询异常	不隔离

故障定位步骤

- **NVML显存重映射行数查询异常**

  GPU驱动或GPU设备存在异常，请根据GPU设备所在的节点类型（ECS或BMS），联系对应的客服进行处理。

- **GPU设备DBE错误与SBE错误总数过高**

  GPU驱动或GPU设备存在异常，请根据GPU设备所在的节点类型（ECS或BMS），联系对应的客服进行处理。

- **GPU设备存在Uncorrectable ECC错误**

a. 登录GPU隔离事件发生的节点。
b. 进入/usr/local/nvidia/bin目录，执行**nvidia-smi -q**命令。

   若nvidia-smi命令不存在或执行失败，有可能是驱动安装未就绪导致，可以重新安装GPU驱动后，再重试。
c. 观察执行结果中的ECC ERROR（发生ECC故障的记录）。

   - Correctable Error：不会影响业务，不会触发GPU隔离。
   - Uncorrectable Error：会导致业务中断，会触发GPU隔离。
d. 若存在Uncorrectable Error，可以尝试通过以下手段恢复：

   i. 配置目标节点污点（taints），驱逐目标节点存量的业务负载。
   ii. 重启目标节点。
   iii. 若重启后仍有该现象，则需要收集**nvidia-smi -q**命令的输出，然后根据GPU设备所在的节点类型（ECS或BMS），联系对应的客服进行处理。

- **NVML获取设备UUID异常**

  a. 登录GPU隔离事件发生的节点。
  b. 进入/usr/local/nvidia/bin目录。
  c. 执行**nvidia-smi**，观察执行结果中的设备ID，例如：00:0D.0。

     若nvidia-smi命令不存在或执行失败，有可能是驱动安装未就绪导致，可以重新安装GPU驱动后，再重试。
  d. 执行**lspci | grep NVIDIA**，观察执行结果中的设备ID。
  e. 比对上述两者结果，若存在不匹配的现象，收集两者输出结果，然后根据GPU设备所在的节点类型（ECS或BMS），联系对应的客服进行处理。

- **GPU设备存在Xid错误**

  a. 登录GPU隔离事件发生的节点。
  b. 执行**dmesg -T | grep -i NVRM**，观察结果输出。
  c. 假如存在Xid(PCI:0000:00:0x): xx格式的信息，则需要收集错误码，根据**Nvidia Xid Error**页面中确认详细原因。然后将详细原因和错误信息，根据GPU设备所在的节点类型（ECS或BMS），联系对应的客服进行处理。

- **GPU虚拟化设备可用内存远小于GPU物理显存**

  a. 登录GPU虚拟化节点。
  b. 执行**/usr/local/nvidia/bin/nvidia-smi**，观测目标GPU卡的物理显存，记录其序号。
  c. 执行**cat /proc/xgpu/{GPU卡序号}/meminfo**，注意替换命令中的{GPU卡序号}为步骤2获取的GPU卡序号，观测GPU虚拟化的可用显存。
  d. 比较步骤**2**和步骤**3**的可用显存。

     说明

     由于GPU厂商的驱动程序，本身就会占用一定量的物理显存，量级在300MB左右，这属于正常现象。例如Tesla T4配套510.47.03，驱动程序默认会占用280MiB；而该显存占用与厂商的驱动程序版本也有一定相关性，例如535系列驱动比470系列占用更多。

     若发现GPU虚拟化的可用显存远小于GPU卡的物理显存，一般是因为存在一些非GPU虚拟化发放的容器，占用了显存。
  e. 通过CCE控制台或kubectl命令，将目标节点的GPU负载排空。

f. 执行**rmmod xgpu_km**，进行GPU虚拟化模块的删除。
g. 通过CCE控制台或kubectl命令，将目标节点的nvidia-gpu-device-plugin Pod进行删除。
h. 等待nvidia-gpu-device-plugin Pod重建完成后，重新按照步骤2和步骤3进行结果核验。

6.4 NPU 调度

CCE支持在容器中使用NPU资源。

前提条件

- 创建NPU类型节点，具体请参见**创建节点**。
- 安装huawei-npu插件，具体请参见**CCE AI套件（Ascend NPU）**。

使用 NPU

创建工作负载申请NPU资源，可按如下方法配置，指定显卡的数量。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: npu-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: npu-test
  template:
    metadata:
      labels:
        app: npu-test
    spec:
      containers:
        - name: container-0
          image: nginx:perl
          resources:
            limits:
              cpu: 250m
              huawei.com/ascend-310: '1'
              memory: 512Mi
            requests:
              cpu: 250m
              huawei.com/ascend-310: '1'
              memory: 512Mi
      imagePullSecrets:
      - name: default-secret
```

通过**huawei.com/ascend-310**指定申请NPU的数量。

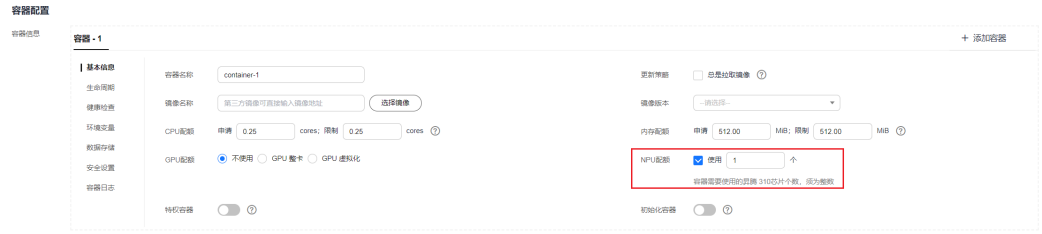
说明

使用**huawei.com/ascend-310**参数指定NPU数量时，requests和limits值需要保持一致。

指定**huawei.com/ascend-310**后，在调度时不会将负载调度到没有NPU的节点。如果缺乏NPU资源，会报类似“0/2 nodes are available: 2 Insufficient huawei.com/ascend-310.”的Kubernetes事件。

在CCE控制台使用NPU资源，只需在创建工作负载时，勾选NPU配额，并指定使用NPU芯片的数量。

图 6-13 使用 NPU



NPU 节点标签

创建NPU节点后，CCE会给节点打上对应标签，如下所示。

```
$ kubectl get node -L accelerator/huawei-npu
NAME          STATUS   ROLES    AGE     VERSION                                HUAWEI-NPU
10.100.2.59   Ready    <none>   2m18s   v1.19.10-r0-CCE21.11.1.B006-21.11.1.B006   ascend-310
```

在使用NPU时，可以根据标签让Pod与节点亲和，从而让Pod选择正确的节点，如下所示。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: npu-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: npu-test
  template:
    metadata:
      labels:
        app: npu-test
    spec:
      nodeSelector:
        accelerator/huawei-npu: ascend-310
      containers:
        - name: container-0
          image: nginx:perl
          resources:
            limits:
              cpu: 250m
              huawei.com/ascend-310: '1'
              memory: 512Mi
            requests:
              cpu: 250m
              huawei.com/ascend-310: '1'
              memory: 512Mi
      imagePullSecrets:
      - name: default-secret
```

6.5 Volcano 调度

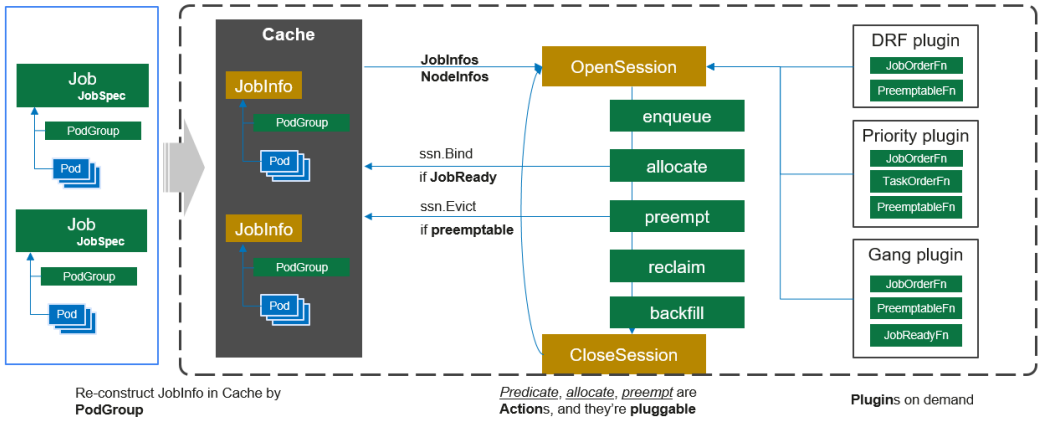
6.5.1 Volcano 调度概述

Volcano是一个基于Kubernetes的批处理平台，提供了机器学习、深度学习、生物信息学、基因组学及其他大数据应用所需要而Kubernetes当前缺失的一系列特性，提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力。

Volcano Scheduler

Volcano Scheduler是负责Pod调度的组件，它由一系列action和plugin组成。action定义了调度各环节中需要执行的动作；plugin根据不同场景提供了action 中算法的具体实现细节。Volcano Scheduler具有高度的可扩展性，您可以根据需要实现自己的action和plugin。

图 6-14 Volcano Scheduler 工作流



Volcano Scheduler的工作流程如下：

1. 客户端提交的Job被调度器识别到并缓存起来。
2. 周期性开启会话，一个调度周期开始。
3. 将没有被调度的Job发送到会话的待调度队列中。
4. 遍历所有的待调度Job，按照定义的次序依次执行enqueue、allocate、preempt、reclaim、backfill等动作，为每个Job找到一个最合适的节点。将该Job绑定到这个节点。action中执行的具体算法逻辑取决于注册的plugin中各函数的实现。
5. 关闭本次会话。

Volcano 自定义资源

- Pod组（PodGroup）：Pod组是Volcano自定义资源类型，代表一组强关联Pod的集合，主要用于批处理工作负载场景，比如Tensorflow中的一组ps和worker。
- 队列（Queue）：容纳一组PodGroup的队列，也是该组PodGroup获取集群资源的划分依据。
- 作业（Volcano Job，简称vcjob）：Volcano自定义的Job资源类型。区别于Kubernetes Job，vcjob提供了更多高级功能，如可指定调度器、支持最小运行Pod数、 支持task、支持生命周期管理、支持指定队列、支持优先级调度等。Volcano Job更加适用于机器学习、大数据、科学计算等高性能计算场景。
- 应用扩缩容优先级策略（Balancer与BalancerPolicyTemplate）：开启Volcano应用扩缩容优先级策略后，将会在集群中新增两类CRD资源，其中BalancerPolicyTemplate用来进行优先级策略定义，Balancer用来申明扩缩容优先级的作用范围。一个Balancer CR资源对应一个BalancerPolicyTemplate CR资源，两者结合共同申明哪些工作负载使用了哪些优先级策略。详情请参见**应用扩缩容优先级策略**。

6.5.2 使用 Volcano 调度工作负载

Volcano是一个基于Kubernetes的批处理平台，提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力，通过接入AI、大数据、基因、渲染等诸多行业计算框架服务终端用户，并针对计算型应用提供了作业调度、作业管理、队列管理等多项功能。

一般情况下，Kubernetes在调度工作负载时会使用自带的默认调度器，若需要使用Volcano调度器的能力，您可以为工作负载指定调度器。关于Kubernetes调度器的详情请参见为Pod指定调度器。

约束与限制

调度大量工作负载的场景下，Volcano会打印较多的日志，建议搭配日志服务使用，否则可能导致日志过多占满所在节点磁盘。

使用 Volcano 调度工作负载

使用Volcano调度工作负载时，只需要在Pod的spec字段中设置schedulerName参数并指定参数值为volcano，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        # 指定作业到q1队列
        scheduling.volcano.sh/queue-name: "q1"
        volcano.sh/preemptable: "true"
      labels:
        app: nginx
    spec:
      # 指定调度器为Volcano
      schedulerName: volcano
      containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: 1
            memory: 100Mi
          requests:
            cpu: 1
            memory: 100Mi
        ports:
        - containerPort: 80
```

同时，Volcano还支持设置负载所属队列和抢占属性等，可通过Pod的注解实现。目前Volcano支持的Pod注解配置如下：

表 6-5 Volcano 支持的 Pod 注解

Pod注解	说明
scheduling.volcano.sh/queue-name: "<queue-name>"	指定负载所在队列，其中<queue-name>为队列名称。
volcano.sh/preemptable: "true"	表示作业是否可抢占。开启后，认为该作业可以被抢占。 取值范围： • true：开启抢占。（默认为开启状态） • false：关闭抢占。

可通过查询Pod详情查看Pod是否由Volcano调度，以及被分配的队列：

```
kubectl describe pod <pod_name>
```

回显如下：

```
Spec:
  Min Member:  1
  Min Resources:
    Cpu:     100m
    Memory:  100Mi
  Queue:     q1
Status:
  Conditions:
    Last Transition Time:  2023-05-30T01:54:43Z
    Reason:                tasks in gang are ready to be scheduled
    Status:                True
    Transition ID:         70be1d7d-3532-41e0-8324-c7644026b38f
    Type:                  Scheduled
  Phase:                   Running
Events:
  Type    Reason     Age            From     Message
  ----    ------     ----           ----     -------
  Normal  Scheduled  0s (x3 over 2s)  volcano  pod group is ready
```

6.5.3 资源利用率优化调度

6.5.3.1 装箱调度（Binpack）

装箱调度（Binpack）是一种优化算法，以最小化资源使用量为目标，将资源合理地分配给每个任务，使所有资源都可以实现最大化的利用价值。在集群工作负载的调度过程中使用Binpack调度策略，调度器会优先将Pod调度到资源消耗较多的节点，减少各节点空闲资源碎片，提高集群资源利用率。

前提条件

- 已创建v1.19及以上版本的集群，详情请参见**购买Standard/Turbo集群**。
- 已安装Volcano插件，详情请参见**Volcano调度器**。

Binpack 功能介绍

Binpack调度算法的目标是尽量把已有的节点填满（即尽量不往空白节点分配）。具体实现上，Binpack调度算法为满足调度条件的节点打分，节点的资源利用率越高得分越

高。Binpack算法能够尽可能填满节点，将应用负载靠拢在部分节点，这非常有利于集群节点的自动扩缩容功能。

Binpack为调度器的多个调度插件之一，与其他插件共同为节点打分，用户可以自定义该插件整体权重和各资源维度打分权重，用以提高或降低Binpack在整体调度中的影响力。调度器在计算Binpack策略得分时，会考虑Pod请求的各种资源，如：CPU、Memory和GPU等扩展资源，并根据各种资源所配置的权重做平均。

Binpack 算法原理

Binpack在对一个节点打分时，会根据Binpack插件自身权重和各资源设置的权重值综合打分。首先，对Pod请求资源中的每类资源依次打分，以CPU为例，CPU资源在待调度节点的得分信息如下：

CPU.weight * (request + used) / allocatable

即CPU权重值越高，得分越高，节点资源使用量越满，得分越高。Memory、GPU等资源原理类似。其中：

- CPU.weight为用户设置的CPU权重
- request为当前Pod请求的CPU资源量
- used为当前节点已经分配使用的CPU量
- allocatable为当前节点CPU可用总量

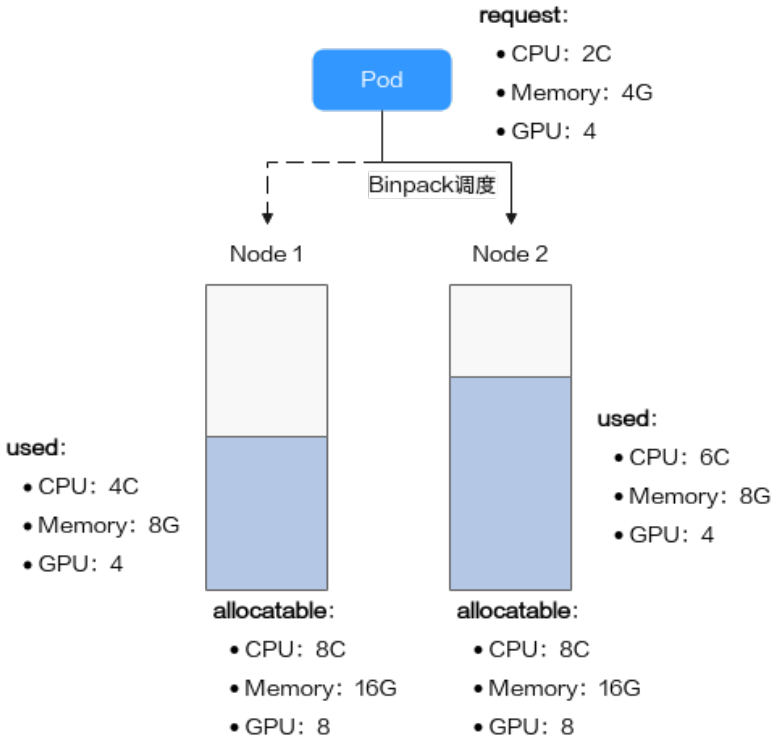
通过Binpack策略的节点总得分如下：

binpack.weight * (CPU.score + Memory.score + GPU.score) / (CPU.weight+ Memory.weight+ GPU.weight) * 100

即binpack插件的权重值越大，得分越高，某类资源的权重越大，该资源在打分时的占比越大。其中：

- binpack.weight为用户设置的装箱调度策略权重
- CPU.score为CPU资源得分，CPU.weight为CPU权重
- Memory.score为Memory资源得分，Memory.weight为Memory权重
- GPU.score为GPU资源得分，GPU.weight为GPU权重

图 6-15 Binpack 策略示例



如图所示，集群中存在两个节点，分别为Node 1和Node 2，在调度Pod时，Binpack策略对两个节点分别打分。

假设集群中CPU.weight配置为1，Memory.weight配置为1，GPU.weight配置为2，binpack.weight配置为5。

1. Binpack对Node 1的打分信息如下：

   各资源按照公式计算得分，具体信息如下：

   - CPU Score：CPU.weight * (request + used) / allocatable = 1 * （2 + 4）/ 8 = 0.75
   - Memory Score：Memory.weight * (request + used) / allocatable = 1 * (4 + 8) / 16 = 0.75
   - GPU Score： GPU.weight * (request + used) / allocatable = 2 * （4 + 4）/ 8 = 2

   节点总得分按照**binpack.weight * (CPU.score + Memory.score + GPU.score) / (CPU.weight+ Memory.weight+ GPU.weight) * 100**公式进行计算，具体如下：

   假设binpack.weight配置为5，Node 1在Binpack策略下的得分：5 * （0.75 + 0.75 + 2）/（1 + 1 + 2）* 100 = 437.5

2. Binpack对Node 2的打分信息如下：

   - CPU Score：CPU.weight * (request + used) / allocatable = 1 * （2 + 6）/ 8 = 1
   - Memory Score：Memory.weight * (request + used) / allocatable = 1 * (4 + 8) / 16 = 0.75
   - GPU Score：GPU.weight * (request + used) / allocatable = 2 * （4 + 4）/ 8 = 2

Node 2在Binpack策略下的得分：5 *（1 + 0.75 + 2）/（1 + 1 + 2）* 100 = 468.75

综上，Node 2得分大于Node 1，按照Binpack策略，Pod将会优先调度至Node 2。

配置装箱调度策略

安装Volcano后，Binpack策略默认生效。如果默认配置无法达到您降低资源碎片的目标，可以通过“配置中心 > 调度配置”页面自定义Binpack策略权重和各资源维度权重值，增加或降低Binpack策略在整体调度中的影响力。

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，在右侧选择“调度配置”页签。

步骤3 在“资源利用率优化调度”配置中，启用装箱策略 (binpack)。

表 6-6 装箱策略权重配置

名称	说明	默认值
装箱调度策略权重	增大该权重值，可提高装箱策略在整体调度中的影响力。	10
CPU权重	增大该权重值，优先提高集群CPU利用率。	1
内存权重	增大该权重值，优先提高集群Memory利用率。	1
自定义资源类型	指定Pod请求的其他自定义资源类型，例如nvidia.com/gpu。增大该权重值，优先提高指定资源的利用率。	-

图 6-16 资源利用率优化调度

步骤4 修改完成后，单击“确认配置”。

----结束

6.5.3.2 重调度（Descheduler）

集群中的调度是将pending状态的Pod分配到节点运行的过程，在CCE集群之中，Pod的调度依赖于集群中的调度器（kube-scheduler或者Volcano调度器）。调度器是通过

一系列算法计算出Pod运行的最佳节点，但是Kubernetes集群环境是存在动态变化的，例如某一个节点需要维护，这个节点上的所有Pod会被驱逐到其他节点，但是当维护完成后，之前被驱逐的Pod并不会自动回到该节点上来，因为Pod一旦被绑定了节点是不会触发重新调度的。由于这些变化，集群在一段时间之后就可能会出现不均衡的状态。

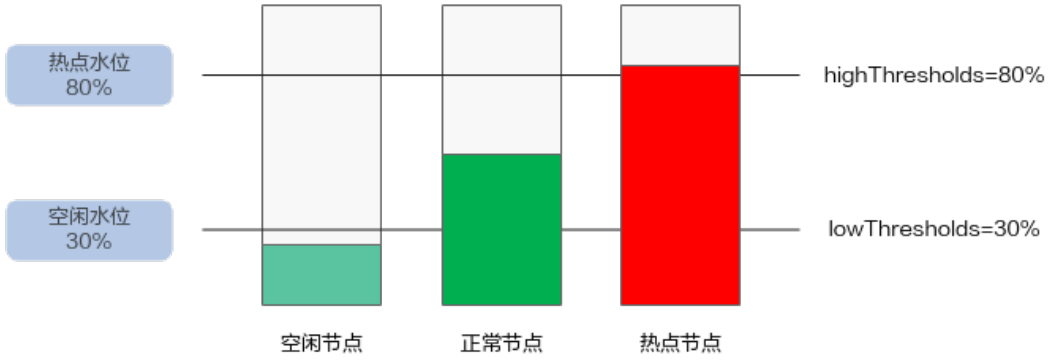
为了解决上述问题，Volcano调度器可以根据设置的策略，驱逐不符合配置策略的Pod，让其重新进行调度，达到均衡集群负载、减少资源碎片化的目的。

重调度功能介绍

负载感知重调度（LoadAware）

在K8s集群治理过程中，常常会因CPU、内存等高使用率状况而形成热点，既影响了当前节点上Pod的稳定运行，也会导致节点发生故障的几率的激增。为了应对集群节负载不均衡等问题，动态平衡各个节点之间的资源使用率，需要基于节点的相关监控指标，构建集群资源视图，在集群治理阶段，通过实时监控，在观测到节点资源率较高、节点故障、Pod 数量较多等情况时，可以自动干预，迁移资源使用率高的节点上的一些Pod到利用率低的节点上。

图 6-17 LoadAware 策略示意图



使用该插件时，highThresholds需要大于lowThresholds，否则重调度器无法启用。

- 正常节点：资源利用率大于等于30%且小于等于80%的节点。此节点的负载水位区间是期望达到的合理区间范围。
- 热点节点：资源利用率高于80%的节点。热点节点将驱逐一部分Pod，降低负载水位，使其不超过80%。重调度器会将热点节点上面的Pod调度到空闲节点上面。
- 空闲节点：资源利用率低于30%的节点。

CPU和内存资源碎片率整理策略（HighNodeUtilization）

从分配率低的节点上驱逐Pod。这个策略必须与Volcano调度器的binpack策略或者kube-scheduler调度器的MostAllocated策略一起使用。阈值可以分为CPU和内存两种资源角度进行配置。

前提条件

- 已创建v1.19.16及以上版本的集群，具体操作请参见**购买Standard/Turbo集群**。
- 集群中已安装1.11.5及以上版本的Volcano插件，具体操作请参见**Volcano调度器**。

约束与限制

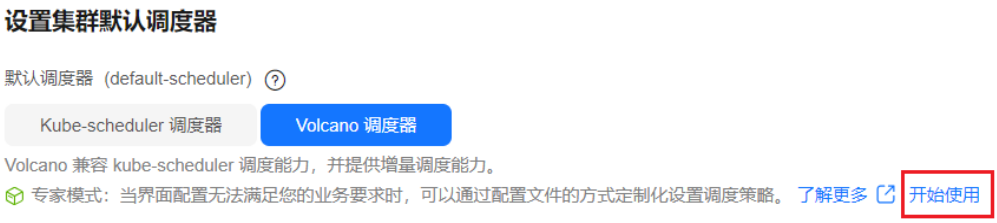
- 重调度之后的Pod，需要调度器进行调度，重调度器并未进行任何对于Pod和节点的标记行为，所以被驱逐的Pod调度到节点的行为完全被调度器控制，存在驱逐之后，被驱逐的Pod调度到原来节点的可能性。
- 重调度功能暂不支持Pod间存在反亲和性的场景。如果使用重调度功能驱逐某个Pod后，由于该Pod与其他已运行的Pod存在反亲和性，调度器仍可能将其调度回驱逐前的节点上。
- 配置负载感知重调度（LoadAware）时，Volcano调度器需要同时开启负载感知调度；配置CPU和内存资源碎片率整理策略（HighNodeUtilization）时，Volcano调度器需要同时开启binpack调度策略。

配置负载感知重调度策略

配置负载感知重调度（LoadAware）时，Volcano调度器需要同时开启负载感知调度，示例步骤如下。

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“配置中心”，通过“调度配置”页面启用负载感知调度。

步骤2 在“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



步骤3 配置负载感知重调度策略。使用Volcano 1.11.21及更新版本，JSON格式的配置示例如下：

```
{
  "colocation_enable": "",
  "default_scheduler_conf": {
    "actions": "allocate, backfill, preempt",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
          },
          {
            "enablePreemptable": false,
            "name": "gang"
          },
          {
            "name": "conformance"
          }
        ]
      },
      {
        "plugins": [
          {
            "enablePreemptable": false,
            "name": "drf"
          },
          {
```



```
        "name": "predicates"
       },
       {
        "name": "nodeorder"
       },
       {
        "name": "usage",
        "enablePredicate": true,
        "arguments": {
         "usage.weight": 5,
         "cpu.weight": 1,
         "memory.weight": 1,
         "thresholds": {
          "cpu": 80,
          "mem": 80
         }
        }
       }
      ]
    },
    {
      "plugins": [
       {
        "name": "cce-gpu-topology-predicate"
       },
       {
        "name": "cce-gpu-topology-priority"
       },
       {
        "name": "cce-gpu"
       }
      ]
    },
    {
      "plugins": [
       {
        "name": "nodelocalvolume"
       },
       {
        "name": "nodeemptydirvolume"
       },
       {
        "name": "nodeCSIscheduling"
       },
       {
        "name": "networkresource"
       }
      ]
    }
  ]
},
"deschedulerPolicy": {
  "profiles": [
    {
      "name": "ProfileName",
      "pluginConfig": [
       {
        "args": {
         "ignorePvcPods": true,
         "nodeFit": true,
         "priorityThreshold": {
          "value": 100
         }
        },
        "name": "DefaultEvictor"
       },
       {
        "args": {
         "evictableNamespaces": {
```



```
            "exclude": ["kube-system"]
          },
          "metrics": {
            "type": "prometheus_adaptor"
          },
          "targetThresholds": {
            "cpu": 80,
            "memory": 85
          },
          "thresholds": {
            "cpu": 30,
            "memory": 30
          }
        },
        "name": "LoadAware"
      }
    ],
    "plugins": {
      "balance": {
        "enabled": ["LoadAware"]
      }
    }
  }
 ]
},
"descheduler_enable": "true",
"deschedulingInterval": "10m"
}
```

表 6-7 集群重调度策略关键参数

参数	说明
descheduler_enable	集群重调度策略开关。 ● true：启用集群重调度策略。 ● false：不启用集群重调度策略。
deschedulingInterval	重调度的周期。
deschedulerPolicy	集群重调度策略，详情请参见表6-8。

表 6-8 deschedulerPolicy 配置参数

参数	说明
profiles.[].plugins.balance.enable.[]	指定集群重调度策略类型。 LoadAware：表示使用负载感知重调度策略。
profiles.[].pluginConfig.[].name	使用负载感知重调度策略时，会使用以下配置： ● DefaultEvictor：默认驱逐策略。 ● LoadAware：负载感知重调度策略。

参数	说明

profiles.[].pluginConfig.[].args	集群重调度策略的具体配置。
	● 对于DefaultEvictor配置，配置参数如下：
	– ignorePvcPods：是否忽略挂载PVC的Pod，true表示忽略，false表示不忽略。该忽略动作未根据PVC类型（LocalPV/SFS/EVS等）进行区分。
	– nodeFit：是否重调度时是否考虑节点上存在的调度配置，例如节点亲和性、污点等。true表示考虑，false表示不考虑。
	– priorityThreshold：优先级设置。当Pod的优先级大于或者等于该值时，不会被驱逐。示例如下：
	{
	"value": 100
	}
	● 对于LoadAware配置，配置参数如下：
	– evictableNamespaces：驱逐策略的适用命名空间，默认范围设置为除kube-system命名空间。示例如下：
	{
	"exclude": ["kube-system"]
	}
	– metrics：监控数据采集方式，当前支持通过Custom Metrics API（prometheus_adaptor聚合数据）和Prometheus 直接查询。
	Volcano 1.11.17及之后的版本推荐使用Custom Metrics API的方式获取监控数据，示例如下：
	{
	"type": "prometheus_adaptor"
	}
	Volcano 1.11.5至1.11.16版本推荐使用Prometheus直接查询的方式获取监控数据，需填写prometheus server的地址信息，示例如下：
	{
	"address": "http://10.247.119.103:9090",
	"type": "prometheus"
	}
	– targetThresholds：节点驱逐Pod的阈值，当节点的真实利用率高于此阈值时，上面的Pod会被驱逐。示例如下：
	{
	"cpu": 60,
	"memory": 65
	}
	– thresholds：节点承载Pod的阈值，当节点的真实利用率低于此阈值时，表示该节点可以承载被驱逐的Pod。示例如下：
	{
	"cpu": 30,
	"memory": 30
	}

步骤4 完成以上配置后，单击“确定”。

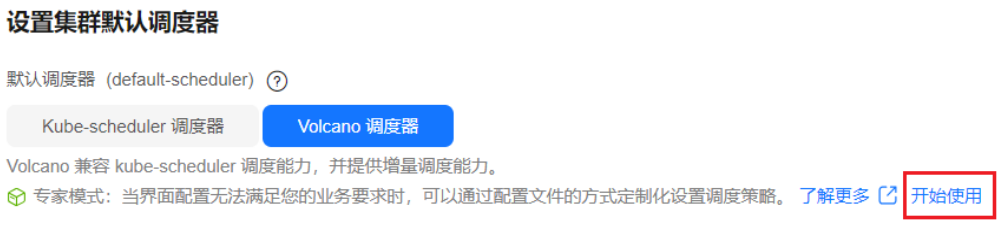
----结束

配置资源碎片整理策略

配置CPU和内存资源碎片率整理策略（HighNodeUtilization）时，Volcano调度器需要同时开启binpack调度策略，示例步骤如下。

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“配置中心”，通过“调度配置”页面启用装箱策略(binpack)。详情请参见**装箱调度（Binpack）**。

步骤2 在“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



步骤3 配置资源碎片整理策略，JSON格式的配置示例如下。

```
{
  "colocation_enable": "",
  "default_scheduler_conf": {
    "actions": "allocate, backfill, preempt",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
          },
          {
            "enablePreemptable": false,
            "name": "gang"
          },
          {
            "name": "conformance"
          },
          {
            "arguments": {
              "binpack.weight": 5
            },
            "name": "binpack"
          }
        ]
      },
      {
        "plugins": [
          {
            "enablePreemptable": false,
            "name": "drf"
          },
          {
            "name": "predicates"
          },
          {
            "name": "nodeorder"
          }
        ]
      },
```



```
    {
      "plugins": [
        {
          "name": "cce-gpu-topology-predicate"
        },
        {
          "name": "cce-gpu-topology-priority"
        },
        {
          "name": "cce-gpu"
        }
      ]
    },
    {
      "plugins": [
        {
          "name": "nodelocalvolume"
        },
        {
          "name": "nodeemptydirvolume"
        },
        {
          "name": "nodeCSIscheduling"
        },
        {
          "name": "networkresource"
        }
      ]
    }
  ]
},
"deschedulerPolicy": {
  "profiles": [
    {
      "name": "ProfileName",
      "pluginConfig": [
        {
          "args": {
            "ignorePvcPods": true,
            "nodeFit": true,
            "priorityThreshold": {
              "value": 100
            }
          },
          "name": "DefaultEvictor"
        },
        {
          "args": {
            "evictableNamespaces": {
              "exclude": ["kube-system"]
            },
            "thresholds": {
              "cpu": 25,
              "memory": 25
            }
          },
          "name": "HighNodeUtilization"
        }
      ],
      "plugins": {
        "balance": {
          "enabled": ["HighNodeUtilization"]
        }
      }
    }
  ]
},
"descheduler_enable": "true",
```



```
  "deschedulingInterval": "10m"
}
```

表 6-9 集群重调度策略关键参数

参数	说明
descheduler_enable	集群重调度策略开关。 ● true：启用集群重调度策略。 ● false：不启用集群重调度策略。
deschedulingInterval	重调度的周期。
deschedulerPolicy	集群重调度策略，详情请参见表6-10。

表 6-10 deschedulerPolicy 配置参数

参数	说明
profiles.[].plugins.balance.enable.[]	指定集群重调度策略类型。 HighNodeUtilization：表示使用资源碎片整理策略。
profiles.[].pluginConfig.[].name	使用负载感知重调度策略时，会使用以下配置： ● DefaultEvictor：默认驱逐策略。 ● HighNodeUtilization：资源碎片整理策略。

参数	说明

profiles.[].pluginConfig.[].args	集群重调度策略的具体配置。
	● 对于DefaultEvictor配置，配置参数如下：
	– ignorePvcPods：是否忽略挂载PVC的Pod，true表示忽略，false表示不忽略。该忽略动作未根据PVC类型（LocalPV/SFS/EVS等）进行区分。
	– nodeFit：是否重调度时是否考虑节点上存在的调度配置，例如节点亲和性、污点等。true表示考虑，false表示不考虑。
	– priorityThreshold：优先级设置。当Pod的优先级大于或者等于该值时，不会被驱逐。示例如下：
	{
	"value": 100
	}
	● 对于HighNodeUtilization配置，配置参数如下：
	– evictableNamespaces：驱逐策略的适用命名空间，默认范围设置为除kube-system命名空间。示例如下：
	{
	"exclude": ["kube-system"]
	}
	– thresholds：节点驱逐Pod的阈值，当节点的真实利用率低于此阈值时，该节点上的Pod会被驱逐。示例如下：
	{
	"cpu": 25,
	"memory": 25
	}

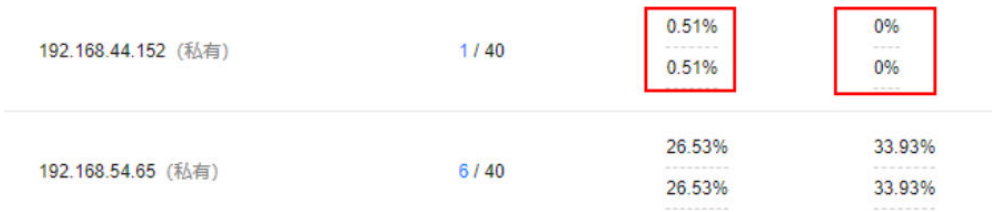
步骤4 完成以上配置后，单击“确定”。

----结束

使用案例

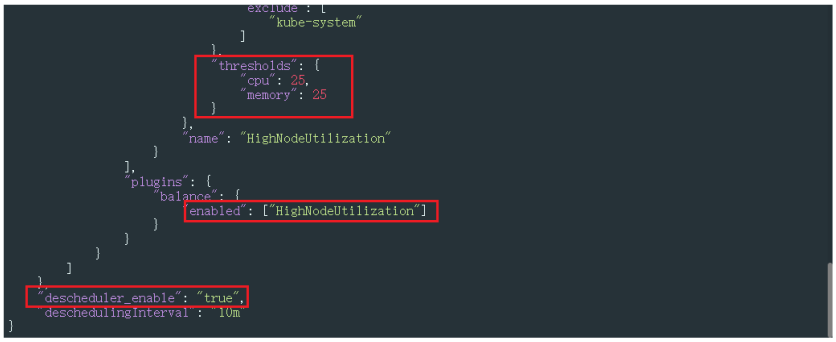
资源碎片整理策略（HighNodeUtilization）使用案例

1. 查看集群之中的节点，发现存在部分分配率过低的节点。



2. 编辑Volcano参数，开启重调度器，并设置CPU和内存的阈值为25。即表示节点的分配率小于25%时，该节点上的Pod会被驱逐。

高级配置



3. 设置该策略后，将192.168.44.152节点上的Pod迁移到节点192.168.54.65，达到碎片整理的目的。

常见问题

当输入参数错误时，会有报警事件，例如：输入的配置不符合阈值范围、输入的配置格式不正确。如下图所示，可以按照事件提示进行修改。

事件

6.5.3.3 节点池亲和性调度

在替换节点池、节点滚动升级等场景中，需要使用新节点池替换旧节点池。在这些场景下，为做到业务不感知，可以在业务触发变更时，将业务的Pod软亲和调度到新的节点池上。这种软亲和调度会尽量将新创建的Pod或者重调度的Pod调度到新的节点池，如果新节点池资源不足，或者新节点池无法调度，也要能将Pod调度到旧节点池上。节点池替换、节点滚动升级等场景中，业务不需要也不应该感知，所以不会在业务负载中声明节点亲和配置，而需要在集群调度层面，使用软亲和方式，在业务变更时将Pod尽量调度到新的节点池上。

Volcano的目标是在业务负载未配置节点软亲和时，在调度层将业务的Pod软调度到指定节点上。

调度优先级介绍

节点池软亲和调度，是通过节点池上的标签（Label）进行软亲和，具体是通过给每一个节点进行打分的机制来排序筛选最优节点。

原则：尽可能把Pod调度到带有相关标签的节点上。

打分公式如下：

score = weight * MaxNodeScore * haveLabel

参数说明：

- weight：节点池软亲和plugin的权重。
- MaxNodeScore：节点最大得分，值为100。
- haveLabel：节点上是否存在插件中配置的label，如果有，值为1，如果节点上没有，值为0。

前提条件

- 已创建v1.19.16及以上版本的集群，具体操作请参见**购买Standard/Turbo集群**。
- 集群中已安装1.11.5及以上版本的Volcano插件，具体操作请参见**Volcano调度器**。

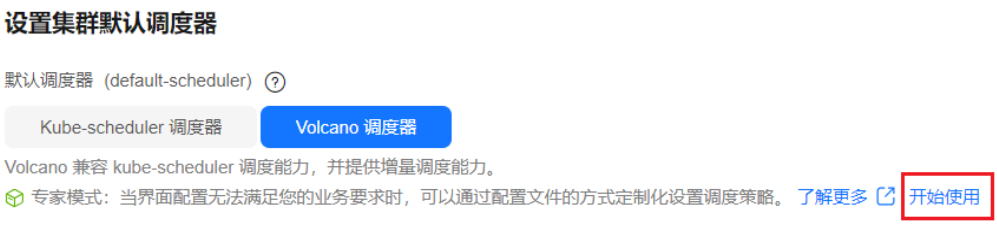
配置 Volcano 节点池软亲和调度策略

步骤1 在节点池上配置用于亲和调度的标签。

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签。
3. 单击节点池名称后的“更新”，在弹出的“更新节点池”页面中配置参数，在“K8s标签”中配置对应的标签。

   示例如下：

步骤2 单击左侧导航栏的“配置中心”，切换至“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



步骤3 设置Volcano调度器配置参数，JSON格式的配置示例如下。

```
...
  "default_scheduler_conf": {
    "actions": "allocate, backfill, preempt",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
```

```
            },
            {
               "name": "gang"
            },
            {
               "name": "conformance"
            }
         ]
      },
      {
         "plugins": [
            {
               "name": "drf"
            },
            {
               "name": "predicates"
            },
            {
               "name": "nodeorder"
            }
         ]
      },
      {
         "plugins": [
            {
               "name": "cce-gpu-topology-predicate"
            },
            {
               "name": "cce-gpu-topology-priority"
            },
            {
               "name": "cce-gpu"
            },
            {
               // 开启节点池亲和性调度
               "name": "nodepoolaffinity",
               // 节点池亲和性调度权重及标签设置
               "arguments": {
                  "nodepoolaffinity.weight": 10000,
                  "nodepoolaffinity.label": "nodepool1=nodepool1"
               }
            }
         ]
      },
      {
         "plugins": [
            {
               "name": "nodelocalvolume"
            },
            {
               "name": "nodeemptydirvolume"
            },
            {
               "name": "nodeCSIscheduling"
            },
            {
               "name": "networkresource"
            }
         ]
      }
   ]
},
...
```

步骤4 完成以上配置后，单击“确定”。

----结束

6.5.3.4 负载感知调度

Volcano调度器提供节点CPU、Memory的负载感知调度能力，感知集群内节点CPU、Memory的负载情况，将Pod优先调度到负载较低的节点，实现节点负载均衡，避免出现因单个节点负载过高而导致的应用程序或节点故障。

前提条件

- 已创建v1.21及以上版本的集群，详情请参见**购买Standard/Turbo集群**。
- 已安装Volcano 1.11.14及以上版本的插件，详情请参见**Volcano调度器**。
- 已安装CCE云原生监控插件（kube-prometheus-stack），并开启“本地数据存储”模式，详情请参见**云原生监控插件**。
- 使用kubectl连接集群，具体操作步骤请参见**通过kubectl连接集群**。

功能介绍

原生Kubernetes调度器只能基于资源的申请值进行调度，然而Pod的真实资源使用率，往往与其所申请资源的Request/Limit差异很大，这直接导致了集群负载不均的问题：

1. 集群中的部分节点，资源的真实使用率远低于资源申请值的分配率，却没有被调度更多的Pod，这造成了比较大的资源浪费。
2. 集群中的另外一些节点，其资源的真实使用率事实上已经过载，却无法为调度器所感知到，这极大可能影响到业务的稳定性。

Volcano提供基于真实负载调度的能力，在资源满足的情况下，Pod优先被调度至真实负载低的节点，集群各节点负载趋于均衡。

随着集群状态，工作负载流量与请求的动态变化，节点的利用率也在实时变化，为防止Pod调度完成后，集群再次出现负载极端不均衡的情况下，Volcano同时提供重调度能力，通过负载感知和热点打散重调度结合使用，可以获得集群最佳的负载均衡效果。关于热点打散重调度能力的使用请参见**重调度（Descheduler）**。

工作原理

负载感知调度能力由Volcano与CCE云原生监控插件配合完成，开启该能力时，按照Prometheus adapt规则定义负载感知调度所需的CPU、Memory指标信息，CCE云原生监控系统按照定义的指标规则采集并保存各节点的CPU、Memory的真实负载信息，Volcano根据CCE云原生监控系统提供的CPU、Memory真实负载信息对节点进行打分排序，优先选择负载更低的节点参与调度。

负载感知调度能力考虑CPU和Memory两个维度，采用加权平均的方式对个节点打分，包括CPU、Memory资源维度权重和负载感知策略自身权重，优先选择得分最高的节点参与调度。CPU、Memory和插件自身权重可以通过“配置中心>调度配置”自定义配置。

节点得分计算公式： 负载感知策略权重 *（（1 - CPU资源利用率）* CPU权重 +（1 - Memory资源利用率）* 内存权重）/（CPU权重 + 内存权重）

- CPU资源利用率：所有节点最近10分钟的CPU平均利用率，采集频率可通过Prometheus adapt规则进行修改。
- Memory资源利用率：所有节点最近10分钟的Memory平均利用率

使用 CCE 云原生监控插件设置负载感知调度

步骤1 安装CCE云原生监控插件后，您需要开启Metrics API以提供容器资源指标的能力，如CPU、内存使用量。

说明

仅云原生监控插件开启本地数据存储时，可通过Metrics API提供资源指标。

首先执行以下命令查询集群中是否已开启Metrics API，如果已开启则可跳过本步骤。

```
kubectl get APIServices | grep v1beta1.metrics.k8s.io
```

若存在回显，则表示Metrics API已开启，可跳过本步骤进行下一步添加指标采集规则。

若未查询到Metrics API，要将其开启，可手动创建对应APIService对象。

1. 创建一个文件，命名为metrics-apiservice.yaml。文件内容如下：

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  labels:
    app: custom-metrics-apiserver
    release: cceaddon-prometheus
  name: v1beta1.metrics.k8s.io
spec:
  group: metrics.k8s.io
  groupPriorityMinimum: 100
  insecureSkipTLSVerify: true
  service:
    name: custom-metrics-apiserver
    namespace: monitoring
    port: 443
  version: v1beta1
  versionPriority: 100
```

2. 然后执行以下命令创建对应APIService对象。

```
kubectl create -f metrics-apiservice.yaml
```

3. 再次执行以下命令查询集群中是否已开启Metrics API。

```
kubectl get APIServices | grep v1beta1.metrics.k8s.io
```

说明

开启Metrics API后，如果需要卸载云原生监控插件，请执行以下kubectl命令，同时删除APIService对象，否则残留的APIService资源将导致Kubernetes Metrics Server插件安装失败。

```
kubectl delete APIService v1beta1.metrics.k8s.io
```

步骤2 添加自定义指标采集规则。

1. 修改user-adapter-config配置项。

```
kubectl edit configmap user-adapter-config -n monitoring
```

2. 在rules字段下添加自定义指标采集规则。

自定义指标采集规则如下，红色为本次添加的指标采集规则，黑色为已有规则，在已有规则基础上添加红色规则即可。

```
...
data:
  config.yaml: >
    rules:
    - seriesQuery: '{__name__=~"node_cpu_seconds_total"}'
      resources:
      overrides:
        instance:
```

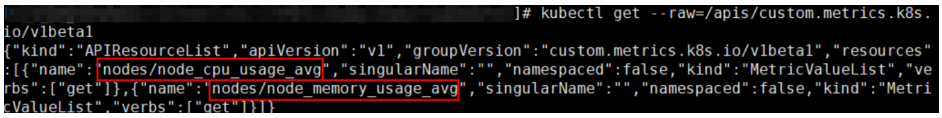


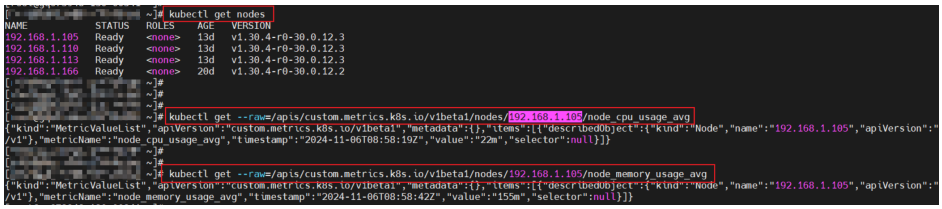
```
          resource: node
      name:
      matches: node_cpu_seconds_total
      as: node_cpu_usage_avg
      metricsQuery: avg_over_time((1 - avg (irate(<<.Series>>{mode="idle"}[5m])) by (instance))
[10m:30s])
  - seriesQuery: '{__name__=~"node_memory_MemTotal_bytes"}'
      resources:
      overrides:
          instance:
          resource: node
      name:
      matches: node_memory_MemTotal_bytes
      as: node_memory_usage_avg
      metricsQuery: avg_over_time(((1-node_memory_MemAvailable_bytes/<<.Series>>))[10m:30s])
  resourceRules:
  cpu:
      containerQuery: sum(rate(container_cpu_usage_seconds_total{<<.LabelMatchers>>,container!
="",pod!=""}[1m])) by (<<.GroupBy>>)
      nodeQuery: sum(rate(container_cpu_usage_seconds_total{<<.LabelMatchers>>, id='/'}[1m])) by
(<<.GroupBy>>)
      resources:
      overrides:
          instance:
          resource: node
          namespace:
          resource: namespace
          pod:
          resource: pod
      containerLabel: container
  memory:
      containerQuery: sum(container_memory_working_set_bytes{<<.LabelMatchers>>,container!
="",pod!=""}) by (<<.GroupBy>>)
      nodeQuery: sum(container_memory_working_set_bytes{<<.LabelMatchers>>,id='/'}) by
(<<.GroupBy>>)
      resources:
      overrides:
          instance:
          resource: node
          namespace:
          resource: namespace
          pod:
          resource: pod
      containerLabel: container
  window: 1m
...
```

- **CPU平均利用率采集规则**
  - node_cpu_usage_avg：表示节点的CPU平均利用率，该指标名不可修改。
  - metricsQuery: avg_over_time((1 - avg (irate(<<.Series>>{mode="idle"}[5m])) by (instance))[10m:30s])：为节点CPU平均利用率的查询语句。

    当前metricsQuery表示查询所有节点最近10分钟的CPU平均利用率，如果希望调整平均值的计算周期为最近5分钟或者30分钟，可以修改上述标红的10m为5m或者30m即可。
- **Memory平均利用率采集规则**
  - node_memory_usage_avg：表示节点的Memory利用率，该指标名不可修改。
  - metricsQuery: avg_over_time(((1-node_memory_MemAvailable_bytes/<<.Series>>))[10m:30s])：为节点Memory平均利用率的查询语句。

当前metricsQuery表示查询所有节点最近10分钟的Memory平均利用率，如果希望调整平均值的计算周期为最近5分钟或者30分钟，可以修改上述标红的10m为5m或者30m即可。

3. 重新部署monitoring命名空间下的custom-metrics-apiserver工作负载。
   ```
   kubectl rollout restart deployment custom-metrics-apiserver -n monitoring
   ```
4. 验证自定义规则配置成功。
   a. 执行以下命令，若可以正常返回自定义指标信息，表示Prometheus侧指标采集配置成功。
      ```
      kubectl get --raw=/apis/custom.metrics.k8s.io/v1beta1
      ```
      

   b. 执行以下命令，查询集群内节点信息。
      ```
      kubectl get nodes
      ```
      然后任选一个节点执行以下命令，其中*xxxx*替换为查询到的node_name，如果需要查询所有节点资源信息，可以使用*代替*xxxx*：
      ```
      kubectl get --raw=/apis/custom.metrics.k8s.io/v1beta1/nodes/xxxx/node_cpu_usage_avg
      ```
      查询结果示例如下：
      

步骤3 开启负载感知调度能力。

安装Volcano后，您可通过“配置中心 > 调度配置”选择开启或关闭负载感知调度能力，默认关闭。

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“配置中心”，在右侧选择“调度配置”页签。
3. 在“资源利用率优化调度”配置中，修改负载感知调度配置。

说明

为达到最优的负载感知调度效果，可以选择关闭装箱（binpack）策略。装箱策略（binpack）根据Pod的Request资源信息，将Pod优先调度到资源消耗较多的节点，在一定程度上会影响负载感知调度的效果。多种策略的结合使用案例可参考**资源利用率优化调度配置案例**。

参数	说明	默认值
负载感知调度策略权重	增大该权重值，可提高负载感知策略在整体调度中的影响力。	5
CPU权重	增大该权重值，优先均衡CPU资源。	1
内存权重	增大该权重值，优先均衡内存资源。	1

参数	说明	默认值
真实负载阈值生效方式	– 软约束：节点CPU、内存真实负载达到阈值后，新的任务优先被分配至真实负载未达到阈值的节点，但是该节点依然允许调度。 – 硬约束：节点CPU、内存真实负载达到阈值后，该节点不允许调度新的任务。	硬约束
CPU真实负载阈值	节点CPU真实利用率超过该阈值后，会根据真实负载阈值生效方式中的约束调度工作负载。新下发的工作负载将被优先或强制调度到其他节点，节点中已经运行的工作负载不受影响。	80
内存真实负载阈值	节点内存真实利用率超过该阈值后，会根据真实负载阈值生效方式中的约束调度工作负载。新下发的工作负载将被优先或强制调度到其他节点，节点中已经运行的工作负载不受影响。	80

----结束

6.5.3.5 资源利用率优化调度配置案例

概述

Volcano调度分为两个阶段，分别为节点过滤和节点优选，过滤阶段筛选出符合调度条件的节点，优选阶段对所有符合调度条件的节点打分，最终选取得分最高的节点进行调度。Volcano提供多种调度策略进行节点打分优选，每种调度策略可以根据实际业务场景调整对应的权重值，提高或降低该策略在节点打分过程中的影响性。

节点优选调度策略介绍

Volcano插件支持的节点调度策略如下：

调度策略	参数	说明	使用指导
装箱调度（binpack）	binpack.weight	装箱策略，开启后默认值是 10	装箱调度（Binpack）
兼容kube-scheduler节点排序策略（nodeorder）	nodeaffinity.weight	节点亲和性优先调度，默认值是2。	默认开启
	podaffinity.weight	Pod亲和性优先调度，默认值是2。	
	leastrequested.weight	资源分配最少的节点优先，默认值是1。	
	balancedresource.weight	节点上面的不同资源分配平衡的优先，默认值是1。	

调度策略	参数	说明	使用指导
	mostrequested.weight	资源分配最多的节点优先，默认值是0。	
	tainttoleration.weight	污点容忍高的优先调度，默认值是3。	
	imagelocality.weight	节点上面有Pod需要镜像的优先调度，默认值是1。	
	selectorspread.weight	把Pod均匀调度到不同的节点上，默认值是0。	
	podtopologyspread.weight	Pod拓扑调度，默认值是2。	
numa亲和性调度（numa-aware）	weight	numa亲和性调度，开启后默认值是 1。	**NUMA亲和性调度**
负载感知调度（usage）	weight	负载感知调度，开启后默认值是 5	**负载感知调度**
节点池亲和性调度（nodepoolaffinity）	nodepoolaffinity.weight	节点池亲和调度，开启后默认是 10000	**节点池亲和性调度**

如何减少节点资源碎片，提高集群资源利用率

集群中存在大作业（request资源量较大）和小作业（request资源量较少）混合提交并运行，希望小作业可以优先填满集群各节点的资源碎片，将空闲的节点资源优先预留给大作业运行，避免大作业由于节点资源不足长时间无法调度。

开启**装箱策略（binpack）**，使用默认权重值10。插件详情与配置方法请参见**装箱调度（Binpack）**。

配置建议如下：

- 优先减少集群中的CPU资源碎片：建议提高binpack策略中的CPU权重为5，Memory权重保持为1。
- 优先减少集群中的Memory资源碎片：建议提高binpack策略中的Memory权重为5，CPU权重保持为1。
- 优先减少集群中的GPU资源碎片：建议自定义资源类型（GPU），并设置GPU资源权重为10，CPU权重保持为1，Memory权重保持为1。

如何使节点 CPU、内存的真实负载趋于均衡

工作负载运行过程中，真实消耗的CPU和内存存在大的波动，通过工作负载request资源无法准确评估的场景中，希望调度器可以结合集群内节点CPU、内存的负载情况，将Pod优先调度到负载较低的节点，实现节点负载均衡，避免出现因单个节点负载过高而导致的应用程序或节点故障。

配置案例1

1. 开启**负载感知调度策略**，使用默认权重值5。插件详情与配置方法请参见**负载感知调度**。
2. 关闭**装箱调度策略（binpack）**。插件详情与配置方法请参见**装箱调度（Binpack）**。

配置建议如下：

- 优先确保各节点CPU资源负载趋于均衡：建议提高负载感知调度的CPU权重为5，内存权重保持为1。
- 优先确保各节点的内存资源负载趋于均衡：建议提高负载感知调度的内存权重为5，CPU权重保持为1。
- 真实负载阈值生效方式与CPU真实负载阈值和内存真实负载阈值联合生效：
  - 硬约束场景：
    - 节点CPU真实利用率超过CPU真实负载阈值后，该节点不允许调度新的工作负载。
    - 节点内存真实利用率超过内存真实负载阈值后，该节点不允许调度新的工作负载。
  - 软约束场景：
    - 节点CPU真实利用率超过CPU真实负载阈值后，尽可能不向该节点调度新的工作负载。
    - 节点内存真实利用率超过内存真实负载阈值后，尽可能不向该节点调度新的工作负载。
  - 希望集群内各节点的负载趋于均衡，同时希望尽可能提升集群资源利用率的场景：可以设置真实负载阈值生效方式为软约束，CPU真实负载阈值和内存真实负载阈值使用默认值80。
  - 希望优先确保工作负载的稳定性，降低热点节点CPU、内存压力的场景：可以设置真实负载阈值生效方式为硬约束，CPU真实负载阈值和内存真实负载阈值在60~80之间设置。

配置案例2

随着集群状态，工作负载流量与请求的动态变化，节点的利用率也在实时变化，集群有可能会再次出现负载极端不均衡的情况，在业务Pod允许被驱逐重新调度的场景中，通过负载感知和热点打散重调度结合使用，可以获得集群最佳的负载均衡效果。关于热点打散重调度能力的使用请参见**重调度（Descheduler）**。

1. 开启**负载感知调度策略**，使用默认权重值5。插件详情与配置方法请参见**负载感知调度**。
2. 开启**重调度能力**，完成负载感知重调度策略配置。插件详情与配置方法请参见**重调度（Descheduler）**。
3. 关闭**装箱调度策略（binpack）**。插件详情与配置方法请参见**装箱调度（Binpack）**。

配置建议如下：

- 负载感知重调度策略配置推荐
  - 高负载节点驱逐pod的阈值信息targetThreshold：cpu为75、memory为70。

  - 低负载节点承接pod的阈值信息thresholds：cpu为30、memory为30。
- 负载感知调度的真实负载阈值应介于重调度高负载节点与低负载节点阈值之间
  - CPU真实负载阈值 65
  - 内存真实负载阈值 60

6.5.4 业务优先级保障调度

6.5.4.1 优先级调度与抢占

优先级表示一个作业相对于其他作业的重要性，Volcano兼容Kubernetes中的Pod优先级定义（**PriorityClass**）。启用该能力后，调度器将优先保障高优先级业务调度。集群资源不足时，调度器主动驱逐低优先级业务，保障调度高优先级业务可以正常调度。

前提条件

- 已创建v1.19及以上版本的集群，详情请参见**购买Standard/Turbo集群**。
- 已安装Volcano插件，详情请参见**Volcano调度器**。

优先级调度与抢占介绍

用户在集群中运行的业务丰富多样，包括核心业务、非核心业务，在线业务、离线业务等，根据业务的重要程度和SLA要求，可以对不同业务类型设置相应的高优先级。比如对核心业务和在线业务设置高优先级，可以保证该类业务优先获取集群资源。当集群资源被非核心业务占用，整体资源不足时，如果有新的核心业务提交部署请求，可以通过抢占的方式驱逐部分非核心业务，释放集群资源用于核心业务的调度运行。

CCE集群支持的优先级调度如**表6-11**所示。

表 6-11 业务优先级保障调度

调度类型	说明
基于优先级调度	调度器优先保障高优先级业务运行，但不会主动驱逐已运行的低优先级业务。基于优先级调度配置默认开启，不支持关闭。
基于优先级抢占调度	当集群资源不足时，调度器主动驱逐低优先级业务，保障高优先级业务正常调度。

配置优先级调度与抢占策略

安装Volcano后，您可通过“配置中心 > 调度配置”页面选择开启或关闭优先级抢占调度能力。

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，在右侧选择“调度配置”页签。

步骤3 在“业务优先级保障调度”配置中，进行优先级调度配置。

- 基于优先级调度：调度器优先保障高优先级业务运行，但不会主动驱逐已运行的低优先级业务。基于优先级调度配置默认开启，不支持关闭。
- 基于优先级抢占调度：将Volcano调度器设置为集群默认调度器时，支持基于优先级抢占调度。当集群资源不足时，调度器主动驱逐低优先级业务，保障高优先级业务正常调度。

  说明

  - 开启优先级抢占调度时，不支持使用Pod延迟创建。
  - 优先级抢占暂不支持eni/sub-eni自定义资源、hostPort端口的抢占。

图 6-18 业务优先级保障调度

步骤4 修改完成后，单击“确认配置”。

步骤5 配置完成后，可以在工作负载或Volcano Job中使用优先级定义（**PriorityClass**）进行优先级调度。

1. 创建一个或多个优先级定义（**PriorityClass**）。

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: ""
```

2. 创建工作负载或Volcano Job，并指定priorityClassName。
   - 工作负载

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: high-test
  labels:
    app: high-test
spec:
  replicas: 5
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
```



```
      app: test
  spec:
    priorityClassName: high-priority
    schedulerName: volcano
    containers:
    - name: test
      image: busybox
      imagePullPolicy: IfNotPresent
      command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
      resources:
        requests:
          cpu: 500m
        limits:
          cpu: 500m
```

- Volcano Job

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob
spec:
  schedulerName: volcano
  minAvailable: 4
  priorityClassName: high-priority
  tasks:
    - replicas: 4
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

----结束

基于优先级调度示例

如果集群中存在两个空闲节点，存在3个优先级的工作负载，分别为high-priority，med-priority，low-priority，首先运行high-priority占满集群资源，然后提交med-priority，low-priority的工作负载，由于集群资源全部被更高优先级工作负载占用，med-priority，low-priority的工作负载为pending状态，当high-priority工作负载结束，按照优先级调度原则，med-priority工作负载将优先调度。

步骤1 通过priority.yaml创建3个优先级定义（**PriorityClass**），分别为：high-priority，med-priority，low-priority。

priority.yaml文件内容如下：

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 100
globalDefault: false
description: "This priority class should be used for volcano job only."
---
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: med-priority
value: 50
globalDefault: false
```



```
description: "This priority class should be used for volcano job only."
---
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 10
globalDefault: false
description: "This priority class should be used for volcano job only."
```

创建PriorityClass：

```
kubectl apply -f priority.yaml
```

步骤2 查看优先级定义信息。

```
kubectl get PriorityClass
```

回显如下：

```
NAME                      VALUE       GLOBAL-DEFAULT   AGE
high-priority             100         false            97s
low-priority              10          false            97s
med-priority              50          false            97s
system-cluster-critical   2000000000  false            6d6h
system-node-critical      2000001000  false            6d6h
```

步骤3 创建高优先级工作负载high-priority-job，占用集群的全部资源。

high-priority-job.yaml

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-high
spec:
  schedulerName: volcano
  minAvailable: 4
  priorityClassName: high-priority
  tasks:
    - replicas: 4
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

执行以下命令下发作业：

```
kubectl apply -f high_priority_job.yaml
```

通过 **kubectl get pod** 查看Pod运行信息，如下：

```
NAME                 READY   STATUS    RESTARTS   AGE
priority-high-test-0   1/1     Running   0          3s
priority-high-test-1   1/1     Running   0          3s
priority-high-test-2   1/1     Running   0          3s
priority-high-test-3   1/1     Running   0          3s
```

此时，集群节点资源已全部被占用。

步骤4 创建中优先级工作负载med-priority-job和低优先级工作负载low-priority-job。

med-priority-job.yaml


```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-medium
spec:
  schedulerName: volcano
  minAvailable: 4
  priorityClassName: med-priority
  tasks:
    - replicas: 4
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

low-priority-job.yaml

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: priority-low
spec:
  schedulerName: volcano
  minAvailable: 4
  priorityClassName: low-priority
  tasks:
    - replicas: 4
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

执行以下命令下发作业：

```
kubectl apply -f med_priority_job.yaml
kubectl apply -f low_priority_job.yaml
```

通过 **kubectl get pod** 查看Pod运行信息，集群资源不足，Pod处于Pending状态，如下：

```
NAME                   READY   STATUS    RESTARTS   AGE
priority-high-test-0     1/1     Running   0          3m29s
priority-high-test-1     1/1     Running   0          3m29s
priority-high-test-2     1/1     Running   0          3m29s
priority-high-test-3     1/1     Running   0          3m29s
priority-low-test-0      0/1     Pending   0          2m26s
priority-low-test-1      0/1     Pending   0          2m26s
priority-low-test-2      0/1     Pending   0          2m26s
priority-low-test-3      0/1     Pending   0          2m26s
priority-medium-test-0   0/1     Pending   0          2m36s
priority-medium-test-1   0/1     Pending   0          2m36s
priority-medium-test-2   0/1     Pending   0          2m36s
priority-medium-test-3   0/1     Pending   0          2m36s
```


步骤5 删除high_priority_job工作负载，释放集群资源，med_priority_job会被优先调度。

执行 **kubectl delete -f high_priority_job.yaml** 释放集群资源，查看Pod的调度信息，如下：

```
NAME                    READY   STATUS    RESTARTS   AGE
priority-low-test-0     0/1     Pending   0          5m18s
priority-low-test-1     0/1     Pending   0          5m18s
priority-low-test-2     0/1     Pending   0          5m18s
priority-low-test-3     0/1     Pending   0          5m18s
priority-medium-test-0  1/1     Running   0          5m28s
priority-medium-test-1  1/1     Running   0          5m28s
priority-medium-test-2  1/1     Running   0          5m28s
priority-medium-test-3  1/1     Running   0          5m28s
```

----结束

基于优先级抢占调度示例

步骤1 登录CCE控制台，进入“配置中心 > 调度配置”页面。

步骤2 修改以下配置并确认。

1. 设置集群默认调度器：选择“Volcano调度器”。
2. 业务优先级保障调度：选择开启“基于优先级抢占调度”能力。

步骤3 在基于优先级调度的场景下，再次下发high_priority_job工作负载，则调度器会驱逐med_priority_job工作负载，保证high_priority_job可以成功调度。

执行 **kubectl apply -f high_priority_job.yaml**，作业下发成功，查看Pod状态信息，如下：

```
NAME                    READY   STATUS        RESTARTS   AGE
priority-high-test-0    0/1     Pending       0          2s
priority-high-test-1    0/1     Pending       0          2s
priority-high-test-2    0/1     Pending       0          2s
priority-high-test-3    0/1     Pending       0          2s
priority-low-test-0     0/1     Pending       0          14s
priority-low-test-1     0/1     Pending       0          14s
priority-low-test-2     0/1     Pending       0          14s
priority-low-test-3     0/1     Pending       0          14s
priority-medium-test-0  1/1     Terminating   0          21s
priority-medium-test-1  1/1     Terminating   0          21s
priority-medium-test-2  1/1     Terminating   0          21s
priority-medium-test-3  1/1     Terminating   0          21s
```

等待med_priority_job资源释放成功后，high_priority_job成功调度，如下：

```
NAME                    READY   STATUS    RESTARTS   AGE
priority-high-test-0    1/1     Running   0          70s
priority-high-test-1    1/1     Running   0          70s
priority-high-test-2    1/1     Running   0          70s
priority-high-test-3    1/1     Running   0          70s
priority-low-test-0     0/1     Pending   0          82s
priority-low-test-1     0/1     Pending   0          82s
priority-low-test-2     0/1     Pending   0          82s
priority-low-test-3     0/1     Pending   0          82s
priority-medium-test-0  0/1     Pending   0          37s
priority-medium-test-1  0/1     Pending   0          36s
priority-medium-test-2  0/1     Pending   0          37s
priority-medium-test-3  0/1     Pending   0          37s
```

在节点资源无法满足high_priority_job的情况下，volcano-scheduler的优先级抢占机制将被启用，驱逐med_priority_job后，将high_priority_job部署到节点上。在Cluster Autoscaler新扩容节点后，volcano-scheduler再将med_priority_job调度到新节点上。

根据上述结果，在启用优先级抢占调度时，建议您开启节点弹性，以保证集群资源的按需供给，进而保证应用SLA。

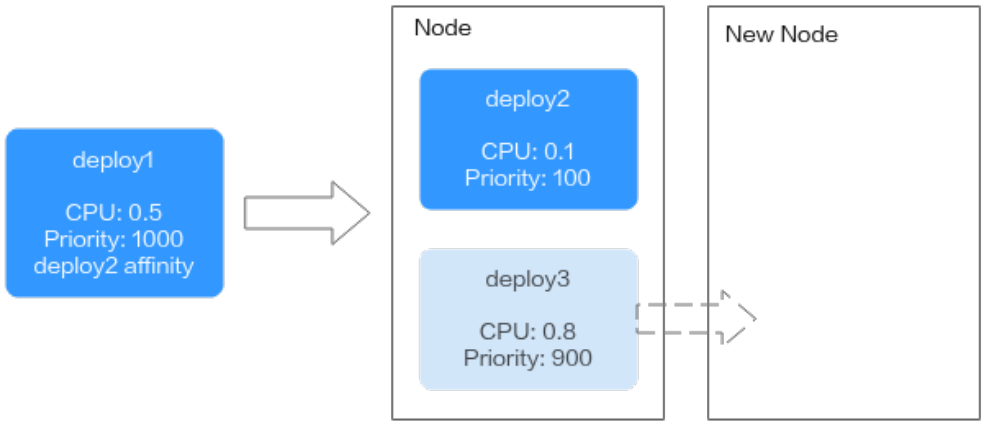
----结束

基于优先级抢占调度的亲和/反亲和示例

在Pod间亲和场景中，不推荐Pod与比其优先级低的Pod亲和。如果pending状态的Pod与节点上的一个或多个较低优先级Pod具有Pod间亲和性，对较低优先级的Pod发起抢占时，会无法满足Pod间亲和性规则，抢占规则和亲和性规则产生矛盾。 在这种情况下，调度程序无法保证pending状态的Pod可以被调度。推荐的解决方案是仅针对同等或更高优先级的Pod设置Pod间亲和性。详情请参见**与低优先级Pod之间的Pod间亲和性**。

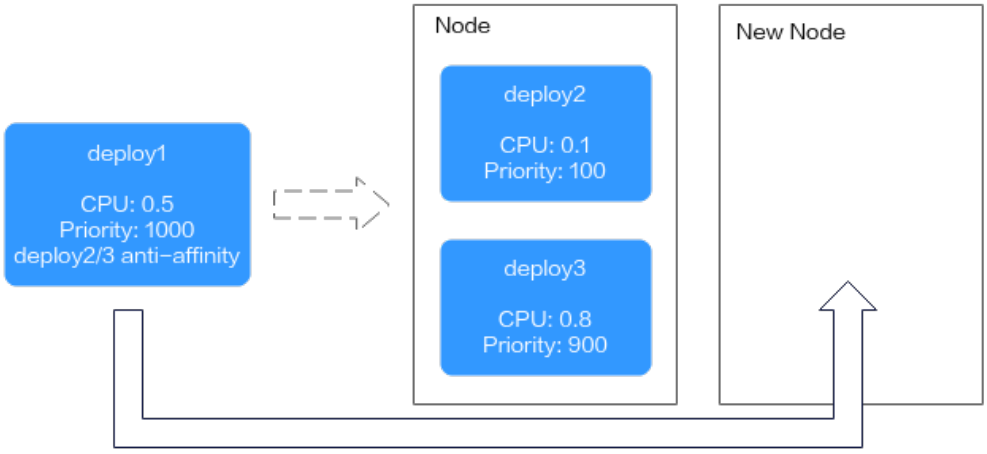
在Pod间亲和场景中，如果启用优先级抢占，当deploy1与比其优先级低的deploy2亲和，volcano-scheduler为保证业务自运维，将驱逐deploy3，并将deploy1调度到节点上。被驱逐的deploy3将会在新节点准备好后，调度到新节点上。

图 6-19 与低优先级的 Pod 亲和场景



在Pod间反亲和场景中，如果启用优先级抢占，当deploy1与deploy2/3反亲和，volcano-scheduler为减少对其它业务的影响，将不驱逐deploy2和deploy3，而是在新节点准备好后，将deploy1调度到新节点上。

图 6-20 与低优先级 Pod 反亲和场景



6.5.5 AI 任务性能增强调度

6.5.5.1 公平调度（DRF）

DRF（Dominant Resource Fairness）是主资源公平调度策略，应用于大批量提交AI训练和大数据作业的场景，可增强集群业务的吞吐量，整体缩短业务执行时间，提高训练性能。

前提条件

- 已创建v1.19及以上版本的集群，详情请参见**购买Standard/Turbo集群**。
- 已安装Volcano插件，详情请参见**Volcano调度器**。

公平调度介绍

在实际业务中，经常会遇到将集群稀缺资源分配给多个用户的情况，每个用户获得资源的权利都相同，但是需求数却可能不同，如何公平的将资源分配给每个用户是一项非常有意义的事情。调度层面有一种常用的方法为最大最小化公平分配算法（max-min fairness share），尽量满足用户中的最小的需求，然后将剩余的资源公平分配给剩下的用户。形式化定义如下：

1. 资源分配以需求递增的方式进行分配
2. 每个用户获得的资源不超过其需求
3. 未得到满足的用户等价平分剩下的资源

max-min fairness算法的最大问题是认为资源是单一的，但是现实情况中资源却不是单一的，例如CPU、Memory、GPU等资源在分配时都需要考虑。这个时候DRF应运而生，简单来说DRF就是 max-min fairness 算法的泛化版本，可以支持多种类型资源的公平分配， 即每个用户的主资源满足 max-min fairness 要求。

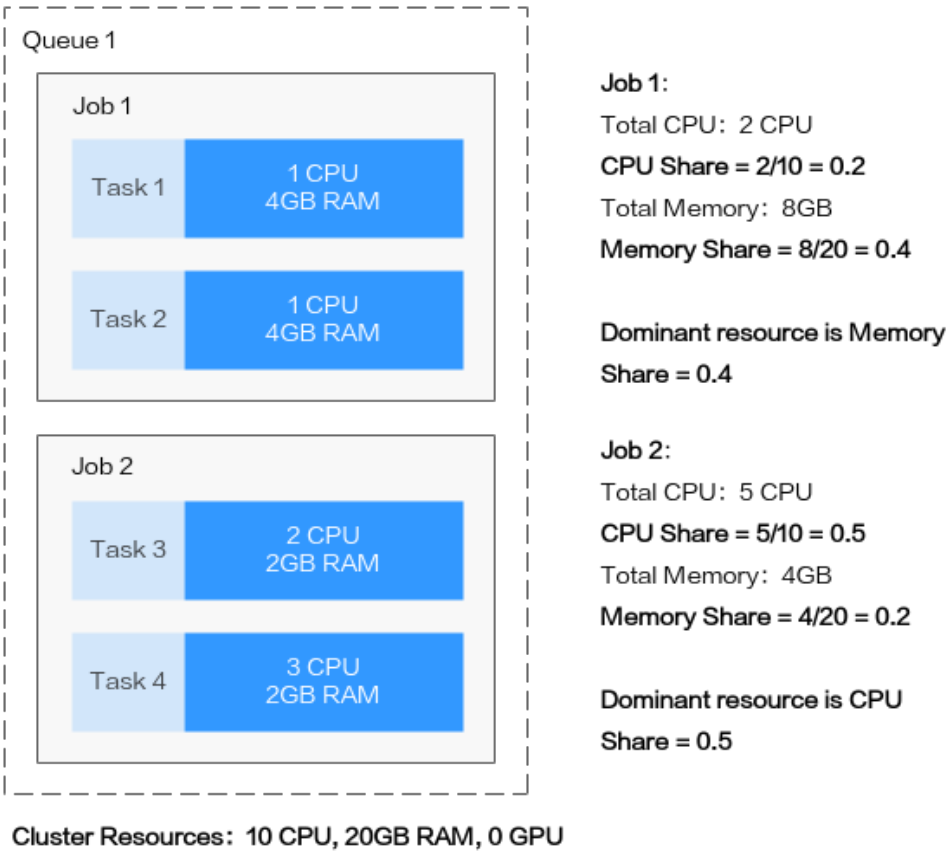
每个Job资源的Share值计算如下：

Share = Total Request / Cluster Resources

当Job具有多个资源时，将Share值最大的资源作为主资源，在进行优先级调度时，仅根据主资源的Share值进行优先级调度。

例如，Job 1和Job 2分别为两个工作负载，其请求的资源量如图所示， 通过DRF计算之后，Job 1的主资源为Memory，对应的Share值为0.4，Job 2的主资源为CPU，对应的Share值为0.5，根据Share值对比，Job 1的资源请求量小于Job 2，按照最大最小公平算法分配策略，Job 1的优先级高于Job 2。

图 6-21 DRF 调度示意图



配置公平调度策略

安装Volcano后，您可通过“配置中心 > 调度配置”选择开启或关闭DRF调度能力，默认开启。

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，在右侧选择“调度配置”页签。

步骤3 在“AI任务性能增强调度”配置中，选择是否开启“公平调度 (drf) ”。

启用该能力后，可增强集群业务的吞吐量，提高业务运行性能。

步骤4 修改完成后，单击“确认配置”。

----结束

6.5.5.2 组调度（Gang）

组调度（Gang）满足了调度过程中“All or nothing”的调度需求，避免Pod的任意调度导致集群资源的浪费，主要应用于AI、大数据等多任务协作场景。启用该能力后，可以解决分布式训练任务之间的资源忙等待和死锁等痛点问题，大幅度提升整体训练性能。

前提条件

- 已创建v1.19及以上版本的集群，详情请参见**购买Standard/Turbo集群**。
- 已安装Volcano插件，详情请参见**Volcano调度器**。

组调度介绍

Gang调度策略是volcano-scheduler的核心调度算法之一，它满足了调度过程中的“All or nothing”的调度需求，避免Pod的任意调度导致集群资源的浪费。具体算法是，观察Job下的Pod已调度数量是否满足了最小运行数量，当Job的最小运行数量得到满足时，为Job下的所有Pod执行调度动作，否则，不执行。

基于容器组概念的Gang调度算法十分适合需要多进程协作的场景。AI场景往往包含复杂的流程，Data Ingestion、Data Analysts、Data Splitting、Trainer、Serving、Logging等，需要一组容器进行协同工作，就很适合基于容器组的Gang调度策略。MPI计算框架下的多线程并行计算通信场景，由于需要主从进程协同工作，也非常适合使用Gang调度策略。容器组下的容器高度相关也可能存在资源争抢，整体调度分配，能够有效解决死锁。在集群资源不足的场景下，Gang的调度策略对于集群资源的利用率的提升是非常明显的。

配置组调度策略

安装Volcano后，您可通过“配置中心 > 调度配置”选择开启或关闭Gang调度能力，默认开启。

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，在右侧选择“调度配置”页签。

步骤3 在“AI任务性能增强调度”配置中，选择是否开启“组调度 (Gang) ”。

启用该能力后，可增强集群业务的吞吐量，提高业务运行性能。

步骤4 修改完成后，单击“确认配置”。

步骤5 配置完成后，可以在工作负载或Volcano Job中使用Gang调度能力。

- 创建工作负载使用Gang调度能力
  a. 首先创建PodGroup，需指定minMember和minResources信息如下：

```
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
metadata:
  name: pg-test1
spec:
  minMember: 3
  minResources:
    cpu: 3
    memory: 3Gi
```

  - minMember：归属于当前PodGroup的一组Pod满足minMember数量时，才会被统一调度。

- minResources：集群空闲资源满足minResources要求时，该组Pod才会被统一调度。

b. 创建工作负载时，通过schedulerName指定Volcano调度器，并通过annotation指定其归属的PodGroup，如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podgroup-test
  labels:
    app: podgroup-test
spec:
  replicas: 6
  selector:
    matchLabels:
      app: podgroup-test
  template:
    metadata:
      annotations:
        scheduling.k8s.io/group-name: pg-test1
      labels:
        app: podgroup-test
    spec:
      schedulerName: volcano
      containers:
      - name: test
        image: busybox
        imagePullPolicy: IfNotPresent
        command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
        resources:
          requests:
            cpu: 500m
          limits:
            cpu: 500m
```

- schedulerName：设置为volcano，表示使用Volcano调度该工作负载。
- scheduling.k8s.io/group-name：指定上一步中创建的PodGroup，示例为pg-test1。

- 创建Volcano Job使用Gang调度能力

  创建Volcano Job时，仅需要指定minAvailable数量和schedulerName为volcano即可，Volcano调度器会自动创建并管理PodGroup，示例如下：

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob
spec:
  schedulerName: volcano
  minAvailable: 2
  tasks:
    - replicas: 4
      name: "test"
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                requests:
                  cpu: "1"
          restartPolicy: OnFailure
```

----结束

6.5.6 NUMA 亲和性调度

NUMA节点是Non-Uniform Memory Access（非统一内存访问）架构中的一个基本组成单元，每个节点包含自己的处理器和本地内存，这些节点在物理上彼此独立，但通过高速互连总线连接在一起，形成一个整体系统。NUMA节点能够通过提供更快的本地内存访问来提高系统性能，但通常一个Node节点是多个NUMA节点的集合，在多个NUMA节点之间进行内存访问时会产生延迟，开发者可以通过优化任务调度和内存分配策略，来提高内存访问效率和整体性能。

在云原生环境中，对于高性能计算（HPC）、实时应用和内存密集型工作负载等需要CPU间通信频繁的场景下，跨NUMA节点访问会导致增加延迟和开销，从而降低系统性能。为此，volcano提供了NUMA亲和性调度能力，尽可能把Pod调度到需要跨NUMA节点最少的工作节点上，这种调度策略能够降低数据传输开销，优化资源利用率，从而增强系统的整体性能。

更多资料请查看社区NUMA亲和性插件指导链接：**https://github.com/volcano-sh/volcano/blob/master/docs/design/numa-aware.md**

前提条件

- 已创建一个CCE Standard集群或CCE Turbo集群，详情请参见**购买Standard/Turbo集群**。
- 集群中已安装Volcano插件，详情请参见**Volcano调度器**。

Pod 调度行为说明

当Pod设置了拓扑策略时，Volcano会根据Pod设置的拓扑策略预测匹配的节点列表。Pod的拓扑策略配置请参考**NUMA亲和性调度使用示例**。调度过程如下：

1. 根据Pod设置的Volcano拓扑策略，筛选具有相同策略的节点。Volcano提供的拓扑策略与**拓扑管理器**相同。
2. 在设置了相同策略的节点中，筛选CPU拓扑满足该策略要求的节点进行调度。

Pod可配置的拓扑策略	Pod调度时筛选节点行为说明	
	1.根据Pod设置的拓扑策略，筛选可调度的节点	2.筛选可调度的节点后，进一步筛选CPU拓扑满足策略的节点进行调度
none	针对配置了以下几种拓扑策略的节点，调度时均无筛选行为： • none：可调度 • best-effort：可调度 • restricted：可调度 • single-numa-node：可调度	-

Pod可配置的拓扑策略	Pod调度时筛选节点行为说明	
	1.根据Pod设置的拓扑策略，筛选可调度的节点	**2.筛选可调度的节点后，进一步筛选CPU拓扑满足策略的节点进行调度**
best-effort	筛选拓扑策略同样为“best-effort”的节点： ● none：不可调度 ● **best-effort：可调度** ● restricted：不可调度 ● single-numa-node：不可调度	尽可能满足策略要求进行调度： 优先调度至单NUMA节点，如果单NUMA节点无法满足CPU申请值，允许调度至多个NUMA节点。
restricted	筛选拓扑策略同样为“restricted”的节点： ● none：不可调度 ● best-effort：不可调度 ● **restricted：可调度** ● single-numa-node：不可调度	严格限制的调度策略： ● 单NUMA节点的CPU容量上限大于等于CPU的申请值时，仅允许调度至单NUMA节点。此时如果单NUMA节点剩余的CPU可使用量不足，则Pod无法调度。 ● 单NUMA节点的CPU容量上限小于CPU的申请值时，可允许调度至多个NUMA节点。
single-numa-node	筛选拓扑策略同样为“single-numa-node”的节点： ● none：不可调度 ● best-effort：不可调度 ● restricted：不可调度 ● **single-numa-node：可调度**	仅允许调度至单NUMA节点。

假设单个节点CPU总量为32U，由2个NUMA节点提供资源，分配如下：

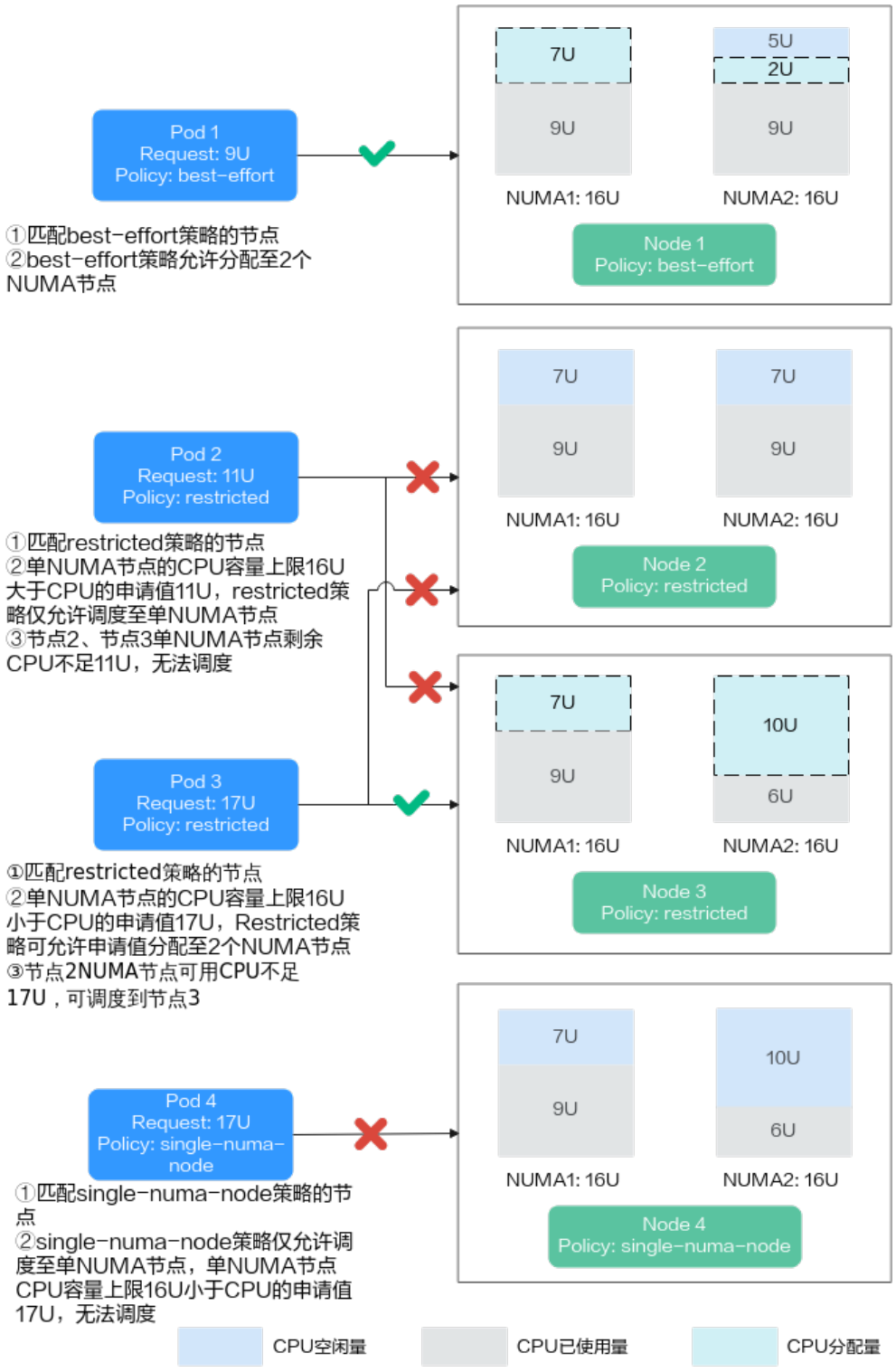
工作节点	节点拓扑策略	NUMA节点1上的CPU总量		NUMA节点2上的CPU总量	
		CPU总量	CPU空闲量	CPU总量	CPU空闲量
节点-1	best-effort	16U	7U	16U	7U
节点-2	restricted	16U	7U	16U	7U
节点-3	restricted	16U	7U	16U	10U

工作节点	节点拓扑策略	NUMA节点1上的CPU总量		NUMA节点2上的CPU总量	
		CPU总量	CPU空闲量	CPU总量	CPU空闲量
节点-4	single-numa-node	16U	7U	16U	10U

Pod设置拓扑策略后，调度情况如图6-22所示。

- 当Pod的CPU申请值为9U时，设置拓扑策略为“best-effort”，Volcano会匹配拓扑策略同样为“best-effort”的节点-1，且该策略允许调度至多个NUMA节点，因此9U的申请值会被分配到2个NUMA节点，该Pod可成功调度至节点-1。
- 当Pod的CPU申请值为11U时，设置拓扑策略为“restricted”，Volcano会匹配拓扑策略同样为“restricted”的节点-2/节点-3，且单NUMA节点CPU总量满足11U的申请值，但单NUMA节点剩余可用的CPU量无法满足，因此该Pod无法调度。
- 当Pod的CPU申请值为17U时，设置拓扑策略为“restricted”，Volcano会匹配拓扑策略同样为“restricted”的节点-2/节点-3，且单NUMA节点CPU总量无法满足17U的申请值，可允许分配到2个NUMA节点，该Pod可成功调度至节点-3。
- 当Pod的CPU申请值为17U时，设置拓扑策略为“single-numa-node”，Volcano会匹配拓扑策略同样为“single-numa-node”的节点，但由于单NUMA节点CPU总量均无法满足17U的申请值，因此该Pod无法调度。

图 6-22 NUMA 调度策略对比



调度优先级

不管是什么拓扑策略，都是希望把Pod调度到当时最优的节点上，这里通过给每一个节点进行打分的机制来排序筛选最优节点。

原则：尽可能把Pod调度到需要跨NUMA节点最少的工作节点上。

打分公式如下：

score = weight * (100 - 100 * numaNodeNum / maxNumaNodeNum)

参数说明：

- **weight**：NUMA Aware Plugin的权重。
- **numaNodeNum**：表示工作节点上运行该Pod需要NUMA节点的个数。
- **maxNumaNodeNum**：表示所有工作节点中该Pod的最大NUMA节点个数。

例如，假设有三个节点满足Pod的CPU拓扑策略，且NUMA Aware Plugin的权重设为10：

- Node A：由1个NUMA节点提供Pod所需的CPU资源，即numaNodeNum=1
- Node B：由2个NUMA节点提供Pod所需的CPU资源，即numaNodeNum=2
- Node C：由4个NUMA节点提供Pod所需的CPU资源，即numaNodeNum=4

则根据以上公式，maxNumaNodeNum=4

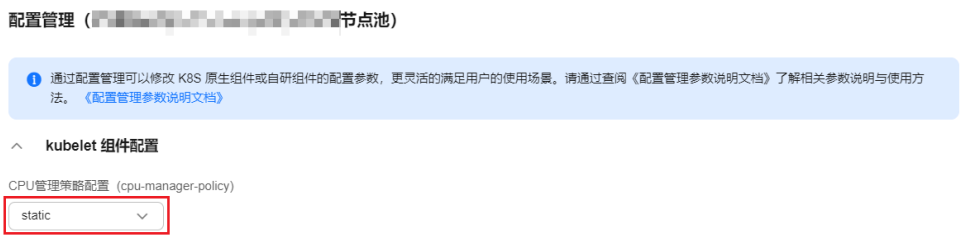
- score(Node A) = 10 * (100 - 100 * 1 / 4) = 750
- score(Node B) = 10 * (100 - 100 * 2 / 4) = 500
- score(Node C) = 10 * (100 - 100 * 4 / 4) = 0

因此最优节点为Node A。

Volcano 开启 NUMA 亲和性调度

步骤1 在节点池中开启静态（static）CPU管理策略，具体请参考 **为自定义节点池开启CPU管理策略**。

1. 登录CCE控制台，单击集群名称进入集群。
2. 在左侧选择“节点管理”，在右侧选择“节点池”页签，单击节点池名称后的“更多 > 配置管理”。
3. 在侧边栏滑出的“配置管理”窗口中，修改kubelet组件的CPU管理策略配置（cpu-manager-policy）参数值，选择**static**。

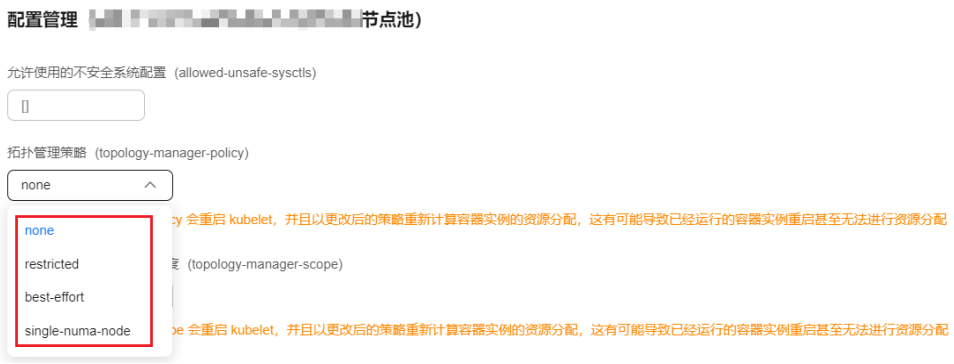


4. 单击“确定”，完成配置操作。

步骤2 在节点池中配置CPU拓扑策略。

1. 登录CCE控制台，单击集群名称进入集群，在左侧选择“节点管理”，在右侧选择“节点池”页签，单击节点池名称后的“ 配置管理”。
2. 将kubelet的**拓扑管理策略（topology-manager-policy）**的值修改为需要的CPU拓扑策略即可。

   有效拓扑策略为“none”、“best-effort”、“restricted”、“single-numa-node”，具体策略对应的调度行为请参见**Pod调度行为说明**。



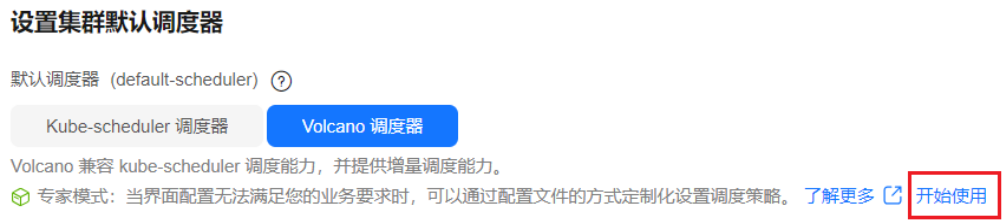
步骤3 开启numa-aware插件功能和resource_exporter功能。

Volcano 1.7.1及以上版本

1. 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到Volcano，单击“编辑”。
2. 在“扩展功能”中开启“NUMA拓扑调度”能力，单击“确定”。

Volcano 1.7.1以下版本

1. 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“配置中心”，切换至“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



2. 开启resource_exporter_enable参数，用于收集节点numa拓扑信息。JSON格式的示例如下：

```
{
  "plugins": {
    "eas_service": {
      "availability_zone_id": "",
      "driver_id": "",
      "enable": "false",
      "endpoint": "",
      "flavor_id": "",
      "network_type": "",
      "network_virtual_subnet_id": "",
      "pool_id": "",
      "project_id": "",
      "secret_name": "eas-service-secret"
    }
  },
  "resource_exporter_enable": "true"
}
```

开启后可以查看当前节点的numa拓扑信息。

```
kubectl get numatopo
NAME          AGE
node-1        4h8m
node-2        4h8m
node-3        4h8m
```


3. 启用Volcano numa-aware算法插件。

   kubectl edit cm -n kube-system volcano-scheduler-configmap

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: volcano-scheduler-configmap
  namespace: kube-system
data:
  default-scheduler.conf: |-
    actions: "allocate, backfill, preempt"
    tiers:
    - plugins:
      - name: priority
      - name: gang
      - name: conformance
    - plugins:
      - name: overcommit
      - name: drf
      - name: predicates
      - name: nodeorder
    - plugins:
      - name: cce-gpu-topology-predicate
      - name: cce-gpu-topology-priority
      - name: cce-gpu
    - plugins:
      - name: nodelocalvolume
      - name: nodeemptydirvolume
      - name: nodeCSIscheduling
      - name: networkresource
        arguments:
          NetworkType: vpc-router
      - name: numa-aware # add it to enable numa-aware plugin
        arguments:
          weight: 10 # the weight of the NUMA Aware Plugin
```

----结束

NUMA 亲和性调度使用示例

Pod调度时可以采用的NUMA放置策略，具体策略对应的调度行为请参见**Pod调度行为说明**。

- single-numa-node：Pod调度时会选择拓扑管理策略已经设置为single-numa-node的节点池中的节点，且CPU需要放置在相同NUMA下，如果节点池中没有满足条件的节点，Pod将无法被调度。
- restricted：Pod调度时会选择拓扑管理策略已经设置为restricted节点池的节点，且CPU需要放置在相同的NUMA集合下，如果节点池中没有满足条件的节点，Pod将无法被调度。
- best-effort：Pod调度时会选择拓扑管理策略已经设置为best-effort节点池的节点，且尽量将CPU放置在相同NUMA下，如果没有节点满足这一条件，则选择最优节点进行放置。

步骤1 以下为使用Volcano设置NUMA亲和性调度的示例。

1. 示例一：在无状态工作负载中配置NUMA亲和性。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: numa-tset
spec:
  replicas: 1
  selector:
    matchLabels:
```



```
    app: numa-tset
template:
  metadata:
    labels:
      app: numa-tset
    annotations:
      volcano.sh/numa-topology-policy: single-numa-node    # set the topology policy
  spec:
    containers:
      - name: container-1
        image: nginx:alpine
        resources:
          requests:
            cpu: 2           # 必须为整数，且需要与limits中一致
            memory: 2048Mi
          limits:
            cpu: 2           # 必须为整数，且需要与requests中一致
            memory: 2048Mi
    imagePullSecrets:
    - name: default-secret
```

2. 示例二：创建一个Volcano Job，并使用NUMA亲和性。

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vj-test
spec:
  schedulerName: volcano
  minAvailable: 1
  tasks:
    - replicas: 1
      name: "test"
      topologyPolicy: best-effort   # set the topology policy for task
      template:
        spec:
          containers:
            - image: alpine
              command: ["/bin/sh", "-c", "sleep 1000"]
              imagePullPolicy: IfNotPresent
              name: running
              resources:
                limits:
                  cpu: 20
                  memory: "100Mi"
          restartPolicy: OnFailure
```

步骤2 NUMA调度分析。

假设NUMA节点情况如下：

工作节点	节点策略拓扑管理器策略	NUMA 节点 0 上的可分配 CPU	NUMA 节点 1 上的可分配 CPU
node-1	single-numa-node	16U	16U
node-2	best-effort	16U	16U
node-3	best-effort	20U	20U

则根据以上示例，

- 示例一中，Pod的CPU申请值为2U，设置拓扑策略为“single-numa-node”，因此会被调度到相同策略的node-1。

- 示例二中，Pod的CPU申请值为20U，设置拓扑策略为“best-effort”，它将被调度到node-3，因为node-3可以在单个NUMA节点上分配Pod的CPU请求，而node-2需要在两个NUMA节点上执行此操作。

----结束

确认 NUMA 使用情况

您可以通过**lscpu**命令查看当前节点的CPU概况：

```
# 查看当前节点的CPU概况
lscpu
...
CPU(s):              32
NUMA node(s):        2
NUMA node0 CPU(s):   0-15
NUMA node1 CPU(s):   16-31
```

然后查看NUMA节点使用情况。

```
# 查看当前节点的CPU分配
cat /var/lib/kubelet/cpu_manager_state
{"policyName":"static","defaultCpuSet":"0,10-15,25-31","entries":{"777870b5-c64f-42f5-9296-688b9dc212ba":{"container-1":"16-24"},"fb15e10a-b6a5-4aaa-8fcd-76c1aa64e6fd":{"container-1":"1-9"}},"checksum":318470969}
```

以上示例中表示，节点上运行了两个容器，一个占用了NUMA node0的1-9核，另一个占用了NUMA node1的16-24核。

常见问题

Pod调度失败

在使用过程中，如果只开启了Volcano插件的NUMA开关，没有配置CPU管理策略，且调度器为volcano时，可能导致作业调度失败，请根据以下要点进行问题排查。

- 在使用NUMA亲和性调度前，请保证已部署Volcano插件且插件运行状态正常。
- 在使用NUMA亲和性调度时：
  a. 请保证节点池的“CPU管理策略配置（cpu-manager-policy）”已设置为**static**。
  b. 请保证节点池的“拓扑管理策略（topology-manager-policy）”已设置正确。
  c. 请保证为Pod设置正确的拓扑策略，来筛选节点池中已配置了相同拓扑策略的节点，具体设置参考**NUMA亲和性调度使用示例**。
  d. 请保证应用Pod使用的是Volcano调度器，具体配置可参考**使用Volcano调度工作负载**；Pod中的所有容器的CPU Request必须为整数（单位：Core），且Request与Limit相同。

6.5.7 应用扩缩容优先级策略

通过应用扩缩容优先级策略，您可以精细调整Pod在不同类型节点上的扩容和缩容顺序，实现资源管理的最优化。在使用默认扩缩容优先级策略的情况下，扩容过程中Pod优先被调度到包周期的节点，其次被调度到按需计费的节点，最后被调度到virtual-kubelet节点（弹性至CCI）；缩容过程中优先删除virtual-kubelet节点（弹性至CCI）的Pod，其次删除按需计费节点上的Pod，最后删除包周期节点上的Pod。

应用扩缩容优先级策略包括两个方面：

- 针对扩容：集群中新建的Pod，Volcano会按照设定的节点优先级进行调度。
- 针对缩容：指定工作负载时，Volcano会按照设定的节点优先级对其进行打分，用于缩容时决定Pod删除顺序。

约束与限制

- 集群版本为v1.23.11及以上、v1.25.6及以上、v1.27.3及以上，以及其他更高版本集群。
- 集群中需安装Volcano调度器插件（1.12.1及以上版本，并开启应用扩缩容优先级策略开关）。
- 当前扩缩容优先级功能默认支持对Deployment（也间接包括**ReplicaSet**）生效。若希望对第三方工作负载生效，可以通过高级配置指定，详情请参见**配置第三方工作负载应用扩缩容优先级策略**。
- 若要使用扩容调度优先级策略，需要将工作负载的spec.schedulerName设置成volcano或者将集群默认调度器设置成volcano。目前对于没有设置资源Requests和Limits属性的工作负载，扩容优先级功能不生效。
- 以使用默认优先级策略为例，调度器在调度工作负载时会按照包周期节点 > 按需节点 > virtual-kubelet节点（弹性至CCI）的优先级进行调度。但由于调度器会从多个维度对调度结果进行考虑，本优先级策略只是其中一个维度，因此无法百分百确保完全实现上述优先级。
- Volcano调度器存在调度性能和调度结果之间的权衡。当集群存在大量可调度节点时，Volcano出于调度性能的考虑会只选择其中一部分节点来进行调度选择，不会计算全局调度最优解，详情请参见**社区文档链接**。因此该行为与本优先级策略存在冲突，您可以**调整Volcano可调度的节点比例**，来使得Volcano选择所有节点进行调度。

应用扩缩容优先级策略介绍

开启应用扩缩容优先级策略，将会在集群中新增两类CRD资源，分别为Balancer与BalancerPolicyTemplate，并创建默认的扩缩容优先级策略，详情请参见**默认应用扩缩容优先级策略**。Volcano插件根据BalancerPolicyTemplate来获取各个节点的优先级，以控制应用扩容时Pod调度的优先级，同时volcano插件基于Balancer和BalancerPolicyTemplate来设置应用缩容时的优先级。

- BalancerPolicyTemplate CRD资源用来进行优先级策略定义。以默认扩缩容优先级策略为例，默认BalancerPolicyTemplate CR资源将会把包周期节点的优先级设置为最高，按需计费节点次之，virtual-kubelet节点（弹性至CCI）设置为最低。

  BalancerPolicyTemplate CR资源不支持更新操作。
- Balancer CRD资源用来申明扩缩容优先级的作用范围。创建Balancer CR资源时，可以指定某个命名空间下的工作负载作为生效范围，也可以具体指定某个Deployment或者某个ReplicaSet工作负载作为生效范围。

一个Balancer CR资源对应一个BalancerPolicyTemplate CR资源，两者结合共同申明哪些工作负载使用了哪些优先级策略。

插件默认的扩缩容优先级策略通过BalancerPolicyTemplate对象将包周期节点、按需计费节点、virtual-kubelet节点（弹性至CCI）三种类型分成不同的优先级，扩容时，volcano在调度新增Pod时将会考虑这些优先级，优先将Pod调度到优先级高的包周期节点上。

通过Balancer和BalancerPolicyTemplate对象，Volcano插件对Balancer作用范围内的Pod，根据BalancerPolicyTemplate对象定义的优先级分别打上以下注解：

- openvessel.io/workload-balancer-score：表示Pod的分值，对于高优先级节点上的Pod，其对应的分值也相对较大。
- autoscaling.volcano.sh/dominated-by-balancer：表示当前Pod受哪个Balancer对象控制，缩容时会优先缩容分值低的Pod。

说明

如果您原先的Pod已经存在社区支持的**controller.kubernetes.io/pod-deletion-cost**注解，那么缩容时将会按照该值定义的优先级来进行缩容。当两个Pod对应controller.kubernetes.io/pod-deletion-cost值相同时，才会按照openvessel.io/workload-balancer-score注解定义的优先级进行缩容。

您可以通过高级配置中的“workload_balancer_score_annotation_key”参数来指定存放Pod分值的注解key，详情请参见**配置第三方工作负载应用扩缩容优先级策略**。

配置应用扩缩容优先级策略

步骤1 开启应用扩缩容优先级策略开关并成功安装Volcano插件后，会在集群中创建默认扩缩容优先级策略。

1. 获取默认Balancer CR资源。

```
# kubectl get balancer default-balancer -oyaml

apiVersion: autoscaling.volcano.sh/v1alpha1
kind: Balancer
metadata:
  name: default-balancer
spec:
  balancerPolicyTemplateName: default-balancerpolicytemplate
  targets:
  - namespaceSelector:
      matchExpressions:
        - key: kubernetes.io/metadata.name
          operator: Exists
  weight: 10
```

2. 获取默认BalancerPolicyTemplate CR资源。

```
# kubectl get balancerpolicytemplate default-balancerpolicytemplate -oyaml

apiVersion: autoscaling.volcano.sh/v1alpha1
kind: BalancerPolicyTemplate
metadata:
  name: default-balancerpolicytemplate
spec:
  policy:
    policyName: Priority
    priorities:
      priorityGroups:
      - priority: 10
        requirements:
        - key: node.cce.io/billing-mode
          operator: In
          values:
          - post-paid
      - priority: 100
        requirements:
        - key: node.cce.io/billing-mode
          operator: In
          values:
          - pre-paid
      - priority: 1
        requirements:
        - key: kubernetes.io/role
          operator: In
          values:
          - virtual-kubelet
          - bursting
```


具体参数含义请参见**默认应用扩缩容优先级策略**。

步骤2 部署工作负载，设定实例数为1。

当前应用的Pod将会优先调度到包周期节点上。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: balancer-test
  namespace: default
  labels:
    virtual-kubelet.io/burst-to-cci: 'auto'  #如果集群资源不足时，支持将Pod部署到CCI集群
spec:
  replicas: 1
  selector:
    matchLabels:
      app: balancer-test
  template:
    metadata:
      labels:
        app: balancer-test
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: container-1
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
      schedulerName: volcano
```

步骤3 调整工作负载实例数为5。

当前应用的Pod将会优先调度到包周期节点上。在包周期节点资源不足情况下，优先调度到按需计费节点上。在按需计费节点资源不足情况下，优先调度到virtual-kubelet节点（弹性至CCI）。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: balancer-test
  namespace: default
  labels:
    virtual-kubelet.io/burst-to-cci: 'auto'  #如果集群资源不足时，支持将Pod部署到CCI集群
spec:
  replicas: 5
  selector:
    matchLabels:
      app: balancer-test
  template:
    metadata:
      labels:
        app: balancer-test
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: container-1
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
```



```
        memory: 512Mi
  schedulerName: volcano
```

步骤4 查看各种类型节点上Pod的分值。

1. 包周期节点上的Pod。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    autoscaling.volcano.sh/dominated-by-balancer: default-balancer  #当前Pod通过名为default-balancer的Balancer CR资源控制扩缩优先级
    openvessel.io/workload-balancer-score: "100" #当前包周期节点对应的优先级，也代表着Pod的分值
    ...
  nodeName: 192.168.20.100 #包周期节点
```

2. 按需计费节点上的Pod。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    autoscaling.volcano.sh/dominated-by-balancer: default-balancer #当前Pod通过名为default-balancer的Balancer CR资源控制扩缩优先级
    openvessel.io/workload-balancer-score "10"  #当前按需计费节点对应的优先级，也代表着Pod的分值
    ...
  nodeName: 192.168.20.196 #按需计费节点
```

3. virtual-kubelet节点（弹性至CCI）的Pod。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    autoscaling.volcano.sh/dominated-by-balancer: default-balancer #当前Pod通过名为default-balancer的Balancer CR资源控制扩缩优先级
    openvessel.io/workload-balancer-score: "1"  #当前virtual-kubelet节点对应的优先级，也代表着pod的分值
    ...
  nodeName: virtual-kubelet #virtual-kubelet节点
```

步骤5 逐步进行工作负载的缩容操作，调小实例数。

virtual-kubelet节点（弹性至CCI）的Pod将优先被删除，其次为按需计费节点上的Pod，最后为包周期节点上的Pod。

----结束

默认应用扩缩容优先级策略

使用默认应用扩缩容优先级策略的情况下，集群中存在两个默认CR资源：

- Balancer CRD对应的CR资源

```
apiVersion: autoscaling.volcano.sh/v1alpha1
kind: Balancer
metadata:
  name: default-balancer
spec:
  balancerPolicyTemplateName: default-balancerpolicytemplate
  targets:
  - namespaceSelector:
      matchExpressions:
        - key: kubernetes.io/metadata.name
          operator: Exists
  weight: 10
```


表 6-12 Balancer 对象关键参数说明

字段	含义	类型	备注
metadata.name	名称	String	必填字段。
spec.balancerPolicyTemplateName	优先级策略名称	String	必填字段。值为集群中相应BalancerPolicyTemplate CR资源名。
spec.targets	优先级策略作用范围	Slice	必填字段。举例： • 针对default命名空间下的应用生效 spec: targets: - namespaceSelector: matchLabels: kubernetes.io/metadata.name: default • 针对default、other、another多个命名空间下的应用生效 spec: targets: - namespaceSelector: matchExpressions: - key: kubernetes.io/metadata.name operator: In values: - default - other - another • 针对所有命名空间下的应用生效 spec: targets: - namespaceSelector: matchExpressions: - key: kubernetes.io/metadata.name operator: Exists • 只针对名为xxx-xxx-xxx，类型为Deployment的应用生效 spec: targets: - objectSelectors: - name: xxx-xxx-xxx kind: Deployment • 只针对命名空间为default，名为xxx-xxx-xxx类型为Deployment的应用生效 spec: targets: - namespaceSelector: matchLabels: kubernetes.io/metadata.name: default objectSelectors: - name: xxx-xxx-xxx kind: Deployment

字段	含义	类型	备注
spec.weight	优先级策略权重	int32	必填字段。在集群存在多个Balancer 对象资源情况下，某个应用可能存在于多个Balancer对象作用范围的交集中，此时选择weight值大的Balancer对象生效。

- BalancerPolicyTemplate CRD对应的CR资源

```
apiVersion: autoscaling.volcano.sh/v1alpha1
kind: BalancerPolicyTemplate
metadata:
  name: default-balancerpolicytemplate
spec:
  policy:
    policyName: Priority
    priorities:
      priorityGroups:
      - priority: 10
        requirements:
        - key: node.cce.io/billing-mode
          operator: In
          values:
          - post-paid
      - priority: 100
        requirements:
        - key: node.cce.io/billing-mode
          operator: In
          values:
          - pre-paid
      - priority: 1
        requirements:
        - key: kubernetes.io/role
          operator: In
          values:
          - virtual-kubelet
          - bursting
```

表 6-13 BalancerPolicyTemplate 对象关键参数说明

字段	含义	类型	备注
metadata.name	名称	String	必填字段。
spec.policy	优先级策略内容	Struct	必填字段。
spec.policy.policyname	优先级策略名	String	必填字段。当前只支持名为“Priority”的优先级策略。

字段	含义	类型	备注
spec.policy.priorities.priorityGroups	优先级策略中定义的具体优先级	Slice	必填字段。举例： ● 将包周期节点的优先级设置为100 priorityGroups: - priority: 100 requirements: - key: node.cce.io/billing-mode operator: In values: - pre-paid ● 将按需计费节点的优先级设置为10 priorityGroups: - priority: 10 requirements: - key: node.cce.io/billing-mode operator: In values: - post-paid ● 将virtual-kubelet/bursting节点的优先级设置为1 priorityGroups: - priority: 1 requirements: - key: kubernetes.io/role operator: In values: - virtual-kubelet - bursting

自定义应用扩缩容优先级策略

BalancerPolicyTemplate 资源用来进行优先级策略定义，如果用户需要自定义应用扩缩容优先级策略，则需要针对其内容进行修改。

说明

如果存在多个BalancerPolicyTemplate资源，扩缩策略执行结果将受到这些资源对象的共同作用。因此，如果用户不存在默认扩缩容优先级策略的应用场景，可以执行如下命令对默认优先级策略进行删除。

```
kubectl delete balancerpolicytemplate default-balancerpolicytemplate
```

以“**扩容时优先将工作负载调度到HCE2.0操作系统的节点，其次调度到欧拉操作系统的节点；缩容时优先删除欧拉操作系统节点上的工作负载，其次删除HCE2.0操作系统上的工作负载**”为例：

步骤1 编写新BalancerPolicyTemplate 资源对象。

```
vim new-balancerpolicytemplate.yaml
```

内容如下：

```
apiVersion: autoscaling.volcano.sh/v1alpha1
kind: BalancerPolicyTemplate
metadata:
  name: new-balancerpolicytemplate
spec:
  policy:
    policyName: Priority
    priorities:
      priorityGroups:
      - priority: 10    # 设置欧拉操作系统节点优先级为10
```



```
    requirements:
    - key: os.name  # 节点操作系统标签
      operator: In
      values:
      - EulerOS_2.0_SP9x86_64  # 可能涉及操作系统的小版本号，用户可以根据自身场景，任意添加
  - priority: 100   # 设置HCE2.0操作系统节点优先级为100
    requirements:
    - key: os.name  # 节点操作系统标签
      operator: In
      values:
      - Huawei_Cloud_EulerOS_2.0_x86_64
```

步骤2 创建新BalancerPolicyTemplate资源对象。

```
kubectl create -f new-balancerpolicytemplate.yaml
```

步骤3 修改default-balancer对象内容，也可以按需进行新建balancer对象

```
kubectl edit balancer default-balancer
```

修改内容如下：

```
apiVersion: autoscaling.volcano.sh/v1alpha1
kind: Balancer
metadata:
  name: default-balancer
spec:
  balancerPolicyTemplateName: new-balancerpolicytemplate
  targets:
  - namespaceSelector:
      matchExpressions:
        - key: kubernetes.io/metadata.name
          operator: Exists
  weight: 10
```

步骤4 查看各个Pod的注解中openvessel.io/workload-balancer-score对应的值是否满足预期。

EulerOS节点上Pod的openvessel.io/workload-balancer-score注解对应值是10；HCE2.0节点上的pod 的openvessel.io/workload-balancer-score注解对应值是100。

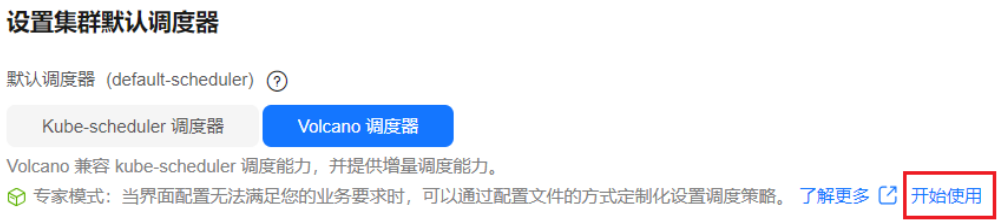
----结束

配置第三方工作负载应用扩缩容优先级策略

若工作负载不是Deployment类型，而是通过CRD进行管理，则可以通过在高级配置中进行相应设置，使volcano支持该工作负载的扩缩容优先级策略。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“配置中心”，在“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



步骤3 单击左侧导航栏的“插件中心”，在右侧找到Volcano调度器，单击“安装”或“编辑”，并在“参数配置”中设置Volcano调度器配置参数。

步骤4 指定需要支持的第三方工作负载类型，JSON示例如下：

```
{
    "default_scheduler_conf": {
...
    },
    "workload_balancer_score_annotation_key": "",
    "workload_balancer_third_party_types": "apps.kruise.io/v1alpha1/clonesets,apps.kruise.io/v1beta1/
statefulsets"
}
```

- workload_balancer_score_annotation_key：指定Pod的分值注解key，目前支持“openvessel.io/workload-balancer-score”或者“controller.kubernetes.io/pod-deletion-cost”，除此之外的值会导致volcano异常退出。
- workload_balancer_third_party_types：第三方工作负载的group + version + kind拼接成的字符串，多个CRD间以英文逗号分隔。

  例如“apps.kruise.io/v1alpha1/clonesets,apps.kruise.io/v1beta1/statefulsets”，注意kind需要为复数形式，例如 “apps.kruise.io/v1alpha1/cloneset”的非复数形式会导致监听不到对应的CRD。

  如果格式错误，会导致volcano异常退出；如果指定的CRD在集群上不存在，会导致应用扩缩容优先级策略无法正常工作。

若期望配置的CRD可以按照优先级缩容，则需要管理该CRD的controller可以在缩容时感知到Pod的分值注解，并按照得分控制缩容顺序。

----结束

附录：调整 Volcano 可调度的节点比例

编写volcano-scheduler资源对象。

```
kubectl edit deploy volcano-scheduler -nkube-system
```

内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: volcano-scheduler
    app.kubernetes.io/managed-by: Helm
    release: cceaddon-volcano
  name: volcano-scheduler
  namespace: kube-system
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: volcano-scheduler
  strategy:
    rollingUpdate:
      maxSurge: 10%
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: volcano-scheduler
        release: cceaddon-volcano
    spec:
      affinity:
        podAntiAffinity:
```



```
        preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - volcano-scheduler
            topologyKey: topology.kubernetes.io/zone
          weight: 100
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - volcano-scheduler
          topologyKey: kubernetes.io/hostname
      containers:
      - command:
        - /bin/sh
        - -c
        - /volcano-scheduler --leader-elect=true --lock-object-namespace=kube-system
          --feature-
gates=CSIMigrationFlexVolumeFuxi=true,CSIMigrationFlexVolumeFuxiComplete=true,MultiGPUScheduling=tr
ue
          --kube-api-qps=200 --alsologtostderr --listen-address=$(MY_POD_IP):8080
          --enable-healthz=true --healthz-address=$(MY_POD_IP):11251 --enable-metrics=true --percentage-
nodes-to-find=100
          --scheduler-conf=/volcano.scheduler/default-scheduler.conf -v=3 1>>/var/log/volcano/volcano-
scheduler.log
```

其中**--percentage-nodes-to-find=100**表示Volcano在进行调度选择时可以遍历集群中的所有节点。

6.6 云原生混部

6.6.1 云原生混部概述

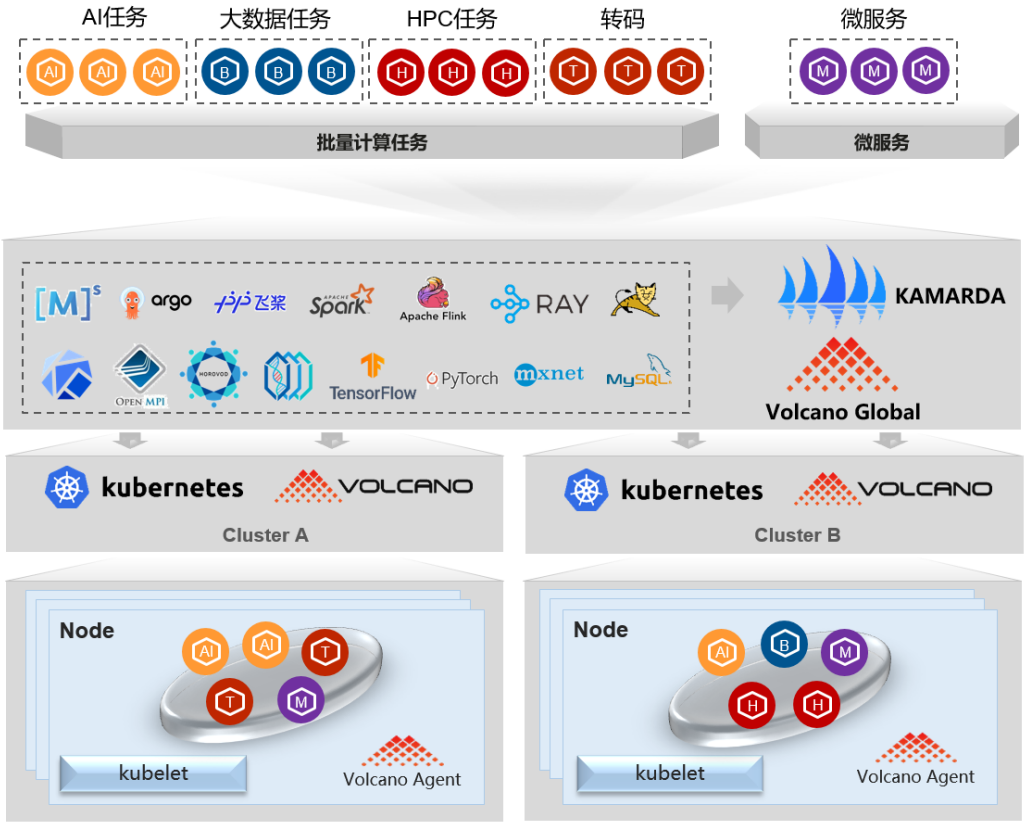
随着云原生技术迅速发展，海量应用正在走向云原生化。从2021年到2022年，Kubernetes集群中的云原生应用总数同比增长30%+，Kubernetes正在成为云时代的“操作系统”。但随着进一步调研发现，应用部署在Kubernetes集群后，大部分用户节点的CPU利用率不足15%。在调研不同类型客户，排除一些闲置资源、套餐活动等干扰因素后，发现造成资源利用率低的主要因素可归纳为如下几点：

1. 集群规划粒度过细，节点分布过散：集群规划粒度过细，节点分布在多个不同的集群中，使得计算资源无法共享，计算资源碎片数量增加。
2. 节点规格没有跟随应用迭代而变化，资源分配率低：初期节点规格与应用规格匹配度较好，资源分配率较高；随着应用版本迭代，应用申请资源发生变化，与节点规格比例差异较大，使节点分配率降低，计算资源碎片数量增加。
3. 业务“潮汐”特性明显，预留资源较多：在线业务具有明显日级别波峰、波谷特性，用户为保证服务的性能和稳定性按照波峰申请资源，集群的大部分资源处于闲置状态。
4. 在线和离线作业分布不同集群，资源无法分时复用：用户为在线和离线作业划分不同的K8s集群中，在线业务在波谷时，无法部署离线作业使用这部分资源。

这些都是云原生应用粗犷发展阶段的典型表现。在业务云原生化过程中，不同的业务架构有着不同的部署方案，不同架构的应用有着不同的演进节奏，不同的团队有着性

能和服务质量的平衡点。面对这样复杂的场景，应该如何化繁为简，帮助用户有步骤的提升资源利用率和控制成本呢?

CCE通过多年在混合部署领域的探索和实践，围绕Volcano和Kubernetes生态，构建帮助用户提升资源利用率，实现降本增效的云原生混部解决方案。



如上图所示，混部不是简单将小集群合并成一个大集群，然后将多个不同的业务部署在同一个集群中那么简单，而需要确保用户的应用能够部署到合适的位置，并能保障其需要的资源。这也是云原生混部解决方案中的两个核心设计：全域统一调度和资源分级管控。

全域统一调度和资源分级管控

全域统一调度

应用的全域统一调度的核心是全域和统一，比如：分布式云场景中跨云、跨集群的统一调度，以及不同在线应用、离线任务的统一调度。

- 首先，Volcano通过静态分析，获取应用的静态特征，如：CPU、内存、存储、GPU等资源的需求，应用间亲和性、区域亲和性、云平台亲和性等。
- 接着，Volcano对接监控系统，获取不同云平台资源、集群资源的动态数据，以及应用运行的数据，分析其规律，获得其运行态势，如：业务分级(天/周/月)的潮汐规律性、CPU敏感型、L3缓存敏感型、内存敏感型等。
- 最后，通过Volcano丰富多样的、按需启用的调度策略，将应用调度到合适的环境中。如：基于预测的智能调度策略、基于业务的binpack装箱/重调度策略、基于运行态势的资源超卖策略等。

Volcano将分布式云平台中的资源统一管理，将不同类型的应用调度到合适的位置，有效的解决了多集群带来的资源碎片问题和因应用迭代带来的节点规格不匹配问题，帮助用户从繁杂的资源规划和版本迭代带来的变化中解脱出来。

资源分级管控

应用被调度到合适的运行环境后，如何来保障其所需要的资源呢?

基于Huawei Cloud EulerOS 2.0操作系统，从CPU、L3缓存、内存、网络、存储等全方位提供资源隔离能力，并以内核态为主，用户态为辅，通过快速抢占（毫秒）和快速驱逐（秒级），保障在线业务的服务质量。

- 资源隔离的措施，如：CPU的绑核、NUMA亲和性、潮汐亲和特性，网络带宽控制等，有效的保障资源敏感型业务的SLO。
- 资源优先级控制的措施，如：CPU分级控制、内存分级控制、网络优先级控制、磁盘IO的优先级控制等，在提升资源分配率的同时，又少影响或不影响优先级高的业务SLO。

资源分级管控为业务潮汐明显的在线业务间混部、在线和离线业务混部奠定了基础。解决了应用预留资源较多、资源无法分时复用的问题。

在线作业与离线作业

从业务是否一直在线的角度看，其类型可分为在线作业和离线作业。

- **在线作业**：一般运行时间长，服务流量呈周期性，资源存在潮汐现象，但对服务SLA要求较高，如广告业务、电商业务等。
- **离线作业**：往往运行时间短，计算需求大，可容忍较高的时延，如AI/大数据业务。

功能介绍

功能	描述	参考文档
动态资源超卖	根据在线作业和离线作业类型，通过Volcano调度将集群中申请而未使用的资源（即申请量与使用量的差值）利用起来，实现资源超卖和混合部署，提升集群资源利用率。	动态资源超卖
CPU Burst弹性限流	提供一种可以短暂突破CPU Limit值的弹性限流机制，以降低业务长尾响应时间，可以有效提升时延敏感型业务的服务质量。	CPU Burst弹性限流
出口网络带宽保障	平衡在线业务与离线业务对出口网络带宽的使用，保证在线业务有足够的网络带宽。	出口网络带宽保障

6.6.2 开启云原生混部

前提条件

- 已创建一个CCE Standard集群或CCE Turbo集群，且版本满足以下要求：
  - v1.23集群：v1.23.9-r0及以上

  - v1.25集群：v1.25.4-r0及以上
- 集群中已安装1.10.0及以上版本的Volcano插件。

约束与限制

- 开启云原生混部后，Volcano调度器会开启超卖插件oversubscription，使用云原生混部过程中请确保该插件处于启用状态。
- 混部agent以DaemonSet方式亲和部署在OS类型为Huawei Cloud EulerOS 2.0 x86的节点上，非Huawei Cloud EulerOS 2.0 x86的节点不会部署agent。
- 默认节点池不支持修改混部配置。

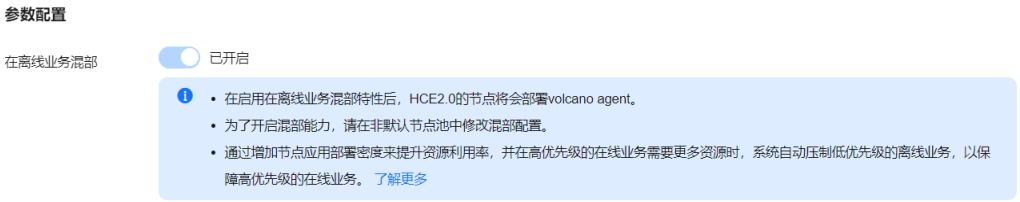
云原生混部配置

云原生混部以节点池粒度进行管理，您需要在节点池打开混部开关并进行混部配置。默认的混部配置，会启用混部所有能力并设置默认参数。您可以对默认的混部配置进行修改。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，单击节点池的“更多 > 混部配置”。

步骤3 如果Volcano插件未开启在离线业务混部功能，您需要在弹出的配置页面中先开启该功能，等待Volcano插件安装或更新完成后继续配置。



步骤4 在混部配置页面中，打开“节点池混部开关”。

说明

在打开节池混部开关时，会校验您之前是否启用了kubelet混部超卖配置，若已启用请在开启提示中确认将kubelet混部超卖自动迁移到云原生混部，具体迁移说明可查看**kubelet超卖迁移至云原生混部超卖说明**。

您可以对以下混部配置进行配置：

参数	默认行为	参数说明
CPU 弹性限流	开启	开启CPU Burst弹性限流后，当节点资源充足时，Pod的CPU实际使用量可以短暂突破CPU Limit值，以降低业务长尾响应时延，详情请参见**CPU Burst弹性限流**。
出口网络带宽	开启	在CCE Turbo集群中，支持在线业务与离线业务的网络隔离，详情请参见**出口网络带宽保障**。
资源超卖	开启	通过实时采集节点负载信息，挖掘节点已分配、但未使用的资源，实现动态超卖节点资源。您可以选择需要超卖的资源类型，默认同时开启CPU和内存资源超卖。详情请参见**动态资源超卖**。 **说明** • 若集群版本不满足条件，资源超卖功能配置不会生效，详情请参见**表6-14**。 • 修改超卖资源配置时： – 对于增加超卖资源类型，如超卖资源由CPU变为CPU、内存，此时可以随时添加。 – 对于减少超卖资源类型，如由CPU、内存变为仅超卖CPU，此时需要在合适的时间进行更改，即分配率不超过100%时才可进行安全更改。

步骤5 参数配置完成后，单击“确定”。

----结束

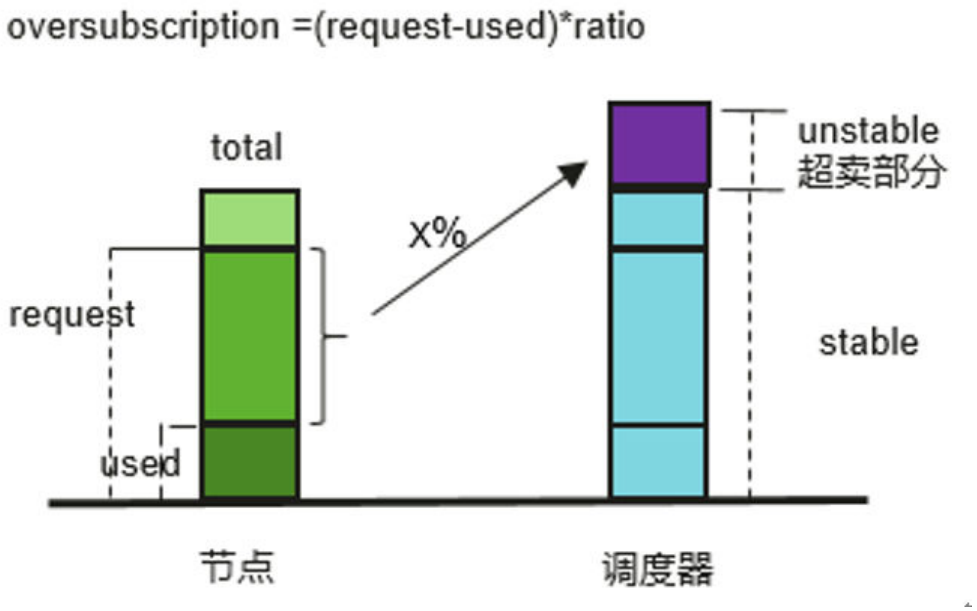
6.6.3 动态资源超卖

当前很多业务有波峰和波谷，部署服务时，为了保证服务的性能和稳定性，通常会按照波峰时需要的资源申请，但是波峰的时间可能很短，这样在非波峰时段就有资源浪费。另外，由于在线作业SLA要求较高，为了保证服务的性能和可靠性，通常会申请大量的冗余资源，因此，会导致资源利用率很低、浪费比较严重。

将这些申请而未使用的资源（即申请量与使用量的差值）利用起来，就是资源超卖。超卖资源适合部署离线作业，离线作业通常关注吞吐量，SLA要求不高，容忍一定的失败。

在线作业和离线作业混合部署在Kubernetes集群中将有效的提升集群整体资源利用率。

图 6-23 资源超卖示意图



资源超卖功能特性

说明

当节点池启用动态资源超卖和弹性伸缩时，由于高优先级应用业务资源使用量实时变化，导致超卖资源变化较快，为了避免节点频繁缩容和扩容，在节点缩容评估时暂不考虑超卖资源。

当前特性支持集群内在离线作业混部以及节点CPU和内存资源超卖，关键特性如下：

- 离线作业优先使用超卖节点

  若同时存在超卖与非超卖节点，在离线作业调度过程中，超卖节点得分高于非超卖节点，离线作业优先调度到超卖节点。

- 在线作业预选超卖节点时只能使用其非超卖资源

  在线作业只能使用超卖节点的非超卖资源，离线作业可以使用超卖节点的超卖及非超卖资源。

- 同一调度周期在线作业先于离线作业调度

  在线作业和离线作业同时存在时，优先调度在线作业。当节点资源使用率超过设定的驱逐上限且节点request值超过100%时，将会驱逐离线作业。

- 内核提供CPU/内存隔离特性

  CPU隔离：在线作业能够快速抢占离线作业的CPU资源，并压制离线作业的CPU使用。

  内存隔离：系统内存资源用尽触发OOM Kill时，内核优先驱逐离线作业。

- Kubelet离线作业准入规则

  在调度器将Pod调度到某个节点上之后，Kubelet在启动该Pod之前，会对该Pod进行准入判断，如果此时节点资源无法满足该Pod的Request值，kubelet则拒绝启动该Pod（predicateAdmitHandler.Admit）。满足以下两个条件时，Kubelet准入该Pod：

  - 待启动Pod Request与运行的在线作业Request之和 < 节点Allocatable
  - 待启动Pod Request与运行的在/离线作业Request之和 < 节点Allocatable + 节点超卖资源

- 支持超卖和混部分离

  开启节点池的混部开关后，默认的混部配置同时打开了混部和超卖功能，节点会被同时打上volcano.sh/colocation="true"和volcano.sh/oversubscription="true"的标签。若只进行在离线作业混部，而不使用超卖资源，需在混部配置中关闭资源超卖特性，关闭超卖特性后volcano.sh/oversubscription="true"标签会被移除。

  开启混部或超卖后可使用的特性组合如下：

开启混部	开启超卖	可以使用超卖资源	驱逐离线Pod场景
否	否	否	无
是	否	否	当节点实际资源使用率超过高水位线时，触发离线Pod驱逐
否	是	是	当节点实际资源使用率超过高水位线并且节点Pod的Request和大于100%，触发离线Pod驱逐
是	是	是	当节点实际资源使用率超过高水位线，触发离线Pod驱逐

使用方式

请根据集群版本确定资源超卖使用方式，详情请参见**表6-14**。

云原生混部资源超卖与兼容模式存在冲突，使用时选择两者中的一种即可。当集群版本不支持云原生混部时，云原生混部里的资源超卖的功能和参数配置均不会生效，若要使用资源超卖功能，请使用**kubelet超卖**。

表 6-14 集群版本与资源超卖功能生效方式对应关系表

集群版本	具体版本号	资源超卖功能生效方式	说明
v1.25以上	-	**云原生混部资源超卖**	-
v1.25	v1.25.4-r0及以上	**云原生混部资源超卖**	-

集群版本	具体版本号	资源超卖功能生效方式	说明
	v.1.25.4-r0以前的版本已经使用kubelet超卖，升级至v1.25.4-r0及以上版本	存量节点池：**kubelet超卖** 新建节点池：**云原生混部资源超卖**	对于存量的节点池，推荐将kubelet超卖迁移至云原生混部超卖进行统一管理，详情请参见**kubelet超卖迁移至云原生混部超卖说明**。
	v1.25.4-r0以下	**kubelet超卖**	不支持将kubelet超卖能力迁移到云原生混部超卖
v1.23	v1.23.9-r0及以上	**云原生混部资源超卖**	-
	v.1.23.9-r0以前的版本已经使用kubelet超卖，升级至v1.23.9-r0及以上版本	存量节点池：**kubelet超卖** 新建节点池：**云原生混部资源超卖**	对于存量的节点池，推荐将kubelet超卖迁移至云原生混部超卖进行统一管理，详情请参见**kubelet超卖迁移至云原生混部超卖说明**。
	v1.23.9-r0以下，但需满足集群版本大于等于v1.23.5-r0	**kubelet超卖**	不支持将kubelet超卖能力迁移到云原生混部超卖
v1.21	v1.21.7-r0及以上	**kubelet超卖**	-
v1.19	v1.19.16-r4及以上	**kubelet超卖**	-

开启云原生混部功能后，默认即开启了资源超卖功能，详情请参见**云原生混部资源超卖**。云原生混部volcano-agent负责超卖资源上报和节点压力驱逐，其核心特性包括CPU/内存压制、动态资源超卖、CPU Burst、出口网络QoS分级控制等。

为了兼容旧版本支持的资源超卖特性，仍然保留了通过人工设置kubelet参数来开启资源超卖的能力，详情请参见**kubelet超卖兼容模式（不推荐）**，但该方案仅支持基础能力，且不再演进，不支持CPU Burst、出口网络QoS分级控制等后续新特性。

云原生混部资源超卖

须知

规格约束

- 集群版本：
  - v1.23集群：v1.23.9-r0及以上
  - v1.25集群：v1.25.4-r0及以上
- 集群类型：CCE Standard集群或CCE Turbo集群。
- 节点OS：Huawei Cloud EulerOS 2.0
- 节点类型：x86架构的弹性虚拟机。
- Volcano插件版本：1.10.0及以上版本。

使用限制

- 使用超卖特性时，需保证Volcano未启用overcommit插件。
- 运行中的Pod无法进行在线和离线业务转换，如需转换需要重建Pod。
- 当节点设置cpu-manager-policy为静态绑核时，不允许将离线Pod设置为Guaranteed的Pod，若需要绑核则需要调整Pod为在线Pod，否则可能会发生离线Pod占用在线Pod的CPU导致在线Pod启动失败，以及离线Pod虽然调度成功但仍然启动失败的情况。
- 当节点设置cpu-manager-policy为静态绑核时，不应对所有在线Pod进行绑核，否则会出现在线Pod占用了所有的CPU或者memory资源导致上报的超卖资源很少的情况。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，在需要开启动态资源超卖的节点池中，单击“更多 > 混部配置”。

确认“节点池混部开关”及“资源超卖”开关已打开，详情请参见**云原生混部配置**。

步骤3 （可选）调整资源超卖参数。

表 6-15 资源超卖参数

名称	说明
CPU驱逐高水位线(%)	当节点CPU使用率超过设置值时，触发离线作业驱逐，节点不可调度。 默认值80，即当节点CPU使用率超过80%时，触发离线作业驱逐。
CPU驱逐低水位线(%)	CPU使用率高于高水位线时，触发离线作业驱逐，等到节点CPU使用率低于低水位线后，该节点才会重新接纳离线作业。 默认值为30，即当节点CPU使用率低于30%后，重新接纳离线作业。
内存驱逐高水位线(%)	当节点内存使用率超过设置值时，触发离线作业驱逐，节点不可调度。 默认值60，即当节点内存使用率超过60%时，触发离线作业驱逐。
内存驱逐低水位线(%)	节点内存使用率高于高水位线时，触发离线作业驱逐，等到节点内存利用率低于低水位线后，该节点才会重新接纳离线作业。 默认值为30，即当节点内存使用率低于30%后，重新接纳离线作业。

步骤4 确认Volcano插件配置。

```
kubectl edit cm volcano-scheduler-configmap -n kube-system
```

在volcano- scheduler-configmap中查看超卖的相关配置如下。同时确保插件配置中不能包含overcommit插件，如果存在（ - name: overcommit ），则需要删除该配置。

```
...
data:
  volcano-scheduler.conf: |
    actions: "allocate, backfill, preempt"    #设置preempt action
    tiers:
    - plugins:
      - name: gang
        enablePreemptable: false
        enableJobStarving: false
      - name: priority
      - name: conformance
      - name: oversubscription
    - plugins:
      - name: drf
      - name: predicates
      - name: nodeorder
      - name: binpack
    - plugins:
      - name: cce-gpu-topology-predicate
      - name: cce-gpu-topology-priority
      - name: cce-gpu
...
```

步骤5 创建高优、低优priorityClass资源。

```
cat <<EOF | kubectl apply -f -

apiVersion: scheduling.k8s.io/v1
description: Used for high priority pods
```



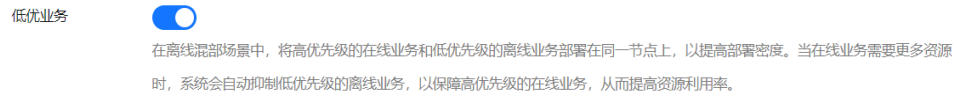
```
kind: PriorityClass
metadata:
  name: volcano-production
preemptionPolicy: PreemptLowerPriority
value: 999999
---
apiVersion: scheduling.k8s.io/v1
description: Used for low priority pods
kind: PriorityClass
metadata:
  name: volcano-free
preemptionPolicy: PreemptLowerPriority
value: -90000

EOF
```

步骤6 部署在离线作业。

在线、离线作业均需设置schedulerName字段的值为“volcano”，启用Volcano调度器。

- 对于离线作业，在创建工作负载时，开启“低优业务”，该工作负载会被添加对应的离线作业注解**volcano.sh/qos-level: "-1"**。



并设置priorityClassName字段为volcano-free：

```
kind: Deployment
apiVersion: apps/v1
spec:
  replicas: 4
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        volcano.sh/qos-level: "-1" # 离线作业注解
    spec:
      schedulerName: volcano             # 调度器使用Volcano
      priorityClassName: volcano-free          # 设置volcano-free priorityClass
      ...
```

- 对于在线作业，设置priorityClassName字段为volcano-production：

```
kind: Deployment
apiVersion: apps/v1
spec:
  replicas: 4
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
    spec:
      schedulerName: volcano          # 调度器使用Volcano
      priorityClassName: volcano-production    # 设置volcano-production priorityClass
      ...
```

步骤7 通过如下命令可查看当前超卖资源量以及资源使用情况。

```
kubectl describe node <nodeIP>
# kubectl describe node 192.168.0.0
Name:              192.168.0.0
Roles:             <none>
Labels:            ...
                   volcano.sh/oversubscription=true
Annotations:       ...
                   volcano.sh/oversubscription-cpu: 2335
                   volcano.sh/oversubscription-memory: 341753856
```



```
Allocatable:
  cpu:                3920m
  memory:             6263988Ki
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource            Requests      Limits
  --------            --------      ------
  cpu                 4950m (126%)  4950m (126%)
  memory              1712Mi (27%)  1712Mi (27%)
```

----结束

kubelet 超卖兼容模式（不推荐）

须知

规格约束

- 集群版本：
  - v1.19集群：v1.19.16-r4及以上版本
  - v1.21集群：v1.21.7-r0及以上版本
  - v1.23集群：v1.23.5-r0及以上版本
  - v1.25及以上版本
- 集群类型：CCE Standard集群或CCE Turbo集群。
- 节点OS：EulerOS 2.9 (内核kernel-4.18.0-147.5.1.6.h729.6.eulerosv2r9.x86_64)或者Huawei Cloud EulerOS 2.0
- 节点类型：弹性虚拟机。
- Volcano插件版本：1.7.0及以上版本。

使用限制

- 使用超卖特性时，需保证Volcano未启用overcommit插件。
- 修改超卖节点标签不会影响已经运行的pod。
- 运行中的pod无法进行在线和离线业务转换，如需转换需要重建pod。
- 集群中有节点配置超卖标签volcano.sh/oversubscription=true时，Volcano插件必须要增加oversubscription配置，否则会导致超卖节点调度异常。标签配置需要由用户保证，调度器不会对插件和节点配置进行检查。详细标签说明请参见**表6-16**。
- 超卖特性开关目前不支持统一配置，若要关闭超卖特性，需要同时进行以下操作：
  - 去掉超卖节点的volcano.sh/oversubscription标签。
  - 设置节点池的超卖开关over-subscription-resource为false。
  - 修改Volcano调度器的名字为volcano-scheduler-configmap的configmap，并去掉oversubscription插件。
- 当节点设置cpu-manager-policy为静态绑核时，不允许将离线Pod设置为Guaranteed的Pod，若需要绑核则需要调整Pod为在线Pod，否则可能会发生离线Pod占用在线Pod的CPU导致在线Pod启动失败，以及离线Pod虽然调度成功但仍然启动失败的情况。
- 当节点设置cpu-manager-policy为静态绑核时，不应对所有在线Pod进行绑核，否则会出现在线Pod占用了所有的CPU或者memory资源导致上报的超卖资源很少的情况。

集群中有节点配置超卖标签volcano.sh/oversubscription=true时，Volcano插件必须要增加oversubscription配置，否则会导致超卖节点调度异常。相关配置情况如**表6-16**所示。

标签配置需要您自行保证，调度器不会对插件和节点配置进行检查。

表 6-16 超卖标签配置调度说明

插件超卖配置	节点超卖标签	调度行为
有	有	超卖调度
有	无	正常调度
无	无	正常调度
无	有	无法调度，或者调度失败，应避免这种配置

步骤1 使用kubectl连接集群。

步骤2 确认Volcano插件配置。

```
kubectl edit cm volcano-scheduler-configmap -n kube-system
```

在volcano- scheduler-configmap中查看超卖的相关配置如下。同时确保插件配置中不能包含overcommit插件，如果存在（ - name: overcommit ），则需要删除该配置。

```
...
data:
  volcano-scheduler.conf: |
    actions: "allocate, backfill, preempt"    #设置preempt action
    tiers:
    - plugins:
      - name: gang
        enablePreemptable: false
        enableJobStarving: false
      - name: priority
      - name: conformance
      - name: oversubscription
    - plugins:
      - name: drf
      - name: predicates
      - name: nodeorder
      - name: binpack
    - plugins:
      - name: cce-gpu-topology-predicate
      - name: cce-gpu-topology-priority
      - name: cce-gpu
...
```

步骤3 开启节点超卖特性。

开启超卖特性的节点，才能配置标签使用超卖资源。相关节点只能在节点池中创建。当前超卖特性开关配置方式如下：

1. 创建节点池。
2. 节点池创建完成后，单击节点池名称后的“配置管理”。
3. 在侧边栏滑出的“配置管理”窗口中，将kubelet组件配置中的“节点超卖特性(over-subscription-resource)”参数设置为开启，并单击“确定”。

步骤4 设置节点超卖标签。

超卖节点需增加超卖标签volcano.sh/oversubscription。当节点设置该标签并且值为true时，该节点为超卖节点，否则为非超卖节点。

```
kubectl label node 192.168.0.0 volcano.sh/oversubscription=true
```

节点还支持如下超卖相关的阈值，如表6-17所示。示例如下：

```
kubectl annotate node 192.168.0.0 volcano.sh/evicting-cpu-high-watermark=70
```


查询该节点信息：

```
# kubectl describe node 192.168.0.0
Name:               192.168.0.0
Roles:              <none>
Labels:             ...
                    volcano.sh/oversubscription=true
Annotations:        ...
                    volcano.sh/evicting-cpu-high-watermark: 70
```

表 6-17 节点超卖 Annotations

名称	说明
volcano.sh/evicting-cpu-high-watermark	CPU使用率高水位线。当节点CPU使用率超过设置值时，触发离线作业驱逐，节点不可调度。 默认值80，即当节点CPU使用率超过80%时，触发离线作业驱逐。
volcano.sh/evicting-cpu-low-watermark	CPU使用率低水位线。CPU使用率高于高水位线时，触发离线作业驱逐，等到节点CPU使用率低于低水位线后，该节点才会重新接纳离线作业。 默认值为30，即当节点CPU使用率低于30%后，重新接纳离线作业。
volcano.sh/evicting-memory-high-watermark	内存使用率高水位线。当节点内存使用率超过设置值时，触发离线作业驱逐，节点不可调度。 默认值60，即当节点内存使用率超过60%时，触发离线作业驱逐。
volcano.sh/evicting-memory-low-watermark	内存使用率低水位线。节点内存使用率高于高水位线时，触发离线作业驱逐，等到节点内存利用率低于低水位线后，该节点才会重新接纳离线作业。 默认值为30，即当节点内存使用率低于30%后，重新接纳离线作业。
volcano.sh/oversubscription-types	超卖资源类型，支持如下三种配置： • cpu (超卖CPU) • memory (超卖内存) • cpu,memory (超卖CPU和内存) 默认值为“cpu,memory”

步骤5 创建高优、低优priorityClass资源。

```
cat <<EOF | kubectl apply -f -

apiVersion: scheduling.k8s.io/v1
description: Used for high priority pods
kind: PriorityClass
metadata:
  name: volcano-production
```



```
preemptionPolicy: PreemptLowerPriority
value: 999999
---
apiVersion: scheduling.k8s.io/v1
description: Used for low priority pods
kind: PriorityClass
metadata:
  name: volcano-free
preemptionPolicy: PreemptLowerPriority
value: -90000

EOF
```

步骤6 部署在离线作业，并分别为在离线作业设置priorityClass。

离线作业需在annotation中增加volcano.sh/qos-level标签以区分其为离线作业，值的范围为-7~7之间的整数，小于0代表低优先级任务，即离线作业，大于等于0代表高优先级任务，即在线作业。在线作业不需要设置该标签。在线、离线作业均需设置schedulerName字段的值为“volcano”，启用Volcano调度器。

说明

在线/在线、离线/离线作业间的优先级暂时未做区分，且未对值的合法性做校验，若设置的离线作业的volcano.sh/qos-level标签值不是-7~0之间的负整数，则统一按在线作业处理。

离线作业：

```
kind: Deployment
apiVersion: apps/v1
spec:
  replicas: 4
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        volcano.sh/qos-level: "-1"        # 离线作业注解
    spec:
      schedulerName: volcano              # 调度器使用Volcano
      priorityClassName: volcano-free          # 设置volcano-free priorityClass
      ...
```

在线作业：

```
kind: Deployment
apiVersion: apps/v1
spec:
  replicas: 4
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
    spec:
      schedulerName: volcano          # 调度器使用Volcano
      priorityClassName: volcano-production    # 设置volcano-production priorityClass
      ...
```

步骤7 通过如下命令可查看当前超卖资源量以及资源使用情况。

kubectl describe node *<nodeIP>*

```
# kubectl describe node 192.168.0.0
Name:               192.168.0.0
Roles:              <none>
Labels:             ...
                    volcano.sh/oversubscription=true
Annotations:        ...
                    volcano.sh/oversubscription-cpu: 2335
                    volcano.sh/oversubscription-memory: 341753856
```



```
Allocatable:
  cpu:                3920m
  memory:             6263988Ki
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests      Limits
  --------           --------      ------
  cpu                4950m (126%)  4950m (126%)
  memory             1712Mi (27%)  1712Mi (27%)
```

其中，CPU单位为m，内存单位为字节。

----结束

资源超卖部署示例

下面将通过示例演示混合部署离线作业和在线作业。

步骤1 假设一个集群存在两个节点，1个超卖节点和1个非超卖节点，如下所示。

```
# kubectl get node
NAME           STATUS   ROLES    AGE    VERSION
192.168.0.173  Ready    <none>   4h58m  v1.19.16-r2-CCE22.5.1
192.168.0.3    Ready    <none>   148m   v1.19.16-r2-CCE22.5.1
```

- 192.168.0.173为超卖节点（包含标签volcano.sh/oversubscirption=true）
- 192.168.0.3为非超卖节点（不包含标签volcano.sh/oversubscirption=true）

```
# kubectl describe node 192.168.0.173
Name:               192.168.0.173
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    ...
                    volcano.sh/oversubscription=true
```

步骤2 提交离线作业，资源充足的情况下，离线作业都调度到了超卖节点上。

离线作业模板如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: offline
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: offline
  template:
    metadata:
      labels:
        app: offline
      annotations:
        volcano.sh/qos-level: "-1"       #离线作业标签
    spec:
      schedulerName: volcano             #调度器使用Volcano
      priorityClassName: volcano-free         # 设置volcano-free priorityClass
      containers:
        - name: container-1
          image: nginx:latest
          imagePullPolicy: IfNotPresent
          resources:
            requests:
              cpu: 500m
              memory: 512Mi
            limits:
              cpu: "1"
              memory: 512Mi
```



```
    imagePullSecrets:
      - name: default-secret
```

离线作业调度到超卖节点上运行。

```
# kubectl get pod -o wide
NAME                       READY   STATUS    RESTARTS   AGE     IP              NODE
offline-69cdd49bf4-pmjp8   1/1     Running   0          5s    192.168.10.178   192.168.0.173
offline-69cdd49bf4-z8kxh   1/1     Running   0          5s    192.168.10.131   192.168.0.173
```

步骤3 提交在线作业，资源充足时，在线作业调度到了非超卖节点。

在线作业模板如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: online
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: online
  template:
    metadata:
      labels:
        app: online
    spec:
      schedulerName: volcano                # 调度器使用Volcano
      priorityClassName: volcano-production         # 设置volcano-production priorityClass
      containers:
        - name: container-1
          image: resource_consumer:latest
          imagePullPolicy: IfNotPresent
          resources:
            requests:
              cpu: 1400m
              memory: 512Mi
            limits:
              cpu: "2"
              memory: 512Mi
      imagePullSecrets:
        - name: default-secret
```

在线作业调度到非超卖节点上运行。

```
# kubectl get pod -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP              NODE
online-ffb46f656-4mwr6   1/1     Running   0          5s    192.168.10.146   192.168.0.3
online-ffb46f656-dqdv2   1/1     Running   0          5s    192.168.10.67    192.168.0.3
```

步骤4 提升超卖节点资源使用率，观察触发离线作业驱逐。

部署在线任务到超卖节点（192.168.0.173）上：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: online
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: online
  template:
    metadata:
      labels:
        app: online
    spec:
      affinity:                         # 提交在线任务至超卖节点
        nodeAffinity:
```



```
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
            - 192.168.0.173
  schedulerName: volcano                # 调度器使用Volcano
  priorityClassName: volcano-production          # 设置volcano-production priorityClass
  containers:
    - name: container-1
      image: resource_consumer:latest
      imagePullPolicy: IfNotPresent
      resources:
        requests:
          cpu: 700m
          memory: 512Mi
        limits:
          cpu: 700m
          memory: 512Mi
  imagePullSecrets:
    - name: default-secret
```

同时提交在/离线作业到超卖节点（192.168.0.173）上。

```
# kubectl get pod -o wide
NAME                    READY   STATUS    RESTARTS  AGE     IP             NODE
offline-69cdd49bf4-pmjp8  1/1     Running    0      13m   192.168.10.178   192.168.0.173
offline-69cdd49bf4-z8kxh  1/1     Running    0      13m   192.168.10.131   192.168.0.173
online-6f44bb68bd-b8z9p  1/1     Running     0     3m4s   192.168.10.18   192.168.0.173
online-6f44bb68bd-g6xk8  1/1     Running     0     3m12s   192.168.10.69   192.168.0.173
```

观察超卖节点（192.168.0.173），可以看出存在超卖资源，其中CPU为2343m，内存为3073653200字节。同时CPU分配率已超过100%。

```
# kubectl describe node 192.168.0.173
Name:              192.168.0.173
Roles:             <none>
Labels:             ...
                   volcano.sh/oversubscription=true
Annotations:        ...
                   volcano.sh/oversubscription-cpu: 2343
                   volcano.sh/oversubscription-memory: 3073653200
                   ...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource             Requests      Limits
  --------             --------      ------
  cpu                  4750m (121%)  7350m (187%)
  memory                 3760Mi (61%)  4660Mi (76%)
                    ...
```

增大节点上在线作业的CPU使用率，可以观察到触发离线作业驱逐。

```
# kubectl get pod -o wide
NAME                    READY   STATUS    RESTARTS  AGE     IP             NODE
offline-69cdd49bf4-bwdm7  1/1     Running    0      11m   192.168.10.208   192.168.0.3
offline-69cdd49bf4-pmjp8   0/1     Evicted    0      26m   <none>         192.168.0.173
offline-69cdd49bf4-qpdss   1/1     Running    0      11m   192.168.10.174   192.168.0.3
offline-69cdd49bf4-z8kxh   0/1     Evicted    0      26m   <none>         192.168.0.173
online-6f44bb68bd-b8z9p   1/1     Running     0     24m   192.168.10.18   192.168.0.173
online-6f44bb68bd-g6xk8   1/1     Running     0     24m   192.168.10.69   192.168.0.173
```

----结束

错误处理建议

- 超卖节点kubelet重启后，由于Volcano调度器和kubelet的资源视图不同步，部分新调度的作业会出现OutOfCPU的情况，属于正常现象，一段时间后会恢复正常，Volcano调度器能够正常调度在/离线作业。

- 在/离线作业提交后，因当前内核不支持离线作业修改为在线作业，因此不建议动态修改作业类型（添加或者删除Pod的annotation volcano.sh/qos-level: "-1"）。
- CCE通过cgroups系统中的状态信息收集节点上所有运行的Pod占用的资源量（CPU/内存），可能与用户监控到的资源使用率有所不同，例如使用top命令看到的资源统计。
- 对于增加超卖资源类型，如超卖资源由cpu变为cpu、memory，此时可以随时添加。

  对于减少超卖资源类型，如由cpu、memory变为仅超卖cpu，此时需要在合适的时间进行更改，即分配率不超过100%时才可进行安全更改。
- 当离线作业先部署到节点，并占用了在线作业的资源，导致资源不足在线作业无法调度时，需要为在线作业设置比离线作业更高的priorityClass。
- 若节点上只有在线作业，且达到了驱逐水位线，则离线作业调度到当前节点后会很快被驱逐，此为正常现象。

kubelet 超卖迁移至云原生混部超卖说明

当集群满足表6-14中的迁移场景时，您可以参照以下步骤进行迁移：

1. 在节点池中开启云原生混部资源超卖时，若检测到之前已开启了kubelet混部超卖，由于云原生混部资源超卖与kubelet超卖兼容模式无法同时生效，此时将自动关闭kubelet超卖实现迁移。
2. kubelet超卖将自动迁移至云原生混部超卖，切换过程中kubelet会短暂清除掉节点annotation上报的超卖资源，并驱逐离线Pod直至节点分配率小于100%，此为正常现象，随后由volcano-agent接管超卖，超卖功能恢复正常。

6.6.4 基于 Pod 实例画像的资源超卖

Volcano新增基于Pod实例画像的超卖量算法。该算法持续采集并累积节点上Pod的CPU和内存利用率，统计Pod资源用量的概率分布特征，进而计算出节点资源用量的概率分布特征，从而在一定的置信度下给出节点资源用量的评估值。基于Pod实例画像的超卖量算法会同时考虑节点资源使用的整体水位和起伏变化，计算出相对稳定的超卖量，减少资源竞争几率，避免业务波动导致Pod频繁驱逐。

相比于直接利用节点实时CPU内存利用率的算法，基于Pod实例画像的算法能够避免超卖量波动大，对突发资源尖峰覆盖不足的问题，在保障业务性能相对稳定的前提下超卖资源。

工作原理

基于Pod实例画像的资源超卖由Volcano agent和Volcano scheduler配合完成。开启该能力后，Volcano agent会周期性采集节点上Pod的CPU和内存利用率，计算每个Pod的CPU和内存用量均值、峰值和标准差，并基于Pod的这些统计特征值，进一步计算节点的CPU和内存用量评估值。

超卖量的计算算法：**节点资源超卖量 = (节点资源分配量 - 节点资源用量评估值) * 超卖比例**

超卖量会周期性更新到节点的annotation中，以便Volcano scheduler基于各个节点的超卖量进行Pod调度。

前提条件

已启用动态资源超卖。具体操作请参见动态资源超卖。

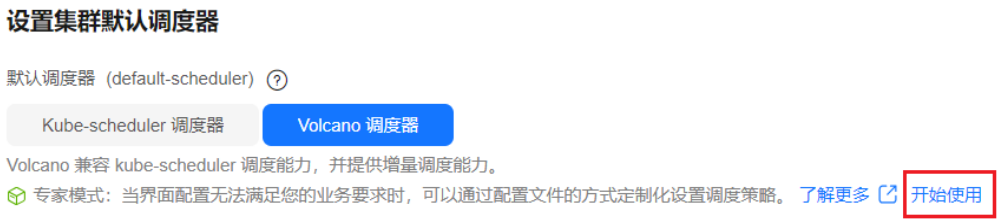
使用方法

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，在需要开启动态资源超卖的节点池中，单击“更多 > 混部配置”。

确认“节点池混部开关”及“资源超卖”开关已打开，详情请参见**云原生混部配置**。

步骤3 在左侧导航栏中选择“配置中心”，在上方的标签中选择“调度配置”，在“设置集群默认调度器”配置中选择“Volcano 调度器”，并单击“专家模式”右侧的“开始使用”，进入专家模式配置页面。



步骤4 进入CCE专家模式配置页面，配置以下两个选项：

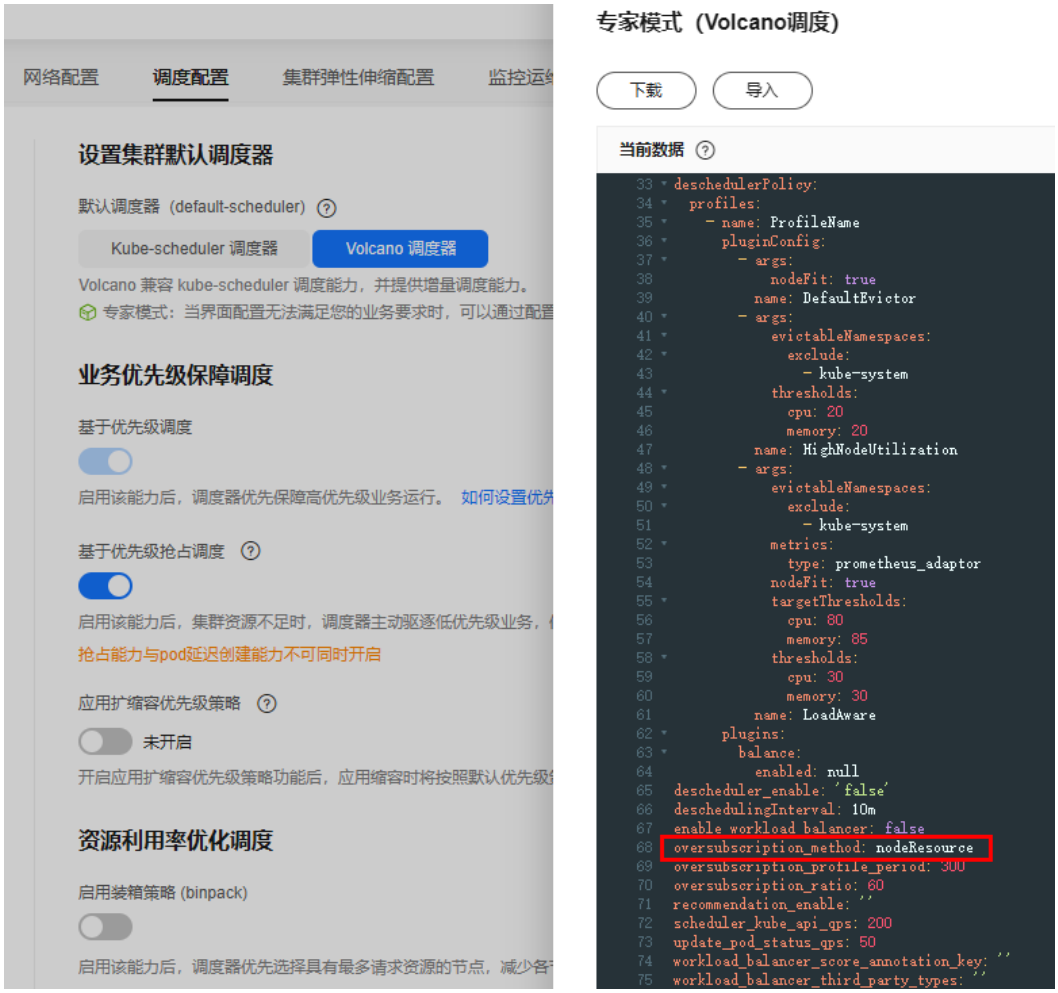
配置项	说明
overSubscriptionMethod	超卖量计算方法，目前支持nodeResource和podProfile。nodeResource为默认的基于节点资源用量的算法，podProfile为基于Pod实例画像的算法。
profilePeriod	Pod实例画像的周期，单位为秒，支持范围是60-2592000，即1分钟到1个月。对于指标采集累积时长未达到周期的Pod，将使用Pod资源请求量来计算节点的资源用量。 因此，初始启用基于Pod实例画像的算法，未达到画像周期之前，节点的超卖量会为0。

----结束

使用示例

使用基于Pod实例画像的资源超卖前

步骤1 确认专家模式中，oversubscription_method配置项的值为“nodeResource”，这表明当前集群采用的为默认的基于节点资源用量的算法。



步骤2 通过CCE控制台，创建一个redis工作负载作为示例，并绑定一个“节点访问”类型的服务。

步骤3 通过以下命令，对刚发放的Redis服务施加负载，模拟业务负载变化的场景。

```
./redis-benchmark -h <node_ip> -p 32293 -t set,get -n 3000000 -q
sleep 30
./redis-benchmark -h <node_ip> -p 32293 -t set,get -n 2000000 -q
sleep 20
./redis-benchmark -h <node_ip> -p 32293 -t set,get -n 2500000 -q
```

请将上述脚本中的*<node_ip>*替换成集群中节点的实际IP，另外32293端口则为上一步服务详情中查询到的节点端口。

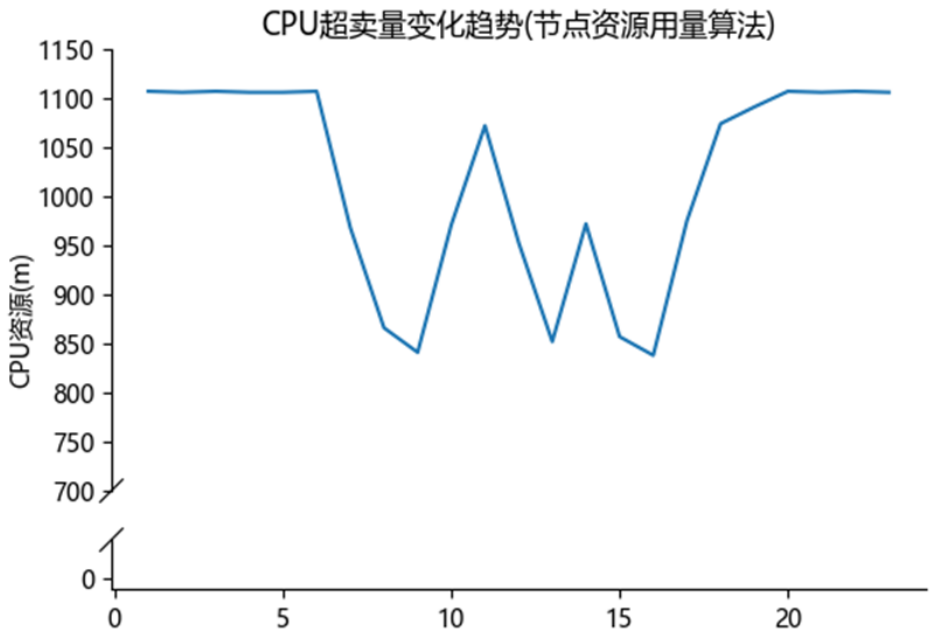
步骤4 通过以下命令，可以查询节点当前的超卖资源量并持续观察其变化。


```
kubectl describe node 192.168.98.230
```

回显如下：

```
Name:               192.168.98.230
Roles:              <none>
Labels:             ...
                    volcano.sh/colocation=true
                    volcano.sh/oversubscription=true
Annotations:        ...
                    volcano.sh/oversubscription-cpu: 1103
                    volcano.sh/oversubscription-memory: 1076471825
                    ...
CreationTimestamp:  Fri, 20 Sep 2024 16:12:33 +0800
...
```

步骤5 以下是根据Redis负载施加前后一段时间查询的CPU超卖量绘制的图表，可以看到在Redis负载暂停的阶段，CPU超卖量会有一个增长，但在负载重新启动后，CPU超卖量会立即减少。如果在增长的时刻节点调度了其他Pod进来，当负载重新启动后，节点就可能发生CPU争抢，导致Pod驱逐。



----结束

使用基于Pod实例画像的资源超卖后

步骤1 重新进入专家模式配置页面，将oversubscription_method配置项设置为“podProfile”，将oversubscription_profile_period配置项设置为“60”。出于快速演示的考虑，这里将画像周期设置为60秒，实际使用时请根据业务的特点选择合适的画像周期，以覆盖业务完整的资源使用规律周期。

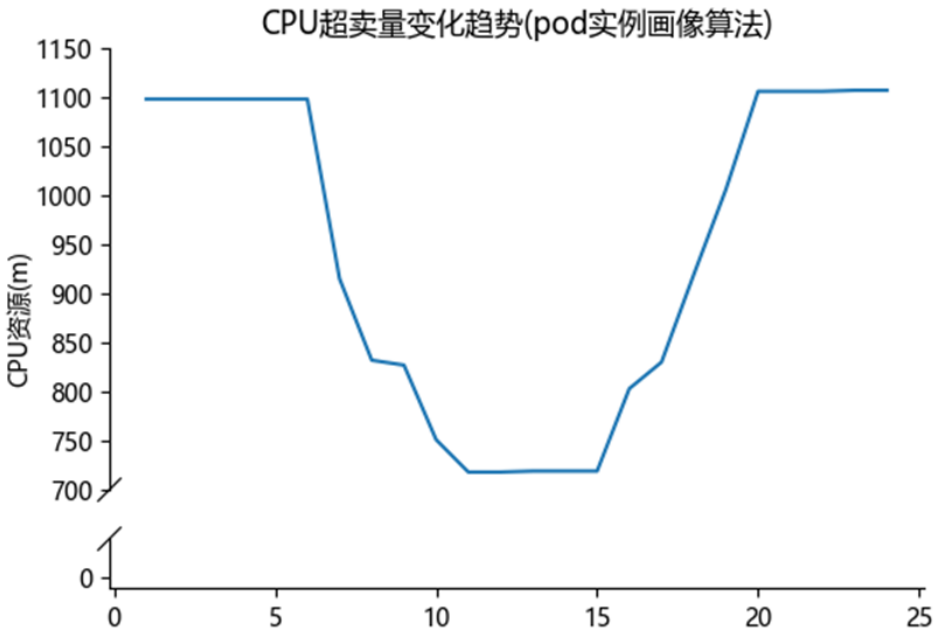
专家模式（Volcano调度）

下载　导入

当前数据

```
      pluginConfig:
        - args:
            nodeFit: true
          name: DefaultEvictor
        - args:
            evictableNamespaces:
              exclude:
                - kube-system
            thresholds:
              cpu: 20
              memory: 20
          name: HighNodeUtilization
        - args:
            evictableNamespaces:
              exclude:
                - kube-system
            metrics:
              type: prometheus_adaptor
            nodeFit: true
            targetThresholds:
              cpu: 80
              memory: 85
            thresholds:
              cpu: 30
              memory: 30
          name: LoadAware
      plugins:
        balance:
          enabled: null
descheduler_enable: 'false'
deschedulingInterval: 10m
enable_workload_balancer: false
oversubscription_method: podProfile
oversubscription_profile_period: 60
oversubscription_ratio: 60
recommendation_enable: ''
scheduler_kube_api_qps: 200
update_pod_status_qps: 50
workload_balancer_score_annotation_key: ''
workload_balancer_third_party_types: ''
```

步骤2 等待大约2分钟让Volcano agent完成配置切换和画像数据的积累，然后重新运行**步骤3**中的施加负载命令，通过查询的超卖资源量，可以再次绘制CPU超卖量图表。



可以看到，在整个脚本运行过程中，基于Pod实例画像的算法会将Pod资源用量的波动也考虑进去，计算出一个平稳的超卖量，避免Pod资源用量波动导致的资源竞争和Pod驱逐。

----结束

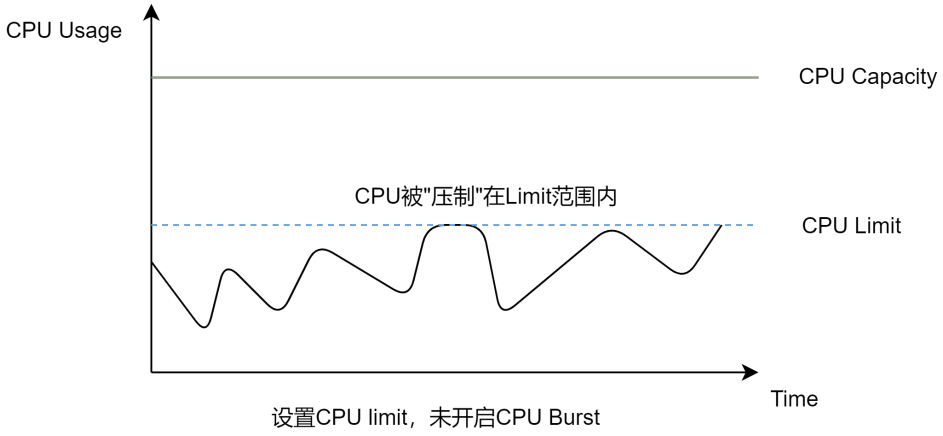
6.6.5 CPU Burst 弹性限流

若Pod中容器设置了CPU Limit值，则该容器CPU使用将会被限制在Limit值以内，形成对CPU的限流。频繁的CPU限流会影响业务性能，增大业务长尾响应时延，对于时延敏感型业务的影响尤为明显。

CPU Burst提供了一种可以短暂突破CPU Limit值的弹性限流机制，以降低业务长尾响应时间。其原理是业务在每个CPU调度周期内使用的CPU配额有剩余时，系统对这些CPU配额进行累计，在后续的调度周期内如果需要突破CPU Limit时，使用之前累计的CPU配额，以达到突破CPU Limit的效果。

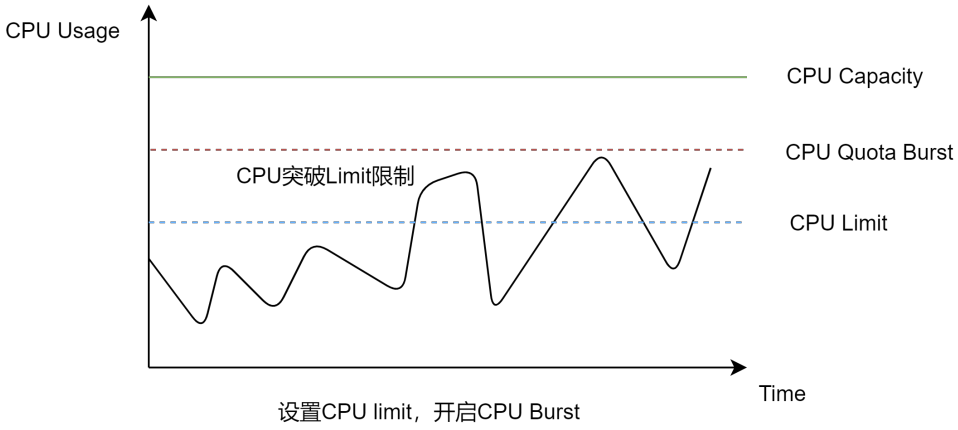
- 未开启CPU Burst时，容器可以使用的CPU配额会被限制在Limit以内，无法实现Burst。

图 6-24 未开启 CPU Burst



- 开启CPU Burst后，容器使用的CPU配额可以突破Limit限制，实现Burst。

图 6-25 开启 CPU Burst



约束与限制

- 集群版本：CCE Turbo集群且集群版本为v1.23.5-r0及以上。
- OS版本：Huawei Cloud EulerOS 2.0。
- 集群中需要安装Volcano 1.9.0及以上版本的插件，且开启混合部署开关。

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，在需要开启CPU Burst弹性限流的节点池中，单击“更多 > 混部配置”。

确认“节点池混部开关”及“CPU Burst 弹性限流”开关已打开。

说明

已有的混部策略中，当CPU Burst开关由打开到关闭时，已经设置CPU Burst的存量Pod不会关闭CPU Burst功能，关闭CPU Burst仅针对新建的Pod生效。

步骤3 在已打开混合部署的节点池中部署工作负载。以nginx为例，设置CPU Request为2，Limit为4，并为工作负载创建集群内访问的Service。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        volcano.sh/enable-quota-burst: "true"
        volcano.sh/quota-burst-time: "200000"
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        resources:
          limits:
            cpu: "4"
          requests:
            cpu: "2"
      imagePullSecrets:
        - name: default-secret
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  selector:
    app: nginx
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```


注解	是否必选	描述
volcano.sh/enable-quota-burst	是	为工作负载开启CPU Burst。
volcano.sh/quota-burst-time	否	为保障CPU调度稳定性以及降低多个容器同时发生CPU Burst时的争用，默认设置的CPU Burst值与CPU Quota值相等，即容器最多可以使用两倍的CPU Limit值，默认为Pod内所有业务容器设置CPU Burst。 例如容器的CPU Limit值为4时，CPU Burst默认值为400000（此处1核=100000），表示达到Limit值后最多可以额外使用4核CPU。

步骤4 验证CPU Burst。

您可以使用wrk工具对工作负载进行加压，观察开启和关闭CPU Burst时业务的时延、限流情况、突破CPU limit的情况。

1. 使用以下命令为Pod加压，其中<service_ip>为Pod关联的Service IP。
```
# 您需要在节点上下载并安装wrk工具
# 在Apache配置中开启了Gzip压缩模块, 用于模拟服务端处理请求的计算逻辑。
# 执行加压命令, 需注意修改目标应用的IP地址。
wrk -H "Accept-Encoding: deflate, gzip" -t 4 -c 28 -d 120  --latency --timeout 2s http://<service_ip>
```
2. 获取Pod ID。
```
kubectl get pod -n <namespace> <pod_name> -o jsonpath='{.metadata.uid}'
```
3. 限流情况和突破CPU limit的情况可以在节点上通过以下命令查看，其中*<pod_id>*为Pod的ID。
```
cat /sys/fs/cgroup/cpu/kubepods/burstable/pod<pod_id>/cpu.stat
```
回显示例如下：
```
nr_periods 0  #经过多少个调度周期
nr_throttled 0  #容器被限流的次数
throttled_time 0  #容器总的被限流的时间（纳秒）
nr_bursts 0 #容器突破CPU limit的次数
burst_time 0 #容器总共burst的时间
```

表 6-18 本例中结果汇总

是否开启CPU Burst	p99时延	nr_throttled 限流次数	throttled_time 总限流时间	nr_bursts 突破limit值次数	bursts_time 总突破limit时间
未开启CPU Burst	2.96ms	986	14.3s	0	0
开启CPU Burst	456us	0	0	469	3.7s

----结束

6.6.6 出口网络带宽保障

出口网络带宽保障通过设置网络优先级实现，具有如下优点：

- 平衡在线业务与离线业务对出口网络带宽的使用，保证在线业务有足够的网络带宽，在线业务触发阈值时，压缩离线业务带宽使用。
- 在线业务所占用的网络资源较少时，离线业务可使用更多带宽；在线业务所占用的网络资源较多时，降低离线业务资源占用量，从而优先保障在线业务的网络带宽。

约束与限制

使用出口网络带宽保障特性需满足以下要求：

- 仅支持Huawei Cloud EulerOS 2.0操作系统的节点。
- 仅支持CCE Turbo集群，且集群版本为v1.23及以上。
- 集群中需要安装Volcano 1.9.0及以上版本的插件，且开启混合部署开关（即将插件高级配置中的colocation_enable设置为true）。
- 开启、修改或者关闭出口网络带宽保障特性，均需要保证Volcano插件处于正常运行状态。
- 对于安装Volcano插件之前节点上已运行的Pod，开启网络带宽保障后需要手动重启Pod才可生效。
- 卸载Volcano插件或关闭混合部署开关不会影响节点上已有的出口网络带宽保障设置。如需关闭该特性，请关闭“网络隔离”开关。
- 使用带宽限速有可能造成协议栈缓存积压。对于UDP等无反压机制的协议场景，可能出现有丢包、ENOBUFS等问题。
- 使用带宽限速会增加离线业务得不到带宽的风险，极端场景可能会出现业务因为带宽不足异常、Pod健康检查失败等问题。
- 出口网络带宽保障的例外场景：
  - 当混部的在线Pod或者是离线Pod使用了**网络带宽限速功能**时，网络带宽限速功能的优先级会高于当前功能。
  - 当Pod使用节点网络（**hostNetwork**）时，使用出口网络保障功能无法生效。

操作步骤

下面介绍如何开启或关闭出口网络带宽保障。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“节点管理”，在需要开启出口网络带宽保障的节点池中，单击“更多 > 混部配置”。

确认“节点池混部开关”及“网络隔离”开关已打开。

步骤3 （可选）调整出口网络带宽保障参数。

说明

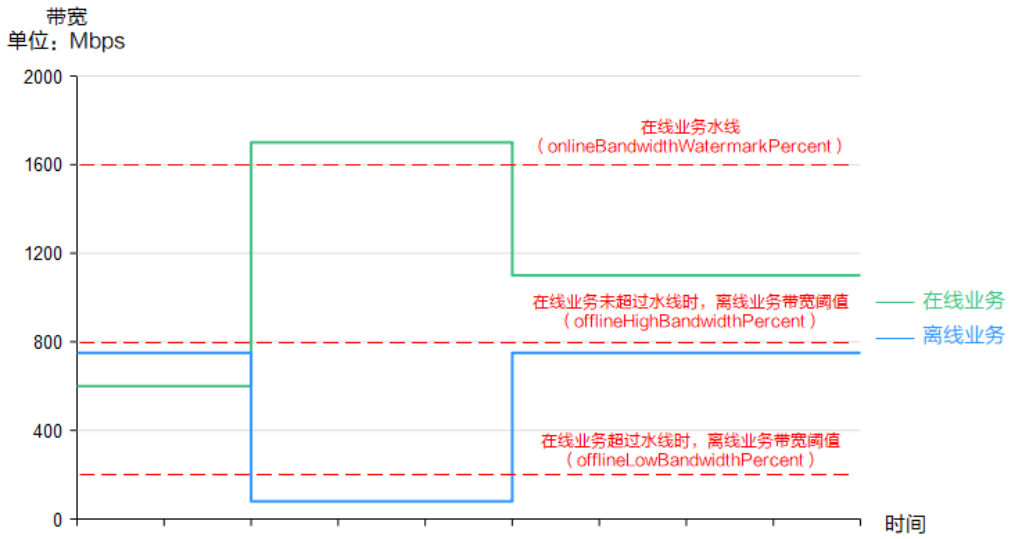
参数编辑后会对集群中所有Huawei Cloud EulerOS 2.0操作系统的节点生效。

表 6-19 网络隔离参数说明

名称	参数	说明	默认值	配置范围
在线作业带宽占比水位线(%)	onlineBandwidthWatermarkPercent	在线业务总带宽水线值与机型的基准带宽的比值，即： *在线业务总带宽水线值 = 节点机型基准带宽 * onlineBandwidthWatermarkPercent/100*	80	配置有效值范围：1-1000 **说明** 由于实际的网络环境可能优于基准带宽，处于在基准带宽和最大带宽之间，因此配置范围支持大于100。
在线作业带宽占用未超水位线时离线作业带宽占比上限(%)	offlineHighBandwidthPercent	在线业务带宽使用未超过水线时，离线业务最高总带宽占用量在机型基准带宽中的占比。 如果同节点的在线业务总带宽的未超过 *节点机型基准带宽 * onlineBandwidthWatermarkPercent/100*，则同节点的离线业务总带宽的不超过 *节点机型基准带宽 * offlineHighBandwidthPercent/100*	40	

名称	参数	说明	默认值	配置范围
在线作业带宽占用超水位线时离线作业带宽占比上限(%)	offlineLowBandwidthPercent	在线业务带宽使用超过水线时，离线业务最高总带宽占用量在机型基准带宽中的占比。 如果同节点的在线业务总带宽的超过 *节点机型基准带宽 * onlineBandwidthWatermarkPercent/100* ，则同节点的离线业务总带宽的不超过 *节点机型基准带宽 * offlineLowBandwidthPercent/100*	10	

图 6-26 出口网络带宽保障示例图



上图中，当在线业务带宽低于在线业务水线时，离线业务的带宽阈值处于一个相对较高的水平，即表示允许离线业务占用一定的带宽；当在线业务带宽超过在线业务水线时，则会相应地调低离线业务带宽阈值，以降低离线业务占用的带宽，预留出更多的带宽供在线业务使用。

----结束

7 网络

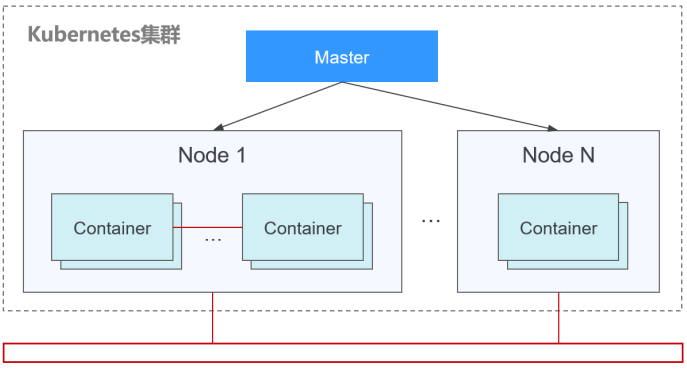
7.1 网络概述

关于集群的网络，可以从如下两个角度进行了解：

- 集群网络是什么样的：集群由多个节点构成，集群中又运行着Pod（容器），每个Pod都需要访问，节点与节点、节点与Pod、Pod与Pod都需要访问。那集群中包含有哪些网络，各自的用处是什么，具体请参见**集群网络构成**。
- 集群中的Pod是如何访问的：访问Pod就是访问容器，也就是访问用户的业务，Kubernetes提供**Service**和**Ingress**来解决Pod的访问问题。本章节根据用户使用场景总结了常见的**网络访问场景**，让您能够在不同使用场景下选择合适的使用方法。

集群网络构成

集群中节点都位于VPC中，节点使用VPC的网络，容器的网络是使用专门的网络插件来管理。



- **节点网络**

  节点网络为集群内主机（节点，图中的Node）分配IP地址，您需要选择VPC中的子网用于CCE集群的节点网络。子网的可用IP数量决定了集群中可以创建节点数量的上限（包括Master节点和Node节点），集群中可创建节点数量还受容器网络的影响，在容器网络模型中会进一步说明。

- **容器网络**

为集群内Pod分配IP地址。CCE继承Kubernetes的IP-Per-Pod-Per-Network的容器网络模型，即每个Pod在每个网络平面下都拥有一个独立的IP地址，Pod内所有容器共享同一个网络命名空间，集群内所有Pod都在一个直接连通的扁平网络中，无需NAT可直接通过Pod的IP地址访问。Kubernetes只提供了如何为Pod提供网络的机制，并不直接负责配置Pod网络；Pod网络的具体配置操作交由具体的容器网络插件实现。容器网络插件负责为Pod配置网络并管理容器IP地址。

当前CCE支持如下容器网络模型。

- 容器隧道网络：容器隧道网络在节点网络基础上通过隧道封装构建的独立于节点网络平面的容器网络平面，CCE集群容器隧道网络使用的封装协议为VXLAN，后端虚拟交换机采用的是openvswitch，VXLAN是将以太网报文封装成UDP报文进行隧道传输。
- VPC网络：VPC网络采用VPC路由方式与底层网络深度整合，适用于高性能场景，节点数量受限于虚拟私有云VPC的路由配额。每个节点将会被分配固定大小的IP地址段。VPC网络由于没有隧道封装的消耗，容器网络性能相对于容器隧道网络有一定优势。VPC网络集群由于VPC路由中配置有容器网段与节点IP的路由，可以支持集群外直接访问容器实例等特殊场景。
- 云原生2.0网络：云原生网络2.0是自研的新一代容器网络模型，深度整合了虚拟私有云VPC的弹性网卡（Elastic Network Interface，简称ENI）和辅助弹性网卡（Sub Network Interface，简称Sub-ENI）的能力，直接从VPC网段内分配容器IP地址，支持ELB直通容器，绑定安全组，绑定弹性公网IP，享有高性能。

不同容器网络模型，容器网络的性能、组网规模、适用场景各不相同，在**容器网络模型对比**章节，将会详细介绍不同容器网络模型的功能特性，了解这些有助于您选择容器网络模型。

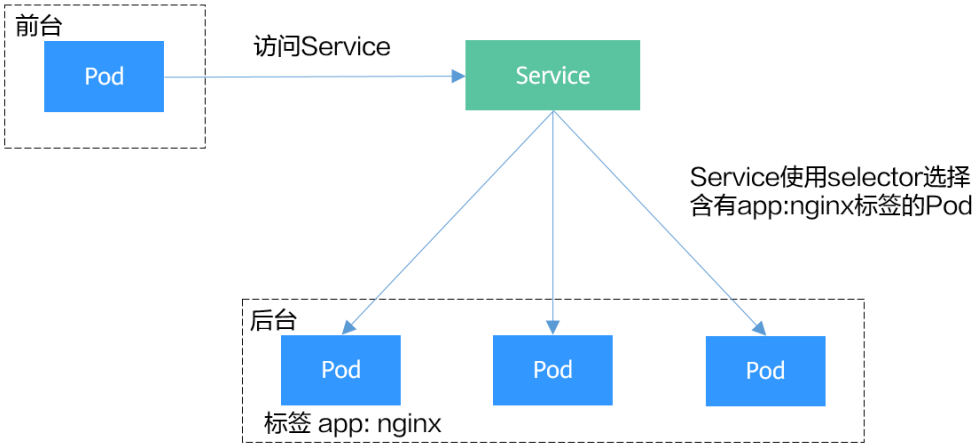
- **服务网络**

  服务（Service）是Kubernetes内的概念，每个Service都有一个固定的IP地址，在CCE上创建集群时，可以指定Service的地址段（即服务网段）。服务网段不能和节点网段、容器网段重叠。服务网段只在集群内使用，不能在集群外使用。

Service

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

图 7-1 通过 Service 访问 Pod



根据创建Service的类型不同，可分成如下模式：

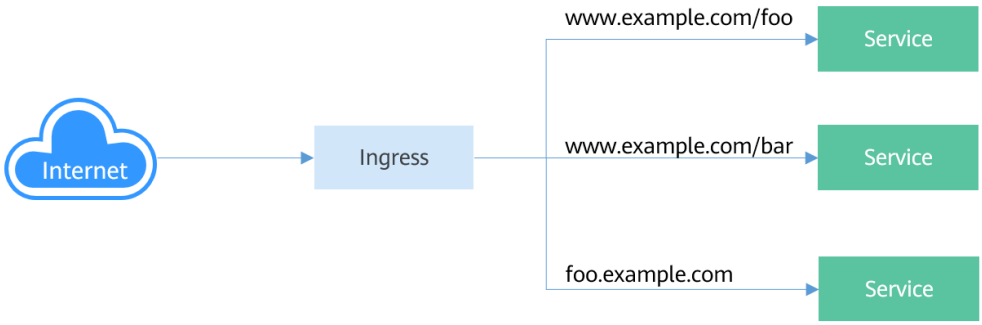
- ClusterIP：用于在集群内部互相访问的场景，通过ClusterIP访问Service。
- NodePort：用于从集群外部访问的场景，通过节点上的端口访问Service。
- LoadBalancer：用于从集群外部访问的场景，其实是NodePort的扩展，通过一个特定的LoadBalancer访问Service，这个LoadBalancer将请求转发到节点的NodePort，而外部只需要访问LoadBalancer。
- DNAT：用于从集群外部访问的场景，为集群节点提供网络地址转换服务，使多个节点可以共享使用弹性IP。

Service的详细介绍请参见**服务概述**。

Ingress

Service是基于四层TCP和UDP协议转发的，而Ingress可以基于七层的HTTP和HTTPS协议转发，可以通过域名和路径做到更细粒度的划分，如下图所示。

图 7-2 Ingress-Service



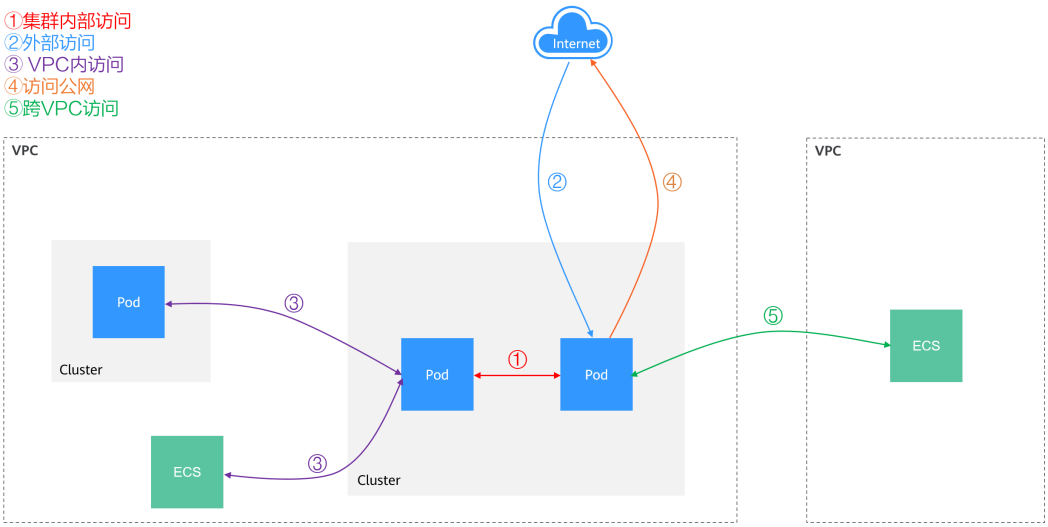
Ingress的详细介绍请参见**路由概述**。

网络访问场景

工作负载网络访问可以分为如下几种场景。

- 从集群内部访问工作负载：创建ClusterIP类型的Service，通过Service访问工作负载。
- 从集群外部访问工作负载：从集群外部访问工作负载推荐使用Service（NodePort类型或LoadBalancer类型）或Ingress访问。
  - 通过公网访问工作负载：需要节点或LoadBalancer绑定公网IP。
  - 通过内网访问工作负载：通过节点或LoadBalancer的内网IP即可访问工作负载。如果跨VPC需要通过对等连接等手段打通不同VPC网络。
- 工作负载访问外部网络：
  - 工作负载访问内网：负载访问内网地址，在不同容器网络模型下有不同的表现，需要注意在对端安全组放通容器网段，具体请参见**容器如何访问VPC内部网络**。
  - 工作负载访问公网：访问公网有几种方法可以实现，一是让容器所在节点绑定公网IP（容器网络模型为VPC网络或容器隧道网络），或给Pod IP绑定公网IP（云原生2.0网络），另一个是通过NAT网关配置SNAT规则，具体请参见**从容器访问公网**。

图 7-3 网络访问示意图



7.2 容器网络

7.2.1 容器网络模型对比

容器网络为集群内Pod分配IP地址并提供网络服务，CCE支持如下几种网络模型，您可在创建集群时进行选择。

- **云原生网络2.0**
- **VPC网络**
- **容器隧道网络**

网络模型对比

表7-1主要介绍CCE所支持的网络模型，您可根据实际业务需求进行选择。

注意

集群创建成功后，网络模型不可更改，请谨慎选择。

表 7-1 网络模型对比

对比维度	容器隧道网络	VPC网络	云原生网络2.0
适用场景	• 对性能要求不高：由于需要额外的VXLAN隧道封装，相对于另外两种容器网络模式，性能存在一定的损耗（约5%-15%）。所以容器隧道网络适用于对性能要求不是特别高的业务场景，比如：Web应用、访问量不大的数据中台、后台服务等。 • 大规模组网：相比VPC路由网络受限于VPC路由条目配额的限制，容器隧道网络没有网络基础设施的任何限制；同时容器隧道网络把广播域控制到了节点级别，容器隧道网络最大可支持2000节点规模。	• 性能要求较高：由于没有额外的隧道封装，相比于容器隧道网络模式，VPC网络模型集群的容器网络性能接近于VPC网络性能，所以适用于对性能要求较高的业务场景，比如：AI计算、大数据计算等。 • 中小规模组网：由于VPC路由网络受限于VPC路由表条目配额的限制，建议集群规模为1000节点及以下。	• 性能要求高：由于云原生网络2.0直接使用VPC网络构建容器网络，容器通信不需要进行隧道封装和NAT转换，所以适用于对带宽、时延要求极高的业务场景，比如：线上直播、电商抢购等。 • 大规模组网：云原生网络2.0当前最大可支持2000个ECS节点，10万个Pod。
核心技术	OVS	IPVlan，VPC路由	VPC弹性网卡/弹性辅助网卡
适用集群	CCE Standard集群	CCE Standard集群	CCE Turbo集群
容器网络隔离	Pod支持Kubernetes原生NetworkPolicy	否	Pod支持使用安全组隔离
ELB对接Pod	ELB对接Pod需要通过节点NodePort转发	ELB对接Pod需要通过节点NodePort转发	使用独享型ELB时可直接对接Pod 使用共享型ELB对接Pod需要通过节点NodePort转发

对比维度	容器隧道网络	VPC网络	云原生网络2.0
容器IP地址管理	• 需设置单独的容器网段 • 按节点划分容器地址段，动态分配（地址段分配后可动态增加）	• 需设置单独的容器网段 • 按节点划分容器地址段，静态分配（节点创建完成后，地址段分配即固定，不可更改）	容器网段从VPC子网划分，无需设置单独的容器网段
网络性能	基于VxLAN隧道封装，有一定性能损耗。	无隧道封装，跨节点通过VPC路由器转发，性能较好，可媲美主机网络，但存在NAT转换损耗。	容器网络与VPC网络融合，性能无损耗。
组网规模	最大可支持2000节点	受限于VPC路由表能力，适合中小规模组网，建议规模为1000节点及以下。 VPC网络模式下，集群每添加一个节点，会在VPC的路由表中添加一条路由（包括默认路由表和自定义路由表），因此集群本身规模受VPC路由表上限限制，创建前请提前评估集群规模。路由表配额请参见**使用限制**。	最大可支持2000节点 由于云原生网络2.0集群中的容器从VPC网段内分配IP地址，消耗VPC的地址空间，实际支持规模受限于VPC子网网段大小，因此创建前请提前评估集群规模。

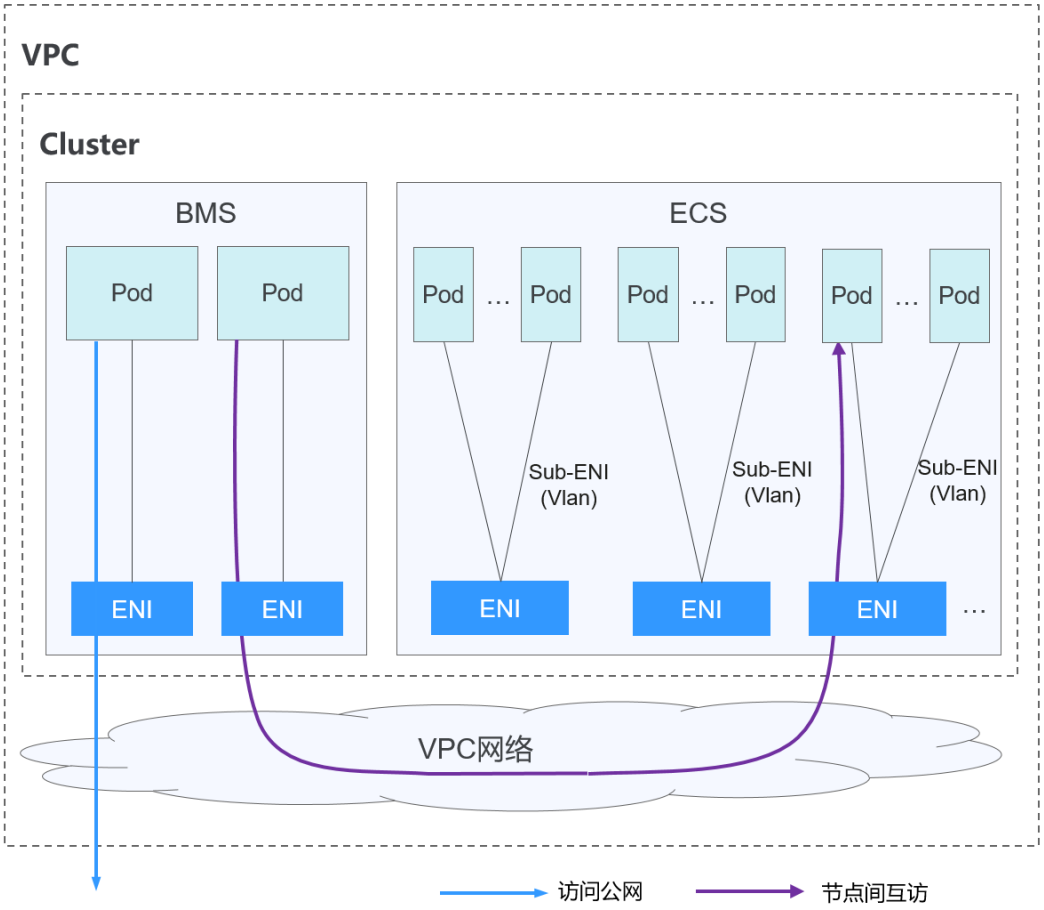
7.2.2 云原生网络 2.0 模型

7.2.2.1 云原生网络 2.0 模型说明

云原生网络 2.0 模型

云原生网络2.0是自研的新一代容器网络模型，深度整合了虚拟私有云VPC的弹性网卡（Elastic Network Interface，简称ENI）和辅助弹性网卡（Sub Network Interface，简称Sub-ENI）的能力，将Pod直接绑定弹性网卡或辅助弹性网卡，使每个Pod在VPC内均拥有独立的IP地址，且支持ELB直通容器、Pod绑定安全组、Pod绑定弹性公网IP等特性。由于不需要使用容器隧道封装和NAT地址转换，云原生网络2.0模型与容器隧道网络模型和VPC网络模型相比具有比较高的网络性能。

图 7-4 云原生网络 2.0



在云原生网络2.0模型的集群中，Pod依赖弹性网卡/辅助弹性网卡对外进行网络通信：

- 裸金属节点上运行的Pod使用ENI网卡。
- ECS节点上运行的Pod使用Sub-ENI网卡，Sub-ENI网卡通过VLAN子接口挂载在ECS的ENI网卡上。
- 由于需要为每个Pod绑定网卡，因此节点上可运行的Pod数量上限由该节点的能绑定的网卡个数和网卡端口数决定。
- 节点内Pod间通信、跨节点Pod间通信、Pod访问集群外网络的情况均直接通过VPC的弹性网卡/弹性辅助网卡进行流量转发。

约束与限制

仅CCE Turbo集群支持使用云原生网络2.0模型。

优缺点

优点

- 基于VPC构建容器网络，每个Pod具有独立的网卡及IP地址，易于排查网络问题，且具有最高的性能表现。
- 在同一个VPC内，由于Pod直接绑定VPC网卡，集群外部的资源可以与集群内部的容器直接进行网络通信。

> **说明**
>
> 同理，如果该VPC和其他VPC或数据中心网络环境连通，在网段不冲突的情况下，其他VPC或数据中心所属的资源也可以与集群内部的容器直接进行网络通信。

- Pod可直接使用VPC提供的负载均衡、安全组、弹性公网IP等能力。

缺点

由于容器网络基于VPC构建，每个Pod都会从VPC网段内分配IP地址，消耗VPC的地址空间，创建集群前需要合理规划好容器网段。

适用场景

- 性能要求高：由于云原生网络2.0直接使用VPC网络构建容器网络，容器通信不需要进行隧道封装和NAT转换，所以适用于对带宽、时延要求极高的业务场景，比如：线上直播、电商抢购等。
- 大规模组网：云原生网络2.0当前最大可支持2000个ECS节点，10万个Pod。

容器 IP 地址管理

云原生网络2.0下的BMS节点和ECS节点分别使用的是弹性网卡和辅助弹性网卡：

- Pod的IP地址从配置给容器网络的VPC子网上直接分配，无需为节点分配一个单独的小网段。
- ECS节点添加到集群中，先绑定用于承载辅助弹性网卡的弹性网卡，待弹性网卡绑定完成后，即可绑定辅助弹性网卡。
- ECS节点上绑定的弹性网卡数：**v1.19.16-r40、v1.21.11-r0、v1.23.9-r0、v1.25.4-r0、v1.27.1-r0及以上版本的集群中该值为1；上述版本以下的集群中该值为节点最多可绑定的辅助弹性网卡数/64，向上取整。**
- ECS节点上绑定的总网卡数：**用于承载辅助弹性网卡的弹性网卡数+当前Pod使用的辅助弹性网卡数+预热的辅助弹性网卡数**。
- BMS节点上绑定的网卡数：**当前Pod使用的弹性网卡数+预热的弹性网卡数**。
- Pod创建时，优先从节点的预热网卡池中随机分配一个可用的网卡。
- Pod删除时，网卡释放回节点的预热网卡池。
- 节点删除时，将释放节点上所有已绑定的网卡（弹性网卡释放回集群预申请的网卡池，辅助弹性网卡直接删除）。

云原生2.0网络目前支持以下网卡预热策略：**节点容器网卡动态预热策略**和**节点绑定容器网卡数总量高低水位策略（废弃中）**。使用场景如下表所示：

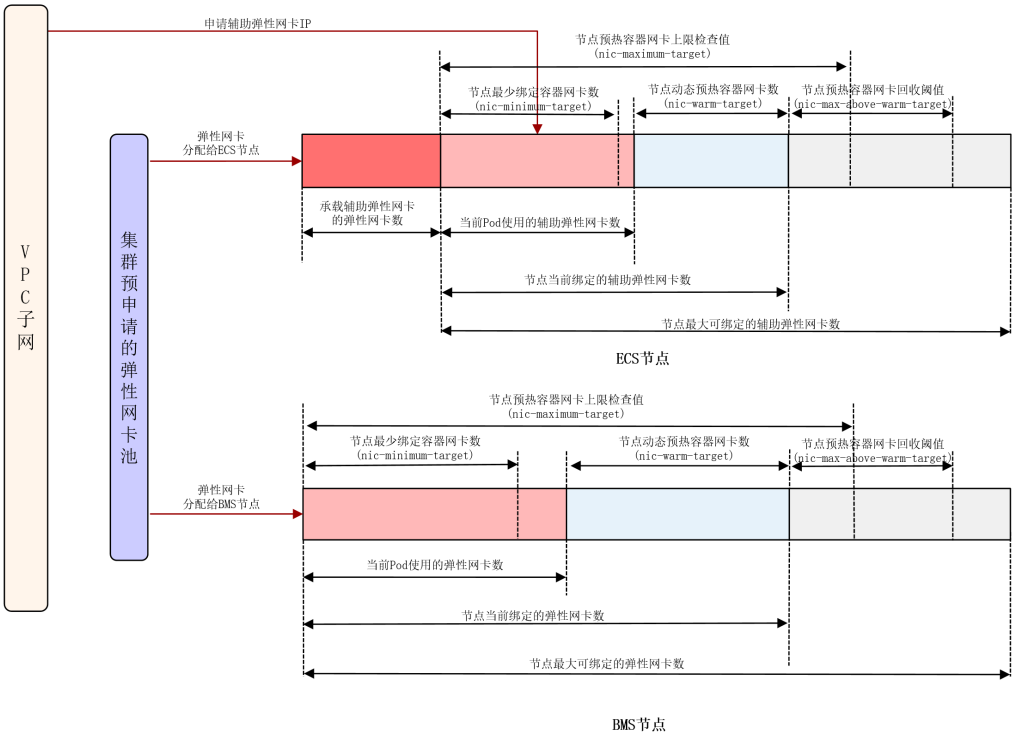
表 7-2 容器网卡预热策略对比表

容器网卡预热策略	节点容器网卡动态预热策略（默认策略）	节点绑定容器网卡数总量高低水位策略（废弃中）
管理策略	节点最少绑定容器网卡数（nic-minimum-target）：保障节点最少有多少张容器网卡绑定在节点上（预热未被Pod使用+已被Pod使用） 节点预热容器网卡上限检查值（nic-maximum-target）：当节点绑定的容器网卡数超过该值，不再主动预热容器网卡 节点动态预热容器网卡数（nic-warm-target）：保障节点至少预热的容器网卡数 节点预热容器网卡回收阈值(nic-max-above-warm-target)：只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值时，才会触发预热容器网卡的解绑回收	节点绑定容器网卡数低水位：保障节点至少会绑定多少张网卡（未被Pod使用+已被Pod使用） 节点绑定容器网卡数高水位：保障节点至多会绑定多少张网卡，超过该值会尝试解绑未被使用的空闲网卡
适用场景	在尽可能提高IP资源利用率的前提下，尽可能加快Pod的启动速度，适用于容器网段IP地址数紧张的场景 通过合理配置上述四个参数，可适用于各种业务场景，详情请参见**CCE Turbo配置容器网卡动态预热**。	适用于容器网段IP地址数充足，且节点上Pod数变化剧烈，但固定在某个范围的场景

说明

- 1.19.16-r2、1.21.5-r0、1.23.3-r0到1.19.16-r4、1.21.7-r0、1.23.5-r0之间的集群版本只支持nic-minimum-target和nic-warm-target两个容器网卡动态预热参数配置，绑定网卡数总量高低水位配置优先级高于容器网卡动态预热配置。
- 1.19.16-r4、1.21.7-r0、1.23.5-r0、1.25.1-r0及以上集群版本支持全部四个容器网卡动态预热参数配置，容器网卡动态预热配置优先级高于绑定网卡数总量高低水位配置。

图 7-5 节点容器网卡动态预热策略



针对节点容器网卡动态预热策略，CCE提供了四个参数配置，您可以根据业务规划，集群规模以及节点上可绑定的网卡数，合理设置这四个参数。

表 7-3 容器网卡动态预热参数

容器网卡动态预热参数	默认值	参数说明	配置建议
节点最少绑定容器网卡数(nic-minimum-target)	10	保障节点最少有多少张容器网卡绑定在节点上，支持数值跟百分比两种配置方式。 ● 数值配置：参数值需为正整数。例如10，表示节点最少有10张容器网卡绑定在节点上。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。 ● 百分比配置：参数值范围为1%-100%。例如10%，如果节点容器网卡配额128，表示节点最少有12张（向下取整）容器网卡绑定在节点上。 建议nic-minimum-target与nic-maximum-target为同类型的配置方式（同采用数值配置或同采用百分比配置）。	建议配置为大部分节点平时日常运行的Pod数。

容器网卡动态预热参数	默认值	参数说明	配置建议
节点预热容器网卡上限检查值(nic-maximum-target)	0	当节点绑定的容器网卡数超过节点预热容器网卡上限检查值(nic-maximum-target)，不再主动预热容器网卡。 当该参数大于等于节点最少绑定容器网卡数(nic-minimum-target)时，则开启预热容器网卡上限值检查；反之，则关闭预热容器网卡上限值检查。支持数值跟百分比两种配置方式。 ● 数值配置：参数值需为正整数。例如0，表示关闭预热容器网卡上限值检查。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。 ● 百分比配置：参数值范围为1%-100%。例如50%，如果节点容器网卡配额128，表示节点预热容器网卡上限检查值64（向下取整）。 建议nic-minimum-target与nic-maximum-target为同类型的配置方式（同采用数值配置或同采用百分比配置）。	建议配置为大部分节点平时最多运行的Pod数。
节点动态预热容器网卡数(nic-warm-target)	2	保障节点至少预热的容器网卡数，只支持数值配置。 当 节点动态预热容器网卡数(nic-warm-target) + 节点当前绑定的容器网卡数 大于 节点预热容器网卡上限检查值(nic-maximum-target) 时，只会预热nic-maximum-target与节点当前绑定的容器网卡数的差值。	建议配置为大部分节点日常10s内会瞬时弹性扩容的Pod数。
节点预热容器网卡回收阈值(nic-max-above-warm-target)	2	只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值 时，才会触发预热容器网卡的解绑回收。只支持数值配置。 ● 调大此值会减慢空闲容器网卡的回收，加快Pod的启动速度，但会降低IP地址的利用率，特别是在IP地址紧张的场景，**请谨慎调大**。 ● 调小此值会加快空闲容器网卡的回收，提高IP地址的利用率，但在瞬时大量Pod激增的场景，部分Pod启动会稍微变慢。	建议配置为大部分节点日常在分钟级时间范围内会频繁弹性扩容缩容的Pod数 - 大部分节点日常10s内会瞬时弹性扩容的Pod数。

说明

上述容器网卡动态预热参数支持集群级别的全局配置和节点池级别的差异化配置，其中节点池级别的容器网卡动态预热配置优先级高于集群级别的容器网卡动态预热配置。

容器网络组件会为每个节点维护一个可弹性伸缩的预热容器网卡池，定时（约10s一次）检测并计算**需要绑定的预热容器网卡数**或**需要解绑的空闲容器网卡数**：

- **需要绑定的预热容器网卡数 = min(nic-maximum-target - 当前绑定的容器网卡总数，max(nic-minimum-target - 当前绑定的容器网卡总数，nic-warm-target - 当前空闲的容器网卡数))**
- **需要解绑的空闲容器网卡数 = min(当前空闲的容器网卡数 - nic-warm-target - nic-max-above-warm-target，当前绑定的容器网卡总数 - nic-minimum-target)**

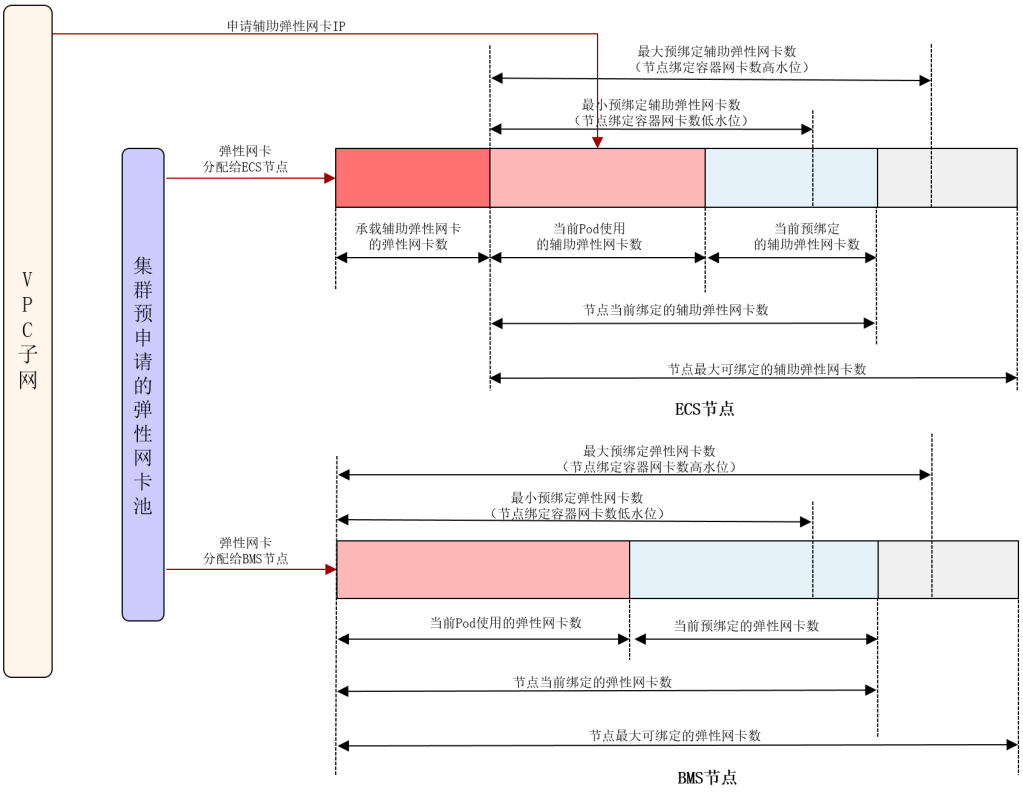
节点上当前预热的容器网卡数稳态后会维持在以下区间内：

- **当前预热的容器网卡数区间最小值 = min(max(nic-minimum-target - 当前绑定的容器网卡总数，nic-warm-target), nic-maximum-target - 当前绑定的容器网卡总数)**
- **当前预热的容器网卡数区间最大值 = max(nic-warm-target + nic-max-above-warm-target, 当前绑定的容器网卡总数 - nic-minimum-target)**

Pod创建时，优先从节点的预热容器网卡池中顺序分配（最早未被使用的）一张空闲的容器网卡，如没有可用的空闲网卡，会新创建一张网卡（辅助弹性网卡）或 新绑定一张网卡（弹性网卡）以分配给该Pod。

Pod删除时，对应的容器网卡先释放回节点的预热容器网卡池，2分钟冷却时间内可供下一个Pod循环使用，超过2分钟冷却时间后且节点预热容器网卡池计算出需要释放该容器网卡，才会释放该容器网卡。

图 7-6 节点绑定容器网卡数总量高低水位策略



针对总量高低水位算法，CCE提供了一个配置参数，您可以根据业务规划，集群规模以及节点上可绑定的网卡数，合理设置这个参数：

- 节点绑定容器网卡数低水位：默认为0，保障节点至少会绑定多少张网卡（未被Pod使用+已被Pod使用）。ECS节点预绑定低水位网卡数=节点绑定网卡数低水位*节点总辅助弹性网卡数；BMS节点预绑定低水位网卡数=节点绑定网卡数低水位*节点总弹性网卡数。
- 节点绑定容器网卡数高水位：默认为0，保障节点至多会绑定多少张网卡，超过该值会尝试解绑未被使用的空闲网卡。ECS节点预绑定高水位网卡数=节点绑定网卡数高水位*节点总辅助弹性网卡数；BMS节点预绑定高水位网卡数=节点绑定网卡数高水位*节点总弹性网卡数。

容器网络组件会为每个节点维护一个可弹性伸缩的容器网卡池：

- 当**已绑定容器网卡数量（Pod使用的容器网卡数+预绑定的容器网卡数）< 预绑定低水位容器网卡数**时，会绑定网卡直到节点上**已绑定容器网卡数量（Pod使用的容器网卡数+预绑定的容器网卡数）=预绑定低水位容器网卡数**。
- 当**已绑定容器网卡数量（Pod使用的容器网卡数+预绑定的容器网卡数）> 预绑定高水位容器网卡数**，且 **节点预绑定的容器网卡数>0** 时，会定时释放预绑定的容器网卡（超过2分钟未被使用的空闲网卡），直到**Pod使用的容器网卡数+预绑定的容器网卡数=节点预绑定高水位容器网卡数** 或 **Pod使用的容器网卡数 > 节点预绑定高水位容器网卡数** 且 **节点预绑定的容器网卡数=0**。

网段规划建议

在**集群网络构成**中介绍集群中网络地址可分为集群网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- 集群所在VPC下所有子网（包括扩展网段子网）不能和服务网段冲突。
- 保证**每个网段有足够的IP地址可用**。
  - 集群网段的IP地址要与集群规模相匹配，否则会因为IP地址不足导致无法创建节点。
  - 容器网段的IP地址要与业务规模相匹配，否则会因为IP地址不足导致无法创建Pod。

云原生网络2.0模型下，由于容器网段与节点网段共同使用VPC下的网络地址，建议容器子网与节点子网不要使用同一个子网，否则容易出现IP资源不足导致容器或节点创建失败的情况。

另外云原生网络2.0模型下容器网段支持在创建集群后增加子网，扩展可用IP数量，此时需要注意增加的子网不要与容器网段其他子网存在网络冲突。

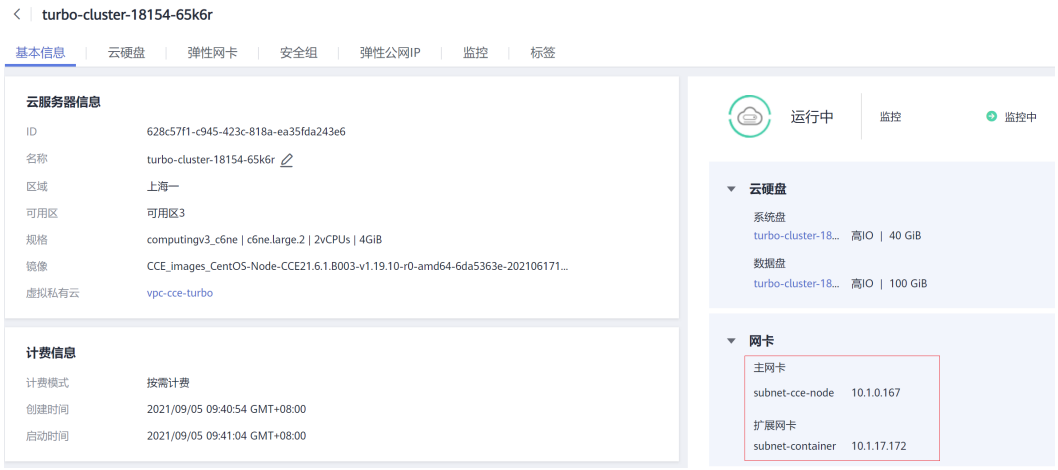
图 7-7 网段配置（创建集群时配置）

云原生网络 2.0 访问示例

本示例中，已创建一个CCE Turbo集群，且集群中包含3个ECS节点。

在ECS控制台中查看其中一个节点的基本信息，在网卡信息中可以看到节点上绑定了一个主网卡和扩展网卡，这两个网卡都属于弹性网卡，其中扩展网卡的IP地址属于容器网络网段，用于给Pod挂载辅助弹性网卡Sub-ENI。

图 7-8 节点网卡



在云原生网络2.0集群中创建工作负载的访问示例如下。

步骤1 使用kubectl命令行工具连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在集群中创建一个Deployment。

创建deployment.yaml文件，文件内容示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
```



```
  name: example
  namespace: default
spec:
  replicas: 6
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: container-0
          image: 'nginx:perl'
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      imagePullSecrets:
        - name: default-secret
```

创建该工作负载：

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

步骤3 查看已运行的Pod。

```
kubectl get pod -owide
```

回显如下：

```
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE         NOMINATED NODE
READINESS GATES
example-5bdc5699b7-54v7g   1/1     Running   0          7s    10.1.18.2    10.1.0.167   <none>           <none>
example-5bdc5699b7-6dzx5   1/1     Running   0          7s    10.1.18.216  10.1.0.186   <none>           <none>
example-5bdc5699b7-gq7xs   1/1     Running   0          7s    10.1.16.63   10.1.0.144   <none>           <none>
example-5bdc5699b7-h9rvb   1/1     Running   0          7s    10.1.16.125  10.1.0.167   <none>           <none>
example-5bdc5699b7-s9fts   1/1     Running   0          7s    10.1.16.89   10.1.0.144   <none>           <none>
example-5bdc5699b7-swq6q   1/1     Running   0          7s    10.1.17.111  10.1.0.167   <none>
<none>
```

这里Pod的IP都是Sub-ENI，挂载在节点的ENI上（扩展网卡）。

例如10.1.0.167节点对应的扩展网卡IP地址为10.1.17.172。在弹性网卡控制台上查看IP地址为10.1.17.172的扩展网卡上挂载了3个Sub-ENI，均为该节点上运行的Pod IP。

图 7-9 Pod 网卡

步骤4 登录同一VPC内的云服务器，从集群外直接访问Pod的IP，本示例中为*10.1.18.2*。

```
curl 10.1.18.2
```

回显如下，说明可正常访问工作负载应用：

```
<!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>
```

----结束

CCE Turbo 集群 Pod 批量创建性能说明

CCE Turbo集群的Pod容器网卡申请自VPC的弹性网卡或者辅助弹性网卡，目前Pod与网卡（弹性网卡或辅助弹性网卡）的关联操作发生在Pod调度完成之后，Pod创建的速度受网卡创建与绑定速度的影响，具体限制如下表所示。

表 7-4 容器网卡创建耗时

节点类型	网卡类型	可支持的最大网卡数	网卡绑定到节点上的操作	网卡可用耗时	并发控制	节点上的容器网卡默认预热配置
ECS节点	辅助弹性网卡	256	指定该节点的弹性网卡创建辅助弹性网卡	1s以内	租户级别：600/分钟	1.19.16-r2、1.21.5-r0、1.23.3-r0之前的集群版本：容器网卡不预热 1.19.16-r2、1.21.5-r0、1.23.3-r0到1.19.16-r4、1.21.7-r0、1.23.5-r0之间的集群版本：容器网卡动态预热（nic-minimum-target=10；nic-warm-target=2） 1.19.16-r4、1.21.7-r0、1.23.5-r0、1.25.1-r0及以上集群版本：容器网卡动态预热（nic-minimum-target=10；nic-maximum-target=0；nic-warm-target=2；nic-max-above-warm-target=2）

节点类型	网卡类型	可支持的最大网卡数	网卡绑定到节点上的操作	网卡可用耗时	并发控制	节点上的容器网卡默认预热配置
BMS节点	弹性网卡	128	节点绑定弹性网卡	20s-30s	节点级别：3并发	1.19.16-r4、1.21.7-r0、1.23.5-r0之前的集群版本：容器网卡总数高低水位预热（nic-threshold=0.3:0.6） 1.19.16-r4、1.21.7-r0、1.23.5-r0、1.25.1-r0、1.28.1-r0及以上集群版本：容器网卡动态预热（nic-minimum-target=10；nic-maximum-target=0；nic-warm-target=2；nic-max-above-warm-target=2）

说明

容器网卡预热会提前消耗容器子网的IP地址，进而影响集群可运行的Pod数规模，请根据业务规模合理规划配置容器网卡动态预热参数，详情请参见**CCE Turbo配置容器网卡动态预热**。

ECS节点创建Pod说明（采用辅助弹性网卡）

- 当Pod调度的节点上没有可用的已经预热的容器网卡时，会调用辅助弹性网卡的创建API，在该节点的一个弹性网卡上创建一个辅助弹性网卡；并把该辅助弹性网卡分配给该Pod。
- 当Pod调度的节点上有可用的已经预热的容器网卡时，会选择创建时间最长且未使用的一张辅助弹性网卡分配给该Pod。
- 受限于辅助弹性网卡的租户并发创建速度，容器网卡不预热的场景下，每分钟最多创建成功600个Pod；如果有更高的弹性要求，可根据业务场景合理配置容器网卡动态预热参数。

BMS节点创建Pod说明（采用弹性网卡）

- 当Pod调度的节点上没有可用的已经预热的容器网卡时，会调用节点绑定网卡的API，在该节点上绑定一个弹性网卡；并把该弹性网卡分配给该Pod。目前BMS节点绑定一张弹性网卡直至完全可用大约耗时在20s到30s不等。
- 当Pod调度的节点上有可用的已经预热的容器网卡时，会选择创建时间最长且未使用的一张弹性网卡分配给该Pod。
- 受限于BMS节点绑定弹性网卡的速度，容器网卡不预热的场景下，同一节点的Pod启动速度为：3个/20秒；所以针对BMS节点，强烈建议用户配置容器网卡全预热。

7.2.2.2 配置集群容器子网

操作场景

当创建CCE Turbo集群时设置的容器子网太小，无法满足业务扩容需求时，您通过扩展集群容器子网的方法来解决。本文介绍如何为CCE Turbo集群添加容器子网。

约束与限制

- 仅支持v1.19及以上版本的CCE Turbo集群。

为 CCE Turbo 集群添加容器子网

步骤1 登录CCE控制台，单击CCE Turbo集群名称，进入集群。

步骤2 在“总览”页面，找到“网络信息”版块，并单击“添加”。

图 7-10 添加容器子网

步骤3 选择同一VPC下的容器子网，您可一次性添加多个容器子网。如没有其他可用的容器子网，可前往VPC控制台创建。

图 7-11 选择容器子网

步骤4 单击“确定”。

----结束

为 CCE Turbo 集群删除容器子网

说明

v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本的集群支持删除容器子网。

步骤1 登录CCE控制台，单击CCE Turbo集群名称，进入集群。

步骤2 在“配置中心”页面，切换至“网络配置”页签。

步骤3 在“容器网络配置”中，单击default-network（默认容器子网）后的“更新”。

步骤4 取消选择需要删除的容器子网，单击“确定”。

须知

- 删除容器子网属高危操作，请确保当前集群中没有已经使用待删除子网的网卡，包含Pod正在使用和集群预热的网卡。

  您可以复制需要删除的子网ID，在**弹性网卡**页面的“弹性网卡”和“辅助弹性网卡”列表中，通过子网ID进行筛选，如果筛选出的网卡“名称”或者“描述”里包含当前集群的ID，表示网卡被集群占用。
- 删除子网后，集群节点安全组不会自动清理该子网对应的规则，请确认集群中不存在该子网下的网卡后手动清理安全组规则。

----结束

7.2.2.3 使用注解为 Pod 绑定安全组

使用场景

云原生网络2.0网络模式下，Pod使用的是VPC的弹性网卡/辅助弹性网卡，可以通过配置Pod的annotation为Pod配置安全组。

支持两种方式的安全组配置：

- Pod的网卡使用annotation配置的安全组，对应annotation配置：yangtse.io/security-group-ids。
- Pod的网卡在使用已有安全组的基础上，额外再增加annotation配置的安全组，对应annotation配置：yangtse.io/additional-security-group-ids。

说明

通过注解（yangtse.io/security-group-ids）为Pod绑定的安全组优先级高于**使用安全组策略**（SecurityGroup）和**集群容器网络配置**（NetworkAttachmentDefinition）中的安全组。

前提条件

您已创建一个CCE Turbo集群，且集群版本满足以下要求：

- v1.23集群：v1.23.16-r0及以上版本
- v1.25集群：v1.25.11-r0及以上版本
- v1.27集群：v1.27.8-r0及以上版本

- v1.28集群：v1.28.6-r0及以上版本
- v1.29集群：v1.29.2-r0及以上版本
- v1.29以上版本集群

通过 kubectl 命令行设置

- 创建一个配置安全组的工作负载，最终Pod关联安全组以annotation中的配置为准：

  说明

  如果Pod已绑定安全组，则会被覆盖。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/security-group-ids: *****  # 安全组ID，多个安全组以逗号分隔
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

- 为工作负载额外增加一个安全组：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/additional-security-group-ids: *****  # 安全组ID，多个安全组以逗号分隔
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
```



```
        cpu: 100m
        memory: 200Mi
imagePullSecrets:
  - name: default-secret
```

表 7-5 为 Pod 配置安全组的 annotation 说明

annotation	参数说明	取值范围
yangtse.io/security-group-ids	为Pod配置安全组，最终Pod的安全组将以该annotation中的配置为准。即如果Pod已有安全组，原安全组将会被覆盖。	安全组ID，数量不多于5个，多个安全组之间以英文逗号分隔。
yangtse.io/additional-security-group-ids	为Pod添加安全组配置，即在已有的配置上增加指定的安全组。	安全组ID，安全组ID数量和已有的安全组数量相加不多于5个，多个安全组之间以英文逗号分隔。

7.2.2.4 使用安全组策略为工作负载绑定安全组

云原生网络2.0网络模式下，Pod使用的是VPC的弹性网卡/辅助弹性网卡，可直接绑定安全组，绑定弹性公网IP。为方便用户在CCE内直接为Pod关联安全组，CCE新增了一个名为SecurityGroup的自定义资源对象。通过SecurityGroup资源对象，用户可对工作负载实现自定义的安全隔离诉求。

说明

使用安全组策略（SecurityGroup）为Pod绑定的安全组优先级高于**集群容器网络配置**（NetworkAttachmentDefinition）中的安全组。

约束与限制

- v1.19及以上的CCE Turbo集群支持此功能，v1.19以下版本CCE Turbo集群需要升级到v1.19及以上版本后才能启用此功能。
- 1个工作负载最多可绑定5个安全组。

通过界面创建

步骤1 登录CCE控制台，单击集群名称，进入集群。

步骤2 在左侧选择“工作负载”，单击工作负载名称。

步骤3 在“安全组策略”页签下，单击“创建”。

步骤4 根据界面提示，配置参数， 具体如表7-6所示。

表 7-6 配置参数

参数名称	描述	示例
安全组策略名称	输入安全组策略名称。 请输入1-63个字符，以小写字母开头，由小写字母、数字、连接符（-）组成，且不能以连接符（-）结尾。	security-group
关联安全组	选中的安全组将绑定到选中的工作负载的弹性网卡/辅助弹性网卡上，在下拉框中最多可以选择5条，安全组必选，不可缺省。 如将绑定的安全组未创建，可单击“创建安全组”，完成创建后单击刷新按钮。 **须知** • 最多可选择5个安全组。 • 鼠标悬浮在安全组名称上，可查看安全组的详细信息。	64566556-bd6f-48fb-b2c6-df8f44617953 5451f1b0-bd6f-48fb-b2c6-df8f44617953

步骤5 参数配置后，单击“确定”。

创建完成后页面将自动返回到安全组策略列表页，可以看到新添加的安全组策略已在列表中。

----结束

通过 kubectl 命令行创建

步骤1 请参见通过kubectl连接集群，使用kubectl连接集群。

步骤2 创建一个名为securitygroup-demo.yaml的描述文件。

vi securitygroup-demo.yaml

例如，用户创建如下的SecurityGroup资源对象，给所有的app：nginx工作负载绑定上提前已经创建的64566556-bd6f-48fb-b2c6-df8f44617953，5451f1b0-bd6f-48fb-b2c6-df8f44617953的两个安全组。示例如下：

```
apiVersion: crd.yangtse.cni/v1
kind: SecurityGroup
metadata:
  name: demo
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: nginx
  securityGroups:
  - id: 64566556-bd6f-48fb-b2c6-df8f44617953
  - id: 5451f1b0-bd6f-48fb-b2c6-df8f44617953
```


以上yaml参数说明如表7-7。

表 7-7 参数说明

字段名称	字段说明	必选/可选
apiVersion	表示API的版本号，版本号为crd.yangtse.cni/v1。	必选
kind	创建的对象类别。	必选
metadata	资源对象的元数据定义。	必选
name	SecurityGroup的名称。	必选
namespace	工作空间名称。	必选
spec	用户对SecurityGroup的详细描述的主体部分都在spec中给出。	必选
podSelector	定义SecurityGroup中需要关联安全组的工作负载。	必选
securityGroups	id为安全组的ID。	必选

步骤3 执行以下命令，创建SecurityGroup。

kubectl create -f securitygroup-demo.yaml

回显如下表示已开始创建SecurityGroup

```
securitygroup.crd.yangtse.cni/demo created
```

步骤4 执行以下命令，查看SecurityGroup。

kubectl get sg

回显信息中有创建的SecurityGroup名称为demo，表示SecurityGroup已创建成功。

```
NAME                POD-SELECTOR                    AGE
all-no              map[matchLabels:map[app:nginx]]   4h1m
s001test            map[matchLabels:map[app:nginx]]   19m
demo                map[matchLabels:map[app:nginx]]   2m9s
```

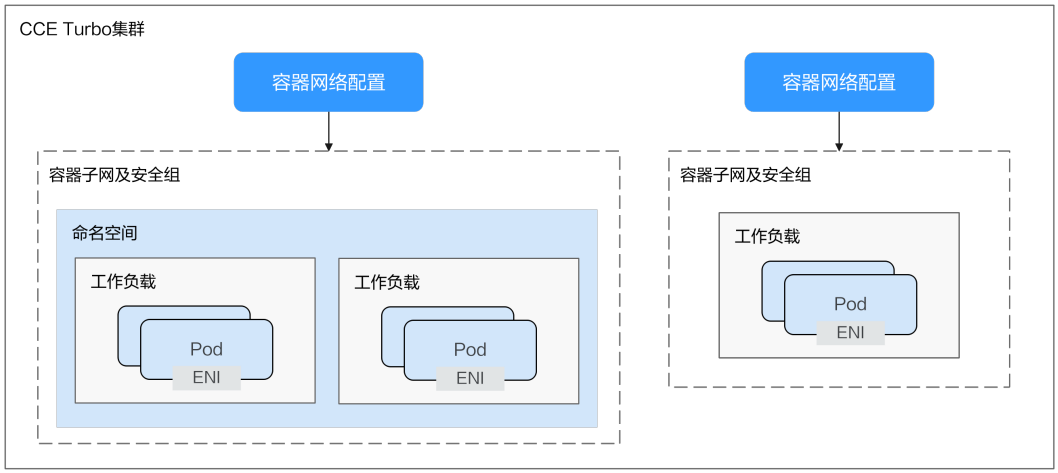
----结束

7.2.2.5 使用容器网络配置为命名空间/工作负载绑定子网及安全组

操作场景

CCE Turbo集群支持以命名空间或工作负载粒度设置容器所在的容器子网及安全组，该功能通过名为NetworkAttachmentDefinition的CRD资源实现。如您想为指定的命名空间或工作负载配置指定的容器子网和安全组，可创建自定义容器网络配置（NetworkAttachmentDefinition），并将该容器网络配置与相应的命名空间或工作负载关联，进而实现业务的子网划分或业务安全隔离的诉求。

图 7-12 自定义容器网络配置示意图



目前容器网络配置（NetworkAttachmentDefinition）支持关联的资源类型对比如下：

表 7-8 关联资源类型对比

维度	容器网络配置（NetworkAttachmentDefinition）关联的资源类型	
	命名空间	工作负载
容器网络划分维度说明	为命名空间关联容器网络配置，该命名空间下创建的所有工作负载使用相同的子网配置跟安全组配置	为工作负载关联容器网络配置，指定了相同容器网络配置的工作负载使用相同的子网配置跟安全组配置
支持的集群版本	仅在CCE Turbo集群中可用，且集群为1.23.8-r0、1.25.3-r0及以上版本	仅在CCE Turbo集群中可用，且集群为1.23.11-r0、1.25.6-r0、1.27.3-r0、1.28.1-r0及以上版本
约束限制	同一个命名空间最多只能被关联到一个容器网络配置	只能指定未关联命名空间的自定义容器网络配置

说明

- Pod使用的容器网络配置优先级如下：Pod直接关联的容器网络配置 > Pod的命名空间关联的容器网络配置 > 集群默认容器网络配置（default-network）。
- 集群中存在默认容器网络配置default-network，对所有未配置自定义容器网络配置的Pod生效，“总览”页面的网络信息中的“默认容器子网”即为default-network中的容器子网。default-network不支持删除。
- 当集群中只有一个容器网络配置，该容器网络配置即为默认容器网络配置；当集群中有多个容器网络配置时，带有名为“yangtse.io/default-network: true”annotation的容器网络配置即为默认容器网络配置。当集群中不存在默认容器网络配置时，未关联任何容器网络配置的Pod创建后由于无法分配到网卡而启动失败。

约束与限制

- 仅默认容器网络配置支持开启容器网卡动态预热。当节点的网卡配额耗尽时，使用自定义容器网络配置的Pod会尝试解绑默认容器网络配置的预热网卡，绑定该自

定义容器网络配置的网卡，此时Pod的启动速度会更慢；因此当您频繁使用自定义容器网络配置时，强烈建议您关闭集群级别的全局容器网卡动态预热。如您还有使用默认容器网络配置的Pod极速弹性述求，请结合调度，合理规划节点池级别的容器网卡动态预热配置。

- 已开启固定IP的工作负载，如果需要关联新的容器网络配置，Pod重建时，固定IP功能会失效。请删除工作负载并释放已经固定的IP，然后重新创建工作负载。
- 如需删除创建的自定义容器网络配置(NetworkAttachmentDefinition)，请先删除对应的命名空间下使用该配置创建的Pod（带有名为“cni.yangtse.io/network-status”的annotation），详情请参见**删除网络配置**。

创建命名空间类型的容器网络配置

创建命名空间类型的容器网络配置后，关联命名空间下创建的所有工作负载使用相同的子网配置跟安全组配置。

通过控制台创建

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，选择“网络配置”页签。

步骤3 查看“自定义容器网络配置”，单击“添加自定义容器网络配置”，在弹窗中配置容器子网和安全组等信息。

- 名称：自定义容器网络配置名称，最长支持63个字符。default-network、default、mgnt0、mgnt1四个名称为系统预留，请勿使用。
- 关联资源类型：自定义容器网络配置关联的资源类型，详情请参见**表7-8**。创建命名空间类型的容器网络配置请选择“命名空间”类型。
- 命名空间：请选择您需要关联的命名空间。不同容器网络配置之间关联的命名空间不可重复。若无命名空间可选请单击后方的“创建命名空间”进行创建。
- 容器子网：请选择子网。若无子网可选请单击后方的“创建子网”进行创建，创建完成后单击刷新按钮。
- 关联安全组：默认为容器ENI安全组，您也可以选择单击后方的“创建安全组”进行创建，创建完成后单击刷新按钮。最多可选择5个安全组。

图 7-13 创建命名空间类型的容器网络配置

步骤4 完成基本配置后单击“确定”，创建完成后页面自动返回到自定义容器网络配置列表，可以看到新创建的容器网络配置已在列表中。

图 7-14 容器网络配置列表

----结束

通过 kubectl 命令行创建

本节说明通过kubectl命令创建命名空间类型的NetworkAttachmentDefinition的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 修改networkattachment-test.yaml。

```
vi networkattachment-test.yaml
```

文件内容如下：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    yangtse.io/project-id: 05e38**
  name: example
  namespace: kube-system
spec:
  config: |
   {
    "type":"eni-neutron",
    "args":{
      "securityGroups":"41891**",
      "subnets":[
        {
          "subnetID":"27d95**"
        }
      ]
    },
    "selector":{
      "namespaceSelector":{
        "matchLabels":{
          "kubernetes.io/metadata.name":"default"
        }
      }
    }
   }
```

表 7-9 关键参数说明

参数	是否必填	参数类型	描述
apiVersion	是	String	表示API的版本号。固定为k8s.cni.cncf.io/v1。

参数	是否必填	参数类型	描述
kind	是	String	创建的对象类别。固定为NetworkAttachmentDefinition。
yangtse.io/project-id	是	String	当前所在Region的项目ID，获取方式请参见**获取项目ID**。
name	是	String	配置名称。
namespace	是	String	配置资源所在命名空间，固定为 kube-system。
config	是	**表2 config字段数据结构说明** Object	配置内容，为json格式的字符串。

表 7-10 config 字段数据结构说明

参数	是否必填	参数类型	描述
type	是	String	固定为eni-neutron。
args	否	**表3 args字段数据结构说明** Object	配置参数。
selector	否	**表7-12**Object	选择该配置所作用的命名空间。

表 7-11 args 字段数据结构说明

参数	是否必填	参数类型	描述
securityGroups	否	String	安全组ID。如果没有对安全组进行规划，请和default-network中的安全组保持一致。 获取方式： 登录虚拟私有云控制台，在左侧导航栏选择“访问控制 > 安全组”，单击安全组名称，在“基本信息”页签下找到“ID”字段复制即可。

参数	是否必填	参数类型	描述
subnets	是	Array of subnetID Objects	容器子网ID列表，至少需填写一个，不可以为空，格式如下： [{"subnetID":"27d95**"},{"subnetID":"827bb**"},{"subnetID":"bdd6b**"}] 同一VPC下非集群的子网ID。 获取方式： 登录虚拟私有云控制台，在左侧导航栏选择“虚拟私有云 > 子网”，单击子网名称，在“基本信息”页签下找到“子网ID”字段复制即可。

表 7-12 selector 字段数据结构说明

参数	是否必填	参数类型	描述

namespaceSelector	否	matchLabels Object	该选择器为Kubernetes标准的选择器，需填写命名空间标签，格式如下：
			"matchLabels":{
			"kubernetes.io/metadata.name":"default"
			}
			不同配置之间的命名空间不可重合。

步骤3 创建NetworkAttachmentDefinition。

```
kubectl create -f networkattachment-test.yaml
```

回显如下，表示NetworkAttachmentDefinition已创建。

```
networkattachmentdefinition.k8s.cni.cncf.io/example created
```

----结束

创建工作负载类型的容器网络配置

创建工作负载类型的容器网络配置后，需要为工作负载关联容器网络配置，指定了相同容器网络配置的工作负载使用相同的子网配置跟安全组配置。

通过控制台创建

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“配置中心”，选择“网络配置”页签。

步骤3 查看“自定义容器网络配置”，单击“添加自定义容器网络配置”，在弹窗中配置容器子网和安全组等信息。

- 名称：自定义容器网络配置名称，最长支持63个字符。default-network、default、mgnt0、mgnt1四个名称为系统预留，请勿使用。
- 关联资源类型：自定义容器网络配置关联的资源类型，详情请参见**表7-8**。创建工作负载类型的容器网络配置请选择“工作负载”类型。

- 容器子网：请选择子网。若无子网可选请单击后方的“创建子网”进行创建，创建完成后单击刷新按钮。
- 关联安全组：默认为容器ENI安全组，您也可以选择单击后方的“创建安全组”进行创建，创建完成后单击刷新按钮。最多可选择5个安全组。

图 7-15 创建工作负载类型的容器网络配置

步骤4 完成基本配置后单击“确定”，创建完成后页面自动返回到自定义容器网络配置列表，可以看到新创建的容器网络配置已在列表中。

图 7-16 容器网络配置列表

步骤5 在创建工作负载时，可选择自定义的容器网络配置。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“高级配置”中选择“网络配置”页签，并选择是否开启指定容器网络配置。
3. 选择一个已有的容器网络配置。如果没有可用的网络配置，可单击“添加自定义容器网络配置”进行创建。

图 7-17 选择容器网络配置

4. 工作负载其余信息都配置完成后，单击“创建工作负载”。

   返回到“配置中心”，在容器网络配置列表里可以看到创建的容器网络配置关联的资源名称。

图 7-18 查看容器网络配置关联的资源

----结束

通过 kubectl 命令行创建

本节说明通过kubectl命令创建工作负载类型的NetworkAttachmentDefinition的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 修改networkattachment-test.yaml。

```
vi networkattachment-test.yaml
```

文件内容如下：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    yangtse.io/project-id: 80d5a**
  name: example
  namespace: kube-system
spec:
  config: |
   {
    "type":"eni-neutron",
    "args":{
      "securityGroups":"f4983**",
      "subnets":[
        {
          "subnetID":"5594b**"
        }
      ]
    }
   }
```


表 7-13 关键参数说明

参数	是否必填	参数类型	描述
apiVersion	是	String	表示API的版本号。固定为k8s.cni.cncf.io/v1。
kind	是	String	创建的对象类别。固定为NetworkAttachmentDefinition。
yangtse.io/project-id	是	String	当前所在Region的项目ID，获取方式请参见**获取项目ID**。
name	是	String	配置名称。
namespace	是	String	配置资源所在命名空间，固定为 kube-system。
config	是	**表2 config字段数据结构说明** Object	配置内容，为json格式的字符串。

表 7-14 config 字段数据结构说明

参数	是否必填	参数类型	描述
type	是	String	固定为eni-neutron。
args	否	**表3 args字段数据结构说明** Object	配置参数。

表 7-15 args 字段数据结构说明

参数	是否必填	参数类型	描述
securityGroups	否	String	安全组ID。如果没有对安全组进行规划，请和default-network中的安全组保持一致。 获取方式： 登录虚拟私有云控制台，在左侧导航栏选择“访问控制 > 安全组”，单击安全组名称，在“基本信息”页签下找到“ID”字段复制即可。

参数	是否必填	参数类型	描述
subnets	是	Array of subnetID Objects	容器子网ID列表，至少需填写一个，不可以为空，格式如下： [{"subnetID":"27d95**"},{"subnetID":"827bb**"},{"subnetID":"bdd6b**"}] 同一VPC下非集群的子网ID。 获取方式： 登录虚拟私有云控制台，在左侧导航栏选择“虚拟私有云 > 子网”，单击子网名称，在“基本信息”页签下找到“子网ID”字段复制即可。

步骤3 创建NetworkAttachmentDefinition。

```
kubectl create -f networkattachment-test.yaml
```

回显如下，表示NetworkAttachmentDefinition已创建。

```
networkattachmentdefinition.k8s.cni.cncf.io/example created
```

步骤4 创建工作负载（此处创建Deployment类型的工作负载），并关联刚创建的容器网络配置。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        yangtse.io/network: "example" # 自定义容器网络配置名称，便于通过label查找所有关联此容器网络配置的Pod
      annotations:
        yangtse.io/network: "example" # 自定义容器网络配置名称
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

- yangtse.io/network：指定的自定义容器网络配置名称，只能指定未关联命名空间的容器网络配置。请在label上同时添加此参数，便于通过label查找所有关联此容器网络配置的Pod。

----结束

验证命名空间/工作负载是否绑定容器网络配置

您可以通过以下步骤确认工作负载是否成功绑定容器网络配置中的子网和安全组。如果需要验证命名空间是否绑定容器网络配置，您可以查看该命名空间中的具体工作负载是否绑定子网和安全组。

步骤1 验证子网是否绑定成功。

1. 查看工作负载中Pod对应的IP地址。

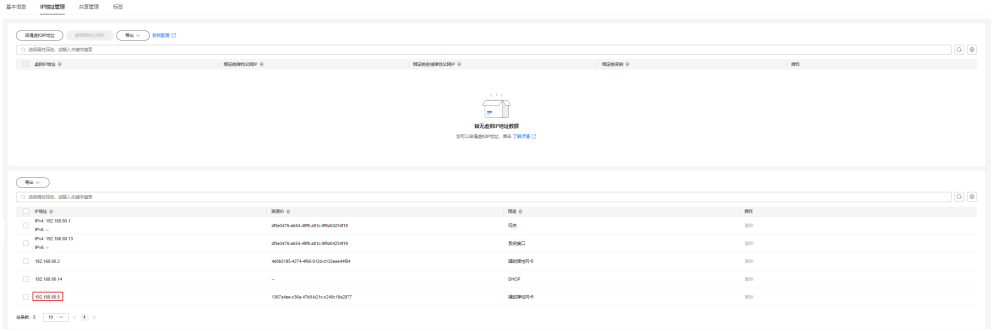
```
kubectl get pod -o wide
```

回显结果如下：

```
NAME                     READY   STATUS    RESTARTS   AGE     IP              NODE
NOMINATED NODE   READINESS GATES
nginx-85dbdb8c5d-ng5bb   1/1     Running   0          5d18h   192.168.60.5    ca50a5ae-e1ef-41c6-
b3fc-6ebcd10a1e07   <none>           <none>
```

2. 进入网络控制台，右侧导航栏单击“虚拟私有云 > 子网”，单击对应的子网名称。
3. 单击“IP地址管理”，“IP地址管理”中若有Pod对应的IP地址则说明子网绑定成功。

图 7-19 查看子网绑定的 IP 地址



步骤2 验证安全组是否绑定成功。

1. 返回网络控制台，右侧导航栏单击“访问控制 > 安全组”，单击对应的安全组名称。
2. 单击“关联实例”，当前页签中单击“辅助弹性网卡”。
3. “辅助弹性网卡”页签中，若私有IP地址列表有Pod对应的IP地址，则说明安全组绑定成功。

图 7-20 查看安全组绑定的 IP 地址

----结束

删除网络配置

您可以查看新添加网络配置的YAML，也可以对新添加的配置进行“删除”操作。

说明

在删除网络配置时，需先删除该配置所对应的容器，否则将删除失败。

1. 执行以下命令筛选集群中使用该配置的Pod（其中example为示例配置名称，请自行替换）：
   ```
   kubectl get pod -A -o=jsonpath="{.items[?(@.metadata.annotations.cni\.yangtse\.io/network-status=='[{\"name\":\"example\"}]')]['metadata.namespace', 'metadata.name']}"
   ```
   返回结果中包含了该配置关联的Pod名字及命名空间。
2. 删除创建该Pod的Owner，其Owner可能为Deployment、StatefulSet、DaemonSet或Job类型的工作负载。

7.2.2.6 为 Pod 配置固定 IP

使用场景

在云原生网络2.0下，会为每个Pod分配用户VPC网络下的一张网卡，支持为StatefulSet工作负载的Pod（容器网卡）固定IP，适用于需要针对具体IP地址做访问控制、服务注册、服务发现、日志审计等场景。

例如，当有一个需要访问云上数据库的StatefulSet类型业务，需要在对云上数据库进行严格的访问控制，只允许该业务进行访问，则可固定该业务的Pod IP，配置云上数据库的安全组只允许该业务的容器IP可进行访问。

约束限制

- 仅以下指定版本的CCE Turbo集群支持用户配置Pod固定IP：
  - v1.23集群：v1.23.7-r0及以上版本
  - v1.25集群：v1.25.3-r0及以上版本
  - v1.25以上版本集群
- 目前只支持StatefulSet类型的Pod或无ownerReferences的Pod固定IP，暂不支持Deployment、DaemonSet等其他类型的工作负载配置Pod固定IP，且不支持已设置**HostNetwork**的Pod配置固定IP。
- 对Pod的IP地址无明确要求的业务不建议配置固定IP，因为配置了固定IP的Pod，Pod重建的耗时会略微变长同时IP地址的利用率会下降。
- 不支持直接修改Pod对象的固定IP的annotations的值，如果直接修改的话，后台不会生效；如有修改诉求，请直接修改对应的StatefulSet工作负载spec.template字段下的annotations配置。
- 当固定IP的Pod重建调度时，如果待调度的节点上没有剩余的网卡配额（预绑定的网卡会占用节点的网卡配额），固定IP的网卡会尝试抢占预绑定的网卡，此时固定IP的Pod启动会略微变慢。使用固定IP的节点，请合理配置节点网卡动态预热以确保不会全预热网卡。

通过 kubectl 命令行设置

您可以通过对StatefulSet添加annotations来设置是否开启Pod固定IP功能，如下所示。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  serviceName: nginx
```



```
replicas: 3
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
    annotations:
      pod.alpha.kubernetes.io/initialized: 'true'
      yangtse.io/static-ip: 'true'
      yangtse.io/static-ip-expire-no-cascading: 'false'
      yangtse.io/static-ip-expire-duration: 5m
  spec:
    containers:
      - name: container-0
        image: nginx:alpine
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
    imagePullSecrets:
      - name: default-secret
```

表 7-16 Pod 固定 IP 的 annotation 配置

annotation	默认值	参数说明	取值范围
yangtse.io/static-ip	false	是否开启Pod固定IP，只有StatefulSet类型的Pod或无ownerReferences的Pod支持，默认不开启。	"false"或"true"
yangtse.io/static-ip-expire-duration	5m	删除固定IP的Pod后，对应的固定IP网卡过期回收的时间间隔。	支持时间格式为Go time type，例如1h30m、5m。关于Go time type，请参见**Go time type**。
yangtse.io/static-ip-expire-no-cascading	false	是否关闭StatefulSet工作负载的级联回收。 默认为false，表示StatefulSet删除后，会级联删除对应的固定IP网卡。如果您需要在删除StatefulSet对象后，在网卡过期回收时间内保留对应的固定IP，用于下一次重建同名的StatefulSet再次使用对应的固定IP，请将该参数设为true。	"false"或"true"

7.2.2.7 为 Pod 配置 EIP

使用场景

云原生网络2.0网络模式下，Pod使用的是VPC的弹性网卡/辅助弹性网卡，可直接绑定弹性公网IP。

为方便用户在CCE内直接为Pod关联弹性公网IP，用户只需在创建Pod时，配置annotation（yangtse.io/pod-with-eip: "true"），弹性公网IP就会随Pod自动创建并绑定至该Pod。

前提条件

您已创建一个CCE Turbo集群，且集群版本满足以下要求：

为Pod配置EIP场景	集群版本要求
EIP跟随Pod创建	• v1.19集群：v1.19.16-r20及以上版本 • v1.21集群：v1.21.10-r0及以上版本 • v1.23集群：v1.23.8-r0及以上版本 • v1.25集群：v1.25.3-r0及以上版本 • v1.25以上版本集群
Pod使用已有EIP	v1.23集群：v1.23.16-r0及以上版本 v1.25集群：v1.25.11-r0及以上版本 v1.27集群：v1.27.8-r0及以上版本 v1.28集群：v1.28.6-r0及以上版本 v1.29集群：v1.29.2-r0及以上版本 v1.29以上版本集群

约束限制

- 绑定EIP的Pod，如果要被公网成功访问，需要添加放通相应请求流量的安全组规则。
- 单个Pod只能绑定单个EIP。
- 创建Pod时，可指定相关的annotation配置EIP的属性，创建完成后，更新EIP相关的annotation均无效。
- 与Pod关联的EIP不要通过弹性公网IP的console或API直接操作（修改名称/删除/解绑/绑定/转包周期等操作），否则可能导致EIP功能异常。
- 自动创建的EIP被手动删除后，会导致网络异常，需要重建Pod。

EIP 跟随 Pod 创建

创建Pod时，填写pod-with-eip的annotation后，EIP会随Pod自动创建并绑定至该Pod。

说明

- 自动创建EIP时，系统会自动为该EIP添加集群ID、命名空间、Pod名称的标签。
- 当自动创建EIP的Pod被删除时，自动创建的EIP会随Pod一起被删除。

以下示例创建一个名为nginx的无状态负载，EIP将随Pod自动创建并绑定至Pod。具体字段含义请参见**表7-18**。

- 创建Deployment时自动创建**独占带宽**类型的EIP，无需指定带宽ID，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/pod-with-eip: "true"  # EIP跟随Pod创建
        yangtse.io/eip-bandwidth-size: "5"  # EIP带宽
        yangtse.io/eip-network-type: 5_bgp  # EIP类型
        yangtse.io/eip-charge-mode: bandwidth  # EIP计费模式
        yangtse.io/eip-bandwidth-name: <eip_bandwidth_name>  # EIP带宽名称
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

表 7-17 独占带宽 EIP 跟随 Pod 创建的 annotation 配置

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/pod-with-eip	必选	false	是否需要跟随Pod创建EIP并绑定到该Pod。	"false"或"true"

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/eip-bandwidth-size	可选	5	带宽大小，单位为Mbit/s。	具体范围以各区域配置为准，根据带宽的计费类型不同可能存在差异，详情请参见弹性公网IP控制台的购买页面。 例如，“亚太-新加坡”区域按带宽计费类型的带宽大小范围为1Mbit/s~2000Mbit/s、按流量计费类型的带宽大小范围为1Mbit/s~300Mbit/s。
yangtse.io/eip-network-type	可选	5_bgp	公网IP类型。	具体类型以各区域配置为准，详情请参见弹性公网IP控制台的购买页面。 例如，“亚太-新加坡”区域支持以下类型： ● 5_bgp：全动态BGP
yangtse.io/eip-charge-mode	可选	空	按流量计费或按带宽计费。 **建议填写该参数。**若该参数为空，表示不指定计费模式，则以该区域下弹性公网IP接口的默认值为准。	● bandwidth：按带宽计费 ● traffic：按流量计费
yangtse.io/eip-bandwidth-name	可选	Pod名称	带宽名称。	● 1-64个字符，支持数字、字母、中文、_(下划线)、-（中划线）、.（点） ● 最小长度：1 ● 最大长度：64

- 创建Deployment时自动创建**共享带宽**类型的EIP，必须指定共享带宽ID，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
```



```
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
    annotations:
      yangtse.io/pod-with-eip: "true" # EIP跟随Pod创建
      yangtse.io/eip-network-type: 5_bgp # EIP类型
      yangtse.io/eip-bandwidth-id: <eip_bandwidth_id> # EIP共享带宽ID
  spec:
    containers:
      - name: container-0
        image: nginx:alpine
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
    imagePullSecrets:
      - name: default-secret
```

表 7-18 共享带宽类型 EIP 跟随 Pod 创建的 annotation 配置

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/pod-with-eip	必选	false	是否需要跟随Pod创建EIP并绑定到该Pod。	"false"或"true"
yangtse.io/eip-network-type	可选	5_bgp	公网IP类型。	• 5_bgp • 5_union • 5_sbgp 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
yangtse.io/eip-bandwidth-id	使用共享型带宽时必选	空	已有的带宽ID。 • 不填写该字段时，则默认使用独占带宽的EIP。独占带宽EIP的参数设置请参见**表 7-17**。 • 填写该字段时，只允许同时指定yangtse.io/eip-network-type字段，且该字段为可选。	-

Pod 使用已有 EIP

创建Pod时，填写yangtse.io/eip-id的annotation后，EIP会随Pod自动完成绑定。

说明

- 已有EIP绑定Pod后，系统会自动为该EIP添加集群ID、命名空间、Pod名称的标签。
- 当使用已有EIP的Pod被删除时，已有EIP会保留。系统会自动删除绑定EIP时添加的标签（集群ID、命名空间、Pod名称）。

以下示例创建一个名为nginx的实例数为1的无状态负载，EIP将随Pod自动绑定至Pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/eip-id: <eip_id> # 已有EIP的ID
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

表 7-19 使用已有 EIP 的 annotation 配置

annotation	是否可选	参数说明
yangtse.io/eip-id	必选	弹性公网IP的ID。 **获取方法：** 登录弹性公网IP控制台，在弹性公网IP列表单击需要绑定的EIP名称，找到“ID”字段复制即可。

检查 Pod 的 EIP 就绪

容器网络控制器会在Pod IP分配后，为Pod绑定EIP并回写分配结果至Pod的annotation（yangtse.io/allocated-ipv4-eip），Pod业务容器的启动时间可能早于EIP分配结果回写成功时间。

您可以尝试为Pod配置init container并使用downwardAPI类型的存储卷把yangtse.io/allocated-ipv4-eip的annotation通过volume挂载到init container里，并在init container中检查EIP是否已经分配成功。您可以参考以下示例配置init container。

```
apiVersion: v1
kind: Pod
```



```
metadata:
  name: example
  annotations:
    yangtse.io/pod-with-eip: "true"
    yangtse.io/eip-bandwidth-size: "5"
    yangtse.io/eip-network-type: 5_bgp
    yangtse.io/eip-charge-mode: bandwidth
    yangtse.io/eip-bandwidth-name: "xxx"
spec:
  initContainers:
  - name: init
    image: busybox:latest
    command: ['timeout', '60', 'sh', '-c', "until grep -E '[0-9]+' /etc/eipinfo/allocated-ipv4-eip; do echo waiting for allocated-ipv4-eip; sleep 2; done"]
    volumeMounts:
      - name: eipinfo
        mountPath: /etc/eipinfo
  volumes:
    - name: eipinfo
      downwardAPI:
        items:
          - path: "allocated-ipv4-eip"
            fieldRef:
              fieldPath: metadata.annotations['yangtse.io/allocated-ipv4-eip']
...
```

7.2.2.8 为 Pod 配置固定 EIP

使用场景

在云原生网络2.0下，支持为StatefulSet工作负载或直接创建的Pod分配固定的公网IP（EIP）。

约束限制

- 仅以下指定版本的CCE Turbo集群支持用户配置Pod固定EIP：
  - v1.19集群：v1.19.16-r20及以上版本
  - v1.21集群：v1.21.10-r0及以上版本
  - v1.23集群：v1.23.8-r0及以上版本
  - v1.25集群：v1.25.3-r0及以上版本
  - v1.25以上版本集群
- 开启固定EIP功能需要和Pod自动创建EIP功能配合使用，详情请参见**为Pod配置EIP**。
- 目前只支持StatefulSet类型的Pod或直接创建的Pod固定EIP，暂不支持Deployment、DaemonSet等其他类型的工作负载配置Pod固定EIP。
- 固定EIP创建后，生命周期内（如过期时间未到/Pod还在使用中）不支持通过Pod修改EIP属性。
- 对Pod的EIP地址无明确要求的业务不建议配置固定EIP，因为配置了固定EIP的Pod，Pod重建的耗时会略微变长。

配置固定 EIP

创建固定EIP的Pod时，填写EIP相关的annotation后，EIP会随Pod自动创建并绑定至该Pod。

以下示例创建一个名为nginx的有状态负载，EIP将随Pod自动创建并绑定至Pod。具体字段含义请参见**表7-20**。

- 创建有状态负载时固定**独占带宽**类型的EIP，无需指定带宽ID，示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  serviceName: nginx
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/static-eip: 'true'  # Pod固定EIP
        yangtse.io/static-eip-expire-no-cascading: 'false'  # EIP级联删除
        yangtse.io/static-eip-expire-duration: 5m  # 固定EIP过期回收的时间间隔
        yangtse.io/pod-with-eip: 'true'  # EIP跟随Pod创建
        yangtse.io/eip-bandwidth-size: '5'  # EIP带宽
        yangtse.io/eip-network-type: 5_bgp  # EIP类型
        yangtse.io/eip-charge-mode: bandwidth  # EIP计费模式
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

- 创建有状态负载时固定**共享带宽**类型的EIP，必须指定带宽ID，示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  serviceName: nginx
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        yangtse.io/static-eip: 'true'  # Pod固定EIP
        yangtse.io/pod-with-eip: 'true'  # EIP跟随Pod创建
        yangtse.io/eip-network-type: 5_bgp  # EIP类型
        yangtse.io/eip-bandwidth-id: <eip_bandwidth_id>  # EIP共享带宽ID
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
```



```
imagePullSecrets:
  - name: default-secret
```

表 7-20 Pod 固定 EIP 的 annotation 配置

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/static-eip	必选	false	是否开启Pod固定EIP，只有StatefulSet类型的Pod或无ownerReferences的Pod支持，默认不开启。	"false"或"true"
yangtse.io/static-eip-expire-duration	可选	5m	删除固定EIP的Pod后，对应的固定EIP过期回收的时间间隔。	支持时间格式为Go time type，例如1h30m、5m。关于Go time type，请参见**Go time type**。
yangtse.io/static-eip-expire-no-cascading	可选	false	是否关闭StatefulSet工作负载的级联回收。 默认为false，表示StatefulSet删除后，会级联删除对应的固定EIP。如果您需要在删除StatefulSet对象后，在EIP过期回收时间内保留对应的固定EIP，用于下一次重建同名的StatefulSet再次使用对应的固定EIP，请将该参数设为true。	"false"或"true"

表 7-21 独占带宽 EIP 跟随 Pod 创建的 annotation 配置

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/pod-with-eip	必选	false	是否需要跟随Pod创建EIP并绑定到该Pod。	"false"或"true"

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/eip-bandwidth-size	可选	5	带宽大小，单位为Mbit/s。	具体范围以各区域配置为准，根据带宽的计费类型不同可能存在差异，详情请参见弹性公网IP控制台的购买页面。 例如，“亚太-新加坡”区域按带宽计费类型的带宽大小范围为1Mbit/s~2000Mbit/s、按流量计费类型的带宽大小范围为1Mbit/s~300Mbit/s。
yangtse.io/eip-network-type	可选	5_bgp	公网IP类型。	具体类型以各区域配置为准，详情请参见弹性公网IP控制台的购买页面。 例如，“亚太-新加坡”区域支持以下类型： ● 5_bgp：全动态BGP
yangtse.io/eip-charge-mode	可选	空	按流量计费或按带宽计费。 **建议填写该参数。**若该参数为空，表示不指定计费模式，则以该区域下弹性公网IP接口的默认值为准。	● bandwidth：按带宽计费 ● traffic：按流量计费
yangtse.io/eip-bandwidth-name	可选	Pod名称	带宽名称。	● 1-64个字符，支持数字、字母、中文、_(下划线)、-（中划线）、.（点） ● 最小长度：1 ● 最大长度：64

表 7-22 共享带宽类型 EIP 跟随 Pod 创建的 annotation 配置

annotation	是否可选	默认值	参数说明	取值范围
yangtse.io/pod-with-eip	必选	false	是否需要跟随Pod创建EIP并绑定到该Pod。	"false"或"true"
yangtse.io/eip-network-type	可选	5_bgp	公网IP类型。	• 5_bgp • 5_union • 5_sbgp 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
yangtse.io/eip-bandwidth-id	使用共享型带宽时必选	空	已有的带宽ID。 • 不填写该字段时，则默认使用独占带宽的EIP。独占带宽EIP的参数设置请参见表7-17。 • 填写该字段时，只允许同时指定yangtse.io/eip-network-type字段，且该字段为可选。	-

删除固定 EIP

删除Pod后，在配置的固定EIP过期时间内，如果有同名的Pod创建，EIP依旧可用。只有在EIP过期时间内没有同名Pod创建或者删除StatefulSet时开启级联删除EIP时，固定EIP才会删除。

7.2.2.9 为 IPv6 双栈网卡的 Pod 配置共享带宽

使用场景

默认情况下具有IPv6双栈网卡的Pod只具备IPv6私网访问能力，如果需要访问公网，则需要为该IPv6双栈网卡的Pod配置共享带宽。

约束限制

- 仅支持CCE Turbo集群，且需要满足以下条件：
  - 集群已开启IPv6双栈。
  - 集群版本为v1.23.8-r0、v1.25.3-r0及以上。
- 共享带宽可加入的IPv6网卡数受限于租户配额，目前默认为20。
- 不支持HostNetwork的Pod。
- 支持所有类型的工作负载，特别地，为Deployment，Statefulset等有副本数属性的工作负载配置IPv6共享带宽时，需确保副本数以及升级过程中最大的Pod数小于共享带宽当前剩余可加入的IPv6网卡数。

- 配置了共享带宽的IPv6双栈Pod：Pod创建时，CNI会等待IPv6双栈网卡插入共享带宽完成后才会返回成功；Pod删除时，会等待Pod完全删除或最长30秒删除状态后进行IPv6双栈网卡移出共享带宽。
- 如果Pod对应的IPv6双栈网卡加入共享带宽失败，Pod上会有Event告警事件FailedIPv6InsertBandwidth（如超过配额，触发流控等），请根据告警事件进行相应的处理。
- 弹性公网IP控制台中的“共享带宽”页面，单击共享带宽详情下的“IPv6网卡”页签，可以看到所属实例为“云容器引擎”的IPv6双栈网卡，请勿在页面上直接移除或调用VPC的API移除，以免影响您的业务。

通过控制台设置

您可以在创建工作负载时，选择“高级配置 > 网络配置”设置IPv6共享带宽。

通过 kubectl 命令行设置

您可以通过对Deployment添加annotations指定Pod的IPv6双栈网卡将要加入的共享带宽，如下所示。

```
...
spec:
  selector:
    matchLabels:
      app: demo
      version: v1
  template:
    metadata:
      annotations:
        yangtse.io/ipv6-bandwidth-id: "xxx"
```

- yangtse.io/ipv6-bandwidth-id：共享带宽的ID，Pod对应的IPv6双栈网卡将加入此共享带宽。该ID可前往弹性公网IP控制台中的“共享带宽”页面查询。

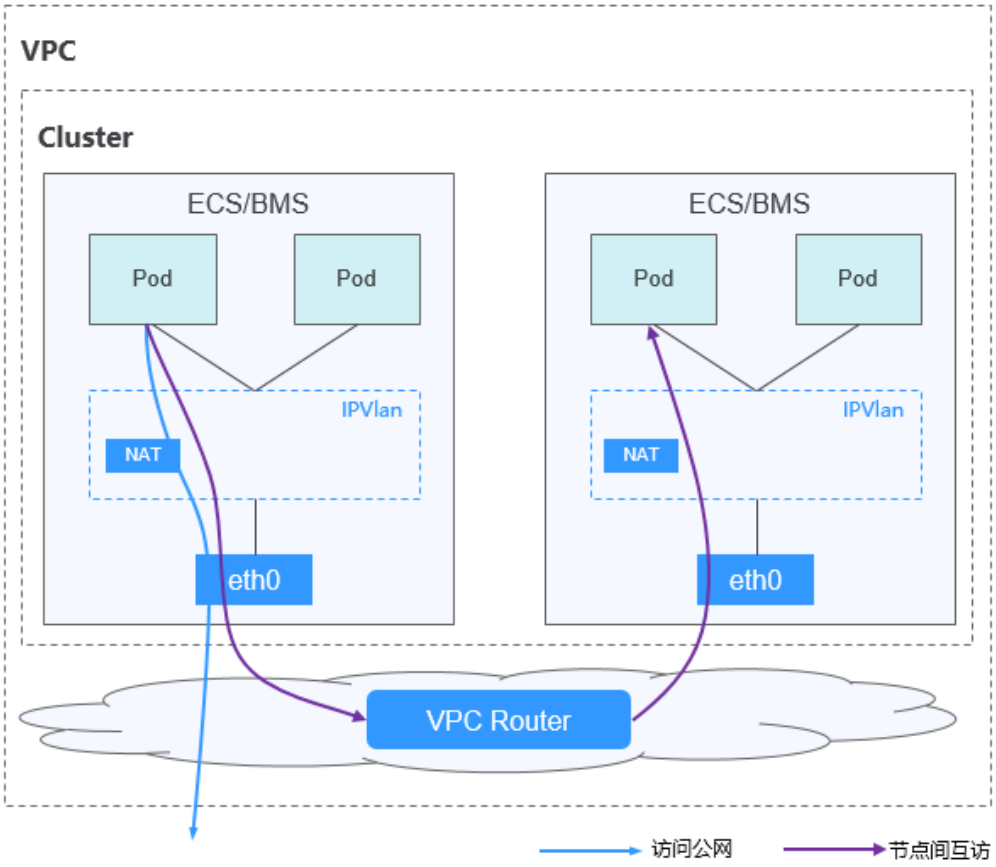
7.2.3 VPC 网络模型

7.2.3.1 VPC 网络模型说明

VPC 网络模型

VPC网络模型将虚拟私有云VPC的路由方式与底层网络深度整合，适用于高性能场景，但节点数量受限于虚拟私有云VPC的路由配额。在VPC网络模型中，容器网段独立于节点网段进行单独设置。在容器IP地址分配时，集群中的每个节点会被分配固定大小的容器IP地址段，用于给该节点上运行的容器分配容器IP。由于VPC网络模型没有隧道封装的消耗，容器网络性能相对于容器隧道网络有一定优势。此外，使用VPC网络模型的集群时，由于VPC路由表中自动配置了容器网段与VPC网段之间的路由，可以支持同一VPC内的云服务器从集群外直接访问容器实例等特殊场景。

图 7-21 VPC 网络



在VPC网络模型的集群中，不同形式的网络通信路径不同：

- 节点内Pod间通信：IPVlan子接口分配给节点上的Pod，因此同节点的Pod间通信可以直接通过IPVlan进行转发。
- 跨节点Pod间通信：所有跨节点Pod间的通信均根据VPC路由表中的路由先访问到默认网关，然后借助VPC的路由转发能力，将访问流量转发到另一个节点上的Pod。
- Pod访问公网：集群内的容器在访问公网时，系统会将容器IP通过NAT转换成节点IP，使Pod以节点IP的形式与外部进行通信。

说明

VPC网络集群中默认将10.0.0.0/8、172.16.0.0/12、192.168.0.0/16这三个VPC私有网段视为集群私有网段。如果集群所在VPC使用了扩展网段，创建、重置节点等操作也会将扩展网段添加到集群私有网段中。

在Pod中发起访问请求时，如果Pod访问的目的地址在集群私有网段范围内，则节点不会将Pod IP进行网络地址转换，可以借由上层VPC直接将报文送达至目的地址，即直接使用Pod IP与集群私有网络地址进行通信。

当VPC网络集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时，CCE提供nonMasqueradeCIDRs参数设置集群私有网段，以满足不同的使用场景，详情请参见**在VPC网络集群中访问集群外地址时使用Pod IP作为客户端源IP**。

优缺点

优点

- 由于没有隧道封装，网络问题易排查、性能较高。
- 在同一个VPC内，由于VPC路由表中自动配置了容器网段与VPC网段之间的路由，同VPC内的资源可以与集群内部的容器直接进行网络通信。

  说明

  同理，如果该VPC和其他VPC或数据中心网络环境连通，且在VPC路由表中添加容器网段的路由，在网段不冲突的情况下，其他VPC或数据中心所属的资源也可以与集群内部的容器直接进行网络通信。

缺点

- 节点数量受限于虚拟私有云VPC的路由配额。
- 每个节点将会被分配固定大小的IP地址段，存在一定的容器网段IP地址浪费。
- Pod无法直接利用EIP、安全组等能力。

应用场景

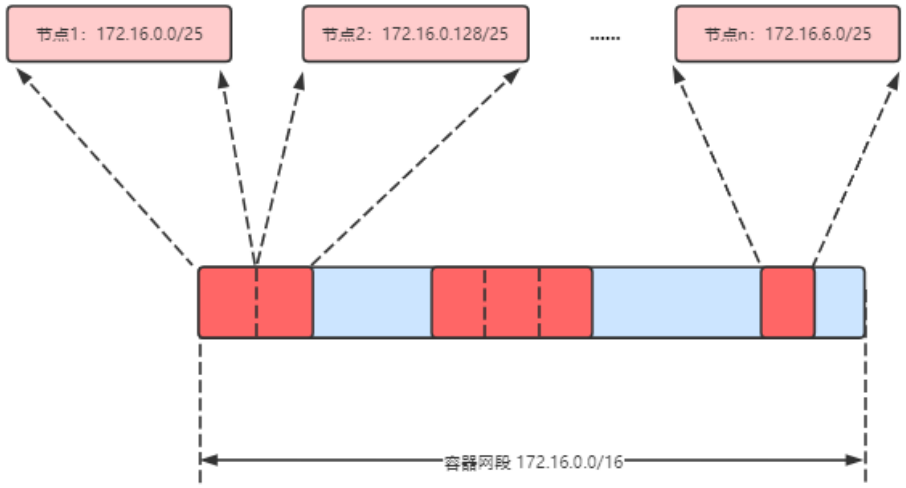
- 性能要求较高：由于没有额外的隧道封装，相比于容器隧道网络模式，VPC网络模型集群的容器网络性能接近于VPC网络性能，所以适用于对性能要求较高的业务场景，比如：AI计算、大数据计算等。
- 中小规模组网：由于VPC路由网络受限于VPC路由表条目配额的限制，建议集群规模为1000节点及以下。

容器 IP 地址管理

VPC网络模型根据如下规则分配容器IP：

- 容器网段独立于节点网段进行单独设置。
- 按节点维度划分地址段，集群的所有节点从容器网段中分配一个固定大小（用户自己配置）的IP网段。
- 容器网段依次循环分配IP网段给新增节点。
- 调度到节点上的Pod依次循环从分配给节点的IP网段内分配IP地址。

图 7-22 VPC 网络 IP 地址管理



按如上IP分配，VPC网络的集群最多能创建节点数量 = 容器网段IP数量 ÷ 节点从容器网段中分配IP网段的IP数量

比如容器网段为172.16.0.0/16，则IP数量为65536，节点分配容器网段掩码为25，也就是每个节点容器IP数量为128，则最多可创建节点数量为65536/128=512。另外，集群能创建多少节点，还受节点子网的可用IP数和集群规模的影响，详情请参见**网段规划建议**。

网段规划建议

在**集群网络构成**中介绍集群中网络地址可分为集群网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- **三个网段不能重叠**，否则会导致冲突。且集群所在VPC下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
- 保证**每个网段有足够的IP地址可用**。
  - 集群网段的IP地址要与集群规模相匹配，否则会因为IP地址不足导致无法创建节点。
  - 容器网段的IP地址要与业务规模相匹配，否则会因为IP地址不足导致无法创建Pod。每个节点上可以创建多少Pod还与其他参数设置相关，具体请参见**节点可创建的最大Pod数量说明**。

例如集群规模为200节点，容器网络模型为VPC网络。

则此时选择子网的可用IP数量需要超过200，否则会因为IP地址不足导致无法创建节点。

容器网段为172.16.0.0/16，可用IP数量为65536，如**容器IP地址管理**中所述，VPC网络IP分配是分配固定大小的网段（使用掩码实现，确定每个节点最多分配多少容器IP），例如上限为128，则此时集群最多支撑65536/128=512个节点。

图 7-23 容器网段配置（创建集群时配置）

VPC 网络访问示例

本示例中，创建一个VPC网络的集群，且集群中包含一个Node节点。

在VPC控制台中，找到集群所在VPC，查看该VPC的路由表。

在路由表中存在一条由CCE自动添加的自定义路由，该路由的目的地址为分配给该节点的容器网段，下一跳指向对应的节点。示例中集群容器网段为172.16.0.0/16，每个节点容器IP数量为128，则分配给该节点的容器网段为172.16.0.0/25，该网段包含128个容器IP。

图 7-24 路由

当访问容器IP时，VPC路由就会将指向目的地址的流量转发到下一跳的节点，访问示例如下。

步骤1 使用kubectl命令行工具连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在集群中创建一个Deployment。

创建deployment.yaml文件，文件内容示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: container-0
          image: 'nginx:perl'
      imagePullSecrets:
        - name: default-secret
```

创建该工作负载：

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

步骤3 查看已运行的Pod。

```
kubectl get pod -owide
```

回显如下：

```
NAME                       READY   STATUS    RESTARTS   AGE   IP          NODE          NOMINATED NODE
READINESS GATES
example-86b9779494-l8qrw   1/1     Running   0          14s   172.16.0.6   192.168.0.99   <none>
<none>
example-86b9779494-svs8t   1/1     Running   0          14s   172.16.0.7   192.168.0.99   <none>
<none>
example-86b9779494-x8kl5   1/1     Running   0          14s   172.16.0.5   192.168.0.99   <none>
<none>
example-86b9779494-zt627   1/1     Running   0          14s   172.16.0.8   192.168.0.99   <none>
<none>
```

步骤4 您可以使用同一VPC内的云服务器从集群外直接访问Pod的IP。而在集群内部节点或Pod内，也可以使用Pod IP正常访问Pod。例如以下示例中，进入到容器中直接访问Pod IP，其中*example-86b9779494-l8qrw*为Pod名称，*172.16.0.7*为Pod IP。

```
kubectl exec -it example-86b9779494-l8qrw -- curl 172.16.0.7
```

回显如下，说明可正常访问工作负载应用：

```
<!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>
```

----结束

7.2.3.2 扩展集群容器网段

操作场景

当创建CCE集群时设置的容器网段太小，无法满足业务扩容需求时，您通过扩展集群容器网段的方法来解决。本文介绍如何为集群添加容器网段。

约束与限制

- 仅支持v1.19及以上版本的“VPC网络”模型集群。
- 容器网段添加后无法删除，请谨慎操作。

为 CCE Standard 集群添加容器网段

步骤1 登录CCE控制台，单击CCE集群名称，进入集群。

步骤2 在“总览”页面，找到“网络信息”版块，并单击“添加”。

图 7-25 添加容器网段

步骤3 设置需要添加的容器网段，您可单击 + 一次性添加多个容器网段。

说明

新增的容器网段不能与服务网段、VPC网段及已有的容器网段冲突。

图 7-26 设置网段

步骤4 单击“确定”。

----结束

7.2.4 容器隧道网络模型

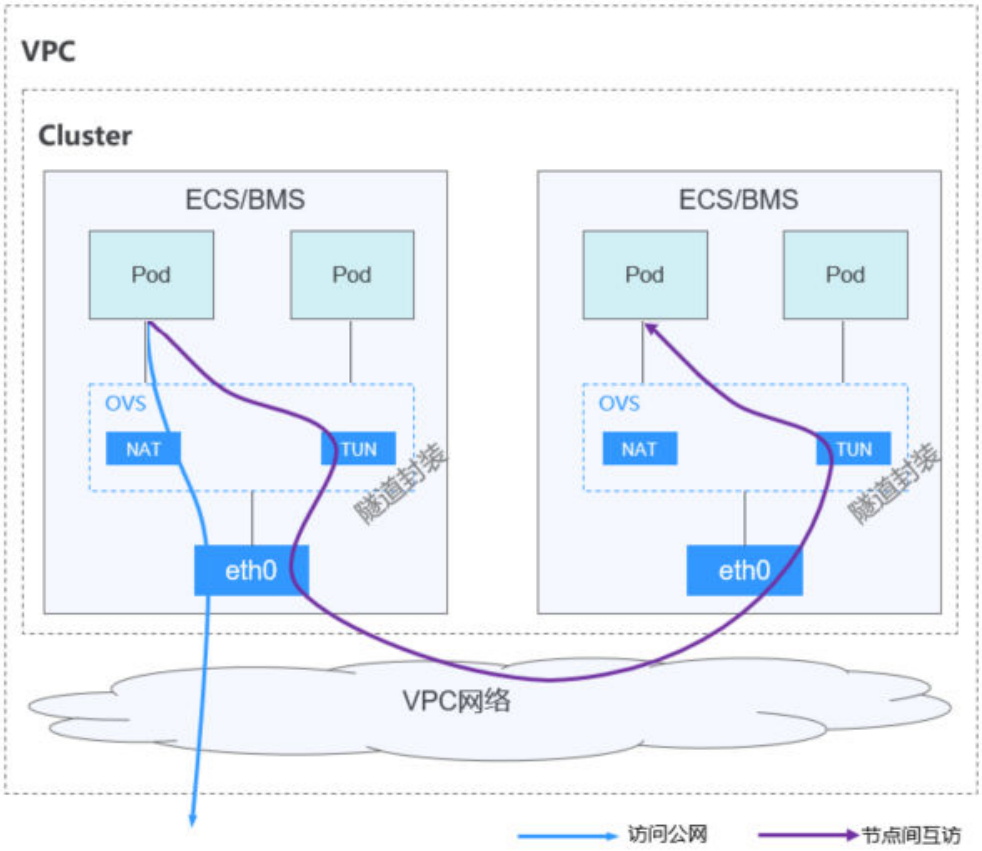
7.2.4.1 容器隧道网络模型说明

容器隧道网络模型

容器隧道网络是在主机网络平面的基础上，通过隧道封装技术来构建一个独立的容器网络平面。CCE集群容器隧道网络使用了VXLAN作为隧道封装协议，并使用了Open vSwitch作为后端虚拟交换机。VXLAN是一种将以太网报文封装成UDP报文进行隧道传输的协议，而Open vSwitch是一款开源的虚拟交换机软件，提供网络隔离和数据转发等功能。

容器隧道网络虽然会有少量隧道封装性能损耗，但具有通用性强、互通性强、高级特性支持全面（例如NetworkPolicy网络隔离）等优势，适用于大多数性能要求不高的场景。

图 7-27 容器隧道网络



在容器隧道模型的集群中，节点内Pod间通信和跨节点Pod间通信路径不同：

- 节点内Pod间通信：同节点的Pod间通信通过本节点的OVS网桥直接转发。
- 跨节点Pod间通信：所有跨节点Pod间的通信通过OVS隧道网桥进行封装后，通过主机网卡转发到另一个节点上的Pod。

优缺点

优点

- 容器网络和节点网络解耦，不受VPC配额规格、响应速度的限制（如VPC路由条目数、弹性网卡数、创建速度限制）。
- 支持网络隔离，具体请参见**配置网络策略（NetworkPolicy）限制Pod访问的对象**。
- 支持带宽限制。
- 支持大规模组网，最大可支持2000节点规模。

缺点

- 由于隧道封装，网络问题排查难度较大，整体性能较低。
- Pod无法直接利用EIP、安全组等能力。
- 不支持外部网络与容器IP直接进行网络通信。

应用场景

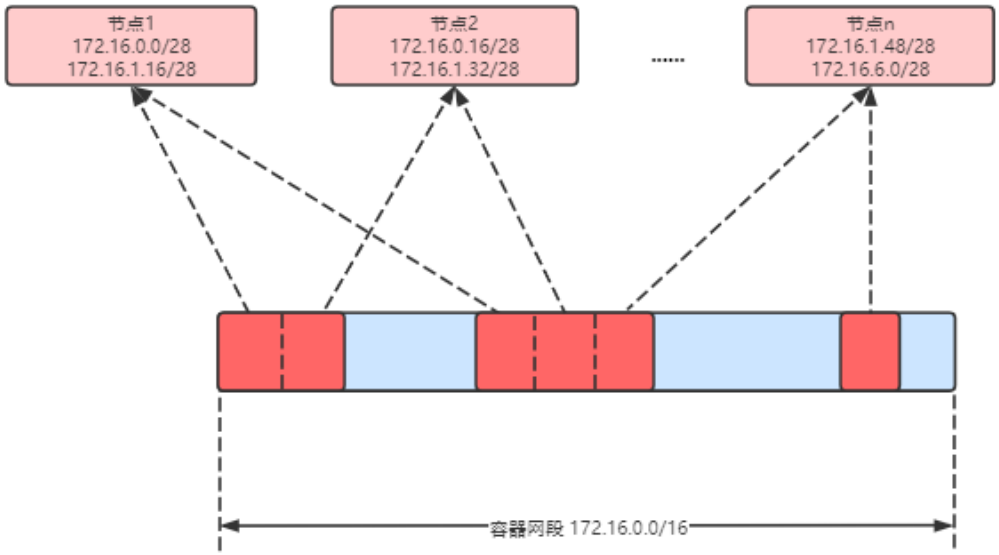
- 对性能要求不高：由于需要额外的VXLAN隧道封装，相对于另外两种容器网络模式，性能存在一定的损耗（约5%-15%）。所以容器隧道网络适用于对性能要求不是特别高的业务场景，比如：Web应用、访问量不大的数据中台、后台服务等。
- 大规模组网：相比VPC路由网络受限于VPC路由条目配额的限制，容器隧道网络没有网络基础设施的任何限制；同时容器隧道网络把广播域控制到了节点级别，容器隧道网络最大可支持2000节点规模。

容器 IP 地址管理

容器隧道网络按如下规则分配容器IP：

- 容器网段独立于节点网段进行单独设置。
- 按节点维度划分地址段，集群的所有节点从容器网段中分配一个或多个固定大小（默认16）的IP网段。
- 当节点上的IP地址使用完后，可再次申请分配一个新的IP网段。
- 容器网段依次循环分配IP网段给新增节点或存量节点。
- 调度到节点上的Pod依次循环从分配给节点的一个或多个IP网段内分配IP地址。

图 7-28 容器隧道网络 IP 地址分配



按如上IP分配，容器隧道网络的集群最多能创建节点数量 = 容器网段IP数量 ÷ 节点从容器网段中一次分配的IP网段大小（默认为16）

比如容器网段为172.16.0.0/16，则IP数量为65536，节点分配容器网段掩码为28，也就是每次分配16个容器IP，则最多可创建节点数量为65536/16=4096。这是一种极端情况，如果创建4096个节点，则每个节点最多只能创建16个Pod，因为给每个节点只分配了16个IP的网段。另外集群能创建多少节点，还受节点子网的可用IP数和集群规模的影响。

图 7-29 网络模型选择（创建集群时配置）

网段规划建议

在**集群网络构成**中介绍集群中网络地址可分为集群网络、容器网络、服务网络三块，在规划网络地址时需要从如下方面考虑：

- **三个网段不能重叠**，否则会导致冲突。且集群所在VPC下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
- 保证**每个网段有足够的IP地址可用**。
  - 集群网段的IP地址要与集群规模相匹配，否则会因为IP地址不足导致无法创建节点。
  - 容器网段的IP地址要与业务规模相匹配，否则会因为IP地址不足导致无法创建Pod。每个节点上可以创建多少Pod还与其他参数设置相关，具体请参见**节点可创建的最大Pod数量说明**。

容器隧道网络访问示例

在容器隧道网络集群中创建工作负载的访问示例如下。

步骤1 使用kubectl命令行工具连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在集群中创建一个Deployment。

创建deployment.yaml文件，文件内容示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: container-0
          image: 'nginx:perl'
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      imagePullSecrets:
        - name: default-secret
```

创建该工作负载：

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

步骤3 查看已运行的Pod。

```
kubectl get pod -owide
```

回显如下：

```
NAME                       READY   STATUS    RESTARTS   AGE     IP          NODE          NOMINATED NODE
READINESS GATES
example-5bdc5699b7-5rvq4   1/1     Running   0          3m28s   10.0.0.20   192.168.0.42   <none>
<none>
example-5bdc5699b7-984j9   1/1     Running   0          3m28s   10.0.0.21   192.168.0.42   <none>
<none>
example-5bdc5699b7-lfxkm   1/1     Running   0          3m28s   10.0.0.22   192.168.0.42   <none>
<none>
```



```
example-5bdc5699b7-wjcmg   1/1     Running   0          3m28s   10.0.0.52   192.168.0.64   <none>
<none>
```

步骤4 如果使用同一VPC内的云服务器从集群外直接访问Pod的IP，会发现无法访问。

而在集群内部节点或Pod内，可以使用Pod IP正常访问Pod。例如以下示例中，进入到容器中直接访问Pod IP，其中*example-5bdc5699b7-5rvq4*为Pod名称，*10.0.0.21*为Pod IP。

```
kubectl exec -it example-5bdc5699b7-5rvq4 -- curl 10.0.0.21
```

回显如下，说明可正常访问工作负载应用：

```
<!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>
```

----结束

7.2.5 Pod 网络配置

7.2.5.1 在 Pod 中配置主机网络（hostNetwork）

背景信息

Kubernetes支持Pod直接使用主机（节点）的网络，当Pod配置为hostNetwork: true时，在此Pod中运行的应用程序可以直接看到Pod所在主机的网络接口。

配置说明

Pod使用主机网络只需要在配置中添加hostNetwork: true即可，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
```



```
  metadata:
    labels:
      app: nginx
  spec:
    hostNetwork: true
    containers:
    - image: nginx:alpine
      name: nginx
    imagePullSecrets:
    - name: default-secret
```

部署后可以看到Pod的IP与节点的IP相同，说明Pod直接使用了主机网络。

```
$ kubectl get pod -owide
NAME                     READY   STATUS    RESTARTS   AGE     IP          NODE        NOMINATED NODE
READINESS GATES
nginx-6fdf99c8b-6wwft    1/1     Running   0          3m41s   10.1.0.55   10.1.0.55   <none>          <none>
```

hostNetwork 使用注意事项

Pod直接使用主机的网络会占用宿主机的端口，Pod的IP就是宿主机的IP，使用时需要考虑是否与主机上的端口冲突，因此一般情况下除非某个特定应用必须占用宿主机上的特定端口，否则不建议使用主机网络。

由于Pod使用主机网络，访问Pod需要直接通过节点端口，因此要**注意放通节点安全组端口**，否则会出现访问不通的情况。

另外由于占用主机端口，使用Deployment部署hostNetwork类型Pod时，要注意**Pod的副本数不要超过节点数量**，否则会导致一个节点上调度了多个Pod，Pod启动时端口冲突无法创建。例如上面例子中的nginx，如果服务数为2，并部署在只有1个节点的集群上，就会有一个Pod无法创建，查询Pod日志会发现是由于端口占用导致nginx无法启动。

⚠ 注意

请避免在同一个节点上调度多个使用主机网络的Pod，否则在创建ClusterIP类型的Service访问Pod时，会出现访问ClusterIP不通的情况。

```
$ kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/2     2            1           67m
$ kubectl get pod
NAME                     READY   STATUS             RESTARTS   AGE
nginx-6fdf99c8b-6wwft    1/1     Running            0          67m
nginx-6fdf99c8b-rglm7    0/1     CrashLoopBackOff   13         44m
$ kubectl logs nginx-6fdf99c8b-rglm7
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
```

```
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: bind() to [::]:80 failed (98: Address in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address in use)
2022/05/11 07:18:11 [emerg] 1#1: still could not bind()
nginx: [emerg] still could not bind()
```

7.2.5.2 为 Pod 配置 QoS

操作场景

部署在同一节点上的不同业务容器之间存在带宽抢占，容易造成业务抖动。您可以通过对Pod间互访进行QoS限速来解决这个问题。

约束与限制

Pod互访限速设置需遵循以下约束：

约束类别	容器隧道网络模式	VPC网络模式	云原生2.0网络模式
支持的版本	所有版本都支持	v1.19.10以上集群版本	v1.19.10以上集群版本
支持的运行时类型	仅支持普通容器（容器运行时为runC），不支持安全容器（容器运行时为Kata）		
支持的Pod类型	仅支持非HostNetwork类型Pod		
支持的场景	支持Pod间互访、Pod访问Node、Pod访问Service的场景限速		
限制的场景	无	无	● 不支持Pod访问100.64.0.0/10和214.0.0.0/8外部云服务网段的限速场景 ● 不支持健康检查的流量限速场景

约束类别	容器隧道网络模式	VPC网络模式	云原生2.0网络模式
限速值取值范围	只支持单位M或G的限速配置，如100M，1G；最小取值1M，最大取值4.29G。		

通过控制台设置

通过控制台创建工作负载时，您可在创建工作负载页面的“高级配置 > 网络配置”中设置Pod入/出口带宽限速。

图 7-30 网络配置

通过 kubectl 命令行设置

您可以通过对工作负载添加annotations指定出口带宽和入口带宽，如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
  namespace: default
  labels:
    app: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
      annotations:
        kubernetes.io/ingress-bandwidth: 100M
        kubernetes.io/egress-bandwidth: 100M
    spec:
      containers:
        - name: container-1
          image: nginx:alpine
          imagePullPolicy: IfNotPresent
```



```
imagePullSecrets:
  - name: default-secret
```

- kubernetes.io/ingress-bandwidth：Pod的入口带宽
- kubernetes.io/egress-bandwidth：Pod的出口带宽

如果不设置这两个参数，则表示不限制带宽。

说明

修改Pod出/入口带宽限速后，需要重启容器才可生效。由于独立创建的Pod（不通过工作负载管理）修改annotations后不会触发容器重启，因此带宽限制不会生效，您可以重新创建Pod或手动触发容器重启。

7.2.5.3 配置网络策略（NetworkPolicy）限制 Pod 访问的对象

网络策略（NetworkPolicy）是Kubernetes设计用来限制Pod访问的对象，相当于从应用的层面构建了一道防火墙，进一步保证了网络安全。NetworkPolicy支持的能力取决于集群的网络插件的能力。

默认情况下，如果命名空间中不存在任何策略，则所有进出该命名空间中的Pod的流量都被允许。

NetworkPolicy的规则可以选择如下3种：

- namespaceSelector：根据命名空间的标签选择，具有该标签的命名空间都可以访问。
- podSelector：根据Pod的标签选择，具有该标签的Pod都可以访问。
- ipBlock：根据网络选择，网段内的IP地址都可以访问。（仅Egress支持IPBlock）

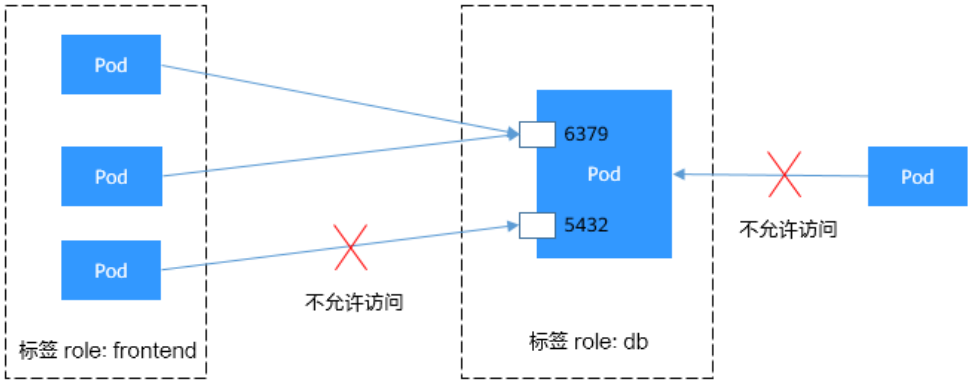
约束与限制

- 当前仅**容器隧道网络模型**的集群支持网络策略（NetworkPolicy）。网络策略可分为以下规则：
  - 入规则（Ingress）：所有版本均支持。
  - 出规则（Egress）：v1.23及以上集群版本。
- 不支持对IPv6地址网络隔离。
- 通过原地升级到支持Egress的集群版本，由于不会升级节点操作系统，会导致无法使用Egress，此种情况下，请重置节点。

通过 YAML 使用 Ingress 规则

- **场景一：通过网络策略限制Pod只能被带有特定标签的Pod访问**

图 7-31 podSelector



目标Pod具有role=db标签，该Pod只允许带有role=frontend标签的Pod访问其6379端口。设置该网络策略的具体操作步骤如下：

a. 创建名为access-demo1.yaml文件。

```
vim access-demo1.yaml
```

以下为YAML文件内容：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-demo1
  namespace: default
spec:
  podSelector:                # 规则对具有role=db标签的Pod生效
    matchLabels:
      role: db
  ingress:                    # 表示入规则
  - from:
    - podSelector:            # 只允许具有role=frontend标签的Pod访问
        matchLabels:
          role: frontend
    ports:                    # 只能使用TCP协议访问6379端口
    - protocol: TCP
      port: 6379
```

b. 执行以下命令，根据上述的access-demo1.yaml文件创建网络策略。

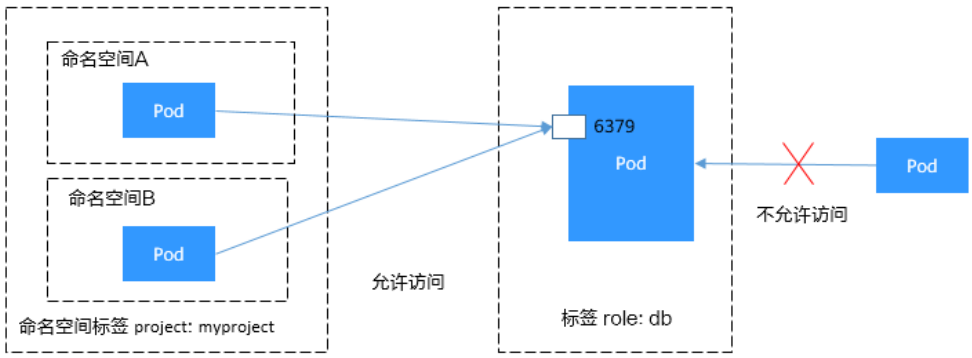
```
kubectl apply -f access-demo1.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/access-demo1 created
```

- **场景二：通过网络策略限制Pod只能被指定命名空间下的Pod访问**

图 7-32 namespaceSelector



目标Pod具有role=db标签，该Pod只允许project=myproject标签的命名空间中的Pod访问其6379端口。设置该网络策略的具体操作步骤如下：

a. 创建名为access-demo2.yaml文件。

```
vim access-demo2.yaml
```

以下为YAML文件内容：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-demo2
spec:
  podSelector:                  # 规则对具有role=db标签的Pod生效
    matchLabels:
      role: db
  ingress:                      # 表示入规则
  - from:
    - namespaceSelector:        # 只允许具有project=myproject标签的命名空间中的Pod访问
        matchLabels:
          project: myproject
    ports:                      # 只能使用TCP协议访问6379端口
    - protocol: TCP
      port: 6379
```

b. 执行以下命令，根据上述的access-demo2.yaml文件创建网络策略。

```
kubectl apply -f access-demo2.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/access-demo2 created
```

通过 YAML 使用 Egress 规则

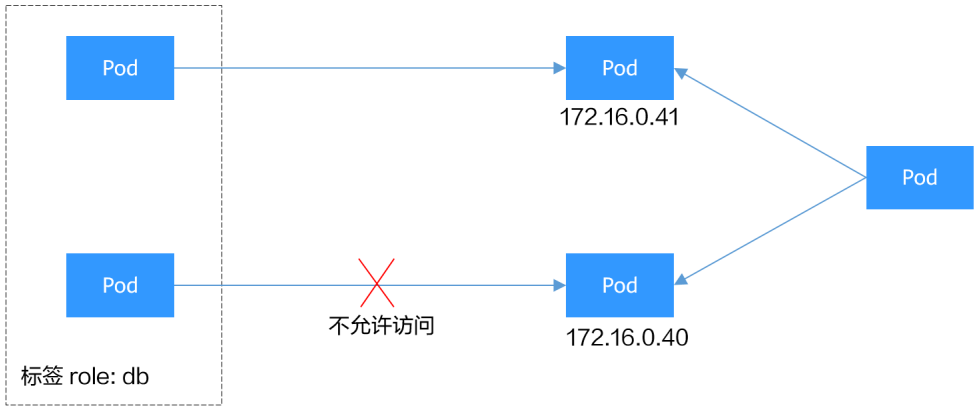
Egress不仅支持podSelector和namespaceSelector，还支持ipBlock。

说明

仅1.23及以上版本集群支持Egress规则。

- **场景一：通过网络策略限制Pod只能访问指定地址**

图 7-33 ipBlock



目标Pod具有role=db标签，该Pod只允许访问172.16.0.16/16网段，但不允许访问该网段中的172.16.0.40/32地址。设置该网络策略的具体操作步骤如下：

a. 创建名为access-demo3.yaml文件。

```
vim access-demo2.yaml
```

以下为YAML文件内容：


```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-demo3
  namespace: default
spec:
  policyTypes:                  # 使用Egress必须指定policyType
    - Egress
  podSelector:                  # 规则对具有role=db标签的Pod生效
    matchLabels:
      role: db
  egress:                       # 表示出规则
  - to:
    - ipBlock:
        cidr: 172.16.0.16/16    # 允许在出方向访问此网段
        except:
        - 172.16.0.40/32        # 不允许在出方向访问此网段，except网段需在cidr网段内
```

b. 执行以下命令，根据上述的access-demo3.yaml文件创建网络策略。

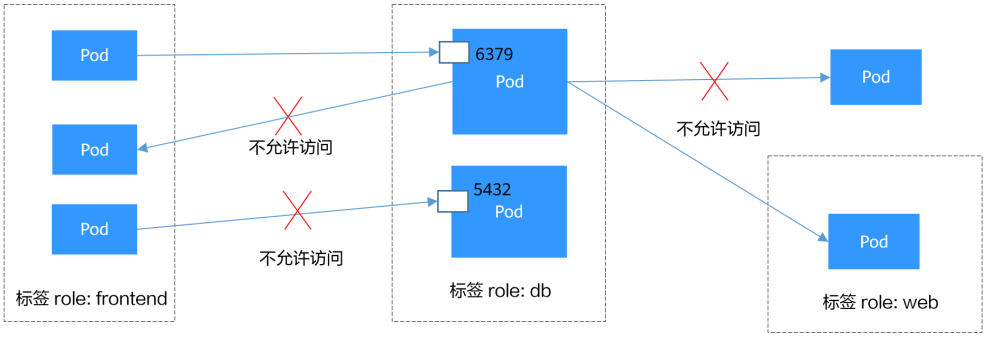
```
kubectl apply -f access-demo3.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/access-demo3 created
```

- **场景二：通过网络策略限制Pod只能被带有特定标签的Pod访问，且只能访问指定Pod**

图 7-34 同时使用 Ingress 和 Egress



目标Pod具有role=db标签，该Pod只允许带有role=frontend标签的Pod访问其6379端口，且该Pod只能访问带有role=web标签的Pod。网络策略中的Ingress和Egress可以定义在同一个规则中，具体操作步骤如下：

a. 创建名为access-demo4.yaml文件。

```
vim access-demo2.yaml
```

以下为YAML文件内容：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-demo4
  namespace: default
spec:
  policyTypes:
  - Ingress
  - Egress
  podSelector:                  # 规则对具有role=db标签的Pod生效
    matchLabels:
      role: db
  ingress:                      # 表示入规则
  - from:
    - podSelector:              # 只允许具有role=frontend标签的Pod访问
        matchLabels:
```



```
      role: frontend
  ports:                  # 只能使用TCP协议访问6379端口
  - protocol: TCP
    port: 6379
egress:                   # 表示出规则
- to:
  - podSelector:           # 只允许访问具有role=web标签的Pod
      matchLabels:
        role: web
```

b. 执行以下命令，根据上述的access-demo4.yaml文件创建网络策略。

```
kubectl apply -f access-demo4.yaml
```

预期输出：

```
networkpolicy.networking.k8s.io/access-demo4 created
```

通过控制台创建网络策略

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“策略”，在右侧选择“网络策略”页签，单击右上角“创建网络策略”。

- 策略名称：自定义输入NetworkPolicy名称。
- 命名空间：选择网络策略所在命名空间。
- 选择器：输入标签选择要关联的Pod，然后单击添加。您也可以单击“引用负载标签”直接引用已有负载的标签。
- 入方向规则：单击+添加入方向规则，参数设置请参见**表7-23**。

表 7-23 添加入方向规则

参数	参数说明
协议端口	请选择对应的协议类型和端口，目前支持TCP和UDP协议。
源对象命名空间	选择允许哪个命名空间的对象访问。不填写表示和当前策略属于同一命名空间。
源对象Pod标签	允许带有这个标签的Pod访问，不填写表示命名空间下全部Pod。

- 出方向规则：单击+添加出方向规则，参数设置请参见**表7-23**。

表 7-24 添加出方向规则

参数	参数说明
协议端口	请选择对应的协议类型和端口，目前支持TCP和UDP协议。不填写表示不限制。
目标网段	允许将流量转发至指定的一个网段内（可指定多个例外网段）。指定网段和例外网段用竖线（|）分隔，多个例外网段用逗号（,）分隔。例如 172.17.0.0/16|172.17.1.0/24,172.17.2.0/24 表示允许访问 172.17.0.0/16 网段，其中 172.17.1.0/24 和 172.17.2.0/24 两个网段例外。
目标对象命名空间	选择允许访问哪个命名空间中的对象。不填写表示和当前策略属于同一命名空间。
目标对象Pod标签	允许访问携带此标签的Pod，不填写表示命名空间下全部Pod。

步骤3 设置完成后，单击“确定”。

----结束

7.3 服务（Service）

7.3.1 服务概述

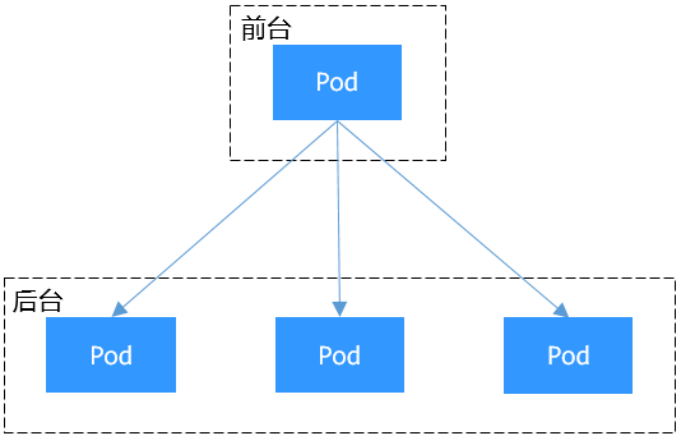
直接访问 Pod 的问题

Pod创建完成后，如何访问Pod呢？直接访问Pod会有如下几个问题：

- Pod会随时被Deployment这样的控制器删除重建，那访问Pod的结果就会变得不可预知。
- Pod的IP地址是在Pod启动后才被分配，在启动前并不知道Pod的IP地址。
- 应用往往都是由多个运行相同镜像的一组Pod组成，逐个访问Pod也变得不现实。

举个例子，假设有这样一个应用程序，使用Deployment创建了前台和后台，前台会调用后台做一些计算处理，如图7-35所示。后台运行了3个Pod，这些Pod是相互独立且可被替换的，当Pod出现状况被重建时，新建的Pod的IP地址是新IP，前台的Pod无法直接感知。

图 7-35 Pod 间访问

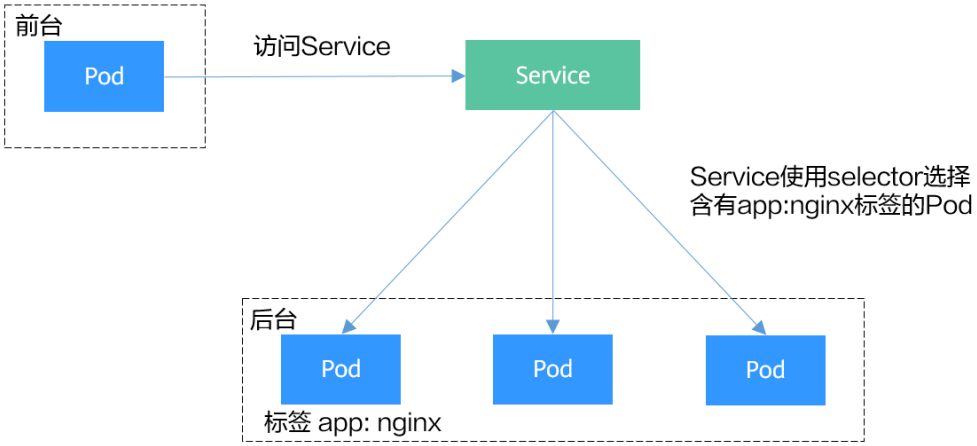


使用 Service 解决 Pod 的访问问题

Kubernetes中的Service对象就是用来解决上述Pod访问问题的。Service有一个固定IP地址（在创建CCE集群时有一个服务网段的设置，这个网段专门用于给Service分配IP地址），Service将访问它的流量转发给Pod，具体转发给哪些Pod通过Label来选择，而且Service可以给这些Pod做负载均衡。

那么对于上面的例子，为后台添加一个Service，通过Service来访问Pod，这样前台Pod就无需感知后台Pod的变化，如图7-36所示。

图 7-36 通过 Service 访问 Pod



Service 的类型

Kubernetes允许指定一个需要的类型的Service，类型的取值以及行为如下：

- **集群内访问(ClusterIP)**

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

- **节点访问(NodePort)**

  节点访问（NodePort）是指在每个节点的IP上开放一个静态端口，通过静态端口对外暴露服务。节点访问（NodePort）会路由到ClusterIP服务，这个ClusterIP服

务会自动创建。通过请求<NodeIP>:<NodePort>，可以从集群的外部访问一个NodePort服务。

- **负载均衡(LoadBalancer)**

  负载均衡(LoadBalancer)可以通过弹性负载均衡从公网访问到工作负载，与弹性IP方式相比提供了高可靠的保障。集群外访问推荐使用负载均衡类型。

- **DNAT网关(DNAT)**

  可以为集群节点提供网络地址转换服务，使多个节点可以共享使用弹性IP。与弹性IP方式相比增强了可靠性，弹性IP无需与单个节点绑定，任何节点状态的异常不影响其访问。

服务亲和（externalTrafficPolicy）

NodePort类型及LoadBalancer类型的Service接收请求时，会先访问到节点，然后转到Service，再由Service选择一个Pod转发到该Pod，但Service选择的Pod不一定在接收请求的节点上。默认情况下，从任意节点IP+服务端口都能访问到后端工作负载，当Pod不在接收请求的节点上时，请求会再跳转到Pod所在的节点，带来一定性能损失。

Service有一个配置参数（externalTrafficPolicy），用于设置Service是否希望将外部流量路由到节点本地或集群范围的端点，示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
spec:
  externalTrafficPolicy: Local
  ports:
  - name: service
    nodePort: 30000
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: NodePort
```

当externalTrafficPolicy取值为**Local**时，通过节点IP:服务端口的请求只会转发给本节点上的Pod，如果节点没有Pod的话请求会挂起。

当externalTrafficPolicy取值为**Cluster**时，请求会在集群内转发，从任意节点IP+服务端口都能访问到后端工作负载。

如不设置externalTrafficPolicy，默认取值为**Cluster**。

在CCE 控制台创建NodePort类型Service时，也可以通过“服务亲和”选项配置该参数。

总结服务亲和（externalTrafficPolicy）的两个选项对比如下：

表 7-25 服务亲和特性对比

对比维度	服务亲和（externalTrafficPolicy）	
	集群级别（Cluster）	节点级别（Local）
使用场景	适用于对性能要求不高，无需保留客户端源IP场景，此方式能为集群各节点带来更均衡的负载。	适用于客户端源IP需要保留且对性能要求较高的业务，但是流量仅会转发至容器所在的节点，不会做源地址转换。
访问方式	集群下所有节点的IP+访问端口均可以访问到此服务关联的负载。	只有通过负载所在节点的IP+访问端口才可以访问此服务关联的负载。
获取客户端源IP	无法获取到客户端源IP。	可以获取到客户端源IP。
访问性能	服务访问会因路由跳转导致一定性能损失，可能导致第二跳到另一个节点。	服务访问没有因路由跳转导致的性能损失。
负载均衡性	流量传播具有良好的整体负载均衡性。	存在潜在的不均衡流量传播风险。
其他特殊情况	-	在不同容器网络模型和服务转发模式下，可能出现集群内无法访问Service的情况，详情请参见**集群内无法访问Service的说明**。

集群内无法访问 Service 的说明

当Service设置了服务亲和为节点级别，即externalTrafficPolicy取值为Local时，在使用中可能会碰到从集群内部（节点上或容器中）访问不通的情况，回显类似如下内容：

```
upstream connect error or disconnect/reset before headers. reset reason: connection failure
或：
curl: (7) Failed to connect to 192.168.10.36 port 900: Connection refused
```

在集群中访问ELB地址时出现无法访问的场景较为常见，这是由于Kubernetes在创建Service时，kube-proxy会把ELB的访问地址作为外部IP（即External-IP，如下方回显所示）添加到iptables或IPVS中。如果客户端从集群内部发起访问ELB地址的请求，该地址会被认为是服务的外部IP，被kube-proxy直接转发，而不再经过集群外部的ELB。

```
# kubectl get svc nginx
NAME    TYPE          CLUSTER-IP      EXTERNAL-IP                PORT(S)       AGE
nginx   LoadBalancer  10.247.76.156   123.**.**.**,192.168.0.133  80:32146/TCP  37s
```

当externalTrafficPolicy的取值为Local时，在不同容器网络模型和服务转发模式下访问不通的场景如下：

说明

- 多实例的工作负载需要保证所有实例均可正常访问，否则可能出现概率性访问不通的情况。
- CCE Turbo集群（云原生2.0网络模型）中，仅当Service的后端对接使用主机网络（HostNetwork）的Pod时，亲和级别支持配置为节点级别。
- 表格中仅列举了可能存在访问不通的场景，其他不在表格中的场景即表示可以正常访问。

服务端发布服务类型	访问类型	客户端请求发起位置	容器隧道集群（IPVS）	VPC集群（IPVS）	容器隧道集群（IPTABLES）	VPC集群（IPTABLES）
节点访问类型Service	公网/私网	与服务Pod同节点	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问
		与服务Pod不同节点	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	正常访问	正常访问
		与服务Pod同节点的其他容器	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	无法访问

服务端发布服务类型	访问类型	客户端请求发起位置	容器隧道集群（IPVS）	VPC集群（IPVS）	容器隧道集群（IPTABLES）	VPC集群（IPTABLES）
		与服务Pod不同节点的其他容器	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问	访问服务端所在节点IP+NodePort — 正常访问 访问非服务端所在节点IP+NodePort — 无法访问
独享型负载均衡类型Service	私网	与服务Pod同节点	无法访问	无法访问	无法访问	无法访问
		与服务Pod同节点的其他容器	无法访问	无法访问	无法访问	无法访问
DNAT网关类型Service	公网	与服务Pod同节点	无法访问	无法访问	无法访问	无法访问
		与服务Pod不同节点	无法访问	无法访问	无法访问	无法访问
		与服务Pod同节点的其他容器	无法访问	无法访问	无法访问	无法访问
		与服务Pod不同节点的其他容器	无法访问	无法访问	无法访问	无法访问
nginx-ingress插件对接独享型ELB（Local）	私网	与cceaddon-nginx-ingress-controller Pod同节点	无法访问	无法访问	无法访问	无法访问

服务端发布服务类型	访问类型	客户端请求发起位置	容器隧道集群（IPVS）	VPC集群（IPVS）	容器隧道集群（IPTABLES）	VPC集群（IPTABLES）
		与cceaddon-nginx-ingress-controller Pod同节点的其他容器	无法访问	无法访问	无法访问	无法访问

解决这个问题通常有如下办法：

- （**推荐**）在集群内部访问使用Service的ClusterIP或服务域名访问。
- 将Service的externalTrafficPolicy设置为Cluster，即集群级别服务亲和。不过需要注意这会影响源地址保持。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","
eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

- 使用Service的pass-through特性，使用ELB地址访问时绕过kube-proxy，先访问ELB，经过ELB再访问到负载。具体请参见**LoadBalancer类型Service使用pass-through能力**。

  说明

  - 在CCE Standard集群中，当使用独享型负载均衡配置pass-through后，从工作负载Pod所在节点或同节点的其他容器中访问ELB的私网IP地址，会出现无法访问的问题。
  - 1.15及以下老版本集群暂不支持该能力。
  - IPVS网络模式下，对接同一个ELB的Service需保持pass-through设置情况一致。
  - 使用节点级别（Local）的服务亲和的场景下，会自动设置kubernetes.io/elb.pass-through为onlyLocal，开启pass-through能力。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.pass-through: "true"
```



```
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","
eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Local
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

7.3.2 集群内访问（ClusterIP）

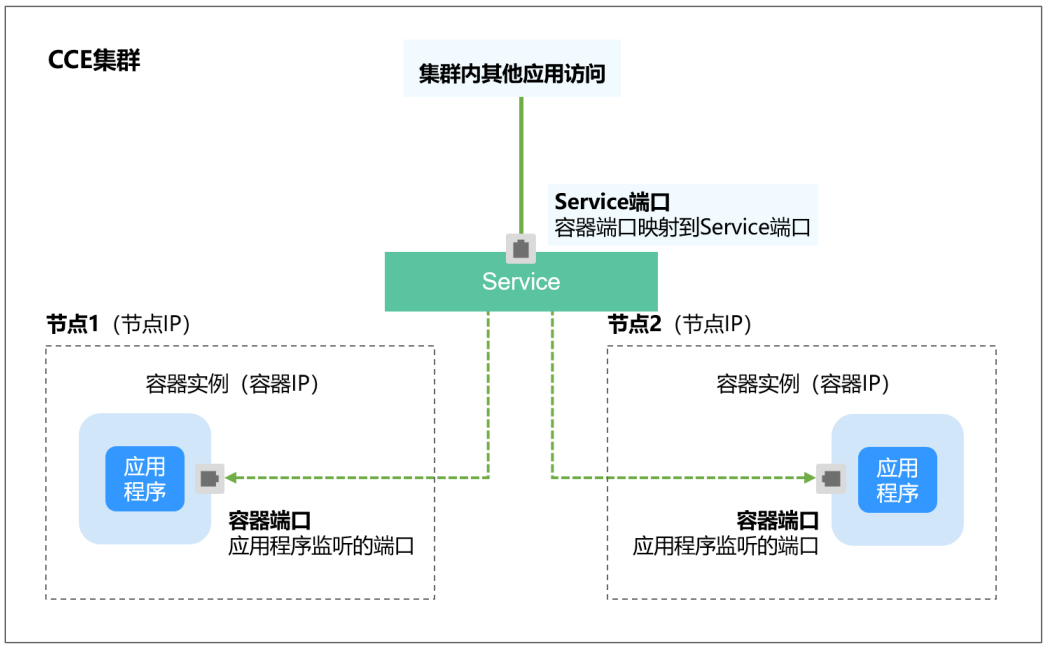
操作场景

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

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

访问通道、容器端口与访问端口映射如图7-37所示。

图 7-37 集群内访问



创建 ClusterIP 类型 Service

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置集群内访问参数。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“集群内访问”。
- **命名空间：**工作负载所在命名空间。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **协议版本：**请根据业务选择不同版本的IP地址，具体请参见**如何通过CCE搭建IPv4/IPv6双栈集群？**。**该功能仅在1.15及以上版本的集群创建时开启了IPv6功能才会显示。**
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

您可以通过kubectl命令行设置Service访问方式。本节以nginx为例，说明kubectl命令实现集群内访问的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建并编辑nginx-deployment.yaml和nginx-clusterip-svc.yaml文件。

其中，nginx-deployment.yaml和nginx-clusterip-svc.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

vi nginx-clusterip-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
```



```
    app: nginx
  name: nginx-clusterip
spec:
  ports:
  - name: service0
    port: 8080              # 访问Service的端口
    protocol: TCP           # 访问Service的协议，支持TCP和UDP
    targetPort: 80          # Service访问目标容器的端口，此端口与容器中运行的应用强相关，如本例中nginx镜像默认使用80端口
  selector:                 # 标签选择器，Service通过标签选择Pod，将访问Service的流量转发给Pod，此处选择带有 app:nginx 标签的Pod
    app: nginx
  type: ClusterIP           # Service的类型，ClusterIP表示在集群内访问
```

步骤3 创建工作负载。

kubectl create -f nginx-deployment.yaml

回显如下，表示工作负载已经创建。

```
deployment "nginx" created
```

kubectl get po

回显如下，工作负载状态为Running，表示工作负载已处于运行中状态。

```
NAME                     READY     STATUS            RESTARTS   AGE
nginx-2601814895-znhbr   1/1       Running           0          15s
```

步骤4 创建服务。

kubectl create -f nginx-clusterip-svc.yaml

回显如下，表示服务已开始创建。

```
service "nginx-clusterip" created
```

kubectl get svc

回显如下，表示服务已创建成功，CLUSTER-IP已生成。

```
# kubectl get svc
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes        ClusterIP   10.247.0.1     <none>        443/TCP    4d6h
nginx-clusterip   ClusterIP   10.247.74.52   <none>        8080/TCP   14m
```

步骤5 访问Service。

在集群内的容器或节点上都能够访问Service。

创建一个Pod并进入到容器内，使用curl命令访问Service的IP:Port或域名，如下所示。

其中域名后缀可以省略，在同个命名空间内可以直接使用nginx-clusterip:8080访问，跨命名空间可以使用nginx-clusterip.default:8080访问。

```
# kubectl run -i --tty --image nginx:alpine test --rm /bin/sh
If you don't see a command prompt, try pressing enter.
/ # curl 10.247.74.52:8080
<!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>
/ # curl nginx-clusterip.default.svc.cluster.local:8080
...
<h1>Welcome to nginx!</h1>
...
/ # curl nginx-clusterip.default:8080
...
<h1>Welcome to nginx!</h1>
...
/ # curl nginx-clusterip:8080
...
<h1>Welcome to nginx!</h1>
...
```

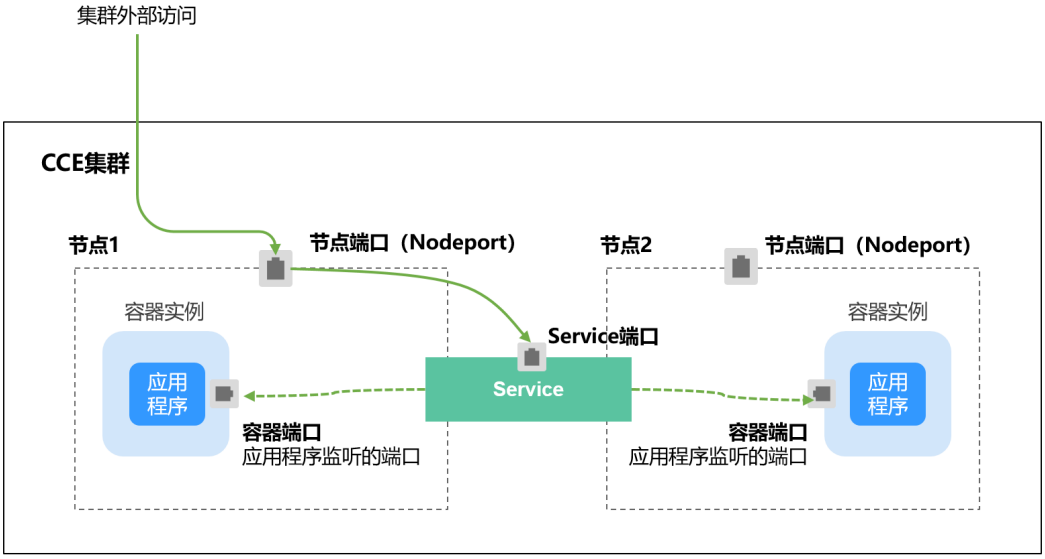
----结束

7.3.3 节点访问（NodePort）

操作场景

节点访问（NodePort）是指在每个节点的IP上开放一个静态端口，通过静态端口对外暴露服务。创建NodePort服务时，Kubernetes会自动创建一个集群内部IP地址（ClusterIP），集群外部的客户端通过访问 <NodeIP>:<NodePort>，流量会通过NodePort服务对应的ClusterIP转发到对应的Pod。

图 7-38 NodePort 访问



约束与限制

- “节点访问（NodePort）”默认为VPC内网访问，如果需要使用弹性IP通过公网访问该服务，请提前在集群的节点上绑定弹性IP。
- 创建Service后，如果服务亲和从集群级别切换为节点级别，连接跟踪表将不会被清理，建议用户创建Service后不要修改服务亲和属性，如需修改请重新创建Service。
- CCE Turbo集群中，仅当Service的后端对接使用主机网络（HostNetwork）的Pod时，亲和级别支持配置为节点级别。
- VPC网络模式下，当某容器A通过NodePort类型服务发布时，且服务亲和设置为节点级别（即externalTrafficPolicy为local），部署在同节点的容器B将无法通过节点IP+NodePort访问容器A。
- v1.21.7及以上的集群创建的NodePort类型服务时，节点上的NodePort端口默认不会用netstat显示：如果集群转发模式为iptables，可使用**iptables -t nat -L**查看端口；如果集群转发模式为IPVS，可使用**ipvsadm -Ln**查看端口。

创建 NodePort 类型 Service

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置集群内访问参数。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“节点访问”。
- **命名空间：**工作负载所在命名空间。
- **服务亲和：**详情请参见**服务亲和（externalTrafficPolicy）**。
  - 集群级别：集群下所有节点的IP+节点端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源IP。
  - 节点级别：只有通过负载所在节点的IP+节点端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源IP。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **IPv6：**默认不开启，开启后服务的集群内IP地址（ClusterIP）变为IPv6地址，具体请参见**如何通过CCE搭建IPv4/IPv6双栈集群?** 。**该功能仅在1.15及以上版本的集群创建时开启了IPv6功能才会显示。**
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 节点端口：即NodePort，建议选择“自动生成”；也可以指定端口，默认范围为30000-32767。

步骤4 单击“确定”，创建Service。

----结束

kubectl 命令行创建

您可以通过kubectl命令行设置Service访问方式。本节以nginx为例，说明kubectl命令实现节点访问的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建并编辑nginx-deployment.yaml以及nginx-nodeport-svc.yaml文件。

其中，nginx-deployment.yaml和nginx-nodeport-svc.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

vi nginx-nodeport-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx-nodeport
spec:
  ports:
  - name: service
    nodePort: 30000     # 节点端口，取值范围为30000-32767
    port: 8080          # 访问Service的端口
    protocol: TCP       # 访问Service的协议，支持TCP和UDP
    targetPort: 80      # Service访问目标容器的端口，此端口与容器中运行的应用强相关，如本例中nginx镜像默认使用80端口
  selector:             # 标签选择器，Service通过标签选择Pod，将访问Service的流量转发给Pod，此处选择带有app:nginx 标签的Pod
    app: nginx
  type: NodePort        # Service的类型，NodePort表示在通过节点端口访问
```

步骤3 创建工作负载。

kubectl create -f nginx-deployment.yaml

回显如下，表示工作负载已创建完成。

```
deployment "nginx" created
```

kubectl get po

回显如下，工作负载状态为Running，表示工作负载已处于运行状态。


```
NAME                     READY     STATUS            RESTARTS   AGE
nginx-2601814895-qhxqv   1/1       Running           0          9s
```

步骤4 创建服务。

kubectl create -f nginx-nodeport-svc.yaml

回显如下，表示服务开始创建。

```
service "nginx-nodeport" created
```

kubectl get svc

回显如下，表示服务已创建完成。

```
# kubectl get svc
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
kubernetes       ClusterIP   10.247.0.1     <none>         443/TCP          4d8h
nginx-nodeport   NodePort    10.247.30.40   <none>         8080:30000/TCP   18s
```

步骤5 访问Service。

默认情况下，NodePort类型Service可以通过**任意节点IP:节点端口**访问。

在集群同VPC下或集群容器内都可以访问，如果给节点绑定公网IP，也可以使用公网IP访问。如下所示，在集群上创建一个容器，从容器中使用节点IP:节点端口访问。

```
# kubectl get node -owide
NAME          STATUS   ROLES    AGE    INTERNAL-IP    EXTERNAL-IP   OS-IMAGE              KERNEL-
VERSION               CONTAINER-RUNTIME
10.100.0.136   Ready    <none>   152m   10.100.0.136   <none>        CentOS Linux 7 (Core)
3.10.0-1160.25.1.el7.x86_64   docker://18.9.0
10.100.0.5     Ready    <none>   152m   10.100.0.5     <none>        CentOS Linux 7 (Core)
3.10.0-1160.25.1.el7.x86_64   docker://18.9.0
# kubectl run -i --tty --image nginx:alpine test --rm /bin/sh
If you don't see a command prompt, try pressing enter.
/ # curl 10.100.0.136:30000
<!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>
/ #
```

----结束

7.3.4 负载均衡（LoadBalancer）

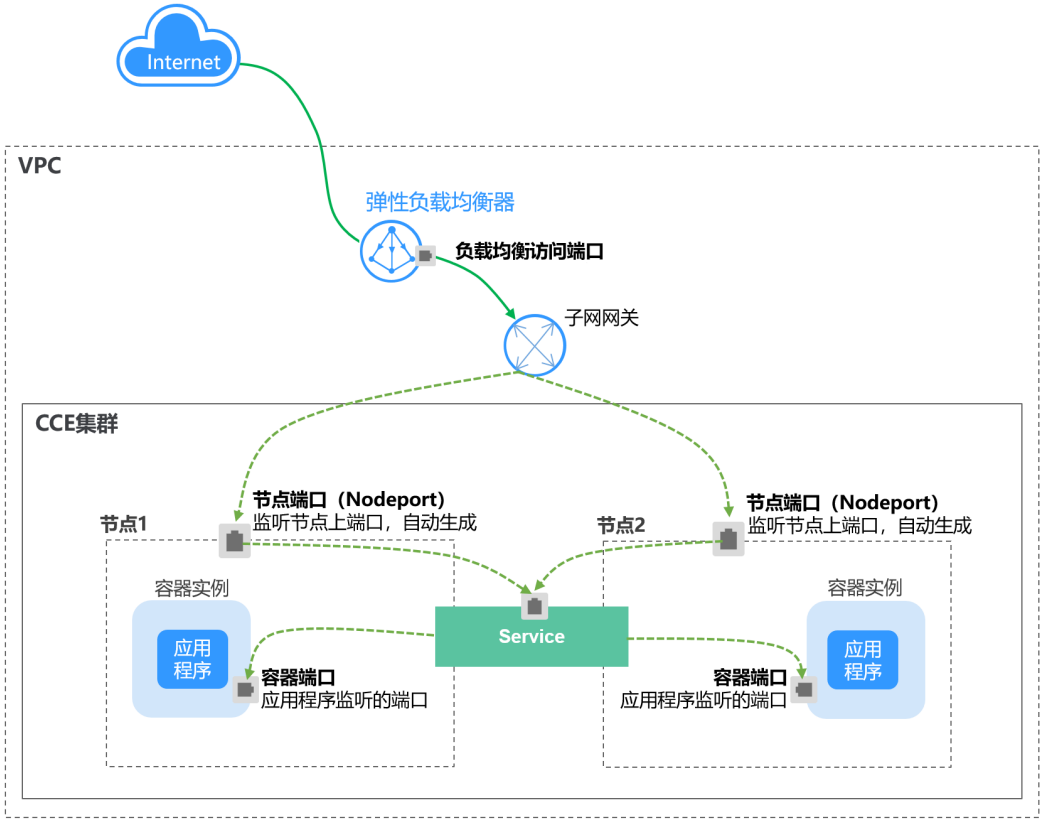
7.3.4.1 创建负载均衡类型的服务

操作场景

负载均衡（LoadBalancer）类型的服务可以通过弹性负载均衡（ELB）从公网访问到工作负载，与弹性IP方式相比提供了高可靠的保障。负载均衡访问方式由公网弹性负载均衡服务地址以及设置的访问端口组成，例如“10.117.117.117:80”。

在访问负载均衡类型的服务时，从ELB过来的流量会先访问到节点，然后通过Service转发到Pod。

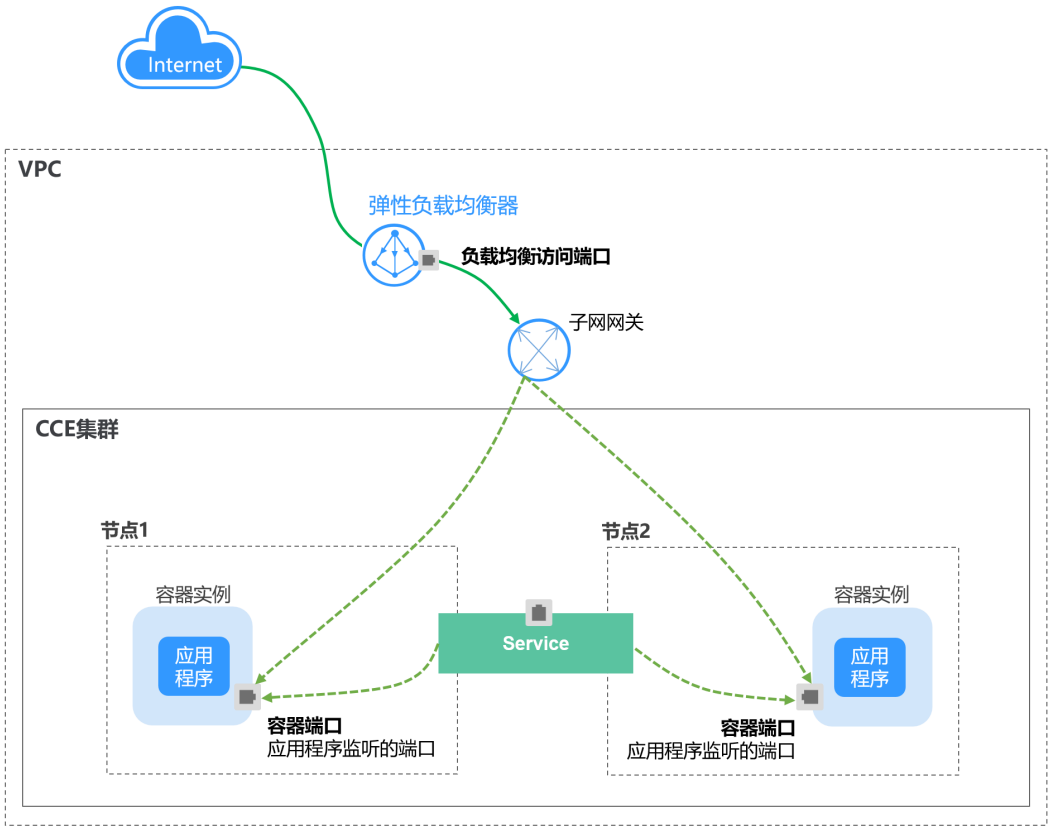
图 7-39 负载均衡（LoadBalancer）



在使用**CCE Turbo集群 + 独享型ELB实例**时，支持ELB直通Pod，使部署在容器中的业务时延降低、性能无损耗。

从集群外部访问时，从ELB直接转发到Pod；集群内部访问可通过Service转发到Pod。

图 7-40 ELB 直通容器



约束与限制

- CCE中的负载均衡（LoadBalancer）访问类型使用弹性负载均衡 ELB提供网络访问，存在如下产品约束：
  - 自动创建的ELB实例建议不要被其他资源使用，否则会在删除时被占用，导致资源残留。
  - v1.15及之前版本集群使用的ELB实例请不要修改监听器名称，否则可能导致无法正常访问。
- 创建Service后，如果服务亲和从集群级别切换为节点级别，连接跟踪表将不会被清理，建议用户创建Service后不要修改服务亲和属性，如需修改请重新创建Service。
- 当服务亲和设置为节点级别（即**externalTrafficPolicy**为Local）时，集群内部可能使用ELB地址访问不通，具体情况请参见**集群内无法访问Service的说明**。
- CCE Turbo集群（云原生2.0网络模型）中，仅当Service的后端对接使用主机网络（HostNetwork）的Pod时，亲和级别支持配置为节点级别。
- 独享型ELB仅支持1.17及以上集群。
- 独享型ELB规格必须支持网络型（TCP/UDP），且网络类型必须支持私网（有私有IP地址）。如果需要Service支持HTTP，则独享型ELB规格需要为网络型（TCP/UDP）和应用型（HTTP/HTTPS）。
- 集群服务转发模式为IPVS时，不支持配置节点的IP作为Service的externalIP，会导致节点不可用。
- IPVS模式集群下，Ingress和Service使用相同ELB实例时，无法在集群内的节点和容器中访问Ingress，因为kube-proxy会在ipvs-0的网桥上挂载LB类型的Service地

址，Ingress对接的ELB的流量会被ipvs-0网桥劫持。建议Ingress和Service使用不同ELB实例。

创建 LoadBalancer 类型 Service

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置参数。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **命名空间：**工作负载所在命名空间。
- **服务亲和：**详情请参见**服务亲和（externalTrafficPolicy）**。
  - 集群级别：集群下所有节点的IP+访问端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源IP。
  - 节点级别：只有通过负载所在节点的IP+访问端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源IP。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **协议版本：**默认不开启，开启后服务的集群内IP地址（ClusterIP）可以选择设置为IPv6地址，具体请参见**如何通过CCE搭建IPv4/IPv6双栈集群？**。**该功能仅在1.15及以上版本的集群创建时开启了IPv6功能才会显示。**
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。

  ELB类型可选择“独享型”或“共享型”，独享型ELB还可以根据支持的协议类型选择“网络型（TCP/UDP）”、“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP）&应用型（HTTP/HTTPS）”。

  创建方式可选择“选择已有”或“自动创建”。不同创建方式的配置详情请参见**表7-26**。

表 7-26 ELB 配置

创建方式	配置
选择已有	仅支持选择与集群在同一个VPC下的ELB实例。如果没有可选的ELB实例，请单击“创建负载均衡器”跳转到ELB控制台创建。

创建方式	配置
自动创建	- 实例名称：请填写ELB名称。 - 企业项目：该参数仅对开通企业项目的企业客户账号显示。企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 - 可用区（仅独享型ELB支持）：可以选择在多个可用区创建负载均衡实例，提高服务的可用性。如果业务需要考虑容灾能力，建议选择多个可用区。 - 前端子网（仅独享型ELB支持）：用于分配ELB实例对外服务的IP地址。 - 后端子网（仅独享型ELB支持）：用于与后端服务建立连接的IP地址。 - 网络型规格/应用型规格/规格（仅独享型ELB支持）： ■ 固定规格：适用于业务用量较为稳定的场景，按固定规格折算收取每小时使用的容量费用。 - 弹性公网IP：选择“自动创建”时，可配置公网带宽的计费方式及带宽大小。 - 资源标签：通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。

负载均衡配置：您可以单击负载均衡配置的“编辑”按钮配置ELB实例的参数，在弹出窗口中配置ELB实例的参数。

- 分配策略：可选择加权轮询算法、加权最少连接或源IP算法。

  说明

  - 加权轮询算法：根据后端服务器的权重，按顺序依次将请求分发给不同的服务器。它用相应的权重表示服务器的处理性能，按照权重的高低以及轮询方式将请求分配给各服务器，相同权重的服务器处理相同数目的连接数。常用于短连接服务，例如HTTP等服务。
  - 加权最少连接：最少连接是通过当前活跃的连接数来估计服务器负载情况的一种动态调度算法。加权最少连接就是在最少连接数的基础上，根据服务器的不同处理能力，给每个服务器分配不同的权重，使其能够接受相应权值数的服务请求。常用于长连接服务，例如数据库连接等服务。
  - 源IP算法：将请求的源IP地址进行Hash运算，得到一个具体的数值，同时对后端服务器进行编号，按照运算结果将请求分发到对应编号的服务器上。这可以使得对不同源IP的访问进行负载分发，同时使得同一个客户端IP的请求始终被派发至某特定的服务器。该方式适合负载均衡无cookie功能的TCP协议。

- 会话保持：默认不启用，可选择“源IP地址”。基于源IP地址的简单会话保持，即来自同一IP地址的访问请求转发到同一台后端服务器上。

  说明

  当**分配策略**使用源IP算法时，不支持设置会话保持。

- **健康检查**：设置负载均衡的健康检查配置。
  - 全局检查：全局检查仅支持使用相同协议的端口，无法对多个使用不同协议的端口生效，建议使用“自定义检查”。
  - 自定义检查：在**端口配置**中对多种不同协议的端口设置健康检查。关于自定义检查的YAML定义，请参见**为负载均衡类型的Service指定多个端口配置健康检查**。

表 7-27 健康检查参数

参数	说明
协议	当**端口配置**协议为TCP时，支持TCP和HTTP协议；当**端口配置**协议为UDP时，支持UDP协议。 – 检查路径（仅HTTP健康检查协议支持）：指定健康检查的URL地址。检查路径只能以/开头，长度范围为1-80。
端口	健康检查默认使用业务端口（Service的NodePort和容器端口）作为健康检查的端口；您也可以重新指定端口用于健康检查，重新指定端口会为服务增加一个名为cce-healthz的服务端口配置。 – 节点端口：使用共享型负载均衡或不关联ENI实例时，节点端口作为健康检查的检查端口；如不指定将随机一个端口。取值范围为30000-32767。 – 容器端口：使用独享型负载均衡关联ENI实例时，容器端口作为健康检查的检查端口。取值范围为1-65535。
检查周期（秒）	每次健康检查响应的最大间隔时间，取值范围为1-50。
超时时间（秒）	每次健康检查响应的最大超时时间，取值范围为1-50。
最大重试次数	健康检查最大的重试次数，取值范围为1-10。

- **端口配置**：
  - 协议：请根据业务的协议类型选择。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 服务端口：Service使用的端口，端口范围为1-65535。

说明

负载均衡服务支持创建某个端口范围的监听器，您最多可为每个监听器添加10个互不重叠的监听端口段。

为负载均衡服务配置区间端口监听需满足以下条件：

- 集群版本为v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0、v1.30.1-r0及以上。
- 使用独享型ELB且选择TCP/UDP/TLS协议。
- 该功能依赖ELB能力，使用该功能前请确认当前区域是否支持。ELB已发布区域请参见**四层协议全端口监听和转发**。

- 监听器前端协议：ELB监听器的前端协议，是客户端与负载均衡监听器建立流量分发连接所使用的协议。当选择独享型负载均衡器类型时，包含“应用型（HTTP/HTTPS）”方可支持配置HTTP/HTTPS。
- 健康检查：**健康检查**选项设置为“自定义检查”时，可以为不同协议的端口配置健康检查，参数说明请参见**表7-27**。

说明

在创建LoadBalancer类型Service时，会自动生成一个随机节点端口号（NodePort）。

- **监听器配置：**
  - SSL解析方式：当监听器端口**启用HTTPS/TLS**时可选择SSL解析方式。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
    - 单向认证：仅进行服务器端认证。如需认证客户端身份，请选择双向认证。
    - 双向认证：双向认证需要负载均衡实例与访问用户互相提供身份认证，从而允许通过认证的用户访问负载均衡实例，后端服务器无需额外配置双向认证。
  - CA证书：SSL解析方式选择“双向认证”时需要添加CA证书，用于认证客户端身份。CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在开启HTTPS双向认证功能时，只有当客户端能够出具指定CA签发的证书时，HTTPS连接才能成功。
  - 服务器证书：当监听器端口**启用HTTPS/TLS**时，必须选择一个服务器证书。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
  - SNI：当监听器端口**启用HTTPS/TLS**时，可以选择是否添加SNI证书。如果需要添加SNI证书，则证书中必须包含域名。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。

    如果无法根据客户端请求的域名查找到域名对应的SNI证书，则默认返回服务器证书。
  - 安全策略：当监听器端口**启用HTTPS/TLS**时，支持选择可用的安全策略。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
  - 后端协议：当监听器端口**启用HTTPS**时，支持使用HTTP或HTTPS协议对接后端服务，默认为HTTP。当监听器端口**启用TLS**时，支持使用TCP或TLS协议对接后端服务，默认为TCP。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
  - 访问控制：
    - 继承ELB已有配置：CCE不对ELB侧已有的访问控制进行修改。
    - 允许所有IP访问：不设置访问控制。
    - 白名单：仅所选IP地址组可以访问ELB地址。
    - 黑名单：所选IP地址组无法访问ELB地址。
  - 高级配置：

配置	说明	使用限制
获取监听器端口号	开启后可以将ELB实例的监听端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
获取客户端请求端口号	开启后可以将客户端的源端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
重写X-Forwarded-Host	开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
数据压缩	开启将对特定文件类型进行压缩；关闭则不会对任何文件类型进行压缩。 ■ Brotli支持压缩所有类型。 ■ Gzip支持压缩的类型如下：text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
空闲超时时间（秒）	客户端连接空闲超时时间。在超过空闲超时时间一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。	共享型ELB实例的端口使用UDP协议时不支持此配置。
请求超时时间（秒）	等待客户端请求超时时间。包括两种情况： ■ 读取整个客户端请求头的超时时长，如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 ■ 两个连续body体的数据包到达LB的时间间隔，超出请求超时时间将会断开连接。	仅端口**启用HTTP/HTTPS**时支持配置。
响应超时时间（秒）	等待后端服务器响应超时时间。请求转发后端服务器后，在等待超过响应超时时间没有响应，负载均衡将终止等待，并返回HTTP504错误码。	仅端口**启用HTTP/HTTPS**时支持配置。

配置	说明	使用限制
开启HTTP2	客户端与ELB之间的HTTPS请求的HTTP2功能的开启状态。 开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。	仅端口启用HTTPS时支持配置。

- **注解**：LoadBalancer类型Service有一些CCE定制的高级功能，通过注解annotations实现，具体注解的内容请参见**使用Annotation配置负载均衡类型的服务**。

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建-使用已有 ELB

您可以在创建工作负载时通过kubectl命令行设置Service访问方式。本节以nginx为例，说明kubectl命令实现负载均衡（ LoadBalancer)访问的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建并编辑nginx-deployment.yaml以及nginx-elb-svc.yaml文件。

其中，nginx-deployment.yaml和nginx-elb-svc.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

vi nginx-elb-svc.yaml

说明

若需要开启会话保持，工作负载的各实例需设置反亲和部署，即所有的实例都部署在不同节点上。具体请参见**设置工作负载亲和/反亲和调度（ podAffinity/podAntiAffinity ）**。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
```



```
annotations:
  kubernetes.io/elb.id: <your_elb_id>                    # ELB ID，替换为实际值
  kubernetes.io/elb.class: performance                   # 负载均衡器类型
  kubernetes.io/elb.lb-algorithm: ROUND_ROBIN            # 负载均衡器算法
  kubernetes.io/elb.session-affinity-mode: SOURCE_IP     # 会话保持类型为源IP
  kubernetes.io/elb.session-affinity-option: '{"persistence_timeout": "30"}'    # 会话保持时间（分钟）
  kubernetes.io/elb.health-check-flag: 'on'              # 开启ELB健康检查功能
  kubernetes.io/elb.health-check-option: '{
    "protocol":"TCP",
    "delay":"5",
    "timeout":"10",
    "max_retries":"3"
  }'
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80     #访问Service的端口，也是负载均衡上的监听器端口。
    protocol: TCP
    targetPort: 80  #Service访问目标容器的端口，此端口与容器中运行的应用强相关
    nodePort: 31128  #节点的端口号，如不指定，将在30000-32767范围内随机生成一个端口号
  type: LoadBalancer
```

上述示例通过Annotation（注解）实现负载均衡的一些高级功能，例如会话保持、健康检查等，对应的说明请参见**表7-28**。

除本示例中的功能外，如需了解更多高级功能相关注解及示例，请参见**使用Annotation配置负载均衡类型的服务**。

表 7-28 annotations 参数

参数	是否必填	参数类型	描述
kubernetes.io/elb.id	是	String	为负载均衡实例的ID。 在关联已有ELB时：必填。 **获取方法：** 在控制台的“服务列表”中，单击“网络 > 弹性负载均衡 ELB”，单击ELB的名称，在ELB详情页的“基本信息”页签下找到“ID”字段复制即可。 **说明** 系统优先根据kubernetes.io/elb.id注解对接ELB，若此字段未指定，则会根据spec.loadBalancerIP字段（非必填，且仅1.23及以前版本可用）对接ELB。 请尽量不要使用spec.loadBalancerIP字段对接ELB，该字段在将来的集群版本中会被Kubernetes官方废弃，详情请参见**Deprecation**。

参数	是否必填	参数类型	描述
kubernetes.io/elb.class	是	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： • union：共享型负载均衡。 • performance：独享型负载均衡，仅支持1.17及以上集群，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。 **说明** 负载均衡类型的服务对接已有的独享型ELB时，该独享型ELB必须支持网络型（TCP/UDP）规格。
kubernetes.io/elb.lb-algorithm	否	String	后端云服务器组的负载均衡算法，默认值为“ROUND_ROBIN”。 取值范围： • ROUND_ROBIN：加权轮询算法。 • LEAST_CONNECTIONS：加权最少连接算法。 • SOURCE_IP：源IP算法。 **说明** 当该字段的取值为SOURCE_IP时，后端云服务器组绑定的后端云服务器的权重设置（weight字段）无效，且不支持开启会话保持。
kubernetes.io/elb.session-affinity-mode	否	String	支持基于源IP地址的简单会话保持，即来自同一IP地址的访问请求转发到同一台后端服务器上。 • 不启用：不填写该参数。 • 开启会话保持：需增加该参数，取值“SOURCE_IP”，表示基于源IP地址。 **说明** 当kubernetes.io/elb.lb-algorithm设置为“SOURCE_IP”（源IP算法）时，不支持开启会话保持。
kubernetes.io/elb.session-affinity-option	否	**表7-29** Object	ELB会话保持配置选项，可设置会话保持的超时时间。
kubernetes.io/elb.health-check-flag	否	String	是否开启ELB健康检查功能。 • 开启：空值或"on" • 关闭："off" 开启时需同时填写**kubernetes.io/elb.health-check-option**字段。

参数	是否必填	参数类型	描述
kubernetes.io/elb.health-check-option	否	表7-30 Object	ELB健康检查配置选项。

表 7-29 elb.session-affinity-option 字段数据结构说明

参数	是否必填	参数类型	描述
persistence_timeout	是	String	当elb.session-affinity-mode是“SOURCE_IP”时生效，设置会话保持的超时时间（分钟）。 默认值为："60"，取值范围：1-60。

表 7-30 elb.health-check-option 字段数据结构说明

参数	是否必填	参数类型	描述
delay	否	String	健康检查间隔（秒）。 默认值：5，取值范围：1-50
timeout	否	String	健康检查的超时时间（秒）。 默认值：10，取值范围1-50
max_retries	否	String	健康检查的最大重试次数。 默认值：3，取值范围1-10
protocol	否	String	健康检查的协议。 取值范围：“TCP”或者“HTTP”
path	否	String	健康检查的URL，协议是“HTTP”时配置。 默认值：“/” 取值范围：1-80字符

步骤3 创建工作负载。

kubectl create -f nginx-deployment.yaml

回显如下，表示工作负载已创建完成。

```
deployment/nginx created
```

kubectl get pod

回显如下，工作负载状态为Running状态，表示工作负载已运行中。


```
NAME                READY   STATUS    RESTARTS   AGE
nginx-2601814895-c1xhw   1/1     Running   0          6s
```

步骤4 创建服务。

kubectl create -f nginx-elb-svc.yaml

回显如下，表示服务已创建。

```
service/nginx created
```

kubectl get svc

回显如下，表示工作负载访问方式已设置成功。

```
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP      10.247.0.1       <none>         443/TCP        3d
nginx        LoadBalancer   10.247.130.196   10.78.42.242   80:31540/TCP   51s
```

步骤5 在浏览器中输入访问地址，例如输入10.78.42.242:80。10.78.42.242为负载均衡实例IP地址，80为对应界面上的访问端口。

可成功访问nginx。

图 7-41 通过负载均衡访问 nginx

----结束

通过 kubectl 命令行创建-自动创建 ELB

您可以在创建工作负载时通过kubectl命令行设置Service访问方式。本节以nginx为例，说明kubectl命令实现负载均衡（LoadBalancer）访问的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建并编辑nginx-deployment.yaml以及nginx-elb-svc.yaml文件。

其中，nginx-deployment.yaml和nginx-elb-svc.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
```



```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - image: nginx
      name: nginx
    imagePullSecrets:
    - name: default-secret
```

vi nginx-elb-svc.yaml

说明

若需要开启会话保持，工作负载的各实例需设置反亲和部署，即所有的实例都部署在不同节点上。具体请参见**设置工作负载亲和/反亲和调度（podAffinity/podAntiAffinity）**。

共享型负载均衡（公网访问）Service示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{
      "type": "public",
      "bandwidth_name": "cce-bandwidth-1551163379627",
      "bandwidth_chargemode": "bandwidth",
      "bandwidth_size": 5,
      "bandwidth_sharetype": "PER",
      "vip_subnet_cidr_id": "******",
      "vip_address": "**.**.**.**",
      "eip_type": "5_bgp"
    }'
    kubernetes.io/elb.enterpriseID: '0'       # 负载均衡所属企业项目ID
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN                 # 负载均衡器算法
    kubernetes.io/elb.session-affinity-mode: SOURCE_IP          # 会话保持类型为源IP
    kubernetes.io/elb.session-affinity-option: '{"persistence_timeout": "30"}'     # 会话保持时间（分钟）
    kubernetes.io/elb.health-check-flag: 'on'                   # 开启ELB健康检查功能
    kubernetes.io/elb.health-check-option: '{
      "protocol":"TCP",
      "delay":"5",
      "timeout":"10",
      "max_retries":"3"
    }'
    kubernetes.io/elb.tags: key1=value1,key2=value2           # 添加ELB资源标签
  labels:
    app: nginx
  name: nginx
spec:
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

独享型负载均衡（公网访问）Service示例 - 仅支持1.17及以上集群：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
  namespace: default
  annotations:
```



```
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.autocreate: '{
      "type": "public",
      "bandwidth_name": "cce-bandwidth-1626694478577",
      "bandwidth_chargemode": "bandwidth",
      "bandwidth_size": 5,
      "bandwidth_sharetype": "PER",
      "eip_type": "5_bgp",
      "vip_subnet_cidr_id": "******",
      "vip_address": "**.**.**.**",
      "elb_virsubnet_ids": ["******"],
      "ipv6_vip_virsubnet_id": "******",
      "available_zone": [
        ""
      ],
      "l4_flavor_name": "L4_flavor.elb.s1.small"
    }'
    kubernetes.io/elb.enterpriseID: '0'      # 负载均衡所属企业项目ID
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN                # 负载均衡器算法
    kubernetes.io/elb.session-affinity-mode: SOURCE_IP         # 会话保持类型为源IP
    kubernetes.io/elb.session-affinity-option: '{"persistence_timeout": "30"}'    # 会话保持时间（分钟）
    kubernetes.io/elb.health-check-flag: 'on'                  # 开启ELB健康检查功能
    kubernetes.io/elb.health-check-option: '{
      "protocol":"TCP",
      "delay":"5",
      "timeout":"10",
      "max_retries":"3"
    }'
    kubernetes.io/elb.tags: key1=value1,key2=value2          # 添加ELB资源标签
spec:
  selector:
    app: nginx
  ports:
  - name: cce-service-0
    targetPort: 80
    nodePort: 0
    port: 80
    protocol: TCP
  type: LoadBalancer
```

上述示例通过Annotation（注解）实现负载均衡的一些高级功能，例如会话保持、健康检查等，对应的说明请参见**表7-31**。

除本示例中的功能外，如需了解更多高级功能相关注解及示例，请参见**使用Annotation配置负载均衡类型的服务**。

表 7-31 annotations 参数

参数	是否必填	参数类型	描述
kubernetes.io/elb.class	是	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： • union：共享型负载均衡。 • performance：独享型负载均衡，仅支持1.17及以上集群，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。

参数	是否必填	参数类型	描述
kubernetes.io/elb.autocreate	是	**elb.autocreate** object	自动创建service关联的ELB **示例：** • 自动创建公网共享型ELB： 值为 '{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}' • 自动创建私网共享型ELB： 值为 '{"type":"inner", "name": "A-location-d-test"}'
kubernetes.io/elb.subnet-id	-	String	为集群所在子网的ID，取值范围：1-100字符。 • Kubernetes v1.11.7-r0及以下版本的集群自动创建时为必填参数。 • Kubernetes v1.11.7-r0以上版本的集群：可不填。 获取方法请参见：**VPC子网接口与OpenStack Neutron子网接口的区别是什么?**
kubernetes.io/elb.enterpriseID	否	String	**v1.15及以上版本的集群支持此字段，v1.15以下版本默认创建到default项目下。** 为ELB企业项目ID，选择后可以直接创建在具体的ELB企业项目下。 该字段不传（或传为字符串'0'），则将资源绑定给默认企业项目。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“企业 > 项目管理”，在打开的企业项目列表中单击要加入的企业项目名称，进入企业项目详情页，找到“ID”字段复制即可。

参数	是否必填	参数类型	描述
kubernetes.io/elb.lb-algorithm	否	String	后端云服务器组的负载均衡算法，默认值为“ROUND_ROBIN”。 取值范围： • ROUND_ROBIN：加权轮询算法。 • LEAST_CONNECTIONS：加权最少连接算法。 • SOURCE_IP：源IP算法。 **说明** 当该字段的取值为SOURCE_IP时，后端云服务器组绑定的后端云服务器的权重设置（weight字段）无效，且不支持开启会话保持。
kubernetes.io/elb.session-affinity-mode	否	String	支持基于源IP地址的简单会话保持，即来自同一IP地址的访问请求转发到同一台后端服务器上。 • 不启用：不填写该参数。 • 开启会话保持：需增加该参数，取值“SOURCE_IP”，表示基于源IP地址。 **说明** 当kubernetes.io/elb.lb-algorithm设置为“SOURCE_IP”（源IP算法）时，不支持开启会话保持。
kubernetes.io/elb.session-affinity-option	否	表7-29 Object	ELB会话保持配置选项，可设置会话保持的超时时间。
kubernetes.io/elb.health-check-flag	否	String	是否开启ELB健康检查功能。 • 开启：“（空值）”或“on” • 关闭：“off” 开启时需同时填写**kubernetes.io/elb.health-check-option**字段。
kubernetes.io/elb.health-check-option	否	表7-30 Object	ELB健康检查配置选项。
kubernetes.io/elb.tags	否	String	为ELB添加资源标签，仅自动创建ELB时支持设置，且集群版本需满足v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上。 格式为key=value，同时添加多个标签时以英文逗号（,）隔开。

表 7-32 elb.autocreate 字段数据结构说明

参数	是否必填	参数类型	描述
name	否	String	自动创建的负载均衡的名称。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。 默认名称：cce-lb+service.UID
type	否	String	负载均衡实例网络类型，公网或者私网。 ● public：公网型负载均衡 ● inner：私网型负载均衡 默认类型：inner
bandwidth_name	公网型负载均衡必填	String	带宽的名称，默认值为：cce-bandwidth-******。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。
bandwidth_chargemode	否	String	带宽付费模式。 ● bandwidth：按带宽 ● traffic：按流量 默认类型：bandwidth
bandwidth_size	公网型负载均衡必填	Integer	带宽大小，默认1Mbit/s~2000Mbit/s，请根据Region带宽支持范围设置。 调整带宽时的最小单位会根据带宽范围不同存在差异。 ● 小于等于300Mbit/s：默认最小单位为1Mbit/s。 ● 300Mbit/s~1000Mbit/s：默认最小单位为50Mbit/s。 ● 大于1000Mbit/s：默认最小单位为500Mbit/s。
bandwidth_sharetype	公网型负载均衡必填	String	带宽共享方式。 ● PER：独享带宽

参数	是否必填	参数类型	描述
eip_type	公网型负载均衡必填	String	弹性公网IP类型。 ● 5_telcom：电信 ● 5_union：联通 ● 5_bgp：全动态BGP ● 5_sbgp：静态BGP 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
vip_subnet_cidr_id	否	String	指定ELB所在的子网，该子网必须属于集群所在的VPC。 如不指定，则ELB与集群在同一个子网。 仅v1.21及以上版本的集群支持指定该字段。
vip_address	否	String	负载均衡器的内网IP。仅支持指定IPv4地址，不支持指定IPv6地址。 该IP必须为ELB所在子网网段中的IP。若不指定，自动从ELB所在子网网段中生成一个IP地址。 仅v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上版本集群支持指定该字段。
available_zone	是	Array of strings	负载均衡所在可用区。 可以通过**查询可用区列表**获取所有支持的可用区。 独享型负载均衡器独有字段。
l4_flavor_name	是	String	四层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段。
l7_flavor_name	否	String	七层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段，必须与l4_flavor_name对应规格的类型一致，即都为弹性规格或都为固定规格。

参数	是否必填	参数类型	描述
elb_virsubnet_ids	否	Array of strings	负载均衡后端所在子网，不填默认为集群子网。不同实例规格将占用不同数量子网IP，不建议使用其他资源（如集群，节点等）的子网网段。 独享型负载均衡器独有字段。 示例： "elb_virsubnet_ids": ["14567f27-8ae4-42b8-ae47-9f847a4690dd"]
ipv6_vip_virsubnet_id	否	String	双栈类型负载均衡器所在子网的IPv6网络ID，需要对应的子网开启IPv6，仅使用双栈集群时需填写。 独享型负载均衡器独有字段。

步骤3 创建工作负载。

kubectl create -f nginx-deployment.yaml

回显如下，表示工作负载已开始创建。

```
deployment/nginx created
```

kubectl get pod

回显如下，工作负载状态为Running状态，表示工作负载已运行中。

```
NAME                     READY   STATUS          RESTARTS   AGE
nginx-2601814895-c1xhw   1/1     Running         0          6s
```

步骤4 创建服务。

kubectl create -f nginx-elb-svc.yaml

回显如下，表示服务已创建。

```
service/nginx created
```

kubectl get svc

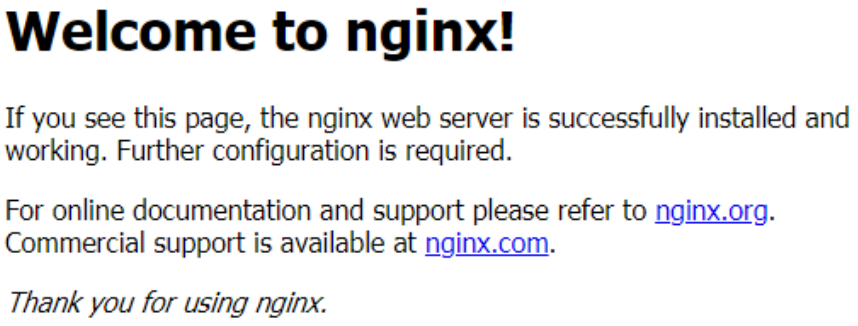
回显如下，表示工作负载访问方式已设置成功。

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP      10.247.0.1      <none>         443/TCP        3d
nginx        LoadBalancer   10.247.130.196  10.78.42.242   80:31540/TCP   51s
```

步骤5 在浏览器中输入访问地址，例如输入10.78.42.242:80。10.78.42.242为负载均衡实例IP地址，80为对应界面上的访问端口。

可成功访问nginx。

图 7-42 通过负载均衡访问 nginx



----结束

7.3.4.2 使用 Annotation 配置负载均衡类型的服务

通过在YAML中添加注解Annotation（注解），您可以实现CCE提供的一些高级功能。本文介绍在创建LoadBalancer类型的Service时可供使用的Annotation。

- **对接ELB**
- **会话保持**
- **健康检查**
- **使用HTTP/HTTPS协议**
- **配置服务器名称指示（SNI）**
- **动态调整后端云服务器权重**
- **pass-through能力**
- **黑名单/白名单设置**
- **主机网络**
- **设置超时时间**
- **添加资源标签**
- **使用HTTP/2**
- **开启gzip压缩**
- **配置获取客户端IP**
- **配置自定义EIP**
- **配置区间端口监听**

对接 ELB

表 7-33 对接 ELB 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.class	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： • union：共享型负载均衡。 • performance：独享型负载均衡，仅支持1.17及以上集群，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。	v1.9及以上
kubernetes.io/elb.id	String	**仅关联已有ELB的场景**：必填。 为负载均衡实例的ID。 **获取方法：** 在控制台的“服务列表”中，单击“网络 > 弹性负载均衡 ELB”，单击ELB的名称，在ELB详情页的“基本信息”页签下找到“ID”字段复制即可。 **说明** 系统优先根据kubernetes.io/elb.id注解对接ELB，若此字段未指定，则会根据spec.loadBalancerIP字段（非必填，且仅1.23及以前版本可用）对接ELB。 请尽量不要使用spec.loadBalancerIP字段对接ELB，该字段在将来的集群版本中会被Kubernetes官方废弃，详情请参见**Deprecation**。	v1.9及以上
kubernetes.io/elb.autocreate	**表7-50**	**仅自动创建ELB的场景**：必填。 **示例：** • 自动创建公网共享型ELB： 值为 '{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}' • 自动创建私网共享型ELB： 值为 '{"type":"inner", "name": "A-location-d-test"}'	v1.9及以上

参数	类型	描述	支持的集群版本
kubernetes.io/elb.enterpriseID	String	**仅自动创建ELB的场景**：选填。 **v1.15及以上版本的集群支持此字段，v1.15以下版本默认创建到default项目下。** 为ELB企业项目ID，选择后可以直接创建在具体的ELB企业项目下。 该字段不传（或传为字符串'0'），则将资源绑定给默认企业项目。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“企业 > 项目管理”，在打开的企业项目列表中单击要加入的企业项目名称，进入企业项目详情页，找到“ID”字段复制即可。	v1.15及以上
kubernetes.io/elb.subnet-id	String	**仅自动创建ELB的场景**：选填。 为集群所在子网的ID，取值范围：1-100字符。 ● Kubernetes v1.11.7-r0及以下版本的集群自动创建时：必填 ● Kubernetes v1.11.7-r0以上版本的集群：可不填。	v1.11.7-r0以下必填 v1.11.7-r0以上该字段废弃
kubernetes.io/elb.lb-algorithm	String	后端云服务器组的负载均衡算法，默认值为“ROUND_ROBIN”。 取值范围： ● ROUND_ROBIN：加权轮询算法。 ● LEAST_CONNECTIONS：加权最少连接算法。 ● SOURCE_IP：源IP算法。 **说明** 当该字段的取值为SOURCE_IP时，后端云服务器组绑定的后端云服务器的权重设置（weight字段）无效，且不支持开启会话保持。	v1.9及以上

上述注解的使用方法如下：

- 关联已有ELB场景：详情请参见**通过kubectl命令行创建-使用已有ELB**

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                    # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                   # 负载均衡器类型
```



```
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN              # 负载均衡器算法
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

- 自动创建ELB场景：详情请参见**通过kubectl命令行创建-自动创建ELB**

  共享型负载均衡：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{
      "type": "public",
      "bandwidth_name": "cce-bandwidth-1551163379627",
      "bandwidth_chargemode": "bandwidth",
      "bandwidth_size": 5,
      "bandwidth_sharetype": "PER",
      "eip_type": "5_bgp"
    }'
    kubernetes.io/elb.enterpriseID: '0'              # 负载均衡所属企业项目ID
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN      # 负载均衡器算法
  labels:
    app: nginx
  name: nginx
spec:
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

  独享型负载均衡：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.autocreate: '{
      "type": "public",
      "bandwidth_name": "cce-bandwidth-1626694478577",
      "bandwidth_chargemode": "bandwidth",
      "bandwidth_size": 5,
      "bandwidth_sharetype": "PER",
      "eip_type": "5_bgp",
      "available_zone": [
        ""
      ],
      "l4_flavor_name": "L4_flavor.elb.s1.small"
    }'
    kubernetes.io/elb.enterpriseID: '0'              # 负载均衡所属企业项目ID
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN      # 负载均衡器算法
spec:
  selector:
    app: nginx
  ports:
  - name: cce-service-0
```



```
  targetPort: 80
  nodePort: 0
  port: 80
  protocol: TCP
type: LoadBalancer
```

会话保持

表 7-34 会话保持注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.session-affinity-mode	String	支持基于源IP地址的简单会话保持，即来自同一IP地址的访问请求转发到同一台后端服务器上。 ● 不启用：不填写该参数。 ● 开启会话保持：需增加该参数，取值“SOURCE_IP”，表示基于源IP地址。 **说明** 当kubernetes.io/elb.lb-algorithm设置为“SOURCE_IP”（源IP算法）时，不支持开启会话保持。	v1.9及以上
kubernetes.io/elb.session-affinity-option	表7-53	ELB会话保持配置选项，可设置会话保持的超时时间。	v1.9及以上

上述注解的使用方法如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                    # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                   # 负载均衡器类型
    kubernetes.io/elb.session-affinity-mode: SOURCE_IP     # 会话保持类型为源IP
    kubernetes.io/elb.session-affinity-option: '{"persistence_timeout": "30"}'    # 会话保持时间（分钟）
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

健康检查

表 7-35 健康检查注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.health-check-flag	String	是否开启ELB健康检查功能。 • 开启：“（空值）”或“on” • 关闭：“off” 开启时需同时填写**kubernetes.io/elb.health-check-option**字段。	v1.9及以上
kubernetes.io/elb.health-check-option	**表7-51**	ELB健康检查配置选项。	v1.9及以上
kubernetes.io/elb.health-check-options	**表7-52**	ELB健康检查配置选项。支持Service每个端口单独配置，且可以只配置部分端口。 **说明** 不允许同时配置 "kubernetes.io/elb.health-check-option" 和 "kubernetes.io/elb.health-check-options"。	v1.19.16-r5及以上 v1.21.8-r0及以上 v1.23.6-r0及以上 v1.25.2-r0及以上

- kubernetes.io/elb.health-check-option的使用方法如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                    # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                  # 负载均衡器类型
    kubernetes.io/elb.health-check-flag: 'on'             # 开启ELB健康检查功能
    kubernetes.io/elb.health-check-option: '{
      "protocol":"TCP",
      "delay":"5",
      "timeout":"10",
      "max_retries":"3"
    }'
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

- kubernetes.io/elb.health-check-options的使用方法请参见**为负载均衡类型的Service指定多个端口配置健康检查**。

使用 HTTP/HTTPS 协议

表 7-36 使用 HTTP/HTTPS 协议注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.protocol-port	String	Service使用HTTP/HTTPS时，需设置协议及端口号，格式为protocol:port。 其中， • protocol：为监听器端口对应的协议，取值为http或https。 • ports：为Service的服务端口，即spec.ports[].port指定的端口。	v1.19.16及以上
kubernetes.io/elb.cert-id	String	ELB服务中的证书ID，作为HTTPS服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。	v1.19.16及以上

具体使用场景和说明请参见**为负载均衡类型的Service配置HTTP/HTTPS协议**。

配置服务器名称指示（SNI）

表 7-37 配置服务器名称指示（SNI）注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的SNI证书ID列表（SNI证书中必须带有域名），不同ID间使用英文逗号隔开。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。	v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本

需要开启HTTPS协议配合使用，具体使用场景和说明请参见**为负载均衡类型的Service配置服务器名称指示（SNI）**。

动态调整后端云服务器权重

表 7-38 动态调整后端云服务器权重注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.adaptive-weight	String	根据节点上的Pod数量动态调整ELB后端云服务器的权重。每个Pod收到的负载请求更加均衡。 • 开启：true • 关闭：false	v1.21及以上

说明

该参数在ELB直通Pod场景（即CCE Turbo集群中使用独享型ELB实例的场景）中无效。

上述注解的使用方法如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                    # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                   # 负载均衡器类型
    kubernetes.io/elb.adaptive-weight: 'true'              # 开启动态调整后端云服务器权重功能
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

pass-through 能力

表 7-39 pass-through 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.pass-through	String	集群内访问Service是否经过ELB。	v1.19及以上

具体使用场景和说明请参见**为负载均衡类型的Service配置pass-through能力**。

黑名单/白名单设置

表 7-40 ELB 访问控制注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.acl-id	String	• 不填写该参数时：表示CCE不对ELB侧访问控制进行修改。 • 参数值填写为空值时：表示允许所有IP访问。 • 参数值填写为ELB的IP地址组ID时：表示开启访问控制，为ELB设置IP地址黑名单或白名单。此时需同时填写kubernetes.io/elb.acl-status和kubernetes.io/elb.acl-type参数。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“网络 > 弹性负载均衡ELB”，在网络控制台中单击“弹性负载均衡 > IP地址组”，复制目标IP地址组的“ID”即可。详情请参见**IP地址组**。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上
kubernetes.io/elb.acl-status	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • on：表示开启访问控制。 • off：表示不开启访问控制。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上
kubernetes.io/elb.acl-type	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • black：表示黑名单，所选IP地址组无法访问ELB地址。 • white：表示白名单，仅所选IP地址组可以访问ELB地址。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上

上述注解的使用方法如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
```



```
    kubernetes.io/elb.id: <your_elb_id>                 # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                 # 负载均衡器类型
    kubernetes.io/elb.acl-id: <your_acl_id>              # ELB的IP地址组ID
    kubernetes.io/elb.acl-status: 'on'                  # 开启访问控制
    kubernetes.io/elb.acl-type: 'white'                  # 白名单控制
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

主机网络

表 7-41 主机网络注解

参数	类型	描述	支持的集群版本
kubernetes.io/hws-hostNetwork	String	如果Pod使用hostNetwork主机网络，使用该注解后ELB会将请求转发至主机网络。 取值范围： ● 开启：true ● 关闭：false，默认为false	v1.9及以上

上述注解的使用方法如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                 # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                # 负载均衡器类型
    kubernetes.io/hws-hostNetwork: 'true'                    # ELB会将请求转发至主机网络
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```


设置超时时间

表 7-42 设置超时时间注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.keepalive_timeout	String	客户端连接空闲超时时间，在超过keepalive_timeout时长一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。 取值： ● 若为TCP协议，取值范围为（10-4000s）默认值为300s。 ● 若为HTTP/HTTPS/TERMINATED_HTTPS监听器，取值范围为（0-4000s）默认值为60s。 ● UDP监听器不支持此字段。	独享型ELB：v1.19.16-r30、v1.21.10-r10、v1.23.8-r10、v1.25.3-r10及以上 共享型ELB：v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置超时时间**。

添加资源标签

表 7-43 添加资源标签注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.tags	String	为ELB添加资源标签，仅自动创建ELB时支持设置。 格式为key=value，同时添加多个标签时以英文逗号（,）隔开。	v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上

具体使用场景和说明请参见**通过kubectl命令行创建-自动创建ELB**。

使用 HTTP/2

表 7-44 使用 HTTP/2 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.http2-enable	String	表示HTTP/2功能的开启状态。开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。 取值范围： ● true：开启HTTP/2功能； ● false：关闭HTTP/2功能（默认为关闭状态）。 注意：**只有当监听器的协议为HTTPS时，才支持开启或关闭HTTP/2功能**。当监听器的协议为HTTP时，该字段无效，默认将其设置为false。	v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置HTTP/2**。

开启 gzip 压缩

表 7-45 开启 gzip 压缩注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.gzip-enabled	String	ELB支持开启数据压缩，通过数据压缩可缩小传输文件大小，提升文件传输效率减少带宽消耗。 开启将对特定文件类型进行压缩，关闭则不会对任何文件类型进行压缩。在默认情况下数据压缩为关闭。 支持的压缩类型如下： • Brotli支持压缩所有类型。 • Gzip支持压缩的类型包括：text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。 仅独享型ELB的HTTP/HTTPS类型监听器支持配置。删除开启数据压缩高级配置或对应的annotation，将不会对ELB侧配置进行修改。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置gzip数据压缩**。

配置 ELB 后端优雅退出

当前支持Service 4层场景配置ELB后端优雅退出时间，默认为开启，优雅退出时间默认为300s。

说明

该特性由ELB受限开放，使用前请提交工单给ELB服务进行申请。

表 7-46 转发规则优先级注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.connection-drain-enable	String	是否开启ELB后端优雅退出。	v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本
kubernetes.io/elb.connection-drain-timeout	String	ELB后端优雅退出时间。	

配置获取客户端 IP

表 7-47 配置获取客户端 IP 注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.transparent-client-ip	String	仅使用共享型ELB创建LoadBalancer类型的服务，且使用TCP/UDP协议时支持配置。 ● true：表示开启客户端源IP能力。 ● false：表示关闭客户端源IP能力。	v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置获取客户端IP**。

配置自定义 EIP

表 7-48 配置自定义 EIP 注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.custom-eip-id	String	自定义EIP的ID，您可以前往EIP控制台查看。该EIP必须是处于可绑定状态。	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0、v1.30.1-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置自定义EIP**。

配置区间端口监听

表 7-49 配置区间端口监听注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.port-ranges	String	使用独享型ELB且选择TCP/UDP/TLS协议时，支持创建某个端口范围的监听器，端口范围1~65535，您最多可为每个监听器添加10个互不重叠的监听端口段。 参数值格式如下，ports_name和port均不允许重复： '{"*<ports_name_1>*":["*<port_1>*,*<port_2>*","*<port_3>*,*<port_4>*"],"*<ports_name_2>*":["*<port_5>*,*<port_6>*","*<port_7>*,*<port_8>*"]}' 例如以下示例表示，端口配置名称为cce-service-0，其监听端口范围为100~200和300~400；端口配置名称为cce-service-1，其监听端口范围为500~600和700~800。 '{"cce-service-0":["100,200", "300,400"], "cce-service-1":["500,600", "700,800"]}' **说明** 该功能依赖ELB能力，使用该功能前请确认当前区域是否支持。ELB已发布区域请参见**四层协议全端口监听和转发**。	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0、v1.30.1-r0及以上版本

具体使用场景和说明请参见**为负载均衡类型的Service配置区间端口监听**。

自动创建 ELB 的参数说明

表 7-50 elb.autocreate 字段数据结构说明

参数	是否必填	参数类型	描述
name	否	String	自动创建的负载均衡的名称。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。 默认名称：cce-lb+service.UID
type	否	String	负载均衡实例网络类型，公网或者私网。 • public：公网型负载均衡 • inner：私网型负载均衡 默认类型：inner

参数	是否必填	参数类型	描述
bandwidth_name	公网型负载均衡必填	String	带宽的名称，默认值为：cce-bandwidth-******。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。
bandwidth_chargemode	否	String	带宽付费模式。 ● bandwidth：按带宽 ● traffic：按流量 默认类型：bandwidth
bandwidth_size	公网型负载均衡必填	Integer	带宽大小，默认1Mbit/s~2000Mbit/s，请根据Region带宽支持范围设置。 调整带宽时的最小单位会根据带宽范围不同存在差异。 ● 小于等于300Mbit/s：默认最小单位为1Mbit/s。 ● 300Mbit/s~1000Mbit/s：默认最小单位为50Mbit/s。 ● 大于1000Mbit/s：默认最小单位为500Mbit/s。
bandwidth_sharetype	公网型负载均衡必填	String	带宽共享方式。 ● PER：独享带宽
eip_type	公网型负载均衡必填	String	弹性公网IP类型。 ● 5_telcom：电信 ● 5_union：联通 ● 5_bgp：全动态BGP ● 5_sbgp：静态BGP 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
vip_subnet_cidr_id	否	String	指定ELB所在的子网，该子网必须属于集群所在的VPC。 如不指定，则ELB与集群在同一个子网。 仅v1.21及以上版本的集群支持指定该字段。

参数	是否必填	参数类型	描述
vip_address	否	String	负载均衡器的内网IP。仅支持指定IPv4地址，不支持指定IPv6地址。 该IP必须为ELB所在子网网段中的IP。若不指定，自动从ELB所在子网网段中生成一个IP地址。 仅v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上版本集群支持指定该字段。
available_zone	是	Array of strings	负载均衡所在可用区。 可以通过**查询可用区列表**获取所有支持的可用区。 独享型负载均衡器独有字段。
l4_flavor_name	是	String	四层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段。
l7_flavor_name	否	String	七层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段，必须与l4_flavor_name对应规格的类型一致，即都为弹性规格或都为固定规格。
elb_virsubnet_ids	否	Array of strings	负载均衡后端所在子网，不填默认为集群子网。不同实例规格将占用不同数量子网IP，不建议使用其他资源（如集群，节点等）的子网网段。 独享型负载均衡器独有字段。 示例： `"elb_virsubnet_ids": [ "14567f27-8ae4-42b8-ae47-9f847a4690dd" ]`
ipv6_vip_virsubnet_id	否	String	双栈类型负载均衡器所在子网的IPv6网络ID，需要对应的子网开启IPv6，仅使用双栈集群时需填写。 独享型负载均衡器独有字段。

表 7-51 elb.health-check-option 字段数据结构说明

参数	是否必填	参数类型	描述
delay	否	String	健康检查间隔（秒）。 默认值：5，取值范围：1-50
timeout	否	String	健康检查的超时时间（秒）。 默认值：10，取值范围1-50
max_retries	否	String	健康检查的最大重试次数。 默认值：3，取值范围1-10
protocol	否	String	健康检查的协议。 取值范围：“TCP”或者“HTTP”
path	否	String	健康检查的URL，协议是“HTTP”时配置。 默认值：“/” 取值范围：1-80字符

表 7-52 elb.health-check-options 字段数据结构说明

参数	是否必填	参数类型	描述
target_service_port	是	String	spec.ports添加健康检查的目标端口，由协议、端口号组成，如：TCP:80
monitor_port	否	String	重新指定的健康检查端口，不指定时默认使用业务端口。 **说明** 请确保该端口在Pod所在节点已被监听，否则会影响健康检查结果。
delay	否	String	健康检查间隔（秒） 默认值：5，取值范围：1-50
timeout	否	String	健康检查的超时时间（秒） 默认值：10，取值范围1-50
max_retries	否	String	健康检查的最大重试次数 默认值：3，取值范围1-10
protocol	否	String	健康检查的协议 默认值：取关联服务的协议 取值范围：“TCP”、“UDP”或者“HTTP”

参数	是否必填	参数类型	描述
path	否	String	健康检查的URL，协议是“HTTP”时需要配置 默认值：“/” 取值范围：1-80字符

表 7-53 elb.session-affinity-option 字段数据结构说明

参数	是否必填	参数类型	描述
persistence_timeout	是	String	当elb.session-affinity-mode是“SOURCE_IP”时生效，设置会话保持的超时时间（分钟）。 默认值为："60"，取值范围：1-60。

7.3.4.3 为负载均衡类型的 Service 配置 HTTP/HTTPS 协议

约束与限制

- Service使用HTTP/HTTPS协议仅v1.19.16及以上版本集群支持。

表 7-54 ELB 支持 HTTP/HTTPS 协议的场景

ELB类型	使用场景	是否支持HTTP/HTTPS协议	说明
共享型ELB	对接已有ELB	支持	-
	自动创建ELB	支持	-
独享型ELB	对接已有ELB	支持	• v1.19.16-r50、v1.21.11-r10、v1.23.9-r10、v1.25.4-r10、v1.27.1-r10以下版本：需要**同时支持4层和7层的flavor** • v1.19.16-r50、v1.21.11-r10、v1.23.9-r10、v1.25.4-r10、v1.27.1-r10及以上版本：需要**支持7层的flavor**

ELB类型	使用场景	是否支持HTTP/HTTPS协议	说明
	自动创建ELB	支持	• v1.19.16-r50、v1.21.11-r10、v1.23.9-r10、v1.25.4-r10、v1.27.1-r10以下版本：需要**同时支持4层和7层的flavor** • v1.19.16-r50、v1.21.11-r10、v1.23.9-r10、v1.25.4-r10、v1.27.1-r10及以上版本：需要**支持7层的flavor**

- 请勿将Ingress与使用HTTP/HTTPS的Service对接同一个ELB下的同一个监听器，否则将产生端口冲突。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举使用HTTP/HTTPS协议必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：“独享型”或“共享型”，其中独享型ELB需选择“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP）&应用型（HTTP/HTTPS）”，否则监听器端口将无法启用HTTP/HTTPS。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法使用HTTP/HTTPS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：设置监听器端口是否开启HTTP/HTTPS。当选择**独享型负载均衡器类型**时，需包含“应用型（HTTP/HTTPS）”方可支持配置HTTP/HTTPS。
- **监听器配置：**
  - SSL解析方式：当监听器端口**启用HTTPS**时可选择SSL解析方式。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
    - 单向认证：仅进行服务器端认证。如需认证客户端身份，请选择双向认证。

  - 双向认证：双向认证需要负载均衡实例与访问用户互相提供身份认证，从而允许通过认证的用户访问负载均衡实例，后端服务器无需额外配置双向认证。
- CA证书：SSL解析方式选择“双向认证”时需要添加CA证书，用于认证客户端身份。CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在开启HTTPS双向认证功能时，只有当客户端能够出具指定CA签发的证书时，HTTPS连接才能成功。
- 服务器证书：当监听器端口**启用HTTPS**时，必须选择一个服务器证书。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
- SNI：当监听器端口**启用HTTPS**时，可以选择是否添加SNI证书。如果需要添加SNI证书，证书中必须包含域名。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
- 安全策略：当监听器端口**启用HTTPS**时，支持选择可用的安全策略。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
- 后端协议：当监听器端口**启用HTTPS**时，支持使用HTTP或HTTPS协议对接后端服务，默认为HTTP。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。

说明

当发布多个HTTPS的服务，所有监听器会使用相同的证书配置。

图 7-43 配置 HTTP/HTTPS 协议

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

Service使用HTTP/HTTPS协议时，需要注意以下配置要求：

- 不同的ELB类型以及集群版本对flavor存在不同的要求，详情请参见**表7-54**。
- spec.ports中两个端口需要与kubernetes.io/elb.protocol-port中对应，本例中将443端口、80端口分别发布成HTTPS、HTTP协议。

以自动创建独享型ELB为例，配置示例如下，其中关键配置通过红色字体标出：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    # 在自动创建ELB参数中指定4层和7层的flavor
    kubernetes.io/elb.autocreate: '
      {
          "type": "public",
          "bandwidth_name": "cce-bandwidth-1634816602057",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "available_zone": [
              ""
          ],
          "l7_flavor_name": "L7_flavor.elb.s2.small",
          "l4_flavor_name": "L4_flavor.elb.s1.medium"
      }'
    kubernetes.io/elb.class: performance  # 独享型ELB
    kubernetes.io/elb.protocol-port: "https:443,http:80"  # HTTP/HTTPS协议及端口号，需要与spec.ports中的端口号对应
    kubernetes.io/elb.cert-id: "17e3b4f4bc40471c86741dc3aa211379"  # ELB服务中的证书ID
  labels:
    app: nginx
    name: test
  name: test
  namespace: default
spec:
  ports:
  - name: cce-service-0
    port: 443
    protocol: TCP
    targetPort: 80
  - name: cce-service-1
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
    version: v1
  sessionAffinity: None
  type: LoadBalancer
```


表 7-55 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.protocol-port	String	Service使用TLS/HTTP/HTTPS时，需设置协议及端口号，格式为protocol:port。 其中， • protocol：为监听器端口对应的协议，取值为tls、http或https。 • port：为Service的服务端口，即spec.ports[].port指定的端口。 例如，本示例中将443端口、80端口分别发布成HTTPS、HTTP协议，因此参数值为*https:443,http:80*。
kubernetes.io/elb.cert-id	String	ELB服务中的证书ID，作为TLS/HTTPS服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。

7.3.4.4 为负载均衡类型的 Service 配置服务器名称指示（SNI）

SNI证书是一种扩展服务器证书，允许同一个IP地址和端口号下对外提供多个访问域名，可以根据客户端请求的不同域名来使用不同的安全证书，确保HTTPS通信的安全性。

在配置SNI时，用户需要添加绑定域名的证书，客户端会在发起SSL握手请求时就提交请求的域名信息，负载均衡收到SSL请求后，会根据域名去查找证书。如果找到域名对应的证书，则返回该证书；如果没有找到域名对应的证书，则返回服务器默认证书。

说明

配置SNI后，如果您在CCE控制台删除SNI配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.13-r0及以上版本
  - v1.25集群：v1.25.8-r0及以上版本
  - v1.27集群：v1.27.5-r0及以上版本
  - v1.28集群：v1.28.3-r0及以上版本
  - 其他更高版本的集群
- 您已经在弹性负载均衡服务中创建好一个或多个SNI证书，且证书中指定了域名。详情请参见**创建证书**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举使用SNI的必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：“独享型”或“共享型”，其中独享型ELB需选择“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP）&应用型（HTTP/HTTPS）”，否则监听器端口将无法启用HTTP/HTTPS。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法使用HTTP/HTTPS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：本例中Service使用SNI需选择开启HTTPS。当选择**独享型负载均衡器类型**时，需包含“应用型（HTTP/HTTPS）”方可支持配置HTTP/HTTPS。
- **监听器配置：**
  - SSL解析方式：当监听器端口**启用HTTPS**时可选择SSL解析方式。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
    - 单向认证：仅进行服务器端认证。如需认证客户端身份，请选择双向认证。
    - 双向认证：双向认证需要负载均衡实例与访问用户互相提供身份认证，从而允许通过认证的用户访问负载均衡实例，后端服务器无需额外配置双向认证。
  - CA证书：SSL解析方式选择“双向认证”时需要添加CA证书，用于认证客户端身份。CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在开启HTTPS双向认证功能时，只有当客户端能够出具指定CA签发的证书时，HTTPS连接才能成功。
  - 服务器证书：选择一个服务器证书作为默认证书。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
  - SNI：选择添加SNI证书，证书中必须包含域名。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。

    如果无法根据客户端请求的域名查找到对应的SNI证书，则默认返回服务器证书。
  - 安全策略：当监听器端口**启用HTTPS**时，支持选择可用的安全策略。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。

- 后端协议：当监听器端口**启用HTTPS**时，支持使用HTTP或HTTPS协议对接后端服务，默认为HTTP。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。

图 7-44 配置服务器名称指示（SNI）

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

以关联已有ELB为例，Service使用SNI的YAML文件配置如下：

```
apiVersion: v1
kind: Service
metadata:
  name: test
  labels:
    app: test
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance        # ELB类型
    kubernetes.io/elb.id: 65318265-4f01-4541-a654-fa74e439dfd3  # 已有ELB的ID
    kubernetes.io/elb.protocol-port: https:80    # 需要开启的SNI的端口
    kubernetes.io/elb.cert-id: b64ab636f1614e1a960b5249c497a880    # HTTPS的服务器证书
    kubernetes.io/elb.tls-certificate-ids:
5196aa70b0f143189e4cb54991ba2286,8125d71fcc124aabbe007610cba42d60    # SNI证书ID列表
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN
spec:
  selector:
    app: test
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
```



```
type: LoadBalancer
loadBalancerIP: **.**.**.**      # ELB的私有IP
```

表 7-56 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.protocol-port	String	Service使用HTTP/HTTPS时，需设置协议及端口号，格式为protocol:port。 其中， ● protocol：为监听器端口对应的协议，取值为http或https。 ● ports：为Service的服务端口，即spec.ports[].port指定的端口。 例如，本示例中使用SNI时，Service协议必须设置为https，Service服务端口为80，因此参数值为*https:80*。
kubernetes.io/elb.cert-id	String	ELB服务中的证书ID，作为HTTPS服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的SNI证书ID列表（SNI证书中必须带有域名），不同ID间使用英文逗号隔开。 如果无法根据客户端请求的域名查找到对应的SNI证书，则默认返回服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。

7.3.4.5 为负载均衡类型的 Service 配置 HTTP/2

Service支持HTTP/2的方式暴露服务。在默认情况下，客户端与负载均衡之间采用HTTP1.X协议，使用HTTP/2可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。

说明

- 当负载均衡端口使用HTTPS协议时，支持使用HTTP/2功能。
- 配置HTTP/2后，如果您在CCE控制台删除开启HTTP/2的高级配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.13-r0及以上版本
  - v1.25集群：v1.25.8-r0及以上版本

  - v1.27集群：v1.27.5-r0及以上版本
  - v1.28集群：v1.28.3-r0及以上版本
  - 其他更高版本的集群
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：“独享型”或“共享型”，其中独享型ELB需选择“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP）&应用型（HTTP/HTTPS）”，否则监听器端口将无法启用HTTP/HTTPS。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法使用HTTP/HTTPS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：本例中Service使用HTTP/2需选择开启HTTPS。当选择**独享型负载均衡器类型**时，需包含“应用型（HTTP/HTTPS）”方可支持配置HTTP/HTTPS。
- **监听器配置：**
  - SSL解析方式：v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
    - 单向认证：仅进行服务器端认证。如需认证客户端身份，请选择双向认证。
    - 双向认证：双向认证需要负载均衡实例与访问用户互相提供身份认证，从而允许通过认证的用户访问负载均衡实例，后端服务器无需额外配置双向认证。
  - CA证书：SSL解析方式选择“双向认证”时需要添加CA证书，用于认证客户端身份。CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在开启HTTPS双向认证功能时，只有当客户端能够出具指定CA签发的证书时，HTTPS连接才能成功。
  - 服务器证书：使用HTTPS协议时需要选择一个服务器证书。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。

- SNI：选择添加SNI证书，证书中必须包含域名。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
- 高级配置：单击“添加自定义容器网络配置”，选择“开启HTTP/2”，并将状态设置为“开启”。

图 7-45 开启 HTTP/2

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

若需开启HTTP2功能，可在annotation字段中加入如下配置：

```
kubernetes.io/elb.http2-enable: 'true'
```

以关联已有ELB为例，yaml配置文件如下。

```
apiVersion: v1
kind: Service
metadata:
  name: test
  labels:
    app: test
    version: v1
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: 35cb350b-23e6-4551-ac77-10d5298f5204
    kubernetes.io/elb.protocol-port: https:443
    kubernetes.io/elb.cert-id: b64ab636f1614e1a960b5249c497a880
    kubernetes.io/elb.http2-enable: 'true'
```

```
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN
spec:
  selector:
    app: test
    version: v1
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 443
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: **.**.**.**
```

表 7-57 HTTP/2 参数说明

参数	参数类型	描述
kubernetes.io/elb.protocol-port	String	Service使用HTTP/HTTPS时，需设置协议及端口号，格式为protocol:port。 其中， ● protocol：为监听器端口对应的协议，取值为http或https。 ● ports：为Service的服务端口，即spec.ports[].port指定的端口。 例如，本示例中使用HTTPS协议，Service服务端口为443，因此参数值为*https:443*。
kubernetes.io/elb.cert-id	String	ELB服务中的证书ID，作为HTTPS服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。
kubernetes.io/elb.http2-enable	String	表示HTTP/2功能的开启状态。开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。 取值范围： ● true：开启HTTP/2功能； ● false：关闭HTTP/2功能（默认为关闭状态）。 注意：**只有当监听器的协议为HTTPS时，才支持开启或关闭HTTP/2功能**。当监听器的协议为HTTP时，该字段无效，默认将其设置为false。

7.3.4.6 为负载均衡类型的 Service 配置 HTTP/HTTPS 头字段

HTTP头字段是指在超文本传输协议（HTTP）的请求和响应消息中的消息头部分。HTTP头部字段可以根据需要自定义，本文介绍可通过HTTP和HTTPS监听器支持的非标准头字段实现的功能特性。

说明

- 配置HTTP/HTTPS头字段依赖ELB能力，使用该功能前请确认当前区域是否支持使用HTTP/HTTPS头字段，详情请参见**HTTP/HTTPS头字段**。
- 配置HTTP/HTTPS头字段后，如果您在CCE控制台删除HTTP/HTTPS头字段配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

表 7-58 头字段开关

头字段	功能开关	功能说明
X-Forwarded-Port	获取监听器端口号	开启获取监听器端口号开关，ELB可通过X-Forwarded-Port头字段获取监听器的端口号，传输到后端服务器的报文中。
X-Forwarded-For-Port	获取客户端请求端口号	开启获取客户端请求端口号开关，ELB可通过X-Forwarded-For-Port头字段获取客户端请求的端口号，传输到后端服务器的报文中。
X-Forwarded-Host	重写X-Forwarded-Host	开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.13-r0及以上版本
  - v1.25集群：v1.25.8-r0及以上版本
  - v1.27集群：v1.27.5-r0及以上版本
  - v1.28集群：v1.28.3-r0及以上版本
  - 其他更高版本的集群
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。

  - 类型：本例中仅支持选择“独享型”，且需选择“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP/TLS）&应用型（HTTP/HTTPS）”，否则监听器端口将无法启用HTTP或HTTPS。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法启用HTTP或HTTPS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：本例中Service需选择HTTP或HTTPS协议。
- **监听器配置：**
  - 高级配置：选择合适的头字段进行设置。

配置	说明	使用限制
获取监听器端口号	开启后可以将ELB实例的监听端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
获取客户端请求端口号	开启后可以将客户端的源端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。
重写X-Forwarded-Host	开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。

图 7-46 配置 HTTP/HTTPS 头字段

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

以关联已有ELB为例，Service配置HTTP/HTTPS头字段的YAML文件配置如下：

```
apiVersion: v1
kind: Service
metadata:
  name: test
  labels:
    app: nginx
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance                        # ELB类型，仅支持performance，即独享型ELB
    kubernetes.io/elb.id: 35cb350b-23e6-4551-ac77-10d5298f5204      # 已有ELB的ID
    kubernetes.io/elb.protocol-port: http:80                    # 使用HTTP协议80端口
    kubernetes.io/elb.x-forwarded-port: 'true'                    # 获取监听器端口号
    kubernetes.io/elb.x-forwarded-for-port: 'true'                 # 获取客户端请求端口号
    kubernetes.io/elb.x-forwarded-host: 'true'                    # 重写X-Forwarded-Host
spec:
  selector:
    app: nginx
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: **.**.**.**    # ELB的IP
```

表 7-59 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.x-forwarded-port	String	ELB可通过X-Forwarded-Port头字段获取监听器的端口号，传输到后端服务器的报文中。 ● true：开启获取监听器端口号开关。 ● false：关闭获取监听器端口号开关。
kubernetes.io/elb.x-forwarded-for-port	String	ELB可通过X-Forwarded-For-Port头字段获取客户端请求的端口号，传输到后端服务器的报文中。 ● true：开启获取客户端请求端口号开关。 ● false：关闭获取客户端请求端口号开关。
kubernetes.io/elb.x-forwarded-host	String	● true：开启重写X-Forwarded-Host开关，ELB以客户端请求头的Host重写X-Forwarded-Host传递到后端服务器。 ● false：关闭重写X-Forwarded-Host开关，ELB透传客户端的X-Forwarded-Host到后端服务器。

7.3.4.7 为负载均衡类型的 Service 配置超时时间

LoadBalancer Service支持设置连接空闲超时时间，即没有收到客户端请求的情况下保持连接的最长时间。如果在这个时间内没有新的请求， 负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。

说明

配置超时时间后，如果您在CCE控制台删除超时时间配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

约束与限制

支持设置超时时间的场景如下：

超时时间类型	支持的ELB类型	使用限制	支持的集群版本
空闲超时时间	独享型	-	• v1.19集群：v1.19.16-r30及以上版本 • v1.21集群：v1.21.10-r10及以上版本 • v1.23集群：v1.23.8-r10及以上版本 • v1.25集群：v1.25.3-r10及以上版本 • 其他更高版本集群
空闲超时时间	共享型	不支持UDP协议。	• v1.23集群：v1.23.13-r0及以上版本 • v1.25集群：v1.25.8-r0及以上版本 • v1.27集群：v1.27.5-r0及以上版本 • v1.28集群：v1.28.3-r0及以上版本 • 其他更高版本的集群
请求超时时间	独享型、共享型	仅支持HTTP、HTTPS协议。	
响应超时时间	独享型、共享型	仅支持HTTP、HTTPS协议。	

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

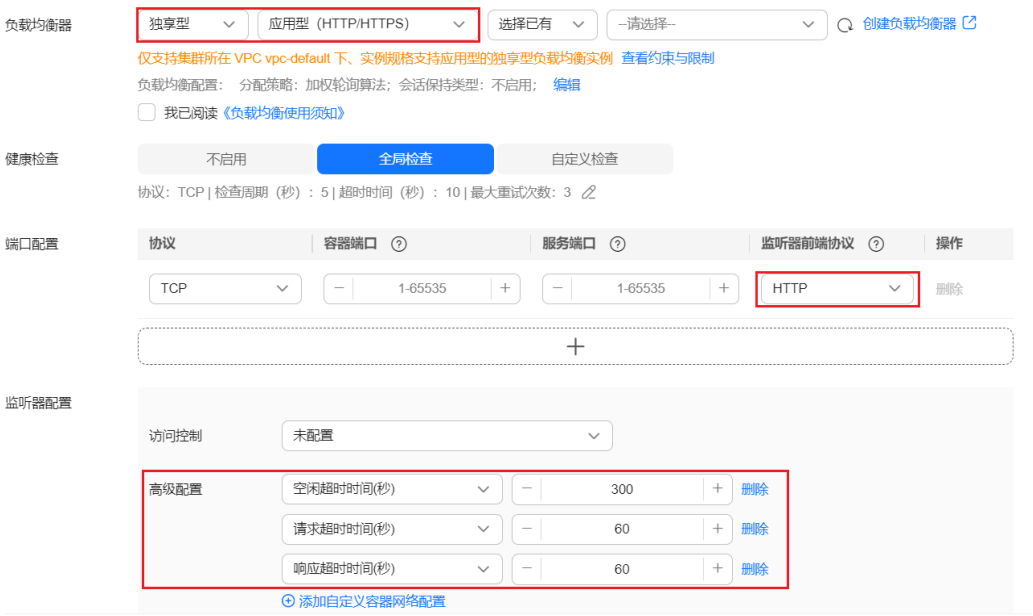
步骤3 设置Service参数。本示例中仅列举设置超时时间的必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。

- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：“独享型”或“共享型”。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择协议，其中共享型ELB使用UDP协议时不支持设置超时时间。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：请选择监听器的协议。不启用HTTP/HTTPS协议时，仅支持设置空闲超时时间。
- **监听器配置：**
  - 高级配置：选择合适的超时时间进行设置。

配置	说明	使用限制
空闲超时时间	客户端连接空闲超时时间。在超过空闲超时时间一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。	共享型ELB实例的端口使用UDP协议时不支持此配置。
请求超时时间	等待客户端请求超时时间。包括两种情况： ■ 读取整个客户端请求头的超时时长，如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 ■ 两个连续body体的数据包到达LB的时间间隔，超出请求超时时间将会断开连接。	仅端口**启用HTTP/HTTPS**时支持配置。
响应超时时间	等待后端服务器响应超时时间。请求转发后端服务器后，在等待超过响应超时时间没有响应，负载均衡将终止等待，并返回HTTP504错误码。	仅端口**启用HTTP/HTTPS**时支持配置。

图 7-47 配置超时时间



步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

当前支持通过注解的方式设置客户端连接空闲超时时间，示例如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.id: <your_elb_id>    #本示例中使用已有的独享型ELB，请替换为您的独享型ELB ID
    kubernetes.io/elb.class: performance  # ELB类型
    kubernetes.io/elb.protocol-port: http:80 # 使用HTTP协议80端口
    kubernetes.io/elb.keepalive_timeout: '300'  # 客户端连接空闲超时时间
    kubernetes.io/elb.client_timeout: '60'      # 等待客户端请求超时时间
    kubernetes.io/elb.member_timeout: '60'      # 等待后端服务器响应超时时间
  name: nginx
spec:
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```


表 7-60 annotation 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.keepalive_timeout	否	String	客户端连接空闲超时时间，在超过keepalive_timeout时长一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。 取值： ● 若为TCP协议，取值范围为10-4000s，默认值为300s。 ● 若为HTTP/HTTPS/TERMINATED_HTTPS监听器，取值范围为（0-4000s）默认值为60s。 ● 若为UDP协议，取值范围为10-4000s，默认值为300s。
kubernetes.io/elb.client_timeout	否	String	等待客户端请求超时时间，包括两种情况： ● 读取整个客户端请求头的超时时长：如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 ● 两个连续body体的数据包到达LB的时间间隔，超出client_timeout将会断开连接。 取值范围为1-300s，默认值为60s。
kubernetes.io/elb.member_timeout	否	String	等待后端服务器响应超时时间。请求转发后端服务器后，等待超过member_timeout时长没有响应，负载均衡将终止等待，并返回 HTTP504错误码。 取值范围为1-300s，默认值为60s。

7.3.4.8 为负载均衡类型的 Service 配置 TLS

TLS协议适用于需要超高性能和大规模TLS卸载的场景。您可以为Service配置TLS协议，转发来自客户端加密的TCP协议请求。

说明

Service配置TLS协议依赖ELB能力，使用该功能前请确认当前区域是否支持使用TLS协议。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上版本
  - v1.25集群：v1.25.9-r0及以上版本

  - v1.27集群：v1.27.6-r0及以上版本
  - v1.28集群：v1.28.4-r0及以上版本
  - 其他更高版本的集群
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

使用TLS协议时，不支持同时配置会话保持。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举使用TLS的必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：本例中仅支持选择“独享型”，且需选择“网络型（TCP/UDP/TLS）”或“网络型（TCP/UDP/TLS）&应用型（HTTP/HTTPS）”，否则监听器端口将无法启用TLS。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法使用TLS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 监听器前端协议：本例中Service需选择TLS协议。当选择**独享型负载均衡器类型**时，需包含“网络型（TCP/UDP/TLS）”方可支持配置TLS协议。
- **监听器配置：**
  - SSL解析方式：当监听器端口**启用HTTPS/TLS**时可选择SSL解析方式。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
    - 单向认证：仅进行服务器端认证。如需认证客户端身份，请选择双向认证。
    - 双向认证：双向认证需要负载均衡实例与访问用户互相提供身份认证，从而允许通过认证的用户访问负载均衡实例，后端服务器无需额外配置双向认证。
  - CA证书：SSL解析方式选择“双向认证”时需要添加CA证书，用于认证客户端身份。CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在

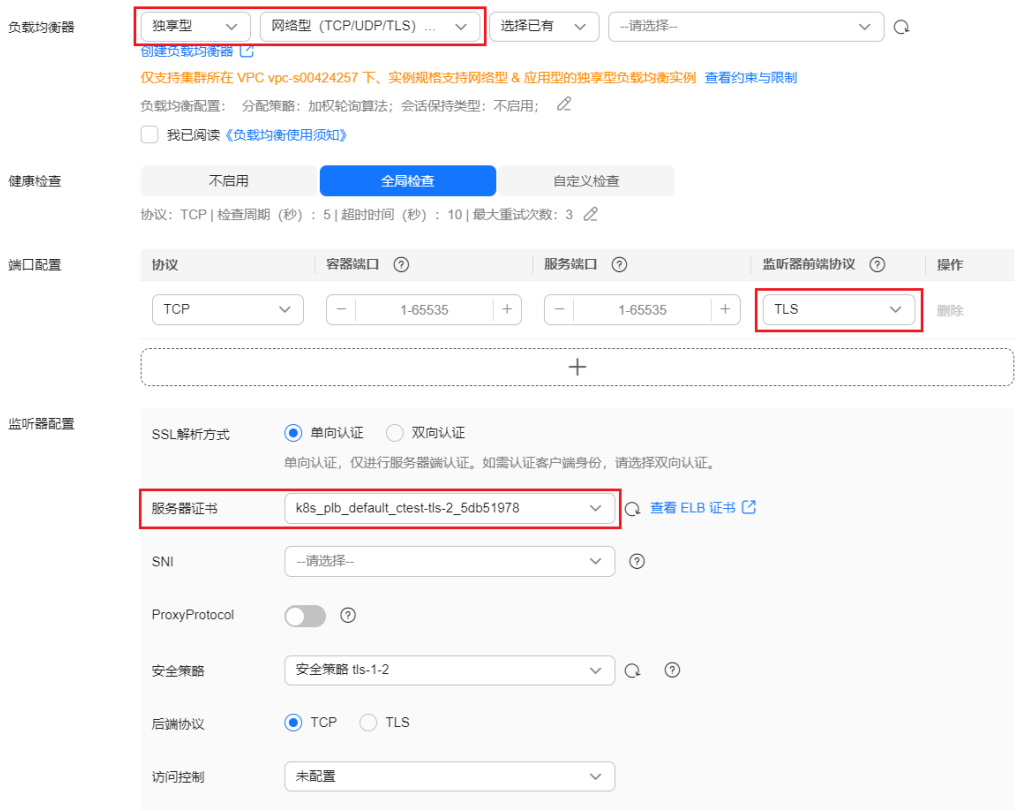
开启双向认证功能时，只有当客户端能够出具指定CA签发的证书时，连接才能成功。

- 服务器证书：选择一个服务器证书。如果当前无可选证书，需前往弹性负载均衡控制台进行创建，详情请参见**创建证书**。
- ProxyProtocol：支持通过ProxyProtocol协议携带客户端真实IP到后端服务器。

  说明

  请确保后端服务器具有解析ProxyProtocol协议的能力，否则可能导致业务中断，请谨慎开启。
- 安全策略：当监听器端口**启用TLS**时，支持选择可用的安全策略。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。
- 后端协议：当监听器端口**启用TLS**时，支持使用TCP或TLS协议对接后端服务，默认为TCP。v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持。

图 7-48 配置 TLS



步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

以关联已有ELB为例，Service使用TLS的YAML文件配置如下：

```
apiVersion: v1
kind: Service
metadata:
```



```
  name: test-tls
  labels:
    app: nginx
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance                                  # ELB类型，TLS监听器仅支持
performance，即独享型ELB
    kubernetes.io/elb.id: 35cb350b-23e6-4551-ac77-10d5298f5204            # 已有ELB的ID
    kubernetes.io/elb.protocol-port: tls:443                              # 需要开启TLS的端口
    kubernetes.io/elb.cert-id: 98e91cb03dea418582a438a212b461d5           # TLS的服务器证书
    kubernetes.io/elb.tls-certificate-ids: e59934f5bc7044f58693de79f1cb4b6d    # TLS的SNI证书
    kubernetes.io/elb.client-ca-cert-id: 5b5178323a2f4eddbafed065945d9069      # TLS双向认证中客户端
的CA证书
    kubernetes.io/elb.proxy-protocol-enable: 'true'                       # 是否开启ProxyProtocol协议，通过
ProxyProtocol协议携带客户端真实IP到后端服务器。
    kubernetes.io/elb.security-pool-protocol: 'on'                        # 后端安全协议，开启后，后端协议为
TLS，否则为TCP
    kubernetes.io/elb.security-policy-id: 175318e0-b6cb-44c5-80a2-0dc372f20df5 # 自定义安全策略ID，
优先级高于系统预置的安全策略
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2-fs                      # 系统预置的安全策略
spec:
  selector:
    app: nginx
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 443
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: **.**.**.**
```

表 7-61 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.protocol-port	String	Service使用TLS/HTTP/HTTPS时，需设置协议及端口号，格式为protocol:port。 其中， ● protocol：为监听器端口对应的协议，取值为tls、http或https。 ● port：为Service的服务端口，即spec.ports[].port指定的端口。 例如，本示例中创建TLS监听器，Service协议必须设置为tls，Service服务端口为443，因此参数值为*tls:443*。
kubernetes.io/elb.cert-id	String	ELB服务中的证书ID，作为TLS/HTTPS服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。

参数	参数类型	描述
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的SNI证书ID列表（SNI证书中必须带有域名），不同ID间使用英文逗号隔开。更新时，通过指定空字符串""来移除SNI证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。
kubernetes.io/elb.client-ca-cert-id	String	ELB服务中的CA证书ID，CA证书又称客户端CA公钥证书，用于验证客户端证书的签发者；在开启TLS/HTTPS双向认证功能时，只有当客户端能够出具指定CA签发的证书时，TLS/HTTPS连接才能成功。更新时，通过指定空字符串""来移除CA证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”，并筛选证书类型为CA证书。在列表中复制对应证书名称下的ID即可。
kubernetes.io/elb.security-pool-protocol	String	监听器前端协议为TLS/HTTPS时，可以开启后端安全协议，开启后后端协议为TLS/HTTPS。不支持更新已有监听器的后端安全协议，更新后仅对新创建的监听器生效（通过更新协议或者端口会创建新的监听器）。 ● on/true：表示开启 ● off/false：表示关闭
kubernetes.io/elb.security-policy-id	String	自定义安全策略的ID，优先级高于系统预置的安全策略。更新时，通过指定空字符串""来移除。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡 > TLS安全策略”，并选择“自定义策略”页签。在列表中复制对应策略名称下的ID即可。
kubernetes.io/elb.tls-ciphers-policy	String	系统预置的安全策略，默认为tls-1-0。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡 > TLS安全策略”，并选择“默认策略”页签。在列表中复制对应策略名称即可。
kubernetes.io/elb.proxy-protocol-enable	String	开启ProxyProtocol协议，仅在前端协议为TLS时生效，通过ProxyProtocol协议携带客户端真实IP到后端服务器。 ● on/true：表示开启 ● off/false：表示关闭 **说明** 请确保后端服务器具体解析ProxyProtocol协议的能力，否则可能导致业务中断，请谨慎开启。

7.3.4.9 为负载均衡类型的 Service 配置 gzip 数据压缩

ELB支持开启数据压缩，通过数据压缩可缩小传输文件大小，提升文件传输效率减少带宽消耗。

说明

- 该功能依赖ELB能力，使用该功能前请确认当前区域是否支持。ELB已支持的区域请参见**数据压缩**。
- 配置数据压缩后，如果您在CCE控制台删除数据压缩配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上版本
  - v1.25集群：v1.25.9-r0及以上版本
  - v1.27集群：v1.27.6-r0及以上版本
  - v1.28集群：v1.28.4-r0及以上版本
  - 其他更高版本的集群
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台创建

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置Service参数。本示例中仅列举必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：本例中仅支持选择“独享型”，且需选择“应用型（HTTP/HTTPS）”或“网络型（TCP/UDP/TLS）&应用型（HTTP/HTTPS）”，否则监听器端口将无法选择HTTP或HTTPS协议。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：请选择TCP协议，选择UDP协议将无法使用HTTP或HTTPS。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。

  - 监听器前端协议：本例中Service需选择HTTP或HTTPS协议。
- **监听器配置：**
  - 高级配置：选择合适的头字段进行设置。

配置	说明	使用限制
数据压缩	开启将对特定文件类型进行压缩；关闭则不会对任何文件类型进行压缩。 ■ Brotli支持压缩所有类型。 ■ Gzip支持压缩的类型如下：text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。	独享型ELB实例的端口**启用HTTP/HTTPS**时支持配置。

图 7-49 配置数据压缩

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

以关联已有ELB为例，Service配置数据压缩的YAML文件配置如下：

```
apiVersion: v1
kind: Service
metadata:
  name: test
  labels:
    app: nginx
  namespace: default
  annotations:
```



```
    kubernetes.io/elb.class: performance                # ELB类型，仅支持performance，即独享型ELB
    kubernetes.io/elb.id: 35cb350b-23e6-4551-ac77-10d5298f5204      # 已有ELB的ID
    kubernetes.io/elb.protocol-port: http:80                # 使用HTTP协议80端口
    kubernetes.io/elb.gzip-enabled: 'true'                  # 开启数据压缩
spec:
  selector:
    app: nginx
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: **.**.**.**    # ELB的IP
```

表 7-62 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.gzip-enabled	String	● true：开启，将对特定文件类型进行压缩。 ● false：关闭，不会对任何文件类型进行压缩。在默认情况下数据压缩为关闭。 支持的压缩类型如下： ● Brotli支持压缩所有类型。 ● Gzip支持压缩的类型包括：text/xml text/plain text/css application/javascript application/x-javascript application/rss +xml application/atom+xml application/xml application/json。 仅独享型ELB的HTTP/HTTPS类型监听器支持配置。

7.3.4.10 为负载均衡类型的 Service 配置黑名单/白名单访问策略

使用负载均衡类型的服务时，您可以通过添加白名单和黑名单的方式控制访问负载均衡监听器的IP。

- 白名单：指定的IP允许访问，而其它IP不能访问。
- 黑名单：指定的IP不能访问，而其它IP允许访问。

说明

配置黑名单/白名单访问策略后，如果您在CCE控制台删除黑名单/白名单访问策略配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.12-r0及以上版本
  - v1.25集群：v1.25.7-r0及以上版本
  - v1.27集群：v1.27.4-r0及以上版本

  - v1.28集群：v1.28.2-r0及以上版本
  - 其他更高版本的集群
- 已在ELB控制台创建一个IP地址组，详情请参见**创建IP地址组**。

通过控制台设置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“服务”页签，单击右上角“创建服务”。

步骤3 设置Service参数。

- **Service名称：**自定义Service名称，例如service-acl。
- **访问类型：**选择“负载均衡”类型。
- **服务亲和：**您可以根据需求选择“集群级别”或“节点级别”。二者差异说明请参见**服务亲和（externalTrafficPolicy）**。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**

  选择对接的ELB实例，仅支持与集群在同一个VPC下的ELB实例。如果没有可选的ELB实例，请单击“创建负载均衡器”跳转到ELB控制台创建。或者选择“自动创建”一个ELB实例，配置参数请参见**表7-26**。
- **健康检查：**默认为“全局检查”，您可根据需求进行设置。
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 服务端口：Service使用的端口，端口范围为1-65535。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
- **访问控制：**
  - 继承ELB已有配置：CCE不对ELB侧已有的访问控制进行修改。
  - 允许所有IP访问：不设置访问控制。
  - 白名单：仅所选IP地址组可以访问ELB地址。
  - 黑名单：所选IP地址组无法访问ELB地址。

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行设置

以使用已有ELB创建Service的场景为例，YAML配置示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                 # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                 # 负载均衡器类型
    kubernetes.io/elb.acl-id: <your_acl_id>              # ELB的IP地址组ID
    kubernetes.io/elb.acl-status: 'on'                  # 开启访问控制
    kubernetes.io/elb.acl-type: 'white'                  # 白名单控制
```



```
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

表 7-63 ELB 访问控制注解

参数	类型	描述
kubernetes.io/elb.acl-id	String	• 不填写该参数时：表示CCE不对ELB侧访问控制进行修改。 • 参数值填写为空值时：表示允许所有IP访问。 • 参数值填写为ELB的IP地址组ID时：表示开启访问控制，为ELB设置IP地址黑名单或白名单。此时需同时填写kubernetes.io/elb.acl-status和kubernetes.io/elb.acl-type参数。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“网络 > 弹性负载均衡ELB”，在网络控制台中单击“弹性负载均衡 > IP地址组”，复制目标IP地址组的“ID”即可。详情请参见**IP地址组**。
kubernetes.io/elb.acl-status	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • on：表示开启访问控制。 • off：表示不开启访问控制。
kubernetes.io/elb.acl-type	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • black：表示黑名单，所选IP地址组无法访问ELB地址。 • white：表示白名单，仅所选IP地址组可以访问ELB地址。

7.3.4.11 为负载均衡类型的 Service 指定多个端口配置健康检查

LoadBalancer Service的健康检查相关注解字段由"kubernetes.io/elb.health-check-option"升级为"kubernetes.io/elb.health-check-options"，支持Service每个端口单独配置，且可以只配置部分端口。如无需单独配置端口协议，原有注解字段依旧可用无需修改。

约束与限制

- 该特性从以下版本开始支持：
  - v1.19集群：v1.19.16-r5及以上版本
  - v1.21集群：v1.21.8-r0及以上版本
  - v1.23集群：v1.23.6-r0及以上版本
  - v1.25集群：v1.25.2-r0及以上版本
  - v1.25以上版本集群
- 不允许同时配置 "kubernetes.io/elb.health-check-option" 和 "kubernetes.io/elb.health-check-options"。
- target_service_port字段必须配置，且不能重复。
- TCP端口只能配置健康检查协议为TCP、HTTP，UDP端口必须配置健康检查协议为UDP。

操作步骤

使用"kubernetes.io/elb.health-check-options"注解的示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
    version: v1
  annotations:
    kubernetes.io/elb.class: union          # 负载均衡类型
    kubernetes.io/elb.id: <your_elb_id>    # ELB ID，替换为实际值
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN  # 负载均衡器算法
    kubernetes.io/elb.health-check-flag: 'on'    # 开启ELB健康检查功能
    kubernetes.io/elb.health-check-options: '[
    {
        "protocol": "TCP",
        "delay": "5",
        "timeout": "10",
        "max_retries": "3",
        "target_service_port": "TCP:1",
        "monitor_port": "22"
    },
    {
        "protocol": "HTTP",
        "delay": "5",
        "timeout": "10",
        "max_retries": "3",
        "path": "/",
        "target_service_port": "TCP:2",
        "monitor_port": "22",
                "expected_codes": "200-399,401,404"
    }
    ]'
spec:
  selector:
    app: nginx
    version: v1
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 1
      nodePort: 0
      port: 1
      protocol: TCP
    - name: cce-service-1
```



```
    targetPort: 2
    nodePort: 0
    port: 2
    protocol: TCP
type: LoadBalancer
loadBalancerIP: **.**.**.**
```

表 7-64 elb.health-check-options 字段数据结构说明

参数	是否必填	参数类型	描述
target_service_port	是	String	spec.ports添加健康检查的目标端口，由协议、端口号组成，如：TCP:80
monitor_port	否	String	重新指定的健康检查端口，不指定时默认使用业务端口。 **说明** 请确保该端口在Pod所在节点已被监听，否则会影响健康检查结果。
delay	否	String	健康检查间隔（秒） 默认值：5，取值范围：1-50
timeout	否	String	健康检查的超时时间（秒） 默认值：10，取值范围1-50
max_retries	否	String	健康检查的最大重试次数 默认值：3，取值范围1-10
protocol	否	String	健康检查的协议 默认值：取关联服务的协议 取值范围：“TCP”、“UDP”或者“HTTP”
path	否	String	健康检查的URL，协议是“HTTP”时需要配置 默认值：“/” 取值范围：1-80字符

7.3.4.12 为负载均衡类型的 Service 配置 pass-through 能力

操作场景

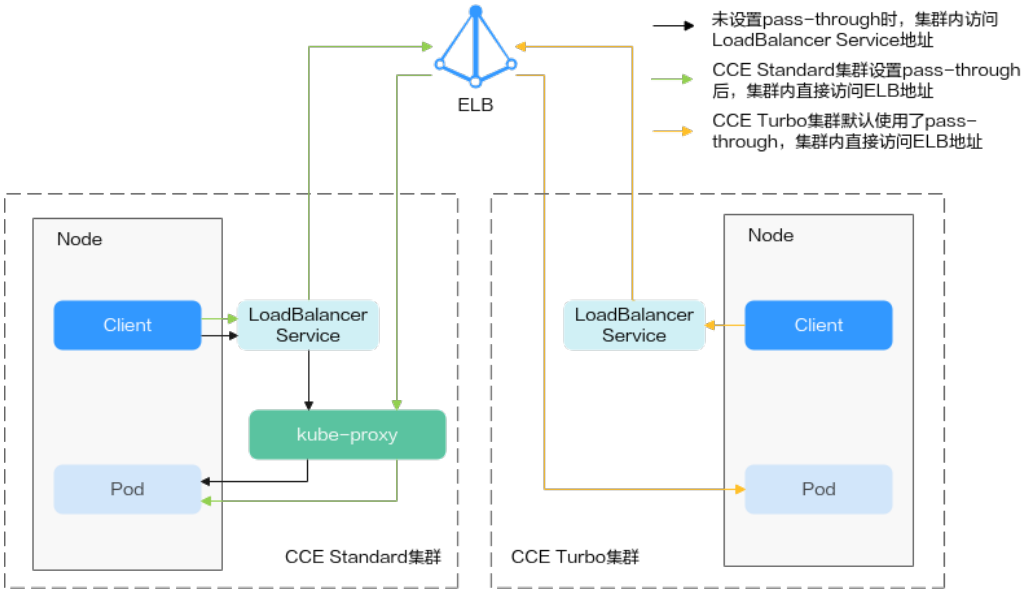
对于负载均衡类型的Service，负责集群内部流量转发的kube-proxy组件默认会将Service绑定的ELB IP地址配置到节点本地的转发规则中，从集群内部访问ELB的地址时，流量就会直接在集群内部转发，而不会经过ELB转发。

如果Service设置了服务亲和为节点级别，即externalTrafficPolicy取值为Local，Service将只会把流量转发给本节点上的Pod。从集群内部（节点上或容器中）访问Pod时，如果客户端所在节点正好没有相应的后端服务Pod，可能会出现访问不通的情况。

解决方案

CCE服务支持pass-through能力，在负载均衡类型的Service中配置kubernetes.io/elb.pass-through的annotation，可以实现集群内部访问Service的ELB地址时绕出集群，并通过ELB的转发最终转发到后端的Pod。

图 7-50 pass-through 访问示例



- 对于CCE集群：

  集群内部客户端访问LB类型Service时，访问请求默认是通过集群服务转发规则（iptables或IPVS）转发到后端的容器实例。

  当LB类型Service配置elb.pass-through后，集群内部客户端访问Service地址时会先访问到ELB，再通过ELB的负载均衡能力先访问到节点，然后通过集群服务转发规则（iptables或IPVS）转发到后端的容器实例。

- 对于CCE Turbo集群：

  集群内部客户端访问LB类型Service时，默认使用pass-through方式，此时客户端会直接访问ELB私网地址，然后通过ELB直接连接容器实例。

约束限制

- 在CCE Standard集群中，当使用独享型负载均衡配置pass-through后，从工作负载Pod所在节点或同节点的其他容器中访问ELB的私网IP地址，会出现无法访问的问题。
- 1.15及以下老版本集群暂不支持该能力。
- IPVS网络模式下，对接同一个ELB的Service需保持pass-through设置情况一致。
- 使用节点级别（Local）的服务亲和的场景下，会自动设置kubernetes.io/elb.pass-through为onlyLocal，开启pass-through能力。

操作步骤

本文以Nginx镜像创建无状态工作负载为例，创建一个具有pass-through的Service。

步骤1 使用kubectl命令行工具连接集群。详情请参见**通过kubectl连接集群**。

步骤2 使用Nginx镜像创建无状态工作负载。

创建nginx-deployment.yaml文件，内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: container-0
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
      imagePullSecrets:
      - name: default-secret
```

执行以下命令，部署工作负载。

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

步骤3 创建负载均衡类型的Service，并设置“kubernetes.io/elb.pass-through”为“true”。关于负载均衡类型的Service具体创建方法，请参见**自动创建负载均衡类型Service**。

创建nginx-elb-svc.yaml文件内容如下，本示例中自动创建了一个名为james的共享型ELB实例。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.pass-through: "true"
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_ty
pe":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Local
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

步骤4 执行以下命令创建服务。

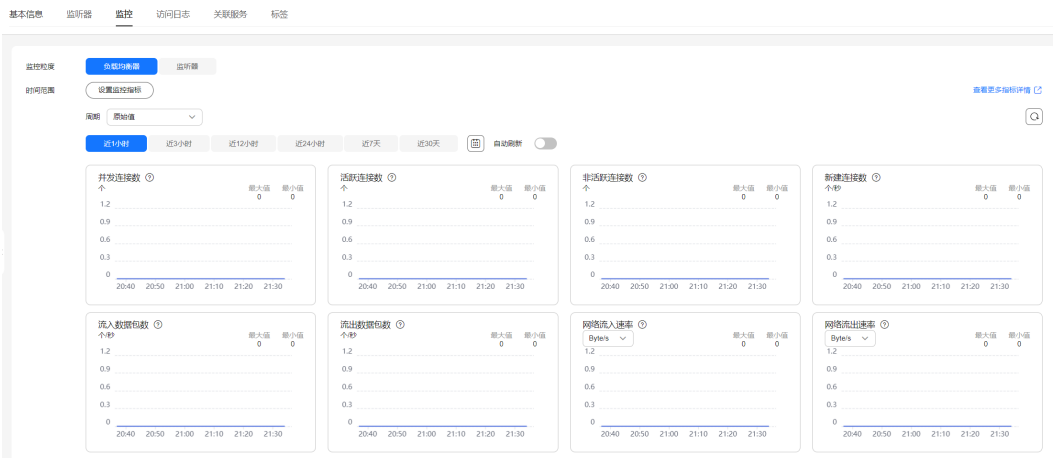

```
kubectl create -f nginx-elb-svc.yaml
```

----结束

配置验证

步骤1 登录ELB控制台，查看Service对应的ELB（本示例中名为james）。

步骤2 单击ELB名称，并切换至“监控”，可以看到ELB的连接数为0。



步骤3 使用kubectl命令行登录集群中的任意一个Nginx容器中，然后访问ELB的地址。

1. 查询集群中的Nginx容器。

   ```
   kubectl get pod
   ```

   回显如下：

   ```
   NAME                     READY   STATUS    RESTARTS   AGE
   nginx-7c4c5cc6b5-vpncx   1/1     Running   0          9m47s
   nginx-7c4c5cc6b5-xj5wl   1/1     Running   0          9m47s
   ```

2. 登录任意一个Nginx容器。

   ```
   kubectl exec -it nginx-7c4c5cc6b5-vpncx -- /bin/sh
   ```

3. 访问ELB地址。

   ```
   curl **.**.**.**
   ```

步骤4 稍微等待一段时间，查看ELB控制台的监控数据。

如果ELB出现新建访问连接，说明本次访问经过ELB转发，与预期一致。

----结束

7.3.4.13 为负载均衡类型的 Service 配置获取客户端 IP

使用共享型ELB创建负载均衡类型的服务时，您可以通过配置annotation配置ELB的监听器获取客户端IP的能力。

说明

- 使用独享型ELB时默认开启获取客户端IP，无需配置。
- 配置获取客户端IP后，如果您在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：

  - v1.23集群：v1.23.17-r0及以上版本
  - v1.25集群：v1.25.12-r0及以上版本
  - v1.27集群：v1.27.9-r0及以上版本
  - v1.28集群：v1.28.7-r0及以上版本
  - v1.29集群：v1.29.3-r0及以上版本
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

在CCE Standard集群中，当Service服务亲和类型配置成节点级别（即externalTrafficPolicy配置为Local）时该配置生效，配置集群级别（即externalTrafficPolicy配置为Cluster）时不生效。

在CCE Turbo集群中，Service服务亲和类型不支持配置成节点级别（即externalTrafficPolicy不支持配置为Local），该配置无法生效。

通过 kubectl 命令行创建

以关联已有ELB为例，YAML文件配置如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: <your_elb_id>            # ELB ID，替换为实际值
    kubernetes.io/elb.class: union                 # 负载均衡器类型
    kubernetes.io/elb.transparent-client-ip: 'true'  # 开启获取客户端源IP
spec:
  selector:
    app: nginx
  externalTrafficPolicy: Local
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  type: LoadBalancer
```

表 7-65 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.transparent-client-ip	String	仅TCP/UDP协议的服务支持配置。 ● true：表示开启客户端源IP能力。 ● false：表示关闭客户端源IP能力。

7.3.4.14 为负载均衡类型的 Service 配置自定义 EIP

通过CCE自动创建的带有EIP的ELB，可以通过添加Service的annotation（kubernetes.io/elb.custom-eip-id）完成ELB的EIP的自定义配置。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r0及以上
  - v1.25集群：v1.25.13-r0及以上
  - v1.27集群：v1.27.10-r0及以上
  - v1.28集群：v1.28.8-r0及以上
  - v1.29集群：v1.29.4-r0及以上
  - v1.30集群：v1.30.1-r0及以上
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 自定义EIP仅支持Service更新场景下配置，且Service的annotation中包含kubernetes.io/elb.eip-id。
- 自定义的EIP必须是未绑定状态。
- 配置自定义EIP后，如果ELB上的已有EIP是由CCE创建ELB时自动创建的且未被其他资源使用时，删除Service时会自动将EIP删除；如果ELB上的已有EIP是由您手动创建，删除Service时仅解绑EIP，您需要手动删除原先的EIP。

通过 kubectl 命令行创建

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 在创建Service时自动创建一个使用EIP的ELB，详情请参见**通过kubectl命令行创建-自动创建ELB**。

以使用独享型ELB的Service场景为例，查看该Service的YAML配置如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.autocreate:
'{"type":"public","bandwidth_name":"aaaaa","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_g-vm","name":"xxx","available_zone":["xxx"],"elb_virsubnet_ids":["fc0c61cd-c987-49c4-99a4-b7d816b57581"],"l7_flavor_name":"","l4_flavor_name":"L4_flavor.elb.pro.max","vip_subnet_cidr_id":"cf35b03f-c6ca-4f75-aa70-e2166cb1f800"}'
    kubernetes.io/elb.eip-id: 8560972c-2cc5-4699-94d6-e46f146eb73d     # 表示创建ELB时自动创建的EIP的ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: 0e78a84a-7deb-4747-aeb6-09b6a820b001
  labels:
    app: test-svc
    version: v1
  name: test-eip
  namespace: default
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.247.93.235
  clusterIPs:
  - 10.247.93.235
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  loadBalancerIP: *.*.*.*
  ports:
```



```
  - name: cce-service-0
    nodePort: 31354
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: test-svc
    version: v1
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: *.*.*.*
    - ip: 192.168.0.15
```

步骤3 修改该Service配置，添加annotation（kubernetes.io/elb.custom-eip-id）。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.autocreate:
'{"type":"public","bandwidth_name":"aaaaa","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"ban
dwidth_sharetype":"PER","eip_type":"5_g-vm","name":"xxx","available_zone":["xxx"],"elb_virsubnet_ids":
["fc0c61cd-c987-49c4-99a4-
b7d816b57581"],"l7_flavor_name":"","l4_flavor_name":"L4_flavor.elb.pro.max","vip_subnet_cidr_id":"cf35b03f
-c6ca-4f75-aa70-e2166cb1f800"}'
    kubernetes.io/elb.eip-id: 8560972c-2cc5-4699-94d6-e46f146eb73d     # 表示创建ELB时自动创建的EIP的ID
    kubernetes.io/elb.custom-eip-id: 88c197a1-cb85-4b38-b672-1d60dc5d00db  # 自定义的EIP的ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: 0e78a84a-7deb-4747-aeb6-09b6a820b001
  labels:
    app: test-svc
    version: v1
  name: test-eip
  namespace: default
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.247.93.235
  clusterIPs:
  - 10.247.93.235
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  loadBalancerIP: *.*.*.*
  ports:
  - name: cce-service-0
    nodePort: 31354
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: test-svc
    version: v1
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: *.*.*.*
    - ip: 192.168.0.15
```


表 7-66 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.custom-eip-id	String	自定义EIP的ID，您可以前往EIP控制台查看。该EIP必须是处于可绑定状态。

步骤4 Service更新成功后，重新查看Service。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.autocreate:
'{"type":"public","bandwidth_name":"aaaaa","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_g-vm","name":"xxx","available_zone":["xxx"],"elb_virsubnet_ids":["fc0c61cd-c987-49c4-99a4-b7d816b57581"],"l7_flavor_name":"","l4_flavor_name":"L4_flavor.elb.pro.max","vip_subnet_cidr_id":"cf35b03f-c6ca-4f75-aa70-e2166cb1f800"}'
    kubernetes.io/elb.eip-id: 8560972c-2cc5-4699-94d6-e46f146eb73d     # 表示创建ELB时自动创建的EIP的ID
    kubernetes.io/elb.custom-eip-id: 88c197a1-cb85-4b38-b672-1d60dc5d00db  # 自定义的EIP的ID
    kubernetes.io/elb.custom-eip-status: '{"id":"88c197a1-cb85-4b38-b672-1d60dc5d00db","public_ip_address":"2.2.2.2"}' # 自定义的EIP配置成功后，记录了配置的EIP的ID和IP地址
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: 0e78a84a-7deb-4747-aeb6-09b6a820b001
  labels:
    app: test-svc
    version: v1
  name: test-eip
  namespace: default
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.247.93.235
  clusterIPs:
  - 10.247.93.235
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  loadBalancerIP: 2.2.2.2
  ports:
  - name: cce-service-0
    nodePort: 31354
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: test-svc
    version: v1
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 2.2.2.2
    - ip: 192.168.0.15
```

----结束

7.3.4.15 为负载均衡类型的 Service 配置区间端口监听

在创建负载均衡类型的Service时，您可以为关联的ELB监听器指定需要监听的端口范围，使ELB监听器可以同时监听指定端口号范围内的端口，将这些端口收到的请求都转发到对应的后端服务。

说明

该功能依赖ELB能力，使用该功能前请确认当前区域是否支持。ELB已发布区域请参见**四层协议全端口监听和转发**。

前提条件

- 已创建Kubernetes集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r0及以上
  - v1.25集群：v1.25.13-r0及以上
  - v1.27集群：v1.27.10-r0及以上
  - v1.28集群：v1.28.8-r0及以上
  - v1.29集群：v1.29.4-r0及以上
  - v1.30集群：v1.30.1-r0及以上
- 如果您需要通过命令行创建，请使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

注意事项

仅使用独享型ELB且选择TCP/UDP/TLS协议时，支持配置区间端口监听。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“服务”页签，单击右上角“创建服务”。

步骤3 设置Service参数。本示例中仅列举必选参数，其余参数可根据需求参考**创建LoadBalancer类型Service**进行设置。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“负载均衡”。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“确认添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。
  - 类型：本例中仅支持选择“独享型”。
  - 创建方式：本文中以选择已有ELB为例进行说明，关于自动创建的配置参数请参见**表7-26**。
- **端口配置：**
  - 协议：支持选择TCP协议或UDP协议。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。例如nginx默认使用80端口。
  - 服务端口：Service使用的端口。本例中选择“区间端口监听”，可以同时监听指定端口号范围内的端口，将这些端口收到的请求都转发到对应的后端服务。端口范围取值为1-65535。您最多可添加10个互不重叠的监听端口段。
  - 监听器前端协议：当选择**独享型负载均衡器类型**时，需包含“网络型（TCP/UDP/TLS）”方可支持配置TLS协议。

图 7-51 配置区间端口监听

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**service-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi service-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: v1
kind: Service
metadata:
  name: service-test
  labels:
    app: test
    version: v1
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance  #需使用独享型ELB
    kubernetes.io/elb.id: <your_elb_id> #替换为您已有的ELB ID
    kubernetes.io/elb.port-ranges: '{"cce-service-0":["100,200", "300,400"], "cce-service-1":["500,600", "700,800"]}' #配置区间端口监听
spec:
  selector:
    app: test
    version: v1
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80          #替换为您的容器端口
      nodePort: 0
      port: 100       #配置区间端口监听时，该参数值不生效，但不可缺少或重复，默认取值为区间起始端口
      protocol: TCP
    - name: cce-service-1
      targetPort: 81          #替换为您的容器端口
      nodePort: 0
      port: 500       #配置区间端口监听时，该参数值不生效，但不可缺少或重复
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <your_elb_ip> #替换为您已有的ELB私有IP
```


表 7-67 区间端口监听参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.port-ranges	是	String	使用独享型ELB且选择TCP/UDP/TLS协议时，支持创建某个端口范围的监听器，端口范围1~65535，您最多可为每个监听器添加10个互不重叠的监听端口段。 参数值格式如下，ports_name和port均不允许重复： '{"*<ports_name_1>*":["*<port_1>*,*<port_2>*","*<port_3>*,*<port_4>*"], "*<ports_name_2>*":["*<port_5>*,*<port_6>*","*<port_7>*,*<port_8>*"]}' 例如以下示例表示，端口配置名称为cce-service-0，其监听端口范围为100~200和300~400；端口配置名称为cce-service-1，其监听端口范围为500~600和700~800。 '{"cce-service-0":["100,200", "300,400"], "cce-service-1":["500,600", "700,800"]}'

步骤3 创建Service。

```
kubectl create -f service-test.yaml
```

回显如下，表示Service已创建。

```
service/service-test created
```

----结束

7.3.4.16 通过 ELB 健康检查设置 Pod 就绪状态

Pod的就绪状态可与挂载到ELB后端的健康检查联动，在健康检查成功后，将Pod置为就绪。与Pod的strategy.rollingUpdate.maxSurge和strategy.rollingUpdate.maxUnavailable参数配合，可实现负载的优雅滚动升级。

约束与限制

- 该特性从以下版本开始支持：
  - v1.19集群：v1.19.16-r5及以上版本
  - v1.21集群：v1.21.8-r0及以上版本
  - v1.23集群：v1.23.6-r0及以上版本
  - v1.25集群：v1.25.2-r0及以上版本
  - v1.25以上版本集群
- 该功能只支持直通场景，即CCE Turbo集群中使用独享型ELB的场景。
- 该功能需在Pod中配置特定的readinessGates字段，指定标签target-health.elb.k8s.cce/{serviceName}，其中{serviceName}为服务名称。
- Pod就绪状态只在最初对接ELB后端时生效，后续健康检查状态不再影响Pod就绪状态。

通过 ELB 健康检查设置 Pod 就绪状态

使用Pod ReadinessGates方式如下：

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右上角单击“YAML创建”。

YAML内容如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
  labels:
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      containers:
        - name: container-1
          image: nginx:latest
      imagePullSecrets:
        - name: default-secret
      readinessGates:
        - conditionType: target-health.elb.k8s.cce/specific-service-name  # 指定ServiceName。
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%          # 配合以下两个参数，可以控制ELB后端个数，实现优雅滚动升级。
      maxSurge: 25%
```

步骤3 单击“确定”创建工作负载后，查看工作负载状态，Pod处于未就绪状态。

事件提示如下：

```
实例未就绪：correspondingcondition of pod readinessgate "target-health.elb.k8s.cce/specific-service-name" does not exist.
```

步骤4 在左侧导航栏中选择“服务”，在右上角单击“创建服务”，并进行以下配置。

- Service名称：需要与Pod中readinessGates字段设置的名称一致。
- 访问类型：选择负载均衡型Service。
- 选择器：单击“引用负载标签”，选择上一步中创建的工作负载并单击“确定”。
- 负载均衡器：必须使用独享型ELB，您可以选择已有的ELB或自动创建新的ELB。
- 健康检查：开启健康检查（不开启则默认为健康检查成功）。

图 7-52 负载均衡配置

步骤5 前往ELB控制台，查看对应的后端服务器组，健康检查状态正常。

步骤6 在CCE控制台中查看工作负载状态处于“运行中”。

----结束

7.3.4.17 健康检查使用 UDP 协议的安全组规则说明

操作场景

当负载均衡协议为UDP时，健康检查也采用的UDP协议，您需要打开其后端服务器的ICMP协议安全组规则。关于使用UDP协议健康检查的详细说明，请参见**使用UDP协议有什么注意事项？**。

操作步骤

步骤1 登录CCE控制台，单击服务列表中的“网络 > 虚拟私有云 VPC”，在网络控制台单击“访问控制 > 安全组”。

步骤2 在界面右侧的安全组列表中找到集群的安全组。单击“入方向规则”页签，单击“添加规则”，添加入方向规则如下。

集群类型	ELB类型	放通安全组	协议端口	放通源地址网段
CCE Standard	共享型ELB	节点安全组，名称规则默认是**{集群名}-cce-node-{随机ID}** 如果集群中绑定了自定义的节点安全组，请根据实际进行选择。	ICMP的全部端口	共享型ELB网段100.125.0.0/16
	独享型ELB	节点安全组，名称规则默认是**{集群名}-cce-node-{随机ID}** 如果集群中绑定了自定义的节点安全组，请根据实际进行选择。	ICMP的全部端口	ELB后端子网网段
CCE Turbo	共享型ELB	节点安全组，名称规则默认是**{集群名}-cce-node-{随机ID}** 如果集群中绑定了自定义的节点安全组，请根据实际进行选择。	ICMP的全部端口	共享型ELB网段100.125.0.0/16
	独享型ELB	ENI安全组，名称规则默认是**{集群名}-cce-eni-{随机ID}** 如果集群中绑定了自定义的容器安全组，请根据实际进行选择。	ICMP的全部端口	ELB后端子网网段

图 7-53 添加安全组规则

步骤3 单击“确定”。

----结束

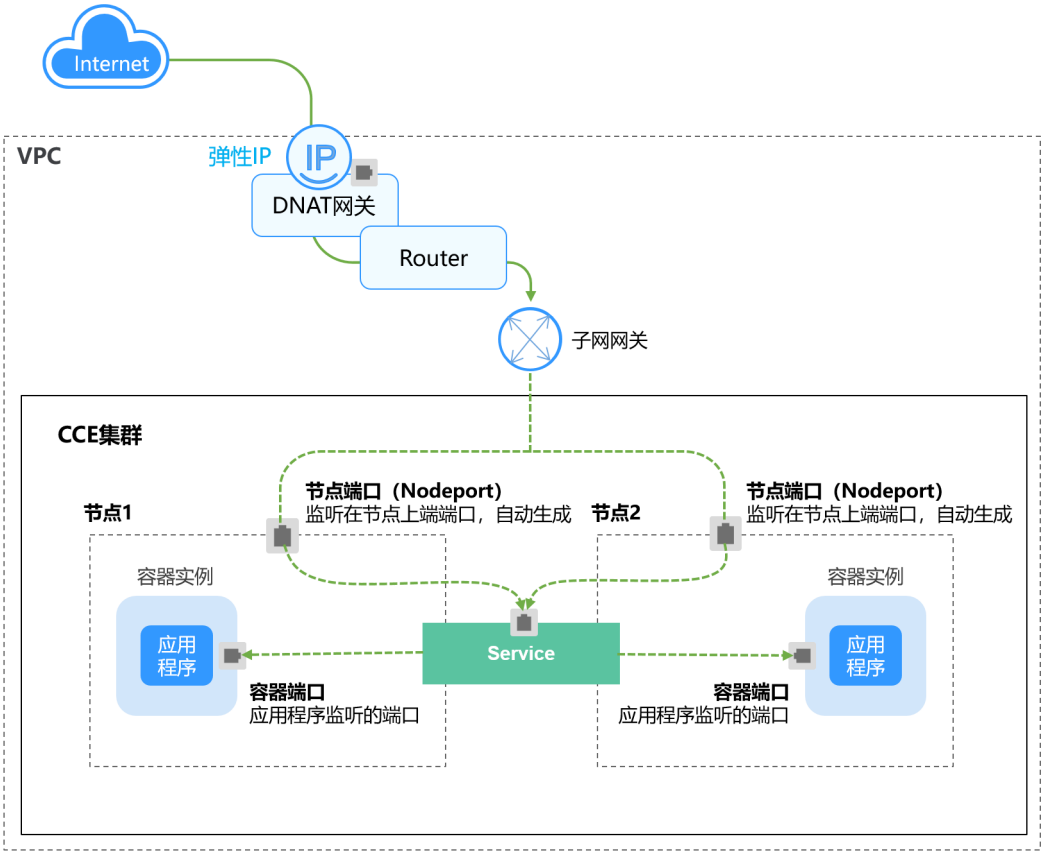
7.3.5 DNAT 网关（DNAT）

操作场景

“DNAT网关”可以为集群节点提供网络地址转换服务，使多个节点可以共享使用弹性IP。

NAT网关与弹性IP方式相比增强了可靠性，弹性IP无需与单个节点绑定，任何节点状态的异常不影响其访问。访问方式由公网弹性IP地址以及设置的访问端口组成，例如“10.117.117.117:80”。

图 7-54 DNAT 网关（DNAT）

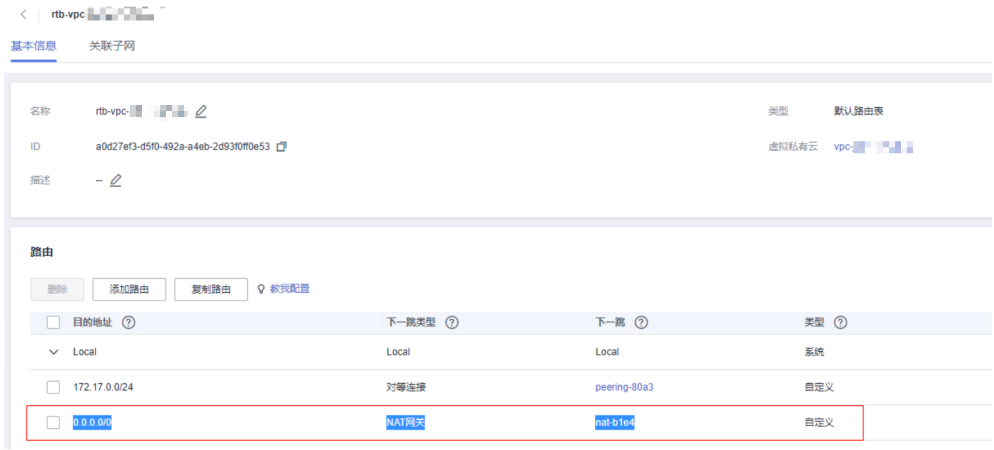


约束与限制

关于NAT网关的使用，您需要注意以下几点：

- DNAT规则不支持企业项目授权。
- 集群内容器不支持访问externalTrafficPolicy为Local模式的DNAT Service。
- 同一个NAT网关下的多条规则可以复用同一个弹性公网IP，不同网关下的规则必须使用不同的弹性公网IP。
- 每个VPC支持的NAT网关数为1。
- 用户不能在VPC下手动添加默认路由。
- VPC内的每个子网只能添加一条SNAT规则。

- SNAT规则和DNAT规则一般面向不同的业务，如果使用相同的EIP，会面临业务相互抢占问题，请尽量避免。SNAT规则不能和全端口的DNAT规则共用EIP。
- DNAT规则不支持将弹性公网IP绑定到虚拟IP。
- 当云主机同时配置弹性公网IP服务和NAT网关服务时，数据均通过弹性公网IP转发。
- SNAT规则中添加的自定义网段，对于虚拟私有云的配置，必须是虚拟私有云子网网段的子集，不能相等。
- SNAT规则中添加的自定义网段，对于云专线的配置，必须是云专线侧网段，且不能与虚拟私有云侧的网段冲突。
- 当执行云服务器底层资源操作（如变更规格）时，会导致已配置的NAT规则失效，需要删除后重新配置。
- 创建service后，如果服务亲和从集群级别切换为节点级别，连接跟踪表将不会被清理，建议用户创建service后不要修改服务亲和属性，如需修改请重新创建service。
- 当集群的节点子网关联了自定义路由表时，使用DNAT类型service同时需要将NAT的路由加入到自定义路由表中。



说明

其余关于NAT网关的产品限制，请参见**NAT网关约束与限制**。

创建 NAT 网关和弹性公网 IP

您需要提前创建NAT网关实例和弹性公网IP，具体操作步骤如下：

步骤1 登录管理控制台，在服务列表中选择“网络 > NAT网关”，单击页面右上角的“购买公网NAT网关”。根据实际业务需求填写相关内容。

说明

购买NAT网关，选择VPC和子网时，请确保与CCE中运行业务的集群VPC和子网一致。

步骤2 在管理控制台，在服务列表中选择“网络 > 弹性公网IP”，单击右上角的“购买弹性公网IP”。根据实际业务需求填写相关内容。

----结束

创建 DNAT 网关类型 Service

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，在右上角单击“创建服务”。

步骤3 设置集群内访问参数。

- **Service名称：**自定义服务名称，可与工作负载名称保持一致。
- **访问类型**：选择“DNAT网关”。
- **命名空间：**工作负载所在命名空间。
- **服务亲和：**详情请参见**服务亲和（externalTrafficPolicy）**。
  - 集群级别：集群下所有节点的IP+访问端口均可以访问到此服务关联的负载，服务访问会因路由跳转导致一定性能损失，且无法获取到客户端源IP。
  - 节点级别：只有通过负载所在节点的IP+访问端口才可以访问此服务关联的负载，服务访问没有因路由跳转导致的性能损失，且可以获取到客户端源IP。
- **选择器：**添加标签，Service根据标签选择Pod，填写后单击“添加”。也可以引用已有工作负载的标签，单击“引用负载标签”，在弹出的窗口中选择负载，然后单击“确定”。
- **IPv6：**默认不开启，开启后服务的集群内IP地址（ClusterIP）变为IPv6地址，具体请参见**如何通过CCE搭建IPv4/IPv6双栈集群？**。**该功能仅在1.15及以上版本的集群创建时开启了IPv6功能才会显示。**
- **DNAT网关：**选择**创建NAT网关和弹性公网IP**中创建的DNAT网关实例和弹性公网IP。
- **端口配置：**
  - 协议：请根据业务的协议类型选择。
  - 容器端口：工作负载程序实际监听的端口，需用户确定。nginx程序实际监听的端口为80。
  - 服务端口：容器端口映射到集群虚拟IP上的端口，用虚拟IP访问工作负载时使用，端口范围为1-65535，可任意指定。

步骤4 单击“确定”，创建Service。

----结束

通过 kubectl 命令行创建

您可以在创建工作负载时通过kubectl命令行设置Service访问方式。本节以nginx为例，说明kubectl命令实现集群内访问的方法。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建并编辑nginx-deployment.yaml以及nginx-nat-svc.yaml文件。

其中，nginx-deployment.yaml和nginx-nat-svc.yaml为自定义名称，您可以随意命名。

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
```



```
replicas: 1
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - image: nginx:latest
      name: nginx
    imagePullSecrets:
    - name: default-secret
```

以上字段的解释请参见表5-2。

vi nginx-nat-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.class: dnat
    kubernetes.io/natgateway.id: e4a1cfcf-29df-4ab8-a4ea-c05dc860f554
spec:
  loadBalancerIP: 10.78.42.242
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

表 7-68 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.class	是	String	该参数配置为DNAT用于对接NAT网关服务添加DNAT规则。
kubernetes.io/natgateway.id	是	String	用于指定NAT网关ID。
loadBalancerIP	是	String	公网弹性IP。
port	是	Integer	对应界面上的访问端口，取值范围为1 ~ 65535。
targetPort	是	String	对应界面上的容器端口，取值范围为1 ~ 65535。
type	是	String	NAT网关服务需要配置为LoadBalancer类型。

步骤3 创建工作负载。

kubectl create -f nginx-deployment.yaml

回显如下表示工作负载开始创建。


```
deployment "nginx" created
```

kubectl get po

回显如下，工作负载状态为Running，表示工作负载已运行中。

```
NAME                     READY     STATUS           RESTARTS   AGE
nginx-2601814895-sf71t   1/1       Running          0          8s
```

步骤4 创建服务。

kubectl create -f nginx-nat-svc.yaml

回显如下表示服务已创建成功。

```
service "nginx-eip" created
```

kubectl get svc

回显如下表示服务访问方式已设置成功。

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP   10.247.0.1       <none>         443/TCP        3d
nginx-nat    LoadBalancer 10.247.226.2   10.154.74.98   80:30589/TCP   5s
```

步骤5 在浏览器中输入访问地址，例如为**10.154.74.98**:80访问地址。

其中**10.154.74.98**为弹性IP地址，80为上一步中获取的节点端口号。

----结束

7.3.6 Headless Service

Service解决了Pod的内外部访问问题，但还有下面这些问题没解决。

- 同时访问所有Pod
- 一个Service内部的Pod互相访问

Headless Service正是解决这个问题的，Headless Service不会创建ClusterIP，并且查询会返回所有Pod的DNS记录，这样就可查询到所有Pod的IP地址。**有状态负载StatefulSet**正是使用Headless Service解决Pod间互相访问的问题。

```
apiVersion: v1
kind: Service       # 对象类型为Service
metadata:
  name: nginx-headless
  labels:
    app: nginx
spec:
  ports:
    - name: nginx     # Pod间通信的端口名称
      port: 80        # Pod间通信的端口号
  selector:
    app: nginx        # 选择标签为app:nginx的Pod
  clusterIP: None     # 必须设置为None，表示Headless Service
```

执行如下命令创建Headless Service。

```
# kubectl create -f headless.yaml
service/nginx-headless created
```

创建完成后可以查询Service。

```
# kubectl get svc
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
nginx-headless   ClusterIP   None         <none>        80/TCP     5s
```


创建一个Pod来查询DNS，可以看到能返回所有Pod的记录，这就解决了访问所有Pod的问题了。

```
$ kubectl run -i --tty --image tutum/dnsutils dnsutils --restart=Never --rm /bin/sh
If you don't see a command prompt, try pressing enter.
/ # nslookup nginx-0.nginx
Server:         10.247.3.10
Address:        10.247.3.10#53
Name:   nginx-0.nginx.default.svc.cluster.local
Address: 172.16.0.31

/ # nslookup nginx-1.nginx
Server:         10.247.3.10
Address:        10.247.3.10#53
Name:   nginx-1.nginx.default.svc.cluster.local
Address: 172.16.0.18

/ # nslookup nginx-2.nginx
Server:         10.247.3.10
Address:        10.247.3.10#53
Name:   nginx-2.nginx.default.svc.cluster.local
Address: 172.16.0.19
```

7.4 路由（Ingress）

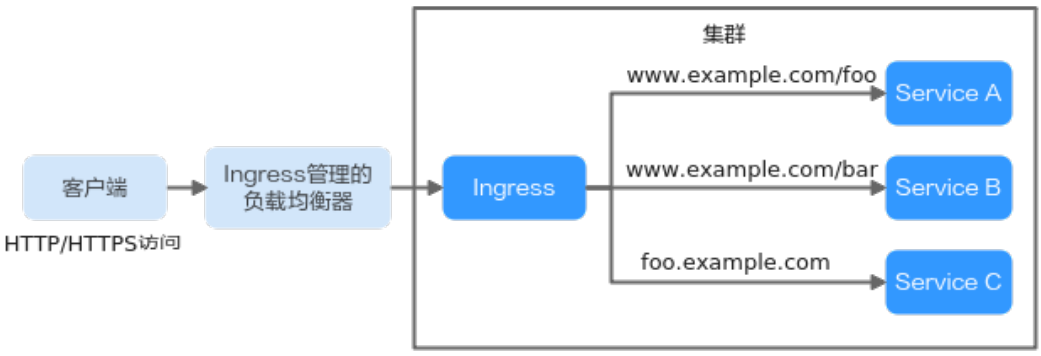
7.4.1 路由概述

为什么需要 Ingress

Service基于TCP和UDP协议进行访问转发，为集群提供了四层负载均衡的能力。但是在实际场景中，Service无法满足应用层中存在着大量的HTTP/HTTPS访问需求。因此，Kubernetes集群提供了另一种基于HTTP协议的访问方式——Ingress。

Ingress是Kubernetes集群中一种独立的资源，制定了集群外部访问流量的转发规则。如图7-55所示，用户可根据域名和路径对转发规则进行自定义，完成对访问流量的细粒度划分。

图 7-55 Ingress 示意图



Ingress 简介

在Kubernetes中，Ingress资源本身是一个抽象的概念，实际的流量处理是由Ingress Controller来完成的。

- **Ingress资源**：一组基于域名或路径把请求转发到指定Service实例的访问规则，是Kubernetes的一种资源对象，通过接口服务实现增、删、改、查的操作。
- **Ingress Controller**：请求转发的执行器，用以实时监控资源对象Ingress、Service、Endpoint、Secret（主要是TLS证书和Key）、Node、ConfigMap的变化，解析Ingress定义的规则并负责将请求转发到相应的后端Service。

  Ingress Controller在不同厂商之间的实现方式不同，CCE支持ELB型和Nginx型两种Ingress Controller类型：
  - ELB Ingress Controller部署在master节点，基于弹性负载均衡服务（ELB）实现流量转发，所有策略配置和转发行为均在ELB侧完成。
  - Nginx Ingress Controller使用Kubernetes社区维护的模板与镜像部署在集群内部，并通过NodePort对外提供访问，外部流量经过Nginx组件转发到集群内其他业务，流量转发行为及转发对象均在集群内部。

Ingress 特性对比

表 7-69 Ingress 特性对比

特性	ELB Ingress Controller	Nginx Ingress Controller
运维	免运维	自行安装、升级、维护
性能	一个Ingress支持一个ELB实例	多个Ingress只支持一个ELB实例
	使用企业级LB，高性能高可用，升级、故障等场景不影响业务转发	性能依赖pod的资源配置
	支持配置动态加载	• 非后端端点变更需要Reload进程，对长连接有损。 • 端点变更使用Lua实现热更新。 • Lua插件变更需要Reload进程。
组件部署	Master节点，不占用工作节点	Worker节点，需要Nginx组件运行成本
路由重定向	支持	支持
SSL配置	支持	支持
代理HTTPS协议的后端服务	支持	支持，可通过backend-protocol: "HTTPS"注解实现

由于ELB Ingress和社区开源的Nginx Ingress在原理上存在本质区别，因此支持的Service类型不同，详情请参见**ELB Ingress支持的Service类型**。

ELB Ingress Controller部署在master节点，所有策略配置和转发行为均在ELB侧完成。非ELB直通Pod场景下，集群外部的ELB只能通过VPC的IP对接集群内部节点，因此ELB

Ingress只支持NodePort类型的Service。但是ELB直通Pod场景（CCE Turbo集群 + 独享型ELB实例）下，ELB可直接将流量转发到集群内Pod，此时Ingress仅支持对接ClusterIP类型的Service。

Nginx Ingress Controller运行在集群中，作为服务通过NodePort对外暴露，流量经过Nginx-ingress转发到集群内其他业务，流量转发行为及转发对象均在集群内部，因此支持ClusterIP和NodePort类型的Service。

综上，ELB Ingress使用企业级LB进行流量转发，拥有高性能和高稳定性的优点，而Nginx Ingress Controller部署在集群节点上，牺牲了一定的集群资源但可配置性相对更好。

ELB Ingress Controller 工作原理

CCE自研的ELB Ingress Controller基于弹性负载均衡服务ELB实现公网和内网（同一VPC内）的七层网络访问，通过不同的路径将访问流量分发到对应的服务。

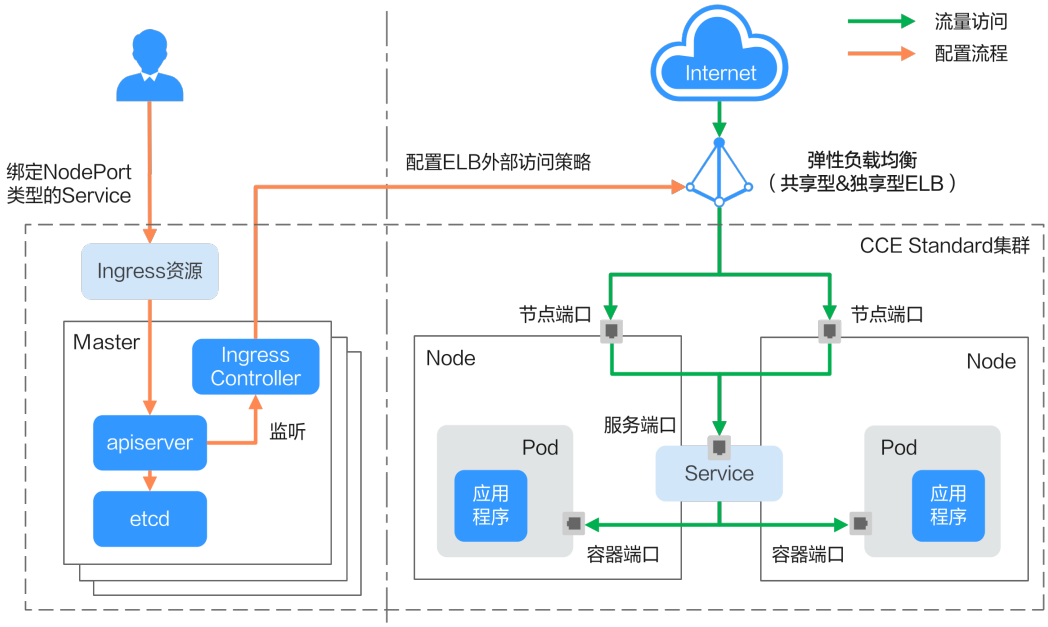
ELB Ingress Controller部署于Master节点上，与集群所在VPC下的弹性负载均衡器绑定，支持在同一个ELB实例（同一IP）下进行不同域名、端口和转发策略的设置。ELB Ingress Controller的工作原理如下：

1. 用户创建Ingress资源，在Ingress中配置流量访问规则，包括负载均衡器、访问路径、SSL以及访问的后端Service端口等。
2. Ingress Controller监听到Ingress资源发生变化时，就会根据其中定义的流量访问规则，在ELB侧重新配置监听器以及后端服务器路由。
3. 当用户进行访问时，流量根据ELB中配置的转发策略转发到关联的各个工作负载。

ELB Ingress Controller的工作原理和集群类型及ELB类型有关，不同场景的配置流程及网络流向如下所示：

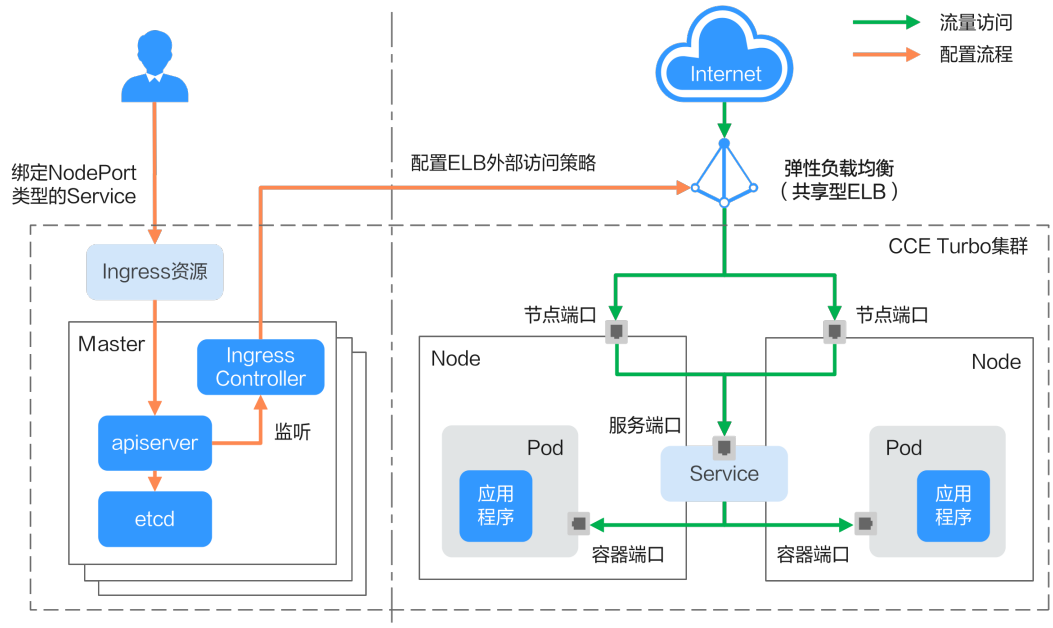
CCE Standard 集群场景

图 7-56 ELB Ingress 工作原理（CCE Standard 集群场景）



CCE Turbo 集群使用共享型 ELB 场景

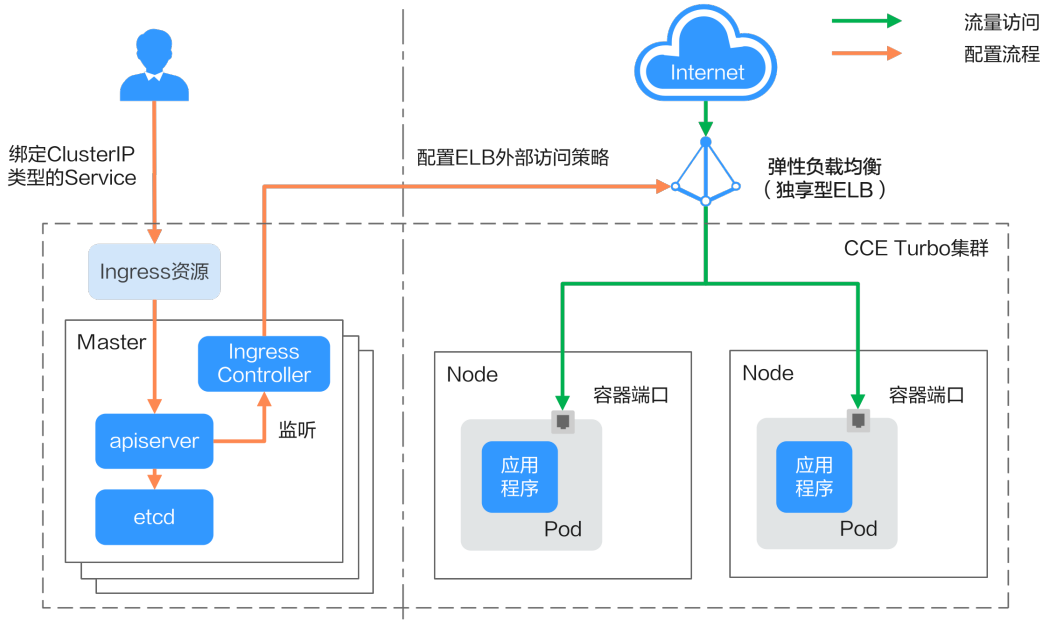
图 7-57 ELB Ingress 工作原理（CCE Turbo 集群使用共享型 ELB 场景）



CCE Turbo 集群使用独享型 ELB 场景

在使用**CCE Turbo集群**时，Pod IP直接从VPC中分配，支持使用**独享型ELB**直接连接Pod。创建集群外部访问的Ingress时，可使用ELB对接ClusterIP服务，直接将Pod作为ELB监听器的后端服务器，外部流量可以不经过节点端口转发而直接访问集群中的Pod。

图 7-58 ELB Ingress 直通容器原理（CCE Turbo 集群使用独享型 ELB 场景）



Nginx Ingress Controller 工作原理

Nginx型的Ingress使用弹性负载均衡（ELB）作为流量入口，并在集群中部署**NGINX Ingress控制器**来对流量进行负载均衡及访问控制。

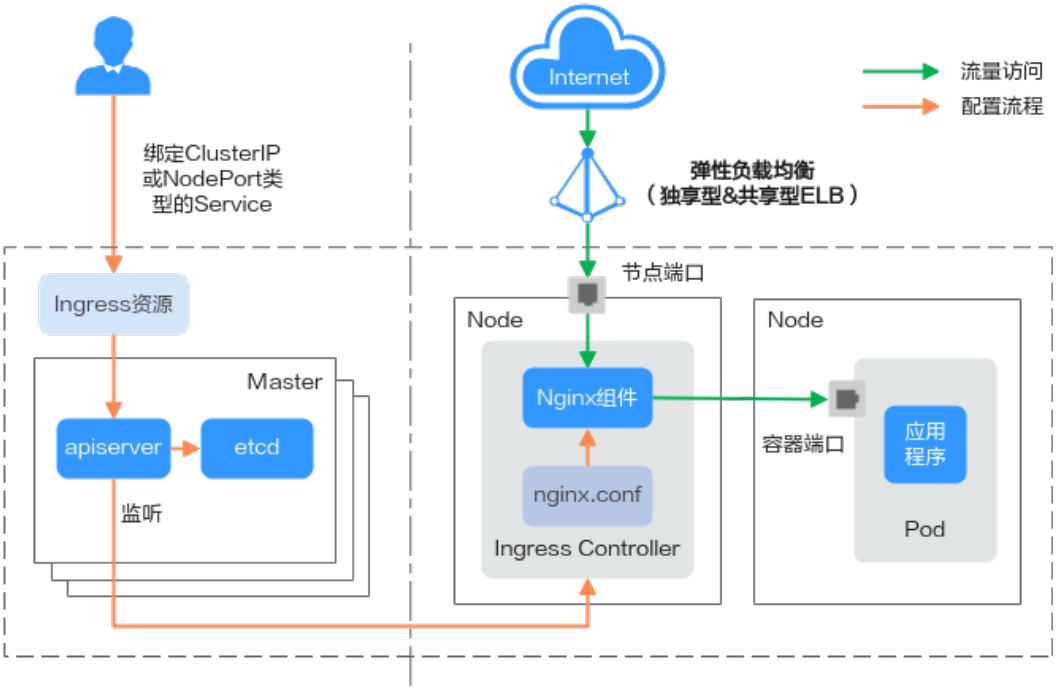
说明

NGINX Ingress控制器插件使用**开源社区**的模板与镜像，使用过程中可能存在缺陷，CCE会定期同步社区版本来修复已知漏洞。请评估是否满足您的业务场景要求。

Nginx型的Ingress Controller通过pod部署在工作节点上，因此引入了相应的运维成本和Nginx组件运行成本，其工作原理如**图7-59**，实现步骤如下：

1. 当用户更新Ingress资源后，Ingress Controller就会将其中定义的转发规则写入到Nginx的配置文件（nginx.conf）中。
2. 内置的Nginx组件进行reload，加载更新后的配置文件，完成Nginx转发规则的修改和更新。
3. 在流量访问集群时，首先被已创建的负载均衡实例转发到集群内部的Nginx组件，然后Nginx组件再根据转发规则将其转发至对应的各个工作负载。

图 7-59 Nginx Ingress Controller 工作原理



Ingress 支持的 Service 类型

由于ELB Ingress和社区开源的Nginx Ingress的实现原理不同，因此支持的Service类型不同。

ELB Ingress 支持的 Service 类型

表 7-70 ELB Ingress 支持的 Service 类型

集群类型	ELB类型	集群内访问（ClusterIP）	节点访问（NodePort）
CCE Standard集群	共享型负载均衡	不支持	支持
	独享型负载均衡	不支持（集群内访问服务关联实例未绑定eni网卡，独享型负载均衡无法对接）	支持
CCE Turbo集群	共享型负载均衡	不支持	支持
	独享型负载均衡	支持	不支持（节点访问服务关联实例已绑定eni网卡，独享型负载均衡无法对接）

Nginx Ingress 支持的 Service 类型

表 7-71 Nginx Ingress 支持的 Service 类型

集群类型	ELB类型	集群内访问（ClusterIP）	节点访问（NodePort）
CCE Standard集群	共享型负载均衡	支持	支持
	独享型负载均衡	支持	支持
CCE Turbo集群	共享型负载均衡	支持	支持
	独享型负载均衡	支持	支持

7.4.2 ELB Ingress 管理

7.4.2.1 通过控制台创建 ELB Ingress

Ingress是Kubernetes中的一种资源对象，用来管理集群外部访问集群内部服务的方式。您可以通过Ingress资源来配置不同的转发规则，从而根据转发规则访问集群内Pod。本文以**Nginx工作负载**为例，为您介绍如何使用控制台创建ELB Ingress。

前提条件

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。

- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

- 建议其他资源不要使用Ingress自动创建的ELB实例，否则在删除Ingress时，ELB实例会被占用，导致资源残留。
- 添加Ingress后请在CCE页面对所选ELB实例进行配置升级和维护，不可在ELB页面对配置进行更改，否则可能导致Ingress服务异常。
- Ingress转发策略中注册的URL需与后端应用提供访问的URL一致，否则将返回404错误。
- IPVS模式集群下，Ingress和Service使用相同ELB实例时，无法在集群内的节点和容器中访问Ingress，因为kube-proxy会在ipvs-0的网桥上挂载LB类型的Service地址，Ingress对接的ELB的流量会被ipvs-0网桥劫持。建议Ingress和Service使用不同ELB实例。
- 请勿将Ingress与**使用HTTP的Service**对接同一个ELB下的同一个监听器，否则将产生端口冲突。
- 独享型ELB规格必须支持应用型（HTTP/HTTPS），且网络类型必须支持私网（有私有IP地址）。
- 同集群使用多个Ingress对接同一个ELB端口时，监听器的配置项（例如监听器关联的证书、监听器HTTP2属性等）均以首次创建监听器的Ingress配置为准。

添加 ELB Ingress

本节以nginx作为工作负载并添加ELB Ingress为例进行说明。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

- **名称：**自定义Ingress名称，例如ingress-demo。
- **对接Nginx：**此选项只有在安装了**NGINX Ingress控制器**插件后才会显示。如显示了“对接Nginx”，则说明您安装了NGINX Ingress控制器插件，创建ELB Ingress时不能打开该项开关，如果打开则是使用Nginx Ingress Controller，具体请参见**通过控制台创建Nginx Ingress**。
- **负载均衡器：**选择弹性负载均衡的类型、创建方式。

  ELB类型可选择“独享型”或“共享型”。独享型ELB规格需要支持应用型（HTTP/HTTPS），且网络类型必须支持私网。

  创建方式可选择“选择已有”或“自动创建”。不同创建方式的配置详情请参见**表7-72**。

  表 7-72 ELB 配置

创建方式	配置
选择已有	仅支持选择与集群在同一个VPC下的ELB实例。如果没有可选的ELB实例，请单击“创建负载均衡器”跳转到ELB控制台创建。

创建方式	配置
自动创建	- 实例名称：请填写ELB名称。 - 企业项目：该参数仅对开通企业项目的企业客户账号显示。企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 - 可用区（仅独享型ELB支持）：可以选择在多个可用区创建负载均衡实例，提高服务的可用性。如果业务需要考虑容灾能力，建议选择多个可用区。 - 前端子网（仅独享型ELB支持）：用于分配ELB实例对外服务的IP地址。 - 后端子网（仅独享型ELB支持）：用于与后端服务建立连接的IP地址。 - 网络型规格/应用型规格/规格（仅独享型ELB支持）： ■ 固定规格：适用于业务用量较为稳定的场景，按固定规格折算收取每小时使用的容量费用。 - 弹性公网IP：选择“自动创建”时，可配置公网带宽的计费方式及带宽大小。 - 资源标签：通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。

- **监听器配置：**Ingress为负载均衡器配置监听器，监听器对负载均衡器上的请求进行监听，并分发流量。配置完成后ELB实例侧将会创建对应的监听器，名称默认为k8s_<协议类型>_<端口号>，例如“k8s_HTTP_80”。
  - 前端协议：支持HTTP和HTTPS。
  - 对外端口：开放在负载均衡服务地址的端口，可任意指定。
  - 访问控制：
    - 继承ELB已有配置：CCE不对ELB侧已有的访问控制进行修改。
    - 允许所有IP访问：不设置访问控制。
    - 白名单：仅所选IP地址组可以访问ELB地址。
    - 黑名单：所选IP地址组无法访问ELB地址。
  - 证书来源：支持TLS密钥和ELB服务器证书。
    - TLS密钥：创建密钥证书的方法请参见**创建密钥**。
    - ELB服务器证书：使用在ELB服务中创建的证书。
  - 服务器证书：负载均衡器创建HTTPS协议监听时需要绑定证书，以支持HTTPS数据传输加密认证。

    说明

    同一个ELB实例的同一个端口配置HTTPS时，一个监听器只支持配置一个密钥证书。若使用两个不同的密钥证书将两个Ingress添加到同一个ELB下的同一个监听器，ELB侧实际只生效最先添加的证书。

- SNI：SNI（Server Name Indication）是TLS的扩展协议，在该协议下允许同一个IP地址和端口号下对外提供多个基于TLS的访问域名，且不同的域名可以使用不同的安全证书。开启SNI后，允许客户端在发起TLS握手请求时就提交请求的域名信息。负载均衡收到TLS请求后，会根据请求的域名去查找证书：若找到域名对应的证书，则返回该证书认证鉴权；否则，返回缺省证书（服务器证书）认证鉴权。

  说明

  - 当选择HTTPS协议时，才支持配置“SNI”选项。
  - 该功能仅支持1.15.11及以上版本的集群。
  - 用于SNI的证书需要指定域名，每个证书只能指定一个域名。支持泛域名证书。
  - 对接到同一个ELB端口的ingress，请勿配置域名相同但证书不同的SNI，否则将会被覆盖。

- 安全策略：安全策略包含HTTPS可选的TLS协议版本和配套的加密算法套件。

  关于安全策略的详细说明，请参见**安全策略**。

  说明

  - 选择HTTPS协议时，才支持配置“安全策略”选项。
  - 该功能仅支持1.17.9及以上版本的集群。

- 后端协议：

  当**监听器配置**为HTTP协议时，仅可选择“HTTP”。

  当**监听器配置**为HTTPS协议时，可选择“HTTP”、“HTTPS”或“GRPC”。仅独享型ELB支持选择GRPC协议，且需开启HTTP/2，系统将自动为您添加**kubernetes.io/elb.http2-enable:true**注解。当前GRPC功能仅在部分区域开放，请以控制台呈现为准。

- 高级配置：

配置	说明	使用限制
获取监听器端口号	开启后可以将ELB实例的监听端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例支持配置。
获取客户端请求端口号	开启后可以将客户端的源端口从报文的HTTP头中带到后端云服务器。	独享型ELB实例支持配置。
重写X-Forwarded-Host	开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。	独享型ELB实例支持配置。

配置	说明	使用限制
数据压缩	开启将对特定文件类型进行压缩；关闭则不会对任何文件类型进行压缩。 ■ Brotli支持压缩所有类型。 ■ Gzip支持压缩的类型如下：text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。	独享型ELB实例支持配置。
空闲超时时间	客户端连接空闲超时时间。在超过空闲超时时间一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。	-
请求超时时间	等待客户端请求超时时间。包括两种情况： ■ 读取整个客户端请求头的超时时长，如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 ■ 两个连续body体的数据包到达LB的时间间隔，超出请求超时时间将会断开连接。	-
响应超时时间	等待后端服务器响应超时时间。请求转发后端服务器后，在等待超过响应超时时间没有响应，负载均衡将终止等待，并返回HTTP504错误码。	-
开启HTTP2	客户端与ELB之间的HTTPS请求的HTTP2功能的开启状态。 开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。	当**监听器配置**为HTTPS协议时支持配置。

  - 重定向至HTTPS：**前端协议**选择使用HTTP时，支持重定向至HTTPS。开启后可单击“修改”，对HTTPS协议的端口进行配置，详细配置说明请参见HTTPS协议的**监听器配置**。

- **灰度发布**：路由创建完成后，可在路由操作列中创建灰度发布策略。详情请参见**为ELB Ingress配置灰度发布**章节。

- **转发策略配置：**请求的访问地址与转发规则匹配时（转发规则由域名、URL组成，例如：10.117.117.117:80/helloworld），此请求将被转发到对应的目标Service处理。单击 + 按钮可添加多条转发策略。
  - 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。
  - 路径匹配规则：
    - 前缀匹配：例如映射URL为/healthz，只要符合此前缀的URL均可访问。例如/healthz/v1，/healthz/v2。
    - 精确匹配：表示只有URL完全匹配时，访问才能生效。例如映射URL为/healthz，则必须为此URL才能访问。
    - 正则匹配：按正则表达式方式匹配URL。例如正则表达式为**/[A-Za-z0-9_.-]+/test**。只要符合此规则的URL均可访问，例如/abcA9/test，/v1-Ab/test。正则匹配规则支持POSIX与Perl两种标准。
  - 路径：需要注册的访问路径，例如：/healthz。

    说明

    此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。

    例如，Nginx应用默认的Web访问路径为“/usr/share/nginx/html”，在为Ingress转发策略添加“/test”路径时，需要应用的Web访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回404。
  - 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。
  - 目标服务访问端口：可选择目标Service的访问端口。
  - 负载均衡配置：
    - 分配策略：可选择加权轮询算法、加权最少连接或源IP算法。

      说明

      - 加权轮询算法：根据后端服务器的权重，按顺序依次将请求分发给不同的服务器。它用相应的权重表示服务器的处理性能，按照权重的高低以及轮询方式将请求分配给各服务器，相同权重的服务器处理相同数目的连接数。常用于短连接服务，例如HTTP等服务。
      - 加权最少连接：最少连接是通过当前活跃的连接数来估计服务器负载情况的一种动态调度算法。加权最少连接就是在最少连接数的基础上，根据服务器的不同处理能力，给每个服务器分配不同的权重，使其能够接受相应权值数的服务请求。常用于长连接服务，例如数据库连接等服务。
      - 源IP算法：将请求的源IP地址进行Hash运算，得到一个具体的数值，同时对后端服务器进行编号，按照运算结果将请求分发到对应编号的服务器上。这可以使得对不同源IP的访问进行负载分发，同时使得同一个客户端IP的请求始终被派发至某特定的服务器。该方式适合负载均衡无cookie功能的TCP协议。
    - 会话保持类型：默认不启用。支持以下类型：
      - 负载均衡器cookie：同时需填写“会话保持时间”，范围为1-1440分钟。
      - 应用程序cookie：仅共享型ELB支持设置。同时需填写“cookie名称”，长度范围为1-64。

说明

- 当**分配策略**使用源IP算法时，不支持设置会话保持。
- 1.21 以下集群的独享型负载均衡无法使用会话保持能力，如果需要使用会话保持能力，推荐使用共享型负载均衡。

- 健康检查：设置负载均衡的健康检查配置，启用时支持以下配置。

参数	说明
协议	当目标服务端口配置协议为TCP时，支持TCP/HTTP/GRPC协议。仅独享型ELB支持选择GRPC协议。当前GRPC功能仅在部分区域开放，请以控制台呈现为准。 ○ 检查路径（仅HTTP/GRPC健康检查协议支持）：指定健康检查的URL地址。检查路径只能以/开头，长度范围为1-80。
端口	健康检查默认使用业务端口（Service的NodePort或容器端口）作为健康检查的端口；您也可以重新指定端口用于健康检查，重新指定端口会为服务增加一个名为cce-healthz的服务端口配置。 ○ 节点端口：使用共享型负载均衡或不关联ENI实例时，节点端口作为健康检查的检查端口；如不指定将随机一个端口。取值范围为30000-32767。 ○ 容器端口：使用独享型负载均衡关联ENI实例时，容器端口作为健康检查的检查端口。取值范围为1-65535。
检查周期（秒）	每次健康检查响应的最大间隔时间，取值范围为1-50。
超时时间（秒）	每次健康检查响应的最大超时时间，取值范围为1-50。
最大重试次数	健康检查最大的重试次数，取值范围为1-10。

  - 操作：可单击“删除”按钮删除该配置。
  - 执行动作：支持“重定向至URL”和“重写URL”，仅独享型ELB支持。
    - 重定向至URL：当访问请求满足转发策略时，会将请求重定向至指定的URL，并返回指定的状态码。
    - 重写URL：当访问请求满足转发策略时，会根据匹配规则重写URL路径。支持使用正则表达式匹配的结果作为重写路径，您需要选择路径匹配规则为“正则匹配”。例如，转发规则中的URL配置为正则表达式/first/(.*)/(.*)/end，重写路径配置为/${1}/${2}。当客户端发送请求的路径为/first/aaa/bbb/end时，转发规则会匹配到/first/(.*)/(.*)/end，重写路径中的${1}会替换为aaa，${2}会替换为bbb，最终后端服务器接收到的请求路径为/aaa/bbb。

- **注解**：Ingress有一些CCE定制的高级功能，通过注解annotations实现。在使用kubectl创建时，会用到注解，具体请参见**添加Ingress时自动创建ELB**和**添加Ingress时对接已有ELB**。

步骤4 配置完成后，单击“确定”。创建完成后，在Ingress列表可查看到已添加的Ingress。

在ELB控制台可查看通过CCE自动创建的ELB，名称默认为“cce-lb-<ingress.UID>”。单击ELB名称进入详情页，在“监听器”页签下即可查看Ingress对应的监听器及转发策略。

须知

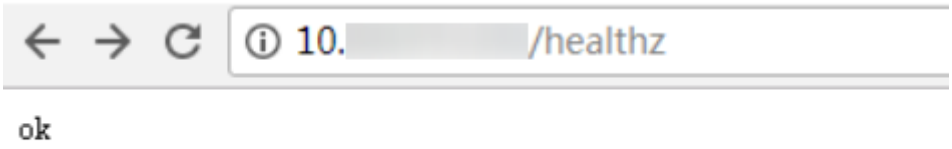
Ingress创建后请在CCE页面对所选ELB实例进行配置升级和维护，不要在ELB控制台对ELB实例进行维护，否则可能导致Ingress服务异常。

图 7-60 ELB 路由设置

步骤5 访问工作负载（例如名称为defaultbackend）的“/healthz”接口。

1. 获取工作负载“/healthz”接口的访问地址。访问地址由负载均衡实例IP、对外端口、映射URL组成，例如：10.**.**.**:80/healthz。
2. 在浏览器中输入“/healthz”接口的访问地址，如：http://10.**.**.**:80/healthz，即可成功访问工作负载，如图7-61。

图 7-61 访问 defaultbackend“/healthz”接口



----结束

相关操作

由于社区Ingress结构体中没有property属性，用户使用client-go调用创建ingress的api接口时，创建的Ingress中没有property属性。为了与社区的client-go兼容，CCE提供了相关解决方案，具体请参见**Ingress中的property字段如何实现与社区client-go兼容?** 。

7.4.2.2 通过 Kubectl 命令行创建 ELB Ingress

本文以**Nginx工作负载**为例，说明通过kubectl命令添加ELB Ingress的方法。

- 如您在同一VPC下没有可用的ELB，CCE支持在添加Ingress时自动创建ELB，请参考**添加Ingress时自动创建ELB**。

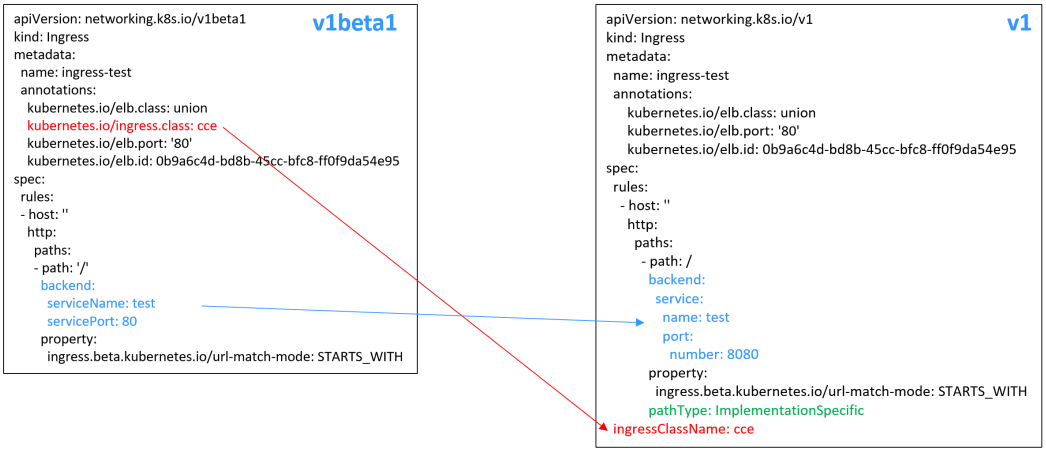
- 如您已在同一VPC下提前创建了一个可用的ELB，则可参考**添加Ingress时对接已有ELB**。

关于 CCE v1.23 集群中 Ingress API 版本升级的说明

CCE从v1.23版本集群开始，将Ingress切换到**networking.k8s.io/v1**版本。

v1版本的参数相较v1beta1版本的参数有如下区别：

- ingress类型由annotations中**kubernetes.io/ingress.class**变为使用**spec.ingressClassName**字段。
- **backend**的写法变化。
- 每个路径下必须指定路径类型**pathType**，支持如下类型。
  - ImplementationSpecific: 对于这种路径类型，匹配方法取决于具体Ingress Controller的实现。在CCE中会使用ingress.beta.kubernetes.io/url-match-mode指定的匹配方式，这与v1beta1方式相同。
  - Exact：精确匹配 URL 路径，且区分大小写。
  - Prefix：基于以 / 分隔的 URL 路径前缀匹配。匹配区分大小写，并且对路径中的元素逐个匹配。 路径元素指的是由 / 分隔符分隔的路径中的标签列表。



前提条件

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 独享型ELB规格必须支持应用型（HTTP/HTTPS），且网络类型必须支持私网（有私有IP地址）。

使用 kubectl 命令行创建 Ingress

您可以在添加Ingress时选择是否自动创建ELB或使用已有的ELB。

添加 Ingress 时自动创建 ELB

下面介绍如何通过kubectl命令在添加Ingress时自动创建ELB。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

说明

CCE在1.23版本集群开始Ingress切换到networking.k8s.io/v1版本，之前版本集群使用networking.k8s.io/v1beta1。v1版本与v1beta1版本的区别请参见**关于CCE v1.23集群中Ingress API版本升级的说明**。

共享型负载均衡（公网访问）示例 -1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.autocreate:
      '{
          "type":"public",
          "bandwidth_name":"cce-bandwidth-******",
          "bandwidth_chargemode":"bandwidth",
          "bandwidth_size":5,
          "bandwidth_sharetype":"PER",
          "vip_subnet_cidr_id":  "******",
          "vip_address":  "**.**.**.**",
          "eip_type":"5_bgp"
        }'
    kubernetes.io/elb.tags: key1=value1,key2=value2          # 添加ELB资源标签
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: <your_service_port>  #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce    # 表示使用ELB Ingress
```

共享型负载均衡（公网访问）示例 - 1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/ingress.class: cce    # 表示使用ELB Ingress
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.autocreate:
      '{
          "type":"public",
          "bandwidth_name":"cce-bandwidth-******",
          "bandwidth_chargemode":"bandwidth",
          "bandwidth_size":5,
          "bandwidth_sharetype":"PER",
          "eip_type":"5_bgp"
        }'
    kubernetes.io/elb.tags: key1=value1,key2=value2          # 添加ELB资源标签
spec:
  rules:
```



```
- host: ''
  http:
    paths:
    - path: '/'
      backend:
        serviceName: <your_service_name>  #替换为您的目标服务名称
        servicePort: <your_service_port>  #替换为您的目标服务端口
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

独享型负载均衡（公网访问）示例 - 1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "vip_subnet_cidr_id": "*****",
          "vip_address": "**.**.**.**",
          "elb_virsubnet_ids":["*****"],
          "available_zone": [
              "ap-southeast-1a"
          ],
          "l7_flavor_name": "L7_flavor.elb.s1.small"
      }'
    kubernetes.io/elb.tags: key1=value1,key2=value2           # 添加ELB资源标签
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: <your_service_port>  #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

独享型负载均衡（公网访问）示例 - 1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
```



```
        "elb_virsubnet_ids":[ "*****"],
        "available_zone": [
            "ap-southeast-1a"
        ],
        "l7_flavor_name": "L7_flavor.elb.s1.small"
    }'
  kubernetes.io/elb.tags: key1=value1,key2=value2          # 添加ELB资源标签
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: <your_service_port>  #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

表 7-73 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.class	是	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： ● union：共享型负载均衡。 ● performance：独享型负载均衡，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。 默认值为“union”
kubernetes.io/ingress.class	是 （仅1.21及以下集群）	String	cce：表示使用自研ELBIngress。 通过API接口创建Ingress时必须增加该参数。
ingressClassName	是 （仅1.23及以上集群）	String	cce：表示使用自研ELBIngress。 通过API接口创建Ingress时必须增加该参数。
kubernetes.io/elb.port	是	String	界面上的对外端口，为注册到负载均衡服务地址上的端口。 取值范围：1~65535。 **说明** 部分端口为高危端口，默认被屏蔽，如21端口。

参数	是否必填	参数类型	描述
kubernetes.io/elb.subnet-id	-	String	为集群所在子网的ID，取值范围：1~100字符。 • Kubernetes v1.11.7-r0及以下版本的集群自动创建时：必填。 • Kubernetes v1.11.7-r0以上版本的集群：可不填，默认为""。 获取方法请参见：**VPC子网接口与OpenStack Neutron子网接口的区别是什么？**
kubernetes.io/elb.enterpriseID	否	String	**Kubernetes v1.15及以上版本的集群支持此字段；Kubernetes v1.15以下版本默认创建到default项目下。** 企业项目ID，选择后可以直接创建在具体的企业项目下。 取值范围：1~100字符。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“企业 > 项目管理”，在打开的企业项目列表中单击要加入的企业项目名称，进入企业项目详情页，找到“ID”字段复制即可。
kubernetes.io/elb.autocreate	是	elb.autocreate object	自动创建Ingress关联的ELB，详细字段说明参见**表7-74**。 **示例：** • 自动创建公网共享型ELB： 值为 '{"type":"public","bandwidth_name":"cce-bandwidth-******","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}' • 自动创建共享型ELB： 值为 '{"type":"inner", "name": "A-location-d-test"}'
kubernetes.io/elb.tags	否	String	为ELB添加资源标签，仅自动创建ELB时支持设置，且集群版本需满足v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上。 格式为key=value，同时添加多个标签时以英文逗号（,）隔开。

参数	是否必填	参数类型	描述
host	否	String	为服务访问域名配置，默认为""，表示域名全匹配。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。
path	是	String	为路由路径，用户自定义设置。所有外部访问请求需要匹配host和path。 **说明** 此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 例如，Nginx应用默认的Web访问路径为“/usr/share/nginx/html”，在为Ingress转发策略添加“/test”路径时，需要应用的Web访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回404。
ingress.beta.kubernetes.io/url-match-mode	否	String	路由匹配策略。 默认值为“STARTS_WITH”(前缀匹配)。 取值范围： ● EQUAL_TO：精确匹配 ● STARTS_WITH：前缀匹配 ● REGEX：正则匹配

参数	是否必填	参数类型	描述
pathType	是	String	路径类型，该字段仅v1.23及以上集群支持。 ● ImplementationSpecific: 匹配方法取决于具体Ingress Controller的实现。在CCE中会使用ingress.beta.kubernetes.io/url-match-mode指定的匹配方式。 ● Exact：精确匹配 URL 路径，且区分大小写。 ● Prefix：前缀匹配，且区分大小写。该方式是将URL路径通过“/”分隔成多个元素 ，并且对元素进行逐个匹配。 如果URL中的每个元素均和路径匹配，则说明该URL的子路径均可以正常路由。 **说明** – Prefix匹配时每个元素均需精确匹配，如果URL的最后一个元素是请求路径中最后一个元素的子字符串，则不会匹配 。例如：/foo/bar匹配/foo/bar/baz，但不匹配/foo/barbaz。 – 通过“/”分隔元素时，若URL或请求路径以“/”结尾，将会忽略结尾的“/”。例如：/foo/bar会匹配/foo/bar/。 关于Ingress路径匹配示例，请参见示例。

表 7-74 elb.autocreate 字段数据结构说明

参数	是否必填	参数类型	描述
name	否	String	自动创建的负载均衡的名称。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。 默认名称：cce-lb+service.UID
type	否	String	负载均衡实例网络类型，公网或者私网。 ● public：公网型负载均衡 ● inner：私网型负载均衡 默认类型：inner

参数	是否必填	参数类型	描述
bandwidth_name	公网型负载均衡必填	String	带宽的名称，默认值为：cce-bandwidth-******。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。
bandwidth_chargemode	否	String	带宽付费模式。 ● bandwidth：按带宽 ● traffic：按流量 默认类型：bandwidth
bandwidth_size	公网型负载均衡必填	Integer	带宽大小，默认1Mbit/s~2000Mbit/s，请根据Region带宽支持范围设置。 调整带宽时的最小单位会根据带宽范围不同存在差异。 ● 小于等于300Mbit/s：默认最小单位为1Mbit/s。 ● 300Mbit/s~1000Mbit/s：默认最小单位为50Mbit/s。 ● 大于1000Mbit/s：默认最小单位为500Mbit/s。
bandwidth_sharetype	公网型负载均衡必填	String	带宽共享方式。 ● PER：独享带宽
eip_type	公网型负载均衡必填	String	弹性公网IP类型。 ● 5_telcom：电信 ● 5_union：联通 ● 5_bgp：全动态BGP ● 5_sbgp：静态BGP 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
vip_subnet_cidr_id	否	String	指定ELB所在的子网，该子网必须属于集群所在的VPC。 如不指定，则ELB与集群在同一个子网。 仅v1.21及以上版本的集群支持指定该字段。

参数	是否必填	参数类型	描述
vip_address	否	String	负载均衡器的内网IP。仅支持指定IPv4地址，不支持指定IPv6地址。 该IP必须为ELB所在子网网段中的IP。若不指定，自动从ELB所在子网网段中生成一个IP地址。 仅v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上版本集群支持指定该字段。
available_zone	是	Array of strings	负载均衡所在可用区。 可以通过**查询可用区列表**获取所有支持的可用区。 独享型负载均衡器独有字段。
l4_flavor_name	是	String	四层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段。
l7_flavor_name	否	String	七层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段，必须与l4_flavor_name对应规格的类型一致，即都为弹性规格或都为固定规格。
elb_virsubnet_ids	否	Array of strings	负载均衡后端所在子网，不填默认为集群子网。不同实例规格将占用不同数量子网IP，不建议使用其他资源（如集群，节点等）的子网网段。 独享型负载均衡器独有字段。 示例： `"elb_virsubnet_ids": [ "14567f27-8ae4-42b8-ae47-9f847a4690dd" ]`
ipv6_vip_virsubnet_id	否	String	双栈类型负载均衡器所在子网的IPv6网络ID，需要对应的子网开启IPv6，仅使用双栈集群时需填写。 独享型负载均衡器独有字段。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```


回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS     ADDRESS         PORTS   AGE
ingress-test  cce      *         121.**.**.**    80      10s
```

步骤5 访问工作负载（例如**Nginx工作负载**），在浏览器中输入访问地址http://121.**.**.**:80进行验证。

其中，121.**.**.**为统一负载均衡实例的IP地址。

----结束

添加 Ingress 时对接已有 ELB

CCE支持在添加Ingress时选择对接已有的ELB。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

说明

- CCE在1.23版本集群开始Ingress切换到networking.k8s.io/v1版本，之前版本集群使用networking.k8s.io/v1beta1。v1版本与v1beta1版本的区别请参见**关于CCE v1.23集群中Ingress API版本升级的说明**。
- 对接已有独享型ELB规格必须支持应用型（HTTP/HTTPS），且网络类型必须支持私网（有私有IP）。

以1.23及以上版本集群为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.ip: <your_elb_ip>  #替换为您已有的ELB IP
    kubernetes.io/elb.class: performance  #ELB类型
    kubernetes.io/elb.port: '80'
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 8080            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

以1.21及以下版本集群为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.ip: <your_elb_ip>  #替换为您已有的ELB IP
    kubernetes.io/elb.class: performance  #ELB类型
```



```
    kubernetes.io/elb.port: '80'
    kubernetes.io/ingress.class: cce
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

表 7-75 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.id	是	String	为负载均衡实例的ID，取值范围：1-100字符。 **获取方法：** 在控制台的“服务列表”中，单击“网络 > 弹性负载均衡 ELB”，单击ELB的名称，在ELB详情页的“基本信息”页签下找到“ID”字段复制即可。
kubernetes.io/elb.ip	否	String	为负载均衡实例的服务地址，公网ELB配置为公网IP，私网ELB配置为私网IP。
kubernetes.io/elb.class	是	String	负载均衡器类型。 • union：共享型负载均衡。 • performance：独享型负载均衡，仅支持1.17及以上集群，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。 **说明** ELB Ingress对接已有的独享型ELB时，该独享型ELB必须支持应用型（HTTP/HTTPS）规格。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80      10s
```

步骤5 访问工作负载（例如**Nginx工作负载**），在浏览器中输入访问地址http://121.**.**.**:80进行验证。

其中，121.**.**.**为统一负载均衡实例的IP地址。

----结束

7.4.2.3 用于配置 ELB Ingress 的注解（Annotations）

通过在YAML中添加注解Annotation（注解），您可以实现更多的Ingress高级功能。本文介绍在创建ELB类型的Ingress时可供使用的Annotation。

索引

功能分类	Ingress注解配置
ELB配置	• 对接ELB的基本配置 • 添加资源标签
端口/协议配置	• 配置ELB证书 • 使用HTTP/2 • 对接HTTPS/GRPC协议的后端服务 • 配置多个监听端口
ELB监听器高级配置	• 配置Ingress超时时间 • 设置慢启动持续时间 • 黑名单/白名单设置 • 配置HTTP/HTTPS头字段 • 开启gzip压缩 • 配置自定义EIP
转发策略配置	• 配置灰度发布 • 配置URL重定向 • 配置Rewrite重写 • 配置HTTP重定向到HTTPS • 配置转发规则优先级 • 配置自定义Header转发策略 • 配置跨域访问 • 写入/删除Header • 配置高级转发规则

对接 ELB 的基本配置

具体使用场景和示例如下：

- 关联已有ELB场景：详情请参见**添加Ingress时对接已有ELB**
- 自动创建ELB场景：详情请参见**添加Ingress时自动创建ELB**

表 7-76 对接 ELB 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.class	String	请根据不同的应用场景和功能需求选择合适的负载均衡器类型。 取值如下： ● union：共享型负载均衡。 ● performance：独享型负载均衡，仅支持1.17及以上集群，详情请参见**共享型弹性负载均衡与独享型负载均衡的功能区别**。	v1.9及以上
kubernetes.io/ingress.class	String	● cce：表示使用自研ELB Ingress。 ● nginx：表示使用Nginx Ingress。 通过API接口创建Ingress时必须增加该参数。 v1.23及以上集群使用ingressClassName参数代替，详情请参见**通过Kubectl命令行创建ELB Ingress**。	仅v1.21及以下集群
kubernetes.io/elb.port	String	界面上的对外端口，为注册到负载均衡服务地址上的端口。 取值范围：1~65535。 **说明** 部分端口为高危端口，默认被屏蔽，如21端口。	v1.9及以上
kubernetes.io/elb.id	String	**仅关联已有ELB的场景**：必填。 为负载均衡实例的ID。 **获取方法：** 在控制台的“服务列表”中，单击“网络 > 弹性负载均衡 ELB”，单击ELB的名称，在ELB详情页的“基本信息”页签下找到“ID”字段复制即可。	v1.9及以上
kubernetes.io/elb.ip	String	**仅关联已有ELB的场景**：必填。 为负载均衡实例的服务地址，公网ELB配置为公网IP，私网ELB配置为私网IP。	v1.9及以上

参数	类型	描述	支持的集群版本
kubernetes.io/elb.autocreate	表7-97 Object	**仅自动创建ELB的场景**：必填。 **示例：** • 自动创建公网共享型ELB： 值为 '{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}' • 自动创建私网共享型ELB： 值为 '{"type":"inner", "name": "A-location-d-test"}'	v1.9及以上
kubernetes.io/elb.enterpriseID	String	**仅自动创建ELB的场景**：选填。 **v1.15及以上版本的集群支持此字段，v1.15以下版本默认创建到default项目下。** 为ELB企业项目ID，选择后可以直接创建在具体的ELB企业项目下。 该字段不传（或传为字符串'0'），则将资源绑定给默认企业项目。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“企业 > 项目管理”，在打开的企业项目列表中单击要加入的企业项目名称，进入企业项目详情页，找到“ID”字段复制即可。	v1.15及以上
kubernetes.io/elb.subnet-id	String	**仅自动创建ELB的场景**：选填。 为集群所在子网的ID，取值范围：1-100字符。 • Kubernetes v1.11.7-r0及以下版本的集群自动创建时：必填 • Kubernetes v1.11.7-r0以上版本的集群：可不填。	v1.11.7-r0以下必填 v1.11.7-r0以上该字段废弃

配置 ELB 证书

具体使用场景和示例请参见**通过kubectl命令行配置**。

表 7-77 配置 ELB 证书注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的证书ID列表，不同ID间使用英文逗号隔开，列表长度大于等于1。列表中的首个ID为服务器证书，其余ID为SNI证书（SNI证书中必须带有域名）。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。	v1.19.16-r2、v1.21.5-r0、v1.23.3-r0及以上版本

添加资源标签

具体使用场景和示例请参见**添加Ingress时自动创建ELB**。

表 7-78 添加资源标签注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.tags	String	为ELB添加资源标签，仅自动创建ELB时支持设置。 格式为key=value，同时添加多个标签时以英文逗号（,）隔开。	v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上

使用 HTTP/2

具体使用场景和示例请参见**为ELB Ingress配置HTTP/2**。

表 7-79 使用 HTTP/2 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.http2-enable	String	表示HTTP/2功能的开启状态。开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。 取值范围： • true：开启HTTP/2功能； • false：关闭HTTP/2功能（默认为关闭状态）。 注意：**只有当监听器的协议为HTTPS时，才支持开启或关闭HTTP/2功能**。当监听器的协议为HTTP时，该字段无效，默认将其设置为false。	v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本

对接 HTTPS/GRPC 协议的后端服务

具体使用场景和示例请参见**为ELB Ingress配置HTTPS协议的后端服务**和**为ELB Ingress配置GRPC协议的后端服务**。

表 7-80 对接 HTTPS 协议的后端服务注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.pool-protocol	String	对接HTTPS协议的后端服务，取值为'https'。	v1.23.8、v1.25.3及以上
		对接GRPC协议的后端服务，取值为'grpc'。	v1.23.10-r20、v1.25.5-r20、v1.27.2-r20、v1.28.1-r0及以上

配置 Ingress 超时时间

具体使用场景和示例请参见**为ELB Ingress配置超时时间**。

表 7-81 配置 Ingress 超时时间注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.keepalive_timeout	String	客户端连接空闲超时时间，在超过keepalive_timeout时长一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。 取值： • 若为TCP协议，取值范围为（10-4000s）默认值为300s。 • 若为HTTP/HTTPS协议，取值范围为（0-4000s）默认值为60s。 UDP监听器不支持此字段。	独享型ELB：v1.19.16-r30、v1.21.10-r10、v1.23.8-r10、v1.25.3-r10及以上 共享型ELB：v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本
kubernetes.io/elb.client_timeout	String	等待客户端请求超时时间，包括两种情况： • 读取整个客户端请求头的超时时长：如果客户端未在超时时长内发送完整个请求头，则请求将被中断 • 两个连续body体的数据包到达LB的时间间隔，超出client_timeout将会断开连接。 取值范围为1-300s，默认值为60s。 使用说明：仅协议为HTTP/HTTPS的监听器支持该字段。 最小值：1 最大值：300 缺省值：60	
kubernetes.io/elb.member_timeout	String	等待后端服务器响应超时时间。请求转发后端服务器后，在等待超时member_timeout时长没有响应，负载均衡将终止等待，并返回 HTTP504错误码。 取值：1-300s，默认为60s。 使用说明：仅支持协议为HTTP/HTTPS的监听器。 最小值：1 最大值：300 缺省值：60	

设置慢启动持续时间

具体使用场景和示例请参见**为ELB Ingress配置慢启动持续时间**。

表 7-82 设置慢启动持续时间注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.slowstart	String	参数说明：慢启动持续时间，单位秒。 取值范围：30-1200。 ● 独享型负载均衡器生效。 ● 目标服务分配策略类型为加权轮询算法且不开启会话保持时生效。 **说明** 负载均衡器向慢启动模式下Pod线性增加请求分配权重，当配置的慢启动持续时间期限结束后，负载均衡器向Pod发送完整的请求比例，此后本次添加的后端服务器退出慢启动模式。	v1.23及以上

配置灰度发布

具体使用场景和示例请参见**为ELB Ingress配置灰度发布**。

表 7-83 灰度发布参数说明

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.canary	string	设置Ingress灰度发布的开关。设置为true后，配合不同的注解，可以实现不同的灰度发布功能。 取值范围：true ● 独享型负载均衡器生效。 ● 设置为true后，不允许删除或修改。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.canary-weight	string	权重灰度发布权重值，设置后Ingress以权重灰度形式发布。 ● 取值为0-100的正整数，为灰度流量分配的百分比。 ● 发布成权重灰度Ingress时，该参数必填。 ● 不能与其他灰度发布功能同时设置。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本
kubernetes.io/elb.session-affinity-mode	string	开启权重灰度发布后，配置会话保持能力。 灰度发布仅支持设置为 "HTTP_COOKIE"。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本
kubernetes.io/elb.session-affinity-option	string	开启权重灰度发布会话保持能力后，会话保持的超时时间。 参数值为json字符串，格式如下： `{"persistence_timeout": "1440"}` 参数说明： ● 超时时间范围为1-1440。 ● 默认值为1440。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本
kubernetes.io/elb.canary-by-header	string	header灰度发布的Key值，表示请求头参数的名称。需要与kubernetes.io/elb.canary-by-header-value成对使用。 参数说明： 长度限制1-40字符，只允许包含字母、数字、中划线（-）和下划线（_）。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.canary-by-header-value	string	header灰度发布的Values值，需要与kubernetes.io/elb.canary-by-header成对使用。 参数值为json格式的数组，例如： `'{"values":["a","b"]}'` 参数说明： ● Values数组长度：至少需要配置1个Values值。 – 如果Ingress转发策略配置了域名和路径，最多支持配置8个Values值。 – 如果Ingress转发策略仅配置了路径，最多支持配置9个Values值。 ● Values数组取值：长度限制1-128字符，不支持空格，双引号，支持以下通配符：*（匹配0个或更多字符）和?（正好匹配1个字符）。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本
kubernetes.io/elb.canary-by-cookie	string	cookie灰度发布的key值，表示请求cookie参数的名称。需要与kubernetes.io/elb.canary-by-cookie-value成对使用。 参数说明： 长度限制1-100字符，支持包含字母、数字、以及 !%'"()*+,./:=?@^\\-_`~ 等字符。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本
kubernetes.io/elb.canary-by-cookie-value	string	cookie灰度发布的values值，需要与kubernetes.io/elb.canary-by-cookie成对使用。 参数值为json格式的数组，例如： `'{"values":["a","b"]}'` 参数说明： ● Values数组长度：至少需要配置1个Values值。 – 如果Ingress转发策略配置了域名和路径，最多支持配置8个Values值。 – 如果Ingress转发策略仅配置了路径，最多支持配置9个Values值。 ● Values数组取值：长度限制1-100字符，不支持空格，支持包含字母、数字、以及 !%'"()*+,./:=?@^\\-_`~ 等字符。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.canary-related-ingress-uid	string	灰度发布Ingress关联的原始Ingress的uid信息，用于前端展示原始Ingress和灰度发布的Ingress的关联关系。 ● 参数格式：字符串 ● 取值：原始Ingress的metadata.uid字段	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

黑名单/白名单设置

具体使用场景和示例请参见**为ELB Ingress配置黑名单/白名单访问策略**。

表 7-84 ELB 访问控制注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.acl-id	String	● 不填写该参数时：表示CCE不对ELB侧访问控制进行修改。 ● 参数值填写为空值时：表示允许所有IP访问。 ● 参数值填写为ELB的IP地址组ID时：表示开启访问控制，为ELB设置IP地址黑名单或白名单。此时需同时填写kubernetes.io/elb.acl-status和kubernetes.io/elb.acl-type参数。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“网络 > 弹性负载均衡ELB”，在网络控制台中单击“弹性负载均衡 > IP地址组”，复制目标IP地址组的“ID”即可。详情请参见**IP地址组**。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上

参数	类型	描述	支持的集群版本
kubernetes.io/elb.acl-status	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： ● on：表示开启访问控制。 ● off：表示不开启访问控制。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上
kubernetes.io/elb.acl-type	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： ● black：表示黑名单，所选IP地址组无法访问ELB地址。 ● white：表示白名单，仅所选IP地址组可以访问ELB地址。	v1.23.12-r0、v1.25.7-r0、v1.27.4-r0、v1.28.2-r0及以上

配置多个监听端口

当前支持Ingress配置自定义监听端口。通过该方式，可以将服务同时暴露80端口和443端口。

具体使用场景和示例请参见**为ELB Ingress配置多个监听端口**。

表 7-85 自定义监听端口注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.listen-ports	String	为同一个Ingress创建多个监听端口，端口号范围为1~65535。 参数值为JSON格式的字符串，示例如下： `kubernetes.io/elb.listen-ports: '[{"HTTP":80},{"HTTPS":443}]'` ● 仅支持同时配置HTTP和HTTPS协议的监听端口。 ● v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0以下集群版本中仅支持新建Ingress场景，且配置多个监听端口后annotation不支持修改和删除。v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上集群版本中支持修改和删除。 ● 同时指定多监听器（kubernetes.io/elb.listen-ports）和单监听器（kubernetes.io/elb.port）配置时，多监听器优先级更高。 ● Ingress内配置项对多个监听器同时生效，如黑白名单配置、超时时间配置。Ingress开启HTTP/2配置时，HTTP/2仅在HTTPS端口生效。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

配置 HTTP/HTTPS 头字段

具体使用场景和示例请参见**为ELB Ingress配置HTTP/HTTPS头字段**。

表 7-86 配置 HTTP/HTTPS 头字段注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.x-forwarded-port	String	ELB可通过X-Forwarded-Port头字段获取监听器的端口号，传输到后端服务器的报文中。 ● true：开启获取监听器端口号开关。 ● false：关闭获取监听器端口号开关。	v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本
kubernetes.io/elb.x-forwarded-for-port	String	ELB可通过X-Forwarded-For-Port头字段获取客户端请求的端口号，传输到后端服务器的报文中。 ● true：开启获取客户端请求端口号开关。 ● false：关闭获取客户端请求端口号开关。	
kubernetes.io/elb.x-forwarded-host	String	● true：开启重写X-Forwarded-Host开关，ELB以客户端请求头的Host重写X-Forwarded-Host传递到后端服务器。 ● false：关闭重写X-Forwarded-Host开关，ELB透传客户端的X-Forwarded-Host到后端服务器。	

开启 gzip 压缩

具体使用场景和示例请参见**为ELB Ingress配置gzip数据压缩**。

表 7-87 开启 gzip 压缩注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.gzip-enabled	String	LoadBalancer支持开启数据压缩，通过数据压缩可缩小传输文件大小，提升文件传输效率减少带宽消耗。 开启将对特定文件类型进行压缩，关闭则不会对任何文件类型进行压缩。在默认情况下数据压缩为关闭。 支持的压缩类型如下： • Brotli支持压缩所有类型。 • Gzip支持压缩的类型包括：text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。 仅独享型ELB的HTTP/HTTPS类型监听器支持配置。删除开启数据压缩高级配置或对应的annotation，将不会对ELB侧配置进行修改。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

配置 URL 重定向

具体使用场景和示例请参见**为ELB Ingress配置URL重定向**。

表 7-88 配置 URL 重定向注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.redirect-url	String	重定向URL信息。 格式说明：以 "http://" 或 "https://" 开头的合法的URL，如 https://example.com/。 参数说明：对单个Ingress下所有的转发规则均生效，配置删除后自动清理对应的重定向URL规则。 该注解不能和灰度发布的注解一起配置。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

参数	类型	描述	支持的集群版本
kubernetes.io/elb.redirect-url-code	String	重定向URL后的返回码。 格式说明：支持返回码包括"301"、"302"、"303"、"307"、"308"。 参数说明：默认值为"301"。	

配置 Rewrite 重写

具体使用场景和示例请参见**为ELB Ingress配置Rewrite重写**。

表 7-89 配置 Rewrite 重写注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.rewrite-target	String	重写路径的信息。 格式说明：以 "/" 开头的合理的正则匹配规则。 参数说明：对单个Ingress下正则匹配的URL转发规则生效，配置删除后自动清理对应的重写规则。 该注解不能和灰度发布的注解一起配置。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

配置 HTTP 重定向到 HTTPS

具体使用场景和示例请参见**为ELB Ingress配置HTTP重定向到HTTPS**。

表 7-90 配置 HTTP 重定向到 HTTPS 注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.ssl-redirect	String	是否开启HTTP重定向到HTTPS。 格式说明：支持字段 "true" 和 "false" 参数说明："true" 表示开启重定向能力，"false"或参数不存在表示不开启重定向能力 **说明** 该注解不能和灰度发布的注解一起配置。	v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本

配置转发规则优先级

Ingress使用同一个ELB监听器时，支持按照以下规则进行转发规则优先级排序：

- 不同Ingress的转发规则：按照“kubernetes.io/elb.ingress-order”注解的优先级（取值范围为1~1000）进行排序，值越小表示优先级越高。
- 同一个Ingress的转发规则：“kubernetes.io/elb.rule-priority-enabled”注解设置为“true”时，根据创建Ingress时的转发规则先后顺序进行排序，配置在上面的优先级高；未配置该注解时，同一个Ingress的转发规则会使用ELB侧的默认排序。

未配置以上注解时，同一个ELB监听器下无论包含多个Ingress还是同一个Ingress的转发规则，都会使用ELB侧的默认排序规则。

具体使用场景和说明请参见**为ELB Ingress配置转发规则优先级**。

表 7-91 转发规则优先级注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.ingress-order	String	不同Ingress间的转发规则排序，参数值越小表示优先级越高，且同一个监听内规则优先级必须唯一，取值范围为1~1000。 仅独享型ELB支持配置。 **说明** 配置该注解时，默认开启“kubernetes.io/elb.rule-priority-enabled”注解，即同时对每个Ingress的转发规则进行排序。	v1.23.15-r0、v1.25.10-r0、v1.27.7-r0、v1.28.5-r0、v1.29.1-r10及以上版本
kubernetes.io/elb.rule-priority-enabled	String	仅支持设置为“true”，表示对同一个Ingress的转发规则进行排序，系统将会根据创建Ingress时的转发规则先后顺序确定优先级，配置在上面的优先级高。 未开启该参数时，同一个Ingress的转发规则会使用ELB侧的默认排序，开启后则不允许关闭。 仅独享型ELB支持配置。	

配置自定义 Header 转发策略

具体使用场景和示例请参见**为ELB Ingress配置自定义Header转发策略**。

表 7-92 自定义 Header 转发策略注解

参数	类型	描述	支持的集群版本
kubernetes.io/elb.headers.*${svc_name}*	String	Ingress关联的Service配置自定义的Header，${svc_name}即对应的Service名称。 格式说明：Json字符串，如 {"key": "test", "values": ["value1", "value2"]} • key/value表示自定义Header的键值对，value最多可以配置8个。 key的取值范围：长度限制1-40字符，只允许包含字母、数字、中划线（-）和下划线（_） value的取值范围：长度限制1-128字符，不支持空格，双引号，支持以下通配符：*（匹配0个或更多字符）和?（正好匹配1个字符） • 不能和灰度发布同时配置 • svc_name最大长度51个字符	v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本

配置自定义 EIP

具体使用场景和示例请参见**为ELB Ingress配置自定义EIP**。

表 7-93 配置自定义 EIP 注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.custom-eip-id	String	自定义EIP的ID，您可以前往EIP控制台查看。该EIP必须是处于可绑定状态。	v1.23.18-r0、v1.25.13-r0、v1.27.10-r0、v1.28.8-r0、v1.29.4-r0、v1.30.1-r0及以上版本

配置跨域访问

具体使用场景和示例请参见**为ELB Ingress配置跨域访问**。

表 7-94 配置跨域访问注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.cors-allow-origin	Array[string]	指定Access-Control-Allow-Origin响应头的值，表示允许访问的域。 支持以下取值： • 通配符*：表示允许所有域名访问。 • 配置域名列表：必须为http://或者https://开头的域名，支持填写一级泛域名。格式为“http(s)://example.com”或“http(s)://example.com:port”，端口范围为1~65535。 可填写多个值，以英文逗号分隔。	v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上版本
kubernetes.io/elb.cors-allow-headers	Array[string]	指定Access-Control-Allow-Headers响应头的值，表示允许的请求头。可填写多个值，以英文逗号分隔。	
kubernetes.io/elb.cors-expose-headers	Array[string]	指定Access-Control-Expose-Headers响应头的值，表示可以被跨域请求读取的自定义响应头部，例如通过客户端的JavaScript代码获取非标准响应头字段。可填写多个值，以英文逗号分隔。	
kubernetes.io/elb.cors-allow-methods	Array[string]	指定Access-Control-Allow-Methods响应头的值，表示允许的HTTP请求方法。 可填写多个值，以英文逗号分隔。	
kubernetes.io/elb.cors-allow-credentials	String	指定Access-Control-Allow-Credentials响应头的值，表示是否允许发送凭据（如Cookies）。 取值如下： • true：允许发送凭据。 • false：不允许发送凭据。 设置后不允许删除，如需可通过kubernetes.io/elb.cors-disabled删除所有跨域配置。	

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.cors-max-age	String	指定Access-Control-Max-Age响应头的值，表示CORS预检请求的缓存时长。单位：秒。取值范围：**-1~172800**。 该参数值应该根据实际需求合理设置。如果设置得太短，可能会导致频繁的预检请求；如果设置得太长，可能会在CORS策略更新后延迟生效。	
kubernetes.io/elb.cors-disabled	String	该参数用于关闭所有跨域配置。取值如下： • **true**：关闭所有跨域配置。YAML中的参数值不会被删除，但所有配置均不生效。 • **false**：默认取值，跨域配置将根据用户设置生效。	

写入/删除 Header

具体使用场景和示例请参见**为ELB Ingress配置写入/删除Header**。

表 7-95 写入/删除 Header 注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.actions.*${svc_name}*	String	Ingress关联的Service配置重写Header，*${svc_name}*即对应的Service名称，其最大长度为51个字符。 如果该annotation值配置成*[]*则表示删除相应的重写Header的策略。 注解值格式为Json字符串数组，例如： `[{"type":"InsertHeader","InsertHeaderConfig":{"key":"aa","value_type":"USER_DEFINED","value":"aa"}}]` **说明** 最多添加5条写入Header或删除Header配置。	v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上版本

配置高级转发规则

具体使用场景和示例请参见**为ELB Ingress配置高级转发规则**。

表 7-96 写入/删除 Header 注解

参数	参数类型	描述	支持的集群版本
kubernetes.io/elb.conditions.*${svc_name}*	String	配置高级转发规则，*${svc_name}*即对应的Service名称，其最大长度为51个字符。 如果该annotation值配置成*[]*则表示删除相应的高级转发规则。 注解值格式为Json字符串数组，具体格式请参见**表7-137**。 **须知** • 由于ELB的API限制，conditions设置的数量上限为10，即一条kubernetes.io/elb.conditions.{svcName}中，最多包含十条key-value键值对。 • 一条conditions中的数组中不同的规则是“与”关系，但同一个规则块中的值是“或”关系。例如，Method和QueryString两种转发条件都配置时，需要同时满足，才能实现目标流量分发。但如果Method值为GET，POST，即只需要满足Method为GET或POST，且QueryString满足条件即可。	v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上版本

附：关于自动创建 ELB 的参数说明

表 7-97 elb.autocreate 字段数据结构说明

参数	是否必填	参数类型	描述
name	否	String	自动创建的负载均衡的名称。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。 默认名称：cce-lb+service.UID
type	否	String	负载均衡实例网络类型，公网或者私网。 • public：公网型负载均衡 • inner：私网型负载均衡 默认类型：inner

参数	是否必填	参数类型	描述
bandwidth_name	公网型负载均衡必填	String	带宽的名称，默认值为：cce-bandwidth-******。 取值范围：只能由中文、英文字母、数字、下划线、中划线、点组成，且长度范围为1-64个字符。
bandwidth_chargemode	否	String	带宽付费模式。 ● bandwidth：按带宽 ● traffic：按流量 默认类型：bandwidth
bandwidth_size	公网型负载均衡必填	Integer	带宽大小，默认1Mbit/s~2000Mbit/s，请根据Region带宽支持范围设置。 调整带宽时的最小单位会根据带宽范围不同存在差异。 ● 小于等于300Mbit/s：默认最小单位为1Mbit/s。 ● 300Mbit/s~1000Mbit/s：默认最小单位为50Mbit/s。 ● 大于1000Mbit/s：默认最小单位为500Mbit/s。
bandwidth_sharetype	公网型负载均衡必填	String	带宽共享方式。 ● PER：独享带宽
eip_type	公网型负载均衡必填	String	弹性公网IP类型。 ● 5_telcom：电信 ● 5_union：联通 ● 5_bgp：全动态BGP ● 5_sbgp：静态BGP 具体类型以各区域配置为准，详情请参见弹性公网IP控制台。
vip_subnet_cidr_id	否	String	指定ELB所在的子网，该子网必须属于集群所在的VPC。 如不指定，则ELB与集群在同一个子网。 仅v1.21及以上版本的集群支持指定该字段。

参数	是否必填	参数类型	描述
vip_address	否	String	负载均衡器的内网IP。仅支持指定IPv4地址，不支持指定IPv6地址。 该IP必须为ELB所在子网网段中的IP。若不指定，自动从ELB所在子网网段中生成一个IP地址。 仅v1.23.11-r0、v1.25.6-r0、v1.27.3-r0及以上版本集群支持指定该字段。
available_zone	是	Array of strings	负载均衡所在可用区。 可以通过**查询可用区列表**获取所有支持的可用区。 独享型负载均衡器独有字段。
l4_flavor_name	是	String	四层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段。
l7_flavor_name	否	String	七层负载均衡实例规格名称。 可以通过**查询规格列表**获取所有支持的类型。 独享型负载均衡器独有字段，必须与l4_flavor_name对应规格的类型一致，即都为弹性规格或都为固定规格。
elb_virsubnet_ids	否	Array of strings	负载均衡后端所在子网，不填默认为集群子网。不同实例规格将占用不同数量子网IP，不建议使用其他资源（如集群，节点等）的子网网段。 独享型负载均衡器独有字段。 示例： `"elb_virsubnet_ids": [ "14567f27-8ae4-42b8-ae47-9f847a4690dd" ]`
ipv6_vip_virsubnet_id	否	String	双栈类型负载均衡器所在子网的IPv6网络ID，需要对应的子网开启IPv6，仅使用双栈集群时需填写。 独享型负载均衡器独有字段。

7.4.2.4 ELB Ingress 高级配置示例

7.4.2.4.1 为 ELB Ingress 配置 HTTPS 证书

Ingress支持配置SSL/TLS证书，以HTTPS协议的方式对外提供安全服务。

当前支持在集群中使用以下方式配置Ingress证书：

- **使用TLS类型的密钥证书**：需要将证书导入至Secret中，CCE会将该证书自动配置到ELB侧（证书名以k8s_plb_default开头），由CCE自动创建的证书**在ELB侧不可修改或删除**。如果您需要修改证书，请在CCE侧更新对应的Secret。
- **使用ELB服务中的证书**：直接使用ELB服务中创建的证书，无需手动配置集群Secret，且可以在ELB侧修改证书。

说明

同一个ELB实例的同一个端口配置HTTPS时，需要选择一样的证书。详情请参见**多个Ingress配置同一端口说明**。

前提条件

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 已准备可信的证书，您可以从证书提供商处获取证书。操作详情请参见**购买SSL证书**。

使用 TLS 类型的密钥证书

您可以使用以下方式配置TLS类型的密钥证书。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

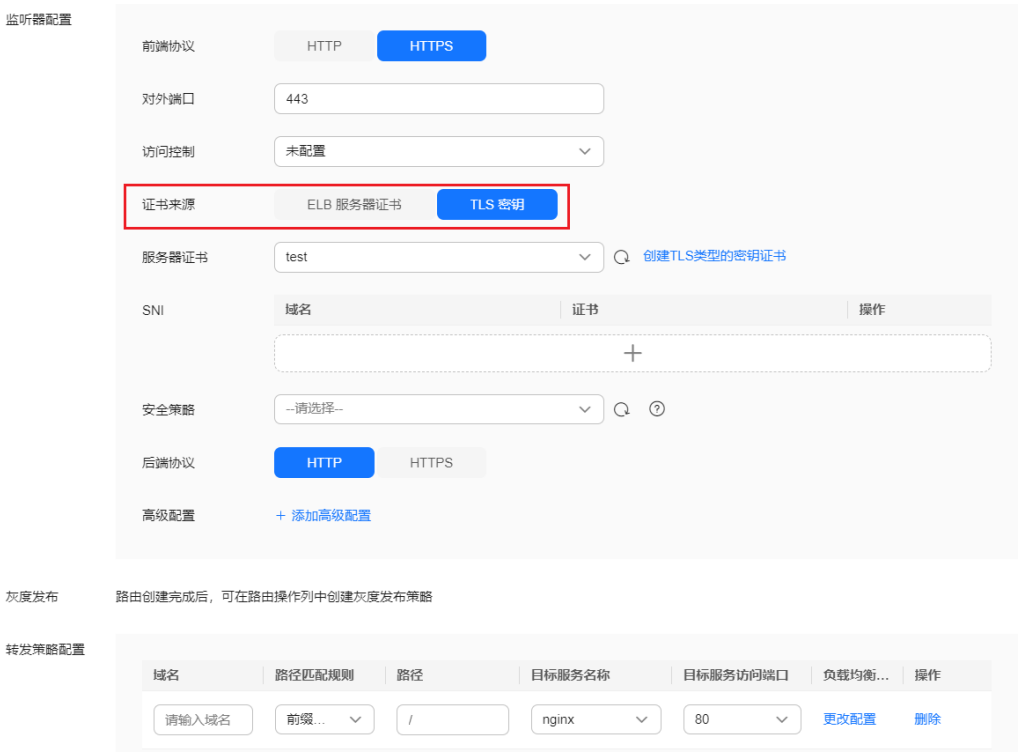
本示例中展示配置HTTPS证书的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-98 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：为Ingress配置证书需选择“HTTPS”。 • 对外端口：ELB监听器端口，HTTPS协议的端口默认为443。 • 证书来源：选择“TLS密钥”。 • 服务器证书：支持使用kubernetes.io/tls和IngressTLS两种TLS密钥类型。 如果您没有可选择的密钥证书，可新建TLS类型的密钥证书，对应参数详情请参见**创建密钥**。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：“TLS密钥” • 服务器证书：test
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-62 使用 TLS 类型的密钥证书



步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 Ingress支持使用kubernetes.io/tls和IngressTLS两种TLS密钥类型，此处以IngressTLS类型为例，详情请参见**创建密钥**。kubernetes.io/tls类型的密钥示例及说明请参见**TLS Secret**。

执行如下命令，创建名为“**ingress-test-secret.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test-secret.yaml
```

YAML文件配置如下：

```
apiVersion: v1
data:
  tls.crt: LS0******tLS0tCg==
  tls.key: LS0tL******0tLS0K
kind: Secret
metadata:
  annotations:
    description: test for ingressTLS secrets
  name: ingress-test-secret
  namespace: default
type: IngressTLS
```

说明

此处tls.crt和tls.key为示例，请获取真实的证书和密钥进行替换。tls.crt和tls.key的值为Base64编码后的内容。

步骤3 创建密钥。

```
kubectl create -f ingress-test-secret.yaml
```

回显如下，表明密钥已创建。

```
secret/ingress-test-secret created
```

步骤4 查看已创建的密钥。

```
kubectl get secrets
```

回显如下，表明密钥创建成功。

```
NAME                  TYPE                DATA   AGE
ingress-test-secret   IngressTLS          2      13s
```

步骤5 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

说明

默认安全策略选择（kubernetes.io/elb.tls-ciphers-policy）仅在1.17.17及以上版本的集群中支持。

自定义安全策略选择（kubernetes.io/elb.security_policy_id）仅在1.17.17及以上版本的集群中支持。

以自动创建关联ELB为例，YAML文件配置如下：

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "available_zone": [
              "ap-southeast-1a"
          ],
          "elb_virsubnet_ids":["b4bf8152-6c36-4c3b-9f74-2229f8e640c9"],
          "l7_flavor_name": "L7_flavor.elb.s1.small"
      }'
    kubernetes.io/elb.security_policy_id: 99bec42b-0dd4-4583-98e9-b05ce628d157 #自定义安全策略优先级高于系统默认安全策略
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```


1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "available_zone": [
              "ap-southeast-1a"
          ],
          "elb_virsubnet_ids":["b4bf8152-6c36-4c3b-9f74-2229f8e640c9"],
          "l7_flavor_name": "L7_flavor.elb.s1.small"
      }'
    kubernetes.io/elb.security_policy_id: 99bec42b-0dd4-4583-98e9-b05ce628d157  #自定义安全策略优先级高于系统默认安全策略
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80              #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-99 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.security_policy_id	否	String	ELB中自定义安全组策略ID，请前往ELB控制台获取。该字段仅在HTTPS协议下生效，且优先级高于默认安全策略。 自定义安全策略的创建、更新步骤请参见**TLS安全策略**。

参数	是否必填	参数类型	描述
kubernetes.io/elb.tls-ciphers-policy	否	String	默认值为“tls-1-2”，为监听器使用的默认安全策略，仅在HTTPS协议下生效。 取值范围： • tls-1-0 • tls-1-1 • tls-1-2 • tls-1-2-strict 各安全策略使用的加密套件列表详细参见**表7-100**。
tls	否	Array of strings	HTTPS协议时，需添加此字段用于指定密钥证书。 该字段支持添加多项独立的域名和证书，详见**为ELB Ingress配置服务器名称指示（SNI）**。
secretName	否	String	HTTPS协议时添加，配置为创建的密钥证书名称。

表 7-100 tls_ciphers_policy 取值说明

安全策略	支持的TLS版本类型	使用的加密套件列表
tls-1-0	TLS 1.2 TLS 1.1 TLS 1.0	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:AES128-SHA256:AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:AES128-SHA:AES256-SHA
tls-1-1	TLS 1.2 TLS 1.1	
tls-1-2	TLS 1.2	

安全策略	支持的TLS版本类型	使用的加密套件列表
tls-1-2-strict	TLS 1.2	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:AES128-SHA256:AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384
TLS-1-0-WITH-1-3（独享型实例）	TLS 1.3 TLS 1.2 TLS 1.1 TLS 1.0	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:AES128-SHA256:AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:AES128-SHA:AES256-SHA:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:TLS_AES_128_CCM_8_SHA256:TLS_AES_128_CCM_SHA256
TLS-1-2-FS（独享型实例）	TLS 1.3 TLS 1.2	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384
TLS-1-2-FS-WITH-1-3（独享型实例）	TLS 1.3 TLS 1.2	ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:TLS_AES_128_CCM_8_SHA256:TLS_AES_128_CCM_SHA256

步骤6 创建Ingress。

```
kubectl create -f ingress-test.yaml
```


回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤7 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS     ADDRESS         PORTS   AGE
ingress-test  cce      *         121.**.**.**    80,443  10s
```

步骤8 访问工作负载（例如**Nginx工作负载**），在浏览器中输入安全访问地址https://121.**.**.**:443进行验证。

其中，121.**.**.**为统一负载均衡实例的IP地址。

----结束

使用 ELB 服务中的证书

您可以使用以下方式为Ingress配置ELB服务中的证书。

说明

- 当同时指定ELB证书和IngressTLS证书时，Ingress将使用ELB服务中的证书。
- CCE不校验ELB服务中的证书是否有效，只校验证书是否存在。
- 仅v1.19.16-r2、v1.21.5-r0、v1.23.3-r0及以上版本的集群支持使用ELB服务中的证书。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

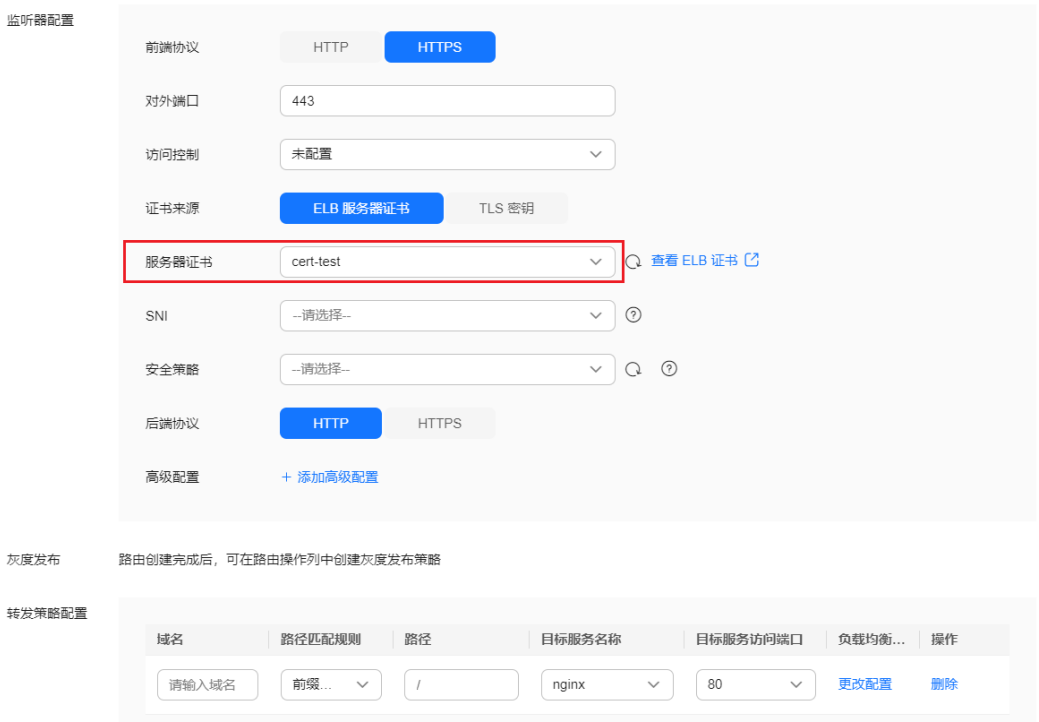
本示例中展示配置HTTPS证书的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-101 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：选择“HTTPS”。 • 对外端口：ELB监听器端口，HTTPS协议的端口默认为443。 • 证书来源：选择“ELB服务器证书”。 • 服务器证书：使用在ELB服务中创建的证书。如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：“ELB服务器证书” • 服务器证书：cert-test
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-63 使用 ELB 服务中的证书



步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

使用ELB服务中的证书，可以通过指定kubernetes.io/elb.tls-certificate-ids注解实现。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: 0b9a6c4d-bd8b-45cc-bfc8-ff0f9da54e95
    kubernetes.io/elb.class: union
    kubernetes.io/elb.tls-certificate-ids:
058cc023690d48a3867ad69dbe9cd6e5,b98382b1f01c473286653afd1ed9ab63
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
```

```
      servicePort: 80
    property:
      ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: 0b9a6c4d-bd8b-45cc-bfc8-ff0f9da54e95
    kubernetes.io/elb.class: union
    kubernetes.io/elb.tls-certificate-ids:
058cc023690d48a3867ad69dbe9cd6e5,b98382b1f01c473286653afd1ed9ab63
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: 80            #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-102 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的证书ID列表，不同ID间使用英文逗号隔开，列表长度大于等于1。列表中的首个ID为服务器证书，其余ID为SNI证书（SNI证书中必须带有域名）。 如果无法根据客户端请求的域名查找到对应的SNI证书，则默认返回服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。


```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80,443  10s
```

----结束

多个 Ingress 配置同一端口说明

在同一个集群中，多个Ingress可以使用同一个监听器（即使用同一个负载均衡器的同一个端口）。如果两个Ingress均配置了HTTPS证书，则生效的服务器证书将以最早创建的Ingress配置为准。

从集群版本v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0、v1.29.1-r0开始，Ingress的YAML中加入了annotation: kubernetes.io/elb.listener-master-ingress，其值指向的Ingress配置的服务器证书，就是当前Ingress实际生效的服务器证书。

例如，两个Ingress配置如下：

Ingress名称	ingress1	ingress2
命名空间	default	default
创建时间	2024/04/01	2024/04/02
协议	https	https
负载均衡器	elb1	elb1
端口	443	443
kubernetes.io/elb.listener-master-ingress	default/ingress1	default/ingress1

这两个Ingress的配置中，都有annotation: kubernetes.io/elb.listener-master-ingress，值为default/ingress1，表示这两个Ingress实际生效的服务器证书，都是命名空间default下ingress1配置的服务器证书。

说明

不同命名空间下的Ingress对接同一个监听器且使用TLS密钥类型证书时，由于命名空间隔离，后创建的Ingress可能出现无法正常显示TLS密钥对应Secret的场景。详情请参见**不同命名空间下的Ingress共用监听器时如何同步生效的证书？**。

当需要修改服务器证书时，需要在default命名空间下ingress1的配置中修改服务器证书，修改后ingress1和ingress2实际生效的证书都同步修改。

您可以通过执行以下命令查询实际生效的证书配置所在的Ingress：

```
kubectl get ingress -n ${namespace} ${ingress_name} -oyaml | grep kubernetes.io/elb.listener-master-ingress
```

7.4.2.4.2 更新 ELB Ingress 的 HTTPS 证书

当您面临ELB Ingress的HTTPS证书即将到期或已经过期的情况时，您可以参考本文指导更新HTTPS证书，以免对您的服务造成不必要的中断。

更新 ELB Ingress 证书场景

更新证书场景	说明
使用ELB服务中的证书	更新HTTPS证书时，需要在ELB服务中创建新的证书，然后在修改Ingress时选择新的证书。 或者您可以在ELB的证书管理页面，直接修改对应的证书内容。
使用TLS类型的密钥证书	更新HTTPS证书时，需要更新集群中对应的密钥，CCE会将该证书自动配置到ELB侧（证书名以k8s_plb_default开头），由CCE自动创建的证书**在ELB侧不可修改或删除**。

使用 TLS 类型的密钥证书场景

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“配置与密钥”，在右侧选择“密钥”页签，找到Ingress使用的密钥，单击“更新”。

步骤3 修改密钥中的证书文件，单击“确定”更新密钥配置。CCE会将该证书自动同步到ELB侧。

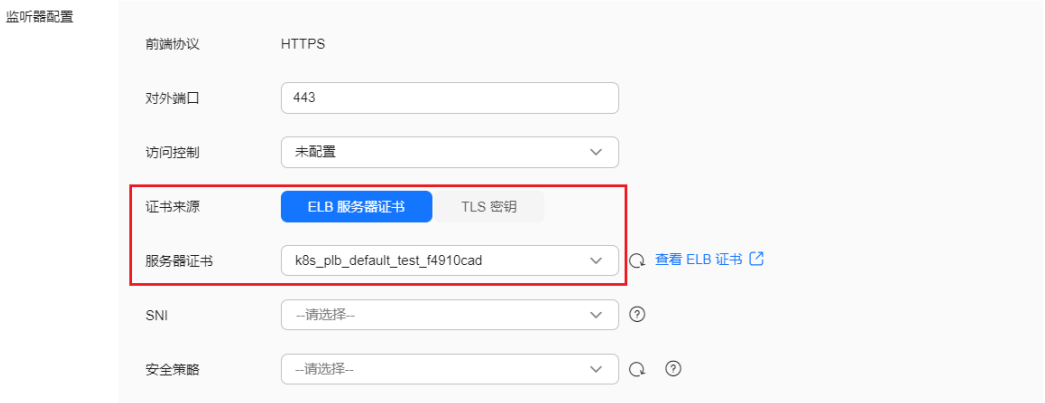
----结束

使用 ELB 服务中的证书场景

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击路由对应的“更多 > 更新”。

步骤3 选择证书来源为ELB服务器证书，并选择新的服务器证书，单击“确定”更新Ingress配置。



----结束

7.4.2.4.3 为 ELB Ingress 配置服务器名称指示（SNI）

SNI证书是一种扩展服务器证书，允许同一个IP地址和端口号下对外提供多个访问域名，可以根据客户端请求的不同域名来使用不同的安全证书，确保HTTPS通信的安全性。

在配置SNI时，用户需要添加绑定域名的证书，客户端会在发起SSL握手请求时就提交请求的域名信息，负载均衡收到SSL请求后，会根据域名去查找证书。如果找到域名对应的证书，则返回该证书；如果没有找到域名对应的证书，则返回服务器默认证书。

当前支持在集群中使用以下方式配置Ingress证书：

- **使用TLS类型的密钥证书配置SNI**：需要将证书导入至Secret中，CCE会将该证书自动配置到ELB侧（证书名以k8s_plb_default开头），由CCE自动创建的证书在ELB侧不可修改或删除。
- **使用ELB服务中的证书配置SNI**：直接使用ELB服务中创建的证书，无需手动配置集群Secret，且可以在ELB侧修改证书。

前提条件

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 已准备可信的证书，您可以从证书提供商处获取证书。操作详情请参见**购买SSL证书**。

使用 TLS 类型的密钥证书配置 SNI

您可以使用以下方式使用TLS类型的密钥证书配置SNI。

说明

- 当使用HTTPS协议时，才支持配置SNI。
- 用于SNI的证书需要指定域名，每个证书只能指定一个域名。支持泛域名证书。
- 安全策略选择（kubernetes.io/elb.tls-ciphers-policy）仅在1.17.11及以上版本的集群中支持。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

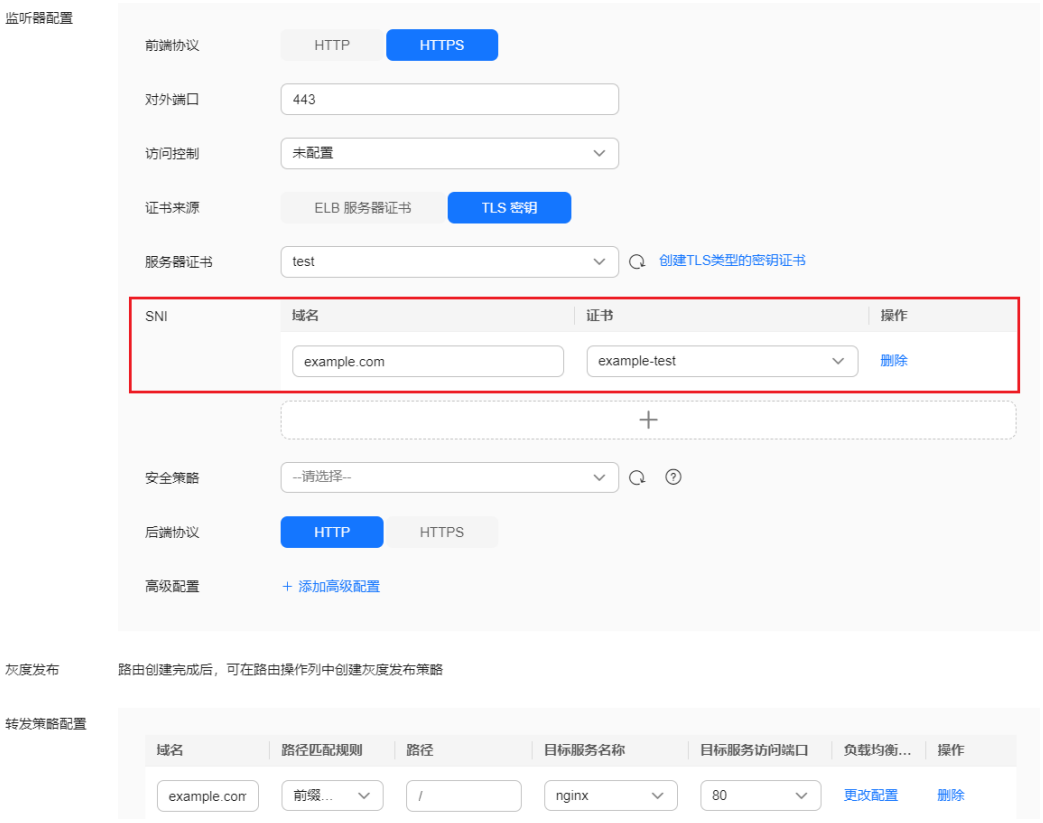
说明

本示例中展示配置SNI证书的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-103 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB
监听器配置	• 前端协议：为Ingress配置证书需选择“HTTPS”。 • 对外端口：ELB监听器端口，HTTPS协议的端口默认为443。 • 证书来源：选择“TLS密钥”。 • 服务器证书：支持使用kubernetes.io/tls和IngressTLS两种TLS密钥类型。 如果您没有可选择的密钥证书，可新建TLS类型的密钥证书，对应参数详情请参见**创建密钥**。 • SNI：输入域名并选择对应的SNI证书，SNI证书中需要包含域名信息，SNI证书支持使用kubernetes.io/tls和IngressTLS两种TLS密钥类型。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：“TLS密钥” • 服务器证书：test • SNI： – 域名：example.com – 证书：example-test
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：example.com • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-64 使用 TLS 类型的密钥证书配置 SNI



步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

本例中**sni-test-secret**为SNI证书，该证书指定的域名必须与证书中的域名一致。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以自动创建关联ELB为例，YAML文件配置如下：

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
```



```
        "available_zone": [
            "ap-southeast-1a"
        ],
        "elb_virsubnet_ids":["b4bf8152-6c36-4c3b-9f74-2229f8e640c9"],
        "l7_flavor_name": "L7_flavor.elb.s1.small"
    }'
  kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  - hosts:
    - example.com  #签发证书时指定域名为example.com
    secretName: sni-test-secret   #SNI证书
  rules:
  - host: example.com   #域名需要和tls字段中的hosts取值一致
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.autocreate:
      '{
          "type": "public",
          "bandwidth_name": "cce-bandwidth-******",
          "bandwidth_chargemode": "bandwidth",
          "bandwidth_size": 5,
          "bandwidth_sharetype": "PER",
          "eip_type": "5_bgp",
          "available_zone": [
              "ap-southeast-1a"
          ],
          "elb_virsubnet_ids":["b4bf8152-6c36-4c3b-9f74-2229f8e640c9"],
          "l7_flavor_name": "L7_flavor.elb.s1.small"
      }'
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
  - secretName: ingress-test-secret
  - hosts:
    - example.com  #签发证书时指定域名为example.com
    secretName: sni-test-secret #SNI证书
  rules:
  - host: example.com   #域名需要和tls字段中的hosts取值一致
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80             #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

步骤3 创建Ingress。


```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS  HOSTS         ADDRESS         PORTS   AGE
ingress-test  cce    example.com   121.**.**.**    80,443  10s
```

步骤5 使用HTTPS协议访问Ingress，其中${ELB_IP}为Ingress访问的IP。

```
curl -H "Host:example.com" -k https://${ELB_IP}:443
```

可正常访问则表明证书配置成功。

----结束

使用 ELB 服务中的证书配置 SNI

您可以使用以下方式为Ingress配置ELB服务中的证书。

说明

- 当同时指定ELB证书和IngressTLS证书时，Ingress将使用ELB服务中的证书。
- CCE不校验ELB服务中的证书是否有效，只校验证书是否存在。
- 仅v1.19.16-r2、v1.21.5-r0、v1.23.3-r0及以上版本的集群支持使用ELB服务中的证书。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置SNI证书的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-104 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：选择“HTTPS”。 • 对外端口：ELB监听器端口，HTTPS协议的端口默认为443。 • 证书来源：选择“ELB服务器证书”。 • 服务器证书：使用在ELB服务中创建的证书。如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。 • SNI：选择对应的SNI证书，SNI证书中需要包含域名信息。如果您没有可选择的证书，可前往ELB服务创建，详情请参见**创建证书**。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：“ELB服务器证书” • 服务器证书：cert-test • SNI：cert-example
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-65 使用 ELB 服务中的证书配置 SNI

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

您可以通过指定kubernetes.io/elb.tls-certificate-ids注解，在Ingress使用ELB服务中的证书。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: 0b9a6c4d-bd8b-45cc-bfc8-ff0f9da54e95
    kubernetes.io/elb.class: union
    kubernetes.io/elb.tls-certificate-ids:
058cc023690d48a3867ad69dbe9cd6e5,b98382b1f01c473286653afd1ed9ab63
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
```

```
      backend:
        serviceName: <your_service_name>  #替换为您的目标服务名称
        servicePort: 80
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: 0b9a6c4d-bd8b-45cc-bfc8-ff0f9da54e95
    kubernetes.io/elb.class: union
    kubernetes.io/elb.tls-certificate-ids:
058cc023690d48a3867ad69dbe9cd6e5,b98382b1f01c473286653afd1ed9ab63
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: 80              #替换为您的目标服务端口
              property:
                ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
              pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-105 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.tls-certificate-ids	String	ELB服务中的证书ID列表，不同ID间使用英文逗号隔开，列表长度大于等于1。列表中的首个ID为服务器证书，其余ID为SNI证书（SNI证书中必须带有域名）。 如果无法根据客户端请求的域名查找到对应的SNI证书，则默认返回服务器证书。 获取方法：在CCE控制台，单击顶部的“服务列表 > 网络 > 弹性负载均衡”，并选择“证书管理”。在列表中复制对应证书名称下的ID即可。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。


```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80,443  10s
```

步骤5 使用HTTPS协议访问Ingress，其中${ELB_IP}为Ingress访问的IP。

```
curl -H "Host:example.com" -k https://${ELB_IP}:443
```

可正常访问则表明证书配置成功。

----结束

7.4.2.4.4 为 ELB Ingress 配置多个转发策略

Ingress可通过不同的匹配策略同时路由到多个后端服务，例如，通过访问“www.example.com/foo”、“www.example.com/bar”、“foo.example.com/”即可分别路由到三个不同的后端Service。

须知

Ingress转发策略中注册的URL需与后端应用提供访问的URL一致，否则将返回404错误。

例如，Nginx应用默认的Web访问路径为“/usr/share/nginx/html”，在为Ingress转发策略添加“/test”路径时，需要应用的Web访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回404。

前提条件

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

Ingress 路由到多个服务

您可以使用以下方式路由到多个服务。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置转发策略的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-106 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test

参数	配置说明	示例
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB
监听器配置	• 前端协议：可选择“HTTP”或“HTTPS”。 • 对外端口：ELB监听器的端口。	• 前端协议：“HTTP” • 对外端口：80
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	转发规则一： • 域名：www.example.com • 路径匹配规则：前缀匹配 • 路径：/foo • 目标服务名称：nginx • 目标服务访问端口：80 转发规则二： • 域名：www.example.com • 路径匹配规则：前缀匹配 • 路径：/bar • 目标服务名称：nginx • 目标服务访问端口：80 转发规则三： • 域名：foo.example.com • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-66 路由到多个服务

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.class: performance  #ELB类型
    kubernetes.io/elb.port: '80'
spec:
  rules:
  - host: 'www.example.com'
    http:
      paths:
      - path: '/foo'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
      - path: '/bar'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  - host: 'foo.example.com'
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
```



```
          port:
            number: 80            #替换为您的目标服务端口
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
      pathType: ImplementationSpecific
  ingressClassName: cce
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.class: performance  #ELB类型
spec:
  rules:
  - host: 'www.example.com'
    http:
      paths:
      - path: '/foo'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
      - path: '/bar'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
  - host: 'foo.example.com'
    http:
      paths:
      - path: '/'
        backend:
          serviceName: <your_service_name>  #替换为您的目标服务名称
          servicePort: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME           CLASS  HOSTS                          ADDRESS         PORTS   AGE
ingress-test   cce    www.example.com,foo.example.com    121.**.**.**    80      10s
```

----结束

7.4.2.4.5 为 ELB Ingress 配置 HTTP/2

Ingress支持HTTP/2的方式暴露服务，在默认情况下，客户端与负载均衡之间采用HTTP1.X协议，若需开启HTTP2功能，可**通过控制台配置**和**通过kubectl命令行配置**。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.13-r0及以上版本
  - v1.25集群：v1.25.8-r0及以上版本
  - v1.28集群：v1.28.3-r0及以上版本
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 已准备可信的证书，您可以从证书提供商处获取证书。操作详情请参见**购买SSL证书**。

约束与限制

- 仅当负载均衡端口使用HTTPS协议时，支持使用HTTP/2功能。
- 配置HTTP/2后，如果您在CCE控制台删除开启HTTP/2的高级配置或在YAML中删除对应的annotation，ELB侧会同时关闭HTTP/2。

Ingress 配置 HTTP/2

您可以使用以下方式配置HTTP/2。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

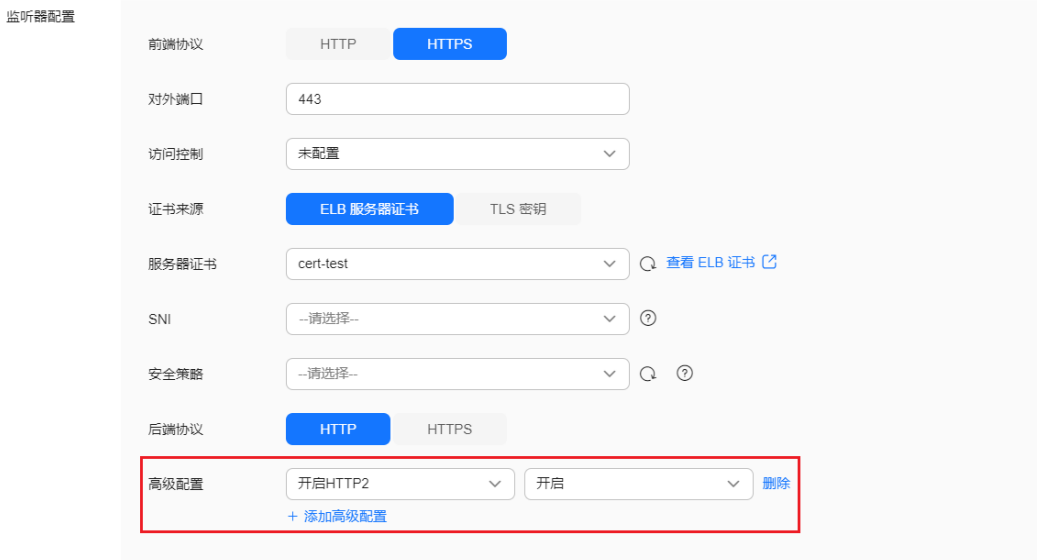
本示例中展示配置HTTP/2的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-107 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。	共享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：选择“HTTPS”。 • 对外端口：ELB监听器端口，HTTPS协议的端口默认为443。 • 证书来源：选择“ELB服务器证书”。 • 服务器证书：使用在ELB服务中创建的证书。如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。 • 高级配置：添加高级配置，选择“开启HTTP2”，状态设置为“开启”。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：“ELB服务器证书” • 服务器证书：cert-test • 高级配置：开启HTTP2
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-67 配置 HTTP/2



步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.ip: <your_elb_ip>  #替换为您已有的ELB IP
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.http2-enable: 'true' # 开启HTTP/2功能
spec:
  tls:
  - secretName: ingress-test-secret
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-108 HTTP/2 参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.http2-enable	否	String	表示HTTP/2功能的开启状态。开启后，可提升客户端与ELB间的访问性能，但ELB与后端服务器间仍采用HTTP1.X协议。 取值范围： ● true：开启HTTP/2功能； ● false：关闭HTTP/2功能（默认为关闭状态）。 注意：**只有当监听器的协议为HTTPS时，才支持开启或关闭HTTP/2功能**。当监听器的协议为HTTP时，该字段无效，默认将其设置为false。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```


回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce     *        121.**.**.**    80,443  10s
```

----结束

7.4.2.4.6 为 ELB Ingress 配置 HTTPS 协议的后端服务

Ingress可以对接不同协议的后端服务，在默认情况下Ingress的后端代理通道是HTTP协议的，若需要建立HTTPS协议的通道，可在annotation字段中加入如下配置：

```
kubernetes.io/elb.pool-protocol: https
```

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.8-r0及以上版本
  - v1.25集群：v1.25.3-r0及以上版本
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 已准备可信的证书，您可以从证书提供商处获取证书。操作详情请参见**购买SSL证书**。

约束与限制

- 仅使用独享型ELB时，Ingress支持对接HTTPS协议的后端服务。
- 对接HTTPS协议的后端服务时，Ingress的对外协议也需要选择HTTPS。

配置 HTTPS 协议的后端服务

您可以使用以下方式为Ingress配置HTTPS协议的后端服务。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置HTTPS协议后端服务的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-109 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：为Ingress配置HTTPS协议的后端服务需选择“HTTPS”。 • 对外端口：ELB监听器的端口，HTTPS协议的端口默认为443。 • 证书来源：选择“ELB服务器证书”。 • 服务器证书：使用在ELB服务中创建的证书。如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。 • 后端协议：选择“HTTPS”。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：ELB服务器证书 • 服务器证书：cert-test • 后端协议：“HTTPS”
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-68 配置 HTTPS 协议的后端服务

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: <your_elb_id>    #本示例中使用已有的独享型ELB，请替换为您的独享型ELB ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.pool-protocol: https  # 对接HTTPS协议的后端服务
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
    - secretName: ingress-test-secret
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: 80
              property:
                ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
              pathType: ImplementationSpecific
  ingressClassName: cce
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```


回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS     ADDRESS         PORTS   AGE
ingress-test  cce      *         121.**.**.**    80,443  10s
```

----结束

7.4.2.4.7 为 ELB Ingress 配置 GRPC 协议的后端服务

Ingress可以对接不同协议的后端服务，在默认情况下Ingress的后端代理通道是HTTP协议的，若需要建立GRPC协议的通道，可在annotation字段中加入如下配置：

```
kubernetes.io/elb.pool-protocol: grpc
```

说明

当前该功能依赖ELB的监听器能力，仅在部分区域开放，请根据当前区域的控制台选项进行确认。ELB已支持的区域请参见**后端服务器组支持GRPC协议**。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.10-r20及以上版本
  - v1.25集群：v1.25.5-r20及以上版本
  - v1.27集群：v1.27.2-r20及以上版本
  - v1.28集群：v1.28.1-r0及以上版本
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。
- 已准备可信的证书，您可以从证书提供商处获取证书。操作详情请参见**购买SSL证书**。

约束与限制

- 仅使用独享型ELB时，Ingress支持对接GRPC协议的后端服务。
- 对接GRPC协议的后端服务时，Ingress的对外协议也需要选择HTTPS，且需要开启HTTP/2。

配置 GRPC 协议的后端服务

您可以使用以下方式为Ingress配置GRPC协议的后端服务。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置GRPC协议后端服务的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-110 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：为Ingress配置GRPC协议的后端服务需选择“HTTPS”。 • 对外端口：ELB监听器的端口，HTTPS协议的端口默认为443。 • 证书来源：选择“ELB服务器证书”。 • 服务器证书：使用在ELB服务中创建的证书。如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。 • 后端协议：选择“GRPC”。 **说明** 仅独享型ELB支持选择GRPC协议，且需开启HTTP/2，系统将自动为您添加**kubernetes.io/elb.http2-enable:true**注解。 当前GRPC功能仅在部分区域开放，请以控制台呈现为准。	• 前端协议：“HTTPS” • 对外端口：443 • 证书来源：ELB服务器证书 • 服务器证书：cert-test • 后端协议：“GRPC”

参数	配置说明	示例
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。 • （可选）负载均衡配置：可设置健康检查协议为GRPC。单击转发策略中的“更改配置”，启用健康检查，并选择GRPC协议。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-69 配置 GRPC 协议的后端服务

图 7-70 设置健康检查协议为 GRPC

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见[通过kubectl连接集群](#)，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '443'
    kubernetes.io/elb.id: <your_elb_id>    #本示例中使用已有的独享型ELB，请替换为您的独享型ELB ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.pool-protocol: grpc  # 对接GRPC协议的后端服务
    kubernetes.io/elb.http2-enable: 'true' # 开启HTTP/2功能
    kubernetes.io/elb.tls-ciphers-policy: tls-1-2
spec:
  tls:
    - secretName: ingress-test-secret
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
```



```
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS     ADDRESS         PORTS   AGE
ingress-test  cce      *         121.**.**.**    80      10s
```

----结束

7.4.2.4.8 为 ELB Ingress 配置超时时间

ELB Ingress支持设置以下超时时间：

- 客户端连接空闲超时时间：没有收到客户端请求的情况下保持连接的最长时间。如果在这个时间内没有新的请求， 负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。
- 等待客户端请求超时时间：如果在规定的时间内客户端没有发送完请求头，或body体数据发送间隔超过一定时间，负载均衡会自动关闭连接。
- 等待后端服务器响应超时时间：向后端服务器发送请求后，如果在一定时间内没有收到响应，负载均衡将返回504错误码。

说明

更新Ingress时，如果删除超时时间配置，已有监听器的超时时间配置会被重置为默认值。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，支持设置超时时间的集群版本如下：

超时时间类型	支持的ELB类型	支持的集群版本
空闲超时时间	独享型	• v1.19集群：v1.19.16-r30及以上版本 • v1.21集群：v1.21.10-r10及以上版本 • v1.23集群：v1.23.8-r10及以上版本 • v1.25集群：v1.25.3-r10及以上版本 • 其他更高版本集群
请求超时时间	独享型	
响应超时时间	独享型	

超时时间类型	支持的ELB类型	支持的集群版本
空闲超时时间	共享型	• v1.23集群：v1.23.13-r0及以上版本 • v1.25集群：v1.25.8-r0及以上版本 • v1.27集群：v1.27.5-r0及以上版本 • v1.28集群：v1.28.3-r0及以上版本 • 其他更高版本的集群
请求超时时间	共享型	
响应超时时间	共享型	

- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

配置超时时间

您可以使用以下方式为Ingress配置超时时间。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，切换至“路由”页签，在右上角单击“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配超时时间的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-111 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。可选择“共享型”或“独享型”。	独享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：支持“HTTP”和“HTTPS”。 • 对外端口：ELB监听器的端口。 • 高级配置： – 空闲超时时间：客户端连接空闲超时时间。在超过空闲超时时间一直没有请求， 负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。 – 请求超时时间：等待客户端请求超时时间。包含以下两种情况： 1.读取整个客户端请求头的超时时长，如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 2.两个连续body体的数据包到达LB的时间间隔，超出请求超时时间将会断开连接。 – 响应超时时间：等待后端服务器响应超时时间。请求转发后端服务器后，在等待超过响应超时时间没有响应，负载均衡将终止等待，并返回 HTTP504错误码。	• 前端协议：“HTTP” • 对外端口：80 • 高级配置： – 空闲超时时间：60 – 请求超时时间：60 – 响应超时时间：60
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-71 配置超时时间

步骤4 单击“确定”，创建Ingress。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.id: <your_elb_id>    #本示例中使用已有的独享型ELB，请替换为您的独享型ELB ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.keepalive_timeout: '300' # 客户端连接空闲超时时间
    kubernetes.io/elb.client_timeout: '60'     # 等待客户端请求超时时间
    kubernetes.io/elb.member_timeout: '60'     # 等待后端服务器响应超时时间
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: test
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-112 annotation 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.keepalive_timeout	否	String	客户端连接空闲超时时间，在超过keepalive_timeout时长一直没有请求，负载均衡会暂时中断当前连接，直到下一次请求时重新建立新的连接。 取值范围为0-4000s，默认值为60s。
kubernetes.io/elb.client_timeout	否	String	等待客户端请求超时时间，包括两种情况： ● 读取整个客户端请求头的超时时长：如果客户端未在超时时长内发送完整个请求头，则请求将被中断。 ● 两个连续body体的数据包到达LB的时间间隔，超出client_timeout将会断开连接。 取值范围为1-300s，默认值为60s。
kubernetes.io/elb.member_timeout	否	String	等待后端服务器响应超时时间。请求转发后端服务器后，等待超过member_timeout时长没有响应，负载均衡将终止等待，并返回 HTTP504错误码。 取值范围为1-300s，默认值为60s。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.9 为 ELB Ingress 配置慢启动持续时间

慢启动指负载均衡器向组内新增的后端服务器Pod线性增加请求分配权重，直到配置的慢启动时间结束，负载均衡器向后端服务器Pod正常发送完请求的启动模式。慢启动能够实现业务的平滑启动，完美避免业务抖动问题。

说明

配置慢启动持续时间后，如果您在YAML中删除对应的annotation，将不启用慢启动。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本为v1.23及以上。
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

- 仅独享型负载均衡支持HTTP和HTTPS类型的后端服务器组Pod开启慢启动功能。
- 仅在流量分配策略使用加权轮询算法时生效。
- 慢启动仅对新增后端服务器Pod生效，后端服务器组Pod首次添加后端服务器慢启动不生效。
- 后端服务器的慢启动结束之后，不会再次进入慢启动模式。
- 在健康检查开启时，后端服务器Pod在线后慢启动生效。
- 在健康检查关闭时，慢启动立即生效。
- 在配置慢启动后，该Ingress下的所有转发策略都会生效。

设置慢启动持续时间

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.id: <your_elb_id>  #替换为您已有的ELB ID
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.slowstart: '30' #设置慢启动持续时间
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: 80             #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-113 慢启动参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.slowstart	否	String	负载均衡器向慢启动模式下的后端服务器Pod线性增加请求分配权重，当配置的慢启动持续时间期限结束后，负载均衡器向后端服务器Pod发送完整的请求比例，此后本次添加的后端服务器Pod退出慢启动模式。 **v1.23以上版本的集群支持此字段**。 取值范围：30-1200。 参数说明：慢启动持续时间，单位秒。 ● 独享型负载均衡器生效。 ● 目标服务分配策略类型为加权轮询算法且不开启会话保持时生效。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce     *        121.**.**.**    80      10s
```

----结束

7.4.2.4.10 为 ELB Ingress 配置灰度发布

独享型ELB Ingress支持使用以下方式设置灰度发布：

- 支持按比例的方式发布灰度Ingress
- 支持按HTTP请求头的方式发布灰度Ingress
- 支持按Cookie发布的方式发布灰度Ingress

说明

Ingress的灰度发布功能依赖ELB能力，使用该功能前请提交工单申请开通ELB灰度发布能力。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0 及以上版本
  - v1.25集群：v1.25.9-r0 及以上版本
  - v1.27集群：v1.27.6-r0 及以上版本
  - v1.28集群：v1.28.4-r0 及以上版本

- 其他更高版本的集群

约束与限制

- 创建灰度Ingress后，不应删除原Ingress。
- 单个ELB下的监听器，如果关联的多个Ingress配置了多个灰度策略，按HTTP请求头的灰度策略优先级最高，按Cookie的灰度策略次之，按比例的灰度策略优先级最低。

通过控制台配置灰度发布服务

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 部署原服务。

1. 在左侧导航栏中选择“工作负载”，单击右上角“创建工作负载”，部署名为origin-server的工作负载。配置参数说明请参见**创建无状态负载（Deployment）**。
2. 在左侧导航栏中选择“服务”，切换至“服务”页签，单击右上角“创建服务”，创建名为origin-service的Service，并关联刚创建的origin-server工作负载（CCE Standard集群创建nodeport类型Service，Turbo集群创建ClusterIP类型Service）。
3. 在左侧导航栏中选择“服务”，切换至“路由”页签，单击右上角“创建路由”，创建名为origin-ingress的Ingress，并关联刚创建的origin-service服务。参数配置说明请参见**通过控制台创建ELB Ingress**。

步骤3 灰度发布新版本服务。

1. 在左侧导航栏中选择“服务”，切换至“路由”页签，选择需要设置灰度发布的路由，在操作列单击“更多>创建灰度发布”。
2. 设置灰度发布参数。

表 7-114 关键参数说明

参数	配置说明	示例

灰度发布	- 按HTTP请求头：当HTTP请求的header键值对匹配时，访问灰度发布服务。
	  ■ 如果Ingress转发策略配置了域名和路径，最多支持配置8个Values值。
	  ■ 如果Ingress转发策略仅配置了路径，最多支持配置9个Values值。
	- 按Cookie发布：当请求的Cookie键值对匹配时，访问灰度发布服务。
	- 按比例发布：按访问灰度发布服务的请求比例。
		按HTTP请求头
		- 键：a
		- 值：b

灰度转发策略配置	- 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 - 开启灰度：开启灰度后，集群外部访问流量将按此规则转发至目标服务。 - 灰度规则：支持按HTTP请求头、按Cookie发布、按比例（灰度服务请求比例0%），请按需选择。 - 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 - 目标服务访问端口：可选择目标Service的访问端口。	- 域名：无需填写 - 开启灰度：开启 - 灰度规则：按HTTP请求头 - 目标服务名称：nginx - 目标服务访问端口：80

3. 单击“确定”，创建灰度发布服务。
4. 查看灰度发布新版本服务是否成功。

步骤4 老版本服务下线。

新版本服务运行稳定后，需要下线老版本服务。下线流程将原服务Ingress中的Service修改为新版本服务的Service，使流量都路由到新版本服务上，然后删除灰度Ingress。

----结束

通过 kubectl 命令行配置灰度发布服务

步骤1 部署原服务。

1. 部署名为origin-server的工作负载。

2. 部署名为origin-service的service，关联刚创建的origin-server工作负载。（CCE Standard集群创建nodeport类型Service，Turbo集群创建ClusterIP类型Service）
3. 部署名为origin-ingress的Ingress，关联刚创建的origin-service服务。

   Ingress的配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: origin-ingress
  namespace: default
  annotations:
    kubernetes.io/elb.port: '81'
    kubernetes.io/elb.id: e491f4e7-2351-4984-ad0a-8569e5e964a3
    kubernetes.io/elb.class: performance
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            backend:
              service:
                name: origin-service
                port:
                  number: 81
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```

步骤2 灰度发布新版本服务。

部署新版本工作负载、服务和Ingress，使得流量能够通过权重或者头域导流到新版本服务上。

1. 部署名为canary-server的工作负载。
2. 部署名为canary-service的服务，关联刚创建的canary-server工作负载。（CCE Standard集群创建nodeport类型Service，Turbo集群创建ClusterIP类型Service）
3. 创建灰度发布的Ingress。关于灰度发布的参数说明请参见**参数说明**。

   a. **场景一**：按HTTP请求头，表示当HTTP请求的header键值对匹配时，访问灰度发布服务。

      部署名为canary-header-ingress的Ingress，关联刚创建的canary-service服务，实现Header灰度发布。

      Ingress的配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: canary-header-ingress
  namespace: default
  annotations:
    kubernetes.io/elb.canary: 'true'                          #设置为true，表示启用canary注解功能
    kubernetes.io/elb.canary-by-header: 'location'            # header的key值设置为location
    kubernetes.io/elb.canary-by-header-value: '{"values":["hz","sz","sh"]}'  #header的value值设置为hz、sz、sh
    kubernetes.io/elb.class: performance          #仅独享型ELB支持灰度发布
    kubernetes.io/elb.id: e491f4e7-2351-4984-ad0a-8569e5e964a3
    kubernetes.io/elb.port: '81'
spec:
  ingressClassName: cce
  rules:
    - host: example.com
      http:
        paths:
```



```
        - path: /
          pathType: ImplementationSpecific
          backend:
            service:
              name: canary-service
              port:
                number: 80
          property:
            ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

b. **场景二**：按比例，即设置访问灰度发布服务的请求比例。

部署名为canary-weight-ingress的Ingress，关联刚创建的canary-service服务，实现权重灰度发布。

Ingress的配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: canary-weight-ingress
  namespace: default
  annotations:
    kubernetes.io/elb.canary: 'true'              #设置为true，表示启用canary注解功能
    kubernetes.io/elb.canary-weight: '40'         #权重值设置，表示对应请求40%的流量将被导流到新版本服务上
    kubernetes.io/elb.class: performance          #仅独享型ELB支持灰度发布
    kubernetes.io/elb.id: e491f4e7-2351-4984-ad0a-8569e5e964a3
    kubernetes.io/elb.port: '81'
spec:
  ingressClassName: cce
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: ImplementationSpecific
            backend:
              service:
                name: canary-service
                port:
                  number: 81
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

c. **场景三**：按Cookie，表示当请求的Cookie键值对匹配时，访问灰度发布服务。

部署名为canary-cookie-ingress的Ingress，关联刚创建的canary-service服务，实现按Cookie灰度发布。

Ingress的配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: canary-cookie-ingress
  namespace: default
  annotations:
    kubernetes.io/elb.canary: 'true'                    #设置为true，表示启用canary注解功能
    kubernetes.io/elb.canary-by-cookie: 'location'      # header的key值设置为location
    kubernetes.io/elb.canary-by-cookie-value: '{"values":["hz","sz","sh"]}'  // header的value值设置为hz、sz、sh
    kubernetes.io/elb.class: performance                #仅独享型ELB支持灰度发布
    kubernetes.io/elb.id: e491f4e7-2351-4984-ad0a-8569e5e964a3
    kubernetes.io/elb.port: '81'
spec:
  ingressClassName: cce
  rules:
    - host: example.com
      http:
        paths:
          - path: /
```



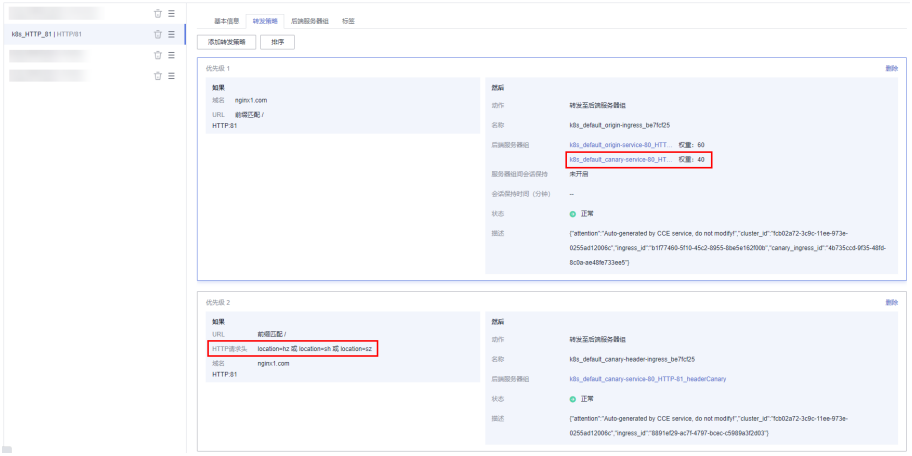
```
      pathType: ImplementationSpecific
      backend:
        service:
          name: canary-service
          port:
            number: 81
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

4. 查看灰度发布新版本服务是否成功。

   a. 查看Ingress创建：

   ```
   $ kubectl get ingress
   NAME                     CLASS    HOSTS         ADDRESS         PORTS   AGE
   canary-cookie-ingress    cce      example.com   88.88.88.165    80      16s
   canary-header-ingress    cce      example.com   88.88.88.165    80      46s
   canary-weight-ingress    cce      example.com   88.88.88.165    80      117s
   origin-ingress           cce      example.com   88.88.88.165    80      2m25s
   ```

   b. 查看ELB生成规则：

   c. 服务请求返回结果：

      i. 使用location：hz请求头的请求访问服务：

      ```
      $ curl -H "Host: example.com" -H "location: hz" http://88.88.88.165:81/
      ```

      预期结果：

      ```
      $ new
      ```

      多次访问，返回结果均为new。

      ii. 不使用请求头的请求访问服务：

      ```
      $ curl -H "Host: example.com" http://88.88.88.165:81/
      ```

      预期结果：返回值40%为new，60%为old。

步骤3 老版本服务下线。

新版本服务运行稳定后，需要下线老版本服务。下线流程将原服务Ingress中的Service修改为新版本服务的Service，使流量都路由到新版本服务上，然后删除灰度Ingress。

1. 将原服务Ingress关联的Service更新为新版本Service。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: origin-ingress
  namespace: default
  annotations:
    kubernetes.io/elb.port: '81'
    kubernetes.io/elb.id: e491f4e7-2351-4984-ad0a-8569e5e964a3
    kubernetes.io/elb.class: performance
spec:
  rules:
```



```
  - host: example.com
    http:
      paths:
        - path: /
          backend:
            service:
              name: canary-service
              port:
                number: 81
          property:
            ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
          pathType: ImplementationSpecific
ingressClassName: cce
```

2. 验证老版本服务是否下线完成。
   a. 使用带header和不带header的请求，多次访问。
      ```
      $ curl -H "Host: example.com" -H "location: hz" http://88.88.88.165:81/
      $ curl -H "Host: example.com" http://88.88.88.165:81/
      ```
   b. 预期结果：输出均为new，无old返回。
3. 确认老版本服务下线完成后，删除灰度发布Ingress。
   ```
   $ kubectl delete ingress canary-weight-ingress canary-header-ingress
   ```

----结束

参数说明

表 7-115 灰度发布参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.canary	否	string	设置Ingress灰度发布的开关。设置为true后，配合不同的注解，可以实现不同的灰度发布功能。 取值范围：true ● 独享型负载均衡器生效。 ● 设置为true后，不允许删除或修改。
kubernetes.io/elb.canary-weight	否	string	权重灰度发布权重值，设置后Ingress以权重灰度形式发布。 ● 取值为0-100的正整数，为灰度流量分配的百分比。 ● 发布成权重灰度Ingress时，该参数必填。 ● 不能与其他灰度发布功能同时设置。
kubernetes.io/elb.session-affinity-mode	否	string	开启权重灰度发布后，配置会话保持能力。 灰度发布仅支持设置为 "HTTP_COOKIE"。

参数	是否必填	参数类型	描述
kubernetes.io/elb.session-affinity-option	否	string	开启权重灰度发布会话保持能力后，会话保持的超时时间。 参数值为json字符串，格式如下： `{"persistence_timeout": "1440"}` 参数说明： ● 超时时间范围为1-1440。 ● 默认值为1440。
kubernetes.io/elb.canary-by-header	否	string	header灰度发布的Key值，表示请求头参数的名称。需要与kubernetes.io/elb.canary-by-header-value成对使用。 参数说明： 长度限制1-40字符，只允许包含字母、数字、中划线（-）和下划线（_）。
kubernetes.io/elb.canary-by-header-value	否	string	header灰度发布的Values值，需要与kubernetes.io/elb.canary-by-header成对使用。 参数值为json格式的数组，例如： `'{"values":["a","b"]}'` 参数说明： ● Values数组长度：至少需要配置1个Values值。 – 如果Ingress转发策略配置了域名和路径，最多支持配置8个Values值。 – 如果Ingress转发策略仅配置了路径，最多支持配置9个Values值。 ● Values数组取值：长度限制1-128字符，不支持空格，双引号，支持以下通配符：*（匹配0个或更多字符）和?（正好匹配1个字符）。
kubernetes.io/elb.canary-by-cookie	否	string	cookie灰度发布的key值，表示请求cookie参数的名称。需要与kubernetes.io/elb.canary-by-cookie-value成对使用。 参数说明： 长度限制1-100字符，支持包含字母、数字、以及 !%'"()*+,./:=?@^\\-_`~ 等字符。

参数	是否必填	参数类型	描述
kubernetes.io/elb.canary-by-cookie-value	否	string	cookie灰度发布的values值，需要与kubernetes.io/elb.canary-by-cookie成对使用。 参数值为json格式的数组，例如： `'{"values":["a","b"]}'` 参数说明： ● Values数组长度：至少需要配置1个Values值。 – 如果Ingress转发策略配置了域名和路径，最多支持配置8个Values值。 – 如果Ingress转发策略仅配置了路径，最多支持配置9个Values值。 ● Values数组取值：长度限制1-100字符，不支持空格，支持包含字母、数字、以及！%'"()*+,./:=?@^\\-_`~ 等字符。
kubernetes.io/elb.canary-related-ingress-uid	否	string	灰度发布Ingress关联的原始Ingress的uid信息，用于前端展示原始Ingress和灰度发布的Ingress的关联关系。 ● 参数格式：字符串 ● 取值：原始Ingress的metadata.uid字段

7.4.2.4.11 为 ELB Ingress 配置黑名单/白名单访问策略

使用ELB Ingress时，您可以通过添加白名单和黑名单的方式控制访问负载均衡监听器的IP。

- 白名单：指定的IP允许访问，而其它IP不能访问。
- 黑名单：指定的IP不能访问，而其它IP允许访问。

说明

配置黑名单/白名单访问策略后，如果您在CCE控制台删除黑名单/白名单访问策略配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  – v1.23集群：v1.23.12-r0及以上版本
  – v1.25集群：v1.25.7-r0及以上版本
  – v1.27集群：v1.27.4-r0及以上版本
  – v1.28集群：v1.28.2-r0及以上版本
  – 其他更高版本的集群
- 已在ELB控制台创建一个IP地址组，详情请参见**创建IP地址组**。

配置黑名单/白名单访问策略

您可以使用以下方式为Ingress配置黑名单/白名单访问策略。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置黑名单/白名单访问策略的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-116 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。可选择“共享型”或“独享型”。	独享型ELB
监听器配置	• 前端协议：支持“HTTP”和“HTTPS”。 • 对外端口：ELB监听器的端口。 • 访问控制： – 继承ELB已有配置：CCE不对ELB侧已有的访问控制进行修改。 – 允许所有IP访问：不设置访问控制。 – 白名单：仅所选IP地址组可以访问ELB地址。 – 黑名单：所选IP地址组无法访问ELB地址。	• 前端协议：“HTTP” • 对外端口：80 • 高级配置：访问控制：黑名单
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-72 配置黑名单/白名单访问策略

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB创建Ingress的场景为例，YAML配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                 # ELB ID，替换为实际值
    kubernetes.io/elb.class: performance                # 负载均衡器类型
    kubernetes.io/elb.port: '80'                        # 负载均衡监听器的对外端口
    kubernetes.io/elb.acl-id: <your_acl_id>             # ELB的IP地址组ID
    kubernetes.io/elb.acl-status: 'on'                  # 开启访问控制
    kubernetes.io/elb.acl-type: 'white'                 # 白名单控制
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-117 ELB 访问控制注解

参数	类型	描述
kubernetes.io/elb.acl-id	String	• 不填写该参数时：表示CCE不对ELB侧访问控制进行修改。 • 参数值填写为空值时：表示允许所有IP访问。 • 参数值填写为ELB的IP地址组ID时：表示开启访问控制，为ELB设置IP地址黑名单或白名单。此时需同时填写kubernetes.io/elb.acl-status和kubernetes.io/elb.acl-type参数。 **获取方法：** 登录控制台后，单击顶部菜单右侧的“网络 > 弹性负载均衡ELB”，在网络控制台中单击“弹性负载均衡 > IP地址组”，复制目标IP地址组的“ID”即可。详情请参见**IP地址组**。
kubernetes.io/elb.acl-status	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • on：表示开启访问控制。 • off：表示不开启访问控制。
kubernetes.io/elb.acl-type	String	为ELB设置IP地址黑名单或白名单时需填写，取值如下： • black：表示黑名单，所选IP地址组无法访问ELB地址。 • white：表示白名单，仅所选IP地址组可以访问ELB地址。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.12 为 ELB Ingress 配置多个监听端口

Ingress支持配置自定义监听端口，可为同一个服务配置HTTP和HTTPS协议的监听器，例如一个服务可以同时暴露HTTP协议的80端口和HTTPS的443端口对外提供访问。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上
  - v1.25集群：v1.25.9-r0及以上
  - v1.27集群：v1.27.6-r0及以上
  - v1.28集群：v1.28.4-r0及以上
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB为例，配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.id: 2c623150-17bf-45f1-ae6f-384b036f547e     # 已有ELB的ID
    kubernetes.io/elb.class: performance     # ELB的类型
    kubernetes.io/elb.listen-ports: '[{"HTTP": 80},{"HTTPS": 443}]'     # 多监听器配置
    kubernetes.io/elb.tls-certificate-ids:
6cfb43c9de1a41a18478b868e34b0a82,6cfb43c9de1a41a18478b868e34b0a82   # HTTPS证书配置
  name: ingress-test
  namespace: default
spec:
  ingressClassName: cce
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: test
            port:
              number: 8888
        path: /
        pathType: ImplementationSpecific
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```


表 7-118 自定义监听端口注解

参数	类型	描述
kubernetes.io/elb.listen-ports	String	为同一个Ingress创建多个监听端口，端口号范围为1~65535。 参数值为JSON格式的字符串，示例如下： `kubernetes.io/elb.listen-ports: '[{"HTTP":80}, {"HTTPS":443}]'` ● 仅支持同时配置HTTP和HTTPS协议的监听端口。 ● v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0以下集群版本中仅支持新建Ingress场景，且配置多个监听端口后annotation不支持修改和删除。v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上集群版本中支持修改和删除。 ● 同时指定多监听器（kubernetes.io/elb.listen-ports）和单监听器（kubernetes.io/elb.port）配置时，多监听器优先级更高。 ● Ingress内配置项对多个监听器同时生效，如黑白名单配置、超时时间配置。Ingress开启HTTP/2配置时，HTTP/2仅在HTTPS端口生效。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS         ADDRESS         PORTS   AGE
ingress-test  cce      example.com   121.**.**.**    80,443  10s
```

----结束

7.4.2.4.13 为 ELB Ingress 配置 HTTP/HTTPS 头字段

HTTP头字段是指在超文本传输协议（HTTP）的请求和响应消息中的消息头部分。HTTP头部字段可以根据需要自定义，本文介绍可通过HTTP和HTTPS监听器支持的非标准头字段实现的功能特性。

说明

- 配置HTTP/HTTPS头字段依赖ELB能力，使用该功能前请确认当前区域是否支持使用HTTP/HTTPS头字段，详情请参见**HTTP/HTTPS头字段**。
- 配置HTTP/HTTPS头字段后，如果您在CCE控制台删除HTTP/HTTPS头字段配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

表 7-119 头字段开关

头字段	功能开关	功能说明
X-Forwarded-Port	获取监听器端口号	开启获取监听器端口号开关，ELB可通过X-Forwarded-Port头字段获取监听器的端口号，传输到后端服务器的报文中。
X-Forwarded-For-Port	获取客户端请求端口号	开启获取客户端请求端口号开关，ELB可通过X-Forwarded-For-Port头字段获取客户端请求的端口号，传输到后端服务器的报文中。
X-Forwarded-Host	重写X-Forwarded-Host	开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.13-r0及以上版本
  - v1.25集群：v1.25.8-r0及以上版本
  - v1.27集群：v1.27.5-r0及以上版本
  - v1.28集群：v1.28.3-r0及以上版本
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

配置 HTTP/HTTPS 头字段

您可以使用以下方式为Ingress配置HTTP/HTTPS头字段。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，切换至“路由”页签，在右上角单击“创建路由”。

步骤3 设置Ingress参数。

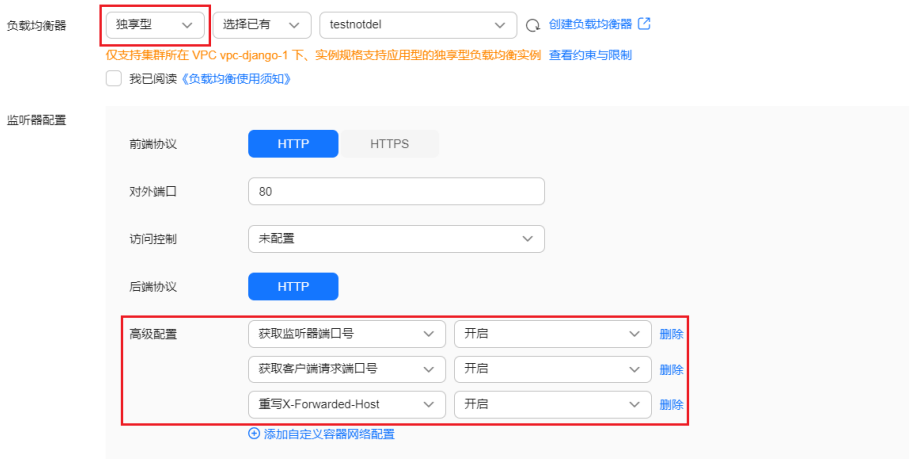
说明

本示例中展示配置HTTP/HTTPS头字段的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-120 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：支持“HTTP”和“HTTPS”。 • 对外端口：ELB监听器的端口。 • 高级配置： – 获取监听器端口号：开启后可以将ELB实例的监听端口从报文的HTTP头中带到后端云服务器。 – 获取客户端请求端口号：开启后可以将客户端的源端口从报文的HTTP头中带到后端云服务器。 – 重写X-Forwarded-Host：开启后将以客户端请求头的Host重写X-Forwarded-Host传递到后端云服务器。	• 前端协议：“HTTP” • 对外端口：80 • 高级配置： – 获取监听器端口号：开启 – 获取客户端请求端口号：开启 – 重写X-Forwarded-Host：开启
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-73 配置 HTTP/HTTPS 头字段



步骤4 单击“确定”，创建Ingress。

----结束

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>                #替换为您已有的ELB ID
    kubernetes.io/elb.class: performance               #ELB类型
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.x-forwarded-port: 'true'         # 获取监听器端口号
    kubernetes.io/elb.x-forwarded-for-port: 'true'     # 获取客户端请求端口号
    kubernetes.io/elb.x-forwarded-host: 'true'         # 重写X-Forwarded-Host
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 80               #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-121 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.x-forwarded-port	String	ELB可通过X-Forwarded-Port头字段获取监听器的端口号，传输到后端服务器的报文中。 ● true：开启获取监听器端口号开关。 ● false：关闭获取监听器端口号开关。
kubernetes.io/elb.x-forwarded-for-port	String	ELB可通过X-Forwarded-For-Port头字段获取客户端请求的端口号，传输到后端服务器的报文中。 ● true：开启获取客户端请求端口号开关。 ● false：关闭获取客户端请求端口号开关。
kubernetes.io/elb.x-forwarded-host	String	● true：开启重写X-Forwarded-Host开关，ELB以客户端请求头的Host重写X-Forwarded-Host传递到后端服务器。 ● false：关闭重写X-Forwarded-Host开关，ELB透传客户端的X-Forwarded-Host到后端服务器。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.14 为 ELB Ingress 配置 gzip 数据压缩

ELB支持开启数据压缩，通过数据压缩可缩小传输文件大小，提升文件传输效率减少带宽消耗。

说明

- 该功能依赖ELB能力，使用该功能前请确认当前区域是否支持。ELB已支持的区域请参见**数据压缩**。
- 配置数据压缩后，如果您在CCE控制台删除数据压缩配置或在YAML中删除对应的annotation，ELB侧的配置将会保留。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：

  - v1.23集群：v1.23.14-r0及以上版本
  - v1.25集群：v1.25.9-r0及以上版本
  - v1.27集群：v1.27.6-r0及以上版本
  - v1.28集群：v1.28.4-r0及以上版本
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

仅使用独享型ELB时，Ingress支持配置gzip数据压缩。

配置 gzip 数据压缩

您可以使用以下方式为Ingress配置gzip数据压缩。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“服务”，切换至“路由”页签，在右上角单击“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置gzip数据压缩的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-122 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB

参数	配置说明	示例
监听器配置	• 前端协议：支持“HTTP”和“HTTPS”。 • 对外端口：ELB监听器的端口。 • 高级配置： 数据压缩：开启将对特定文件类型进行压缩；关闭则不会对任何文件类型进行压缩。 **说明** – Brotli支持压缩所有类型。 – Gzip支持压缩的类型如下： text/xml text/plain text/css application/javascript application/x-javascript application/rss+xml application/atom+xml application/xml application/json。	• 前端协议：“HTTP” • 对外端口：80 • 高级配置：数据压缩：开启
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-74 配置 gzip 数据压缩

步骤4 单击“确定”，创建Ingress。

----结束

通过 kubectl 命令行配置

步骤1 请参见通过kubectl连接集群，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML文件配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/elb.id: <your_elb_id>               #替换为您已有的ELB ID
    kubernetes.io/elb.class: performance               #ELB类型
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.gzip-enabled: 'true'            # 开启数据压缩
spec:
  rules:
  - host: ''
    http:
      paths:
      - path: '/'
        backend:
          service:
            name: <your_service_name>  #替换为您的目标服务名称
            port:
              number: 8080            #替换为您的目标服务端口
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-123 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.gzip-enabled	String	• true：开启，将对特定文件类型进行压缩。 • false：关闭，不会对任何文件类型进行压缩。在默认情况下数据压缩为关闭。 支持的压缩类型如下： • Brotli支持压缩所有类型。 • Gzip支持压缩的类型包括：text/xml text/plain text/css application/javascript application/x-javascript application/rss +xml application/atom+xml application/xml application/json。 仅独享型ELB的HTTP/HTTPS类型监听器支持配置。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。


```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS     ADDRESS         PORTS   AGE
ingress-test  cce      *         121.**.**.**    80      10s
```

----结束

7.4.2.4.15 为 ELB Ingress 配置 URL 重定向

Ingress支持将特定的访问请求重定向至指定的路径。配置Ingress重定向规则的YAML示例如下，本示例将访问example.com的请求重定向至指定路径example.com/testa，并返回301状态码。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上
  - v1.25集群：v1.25.9-r0及以上
  - v1.27集群：v1.27.6-r0及以上
  - v1.28集群：v1.28.4-r0及以上
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

仅使用独享型ELB时，Ingress支持配置URL重定向。

配置 Ingress 重定向至 URL 规则

您可以使用以下方式配置Ingress重定向至URL规则。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置URL重定向的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-124 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：可选择“HTTP”或“HTTPS”。 • 对外端口：ELB监听器的端口。	• 前端协议：“HTTP” • 对外端口：80
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。 • 执行动作 – 动作：选择“重定向至URL”，当访问请求满足转发策略时，会将请求重定向至指定的URL，并返回指定的状态码。 – URL：以 "http://" 或 "https://" 开头的合法的URL。 **说明** HTTP协议的路由，支持重定向到HTTP/HTTPS协议；HTTPS协议的路由，只能重定向到HTTPS协议。 – 返回码：支持返回码包括"301"、"302"、"303"、"307"、"308"。	• 域名：example.com • 路径匹配规则：前缀匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80 • 执行动作： – 动作：重定向至URL – URL：https://example.com/testa – 返回码：301

图 7-75 配置重定向至 URL 规则

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

Ingress重定向至URL可通过注解实现，示例如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-redirect-url
  namespace: default
  annotations:
    kubernetes.io/elb.id: df76342f-e898-402a-bac8-bde5bf974da8
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.redirect-url: https://example.com/testa       # 重定向URL的信息
    kubernetes.io/elb.redirect-url-code: '301'     # 重定向URL后的返回码
spec:
  rules:
    - host: "example.com"
      http:
        paths:
          - path: /
            backend:
              service:
                name: test-service
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-125 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.redirect-url	是	string	重定向URL信息。 格式说明：以 "http://" 或 "https://" 开头的合法的URL，如 https://example.com/。 参数说明：对单个Ingress下所有的转发规则均生效，配置删除后自动清理对应的重定向URL规则。 该注解不能和灰度发布的注解一起配置。
kubernetes.io/elb.redirect-url-code	否	string	重定向URL后的返回码。 格式说明：支持返回码包括"301"、"302"、"303"、"307"、"308"。 参数说明：默认值为"301"。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/test-redirect-url created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME              CLASS    HOSTS         ADDRESS         PORTS   AGE
test-redirect-url  cce      example.com   121.**.**.**    80      10s
```

步骤5 使用curl测试重定向后的能力，其中${ELB_IP}为Ingress访问的IP。

```
curl -I -H "Host:example.com" ${ELB_IP}
```

最终访问路径会被重定向至example.com/testa。

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 07 Mar 2024 11:04:31 GMT
Content-Type: text/html
Content-Length: 134
Connection: keep-alive
Location: https://example.com/testa
Server: elb
```

----结束

7.4.2.4.16 为 ELB Ingress 配置 Rewrite 重写

独享型ELB的Ingress支持为正则匹配的URL配置Rewrite重写的能力。规则如下：

- 为Ingress配置一个path为正则匹配的URL，如 /first/(.*)/(.*)/end
- 配置Rewrite重写的注解，匹配path中的正则表达式

例如：

- path 配置为 /first/(.*)/(.*)/end，注解配置为 /$1/$2。当客户发送的请求为 /first/aaa/bbb/end 时，path会匹配到 /first/(.*)/(.*)/end，重写规则会把 $1 替换成 aaa，$2替换成 bbb，最终后端服务接收到的请求路径为 /aaa/bbb。
- path配置为 /first/(.*)/end，注解配置为 /newpath/$1。当客户发送的请求为 /first/aaa/end 时，path会匹配到 /first/(.*)/end，重写规则会把 $1 替换成 aaa，最终后端服务接收到的请求路径为 /newpath/aaa。

说明

该功能依赖ELB能力，当前正陆续上线中，使用该功能前请根据当前区域的控制台选项确认是否支持。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上
  - v1.25集群：v1.25.9-r0及以上
  - v1.27集群：v1.27.6-r0及以上
  - v1.28集群：v1.28.4-r0及以上
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

仅使用独享型ELB时，Ingress支持配置Rewrite重写。

为 Ingress 配置 Rewrite 重写

您可以使用以下方式为Ingress配置Rewrite重写。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置Rewrite重写的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-126 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test

参数	配置说明	示例
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：可选择“HTTP”或“HTTPS”。 • 对外端口：ELB监听器的端口。	• 前端协议：“HTTP” • 对外端口：80
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：使用重写URL需要选择“正则匹配”。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。 • 执行动作 – 动作：选择“重写URL”，当访问请求满足转发策略时，会根据匹配规则重写URL路径。支持使用正则表达式匹配的结果作为重写路径，您需要选择路径匹配规则为“正则匹配”。 – 路径：以 "/" 开头的合理的正则匹配规则。	• 域名：example.com • 路径匹配规则：正则匹配 • 路径：/first/(.*)/(.*)/end • 目标服务名称：nginx • 目标服务访问端口：80 • 执行动作： – 动作：重写URL – 路径：/$1/$2

图 7-76 配置重写 URL 规则

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

Ingress重写URL规则可通过注解实现，示例如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-rewrite-url
  namespace: default
  annotations:
    kubernetes.io/elb.id: df76342f-e898-402a-bac8-bde5bf974da8
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.rewrite-target: /$1/$2       # 重写路径的配置
spec:
  rules:
    - host: "example.com"
      http:
        paths:
          - path: /first/(.*)/(.*)/end      # 该path会被重写
            backend:
              service:
                name: test-service
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: REGEX
            pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-127 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.rewrite-target	是	string	重写路径的信息。 格式说明：以 "/" 开头的合理的正则匹配规则。 参数说明：对单个Ingress下正则匹配的URL转发规则生效，配置删除后自动清理对应的重写规则。 该注解不能和灰度发布的注解一起配置。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/test-rewrite-url created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。


```
NAME              CLASS    HOSTS    ADDRESS         PORTS   AGE
test-rewrite-url  cce      *        121.**.**.**    80      10s
```

步骤5 使用curl测试重写的能力，其中${ELB_IP}为Ingress访问的IP。

```
# curl -H "Host:example.com" ${ELB_IP}/first/aaa/bbb/end
```

最终访问路径会被重写为 /aaa/bbb。

----结束

7.4.2.4.17 为 ELB Ingress 配置 HTTP 重定向到 HTTPS

Ingress支持将HTTP协议的访问请求转发至HTTPS协议的监听器上。配置Ingress重定向到重定向到HTTPS的示例如下，本示例将访问example.com/test的HTTP请求重定向至HTTPS 443端口。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.14-r0及以上
  - v1.25集群：v1.25.9-r0及以上
  - v1.27集群：v1.27.6-r0及以上
  - v1.28集群：v1.28.4-r0及以上
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

仅使用独享型ELB时，Ingress支持配置HTTP重定向到HTTPS。

配置 HTTP 重定向到 HTTPS

您可以使用以下方式配置HTTP重定向到HTTPS。

通过控制台配置

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

说明

本示例中展示配置HTTP重定向到HTTPS的关键参数，其余参数可按需配置，详情请参见**通过控制台创建ELB Ingress**。

表 7-128 关键参数说明

参数	配置说明	示例
名称	自定义Ingress名称。	ingress-test
负载均衡器	选择对接的ELB实例或自动创建ELB实例。本例中仅支持选择“独享型”。	独享型ELB
监听器配置	• 前端协议：可选择“HTTP”或“HTTPS”。 • 对外端口：ELB监听器的端口。 **说明** v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0以下集群版本中配置后不支持修改。v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上集群版本中支持修改。 • 重定向至HTTPS：开启后进行HTTPS端口配置。 – 对外端口：HTTPS协议的端口。 – 证书来源：支持TLS密钥和ELB服务器证书。 – 服务器证书：使用在ELB服务中创建的证书。 如果您没有可选择的ELB证书，可前往ELB服务创建，详情请参见**创建证书**。	• 前端协议：“HTTP” • 对外端口：80 • 重定向至HTTPS：开启 – 对外端口：443 – 证书来源：ELB服务器证书 – 服务器证书：cert-test
转发策略配置	• 域名：实际访问的域名地址，不配置时可通过IP地址访问Ingress。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。 • 路径匹配规则：支持前缀匹配、精确匹配、正则匹配，请按需选择。 • 路径：后端应用对外提供访问的路径，此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。 • 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。 • 目标服务访问端口：可选择目标Service的访问端口。	• 域名：无需填写 • 路径匹配规则：正则匹配 • 路径：/ • 目标服务名称：nginx • 目标服务访问端口：80

图 7-77 配置 HTTP 重定向到 HTTPS

图 7-78 重定向至 HTTPS 的端口配置

步骤4 配置完成后，单击“确定”。

----结束

通过 kubectl 命令行配置

Ingress重定向到HTTPS监听器可通过注解实现，示例如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以关联已有ELB为例，YAML配置文件如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-redirect-listener
  namespace: default
  annotations:
    kubernetes.io/elb.id: df76342f-e898-402a-bac8-bde5bf974da8
    kubernetes.io/elb.class: performance
```



```
    kubernetes.io/elb.listen-ports: '[{"HTTP":80},{"HTTPS":443}]'          # 多端口配置
    kubernetes.io/elb.ssl-redirect: 'true'       # 开启HTTP重定向到HTTPS
    kubernetes.io/elb.tls-certificate-ids: 6cfb43c9de1a41a18478b868e34b0a82      # HTTPS监听器服务端证书
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: test-service
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-129 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/elb.listen-ports	是	string	多端口监听配置。v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0以下集群版本中配置后不支持修改。v1.23.18-r10、v1.25.16-r0、v1.27.16-r0、v1.28.13-r0、v1.29.8-r0、v1.30.4-r0及以上集群版本中支持修改。 格式为json字符串格式，例如： [{"HTTP":80},{"HTTPS":443}] 参数说明：端口号范围 1~65535 **说明** 该注解可以和开启HTTP/2的注解同时配置，且HTTP/2仅在HTTPS端口生效。
kubernetes.io/elb.ssl-redirect	是	string	是否开启HTTP重定向到HTTPS。 格式说明：支持字段 "true" 和 "false" 参数说明："true" 表示开启重定向能力，"false"或参数不存在表示不开启重定向能力 **说明** 该注解不能和灰度发布的注解一起配置。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/test-redirect-listener created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。


```
NAME                    CLASS    HOSTS    ADDRESS         PORTS   AGE
test-redirect-listener  cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.18 为 ELB Ingress 配置转发规则优先级

Ingress使用同一个ELB监听器时，支持按照以下规则进行转发规则优先级排序：

- 不同Ingress的转发规则：按照“kubernetes.io/elb.ingress-order”注解的优先级（取值范围为1~1000）进行排序，值越小表示优先级越高。
- 同一个Ingress的转发规则：“kubernetes.io/elb.rule-priority-enabled”注解设置为“true”时，根据创建Ingress时的转发规则先后顺序进行排序，配置在上面的优先级高；未配置该注解时，同一个Ingress的转发规则会使用ELB侧的默认排序。

未配置以上注解时，同一个ELB监听器下无论包含多个Ingress还是同一个Ingress的转发规则，都会使用ELB侧的默认排序规则。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.15-r0及以上
  - v1.25集群：v1.25.10-r0-r0及以上
  - v1.27集群：v1.27.7-r0及以上
  - v1.28集群：v1.28.5-r0及以上
  - v1.29集群：v1.29.1-r10及以上
  - 其他更高版本的集群
- 集群中需提前部署可用的工作负载用于对外提供访问。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为提供对外访问的工作负载配置Service，ELB Ingress支持的Service类型请参见**ELB Ingress支持的Service类型**。

约束与限制

仅使用独享型ELB时，Ingress支持配置转发规则优先级。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB为例，配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '88'
    kubernetes.io/elb.id: 2c623150-17bf-45f1-ae6f-384b036f547e    # 已有ELB的ID
```

```
    kubernetes.io/elb.class: performance    # ELB的类型
    kubernetes.io/elb.ingress-order: '1'    # 不同Ingress间的转发规则优先级
    kubernetes.io/elb.rule-priority-enabled: 'true'    # 同一个Ingress的转发规则按照paths字段下的转发规则顺序进行排序
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /test1
            backend:
              service:
                name: test1
                port:
                  number: 8081
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
          - path: /test2
            backend:
              service:
                name: test2
                port:
                  number: 8081
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-130 转发规则优先级注解

参数	类型	描述
kubernetes.io/elb.ingress-order	String	不同Ingress间的转发规则排序，参数值越小表示优先级越高，且同一个监听内规则优先级必须唯一，取值范围为1~1000。 仅独享型ELB支持配置。 **说明** 配置该注解时，默认开启“kubernetes.io/elb.rule-priority-enabled”注解，即同时对每个Ingress的转发规则进行排序。
kubernetes.io/elb.rule-priority-enabled	String	仅支持设置为“true”，表示对同一个Ingress的转发规则进行排序，系统将会根据创建Ingress时的转发规则先后顺序确定优先级，配置在上面的优先级高。 未开启该参数时，同一个Ingress的转发规则会使用ELB侧的默认排序，开启后则不允许关闭。 仅独享型ELB支持配置。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。


```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    88      10s
```

----结束

7.4.2.4.19 为 ELB Ingress 配置自定义 Header 转发策略

独享型ELB的Ingress支持自定义Header的转发策略，可通过不同的Header键值来确定转发的后端Service。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.16-r0及以上
  - v1.25集群：v1.25.11-r0及以上
  - v1.27集群：v1.27.8-r0及以上
  - v1.28集群：v1.28.6-r0及以上
  - v1.29集群：v1.29.2-r0及以上
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB创建Ingress的场景为例，YAML配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.id: 08eab934-1636-4d90-a4cd-cb3fa4330411
    kubernetes.io/elb.class: performance  #需使用独享型ELB
    # 表示只能通过Header key1=value1或者key1=value2访问/a路径，最终访问到的服务是 svc-a:80
    kubernetes.io/elb.headers.svc-a: |
      {
          "key": "key1",
          "values": [
              "value1",
              "value2"
          ]
      }
    # 表示只能通过Header key2=value1或者key2=value2访问/b路径，最终访问到的服务是 svc-b:81
    kubernetes.io/elb.headers.svc-b: |
      {
          "key": "key2",
          "values": [
              "value1",
              "value2"
          ]
      }
```



```
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /a
            backend:
              service:
                name: svc-a
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
          - path: /b
            backend:
              service:
                name: svc-b
                port:
                  number: 81
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```

表 7-131 自定义 Header 转发策略注解

参数	类型	描述
kubernetes.io/elb.headers.*${svc_name}*	String	Ingress关联的Service配置自定义的Header，${svc_name}即对应的Service名称。 格式说明：Json字符串，如 {"key": "test", "values": ["value1", "value2"]} ● key/value表示自定义Header的键值对，value最多可以配置8个。 key的取值范围：长度限制1-40字符，只允许包含字母、数字、中划线（-）和下划线（_） value的取值范围：长度限制1-128字符，不支持空格，双引号，支持以下通配符：*（匹配0个或更多字符）和?（正好匹配1个字符） ● 设置自定义Header转发策略后，Ingress不能再同时创建灰度发布策略 ● svc_name最大长度51个字符

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。


```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.20 为 ELB Ingress 配置自定义 EIP

通过CCE自动创建的带有EIP的ELB，可以通过添加Ingress的annotation（kubernetes.io/elb.custom-eip-id）完成ELB的EIP的自定义配置。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r0及以上
  - v1.25集群：v1.25.13-r0及以上
  - v1.27集群：v1.27.10-r0及以上
  - v1.28集群：v1.28.8-r0及以上
  - v1.29集群：v1.29.4-r0及以上
  - v1.30集群：v1.30.1-r0及以上
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 自定义EIP仅支持Ingress更新场景下配置，且Ingress的annotation中包含kubernetes.io/elb.eip-id。
- 自定义的EIP必须是未绑定状态。
- 配置自定义EIP后，如果ELB上的已有EIP是由CCE创建ELB时自动创建的且未被其他资源使用时，删除Ingress时会自动将EIP删除；如果ELB上的已有EIP是由您手动创建，删除Ingress时仅解绑EIP，您需要手动删除原先的EIP。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 在创建Ingress时自动创建一个使用EIP的ELB，详情请参见**添加Ingress时自动创建ELB**。

以使用共享型ELB的Ingress场景为例，查看该Ingress的YAML配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"test-eip
","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_g-
vm","name":"test-eip"}'
    kubernetes.io/elb.class: union
    kubernetes.io/elb.eip-id: 10183660-0bb7-47d4-a899-18891b1ab2f7    # 表示创建ELB时自动创建的EIP
的ID
    kubernetes.io/elb.id: aed5d5c9-65eb-42ab-9f80-57825cbae309
    kubernetes.io/elb.ip: 1.1.1.1
    kubernetes.io/elb.port: "80"
  name: test-eip
  namespace: default
spec:
  ingressClassName: cce
```



```
  rules:
  - http:
      paths:
      - backend:
          service:
            name: test-eip
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
status:
  loadBalancer:
    ingress:
    - ip: 192.168.1.138
```

步骤3 修改该Ingress配置，添加annotation（kubernetes.io/elb.custom-eip-id）。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"test-eip
","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_g-
vm","name":"test-eip"}'
    kubernetes.io/elb.class: union
    kubernetes.io/elb.eip-id: 10183660-0bb7-47d4-a899-18891b1ab2f7     # 表示创建ELB时自动创建的EIP的ID
    kubernetes.io/elb.custom-eip-id: 57bf8bb2-8c7d-4d07-8799-aae16a421802  # 自定义的EIP的ID
    kubernetes.io/elb.id: aed5d5c9-65eb-42ab-9f80-57825cbae309
    kubernetes.io/elb.ip: 1.1.1.1
    kubernetes.io/elb.port: "80"
  name: test-eip
  namespace: default
spec:
  ingressClassName: cce
  rules:
  - http:
      paths:
      - backend:
          service:
            name: test-eip
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
status:
  loadBalancer:
    ingress:
    - ip: 192.168.1.138
```

表 7-132 关键参数说明

参数	参数类型	描述
kubernetes.io/elb.custom-eip-id	String	自定义EIP的ID，您可以前往EIP控制台查看。该EIP必须是处于可绑定状态。

步骤4 Ingress更新成功后，重新查看Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"test-eip
```



```
","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_g-vm","name":"test-eip"}'
    kubernetes.io/elb.class: union
    kubernetes.io/elb.eip-id: 10183660-0bb7-47d4-a899-18891b1ab2f7     # 表示创建ELB时自动创建的EIP的ID
    kubernetes.io/elb.custom-eip-id: 57bf8bb2-8c7d-4d07-8799-aae16a421802  # 自定义的EIP的ID
    kubernetes.io/elb.custom-eip-status: '{"id":"57bf8bb2-8c7d-4d07-8799-aae16a421802","public_ip_address":"3.3.3.3"}' # 自定义的EIP配置成功后，记录了配置的EIP的ID和IP地址
    kubernetes.io/elb.id: aed5d5c9-65eb-42ab-9f80-57825cbae309
    kubernetes.io/elb.ip: 1.1.1.1
    kubernetes.io/elb.port: "80"
  name: test-eip
  namespace: default
spec:
  ingressClassName: cce
  rules:
  - http:
      paths:
      - backend:
          service:
            name: test-eip
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
status:
  loadBalancer:
    ingress:
    - ip: 192.168.1.138
```

----结束

7.4.2.4.21 为 ELB Ingress 配置跨域访问

在Web开发中，由于浏览器的同源策略，一个域下的网页通常不能直接请求另一个域下的资源。CORS（跨资源共享，Cross-Origin Resource Sharing）提供了一种安全的方式来绕过这个限制，允许跨域请求。

使用CORS允许跨域访问的场景较多，可能的场景如下：

- 前后端分离：前端应用部署在一个域名下（如 app.example.com），而后端API服务使用另一个域名（如 api.example.com），前端应用在尝试从API服务获取数据时会遇到跨域资源共享问题，需要配置CORS允许跨域访问。
- 第三方服务集成：网站可能需要调用第三方服务（例如地图服务、社交平台登录等）的API接口，则需要配置CORS允许跨域访问。
- 使用内容分发网络CDN：静态资源可能通过CDN提供，而CDN域名与主站域名不同，需要使用跨域访问来加载这些资源。

说明

- ELB Ingress的跨域访问功能依赖ELB能力，使用该功能前请提交工单申请开通ELB跨域访问能力。
- 为ELB Ingress配置跨域访问时，不支持同时设置**URL重定向**、**Rewrite重写**、**HTTP重定向到HTTPS**。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r10及以上
  - v1.25集群：v1.25.16-r0及以上

  - v1.27集群：v1.27.16-r0及以上
  - v1.28集群：v1.28.13-r0及以上
  - v1.29集群：v1.29.8-r0及以上
  - v1.30集群：v1.30.4-r0及以上
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB创建Ingress的场景为例，YAML配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.class: performance # 仅支持独享型ELB
    kubernetes.io/elb.cors-allow-origin: 'http://example.com' # 允许访问的域
    kubernetes.io/elb.cors-allow-headers: 'fake-header-1' # 允许的请求头
    kubernetes.io/elb.cors-expose-headers: 'fake-header-2' # 需要公开的响应头
    kubernetes.io/elb.cors-allow-methods: 'GET,POST' # 允许的HTTP请求方法
    kubernetes.io/elb.cors-allow-credentials: 'true' # 允许发送凭据
    kubernetes.io/elb.cors-max-age: '3600' # 预检请求的缓存时长
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.id: 3f5906fb-2b07-4e33-b3fe-b095f03d86a6
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            backend:
              service:
                name: nginx-03657
                port:
                  number: 80
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: cce
```


表 7-133 跨域访问注解

参数	类型	描述	样例
kubernetes.io/elb.cors-allow-origin	Array[string]	指定Access-Control-Allow-Origin响应头的值，表示允许访问的域。 支持以下取值： ● 通配符*：表示允许所有域名访问。 ● 配置域名列表：必须为http://或者https://开头的域名，支持填写一级泛域名。格式为“http(s)://example.com”或“http(s)://example.com:port”，端口范围为1~65535。 可填写多个值，以英文逗号分隔。	kubernetes.io/elb.cors-allow-origin: 'http://example.com'
kubernetes.io/elb.cors-allow-headers	Array[string]	指定Access-Control-Allow-Headers响应头的值，表示允许的请求头。可填写多个值，以英文逗号分隔。	kubernetes.io/elb.cors-allow-headers: 'fake-header-1'
kubernetes.io/elb.cors-expose-headers	Array[string]	指定Access-Control-Expose-Headers响应头的值，表示可以被跨域请求读取的自定义响应头部，例如通过客户端的JavaScript代码获取非标准响应头字段。可填写多个值，以英文逗号分隔。	kubernetes.io/elb.cors-expose-headers: 'fake-header-2'
kubernetes.io/elb.cors-allow-methods	Array[string]	指定Access-Control-Allow-Methods响应头的值，表示允许的HTTP请求方法。 可填写多个值，以英文逗号分隔。	kubernetes.io/elb.cors-allow-methods: 'GET,POST'
kubernetes.io/elb.cors-allow-credentials	String	指定Access-Control-Allow-Credentials响应头的值，表示是否允许发送凭据（如Cookies）。 取值如下： ● true：允许发送凭据。 ● false：不允许发送凭据。 设置后不允许删除，如需可通过kubernetes.io/elb.cors-disabled删除所有跨域配置。	kubernetes.io/elb.cors-allow-credentials: 'true'

参数	类型	描述	样例
kubernetes.io/elb.cors-max-age	String	指定Access-Control-Max-Age响应头的值，表示CORS预检请求的缓存时长。单位：秒。取值范围：**-1~172800**。 该参数值应该根据实际需求合理设置。如果设置得太短，可能会导致频繁的预检请求；如果设置得太长，可能会在CORS策略更新后延迟生效。	kubernetes.io/elb.cors-max-age: '3600'
kubernetes.io/elb.cors-disabled	String	该参数用于关闭所有跨域配置。取值如下： • **true**：关闭所有跨域配置。YAML中的参数值不会被删除，但所有配置均不生效。 • **false**：默认取值，跨域配置将根据用户设置生效。	kubernetes.io/elb.cors-disabled: 'true'

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

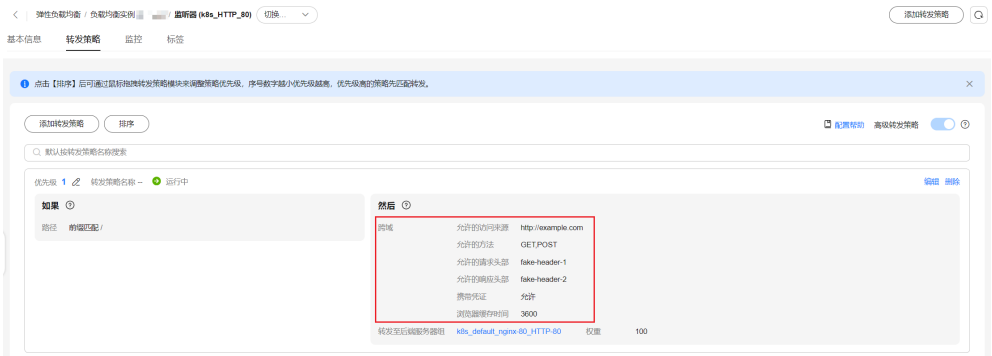
回显如下，表示Ingress服务创建成功。

```
NAME         CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce     *        121.**.**.**    80      10s
```

----结束

验证跨域访问配置是否生效

1. 登录ELB控制台，找到Ingress使用的ELB实例，查看监听器的转发策略中是否存在跨域配置。



2. 使用curl命令行工具发送跨域请求，并检查响应头部。

```
curl -X OPTIONS -H 'Origin: {源站点} {Ingress访问URL}
```


例如，本示例中源站点为example.com，访问的Ingress URL为121.**.**.**:80，则命令如下：

```
curl -X OPTIONS -H 'Origin: example.com' 121.**.**.**:80
```

回显如下，跨域访问会在原始后端响应中添加Access-Control-**相关的请求头：

```
HTTP/1.1 200 OK
Access-Control-Allow-Headers: fake-header-1
Access-Control-Expose-Headers: fake-header-2
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 3600
```

7.4.2.4.22 为 ELB Ingress 配置写入/删除 Header

独享型ELB的Ingress支持自定义重写Header的转发策略，可将在请求中写入或删除配置的Header后再访问后端Service。

说明

ELB Ingress配置重写Header的功能依赖ELB**写入Header和删除Header能力**，使用该功能前请提交工单申请开通ELB相关能力。

前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r10及以上
  - v1.25集群：v1.25.16-r0及以上
  - v1.27集群：v1.27.16-r0及以上
  - v1.28集群：v1.28.13-r0及以上
  - v1.29集群：v1.29.8-r0及以上
  - v1.30集群：v1.30.4-r0及以上
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB创建Ingress的场景为例，YAML配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: 034baaf0-40e8-4e39-b0d9-bf6e5b883cf9
    kubernetes.io/elb.port: "80"
    # 表示对应Rule的Service是test-service时，最终配置的转发策略添加重写Header的能力
    kubernetes.io/elb.actions.test-service: |
      [{
          "type": "InsertHeader",
          "InsertHeaderConfig": {
              "key": "aa",
              "value_type": "USER_DEFINED",
              "value": "aa"
          }
```

```
    },
    {
        "type": "InsertHeader",
        "InsertHeaderConfig": {
            "key": "bb",
            "value_type": "SYSTEM_DEFINED",
            "value": "ELB-ID"
        }
    },
    {
        "type": "InsertHeader",
        "InsertHeaderConfig": {
            "key": "cc",
            "value_type": "REFERENCE_HEADER",
            "value": "cc"
        }
    },
    {
        "type": "RemoveHeader",
        "RemoveHeaderConfig": {
            "key": "dd"
        }
    },
    {
        "type": "RemoveHeader",
        "RemoveHeaderConfig": {
            "key": "ee"
        }
    }]
  name: test
  namespace: default
spec:
  ingressClassName: cce
  rules:
    - http:
        paths:
          - backend:
              service:
                name: test-service
                port:
                  number: 8888
            path: /
            pathType: ImplementationSpecific
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

表 7-134 重写 Header 转发策略注解

参数	类型	描述
kubernetes.io/elb.actions.*${svc_name}*	String	Ingress关联的Service配置重写Header，*${svc_name}*即对应的Service名称，其最大长度为51个字符。 如果该annotation值配置成*[]*则表示删除相应的重写Header的策略。 注解值格式为Json字符串数组，具体格式请参见**表7-135**，例如： [{"type":"InsertHeader","InsertHeaderConfig":{"key":"aa","value_type":"USER_DEFINED","value":"aa"}}] **说明** 最多添加5条写入Header或删除Header配置。

表 7-135 数组结构

参数	使用说明	示例
type	表示重写Header的类型，支持设置以下几种参数值： ● InsertHeader：表示写入Header，需要与InsertHeaderConfig字段搭配使用。 ● RemoveHeader：表示删除Header，需要与RemoveHeaderConfig字段搭配使用。 不支持其他字段。	-

参数	使用说明	示例

InsertHeaderConfig	表示写入Header，仅当type参数值为InsertHeader时使用。
	● key：重写Header的Key值，长度限制1-40字符, 只能由英文字母、数字、下划线和中划线组成。不能是以下字符中的一种（不区分大小写）：connection、upgrade、content-length、transfer-encoding、keep-alive、te、host、cookie、remoteip、authority、x-forwarded-host、x-forwarded-for、x-forwarded-for-port、x-forwarded-tls-certificate-id、x-forwarded-tls-protocol、x-forwarded-tls-cipher、x-forwarded-elb-ip、x-forwarded-port、x-forwarded-elb-id、x-forwarded-elb-vip、x-real-ip、x-forwarded-proto、x-nuwa-trace-ne-in、x-nuwa-trace-ne-out
	● value_type：写入Header的类型（删除Header时该字段配置无效）。仅支持以下字段：
	– USER_DEFINED：表示用户自定义的Header。
	– REFERENCE_HEADER：表示用户引用的Header。
	– SYSTEM_DEFINED：表示系统定义的Header。
	● value：写入Header的value值（删除Header时该字段配置无效）。value_type为SYSTEM_DEFINED时，value只可从CLIENT-PORT、CLIENT-IP、ELB-PROTOCOL、ELB-ID、ELB-PORT、ELB-EIP、ELB-VIP中取值。
		{
		"type": "InsertHeader",
		"InsertHeaderConfig": {
		"key": "aa",
		"value_type": "USER_DEFINED",
		"value": "aa"
		}
		}

参数	使用说明	示例

RemoveHeaderConfig	表示删除Header，仅当type参数值为RemoveHeader时使用。
	● key：删除Header的Key值，长度限制1-40字符, 只能由英文字母、数字、下划线和中划线组成。不能是以下字符中的一种（不区分大小写）：connection、upgrade、content-length、transfer-encoding、keep-alive、te、host、cookie、remoteip、authority、x-forwarded-host、x-forwarded-for、x-forwarded-for-port、x-forwarded-tls-certificate-id、x-forwarded-tls-protocol、x-forwarded-tls-cipher、x-forwarded-elb-ip、x-forwarded-port、x-forwarded-elb-id、x-forwarded-elb-vip、x-real-ip、x-forwarded-proto、x-nuwa-trace-ne-in、x-nuwa-trace-ne-out
		{
		"type": "RemoveHeader",
		"RemoveHeaderConfig": {
		"key": "ee"
		}
		}

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

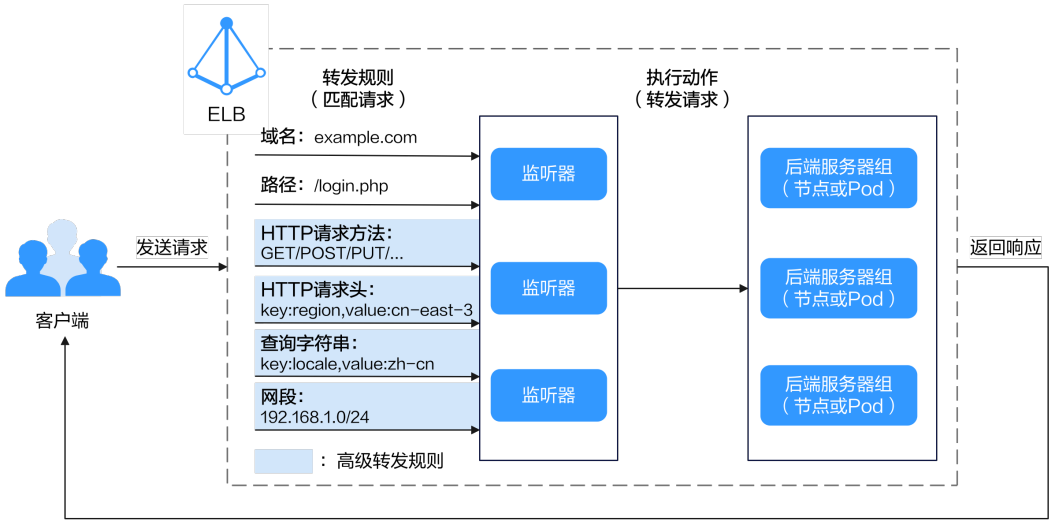
```
NAME          CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce      *        121.**.**.**    80      10s
```

----结束

7.4.2.4.23 为 ELB Ingress 配置高级转发规则

Ingress支持多样化的转发规则，可以根据HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie等请求参数匹配不同的监听器（每个监听器对应一个ELB访问端口），便于灵活地分流业务，合理分配资源。

图 7-79 高级转发规则示意图



前提条件

- 已创建一个CCE Standard或CCE Turbo集群，且集群版本满足以下要求：
  - v1.23集群：v1.23.18-r10及以上
  - v1.25集群：v1.25.16-r0及以上
  - v1.27集群：v1.27.16-r0及以上
  - v1.28集群：v1.28.13-r0及以上
  - v1.29集群：v1.29.8-r0及以上
  - v1.30集群：v1.30.4-r0及以上
  - 其他更高版本的集群
- 您需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

仅使用独享型ELB时，Ingress支持配置高级转发规则。

通过 kubectl 命令行配置

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以使用已有ELB创建Ingress的场景为例，YAML配置如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.class: performance
    kubernetes.io/elb.id: ab53c3b2-xxxx-xxxx-xxxx-5ac3eb2887be
    kubernetes.io/elb.port: '80'
    # 表示访问svc-hello1服务，请确保svc-hello1服务存在
    kubernetes.io/elb.conditions.svc-hello1: |
      [
        {
          "type": "Method",
```



```
        "methodConfig": {
          "values": [
            "GET",
            "POST"
          ]
        }
      },
      {
        "type": "Header",
        "headerConfig": {
          "key": "gray-hello",
          "values": [
            "value1",
            "value2"
          ]
        }
      },
      {
        "type": "Cookie",
        "cookieConfig": {
          "values": [
            {
              "key": "querystringkey1",
              "value": "querystringvalue2"
            },
            {
              "key": "querystringkey3",
              "value": "querystringvalue4"
            }
          ]
        }
      },
      {
        "type": "QueryString",
        "queryStringConfig": {
          "key": "testKey",
          "values": [
            "testValue"
          ]
        }
      },
      {
        "type": "SourceIp",
        "sourceIpConfig": {
          "values": [
            "192.168.0.0/16",
            "172.16.0.0/16"
          ]
        }
      }
    ]
  name: ingress-test
spec:
  ingressClassName: cce
  rules:
  - http:
      paths:
      - path: /hello1
        pathType: ImplementationSpecific
        backend:
          service:
            name: svc-hello1
            port:
              number: 80
```


表 7-136 高级转发规则注解

参数	类型	描述
kubernetes.io/elb.conditions.*${svc_name}*	String	配置高级转发规则，*${svc_name}*即对应的Service名称，其最大长度为48个字符。 如果该annotation值配置成*[]*则表示删除相应的高级转发规则。 注解值格式为Json字符串数组，具体格式请参见表7-137。 **须知** • 由于ELB的API限制，conditions设置的数量上限为10，即一条kubernetes.io/elb.conditions.{svcName}中，最多包含十条key-value键值对。 • 一条conditions中的数组中不同的规则是“与”关系，但同一个规则块中的值是“或”关系。例如，Method和QueryString两种转发条件都配置时，需要同时满足，才能实现目标流量分发。但如果Method值为GET，POST，即只需要满足Method为GET或POST，且QueryString满足条件即可。

表 7-137 数组结构

参数	使用说明	示例
type	匹配类型，支持设置以下几种参数值： • Method：根据匹配的HTTP请求方法进行转发，需要与methodConfig字段搭配使用。Method仅可配置一次。 • Header：根据匹配的HTTP请求头进行转发，需要与headerConfig字段搭配使用。 • Cookie：根据匹配的Cookie进行转发，需要与cookieConfig字段搭配使用。 • QueryString：根据请求中匹配的字符串进行转发，需要与queryStringConfig字段搭配使用。 • SourceIp：根据匹配的请求网段进行转发，需要与sourceIpConfig字段搭配使用。SourceIp仅可配置一次。	-

参数	使用说明	示例

methodConfig	触发转发的HTTP请求方法，仅当type参数值为Method时使用。
	可以并列设置多个请求方法，支持以下几种请求方法：GET、POST、PUT、DELETE、PATCH、HEAD、OPTIONS。
		methodConfig无需设置key，仅设置values即可，values为数组。
		{
		"type": "Method",
		"methodConfig": {
		"values": [
		"GET",
		"POST"
		]
		}
		}

headerConfig	触发转发的HTTP请求头，仅当type参数值为Header时使用。
	● 键（key）：只能由英文字母、数字、下划线和中划线组成。HTTP请求头User-agent和Connection仅支持首字母大写的形式。
	● 值（value）一个键下可以配置多个值。只能包含英文字母、数字和特殊字符!#$%&'()*+,.\/:;<=>?@[]^-_'{|}~。还支持*和?两种通配符。
		一个headerConfig结构体中仅能设置一个Key，如需设置多个Key，请配置多组HeaderConfig规则块。
		{
		"type": "Header",
		"headerConfig": {
		"key": "gray-hello",
		"values": [
		"value1",
		"value2"
		]
		}
		}

cookieConfig	触发转发的Cookie，仅当type参数值为Cookie时使用。Cookie是键值对的形式，需要分别设置值：
	● 键（key）：键的长度为1~100个字符，且首尾字符不能为空格。
	● 值（value）：一个键下配置一个值，值的长度为1~100个字符。
	支持输入多个Cookie键值对，键值对支持英文字母、数字和特殊字符!%'"()*+,./:=?@^-_`~。
		设置cookieConfig时，values字段中支持设置多个key，value对数组，多个数组之间为或关系。
		{
		"type": "Cookie",
		"cookieConfig": {
		"values": [
		{
		"key": "querystringkey1",
		"value": "querystringvalue2"
		},
		{
		"key": "querystringkey3",
		"value": "querystringvalue4"
		}
		]
		}
		}

参数	使用说明	示例

queryStringConfig	当请求中的字符串与设置好的转发规则中的字符串相匹配时，触发转发，仅当type参数值为QueryString时使用。
	查询字符串是键值对的形式，需要分别设置值：
	• 键（key）：只能包含英文字母、数字和特殊字符!$'()*+,./:;=?@^-_'。
	• 值（value）：一个键下可以配置多个值。只能包含英文字母、数字和特殊字符!$'()*+,./:;=?@^-_'。还支持*和?两种通配符。
		queryStringConfig仅能设置一个Key，如需设置多个Key，请配置多组QueryStringConfig规则块。
		{
		"type": "QueryString",
		"queryStringConfig": {
		"key": "testKey",
		"values": [
		"testValue"
		]
		}
		}

sourceIpConfig	触发转发的请求网段，仅当type参数值为SourceIp时使用。
	网段示例： 192.168.1.0/24或2020:50::44/127
		sourceIpConfig无需设置key，仅设置values即可，values为数组。
		{
		"type": "SourceIp",
		"sourceIpConfig": {
		"values": [
		"192.168.0.0/16",
		"172.16.0.0/16"
		]
		}
		}

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS    HOSTS    ADDRESS         PORTS   AGE
ingress-test  cce     *        121.**.**.**    80      10s
```

----结束

7.4.3 Nginx Ingress 管理

7.4.3.1 通过控制台创建 Nginx Ingress

Ingress是Kubernetes中的一种资源对象，用来管理集群外部访问集群内部服务的方式。您可以通过Ingress资源来配置不同的转发规则，从而根据转发规则访问集群内Pod。本文以**Nginx工作负载**为例，为您介绍如何使用控制台创建Nginx Ingress。

前提条件

- Ingress为后端工作负载提供网络访问，因此集群中需提前部署可用的工作负载。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为上述工作负载配置ClusterIP类型或NodePort类型的Service，可参考**集群内访问（ClusterIP）**或**节点访问（NodePort）**配置示例Service。
- 添加Nginx Ingress时，需在集群中提前安装NGINX Ingress 控制器，具体操作可参考**安装插件**。

约束与限制

- **不建议在ELB服务页面修改ELB实例的任何配置，否则将导致服务异常。**如果您已经误操作，请卸载Nginx Ingress插件后重装。
- Ingress转发策略中注册的URL需与后端应用提供访问的URL一致，否则将返回404错误。
- 负载均衡实例需与当前集群处于相同VPC 且为相同公网或私网类型。
- 负载均衡实例需要拥有至少两个监听器配额，且端口80和443没有被监听器占用。

添加 Nginx Ingress

本节以Nginx作为工作负载并添加Nginx Ingress为例进行说明。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击右上角“创建路由”。

步骤3 设置Ingress参数。

- **名称：**自定义Ingress名称，例如nginx-ingress-demo。
- **命名空间：**选择需要添加Ingress的命名空间。
- **对接Nginx：**集群中已安装**NGINX Ingress控制器**插件后显示此选项，未安装该插件时本选项不显示。
  - **前端协议：**支持HTTP和HTTPS，安装NGINX Ingress控制器插件时预留的监听端口，默认HTTP为80，HTTPS为443。使用HTTPS需要配置相关证书。
  - **证书来源：**使用证书以支持HTTPS数据传输加密认证。
    - 如果您选择“TLS密钥”，需要提前创建IngressTLS或kubernetes.io/tls类型的密钥证书，创建密钥的方法请参见**创建密钥**。
    - 如果您选择“默认证书”，NGINX Ingress控制器会使用插件默认证书进行加密认证。默认证书可在安装**NGINX Ingress控制器**插件时进行自定义配置，未配置自定义证书时将使用NGINX Ingress控制器自带证书。
  - **SNI：**SNI（Server Name Indication）是TLS的扩展协议，在该协议下允许同一个IP地址和端口号下对外提供多个基于TLS的访问域名，且不同的域名可以使用不同的安全证书。开启SNI后，允许客户端在发起TLS握手请求时就提交请求的域名信息。负载均衡收到TLS请求后，会根据请求的域名去查找证书：若找到域名对应的证书，则返回该证书认证鉴权；否则，返回缺省证书（服务器证书）认证鉴权。

- **转发策略配置：**请求的访问地址与转发规则匹配时（转发规则由域名、URL组成），此请求将被转发到对应的目标Service处理。单击“添加转发策略”按钮可添加多条转发策略。
  - 域名：实际访问的域名地址。请确保所填写的域名已注册并备案，在Ingress创建完成后，将域名与自动创建的负载均衡实例的IP（即Ingress访问地址的IP部分）绑定。一旦配置了域名规则，则必须使用域名访问。
  - 路径匹配规则：
    - 默认：默认为前缀匹配。
    - 前缀匹配：例如映射URL为/healthz，只要符合此前缀的URL均可访问。例如/healthz/v1，/healthz/v2。
    - 精确匹配：表示只有URL完全匹配时，访问才能生效。例如映射URL为/healthz，则必须为此URL才能访问。
  - 路径：需要注册的访问路径，例如：/healthz。

    说明

    - Nginx Ingress的访问路径匹配规则是基于“/”符号分隔的路径前缀匹配，并区分大小写。只要访问路径以“/”符号分隔后的子路径匹配此前缀，均可正常访问，但如果该前缀仅是子路径中的部分字符串，则不会匹配。例如URL设置为/healthz，则匹配/healthz/v1，但不匹配/healthzv1。
    - 此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。

      例如，Nginx应用默认的Web访问路径为“/usr/share/nginx/html”，在为Ingress转发策略添加“/test”路径时，需要应用的Web访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回404。
  - 目标服务名称：请选择已有Service或新建Service。页面列表中的查询结果已自动过滤不符合要求的Service。
  - 目标服务访问端口：可选择目标Service的访问端口。
  - 操作：可单击“删除”按钮删除该配置。
- **注解**：以“key: value”形式设置，可通过**Annotations**查询nginx-ingress支持的配置。

步骤4 配置完成后，单击“确定”。

创建完成后，在Ingress列表可查看到已添加的Ingress。

----结束

7.4.3.2 通过 Kubectl 命令行创建 Nginx Ingress

本节以**Nginx工作负载**为例，说明kubectl命令添加Nginx Ingress的方法。

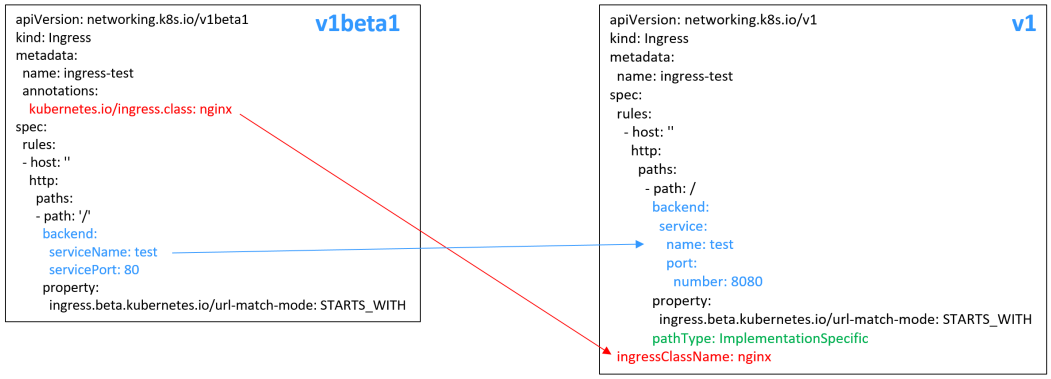
关于 CCE v1.23 集群中 Ingress API 版本升级的说明

CCE从v1.23版本集群开始，将Ingress切换到**networking.k8s.io/v1**版本。

v1版本的参数相较v1beta1版本的参数有如下区别：

- ingress类型由annotations中**kubernetes.io/ingress.class**变为使用**spec.ingressClassName**字段。

- **backend**的写法变化。
- 每个路径下必须指定路径类型**pathType**，支持如下类型。
  - ImplementationSpecific: 对于这种路径类型，匹配方法取决于具体Ingress Controller的实现。在CCE中会使用ingress.beta.kubernetes.io/url-match-mode指定的匹配方式，这与v1beta1方式相同。
  - Exact：精确匹配 URL 路径，且区分大小写。
  - Prefix：基于以 / 分隔的 URL 路径前缀匹配。匹配区分大小写，并且对路径中的元素逐个匹配。 路径元素指的是由 / 分隔符分隔的路径中的标签列表。



前提条件

- 集群必须已安装NGINX Ingress 控制器，具体操作可参考**安装插件**。
- Ingress为后端工作负载提供网络访问，因此集群中需提前部署可用的工作负载。若您无可用工作负载，可参考**创建无状态负载（Deployment）**、**创建有状态负载（StatefulSet）**或**创建守护进程集（DaemonSet）**部署工作负载。
- 为上述工作负载配置ClusterIP类型或NodePort类型的Service，可参考**集群内访问（ClusterIP）**或**节点访问（NodePort）**配置示例Service。

添加 Nginx Ingress

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

说明

CCE在1.23版本集群开始Ingress切换到networking.k8s.io/v1版本，之前版本集群使用networking.k8s.io/v1beta1。v1版本与v1beta1版本的区别请参见**关于CCE v1.23集群中Ingress API版本升级的说明**。

以HTTP协议访问为例，YAML文件配置如下。

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
spec:
  rules:
    - host: ''
      http:
        paths:
```



```
        - path: /
          backend:
            service:
              name: <your_service_name>  #替换为您的目标服务名称
              port:
                number: <your_service_port>  #替换为您的目标服务端口
          property:
            ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
          pathType: ImplementationSpecific
  ingressClassName: nginx  # 表示使用Nginx Ingress。如果集群中安装了多套NGINX Ingress控制器，需将nginx替换为自定义的控制器名称，用于识别Ingress对接的控制器实例
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx  # 表示使用Nginx Ingress
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

表 7-138 关键参数说明

参数	是否必填	参数类型	描述
kubernetes.io/ingress.class	是（仅1.21及以下集群）	String	nginx：表示使用Nginx Ingress，未安装NGINX Ingress控制器插件时无法使用。 通过API接口创建Ingress时必须增加该参数。
ingressClassName	是 （仅1.23及以上集群）	String	nginx：表示使用Nginx Ingress，未安装NGINX Ingress控制器插件时无法使用。如果集群中安装了多套NGINX Ingress控制器，需将*nginx*替换为自定义的**控制器名称**，用于识别Ingress对接的控制器实例。 当NGINX Ingress控制器插件为2.5.4及以上时，集群中支持同时安装多套NGINX Ingress控制器，该参数值需设置为安装控制器时指定的自定义**控制器名称**，表示该Ingress由此控制器进行管理。 通过API接口创建Ingress时必须增加该参数。
host	否	String	为服务访问域名配置，默认为""，表示域名全匹配。请确保所填写的域名已注册并备案，一旦配置了域名规则后，必须使用域名访问。

参数	是否必填	参数类型	描述
path	是	String	为路由路径，用户自定义设置。所有外部访问请求需要匹配host和path。 **说明** • Nginx Ingress的访问路径匹配规则是基于“/”符号分隔的路径前缀匹配，并区分大小写。只要访问路径以“/”符号分隔后的子路径匹配此前缀，均可正常访问，但如果该前缀仅是子路径中的部分字符串，则不会匹配。例如URL设置为/healthz，则匹配/healthz/v1，但不匹配/healthzv1。 • 此处添加的访问路径要求后端应用内存在相同的路径，否则转发无法生效。例如，Nginx应用默认的Web访问路径为“/usr/share/nginx/html”，在为Ingress转发策略添加“/test”路径时，需要应用的Web访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回404。
ingress.beta.kubernetes.io/url-match-mode	否	String	路由匹配策略。 默认值为“STARTS_WITH”(前缀匹配)。 取值范围： • EQUAL_TO：精确匹配 • STARTS_WITH：前缀匹配

参数	是否必填	参数类型	描述
pathType	是	String	路径类型，该字段仅v1.23及以上集群支持。 • ImplementationSpecific: 匹配方法取决于具体Ingress Controller的实现。在CCE中会使用ingress.beta.kubernetes.io/url-match-mode指定的匹配方式。 • Exact：精确匹配 URL 路径，且区分大小写。 • Prefix：前缀匹配，且区分大小写。该方式是将URL路径通过“/”分隔成多个元素 ，并且对元素进行逐个匹配。 如果URL中的每个元素均和路径匹配，则说明该URL的子路径均可以正常路由。 **说明** – Prefix匹配时每个元素均需精确匹配，如果URL的最后一个元素是请求路径中最后一个元素的子字符串，则不会匹配 。例如：/foo/bar匹配/foo/bar/baz，但不匹配/foo/barbaz。 – 通过“/”分隔元素时，若URL或请求路径以“/”结尾，将会忽略结尾的“/”。例如：/foo/bar会匹配/foo/bar/。 关于Ingress路径匹配示例，请参见**示例**。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤5 访问工作负载（例如**Nginx工作负载**），在浏览器中输入访问地址“http://121.**.**.**:80”进行验证。

其中，“121.**.**.**”为统一负载均衡实例的IP地址。

----结束

7.4.3.3 用于配置 Nginx Ingress 的注解（Annotations）

CCE的Nginx Ingress插件使用社区模板与镜像，Nginx Ingress默认的其他参数无法满足业务需求时，也可通过添加注解Annotation（注解）的方式自定义参数，例如默认后端、超时时间、请求body体大小等。

本文介绍在创建Nginx类型的Ingress时常用的Annotation。

说明

- 注解的键值只能是字符串，其他类型（如布尔值或数值）必须使用引号，例如"true"、"false"、"100"。
- Nginx Ingress支持社区的原生注解，详情请参考**Annotations**。

Ingress 类型

表 7-139 Ingress 类型注解

参数	类型	描述	支持的集群版本
kubernetes.io/ingress.class	String	• nginx：表示使用Nginx Ingress。 • cce：表示使用自研ELB Ingress。 通过API接口创建Ingress时必须增加该参数。 v1.23及以上集群使用ingressClassName参数代替，详情请参见**通过Kubectl命令行创建Nginx Ingress**。	仅v1.21及以下集群

上述注解的使用方法详情请参见**通过Kubectl命令行创建Nginx Ingress**。

配置重定向规则

表 7-140 重定向规则注解

参数	类型	描述
nginx.ingress.kubernetes.io/permanent-redirect	String	将访问请求永久重定向至某个目标网址（状态码为301）。
nginx.ingress.kubernetes.io/permanent-redirect-code	String	修改永久重定向的返回状态码为指定值。
nginx.ingress.kubernetes.io/temporal-redirect	String	将访问请求临时重定向至某个目标网址（状态码为302）。
nginx.ingress.kubernetes.io/ssl-redirect	String	是否只能通过SSL访问（当Ingress包含证书时默认为true），将HTTP请求重定向至HTTPS。
nginx.ingress.kubernetes.io/force-ssl-redirect	String	是否强制重定向到HTTPS，即使Ingress未启用TLS，通过HTTP访问时，请求将会被强制重定向（状态码为308）到HTTPS。

具体使用场景和说明请参见**为Nginx Ingress配置重定向规则**。

配置 URL 重写规则

表 7-141 URL 重写规则注解

参数	类型	描述
nginx.ingress.kubernetes.io/rewrite-target	String	重定向流量的目标URI。

具体使用场景和说明请参见**为Nginx Ingress配置URL重写规则**。

对接 HTTPS 协议的后端服务

表 7-142 对接 HTTPS 协议的后端服务注解

参数	类型	描述
nginx.ingress.kubernetes.io/backend-protocol	String	参数值为'HTTPS'，表示使用HTTPS协议转发请求到后端业务容器。

具体使用场景和说明请参见**为Nginx Ingress配置HTTPS协议的后端服务**。

创建一致性哈希负载均衡规则

表 7-143 一致性哈希负载均衡注解

参数	类型	描述
nginx.ingress.kubernetes.io/upstream-hash-by	String	为后端启用一致性哈希进行负载均衡，参数值支持nginx参数、文本值或任意组合，例如： • nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri"代表按照请求uri进行hash。 • nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri$host"代表按照请求uri和域名进行hash。 • nginx.ingress.kubernetes.io/upstream-hash-by: "${request_uri}-text-value"代表按照请求uri和文本值进行hash。

具体使用场景和说明请参见**为Nginx Ingress配置一致性哈希负载均衡**。

自定义超时时长

表 7-144 自定义超时时长注解

参数	类型	描述
nginx.ingress.kubernetes.io/proxy-connect-timeout	String	自定义连接超时时长，设置超时值时无需填写单位，默认单位为秒。 例如： `nginx.ingress.kubernetes.io/proxy-connect-timeout: '120'`

自定义 Body 体大小

表 7-145 自定义 Body 体大小注解

参数	类型	描述
nginx.ingress.kubernetes.io/proxy-body-size	String	当请求中的Body体大小超过允许的最大值时，将向客户端返回413错误，您可通过该参数调整Body体的限制大小。该参数值的基本单位为字节，您可以使用k、m、g等参数单位，换算关系如下： 1g=1024m；1m=1024k；1k=1024字节 例如： `nginx.ingress.kubernetes.io/proxy-body-size: 8m`

HTTPS 双向认证

Nginx Ingress支持配置服务器与客户端之间的双向HTTPS认证来保证连接的安全性。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行以下命令，创建自签名的CA证书。

```
openssl req -x509 -sha256 -newkey rsa:4096 -keyout ca.key -out ca.crt -days 356 -nodes -subj '/CN=Ingress Cert Authority'
```

预期输出：

```
Generating a RSA private key
.....++++
.....++++
writing new private key to 'ca.key'
-----
```

步骤3 执行以下命令，创建Server端证书。

1. 执行以下命令，生成Server端证书的请求文件。
   ```
   openssl req -new -newkey rsa:4096 -keyout server.key -out server.csr -nodes -subj '/CN=foo.bar.com'
   ```
   预期输出：
   ```
   Generating a RSA private key
   .....++++
   .....++++
   writing new private key to 'server.key'
   -----
   ```
2. 执行以下命令，使用根证书签发Server端请求文件，生成Server端证书。
   ```
   openssl x509 -req -sha256 -days 365 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
   ```
   预期输出：
   ```
   Signature ok
   subject=CN = foo.bar.com
   Getting CA Private Key
   ```

步骤4 执行以下命令，生成Client端证书。

1. 执行以下命令，生成Client端证书的请求文件。
   ```
   openssl req -new -newkey rsa:4096 -keyout client.key -out client.csr -nodes -subj '/CN=Ingress'
   ```
   预期输出：
   ```
   Generating a RSA private key
   .....++++
   .....++++
   writing new private key to 'client.key'
   -----
   ```
2. 执行以下命令，使用根证书签发Client端请求文件，生成Client端证书。
   ```
   openssl x509 -req -sha256 -days 365 -in client.csr -CA ca.crt -CAkey ca.key -set_serial 02 -out client.crt
   ```
   预期输出：
   ```
   Signature ok
   subject=CN = Ingress
   Getting CA Private Key
   ```

步骤5 执行**ls**命令，查看创建的证书。

预期输出：

```
ca.crt  ca.key  client.crt  client.csr  client.key  server.crt  server.csr  server.key
```

步骤6 执行以下命令，生成CA证书的Secret。

```
kubectl create secret generic ca-secret --from-file=ca.crt=ca.crt
```

预期输出：


```
secret/ca-secret created
```

步骤7 执行以下命令，生成Server端证书的Secret。

```
kubectl create secret generic tls-secret --from-file=tls.crt=server.crt --from-file=tls.key=server.key
```

预期输出：

```
secret/tls-secret created
```

步骤8 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

- **1.23及以上版本集群**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    nginx.ingress.kubernetes.io/auth-tls-secret: "default/ca-secret"   #替换您的CA证书密钥
    nginx.ingress.kubernetes.io/auth-tls-verify-depth: "1"
    nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream: "true"
  name: ingress-test
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          service:
            name: nginx-test  #替换为您的目标服务名称
            port:
              number: 80  #替换为您的目标服务端口
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - foo.bar.com
    secretName: tls-secret   #替换您的TLS证书密钥
  ingressClassName: nginx
```

- **1.21及以下版本集群**

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    nginx.ingress.kubernetes.io/auth-tls-secret: "default/ca-secret"   #替换您的CA证书密钥
    nginx.ingress.kubernetes.io/auth-tls-verify-depth: "1"
    nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream: "true"
  name: ingress-test
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: '/'
        backend:
          serviceName: nginx-test  #替换为您的目标服务名称
          servicePort: 80  #替换为您的目标服务端口
  tls:
  - hosts:
    - foo.bar.com
    secretName: tls-secret   #替换为您的TLS密钥证书
```

步骤9 执行以下命令，创建Ingress。

```
kubectl create -f ingress-test.yaml
```


预期输出：

```
ingress.networking.k8s.io/ingress-test created
```

步骤10 执行以下命令，查看Ingress的IP地址。

```
kubectl get ingress
```

预期输出：

```
NAME         CLASS   HOSTS         ADDRESS      PORTS     AGE
nginx-test   nginx   foo.bar.com   10.3.xx.xx   80, 443   27m
```

步骤11 执行以下命令，将Ingress的IP地址更新到Hosts文件中，替换下面的IP地址为真实获取的Ingress的IP地址

```
echo "10.3.xx.xx  foo.bar.com" | sudo tee -a /etc/hosts
```

预期输出：

```
10.3.xx.xx  foo.bar.com
```

步骤12 结果验证。

- 客户端不传证书访问

  ```
  curl --cacert ./ca.crt  https://foo.bar.com
  ```

  预期输出：

  ```
  <html>
  <head><title>400 No required SSL certificate was sent</title></head>
  <body>
  <center><h1>400 Bad Request</h1></center>
  <center>No required SSL certificate was sent</center>
  <hr><center>nginx</center>
  </body>
  </html>
  ```

- 客户端传证书访问

  ```
  curl --cacert ./ca.crt --cert ./client.crt --key ./client.key https://foo.bar.com
  ```

  预期输出：

  ```
  <!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>
  ```

----结束

域名正则化

Nginx Ingress支持配置“nginx.ingress.kubernetes.io/server-alias”注解实现域名配置正则表达式。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

以正则表达式**~^www\.\d+\.example\.com$,abc.example.com**为例，表示使用www.{一个或多个数字}.example.com和abc.example.com域名也可正常访问Ingress。

- **1.23及以上版本集群**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/server-alias: '~^www\.\d+\.example\.com$,abc.example.com'
  name: ingress-test
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          service:
            name: nginx-93244  #替换为您的目标服务名称
            port:
              number: 80  #替换为您的目标服务端口
        path: /
        pathType: ImplementationSpecific
  ingressClassName: nginx
```

- **1.21及以下版本集群**

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ server-alias: '~^www\.\d+\.example\.com$,abc.example.com'
  name: ingress-test
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: '/'
        backend:
          serviceName: nginx-test  #替换为您的目标服务名称
          servicePort: 80  #替换为您的目标服务端口
```

步骤3 执行以下命令，创建Ingress。

```
kubectl create -f ingress-test.yaml
```

预期输出：

```
ingress.networking.k8s.io/ingress-test created
```

步骤4 查看Nginx Ingress Controller的配置。

1. 执行以下命令，查看Nginx Ingress Controller服务的Pod

```
kubectl get pods -n kube-system | grep nginx-ingress-controller
```

预期输出：


```
cceaddon-nginx-ingress-controller-68d7bcc67-dxxxx       1/1     Running   0          18h
cceaddon-nginx-ingress-controller-68d7bcc67-cxxxx       1/1     Running   0          18h
```

2. 执行以下命令，查看Nginx Ingress Controller的配置

```
kubectl exec -n kube-system cceaddon-nginx-ingress-controller-68d7bcc67-dxxxx cat /etc/nginx/nginx.conf | grep -C3 "foo.bar.com"
```

预期输出：

```
        ## start server foo.bar.com
        server {
                server_name foo.bar.com abc.example.com ~^www\.\d+\.example\.com$ ;

                listen 80  ;
                listen [::]:80  ;
--
                }

        }
        ## end server foo.bar.com
```

步骤5 执行以下命令，获取Ingress对应的IP。

```
kubectl get ingress
```

预期输出：

```
NAME         CLASS   HOSTS         ADDRESS      PORTS   AGE
nginx-test   nginx   foo.bar.com   10.3.xx.xx   80      14m
```

步骤6 执行以下命令，测试不同规则下的服务访问。

- 执行以下命令，通过Host: foo.bar.com访问服务。

  ```
  curl -H "Host: foo.bar.com" 10.3.xx.xx/
  ```

  预期可正常访问网页。

- 执行以下命令，通过Host: www.123.example.com访问服务

  ```
  curl -H "Host: www.123.example.com" 10.3.xx.xx/
  ```

  预期可正常访问网页。

- 执行以下命令，通过Host: www.321.example.com访问服务

  ```
  curl -H "Host: www.321.example.com" 10.3.xx.xx/
  ```

  预期可正常访问网页。

----结束

灰度发布

灰度发布功能可以通过设置注解来实现，为了启用灰度发布功能，需要设置注解**nginx.ingress.kubernetes.io/canary: "true"**，通过不同注解可以实现不同的灰度发布功能：

表 7-146 灰度发布注解

参数	类型	描述
nginx.ingress.kubernetes.io/canary-weight	String	设置请求到指定服务的百分比（值为0~100的整数）。

参数	类型	描述
nginx.ingress.kubernetes.io/canary-by-header	String	基于Request Header的流量切分，如果请求头中包含指定的header名称，并且值为“always”，就将该请求转发给Canary Ingress定义的对应后端服务。如果值为“never”则不转发，可用于回滚到旧版本。如果为其他值则忽略该annotation，并通过优先级将请求流量分配到其他规则。
nginx.ingress.kubernetes.io/canary-by-header-value	String	必须与canary-by-header一起使用，可自定义请求头的取值，包含但不限于“always”或“never”。当请求头的值命中指定的自定义值时，请求将会转发给Canary Ingress定义的对应后端服务，如果是其他值则忽略该annotation，并通过优先级将请求流量分配到其他规则。
nginx.ingress.kubernetes.io/canary-by-header-pattern	String	与canary-by-header-value类似，唯一区别是该annotation用正则表达式匹配请求头的值，而不是某一个固定值。如果该annotation与canary-by-header-value同时存在，该annotation将被忽略。
nginx.ingress.kubernetes.io/canary-by-cookie	String	基于Cookie的流量切分，当配置的cookie值为always时，请求流量将被分配到灰度服务入口；当配置的cookie值为never时，请求流量将不会分配到灰度服务入口。

说明

- 以上注解规则会按优先级进行评估，优先级为：canary-by-header -> canary-by-cookie -> canary-weight。
- 当Ingress被标记为Canary Ingress时，除了nginx.ingress.kubernetes.io/load-balance和nginx.ingress.kubernetes.io/upstream-hash-by外，所有其他非Canary的注解都将被忽略。

更多信息，请参见**通过Nginx Ingress实现灰度发布和蓝绿发布**。

以下为部分注解配置示例：

- 基于权重灰度：配置灰度服务的权重为20%。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "20"
...
```

- 基于Header灰度：请求Header为cce:always时将访问灰度服务；请求Header为cce:never时将不访问灰度服务；其他Header将根据灰度权重将流量分配给灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
```



```
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "50"
    nginx.ingress.kubernetes.io/canary-by-header: "cce"
...
```

- 基于Header灰度（自定义header值）：当请求Header为cce:test时将访问灰度服务；其他Header将根据灰度权重将流量分配给灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "20"
    nginx.ingress.kubernetes.io/canary-by-header: "cce"
    nginx.ingress.kubernetes.io/canary-by-header-value: "test"
...
```

- 基于Cookie灰度：当Header不匹配时，请求的Cookie为region=always时将访问灰度服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "20"
    nginx.ingress.kubernetes.io/canary-by-header: "cce"
    nginx.ingress.kubernetes.io/canary-by-header-value: "test"
    nginx.ingress.kubernetes.io/canary-by-cookie: "region"
...
```

相关文档

更多关于Nginx Ingress支持的注解参数，请参见**Annotations**。

7.4.3.4 Nginx Ingress 高级配置示例

7.4.3.4.1 为 Nginx Ingress 配置 HTTPS 证书

Ingress支持配置HTTPS证书以提供安全服务。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 Ingress支持使用kubernetes.io/tls和IngressTLS两种TLS密钥类型，此处以IngressTLS类型为例，详情请参见**创建密钥**。kubernetes.io/tls类型的密钥示例及说明请参见**TLS Secret**。

执行如下命令，创建名为“**ingress-test-secret.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test-secret.yaml
```

YAML文件配置如下：

```
apiVersion: v1
data:
  tls.crt: LS0******tLS0tCg==
  tls.key: LS0tL******0tLS0K
kind: Secret
metadata:
  annotations:
    description: test for ingressTLS secrets
  name: ingress-test-secret
```

```
  namespace: default
type: IngressTLS
```

说明

此处tls.crt和tls.key为示例，请获取真实的证书和密钥进行替换。tls.crt和tls.key的值为Base64编码后的内容。

步骤3 创建密钥。

```
kubectl create -f ingress-test-secret.yaml
```

回显如下，表明密钥已创建。

```
secret/ingress-test-secret created
```

查看已创建的密钥。

```
kubectl get secrets
```

回显如下，表明密钥创建成功。

```
NAME                  TYPE                DATA   AGE
ingress-test-secret   IngressTLS          2      13s
```

步骤4 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
spec:
  tls:
  - hosts:
    - foo.bar.com
    secretName: ingress-test-secret  #替换为您的TLS密钥证书
  rules:
    - host: foo.bar.com
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  tls:
  - hosts:
    - foo.bar.com
    secretName: ingress-test-secret   #替换为您的TLS密钥证书
  rules:
  - host: foo.bar.com
    http:
```



```
    paths:
    - path: '/'
      backend:
        serviceName: <your_service_name>  #替换为您的目标服务名称
        servicePort: <your_service_port>  #替换为您的目标服务端口
ingressClassName: nginx
```

步骤5 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤6 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤7 访问工作负载（例如**Nginx工作负载**），在浏览器中输入安全访问地址https://121.**.**.**:443进行验证。

其中，121.**.**.**为统一负载均衡实例的IP地址。

----结束

7.4.3.4.2 为 Nginx Ingress 配置重定向规则

配置永久重定向规则

如果您想将访问请求永久重定向至某个目标网址（状态码为301），您可以通过**nginx.ingress.kubernetes.io/permanent-redirect**注解进行配置。例如将所有内容永久重定向到www.example.com：

```
nginx.ingress.kubernetes.io/permanent-redirect: https://www.example.com
```

在Nginx Ingress中的配置如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.example.com
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
```



```
        property:
          ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
        pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.example.com
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME         CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**     80      10s
```

步骤5 访问Ingress，其中${ELB_IP}为Nginx Ingress所绑定的ELB IP。

```
curl -I ${ELB_IP}
```

最终访问路径会被永久重定向至www.example.com。

```
HTTP/1.1 301 Moved Permanently
Date: Sat, 20 Jul 2024 07:55:49 GMT
Content-Type: text/html
Content-Length: 162
Connection: keep-alive
Location: https://www.example.com
```

----结束

配置永久重定向的返回状态码

配置永久重定向时，您可以通过**nginx.ingress.kubernetes.io/permanent-redirect-code**注解修改永久重定向的返回状态码。例如将永久重定向的状态码设置为308：

```
nginx.ingress.kubernetes.io/permanent-redirect-code: '308'
```

在Nginx Ingress中的配置如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```


1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.example.com
    nginx.ingress.kubernetes.io/permanent-redirect-code: '308'
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.example.com
    nginx.ingress.kubernetes.io/permanent-redirect-code: '308'
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤5 访问Ingress，其中${ELB_IP}为Nginx Ingress所绑定的ELB IP。

```
curl -I ${ELB_IP}
```

最终访问路径会被永久重定向至www.example.com，且状态码为308。

```
HTTP/1.1 308 Moved Permanently
Date: Sat, 20 Jul 2024 07:55:49 GMT
Content-Type: text/html
```



```
Content-Length: 162
Connection: keep-alive
Location: https://www.example.com
```

----结束

配置临时重定向规则

如果您想将访问请求临时重定向至某个目标网址（状态码为302），您可以通过**nginx.ingress.kubernetes.io/temporal-redirect**注解进行配置。例如将所有内容临时重定向到www.example.com：

```
nginx.ingress.kubernetes.io/temporal-redirect: https://www.example.com
```

在Nginx Ingress中的配置如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/temporal-redirect: https://www.example.com
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/temporal-redirect: https://www.example.com
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```


回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤5 访问Ingress，其中${ELB_IP}为Nginx Ingress所绑定的ELB IP。

```
curl -I ${ELB_IP}
```

最终访问路径会被临时重定向至www.example.com，且状态码为302。

```
HTTP/1.1 302 Moved Temporarily
Date: Sat, 20 Jul 2024 08:01:02 GMT
Content-Type: text/html
Content-Length: 138
Connection: keep-alive
Location: https://www.example.com
```

----结束

配置 HTTP 重定向到 HTTPS

默认情况下，若Ingress使用了TLS，通过HTTP访问时，请求将会被重定向（状态码为308）到HTTPS。您可以通过**nginx.ingress.kubernetes.io/ssl-redirect**注解进行配置，对应的参数值可设置为“true”或“false”：

- true：使用TLS证书时，将HTTP访问重定向至HTTPS（状态码为308）。
- false：使用TLS证书时，禁止将HTTP访问重定向至HTTPS。

如果您在没有使用TLS证书的情况下需要强制重定向到HTTPS，您可以使用**nginx.ingress.kubernetes.io/force-ssl-redirect: "true"**注释实现。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```



```
          pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: /
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤5 访问Ingress，其中${ELB_IP}为Nginx Ingress所绑定的ELB IP。

```
curl -I ${ELB_IP}
```

最终访问路径会被重定向至HTTPS（状态码为308）。

```
HTTP/1.1 308 Permanent Redirect
Date: Sat, 20 Jul 2024 08:03:36 GMT
Content-Type: text/html
Content-Length: 164
Connection: keep-alive
Location: https://${ELB_IP}
```

----结束

7.4.3.4.3 为 Nginx Ingress 配置 URL 重写规则

在一些使用场景中后端服务提供访问的URL与Ingress规则中指定的路径不同，而Ingress会将访问路径直接转发到后端相同路径，如果不进行URL重写配置，所有访问都将返回404。例如，Ingress规则中的访问路径设置为/app/demo，而后端服务提供的访问路径为/demo，在实际访问Ingress时会直接转发到后端服务的/app/demo路径，与后端实际提供的访问路径（/demo）不匹配，导致404的情况发生。

此时，您可以通过Rewrite方法实现URL重写，即使用“nginx.ingress.kubernetes.io/rewrite-target”注解可以实现不同路径的重写规则。

配置重写规则

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/something(/|$)(.*)'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/something(/|$)(.*)'
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

说明

只要有一个Ingress使用了rewrite-target，则所有Ingress定义下同一个host下所有path都会正则大小写敏感，包括没有使用rewrite-target的Ingress。

以上示例中，占位符$2表示将第二个括号即(.*)中匹配到的所有字符填写到“nginx.ingress.kubernetes.io/rewrite-target”注解中，作为重写后的URL路径。

例如，上面的Ingress正则匹配式将导致多种情况的URL重写，可能的情形如下：

- 访问“/something”路径重写为“/”路径
- 访问“/something/”路径重写为“/”路径
- 访问“/something/new”路径重写为“/new”路径

步骤3 创建Ingress。


```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

步骤5 访问Ingress，其中${ELB_IP}为Nginx Ingress所绑定的ELB IP。

```
curl ${ELB_IP}/something
```

最终访问路径会被重定向至“/”路径。

----结束

Ingress 配置文件说明

在nginx-ingress-controller容器中，“/etc/nginx”路径下的nginx.conf文件可查看所有Ingress配置。

步骤1 查询nginx-ingress-controller的Pod名称。

```
kubectl get pod -n kube-system | grep nginx-ingress-controller
```

回显如下：

```
cceaddon-nginx-ingress-controller-66855ccb9b-52qcd        1/1     Running   0          37m
```

步骤2 登录nginx-ingress-controller容器。

```
kubectl exec -it cceaddon-nginx-ingress-controller-66855ccb9b-52qcd -- /bin/bash
```

步骤3 查看“/etc/nginx”路径下的nginx.conf文件。

```
cat /etc/nginx/nginx.conf
```

本文示例中的重写规则将生成一条Rewrite指令，并写入到nginx.conf的location字段中，如下所示：

```
## start server _
    server {
        server_name _ ;
        ...
        location ~* "^/something(/|$)(.*)" {
            set $namespace      "default";
            set $ingress_name   "ingress-test";
            set $service_name   "<your_service_name>";
            set $service_port   "80";
            ...
            rewrite "(?i)/something(/|$)(.*)" /$2 break;
            ...
        }
    }
    ## end server _
```

上面的Rewrite指令基本语法结构为：

```
rewrite regex replacement [flag];
```

- regex：匹配URI的正则表达式。在上述例子中，“(?i)/something(/|$)(.*)”即为匹配URI的正则表达式，其中“(?i)”表示不区分大小写。
- replacement：重写内容。在上述例子中，“/$2”即为重写内容，表示把路径重写为第二个括号“(.*)”中匹配到的所有字符。

- flag：表示重写形式的标记，包括：
  - last：表示本条规则匹配完成后继续向下匹配。
  - break：表示本条规则匹配完成后停止匹配。
  - redirect：表示临时重定向，返回状态码302。
  - permanent：表示永久重定向，返回状态码301。

----结束

高级 Rewrite 配置

对于一些复杂高级的Rewrite需求，可以通过如下注解来实现，其本质也是修改Nginx的配置文件（nginx.conf），可以实现上面提到的“nginx.ingress.kubernetes.io/rewrite-target”注解的功能，但是自定义程度更高，适合更加复杂的Rewrite需求。

- nginx.ingress.kubernetes.io/server-snippet：在nginx.conf的“server”字段中添加自定义配置。
- nginx.ingress.kubernetes.io/configuration-snippet：在nginx.conf的“location”字段中添加自定义配置。

说明

snippet配置在NGINX Ingress控制器版本为2.4.6版本及以上时（对应社区版本为v1.9.3）不再默认启用，详情请参见**Changelog**。如果您仍需要使用snippet配置，可以通过**allow-snippet-annotations**启用。

通过以上两个注解可以在nginx.conf中的“server”或“location”字段中插入Rewrite指令，完成URL的重写，示例如下：

```
annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      rewrite ^/stylesheets/(.*)$ /something/stylesheets/$1 redirect;  # 添加 /something 前缀
      rewrite ^/images/(.*)$ /something/images/$1 redirect;  # 添加 /something 前缀
```

如上两条规则在访问URL中添加了“/something”路径前缀，即：

- 当用户访问“/stylesheets/new.css”路径时，重写为“/something/stylesheets/new.css”路径。
- 当用户访问“/images/new.jpg”路径时，重写为“/something/images/new.jpg”路径。

7.4.3.4.4 为 Nginx Ingress 配置 HTTPS 协议的后端服务

Ingress可以代理不同协议的后端服务，在默认情况下Ingress的后端代理通道是HTTP协议的，若需要建立HTTPS协议的通道，可在annotation字段中加入如下配置：

```
nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
```

Ingress配置示例如下：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
```



```
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  tls:
    - secretName: ingress-test-secret  #替换为您的TLS密钥证书
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  tls:
    - secretName: ingress-test-secret  #替换为您的TLS密钥证书
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

----结束

7.4.3.4.5 为 Nginx Ingress 配置一致性哈希负载均衡

原生的Nginx支持多种负载均衡规则，其中常用的有加权轮询、IP hash等。Nginx Ingress在原生的Nginx能力基础上，支持使用一致性哈希方法进行负载均衡。

Nginx默认支持的IP hash方法使用的是线性的hash空间，根据IP的hash运算值来选取后端的目标服务器。但是这种方法在添加删除节点时，所有IP值都需要重新进行hash

运算，然后重新路由，这样的话就会导致大面积的会话丢失或缓存失效，因此Nginx Ingress引入了一致性哈希来解决这一问题。

一致性哈希是一种特殊的哈希算法，通过构建环状的hash空间来替代普通的线性hash空间，在增删节点时仅需要将路由的目标按顺时针原则向下迁移，而其他路由无需改变，可以尽可能地减少重新路由，有效解决动态增删节点带来的负载均衡问题。

通过配置一致性哈希规则，在增加一台服务器时，新的服务器会尽量分担其他所有服务器的压力；同样，在减少一台服务器时，其他所有服务器也可以尽量分担它的资源，可以有效减少集群局部节点的压力，防止由于某一节点宕机带来的集群雪崩效应。

配置一致性哈希规则

Nginx Ingress可以通过“nginx.ingress.kubernetes.io/upstream-hash-by”注解实现一致性哈希规则的配置，如下所示：

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**ingress-test.yaml**”的YAML文件，此处文件名可自定义。

```
vi ingress-test.yaml
```

1.23及以上版本集群：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri"  #按照请求uri进行hash
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: <your_service_name>  #替换为您的目标服务名称
                port:
                  number: <your_service_port>  #替换为您的目标服务端口
            property:
              ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
            pathType: ImplementationSpecific
  ingressClassName: nginx
```

1.21及以下版本集群：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri"  #按照请求uri进行hash
spec:
  rules:
    - host: ''
      http:
        paths:
          - path: '/'
            backend:
              serviceName: <your_service_name>  #替换为您的目标服务名称
              servicePort: <your_service_port>  #替换为您的目标服务端口
```


注解“nginx.ingress.kubernetes.io/upstream-hash-by”的参数值支持nginx参数、文本值或任意组合，例如：

- nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri"代表按照请求uri进行hash。
- nginx.ingress.kubernetes.io/upstream-hash-by: "$request_uri$host"代表按照请求uri和域名进行hash。
- nginx.ingress.kubernetes.io/upstream-hash-by: "${request_uri}-text-value"代表按照请求uri和文本值进行hash。

步骤3 创建Ingress。

```
kubectl create -f ingress-test.yaml
```

回显如下，表示Ingress服务已创建。

```
ingress/ingress-test created
```

步骤4 查看已创建的Ingress。

```
kubectl get ingress
```

回显如下，表示Ingress服务创建成功。

```
NAME          CLASS   HOSTS     ADDRESS         PORTS   AGE
ingress-test  nginx   *         121.**.**.**    80      10s
```

----结束

相关文档

Custom NGINX upstream hashing

7.5 DNS

7.5.1 DNS 概述

CoreDNS 介绍

创建集群时会安装**CoreDNS插件**，CoreDNS是用来做集群内部域名解析。

在kube-system命名空间下可以查看到CoreDNS的Pod。

```
$ kubectl get po --namespace=kube-system
NAME                              READY   STATUS    RESTARTS   AGE
coredns-7689f8bdf-295rk           1/1     Running   0          9m11s
coredns-7689f8bdf-h7n68           1/1     Running   0          11m
```

CoreDNS安装成功后会成为DNS服务器，当创建Service后，CoreDNS会将Service的名称与IP记录起来，这样Pod就可以通过向CoreDNS查询Service的名称获得Service的IP地址。

访问时通过nginx.<namespace>.svc.cluster.local访问，其中nginx为Service的名称，<namespace>为命名空间名称，svc.cluster.local为域名后缀，在实际使用中，在同一个命名空间下可以省略<namespace>.svc.cluster.local，直接使用ServiceName即可。

使用ServiceName的方式有个主要的优点就是可以在开发应用程序时可以将ServiceName写在程序中，这样无需感知具体Service的IP地址。

CoreDNS插件安装后也有一个Service，在kube-system命名空间下，如下所示。

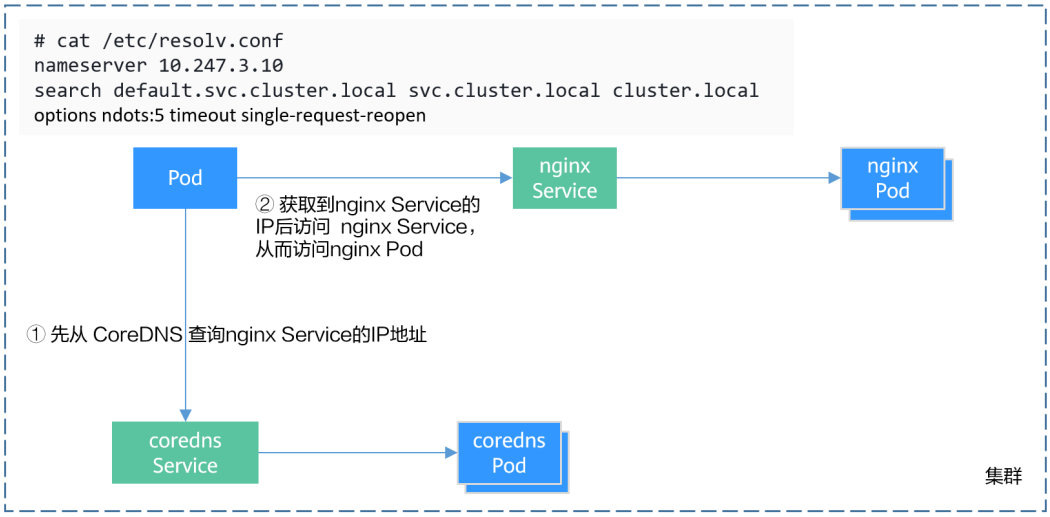
```
$ kubectl get svc -n kube-system
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
coredns       ClusterIP   10.247.3.10    <none>        53/UDP,53/TCP,8080/TCP   13d
```

默认情况下，其他Pod创建后，会将coredns Service的地址作为域名解析服务器的地址写在Pod的 /etc/resolv.conf 文件中，创建一个Pod，查看/etc/resolv.conf文件，如下所示。

```
$ kubectl exec test01-6cbbf97b78-krj6h -it -- /bin/sh
/ # cat /etc/resolv.conf
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5 timeout single-request-reopen
```

在Pod中访问nginx Pod的ServiceName:Port，会先从CoreDNS中解析出nginx Service的IP地址，然后再访问nginx Service的IP地址，从而访问到nginx Pod。

图 7-80 集群内域名解析示例图



Kubernetes 中的域名解析逻辑

DNS策略可以在每个pod基础上进行设置，目前，Kubernetes支持**Default**、**ClusterFirst**、**ClusterFirstWithHostNet**和**None**四种DNS策略，具体请参见**Service与Pod的DNS**。这些策略在pod-specific的**dnsPolicy**字段中指定。

- **“Default”**：如果dnsPolicy被设置为“Default”，则名称解析配置将从pod运行的节点继承。 自定义上游域名服务器和存根域不能够与这个策略一起使用。
- **“ClusterFirst”**：如果dnsPolicy被设置为“ClusterFirst”，任何与配置的集群域后缀不匹配的DNS查询（例如，www.kubernetes.io）将转发到从该节点继承的上游名称服务器。集群管理员可能配置了额外的存根域和上游DNS服务器。
- **“ClusterFirstWithHostNet”**：对于使用hostNetwork运行的Pod，您应该明确设置其DNS策略“ClusterFirstWithHostNet”。
- **“None”**：它允许Pod忽略Kubernetes环境中的DNS设置。应使用dnsConfigPod规范中的字段提供所有DNS设置 。

说明

- Kubernetes 1.10及以上版本，支持Default、ClusterFirst、ClusterFirstWithHostNet和None四种策略；低于Kubernetes 1.10版本，仅支持default、ClusterFirst和ClusterFirstWithHostNet三种。
- “Default”不是默认的DNS策略。如果dnsPolicy的Flag没有特别指明，则默认使用“**ClusterFirst**”。

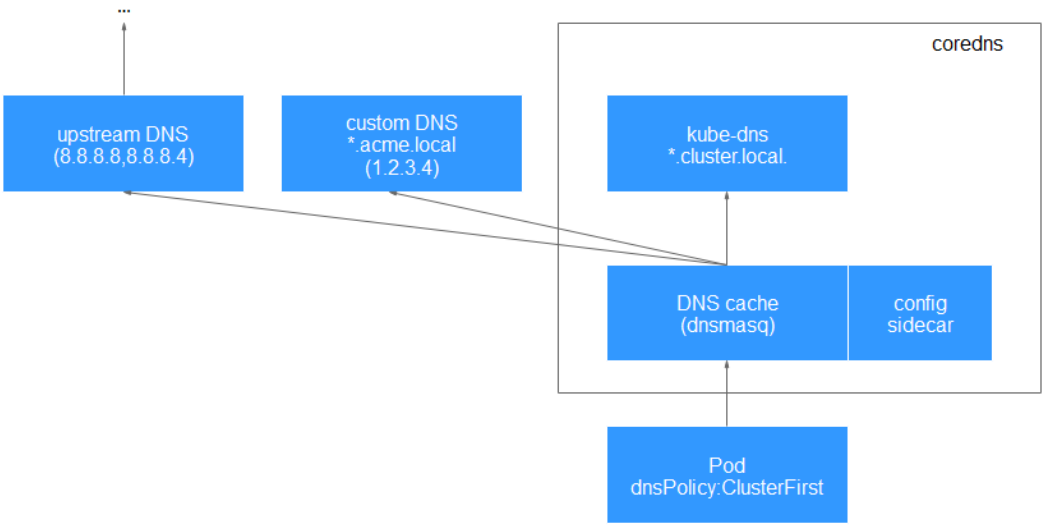
路由请求流程：

未配置存根域：没有匹配上配置的集群域名后缀的任何请求，例如“www.kubernetes.io”，将会被转发到继承自节点的上游域名服务器。

已配置存根域：如果配置了存根域和上游DNS服务器，DNS查询将基于下面的流程对请求进行路由：

1. 查询首先被发送到coredns中的DNS缓存层。
2. 从缓存层，检查请求的后缀，并根据下面的情况转发到对应的DNS上：
   - 具有集群后缀的名字（例如“.cluster.local”）：请求被发送到coredns。
   - 具有存根域后缀的名字（例如“.acme.local”）：请求被发送到配置的自定义DNS解析器（例如：监听在 1.2.3.4）。
   - 未能匹配上后缀的名字（例如“widget.com”）：请求被转发到上游DNS。

图 7-81 路由请求流程



相关操作

您还可以在工作负载中进行DNS配置，具体请参见**工作负载DNS配置说明**。

您还可以使用CoreDNS实现自定义域名解析，具体请参见**使用CoreDNS实现自定义域名解析**。

您还可以使用DNSCache提升DNS解析的性能，具体请参见**使用NodeLocal DNSCache提升DNS性能**。

7.5.2 工作负载 DNS 配置说明

Kubernetes集群内置DNS插件Kube-DNS/CoreDNS，为集群内的工作负载提供域名解析服务。业务在高并发调用场景下，如果使用到域名解析服务，可能会触及到Kube-

DNS/CoreDNS的性能瓶颈，导致DNS请求概率失败，影响用户业务正常运行。在Kubernetes使用的过程中，发现有些场景下工作负载的域名解析存在冗余的DNS查询，使得高并发场景更容易触及DNS的性能瓶颈。根据业务使用场景，对工作负载的DNS配置进行优化，能够在一定程度上减少DNS请求概率失败的问题。

更多DNS相关信息请参见**CoreDNS域名解析**。

DNS 配置项说明

在Linux系统的节点或者容器里执行cat /etc/resolv.conf命令，能够查看到DNS配置，以Kubernetes集群的容器DNS配置为例：

```
nameserver 10.247.x.x
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

配置项说明：

- nameserver：容器解析域名时查询的DNS服务器的IP地址列表。如果设置为10.247.x.x说明DNS对接到Kube-DNS/CoreDNS，如果是其他IP地址，则表示采用云上DNS或者用户自建的DNS。
- search：定义域名的搜索域列表，当访问的域名不能被DNS解析时，会把该域名与搜索域列表中的域依次进行组合，并重新向DNS发起请求，直到域名被正确解析或者尝试完搜索域列表为止。对于CCE集群来说，容器的搜索域列表配置3个域，当解析一个不存在的域名时，会产生8次DNS查询，因为对于每个域名需要查询两次，分别是IPv4和IPv6。
- options：定义域名解析配置文件的其他选项，常见的有timeout、ndots等等。

  Kubernetes集群容器的域名解析文件设置为options ndots:5，该参数的含义是当域名的“.”个数小于ndots的值，会先把域名与search搜索域列表进行组合后进行DNS查询，如果均没有被正确解析，再以域名本身去进行DNS查询。当域名的“.”个数大于或者等于ndots的值，会先对域名本身进行DNS查询，如果没有被正确解析，再把域名与search搜索域列表依次进行组合后进行DNS查询。

  如查询www.***.com域名时，由于该域名的“.”个数为2，小于ndots的值，所以DNS查询请求的顺序依次为：www.***.com.default.svc.cluster.local、www.***.com.svc.cluster.local、 www.***.com.cluster.local和 www.***.com，需要发起至少7次DNS查询请求才能解析出该域名的IP。可以看出，这种配置在访问外部域名时，存在大量冗余的DNS查询，存在优化点。

说明

完整的Linux域名解析文件配置项说明可以参考文档：**http://man7.org/linux/man-pages/man5/resolv.conf.5.html**。

通过控制台进行工作负载 DNS 配置

Kubernetes为应用提供了与DNS相关的配置选项，通过对应用进行DNS配置，能够在某些场景下有效地减少冗余的DNS查询，提升业务并发量。以下步骤以nginx应用为例，介绍如何通过控制台为工作负载添加DNS配置。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧选择“工作负载”，在右上角单击“创建工作负载”。

步骤2 设置工作负载基本参数，详情请参见**创建工作负载**。

步骤3 在“高级配置”中，选择“DNS配置”页签，并按需填写以下参数。

- DNS策略：控制台中提供的DNS策略与YAML中的dnsPolicy字段对应，详情请参见**表7-147**。

  - 追加域名解析配置：即dnsPolicy字段设置为ClusterFirst，此时容器中既能够解析service注册的集群内部域名，也能够解析发布到互联网上的外部域名。
  - 替换域名解析配置：即dnsPolicy字段设置为None，此时必须填写“IP地址”和“搜索域”参数。容器将仅使用自定义的IP地址和搜索域配置进行域名解析。
  - 继承Pod所在节点域名解析配置：即dnsPolicy字段设置为Default，此时容器将使用Pod所在节点的域名解析配置，无法解析集群内部域名。
- 可选对象：即**dnsConfig字段**中的options参数。每个对象可以具有name属性（必需）和value属性（可选），填写完成后需单击“确认添加”。
  - timeout：超时时间 (s)。
  - ndots：域名中必须出现的"."的个数。如果域名中的"."的个数不小于ndots，则该域名为一个全限定域名，操作系统会直接查询；如果域名中的"."的个数小于ndots，操作系统会在搜索域中进行查询。
- 域名解析服务器地址：即**dnsConfig字段**中的nameservers参数，您可对自定义的域名配置域名服务器，值为一个或一组DNS IP地址。
- 搜索域：即**dnsConfig字段**中的searches参数，表示域名查询时的DNS搜索域列表，此属性是可选的。指定后，提供的搜索域列表将合并到基于dnsPolicy生成的域名解析文件的search字段中，并删除重复的域名。
- 启用hostAliases：配置Pod的本地配置文件“/etc/hosts”，可以将域名和IP地址映射加入到hosts文件中，在本地系统中实现简单的域名解析。更多使用详情请参见**使用HostAliases向Pod /etc/hosts文件添加条目**。

图 7-82 工作负载 DNS 配置

步骤4 单击“创建工作负载”。

----结束

通过工作负载 YAML 进行 DNS 配置

您也可以通过YAML的方式创建工作负载，以nginx应用为例，其YAML文件中的DNS配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```



```
spec:
  containers:
    - name: container-1
      image: nginx:latest
      imagePullPolicy: IfNotPresent
  imagePullSecrets:
    - name: default-secret
  dnsPolicy: None
  dnsConfig:
    options:
      - name: ndots
        value: '5'
      - name: timeout
        value: '3'
    nameservers:
      - 10.2.3.4
    searches:
      - my.dns.search.suffix
```

- **dnsPolicy字段说明：**

  dnsPolicy字段是应用设置的DNS策略，默认值为“ClusterFirst”。dnsPolicy当前支持四种参数值：

  表 7-147 dnsPolicy 字段说明

参数	说明
ClusterFirst（默认值）	即在默认DNS配置中追加自定义的域名解析配置。应用会默认对接CoreDNS（CCE集群的CoreDNS默认级联云上DNS），自定义填写的dnsConfig会追加到默认DNS参数中。这种场景下，容器既能够解析service注册的集群内部域名，也能够解析发布到互联网上的外部域名。由于该配置下，域名解析文件设置了search搜索域列表和ndots: 5，因此当访问外部域名和集群内部长域名（如kubernetes.default.svc.cluster.local）时，大部分域名都会优先遍历search搜索域列表，导致至少有6次无效的DNS查询，只有访问集群内部短域名（如kubernetes）时，才不存在无效的DNS查询。

ClusterFirstWithHostNet	对于配置**主机网络（hostNetwork）**的应用，默认对接Pod所在节点域名解析配置，即kubelet的“--resolv-conf”参数指向的域名解析文件（CCE集群在该配置下对接云上DNS）。如需对接集群的Kube-DNS/CoreDNS，dnsPolicy字段需设置为ClusterFirstWithHostNet，此时容器的域名解析文件配置与“ClusterFirst”一致，也存在无效的DNS查询。
	`...`
	`spec:`
	`  containers:`
	`  - image: nginx:latest`
	`    imagePullPolicy: IfNotPresent`
	`    name: container-1`
	`  restartPolicy: Always`
	`  **hostNetwork: true**`
	`  dnsPolicy: **ClusterFirstWithHostNet**`

Default	即继承Pod所在节点域名解析配置，并在此基础上追加自定义的域名解析配置。容器的域名解析文件使用kubelet的“--resolv-conf”参数指向的域名解析文件（CCE集群在该配置下对接云上DNS），没有配置search搜索域列表和options。该配置只能解析注册到互联网上的外部域名，无法解析集群内部域名，且不存在无效的DNS查询。

参数	说明
None	即替换默认的域名解析配置，完全使用自定义的域名解析配置。设置为None之后，必须设置dnsConfig字段，此时容器的域名解析文件将完全通过dnsConfig的配置来生成。

说明

此处如果dnsPolicy字段未被指定，其默认值为ClusterFirst，而不是Default。

- **dnsConfig字段说明：**

  dnsConfig为应用设置DNS参数，设置的参数将合并到基于dnsPolicy策略生成的域名解析文件中。当dnsPolicy为“None”，应用的域名解析文件完全由dnsConfig指定；当dnsPolicy不为“None”时，会在基于dnsPolicy生成的域名解析文件的基础上，追加dnsConfig中配置的dns参数。

  表 7-148 dnsConfig 字段说明

参数	说明
options	DNS的配置选项，其中每个对象可以具有name属性（必需）和value属性（可选）。该字段中的内容将合并到基于dnsPolicy生成的域名解析文件的options字段中，dnsConfig的options的某些选项如果与基于dnsPolicy生成的域名解析文件的选项冲突，则会被dnsConfig所覆盖。
nameservers	DNS的IP地址列表。当应用的dnsPolicy设置为“None”时，列表必须至少包含一个IP地址，否则此属性是可选的。列出的DNS的IP列表将合并到基于dnsPolicy生成的域名解析文件的nameserver字段中，并删除重复的地址。 **说明** 容器DNS配置文件中nameserver最多可设置3个DNS地址： • 当dnsPolicy设置为ClusterFirst时，如果集群使用**CoreDNS**，除CoreDNS地址外可追加2个自定义DNS地址，超出部分无效。 • 当dnsPolicy设置为ClusterFirst时，如果集群同时使用**CoreDNS**和**NodeLocal DNSCache**，除CoreDNS和NodeLocal DNSCache地址外可追加1个自定义DNS地址，超出部分无效。
searches	域名查询时的DNS搜索域列表，此属性是可选的。指定后，提供的搜索域列表将合并到基于dnsPolicy生成的域名解析文件的search字段中，并删除重复的域名。Kubernetes最多允许6个搜索域。

工作负载的 DNS 配置实践

前面介绍了Linux系统域名解析文件以及Kubernetes为应用提供的DNS相关配置项，下面将举例介绍应用如何进行DNS配置。

- **场景1 对接Kubernetes内置的Kube-DNS/CoreDNS**

  场景说明：

这种方式适用于应用中的域名解析只涉及集群内部域名，或者集群内部域名+外部域名两种方式，应用默认采用这种配置。

示例：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: ClusterFirst
  imagePullSecrets:
    - name: default-secret
```

该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

- **场景2 直接对接云DNS**

  场景说明：

  这种方式适用于应用只访问注册到互联网的外部域名，该场景不能解析集群内部域名。

  示例：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: Default  #使用kubelet的“--resolv-conf”参数指向的域名解析文件（CCE集群在该配置下对接云DNS）
  imagePullSecrets:
    - name: default-secret
```

  该配置下容器的域名解析文件将如下所示：

```
nameserver 100.125.x.x
```

- **场景3 主机网络模式的应用对接Kube-DNS/CoreDNS**

  场景说明：

  对于配置主机网络模式的应用，默认对接云DNS，如果应用需要对接Kube-DNS/CoreDNS，需将dnsPolicy设置为“ClusterFirstWithHostNet”。

  示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  containers:
  - name: nginx
    image: nginx:alpine
    ports:
    - containerPort: 80
  imagePullSecrets:
    - name: default-secret
```


该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

- **场景4 自定义应用的域名配置**

  场景说明：

  用户可以完全自定义配置应用的域名解析文件，这种方式非常灵活，dnsPolicy和dnsConfig配合使用，几乎能够满足所有使用场景，如对接用户自建DNS的场景、串联多个DNS的场景以及优化DNS配置选项的场景等等。

  示例1：对接用户自建DNS

  该配置下，dnsPolicy为“None”，应用的域名解析文件完全根据dnsConfig配置生成。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
    - 10.2.3.4  #用户自建DNS的IP地址
    searches:
    - ns1.svc.cluster.local
    - my.dns.search.suffix
    options:
    - name: ndots
      value: "2"
    - name: timeout
      value: "3"
  imagePullSecrets:
    - name: default-secret
```

  该配置下容器的域名解析文件将如下所示：

```
nameserver 10.2.3.4
search ns1.svc.cluster.local my.dns.search.suffix
options timeout:3 ndots:2
```

  示例2：修改域名解析文件的ndots选项，减少无效的DNS查询

  该配置下，dnsPolicy不为“None”，会在基于dnsPolicy生成的域名解析文件的基础上，追加dnsConfig中配置的dns参数。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: "ClusterFirst"
  dnsConfig:
    options:
    - name: ndots
      value: "2" #该配置会将基于ClusterFirst策略生成的域名解析文件的ndots:5参数改写为ndots:2
  imagePullSecrets:
    - name: default-secret
```

  该配置下容器的域名解析文件将如下所示：


```
nameserver 10.247.3.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:2
```

示例3：串联使用多个DNS

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
  - name: test
    image: nginx:alpine
  dnsPolicy: ClusterFirst  # 追加域名解析配置，集群中会默认对接CoreDNS
  dnsConfig:
    nameservers:
    - 10.2.3.4 # 用户自建DNS的IP地址
  imagePullSecrets:
    - name: default-secret
```

说明

容器DNS配置文件中nameserver最多可设置3个DNS地址：

- 当dnsPolicy设置为ClusterFirst时，如果集群使用CoreDNS，除CoreDNS地址外可追加2个自定义DNS地址，超出部分无效。
- 当dnsPolicy设置为ClusterFirst时，如果集群同时使用CoreDNS和NodeLocal DNSCache，除CoreDNS和NodeLocal DNSCache地址外可追加1个自定义DNS地址，超出部分无效。

该配置下容器的域名解析文件将如下所示：

```
nameserver 10.247.3.10 10.2.3.4
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

7.5.3 使用 CoreDNS 实现自定义域名解析

应用现状

在使用CCE时，可能会有解析自定义内部域名的需求，例如：

- 存量代码配置了用固定域名调用内部其他服务，如果要切换到Kubernetes Service方式，修改配置工作量大。
- 在集群外自建了一个其他服务，需要将集群中的数据通过固定域名发送到这个服务。

解决方案

使用CoreDNS有以下几种自定义域名解析的方案。

- 为CoreDNS配置存根域：为特定域名指定域名解析服务器，可以直接在控制台添加，简单易操作。
- 使用 CoreDNS Hosts 插件配置任意域名解析：为特定域名配置本地解析记录，简单直观，可以添加任意解析记录，类似在本地/etc/hosts中添加解析记录。
- 使用 CoreDNS Rewrite 插件指向域名到集群内服务：在集群内对域名进行CNAME解析，将一个域名指向另一个集群内域名，相当于给Kubernetes中的Service名称取了个别名，无需提前知道解析记录的IP地址。

- **使用 CoreDNS Forward 插件将自建 DNS 设为上游 DNS**：完全使用自建DNS作为外部域名解析服务器。自建DNS中，可以管理大量的解析记录，解析记录专门管理，增删记录无需修改CoreDNS配置。

注意事项

CoreDNS修改配置需额外谨慎，因为CoreDNS负责集群的域名解析任务，修改不当可能会导致集群解析出现异常。请做好修改前后的测试验证。

为 CoreDNS 配置存根域

集群管理员可以修改CoreDNS Corefile的ConfigMap以更改服务发现的工作方式。

若集群管理员有一个位于10.150.0.1的Consul域名解析服务器，并且所有Consul的域名都带有.consul.local的后缀。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“插件中心”，在CoreDNS下单击“编辑”，进入插件详情页。

步骤3 在“参数配置”下添加存根域。格式为一个键值对，键为DNS后缀域名，值为一个或一组DNS IP地址，如 'consul.local --10.150.0.1'。

图 7-83 添加存根域

对应Corefile内容如下：

```
.:5353 {
    bind {$POD_IP}
    cache 30 {
        servfail 5s
    }
    errors
    health {$POD_IP}:8080
    kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
    }
    loadbalance round_robin
    prometheus {$POD_IP}:9153
    forward . /etc/resolv.conf {
        policy random
    }
    reload
    ready {$POD_IP}:8081
}
consul.local:5353 {
    bind {$POD_IP}
    errors
    cache 30
    forward . 10.150.0.1
}
```

步骤4 单击“确定”完成配置更新。

步骤5 在左侧导航栏中选择“配置与密钥”，在“kube-system”命名空间下，查看名为coredns的配置项数据，确认是否更新成功。

----结束

修改 CoreDNS Hosts 配置

在CoreDNS中修改hosts后，可以不用单独在每个Pod中配置hosts添加解析记录。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“插件中心”，在CoreDNS下单击“编辑”，进入插件详情页。

步骤3 在“参数配置”下编辑扩展参数，在plugins字段添加以下内容。

```
{
  "configBlock": "192.168.1.1 www.example.com\nfallthrough",
  "name": "hosts"
}
```

须知

此处配置不能遗漏fallthrough字段，fallthrough表示当在hosts找不到要解析的域名时，会将解析任务传递给CoreDNS的下一个插件。如果不写fallthrough的话，任务就此结束，不会继续解析，会导致集群内部域名解析失败的情况。

hosts的详细配置请参见https://coredns.io/plugins/hosts/。

图 7-84 修改 CoreDNS Hosts 配置

对应Corefile内容如下：

```
.:5353 {
    bind {$POD_IP}
    hosts {
      192.168.1.1 www.example.com
      fallthrough
    }
    cache 30
    errors
    health {$POD_IP}:8080
    kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
    }
    loadbalance round_robin
    prometheus {$POD_IP}:9153
```



```
    forward . /etc/resolv.conf {
       policy random
    }
    reload
    ready {$POD_IP}:8081
}
```

步骤4 单击“确定”完成配置更新。

步骤5 在左侧导航栏中选择“配置与密钥”，在“kube-system”命名空间下，查看名为coredns的配置项数据，确认是否更新成功。

----结束

添加 CoreDNS Rewrite 配置指向域名到集群内服务

使用 CoreDNS 的 Rewrite 插件，将指定域名解析到某个 Service 的域名。例如，将访问example.com域名的请求重新指向到example.default.svc.cluster.local域名，即指向到default命名空间下的example服务。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“插件中心”，在CoreDNS 域名解析下单击“编辑”，进入插件详情页。

步骤3 在“参数配置”下编辑扩展参数，在plugins字段添加以下内容。

```
{
  "name": "rewrite",
  "parameters": "name example.com example.default.svc.cluster.local"
}
```

图 7-85 添加 CoreDNS Rewrite 配置指向域名到集群内服务

对应Corefile内容如下：

```
.:5353 {
    bind {$POD_IP}
    rewrite name example.com example.default.svc.cluster.local
    cache 30
    errors
    health {$POD_IP}:8080
    kubernetes cluster.local in-addr.arpa ip6.arpa {
       pods insecure
       fallthrough in-addr.arpa ip6.arpa
    }
    loadbalance round_robin
    prometheus {$POD_IP}:9153
    forward . /etc/resolv.conf {
       policy random
```



```
    }
    reload
    ready {$POD_IP}:8081
}
```

步骤4 单击“确定”完成配置更新。

步骤5 在左侧导航栏中选择“配置与密钥”，在“kube-system”命名空间下，查看名为coredns的配置项数据，确认是否更新成功。

----结束

使用 CoreDNS 级联自建 DNS

当域名不在Kubernetes域时，CoreDNS默认使用节点的/etc/resolv.conf文件进行解析，您也可以修改成外部DNS的解析地址。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“插件中心”，在CoreDNS 域名解析下单击“编辑”，进入插件详情页。

步骤3 在“参数配置”下编辑扩展参数，在plugins字段修改以下内容。

```
{
    "configBlock": "policy random",
    "name": "forward",
    "parameters": ". 192.168.1.1"
}
```

图 7-86 使用 CoreDNS 级联自建 DNS

对应Corefile内容如下：

```
.:5353 {
    bind {$POD_IP}
    cache 30
    errors
    health {$POD_IP}:8080
    kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
    }
    loadbalance round_robin
    prometheus {$POD_IP}:9153
    forward . 192.168.1.1 {
        policy random
    }
    reload
    ready {$POD_IP}:8081
}
```


步骤4 单击“确定”完成配置更新。

步骤5 在左侧导航栏中选择“配置与密钥”，在“kube-system”命名空间下，查看名为coredns的配置项数据，确认是否更新成功。

----结束

7.5.4 使用 NodeLocal DNSCache 提升 DNS 性能

应用现状

当集群中的DNS请求量增加时，CoreDNS将会承受更大的压力，可能会导致如下影响：

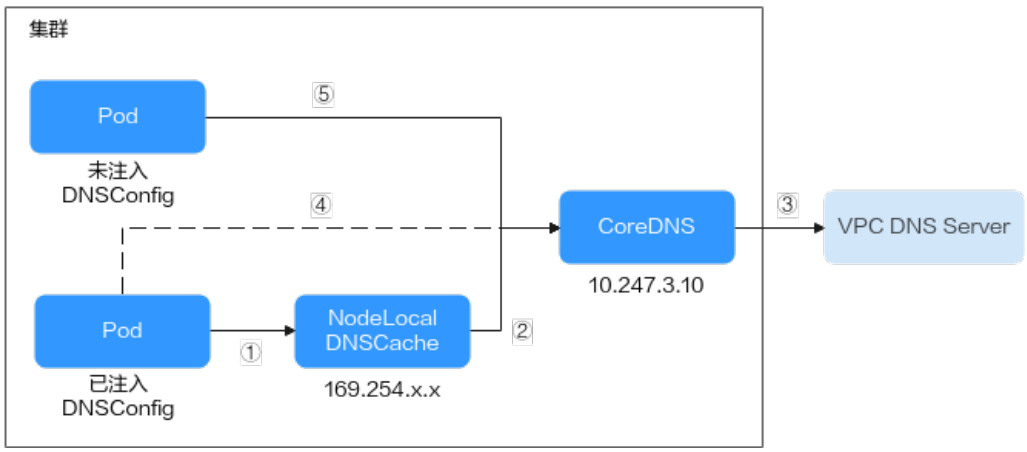
- 延迟增加：CoreDNS需要处理更多的请求，可能会导致DNS查询变慢，从而影响业务性能。
- 资源占用率增加：为保证DNS性能，CoreDNS往往需要更高规格的配置。

解决方案

为了避免DNS延迟的影响，可以在集群中部署NodeLocal DNSCache来提升服务发现的稳定性和性能。NodeLocal DNSCache会在集群节点上运行DNS缓存代理，所有注入DNS配置的Pod都会使用节点上运行的DNS缓存代理进行域名解析，而不是使用CoreDNS服务，以此来减少CoreDNS服务的压力，提高集群DNS性能。

启用NodeLocal DNSCache之后，DNS查询所遵循的路径如下图所示。

图 7-87 NodeLocal DNSCache 查询路径



其中解析线路说明如下：

- ①：已注入DNS本地缓存的Pod，默认会通过NodeLocal DNSCache解析请求域名。
- ②：NodeLocal DNSCache本地缓存如果无法解析请求，则会请求集群CoreDNS进行解析。
- ③：对于非集群内的域名，CoreDNS会通过VPC的DNS服务器进行解析。
- ④：已注入DNS本地缓存的Pod，如果无法连通NodeLocal DNSCache，则会直接通过CoreDNS解析域名。

- ⑤：未注入DNS本地缓存的Pod，默认会通过CoreDNS解析域名。

约束与限制

- **节点本地域名解析加速**插件仅支持1.19及以上版本集群。
- CCI上不支持NodeLocal DNSCache。当负载动态弹性伸缩到CCI时，CCI上Pod会因为无法联通NodeLocal DNSCache导致域名解析超时。因此创建弹性到CCI的负载时，需为Pod添加**node-local-dns-injection: disabled**的标签，避免DNSConfig自动注入，详情请参见**常见问题**。
- **node-local-dns-injection**标签为NodeLocal DNSCache使用的系统标签，除**避免DNSConfig自动注入**的场景外，应避免使用该标签。

插件安装

CCE提供了**节点本地域名解析加速**插件，可以快捷安装NodeLocal DNSCache。

说明

NodeLocal DNSCache不提供Hosts、Rewrite等插件能力，仅作为CoreDNS的透明缓存代理。如有需要，可在CoreDNS配置中修改。

步骤1 （可选）修改CoreDNS配置，让CoreDNS优先采用UDP协议与上游DNS服务器通信。

NodeLocal DNSCache采用TCP协议与CoreDNS进行通信，CoreDNS会根据请求来源使用的协议与上游DNS服务器进行通信。当使用了NodeLocal DNSCache时，访问上游DNS服务器时会使用TCP协议，而云上DNS服务器对TCP协议支持有限，如果您使用了NodeLocal DNSCache，您需要修改CoreDNS配置，让其总是优先采用UDP协议与上游DNS服务器进行通信，避免解析异常。

执行如下步骤，在forward插件中指定请求上游的协议为perfer_udp，修改之后CoreDNS会优先使用UDP协议与上游通信。

1. 登录CCE控制台，单击集群名称进入集群。
2. 在左侧导航栏中选择“插件中心”，在CoreDNS下单击“编辑”，进入插件详情页。
3. 在“参数配置”下编辑扩展参数，在plugins字段修改以下内容。

```
{
    "configBlock": "prefer_udp",
    "name": "forward",
    "parameters": ". /etc/resolv.conf"
}
```

如果使用Corefile视图，则对应的Corefile为：

```
Corefile: |-
  .:5353 {
      bind {$POD_IP}
      cache 30 {
          servfail 5s
      }
      errors
      health {$POD_IP}:8080
      kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
      }
      loadbalance round_robin
      prometheus {$POD_IP}:9153
      forward . /etc/resolv.conf {
          prefer_udp
      }
```



```
    reload
    ready {$POD_IP}:8081
}
```

步骤2 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**节点本地域名解析加速**插件，单击“安装”。

步骤3 在安装插件页面，选择插件规格，并配置相关参数。

- DNSConfig自动注入：启用后，会创建DNSConfig动态注入控制器，该控制器基于Admission Webhook机制拦截目标命名空间（即命名空间包含标签node-local-dns-injection=enabled）下Pod的创建请求，自动为Pod配置DNSConfig。未开启DNSConfig自动注入或Pod属于非目标命名空间，则需要手动给Pod配置DNSConfig。

  开启自动注入后，您可以为DNSConfig自定义以下配置项（插件版本为1.6.7及以上支持）：

  说明

  如果开启自动注入时Pod中已经配置了DNSConfig，则优先使用Pod中的DNSConfig。

  - 域名解析服务器地址nameserver（可选）：容器解析域名时查询的DNS服务器的IP地址列表。默认会添加NodeLocal DNSCache的地址，以及CoreDNS的地址，允许用户额外追加1个地址，重复的IP地址将被删除。
  - 搜索域search（可选）：定义域名的搜索域列表，当访问的域名不能被DNS解析时，会把该域名与搜索域列表中的域依次进行组合，并重新向DNS发起请求，直到域名被正确解析或者尝试完搜索域列表为止。允许用户额外追加3个搜索域，重复的域名将被删除。
  - ndots（可选）：该参数的含义是当域名的“.”个数小于ndots的值，会先把域名与search搜索域列表进行组合后进行DNS查询，如果均没有被正确解析，再以域名本身去进行DNS查询。当域名的“.”个数大于或者等于ndots的值，会先对域名本身进行DNS查询，如果没有被正确解析，再把域名与search搜索域列表依次进行组合后进行DNS查询。
- 目标命名空间：启用DNSConfig自动注入时支持设置。仅1.3.0及以上版本的插件支持。
  - 全部开启：CCE会为已创建的命名空间添加标签（node-local-dns-injection=enabled），同时会识别命名空间的创建请求并自动添加标签，这些操作的目标不包含系统内置的命名空间（如kube-system）。
  - 手动配置：手动为需要注入DNSConfig的命名空间添加标签（node-local-dns-injection=enabled），操作步骤请参见**管理命名空间标签**。

步骤4 完成以上配置后，单击“安装”。

----结束

使用 NodeLocal DNSCache

默认情况下，应用的请求会通过CoreDNS代理，如果需要使用node-local-dns进行DNS缓存代理，您有以下几种方式可以选择：

- 自动注入：创建Pod时自动配置Pod的dnsConfig字段。（kube-system等系统命名空间下的Pod不支持自动注入）
- 手动配置：手动配置Pod的dnsConfig字段，从而使用NodeLocal DNSCache。

自动注入

开启自动注入的操作步骤如下：

步骤1 安装节点本地域名解析加速插件，并开启“DNSConfig自动注入”，详情请参见**插件安装**。

步骤2 使用kubectl连接集群，为命名空间添加node-local-dns-injection=enabled标签。例如，为default命名空间添加该标签的命令如下：

```
kubectl label namespace default  node-local-dns-injection=enabled
```

步骤3 在开启自动注入的命名空间中，任意创建一个工作负载。操作步骤详情请参见**创建无状态负载（Deployment）**。

须知

开启自动注入的Pod需满足以下要求：

- 新建Pod不位于kube-system和kube-public等系统命名空间。
- 新建Pod没有被打上禁用DNS注入的标签node-local-dns-injection=disabled。
- 新建Pod的DNSPolicy设置为ClusterFirstWithHostNet，或Pod未使用hostNetwork且DNSPolicy为ClusterFirst。

步骤4 查看该Pod的配置。

```
kubectl get pod <pod_name> -oyaml
```

开启自动注入后，创建的Pod会自动添加如下dnsConfig字段，其中nameservers字段中除了NodeLocal DNSCache的地址（169.254.20.10）外，还添加了CoreDNS的地址（10.247.3.10），保障了业务DNS请求高可用。

```
...
  dnsConfig:
    nameservers:
      - 169.254.20.10
      - 10.247.3.10
    searches:
      - default.svc.cluster.local
      - svc.cluster.local
      - cluster.local
    options:
      - name: timeout
        value: ''
      - name: ndots
        value: '5'
      - name: single-request-reopen
...
```

----结束

手动配置

手动配置即自行给Pod加上dnsConfig配置。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 创建一个Pod，并在dnsConfig中的nameservers配置中添加NodeLocal DNSCache的地址（169.254.20.10）。

说明

不同集群类型的NodeLocal DNSCache地址如下：

- CCE Standard集群：169.254.20.10
- CCE Turbo集群：169.254.1.1

创建nginx.yaml文件，示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
  dnsConfig:
    nameservers:
    - 169.254.20.10
    - 10.247.3.10
    searches:
    - default.svc.cluster.local
    - svc.cluster.local
    - cluster.local
    options:
    - name: ndots
      value: '2'
  imagePullSecrets:
  - name: default-secret
```

步骤3 执行以下命令创建Pod。

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

----结束

常见问题

- 如何让指定Pod避免自动注入DNSConfig?

  解决方案：

  如果某个工作负载需要避免DNSConfig自动注入，可在Pod模板的labels字段中添加**node-local-dns-injection: disabled**的标签。示例如下：

  ```
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: test
    namespace: default
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: test
    template:
      metadata:
        labels:
          app: test
          node-local-dns-injection: disabled   # 避免DNSConfig自动注入
      spec:
        containers:
          - name: container-1
            image: nginx:latest
            imagePullPolicy: IfNotPresent
        imagePullSecrets:
          - name: default-secret
  ```


7.6 集群网络配置

7.6.1 扩展集群 VPC 网段

操作场景

在创建集群时会选择集群位于某个VPC内，如果VPC规划太小出现没有足够可用IP时，您可以采用VPC扩展网段满足业务扩容需求。本文介绍如何使用VPC扩展网段扩充集群网段。

约束与限制

仅支持v1.21及以上版本的CCE Standard集群和CCE Turbo集群。

扩展网段规划说明

在添加扩展网段前，需做好网段规划，避免造成网段冲突。注意以下几点：

1. 集群所在VPC下所有子网（包括扩展网段子网）不能和容器网段、服务网段冲突。
2. 扩展网段选择10.0.0.0/8、172.16.0.0/12、192.168.0.0/16可能与集群Master分配的IP冲突，尽量避免选择这三个网段作为扩展网段。
3. 同VPC的非集群内ECS，如果需要和集群互访，访问会做SNAT， Pod源地址是节点IP而非Pod IP。
4. 如果扩展网段没添加过集群节点，那扩展网段的ECS不能访问集群内Pod；扩展网段添加集群节点后，扩展网段的ECS可以访问集群内Pod。

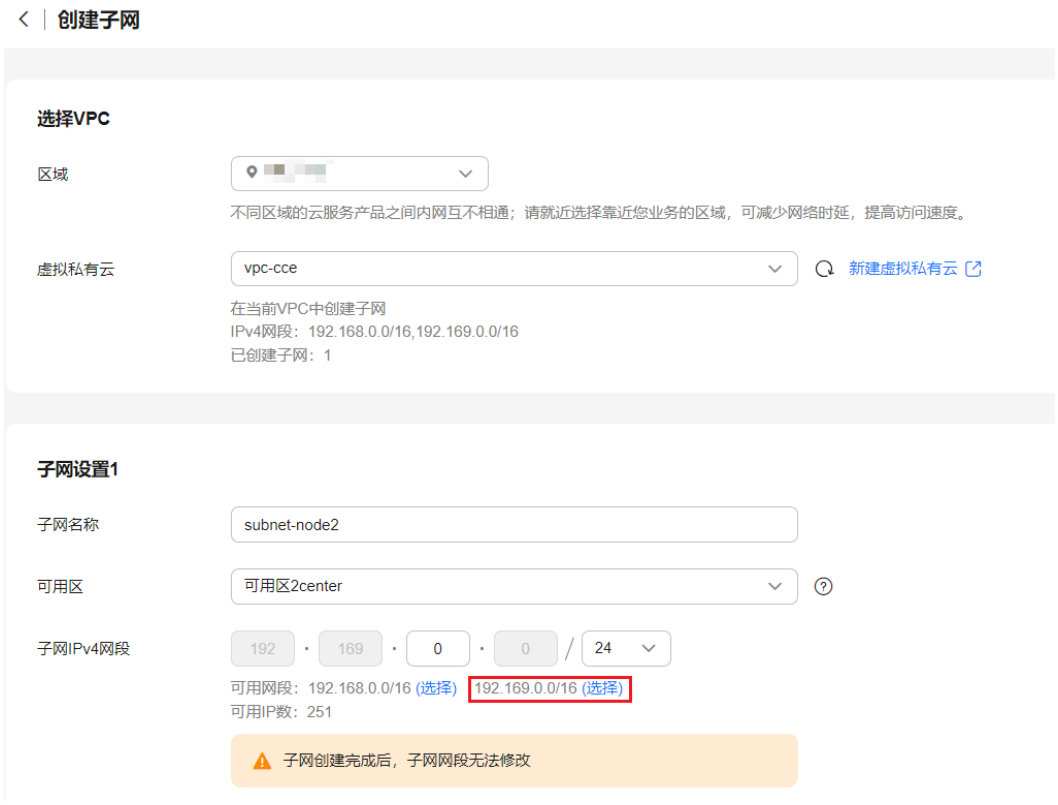
操作步骤

步骤1 登录VPC控制台，在左侧导航栏选择“虚拟私有云 > 我的VPC”，在集群所属VPC的“操作”区域, 单击编辑网段“编辑网段”，单击“添加IPv4扩展网段”。

输入需要添加的扩展网段，例如192.169.0.0/16。

图 7-88 添加 IPv4 扩展网段

步骤2 在左侧导航栏选择“虚拟私有云 > 子网”，单击“创建子网”，在“子网IPv4网段”选择新添加的IPv4扩展网段，其余参数根据规划配置，单击“确定”。即可在扩展网段下创建新的子网，供集群使用。



步骤3 扩展网段子网创建完成后，您可以在创建节点或节点池时，在“网络配置”中选择扩展网段子网。

----结束

7.7 容器如何访问 VPC 内部网络

前面章节介绍了使用Service和Ingress访问容器，本节将介绍如何从容器访问内部网络（**VPC内集群外**），包括VPC内访问和跨VPC访问。

VPC 内访问

根据集群容器网络模型不同，从容器访问内部网络有不同表现。

- **容器隧道网络**

  容器隧道网络在节点网络基础上通过隧道封装网络数据包，容器访问同VPC下其他资源时，只要节点能访问通，容器就能访问通。如果访问不通，需要确认对端资源的安全组配置是否能够允许容器所在节点访问。

- **云原生网络2.0**

  云原生网络2.0模型下，容器直接从VPC网段内分配IP地址，容器网段是节点所在VPC的子网，容器与VPC内其他地址天然能够互通。如果访问不通，需要确认对端资源的安全组配置是否能够允许容器网段访问。

- **VPC网络**

  VPC网络使用了VPC路由功能来转发容器的流量，容器网段与节点VPC不在同一个网段，容器访问同VPC下其他资源时，**需要对端资源的安全组能够允许容器网段访问**。

  例如集群节点所在网段为192.168.10.0/24，容器网段为172.16.0.0/16。

  VPC下（集群外）有一个地址为192.168.10.52的ECS，其安全组规则仅允许集群节点的IP网段访问。

  此时如果从容器中ping 192.168.10.52，会发现无法ping通。

```
kubectl exec test01-6cbbf97b78-krj6h  -it -- /bin/sh
/ # ping 192.168.10.25
PING 192.168.10.25 (192.168.10.25): 56 data bytes
^C
--- 192.168.10.25 ping statistics ---
104 packets transmitted, 0 packets received, 100% packet loss
```

  在安全组放通容器网段172.16.0.0/16访问。

  此时再从容器中ping 192.168.10.52，会发现可以ping通。

```
$ kubectl exec test01-6cbbf97b78-krj6h  -it -- /bin/sh
/ # ping 192.168.10.25
PING 192.168.10.25 (192.168.10.25): 56 data bytes
64 bytes from 192.168.10.25: seq=0 ttl=64 time=1.412 ms
64 bytes from 192.168.10.25: seq=1 ttl=64 time=1.400 ms
64 bytes from 192.168.10.25: seq=2 ttl=64 time=1.299 ms
64 bytes from 192.168.10.25: seq=3 ttl=64 time=1.283 ms
^C
--- 192.168.10.25 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
```

跨 VPC 访问

跨VPC访问通常采用对等连接等方法打通VPC。

- 容器隧道网络只需将节点网络与对端VPC打通，容器自然就能访问对端VPC。
- 云原生网络2.0与容器隧道网络类似，将容器所在子网网段与对端VPC打通即可。
- VPC网络由于容器网段独立，除了要打通VPC网段，还要打通容器网段。

  例如有如下两个VPC。

- vpc-demo：网段为192.168.0.0/16，集群在vpc-demo内，容器网段为10.0.0.0/16。
- vpc-demo2：网段为10.1.0.0/16。

创建一个名为peering-demo的对等连接（本端为vpc-demo，对端为vpc-demo2），注意对端VPC的路由添加容器网段。

这样配置后，在vpc-demo2中就能够访问容器网段10.0.0.0/16。具体访问时要关注安全组配置，打通端口配置。

访问其他云服务

与CCE进行内网通信的与服务常见服务有：RDS、DCS、Kafka、RabbitMQ、ModelArts等。

访问其他云服务除了上面所说的**VPC内访问**和**跨VPC访问**的网络配置外，还**需要关注所访问的云服务是否允许外部访问**，如DCS的Redis实例，需要添加白名单才允许访问。通常这些云服务会允许同VPC下IP访问，但是VPC网络模型下容器网段与VPC网段不同，需要特殊处理，将容器网段加入到白名单中。

容器访问内网不通的定位方法

如前所述，从容器中访问内部网络不通的情况可以按如下路径排查：

1. 查看要访问的对端服务器安全组规则，确认是否允许容器访问。
   - 容器隧道网络模型需要放通容器所在节点的IP地址
   - VPC网络模型需要放通容器网段
   - 云原生网络2.0需要放通容器所在子网网段
2. 查看要访问的对端服务器是否设置了白名单，如DCS的Redis实例，需要添加白名单才允许访问。添加容器和节点网段到白名单后可解决问题。
3. 查看要访问的对端服务器上是否安装了容器引擎，是否存在与CCE中容器网段冲突的情况。如果有网络冲突，会导致无法访问。

7.8 从容器访问公网

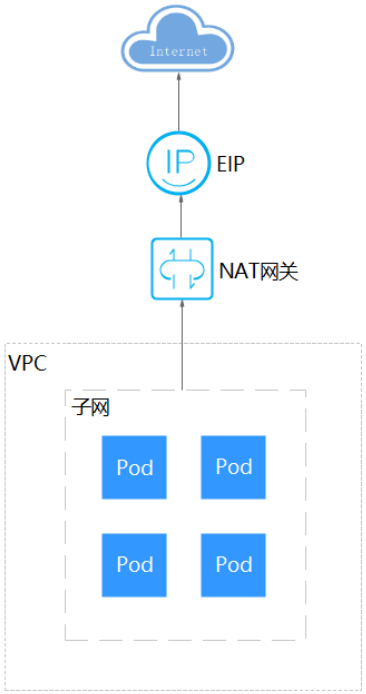
容器访问公网有如下方法可以实现。

- 给容器所在节点绑定弹性公网IP（容器网络模型为VPC网络或容器隧道网络）。
- 给Pod IP绑定弹性公网IP（仅支持云原生2.0网络模型集群，在VPC控制台中手动为Pod的弹性网卡或辅助弹性网卡绑定弹性公网IP。不推荐使用，因为Pod被重调度后IP会变化导致新的Pod无法访问公网）。

- 通过NAT网关配置SNAT规则，通过NAT网关访问公网。

下面将详细讲解通过NAT网关访问公网的方法，NAT网关能够为VPC内的容器实例提供网络地址转换（Network Address Translation）服务，SNAT功能通过绑定弹性公网IP，实现私有IP向公有IP的转换，可实现VPC内的容器实例共享弹性公网IP访问Internet。其原理如图7-89所示。通过NAT网关的SNAT功能，即使VPC内的容器实例不配置弹性公网IP也可以直接访问Internet，提供超大并发数的连接服务，适用于请求量大、连接数多的服务。

图 7-89 SNAT



您可以通过如下步骤实现容器实例访问Internet。

步骤1 创建弹性公网IP，具体请参见**申请弹性公网IP**。

1. 登录管理控制台。
2. 在管理控制台左上角单击，选择区域和项目。
3. 在控制台首页，单击左上角的，在展开的列表中单击“网络 > 弹性公网IP”。
4. 在“弹性公网IP”界面，单击“购买弹性公网IP”。
5. 根据界面提示配置参数。

   说明

   此处“区域”需选择容器实例所在区域。

图 7-90 购买弹性公网 IP

步骤2 创建NAT网关，具体请参见**购买NAT网关**。

1. 在控制台首页，单击左上角的 ☰，在展开的列表中单击“网络 > NAT网关”。
2. 在NAT网关页面，单击右上角的“购买公网NAT网关”。
3. 根据界面提示配置参数。

说明

此处需选择集群相同的VPC。

图 7-91 购买公网 NAT 网关

步骤3 配置SNAT规则，为子网绑定弹性公网IP，具体请参见**添加SNAT规则**。

1. 在NAT网关页面，单击需要添加SNAT规则的NAT网关名称。
2. 在SNAT规则页签中，单击“添加SNAT规则”。
3. 根据界面提示配置参数。

说明

SNAT规则是按网段生效，因为不同容器网络模型通信方式不同，此处子网需按如下规则选择。

- 容器隧道网络、VPC网络：需要选择节点所在子网，即创建节点时选择的子网。

对于存在多个网段的情况，可以创建多个SNAT规则或选择自定义网段，只要网段能包含节点子网即可。

图 7-92 配置 SNAT 规则

SNAT规则配置完成后，您就可以从容器中访问公网了，从容器中能够ping通公网。

----结束

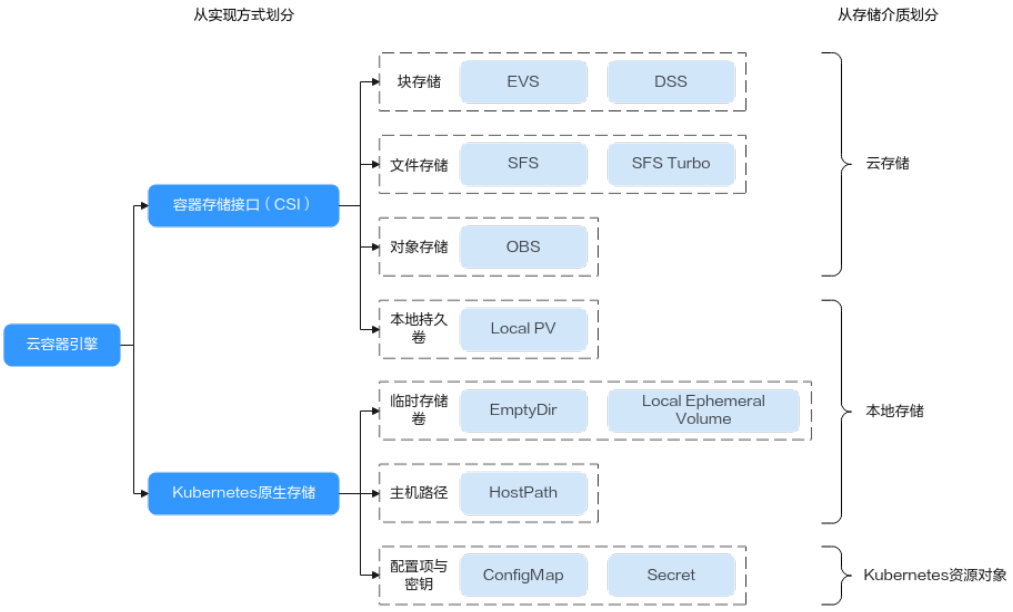
8 存储

8.1 存储概述

存储概览

CCE的容器存储功能基于Kubernetes容器存储接口（CSI）实现，深度融合多种类型的云存储并全面覆盖不同的应用场景，而且完全兼容Kubernetes原生的存储服务，例如EmptyDir、HostPath、Secret、ConfigMap等存储类型。

图 8-1 容器存储概览类型



CCE支持工作负载Pod绑定多种类型的存储：

- 从实现方式上划分，可以分为容器存储接口和Kubernetes原生存储。

类别	说明
容器存储接口	**Out-of-Tree**的形式，规定了标准的容器存储接口，可以允许存储供应商使用符合标准的自定义存储插件，通过PVC/PV的形式实现挂载，摒弃了以往需要将插件源码添加到Kubernetes代码仓库统一构建、编译、发布的方式。从Kubernetes 1.13开始，容器存储接口（CSI）是实现卷插件的推荐方法。
Kubernetes原生存储	In-Tree的形式，通过Kubernetes代码仓库统一构建、编译、发布。

- 从存储介质上划分，可以分为云存储、本地存储和Kubernetes资源对象。

类别	说明	应用场景
云存储	存储介质为存储供应商提供的云存储，该类别的存储卷挂载均通过PVC/PV形式。	一般用于存储可用性要求较高的数据，或部分数据需要共享的场景，例如日志保存、媒体资源存放等。 根据具体的使用场景，您可以选择合适的云存储类型，详情请参见**云存储对比**。
本地存储	存储介质为节点本地数据盘或内存，其中本地持久卷为CCE提供的自定义存储类型，通过容器存储接口以PVC/PV形式挂载，其余类型均为Kubernetes原生存储。	用于存储非高可用数据，可在IO要求较高、延迟低的场景下使用。 根据具体的使用场景，您可以选择合适的本地存储类型，详情请参见**本地存储对比**。
Kubernetes资源对象	ConfigMap和Secret是集群中创建的资源，属于比较特殊的存储类型，由Kubernetes API服务器上的tmpfs（基于RAM的文件系统）提供存储。	ConfigMap一般用于给Pod注入配置数据。 Secret一般用于给Pod传递敏感信息，例如密码。

云存储对比

对比维度	云硬盘EVS	文件存储SFS	极速文件存储SFS Turbo	对象存储OBS	专属存储DSS
概念	云硬盘（Elastic Volume Service）可以为云服务器提供高可靠、高性能、规格丰富并且可弹性扩展的块存储服务，可满足不同场景的业务需求，适用于分布式文件系统、开发测试、数据仓库以及高性能计算等场景。	SFS为用户提供一个完全托管的共享文件存储，能够弹性伸缩至PB规模，具备高可用性和持久性，为海量数据、高带宽型应用提供有力支持。适用于多种应用场景，包括HPC、媒体处理、文件共享、内容管理和Web服务等。	SFS Turbo为用户提供一个完全托管的共享文件存储，能够弹性伸缩至320TB规模，具备高可用性和持久性，为海量的小文件、低延迟高IOPS型应用提供有力支持。适用于多种应用场景，包括高性能网站、日志存储、压缩解压、DevOps、企业办公、容器应用等。	对象存储服务（Object Storage Service，OBS）提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。适合企业备份/归档、视频点播、视频监控等多种数据存储场景。	专属分布式存储服务（Dedicated Distributed Storage Service，DSS）可以为您提供独享的物理存储资源，通过数据冗余和缓存加速等多项技术，提供高可用性和持久性，以及稳定的低时延性能。
存储数据的逻辑	存放的是二进制数据，无法直接存放文件，如果需要存放文件，需要先格式化文件系统后使用。	存放的是文件，会以文件和文件夹的层次结构来整理和呈现数据。	存放的是文件，会以文件和文件夹的层次结构来整理和呈现数据。	存放的是对象，可以直接存放文件，文件会自动产生对应的系统元数据，用户也可以自定义文件的元数据。	存放的是二进制数据，无法直接存放文件，如果需要存放文件，需要先格式化文件系统后使用。
访问方式	只能在ECS/BMS中挂载使用，不能被操作系统应用直接访问，需要格式化成文件系统进行访问。	在ECS/BMS中通过网络协议挂载使用。需要指定网络地址进行访问，也可以将网络地址映射为本地目录后进行访问。	提供标准的文件访问协议NFS（仅支持NFSv3），用户可以将现有应用和工具与SFS Turbo无缝集成。	可以通过互联网或专线访问。需要指定桶地址进行访问，使用的是HTTP和HTTPS等传输协议。	只能在ECS/BMS中挂载使用，不能被操作系统应用直接访问，需要格式化成文件系统进行访问。

对比维度	云硬盘EVS	文件存储SFS	极速文件存储SFS Turbo	对象存储OBS	专属存储DSS
静态存储卷	支持，请参见**通过静态存储卷使用已有云硬盘**。	支持，请参见**通过静态存储卷使用已有文件存储**。	支持，请参见**通过静态存储卷使用已有极速文件存储**。	支持，请参见**通过静态存储卷使用已有对象存储**。	支持，请参见**通过静态存储卷使用专属存储**。
动态存储卷	支持，请参见**通过动态存储卷使用云硬盘**。	支持，请参见**通过动态存储卷使用文件存储**。	不支持	支持，请参见**通过动态存储卷使用对象存储**。	支持，请参见**通过动态存储卷使用专属存储**。
主要特点	非共享存储，每个云盘只能在单个节点挂载。	共享存储，可提供高性能、高吞吐存储服务。	高性能、高带宽、共享存储。	共享存储，用户态文件系统。	非共享存储，每个云盘只能在单个节点挂载。
应用场景	HPC高性能计算、企业核心集群应用、企业应用系统和开发测试等。 **说明** 高性能计算：主要是高速率、高IOPS的需求，用于作为高性能存储，比如工业设计、能源勘探等。	HPC高性能计算、媒体处理、内容管理和Web服务、大数据和分析应用程序等。 **说明** 高性能计算：主要是高带宽的需求，用于共享文件存储，比如基因测序、图片渲染等。	高性能网站、日志存储、DevOps、企业办公等。	大数据分析、静态网站托管、在线视频点播、基因测序、智能视频监控、备份归档、企业云盘（网盘）等。	• 混合负载，专属分布式存储可同时支持HPC、数据库、Email、OA办公、Web等多个应用混合部署 • 高性能计算 • OLAP应用
容量	TB级别	SFS 1.0：PB级别	通用型：TB级别	EB级别	TB级别
时延	1~2ms	SFS 1.0：3~20ms	通用型：1~5ms	10ms	1~3ms
最大IOPS	因规格而异，范围为2.2K~256K	SFS 1.0：2K	通用型：最大达100K	千万级	因规格而异，范围为1.5K~8K
带宽	MB/s级别	SFS 1.0：GB/s级别	通用型：最大为GB/s级别	TB/s级别	MB/s级别

本地存储对比

对比维度	本地持久卷（Local PV）	本地临时卷（Local Ephemeral Volume）	临时路径（EmptyDir）	主机路径（HostPath）
概念	将节点的本地数据盘通过LVM组成存储池（VolumeGroup），然后划分LV给容器挂载使用。	基于Kubernetes原生的EmptyDir类型，将节点的本地数据盘通过LVM组成存储池（VolumeGroup），然后划分LV作为EmptyDir的存储介质给容器挂载使用，相比原生EmptyDir默认的存储介质类型性能更好。	Kubernetes原生的EmptyDir类型，生命周期与容器实例相同，并支持指定内存作为存储介质。容器实例消亡时，EmptyDir会被删除，数据会永久丢失。	将容器所在宿主机的文件目录挂载到容器指定的挂载点中。
主要特点	低延迟、高IO，非高可用的持久卷。 存储卷通过Label绑定节点，且为非共享存储，只能在单个Pod中挂载。	本地临时卷，存储空间来自本地LV。	本地临时卷，存储空间来自本地的kubelet根目录或内存。	挂载主机节点文件系统上的文件或目录，支持自动创建主机目录，Pod可迁移（不绑定节点）。
存储卷挂载方式	不支持静态存储卷 支持**通过动态存储卷使用本地持久卷**	详情请参见**使用本地临时卷**。	详情请参见**使用临时路径**。	详情请参见**主机路径（HostPath）**。

对比维度	本地持久卷（Local PV）	本地临时卷（Local Ephemeral Volume）	临时路径（EmptyDir）	主机路径（HostPath）
应用场景	IO要求高、应用自带高可用方案的场景，例如：高可用部署MySQL。	• 缓存空间，例如基于磁盘的归并排序。 • 为耗时较长的计算任务提供检查点，以便任务能方便地从崩溃前状态恢复执行。 • 在Web服务器容器服务数据时，保存内容管理器容器获取的文件。	• 缓存空间，例如基于磁盘的归并排序。 • 为耗时较长的计算任务提供检查点，以便任务能方便地从崩溃前状态恢复执行。 • 在Web服务器容器服务数据时，保存内容管理器容器获取的文件。	运行一个需要使用节点文件。例如容器中需使用Docker，可使用HostPath挂载节点的/var/lib/docker路径。 **须知** HostPath卷存在许多安全风险，最佳做法是尽可能避免使用HostPath。当必须使用HostPath卷时，它的范围应仅限于所需的文件或目录，并以只读方式挂载。

企业项目支持说明

说明

该功能需要everest插件升级到1.2.33及以上版本。

- 自动创建存储：

  CCE支持使用存储类创建云硬盘和对象存储类型PVC时指定企业项目，将创建的存储资源（云硬盘和对象存储）归属于指定的企业项目下，**企业项目可选为集群所属的企业项目或default企业项目**。

  若不指定企业项目，则创建的存储资源默认使用存储类StorageClass中指定的企业项目。

  - 对于自定义的StorageClass，可以在StorageClass中指定企业项目，详见**场景三：指定StorageClass的企业项目**。StorageClass中如不指定的企业项目，则默认为default企业项目。
  - 对于CCE提供的 csi-disk 和 csi-obs 存储类，所创建的存储资源属于default企业项目。

- 使用已有存储：

  使用PV创建PVC时，因为存储资源在创建时已经指定了企业项目，如果PVC中指定企业项目，则务必确保在PVC和PV中指定的everest.io/enterprise-project-id保持一致，否则两者无法正常绑定。

相关文档

- **存储基础知识**

- 云硬盘存储（EVS）
- 文件存储（SFS）
- 极速文件存储（SFS Turbo）
- 对象存储（OBS）

8.2 存储基础知识

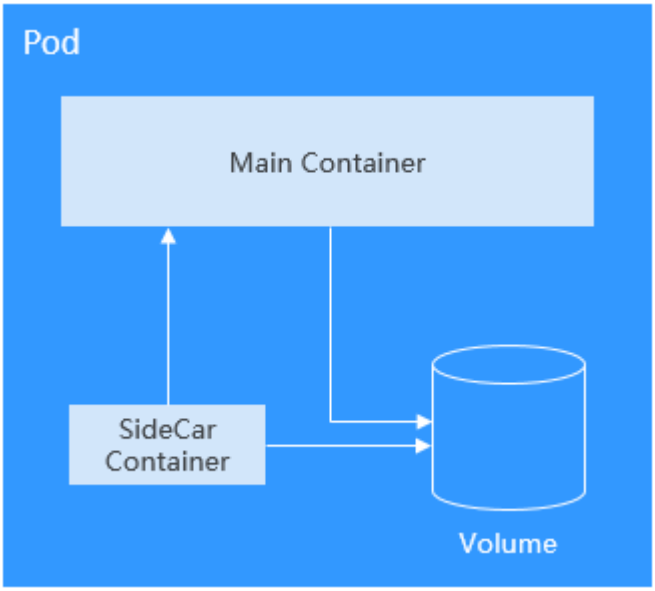
Volume（卷）

容器中的文件在磁盘上是临时存放的，这给容器中运行的较重要的应用程序带来如下两个问题：

1. 当容器重建时，容器中的文件将会丢失。
2. 当在一个Pod中同时运行多个容器时，容器间需要共享文件。

Kubernetes抽象出了Volume（卷）来解决以上两个问题。Kubernetes的Volume是Pod的一部分，Volume不是单独的对象，不能独立创建，只能在Pod中定义。Pod中的所有容器都可以使用Volume，但需要将Volume挂载到容器中的目录下。

实际中使用容器存储如下图所示，可将同一个Volume挂载到不同的容器中，实现不同容器间的存储共享。



存储卷的基本使用原则如下：

- 一个Pod可以挂载多个Volume。虽然单Pod可以挂载多个Volume，但是并不建议给一个Pod挂载过多卷。
- 一个Pod可以挂载多种类型的Volume。
- 每个被Pod挂载的Volume卷，可以在不同的容器间共享。
- Kubernetes环境推荐使用PVC和PV方式挂载Volume。

说明

卷（Volume）的生命周期与挂载它的Pod相同，即Pod被删除的时候，Volume也一起被删除。但是Volume里面的文件可能在Volume消失后仍然存在，这取决于Volume的类型。

Kubernetes提供了非常丰富的Volume类型，主要可分为In-Tree和Out-of-Tree两个大类：

卷（Volume）分类	描述
In-Tree	In-Tree卷是通过Kubernetes代码仓库维护的，与Kubernetes二进制文件一起构建、编译、发布，当前Kubernetes已不再接受这种模式的卷类型。 例如HostPath、EmptyDir、Secret和ConfigMap等Kubernetes原生支持的卷都属于这个类型。 而PVC（PersistentVolumeClaim）可以说是一种特殊的In-Tree卷，Kubernetes使用这种类型的卷从In-Tree模式向Out-of-Tree模式进行转换，这种类型的卷允许使用者在不同的存储供应商环境中“申请”使用底层存储创建的PV（PersistentVolume）。
Out-of-Tree	Out-of-Tree卷包括容器存储接口（CSI）和FlexVolume（已弃用），存储供应商只需遵循一定的规范即可创建自定义存储插件，创建可供Kubernetes使用的PV，而无需将插件源码添加到Kubernetes代码仓库。例如SFS、OBS等云存储都是通过在集群中安装存储驱动的形式使用的，需要在集群中创建对应的PV，然后使用PVC挂载到Pod中。

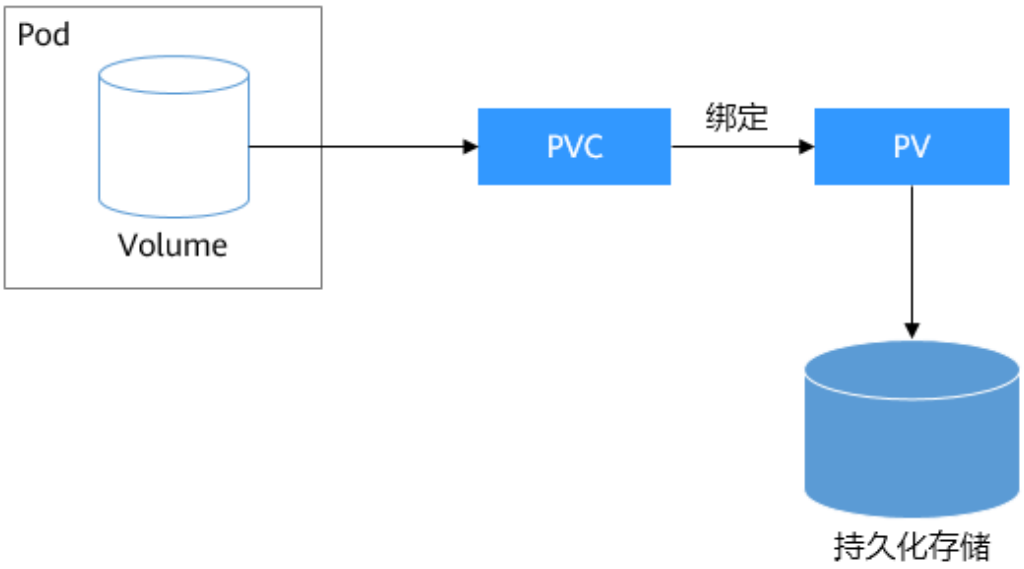
PV 与 PVC

Kubernetes抽象了PV（PersistentVolume）和PVC（PersistentVolumeClaim）来定义和使用存储，从而让使用者不用关心具体的基础设施，当需要存储资源的时候，只要像CPU和内存一样，声明要多少即可。

- PV：PV是PersistentVolume的缩写，译为持久化存储卷，描述的是一个集群里的持久化存储卷，它和节点一样，属于集群级别资源，其对象作用范围是整个Kubernetes集群。PV可以有自己的独立生命周期，不依附于Pod。
- PVC：PVC是PersistentVolumeClaim的缩写，译为持久化存储卷声明，描述的是负载对存储的申领。为应用配置存储时，需要声明一个存储需求（即PVC），Kubernetes会通过最佳匹配的方式选择一个满足需求的PV，并与PVC绑定。PVC与PV是一一对应关系，在创建PVC时，需描述请求的持久化存储的属性，比如，存储的大小、可读写权限等等。

在Pod中可以使用Volume关联PVC，即可让Pod使用到存储资源，它们之间的关系如下图所示。

图 8-2 PVC 绑定 PV



CSI

CSI（Container Storage Interface，容器存储接口）是容器标准存储接口规范，也是Kubernetes社区推荐的存储插件实现方案。everest是CCE基于CSI开发的存储插件，能够为容器提供不同类型的持久化存储功能。

存储卷访问模式

存储卷只能以底层存储资源所支持的方式挂载到宿主系统上。例如，文件存储可以支持多个节点读写，云硬盘只能被一个节点读写。

- ReadWriteOnce：存储卷可以被一个节点以读写方式挂载。
- ReadWriteMany：存储卷可以被多个节点以读写方式挂载。

表 8-1 存储卷支持的访问模式

存储类型	ReadWriteOnce	ReadWriteMany
云硬盘EVS	√	×
文件存储SFS	×	√
对象存储OBS	×	√
极速文件存储SFS Turbo	×	√
本地持久卷 LocalPV	√	×
专属存储DSS	√	×

存储卷挂载方式

通常在使用存储卷时，可以通过以下方式挂载：

可以使用PV描述已有的存储资源，然后通过创建PVC在Pod中使用存储资源。也可以使用动态创建的方式，在PVC中指定**存储类（StorageClass）**，利用StorageClass中的Provisioner自动创建PV来绑定PVC。

表 8-2 挂载存储卷的方式

挂载方式	说明	支持的存储卷类型	其他限制
静态创建存储卷（使用已有存储）	即使用已有的存储（例如云硬盘、文件存储等）创建好PV，并通过PVC在工作负载中挂载。Kubernetes会将PVC和匹配的PV进行绑定，这样就实现了工作负载访问存储服务的能力。	所有存储卷均支持	无
动态创建存储卷（自动创建存储）	即在PVC中指定**存储类（StorageClass）**，由存储Provisioner根据需求创建底层存储介质，实现PV的自动化创建并直接绑定至PVC。	云硬盘存储、对象存储、文件存储、本地持久卷、专属存储	无
动态挂载（VolumeClaimTemplate）	动态挂载能力通过卷申领模板（**volumeClaimTemplates**字段）实现，并依赖于StorageClass的动态创建PV能力。动态挂载可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。	仅云硬盘存储、本地持久卷、专属存储支持	仅有状态工作负载支持

PV 回收策略

PV回收策略用于指定删除PVC时，底层卷的回收策略，支持设定Delete、Retain回收策略。

- Delete：删除PVC的动作会将PV对象从Kubernetes中移除，同时也会从外部基础设施中移除所关联的底层存储资产。

  说明

  包周期的资源无法通过Delete回收策略进行级联删除。

- Retain：当PVC对象被删除时，PV对象与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，且不能直接再次被PVC绑定使用。

  您可以通过以下步骤来手动删除回收：

  a. 手动删除PersistentVolume对象。

  b. 根据情况，手动清除所关联的底层存储资源上的数据。

c. 手动删除所关联的底层存储资源。

如果您希望重用该底层存储资源，可以重新创建新的PersistentVolume对象。

CCE还支持一种删除PVC时不删除底层存储资源的使用方法，当前仅支持使用YAML创建：PV回收策略设置为Delete，并添加annotations“everest.io/reclaim-policy: retain-volume-only”。这样在删除PVC时，PV会被删除，但底层存储资源会保留。

以云硬盘为例，YAML示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/disk-volume-type: SAS
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
  volumeName: pv-evs-test

---
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only
  name: pv-evs-test
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  csi:
    driver: disk.csi.everest.io
    fsType: ext4
    volumeHandle: 2af98016-6082-4ad6-bedc-1a9c673aef20
    volumeAttributes:
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/disk-mode: SCSI
      everest.io/disk-volume-type: SAS
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-disk
```

相关文档

- 更多关于Kubernetes存储的信息，请参见**Storage**。
- 更多关于CCE容器存储的信息，请参见**存储概述**。

8.3 云硬盘存储（EVS）

8.3.1 云硬盘概述

为满足数据持久化的需求，CCE支持将云硬盘（EVS）创建的存储卷挂载到容器的某一路径下，当容器在同一可用区内迁移时，挂载的云硬盘将一同迁移。通过云硬盘，可以将存储系统的远端文件目录挂载到容器中，数据卷中的数据将被永久保存，即使删除了容器，数据卷中的数据依然保存在存储系统中。

云硬盘性能规格

云硬盘性能的主要指标包括：

- IOPS：云硬盘每秒进行读写的操作次数。
- 吞吐量：云硬盘每秒成功传送的数据量，即读取和写入的数据量。
- IO读写时延：云硬盘连续两次进行读写操作所需要的最小时间间隔。

表 8-3 云硬盘性能规格

参数	极速型SSD	通用SSD	超高IO	高IO
云硬盘最大容量（GiB）	• 系统盘：1024 • 数据盘：32768	• 系统盘：1024 • 数据盘：32768	• 系统盘：1024 • 数据盘：32768	• 系统盘：1024 • 数据盘：32768
最大IOPS	128000	20000	50000	5000
最大吞吐量（MiB/s）	1000	250	350	150
IOPS突发上限	64000	8000	16000	5000
云硬盘IOPS性能计算公式	IOPS = min (128000, 1800 + 50 × 容量)	IOPS=min (20000, 1800 + 12 × 容量)	IOPS = min (50000, 1800 + 50 × 容量)	IOPS = min (5000, 1800 + 8 × 容量)
云硬盘吞吐量性能计算公式（MiB/s）	吞吐量 = min (1000, 120 + 0.5 × 容量)	吞吐量 = min (250, 100 + 0.5 × 容量)	吞吐量 = min (350, 120 + 0.5 × 容量)	吞吐量 = min (150, 100 + 0.15 × 容量)
单队列访问时延（ms）	亚毫秒级	1	1	1~ 3
API名称	ESSD	GPSSD	SSD	SAS

关于云硬盘性能的详细介绍，请以**磁盘类型及性能介绍**为准。

使用场景

根据使用场景不同，云硬盘类型的存储支持以下挂载方式：

- **通过静态存储卷使用已有云硬盘**：即静态创建的方式，需要先使用已有的云硬盘创建PV，然后通过PVC在工作负载中挂载存储。适用于已有可用的底层存储或底层存储需要包周期的场景。
- **通过动态存储卷使用云硬盘**：即动态创建的方式，无需预先创建云硬盘，在创建PVC时通过指定存储类（StorageClass），即可自动创建云硬盘和对应的PV对象。适用于无可用的底层存储，需要新创建的场景。
- **在有状态负载中动态挂载云硬盘存储**：仅有状态工作负载支持，可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。适用于多实例的有状态工作负载。

计费说明

- 挂载云硬盘类型的存储卷时，通过StorageClass**自动创建**的云硬盘默认创建计费模式为“按需计费”，且不支持将云硬盘单独转包周期。如需使用包周期的云硬盘，请**使用已有的云硬盘存储**进行挂载。
- 关于云硬盘的价格信息，请参见**云硬盘计费说明**。

8.3.2 通过静态存储卷使用已有云硬盘

CCE支持使用已有的云硬盘创建存储卷（PersistentVolume）。创建成功后，通过创建相应的PersistentVolumeClaim绑定当前PersistentVolume使用。适用于已有底层存储或底层存储需要包周期的场景。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 您已经创建好一块云硬盘，并且云硬盘满足以下条件：
  - 已有的云硬盘不可以是系统盘、专属盘或共享盘。
  - 云硬盘模式需选择SCSI（购买云硬盘时默认为VBD模式）。
  - 云硬盘的状态可用，且未被其他资源使用。
  - 云硬盘的可用区需要与集群节点的可用区相同，否则无法挂载将导致实例启动失败。
  - 若云硬盘加密，所使用的密钥状态需可用。
  - 仅支持选择集群所属企业项目和default企业项目下的云硬盘。
  - 不支持使用已进行分区的云硬盘。
  - 仅支持使用ext4类型的云硬盘。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 云硬盘不支持跨可用区挂载，且不支持被多个工作负载、同一个工作负载的多个实例或多个任务使用。由于CCE集群各节点之间暂不支持共享盘的数据共享功能，多个节点挂载使用同一个云硬盘可能会出现读写冲突、数据缓存冲突等问题，所以创建无状态工作负载时，若使用了EVS云硬盘，建议工作负载只选择一个实例。
- 1.19.10以下版本的集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，当新Pod被调度到另一个节点时，会导致之前Pod不能正常读写。

  1.19.10及以上版本集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，新Pod会因为无法挂载云硬盘导致无法成功启动。

通过控制台使用已有云硬盘

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 静态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“云硬盘”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 已有底层存储的场景下，根据是否已经创建存储卷可选择“新建存储卷 PV”或“已有存储卷 PV”来静态创建PVC。 - 无可用底层存储的场景下，可选择“动态创建”，具体操作请参见**通过动态存储卷使用云硬盘**。 本文示例中选择“新建存储卷”，可通过控制台同时创建PV及PVC。
关联存储卷[a]	选择集群中已有的PV卷，需要提前创建PV，请参考**相关操作**中的“创建存储卷”操作。 本文示例中无需选择。
云硬盘[b]	单击“选择云硬盘”，您可以在新页面中勾选满足要求的云硬盘，并单击“确定”。
PV名称[b]	输入PV名称，同一集群内的PV名称需唯一。
访问模式[b]	云硬盘类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
回收策略[b]	您可以选择Delete或Retain，用于指定删除PVC时底层存储的回收策略，详情请参见**PV回收策略**。

说明

a：创建方式选择“已有存储卷 PV”时可设置。

b：创建方式选择“新建存储卷 PV”时可设置。

2. 单击“创建”，将同时为您创建存储卷声明及存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

   本文主要为您介绍存储卷的挂载使用，如**表8-4**，其他参数详情请参见**工作负载**。

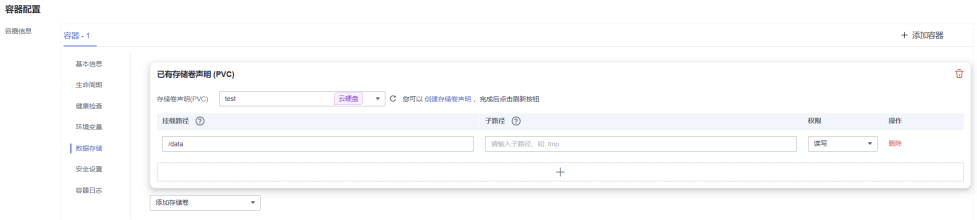
表 8-4 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的云硬盘存储卷。 云硬盘存储卷无法被多个工作负载重复挂载。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将磁盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到云硬盘中。

说明

由于云硬盘为非共享模式，工作负载下多个实例无法同时挂载，会导致实例启动异常。因此挂载云硬盘时，工作负载实例数需为1。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

通过 kubectl 命令行使用已有云硬盘

步骤1 使用kubectl连接集群。

步骤2 创建PV。当您的集群中已存在创建完成的PV时，可跳过本步骤。

1. 创建pv-evs.yaml文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only        # 可选字段，删除PV时可保留底层存储卷
  name: pv-evs    # PV的名称
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>      # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
    - ReadWriteOnce     # 访问模式，云硬盘必须为ReadWriteOnce
  capacity:
    storage: 10Gi       # 云硬盘的容量，单位为Gi，取值范围 1-32768
  csi:
    driver: disk.csi.everest.io     # 挂载依赖的存储驱动
    fsType: ext4    # 与磁盘原文件系统保持一致
    volumeHandle: <your_volume_id>   #云硬盘的volumeID
    volumeAttributes:
      everest.io/disk-mode: SCSI          # 云硬盘的磁盘模式，仅支持SCSI
      everest.io/disk-volume-type: SAS     # 云硬盘的类型
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
      everest.io/enterprise-project-id: <your_project_id>  # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
      everest.io/disk-iops: '3000'      # 可选字段，设置云硬盘的IOPS，仅云硬盘类型为GPSSD2时支持配置      everest.io/disk-throughput: '125' # 可选字段，设置云硬盘的吞吐量，仅云硬盘类型为GPSSD2时支持配置
  persistentVolumeReclaimPolicy: Delete    # 回收策略
  storageClassName: csi-disk             # 存储类名称，云硬盘必须为csi-disk
```

表 8-5 关键参数说明

参数	是否必选	描述
everest.io/reclaim-policy: retain-volume-only	否	可选字段 目前仅支持配置“retain-volume-only” everest插件版本需 >= 1.2.9且回收策略为Delete时生效。如果回收策略是Delete且当前值设置为“retain-volume-only”删除PVC回收逻辑为：删除PV，保留底层存储卷。
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
fsType	是	设置文件系统类型，默认为ext4。

参数	是否必选	描述
volumeHandle	是	云硬盘的volumeID。 **获取方法：**在云服务器控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下单击“ID”后的复制图标即可获取云硬盘的volumeID。
everest.io/disk-volume-type	是	云硬盘类型，全大写。 – SAS：高I/O – SSD：超高I/O – GPSSD：通用型SSD – ESSD：极速型SSD – GPSSD2：通用型SSD v2，Everest版本为2.4.4及以上支持使用，使用时需同时指定everest.io/disk-iops和everest.io/disk-throughput注解。
everest.io/disk-iops	否	IOPS值由用户预配置，仅使用通用型SSD v2的云硬盘支持设置。 – 通用型SSD v2类型的IOPS范围为3000~128000，最高可配置 500*容量（GiB）。 选择通用型SSD V2，当IOPS大于3000时会收取额外IOPS费用，详情请参见**价格计算器**。
everest.io/disk-throughput	否	吞吐量由用户预配置，仅使用通用型SSD v2类型的云硬盘支持设置。 范围为125~1000MiB/s，最高可配置IOPS/4。 吞吐量大于125MiB/s时会收取额外吞吐量费用，详情请参见**价格计算器**。
everest.io/crypt-key-id	否	当云硬盘是加密卷时为必填，填写创建云硬盘时选择的加密密钥ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下找到“配置信息”，复制密钥ID值即可。

参数	是否必选	描述
everest.io/enterprise-project-id	否	可选字段 云硬盘的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下找到“管理信息”中的企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可获取云硬盘所属的企业项目的ID。
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。 支持Delete、Retain回收策略，详情请参见**PV回收策略**。如果数据安全性要求较高，建议使用**Retain**以免误删数据。 **Delete**: – Delete且不设置everest.io/reclaim-policy：删除PVC，PV资源与云硬盘均被删除。 – Delete且设置everest.io/reclaim-policy=retain-volume-only：删除PVC，PV资源被删除，云硬盘资源会保留。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。
storageClassName	是	云硬盘存储对应的存储类名称为csi-disk。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-evs.yaml
```

步骤3 创建PVC。

1. 创建pvc-evs.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS    # 云硬盘的类型
    everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
    everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
  - ReadWriteOnce             # 云硬盘必须为ReadWriteOnce
```



```
resources:
  requests:
    storage: 10Gi            # 云硬盘大小，取值范围 1-32768，必须和已有PV的storage大小保持一致。
storageClassName: csi-disk   # StorageClass类型为云硬盘
volumeName: pv-evs           # PV的名称
```

表 8-6 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
volumeName	是	PV的名称，必须与**1**中PV的名称一致。
storageClassName	是	存储类名称，必须与**1**中PV的存储类一致。 云硬盘存储对应的存储类名称为csi-disk。

2. 执行以下命令，创建PVC。

```
kubectl apply -f pvc-evs.yaml
```

步骤4 创建应用。

1. 创建web-evs.yaml文件，本示例中将云硬盘挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web-evs
  namespace: default
spec:
  replicas: 1            # 使用云硬盘的工作负载副本数必须是1
  selector:
    matchLabels:
      app: web-evs
  serviceName: web-evs   # Headless Service名称
  template:
    metadata:
      labels:
        app: web-evs
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-disk     #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-disk     #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-evs     #已创建的PVC名称
---
```



```
apiVersion: v1
kind: Service
metadata:
  name: web-evs   # Headless Service名称
  namespace: default
  labels:
    app: web-evs
spec:
  selector:
    app: web-evs
  clusterIP: None
  ports:
    - name: web-evs
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

2. 执行以下命令，创建一个挂载云硬盘存储的应用。

```
kubectl apply -f web-evs.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及云硬盘文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-evs
```

预期输出如下：

```
web-evs-0               1/1     Running   0             38s
```

2. 执行以下命令，查看云硬盘是否挂载至/data路径。

```
kubectl exec web-evs-0 -- df | grep data
```

预期输出如下：

```
/dev/sdc            10255636     36888  10202364   0% /data
```

3. 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-evs-0 -- ls /data
```

预期输出如下：

```
lost+found
```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-evs-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-evs-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-evs-0的Pod。

```
kubectl delete pod web-evs-0
```

预期输出如下：

```
pod "web-evs-0" deleted
```


步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec web-evs-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明云硬盘中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-7中的操作。

表 8-7 其他操作

操作	说明	操作步骤
创建存储卷	通过CCE控制台单独创建PV。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷”页签。单击右上角“创建存储卷PV”，在弹出的窗口中填写存储卷声明参数。 • 存储卷类型：选择“云硬盘”。 • 云硬盘：单击“选择云硬盘”，在新页面中勾选满足要求的云硬盘，并单击“确定”。 • PV名称：输入PV名称，同一集群内的PV名称需唯一。 • 访问模式：仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。 • 回收策略：Delete或Retain，详情请参见**PV回收策略**。 2. 单击“创建”。
扩容云硬盘存储卷	通过CCE控制台快速扩容已挂载的云硬盘。 仅按需计费的云硬盘支持扩容，包周期的云硬盘请单击卷名称，前往存储服务扩容。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。

操作	说明	操作步骤
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.3.3 通过动态存储卷使用云硬盘

CCE支持指定存储类（StorageClass），自动创建云硬盘类型的底层存储和对应的存储卷，适用于无可用的底层存储，需要新创建的场景。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 云硬盘不支持跨可用区挂载，且不支持被多个工作负载、同一个工作负载的多个实例或多个任务使用。由于CCE集群各节点之间暂不支持共享盘的数据共享功能，多个节点挂载使用同一个云硬盘可能会出现读写冲突、数据缓存冲突等问题，所以创建无状态工作负载时，若使用了EVS云硬盘，建议工作负载只选择一个实例。
- 1.19.10以下版本的集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，当新Pod被调度到另一个节点时，会导致之前Pod不能正常读写。

  1.19.10及以上版本集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，新Pod会因为无法挂载云硬盘导致无法成功启动。
- 动态创建云硬盘存储卷时支持添加资源标签，且云硬盘创建完成后无法在CCE侧更新资源标签，需要前往云硬盘控制台更新。如果使用已有的云硬盘创建存储卷，也需要在云硬盘控制台添加或更新资源标签。

通过控制台自动创建云硬盘存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 动态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“云硬盘”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。

参数	描述
创建方式	- 无可用底层存储的场景下，可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。 - 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”，静态创建PVC，具体操作请参见**通过静态存储卷使用已有云硬盘**。 本文中选择“动态创建”。
存储类	云硬盘对应的默认存储类为csi-disk、csi-disk-topology。 **说明** 使用csi-disk（云硬盘）存储类，会立即创建PVC和PV（创建PV会同时创建云硬盘），然后PVC绑定PV。 使用csi-disk-topology（延迟创建的云硬盘）存储类，创建PVC时，不会立即创建PV，而是等需要挂载该PVC的Pod被调度后，再创建云硬盘及存储卷PV，并与PVC绑定。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
可用区	选择云硬盘的可用区，需要与集群节点的可用区相同。 **说明** 云硬盘只能挂载到同一可用区的节点上，创建后不支持更换可用区，请谨慎选择。
云硬盘类型	选择云硬盘类型。不同区域支持的云硬盘类型存在差异，请以控制台选项为准。 **说明** Everest版本为2.4.4及以上支持使用通用型SSD V2。通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。
容量（GiB）	申请的存储卷容量大小。
计费模式	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.16及以上版本的Everest插件。支持以下计费方式： - 包年/包月 使用包年/包月的计费模式需要选择购买时长，且默认开启自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。 - 按需计费 按资源的实际使用时长计费，可以随时开通/删除资源。

参数	描述
访问模式	云硬盘类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
加密	选择底层存储是否加密，使用加密时需要选择使用的加密密钥。使用前请确认云硬盘所在区域（Region）是否支持硬盘加密能力。
企业项目	仅支持default、集群所在企业项目或存储类指定的企业项目。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。集群中everest版本为2.1.39及以上时支持。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动创建“CCE-Cluster-ID=<集群ID>”、“CCE-Cluster-Name=<集群名称>”、“CCE-Namespace=<命名空间名称>”的系统标签，您无法自定义修改。 **说明** 云硬盘类型的动态存储卷创建完成后，不支持在CCE侧更新资源标签。如需更新云硬盘的资源标签，请前往云硬盘控制台。

2. 单击“创建”。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

   本文主要为您介绍存储卷的挂载使用，如**表8-8**，其他参数详情请参见**工作负载**。

表 8-8 存储卷挂载

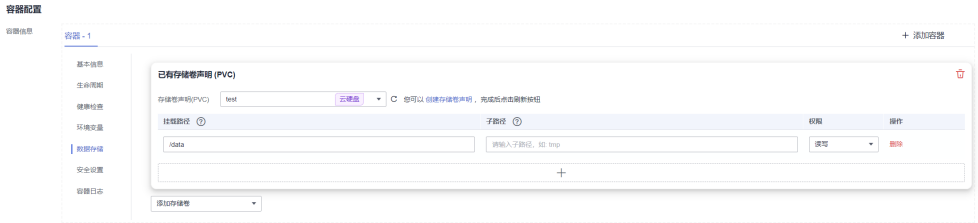
参数	参数说明
存储卷声明（PVC）	选择已有的云硬盘存储卷。 云硬盘存储卷无法被多个工作负载重复挂载。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。

参数	参数说明
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将磁盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到云硬盘中。

说明

由于云硬盘为非共享模式，工作负载下多个实例无法同时挂载，会导致实例启动异常。因此挂载云硬盘时，工作负载实例数需为1。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

使用 kubectl 自动创建云硬盘存储

步骤1 使用kubectl连接集群。

步骤2 使用StorageClass动态创建PVC及PV。

1. 创建pvc-evs-auto.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-auto
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS    # 云硬盘的类型
    everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
    everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
    everest.io/disk-iops: '3000'     # 可选字段，设置云硬盘的IOPS，仅云硬盘类型为GPSSD2时支持配置    everest.io/disk-throughput: '125'# 可选字段，设置云硬盘的吞吐量，仅云硬盘类型为GPSSD2时支持配置
    everest.io/csi.charging-mode: prePaid  # 可选字段，表示使用包年/包月计费模式
    everest.io/csi.period-type: month     # 使用包年/包月计费模式时必填，表示计费周期，此处表示按月计费
    everest.io/csi.period-num: '1'        # 使用包年/包月计费模式时必填，表示计费时长，此处表示1个月
    everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
```

```
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
  - ReadWriteOnce               # 云硬盘必须为ReadWriteOnce
  resources:
    requests:
      storage: 10Gi             # 云硬盘大小，取值范围 1-32768
  storageClassName: csi-disk    # StorageClass类型为云硬盘
```

表 8-9 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
everest.io/disk-volume-type	是	云硬盘类型，全大写。 – SAS：高I/O – SSD：超高I/O – GPSSD：通用型SSD – ESSD：极速型SSD – GPSSD2：通用型SSD v2，Everest版本为2.4.4及以上支持使用，使用时需同时指定everest.io/disk-iops和everest.io/disk-throughput注解。
everest.io/disk-iops	否	IOPS值由用户预配置，仅使用通用型SSD v2的云硬盘支持设置。 – 通用型SSD v2类型的IOPS范围为3000~128000，最高可配置 500*容量（GiB）。 选择通用型SSD V2，当IOPS大于3000时会收取额外IOPS费用，详情请参见**价格计算器**。
everest.io/disk-throughput	否	吞吐量由用户预配置，仅使用通用型SSD v2类型的云硬盘支持设置。 范围为125~1000MiB/s，最高可配置IOPS/4。 吞吐量大于125MiB/s时会收取额外吞吐量费用，详情请参见**价格计算器**。

参数	是否必选	描述
everest.io/crypt-key-id	否	当云硬盘是加密卷时为必填，填写创建云硬盘时选择的加密密钥ID，可使用自定义密钥或名为“evs/default”的云硬盘默认密钥。 **获取方法**：在数据加密控制台，找到需要加密的密钥，复制密钥ID值即可。
everest.io/enterprise-project-id	否	可选字段 云硬盘的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在企业项目管理控制台，单击要对接的企业项目名称，复制企业项目ID值即可。
everest.io/csi.charging-mode	否	可选字段，不填写该字段时默认为按需计费模式。该字段在集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.16及以上版本的Everest插件。 取值如下： – prePaid：表示云硬盘存储卷使用包年/包月计费模式，且默认开启自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。 – postPaid：表示云硬盘存储卷使用按需计费模式。 **说明** 创建包年/包月的云硬盘存储卷时，需要为cce_cluster_agency委托添加费用中心的bss:order:pay权限。建议您在首次创建时使用主账号通过控制台创建，在进行确认后将自动为您授权。
everest.io/csi.period-type	否	可选字段，使用包年/包月计费模式时必填，表示计费周期，取值如下： – month：表示按月计费。 – year：表示按年计费。
everest.io/csi.period-num	否	可选字段，使用包年/包月计费模式时必填，表示计费时长。 – 按月计费时，取值范围为1-9。 – 按年计费时，取值为1。

参数	是否必选	描述
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi，取值范围为1-32768。
storageClassName	是	云硬盘存储对应的存储类名称为csi-disk。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-evs-auto.yaml
```

步骤3 创建应用。

1. 创建web-evs-auto.yaml文件，本示例中将云硬盘挂载至/data路径。
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web-evs-auto
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-evs-auto
  serviceName: web-evs-auto   # Headless Service名称
  template:
    metadata:
      labels:
        app: web-evs-auto
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-disk    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-disk    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-evs-auto    #已创建的PVC名称
---
apiVersion: v1
kind: Service
metadata:
  name: web-evs-auto   # Headless Service名称
```



```
  namespace: default
  labels:
    app: web-evs-auto
spec:
  selector:
    app: web-evs-auto
  clusterIP: None
  ports:
    - name: web-evs-auto
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

2. 执行以下命令，创建一个挂载云硬盘存储的应用。

```
kubectl apply -f web-evs-auto.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及云硬盘文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-evs-auto
```

预期输出如下：

```
web-evs-auto-0                1/1     Running   0             38s
```

2. 执行以下命令，查看云硬盘是否挂载至/data路径。

```
kubectl exec web-evs-auto-0 -- df | grep data
```

预期输出如下：

```
/dev/sdc             10255636     36888  10202364   0% /data
```

3. 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-evs-auto-0 -- ls /data
```

预期输出如下：

```
lost+found
```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-evs-auto-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-evs-auto-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-evs-auto-0的Pod。

```
kubectl delete pod web-evs-auto-0
```

预期输出如下：

```
pod "web-evs-auto-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec web-evs-auto-0 -- ls /data
```


预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明云硬盘中的数据可持久化保存。

----结束

相关操作

您还可以执行**表8-10**中的操作。

表 8-10 其他操作

操作	说明	操作步骤
扩容云硬盘存储卷	通过CCE控制台快速扩容已挂载的云硬盘。 仅按需计费的云硬盘支持扩容，包周期的云硬盘请单击卷名称，前往存储服务扩容。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.3.4 在有状态负载中动态挂载云硬盘存储

使用场景

动态挂载仅可在创建**有状态负载（StatefulSet）**时使用，通过卷声明模板（**volumeClaimTemplates**字段）实现，并依赖于StorageClass的动态创建PV能力。在多实例的有状态负载中，动态挂载可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。而在无状态工作负载的普通挂载方式中，当存储支持多点挂载（ReadWriteMany）时，工作负载下的多个Pod会被挂载到同一个底层存储中。

说明

Kubernetes不允许在更新StatefulSet时添加或删除volumeClaimTemplates字段，您只能在创建StatefulSet时设置volumeClaimTemplates。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台动态挂载云硬盘存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。

步骤3 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 动态挂载 (VolumeClaimTemplate)”。

步骤4 单击“创建存储卷声明 PVC”，在弹出窗口中填写存储卷声明参数。

参数填写完成后，单击“创建”。

参数	描述
存储卷声明类型	本文中选择“云硬盘”。
PVC名称	输入PVC的名称。创建后将根据实例数自动增加后缀，格式为<自定义PVC名称>-<序号>，例如example-0。
创建方式	可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。
存储类	云硬盘对应的默认存储类为csi-disk、csi-disk-topology。 **说明** 使用csi-disk（云硬盘）存储类，会立即创建PVC和PV（创建PV会同时创建云硬盘），然后PVC绑定PV。 使用csi-disk-topology（延迟创建的云硬盘）存储类，创建PVC时，不会立即创建PV，而是等需要挂载该PVC的Pod被调度后，再创建云硬盘及存储卷PV，并与PVC绑定。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
可用区	选择云硬盘的可用区，需要与集群节点的可用区相同。 **说明** 云硬盘只能挂载到同一可用区的节点上，创建后不支持更换可用区，请谨慎选择。

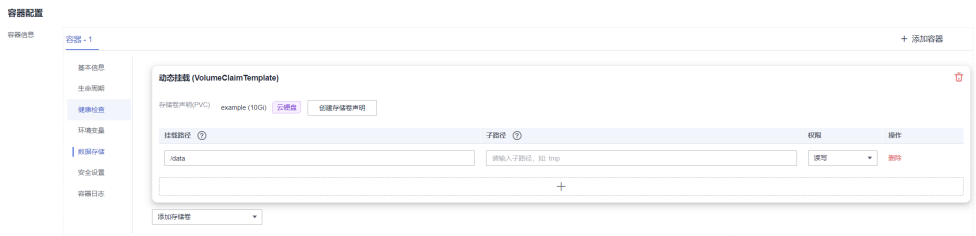
参数	描述
云硬盘类型	选择云硬盘类型。不同区域支持的云硬盘类型存在差异，请以控制台选项为准。 **说明** Everest版本为2.4.4及以上支持使用通用型SSD V2。通用型SSD V2支持自定义设置IOPS和吞吐量，设置范围参见**云硬盘性能数据表**。
容量（GiB）	申请的存储卷容量大小。
计费模式	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.16及以上版本的Everest插件。支持以下计费方式： • 包年/包月 使用包年/包月的计费模式需要选择购买时长，且默认开启自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。 • 按需计费 按资源的实际使用时长计费，可以随时开通/删除资源。
访问模式	云硬盘类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
加密	选择底层存储是否加密，使用加密时需要选择使用的加密密钥。使用前请确认云硬盘所在区域（Region）是否支持硬盘加密能力。
企业项目	仅支持default、集群所在企业项目或存储类指定的企业项目。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。集群中everest版本为2.1.39及以上时支持。 您可以在资源标签管理服务中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动创建“CCE-Cluster-ID=<集群ID>”、“CCE-Cluster-Name=<集群名称>”、“CCE-Namespace=<命名空间名称>”的系统标签，您无法自定义修改。 **说明** 云硬盘类型的动态存储卷创建完成后，不支持在CCE侧更新资源标签。如需更新云硬盘的资源标签，请前往云硬盘控制台。

步骤5 填写挂载路径。

表 8-11 存储卷挂载

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将磁盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到云硬盘中。



步骤6 本文主要为您介绍存储卷的动态挂载使用，其他参数详情请参见**创建有状态负载（StatefulSet）**。其余信息都配置完成后，单击“创建工作负载”。

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

通过 kubectl 命令行动态挂载云硬盘存储

步骤1 使用kubectl连接集群。

步骤2 创建statefulset-evs.yaml文件，本示例中将云硬盘挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-evs
  namespace: default
spec:
  selector:
    matchLabels:
      app: statefulset-evs
```

```
  template:
    metadata:
      labels:
        app: statefulset-evs
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - name: pvc-disk          # 需与volumeClaimTemplates字段中的名称对应
              mountPath: /data        # 存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
  serviceName: statefulset-evs        # Headless Service名称
  replicas: 2
  volumeClaimTemplates:
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: pvc-disk
        namespace: default
        annotations:
          everest.io/disk-volume-type: SAS    # 云硬盘的类型
          everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
          everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
          everest.io/disk-iops: '3000'     # 可选字段，设置云硬盘的IOPS，仅云硬盘类型为GPSSD2时支持配置
          everest.io/disk-throughput: '125' # 可选字段，设置云硬盘的吞吐量，仅云硬盘类型为GPSSD2时支持配置
          everest.io/csi.charging-mode: prePaid  # 可选字段，表示使用包年/包月计费模式
          everest.io/csi.period-type: month      # 使用包年/包月计费模式时必填，表示计费周期，此处表示按月计费
          everest.io/csi.period-num: '1'         # 使用包年/包月计费模式时必填，表示计费时长，此处表示1个月
          everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
        labels:
          failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
          failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
      spec:
        accessModes:
          - ReadWriteOnce              # 云硬盘必须为ReadWriteOnce
        resources:
          requests:
            storage: 10Gi              # 云硬盘大小，取值范围 1-32768
        storageClassName: csi-disk      # StorageClass类型为云硬盘
---
apiVersion: v1
kind: Service
metadata:
  name: statefulset-evs   # Headless Service名称
  namespace: default
  labels:
    app: statefulset-evs
spec:
  selector:
    app: statefulset-evs
  clusterIP: None
  ports:
    - name: statefulset-evs
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

表 8-12 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
everest.io/disk-volume-type	是	云硬盘类型，全大写。 ● SAS：高I/O ● SSD：超高I/O ● GPSSD：通用型SSD ● ESSD：极速型SSD ● GPSSD2：通用型SSD v2，Everest版本为2.4.4及以上支持使用，使用时需同时指定everest.io/disk-iops和everest.io/disk-throughput注解。
everest.io/disk-iops	否	IOPS值由用户预配置，仅使用通用型SSD v2的云硬盘支持设置。 ● 通用型SSD v2类型的IOPS范围为3000~128000，最高可配置 500*容量（GiB）。 选择通用型SSD V2，当IOPS大于3000时会收取额外IOPS费用，详情请参见**价格计算器**。
everest.io/disk-throughput	否	吞吐量由用户预配置，仅使用通用型SSD v2类型的云硬盘支持设置。 范围为125~1000MiB/s，最高可配置 IOPS/4。 吞吐量大于125MiB/s时会收取额外吞吐量费用，详情请参见**价格计算器**。
everest.io/crypt-key-id	否	当云硬盘是加密卷时为必填，填写创建云硬盘时选择的加密密钥ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下找到“配置信息”，复制密钥ID值即可。

参数	是否必选	描述
everest.io/enterprise-project-id	否	可选字段 云硬盘的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“云硬盘 > 磁盘”，单击要对接的云硬盘名称进入详情页，在“概览信息”页签下找到“管理信息”中的企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可获取云硬盘所属的企业项目的ID。
everest.io/csi.charging-mode	否	可选字段，不填写该字段时默认为按需计费模式。该字段在集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.16及以上版本的Everest插件。 取值如下： ● prePaid：表示云硬盘存储卷使用包年/包月计费模式，且默认开启自动续费。按月购买自动续费周期为1个月，按年购买自动续费周期为1年。 ● postPaid：表示云硬盘存储卷使用按需计费模式。 **说明** 创建包年/包月的云硬盘存储卷时，需要为cce_cluster_agency委托添加费用中心的bss:order:pay权限。建议您在首次创建时使用主账号通过控制台创建，在进行确认后将自动为您授权。
everest.io/csi.period-type	否	可选字段，使用包年/包月计费模式时必填，表示计费周期，取值如下： ● month：表示按月计费。 ● year：表示按年计费。
everest.io/csi.period-num	否	可选字段，使用包年/包月计费模式时必填，表示计费时长。 ● 按月计费时，取值范围为1-9。 ● 按年计费时，取值为1。

参数	是否必选	描述
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi，取值范围为1-32768。
storageClassName	是	云硬盘存储对应的存储类名称为csi-disk。

步骤3 执行以下命令，创建一个挂载云硬盘存储的应用。

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

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及云硬盘文件。

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep statefulset-evs
   ```
   预期输出如下：
   ```
   statefulset-evs-0       1/1     Running   0          45s
   statefulset-evs-1       1/1     Running   0          28s
   ```
2. 执行以下命令，查看云硬盘是否挂载至/data路径。
   ```
   kubectl exec statefulset-evs-0 -- df | grep data
   ```
   预期输出如下：
   ```
   /dev/sdd            10255636     36888  10202364   0% /data
   ```
3. 执行以下命令，查看/data路径下的文件。
   ```
   kubectl exec statefulset-evs-0 -- ls /data
   ```
   预期输出如下：
   ```
   lost+found
   ```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec statefulset-evs-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec statefulset-evs-0 -- ls /data
```

预期输出如下：


```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-evs-auto-0的Pod。

```
kubectl delete pod statefulset-evs-0
```

预期输出如下：

```
pod "statefulset-evs-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec statefulset-evs-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明云硬盘中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-13中的操作。

表 8-13 其他操作

操作	说明	操作步骤
扩容云硬盘存储卷	通过CCE控制台快速扩容已挂载的云硬盘。 仅按需计费的云硬盘支持扩容，包周期的云硬盘请单击卷名称，前往存储服务扩容。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.3.5 快照与备份

CCE通过云硬盘EVS服务为您提供快照功能，云硬盘快照简称快照，指云硬盘数据在某个时刻的完整复制或镜像，是一种重要的数据容灾手段，当数据丢失时，可通过快照将数据完整的恢复到快照时间点。

您可以创建快照，从而快速保存指定时刻云硬盘的数据。同时，您还可以通过快照创建新的云硬盘，这样云硬盘在初始状态就具有快照中的数据。

使用须知

- 快照功能**仅支持v1.15及以上版本**的集群，且需要安装基于CSI的everest插件才可以使用。
- 基于快照创建的云硬盘，其子类型（普通IO/高IO/超高IO）、是否加密、磁盘模式（VBD/SCSI）、共享性(非共享/共享)、容量等都要与快照关联磁盘保持一致，这些属性查询和设置出来后不能够修改。
- 只有可用或正在使用状态的磁盘能创建快照，且单个磁盘最大支持创建7个快照。
- 创建快照功能仅支持使用everest插件提供的存储类（StorageClass名称以csi开头）创建的PVC。使用Flexvolume存储类（StorageClass名为ssd、sas、sata）创建的PVC，无法创建快照。
- 加密磁盘的快照数据以加密方式存放，非加密磁盘的快照数据以非加密方式存放。

使用场景

快照功能可以帮助您实现以下需求：

- **日常备份数据**

  通过对云硬盘定期创建快照，实现数据的日常备份，可以应对由于误操作、病毒以及黑客攻击等导致数据丢失或不一致的情况。

- **快速恢复数据**

  更换操作系统、应用软件升级或业务数据迁移等重大操作前，您可以创建一份或多份快照，一旦升级或迁移过程中出现问题，可以通过快照及时将业务恢复到快照创建点的数据状态。

  例如，当由于云服务器 A的系统盘 A发生故障而无法正常开机时，由于系统盘 A已经故障，因此也无法将快照数据回滚至系统盘A。此时您可以使用系统盘 A已有的快照新创建一块云硬盘 B并挂载至正常运行的云服务器 B上，从而云服务器 B能够通过云硬盘 B读取原系统盘 A的数据。

  说明

  当前CCE提供的快照能力与K8s社区CSI快照功能一致：只支持基于快照创建新云硬盘，不支持将快照回滚到源云硬盘。

- **快速部署多个业务**

  通过同一个快照可以快速创建出多个具有相同数据的云硬盘，从而可以同时为多种业务提供数据资源。例如数据挖掘、报表查询和开发测试等业务。这种方式既保护了原始数据，又能通过快照创建的新云硬盘快速部署其他业务，满足企业对业务数据的多元化需求。

创建快照

使用控制台创建

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“存储”，在右侧选择“快照与备份”页签。

步骤3 单击右上角“创建快照”，在弹出的窗口中设置相关参数。

- 快照名称：填写快照的名称。
- 选择存储：选择要创建快照的PVC，仅能选择云硬盘类型PVC。

步骤4 单击“创建”。

----结束

使用YAML创建

```
kind: VolumeSnapshot
apiVersion: snapshot.storage.k8s.io/v1beta1
metadata:
  name: cce-disksnap-test   # 快照名称
  namespace: default
spec:
  source:
    persistentVolumeClaimName: pvc-evs-test    # PVC的名称，仅能选择云硬盘类型PVC
  volumeSnapshotClassName: csi-disk-snapclass
```

使用快照创建 PVC

通过快照创建云硬盘PVC时，磁盘类型、磁盘模式、加密属性需和快照源云硬盘保持一致。

使用控制台创建

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“存储”，在右侧选择“快照与备份”页签。

步骤3 找到需要创建PVC的快照，单击“创建存储卷声明 PVC”，并在弹出窗口中设置PVC参数。

- PVC名称：请输入PVC名称。
- 资源标签：通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。集群中everest版本为2.1.39及以上时支持。

  您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。

  CCE服务会自动创建“CCE-Cluster-ID=<集群ID>”、“CCE-Cluster-Name=<集群名称>”、“CCE-Namespace=<命名空间名称>”的系统标签，您无法自定义修改。

步骤4 单击“创建”。

----结束

使用YAML创建

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test
  namespace: default
  annotations:
    everest.io/disk-volume-type: SSD    # 云硬盘类型，需要与快照源云硬盘保持一致
```



```
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为云硬盘所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为云硬盘所在的可用区
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
  dataSource:
    name: cce-disksnap-test            # 快照的名称
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

8.4 文件存储（SFS）

8.4.1 文件存储概述

文件存储介绍

CCE支持将弹性文件存储（SFS）创建的存储卷挂载到容器的某一路径下，以满足数据持久化需求，SFS存储卷适用于多读多写的持久化存储，适用大容量扩展以及成本敏感型的业务场景，包括媒体处理、内容管理、大数据分析和分析工作负载程序等。SFS容量型文件系统不适合海量小文件业务，推荐使用SFS Turbo文件系统。

SFS为用户提供一个完全托管的共享文件存储，能够弹性伸缩至PB规模，具备高可用性和持久性，为海量数据、高带宽型应用提供有力支持。

- **符合标准文件协议：**用户可以将文件系统挂载给服务器，像使用本地文件目录一样。
- **数据共享：**多台服务器可挂载相同的文件系统，数据可以共享操作和访问。
- **私有网络：**数据访问必须在数据中心内部网络中。
- **容量与性能：**单文件系统容量较高（PB级），性能极佳（IO读写时延ms级）。
- **应用场景：**适用于多读多写（ReadWriteMany）场景下的各种工作负载（Deployment/StatefulSet）和普通任务（Job）使用，主要面向高性能计算、媒体处理、内容管理和Web服务、大数据和分析应用程序等场景。

文件存储性能

CCE支持使用SFS容量型文件存储、通用文件系统（SFS 3.0）。更多关于文件存储类型的详细介绍，请参见**文件系统类型**。

说明

- SFS容量型文件存储当前正处于售罄状态，CCE控制台已经不再支持新建，存量SFS容量型仍可以**通过kubectl命令行方式**创建PV。
- 通用文件系统（SFS 3.0）当前正在各区域（Region）逐步上线中，部分区域可能还未支持，请您耐心等待或咨询SFS服务客户支持。若您的应用所在区域已经支持通用文件系统，建议新应用使用通用文件系统，并尽快将已有的SFS容量型文件存储迁移到通用文件系统中，以免容量不足影响业务。

表 8-14 文件存储性能

参数	SFS容量型	通用文件系统（SFS 3.0）
最大带宽	2GB/s	1.25TB/s
最高IOPS	2K	百万
时延	3~20ms	10ms
最大容量	4PB	EB

使用场景

根据使用场景不同，文件存储支持以下挂载方式：

- **通过静态存储卷使用已有文件存储**：即静态创建的方式，需要先使用已有的文件存储创建PV，然后通过PVC在工作负载中挂载存储。适用于已有可用的底层存储或底层存储需要包周期的场景。
- **通过动态存储卷使用文件存储**：即动态创建的方式，无需预先创建文件存储，在创建PVC时通过指定存储类（StorageClass），即可自动创建文件存储和对应的PV对象。适用于无可用的底层存储，需要新创建的场景。

计费说明

- 挂载文件存储类型的存储卷时，通过StorageClass**自动创建**的文件存储默认创建计费模式为“按需计费”，按实际使用的存储容量和时长收费。关于文件存储的价格信息，请参见**文件存储计费说明**。
- 如需使用包周期的文件存储，请**使用已有的文件存储**进行挂载。

8.4.2 通过静态存储卷使用已有文件存储

文件存储（SFS）是一种可共享访问，并提供按需扩展的高性能文件系统（NAS），适用大容量扩展以及成本敏感型的业务场景。本文介绍如何使用已有的文件存储静态创建PV和PVC，并在工作负载中实现数据持久化与共享性。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经创建好一个文件存储，并且文件存储与集群在同一个VPC内。
- 使用通用文件系统（SFS 3.0）时，您需要提前在集群所在VPC创建一个VPC终端节点，集群需要通过VPC终端节点访问通用文件系统。配置VPC终端节点的方法请参见**配置VPC终端节点**。

约束与限制

- 支持多个PV挂载同一个SFS或SFS Turbo，但有如下限制：
  - 多个不同的PVC/PV使用同一个底层SFS或SFS Turbo卷时，如果挂载至同一Pod使用，会因为PV的volumeHandle参数值相同导致无法为Pod挂载所有PVC，出现Pod无法启动的问题，请避免该使用场景。

  - PV中persistentVolumeReclaimPolicy参数建议设置为Retain，否则可能存在一个PV删除时级联删除底层卷，其他关联这个底层卷的PV会由于底层存储被删除导致使用出现异常。
  - 重复用底层存储时，建议在应用层做好多读多写的隔离保护，防止产生的数据覆盖和丢失。
- 使用通用文件系统（SFS 3.0）存储卷时，集群中需要安装2.0.9及以上版本的CCE容器存储（Everest）插件。
- 使用通用文件系统（SFS 3.0）存储卷时，挂载点不支持修改属组和权限，挂载点默认属主为root。
- 使用通用文件系统（SFS 3.0）时，创建、删除PVC和PV过程中可能存在时延，实际计费时长请以SFS侧创建、删除时刻为准。
- 通用文件系统（SFS 3.0）使用Delete回收策略时，需要挂载文件系统手动删除所有文件后才可以正常删除PV和PVC。

通过控制台使用已有文件存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 静态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“文件存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”来静态创建PVC。 - 无可用底层存储的场景下，可选择“动态创建存储实例”，具体操作请参见**通过动态存储卷使用文件存储**。 本文示例中选择“新建存储卷”，可通过控制台同时创建PV及PVC。
关联存储卷[a]	选择集群中已有的PV卷，需要提前创建PV，请参考**相关操作**中的“创建存储卷”操作。 本文示例中无需选择。
文件存储[b]	单击“选择文件存储”，您可以在新页面中勾选满足要求的文件存储，并单击“确定”。 **说明** 当前仅支持选择通用文件系统（SFS 3.0）类型的文件存储。
子目录[b]	选择是否使用子目录创建PV。请填写子目录绝对路径，例如/a/b，并确保子目录已存在且可用。
PV名称[b]	输入PV名称，同一集群内的PV名称需唯一。
访问模式[b]	文件存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。

参数	描述
回收策略[b]	您可以选择Delete或Retain，用于指定删除PVC时底层存储的回收策略，详情请参见**PV回收策略**。 **说明** 多个PV使用同一个底层存储时建议使用**Retain**，避免级联删除底层卷。
子目录回收策略[b]	删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用，当使用子目录创建PV，且PV回收策略为"Delete"时支持配置。 – 保留：删除PVC，PV会被删除，但**PV关联的子目录会被保留**。 – 删除：删除PVC，**PV及其关联的子目录均会被删除**。
挂载参数[b]	输入挂载参数键值对，详情请参见**设置文件存储挂载参数**。

说明

a：创建方式选择“已有存储卷”时可设置。

b：创建方式选择“新建存储卷”时可设置。

2. 单击“创建”，将同时为您创建存储卷声明和存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

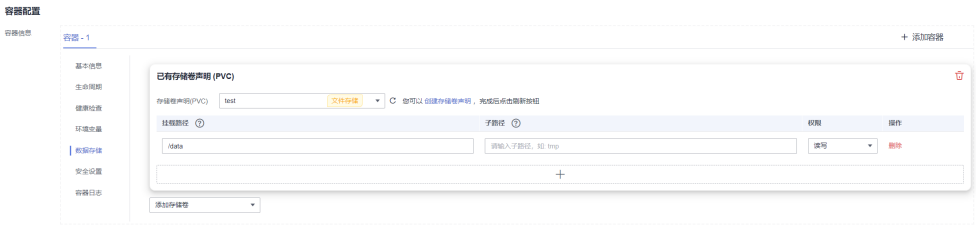
   本文主要为您介绍存储卷的挂载使用，如**表8-15**，其他参数详情请参见**工作负载**。

表 8-15 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的文件存储卷。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。

参数	参数说明
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到文件存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

通过 kubectl 命令行使用已有文件存储

您可以根据不同的使用场景选择不同的创建方式。

使用已有通用文件系统（SFS 3.0）

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-sfs.yaml文件。

   示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除PV，保留底层存储卷
  name: pv-sfs    # PV的名称
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，文件存储必须为ReadWriteMany
  capacity:
    storage: 1Gi       # 文件存储容量大小
  csi:
    driver: nas.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: <sfs30_name>   # 使用通用文件系统（SFS 3.0）时填写文件存储名称
```

```
  volumeAttributes:
    everest.io/share-export-location: <your_location>  #文件存储的共享路径
    storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    everest.io/sfs-version: sfs3.0       # 使用通用文件系统（SFS 3.0）
persistentVolumeReclaimPolicy: Retain    # 回收策略
storageClassName: csi-sfs                # 存储类名称，csi-sfs表示使用通用文件系统（SFS 3.0）
mountOptions: []                         # 挂载参数
```

表 8-16 关键参数说明

参数	是否必选	描述
everest.io/reclaim-policy	否	目前仅支持配置“retain-volume-only” everest插件版本需 >= 1.2.9且回收策略为Delete时生效。如果回收策略是Delete且当前值设置为“retain-volume-only”删除PVC回收逻辑为：删除PV，保留底层存储卷。
volumeHandle	是	使用通用文件系统（SFS 3.0）时，填写文件存储的名称。
everest.io/share-export-location	是	通用文件系统（SFS 3.0）的共享路径。 共享路径格式如下： *{your_sfs30_name}*.sfs3.*{region}*.myhuaweicloud.com:/*{your_sfs30_name}*
mountOptions	是	挂载参数。 不设置时默认配置为如下配置，具体说明请参见**设置文件存储挂载参数**。 mountOptions: - vers=3 - timeo=600 - nolock - hard
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。 支持Delete、Retain回收策略，详情请参见**PV回收策略**。多个PV使用同一个文件存储时建议使用**Retain**，避免级联删除底层卷。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。 **Delete**：表示删除PVC时，同时删除PV。
storage	是	PVC申请容量，单位为Gi。 对文件存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	存储类名称，填写csi-sfs，表示使用通用文件系统（SFS 3.0）。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-sfs.yaml
```

步骤3 创建PVC。

1. 创建pvc-sfs.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany                # 文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi               # 文件存储大小
  storageClassName: csi-sfs      # 存储类名称，必须与PV的存储类一致
  volumeName: pv-sfs     # PV的名称
```

表 8-17 关键参数说明

参数	是否必选	描述
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
storageClassName	是	存储类名称，填写csi-sfs，必须与1中PV的存储类一致，表示使用通用文件系统（SFS 3.0）。
volumeName	是	PV的名称，必须与1中PV的名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-sfs.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将文件存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfs-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
```



```
      - name: default-secret
    volumes:
      - name: pvc-sfs-volume    #卷名称，可自定义
        persistentVolumeClaim:
          claimName: pvc-sfs    #已创建的PVC名称
```

2. 执行以下命令，创建一个挂载文件存储的应用。

```
kubectl apply -f web-demo.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

使用已有通用文件系统（SFS 3.0）的子目录

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-sfs.yaml文件。

   示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
  name: pv-sfs    # PV的名称
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，文件存储必须为ReadWriteMany
  capacity:
    storage: 1Gi       # 文件存储容量大小
  csi:
    driver: nas.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: pv-sfs    # 使用子目录时填写PV的名称
    volumeAttributes:
      everest.io/share-export-location: <your_location>  #文件存储的共享路径
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/sfs-version: sfs3.0       # 使用通用文件系统（SFS 3.0）
      everest.io/csi.volume-as: absolute-path    # 表示使用SFS子目录
      everest.io/csi.path: /a  # 表示自动创建的子目录，必须为绝对路径
      everest.io/csi.sfs30-name: <sfs_name>     # 仅使用子目录时使用，参数值为SFS实例的名称
      everest.io/csi.reclaim-policy: retain-volume-only    # 仅使用子目录且回收策略为Delete时使用，表示删除PVC时，PV会被删除，但PV关联的子目录会被保留
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-sfs             # 存储类名称：csi-sfs表示使用通用文件系统（SFS 3.0）
  mountOptions: []                      # 挂载参数
```

表 8-18 关键参数说明

参数	是否必选	描述
volumeHandle	是	使用SFS子目录时，填写PV名称。
everest.io/share-export-location	是	通用文件系统（SFS 3.0）的共享路径。 共享路径格式如下： *{your_sfs30_name}*.sfs3.*{region}*.myhuaweicloud.com:/*{your_sfs30_name}*/*{absolute_path}*

参数	是否必选	描述
mountOptions	是	挂载参数。 不设置时默认配置为如下配置，具体说明请参见**设置文件存储挂载参数**。 mountOptions: - vers=3 - timeo=600 - nolock - hard
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。 支持Delete、Retain回收策略，详情请参见**PV回收策略**。多个PV使用同一个文件存储时建议使用**Retain**，避免级联删除底层卷。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。 **Delete**：表示删除PVC时，同时删除PV。
everest.io/csi.volume-as	否	固定取值为“absolute-path”，表示使用动态创建SFS子目录。 集群中需安装2.4.41及以上版本的Everest插件。
everest.io/csi.path	否	自动创建的子目录，必须为绝对路径。
everest.io/csi.sfs30-name	否	仅动态创建SFS子目录时使用，参数值为SFS实例的名称。
everest.io/csi.reclaim-policy	否	仅动态创建SFS子目录时使用，表示删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用。仅当PV回收策略为"Delete"时生效，取值如下： – retain-volume-only：表示删除PVC时，PV会被删除，但**PV关联的子目录会被保留**。 – delete：表示删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。
storage	是	PVC申请容量，单位为Gi。 对文件存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	存储类名称，填写csi-sfs，表示使用通用文件系统（SFS 3.0）。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-sfs.yaml
```

步骤3 创建PVC。

1. 创建pvc-sfs.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/csi.volume-as: absolute-path # 可选，表示使用SFS子目录
    everest.io/csi.path: /a  # 自动创建的子目录，必须为绝对路径
    everest.io/csi.sfs30-name: <sfs_name>     # 使用子目录时使用，参数值为SFS实例的名称
    everest.io/csi.reclaim-policy: retain-volume-only
spec:
  accessModes:
  - ReadWriteMany               # 文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi            # 文件存储大小
  storageClassName: csi-sfs     # 存储类名称，必须与PV的存储类一致
  volumeName: pv-sfs    # PV的名称
```

表 8-19 关键参数说明

参数	是否必选	描述
everest.io/csi.volume-as	是	固定取值为“absolute-path”，表示使用动态创建SFS子目录。 集群中需安装2.4.41及以上版本的Everest插件。
everest.io/csi.path	是	自动创建的子目录，必须为绝对路径。
everest.io/csi.sfs30-name	是	仅动态创建SFS子目录时使用，参数值为SFS实例的名称。
everest.io/csi.reclaim-policy	是	仅动态创建SFS子目录时使用，表示删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用。仅当PV回收策略为"Delete"时生效，取值如下： – retain-volume-only：表示删除PVC时，PV会被删除，但**PV关联的子目录会被保留**。 – delete：表示删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
storageClassName	是	存储类名称，填写csi-sfs，必须与1中PV的存储类一致，表示使用通用文件系统（SFS 3.0）。

参数	是否必选	描述
volumeName	是	PV的名称，必须与1中PV的名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-sfs.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将文件存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfs-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-sfs-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-sfs    #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载文件存储的应用。
```
kubectl apply -f web-demo.yaml
```
工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

使用已有 SFS 容量型存储

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-sfs.yaml文件。

   示例如下：
```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除PV，保留底层存储卷
  name: pv-sfs    # PV的名称
spec:
  accessModes:
```



```
- ReadWriteMany      # 访问模式，文件存储必须为ReadWriteMany
capacity:
  storage: 1Gi       # 文件存储容量大小
csi:
  driver: nas.csi.everest.io    # 挂载依赖的存储驱动
  fsType: nfs
  volumeHandle: <your_volume_id>   # SFS容量型文件存储的ID
  volumeAttributes:
    everest.io/share-export-location: <your_location>  #文件存储的共享路径
    storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
persistentVolumeReclaimPolicy: Retain    # 回收策略
storageClassName: csi-nas              # 存储类名称，csi-nas表示使用SFS容量型
mountOptions: []                       # 挂载参数
```

表 8-20 关键参数说明

参数	是否必选	描述
everest.io/reclaim-policy	否	目前仅支持配置“retain-volume-only” everest插件版本需 >= 1.2.9且回收策略为Delete时生效。如果回收策略是Delete且当前值设置为“retain-volume-only”删除PVC回收逻辑为：删除PV，保留底层存储卷。
volumeHandle	是	使用SFS容量型文件存储时，填写文件存储的ID。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，并选择SFS容量型。在列表中单击对应的弹性文件存储名称，在详情页中复制“ID”后的内容即可。
everest.io/share-export-location	是	文件存储的共享路径。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，在弹性文件服务列表中可以看到“挂载地址”列，即为文件存储的共享路径。
mountOptions	是	挂载参数。 不设置时默认配置为如下配置，具体说明请参见**设置文件存储挂载参数**。 mountOptions: - vers=3 - timeo=600 - nolock - hard

参数	是否必选	描述
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。 支持Delete、Retain回收策略，详情请参见**PV回收策略**。多个PV使用同一个文件存储时建议使用**Retain**，避免级联删除底层卷。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。 **Delete**：表示删除PVC时，同时删除PV。
storage	是	PVC申请容量，单位为Gi。 对文件存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	存储类名称csi-nas，表示使用SFS 1.0容量型文件存储。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-sfs.yaml
```

步骤3 创建PVC。

1. 创建pvc-sfs.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany                # 文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi               # 文件存储大小
  storageClassName: csi-nas      # 存储类名称，必须与PV的存储类一致
  volumeName: pv-sfs     # PV的名称
```

表 8-21 关键参数说明

参数	是否必选	描述
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
storageClassName	是	存储类名称，填写csi-nas，必须与1中PV的存储类一致，表示使用SFS 1.0容量型文件存储。
volumeName	是	PV的名称，必须与1中PV的名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-sfs.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将文件存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfs-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-sfs-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-sfs    #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载文件存储的应用。
```
kubectl apply -f web-demo.yaml
```
工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

验证数据持久化及共享性

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-mjhm9   1/1     Running   0          46s
web-demo-846b489584-wvv5s   1/1     Running   0          46s
```
2. 依次执行以下命令，查看Pod的/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
两个Pod均无返回结果，说明/data路径下无文件。

步骤2 执行以下命令，在/data路径下创建static文件。
```
kubectl exec web-demo-846b489584-mjhm9 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
```
预期输出如下：
```
static
```


步骤4 验证数据持久化

1. 执行以下命令，删除名称为web-demo-846b489584-mjhm9的Pod。
   ```
   kubectl delete pod web-demo-846b489584-mjhm9
   ```
   预期输出如下：
   ```
   pod "web-demo-846b489584-mjhm9" deleted
   ```
   删除后，Deployment控制器会自动重新创建一个副本。
2. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep web-demo
   ```
   预期输出如下，web-demo-846b489584-d4d4j为新建的Pod：
   ```
   web-demo-846b489584-d4d4j   1/1     Running   0          110s
   web-demo-846b489584-wvv5s   1/1     Running   0          7m50s
   ```
3. 执行以下命令，验证新建的Pod中/data路径下的文件是否更改。
   ```
   kubectl exec web-demo-846b489584-d4d4j -- ls /data
   ```
   预期输出如下：
   ```
   static
   ```
   static文件仍然存在，则说明数据可持久化保存。

步骤5 验证数据共享性

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep web-demo
   ```
   预期输出如下：
   ```
   web-demo-846b489584-d4d4j   1/1     Running   0          7m
   web-demo-846b489584-wvv5s   1/1     Running   0          13m
   ```
2. 执行以下命令，在任意一个Pod的/data路径下创建share文件。本例中选择名为web-demo-846b489584-d4d4j的Pod。
   ```
   kubectl exec web-demo-846b489584-d4d4j --  touch /data/share
   ```
   并查看该Pod中/data路径下的文件。
   ```
   kubectl exec web-demo-846b489584-d4d4j -- ls /data
   ```
   预期输出如下：
   ```
   share
   static
   ```
3. 由于写入share文件的操作未在名为web-demo-846b489584-wvv5s的Pod中执行，在该Pod中查看/data路径下是否存在文件以验证数据共享性。
   ```
   kubectl exec web-demo-846b489584-wvv5s -- ls /data
   ```
   预期输出如下：
   ```
   share
   static
   ```
   如果在任意一个Pod中的/data路径下创建文件，其他Pod下的/data路径下均存在此文件，则说明两个Pod共享一个存储卷。

----结束

相关操作

您还可以执行表8-22中的操作。

表 8-22 其他操作

操作	说明	操作步骤
创建存储卷	通过CCE控制台单独创建PV。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷”页签。单击右上角“创建存储卷”，在弹出的窗口中填写存储卷参数。 • 存储卷类型：选择“文件存储”。 • 文件存储：单击“选择文件存储”，在新页面中勾选满足要求的文件存储，并单击“确定”。 • 子目录：选择是否使用子目录创建PV。请填写子目录绝对路径，例如/a/b，并确保子目录已存在且可用。 • PV名称：输入PV名称，同一集群内的PV名称需唯一。 • 访问模式：仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。 • 回收策略：Delete或Retain，详情请参见**PV回收策略**。 **说明** 多个PV使用同一个底层存储时建议使用**Retain**，避免级联删除底层卷。 • 子目录回收策略：删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用，当使用子目录创建PV，且PV回收策略为"Delete"时支持配置。 保留：删除PVC，PV会被删除，但**PV关联的子目录会被保留**。 删除：删除PVC，**PV及其关联的子目录均会被删除**。 • 挂载参数：输入挂载参数键值对，详情请参见**设置文件存储挂载参数**。 2. 单击“创建”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.4.3 通过动态存储卷使用文件存储

本文介绍如何通过存储类动态创建PV和PVC，并在工作负载中实现数据持久化与共享性。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经创建好一个文件存储，并且文件存储与集群在同一个VPC内。
- 使用通用文件系统（SFS 3.0）时，您需要提前在集群所在VPC创建一个VPC终端节点，集群需要通过VPC终端节点访问通用文件系统。配置VPC终端节点的方法请参见**配置VPC终端节点**。

约束与限制

- 使用通用文件系统（SFS 3.0）SFS存储卷时，集群中需要安装2.0.9及以上版本的**CCE容器存储（Everest）**。
- 使用通用文件系统（SFS 3.0）SFS存储卷时，挂载点不支持修改属组和权限，挂载点默认属主为root。
- 使用通用文件系统（SFS 3.0）时，创建、删除PVC和PV过程中可能存在时延，实际计费时长请以SFS侧创建、删除时刻为准。
- 通用文件系统（SFS 3.0）使用Delete回收策略时，需要挂载文件系统手动删除所有文件后才可以正常删除PV和PVC。

通过控制台自动创建文件存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 动态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“文件存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 无可用底层存储的场景下，可选择“动态创建存储实例”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。 - 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”，静态创建PVC，具体操作请参见**通过静态存储卷使用已有文件存储**。 本文中选择“动态创建存储实例”。
存储类	文件存储对应的默认存储类为csi-sfs或csi-nas。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。

参数	描述
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
访问模式	文件存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。
加密	存储类为csi-nas时，可选择底层存储是否加密，使用加密时需要选择使用的加密密钥。

2. 单击“创建”，将同时为您创建存储卷声明和存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

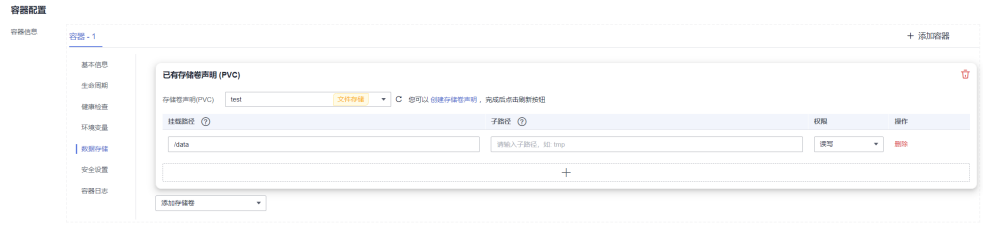
   本文主要为您介绍存储卷的挂载使用，如**表8-23**，其他参数详情请参见**工作负载**。

表 8-23 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的文件存储卷。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。

参数	参数说明
权限	– 只读：只能读容器路径中的数据卷。 – 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到文件存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

使用 kubectl 自动创建文件存储

步骤1 使用kubectl连接集群。

步骤2 使用StorageClass动态创建PVC及PV。

1. 创建pvc-sfs-auto.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs-auto
  namespace: default
  annotations:
    everest.io/crypt-key-id: <your_key_id>      # 可选字段，密钥的id，使用该密钥加密文件存储
    everest.io/crypt-alias: sfs/default         # 可选字段，密钥的名称，创建加密卷时必须提供该字段
    everest.io/crypt-domain-id: <your_domain_id>  # 可选字段，指定加密卷所属租户的ID，创建加密卷时必须提供该字段
    everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
spec:
  accessModes:
    - ReadWriteMany              # 文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi               # 文件存储大小
  storageClassName: csi-nas    # StorageClass类型为文件存储。csi-nas表示使用SFS容量型；csi-sfs表示使用通用文件系统（SFS 3.0）
```


表 8-24 关键参数说明

参数	是否必选	描述
storage	是	PVC申请容量，单位为Gi。 对文件存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	存储类名称。 – csi-sfs：推荐使用，表示使用通用文件系统（SFS 3.0）。 – csi-nas：表示使用SFS 1.0容量型文件存储。
everest.io/crypt-key-id	否	存储类为csi-nas时，可选择底层存储是否加密。 当文件存储是加密卷时为必填，填写创建文件存储时选择的加密密钥ID，可使用自定义密钥或名为“sfs/default”的云硬盘默认密钥。 **获取方法**：在数据加密控制台，找到需要加密的密钥，复制密钥ID值即可。
everest.io/crypt-alias	否	密钥的名称，创建加密卷时必须提供该字段。 **获取方法**：在数据加密控制台，找到需要加密的密钥，复制密钥名称即可。
everest.io/crypt-domain-id	否	指定加密卷所属租户的ID，创建加密卷时必须提供该字段。 **获取方法**：在云服务器控制台，鼠标悬浮至右上角的用户名称并单击“我的凭证”，复制账号ID即可。
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。

2. 执行以下命令，创建PVC。

```
kubectl apply -f pvc-sfs-auto.yaml
```


步骤3 创建应用。

1. 创建web-demo.yaml文件，本示例中将文件存储挂载至/data路径。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfs-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-sfs-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-sfs-auto    #已创建的PVC名称
```

2. 执行以下命令，创建一个挂载文件存储的应用。

```
kubectl apply -f web-demo.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

验证数据持久化及共享性

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-demo
```

预期输出如下：

```
web-demo-846b489584-mjhm9   1/1     Running   0          46s
web-demo-846b489584-wvv5s   1/1     Running   0          46s
```

2. 依次执行以下命令，查看Pod的/data路径下的文件。

```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```

两个Pod均无返回结果，说明/data路径下无文件。

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-demo-846b489584-mjhm9 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
```

预期输出如下：

```
static
```

步骤4 验证数据持久化

1. 执行以下命令，删除名称为web-demo-846b489584-mjhm9的Pod。
```
kubectl delete pod web-demo-846b489584-mjhm9
```
预期输出如下：
```
pod "web-demo-846b489584-mjhm9" deleted
```
删除后，Deployment控制器会自动重新创建一个副本。
2. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下，web-demo-846b489584-d4d4j为新建的Pod：
```
web-demo-846b489584-d4d4j   1/1     Running   0          110s
web-demo-846b489584-wvv5s   1/1     Running   0          7m50s
```
3. 执行以下命令，验证新建的Pod中/data路径下的文件是否更改。
```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```
预期输出如下：
```
static
```
static文件仍然存在，则说明数据可持久化保存。

步骤5 验证数据共享性

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-d4d4j   1/1     Running   0          7m
web-demo-846b489584-wvv5s   1/1     Running   0          13m
```
2. 执行以下命令，在任意一个Pod的/data路径下创建share文件。本例中选择名为web-demo-846b489584-d4d4j的Pod。
```
kubectl exec web-demo-846b489584-d4d4j --  touch /data/share
```
并查看该Pod中/data路径下的文件。
```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```
预期输出如下：
```
share
static
```
3. 由于写入share文件的操作未在名为web-demo-846b489584-wvv5s的Pod中执行，在该Pod中查看/data路径下是否存在文件以验证数据共享性。
```
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
预期输出如下：
```
share
static
```
如果在任意一个Pod中的/data路径下创建文件，其他Pod下的/data路径下均存在此文件，则说明两个Pod共享一个存储卷。

----结束

相关操作

您还可以执行**表8-25**中的操作。

表 8-25 其他操作

操作	说明	操作步骤
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

常见问题

挂载SFS 3.0存储时，出现挂载超时的问题，报错如下：

MountVolume.SetUp failed for volume "*"** : rpc error: code = Internal desc = [30834707-b8fc-11ee-ba7a-fa163eaacb17] failed to execute cmd: "systemd-run --scope mount -t nfs -o proto=tcp -o vers=3 -o timeo=600 -o noresvport -o nolock ***.sfs3.cn-east-3.myhuaweicloud.com:/*** /mnt/paas/kubernetes/kubelet/pods/add9a323-10e2-434f-b151-42675f83860e/volumes/kubernetes.io~csi/***/mount". outputs: Running scope as unit run-1597979.scope.; error: signal: killed. **please check whether the volumeAttributes of related PersistentVolume of the volume is correct and whether it can be mounted.**

解决方案

使用SFS 3.0文件存储前，需要提前创建VPC终端节点。如果未创建VPC终端节点，集群会无法访问SFS 3.0文件存储导致挂载超时失败。配置VPC终端节点的方法请参见**配置VPC终端节点**。

8.4.4 通过动态存储卷创建 SFS 子目录

通常情况下，在工作负载容器中挂载SFS类型的存储卷时，默认会将根目录挂载到容器中。为了更加经济合理地利用存储容量，CCE支持在创建PVC时动态创建SFS子目录，实现不同工作负载共享使用SFS。

须知

仅通用文件系统（SFS 3.0）支持动态创建子目录。

前提条件

- 您已经创建好一个集群，并且在该集群中安装2.4.41及以上版本的**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经创建好一个状态可用的SFS，并且SFS与集群在同一个VPC内。使用通用文件系统（SFS 3.0）时，您需要提前在集群所在VPC创建一个VPC终端节点，集群

需要通过VPC终端节点访问通用文件系统。配置VPC终端节点的方法请参见**配置VPC终端节点**。

通过控制台动态创建 SFS 子目录

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“文件存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	选择“动态创建子目录”。
存储类	选择文件存储对应的存储类为csi-sfs。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
访问模式	文件存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。
文件存储	单击“选择文件存储”，您可以在新页面中勾选满足要求的文件存储，并单击“确定”。
子目录	请填写子目录绝对路径，例如/a/b。
子目录回收策略	删除PVC时是否保留子目录。 • 保留：删除PVC，PV会被删除，但**PV关联的子目录会被保留**。 • 删除：删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。

步骤3 单击“创建”，将同时为您创建存储卷声明和存储卷。

您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

----结束

通过 kubectl 命令行动态创建 SFS 子目录

步骤1 使用kubectl连接集群。

步骤2 创建pvc-sfs-subpath.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfs-subpath    # PVC的名称
  namespace: default
  annotations:
```



```
  everest.io/csi.volume-as: absolute-path               # 表示使用SFS子目录
  everest.io/csi.sfs30-name: <sfs_name>                   # SFS实例的名称
  everest.io/csi.path: /a                              # 自动创建的子目录，必须为绝对路径
  everest.io/csi.reclaim-policy: retain-volume-only      # 表示删除PVC时，PV会被删除，但PV关联的子目录会被保留
spec:
  accessModes:
  - ReadWriteMany      # SFS必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi      # 对于SFS子目录类型的PVC，此处无实际意义，仅作校验需要（不能为空和0）
  storageClassName: csi-sfs      # 通用文件系统（SFS 3.0）存储类名称
```

表 8-26 关键参数说明

参数	是否必选	描述
everest.io/csi.volume-as	是	使用动态创建SFS 子目录时，固定取值为“absolute-path”。
everest.io/csi.sfs30-name	是	SFS实例的名称。
everest.io/csi.path	是	自动创建的子目录，必须为绝对路径。
everest.io/csi.reclaim-policy	是	删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用。仅当PV回收策略为"Delete"时生效，取值如下： ● retain-volume-only：表示删除PVC时，PV会被删除，但**PV关联的子目录会被保留**。 ● delete：表示删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。
storage	是	PVC申请容量，单位为Gi。 对SFS子目录类型的PVC来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处可以设定为固定值1Gi。

步骤3 执行以下命令，创建PVC。

```
kubectl apply -f pvc-sfs-subpath.yaml
```

----结束

8.4.5 设置文件存储挂载参数

本章节主要介绍如何设置文件存储的挂载参数。您可以在PV中设置挂载参数，然后通过PVC绑定PV，也可以在StorageClass中设置挂载参数，然后使用StorageClass创建PVC，动态创建出的PV会默认带有StorageClass中设置的挂载参数。

前提条件

CCE容器存储（Everest）版本要求**1.2.8及以上**版本。插件主要负责将挂载参数识别并传递给底层存储，指定参数是否有效依赖于底层存储是否支持。

约束与限制

- 挂载参数暂不支持安全容器。
- 由于NFS协议限制，默认情况下，对于某个节点多次挂载同一文件存储的场景，涉及链路的挂载参数（如timeo）仅在第一次挂载时生效。例如，节点上运行的多个Pod同时挂载同一文件存储，后设置的挂载参数不会覆盖已有参数值。针对上述场景希望设置不同的挂载参数，可以同时设置nosharecache挂载参数。

文件存储挂载参数

CCE的存储插件everest在挂载文件存储时默认设置了如**表8-27**所示的参数。

表 8-27 文件存储挂载参数

参数	参数值	描述
keep-original-ownership	无需填写	表示是否保留文件挂载点的ownership，使用该参数时，要求everest插件版本为1.2.63或2.1.2以上。 ● 默认为不添加该参数，此时挂载文件存储时将会默认把挂载点的ownership修改为root:root。 ● 如添加该参数，挂载文件存储时将保持文件系统原有的ownership。
vers	3	文件系统版本，目前只支持NFSv3。取值：3
nolock	无需填写	选择是否使用NLM协议在服务器上锁文件。当选择nolock选项时，锁对于同一主机的应用有效，对不同主机不受锁的影响。
timeo	600	NFS客户端重传请求前的等待时间(单位为0.1秒)。建议值：600。
hard/soft	无需填写	挂载方式类型。 ● 取值为hard，即使用硬连接方式，若NFS请求超时，则客户端一直重新请求直至成功。 ● 取值为soft，即软挂载方式挂载系统，若NFS请求超时，则客户端向调用程序返回错误。 默认为hard。

参数	参数值	描述
sharecache/nosharecache	无需填写	设置客户端并发挂载同一文件系统时数据缓存和属性缓存的共享方式。设置为sharecache时，多个挂载共享同一缓存。设为nosharecache时，每个挂载各有一个缓存。默认为sharecache。 **说明** 设置nosharecache禁用共享缓存会对性能产生一定影响。每次挂载都会重新获取挂载信息，会增加与NFS服务器的通信开销和NFS客户端的内存消耗，同时同客户端设置nosharecache存在cache不一致的风险。因此，应该根据具体情况进行权衡，以确定是否需要使用nosharecache选项。

除了以上参数外，您还可以设置其他的文件存储挂载参数，具体请参见**挂载NFS文件系统到云服务器（Linux）**。

在 PV 中设置挂载参数

在PV中设置挂载参数可以通过mountOptions字段实现，如下所示，mountOptions支持挂载的字段请参见**文件存储挂载参数**。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在PV中设置挂载参数，示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除PV，保留底层存储卷
  name: pv-sfs
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，文件存储必须为ReadWriteMany
  capacity:
    storage: 1Gi       # 文件存储容量大小
  csi:
    driver: nas.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: <your_volume_id>   # SFS容量型文件存储的ID
    volumeAttributes:
      everest.io/share-export-location: <your_location>  # 文件存储的共享路径
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-nas                # 存储类名称
  mountOptions:                          # 挂载参数
  - vers=3
  - nolock
  - timeo=600
  - hard
```

步骤3 PV创建后，可以创建PVC关联PV，然后在工作负载的容器中挂载，具体操作步骤请参见**通过静态存储卷使用已有文件存储**。

步骤4 验证挂载参数是否生效。

本例中将PVC挂载至使用nginx:latest镜像的工作负载，并通过**mount -l**命令查看挂载参数是否生效。

1. 查看已挂载文件存储的Pod，本文中的示例工作负载名称为web-sfs。
   ```
   kubectl get pod | grep web-sfs
   ```
   回显如下：
   ```
   web-sfs-***   1/1     Running   0          23m
   ```
2. 执行以下命令查看挂载参数，其中web-sfs-***为示例Pod。
   ```
   kubectl exec -it web-sfs-*** -- mount -l | grep nfs
   ```
   若回显中的挂载信息与设置的挂载参数一致，说明挂载参数设置成功。
   ```
   <您的共享路径> on /data type nfs
   (rw,relatime,vers=3,rsize=1048576,wsize=1048576,namlen=255,hard,nolock,noresvport,proto=tcp,
   timeo=600,retrans=2,sec=sys,mountaddr=**.**.**.**,mountvers=3,mountport=2050,mountproto=tcp
   ,local_lock=all,addr=**.**.**.**)
   ```

----结束

在 StorageClass 中设置挂载参数

在StorageClass中设置挂载参数同样可以通过mountOptions字段实现，如下所示，mountOptions支持挂载的字段请参见**文件存储挂载参数**。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 创建自定义的StorageClass，示例如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-sfs-mount-option
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: nas.csi.everest.io
  csi.storage.k8s.io/fstype: nfs
  everest.io/share-access-to: <your_vpc_id>   # 集群所在VPC的ID
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:                          # 挂载参数
- vers=3
- nolock
- timeo=600
- hard
```

步骤3 StorageClass设置好后，就可以使用这个StorageClass创建PVC，动态创建出的PV会默认带有StorageClass中设置的挂载参数，具体操作步骤请参见**通过动态存储卷使用文件存储**。

步骤4 验证挂载参数是否生效。

本例中将PVC挂载至使用nginx:latest镜像的工作负载，并通过**mount -l**命令查看挂载参数是否生效。

1. 查看已挂载文件存储的Pod，本文中的示例工作负载名称为web-sfs。
   ```
   kubectl get pod | grep web-sfs
   ```
   回显如下：
   ```
   web-sfs-***   1/1     Running   0          23m
   ```
2. 执行以下命令查看挂载参数，其中web-sfs-***为示例Pod。
   ```
   kubectl exec -it web-sfs-*** -- mount -l | grep nfs
   ```
   若回显中的挂载信息与设置的挂载参数一致，说明挂载参数设置成功。
   ```
   <您的共享路径> on /data type nfs
   (rw,relatime,vers=3,rsize=1048576,wsize=1048576,namlen=255,hard,nolock,noresvport,proto=tcp,
   ```



```
timeo=600,retrans=2,sec=sys,mountaddr=**.**.**.**,mountvers=3,mountport=2050,mountproto=tcp
,local_lock=all,addr=**.**.**.**)
```

----结束

8.4.6 将容器应用从 SFS 1.0 迁移到通用文件系统（SFS 3.0）或 SFS Turbo

弹性文件服务（SFS）提供了SFS容量型（SFS 1.0）、通用文件系统（SFS 3.0）和SFS Turbo三种类型的文件系统，关于各类型文件系统的特点和优势请参见**文件系统类型**。

历史版本中，CCE支持在工作负载中挂载SFS 1.0，建议迁移至通用文件系统（SFS 3.0）或SFS Turbo。

根据工作负载类型不同，应用可实现的存储挂载方式也不同。此处动态挂载和静态挂载是从工作负载挂载存储卷的方式进行区分的。

- 动态挂载：仅有状态工作负载支持使用动态挂载，该功能通过**volumeClaimTemplates**字段实现，并依赖于StorageClass动态创建能力。有状态工作负载通过volumeClaimTemplates字段为每一个Pod关联了一个独有的PVC，而这个PVC又会和对应的PV绑定。因此当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。
- 静态挂载：与动态挂载相对，即工作负载中通过**volumes**字段挂载存储卷，所有类型的工作负载均可通过该方法挂载存储。

须知

将容器应用从SFS 1.0迁移到通用文件系统（SFS 3.0）或SFS Turbo的操作步骤一样，两者区别点仅限于：SFS Turbo不支持动态创建，有状态应用在使用SFS Turbo时会限制“动态挂载”的扩容能力。

约束与限制

- 您需要提前将SFS 1.0中的数据迁移至通用文件系统（SFS 3.0）或SFS Turbo，操作步骤请参见**SFS容量型文件系统迁移至其他文件系统**，必要时请联系SFS服务的客服提供支撑。
- SFS必须与集群在同一个VPC内。
- 使用通用文件系统（SFS 3.0）时，您需要提前在集群所在VPC创建一个VPC终端节点，集群需要通过VPC终端节点访问通用文件系统。配置VPC终端节点的方法请参见**配置VPC终端节点**。

静态挂载存储的迁移

静态挂载存储场景下，即工作负载中通过**volumes**字段挂载存储卷，所有类型的工作负载均可通过该方法挂载存储。使用该挂载方法的存储从SFS 1.0迁移到通用文件系统（SFS 3.0）或SFS Turbo的迁移操作步骤一致，本示例以SFS Turbo为例进行操作。

步骤1 选择对应的CCE集群，进入“存储”界面，在“存储卷”页签下单击右上角“创建存储卷PV”。

设置以下关键参数。

- 存储卷类型：选择“极速文件存储”。

- 极速文件存储：选择数据迁移后的极速文件存储卷。
- PV名称：自定义PV名称。
- 访问模式：选择“ReadWriteMany”。
- 回收策略：选择“Retain”。

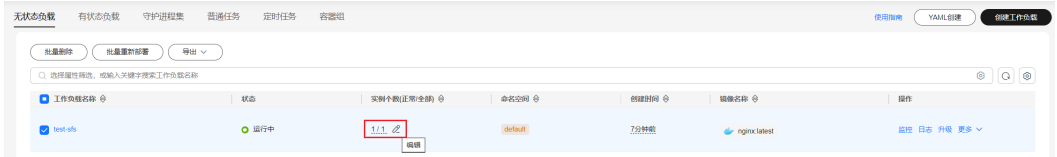
步骤2 PV创建完成后，在“存储卷声明”页签下单击右上角“创建存储卷声明”。

设置以下参数。

- 存储卷声明类型：选择“极速文件存储”。
- PVC名称：自定义PVC名称，需与原SFS1.0 PVC名称不同。
- 创建方式：选择“已有存储卷”。
- 关联存储卷：选择上一步中已创建的存储卷。

PVC创建完成后，如下图所示。

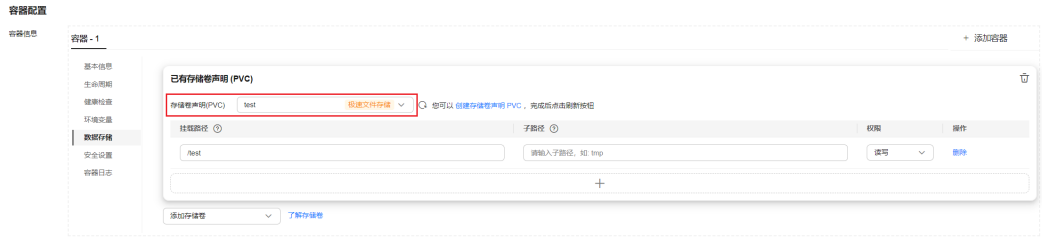
步骤3 在左侧导航栏中选择“工作负载”，找到目标工作负载，将工作负载实例数需要缩容到0。



步骤4 单击工作负载“操作”栏中的“升级”按钮。在“容器配置”中，选择“数据存储”，添加“已有存储卷声明（PVC）”，使用**步骤2**创建的PVC替换工作负载使用的存储卷声明。

须知

迁移后，请保持容器内的挂载路径和子路径与之前挂载SFS1.0时一致。



步骤5 替换完成后，可扩容工作负载实例数。

确认无问题后，可清理CCE侧的SFS 1.0的存储卷。

----结束

有状态应用中的动态挂载存储的迁移

有状态应用中动态挂载的存储从SFS 1.0迁移到通用文件系统（SFS 3.0）或SFS Turbo时，请参考以下步骤进行操作。

迁移至 SFS Turbo

须知

- “动态挂载”的自动扩容能力仅有状态应用支持。
- 由于SFS Turbo不支持动态创建，因此SFS1.0在迁移至SFS Turbo后，该有状态应用不再支持“动态挂载”的自动扩容能力。

步骤1 在集群控制台左侧导航栏中选择“工作负载”，切换至“有状态负载”页签，找到目标工作负载，记录缩容前的实例数，并将工作负载实例数需要缩容到0。

说明

对每个实例使用的PVC均需要执行步骤**步骤2**~**步骤6**。

步骤2 查看有状态应用使用PVC的方式。

```
kubectl get statefulset {statefulset-name} -n {namespace} -ojsonpath='{range .items[*]}{.spec.volumeClaimTemplates}{"\n"}{end}'
```

- 如存在回显，表示使用“动态挂载”，则需继续执行**步骤3**~**步骤7**。
- 如不存在回显，表示未使用“动态挂载”，则无需执行以下步骤，请参考**静态挂载存储的迁移**。

步骤3 修改Pod所使用的PVC对应PV的persistentVolumeReclaimPolicy参数，从“Delete”修改为“Retain”，命令如下：

```
kubectl edit pv {pv-name} -n {namespace}
```

查看修改结果：

```
kubectl get pv {pv-name} -n {namespace} -o yaml |grep persistentVolumeReclaimPolicy
```

例如：

```
 # kubectl get pv pvc-29467e4a-0120-4698-a147-5b75f0ae9a43 -o yaml |grep
persistentVolumeReclaimPolicy
   persistentVolumeReclaimPolicy: Retain
```

步骤4 进入“存储”界面，在“存储卷”页签下记录SFS1.0 PV对应的PVC名称，并删除该PVC。此时该PV处于“已释放”状态。

步骤5 单击右上角“创建存储卷PV”，并设置以下参数。

- 存储卷类型：选择“极速文件存储”。
- 极速文件存储：选择数据迁移后的极速文件存储卷。
- PV名称：自定义PV名称。
- 访问模式：选择“ReadWriteMany”。
- 回收策略：选择“Retain”。

步骤6 进入“存储”界面，在“存储卷声明”页签下单击右上角“创建存储卷声明PVC”，创建与**步骤4**中同名的PVC，并绑定上一步中新建的SFS Turbo PV。

步骤7 所有实例对应的PVC均迁移后，将有状态应用扩容到原来的实例数。

确认无问题后，可前往SFS控制台删除对应的SFS 1.0卷，并在CCE控制台中删除SFS 1.0对应的PV。

----结束

迁移至通用文件系统（SFS 3.0）

须知

- “动态挂载”的自动扩容能力仅有状态应用支持。
- 如有状态应用从SFS 1.0迁移至通用文件系统（SFS 3.0），则该有状态应用支持自动扩容能力。

为了让有状态负载在完成SFS 1.0迁移至通用文件系统（SFS 3.0）后仍支持动态扩容，需将有状态应用中的volumeClaimTemplates使用的存储类从csi-nas修改为csi-sfs。由于使用动态挂载的有状态应用不支持修改volumeClaimTemplates，因此需要先删除有状态应用，然后重建，过程中需要确保配置与迁移前完全一致，包括实例数。

步骤1 在左侧导航栏中选择“工作负载”，找到目标工作负载，记录缩容前的实例数，将工作负载实例数需要缩容到0。

说明

对每个实例使用的PVC均需要执行步骤**步骤2**~**步骤6**。

步骤2 查看有状态应用使用PVC的方式。

```
kubectl get statefulset {statefulset-name} -n {namespace} -ojsonpath='{range .items[*]}{.spec.volumeClaimTemplates}{"\n"}{end}'
```

- 如存在回显，表示使用“动态挂载”，则需继续执行**步骤3**~**步骤7**。
- 如不存在回显，表示未使用“动态挂载”，则无需执行以下步骤，请参考**静态挂载存储的迁移**。

步骤3 修改SFS 1.0对应PV的persistentVolumeReclaimPolicy参数，从“Delete”修改为“Retain”，命令如下：

```
kubectl edit pv {pv-name} -n {namespace}
```

```
claimRef:
  kind: PersistentVolumeClaim
  namespace: default
  name: test-test-0
  uid: 718fa030-ca46-4972-9321-acb06ff73456
  apiVersion: v1
  resourceVersion: '4259856'
persistentVolumeReclaimPolicy: Delete
```

Retain

查看修改结果：

```
kubectl get pv {pv-name} -n {namespace} -o yaml |grep persistentVolumeReclaimPolicy
```

例如：

```
 # kubectl get pv pvc-29467e4a-0120-4698-a147-5b75f0ae9a43 -o yaml |grep
persistentVolumeReclaimPolicy
  persistentVolumeReclaimPolicy: Retain
```

步骤4 进入“存储”界面，在“存储卷”页签下记录SFS1.0 PV对应的PVC名称，并删除该PVC。此时该PV处于“已释放”状态。

步骤5 单击右上角“创建存储卷PV”，并设置以下参数。

- 存储卷类型：选择“文件存储”。
- 文件存储：选择数据迁移后的通用文件系统（SFS 3.0）存储卷。
- PV名称：自定义PV名称。
- 访问模式：选择“ReadWriteMany”。
- 回收策略：请按需设置。
  - Delete：删除动作会将PersistentVolume对象从Kubernetes中移除，同时也会从外部基础设施中移除所关联的存储资产。
  - Retain：当PersistentVolumeClaim对象被删除时，PersistentVolume卷仍然存在，对应的数据卷被视为“已释放(released)”。

步骤6 进入“存储”界面，在“存储卷声明”页签下单击右上角“创建存储卷声明PVC”，创建同名的PVC，并绑定上一步中新建的通用文件系统（SFS 3.0） PV。

- 存储卷声明类型：选择“文件存储”。
- PVC名称：修改为**步骤4**中同名的PVC。
- 创建方式：选择“已有存储卷”。
- 关联存储卷：选择上一步中已创建的存储卷。

步骤7 前往“工作负载”页面，查看原来的有状态工作负载，单击“更多>编辑YAML”，单击“下载”或复制YAML文件的全部内容，在本地进行备份。

步骤8 删除原来的有状态应用，并将上一步复制的工作负载YAML配置进行以下修改：

- volumeClaimTemplates字段下的storageClassName: 'csi-nas'修改为'csi-sfs'。
- 删除"resourceVersion"字段及其参数，因为该字段在创建时不可指定。

```
volumeClaimTemplates:
  - apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: tets
      namespace: default
      labels:
        failure-domain.beta.kubernetes.io/region: cn-east-3
        failure-domain.beta.kubernetes.io/zone: cn-east-3a
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 10Gi
      storageClassName: csi-sfs
podManagementPolicy: OrderedReady
```

步骤9 单击右上角的“YAML创建”，单击“导入”或粘贴修改后的YAML文件内容，并单击“创建”。

步骤10 待工作负载创建完成后，将有状态应用扩容到原来的实例数。

确认无问题后，可前往SFS控制台删除对应的SFS 1.0卷，并在CCE控制台中删除SFS 1.0对应的PV。

----结束

8.5 极速文件存储（SFS Turbo）

8.5.1 极速文件存储概述

极速文件存储介绍

CCE支持将极速文件存储（SFS Turbo）创建的存储卷挂载到容器的某一路径下，以满足数据持久化的需求。极速文件存储具有按需申请，快速供给，弹性扩展，方便灵活等特点，适用于海量小文件业务，例如DevOps、容器微服务、企业办公等应用场景。

SFS Turbo为用户提供一个完全托管的共享文件存储，能够弹性伸缩至320TB规模，具备高可用性和持久性，为海量的小文件、低延迟高IOPS型应用提供有力支持。

- **符合标准文件协议：**用户可以将文件系统挂载给服务器，像使用本地文件目录一样。
- **数据共享：**多台服务器可挂载相同的文件系统，数据可以共享操作和访问。
- **私有网络：**数据访问必须在数据中心内部网络中。
- **安全隔离：**直接使用云上现有IaaS服务构建独享的云文件存储，为租户提供数据隔离保护和IOPS性能保障。
- **应用场景：**适用于多读多写（ReadWriteMany）场景下的各种工作负载（Deployment/StatefulSet）、守护进程集（DaemonSet）和普通任务（Job）使用，主要面向高性能网站、日志存储、DevOps、企业办公等场景。

极速文件存储性能

关于极速文件存储的性能参数，请参考**文件系统类型**。

使用场景

极速文件存储支持以下挂载方式：

- **通过静态存储卷使用已有极速文件存储**：即静态创建的方式，需要先使用已有的文件存储创建PV，然后通过PVC在工作负载中挂载存储。
- **通过StorageClass动态创建SFS Turbo子目录**：SFS Turbo支持动态创建子目录并挂载到容器，实现共享使用SFS Turbo，从而更加经济合理的利用SFS Turbo存储容量。

计费说明

极速文件存储不具备动态创建能力，只能挂载已创建完成的极速文件存储，您可根据需求选择按需计费或者包年包月套餐。关于极速文件存储的价格详情，请参见**文件存储计费说明**。

8.5.2 通过静态存储卷使用已有极速文件存储

极速文件存储（SFS Turbo）是一种具备高可用性和持久性的共享文件系统，适合海量的小文件、低延迟高IOPS的应用。本文介绍如何使用已有的极速文件存储静态创建PV和PVC，并在工作负载中实现数据持久化与共享性。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经创建好一个状态可用的SFS Turbo，并且SFS Turbo与集群在同一个VPC内。

约束与限制

- 支持多个PV挂载同一个SFS或SFS Turbo，但有如下限制：
  - 多个不同的PVC/PV使用同一个底层SFS或SFS Turbo卷时，如果挂载至同一Pod使用，会因为PV的volumeHandle参数值相同导致无法为Pod挂载所有PVC，出现Pod无法启动的问题，请避免该使用场景。
  - PV中persistentVolumeReclaimPolicy参数建议设置为Retain，否则可能存在一个PV删除时级联删除底层卷，其他关联这个底层卷的PV会由于底层存储被删除导致使用出现异常。
  - 重复用底层存储时，建议在应用层做好多读多写的隔离保护，防止产生的数据覆盖和丢失。
- 对SFS Turbo类型的存储来说，删除集群或删除PVC时不会回收包周期的SFS Turbo资源，您需要在SFS Turbo控制台中自行回收。

通过控制台使用已有极速文件存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 静态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“极速文件存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”来静态创建PVC。 本文示例中选择“新建存储卷”，可通过控制台同时创建PV及PVC。
关联存储卷[a]	选择集群中已有的PV卷，需要提前创建PV，请参考**相关操作**中的“创建存储卷”操作。 本文示例中无需选择。

参数	描述
极速文件存储[b]	单击“选择极速文件存储”，您可以在新页面中勾选满足要求的极速文件存储，并单击“确定”。
子目录[b]	选择是否使用子目录创建PV。请填写子目录绝对路径，例如/a/b，并确保子目录已存在且可用。
PV名称[b]	输入PV名称，同一集群内的PV名称需唯一。
访问模式[b]	极速文件存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。
回收策略[b]	不使用子目录创建PV时，仅支持Retain，表示删除PVC时PV不会被同时删除，详情请参见**PV回收策略**。选择使用子目录创建PV时，支持选择Delete。
子目录回收策略[b]	删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用，当PV回收策略为"Delete"时支持配置。 – 保留：删除PVC，PV会被删除，但**PV关联的子目录会被保留**。 – 删除：删除PVC，**PV及其关联的子目录均会被删除**。
挂载参数[b]	输入挂载参数键值对，详情请参见**设置极速文件存储挂载参数**。

说明

a：创建方式选择“已有存储卷”时可设置。

b：创建方式选择“新建存储卷”时可设置。

2. 单击“创建”，将同时为您创建存储卷声明和存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

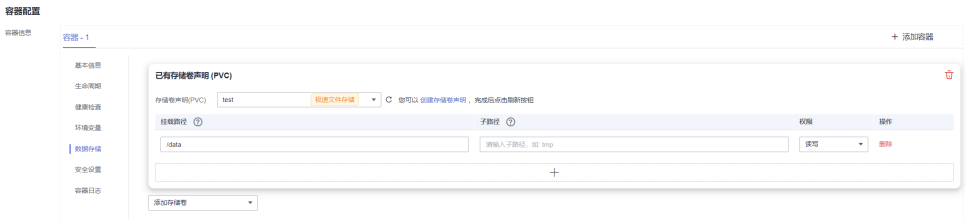
   本文主要为您介绍存储卷的挂载使用，如**表8-28**，其他参数详情请参见**工作负载**。

表 8-28 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的极速文件存储卷。

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到极速文件存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

通过 kubectl 命令行使用已有极速文件存储

您可以根据不同的使用场景选择不同的创建方式。

使用已有极速文件存储

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-sfsturbo.yaml文件。

   示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
```



```
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
  name: pv-sfsturbo    # PV的名称
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，极速文件存储必须为ReadWriteMany
  capacity:
    storage: 500Gi     # 极速文件存储容量大小
  csi:
    driver: sfsturbo.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: <your_volume_id>  # 极速文件存储的ID
    volumeAttributes:
      everest.io/share-export-location: <your_location>  # 极速文件存储的共享路径
      everest.io/enterprise-project-id: <your_project_id>  # 极速文件存储的项目ID
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-sfsturbo          # SFS Turbo存储类名称
  mountOptions: []                        # 挂载参数
```

表 8-29 关键参数说明

参数	是否必选	描述
volumeHandle	是	使用整个SFS Turbo创建PV时，填写极速文件存储的ID。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，并选择SFS Turbo。在列表中单击对应的SFS Turbo文件存储名称，在详情页中复制“ID”后的内容即可。
everest.io/share-export-location	是	极速文件存储的共享路径。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，选择SFS Turbo，在弹性文件服务列表中可以看到“共享路径”列，即为极速文件存储的共享路径。
everest.io/enterprise-project-id	否	极速文件存储的项目ID。 获取方法：在弹性文件服务控制台，单击左侧栏目树中的“SFS Turbo”，单击要对接的SFS Turbo名称进入详情页，在“基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可。

mountOptions	否	挂载参数。
		不设置时默认配置为如下配置，具体说明请参见**设置极速文件存储挂载参数**。
		`mountOptions:`
		`- vers=3`
		`- timeo=600`
		`- nolock`
		`- hard`

参数	是否必选	描述
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。详情请参见**PV回收策略**。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。
storage	是	PVC申请容量，单位为Gi。
storageClassName	是	极速文件存储对应的存储类名称为csi-sfsturbo。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-sfsturbo.yaml
```

步骤3 创建PVC。

1. 创建pvc-sfsturbo.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfsturbo
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/enterprise-project-id: <your_project_id>  # 极速文件存储的项目ID
spec:
  accessModes:
  - ReadWriteMany                # 极速文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 500Gi             # 极速文件存储大小
  storageClassName: csi-sfsturbo      # SFS Turbo存储类名称，必须与PV的存储类一致
  volumeName: pv-sfsturbo    # PV的名称
```

表 8-30 关键参数说明

参数	是否必选	描述
everest.io/enterprise-project-id	否	极速文件存储的项目ID。 获取方法：在弹性文件服务控制台，单击左侧栏目树中的“SFS Turbo”，单击要对接的SFS Turbo名称进入详情页，在“基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可。
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
storageClassName	是	存储类名称，必须与1中PV的存储类一致。 极速文件存储对应的存储类名称为csi-sfsturbo。

参数	是否必选	描述
volumeName	是	PV的名称，必须与1中PV名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-sfsturbo.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将极速文件存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfsturbo-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-sfsturbo-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-sfsturbo    #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载极速文件存储的应用。
```
kubectl apply -f web-demo.yaml
```
工作负载创建成功后，您可以尝试**验证数据持久化及共享性**。

----结束

使用已有极速文件存储的子目录

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-sfsturbo.yaml文件。

示例如下：
```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only    # 可选，当使用子目录且回收策略为Delete时使用，表示删除PVC时，PV会被删除，但PV关联的子目录会被保留
  name: pv-sfsturbo    # PV的名称
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，极速文件存储必须为ReadWriteMany
```



```
capacity:
  storage: 500Gi      # 极速文件存储容量大小
csi:
  driver: sfsturbo.csi.everest.io    # 挂载依赖的存储驱动
  fsType: nfs
  volumeHandle: pv-sfsturbo   # 子目录场景下为pv名称
  volumeAttributes:
    everest.io/share-export-location: <sfsturbo_path>:/<absolute_path>   # 极速文件存储的共享路径+子目录
    everest.io/enterprise-project-id: <your_project_id>  # 极速文件存储的项目ID
    storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    everest.io/volume-as: absolute-path   # 可选，表示使用SFS Turbo子目录
persistentVolumeReclaimPolicy: Retain    # 回收策略，自动创建子目录时支持设置为Delete
storageClassName: csi-sfsturbo          # SFS Turbo存储类名称
mountOptions: []                        # 挂载参数
```

表 8-31 关键参数说明

参数	是否必选	描述
volumeHandle	是	使用SFS Turbo子目录创建PV时，填写PV名称。
everest.io/share-export-location	是	极速文件存储子目录的共享路径。 格式为： *{sfsturbo_path}:/{absolute_path}* 其中极速文件存储的共享路径获取方法如下： 在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，选择SFS Turbo，在弹性文件服务列表中可以看到“共享路径”列，即为极速文件存储的共享路径。
everest.io/enterprise-project-id	否	极速文件存储的项目ID。 获取方法：在弹性文件服务控制台，单击左侧栏目树中的“SFS Turbo”，单击要对接的SFS Turbo名称进入详情页，在“基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可。
mountOptions	否	挂载参数。 不设置时默认配置为如下配置，具体说明请参见**设置极速文件存储挂载参数**。 mountOptions: - vers=3 - timeo=600 - nolock - hard

参数	是否必选	描述
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。详情请参见**PV回收策略**。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。 **Delete**：自动创建子目录时支持设置，表示删除PVC时，同时删除PV。
everest.io/reclaim-policy	否	删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用。仅当PV回收策略为"Delete"时生效，取值如下： – retain-volume-only：表示删除PVC时，PV会被删除，但**PV关联的子目录会被保留**。 – delete：表示删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。
everest.io/volume-as	否	固定取值为“absolute-path”，表示使用动态创建SFS Turbo子目录。 集群中需安装2.3.23及以上版本的Everest插件。
storage	是	PVC申请容量，单位为Gi。使用子目录时，该参数值无实际意义，仅作校验需要（不能为空和0）。
storageClassName	是	极速文件存储对应的存储类名称为csi-sfsturbo。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-sfsturbo.yaml
```

步骤3 创建PVC。

1. 创建pvc-sfsturbo.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfsturbo
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/enterprise-project-id: <your_project_id>  # 极速文件存储的项目ID
spec:
  accessModes:
  - ReadWriteMany                # 极速文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 500Gi             # 极速文件存储大小
```



```
storageClassName: csi-sfsturbo      # SFS Turbo存储类名称，必须与PV的存储类一致
volumeName: pv-sfsturbo    # PV的名称
```

表 8-32 关键参数说明

参数	是否必选	描述
everest.io/enterprise-project-id	否	极速文件存储的项目ID。 获取方法：在弹性文件服务控制台，单击左侧栏目树中的“SFS Turbo”，单击要对接的SFS Turbo名称进入详情页，在“基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可。
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
storageClassName	是	存储类名称，必须与1中PV的存储类一致。 极速文件存储对应的存储类名称为csi-sfsturbo。
volumeName	是	PV的名称，必须与1中PV名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-sfsturbo.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将极速文件存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-sfsturbo-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-sfsturbo-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-sfsturbo    #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载极速文件存储的应用。
```
kubectl apply -f web-demo.yaml
```


工作负载创建成功后，您可以尝试**验证数据持久化及共享性**。

----结束

验证数据持久化及共享性

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-mjhm9   1/1     Running   0          46s
web-demo-846b489584-wvv5s   1/1     Running   0          46s
```
2. 依次执行以下命令，查看Pod的/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
两个Pod均无返回结果，说明/data路径下无文件。

步骤2 执行以下命令，在/data路径下创建static文件。
```
kubectl exec web-demo-846b489584-mjhm9 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
```
预期输出如下：
```
static
```

步骤4 验证数据持久化

1. 执行以下命令，删除名称为web-demo-846b489584-mjhm9的Pod。
```
kubectl delete pod web-demo-846b489584-mjhm9
```
预期输出如下：
```
pod "web-demo-846b489584-mjhm9" deleted
```
删除后，Deployment控制器会自动重新创建一个副本。
2. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下，web-demo-846b489584-d4d4j为新建的Pod：
```
web-demo-846b489584-d4d4j   1/1     Running   0          110s
web-demo-846b489584-wvv5s   1/1     Running   0          7m50s
```
3. 执行以下命令，验证新建的Pod中/data路径下的文件是否更改。
```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```
预期输出如下：
```
static
```
static文件仍然存在，则说明数据可持久化保存。

步骤5 验证数据共享性

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-d4d4j   1/1     Running   0          7m
web-demo-846b489584-wvv5s   1/1     Running   0          13m
```
2. 执行以下命令，在任意一个Pod的/data路径下创建share文件。本例中选择名为web-demo-846b489584-d4d4j的Pod。
```
kubectl exec web-demo-846b489584-d4d4j --  touch /data/share
```
并查看该Pod中/data路径下的文件。


```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```

预期输出如下：

```
share
static
```

3. 由于写入share文件的操作未在名为web-demo-846b489584-wvv5s的Pod中执行，在该Pod中查看/data路径下是否存在文件以验证数据共享性。

```
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```

预期输出如下：

```
share
static
```

如果在任意一个Pod中的/data路径下创建文件，其他Pod下的/data路径下均存在此文件，则说明两个Pod共享一个存储卷。

----结束

相关操作

您还可以执行表8-33中的基本操作。

表 8-33 其他操作

操作	说明	操作步骤
创建存储卷	通过CCE控制台单独创建PV。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷”页签。单击右上角“创建存储卷”，在弹出的窗口中填写存储卷声明参数。 ● 存储卷类型：选择“极速文件存储”。 ● 极速文件存储：单击“选择极速文件存储”，在新页面中勾选满足要求的极速文件存储，并单击“确定”。 ● PV名称：输入PV名称，同一集群内的PV名称需唯一。 ● 访问模式：仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见存储卷访问模式。 ● 回收策略：仅支持Retain，详情请参见PV回收策略。 ● 挂载参数：输入挂载参数键值对，详情请参见设置极速文件存储挂载参数。 2. 单击“创建”。
扩容极速文件存储存储卷	通过CCE控制台快速扩容已挂载的极速文件存储。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。

操作	说明	操作步骤
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.5.3 设置极速文件存储挂载参数

本章节主要介绍如何设置极速文件存储的挂载参数。极速文件存储仅支持在PV中设置挂载参数，然后通过创建PVC绑定PV。

前提条件

CCE容器存储（Everest）版本要求**1.2.8及以上**版本。插件主要负责将挂载参数识别并传递给底层存储，指定参数是否有效依赖于底层存储是否支持。

约束与限制

- 挂载参数暂不支持安全容器。
- 由于NFS协议限制，默认情况下，对于某个节点多次挂载同一文件存储的场景，涉及链路的挂载参数（如timeo）仅在第一次挂载时生效。例如，节点上运行的多个Pod同时挂载同一文件存储，后设置的挂载参数不会覆盖已有参数值。针对上述场景希望设置不同的挂载参数，可以同时设置nosharecache挂载参数。

极速文件存储挂载参数

CCE的存储插件everest在挂载极速文件存储时默认设置了如**表8-34**所示的参数。

表 8-34 极速文件存储挂载参数

参数	参数值	描述
vers	3	文件系统版本，目前只支持NFSv3。取值：3
nolock	无需填写	选择是否使用NLM协议在服务器上锁文件。当选择nolock选项时，锁对于同一主机的应用有效，对不同主机不受锁的影响。
timeo	600	NFS客户端重传请求前的等待时间(单位为0.1秒)。建议值：600。

参数	参数值	描述
hard/soft	无需填写	挂载方式类型。 • 取值为hard，即使用硬连接方式，若NFS请求超时，则客户端一直重新请求直至成功。 • 取值为soft，即软挂载方式挂载系统，若NFS请求超时，则客户端向调用程序返回错误。 默认为hard。
sharecache/nosharecache	无需填写	设置客户端并发挂载同一文件系统时数据缓存和属性缓存的共享方式。设置为sharecache时，多个挂载共享共享同一缓存。设为nosharecache时，每个挂载各有一个缓存。默认为sharecache。 **说明** 设置nosharecache禁用共享缓存会对性能产生一定影响。每次挂载都会重新获取挂载信息，会增加与NFS服务器的通信开销和NFS客户端的内存消耗，同时同客户端设置nosharecache存在cache不一致的风险。因此，应该根据具体情况进行权衡，以确定是否需要使用nosharecache选项。

除了以上参数外，您还可以设置其他的文件存储挂载参数，具体请参见**挂载NFS文件系统到云服务器（Linux）**。

在 PV 中设置挂载参数

在PV中设置挂载参数可以通过mountOptions字段实现，如下所示，mountOptions支持挂载的字段请参见**极速文件存储挂载参数**。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在PV中设置挂载参数，示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
  name: pv-sfsturbo    # PV的名称
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，极速文件存储必须为ReadWriteMany
  capacity:
    storage: 500Gi       # 极速文件存储容量大小
  csi:
    driver: sfsturbo.csi.everest.io    # 挂载依赖的存储驱动
    fsType: nfs
    volumeHandle: {your_volume_id}  # 极速文件存储的ID
    volumeAttributes:
      everest.io/share-export-location: {your_location}  # 极速文件存储的共享路径
      everest.io/enterprise-project-id: {your_project_id} # 极速文件存储的项目ID
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-sfsturbo          # SFS Turbo存储类名称
  mountOptions:                           # 挂载参数
  - vers=3
  - nolock
  - timeo=600
  - hard
```


步骤3 PV创建后，可以创建PVC关联PV，然后在工作负载的容器中挂载，具体操作步骤请参见**通过静态存储卷使用已有极速文件存储**。

步骤4 验证挂载参数是否生效。

本例中将PVC挂载至使用nginx:latest镜像的工作负载，并通过**mount -l**命令查看挂载参数是否生效。

1. 查看已挂载文件存储的Pod，本文中的示例工作负载名称为web-sfsturbo。
   ```
   kubectl get pod | grep web-sfsturbo
   ```
   回显如下：
   ```
   web-sfsturbo-***  1/1   Running  0        23m
   ```
2. 执行以下命令查看挂载参数，其中web-sfsturbo-***为示例Pod。
   ```
   kubectl exec -it web-sfsturbo-*** -- mount -l | grep nfs
   ```
   若回显中的挂载信息与设置的挂载参数一致，说明挂载参数设置成功。
   ```
   {您的挂载地址} on /data type nfs
   (rw,relatime,vers=3,rsize=1048576,wsize=1048576,namlen=255,hard,nolock,noresvport,proto=tcp,
   timeo=600,retrans=2,sec=sys,mountaddr=**.**.**.**,mountvers=3,mountport=20048,mountproto=tc
   p,local_lock=all,addr=**.**.**.**)
   ```

----结束

8.5.4 通过动态存储卷创建 SFS Turbo 子目录（推荐）

通常情况下，在工作负载容器中挂载SFS Turbo类型的存储卷时，默认会将根目录挂载到容器中。而SFS Turbo的容量最小为500G，超出了大多数工作负载所需的容量，导致存储容量的浪费。为了更加经济合理地利用存储容量，CCE支持在创建PVC时动态创建SFS Turbo子目录，实现不同工作负载共享使用SFS Turbo。

前提条件

- 您已经创建好一个集群，并且在该集群中安装2.3.23及以上版本的**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经创建好一个状态可用的SFS Turbo，并且SFS Turbo与集群在同一个VPC内。

通过控制台动态创建 SFS Turbo 子目录

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“极速文件存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	选择“动态创建子目录”。
存储类	选择极速文件存储对应的存储类为csi-sfsturbo。

参数	描述
访问模式	极速文件存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。
极速文件存储	单击“选择极速文件存储”，您可以在新页面中勾选满足要求的极速文件存储，并单击“确定”。
子目录	请填写子目录绝对路径，例如/a/b。
子目录回收策略	删除PVC时是否保留子目录。 ● 保留：删除PVC，PV会被删除，但**PV关联的子目录会被保留**。 ● 删除：删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。

步骤3 单击“创建”，将同时为您创建存储卷声明和存储卷。

您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

----结束

通过 kubectl 命令行动态创建 SFS Turbo 子目录

步骤1 使用kubectl连接集群。

步骤2 创建pvc-sfsturbo-subpath.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sfsturbo-subpath    # PVC的名称
  namespace: default
  annotations:
    everest.io/volume-as: absolute-path              # 表示使用SFS Turbo子目录
    everest.io/sfsturbo-share-id: <sfsturbo_id>       # SFS Turbo的ID
    everest.io/path: /a                              # 自动创建的子目录，必须为绝对路径
    everest.io/reclaim-policy: retain-volume-only      # 表示删除PVC时，PV会被删除，但PV关联的子目录会被保留
spec:
  accessModes:
    - ReadWriteMany      # SFS Turbo必须为ReadWriteMany
  resources:
    requests:
      storage: 10Gi      # 对于SFS Turbo子目录类型的PVC，此处无实际意义，仅作校验需要（不能为空和0）
  storageClassName: csi-sfsturbo      # SFS Turbo存储类名称
```

表 8-35 关键参数说明

参数	是否必选	描述
everest.io/volume-as	否	固定取值为“absolute-path”，表示使用动态创建SFS Turbo子目录。

参数	是否必选	描述
everest.io/sfsturbo-share-id	否	SFS Turbo的ID。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，并选择SFS Turbo。在列表中单击对应的极速弹性文件存储名称，在详情页中复制“ID”后的内容即可。
everest.io/path	否	自动创建的子目录，必须为绝对路径。
everest.io/reclaim-policy	否	删除PVC时是否保留子目录，该参数需与**PV回收策略**配合使用。仅当PV回收策略为"Delete"时生效，取值如下： ● retain-volume-only：表示删除PVC时，PV会被删除，但**PV关联的子目录会被保留**。 ● delete：表示删除PVC，**PV及其关联的子目录均会被删除**。 **说明** 删除子目录时，仅删除PVC参数中设置的子目录绝对路径，不会级联删除上层目录。
storage	是	PVC申请容量，单位为Gi。 对SFS Turbo子目录类型的PVC来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处可以设定为固定值10Gi。

步骤3 执行以下命令，创建PVC。

```
kubectl apply -f pvc-sfsturbo-subpath.yaml
```

----结束

8.5.5 通过 StorageClass 动态创建 SFS Turbo 子目录

背景信息

SFS Turbo容量最小500G，且不是按使用量计费。SFS Turbo挂载时默认将根目录挂载到容器，而通常情况下负载不需要这么大容量，造成浪费。

everest插件支持一种在SFS Turbo下动态创建子目录的方法，能够在SFS Turbo下动态创建子目录并挂载到容器，这种方法能够共享使用SFS Turbo，从而更加经济合理的利用SFS Turbo存储容量。

约束与限制

- 仅支持1.15+集群。
- 集群必须使用everest插件，插件版本要求1.1.13+。
- 不支持安全容器。
- 使用everest 1.2.69之前或2.1.11之前的版本时，使用子目录功能时不能同时并发创建超过10个PVC。推荐使用everest 1.2.69及以上或2.1.11及以上的版本。

- subpath类型的卷实际为SFS Turbo的子目录，对该类型的PVC进行扩容仅会调整PVC声明的资源范围，并不会调整SFS Turbo资源的总容量。若SFS Turbo资源总容量不足，subpath类型卷的实际可使用的容量大小也会受限，您需要前往SFS Turbo界面进行扩容。

  同理，删除subpath类型的卷也不会实际删除后端的SFS Turbo资源。

创建 subpath 类型 SFS Turbo 存储卷

步骤1 创建SFS Turbo资源，选择网络时，请选择与集群相同的VPC与子网。

步骤2 新建一个StorageClass的YAML文件，例如sfsturbo-subpath-sc.yaml。

配置示例：

```
apiVersion: storage.k8s.io/v1
allowVolumeExpansion: true
kind: StorageClass
metadata:
  name: sfsturbo-subpath-sc
mountOptions:
- lock
parameters:
  csi.storage.k8s.io/csi-driver-name: sfsturbo.csi.everest.io
  csi.storage.k8s.io/fstype: nfs
  everest.io/archive-on-delete: "true"
  everest.io/share-access-to: 7ca2dba2-1234-1234-1234-626371a8fb3a
  everest.io/share-expand-type: bandwidth
  everest.io/share-export-location: 192.168.1.1:/sfsturbo/
  everest.io/share-source: sfs-turbo
  everest.io/share-volume-type: STANDARD
  everest.io/volume-as: subpath
  everest.io/volume-id: 0d773f2e-1234-1234-1234-de6a35074696
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

其中：

- name：storageclass的名称。
- mountOptions：选填字段；mount挂载参数。
  - everest 1.2.8以下，1.1.13以上版本仅开放对nolock参数配置，mount操作默认使用nolock参数，无需配置。nolock=false时，使用lock参数。
  - everest 1.2.8及以上版本支持更多参数，默认使用如下所示配置，具体请参见**设置挂载参数**。**此处不能配置为nolock=true，会导致挂载失败。**

    ```
    mountOptions:
    - vers=3
    - timeo=600
    - nolock
    - hard
    ```
- everest.io/volume-as：该参数需设置为“subpath”来使用subpath模式。
- everest.io/share-access-to：选填字段。subpath模式下，填写SFS Turbo资源的所在VPC的ID。
- everest.io/share-expand-type：选填字段。若SFS Turbo资源存储类型为增强版（标准型增强版、性能型增强版），设置为bandwidth。
- everest.io/share-export-location：挂载目录配置。由SFS Turbo共享路径和子目录组成，共享路径可至SFS Turbo服务页面查询，子路径由用户自定义，后续指定该StorageClass创建的PVC均位于该子目录下。

- everest.io/share-volume-type：选填字段。填写SFS Turbo的类型。标准型为STANDARD，性能型为PERFORMANCE。对于增强型需配合“everest.io/share-expand-type”字段使用，everest.io/share-expand-type设置为“bandwidth”。
- everest.io/zone：选填字段。指定SFS Turbo资源所在的可用区。
- everest.io/volume-id：SFS Turbo资源的卷ID，可至SFS Turbo界面查询。
- everest.io/archive-on-delete：若该参数设置为“true”，在回收策略为“Delete”时，删除PVC会将PV的原文档进行归档，归档目录的命名规则“archived-$pv名称.时间戳”。该参数设置为“false”时，会将PV对应的SFS Turbo子目录删除。默认设置为“true”，即删除PVC时进行归档。

步骤3 执行**kubectl create -f sfsturbo-subpath-sc.yaml**。

步骤4 新建一个PVC的YAML文件，sfs-turbo-test.yaml。

配置示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sfs-turbo-test
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Gi
  storageClassName: sfsturbo-subpath-sc
  volumeMode: Filesystem
```

其中：

- name: PVC的名称。
- storageClassName：SC的名称。
- storage：subpath模式下，调整该参数的大小不会对SFS Turbo容量进行调整。实际上，subpath类型的卷是SFS Turbo中的一个文件路径，因此在PVC中对subpath类型的卷扩容时，不会同时扩容SFS Turbo资源。

  说明

  subpath子目录的容量受限于SFS Turbo资源的总容量，若SFS Turbo资源总容量不足，请您及时到SFS Turbo界面调整。

步骤5 执行**kubectl create -f sfs-turbo-test.yaml**。

----结束

创建 Deployment 挂载已有数据卷

步骤1 新建一个Deployment的YAML文件，例如deployment-test.yaml。

配置示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-turbo-subpath-example
  namespace: default
  generation: 1
  labels:
    appgroup: ''
```



```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-turbo-subpath-example
  template:
    metadata:
      labels:
        app: test-turbo-subpath-example
    spec:
      containers:
      - image: nginx:latest
        name: container-0
        volumeMounts:
        - mountPath: /tmp
          name: pvc-sfs-turbo-example
      restartPolicy: Always
      imagePullSecrets:
      - name: default-secret
      volumes:
      - name: pvc-sfs-turbo-example
        persistentVolumeClaim:
          claimName: sfs-turbo-test
```

其中：

- name：创建的工作负载名称。
- image：工作负载的镜像。
- mountPath：容器内挂载路径，示例中挂载到“/tmp”路径。
- claimName：已有的PVC名称。

步骤2 创建Deployment负载。

kubectl create -f deployment-test.yaml

----结束

StatefulSet 动态创建 subpath 模式的数据卷

步骤1 新建一个StatefulSet的YAML文件，例如statefulset-test.yaml。

配置示例：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: test-turbo-subpath
  namespace: default
  generation: 1
  labels:
    appgroup: ''
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test-turbo-subpath
  template:
    metadata:
      labels:
        app: test-turbo-subpath
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        pod.alpha.kubernetes.io/initialized: 'true'
    spec:
      containers:
```



```
      - name: container-0
        image: 'nginx:latest'
        resources: {}
        volumeMounts:
          - name: sfs-turbo-160024548582479676
            mountPath: /tmp
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        imagePullPolicy: IfNotPresent
    restartPolicy: Always
    terminationGracePeriodSeconds: 30
    dnsPolicy: ClusterFirst
    securityContext: {}
    imagePullSecrets:
      - name: default-secret
    affinity: {}
    schedulerName: default-scheduler
volumeClaimTemplates:
  - metadata:
      name: sfs-turbo-160024548582479676
      namespace: default
      annotations: {}
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 10Gi
      storageClassName: sfsturbo-subpath-sc
serviceName: wwww
podManagementPolicy: OrderedReady
updateStrategy:
  type: RollingUpdate
revisionHistoryLimit: 10
```

其中：

- name：创建的工作负载名称。
- image：工作负载的镜像。
- mountPath：容器内挂载路径，示例中挂载到“/tmp”路径。
- “spec.template.spec.containers.volumeMounts.name ”和“spec.volumeClaimTemplates.metadata.name”有映射关系，必须保持一致。
- storageClassName: 填写自建的SC名称。

步骤2 创建StatefulSet负载。

kubectl create -f statefulset-test.yaml

----结束

8.6 对象存储（OBS）

8.6.1 对象存储概述

对象存储介绍

对象存储服务（Object Storage Service，OBS）提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。适合企业备份/归档、视频点播、视频监控等多种数据存储场景。

- **标准接口：**具备标准Http Restful API接口，用户必须通过编程或第三方工具访问对象存储。
- **数据共享：**服务器、嵌入式设备、IOT设备等所有调用相同路径，均可访问共享的对象存储数据。
- **公共/私有网络：**对象存储数据允许在公网访问，满足互联网应用需求。
- **容量与性能：**容量无限制，性能较高（IO读写时延10ms级）。
- **应用场景：**适用于（基于OBS界面、OBS工具、OBS SDK等）的一次上传共享多读（ReadOnlyMany）的各种工作负载（Deployment/StatefulSet）和普通任务（Job）使用，主要面向大数据分析、静态网站托管、在线视频点播、基因测序、智能视频监控、备份归档、企业云盘（网盘）等场景。

对象存储规格

对象存储提供了多种存储类别，从而满足客户业务对存储性能、成本的不同诉求。

- 对象桶：提供高可靠、高性能、高安全、低成本的数据存储能力，无文件数量限制、容量限制。
  - 标准存储：访问时延低和吞吐量高，因而适用于有大量热点文件（平均一个月多次）或小文件（小于1MB），且需要频繁访问数据的业务场景，例如：大数据、移动应用、热点视频、社交图片等场景。
  - 低频访问存储：适用于不频繁访问（平均一年少于12次）但在需要时也要求快速访问数据的业务场景，例如：文件同步/共享、企业备份等场景。与标准存储相比，低频访问存储有相同的数据持久性、吞吐量以及访问时延，且成本较低，但是可用性略低于标准存储。
- 并行文件系统：并行文件系统（Parallel File System）是对象存储服务的子产品，是经过优化的高性能文件语义系统，主要应用于大数据场景。详细介绍请参见**什么是并行文件系统**。

关于对象存储的详细介绍，请以**对象存储类别**为准。

性能说明

容器负载挂载对象存储时，每挂载一个对象存储卷，后端会产生一个常驻进程。当负载使用对象存储数过多或大量读写对象存储文件时，常驻进程会占用大量内存，部分场景下内存消耗量参考**表8-36**，为保证负载稳定运行，建议负载使用的对象存储卷数量不超过其申请的内存GiB数量，如负载的申请的内存规格为4GiB，则建议其使用的对象存储数**不超过**4。

表 8-36 单个对象存储常驻进程内存消耗

测试项目	内存消耗
长稳运行	约50m
2并发写10M文件	约110m
4并发写10M文件	约220m
单写100G文件	约300m

使用场景

根据使用场景不同，对象存储支持以下挂载方式：

- **通过静态存储卷使用已有对象存储**：即静态创建的方式，需要先使用已有的对象存储创建PV，然后通过PVC在工作负载中挂载存储。适用于已有可用的底层存储或底层存储需要包周期的场景。
- **通过动态存储卷使用对象存储**：即动态创建的方式，无需预先创建对象存储，在创建PVC时通过指定存储类（StorageClass），即可自动创建对象存储和对应的PV对象。适用于无可用的底层存储，需要新创建的场景。

计费说明

- 挂载对象存储类型的存储卷时，通过StorageClass**自动创建**的对象存储默认创建计费模式为“按需计费”。关于对象存储的价格信息，请参见**对象存储计费说明**。
- 如需使用包周期的对象存储，请**使用已有的对象存储**进行挂载。

8.6.2 通过静态存储卷使用已有对象存储

本文介绍如何使用已有的对象存储静态创建PV和PVC，并在工作负载中实现数据持久化与共享性。

前提条件

- 您已经创建好一个集群，并且在该集群中安装**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 使用对象存储时，挂载点不支持修改属组和权限。
- 使用PVC挂载对象存储时，负载每挂载一个对象存储卷，后端会产生一个常驻进程。当负载使用对象存储数过多或大量读写对象存储文件时，常驻进程会占用大量内存，为保证负载稳定运行，建议负载使用的对象存储卷数量不超过其申请的内存GiB数量，如负载的申请的内存规格为4GiB，则建议其使用的对象存储数**不超过**4。
- 安全容器不支持使用对象存储。
- 挂载普通桶时不支持硬链接（Hard Link）。
- 支持多个PV挂载同一个对象存储，但有如下限制：
  - 多个不同的PVC/PV使用同一个底层对象存储卷时，如果挂载至同一Pod使用，会因为PV的volumeHandle参数值相同导致无法挂载，请避免该使用场景。
  - PV中persistentVolumeReclaimPolicy参数建议设置为Retain，否则可能存在一个PV删除时，级联删除底层卷，其他关联这个底层卷的PV会由于底层存储被删除导致使用出现异常。
  - 重复用底层存储时，数据一致性由您自行维护。建议在应用层做好多读多写的隔离保护，合理规划文件使用时间，避免出现多个客户端写同一个文件的情况，防止产生数据覆盖和丢失。

通过控制台使用已有对象存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 静态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“对象存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”来静态创建PVC。 - 无可用底层存储的场景下，可选择“动态创建”，具体操作请参见**通过动态存储卷使用对象存储**。 本文示例中选择“新建存储卷”，可通过控制台同时创建PV及PVC。
关联存储卷[a]	选择集群中已有的PV卷，需要提前创建PV，请参考**相关操作**中的“创建存储卷”操作。 本文示例中无需选择。
对象存储[b]	单击“选择对象存储”，您可以在新页面中勾选满足要求的对象存储，并单击“确定”。
PV名称[b]	输入PV名称，同一集群内的PV名称需唯一。
访问模式[b]	对象存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。
回收策略[b]	您可以选择Delete或Retain，用于指定删除PVC时底层存储的回收策略，详情请参见**PV回收策略**。 **说明** 多个PV使用同一个对象存储时建议使用**Retain**，避免级联删除底层卷。
访问密钥(AK/SK)[b]	自定义密钥：如果您需要为不同OBS存储分配不同的用户权限时，可通过选择不同的Secret实现更灵活的权限控制（推荐使用）。具体使用请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。 仅支持选择带有 secret.kubernetes.io/used-by = csi 标签的密钥，密钥类型为cfe/secure-opaque。如果无可用密钥，可单击“创建密钥”进行创建： - 名称：请输入密钥名称。 - 命名空间：密钥所在的命名空间。 - 访问密钥（AK/SK）：上传.csv格式的密钥文件，详情请参见**获取访问密钥**。
挂载参数[b]	输入挂载参数键值对，详情请参见**设置对象存储挂载参数**。

说明

a：创建方式选择“已有存储卷 PV”时可设置。

b：创建方式选择“新建存储卷 PV”时可设置。

2. 单击“创建”，将同时为您创建存储卷声明和存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

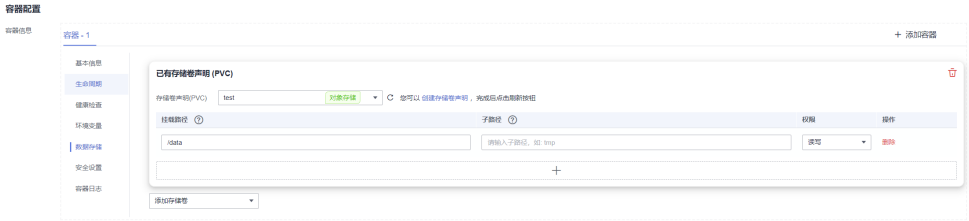
1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

   本文主要为您介绍存储卷的挂载使用，如**表8-37**，其他参数详情请参见**工作负载**。

表 8-37 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的对象存储卷。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	– 只读：只能读容器路径中的数据卷。 – 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到对象存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

通过 kubectl 命令行使用已有对象存储

步骤1 使用kubectl连接集群。

步骤2 创建PV。

1. 创建pv-obs.yaml文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除PV，保留底层存储卷
  name: pv-obs       # PV的名称
spec:
  accessModes:
  - ReadWriteMany    # 访问模式，对象存储必须为ReadWriteMany
  capacity:
    storage: 1Gi     # 对象存储容量大小
  csi:
    driver: obs.csi.everest.io        # 挂载依赖的存储驱动
    fsType: obsfs                     # 实例类型
    volumeHandle: <your_volume_id>    # 对象存储的名称
    volumeAttributes:
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/obs-volume-type: STANDARD
      everest.io/region: <your_region>                      # 对象存储的区域
      everest.io/enterprise-project-id: <your_project_id>     # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
    nodePublishSecretRef:           # 设置对象存储的自定义密钥
      name: <your_secret_name>       # 自定义密钥的名称
      namespace: <your_namespace>    # 自定义密钥的命名空间
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-obs             # 存储类名称
  mountOptions: []                      # 挂载参数
```

表 8-38 关键参数说明

参数	是否必填	描述
everest.io/reclaim-policy: retain-volume-only	否	可选字段 目前仅支持配置“retain-volume-only” everest插件版本需 >= 1.2.9且回收策略为Delete时生效。如果回收策略是Delete且当前值设置为“retain-volume-only”删除PVC回收逻辑为：删除PV，保留底层存储卷。
fsType	是	实例类型，支持“obsfs”与“s3fs”。 - obsfs：并行文件系统。 - s3fs：对象桶。
volumeHandle	是	对象存储的名称。

参数	是否必填	描述
everest.io/obs-volume-type	是	对象存储类型。 – fsType设置为s3fs时，支持STANDARD（标准桶）、WARM（低频访问桶）。 – fsType设置为obsfs时，该字段不起作用。
everest.io/region	是	OBS存储区域。 Region对应的值请参见**地区和终端节点**。
everest.io/enterprise-project-id	否	可选字段 对象存储的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在对象存储服务控制台，单击左侧栏目树中的“桶列表”或“并行文件系统”，单击要对接的对象存储名称进入详情页，在“概览 > 基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可获取对象存储所属的企业项目的ID。

nodePublishSecretRef	否	对象存储卷挂载支持设置自定义访问密钥（AK/SK），您可以使用AK/SK创建一个Secret，然后挂载到PV。详细说明请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。
		示例如下：
		`nodePublishSecretRef:`
		`  name: secret-demo`
		`  namespace: default`

mountOptions	否	挂载参数，具体请参见**设置对象存储挂载参数**。
persistentVolumeReclaimPolicy	是	集群版本号>=1.19.10且everest插件版本>=1.2.9时正式开放回收策略支持。 支持Delete、Retain回收策略，详情请参见**PV回收策略**。多个PV使用同一个对象存储时建议使用**Retain**，避免级联删除底层卷。 **Delete**: – Delete且不设置everest.io/reclaim-policy：删除PVC，PV资源与存储均被删除。 – Delete且设置everest.io/reclaim-policy=retain-volume-only：删除PVC，PV资源被删除，存储资源会保留。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。

参数	是否必填	描述
storage	是	存储容量，单位为Gi。 对对象存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	对象存储对应的存储类名称为csi-obs。

2. 执行以下命令，创建PV。
```
kubectl apply -f pv-obs.yaml
```

步骤3 创建PVC。

1. 创建pvc-obs.yaml文件。
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-obs
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: obsfs
    csi.storage.k8s.io/node-publish-secret-name: <your_secret_name>  # 自定义密钥的名称
    csi.storage.k8s.io/node-publish-secret-namespace: <your_namespace>       # 自定义密钥的命名空间
    everest.io/enterprise-project-id: <your_project_id>    # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
spec:
  accessModes:
  - ReadWriteMany                # 对象存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs      # 存储类名称，必须与PV的存储类一致。
  volumeName: pv-obs    # PV的名称
```

表 8-39 关键参数说明

参数	是否必填	描述
csi.storage.k8s.io/node-publish-secret-name	否	PV中指定的自定义密钥的名称。
csi.storage.k8s.io/node-publish-secret-namespace	否	PV中指定的自定义密钥的命名空间。
everest.io/enterprise-project-id	否	对象存储的项目ID。 获取方法：在对象存储服务控制台，单击左侧栏目树中的“桶列表”或“并行文件系统”，单击要对接的对象存储名称进入详情页，在“概览 > 基本信息”页签下找到企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可获取对象存储所属的企业项目的ID。

参数	是否必填	描述
storage	是	PVC申请容量，单位为Gi。 对于对象存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。
storageClassName	是	存储类名称，必须与1中PV的存储类一致。 对象存储对应的存储类名称为csi-obs。
volumeName	是	PV的名称，必须与1中PV名称一致。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-obs.yaml
```

步骤4 创建应用。

1. 创建web-demo.yaml文件，本示例中将对象存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-obs-volume    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-obs-volume    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-obs    #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载对象存储的应用。
```
kubectl apply -f web-demo.yaml
```
工作负载创建成功后，您可以尝试**验证数据持久化及共享性**。

----结束

验证数据持久化及共享性

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-mjhm9   1/1     Running   0          46s
web-demo-846b489584-wvv5s   1/1     Running   0          46s
```


2. 依次执行以下命令，查看Pod的/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
两个Pod均无返回结果，说明/data路径下无文件。

步骤2 执行以下命令，在/data路径下创建static文件。
```
kubectl exec web-demo-846b489584-mjhm9 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
```
预期输出如下：
```
static
```

步骤4 验证数据持久化

1. 执行以下命令，删除名称为web-demo-846b489584-mjhm9的Pod。
```
kubectl delete pod web-demo-846b489584-mjhm9
```
预期输出如下：
```
pod "web-demo-846b489584-mjhm9" deleted
```
删除后，Deployment控制器会自动重新创建一个副本。
2. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下，web-demo-846b489584-d4d4j为新建的Pod：
```
web-demo-846b489584-d4d4j  1/1    Running   0          110s
web-demo-846b489584-wvv5s  1/1    Running   0          7m50s
```
3. 执行以下命令，验证新建的Pod中/data路径下的文件是否更改。
```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```
预期输出如下：
```
static
```
static文件仍然存在，则说明数据可持久化保存。

步骤5 验证数据共享性

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-d4d4j  1/1    Running   0          7m
web-demo-846b489584-wvv5s  1/1    Running   0          13m
```
2. 执行以下命令，在任意一个Pod的/data路径下创建share文件。本例中选择名为web-demo-846b489584-d4d4j的Pod。
```
kubectl exec web-demo-846b489584-d4d4j --  touch /data/share
```
并查看该Pod中/data路径下的文件。
```
kubectl exec web-demo-846b489584-d4d4j -- ls /data
```
预期输出如下：
```
share
static
```
3. 由于写入share文件的操作未在名为web-demo-846b489584-wvv5s的Pod中执行，在该Pod中查看/data路径下是否存在文件以验证数据共享性。
```
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
预期输出如下：
```
share
static
```


如果在任意一个Pod中的/data路径下创建文件，其他Pod下的/data路径下均存在此文件，则说明两个Pod共享一个存储卷。

----结束

相关操作

您还可以执行表8-40中的操作。

表 8-40 其他操作

操作	说明	操作步骤
创建存储卷	通过CCE控制台单独创建PV。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷”页签。单击右上角“创建存储卷”，在弹出的窗口中填写存储卷声明参数。 ● 存储卷类型：选择“对象存储”。 ● 对象存储：单击“选择对象存储”，在新页面中勾选满足要求的对象存储，并单击“确定”。 ● PV名称：输入PV名称，同一集群内的PV名称需唯一。 ● 访问模式：仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。 ● 回收策略：Delete或Retain，详情请参见**PV回收策略**。 **说明** 多个PV使用同一个底层存储时建议使用**Retain**，避免级联删除底层卷。 ● 访问密钥(AK/SK)：自定义访问密钥如果您需要为不同OBS存储分配不同的用户权限时，可通过选择不同的Secret实现更灵活的权限控制（推荐使用）。具体使用请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。 仅支持选择带有 secret.kubernetes.io/used-by = csi 标签的密钥，密钥类型为 cfe/secure-opaque。如果无可用密钥，可单击“创建密钥”进行创建。 ● 挂载参数：输入挂载参数键值对，详情请参见**设置对象存储挂载参数**。 2. 单击“创建”。

操作	说明	操作步骤
更新访问密钥	通过CCE控制台更新对象存储的访问密钥。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 更新访问密钥”。 2. 上传.csv格式的密钥文件，详情请参见**获取访问密钥**。单击“确定”。 **说明** 更新全局访问密钥后，租户下所有挂载使用全局访问密钥的对象存储的负载实例需要重启后才能正常访问。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.6.3 通过动态存储卷使用对象存储

本文介绍如何自动创建对象存储，适用于无可用的底层存储卷，需要新创建的场景。

约束与限制

- 使用对象存储时，挂载点不支持修改属组和权限。
- 使用PVC挂载对象存储时，负载每挂载一个对象存储卷，后端会产生一个常驻进程。当负载使用对象存储数过多或大量读写对象存储文件时，常驻进程会占用大量内存，为保证负载稳定运行，建议负载使用的对象存储卷数量不超过其申请的内存GiB数量，如负载的申请的内存规格为4GiB，则建议其使用的对象存储数**不超过**4。
- 安全容器不支持使用对象存储。
- 挂载普通桶时不支持硬链接（Hard Link）。
- OBS限制单用户创建100个桶，当动态创建的PVC数量较多时，容易导致桶数量超过限制，OBS桶无法创建。此种场景下建议直接调用OBS的API或SDK使用OBS，不在工作负载中挂载OBS桶。

通过控制台自动创建对象存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 动态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“对象存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 无可用底层存储的场景下，可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。 - 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”，静态创建PVC，具体操作请参见**通过静态存储卷使用已有对象存储**。 本文中选择“动态创建”。
存储类	对象存储对应的默认存储类为csi-obs。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
实例类型	- 并行文件系统：一种对象存储服务提供的高性能文件系统，提供毫秒级别访问时延，以及TB/s级别带宽和百万级别的IOPS。 - 对象桶：OBS对象存储提供高可靠、高性能、高安全、低成本的数据存储能力，无文件数量限制、容量限制。
对象存储类型	选择“对象桶”时，支持选择以下类别： - 标准存储：适用于有大量热点文件或小文件，且需要频繁访问（平均一个月多次）并快速获取数据的业务场景。 - 低频访问存储：适用于不频繁访问（平均一年少于12次），但需要快速获取数据的业务场景。
数据冗余存储策略	集群中Everest版本要求2.4.14及以上。 - 多AZ存储：数据冗余存储至多个可用区（AZ），可靠性更高，但采用相对较高计费标准。关于计费详情请参见**价格计算器**。 - 单AZ存储：数据仅存储在单个可用区（AZ），成本更低。
访问模式	对象存储类型的存储卷仅支持ReadWriteMany，表示存储卷可以被多个节点以读写方式挂载，详情请参见**存储卷访问模式**。

参数	描述
访问密钥（AK/SK）	自定义密钥：如果您需要为不同OBS存储分配不同的用户权限时，可通过选择不同的Secret实现更灵活的权限控制（推荐使用）。具体使用请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。 仅支持选择带有 secret.kubernetes.io/used-by = csi 标签的密钥，密钥类型为cfe/secure-opaque。如果无可用密钥，可单击“创建密钥”进行创建： – 名称：请输入密钥名称。 – 命名空间：密钥所在的命名空间。 – 访问密钥（AK/SK）：上传.csv格式的密钥文件，详情请参见**获取访问密钥**。
企业项目	仅支持default、集群所在企业项目或存储类指定的企业项目。

2. 单击“创建”，将同时为您创建存储卷声明和存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明（PVC）”。

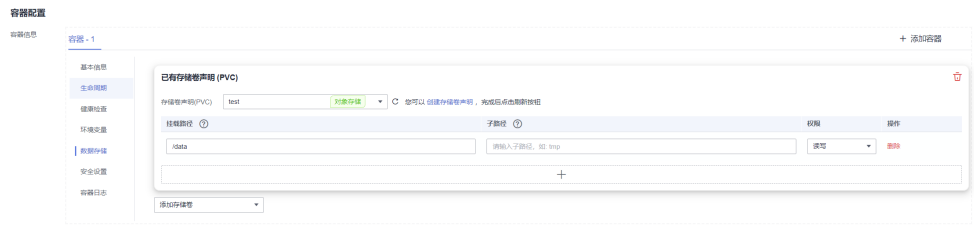
   本文主要为您介绍存储卷的挂载使用，如**表8-41**，其他参数详情请参见**工作负载**。

表 8-41 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的对象存储卷。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。

参数	参数说明
权限	– 只读：只能读容器路径中的数据卷。 – 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到对象存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化及共享性**中的步骤进行验证。

----结束

使用 kubectl 自动创建对象存储

步骤1 使用kubectl连接集群。

步骤2 使用StorageClass动态创建PVC及PV。

1. 创建pvc-obs-auto.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-obs-auto
  namespace: default
  annotations:
    everest.io/obs-volume-type: STANDARD    # 对象存储类型
    csi.storage.k8s.io/fstype: obsfs        # 实例类型
    csi.storage.k8s.io/node-publish-secret-name: <your_secret_name>       # 自定义密钥的名称
    csi.storage.k8s.io/node-publish-secret-namespace: <your_namespace>    # 自定义密钥的命名空间
    everest.io/enterprise-project-id: <your_project_id>     # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
    everest.io/csi.obs-az-redundancy: 'true'  # 可选字段，指定对象存储为多AZ存储
    everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
spec:
  accessModes:
    - ReadWriteMany              # 对象存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi               # 对象存储大小
  storageClassName: csi-obs      # StorageClass类型为对象存储
```


表 8-42 关键参数说明

参数	是否必选	描述
everest.io/obs-volume-type	是	对象存储类型。 – fsType设置为s3fs时，支持STANDARD（标准桶）、WARM（低频访问桶）。 – fsType设置为obsfs时，该字段不起作用。
csi.storage.k8s.io/fstype	是	实例类型，支持“obsfs”与“s3fs”。 – obsfs：并行文件系统。 – s3fs：对象桶。
csi.storage.k8s.io/node-publish-secret-name	否	自定义密钥的名称。 如果您需要为不同OBS存储分配不同的用户权限时，可通过选择不同的Secret实现更灵活的权限控制（推荐使用）。具体使用请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。
csi.storage.k8s.io/node-publish-secret-namespace	否	自定义密钥的命名空间。
everest.io/enterprise-project-id	否	对象存储的项目ID。 **获取方法**：在企业项目管理控制台，单击要对接的企业项目名称，复制企业项目ID值即可。
everest.io/csi.obs-az-redundancy	否	设置为true，表示将对象存储数据冗余存储至多个可用区（AZ），可靠性更高，但采用相对较高计费标准。关于计费详情请参见**价格计算器**。 集群中Everest版本要求2.4.14及以上。
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi。 对对象存储来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。

参数	是否必选	描述
storageClassName	是	存储类名称，对象存储对应的存储类名称为csi-obs。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-obs-auto.yaml
```

步骤3 创建应用。

1. 创建web-demo.yaml文件，本示例中将对象存储挂载至/data路径。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-demo
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-demo
  template:
    metadata:
      labels:
        app: web-demo
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-obs-volume     #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-obs-volume     #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-obs-auto     #已创建的PVC名称
```
2. 执行以下命令，创建一个挂载对象存储的应用。
```
kubectl apply -f web-demo.yaml
```
工作负载创建成功后，您可以尝试**验证数据持久化及共享性**。

----结束

验证数据持久化及共享性

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。
```
kubectl get pod | grep web-demo
```
预期输出如下：
```
web-demo-846b489584-mjhm9   1/1     Running   0          46s
web-demo-846b489584-wvv5s   1/1     Running   0          46s
```
2. 依次执行以下命令，查看Pod的/data路径下的文件。
```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
kubectl exec web-demo-846b489584-wvv5s -- ls /data
```
两个Pod均无返回结果，说明/data路径下无文件。

步骤2 执行以下命令，在/data路径下创建static文件。
```
kubectl exec web-demo-846b489584-mjhm9 --  touch /data/static
```


步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-demo-846b489584-mjhm9 -- ls /data
```

预期输出如下：

```
static
```

步骤4 验证数据持久化

1. 执行以下命令，删除名称为web-demo-846b489584-mjhm9的Pod。
   ```
   kubectl delete pod web-demo-846b489584-mjhm9
   ```
   预期输出如下：
   ```
   pod "web-demo-846b489584-mjhm9" deleted
   ```
   删除后，Deployment控制器会自动重新创建一个副本。
2. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep web-demo
   ```
   预期输出如下，web-demo-846b489584-d4d4j为新建的Pod：
   ```
   web-demo-846b489584-d4d4j   1/1     Running   0          110s
   web-demo-846b489584-wvv5s   1/1     Running   0          7m50s
   ```
3. 执行以下命令，验证新建的Pod中/data路径下的文件是否更改。
   ```
   kubectl exec web-demo-846b489584-d4d4j -- ls /data
   ```
   预期输出如下：
   ```
   static
   ```
   static文件仍然存在，则说明数据可持久化保存。

步骤5 验证数据共享性

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep web-demo
   ```
   预期输出如下：
   ```
   web-demo-846b489584-d4d4j   1/1     Running   0          7m
   web-demo-846b489584-wvv5s   1/1     Running   0          13m
   ```
2. 执行以下命令，在任意一个Pod的/data路径下创建share文件。本例中选择名为web-demo-846b489584-d4d4j的Pod。
   ```
   kubectl exec web-demo-846b489584-d4d4j --  touch /data/share
   ```
   并查看该Pod中/data路径下的文件。
   ```
   kubectl exec web-demo-846b489584-d4d4j -- ls /data
   ```
   预期输出如下：
   ```
   share
   static
   ```
3. 由于写入share文件的操作未在名为web-demo-846b489584-wvv5s的Pod中执行，在该Pod中查看/data路径下是否存在文件以验证数据共享性。
   ```
   kubectl exec web-demo-846b489584-wvv5s -- ls /data
   ```
   预期输出如下：
   ```
   share
   static
   ```
   如果在任意一个Pod中的/data路径下创建文件，其他Pod下的/data路径下均存在此文件，则说明两个Pod共享一个存储卷。

----结束

相关操作

您还可以执行**表8-43**中的操作。

表 8-43 其他操作

操作	说明	操作步骤
更新访问密钥	通过CCE控制台更新对象存储的访问密钥。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 更新访问密钥”。 2. 上传.csv格式的密钥文件，详情请参见**获取访问密钥**。单击“确定”。 **说明** 更新全局访问密钥后，租户下所有挂载使用全局访问密钥的对象存储的负载实例需要重启后才能正常访问。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.6.4 设置对象存储挂载参数

本章节主要介绍如何设置对象存储的挂载参数。您可以在PV中设置挂载参数，然后通过PVC绑定PV，也可以在StorageClass中设置挂载参数，然后使用StorageClass创建PVC，动态创建出的PV会默认带有StorageClass中设置的挂载参数。

前提条件

CCE容器存储（Everest）版本要求**1.2.8及以上**版本。插件主要负责将挂载参数识别并传递给底层存储，指定参数是否有效依赖于底层存储是否支持。

约束与限制

挂载参数暂不支持安全容器。

对象存储挂载参数

CCE的存储插件everest在挂载对象存储时默认设置了**表8-44**和**表8-45**的参数，其中**表8-44**中的参数不可取消。除了这些参数外，您还可以设置其他的对象存储挂载参数，具体请参见**挂载并行文件系统**。

表 8-44 默认使用且不可取消的挂载参数

参数	参数值	描述
use_ino	无需填写	使用该选项，由obsfs分配inode编号。读写模式下自动开启。
big_writes	无需填写	配置后可更改写缓存最大值大小
nonempty	无需填写	允许挂载目录非空
allow_other	无需填写	允许其他用户访问并行文件系统
no_check_certificate	无需填写	不校验服务端证书
enable_noobj_cache	无需填写	为不存在的对象启用缓存条目，可提高性能。对象桶读写模式下自动使用。 **从everest 1.2.40版本开始不再默认设置enable_noobj_cache参数。**
sigv2	无需填写	签名版本。对象桶自动使用。
public_bucket	1	设置为1时匿名挂载公共桶。对象桶只读模式下自动使用。

表 8-45 默认使用且可修改的挂载参数

参数	参数值	描述
max_write	131072	仅配置big_writes的情况下才生效，推荐使用128KB。
ssl_verify_hostname	0	不根据主机名验证SSL证书。
max_background	100	可配置后台最大等待请求数。并行文件系统自动使用。
umask	0	配置文件权限的掩码。 例如，如果umask值为022，而目录最大权限为777，则设置umask后该目录权限为777 - 022 = 755，即rwxr-xr-x。

在 PV 中设置挂载参数

在PV中设置挂载参数可以通过mountOptions字段实现，如下所示，mountOptions支持挂载的字段请参见**对象存储挂载参数**。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 在PV中设置挂载参数，示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only      # 可选字段，删除PV，保留底层存储卷
  name: pv-obs       # PV的名称
spec:
  accessModes:
  - ReadWriteMany    # 访问模式，对象存储必须为ReadWriteMany
  capacity:
    storage: 1Gi     # 对象存储容量大小
  csi:
    driver: obs.csi.everest.io        # 挂载依赖的存储驱动
    fsType: obsfs                     # 实例类型
    volumeHandle: <your_volume_id>    # 对象存储的名称
    volumeAttributes:
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/obs-volume-type: STANDARD
      everest.io/region: <your_region>                    # 对象存储的区域
      everest.io/enterprise-project-id: <your_project_id>     # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
    nodePublishSecretRef:           # 设置对象存储的自定义密钥
      name: <your_secret_name>       # 自定义密钥的名称
      namespace: <your_namespace>    # 自定义密钥的命名空间
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-obs              # 存储类名称
  mountOptions:                          # 挂载参数
  - umask=027
```

步骤3 PV创建后，可以创建PVC关联PV，然后在工作负载的容器中挂载，具体操作步骤请参见**通过静态存储卷使用已有对象存储**。

步骤4 验证挂载参数是否生效。

本例中将PVC挂载至使用nginx:latest镜像的工作负载，可以登录到运行挂载对象存储卷的Pod所在节点上通过进程详情观察。

执行以下命令：

- 对象桶：ps -ef | grep s3fs

  ```
  root     22142     1  0 Jun03 ?        00:00:00 /usr/bin/s3fs {your_obs_name} /mnt/paas/kubernetes/kubelet/pods/{pod_uid}/volumes/kubernetes.io~csi/{your_pv_name}/mount -o url=https://{endpoint}:443 -o endpoint={region} -o passwd_file=/opt/everest-host-connector/***_obstmpcred/{your_obs_name} -o nonempty -o big_writes -o sigv2 -o allow_other -o no_check_certificate -o ssl_verify_hostname=0 -o umask=027 -o max_write=131072 -o multipart_size=20
  ```

- 并行文件系统：ps -ef | grep obsfs

  ```
  root      1355     1  0 Jun03 ?        00:03:16 /usr/bin/obsfs {your_obs_name} /mnt/paas/kubernetes/kubelet/pods/{pod_uid}/volumes/kubernetes.io~csi/{your_pv_name}/mount -o url=https://{endpoint}:443 -o endpoint={region} -o passwd_file=/opt/everest-host-connector/***_obstmpcred/{your_obs_name} -o allow_other -o nonempty -o big_writes -o use_ino -o no_check_certificate -o ssl_verify_hostname=0 -o max_background=100 -o umask=027 -o max_write=131072
  ```

----结束

在 StorageClass 中设置挂载参数

在StorageClass中设置挂载参数同样可以通过mountOptions字段实现，如下所示，mountOptions支持挂载的字段请参见**对象存储挂载参数**。

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 创建自定义的StorageClass，示例如下：

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-obs-mount-option
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: obs.csi.everest.io
  csi.storage.k8s.io/fstype: s3fs
  everest.io/obs-volume-type: STANDARD
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:                          # 挂载参数
- umask=027
```

步骤3 StorageClass设置好后，就可以使用这个StorageClass创建PVC，动态创建出的PV会默认带有StorageClass中设置的挂载参数，具体操作步骤请参见**通过动态存储卷使用对象存储**。

步骤4 验证挂载参数是否生效。

本例中将PVC挂载至使用nginx:latest镜像的工作负载，可以登录到运行挂载对象存储卷的Pod所在节点上通过进程详情观察。

执行以下命令：

- 对象桶：ps -ef | grep s3fs

```
root     22142     1  0 Jun03 ?        00:00:00 /usr/bin/s3fs {your_obs_name} /mnt/paas/kubernetes/kubelet/pods/{pod_uid}/volumes/kubernetes.io~csi/{your_pv_name}/mount -o url=https://{endpoint}:443 -o endpoint={region} -o passwd_file=/opt/everest-host-connector/***_obstmpcred/{your_obs_name} -o nonempty -o big_writes -o sigv2 -o allow_other -o no_check_certificate -o ssl_verify_hostname=0 -o umask=027 -o max_write=131072 -o multipart_size=20
```

- 并行文件系统：ps -ef | grep obsfs

```
root     1355     1  0 Jun03 ?        00:03:16 /usr/bin/obsfs {your_obs_name} /mnt/paas/kubernetes/kubelet/pods/{pod_uid}/volumes/kubernetes.io~csi/{your_pv_name}/mount -o url=https://{endpoint}:443 -o endpoint={region} -o passwd_file=/opt/everest-host-connector/***_obstmpcred/{your_obs_name} -o allow_other -o nonempty -o big_writes -o use_ino -o no_check_certificate -o ssl_verify_hostname=0 -o max_background=100 -o umask=027 -o max_write=131072
```

----结束

8.6.5 对象存储卷挂载设置自定义访问密钥（AK/SK）

背景信息

CCE容器存储（Everest）在1.2.8及以上版本提供了设置自定义访问密钥的能力，这样可以让IAM用户使用自己的访问密钥挂载对象存储卷，从而可以对OBS进行访问权限控制（具体请参见**OBS不同权限控制方式的区别**）。

前提条件

- **CCE容器存储（Everest）**要求1.2.8及以上版本。
- 集群要求1.15.11及以上版本。

约束与限制

- 对象存储卷使用自定义访问密钥（AK/SK）时，对应的AK/SK不允许删除或禁用，否则业务容器将无法访问已挂载的对象存储。
- 自定义访问密钥暂不支持安全容器。

关闭自动挂载访问密钥

老版本控制台会要求您上传AK/SK，对象存储卷挂载时默认使用您上传的访问密钥，相当于所有IAM用户（即子用户）都使用的是同一个访问密钥挂载的对象桶，对桶的权限都是一样的，导致无法对IAM用户使用对象存储桶进行权限控制。

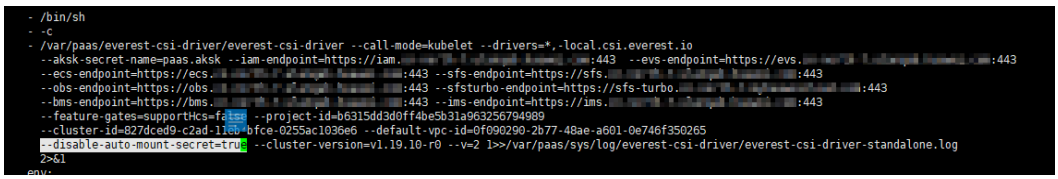
如果您之前上传过AK/SK，为防止IAM用户越权，建议关闭自动挂载访问密钥，即需要在everest插件中将**disable_auto_mount_secret**参数打开，这样使用对象存储时就不会自动使用在控制台上传的访问密钥。

说明

- 设置disable-auto-mount-secret时要求当前集群中无对象存储卷，否则挂载了该对象卷的工作负载扩容或重启的时候会由于必须指定访问密钥而导致挂卷失败。
- disable-auto-mount-secret设置为true后，则创建PV和PVC时必须指定挂载访问密钥，否则会导致对象卷挂载失败。

kubectl edit ds everest-csi-driver -nkube-system

搜索disable-auto-mount-secret，并将值设置为true。



执行 **:wq** 保存退出，等待实例重启完毕即可。

获取访问密钥

步骤1 登录控制台。

步骤2 鼠标指向界面右上角的登录用户名，在下拉列表中单击“我的凭证”。

步骤3 在左侧导航栏单击“访问密钥”。

步骤4 单击“新增访问密钥”，进入“新增访问密钥”页面。

步骤5 单击“确定”，下载访问密钥。

----结束

使用访问密钥创建 Secret

步骤1 获取访问密钥。

步骤2 对访问密钥进行base64编码（假设上文获取到的ak为“xxx”，sk为“yyy”）。

echo -n xxx|base64

echo -n yyy|base64

记录编码后的AK和SK。

步骤3 新建一个secret的yaml，如test-user.yaml。

```
apiVersion: v1
data:
  access.key: WE5WWVhVNU*****
  secret.key: Nnk4emJyZ0*****
kind: Secret
metadata:
  name: test-user
  namespace: default
  labels:
    secret.kubernetes.io/used-by: csi
type: cfe/secure-opaque
```

其中：

参数	描述
access.key	base64编码后的ak。
secret.key	base64编码后的sk。
name	secret的名称
namespace	secret的命名空间
secret.kubernetes.io/used-by: csi	带上这个标签才能在控制台上创建OBS PV/PVC时可见。
type	密钥类型，该值必须为cfe/secure-opaque 使用该类型，用户输入的数据会自动加密。

步骤4 创建Secret。

kubectl create -f test-user.yaml

----结束

静态创建对象存储卷时指定挂载 Secret

使用访问密钥创建Secret后，在创建PV时只需要关联上Secret，就可以使用Secret中的访问密钥（AK/SK）挂载对象存储卷。

步骤1 登录OBS控制台，创建对象存储桶，记录桶名称和存储类型，以并行文件系统为例。

步骤2 新建一个pv的yaml文件，如pv-example.yaml。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  csi:
    nodePublishSecretRef:
```



```
    name: test-user
    namespace: default
  driver: obs.csi.everest.io
  fsType: obsfs
  volumeAttributes:
    everest.io/obs-volume-type: STANDARD
    everest.io/region: ap-southeast-1
    storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  volumeHandle: obs-normal-static-pv
persistentVolumeReclaimPolicy: Delete
storageClassName: csi-obs
```

参数	描述
nodePublishSecretRef	挂载时指定的密钥，其中 ● name：指定secret的名字 ● namespace：指定secret的命令空间
fsType	文件类型，支持“obsfs”与“s3fs”，取值为s3fs时创建是obs对象桶；取值为obsfs时创建的是obs并行文件系统。
volumeHandle	对象存储的桶名称。

步骤3 创建PV。

kubectl create -f pv-example.yaml

PV创建完成后，就可以创建PVC关联PV。

步骤4 新建一个PVC的yaml文件，如pvc-example.yaml。

PVC yaml文件配置示例：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    csi.storage.k8s.io/node-publish-secret-name: test-user
    csi.storage.k8s.io/node-publish-secret-namespace: default
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: obsfs
  name: obs-secret
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
  volumeName: pv-obs-example
```

参数	描述
csi.storage.k8s.io/node-publish-secret-name	指定secret的名字
csi.storage.k8s.io/node-publish-secret-namespace	指定secret的命令空间

步骤5 创建PVC。

kubectl create -f pvc-example.yaml

PVC创建后，就可以创建工作负载挂载PVC使用存储。

----结束

动态创建对象存储卷时指定挂载密钥

动态创建对象存储卷时，可通过如下方法指定挂载密钥。

步骤1 新建一个pvc的yaml文件，如pvc-example.yaml。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    csi.storage.k8s.io/node-publish-secret-name: test-user
    csi.storage.k8s.io/node-publish-secret-namespace: default
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: obsfs
  name: obs-secret
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
```

参数	描述
csi.storage.k8s.io/node-publish-secret-name	指定secret的名字
csi.storage.k8s.io/node-publish-secret-namespace	指定secret的命令空间

步骤2 创建PVC。

kubectl create -f pvc-example.yaml

PVC创建后，就可以创建工作负载挂载PVC使用存储。

----结束

配置验证

根据上述步骤，使用IAM用户的密钥挂载对象存储卷。假设工作负载名称为obs-secret，容器内挂载目录是/temp，IAM用户权限为CCE ReadOnlyAccess和Tenant Guest。

1. 查询工作负载实例名称。

   kubectl get po | grep obs-secret

   期望输出：

   ```
   obs-secret-5cd558f76f-vxslv         1/1     Running   0          3m22s
   ```

2. 查询挂载目录下对象，查询正常。

kubectl exec obs-secret-5cd558f76f-vxslv -- ls -l /temp/

3. 尝试在挂载目录内写入数据，写入失败。

   kubectl exec obs-secret-5cd558f76f-vxslv -- touch /temp/test

   期望输出：

   ```
   touch: setting times of '/temp/test': No such file or directory
   command terminated with exit code 1
   ```

4. 参考桶策略配置，给挂载桶的子用户设置读写权限。

5. 再次尝试在挂载目录内写入数据，写入成功。

   kubectl exec obs-secret-5cd558f76f-vxslv -- touch /temp/test

6. 查看容器内挂载目录，验证数据写入成功。

   kubectl exec obs-secret-5cd558f76f-vxslv -- ls -l /temp/

   期望输出：

   ```
   -rwxrwxrwx 1 root root 0 Jun  7 01:52 test
   ```

8.6.6 跨区域使用 OBS 桶

默认情况下，Pod仅支持使用同一个区域（Region）的OBS桶。CCE支持工作负载使用其他区域的OBS桶，在某些场景下有利于提升OBS桶的资源利用率，但跨区域使用OBS相比同区域访问时延波动要更大。

约束与限制

- **CCE容器存储（Everest）**版本要求**1.2.42及以上**版本。
- 挂载存储的节点必须能够访问OBS桶，跨区域通常使用公网或专线打通。您可以在需要使用OBS的节点上Ping OBS的Endpoint来确定是否能够访问。
- 仅支持PV跨区域使用OBS桶，然后再使用PVC绑定PV，且PV回收策略必须为Retain。不支持使用StorageClass动态创建PVC跨区域使用OBS桶。

操作步骤

步骤1 创建名为paas-obs-endpoint的配置项，配置OBS所在区域和Endpoint。

配置项名称固定为paas-obs-endpoint，命名空间固定为kube-system。

区域名称和Endpoint以键值对形式对应，<region_name>和<endpoint_address>需替换为具体值，多个取值间使用逗号隔开。

Region对应的值请参见**地区和终端节点**。

例如{"ap-southeast-1": "https://obs.ap-southeast-1.myhuaweicloud.com:443", "ap-southeast-3": "https://obs.ap-southeast-3.myhuaweicloud.com:443"}

```
apiVersion: v1
kind: ConfigMap
metadata:
```



```
  name: paas-obs-endpoint   # 名称必须为paas-obs-endpoint
  namespace: kube-system    # 必须在kube-system命名空间下
data:
  obs-endpoint: |
    {"<region_name>": "<endpoint_address>"}
```

步骤2 创建PV。

如下所示，everest.io/region填入对应的OBS存储所在的region。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: testing-abc
  annotations:
    pv.kubernetes.io/bound-by-controller: 'yes'
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  capacity:
    storage: 1Gi
  csi:
    driver: obs.csi.everest.io
    volumeHandle: testing-abc            # OBS桶的名称
    fsType: s3fs                         # obsfs表示并行文件系统；s3fs表示对象桶
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: <region_name>       # OBS桶所在区域，需替换为具体取值
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    nodePublishSecretRef:                # 挂载OBS桶使用的AK/SK
      name: test-user
      namespace: default
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain   # PV回收策略必须为Retain
  storageClassName: csi-obs
  volumeMode: Filesystem
```

nodePublishSecretRef为对象存储卷挂载使用的访问密钥（AK/SK），您需要使用AK/SK创建一个Secret，在创建PV时使用。详细说明请参见**对象存储卷挂载设置自定义访问密钥（AK/SK）**。

步骤3 创建PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-test-abc
  namespace: default
  annotations:
    everest.io/obs-volume-type: STANDARD                # 桶类型，使用对象桶时支持标准（STANDARD）和低频（WARM）两种桶。
    csi.storage.k8s.io/fstype: s3fs                     # 文件类型，obsfs表示并行文件系统；s3fs表示对象桶
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany            # 对象存储必须为ReadWriteMany
  resources:
    requests:
      storage: 1Gi           # PVC申请容量大小，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi
  storageClassName: csi-obs   # StorageClass的名称，对象存储为csi-obs
  volumeName: testing-abc     # PV的名称
```

步骤4 创建工作负载，并在容器配置中的数据存储选项中选择存储卷声明PVC，添加上述创建的PVC，如果工作负载能够正常创建成功，则说明可以跨区域使用OBS桶。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: obs-deployment-example                  # 工作负载名称
  namespace: default
```



```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: obs-deployment-example
  template:
    metadata:
      labels:
        app: obs-deployment-example
    spec:
      containers:
      - image: nginx
        name: container-0
        volumeMounts:
        - mountPath: /tmp                    # 挂载路径
          name: pvc-obs-example
      restartPolicy: Always
      imagePullSecrets:
        - name: default-secret
      volumes:
      - name: pvc-obs-example
        persistentVolumeClaim:
          claimName: pvc-test-abc            # PVC名称
```

----结束

8.7 专属存储（DSS）

8.7.1 专属存储概述

专属分布式存储服务（Dedicated Distributed Storage Service，DSS）可以为您提供独享的物理存储资源，通过数据冗余和缓存加速等多项技术，提供高可用性和持久性，以及稳定的低时延性能。CCE支持将使用DSS创建的存储卷挂载到容器。

专属存储性能规格

存储池性能的主要指标有IO读写延时、IOPS和吞吐量。

- IOPS：每秒进行读写的操作次数。
- 吞吐量：每秒成功传送的数据量，即读取和写入的数据量。
- IO读写延时：连续两次进行读写操作所需的最小时间间隔。

表 8-46 专属存储性能规格

参数	高IO	超高IO
IOPS	1500 IOPS/TB	8000 IOPS/TB
IO读写时延（单队列，4KiB数据块大小）	1 ms ~ 3 ms	1 ms

参数	高IO	超高IO
典型应用场景	普通开发测试	• 转码类业务。 • I/O密集型场景，例如： – NoSQL – SQL Server – PostgreSQL • 时延敏感型场景，例如： – Redis – Memcache

关于专属存储性能的详细介绍，请以**存储池类型及性能介绍**为准。

使用场景

根据使用场景不同，专属存储支持以下挂载方式：

- **通过静态存储卷使用专属存储**：即静态创建的方式，需要先使用已有的磁盘创建PV，然后通过PVC在工作负载中挂载存储。适用于已有可用磁盘的场景。
- **通过动态存储卷使用专属存储**：即动态创建的方式，无需预先创建磁盘，在创建PVC时通过指定存储类（StorageClass），即可自动创建磁盘和对应的PV对象。适用于无可用的磁盘，需要新创建的场景。
- **在有状态负载中动态挂载专属存储**：仅有状态工作负载支持，可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。适用于多实例的有状态工作负载。

计费说明

- 您需要提前创建专属存储池资源并付费，在容器中挂载专属存储时将使用存储中已购买的资源。
- 关于专属存储的价格信息，请参见**计费说明**。

8.7.2 通过静态存储卷使用专属存储

CCE支持使用已有的专属存储创建存储卷（PersistentVolume）。创建成功后，通过创建相应的PersistentVolumeClaim绑定当前PersistentVolume使用。适用于已有底层存储的场景。

前提条件

- 您已经创建好一个集群，集群版本满足v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上，并且在该集群中安装2.4.5及以上版本的**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

- 您已经创建好一块专属盘，并且专属盘满足以下条件：
  - 已有的磁盘不可以是系统盘或共享盘。
  - 磁盘模式需选择SCSI（创建磁盘时默认为VBD模式）。
  - 磁盘的状态可用，且未被其他资源使用。
  - 若磁盘加密，所使用的密钥状态需可用。
  - 仅支持选择集群所属企业项目和default企业项目下的磁盘。

约束与限制

- 专属存储不支持跨可用区挂载，且不支持被多个工作负载、同一个工作负载的多个实例或多个任务使用。由于CCE集群各节点之间暂不支持共享盘的数据共享功能，多个节点挂载使用同一个磁盘可能会出现读写冲突、数据缓存冲突等问题，所以创建无状态工作负载时，若使用了专属存储，建议工作负载只选择一个实例。
- 如果使用HPA策略对挂载了专属存储的负载进行扩容，新Pod会因为无法挂载磁盘导致无法成功启动。

通过控制台使用已有专属存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 静态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“专属存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	- 已有底层存储的场景下，根据是否已经创建存储卷可选择“新建存储卷 PV”或“已有存储卷 PV”来静态创建PVC。 - 无可用底层存储的场景下，可选择“动态创建”，具体操作请参见**通过动态存储卷使用专属存储**。 本文示例中选择“新建存储卷”，可通过控制台同时创建PV及PVC。
关联存储卷[a]	选择集群中已有的PV卷，需要提前创建PV，请参考**相关操作**中的“创建存储卷”操作。 本文示例中无需选择。
专属存储[b]	单击“选择专属存储”，您可以在新页面中勾选满足要求的磁盘，并单击“确定”。
PV名称[b]	输入PV名称，同一集群内的PV名称需唯一。
访问模式[b]	专属存储类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。

参数	描述
回收策略[b]	您可以选择Delete或Retain，用于指定删除PVC时底层存储的回收策略，详情请参见**PV回收策略**。

说明

a：创建方式选择“已有存储卷 PV”时可设置。

b：创建方式选择“新建存储卷 PV”时可设置。

2. 单击“创建”，将同时为您创建存储卷声明及存储卷。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

   本文主要为您介绍存储卷的挂载使用，如**表8-47**，其他参数详情请参见**工作负载**。

表 8-47 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的专属存储卷。 专属存储卷无法被多个工作负载重复挂载。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到磁盘中。

说明

由于专属存储为非共享模式，工作负载下多个实例无法同时挂载，会导致实例启动异常。因此挂载专属存储盘时，工作负载实例数需为1。

如果您需要使用多实例的工作负载，请选择创建有状态工作负载，并使用动态挂载能力为每个实例挂载一个PV，详情请参考**在有状态负载中动态挂载专属存储**。

3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

通过 kubectl 命令行使用已有专属存储

步骤1 使用kubectl连接集群。

步骤2 创建PV。当您的集群中已存在创建完成的PV时，可跳过本步骤。

1. 创建pv-dss.yaml文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    everest.io/reclaim-policy: retain-volume-only        # 可选字段，删除PV时可保留底层存储卷
  name: pv-dss    # PV的名称
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>      # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
    - ReadWriteOnce     # 访问模式，专属存储必须为ReadWriteOnce
  capacity:
    storage: 10Gi       # 磁盘的容量，单位为Gi，取值范围 1-32768
  csi:
    driver: disk.csi.everest.io     # 挂载依赖的存储驱动
    fsType: ext4     # 与磁盘原文件系统保持一致
    volumeAttributes:
      everest.io/disk-mode: SCSI          # 磁盘模式，仅支持SCSI
      everest.io/disk-volume-type: SAS     # 磁盘的类型
      everest.io/csi.dedicated-storage-id: <dss_id>     # DSS存储池的ID
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
      everest.io/enterprise-project-id: <your_project_id>  # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
  persistentVolumeReclaimPolicy: Delete     # 回收策略
  storageClassName: csi-disk-dss             # 专属存储的存储类名称
```


表 8-48 关键参数说明

参数	是否必选	描述
everest.io/reclaim-policy: retain-volume-only	否	可选字段 目前仅支持配置“retain-volume-only” everest插件版本需 >= 1.2.9且回收策略为Delete时生效。如果回收策略是Delete且当前值设置为“retain-volume-only”删除PVC回收逻辑为：删除PV，保留底层存储卷。
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建专属存储所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
fsType	是	设置文件系统类型，默认为ext4。
everest.io/disk-volume-type	是	磁盘类型，全大写。 - SAS：高I/O - SSD：超高I/O
everest.io/csi.dedicated-storage-id	是	专属盘所在DSS存储池的ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 存储池”，单击要对接的存储池名称展开详情，复制ID值即可。
everest.io/crypt-key-id	否	当磁盘是加密卷时为必填，填写创建磁盘时选择的加密密钥ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 磁盘”，单击要对接的磁盘名称进入详情页，在“概览信息”页签下找到“配置信息”，复制密钥ID值即可。
everest.io/enterprise-project-id	否	可选字段 专属分布式存储的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 磁盘”，单击要对接的磁盘名称进入详情页，在“概览信息”页签下找到“管理信息”中的企业项目，单击并进入对应的企业项目控制台，复制对应的ID值即可获取专属分布式存储所属的企业项目的ID。

参数	是否必选	描述
persistentVolumeReclaimPolicy	是	支持Delete、Retain回收策略，详情请参见**PV回收策略**。如果数据安全性要求较高，建议使用**Retain**以免误删数据。 **Delete**: – Delete且不设置everest.io/reclaim-policy：删除PVC，PV资源与磁盘均被删除。 – Delete且设置everest.io/reclaim-policy=retain-volume-only：删除PVC，PV资源被删除，磁盘资源会保留。 **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。
storageClassName	是	专属存储对应的存储类名称为csi-disk-dss。

2. 执行以下命令，创建PV。

```
kubectl apply -f pv-dss.yaml
```

步骤3 创建PVC。

1. 创建pvc-dss.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-dss
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS    # 磁盘的类型
    everest.io/csi.dedicated-storage-id: <dss_id>    # DSS存储池的ID
    everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
    everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>      # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
  - ReadWriteOnce              # 专属存储必须为ReadWriteOnce
  resources:
    requests:
      storage: 10Gi           # 磁盘大小，取值范围 1-32768，必须和已有PV的storage大小保持一致。
  storageClassName: csi-disk-dss    # StorageClass类型为专属存储
  volumeName: pv-dss           # PV的名称
```

表 8-49 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。

参数	是否必选	描述
failure-domain.beta.kubernetes.io/zone	是	创建磁盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
everest.io/csi.dedicated-storage-id	是	专属盘所在DSS存储池的ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 存储池”，单击要对接的存储池名称展开详情，复制ID值即可。
storage	是	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
volumeName	是	PV的名称，必须与**1**中PV的名称一致。
storageClassName	是	存储类名称，必须与**1**中PV的存储类一致。 专属存储对应的存储类名称为csi-disk-dss。

2. 执行以下命令，创建PVC。

```
kubectl apply -f pvc-dss.yaml
```

步骤4 创建应用。

1. 创建web-dss.yaml文件，本示例中将磁盘挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web-dss
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-dss
  serviceName: web-dss   # Headless Service名称
  template:
    metadata:
      labels:
        app: web-dss
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-disk-dss    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-disk-dss    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-dss    #已创建的PVC名称
---
apiVersion: v1
kind: Service
metadata:
  name: web-dss   # Headless Service名称
  namespace: default
  labels:
    app: web-dss
```



```
spec:
  selector:
    app: web-dss
  clusterIP: None
  ports:
    - name: web-dss
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

2. 执行以下命令，创建一个挂载专属存储的应用。

```
kubectl apply -f web-dss.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及磁盘文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-dss
```

预期输出如下：

```
web-dss-0                1/1     Running   0             38s
```

2. 执行以下命令，查看磁盘是否挂载至/data路径。

```
kubectl exec web-dss-0 -- df | grep data
```

预期输出如下：

```
/dev/sdc            10255636     36888  10202364   0% /data
```

3. 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-dss-0 -- ls /data
```

预期输出如下：

```
lost+found
```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-dss-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-dss-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-dss-0的Pod。

```
kubectl delete pod web-dss-0
```

预期输出如下：

```
pod "web-dss-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec web-dss-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```


static文件仍然存在，则说明磁盘中的数据可持久化保存。

----结束

相关操作

您还可以执行**表8-50**中的操作。

表 8-50 其他操作

操作	说明	操作步骤
创建存储卷	通过CCE控制台单独创建PV。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷”页签。单击右上角“创建存储卷PV”，在弹出的窗口中填写存储卷声明参数。 ● 存储卷类型：选择“专属存储”。 ● 专属存储：单击“选择专属存储”，在新页面中勾选满足要求的磁盘，并单击“确定”。 ● PV名称：输入PV名称，同一集群内的PV名称需唯一。 ● 访问模式：仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。 ● 回收策略：Delete或Retain，详情请参见**PV回收策略**。 2. 单击“创建”。
扩容专属存储卷	通过CCE控制台快速扩容已挂载的专属存储。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.7.3 通过动态存储卷使用专属存储

CCE支持指定存储类（StorageClass），自动创建专属存储类型的底层存储和对应的存储卷，适用于无可用的底层存储，需要新创建的场景。

前提条件

- 您已经创建好一个集群，集群版本满足v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上，并且在该集群中安装2.4.5及以上版本的**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

约束与限制

- 专属存储不支持跨可用区挂载，且不支持被多个工作负载、同一个工作负载的多个实例或多个任务使用。由于CCE集群各节点之间暂不支持共享盘的数据共享功能，多个节点挂载使用同一个磁盘可能会出现读写冲突、数据缓存冲突等问题，所以创建无状态工作负载时，若使用了专属存储，建议工作负载只选择一个实例。
- 如果使用HPA策略对挂载了专属存储的负载进行扩容，新Pod会因为无法挂载磁盘导致无法成功启动。
- 动态创建专属存储卷时支持添加资源标签，且专属存储创建完成后无法在CCE侧更新资源标签，需要前往专属存储控制台更新。如果使用已有的专属存储创建存储卷，也需要在专属存储控制台添加或更新资源标签。

通过控制台自动创建专属存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 动态创建存储卷声明和存储卷。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“专属存储”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	– 无可用底层存储的场景下，可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。 – 已有底层存储的场景下，根据是否已经创建PV可选择“新建存储卷”或“已有存储卷”，静态创建PVC，具体操作请参见**通过静态存储卷使用专属存储**。 本文中选择“动态创建”。
存储类	专属存储对应的默认存储类为csi-disk-dss。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。

参数	描述
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
专属实例	选择一个已有的专属存储实例。
容量（GiB）	申请的存储卷容量大小。
访问模式	专属存储类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
加密	选择底层存储是否加密，使用加密时需要选择使用的加密密钥。
企业项目	仅支持default、集群所在企业项目或存储类指定的企业项目。
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。集群中everest版本为2.1.39及以上时支持。 您可以在TMS中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动创建“CCE-Cluster-ID=<集群ID>”、“CCE-Cluster-Name=<集群名称>”、“CCE-Namespace=<命名空间名称>”的系统标签，您无法自定义修改。 **说明** 专属存储类型的动态存储卷创建完成后，不支持在CCE侧更新资源标签。如需更新专属存储的资源标签，请前往专属存储控制台。

2. 单击“创建”。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的存储卷声明和存储卷。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

   本文主要为您介绍存储卷的挂载使用，如**表8-51**，其他参数详情请参见**工作负载**。

表 8-51 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的专属存储卷。 专属存储卷无法被多个工作负载重复挂载。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到磁盘中。

说明

由于专属存储为非共享模式，工作负载下多个实例无法同时挂载，会导致实例启动异常。因此挂载专属存储盘时，工作负载实例数需为1。

如果您需要使用多实例的工作负载，请选择创建有状态工作负载，并使用动态挂载能力为每个实例挂载一个PV，详情请参考**在有状态负载中动态挂载专属存储**。

3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

使用 kubectl 自动创建专属存储

步骤1 使用kubectl连接集群。

步骤2 使用StorageClass动态创建PVC及PV。

1. 创建pvc-dss-auto.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-dss-auto
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS    # 磁盘的类型
```



```
    everest.io/csi.dedicated-storage-id: <dss_id>     # DSS存储池的ID
    everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
    everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
    everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
  labels:
    failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
    failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
spec:
  accessModes:
  - ReadWriteOnce              # 专属存储必须为ReadWriteOnce
  resources:
    requests:
      storage: 10Gi            # 磁盘大小，取值范围 1-32768
  storageClassName: csi-disk-dss    # StorageClass类型为专属存储
```

表 8-52 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建磁盘所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
everest.io/disk-volume-type	是	磁盘类型，全大写。 - SAS：高I/O - SSD：超高I/O
everest.io/csi.dedicated-storage-id	是	专属盘所在DSS存储池的ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 存储池”，单击要对接的存储池名称展开详情，复制ID值即可。
everest.io/crypt-key-id	否	当专属存储是加密卷时为必填，填写创建磁盘时选择的加密密钥ID。 **获取方法**：在数据加密控制台，找到需要加密的密钥，复制密钥ID值即可。
everest.io/enterprise-project-id	否	可选字段 专属存储的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在企业项目管理控制台，单击要对接的企业项目名称，复制企业项目ID值即可。

参数	是否必选	描述
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi，取值范围为1-32768。
storageClassName	是	专属存储对应的存储类名称为csi-disk-dss。

2. 执行以下命令，创建PVC。
```
kubectl apply -f pvc-dss-auto.yaml
```

步骤3 创建应用。

1. 创建web-dss-auto.yaml文件，本示例中将磁盘挂载至/data路径。
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web-dss-auto
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-dss-auto
  serviceName: web-dss-auto   # Headless Service名称
  template:
    metadata:
      labels:
        app: web-dss-auto
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: pvc-disk-dss    #卷名称，需与volumes字段中的卷名称对应
          mountPath: /data  #存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: pvc-disk-dss    #卷名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-dss-auto    #已创建的PVC名称
---
apiVersion: v1
kind: Service
metadata:
  name: web-dss-auto   # Headless Service名称
```



```
  namespace: default
  labels:
    app: web-dss-auto
spec:
  selector:
    app: web-dss-auto
  clusterIP: None
  ports:
    - name: web-dss-auto
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

2. 执行以下命令，创建一个挂载专属存储的应用。

```
kubectl apply -f web-dss-auto.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及磁盘文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-dss-auto
```

预期输出如下：

```
web-dss-auto-0                1/1     Running   0             38s
```

2. 执行以下命令，查看磁盘是否挂载至/data路径。

```
kubectl exec web-dss-auto-0 -- df | grep data
```

预期输出如下：

```
/dev/sdc            10255636     36888  10202364   0% /data
```

3. 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-dss-auto-0 -- ls /data
```

预期输出如下：

```
lost+found
```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-dss-auto-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-dss-auto-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-dss-auto-0的Pod。

```
kubectl delete pod web-dss-auto-0
```

预期输出如下：

```
pod "web-dss-auto-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec web-dss-auto-0 -- ls /data
```


预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明磁盘中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-53中的操作。

表 8-53 其他操作

操作	说明	操作步骤
扩容存储卷	通过CCE控制台快速扩容已挂载的专属存储。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.7.4 在有状态负载中动态挂载专属存储

使用场景

动态挂载仅可在创建**有状态负载（StatefulSet）**时使用，通过卷声明模板（**volumeClaimTemplates**字段）实现，并依赖于StorageClass的动态创建PV能力。在多实例的有状态负载中，动态挂载可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。而在无状态工作负载的普通挂载方式中，当存储支持多点挂载（ReadWriteMany）时，工作负载下的多个Pod会被挂载到同一个底层存储中。

说明

Kubernetes不允许在更新StatefulSet时添加或删除volumeClaimTemplates字段，您只能在创建StatefulSet时设置volumeClaimTemplates。

前提条件

- 您已经创建好一个集群，集群版本满足v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上，并且在该集群中安装2.4.5及以上版本的**CCE容器存储（Everest）**。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。

通过控制台动态挂载专属存储

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。

步骤3 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 动态挂载 (VolumeClaimTemplate)”。

步骤4 单击“创建存储卷声明 PVC”，在弹出窗口中填写存储卷声明参数。

参数填写完成后，单击“创建”。

参数	描述
存储卷声明类型	本文中选择“专属存储”。
PVC名称	输入PVC的名称。创建后将根据实例数自动增加后缀，格式为<自定义PVC名称>-<序号>，例如example-0。
创建方式	可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。
存储类	专属存储对应的默认存储类为csi-disk-dss。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
专属实例	选择一个已有的专属存储实例。
容量（GiB）	申请的存储卷容量大小。
访问模式	专属存储类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
加密	选择底层存储是否加密，使用加密时需要选择使用的加密密钥。
企业项目	仅支持default、集群所在企业项目或存储类指定的企业项目。

参数	描述
资源标签	通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。集群中everest版本为2.1.39及以上时支持。 您可以在资源标签管理服务中创建“预定义标签”，预定义标签对所有支持标签功能的服务资源可见，通过使用预定义标签可以提升标签创建和迁移效率。具体请参见**创建预定义标签**。 CCE服务会自动创建“CCE-Cluster-ID=<集群ID>”、“CCE-Cluster-Name=<集群名称>”、“CCE-Namespace=<命名空间名称>”的系统标签，您无法自定义修改。 **说明** 专属存储类型的动态存储卷创建完成后，不支持在CCE侧更新资源标签。如需更新专属存储的资源标签，请前往专属存储控制台。

步骤5 填写挂载路径。

表 8-54 存储卷挂载

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将磁盘挂载到容器中/data路径下，在该路径下生成的容器数据会存储到磁盘中。

步骤6 本文主要为您介绍存储卷的动态挂载使用，其他参数详情请参见**创建有状态负载（StatefulSet）**。其余信息都配置完成后，单击“创建工作负载”。

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

通过 kubectl 命令行动态挂载专属存储

步骤1 使用kubectl连接集群。

步骤2 创建statefulset-dss.yaml文件，本示例中将磁盘挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-dss
  namespace: default
spec:
  selector:
    matchLabels:
      app: statefulset-dss
  template:
    metadata:
      labels:
        app: statefulset-dss
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - name: pvc-disk          # 需与volumeClaimTemplates字段中的名称对应
              mountPath: /data        # 存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
  serviceName: statefulset-dss        # Headless Service名称
  replicas: 2
  volumeClaimTemplates:
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: pvc-disk
        namespace: default
        annotations:
          everest.io/disk-volume-type: SAS    # 磁盘的类型
          everest.io/csi.dedicated-storage-id: <dss_id>     # DSS存储池的ID
          everest.io/crypt-key-id: <your_key_id>    # 可选字段，加密密钥ID，使用加密盘的时候填写
          everest.io/enterprise-project-id: <your_project_id> # 可选字段，企业项目ID，如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。
          everest.io/disk-volume-tags: '{"key1":"value1","key2":"value2"}' # 可选字段，用户自定义资源标签
          everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
        labels:
          failure-domain.beta.kubernetes.io/region: <your_region>   # 替换为您待部署应用的节点所在的区域
          failure-domain.beta.kubernetes.io/zone: <your_zone>       # 替换为您待部署应用的节点所在的可用区
      spec:
        accessModes:
          - ReadWriteOnce              # 专属存储必须为ReadWriteOnce
        resources:
          requests:
            storage: 10Gi              # 磁盘大小，取值范围 1-32768
        storageClassName: csi-disk-dss      # StorageClass类型为专属存储
---
apiVersion: v1
kind: Service
metadata:
  name: statefulset-dss   # Headless Service名称
  namespace: default
  labels:
    app: statefulset-dss
spec:
  selector:
    app: statefulset-dss
  clusterIP: None
  ports:
    - name: statefulset-dss
      targetPort: 80
```



```
    nodePort: 0
    port: 80
    protocol: TCP
type: ClusterIP
```

表 8-55 关键参数说明

参数	是否必选	描述
failure-domain.beta.kubernetes.io/region	是	集群所在的region。 Region对应的值请参见**地区和终端节点**。
failure-domain.beta.kubernetes.io/zone	是	创建专属存储所在的可用区，必须和工作负载规划的可用区保持一致。 zone对应的值请参见**地区和终端节点**。
everest.io/disk-volume-type	是	磁盘类型，全大写。 ● SAS：高I/O ● SSD：超高I/O
everest.io/csi.dedicated-storage-id	是	专属盘所在DSS存储池的ID。 **获取方法**：在云服务器控制台，单击左侧栏目树中的“专属分布式存储 > 存储池”，单击要对接的存储池名称展开详情，复制ID值即可。
everest.io/crypt-key-id	否	当专属存储是加密卷时为必填，填写创建磁盘时选择的加密密钥ID。 **获取方法**：在数据加密控制台，找到需要加密的密钥，复制密钥ID值即可。
everest.io/enterprise-project-id	否	可选字段 专属存储的企业项目ID。如果指定企业项目，则创建PVC时也需要指定相同的企业项目，否则PVC无法绑定PV。 **获取方法**：在企业项目管理控制台，单击要对接的企业项目名称，复制企业项目ID值即可。
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi，取值范围为1-32768。

参数	是否必选	描述
storageClassName	是	专属存储对应的存储类名称为csi-disk-dss。

步骤3 执行以下命令，创建一个挂载专属存储的应用。

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

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及磁盘文件。

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep statefulset-dss
   ```
   预期输出如下：
   ```
   statefulset-dss-0        1/1     Running   0          45s
   statefulset-dss-1        1/1     Running   0          28s
   ```
2. 执行以下命令，查看磁盘是否挂载至/data路径。
   ```
   kubectl exec statefulset-dss-0 -- df | grep data
   ```
   预期输出如下：
   ```
   /dev/sdd            10255636     36888  10202364   0% /data
   ```
3. 执行以下命令，查看/data路径下的文件。
   ```
   kubectl exec statefulset-dss-0 -- ls /data
   ```
   预期输出如下：
   ```
   lost+found
   ```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec statefulset-dss-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec statefulset-dss-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-dss-auto-0的Pod。

```
kubectl delete pod statefulset-dss-0
```

预期输出如下：

```
pod "statefulset-dss-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec statefulset-dss-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```


static文件仍然存在，则说明磁盘中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-56中的操作。

表 8-56 其他操作

操作	说明	操作步骤
扩容存储卷	通过CCE控制台快速扩容已挂载的专属存储。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击PVC操作列的“更多 > 扩容”。 2. 输入新增容量，并单击“确定”。
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.8 本地持久卷（Local PV）

8.8.1 本地持久卷概述

本地持久卷介绍

CCE支持使用LVM将节点上的数据卷组成存储池（VolumeGroup），然后划分LV给容器挂载使用。使用本地持久卷作为存储介质的PV的类型可称之为Local PV。

与HostPath卷相比，本地持久卷能够以持久和可移植的方式使用，而且本地持久卷的PV会存在节点亲和性配置，其挂载的Pod会自动根据该亲和性配置进行调度，无需手动将Pod调度到特定节点。

挂载方式

本地持久卷仅支持以下挂载方式：

- **通过动态存储卷使用本地持久卷**：即动态创建的方式，在创建PVC时通过指定存储类（StorageClass），即可自动创建对象存储和对应的PV对象。
- **在有状态负载中动态挂载本地持久卷**：仅有状态工作负载支持，可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。适用于多实例的有状态工作负载。

说明

本地持久卷不支持通过静态PV使用，即不支持先手动创建PV然后通过PVC在工作负载中挂载的方式使用。

约束与限制

- 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。
- **移除节点**、**删除节点**、**重置节点**和**缩容节点**会导致与节点关联的本地持久存储卷类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。移除节点、删除节点、重置节点和缩容节点时使用了本地持久存储卷的Pod会从待删除、重置的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。节点重置完成后，Pod可能调度到重置好的节点上，此时Pod会一直处于creating状态，因为该PVC对应的底层逻辑卷已不存在。
- 请勿在节点上手动删除对应的存储池或卸载数据盘，否则会导致数据丢失等异常情况。
- 本地持久卷为非共享模式，不支持被多个工作负载或者多个任务同时挂载，且不支持被工作负载下多个实例同时挂载。

8.8.2 在存储池中导入持久卷

CCE支持使用LVM将节点上的数据卷组成存储池（VolumeGroup），然后划分LV给容器挂载使用。在创建本地持久卷前，需将节点数据盘导入存储池。

约束与限制

- 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。
- 节点上的第一块数据盘（供容器运行时和Kubelet组件使用）不支持导入为存储池。
- 条带化模式的存储池不支持扩容，条带化存储池扩容后可能造成碎片空间，无法使用。
- 存储池不支持缩容和删除。
- 如果删除节点上存储池的磁盘，会导致存储池异常。

导入存储池

创建节点时导入

在创建节点时，在存储配置中可以为节点添加数据盘，选择“作为持久存储卷”导入存储池，详情请参见**创建节点**。

手动导入

如果创建节点时没有导入持久存储卷，或当前存储卷容量不够，可以进行手动导入。

步骤1 前往ECS控制台为节点添加SCSI类型的磁盘。操作步骤详情请参见**新增磁盘**。

步骤2 登录CCE控制台，单击集群名称进入集群。

步骤3 在左侧导航栏中选择“存储”，并切换至“存储池”页签。

步骤4 查看已添加磁盘的节点，选择“导入持久卷”，导入时可以选择写入模式。

说明

如存储池列表中未找到手动挂载的磁盘，请耐心等待1分钟后刷新列表。

- 线性：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。
- 条带化：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。多块卷才能选择条带化。

----结束

扩容存储池

存储池扩容可采用两种方式：

1. 通过以上手动导入的方式新增大容量磁盘。
2. 前往ECS界面扩容已导入的云硬盘，然后等待存储池自动扩容。

以上两种方式均可达到存储池扩容的目的。

8.8.3 通过动态存储卷使用本地持久卷

前提条件

- 您已经创建好一个集群，并且在该集群中安装CSI插件（**everest**）。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经将一块节点数据盘导入本地持久卷存储池，详情请参见**在存储池中导入持久卷**。

约束与限制

- 本地持久卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本>=2.1.23，推荐使用>=2.1.23版本。
- **移除节点**、**删除节点**、**重置节点**和**缩容节点**会导致与节点关联的本地持久存储卷类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。移除节点、删除节点、重置节点和缩容节点时使用了本地持久存储卷的Pod会从待删除、重置的节点上驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。节点重置完成后，Pod可能调度到重置好的节点上，此时Pod会一直处于creating状态，因为该PVC对应的底层逻辑卷已不存在。
- 请勿在节点上手动删除对应的存储池或卸载数据盘，否则会导致数据丢失等异常情况。
- 本地持久卷为非共享模式，不支持被多个工作负载或者多个任务同时挂载，且不支持被工作负载下多个实例同时挂载。

通过控制台自动创建本地持久卷

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 动态创建PVC及PV。

1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”页签。单击右上角“创建存储卷声明 PVC”，在弹出的窗口中填写存储卷声明参数。

参数	描述
存储卷声明类型	本文中选择“本地持久卷”。
PVC名称	输入PVC的名称，同一命名空间下的PVC名称需唯一。
创建方式	仅可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。
存储类	本地持久卷对应的默认存储类为csi-local-topology。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。

参数	描述
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
容量	申请的存储卷容量大小，支持GiB和MiB。 **说明** 由于本地持久卷使用LVM实现，LVM基本单位逻辑区域（Logical Extent，LE）为4MiB，当PVC中申请的大小不为4MiB的整数倍时，实际申请的LVM逻辑卷大小将会向上取整为4MiB的整数倍。例如申请401MiB PVC，其实际LVM逻辑卷占用为404MiB（占用101个LE），最终在页面中看到的使用量为LVM实际使用量。
访问模式	本地持久卷类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
存储池	查看已导入的存储池，如需将新的数据卷导入存储池，请参见**在存储池中导入持久卷**。

2. 单击“创建”，将同时为您创建PVC和PV。

   您可以在左侧导航栏中选择“存储”，在“存储卷声明”和“存储卷”页签下查看已经创建的PVC和PV。

   说明

   本地存储卷存储类（名为csi-local-topology）的卷绑定模式为延迟绑定（即volumeBindingMode参数值为WaitForFirstConsumer）。该模式会延迟PV的创建和绑定，只有在创建工作负载时声明使用该PVC，对应的PV才会创建并绑定。

步骤3 创建应用。

1. 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。
2. 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 已有存储卷声明 (PVC)”。

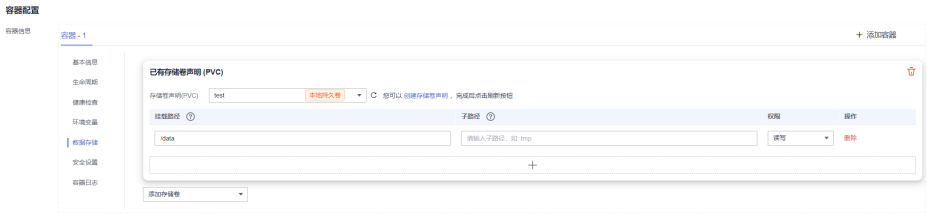
   本文主要为您介绍存储卷的挂载使用，如**表8-57**，其他参数详情请参见**工作负载**。

表 8-57 存储卷挂载

参数	参数说明
存储卷声明（PVC）	选择已有的本地持久卷。 本地持久卷无法被多个工作负载重复挂载。

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	- 只读：只能读容器路径中的数据卷。 - 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到本地持久存储中。



3. 其余信息都配置完成后，单击“创建工作负载”。

   工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

使用 kubectl 自动创建本地持久卷

步骤1 使用kubectl连接集群。

步骤2 使用StorageClass动态创建PVC及PV。

1. 创建pvc-local.yaml文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-local
  namespace: default
  annotations:
    everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
spec:
  accessModes:
    - ReadWriteOnce             # 本地持久卷必须为ReadWriteOnce
  resources:
```



```
  requests:
    storage: 10Gi          # 本地持久卷大小
storageClassName: csi-local-topology    # StorageClass类型为本地持久卷
```

表 8-58 关键参数说明

参数	是否必选	描述
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi和Mi。 **说明** 由于本地持久卷使用LVM实现，LVM基本单位逻辑区域（Logical Extent，LE）为4MiB，当PVC中申请的大小不为4MiB的整数倍时，实际申请的LVM逻辑卷大小将会向上取整为4MiB的整数倍。例如申请401MiB PVC，其实际LVM逻辑卷占用为404MiB（占用101个LE），最终在页面中看到的使用量为LVM实际使用量。
storageClassName	是	存储类名称，本地持久卷对应的存储类名称为csi-local-topology。

2. 执行以下命令，创建PVC。

```
kubectl apply -f pvc-local.yaml
```

步骤3 创建应用。

1. 创建web-local.yaml文件，本示例中将本地持久卷挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web-local
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-local
  serviceName: web-local   # Headless Service名称
  template:
    metadata:
      labels:
        app: web-local
    spec:
      containers:
```



```
    - name: container-1
      image: nginx:latest
      volumeMounts:
      - name: pvc-disk    #卷名称，需与volumes字段中的卷名称对应
        mountPath: /data  #存储卷挂载的位置
    imagePullSecrets:
      - name: default-secret
    volumes:
      - name: pvc-disk    #卷名称，可自定义
        persistentVolumeClaim:
          claimName: pvc-local    #已创建的PVC名称
---
apiVersion: v1
kind: Service
metadata:
  name: web-local   # Headless Service名称
  namespace: default
  labels:
    app: web-local
spec:
  selector:
    app: web-local
  clusterIP: None
  ports:
    - name: web-local
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```

2. 执行以下命令，创建一个挂载本地持久存储的应用。

```
kubectl apply -f web-local.yaml
```

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

验证数据持久化

步骤1 查看部署的应用及本地文件。

1. 执行以下命令，查看已创建的Pod。

```
kubectl get pod | grep web-local
```

预期输出如下：

```
web-local-0                1/1     Running   0             38s
```

2. 执行以下命令，查看本地持久卷是否挂载至/data路径。

```
kubectl exec web-local-0 -- df | grep data
```

预期输出如下：

```
/dev/mapper/vg--everest--localvolume--persistent-pvc-local       10255636     36888  10202364   0% /data
```

3. 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-local-0 -- ls /data
```

预期输出如下：

```
lost+found
```

步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec web-local-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec web-local-0 -- ls /data
```

预期输出如下：


```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-local-0的Pod。

```
kubectl delete pod web-local-0
```

预期输出如下：

```
pod "web-local-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec web-local-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明本地持久存储中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-59中的操作。

表 8-59 其他操作

操作	说明	操作步骤
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.8.4 在有状态负载中动态挂载本地持久卷

使用场景

动态挂载仅可在创建**有状态负载（StatefulSet）**时使用，通过卷声明模板（**volumeClaimTemplates**字段）实现，并依赖于StorageClass的动态创建PV能力。在多实例的有状态负载中，动态挂载可以为每一个Pod关联一个独有的PVC及PV，当Pod被重新调度后，仍然能够根据该PVC名称挂载原有的数据。而在无状态工作负载的普通挂载方式中，当存储支持多点挂载（ReadWriteMany）时，工作负载下的多个Pod会被挂载到同一个底层存储中。

说明

Kubernetes不允许在更新StatefulSet时添加或删除volumeClaimTemplates字段，您只能在创建StatefulSet时设置volumeClaimTemplates。

前提条件

- 您已经创建好一个集群，并且在该集群中安装CSI插件（**everest**）。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 您已经将一块节点数据盘导入本地持久卷存储池，详情请参见**在存储池中导入持久卷**。

通过控制台动态挂载本地持久卷

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右侧选择“有状态负载”页签。

步骤3 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 动态挂载 (VolumeClaimTemplate)”。

步骤4 单击“创建存储卷声明 PVC”，在弹出窗口中填写卷声明模板参数。

参数填写完成后，单击“创建”。

参数	描述
存储卷声明类型	本文中选择“本地持久卷”。
PVC名称	输入PVC的名称。创建后将根据实例数自动增加后缀，格式为<自定义PVC名称>-<序号>，例如example-0。
创建方式	仅可选择“动态创建”，通过控制台级联创建存储卷声明PVC、存储卷PV和底层存储。
存储类	本地持久卷对应的默认存储类为csi-local-topology。 您可以自建存储类并配置回收策略和绑定模式，具体操作请参见**通过控制台创建StorageClass**。
存储卷名称前缀（可选）	集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
容量	申请的存储卷容量大小，支持GiB和MiB。 **说明** 由于本地持久卷使用LVM实现，LVM基本单位逻辑区域（Logical Extent，LE）为4MiB，当PVC中申请的大小不为4MiB的整数倍时，实际申请的LVM逻辑卷大小将会向上取整为4MiB的整数倍。例如申请401MiB PVC，其实际LVM逻辑卷占用为404MiB（占用101个LE），最终在页面中看到的使用量为LVM实际使用量。

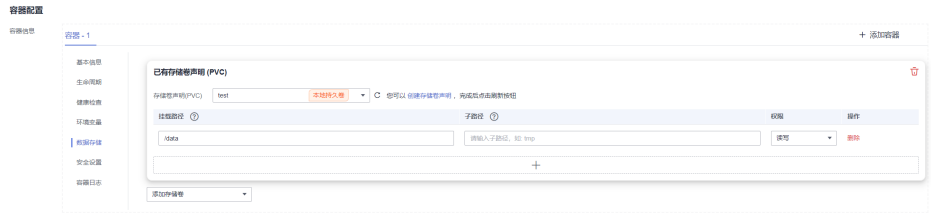
参数	描述
访问模式	本地持久卷类型的存储卷仅支持ReadWriteOnce，表示存储卷可以被一个节点以读写方式挂载，详情请参见**存储卷访问模式**。
存储池	查看已导入的存储池，如需将新的数据卷导入存储池，请参见**在存储池中导入持久卷**。

步骤5 填写挂载路径。

表 8-60 存储卷挂载

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

本例中将该存储卷挂载到容器中/data路径下，在该路径下生成的容器数据会存储到本地持久卷中。



步骤6 本文主要为您介绍存储卷的动态挂载使用，其他参数详情请参见**创建有状态负载（StatefulSet）**。其余信息都配置完成后，单击“创建工作负载”。

工作负载创建成功后，容器挂载目录下的数据将会持久化保持，您可以参考**验证数据持久化**中的步骤进行验证。

----结束

通过 kubectl 命令行动态挂载本地持久卷

步骤1 使用kubectl连接集群。

步骤2 创建statefulset-local.yaml文件，本示例中将本地持久卷挂载至/data路径。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-local
  namespace: default
spec:
  selector:
    matchLabels:
      app: statefulset-local
  template:
    metadata:
      labels:
        app: statefulset-local
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - name: pvc-local          # 需与volumeClaimTemplates字段中的名称对应
              mountPath: /data         # 存储卷挂载的位置
      imagePullSecrets:
        - name: default-secret
  serviceName: statefulset-local       # Headless Service名称
  replicas: 2
  volumeClaimTemplates:
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: pvc-local
        namespace: default
        annotations:
          everest.io/csi.volume-name-prefix: test  # 可选字段，定义自动创建的底层存储名称前缀
      spec:
        accessModes:
          - ReadWriteOnce              # 本地持久卷必须为ReadWriteOnce
        resources:
          requests:
            storage: 10Gi              # 存储卷容量大小
        storageClassName: csi-local-topology      # StorageClass类型为本地持久卷
---
apiVersion: v1
kind: Service
metadata:
  name: statefulset-local   # Headless Service名称
  namespace: default
  labels:
    app: statefulset-local
spec:
  selector:
    app: statefulset-local
  clusterIP: None
  ports:
    - name: statefulset-local
      targetPort: 80
      nodePort: 0
      port: 80
      protocol: TCP
  type: ClusterIP
```


表 8-61 关键参数说明

参数	是否必选	描述
everest.io/csi.volume-name-prefix	否	可选字段，集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持，且集群中需安装2.4.15及以上版本的Everest插件。 定义自动创建的底层存储名称，实际创建的底层存储名称为“存储卷名称前缀”与“PVC UID”的拼接组合，如果不填写该参数，默认前缀为“pvc”。 取值范围：参数值长度为1~26，且必须是小写字母、数字、中划线，不能以中划线开头或结尾。 例如，存储卷名称前缀设置为“test”，则实际创建的底层存储名称test-{uid}。
storage	是	PVC申请容量，单位为Gi和Mi。 **说明** 由于本地持久卷使用LVM实现，LVM基本单位逻辑区域（Logical Extent，LE）为4MiB，当PVC中申请的大小不为4MiB的整数倍时，实际申请的LVM逻辑卷大小将会向上取整为4MiB的整数倍。例如申请401MiB PVC，其实际LVM逻辑卷占用为404MiB（占用101个LE），最终在页面中看到的使用量为LVM实际使用量。
storageClassName	是	本地持久卷对应的存储类名称为csi-local-topology。

步骤3 执行以下命令，创建一个挂载本地持久卷存储的应用。

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

工作负载创建成功后，您可以尝试**验证数据持久化**。

----结束

验证数据持久化

步骤1 查看部署的应用及文件。

1. 执行以下命令，查看已创建的Pod。

   ```
   kubectl get pod | grep statefulset-local
   ```

   预期输出如下：

   ```
   statefulset-local-0         1/1     Running   0          45s
   statefulset-local-1         1/1     Running   0          28s
   ```

2. 执行以下命令，查看本地持久卷是否挂载至/data路径。

   ```
   kubectl exec statefulset-local-0 -- df | grep data
   ```

   预期输出如下：

   ```
   /dev/mapper/vg--everest--localvolume--persistent-pvc-local  10255636     36888  10202364   0% /data
   ```

3. 执行以下命令，查看/data路径下的文件。

   ```
   kubectl exec statefulset-local-0 -- ls /data
   ```

   预期输出如下：

   ```
   lost+found
   ```


步骤2 执行以下命令，在/data路径下创建static文件。

```
kubectl exec statefulset-local-0 --  touch /data/static
```

步骤3 执行以下命令，查看/data路径下的文件。

```
kubectl exec statefulset-local-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

步骤4 执行以下命令，删除名称为web-local-auto-0的Pod。

```
kubectl delete pod statefulset-local-0
```

预期输出如下：

```
pod "statefulset-local-0" deleted
```

步骤5 删除后，StatefulSet控制器会自动重新创建一个同名副本。执行以下命令，验证/data路径下的文件是否更改。

```
kubectl exec statefulset-local-0 -- ls /data
```

预期输出如下：

```
lost+found
static
```

static文件仍然存在，则说明本地持久卷中的数据可持久化保存。

----结束

相关操作

您还可以执行表8-62中的操作。

表 8-62 其他操作

操作	说明	操作步骤
事件	查看PVC或PV的事件名称、事件类型、发生次数、Kubernetes事件、首次和最近发生的时间，便于定位问题。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“事件”，即可查看1小时内的事件（事件保存时间为1小时）。
查看YAML	可对PVC或PV的YAML文件进行查看、复制和下载。	1. 在左侧导航栏选择“存储”，在右侧选择“存储卷声明”或“存储卷”页签。 2. 单击目标实例操作列的“查看YAML”，即可查看或下载YAML。

8.9 临时存储卷（EmptyDir）

8.9.1 临时存储卷概述

临时卷介绍

当有些应用程序需要额外的存储，但并不关心数据在重启后是否仍然可用。 例如，缓存服务经常受限于内存大小，而且可以将不常用的数据转移到比内存慢的存储中，对总体性能的影响并不大。另有些应用程序需要以文件形式注入的只读数据，比如配置数据或密钥。

Kubernetes中的**临时卷**（Ephemeral Volume），就是为此类场景设计的。临时卷会遵从Pod的生命周期，与 Pod一起创建和删除。

Kubernetes中常用的临时卷：

- **EmptyDir**：Pod启动时为空，存储空间来自本地的kubelet根目录（通常是根磁盘）或内存。EmptyDir是从**节点临时存储**中分配的，如果来自其他来源（如日志文件或镜像分层数据）的数据占满了临时存储，可能会发生存储容量不足的问题。
- **ConfigMap**：将ConfigMap类型的Kubernetes数据以数据卷的形式挂载到Pod中。
- **Secret**：将Secret类型的Kubernetes数据以数据卷的形式挂载到Pod中。

EmptyDir 的类型

CCE提供了如下两种EmptyDir类型：

- **临时路径**：Kubernetes原生的EmptyDir类型，生命周期与容器实例相同，并支持指定内存作为存储介质。容器实例消亡时，EmptyDir会被删除，数据会永久丢失。
- **本地临时卷**：本地临时存储卷将节点的本地数据盘通过LVM组成**存储池**（VolumeGroup），然后划分LV作为EmptyDir的存储介质给容器挂载使用，相比原生EmptyDir默认的存储介质类型性能更好。

约束与限制

- 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本 >=1.2.29。
- 请勿在节点上手动删除对应的存储池或卸载数据盘，否则会导致数据丢失等异常情况。
- 请确保节点上Pod不要挂载/var/lib/kubelet/pods/目录，否则可能会导致使用了临时存储卷的Pod无法正常删除。

8.9.2 在存储池中导入临时卷

CCE支持使用LVM将节点上的数据卷组成存储池（VolumeGroup），然后划分LV给容器挂载使用。在创建本地临时卷前，需将节点数据盘导入存储池。

约束与限制

- 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本 >=1.2.29。
- 节点上的第一块数据盘（供容器运行时和Kubelet组件使用）不支持导入为存储池。

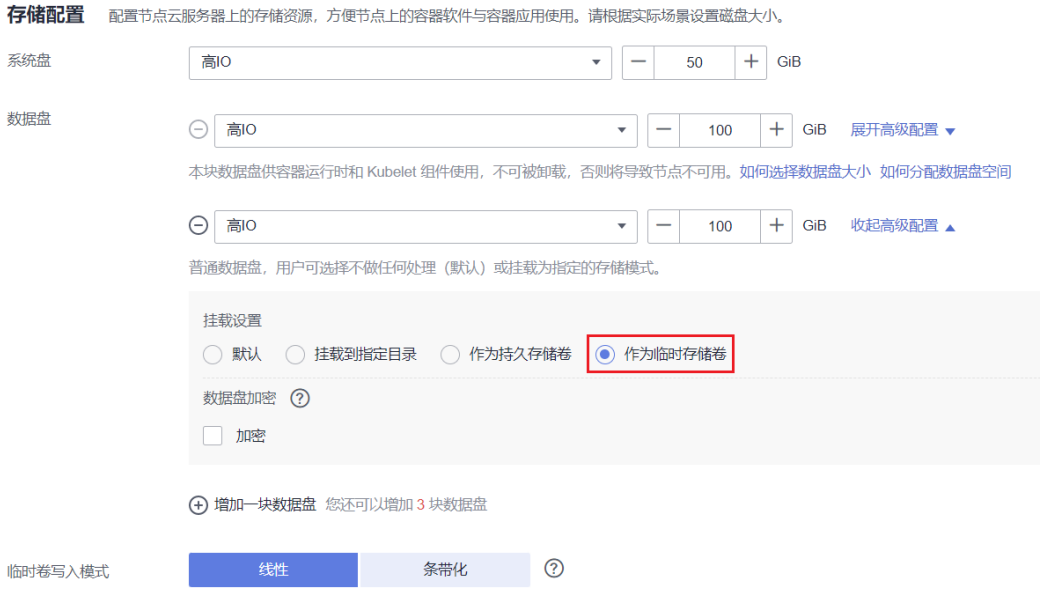
- 条带化模式的存储池不支持扩容，条带化存储池扩容后可能造成碎片空间，无法使用。
- 存储池不支持缩容和删除。
- 如果删除节点上存储池的磁盘，会导致存储池异常。

导入存储池

创建节点时导入

在创建节点时，在存储配置中可以为节点添加数据盘，选择“作为临时存储卷”导入存储池，详情请参见**创建节点**。

图 8-3 导入临时卷



手动导入

如果创建节点时没有导入临时存储卷，或当前存储卷容量不够，可以进行手动导入。

步骤1 前往ECS控制台为节点添加SCSI类型的磁盘。操作步骤详情请参见**新增磁盘**。

步骤2 登录CCE控制台，单击集群名称进入集群。

步骤3 在左侧导航栏中选择“存储”，并切换至“存储池”页签。

步骤4 查看已添加磁盘的节点，选择“导入临时卷”，导入时可以选择写入模式。

说明

如存储池列表中未找到手动挂载的磁盘，请耐心等待1分钟后刷新列表。

- 线性：线性逻辑卷是将一个或多个物理卷整合为一个逻辑卷，实际写入数据时会先往一个基本物理卷上写入，当存储空间占满时再往另一个基本物理卷写入。
- 条带化：创建逻辑卷时指定条带化，当实际写入数据时会将连续数据分成大小相同的块，然后依次存储在多个物理卷上，实现数据的并发读写从而提高读写性能。多块卷才能选择条带化。

----结束

扩容存储池

存储池扩容可采用两种方式

1. 通过以上手动导入的方式新增大容量磁盘。
2. 前往ECS界面扩容已导入的云硬盘，然后等待存储池自动扩容。

以上两种方式均可达到存储池扩容的目的。

8.9.3 使用本地临时卷

本地临时卷（Local Ephemeral Volume）存储在临时卷**存储池**，相比原生EmptyDir默认的存储介质类型性能要更好，且支持扩容。

前提条件

- 您已经创建好一个集群，并且在该集群中安装CSI插件（**everest**）。
- 如果您需要通过命令行创建，需要使用kubectl连接到集群，详情请参见**通过kubectl连接集群**。
- 如需使用本地临时卷，您需要将一块节点数据盘导入本地临时卷存储池，详情请参见**在存储池中导入临时卷**。

约束与限制

- 本地临时卷仅在集群版本 >= v1.21.2-r0 时支持，且需要everest插件版本 >=1.2.29。
- 请勿在节点上手动删除对应的存储池或卸载数据盘，否则会导致数据丢失等异常情况。
- 请确保节点上Pod不要挂载/var/lib/kubelet/pods/目录，否则可能会导致使用了临时存储卷的Pod无法正常删除。

通过控制台使用本地临时卷

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。

步骤3 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 本地临时卷(EmptyDir)”。

步骤4 本文主要为您介绍存储卷的挂载使用，如**表8-63**，其他参数详情请参见**工作负载**。

表 8-63 本地临时卷挂载

参数	参数说明
容量	申请的存储卷容量大小。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

步骤5 其余工作负载参数都配置完成后，单击“创建工作负载”。

----结束

通过 kubectl 使用本地临时卷

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-emptydir.yaml文件。

vi nginx-emptydir.yaml

YAML文件内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-emptydir
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-emptydir
  template:
    metadata:
      labels:
        app: nginx-emptydir
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - name: vol-emptydir          # 卷名称，需与volumes字段中的卷名称对应
              mountPath: /tmp             # emptyDir挂载路径
      imagePullSecrets:
```



```
    - name: default-secret
  volumes:
    - name: vol-emptydir            # 卷名称，可自定义
      emptyDir:
        medium: LocalVolume         # emptyDir磁盘介质设置为LocalVolume，表示使用本地临时卷
        sizeLimit: 1Gi              # 卷容量大小
```

步骤3 创建工作负载。

kubectl apply -f nginx-emptydir.yaml

----结束

本地临时卷异常处理说明

用户如果手动从ECS侧卸盘、手动执行vgremove两种误操作致临时卷存储池异常。可以先将节点设置为不可调度，具体方法请参见**一键设置节点调度策略**，然后通过重置节点进行恢复。

8.9.4 使用临时路径

临时路径是Kubernetes原生的EmptyDir类型，生命周期与容器实例相同，并支持指定内存作为存储介质。容器实例消亡时，EmptyDir会被删除，数据会永久丢失。

通过控制台使用临时路径

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“工作负载”，在右侧选择“无状态负载”页签。

步骤3 单击页面右上角“创建工作负载”，在“容器配置”中选择“数据存储”页签，并单击“添加存储卷 > 临时路径(EmptyDir)”。

步骤4 本文主要为您介绍存储卷的挂载使用，如**表8-64**，其他参数详情请参见**工作负载**。

表 8-64 临时路径挂载

参数	参数说明
存储介质	开启内存： • 开启后可以使用内存提高运行速度，但存储容量受内存大小限制。适用于数据量少，读写效率要求高的场景。 • 未开启时默认存储在硬盘上，适用于数据量大，读写效率要求低的场景。 **说明** • 开启内存后请注意内存大小，如果存储容量超出内存大小会发生OOM事件。 • 使用内存时的EmptyDir的大小为Pod规格限制值的100%。 • 不使用内存的EmptyDir不会占用系统内存。

参数	参数说明
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

步骤5 其余工作负载参数都配置完成后，单击“创建工作负载”。

----结束

通过 kubectl 使用临时路径

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-emptydir.yaml文件。

vi nginx-emptydir.yaml

YAML文件内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-emptydir
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-emptydir
  template:
    metadata:
      labels:
        app: nginx-emptydir
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - name: vol-emptydir     # 卷名称，需与volumes字段中的卷名称对应
              mountPath: /tmp        # emptyDir挂载路径
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: vol-emptydir         # 卷名称，可自定义
          emptyDir:
```



```
      medium: Memory        # emptyDir磁盘介质：设置为Memory时，表示开启内存；设置为空时为原生默认的存储介质类型
      sizeLimit: 1Gi        # 卷容量大小
```

步骤3 创建工作负载。

kubectl apply -f nginx-emptydir.yaml

----结束

8.10 主机路径（HostPath）

主机路径（HostPath）可以将容器所在宿主机的文件目录挂载到容器指定的挂载点中，如容器需要访问/etc/hosts则可以使用HostPath映射/etc/hosts等场景。

须知

- HostPath卷存在许多安全风险，最佳做法是尽可能避免使用HostPath。 当必须使用HostPath卷时，它的范围应仅限于所需的文件或目录，并以只读方式挂载。
- 当挂载HostPath卷的Pod删除后，HostPath中的数据依然会保留。

通过控制台使用主机路径

主机路径(HostPath)挂载表示将主机上的路径挂载到指定的容器路径。通常用于：“容器工作负载程序生成的日志文件需要永久保存”或者“需要访问宿主机上Docker引擎内部数据结构的容器工作负载”。

步骤1 登录CCE控制台。

步骤2 在创建工作负载时，在“容器配置”中找到“数据存储”，选择“主机路径(HostPath)”。

步骤3 设置添加本地磁盘参数，如表8-65。

表 8-65 卷类型选择主机路径挂载

参数	参数说明
存储类型	主机路径(HostPath)。

参数	参数说明
主机路径	输入主机路径，如/etc/hosts。 **说明** 请注意“主机路径”不能设置为根目录“/”，否则将导致挂载失败。挂载路径一般设置为： • /opt/xxxx（但不能为/opt/cloud） • /mnt/xxxx（但不能为/mnt/paas） • /tmp/xxx • /var/xxx（但不能为/var/lib、/var/script、/var/paas等关键目录） • /xxxx（但不能和系统目录冲突，例如bin、lib、home、root、boot、dev、etc、lost+found、mnt、proc、sbin、srv、tmp、var、media、opt、selinux、sys、usr等） 注意不能设置为/home/paas、/var/paas、/var/lib、/var/script、/mnt/paas、/opt/cloud，否则会导致系统或节点安装失败。
挂载路径	请输入挂载路径，如：/tmp。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入存储卷的子路径，将存储卷中的某个路径挂载至容器，可以实现在单一Pod中使用同一个存储卷的不同文件夹。如：tmp，表示容器中挂载路径下的数据会存储在存储卷的tmp文件夹中。不填写时默认为根路径。
权限	• 只读：只能读容器路径中的数据卷。 • 读写：可修改容器路径中的数据卷，容器迁移时新写入的数据不会随之迁移，会造成数据丢失。

步骤4 其余信息都配置完成后，单击“创建工作负载”。

----结束

使用 kubectl 使用主机路径

步骤1 使用kubectl连接集群。

步骤2 创建并编辑nginx-hostpath.yaml文件。

vi nginx-hostpath.yaml

YAML文件内容如下，将节点上的/data目录挂载至容器中的/data目录下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-hostpath
  namespace: default
spec:
  replicas: 2
```



```
selector:
  matchLabels:
    app: nginx-hostpath
template:
  metadata:
    labels:
      app: nginx-hostpath
  spec:
    containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
          - name: vol-hostpath        # 卷名称，需与volumes字段中的卷名称对应
            mountPath: /data          # 容器中的挂载路径
    imagePullSecrets:
      - name: default-secret
    volumes:
      - name: vol-hostpath            # 卷名称，可自定义
        hostPath:
          path: /data                 # 宿主机节点上的目录位置
```

步骤3 创建工作负载。

kubectl apply -f nginx-hostpath.yaml

----结束

8.11 存储类（StorageClass）

存储类介绍

在Kubernetes中，StorageClass是一种资源对象，描述了集群中的存储类型“分类”，用于定义存储卷的配置模板。每个StorageClass对象都定义了一种存储方式，包括动态卷供应的配置参数，如卷的类型、访问模式、卷的生命周期策略等，在创建PVC/PV均需要指定StorageClass。

当您在创建一个PVC时，您只需要指定StorageClassName，Kubernetes即可根据这个StorageClass自动创建PV及底层存储，大大减少了手动创建并维护PV的工作量。

除了使用CCE提供的**默认存储类**外，您也可以根据需求自定义存储类，可参考**自定义存储类应用场景**。

通过控制台创建 StorageClass

说明

集群中的CCE容器存储（Everest）插件需要处于运行中状态。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“存储”，在右侧选择“存储类”页签。单击右上角“创建存储类”，在弹出的窗口中填写存储类参数。

参数	描述
存储类类型	选择底层存储类型。
名称	输入存储类的名称，同一集群的存储类名称需唯一。

参数	描述
回收策略	您可以选择Delete或Retain，用于指定删除PVC时底层存储的回收策略，详情请参见**PV回收策略**。 • **Delete**：存储卷声明PVC删除时，会将关联的底层存储资源删除，并同步移除PV资源，请谨慎使用。 • **Retain**：存储卷声明PVC删除时，PV和关联的底层存储资源均会保留，其中PV状态被设置为已释放，继续手动删除PV不会删除底层存储资源，若希望该PV还能被PVC绑定，需去除PV上与原PVC绑定的相关信息。
绑定模式	动态创建PV的时间，分为立即创建和延迟创建。 • **Immediate**：PVC创建后，会立即创建底层存储资源及存储卷PV，并与PVC绑定。本地持久卷类型不支持设置为Immediate。 • **WaitForFirstConsumer**：PVC创建后，不会立即与存储卷PV绑定，而是等需要挂载该PVC的Pod被调度后，再创建底层存储资源及存储卷PV，并与PVC绑定。对象存储、文件存储、极速文件存储类型不支持设置为WaitForFirstConsumer。

步骤3 单击“创建”。您可以在“存储类”页签下查看已经创建的存储类及相关信息。

----结束

通过 YAML 创建 StorageClass

目前CCE默认提供csi-disk、csi-nas、csi-obs等StorageClass，在声明PVC时使用对应StorageClassName，就可以自动创建对应类型PV，并自动创建底层的存储资源。

执行如下kubectl命令即可查询CCE提供的默认StorageClass。您可以使用CCE提供的CSI插件自定义创建StorageClass。

```
# kubectl get sc
NAME                PROVISIONER                  AGE
csi-disk            everest-csi-provisioner      17d       # 云硬盘
csi-disk-topology   everest-csi-provisioner      17d       # 延迟创建的云硬盘
csi-nas             everest-csi-provisioner      17d       # 文件存储 1.0
csi-sfs             everest-csi-provisioner      17d       # 通用文件存储（SFS 3.0）
csi-obs             everest-csi-provisioner      17d       # 对象存储
csi-sfsturbo        everest-csi-provisioner      17d       # 极速文件存储
csi-local           everest-csi-provisioner      17d       # 本地持久卷
csi-local-topology  everest-csi-provisioner      17d       # 延迟创建的本地持久卷
```

每个StorageClass都包含了动态制备PersistentVolume时会使用到的默认参数。如以下云硬盘存储类的示例：

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-disk
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/passthrough: 'true'
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate
```


表 8-66 关键参数说明

参数	描述
provisioner	存储资源提供商，CCE均由everest插件提供，此处只能填写everest-csi-provisioner。
parameters	存储参数，不同类型的存储支持的参数不同。详情请参见**表8-67**。
reclaimPolicy	用来指定创建PV的persistentVolumeReclaimPolicy字段值，支持Delete和Retain。如果StorageClass 对象被创建时没有指定reclaimPolicy，它将默认为Delete。 • **Delete**：存储卷声明PVC删除时，会将关联的底层存储资源删除，并同步移除PV资源，请谨慎使用。 • **Retain**：存储卷声明PVC删除时，PV和关联的底层存储资源均会保留，其中PV状态被设置为已释放，继续手动删除PV不会删除底层存储资源，若希望该PV还能被PVC绑定，需去除PV上与原PVC绑定的相关信息。
allowVolumeExpansion	定义由此存储类创建的PV是否支持动态扩容，默认为false。是否能动态扩容是由底层存储插件来实现的，这里只是一个开关。
volumeBindingMode	表示卷绑定模式，即动态创建PV的时间，分为立即创建和延迟创建。 • **Immediate**：PVC创建后，会立即创建底层存储资源及存储卷PV，并与PVC绑定。 • **WaitForFirstConsumer**：PVC创建后，不会立即与存储卷PV绑定，而是等需要挂载该PVC的Pod被调度后，再创建底层存储资源及存储卷PV，并与PVC绑定。
mountOptions	该字段需要底层存储支持，如果不支持挂载选项，却指定了挂载选项，会导致创建PV操作失败。

表 8-67 parameters 参数说明

存储类型	参数	是否必选	描述
云硬盘	csi.storage.k8s.io/csi-driver-name	是	驱动类型，使用云硬盘类型时，参数取值固定为“disk.csi.everest.io”。
	csi.storage.k8s.io/fstype	是	使用云硬盘时，支持的参数值为“ext4”。

存储类型	参数	是否必选	描述
	everest.io/disk-volume-type	是	云硬盘类型，全大写。 ● SAS：高I/O ● SSD：超高I/O ● GPSSD：通用型SSD ● ESSD：极速型SSD ● GPSSD2：通用型SSD v2，Everest版本为2.4.4及以上支持使用，使用时需同时指定everest.io/disk-iops和everest.io/disk-throughput注解。
	everest.io/passthrough	是	参数取值固定为“true”，表示云硬盘的设备类型为SCSI类型。不允许设置为其他值。
	everest.io/disk-iops	否	IOPS值由用户预配置，仅使用通用型SSD v2和极速型SSD v2类型的云硬盘支持设置。 ● 通用型SSD v2类型的IOPS范围为3000~128000，最高可配置 500*容量（GiB）。 选择通用型SSD V2，当IOPS大于3000时会收取额外IOPS费用，详情请参见**价格计算器**。 ● 极速型SSD v2类型的IOPS范围为100~256000，最高可配置 1000*容量（GiB）。 选择极速型SSD V2，使用IOPS会收取额外IOPS费用，详情请参见**价格计算器**。
	everest.io/disk-throughput	否	吞吐量由用户预配置，仅使用通用型SSD v2类型的云硬盘支持设置。 范围为125~1000MiB/s，最高可配置IOPS/4。 吞吐量大于125MiB/s时会收取额外吞吐量费用，详情请参见**价格计算器**。
文件存储	csi.storage.k8s.io/csi-driver-name	是	驱动类型，使用文件存储类型时，参数取值固定为“nas.csi.everest.io”。
	csi.storage.k8s.io/fstype	是	使用文件存储时，支持的参数值为“nfs”。
	everest.io/share-access-level	是	参数取值固定为“rw”，表示文件存储可读写。
	everest.io/share-access-to	是	集群所在VPC ID。

存储类型	参数	是否必选	描述
	everest.io/share-is-public	否	参数取值固定为“false”，表示文件共享为私人可见。 使用通用文件存储（SFS 3.0）时无需填写。
	everest.io/sfs-version	否	仅使用通用文件存储（SFS 3.0）时需要填写，固定值为“sfs3.0”。
极速文件存储	csi.storage.k8s.io/csi-driver-name	是	驱动类型，使用极速文件存储类型时，参数取值固定为“sfsturbo.csi.everest.io”。
	csi.storage.k8s.io/fstype	是	使用极速文件存储时，支持的参数值为“nfs”。
	everest.io/share-access-to	是	集群所在VPC ID。
	everest.io/share-expand-type	否	扩展类型，默认值为“bandwidth”，表示增强型的文件系统。该字段不起作用。
	everest.io/share-source	是	参数取值固定为“sfs-turbo”。
	everest.io/share-volume-type	否	极速文件存储类型，默认值为“STANDARD”，表示标准型和标准型增强版。该字段不起作用。
对象存储	csi.storage.k8s.io/csi-driver-name	是	驱动类型，使用对象存储类型时，参数取值固定为“obs.csi.everest.io”。
	csi.storage.k8s.io/fstype	是	实例类型，支持的参数值为“s3fs”和“obsfs”。 ● obsfs：并行文件系统。 ● s3fs：对象桶。
	everest.io/obs-volume-type	是	对象存储类型。 ● fsType设置为s3fs时，支持STANDARD（标准桶）、WARM（低频访问桶）。 ● fsType设置为obsfs时，该字段不起作用。

自定义存储类应用场景

在CCE中使用存储时，最常见的方法是创建PVC时通过指定StorageClassName定义要创建存储的类型，如下所示，使用PVC申请一个SAS（高I/O）类型云硬盘/块存储。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```



```
  name: pvc-evs-example
  namespace: default
  annotations:
    everest.io/disk-volume-type: SAS
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
```

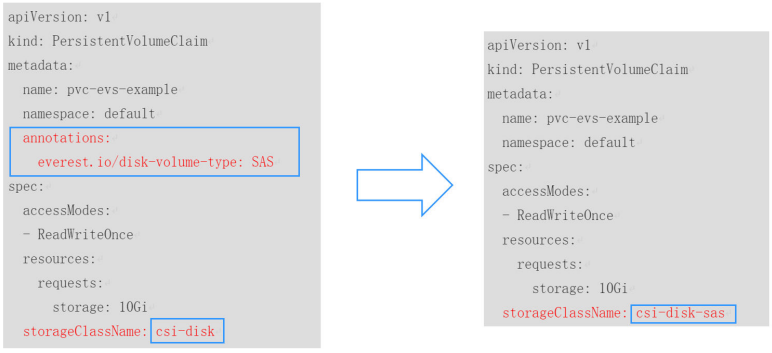
可以看到在CCE中如果需要指定云硬盘的类型，是通过everest.io/disk-volume-type字段指定，这里SAS是云硬盘的类型。

以上是较为基础的StorageClass使用方法，在实际应用中，也可以使用StorageClass进行更为简便的操作：

应用场景	解决方案	操作步骤
在使用annotations指定存储配置时，配置较为繁琐。例如此处使用everest.io/disk-volume-type字段指定云硬盘的类型。	在StorageClass的parameters字段中定义PVC的annotations，编写YAML时只需要指定StorageClassName。 例如，将SAS、SSD类型云硬盘分别定义一个StorageClass，比如定义一个名为csi-disk-sas的StorageClass，这个StorageClass创建SAS类型的存储，	场景一：指定StorageClass中的磁盘类型
当用户从自建Kubernetes或其他Kubernetes服务迁移到CCE，原先的应用YAML中使用的StorageClass与CCE中使用的不同，导致使用存储时需要修改大量YAML文件或Helm Chart包，非常繁琐且容易出错。	在CCE集群中创建与原有应用YAML中相同名称的StorageClass，迁移后无需再修改应用YAML中的StorageClassName。 例如，迁移前使用的云硬盘存储类为disk-standard，在迁移CCE集群后，可以复制CCE集群中csi-disk存储类的YAML，将其名称修改为disk-standard后重新创建。	
在YAML中必须指定StorageClassName才能使用存储，不指定StorageClass时无法正常创建。	在集群中设置默认的StorageClass，则YAML中无需指定StorageClassName也能创建存储。	场景二：指定默认StorageClass
创建的存储资源需要指定企业项目，在每个PVC中配置annotation较为繁琐。	在StorageClass中添加企业项目，创建PVC时无需指定企业项目，存储资源将默认创建在StorageClass中指定的企业项目下。	场景三：指定StorageClass的企业项目

场景一：指定 StorageClass 中的磁盘类型

本文以自定义云硬盘类型的StorageClass为例，将SAS、SSD类型云硬盘分别定义一个StorageClass，比如定义一个名为csi-disk-sas的StorageClass，这个StorageClass创建SAS类型的存储，则前后使用的差异如下图所示，编写PVC的YAML时只需要指定StorageClassName。



未使用自定义StorageClass的写法　　使用自定义StorageClass的写法

- 自定义高I/O类型StorageClass，使用YAML描述如下，这里取名为csi-disk-sas，指定云硬盘类型为SAS，即高I/O。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-sas                          # 高IO StorageClass名字，用户可自定义
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS            # 云硬盘高I/O类型，用户不可自定义
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true                    # true表示允许扩容
```

- 超高I/O类型StorageClass，这里取名为csi-disk-ssd，指定云硬盘类型为SSD，即超高I/O。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd                          # 超高I/O StorageClass名字，用户可自定义
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD            # 云硬盘超高I/O类型，用户不可自定义
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

reclaimPolicy：底层云存储的回收策略，支持Delete、Retain回收策略。

- **Delete**：删除PVC，PV资源与云硬盘均被删除。
- **Retain**：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。

如果数据安全性要求较高，建议使用**Retain**以免误删数据。

定义完之后，使用kubectl create命令创建。

```
# kubectl create -f sas.yaml
storageclass.storage.k8s.io/csi-disk-sas created
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
```

再次查询StorageClass，回显如下：

```
# kubectl get sc
NAME                PROVISIONER                     AGE
csi-disk            everest-csi-provisioner         17d
csi-disk-sas        everest-csi-provisioner         2m28s
csi-disk-ssd        everest-csi-provisioner         16s
csi-disk-topology   everest-csi-provisioner         17d
csi-nas             everest-csi-provisioner         17d
csi-obs             everest-csi-provisioner         17d
csi-sfsturbo        everest-csi-provisioner         17d
```

场景二：指定默认 StorageClass

您还可以指定某个StorageClass作为默认StorageClass，这样在创建PVC时不指定StorageClassName就会使用默认StorageClass创建。

例如将csi-disk-ssd指定为默认StorageClass，则可以按如下方式设置。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"   # 指定集群中默认的StorageClass，一个集群中只能有一个默认的StorageClass
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

使用kubectl create命令创建，然后再查询StorageClass，显示如下。

```
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
# kubectl get sc
NAME                    PROVISIONER                     AGE
csi-disk                everest-csi-provisioner         17d
csi-disk-sas            everest-csi-provisioner         114m
csi-disk-ssd (default)  everest-csi-provisioner         9s
csi-disk-topology       everest-csi-provisioner         17d
csi-nas                 everest-csi-provisioner         17d
csi-obs                 everest-csi-provisioner         17d
csi-sfsturbo            everest-csi-provisioner         17d
```

场景三：指定 StorageClass 的企业项目

CCE支持使用存储类创建云硬盘和对象存储类型PVC时指定企业项目，将创建的存储资源（云硬盘和对象存储）归属于指定的企业项目下，**企业项目可选为集群所属的企业项目或default企业项目**。

若不指定企业项目，则创建的存储资源默认使用存储类StorageClass中指定的企业项目，CCE提供的 csi-disk 和 csi-obs 存储类，所创建的存储资源属于default企业项目。

如果您希望通过StorageClass创建的存储资源能与集群在同一个企业项目，则可以自定义StorageClass，并指定企业项目ID，如下所示。

说明

该功能需要everest插件升级到1.2.33及以上版本。

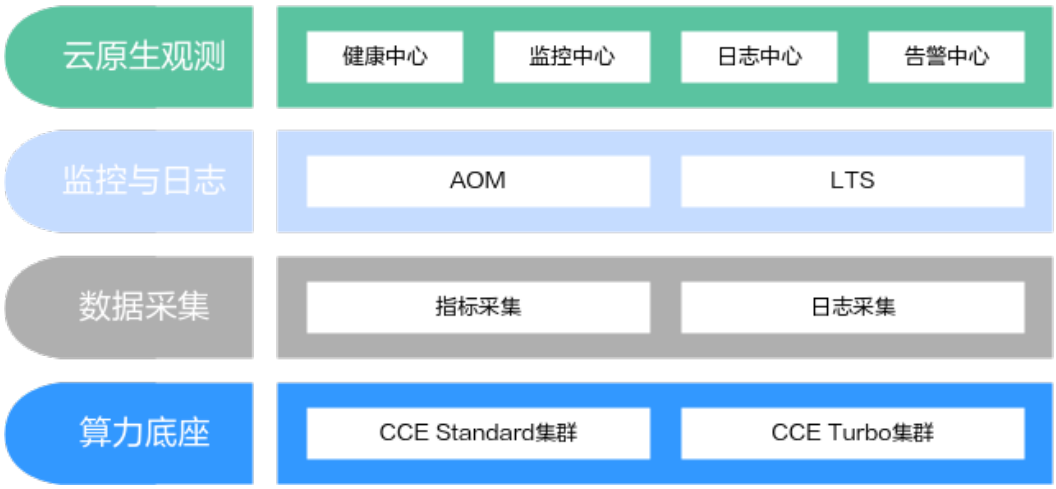
```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-disk-epid      # 自定义名称
provisioner: everest-csi-provisioner
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183  # 指定企业项目id
  everest.io/passthrough: 'true'
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate
```


9 可观测性

9.1 可观测性体系概述

云原生可观测性是指在云原生架构中，通过使用各种工具和技术来实现对应用程序和基础设施的监控告警、日志、故障排除等功能的一套完整的解决方案。本文介绍云容器引擎CCE可观测性架构分层和主要的可观测能力，以帮助您对CCE云原生可观测性生态有一个全面的认识。

图 9-1 可观测性体系



从架构分层的角度，CCE可观测性分为四个层次。自下而上分别为：**算力底座**、**数据采集**、**监控与日志**、**云原生观测**。

算力底座

云容器引擎CCE支持多种类型的集群创建，包括CCE Turbo集群与CCE Standard集群，以满足您各种业务需求。CCE集群相关介绍请前往**CCE产品介绍**。CCE服务为不同的集群类型，提供统一的数据采集方案与一致云原生观测体验。

数据采集

指标采集：CCE提供基于Prometheus的云原生监控插件，相比于开源版本，具备轻量化，开箱即用等优势。详情请参见**云原生监控插件**。

日志采集：CCE提供基于fluent-bit和opentelemetry的云原生日志采集插件，具备高性能，资源占用低的优点；同时支持基于CRD的日志采集策略配置，更加灵活易用。详情请参见**云原生日志采集插件**。

监控与日志

AOM：应用运维管理（Application Operations Management，简称AOM）是云上应用的一站式立体化运维管理平台，实时监控您的应用及相关云资源，分析应用健康状态，提供灵活丰富的数据可视化功能，帮助您及时发现故障，全面掌握应用、资源及业务的实时运行状况。

LTS：云日志服务（Log Tank Service，简称LTS），用于收集来自主机和云服务的日志数据，通过海量日志数据的分析与处理，可以将云服务和应用程序的可用性和性能最大化，为您提供实时、高效、安全的日志处理能力，帮助您快速高效地进行实时决策分析、设备运维管理、用户业务趋势分析等。

云原生观测

CCE云原生观测相关的功能包括健康中心、监控中心、日志中心、告警中心等。以下分别介绍CCE云原生观测的主要功能。

- **健康中心**

  集群健康诊断基于容器运维专家经验对集群健康状况进行全面检查，能够及时发现集群故障与潜在风险并给出修复建议。

- **监控中心**

  监控中心提供不同维度的数据洞察、仪表盘等功能。监控中心提供容器视角的可视化视图，支持集群、节点、工作负载和Pod等多种维度的监控视图，支持多级下钻与关联分析。仪表盘功能内置常见的容器监控大盘，如Kubernetes APIServer组件监控、CoreDNS组件监控和PVC监控等。

- **日志中心**

  CCE日志中心集成了云日志服务LTS。启用日志采集与管理，您可以快速采集CCE控制面组件日志（kube-apiserver、kube-controller-manager、kube-scheduler）、kubernetes审计日志、Kubernetes事件和容器日志（容器的标准输出、容器内的文本文件、节点日志）。

- **告警中心**

  告警中心集成应用运维管理服务AOM2.0的告警功能，提供容器告警一键开启能力，覆盖集群和容器常见故障场景。

资源权限

由于云原生观测相关的功能在运行中对监控、告警、通知服务等各类云服务资源都存在依赖关系，因此当您首次使用云原生观测相关的功能时，系统将自动请求获取当前区域下的云资源权限，从而更好地为您提供服务。

具体授予的权限如下表：

授权类型	权限名称	描述
CCE	IAM ReadOnlyAccess	监控中心、告警中心获得该权限后，支持子用户访问监控中心与告警中心，因此需要获得该权限。
CCE	Tenant Guest	监控中心、告警中心支持对集群关联的OBS、DNS等全局资源配置进行检查，提前发现配置问题，因此需要获得该权限。
CCE	CCE Administrator	监控中心、告警中心在运行过程中需要访问CCE获取集群、节点、工作负载等信息，以此来检测对应资源的健康状态，因此需要获得该权限。
CCE	SWR Administrator	监控中心、告警中心在运行过程中需要访问SWR获取镜像信息，因此需要获得该权限。
CCE	SMN Administrator	监控中心、告警中心在运行过程中需要访问SMN获取联系组信息，因此需要获得该权限。
CCE	AOM Administrator	监控中心、告警中心在运行过程中需要访问AOM获取监控指标信息，因此需要获得该权限。
CCE	LTS Administrator	监控中心、告警中心在运行过程中需要访问LTS获取日志信息，因此需要获得该权限。
AOM	DMS UserAccess	AOM支持用户通过DMS获取数据订阅的功能，因此需要获得该权限。
AOM	ECS CommonOperations	AOM支持通过在ECS上安装UniAgent和ICAgent获取系统指标、日志数据，因此需要获得该权限。
AOM	CES ReadOnlyAccess	AOM支持从CES同步监控指标数据，因此需要获得该权限。
AOM	CCE FullAccess	AOM支持从CCE同步容器监控指标数据，因此需要获得访问权限。
AOM	RMS ReadOnlyAccess	AOM的CMDB支持管理云服务实例数据，因此需要获得该权限。
AOM	ECS ReadOnlyAccess	AOM支持通过在ECS上安装UniAgent和ICAgent获取系统指标、日志数据，因此需要获得该权限。
AOM	LTS FullAccess	AOM支持访问LTS数据，因此需要获得该权限。
AOM	CCI FullAccess	AOM支持从CCI同步容器监控指标数据，因此需要获得该权限。

当您同意授权后，将在IAM中自动创建账号委托，将账号内的其他资源操作权限委托给华为云CCE服务和华为云AOM服务进行操作。关于委托详情，您可参考**委托其他云服务管理资源**进行了解。自动创建的委托如下：

- cia_admin_trust

  cia_admin_trust委托具有全局的Tenant Guest、IAM ReadOnlyAccess权限，区域级的Tenant Guest、CCE Administrator、SWR Administrator权限，用于对云原生观测功能所依赖的其他云服务资源进行调用。

  如果您在多个区域中使用CCE服务的云原生观测功能，则需在每个区域中分别申请Tenant Guest、CCE Administrator、SWR Administrator的云资源权限。您可前往“IAM控制台 > 委托”页签，单击“cia_admin_trust”查看各区域的授权记录。

- aom_admin_trust

  aom_admin_trust委托的说明请参见**AOM云服务授权**。

说明

由于云原生观测功能对其他云服务有许多依赖，如果没有所需的权限，可能会因为某个服务权限不足而影响云原生观测功能的正常使用。因此在使用云原生观测功能期间，请不要自行删除或者修改“cia_admin_trust”、“aom_admin_trust”委托。

9.2 健康中心

9.2.1 健康中心概述

集群健康诊断用于诊断集群的健康状态，该功能集合了容器运维专家的经验，为您提供了集群级别的健康诊断最佳实践。可对集群健康状况进行全面检查，帮助您及时发现集群故障与潜在风险，并给出应对的修复建议供您参考。

健康诊断覆盖范围

健康诊断覆盖范围如下图所示：

图 9-2 健康诊断覆盖范围

健康诊断能力项

- 支持开箱即用，可以在不开通监控中心情况下，进行基础的集群健康诊断
- 支持全量检查集群整体运行状况（开通监控中心后），发现集群故障与潜在风险

- 针对诊断结果，智能给出健康评分
- 支持定时巡检，并可视化巡检结果
- 支持查看巡检历史，方便用户分析故障原因
- 针对故障和潜在风险，给出风险等级并提供修复建议

使用场景

- 运维对集群做变更前的集群状况检测，可随时主动触发健康诊断
- 支持运维的定时巡检，可设置定时执行时间，定期检查集群风险

集群诊断健康提炼了运维专家提供的高频故障案例，分别从如下方面进行检查：

维度	检查项
运维层面	• 集群运维能力 • 集群安全组配置正确性 • 集群资源规划合理性 • 租户配额是否充足
资源与业务层面	• 存储插件（everest）健康程度 • 日志采集插件（log-agent）健康程度 • 域名解析插件（coredns）健康程度 • 业务节点负载情况 • 业务节点状态 • Pod配置健康程度 • Pod负载情况 • Pod运行状态

更多内容请参见**诊断项及修复方案**。

9.2.2 使用健康中心

云容器引擎CCE服务提供一键集群诊断能力，包括集群诊断、节点诊断、工作负载诊断、核心插件诊断和外部依赖诊断，可以辅助您定位集群中出现的问题。本文介绍如何在集群中使用集群诊断功能。

前提条件

- 已**获取资源权限**
- 集群版本高于v1.17。
- 集群处于“运行中”状态。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“健康中心”。

您可以在不开通监控中心的情况下，进行基础的集群健康诊断。如果想体验更丰富的诊断能力，请参考**开通监控中心**开通。

----结束

配置定时巡检规则

在“健康诊断”页面右上角打开“定时巡检”开关，并配置定时巡检启动的时间。集群将在指定时间自动开始集群巡检任务。单个集群，每天仅支持配置一个定时巡检时间。

图 9-3 定时巡检

手动发起诊断

当您初次使用健康诊断时，单击“马上诊断”，集群将开始执行诊断。等待一段时间后，健康诊断页面将显示健康评分、风险分布雷达图、诊断风险汇总、历史风险分布，以及诊断结果。

图 9-4 诊断概览

查看诊断结果

诊断结束后，页面将自动刷新并展示诊断结果，其中无风险项将自动隐藏。

健康诊断将针对不同维度的巡检项，归纳Kubernetes中常见的问题，并提供相应的修复建议。用户可以单击“诊断详情”查看具体诊断项的详细信息以及存在异常的资源。在部分场景下，页面还提供相应的排查文档，供用户参考排查。

图 9-5 诊断结果

9.2.3 诊断项及修复方案

集群维度

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
集群资源规划能力	集群Master节点是否高可用	是	集群为单控制节点或者存在控制节点异常，当再有控制节点故障时，集群将不可用，进而会影响集群中运行服务的可靠性。提升服务韧性建议使用高可用集群或者修复节点异常，当某个控制节点故障时，不影响集群业务。
	集群当前时间CPU的Request水位是否超过80%	是	Request代表工作负载运行的最低资源要求，集群水位过高，剩余资源不能够满足新应用Request要求时，应用将不能被创建。需要根据业务情况，合理规划资源分配。详情请参见**设置容器规格**。
	集群当前时间内存的Request水位是否超过80%	是	
	集群版本是否超期	否	集群版本EOS后，云容器引擎（CCE）将不再支持对该版本的集群创建，同时不提供相应的技术支持，包含新特性更新、漏洞/问题修复、补丁升级以及工单指导、在线排查等客户支持，不再适用于CCE服务SLA保障。请前往CCE的集群管理页面，升级集群版本。详情请参见**集群升级指导**。
集群运维能力	集群kube-prometheus-stack插件状态是否正常	否	云原生监控插件kube-prometheus-stack主要提供了集群运维监控的能力，要体验一站式监控体系，需前往插件市场，安装插件并检查插件状态。详情请参见**云原生监控插件**。

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
	集群log-agent插件状态是否正常	否	运维插件log-agent提供了集群中负载的日志采集、日志管理的能力，体验日志管理能力，帮助集群中服务问题快速定位定界。需前往插件市场，安装插件并检查插件状态。
	集群npd插件状态是否正常	否	运维插件npd（node-problem-detector）提供了节点异常监控的能力。如需体验节点监控能力，检查节点资源异常情况。需前往插件市场，安装插件并检查插件状态。详情请参见**CCE节点故障检测**。
集群配置	安全组配置是否正确	否	集群安全组配置异常，直接影响节点之前的通信，导致节点不可用。请使用默认安全组配置。

核心插件维度

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
coredns插件状态	coredns插件状态	否	coredns插件是系统必装的资源插件，为集群提供域名解析服务。插件未安装或者异常将影响集群整体业务响应，影响范围大。需前往插件市场，安装插件或者检查插件状态。
	coredns近24小时CPU使用率最大值是否超过80%	是	coredns插件负责为集群提供域名解析服务，资源使用率过高有过载风险，过载会影响域名解析成功率，增大解析时延。为保证业务不受损，请分析coredns近期QPS情况，分析可前往“监控中心 > 仪表盘 > CoreDNS视图”查看相关指标情况，如果长时间达到瓶颈，可调整coredns实例规格。
	coredns近24小时内存使用率最大值是否超过80%	是	
	coredns近24小时是否存在域名解析失败的请求	是	coredns域名解析失败，将导致业务直接受损。
	coredns近24小时P99请求时延是否超过5s	是	coredns时延增加，将影响业务响应，导致业务受损。

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
everest插件状态	everest插件状态	否	everest插件是系统必装的存储插件，为集群提供云存储服务。插件未安装或者异常会直接导致集群存储能力受影响。需前往插件市场，安装插件或者检查插件状。
	everest-controller近24小时CPU使用率最大值是否超过80%	是	everest插件负责为集群提供云存储服务。资源使用率过高会导致存在过载风险，影响集群云存储能力。为保证云存储不受影响，请分析近期everest-controller负载情况，可前往“监控中心 > 工作负载”监控中查看everest实例相关指标情况，如果长时间达到瓶颈，可调整everest实例规格，详情请参见**CCE容器存储（Everest）**。
	everest-controller近24小时内存使用率最大值是否超过80%	是	
kube-prometheus-stack插件状态	kube-prometheus-stack插件状态	否	同上kube-prometheus-stack插件状态。
	prometheus工作负载近24小时CPU使用率最大值是否超过80%	是	kube-prometheus-stack主要提供了集群运维监控的能力，资源使用率过高会导致存在过载风险，影响集群监控能力。可前往“监控中心 > 工作负载”监控中查看prometheus实例相关指标情况，如果长时间达到瓶颈，可调整prometheus实例规格。 **说明** PVC资源使用率的检查在kube-prometheus-stack插件为“本地数据存储”时执行，该模式下，采集到的指标数据会存放入集群PV中。
	prometheus工作负载近24小时内存使用率最大值是否超过80%	是	
	prometheus工作负载在Server部署模式下，prometheus-server的PVC使用率是否超过80%	是	
	prometheus工作负载近24小时是否出现OOM	否	kube-prometheus-stack主要提供了集群运维监控的能力，插件实例由于内存使用量超过限制量，出现OOM。会导致指标上报受损，非高可用模式监控能力不可用，建议调整prometheus实例规格配置。

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
autoscaler插件状态	集群在开启节点池弹性扩缩容条件下，autoscaler插件状态是否可用	否	autoscaler插件为集群提供了弹性扩展能力。在节点池开启弹性扩缩容条件下，该插件状态异常将导致无法进行扩缩容，需前往插件中心，检查插件状态。 **说明** autoscaler插件状态检查，是在开启节点池弹性扩缩容条件下检查的。节点池未开启弹性扩缩容将不进行检查。
log-agent插件状态	log-agent插件状态	否	同上log-agent插件状态
	LTS默认事件日志组、日志流是否创建成功	否	默认事件日志组和日志流分别是监控中心-事件功能的基本单位。缺失会导致监控中心-事件功能不可用，参见**通过云原生日志采集插件采集容器日志**中默认事件日志组、日志流要求进行创建。

节点维度

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
节点状态	节点状态是否就绪	是	节点为承载业务的核心资源，状态不就绪可能直接导致承载在节点上的业务受到影响，需立即修复。
	节点状态不可调度	是	节点不可调度将导致节点资源不能被正常使用，请前往CCE节点管理，查看节点状态是否符合预期。
	节点kubelet状态	是	kubelet为节点关键组件，不可用可能会导致节点异常，Pod状态不符合预期（与APIServer的Pod状态不一致）。可以到节点上通过如下命令查看kubelet日志，并分析异常原因。命令参考：**journalctl -l -u kubelet**
节点配置	节点当前时间内存的Requset水位是否超过80%	是	节点的Request水位将影响新应用能否被调度到该节点上。水位过高，剩余资源不满足应用要求时，该节点将不会被调度到。本诊断项已为您检测出了Request水位高出阈值的节点资源，可根据检测结果合理规划您的应用。
	节点当前时间CPU的Request水位是否超过80%	是	

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
节点资源水位诊断	节点24小时内CPU使用率最大值是否超过80%	是	节点的cpu过高将导致节点处理能力下降，影响节点上运行的服务。请前往监控中心，查询节点CPU使用状况，合理规划节点资源，或者对节点进行扩容。
	节点24小时内内存使用率最大值是否超过80%	是	节点内存过高，存在节点OOM风险，影响节点上服务的可用性。请前往监控中心，查看节点内存使用状况，合理规划节点资源，或者对节点进行扩容。
	节点磁盘使用率是否超过80%	是	节点磁盘使用率过高将影响系统Pod和业务Pod，请及时扩容。建议通过如下命令查看磁盘信息： • **lsblk** 列出所有可用块设备的信息 • **df -h** 列出挂载的每个磁盘中的可用磁盘空间量 • **fdisk -l** 列出所有的分区
	节点PID使用量是否正常	是	节点PID出现压力，可能导致节点不稳定，需释放无用进程或者**修改PID上限**。可以通过如下命令查看PID信息。 • 查看最大PID数：**sysctl kernel.pid_max** • 查看当前的最大PID：**ps -eLf| awk '{print $2}' | sort -rn| head -n 1** • 查看占用SPID最多的前5个进程：**ps -elT | awk '{print $4}' | sort | uniq -c | sort -k1 -g | tail -5**
	节点24小时内是否发生OOM事件	是	节点出现OOM将使节点中的服务功能受损，可前往监控中心分析内存运行状况，合理规划资源，或者进行扩容。

负载维度

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
Pod状态	Pod状态检查	否	Pod状态异常，可能会降低Pod所属工作负载的服务能力；所有副本均不可用时，会导致业务不可用。可以通过如下命令来查看 Pod 的信息： ● 查看 Pod 的配置是否正确：**kubectl get pod <PodName> -n <Namespace> -o yaml** ● 查看 Pod 的事件：**kubectl describe pod <PodName> -n <Namespace>** ● 查看容器日志：**kubectl logs <PodName> -n<Namspace> -c <ContainerName>**
Pod负载状态	Pod在24小时内是否发生OOM	否	Pod出现OOM将使对应服务功能受损，可前往监控中心分析Pod的内存运行状况，合理调整工作负载规格。
	Pod的24小时内CPU使用率最大值是否超过80%	是	资源使用率过高，业务有过载风险，将导致业务时延增加，影响业务正常响应。可前往“监控中心 > Pod”查看对应实例指标状况，如果长时间达到瓶颈，可调整容器规格。
	Pod的24小时内内存使用率最大值是否超过80%	是	
Pod配置	Pod中的容器是否配置Request	否	建议配置request，如果request未设置，会影响Scheduler的调度决策；从而可能导致Pod被调度到资源无法满足要求的节点上，导致Pod无法运行；状态混乱过高的Request同样会降低节点的资源利用率。
Pod探针配置	Pod中的容器是否配置存活探针	否	建议配置存活探针，若未配置存活探针（livenessProbe），在Pod应用异常时，无法被及时感知并重启，从而影响业务QoS。建议配置存活探针（livenessProbe），规避存在在容器内应用异常需要重启容器才能恢复时未及时重启导致业务异常的风险。

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
	Pod中的容器是否配置就绪探针	否	建议配置就绪探针，若未配置就绪探针（readinessProbe），在Pod异常无法处理请求时，仍会有请求发送过至异常Pod从而影响业务QoS。建议配置就绪探针（readinessProbe），规避存在在容器内应用异常无法处理请求时仍旧有请求发过来导致业务异常的风险。

外部依赖维度

集群诊断场景	诊断项	是否需要开通监控中心	修复方案
租户节点资源配额	租户云硬盘配额使用率是否超过90%	是	资源的创建需要基于配额完成，租户配额不足，将影响集群节点的创建，请至“资源 > 我的配额”页面，联系客服申请账号配额。
	租户ECS配额使用率是否超过90%	是	

9.3 监控中心

9.3.1 监控中心概述

监控中心是华为云打造的新一代云原生容器运维平台，可实时监控应用及资源，采集各项指标及事件等数据以分析应用健康状态，提供全面、清晰、多维度数据可视化能力，兼容主流开源组件，并提供快捷故障定位的能力。

功能介绍

- 多维度数据洞察：提供基于Kubernetes原生类型的容器监控能力，支持**集群**、**节点**、**工作负载**、**Pod**和**事件**的指标展示，全面监控集群的健康状态和负荷程度。
- **仪表盘**：仪表盘可将不同图表汇聚到同一个屏幕上，通过不同的仪表形式来展示资源数据，例如，曲线图、数字图等，进而全面、深入地掌握监控数据。

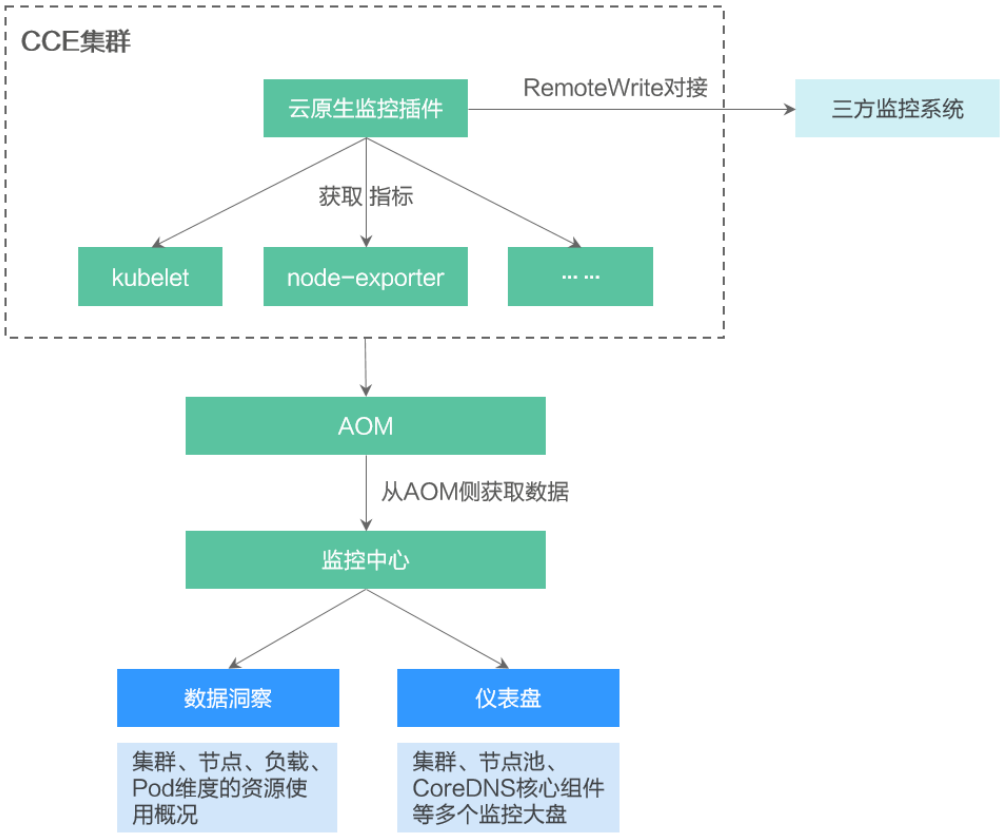
优势

- 监控中心深度整合云原生基金会（CNCF）的监控项目Prometheus。对关键指标、事件等运维数据进行统一采集、存储和可视化展现，精心打造云原生应用的良好可观测性能力。
- 将云原生基础设施监控和应用负载监控进行关联，提供全栈监控，使用户能够随时随地清晰地感知基础设施和应用负载状态。

- 能够对Kubernetes集群、节点、容器组（Pod）等进行详细监控，对业务提供端到端追踪和可视化，提供集群健康诊断能力，大大缩短问题分析定位时间。
- 提供开箱即用的插件安装、数据采集、云原生监控能力，相比基于开源组件构建的监控能力，在可靠性、高可用、安装部署便捷性上更具有竞争力，能够更好地为您的云原生应用保驾护航。
- 提供了轻量化的指标采集插件，和社区Prometheus相比，资源使用量大大降低，部署模式方便快捷。

监控中心架构

图 9-6 监控中心架构



云原生监控插件将在CCE集群中采集exporter暴露的指标，通过Prometheus RemoteWrite的方式，将数据写入至AOM实例。

监控中心将基于AOM实例中存储的指标，提供多维度数据洞察、仪表盘的功能。

云原生监控插件也提供了基于RemoteWrite对接三方云原生监控平台的能力，将集群内的监控指标通过Bearer Token认证鉴权的方式上报三方监控平台。

Prometheus 监控

Prometheus已经成为了当前云原生可观测性的最常见工具，其强大的监控能力和活跃的社区生态，使其成功CNCF最活跃的托管项目之一。当前CCE插件市场提供了**云原生监控插件**用于Kubernetes集群的监控。

华为云AOM云服务基于Prometheus监控生态，提供了托管式的Prometheus实例 for CCE，适合需要对容器服务集群及其上面运行的应用进行一体化监控场景。AOM实例默认提供对容器服务CCE集群的云原生监控插件的集成，监控中心开通后，指标将自动上报指标到指定的AOM实例。

AOM ICAgent 监控

华为云AOM云服务提供了基于主机的ICAgent的组件，用于采集指标、日志和应用性能数据。对于在ECS、BMS控制台直接购买的主机，您需手动安装ICAgent。对于集群节点，ICAgent会自动安装，您不用手动安装ICAgent。

9.3.2 开通监控中心

开通监控中心将在集群中安装云原生监控插件，该插件提供监控中心的指标采集功能。开通后，监控中心将采集集群中的指标并上报至AOM实例。本章节介绍如何为集群开通监控中心功能。

须知

- 开通监控中心后，集群中的指标将上报至AOM实例，AOM针对基础指标免费，自定义指标由AOM服务收费，具体请参考**价格详情**。
- 云原生监控插件在集群中运行需要消耗集群资源，请确保集群资源能够满足插件的安装。具体资源消耗可以前往“插件中心”云原生监控插件安装页面获取。

前提条件

开通监控中心前，用户需要使用具有admin用户组的账户完成对CCE及其依赖服务的委托授权。

授权方式：监控中心页面自动弹出“确认授权”页面，用户单击“确认授权”按钮后系统自动完成授权。所授予的权限类型请参考**资源权限**。

约束与限制

- 集群版本仅支持v1.17及以上。
- 使用监控中心前，用户需要使用具有admin用户组的账户完成对CCE及其依赖服务的委托授权。授权完成后，拥有CCE Administrator角色或CCE FullAccess权限的用户可进行监控中心所有操作；拥有CCE ReadOnlyAccess权限的用户可以查看所有资源信息，但是无法进行任何操作。
- 集群中未安装用户自建的Prometheus或**Prometheus（停止维护）**插件。

开通监控中心

- **购买集群时开通**
  a. 登录云容器引擎控制台，购买集群。
  b. 在“插件选择”页面，勾选云原生监控插件。
  c. 在“插件配置”页面，选择云原生监控插件需要对接的AOM实例。如AccessCode未创建，请先创建AccessCode。

图 9-7 启用容器监控

d. 集群创建完成后，在“节点管理”中创建节点。待节点创建成功后，云原生监控插件将自动部署至节点上。

- **在监控中心页面开通**

a. 在目标集群左侧导航栏选择“监控中心”。

b. 请确认集群中是否存在自建Prometheus，开通监控中心时安装的云原生监控插件可能与您的自建Prometheus产生冲突。

如果您的集群中已存在自建Prometheus，您可以勾选“兼容模式”，云原生监控插件将会安装在cce-monitoring命名空间下并与您的自建Prometheus共同工作，但兼容模式将存在部分约束与限制，详情请参见**云原生监控插件兼容自建Prometheus**。

c. 单击“立即开通”，并选择指标上报的AOM实例。

图 9-8 开通监控中心

d. 开通成功后，等待3-5分钟，监控数据将上报至AOM实例，随即可以使用监控中心相关功能。

- **在插件管理页面开通**

a. 在目标集群左侧导航栏选择“插件中心”。

b. 选择云原生监控插件，单击“安装”。

c. 请选择“监控数据上报至AOM服务”，其余两项数据存储配置可按需选择。

图 9-9 安装云原生监控插件

d. 插件安装完成3-5分钟后，监控数据将上报至AOM实例，随即可以使用监控中心相关功能。

说明

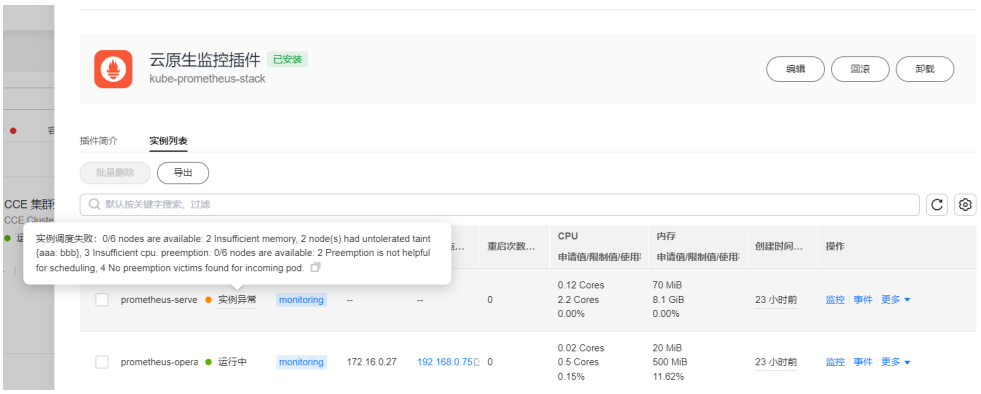
如需关闭监控中心，请前往CCE控制台“插件管理”页面卸载云原生监控插件，或关闭AOM对接，即可以停止使用该功能。

常见问题

- 监控中心开通失败，插件状态异常。

  解决方案：请前往“插件管理”页面查看已安装插件列表，单击云原生监控插件名称，展开实例列表，检查状态为异常的Pod的事件，根据界面报错信息排查异常原因。

  图 9-10 插件状态异常

- 成功进入监控中心页面，但页面数据为空。

  解决方案：

  a. 请前往“插件中心”页面查看已安装插件列表，单击云原生监控插件名称，展开实例列表，检查Prometheus的实例是否正常运行。如果未正常运行，请查询Pod的事件，获取异常信息。

  例如：报错信息为实例调度失败：0/6 nodes are available: 1 Insufficient cpu, 2 node(s) had taint {cie.manage: proxy}, that the pod didn't tolerate, 3 node(s) had taint {node.kubernetes.io/unreachable: }, that the pod didn't tolerate，说明当前集群中总共6台节点，1台节点CPU不足，剩下的5台节点标记有污点，导致Pod无法调度。

  b. 如果插件状态正常，则可以查询prometheus实例的日志，检查日志中是否存在报错日志。如果日志中含有remote_write相关的报错信息，则表示指标上报时失败，请检查指标上报的网络是否通畅。

9.3.3 管理监控采集任务

您可以简单、方便地可视化管理采集任务，所有的配置均可在升级云原生监控插件时得到保留。

前提条件

集群中已安装云原生监控插件3.11.0及以上版本。

管理监控采集任务

须知

开启默认关闭的采集任务、添加**基础免费指标**之外的指标后，若您已对接AOM，AOM服务会按量收取费用。具体请参考**价格详情**。

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“配置中心”，切换至“监控运维配置”页签。

步骤3 修改“采集配置”。

监控采集任务配置由**系统预置采集配置**、**ServiceMonitor采集配置**、**PodMonitor采集配置**和**Targets采集配置**配置项共同提供。

----结束

系统预置采集配置

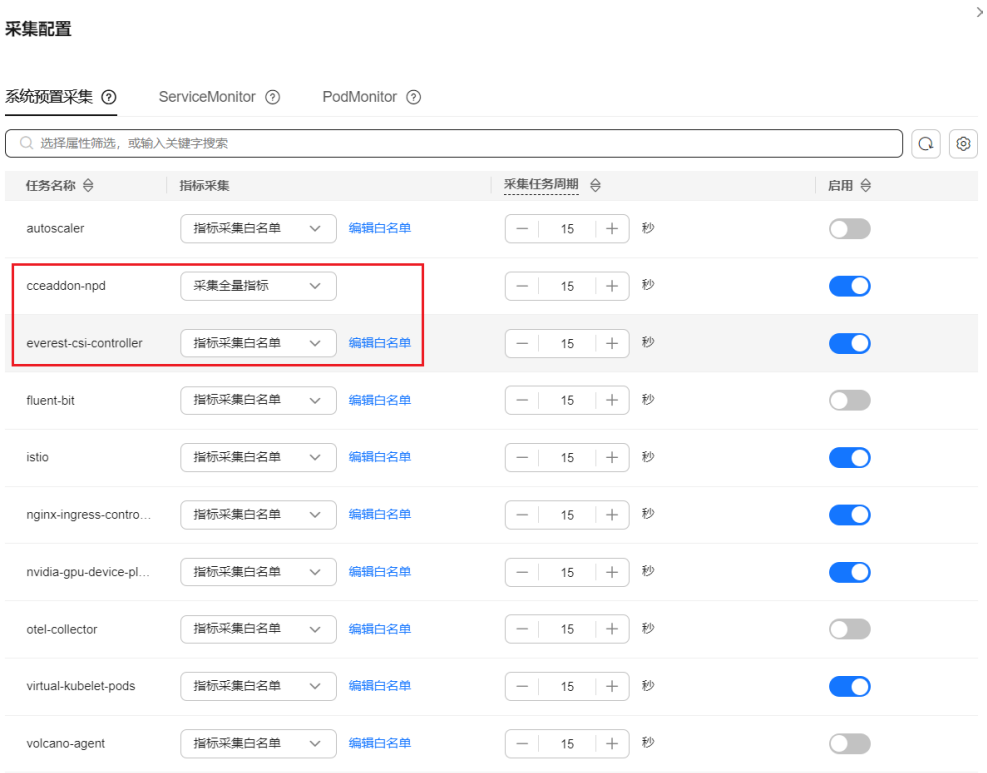
说明

为保证插件默认行为的一致性，系统预置采集功能默认不开启，强烈建议您开启系统预置采集功能。

开启预置采集后，系统预置的采集任务会由ServiceMonitor/PodMonitor形式转换为方便可视化管理的采集任务形式，您可以方便地对云原生监控插件的系统预置采集任务进行管理，按需开启或关闭采集任务，添加**基础免费指标**外的采集指标等。

您对系统预置采集任务的管理，在插件升级时可以得到继承和保留。与此同时，kube-state-metrics和node-exporter两个工作负载也会升级为由Operator统一管理，您后续对这两个工作负载的个性化配置也会在插件升级时最大程度的得到保留。

- 指标采集管理



您可按需选择每一个系统预置采集任务的指标采集行为进行管理：

  - 若您选择采集全量指标，则会采集该采集任务的所有指标。
  - 若您选择指标采集白名单，则可以按需编辑白名单（**基础免费指标**无需添加），更加精确的控制自定义采集内容，降低您集群的资源消耗及指标上报成本。

- 采集任务周期管理

  您可以按需对特定的系统采集任务的采集周期进行个性化配置。

  须知

  建议您保持kubelet、kubelet-cadvisor、kube-state-metrics、virtual-kubelet-pods四个采集任务的采集周期一致。

- 采集任务启停

  您可以按需开启或关闭系统采集任务。

ServiceMonitor 采集配置

您可以按需创建、修改、删除、启停ServiceMonitor。关于ServiceMonitor的创建方式请参见**配置Service Monitor监控自定义指标**。

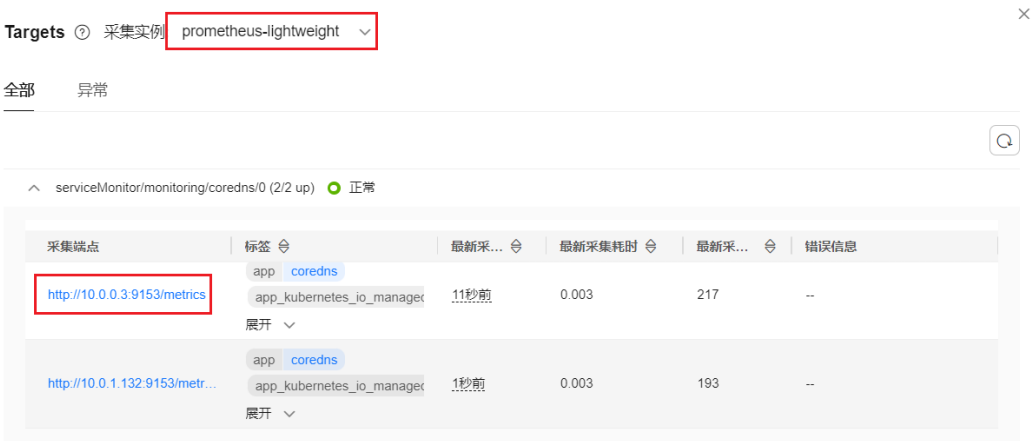
PodMonitor 采集配置

您可以按需创建、修改、删除、启停PodMonitor。关于PodMonitor的创建方式请参见**配置Pod Monitor监控自定义指标**。

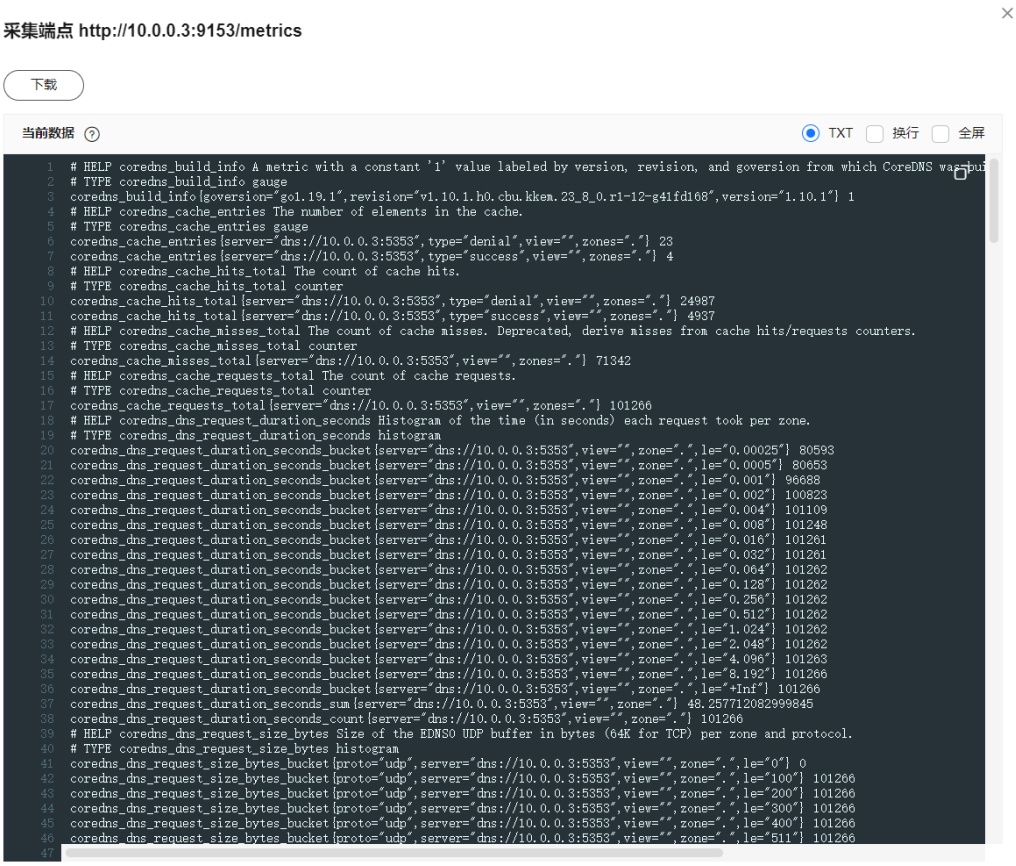
Targets 采集配置

您可以在targets页面方便的查看您的采集任务状态，包含采集端点、标签、最新采集时间、最新采集耗时、最新采集Sample数、错误信息。

若您的云原生监控插件开启了分片，则会有多个采集实例，可在采集实例处进行切换。



在云原生监控插件本地数据存储关闭的情况下，采集端点支持单击直接访问，查看采集结果，方便您对采集任务进行查看和分析管理。



采集端点访问 403 的原因是什么？该如何处理？

问题根因

您的采集端点对应的采集任务ServiceMonitor/PodMonitor配置了认证，出于安全考虑，页面访问默认不支持访问需认证的端点。

解决方案：您可以通过配置，允许访问带认证的端点。

须知

配置允许访问带认证的端点，会导致您需认证的端点可在集群内通过访问prometheus-lightweight服务的方式直接访问，因此请勿将prometheus-lightweight服务端口暴露至集群外部。

1. 登录CCE控制台，单击集群名称进入集群详情页。
2. 在左侧导航栏中选择“配置与密钥”，并切换至“全部命名空间”，找到名为“persistent-user-config”的配置项。
3. 单击“更新”，对lightweight-user-config.yaml配置数据进行编辑，在operatorConfigOverride字段下增加一条配置。

```
customSettings:
  operatorEnvOverride: []
  operatorConfigOverride:
  - --target-response-auto-auth=true
  promAdapterConfigOverride: []
```

4. 单击“确定”保存配置项，等待约1分钟即可生效。

9.3.4 集群监控

当您想观测整个集群的资源使用情况和健康度时，可以在“监控中心 > 集群”页面查看，该页面提供了单个集群的监控情况，包含**集群健康度**、**健康概况**、**资源消耗Top统计**和**数据面监控**多维度的信息概况。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“集群”页签。

----结束

集群健康度

集群健康度评估包括多个维度，如健康评分、待处理风险项数、风险等级，以及诊断风险项在Master、集群、节点、工作负载和外部依赖五个方面的占比（异常数据使用红色突出显示）。欲了解更多诊断结果，请前往**健康中心**页面查看。

图 9-11 集群健康度

健康概况

资源健康概况

资源健康概况涵盖了节点、工作负载和Pod三类资源中异常资源所占比例，以及命名空间的总数，以便及时发现和解决业务异常。

控制面健康概况

除了控制面组件和Master节点的异常占比，控制面资源概况中还提供了API Server的总QPS和请求错误率指标。作为集群的API服务提供者，控制面API Server的异常可能会导致整个集群无法访问，同时也会影响依赖API Server的工作负载的正常运行，QPS和请求错误率可以帮助您快速识别和修复问题。

图 9-12 健康概况

资源消耗 Top 统计

在资源消耗Top统计中，CCE服务会将CPU使用率和内存使用率排名前五的节点、无状态负载、有状态负载和Pod纳入统计范围，以帮助您识别资源消耗“大户”。如果您需要查看全部数据，可前往**节点**、**工作负载**或**Pod**页面。

图 9-13 资源消耗 Top 统计

监控名词解释：

- CPU使用率
  - 节点CPU使用率 = 节点的CPU非空闲时间所占的平均比例。
  - 工作负载CPU使用率 = 工作负载各个Pod中CPU使用率的平均值
  - Pod CPU使用率 = Pod实际使用的CPU核数 / 业务容器CPU核数限制值之和（未配置限制值时采用节点总量）
- 内存使用率
  - 节点内存使用率 = 节点的内存使用量除以节点的内存总量。
  - 工作负载内存使用率 = 工作负载各个Pod中内存使用率的平均值
  - Pod内存使用率 = Pod实际使用的物理内存 / 业务容器物理内存限制值之和（未配置限制值时采用节点总量）

数据面监控

此处默认统计近1小时、近8小时和近24小时的各维度资源用量。如需查看更多监控信息，请单击“查看全部监控”，跳转至“仪表盘”页面，相应指导请参见**使用仪表盘**。

说明

您可以将鼠标悬停在图表上，以便查看每分钟的监控数据。

- **CPU**：单位时间内集群CPU使用情况的统计。
- **内存**：单位时间内集群内存使用情况的统计。
- **PVC存储状态**：PVC和PV的绑定情况。
- **Pod数量状态趋势**：实时监控集群Pod的状态。
- **Pod总重启次数趋势**：近5分钟的集群的Pod重启次数总和。
- **节点状态趋势**：实时监控集群节点的状态。

9.3.5 节点监控

如果您需要监控节点的资源使用情况，可以前往“监控中心 > 节点”页面查看。该页面提供了指定集群下所有节点的综合信息，以及单个节点的详细监控数据，包括CPU/内存使用率、网络流入/流出速率、磁盘读/写IO等。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“节点”页签。

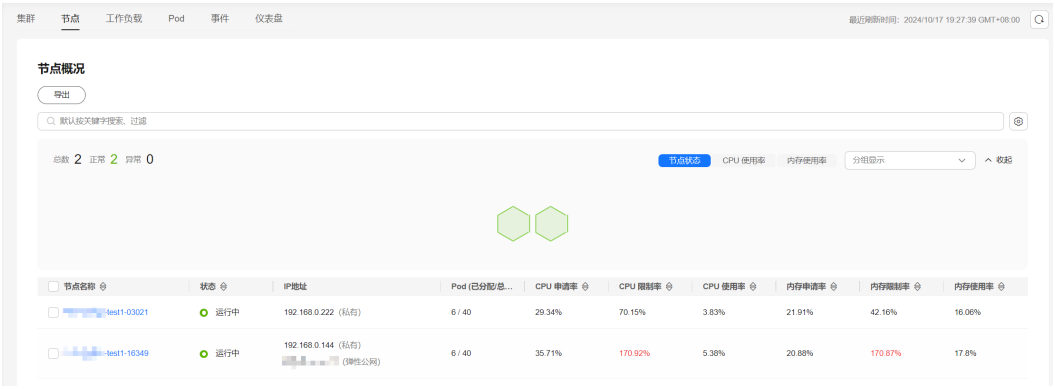
节点列表页面呈现了所有节点的综合信息，如需深入了解单个节点的监控情况，可单击节点名称，进入该节点的“**概览**”页面，通过切换“**Pod列表**”、“**监控**”页签查看相应内容。

----结束

节点列表

节点列表中包含节点名称、状态、IP地址、Pod（已分配/总额度）、CPU申请率/限制率/使用率，以及内存申请率/限制率/使用率等信息。

图 9-14 节点列表

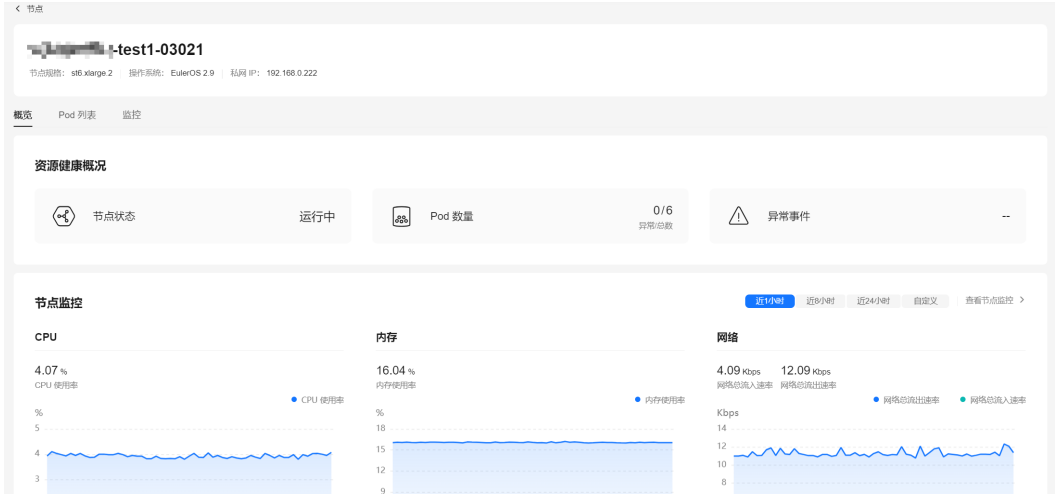


您可以通过在列表上方按照节点名称、状态、私有地址和公网地址进行筛选，快速找到需要的节点。您也可以单击“导出”按钮导出全部节点数据，或者选择部分节点进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

当节点的CPU限制率或内存限制率超过100%时，意味着节点资源超分，节点上的负载限制值（可使用的最大值）之和已经超过了节点规格。如果负载占用资源过高，可能会导致业务负载互相抢占资源，引发业务异常乃至节点异常。

概览

图 9-15 资源概况和监控概览



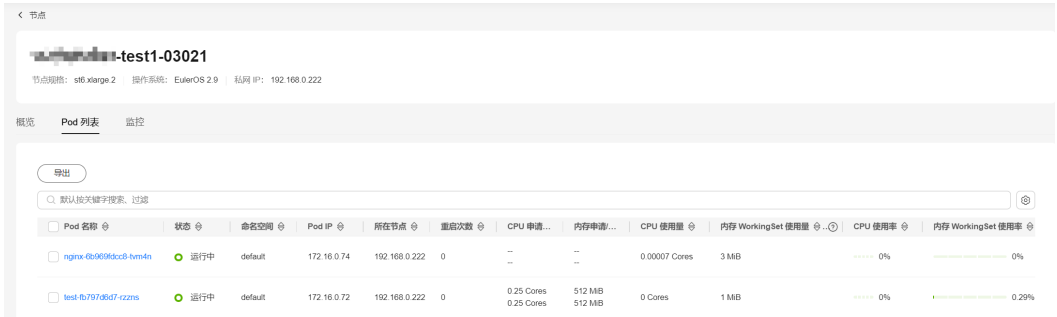
- **资源健康概况**：包括节点状态、Pod数量以及异常事件。
- **节点监控**：您可以浏览近一小时的监控概览，其中包括CPU使用率、内存使用率和网络流入/流出速率这些常见的监控指标。
- **Pod使用趋势**：您可以从中了解节点中各Pod的资源使用情况，并且支持查看降序Top5和升序Top5数据。

如需了解更多指标，请前往**监控**页面查看。

Pod 列表

Pod列表中包含了Pod名称、状态、命名空间、Pod IP、所在节点、重启次数以及各类CPU/内存资源使用指标等详细信息。

图 9-16 Pod 列表



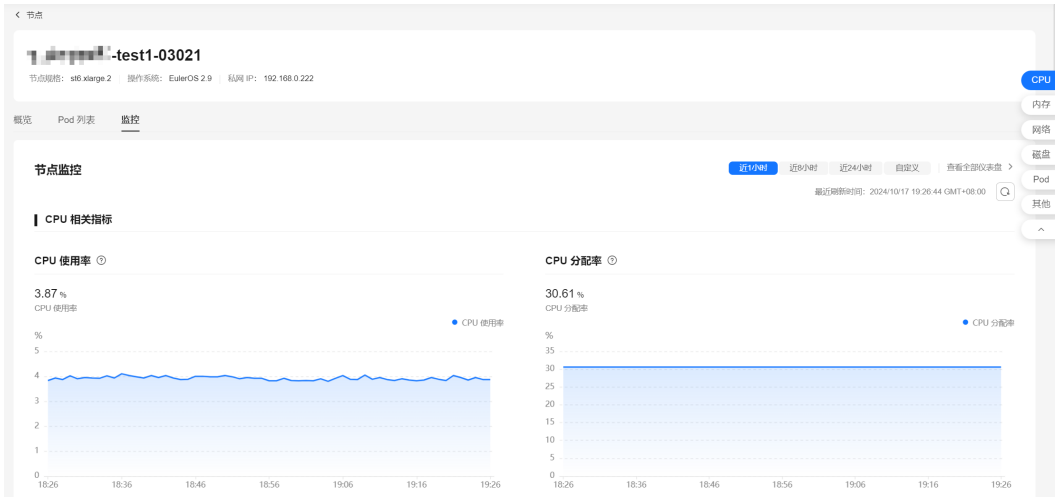
您可以通过在列表上方按照Pod名称、状态、命名空间、Pod IP和所在节点进行筛选，快速找到需要的Pod。您也可以单击“导出”按钮来导出全部Pod数据，或者选择部分Pod进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

单击Pod名称可以查看Pod的详细监控数据。更多相关内容，请参见**Pod监控**。

监控

在此处，您可以方便地查看节点在近1小时、近8小时、近24小时以及自定义时间段内各维度资源的使用情况。如需查看更多监控信息，请单击“查看全部仪表盘”，跳转至“仪表盘”页面，相应指导请参见**使用仪表盘**。

图 9-17 节点监控



- **CPU相关指标**
  - CPU使用率：节点的CPU非空闲时间所占的平均比例。
  - CPU分配率：节点上所有容器对CPU的Request之和除以节点的CPU总核数。
  - 节点CPU单核使用率：节点上每个CPU核非空闲时间各自所占的比例。
- **内存相关指标**
  - 内存使用率：节点的内存使用量除以节点的内存总量。
  - 内存分配率：节点上所有容器对内存的 Request 之和占节点的内存总量的比例。
- **网络相关指标**
  - 网络流出速率：节点上的物理网卡在不同的时间段的每秒钟发送的字节数。
  - 网络流入速率：节点上的物理网卡在不同的时间段的每秒钟接收的字节数。
  - 网络发送丢包率：节点的物理网卡网络发送丢包速率。
  - 网络接收丢包率：节点的物理网卡网络接收丢包率。
- **磁盘相关指标**
  - 磁盘读取速率：节点上的每个文件系统在不同的时间段的每秒钟读取的字节数。
  - 磁盘写入速率：节点上的每个文件系统在不同的时间段的每秒钟写入的字节数。
  - 磁盘使用率：节点上的每个文件系统在不同的时间段已使用的空间所占各自总空间的比例。
- **Pod相关指标**
  - Pod CPU使用率：节点上每个Pod在不同的时间段的CPU使用量占它们的CPU Limit量的比例。

  - Pod内存使用率：节点上每个Pod在不同的时间段的内存使用量占它们的内存Limit量的比例。
  - Pod状态数量趋势：节点上在不同的时间段分别处于不可用、未就绪、运行中、已完成或其他的状态Pod数量之和。
  - Pod数量变化趋势：节点上所有的Pod在不同的时间段的数量。
- **其他指标**
  - 节点平均负载：节点的平均负载是指在一定时间内，节点上正在运行的进程数量的平均值。即节点上正在运行的进程数量是否过多，是否超出了节点的处理能力。通常情况下，节点平均负载应该保持在一个合理的范围内，以确保节点的稳定性和可靠性。
  - Iptables 连接数：连接跟踪表的最大条目数和当前已分配的条目数。

9.3.6 工作负载监控

如果您需要监控工作负载的资源使用情况，可以前往“监控中心 > 工作负载”页面查看。该页面提供了指定集群下所有工作负载的综合信息，以及单个工作负载的详细监控数据，包括CPU/内存使用率、网络流入/流出速率等。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“工作负载”页签。

工作负载列表页面呈现了所有工作负载的综合信息，如需深入了解单个工作负载的监控情况，可单击工作负载名称，进入该工作负载的“**概览**”页面，通过切换“**Pod列表**”、“**监控**”页签查看相应内容。

----结束

工作负载列表

工作负载列表中包含工作负载名称、状态、Pod个数（正常/全部）、命名空间、镜像名称以及各类CPU/内存资源使用指标等信息。

图 9-18 工作负载列表

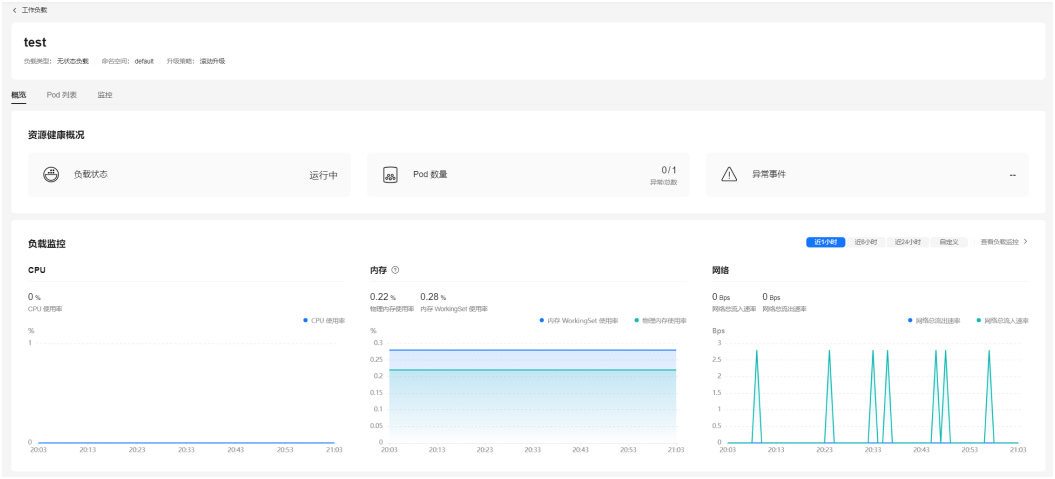
您可以利用页面右上角的工作负载类型，以及列表上方的工作负载名称、状态和命名空间进行筛选，快速定位所需的工作负载。

您也可以单击“导出”按钮来导出全部工作负载数据，或者选择部分工作负载进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

概览

单击工作负载名称，您可以方便地查看资源概况，包括负载状态、Pod数量（异常/总数）以及异常事件。此外，还可以浏览近一小时的监控概览，其中包括CPU使用率、内存使用率和网络流入/流出速率这些常见的监控指标。

图 9-19 资源概况和监控概览



同时，概览页面还提供了Pod使用趋势功能，您可以从中了解工作负载中各Pod的资源使用情况，并且支持查看降序Top5和升序Top5数据。

如需了解更多指标，请前往**监控**页面查看。

Pod 列表

Pod列表中包含了Pod名称、状态、命名空间、Pod IP、所在节点、重启次数以及各类CPU/内存资源使用指标等详细信息。

图 9-20 Pod 列表

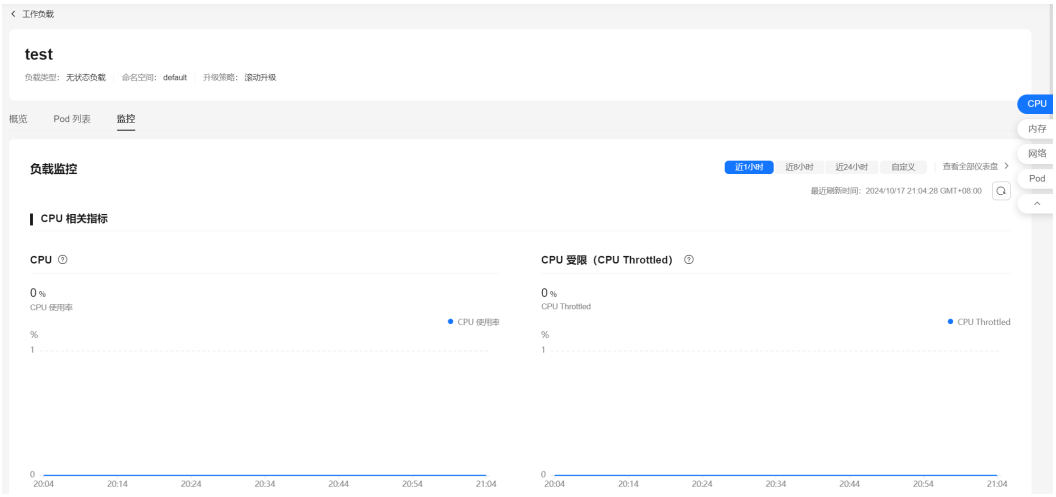
您可以通过在列表上方按照Pod名称、状态、命名空间、Pod IP和所在节点进行筛选，快速找到需要的Pod。您也可以单击“导出”按钮来导出全部Pod数据，或者选择部分Pod进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

单击Pod名称可以查看Pod的详细监控数据。更多相关内容，请参见**Pod监控**。

监控

在此处，您可以方便地查看工作负载在近1小时、近8小时、近24小时以及自定义时间段内各维度资源的使用情况。如需查看更多监控信息，请单击“查看全部仪表盘”，跳转至“仪表盘”页面，相应指导请参见**使用仪表盘**。

图 9-21 工作负载监控



- **CPU相关指标**
  - CPU：负载的所有Pod的容器在不同的时间段使用的CPU总量占负载的所有Pod的容器的CPU Limit总量的比例。
  - CPU 受限（CPU Throttled）：负载的所有Pod的容器在不同的时间段的CPU受限时间所占的平均比例。
- **内存相关指标**
  - 内存使用率：负载的所有Pod的容器在不同的时间段使用的内存总量占负载的所有Pod的容器的内存Limit总量比例。
- **网络相关指标**
  - 网络总流出速率：负载的所有Pod的容器在不同的时间段的每秒钟发送的总字节数。
  - 网络总流入速率：负载的所有Pod的容器在不同的时间段的每秒钟接收的总字节数。
  - 网络发送丢包率：负载的所有Pod的容器在不同的时间段的发送丢失的数据包总量占发送的数据包总量的比例。
  - 网络接收丢包率：负载的所有Pod的容器在不同的时间段的接收丢失的数据包总量占接收的数据包总量的比例。
- **Pod相关指标**
  - Pod CPU使用率：负载的每个Pod在不同的时间段的CPU使用量除以它们的CPU Limit量。
  - Pod内存使用率：负载的每个Pod在不同的时间段的内存使用量除以它们的内存Limit量。
  - Pod状态数量趋势：负载在不同的时间段分别处于不可用、未就绪、运行中、已完成或其他的状态Pod数量之和。

- Pod数量变化趋势：负载的Pod（副本）在不同的时间段的数量。

9.3.7 Pod 监控

如果您需要监控Pod的资源使用情况，可以前往“监控中心 > Pod”页面查看。该页面提供了指定集群下所有Pod的综合信息，以及单个Pod的详细监控数据，包括CPU/内存使用率、网络流入/流出速率等。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“Pod”页签。

Pod列表页面呈现了所有Pod的综合信息，如需深入了解单个Pod的监控情况，可单击Pod名称，进入该Pod的“**概览**”页面，通过切换“**容器列表**”、“**监控**”页签查看相应内容。

----结束

Pod 列表

Pod列表中包含Pod名称、状态、命名空间、Pod IP、所在节点、重启次数以及各类CPU/内存资源使用指标等信息。

图 9-22 Pod 列表

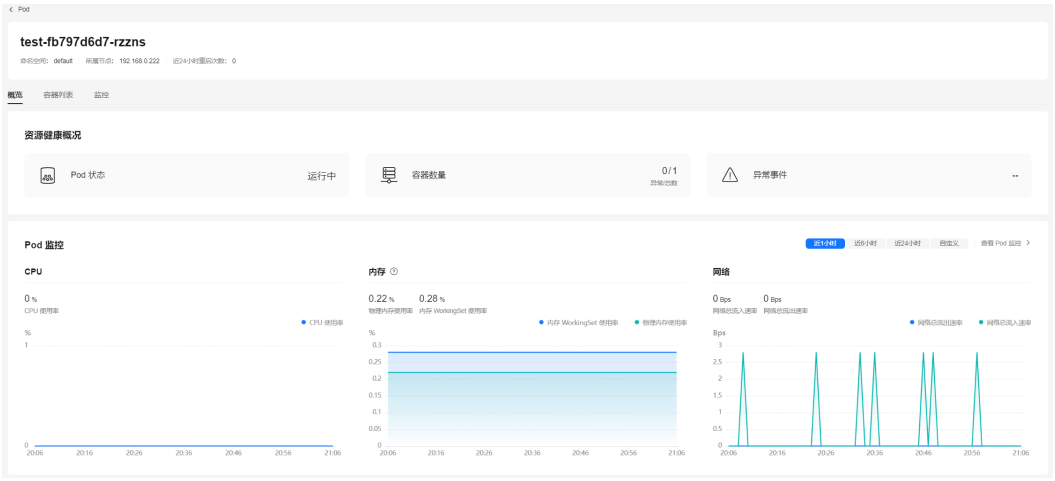
您可以利用列表上方的命名空间，以及搜索栏中的Pod名称、状态、Pod IP和所在节点进行筛选，快速定位所需的Pod。

您也可以单击“导出”按钮来导出全部Pod数据，或者选择部分Pod进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

概览

单击Pod名称，您可以方便地查看资源概况，包括Pod状态、容器数量（异常/总数）以及异常事件。此外，还可以浏览Pod及Pod所在节点近一小时的监控概览，其中包括CPU使用率、内存使用率和网络流入/流出速率这些常见的监控指标。

图 9-23 资源概况和监控概览



同时，概览页面还提供了容器使用趋势功能，您可以从中了解Pod中各容器的资源使用情况，并且支持查看降序Top5和升序Top5数据。

如需了解更多指标，请前往**监控**页面查看。

容器列表

容器列表中包含了容器名称、状态、命名空间、重启次数，以及镜像等详细信息。

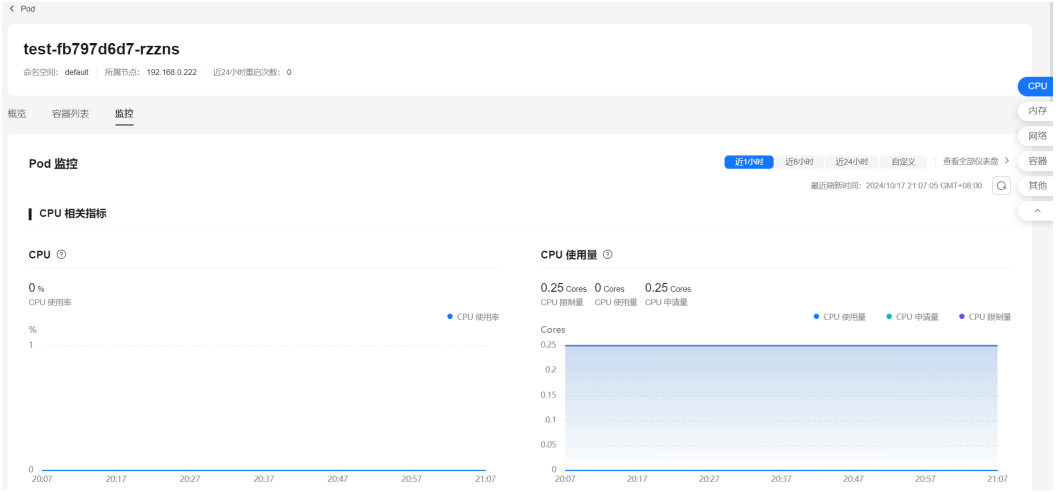
图 9-24 容器列表

您可以通过在列表上方按照容器名称、状态和命名空间进行筛选，快速找到需要的容器。您也可以单击“导出”按钮导出全部容器数据，或者选择部分容器进行导出，此时仅导出所选中的数据。导出的文件为“.xlsx”格式，文件命名中包含时间戳。

监控

在此处，您可以方便地查看Pod在近1小时、近8小时、近24小时以及自定义时间段内各维度资源的使用情况。如需查看更多监控信息，请单击“查看全部仪表盘”，跳转至“仪表盘”页面，相应指导请参见**使用仪表盘**。

图 9-25 Pod 监控



- **CPU相关指标**
  - CPU：Pod的所有容器在不同的时间段CPU使用总量占Pod的所有容器CPU Limit总量的比例。
  - CPU 使用量：Pod已经使用的CPU核数。
  - CPU 申请量：Pod CPU Request值。
  - CPU 限制量：Pod CPU Limit值，使用量接近该值时容器的CPU资源会被限流，影响容器性能。
- **内存相关指标**
  - 内存使用率：Pod的所有容器在不同的时间段内存使用总量占Pod的所有容器内存Limit总量。
  - 内存使用量：Pod已经使用的内存量。
  - 内存申请量：Pod内存Request值。
  - 内存限制量：Pod内存Limit值，使用量到达该值时会导致容器OOM。
- **网络相关指标**
  - 网络总流出速率：Pod的所有容器每秒钟发送的总字节数。
  - 网络总流入速率：Pod的所有容器每秒钟接收的总字节数。
- **容器相关指标**
  - 容器CPU使用率：Pod的每个容器在不同的时间段的CPU使用量占它们的CPU Limit量的比例。
  - 容器内存使用率：Pod的每个容器在不同的时间段的内存使用量占它们的内存Limit量的比例。
  - 容器CPU受限：Pod的每个容器在不同的时间段的CPU受限时间所占的比例。
  - 容器网络丢包率：Pod的每个的容器在不同的时间段接收丢失的数据包总量占接收的数据包总量的比例。
- **其他指标**
  - Pod 历史状态：Pod在不同时间段所处的状态。
  - 容器历史状态：Pod的每个容器在不同的时间段所处的状态。

9.3.8 事件监控

Kubernetes事件涵盖了集群的运行状态和各类资源的调度情况，对运维人员日常观察资源的变更以及定位问题均有帮助。为了实现这一目标，您需要为集群安装log-agent插件，该插件可以采集Kubernetes事件，并在“监控中心 > 事件”页面进行展示。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“事件”页签。

事件页面分为两个页签：“**概览**”和“**事件**”。在“概览”页签中，您可以查看集群中事件的总数、趋势和排序信息；在“事件”中，可以查看事件的详细信息，包括事件名称、类型、内容，以及触发该事件的资源的相关信息等。

----结束

概览

“概览”页面默认展示集群中所有命名空间的事件统计信息，您也可以在右上角的下拉框中切换命名空间，以查看指定命名空间下的事件数据。

根据图9-26的事件统计数据，您可以清晰地了解到Normal和Warning事件的数量分布情况，呈现为一个圆环图；Warning事件资源维度TOP5指的是排名前五的Warning事件数量所对应的资源信息；Warning事件环比指近24小时Warning事件数与前24小时之间的比较。

图 9-26 事件统计

通过图9-27中的柱状图，您可以观察24小时内Normal事件和Warning事件的数量变化趋势。

图 9-27 Warning/Normal 事件趋势

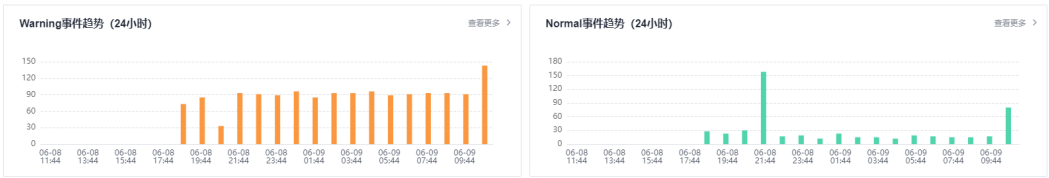


图9-28展示了24小时内事件数量排名前十的事件名称。

图 9-28 24 小时事件数量 TOP 10

事件

事件搜索

事件页面的主要功能是展示按照一定条件搜索出的指定资源的事件信息，包括Normal/Warning事件趋势和详情。这样，用户可以更加方便地查看与该资源相关的事件信息。

您可以按照如下几种方式进行事件搜索。

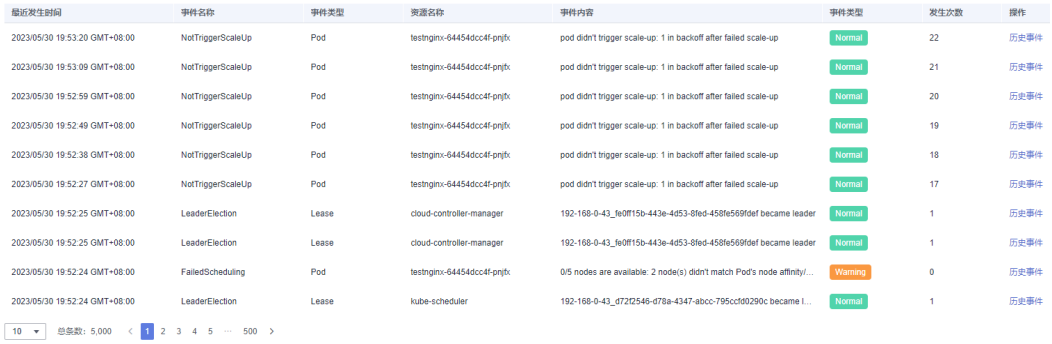
- 在输入框里输入要搜索的事件名称，选择命名空间或者事件类型，单击“搜索”按钮进行搜索。
- 单击“高级搜索”，按照工作负载、节点、Pod名称、事件内容、资源类型或者资源名称进行搜索。
- 也可以在左上角选择事件发生的时间范围，包括近1小时、近1天、近1周和自定义。

图 9-29 搜索事件

事件列表

您可以在列表中查看满足搜索条件的事件详情，包括最近发生时间、事件名称、资源类型、资源名称、事件内容、事件类型和发生次数。单击操作列的“历史事件”，在弹出的对话框中将展示当前资源类型和资源名称下的所有事件。

图 9-30 事件列表



9.3.9 仪表盘

9.3.9.1 使用仪表盘

仪表盘集合了不同视角、不同组件的高频监控指标。将不同的指标以图表的形式直观、综合性地汇集在同一个屏幕上，帮助您实时全面地掌握集群整体运行状况。

仪表盘提供了丰富的视图监控指标呈现，包括集群视图、APIServer视图、Pod视图、主机视图、Node视图等等。

前提条件

- 集群版本高于v1.17。
- 集群处于“运行中”状态。
- 集群已开通“监控中心”。

查看/切换视图

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“监控中心”，单击“仪表盘”页签，默认展示集群视图。

步骤3 监控中心仪表盘提供了预置视图，您可单击视图名称边上的“切换视图”按钮，选择需要的视图查看监控数据。

步骤4 设置查看视图的相关参数。

步骤5 设置视图的时间窗。

在页面右上角处，选择时间段，或者自定义时间，并单击刷新界面。

----结束

9.3.9.2 集群视图

基于集群的指标和PromQL语句，提供了集群节点、CPU、内存、网络、磁盘等关键资源相关图表，帮助您了解整体集群的资源运行状态。接下来主要从指标说明、指标清单两个部分来进行图表的说明，其中图表中对于数值过大的字节（bytes）会换算为MB、KB、GB等。

指标说明

集群视图暴露的指标包括基础资源指标、网络指标和磁盘指标，具体说明如下：

图 9-31 基础资源图表

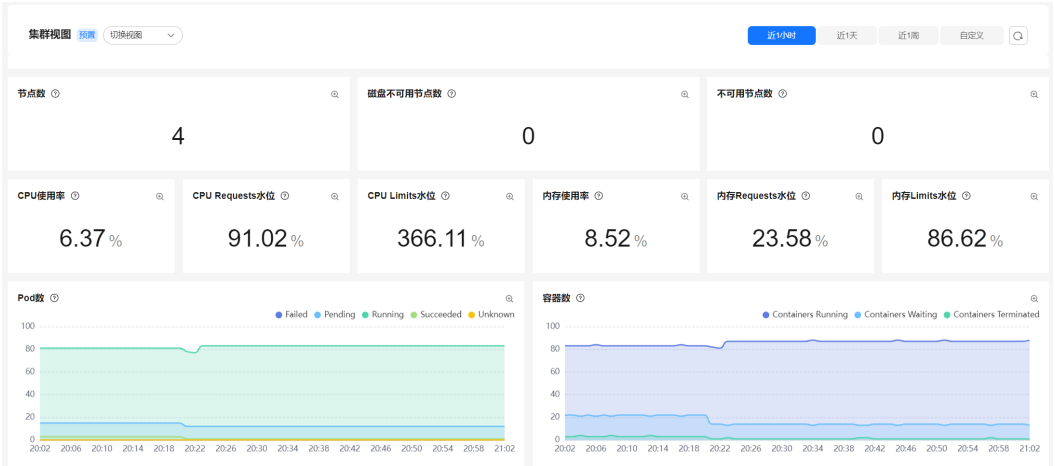


表 9-1 基础资源图表说明

指标名称	单位	说明
节点数	个	集群中的节点个数。
磁盘不可用节点数	个	集群中磁盘不可用的节点个数。
不可用节点数	个	集群中未就绪的节点个数。
CPU使用率	百分比	集群中所有容器的CPU使用量总和占所有容器设置的Limit总和的百分比。
CPU Requests水位	百分比	集群整体CPU Requests占集群CPU容量的百分比。
CPU Limits水位	百分比	集群整体CPU Limits占集群CPU容量的百分比。
内存使用率	百分比	集群中所有容器的内存使用量总和占所有容器设置的Limit总和的百分比。
内存Request水位	百分比	集群整体内存Requests占集群内存容量的百分比。
内存Limit水位	百分比	集群整体内存Limits占集群内存容量的百分比。
Pod数	个	集群中处在不同运行状态下的Pod个数（状态包含：Failed、Pending、Running、Succeeded、Unknown等）。
容器数	个	集群中处在不同运行状态下的容器个数（状态包含：Containers Running、Containers Waiting、Containers Terminated等）。
CPU使用量	Cores	以命名空间为粒度统计各个命名空间内的所有容器的CPU使用量之和。
内存使用量	字节	以命名空间为粒度统计各个命名空间内的所有容器的内存使用量之和。

图 9-32 网络图表

表 9-2 网络图表说明

指标名称	单位	说明
网络接收速率	字节/秒	以命名空间为粒度统计各个命名空间内的所有容器每秒接收的字节数之和。
网络发送速率	字节/秒	以命名空间为粒度统计各个命名空间内的所有容器每秒传输的字节数之和。
网络平均接收速率	字节/秒	以命名空间为粒度统计各个命名空间内的容器每秒平均接收的字节数。
网络平均发送速率	字节/秒	以命名空间为粒度统计各个命名空间内的容器每秒平均传输的字节数。
接收数据包速率	个/秒	以命名空间为粒度统计各个命名空间内的所有容器每秒接收的数据包数之和。
集群发送数据包速率	个/秒	以命名空间为粒度统计各个命名空间内所有容器每秒发送的数据包数之和。
丢包速率（接收）	个/秒	以命名空间为粒度统计各个命名空间内所有容器每秒接收的数据丢包数之和。
丢包速率（发送）	个/秒	以命名空间为粒度统计各个命名空间内所有容器每秒发送的数据丢包数之和。

图 9-33 磁盘图表

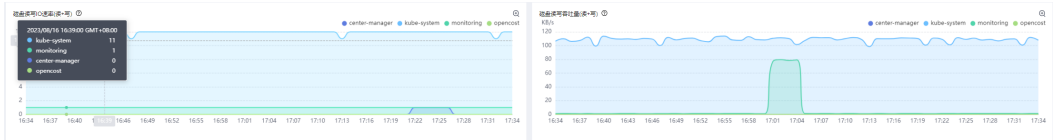


表 9-3 磁盘图表说明

指标说明	单位	说明
磁盘读写IO速率(读+写)	次数/秒	以命名空间为粒度统计各个命名空间内所有容器每秒的磁盘读写IO的次数之和。
磁盘读写吞吐量(读+写)	字节/秒	以命名空间为粒度统计各个命名空间内所有容器每秒的磁盘读写字节量之和。

指标清单

集群视图使用的指标清单如下：

表 9-4 集群视图指标清单

指标	指标类型	说明
kube_pod_container_resource_requests	gauge	容器请求的请求资源数
kube_pod_container_resource_limits	gauge	容器请求的限制资源数
kube_node_status_allocatable	gauge	节点可分配的资源总数
kube_pod_status_phase	gauge	Pod当前阶段
node_memory_MemAvailable_bytes	gauge	节点内存可用字节数
node_memory_MemTotal_bytes	gauge	节点内存总字节数
node_cpu_seconds_total	counter	在不同模式下节点累计CPU花费的时间
kube_node_info	gauge	节点信息
kube_node_status_condition	gauge	节点状态信息
kube_pod_container_status_waiting	gauge	容器是否处在waiting状态
kube_pod_container_status_terminated	gauge	容器是否处在终止状态
container_cpu_usage_seconds_total	counter	容器CPU累计使用时间
container_memory_rss	gauge	RSS内存，即常驻内存集。是分配给进程使用的实际物理内存字节数，不是磁盘上缓存的虚机内存。

指标	指标类型	说明
container_network_receive_bytes_total	counter	容器网络累积接收字节数
container_network_transmit_bytes_total	counter	容器网络累积传输字节数
container_network_receive_packets_total	counter	容器网络收到的累计数据包数
container_network_transmit_packets_total	counter	容器网络传输的累计数据包数
container_network_receive_packets_dropped_total	counter	容器网络接收时丢失的数据包数
container_network_transmit_packets_dropped_total	counter	容器网络传输时丢失的数据包数
container_fs_reads_total	counter	容器磁盘读取次数
container_fs_reads_bytes_total	counter	容器磁盘读取的总字节数

9.3.9.3 APIServer 视图

提供了Kubernetes核心组件APIServer主要监控视图，帮助您更好的监控APIServer的运行状态。主要包括APIServer组件的请求、资源、工作队列等相关指标。

指标说明

APIServer视图暴露的指标包括请求指标、工作队列指标和资源指标，具体说明如下：

图 9-34 请求指标

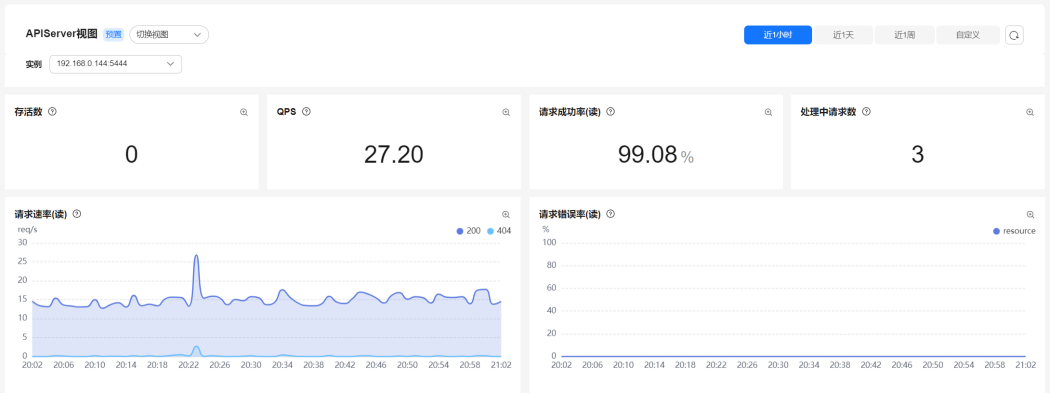


表 9-5 请求指标说明

指标名称	单位	说明
存活数	个	组件存活实例数
QPS	请求数/秒	每秒不同响应码的请求个数
请求成功率(读)	百分比	每秒读请求中响应码为20x的请求比例
处理中请求数	个数	APIServer在处理中的请求个数
请求速率(读)	请求数/秒	每秒不同响应码的读请求个数
请求错误率(读)	百分比	每秒读请求的错误请求比例
请求时延(读)(P99)	毫秒	P99读请求时延
请求速率(写)	请求数/秒	每秒不同响应码的写请求个数
请求错误率(写)	百分比	每秒写请求的错误请求个数
请求时延(写)(P99)	毫秒	P99写请求时延

图 9-35 工作队列指标

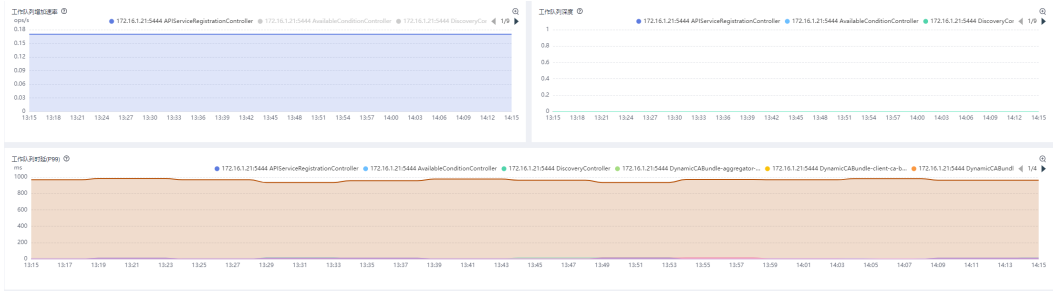


表 9-6 工作队列指标说明

指标名称	单位	说明
工作队列增加速率	操作次数/秒	APIServer每秒工作队列增加的次数
工作队列深度	个	工作队列深度
工作队列时延(P99)	毫秒	APIServer请求P99在工作队列中停留时间

图 9-36 资源指标

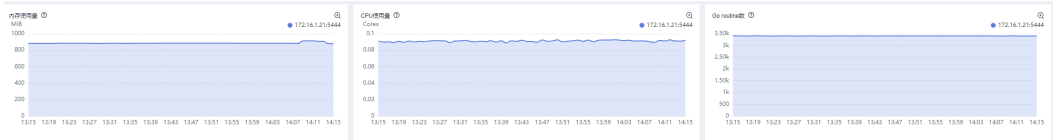


表 9-7 资源指标说明

指标名称	单位	说明
内存使用量	字节	APIServer内存使用量
CPU使用量	Cores	APIServerCPU使用量
Go routine数	次	Go routine次数

指标清单

APIServer视图使用的指标清单如下：

表 9-8 APIServer 视图指标清单

指标	指标类型	说明
up	gauge	组件状态
apiserver_request_total	counter	apiserver请求数按照返回码等维度的计数
go_goroutines	gauge	当前时间goroutines个数
apiserver_current_inflight_requests	gauge	最后一个窗口中，正在处理的请求数量
apiserver_request_duration_seconds_bucket	histogram	APIServer请求延时秒数
workqueue_depth	gauge	当前工作队列深度
workqueue_adds_total	counter	工作队列增加总数
workqueue_queue_duration_seconds_bucket	histogram	请求在工作队列中停留时间
process_resident_memory_bytes	gauge	常驻内存大小
process_cpu_seconds_total	counter	进程CPU总花费时间

9.3.9.4 Pod 视图

从Pod视角呈现Pod维度集群资源、网络、磁盘等监控情况，帮助您详细了解Pod的运行状态。

指标说明

Pod视图暴露的指标包括Pod资源指标、Pod网络指标和Pod磁盘指标，具体说明如下：

图 9-37 Pod 资源指标

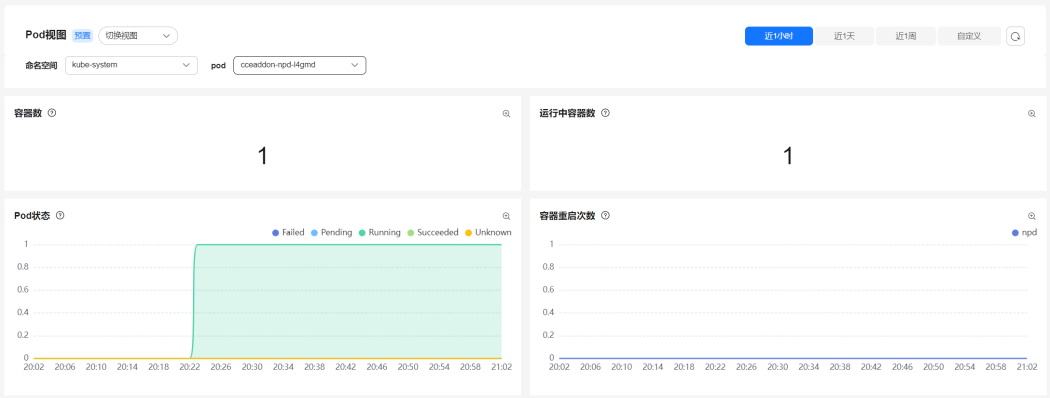


表 9-9 Pod 资源指标说明

指标名称	单位	说明
容器数	个	Pod中的容器总数
运行中容器数	个	Pod中正在运行的容器个数
Pod状态	个	处在不同状态下的Pod个数
容器重启次数	次	容器被重启的次数
CPU使用量	Cores	Pod CPU使用量
CPU 有效率&使用率	百分比	有效率：使用量/请求量；使用率：使用量/总量
内存使用量	字节	内存使用量
内存 有效率&使用率	百分比	有效率：使用量/请求量；使用率：使用量/总量
CPU Throttling	百分比	CPU节流周期限制率

图 9-38 Pod 网络指标

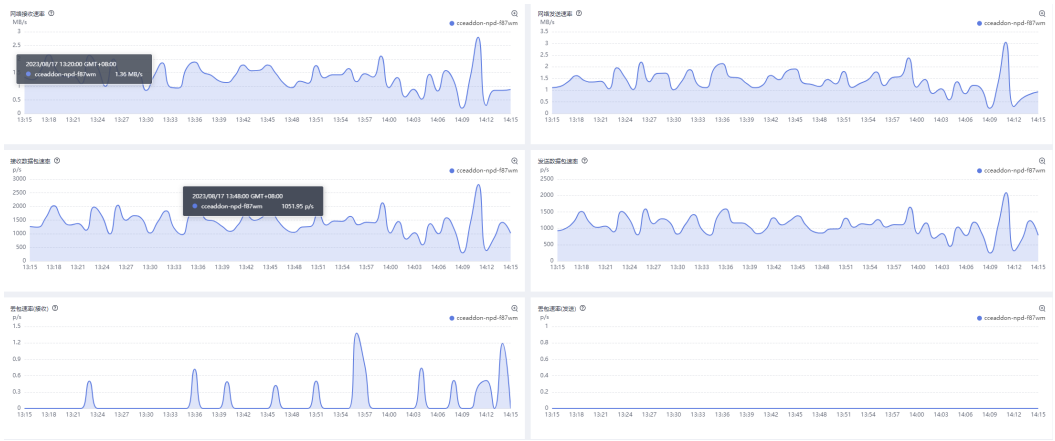


表 9-10 Pod 网络指标说明

指标名称	单位	说明
网络接收速	字节/秒	容器每秒接收的字节数
网络发送速率	字节/秒	容器每秒发送的字节数
接收数据包速率	个/秒	容器每秒接收数据包数
发送数据包速率	个/秒	容器每秒发送数据包数
丢包速率(接收)	个/秒	容器每秒接收的数据丢包数
丢包速率(发送)	个/秒	容器每秒发送的数据丢包数

图 9-39 Pod 磁盘指标

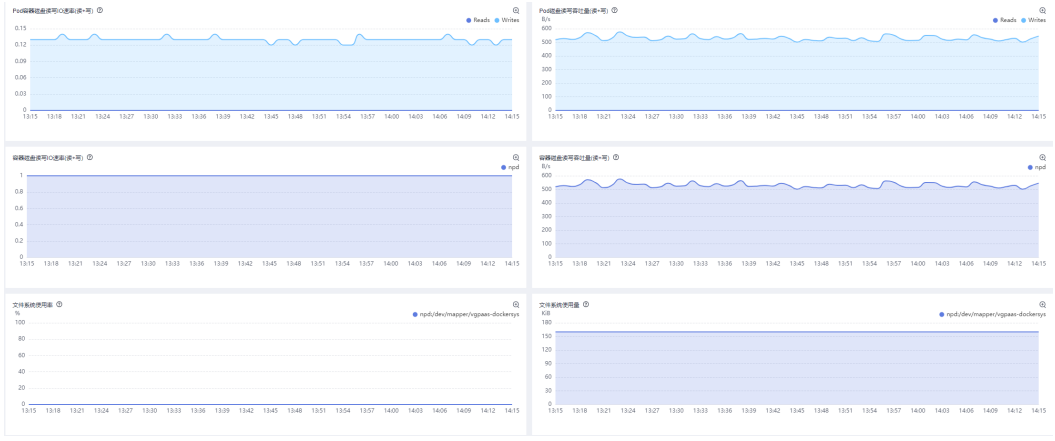


表 9-11 Pod 磁盘指标说明

指标名称	单位	说明
Pod容器磁盘读写IO速率(读+写)	次数/秒	Pod磁盘每秒读写IO次数
Pod磁盘读写吞吐量(读+写)	字节/秒	Pod磁盘每秒读写字节数
容器磁盘读写IO速率(读+写)	次数/秒	容器磁盘每秒读写IO次数
容器磁盘读写吞吐量(读+写)	字节/秒	容器磁盘每秒读写字节数
文件系统使用率	百分比	文件系统的使用率
文件系统使用量	字节	文件系统已经使用的字节数

指标清单

Pod视图使用的指标清单如下：

表 9-12 Pod 视图指标清单

指标	指标类型	说明
kube_pod_container_status_running	gauge	容器当前是否在运行中的状态
kube_pod_container_info	gauge	Pod中的容器信息
kube_pod_status_phase	gauge	Pod当前的阶段
kube_pod_container_status_restarts_total	counter	容器重启次数
container_cpu_usage_seconds_total	counter	容器CPU累计使用时间
kube_pod_container_resource_requests	gauge	容器请求的请求资源数
container_spec_cpu_quota	gauge	容器的CPU配额
container_memory_working_set_bytes	gauge	容器内存使用量
container_spec_memory_limit_bytes	gauge	容器内存限制量
container_cpu_cfs_throttled_periods_total	counter	容器限制周期间隔数

指标	指标类型	说明
container_cpu_cfs_periods_total	counter	容器经过强制限制的周期间隔数
container_network_receive_bytes_total	counter	容器接收字节的累计计数
container_network_transmit_bytes_total	counter	容器传输字节的累计计数
container_network_receive_packets_total	counter	容器接收数据包的累计计数
container_network_transmit_packets_total	counter	容器传输数据包的累计计数
container_network_receive_packets_dropped_total	counter	容器接收丢失的数据包的累计计数
container_network_transmit_packets_dropped_total	counter	容器传输丢失的数据包的累计计数
container_fs_reads_total	counter	容器已完成磁盘读取的累计计数
container_fs_writes_total	counter	容器已完成磁盘写入的累计计数
container_fs_reads_bytes_total	counter	容器读取的累计字节数
container_fs_writes_bytes_total	counter	容器写入的累计在节数
container_fs_usage_bytes	gauge	文件系统上容器已经使用的字节数
container_fs_limit_bytes	gauge	文件系统上容器限制的字节数

9.3.9.5 主机视图

从主机视角出发，监控主机的资源占用与健康状态，查看主机的磁盘、文件系统等常用系统设备指标，帮助您掌控节点运行状况。

指标说明

主机视图暴露的指标具体说明如下：

图 9-40 主机资源指标

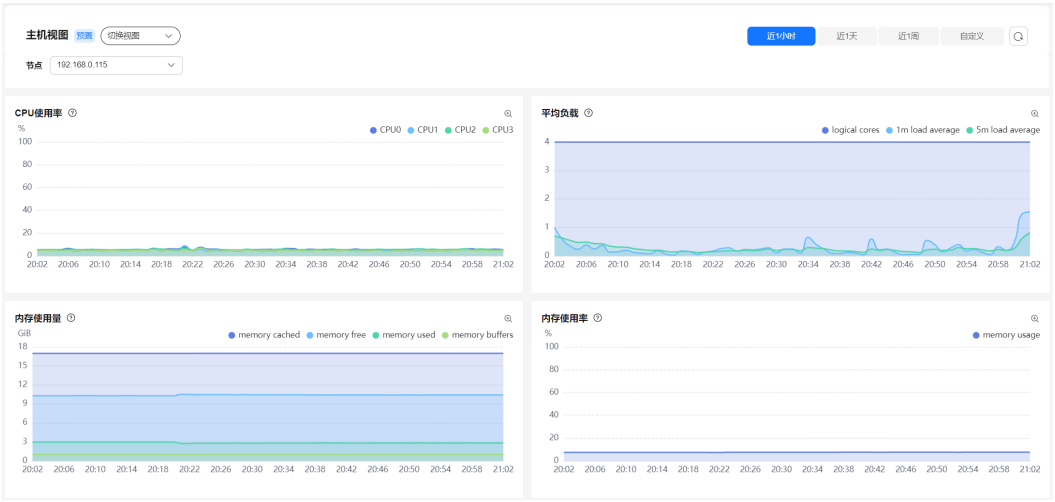


表 9-13 视图说明

图表名称	单位	说明
CPU使用率	百分比	每个CPU核的使用率
平均负载	/	平均负载反映了CPU资源的竞争情况 ● 值小于1时，说明部分CPU资源在处理请求。 ● 值等于1时，说明所有的CPU资源都在处理请求。 ● 值大于1时，说明有部分线程在等待处理。
内存使用量	字节	不同模式的内存使用情况
内存使用率	百分比	主机内存使用率
磁盘写入速率	字节/秒	不同的磁盘的写入速率
磁盘读取速率	字节/秒	不同的磁盘的读取速率
磁盘空间使用	字节	磁盘可用量和已使用量
磁盘空间使用率	百分比	不同设备的磁盘使用率
磁盘IO延迟（秒）	秒	磁盘IO延迟
TCP连接	个	TCP连接数 ● alloc：已分配（已建立、已申请到sk_buff）的TCP套接字数量。 ● inuse：正在使用（侦听）的TCP套接字数量。 ● orphan：已分配（已建立、已申请到sk_buff）的TCP套接字数量。 ● tw：等待关闭的TCP连接数。

图表名称	单位	说明
UDP使用情况	个	UDP使用情况 ● liteInuse：正在使用的UDP-Lite套接字数量。 ● inuse：正在使用的UDP套接字数量。 ● useMemory：UPD缓冲区使用量。
最大文件描述符	EB：10的18次方	最大文件描述符数
已使用文件描述符	个	当前已分配使用的文件描述符数量
Socket使用情况	个	Socket使用情况 ● socketsUsed：使用的所有协议套接字总量。 ● fragInuse：正在使用的Frag套接字数量。 ● fragMemroy：Frag缓冲区使用量。 ● rawInuse：正在使用的Raw套接字数量。
文件系统异常	/	文件系统状态 ● readonly：文件系统只读 ● deviceError：文件系统错误
磁盘读写速率	次/秒	磁盘每秒进行的读写次数
磁盘读和写延迟（秒）	秒	磁盘读写时延
IO队列数	/	磁盘设备平均IO队列长度，节点磁盘IO时间加权值。该值越大，表示节点的磁盘性能越好。
进程状态	个	处于不同状态的进程数量
连接跟踪表条目数	个	● 已分配：连接跟踪表当前已分配的条目数 ● 总容量：连接跟踪表的最大条目数

指标清单

主机视图使用的指标清单如下：

表 9-14 指标说明

指标名称	类型	说明
node_cpu_seconds_total	Counter	节点不同模式下花费的CPU秒

指标名称	类型	说明
node_load1	Gauge	1分钟内CPU平均负载，反映了CPU资源的竞争情况。 ● 值小于1时，说明部分CPU资源在处理请求。 ● 值等于1时，说明所有的CPU资源都在处理请求。 ● 值大于1时，说明有部分线程在等待处理。
node_load15	Gauge	15分钟内CPU平均负载
node_memory_MemTotal_bytes	Gauge	节点内存总量
node_memory_MemAvailable_bytes	Gauge	节点可用内存量
node_disk_written_bytes_total	Gauge	节点磁盘写入量
node_disk_read_bytes_total	Gauge	节点磁盘读取量
node_filesystem_size_bytes	Gauge	节点文件系统大小
node_filesystem_avail_bytes	Gauge	节点可用文件系统大小
node_disk_io_time_seconds_total	Counter	I/O操作所花费的总秒数
node_sockstat_TCP_alloc	Gauge	已分配的TCP套接字数
node_sockstat_UDPLITE_inuse	Gauge	已使用的UDPLITE套接字
node_filefd_maximum	Gauge	文件描述符统计信息：最大值。
node_filefd_allocated	Gauge	文件描述符统计信息：已分配。
node_sockstat_sockets_used	Gauge	已使用的IPv4套接字数
node_filesystem_readonly	Gauge	文件系统只读状态
node_disk_reads_completed_total	Counter	磁盘读取完成的次数
node_disk_read_time_seconds_total	Counter	磁盘读取完成的次数的总耗时

指标名称	类型	说明
node_disk_io_time_weighted_seconds_total	Counter	节点磁盘IO时间加权值。该值越大，表示节点的磁盘性能越好。
node_procs_blocked	Gauge	等待I/O完成的阻塞进程数
node_nf_conntrack_entries	Gauge	连接跟踪表的最大大小

9.3.9.6 Node 视图

从节点视角出发，加入了节点资源、网络、磁盘等关键指标呈现，帮助您掌控节点运行状况。

指标说明

Node视图暴露的指标如下：

图 9-41 Node 资源指标

表 9-15 Node 资源指标说明

指标名称	单位	说明
节点CPU使用率	百分比	节点CPU使用率
CPURequests水位	百分比	节点CPU Requests占节点CPU容量的百分比
CPULimits水位	百分比	节点CPU Limits占节点CPU容量的百分比
内存使用率	百分比	节点内存使用率
内存Requests水位	百分比	节点内存Requests占节点内存容量的百分比
内存Limits水位	百分比	节点内存Limits占节点内存容量的百分比
PodCPU使用率	百分比	节点上Pod的CPU使用率

指标名称	单位	说明
内存使用量	字节	节点上Pod的内存使用量

图 9-42 节点网络指标

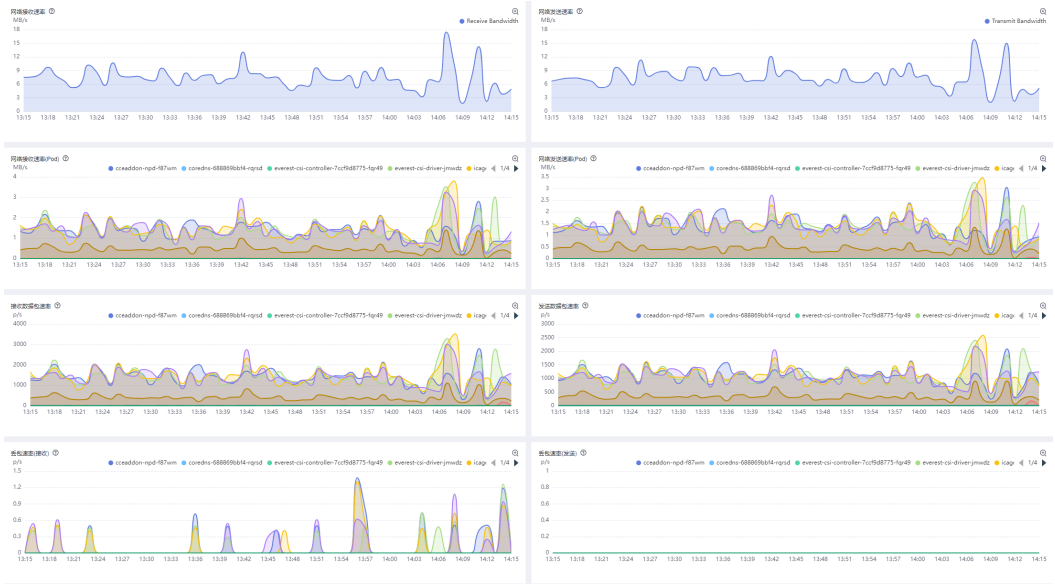


表 9-16 节点网络指标说明

指标名称	单位	说明
网络接收速率	字节/秒	节点每秒接收的字节数
网络发送速率	字节/秒	节点每秒发送的字节数
网络接收速率(Pod)	字节/秒	节点上的Pod每秒接收的字节数
网络发送速率(Pod)	字节/秒	节点上的Pod每秒发送的字节数
接收数据包速率	个/秒	节点上的Pod每秒接收的数据包个数
发送数据包速率	个/秒	节点上的Pod每秒发送的数据包个数
丢包速率(接收)	个/秒	节点上的Pod每秒接收丢失的数据包个数
丢包速率(发送)	个/秒	节点上的Pod每秒发送丢失的数据包个数

图 9-43 节点磁盘指标

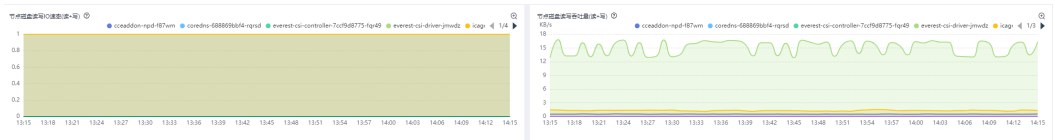


表 9-17 节点磁盘指标说明

指标名称	单位	说明
节点磁盘读写IO速率(读+写)	次数/秒	节点磁盘每秒读写IO次数
节点磁盘读写吞吐量(读+写)	字节/秒	节点中Pod每秒读写磁盘字节数

指标清单

节点视图使用的指标清单如下：

表 9-18 节点指标清单

指标	类型	说明
kube_pod_container_resource_limits	gauge	容器请求的限制资源数
kube_pod_status_phase	gauge	Pod当前阶段
kube_node_status_allocatable	gauge	节点可分配的资源总数
kube_pod_container_resource_requests	gauge	容器请求的请求资源数
node_memory_MemAvailable_bytes	gauge	节点内存可用字节数
node_memory_MemTotal_bytes	gauge	节点内存总字节数
node_cpu_seconds_total	counter	在不同模式下节点累计CPU花费的时间
container_cpu_usage_seconds_total	counter	容器CPU累计使用时间
container_memory_rss	gauge	RSS内存，即常驻内存集。是分配给进程使用的实际物理内存字节数，不是磁盘上缓存的虚机内存。
container_network_receive_bytes_total	counter	容器网络累积接收字节数
container_network_transmit_bytes_total	counter	容器网络累积传输字节数
container_network_receive_packets_total	counter	容器网络收到的累计数据包数
container_network_transmit_packets_total	counter	容器网络传输的累计数据包数

指标	类型	说明
container_network_transmit_packets_dropped_total	counter	容器网络传输时丢失的数据包数
container_fs_reads_total	counter	容器磁盘读取次数
container_fs_writes_total	counter	容器已完成磁盘写入的累计计数
container_fs_reads_bytes_total	counter	容器磁盘读取的总字节数
container_fs_writes_bytes_total	counter	容器写入的累计在节数

9.3.9.7 节点池视图

从节点池视角呈现节点池资源的占用和分配情况，帮助您详细了解节点池的负载状态。

指标说明

节点池视图暴露的指标如下：

图 9-44 节点池资源指标

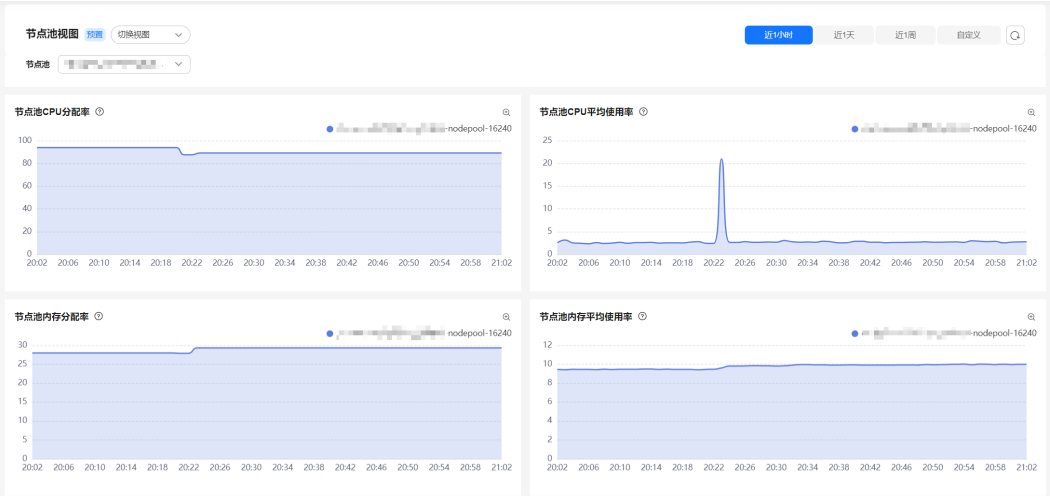


表 9-19 视图说明

图表名称	单位	说明
节点池CPU分配率	百分比	节点池里的所有节点的Pod CPU Request总量占所有节点CPU总量的比例
节点池CPU使用率	百分比	节点池里的所有节点CPU使用量占总量的比例

图表名称	单位	说明
节点池内存分配率	百分比	节点池里的所有节点的Pod 内存Request总量占所有节点内存总量的比例
节点池内存使用率	百分比	节点池里的所有节点内存使用量占总量的比例
节点数量趋势	个	节点池里的节点数量

指标清单

节点池视图使用的指标清单如下：

表 9-20 指标说明

指标名称	单位	说明
kube_node_labels	Gauge	节点标签，其中label_cce_cloud_com_cce_nodepool为CCE节点池名称，若无该标签值则为Default Pool。
node_cpu_seconds_total	Counter	节点不同模式下花费的CPU秒
node_memory_MemAvailable_bytes	Gauge	节点的可用内存量
node_memory_MemTotal_bytes	Gauge	节点的内存总量
kube_pod_container_resource_requests	Gauge	Pod 容器的资源申请量

9.3.9.8 GPU 视图

GPU资源指标可以衡量GPU性能和使用情况，包括GPU的利用率、温度、显存等方面的监控数据，帮助您掌控GPU运行状况。

指标说明

图 9-45 GPU 资源指标

表 9-21 GPU 图表说明

图表名称	单位	说明
集群-显存使用率	百分比	集群的显存使用率 计算公式：集群内容器显存使用总量/集群内显存总量
集群-算力使用率	百分比	集群的算力使用率 计算公式：集群内容器算力使用总量/集群内算力总量
节点-显存使用量	字节	每个节点的显存使用量
节点-算力使用率	百分比	每个节点的算力使用率 计算公式：节点上容器算力使用总量/节点上算力总量
节点-显存使用率	百分比	每个节点的显存使用率 计算公式：节点上容器显存使用总量/节点上显存总量
GPU卡-显存使用量	字节	显卡上容器显存使用总量
GPU卡-算力使用率	百分比	每张GPU卡的算力使用率 计算公式：显卡上容器算力使用总量/显卡的算力总量
GPU卡-温度	摄氏度	每张GPU卡的温度
GPU-显存频率	赫兹	每张GPU卡的显存频率
GPU卡-PCIe带宽	字节/秒	每张GPU卡的PCIe带宽

指标清单

GPU视图使用的指标清单如下：

表 9-22 GPU 指标说明

指标名称	类型	说明
cce_gpu_gpu_utilization	Gauge	GPU卡算力使用率
cce_gpu_memory_utilization	Gauge	GPU卡显存使用率
cce_gpu_memory_used	Gauge	GPU显存使用量
cce_gpu_memory_total	Gauge	GPU显存总量
cce_gpu_memory_free	Gauge	GPU显存空闲量
cce_gpu_memory_clock	Gauge	GPU显存频率

指标名称	类型	说明
cce_gpu_gpu_temperature	Gauge	GPU温度
cce_gpu_pcie_link_bandwidth	Gauge	GPU pcie带宽
cce_gpu_pcie_throughput_rx	Gauge	GPU pcie接收带宽

9.3.9.9 XGPU 视图

XGPU是虚拟化的GPU设备，从XGPU视图可以在节点、GPU卡、容器等多个角度监控XGPU虚拟化设备的显存、算力分配率，帮助您掌控GPU运行状况。

指标说明

图 9-46 XGPU 资源指标

表 9-23 XGPU 视图图表说明

图表名称	单位	说明
集群-XGPU设备显存使用率	百分比	集群的GPU虚拟化设备显存使用率 计算公式：集群中所有XGPU设备的显存使用量之和 / 集群显存总量
集群-XGPU设备算力使用率	百分比	集群的GPU虚拟化设备算力使用率 计算公式：集群中所有XGPU设备的算力使用量之和 / 集群算力总量

图表名称	单位	说明
节点-XGPU设备显存使用率	百分比	每个节点的GPU虚拟化设备显存使用率 计算公式：节点上所有XGPU设备的显存使用量之和 / 节点显存总量
节点-XGPU设备算力使用率	百分比	每个节点的GPU虚拟化设备算力使用率 计算公式：节点上所有XGPU设备的算力使用量之和 / 节点算力总量
节点-XGPU设备数量	个	每个节点的GPU虚拟化设备数量
节点-XGPU设备显存分配量	字节	每个节点上的GPU虚拟化设备显存总量
GPU卡-XGPU设备显存使用率	百分比	每张GPU卡上的GPU虚拟化设备显存使用率 计算公式：显卡上所有XGPU设备的显存使用量之和 / 显卡显存总量
GPU卡-XGPU设备显存分配量	字节	每张GPU卡上的GPU虚拟化设备的显存总量
GPU卡-XGPU设备显存分配率	百分比	每张GPU卡上的GPU虚拟化设备的显存总量占这张GPU卡显存总量的比例 计算公式：显卡上所有XGPU设备能使用的显存上限之和 / 显卡显存总量
GPU卡-XGPU设备算力使用率	百分比	每张GPU卡的GPU虚拟化设备的算力使用率 计算公式：显卡上所有XGPU设备当前所使用的算力之和 / 显卡算力总量
GPU卡-XGPU设备数量	个	每张GPU卡的GPU虚拟设备的数量
GPU卡-调度策略	数字	• 0为显存隔离算力共享模式 • 1为显存算力隔离模式 • 2为默认模式表示当前卡还没被用于XGPU设备分配
GPU卡-不健康的XGPU设备数量	个	每张GPU卡的不健康的GPU虚拟化设备的数量
容器显存分配量	字节	容器所能使用的显存上限
容器算力使用率	百分比	每个容器的算力使用率 计算公式：XGPU设备上容器算力使用量 / XGPU设备算力总量
容器显存使用量	字节	每个容器的显存使用量
容器显存使用率	百分比	每个容器的显存使用率 计算公式：XGPU设备上容器显存使用量 / XGPU设备显存总量

指标清单

XGPU视图使用的指标清单如下：

表 9-24 XGPU 指标说明

指标名称	类型	说明
xgpu_memory_total	Gauge	XGPU显存总量
xgpu_memory_used	Gauge	XGPU显存使用量
xgpu_core_percentage_total	Gauge	XGPU算力总量
xgpu_core_percentage_used	Gauge	XGPU算力使用率
gpu_schedule_policy	Gauge	GPU模式分成0、1、2三种，具体说明如下： ● 0为显存隔离算力共享模式 ● 1为显存算力隔离模式 ● 2为默认模式表示当前卡还没被用于XGPU设备分配
xgpu_device_health	Gauge	XGPU设备的健康情况。当前虚拟化域侧并没有提供特定的接口来检查XGPU的健康情况，所以根据XGPU设备所在物理GPU设备的健康情况反推。0表示XGPU设备为健康状态，1表示为非健康状态。

9.3.9.10 CoreDNS 视图

提供了负载域名解析的CoreDNS监控视图，包含请求、响应情况，以及缓存状况。

指标说明

CoreDNS视图暴露的指标如下：

图 9-47 CoreDNS 视图指标

表 9-25 CoreDNS 指标说明

指标名称	单位	说明
请求速率	个/秒	CoreDNS每秒请求个数
请求速率(记录类型)	个/秒	根据请求类型统计CoreDNS的请求速率
请求速率(区域)	个/秒	根据区域统计CoreDNS的请求速率
请求速率(DO标志位)	个/秒	设置了DO标志位的请求速率
请求数据包(UDP)	字节数	基于UDP协议的请求数据包的P99、P90、P50的大小
请求数据包(TCP)	字节数	基于TCP协议的请求数据包的P99、P90、P50的大小
响应速率(响应状态码)	个/秒	CoreDNS不同状态码每秒的请求数
响应时延	毫秒	CoreDNS P99、P90、P50的请求时延
响应数据包(UDP)	字节数	基于UDP协议的响应数据包的P99、P90、P50的大小
响应数据包(TCP)	字节数	基于TCP协议的响应数据包的P99、P90、P50的大小
缓存记录数	个	CoreDNS缓存的DNS记录数
缓存命中率	个/秒	CoreDNS缓存每秒的命中请求数

指标清单

CoreDNS视图使用的指标清单如下：

表 9-26 CoreDNS 视图指标清单

指标	类型	说明
coredns_dns_request_count_total	counter	记录所有请求查询的累计值
coredns_dns_requests_total	counter	DNS请求的总数
coredns_dns_request_type_count_total	counter	每种类型的请求累计值
coredns_dns_request_do_count_total	counter	设置了DO标志位的请求次数累计值
coredns_dns_do_requests_total	counter	设置了DO标志位的请求总数
coredns_dns_request_size_bytes_bucket	histogram	CoreDNS请求字节数
coredns_dns_response_rcode_count_total	counter	不同返回码个数的累计值
coredns_dns_responses_total	counter	返回码的总数
coredns_dns_request_duration_seconds_bucket	histogram	CoreDNS请求时延
coredns_dns_response_size_bytes_bucket	histogram	CoreDNS返回字节数
coredns_cache_size	gauge	CoreDNS缓存大小
coredns_cache_hits_total	counter	CoreDNS缓存命中个数

9.3.9.11 PVC 视图

提供了集群中的PVC监控视图，包含PV/PVC的状态、使用率情况。

说明

支持以下PVC类型监控：

- 云硬盘类型的PVC（要求volumeMode参数值为Filesystem）支持使用量监控。
- 本地持久卷类型的PVC（要求集群中安装的Everest版本大于等于2.4.41）支持使用量监控。
- 极速文件存储类型的PVC支持使用量监控（包括子目录场景，但子目录PVC采集到的使用量和容量与SFS Turbo实例的使用量和容量一致）。
- 挂载到普通容器的PVC支持采集使用量及inodes数据，挂载至安全容器PVC不支持。

指标说明

PVC视图暴露的指标如下：

表 9-27 PVC 图表说明

指标名称	单位	说明
PV状态	/	PV当前所处的状态，包含Available、Bound、Failed、Pending、Released。
PVC状态	/	PVC当前所处的状态，包含Bound、Lost、Pending。
PVC使用量	GiB	PVC已使用量及空闲量
PVC使用率	百分比	PVC已使用的空间占比
PVC inodes	个	PVC inodes已使用及空闲量
PVC inodes使用率	百分比	PVC已使用的inodes占比

指标清单

PVC视图使用的指标清单如下：

表 9-28 PVC 指标说明

指标	类型	描述
kubelet_volume_stats_inodes_used	Gauge	卷中已使用的inode数
kubelet_volume_stats_inodes	Gauge	卷中的最大inode数
kubelet_volume_stats_capacity_bytes	Gauge	卷的容量（以字节为单位）
kubelet_volume_stats_available_bytes	Gauge	卷中可用字节数
kubelet_volume_stats_used_bytes	Gauge	卷中已使用的字节数
kube_persistentvolume_status_phase	Gauge	PV的状态
kube_persistentvolumeclaim_status_phase	Gauge	PVC的状态

9.3.9.12 Kubelet 视图

Kubelet是运行在集群中每个节点上的代理程序，它提供了一些指标可以更好地了解集群的运行状态。

指标说明

Kubelet视图暴露的指标如下：

表 9-29 Kubelet 图表说明

视图名称	单位	说明
运行中Kubelet	个	集群运行中的kubelet的数量
运行中Pod	个	当前Kubelet所在节点上运行中Pod的数量
运行中容器	个	当前Kubelet所在节点上运行中容器的数量
实际卷数量	个	当前Kubelet所在节点的实际卷数量
期望卷数量	个	当前Kubelet所在节点的期望卷数量
配置错误数量	个	当前Kubelet所在节点的Kubelet配置错误数量
操作速率	次/秒	Kubelet每秒执行的操作的次数
操作错误率	次/秒	Kubelet每秒执行的操作失败的次数
操作时延	秒	Kubelet的不同操作的操作时延
Pod启动速率	次/秒	Kubelet每秒执行了pod start的次数
Pod启动时延（99分位）	秒	Kubelet执行pod start操作中99%的操作的时延分布情况
存储操作速率	次/秒	Kubelet每秒执行的存储相关操作的次数
存储操作错误率	次/秒	Kubelet每秒执行的存储相关操作失败的次数
存储操作时延（99分位）	秒	Kubelet执行存储操作中99%的操作的时延分布情况
控制组管理器操作速率	次/秒	每秒执行销毁或更新操作的次数
控制组管理器操作时延（99分位）	秒	Kubelet执行销毁或更新操作中99%的操作的时延分布情况
PLEG relist速率	次/秒	Kubelet PLEG每秒执行relist的次数
PLEG relist间隔（99分位）	秒	Kubelet PLEG relist中99%的操作的间隔分布情况
PLEG relist时延（99分位）	秒	Kubelet PLEG relist中99%的操作的时延分布情况
RPC速率	次/秒	不同状态响应码的RPC请求的次数

视图名称	单位	说明
请求时延（99分位）	秒	不同method的请求的99%的时延分布情况
内存使用量	字节	Kubelet的内存使用量
CPU使用量	字节	Kubelet的CPU使用量
Go routine数	个	Go协程数量

指标清单

Kubelet视图使用的指标清单如下：

表 9-30 Kubelet 指标说明

指标	类型	说明
storage_operation_errors_total	Counter	存储操作期间发生的错误次数
storage_operation_duration_seconds_count	Counter	存储操作的操作次数
storage_operation_duration_seconds_bucket	Histogram	存储操作的持续时间
kubelet_pod_start_duration_seconds_count	Counter	进行过pod start的数量
kubelet_pod_start_duration_seconds_bucket	Histogram	pod start的耗时分布情况
kubelet_runtime_operations_duration_seconds_bucket	Histogram	不同操作的累计操作耗时分布情况
kubelet_runtime_operations_errors_total	Counter	不同操作的累计操作失败的数量
kubelet_node_config_error	Gauge	如果节点遇到与配置相关的错误，则此指标为true（1），否则为false（0）
volume_manager_total_volumes	Gauge	Volume Manager中的卷数
kubelet_running_containers	Gauge	当前运行的Containers数
kubelet_running_pods	Gauge	当前运行的pod数
kubelet_node_name	Gauge	节点名称，值始终为1
kubelet_runtime_operations_total	Counter	运行过程中不同的操作类型的累计操作次数

指标	类型	说明
kubelet_cgroup_manager_duration_seconds_count	Counter	销毁和更新的数量
kubelet_cgroup_manager_duration_seconds_bucket	Histogram	销毁和更新操作的耗时分布情况
kubelet_pleg_relist_duration_seconds_count	Counter	PLEG relist pod不同耗时的数量
kubelet_pleg_relist_interval_seconds_bucket	Histogram	PLEG relist 间隔的分布情况
kubelet_pleg_relist_duration_seconds_bucket	Histogram	PLEG relist pod耗时的分布情况
rest_client_requests_total	Counter	请求apiserver的总次数（按照返回码code和请求类型method统计）
rest_client_request_duration_seconds_bucket	Histogram	请求apiserver的总次数（按照返回码code和请求类型method统计）的分布情况
process_resident_memory_bytes	Gauge	进程驻留内存大小（以字节为单位）
process_cpu_seconds_total	Counter	进程用户和系统 CPU 总时间（以秒为单位）
go_goroutines	Gauge	协程数量

9.3.9.13 Prometheus Server 视图

Prometheus本地数据存储模式可以收集有关主机和应用程序的指标数据并存储在集群中，监控数据可以选择上报并存储到AOM或三方监控平台。Prometheus Server视图展示了Prometheus提供的一些内置指标，可用于监控和度量系统的性能和状态。

指标说明

Prometheus Server视图暴露的指标如下：

图 9-48 Prometheus Server 资源指标

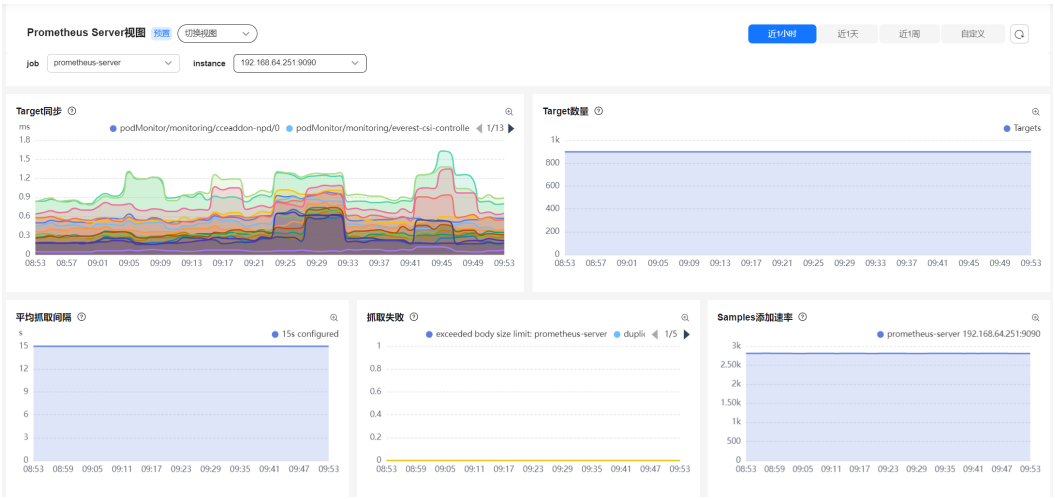


表 9-31 Prometheus Server 图表说明

视图指标	单位	描述
Target同步	秒	target指标采集时延
Target数量	个	target采集的指标总数
平均抓取间隔	秒	每一次指标采集的时间间隔
抓取失败	次	采集失败的次数
Samples添加速率	个	head 添加Samples的速率
Head中Series数量	个	head中Series数量
Head块数量	个	head块数量
查询速率	次/秒	每秒执行普罗query的次数
P90查询耗时	秒	不同分片的90%的操作的查询耗时
远端样本滞后比率	秒	存储在WAL中的样本的最高时间戳与远程写入成功的最高时间戳的比率
远程写流量	字节/秒	远程写入的速率
当前队列数	个	当前用于并行发送到远程存储的分片数
最大队列数	个	可用于并行发送到远程存储的分片数的最大值
最小队列数	个	可用于并行发送到远程存储的分片数的最小值
期望队列数	个	分片队列期望基于输入样本和输出样本的比率运行的分片数

视图指标	单位	描述
队列容量	个	用于并行发送到远程存储的队列每个分片的容量
挂起中的样本数	个	用于并行发送到远程存储的队列中每个分片的容量
TSDB当前段	段	TSDB当前正在写入的WAL段索引
远程写入当前段	段	WAL watcher正在从中读取记录的当前段
样本丢弃率	次/秒	在通过远程写入发送之前，从WAL读取后丢弃的样本速率。
样本失败率	次/秒	发送到远程存储时失败的样本失败速率，不可恢复错误。
样本重试率	次/秒	在发送到远程存储时失败但由于发送错误可恢复而重试的样本速率
入队失败重试率	次/秒	由于分片队列已满而入队失败重试速率

指标清单

Prometheus Server视图使用的指标清单如下：

表 9-32 Prometheus Server 指标说明

指标	类型	描述
prometheus_target_sync_length_seconds_sum	Summary	不同的target的采集时延
prometheus_sd_discovered_targets	Gauge	不同的target采集的指标数
prometheus_target_interval_length_seconds_sum	Summary	指标采集间隔
prometheus_target_scrapes_exceeded_body_size_limit_total	Counter	采集失败的次数
prometheus_tsdb_head_samples_appended_total	Counter	head中添加的samples的总数
prometheus_tsdb_head_series	Gauge	head中保存的series数量
prometheus_tsdb_head_chunks	Gauge	head中存放的chunk数量
prometheus_engine_query_duration_seconds_count	Counter	查询普罗query的次数

指标	类型	描述
prometheus_engine_query_duration_seconds	Counter	不同分片的响应时间耗时
prometheus_remote_storage_highest_timestamp_in_seconds	Gauge	远程存储里最新的时间戳
prometheus_remote_storage_queue_highest_sent_timestamp_seconds	Gauge	普罗分片里最新的时间戳
prometheus_remote_storage_bytes_total	Counter	压缩后队列发送的数据（非元数据）的总字节数
prometheus_remote_storage_shards	Gauge	当前用于并行发送到远程存储的分片数
prometheus_remote_storage_shards_max	Gauge	可用于并行发送到远程存储的分片数的最大值
prometheus_remote_storage_shards_min	Gauge	可用于并行发送到远程存储的分片数的最小值
prometheus_remote_storage_shards_desired	Gauge	分片队列期望基于输入样本和输出样本的比率运行的分片数
prometheus_remote_storage_shard_capacity	Gauge	用于并行发送到远程存储的队列中每个分片的容量
prometheus_remote_storage_pending_samples	Gauge	要发送到远程存储的队列分片中挂起的样本数
prometheus_tsdb_wal_segment_current	Gauge	TSDB当前正在写入的WAL段索引
prometheus_wal_watcher_current_segment	Gauge	WAL正在从中读取记录的当前段
prometheus_remote_storage_dropped_samples_total	Gauge	在通过远程写入发送之前，从WAL读取后丢弃的样本速率。
prometheus_remote_storage_failed_samples_total	Gauge	发送到远程存储时失败的样本失败速率，不可恢复的错误的次数。
prometheus_remote_storage_retried_samples_total	Gauge	在发送到远程存储时失败但由于发送错误可恢复而重试的次数
prometheus_remote_storage_enqueue_retries_total	Gauge	由于分片队列已满而入队失败重试的次数

9.3.9.14 Prometheus Agent 视图

Prometheus Agent是轻量化的容器监控模式，可以收集有关主机和应用程序的指标数据，并将数据上报并存储到AOM或三方监控平台。Prometheus Agent视图展示了Prometheus提供的一些内置指标，可用于监控和度量系统的性能和状态。

指标说明

Prometheus Agent视图暴露的指标如下：

图 9-49 Prometheus Agent 资源指标

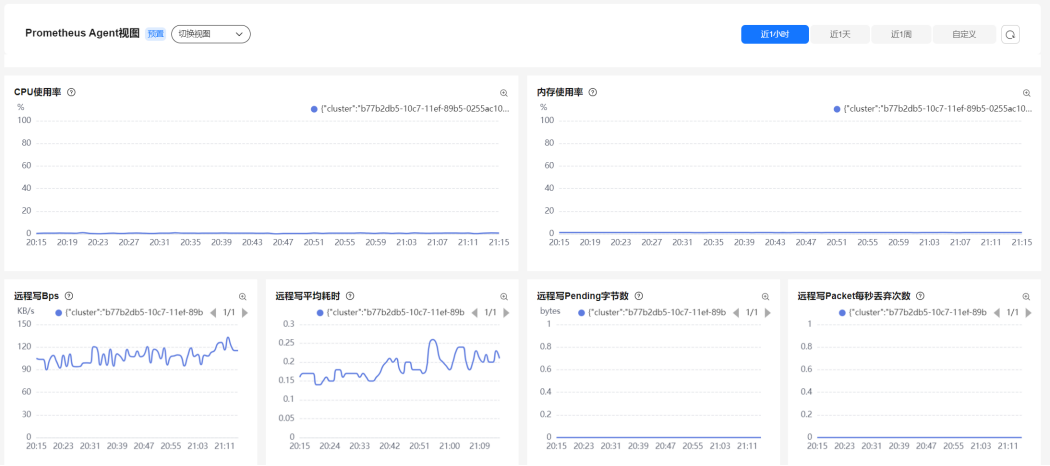


表 9-33 Prometheus Agent 视图说明

视图指标	单位	描述
CPU使用率	百分比	Prometheus Agent Pod CPU平均使用率
内存使用率	百分比	Prometheus Agent Pod 内存平均使用率
远程写Bps	字节/秒	每秒远程写入的字节数
远程写平均耗时	秒	远程写入平均耗时
远程写Pending字节数	字节	远程写入挂起的数据字节数
远程写Packet每秒丢弃次数	次	远程写入每秒丢弃的数据包数
远程写每秒错误请求次数	次	远程写每秒错误请求次数
远程写错误请求百分比	百分比	远程写错误请求百分比
远程写每秒重试次数	次	远程写每秒重试次数
采集Scrapers数量	个	采集Scrapers数量
每秒采集次数	次	每秒采集次数

视图指标	单位	描述
采集平均耗时	秒	采集平均耗时
每秒采集读取错误次数	次	每秒采集读取错误次数
每秒采集写入错误次数	次	每秒采集写入错误次数
每秒采集大小超限次数	次	每秒采集大小超限次数
发送队列读取Bps	字节/秒	发送队列每秒读的字节数
发送队列写Bps	字节/秒	发送队列每秒写的字节数
发送队列Pending大小	字节	发送队列挂起的数据字节数
每秒Block读取次数	次	发送队列每秒读block的次数
每秒Block写入次数	次	发送队列每秒写block的次数
每秒Block丢弃次数	次	发送队列每秒block丢弃的次数

指标清单

Prometheus Agent视图使用的指标清单如下：

表 9-34 Prometheus Agent 指标说明

指标名称	类型	说明
container_cpu_usage_seconds_total	Gauge	容器CPU使用量
container_memory_working_set_bytes	Gauge	容器工作内存使用量
vmagent_remotewrite_bytes_sent_total	Counter	Prometheus Agent远程写入字节数发送总计
vmagent_remotewrite_duration_seconds_sum	Summary	Prometheus Agent远程写入耗时
vmagent_remotewrite_pending_data_bytes	Gauge	Prometheus Agent远程写入挂起数据字节数
vmagent_remotewrite_packets_dropped_total	Counter	Prometheus Agent远程写入数据包丢弃总数
vmagent_remotewrite_requests_total	Counter	Prometheus Agent远程写入请求总数
vmagent_remotewrite_retries_count_total	Counter	Prometheus Agent远程写入重试次数总数
vm_promscrape_active_scrapers	Gauge	采集的分片数量

指标名称	类型	说明
vm_promscrape_scrapes_total	Counter	采集次数
vm_promscrape_scrape_duration_seconds_sum	Summary	vmagent采集指标的耗时
vm_promscrape_conn_read_errors_total	Counter	采集读取错误次数
vm_promscrape_conn_write_errors_total	Counter	采集写入错误次数
vm_promscrape_max_scrape_size_exceeded_errors_total	Counter	采集大小超过限制的次数
vm_persistentqueue_bytes_read_total	Counter	发送队列读取的总字节数
vm_persistentqueue_bytes_written_total	Counter	发送队列写入的总字节数
vm_persistentqueue_bytes_pending	Gauge	发送队列Pending的字节数
vm_persistentqueue_blocks_read_total	Counter	发送队列读block的次数
vm_persistentqueue_blocks_written_total	Counter	发送队列写block的次数
vm_persistentqueue_blocks_dropped_total	Counter	发送队列block丢弃的次数

9.4 日志中心

9.4.1 日志中心概述

Kubernetes日志可以协助您排查和诊断问题。本文介绍CCE如何通过多种方式进行Kubernetes日志管理。

CCE提供给您多种方式进行Kubernetes日志管理。

- 您可以方便地使用CCE 云原生日志采集插件采集应用日志并上报LTS，从而更好地利用LTS日志服务提供给您的各种日志统计分析等功能。具体操作，请参见**通过云原生日志采集插件采集容器日志**。
- （不推荐）支持收集容器日志到应用运维管理服务AOM。具体操作，请参见**通过ICAgent采集容器日志（不推荐）**。
- 支持收集CCE集群控制平面组件日志和Kubernetes审计日志，将日志从CCE控制层采集到您账号的LTS日志服务的日志流中。具体操作，请参见**采集控制面组件日志**和**采集Kubernetes审计日志**。

- 支持收集CCE集群Kubernetes事件，将Kubernetes事件从CCE集群内采集到您账号的LTS日志服务的日志流中，以便对Kubernetes事件进行持久化存储和统计分析。具体操作，请参见**采集Kubernetes事件**。
- 支持收集NGINX Ingress控制器插件日志，可分析历史流量变化情况，得到业务流量特征，为业务决策提供数据支持。具体操作，请参见**采集NGINX Ingress访问日志**。

ICAgent 和云原生日志采集插件比较

表 9-35 ICAgent 和云原生日志采集插件比较

采集工具	ICAgent		云原生日志采集插件	
日志存储位置	LTS	AOM	LTS	AOM
支持采集内容	容器标准输出 容器内日志文件 节点日志文件 Kubernetes事件	容器标准输出 容器内日志文件	容器标准输出 容器内日志文件 节点日志文件 Kubernetes事件	Kubernetes事件
优缺点说明	日志采集策略与工作负载分开配置，调整日志采集策略不影响Pod运行。 可以指定采集某个容器的日志。 支持Docker容器引擎和Containerd容器引擎的节点，其中Containerd容器引擎的节点要求ICAgent版本为5.12.130及以上。 容器文件日志采集目前仅支持overlay2存储驱动，不支持Device Mapper作为存储驱动的节点。	每个工作负载需单独配置。 日志采集策略与Pod配置耦合，修改日志配置会重启Pod。	日志采集策略与工作负载分开配置，调整日志采集策略不影响Pod运行。 可以指定采集某个容器的日志。 采集容器文件日志时，若节点存储模式为Device Mapper模式，路径配置必须为节点数据盘挂载路径。	默认会将上报所有Warning级别事件以及部分Normal级别事件，上报的事件可用于配置告警。

配置方法	在LTS中创建采集策略，详细方法请参见**CCE接入**。	在工作负载中创建采集策略，详细方法请参见**通过ICAgent采集容器日志（不推荐）**	在日志中心创建策略，详细方法请参见**通过云原生日志采集插件采集容器日志**	详细方法请参见**Kubernetes事件上报应用运维管理（AOM）**
监控目录数	目录递归深度最多5层，最大不超过1000个文件。		支持最多3层模糊匹配目录。	-
监控文件数	• 每个通过卷挂载日志的路径下，ICAgent最多采集20个日志文件。 • 每个ICAgent最多采集1000个容器标准输出日志文件，容器标准输出日志只支持json-file类型。		每个节点上，所有日志策略采集的日志文件总数不能超过4096个。	-

9.4.2 收集容器日志

9.4.2.1 通过云原生日志采集插件采集容器日志

云原生日志采集插件是基于开源fluent-bit和opentelemetry构建的云原生日志、Kubernetes事件采集插件。CCE 云原生日志采集插件支持基于CRD的日志采集策略，可以根据您配置的策略规则，对集群中的容器标准输出日志、容器文件日志、节点日志及Kubernetes事件日志进行采集与转发。

约束与限制

- 每个集群最多支持创建50条日志规则。
- 云原生日志采集插件不会采集.gz、.tar、.zip后缀类型的日志文件，且不支持采集日志文件的软链接。
- 采集容器文件日志时，若节点**存储模式**为Device Mapper模式，路径配置必须为节点数据盘挂载路径。
- 若容器运行时为containerd模式，容器标准输出日志中的多行配置暂不生效。(插件1.3.0及以上版本没有该限制)
- 如果业务容器的数据目录是通过数据卷（Volume）挂载的，插件不支持采集它的父目录，需设置采集目录为完整的数据目录。
- 当容器存活时间低于1分钟时，日志无法及时采集，可能存在日志丢失的情况。

费用说明

LTS创建日志组免费，并每月赠送每个账号一定量免费日志采集额度，超过免费额度部分将产生费用（**价格计算器**）。

通过控制台配置日志采集

步骤1 启用云原生日志采集插件日志采集功能

在集群创建时启用云原生日志采集插件日志采集

1. 登录云容器引擎（CCE）控制台。
2. 在控制台上方导航栏，单击“购买集群”。
3. 在“插件选择”页面中，选择“云原生日志采集插件”。
4. 单击右下角“下一步：插件配置”，根据需求勾选以下配置。
   - 容器日志：开启后，将创建名为default-stdout的日志策略，并上报所有命名空间下的标准输出到云日志服务（LTS）。
   - Kubernetes事件：开启后，将创建名为default-event的日志策略，并上报所有命名空间下的Kubernetes事件到云日志服务（LTS）。
5. 配置完成后，单击右下角“下一步：确认配置”，在弹出的窗口中单击“确定”，完成创建。

为已有集群启用CCE 云原生日志采集插件日志采集

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 未进行授权的用户需要先授权，已授权的用户直接跳转下一步。
   在弹出框中单击“确认授权”。

图 9-50 添加授权

3. 页面单击“开启”，等待约30秒后，页面自动跳转。

图 9-51 开启

   - 采集容器标准输出：开启后，将创建名为default-stdout的日志策略，并上报所有命名空间下的标准输出到云日志服务（LTS）。

- 采集Kubernetes事件：开启后，将创建名为default-event的日志策略，并上报所有命名空间下的Kubernetes事件到云日志服务（LTS）。
- 采集插件日志（NGINX Ingress控制器容器标准输出）：需要安装NGINX Ingress控制器插件，并在插件中开启“日志采集”功能。

  开启后，将创建名为default-nginx-ingress的日志策略，并上报所有命名空间下带有采集标签的nginx-ingress容器标准输出到云日志服务（LTS）。

步骤2 查看并配置日志采集策略。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 右上角单击“日志采集策略”，将显示当前集群所有上报LTS的日志策略。

图 9-52 查看日志策略

3. 单击上方“创建日志采集策略”。

   策略模板：若启用日志采集功能时未开启CCE默认提供的日志采集策略，或者删除了默认的日志采集策略，可通过该方式重新创建。

   自定义策略：用于配置自定义日志策略。

图 9-53 自定义策略

说明

- 不同日志类型的日志采集策略，建议选择不同的日志流上报日志，避免日志混乱。
- 容器/节点文件日志路径配置要求如下：
  - 日志目录：请填写绝对路径，如/log。文件路径必须以/ 开头，只能包含大写字母、小写字母、数字或特殊字符-_/*?，且长度不能超过512个字符。
  - 日志文件名：文件名称只能包含大写字母、小写字母、数字或特殊字符-_*?.。且不支持.gz、.tar、.zip后缀类型。

  目录名和文件名支持完整名称和通配符模式，最多有三级目录采用通配符匹配，且第一级目录不能使用通配符。通配符只支持星号（*）和半角问号（?）。星号（*）表示匹配多个任意字符。半角问号（?）表示匹配单个任意字符。例如：
  - 日志路径为/var/logs/*，文件名为*.log，则匹配表达式为/var/logs/*/*.log，表示匹配/var/logs下的所有**一级目录**中后缀名为.log的文件。注意，该表达式无法匹配/var/logs**当前目录**以及/var/logs下**多级目录**中后缀名为.log的文件。
  - 日志路径为 /var/logs/app_*，文件名为*.log，表示/var/logs目录下所有符合app_*格式的目录中后缀名为.log的文件。

表 9-36 自定义策略参数说明

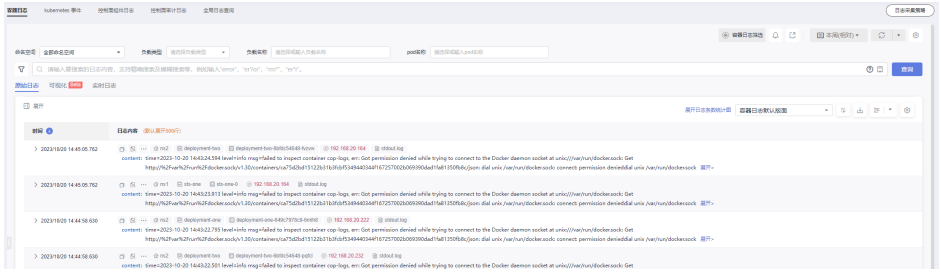
参数	配置说明		
日志类型	**容器标准输出**：用于采集容器标准输出，可以按命名空间、工作负载名称、实例标签配置采集策略。	**容器文件日志**：用于采集容器内的日志，可以指定工作负载或指定实例标签配置采集策略。	**节点文件日志**：用于采集节点上的日志文件，一条日志策略只能配置一个文件路径。
日志源	- **所有容器**：可以指定采集某个命名空间的所有容器，如不指定则采集所有命名空间的容器。 - **指定工作负载**：指定采集哪些工作负载容器的日志，可以指定采集工作负载中具体容器的日志，如不指定则采集所有容器的日志。 - **指定实例标签**：根据标签指定采集哪些工作负载容器的日志，可以指定采集工作负载中具体容器的日志，如不指定则采集所有容器的日志。	- **指定工作负载**：指定采集哪些工作负载容器的日志，可以指定采集工作负载中具体容器的日志，如不指定则采集所有容器的日志。 - **指定实例标签**：根据标签指定采集哪些工作负载容器的日志，可以指定采集工作负载中具体容器的日志，如不指定则采集所有容器的日志。 采集容器文件日志时，还需指定容器中的日志路径，详情请参见**文件日志路径配置要求**。	**路径配置**：用于配置需要采集的日志路径，详情请参见**文件日志路径配置要求**。
日志格式	- **单行文本** 每条日志仅包含一行文本，以换行符 \n 作为各条日志的分界线。 - **多行文本** 有些程序打印的日志存在一条完整的日志数据跨占多行（例如 Java 程序日志）情况，日志采集系统默认是按行采集。如果您想在日志采集系统中按整条显示日志，可以开启多行文本，采用首行正则的方式进行匹配，当选择多行文本时，需填写“日志匹配格式”。 例如，假设采集的日志格式如下，每条日志以日期开头但是占据三行，则可在“日志匹配格式”处填写时间的正则匹配，如\d{4}-\d{2}-\d{2} \d{2}\:\d{2}\:\d{2}.* 在采集日志时，以日期开头三行日志会作为一条完整日志。 2022-01-01 00:00:00 Exception in thread "main" java.lang.RuntimeException: Something has gone wrong, aborting! at com.myproject.module.MyProject.badMethod(MyProject.java:22) at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)		

参数	配置说明
上报到云日志服务(LTS)	用于配置日志上报的日志组和日志流。 – **使用默认日志组/日志流**：将为您自动选择默认日志组（k8s-log-{集群ID}）和默认的日志流（stdout-{集群ID}）。 – **自定义日志组/日志流**：可在下拉框选择任意日志组和日志流。 ■ 日志组：云日志服务进行日志管理的基本单位。如果您未创建日志组，CCE会提示您进行创建，默认名称为k8s-log-{集群ID}，如k8s-log-bb7eaa87-07dd-11ed-ab6c-0255ac1001b3。 ■ 日志流：日志读写的基本单位，日志组中可以创建日志流，将不同类型的日志分类存储，方便对日志进一步分类管理。在安装插件或者根据模板创建日志策略时，会自动创建以下日志流： 容器日志：默认名称为stdout-{集群ID}，如 stdout-bb7eaa87-07dd-11ed-ab6c-0255ac1001b3 k8s事件：默认名称为event-{集群ID}，如 event-bb7eaa87-07dd-11ed-ab6c-0255ac1001b3 NGINX Ingress控制器日志：默认名称为cceaddon-nginx-ingress-{集群ID}，如 cceaddon-nginx-ingress-bb7eaa87-07dd-11ed-ab6c-0255ac1001b3

步骤3 查看日志。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 日志中心下有多个页签，支持不同类型日志查看。
   – 容器日志：显示默认日志组（k8s-log-{集群ID}）下默认日志流（stdout-{集群ID}）中的所有日志数据，支持通过工作负载搜索。

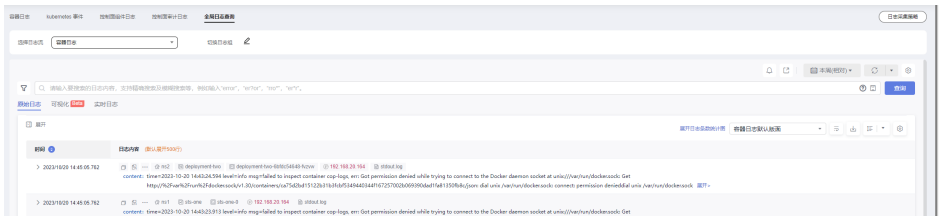
   图 9-54 容器日志查询

   – Kubernetes事件：显示默认日志组（k8s-log-{集群ID}）下默认日志流（event-{集群ID}）中的所有日志数据，用于查询集群产生的Kubernetes事件。
   – 控制面组件日志：显示默认日志组（k8s-log-{集群ID}）下默认日志流（{组件名}-{集群ID}）中的所有日志数据，用于查看集群控制面重要组件的日志信息。
   – 控制面审计日志：显示默认日志组（k8s-log-{集群ID}）下默认日志流audit-{集群ID}）中的所有日志数据，用于查看集群控制面审计日志信息。

- 全局日志查询：支持查看所有日志组日志流下的日志信息。可通过选择日志流查看所选日志流中的日志信息，默认会选择集群默认日志组（k8s-log-{集群ID}），可通过单击切换日志组右侧的图标切换其他日志组。

  图 9-55 全局日志查询

- 插件日志：显示默认日志组（k8s-log-{集群ID}）下的插件上报的日志数据，用于查看集群插件日志信息。

3. 单击右上角“日志采集策略”，单击“查看日志”，可以直接跳转至对应日志策略的日志列表。

   图 9-56 查看日志

----结束

通过 YAML 配置日志采集

须知

云原生日志采集插件版本要求为1.6.1及以上。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 创建名为“**log-config.yaml**”的YAML文件，此处文件名可自定义。

```
vi log-config.yaml
```

YAML配置示例如下，详细字段说明请参见**表9-37**：

- **场景一：采集指定所有工作负载的标准输出**

```
apiVersion: logging.openvessel.io/v1
kind: LogConfig
metadata:
  name: test-log-01  # 规则名称按需修改
  namespace: kube-system  # 采集规则命名空间，固定为kube-system
spec:
  inputDetail:  # 输入配置
    type: container_stdout  # 输入类型，container_stdout表示标准输出
```



```
    containerStdout:  # 标准输出相关配置，仅当type为container_stdout时生效，其他type无需该字段
      allContainers: true  # 是否采集所有容器
      namespaces: []  # 命名空间数组，仅在allContainers为true时生效，采集指定命名空间内的容器标准输出，空数组表示采集所有命名空间容器标准输出
  outputDetail:  # 输出配置
    type: LTS    # 输出类型，固定LTS
    LTS:
      ltsGroupID: abf5f0ad-627e-41cc-8d3f-61c9e1f57f5a      # LTS日志组ID，指定的ID必须存在
      ltsStreamID: f7ed71e9-6b9d-4ba3-86e4-b1b9d22ef4fb     # LTS日志流ID，指定的ID必须存在
```

- **场景二：采集指定工作负载的容器内日志**

```
apiVersion: logging.openvessel.io/v1
kind: LogConfig
metadata:
  name: test-log-02  # 规则名称按需修改
  namespace: kube-system  # 采集规则命名空间，固定为kube-system
spec:
  inputDetail:  # 输入配置
    type: container_file   # 输入类型，container_file表示容器内日志文件
    containerFile:    # 容器内文件日志相关配置，仅当type为container_file时生效，其他type无需该字段
      workloads:        # 工作负载信息需要按照实际情况修改
      - namespace: monitoring  # 工作负载所属命名空间
        kind: Deployment  # 工作负载类型，支持Deployment、DaemonSet、StatefulSet、Job、CronJob
        name: prometheus-lightweight  # 工作负载名称
        container: prometheus  # 容器名称
        files:
        - logPath: "/var/log"  # 日志目录，绝对路径
          filePattern: "*.log"  # 日志文件名，支持通配
    processors:    # 多行日志定义，若不需要多行，可直接删除processors部分
      type: multiline # 日志类型，可选字段，支持singleline、multiline，默认为singleline
      multilineRegulation: '\d+:\d+:\d+.*?'  # 多行正则表达式，可选字段，类型为multiline时有效
  outputDetail:  # 输出配置
    type: LTS    # 输出类型，固定LTS
    LTS:
      ltsGroupID: abf5f0ad-627e-41cc-8d3f-61c9e1f57f5a      # LTS日志组ID，指定的ID必须存在
      ltsStreamID: f7ed71e9-6b9d-4ba3-86e4-b1b9d22ef4fb     # LTS日志流ID，指定的ID必须存在
```

- **场景三：采集携带指定标签的Pod的容器内日志**

```
apiVersion: logging.openvessel.io/v1
kind: LogConfig
metadata:
  name: test-log-03  # 规则名称按需修改
  namespace: kube-system  # 采集规则命名空间，固定为kube-system
spec:
  inputDetail:  # 输入配置
    type: container_file   # 输入类型，container_file表示容器内日志文件
    containerFile:    # 容器内文件日志相关配置，仅当type为container_file时生效，其他type无需该字段
      podLabels:      # podLabels信息需要参照CRD说明按照实际情况修改
      - includeLabels: # 标签集合，包含以下Labels的Pod会被采集，至少需要指定一个，注意是Pod标签不是工作负载的标签
          foo: bar
        namespaces:  # 命名空间，数组类型，空数组表示所有命名空间
        - monitoring
        - kube-system
        containers: []  # 指定容器名称，数组类型，空数组表示所有容器
        files:
        - logPath: "/var/log"  # 日志目录，绝对路径
          filePattern: "*.log"  # 日志文件名，支持通配
  outputDetail:  # 输出配置
    type: LTS    # 输出类型，固定LTS
    LTS:
      ltsGroupID: abf5f0ad-627e-41cc-8d3f-61c9e1f57f5a      # LTS日志组ID，指定的ID必须存在
      ltsStreamID: f7ed71e9-6b9d-4ba3-86e4-b1b9d22ef4fb     # LTS日志流ID，指定的ID必须存在
```

- **场景四：采集主机目录的日志**

```
apiVersion: logging.openvessel.io/v1
kind: LogConfig
metadata:
  name: test-log-04  # 规则名称按需修改
  namespace: kube-system  # 采集规则命名空间，固定为kube-system
```



```
spec:
  inputDetail:  # 输入配置
    type: host_file   # 输入类型，host_file表示节点日志文件
    hostFile:    # 主机文件日志相关配置，仅当type为host_file时生效，其他type无需该字段
      file:
        logPath: "/var/log"    # 日志目录，绝对路径，按需修改
        filePattern: "messages" # 日志文件，支持通配，按需修改
  outputDetail:  # 输出配置
    type: LTS    # 输出类型，固定LTS
    LTS:
      ltsGroupID: abf5f0ad-627e-41cc-8d3f-61c9e1f57f5a      # LTS日志组ID，指定的ID必须存在
      ltsStreamID: f7ed71e9-6b9d-4ba3-86e4-b1b9d22ef4fb      # LTS日志流ID，指定的ID必须存在
```

表 9-37 参数说明

参数	类型	描述	示例
spec.inputDetail.type	String	输入类型，支持以下参数值： ● container_stdout：表示采集标准输出，需要和containerStdout字段搭配使用。 ● container_file：表示采集容器内日志文件，需要和containerFile字段搭配使用。 ● host_file：表示采集节点日志文件，需要和hostFile字段搭配使用。	-

参数	类型	描述	示例
spec.inputDetail.containerStdout	Object	标准输出相关配置，仅当type为container_stdout时生效，其他type无需该字段。 包含以下字段： • allContainers：是否采集所有容器。true表示采集所有容器，需指定namespaces字段；false表示采集指定工作负载，需指定workloads字段。 • namespaces：命名空间数组，仅在allContainers为true时生效。采集指定命名空间内的容器标准输出，空数组表示采集所有命名空间容器标准输出。 • workloads：指定工作负载采集，数组类型，仅在allContainers为false时生效。 – namespace：工作负载所属命名空间。 – kind：工作负载类型，支持Deployment、DaemonSet、StatefulSet、Job、CronJob。 – name：工作负载名称。 – containers：指定容器名称，数组类型，空数组表示所有容器。 • podLabels：指定Pod标签采集，数组类型，仅在allContainers为false且workloads数组为空时生效。 – includeLabels：标签集合，包含以下Labels的Pod会被采集，至少需要指定一个。注意是Pod标签不是工作负载的标签。 – namespaces：命名空间，数组类型，空数组表示所有命名空间。 – containers：指定容器名称，数组类型，空数组表示所有容器。	示例一：采集某个命名空间下的所有容器标准输出 ... spec: inputDetail: **type: container_stdout** containerStdout: **allContainers: true** namespaces: - monitoring ... 示例二：采集指定工作负载标准输出 ... spec: inputDetail: **type: container_stdout** containerStdout: **allContainers: false** workloads: - namespaces: monitoring kind: Deployment name: prometheus-lightweight container: prometheus ... 示例三：采集指定Pod标准输出 ... spec: inputDetail: **type: container_stdout** containerStdout: **allContainers: false** **workloads: []** podLabels: - includeLabels: foo: bar namespaces: - monitoring containers: [] ...

参数	类型	描述	示例
spec.inputDetail.containerFile	Object	容器内文件日志相关配置，仅当type为container_file时生效，其他type无需该字段。 包含以下字段： ● workloads：指定工作负载采集，数组类型。 – namespace：工作负载所属命名空间。 – kind：工作负载类型，支持Deployment、DaemonSet、StatefulSet、Job、CronJob。 – name：工作负载名称。 – container：指定容器名称。 – files：文件信息，数组类型，包含logPath和filePattern两个字段。 logPath：日志目录，绝对路径，例如"/var/log"。 filePattern：日志文件名，支持通配，例如"*.log"。 ● podLabels：指定Pod标签采集，数组类型，仅在workloads为空时生效。 – includeLabels：标签集合，包含以下Labels的Pod会被采集，至少需要指定一个。注意是Pod标签不是工作负载的标签。 – namespaces：命名空间，数组类型，空数组表示所有命名空间。 – containers：指定容器名称，数组类型，空数组表示所有容器。 – files：文件信息，数组类型，包含logPath和filePattern两个字段。 logPath：日志目录，绝对路径，例如"/var/log"。 filePattern：日志文件名，支持通配，例如"*.log"。	示例一：采集指定工作负载容器内文件 ... spec: inputDetail: **type: container_file** containerFile: workloads: - namespaces: monitoring kind: Deployment name: prometheus-lightweight container: prometheus files: - logPath: "/var/log" filePattern: "*.log" ... 示例二：采集指定Pod容器内文件 ... spec: inputDetail: **type: container_file** containerFile: **workloads: []** podLabels: - includeLabels: foo: bar namespaces: - monitoring containers: [] files: - logPath: "/var/log" filePattern: "*.log" ...

参数	类型	描述	示例
spec.inputDetail.hostFile	Object	主机文件日志相关配置，仅当type为host_file时生效，其他type无需该字段。 ● file： – logPath：日志目录，绝对路径，例如"/var/log"。 – filePattern：日志文件名，支持通配，例如"*.log"。	... spec: inputDetail: **type: host_file** hostFile: files: logPath: "/var/log" filePattern: "*.log" ...
spec.inputDetail.processors	Object	● type：日志类型，可选字段，支持singleline、multiline，默认为singleline。 ● multilineRegulation：多行正则表达式，可选字段，类型为multiline时有效。	... processors: type: multiline multilineRegulation: '\d+:\d+:\d+.*?' ...
spec.outputDetail.type	String	输出类型，固定值为LTS。	-
spec.outputDetail.LTS	Object	支持以下字段： ● ltsGroupID：LTS日志组ID，指定的ID必须存在。 ● ltsStreamID：LTS日志流ID，指定的ID必须存在。ltsStreamID和ltsStreamName字段必须配置一个，但不能同时配置。 ● ltsStreamName：LTS日志流名称，若指定的日志流名称不存在，则会自动创建。ltsStreamID和ltsStreamName字段必须配置一个，但不能同时配置。 ● ltsStreamCreateParam：日志流创建参数，可选字段，仅在指定ltsStreamName且自动创建LTS日志流时生效。 – enterpriseProjectID：LTS日志组企业项目ID，可选字段，未指定则使用集群所属的企业项目ID。	示例一：指定已有的日志组ID和日志流ID ... LTS: ltsGroupID: ***** ltsStreamID: ***** 示例二：指定已有的日志组ID，并指定已有的日志流名称 ... LTS: ltsGroupID: ***** ltsStreamName: test-stream-name-1 示例三：指定已有的日志组ID，并指定不存在的日志流名称，自动创建日志流 ... LTS: ltsGroupID: ***** ltsStreamName: test-stream-name-2 ltsStreamCreateParam: enterpriseProjectID: ""

步骤3 创建LogConfig。

```
kubectl create -f log-config.yaml
```

回显如下，表示LogConfig已创建。


```
logconfig.logging.openvessel.io/test-log-xx created
```

步骤4 查看已创建的LogConfig。

```
kubectl get LogConfig -n kube-system
```

回显如下，表示日志采集策略创建成功。

```
NAME            AGE
test-log-xx     30s
```

----结束

9.4.2.2 通过 ICAgent 采集容器日志（不推荐）

CCE配合AOM收集工作负载的日志，在创建节点时会默认安装AOM的ICAgent（在集群kube-system命名空间下名为icagent的DaemonSet），ICAgent负责收集工作负载的日志并上报到AOM，您可以在CCE控制台和AOM控制台查看工作负载的日志。

约束与限制

ICAgent只采集*.log、*.trace和*.out类型的文本日志文件。

费用说明

AOM每月赠送每个账号500M免费日志采集额度，超过免费额度部分将产生费用（**了解计费详情**）。当前日志使用情况请**点此查看**。

使用 ICAgent 采集日志

在工作负载中可以单独配置日志采集策略，此策略需要使用ICAgent。

步骤1 在CCE中创建**工作负载**时，在配置容器信息时可以设置容器日志。

步骤2 单击┼添加日志策略。

以nginx为例，不同工作负载根据实际情况配置。

图 9-57 添加日志策略

步骤3 存储类型有“主机路径”和“容器路径”两种类型可供选择：

表 9-38 配置日志策略

参数	参数说明
存储类型	• 主机路径：HostPath模式，将主机路径挂载到指定的容器路径（挂载路径）。用户可以在节点的主机路径中查看到容器输出在挂载路径中的日志信息。 • 容器路径：EmptyDir模式，将节点的临时路径挂载到指定的路径（挂载路径）。临时路径中存在的但暂未被采集器上报到AOM的日志数据在Pod实例删除后会消失。
主机路径	请输入主机的路径，如：/var/paas/sys/log/nginx
挂载路径	请输入数据存储要挂载到容器上的路径，如：/tmp **须知** • 请不要挂载到系统目录下，如“/”、“**/var/run**”等，否则会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 • 挂载高危目录的情况下，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。 • AOM只采集最近修改过的前20个日志文件，且默认采集两级子目录。 • AOM只采集挂载路径下的“.log”、“.trace”、“.out”文本日志文件。 • 容器中挂载点的权限设置方法，请参见**为Pod或容器配置安全性上下文**。
主机扩展路径	仅“主机路径”类型需要填写 通过实例的ID或者容器的名称扩展主机路径，实现同一个主机路径下区分来自不同容器的挂载。 会在原先的“卷目录/子目录”中增加一个三级目录。使用户更方便获取单个Pod输出的文件。 • None：不配置拓展路径。 • PodUID：Pod的ID。 • PodName：Pod的名称。 • PodUID/ContainerName：Pod的ID/容器名称。 • PodName/ContainerName：Pod名称/容器名称。
采集路径	设置采集路径可以更精确的指定采集内容，当前支持以下设置方式： • 不设置则默认采集当前路径下.log .trace .out文件 • 设置**表示递归采集5层目录下的.log .trace .out文件 • 设置*表示模糊匹配 例子： 采集路径为/tmp/**/test*.log表示采集/tmp目录及其1-5层子目录下的全部以test开头的.log文件。 **注意** 使用采集路径功能请确认您的**采集器ICAgent**版本为5.12.22或以上版本。

参数	参数说明
日志转储	此处日志转储是指日志的本地绕接。 ● 设置：AOM每分钟扫描一次日志文件，当某个日志文件超过50MB时会对其转储（转储时会在该日志文件所在的目录下生成一个新的zip文件。对于一个日志文件，AOM只保留最近生成的20个zip文件，当zip文件超过20个时，时间较早的zip文件会被删除）。 ● 不设置：若您在下拉列表框中选择“不设置”，则AOM不会对日志文件进行转储。 **说明** ● AOM的日志绕接能力是使用copytruncate方式实现的，如果选择了设置，请务必保证您写日志文件的方式是append（追加模式），否则可能出现文件空洞问题。 ● 当前主流的日志组件例如Log4j、Logback等均已经具备日志文件的绕接能力，如果您的日志文件已经实现了绕接能力，则无需设置。否则可能出现冲突。 ● 建议您的业务自己实现绕接，可以更灵活的控制绕接文件的大小和个数。

步骤4 单击“确定”，并完成创建工作负载。

----结束

YAML 示例（ICAgent）

您可以通过在YAML定义的方式设置容器日志存储路径。

如下所示，使用EmptyDir挂载到容器的“/var/log/nginx”路径下，这样ICAgent就会采集容器“/var/log/nginx”路径下的日志。其中policy字段是CCE自定义的字段，能够让ICAgent识别并采集日志。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testlog
  namespace: default
spec:
  selector:
    matchLabels:
      app: testlog
  template:
    replicas: 1
    metadata:
      labels:
        app: testlog
    spec:
      containers:
        - image: 'nginx:alpine'
          name: container-0
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:
```



```
        - name: vol-log
          mountPath: /var/log/nginx
          policy:
            logs:
              rotate: ''
    volumes:
      - emptyDir: {}
        name: vol-log
    imagePullSecrets:
      - name: default-secret
```

使用HostPath方法如下所示，相比EmptyDir就是volume的类型变成hostPath，且需要配置hostPath在主机上的路径。下面示例中将主机上“/tmp/log”挂载到容器的“/var/log/nginx”路径下，这样ICAgent就会采集容器“/var/log/nginx”路径下的日志，且日志还会在主机的“/tmp/log”路径下存储。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testlog
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testlog
  template:
    metadata:
      labels:
        app: testlog
    spec:
      containers:
        - image: 'nginx:alpine'
          name: container-0
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:
            - name: vol-log
              mountPath: /var/log/nginx
              readOnly: false
              extendPathMode: PodUID
              policy:
                logs:
                  rotate: Hourly
                  annotations:
                    pathPattern: '**'
                    format: ''
      volumes:
        - hostPath:
            path: /tmp/log
          name: vol-log
      imagePullSecrets:
        - name: default-secret
```


表 9-39 关键参数解释

参数	解释	说明
extendPathMode	主机扩展路径	通过实例的ID或者容器的名称扩展主机路径，实现同一个主机路径下区分来自不同容器的挂载。 会在原先的“卷目录/子目录”中增加一个三级目录。使用户更方便获取单个Pod输出的文件。 ● None：不配置拓展路径。 ● PodUID：Pod的ID。 ● PodName：Pod的名称。 ● PodUID/ContainerName：Pod的ID/容器名称。 ● PodName/ContainerName：Pod名称/容器名称。
policy.logs.rotate	日志转储	此处日志转储是指日志的本地绕接。 ● 设置：AOM每分钟扫描一次日志文件，当某个日志文件超过50MB时，会立即对其转储（转储时会在该日志文件所在的目录下生成一个新的zip文件。对于一个日志文件，AOM只保留最近生成的20个zip文件，当zip文件超过20个时，时间较早的zip文件会被删除），转储完成后AOM会将该日志文件清空。 ● 不设置：若您在下拉列表框中选择“不设置”，则AOM不会对日志文件进行转储。 **说明** ● AOM的日志绕接能力是使用copytruncate方式实现的，如果选择了设置，请务必保证您写日志文件的方式是append（追加模式），否则可能出现文件空洞问题。 ● 当前主流的日志组件例如Log4j、Logback等均已经具备日志文件的绕接能力，如果您的日志文件已经实现了绕接能力，则无需设置。否则可能出现冲突。 ● 建议您的业务自己实现绕接，可以更灵活的控制绕接文件的大小和个数。
policy.logs.annotations.pathPattern	采集路径	设置采集路径可以更精确的指定采集内容，当前支持以下设置方式： ● 不设置则默认采集当前路径下.log .trace .out文件 ● 设置**表示递归采集5层目录下的.log .trace .out文件 ● 设置*表示模糊匹配 例子： 采集路径为/tmp/**/test*.log表示采集/tmp目录及其1-5层子目录下的全部以test开头的.log文件。 **注意** 使用采集路径功能请确认您的**采集器ICAgent**版本为5.12.22或以上版本。

参数	解释	说明

policy.logs.annotations.format	多行日志匹配	有些程序打印的日志存在一条完整的日志数据跨占多行（例如 Java 程序日志）情况，日志采集系统默认是按行采集。如果您想在日志采集系统中按整条显示日志，可以开启多行日志，采用时间或正则匹配的方式，当某行日志匹配上预先设置的时间格式或正则表达式，就认为是一条日志的开头，而下一个行首出现作为该条日志的结束标识符。
		格式如下
		{
		"multi": {
		"mode": "time",
		"value": "YYYY-MM-DD hh:mm:ss"
		}
		}
		multi表示分行模式：
		● time：日志时间，请输入时间通配符。时间通配符填写参考示例：如日志中的时间是2017-01-01 23:59:59，按照规则应该填写：YYYY-MM-DD hh:mm:ss。
		● regular：正则模式，请输入正则表达式。

查看日志

日志采集路径配置和工作负载创建完成后，若已配置的路径下存在日志文件，则ICAgent会从已配置的路径中采集日志文件，采集大概需要1分钟，请您耐心等待。

待采集完成后，进入工作负载详情页，单击右上角的“日志”按钮查看日志详情。

您还可以在AOM控制台查看日志。

另外您还可以使用kubectl logs命令查看容器的标准输出，具体如下所示。

```
# 查看指定pod的日志
kubectl logs <pod_name>
kubectl logs -f <pod_name> #类似tail -f的方式查看

# 查看指定pod中指定容器的日志
kubectl logs <pod_name> -c <container_name>

kubectl logs pod_name -c container_name -n namespace (一次性查看)
kubectl logs -f <pod_name> -n namespace (tail -f方式实时查看)
```

9.4.3 采集 Kubernetes 事件

CCE 云原生日志采集插件插件可采集Kubernetes事件上报到云日志服务（LTS）和应用运维管理（AOM），用于保存事件信息和事件告警。

Kubernetes 事件上报云日志服务（LTS）

根据不同的场景，开通Kubernetes事件采集的步骤如下：

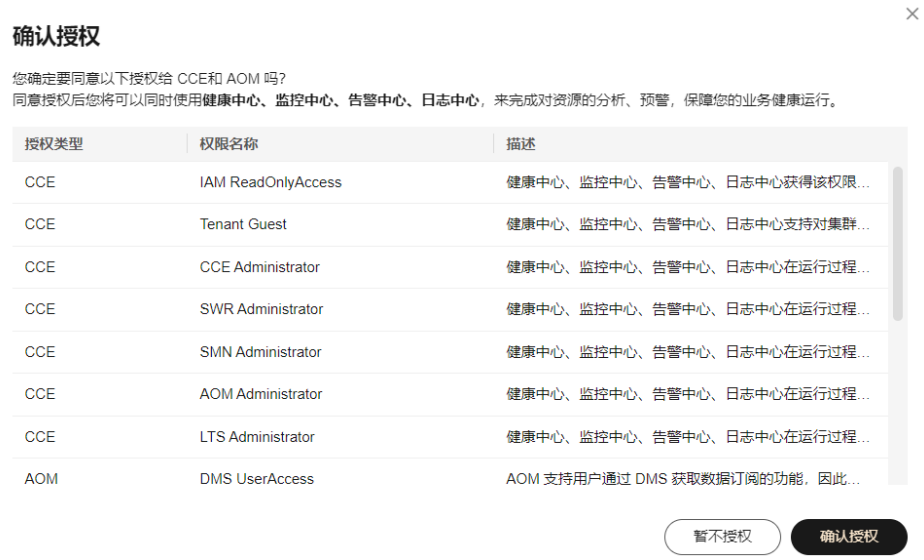
集群未开通日志中心

开通日志中心时，可通过勾选采集Kubernetes事件，创建默认日志采集策略，采集所有事件上报到LTS。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 未进行授权的用户需要先授权，已授权的用户直接跳转下一步。

   在弹出框中单击“确认授权”。

   图 9-58 添加授权



3. 页面单击“开启”，等待约30秒后，页面自动跳转。

   图 9-59 开启

   - 采集容器标准输出：开启后，将创建名为default-stdout的日志策略，并上报所有命名空间下的标准输出到云日志服务（LTS）。
   - 采集Kubernetes事件：开启后，将创建名为default-event的日志策略，并上报所有命名空间下的Kubernetes事件到云日志服务（LTS）。
   - 采集插件日志（NGINX Ingress控制器容器标准输出）：需要安装NGINX Ingress控制器插件，并在插件中开启“日志采集”功能。

     开启后，将创建名为default-nginx-ingress的日志策略，并上报所有命名空间下带有采集标签的nginx-ingress容器标准输出到云日志服务（LTS）。

集群已开通日志中心

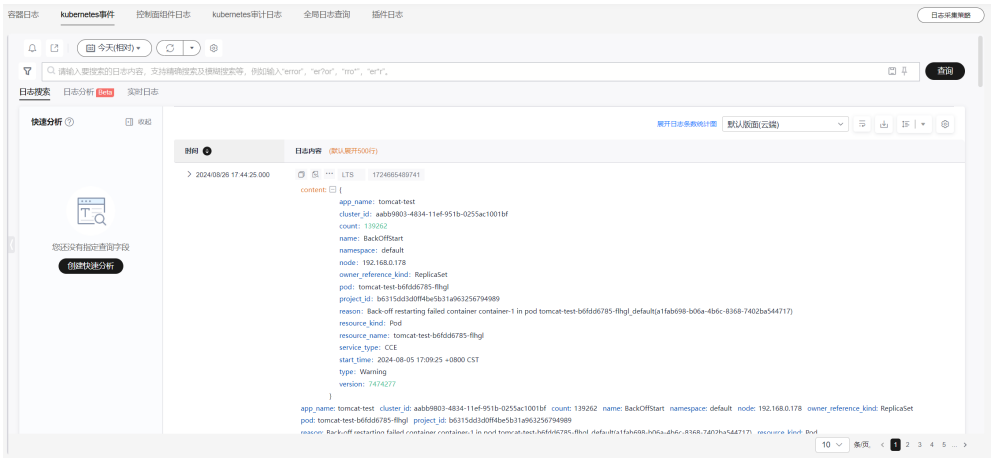
如果集群已开通日志中心，但未开通采集Kubernetes事件，或者删除了对应的日志策略，您可以参考以下步骤手动创建一个日志采集策略。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 右上角单击“日志采集策略”，将显示当前集群所有上报LTS的日志策略。
3. 单击上方“创建日志采集策略”，勾选“采集Kubernetes事件”，然后单击“确定”。

图 9-60 创建日志策略

4. 创建完成后，您可直接在“日志中心”页面查看日志。选择日志策略配置的日志流名称，即可查看上报到云日志服务（LTS）的事件。

图 9-61 查看事件



Kubernetes 事件上报应用运维管理（AOM）

自1.3.2版本起，云原生日志采集插件默认会将上报所有Warning级别事件以及部分Normal级别事件到应用运维管理（AOM），上报的事件可用于配置告警。当集群版本为1.19.16、1.21.11、1.23.9或1.25.4及以上时，安装云原生日志采集插件后，事件上报AOM将不再由控制面组件上报，改为由云原生日志采集插件上报，卸载插件后将不再上报事件到AOM。

自定义事件上报

若已上报的事件不能满足需求，可通过修改配置，修改需要上报到应用运维管理（AOM）的事件。

通过控制台配置

步骤1 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“配置中心”。

步骤2 选择“监控运维配置”页签，在“日志配置”中修改Kubernetes事件上报至AOM的策略。

- 异常事件上报：默认开启，会将所有异常事件上报至AOM。您可以单击“配置黑名单”，将不需要上报的事件添加至黑名单进行管理，其中“事件名称”可通过**CCE事件列表**查询。
- 普通事件上报：开启后，会将普通事件上报至AOM，系统默认配置了部分需要上报的普通事件。如果您需要自定义上报的事件，可以单击“配置白名单”，将需要上报添加至白名单进行管理，其中“事件名称”可通过**CCE事件列表**查询。

步骤3 配置修改完成后，单击“确认配置”。

----结束

通过 kubectl 配置

步骤1 在集群上执行以下命令，编辑当前的事件采集配置。

kubectl edit logconfig -n kube-system default-event-aom

步骤2 根据需要修改事件采集配置。

```
apiVersion: logging.openvessel.io/v1
kind: LogConfig
metadata:
  annotations:
    helm.sh/resource-policy: keep
  name: default-event-aom
  namespace: kube-system
spec:
  inputDetail:    #采集端配置
    type: event    #采集端类型，请勿修改
    event:
      normalEvents:    #Normal级别事件采集配置
        enable: true    #是否开启Normal级别事件采集
        includeNames:    #需要采集的事件名，不指定则采集所有事件
        - NotTriggerScaleUp
        excludeNames:    #不采集的事件名，不指定则采集所有事件
        - ScaleDown
      warningEvents:    #Warning级别事件采集配置
        enable: true    #是否开启Warning级别事件采集
        includeNames:    #需要采集的事件名，不指定则采集所有事件
        - NotTriggerScaleUp
        excludeNames:    #不采集的事件名，不指定则采集所有事件
```



```
      - ScaleDown
 outputDetail:
   type: AOM    #输出端类型，请勿修改
   AOM:
     events:
     - name: DeleteNodeWithNoServer    #事件名，必选
       resourceType: Namespace    #事件对应的资源类型
       severity: Major    #事件上报到AOM后的事件级别，默认Major。可选值：Critical：紧急；Major：重要；
Minor：次要；Info：提示
```

----结束

9.4.4 采集 NGINX Ingress 访问日志

CCE云原生日志采集插件支持收集NGINX Ingress控制器插件日志，可分析历史流量变化情况，得到业务流量特征，为业务决策提供数据支持。

约束与限制

- 集群中需安装2.2.82及以上、2.6.32及以上、3.0.8及以上版本的**NGINX Ingress控制器**插件。
- 集群中需安装1.6.0及以上版本的**云原生日志采集插件**插件。

费用说明

LTS创建日志组免费，并每月赠送每个账号一定量免费日志采集额度，超过免费额度部分将产生费用（**价格计算器**）。

步骤一：开启插件日志采集

集群中需要安装NGINX Ingress控制器插件，并在插件中开启“日志采集”功能。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“插件中心”。
2. 找到**NGINX Ingress控制器**插件并开启“日志采集”功能。
   - 已安装插件时：单击“管理”，找到已安装的插件实例，单击右侧“编辑”，在参数配置中找到“日志采集”并开启。

     如果集群中安装了多个NGINX Ingress控制器，需修改每个插件实例的配置才可采集所有实例的日志。
   - 未安装插件时：单击“安装”，在参数配置中找到“日志采集”并开启。其余参数请根据需求选择，配置说明请参见**NGINX Ingress控制器**。

步骤二：在日志中心采集 NGINX Ingress 控制器插件日志

根据不同的场景，开通NGINX Ingress控制器插件日志采集的步骤如下：

集群未开通日志中心

如果集群未开通日志中心，您可以在开通日志中心时通过勾选“采集插件日志（NGINX Ingress控制器容器标准输出）”选项，直接创建NGINX Ingress插件的默认日志采集策略。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 未进行授权的用户需要先授权，已授权的用户可忽略本步骤。
   在弹出框中单击“确认授权”。

图 9-62 添加授权

3. 在页面中勾选需要采集的日志类型，单击“开启”，等待约30秒后，页面自动跳转。

图 9-63 开启

- 采集容器标准输出：您可按需开启，开启后将创建名为default-stdout的日志策略，并上报所有命名空间下的标准输出到云日志服务（LTS）。
- 采集Kubernetes事件：您可按需开启，开启后将创建名为default-event的日志策略，并上报所有命名空间下的Kubernetes事件到云日志服务（LTS）。
- 采集插件日志（NGINX Ingress控制器容器标准输出）：本示例中必选，需要安装NGINX Ingress控制器插件，并在插件中开启“日志采集”功能。
  开启后，将创建名为default-nginx-ingress的日志策略，并上报所有命名空间下带有采集标签的NGINX Ingress控制器容器标准输出到云日志服务（LTS）。

集群已开通日志中心

如果集群已开通日志中心，但未开通NGINX Ingress控制器插件日志采集，您可以手动创建一个日志采集策略。

1. 登录云容器引擎（CCE）控制台，单击集群名称进入集群，选择左侧导航栏的“日志中心”。
2. 右上角单击“日志采集策略”，将显示当前集群所有上报LTS的日志策略。
3. 单击上方“创建日志采集策略”，勾选“采集插件日志（NGINX Ingress控制器容器标准输出）”，单击确定。

图 9-64 创建日志策略

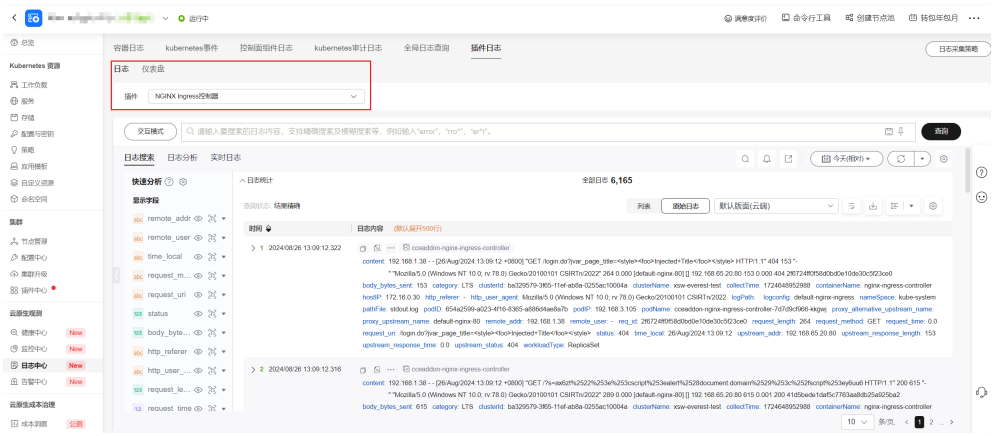
4. 系统将自动创建名为default-nginx-ingress的日志采集策略。创建完成后，您可前往“日志中心”页面，选择“插件日志”页签，即可查看该插件上报到云日志服务（LTS）的日志。

图 9-65 查看日志

步骤三：查看 NGINX Ingress 控制器插件日志

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“插件日志”页签，选中插件为“NGINX Ingress控制器”。关于该页面的操作详情，请参见**LTS用户指南**。

图 9-66 查看 NGINX Ingress 控制器插件日志



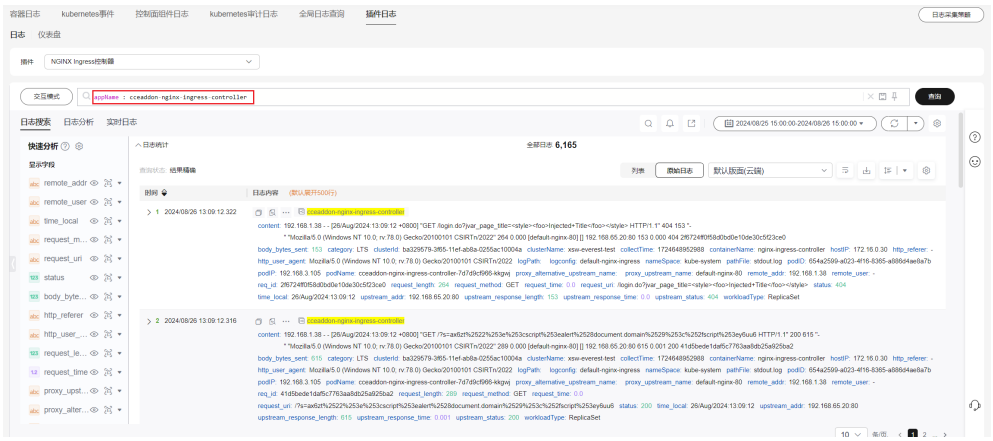
3. 如果集群中同时安装多套NGINX Ingress控制器，且多个控制器均开启了日志采集，此处的日志将显示所有控制器的日志，您可以在筛选栏中通过appName进行筛选。

   其中appName的取值如下：

   - NGINX Ingress控制器名称为nginx，appName为**cceaddon-nginx-ingress-controller**。
   - NGINX Ingress控制器名称为其他自定义名称，如{className}，则appName为**cceaddon-nginx-ingress-*{className}*-controller**。

   筛选cceaddon-nginx-ingress-controller的日志示例如下：

   appName : cceaddon-nginx-ingress-controller



步骤四：查看 NGINX Ingress 仪表盘

CCE支持采集NGINX Ingress日志，并借助LTS日志服务进行多维度分析，并为NGINX日志配置结构化和仪表盘，支持**监控中心**、**访问中心**和**秒级监控**仪表盘，满足不同场景的监控需求。

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“插件日志”页签，选中“仪表盘”页面，您可以选择不同的仪表盘模板。
3. 如果集群中同时安装多套NGINX Ingress控制器，且多个控制器均开启了日志采集，此处的日志将显示所有控制器的日志，您可以在筛选栏中通过appName进行筛选。

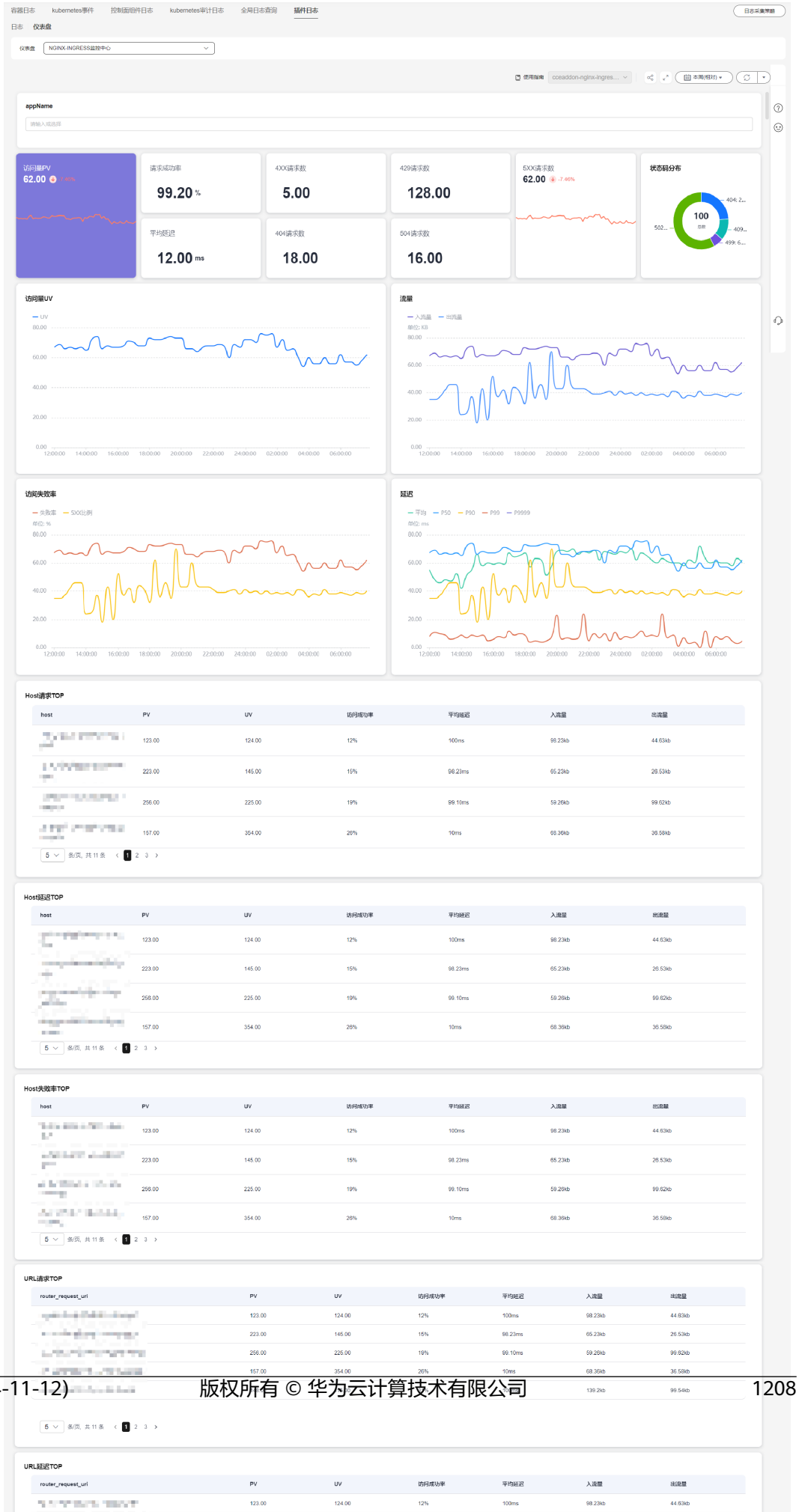
   其中appName的取值如下：

   - NGINX Ingress控制器名称为nginx，appName为**cceaddon-nginx-ingress-controller**。
   - NGINX Ingress控制器名称为其他自定义名称，如{className}，则appName为**cceaddon-nginx-ingress-*{className}*-controller**。

监控中心

监控中心主要展示NGINX Ingress的基本信息，支持不同时间维度（相对时间、整点时间、自定义时间段）的呈现，包括以下参数报表：

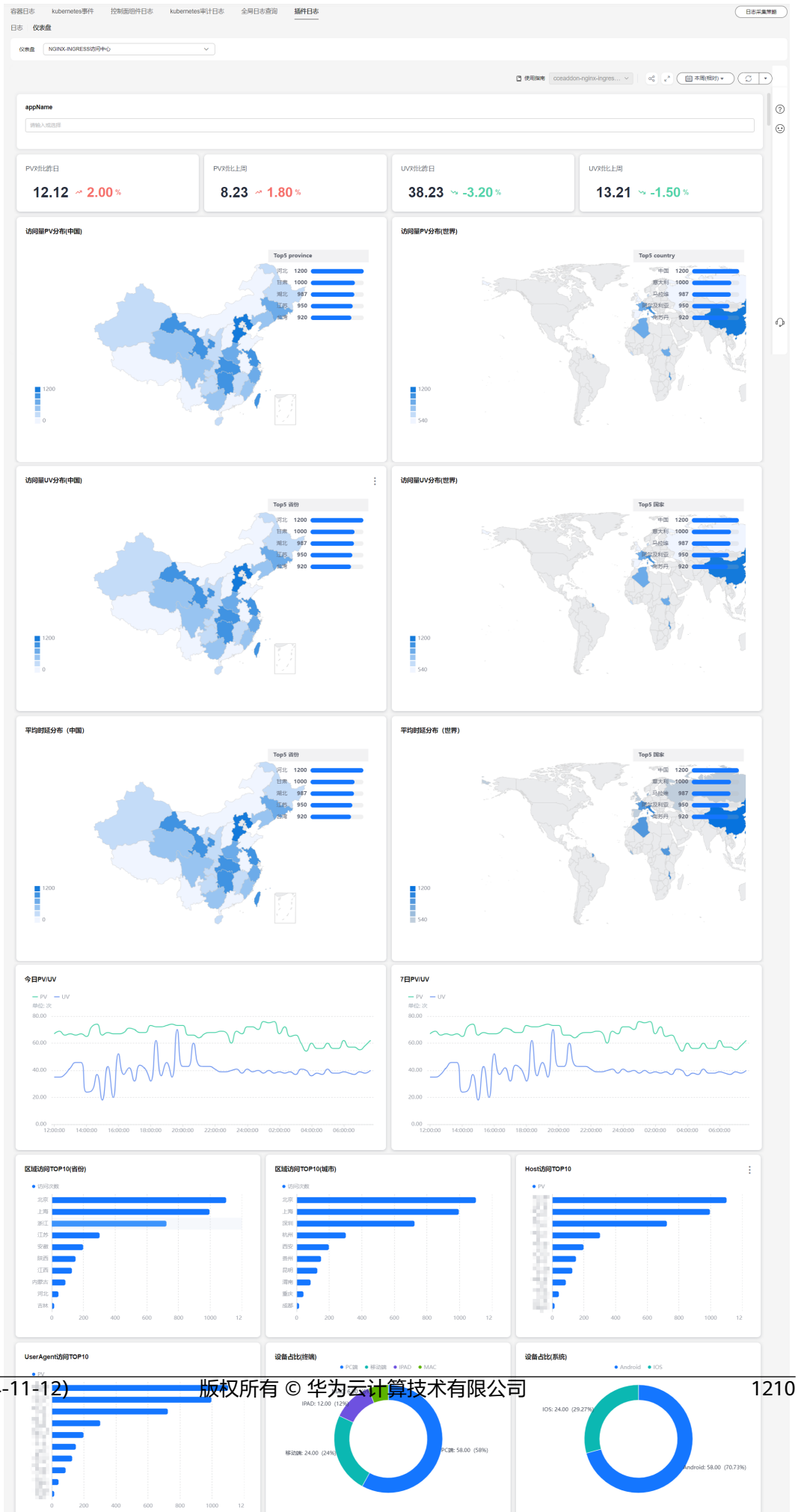
- 访问数据：访问量PV、访问量UV、请求成功率、平均延迟、4XX请求数、404请求数、429请求数、5XX请求数、504请求数、状态码分布、流量、访问失败率和延迟。
- TOP统计：包括Host请求TOP、Host延迟TOP、Host失败率TOP、URL请求TOP、URL延迟TOP、URL失败率TOP、后端请求TOP、后端延迟TOP和后端失败率TOP。



访问中心

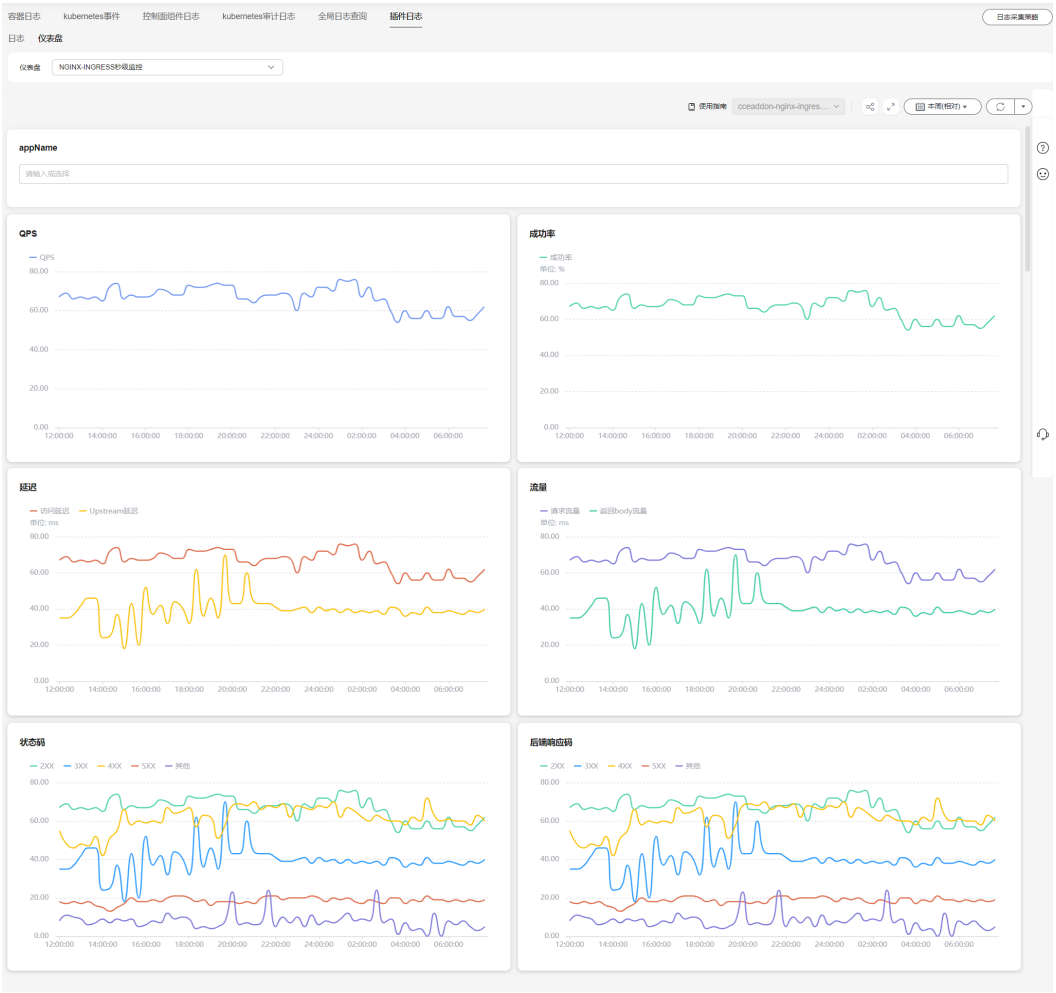
访问中心主要提供细致的访问请求统计信息，支持不同时间维度（相对时间、整点时间、自定义时间段）的呈现，包括以下参数报表：

- 数据对比：PV对比昨日、PV对比上周、UV对比昨日、UV对比上周、今日PV/UV、7日PV/UV。
- 数据分布：访问量PV分布（中国）、访问量PV分布（世界）、访问量UV分布（中国）、访问量UV分布（世界）、平均时延分布（中国）、平均时延分布（世界）、设备占比（终端）、设备占比（系统）。
- TOP统计：区域访问TOP10（省份）、区域访问TOP10（城市）、Host访问TOP10、UserAgent访问TOP10、TOP URL、TOP 访问IP。



秒级监控

CCE提供NGINX Ingress秒级监控能力，收集和分析关键性能指标，可实时洞察NGINX Ingress网络流量和应用性能，包括QPS、成功率、延迟、流量、状态码、后端响应码等图表。



9.4.5 采集控制面组件日志

集群支持对用户开放集群控制节点的日志信息。在日志中心页面可以选择需要上报日志的控制面组件，支持kube-controller-manager、kube-apiserver、kube-scheduler三个组件。

约束与限制

- 如您需要查看集群控制面组件日志，集群必须为v1.21.7-r0及以上补丁版本、v1.23.5-r0及以上补丁版本或1.25版本。
- 请确保云日志服务LTS资源配额充足，LTS的默认配额请参见**基础资源**。

集群控制面组件说明

当前CCE支持收集以下三种类型的控制面日志，每个日志流对应一个Kubernetes控制层面组件。关于这些组件的更多信息，请参见**Kubernetes组件**。

表 9-40 集群控制面组件说明

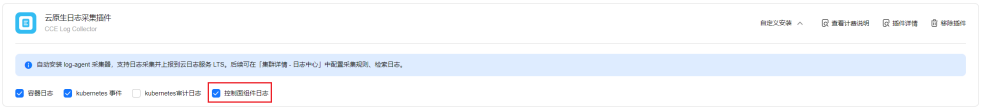
类别	组件	日志流	说明
控制面组件日志	kube-apiserver	kube-apiserver-{{clusterID}}	kube-apiserver组件是暴露Kubernetes API接口的控制层面的组件。更多信息，请参见**kube-apiserver**。
	kube-controller-manager	kube-controller-manager-{{clusterID}}	kube-controller-manager组件是Kubernetes集群内部的管理控制中心，内嵌了Kubernetes发布版本中核心的控制链路。更多信息，请参见**kube-controller-manager**。
	kube-scheduler	kube-scheduler-{{clusterID}}	kube-scheduler组件是Kubernetes集群的默认调度器。更多信息，请参见**kube-scheduler**。

开启集群控制面日志

创建集群时开启

1. 登录云容器引擎（CCE）控制台。
2. 在控制台上方导航栏，单击“购买集群”，填写集群配置并单击“下一步：插件选择”。
3. 在“插件选择”页面中，选择安装“云原生日志采集插件”并单击“下一步：插件配置”。
4. 在“插件配置”页面中，在“云原生日志采集插件”配置中勾选“控制面组件日志”。

   图 9-67 创建集群时开启集群控制面日志

5. 单击“下一步：确认配置”完成集群创建。

已有集群中开启

1. 登录云容器引擎（CCE）控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面组件日志”页签，选择需要采集的控制面组件，单击“一键开启”。

图 9-68 选择控制面组件

查看集群控制面组件日志

通过CCE控制台查看目标集群控制面组件日志

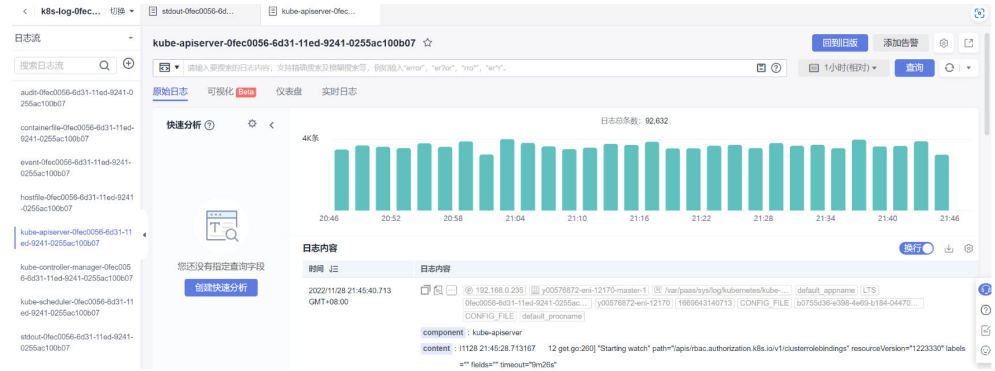
1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面组件日志”页签，在控制面日志中选中需要查看的日志主题，支持的控制面组件日志请参见**集群控制面组件说明**。关于该页面的操作详情，请参见**LTS用户指南**。

图 9-69 查看控制面组件日志

通过LTS控制台查看目标集群控制面组件日志

1. 登录LTS控制台，选择“日志管理”页面。
2. 通过集群ID查到对应的日志组，单击该日志组名称，查看日志流，详情请参见**LTS用户指南**。

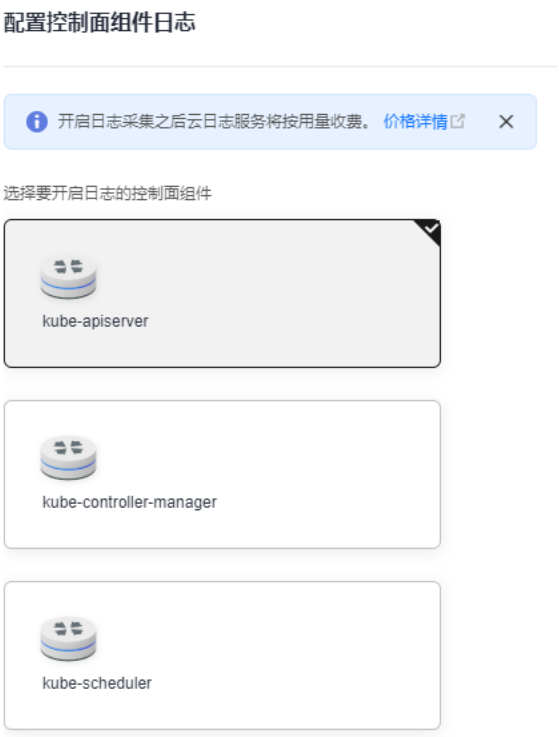
图 9-70 通过 LTS 控制台查看控制面组件日志



关闭集群控制面组件日志

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面组件日志”页签，单击右上角“配置控制面组件日志”，在“配置控制面组件日志”中修改日志配置。

图 9-71 关闭控制面组件日志



3. 选择是否开启各个组件日志，并单击“确定”。

说明

关闭集群控制面组件日志后，原有的日志流将不再更新日志，但已有的日志不会被删除，因此可能会产生LTS日志费用。

9.4.6 采集 Kubernetes 审计日志

集群支持对用户开放集群Master节点的日志信息。您可以在“日志中心”页面的“控制面审计日志”页签选择是否上报Kubernetes审计日志到云日志服务（LTS）。

约束与限制

- 如您需要查看集群Kubernetes审计日志，集群必须为v1.21.7-r0及以上补丁版本、v1.23.5-r0及以上补丁版本或1.25版本。
- 请确保云日志服务LTS资源配额充足，LTS的默认配额请参见**基础资源**。

集群 Kubernetes 审计日志说明

表 9-41 集群 Kubernetes 审计日志说明

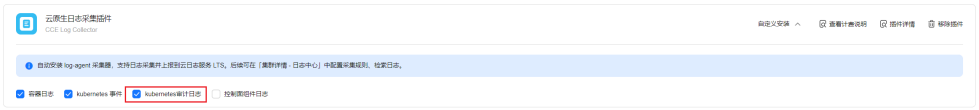
类别	组件	日志流	说明
控制面审计日志	audit	audit-{{clusterID}}	集群审计日志提供了与安全相关的、按时间顺序排列的记录集，记录每个用户使用Kubernetes API的应用以及控制面自身引发的活动。

开启集群控制面审计日志

创建集群时开启

1. 登录云容器引擎（CCE）控制台。
2. 在控制台上方导航栏，单击“购买集群”，填写集群配置并单击“下一步：插件选择”。
3. 在“插件选择”页面中，选择安装“云原生日志采集插件”并单击“下一步：插件配置”。
4. 在“插件配置”页面中，在“云原生日志采集插件”配置中勾选“kubernetes审计日志”。

   图 9-72 创建集群时开启集群审计日志

5. 单击“下一步：确认配置”完成集群创建。

已有集群中开启

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面审计日志”页签，选择audit组件，单击“一键开启”。

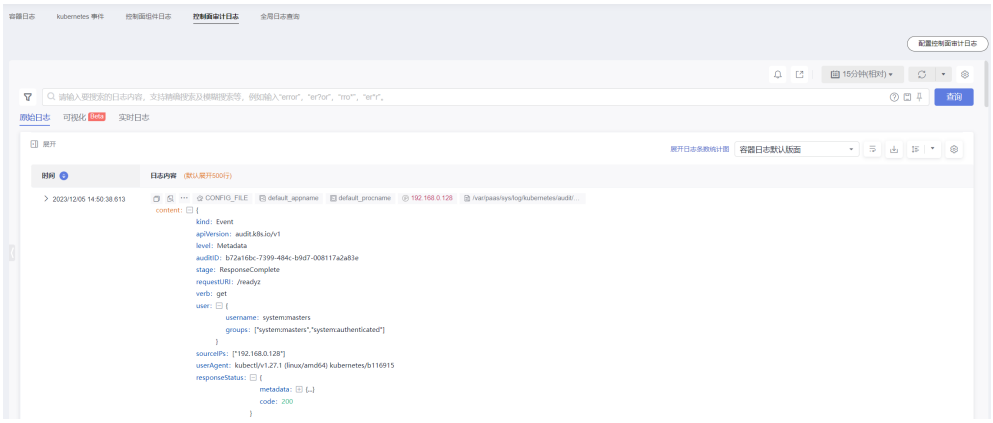
图 9-73 已有集群中开启审计日志

查看集群控制面审计日志

通过CCE控制台查看目标集群控制面审计日志

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面审计日志”页签，查看集群中的审计日志。关于该页面的操作详情，请参见**LTS用户指南**。

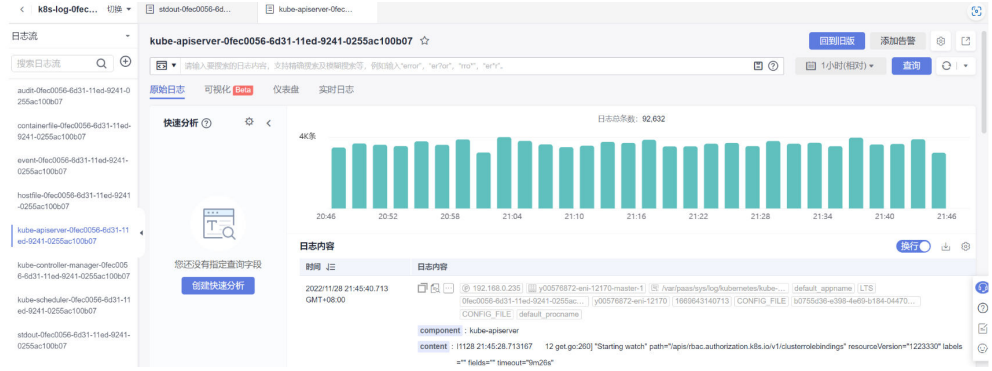
图 9-74 通过 CCE 控制台查看目标集群控制面审计日志



通过LTS控制台查看目标集群控制面审计日志

1. 登录LTS控制台，选择“日志管理”页面。
2. 通过集群ID查到对应的日志组，单击该日志组名称，查看日志流。详情请参见**LTS用户指南**。

图 9-75 通过 LTS 控制台查看目标集群控制面审计日志



关闭集群控制面审计日志

1. 登录CCE控制台，进入一个已有的集群，在左侧导航栏中选择“日志中心”。
2. 选择“控制面审计日志”页签，单击右上角“配置控制面审计日志”，选择是否开启控制面审计日志。

图 9-76 关闭集群控制面审计日志

3. 取消勾选audit，并单击“确定”。

说明

关闭集群控制面审计日志后，原有的日志流将不再更新日志，但已有的日志不会被删除，因此可能会产生LTS日志费用。

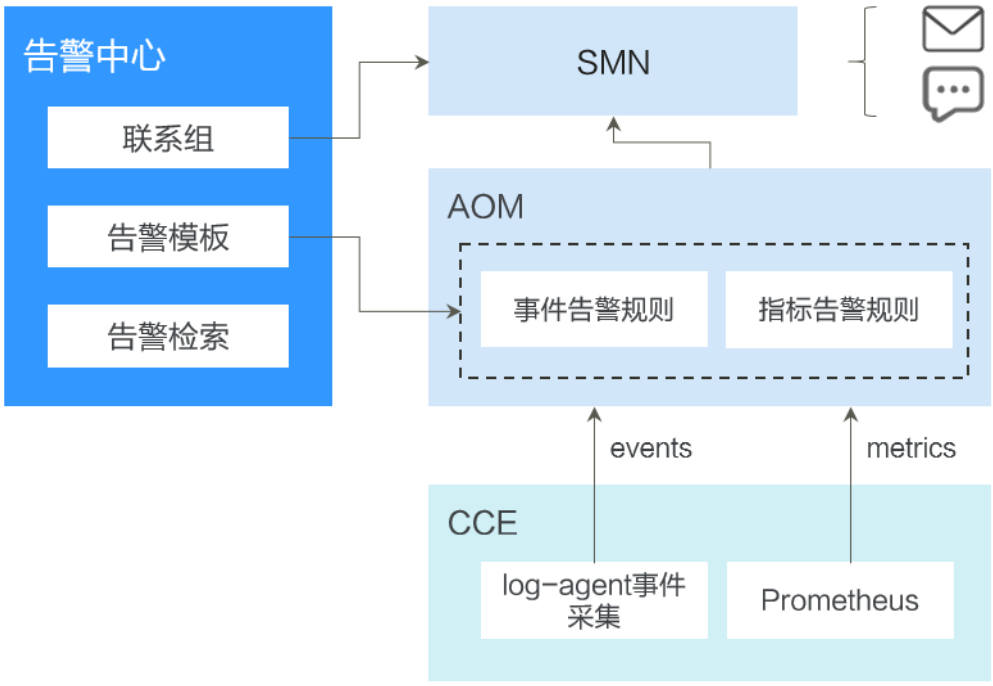
9.5 告警中心

9.5.1 告警中心概述

云原生告警是可观测性体系里面比较重要的一环。在云原生告警中，除了传统的CPU、内存等资源使用量的告警以外，还有容器重启等事件告警、应用访问失败等自定义的监控指标告警。

CCE的云原生告警能力是由AOM服务提供的，支持指标和事件的告警。同时，CCE集群详情中增加了告警中心能力，能支持快速配置资源等常用告警和告警查看。

图 9-77 告警中心架构



- 告警中心

  基于AOM服务的告警能力实现，提供集群内的告警快速检索、告警快速配置的能力。用户可以通过告警中心一键配置常用的告警规则。

- AOM服务

  华为云应用运维管理服务，是云上应用的一站式立体化运维管理平台，是云上监控、告警的基础。

- SMN服务

  华为云的消息通知服务，是云上应用发送告警或通知的依赖服务。在云原生场景中，在AOM服务触发的告警将通过SMN里面配置的短信、电子邮件、HTTP等方式发送。

9.5.2 通过告警中心一键配置告警

告警中心基于AOM告警功能，提供集群内置告警一键开启能力，在集群发生故障时能够及时发现并预警，协助您维护业务稳定性。智能告警中心可有效节省您在AOM侧手动配置告警规则的工作量，并且内置的告警规则基于华为云容器团队大规模集群运维经验，能够满足您的日常运维所需，覆盖容器服务异常事件告警、集群相关基础资源的关键指标告警及集群中应用的指标告警。

约束与限制

- 集群版本仅支持v1.17及以上。
- 仅华为云/华为账号，或者拥有CCE Administrator权限或CCE FullAccess权限的IAM用户可进行告警中心所有操作。CCE ReadOnlyAccess权限的IAM用户可以查看所有资源信息，但是无法进行任何操作。

开启告警中心

CCE Standard集群和CCE Turbo集群均支持开启告警中心。

步骤1 在目标集群左侧导航栏选择“告警中心”。

步骤2 选择“告警规则”页签，单击“开启告警中心”，在弹出的页面中选择一个或多个联系组，以便分组管理订阅终端并接收告警消息。如果当前还没有联系组，请参考**配置告警通知人**进行创建。

步骤3 单击“确认”完成功能开启。

说明

告警中心中的指标类告警规则依赖云原生监控插件上报指标数据到AOM Prometheus实例，需要开通监控中心。当您的集群未安装插件或者在安装插件时未对接AOM Prometheus实例，告警中心将不会创建指标类告警规则。开通监控中心请参考**开通监控中心**。

表9-42中使用problem_gauge指标的指标类告警规则依赖**CCE节点故障检测**插件（NPD）。如需要使用相关的告警规则，请确保节点故障检测插件（NPD）已安装且正常运行。

表9-42中的事件类告警依赖日志中心开启收集Kubernetes事件的能力，详情请参见**采集Kubernetes事件**。

----结束

配置告警规则

CCE Standard集群、CCE Turbo集群开启智能告警中心后，可以进行告警规则的配置和管理。

步骤1 登录CCE控制台。

步骤2 在集群列表页面，单击目标集群名称进入详情页。

步骤3 在左侧导航栏选择“告警中心”，选择“告警规则”页签，在此处进行告警规则的配置和管理。

智能告警中心功能会默认生成容器场景下的告警规则模板（包含异常事件告警、异常指标告警）。告警规则被分类为若干个告警规则集，您可以为告警规则集关联多个联系组，并开启或关闭告警项。告警规则集中包含多个告警规则，一个告警规则对应单个异常的检查项。关于默认告警规则模板，请参见**表9-42**。

----结束

表 9-42 默认告警规则

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
负载规则集	Pod状态异常	检查Pod状态是否异常	指标类	云原生监控插件	sum(min_over_time(kube_pod_status_phase{phase=~"Pending|Unknown|Failed"}[10m]) and count_over_time(kube_pod_status_phase{phase=~"Pending|Unknown|Failed"}[10m]) > 18)by (namespace,pod, phase, cluster_name, cluster) > 0
	Pod频繁重启	检查Pod是否频繁重启	指标类	云原生监控插件	increase(kube_pod_container_status_restarts_total[5m]) > 3
	Deployment副本数不匹配	检查无状态负载的副本数是否匹配	指标类	云原生监控插件	(kube_deployment_spec_replicas != kube_deployment_status_replicas_available) and (changes(kube_deployment_status_replicas_updated[5m]) == 0)
	Statefulset副本数不匹配	检查有状态负载的副本数是否匹配	指标类	云原生监控插件	(kube_statefulset_status_replicas_ready != kube_statefulset_status_replicas) and (changes(kube_statefulset_status_replicas_updated[5m]) == 0)

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	容器CPU使用率大于百分之八十	检查容器CPU使用率是否大于80%	指标类	云原生监控插件	100 * (sum(rate(container_cpu_usage_seconds_total{image!="", container!="POD"}[1m])) by (cluster_name,pod,node,namespace,container, cluster) / sum(kube_pod_container_resource_limits{resource="cpu"}) by (cluster_name,pod,node,namespace,container, cluster)) > 80
	容器内存使用率大于百分之八十	检查容器内存使用率是否大于80%	指标类	云原生监控插件	(sum(container_memory_working_set_bytes{image!="", container!="POD"}) BY (cluster_name, node,container, pod , namespace, cluster) / sum(container_spec_memory_limit_bytes > 0) BY (cluster_name, node, container, pod , namespace, cluster) * 100) > 80
	容器状态异常	检查容器状态是否异常	指标类	云原生监控插件	sum by (namespace, pod, container, cluster_name, cluster) (kube_pod_container_status_waiting_reason) > 0
	更新负载均衡失败	检查更新负载均衡是否成功	事件类	云原生日志采集插件	不涉及

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	Pod内存不足OOM	检查Pod是否OOM	事件类	节点故障检测插件（1.18.41及以上版本） 云原生日志采集插件（1.3.2及以上版本）	PodOOMKilling
节点资源规则集	Kube持久卷使用率高	检查节点上持久卷使用率是否过高	指标类	云原生监控插件	(kubelet_volume_stats_available_bytes{job="kubelet"} / kubelet_volume_stats_capacity_bytes{job="kubelet"}) < 0.03 and kubelet_volume_stats_used_bytes{job="kubelet"} > 0
	Kube持久卷声明状态异常	检查持久卷声明状态是否异常	指标类	云原生监控插件	kube_persistentvolumeclaim_status_phase{phase=~"Failed|Pending|Lost"} > 0
	Kube持久卷状态异常	检查持久卷状态是否异常	指标类	云原生监控插件	kube_persistentvolume_status_phase{phase=~"Failed|Pending"} > 0
	节点CPU使用率超过百分之八十	检查节点CPU使用率是否大于80%	指标类	云原生监控插件	100 - (avg by(node, cluster_name, cluster) (rate(node_cpu_seconds_total{mode="idle"}[2m])) * 100) > 80
	节点内存可用率不足百分之十	检查节点可用内存是否不足10%	指标类	云原生监控插件	node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 < 10
	节点磁盘可用率不足百分之十	检查节点可用磁盘是否不足10%	指标类	云原生监控插件	avg((node_filesystem_avail_bytes * 100) / node_filesystem_size_bytes) by (device, node, cluster_name, cluster) < 10

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	节点磁盘空间不足	检查节点磁盘空间是否充足	事件类	云原生日志采集插件	不涉及
	节点EmptyDir存储池异常	检查节点临时卷存储池是否异常	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="EmptyDirVolumeGroupStatusError"} >= 1
	节点内存资源不足	检查节点整体内存是否充足	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="MemoryProblem"} >= 1
	节点持久卷存储池异常	检查节点上持久卷存储池是否异常	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="LocalPvVolumeGroupStatusError"} >= 1
	节点挂载点异常	检查节点上的挂载点是否异常	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="MountPointProblem"} >= 1
	节点文件句柄数不足	检查系统关键资源FD文件句柄数是否充足	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="FDProblem"} >= 1
	节点磁盘卡IO	检查节点磁盘是否存在卡IO故障	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="DiskHung"} >= 1
	节点磁盘只读	检查节点磁盘是否只读	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="DiskReadonly"} >= 1

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	节点磁盘异常	检查节点系统盘、CCE数据盘（包含Docker逻辑盘与Kubelet逻辑盘）的磁盘使用情况	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="DiskProblem"} >= 1
	节点磁盘慢IO	检测节点磁盘是否存在慢IO故障	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="DiskSlow"} >= 1
	节点进程资源不足	检查系统关键资源PID进程资源是否充足	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="PIDProblem"} >= 1
	节点链接跟踪表不足	检查节点链接跟踪表是否充足	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="ConntrackFullProblem"} >= 1
节点状态规则集	ResolvConf配置文件异常	检查ResolvConf配置文件是否异常	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="ResolvConfFileProblem"} >= 1
	节点CNI组件异常	检查节点CNI（容器网络）组件是否正常运行	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="CNIProblem"} >= 1
	节点CRI组件异常	检查关键组件CRI（容器运行时组件）Docker或Containerd的运行状态	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="CRIProblem"} >= 1

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	节点Kube-proxy故障	检查Kube-proxy是否正常运行	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="KUBEPROXYProblem"} >= 1
	节点Kubelet异常	检查Kubelet状态是否异常	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="KUBELETProblem"} >= 1
	节点存在计划事件	检查节点是否存在主机计划事件	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="ScheduledEvent"} >= 1
	Node状态抖动	检查节点状态是否在正常和异常之间抖动	指标类	云原生监控插件 节点故障检测插件	sum(changes(kube_node_status_condition{status="true",condition="Ready"}[15m])) by (cluster_name, node, cluster) > 2
	节点Containerd频繁重启	检查Containerd是否频繁重启	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="FrequentContainerdRestart"} >= 1
	节点任务夯住	检查节点是否存在任务夯住	事件类	云原生日志采集插件	TaskHung
	节点存储池配置有误	检查节点临时卷及持久卷存储池配置是否异常	事件类	云原生日志采集插件	InvalidStoragePool
	节点状态异常	检查节点状态是否异常	事件类	云原生日志采集插件	NodeNotReady
	节点进程D异常	检查节点是否存在D进程	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="ProcessD"} >= 1

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	节点进程Z异常	检查节点是否存在Z进程	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="ProcessZ"} >= 1
	节点CRI频繁重启	检查CRI是否频繁重启	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="FrequentCRIRestart"} >= 1
	节点Docker频繁重启	检查Docker是否频繁重启	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="FrequentDockerRestart"} >= 1
	节点Kubelet频繁重启	检查Kubelet是否频繁重启	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="FrequentKubeletRestart"} >= 1
	节点NTP服务故障	检查关键系统服务节点时钟同步服务ntpd或chronyd是否正常运行	指标类	云原生监控插件 节点故障检测插件	problem_gauge{type="NTPProblem"} >= 1
	节点内存不足强杀进程	检查节点是否存在OOM事件	事件类	节点故障检测插件	OOMKilling
节点扩缩容规则集	节点池资源售罄	检查节点池资源是否充足	事件类	云原生日志采集插件	NodePoolSoldOut
	扩容节点超时	检查节点池扩容节点是否超时	事件类	云原生日志采集插件	ScaleUpTimedOut
	节点池扩容节点失败	检查节点池扩容节点是否异常	事件类	云原生日志采集插件	FailedToScaleUpGroup

告警规则类型	告警项	告警说明	告警类型	依赖项	PromQL/事件名称
	节点池缩容节点失败	检查节点池缩容节点是否异常	事件类	云原生日志采集插件	ScaleDownFailed
集群状态规则集	集群状态不可用	检查集群状态是否可用	事件类	云原生日志采集插件	不涉及

配置告警通知人

联系组是基于**消息通知服务 SMN**的主题功能实现的，目的是为消息发布者和订阅者提供一个可以相互交流的通道。联系组包含一个或多个订阅终端，您可以通过配置告警联系组，分组管理订阅终端，接收告警信息。联系组创建完成后，需要绑定至告警规则集，这样，当有告警触发时，联系组中的订阅终端就可以收到告警消息了。

步骤1 登录CCE控制台。

步骤2 在集群列表页面，单击目标集群名称进入详情页。

步骤3 在左侧导航栏选择“告警中心”，选择“默认联系组”页签。

步骤4 单击“新建联系组”，在弹出的页面中输入联系组参数。

- **联系组名称**：输入联系组名称，创建后不可修改。名称只能包含大写字母、小写字母、数字、-和_，且必须由大写字母、小写字母或数字开头，名称长度为1~255字符。
- **告警消息显示名**：即订阅终端接收消息的标题名称。假设订阅终端为邮件，推送邮件消息时，若已设置告警消息显示名，发件人则呈现为“显示名”，若未设置告警消息显示名，发件人呈现为“username@example.com”。支持在联系组创建完成后修改告警消息显示名。
- **添加订阅终端**：您需要添加一个或多个订阅终端来接收告警消息。终端类型包括短信和邮件，选择“短信”时，请输入有效的手机号码；选择“邮件”时，请输入有效的电子邮件地址。

步骤5 单击“确定”完成联系组的创建。

返回联系组列表，订阅终端状态为“未确认”，您需要继续执行后续操作，向该终端发送订阅请求，以验证终端有效性。

步骤6 单击操作列“请求订阅”，向该终端发送订阅请求。若终端收到请求，请按照提示进行确认，确认完成后订阅终端状态将变为“已确认”。

步骤7 联系组创建并确认后，单击  图标启用联系组，实现联系组和告警规则集的绑定。

说明

告警规则集最多支持绑定5个联系组。

----结束

查看告警列表

您可以在“告警列表”页面查看最近发送的历史记录。

步骤1 登录CCE控制台。

步骤2 在集群列表页面，单击目标集群名称进入详情页。

步骤3 在左侧导航栏选择“告警中心”，选择“告警列表”页签。

列表中默认展示全部待解决告警，支持按照告警关键字、告警等级，以及告警发生的时间范围筛选。同时支持查看指定筛选条件的告警在不同时间段的分布情况。

待解决告警若十分钟内不再触发，则会默认已解决并转换为历史告警。如果提前确认某条告警已解决，也可以单击操作列的“清除”，清除后的告警可在历史告警中查询。

图 9-78 告警列表

----结束

9.5.3 通过 CCE 配置自定义告警

当默认的告警规则无法满足您的述求时，可以创建自定义告警规则。通过在CCE中创建告警规则，您可以及时了解集群中各种资源是否存在异常。

添加指标类告警示例

说明

- 基于Prometheus指标的阈值告警规则，指标告警规则依赖开通监控中心，请前往监控中心一键开通。详情请参见**开通监控中心**。
- 部分指标模板依赖**CCE节点故障检测**插件（NPD）进行上报，指标详情请参见**表9-42**。如需要使用相关的告警规则，请确保节点故障检测插件（NPD）已安装且正常运行。

步骤1 登录CCE控制台，单击集群名称进入一个已有的集群。

步骤2 在左侧导航栏选择“告警中心”，切换至“告警规则 > 自定义告警规则”页签，单击“创建告警规则”。

步骤3 设置告警规则，在创建告警规则面板填写配置。

- 规则类型：选择“指标告警”，设置基于Prometheus指标的阈值告警规则。
- 告警模板：不使用模板场景下，需填写手动规则详情。您也可以使用告警模板，快速定义告警规则（PromQL）或基于已有模板进行修改。

- 规则详情：

参数	说明	场景示例
规则名称	自定义告警规则的名称	CoreDNS内存使用率超过百分之八十
描述（可选）	添加告警规则描述。	检查CoreDNS容器内存使用率是否大于80%。
告警规则（PromQL）	输入普罗查询语句。关于如何编写普罗查询语句，请参见**查询示例**。	本例中设置CoreDNS当内存使用率的最大值大于80%产生告警，示例如下： (sum(container_memory_working_set_bytes{image!="", container!="POD",namespace="kube-system",container="coredns"}) BY (cluster_name, node,container, pod , namespace, cluster) / sum(container_spec_memory_limit_bytes{namespace="kube-system", container="coredns"} > 0) BY (cluster_name, node, container, pod , namespace, cluster) * 100) > 80
告警等级	根据重要性选择告警等级，分为“紧急”、“重要”、“次要”、“提示”四个等级。	紧急
持续时长	通过下拉菜单选择告警持续时长，默认为1分钟。	1分钟
告警内容	定义告警通知中的内容，可通过“${变量}”的形式捕获Prometheus中的变量	示例如下： 集群：${cluster_name}/命名空间：${namespace}/Pod：${pod}/容器：${container} 内存使用率超过80%, 当前值${value}%。
联系组	选择一个已有的联系组。您也可以单击“新建联系组”进行创建，配置参数详情请参见**配置告警通知人**。	CCEGroup

上述示例为kube-system空间下的CoreDNS设置一条名为“CoreDNS内存使用率超过百分之八十”的告警规则，告警等级为紧急。当内存使用率的最大值大于80%，且持续了1分钟时，给联系组CCEGroup内的所有告警联系人发送通知（通知方式为短信或邮件）。 通知内容包含集群名称、命名空间、Pod名称、容器名称以及当前的内存使用率。

- 高级设置（可选）
  - 告警标签：添加告警标识性属性，用于告警降噪分组条件，标签值可用在通知内容模板中以*$event.metadate.*标签名被引用。一共可以添加10个告警标签。
  - 告警标注：添加告警非标识性属性，标注值可用在通知内容模板中以*$event.annotations.*标注名被引用。一共可以添加10个告警标注。

步骤4 单击“确定”，然后可前往自定义告警规则列表中查看规则是否创建成功。

----结束

添加事件类告警

说明

- 基于事件触发的告警规则依赖开通日志中心并开启Kubernetes事件采集，前往日志中心一键开通。详情请参见**通过云原生日志采集插件采集容器日志**。
- 部分指标模板依赖**CCE节点故障检测**插件（NPD）进行上报，指标详情请参见**表9-42**。如需要使用相关的告警规则，请确保节点故障检测插件（NPD）已安装且正常运行。

步骤1 登录CCE控制台，单击集群名称进入一个已有的集群。

步骤2 在左侧导航栏选择“告警中心”，切换至“告警规则 > 自定义告警规则”页签，单击“创建告警规则”。

步骤3 设置告警规则，在创建告警规则面板填写配置。

- 规则类型：选择“事件告警”，设置基于事件触发的告警规则，常见事件来源为Kubernetes事件和云服务事件。
- 规则详情：

参数	说明	场景示例
规则名称	自定义告警规则的名称	ReplicaSet副本数变化
描述（可选）	添加告警规则描述。	ReplicaSet副本数在5分钟内变化次数超过3次
事件名称	输入事件的名称，该名称需要与实际产生的Kubernetes事件或云服务事件相匹配。具体事件名称可请参见**CCE事件列表**。	ScalingReplicaSet
触发方式	- 立即触发：只要事件出现即发生告警。 - 累计触发：在指定的监控周期内，累计次数满足数值要求，才会发生告警。	选择“累计触发”，并设置监控周期为“5分钟”，累计次数为“> 3”。
告警等级	根据重要性选择告警等级，分为“紧急”、“重要”、“次要”、“提示”四个等级。	次要
联系组	选择一个已有的联系组。您也可以单击“新建联系组”进行创建，配置参数详情请参见**配置告警通知人**。	CCEGroup

上述示例为ScalingReplicaSet事件设置一条名为“ReplicaSet副本数变化”的告警，告警等级为次要。当5分钟内累计次数超过3次时，CCEGroup内的所有告警联系人发送通知（通知方式为短信或邮件）。

步骤4 单击“确定”，然后可前往自定义告警规则列表中查看规则是否创建成功。

----结束

9.5.4 通过 AOM 配置自定义告警

CCE对接AOM并上报告警和事件，通过在AOM中设置告警规则，您可以及时了解集群中各种资源是否存在异常。

告警配置流程

1. **在SMN创建主题**。
2. **创建行动规则**。
3. 添加告警规则。
   a. 事件类告警：根据集群上报到AOM的事件配置告警。推荐配置的事件和配置方法请参见**添加事件类告警**。
   b. 指标类告警：实时监控环境中主机、组件等资源使用情况，根据监控指标阈值告警。推荐配置阈值指标和配置方法请参见**添加指标类告警**。

在 SMN 创建主题

SMN（Simple Message Notification，消息通知服务）是向订阅者主动推送消息的服务，订阅者可以是电子邮件、短信、HTTP和HTTPS等。

主题是消息发布或客户端订阅通知的特定事件类型。它作为发送消息和订阅通知的信道，为发布者和订阅者提供一个可以相互交流的通道。

您需要创建一个主题，并订阅。具体方法请参见**创建主题**和**订阅主题**。

说明

订阅主题后，请前往您的订阅终端（邮件或短信）手动确认添加订阅，消息通知才可生效。

创建行动规则

AOM提供告警行动规则定制功能，您可以通过创建告警行动规则关联SMN主题与消息模板，通过创建消息模板，自定义通知消息配置。

具体方法请参见**创建告警行动规则**。创建时选择**在SMN创建主题**创建并订阅的主题。

添加事件类告警

以添加“节点状态异常告警”为例，展示添加事件类告警的步骤，您可以参考**表9-43**添加其他告警。

表 9-43 推荐配置的事件类告警

事件名称	来源	事件说明	处理建议
节点状态异常	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点
节点重启	CCE	节点重启立即触发告警	登录集群查看告警节点状态，并确保节点正常启动可用，关注重启原因
节点kubelet故障	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点；重启kubelet
节点docker故障	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点；重启docker
节点kube-proxy故障	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点
节点操作系统内核故障	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点
节点的连接跟踪表已满	CCE	节点异常立即触发告警	登录集群查看告警节点状态，确认异常后，优先将此节点设置为不可调度，并将业务pod调度到其他节点
节点池资源售罄	CCE	节点池资源售罄立即告警	设置自动节点池切换或更换节点池规格
节点创建失败	CCE	创建节点失败立即触发	查看创建节点失败原因，尝试重新创建节点
扩容节点超时	CCE	扩容节点超时立即触发	查看扩容节点失超时原因，尝试重新扩容节点
缩容节点失败	CCE	缩容节点超时立即触发	查看缩容节点失超时原因，尝试重新缩容节点
拉取镜像重试失败	CCE	拉取镜像重试失败	登录集群查看拉取镜像失败原因，重新部署业务负载

步骤1 登录AOM 2.0控制台。

步骤2 在左侧导航栏选择“告警管理 > 告警规则”，单击“创建告警规则”。

步骤3 根据页面提示填写基本信息后，设置告警规则。关键参数如下：

详细参数说明请参见**创建事件类告警规则**。

- 规则类型：选择“事件告警规则”。
- 事件类型：选择“系统事件”。
- 事件来源：选择“CCE”。
- 监控对象：监控对象可以通过多个维度（通知类型、事件名称、告警级别、自定义属性、命名空间、集群名称）进行筛选，您可以根据需要选择。

  本示例中根据“事件名称”进行筛选，选择“节点状态异常”事件，触发方式选择“立即触发”。
- 告警方式：选择“直接告警”。
- 行动规则：选择**创建行动规则**步骤中创建的行动规则。

其余参数可按需求配置。

本示例中的设置的告警为：

当集群中存在节点状态异常时，CCE会上报“节点状态异常”的事件到AOM，AOM根据设置的告警规则，立即触发告警通知，并根据行动规则，通过SMN通知您。

图 9-79 创建事件类告警

步骤4 单击“立即创建”。

创建后在规则列表中可以看到对应的告警规则，表示创建成功。

----结束

添加指标类告警

以使用PromQL语句配置告警规则为例，展示添加指标类告警的步骤。

步骤1 登录AOM 2.0控制台。

步骤2 在左侧导航栏选择“告警管理 > 告警规则”，单击“创建告警规则”。

步骤3 设置告警规则，关键参数如下。

详细参数说明请参见**创建指标告警规则**。

- 规则类型：选择“指标告警规则”。
- 配置方式：选择“PromQL”。采用原生PromQL语句配置告警规则，CCE提供了告警规则模板，可供您选择。

- Prometheus实例：选择集群中“云原生监控插件”上报指标的AOM实例。
- 默认规则：
  - 自定义：输入自定义PromQL语句配置告警规则，例如：
    ```
    kube_persistentvolume_status_phase{phase=~"Failed|Pending",cluster="${cluster_id}"} > 0
    ```
    其中*${cluster_id}*为集群名称，表示当集群中有PV处于Failed或Pending状态时产生告警。
  - CCEFromProm：选择CCE提供的告警模板。

图 9-80 指标类告警

- 告警方式：选择“直接告警”。
- 行动规则：选择**创建行动规则**步骤中创建的行动规则。

其余参数可按需求配置。

步骤4 单击“立即创建”。

创建后在规则列表中可以看到对应的告警规则，表示创建成功。

----结束

9.5.5 CCE 事件列表

在集群运行过程中，CCE会上报一系列事件至AOM，您可以根据自身需求添加事件类告警，监控集群数据面和控制面组件的健康状态，及时发现和解决问题，保证集群的稳定性和可靠性。

- **集群数据面事件**：集群运行过程中与用户操作相关的事件，包括工作负载、网络、节点、存储、弹性伸缩等事件。
  - **工作负载相关事件**
  - **网络相关事件**
  - **节点相关事件**
  - **存储相关事件**
  - **弹性伸缩相关事件**
- **集群控制面事件**：集群运行过程中控制节点上报的事件，这些事件通常是由于控制面组件的故障、升级等情况引起。

集群数据面事件

表 9-44 工作负载相关事件

类别	事件描述	事件名称	事件级别	更多说明
Pod	Pod内存不足OOM	PodOOMKilling	重要	检查Pod是否因OOM退出。 该事件依赖节点故障检测插件（1.18.41及以上版本）和云原生日志采集插件（1.3.2及以上版本）。
Pod	启动失败	FailedStart	重要	检查Pod是否启动成功。
Pod	拉取镜像失败	FailedPullImage	重要	检查Pod是否拉取镜像成功。
Pod	启动重试失败	BackOffStart	重要	检查Pod是否重启失败。
Pod	调度失败	FailedScheduling	重要	检查Pod是否调度成功。
Pod	拉取镜像重试失败	BackOffPullImage	重要	检查Pod重试拉取镜像是否成功。
Pod	创建失败	FailedCreate	重要	检查Pod创建是否成功。
Pod	状态异常	Unhealthy	次要	检查Pod健康检查是否成功。
Pod	删除失败	FailedDelete	次要	检查工作负载是否删除成功。
Pod	未拉取镜像异常	ErrImageNeverPull	次要	检查工作负载是否拉取镜像。
Pod	扩容失败	FailedScaleOut	次要	检查工作负载副本扩容是否正常。
Pod	待机失败	FailedStandBy	次要	检查Pod待机是否成功。
Pod	更新配置失败	FailedReconfig	次要	检查Pod更新配置是否成功。
Pod	激活失败	FailedActive	次要	检查Pod是否激活成功。
Pod	回滚失败	FailedRollback	次要	检查Pod回滚是否成功。
Pod	更新失败	FailedUpdate	次要	检查Pod更新是否成功。
Pod	缩容失败	FailedScaleIn	次要	检查Pod缩容是否失败。
Pod	重启失败	FailedRestart	次要	检查Pod重启是否成功。
Deployment	标签选择器冲突	SelectorOverlap	次要	检查集群中标签选择器是否存在冲突。

类别	事件描述	事件名称	事件级别	更多说明
Deployment	副本集创建异常	ReplicaSetCreateError	次要	检查工作负载ReplicaSet创建副本是否正常。
Deployment	部署回滚版本未发现	DeploymentRollbackRevisionNotFound	次要	检查Deployment负载回滚版本是否存在。
DaemonSet	标签选择器异常	SelectingAll	次要	检查工作负载标签选择器是否设置异常。
Job	太多活跃Pod	TooManyActivePods	次要	检查Job达到预定的Pod数后，是否还存在活动状态的Pod。
Job	太多成功Pod	TooManySucceededPods	次要	检查Job达到预定的数量后，是否存在过多运行成功的Pod。
CronJob	查询失败	FailedGet	次要	查询CronJob是否成功。
CronJob	查询Pod列表失败	FailedList	次要	检查查询Pod列表是否成功。
CronJob	未知Job	UnexpectedJob	次要	检查CronJob是否出现未知的Job。

表 9-45 网络相关事件

类别	事件描述	事件名称	事件级别	更多说明
Service	创建负载均衡失败	CreatingLoadBalancerFailed	次要	检查创建ELB是否成功。
Service	删除负载均衡失败	DeletingLoadBalancerFailed	次要	检查删除ELB是否成功。
Service	更新负载均衡失败	UpdateLoadBalancerFailed	次要	检查更新ELB是否成功。

表 9-46 节点相关事件

类别	事件描述	事件名称	事件级别	更多说明
Node	节点重启	Rebooted	重要	检查节点是否重启。

类别	事件描述	事件名称	事件级别	更多说明
Node	节点不可调度	NodeNotSchedulable	重要	检查节点是否可调度。
Node	节点状态异常	NodeNotReady	重要	检查节点状态是否异常。
Node	节点创建失败	NodeCreateFailed	重要	检查节点是否创建成功。
Node	节点kubelet故障	KUBELETIsDown	次要	检查节点kubelet是否正常。
Node	节点内存空间不足	NodeHasInsufficientMemory	次要	检查节点内存空间是否充足。
Node	节点上发现未注册的网络设备	UnregisterNetDevice	次要	检查节点上是否绑定了未注册的网络设备。
Node	网卡未发现	NetworkCardNotFound	次要	检查节点网卡状态。
Node	节点kube-proxy故障	KUBEPROXYIsDown	次要	检查节点上的kube-proxy是否正常。
Node	节点磁盘空间已满	NodeOutOfDisk	次要	检查节点磁盘空间是否正常。
Node	节点任务夯住	TaskHung	次要	检查节点上是否存在夯住的任务。
Node	CIDR不可用	CIDRNotAvailable	次要	检查节点CIDR是否可用。
Node	节点的连接跟踪表已满	ConntrackFull	次要	检查节点的连接跟踪表是否已满。
Node	节点磁盘空间不足	NodeHasDiskPressure	次要	检查节点磁盘空间是否充足。
Node	节点纳管失败	NodeInstallFailed	次要	检查集群纳管节点是否成功。
Node	节点操作系统内核故障	KernelOops	次要	检查节点操作系统内核是否故障。
Node	节点内存不足强杀进程	OOMKilling	次要	• 节点上的Pod内存使用超过Limit值导致进程终止。 • 节点上的Pod内存使用未超过Limit值，但节点可用内存不足出现OOM。

类别	事件描述	事件名称	事件级别	更多说明
Node	节点docker故障	DOCKERIsDown	次要	检查节点容器引擎是否正常。
Node	CIDR分配失败	CIDRAssignmentFailed	次要	检查节点CIDR分配是否成功。
Node	节点docker夯住	DockerHung	次要	检查节点Docker进程是否夯住。
Node	节点文件系统只读	FilesystemIsReadOnly	次要	检查节点文件系统是否只读。
Node	节点ntp服务故障	NTPIsDown	次要	检查节点NTP服务是否正常。
Node	节点卸载失败	NodeUninstallFailed	次要	检查节点卸载是否成功。
Node	节点磁盘卸载夯住	AUFSUmountHung	次要	检查节点磁盘卸载是否夯住。
Node	节点cni插件故障	CNIIsDown	次要	检查节点CNI插件是否故障。
Namespace	废弃节点清理	DeleteNodeWithNoServer	次要	检查是否清理废弃节点。

表 9-47 存储相关事件

类别	事件描述	事件名称	事件级别	更多说明
PV	主机卸载块存储失败	DetachVolumeFailed	次要	检查卸载块存储是否成功。
PV	卷回收策略未知	VolumeUnknownReclaimPolicy	次要	检查是否指定卷回收策略。
PV	挂载数据卷失败	SetUpAtVolumeFailed	次要	检查数据卷挂载是否成功。
PV	数据卷回收失败	VolumeFailedRecycle	次要	检查数据卷是否成功回收。
PV	等待主机挂载块存储失败	WaitForAttachVolumeFailed	次要	检查节点挂载块存储是否成功。
PV	数据卷删除失败	VolumeFailedDelete	次要	检查数据卷删除是否成功。

类别	事件描述	事件名称	事件级别	更多说明
PV	挂载盘符失败	MountDeviceFailed	次要	检查数据卷挂盘是否成功。
PV	卸载数据卷失败	TearDownAtVolumeFailed	次要	检查数据卷卸载是否成功。
PV	卸载盘符失败	UnmountDeviceFailed	次要	检查数据卷卸载盘符是否成功。
PV	主机挂载块存储失败	AttachVolumeFailed	次要	检查节点卸载块存储是否成功。
PVC	数据卷扩容失败	VolumeResizeFailed	次要	检查数据卷扩容是否成功。
PVC	卷PVC丢失	ClaimLost	次要	检查PVC卷是否正常。
PVC	创建卷失败	ProvisioningFailed	次要	检查创建数据卷是否正常。
PVC	创建卷清理失败	ProvisioningCleanupFailed	次要	检查清理数据卷是否正常。
PVC	卷误绑定	ClaimMisbound	次要	检查PVC是否绑定错误的卷。

表 9-48 弹性伸缩相关事件

类别	事件描述	事件名称	事件级别	更多说明
Autoscaler	扩容节点超时	ScaleUpTimedOut	重要	检查节点池扩容节点是否超时。
Autoscaler	节点池资源充足	NodePoolAvailable	重要	检查节点池资源是否充足。
Autoscaler	缩容节点	ScaleDown	重要	集群正在缩容节点。
Autoscaler	未触发节点扩容	NotTriggerScaleUp	重要	检查节点是否成功触发扩容。
Autoscaler	删除未注册节点成功	DeleteUnregistered	重要	检查删除未注册的节点是否成功。
Autoscaler	缩容空闲节点成功	ScaleDownEmpty	重要	检查缩容空闲节点是否成功。
Autoscaler	缩容节点失败	ScaleDownFailed	重要	检查缩容节点是否成功。

类别	事件描述	事件名称	事件级别	更多说明
Autoscaler	节点池扩容节点失败	FailedToScaleUpGroup	重要	检查节点池扩容节点是否异常。
Autoscaler	节点池扩容节点成功	ScaledUpGroup	重要	检查节点池扩容是否成功。
Autoscaler	扩容节点失败	ScaleUpFailed	重要	检查节点扩容是否成功。
Autoscaler	修复节点池节点个数成功	FixNodeGroupSizeDone	重要	检查修复节点池节点个数是否成功。
Autoscaler	节点池退避重试中	NodeGroupInBackOff	重要	检查节点池扩缩容是否存在回退重试。
Autoscaler	修复节点池节点个数失败	FixNodeGroupSizeError	重要	检查修复节点池节点个数是否成功。
Autoscaler	节点池资源售罄	NodePoolSoldOut	重要	检查节点池资源是否充足。
Autoscaler	触发节点扩容	TriggeredScaleUp	重要	检查节点是否触发扩容。
Autoscaler	节点池扩容节点启动	StartScaledUpGroup	重要	检查节点池扩容是否启动。
Autoscaler	删除未注册节点失败	DeleteUnregisteredFailed	重要	检查删除未注册的节点是否成功。
HPA	HPA非法指标范围	InvalidTargetRange	重要	• HPA的annotations中配置了非法的extendedhpa.metrics。 • HPA的spec中metric type填写错误。
HPA	HPA获取伸缩对象失败	FailedGetScale	重要	HPA无法获取待伸缩的资源对象。
HPA	HPA计算资源扩缩副本数失败	FailedComputeMetricsReplicas	重要	一般是由于在计算需要为资源调整多少个副本数时出现了问题，例如metric-server不可用、资源指标采集失败、CPU利用率等设置不正确等。 可以通过以下命令查看详细的信息： `kubectl describe horizontalpodautoscaler <hpa-name>`
HPA	HPA获取对象指标失败	FailedGetObjectMetric	重要	获取指定对象（PVC、ConfigMaps等）的指标失败。

类别	事件描述	事件名称	事件级别	更多说明
HPA	HPA获取Pod资源指标失败	FailedGetPodsMetric	重要	获取Pod资源指标失败（单个Pod的资源利用率）。
HPA	HPA获取集群资源指标失败	FailedGetResourceMetric	重要	获取集群资源指标失败（整个集群的资源利用率）。
HPA	HPA获取容器资源指标失败	FailedGetContainerResourceMetric	重要	获取单个容器资源指标失败。
HPA	HPA获取外部指标失败	FailedGetExternalMetric	重要	获取外部指标失败。
HPA	HPA伸缩Pod失败	FailedRescale	重要	更新待伸缩资源对象的期望副本数失败。
HPA	Pod扩缩容成功	SuccessfulRescale	次要	更新待伸缩资源对象的期望副本数成功。
CronHPA	CronHPA伸缩失败	ScaleFailed	重要	CronHPA更新待伸缩资源对象的期望副本数失败。
CronHPA	CronHPA查询关联HPA失败	FailedGetHorizontalPodAutoscaler	重要	CronHPA查询关联的HPA对象失败（通常是kube-apiserver侧响应失败）。
CronHPA	CronHPA查询伸缩对象失败	FailedGetHpaScale	重要	CronHPA获取待伸缩资源对象失败。
CronHPA	CronHPA更新关联HPA失败	UpdateHPAFailed	重要	CronHPA更新关联的HPA对象失败。
CronHPA	更新HPA策略成功	UpdateHPASuccess	次要	CronHPA更新关联的HPA对象成功。
CronHPA	跳过更新HPA策略	SkipUpdateHPA	次要	CronHPA跳过更新关联的HPA对象。
CronHPA	跳过更新工作负载实例数	SkipUpdateTarget	次要	CronHPA跳过更新待伸缩资源对象的副本数。
CronHPA	更新工作负载实例数成功	UpdateTargetSuccess	次要	CronHPA更新待伸缩资源对象的副本数成功。
CustomedHPA	CustomedHPA解析冷却时间失败	FailedSetPolicySettings	重要	解析CustomedHPA的冷却时间失败。

类别	事件描述	事件名称	事件级别	更多说明
CustomedHPA	CustomedHPA处理定时/指标规则失败	FailedSubmitRule	重要	CustomedHPA处理定时规则或指标规则失败。
CustomedHPA	CustomedHPA计算资源扩缩副本数失败	FailedComputeReplicas	重要	CustomedHPA计算指标触发资源扩缩容失败。
CustomedHPA	CustomedHPA伸缩Pod失败	FailedScale	重要	CustomedHPA更新待伸缩资源对象的期望副本数失败（通常是kube-apiserver侧响应失败）。
CustomedHPA	CustomedHPA指标扩缩容成功	MetricScaleSuccess	次要	CustomedHPA根据指标规则触发资源扩缩容成功。
CustomedHPA	CustomedHPA周期扩缩容成功	CronScaleSuccess	次要	CustomedHPA根据周期规则触发资源扩缩容成功。

集群控制面事件

表 9-49 集群控制面事件

事件名称	事件ID	事件级别	事件说明
内部故障	Internal error	重要	检查集群是否出现内部故障。
外部依赖异常	External dependency error	重要	检查集群是否存在外部依赖异常。
初始化执行线程失败	Failed to initialize process thread	重要	检查集群初始化执行线程是否成功。
更新数据库失败	Failed to update database	重要	检查集群更新数据库是否成功。
节点池触发创建节点失败	Failed to create node by nodepool	重要	检查节点池中创建节点是否成功。
节点池触发删除节点失败	Failed to delete node by nodepool	重要	检查节点池中删除节点是否成功。

事件名称	事件ID	事件级别	事件说明
创建包周期节点失败	Failed to create yearly/monthly subscription node	重要	检查集群创建包周期的节点是否成功。
解除资源租户访问控制节点镜像的授权失败	Failed to cancel the authorization of accessing the image of the master.	重要	创建集群时，检查解除资源租户访问控制节点镜像的授权是否成功。
创建虚拟IP失败	Failed to create the virtual IP for the master	重要	创建集群时检查创建虚拟IP是否成功。
删除节点虚拟机失败	Failed to delete the node VM	重要	检查集群删除节点虚拟机是否成功。
删除节点安全组失败	Failed to delete the security group of node	重要	检查集群删除节点安全组是否成功。
删除控制节点安全组失败	Failed to delete the security group of master	重要	检查集群删除控制节点安全组是否成功。
删除控制节点网卡安全组失败	Failed to delete the security group of port	重要	检查集群删除控制节点网卡安全组是否成功。
删除集群ENI/SubENI安全组失败	Failed to delete the security group of eni or subeni	重要	检查集群删除ENI/SubENI安全组是否成功。
解绑控制节点网卡失败	Failed to detach the port of master	重要	检查集群解绑控制节点网卡是否成功。
删除控制节点网卡失败	Failed to delete the port of master	重要	检查集群删除控制节点网卡是否成功。
删除控制节点虚拟机失败	Failed to delete the master VM	重要	检查集群删除控制节点虚拟机是否成功。
删除控制节点密钥对失败	Failed to delete the key pair of master	重要	检查集群删除控制节点密钥对是否成功。
删除控制节点subnet失败	Failed to delete the subnet of master	重要	检查集群删除控制节点subnet是否成功。
删除控制节点VPC失败	Failed to delete the VPC of master	重要	检查集群删除控制节点VPC是否成功。

事件名称	事件ID	事件级别	事件说明
删除集群证书失败	Failed to delete certificate of cluster	重要	检查集群删除集群证书是否成功。
删除控制节点云服务器组失败	Failed to delete the server group of master	重要	检查集群删除控制节点云服务器组是否成功。
删除虚拟IP失败	Failed to delete the virtual IP for the master	重要	检查集群删除虚拟IP是否成功。
获取控制节点浮动IP失败	Failed to get floating IP of the master	重要	检查获取控制节点浮动IP是否成功。
获取集群规格信息失败	Failed to get cluster flavor	重要	检查获取集群规格信息是否成功。
获取集群endpoint失败	Failed to get cluster endpoint	重要	检查获取集群endpoint是否成功。
获取Kubernetes集群连接失败	Failed to get kubernetes connection	重要	检查获取Kubernetes集群连接是否成功。
更新集群Secret失败	Failed to update secret	重要	检查更新集群Secret是否成功。
处理用户操作超时	Operation timed out	重要	检查处理用户操作是否超时。
连接Kubernetes集群超时	Connecting to Kubernetes cluster timed out	重要	检查连接Kubernetes集群是否超时。
检查组件状态失败或组件状态异常	Failed to check component status or components are abnormal	重要	检查集群检查组件状态是否成功，或组件状态是否异常。
无法在Kubernetes集群中找到该节点	The node is not found in kubernetes cluster	重要	检查是否能在Kubernetes集群中找到该节点。
节点在Kubernetes集群中状态异常	The status of node is not ready in kubernetes cluster	重要	检查节点在Kubernetes集群中状态是否正常。

事件名称	事件ID	事件级别	事件说明
无法在ECS服务中找到该节点对应的虚拟机	Can't find corresponding vm of this node in ECS	重要	检查能否在ECS服务中找到该节点对应的虚拟机。
升级控制节点失败	Failed to upgrade the master	重要	检查升级控制节点是否成功。
升级节点失败	Failed to upgrade the node	重要	检查升级节点是否成功。
变更控制节点规格失败	Failed to change flavor of the master	重要	检查变更控制节点规格是否成功。
变更控制节点规格超时	Change flavor of the master timeout	重要	检查变更控制节点规格是否超时。
创建包周期节点校验不通过	Failed to pass verification while creating yearly/monthly subscription node	重要	检查创建包周期节点校验是否成功。
安装节点失败	Failed to install the node	重要	检查集群安装节点是否成功。
清理VPC中集群容器网络路由表条目失败	Failed to clean routes of cluster container network in VPC	重要	检查清理VPC中集群容器网络路由表条目是否成功。
集群状态不可用	Cluster status is Unavailable	重要	检查集群状态是否可用。
集群状态故障	Cluster status is Error	重要	检查集群是否出现故障。
集群状态长时间不更新	Cluster status is not updated for a long time	重要	检查集群状态是否长时间不更新。
集群升级超时后更新控制节点状态失败	Failed to update master status after upgrading cluster timeout	重要	检查集群升级超时后更新控制节点状态是否成功。
集群升级超时后更新运行中的任务失败	Failed to update running jobs after upgrading cluster timeout	重要	检查集群升级超时后更新运行中的任务是否成功。
更新集群状态失败	Failed to update cluster status	重要	检查更新集群状态是否成功。

事件名称	事件ID	事件级别	事件说明
更新节点状态失败	Failed to update node status	重要	检查更新节点状态是否成功。
纳管节点超时后移除数据库中的节点记录失败	Failed to remove the static node from database	重要	检查纳管节点超时后移除数据库中的节点记录是否成功。
节点处理超时后更新节点状态为异常失败	Failed to update node status to abnormal after node processing timeout	重要	检查节点处理超时后，是否更新节点状态为异常。
更新集群访问地址失败	Failed to update the cluster endpoint	重要	检查更新集群访问地址是否成功。
删除不可用的Kubernetes连接失败	Failed to delete the unavailable connection of the Kubernetes cluster	重要	检查删除不可用的Kubernetes连接是否成功。
同步集群证书失败	Failed to sync the cluster cert	重要	检查同步集群证书是否成功。

9.6 日志审计

9.6.1 云审计服务支持的 CCE 操作列表

CCE通过云审计服务（Cloud Trace Service，简称CTS）为您提供云服务资源的操作记录，记录内容包括您从云管理控制台或者开放API发起的云服务资源操作请求以及每次请求的结果，供您查询、审计和回溯使用。

表 9-50 云审计服务支持的 CCE 操作列表

操作名称	资源类型	事件名称
创建用户委托	集群	createUserAgencies
创建集群	集群	createCluster
创建包周期集群	集群	createCluster/createPeriodicCluster
更新集群描述	集群	updateCluster
升级集群	集群	clusterUpgrade
删除集群	集群	claimCluster/deleteCluster
下载集群证书	集群	getClusterCertByUID

操作名称	资源类型	事件名称
绑定、解绑eip	集群	operateMasterEIP
集群休眠唤醒、节点纳管重置（V2）	集群	operateCluster
集群休眠（V3）	集群	hibernateCluster
集群唤醒（V3）	集群	awakeCluster
按需集群规格变更	集群	resizeCluster
包周期集群规格变更	集群	resizePeriodCluster
修改集群配置	集群	updateConfiguration
创建节点池	节点池	createNodePool
更新节点池	节点池	updateNodePool
删除节点池	节点池	claimNodePool
迁移节点池	节点池	migrateNodepool
修改节点池配置	节点池	updateConfiguration
创建节点	节点	createNode
创建包周期节点	节点	createPeriodNode
删除集群下所有节点	节点	deleteAllHosts
删除单个节点	节点	deleteOneHost/claimOneHost
更新节点描述	节点	updateNode
创建插件实例	插件实例	createAddonInstance
删除插件实例	插件实例	deleteAddonInstance
上传模板	模板	uploadChart
更新模板	模板	updateChart
删除模板	模板	deleteChart
创建模板实例	模板实例	createRelease
升级模板实例	模板实例	updateRelease
删除模板实例	模板实例	deleteRelease
创建ConfigMap	Kubernetes资源	createConfigmaps
创建DaemonSet	Kubernetes资源	createDaemonsets
创建Deployment	Kubernetes资源	createDeployments
创建Event	Kubernetes资源	createEvents

操作名称	资源类型	事件名称
创建Ingress	Kubernetes资源	createIngresses
创建Job	Kubernetes资源	createJobs
创建namespace	Kubernetes资源	createNamespaces
创建Node	Kubernetes资源	createNodes
创建PersistentVolumeClaim	Kubernetes资源	createPersistentvolumeclaims
创建Pod	Kubernetes资源	createPods
创建ReplicaSet	Kubernetes资源	createReplicasets
创建ResourceQuota	Kubernetes资源	createResourcequotas
创建密钥	Kubernetes资源	createSecrets
创建服务	Kubernetes资源	createServices
创建StatefulSet	Kubernetes资源	createStatefulsets
创建卷	Kubernetes资源	createVolumes
删除ConfigMap	Kubernetes资源	deleteConfigmaps
删除DaemonSet	Kubernetes资源	deleteDaemonsets
删除Deployment	Kubernetes资源	deleteDeployments
删除Event	Kubernetes资源	deleteEvents
删除Ingress	Kubernetes资源	deleteIngresses
删除Job	Kubernetes资源	deleteJobs
删除Namespace	Kubernetes资源	deleteNamespaces
删除Node	Kubernetes资源	deleteNodes
删除Pod	Kubernetes资源	deletePods
删除ReplicaSet	Kubernetes资源	deleteReplicasets
删除ResourceQuota	Kubernetes资源	deleteResourcequotas
删除Secret	Kubernetes资源	deleteSecrets
删除Service	Kubernetes资源	deleteServices
删除StatefulSet	Kubernetes资源	deleteStatefulsets
删除卷	Kubernetes资源	deleteVolumes
替换指定的ConfigMap	Kubernetes资源	updateConfigmaps

操作名称	资源类型	事件名称
替换指定的DaemonSet	Kubernetes资源	updateDaemonsets
替换指定的Deployment	Kubernetes资源	updateDeployments
替换指定的Event	Kubernetes资源	updateEvents
替换指定的Ingress	Kubernetes资源	updateIngresses
替换指定的Job	Kubernetes资源	updateJobs
替换指定的Namespace	Kubernetes资源	updateNamespaces
替换指定的Node	Kubernetes资源	updateNodes
替换指定的PersistentVolumeClaim	Kubernetes资源	updatePersistentvolumeclaims
替换指定的Pod	Kubernetes资源	updatePods
替换指定的Replicaset	Kubernetes资源	updateReplicasets
替换指定的ResourceQuota	Kubernetes资源	updateResourcequotas
替换指定的Secret	Kubernetes资源	updateSecrets
替换指定的Service	Kubernetes资源	updateServices
替换指定的Statefulset	Kubernetes资源	updateStatefulsets
替换指定的状态	Kubernetes资源	updateStatus
上传组件模板	Kubernetes资源	uploadChart
更新组件模板	Kubernetes资源	updateChart
删除组件模板	Kubernetes资源	deleteChart
创建模板应用	Kubernetes资源	createRelease
更新模板应用	Kubernetes资源	updateRelease
删除模板应用	Kubernetes资源	deleteRelease

9.6.2 在 CTS 事件列表查看云审计事件

操作场景

用户进入云审计服务创建管理类追踪器后，系统开始记录云服务资源的操作。在创建数据类追踪器后，系统开始记录用户对OBS桶中数据的操作。云审计服务管理控制台会保存最近7天的操作记录。

说明

云审计控制台对用户的操作事件日志保留7天，过期自动删除，不支持人工删除。

本节介绍如何在云审计服务管理控制台查看或导出最近7天的操作记录：

- **在新版事件列表查看审计事件**
- **在旧版事件列表查看审计事件**

使用限制

- 单账号跟踪的事件可以通过云审计控制台查询。多账号的事件只能在账号自己的事件列表页面去查看，或者到组织追踪器配置的OBS桶中查看，也可以到组织追踪器配置的CTS/system日志流下面去查看。
- 用户通过云审计控制台只能查询最近7天的操作记录。如果需要查询超过7天的操作记录，您必须配置转储到对象存储服务(OBS)或云日志服务(LTS)，才可在OBS桶或LTS日志组里面查看历史事件信息。否则，您将无法追溯7天以前的操作记录。
- 云上操作后，1分钟内可以通过云审计控制台查询管理类事件操作记录，5分钟后才可通过云审计控制台查询数据类事件操作记录。

在新版事件列表查看审计事件

1. 登录管理控制台。
2. 单击左上角 ☰ ，选择“管理与监管 > 云审计服务 CTS”，进入云审计服务页面。
3. 单击左侧导航树的“事件列表”，进入事件列表信息页面。
4. 事件列表支持通过高级搜索来查询对应的操作事件，您可以在筛选器组合一个或多个筛选条件：
   - 事件名称：输入事件的名称。
   - 事件ID：输入事件ID。
   - 资源名称：输入资源的名称，当该事件所涉及的云资源无资源名称或对应的API接口操作不涉及资源名称参数时，该字段为空。
   - 资源ID：输入资源ID，当该资源类型无资源ID或资源创建失败时，该字段为空。
   - 云服务：在下拉框中选择对应的云服务名称。
   - 资源类型：在下拉框中选择对应的资源类型。
   - 操作用户：在下拉框中选择一个或多个具体的操作用户。
   - 事件级别：可选项为“normal”、“warning”、“incident”，只可选择其中一项。

- normal：表示操作成功。
- warning：表示操作失败。
- incident：表示比操作失败更严重的情况，例如引起其他故障等。

– 企业项目ID：输入企业项目ID。
– 访问密钥ID：输入访问密钥ID（包含临时访问凭证和永久访问密钥）。
– 时间范围：可选择查询最近1小时、最近1天、最近1周的操作事件，也可以自定义最近7天内任意时间段的操作事件。

5. 在事件列表页面，您还可以导出操作记录文件、刷新列表、设置列表展示信息等。
   – 在搜索框中输入任意关键字，按下Enter键，可以在事件列表搜索符合条件的数据。
   – 单击“导出”按钮，云审计服务会将查询结果以.xlsx格式的表格文件导出，该.xlsx文件包含了本次查询结果的所有事件，且最多导出5000条信息。
   – 单击按钮，可以获取到事件操作记录的最新信息。
   – 单击按钮，可以自定义事件列表的展示信息。启用表格内容折行开关，可让表格内容自动折行，禁用此功能将会截断文本，默认停用此开关。
6. 关于事件结构的关键字段详解，请参见**事件结构**和**事件样例**。
7. （可选）在新版事件列表页面，单击右上方的“返回旧版”按钮，可切换至旧版事件列表页面。

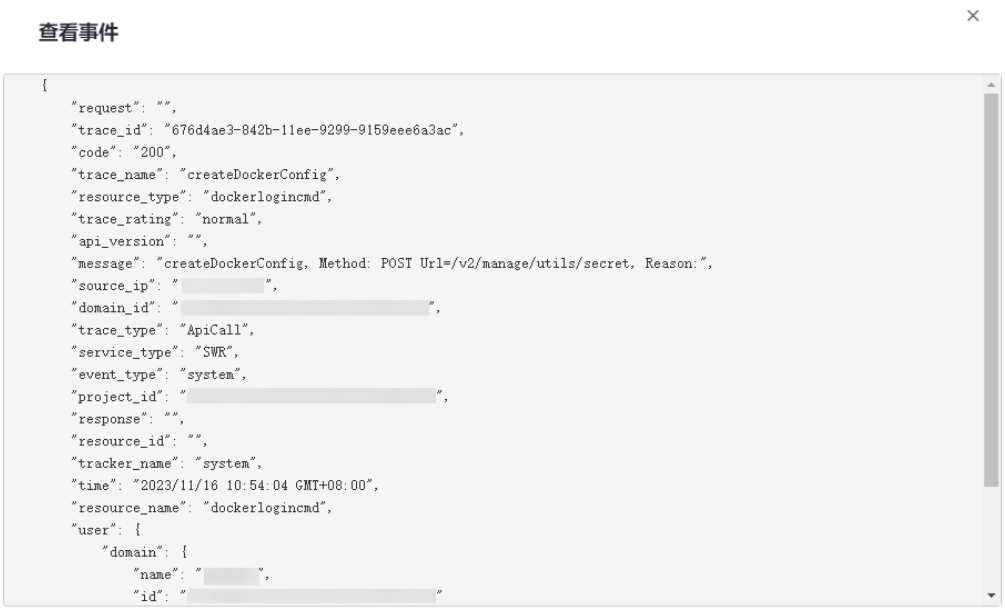
在旧版事件列表查看审计事件

1. 登录管理控制台。
2. 单击左上角，选择“管理与监管 > 云审计服务 CTS”，进入云审计服务页面。
3. 单击左侧导航树的“事件列表”，进入事件列表信息页面。
4. 用户每次登录云审计控制台时，控制台默认显示新版事件列表，单击页面右上方的“返回旧版”按钮，切换至旧版事件列表页面。
5. 事件列表支持通过筛选来查询对应的操作事件。当前事件列表支持四个维度的组合查询，详细信息如下：
   – 事件类型、事件来源、资源类型和筛选类型，在下拉框中选择查询条件。
     - 筛选类型按资源ID筛选时，还需手动输入某个具体的资源ID。
     - 筛选类型按事件名称筛选时，还需选择某个具体的事件名称。
     - 筛选类型按资源名称筛选时，还需选择或手动输入某个具体的资源名称。
   – 操作用户：在下拉框中选择某一具体的操作用户，此操作用户指用户级别，而非租户级别。
   – 事件级别：可选项为“所有事件级别”、“Normal”、“Warning”、“Incident”，只可选择其中一项。

- 时间范围：可选择查询最近1小时、最近1天、最近1周的操作事件，也可以自定义最近7天内任意时间段的操作事件。
- 单击“导出”按钮，云审计服务会将查询结果以CSV格式的表格文件导出，该CSV文件包含了本次查询结果的所有事件，且最多导出5000条信息。

6. 选择完查询条件后，单击“查询”。
7. 在事件列表页面，您还可以导出操作记录文件和刷新列表。
   - 单击“导出”按钮，云审计服务会将查询结果以CSV格式的表格文件导出，该CSV文件包含了本次查询结果的所有事件，且最多导出5000条信息。
   - 单击按钮，可以获取到事件操作记录的最新信息。
8. 在需要查看的事件左侧，单击 展开该记录的详细信息。

9. 在需要查看的记录右侧，单击“查看事件”，会弹出一个窗口显示该操作事件结构的详细信息。



10. 关于事件结构的关键字段详解，请参见《云审计服务用户指南》中的**事件结构**和**事件样例**。
11. （可选）在旧版事件列表页面，单击右上方的“体验新版”按钮，可切换至新版事件列表页面。

9.7 可观测性 FAQ

9.7.1 计费相关 FAQ

索引

可观测性（监控中心、日志中心、告警中心）如何收费?

免费场景

监控中心自身免费使用，监控中心所使用的指标都上报并存储在AOM服务，其中在AOM范畴内的基础指标不收费，存储时长15天（暂不支持修改）。详情请参见**基础指标**。

日志中心自身免费使用，集群内产生的日志都上报并存储在LTS服务，云日志服务的计费项由日志读写流量、日志索引流量、日志存储量的费用组成（有500MB/月的免费额度）。

告警中心自身免费使用，集群内产生的告警由SMN消息通知服务进行推送，在短信数量小于100/条/月、邮件数量小于1000/封/月的，推送免费。其中短信条数计算规则请参见**短信内容长度计算规则**。

收费场景

监控中心：集群内配置的除基础指标以外的自定义指标（基础指标不收费）会根据AOM的收费规则进行收费。详情请参见**计费项**。

日志中心：对超出每月免费限额（500MB/月）的日志读写、日志索引流量、日志存储量进行收费。详情请参见**LTS服务的收费标准**。

告警中心：告警中心依赖SMN消息通知服务对告警进行推送，联系组当前只支持短信和邮件，超过SMN免费范围(短信数量小于100/条/月、邮件数量小于1000/封/月的，推送免费)的将会收费。详情请参见**SMN计费说明**。

为什么关闭日志中心后还有收费产生?

日志中心的收费由LTS进行收费，日志中心关闭之后，不会产生读写和索引的费用，但会收取日志存储时长内的费用，如将日志存储时间修改为1天，1天以后将不会产生存储费用。详情请参见**日志如何计费**。修改方法如下：

在LTS页面找到对应的集群的日志组（以集群ID命名），单击“修改”，在弹出的“修改日志组”页面修改日志存储时间。

图 9-81 修改日志组

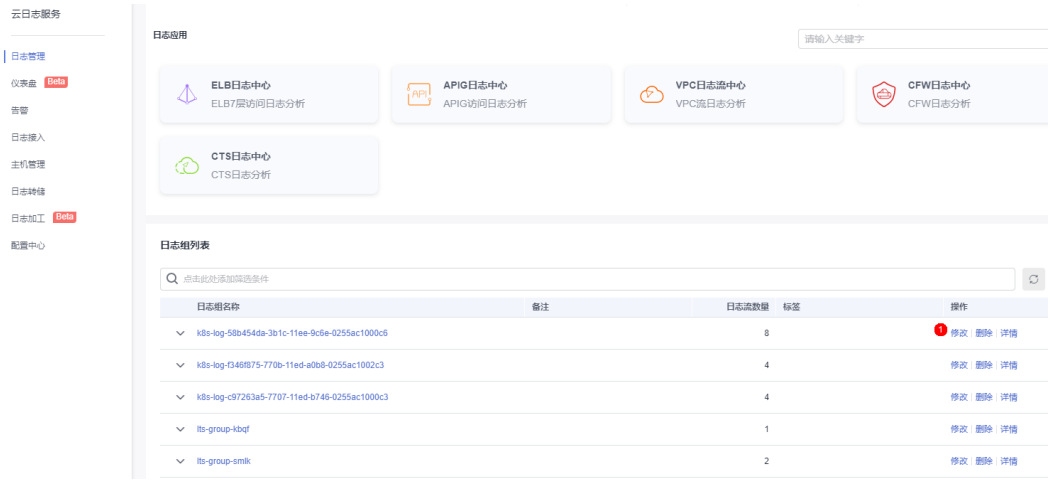


图 9-82 修改日志存储时间

9.7.2 监控中心 FAQ

索引

为什么监控中心没有数据了?

- 可能原因一：云原生监控插件异常

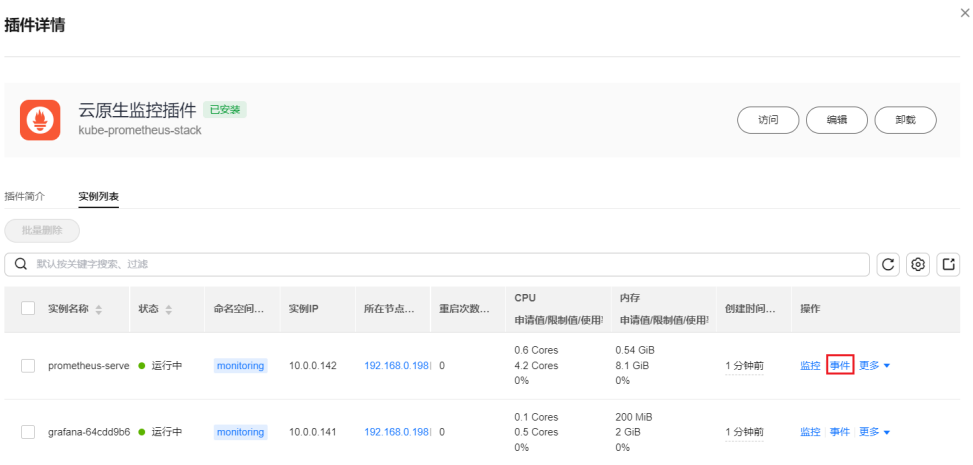
  请前往集群详情的“插件中心”页面，先检查插件云原生监控插件是否为“运行中”。

  图 9-83 检查插件运行状态

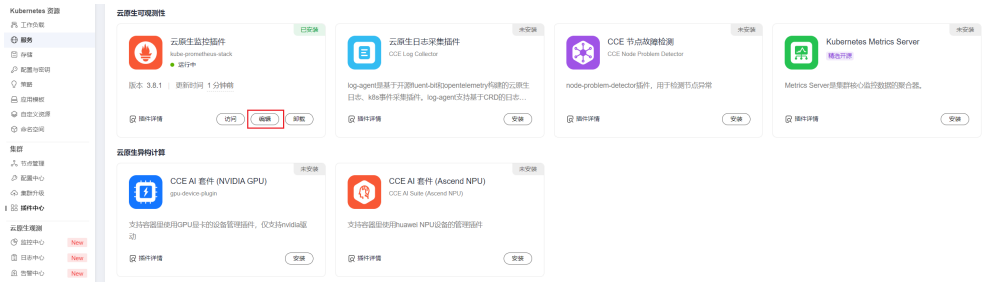
  如果插件运行异常，可以根据云原生监控插件的实例的事件进行排查。

  图 9-84 查看插件事件

- 可能原因二：云原生监控插件对接的AOM实例被删除
  请在集群详情的“插件中心”页面，检查插件云原生监控插件的配置。

  图 9-85 编辑插件配置

  确认AOM实例非空。

  图 9-86 查看 AOM 实例

如何关闭监控中心?

如需关闭监控中心，请前往CCE控制台“插件管理”页面卸载云原生监控插件，或者关闭AOM对接，即可以停止使用该功能。

监控中心为什么没有展示自定义指标?

监控中心暂不支持用户自定义指标的展示，如果需要查看自定义指标，可以到AOM服务监控中心的仪表盘配置自定义指标的仪表盘。

为什么云原生监控插件开启本地数据存储时，重启 prometheus-server 实例可能会导致节点列表的资源信息短时间（1-2 分钟）无法正常显示?

因为当prometheus-server实例重启后，实例指标的uid标签值发生了变化。而由于本地存储了数据的机制，导致prometheus-server实例滚动重启的这段时间里指标重叠，即云原生监控插件上报到AOM的指标同时存在新老prometheus-server实例的指标，因而导致节点列表的资源信息不准确。故在指标重叠的这段时间内，不展示节点列表的资源信息。若无特殊场景，对接AOM推荐使用关闭本地数据存储的云原生监控插件。

为什么云原生监控插件开启本地数据存储时，重启 kube-state-metrics 实例可能会导致页面部分数据翻倍?

当kube-state-metrics实例被调度到一个新的节点上，kube-state-metrics采集的指标中的instance标签值就发生了变化。而由于本地存储了数据的机制，导致kube-state-

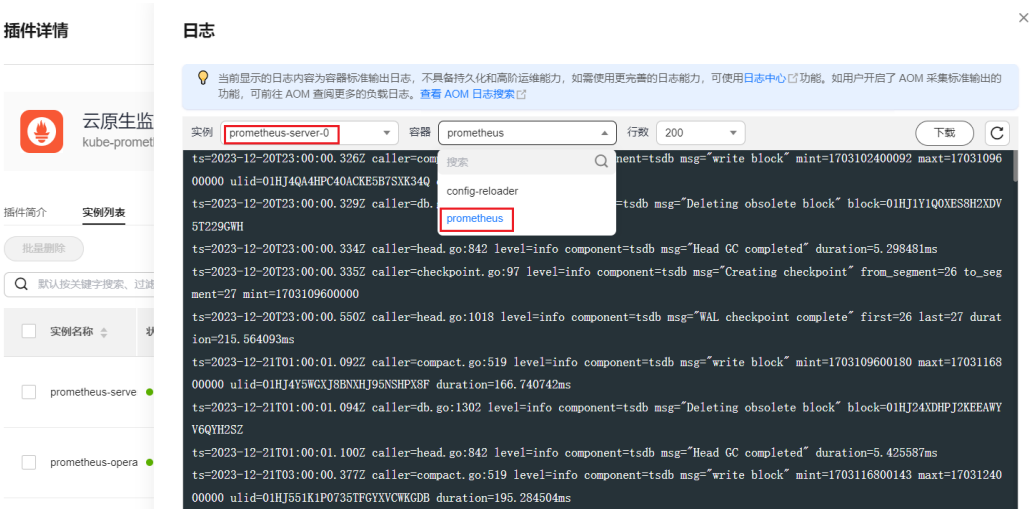
metrics滚动重启的这段时间里指标重叠，即云原生监控插件上报到AOM的指标同时存在新老kube-state-metrics实例的数据。又因为instance标签值不一致，这两次上报的指标都被认为是有效数据。从而导致“监控中心 > 集群”页面在统计的节点、工作负载、Pod、命名空间、控制面组件的数量时翻倍。若无特殊场景，对接AOM推荐使用关闭本地数据存储的云原生监控插件。

云原生监控插件开启本地数据存储时为什么不能正常上报指标?

出现该问题的原因可能为开启本地数据存储时插件实例挂载的PV存储空间已满，导致指标无法写入。

请到插件中心，选中prometheus-server-x实例，查看日志。如果日志中存在：“no space left on device”类似的日志打印，则说明Prometheus挂载的磁盘空间不足。

图 9-87 查看 Prometheus 实例日志



解决方案

- 方案一：推荐使用关闭本地数据存储模式对接AOM实例。使用AOM托管指标数据，无需管理存储。
- 方案二：在左侧导航栏中选择“存储”，并切换至monitoring命名空间，选中pvc-prometheus-server-0的磁盘，扩容对应的存储资源。扩容完成后前往有状态负载页面，将prometheus-server的实例重启。

图 9-88 扩容 PVC

说明

在磁盘空间不足后已无法写入Prometheus指标，将导致数据无法采集，因此扩容完成重启后，该时段的监控数据将会丢失。

为什么监控中心的工作负载/节点 CPU 使用率超过 100%?

工作负载CPU使用率是使用container_cpu_usage_seconds_total计算的，系统会定期更新CPU使用量和更新时间点。Prometheus默认情况下不会使用指标自带的时间点，而是使用采集时间点，此时会导致CPU使用量的时间点不真实，出现较小的时延。

举例如下，假设系统每6s更新一次CPU用量，采集周期为15s，Prometheus第一次采集时间为18:30:14（采集到18:30:10的数据），第二次采集是18:30:29（采集到18:30:28的数据）：

CPU用量	时间点
100000	18:30:10
150000	18:30:16
200000	18:30:22
250000	18:30:28
300000	18:30:34

- 实际每秒CPU使用量：(150000-100000)/(18:30:16-18:30:10)=8333.33
- Prometheus计算的每秒CPU使用量：(250000-100000)/(18:30:29-18:30:14)=10000

说明

以上数据值经过人为放大，仅作示例，实际差异一般很小。

解决方案

您可通过配置honorTimestamps使用指标自带的时间点来规避该问题，您可参考以下优缺点分析决定是否配置。

是否配置honorTimestamps	优点	缺点
不配置honorTimestamps（Prometheus默认行为）	• 指标压缩比更高，降低指标总存储量，查询性能优异。 • 不同指标的间隔点一致，均为普罗采集间隔，一般不会出现断点情况。 • 进行多指标联合PromQL计算，偏差的可能性较低。	可能会出现CPU使用率等指标的微小失真。

是否配置honorTimestamps	优点	缺点
配置honorTimestamps	采集的指标和实际时间点完全吻合，在压测等场景下，计算出来的结果更加真实。	• 增加指标总存储量，降低指标压缩比，查询性能降低。 • 不是所有的指标暴露方都会正确的携带时间戳，可能会导致对多个指标进行PromQL计算时，出现较大偏差。 • 极端情况下指标可能会出现断点。

您可以参考以下方式配置honorTimestamps。

说明

集群中需要安装3.11.0版本及以上的云原生监控插件，且已开启系统预置采集功能。

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“配置与密钥”，并切换至“monitoring”命名空间，找到名为“persistent-user-config”的配置项。

步骤3 单击“编辑YAML”，搜索kubelet-cadvisor，然后新增配置honorTimestamps: true。

```
...
  - customBlacklist: []
    customWhitelist: []
    destNamespace: kube-system
    name: kubelet-cadvisor
    namespace: monitoring
    scrapeAllMetrics: false
    honorTimestamps: true
    scrapeInterval: ""
    status: "on"
    type: ServiceMonitor
...
```

步骤4 单击“确定”保存配置项，等待约1分钟即可生效。

----结束

采集端点访问 403 的原因是什么？该如何处理？

问题根因

您的采集端点对应的采集任务ServiceMonitor/PodMonitor配置了认证，出于安全考虑，页面访问默认不支持访问需认证的端点。

解决方案：您可以通过配置，允许访问带认证的端点。

须知

配置允许访问带认证的端点，会导致您需认证的端点可在集群内通过访问prometheus-lightweight服务的方式直接访问，因此请勿将prometheus-lightweight服务端口暴露至集群外部。

1. 登录CCE控制台，单击集群名称进入集群详情页。
2. 在左侧导航栏中选择“配置与密钥”，并切换至“全部命名空间”，找到名为“persistent-user-config”的配置项。
3. 单击“更新”，对lightweight-user-config.yaml配置数据进行编辑，在operatorConfigOverride字段下增加一条配置。

```
customSettings:
  operatorEnvOverride: []
  operatorConfigOverride:
  - --target-response-auto-auth=true
  promAdapterConfigOverride: []
```

4. 单击“确定”保存配置项，等待约1分钟即可生效。

9.7.3 日志中心 FAQ

索引

如何关闭日志中心?

关闭容器日志、kubernetes事件采集

方法一：进入“日志中心”，单击右上角“日志采集策略”，删除对应的日志策略。其中default-event为默认上报kubernetes事件，default-stdout为默认上报标准输出。

图 9-89 删除日志采集策略

方法二：进入“插件中心”，卸载CCE 云原生日志采集插件，**注意：卸载后插件将不再上报kubernetes事件到AOM。**

关闭控制面组件日志采集

进入“日志中心 > 控制面组件日志”，单击“配置控制面组件日志”，取消勾选不需要采集的组件。

图 9-90 配置控制面组件日志

关闭控制面审计日志采集

进入“日志中心 > 控制面审计日志”，单击“配置控制面审计日志”，取消勾选不需要采集的组件。

图 9-91 配置控制面审计日志

插件中除 log-operator 外组件均未就绪

问题现象：插件中除log-operator外组件均未就绪，且出现异常事件“实例挂卷失败”。

解决方案：请查看log-operator日志，安装插件时，其余组件所需的配置文件需要log-operator生成，log-operator生成配置出错，会导致所有组件无法正常启动。

日志信息如下：

```
MountVolume.SetUp failed for volume "otel-collector-config-vol":configmap "log-agent-otel-collector-config" not found
```

log-operator 标准输出报错

问题现象：

```
2023/05/05 12:17:20.799 [E] call 3 times failed, resion: create group failed, projectID: xxx, groupName: k8s-log-xxx, err: create groups status code: 400, response: {"error_code":"LTS.0104","error_msg":"Failed to create log group, the number of log groups exceeds the quota"}, url: https://lts.cn-north-4.myhuaweicloud.com/v2/xxx/groups, process will retry after 45s
```

解决方案：LTS日志组有配额限制，如果出现该报错，请前往LTS下删除部分无用的日志组。限制详情见：日志组。

节点容器引擎为 docker 时采集不到容器文件日志

问题现象：

配置了容器文件路径采集，采集的目录不是挂载到容器内的，且节点容器引擎为docker，采集不到日志。

解决方案：

请检查工作负载所在节点的容器存储模式是否为Device Mapper，Device Mapper不支持采集容器内日志（创建日志策略时已提示此限制）。检查方法如下：

1. 进入业务工作负载所在节点。
2. 执行**docker info | grep "Storage Driver"**。
3. 若返回的Storage Driver值为Device Mapper，则该日志无法采集。

图 9-92 创建日志策略

日志无法上报，otel 组件标准输出报错：log's quota has full

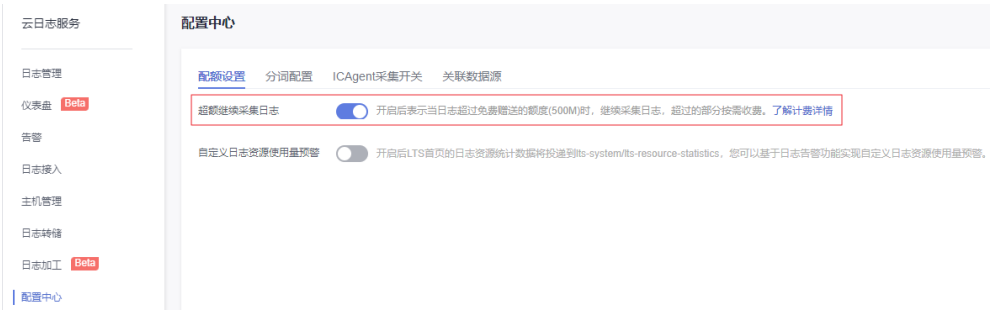
图 9-93 otel 组件报错信息

```
2023-08-16T09:03:20.067+0800    error    exporterhelper/queued_retry.go:361      Exporting failed. Try enabling retry_
on_failure config option to retry on retryable errors    {"kind": "exporter", "data_type": "logs", "name": "lts/defaul
t-event", "error": "fail to push event data via lts exporter: read body {\"errorCode\":\"SVCSTG.ALS.200.210\",\"error
Message\":\"projectid                                    s quota has full!!\",\"result\":null} error"}
go.opentelemetry.io/collector/exporter/exporterhelper.(*retrySender).send
        go.opentelemetry.io/collector@v0.58.0/exporter/exporterhelper/queued_retry.go:361
go.opentelemetry.io/collector/exporter/exporterhelper.(*logsExporterWithObservability).send
        go.opentelemetry.io/collector@v0.58.0/exporter/exporterhelper/logs.go:142
go.opentelemetry.io/collector/exporter/exporterhelper.(*queuedRetrySender).send
        go.opentelemetry.io/collector@v0.58.0/exporter/exporterhelper/queued_retry.go:295
go.opentelemetry.io/collector/exporter/exporterhelper.NewLogsExporterWithContext.func2
        go.opentelemetry.io/collector@v0.58.0/exporter/exporterhelper/logs.go:122
go.opentelemetry.io/collector/consumer.ConsumeLogsFunc.ConsumeLogs
        go.opentelemetry.io/collector@v0.58.0/consumer/logs.go:36
go.opentelemetry.io/collector/service/internal/fanoutconsumer.(*logsConsumer).ConsumeLogs
        go.opentelemetry.io/collector@v0.58.0/service/internal/fanoutconsumer/logs.go:77
cieotelcol/receiver/k8seventsreceiver.(*k8seventsReceiver).handleEvent
        cieotelcol/receiver/k8seventsreceiver/receiver.go:138
cieotelcol/receiver/k8seventsreceiver.(*k8seventsReceiver).startWatch.func1
        cieotelcol/receiver/k8seventsreceiver/receiver.go:116
k8s.io/client-go/tools/cache.ResourceEventHandlerFuncs.OnAdd
        k8s.io/client-go@v0.24.3/tools/cache/controller.go:232
k8s.io/client-go/tools/cache.processDeltas
        k8s.io/client-go@v0.24.3/tools/cache/controller.go:441
k8s.io/client-go/tools/cache.newInformer.func1
```

解决方案：

云日志服务（LTS）有免费赠送的额度，超出后将收费，报错说明免费额度已用完，如果需要继续使用，请前往云日志服务控制台“配置中心”，打开“超额继续采集日志”开关。

图 9-94 配额设置



采集容器内日志，且采集目录配置了通配符，日志无法采集

排查方法：请检查工作负载配置中Volume挂载情况，如果业务容器的数据目录是通过数据卷（Volume）挂载的，插件不支持采集它的父目录，需设置采集目录为完整的数据目录。例如/var/log/service目录是数据卷挂载的路径，则设置采集目录为/var/log或/var/log/*将采集不到该目录下的日志，需设置采集目录为/var/log/service。

解决方案：若日志生成目录为/application/logs/{应用名}/*.log，建议工作负载挂载Volume时，直接挂载/application/logs，日志策略中配置采集路径为/application/logs/*/*.log

fluent-bit 容器组一直重启

排查方法：节点上fluent-bit容器组一直重启，且通过**kubectl describe pod**命令查看Pod重启原因为OOM。查询该fluent-bit所在节点存在大量被驱逐的Pod，资源被占用导致出现OOM。

解决方案：删除节点上被驱逐的Pod。

节点 OS 为 Ubuntu 18.04 时出现日志无法采集

排查方法：重启当前节点的fluent-bit pod，查看日志是否正常采集。如依然无法采集，请确认需要采集的文件是否为打包镜像时已经存在于镜像中的日志文件。对于容器日志采集的场景来说，镜像打包时已存在的文件的日志非运行日志，属于无效日志无法采集。该问题为社区已知问题，详情请参见**开源issue**。

解决方案：若需要采集打包镜像时已经存在于镜像中的日志文件，建议添加在创建工作负载时，设置“生命周期>启动后命令”，在工作负载Pod启动前，先删除原来日志文件，使日志文件重新生成。

采集 Job 日志时出现日志无法采集

排查方法：确认Job的存活时间。若Job存活时间低于1分钟，日志还未被采集，Pod就已经被销毁，可能存在日志采集不到的情况。

解决方案：延长Job的存活时间。

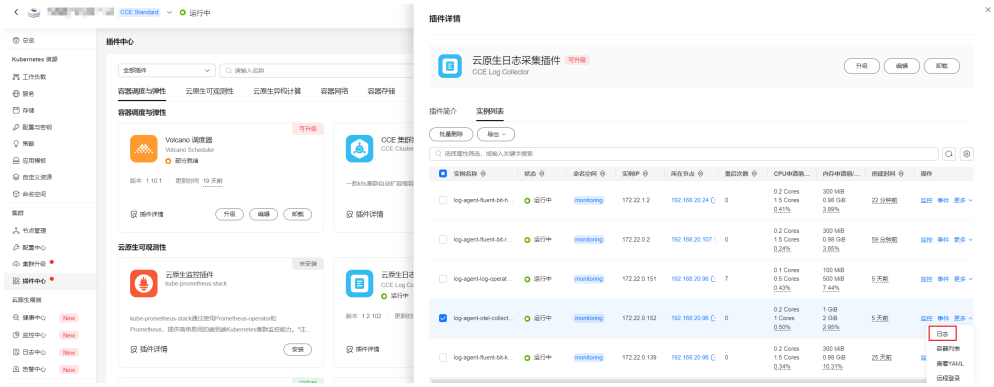
云原生日志采集插件运行正常，部分日志策略未生效

解决方案：

- 若未生效的日志策略采集类型为事件类型或插件版本低于1.5.0，则检查log-agent-otel-collector工作负载的标准输出。

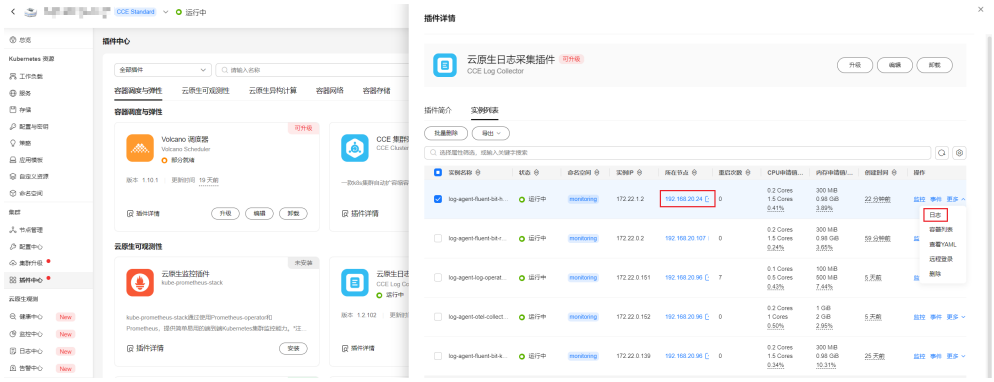
可在插件中心单击 “云原生日志采集插件”名称，在“实例列表”中选择 log-agent-otel-collector 最右侧的日志查看。

图 9-95 查看 log-agent-otel-collector 实例日志



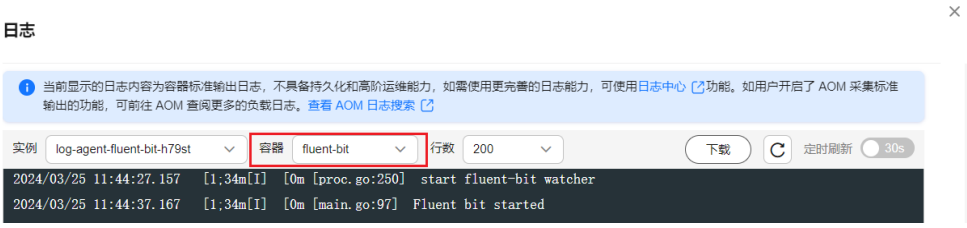
- 若未生效的日志策略类型不为事件类型，且插件版本高于1.5.0，则检查需要采集的容器所在节点的log-agent-fluent-bit实例日志。

图 9-96 查看 log-agent-fluent-bit 实例日志



容器选择fluent-bit，并在日志中查看关键字“fail to push {event/log} data via lts exporter”，查看后面的errorMessage。

图 9-97 查看 fluent-bit 容器日志



a. 若报错为“The log streamId does not exist.”，则日志组或日志流不存在，可前往“日志中心>日志采集策略”中，通过“编辑”或“删除”重建日志策略，更新策略中的日志组日志流。

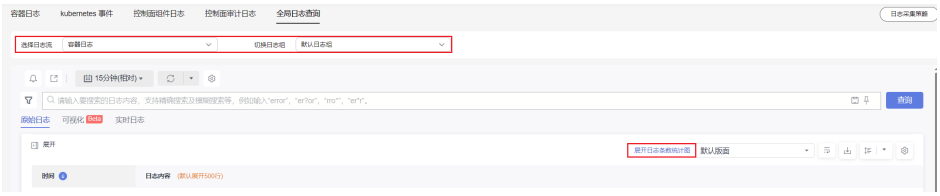
b. 其余报错可前往LTS搜索错误码，查看报错原因。

log-agent-otel-collector 组件出现 OOM

排查方法：

1. 查看log-agent-otel-collector组件标准输出，查看近期是否有错误日志。
   ```
   kubectl logs -n monitoring log-agent-otel-collector-xxx
   ```
   若存在报错请优先处理报错，确认日志恢复正常采集。
2. 若日志近期没有报错，且仍然出现OOM，则参考以下步骤进行处理：
   a. 进入“日志中心”，单击“展开日志条数统计图”查看日志统计图。若上报的日志组日志流不是默认日志组日志流，则单击“全局日志查询”页签，选择上报的日志组和日志流后进行查看。

   图 9-98 查看日志统计

   b. 根据统计图中的柱状图，计算每秒上报的日志量，检查是否超过当前规格的日志采集性能。

      若超过当前规格的日志采集性能，可尝试增加log-agent-otel-collector副本数或提高log-agent-otel-collector的内存上限。

   c. 若CPU使用率超过90%，则需要提高log-agent-otel-collector的CPU上限。

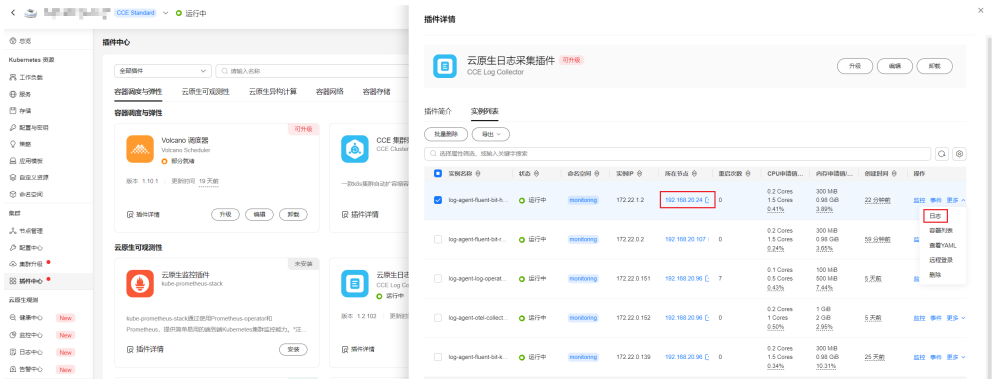
节点负载过多，采集日志时缺少部分 Pod 信息

问题：插件版本在1.5.0以上，采集容器内日志或容器标准输出过程中出现日志缺少部分Pod信息，例如podID、podName等。

排查方法：

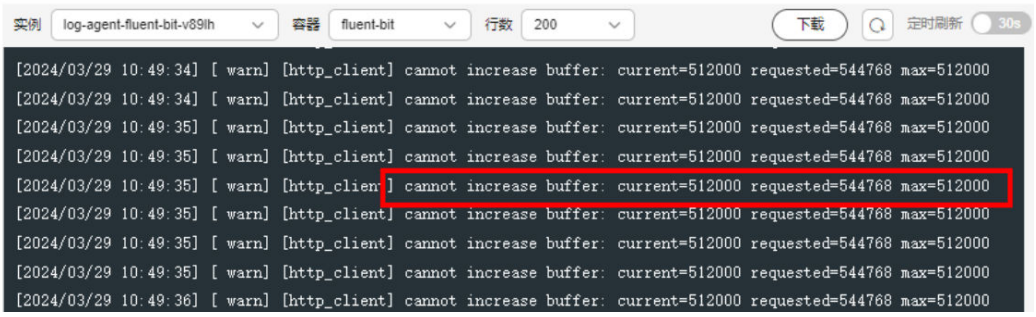
进入“插件中心”，单击“云原生日志采集插件”，选择实例列表，找到对应节点的log-agent-fluent-bit，单击“更多>日志”。

图 9-99 查看 log-agent-fluent-bit 实例日志



容器选择fluent-bit，并在日志中查看关键字“cannot increase buffer: current=512000 requested=*** max=512000”。

图 9-100 查看 fluent-bit 容器日志



解决方案:

前往节点执行**kubectl edit deploy -n monitoring log-agent-log-operator**， 编辑log-operator容器的命令行参数，添加命令行--kubernetes-buffer-size=20MB，当前默认值为16MB，请根据节点pod信息总大小估算该值大小。0为无限制。

注意

若升级插件，则需要重新配置该参数。

图 9-101 修改 log-operator 容器命令行参数

```
containers:
- command:
  - /var/paas/log-operator/log-operator
  - --enable-leader-election
  - --log-print-type=console
  - --enable-fluent-bit-use-kubelet=true
  - --enable-webhook=true
  - --buffer-chunk-size=128k
  - --buffer-max-size=512k
  - --mem-buf-limit=40MB
  - --aom-endpoint=https://aomperform.[redacted].com
  - --lts-endpoint=https://lts.[redacted].com
  - --lts-access-endpoint=https://lts-access.[redacted].com:8102
  - --enable-ssl=true
  - --kubernetes-buffer-size=20MB
  env:
```

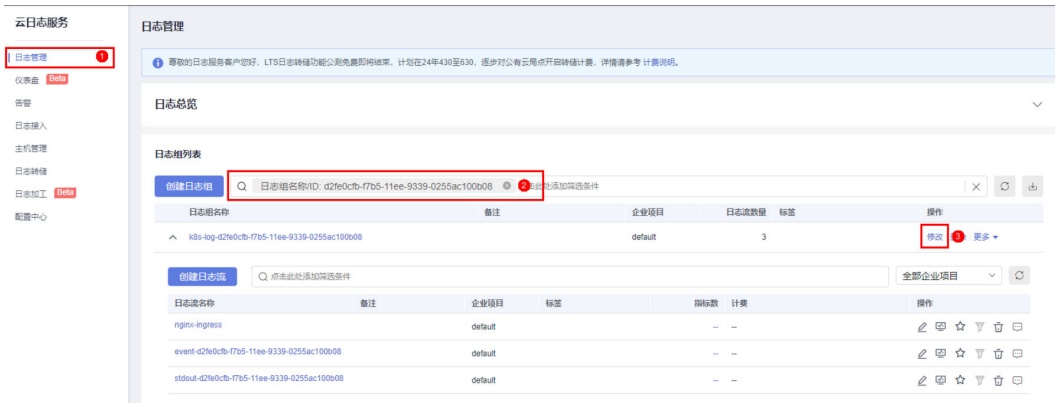
如何修改集群日志中心的日志存储时间?

步骤1 获取当前集群ID。

图 9-102 查看集群 ID

步骤2 进入云日志服务，根据集群ID查询对应的日志组和日志流。

图 9-103 查询日志组



步骤3 找到对应的日志组，单击“修改”，设置日志存储时间。

说明

日志存储时间影响将日志存储费用。

图 9-104 修改日志存储时间

----结束

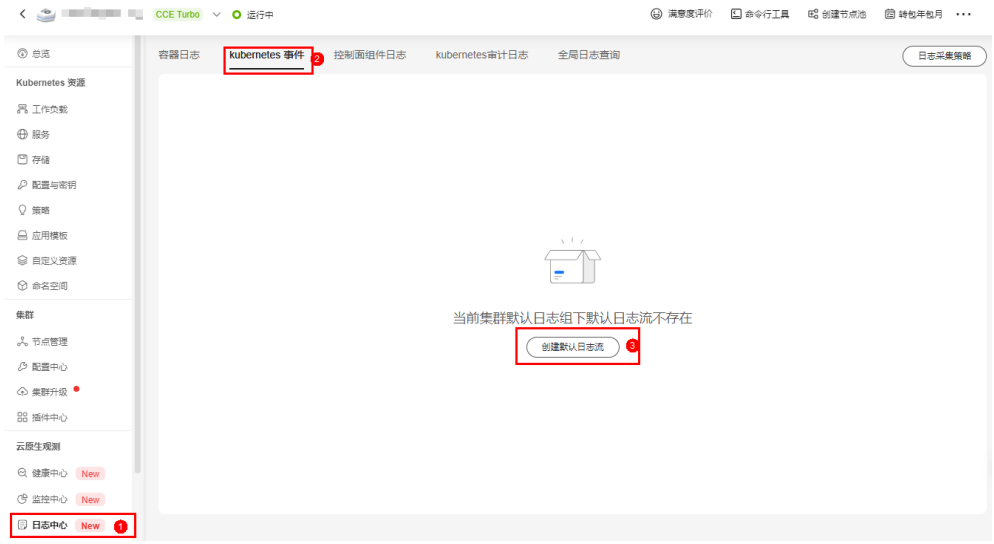
如何修复日志采集策略中日志组（流）不存在的问题?

- **场景一：默认日志组（流）不存在**

  以Kubernetes事件为例：当默认日志组（流）不存在时，控制台中的“Kubernetes事件”页面会提示当前日志组（流）不存在，您可以单击“创建默认日志组（流）”进行重新创建。

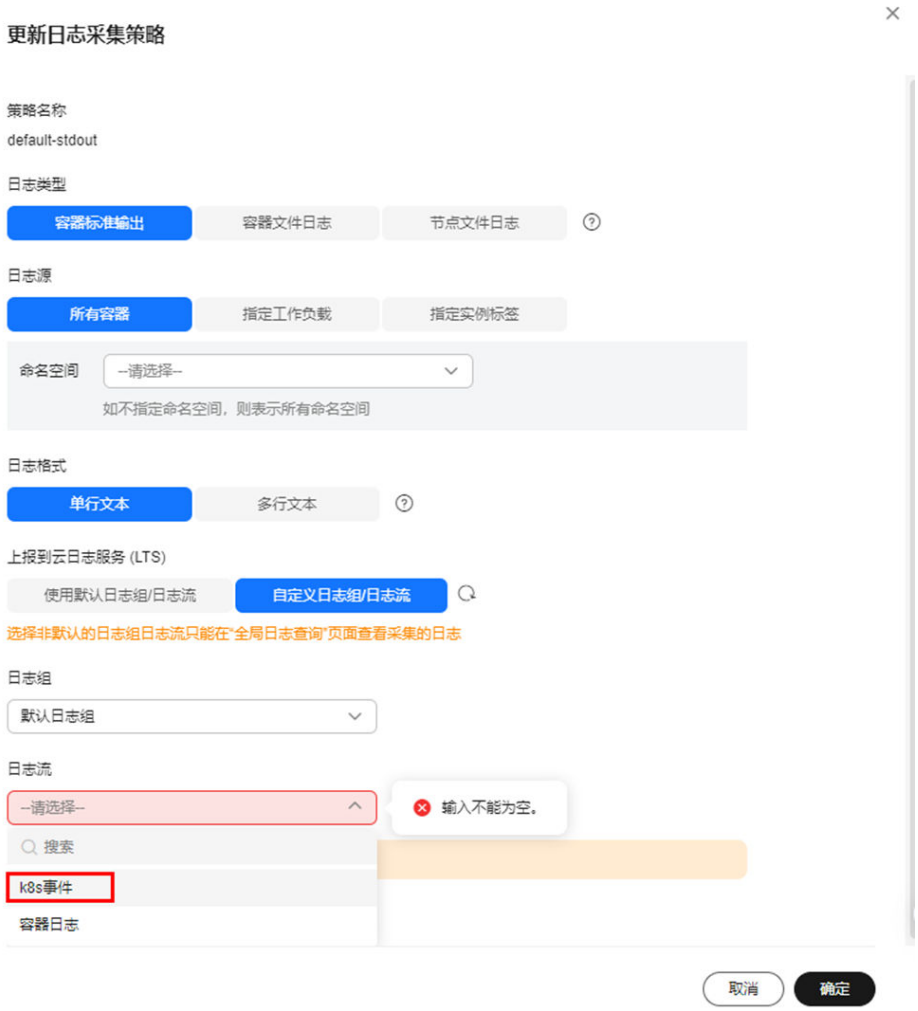
  重建后的默认日志组（流）的ID会发生变化，对接默认日志组（流）的已有采集策略无法生效，请参见**场景二**进行修复。

图 9-105 创建默认日志组（流）



- **场景二：默认日志组（流）存在但与日志采集策略不一致**
  - 支持修改的日志采集策略，例如default-stdout，修复方案如下：
    i. 登录CCE集群控制台，前往“日志中心”。
    ii. 单击右上角“日志采集策略”，在对应的日志采集策略的操作栏中单击“编辑”。
    iii. 选择“自定义日志组/日志流”，然后将其设置为集群的默认日志组（流）。

图 9-106 设置默认日志组（流）



- 不支持修改的日志采集策略，例如default-event，则需要重建对应采集策略，修复方案如下：
  i. 登录CCE集群控制台，前往“日志中心”。
  ii. 单击右上角“日志采集策略”，在对应的日志采集策略的操作栏中单击“删除”。
  iii. 然后单击“创建日志采集策略”，选择策略模板中的“采集kubernetes事件”，单击“确定”进行创建。

- **场景三：自定义日志组（流）不存在**

  CCE界面暂不支持非默认日志组（流）的创建，请到云日志服务（LTS）进行重新创建。

  创建完毕后，参考以下步骤进行修复：

  a. 登录CCE集群控制台，前往“日志中心”。
  b. 单击右上角“日志采集策略”，在对应的日志采集策略的操作栏中单击“编辑”。
  c. 选择“自定义日志组/日志流”，然后将其设置为新建的日志组（流）。

图 9-107 设置自定义日志组（流）

Pod 调度到 CCI 后，采集不到日志

若使用**profile控制Pod调度到CCI**后，出现调度到CCI的Pod采集不到日志的情况，且确认采集策略在CCE侧功能正常。

请检查CCE突发弹性引擎（对接CCI）插件版本是否低于1.3.54，若低于该版本，请升级插件。

9.7.4 告警中心 FAQ

如何停止接收告警?

在“告警中心 > 默认联系组”页面对确认订阅的终端，执行删除即可。

图 9-108 删除联系组

为什么告警清除之后还会继续发送告警?

告警清除仅清除告警规则页面的统计，如该告警持续达到阈值或者异常事件持续发生，仍会产生告警。

告警中心的联系组支持钉钉、飞书等么?

在告警中心的默认联系组页面无法创建钉钉、飞书等通知方式，需要在SMN消息通知服务进行开通，请参考**SMN文档**。

9.8 可观测性最佳实践

9.8.1 云原生监控插件兼容自建 Prometheus

云原生监控插件兼容模式

- 若您已自建Prometheus，且您的Prometheus基于开源，未做深度定制、未与您的监控系统深度整合，建议您卸载自建Prometheus并直接使用云原生监控插件对您的集群进行监控，无需开启“兼容模式”。

  说明

  卸载您自建的Prometheus，需要确保删除您自建Prometheus所有的工作负载以及以monitoring.coreos.com结尾的所有CRDs。详情请参见**如何移除自建Prometheus?** 。

- 若您自建的Prometheus无法卸载，且需要使用成本洞察、监控中心等功能，当您的自建Prometheus满足**兼容性要求**时，您可以选择开启“兼容模式”，详情请参见**开通监控中心**。

  说明

  兼容模式下无法得到完整的云原生监控插件体验，例如，兼容模式不支持成本优化、无法在AOM页面进行指标废弃等，详情请参见**兼容模式约束**。

兼容性要求

请确保您自建的Prometheus满足以下条件，否则无法正常共存运行：

- 若您的Prometheus不是基于Operator社区的KubePrometheus构建的，则满足兼容性要求。
- 若您的Prometheus是基于Operator社区的KubePrometheus构建的，则CRD prometheuses.monitoring.coreos.com的版本至少应该为0.8.0。

  您可以在集群内执行如下命令，快速进行判断：


```
kubectl get crd prometheuses.monitoring.coreos.com -oyaml | grep controller-gen.kubebuilder.io/version
```

若回显如下，则满足兼容性要求。

```
Error from server (NotFound): customresourcedefinitions.apiextensions.k8s.io "prometheuses.monitoring.coreos.com" not found
```

若回显如下，且最后的版本≥v0.8.0，则满足兼容性要求。

```
controller-gen.kubebuilder.io/version: v0.9.2
```

兼容模式约束

- 兼容模式会将云原生监控插件安装在cce-monitoring名空间下，默认不识别任何其他命名空间的ServiceMonitor和PodMonitor。
- 只支持无本地存储的模式。
- 开启后不支持关闭，可通过插件卸载安装的方式切换为正常模式。
- 暂不支持成本优化。
- 暂不支持在AOM页面进行指标废弃。
- 暂不支持在AOM页面进行ServiceMonitor和PodMonitor启停。

如何移除自建 Prometheus?

步骤1 首先，您需要删除所有自建Prometheus相关的工作负载、服务以及其他资源，包含DaemonSet、Deployment、Statefulset、Service等。一般在monitoring命名空间下，请根据您实际安装的命名空间进行调整。

步骤2 查询所有monitoring.coreos.com CRDs。

```
kubectl get crd | grep monitoring.coreos.com | awk '{print $1}' | xargs
```

若回显为空，则无需下一步操作。

步骤3 删除所有monitoring.coreos.com CRDs。

```
kubectl delete crd $(kubectl get crd | grep monitoring.coreos.com | awk '{print $1}' | xargs)
```

----结束

9.8.2 使用云原生监控插件监控自定义指标

CCE提供了云原生监控插件，支持使用Prometheus监控自定义指标。

本文将通过一个Nginx应用的示例演示如何使用Prometheus监控自定义指标，步骤如下：

1. **安装并访问云原生监控插件**

   CCE提供了集成Prometheus功能的插件，支持一键安装。

2. **准备应用**

   您需要准备一个应用镜像，该应用需要提供监控指标接口供Prometheus采集，且监控数据需要**满足Prometheus的规范**。

3. 监控自定义指标

   在集群中使用该应用镜像部署工作负载，将自动上报自定义监控指标至Prometheus。

   自定义指标监控支持四种配置方式。

   - **方法一：配置Pod Annotations监控自定义指标**

- 方法二：配置Service Annotations监控自定义指标
- 方法三：配置Pod Monitor监控自定义指标
- 方法四：配置Service Monitor监控自定义指标
- 方法五：使用AdditionalScrapeConfigs监控自定义指标

自定义指标计费说明

当云原生监控插件对接AOM后，指标会上报到您选择的AOM实例，其中容器**基础指标**免费，自定义指标将由AOM服务进行收费，详情请参见**价格详情**。

Prometheus 监控数据采集说明

Prometheus通过周期性的调用应用程序的监控指标接口（默认为“/metrics”）获取监控数据，应用程序需要提供监控指标接口供Prometheus调用，且监控数据需要满足Prometheus的规范，如下所示。

```
# TYPE nginx_connections_active gauge
nginx_connections_active 2
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
```

Prometheus提供了各种语言的客户端，客户端具体请参见**Prometheus CLIENT LIBRARIES**，开发Exporter具体方法请参见**WRITING EXPORTERS**。Prometheus社区提供丰富的第三方exporter可以直接使用，具体请参见**EXPORTERS AND INTEGRATIONS**。

约束与限制

- 使用Prometheus监控自定义指标时，应用程序需要提供监控指标接口，详情请参见**Prometheus监控数据采集说明**。
- 使用Pod/Service Annotations的方式暂不支持采集kube-system与monitoring命名空间下的指标，如需采集这两个命名空间下的指标，请通过Pod Monitor与Service Monitor的方式配置。
- 本文使用Nginx应用示例会拉取nginx/nginx-prometheus-exporter:0.9.0镜像，需要为应用部署的节点添加EIP或先将此镜像上传到SWR，以免部署应用失败。

安装并访问云原生监控插件

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“插件中心”，在右侧找到**云原生监控插件**，单击“安装”。该插件除提供Prometheus监控能力外，还支持将监控数据与监控中心对接。

安装插件时请关注以下配置，其余配置可根据需求填写，详情请参见**云原生监控插件**。

- 3.8.0及以上版本，需要确认插件配置中开启自定义指标采集。

- 3.8.0以下版本，无需配置自定义指标采集开关。

步骤3 插件安装完成后会在集群中部署一系列工作负载和Service。其中Prometheus的Server端会在monitoring命名空间下以有状态工作负载进行部署。

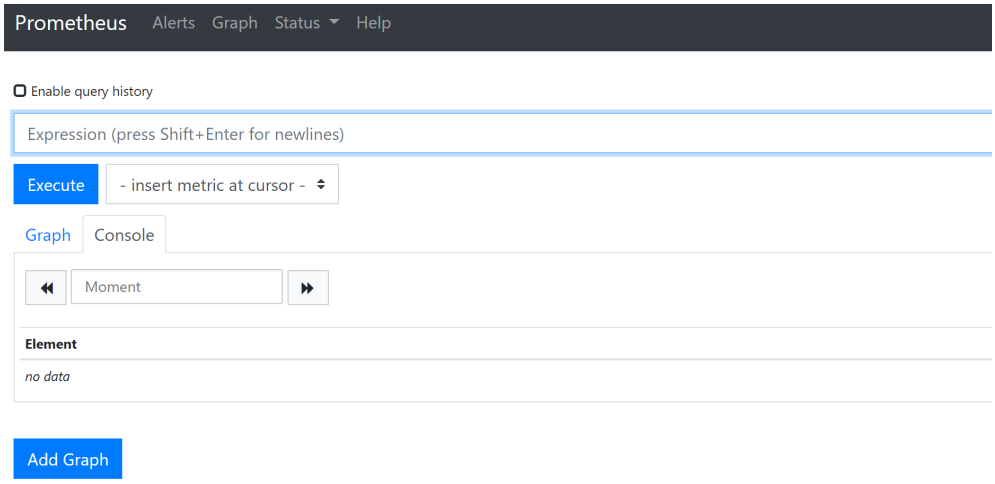
您可以创建一个公网**LoadBalancer类型Service**，这样就可以从外部访问Prometheus。

1. 登录CCE控制台，选择一个已安装Prometheus的集群，在左侧导航栏中选择“服务”。
2. 单击右上角“YAML创建”，创建一个公网LoadBalancer类型的Service。

```
apiVersion: v1
kind: Service
metadata:
  name: prom-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: prometheus
    component: server
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 88            #服务端口号，可自定义
      targetPort: 9090     #Prometheus的默认端口号，无需更改
  selector:               #标签选择器可根据Prometheus Server实例的标签进行调整
    app.kubernetes.io/name: prometheus
    prometheus: server
  type: LoadBalancer
```

3. 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Prometheus。

图 9-109 访问 Prometheus



----结束

准备应用

自行开发的应用程序需要提供监控指标接口供采集，且监控数据需要满足Prometheus的规范，详情请参见**Prometheus监控数据采集说明**。

本文以Nginx为例采集监控数据，Nginx本身有个名叫ngx_http_stub_status_module的模块，这个模块提供了基本的监控功能，通过在nginx.conf的配置可以提供一个对外访问Nginx监控数据的接口。

步骤1 登录一台可连接公网的Linux虚拟机，且要求可执行Docker命令。

步骤2 创建一个nginx.conf文件，如下所示，在http下添加server配置即可让nginx提供对外访问的监控数据的接口。

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;

    server {
      listen 8080;
      server_name  localhost;
      location /stub_status {
         stub_status on;
         access_log off;
      }
    }
}
```

步骤3 使用该配置制作一个镜像，创建Dockerfile文件。

```
vi Dockerfile
```

Dockerfile文件内容如下所示：

```
FROM nginx:1.21.5-alpine
ADD nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

步骤4 使用上面Dockerfile构建镜像并上传到SWR镜像仓库，镜像名称为nginx:exporter。上传镜像的具体方法请参见**客户端上传镜像**。

1. 在左侧导航栏选择“我的镜像”，单击右侧“客户端上传”，在弹出的页面中单击“生成临时登录指令”，单击 复制登录指令。
2. 在集群节点上执行上一步复制的登录指令，登录成功会显示“Login Succeeded”。
3. 执行如下命令构建镜像，镜像名称为nginx，版本为exporter。
   ```
   docker build -t nginx:exporter .
   ```
4. 为镜像打标签并上传至镜像仓库，其中镜像仓库地址和组织名称请根据实际情况修改。
   ```
   docker tag nginx:exporter swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:exporter
   docker push swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:exporter
   ```

步骤5 查看应用指标。

1. 使用nginx:exporter创建工作负载。
2. **登录到容器中**，并通过http://<ip_address>:8080/stub_status获取到nginx的监控数据，其中<ip_address>为容器的IP地址，监控数据如下所示。

```
# curl http://127.0.0.1:8080/stub_status
Active connections: 3
server accepts handled requests
 146269 146269 212
Reading: 0 Writing: 1 Waiting: 2
```

----结束

方法一：配置 Pod Annotations 监控自定义指标

当Pod的Annotations配置符合Prometheus采集规范的规则后，Prometheus会自动采集这些Pod暴露的指标。

如上所述的nginx:exporter提供的监控数据，其数据格式并不满足Prometheus的要求，需要将其转换成Prometheus需要的格式，可以使用**nginx-prometheus-exporter**来转换Nginx的指标，将nginx:exporter和nginx-prometheus-exporter部署到同一个Pod，并在部署时添加如下Annotations就可以自动被Prometheus采集监控指标。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-exporter
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-exporter
  template:
    metadata:
      labels:
        app: nginx-exporter
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9113"
        prometheus.io/path: "/metrics"
        prometheus.io/scheme: "http"
    spec:
      containers:
        - name: container-0
          image: 'nginx:exporter'       # 替换为您上传到SWR的镜像地址
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
        - name: container-1
          image: 'nginx/nginx-prometheus-exporter:0.9.0'
          command:
            - nginx-prometheus-exporter
          args:
            - '-nginx.scrape-uri=http://127.0.0.1:8080/stub_status'
      imagePullSecrets:
        - name: default-secret
```

其中

- prometheus.io/scrape：表示是否需要prometheus采集Pod的监控数据，取值为true。

- prometheus.io/port：表示采集监控数据接口的端口，由需要采集的应用决定。本示例中采集端口为9113。
- prometheus.io/path：表示采集监控数据接口的URL，如不配置则默认为“/metrics”。
- prometheus.io/scheme：表示采集的协议，值可以填写http或https。

应用部署成功后，**访问Prometheus**，根据job名称查询自定义监控指标。

可以查询到nginx相关的自定义监控指标，通过job名称可以判断出是根据Pod配置上报的。

```
nginx_connections_accepted{cluster="2048c170-8359-11ee-9527-0255ac1000cf", cluster_category="CCE", cluster_name="cce-test", container="container-0", instance="10.0.0.46:9113", job="monitoring/kubernetes-pods", kubernetes_namespace="default", kubernetes_pod="nginx-exporter-77bf4d4948-zsb59", namespace="default", pod="nginx-exporter-77bf4d4948-zsb59", prometheus="monitoring/server"}
```

方法二：配置 Service Annotations 监控自定义指标

当Service的Annotations配置符合Prometheus采集规范的规则后，Prometheus会自动采集这些Service暴露的指标。

Service Annotations使用方法和Pod Annotations基本相同，主要是采集的指标的适用场景不同，Pod Annotations更关注Pod的资源使用情况，Service Annotations侧重于对该业务的请求等指标。

部署示例应用如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-test
  template:
    metadata:
      labels:
        app: nginx-test
    spec:
      containers:
        - name: container-0
          image: 'nginx:exporter'      # 替换为您上传到SWR的镜像地址
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
        - name: container-1
          image: 'nginx/nginx-prometheus-exporter:0.9.0'
          command:
            - nginx-prometheus-exporter
          args:
            - '-nginx.scrape-uri=http://127.0.0.1:8080/stub_status'
      imagePullSecrets:
        - name: default-secret
```

部署示例Service如下：

```
apiVersion: v1
kind: Service
```



```
metadata:
  name: nginx-test
  labels:
    app: nginx-test
  namespace: default
  annotations:
    prometheus.io/scrape: "true"  # 配置为 true 表示开启服务发现
    prometheus.io/port: "9113"  # 配置为采集指标暴露的端口号
    prometheus.io/path: "/metrics" # 填写指标暴露的 URI 路径，一般是 /metrics
spec:
  selector:
    app: nginx-test
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 8080
      protocol: TCP
    - name: cce-service-1
      protocol: TCP
      port: 9113
      targetPort: 9113
  type: NodePort
```

应用部署成功后，**访问Prometheus**，查询自定义监控指标。通过Service名称可以判断出该指标是根据Service配置上报的。

```
nginx_connections_accepted{app="nginx-test", cluster="2048c170-8359-11ee-9527-0255ac1000cf", cluster_category="CCE", cluster_name="cce-test", instance="10.0.0.38:9113", job="nginx-test", kubernetes_namespace="default", kubernetes_service="nginx-test", namespace="default", pod="nginx-test-78cfb65889-gtv7z", prometheus="monitoring/server", service="nginx-test"}
```

方法三：配置 Pod Monitor 监控自定义指标

云原生监控插件提供了基于PodMonitor与ServiceMonitor配置指标采集任务的能力。Prometheus Operator将watch的PodMonitor的变化，通过Prometheus的reload机制，将Prometheus的采集任务热更新至Prometheus的实例中。

Prometheus Operator定义的CRD资源github地址：**https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/charts/crds/crds**。

部署示例应用如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test2
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-test2
  template:
    metadata:
      labels:
        app: nginx-test2
    spec:
      containers:
      - image: nginx:exporter     # 替换为您上传到SWR的镜像地址
        name: container-0
        ports:
        - containerPort: 9113     # 指标暴露的端口号
          name: nginx-test2       # 该名称是后续配置PodMonitor时相匹配的名称
          protocol: TCP
```



```
      resources:
        limits:
          cpu: 250m
          memory: 300Mi
        requests:
          cpu: 100m
          memory: 100Mi
    - name: container-1
      image: 'nginx/nginx-prometheus-exporter:0.9.0'
      command:
        - nginx-prometheus-exporter
      args:
        - '-nginx.scrape-uri=http://127.0.0.1:8080/stub_status'
    imagePullSecrets:
      - name: default-secret
```

配置Pod Monitor示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: podmonitor-nginx   # PodMonitor的名称
  namespace: monitoring    # 所属命名空间，建议使用monitoring
spec:
  namespaceSelector:       # 匹配工作负载所在的命名空间
    matchNames:
    - default              # 工作负载所属的命名空间
  jobLabel: podmonitor-nginx
  podMetricsEndpoints:
  - interval: 15s
    path: /metrics           # 工作负载暴露指标的路径
    port: nginx-test2        # 工作负载暴露指标的port名称
    tlsConfig:
      insecureSkipVerify: true
  selector:
    matchLabels:
      app: nginx-test2   # Pod携带的标签，能被选择器选中
```

应用部署成功后，**访问Prometheus**，查询自定义监控指标。通过job名称可以判断出该指标是根据PodMonitor配置上报的。

```
nginx_connections_accepted{cluster="2048c170-8359-11ee-9527-0255ac1000cf", cluster_category="CCE",
cluster_name="cce-test", container="container-0", endpoint="nginx-test2", instance="10.0.0.44:9113",
job="monitoring/podmonitor-nginx", namespace="default", pod="nginx-test2-746b7f8fdd-krzfp",
prometheus="monitoring/server"}
```

方法四：配置 Service Monitor 监控自定义指标

云原生监控插件提供了基于PodMonitor与ServiceMonitor配置指标采集任务的能力。Prometheus Operator将watch的ServiceMonitor的变化，通过Prometheus的reload机制，将Prometheus的采集任务热更新至Prometheus的实例中。

Prometheus Operator定义的CRD资源github地址：**https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/charts/crds/crds**。

部署示例应用如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test3
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
```



```
      app: nginx-test3
  template:
    metadata:
      labels:
        app: nginx-test3
    spec:
      containers:
      - image: nginx:exporter        # 替换为您上传到SWR的镜像地址
        name: container-0
        resources:
          limits:
            cpu: 250m
            memory: 300Mi
          requests:
            cpu: 100m
            memory: 100Mi
      - name: container-1
        image: 'nginx/nginx-prometheus-exporter:0.9.0'
        command:
          - nginx-prometheus-exporter
        args:
          - '-nginx.scrape-uri=http://127.0.0.1:8080/stub_status'
      imagePullSecrets:
        - name: default-secret
```

部署示例Service如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-test3
  labels:
    app: nginx-test3
  namespace: default
spec:
  selector:
    app: nginx-test3
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80
      nodePort: 0
      port: 8080
      protocol: TCP
    - name: servicemonitor-ports
      protocol: TCP
      port: 9113
      targetPort: 9113
  type: NodePort
```

配置Service Monitor示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: servicemonitor-nginx
  namespace: monitoring
spec:
  # 配置service中的暴露指标的port的名称
  endpoints:
  - path: /metrics
    port: servicemonitor-ports
  jobLabel: servicemonitor-nginx
  # 采集任务的作用范围，如果不配置，默认为default
  namespaceSelector:
    matchNames:
    - default
  selector:
    matchLabels:
      app: nginx-test3
```


应用部署成功后，**访问Prometheus**，查询自定义监控指标。通过endpoint名称可以判断出该指标是根据ServiceMonitor配置上报的。

```
nginx_connections_accepted{cluster="2048c170-8359-11ee-9527-0255ac1000cf", cluster_category="CCE", cluster_name="cce-test", endpoint="servicemonitor-ports", instance="10.0.0.47:9113", job="nginx-test3", namespace="default", pod="nginx-test3-6f8bccd9-f27hv", prometheus="monitoring/server", service="nginx-test3"}
```

方法五：使用 AdditionalScrapeConfigs 监控自定义指标

须知

集群中需要已安装3.10.1及以上版本的云原生监控插件。

AdditionalScrapeConfigs允许您指定一个Secret的key，将您额外的Prometheus抓取配置附加至云原生监控插件。

由于使用该机制会绕过常规的抓取配置生成逻辑，而是直接将您指定的配置内容传递给Prometheus，因此需要您保证配置的正确性。建议您参考**scrape_config官方文档**进行配置。

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 使用YAML创建如下Secret。

```
kind: Secret
apiVersion: v1
type: Opaque
metadata:
  name: additional-scrape-configs
  namespace: monitoring  # 命名空间仅为示例，命名空间需要与云原生监控插件相同
stringData:
  # 以下为云原生监控插件未开启本地存储时的采集配置示例，您需要替换为您需要的采集配置
  prometheus-additional.yaml: |-
    - job_name: custom-job-test
      metrics_path: /metrics
      relabel_configs:
      - action: keep
        source_labels:
        - __meta_kubernetes_pod_label_app
        - __meta_kubernetes_pod_labelpresent_app
        regex: (prometheus-lightweight);true
      - action: keep
        source_labels:
        - __meta_kubernetes_pod_container_port_name
        regex: web
      kubernetes_sd_configs:
      - role: pod
        namespaces:
          names:
          - monitoring
        namespaces:
          names:
          - monitoring
```

步骤3 编辑persistent-user-config配置项，开启AdditionalScrapeConfigs能力。

```
kubectl edit configmap persistent-user-config -n monitoring
```

在其中operatorConfigOverride字段下新增一行*--common.prom.default-additional-scrape-configs-key=prometheus-additional.yaml*，开启AdditionalScrapeConfigs能力，示例如下：

```
...
data:
```



```
lightweight-user-config.yaml: |
  customSettings:
    additionalScrapeConfigs: []
    agentExtraArgs: []
    metricsDeprecated:
      globalDeprecateMetrics: []
    nodeExporterConfigOverride: []
    operatorConfigOverride:
    - --common.prom.default-additional-scrape-configs-key=prometheus-additional.yaml
...
```

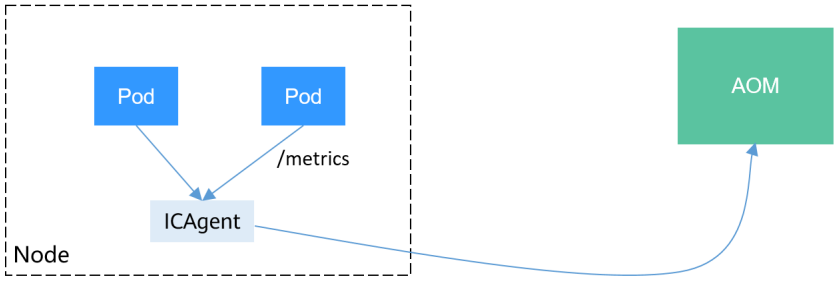
步骤4 前往Grafana或AOM页面处查看您的自定义采集指标是否采集成功。

----结束

9.8.3 使用 AOM 监控自定义指标

CCE支持上传自定义指标到AOM，节点上的ICAgent会定期调用负载中配置的监控指标接口读取监控数据，然后上传到AOM上。

图 9-110 ICAgent 采集监控指标



负载的自定义指标接口可以在创建时配置。本文将通过一个Nginx应用的示例演示如何上报自定义监控指标到AOM，步骤如下：

1. **准备应用**

   您需要准备一个应用镜像，该应用需要提供监控指标接口供ICAgent采集，且监控数据需要**满足Prometheus的规范**。

2. **部署应用并转换指标**

   在集群中使用该应用镜像部署工作负载，将自动上报自定义监控指标。

3. **配置验证**

   前往AOM查看自定义指标是否采集成功。

约束与限制

- ICAgent兼容**Prometheus**的监控数据规范，Pod提供的自定义指标必须满足Prometheus的监控数据规范才能够被ICAgent采集，参见**Prometheus监控数据采集说明**。
- ICAgent仅支持上报**Gauge指标类型**的指标。
- ICAgent调用自定义指标的接口周期为1分钟，不支持修改。

Prometheus 监控数据采集说明

Prometheus通过周期性的调用应用程序的监控指标接口（默认为“/metrics”）获取监控数据，应用程序需要提供监控指标接口供Prometheus调用，且监控数据需要满足Prometheus的规范，如下所示。

```
# TYPE nginx_connections_active gauge
nginx_connections_active 2
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
```

Prometheus提供了各种语言的客户端，客户端具体请参见**Prometheus CLIENT LIBRARIES**，开发Exporter具体方法请参见**WRITING EXPORTERS**。Prometheus社区提供丰富的第三方exporter可以直接使用，具体请参见**EXPORTERS AND INTEGRATIONS**。

准备应用

自行开发的应用程序需要提供监控指标接口供采集，且监控数据需要满足Prometheus的规范，详情请参见**Prometheus监控数据采集说明**。

本文以Nginx为例采集监控数据，Nginx本身有个名叫ngx_http_stub_status_module的模块，这个模块提供了基本的监控功能，通过在nginx.conf的配置可以提供一个对外访问Nginx监控数据的接口。

步骤1 登录一台可连接公网的Linux虚拟机，且要求可执行Docker命令。

步骤2 创建一个nginx.conf文件，如下所示，在http下添加server配置即可让nginx提供对外访问的监控数据的接口。

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;

    server {
      listen 8080;
      server_name  localhost;
      location /stub_status {
         stub_status on;
         access_log off;
      }
    }
}
```

步骤3 使用该配置制作一个镜像，创建Dockerfile文件。


```
vi Dockerfile
```

Dockerfile文件内容如下所示：

```
FROM nginx:1.21.5-alpine
ADD nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

步骤4 使用上面Dockerfile构建镜像并上传到SWR镜像仓库，镜像名称为nginx:exporter。上传镜像的具体方法请参见**客户端上传镜像**。

1. 在左侧导航栏选择“我的镜像”，单击右侧“客户端上传”，在弹出的页面中单击“生成临时登录指令”，单击 复制登录指令。
2. 在集群节点上执行上一步复制的登录指令，登录成功会显示“Login Succeeded”。
3. 执行如下命令构建镜像，镜像名称为nginx，版本为exporter。

   ```
   docker build -t nginx:exporter .
   ```

4. 为镜像打标签并上传至镜像仓库，其中镜像仓库地址和组织名称请根据实际情况修改。

   ```
   docker tag nginx:exporter swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:exporter
   docker push swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:exporter
   ```

步骤5 查看应用指标。

1. 使用nginx:exporter创建工作负载。
2. **登录到容器中**，并通过http://<ip_address>:8080/stub_status获取到nginx的监控数据，其中<ip_address>为容器的IP地址，监控数据如下所示。

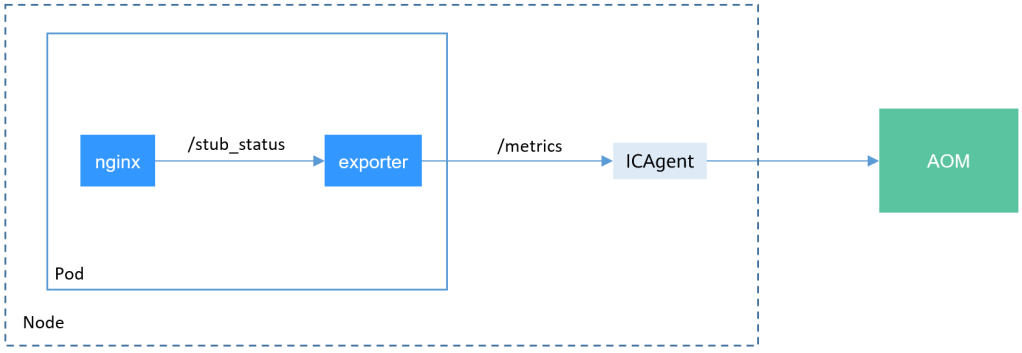
   ```
   # curl http://127.0.0.1:8080/stub_status
   Active connections: 3
   server accepts handled requests
    146269 146269 212
   Reading: 0 Writing: 1 Waiting: 2
   ```

----结束

部署应用并转换指标

如上所述的nginx:exporter提供的监控数据，其数据格式并不满足Prometheus的要求，需要将其转换成Prometheus需要的格式，可以使用**nginx-prometheus-exporter**来转换Nginx的指标，如下所示。

图 9-111 使用 exporter 转换数据格式



使用nginx:exporter和nginx-prometheus-exporter部署到同一个Pod，如下所示。

```
kind: Deployment
apiVersion: apps/v1
```



```
metadata:
  name: nginx-exporter
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-exporter
  template:
    metadata:
      labels:
        app: nginx-exporter
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"prometheus","path":"/
metrics","port":"9113","names":""}]'
    spec:
      containers:
        - name: container-0
          image: 'nginx:exporter'  # 替换为您上传到SWR的镜像地址
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
        - name: container-1
          image: 'nginx/nginx-prometheus-exporter:0.9.0'
          command:
            - nginx-prometheus-exporter
          args:
            - '-nginx.scrape-uri=http://127.0.0.1:8080/stub_status'
      imagePullSecrets:
        - name: default-secret
```

说明

nginx/nginx-prometheus-exporter:0.9.0需要从公网拉取，需要集群节点绑定公网IP。

nginx-prometheus-exporter需要一个启动命令，nginx-prometheus-exporter -nginx.scrape-uri=http://127.0.0.1:8080/stub_status，用于获取nginx的监控数据。

另外Pod需要添加一个annotations，metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"prometheus","path":"**/metrics**","port":"**9113**","names":""}]'。

配置验证

应用部署后，可以通过访问Nginx构造一些访问数据，然后在AOM中查看是否能够获取到相应的监控数据。

步骤1 获取Nginx Pod名称。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
nginx-exporter-78859765db-6j8sw   2/2     Running   0          4m
```

步骤2 登录容器执行命令访问Nginx。

```
$ kubectl exec -it nginx-exporter-78859765db-6j8sw -- /bin/sh
Defaulting container name to container-0.
Use 'kubectl describe pod/nginx-exporter-78859765db-6j8sw -n default' to see all of the containers in this
pod.
/ # curl http://localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```



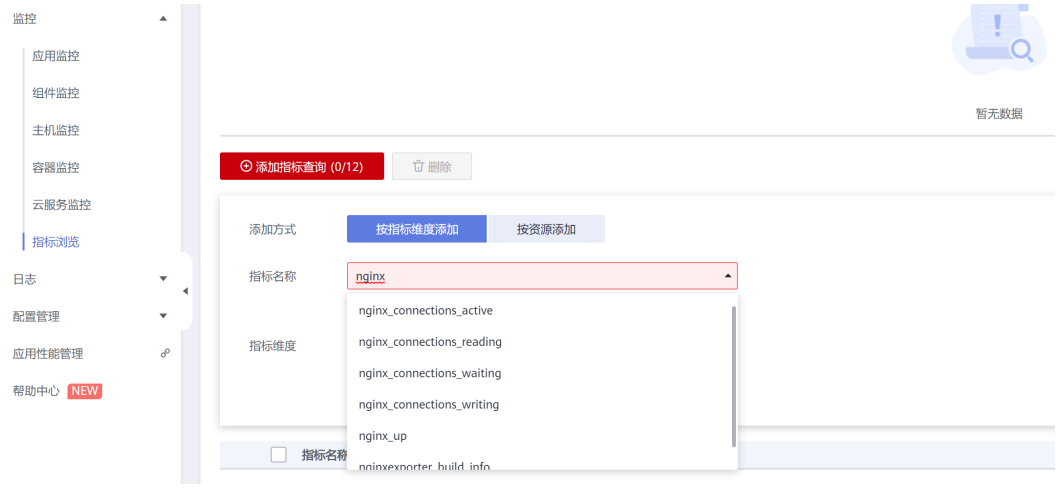
```
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<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>
/ #
```

步骤3 登录AOM，在左侧目录选择“监控 > 指标浏览”，查看Nginx相关的监控指标，如“nginx_connections_active”。

图 9-112 查看监控指标



----结束

9.8.4 使用 Prometheus 监控 Master 节点组件指标

本文将介绍如何使用Prometheus对Master节点的kube-apiserver、kube-controller、kube-scheduler、etcd-server组件进行监控。

通过监控中心查看 Master 节点组件指标

云原生监控中心已支持对Master节点的kube-apiserver组件进行监控，您在集群中开通云原生监控中心后（安装**云原生监控插件**版本为3.5.0及以上），可以查看仪表盘中的**APIServer视图**，监控API指标。

如需对kube-controller、kube-scheduler、etcd-server组件进行监控，请参考以下步骤。

说明

此3个组件监控指标不在容器基础指标范围，监控中心将该类指标上报至AOM后会进行收费，因此监控中心会默认屏蔽采集该类指标。

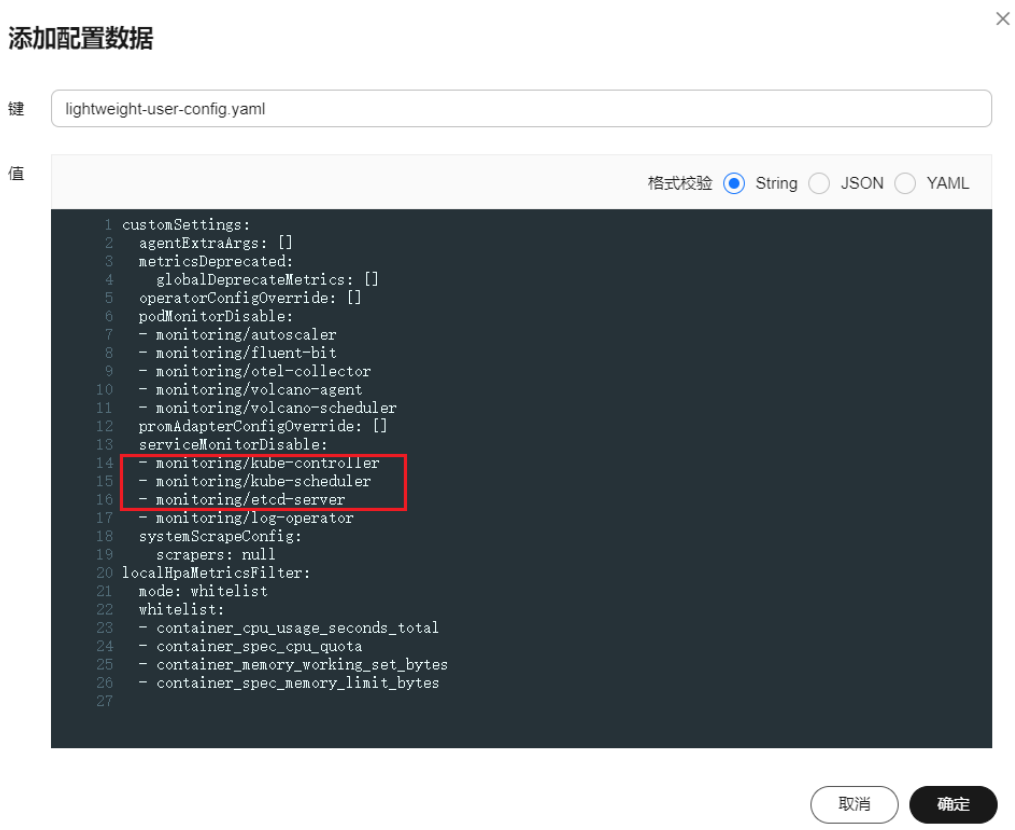
步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“配置与密钥”，并切换至“monitoring”命名空间，找到名为“persistent-user-config”的配置项。

步骤3 单击“更新”，对配置数据进行编辑，并在serviceMonitorDisable字段下**删除**以下配置。

```
serviceMonitorDisable:
- monitoring/kube-controller
- monitoring/kube-scheduler
- monitoring/etcd-server
- monitoring/log-operator
```

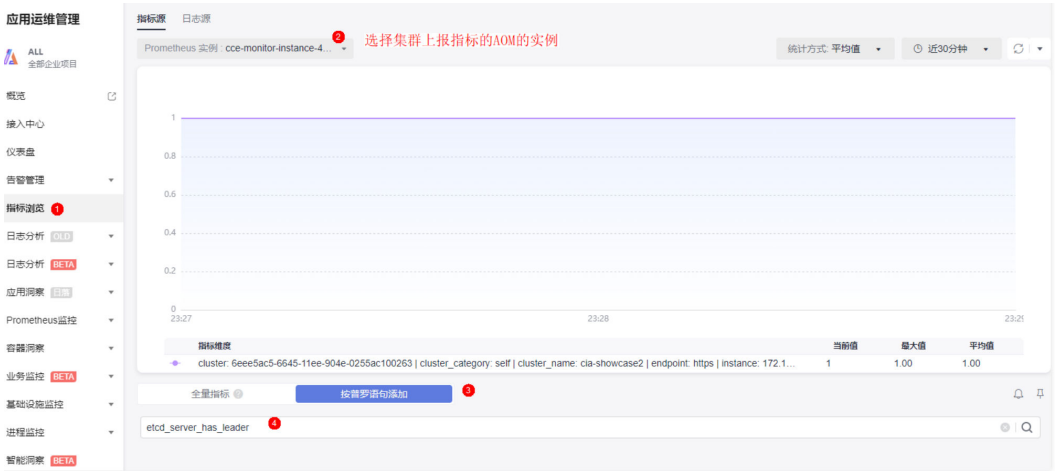
图 9-113 删除配置



步骤4 单击“确定”。

步骤5 等待5分钟后，您可前往AOM控制台，在“指标浏览”中找到集群上报的AOM实例，查看上述组件的指标。

图 9-114 查看指标



----结束

自建 Prometheus 采集 Master 节点组件指标

如果您需要通过Prometheus采集Master节点组件指标，可通过以下指导进行配置。

须知

- 集群版本需要v1.19及以上。
- 在集群中需安装自建的Prometheus，您可参考**Prometheus**使用Helm模板进行安装。安装自建Prometheus后，还需要使用prometheus-operator纳管该Prometheus实例，具体操作步骤请参见**Prometheus Operator**。

  由于**Prometheus（停止维护）**插件版本已停止演进，不再支持该功能特性，请避免使用。

步骤1 使用**kubectl**连接集群。

步骤2 修改Prometheus的ClusterRole。

```
kubectl edit ClusterRole prometheus -n {namespace}
```

在rules字段添加以下内容：

```
rules:
...
- apiGroups:
  - proxy.exporter.k8s.io
  resources:
  - "*"
  verbs: ["get", "list", "watch"]
```

步骤3 创建并编辑kube-apiserver.yaml文件。

```
vi kube-apiserver.yaml
```

文件内容如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: apiserver
  name: kube-apiserver
```



```
  namespace: monitoring    #修改为Prometheus安装的命名空间
spec:
  endpoints:
  - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
    interval: 30s
    metricRelabelings:
    - action: keep
      regex: (aggregator_unavailable_apiservice|
apiserver_admission_controller_admission_duration_seconds_bucket|
apiserver_admission_webhook_admission_duration_seconds_bucket|
apiserver_admission_webhook_admission_duration_seconds_count|
apiserver_client_certificate_expiration_seconds_bucket|apiserver_client_certificate_expiration_seconds_count|
apiserver_current_inflight_requests|apiserver_request_duration_seconds_bucket|apiserver_request_total|
go_goroutines|kubernetes_build_info|process_cpu_seconds_total|process_resident_memory_bytes|
rest_client_requests_total|workqueue_adds_total|workqueue_depth|
workqueue_queue_duration_seconds_bucket|aggregator_unavailable_apiservice_total|
rest_client_request_duration_seconds_bucket)
      sourceLabels:
      - __name__
    - action: drop
      regex: apiserver_request_duration_seconds_bucket;(0.15|0.25|0.3|0.35|0.4|0.45|0.6|0.7|0.8|0.9|1.25|1.5|1.75|
2.5|3|3.5|4.5|6|7|8|9|15|25|30|50)
      sourceLabels:
      - __name__
      - le
    port: https
    scheme: https
    tlsConfig:
      caFile: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      serverName: kubernetes
  jobLabel: component
  namespaceSelector:
    matchNames:
    - default
  selector:
    matchLabels:
      component: apiserver
      provider: kubernetes
```

创建ServiceMonitor：

```
kubectl apply -f kube-apiserver.yaml
```

步骤4 创建并编辑kube-controller.yaml文件。

```
vi kube-controller.yaml
```

文件内容如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-controller
  name: kube-controller-manager
  namespace: monitoring    #修改为Prometheus安装的命名空间
spec:
  endpoints:
    - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
      interval: 15s
      honorLabels: true
      port: https
      relabelings:
        - regex: (.+)
          replacement: /apis/proxy.exporter.k8s.io/v1beta1/kube-controller-proxy/${1}/metrics
          sourceLabels:
            - __address__
          targetLabel: __metrics_path__
        - regex: (.+)
          replacement: ${1}
          sourceLabels:
```



```
        - __address__
        targetLabel: instance
      - replacement: kubernetes.default.svc.cluster.local:443
        targetLabel: __address__
    scheme: https
    tlsConfig:
      caFile: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  jobLabel: app
  namespaceSelector:
    matchNames:
      - kube-system
  selector:
    matchLabels:
      app: kube-controller-proxy
      version: v1
```

创建ServiceMonitor：

```
kubectl apply -f kube-controller.yaml
```

步骤5 创建并编辑kube-scheduler.yaml文件。

```
vi kube-scheduler.yaml
```

文件内容如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-scheduler
  name: kube-scheduler
  namespace: monitoring    #修改为Prometheus安装的命名空间
spec:
  endpoints:
    - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
      interval: 15s
      honorLabels: true
      port: https
      relabelings:
        - regex: (.+)
          replacement: /apis/proxy.exporter.k8s.io/v1beta1/kube-scheduler-proxy/${1}/metrics
          sourceLabels:
            - __address__
          targetLabel: __metrics_path__
        - regex: (.+)
          replacement: ${1}
          sourceLabels:
            - __address__
          targetLabel: instance
        - replacement: kubernetes.default.svc.cluster.local:443
          targetLabel: __address__
      scheme: https
      tlsConfig:
        caFile: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  jobLabel: app
  namespaceSelector:
    matchNames:
      - kube-system
  selector:
    matchLabels:
      app: kube-scheduler-proxy
      version: v1
```

创建ServiceMonitor：

```
kubectl apply -f kube-scheduler.yaml
```

步骤6 创建并编辑etcd-server.yaml文件。

```
vi etcd-server.yaml
```

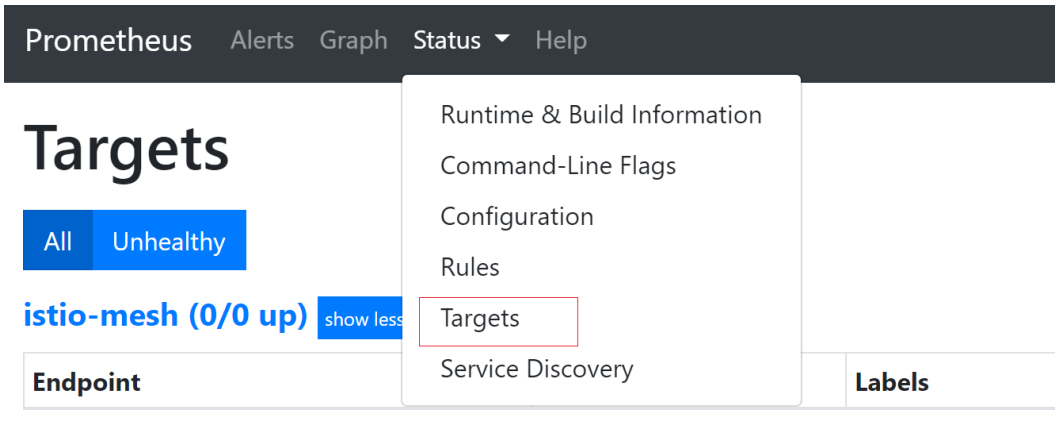

文件内容如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: etcd-server
  name: etcd-server
  namespace: monitoring    #修改为Prometheus安装的命名空间
spec:
  endpoints:
    - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
      interval: 15s
      honorLabels: true
      port: https
      relabelings:
        - regex: (.+)
          replacement: /apis/proxy.exporter.k8s.io/v1beta1/etcd-server-proxy/${1}/metrics
          sourceLabels:
            - __address__
          targetLabel: __metrics_path__
        - regex: (.+)
          replacement: ${1}
          sourceLabels:
            - __address__
          targetLabel: instance
        - replacement: kubernetes.default.svc.cluster.local:443
          targetLabel: __address__
      scheme: https
      tlsConfig:
        caFile: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  jobLabel: app
  namespaceSelector:
    matchNames:
      - kube-system
  selector:
    matchLabels:
      app: etcd-server-proxy
      version: v1
```

创建ServiceMonitor：

```
kubectl apply -f etcd-server.yaml
```

步骤7 创建完成后，访问Prometheus，单击“Status > Targets”，可以查看到Prometheus监控目标中已包含上述三个Master节点组件。



----结束

9.8.5 监控 NGINX Ingress 控制器指标

通过Prometheus和Grafana，可以实现对NGINX Ingress控制器指标的观测。

本文以实际示例介绍如何通过Prometheus查看集群的NGINX Ingress控制器指标，具体步骤如下：

1. **访问Prometheus**

   （可选）为Prometheus绑定LoadBalancer类型的Service，支持从外部访问Prometheus。

2. **监控NGINX Ingress控制器指标**

   在集群中部署使用NGINX Ingress控制器时，打开“开启指标采集”开关后将自动上报NGINX Ingress控制器指标。

前提条件

- 集群中已安装3.9.5及以上版本**云原生监控插件**插件。
- 集群中已安装2.5.4及以上版本的**NGINX Ingress控制器**插件，且已打开“开启指标采集”开关。

访问 Prometheus

云原生监控插件安装完成后会在集群中部署一系列工作负载和Service。其中Prometheus的Server端会在monitoring命名空间下以有状态工作负载进行部署。

您可以创建一个公网**LoadBalancer类型Service**，这样就可以从外部访问Prometheus。

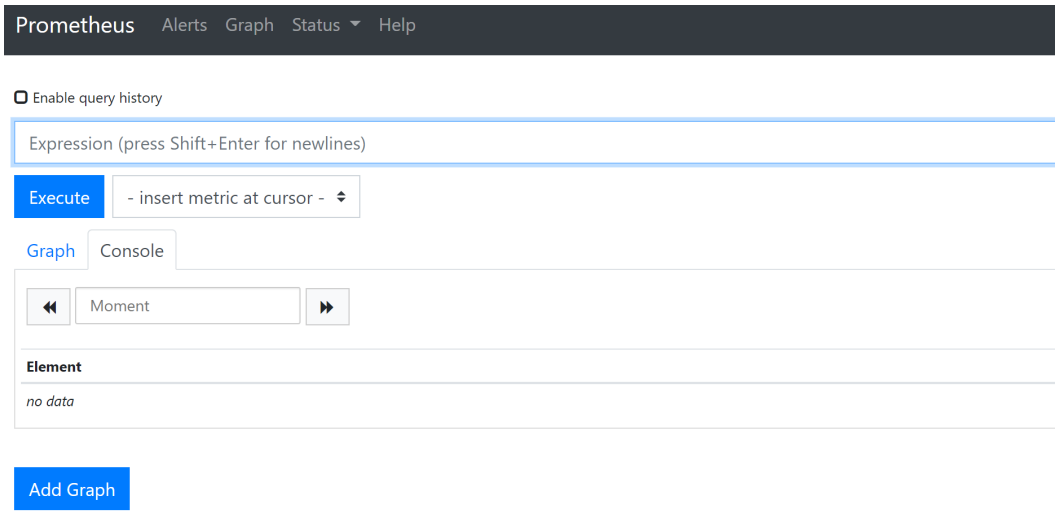
步骤1 登录CCE控制台，选择一个已安装Prometheus的集群，在左侧导航栏中选择“服务”。

步骤2 单击右上角“YAML创建”，创建一个公网LoadBalancer类型的Service。

```
apiVersion: v1
kind: Service
metadata:
  name: prom-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: prometheus
    component: server
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 88            #服务端口号，可自定义
      targetPort: 9090     #Prometheus的默认端口号，无需更改
  selector:               #标签选择器可根据Prometheus Server实例的标签进行调整
    app.kubernetes.io/name: prometheus
    prometheus: server
  type: LoadBalancer
```

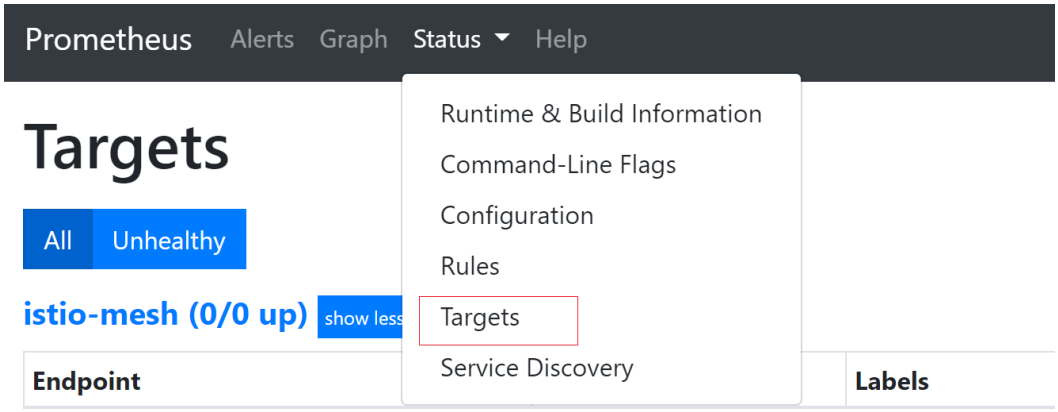
步骤3 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Prometheus。

图 9-115 访问 Prometheus



步骤4 单击“Status > Targets”，可以查看到Prometheus监控了哪些目标。

图 9-116 查看监控目标



----结束

监控 NGINX Ingress 控制器指标

访问Prometheus，在“Graph”页面中，查看NGINX Ingress控制器指标。

图 9-117 查看 NGINX Ingress 控制器监控指标

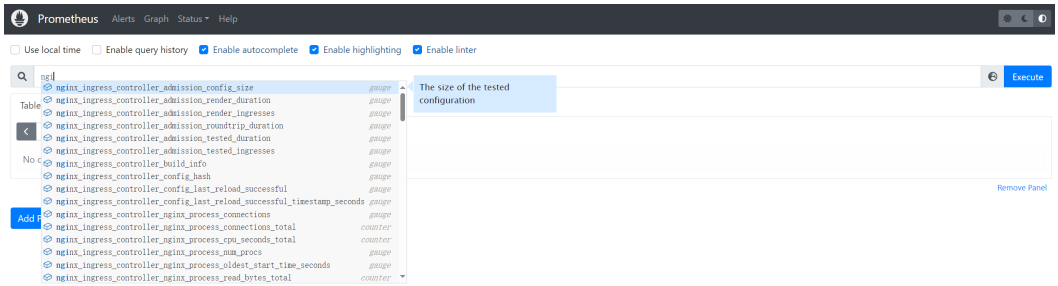


表 9-51 NGINX Ingress 控制器监控指标

指标	指标类型	说明
nginx_ingress_controller_bytes_sent	基础指标	发送到客户端的字节数
nginx_ingress_controller_connect_duration_seconds	基础指标	与上游服务器建立连接花费的时间
nginx_ingress_controller_header_duration_seconds	基础指标	从上游服务器接收第一个报头所花费的时间
nginx_ingress_controller_ingress_upstream_latency_seconds	基础指标	上游服务时延
nginx_ingress_controller_request_duration_seconds	基础指标	请求处理时间（以毫秒为单位）
nginx_ingress_controller_request_size	基础指标	请求长度(包括请求行、请求头和请求体)
nginx_ingress_controller_requests	基础指标	客户端请求的总数
nginx_ingress_controller_response_duration_seconds	基础指标	从上游服务器接收响应所花费的时间
nginx_ingress_controller_response_size	基础指标	响应长度(包括请求行、报头和请求体)
nginx_ingress_controller_nginx_process_connections	基础指标	当前状态为{活动，读取，写入，等待}的客户端连接数
nginx_ingress_controller_nginx_process_connections_total	基础指标	状态为{已接受，已处理}的连接总数
nginx_ingress_controller_nginx_process_cpu_seconds_total	基础指标	CPU使用率(秒)
nginx_ingress_controller_nginx_process_num_procs	基础指标	进程数
nginx_ingress_controller_nginx_process_oldest_start_time_seconds	基础指标	从1970年1月1日开始以秒为单位的开始时间
nginx_ingress_controller_nginx_process_read_bytes_total	基础指标	读取的字节数
nginx_ingress_controller_nginx_process_requests_total	基础指标	客户端请求总数
nginx_ingress_controller_nginx_process_resident_memory_bytes	基础指标	正在使用的内存字节数

指标	指标类型	说明
nginx_ingress_controller_nginx_process_virtual_memory_bytes	基础指标	正在使用的内存字节数
nginx_ingress_controller_nginx_process_write_bytes_total	基础指标	写入字节数
nginx_ingress_controller_build_info	基础指标	一个带有常量“1”的度量，标记有关于构建的信息。
nginx_ingress_controller_check_success	基础指标	Ingress controller语法检查累计次数
nginx_ingress_controller_config_hash	基础指标	正在运行的Nginx配置hash值
nginx_ingress_controller_config_last_reload_successful	基础指标	最后一次尝试重新加载配置是否成功
nginx_ingress_controller_config_last_reload_successful_timestamp_seconds	基础指标	最后一次成功重新加载配置的时间戳
nginx_ingress_controller_ssl_certificate_info	基础指标	保留与证书相关的所有标签
nginx_ingress_controller_success	基础指标	Ingress controller重新加载操作的累计次数
nginx_ingress_controller_orphan_ingress	基础指标	孤立ingress的状态，1表示孤立ingress。 ● namespace：是用于标识ingress名称空间的字符串。 ● ingress：表示ingress名称。 ● type：表示孤立ingress的状态，取值为no-service或no-endpoint。
nginx_ingress_controller_admission_config_size	基础指标	被测试配置的大小
nginx_ingress_controller_admission_render_duration	基础指标	允许ingress渲染入口的处理持续时间(浮点秒)
nginx_ingress_controller_admission_render_ingresses	基础指标	由admission controller渲染的ingress长度
nginx_ingress_controller_admission_roundtrip_duration	基础指标	admission controller在处理新事件时的完整持续时间(浮点秒)
nginx_ingress_controller_admission_tested_duration	基础指标	admission controller测试的处理持续时间(浮点秒)

指标	指标类型	说明
nginx_ingress_controller_admission_tested_ingresses	基础指标	admission controller处理的ingress长度

说明

Nginx Ingress在高负载请求下，开启全量指标采集存在内存泄露的问题（详情请参见**社区issue**）。经过验证，屏蔽以下指标后，能够有效抑制内存增长。为避免内存泄露导致业务受损，Nginx Ingress插件默认屏蔽以下指标。我们将持续关注社区最新动态，及时同步修复该问题。

- nginx_ingress_controller_success
- nginx_ingress_controller_header_duration_seconds
- nginx_ingress_controller_ingress_upstream_latency_seconds

9.8.6 监控 CCE Turbo 集群容器网络扩展指标

CCE容器网络扩展指标插件（dolphin）是一款容器网络流量监控管理插件，可支持CCE Turbo集群非主机网络容器的流量统计，以及节点内容器连通性健康检查。监控信息已适配Prometheus格式，可以通过调用Prometheus接口查看监控数据。

本文以实际示例介绍如何通过Prometheus查看CCE Turbo集群容器网络扩展指标，具体步骤如下：

1. **安装插件**
2. **监控容器网络扩展指标**
3. **（可选）通过Grafana查看图表**

前提条件

- 已创建一个CCE Turbo集群。
- 集群中存在足够的节点资源（不小于4U8G），用于安装**云原生监控插件**和**CCE容器网络扩展指标**插件。
- 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

安装插件

步骤1 登录CCE控制台，单击CCE Turbo集群名称进入集群，单击左侧导航栏的“插件中心”。

步骤2 在“插件中心”页面右侧找到**云原生监控插件**，单击“安装”。

在监控CCE Turbo集群容器网络扩展指标的场景下，建议您关注以下配置。该插件的其他配置可按需进行设置，详情请参见**云原生监控插件**。

- 本地数据存储：此处选择使用本地存储监控数据，监控数据可选择是否对接AOM或三方监控平台。
- 自定义指标采集：该配置在本实践中必须选择开启，否则将无法采集容器网络扩展指标。

- （可选）安装Grafana：选择安装Grafana后，可以使用图表查看指标。

  说明

  该配置在3.9.0以下版本的插件中支持。对于3.9.0及以上版本的插件，如果存在使用Grafana的需求，请**单独安装Grafana**。

步骤3 在“插件中心”页面右侧找到**CCE容器网络扩展指标**插件，单击“安装”。

当前该插件无可配置参数。

步骤4 （可选）对于3.9.0及以上版本的**云原生监控插件**，不再默认提供Grafana组件。您可以在“插件中心”页面右侧找到独立的**Grafana**插件，单击“安装”。

选择“公网访问”，将会在monitoring命名空间创建一条名为grafana-oss的LoadBalancer类型的服务。如果LoadBalancer服务对接的ELB绑定了EIP，可直接使用浏览器输入“eip:port”地址进行访问。

须知

开启“公网访问”将会把开源Grafana服务暴露至公网访问，建议评估安全风险并做好访问策略的管控。

----结束

监控容器网络扩展指标

步骤1 编辑容器网络扩展指标插件的DaemonSet配置，添加Ports信息。

说明

1.3.10以下版本的容器网络扩展指标插件需手动操作，1.3.10及以上版本自动添加该配置，可跳过此步骤。

```
kubectl edit ds -nkube-system dolphin
```

添加如下配置：

```
...
spec:
  containers:
    - name: dolphin
      ports:
      - containerPort: 10001
        name: dolphin
        protocol: TCP
...
```

步骤2 配置PodMonitor后，Prometheus就会自动采集CCE容器网络扩展指标。

pod-monitor.yaml文件配置参考如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: dolphin
  namespace: monitoring
spec:
  namespaceSelector:
    matchNames:
    - kube-system
  jobLabel: podmonitor-dolphin
  podMetricsEndpoints:
```



```
- interval: 15s
  path: /metrics
  port: dolphin
  tlsConfig:
    insecureSkipVerify: true
selector:
  matchLabels:
    app: dolphin
```

创建PodMonitor资源：

```
kubectl apply -f pod-monitor.yaml
```

----结束

通过 Prometheus 查看指标

步骤1 创建一个示例监控任务，详情请参见**下发监控任务**。

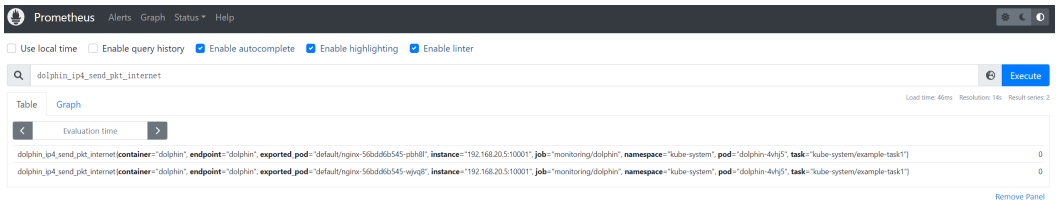
```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task          #监控任务名
  namespace: kube-system       #必填，namespace必须为kube-system
spec:
  selector:                   #选填，配置dolphin插件监控的后端，形如labelSelector格式，默认将监控本节点所有容器
    matchLabels:
      app: nginx
    matchExpressions:
      - key: app
        operator: In
        values:
          - nginx
  podLabel: []                #选填，用户标签
  ip4Tx:                      #选填，ipv4发送报文数和发送字节数这两个指标的开关，默认不开
    enable: true
  ip4Rx:                      #选填，ipv4接收报文数和接收字节数这两个指标的开关，默认不开
    enable: true
  ip4TxInternet:              #选填，ipv4发送公网报文数和发送公网字节数这两个指标的开关，默认不开
    enable: true
  healthCheck:                #选填，本地节点 Pod 健康检查任务中最近一次健康检查是否健康、健康检查总健康&不健康次数这三个指标开关，默认不开
    enable: true              # true false
    failureThreshold: 3       #选填，健康检查不健康判定失败次数，默认1次健康检查失败即判定不健康
    periodSeconds: 5          #选填，健康检查任务检查间隔时间，单位秒，默认60
    command: ""               #选填，健康检查任务检查命令，支持：ping、arping、curl，默认 ping
    ipFamilies: [""]          #选填，健康检查IP地址族，支持：ipv4，默认ipv4
    port: 80                  #选填，使用curl时必选，端口号
    path: ""                  #选填，使用curl时必选，http api 路径
  monitor:
    ip:
      ipReceive:
        aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
      ipSend:
        aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    tcp:
      tcpReceive:
        aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
      tcpSend:
        aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
      tcpRetrans:
        aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
      tcpRtt:
        aggregateType: flow      #选填，支持填写"flow"，表示流粒度监控，单位：微秒
      tcpNewConnection:
        aggregateType: pod       #选填，支持填写"pod"，表示pod粒度监控
```


步骤2 为Prometheus创建一个公网LoadBalancer类型的Service，提供公网访问。

```
apiVersion: v1
kind: Service
metadata:
  name: prom-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: prometheus
    component: server
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 88            #服务端口号，可自定义
      targetPort: 9090     #Prometheus的默认端口号，无需更改
  selector:               #标签选择器可根据Prometheus Server实例的标签进行调整
    app.kubernetes.io/name: prometheus
    prometheus: server
  type: LoadBalancer
```

步骤3 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Prometheus。您可以在Prometheus页面中搜索**支持的监控项**，验证指标是否采集成功。

图 9-118 访问 Prometheus



----结束

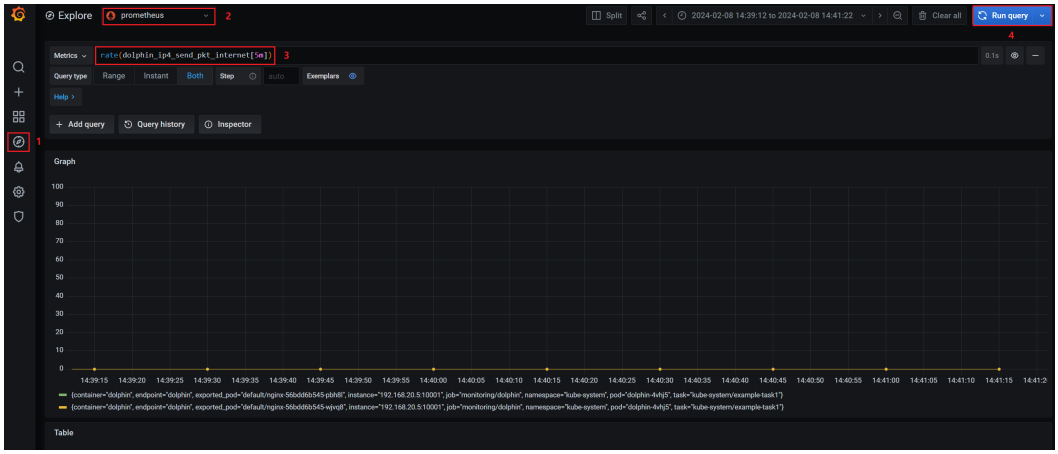
（可选）通过 Grafana 查看图表

步骤1 在集群中安装Grafana后，在“插件中心”页面右侧找到Grafana插件，单击“访问”。

步骤2 输入您的Grafana登录账号及密码。

步骤3 单击Grafana页面左侧导航栏中的“Explore”，然后在页面上方选择“Prometheus”，输入PromQL查询指令，例如“rate(dolphin_ip4_send_pkt_internet[5m])”，然后单击右上角“Run query”即可获取指标图表。

图 9-119 Grafana 图表



步骤4 您也可以将常用图表固定为Grafana Dashboard，详情请参见**Create a dashboard**。

----结束

10 弹性伸缩

10.1 弹性伸缩概述

弹性伸缩是根据业务需求和策略，经济地自动调整弹性计算资源的管理服务。

背景介绍

随着Kubernetes已经成为云原生应用编排、管理的事实标准，越来越多的应用选择向Kubernetes迁移，用户也越来越关心在Kubernetes上应用如何快速扩容面对业务高峰，以及如何在业务低谷时快速缩容节约资源与成本。

在Kubernetes的集群中，“弹性伸缩”一般涉及到扩缩容Pod个数以及Node个数。Pod代表应用的实例数（每个Pod包含一个或多个容器），当业务高峰的时候需要扩容应用的实例个数。所有的Pod都是运行在某一个节点（虚机或裸机）上，当集群中没有足够多的节点来调度新扩容的Pod，那么就需要为集群增加节点，从而保证业务能够正常提供服务。

弹性伸缩在CCE上的使用场景非常广泛，典型的场景包含在线业务弹性、大规模计算训练、深度学习GPU或共享GPU的训练与推理、定时周期性负载变化等。

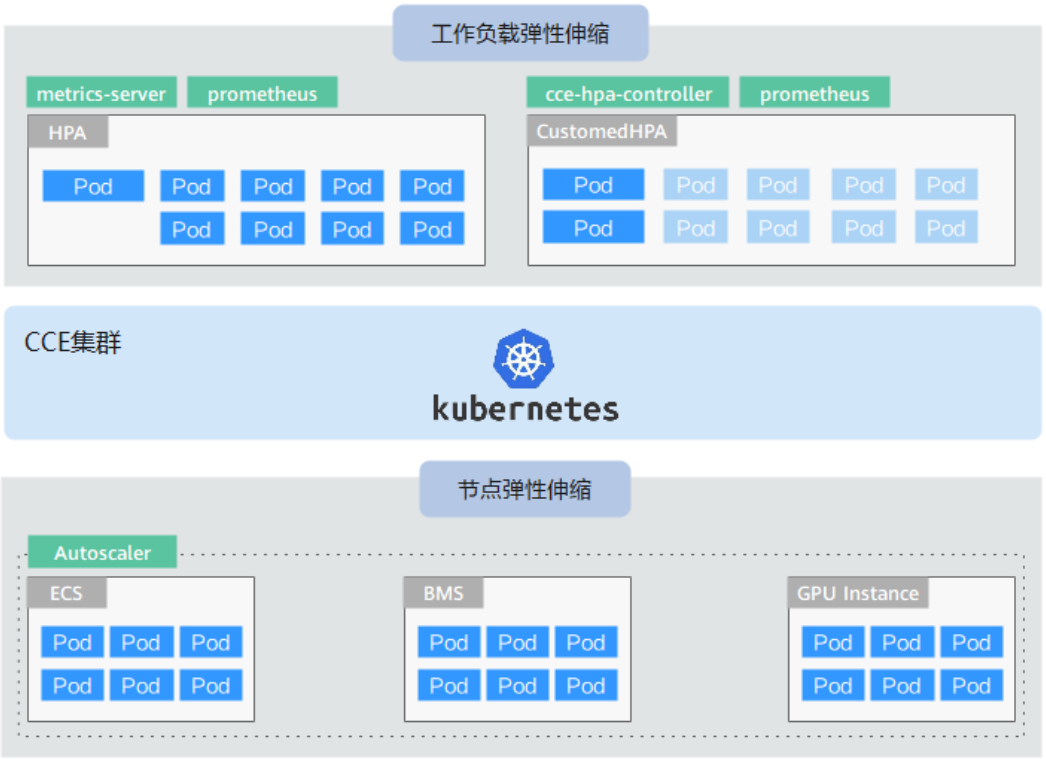
CCE 弹性伸缩

CCE的弹性伸缩能力分为如下两个维度：

- **工作负载弹性伸缩**：即调度层弹性，主要是负责修改负载的调度容量变化。例如，HPA是典型的调度层弹性组件，通过HPA可以调整应用的副本数，调整的副本数会改变当前负载占用的调度容量，从而实现调度层的伸缩。
- **节点弹性伸缩**：即资源层弹性，主要是集群的容量规划不能满足集群调度容量时，会通过弹出ECS资源的方式进行调度容量的补充。

两个维度的弹性组件与能力可以分开使用，也可以结合在一起使用，并且两者之间可以通过调度层面的容量状态进行解耦，详情请参见**使用HPA+CA实现工作负载和节点联动弹性伸缩**。

组件介绍



工作负载弹性伸缩类型介绍

表 10-1 工作负载弹性伸缩类型

类型	组件	组件介绍	参考文档
HPA	HorizontalPodAutoscaler（Kubernetes内置组件）	HorizontalPodAutoscaler是Kubernetes内置组件，实现Pod水平自动伸缩（Horizontal Pod Autoscaling）的功能。CCE在Kubernetes社区HPA功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。	创建HPA策略
CustomedHPA	CCE容器弹性引擎	CustomedHPA提供弹性伸缩增强能力，主要面向无状态工作负载进行弹性扩缩容。能够基于指标（CPU利用率、内存利用率）或周期（每天、每周、每月或每年的具体时间点）。	创建CustomedHPA策略
CronHPA	CCE容器弹性引擎	CronHPA可以实现在固定时间段对集群进行扩缩容，并且可以和HPA策略共同作用，定时调整HPA伸缩范围，实现复杂场景下的工作负载伸缩。	创建CronHPA定时策略

节点弹性伸缩类型介绍

表 10-2 节点弹性伸缩类型

组件名称	组件介绍	适用场景	参考文档
CCE集群弹性引擎	Kubernetes社区开源组件，用于节点水平伸缩，CCE在其基础上提供了独有的调度、弹性优化、成本优化的功能。	全场景支持，适合在线业务、深度学习、大规模成本算力交付等。	节点自动伸缩
CCE突发弹性引擎（对接CCI）	将Kubernetes API扩展到无服务器的容器平台（如CCI），无需关心节点资源。	适合在线突增流量、CI/CD、大数据作业等场景。	CCE容器实例弹性伸缩到CCI服务

10.2 工作负载弹性伸缩

10.2.1 工作负载伸缩原理

CCE支持多种工作负载伸缩方式，策略对比如下：

表 10-3 弹性伸缩策略对比

伸缩策略	HPA策略	CronHPA策略	CustomedHPA策略	VPA策略	AHPA策略
策略介绍	Kubernetes中实现POD水平自动伸缩的功能，即Horizontal Pod Autoscaling。	基于HPA策略的增强能力，主要面向应用资源使用率存在周期性变化的场景。	CCE自研的弹性伸缩增强能力，可实现基于指标触发或定时触发弹性伸缩。	Kubernetes中实现POD垂直自动伸缩的功能，即Vertical Pod Autoscaling。	AHPA策略即Advanced Horizontal Pod Autoscaling，可以根据历史数据提前进行扩缩容动作。
策略规则	基于**指标**（CPU利用率、内存利用率），对无状态工作负载的**副本数**进行弹性扩缩容。	基于**周期**（每天、每周、每月或每年的具体时间点），对无状态工作负载的**副本数**进行弹性扩缩容。	基于**指标**（CPU利用率、内存利用率）或**周期**（每天、每周、每月或每年的具体时间点），对无状态工作负载的**副本数**进行弹性扩缩容。	基于容器资源（CPU、内存）**历史使用情况**，对工作负载的**资源申请量**进行扩缩容。	基于容器资源（CPU、内存）**历史使用情况**进行预测，提前对工作负载**副本数**进行弹性扩缩容。

伸缩策略	HPA策略	CronHPA策略	CustomedHPA策略	VPA策略	AHPA策略
主要功能	在Kubernetes社区HPA功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。	CronHPA提供HPA对象的兼容能力，您可以同时使用CronHPA与HPA。 • CronHPA与HPA策略共同使用：CronHPA作用于HPA策略之上，用于定时调整HPA策略的实例数范围。 • CronHPA策略单独使用：CronHPA直接定时调整工作负载的Pod实例数。	**指标触发** • 支持按照当前实例数的百分比进行扩缩容。 • 支持设置一次扩缩容的最小步长，可分步分级扩缩容。 • 支持按照实际指标值执行不同的扩缩容动作。 **周期触发** 支持选择天、周、月或年的具体时间点或周期作为触发时间	根据CPU、内存历史使用情况自动计算建议值，并调整Pod资源申请量。	根据业务历史指标，识别工作负载弹性周期并对未来波动进行预测，提前进行扩缩容动作，解决原生HPA的滞后问题。
使用方式	**创建HPA策略**	**创建CronHPA定时策略**	**创建CustomedHPA策略**	**创建VPA策略**	**创建AHPA策略**

HPA 工作原理

HPA（Horizontal Pod Autoscaler）是用来控制Pod水平伸缩的控制器，HPA周期性检查Pod的度量数据，计算满足HPA资源所配置的目标数值所需的副本数量，进而调整目标资源（如Deployment）的replicas字段。

想要做到自动弹性伸缩，先决条件就是能感知到各种运行数据，例如集群节点、Pod、容器的CPU、内存使用率等等。而这些数据的监控能力Kubernetes也没有自己实现，而是通过其他项目来扩展Kubernetes的能力，Kubernetes提供**Prometheus**和**Metrics Server**插件来实现该能力：

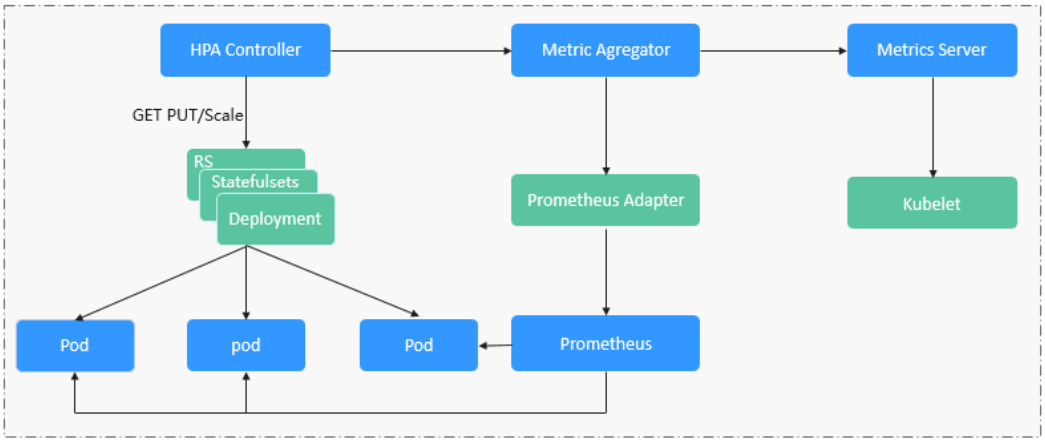
- **Prometheus**是一套开源的系统监控报警框架，能够采集丰富的Metrics（度量数据），目前已经基本是Kubernetes的标准监控方案。
- **Metrics Server**是Kubernetes集群范围资源使用数据的聚合器。Metrics Server从kubelet公开的Summary API中采集度量数据，能够收集包括了Pod、Node、容

器、Service等主要Kubernetes核心资源的度量数据，且对外提供一套标准的API。

使用HPA（Horizontal Pod Autoscaler）配合Metrics Server可以实现基于CPU和内存的自动弹性伸缩，再配合Prometheus还可以实现自定义监控指标的自动弹性伸缩。

HPA主要流程如图10-1所示。

图 10-1 HPA 流程图



HPA的核心有如下2个部分：

- 监控数据来源

  最早社区只提供基于CPU和Mem的HPA，随着应用越来越多搬迁到K8s上以及Prometheus的发展，开发者已经不满足于CPU和Memory，开发者需要应用自身的业务指标，或者是一些接入层的监控信息，例如：Load Balancer的QPS、网站的实时在线人数等。社区经过思考之后，定义了一套标准的Metrics API，通过聚合API对外提供服务。

  - metrics.k8s.io： 主要提供Pod和Node的CPU和Memory相关的监控指标。
  - custom.metrics.k8s.io： 主要提供Kubernetes Object相关的自定义监控指标。
  - external.metrics.k8s.io：指标来源外部，与任何的Kubernetes资源的指标无关。

- 扩缩容决策算法

  HPA controller根据当前指标和期望指标来计算缩放比例，计算公式如下：

  期望实例数 = 向上取整[当前实例数 * (当前的指标值 / 目标值)]

  例如当前的指标值是200m，目标值是100m，那么按照公式计算期望的实例数就会翻倍。那么在实际过程中，可能会遇到实例数值反复伸缩，导致集群震荡。为了保证稳定性，HPA controller从以下几个方面进行优化：

  - 冷却时间：在1.11版本以及之前的版本，社区引入了horizontal-pod-autoscaler-downscale-stabilization-window和horizontal-pod-autoScaler-upscale-stabilization-window这两个启动参数代表缩容冷却时间和扩容冷却时间，这样保证在冷却时间内，跳过扩缩容。1.14版本之后引入延迟队列，保存一段时间内每一次检测的决策建议，然后根据当前所有有效的决策建议来进行决策，从而保证期望的副本数尽量少地发生变更，保证稳定性。
  - 忍受度：可以看成一个缓冲区，当实例变化范围在忍受范围之内的话，保持原有的实例数不变。

首先定义**ratio = 当前的指标值 / 目标值**

当|ratio – 1.0| <= 忍受度时，则会忽略，跳过scale。

当|ratio – 1.0| > 忍受度时， 就会根据之前的公式计算期望值。

当前社区版本中默认值为0.1。

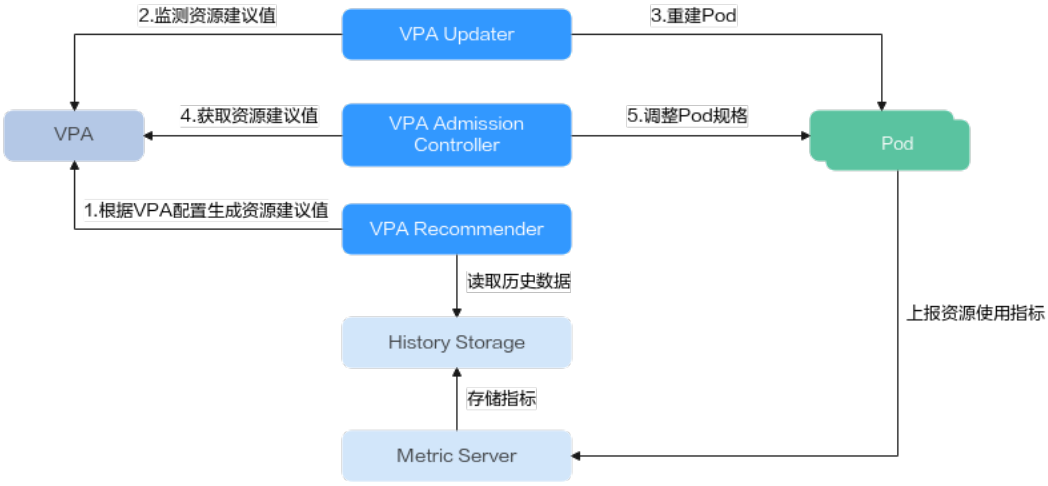
HPA是基于指标阈值进行伸缩的，常见的指标主要是 CPU、内存，也可以通过自定义指标，例如QPS、连接数等进行伸缩。但是存在一个问题：基于指标的伸缩存在一定的时延，这个时延主要包含：采集时延(分钟级) + 判断时延(分钟级) + 伸缩时延(分钟级)。这个分钟级的时延，可能会导致应用CPU飚高，响应时间变慢。为了解决这个问题，CCE提供了定时策略，对于一些有周期性变化的应用，提前扩容资源，而业务低谷时，定时回收资源。

VPA 工作原理

VPA主要包含三个组件：

- VPA Recommender：根据历史数据给出Pod资源调整建议。
- VPA Updater：对比建议值和当前值，不一致时重建Pod。
- VPA Admission Controller：在Pod重建时将Pod的资源申请量修改为建议值。

图 10-2 VPA 工作流程



VPA生效的主要流程如下：

1. 首先VPA Recommender组件会根据Pod的资源使用情况历史数据计算出Pod资源调整建议值。
2. 然后VPA Updater对比Pod资源当前值与VPA建议值是否一致，如果需要调整则重建Pod。
3. Pod发生重建时，VPA Admission Controller会进行拦截，并将VPA建议值调整为Pod的资源申请值。

10.2.2 创建 HPA 策略

HPA策略即Horizontal Pod Autoscaling，是Kubernetes中实现POD水平自动伸缩的功能。该策略在Kubernetes社区HPA功能的基础上，增加了应用级别的冷却时间窗和扩缩容阈值等功能。

前提条件

使用HPA需要安装能够提供Metrics API的插件，您可根据集群版本和实际需求选择其中之一：

- **Kubernetes Metrics Server**：提供基础资源使用指标，例如容器CPU和内存使用率。所有集群版本均可安装。
- **云原生监控插件**：该插件支持v1.17及以后的集群版本。
  - 根据基础资源指标进行弹性伸缩：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供基础资源指标**。
  - 根据自定义指标进行弹性伸缩：需要将自定义指标聚合到Kubernetes API Server，详情请参见**使用自定义指标创建HPA策略**。
- **Prometheus（停止维护）**：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供资源指标**。该插件仅支持v1.21及之前的集群版本。

约束与限制

- HPA策略：仅支持1.13及以上版本的集群创建。
- 1.19.10以下版本的集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，当新Pod被调度到另一个节点时，会导致之前Pod不能正常读写。

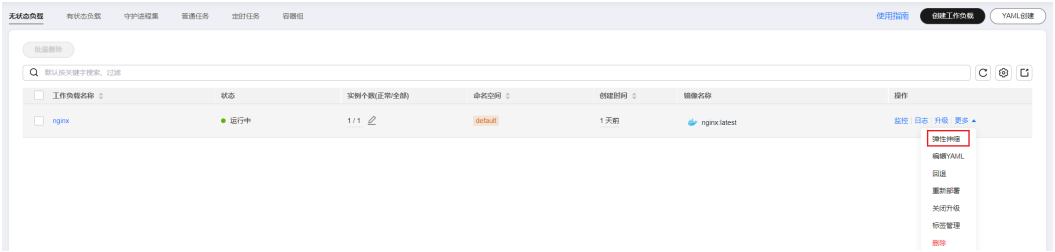
  1.19.10及以上版本集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，新Pod会因为无法挂载云硬盘导致无法成功启动。

创建 HPA 策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“更多 > 弹性伸缩”。

图 10-3 工作负载弹性伸缩



步骤3 策略类型选择“HPA+CronHPA策略”，并启用HPA策略，填写HPA策略配置参数。

本文中仅介绍HPA策略，如需启用CronHPA策略，请参见**创建CronHPA定时策略**。

图 10-4 启用 HPA 策略

表 10-4 HPA 策略配置

参数	参数说明
实例范围	请输入最小实例数和最大实例数。 策略触发时，工作负载实例将在此范围内伸缩。
冷却时间	请输入缩容和扩容的冷却时间，单位为分钟，**缩容扩容冷却时间不能小于1分钟**。 **该设置仅在1.15到1.23版本的集群中显示。** 策略成功触发后，在此缩容/扩容冷却时间内，不会再次触发缩容/扩容，目的是等待伸缩动作完成后在系统稳定且集群正常的情况下进行下一次策略匹配。
伸缩配置	**该设置仅在1.25及以上版本的集群中显示。** • 系统默认：采用社区推荐的默认行为进行负载伸缩，详情请参见**社区默认行为说明**。 • 自定义：自定义扩/缩容配置的稳定窗口、步长、优先级等策略，实现更灵活的配置。未配置的参数将采用社区推荐的默认值。 – 禁止扩/缩容：选择是否禁止扩容或缩容。 – 稳定窗口：需要伸缩时，会在一段时间（设定的稳定窗口值）内持续检测，如在该时间段内始终需要进行伸缩（不满足设定的指标期望值）才进行伸缩，避免短时间的指标抖动造成异常。 – 步长策略：扩/缩容的步长，可设置一定时间内扩/缩容Pod数量或百分比。在存在多条策略时，可以选择使Pod数量最多或最少的策略。

参数	参数说明
系统策略	• 指标：可选择“CPU利用率”或“内存利用率”。 **说明** 利用率 = 工作负载所有Pod实际资源使用量的平均值 / 资源申请量（Request） • 期望值：请输入期望资源平均利用率。 期望值表示所选指标的期望值，通过向上取整（当前指标值 / 期望值 × 当前实例数）来计算目标实例数。 **说明** HPA在计算扩容、缩容实例数时，会选择最近5分钟内实例数的最大值。 • 容忍范围：指标处于范围内时不会触发伸缩，期望值必须在容忍范围之间。 当指标值大于缩容阈值且小于扩容阈值时，不会触发扩容或缩容。**阈值仅在1.15及以上版本的集群中支持**。
自定义策略（仅在1.15及以上版本的集群中支持）	**说明** 使用自定义策略时，集群中需要安装支持采集自定义指标的插件（例如Prometheus），且工作负载需正常上报并采集自定义指标。 采集自定义指标的方法及示例请参见**使用云原生监控插件监控自定义指标**。 • 自定义指标名称：自定义指标的名称，输入时可根据联想值进行选择。 • 指标来源：在下拉框中选择对象类型，可选择“Pod”。 • 期望值：Pod支持指标为平均值。通过向上取整（当前指标值 / 期望值 × 当前实例数）来计算需要伸缩的实例数。 **说明** HPA在计算扩容、缩容实例数时，会选择最近5分钟内实例数的最大值。 • 容忍范围：指标处于范围内时不会触发伸缩，期望值必须在容忍范围之间。

步骤4 设置完成后，单击“创建”。

----结束

10.2.3 创建使用自定义指标的 HPA 策略

Kubernetes默认的HPA策略只支持基于CPU和内存的自动伸缩，在复杂的业务场景中，仅使用CPU和内存使用率指标进行弹性伸缩往往无法满足日常运维需求。通过自定义指标配置工作负载HPA策略，可以根据业务自身特点，通过更多指标实现更灵活的弹性配置。

本文介绍如何部署示例Nginx应用，并通过Prometheus标准方式暴露container_cpu_usage_core_per_second的指标用来标识容器每秒使用CPU核心数。关于Prometheus指标的更多信息，请参见**metric_type**。

步骤一：安装云原生监控插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”。

步骤2 在“插件中心”页面右侧找到云原生监控插件，单击“安装”。

建议您关注以下配置，其他配置可按需进行设置。详情请参见**云原生监控插件**。

- 数据存储配置：必选**本地数据存储**，可选监控数据是否对接AOM或三方监控平台。
- 自定义指标采集：该配置在本实践中必须选择**开启**，否则将无法采集自定义指标。

步骤3 插件配置完成后，单击“安装”。

----结束

步骤二：创建示例工作负载

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在集群控制台左侧导航栏中选择“工作负载”，单击右上角“创建工作负载”。创建一个Nginx工作负载，详情请参见**创建无状态负载（Deployment）**。

----结束

步骤三：修改配置文件

步骤1 在集群控制台左侧导航栏中选择“配置与密钥”，切换至“monitoring”命名空间。

步骤2 更新user-adapter-config配置项，通过修改user-adapter-config中rules字段将Prometheus暴露出的指标转换为HPA可关联的指标。

添加以下示例规则：

```
rules:
  - seriesQuery: 'container_cpu_usage_seconds_total{namespace!="",pod!=""}'
    seriesFilters: []
    resources:
      overrides:
        namespace:
          resource: namespace
        pod:
          resource: pod
    name:
      matches: "^(.*)_seconds_total"
      as: "${1}_core_per_second"
    metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)'
```

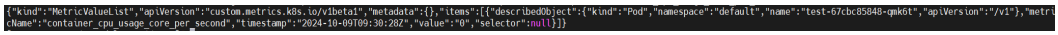
此配置项示例中，通过现有的container_cpu_usage_seconds_total指标，聚合成container_cpu_usage_core_per_second 指标，供后续的HPA策略中使用。关于采集规则配置详情请参见**Metrics Discovery and Presentation Configuration**。

- seriesQuery：PromQL请求数据（用户需要查询的指标，可根据实际情况填写）。
- metricsQuery：对seriesQuery中PromQL请求的数据进行聚合操作。
- resources：是PromQL里的数据Label，与resource进行匹配。此处的resource是指集群内的api-resource，例如Pod、Namespace和Node。您可以通过**kubectl api-resources -o wide**命令查看。此处Key对应Prometheus数据中的LabelName，请确认Prometheus指标数据中有此LabelName。
- name：指根据正则匹配把Prometheus指标名转为比较可读的指标名，此处将container_cpu_usage_seconds_total转为container_cpu_usage_core_per_second。

步骤3 重新部署monitoring命名空间下的custom-metrics-apiserver工作负载。

步骤4 执行命令查看指标是否添加成功。

```
kubectl get --raw  "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/*/
container_cpu_usage_core_per_second"
```



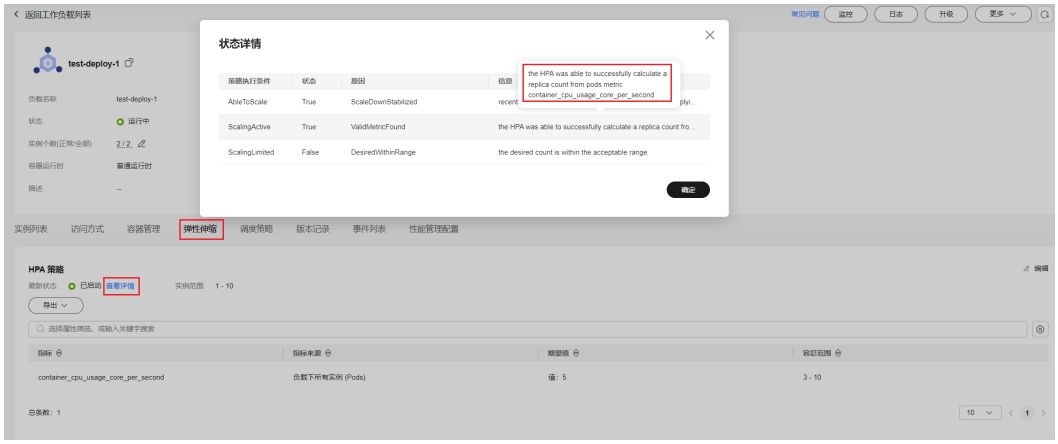
----结束

步骤四：测试 HPA 弹性功能

步骤1 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“更多 > 弹性伸缩”。

步骤2 策略类型选择“HPA+CronHPA策略”，并启用HPA策略， 可以直接选择配置在rules里面的自定义指标创建HPA策略。

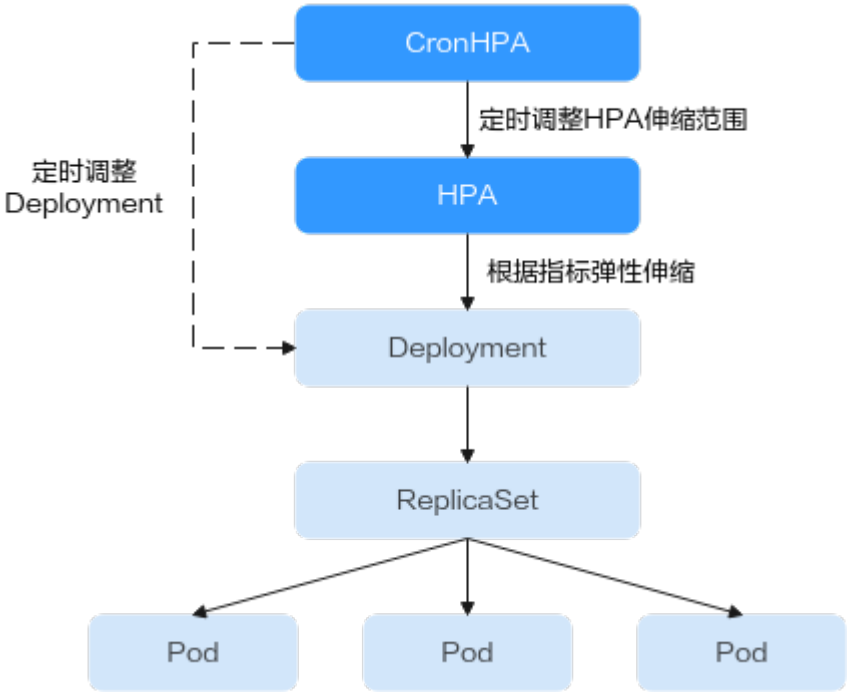
步骤3 单击工作负载名称，切换至“弹性伸缩”页签查看HPA状态，成功触发HPA策略。



----结束

10.2.4 创建 CronHPA 定时策略

在一些复杂的业务场景下，可能有固定时间段高峰业务，又有日常突发高峰业务。此种情况下，用户既期望能定时弹性伸缩应对固定时间段高峰业务，又期望能根据指标弹性伸缩应对日常突发高峰业务。CCE提供CronHPA的自定义资源，实现在固定时间段对集群进行扩缩容，并且可以和HPA策略共同作用，定时调整HPA伸缩范围，实现复杂场景下的工作负载伸缩。



CronHPA支持定时调整HPA策略的最大和最小实例数，也可以直接定时调整Deployment的Pod实例数。

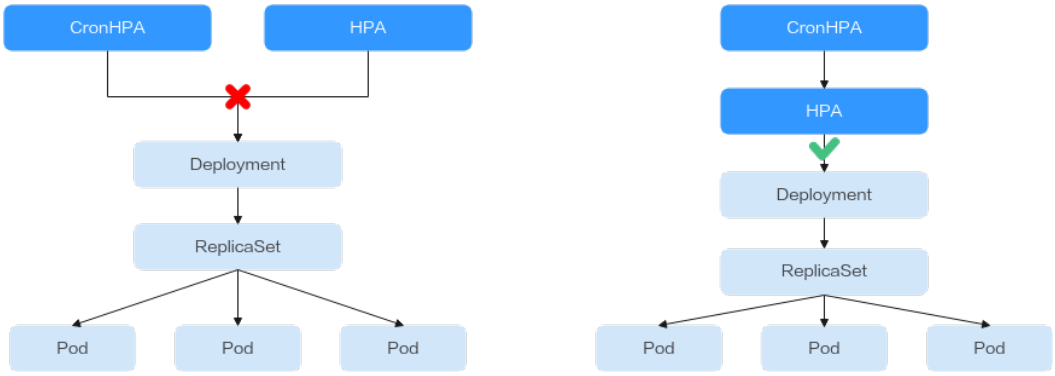
前提条件

已安装1.2.13及以上版本**CCE容器弹性引擎**。

使用 CronHPA 调整 HPA 伸缩范围

CronHPA支持定时调整HPA策略的最大和最小实例数，满足复杂场景下的工作负载伸缩。

由于HPA与CronHPA均通过scaleTargetRef字段来获取伸缩对象，如果CronHPA和HPA同时设置Deployment为伸缩对象，两个伸缩策略相互独立，后执行的操作会覆盖先执行的操作，导致伸缩效果不符合预期，因此需避免这种情况发生。



在CronHPA与HPA共同使用时，CronHPA规则是在HPA策略的基础上生效的，CronHPA不会直接调整Deployment的副本数目，而是通过HPA来操作Deployment，因此了解以下参数可帮助您更好地理解其工作原理。

- CronHPA的目标实例数（targetReplicas）：表示CronHPA设定的实例数，在CronHPA生效时用于调整HPA的最大/最小实例数，从而间接调整Deployment实例数。
- HPA的最小实例数（minReplicas）：Deployment的实例数下限。
- HPA的最大实例数（maxReplicas）：Deployment的实例数上限。
- Deployment的实例数（replicas）：CronHPA策略生效之前Deployment的Pod数量。

在CronHPA规则生效时，通过比较目标实例数（targetReplicas）与实际Deployment的实例数，并结合HPA的最小实例数或最大实例数的数值大小，来调整Deployment实例数的上下限值。

图 10-5 CronHPA 扩缩容场景

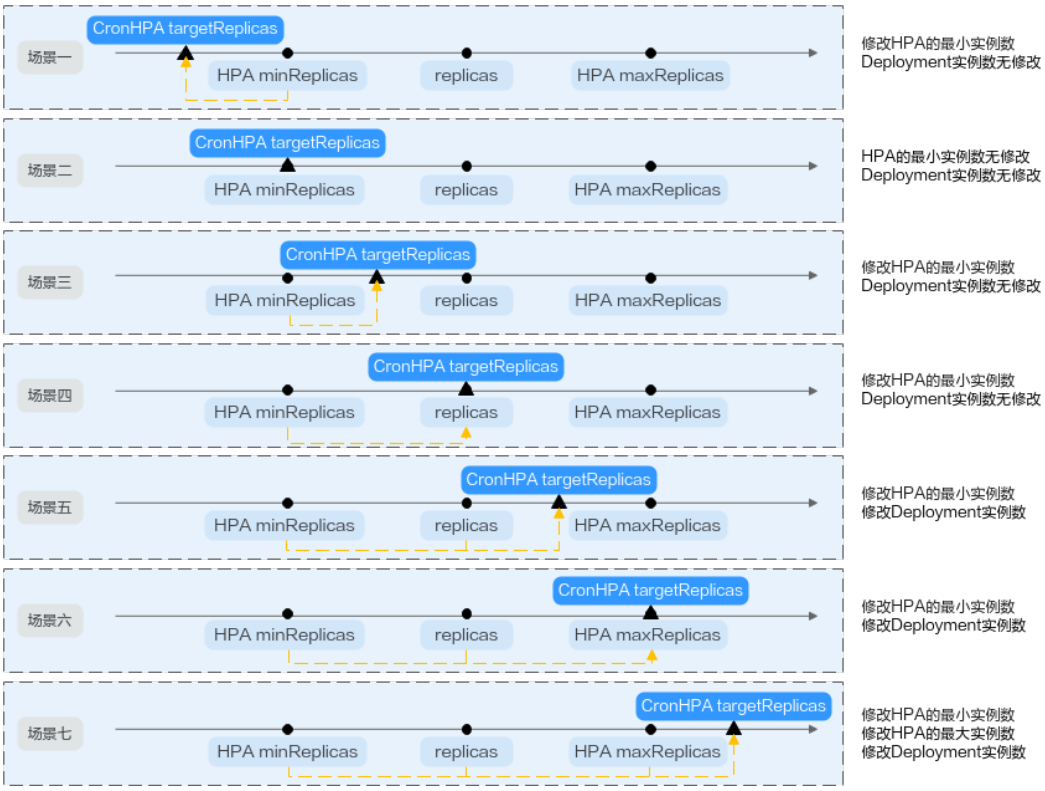


图10-5中为可能存在的扩缩容场景，如下表格以举例的形式说明了不同场景下CronHPA修改HPA的情况。

表 10-5 CronHPA 扩缩容场景

场景	场景说明	扩缩容条件			最终结果	操作说明
		CronHPA目标实例数（targetReplicas）	Deployment实例数（replicas）	HPA实例数上下限（minReplicas / maxReplicas）		
场景一	**targetReplicas** < minReplicas ≤ replicas ≤ maxReplicas	4	5	5/10	HPA: 4/10 Deployment: 5	CronHPA目标实例数低于HPA最小实例数（minReplicas）时： • 修改HPA的最小实例数。 • Deployment实例数无修改。

场景	场景说明	扩缩容条件			最终结果	操作说明
		CronHPA目标实例数（targetReplicas）	Deployment实例数（replicas）	HPA实例数上下限（minReplicas / maxReplicas）		
场景二	**targetReplicas** = minReplicas ≤ replicas ≤ maxReplicas	5	6	5/10	HPA: 5/10 Deployment: 6	CronHPA目标实例数等于HPA最小实例数（minReplicas）时： • HPA的最小实例数无修改。 • Deployment实例数无修改。
场景三	minReplicas < **targetReplicas** < replicas ≤ maxReplicas	4	5	1/10	HPA: 4/10 Deployment: 5	CronHPA目标实例数大于HPA最小实例数（minReplicas），小于Deployment实例数（replicas）时： • 修改HPA的最小实例数。 • Deployment实例数无修改。
场景四	minReplicas < **targetReplicas** = replicas < maxReplicas	5	5	1/10	HPA: 5/10 Deployment: 5	CronHPA目标实例数大于HPA最小实例数（minReplicas），等于Deployment实例数（replicas）时： • 修改HPA的最小实例数。 • Deployment实例数无修改。
场景五	minReplicas ≤ replicas < **targetReplicas** < maxReplicas	6	5	1/10	HPA: 6/10 Deployment: 6	CronHPA目标实例数大于Deployment实例数（replicas），小于HPA最大实例数（maxReplicas）时： • 修改HPA的最小实例数。 • 修改Deployment实例数。

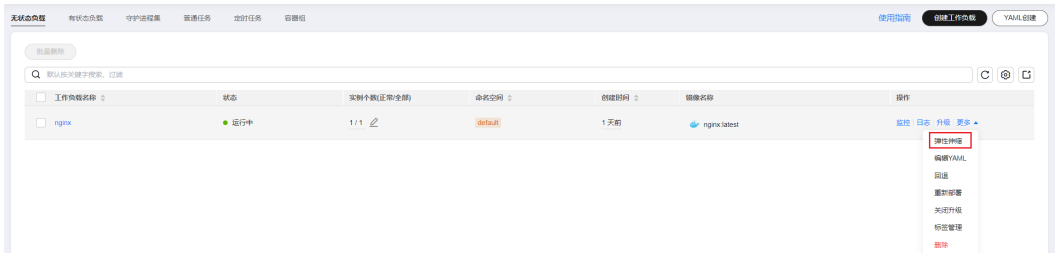
场景	场景说明	扩缩容条件			最终结果	操作说明
		CronHPA目标实例数（targetReplicas）	Deployment实例数（replicas）	HPA实例数上下限（minReplicas / maxReplicas）		
场景六	minReplicas ≤ replicas < **targetReplicas** = maxReplicas	10	5	1/10	HPA: 10/10 Deployment: 10	CronHPA目标实例数大于Deployment实例数（replicas），等于HPA最大实例数（maxReplicas）时： • 修改HPA的最小实例数。 • 修改Deployment实例数。
场景七	minReplicas ≤ replicas ≤ maxReplicas < **targetReplicas**	11	5	5/10	HPA: 11/11 Deployment: 11	CronHPA目标实例数大于HPA最大实例数（maxReplicas）时： • 修改HPA的最小实例数。 • 修改HPA的最大实例数。 • 修改Deployment实例数。

使用控制台创建

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“更多 > 弹性伸缩”。

图 10-6 工作负载弹性伸缩



步骤3 策略类型选择“HPA+CronHPA策略”，启用HPA策略，并同时启用CronHPA策略。此时CronHPA会定时调整HPA策略的最大和最小实例数。

步骤4 设置HPA策略，详情请参见**创建HPA策略**。

图 10-7 启用 HPA 策略

表 10-6 HPA 策略配置

参数	参数说明
实例范围	请输入最小实例数和最大实例数。 策略触发时，工作负载实例将在此范围内伸缩。
冷却时间	请输入缩容和扩容的冷却时间，单位为分钟，**缩容扩容冷却时间不能小于1分钟**。 **该设置仅在1.15到1.23版本的集群中显示。** 策略成功触发后，在此缩容/扩容冷却时间内，不会再次触发缩容/扩容，目的是等待伸缩动作完成后在系统稳定且集群正常的情况下进行下一次策略匹配。

参数	参数说明
伸缩配置	**该设置仅在1.25及以上版本的集群中显示。** ● 系统默认：采用社区推荐的默认行为进行负载伸缩，详情请参见**社区默认行为说明**。 ● 自定义：自定义扩/缩容配置的稳定窗口、步长、优先级等策略，实现更灵活的配置。未配置的参数将采用社区推荐的默认值。 – 禁止扩/缩容：选择是否禁止扩容或缩容。 – 稳定窗口：需要伸缩时，会在一段时间（设定的稳定窗口值）内持续检测，如在该时间段内始终需要进行伸缩（不满足设定的指标期望值）才进行伸缩，避免短时间的指标抖动造成异常。 – 步长策略：扩/缩容的步长，可设置一定时间内扩/缩容Pod数量或百分比。在存在多条策略时，可以选择使Pod数量最多或最少的策略。
系统策略	● 指标：可选择“CPU利用率”或“内存利用率”。 **说明** 利用率 = 工作负载所有Pod实际资源使用量的平均值 / 资源申请量（Request） ● 期望值：请输入期望资源平均利用率。 期望值表示所选指标的期望值，通过向上取整（当前指标值 / 期望值 × 当前实例数）来计算目标实例数。 **说明** HPA在计算扩容、缩容实例数时，会选择最近5分钟内实例数的最大值。 ● 容忍范围：指标处于范围内时不会触发伸缩，期望值必须在容忍范围之间。 当指标值大于缩容阈值且小于扩容阈值时，不会触发扩容或缩容。**阈值仅在1.15及以上版本的集群中支持**。
自定义策略（仅在1.15及以上版本的集群中支持）	**说明** 使用自定义策略时，集群中需要安装支持采集自定义指标的插件（例如Prometheus），且工作负载需正常上报并采集自定义指标。 采集自定义指标的方法及示例请参见**使用云原生监控插件监控自定义指标**。 ● 自定义指标名称：自定义指标的名称，输入时可根据联想值进行选择。 ● 指标来源：在下拉框中选择对象类型，可选择“Pod”。 ● 期望值：Pod支持指标为平均值。通过向上取整（当前指标值 / 期望值 × 当前实例数）来计算需要伸缩的实例数。 **说明** HPA在计算扩容、缩容实例数时，会选择最近5分钟内实例数的最大值。 ● 容忍范围：指标处于范围内时不会触发伸缩，期望值必须在容忍范围之间。

步骤5 在CronHPA的策略规则中单击+，在弹出的窗口中设置伸缩策略参数。

图 10-8 启用 CronHPA 策略

表 10-7 CronHPA 策略参数配置

参数	参数说明
目标实例数	策略触发时，将根据实际情况调整HPA策略实例数范围，详情请参见表10-5。
触发时间	可选择每天、每周、每月或每年的具体时间点。 **说明** 触发时间基于节点所在时区进行计算。
是否启用	可选择启用或关闭该策略规则。

步骤6 填写完成上述参数，单击“确定”，您可以在列表中查看添加的策略规则。重复以上步骤，您可以添加多条策略规则，但策略的触发时间不能相同。

步骤7 设置完成后，单击“创建”。

----结束

使用kubectl命令行创建

当CronHPA与HPA兼容使用时，需要将CronHPA中的scaleTargetRef字段设置为HPA策略，而HPA策略的scaleTargetRef字段设置为Deployment，这样CronHPA策略会在固定的时间调整HPA策略的实例数量上下限，即可实现工作负载定时伸缩和弹性伸缩的兼容。

步骤1 为Deployment创建HPA策略。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-test
  namespace: default
spec:
  maxReplicas: 10            # 最大实例数
  minReplicas: 5             # 最小实例数
  scaleTargetRef:            # 关联Deployment
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  targetCPUUtilizationPercentage: 50
```

步骤2 创建CronHPA策略，并关联步骤1中创建的HPA策略。

```
apiVersion: autoscaling.cce.io/v2alpha1
kind: CronHorizontalPodAutoscaler
metadata:
  name: ccetest
```



```
  namespace: default
spec:
  scaleTargetRef:                # 关联HPA策略
    apiVersion: autoscaling/v1
    kind: HorizontalPodAutoscaler
    name: hpa-test
  rules:
  - ruleName: "scale-down"
    schedule: "15 * * * *"       # 指定任务运行时间与周期，参数格式请参见Cron，例如0 * * * * 或@hourly。
    targetReplicas: 1            # 目标实例数量
    disable: false
  - ruleName: "scale-up"
    schedule: "13 * * * *"
    targetReplicas: 11
    disable: false
```

表 10-8 CronHPA 关键字段说明

字段	说明
apiVersion	API版本，固定值“autoscaling.cce.io/v2alpha1”。
kind	API类型，固定值“CronHorizontalPodAutoscaler”。
metadata.name	CronHPA策略名称。
metadata.namespace	CronHPA策略所在的命名空间。
spec.scaleTargetRef	指定CronHPA的扩缩容对象，可配置以下字段： • apiVersion：CronHPA扩缩容对象的API版本。 • kind：CronHPA扩缩容对象的API类型。 • name：CronHPA扩缩容对象的名称。 CronHPA支持HPA策略或Deployment，具体用法请参见**使用CronHPA调整HPA伸缩范围**或**使用CronHPA直接调整Deployment实例数量**。
spec.rules	CronHPA策略规则，可添加多个规则。每个规则可配置以下字段： • ruleName：CronHPA规则名称，该名称需唯一。 • schedule：指定任务运行时间与周期，参数格式与CronTab类似，请参见**Cron**，例如0 * * * * 或@hourly。 **说明** 触发时间基于节点所在时区进行计算。 • targetReplicas：扩缩容的Pod数目。 • disable：参数值为“true”或“false”。其中“false”表示该规则生效，“true”则表示该规则不生效。

----结束

使用 CronHPA 直接调整 Deployment 实例数量

CronHPA还可以单独调整关联Deployment，定时调整Deployment的实例数，使用方法如下。

使用控制台创建

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“更多 > 弹性伸缩”。

图 10-9 工作负载弹性伸缩

步骤3 策略类型选择“HPA+CronHPA策略”，选择不启用HPA策略，并选择启用CronHPA策略。

此时CronHPA会直接定时调整工作负载的实例数。

步骤4 在CronHPA的策略规则中单击+，在弹出的窗口中设置伸缩策略参数。

图 10-10 使用 CronHPA 调整工作负载实例数

表 10-9 CronHPA 策略参数配置

参数	参数说明
目标实例数	策略触发时，工作负载实例将调整至该数值。
触发时间	可选择每天、每周、每月或每年的具体时间点。 **说明** 触发时间基于节点所在时区进行计算。
是否启用	可选择启用或关闭该策略规则。

步骤5 填写完成上述参数，单击“确定”，您可以在列表中查看添加的策略规则。重复以上步骤，您可以添加多条策略规则，但策略的触发时间不能相同。

步骤6 设置完成后，单击“创建”。

----结束

使用kubectl命令行创建

```
apiVersion: autoscaling.cce.io/v2alpha1
kind: CronHorizontalPodAutoscaler
metadata:
  name: ccetest
  namespace: default
spec:
  scaleTargetRef:           # 关联Deployment
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  rules:
  - ruleName: "scale-down"
    schedule: "08 * * * *"    # 指定任务运行时间与周期，参数格式请参见Cron，例如0 * * * * 或@hourly。
    targetReplicas: 1
    disable: false
  - ruleName: "scale-up"
    schedule: "05 * * * *"
    targetReplicas: 3
    disable: false
```

10.2.5 创建 CustomedHPA 策略

CustomedHPA策略是自研的弹性伸缩增强能力，能够基于指标（CPU利用率、内存利用率）或周期（每天、每周、每月或每年的具体时间点），对无状态工作负载进行弹性扩缩容。

主要功能如下：

- 支持按照当前实例数的百分比进行扩缩容。
- 支持设置一次扩缩容的最小步长。
- 支持按照实际指标值执行不同的扩缩容动作。

前提条件

使用CustomedHPA策略必须安装**CCE容器弹性引擎**，若该插件版本低于1.2.11，则必须安装**prometheus**插件；若插件版本大于或等于1.2.11，则需要安装能够提供Metrics API的插件，您可根据集群版本和实际需求选择其中之一：

- **Kubernetes Metrics Server**：提供基础资源使用指标，例如容器CPU和内存使用率。所有集群版本均可安装。
- **云原生监控插件**：该插件支持v1.17及以后的集群版本。
  - 根据基础资源指标进行弹性伸缩：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供基础资源指标**。
  - 根据自定义指标进行弹性伸缩：需要将自定义指标聚合到Kubernetes API Server，详情请参见**使用自定义指标创建HPA策略**。
- **Prometheus（停止维护）**：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供资源指标**。该插件仅支持v1.21及之前的集群版本。

约束与限制

- CustomedHPA策略仅支持1.15及以上版本的集群。
- 1.19.10以下版本的集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，当新Pod被调度到另一个节点时，会导致之前Pod不能正常读写。

  1.19.10及以上版本集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，新Pod会因为无法挂载云硬盘导致无法成功启动。
- CCE容器弹性引擎插件的资源使用量主要受集群中总容器数量和伸缩策略数量影响，通常场景下建议每5000容器配置CPU 500m, 内存1000Mi资源，每1000伸缩策略CPU 100m，内存500Mi。
- 创建CustomedHPA策略后，不支持将已关联的工作负载修改为其他工作负载。

创建 CustomedHPA 策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“工作负载”，在目标工作负载的操作列中单击“弹性伸缩”。

步骤3 策略类型选择“CustomedHPA策略”，并填写策略参数。

表 10-10 CustomedHPA 策略参数配置

参数	参数说明
实例范围	请输入最小实例数和最大实例数。 策略触发时，工作负载实例将在此范围内伸缩。
冷却时间	请输入冷却时间值，单位为分钟。 策略成功触发后，在此冷却时间内，不会再次触发缩容/扩容，目的是等待伸缩动作完成后在系统稳定且集群正常的情况下进行下一次策略匹配。 **说明** CCE容器弹性引擎插件为1.3.10及以上版本时，冷却时间仅对指标类策略生效，周期类策略不受冷却时间影响。
策略规则	单击 ┼ 在弹出的窗口中设置伸缩策略参数： • 类型：可选择“指标触发”（参见**表10-11**）或“周期触发”（参见**表10-12**）。选择类型后，可设置不同的触发条件及动作。 • 是否启用：可选择启用或关闭该策略规则。 填写完成上述参数，单击“确定”，您可以在列表中查看添加的策略规则。

表 10-11 指标触发类型规则

参数	参数说明
触发条件	请选择“CPU利用率”或“内存利用率”，选择“>”或“<”，并输入百分比的值。 **说明** 利用率 = 工作负载所有Pod实际资源使用量的平均值 / 资源申请量（Request）
执行动作	与上述“触发条件”相对应，达到触发条件值后所要执行的动作，可添加多个执行动作。 • 伸缩至：将实例数调整至设定的目标值，支持填写实例数或百分比。该动作支持扩容或缩容实例数，如果当前实例数小于目标值（或百分比大于100%），会将实例数扩容至目标值；如果当前实例数大于目标值（或百分比小于100%），则会将实例数缩容至目标值。 • 增加：当“触发条件”选择“>”时设置。在当前实例数的基础上增加指定的实例数，支持填写实例数或百分比。该动作仅支持扩容实例数。 • 减少：当“触发条件”选择“<”时设置。在当前实例数的基础上减少指定的实例数，支持填写实例数或百分比。该动作仅支持缩容实例数。 **说明** 以上执行动作均支持填写具体实例数或百分比。 填写百分比时，还需填写至少存在的实例数。此时默认按以下公式计算最终实例个数：（当前实例数×百分比，然后向上取整）。若计算结果小于设置的最少实例数，以设置值为准；否则，以计算结果为准。 如下图中所示，当CPU利用率超过50%时将伸缩至5个实例，当超过70%时伸缩至8个实例，当超过90%时在8个实例基础上再增加10个实例。反之，按此规则执行缩容。 **图 10-11** 触发条件 

表 10-12 周期触发类型规则

参数	参数说明
触发时间	可选择每天、每周、每月或每年的具体时间点。
执行动作	与上述“触发时间”相对应，达到触发时间值后所要执行的动作。如下图中所示，即每天17:00时将执行增加1个实例的动作。 • 伸缩至：将实例数调整至设定的目标值，支持填写实例数或伸缩百分比。该动作支持扩容或缩容实例数，如果当前实例数小于目标值（或百分比大于100%），会将实例数扩容至目标值；如果当前实例数大于目标值（或百分比小于100%），则会将实例数缩容至目标值。 • 增加：在当前实例数的基础上增加指定的实例数，支持填写实例数或百分比。该动作仅支持扩容实例数。 • 减少：在当前实例数的基础上减少指定的实例数，支持填写实例数或百分比。该动作仅支持缩容实例数。 **说明** 以上执行动作均支持填写具体实例数或百分比。 填写百分比时，还需填写至少存在的实例数。此时默认按以下公式计算最终实例个数：（当前实例数×百分比，然后向上取整）。若计算结果小于设置的最少实例数，以设置值为准；否则，以计算结果为准。 **图 10-12** 周期触发-每天 

步骤4 设置完成后，单击“创建”。

----结束

使用 kubectl 创建

CustomHPA是一种CRD资源，可按如下YAML定义。

```
apiVersion: autoscaling.cce.io/v1alpha1
kind: CustomedHorizontalPodAutoscaler
metadata:
  name: customhpa-example
```



```
  namespace: default
spec:
  coolDownTime: 3m             # 冷却时间
  maxReplicas: 10              # 最大实例数
  minReplicas: 1               # 最小实例数
  rules:
    - actions:                      # 策略规则
        - metricRange: 0,0.1          # 指标范围，表示从0到10%
          operationType: ScaleDown      # 伸缩类型，ScaleDown表示减少
          operationUnit: Task           # 操作单位，Task表示个数
          operationValue: 1             # 每次伸缩的数量
        - metricRange: 0.1,0.3          # 指标范围，表示从10%到30%
          operationType: ScaleDown
          operationUnit: Task
          operationValue: 2
      disable: false
      metricTrigger:
        hitThreshold: 1
        metricName: CPURatioToRequest   # 指标名称，CPURatioToRequest为CPU利用率
        metricOperation: <              # 指标表达式操作符
        metricValue: 0.3                # 指标表达式右侧取值
        periodSeconds: 60               #
        statistic: instantaneous        #
      ruleName: low
      ruleType: Metric
    - actions:
        - metricRange: 0.7,0.9
          operationType: ScaleUp
          operationUnit: Task
          operationValue: 1
        - metricRange: 0.9,+Infinity
          operationType: ScaleUp
          operationUnit: Task
          operationValue: 2
      disable: false
      metricTrigger:
        hitThreshold: 1
        metricName: CPURatioToRequest
        metricOperation: '>'
        metricValue: 0.7
        periodSeconds: 60
        statistic: instantaneous
      ruleName: high
      ruleType: Metric
  scaleTargetRef:                # 关联负载
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
```

10.2.6 创建 VPA 策略

VPA策略即Vertical Pod Autoscaling，该功能可以在Kubernetes中实现Pod垂直弹性伸缩，可以根据容器资源历史使用情况自动调整Pod的CPU、Memory资源申请量。当业务负载急剧飙升时，VPA能够快速地在设定范围内扩大容器的资源申请值（Requests），以满足业务需求。而在业务负载变小时，VPA会根据实际情况适当缩小资源申请量，以节省计算资源。此外，VPA还能推荐更合理的资源申请量，在确保容器有足够的资源供使用的前提下，提升容器的资源利用率。

功能概述

VPA以容器为单位对资源指标进行聚合计算，根据容器的资源实际使用情况动态调整容器的资源申请值（Requests），同时保证调整前和调整后资源限制值（Limits）与资源申请值（Requests）的比值不变。目前支持CPU与Memory两类资源的垂直伸缩。

详细功能说明如下：

- VPA计算CPU与Memory建议值时需要数依赖Metrics API采集的数据。
- VPA在计算资源建议值时，Memory资源的单Pod最小理论建议值250Mi，Pod内单容器的最小理论建议值为250Mi/Pod容器数目。CPU资源的单Pod最小理论建议值为25m，Pod内单容器的最小理论建议值为25m/Pod容器数目。

  您可在创建VPA任务时，通过配置containerPolicies字段为容器配置弹性资源上下限。
- 如果容器初始时同时配置了资源申请值与限制值，VPA计算后给出的建议值会修改该容器的资源申请值，而限制值则根据容器初始创建时申请值与限制值的比例进行计算。

  例如，某个容器原来配置了CPU资源申请值为100m与限制值为200m，申请值与限制值的比例为1:2。如果VPA计算后的资源申请值建议为80m，则该容器最终的CPU资源申请值为80m，限制值为160m。
- VPA会尽量让建议值符合其他资源限制要求。但如果VPA建议值与资源限制出现冲突，VPA建议值不会根据资源限制进行调整，可能导致VPA配置值超出其他资源限制要求。

  例如，某一个命名空间的内存申请值不能超过2GiB，而VPA的建议值如果比较大，可能导致Pod更新后整个命名空间的资源申请量超过2GiB从而出现无法调度。

前提条件

- 集群版本需满足v1.25及以上。
- 使用VPA需要在集群中安装能够提供Metrics API的插件，您可根据实际需求选择其中之一：
  - **Kubernetes Metrics Server**：提供基础资源使用指标，例如容器CPU和内存使用率。
  - **云原生监控插件**：使用Prometheus提供基础资源使用指标，需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供基础资源指标**。
- 集群中需要安装**容器垂直弹性引擎**。

注意事项

须知

容器垂直伸缩功能目前处于试验阶段，请谨慎使用。

- VPA对Pod资源进行动态更新时，会导致Pod的重建，重建的Pod可能会调度到一个新的节点上，且VPA无法保证重建的Pod调度成功。
- 只有由副本控制管理器（例如Deployment、StatefulSet等）管理的Pod才会进行资源动态更新，独立运行的Pod不支持资源动态更新。
- 目前VPA不能和监控CPU和内存度量的Horizontal Pod Autoscaler（HPA）同时运行。
- VPA admission webhook会对Pod的配置进行更新，如果集群中有其他的admission webhook，需要确保它们不会与VPA发生冲突。
- VPA会处理大部分的OOM（Out Of Memory）事件，但无法保证处理所有的OOM事件。

- VPA的性能尚未在大规模集群中实践。
- VPA建议值可能大于实际可分配的资源量（例如节点可分配资源上限、资源配额上限），导致重建的Pod处于Pending状态无法调度。
- 为同一个负载的配置多个VPA可能会出现行为不一致的现象。

创建 VPA 策略

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 部署一个示例工作负载。如果已有工作负载可忽略本步骤。

```
kubectl create -f hamster.yaml
```

hamster.yaml文件内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hamster
spec:
  selector:
    matchLabels:
      app: hamster
  replicas: 2
  template:
    metadata:
      labels:
        app: hamster
    spec:
      containers:
        - name: hamster
          image: registry.k8s.io/ubuntu-slim:0.1
          resources:
            requests:
              cpu: 100m
              memory: 50Mi
          command: ["/bin/sh"]
          args:
            - "-c"
            - "while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done"
```

步骤3 创建VPA任务。

```
kubectl create -f hamster-vpa.yaml
```

hamster-vpa.yaml文件内容如下：

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: hamster-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: hamster
  updatePolicy:
    updateMode: "Off"
  resourcePolicy:
    containerPolicies:
      - containerName: '*'
        minAllowed:
          cpu: 100m
          memory: 50Mi
        maxAllowed:
          cpu: 1
          memory: 500Mi
        controlledResources: ["cpu", "memory"]
```


表 10-13 VPA 关键字段说明

字段	是否必填	说明
spec.targetRef	是	指定VPA负载对象。 支持Deployment、Statefulset、Damonset等负载类型。
spec.updatePolicy.updateMode	否	VPA建议值动态更新策略，默认值为“Auto”。 可选配置如下： ● Off：仅生成建议值，不更新Pod资源申请量。 ● Recreate：生成建议值，并自动更新Pod资源申请量。 ● Initial：生成建议值，仅在Pod新建时更新资源申请量，不动态更新正在运行的Pod的资源申请量。 ● Auto：与Recreate配置策略行为一致。
spec.resourcePolicy.containerPolicies	否	为不同的容器指定的VPA策略、VPA资源上下限。详细参数说明请参见**表10-14**。

表 10-14 containerPolicy 关键字段说明

字段	是否必填	说明
containerName	是	容器名称。
minAllowed	否	指定容器VPA资源下限，即VPA建议值不能低于该值。 可选资源类型： ● cpu ● memory
maxAllowed	否	指定容器VPA资源上限，即VPA建议值不能高于该值。 可选资源类型： ● cpu ● memory
controlledResources	否	指定容器VPA资源类型，默认值为["cpu", "memory"]。 可选资源类型： ● cpu ● memory

字段	是否必填	说明
mode	否	该容器的VPA策略是否生效，默认值为“Auto”。 可配置值： ● Auto：打开该容器的VPA策略。 ● Off：关闭该容器的VPA策略。

步骤4 等待VPA生成资源期望值，执行以下命令查看VPA资源详情。

```
kubectl get vpa hamster-vpa -oyaml
```

回显如下：

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: hamster-vpa
  namespace: default
spec:
  resourcePolicy:
    containerPolicies:
    - containerName: '*'
      controlledResources:
      - cpu
      - memory
      maxAllowed:
        cpu: 1
        memory: 500Mi
      minAllowed:
        cpu: 100m
        memory: 50Mi
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hamster
  updatePolicy:
    updateMode: "Off"
status:
  conditions:
  - lastTransitionTime: "2024-06-27T07:37:01Z"
    status: "True"
    type: RecommendationProvided
  recommendation:
    containerRecommendations:
    - containerName: hamster
      lowerBound:
        cpu: 475m
        memory: 262144k
      target:
        cpu: 587m
        memory: 262144k
      uncappedTarget:
        cpu: 587m
        memory: 262144k
      upperBound:
        cpu: 673m
        memory: 262144k
```

其中status.recommendation字段为VPA给出的资源配置建议值。

如果updateMode配置为“Auto”，该值会动态更新到正在运行的Pod资源申请配置上，将会**导致Pod重建**。

表 10-15 containerRecommendation 关键字段说明

字段	说明
containerName	VPA策略生效的容器名称。
target	VPA建议值，该值是结合了containerPolicy字段配置的资源上下限后的计算结果。 VPA使用该值弹性配置Pod资源申请量。
lowerBound	VPA下限建议值。
upperBound	VPA上限建议值。
uncappedTarget	实际计算的VPA建议值，该值是未结合containerPolicy字段配置的资源上下限的计算结果。

----结束

10.2.7 创建 AHPA 策略

Kubernetes原生HPA由于是被动触发，在实际应用中存在弹性滞后的问题。AHPA策略即Advanced Horizontal Pod Autoscaling，可根据业务历史指标，识别工作负载弹性周期并对未来波动进行预测，提前进行扩缩容动作，解决原生HPA的滞后问题。

功能介绍

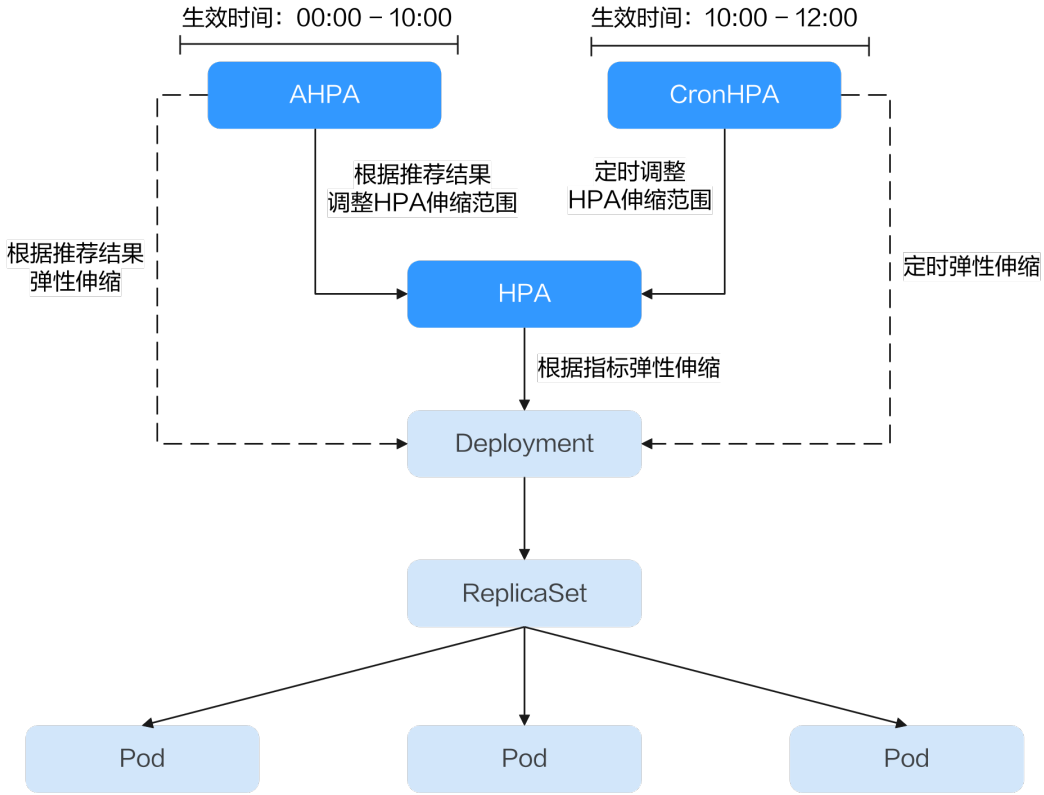
AHPA通过对工作负载的历史指标进行监控，以周为维度进行建模，因此对具有明显周期性的工作负载具有更佳效果。

AHPA启动后拉取指定的工作负载过去一定时间的监控数据（至少一周，至多八周），利用统计学原理分析建模。随后每分钟一次，根据当前时间点的历史监控数据，结合未来一段时间窗口的历史数据，给出当前时间点工作负载的推荐副本数，提前准备Pod应对即将到来的业务量上涨，保障资源供给。

AHPA可与HPA策略以及CronHPA策略共同使用，实现复杂场景下的工作负载伸缩。

AHPA支持根据推荐结果调整HPA策略的最大和最小实例数，或者直接调整Deployment工作负载的副本数。

AHPA调整HPA策略最大和最小实例数的逻辑与CronHPA相同，可参考**使用CronHPA调整HPA伸缩范围**。



前提条件

- 集群中已安装1.5.2及以上版本的**CCE容器弹性引擎**。
- 集群中已安装云原生监控插件，且开启**监控数据上报至AOM服务**，详情请参见**云原生监控插件**。

约束与限制

- AHPA策略仅支持1.23及以上版本的集群。
- 1.19.10及以上版本集群中，如果使用HPA策略对挂载了EVS卷的负载进行扩容，新Pod会因为无法挂载云硬盘导致无法成功启动。
- CCE容器弹性引擎插件的资源使用量主要受集群中总容器数量和伸缩策略数量影响，通常场景下建议每5000容器配置CPU 500m, 内存1000Mi资源，每1000伸缩策略CPU 100m，内存500Mi。
- AHPA需要对工作负载历史数据进行分析处理，需要额外内存，通常场景下建议每100个AHPA策略配置CPU 100m、内存 300Mi。
- 创建AHPA策略后，不支持将已关联的工作负载修改为其他工作负载。
- AHPA策略不支持和CustomedHPA策略同时启用。

使用 AHPA 策略

步骤1 使用kubectl连接集群，详情请参见**通过kubectl连接集群**。

步骤2 部署一个示例工作负载。如果已有工作负载可忽略本步骤。推荐使用已收集超过7天以上监控数据的工作负载，AHPA生效需要7天或更久的监控数据。

```
kubectl create -f hamster.yaml
```


hamster.yaml文件内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hamster
spec:
  selector:
    matchLabels:
      app: hamster
  replicas: 2
  template:
    metadata:
      labels:
        app: hamster
    spec:
      containers:
        - name: hamster
          image: registry.k8s.io/ubuntu-slim:0.1
          resources:
            requests:
              cpu: 100m
              memory: 50Mi
          command: ["/bin/sh"]
          args:
            - "-c"
            - "while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done"
```

步骤3 创建AHPA任务。

```
kubectl create -f hamster-ahpa.yaml
```

hamster-vpa.yaml文件内容如下：

```
apiVersion: autoscaling.cce.io/v1alpha1
kind: AdvancedHorizontalPodAutoscaler
metadata:
  name: hamster-ahpa
  namespace: default
spec:
  scaleTargetRef:  # 关联负载，当前支持 Deployment/HPA
    apiVersion: apps/v1
    kind: Deployment
    name: hamster
  minReplicas: 2  # 最小实例数
  maxReplicas: 10  # 最大实例数
  metrics:  # 指标列表，格式和社区HPA一致
  - type: Resource  # 指标源种类，当前只支持 Resource
    resource:
      name: cpu  # 指标源名称，当前只支持 cpu/memory
      target:
        type: Utilization  # 指标源类型，当前只支持 Utilization
        averageUtilization: 50
  predictConfig:
    predictWindowSeconds: 1800
    stabilizationWindowSeconds: 1800
    quantile: "0.97"
  effectiveTime:
  - '* * 11-22 ? * MON-FRI'  # 每周一到周五的11:00 - 22:00 生效
```

表 10-16 AHPA 关键字段说明

字段	是否必填	说明
scaleTargetRef	是	指定目标Deployment/HPA。

字段	是否必填	说明
metrics	是	用于配置弹性Metrics，当前支持CPU、Memory两种指标。当前仅支持配置一种metric，不支持CPU和Memory同时配置。
maxReplicas	是	最大扩容实例数，取值范围为0~2147483647。
minReplicas	是	最小缩容实例数，取值范围为0~2147483647。
predictConfig.predictWindowSeconds	是	推荐窗口时间，由当前时间点开始，在窗口范围内的指标历史值将参与推荐副本数计算，取值范围为1~3600。
predictConfig.stabilizationWindowSeconds	否	缩容冷却时间，取值范围为0~3600。
predictConfig.quantile	是	预测分位数，业务指标实际值低于设定目标值的概率，越大表示越保守。取值范围为0~1，支持两位小数，默认值为0.99。推荐取值范围为0.90~0.99。
effectiveTime	否	指定多个cron表达式，AHPA将在cron表达式的并集生效。默认总是生效。

步骤4 待新建或已存在的工作负载至少收集7日以上监控数据到AOM中，AHPA即可建模成功并给出副本数推荐，等待AHPA生成副本推荐数，执行以下命令查看AHPA资源详情。

```
kubectl get ahpa hamster-ahpa -oyaml
```

回显如下：

```
apiVersion: autoscaling.cce.io/v1alpha1
kind: AdvancedHorizontalPodAutoscaler
metadata:
  creationTimestamp: "2024-10-07T13:11:58Z"
  generation: 2
  name: hamster-ahpa
  namespace: default
  resourceVersion: "15529454"
  uid: e5ffbb01-50b0-4485-8cf5-bc2be884b1ee
spec:
  effectiveTime:
  - '* * 11-22 ? * MON-FRI'
  maxReplicas: 10
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 50
        type: Utilization
    type: Resource
  minReplicas: 2
  predictConfig:
    predictWindowSeconds: 1800
    quantile: "0.97"
    stabilizationWindowSeconds: 1800
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hamster
status:
```



```
conditions:
- lastTransitionTime: "2024-10-07T13:24:19Z"
  message: the AHPA's model is ready
  reason: ModelIsReady
  status: "True"
  type: ModelAvailable
- lastTransitionTime: "2024-10-07T13:24:19Z"
  message: the AHPA was able to successfully calculate a replica count
  reason: SucceededRunPrediction
  status: "True"
  type: ScalingActive
- lastTransitionTime: "2024-10-07T13:24:19Z"
  message: ths ahpa checkpoint is fresh
  reason: CheckpointIsFresh
  status: "True"
  type: CheckpointAvailable
- lastTransitionTime: "2024-10-07T13:24:19Z"
  message: recommended size matches current size
  reason: ReadyForNewScale
  status: "True"
  type: AbleToScale
- lastTransitionTime: "2024-10-07T13:24:19Z"
  message: the desired replica count is more than the maximum replica count
  reason: TooManyReplicas
  status: "True"
  type: ScalingLimited
currentReplicas: 10
desiredReplicas: 10
lastScaleTime: "2024-10-07T13:24:19Z"
```

----结束

10.2.8 管理工作负载弹性伸缩策略

操作场景

工作负载弹性策略创建完成后，可对创建的策略进行更新、编辑YAML以及删除等操作。

操作步骤

您可以查看工作负载弹性策略的规则、最新状态和事件，参照界面中的报错提示有针对性的解决异常事件。

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中单击“策略”，在“弹性伸缩策略”页签下，根据弹性伸缩策略类型选择HPA / CronHPA / CustomedHPA的页签。

步骤3 您可以查看弹性伸缩策略的最新状态、规则、关联工作负载等信息。

说明

您还可以在工作负载详情页中查看已创建的弹性伸缩策略：

1. 登录CCE控制台，单击集群名称进入集群。
2. 在左侧导航栏中单击“工作负载”，单击工作负载名称查看详情。
3. 在该工作负载详情页的“弹性伸缩”页签下可以看到弹性伸缩策略，您在“策略”页面配置的伸缩策略也会在这里显示。

步骤4 您可以在操作列中单击对应的按钮对弹性伸缩策略进行管理。

弹性伸缩策略类型	操作
HPA策略	• 事件：查看HPA策略事件页签，若策略异常，请参照界面中的报错提示进行定位处理。 • 编辑YAML：在弹出的“编辑YAML”窗口中，您可以对YAML进行修改、复制和下载。 • 编辑：在打开的“编辑HPA策略”页面中，参考表10-4更新策略参数。 • 克隆：根据已有策略创建一个配置相同的弹性伸缩策略，您可以根据需求对参数进行调整。 • 删除：在弹出的窗口中，单击“是”完成删除操作。
CronHPA策略	• 查看YAML：在弹出的“查看YAML”窗口中，您可以对YAML进行复制和下载，不能对其修改。 • 删除：在弹出的窗口中，单击“是”完成删除操作。
CustomedHPA策略	• 编辑：在打开的“编辑HPA策略”页面中，参考表10-10更新策略参数。 • 克隆：根据已有策略创建一个配置相同的弹性伸缩策略，您可以根据需求对参数进行调整。 • 查看YAML：在弹出的“查看YAML”窗口中，您可以对YAML进行复制和下载，不能对其修改。 • 删除：在弹出的窗口中，单击“是”完成删除操作。

----结束

10.3 节点弹性伸缩

10.3.1 节点伸缩原理

HPA是针对Pod级别的，可以根据负载指标动态调整副本数量，但是如果集群的资源不足，新的副本无法运行的情况下，就只能对集群进行扩容。

CCE集群弹性引擎是Kubernetes提供的集群节点弹性伸缩组件，根据Pod调度状态及资源使用情况对集群的节点进行自动扩容缩容，同时支持多可用区、多实例规格、指标触发和周期触发等多种伸缩模式，满足不同的节点伸缩场景。

前提条件

使用节点伸缩功能前，需要安装**CCE集群弹性引擎**插件，插件版本要求1.13.8及以上。

Cluster Autoscaler 工作原理

Cluster Autoscaler主要流程包括两部分：

- 扩容流程： Autoscaler会每隔10s检查一次所有未调度的Pod，根据用户设置的策略，选择出一个符合要求的节点池进行扩容。

> **说明**
>
> Autoscaler检测未调度Pod进行扩容时，使用的是与Kubernetes社区版本一致的调度算法进行模拟调度计算，若应用调度采用非内置kube-scheduler调度器或其他非Kubernetes社区调度策略，此类应用使用Autoscaler扩容时可能因调度算法不一致出现无法扩容或多扩风险。

- 缩容流程：Autoscaler每隔10s会扫描一次所有的Node，如果该Node上所有的Pod Requests少于用户定义的缩容百分比时，Autoscaler会模拟将该节点上的Pod是否能迁移到其他节点。

  当集群节点处于一段时间空闲状态时（默认10min），会触发集群缩容操作（即节点会被自动删除）。当节点存在以下几种状态的Pod时，不可缩容：

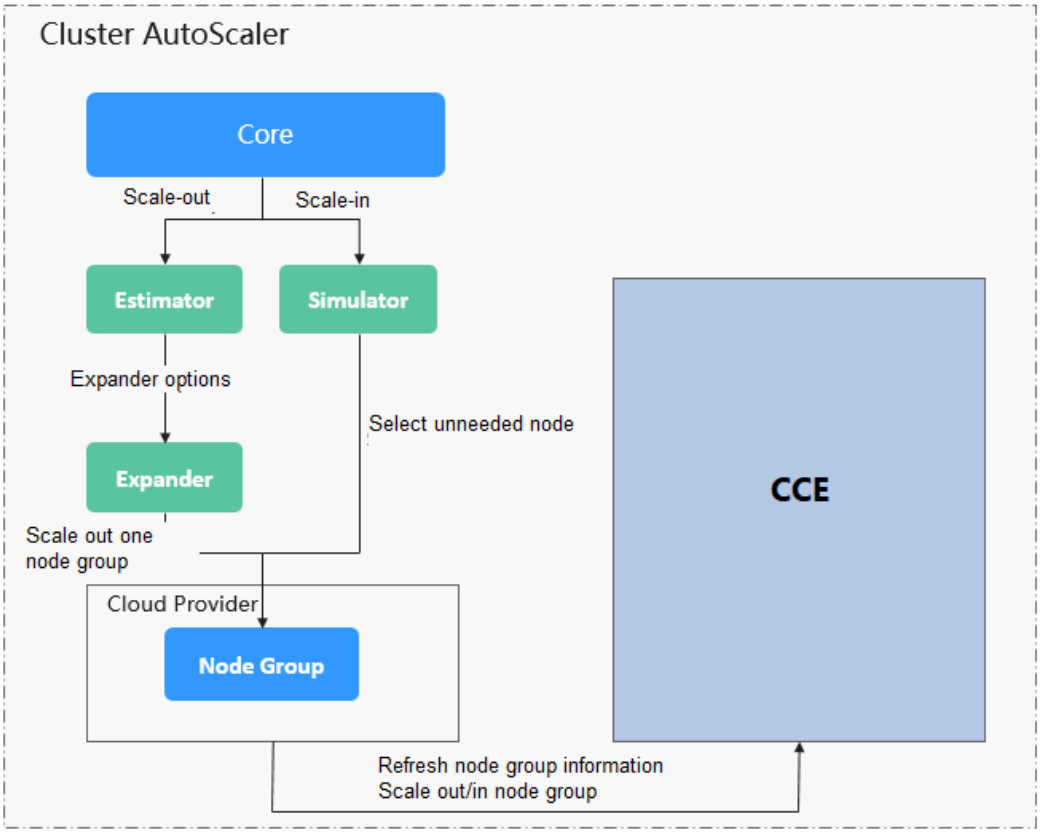
  - Pod有设置Pod Disruption Budget（即**干扰预算**），当移除Pod不满足对应条件时，节点不会缩容。
  - Pod由于一些限制，如亲和、反亲和等，无法调度到其他节点，节点不会缩容。
  - Pod拥有cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'这个annotations时，节点不缩容。
  - 节点上存在kube-system命名空间下的Pod（除kube-system命名空间下由DaemonSet创建的Pod），节点不缩容。
  - 节点上如果有非controller（Deployment/ReplicaSet/Job/StatefulSet）创建的Pod，节点不缩容。

  > **说明**
  >
  > 当节点符合缩容条件时，Autoscaler将预先给节点打上DeletionCandidateOfClusterAutoscaler污点，限制Pod调度到该节点上。当autoscaler插件被卸载后，如果节点上依然存在该污点请您手动进行删除。

Cluster AutoScaler 架构

Cluster AutoScaler架构如**图10-13**所示，主要由以下几个核心模块组成：

图 10-13 Cluster AutoScaler 架构图



说明如下：

- Estimator： 负责扩容场景下，评估满足当前不可调度Pod时，每个节点池需要扩容的节点数量。
- Simulator： 负责缩容场景下，找到满足缩容条件的节点。
- Expander： 负责在扩容场景下，根据用户设置的不同的策略来，从Estimator选出的节点池中，选出一个最佳的选择。当前Expander有多种策略，如**表10-17**。

表 10-17 CCE 支持的 Expander 策略

策略	策略说明	使用场景	模拟样例
Random	随机选择一个可调度节点池中执行本次扩容。	此策略通常作为其他更复杂策略的基础退避策略。仅当其他优选策略无法决策时，作为备选策略使用，通常不建议直接配置使用。	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，评估节点池1和节点池2中扩容的节点均可调度。 3. autoscaler决策优选节点池，将在节点池1和节点池2范围中随机选择一个节点池执行扩容。
most-pods	组合型策略，优先级排序为：most-pods > random。 优先选择扩容后能调度最多Pods的节点池。如果存在多个节点池满足条件，则基于random策略进一步决策。	此策略基于最多可调度Pods数量作为优选依据。	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，评估节点池1和节点池2中扩容的节点均可调度部分Pending Pods。 3. autoscaler决策优选节点池，评估节点池1扩容后可调度工作负载新增的20个Pods，而节点池2扩容仅可调度工作负载新增的10个Pods，因此优选节点池1执行本次扩容。

策略	策略说明	使用场景	模拟样例
least-waste	组合型策略，优先级排序为：least-waste > random。 评估本次扩容的节点池整体CPU或MEM资源分配率，优先选择具有最小浪费的CPU或者Mem资源的节点池。如果存在多个节点池满足条件，则基于random策略进一步决策。	此策略将CPU或者内存资源最小浪费分数作为优选依据。 其中最小浪费分数wastedScore定义公式如下： ● wastedCPU = (待扩容节点的CPU总量 - 待调度Pods的CPU总量) / 待扩容节点的CPU总量 ● wastedMemory = (待扩容节点的MEM总量 - 待调度Pods的MEM总量) / 待扩容节点的MEM总量 ● wastedScore = wastedCPU + wastedMemory	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，评估节点池1和节点池2中扩容的节点均可调度部分Pending Pods。 3. autoscaler决策优选节点池，评估节点池1扩容后最小浪费分数小于节点池2，因此优选节点池1执行本次扩容。

策略	策略说明	使用场景	模拟样例
priority	组合策略，优先级排序为：priority > least-waste > topology-balance > random。 基于节点池/伸缩组优先级配置增强的least-waste策略。如果存在多个节点池满足条件，则基于least-waste策略进一步决策。 在least-waste策略的基础上，topology-balance策略可基于不同可用区均衡节点数量，即优先选择节点数量少的AZ进行扩容，尽可能保证各个AZ间的节点数量均衡。该策略在1.23.122、1.25.117、1.27.85、1.28.52、1.29.14及以上版本的插件中可用。	priority可通过Console/API主动配置节点池/伸缩组优先级，least-waste则在通用场景下降低资源浪费比例。在此基础上，topology-balance策略可以尽可能保证各个AZ间的节点数量均衡。此策略通用性较好，当前作为**默认优选策略**。	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，评估节点池1和节点池2中扩容的节点均可调度部分Pending Pods。 3. autoscaler决策优选节点池，评估节点池1优先级高于节点池2，因此优选节点池1执行本次扩容。

策略	策略说明	使用场景	模拟样例
priority-ratio	组合策略，优先级排序为：priority > priority-ratio > least-waste > random。 基于priority策略的资源碎片重调度场景化配套策略，即在同优先级场景下，优先选择扩容后可使节点可分配资源的CPU/内存比，更接近于所有已调度Pods的申请的CPU/内存比。	此策略基于集群中全局Pods/Nodes全局资源而非仅扩容节点部分，主要配套重调度等相关能力降低集群整体资源碎片率，无相关配套独立使用场景不建议使用。	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，评估节点池1和节点池2中扩容的节点均可调度部分Pending Pods。 3. autoscaler决策优选节点池，评估Pod的CPU/内存比为1:4，节点池1中的节点规格为2U8G（CPU/内存比为1:4），节点池2中的节点规格为2U4G（CPU/内存比为1:2）。因此优选节点池1执行本次扩容。
topology-balance	组合策略，优先级排序为：topology-balance > least-waste > random。 基于不同可用区节点数量均衡的策略，即优先选择节点数量少的AZ进行扩容，尽可能保证各个AZ间的节点数量均衡。	使用topology-balance策略时，节点池优先级不生效。 在触发节点扩容时，可能同时扩容多个节点。本策略无法将一次性扩容的多个节点均分至各AZ，只保证多次扩容后，各AZ间节点数趋于均衡。	假设集群中节点池1和节点池2启用了弹性伸缩，且均未达到扩容上限。当工作负载扩容副本数时，扩容策略如下： 1. Pending Pods触发autoscaler决策扩容流程。 2. autoscaler模拟调度阶段，在开启弹性伸缩的节点池中，评估每个AZ的节点数。 3. autoscaler决策优选节点池，选择节点数量最少的AZ进行扩容。若两个AZ存在相同的节点数，则根据下一优先级的least-waste策略进行排序。

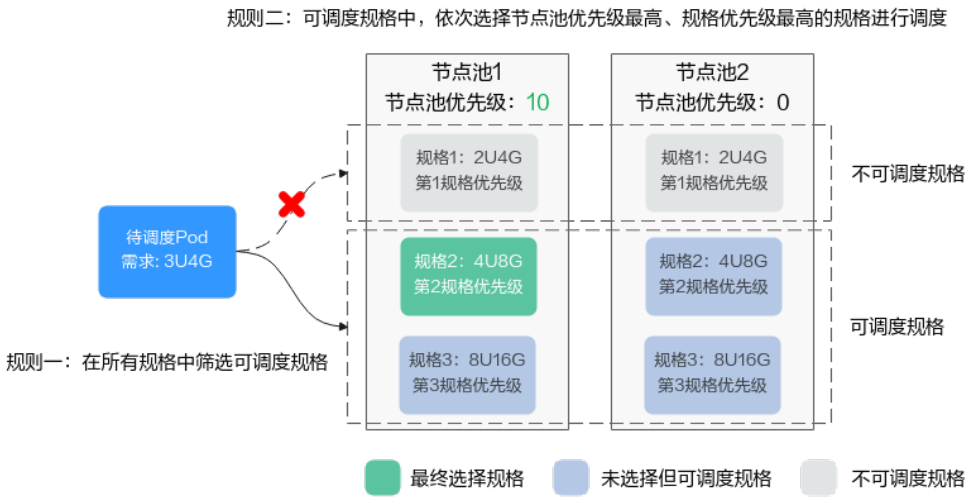
10.3.2 节点池弹性伸缩优先级说明

前提条件

如需使用节点规格优先级功能，autoscaler插件版本要求为1.19.35、1.21.28、1.23.30、1.25.20及以上。其中AZ均衡分布策略在1.23.122、1.25.117、1.27.85、1.28.52及以上支持。

弹性扩容策略

遵循节点池优先级和规格优先级的原则弹性扩容。



1. 预判规格筛选：
   - 通过预判算法，在所有节点池中选择能满足Pending状态的Pod正常调度的规格。
   - 考虑因素包括节点资源是否满足Pod的request值，以及nodeSelector、nodeAffinity和taints等是否满足Pod正常调度的条件。
   - 另外，部分节点池规格由于资源不足等扩容失败进入5min冷却期后，冷却期间扩容算法会自动过滤此类规格。
2. 节点池优先级排序：

   为每个节点池分配一个优先级，根据节点池优先级进行排序，优先选择优先级最高的节点池。
3. 规格优先级选择：

   如果存在多个节点池优先级最高的情况，则根据以下原则挑选优先级最高的规格：
   - 首先，选择节点池中优先级最高的规格。
   - 其次，如果存在规格优先级相同的情况，根据最小浪费原则，选择既能满足Pod正常调度、浪费资源又最少的规格。
   - 最后，如果存在多个规格满足最小浪费原则，则在可用区（AZ）均衡分布的基础上选择。
4. 处理资源不足或创建失败情况处理：

   如果首选规格因可用区资源售罄或配额不足等原因创建失败，将按照节点池内规格优先级的顺序，尝试创建下一个优先级的规格，原实例进入5分钟的冷却时间。

如果一个节点池中的所有规格都无法成功创建实例，系统将顺延至下一个优先级的节点池继续尝试。

手动扩容策略

当节点池进行手动扩缩容时，您可选择指定的规格进行伸缩。当选择的节点规格资源不足或配额不足时，会导致扩容失败。

设置优先级

关于如何设置节点池规格优先级详情请参见**配置集群弹性伸缩策略**。

10.3.3 创建节点弹性策略

CCE的自动伸缩能力是通过节点自动伸缩组件**CCE集群弹性引擎**实现的，可以按需弹出节点实例，支持多可用区、多实例规格、多种伸缩模式，满足不同的节点伸缩场景。

当节点伸缩中创建的策略和弹性伸缩插件中的配置同时生效时（比如不可调度和指标规则同时满足时），将优先执行不可调度扩容。

- 若不可调度执行成功，则会跳过指标规则逻辑，进入下一次循环。
- 若不可调度执行失败，将执行指标规则逻辑。

前提条件

使用节点伸缩功能前，集群中需要安装**CCE集群弹性引擎**插件，插件版本要求1.13.8及以上。

如需使用节点规格优先级功能，autoscaler插件版本要求为1.19.35、1.21.28、1.23.30、1.25.20及以上。其中AZ均衡分布策略在1.23.122、1.25.117、1.27.85、1.28.52及以上支持。

约束限制

- 当节点池中节点为0时，autoscaler插件无法获取节点CPU/内存数据，指标触发的节点弹性规则将不会生效。
- GPU/NPU节点驱动未安装成功时，autoscaler插件会认为该节点未完全可用，通过CPU/内存指标触发的节点弹性规则将不会生效。
- 使用autoscaler插件时，部分污点/注解可能会影响弹性伸缩功能，因此集群中应避免使用以下污点/注解：
  - **节点避免使用ignore-taint.cluster-autoscaler.kubernetes.io的污点**：该污点作用于节点。由于autoscaler原生支持异常扩容保护策略，会定期评估集群的可用节点比例，非Ready分类节点数统计比例超过45%比例会触发保护机制；而集群中任何存在该污点的节点都将从自动缩放器模板节点中过滤掉，记录到非Ready分类的节点中，进而影响集群的扩缩容。
  - **Pod避免使用cluster-autoscaler.kubernetes.io/enable-ds-eviction的注解**：该注解作用于Pod，控制DaemonSet Pod是否可以被autoscaler驱逐。详情请参见**Kubernetes原生的标签、注解和污点**。

配置节点池弹性伸缩策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“节点管理”，在目标节点池所在行右上角单击“弹性伸缩”。

- 若未安装autoscaler插件，请根据业务需求配置插件参数后单击“安装”，并等待插件安装完成。插件配置详情请参见**CCE集群弹性引擎**。
- 若已安装autoscaler插件，则可直接配置弹性伸缩策略。

步骤3 配置节点池弹性伸缩策略。

伸缩配置

- 自定义扩容规则：单击“添加规则”，在弹出的添加规则窗口中设置参数。您可以设置多条节点弹性策略，最多可以添加1条CPU使用率指标规则、1条内存使用率指标规则，且规则总数小于等于10条。

  规则类型可选择“指标触发”或“周期触发”，两种类型区别如下：

  表 10-18 自定义规则类型

规则类型	参数设置
指标触发	– 触发条件：请选择“CPU分配率”或“内存分配率”，输入百分比的值。该百分比应大于**配置集群弹性伸缩策略**时节点缩容的“节点资源条件”。 **说明** ■ 分配率 = 节点池容器组（Pod）资源申请量 / 节点池Pod可用资源量(Node Allocatable)。 ■ **如果多条规则同时满足条件，会有如下两种执行的情况：** 如果同时配置了“CPU分配率”和“内存分配率”的规则，两种或多种规则同时满足扩容条件时，执行扩容节点数更多的规则。 如果同时配置了“CPU分配率”和“周期触发”的规则，当达到“周期触发”的时间值时CPU也满足扩容条件时，较早执行的周期触发规则会将节点池状态置为伸缩中状态，导致指标触发规则无法正常执行。待周期触发规则执行完毕，节点池状态恢复正常后，指标触发规则也不会执行。反之，如果指标触发规则执行较早，则等指标规则执行完毕后周期规则仍会执行。 ■ 配置了“CPU分配率”和“内存分配率”的规则后，策略的检测周期会因autoscaler每次循环的处理逻辑而变动。只要一次检测出满足条件就会触发扩容（还需要满足冷却时间、节点池状态等约束条件）。 ■ 当节点数已到达集群规模上限、**所属节点池的节点数上限**或**该规格的节点数上限**时，将不会触发指标扩容。 ■ 当节点数量、CPU、内存达到autoscaler插件设置的**节点扩容资源上限**时，将不会触发指标扩容。 – 执行动作：达到触发条件后所要执行的动作。 ■ 自定义：为节点池增加指定数量的节点。 ■ 自动计算：当达到触发条件时，自动扩容节点，将分配率恢复到触发条件以下。计算公式如下： 扩容节点数 = 节点池容器组（Pod）资源申请值 /（单节点可用资源值 * 目标分配率） - 当前节点数 + 1

规则类型	参数设置
周期触发	– 触发时间：可选择每天、每周、每月或每年的具体时间点。 – 执行动作：达到触发时间值后所要执行的动作，为节点池增加指定数量的节点。

- 节点数范围：弹性伸缩时节点池下的节点数量会始终介于节点数范围内。
- 冷却时间：指当前节点池扩容出的节点多长时间不能被缩容。

伸缩对象

- 规格选择：对节点池中的节点规格单独设置是否开启弹性伸缩。

  说明

  当节点池中包含多个规格时，您可以对每个规格的节点数范围和优先级进行单独配置。

步骤4 查看集群级别的弹性伸缩配置，集群级别的配置对所有节点池生效。当前页面仅支持查看集群级别的弹性伸缩策略，如需修改请前往“配置中心”进行设置，详情请参见**配置集群弹性伸缩策略**。

步骤5 设置完成后，单击“确定”。

----结束

配置集群弹性伸缩策略

说明

集群弹性伸缩策略对集群下的所有节点池都会生效，且修改后会重启autoscaler插件。

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“配置中心”，单击“集群弹性伸缩配置”页签。

- 若未安装autoscaler插件，请根据业务需求配置插件参数后单击“安装”，并等待插件安装完成。插件配置详情请参见**CCE集群弹性引擎**。
- 若已安装autoscaler插件，则可直接配置弹性伸缩策略。

步骤3 设置弹性扩容配置。

- 负载无法调度时自动扩容：当集群下负载实例无法调度时自动扩容（从节点池），即当出现Pod处于Pending状态无法调度时，集群会自动扩容节点。若Pod已经指定调度到某个节点，则不会自动扩容节点。该功能一般与HPA策略配合使用，具体请参见**使用HPA+CA实现工作负载和节点联动弹性伸缩**。

  如不开启，则只能通过**自定义扩容规则**进行扩缩容。
- 节点扩容资源上限：设置集群中的总资源量上限，包含节点数量、CPU核数、内存总量上限，达到配置的资源上限后将不再自动扩容节点。
- 节点池扩容优先级：节点池列表可通过拖拽调整扩容优先级。

步骤4 设置弹性缩容配置。弹性缩容默认不开启，开启后支持以下配置。

节点缩容条件：当集群下的节点满足缩容条件时会被自动缩容。

- 节点资源条件：当集群节点资源的Request值（CPU和内存需同时满足）连续一段时间（默认10min）低于一定百分比（默认50%）时，会触发集群缩容操作。

- 节点状态条件：节点处于不可用状态下超过一定时间会被自动回收，默认为20分钟。
- 缩容例外场景：节点满足以下例外场景时，即使节点资源或状态满足缩容条件，不会被CCE集群弹性引擎自动缩容。
  a. 集群其它节点资源不足时将不会触发非完全空闲节点缩容。
  b. 节点开启缩容保护时将不会触发节点缩容。如需开启或关闭节点缩容保护，请前往“节点管理 > 节点”页面，单击节点操作列的“更多 > 开启/关闭节点缩容保护”按钮操作。
  c. 节点上存在指定不缩容标记的Pod时，该节点将不会被缩容。
  d. 节点上的部分容器存在可靠性等配置策略时，将有可能不会自动缩容。
  e. 节点上存在kube-system命名空间下的非DaemonSet类容器时，该节点将不会被缩容。
  f. （可选）节点上如果存在已运行的容器由第三方Pod Controller进行管理，则该节点不会被缩容。第三方Pod Controller是指除Kubernetes原生的工作负载（如Deployment、StatefulSet等）外的自定义工作负载，可通过**自定义资源CRD**进行创建。

节点缩容策略

- 缩容并发数：最多支持多少个空闲节点同时缩容，默认10。
  缩容并发数只针对完全空闲节点，完全空闲节点可实现并发缩容。非完全空闲节点则只能逐个缩容。

  说明

  节点在缩容的时候，若节点上的Pod不需要驱逐（DaemonSet的Pod认为不需要驱逐），则认为该节点为完全空闲节点，否则认为该节点为非完全空闲。
- 检查周期：节点被判定不可移除后能再次启动检查的时间间隔，默认5min。
- 冷却时间：
  - 集群触发弹性缩容后，再次启动缩容评估的冷却时间：默认10min。
  - 集群触发弹性扩容后，再次启动缩容评估的冷却时间：删除节点后能再次启动缩容评估的时间间隔，默认10min。

    说明

    集群中如果同时存在自动扩容和自动缩容的场景，建议配置“集群触发弹性扩容后，再次启动缩容评估的冷却时间”为0min，避免由于部分节点池持续扩容或者扩容失败重试而阻塞整体缩容节点行为，导致非预期的节点资源浪费。
  - 集群触发弹性缩容失败后，再次启动缩容评估的冷却时间：缩容失败后能再次启动缩容评估的时间间隔，默认3min。节点池中配置的弹性扩容冷却时间和此处配置的弹性缩容冷却时间之间的影响和关系请参见**冷却时间说明**。

步骤5 配置修改完成后，单击“确认配置”。

----结束

冷却时间说明

节点池中配置的两个冷却时间之间的影响和关系如下：

弹性扩容中的冷却时间

弹性缩容冷却时间：当前节点池扩容出的节点多长时间不能被缩容，作用范围为节点池级别。

弹性缩容中的冷却时间

扩容后缩容冷却时间：autoscaler触发扩容后（不可调度、指标、周期策略）整个集群多长时间内不能被缩容，作用范围为集群级别。

节点删除后缩容冷却时间：autoscaler触发缩容后整个集群多长时间内不能继续缩容，作用范围为集群级别。

缩容失败后缩容冷却时间：autoscaler触发缩容失败后整个集群多长时间内不能继续缩容，作用范围为集群级别。

AutoScaler 重试扩容的周期说明

当节点池扩容因为资源售罄、配额不足、节点安装过程中错误等原因失败时，AutoScaler 能够重试同一节点池或者切换其他节点池扩容，重试扩容的周期如下:

- 当节点池资源售罄或者用户配额不足时，AutoScaler将按照5min、10min、20min对节点池进行冷却，最多冷却30min。 同时AutoScaler将在下一个10s切换其他节点池进行扩容，直到扩容出期望的节点或者所有节点池都进入冷却。
- 当节点池在节点安装过程中发生错误时，节点池会进入5min冷却期。 冷却期过后，AutoScaler才可以重新触发该节点池扩容。 当安装过程中的错误节点被自动回收后，Cluster AutoScaler将在1min内重新评估集群状态，按需触发节点池扩容。
- 在节点池扩容时，如果节点池节点长时间处于安装中状态，Cluster AutoScaler将最多容忍此类节点15min，超过容忍时间后，将重新评估集群状态，按需触发节点池扩容。

YAML 样例

节点弹性策略Yaml样例如下：

```
apiVersion: autoscaling.cce.io/v1alpha1
kind: HorizontalNodeAutoscaler
metadata:
  name: xxxx
  namespace: kube-system
spec:
  disable: false
  rules:
  - action:
      type: ScaleUp
      unit: Node
      value: 1
    cronTrigger:
      schedule: 47 20 * * *
    disable: false
    ruleName: cronrule
    type: Cron
  - action:
      type: ScaleUp
      unit: Node
      value: 2
    disable: false
    metricTrigger:
      metricName: Cpu
      metricOperation: '>'
      metricValue: "40"
      unit: Percent
```



```
  ruleName: metricrule
  type: Metric
targetNodepoolIds:
- 7d48eca7-3419-11ea-bc29-0255ac1001a8
```

表 10-19 关键参数说明

参数	参数类型	描述
spec.disable	Bool	伸缩策略开关，会对策略中的所有规则生效
spec.rules	Array	伸缩策略中的所有规则
spec.rules[x].ruleName	String	规则名称
spec.rules[x].type	String	规则类型，当前支持“Cron”和“Metric”两种类型
spec.rules[x].disable	Bool	规则开关，当前仅支持“false”
spec.rules[x].action.type	String	规则操作类型，当前仅支持“ScaleUp”
spec.rules[x].action.unit	String	规则操作单位，当前仅支持“Node”
spec.rules[x].action.value	Integer	规则操作数值
spec.rules[x].cronTrigger	/	可选，仅在周期规则中有效
spec.rules[x].cronTrigger.schedule	String	周期规则的cron表达式
spec.rules[x].metricTrigger	/	可选，仅在指标规则中有效
spec.rules[x].metricTrigger.metricName	String	指标规则对应的指标，当前支持“Cpu”和“Memory”两种类型
spec.rules[x].metricTrigger.metricOperation	String	指标规则的比较符，当前仅支持“>”
spec.rules[x].metricTrigger.metricValue	String	指标规则的阈值，支持1-100之间的所有整数，需以字符串表示；如果设置为-1则表示自动计算
spec.rules[x].metricTrigger.Unit	String	指标规则阈值的单位，当前仅支持“%”
spec.targetNodepoolIds	Array	伸缩策略关联的所有节点池
spec.targetNodepoolIds[x]	String	伸缩策略关联节点池的uid

10.3.4 管理节点弹性策略

操作场景

节点弹性策略创建完成后，可对创建的策略进行删除、编辑、停用、启用、克隆等操作。

查看节点弹性策略

您可以查看节点弹性策略的关联节点池、执行规则和伸缩历史，参照界面中的提示有针对性的解决异常问题。

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏的“节点管理”，单击已创建弹性伸缩策略的节点池名称，查看节点池详情。

步骤3 在节点池详情中切换至“弹性伸缩”页签，可以看到弹性伸缩策略的配置及伸缩记录。

图 10-14 查看弹性伸缩策略

说明

您还可以在“策略”页面中查看已创建的弹性伸缩策略：

1. 登录CCE控制台，单击集群名称进入集群。
2. 在左侧导航栏中单击“策略”，切换至“节点伸缩策略”页签。
3. 您可以查看弹性伸缩策略的配置。单击要策略后方的“更多 > 伸缩历史”，您可以查看该策略的伸缩记录。

----结束

删除节点弹性策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中单击“策略”，切换至“节点弹性策略”页签，单击要删除的策略后方的“更多 > 删除”。

图 10-15 删除节点弹性策略

步骤3 在弹出的“删除节点弹性策略”窗口中，确认是否删除。

步骤4 单击“是”按钮即完成删除操作。

----结束

编辑节点弹性策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中单击“策略”，切换至“节点弹性策略”页签，单击要编辑的策略后方的“编辑”。

图 10-16 编辑节点弹性策略

步骤3 在打开的“编辑节点弹性策略”页面中，参照表10-19更新策略参数。

步骤4 完成设置后，单击“确定”按钮完成编辑操作。

----结束

克隆节点弹性策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中单击“策略”，切换至“节点弹性策略”页签，单击要克隆的策略后方的“更多 > 克隆”。

图 10-17 克隆节点弹性策略

步骤3 在打开的“创建节点弹性策略”页面中，可以看到部分参数已经克隆过来，请按照业务需求补充或修改其他策略参数。

步骤4 单击“确定”完成策略克隆。

----结束

停用/启用节点弹性策略

步骤1 在CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中单击“策略”，切换至“节点弹性策略”页签，单击策略后方的“停用”，若策略为停用状态时，则单击策略后方的“启用”。

图 10-18 停用节点弹性策略

步骤3 在弹出的“停用节点策略”或“启用节点策略”窗口中，确认是否进行停用或启用操作。

----结束

10.4 使用 HPA+CA 实现工作负载和节点联动弹性伸缩

应用场景

企业应用的流量大小不是每时每刻都一样，有高峰，有低谷，如果每时每刻都要保持能够扛住高峰流量的机器数目，那么成本会很高。通常解决这个问题的办法就是根据流量大小或资源占用率自动调节机器的数量，也就是弹性伸缩。

当使用Pod/容器部署应用时，通常会设置容器的申请/限制值来确定可使用的资源上限，以避免在流量高峰期无限制地占用节点资源。然而，这种方法可能会存在资源瓶颈，达到资源使用上限后可能会导致应用出现异常。为了解决这个问题，可以通过伸缩Pod的数量来分摊每个应用实例的压力。如果增加Pod数量后，节点资源使用率上升到一定程度，继续扩容出来的Pod无法调度，则可以根据节点资源使用率继续伸缩节点数量。

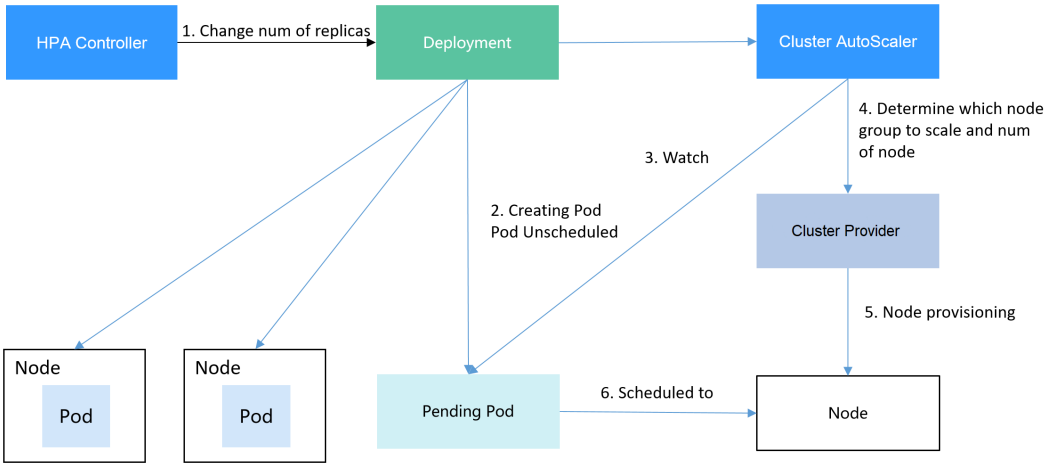
解决方案

CCE中弹性伸缩最主要的就是使用HPA（Horizontal Pod Autoscaling）和CA（Cluster AutoScaling）两种弹性伸缩策略，HPA负责工作负载弹性伸缩，也就是应用层面的弹性伸缩，CA负责节点弹性伸缩，也就是资源层面的弹性伸缩。

通常情况下，两者需要配合使用，因为HPA需要集群有足够的资源才能扩容成功，当集群资源不够时需要CA扩容节点，使得集群有足够资源；而当HPA缩容后集群会有大量空余资源，这时需要CA缩容节点释放资源，才不至于造成浪费。

如图10-19所示，HPA根据监控指标进行扩容，当集群资源不够时，新创建的Pod会处于Pending状态，CA会检查所有Pending状态的Pod，根据用户配置的扩缩容策略，选择出一个最合适的节点池，在这个节点池扩容。HPA和CA的工作原理详情请参见**工作负载伸缩原理**和**节点伸缩原理**。

图 10-19 HPA + CA 工作流程



使用HPA+CA可以很容易做到弹性伸缩，且节点和Pod的伸缩过程可以非常方便地观察到，使用HPA+CA做弹性伸缩能够满足大部分业务场景需求。

本文将通过一个示例介绍HPA+CA两种策略配合使用下弹性伸缩的过程，从而帮助您更好地理解和使用弹性伸缩。

准备工作

步骤1 创建一个有1个节点的集群，节点规格为2U4G及以上，并在创建节点时为节点添加弹性公网IP，以便从外部访问。如创建节点时未绑定弹性公网IP，您也可以前往ECS控制台为该节点进行手动绑定。

步骤2 给集群安装插件。

- autoscaler：节点伸缩插件。
- metrics-server：是Kubernetes集群范围资源使用数据的聚合器，能够收集包括了Pod、Node、容器、Service等主要Kubernetes核心资源的度量数据。

步骤3 登录集群节点，准备一个算力密集型的应用。当用户请求时，需要先计算出结果后才返回给用户结果，如下所示。

1. 创建一个名为index.php的PHP文件，文件内容是在用户请求时先循环开方1000000次，然后再返回“OK!”。

```
vi index.php
```


文件内容如下：

```
<?php
 $x = 0.0001;
 for ($i = 0; $i <= 1000000; $i++) {
   $x += sqrt($x);
 }
 echo "OK!";
?>
```

2. 编写Dockerfile制作镜像。

```
vi Dockerfile
```

Dockerfile内容如下：

```
FROM php:5-apache
COPY index.php /var/www/html/index.php
RUN chmod a+rx index.php
```

3. 执行如下命令构建镜像，镜像名称为hpa-example，版本为latest。

```
docker build -t hpa-example:latest .
```

4. （可选）登录SWR管理控制台，选择左侧导航栏的“组织管理”，单击页面右上角的“创建组织”，创建一个组织。

   如已有组织可跳过此步骤。

5. 在左侧导航栏选择“我的镜像”，单击右侧“客户端上传”，在弹出的页面中单击“生成临时登录指令”，单击 复制登录指令。

6. 在集群节点上执行上一步复制的登录指令，登录成功会显示“Login Succeeded”。

7. 为hpa-example镜像添加标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - **[镜像名称1:版本名称1]**：请替换为您本地所要上传的实际镜像的名称和版本名称。
   - **[镜像仓库地址]**：可在SWR控制台上查询，**登录指令**中末尾的域名即为镜像仓库地址。
   - **[组织名称]**：请替换为**已创建的组织名称**。
   - **[镜像名称2:版本名称2]**：请替换为SWR镜像仓库中需要显示的镜像名称和镜像版本。

   示例：

   docker tag hpa-example:latest swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/hpa-example:latest

8. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   docker push swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/hpa-example:latest

   终端显示如下信息，表明上传镜像成功。

```
6d6b9812c8ae: Pushed
...
fe4c16cbf7a4: Pushed
latest: digest: sha256:eb7e3bbd*** size: **
```

   返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

----结束

创建节点池和节点伸缩策略

步骤1 登录CCE控制台，进入已创建的集群，在左侧单击“节点管理”，选择“节点池”页签并单击右上角“创建节点池”。

步骤2 填写节点池配置。

- 节点类型：选择节点类型
- 节点规格：2核 | 4GiB

其余参数设置可使用默认值，详情请参见**创建节点池**。

步骤3 节点池创建完成后，在目标节点池所在行右上角单击“弹性伸缩”，设置弹性伸缩配置。关于节点伸缩策略设置的详细说明，请参见**创建节点伸缩策略**。

若集群中未安装CCE集群弹性引擎插件，请先安装该插件。详情请参见**CCE集群弹性引擎**。

- 自定义扩容规则：单击“添加规则”，在弹出的添加规则窗口中设置参数。例如CPU分配率大于70%时，关联的节点池都增加一个节点。CA策略需要关联节点池，可以关联多个节点池，当需要对节点扩缩容时，在节点池中根据最小浪费规则挑选合适规格的节点扩缩容。
- 节点数范围：修改节点数范围，弹性伸缩时节点池下的节点数量会始终介于节点数范围内。
- 冷却时间：当前节点池扩容出的节点多长时间不能被缩容。

步骤4 设置完成后，单击“确定”。

----结束

创建工作负载

使用构建的hpa-example镜像创建无状态工作负载，副本数为1，镜像地址与上传到SWR仓库的组织有关，需要替换为实际取值。

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: hpa-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hpa-example
  template:
    metadata:
      labels:
        app: hpa-example
    spec:
      containers:
      - name: container-1
        image: 'hpa-example:latest' # 替换为您上传到SWR的镜像地址
        resources:
          limits:                   # limits与requests建议取值保持一致，避免扩缩容过程中出现震荡
            cpu: 500m
            memory: 200Mi
          requests:
            cpu: 500m
            memory: 200Mi
      imagePullSecrets:
      - name: default-secret
```

然后再为这个负载创建一个Nodeport类型的Service，以便能从外部访问。

说明

Nodeport类型的Service从外网访问需要为集群某个节点创建EIP，创建完后需要同步节点信息，具体请参见**同步节点信息**。如果节点已有EIP则无需再次创建。

或者您也可以创建带ELB的Service从外部访问，具体请参见**通过kubectl命令行创建-自动创建ELB**。

```
kind: Service
apiVersion: v1
metadata:
  name: hpa-example
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 31144
  selector:
    app: hpa-example
  type: NodePort
```

创建 HPA 策略

创建HPA策略，如下所示，该策略关联了名为hpa-example的负载，期望CPU使用率为50%。

另外有两条注解annotations，一条是CPU的阈值范围，最低30，最高70，表示CPU使用率在30%到70%之间时，不会扩缩容，防止小幅度波动造成影响。另一条是扩缩容时间窗，表示策略成功触发后，在缩容/扩容冷却时间内，不会再次触发缩容/扩容，以防止短期波动造成影响。

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-policy
  annotations:
    extendedhpa.metrics: '[{"type":"Resource","name":"cpu","targetType":"Utilization","targetRange":{"low":"30","high":"70"}}]'
    extendedhpa.option: '{"downscaleWindow":"5m","upscaleWindow":"3m"}'
spec:
  scaleTargetRef:
    kind: Deployment
    name: hpa-example
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 100
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

在控制台创建则参数填写如下所示。

观察弹性伸缩过程

步骤1 首先查看集群节点情况，如下所示，有两个节点。

```
# kubectl get node
NAME            STATUS   ROLES    AGE     VERSION
192.168.0.183   Ready    <none>   2m20s   v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.26    Ready    <none>   55m     v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
```

查看HPA策略，可以看到目标负载的指标（CPU使用率）为0%

```
# kubectl get hpa hpa-policy
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-policy   Deployment/hpa-example   0%/50%    1         100       1          4m
```

步骤2 通过如下命令访问负载，如下所示，其中{ip:port}为负载的访问地址，可以在负载的详情页中查询。

while true;do wget -q -O- http://*{ip:port}*; done

说明

如果此处不显示公网IP地址，则说明集群节点没有弹性公网IP，请创建弹性公网IP并绑定到节点，创建完后需要同步节点信息，具体请参见**同步节点信息**。

观察负载的伸缩过程。

```
# kubectl get hpa hpa-policy --watch
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hpa-policy   Deployment/hpa-example   0%/50%     1         100       1          4m
hpa-policy   Deployment/hpa-example   190%/50%   1         100       1          4m23s
hpa-policy   Deployment/hpa-example   190%/50%   1         100       4          4m31s
hpa-policy   Deployment/hpa-example   200%/50%   1         100       4          5m16s
hpa-policy   Deployment/hpa-example   200%/50%   1         100       4          6m16s
hpa-policy   Deployment/hpa-example   85%/50%    1         100       4          7m16s
hpa-policy   Deployment/hpa-example   81%/50%    1         100       4          8m16s
hpa-policy   Deployment/hpa-example   81%/50%    1         100       7          8m31s
hpa-policy   Deployment/hpa-example   57%/50%    1         100       7          9m16s
hpa-policy   Deployment/hpa-example   51%/50%    1         100       7          10m
hpa-policy   Deployment/hpa-example   58%/50%    1         100       7          11m
```

可以看到4m23s时负载的CPU使用率为190%，超过了目标值，此时触发了负载弹性伸缩，将负载扩容为4个副本/Pod，随后的几分钟内，CPU使用并未下降，直到7m16s时CPU使用率才开始下降，这是因为新创建的Pod并不一定创建成功，可能是因为资源不足Pod处于Pending状态，这段时间内在扩容节点。

到7m16s时CPU使用率开始下降，说明Pod创建成功，开始分担请求流量，到8分钟时下降到81%，还是高于目标值，且高于70%，说明还会再次扩容，到9m16s时再次扩容到7个Pod，这时CPU使用率降为51%，在30%-70%的范围内，不会再次伸缩，可以观察到此后Pod数量一直稳定在7个。

观察负载和HPA策略的详情，从事件中可以看到负载的扩容的过程和策略生效的时间，如下所示。

```
# kubectl describe deploy hpa-example
...
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  25m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 1
  Normal  ScalingReplicaSet  20m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 4
  Normal  ScalingReplicaSet  16m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 7
# kubectl describe hpa hpa-policy
...
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  20m    horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  16m    horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization (percentage of request) above target
```

此时查看节点数量，发现节点多了两个，也就是在刚才过程中节点扩容了两个。

```
# kubectl get node
NAME          STATUS   ROLES    AGE     VERSION
192.168.0.120   Ready    <none>   3m5s    v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.136   Ready    <none>   6m58s   v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.183   Ready    <none>   18m     v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.26    Ready    <none>   71m     v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
```

在控制台也可以看到伸缩历史，这里可以看到CA策略执行了一次，当集群中CPU分配率大于70%，将节点池中节点数量从2扩容到3。另一个节点是autoscaler默认的根据Pod的Pending状态扩容而来，在HPA初期。

这里节点扩容过程具体是这样：

1. Pod数量变为4后，由于没有资源，Pod处于Pending状态，触发了autoscaler默认的扩容策略，将节点数增加一个。
2. 第二次节点扩容是因为集群中CPU分配率大于70%，触发了CA策略，从而将节点数增加一个，从控制台上伸缩历史可以看出来。根据分配率扩容，可以保证集群一直处于资源充足的状态。

步骤3 停止访问负载，观察负载Pod数量。

```
# kubectl get hpa hpa-policy --watch
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hpa-policy   Deployment/hpa-example   50%/50%    1         100       7          12m
hpa-policy   Deployment/hpa-example   21%/50%    1         100       7          13m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       7          14m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       7          18m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          18m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          19m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          19m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          19m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          19m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          23m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       3          23m
hpa-policy   Deployment/hpa-example   0%/50%     1         100       1          23m
```


可以看到从13m开始CPU使用率为21%，18m时Pod数量缩为3个，到23m时Pod数量缩为1个。

观察负载和HPA策略的详情，从事件中可以看到负载的扩容的过程和策略生效的时间，如下所示。

```
# kubectl describe deploy hpa-example
...
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  25m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 1
  Normal  ScalingReplicaSet  20m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 4
  Normal  ScalingReplicaSet  16m    deployment-controller  Scaled up replica set hpa-example-79dd795485 to 7
  Normal  ScalingReplicaSet  6m28s  deployment-controller  Scaled down replica set hpa-example-79dd795485 to 3
  Normal  ScalingReplicaSet  72s    deployment-controller  Scaled down replica set hpa-example-79dd795485 to 1
# kubectl describe hpa hpa-policy
...
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  20m    horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  16m    horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  6m45s  horizontal-pod-autoscaler  New size: 3; reason: All metrics below target
  Normal  SuccessfulRescale  90s    horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
```

在控制台同样可以看到HPA策略生效历史，再继续等待，会看到节点也会被缩容一个。

这里为何没有被缩容掉两个节点，是因为节点池中这两个节点都存在kube-system namespace下的Pod（且不是DaemonSets创建的Pod），节点在什么情况下不会缩容请参见**Cluster Autoscaler工作原理**。

----结束

总结

通过上述内容可以看到，使用HPA+CA可以很容易做到弹性伸缩，且节点和Pod的伸缩过程可以非常方便地观察到，使用HPA+CA做弹性伸缩能够满足大部分业务场景需求。

10.5 CCE 容器实例弹性伸缩到 CCI 服务

CCE突发弹性引擎（对接 CCI）作为一种虚拟的kubelet用来连接Kubernetes集群和其他平台的API。Bursting的主要场景是将Kubernetes API扩展到无服务器的容器平台（如CCI）。

基于该插件，支持用户在短时高负载场景下，将部署在云容器引擎CCE上的无状态负载（Deployment）、有状态负载（StatefulSet）、普通任务（Job）、定时任务（CronJob）四种资源类型的容器实例（Pod），弹性创建到**云容器实例CCI**服务上，以减少集群扩容带来的消耗。

安装插件

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“插件中心”，进入插件中心首页。
4. 选择“CCE 突发弹性引擎 (对接 CCI)”插件，单击“安装”。
5. 配置插件参数。

表 10-20 插件参数说明

插件参数	说明
选择版本	插件的版本。插件版本和CCE集群存在配套关系，更多信息可以参考**CCE突发弹性引擎（对接CCI）插件版本记录**。

插件参数	说明

规格配置	用于配置插件负载的实例数及资源配额。
	• 选择“系统预置规格”时，您可选择“单实例”或“高可用”规格。
	• 选择“自定义规格”时，您可根据需求修改插件各个组件的副本数以及CPU/内存配置。
	说明
	• CCE 突发弹性引擎 (对接 CCI) 插件在1.5.2及以上版本，将占用更多节点资源，请在升级CCE突发弹性引擎（对接 CCI）插件前预留空间配额。
	  • 单实例：需要预留一个节点，节点下至少需要有7个Pod空间配额。若开启网络互通，则需要有8个Pod空间配额。
	  • 高可用：需要预留两个节点，节点下至少需要有7个Pod空间配额，共计14个Pod空间配额。若开启网络互通，则需要有8个Pod空间配额，共计16个Pod空间配额。
	• 弹性到CCI的业务量不同时，插件的资源占用也不相同。业务申请的POD、Secret、Congfigmap、PV、PVC会占用虚机资源。建议用户评估自己的业务使用量，按以下规格申请对应的虚机大小：1000pod+1000CM（300KB）推荐2U4G规格节点，2000pod+2000CM推荐4U8G规格节点，4000pod+4000CM推荐8U16G规格节点。

网络互通	开启后，支持CCE集群中的Pod与CCI集群中的Pod通过Kubernetes Service互通，并在插件安装时部署组件proxy。详细功能介绍请参考**网络**。

工作负载下发

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“工作负载”，进入工作负载首页。
4. 单击“创建工作负载”，具体操作步骤详情请参见**创建工作负载**。
5. 填写基本信息。“CCI弹性承载”选择“强制调度策略”。关于调度策略更多信息，请参考**调度负载到CCI**。

> **⚠ 注意**
>
> CCE集群创建工作负载时，需要弹性到CCI，健康检查不支持配置TCP启动探针。

6. 进行容器配置。
7. 配置完成后，单击“创建工作负载”。
8. 在工作负载页面，选择工作负载名称，单击进入工作负载管理界面。
9. 工作负载所在节点为CCI集群，说明负载成功已调度到CCI。

插件卸载

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“插件中心”，进入插件中心首页。
4. 选择“CCE 突发弹性引擎 (对接 CCI)”插件，单击“卸载”。

表 10-21 特殊场景说明

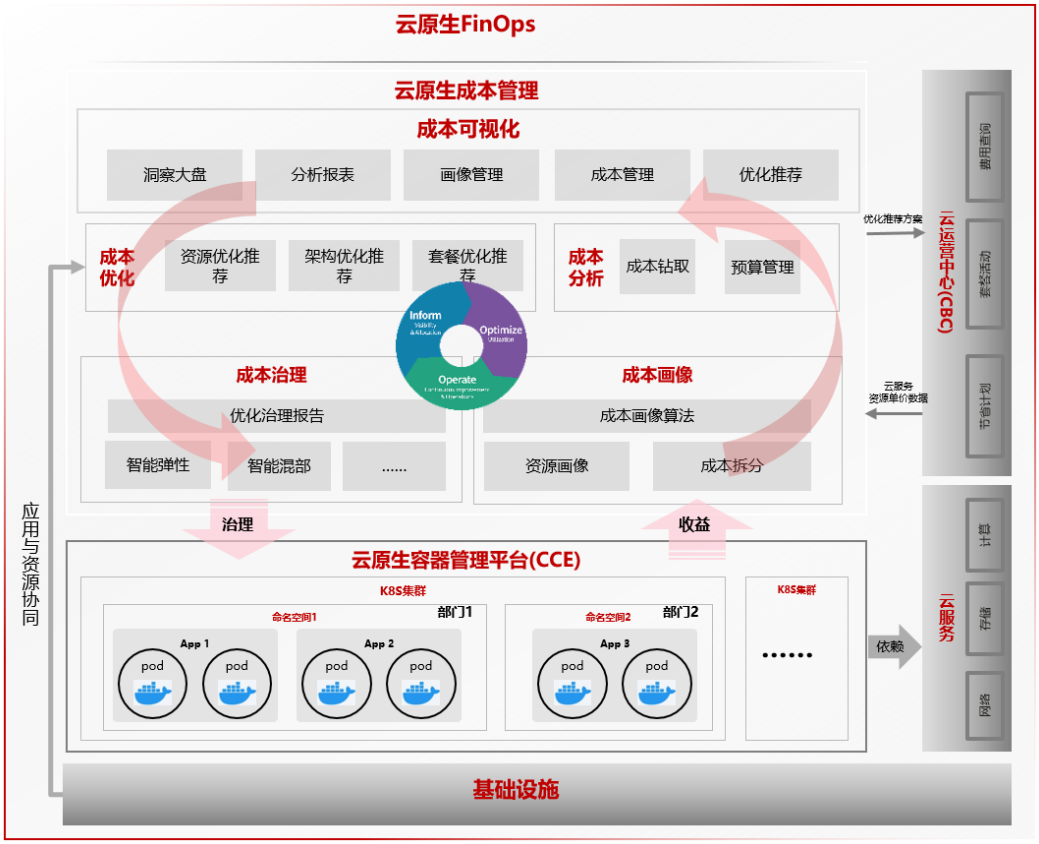
特殊场景描述	场景现象	场景说明
CCE集群无节点，卸载插件。	插件卸载失败。	bursting插件卸载时会在集群中启动Job用于清理资源，卸载插件时请保证集群中至少有一个可以调度的节点。
用户直接删除集群，未卸载插件。	用户在CCI侧的命名空间中有资源残留，如果命名空间有计费资源，会造成额外计费。	由于直接删除集群，没有执行插件的资源清理Job，造成资源残留。用户可以手动清除残留命名空间及其下的计费资源来避免额外计费。

关于CCE突发弹性引擎（对接CCI）更多内容详情请参见：**CCE突发弹性引擎（对接CCI）**。

11 云原生成本治理

11.1 云原生成本治理概述

云原生成本治理是基于FinOps理念的容器成本治理解决方案，提供部门维度、集群维度、命名空间维度的成本和资源画像，并通过工作负载资源推荐等优化手段协助企业IT成本管理人员实现容器集群的提效降本诉求。



成本洞察

成本洞察基于真实账单和集群资源用量统计数据，通过自研的成本画像算法进行成本拆分，提供以部门、集群、命名空间、应用等维度的成本画像。成本洞察能够帮助成

本管理人员分析集群成本开销、资源使用状况，识别资源浪费，为下一步的成本优化提供输入。

11.2 成本洞察

11.2.1 成本洞察概述

成本洞察基于真实账单和集群资源用量统计数据，通过自研的成本画像算法进行成本拆分，提供以部门、集群、命名空间、应用等维度的成本画像。成本洞察能够帮助成本管理人员分析集群成本开销、资源使用状况，识别资源浪费，为下一步的成本优化提供输入。

成本洞察从Region视角和集群资源视角展示用户的容器成本使用情况。其中：

- Region视角成本洞察：以企业管理人员的角度，呈现整体Region级别容器成本分析报告。该视角支持用户按照集群、命名空间粒度进行部门划分，并形成部门的成本分析报告。通过部门的成本分析报告，企业管理人员可以识别成本增长趋势、部门成本对比，能制定更好的成本管理方案。
- 集群资源视角成本洞察：以成本运维人员的角度，着重呈现CCE集群内部从命名空间、应用、节点池等多个维度的集群成本开销和资源使用状况，进而识别可优化的应用。

成本洞察关键能力

- 丰富的容器成本覆盖范围：支持成本分析的费用包括CCE集群管理费用、CCE集群关联的ECS和EVS资源费用。
- 基于计费账单的精准成本计算：使用真实账单进行成本分摊计算，精准统计集群成本。
- 灵活的成本分摊策略：支持集群、命名空间、节点池、应用等多种维度的成本可视化与成本分摊策略。
- 支持长期的成本数据存储与检索：最大支持长达2年的成本分析。
- 分钟级负载计费，轻松应对快速弹性场景：针对应用快速弹性场景，支持分钟级的负载发现与计费能力，让所有成本无一遗漏。

约束与限制

- PVC存储费用：当前只统计云硬盘（EVS）类型存储费用，不支持对象存储类型（OBS）、本地持久卷、文件存储类型（SFS）、极速文件存储（SFS Turbo）。
- 节点成本按照CPU、内存进行成本拆分，暂不支持GPU、NPU等异构资源的拆分。如GPU类型的节点在拆分时，会出现CPU核时单价偏高。
- 开通成本洞察后需要等待2天时间，才能显示分析结果。
- 成本洞察以天为粒度呈现成本分析结果。

11.2.2 成本计算模型

工作负载成本计算原理

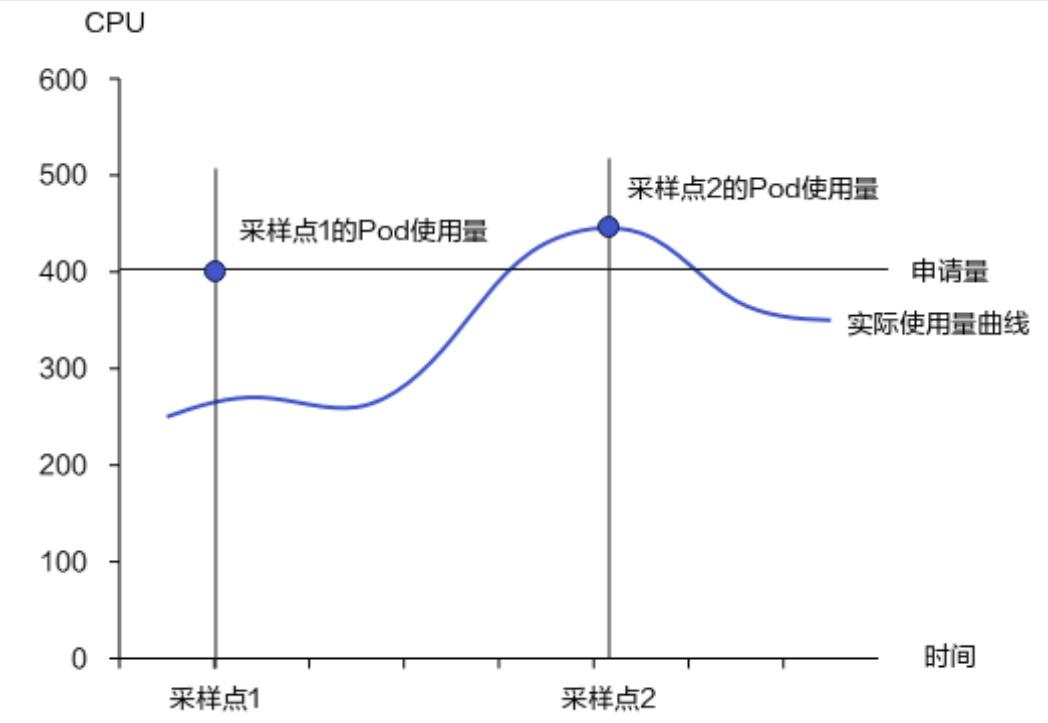
工作负载成本是由Pod成本聚合而成。

- Pod成本：使用监控指标和实际账单作为输入，通过CPU、内存使用量占整体节点资源比例计算出来的成本，结合Pod关联PVC存储的成本。

$$Pod成本 = \frac{\max(Pod\ CPU\ Request,\ Pod\ CPU\ Usage)}{Node\ CPU\ Capacity} \times Node花费 \times CPU占比 + \frac{\max(Pod\ Memory\ Request,\ Pod\ Memory\ Usage)}{Node\ Memory\ Capacity} \times Node价格 \times Memory占比 + Pod关联PVC成本$$

计算过程中，Pod的使用量为当前采样时刻下申请量（Request）和实际使用量（Real Used）中的最大值。如下图：

图 11-1 工作负载成本计算原理



如：采样点1，Request CPU > Real Used CPU，Pod使用量取Request CPU值
采样点2，Request CPU < Real Used CPU，Pod使用量取Real Used CPU值

- 工作负载成本：该工作负载中所有Pod的成本总和。

$$工作负载成本 = \sum_{工作负载}^{all} Pod成本$$

命名空间成本计算原理

命名空间成本是由工作负载成本聚合而成，其计算公式为该命名空间下所有工作负载的成本总和。

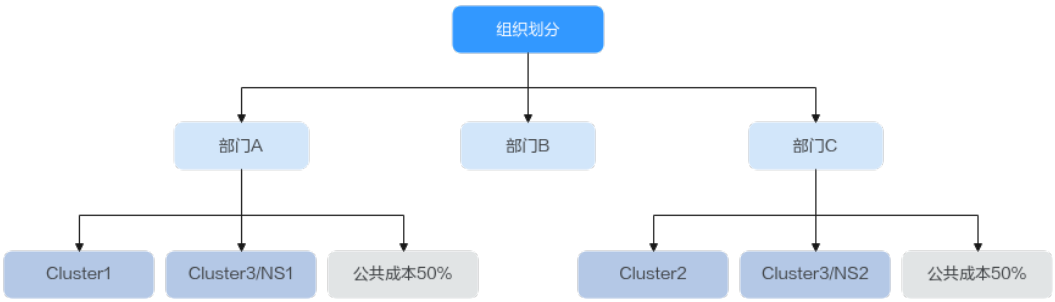
$$命名空间成本 = \sum_{命名空间}^{all} 工作负载成本$$

部门成本计算原理

部门是一种逻辑的成本归结单元，用于将不同的集群、命名空间的成本聚合分析。为贴合实际的业务场景，一般会按照实际业务部门设立该成本单元，并关联业务部门使用的集群或者命名空间。

单个集群的成本由业务命名空间成本、未被分配的空闲成本、集群管理成本（CCE集群Master成本+系统命名空间成本）组成。其中未被分配空闲成本以及集群管理成本，被定义为公共成本。当部门按照命名空间进行设置时，需要关联业务命名空间，并设置公共成本的分摊比例。

图 11-2 部门成本计算示例



示例中，Cluster1是部门A的专属集群，Cluster2是部门C的专属集群，而Cluster3为部门A和部门C的共享集群，且命名空间NS1属于部门A，NS2属于部门C。在计算部门成本时，可按照如上图将对应集群或命名空间划入对应部门，并将Cluster3中产生的公共成本设置分摊比例，在部门A和部门C中进行分摊。

计算过程常见问题

- Pod Request和Used的资源，应该根据哪个来估算成本，进行计算Pod的费用，Pod的CPU、内存使用量等资源指标是动态变化的，如何做到准确的估算?

  在计算成本时的Pod使用量取值为Pod申请量（Request）和实际使用量（used）中的最大值。基于普罗监控数据，可以清晰识别分钟级别的应用资源，进行成本计算。

- 节点中没有被分配的空闲成本，是如何处理的?

  节点中的空闲成本不会被分摊到工作负载或者命名空间成本中，可以作为集群的公共成本分摊到部门。空闲成本在各个部门的分摊比例支持设置。

11.2.3 开通成本洞察

成本洞察基于真实账单和集群资源用量统计数据，通过自研的成本画像算法进行成本拆分，提供以部门、集群、命名空间、应用等维度的成本画像。成本洞察能够帮助成本管理人员分析集群成本开销、资源使用状况，识别资源浪费，为下一步的成本优化提供输入。

本文主要介绍如何开通成本洞察功能。

- **开通Region视角的成本洞察**
- **开通单集群视角成本洞察**

须知

- 开通成本洞察需要安装云原生监控插件，插件采集的监控指标将上报至AOM实例，AOM针对基础指标免费，自定义指标由AOM服务收费，具体请参考**价格详情**。成本洞察能力使用的监控指标均为基础指标。
- 开通成本洞察会在“亚太-新加坡”区域创建一个OBS桶，用来存放从费用中心订阅的基础账单数据。OBS产生的费用详情请参考**价格详情**。

前提条件

开通云原生成本治理前，用户需要使用具有admin用户组的账户完成对CCE及其依赖服务的委托授权。

授权方式：用户进入云原生成本治理页面，界面会自动弹出“确认授权”页面，用户单击“确认授权”按钮后系统自动完成授权。所授予的权限类型请参考**表11-1**。

表 11-1 资源权限

授权类型	权限名称	描述
CCE	IAM ReadOnlyAccess	云原生成本治理获得该权限后，支持子用户进行访问云原生成本治理，因此需要获得该权限。
CCE	Tenant Guest	云原生成本治理支持对集群关联的 OBS、DNS 等全局资源配置进行检查，提前发现配置问题，因此需要获得该权限。
CCE	CCE Administrator	云原生成本治理在运行过程中需要访问 CCE 获取集群、节点、工作负载等信息，以此来检测对应资源的健康状态，因此需要获得该权限。
CCE	AOM Administrator	云原生成本治理在运行过程中需要访问 AOM 获取监控指标信息，因此需要获得该权限。
CCE	OBS Administrator	云原生成本治理的成本可视化需要结合实际账单，账单信息需要放入用户OBS桶中，需要访问用户OBS桶，因此需要获得该权限。
CCE	CBC Finance	云原生成本治理需要帮用户订阅用户的CBC账单信息。订阅后，CBC会定期将用户账单信息放入用户OBS桶中，供云原生成本治理使用，因此需要获得该权限。
AOM	DMS UserAccess	AOM 支持用户通过 DMS 获取数据订阅的功能，因此需要获得该权限。
AOM	ECS CommonOperations	AOM 支持通过在 ECS 上安装 UniAgent 和 ICAgent 获取系统指标、日志数据，因此需要获得该权限。
AOM	CES ReadOnlyAccess	AOM 支持从 CES 同步监控指标数据，因此需要获得该权限。

授权类型	权限名称	描述
AOM	CCE FullAccess	AOM 支持从 CCE 同步容器监控指标数据，因此需要获得访问权限。
AOM	RMS ReadOnlyAccess	AOM 的 CMDB 支持管理云服务实例数据，因此需要获得该权限。
AOM	ECS ReadOnlyAccess	AOM支持通过在 ECS 上安装 UniAgent 和 ICAgent 获取系统指标、日志数据，因此需要获得该权限。
AOM	LTS FullAccess	AOM 支持访问 LTS 数据，因此需要获得该权限。
AOM	CCI FullAccess	AOM 支持从 CCI 同步容器监控指标数据，因此需要获得该权限。

约束与限制

- 集群版本仅支持v1.17及以上。
- 使用成本治理前，用户需要使用具有admin用户组的账户完成对CCE及其依赖服务的委托授权。完成授权后，拥有CCE Administrator角色或CCE FullAccess权限的用户可进行成本治理所有操作。

开通 Region 视角的成本洞察

步骤1 登录CCE控制台，单击左侧导航栏中的“云原生成本治理”。

图 11-3 云原生成本治理

步骤2 单击“立即开通”选择要开通的集群后，单击“确认开通”。

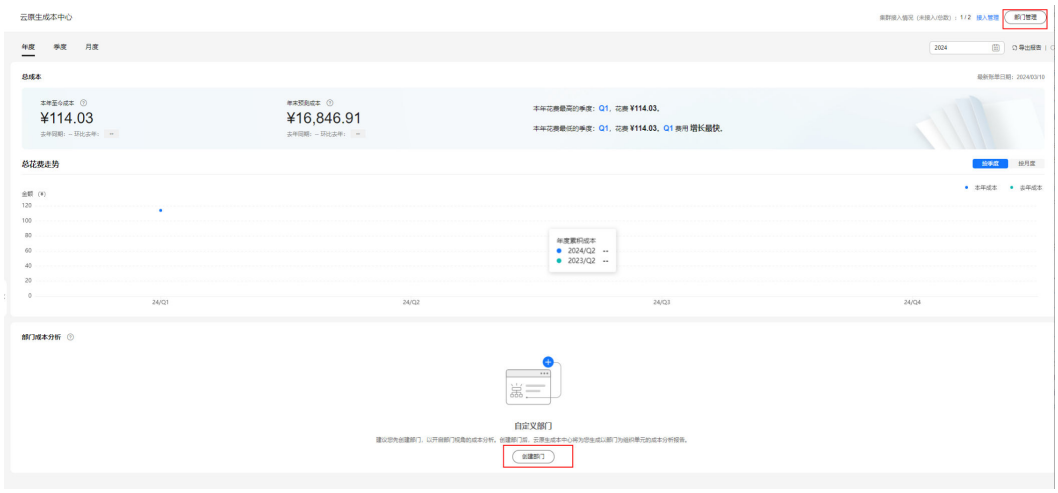
开通过程中系统将自动执行如下步骤：安装云原生监控插件、成本标签激活、创建默认租户OBS桶、订阅账单数据。等待3-5分钟，即可进入洞察界面。

1. 安装云原生监控插件：为成本洞察功能提供基础监控数据。
2. 成本标签激活：成本标签激活后费用中心导出的账单会增加集群的标签，成本洞察后台将按照集群进行分类。该步骤完成后，可在云原生成本治理的成本标签界面，看到CCE-Cluster-ID、CCE-Dynamic-Provisioning-Node标签被激活。
3. 创建默认租户OBS桶：创建名称为cce-cost-{region}-{domain_id}的默认OBS桶，该OBS桶用来存储从费用中心导出的账单数据。
4. 订阅账单数据：订阅账单后，费用中心会定期将账单推送到OBS桶中，供成本洞察使用。

图 11-4 开通集群

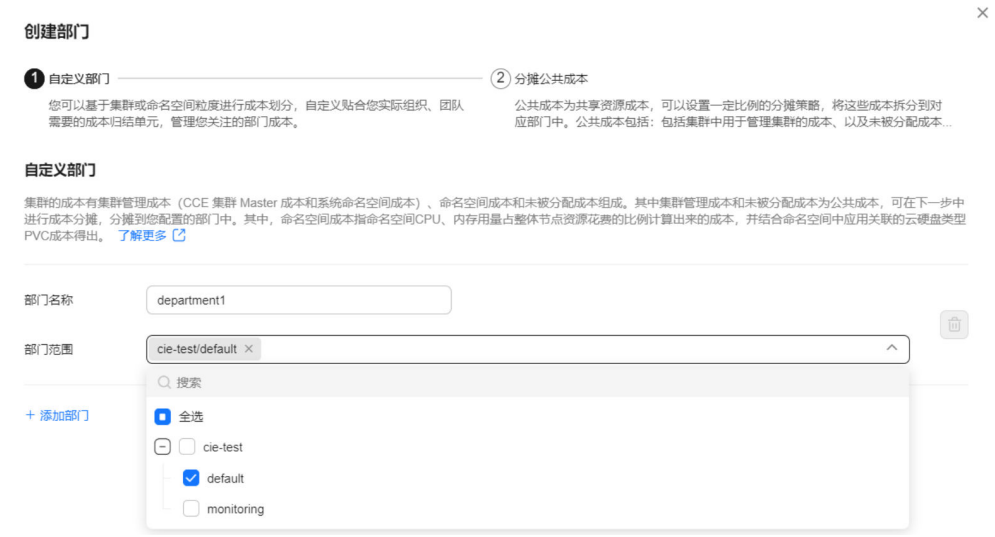
步骤3 （可选）单击“创建部门”，进行部门的配置。部门的配置包含如下步骤：

图 11-5 创建部门



1. 自定义部门：为贴合实际的业务场景，一般会按照实际业务部门设立该成本单元，并关联业务部门使用的集群或者命名空间。
   - 部门名称：建议使用实际的业务部门名称；
   - 部门范围：该部门使用的集群或者命名空间。如下按照命名空间粒度，配置部门department1、department2。

图 11-6 自定义部门

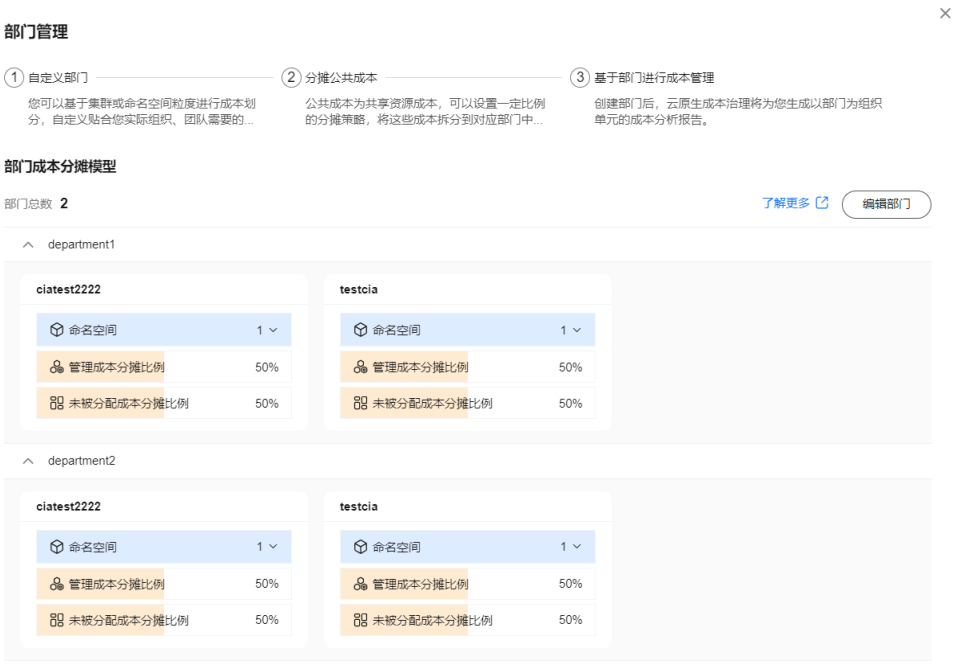


2. 分摊公共成本：将集群中的公共成本分摊到部门。默认集群中的管理成本和未被分配成本，在其关联的部门中进行平均分摊。支持修改分摊比例。

图 11-7 分摊公共成本

3. 基于部门进行成本管理：部门配置完成后，单击“提交配置”，便可以在部门管理界面看到配置的结果。部门配置结果如下：

图 11-8 部门配置



----结束

开通单集群视角的成本洞察

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏中的“云原生成本治理 > 成本洞察”。

步骤3 单击“立即开通”，同样系统会自动执行安装云原生监控插件、成本标签激活、创建默认租户OBS桶、订阅账单数据这几个步骤。等待3-5分钟，即可进入洞察界面。

图 11-9 开通成本洞察

----结束

11.2.4 Region 视角的成本洞察

为站在企业管理人员角色的角度，呈现整体Region级别成本分析报表。从云原生角度出发，用户可以灵活去组织成本。可以自由按照集群、命名空间粒度去归结成本到部门，形成部门的成本分析报告。并支持成本报表导出功能。

前提条件

- 已开通成本洞察功能

约束与限制

- 由于实际账单的获取存在两天时间延迟，开通成本洞察后，成本洞察成本数据会延迟2天显示。
- 使用成本洞察期间，要保证云原生监控插件运行正常，否则影响成本洞察中命名空间、工作负载、节点池等相关视图的呈现。

接入管理

步骤1 登录CCE控制台，单击左侧导航栏中的“云原生成本治理”。

图 11-10 云原生成本治理

步骤2 单击“接入管理”，查看集群接入情况，并对剩余未接入集群进行接入。

图 11-11 接入集群

步骤3 选中需要接入的集群，单击“批量接入”，可批量将选中的集群进行开通。开通成功后，可以在列表中查看接入状态。集群首次接入云原生成本治理，需要等待2天时间，才可以看到相应成本数据。

图 11-12 批量接入

图 11-13 接入集群管理

----结束

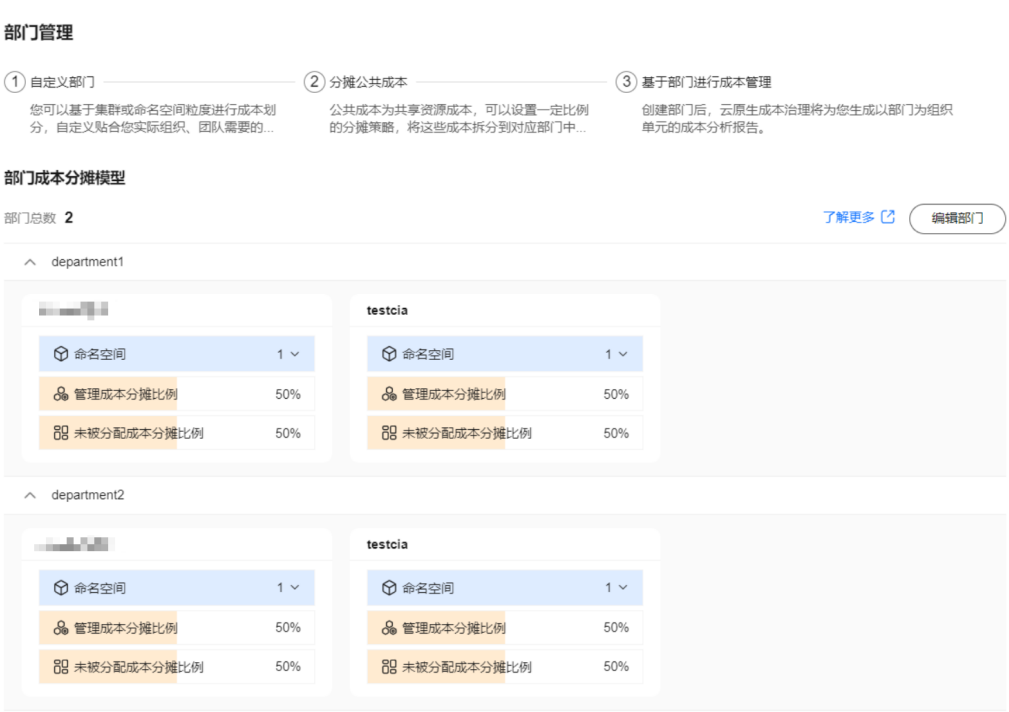
部门管理

步骤1 登录CCE控制台，单击左侧导航栏中的“云原生成本治理”。

图 11-14 云原生成本治理

步骤2 单击“部门管理”，进行部门配置查看。

图 11-15 部门管理

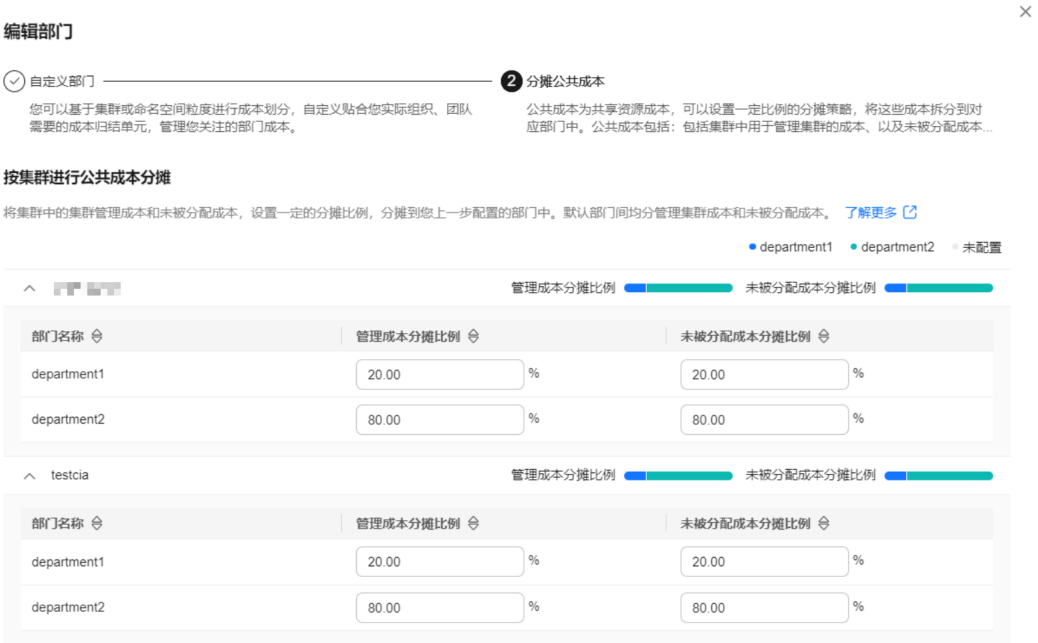


步骤3 单击“编辑部门”，修改自定义部门配置。

图 11-16 编辑部门

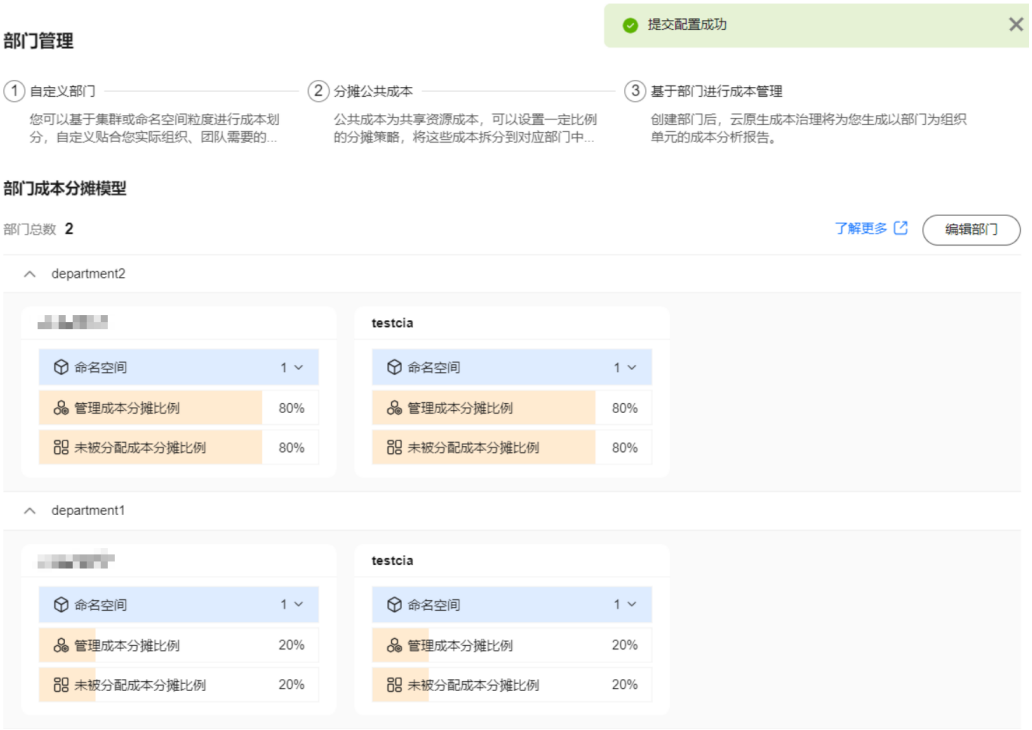
步骤4 单击“下一步”，修改公共成本分摊到部门的比例。

图 11-17 修改公共成本分摊



步骤5 单击“提交配置”，便可以在部门管理界面看到配置的结果。

图 11-18 提交配置



步骤6 配置完成后，关闭部门管理界面，即可在云原生成本治理查看相应部门的成本分析报告。

图 11-19 查看成本分析报告

----结束

使用成本洞察

开通后，进入成本洞察界面，查看已接入集群的成本状况。您可以切换页签，进行年度、季度、月度成本分析报表查看。

图 11-20 查看成本状况

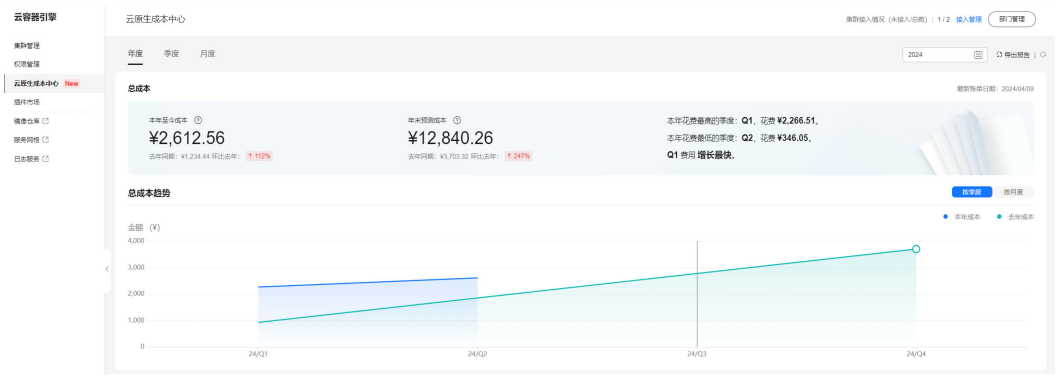


表 11-2 界面功能说明

名称	所属报告	说明
本年至今成本（去年同期、环比去年）	年度	本年至今：本年开始到最新账单日期产生的成本 去年同期：对比去年相同日期产生的成本 环比去年：（本年至今成本 - 去年同期成本）/ 去年同期成本
年末预测成本（去年同期、环比去年）	年度	年末预测成本：到本年年末预计产生的总成本开销 去年同期：去年整年产生的成本 环比去年：（年末预测成本 - 去年同期成本）/ 去年同期成本
本季至今成本（上季度同期、环比上季度）	季度	本季至今成本：本季开始到最新账单日期产生的成本 上季度同期：对比上季度相同日期产生的成本 环比上季度：（本季至今成本 - 上季度同期成本）/ 上季度同期成本
季末预测成本（上季度同期、环比上季度）	季度	季末预测成本：到本季度末，整季预估产生的总成本开销 上季度同期：上季度整季产生的成本 环比上季度：（季末预测成本 - 上季度同期成本）/ 上季度同期成本
本月至今成本（上月同期、环比上月）	月度	本月至今成本：本月开始到最新账单日期产生的成本 上月同期：对比上月相同日期产生的成本 环比上月：（本月至今成本 - 上月同期成本）/ 上月同期成本

名称	所属报告	说明
月末预测成本（上月同期、环比上月）	月度	月末预测成本：到本月月末，整月预估产生的总成本开销 上月同期：上月整月产生的成本 环比上月：（月末预测成本 - 上月同期成本）/ 上月同期成本
总成本趋势	年度、季度、月度	呈现本年、本季度、本月成本详情，以及分别和上年、上季、上月的成本对比趋势

部门成本分析

在部门成本分析模块查看部门成本分析报告。

图 11-21 查看部门成本分析

表 11-3 部门成本明细报表功能说明

名称	所属报告	说明
部门名称	年度、季度、月度	部门配置名称
本年至今成本	年度	本年开始到最新账单日期产生的成本
去年同期成本	年度	对比去年相同日期产生的成本
去年总成本	年度	去年整年产生的成本
对比去年同期增长值	年度	本年至今成本 - 去年同期成本
对比去年同期增长率	年度	（本年至今成本 - 去年同期成本）/ 去年同期成本

名称	所属报告	说明
本年CPU成本	年度	本年开始到最新账单日期产生的CPU成本
本年CPU成本占部门整体成本百分比	年度	本年CPU成本 / 本年至今成本（部门成本）
本年至今相比去年同期CPU增长成本	年度	本年CPU成本 - 去年同期CPU成本
本季至今成本	季度	本季开始到最新账单日期产生的成本
上季度同期成本	季度	对比上季度相同日期产生的成本
上季度总成本	季度	上季度整季产生的成本
对比上季度同期增长值	季度	本季至今成本 - 上季度同期成本
对比上季度同期增长率	季度	（本季至今成本 - 上季度同期成本）/ 上季度同期成本
本季度CPU成本	季度	本季开始到最新账单日期产生的CPU成本
本季度CPU成本占部门整体成本百分比	季度	本季CPU成本 / 本季至今成本（部门成本）
本季度相比上季度同期CPU增长成本	季度	本季CPU成本 - 上季度CPU成本
本月至今成本	月度	本月开始到最新账单日期产生的成本
上月同期成本	月度	对比上月相同日期产生的成本
上月总成本	月度	上月整月产生的成本
对比上月同期增长值	月度	本月至今成本 - 上月同期成本
对比上月同期增长率	月度	（本月至今成本 - 上月同期成本）/ 上月同期成本
本月CPU成本	月度	本月开始到最新账单日期产生的CPU成本
本月CPU成本占部门整体成本百分比	月度	本月CPU成本 / 本月至今成本（部门成本）
本月相比上月同期CPU增长成本	月度	本月CPU成本 - 上月CPU成本

11.2.5 单部门视角的成本洞察

单部门视角成本洞察，提供单一部门的成本分析报告。在部门成本分析模块，进行整体部门成本状况查看，并可单击部门列表中的某一部门，进行单部门的详细成本分析。

前提条件

- 已开通成本洞察功能
- 已完成部门配置

约束与限制

- 由于实际账单的获取存在两天时间延迟，开通成本洞察后，成本洞察成本数据会延迟2天显示。
- 使用成本洞察期间，要保证云原生监控插件运行正常，否则影响成本洞察中命名空间、工作负载、节点池等相关视图的呈现。

操作入口

步骤1 登录CCE控制台，单击左侧导航栏中的“云原生成本治理”。

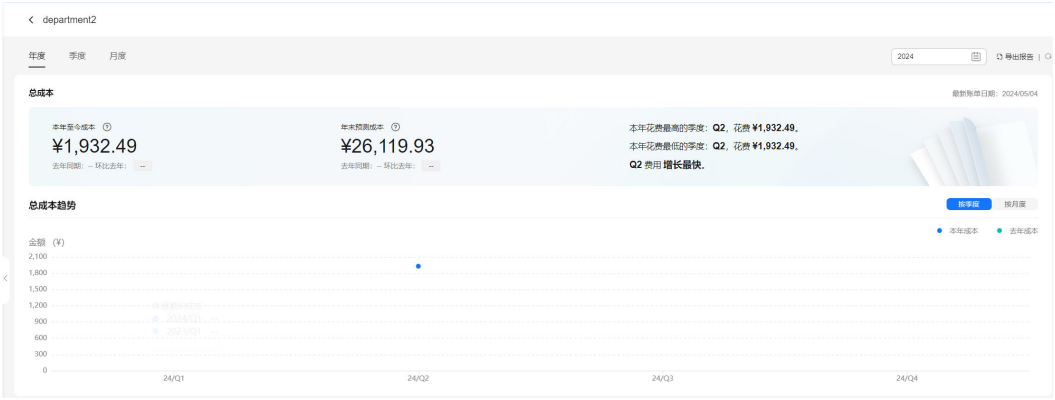
图 11-22 云原生成本治理

步骤2 查看部门分析模块。

图 11-23 查看部门分析

步骤3 单击部门成本明细中的某一部门名称，进入对应单部门视角。

图 11-24 查看单部门成本



----结束

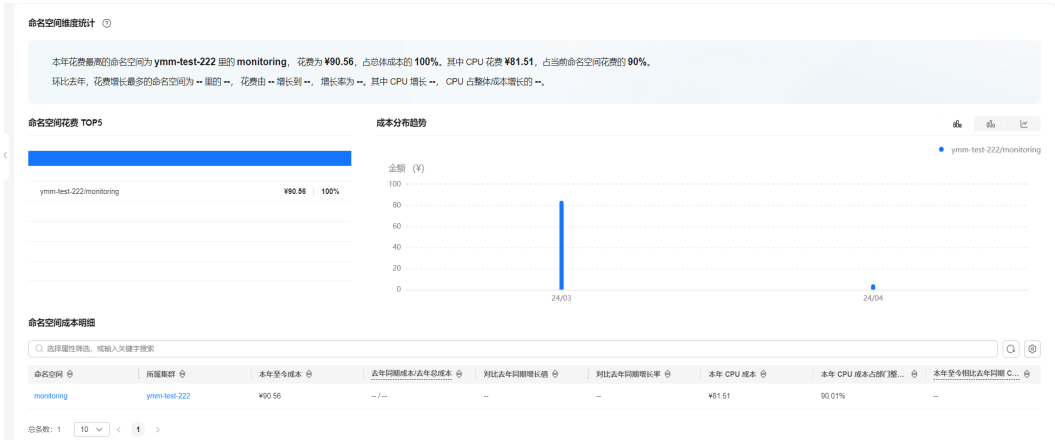
表 11-4 Region 视角总成本功能说明

名称	所属报告	说明
本年至今成本（去年同期、环比去年）	年度	本年至今：当前部门本年开始到最新账单日期产生的成本 去年同期：当前部门对比去年相同日期产生的成本 环比去年：（本年至今成本 - 去年同期成本）/ 去年同期成本
年末预测成本（去年同期、环比去年）	年度	年末预测成本：当前部门到本年年末预计产生的总成本开销 去年同期：当前部门去年整年产生的成本 环比去年：（年末预测成本 - 去年同期成本）/ 去年同期成本
本季至今成本（上季度同期、环比上季度）	季度	本季至今成本：当前部门本季开始到最新账单日期产生的成本 上季度同期：当前部门对比上季度相同日期产生的成本 环比上季度：（本季至今成本 - 上季度同期成本）/ 上季度同期成本
季末预测成本（上季度同期、环比上季度）	季度	季末预测成本：当前部门到本季度末，整季预估产生的总成本开销 上季度同期：当前部门上季度整季产生的成本 环比上季度：（季末预测成本 - 上季度同期成本）/ 上季度同期成本

名称	所属报告	说明
本月至今成本（上月同期、环比上月）	月度	本月至今成本：当前部门本月开始到最新账单日期产生的成本 上月同期：当前部门对比上月相同日期产生的成本 环比上月：（本月至今成本 - 上月同期成本）/ 上月同期成本
月末预测成本（上月同期、环比上月）	月度	月末预测成本：当前部门到本月月末，整月预估产生的总成本开销 上月同期：当前部门上月整月产生的成本 环比上月：（月末预测成本 - 上月同期成本）/ 上月同期成本
成本趋势	年度、季度、月度	呈现本年、本季度、本月成本详情，以及分别和上年、上季、上月的成本对比趋势

集群维度统计、命名空间维度统计对应部门配置中关联的集群、命名空间的成本统计，不包含部门中的公共成本。如下示例中，部门中只配置了命名空间的划分方式，只呈现命名空间成本统计。

图 11-25 查看命名空间分析



11.2.6 单集群视角的成本洞察

单集群视角的成本洞察是为了帮助成本运维人员深入集群内部，从命名空间、应用、节点池等多个维度分析集群成本开销、资源使用状况，进而提供成本优化的依据。当前支持集群维度和命名空间维度两个视角的成本洞察。

前提条件

- 已开通成本洞察功能

约束与限制

- 由于实际账单的获取存在两天时间延迟，开通成本洞察后，成本洞察成本数据会延迟2天显示。
- 使用成本洞察期间，需要保证云原生监控插件运行正常，否则影响成本洞察中命名空间、工作负载、节点池等相关视图的呈现。

操作入口

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 单击左侧导航栏中的“云原生成本治理 > 成本洞察”。

步骤3 在洞察界面，进行成本优化分析。

图 11-26 单集群视角的成本洞察

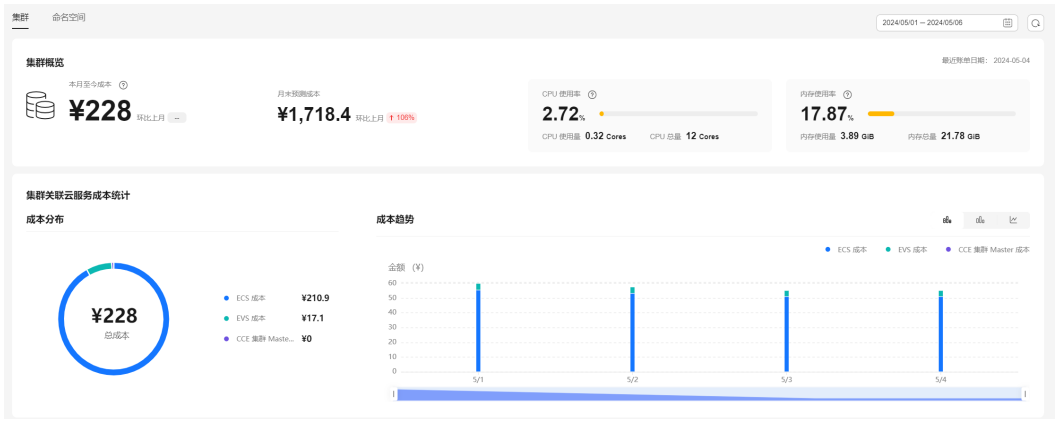
----结束

集群维度

集群维度是单集群视角成本洞察的总览界面，涵盖了命名空间、工作负载、节点池等维度的成本开销和资源消耗情况，帮助运维人员识别成本开销大、资源利用率低的应用。

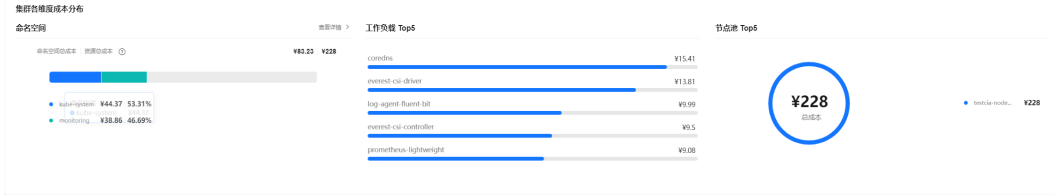
您可以在右上角进行时间过滤。

图 11-27 单集群视角的成本总览



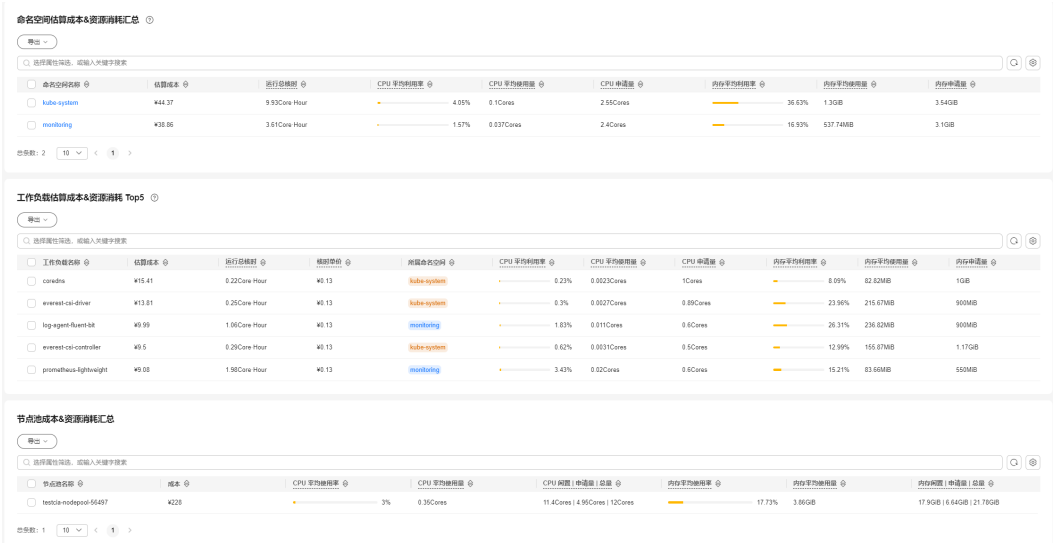
名称	含义
本月至今成本 环比上月	本月至今：集群从月初到最新账单日期产生的成本。如果本月开通服务，则为开通时到最新账单日期产生的成本。 环比上月：（本月至今成本 - 上月同期成本）/ 上月同期成本
月末预测成本 环比上月	月末预测成本：到本月月末，整月预估产生的总成本开销 环比上月：（月末预测成本 - 上月整月成本）/ 上月整月成本
CPU使用率 CPU使用量 CPU总量	CPU使用率：当前时间集群所有节点的 CPU 平均使用率 CPU 使用率 = 所有节点 CPU 使用总量 / 所有节点 CPU 总量 * 100%
内存使用率 内存使用量 内存总量	内存使用率：当前时间集群所有节点的内存平均使用率 内存使用率 = 所有节点内存使用量 / 所有节点内存总量 * 100%
成本分布	选中时间内成本分布。当前资源涵盖范围有：ECS成本、EVS成本、CCE集群管理成本
成本趋势	每天的成本分布趋势。从中可以看出，集群内成本开销趋势，可识别花费较高的资源

图 11-28 集群各维度成本分析



名称	含义
命名空间	命名空间总成本：按命名空间聚合工作负载的成本，包括 CPU 成本（ECS）、内存成本（ECS）、EVS 成本。资源总成本：为计算资源总成本，包括集群所有 ECS 成本，以及 EVS 成本。 资源总成本 = 命名空间总成本 + 未被分配资源成本 如果灰色区域占用过大，则表示未被使用的资源过多，有资源浪费的现象。
工作负载 Top5	成本开销Top5的工作负载，便于识别大应用
节点池 Top5	成本开销Top5的节点池

图 11-29 成本估算&资源消耗汇总



功能	名称	含义
命名空间估算成本&资源消耗汇总	命名空间名称	命名空间名称
	估算成本	命名空间成本，计算方式：根据命名空间资源用量（CPU、Memory）占整体节点资源比例计算出来的成本，结合命名空间中工作负载关联存储的成本
	运行总核时	所选时间周期内，命名空间总消耗的核时资源数
	CPU平均利用率	所选时间周期内，命名空间的 CPU 平均利用率。 CPU 利用率 = CPU 使用量 / CPU 申请量 * 100%
	CPU平均使用量	所选时间周期内，命名空间的 CPU 平均使用量
	CPU申请量	所选时间周期最近账单日期，命名空间下工作负载的CPU申请量的累加

功能	名称	含义
	内存平均利用率	所选时间周期内，命名空间的内存平均利用率。内存利用率 = 内存使用量 / 内存申请量 * 100%
	内存平均使用量	所选时间周期内，命名空间的内存平均使用量
	内存申请量	所选时间周期最近账单日期，命名空间下工作负载的内存申请量的累加
工作负载估算成本&资源消耗Top5	工作负载名称	工作负载名称
	估算成本	工作负载成本，计算方式：根据工作负载资源用量（CPU、Memory）占整体节点资源比例计算出来的成本，结合工作负载关联存储的成本
	运行总核时	所选时间周期内，工作负载总消耗的核时资源数
	核时单价	CPU 每核每小时价格。 用来指导您工作负载所在节点对应的机型的CPU核时单价。如果工作负载或者命名空间对应核时单价很贵，则可以通过变更节点类型等方式进行降本增效。
	所属命名空间	工作负载所在命名空间
	CPU平均利用率	所选时间周期内，工作负载的 CPU 平均利用率，用来指导您工作负载的CPU资源使用效率。 CPU 利用率 = CPU 使用量 / CPU 申请量 * 100%
	CPU平均使用量	所选时间周期内，工作负载的 CPU 平均使用量
	CPU申请量	所选时间周期最近账单日期，工作负载的 CPU Request
	内存平均利用率	所选时间周期内，工作负载的内存平均利用率，用来指导您工作负载的内存资源使用效率。 内存利用率 = 内存使用量 / 内存申请量 * 100%
	内存平均使用量	所选时间周期内，工作负载的内存平均使用量
	内存申请量	所选时间周期最近账单日期，工作负载的内存 Request
节点池成本&资源消耗汇总	节点池名称	节点池名称
	成本	所选时间周期内，节点池中节点的成本开销
	CPU平均使用率	所选时间周期内，节点池的平均 CPU 使用率。 CPU 使用率 = 节点池中节点 CPU 使用总量 / 节点池中节点 CPU 总量 * 100%

功能	名称	含义
	CPU平均使用量	所选时间周期内，节点池的 CPU 平均使用量
	CPU闲置|申请量|总量	CPU 闲置：所选时间周期最后一天，节点池中节点的闲置 CPU 之和 CPU 申请量：所选时间周期最后一天，节点池中节点的申请 CPU 之和 CPU 总量：所选时间周期最后一天，节点池中节点的 CPU 总量之和
	内存平均使用率	所选时间周期内，节点池的平均内存使用率。 内存使用率 = 节点池中节点内存使用总量 / 节点池中节点内存总量
	内存平均使用量	所选时间周期内，节点池的内存平均使用量
	内存闲置|申请量|总量	内存闲置：所选时间周期最后一天，节点池中节点的闲置内存之和 内存申请量：所选时间周期最后一天，节点池中节点的申请内存之和 内存总量：所选时间周期最后一天，节点池中节点的内存总量之和

命名空间维度

命名空间维度支持对选中的命名空间、以及命名空间下的工作负载进行成本优化分析，识别开销较大，利用率较低的工作负载进行优化调整。

图 11-30 命名空间维度的成本总览

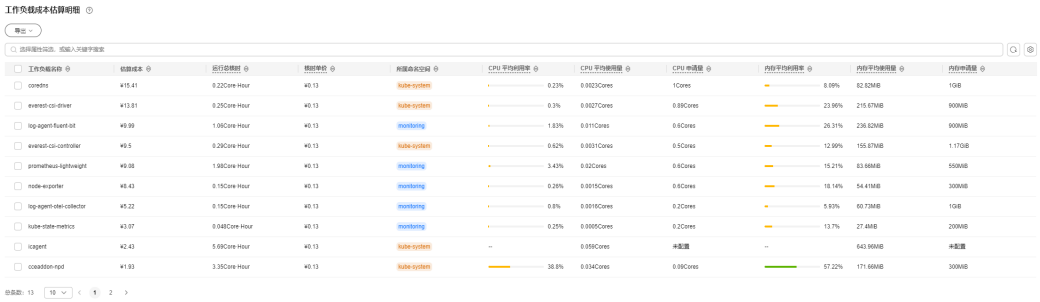
名称	含义
本月至今成本 环比上月	本月至今：选择命名空间从月初到最新账单日期产生的成本。如果本月开通服务，则为开通时到最新账单日期产生的成本。 环比上月：（本月至今成本 - 上月同期成本）/ 上月同期成本

名称	含义
月末预测成本 环比上月	月末预测成本：到本月月末，选中命名空间整月预估产生的总成本开销 环比上月：（月末预测成本 - 上月整月成本）/ 上月整月成本
CPU利用率 CPU使用量 CPU申请量	CPU利用率：当前时间所选命名空间 CPU 的平均利用率。 CPU 利用率 = 所选命名空间 CPU 使用总量 / 所选命名空间 CPU 申请总量 * 100%
内存利用率 内存使用量 内存申请量	当前时间所选命名空间内存的平均利用率。 内存利用率 = 所选命名空间内存使用总量 / 所选命名空间内存申请总量 * 100%
成本分布	选中时间内选中命名空间Top5的成本分布
成本趋势	选中命名空间每天的成本分布趋势，可以识别成本较高的命名空间。

图 11-31 命名空间成本构成统计

名称	含义
资源成本构成	所选命名空间、所选时间周期的成本构成。成本由CPU成本、内存成本、存储成本组成
核时单价走势	所选命名空间下工作负载所在节点的平均核时单价走势。
Top5 运行总核时	所选命名空间、所选时间周期的 Top5 命名空间总消耗的核时资源数。
CPU	所选命名空间的CPU使用量、申请量、限制量走势。
内存	所选命名空间的内存使用量、申请量、限制量走势。
资源利用率	所选命名空间的CPU利用率、内存利用率走势。

图 11-32 工作负载成本估计明细



名称	含义
工作负载名称	工作负载名称
估算成本	根据工作负载资源用量（CPU、Memory）占整体节点资源比例计算出来的成本，并结合了工作负载关联存储的成本构成
运行总核时	所选时间周期内，工作负载总消耗的核时资源数。用于指导资源使用情况。
核时单价	CPU 每核每小时价格，体现了工作负载所在节点的单位价钱，可用于指导进行节点机型优化。
所属命名空间	工作负载归属命名空间
CPU平均利用率	所选时间周期内，工作负载的 CPU 平均利用率。CPU 利用率 = CPU 使用量 / CPU 申请量 * 100%
CPU平均使用量	所选时间周期内，工作负载的 CPU 平均使用量
CPU申请量	所选时间周期最后一天，工作负载的 CPU Request
内存平均利用率	所选时间周期内，工作负载的内存平均利用率。内存利用率 = 内存使用量 / 内存申请量 * 100%
内存平均使用量	所选时间周期内，工作负载的内存平均使用量
内存申请量	所选时间周期最后一天，工作负载的内存 Request

12 命名空间

12.1 创建命名空间

操作场景

命名空间（Namespace）是对一组资源和对象的抽象整合。在同一个集群内可创建不同的命名空间，不同命名空间中的数据彼此隔离。使得它们既可以共享同一个集群的服务，也能够互不干扰。

例如可以将开发环境、测试环境的业务分别放在不同的命名空间。

前提条件

至少已创建一个集群。

约束与限制

每个命名空间下，创建的服务数量不能超过6000个。此处的服务对应kubernetes的service资源，即工作负载所添加的服务。

命名空间类别

命名空间按创建类型分为两大类：集群默认创建的命名空间、用户创建的命名空间。

- 集群默认创建的命名空间：集群在启动时会默认创建default、kube-public、kube-system、kube-node-lease命名空间。
  - default：所有未指定Namespace的对象都会被分配在default命名空间。
  - kube-public：此命名空间下的资源可以被所有人访问（包括未认证用户），使集群中的某些资源可以在整个集群范围内可见可读。 该命名空间为Kubernetes预留的命名空间，其公共属性只是一种约定而并非要求。
  - kube-system：所有由Kubernetes系统创建的资源都处于这个命名空间。
  - kube-node-lease：每个节点在该命名空间中都有一个关联的“Lease”对象，该对象由节点定期更新。NodeStatus和NodeLease都被视为来自节点的心跳，在v1.13之前的版本中，节点的心跳只有NodeStatus，NodeLease特性从v1.13开始引入。NodeLease比NodeStatus更轻量级，该特性在集群规模扩展性和性能上有明显提升。

- 用户创建的命名空间：用户可以按照需要创建命名空间，例如开发环境、联调环境和测试环境分别创建对应的命名空间。或者按照不同的业务创建对应的命名空间，例如系统若分为登录和游戏服务，可以分别创建对应命名空间。

创建命名空间

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“命名空间”，在右上角单击“创建命名空间”。

步骤3 参照表12-1设置命名空间参数。

表 12-1 命名空间基本信息

参数	参数说明
名称	新建命名空间的名称，命名必须唯一。
描述	输入对命名空间的描述信息。
配额管理	资源配额可以限制命名空间下的资源使用，进而支持以命名空间为粒度的资源划分。 **须知** **建议根据需要在命名空间中设置资源配额，避免因资源过载导致集群或节点异常。** 例如：在集群中每个节点可以创建的实例（Pod）数默认为110个，如果您创建的是50节点规格的集群，则最多可以创建5500个实例。因此，您可以在命名空间中自行设置资源配额以确保所有命名空间内的实例总数不超过5500个，以避免资源过载。 请输入整型数值，不输入表示不限制该资源的使用。 若您需要限制CPU或内存的配额，则创建工作负载时必须指定CPU或内存请求值。

步骤4 配置完成后，单击“确定”。

----结束

使用 kubectl 创建 Namespace

使用如下方式定义Namespace。

```
apiVersion: v1
kind: Namespace
metadata:
  name: custom-namespace
```

使用kubectl命令创建。

```
$ kubectl create -f custom-namespace.yaml
namespace/custom-namespace created
```

您还可以使用kubectl create namespace命令创建。

```
$ kubectl create namespace custom-namespace
namespace/custom-namespace created
```


12.2 管理命名空间

使用命名空间

- 创建工作负载时，您可以选择对应的命名空间，实现资源或租户的隔离。
- 查询工作负载时，选择对应的命名空间，查看对应命名空间下的所有工作负载。

命名空间使用实践

- **按照不同环境划分命名空间**

  一般情况下，工作负载发布会经历开发环境、联调环境、测试环境，最后到生产环境的过程。这个过程中不同环境部署的工作负载相同，只是在逻辑上进行了定义。分为两种做法：

  - 分别创建不同集群。

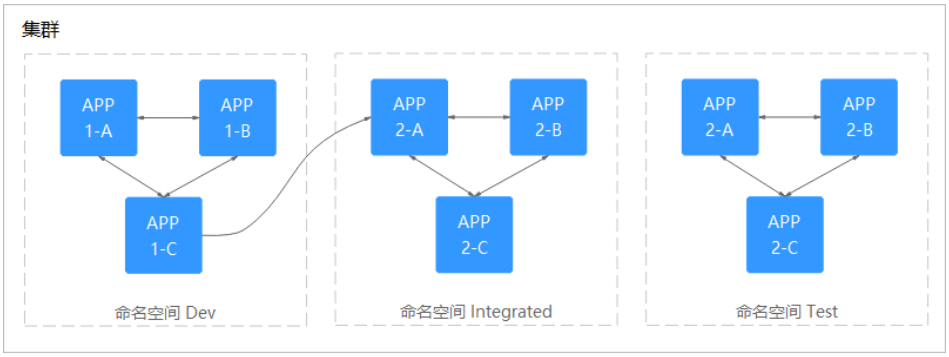
    不同集群之间，资源不能共享。同时，不同环境中的服务互访需要通过负载均衡才能实现。

  - 不同环境创建对应命名空间。

    同个命名空间下，通过服务名称（Service name）可直接访问。跨命名空间的可以通过服务名称、命名空间名称访问。

    例如下图，开发环境/联调环境/测试环境分别创建了命名空间。

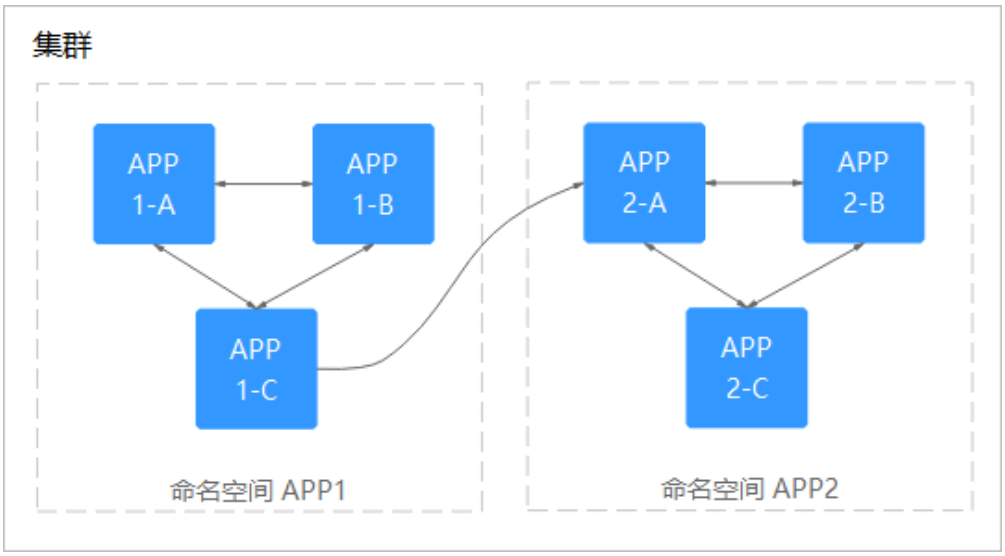
    图 12-1 不同环境创建对应命名空间

- **按照应用划分命名空间**

  对于同个环境中，应用数量较多的情况，建议进一步按照工作负载类型划分命名空间。例如下图中，按照APP1和APP2划分不同命名空间，将不同工作负载在逻辑上当做一个工作负载组进行管理。且同一个命名空间内的工作负载可以通过服务名称访问，不同命名空间下的通过服务名称、命名空间名称访问。

图 12-2 按照工作负载划分命名空间



管理命名空间标签

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧选择“命名空间”。

步骤2 单击目标命名空间操作列的“更多 > 标签管理”。

步骤3 弹出的窗口中将展示命名空间已有的标签，您可根据需要进行修改。

- 添加标签：单击“添加”，填写需要增加标签的“键”和“值”，单击“确定”。

  例如，填写的键为“project”，值为“cicd”，就可以从逻辑概念表示该命名空间是用来部署CICD环境使用。
- 删除标签：单击需要删除标签后的“删除”，并单击“确定”。

图 12-3 添加或删除命名空间标签

步骤4 标签修改成功后，再次进入该界面，在“标签”列下可查看到已经修改的标签。

----结束

启用命名空间节点亲和

启用命名空间节点亲和后，命名空间下新创建的工作负载只能调度到拥有特定标签的节点上，详情请参见**PodNodeSelector**。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧选择“命名空间”。

步骤2 找到目标命名空间，单击“节点亲和”列的。

步骤3 在弹出的窗口中，选择“启用”并单击“确定”。

启用后，命名空间下新创建的工作负载只能调度到拥有特定标签的节点上。例如，对于名为test的命名空间来说，该命名空间下的负载只能调度到添加了标签键为kubelet.kubernetes.io/namespace，值为test的节点上。

步骤4 您可以在节点管理中通过“标签与污点管理”为节点添加上述指定标签，详情请参见**管理节点标签**。

----结束

删除命名空间

删除命名空间会删除该命名空间下所有的资源（如工作负载，短任务、配置项等），请谨慎操作。

步骤1 登录CCE控制台，进入集群。

步骤2 在左侧导航栏中选择“命名空间”，选中待删除的命名空间，单击“更多 > 删除”。

根据系统提示进行删除操作。系统内置的命名空间不支持删除。

----结束

12.3 设置资源配额及限制

Kubernetes在一个物理集群上提供了多个虚拟集群，这些虚拟集群被称为命名空间。命名空间可用于多种工作用途，满足多用户、多环境、多应用的使用需求，通过为每个命名空间配置包括CPU、内存、Pod数量等资源的额度可以有效限制资源滥用，从而保证集群的可靠性，更多信息请参见**资源配额**。

从1.21版本集群开始，如果在**集群配置管理**中开启了enable-resource-quota参数，则创建命名空间将会同时创建默认的资源配额，根据集群规格不同，各个资源的配额如**表12-2**所示。您可以根据实际需求修改。

表 12-2 默认资源配额

集群规模	Pod	Deployment	Secret	ConfigMap	Service
50节点	2000	1000	1000	1000	1000
200节点	2000	1000	1000	1000	1000

集群规模	Pod	Deployment	Secret	ConfigMap	Service
1000节点	5000	2000	2000	2000	2000
2000节点	5000	2000	2000	2000	2000

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“命名空间”。

步骤3 单击对应命名空间后的“管理配额”。

系统级别的命名空间kube-system、kube-public默认不支持设置资源配额。

步骤4 设置资源配额，然后单击“确定”。

须知

- 命名空间设置了CPU或内存资源配额后，创建工作负载时，必须指定CPU或内存的请求值（request）和约束值（limit），否则CCE将拒绝创建实例。若设置资源配额值为0，则不限制该资源的使用。
- 配额累计使用量包含CCE系统默认创建的资源，如default命名空间下系统默认创建的kubernetes服务（该服务可通过后端kubectl工具查看）等，故建议命名空间下的资源配额略大于实际期望值以去除系统默认创建资源的影响。
- 在Kubernetes中，外部用户及内部组件频繁的数据更新操作采用乐观并行的控制方法。通过定义资源版本（resourceVersion）实现乐观锁，资源版本字段包含在对象的元数据（metadata）中。这个字段标识了对象的内部版本号，且对象被修改时，该字段将随之修改。kube-apiserver可以通过该字段判断对象是否已经被修改。当包含resourceVersion的更新请求到达apiserver，服务器端将对比请求数据与服务器中数据的资源版本号，如果不一致，则表明在本次更新提交时，服务端对象已被修改，此时apiserver将返回冲突错误（409）。客户端需重新获取服务端数据，重新修改后再次提交到服务器端；而资源配额对每个命名空间的资源消耗总量提供限制，并且会记录集群中的资源信息，因此开启资源配额后，在大规模并发场景下创建资源冲突概率会变高，会影响批创资源性能。

----结束

13 配置项与密钥

13.1 创建配置项

操作场景

配置项（ConfigMap）是一种用于存储工作负载所需配置信息的资源类型，内容由用户决定。配置项创建完成后，可在容器工作负载中作为文件或者环境变量使用。

配置项允许您将配置文件从容器镜像中解耦，从而增强容器工作负载的可移植性。

配置项价值如下：

- 使用配置项功能可以帮您管理不同环境、不同业务的配置。
- 方便您部署相同工作负载的不同环境，配置文件支持多版本，方便您进行更新和回滚工作负载。
- 方便您快速将您的配置以文件的形式导入到容器中。

约束与限制

- ConfigMap资源文件大小不得超过1MB。
- **静态Pod**中不可使用ConfigMap。

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“配置与密钥”，在右上角单击“创建配置项”。

步骤3 填写参数。

表 13-1 新建配置参数说明

参数	参数说明
名称	新建的配置项名称，同一个命名空间里命名必须唯一。
命名空间	新建配置项所在的命名空间。若不选择，默认为default。

参数	参数说明
描述	配置项的描述信息。
配置数据	配置项的数据。 键值对形式，单击 ┼ 添加。其中值支持String、JSON和YAML格式。
标签	配置项的标签。键值对形式，输入键值对后单击“确认添加”。

步骤4 配置完成后，单击“确定”。

工作负载配置列表中会出现新创建的工作负载配置。

----结束

使用 kubectl 创建配置项

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑cce-configmap.yaml文件。

vi cce-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cce-configmap
data:
  SPECIAL_LEVEL: Hello
  SPECIAL_TYPE: CCE
```

表 13-2 关键参数说明

参数	说明
apiVersion	固定值为v1。
kind	固定值为ConfigMap。
metadata.name	配置项名称，可自定义。
data	配置项的数据，需填写键值对形式。

步骤3 创建配置项。

kubectl create -f cce-configmap.yaml

查看已创建的配置项。

kubectl get cm

```
NAME            DATA      AGE
cce-configmap     3       7m
```

----结束

相关操作

配置项创建完成后，您还可以执行表13-3中的操作。

表 13-3 其他操作

操作	说明
编辑YAML	单击配置项名称后的“编辑YAML”，可编辑当前配置项的YAML文件。
更新配置	1. 选择需要更新的配置项名称，单击“更新”。 2. 根据表13-1更改信息。 3. 单击“确定”。
删除配置	选择要删除的配置项，单击“删除”。 根据系统提示删除配置。

13.2 使用配置项

配置项创建后，可在工作负载环境变量、命令行参数和数据卷三个场景使用。

- 通过配置项设置工作负载环境变量
- 通过配置项设置命令行参数
- 使用配置项挂载到工作负载数据卷

本节以下面这个ConfigMap为例，具体介绍ConfigMap的用法。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cce-configmap
data:
  SPECIAL_LEVEL: Hello
  SPECIAL_TYPE: CCE
```

须知

- 在工作负载里使用ConfigMap时，需要工作负载和ConfigMap处于同一集群和命名空间中。
- 以数据卷挂载使用ConfigMap时，当ConfigMap被更新，Kubernetes会同时更新数据卷中的数据。

  对于以subPath形式挂载的ConfigMap数据卷，当ConfigMap被更新时，Kubernetes无法自动更新数据卷中的数据。
- 以环境变量方式使用ConfigMap时，当ConfigMap被更新，数据不会被自动更新。更新这些数据需要重新启动Pod。

通过配置项设置工作负载环境变量

使用控制台方式

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“工作负载”，单击右上角“创建工作负载”。

在创建工作负载时，在“容器配置”中找到“环境变量”，单击“新增变量”。

- **配置项导入**：选择一个配置项，将配置项中所有键值都导入为环境变量。

- **配置项键值导入**：将配置项中某个键的值导入作为某个环境变量的值。
  - 变量名称：工作负载中的环境变量名称，可自定义，默认为配置项中选择的键名。
  - 变量/变量引用：选择一个配置项及需要导入的键名，将其对应的值导入为工作负载环境变量。

  例如将cce-configmap这个配置项中“SPECIAL_LEVEL”的值“Hello”导入，作为工作负载环境变量“SPECIAL_LEVEL”的值，导入后容器中有一个名为“SPECIAL_LEVEL”的环境变量，其值为“Hello”。

步骤3 配置其他工作负载参数后，单击“创建工作负载”。

等待工作负载正常运行后，您可**登录容器**执行以下语句，查看该配置项是否已被设置为工作负载的环境变量。

```
printenv SPECIAL_LEVEL
```

示例输出如下：

```
Hello
```

----结束

使用kubectl方式

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-configmap.yaml文件。

vi nginx-configmap.yaml

YAML文件内容如下：

- **配置项导入**：如果要将一个配置项中所有数据都添加到环境变量中，可以使用envFrom参数，配置项中的Key会成为工作负载中的环境变量名称。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-configmap
spec:
```



```
replicas: 1
selector:
  matchLabels:
    app: nginx-configmap
template:
  metadata:
    labels:
      app: nginx-configmap
  spec:
    containers:
    - name: container-1
      image: nginx:latest
      envFrom:                    # 使用envFrom来指定环境变量引用的配置项
      - configMapRef:
          name: cce-configmap     # 引用的配置项名称
    imagePullSecrets:
    - name: default-secret
```

- **配置项键值导入**：您可以在创建工作负载时将配置项设置为环境变量，使用valueFrom参数单独引用ConfigMap中的Key/Value。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-configmap
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-configmap
  template:
    metadata:
      labels:
        app: nginx-configmap
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        env:                          # 设置工作负载中的环境变量
        - name: SPECIAL_LEVEL         # 工作负载中的环境变量名称
          valueFrom:                  # 使用valueFrom来指定环境变量引用配置项
            configMapKeyRef:
              name: cce-configmap     # 引用的配置项名称
              key: SPECIAL_LEVEL      # 引用的配置项中的key
        - name: SPECIAL_TYPE          # 添加多个环境变量参数，可同时导入多个环境变量
          valueFrom:
            configMapKeyRef:
              name: cce-configmap
              key: SPECIAL_TYPE
      imagePullSecrets:
      - name: default-secret
```

步骤3 创建工作负载。

```
kubectl apply -f nginx-configmap.yaml
```

步骤4 创建完成后，查看Pod中的环境变量。

1. 执行以下命令，查看已创建的Pod。

   ```
   kubectl get pod | grep nginx-configmap
   ```

   预期输出如下：

   ```
   nginx-configmap-***   1/1     Running   0          2m18s
   ```

2. 执行以下命令，查看该Pod中的环境变量。

   ```
   kubectl exec nginx-configmap-*** -- printenv SPECIAL_LEVEL SPECIAL_TYPE
   ```

   预期输出如下：

   ```
   Hello
   CCE
   ```


说明该配置项已被设置为工作负载的环境变量。

----结束

通过配置项设置命令行参数

您可以使用配置项作为环境变量来设置容器中的命令或者参数值，使用环境变量替换语法$VAR_NAME来进行。

使用控制台方式

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“工作负载”，单击右上角“创建工作负载”。

在创建工作负载时，在“容器配置”中找到“环境变量”，单击“新增变量”。本例中以“配置项导入”为例。

- **配置项导入**：选择一个配置项，将配置项中所有键值都导入为环境变量。

步骤3 在“容器配置”中找到“生命周期”，在右侧选择“启动后处理”页签，并填写以下参数。

- 处理方式：命令行脚本。
- 执行命令：以下命令需分三行填写，其中*SPECIAL_LEVEL*和*SPECIAL_TYPE*为工作负载中的环境变量名，即cce-configmap配置项中的键名。

```
/bin/bash
-c
echo $SPECIAL_LEVEL $SPECIAL_TYPE > /usr/share/nginx/html/index.html
```

步骤4 配置其他工作负载参数后，单击“创建工作负载”。

等待工作负载正常运行后，您可登录容器执行以下语句，查看该配置项是否已被设置为工作负载的环境变量。

```
cat /usr/share/nginx/html/index.html
```

示例输出如下：

```
Hello CCE
```

----结束

使用kubectl方式

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-configmap.yaml文件。

vi nginx-configmap.yaml

如下面的示例所示，在工作负载中导入了cce-configmap配置项，其中*SPECIAL_LEVEL*和*SPECIAL_TYPE*为工作负载中的环境变量名，即cce-configmap配置项中的键名。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-configmap
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-configmap
  template:
    metadata:
      labels:
        app: nginx-configmap
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        lifecycle:
          postStart:
            exec:
              command: [ "/bin/sh", "-c", "echo $SPECIAL_LEVEL $SPECIAL_TYPE > /usr/share/nginx/html/
index.html" ]
        envFrom:                    # 使用envFrom来指定环境变量引用的配置项
        - configMapRef:
            name: cce-configmap     # 引用的配置项名称
      imagePullSecrets:
        - name: default-secret
```

步骤3 创建工作负载。

kubectl apply -f nginx-configmap.yaml

步骤4 等待工作负载正常运行后，容器中的/usr/share/nginx/html/index.html文件将被输入如下内容。

1. 执行以下命令，查看已创建的Pod。

   ```
   kubectl get pod | grep nginx-configmap
   ```

   预期输出如下：

   ```
   nginx-configmap-***   1/1     Running   0          2m18s
   ```

2. 执行以下命令，查看该Pod中的环境变量。

   ```
   kubectl exec nginx-configmap-*** -- cat /usr/share/nginx/html/index.html
   ```

   预期输出如下：

   ```
   Hello CCE
   ```

----结束

使用配置项挂载到工作负载数据卷

配置项(ConfigMap)挂载是将配置项中的数据挂载到指定的容器路径。平台提供工作负载代码和配置文件的分离，“配置项挂载”用于处理工作负载配置参数。用户需要提前创建工作负载配置，操作步骤请参见**创建配置项**。

使用控制台方式

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“工作负载”，单击右上角“创建工作负载”。

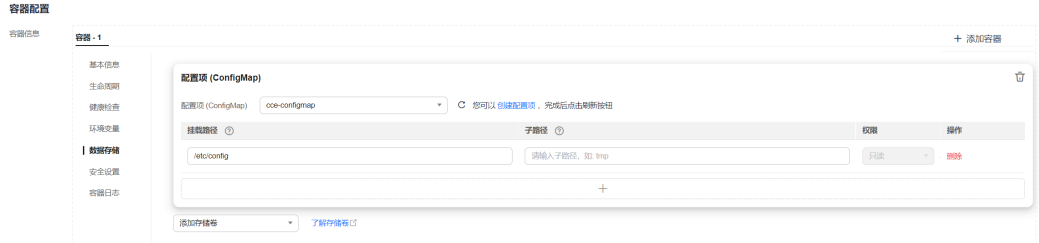
在创建工作负载时，在“容器配置”中找到“数据存储”，选择“添加存储卷 > 配置项(ConfigMap)”。

步骤3 选择配置项挂载参数，如表13-4。

表 13-4 配置项挂载

参数	参数说明
配置项	选择对应的配置项名称。 配置项需要提前创建，具体请参见创建配置项。
挂载路径	请输入挂载路径。配置项挂载完成后，会在容器中的挂载路径下生成以配置项中的key为文件名， value为文件内容的配置文件。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“/var/run”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入挂载路径的子路径。 • 使用子路径挂载本地磁盘，实现在单一Pod中重复使用同一个Volume，不填写时默认为根。 • 子路径可以填写ConfigMap/Secret的键值，子路径若填写为不存在的键值则数据导入不会生效。 • 通过子路径导入的数据不会随ConfigMap/Secret的更新而动态更新。
权限	只读。只能读容器路径中的数据卷。

图 13-1 使用配置项挂载到工作负载数据卷



步骤4 其余信息都配置完成后，单击“创建工作负载”。

等待工作负载正常运行后，本示例将在/etc/config目录下生成SPECIAL_LEVEL和SPECIAL_TYPE两个文件，且文件的内容分别为Hello和CCE。

您可登录容器执行以下语句，查看容器中的SPECIAL_LEVEL或SPECIAL_TYPE文件。


```
cat /etc/config/SPECIAL_LEVEL
```

预期输出如下：

```
Hello
```

----结束

使用kubectl方式

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-configmap.yaml文件。

vi nginx-configmap.yaml

如下面的示例所示，配置项挂载完成后，最终会在容器中的/etc/config目录下生成以配置项中的key为文件名， value为文件内容的配置文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-configmap
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-configmap
  template:
    metadata:
      labels:
        app: nginx-configmap
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
        - name: config-volume
          mountPath: /etc/config          # 挂载到/etc/config目录下
          readOnly: true
    volumes:
    - name: config-volume
      configMap:
        name: cce-configmap               # 引用的配置项名称
```

步骤3 创建工作负载。

kubectl apply -f nginx-configmap.yaml

步骤4 等待工作负载正常运行后，在/etc/config目录下会生成SPECIAL_LEVEL和SPECIAL_TYPE两个文件，且文件的内容分别为Hello和CCE。

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep nginx-configmap
   ```
   预期输出如下：
   ```
   nginx-configmap-***   1/1     Running   0          2m18s
   ```
2. 执行以下命令，查看该Pod中的SPECIAL_LEVEL或SPECIAL_TYPE文件。
   ```
   kubectl exec nginx-configmap-*** -- cat /etc/config/SPECIAL_LEVEL
   ```
   预期输出如下：
   ```
   Hello
   ```

----结束

13.3 创建密钥

操作场景

密钥（Secret）是一种用于存储工作负载所需要认证信息、密钥的敏感信息等的资源类型，内容由用户决定。资源创建完成后，可在容器工作负载中作为文件或者环境变量使用。

约束与限制

静态Pod中不可使用Secret。

操作步骤

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏中选择“配置与密钥”，选择“密钥”页签，在右上角单击“创建密钥”。

步骤3 填写参数。

表 13-5 基本信息说明

参数	参数说明
名称	新建的密钥的名称，同一个命名空间内命名必须唯一。
命名空间	新建密钥所在的命名空间，默认为default。
描述	密钥的描述信息。
密钥类型	新建的密钥类型。 • Opaque：一般密钥类型。 • kubernetes.io/dockerconfigjson：存放拉取私有仓库镜像所需的认证信息。 • kubernetes.io/tls：Kubernetes的TLS密钥类型，用于存放7层负载均衡服务所需的证书。kubernetes.io/tls类型的密钥示例及说明请参见**TLS Secret**。 • IngressTLS：CCE提供的TLS密钥类型，用于存放7层负载均衡服务所需的证书。 • 其他：若需要创建其他类型的密钥，请手动输入密钥类型。

参数	参数说明
密钥数据	工作负载密钥的数据可以在容器中使用。 • 当密钥为Opaque类型时，单击+，在弹出的窗口中输入键值对，并且可以勾选“自动Base64转码”。 • 当密钥为kubernetes.io/dockerconfigjson类型时，输入私有镜像仓库的账号和密码。 • 当密钥为kubernetes.io/tls或IngressTLS类型时，上传证书文件和私钥文件。 **说明** – 证书是自签名或CA签名过的凭据，用来进行身份认证。 – 证书请求是对签名的请求，需要使用私钥进行签名。
密钥标签	密钥的标签。键值对形式，输入键值对后单击“确认添加”。

步骤4 配置完成后，单击“确定”。

密钥列表中会出现新创建的密钥。

----结束

Secret 资源文件配置示例

本章节主要介绍Secret类型的资源描述文件的配置示例。

- Opaque类型

  定义的Secret文件secret.yaml内容如下。其中data字段以键值对的形式填写，value需要用Base64编码，Base64编码方法请参见**如何进行Base64编码**。

  ```
  apiVersion: v1
  kind: Secret
  metadata:
    name: mysecret          # secret的名称
    namespace: default      #命名空间，默认为default
  data:
    <your_key>: <your_value>  #填写键值对，其中value需要用Base64编码
  type: Opaque
  ```

- kubernetes.io/dockerconfigjson类型

  定义的Secret文件secret.yaml内容如下。其中.dockerconfigjson需要用Base64，Base64编码方法请参见**如何进行Base64编码**。

  ```
  apiVersion: v1
  kind: Secret
  metadata:
    name: mysecret          # secret的名称
    namespace: default      #命名空间，默认为default
  data:
    .dockerconfigjson: eyJh*****    #Base64编码后的内容
  type: kubernetes.io/dockerconfigjson
  ```

  获取.dockerconfigjson内容的步骤如下：

  a. 获取镜像仓库的登录信息：

     - 镜像仓库地址：本文中以address为例，请根据实际信息替换。

- 用户名：本文中以username为例，请根据实际信息替换。
- 密码：本文中以password为例，请根据实际信息替换。

b. 使用Base64将键值对username:password进行编码，获取编码后的内容填入3中。

```
echo -n "username:password" | base64
```

回显如下：

```
dXNlcm5hbWU6cGFzc3dvcmQ=
```

c. 使用Base64对以下JSON内容进行编码。

```
echo -n '{"auths":{"address":
{"username":"username","password":"password","auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}'
| base64
```

回显如下：

```
eyJhdXRocyI6eyJhZGRyZXNzIjp7InVzZXJuYW1lIjoidXNlcm5hbWUiLCJwYXNzd29yZCI6InBhc3N3b3JkIiwiYXV0aCI6ImRYTmxjbTVoYldVNmNHRnpjM2R2Y21RPSJ9fX0=
```

编码后的内容即为.dockerconfigjson内容。

- kubernetes.io/tls类型

  其中tls.crt和tls.key需要用Base64，Base64编码方法请参见**如何进行Base64编码**。

```
kind: Secret
apiVersion: v1
metadata:
  name: mysecret          # secret的名称
  namespace: default      #命名空间，默认为default
data:
  tls.crt: LS0tLS1CRU*****FURS0tLS0t  #证书内容，需要Base64编码
  tls.key: LS0tLS1CRU*****VZLS0tLS0=  #私钥内容，需要Base64编码
type: kubernetes.io/tls
```

- IngressTLS类型

  其中tls.crt和tls.key需要用Base64，Base64编码方法请参见**如何进行Base64编码**。

```
kind: Secret
apiVersion: v1
metadata:
  name: mysecret          # secret的名称
  namespace: default      #命名空间，默认为default
data:
  tls.crt: LS0tLS1CRU*****FURS0tLS0t  #证书内容，需要Base64编码
  tls.key: LS0tLS1CRU*****VZLS0tLS0=  #私钥内容，需要Base64编码
type: IngressTLS
```

使用 kubectl 创建密钥

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 通过Base64编码，创建并编辑cce-secret.yaml文件。

```
# echo -n "待编码内容" | base64
******
```

vi cce-secret.yaml

Opaque类型的YAML示例如下，其余类型请参见**Secret资源文件配置示例**：

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
```



```
data:
  <your_key>: <your_value>  #填写键值对，其中value需要用Base64编码
```

步骤3 创建密钥。

kubectl create -f cce-secret.yaml

创建完成后可以查询到密钥。

kubectl get secret -n default

----结束

相关操作

密钥创建完成后，您还可以执行表13-6中的操作。

说明

密钥列表中包含系统密钥资源，系统密钥资源不可更新，也不能删除，只能查看。

表 13-6 其他操作

操作	说明
编辑YAML	单击密钥名称后的“编辑YAML”，可编辑当前密钥的YAML文件。
更新密钥	1. 选择需要更新的密钥名称，单击“更新”。 2. 根据表13-5更改信息。 3. 单击“确定”。
删除密钥	选择要删除的密钥，单击“删除”。 根据系统提示删除密钥。
批量删除密钥	1. 勾选需要删除的密钥名称。 2. 单击页面左上角的“批量删除”，删除选中的密钥。 3. 根据系统提示删除密钥。

如何进行 Base64 编码

对字符串进行Base64编码，可以直接使用“echo -n 要编码的内容 | base64”命令即可，示例如下：

```
root@ubuntu:~# echo -n "待编码内容" | base64
******
```

13.4 使用密钥

密钥创建后，可在工作负载环境变量和数据卷两个场景使用。

须知

请勿对以下CCE系统使用的密钥做任何操作，详情请参见**集群系统密钥说明**。

- 请不要操作kube-system下的secrets。
- 请不要操作任何命名空间下的default-secret、paas.elb。其中，default-secret用于SWR的私有镜像拉取，paas.elb用于该命名空间下的服务对接ELB。

- **使用密钥设置工作负载的环境变量**
- **使用密钥配置工作负载的数据卷**

本节以下面这个Secret为例，具体介绍Secret的用法。

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: ******  #需要用Base64编码
  password: ******  #需要用Base64编码
```

须知

- 在Pod里使用密钥时，需要Pod和密钥处于同一集群和命名空间中。
- 当Secret 被更新时，Kubernetes会同时更新数据卷中的数据。

  但对于以**subPath**形式挂载的Secret数据卷，当Secret 被更新时，Kubernetes无法自动更新数据卷中的数据。

使用密钥设置工作负载的环境变量

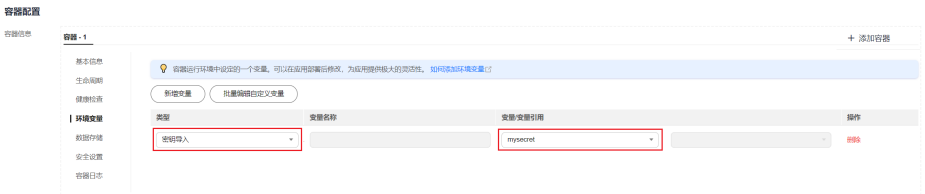
使用控制台方式

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“工作负载”，单击右上角“创建工作负载”。

在创建工作负载时，在“容器配置”中找到“环境变量”，单击“新增变量”。

- **密钥导入**：选择一个密钥，将密钥中所有键值都导入为环境变量。



- **密钥项键值导入**：将密钥中某个键的值导入作为某个环境变量的值。
  - 变量名称：工作负载中的环境变量名称，可自定义，默认为密钥中选择的键名。
  - 变量/变量引用：选择一个密钥及需要导入的键名，将其对应的值导入为工作负载环境变量。

例如将mysecret这个密钥中“username”的值导入，作为工作负载环境变量“username”的值，导入后容器中将会有一个名为“username”的环境变量。

步骤3 配置其他工作负载参数后，单击“创建工作负载”。

等待工作负载正常运行后，您可**登录容器**执行以下语句，查看该密钥是否已被设置为工作负载的环境变量。

```
printenv username
```

如输出与Secret中的内容一致，则说明该密钥已被设置为工作负载的环境变量。

----结束

使用kubectl方式

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-secret.yaml文件。

vi nginx-secret.yaml

YAML文件内容如下：

- **密钥导入**：如果要将一个密钥中所有数据都添加到环境变量中，可以使用envFrom参数，密钥中的Key会成为工作负载中的环境变量名称。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-secret
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-secret
  template:
    metadata:
      labels:
        app: nginx-secret
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        envFrom:                # 使用envFrom来指定环境变量引用的密钥
        - secretRef:
            name: mysecret      # 引用的密钥名称
      imagePullSecrets:
      - name: default-secret
```

- **密钥键值导入**：您可以在创建工作负载时将密钥设置为环境变量，使用valueFrom参数单独引用Secret中的Key/Value。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-secret
spec:
  replicas: 1
  selector:
    matchLabels:
```



```
    app: nginx-secret
template:
  metadata:
    labels:
      app: nginx-secret
  spec:
    containers:
    - name: container-1
      image: nginx:latest
      env:                          # 设置工作负载中的环境变量
      - name: SECRET_USERNAME       # 工作负载中的环境变量名称
        valueFrom:                  # 使用valueFrom来指定环境变量引用的密钥
          secretKeyRef:
            name: mysecret          # 引用的密钥名称
            key: username           # 引用的密钥中的key
      - name: SECRET_PASSWORD       # 添加多个环境变量参数，可同时导入多个环境变量
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: password
    imagePullSecrets:
    - name: default-secret
```

步骤3 创建工作负载。

kubectl apply -f nginx-secret.yaml

步骤4 创建完成后，查看Pod中的环境变量。

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep nginx-secret
   ```
   预期输出如下：
   ```
   nginx-secret-***   1/1     Running   0          2m18s
   ```
2. 执行以下命令，查看该Pod中的环境变量。
   ```
   kubectl exec nginx-secret-*** -- printenv SPECIAL_USERNAME SPECIAL_PASSWORD
   ```
   如输出与Secret中的内容一致，则说明该密钥已被设置为工作负载的环境变量。

----结束

使用密钥配置工作负载的数据卷

密钥(Secret)挂载将密钥中的数据挂载到指定的容器路径，密钥内容由用户决定。用户需要提前创建密钥，操作步骤请参见**创建密钥**。

使用控制台方式

步骤1 登录CCE控制台，单击集群名称进入集群。

步骤2 在左侧导航栏选择“工作负载”，在右侧选择“无状态负载”页签。单击右上角“创建工作负载”。

在创建工作负载时，在“容器配置”中找到“数据存储”，选择“添加存储卷 > 密钥(Secret)”。

步骤3 选择密钥挂载参数，如**表13-7**。

表 13-7 密钥挂载

参数	参数说明
密钥	选择对应的密钥名称。 密钥需要提前创建，具体请参见**创建密钥**。
挂载路径	请输入挂载路径。密钥挂载完成后，会在容器中的挂载路径下生成以密钥中的key为文件名， value为文件内容的密钥文件。 数据存储挂载到容器上的路径。请不要挂载在系统目录下，如“/”、“**/var/run**”等，会导致容器异常。建议挂载在空目录下，若目录不为空，请确保目录下无影响容器启动的文件，否则文件会被替换，导致容器启动异常，工作负载创建失败。 **须知** 挂载高危目录的情况下 ，建议使用低权限账号启动，否则可能会造成宿主机高危文件被破坏。
子路径	请输入挂载路径的子路径。 • 使用子路径挂载本地磁盘，实现在单一Pod中重复使用同一个Volume，不填写时默认为根。 • 子路径可以填写ConfigMap/Secret的键值，子路径若填写为不存在的键值则数据导入不会生效。 • 通过子路径导入的数据不会随ConfigMap/Secret的更新而动态更新。
权限	只读。只能读容器路径中的数据卷。

图 13-2 使用密钥挂载到工作负载数据卷

步骤4 其余信息都配置完成后，单击“创建工作负载”。

等待工作负载正常运行后，本示例将在/etc/foo目录下生成username和password两个文件，且文件的内容分别为密钥值。

您可**登录容器**执行以下语句，查看容器中的username和password两个文件。

```
cat /etc/foo/username
```

预期输出与Secret中的内容一致。

----结束

使用kubectl方式

步骤1 请参见**通过kubectl连接集群**配置kubectl命令。

步骤2 创建并编辑nginx-secret.yaml文件。

vi nginx-secret.yaml

如下面的示例所示，mysecret密钥的username和password以文件方式保存在/etc/foo目录下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-secret
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-secret
  template:
    metadata:
      labels:
        app: nginx-secret
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
       - name: foo
          mountPath: /etc/foo          # 挂载到/etc/foo目录下
          readOnly: true
    volumes:
    - name: foo
      secret:
        secretName: mysecret      # 引用的密钥名称
```

您还可以使用items字段控制Secret键的映射路径，例如，将username存放在容器中的/etc/foo/my-group/my-username目录下。

说明

- 使用items字段指定Secret键的映射路径后，没有被指定的键将不会被以文件形式创建。例如，下面的例子中的password键未被指定，则该文件将不会被创建。
- 如果要使用Secret中全部的键，那么必须将全部的键都列在items字段中。
- items字段中列出的所有键必须存在于相应的Secret 中。否则，该卷不被创建。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-secret
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-secret
  template:
    metadata:
      labels:
        app: nginx-secret
    spec:
      containers:
      - name: container-1
        image: nginx:latest
        volumeMounts:
       - name: foo
          mountPath: /etc/foo          # 挂载到/etc/foo目录下
          readOnly: true
    volumes:
    - name: foo
      secret:
        secretName: mysecret      # 引用的密钥名称
```



```
    items:
    - key: username      # 引用的密钥中的键名
      path: my-group/my-username    # Secret键的映射路径
```

步骤3 创建工作负载。

kubectl apply -f nginx-secret.yaml

步骤4 等待工作负载正常运行后，在/etc/foo目录下会生成username和password两个文件。

1. 执行以下命令，查看已创建的Pod。
   ```
   kubectl get pod | grep nginx-secret
   ```
   预期输出如下：
   ```
   nginx-secret-***   1/1     Running   0          2m18s
   ```
2. 执行以下命令，查看该Pod中的username或password文件。
   ```
   kubectl exec nginx-secret-*** -- cat /etc/foo/username
   ```
   预期输出与Secret中的内容一致。

----结束

13.5 集群系统密钥说明

CCE默认会在每个命名空间下创建如下密钥。

- default-secret
- paas.elb
- default-token-xxxxx（ xxxxx为随机数 ）

下面将详细介绍这个几个密钥的用途。

default-secret

default-secret的类型为kubernetes.io/dockerconfigjson，其data内容是登录SWR镜像仓库的凭据，用于从SWR拉取镜像。在CCE中创建工作负载时如果需要从SWR拉取镜像，需要配置imagePullSecrets的取值为default-secret，如下所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: container-0
    resources:
      limits:
        cpu: 100m
        memory: 200Mi
      requests:
        cpu: 100m
        memory: 200Mi
  imagePullSecrets:
  - name: default-secret
```

default-secret的data数据会定期更新，且当前的data内容会在一定时间后会过期失效。您可以使用describe命令在default-secret的中查看到具体的过期时间，如下所示。

须知

在使用时请直接使用default-secret，而不要复制secret内容重新创建，因为secret里面的凭据会过期，从而导致无法拉取镜像。

```
$ kubectl describe secret default-secret
Name:         default-secret
Namespace:    default
Labels:       secret-generated-by=cce
Annotations:  temporary-ak-sk-expires-at: 2021-11-26 20:55:31.380909 +0000 UTC

Type:  kubernetes.io/dockerconfigjson

Data
====
.dockerconfigjson:  347 bytes
```

paas.elb

paas.elb的data内容是临时AK/SK数据，创建节点或自动创建ELB时均会使用该密钥。paas.elb的data数据同样会定期更新，且在一定时间后会过期失效。

实际使用中您不会直接使用paas.elb，但请不要删除paas.elb，否则会导致创建节点或ELB失败。

default-token-xxxxx

Kubernetes为每个命名空间默认创建一个名为default的ServiceAccount，default-token-xxxxx为这个ServiceAccount的密钥，xxxxx是随机数。

```
$ kubectl get sa
NAME     SECRETS   AGE
default  1         30d
$ kubectl describe sa default
Name:                default
Namespace:           default
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   default-token-xxxxx
Tokens:              default-token-xxxxx
Events:              <none>
```


14 插件

14.1 插件概述

CCE提供了多种类型的插件，用于管理集群的扩展功能，以支持选择性扩展满足特性需求的功能。

须知

CCE插件采用Helm模板方式部署，修改或升级插件请从插件配置页面或开放的插件管理API进行操作。勿直接后台直接修改插件相关资源，以免插件异常或引入其他非预期问题。

容器调度与弹性插件

插件名称	插件简介
Volcano调度器	Volcano调度器提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力，通过接入AI、大数据、基因、渲染等诸多行业计算框架服务终端用户。
CCE集群弹性引擎	集群自动扩缩容插件autoscaler，是根据pod调度状态及资源使用情况对集群的工作节点进行自动扩容缩容的插件。
CCE容器弹性引擎	CCE容器弹性引擎插件是一款CCE自研的插件，能够基于CPU利用率、内存利用率等指标，对无状态工作负载进行弹性扩缩容。
CCE突发弹性引擎（对接CCI）	Virtual Kubelet是基于社区Virtual Kubelet开源项目开发的插件，该插件支持用户在短时高负载场景下，将部署在CCE上的无状态负载（Deployment）、有状态负载（StatefulSet）、普通任务（Job）、定时任务（CronJob）四种资源类型的容器实例（Pod），弹性创建到云容器实例CCI服务上，以减少集群扩容带来的消耗。

云原生可观测性插件

插件名称	插件简介
云原生监控插件	kube-prometheus-stack通过使用Prometheus-operator和Prometheus，提供简单易用的端到端Kubernetes集群监控能力。 使用kube-prometheus-stack可将监控数据与监控中心对接，在监控中心控制台查看监控数据，配置告警等。
云原生日志采集插件	log-agent是基于开源fluent-bit和opentelemetry构建的云原生日志采集插件。log-agent支持基于CRD的日志采集策略，可以根据您配置的策略规则，对集群中的容器标准输出日志、容器文件日志、节点日志及K8s事件日志进行采集与转发。
CCE节点故障检测	CCE节点故障检测插件（node-problem-detector，简称NPD）是一款监控集群节点异常事件的插件，以及对接第三方监控平台功能的组件。它是一个在每个节点上运行的守护程序，可从不同的守护进程中搜集节点问题并将其报告给apiserver。node-problem-detector可以作为DaemonSet运行， 也可以独立运行。
CCE容器网络扩展指标	CCE容器网络扩展指标是一款容器网络流量监控管理插件。支持流量统计信息ipv4发送公网报文数和字节数、ipv4接收报文数和字节数以及ipv4发送报文数和字节数，且支持通过PodSelector来对监控后端作选择，支持多监控任务、可选监控指标，且支持用户获取Pod的label标签信息。监控信息已适配Prometheus格式，可以通过调用Prometheus接口查看监控数据。
Kubernetes Metrics Server	Metrics-Server是集群核心资源监控数据的聚合器。
Grafana	Grafana是一款开源的数据可视化和监控平台，可以为您提供丰富的图表和面板，用于实时监控、分析和可视化各种指标和数据源。
Prometheus（停止维护）	Prometheus是一套开源的系统监控报警框架。在云容器引擎CCE中，支持以插件的方式快捷安装Prometheus。

云原生异构计算插件

插件名称	插件简介
CCE AI套件（NVIDIA GPU）	CCE AI套件（NVIDIA GPU）是支持在容器中使用GPU显卡的设备管理插件，仅支持Nvidia驱动。
CCE AI套件（Ascend NPU）	CCE AI套件（Ascend NPU）是支持容器里使用Huawei NPU设备的管理插件。

容器网络插件

插件名称	插件简介
CoreDNS域名解析	CoreDNS域名解析插件是一款通过链式插件的方式为Kubernetes提供域名解析服务的DNS服务器。
NGINX Ingress控制器	NGINX Ingress控制器为Service提供了可直接被集群外部访问的虚拟主机、负载均衡、SSL代理、HTTP路由等应用层转发功能。
节点本地域名解析加速	NodeLocal DNSCache通过在集群节点上作为守护程序集运行DNS缓存代理，提高集群DNS性能。

容器存储插件

插件名称	插件简介
CCE容器存储（Everest）	CCE容器存储插件（Everest）是一个云原生容器存储系统，基于CSI为Kubernetes v1.15.6及以上版本集群对接云存储服务的能力。

容器安全插件

插件名称	插件简介
CCE密钥管理（对接 DEW）	CCE密钥管理插件用于对接数据加密服务(Data Encryption Workshop, DEW)。该插件允许用户将存储在集群外部（即专门存储敏感信息的数据加密服务）的凭据挂载至业务Pod内，从而将敏感信息与集群环境解耦，有效避免程序硬编码或明文配置等问题导致的敏感信息泄密。
容器镜像签名验证	容器镜像签名验证插件（swr-cosign）提供镜像验签功能，可以对镜像文件进行数字签名验证，以确保镜像文件的完整性和真实性，有效地防止软件被篡改或植入恶意代码，保障用户的安全。

其他插件

插件名称	插件简介
Kubernetes Dashboard	Kubernetes Dashboard是Kubernetes集群基于Web的通用UI，集合了命令行可以操作的所有命令。它允许用户管理在集群中运行应用程序并对其进行故障排除，以及管理集群本身。
OpenKruise	一个基于Kubernetes的扩展套件，主要聚焦于云原生应用的自动化，比如部署、发布、运维以及可用性防护。

插件名称	插件简介
Gatekeeper	一个基于开放策略（OPA）的可定制的云原生策略控制器，有助于策略的执行和治理能力的加强，在集群中提供了更多符合Kubernetes应用场景的安全策略规则。
Kubernetes Web终端（停止维护）	Kubernetes Web终端（web-terminal）是一款支持在Web界面上使用Kubectl的插件。它支持使用WebSocket通过浏览器连接Linux，提供灵活的接口便于集成到独立系统中，可直接作为一个服务连接，通过cmdb获取信息并登录服务器。

插件生命周期

生命周期是指插件从安装到卸载历经的各种状态。

表 14-1 插件生命周期状态说明

状态	状态属性	说明
运行中	稳定状态	插件正常运行状态，所有插件实例均正常部署，插件可正常使用。
部分就绪	稳定状态	插件正常运行状态，部分插件实例未正常部署。此状态下，插件功能可能无法正常使用。
不可用	稳定状态	插件异常状态，所有插件实例均未正常部署。
安装中	中间状态	插件正处于部署状态。 如遇到插件配置错误或资源不足所有实例均无法调度等情况，系统会在10分钟后将该插件置为“不可用”状态。
安装失败	稳定状态	插件安装失败，需要卸载后重新安装。
升级中	中间状态	插件正处于更新状态。
升级失败	稳定状态	插件升级失败，可重试升级或卸载后重新安装。
回滚中	中间状态	插件正在回滚中。
回滚失败	稳定状态	插件回滚失败，可重试回滚或卸载后重新安装。
删除中	中间状态	插件处于正在被删除的状态。 如果长时间处于该状态，则说明出现异常。
删除失败	稳定状态	插件删除失败，可重试卸载。
未知状态	稳定状态	插件模板实例不存在。

说明

当插件处于“安装中”或“删除中”等中间状态时，不可进行编辑、卸载等相关操作。

当插件状态处于“未知状态”且对应插件返回信息的status.Reason字段为"don't install the addon in this cluster"时，一般为集群中对应插件的helm release关联secret被误删导致，此类场景可先卸载插件，然后以相同配置参数重新安装插件恢复。

插件相关操作

您可以在“插件中心”执行表14-2中的操作。

表 14-2 插件相关操作

操作	说明	操作步骤
安装	安装指定的插件。	1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏选择“插件中心”。 2. 单击需要安装插件下的“安装”。 由于不同插件支持的配置参数不同，详细步骤请参见插件章节。 3. 设置完插件参数后，单击“确定”。
升级	将插件升级至新版。	1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏选择“插件中心”。 2. 如存在可升级的插件，该插件将提供“升级”按钮。 单击“升级”。由于不同插件支持的配置参数不同，详细步骤请参见插件章节。 3. 设置完插件参数后，单击“确定”。
编辑	编辑插件参数。	1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏选择“插件中心”。 2. 单击需要编辑插件下的“编辑”。 由于不同插件支持的配置参数不同，详细步骤请参见插件章节。 3. 设置完插件参数后，单击“确定”。
卸载	将插件从集群中卸载。	1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏选择“插件中心”。 2. 单击需要编辑插件下的“卸载”。 3. 在弹出的确认窗口中单击“是”。 卸载操作无法恢复，请谨慎操作。
回滚	将插件回滚至升级前版本。 **说明** • 插件升级后，可**回滚到升级前的版本**，如仅编辑插件参数将无法使用回滚功能。 • 插件回滚后，无法再一次回滚。	1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏选择“插件中心”。 2. 如存在可回滚的插件，该插件将提供“回滚”按钮。 单击“回滚”。 3. 在弹出的确认窗口中单击“是”。

说明

插件回滚能力需要插件版本支持，支持的插件及版本如下：

- coredns：1.25.11及以上版本支持回滚
- everest：2.1.19及以上版本支持回滚
- autoscaler：
  - v1.21集群：1.21.22及以上版本支持回滚
  - v1.23集群：1.23.24及以上版本支持回滚
  - v1.25集群：1.25.14及以上版本支持回滚
- kube-prometheus-stack：3.7.2及以上版本支持回滚
- volcano：1.11.4及以上版本支持回滚
- npd：1.18.22及以上版本支持回滚

14.2 容器调度与弹性插件

14.2.1 Volcano 调度器

插件简介

Volcano是一个基于Kubernetes的批处理平台，提供了机器学习、深度学习、生物信息学、基因组学及其他大数据应用所需要而Kubernetes当前缺失的一系列特性。

Volcano提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力，通过接入AI、大数据、基因、渲染等诸多行业计算框架服务终端用户，最大支持1000Pod/s的调度并发数，轻松应对各种规模的工作负载，大大提高调度效率和资源利用率。

Volcano针对计算型应用提供了作业调度、作业管理、队列管理等多项功能，主要特性包括：

- 丰富的计算框架支持：通过CRD提供了批量计算任务的通用API，通过提供丰富的插件及作业生命周期高级管理，支持TensorFlow，MPI，Spark等计算框架容器化运行在Kubernetes上。
- 高级调度：面向批量计算、高性能计算场景提供丰富的高级调度能力，包括成组调度，优先级抢占、装箱、资源预留、任务拓扑关系等。
- 队列管理：支持分队列调度，提供队列优先级、多级队列等复杂任务调度能力。

目前Volcano项目已经在Github开源，项目开源地址：**https://github.com/volcano-sh/volcano**

本文介绍如何在CCE集群中安装及配置Volcano插件，具体使用方法请参见**Volcano调度**。

说明

在使用Volcano作为调度器时，建议将集群中所有工作负载都使用Volcano进行调度，以避免多调度器同时工作导致的一些调度资源冲突问题。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**Volcano调度器**，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

  其中volcano-admission组件的资源配额设置与集群节点和Pod规模无关，可保持默认值。而volcano-controller和volcano-scheduler组件的资源配额设置与集群节点和Pod规模相关，其建议值如下：

  - 小于100个节点，可使用默认配置，即CPU的申请值为500m，限制值为2000m；内存的申请值为500Mi，限制值为2000Mi。
  - 高于100个节点，每增加100个节点（10000个Pod），建议CPU的申请值增加500m，内存的申请值增加1000Mi；CPU的限制值建议比申请值多1500m，内存的限制值建议比申请值多1000Mi。

    说明

    申请值推荐计算公式：

    - CPU申请值：计算“目标节点数 * 目标Pod规模”的值，并在**表14-3**中根据“集群节点数 * Pod规模”的计算值进行插值查找，向上取最接近规格的申请值及限制值。

      例如2000节点和2w个Pod的场景下，“目标节点数 * 目标Pod规模”等于4000w，向上取最接近的规格为700/7w（“集群节点数 * Pod规模”等于4900w），因此建议CPU申请值为4000m，限制值为5500m。

    - 内存申请值：建议每1000个节点分配2.4G内存，每1w个Pod分配1G内存，二者叠加进行计算。（该计算方法相比**表14-3**中的建议值会存在一定的误差，通过查表或计算均可）

      即：内存申请值 = 目标节点数/1000 * 2.4G + 目标Pod规模/1w * 1G。

      例如2000节点和2w个Pod的场景下，内存申请值 = 2 * 2.4G + 2 * 1G = 6.8G

表 14-3 volcano-controller 和 volcano-scheduler 的建议值

集群节点数/Pod规模	CPU Request(m)	CPU Limit(m)	Memory Request(Mi)	Memory Limit(Mi)
50/5k	500	2000	500	2000
100/1w	1000	2500	1500	2500
200/2w	1500	3000	2500	3500
300/3w	2000	3500	3500	4500
400/4w	2500	4000	4500	5500
500/5w	3000	4500	5500	6500
600/6w	3500	5000	6500	7500

集群节点数/Pod规模	CPU Request(m)	CPU Limit(m)	Memory Request(Mi)	Memory Limit(Mi)
700/7w	4000	5500	7500	8500

步骤3 设置插件支持的“扩展功能”。

- 重调度：启用后，默认部署volcano-descheduler组件，调度器根据您的策略配置驱逐和重新调度不符合要求的pod，实现集群负载均衡或减少资源碎片的效果。详情请参见**重调度（Descheduler）**。
- 在离线业务混部：启用后，开启混部能力的节点池默认部署volcano-agent组件，通过节点QoS保障、CPU Burst和动态资源超卖等方式提升资源利用率，为您降低资源使用成本。详情请参见**开启云原生混部**。
- NUMA拓扑调度：启用后，默认部署resource-exporter组件，调度器按照NUMA亲和的方式调度工作负载，提高高性能训练作业性能。详情请参见**NUMA亲和性调度**。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-4 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。

参数	参数说明
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

插件安装完成后，您可以单击左侧导航栏的“配置中心”，切换至“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，您可以结合实际业务场景定制专属的高阶调度策略。示例如下：

```
colocation_enable: ''
default_scheduler_conf:
  actions: 'allocate, backfill, preempt'
  tiers:
    - plugins:
        - name: 'priority'
        - name: 'gang'
        - name: 'conformance'
        - name: 'lifecycle'
          arguments:
            lifecycle.MaxGrade: 10
            lifecycle.MaxScore: 200.0
            lifecycle.SaturatedTresh: 1.0
            lifecycle.WindowSize: 10
    - plugins:
        - name: 'drf'
        - name: 'predicates'
        - name: 'nodeorder'
    - plugins:
        - name: 'cce-gpu-topology-predicate'
        - name: 'cce-gpu-topology-priority'
        - name: 'cce-gpu'
    - plugins:
        - name: 'nodelocalvolume'
        - name: 'nodeemptydirvolume'
        - name: 'nodeCSIscheduling'
        - name: 'networkresource'
tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 60
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 60
```


表 14-5 Volcano 高级配置参数说明

插件	功能	参数说明	用法演示
colocation_enable	是否开启混部能力。	参数值： • true：表示开启混部。 • false：表示不开启混部。	-
default_scheduler_conf	负责Pod调度的组件配置，由一系列action和plugin组成。具有高度的可扩展性，您可以根据需要实现自己的action和plugin。	主要包括actions和tiers两部分： • actions：定义调度器需要执行的action类型及顺序。 • tiers：配置plugin列表。	-
actions	定义了调度各环节中需要执行的动作，action的配置顺序就是scheduler的执行顺序。详情请参见**Actions**。 调度器会遍历所有的待调度Job，按照定义的次序依次执行enqueue、allocate、preempt、backfill等动作，为每个Job找到一个最合适的节点。	支持的参数值： • enqueue：负责通过一系列的过滤算法筛选出符合要求的待调度任务并将它们送入待调度队列。经过这个action，任务的状态将由pending变为inqueue。 • allocate：负责通过一系列的预选和优选算法筛选出最适合的节点。 • preempt：负责根据优先级规则为同一队列中高优先级任务执行抢占调度。 • backfill：负责将处于pending状态的任务尽可能的调度下去以保证节点资源的最大化利用。	`actions: 'allocate, backfill, preempt'` **说明** 配置action时，preempt和enqueue不可同时使用。
plugins	根据不同场景提供了action中算法的具体实现细节，详情请参见**Plugins**。	支持的参数值请参见**表14-6**。	-

插件	功能	参数说明	用法演示
tolerations	插件实例对节点污点的容忍度设置。	默认配置下，插件实例可以运行在拥有“node.kubernetes.io/not-ready”或“node.kubernetes.io/unreachable”污点，且污点效果值为NoExecute的节点上，但会在60秒后被驱逐。	tolerations: - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 60 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 60

表 14-6 支持的 Plugins 列表

插件	功能	参数说明	用法演示
binpack	将Pod调度到资源使用较高的节点（尽量不往空白节点分配），以减少资源碎片。	arguments参数： ● binpack.weight：binpack插件本身在所有插件打分中的权重。 ● binpack.cpu：CPU资源在所有资源中的权重，默认是1。 ● binpack.memory：内存资源在所有资源中的权重，默认是1。 ● binpack.resources：Pod请求的其他自定义资源类型，例如nvidia.com/gpu。可添加多个并用英文逗号隔开。 ● binpack.resources.*<your_resource>*：自定义资源在所有资源中的权重，可添加多个类型的资源，其中*<your_resource>*为binpack.resources参数中定义的资源类型。例如binpack.resources.nvidia.com/gpu。	- plugins: - **name: binpack** arguments: binpack.weight: 10 binpack.cpu: 1 binpack.memory: 1 binpack.resources: nvidia.com/gpu, example.com/foo binpack.resources.nvidia.com/gpu: 2 binpack.resources.example.com/foo: 3

插件	功能	参数说明	用法演示

conformance	跳过关键Pod（比如在kube-system命名空间的Pod），防止这些Pod被驱逐。	-	- plugins:
			  - name: 'priority'
			  - name: 'gang'
			  enablePreemptable: false
			  - **name: 'conformance'**

lifecycle	通过统计业务伸缩的规律，将有相近生命周期的Pod优先调度到同一节点，配合autoscaler的水平扩缩容能力，快速缩容释放资源，节约成本并提高资源利用率。
	1. 统计业务负载中Pod的生命周期，将有相近生命周期的Pod调度到同一节点
	2. 对配置了自动扩缩容策略的集群，通过调整节点的缩容注解，优先缩容使用率低的节点
		arguments参数：
		● lifecycle.WindowSize：为int型整数，不小于1，默认为10。
		记录副本数变更的次数，负载变化规律、周期性明显时可适当调低；变化不规律，副本数频繁变化需要调大。若过大会导致学习周期变长，记录事件过多。
		● lifecycle.MaxGrade：为int型整数，不小于3，默认为3。
		副本分档数，如设为3，代表分为高中低档。负载变化规律、周期性明显时可适当调低；变化不规律，需要调大。若过小会导致预测的生命周期不够准确。
		● lifecycle.MaxScore：为float64浮点数，不小于50.0，默认为200.0。
		lifecycle插件的最大得分，等效于插件权重。
		● lifecycle.SaturatedTresh：为float64浮点数，小于0.5时取值为0.5；大于1时取值为1，默认为0.8。
		用于判断节点利用率是否过高的阈值，当超过该阈值，调度器会优先调度作业至其他节点。
			- plugins:
			  - name: priority
			  - name: gang
			  enablePreemptable: false
			  - name: conformance
			  - **name: lifecycle**
			  arguments:
			  lifecycle.MaxGrade: 3
			  lifecycle.MaxScore: 200.0
			  lifecycle.SaturatedTresh: 0.8
			  lifecycle.WindowSize: 10
			说明
			● 对不希望被缩容的节点，需要手动标记长周期节点，为节点添加volcano.sh/long-lifecycle-node: true的annotation。对未标记节点，lifecycle插件将根据节点上负载的生命周期自动标记。
			● MaxScore默认值200.0相当于其他插件权重的两倍，当lifecycle插件效果不明显或与其他插件冲突时，需要关闭其他插件，或将MaxScore调大。
			● 调度器重启后，lifecycle插件需要重新记录负载的变化状况，需要统计数个周期后才能达到最优调度效果。

插件	功能	参数说明	用法演示
gang	将一组Pod看做一个整体进行资源分配。观察Job下的Pod已调度数量是否满足了最小运行数量，当Job的最小运行数量得到满足时，为Job下的所有Pod执行调度动作，否则，不执行。 **说明** 使用gang调度策略时，当集群剩余的资源大于等于Job的最小运行数量的1/2、但小于Job的最小运行数量时，不会触发autoscaler扩容。	• enablePreemptable： – true：表示开启抢占。 – false：表示不开启抢占。 • enableJobStarving： – true: 表示按照Job的minAvailable进行抢占。 – false：表示按照Job的replicas进行抢占。 **说明** – Kubernetes原生工作负载（如Deployment）的minAvailable默认值为1，建议配置enableJobStarving: false。 – AI大数据场景，创建vcjob时可指定minAvailable值，推荐配置enableJobStarving: true。 – Volcano 1.11.5之前的版本enableJobStarving默认为true，1.11.5之后的版本默认配置为false。	- plugins: - name: priority **- name: gang** **enablePreemptable: false** **enableJobStarving: false** - name: conformance
priority	根据自定义的负载优先级进行调度。	-	- plugins: **- name: priority** - name: gang enablePreemptable: false - name: conformance
overcommit	将集群的资源放到一定倍数后调度，提高负载入队效率。负载都是deployment的时候，建议去掉此插件或者设置扩大因子为2.0。 **说明** 该插件在Volcano 1.6.5及以上版本中支持使用。	arguments参数： • overcommit-factor：扩大因子，默认是1.2。	- plugins: **- name: overcommit** arguments: overcommit-factor: 2.0

插件	功能	参数说明	用法演示

drf	DRF调度算法（Dominant Resource Fairness）可以根据作业使用的主导资源份额进行调度，资源份额较小的作业将具有更高优先级。	-	- plugins:
			  - **name: 'drf'**
			  - name: 'predicates'
			  - name: 'nodeorder'

predicates	预选节点的常用算法，包括节点亲和、Pod亲和、污点容忍、Node重复，volume limits，volume zone匹配等一系列基础算法。	-	- plugins:
			  - name: 'drf'
			  - **name: 'predicates'**
			  - name: 'nodeorder'

nodeorder	优选节点的常用算法，通过模拟分配从各个维度为节点打分，找到最适合当前作业的节点。	打分参数：
		● nodeaffinity.weight：节点亲和性优先调度，默认值是2。
		● podaffinity.weight：Pod亲和性优先调度，默认值是2。
		● leastrequested.weight：资源分配最少的节点优先，默认值是1。
		● balancedresource.weight：节点上面的不同资源分配平衡的优先，默认值是1。
		● mostrequested.weight：资源分配最多的节点优先，默认值是0。
		● tainttoleration.weight：污点容忍高的优先调度，默认值是3。
		● imagelocality.weight：节点上面有Pod需要镜像的优先调度，默认值是1。
		● podtopologyspread.weight：Pod拓扑调度，默认值是2。
			- plugins:
			  - **name: nodeorder**
			  arguments:
			  leastrequested.weight: 1
			  mostrequested.weight: 0
			  nodeaffinity.weight: 2
			  podaffinity.weight: 2
			  balancedresource.weight: 1
			  tainttoleration.weight: 3
			  imagelocality.weight: 1
			  podtopologyspread.weight: 2

插件	功能	参数说明	用法演示

cce-gpu-topology-predicate	GPU拓扑调度预选算法	-	- plugins:
			  - **name: 'cce-gpu-topology-predicate'**
			  - name: 'cce-gpu-topology-priority'
			  - name: 'cce-gpu'

cce-gpu-topology-priority	GPU拓扑调度优选算法	-	- plugins:
			  - name: 'cce-gpu-topology-predicate'
			  - **name: 'cce-gpu-topology-priority'**
			  - name: 'cce-gpu'

cce-gpu	结合CCE的GPU插件支持GPU资源分配，支持小数GPU配置。
	说明
	• 1.10.5及以上版本的插件不再支持该插件，请使用xgpu插件。
	• 小数GPU配置的前提条件为CCE集群GPU节点为共享模式，检查集群是否关闭GPU共享，请参见**修改CCE集群配置**中的enable-gpu-share参数。
		-	- plugins:
			  - name: 'cce-gpu-topology-predicate'
			  - name: 'cce-gpu-topology-priority'
			  - **name: 'cce-gpu'**

numa-aware	NUMA亲和性调度，详情请参见**NUMA亲和性调度**。	arguments参数：
		• weight：插件的权重。
			- plugins:
			  - name: 'nodelocalvolume'
			  - name: 'nodeemptydirvolume'
			  - name: 'nodeCSIscheduling'
			  - name: 'networkresource'
			  arguments:
			  NetworkType: vpc-router
			  - **name: numa-aware**
			  arguments:
			  weight: 10

network resource	支持预选过滤ENI需求节点，参数由CCE传递，不需要手动配置。	arguments参数：
		• NetworkType：网络类型（eni或者vpc-router类型）。
			- plugins:
			  - name: 'nodelocalvolume'
			  - name: 'nodeemptydirvolume'
			  - name: 'nodeCSIscheduling'
			  - **name: 'networkresource'**
			  arguments:
			  NetworkType: vpc-router

插件	功能	参数说明	用法演示

nodelocalvolume	支持预选过滤不符合local volume需求的节点。	-	- plugins:
			  - **name: 'nodelocalvolume'**
			  - name: 'nodeemptydirvolume'
			  - name: 'nodeCSIscheduling'
			  - name: 'networkresource'

nodeemptydirvolume	支持预选过滤不符合emptydir需求的节点。	-	- plugins:
			  - name: 'nodelocalvolume'
			  - **name: 'nodeemptydirvolume'**
			  - name: 'nodeCSIscheduling'
			  - name: 'networkresource'

nodeCSIscheduling	支持预选过滤everest组件异常的节点。	-	- plugins:
			  - name: 'nodelocalvolume'
			  - name: 'nodeemptydirvolume'
			  - **name: 'nodeCSIscheduling'**
			  - name: 'networkresource'

----结束

组件说明

表 14-7 Volcano 组件

容器组件	说明	资源类型
volcano-scheduler	负责Pod调度。	Deployment
volcano-controller	负责CRD资源的同步。	Deployment
volcano-admission	Webhook server端，负责Pod、Job等资源的校验和更改。	Deployment
volcano-agent	云原生混部agent，负责节点QoS保障、CPU Burst和动态资源超卖等。	DaemonSet
resource-exporter	负责上报节点的NUMA拓扑信息。	DaemonSet
volcano-descheduler	负责集群Pod重调度，开启重调度能力后自动部署。	Deployment
volcano-recommender	根据容器的历史CPU、Memory实际资源使用率生成容器资源申请量配置推荐值。	Deployment
volcano-recommender-prometheus-adapter	从Prometheus采集容器的历史CPU、Memory资源指标值。	Deployment

在控制台中修改 volcano-scheduler 配置

volcano-scheduler是负责Pod调度的组件，它由一系列action和plugin组成。action定义了调度各环节中需要执行的动作；plugin根据不同场景提供了action 中算法的具体实现细节。volcano-scheduler具有高度的可扩展性，您可以根据需要实现自己的action和plugin。

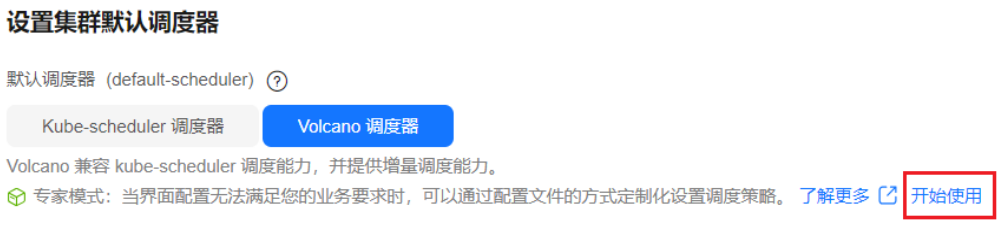
插件安装完成后，您可以单击左侧导航栏的“配置中心”，切换至“调度配置”页面进行基础调度能力设置。您也可以使用Volcano调度器的“专家模式”，结合实际业务场景定制专属的高阶调度策略。

当前小节介绍如何使用自定义配置，以便用户让volcano-scheduler能更适合自己的场景。

说明

仅Volcano 1.7.1及以上版本支持该功能。

您可登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“配置中心”，切换至“调度配置”页面，选择Volcano调度器找到对应的“专家模式”，单击“开始使用”。



- 使用resource_exporter配置，示例如下：

```
...
  "default_scheduler_conf": {
    "actions": "allocate, backfill, preempt",
    "tiers": [
      {
        "plugins": [
          {
            "name": "priority"
          },
          {
            "name": "gang"
          },
          {
            "name": "conformance"
          }
        ]
      },
      {
        "plugins": [
          {
            "name": "drf"
          },
          {
            "name": "predicates"
          },
          {
            "name": "nodeorder"
          }
        ]
      },
      {
```

```
            "plugins": [
               {
                  "name": "cce-gpu-topology-predicate"
               },
               {
                  "name": "cce-gpu-topology-priority"
               },
               {
                  "name": "cce-gpu"
               },
               {
                  "name": "numa-aware" # add this also enable resource_exporter
               }
            ]
         },
         {
            "plugins": [
               {
                  "name": "nodelocalvolume"
               },
               {
                  "name": "nodeemptydirvolume"
               },
               {
                  "name": "nodeCSIscheduling"
               },
               {
                  "name": "networkresource"
               }
            ]
         }
      ]
   },
...
```

开启后可以同时使用volcano-scheduler的numa-aware插件功能和resource_exporter功能。

Prometheus 指标采集

volcano-scheduler通过端口8080暴露Prometheus metrics指标。您可以自建Prometheus采集器识别并通过http://{{volcano-schedulerPodIP}}:{{volcano-schedulerPodPort}}/metrics路径获取volcano-scheduler调度相关指标。

说明

Prometheus指标暴露仅支持Volcano插件1.8.5及以上版本。

表 14-8 关键指标说明

指标名称	指标类型	描述	Labels
e2e_scheduling_latency_milliseconds	Histogram	端到端调度时延毫秒（调度算法+绑定）	-
e2e_job_scheduling_latency_milliseconds	Histogram	端到端作业调度时延（毫秒）	-
e2e_job_scheduling_duration	Gauge	端到端作业调度时长	labels=["job_name", "queue", "job_namespace"]

指标名称	指标类型	描述	Labels
plugin_scheduling_latency_microseconds	Histogram	插件调度延迟（微秒）	labels=["plugin", "OnSession"]
action_scheduling_latency_microseconds	Histogram	动作调度时延（微秒）	labels=["action"]
task_scheduling_latency_milliseconds	Histogram	任务调度时延（毫秒）	-
schedule_attempts_total	Counter	尝试调度Pod的次数。“unschedulable”表示无法调度Pod，而“error”表示内部调度器问题	labels=["result"]
pod_preemption_victims	Gauge	选定的抢占受害者数量	-
total_preemption_attempts	Counter	集群中的抢占尝试总数	-
unschedule_task_count	Gauge	无法调度的任务数	labels=["job_id"]
unschedule_job_count	Gauge	无法调度的作业数	-
job_retry_counts	Counter	作业的重试次数	labels=["job_id"]

Volcano 插件卸载说明

卸载插件时，会将Volcano的自定义资源（表14-9）全部清理，已创建的相关资源也将同步删除，重新安装插件不会继承和恢复卸载之前的任务信息。如集群中还存在正在使用的Volcano自定义资源，请谨慎卸载插件。

表 14-9 Volcano 自定义资源

名称	API组	API版本	资源级别
Command	bus.volcano.sh	v1alpha1	Namespaced
Job	batch.volcano.sh	v1alpha1	Namespaced
Numatopology	nodeinfo.volcano.sh	v1alpha1	Cluster
PodGroup	scheduling.volcano.sh	v1beta1	Namespaced

名称	API组	API版本	资源级别
Queue	scheduling.volcano.sh	v1beta1	Cluster
BalancerPolicyTemplate	autoscaling.volcano.sh	v1alpha1	Cluster
Balancer	autoscaling.volcano.sh	v1alpha1	Cluster

说明

BalancerPolicyTemplate和Balancer资源仅在开启应用扩缩容优先级策略后创建，详情请参见**应用扩缩容优先级策略**。

相关操作

- **动态资源超卖**
- **NUMA亲和性调度**

版本记录

须知

建议升级到跟集群配套的最新Volcano版本。

表 14-10 Volcano 调度器版本记录

插件版本	支持的集群版本	更新特性
1.15.6	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	新增基于应用资源画像的超卖能力
1.14.11	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 新增支持超节点资源调度模型（HyperJob） • 支持超节点亲和调度 • 支持Kubernetes v1.30

插件版本	支持的集群版本	更新特性
1.13.7	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	● 网卡资源调度支持前置预热 ● 支持自定义资源超卖比例
1.13.3	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	● 支持自定义资源按照节点优先级缩容 ● 优化抢占与节点扩容联动能力
1.13.1	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	调度器内存使用优化
1.12.18	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	● 适配CCE v1.29集群 ● 默认开启抢占功能
1.12.1	v1.19.16 v1.21 v1.23 v1.25 v1.27 v1.28	应用弹性扩缩容性能优化
1.11.21	v1.19.16 v1.21 v1.23 v1.25 v1.27 v1.28	● 支持Kubernetes v1.28 ● 支持负载感知调度 ● 镜像OS更新为HCE 2.0 ● 优化CSI资源抢占能力 ● 优化负载感知重调度能力 ● 优化混部场景抢占能力

插件版本	支持的集群版本	更新特性
1.11.6	v1.19.16 v1.21 v1.23 v1.25 v1.27	• 支持Kubernetes v1.27 • 支持重调度功能 • 支持节点池亲和调度能力 • 优化调度性能
1.10.7	v1.19.16 v1.21 v1.23 v1.25	修复本地持久卷插件未计算预绑定到节点的pod的问题
1.10.5	v1.19.16 v1.21 v1.23 v1.25	• volcano agent支持资源超卖。 • 添加针对GPU资源字段的校验admission：nvidia.com/gpu应小于1或者为正整数，volcano.sh/gpu-core.percentage应小于100并为5的倍数。 • 修复存在PVC绑定失败的场景下，后续提交Pod调度慢的问题。 • 修复节点上存在长时间Teminating Pod场景下，新提交Pod无法运行的问题。 • 修复并发创建挂载PVC的Pod的场景下，volcano重启的问题。
1.9.1	v1.19.16 v1.21 v1.23 v1.25	• 修复networkresource插件计数pipeline pod占用subeni问题 • 修复binpack插件对资源不足节点打分问题 • 修复对结束状态未知的Pod的资源的处理 • 优化事件输出 • 默认高可用部署
1.7.2	v1.19.16 v1.21 v1.23 v1.25	• Volcano支持v1.25集群 • 提升Volcano调度性能
1.7.1	v1.19.16 v1.21 v1.23 v1.25	Volcano支持v1.25集群

插件版本	支持的集群版本	更新特性
1.4.7	v1.15 v1.17 v1.19 v1.21	删除Pod状态Undetermined，以适配集群Autoscaler的弹性能力。
1.4.5	v1.17 v1.19 v1.21	volcano-scheduler的部署方式由StatefulSet调整为Deployment，修复节点异常时Pod无法自动迁移的问题
1.4.2	v1.15 v1.17 v1.19 v1.21	• 修复跨GPU分配失败问题 • 适配更新后的EAS API
1.3.7	v1.15 v1.17 v1.19 v1.21	• 支持在/离线作业混合部署及资源超卖功能 • 优化集群调度吞吐性能 • 修复特定场景下调度器panic的问题 • 修复CCE v1.15集群中volcano作业volumes.secret校验失败的问题 • 修复挂载volume，作业调度不成功的问题
1.3.3	v1.15 v1.17 v1.19 v1.21	修复GPU异常导致的调度器崩溃问题；修复特权Init容器准入失败问题
1.3.1	v1.15 v1.17 v1.19	• 升级Volcano框架到最新版本 • 支持Kubernetes v1.19版本 • 添加numa-aware插件 • 修复多队列场景下Deployment扩缩容的问题 • 调整默认开启的算法插件
1.2.5	v1.15 v1.17 v1.19	• 修复某些场景下OutOfcpu的问题 • 修复queue设置部分capability情况下Pod无法调度问题 • 支持volcano组件日志时间与系统时间保持一致 • 修复队列间多抢占问题 • 修复ioaware插件在某些极端场景下结果不符合预期的问题 • 支持混合集群

插件版本	支持的集群版本	更新特性
1.2.3	v1.15 v1.17 v1.19	• 修复因为精度不够引发的训练任务OOM的问题 • 修复CCE v1.15以上版本GPU调度的问题，暂不支持任务分发时的CCE版本滚动升级 • 修复特定场景下队列状态不明的问题 • 修复特定场景下作业挂载PVC panic的问题 • 修复GPU作业无法配置小数的问题 • 添加ioaware插件 • 添加ring controller

14.2.2 CCE 集群弹性引擎

插件简介

CCE集群弹性引擎（autoscaler）是Kubernetes中非常重要的一个Controller，它提供了微服务的弹性能力，并且和Serverless密切相关。

弹性伸缩是很好理解的一个概念，当微服务负载高（CPU/内存使用率过高）时水平扩容，增加pod的数量以降低负载，当负载降低时减少pod的数量，减少资源的消耗，通过这种方式使得微服务始终稳定在一个理想的状态。

云容器引擎简化了Kubernetes集群的创建、升级和手动扩缩容，而集群中应用的负载本身是会随着时间动态变化的，为了更好的平衡资源使用率以及性能，Kubernetes引入了autoscaler插件，它可以根据部署的应用所请求的资源量自动伸缩集群中节点数量，详情请了解**创建节点弹性策略**。

开源社区地址：https://github.com/kubernetes/autoscaler

插件说明

autoscaler可分成扩容和缩容两个方面：

- **自动扩容**

  集群的自动扩容有以下两种方式实现：

  - 当集群中的Pod由于工作节点资源不足而无法调度时，会触发集群扩容，扩容节点与所在节点池资源配额一致。

    此时需要满足以下条件时才会执行自动扩容：

    - 节点上的资源不足。
    - Pod的调度配置中不能包含节点亲和的策略（即Pod若已经设置亲和某个节点，则不会自动扩容节点），节点亲和策略设置方法请参见**设置节点亲和调度（nodeAffinity）**。
  - 当集群满足节点伸缩策略时，也会触发集群扩容，详情请参见**创建节点弹性策略**。

> **说明**
>
> 当前该插件使用的是最小浪费策略，即若Pod创建需要3核，此时有4核、8核两种规格，优先创建规格为4核的节点。

- **自动缩容**

  当集群节点处于一段时间空闲状态时（默认10min），会触发集群缩容操作（即节点会被自动删除）。当节点存在以下几种状态的Pod时，不可缩容：

  - Pod有设置Pod Disruption Budget（即**干扰预算**），当移除Pod不满足对应条件时，节点不会缩容。
  - Pod由于一些限制，如亲和、反亲和等，无法调度到其他节点，节点不会缩容。
  - Pod拥有cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'这个annotations时，节点不缩容。
  - 节点上存在kube-system命名空间下的Pod（除kube-system命名空间下由DaemonSet创建的Pod），节点不缩容。
  - 节点上如果有非controller（Deployment/ReplicaSet/Job/StatefulSet）创建的Pod，节点不缩容。

  > **说明**
  >
  > 当节点符合缩容条件时，Autoscaler将预先给节点打上DeletionCandidateOfClusterAutoscaler污点，限制Pod调度到该节点上。当autoscaler插件被卸载后，如果节点上依然存在该污点请您手动进行删除。

约束与限制

- 安装时请确保有足够的资源安装本插件。
- 该插件功能仅支持**虚拟机节点**，不支持物理机节点和裸金属服务器。
- 默认节点池不支持弹性扩缩容，详情请参见**默认节点池DefaultPool说明**。
- 缩容节点会导致与节点关联的**本地持久卷**类型的PVC/PV数据丢失，无法恢复，且PVC/PV无法再正常使用。缩容节点时使用了本地持久存储卷的Pod会从缩容的节点上被驱逐，并重新创建Pod，Pod会一直处于pending状态，因为Pod使用的PVC带有节点标签，由于冲突无法调度成功。
- 使用autoscaler插件时，部分污点/注解可能会影响弹性伸缩功能，因此集群中应避免使用以下污点/注解：
  - **节点避免使用ignore-taint.cluster-autoscaler.kubernetes.io的污点**：该污点作用于节点。由于autoscaler原生支持异常扩容保护策略，会定期评估集群的可用节点比例，非Ready分类节点数统计比例超过45%比例会触发保护机制；而集群中任何存在该污点的节点都将从自动缩放器模板节点中过滤掉，记录到非Ready分类的节点中，进而影响集群的扩缩容。
  - **Pod避免使用cluster-autoscaler.kubernetes.io/enable-ds-eviction的注解**：该注解作用于Pod，控制DaemonSet Pod是否可以被autoscaler驱逐。详情请参见**Kubernetes原生的标签、注解和污点**。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**CCE集群弹性引擎**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

CCE根据集群规模提供三种“系统预置规格”，您可根据自身需求进行选择，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

步骤3 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-11 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤4 配置完成后，单击“安装”。

----结束

组件说明

表 14-12 autoscaler 组件

容器组件	说明	资源类型
autoscaler	该容器为Kubernetes集群提供自动扩缩容节点的能力。	Deployment

版本记录

表 14-13 v1.30 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.30.18	v1.30	修复部分问题	1.30.1
1.30.15	v1.30	• 支持v1.30集群 • 事件增加节点池名称	1.30.1

表 14-14 v1.29 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.29.50	v1.29	事件增加节点池名称	1.29.1
1.29.17	v1.29	事件优化	1.29.1
1.29.13	v1.29	支持v1.29集群	1.29.1

表 14-15 v1.28 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.28.88	v1.28	事件增加节点池名称	1.28.1
1.28.55	v1.28	事件优化	1.28.1
1.28.51	v1.28	优化节点池资源售罄告警逻辑	1.28.1
1.28.22	v1.28	修复部分问题	1.28.1
1.28.20	v1.28	修复部分问题	1.28.1
1.28.17	v1.28	解决存在自定义控制器类型的Pod时无法缩容的问题	1.28.1

表 14-16 v1.27 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.27.119	v1.27	事件增加节点池名称	1.27.1
1.27.88	v1.27	事件优化	1.27.1
1.27.84	v1.27	优化节点池资源售罄告警逻辑	1.27.1
1.27.55	v1.27	修复部分问题	1.27.1
1.27.53	v1.27	修复部分问题	1.27.1
1.27.51	v1.27	修复部分问题	1.27.1
1.27.14	v1.27	修复多规格情况下无法缩容和非预期PreferNoSchedule污点问题	1.27.1

表 14-17 v1.25 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.25.152	v1.25	事件增加节点池名称	1.25.0
1.25.120	v1.25	事件优化	1.25.0
1.25.116	v1.25	优化节点池资源售罄告警逻辑	1.25.0
1.25.88	v1.25	修复部分问题	1.25.0
1.25.86	v1.25	修复部分问题	1.25.0
1.25.84	v1.25	修复部分问题	1.25.0
1.25.46	v1.25	修复多规格情况下无法缩容和非预期PreferNoSchedule污点问题	1.25.0
1.25.34	v1.25	• 优化异构设备(GPU/NPU)识别方法 • 扩容节点数量超过集群规模时，使用集群支持的剩余节点数量进行扩容	1.25.0
1.25.21	v1.25	• 修复autoscaler伸缩策略least-waste默认未启用的问题 • 修复节点池扩容失败后无法切换到其他节点池扩容且插件有重启动作的问题 • 默认污点容忍时长修改为60s • 扩容规则禁用后仍然触发扩容	1.25.0

插件版本	支持的集群版本	更新特性	社区版本
1.25.11	v1.25	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题	1.25.0
1.25.7	v1.25	• 适配CCE v1.25集群 • 修改自定义规格的内存申请与限制 • 当没有开启弹性伸缩的节点池时上报无法伸缩的事件 • 修复NPU节点在扩容过程中会再次触发扩容的问题	1.25.0

表 14-18 v1.23 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.23.156	v1.23	事件增加节点池名称	1.23.0
1.23.125	v1.23	事件优化	1.23.0
1.23.121	v1.23	优化节点池资源售罄告警逻辑	1.23.0
1.23.95	v1.23	修复部分问题	1.23.0
1.23.93	v1.23	修复部分问题	1.23.0
1.23.91	v1.23	修复部分问题	1.23.0
1.23.54	v1.23	修复多规格情况下无法缩容和非预期PreferNoSchedule污点问题	1.23.0
1.23.44	v1.23	• 优化异构设备(GPU/NPU)识别方法 • 扩容节点数量超过集群规模时，使用集群支持的剩余节点数量进行扩容	1.23.0
1.23.31	v1.23	• 修复autoscaler伸缩策略least-waste默认未启用的问题 • 修复节点池扩容失败后无法切换到其他节点池扩容且插件有重启动作的问题 • 默认污点容忍时长修改为60s • 扩容规则禁用后仍然触发扩容	1.23.0

插件版本	支持的集群版本	更新特性	社区版本
1.23.21	v1.23	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题	1.23.0
1.23.17	v1.23	• 适配NPU和安全容器 • 节点伸缩策略支持不设置步长 • bug修复，自动移除已删除的节点池 • 设置优先调度 • 注册EmptyDir调度策略 • 修复停用节点伸缩策略时，低于缩容阈值的节点未触发缩容的问题 • 修改自定义规格的内存申请与限制 • 当没有开启弹性伸缩的节点池时上报无法伸缩的事件 • 修复NPU节点在扩容过程中会再次触发扩容的问题	1.23.0
1.23.10	v1.23	• 日志优化 • 支持缩容等待，等待用户在节点删除前完成数据转储等操作	1.23.0
1.23.9	v1.23	新增nodenetworkconfigs.crd.yangtse.cni资源对象权限。	1.23.0
1.23.8	v1.23	修复周期性扩容场景下，单次扩容数量超过节点池上限，扩容失败问题。	1.23.0
1.23.7	v1.23	-	1.23.0
1.23.3	v1.23	适配CCE v1.23集群	1.23.0

表 14-19 v1.21 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.21.114	v1.21	优化节点池资源售罄告警逻辑	1.21.0
1.21.89	v1.21	修复部分问题	1.21.0

插件版本	支持的集群版本	更新特性	社区版本
1.21.87	v1.21	修复部分问题	**1.21.0**
1.21.86	v1.21	修复配置节点AZ拓扑约束时，节点池弹性扩容后不符合预期问题	**1.21.0**
1.21.51	v1.21	修复多规格情况下无法缩容和非预期PreferNoSchedule污点问题	**1.21.0**
1.21.43	v1.21	• 优化异构设备(GPU/NPU)识别方法 • 扩容节点数量超过集群规模时，使用集群支持的剩余节点数量进行扩容	**1.21.0**
1.21.29	v1.21	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题 • 修复节点池扩容失败后无法切换到其他节点池扩容且插件有重启动作的问题 • 默认污点容忍时长修改为60s • 扩容规则禁用后仍然触发扩容	**1.21.0**
1.21.20	v1.21	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题	**1.21.0**

插件版本	支持的集群版本	更新特性	社区版本
1.21.16	v1.21	• 适配NPU和安全容器 • 节点伸缩策略支持不设置步长 • bug修复，自动移除已删除的节点池 • 设置优先调度 • 注册EmptyDir调度策略 • 修复停用节点伸缩策略时，低于缩容阈值的节点未触发缩容的问题 • 修改自定义规格的内存申请与限制 • 当没有开启弹性伸缩的节点池时上报无法伸缩的事件 • 修复NPU节点在扩容过程中会再次触发扩容的问题	1.21.0
1.21.9	v1.21	• 日志优化 • 支持缩容等待，等待用户在节点删除前完成数据转储等操作	1.21.0
1.21.8	v1.21	新增nodenetworkconfigs.crd.yangtse.cni资源对象权限	1.21.0
1.21.6	v1.21	修复插件请求重试场景下签名错误导致鉴权失败的问题	1.21.0
1.21.4	v1.21	修复插件请求重试场景下签名错误导致鉴权失败的问题	1.21.0
1.21.2	v1.21	修复未注册节点删除失败可能阻塞弹性伸缩的问题	1.21.0
1.21.1	v1.21	修复在已有的周期弹性伸缩规则里节点池修改不生效的问题。	1.21.0

表 14-20 v1.19 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.19.76	v1.19	• 优化异构设备(GPU/NPU)识别方法 • 扩容节点数量超过集群规模时，使用集群支持的剩余节点数量进行扩容	1.19.0

插件版本	支持的集群版本	更新特性	社区版本
1.19.56	v1.19	修复多规格情况下无法缩容和非预期PreferNoSchedule污点问题	1.19.0
1.19.48	v1.19	• 优化异构设备(GPU/NPU)识别方法 • 扩容节点数量超过集群规模时，使用集群支持的剩余节点数量进行扩容	1.19.0
1.19.35	v1.19	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题 • 修复节点池扩容失败后无法切换到其他节点池扩容且插件有重启动作的问题 • 默认污点容忍时长修改为60s • 扩容规则禁用后仍然触发扩容	1.19.0
1.19.27	v1.19	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 修复伸缩组资源不足时无法正常修复节点池数量问题	1.19.0
1.19.22	v1.19	• 适配NPU和安全容器 • 节点伸缩策略支持不设置步长 • bug修复，自动移除已删除的节点池 • 设置优先调度 • 注册EmptyDir调度策略 • 修复停用节点伸缩策略时，低于缩容阈值的节点未触发缩容的问题 • 修改自定义规格的内存申请与限制 • 当没有开启弹性伸缩的节点池时上报无法伸缩的事件 • 修复NPU节点在扩容过程中会再次触发扩容的问题	1.19.0

插件版本	支持的集群版本	更新特性	社区版本
1.19.14	v1.19	• 日志优化 • 支持缩容等待，等待用户在节点删除前完成数据转储等操作	1.19.0
1.19.13	v1.19	修复周期性扩容场景下，单次扩容数量超过节点池上限，扩容失败问题	1.19.0
1.19.12	v1.19	修复插件请求重试场景下签名错误导致鉴权失败的问题	1.19.0
1.19.11	v1.19	修复插件请求重试场景下签名错误导致鉴权失败的问题	1.19.0
1.19.9	v1.19	修复未注册节点删除失败可能阻塞弹性伸缩的问题	1.19.0
1.19.8	v1.19	修复在已有的周期弹性伸缩规则里节点池修改不生效的问题。	1.19.0
1.19.7	v1.19	插件依赖例行升级。	1.19.0
1.19.6	v1.19	修复污点异步更新场景触发的重复扩容问题。	1.19.0
1.19.3	v1.19	定时策略中能够根据节点总数，CPU，内存限制进行扩缩容。修复其它功能缺陷。	1.19.0

表 14-21 v1.17 集群配套插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.17.27	v1.17	• 日志优化 • bug修复，自动移除已删除的节点池 • 设置优先调度 • 修复刚扩容出的节点打污点会被覆盖的问题 • 修复停用节点伸缩策略时，低于缩容阈值的节点未触发缩容的问题 • 修改自定义规格的内存申请与限制 • 当没有开启弹性伸缩的节点池时上报无法伸缩的事件	1.17.0

插件版本	支持的集群版本	更新特性	社区版本
1.17.22	v1.17	日志优化	1.17.0
1.17.21	v1.17	修复周期性扩容场景下，单次扩容数量超过节点池上限，扩容失败问题	1.17.0
1.17.19	v1.17	修复插件请求重试场景下签名错误导致鉴权失败的问题	1.17.0
1.17.17	v1.17	修复未注册节点删除失败可能阻塞弹性伸缩的问题	1.17.0
1.17.16	v1.17	修复在已有的周期弹性伸缩规则里节点池修改不生效的问题。	1.17.0
1.17.15	v1.17	资源规格配置单位统一化	1.17.0
1.17.14	v1.17	修复Taints异步更新场景触发的重复扩容问题。	1.17.0
1.17.8	v1.17	Bug修复	1.17.0
1.17.7	v1.17	添加日志内容，Bug修复	1.17.0
1.17.5	v1.17	支持v1.17版本的集群，支持页面显示伸缩事件	1.17.0
1.17.2	v1.17	支持v1.17版本的集群	1.17.0

14.2.3 CCE 容器弹性引擎

CCE容器弹性引擎（cce-hpa-controller）插件是一款CCE自研的插件，能够基于CPU利用率、内存利用率等指标，对无状态工作负载进行弹性扩缩容。

安装本插件后，可创建CronHPA定时策略及CustomedHPA策略，具体请参见**创建CronHPA定时策略**或**创建CustomedHPA策略**。

主要功能

- 支持按照当前实例数的百分比进行扩缩容。
- 支持设置一次扩缩容的最小步长。
- 支持按照实际指标值执行不同的扩缩容动作。

约束与限制

- 若cce-hpa-controller版本低于1.2.11，则必须安装**prometheus**插件；若版本大于或等于1.2.11，则需要安装能够提供Metrics API的插件，您可根据集群版本和实际需求选择其中之一：
  - **Kubernetes Metrics Server**：提供基础资源使用指标，例如容器CPU和内存使用率。所有集群版本均可安装。

- **云原生监控插件**：该插件支持v1.17及以后的集群版本。
  - 根据基础资源指标进行弹性伸缩：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供基础资源指标**。
  - 根据自定义指标进行弹性伸缩：需要将自定义指标聚合到Kubernetes API Server，详情请参见**使用自定义指标创建HPA策略**。
- **Prometheus（停止维护）**：需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供资源指标**。该插件仅支持v1.21及之前的集群版本。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE容器弹性引擎**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据CCE推荐的预置值设置插件规格，可满足大多数场景，具体数值请以控制台显示为准。
- 选择“自定义规格”时，您可根据需求修改插件各个组件的副本数以及CPU/内存配置。

说明

副本数：副本数为1时插件不具备高可用能力，仅用于验证场景，商用场景请根据集群规格配置多个副本数。

CPU/内存配额：组件的资源配额主要受集群中总容器数量和伸缩策略数量影响。通常场景下，建议集群中每5000个容器配置CPU 500m、内存1000Mi，每1000条伸缩策略配置CPU 100m、内存500Mi。

步骤3 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-22 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。

参数	参数说明
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤4 单击“安装”。

----结束

组件说明

表 14-23 插件组件

容器组件	说明	资源类型
customedhpa-controller	CCE自研的弹性伸缩组件，可基于CPU利用率、内存利用率等指标，对无状态工作负载进行弹性扩缩容。	Deployment

版本记录

表 14-24 CCE 容器弹性引擎插件版本记录

插件版本	支持的集群版本	更新特性
1.5.3	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	支持AHPA
1.4.30	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30集群
1.4.3	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题
1.4.2	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群
1.3.43	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题

插件版本	支持的集群版本	更新特性
1.3.42	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28集群
1.3.14	v1.19 v1.21 v1.23 v1.25 v1.27	适配CCE v1.27集群
1.3.10	v1.19 v1.21 v1.23 v1.25	周期规则不受冷却时间影响定时触发
1.3.7	v1.19 v1.21 v1.23 v1.25	支持插件实例AZ反亲和配置
1.3.3	v1.19 v1.21 v1.23 v1.25	• 适配CCE v1.25集群 • CronHPA调整Deployment实例数，新增skip场景
1.3.1	v1.19 v1.21 v1.23	适配CCE v1.23集群
1.2.12	v1.15 v1.17 v1.19 v1.21	插件性能优化，降低资源消耗
1.2.11	v1.15 v1.17 v1.19 v1.21	• 从K8s Metrics API查询资源指标 • 计算资源利用率时考虑未就绪的Pod

插件版本	支持的集群版本	更新特性
1.2.10	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群
1.2.4	v1.15 v1.17 v1.19	• 插件依赖例行升级 • 支持配置插件资源规格
1.2.3	v1.15 v1.17 v1.19	适配ARM64节点部署
1.2.2	v1.15 v1.17 v1.19	增强健康检查能力
1.2.1	v1.15 v1.17 v1.19	• 适配CCE v1.19集群 • 更新插件为稳定版本
1.1.3	v1.15 v1.17	支持周期扩缩容规则

14.2.4 CCE 突发弹性引擎（对接 CCI）

CCE突发弹性引擎（对接 CCI）作为一种虚拟的kubelet用来连接Kubernetes集群和其他平台的API。Bursting的主要场景是将Kubernetes API扩展到无服务器的容器平台（如CCI）。

基于该插件，支持用户在短时高负载场景下，将部署在云容器引擎CCE上的无状态负载（Deployment）、有状态负载（StatefulSet）、普通任务（Job）、定时任务（CronJob）四种资源类型的容器实例（Pod），弹性创建到**云容器实例CCI**服务上，以减少集群扩容带来的消耗。

安装插件

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“插件中心”，进入插件中心首页。
4. 选择“CCE 突发弹性引擎 (对接 CCI)”插件，单击“安装”。
5. 配置插件参数。

表 14-25 插件参数说明

插件参数	说明
选择版本	插件的版本。插件版本和CCE集群存在配套关系，更多信息可以参考**CCE突发弹性引擎（对接CCI）插件版本记录**。

规格配置	用于配置插件负载的实例数及资源配额。
	• 选择“系统预置规格”时，您可选择“单实例”或“高可用”规格。
	• 选择“自定义规格”时，您可根据需求修改插件各个组件的副本数以及CPU/内存配置。
	说明
	• CCE 突发弹性引擎 (对接 CCI) 插件在1.5.2及以上版本，将占用更多节点资源，请在升级CCE突发弹性引擎（对接 CCI）插件前预留空间配额。
	  • 单实例：需要预留一个节点，节点下至少需要有7个Pod空间配额。若开启网络互通，则需要有8个Pod空间配额。
	  • 高可用：需要预留两个节点，节点下至少需要有7个Pod空间配额，共计14个Pod空间配额。若开启网络互通，则需要有8个Pod空间配额，共计16个Pod空间配额。
	• 弹性到CCI的业务量不同时，插件的资源占用也不相同。业务申请的POD、Secret、Congfigmap、PV、PVC会占用虚机资源。建议用户评估自己的业务使用量，按以下规格申请对应的虚机大小：1000pod+1000CM（300KB）推荐2U4G规格节点，2000pod+2000CM推荐4U8G规格节点，4000pod+4000CM推荐8U16G规格节点。

插件参数	说明
网络互通	开启后，支持CCE集群中的Pod与CCI集群中的Pod通过Kubernetes Service互通，并在插件安装时部署组件proxy。详细功能介绍请参考**网络**。

工作负载下发

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“工作负载”，进入工作负载首页。
4. 单击“创建工作负载”，具体操作步骤详情请参见**创建工作负载**。
5. 填写基本信息。“CCI弹性承载”选择“强制调度策略”。关于调度策略更多信息，请参考**调度负载到CCI**。

⚠ 注意

CCE集群创建工作负载时，需要弹性到CCI，健康检查不支持配置TCP启动探针。

6. 进行容器配置。
7. 配置完成后，单击“创建工作负载”。
8. 在工作负载页面，选择工作负载名称，单击进入工作负载管理界面。
9. 工作负载所在节点为CCI集群，说明负载成功已调度到CCI。

插件卸载

1. 登录CCE控制台。
2. 选择CCE集群，单击进入CCE集群总览页面。
3. 在导航栏左侧单击“插件中心”，进入插件中心首页。
4. 选择“CCE 突发弹性引擎 (对接 CCI)”插件，单击“卸载”。

表 14-26 特殊场景说明

特殊场景描述	场景现象	场景说明
CCE集群无节点，卸载插件。	插件卸载失败。	bursting插件卸载时会在集群中启动Job用于清理资源，卸载插件时请保证集群中至少有一个可以调度的节点。
用户直接删除集群，未卸载插件。	用户在CCI侧的命名空间中有资源残留，如果命名空间有计费资源，会造成额外计费。	由于直接删除集群，没有执行插件的资源清理Job，造成资源残留。用户可以手动清除残留命名空间及其下的计费资源来避免额外计费。

关于CCE突发弹性引擎（对接CCI）更多内容详情请参见：**CCE突发弹性引擎（对接CCI）**。

版本记录

表 14-27 CCE 突发弹性引擎（对接 CCI）插件版本记录

插件版本	支持的集群版本	更新特性
1.5.16	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	仅进行Pod级别CPU和Memory资源规整

插件版本	支持的集群版本	更新特性
1.5.8	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群
1.3.57	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28集群
1.3.48	v1.21 v1.23 v1.25 v1.27	• 支持v1.25、v1.27版本集群 • 支持JuiceFS类型的存储
1.3.25	v1.17 v1.19 v1.21 v1.23	• 支持DownwardAPI Volume • 支持Projected Volume • 支持自定义StorageClass
1.3.19	v1.17 v1.19 v1.21 v1.23	支持schedule profile
1.3.7	v1.17 v1.19 v1.21 v1.23	支持v1.21、v1.23版本集群
1.2.12	v1.13 v1.15 v1.17 v1.19	• 新增了部分metrics指标 • 支持HPA与CustomedHPA • 支持将弹性到CCI的Pod中的hostPath转换为其它类型存储 • 修复Kubernetes Dashboard无法使用终端问题

插件版本	支持的集群版本	更新特性
1.2.5	v1.13 v1.15 v1.17 v1.19	• 自动清理CCI中不再被Pod依赖的资源 • 支持配置Requests与Limits不相等，弹性到CCI时的资源申请量以Limits为准 • 修复CCI命名空间不存在时插件卸载失败问题 • 增加对Pod规格超过CCI限制的创建请求的拦截
1.2.0	v1.13 v1.15 v1.17 v1.19	• 支持v1.19版本集群 • 支持SFS、SFS Turbo类型存储 • 支持CronJob • 支持配置envFrom • 日志文件自动转储 • 屏蔽TCPSocket类型健康检查 • 支持配置资源标签（pod-tag） • 提升了性能和可靠性 • 修复了一些已知问题
1.0.5	v1.13 v1.15 v1.17	支持v1.17版本集群

14.2.5 容器垂直弹性引擎

CCE容器垂直弹性引擎是一款支持应用垂直弹性伸缩（VPA，Vertical Pod Autoscaling）的插件，可以根据容器资源历史使用情况自动调整Pod的CPU、Memory资源申请量。

开源社区地址：**https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler**

功能概述

VPA以容器为单位对资源指标进行聚合计算，根据容器的资源实际使用情况动态调整容器的资源申请值（Requests），同时保证调整前和调整后资源限制值（Limits）与资源申请值（Requests）的比值不变。目前支持CPU与Memory两类资源的垂直伸缩。

详细功能说明如下：

- VPA计算CPU与Memory建议值时需要数依赖Metrics API采集的数据。
- VPA在计算资源建议值时，Memory资源的单Pod最小理论建议值250Mi，Pod内单容器的最小理论建议值为250Mi/Pod容器数目。CPU资源的单Pod最小理论建议值为25m，Pod内单容器的最小理论建议值为25m/Pod容器数目。

您可在创建VPA任务时，通过配置containerPolicies字段为容器配置弹性资源上下限。

- 如果容器初始时同时配置了资源申请值与限制值，VPA计算后给出的建议值会修改该容器的资源申请值，而限制值则根据容器初始创建时申请值与限制值的比例进行计算。

  例如，某个容器原来配置了CPU资源申请值为100m与限制值为200m，申请值与限制值的比例为1:2。如果VPA计算后的资源申请值建议为80m，则该容器最终的CPU资源申请值为80m，限制值为160m。

- VPA会尽量让建议值符合其他资源限制要求。但如果VPA建议值与资源限制出现冲突，VPA建议值不会根据资源限制进行调整，可能导致VPA配置值超出其他资源限制要求。

  例如，某一个命名空间的内存申请值不能超过2GiB，而VPA的建议值如果比较大，可能导致Pod更新后整个命名空间的资源申请量超过2GiB从而出现无法调度。

前提条件

- 集群版本需满足v1.25及以上。
- 使用VPA需要在集群中安装能够提供Metrics API的插件，您可根据实际需求选择其中之一：
  - **Kubernetes Metrics Server**：提供基础资源使用指标，例如容器CPU和内存使用率。
  - **云原生监控插件**：使用Prometheus提供基础资源使用指标，需将Prometheus注册为Metrics API的服务，详见**通过Metrics API提供基础资源指标**。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**容器垂直弹性引擎**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据集群Pod数量选择“小规格”、“中规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-28 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤4 单击“安装”。

----结束

组件说明

表 14-29 CCE 容器垂直弹性引擎组件

容器组件	说明	资源类型
vpa-admission-controller	Pod创建时，将容器的资源申请量调整为VPA生成的建议值。	Deployment
vpa-recommender	采集容器的CPU、Memory的实际资源指标，根据实际资源使用率生成容器资源申请量配置建议值。	Deployment

容器组件	说明	资源类型
vpa-updater	驱逐实际资源申请量与VPA建议值有偏差的Pod，触发Pod重建以使得资源建议值生效至新建的Pod。	Deployment

版本记录

表 14-30 容器垂直弹性引擎版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.0.4	v1.25 v1.27 v1.28 v1.29 v1.30	支持VPA能力	1.1.2

14.3 云原生可观测性插件

14.3.1 云原生监控插件

插件简介

云原生监控插件（kube-prometheus-stack）通过使用Prometheus-operator和Prometheus，提供简单易用的端到端Kubernetes集群监控能力。

使用kube-prometheus-stack可将监控数据与容器智能分析对接，在容器智能分析控制台查看监控数据，配置告警等。

开源社区地址：**https://github.com/prometheus/prometheus**

约束与限制

- 在默认配置下，插件中的kube-state-metrics组件不采集Kubernetes资源的所有的labels和annotation。如需采集，您需要手动在启动参数中开启采集开关，并同时检查名称为kube-state-metrics的ServiceMonitor中采集白名单是否添加相应指标，详情请参见**采集Pod所有labels和annotations**。
- 自3.8.0版本起，自定义指标采集将默认不再采集kube-system和monitoring命名空间下的组件指标，若您有相关负载在这两个命名空间下，建议使用**Pod Monitor**或**Service Monitor**的方式采集。
- 自3.8.0版本起，默认不再采集etcd-server、kube-controller、kube-scheduler、autoscaler、fluent-bit、volcano-agent、volcano-scheduler、otel-collector的指标，您可按需开启。

开启方式：前往“配置项与密钥”页面并切换至monitoring命名空间，单击名为persistent-user-config的配置项的“编辑YAML”按钮，按需移除customSettings字段下serviceMonitorDisable或podMonitorDisable中的配置或置为空数组。

```
...
  customSettings:
    podMonitorDisable: []
    serviceMonitorDisable: []
```

- 在3.9.0版本后，Grafana组件从云原生监控插件中移除，拆分为独立的Grafana插件。因此，当插件版本从3.9.0后的版本回滚至3.9.0前的版本，请先卸载Grafana插件再进行回滚操作。

权限说明

云原生监控插件中的node-exporter组件会监控Docker的存储磁盘空间，需要读取宿主机的/var/run/docker.sock的获取Docker的info的数据。

node-exporter运行需要以下特权：

- cap_dac_override：读取Docker的info的数据。

安装插件

说明

云原生监控插件当前根据**数据存储配置**自适应选择部署模式（3.7.1及以上版本插件支持），具体如下：

- 原agent模式：关闭**本地数据存储**，且**监控数据上报至AOM服务**和**监控数据上报至第三方监控平台**至少开启其中之一。
- 原server模式：开启**本地数据存储**，同时支持开启**监控数据上报至AOM服务**或**监控数据上报至第三方监控平台**。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**云原生监控插件**，单击“安装”。

步骤2 在安装插件页面，根据需求选择“数据存储配置”，至少需要开启一项。

- **监控数据上报至AOM服务**：将普罗数据上报至 AOM 服务。开启后，可选择对应的AOM实例。采集的基础指标免费，自定义指标将由AOM服务进行收费，详情请参见**价格详情**。对接AOM需要用户具备一定权限，目前仅**华为云/华为账号**，或者在**admin**用户组下的用户支持此操作。
- **监控数据上报至第三方监控平台**：将普罗数据上报至第三方监控系统，需填写第三方监控系统的地址和Token，并选择是否跳过证书认证。
- **本地数据存储**：将普罗数据存储在集群中的PVC存储卷里，选择用于存储监控数据的磁盘类型和大小。**存储卷不随插件卸载而删除**。开启本地数据存储时，将部署全量组件，详情请参见**组件说明**。

  说明

  若monitoring命名空间下已存在可使用的PVC（名称为pvc-prometheus-server），将使用该存储作为存储源。

步骤3 根据需求选择“规格配置”。

- **插件规格**：
  - 选择“系统预置规格”时，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  - 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

- **普罗高可用**：高可用会在集群中将Prometheus-server、Prometheus-operator、thanos-query、custom-metrics-apiserver、alertmanager、kube-state-metrics组件按多实例方式部署。
- **采集分片数**（选择非“本地数据存储”时支持设置）：当Prometheus的数据量很大时，您可以通过设置该参数，将数据分片到指定数量的Prometheus实例上存储和查询。增加分片数量可以使每个分片承担的数据量更少，从而增加指标的采集吞吐上限，但也会消耗更多的资源。建议在集群规模较大时适度增加分片数量，提高采集性能，同时也需要考虑资源占用的影响，根据具体的监控场景进行权衡和调优。
- **安装grafana**：通过 grafana 可视化浏览普罗监控数据。grafana 会默认创建大小为 5 GiB 的存储卷，卸载插件时 grafana 的**存储卷不随插件被删除**。首次登录默认用户名与密码均为 admin，登录后会立即让您修改密码。

  3.9.0版本后，Grafana组件从云原生监控插件中移除，拆分为独立的Grafana插件，不再显示此选项。

步骤4 设置插件支持的“参数配置”。

- **自定义指标采集**：以服务发现的形式自动采集应用的指标。开启后需要在目标应用添加相关配置，详情请参见**使用云原生监控插件监控自定义指标**。
- **采集周期**：设置采集时间间隔周期。
- **数据保留期**（选择“本地数据存储”时支持设置）：监控数据保留的时长。
- **node-exporter监听端口**：该端口使用主机网络，用于监听并暴露所在节点的指标供普罗采集；默认为9100，若与您已有应用的端口冲突，可按需修改。

步骤5 完成以上配置后，单击“安装”。

插件安装完成后，根据您的使用需求，可能还需进行以下操作：

- 如需使用自定义指标创建弹性伸缩策略，请确认云原生监控插件的数据存储配置为开启本地数据存储的模式，然后参考以下步骤：

  a. 采集应用上报的自定义指标至Prometheus，详情请参见**使用云原生监控插件监控自定义指标**。

  b. 将Prometheus采集到的自定义指标聚合到API Server，可供HPA策略使用，详情请参见**使用自定义指标创建HPA策略**。
- 如果您需要使用该插件为工作负载弹性伸缩提供系统资源指标（如CPU、内存使用量），请确认云原生监控插件的数据存储配置为开启本地数据存储的模式，然后开启Metric API，详情请参见**通过Metrics API提供基础资源指标**。配置完成后，可使用Prometheus采集系统资源指标。（该操作可能与Kubernetes Metric Server插件产生冲突，不推荐）

----结束

组件说明

安装云原生监控插件创建的Kubernetes资源，全部都创建在monitoring命名空间下。

表 14-31 云原生监控插件的组件列表

容器组件	说明	支持的部署模式	资源类型
prometheusOperator（负载名称：prometheus-operator）	根据自定义资源（Custom Resource Definition / CRDs）来部署和管理Prometheus Server，同时监控这些自定义资源事件的变化来做相应的处理，是整个系统的控制中心。	所有模式	Deployment
prometheus（负载名称：prometheus-server）	Operator根据自定义资源Prometheus类型中定义的内容而部署Prometheus Server集群，这些自定义资源可以看作是用来管理Prometheus Server集群的StatefulSets资源。	所有模式	StatefulSet
alertmanager（负载名称：alertmanager-alertmanager）	插件的告警中心，主要用于接收Prometheus发送的告警并通过去重、分组、分发等能力管理告警信息。	本地数据存储开启模式	StatefulSet
thanosSidecar	仅在高可用模式下部署。和prometheus-server运行在同一个Pod中，用于实现普罗指标数据的持久化存储。	本地数据存储开启模式	Container
thanosQuery	仅在高可用模式下部署。PromQL查询的入口，能够对来自Store或Prometheus的相同指标进行重复数据删除。	本地数据存储开启模式	Deployment
adapter（负载名称：custom-metrics-apiserver）	将自定义指标聚合到原生的Kubernetes API Server。	本地数据存储开启模式	Deployment
kubeStateMetrics（负载名称：kube-state-metrics）	将Prometheus的metrics数据格式转换成K8s API接口能识别的格式。kube-state-metrics组件在默认配置下，不采集K8s资源的所有labels和annotation。如需采集，请参考**采集Pod所有labels和annotations**进行配置。 **说明** 该组件如果存在多个Pod，只会有一个Pod暴露指标。	所有模式	Deployment
nodeExporter（负载名称：node-exporter）	每个节点上均有部署，收集Node级别的监控数据。	所有模式	DaemonSet

容器组件	说明	支持的部署模式	资源类型
grafana （负载名称：grafana）	可视化浏览普罗监控数据。grafana会默认创建大小为5 GiB的存储卷，卸载插件时grafana的存储卷不随插件被删除。	所有模式	Deployment
clusterProblemDetector （负载名称：cluster-problem-detector）	用于监控集群异常。	本地数据存储开启模式	Deployment

通过 Metrics API 提供基础资源指标

说明

仅云原生监控插件开启本地数据存储时，可通过Metrics API提供基础资源指标。

容器和节点的资源指标，如CPU、内存使用量，可通过Kubernetes的Metrics API获得。这些指标可以直接被用户访问，比如用kubectl top命令，也可以被HPA或者CustomedHPA使用，根据资源使用率使负载弹性伸缩。

插件可为Kubernetes提供Metrics API，但默认未开启，若要将其开启，需要创建以下APIService对象：

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  labels:
    app: custom-metrics-apiserver
    release: cceaddon-prometheus
  name: v1beta1.metrics.k8s.io
spec:
  group: metrics.k8s.io
  groupPriorityMinimum: 100
  insecureSkipTLSVerify: true
  service:
    name: custom-metrics-apiserver
    namespace: monitoring
    port: 443
  version: v1beta1
  versionPriority: 100
```

可以将该对象保存为文件，命名为metrics-apiservice.yaml，然后执行以下命令：

```
kubectl create -f metrics-apiservice.yaml
```

执行kubectl top pod -n monitoring命令，若显示如下，则表示Metrics API能正常访问：

```
# kubectl top pod -n monitoring
NAME                                          CPU(cores)   MEMORY(bytes)
......
custom-metrics-apiserver-d4f556ff9-l2j2m                38m          44Mi
......
```


须知

卸载插件时，需要执行以下kubectl命令，同时删除APIService对象，否则残留的APIService资源将导致Kubernetes Metrics Server插件安装失败。

```
kubectl delete APIService v1beta1.metrics.k8s.io
```

使用自定义指标创建 HPA 策略

云原生监控插件为开启本地数据存储时，才能使用自定义指标HPA功能，您可在user-adapter-config配置项中配置HPA弹性策略需要的自定义指标。

须知

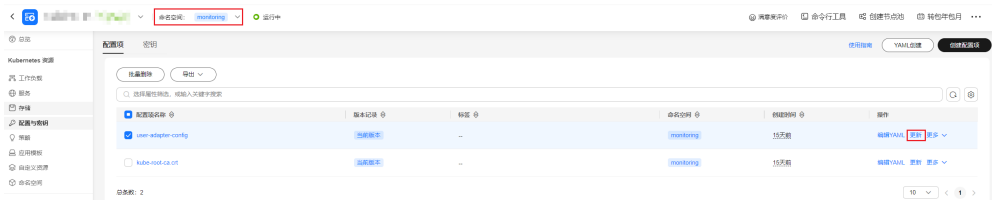
使用Prometheus监控自定义指标时，应用程序需要提供监控指标接口，详情请参见**Prometheus监控数据采集说明**。

以下案例中使用**使用云原生监控插件监控自定义指标**中的nginx指标（nginx_connections_accepted）作为配置示例。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“配置与密钥”。

步骤2 切换至“monitoring”命名空间，在“配置项”页签找到user-adapter-config配置项（或adapter-config），并单击“更新”。

图 14-1 更新配置项



步骤3 在“配置数据”中单击config.yaml对应的“编辑”按钮，在rules字段下添加自定义指标采集规则。修改完成后单击“确定”保存配置。

如果您需要增加多个采集规则，可在rules字段下添加多个配置，关于采集规则配置详情请参见**Metrics Discovery and Presentation Configuration**。

自定义采集规则示例如下：

```
rules:
# 匹配指标名称是nginx_connections_accepted的指标，必须确认指标名称，否则HPA控制器无法获取到指标
- seriesQuery: '{__name__=~"nginx_connections_accepted",container!="POD",namespace!="",pod!=""}'
  resources:
    # 指定Pod和命名空间资源
    overrides:
      namespace:
        resource: namespace
      pod:
        resource: pod
  name:
    #使用nginx_connections_accepted"
    matches: "nginx_connections_accepted"
    #使用nginx_connections_accepted_per_second来代表该指标，该名称即在HPA的自定义策略中的自定义指标名称
```

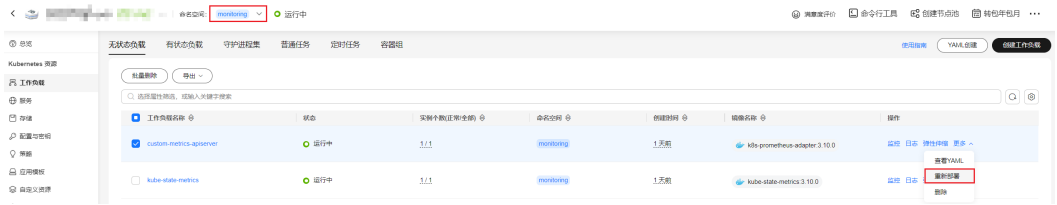


```
  as: "nginx_connections_accepted_per_second"
  #通过计算表达式rate(nginx_connections_accepted[2m])来代表是每秒的请求接收量
metricsQuery: 'rate(<<.Series>>{<<.LabelMatchers>>,container!="POD"}[2m])'
```

图 14-2 修改配置数据

步骤4 重新部署monitoring命名空间下的custom-metrics-apiserver工作负载。

图 14-3 重新部署 custom-metrics-apiserver



步骤5 在左侧导航栏中选择“工作负载”，找到需要创建HPA策略的工作负载单击“更多>弹性伸缩”。您可在“自定义策略”中选择上述参数创建弹性伸缩策略。

图 14-4 创建 HPA 策略

----结束

采集 Pod 所有 labels 和 annotations

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“工作负载”。

步骤2 切换至“monitoring”命名空间，在“无状态负载”页签单击进入**kube-state-metrics**负载，选择“容器管理”页签，在右侧单击“编辑”按钮，进入“升级工作负载”页面。

步骤3 在容器配置的“生命周期”中，编辑启动命令。

图 14-5 编辑启动命令

采集labels时，在原有的kube-state-metrics的启动参数最后添加：

```
--metric-labels-allowlist=pods=[*],nodes=[node,failure-domain.beta.kubernetes.io/zone,topology.kubernetes.io/zone]
```

如需采集annotations时，则在启动参数中以相同方法添加参数：

```
--metric-annotations-allowlist=pods=[*],nodes=[node,failure-domain.beta.kubernetes.io/zone,topology.kubernetes.io/zone]
```


须知

编辑启动命令时，请勿修改其他原有的启动参数，否则可能导致组件异常。

步骤4 kube-state-metrics将开始采集Pod和node的labels/annotations指标，查询kube_pod_labels/kube_pod_annotations是否在普罗的采集任务中。

```
kubectl get servicemonitor kube-state-metrics -nmonitoring -oyaml | grep kube_pod_labels
```

----结束

更多kube-state-metrics的启动参数请参见**kube-state-metrics/cli-arguments**。

版本记录

表 14-32 云原生监控插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
3.11.0	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30集群	**2.37.8**
3.10.1	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	NodeExporter组件升级至1.8.0版本	**2.37.8**
3.10.0	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	支持v1.29集群	**2.37.8**

插件版本	支持的集群版本	更新特性	社区版本
3.9.5	v1.21 v1.23 v1.25 v1.27 v1.28	• 新增采集自定义指标的开关，默认开启 • 移除对1.17和1.19版本集群的支持 • Grafana从云原生监控插件中移除，拆分为独立的Grafana插件 • 默认只采集免费指标和服务发现自定义指标 • 升级开源组件版本	2.37.8
3.8.2	v1.17 v1.19 v1.21 v1.23 v1.25 v1.27	修复部分问题	2.35.0
3.7.3	v1.17 v1.19 v1.21 v1.23 v1.25	-	2.35.0
3.7.2	v1.17 v1.19 v1.21 v1.23 v1.25	支持采集Virtual-Kubelet Pod指标	2.35.0
3.7.1	v1.17 v1.19 v1.21 v1.23 v1.25	支持PrometheusAgent模式	2.35.0
3.6.6	v1.17 v1.19 v1.21 v1.23 v1.25	• Grafana版本升级至7.5.17 • 支持containerd节点	2.35.0

插件版本	支持的集群版本	更新特性	社区版本
3.5.1	v1.17 v1.19 v1.21 v1.23	-	2.35.0
3.5.0	v1.17 v1.19 v1.21 v1.23	更新至社区2.35.0版本	2.35.0

14.3.2 云原生日志采集插件

插件简介

云原生日志采集插件（log-agent）是基于开源fluent-bit和opentelemetry构建的云原生日志、K8s事件采集插件。log-agent支持基于CRD的日志采集策略，可以根据您配置的策略规则，对集群中的容器标准输出日志、容器文件日志、节点日志及K8s事件日志进行采集与转发。同时支持上报K8s事件到AOM，用于配置事件告警，默认上报所有异常事件和部分正常事件。

说明

自1.3.2版本起，云原生日志采集插件默认会将上报所有Warning级别事件以及部分Normal级别事件到应用运维管理（AOM），上报的事件可用于配置告警。当集群版本为1.19.16、1.21.11、1.23.9或1.25.4及以上时，安装云原生日志采集插件后，事件上报AOM将不再由控制面组件上报，改为由云原生日志采集插件上报，卸载插件后将不再上报事件到AOM。

约束与限制

仅支持1.17及以上版本集群。

插件性能规格

性能项	说明	备注
单条日志大小	单条日志不得超过512k，多行日志采集则每行日志单独计算长度。	不涉及
最大采集文件数	单个节点所有日志采集规则监听的文件数不超过4095个文件。	不涉及
日志采集速率	• 插件低于1.5.0版本，每个集群限制单行日志采集速率不超过10000条/秒，多行日志不超过2000条/秒。 • 插件为1.5.0及以上版本，单节点限制日志采集速率不超过20000条/s、10MB/s。	超过限制尽可能提供服务，不保证服务质量。

性能项	说明	备注
配置更新	配置更新生效的延时约1-3分钟。	不涉及

权限说明

云原生日志采集插件中的fluent-bit组件会根据用户的采集配置，读取各节点上容器标准输出、容器内文件日志以及节点日志并采集。

fluent-bit组件运行需要以下权限：

- CAP_DAC_OVERRIDE：忽略文件的 DAC 访问限制。
- CAP_FOWNER：忽略文件属主 ID 必须和进程用户 ID 相匹配的限制。
- DAC_READ_SEARCH：忽略文件读及目录搜索的 DAC 访问限制。
- SYS_PTRACE：允许跟踪任何进程。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**云原生日志采集插件**，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据节点日志量选择“小规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”适用于单节点日志小于5000条/s、5MB/s的集群；“大规格”适用于单节点日志小于10000条/s、10MB/s的集群。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 （1.6.2及以上版本的插件支持）配置插件参数，相关参数说明请参见**Tail**。

参数名称	参数解释	默认取值	
		大规格	其他规格
初始缓冲区大小（Buffer_Chunk_Size）	设置初始缓冲区大小以读取文件。	256k	128k
缓冲区最大限制值（Buffer_Max_Size）	设置每个受监视文件的缓冲区大小限制。当需要增加缓冲区时，此值用于限制内存缓冲区可以增加多少。如果超过此限制，则将从监控文件列表中删除该文件。	1024k	512k

参数名称	参数解释	默认取值	
		大规格	其他规格
内存缓冲区限制（Mem_Buf_Limit）	数据追加到引擎时的内存限制。如果达到此限制，插件读取日志文件数据将暂停；刷新数据后，将再次恢复。	300mb	40mb

说明

参数单位支持k、kb、m、mb、g、gb（不区分大小写），如果不填写单位则表示以字节为单位。

步骤4 设置插件实例的部署策略。

说明

调度策略对于DaemonSet类型的插件实例不会生效。

表 14-33 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。

步骤5 完成以上配置后，单击“安装”。

----结束

组件说明

表 14-34 log-agent 组件

容器组件	说明	资源类型
fluent-bit	轻量级的日志收集器和转发器，部署在每个节点上采集日志。	DaemonSet
cop-logs	负责生成采集文件的软链接，和fluent-bit运行在同一Pod。	DaemonSet
log-operator	负责生成内部的配置文件。	Deployment

容器组件	说明	资源类型
otel-collector	负责收集来自不同应用程序和服务的日志数据，集中后上报至LTS。	Deployment

插件使用说明

该插件支持采集容器标准输出日志、容器文件日志、节点日志及K8s事件日志。您可以选择使用云日志服务（LTS）或应用运维管理服务（AOM）存储日志，但二者支持的日志类型存在差异，详情请参见**表14-35**。

表 14-35 日志存储位置说明

日志存储位置	支持的日志类型	使用说明
LTS	• 容器标准输出日志 • 容器文件日志 • 节点日志 • Kubernetes事件	请前往日志中心创建策略，具体配置方法请参见**通过云原生日志采集插件采集容器日志**。
AOM	Kubernetes事件	当集群版本为1.19.16、1.21.11、1.23.9或1.25.4及以上时，默认上报所有异常事件和部分正常事件，具体配置方法请参见**Kubernetes事件上报应用运维管理（AOM）**。

版本记录

表 14-36 云原生日志采集插件版本记录

插件版本	支持的集群版本	更新特性
1.6.1	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 支持自动创建LTS日志流 • 支持配置Buffer_Chunk_Size、Buffer_Max_Size、Mem_Buf_Limit参数 • 上报到AOM的事件中添加pod_ip字段

插件版本	支持的集群版本	更新特性
1.6.0	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 支持v1.30集群 • 安全加固：将插件使用的查询secret的权限限制在monitoring命名空间
1.5.2	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	新增创建容器日志默认日志流时索引功能
1.4.5	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题
1.4.2	v1.21 v1.23 v1.25 v1.27 v1.28	• 支持v1.28集群 • 支持本地集群日志采集 • 支持GPU事件上报AOM字段特殊处理
1.3.2	v1.17 v1.19 v1.21 v1.23 v1.25	支持Kubernetes事件上报至AOM
1.3.0	v1.17 v1.19 v1.21 v1.23 v1.25	支持v1.25集群
1.2.3	v1.17 v1.19 v1.21 v1.23	-

插件版本	支持的集群版本	更新特性
1.2.2	v1.17 v1.19 v1.21 v1.23	log-agent是基于开源fluent-bit和opentelemetry构建的云原生日志采集插件。log-agent支持基于CRD的日志采集策略，可以根据您配置的策略规则，对集群中的容器标准输出日志、容器文件日志、节点日志及K8s事件日志进行采集与转发。

14.3.3 CCE 节点故障检测

插件简介

CCE节点故障检测插件（node-problem-detector，简称NPD）是一款监控集群节点异常事件的插件，以及对接第三方监控平台功能的组件。它是一个在每个节点上运行的守护程序，可从不同的守护进程中搜集节点问题并将其报告给apiserver。node-problem-detector可以作为DaemonSet运行， 也可以独立运行。

有关社区开源项目node-problem-detector的详细信息，请参见**node-problem-detector**。

约束与限制

- 使用NPD插件时，不可对节点磁盘进行格式化或分区。
- 节点上每个NPD进程标准占用30mCPU，100MB内存。
- 当NPD插件为1.18.45及以上版本时，不再支持宿主机的操作系统为EulerOS 2.5以下版本。

权限说明

NPD插件为监控内核日志，需要读取宿主机/dev/kmsg设备，为此需要开启容器特权，详见**privileged**。

同时CCE根据最小化权限原则进行了风险消减，NPD运行限制只拥有以下特权：

- cap_dac_read_search，为访问/run/log/journal
- cap_sys_admin，为访问/dev/kmsg

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE节点故障检测**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件支持的“参数配置”。

单故障最大节点隔离数：节点批量发生相同故障时，为避免雪崩效应，最多允许被隔离的节点数量。支持按照百分比或个数配置。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-37 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

----结束

组件说明

表 14-38 npd 组件

容器组件	说明	资源类型
node-problem-controller	根据故障探测结果提供基础故障隔离能力。	Deployment
node-problem-detector	提供节点故障探测能力。	DaemonSet

NPD 检查项

说明

当前检查项仅1.16.0及以上版本支持。

NPD的检查项主要分为事件类检查项和状态类检查项。

- 事件类检查项

  对于事件类检查项，当问题发生时，NPD会向APIServer上报一条事件，事件类型分为Normal（正常事件）和Warning（异常事件）

  表 14-39 事件类检查项

故障检查项	功能	说明
OOMKilling	监听内核日志，检查OOM事件发生并上报 典型场景：容器内进程使用的内存超过了Limt，触发OOM并终止该进程	Warning类事件 监听对象：/dev/kmsg 匹配规则："Killed process \\d+ (.+) total-vm:\\d+kB, anon-rss:\\d+kB, file-rss:\\d+kB.*"
TaskHung	监听内核日志，检查taskHung事件发生并上报 典型场景：磁盘卡IO导致进程卡住	Warning类事件 监听对象：/dev/kmsg 匹配规则："task \\S+:\\w+ blocked for more than \\w+ seconds\\."

故障检查项	功能	说明
ReadonlyFilesystem	监听内核日志，检查系统内核是否有Remount root filesystem read-only错误 典型场景：用户从ECS侧误操作卸载节点数据盘，且应用程序对该数据盘的对应挂载点仍有持续写操作，触发内核产生IO错误将磁盘重挂载为只读磁盘。 **说明** 节点容器存储Rootfs为Device Mapper类型时，数据盘卸载会导致thinpool异常，影响NPD运行，NPD将无法检测节点故障。	Warning类事件 监听对象：/dev/kmsg 匹配规则："Remounting filesystem read-only"

- 状态类检查项

  对于状态类检查项，当问题发生时，NPD会向APIServer上报一条事件，并同步修改节点状态，可配合**Node-problem-controller故障隔离**对节点进行隔离。

  下列检查项中若未明确指出检查周期，则默认周期为30秒。

  表 14-40 系统组件检查

故障检查项	功能	说明
容器网络组件异常 CNIProblem	检查CNI组件（容器网络组件）运行状态	无
容器运行时组件异常 CRIProblem	检查节点CRI组件（容器运行时组件）Docker和Containerd的运行状态	检查对象：Docker或Containerd
Kubelet频繁重启 FrequentKubeletRestart	通过定期回溯系统日志，检查关键组件Kubelet是否频繁重启	• 默认阈值：10分钟内重启10次 即在10分钟内组件重启10次表示频繁重启，将会产生故障告警。 • 监听对象：/run/log/journal目录下的日志 **说明** Ubuntu和HCE2.0操作系统由于日志格式不兼容，暂不支持上述检查项。
Docker频繁重启 FrequentDockerRestart	通过定期回溯系统日志，检查容器运行时Docker是否频繁重启	
Containerd频繁重启 FrequentContainerdRestart	通过定期回溯系统日志，检查容器运行时Containerd是否频繁重启	
Kubelet服务异常 KubeletProblem	检查关键组件Kubelet的运行状态	无
KubeProxy异常 KubeProxyProblem	检查关键组件KubeProxy的运行状态	无

表 14-41 系统指标

故障检查项	功能	说明
连接跟踪表耗尽 ConntrackFullProblem	检查连接跟踪表是否耗尽	• 默认阈值:90% • 使用量:nf_conntrack_count • 最大值:nf_conntrack_max
磁盘资源不足 DiskProblem	检查节点系统盘、CCE数据盘（包含CRI逻辑盘与Kubelet逻辑盘）的磁盘使用情况	• 默认阈值：90% • 数据来源: `df -h` 当前暂不支持额外的数据盘
文件句柄数不足 FDProblem	检查系统关键资源FD文件句柄数是否耗尽	• 默认阈值：90% • 使用量：/proc/sys/fs/file-nr中第1个值 • 最大值：/proc/sys/fs/file-nr中第3个值
节点内存资源不足 MemoryProblem	检查系统关键资源Memory内存资源是否耗尽	• 默认阈值：80% • 使用量：/proc/meminfo中MemTotal-MemAvailable • 最大值：/proc/meminfo中MemTotal
进程资源不足 PIDProblem	检查系统关键资源PID进程资源是否耗尽	• 默认阈值：90% • 使用量：/proc/loadavg中nr_threads • 最大值：/proc/sys/kernel/pid_max和/proc/sys/kernel/threads-max两者的较小值。

表 14-42 存储检查

故障检查项	功能	说明
磁盘只读 DiskReadonly	通过定期对节点系统盘、CCE数据盘（包含CRI逻辑盘与Kubelet逻辑盘）进行测试性写操作，检查关键磁盘的可用性	检测路径： ● /mnt/paas/kubernetes/kubelet/ ● /var/lib/docker/ ● /var/lib/containerd/ ● /var/paas/sys/log/cceaddon-npd/ 检测路径下会产生临时文件npd-disk-write-ping 当前暂不支持额外的数据盘
节点emptydir存储池异常 EmptyDirVolumeGroupStatusError	检查节点上临时卷存储池是否正常 故障影响：依赖存储池的Pod无法正常写对应临时卷。临时卷由于IO错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为临时卷存储池，用户误操作删除了部分数据盘导致存储池异常。	● 检测周期：30秒 ● 数据来源：`vgs -o vg_name, vg_attr` ● 检测原理：检查VG（存储池）是否存在p状态，该状态表征部分PV（数据盘）丢失。 ● 节点持久卷存储池异常调度联动：调度器可自动识别此异常状态并避免依赖存储池的Pod调度到该节点上。 ● 例外场景：NPD无法检测所有PV（数据盘）丢失，导致VG（存储池）丢失的场景；此时依赖kubelet自动隔离该节点，其检测到VG（存储池）丢失并更新nodestatus.allocatable中对应资源为0，避免依赖存储池的Pod调度到该节点上。无法检测单个PV损坏；此时依赖ReadonlyFilesystem检测异常。
节点持久卷存储池异常 LocalPvVolumeGroupStatusError	检查节点上持久卷存储池是否正常 故障影响：依赖存储池的Pod无法正常写对应持久卷。持久卷由于IO错误被内核重挂载成只读文件系统。 典型场景：用户在创建节点时配置两个数据盘作为持久卷存储池，用户误操作删除了部分数据盘。	

故障检查项	功能	说明
挂载点异常 MountPointProblem	检查节点上的挂载点是否异常 异常定义：该挂载点不可访问（cd） 典型场景：节点挂载了nfs（网络文件系统，常见有obsfs、s3fs等），当由于网络或对端nfs服务器异常等原因导致连接异常时，所有访问该挂载点的进程均卡死。例如集群升级场景kubelet重启时扫描所有挂载点，当扫描到此异常挂载点会卡死，导致升级失败。	等效检查命令： `for dir in `df -h | grep -v "Mounted on" | awk "{print \\$NF}"`;do cd $dir; done && echo "ok"`
磁盘卡IO DiskHung	检查节点上所有磁盘是否存在卡IO，即IO读写无响应 卡IO定义：系统对磁盘的IO请求下发后未有响应，部分进程卡在D状态 典型场景：操作系统硬盘驱动异常或底层网络严重故障导致磁盘无法响应	• 检查对象：所有数据盘 • 数据来源：/proc/diskstat 等效查询命令：`iostat -xmt 1` • 阈值（需同时满足）： – 平均利用率（ioutil）>= 0.99 – 平均IO队列长度（avgqu-sz）>=1 – 平均IO传输量 <= 1 平均IO传输量 = 每秒完成写次数（iops，单位为w/s）+每秒写数据量（ioth，单位为wMB/s） **说明** 部分操作系统卡IO时无数据变化，此时计算CPU IO时间占用率，iowait > 0.8。

故障检查项	功能	说明
磁盘慢IO DiskSlow	检查节点上所有磁盘是否存在慢IO，即IO读写有响应但响应缓慢 典型场景：云硬盘由于网络波动导致慢IO。	• 检查对象：所有数据盘 • 数据来源：/proc/diskstat 等效查询命令 `iostat -xmt 1` • 默认阈值：平均IO时延，await >= 5000ms **说明** 卡IO场景下该检查项失效，原因为IO请求未有响应，await数据不会刷新。

表 14-43 其他检查

故障检查项	功能	说明
NTP异常 NTPProblem	检查节点时钟同步服务ntpd或chronyd是否正常运行，系统时间是否漂移	默认时钟偏移阈值：8000ms
进程D异常 ProcessD	检查节点是否存在D进程	默认阈值：连续3次存在10个异常进程 数据来源： • /proc/{PID}/stat • 等效命令：ps aux 例外场景：ProcessD忽略BMS节点下的SDI卡驱动依赖的常驻D进程heartbeat、update
进程Z异常 ProcessZ	检查节点是否存在Z进程	
ResolvConf配置文件异常 ResolvConfFileProblem	检查ResolvConf配置文件是否丢失 检查ResolvConf配置文件是否异常 异常定义：不包含任何上游域名解析服务器（nameserver）。	检查对象：/etc/resolv.conf
存在计划事件 ScheduledEvent	检查节点是否存在热迁移计划事件。热迁移计划事件通常由硬件故障触发，是IaaS层的一种自动故障修复手段。 典型场景：底层宿主机异常，例如风扇损坏、磁盘坏道等，导致其上虚机触发热迁移。	数据来源： • http://169.254.169.254/meta-data/latest/events/scheduled 该检查项为Alpha特性，默认不开启。

故障检查项	功能	说明
竞价节点中断回收中 SpotPriceNodeReclaimNotification	检查竞价实例节点是否被抢占而处于中断回收状态	默认检查周期：120秒 默认故障应对策略：驱逐节点负载

另外kubelet组件内置如下检查项，但是存在不足，您可通过集群升级或安装NPD进行补足。

表 14-44 Kubelet 内置检查项

故障检查项	功能	说明
PID资源不足 PIDPressure	检查PID是否充足	• 周期：10秒 • 阈值：90% • 缺点：社区1.23.1及以前版本，该检查项在pid使用量大于65535时失效，详见**issue 107107**。社区1.24及以前版本，该检查项未考虑thread-max。
内存资源不足 MemoryPressure	检查容器可分配空间（allocable）内存是否充足	• 周期：10秒 • 阈值：最大值-100MiB • 最大值（Allocable）：节点总内存-节点预留内存 • 缺点：该检测项没有从节点整体内存维度检查内存耗尽情况，只关注了容器部分(Allocable)。
磁盘资源不足 DiskPressure	检查kubelet盘和docker盘的磁盘使用量及inodes使用量	• 周期：10秒 • 阈值：90%

Node-problem-controller 故障隔离

说明

故障隔离仅1.16.0及以上版本的插件支持。

默认情况下，若多个节点发生故障，NPC至多为10%的节点添加污点，可通过参数npc.maxTaintedNode提高数量限制。

开源NPD插件提供了故障探测能力，但未提供基础故障隔离能力。对此，CCE在开源NPD的基础上，增强了Node-problem-controller（节点故障控制器组件，简称

NPC），该组件参照Kubernetes节点控制器实现，针对NPD探测上报的故障，自动为节点添加污点以进行基本的节点故障隔离。

表 14-45 参数说明

参数	说明	默认值
npc.enable	是否启用npc 1.18.0及以上版本不再支持该参数	true
npc.maxTaintedNode	单个故障在多个节点间发生时，限制多少节点允许被npc添加污点，避免雪崩效应 支持int格式和百分比格式	10% 值域： • int格式，数值范围为1到无穷大 • 百分比格式，数值范围为1%到100%，与集群节点数量乘积计算后最小值为1。
npc.nodeAffinity	Controller的节点亲和性配置	N/A

查看 NPD 事件

NPD上报的事件可以在节点管理页面查询。

步骤1 登录CCE控制台。

步骤2 单击集群名称进入集群，在左侧选择“节点管理”。

步骤3 在节点所在行，单击“事件”，可查看节点相关事件。

图 14-6 查看节点事件

----结束

配置 NPD 指标告警

针对NPD状态类检查项，您可以通过配置告警规则，在出现异常状态时及时通过短信、邮箱等方式通知到您。关于创建自定义告警规则的步骤，请参见通过告警中心一键配置告警。

说明

如果您需要使用NPD检查项配置告警规则，集群中需要安装“云原生监控插件”，且该插件已对接AOM实例。

Prometheus 指标采集

NPD 守护进程POD通过端口19901暴露Prometheus metrics指标，NPD Pod默认被注释metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"prometheus","path":"/metrics","port":"19901","names":""}]'。您可以自建Prometheus采集器识别并通过http://{{NpdPodIP}}:{{NpdPodPort}}/metrics路径获取NPD指标。

说明

NPD插件为1.16.5版本以下时，Prometheus指标的暴露端口为20257。

目前指标信息包含异常状态计数problem_counter与异常状态problem_gauge，如下所示

```
# HELP problem_counter Number of times a specific type of problem have occurred.
# TYPE problem_counter counter
problem_counter{reason="DockerHung"} 0
problem_counter{reason="DockerStart"} 0
problem_counter{reason="EmptyDirVolumeGroupStatusError"} 0
...
# HELP problem_gauge Whether a specific type of problem is affecting the node or not.
# TYPE problem_gauge gauge
problem_gauge{reason="CNIIsDown",type="CNIProblem"} 0
problem_gauge{reason="CNIIsUp",type="CNIProblem"} 0
problem_gauge{reason="CRIIsDown",type="CRIProblem"} 0
problem_gauge{reason="CRIIsUp",type="CRIProblem"} 0
..
```

版本记录

表 14-46 CCE 节点故障检测插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.19.11	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	修复部分问题	0.8.10
1.19.8	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 兼容单系统盘 • 支持插件实例AZ反亲和配置 • 支持在竞价实例被释放前给节点加污点，驱逐节点上的pod • 插件挂载节点时区 • 适配CCE v1.30集群	0.8.10

插件版本	支持的集群版本	更新特性	社区版本
1.19.1	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题	**0.8.10**
1.19.0	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	**0.8.10**
1.18.48	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	**0.8.10**
1.18.46	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28版本	**0.8.10**
1.18.22	v1.19 v1.21 v1.23 v1.25 v1.27	-	**0.8.10**
1.18.14	v1.19 v1.21 v1.23 v1.25	• 支持插件实例AZ反亲和配置 • 支持在竞价实例被释放前给节点加污点，驱逐节点上的pod • 插件挂载节点时区	**0.8.10**

插件版本	支持的集群版本	更新特性	社区版本
1.18.10	v1.19 v1.21 v1.23 v1.25	• 配置界面优化 • 优化DiskSlow检查项，支持阈值配置 • 优化NTPProblem检查项，支持阈值配置 • 支持插件实例AZ反亲和配置 • 支持竞价实例中断检测，中断前驱逐节点上的pod	0.8.10
1.17.4	v1.17 v1.19 v1.21 v1.23 v1.25	优化DiskHung检查项	0.8.10
1.17.3	v1.17 v1.19 v1.21 v1.23 v1.25	• NPC最大可打污点节点数支持百分比配置 • 新增进程Z状态检查项ProcessZ • 优化NTPProblem检查项，支持检测时间偏差 • 修复BMS节点场景存在常驻D状态进程，干扰ProcessD检查项	0.8.10
1.17.2	v1.17 v1.19 v1.21 v1.23 v1.25	• 新增磁盘卡IO检查项DiskHung • 新增磁盘慢IO检查项DiskSlow • 新增进程D状态检查项ProcessD • 新增挂载点健康检查MountPointProblem • 避免与Service端口范围冲突，默认健康检查监听端口修改为19900，默认Prometheus指标暴露端口修改为19901。 • 新增支持1.25集群版本	0.8.10

插件版本	支持的集群版本	更新特性	社区版本
1.16.4	v1.17 v1.19 v1.21 v1.23	• 新增beta检查项ScheduledEvent，支持通过metadata接口检测宿主机异常导致虚拟机进行冷热迁移事件。该检查项默认不开启。	0.8.10
1.16.3	v1.17 v1.19 v1.21 v1.23	新增ResolvConf配置文件检查。	0.8.10
1.16.1	v1.17 v1.19 v1.21 v1.23	• 新增node-problem-controller。支持基本故障隔离能力。 • 新增PID、FD、磁盘、内存、临时卷存储池、持久卷存储池检查项。	0.8.10
1.15.0	v1.17 v1.19 v1.21 v1.23	• 检测项全面加固，避免误报。 • 支持内核巡检。支持OOMKilling事件，TaskHung事件上报。	0.8.10
1.14.11	v1.17 v1.19 v1.21	适配CCE v1.21集群	0.7.1
1.14.5	v1.17 v1.19	修复监控指标无法被获取的问题	0.7.1
1.14.4	v1.17 v1.19	• 适配containerd运行时节点	0.7.1
1.14.2	v1.17 v1.19	• 适配CCE v1.19集群 • 新增支持Ubuntu操作系统和安全容器场景	0.7.1
1.13.8	v1.15.11 v1.17	• 修复容器隧道网络下CNI健康检查问题 • 调整资源配额	0.7.1
1.13.6	v1.15.11 v1.17	修复僵尸进程未被回收的问题	0.7.1
1.13.5	v1.15.11 v1.17	增加污点容忍配置	0.7.1

插件版本	支持的集群版本	更新特性	社区版本
1.13.2	v1.15.11 v1.17	增加资源限制，增强cni插件的检测能力	0.7.1

14.3.4 CCE 容器网络扩展指标

插件简介

CCE容器网络扩展指标插件（CCE Network Metrics Exporter）是一款CCE Turbo集群容器网络可观测性增强插件，能提供CCE Turbo集群容器网络相关的各项监控数据，帮助您观测各种容器网络流量以及快速发现和定位容器网络问题。插件实例仅支持部署在X86/ARM架构的HCE 2.0节点或X86架构的EulerOS的节点上。

当前支持Pod粒度和flow粒度的IP、UDP和TCP协议监控，且支持通过PodSelector来对监控后端作选择，支持多监控任务、可选监控指标，且支持用户获取Pod的label标签信息。监控信息已适配Prometheus格式，可以通过调用Prometheus接口查看监控数据。

使用约束

- 仅支持在v1.19及以上版本的CCE Turbo集群中安装此插件，插件实例仅支持部署在X86/ARM架构的HCE 2.0节点（1.4.5及以上版本的插件支持）或X86架构的EulerOS的节点上。
- 支持节点使用Containerd或Docker容器引擎。其中Containerd节点可实时跟踪Pod更新，Docker节点以轮询方式查询，非实时更新。
- 仅支持统计CCE Turbo集群**安全容器**（容器运行时为kata）以及普通容器（容器运行时为runc）的流量。
- 安装插件后默认将不进行流量监控，用户需通过创建MonitorPolicy来配置监控任务进行流量监控，且MonitorPolicy资源必须创建在kube-system命名空间下。
- 不支持HostNetwork类型Pod监控。
- 安装时请确保节点有足够的资源安装本插件。
- 监控标签及用户标签的来源必须为新创建Pod时就具有的。
- 用户指定的标签最多5个标签（1.3.4版本后最多支持10个标签），且不可以指定系统使用的label用作用户指定label。系统使用lable包括：pod、task、ipfamily、srcip、dstip、srcport、dstport、protocol。
- HCE 2.0 x86节点不支持tcp drop监控项。

安装插件

步骤1 登录CCE控制台，单击CCE Turbo集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE容器网络扩展指标**插件，单击“安装”。

步骤2 在安装插件页面，查看插件配置。

当前该插件无可配置参数。

步骤3 单击“安装”。

待插件安装完成后，选择对应的集群，然后单击左侧导航栏的“插件中心”，可在“已安装插件”页签中查看相应的插件。

----结束

组件说明

表 14-47 dolphin 组件

容器组件	说明	资源类型
dolphin	dolphin负责CCE Turbo集群的容器网络流量监控。	Daemon Set

支持的监控项

当前用户可通过创建MonitorPolicy的方式下发监控任务，当前用户可以通过API或登录工作节点kubectl apply的方式创建MonitorPolicy。一个MonitorPolicy代表着一个监控任务，提供selector、podLabel等可选参数。当前支持的监控指标如下：

表 14-48 当前支持的监控指标

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
IPv4发送公网报文数	dolphin_ip4_send_pkt_internet	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86
IPv4发送公网字节数	dolphin_ip4_send_byte_internet	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86
IPv4接收报文数	dolphin_ip4_rcv_pkt	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86
IPv4接收字节数	dolphin_ip4_rcv_byte	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86
IPv4发送报文数	dolphin_ip4_send_pkt	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
IPv4发送字节数	dolphin_ip4_send_byte	pod	runc/kata	v1.19及以上	1.1.2	EulerOS 2.9 x86 EulerOS 2.10 x86
最近一次的健康检查健康状态	dolphin_health_check_status	pod	runc/kata	v1.19及以上	1.2.2	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
健康检查成功累计次数	dolphin_health_check_successful_counter	pod	runc/kata	v1.19及以上	1.2.2	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
健康检查失败累计次数	dolphin_health_check_failed_counter	pod	runc/kata	v1.19及以上	1.2.2	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
IP接收报文数	dolphin_ip_receive_pkt	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP接收字节数	dolphin_ip_receive_byte	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP发送报文数	dolphin_ip_send_pkt	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP发送字节数	dolphin_ip_send_byte	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
TCP接收报文数	dolphin_tcp_receive_pkt	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP接收字节数	dolphin_tcp_receive_byte	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP发送报文数	dolphin_tcp_send_pkt	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP发送字节数	dolphin_tcp_send_byte	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
TCP重传报文数	dolphin_tcp_retrans	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP新建连接数	dolphin_tcp_connection	pod	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP接收报文数	dolphin_flow_ip_receive_pkt	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP接收字节数	dolphin_flow_ip_receive_byte	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
IP发送报文数	dolphin_flow_ip_send_pkt	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
IP发送字节数	dolphin_flow_ip_send_byte	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP接收报文数	dolphin_flow_tcp_receive_pkt	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP接收字节数	dolphin_flow_tcp_receive_byte	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
TCP发送报文数	dolphin_flow_tcp_send_pkt	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP发送字节数	dolphin_flow_tcp_send_byte	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP重传报文数	dolphin_flow_tcp_retrans	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）
TCP smoothed round trip	dolphin_flow_tcp_srtt	flow	runc	v1.23及以上	1.3.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM（1.4.5及以上版本的插件支持）

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
TCP丢包数	dolphin_tcp_drop	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
TCP丢包数	dolphin_flow_tcp_drop	flow	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
TCP建链失败数	dolphin_tcp_connection_failure	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP接收报文数	dolphin_udp_receive_pkt	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP接收字节数	dolphin_udp_receive_byte	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP发送报文数	dolphin_udp_send_pkt	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM

监控指标	监控项名称	监控粒度	支持的运行时	支持的集群版本	支持的插件版本	支持的操作系统
UDP发送字节数	dolphin_udp_send_byte	pod	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP接收报文数	dolphin_flow_udp_receive_pkt	flow	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP接收字节数	dolphin_flow_udp_receive_byte	flow	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP发送报文数	dolphin_flow_udp_send_pkt	flow	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM
UDP发送字节数	dolphin_flow_udp_send_byte	flow	runc	v1.23及以上	1.4.5	EulerOS 2.9 x86 EulerOS 2.10 x86 HCE 2.0 x86/ARM

下发监控任务

MonitorPolicy创建的模板如下：

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task            #监控任务名
  namespace: kube-system        #必填，namespace必须为kube-system
spec:
  selector:                     #选填，配置dolphin插件监控的后端，形如labelSelector格式，默认将监控本节点所有容器
    matchLabels:
```

```
    app: nginx
  matchExpressions:
    - key: app
      operator: In
      values:
        - nginx
 podLabel: [app]             #选填，用户标签
 ip4Tx:                    #选填，ipv4发送报文数和发送字节数这两个指标的开关，默认不开
  enable: true
 ip4Rx:                    #选填，ipv4接收报文数和接收字节数这两个指标的开关，默认不开
  enable: true
 ip4TxInternet:              #选填，ipv4发送公网报文数和发送公网字节数这两个指标的开关，默认不开
  enable: true
 healthCheck:                #选填，本地节点 Pod 健康检查任务中最近一次健康检查是否健康、健康检查总健康
&不健康次数这三个指标开关，默认不开
  enable: true               # true false
  failureThreshold: 3         #选填，健康检查不健康判定失败次数，默认1次健康检查失败即判定不健康
  periodSeconds: 5            #选填，健康检查任务检查间隔时间，单位秒，默认60
  command: ""                 #选填，健康检查任务检查命令，支持：ping、arping、curl，默认 ping
  ipFamilies: [""]           #选填，健康检查IP地址族，支持：ipv4，默认ipv4
  port: 80                  #选填，使用curl时必选，端口号
  path: ""                  #选填，使用curl时必选，http api 路径
 monitor:
  ip:
    ipReceive:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    ipSend:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
  tcp:
    tcpReceive:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    tcpSend:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    tcpRetrans:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    tcpRtt:
      aggregateType: flow      #选填，支持填写"flow"，表示流粒度监控，单位：微秒
    tcpNewConnection:
      aggregateType: pod       #选填，支持填写"pod"，表示pod粒度监控
    tcpDrop:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    tcpConnectionFailure:
      aggregateType: pod       #选填，支持填写"pod"，表示pod粒度监控
  udp:
    udpReceive:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
    udpSend:
      aggregateType: flow      #选填，支持填写"pod"或"flow"，分别表示pod粒度监控或流粒度监控
```

用户标签PodLabel：可输入多个Pod的label标签，以逗号分隔，如[app, version]。

标签需符合以下规则，对应正则表达式为(^[a-zA-Z_]$)|(^([a-zA-Z][a-zA-Z0-9_]|_[a-zA-Z0-9])([a-zA-Z0-9_]){0,254}$)：

- 支持输入最多5个标签（1.3.4版本后最多支持10个标签），单个标签长度最长256个字符。
- 不能以数字和双下划线__开头。
- 单个标签格式需符合A-Za-z_0-9。

用户可以按照上述格式对监控任务进行创建、修改、及删除，当前仅支持最多10个监控任务的创建，且多个监控任务匹配到同一个监控后端时，每一个监控后端将会产生监控任务数量的监控指标。

说明

- 修改或删除监控任务，都将导致丢失原有监控任务所采集的监控数据，请谨慎操作。
- 用户卸载插件后，用户之前配置的监控任务MonitorPolicy将随着插件卸载一并销毁。

场景化示例如下：

1. 以下示例将监控节点上满足app=nginx的labelselector的所有Pod，生成三个健康检查指标，默认使用ping方式检测本地Pod，若监控的容器携带test及app这两个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLabel: [test, app]
  healthCheck:
    enable: true
    failureThreshold: 3
    periodSeconds: 5
```

2. 以下示例将监控节点上满足app=nginx的labelselector的所有Pod，生成三个健康检查指标，自定义curl方式（此处curl只考虑网络连通性，不论程序返回的http code是什么，只要网络能连通就认为Pod是健康的）。若监控的容器携带test及app这两个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLabel: [test, app]
  healthCheck:
    enable: true
    failureThreshold: 3
    periodSeconds: 5
    command: "curl"
    port: 80
    path: "healthz"
```

3. 以下示例将监控节点上满足app=nginx的labelselector的所有Pod，监控Pod粒度的IP收发报文数、IP收发字节数、TCP收发报文数、TCP收发字节数、TCP重传报文数、TCP新建连接数、TCP丢包数、TCP建链失败数、UDP收发报文数、UDP收发字节数，若监控的容器携带test及app这两个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLabel: [test, app]
  monitor:
```



```
    ip:
      ipReceive:
        aggregateType: pod
      ipSend:
        aggregateType: pod
    tcp:
      tcpReceive:
        aggregateType: pod
      tcpSend:
        aggregateType: pod
      tcpRetrans:
        aggregateType: pod
      tcpNewConnection:
        aggregateType: pod
      tcpDrop:
        aggregateType: pod
      tcpConnectionFailure:
        aggregateType: pod
    udp:
      udpReceive:
        aggregateType: pod
      udpSend:
        aggregateType: pod
```

4. 以下示例将监控节点上满足app=nginx的labelselector的所有Pod，监控流粒度的IP收发报文数、IP收发字节数、TCP收发报文数、TCP收发字节数、TCP重传报文数、tcp round trip time（单位：微秒）、TCP丢包数、UDP收发报文数、UDP收发字节数，若监控的容器携带test及app这两个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。使用流粒度监控能力，用户可以更细腻的感知容器的流量信息。基于流的监控数据量较大，会占用更多的cpu和内存，请按需使用。

   每开启一个基于流的IP监控任务（一个MonitorPolicy中开启一个和多个IP监控项）会占用内核2.6M内存；每开启一个基于流的TCP监控任务（一个MonitorPolicy中开启一个和多个TCP监控项）会占用内核14M内存。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLabel: [test, app]
  monitor:
    ip:
      ipReceive:
        aggregateType: flow
      ipSend:
        aggregateType: flow
    tcp:
      tcpReceive:
        aggregateType: flow
      tcpSend:
        aggregateType: flow
      tcpRetrans:
        aggregateType: flow
      tcpRtt:
        aggregateType: flow
      tcpDrop:
        aggregateType: flow
    udp:
      udpReceive:
        aggregateType: flow
      udpSend:
        aggregateType: flow
```


说明

基于流的监控数据量比较大时，当数据量超过一定限制时，会导致超限的流统计丢失，当前限制如下：

- 10s内内核态最多统计5w条（每监控任务）TCP流信息。
- 10s内内核态最多统计1w条（每监控任务）IP流信息。
- 两次普罗拉取间隔最多缓存6w条（所有监控任务）流统计信息。
- 普罗长时间不拉取时，只缓存1小时内的监控数据。

5. 以下示例将监控节点上所有Pod，生成IPv4发送报文数和发送字节数这两个指标，若监控的容器携带app这个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  podLabel: [app]
  ip4Tx:
    enable: true
```

6. 以下示例将监控节点上满足app=nginx的labelselector的所有Pod，生成IPv4收发报文数、IPv4收发字节数、IPv4发送公网报文数和字节数等指标，若监控的容器携带test及app这两个标签，将在监控指标上携带对应label的key-value信息，否则对应label的value为“not found”。

```
apiVersion: crd.dolphin.io/v1
kind: MonitorPolicy
metadata:
  name: example-task
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: nginx
  podLabel: [test, app]
  ip4Tx:
    enable: true
  ip4Rx:
    enable: true
  ip4TxInternet:
    enable: true
```

查看流量统计

dolphin插件的监控信息以Prometheus exporter格式输出，有以下方式获取dolphin插件的监控信息：

- 直接访问dolphin插件提供的服务端口10001，形如http://{POD_IP}:10001/metrics

  注意，如果在节点上访问dolphin服务端口，需要放通节点和Pod的安全组限制。

获取的监控信息示例如下：

- 示例1（IPv4发送公网报文数）：

```
dolphin_ip4_send_pkt_internet{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 241
```

  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的发送公网报文数为241。

- 示例2（IPv4发送公网字节数）：
  ```
  dolphin_ip4_send_byte_internet{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task" } 23618
  ```
  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的发送公网字节数为23618。
- 示例3（IPv4发送报文数）：
  ```
  dolphin_ip4_send_pkt{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 379
  ```
  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的发送报文数为379。
- 示例4（IPv4发送字节数）：
  ```
  dolphin_ip4_send_byte{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 33129
  ```
  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的发送字节数为33129。
- 示例5（IPv4接收报文数）：
  ```
  dolphin_ip4_rcv_pkt{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 464
  ```
  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的接收报文数为464。
- 示例6（IPv4接收字节数）：
  ```
  dolphin_ip4_rcv_byte{app="nginx",pod="default/nginx-66c9c65dbf-zjg24",task="kube-system/example-task "} 34654
  ```
  如上示例中，Pod所在命名空间为default，Pod名称为nginx-66c9c65dbf-zjg24，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的接收字节数为34654。
- 示例7（健康检查状态）：
  ```
  dolphin_health_check_status{app="nginx",pod="default/nginx-b74766f5f-7582p",task="kube-system/example-task"} 0
  ```
  如上示例中，Pod所在命名空间为kube-system，Pod名称为default/nginx-deployment-b74766f5f-7582p，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的网络健康状态为0（健康），不健康值为 1。
- 示例8（健康检查成功次数）：
  ```
  dolphin_health_check_successful_counter{app="nginx",pod="default/nginx-b74766f5f-7582p",task="kube-system/example-task"} 5
  ```
  如上示例中，Pod所在命名空间为kube-system，Pod名称为default/nginx-deployment-b74766f5f-7582p，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的网络健康检查成功次数为5。
- 示例9（健康检查失败次数）：
  ```
  dolphin_health_check_failed_counter{app="nginx",pod="default/nginx-b74766f5f-7582p",task="kube-system/example-task"} 0
  ```
  如上示例中，Pod所在命名空间为kube-system，Pod名称为default/nginx-deployment-b74766f5f-7582p，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod的网络健康失败次数为0。
- 示例10（流粒度监控结果）：


```
dolphin_flow_tcp_send_byte{app="nginx",dstip="192.168.0.89",dstport="80",ipfamily="ipv4",pod="kube-system/nginx-b74766f5f-7582p",srcip="192.168.1.67",srcport="12973",task="kube-system/example-task"} 1725 1700538280914
```

如上示例中，Pod所在命名空间为kube-system，Pod名称为nginx-b74766f5f-7582p，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，192.168.1.67:12973向192.168.0.89:80的发送IPv4 TCP字节数为1725，时间戳为1700538280914。

- 示例11（Pod粒度监控结果）：

```
dolphin_tcp_send_pkt{app="nginx",ipfamily="ipv4",pod="kube-system/nginx-b74766f5f-7582p",task="kube-system/example-task"} 14
dolphin_tcp_send_pkt{app="nginx",ipfamily="ipv6",pod="kube-system/nginx-b74766f5f-7582p",task="kube-system/example-task"} 0
```

如上示例中，Pod所在命名空间为kube-system，Pod名称为nginx-b74766f5f-7582p，用户指定label为app，其值对应为nginx，该监控指标由名称为example-task的监控任务创建，该Pod发送IPv4报文数为14，发送的IPv6报文数为0。

说明

若容器无用户指定的标签，返回体中标签值为not found。形如：

dolphin_ip4_send_byte_internet{test="not found", pod="default/nginx-66c9c65dbf-zjg24",task="default" } 23618

版本记录

表 14-49 CCE 容器网络扩展指标插件版本记录

插件版本	支持的集群版本	更新特性
1.4.15	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	支持CCE Turbo v1.30集群
1.4.7	v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题
1.4.5	v1.23 v1.25 v1.27 v1.28 v1.29	• 支持普通容器Pod粒度的UDP、TCP drop、TCP connect fail监控 • 支持普通容器flow粒度的UDP、TCP drop监控 • 支持HCE 2.0 x86和HCE 2.0 ARM • 支持CCE v1.29集群

插件版本	支持的集群版本	更新特性
1.3.10	v1.23 v1.25 v1.27 v1.28	修复部分问题
1.3.8	v1.23 v1.25 v1.27 v1.28	• 支持普通容器pod粒度的IP和TCP监控 • 支持普通容器flow粒度的IP和TCP监控 • 支持CCE v1.27集群 • 支持CCE v1.28集群
1.2.27	v1.19 v1.21 v1.23 v1.25	-
1.2.4	v1.19 v1.21 v1.23 v1.25	• 增加不支持EulerOS以外操作系统描述
1.2.2	v1.19 v1.21 v1.23 v1.25	• 本地Pod VPC网络健康检查
1.1.8	v1.19 v1.21 v1.23 v1.25	• 适配CCE v1.25集群
1.1.6	v1.19 v1.21 v1.23	-
1.1.5	v1.19 v1.21 v1.23	• liveness健康检查优化
1.1.2	v1.19 v1.21 v1.23	• 支持操作系统类型宽匹配

插件版本	支持的集群版本	更新特性
1.0.1	v1.19 v1.21	• 支持流量统计数据持久化和本地socket通信.

14.3.5 Kubernetes Metrics Server

从Kubernetes 1.8开始，Kubernetes通过Metrics API提供资源使用指标，例如容器CPU和内存使用率。这些度量可以由用户直接访问（例如，通过使用kubectl top命令），或者由集群中的控制器（例如，Horizontal Pod Autoscaler）使用来进行决策，具体的组件为Metrics-Server，用来替换之前的heapster，heapster从1.11开始逐渐被废弃。

Metrics Server是集群核心资源监控数据的聚合器，您可以在CCE控制台快速安装本插件。

安装本插件后，可创建HPA策略，具体请参见**创建HPA策略**。

社区官方项目及文档：**https://github.com/kubernetes-sigs/metrics-server**。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**Kubernetes Metrics Server**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据需求选择“单实例”或“高可用”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “单实例”不具备高可用能力；“高可用”具有高可用能力，但多实例部署需占用更多的计算资源。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-50 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤4 单击“安装”。

----结束

组件说明

表 14-51 metrics-server 组件

容器组件	说明	资源类型
metrics-server	集群核心资源监控数据的聚合器，用于收集和聚合集群中通过Metrics API提供的资源使用指标。	Deployment

版本记录

表 14-52 Kubernetes Metrics Server 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.3.68	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30集群	**0.6.2**
1.3.60	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群	**0.6.2**
1.3.39	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	**0.6.2**
1.3.37	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28集群	**0.6.2**
1.3.12	v1.19 v1.21 v1.23 v1.25 v1.27	-	**0.6.2**
1.3.8	v1.19 v1.21 v1.23 v1.25	插件挂载节点时区	**0.6.2**

插件版本	支持的集群版本	更新特性	社区版本
1.3.6	v1.19 v1.21 v1.23 v1.25	• 支持插件实例AZ反亲和配置 • 默认污点容忍时长修改为60s	0.6.2
1.3.3	v1.19 v1.21 v1.23 v1.25	• 适配CCE v1.25集群 • CronHPA调整Deployment实例数，新增skip场景	0.6.2
1.3.2	v1.19 v1.21 v1.23 v1.25	适配CCE v1.25集群	0.6.2
1.2.1	v1.19 v1.21 v1.23	适配CCE v1.23集群	0.4.4
1.1.10	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群	0.4.4
1.1.4	v1.15 v1.17 v1.19	资源规格配置单位统一化	0.4.4
1.1.2	v1.15 v1.17 v1.19	同步至社区v0.4.4版本	0.4.4
1.1.1	v1.13 v1.15 v1.17 v1.19	支持自定义资源规格配置，最大无效实例数改为1	0.3.7
1.1.0	v1.13 v1.15 v1.17 v1.19	适配CCE v1.19集群	0.3.7
1.0.5	v1.13 v1.15 v1.17	更新至社区v0.3.7版本	0.3.7

14.3.6 Grafana

插件简介

Grafana是一款开源的数据可视化和监控平台，可以为您提供丰富的图表和面板，用于实时监控、分析和可视化各种指标和数据源。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**Grafana**，单击“安装”。

步骤2 设置插件的“规格配置”，您可根据需求调整插件实例的CPU配额和内存配额。

步骤3 设置插件支持的“参数配置”。

表 14-53 Grafana 插件参数配置

参数	参数说明
存储卷声明类型	安装Grafana需创建存储卷用于存储本地数据，卸载插件时Grafana的存储卷不会删除。 • 选择“云硬盘”类型时，需选择“云硬盘类型”，不同局点支持的云硬盘类型可能不同，请以控制台选择项为准。创建云硬盘会收取存储费用，并占用云硬盘的配额。 • 选择“专属存储”类型时，需选择“专属实例”，创建的存储卷将创建在对应的存储池中。
容量 (GiB)	云硬盘的大小默认为5GiB。您可以在创建完成后对存储卷进行扩容，详情请参见**相关操作**。
对接AOM	将普罗数据上报至 AOM 服务。开启后，可选择对应的AOM实例。采集的基础指标免费，自定义指标将由AOM服务进行收费，详情请参见**价格详情**。对接AOM需要用户具备一定权限，目前仅**华为云/华为账号**，或者在**admin**用户组下的用户支持此操作。
公网访问	1.2.1及以上版本的插件支持开启公网访问，开启后需要选择一个负载均衡器作为Grafana服务入口。仅支持选择集群所在VPC下的负载均衡实例。如果使用独享型ELB，该实例还需要包含网络型规格。 **须知** 开启公网访问将会把Grafana服务暴露至公网，建议评估安全风险并做好访问策略的管控。

步骤4 设置插件实例的部署策略。

表 14-54 插件调度配置

参数	参数说明
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

待插件安装完成后，选择对应的集群，然后单击左侧导航栏的“插件中心”，可筛选“已安装插件”查看相应的插件。

----结束

组件说明

表 14-55 Grafana 组件

容器组件	说明	资源类型
grafana	提供Grafana的数据可视化能力。	Deployment

使用说明

如需通过公网访问Grafana图表，您需要为Grafana容器实例绑定LoadBalancer类型的服务。

步骤1 登录CCE控制台，选择一个已安装Grafana插件的集群，在左侧导航栏中选择“服务”。

步骤2 单击右上角“YAML创建”，为Grafana创建一个公网LoadBalancer类型Service。

```
apiVersion: v1
kind: Service
```



```
metadata:
  name: grafana-lb     #服务名称，可自定义
  namespace: monitoring
  labels:
    app: grafana
  annotations:
    kubernetes.io/elb.id: 038ff***     #请替换为集群所在VPC下的ELB实例ID，且ELB实例为公网访问类型
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 80     #服务端口号，可自定义
      targetPort: 3000     #Grafana的默认端口号，无需更改
  selector:
    app: grafana
  type: LoadBalancer
```

步骤3 创建完成后在浏览器访问“负载均衡公网IP地址:服务端口”，访问Grafana并选择合适的DashBoard，即可以查到相应的聚合内容。

图 14-7 Grafana 面板

----结束

版本记录

表 14-56 Grafana 插件版本记录

插件版本	支持的集群版本	更新特性
1.3.0	v1.17 v1.19 v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	支持v1.30集群

插件版本	支持的集群版本	更新特性
1.2.1	v1.17 v1.19 v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	支持关联LoadBalancer类型的Service
1.2.0	v1.17 v1.19 v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	支持v1.29集群
1.1.0	v1.17 v1.19 v1.21 v1.23 v1.25 v1.27 v1.28	提供Grafana的开源版

14.3.7 Prometheus（停止维护）

插件简介

Prometheus是一套开源的系统监控报警框架。它启发于Google的borgmon监控系统，由工作在SoundCloud的Google前员工在2012年创建，作为社区开源项目进行开发，并于2015年正式发布。2016年，Prometheus正式加入Cloud Native Computing Foundation，成为受欢迎度仅次于Kubernetes的项目。

在云容器引擎CCE中，支持以插件的方式快捷安装Prometheus。

插件官网：**https://prometheus.io/**

开源社区地址：**https://github.com/prometheus/prometheus**

约束与限制

CCE提供的Prometheus插件仅支持1.21及以下版本的集群。

插件特点

作为新一代的监控框架，Prometheus具有以下特点：

- 强大的多维度数据模型：
  a. 时间序列数据通过metric名和键值对来区分。
  b. 所有的metrics都可以设置任意的多维标签。
  c. 数据模型更随意，不需要刻意设置为以点分隔的字符串。
  d. 可以对数据模型进行聚合，切割和切片操作。
  e. 支持双精度浮点类型，标签可以设为全unicode。
- 灵活而强大的查询语句（PromQL）：在同一个查询语句，可以对多个metrics进行乘法、加法、连接、取分数位等操作。
- 易于管理：Prometheus server是一个单独的二进制文件，可直接在本地工作，不依赖于分布式存储。
- 高效：平均每个采样点仅占 3.5 bytes，且一个Prometheus server可以处理数百万的metrics。
- 使用pull模式采集时间序列数据，这样不仅有利于本机测试而且可以避免有问题的服务器推送坏的metrics。
- 可以采用push gateway的方式把时间序列数据推送至Prometheus server端。
- 可以通过服务发现或者静态配置去获取监控的targets。
- 有多种可视化图形界面。
- 易于伸缩。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**Prometheus**，单击“安装”。

步骤2 在“规格配置”步骤中，配置以下参数：

表 14-57 Prometheus 配置参数说明

参数	参数说明
插件规格	根据业务需求，选择插件的规格，包含如下选项： ● 演示规格（100容器以内）：适用于体验和功能演示环境，该规模下prometheus占用资源较少，但处理能力有限。建议在集群内容器数目不超过100时使用。 ● 小规格（2000容器以内）：建议在集群中的容器数目不超过2000时使用。 ● 中规格（5000容器以内）：建议在集群中的容器数目不超过5000时使用。 ● 大规格（超过5000容器）：建议集群中容器数目超过5000时使用此规格。
实例数	选择上方插件规格后，显示插件中的实例数，此处仅作显示。

参数	参数说明
容器	选择插件规格后，显示插件容器的CPU和内存配额，此处仅作显示。
数据保留期	自定义监控数据需要保留的天数，默认为15天。
存储	支持云硬盘作为存储，按照界面提示配置如下参数： ● 可用区：请根据业务需要进行选择。可用区是在同一区域下，电力、网络隔离的物理区域，可用区之间内网互通，不同可用区之间物理隔离。 ● 子类型：支持普通IO、高IO和超高IO三种类型。 ● 容量：请根据业务需要输入存储容量，默认10G。 **说明** 若命名空间monitoring下已存在pvc，将使用此存储作为存储源。

步骤3 单击“安装”。安装完成后，插件会在集群中部署以下实例。

- prometheus-operator：根据自定义资源（Custom Resource Definition / CRDs）来部署和管理Prometheus Server，同时监控这些自定义资源事件的变化来做相应的处理，是整个系统的控制中心。
- prometheus（Server）：Operator根据自定义资源Prometheus类型中定义的内容而部署的Prometheus Server集群，这些自定义资源可以看作是用来管理Prometheus Server集群的 StatefulSets 资源。
- prometheus-kube-state-metrics：将Prometheus的metrics数据格式转换成K8s API接口能识别的格式。
- custom-metrics-apiserver：将自定义指标聚合到原生的kubernetes apiserver。
- prometheus-node-exporter：每个节点上均有部署，收集Node级别的监控数据。
- grafana：可视化浏览普罗监控数据。

----结束

通过 Metrics API 提供资源指标

容器和节点的资源指标，如CPU、内存使用量，可通过Kubernetes的Metrics API获得。这些指标可以直接被用户访问，比如用kubectl top命令，也可以被HPA或者CustomedHPA使用，根据资源使用率使负载弹性伸缩。

插件可为Kubernetes提供Metrics API，但默认未开启，若要将其开启，需要创建以下APIService对象：

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  labels:
    app: custom-metrics-apiserver
    release: cceaddon-prometheus
  name: v1beta1.metrics.k8s.io
spec:
  group: metrics.k8s.io
  groupPriorityMinimum: 100
  insecureSkipTLSVerify: true
  service:
    name: custom-metrics-apiserver
```



```
  namespace: monitoring
  port: 443
version: v1beta1
versionPriority: 100
```

可以将该对象保存为文件，命名为metrics-apiservice.yaml，然后执行以下命令：

```
kubectl create -f metrics-apiservice.yaml
```

执行kubectl top pod -n monitoring命令，若显示如下，则表示Metrics API能正常访问：

```
# kubectl top pod -n monitoring
NAME                                        CPU(cores)   MEMORY(bytes)
......
custom-metrics-apiserver-d4f556ff9-l2j2m                 38m          44Mi
......
```

须知

卸载插件时，需要执行以下kubectl命令，同时删除APIService对象，否则残留的APIService资源将导致metrics-server插件安装失败。

```
kubectl delete APIService v1beta1.metrics.k8s.io
```

参考资源

- Prometheus概念及详细配置请参阅**Prometheus 官方文档**
- Node exporter安装请参考**node_exporter github 仓库**

版本记录

表 14-58 Prometheus 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
2.23.32	v1.17 v1.19 v1.21	-	**2.10.0**
2.23.31	v1.15	• 适配CCE v1.15集群	**2.10.0**
2.23.30	v1.17 v1.19 v1.21	• 适配CCE v1.21集群	**2.10.0**
2.21.14	v1.17 v1.19 v1.21	• 适配CCE v1.21集群	**2.10.0**
2.21.12	v1.15	• 适配CCE v1.15集群	**2.10.0**
2.21.11	v1.17 v1.19	• 适配CCE v1.19集群	**2.10.0**

插件版本	支持的集群版本	更新特性	社区版本
1.15.1	v1.15 v1.17	• Prometheus是一个监控系统和时间序列库	2.10.0

14.4 云原生异构计算插件

14.4.1 CCE AI 套件（NVIDIA GPU）

插件简介

CCE AI套件（NVIDIA GPU）插件是支持在容器中使用GPU显卡的设备管理插件，集群中使用GPU节点时必须安装本插件。

约束与限制

- 下载的驱动必须是后缀为“.run”的文件。
- 仅支持Nvidia Tesla驱动，不支持GRID驱动。
- 安装或重装插件时，需要保证驱动下载链接正确且可正常访问，插件对链接有效性不做额外校验。
- 插件仅提供驱动的下载及安装脚本执行功能，插件的状态仅代表插件本身功能正常，与驱动是否安装成功无关。
- 对于GPU驱动版本与您业务应用的兼容性（GPU驱动版本与CUDA库版本的兼容性），CCE不保证两者之间兼容性，请您自行验证。
- 对于已经安装GPU驱动的自定义操作系统镜像，CCE无法保证其提供的GPU驱动与CCE其他GPU组件兼容（例如监控组件等）。
- 如果您使用不在**GPU驱动支持列表**内的GPU驱动版本，可能引发GPU驱动与操作系统版本、ECS实例类型、Container Runtime等不兼容，继而导致驱动安装失败或者GPU插件异常。对于使用自定义GPU驱动的场景，请您自行验证。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**CCE AI套件（NVIDIA GPU）**插件，单击“安装”。

步骤2 设置插件支持的“参数配置”。

表 14-59 GPU 插件参数配置

参数	参数说明
集群默认驱动	集群下全部GPU节点将使用相同的驱动，请选择合适的GPU驱动版本，或自定义驱动链接地址，填写Nvidia驱动的下载链接。 **须知** • 如果下载链接为公网地址，如nvidia官网地址（ https://us.download.nvidia.com/tesla/470.103.01/NVIDIA-Linux-x86_64-470.103.01.run ），各GPU节点均需要绑定EIP。获取驱动链接方法请参考**获取驱动链接-公网地址**。 • 若下载链接为OBS上的链接，无需绑定EIP 。获取驱动链接方法请参考**获取驱动链接-OBS地址**。 • 请确保Nvidia驱动版本与GPU节点适配。配套关系请参见**GPU驱动支持列表**。 • 更改驱动版本后，需要重启节点才能生效。 • 对于Linux 5.x内核系统：Huawei Cloud EulerOS 2.0建议使用470及以上版本驱动；Ubuntu 22.04建议使用515及以上版本驱动。

说明

插件安装完成后，GPU 虚拟化和节点池驱动配置请前往“配置中心 > 异构资源配置”页进行设置。

步骤3 单击“安装”，安装插件的任务即可提交成功。

说明

卸载插件将会导致重新调度的GPU Pod无法正常运行，但已运行的GPU Pod不会受到影响。

----结束

验证插件

插件安装完成后，在GPU节点及调度了GPU资源的容器中执行nvidia-smi命令，验证GPU设备及驱动的可用性。

- GPU节点：

```
# 插件版本为2.0.0以下时，执行以下命令：
cd /opt/cloud/cce/nvidia/bin && ./nvidia-smi

# 插件版本为2.0.0及以上时，驱动安装路径更改，需执行以下命令：
cd /usr/local/nvidia/bin && ./nvidia-smi
```

- 容器：

```
cd /usr/local/nvidia/bin && ./nvidia-smi
```

若能正常返回GPU信息，说明设备可用，插件安装成功。


```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.118.02   Driver Version: 440.118.02   CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:21:01.0 Off |                    0 |
| N/A   31C    P0    23W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

GPU 驱动支持列表

须知

- 当前GPU驱动支持列表仅针对1.2.28及以上版本的GPU插件。
- 如果您需要安装最新版本的GPU驱动，请将您的GPU插件升级到最新版本。

表 14-60 GPU 驱动支持列表

GPU型号	支持集群类型	机型规格	操作系统							
			HuaweiCloud EulerOS 2.0（支持GPU虚拟化）	Ubuntu 22.04.4	Ubuntu 22.04.3	CentOS Linux release 7.6	EulerOS release 2.9	EulerOS release 2.5	Ubuntu 18.04（停止维护）	EulerOS release 2.3（停止维护）
Tesla T4	CCE Standard集群	g6 pi2	535.54.03 510.47.03 470.57.02	535.161.08	535.54.03 470.141.03	535.54.03 470.141.03	535.54.03 470.141.03	535.54.03 470.141.03	470.141.03	470.141.03

GPU型号	支持集群类型	机型规格	操作系统							
			HuaweiCloud EulerOS 2.0（支持GPU虚拟化）	Ubuntu 22.04.4	Ubuntu 22.04.3	CentOS Linux release 7.6	EulerOS release 2.9	EulerOS release 2.5	Ubuntu 18.04（停止维护）	EulerOS release 2.3（停止维护）
Volta V100	CCE Standard集群	p2s p2vs p2v	535.54.03 510.47.03 470.57.02	535.161.08	535.54.03 470.141.03	535.54.03 470.141.03	535.54.03 470.141.03	535.54.03 470.141.03	470.141.03	470.141.03

获取驱动链接-公网地址

步骤1 登录CCE控制台。

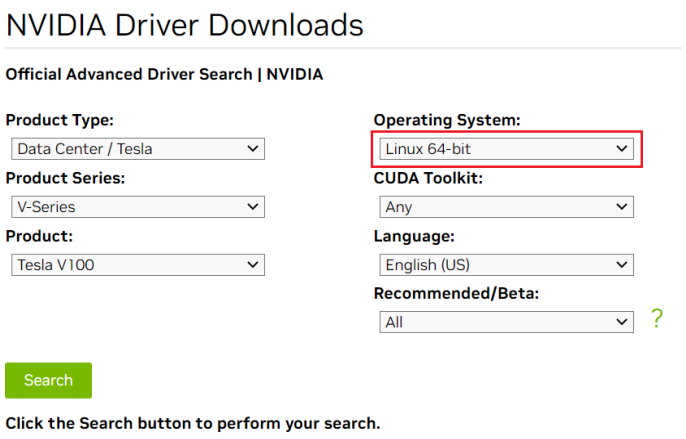
步骤2 创建节点，在节点规格处选择要创建的GPU节点，选中后下方显示的信息中可以看到节点的GPU显卡型号。

图 14-8 查看显卡型号

步骤3 登录到*https://www.nvidia.com/Download/Find.aspx?lang=cn*网站。

步骤4 如图14-9所示，在“NVIDIA驱动程序下载”框内选择对应的驱动信息。其中“操作系统”必须选**Linux 64-bit**。

图 14-9 参数选择



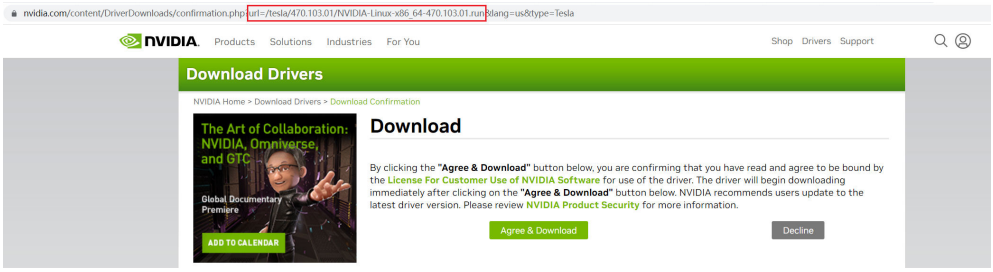
步骤5 驱动信息确认完毕，单击“搜索”按钮，会跳转到驱动信息展示页面，该页面会显示驱动的版本信息如图14-10，单击“下载”到下载页面。

图 14-10 驱动信息

步骤6 获取驱动软件链接方式分两种：

- 方式一：如图14-11，在浏览器的链接中找到路径为*url=/tesla/470.103.01/NVIDIA-Linux-x86_64-470.103.01.run*的路径，补齐全路径**https://us.download.nvidia.com/tesla/470.103.01/NVIDIA-Linux-x86_64-470.103.01.run**该方式节点需要绑定EIP 。
- 方式二：如图14-11，单击“下载”按钮下载驱动，然后上传到OBS，获取软件的链接，该方式节点不需要绑定EIP。

图 14-11 获取链接



----结束

获取驱动链接-OBS 地址

步骤1 将驱动上传到对象存储服务OBS中，并将驱动文件设置为公共读，方法请参见上传文件。

说明

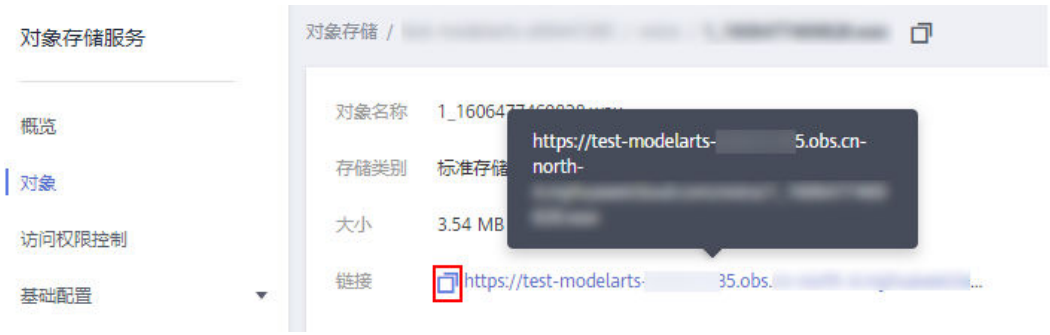
节点重启时会重新下载驱动进行安装，请保证驱动的OBS桶链接长期有效。

步骤2 在桶列表单击待操作的桶，进入“概览”页面。

步骤3 在左侧导航栏，单击“对象”。

步骤4 单击目标对象名称，在对象详情页复制驱动链接。

图 14-12 复制 OBS 链接



----结束

组件说明

表 14-61 gpu 插件组件

容器组件	说明	资源类型
nvidia-driver-installer	为节点安装Nvidia GPU驱动的工作负载，仅在安装场景占用资源，安装完成后无资源占用。	Daemon Set
nvidia-gpu-device-plugin	为容器提供Nvidia GPU异构算力的Kubernetes设备插件。	Daemon Set

容器组件	说明	资源类型
nvidia-operator	为集群提供Nvidia GPU节点管理能力。	Deployment

相关链接

- GPU插件及驱动相关问题的排查思路
- 工作负载异常：GPU相关
- GPU调度

版本记录

表 14-62 CCE AI 套件（NVIDIA GPU）版本记录

插件版本	支持的集群版本	更新特性
2.7.19	v1.28 v1.29 v1.30	修复nvidia-container-toolkit CVE-2024-0132容器逃逸漏洞
2.7.13	v1.28 v1.29 v1.30	• 支持节点池粒度配置XGPU • 支持GPU渲染场景 • 支持v1.30集群
2.6.4	v1.28 v1.29	更新GPU卡逻辑隔离逻辑
2.6.1	v1.28 v1.29	升级GPU插件基础镜像
2.5.6	v1.28	修复安装驱动的问题
2.5.4	v1.28	支持v1.28集群
2.1.14	v1.21 v1.23 v1.25 v1.27	修复nvidia-container-toolkit CVE-2024-0132容器逃逸漏洞
2.1.8	v1.21 v1.23 v1.25 v1.27	修复部分问题

插件版本	支持的集群版本	更新特性
2.0.69	v1.21 v1.23 v1.25 v1.27	升级GPU插件基础镜像
2.0.46	v1.21 v1.23 v1.25 v1.27	• 支持535版本Nvidia驱动 • 支持非root用户使用XGPU • 优化启动逻辑
2.0.18	v1.21 v1.23 v1.25 v1.27	支持HCE 2.0
1.2.28	v1.19 v1.21 v1.23 v1.25	• 适配OS Ubuntu22.04 • GPU驱动目录自动挂载优化
1.2.24	v1.19 v1.21 v1.23 v1.25	• 节点池支持配置GPU驱动版本 • 支持GPU指标采集
1.2.20	v1.19 v1.21 v1.23 v1.25	设置插件别名为gpu
1.2.17	v1.15 v1.17 v1.19 v1.21 v1.23	增加nvidia-driver-install pod limits 配置
1.2.15	v1.15 v1.17 v1.19 v1.21 v1.23	适配CCE v1.23集群

插件版本	支持的集群版本	更新特性
1.2.11	v1.15 v1.17 v1.19 v1.21	支持EulerOS 2.10系统
1.2.10	v1.15 v1.17 v1.19 v1.21	CentOS系统支持新版本GPU驱动
1.2.9	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群
1.2.2	v1.15 v1.17 v1.19	适配EulerOS新内核
1.2.1	v1.15 v1.17 v1.19	• 适配CCE v1.19集群 • 插件增加污点容忍
1.1.13	v1.13 v1.15 v1.17	支持Centos7.6 3.10.0-1127.19.1.el7.x86_64内核系统
1.1.11	v1.15 v1.17	• 支持用户自定义驱动地址下载驱动 • 支持v1.15、v1.17集群

14.4.2 CCE AI 套件（Ascend NPU）

插件简介

CCE AI套件（Ascend NPU）是支持容器里使用huawei NPU设备的管理插件。

安装本插件后，可创建“AI加速型”节点，实现快速高效地处理推理和图像识别等工作。

约束与限制

- 集群中使用“AI加速型”节点时必须安装CCE AI套件（Ascend NPU）插件。
- “AI加速型”节点迁移后会重置节点，需要手动重新安装。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE AI套件（Ascend NPU）**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。您可根据需求调整插件实例数和资源配额。

步骤3 选择是否自动安装驱动（仅插件版本为1.2.5及以上时支持）。

- 开启：可根据NPU机型不同指定相应的驱动版本，驱动维护更灵活。

  根据不同的适用机型选择是否启用驱动，启用后插件将根据用户指定的驱动版本自动进行驱动安装。默认使用“推荐驱动”，您也可以选择“自定义驱动”并填写完整的驱动地址。

  说明

  - 插件将根据用户针对指定机型选择的驱动版本进行驱动安装。仅对未安装NPU驱动的节点生效，已安装 NPU 驱动的节点会保持现状。升级或编辑插件参数时修改驱动版本也只对未安装 NPU 驱动的节点生效。
  - 驱动安装成功后需要重启节点才能生效，驱动安装成功确认方式请参见**如何确认节点NPU驱动已安装完成**。
  - 插件卸载不会自动删除已安装的NPU驱动，如需卸载，卸载方式请参见**NPU驱动卸载**。

- 关闭：无法根据用户诉求指定驱动版本，无法依靠插件进行驱动维护。当不开启驱动选择时，如从控制台创建NPU节点，控制台会自动补充NPU驱动（用户无法指定版本和类型）安装命令，并在安装完成后自动重启节点；如通过API或其他方式创建节点则需要用户在“安装后执行脚本”中添加驱动安装命令。
- 支持的NPU卡类型和对应的操作系统规格如下：

NPU卡类型	支持的操作系统
D310	EulerOS 2.5 x86、CentOS 7.6 x86、EulerOS 2.9 x86、EulerOS 2.8 arm

步骤4 单击“安装”。

----结束

组件说明

表 14-63 CCE AI 套件（Ascend NPU）组件

容器组件	说明	资源类型
npu-driver-installer	该容器运行在NPU节点上，负责安装NPU驱动。	Daemon Set
huawei-npu-device-plugin	支持容器里使用huawei NPU设备的管理插件。	Daemon Set

NPU 指标

指标	监控级别	备注
cce_npu_memory_total	NPU卡	NPU卡显存总量
cce_npu_memory_used	NPU卡	NPU卡显存使用量
cce_npu_utilization	NPU卡	NPU卡算力使用率

如何确认节点 NPU 驱动已安装完成

NPU驱动安装成功后需要重启节点才能生效，且重启节点前需要确认驱动已经安装完成，否则驱动将无法生效，NPU资源不可用。驱动安装完成确认方式如下：

步骤1 在集群 “插件中心”页面，单击插件名称查看插件“实例列表”。

步骤2 查看该节点上部署的 npu-driver-installer 实例状态为“运行中”。

说明

若在NPU驱动安装完成前就重启了节点，可能导致驱动安装失败，节点重启后集群“节点管理”页面对应的节点会显示“驱动未就绪”。此时需要先卸载该节点上的NPU驱动，再重启npu-driver-installer Pod，才能重新安装NPU驱动，按上述步骤确认驱动安装完成后再重启节点。驱动卸载方式请参见**NPU驱动卸载**。

----结束

NPU 驱动卸载

请登录节点，通过 /var/log/ascend_seclog/operation.log 获取驱动操作记录，确认最后一次安装的驱动 run 包；若该日志不存在，则一般使用 npu_x86_latest.run或 npu_arm_latest.run 驱动合一包安装；找到驱动安装包后，执行 bash {run 包名称} --uninstall 命令即可卸载，卸载成功后根据提示决定是否重启节点。

步骤1 登录需要卸载NPU驱动的节点，查看 /var/log/ascend_seclog/operation.log 是否存在。

步骤2 若 /var/log/ascend_seclog/operation.log 日志存在，查看驱动安装日志，可查找到驱动安装记录。

```
[root@y00379955-w-ai1s-e25 ~]# ll /var/log/ascend_seclog/operation.log
-rw-r----- 1 root root 285 Dec  1 20:00 /var/log/ascend_seclog/operation.log
[root@y00379955-w-ai1s-e25 ~]# cat /var/log/ascend_seclog/operation.log
Install SUGGESTION root 2022-12-01 19:53:47 127.0.0.1 Ascend310-hdk-npu-driver 6.0.rc1 linux-x86-64.run success install_type=full; cmdlist=--quiet --full.
```

若 /var/log/ascend_seclog/operation.log 日志不存在，则可能是通过 npu_x86_latest.run或 npu_arm_latest.run 驱动合一包安装的，可通过 /usr/local/HiAI/driver/ 路径是否存在进行确认。

说明

NPU驱动合一包一般放在 /root/d310_driver 目录下，其他驱动安装包一般放在 /root/npu-drivers 目录下。

步骤3 找到驱动安装包后，执行 **bash {run 包路径} --uninstall** 命令即可卸载，以 Ascend310-hdk-npu-driver_6.0.rc1_linux-x86-64.run 为例：

```
bash /root/npu-drivers/Ascend310-hdk-npu-driver_6.0.rc1_linux-x86-64.run --uninstall
```

```
[root@y00379955-w-ai1s-e25 npu-drivers]# ./Ascend310-hdk-npu-driver_6.0.rc1_linux-x86-64.run --uninstall
Verifying archive integrity...  100%   SHA256 checksums are OK. All good.
Uncompressing ASCEND DRIVER RUN PACKAGE  100%
[Driver] [2022-12-01 19:59:53] [INFO]Start time: 2022-12-01 19:59:53
[Driver] [2022-12-01 19:59:53] [INFO]LogFile: /var/log/ascend_seclog/ascend_install.log
[Driver] [2022-12-01 19:59:53] [INFO]OperationLogFile: /var/log/ascend_seclog/operation.log
[Driver] [2022-12-01 19:59:53] [INFO]base version is 22.0.3.


[Driver] [2022-12-01 20:00:04] [INFO]Driver package uninstalled successfully! Reboot needed for uninstallation to take effect!
[Driver] [2022-12-01 20:00:04] [INFO]End time: 2022-12-01 20:00:04
```

步骤4 根据提示信息确认是否需要重启节点（当前使用的NPU驱动安装和卸载都需要重启节点才能生效）。

----结束

版本记录

表 14-64 CCE AI 套件（Ascend NPU）插件版本记录

插件版本	支持的集群版本	更新特性
2.1.23	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	修复部分问题
2.1.22	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 修复了一些页面显示问题 • 支持查询超节点信息 • 支持上报显卡拓扑信息 • 修复了日志打印问题
2.1.14	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	修复部分问题
2.1.7	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题
2.1.5	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	• 适配CCE v1.29集群 • 新增静默故障码

插件版本	支持的集群版本	更新特性
2.0.9	v1.21 v1.23 v1.25 v1.27 v1.28	修复进程级故障恢复和给工作负载添加注解偶现失败问题
2.0.5	v1.21 v1.23 v1.25 v1.27 v1.28	• 适配CCE v1.28集群 • 支持存活探针检查机制
1.2.14	v1.19 v1.21 v1.23 v1.25 v1.27	支持NPU监控
1.2.6	v1.19 v1.21 v1.23 v1.25	支持NPU驱动自动安装
1.2.5	v1.19 v1.21 v1.23 v1.25	支持NPU驱动自动安装
1.2.4	v1.19 v1.21 v1.23 v1.25	适配CCE v1.25集群
1.2.2	v1.19 v1.21 v1.23	适配CCE v1.23集群
1.2.1	v1.19 v1.21 v1.23	适配CCE v1.23集群

插件版本	支持的集群版本	更新特性
1.1.8	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群
1.1.2	v1.15 v1.17 v1.19	配置seccomp默认规则
1.1.1	v1.15 v1.17 v1.19	兼容CCE v1.15集群
1.1.0	v1.17 v1.19	适配CCE v1.19集群
1.0.8	v1.13 v1.15 v1.17	适配D310 C75驱动
1.0.6	v1.13 v1.15 v1.17	支持C75驱动
1.0.5	v1.13 v1.15 v1.17	支持容器里使用huawei NPU设备的管理插件
1.0.3	v1.13 v1.15 v1.17	支持容器里使用huawei NPU设备的管理插件

14.5 容器网络插件

14.5.1 CoreDNS 域名解析

插件简介

CoreDNS域名解析插件是一款通过链式插件的方式为Kubernetes提供域名解析服务的DNS服务器。

CoreDNS是由CNCF孵化的开源软件，用于Cloud-Native环境下的DNS服务器和服务发现解决方案。CoreDNS实现了插件链式架构，能够按需组合插件，运行效率高、配置灵活。在Kubernetes集群中使用CoreDNS能够自动发现集群内的服务，并为这些服务

提供域名解析。同时，通过级联云上DNS服务器，还能够为集群内的工作负载提供外部域名的解析服务。

该插件为系统资源插件，在创建集群时默认安装。

目前CoreDNS已经成为社区Kubernetes集群推荐的DNS服务器解决方案。

CoreDNS官网：**https://coredns.io/**

开源社区地址：**https://github.com/coredns/coredns**

说明

DNS详细使用方法请参见**DNS**。

约束与限制

CoreDNS域名解析插件正常运行或升级时，请确保集群中的可用节点数大于等于插件的实例数，且所有实例都处于运行状态，否则将导致插件异常或升级失败。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**CoreDNS域名解析**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据并发域名解析能力选择“小规格”、“中规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”的外部域名解析能力为2500QPS，内部域名解析能力为10000QPS；
  “中规格”的外部域名解析能力为5000QPS，内部域名解析能力为20000QPS；
  “大规格”的外部域名解析能力为10000QPS，内部域名解析能力为40000QPS。

- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。CoreDNS所能提供的域名解析QPS与CPU消耗成正相关，集群中的节点/容器数量增加时，CoreDNS实例承受的压力也会同步增加。请根据集群的规模，合理调整插件实例数和容器CPU/内存配额，配置建议请参见**表14-65**。

表 14-65 CoreDNS 插件配额建议

节点数量	推荐配置	实例数	CPU申请值	CPU限制值	内存申请值	内存限制值
50	2500QPS	2	500m	500m	512Mi	512Mi
200	5000QPS	2	1000m	1000m	1024Mi	1024Mi
1000	10000QPS	2	2000m	2000m	2048Mi	2048Mi
2000	20000QPS	4	2000m	2000m	2048Mi	2048Mi

步骤3 通过界面设置插件支持的“参数配置”。

表 14-66 CoreDNS 插件参数配置

参数	参数说明
存根域设置	对自定义的域名配置域名服务器，格式为一个键值对，键为DNS后缀域名，值为一个或一组DNS IP地址，如 'acme.local --1.2.3.4,6.7.8.9'。 详情请参见**为CoreDNS配置存根域**。

参数	参数说明

扩展参数配置	• parameterSyncStrategy：插件升级时是否配置一致性检查。
	– ensureConsistent：表示启用配置一致性检查，如果升级插件时下发的配置和当前生效配置不一致，插件将无法升级。
	– force：表示升级时忽略配置一致性检查。将以升级插件时下发的配置为准，请您自行确保升级插件时下发的配置和当前生效配置一致。插件升级完毕后，需将parameterSyncStrategy参数值恢复为ensureConsistent，重新启用配置一致性检查。
	– inherit：如果升级插件时下发的配置和当前生效配置不一致，将忽略升级插件时下发的配置，以当前生效配置为准。插件升级完毕后，parameterSyncStrategy的参数值将自动恢复为ensureConsistent，重新启用配置一致性检查。
	• servers: CoreDNS 1.23.1插件版本开始开放servers配置，用户可对servers做定制化配置，详情请参见**dns-custom-nameservers**。
	其中plugins为CoreDNS中各组件配置。一般场景建议保持默认配置，以免出现配置错误而导致CoreDNS整体不可用。每个plugin组件可包含"name"、"parameters"(可选)、"configBlock"(可选)配置，对应生成的Corefile配置文件中格式如下：
	`$name  $parameters {`
	`$configBlock`
	`}`
	常用plugin的说明请参见**表14-67**。更多配置详情请参见**Plugins**。
	• upstream_nameservers：上游域名服务器地址。
	高级配置示例：
	`{`
	`"annotations": {},`
	`"parameterSyncStrategy": "ensureConsistent",`
	`"servers": [`
	`{`
	`"plugins": [`
	`{`
	`"name": "bind",`
	`"parameters": "{$POD_IP}"`
	`},`
	`{`
	`"name": "cache",`
	`"configBlock": "servfail 5s",`
	`"parameters": 30`
	`},`
	`{`
	`"name": "errors"`
	`},`
	`{`
	`"name": "health",`
	`"parameters": "{$POD_IP}:8080"`
	`},`
	`{`
	`"name": "ready",`
	`"parameters": "{$POD_IP}:8081"`

参数	参数说明

	<pre>{
	            },
	            {
	               "configBlock": "pods insecure\nfallthrough in-addr.arpa ip6.arpa",
	               "name": "kubernetes",
	               "parameters": "cluster.local in-addr.arpa ip6.arpa"
	            },
	            {
	               "name": "loadbalance",
	               "parameters": "round_robin"
	            },
	            {
	               "name": "prometheus",
	               "parameters": "{$POD_IP}:9153"
	            },
	            {
	               "configBlock": "policy random",
	               "name": "forward",
	               "parameters": ". /etc/resolv.conf"
	            },
	            {
	               "name": "reload"
	            }
	         ],
	         "port": 5353,
	         "zones": [
	            {
	               "zone": "."
	            }
	         ]
	      }
	   ],
	   "upstream_nameservers": ["8.8.8.8", "8.8.4.4"]
	}</pre>

表 14-67 CoreDNS 默认配置说明

plugin名称	类型	描述
bind	默认配置	CoreDNS侦听的hostIP配置，建议保持当前默认值{$POD_IP}。详情请参见**bind**。
cache	默认配置	启用DNS缓存。详情请参见**cache**。 插件版本>=1.25.10时，支持关闭servfail缓存。如需关闭servfail缓存，请将configBlock设置为"servfail 0"；否则servfail缓存的单位为s，不可省略。
errors	默认配置	错误信息到标准输出。详情请参见**errors**。
health	默认配置	CoreDNS健康检查配置，当前侦听{$POD_IP}:8080，请保持此默认值，否则导致coredns健康检查失败而不断重启。详情请参见**health**。
ready	默认配置	检查后端服务是否准备好接收流量，当前侦听{$POD_IP}:8081。如果后端服务没有准备好，CoreDNS将会暂停 DNS 解析服务，直到后端服务准备好为止。详情请参见**ready**。

plugin名称	类型	描述
kubernetes	默认配置	CoreDNS Kubernetes插件，提供集群内服务解析能力。详情请参见**kubernetes**。
loadbalance	默认配置	轮转式 DNS 负载均衡器，在应答中随机分配A、AAAA和MX记录的顺序。详情请参见**loadbalance**。
prometheus	默认配置	CoreDNS自身metrics数据接口，默认侦听{$POD_IP}:9153，请保持此默认值，否则普罗无法采集coredns metrics数据。详情请参见**prometheus**。
forward	默认配置	不在 Kubernetes 集群域内的任何查询都将转发到默认的解析器 (/etc/resolv.conf)。详情请参见**forward**。
reload	默认配置	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。详情请参见**reload**。
log	扩展配置	开启CoreDNS域名解析的日志。详情请参见**log**。 示例如下： `{ "name": "log" }`
template	扩展配置	设置快速应答模板，AAAA表示IPv6解析请求，rcode控制应答返回NXDOMAIN，即表示没有IPv6解析结果。详情请参见**template**。 示例如下： `{ "configBlock": "rcode NXDOMAIN", "name": "template", "parameters": "ANY AAAA" }`

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-68 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 完成以上配置后，单击“安装”。

----结束

使用 Corefile 配置 CoreDNS 插件

说明

安装CoreDNS插件时，不支持使用Corefile视图配置，仅编辑/升级场景支持使用Corefile视图配置。

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**CoreDNS域名解析**插件，单击“编辑”。

步骤2 在“参数配置”中，选择是否切换Corefile视图（1.30.3及以上版本的插件支持）。

切换后将通过Corefile格式直接配置kube-system命名空间下的CoreDNS的ConfigMap，且已有的存根域配置和高级配置内 parameterSyncStrategy/servers/upstream_nameservers等参数均不再生效，您需要自行保证Corefile配置的正确性。

关于Corefile格式的说明请参考**如何配置Corefile**。

说明

- 关闭Corefile视图后，依旧会以存根域配置和高级配置内 parameterSyncStrategy/servers/upstream_nameservers 等参数配置CoreDNS的配置项，切换时需要保证配置的合理性。
- 开启Corefile视图后更新/升级插件，再次关闭Corefile视图更新/升级插件，插件更新会拦截当前的配置，需要将 parameterSyncStrategy设置为force或者inherit策略后才能完成升级。
- 修改Corefile配置项后，需要等待CoreDNS内部的Reload机制自动完成配置刷新（约在十秒内配置生效）。

步骤3 完成Corefile编辑后，单击“确定”。

----结束

组件说明

表 14-69 coredns 组件

容器组件	说明	资源类型
coredns	该容器为提供集群域名解析服务的DNS服务器。	Deployment

Kubernetes 中的域名解析逻辑

DNS策略可以在每个pod基础上进行设置，目前，Kubernetes支持**Default**、**ClusterFirst**、**ClusterFirstWithHostNet**和**None**四种DNS策略，具体请参见**Service与Pod的DNS**。这些策略在pod-specific的**dnsPolicy**字段中指定。

- **“Default”**：如果dnsPolicy被设置为“Default”，则名称解析配置将从pod运行的节点继承。 自定义上游域名服务器和存根域不能够与这个策略一起使用。
- **“ClusterFirst”**：如果dnsPolicy被设置为“ClusterFirst”，任何与配置的集群域后缀不匹配的DNS查询（例如，www.kubernetes.io）将转发到从该节点继承的上游名称服务器。集群管理员可能配置了额外的存根域和上游DNS服务器。
- **“ClusterFirstWithHostNet”**：对于使用hostNetwork运行的Pod，您应该明确设置其DNS策略“ClusterFirstWithHostNet”。
- **“None”**：它允许Pod忽略Kubernetes环境中的DNS设置。应使用dnsConfigPod规范中的字段提供所有DNS设置 。

说明

- Kubernetes 1.10及以上版本，支持Default、ClusterFirst、ClusterFirstWithHostNet和None四种策略；低于Kubernetes 1.10版本，仅支持default、ClusterFirst和ClusterFirstWithHostNet三种。
- “Default”不是默认的DNS策略。如果dnsPolicy的Flag没有特别指明，则默认使用“**ClusterFirst**”。

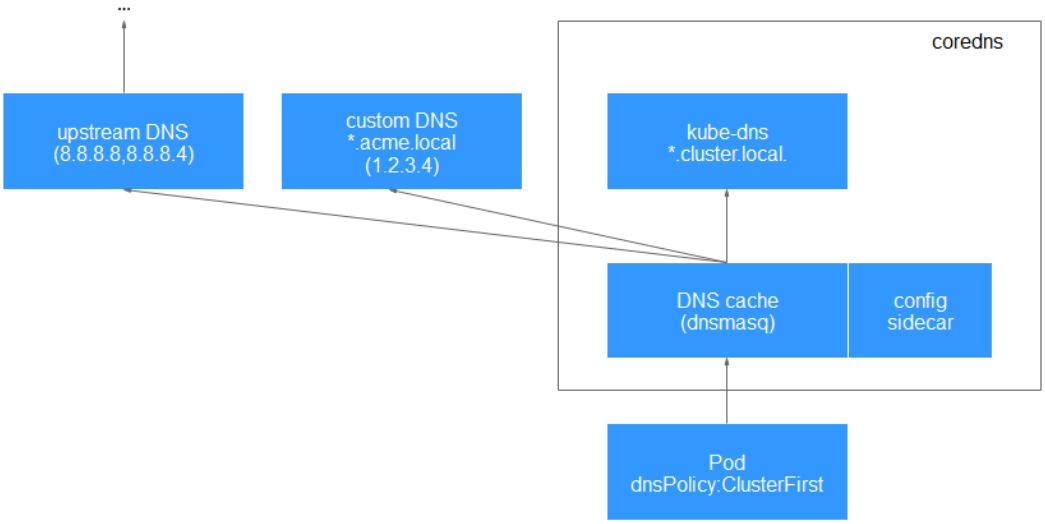
路由请求流程：

未配置存根域：没有匹配上配置的集群域名后缀的任何请求，例如“www.kubernetes.io”，将会被转发到继承自节点的上游域名服务器。

已配置存根域：如果配置了存根域和上游DNS服务器，DNS查询将基于下面的流程对请求进行路由：

1. 查询首先被发送到coredns中的DNS缓存层。
2. 从缓存层，检查请求的后缀，并根据下面的情况转发到对应的DNS上：
   - 具有集群后缀的名字（例如“.cluster.local”）：请求被发送到coredns。
   - 具有存根域后缀的名字（例如“.acme.local”）：请求被发送到配置的自定义DNS解析器（例如：监听在 1.2.3.4）。
   - 未能匹配上后缀的名字（例如“widget.com”）：请求被转发到上游DNS。

图 14-13 路由请求流程



版本记录

表 14-70 CoreDNS 域名解析插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.30.6	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	• 支持Corefile配置 • 适配CCE v1.30集群	1.10.1

插件版本	支持的集群版本	更新特性	社区版本
1.29.5	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题	1.10.1
1.29.4	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群	1.10.1
1.28.7	v1.21 v1.23 v1.25 v1.27 v1.28	支持插件热更新配置，无需滚动升级	1.10.1
1.28.5	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	1.10.1
1.28.4	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28集群	1.10.1
1.27.4	v1.19 v1.21 v1.23 v1.25 v1.27	-	1.10.1
1.25.14	v1.19 v1.21 v1.23 v1.25	• 支持插件规格与集群规格联动 • 插件与节点时区一致	1.10.1

插件版本	支持的集群版本	更新特性	社区版本
1.25.11	v1.19 v1.21 v1.23 v1.25	• 支持插件实例AZ反亲和配置 • 升级至社区版本 1.10.1	1.10.1
1.25.1	v1.19 v1.21 v1.23 v1.25	适配CCE v1.25集群	1.8.4
1.23.3	v1.15 v1.17 v1.19 v1.21 v1.23	插件依赖例行升级	1.8.4
1.23.2	v1.15 v1.17 v1.19 v1.21 v1.23	插件依赖例行升级	1.8.4
1.23.1	v1.15 v1.17 v1.19 v1.21 v1.23	适配CCE v1.23集群	1.8.4
1.17.15	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群	1.8.4
1.17.9	v1.15 v1.17 v1.19	插件依赖例行升级	1.8.4
1.17.7	v1.15 v1.17 v1.19	同步至社区v1.8.4版本	1.8.4
1.17.4	v1.17 v1.19	适配CCE v1.19集群	1.6.5

插件版本	支持的集群版本	更新特性	社区版本
1.17.3	v1.17	支持v1.17集群，修复存根域配置问题	1.6.5
1.17.1	v1.17	支持v1.17集群	1.6.5

14.5.2 NGINX Ingress 控制器

插件简介

Kubernetes通过kube-proxy服务实现了Service的对外发布及负载均衡，它的各种方式都是基于传输层实现的。在实际的互联网应用场景中，不仅要实现单纯的转发，还有更加细致的策略需求，如果使用真正的负载均衡器更会增加操作的灵活性和转发性能。

基于以上需求，Kubernetes引入了资源对象Ingress，Ingress为Service提供了可直接被集群外部访问的虚拟主机、负载均衡、SSL代理、HTTP路由等应用层转发功能。

Kubernetes官方发布了基于Nginx的Ingress控制器，CCE的NGINX Ingress控制器插件直接使用社区模板与镜像。Nginx Ingress控制器会将Ingress生成一段Nginx的配置，并将Nginx配置通过ConfigMap进行储存，这个配置会通过Kubernetes API写到Nginx的Pod中，然后完成Nginx的配置修改和更新，详细工作原理请参见**工作原理**。

开源社区地址：**https://github.com/kubernetes/ingress-nginx**

说明

- 2.3.3及以上版本的Nginx Ingress默认仅支持TLS v1.2及v1.3版本，如果客户端TLS版本低于v1.2，会导致客户端与Nginx Ingress协商时报错。如果需要支持更多TLS版本，请参见**TLS/HTTPS**。
- 安装该插件时，您可以通过“nginx配置参数”添加配置，此处的设置将会全局生效，该参数直接通过配置nginx.conf生成，将影响管理的全部Ingress，相关参数可通过**ConfigMaps**查找，如果您配置的参数不包含在**ConfigMaps**所列出的选项中将不会生效。
- 安装该插件后，您在CCE控制台**创建Ingress**时可以选择对接Nginx控制器，并通过“注解”设置Nginx Ingress功能，支持的注解字段详情请参见**Annotations**。
- 请勿手动修改和删除CCE自动创建的ELB和监听器，否则将出现工作负载异常；若您已经误修改或删除，请卸载Nginx Ingress插件后重装。

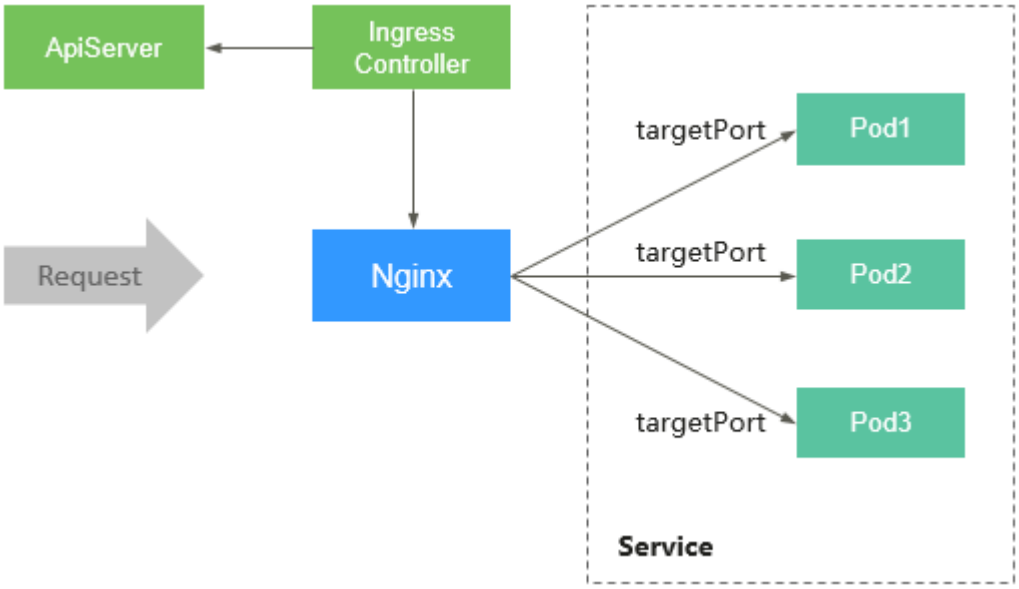
工作原理

Nginx Ingress由资源对象Ingress、Ingress控制器、Nginx三部分组成，Ingress控制器用以将Ingress资源实例组装成Nginx配置文件（nginx.conf），并重新加载Nginx使变更的配置生效。当它监听到Service中Pod变化时通过动态变更的方式实现Nginx上游服务器组配置的变更，无须重新加载Nginx进程。工作原理如**图14-14**所示。

- Ingress：一组基于域名或URL把请求转发到指定Service实例的访问规则，是Kubernetes的一种资源对象，Ingress实例被存储在对象存储服务etcd中，通过接口服务被实现增、删、改、查的操作。
- Ingress控制器（Ingress Controller）：用以实时监控资源对象Ingress、Service、End-point、Secret（主要是TLS证书和Key）、Node、ConfigMap的变化，自动对Nginx进行相应的操作。

- Nginx：实现具体的应用层负载均衡及访问控制。

图 14-14 Nginx Ingress 工作原理



注意事项

- 对于v1.23版本前的集群，通过API接口创建的Ingress在注解中必须添加kubernetes.io/ingress.class: "*nginx*"。如果集群中安装了多套NGINX Ingress控制器，需将*nginx*替换为自定义的**控制器名称**，用于识别Ingress对接的控制器实例。
- 独享型ELB规格必须支持网络型（TCP/UDP），且网络类型必须支持私网（有私有IP地址）。
- 运行Nginx Ingress控制器实例的节点以及该节点上运行的容器，无法访问Nginx Ingress，请将工作负载Pod与Nginx Ingress控制器实例进行反亲和部署，具体操作步骤请参见**工作负载与Nginx Ingress控制器实例反亲和部署**。
- Nginx Ingress控制器实例在升级时会预留10s的宽限时间，用于删除ELB后端的Nginx Ingress控制器。
- Nginx Ingress控制器实例的优雅退出时间为300s，若插件升级时存在超过300s的长连接，长连接会被断开，出现服务短暂中断。

前提条件

在安装此插件之前，您需要存在一个可用集群，且集群中包含一个可用节点。若没有可用集群，请参照**购买Standard/Turbo集群**中的步骤创建。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**NGINX Ingress控制器**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件支持的“参数配置”。

- **控制器名称**：自定义控制器名称，该名称为Ingress控制器的唯一标识，同一个集群中不同的控制器名称必须唯一，且不能设置为**cce**（**cce**是ELB Ingress Controller的唯一标识）。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。
- **插件安装的命名空间**：选择Ingress控制器所在的命名空间。
- **负载均衡器**：支持对接共享型或独享型负载均衡实例，如果无可用实例，请先创建。负载均衡器需要拥有至少两个监听器配额，且端口 80 和 443 没有被监听器占用。

  选择共享型负载均衡实例，支持开启“获取客户端IP”功能，开启后使用Nginx Ingress访问应用服务端，服务端即可获取客户端的源IP。而使用独享型负载均衡实例时默认开启获取客户端IP，无需配置。

  说明

  该功能在v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本的集群支持开启。

  开启/关闭“获取客户端IP”的过程中，如果已经配置了后端服务，则访问流量会中断，请谨慎操作。
- **开启准入校验**：针对Ingress资源的准入控制，以确保控制器能够生成有效的配置。开启后将会对Nginx类型的Ingress资源配置做准入校验，若校验失败，请求将被拦截。关于准入校验详情，请参见**准入控制**。

  说明

  - 开启准入校验后，会在一定程度上影响Ingress资源的请求响应速度。
  - 仅2.4.1及以上版本的插件支持开启准入校验。
- **nginx配置参数**：配置nginx.conf文件，将影响管理的全部Ingress。您可以选择使用“界面化配置”或“YAML配置”，其中“界面化配置”在2.2.75、2.6.26、3.0.1及以上版本的NGINX Ingress控制器插件中支持。

  如果您需要配置社区支持的自定义参数，请选择“YAML配置”，相关参数可通过**ConfigMaps**查找。此处以设置keep-alive-requests参数为例，设置保持活动连接的最大请求数为100。

  ```
  {
    "keep-alive-requests": "100"
  }
  ```

  说明

  - 如果您配置的参数不包含在**ConfigMaps**所列出的选项中，配置将不会生效。
  - 配置参数的值必须为字符串格式，否则无法安装成功。

  CCE在2.2.75、2.6.26、3.0.1及以上版本的NGINX Ingress控制器插件中默认支持可视化页面配置以下配置参数。

配置参数名称	nginx配置参数	说明	默认值
Work最大连接数	max-worker-connections	每个NGINX工作进程能够同时处理的最大连接数。这个参数是用来控制工作进程的负载量的，高并发环境下需要设置较高的值以确保系统稳定性。注意，此处不仅包含客户端连接，还包括到后端服务器的连接。	65536
Keepalive链接最大请求数	keep-alive-requests	用于控制单个keepalive连接可以处理的最大请求数。当达到最大请求数时，连接将被关闭。	100
上游服务器最大保持连接数	upstream-keepalive-connections	激活与上游服务器连接的缓存。该参数设置每个工作进程中保留在缓存中的闲置keepalive连接的最大数量。当超过这个数字时，最久未使用的连接将被关闭。	320
上游服务器最大连接时间	upstream-keepalive-timeout	上游服务器与后端服务器建立的keepalive连接的超时时间，单位是秒。表示NGINX在这段时间内可以保持连接以供重用，这样可以减少建立新连接所需的开销，在高QPS场景下能显著提高性能。	900
请求超时时间	proxy-connect-timeout	客户端到代理服务器间建立连接的超时时间，单位是秒。如果在10秒内无法连接到后端服务器，NGINX将返回502 (Bad Gateway) 错误。适合需要连接速度较快的应用场景。	10
请求体最大值	proxy-body-size	指定NGINX代理发送到后端服务器时，可以接受的请求体的最大值。这个值限制了上传文件或者提交大数据表单的大小。如果请求体超过了这个值，将返回413 (Payload Too Large) 错误。	20m

配置参数名称	nginx配置参数	说明	默认值
允许后端返回Server标头信息	allow-backend-server-header	通常情况下，NGINX会剥离后端服务器的Server头部信息，也就是服务器向客户端发送的标识信息。如设置为true，则NGINX将允许后端服务器的Server头部信息直接传递给客户端。从安全角度来看，最好将其关闭来避免泄露服务器类型及版本的信息。	关闭
允许标头中带下划线	enable-underscores-in-headers	某些HTTP头部可能包含下划线（例如X_Custom-Header），但是依据RFC标准不建议这样使用，因此许多服务器默认不允许这样做。如果您有需要使用这种头部信息的情况，比如某些第三方服务或客户端使用了下划线命名的头部信息，可以启用这个参数。	关闭
生成请求ID	generate-request-id	每个请求被收到时，NGINX会生成一个唯一的请求ID，这个ID通常会被记录到日志中，或者通过头部信息传递到后端服务器。这对于请求的追踪和调试来说非常有用，尤其是在分布式系统中定位问题。	开启
忽略无效标头	ignore-invalid-headers	在接收到包含无效头部的HTTP请求时，NGINX默认会拒绝该请求。这个配置项开启后，NGINX将会忽略这些无效的头部信息并继续处理请求。在面对一些不完全符合HTTP标准的客户端时比较有用。	开启
启用端口重用	reuse-port	启用SO_REUSEPORT选项允许多进程/线程绑定到相同的IP/Port组合，这样可以有效地提高服务器的并发性能，特别是多核CPU情况下。它允许在同一个端口上接受更多的新连接。	开启
响应体中携带Server信息	server-tokens	关闭NGINX默认在响应头中加入的Server信息，这个信息通常会包含NGINX版本。禁用这个选项有助于隐藏服务器的信息，从而增强安全性，避免潜在的攻击者利用版本信息来进行攻击。	关闭

配置参数名称	nginx配置参数	说明	默认值
支持HTTP协议自动重定向为HTTPS	ssl-redirect	禁用从HTTP自动重定向到HTTPS。例如，默认情况下如果您只希望在某些条件下（如登录页面）使用HTTPS而其他页面使用HTTP，那么您可以禁用这个选项来避免默认的重定向行为。	关闭
工作线程CPU亲和性	worker-cpu-affinity	自动分配工作进程到特定的CPU核心，提高多核系统的性能。比如在多核服务器上，可以使某些工作进程固定在特定的CPU核上；这样可以减少上下文切换，提高处理效率。	自动亲和

- **开启指标采集**：插件版本不低于2.4.12时，支持采集Prometheus监控指标。具体操作详情请参见**监控NGINX Ingress控制器指标**。
- **服务器默认证书**：选择一个IngressTLS或kubernetes.io/tls类型的密钥，用于配置Nginx Ingress控制器启动时的默认证书。如果无可选密钥，您可以单击“创建TLS类型的密钥证书”进行新建，详情请参见**创建密钥**。关于默认证书更多说明请参见**Default SSL Certificate**。
- **404服务**：默认使用插件自带的404服务。支持自定义404服务，填写“命名空间/服务名称”，如果服务不存在，插件会安装失败。
- **添加TCP/UDP服务**：Nginx Ingress默认仅支持转发外部HTTP和HTTPS流量，通过添加TCP/UDP端口映射，可实现转发外部TCP/UDP流量到集群内服务。关于添加TCP/UDP服务的更多信息，请参见**暴露TCP/UDP服务**。
  - 协议：选择TCP或UDP。
  - 服务端口：ELB监听器使用的端口，端口范围为1-65535。
  - 目标服务命名空间：请选择Service所在的命名空间。
  - 目标服务名称：请选择已有Service。页面列表中的查询结果已自动过滤不符合要求的Service。
  - 目标服务访问端口：可选择目标Service的访问端口。

  说明

  - 集群版本为v1.19.16-r5、v1.21.8-r0、v1.23.6-r0及以上时，支持设置TCP/UDP混合协议能力。
  - 集群版本为v1.19.16-r5、v1.21.8-r0、v1.23.6-r0、v1.25.2-r0及以上时，支持设置TCP/UDP混合协议使用相同的对外端口。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-71 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

----结束

安装多个 NGINX Ingress 控制器

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到已经安装的**NGINX Ingress控制器**插件，单击“新增”。

步骤2 在安装插件页面，重新配置NGINX Ingress控制器参数，参数说明详情请参见**安装插件**。

步骤3 参数配置完成后，单击“安装”。

步骤4 等待插件安装指令下发完成，您可以返回“插件中心”，单击“管理”，在插件详情页查看已安装的控制器实例。

----结束

组件说明

表 14-72 NGINX Ingress 控制器插件组件

容器组件	说明	资源类型
cceaddon-nginx-ingress-<控制器名称>-controller （2.5.4之前版本中的名称为cceaddon-nginx-ingress-controller）	基于Nginx的Ingress控制器，为集群提供灵活的路由转发能力。	Deployment
cceaddon-nginx-ingress-<控制器名称>-backend （2.5.4之前版本中的名称为cceaddon-nginx-ingress-default-backend）	Nginx的默认后端。返回“default backend - 404”。	Deployment

工作负载与 Nginx Ingress 控制器实例反亲和部署

运行Nginx Ingress控制器实例的节点以及该节点上运行的容器，无法访问Nginx Ingress，为避免这个问题发生，需要将工作负载与Nginx Ingress控制器实例反亲和部署，即工作负载Pod无法调度至Nginx Ingress控制器实例运行的节点。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:aplpine
        imagePullPolicy: IfNotPresent
        name: nginx
      imagePullSecrets:
      - name: default-secret
      affinity:
        podAntiAffinity:
```



```
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:    #通过Nginx Ingress控制器实例的标签实现反亲和
                - key: app
                  operator: In
                  values:
                    - nginx-ingress   #如果集群中安装了多套Nginx Ingress控制器，对应的标签值为nginx-ingress-<控制器名称>
                - key: component
                  operator: In
                  values:
                    - controller
            namespaces:
              - kube-system
            topologyKey: kubernetes.io/hostname
```

版本记录

表 14-73 NGINX Ingress 控制器插件 3.0.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
3.0.8	v1.27 v1.28 v1.29 v1.30	• 更新至社区v1.11.2版本 • 修复CVE-2024-7646漏洞	1.11.2

表 14-74 NGINX Ingress 控制器插件 2.6.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.6.32	v1.25 v1.27 v1.28 v1.29	修复部分问题	1.9.6
2.6.5	v1.25 v1.27 v1.28 v1.29	支持在启动命令中关闭指标采集	1.9.6
2.6.4	v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群	1.9.6

表 14-75 NGINX Ingress 控制器插件 2.5.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.5.6	v1.25 v1.27 v1.28	修复部分问题	1.9.3
2.5.4	v1.25 v1.27 v1.28	• 同一集群支持安装多套NGINX Ingress控制器 • 支持通过控制台配置nginx-ingress默认证书 • 支持将NGINX Ingress控制器指标上报至Prometheus	1.9.3

表 14-76 NGINX Ingress 控制器插件 2.4.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.4.6	v1.25 v1.27 v1.28	• 适配CCE v1.28集群 • 支持开启准入校验 • 支持优雅退出、无损升级能力 • 插件多可用区部署模式支持选择均匀分布 • 修复CVE-2023-44487漏洞	1.9.3

表 14-77 NGINX Ingress 控制器插件 2.3.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.3.5	v1.27	-	1.8.0

表 14-78 NGINX Ingress 控制器插件 2.2.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.2.82	v1.23 v1.25	修复部分问题	1.5.1
2.2.53	v1.23 v1.25	修复部分问题	1.5.1

插件版本	支持的集群版本	更新特性	社区版本
2.2.52	v1.23 v1.25	• 同一集群支持安装多套NGINX Ingress控制器 • 支持通过控制台配置nginx-ingress默认证书	1.5.1
2.2.42	v1.23 v1.25	• 支持优雅退出、无损升级能力 • 插件多可用区部署模式支持选择均匀分布	1.5.1
2.2.7	v1.25	• 插件挂载节点时区 • 支持双栈	1.5.1
2.2.3	v1.25	• 支持插件实例AZ反亲和配置 • 对创建临时存储卷的POD添加不可调度容忍时间 • 默认污点容忍时长修改为60s	1.5.1
2.2.1	v1.25	• 适配CCE v1.25集群 • 更新至社区v1.5.1版本	1.5.1

表 14-79 NGINX Ingress 控制器插件 2.1.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.1.54	v1.19 v1.21 v1.23	修复部分问题	1.2.1
2.1.33	v1.19 v1.21 v1.23	• 支持优雅退出、无损升级能力 • 插件多可用区部署模式支持选择均匀分布	1.2.1
2.1.9	v1.19 v1.21 v1.23	• 支持插件实例AZ反亲和配置 • 默认污点容忍时长修改为60s • 插件挂载节点时区 • 支持双栈	1.2.1

插件版本	支持的集群版本	更新特性	社区版本
2.1.5	v1.19 v1.21 v1.23	• 支持插件实例AZ反亲和配置 • 默认污点容忍时长修改为60s	1.2.1
2.1.3	v1.19 v1.21 v1.23	nginx-ingress支持开启publishService开关	1.2.1
2.1.1	v1.19 v1.21 v1.23	更新至社区v1.2.1版本	1.2.1
2.1.0	v1.19 v1.21 v1.23	• 更新至社区v1.2.0版本 • 修复CVE-2021-25746漏洞，新增规则禁用一些存在越权风险的Anntotations值 • 修复CVE-2021-25745漏洞，新增规则禁用一些存在越权风险的访问路径	1.2.0

表 14-80 NGINX Ingress 控制器插件 2.0.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
2.0.1	v1.19 v1.21 v1.23	• 适配CCE v1.23集群 • 更新至社区v1.1.1版本	1.1.1

表 14-81 NGINX Ingress 控制器插件 1.3.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
1.3.2	v1.15 v1.17 v1.19 v1.21	• 适配CCE v1.21集群 • 同步至社区v0.49.3版本	0.49.3

表 14-82 NGINX Ingress 控制器插件 1.2.x 版本发布记录

插件版本	支持的集群版本	更新特性	社区版本
1.2.6	v1.15 v1.17 v1.19	配置seccomp默认规则	**0.46.0**
1.2.5	v1.15 v1.17 v1.19	同步至社区v0.46.0版本	**0.46.0**
1.2.3	v1.15 v1.17 v1.19	适配CCE v1.19集群	**0.43.0**
1.2.2	v1.15 v1.17	同步至社区v0.43.0版本	**0.43.0**

14.5.3 节点本地域名解析加速

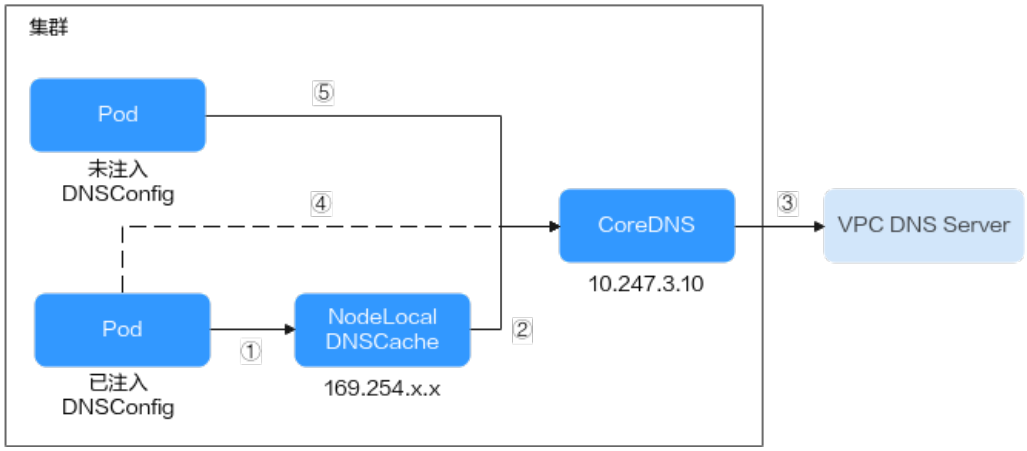
插件简介

节点本地域名解析加速（node-local-dns）是基于社区**NodeLocal DNSCache**提供的插件，通过在集群节点上作为守护程序集运行DNS缓存代理，提高集群DNS性能。

开源社区地址：**https://github.com/kubernetes/dns**

启用NodeLocal DNSCache之后，DNS查询所遵循的路径如下图所示。

图 14-15 NodeLocal DNSCache 查询路径



其中解析线路说明如下：

- ①：已注入DNS本地缓存的Pod，默认会通过NodeLocal DNSCache解析请求域名。

- ②：NodeLocal DNSCache本地缓存如果无法解析请求，则会请求集群CoreDNS进行解析。
- ③：对于非集群内的域名，CoreDNS会通过VPC的DNS服务器进行解析。
- ④：已注入DNS本地缓存的Pod，如果无法连通NodeLocal DNSCache，则会直接通过CoreDNS解析域名。
- ⑤：未注入DNS本地缓存的Pod，默认会通过CoreDNS解析域名。

约束与限制

- 仅支持1.19及以上版本集群。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**节点本地域名解析加速**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据集群规模选择“小规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”为单实例部署，最大支持50节点、1000Pod集群规模；“大规格”为双实例部署，具有高可用能力，最大支持500节点、1000Pod集群规模。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件支持的“参数配置”。

- DNSConfig自动注入：启用后，会创建DNSConfig动态注入控制器，该控制器基于Admission Webhook机制拦截目标命名空间（即命名空间包含标签node-local-dns-injection=enabled）下Pod的创建请求，自动为Pod配置DNSConfig。未开启DNSConfig自动注入或Pod属于非目标命名空间，则需要手动给Pod配置DNSConfig。

  开启自动注入后，您可以为DNSConfig自定义以下配置项（插件版本为1.6.7及以上支持）：

  说明

  如果开启自动注入时Pod中已经配置了DNSConfig，则优先使用Pod中的DNSConfig。

  - 域名解析服务器地址nameserver（可选）：容器解析域名时查询的DNS服务器的IP地址列表。默认会添加NodeLocal DNSCache的地址，以及CoreDNS的地址，允许用户额外追加1个地址，重复的IP地址将被删除。
  - 搜索域search（可选）：定义域名的搜索域列表，当访问的域名不能被DNS解析时，会把该域名与搜索域列表中的域依次进行组合，并重新向DNS发起请求，直到域名被正确解析或者尝试完搜索域列表为止。允许用户额外追加3个搜索域，重复的域名将被删除。
  - ndots（可选）：该参数的含义是当域名的“.”个数小于ndots的值，会先把域名与search搜索域列表进行组合后进行DNS查询，如果均没有被正确解析，再以域名本身去进行DNS查询。当域名的“.”个数大于或者等于ndots的值，会先对域名本身进行DNS查询，如果没有被正确解析，再把域名与search搜索域列表依次进行组合后进行DNS查询。

- 目标命名空间：启用DNSConfig自动注入时支持设置。仅1.3.0及以上版本的插件支持。
  - 全部开启：CCE会为已创建的命名空间添加标签（node-local-dns-injection=enabled），同时会识别命名空间的创建请求并自动添加标签，这些操作的目标不包含系统内置的命名空间（如kube-system）。
  - 手动配置：手动为需要注入DNSConfig的命名空间添加标签（node-local-dns-injection=enabled），操作步骤请参见**管理命名空间标签**。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-83 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。

参数	参数说明
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 完成以上配置后，单击“安装”。

----结束

组件说明

表 14-84 node-local-dns 组件

容器组件	说明	资源类型
node-local-dns-admission-controller	提供自动注入DNSConfig功能。	Deployment
node-local-dns-cache	作为节点上DNS缓存代理提高集群DNS性能。	DaemonSet

使用 NodeLocal DNSCache

默认情况下，应用的请求会通过CoreDNS代理，如果需要使用NodeLocal DNSCache进行DNS缓存代理，您有以下几种方式可以选择，详情请参见**使用NodeLocal DNSCache**。

- 自动注入：创建Pod时自动配置Pod的dnsConfig字段。（kube-system等系统命名空间下的Pod不支持自动注入）
- 手动配置：手动配置Pod的dnsConfig字段，从而使用NodeLocal DNSCache。

卸载插件

卸载插件后将影响已经使用node-local-dns地址进行域名解析的Pod，请谨慎操作。如需卸载，请先清除命名空间上的node-local-dns-injection=enabled标签，然后删除重建带有node-local-dns-webhook.k8s.io/status: injected注解的Pod，待Pod完成重建以后再卸载插件。

步骤1 卸载前检查。

1. 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**node-local-dns**，单击“编辑”。
2. 在“参数配置”中查看是否启用DNS Config自动注入。

   如果已启用DNSConfig自动注入：

a. 在左侧导航栏中选择“命名空间”。
b. 检查哪些命名空间存在node-local-dns-injection=enabled的标签，并删除标签，操作步骤请参见**管理命名空间标签**。
c. 删除上述命名空间中的Pod并重建。

如果未启用DNSConfig自动注入：

a. 使用kubectl连接集群。
b. 检查哪些业务Pod被手动注入DNSConfig。如果在多个命名空间下创建Pod，所有命名空间下的Pod均需要进行检查。

   例如，检查default命名空间下的Pod，您可以执行以下命令：

   ```
   kubectl get pod -n default -o yaml
   ```

c. 手动清除DNSConfig，并重建Pod。

步骤2 卸载node-local-dns插件。

1. 在左侧导航栏中选择“插件中心”，在右侧找到**node-local-dns**，单击“卸载”。
2. 在弹出对话框中单击“确定”。

----结束

相关链接

使用NodeLocal DNSCache提升DNS性能

版本记录

表 14-85 节点本地域名解析加速插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.6.36	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30集群	1.22.20
1.6.8	v1.23 v1.25 v1.27 v1.28 v1.29	优化自定义注入配置体验	1.22.20

插件版本	支持的集群版本	更新特性	社区版本
1.6.7	v1.23 v1.25 v1.27 v1.28 v1.29	新增自定义注入配置	1.22.20
1.6.2	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	适配CCE v1.29集群	1.22.20
1.5.2	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	1.22.20
1.5.1	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题	1.22.20
1.5.0	v1.21 v1.23 v1.25 v1.27 v1.28	• 适配CCE v1.28集群 • 插件多可用区部署模式支持选择均匀分布 • 插件容器基础镜像切HCE20	1.22.20
1.4.0	v1.19 v1.21 v1.23 v1.25 v1.27	修复插件在CCI场景下pod请求耗时长问题	1.22.20
1.2.7	v1.19 v1.21 v1.23 v1.25	支持插件实例AZ反亲和配置	1.21.1

插件版本	支持的集群版本	更新特性	社区版本
1.2.4	v1.19 v1.21 v1.23 v1.25	适配CCE v1.25集群	1.21.1
1.2.2	v1.19 v1.21 v1.23	支持自定义NodeLocal DNSCache规格	1.21.1

14.6 容器存储插件

14.6.1 CCE 容器存储（Everest）

插件简介

CCE容器存储（Everest）是一个云原生容器存储系统，基于CSI（即Container Storage Interface）为Kubernetes v1.15.6及以上版本集群对接云存储服务的能力。

该插件为系统资源插件，Kubernetes 1.15及以上版本的集群在创建时默认安装。

约束与限制

- 插件版本为1.2.0的CCE容器存储插件（Everest）优化了使用OBS存储时的**密钥认证功能**，低于该版本的插件在升级完成后，需要重启集群中使用OBS存储的全部工作负载，否则工作负载使用存储的能力将受影响。
- Huawei Cloud EulerOS 1.1系统的节点支持2.x.x版本（2.1.9及以上）和1.2.x版本（1.2.70及以上）的CCE容器存储插件（Everest），不支持1.3.x版本的插件。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE容器存储（Everest）**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据并发域名解析能力选择“小规格”、“中规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”最大支持50节点、500PVC规模集群；“中规格”最大支持200节点、2000PVC规模集群；“大规格”最大支持1000节点、10000PVC规模集群。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。其中，插件组件的CPU和内存申请值可根据集群节点规模和PVC数量不同进行调整，配置建议请参见**表14-86**。

  非典型场景下，限制值一般估算公式如下：

- everest-csi-controller：
  - CPU限制值：200及以下节点规模设置为250m；1000节点规模设置为350m；2000节点规模设置为500m。
  - 内存限制值 =（200Mi + 节点数 * 1Mi + PVC数 * 0.2Mi）* 1.2
- everest-csi-driver：
  - CPU限制值：200及以下节点规模设置为300m；1000节点规模设置为500m；2000节点规模设置为800m。
  - 内存限制值：200及以下节点规模设置为300Mi；1000节点规模设置为600Mi；2000节点规模设置为900Mi。

表 14-86 典型场景组件限制值建议

配置场景			everest-csi-controller组件		everest-csi-driver组件	
节点数量	PV/PVC数量	插件实例数	CPU（限制值同申请值）	内存（限制值同申请值）	CPU（限制值同申请值）	内存（限制值同申请值）
50	1000	2	250m	600Mi	300m	300Mi
200	1000	2	250m	1Gi	300m	300Mi
1000	1000	2	350m	2Gi	500m	600Mi
1000	5000	2	450m	3Gi	500m	600Mi
2000	5000	2	550m	4Gi	800m	900Mi
2000	10000	2	650m	5Gi	800m	900Mi

步骤3 设置插件支持的“参数配置”。

表 14-87 everest 插件参数配置

配置项	配置方式	参数说明
节点预留给非容器场景的EVS盘槽位（number_of_reserved_disks）	可视化界面配置	插件版本为2.3.11及以上时支持该参数，用于设置节点上预留的挂盘数，预留出部分盘位供用户自定义挂载云硬盘使用。 假设节点可挂载的云硬盘上限为20，并设置该参数值为6，则在调度挂载云硬盘的工作负载时，实际上节点可挂载的云硬盘为20-6=14。预留的6个挂盘数中，除去节点上已挂载的1块系统盘和1块数据盘后，还可以自定义挂载4块云硬盘，可以作为额外的数据盘或者作为裸盘用于创建本地存储池。

配置项	配置方式	参数说明
禁用全局访问密钥挂载对象存储（disable_auto_mount_secret）	可视化界面配置	挂载对象桶/并行文件系统时，是否允许使用默认的AKSK。 ● 是：对象存储PVC未指定自定义挂载密钥时，默认使用用户上传的全局长效密钥挂载对象存储。 ● 否：对象存储PVC未指定自定义挂载密钥时，不使用用户上传的全局长效密钥挂载对象存储，会导致PVC无法挂载。
处理EVS盘挂载任务的并发数（csi_attacher_worker_threads）	可视化界面配置	CCE容器存储插件（Everest）中同时处理挂EVS卷的worker数，默认值为“60”。
处理EVS盘卸载任务的并发数（csi_attacher_detach_worker_threads）	可视化界面配置	CCE容器存储插件（Everest）中同时处理卸载EVS卷的worker数，默认值均为“60”。
分布式挂卷策略（enable_node_attacher）	可视化界面配置	开启时，由每个节点上的everest-csi-driver组件负责attach/detach EVS卷。 不开启时，由everest-csi-controller负责attach/detach EVS卷。
flow_control	扩展参数配置	采用默认的API客户端流控配置，具体配置可在CCE容器存储插件安装好后查看kube-system命名空间下ConfigMap everest-driver-th-config详情。
over_subscription	扩展参数配置	本地存储池（local_storage）的超分比。默认为80，若本地存储池为100GiB，可以超分为180GiB使用，即超分后的总容量=（1+超分比）*实际容量。
enable_local_autoexpander	扩展参数配置	是否开启精简卷本地存储池自动扩容，开启后，会根据本地存储池扩容阈值和扩容步长触发自动扩容。
expansion_threshold	扩展参数配置	精简卷本地存储池使用率扩容阈值，当精简卷本地存储池使用率超过阈值时，触发本地存储池自动扩容。
expansion_step	扩展参数配置	精简卷本地存储池单块EVS盘扩容步长，单位Gi
expansion_max_evs_size	扩展参数配置	精简卷本地存储池单块EVS盘扩容的上限，单位Gi

配置项	配置方式	参数说明
volume_attaching_flow_ctrl	扩展参数配置	CCE容器存储插件（Everest）在1分钟内可以挂载EVS卷的最大数量，此参数的默认值“0”表示everest插件不做挂卷限制，此时挂卷性能由底层存储资源决定。

说明

在扩展参数配置中，您可以自定义设置可视化界面中没有的高级配置。如果扩展参数配置中的设置与可视化界面中的参数设置冲突，将以扩展参数配置为准。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-88 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。

参数	参数说明
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

----结束

组件说明

表 14-89 everest 组件

容器组件	说明	资源类型
everest-csi-controller	此容器负责存储卷的创建、删除、快照、扩容、attach/detach等功能。若集群版本大于等于1.19，且插件版本为1.2.x，everest-csi-controller组件的Pod还会默认带有一个everest-localvolume-manager容器，此容器负责管理节点上的lvm存储池及localpv的创建。	Deployment
everest-csi-driver	此容器负责PV的挂载、卸载、文件系统resize等功能。若插件版本为1.2.x，且集群所在区域支持node-attacher，everest-csi-driver组件的Pod还会带有一个everest-node-attacher的容器，此容器负责分布式attach EVS，该配置项在部分Region开放。	DaemonSet

Prometheus 指标采集

everest-csi-controller通过端口3225暴露Prometheus metrics指标。您可以自建Prometheus采集器识别并通过http://{{everest-csi-controllerPodIP}}:3225/metrics路径获取everest-csi-controller相关指标。

说明

Prometheus指标暴露仅支持Everest插件2.4.4及以上版本。

表 14-90 关键指标说明

指标名称	指标类型	描述	Labels	举例
everest_action_result_total	counter	everest不同功能的调用情况	action：表示不同功能，详情请参见表14-91 result：表示调用成功或失败	everest_action_result_total{action="create_snapshot:disk.csi.everest.io",result="success"} 2
everest_function_duration_seconds_bucket	histogram	everest不同功能在不同执行时间下的次数	function：表示不同功能，详情请参见表14-91	everest_function_duration_seconds_bucket{function="create_snapshot:disk.csi.everest.io",le="10"} 2
everest_function_duration_seconds_sum	histogram	everest不同功能的调用时间总和	function：表示不同功能，详情请参见表14-91	everest_function_duration_seconds_sum{function="create:disk.csi.everest.io"} 24.381399053
everest_function_duration_seconds_count	histogram	everest不同功能的调用次数	function：表示不同功能，详情请参见表14-91	everest_function_duration_seconds_count{function="attach:disk.csi.everest.io"} 4

action及function表示不同CSI驱动及其功能，表示为：{功能}:{CSI驱动}。例如create:disk.csi.everest.io，表示功能为创建卷、卷类型为云硬盘。

表 14-91 功能说明

功能名称	功能描述
create	创建卷
delete	删除卷
attach	卷挂载
detach	卷卸载
expand	卷扩容
create_snapshot	创建卷快照
delete_snapshot	删除卷快照

版本记录

表 14-92 CCE 容器存储插件（Everest）版本记录

插件版本	支持的集群版本	更新特性
2.4.75	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	HCE2.0节点上的云硬盘类型PVC支持指定Fstype类型为xfs
2.4.72	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30版本
2.4.44	v1.23 v1.25 v1.27 v1.28 v1.29	• 支持在SFS3.0文件系统中创建子目录 • 支持获取存储卷的使用情况
2.4.28	v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题
2.4.21	v1.23 v1.25 v1.27 v1.28 v1.29	• 支持OBS多AZ冗余存储策略 • 支持自定义存储卷名称前缀 • 支持包周期EVS卷
2.4.8	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	• 适配CCE v1.29集群 • 支持GPSSD2类型磁盘 • 支持DSS专属分布式存储，同时集群版本需要满足v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上

插件版本	支持的集群版本	更新特性
2.3.23	v1.21 v1.23 v1.25 v1.27 v1.28	支持在SFS Turbo文件系统中创建子目录
2.3.21	v1.21 v1.23 v1.25 v1.27 v1.28	修复部分问题
2.3.14	v1.21 v1.23 v1.25 v1.27 v1.28	适配CCE v1.28版本
2.1.51	v1.19 v1.21 v1.23 v1.25 v1.27	支持Huawei Cloud EulerOS 2.0系统
2.1.30	v1.19 v1.21 v1.23 v1.25	• 支持插件实例AZ反亲和配置 • obsfs包适配Ubuntu 22.04
2.1.13	v1.19 v1.21 v1.23 v1.25	SFS Turbo存储卷subpath PVC批创性能优化
2.1.9	v1.19 v1.21 v1.23 v1.25	• 支持controller优雅退出 • 适配CCE v1.25版本
2.0.9	v1.19 v1.21 v1.23	• 对 everest 部分代码及架构进行重构，改善代码架构，提高插件的可扩展性和稳定性 • 支持优雅退出 • 支持OBS进程监控

插件版本	支持的集群版本	更新特性
1.3.28	v1.19 v1.21 v1.23	● 支持优雅退出 ● 支持OBS进程监控
1.3.22	v1.19 v1.21 v1.23	修复重复挂盘偶现挂载后读写失败的问题
1.3.20	v1.19 v1.21 v1.23	修复重复挂盘偶现挂载后读写失败的问题
1.3.17	v1.19 v1.21 v1.23	● 调整everest-csi-driver滚动更新的最大不可用数：从10更新到10% ● 自定义规格支持Pod反亲和 ● 统计节点上可由csi插件管理的scsi卷个数的上限 ● 支持Driver自定义资源规格部署
1.3.8	v1.23	适配CCE v1.23集群
1.3.6	v1.23	适配CCE v1.23集群
1.2.78	v1.15 v1.17 v1.19 v1.21	支持插件实例AZ反亲和配置
1.2.70	v1.15 v1.17 v1.19 v1.21	SFS Turbo存储卷subpath PVC批创性能优化
1.2.67	v1.15 v1.17 v1.19 v1.21	● 支持controller优雅退出 ● 支持OBS进程监控
1.2.61	v1.15 v1.17 v1.19 v1.21	● 支持优雅退出 ● 支持OBS进程监控

插件版本	支持的集群版本	更新特性
1.2.55	v1.15 v1.17 v1.19 v1.21	修复重复挂盘偶现挂载后读写失败的问题
1.2.53	v1.15 v1.17 v1.19 v1.21	修复重复挂盘偶现挂载后读写失败的问题
1.2.51	v1.15 v1.17 v1.19 v1.21	• 调整everest-csi-driver滚动更新的最大不可用数：从10更新到10% • 自定义规格支持Pod反亲和 • 统计节点上可由csi插件管理的scsi卷个数的上限
1.2.44	v1.15 v1.17 v1.19 v1.21	• EVS、OBS存储卷支持选择企业项目 • OBS对象桶挂载默认不再使用enable_noobj_cache参数
1.2.42	v1.15 v1.17 v1.19 v1.21	• EVS、OBS存储卷支持选择企业项目 • OBS对象桶挂载默认不再使用enable_noobj_cache参数
1.2.30	v1.15 v1.17 v1.19 v1.21	CCE支持EmptyDir
1.2.28	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群
1.2.27	v1.15 v1.17 v1.19 v1.21	支持极速型SSD(ESSD)、通用型SSD(GPSSD)类型云硬盘
1.2.13	v1.15 v1.17 v1.19	支持EulerOS 2.10系统。

插件版本	支持的集群版本	更新特性
1.2.9	v1.15 v1.17 v1.19	• 增强PV资源生命周期维护的可靠性 • 支持1.19.10版本集群使用Attach/Detach Controller挂卸卷能力 • 提高SFS挂载稳定性 • 新建集群EVS默认创建类型调整为SAS
1.2.5	v1.15 v1.17 v1.19	• 提升挂载相关能力可靠性 • 优化了使用OBS存储时的认证功能，需要用户上传密钥 • 提高everest插件对flexvolume卷的兼容能力 • 提高插件运行稳定性
1.1.12	v1.15 v1.17	优化和增强everest-csi-controller组件可靠性
1.1.11	v1.15 v1.17	• 配置安全加固 • 支持挂载三方OBS存储 • 切换更优性能的EVS查询接口 • 默认快照以clone模式创建磁盘 • 优化和增强Attach和Detach磁盘状态检测和日志输出 • 增加认证过期判断可靠性
1.1.8	v1.15 v1.17	支持CCE v1.17，v1.13升级到v1.15场景支持接管Flexvolume
1.1.7	v1.15 v1.17	支持CCE v1.17，v1.13升级到v1.15场景支持接管Flexvolume

14.6.2 CCE 容器存储（Flexvolume，已废弃）

插件简介

CCE容器存储（FlexVolume），即storage-driver，是一款云存储驱动插件，北向遵循标准容器平台存储驱动接口。实现Kubernetes Flex Volume标准接口，提供容器使用EVS块存储、SFS文件存储、OBS 对象存储、SFS Turbo 极速文件存储的能力。通过安装升级云存储插件可以实现云存储功能的快速安装和更新升级。

该插件为系统资源插件，Kubernetes 1.13及以下版本的集群在创建时默认安装。

约束与限制

- 在CCE所创的集群中，Kubernetes v1.15.11版本是Flexvolume插件被CSI插件（**everest**）兼容接管的过渡版本，v1.17及之后版本的集群中将彻底去除对Flexvolume插件的支持，请您将Flexvolume插件的使用切换到CSI插件上。

- CSI插件（everest）兼容接管Flexvolume插件后，原有功能保持不变，但请注意不要新建Flexvolume插件（storage-driver）的存储，否则将导致部分存储功能异常。
- 本插件仅支持在**v1.13及以下版本**的集群中安装，v1.15及以上版本的集群在创建时默认安装everest插件。

  说明

  在**v1.13及以下版本**的集群中，当存储功能有升级或者BUG修复时，用户无需升级集群或新建集群来升级存储功能，仅需安装或升级storage-driver插件。

安装插件

本插件为系统默认安装，若因特殊情况卸载后，可参照如下步骤重新安装。

未安装storage-driver插件的集群，可参考如下步骤进行安装：

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“插件中心”，在右侧找到**CCE容器存储（FlexVolume）**，单击“安装”。

步骤2 云存储插件暂未开放可配置参数，直接单击“安装”。

----结束

14.7 容器安全插件

14.7.1 CCE 密钥管理（对接 DEW）

插件简介

CCE密钥管理（dew-provider）插件用于对接**数据加密服务**(Data Encryption Workshop, DEW)。该插件允许用户将存储在集群外部（即专门存储敏感信息的数据加密服务）的凭据挂载至业务Pod内，从而将敏感信息与集群环境解耦，有效避免程序硬编码或明文配置等问题导致的敏感信息泄密。

约束与限制

- 数据加密服务包含密钥管理(Key Management Service, KMS)、云凭据管理(Cloud Secret Management Service，CSMS)和密钥对管理(Key Pair Service, KPS)等服务。当前，该插件仅支持对接其中的云凭据管理服务。
- 允许创建的SecretProviderClass对象个数上限：500个。
- 插件卸载时，会同时删除相关的CRD资源。即使重装插件，原有的SecretProviderClass对象也不可用，请谨慎操作。插件卸载再重装后，若需使用原有的SecretProviderClass资源，需重新手动创建。

插件说明

- 基础挂载能力：安装完该插件后，通过创建SecretProviderClass对象，在业务Pod中声明Volume并进行引用，当启动Pod时，就会将在SecretProviderClass对象中声明的凭据信息挂载至Pod内。

- 定时轮转能力：当Pod正常运行后，若其在SPC中声明的、存储在云凭据管理服务中的凭据发生了更新，通过定时轮转，可以将最新的凭据值刷新至Pod内。使用该能力时，需要将凭据的版本指定为”latest”。
- 实时感知SPC变化能力：当Pod正常运行后，若用户修改了在SPC中声明的凭据信息（如新增凭据、改变原有凭据的版本号等），插件可实时感知该变化，并将更新后的凭据刷新至Pod内。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**CCE密钥管理（对接DEW）**插件，单击“安装”。

步骤2 在安装插件页面，在参数配置栏进行参数配置。参数配置说明如下。

配置项	参数	参数说明
凭据同步周期	rotation_poll_interval	轮转时间间隔。单位：分钟，即m（注意不是min）。 轮转时间间隔表示向云凭据管理服务发起请求并获取最新的凭据的周期，合理的时间间隔范围为[1m, 1440m]，默认值为2m。

步骤3 单击“安装”。

待插件安装完成后，选择对应的集群，然后单击左侧导航栏的“插件中心”，可在“已安装插件”页签中查看相应的插件。

----结束

组件说明

表 14-93 dew-provider 组件

容器组件	说明	资源类型
dew-provider	dew-provider负责与云凭据管理服务交互，从云凭据管理服务中获取指定的凭据，并挂载到业务Pod内。	DaemonSet
csi-secrets-store	csi-secrets-store负责维护两个CRD资源，即SecretProviderClass（以下简称为SPC）和SecretProviderClassPodStatus（以下简称为spcPodStatus），其中**SPC用于描述用户感兴趣的凭据信息（比如指定凭据的版本、凭据的名称等），由用户创建，并在业务Pod中进行引用**；**spcPodStatus用于跟踪Pod与凭据的绑定关系，由csi-driver自动创建，用户无需关心**。一个Pod对应一个spcPodStatus，当Pod正常启动后，会生成一个与之对应的spcPodStatus；当Pod生命周期结束时，相应的spcPodStatus也会被删除。	DaemonSet

使用 Volume 挂载凭据

步骤1 创建ServiceAccount。

1. 创建ServiceAccount对象，**其中声明了允许业务使用的凭据名称，若用户引用了未在此处声明的凭据，则挂载失败，最终导致Pod无法运行。**

   根据如下模板创建serviceaccount.yaml，在**cce.io/dew-resource**字段中声明允许业务使用的凭据名称。这里声明了secret_1和secret_2，表示允许业务引用这两个凭据对象。在后续的操作中，若用户在业务中引用了secret_3，则无法通过校验，从而导致无法正常挂载该凭据，最终业务Pod将无法运行。

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: nginx-spc-sa
     annotations:
       cce.io/dew-resource: "[\"secret_1\",\"secret_2\"]"  #secrets that allow pod to use
   ```

   这里需要明确，此处声明的凭据应确保在凭据管理服务中是存在的，如下图所示。否则，即使通过了校验，最终向凭据管理服务中获取相应凭据的时候也会出错，从而导致Pod无法正常运行。

2. 执行如下命令创建ServiceAccount对象。

   kubectl apply -f serviceaccount.yaml

3. 查看ServiceAccount对象是否已经正常创建，如下所示：

   ```
   $ kubectl get sa
   NAME       SECRETS   AGE
   default        1        18d   # 此为系统默认的ServiceAccount对象
   nginx-spc-sa   1        19s   # 此为刚刚创建的ServiceAccount对象
   ```

   至此，一个名为“nginx-spc-sa”的ServiceAccount对象已正常创建。该对象将在后续的业务Pod中被引用。

步骤2 创建SecretProviderClass。

1. SecretProviderClass对象**用于描述用户感兴趣的凭据信息（比如指定凭据的版本、凭据的名称等），由用户创建，并在业务Pod中进行引用。**

   根据如下模板创建secretproviderclass.yaml。**用户主要关注parameters.objects字段，它是一个数组，用于声明用户想要挂载的凭据信息。**

   ```
   apiVersion: secrets-store.csi.x-k8s.io/v1
   kind: SecretProviderClass
   metadata:
     name: spc-test
   spec:
     provider: cce     # 固定为cce
     parameters:
       objects: |
           - objectName: "secret_1"
             objectVersion: "v1"
             objectType: "csms"
   ```


参数	参数类型	是否必选	参数说明
objectName	String	是	凭据名称，需填写ServiceAccount中引用的凭据。若同一个SecretProviderClass中定义了多个objectName，不允许重名，否则会挂载失败。
objectAlias	String	否	凭据写入到容器内的文件名称。若不指定，则凭据写入到容器内的文件名默认为objectName；若指定，则objectAlias与其他凭据的objectName和objectAlias均不允许重名，与自身的objectName也不允许重名，否则会挂载失败。
objectType	String	是	凭据类型。当前仅支持”csms”类型，其他均为非法输入。
objectVersion	String	是	凭据的版本。 – 指定某个具体的版本：v1,v2,… – 指定最新版本：latest。当指定objectVersion为” latest”时，若在云凭据管理服务侧对应的凭据发生了更新，更新后的凭据值将在经过一定时间间隔后（即rotation_poll_interval）刷新至Pod内。

2. 执行如下命令创建SecretProviderClass对象。

   kubectl apply -f secretproviderclass.yaml

3. 查看SecretProviderClass对象是否已经正常创建，如下所示：

   ```
   $ kubectl get spc
   NAME   AGE
   spc-test   20h
   ```

   至此，一个名为“spc-test”的SecretProviderClass对象已正常创建。该对象将在后续的业务Pod中被引用。

步骤3 创建业务Pod。

这里以创建一个nginx应用为例。

1. 定义业务负载，在serviceAccountName中引用此前创建好的ServiceAccount对象，secretProviderClass中引用此前创建好的SPC对象，并在mountPath中指定容器内的挂载路径（这里需注意，用户不应该指定”/”，” /var/run”等特殊目录，否则可能影响容器的正常启动）。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: nginx-spc
     labels:
       app: nginx
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: nginx
     template:
   ```



```
  metadata:
    labels:
      app: nginx
  spec:
    serviceAccountName: nginx-spc-sa   # 引用上面创建的ServiceAccount
    volumes:
      - name: secrets-store-inline
        csi:
          driver: secrets-store.csi.k8s.io
          readOnly: true
          volumeAttributes:
            secretProviderClass: "spc-test"  # 引用上面创建的SPC
    containers:
      - name: nginx-spc
        image: nginx:alpine
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - name: secrets-store-inline
            mountPath: "/mnt/secrets-store"  # 定义容器内凭据的挂载路径
            readOnly: true
    imagePullSecrets:
      - name: default-secret
```

2. 创建业务Pod。

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

3. 查看Pod是否已经正常创建，如下所示：

```
$ kubectl get pod
NAME                      READY   STATUS    RESTARTS   AGE
nginx-spc-67c9d5b594-642np     1/1     Running    0          20s
```

4. 进入容器，查看指定的凭据是否正常写入。如下所示：

```
$ kubectl exec -ti nginx-spc-67c9d5b594-642np -- /bin/bash
root@nginx-spc-67c9d5b594-642np:/#
root@nginx-spc-67c9d5b594-642np:/# cd /mnt/secrets-store/
root@nginx-spc-67c9d5b594-642np:/mnt/secrets-store#
root@nginx-spc-67c9d5b594-642np:/mnt/secrets-store# ls
secret_1
```

可以看到，用户在SPC对象中声明的secret_1已正常写入Pod。

此外，还可以通过获取spcPodStatus查看Pod与凭据的绑定情况。如下所示：

```
$ kubectl get spcps
NAME                                                AGE
nginx-spc-67c9d5b594-642np-default-spc-test   103s
$ kubectl get spcps nginx-spc-67c9d5b594-642np-default-spc-test -o yaml
......
status:
mounted: true
objects:    # 挂载的凭据信息
- id: secret_1
version: v1
podName: nginx-spc-67c9d5b594-642np   # 引用了SPC对象的Pod
secretProviderClassName: spc-test              # SPC对象
targetPath: /mnt/paas/kubernetes/kubelet/pods/6dd29596-5b78-44fb-9d4c-a5027c420617/volumes/
kubernetes.io~csi/secrets-store-inline/mount
```

----结束

使用密钥挂载凭据

说明

CCE密钥管理（dew-provider）插件版本要求1.1.1及以上。

步骤1 创建ServiceAccount。

1. 创建ServiceAccount对象，**其中声明了允许业务使用的凭据名称，若用户引用了未在此处声明的凭据，则挂载失败，最终导致Pod无法运行。**

根据如下模板创建serviceaccount.yaml，在**cce.io/dew-resource**字段中声明允许业务使用的凭据名称。这里声明了secret_1和secret_2，表示允许业务引用这两个凭据对象。在后续的操作中，若用户在业务中引用了secret_3，则无法通过校验，从而导致无法正常挂载该凭据，最终业务Pod将无法运行。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-spc-sa
  annotations:
    cce.io/dew-resource: "[\"secret_1\",\"secret_2\"]"  #secrets that allow pod to use
```

这里需要明确，此处声明的凭据应确保在凭据管理服务中是存在的，如下图所示。否则，即使通过了校验，最终向凭据管理服务中获取相应凭据的时候也会出错，从而导致Pod无法正常运行。

2. 执行如下命令创建ServiceAccount对象。

   kubectl apply -f serviceaccount.yaml

3. 查看ServiceAccount对象是否已经正常创建，如下所示：

   ```
   $ kubectl get sa
   NAME       SECRETS   AGE
   default        1        18d   # 此为系统默认的ServiceAccount对象
   nginx-spc-sa   1        19s   # 此为刚刚创建的ServiceAccount对象
   ```

   至此，一个名为“nginx-spc-sa”的ServiceAccount对象已正常创建。该对象将在后续的业务Pod中被引用。

步骤2 创建SecretProviderClass。

1. 根据如下模板创建secretproviderclass.yaml。

   ```
   apiVersion: secrets-store.csi.x-k8s.io/v1
   kind: SecretProviderClass
   metadata:
     name: nginx-deployment-spc-k8s-secrets
   spec:
     provider: cce
     parameters:
       # 引用凭据管理服务中的凭据
       objects: |
         - objectName: "secret_1"
           objectType: "csms"
           objectVersion: "latest"
           jmesPath:
             - path: username
               objectAlias: dbusername
             - path: password
               objectAlias: dbpassword
     # 根据凭据中的内容创建密钥，然后在Pod中进行挂载使用
     secretObjects:
       - secretName: my-secret-01
         type: Opaque
         data:
           - objectName: dbusername
   ```



```
      key: db_username_01
    - objectName: dbpassword
      key: db_password_01
```

表 14-94 objects 参数说明

参数	参数类型	是否必选	参数说明
objectName	String	是	凭据名称，需填写ServiceAccount中引用的凭据。若同一个SecretProviderClass中定义了多个objectName，不允许重名，否则会挂载失败。
objectType	String	是	凭据类型。当前仅支持“csms”类型，其他均为非法输入。
objectVersion	String	是	凭据的版本。 – 指定某个具体的版本：v1,v2,… – 指定最新版本：latest。当指定objectVersion为” latest”时，若在云凭据管理服务侧对应的凭据发生了更新，更新后的凭据值将在经过一定时间间隔后（即rotation_poll_interval）刷新至Pod内。
jmesPath	Array of Object	是	**jmesPath**是一种从json格式的对象中提取key-value的工具，CCE密钥管理插件使用该工具支持Secret挂载功能。 – path：填写DEW服务凭据中的key，其中key不能带有+、-、{}、()等特殊符号。 – objectAlias：挂载到Pod中的文件名，该值需要和secretObjects中定义的**objectName**保持一致。

表 14-95 secretObjects 参数说明

参数	参数类型	是否必选	参数说明
secretName	String	是	密钥名称。
type	String	是	密钥类型。
data	Array of Object	是	– objectName：挂载到Pod中的文件名，该值需要和objects中定义的**objectAlias**保持一致。 – key：密钥中的key，在Pod中可使用key值对加密内容进行引用。

2. 执行如下命令创建SecretProviderClass对象。

   kubectl apply -f secretproviderclass.yaml

3. 查看SecretProviderClass对象是否已经正常创建，如下所示：

   ```
   $ kubectl get spc
   NAME   AGE
   nginx-deployment-spc-k8s-secrets   20h
   ```

步骤3 创建业务Pod。这里以创建一个nginx应用为例。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-k8s-secrets
  labels:
    app: nginx-k8s-secrets
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-k8s-secrets
  template:
    metadata:
      labels:
        app: nginx-k8s-secrets
    spec:
      serviceAccountName: nginx-spc-sa   # 引用上面创建的ServiceAccount
      containers:
      - name: nginx-deployment-k8s-secrets
        image: nginx
        volumeMounts:   # 在容器中挂载密钥
          - name: secrets-store-inline   # 需要挂载的volume名称
            mountPath: "/mnt/secrets"    # 需要挂载的容器路径
            readOnly: true
        env:    # 在环境变量中引用密钥
          - name: DB_USERNAME_01      # 工作负载中的变量名
            valueFrom:
              secretKeyRef:
                name: my-secret-01    # SPC中定义的密钥名称
                key: db_username_01   # SPC中定义的密钥key值
          - name: DB_PASSWORD_01
            valueFrom:
              secretKeyRef:
                name: my-secret-01
                key: db_password_01
      imagePullSecrets:
        - name: default-secret
      volumes:    # 使用SPC中定义的密钥创建volume
        - name: secrets-store-inline  # 自定义的volume名称
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
            volumeAttributes:
              secretProviderClass: nginx-deployment-spc-k8s-secrets   # 上一步中创建的SPC名称
```

步骤4 验证结果。

```
$ kubectl get secrets
NAME            TYPE                                  DATA   AGE
default-secret   kubernetes.io/dockerconfigjson   1      33d
my-secret-01     Opaque                                2      1h
```

结果表明已使用SPC对象中声明的凭据secret_1创建一个密钥my-secret-01。

----结束

定时轮转

在**插件使用说明**，通过使用该插件，用户可完成基本的凭据挂载功能，即能够将存储在凭据管理服务中的凭据写入到Pod内。

若将在SPC对象中声明的凭据版本改为”latest”，如下所示：

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: spc-test
spec:
  provider: cce
  parameters:
    objects: |
          - objectName: "secret_1"
            objectVersion: "latest"  # change "v1"to "latest"
            objectType: "csms"
```

更新该SPC对象后，插件将周期性地向凭据管理服务发起请求，获取凭据secret_1最新版本的值，并将其刷新至引用了该SPC对象的Pod内。此处插件周期性发起请求的时间间隔由**安装插件**时设置的rotation_poll_interval参数确定。

实时感知 SPC 变化

在**使用Volume挂载凭据**、**定时轮转**的演示中，其实已经使用到了实时感知SPC变化的能力。为了演示说明，在SPC对象中新增一个凭据secret_2，如下所示：

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: spc-test
spec:
  provider: cce
  parameters:
    objects: |
          - objectName: "secret_1"
            objectVersion: "latest"
            objectType: "csms"
          - objectName: "secret_2"
            objectVersion: "v1"
            objectType: "csms"
```

更新该SPC对象后，新增的secret_2将很快挂载至引用了该SPC对象的Pod内。

查看组件日志

查看插件的Pod

```
$ kubectl get pod -n kube-system
NAME                        READY   STATUS      RESTARTS   AGE
csi-secrets-store-76tj2       3/3     Running     0          11h
dew-provider-hm5fq            1/1     Running     0          11h
```

查看dew-provider组件Pod日志

```
$ kubectl logs dew-provider-hm5fq -n kube-system
…日志信息略…
…
```

查看csi-secrets-store组件Pod日志，由于csi-secrets-store组件的Pod包含多个容器，在查看日志信息时，**需通过”-c”命令指定某个容器**。其中，secrets-store容器作为该插件的主业务容器，其包含了主要的日志信息。


```
$ kubectl logs csi-secrets-store-76tj2 -c secrets-store -n kube-system
…日志信息略…
…
```

版本记录

表 14-96 CCE 密钥管理（对接 DEW）插件版本记录

插件版本	支持的集群版本	更新特性
1.1.33	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	适配CCE v1.30集群
1.1.3	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	修复部分问题
1.1.2	v1.21 v1.23 v1.25 v1.27 v1.28 v1.29	• 适配CCE v1.29集群 • 支持同步csms凭据时创建secret
1.0.31	v1.21 v1.23 v1.25 v1.27 v1.28	• 适配CCE v1.27集群 • 适配CCE v1.28集群
1.0.9	v1.19 v1.21 v1.23 v1.25	-

插件版本	支持的集群版本	更新特性
1.0.6	v1.19 v1.21 v1.23 v1.25	-
1.0.3	v1.19 v1.21 v1.23 v1.25	适配CCE v1.25集群
1.0.2	v1.19 v1.21 v1.23	适配CCE v1.23集群
1.0.1	v1.19 v1.21	支持主动感知SecretProviderClass对象的变化

14.7.2 容器镜像签名验证

插件简介

容器镜像签名验证插件（swr-cosign）提供镜像验签功能，可以对镜像文件进行数字签名验证，以确保镜像文件的完整性和真实性，有效地防止软件被篡改或植入恶意代码，保障用户的安全。

约束与限制

- 使用镜像验签功能依赖容器镜像仓库企业版，请先创建一个企业版仓库。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**容器镜像签名验证**，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据集群规模选择“小规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”为单实例部署，适用于并发下载不同镜像数量小于50的集群；“大规格”为双实例部署，具有高可用能力，适用于并发下载不同镜像数量小于300的集群。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 设置插件支持的“参数配置”。

表 14-97 swr-cosign 插件参数配置

参数	参数说明
KMS密钥	选择一个密钥，仅支持 EC_P256、EC_P384、SM2 类型的密钥。 您可以前往密钥管理服务新增密钥。
验签镜像	验签镜像地址通过正则表达式进行匹配，例如填写docker.io/**表示对docker.io镜像仓库的镜像进行验签。如需对所有镜像验签，请填写**。

步骤4 单击“安装”。

待插件安装完成后，选择对应的集群，然后单击左侧导航栏的“插件中心”，可筛选“已安装插件”查看相应的插件。

----结束

组件说明

表 14-98 swr-cosign 组件

容器组件	说明	资源类型
swr-cosign	swr-cosign负责对镜像文件进行数字签名验证，以保证镜像文件未被篡改。	Deployment

使用说明

步骤1 参考**安装插件**中的步骤，在集群中安装插件，并设置KMS密钥和验签镜像地址。

步骤2 为需要标签的命名空间加上**policy.sigstore.dev/include:true**标签。

1. 在集群控制台左侧导航栏单击“命名空间”。
2. 找到需要验签的命名空间，单击操作列的“更多>标签管理”。
3. 新增一个标签，键值填写如下：
   - 键：policy.sigstore.dev/include
   - 值：true
4. 单击“确定”。

步骤3 测试镜像验签功能是否生效。

1. 在集群控制台左侧导航栏单击“工作负载”。
2. 单击右上角“创建工作负载”。
3. 选择已添加标签的命名空间，并填写未经签名的镜像地址，其余参数请参考**创建无状态负载（Deployment）**按需填写。
4. 单击“创建工作负载”。

   未经签名的镜像将被拦截，提示如下：

   ```
   admission webhook "policy.sigstore.dev" denied the request: validation failed: failed policy: cip-key-secret-match: spec.template.spec.containers[0].image ...
   ```


步骤4 为镜像添加签名。

1. 登录SWR企业仓库，进入一个已有的仓库实例。
2. 在左侧导航栏中单击“安全可信>镜像签名”，创建一个签名规则。
   - 规则名称：为签名规则命名。
   - 组织：选择一个容器镜像组织。
   - 规则范围：
     - 镜像：选择需要签名的镜像。您也可以通过正则表达式匹配多个镜像。
     - 版本：选择镜像版本，若不填或填写**则表示匹配该镜像的所有版本。
   - 签名方式：选择KMS方式。
   - 签名Key：选择一个KMS密钥，该密钥需要与安装插件时的密钥相同。
   - 触发模式：
     - 手动：创建完成签名规则后，需要手动执行规则来对镜像进行签名。
     - 事件触发+手动：可通过事件触发签名动作，也可以手动执行规则对镜像进行签名。
   - 规则描述：填写规则的描述信息。
3. 签名规则创建完成后，单击操作列的“执行”，对所选镜像进行签名。
4. 签名成功后，在左侧导航栏中单击“制品仓库>镜像仓库”，单击该镜像名称查看镜像详情。该镜像已存在签名附件。

步骤5 重新回到CCE控制台，使用该镜像创建工作负载，创建测试镜像是否可以成功创建。

----结束

版本记录

表 14-99 swr-cosign 插件版本记录

插件版本	支持的集群版本	更新特性
1.0.2	v1.23 v1.25 v1.27	支持v1.27集群
1.0.1	v1.23 v1.25	支持镜像验签

14.8 其他插件

14.8.1 Kubernetes Dashboard

插件简介

Kubernetes Dashboard是一个旨在为Kubernetes世界带来通用监控和操作Web界面的项目，集合了命令行可以操作的所有命令。

使用Kubernetes Dashboard，您可以：

- 向Kubernetes集群部署容器化应用
- 诊断容器化应用的问题
- 管理集群的资源
- 查看集群上所运行的应用程序
- 创建、修改Kubernetes上的资源（例如Deployment、Job、DaemonSet等）
- 展示集群上发生的错误

例如：您可以伸缩一个Deployment、执行滚动更新、重启一个Pod或部署一个新的应用程序。

开源社区地址：https://github.com/kubernetes/dashboard

安装步骤

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**Kubernetes Dashboard**插件，单击“安装”。

步骤2 （3.0.2及以上版本支持）在安装插件页面，设置“规格配置”。

表 14-100 插件规格配置

参数	参数说明
插件规格	该插件可配置系统预置规格或自定义规格。
容器	选择自定义规格时，您可根据需求调整插件实例的容器规格。

步骤3 在参数配置页面，配置以下参数。

- 访问方式：支持“节点访问”，通过集群节点绑定的弹性公网IP进行访问，当集群节点未绑定弹性IP时无法正常使用。
- 证书配置：dashboard服务端使用的证书。
  - 使用自定义证书
    您需要参考样例填写pem格式的“证书文件”和“证书私钥”。
  - 使用默认证书

须知

dashboard默认生成的证书不合法，将影响浏览器正常访问，建议您选择手动上传合法证书，以便通过浏览器校验，保证连接的安全性。

步骤4 单击“安装”。

----结束

访问 dashboard

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，确认dashboard插件状态为“运行中”后，单击“访问”。

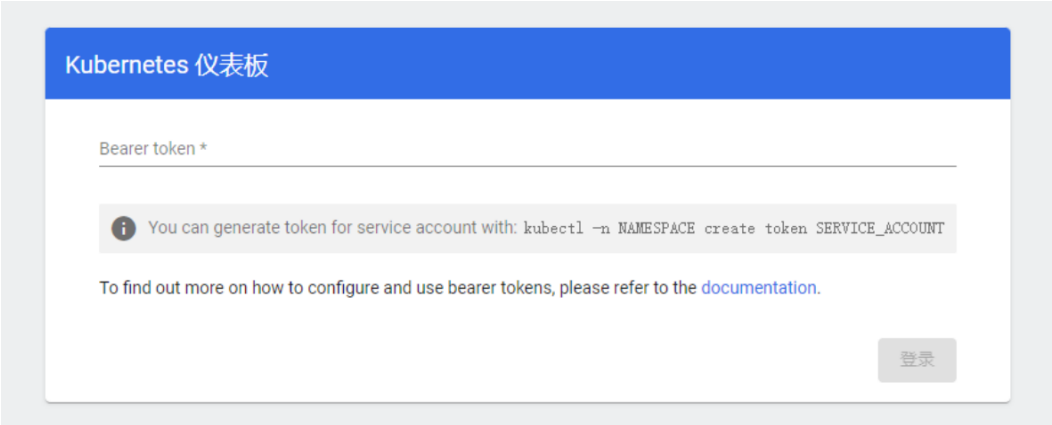
步骤2 在CCE控制台弹出的窗口中复制token。

步骤3 在登录页面中选择“令牌”的登录方式，粘贴输入复制的token，单击“登录”按钮。

说明

本插件默认不支持使用证书认证的kubeconfig进行登录，推荐使用令牌方式登录。详细信息请参考：https://github.com/kubernetes/dashboard/issues/2474#issuecomment-348912376

图 14-16 令牌方式登录



步骤4 登录后效果，如图14-17。

图 14-17 Dashboard 概览页

----结束

权限修改

安装Dashboard插件后初始角色仅拥有对大部分资源的只读权限，若想让Dashboard界面支持更多操作，需自行在后台对RBAC相关资源进行修改。

具体修改方式：

可对名为“kubernetes-dashboard-minimal”这个ClusterRole中的规则进行调整。

关于使用RBAC的具体细节可参考文档：**https://kubernetes.io/docs/reference/access-authn-authz/rbac/**。

组件说明

表 14-101 dashboard 组件（ 3.0.2 以下版本 ）

容器组件	说明	资源类型
dashboard	该容器提供Kubernetes可视化监控界面。	Deployment

表 14-102 dashboard 组件（ 3.0.2 及以上版本 ）

容器组件	说明	资源类型
dashboard-api	Dashboard的后端API，它提供了一个RESTful API，用于与Kubernetes API Server进行通信。API组件负责从Kubernetes API Server中获取集群信息，并将这些信息提供给Web组件。	Deployment
dashboard-auth	Dashboard的身份验证模块，它提供了基于Token的身份验证机制。当用户登录Dashboard时，Auth组件会检查用户提供的Token是否有效，如果Token有效，则允许用户访问Dashboard。	Deployment
dashboard-web	Dashboard的前端界面，它提供了一个用户友好的UI，用于查看和管理Kubernetes集群。Web组件是一个基于React框架的单页应用程序，可以通过HTTPS协议访问。	Deployment
metrics-scraper	Dashboard的指标采集模块，它负责从Kubernetes集群中收集指标数据，并将这些数据提供给Web组件。Metrics-scraper组件使用Heapster或Metrics Server来获取指标数据，并将这些数据存储在Kubernetes API Server中。	Deployment
dashboard-kong	Dashboard依赖的开源API网关组件，帮助管理API，实现认证和授权功能。	Deployment

附：访问报错解决方法

使用Chrome浏览器访问时，会出现如下“ERR_CERT_INVALID”的报错导致无法正常进入登录界面，原因是dashboard默认生成的证书未通过Chrome校验，当前有以下两种解决方式：

图 14-18 Chrome 浏览器报错信息

- 方式一：使用火狐浏览器访问链接，为当前地址添加“例外”后即可进入登录页面。
- 方式二：通过启动Chrome时添加“--ignore-certificate-errors”开关忽略证书报错。

  Windows：保存链接地址，关闭所有已经打开的Chrome浏览器窗口，Windows键 +“R”弹出“运行”对话框，输入“chrome --ignore-certificate-errors”启动新的chrome窗口，输入地址进入登录界面。

版本记录

表 14-103 Kubernetes Dashboard 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
3.0.4	v1.27 v1.28 v1.29	修复部分问题	7.3.2
3.0.2	v1.27 v1.28 v1.29	• 支持v1.27、v1.28、v1.29集群 • 更新至社区7.3.2版本	7.3.2
2.2.27	v1.21 v1.23 v1.25	修复部分问题	2.7.0

插件版本	支持的集群版本	更新特性	社区版本
2.2.7	v1.21 v1.23 v1.25	-	2.7.0
2.2.5	v1.21 v1.23 v1.25	插件与节点时区一致	2.7.0
2.2.3	v1.21 v1.23 v1.25	-	2.7.0
2.1.1	v1.19 v1.21 v1.23	• 适配CCE v1.23集群 • 更新至社区v2.5.0版本	2.5.0
2.0.10	v1.15 v1.17 v1.19 v1.21	适配CCE v1.21集群	2.0.0
2.0.4	v1.15 v1.17 v1.19	配置seccomp默认规则	2.0.0
2.0.3	v1.15 v1.17 v1.19	兼容CCE v1.15集群	2.0.0
2.0.2	v1.17 v1.19	适配CCE v1.19集群	2.0.0
2.0.1	v1.15 v1.17	-	2.0.0
2.0.0	v1.17	支持对接CCE v1.17	2.0.0

14.8.2 OpenKruise

插件简介

OpenKruise是一个基于Kubernetes的扩展套件，使用CRD拓展来提供扩展工作负载和应用管理能力，实现云原生应用的自动化部署、发布、运维和可用性防护，使得应用的管理更加简单和高效。

OpenKruise的核心能力如下：

- 高级工作负载：OpenKruise包含一系列增强版本的工作负载，例如CloneSet、Advanced StatefulSet等工作负载。
- 应用sidecar管理：OpenKruise提供了多种SidecarSet，简化了sidecar的注入，并提供了sidecar原地升级的能力。
- 应用安全防护：OpenKruise可以保护您的Kubernetes资源不受级联删除机制的干扰。
- 高效应用运维能力：OpenKruise提供了很多高级的运维能力来帮助您更好地管理应用，例如使用ImagePullJob在任意范围的节点上预先拉取某些镜像，或者原地重启Pod中的容器等。

开源社区地址：**https://github.com/openkruise/kruise**

约束与限制

如果您已经在集群中部署了社区的OpenKruise，请您先将其卸载后再安装CCE提供的OpenKruise插件，否则可能会出现插件安装失败的情况。

注意事项

OpenKruise开源组件中，引入了Webhook的配置，社区将其默认配置的Pod相关失效策略为Fail，这意味着一旦kruise-controller-manager处于不可用状态，将会造成Pod的创建/删除等操作都将会被拦截。因此在使用该插件前，请慎重评估使用该插件所引入的风险，并且尽量将kruise-controller-manager组件配置为多实例的高可用状态，以确保kruise-controller-manager中的Webhook Server能够正常处理请求。

须知

OpenKurise是CCE基于开源软件进行适配并集成的精选开源插件，CCE将提供全面的技术支持服务。然而，CCE不承担因开源软件缺陷导致的业务损失责任，也不承担赔偿或额外的服务，强烈建议用户定期升级软件以修复潜在问题。

安装步骤

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**OpenKruise**插件，单击“安装”。

步骤2 在安装插件页面，根据需求选择“规格配置”。

- 选择“系统预置规格”时，您可根据集群规模选择“小规格”或“大规格”，系统会根据不同的预置规格配置插件的实例数及资源配额，具体配置值请以控制台显示为准。

  “小规格”为单实例部署，适用50节点以下集群规模；“大规格”为高可用部署，适用50节点以上集群规模。
- 选择“自定义规格”时，您可根据需求调整插件实例数和资源配额。实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。

步骤3 选择是否开启“kruise-daemon配置”。

kruise-daemon是OpenKruise新增的DaemonSet组件，可为您提供镜像预热、容器重启功能。

须知

在v1.25及以上版本的集群中安装1.0.3版本的OpenKruise插件时，kruise-daemon无法在使用docker容器引擎的节点上运行，请使用containerd容器引擎。详细原因请参见**组件说明**。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-104 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

----结束

组件说明

表 14-105 OpenKruise 组件

容器组件	说明	资源类型
kruise-controller-manager	OpenKruise的controller中心组件，同时包含了针对Kruise CRD以及Pod资源的admission webhook。kruise-controller-manager会创建webhook configurations来配置哪些资源需要感知处理，并为kube-apiserver提供可调用的Service。	Deployment
kruise-daemon	通过DaemonSet部署到每个节点上，提供镜像预热、容器重启等功能。	DaemonSet

须知

Kubernetes社区在1.24版本移除了对dockershim的支持。CCE为兼顾用户使用docker运行时的习惯，在CCE的v1.25及以上的集群版本引入了cri-dockerd用于替换原来的dockershim，但是OpenKruise社区当前并未实现对cri-dockerd的支持（参见issue）。该问题将在后续版本中解决。

因此，在v1.25及以上版本的集群中安装1.0.3版本的OpenKruise插件时，kruise-daemon无法在使用docker容器引擎的节点上运行，请使用containerd容器引擎。

常见问题

创建工作负载时，事件中出现以下报错：

```
Error creating: Internal error occurred: failed calling webhook "mpod.kb.io": failed to call webhook: Post "https://kruise-webhook-service.kube-system.svc:443/mutate-pod?timeout=10s": dial tcp 10.247.10.181:443: connect: connection refused
```

出现以上问题的原因是kurise-controller-manager组件不可用，导致部分命名空间下（非kube-system命名空间或不带control-plane: openkruise标签的命名空间）的Pod进行创建/更新/删除操作均会被拦截。

解决方案：

将kruise-controller-manager组件恢复正常即可正常调度。造成kruise-controller-manager异常的原因及解决建议如下：

- kruise-controller-manager所需的资源不够无法正常调度：建议为插件配置合理的资源。
- kruise-controller-manager配置了调度或亲和策略导致该Pod无法被正常调度：建议检查调度策略并配置合适的调度策略，以确保kruise-controller-manager被正常调度。

版本记录

表 14-106 OpenKruise 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.0.12	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	支持CCE v1.30集群	1.5.4
1.0.3	v1.23 v1.25 v1.27 v1.28 v1.29	首次提供OpenKruise插件。	1.5.4

14.8.3 Gatekeeper

插件简介

Gatekeeper是一个基于开放策略（OPA）的可定制的云原生策略控制器，有助于策略的执行和治理能力的加强，在集群中提供了更多符合Kubernetes应用场景的安全策略规则。

开源社区地址：**https://github.com/open-policy-agent/gatekeeper**

使用方式：**https://open-policy-agent.github.io/gatekeeper/website/docs/**

约束与限制

如果您已经在集群中部署了社区的Gatekeeper，请您先将其卸载后再安装CCE提供的Gatekeeper插件，否则可能会出现插件安装失败的情况。

注意事项

Gatekeeper提供的webhook的能力可能会影响Kubernetes基本资源的使用，请确保业务必须使用webhook能力，并慎重评估使用该插件引入的风险。

须知

Gatekeeper是CCE基于开源软件进行适配并集成的精选开源插件，CCE将提供全面的技术支持服务。然而，CCE不承担因开源软件缺陷导致的业务损失责任，也不承担赔偿或额外的服务，强烈建议用户定期升级软件以修复潜在问题。

安装步骤

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**opa-gatekeeper**插件，单击“安装”。

步骤2 在安装插件页面，设置“规格配置”。

表 14-107 插件规格配置

参数	参数说明
插件规格	该插件可配置“高可用”、“单实例”或“自定义”规格。
实例数	选择上方插件规格后，显示插件中的实例数。 选择“自定义”规格时，您可根据需求调整插件实例数。 实例数为1时插件不具备高可用能力，当插件实例所在节点异常时可能导致插件功能无法正常使用，请谨慎选择。
容器	选择“自定义”规格时，您可根据需求调整插件实例的容器规格。

步骤3 选择需要修改的配置，修改参数值。详情请参见**社区参数说明**。

步骤4 设置插件实例的部署策略。

说明

- 调度策略对于DaemonSet类型的插件实例不会生效。
- 设置多可用区部署或节点亲和策略时，需保证集群中存在满足调度策略的节点且拥有足够的资源，否则插件实例将无法运行。

表 14-108 插件调度配置

参数	参数说明
多可用区部署	• 优先模式：优先将插件的Deployment实例调度到不同可用区的节点上，如集群下节点不满足多可用区，插件实例将调度到单可用区下的不同节点。 • 均分模式：插件Deployment实例均匀调度到当前集群下各可用区，增加新的可用区后建议扩容插件实例以实现跨可用区高可用部署；均分模式限制不同可用区间插件实例数相差不超过1，单个可用区资源不足会导致后续其他实例无法调度。 • 强制模式：插件Deployment实例强制调度到不同可用区的节点上，每个可用区下最多运行一个实例。如集群下节点不满足多可用区，插件实例将无法全部运行。节点故障后，插件实例存在无法迁移风险。

参数	参数说明
节点亲和	• 不配置：插件实例不指定节点亲和调度。 • 指定节点调度：指定插件实例部署的节点。若不指定，将根据集群默认调度策略进行随机调度。 • 指定节点池调度：指定插件实例部署的节点池。若不指定，将根据集群默认调度策略进行随机调度。 • 自定义亲和策略：填写期望插件部署的节点标签实现更灵活的调度策略，若不填写将根据集群默认调度策略进行随机调度。 同时设置多条自定义亲和策略时，需要保证集群中存在同时满足所有亲和策略的节点，否则插件实例将无法运行。
容忍策略	容忍策略与节点的污点能力配合使用，允许（不强制）插件的 Deployment 实例调度到带有与之匹配的污点的节点上，也可用于控制插件的 Deployment 实例所在的节点被标记污点后插件的 Deployment 实例的驱逐策略。 插件会对实例添加针对**node.kubernetes.io/not-ready**和**node.kubernetes.io/unreachable**污点的默认容忍策略，容忍时间窗为60s。 详情请参见**设置容忍策略**。

步骤5 单击“安装”。

----结束

组件说明

表 14-109 Gatekeeper 组件

容器组件	说明	资源类型
gatekeeper-audit	提供audit相关信息。	Deployment
gatekeeper-controller-manager	提供Gatekeeper的webhook能力，按照用户自定义的策略对k8s资源进行控制。	Deployment

版本记录

表 14-110 Gatekeeper 插件版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.0.10	v1.23 v1.25 v1.27 v1.28 v1.29 v1.30	支持CCE v1.30集群	3.16.3
1.0.3	v1.23 v1.25 v1.27 v1.28 v1.29	首次提供Gatekeeper插件。	3.16.3

14.8.4 CCE 集群备份恢复（停止维护）

插件简介

CCE集群备份恢复插件（e-backup）提供集群备份恢复能力。它将用户应用数据和业务数据备份到OBS桶中，并提供数据的本地备份和远程备份的能力。

使用约束

- 备份/恢复过程中，用户要保证集群处于稳态，不要触发增删改等变更行为，以免出现备份/恢复失败或不完整；
- 若集群发生变更，建议等15分钟后，集群处于稳态，再做备份操作；
- 使用 云盘快照备份 时，仅提供EVS类型的PV卷做快照备份，并遵循快照的约束（如：不支持跨AZ恢复等），计费参考“云盘快照”；
- 使用 restic备份 时，提供对EVS、SFS、SFS Turbo、OBS类型的PV卷做数据备份，并上传到OBS备份仓库中；
- 开源的restic会对备份时间点的数据做自有快照，并上传数据，不影响用户后续数据的读写，但restic不做文件内容的校验和业务一致性校验，其特性遵循restic约束；
- restic占用内存与初次备份的PV卷数据大小有关，若数据大于500G，建议采用云存储提供的迁移方式进行，若使用本插件可以参考操作指南修改restic容器的资源配额；
- 备份过程中有状态应用业务数据一致性，需要用户可通过Hooks来保证业务数据一致性，比如：同步内存数据到文件中等；
- 在恢复过程中，支持通过配置调整来适应迁移前后的环境差异：
  - 应用可以从原命名空间恢复到指定的另一个命名空间中，但需要用户确认恢复应用间没有通过固定的service来访问该应用；

- 可以将应用的镜像地址(repo)换成另一个镜像地址，镜像名字和版本号在恢复过程中是保持不变的；
- 可以将应用使用的StorageClass的名字更换成新的StorageClass，注意需要使用同类型替换，比如：块存储 -> 块存储。

● 遵循开源velero和restic插件的约束，比如：在恢复过程中Service会清除clusterIP的IP地址，这样才能更适应源、目标K8s集群的差异。

安装插件

步骤1 登录CCE控制台，单击左侧导航栏的“插件市场”，找到e-backup插件，单击e-backup插件下的“安装”。

步骤2 在安装插件页面，选择要安装的集群，配置参数，然后单击“安装”。

当前支持配置如下参数。

volumeWorkerNum：备份volume的工作并发数量，默认为3。

----结束

使用插件

E-Backup使用OBS桶作为后端存储库，在备份前需要准备密钥并创建存储库。

备份支持备用应用（立即备份）和备用应用（定时备份），恢复支持立即恢复。

准备密钥

步骤1 获取访问密钥。

登录CCE控制台，在右上角用户名下选择“我的凭证”，在左侧选择“访问密钥”，单击“新增访问密钥”。

步骤2 创建密钥文件，并通过 base64 格式化成字符串。

```
# 创建密钥文件
$ vi credential-for-huawei-obs
HUAWEI_CLOUD_ACCESS_KEY_ID=your_access_key
HUAWEI_CLOUD_SECRET_ACCESS_KEY=your_secret_key

# 使用 base64 格式化字符串
$ base64 -w 0 credential-for-huawei-obs
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

步骤3 创建Secret。

按如下YAML文件创建Secret。

```
apiVersion: v1
kind: Secret
metadata:
  labels:
    secret.everest.io/backup: 'true'    #标识该secret用于E-Backup访问备份存储库
  name: secret-secure-opaque
  namespace: velero                  #必须和E-Backup置于同一namespace，取值必须为velero
type: cfe/secure-opaque
data:
  # credential文件经过base64编码后得到的字符串
  cloud: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

● secret 所在 namespace 必须和 E-Backup 实例所在namespace一致，即 velero。

- secret.data 中存储的是访问对象存储服务的密钥，其中 key 必须为 cloud，而 value 为**步骤2**中通过 base64 编码得到的字符串。一般通过 base64 编码后显示的字符串会有换行符，请在写入 secret.data 中时手动去除这些换行符。
- secret 需要打上标签“secret.everest.io/backup: true”，标识该 secret 是用于备份存储库的管理。

----结束

创建存储库

这里的备份存储库是指 E-Backup 用于获取和检测后端对象存储服务相关信息的 K8s 资源对象。

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  name: backup-location-001
  namespace: velero            #必须和E-Backup处于同一namespace
spec:
  config:
    endpoint: obs.{regionname}.myhuaweicloud.com   # OBS的endpoint
  credential:
    name: secret-secure-opaque   # 此前创建的secret的名字
    key: cloud                   # secret.data中的key值
  objectStorage:
    bucket: tools-cce        # OBS中的桶名
    prefix: for-backup       # 子路径名
  provider: huawei           # 使用OBS服务
```

- **除了 prefix 字段为选填外，其他字段必填**。provider 为固定值 huawei。
- endpoint 可以到**地区和终端节点**获取，都需要保证集群内各节点可访问该地址。当endpoint 不带协议头时（ http或者https ），默认启用 https。
- credential中的 name 和 key 需要配置正确，否则 E-Backup 无法访问后端存储库。

创建完成后等待30s用于备份存储库的检查和同步等工作，随后查看该备份存储库状态是否可用，PHASE 为 Available 表示可用，其他表示不可用。

```
$ kubectl get backupstoragelocations.velero.io backup-location-001 -n velero
NAME                PHASE       LAST VALIDATED   AGE    DEFAULT
backup-location-001   Available   23s              23m
```

此处如果PHASE 长时间没有Available，可通过查看E-Backup的日志定位问题。E-Backup安装后会在velero命名空间创建一个名为velero的工作负载，查看velero的日志即可。

立即备份

立即备份操作后会立刻执行备份过程，备份完成后停止，适用于克隆/迁移。

编辑备份模板，如下所示，随后通过 kubectl create 命令创建。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: backup-01
  namespace: velero
spec:
  includedNamespaces:
  - nginx
  - mysql
  labelSelector:
    matchExpressions:
    - key: direction
      operator: In
      values:
      - back
      - front
    matchLabels:
      app: nginx
      backup: velero
  runMode: Normal
  appData:
    volumes: Restic
  hooks:
    resources:
    - name: hook01
      includedNamespaces:
      - nginx
      labelSelector: {}
      pre:
      - exec:
          command:
          - /bin/sh
          - -c
          - echo hello > hello.txt && echo goodbye > goodbye.txt
          container: container-0
          onError: Fail
          timeout: 30s
      post:
      - exec:
          command:
          - /bin/sh
          - -c
          - echo hello > hello.txt && echo goodbye > goodbye.txt
          container: container-0
          onError: Fail
          timeout: 30s
  storageLocation: backup-location-001
  ttl: 720h0m0s
```

参数说明如下。

- 备份参数
  - **storageLocation**：指定了使用的备份存储库的名称，备份后的内容将会放置到对应的后端对象存储中，为**必填**字段。
  - **ttl**：指定了备份的内容将会在存储库中存放的时间，超期后会被删除，必须按照指定格式进行配置：h，m，s 分别表示“时，分，秒”。例如24h 表示一天，3h4m5s 表示 三小时四分钟五秒，默认为30天（720h0m0s）。
- 资源过滤相关：以下字段为过滤条件，都配置时取交集，相当于对集群中的所有资源进行筛选。

  - **includedNamespaces/excludedNamespaces**：指定对某些命名空间下的资源备份/不备份，互斥选项，选择一项配置即可，可选择多个namespace，默认表示所有namespace。
  - **labelSelector**：指定对具有特定标签的资源进行备份，参照 K8s 的标准用法，按需选择。
  - **runMode**：选择备份的运行方式，**必填**，Normal（备份应用和数据）/AppOnly（仅备份应用）/DataOnly（仅备份数据）/DryRun（用于验证，不备份）。
- 业务数据备份相关：当前支持两种方式对业务产生的实际数据进行备份，一种是everest 快照，只适用于使用 evs 类型的持久卷（pvc）作为数据卷；另一种是restic 备份，可备份除去 hostpath 类型以外的所有数据卷。两种方式支持混用。
  - **appData**：备份持久卷数据的方式，Restic/Snapshot，Snapshot默认不启用。Snapshot方式是为支持快照能力的存储，且集群中部署了csi快照插件时才能生效。
- **hook**：hook是用于在备份前或备份后执行某些指令，实现用户对备份的精细化控制，hook 类似于执行 kubectl exec 命令，目前只对 Pod 有效。
  - includedNamespaces/excludedNamespaces：指定对某些 namespace 下的 Pod 执行/不执行 hook ，互斥选项，默认表示所有namespace
  - labelSelector：指定对具有某些 label 的 Pod 执行 hook，参照 K8s 的标准用法，按需选择
  - command：指定 hook 的执行命令
  - container ：指定执行命令的容器名，当 Pod 有多个容器时用于精细化控制，默认为Pod 的第一个容器。
  - onError：指定 hook 执行失败时的行为，可选择 Continue/Fail ，默认为 Fail。
  - Continue：表示 hook 执行失败不影响后续动作的继续执行；Fail 表示 hook 执行失败将不会继续后续备份动作。
  - timeout：指定 hook 执行的超时时间，超过时间后认为 hook 执行失败，默认为 30s。

  hook 是针对 Pod 而言，hook 执行失败可能影响的后续备份动作也是针对执行 hook 的 Pod 而言，对其他对象比如 services 等的备份没有影响。

  由于 hook 是针对 Pod 而言，因此 hook 并非全局可用的。当需要执行 hook 的 Pod 没有被选择为备份对象时，hook 不会被执行。可以认为 hook 的 "includedNamespaces/excludedNamespaces" 配置是在被筛选为需要备份的对象中进一步进行筛选。

说明

上文中给出了所有的可配置项，在这里一方面基于实际备份场景，一方面为了用于便于操作，给出**备份配置建议**。

- 备份的保存时间按照"天"（24h）粒度进行控制。
- 当前应用基本会部署到特定namespace中，因此建议使用 includeNamespace 划定备份范围。如果需要更精细的备份对象控制，可以应用 labelSelector，前提是明确所有需要备份的对象具有相应的label。"includeNamespace + labelSelector" 能够满足绝大多数场景的使用。
- 使用 restic 备份业务数据时，如果对 OUT/IN 方式不熟悉，可以不对需要备份卷的 Pod 增加 annotation，通过简单的配置 defaultVolumesToRestic 选项为 true/false 对 Pod 使用的卷整体进行业务数据备份/不备份。
- 在需要对备份进行精细化控制时使用 hook ，尽量避免执行长时间运行的任务。另外，hook 中执行的命令不要直接操作文件系统。

备份执行后，可通过如下命令**查看备份状态**。status中会列出详细的状态。

```
$ kubectl -n velero get backups backup-01 -o yaml | grep  "phase"
  phase: Completed

$ kubectl -n velero get backups backup-01 -o yaml
......
status:
  ......
```

备份状态说明

- FailedValidation：备份模板配置错误，可以查看 Backup.Status.ValidationErrors 发现错误配置原因
- InProgress：备份正在进行中
- Completed：备份完成，没有错误
- PartiallyFailed：备份完成，但是备份某些对象的过程中出现错误（比如 hook 执行错误）
- Failed：备份失败，出现影响整体备份的错误
- Deleting：备份正在删除中

首次备份完成后，OBS桶中会出现backups和restic两个文件夹。

备份日志存储在OBS桶中，假设备份名为 backup-001，进入到OBS存储服务的页面，根据在备份存储库中配置的桶名和子路径名找到存储位置，进入 backups/backup-01 目录下，找到 backup-01-logs.gz 文件，随后下载、解压并查看。

周期备份

操作后会基于配置以一定的周期重复性地执行备份过程，比较适用于容灾。

编辑 Schedule 模板，如下所示，随后通过 kubectl create 命令创建。用户可以自行按照需要给 Schedule 模板打上 label，Schedule 上的 label 都会打到通过 schedule 创建的 backup 上。Schedule 创建到集群后，会立即执行一次备份，后续按照设定的定时周期重复执行备份过程。

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: schedule-backup-001
  namespace: velero
spec:
  schedule: 0 */10 * * *
```

```
template:
  runMode: Normal
  hooks: {}
  includedNamespaces:
  - nginx
  - mysql
  labelSelector:
    matchExpressions:
    - key: direction
      operator: In
      values:
      - back
      - front
    matchLabels:
      app: nginx
      backup: velero
  storageLocation: backup-location-001
  ttl: 720h0m0s
```

参数说明如下。

- schedule：创建的定时表达式，指定备份的周期执行时间。支持 @every格式 和Linux标准cron表达式。
  - @every *N*Unit：其中N表示一个正整数，Unit可以为s, m, h，表示每隔N个Unit时间触发一次，例如：@every 2h30m，每隔2小时30分执行一次。
  - 标准cron表达式：采用五子表达式，分别是 Minute，Hour，Day-of-Month，Month，Day-of-Week。
- template：备份的模板，与**备用应用（立即备份）**中spec一致。

删除备份

删除集群中创建的备份对象及其相关对象（比如：Backup/Restore/Schedule等），并且将后端存储库中的备份内容删除，适用于产生大量备份数据时进行的清理工作。

编辑 DeleteBackupRequest 模板，如下所示，随后通过 kubectl create 命令创建。

```
apiVersion: velero.io/v1
kind: DeleteBackupRequest
metadata:
  name: backup-001-delete
  namespace: velero
spec:
  backupName: backup-001  # 指定要删除的备份名
```

查看状态。

```
$ kubectl -n velero get deletebackuprequests backup-001-delete -o yaml | grep " phase"
  phase: InProgress
```

- InProgress：删除任务正在进行中。
- Processed：删除任务已经被处理过。

⚠ 注意

- Processed 状态只意味着 E-Backup 处理过该任务，但是不一定能够完成该任务。可以通过查看 deletebackuprequest.status.errors 字段查看执行删除任务期间出现的错误。如果 E-Backup 正确完整地处理完删除任务，则该 deletebackuprequest 对象本身也会被删除。
- 后端存储库（OBS桶）中的内容不要人为进行手动删除。

立即恢复

将某个立即备份作为数据源，恢复应用到另一个 namespace/集群 中，全场景适用。

编辑 Restore 模板，如下所示，随后通过 kubectl create 命令创建。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-01
  namespace: velero
spec:
  backupName: backup-01
  hooks:
    resources:
    - name: restore-hook-1
      includedNamespaces:
      - mysql
      labelSelector: {}
      postHooks:
      - init:
          initContainers:
          - name: restore-hook-init1
            image: alpine:latest
            volumeMounts:
            - mountPath: /restores/pvc1-vm
              name: pvc1-vm
            command:
            - /bin/ash
            - -c
            - echo -n "FOOBARBAZ" >> /restores/pvc1-vm/foobarbaz
          - name: restore-hook-init2
            image: alpine:latest
            volumeMounts:
            - mountPath: /restores/pvc2-vm
              name: pvc2-vm
            command:
            - /bin/ash
            - -c
            - echo -n "DEADFEED" >> /restores/pvc2-vm/deadfeed
      - exec:
          execTimeout: 1m
          waitTimeout: 5m
          onError: Fail
          container: mysql
          command:
          - /bin/bash
          - '-c'
          - 'while ! mysql_isready; do sleep 1; done'
      - exec:
          container: mysql
          waitTimeout: 6m
          execTimeout: 1m
          onError: Continue
          command:
          - /bin/bash
          - '-c'
          - 'mysql < /backup/backup.sql'
  includedNamespaces:
  - nginx
  - mysql
  namespaceMapping:
    nginx: nginx-another
    mysql: mysql-another
  labelSelector: {}
  preserveNodePorts: false
  storageClassMapping:
    disk: csi-disk
    obs: csi-obs
```

```
imageRepositoryMapping:
  quay.io/coreos:  swr.ap-southeast-1.myhuaweicloud.com/everest
```

参数说明如下。

- 选择数据源

  backupName 指定某个立即备份作为数据源，对该备份中的内容进行恢复，**必填**项。

- 资源过滤相关：这里的过滤是指对已经备份的内容进行的过滤，参考**备用应用（立即备份）**中资源过滤相关的配置。

- 特殊处理相关
  - **namespaceMapping**：指定将已备份的内容恢复到另一个 namespace 中，以 map 形式给出，格式为"source: target"，不要求新的 namespace 在目的集群存在
  - **storageClassMapping**：改变备份资源PV、PVC等使用的storageClassName，要求StorageClass类型相同。
  - **imageRepositoryMapping**：改变备份资源的images字段，用于仓库的映射关系，不包含镜像名字和标签的改变（防止迁移和升级耦合在一起），比如：quay.io/coreos/etcd:2.5 搬迁到SWR后，使用本地镜像仓库下 s**wr.ap-southeast-1.myhuaweicloud.com/everest/etcd:2.5**，配置格式为：**quay.io/coreos: swr.ap-southeast-1.myhuaweicloud.com/everest**
  - **preserveNodePorts**：如果设置成不保留，则不保留的是 Service 自动生成的 nodePort，用户手动配置的 nodePort 仍然会保留

- **hook**相关：Restore 模板的 hook 配置和 Backup 模板的不太相同，共有两种类型，一种是 init 类型，用于向 Pod 中添加 initContainer；一种是 exec 类型，用于执行某些指令。init 类型的 hook 请参照 K8s 中 initContainers 的定义方式进行配置，下面介绍有关 hook 整体选择的参数和 exec 类型的参数。
  - **includedNamespaces/excludedNamespaces**：指定对某些 namespace 下的 Pod 执行/不执行 hook ，互斥选项，默认表示所有namespace
  - **labelSelector**：指定对具有某些 label 的 Pod 执行 hook，参照 K8s 的标准用法，按需选择。
  - **command**：指定 hook 的执行命令。
  - **container**：指定执行命令的容器名，当 Pod 有多个容器时用于精细化控制，默认为Pod的第一个容器。
  - **onError**：指定 hook 执行失败时的行为，可选择 Continue/Fail ，默认为 Fail。
  - **Continue** ：表示 hook 执行失败不影响后续动作的继续执行；Fail 表示 hook 执行失败将不会继续后续动作。
  - **execTimeout**：指定 hook 执行的超时时间，超过时间后认为 hook 执行失败，默认为 30s。
  - **waitTimeout**：指定 E-Backup 准备执行 hook 时到容器开始执行 hook 的等待超时时间，超时后认为 hook 执行失败，默认为 0s，表示没有超时限制。

说明

- 数据源的选择请指定正确，保证该 Backup 是 Completed 状态
- 资源过滤相关的参数只在确有需要的时候进行配置，否则无需配置
- 业务数据的恢复由 E-Backup 自行根据备份时选择的方式进行针对性的恢复，用户无需担心，也没有相应的配置
- hook 的使用建议参照立即备份中的 hook 使用建议，waitTimeout 在无必要的情况下可以不进行配置
- 恢复时建议配置恢复到新的 namespace 下，按照备份什么就恢复什么的原则，避免自行的配置失误导致恢复后的应用无法启动运行

恢复执行后，可通过如下命令**查看恢复状态**。status中会列出详细的状态。

```
$ kubectl -n velero get restores restore-01 -o yaml | grep " phase"
  phase: Completed

$ kubectl -n velero get restores restore-01 -o yaml
......
status:
  ......
```

恢复状态说明

- FailedValidation：恢复模板配置错误，可以查看 Restore.Status.ValidationErrors 发现错误配置原因。
- InProgress：恢复正在进行中。
- Completed：恢复完成，没有错误。
- PartiallyFailed：恢复完成，但是恢复某些对象的过程中出现错误（比如 hook 执行错误）。
- Failed：恢复失败，出现影响整体恢复的错误。

查看恢复日志以及过程中的 warnings 和 errors 信息。

假设恢复名为 restore-01 ，进入到OBS控制台，根据在备份存储库中配置的桶名和子路径名找到存储位置，进入 restores/restore-01 目录下，有如下两个文件。

- restore-01-logs.gz：日志文件，随后下载、解压并查看日志。
- restore-01-results.gz：恢复结果文件，包含 warnings 和 errors 信息。

版本记录

表 14-111 CCE 集群备份恢复插件版本记录

插件版本	支持的集群版本	更新特性
1.2.0	v1.15 v1.17 v1.19 v1.21	• 支持EulerOS 2.0 (SP5，SP9) • 配置安全加固 • 功能优化

14.8.5 Kubernetes Web 终端（停止维护）

Kubernetes Web终端（web-terminal）是一款非常轻巧的终端服务器，支持在Web界面上使用Kubectl命令。它支持通过标准的Web浏览器和HTTP协议提供远程CLI，提供灵活的接口便于集成到独立系统中，可直接作为一个服务连接，通过cmdb获取信息并登录服务器。

web-terminal可以在Node.js支持的所有操作系统上运行，而不依赖于本机模块，快速且易于安装，支持多会话。

开源社区地址：**https://github.com/rabchev/web-terminal**

约束与限制

- 仅支持在1.21及以下版本的集群中安装此插件，暂不支持ARM集群。
- web-terminal插件当前已停止演进，请使用CloudShell方案代替。
- web-terminal中kubectl具有高级别操作权限，限账号以及具有CCE Administrator权限的IAM用户安装此插件，如需收缩插件中kubectl权限，请参见**收缩web-terminal权限**操作。
- 集群必须安装CoreDNS才能使用web-terminal。

注意事项

web-terminal插件能够对CCE集群进行管理，请用户妥善保管好登录密码，避免密码泄漏造成损失。

安装插件

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”，在右侧找到**web-terminal**，单击“安装”。

步骤2 配置以下参数。

- 访问类型：固定为节点访问，该插件默认以NodePort形式提供访问，需为集群任意一个节点绑定弹性IP才能使用。若集群没有绑定弹性IP，需绑定弹性IP。
- 用户名：默认为root，不可修改。
- 密码：登录web-terminal的密码，请务必记住该密码。web-terminal插件能够对CCE集群进行管理，请用户妥善保管好登录密码，避免密码泄漏造成损失。
- 确认密码：重新准确输入该密码。

步骤3 单击“安装”。

----结束

使用 web-terminal 插件连接集群

步骤1 登录CCE控制台，单击集群名称进入集群，单击左侧导航栏的“插件中心”。

步骤2 在右侧找到web-terminal，单击“访问”。

----结束

收缩 web-terminal 权限

web-terminal插件安装后，其kubectl默认使用cluster-admin ClusterRole权限，能够操作该集群K8s资源，若需手动配置为其他类型权限，可通过kubectl edit clusterrolebinding web-terminal修改web-terminal ServiceAccount绑定其他权限的ClusterRole。

ClusterRole、ClusterRoleBinding相关配置说明请参见**命名空间权限（Kubernetes RBAC授权）**。

须知

- 手动配置的web-terminal权限在web-terminal插件升级时存在被重置风险，需注意做好备份在插件升级后重新配置。
- 使用kubectl修改clusterrolebinding配置需确保当前kubectl有修改clusterrolebinding资源操作权限。

版本记录

表 14-112 Kubernetes Web 终端版本记录

插件版本	支持的集群版本	更新特性	社区版本
1.1.12	v1.15 v1.17 v1.19 v1.21	• 适配CCE v1.21集群	**0.6.6**
1.1.6	v1.15 v1.17 v1.19	• 配置seccomp默认规则	**0.6.6**
1.1.5	v1.15 v1.17 v1.19	• 兼容CCE v1.15集群	**0.6.6**
1.1.3	v1.17 v1.19	• 适配CCE v1.19集群	**0.6.6**
1.0.6	v1.15 v1.17	• 增加Pod安全策略资源	**0.6.6**
1.0.5	v1.9 v1.11 v1.13 v1.15 v1.17	• 支持v1.17版本集群	**0.6.6**

15 模板（Helm Chart）

15.1 模板概述

CCE提供了管理Helm Chart（模板）的控制台，能够帮助您方便的使用模板部署应用，并在控制台上管理应用。CCE使用的Helm版本为v3.8.2，支持上传Helm v3语法的模板包，具体请参见**通过模板部署应用**。

您也可以直接使用Helm客户端直接部署应用，使用Helm客户端部署应用不受版本控制，可以使用Helm v2或v3，具体请参见**通过Helm v2客户端部署应用**及**通过Helm v3客户端部署应用**。

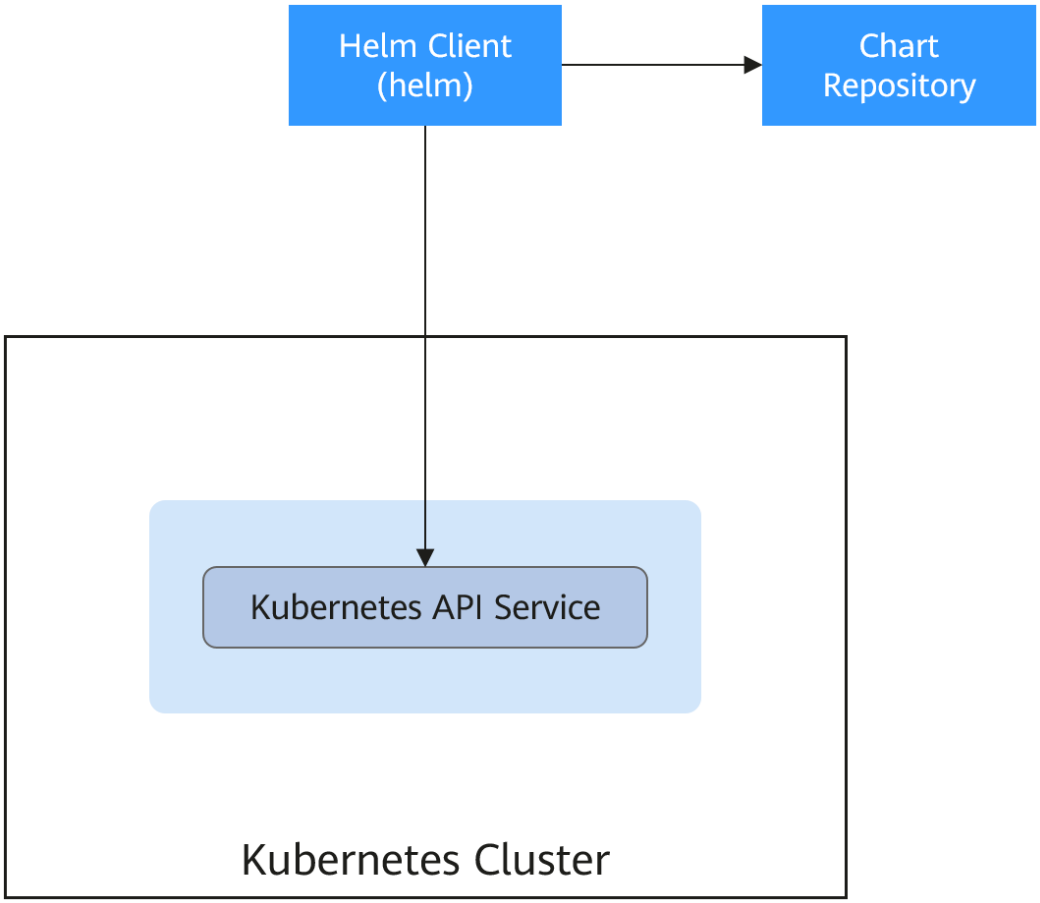
Helm

Helm是Kubernetes的包管理器，主要用来管理Charts。Helm Chart是用来封装Kubernetes原生应用程序的一系列YAML文件。可以在您部署应用的时候自定义应用程序的一些Metadata，以便于应用程序的分发。对于应用发布者而言，可以通过Helm打包应用、管理应用依赖关系、管理应用版本并发布应用到软件仓库。对于使用者而言，使用Helm后不用需要编写复杂的应用部署文件，可以以简单的方式在Kubernetes上查找、安装、升级、回滚、卸载应用程序。

Helm和Kubernetes之间的关系可以如下类比：

- Helm <–> Kubernetes
- Apt <–> Ubuntu
- Yum <–> CentOS
- Pip <–> Python

Helm的整体架构如下图：



Kubernetes的应用编排存在着一些问题，Helm可以用来解决这些问题，如下：

- 管理、编辑与更新大量的Kubernetes配置文件。
- 部署一个含有大量配置文件的复杂Kubernetes应用。
- 分享和复用Kubernetes配置和应用。
- 参数化配置模板支持多个环境。
- 管理应用的发布：回滚、diff和查看发布历史。
- 控制一个部署周期中的某一些环节。
- 发布后的测试验证。

15.2 通过模板部署应用

在CCE控制台上，您可以上传Helm模板包，然后在控制台安装部署，并对部署的实例进行管理。

须知

CCE从2022年9月开始，各region将逐步切换至Helm v3。模板管理不再支持Helm v2版本的模板，若您在短期内不能切换至Helm v3，可通过Helm v2 客户端在后台管理v2版本的模板。

约束与限制

- 单个用户可以上传模板的个数有限制，请以各个Region控制台界面中提示的实际值为准。
- CCE使用的Helm版本为v3.8.2，支持上传Helm v3版本语法的模板包。
- 模板若存在多个版本，则消耗对应数量的模板配额。
- 由于模板的操作权限同时具有较高的集群操作权限，因此租户应当谨慎授予用户对于模板生命周期管理的权限，包括上传模板的权限，以及创建、删除和更新模板实例的权限。

模板包规范

以下以redis为例，在准备redis模板包时根据模板包规范制作模板包。

- **命名要求**

  模板包命名格式为：**{name}-{version}**.tgz，其中**{version}**为版本号，格式为“主版本号.次版本号.修订号”，如redis-0.4.2.tgz。

  说明

  模板名称{name}的长度不能超过64个字符。

  版本号需遵循**语义化版本**规则。

  - 主版本号、次版本号为必选，修订号为可选。
  - 主版本号、次版本号、修订号的数值为整数，均需要≥0，且≤99。

- **目录结构**

  模板包的目录结构如下所示：

```
redis/
  templates/
  values.yaml
  README.md
  Chart.yaml
  .helmignore
```

  目录说明如**表15-1**所示，带*的为必选项：

表 15-1 模板包目录说明

参数	参数说明
* templates	用于存放所有的template（模板）文件。
* values.yaml	用于描述template文件所需的配置参数。 **须知** 定义template文件配置参数时，请注意此处定义的“镜像地址”务必和容器镜像仓库中对应的镜像地址保持一致。否则创建工作负载会异常，提示镜像拉取失败。 镜像地址获取方法如下：在CCE控制台，单击左侧导航栏的“镜像仓库”，进入容器镜像服务控制台。在“我的镜像 > 自有镜像”中，单击已上传镜像的名称，在“镜像版本 ”页签的“下载指令”栏中即可获取镜像地址，单击 按钮即可复制该指令。

参数	参数说明
README.md	一个markdown文件，包括： • 描述Chart提供的工作负载或服务。 • 运行Chart的前提。 • 解释values.yaml文件中的配置。 • 安装和配置Chart的相关信息。
* Chart.yaml	模板的基本信息说明。 注：Helm v3版本apiVersion从v1切换到了v2。
.helmignore	设定在工作负载安装时不需要读取templates的某些文件或数据。

上传模板

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“应用模板”，在右上角单击“上传模板”。

步骤2 单击“添加文件”，选中待上传的工作负载包后，单击“上传”。

说明

由于上传模板时创建OBS桶的命名规则由cce-charts-{region}-**{domain_name}**变为cce-charts-{region}-**{domain_id}**，其中旧命名规则中的domain_name系统会做base64转化并取前63位，如果您在现有命名规则的OBS桶中找不到模板，请在旧命名规则的桶中进行查找。

----结束

创建模板实例

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“应用模板”。

步骤2 在“我的模板”页签中，单击目标模板下的“安装”。

步骤3 参照表15-2设置安装工作负载参数。

表 15-2 安装工作负载参数说明

参数	参数说明
实例名称	新建模板实例名称，命名必须唯一。
命名空间	指定部署的命名空间。
选择版本	选择模板的版本。

参数	参数说明
配置文件	用户可以导入values.yaml文件，导入后可替换模板包中的values.yaml文件；也可直接在配置框中在线编辑模板参数。 **说明** 此处导入的values.yaml文件需符合yaml规范，即KEY:VALUE格式。对于文件中的字段不做任何限制。 导入的value.yaml的key值必须与所选的模板包的values.yaml保持一致，否则不会生效。即key不能修改。 1. 单击“添加文件”。 2. 选择对应的values.yaml文件，单击“打开”。

步骤4 配置完成后，单击“安装”。

在“模板实例”页签下可以查看模板实例的安装情况。

----结束

升级模板工作负载

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“应用模板”，在右侧选择“模板实例”页签。

步骤2 单击待升级工作负载后的“升级”，设置升级模板工作负载的参数。

步骤3 选择对应的模板版本。

步骤4 参照界面提示修改模板参数。单击“升级”。

步骤5 执行状态为“升级成功”时，表明工作负载升级成功。

----结束

回退模板工作负载

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“应用模板”，在右侧选择“模板实例”页签。

步骤2 单击待回退工作负载后的“回退”，选择要回退的工作负载版本，单击“回退”。

模板工作负载列表中，状态为“回退成功”时，表明工作负载回退成功。

----结束

卸载模板工作负载

步骤1 登录CCE控制台，单击集群名称进入集群，在左侧导航栏中选择“应用模板”，在右侧选择“模板实例”页签。

步骤2 单击待卸载模板实例后的“更多 > 卸载”，确认待卸载模板实例后，单击“是”。模板实例卸载后不能恢复，请谨慎操作。

----结束

15.3 Helm v2 与 Helm v3 的差异及适配方案

随着Helm v2 发布最终版本Helm 2.17.0，Helm v3 现在已是 Helm 开发者社区支持的唯一标准。为便于管理，建议用户尽快将模板切换至**Helm v3格式**。

当前社区从Helm v2演进到Helm v3，主要有以下变化：

1. **移除tiller**

   Helm v3 使用更加简单和灵活的架构，移除了 tiller，直接通过kubeconfig连接apiserver，简化安全模块，降低了用户的使用壁垒。

2. **改进了升级策略，采用三路策略合并补丁**

   Helm v2 使用双路策略合并补丁。在升级过程中，会对比最近一次发布的chart manifest和本次发布的chart manifest的差异，来决定哪些更改会应用到Kubernetes资源中。如果更改是集群外带的（比如通过kubectl edit），则修改不会被Helm识别和考虑。结果就是资源不会回滚到之前的状态。

   Helm v3 使用三路策略来合并补丁，Helm在生成一个补丁时，会考虑之前原来的manifest的活动状态。因此，Helm在使用原来的chart manifest生成新补丁时会考虑当前活动状态，并将其与之前原来的 manifest 进行比对，并再比对新的 manifest 是否有改动，并进行自动补全，以此来生成最终的更新补丁。

   详情及示例请见Helm官方文档：**https://v3.helm.sh/docs/faq/changes_since_helm2**

3. **默认存储驱动程序更改为secrets**

   Helm v2 默认情况下使用 ConfigMaps 存储发行信息，而在 Helm v3 中默认使用 Secrets。

4. **发布名称限制在namespace范围内**

   Helm v2 只使用tiller 的namespace 作为release信息的存储，这样全集群的release名字都不能重复。Helm v3只会在release安装的所在namespace记录对应的信息，这样release名称可在不同命名空间重用。应用和release命名空间一致。

5. **校验方式改变**

   Helm v3 对模板格式的校验更加严格，如Helm v3 将chart.yaml的apiVersion从v1切换到v2，针对Helm v2的chart.yaml，强要求指定apiVersion为v1。可安装Helm v3客户端后，通过执行helm lint命令校验模板格式是否符合v3规范。

   适配方案：根据Helm官方文档 **https://helm.sh/docs/topics/charts/**适配Helm v3模板，apiVersion字段必填。

6. **废弃crd-install**

   Helm v3删除了crd-install hook， 并用chart中的crds目录替换。需要注意的是，crds目录中的资源只有在release安装时会部署，升级时不会更新，删除时不会卸载crds目录中的资源。若crd已存在，则重复安装不会报错。

   适配方案：根据Helm官方文档 **https://helm.sh/docs/chart_best_practices/custom_resource_definitions/**, 当前可使用crds目录或者将crd定义单独放入chart。考虑到目前不支持使用Helm升级或删除CRD，推荐分隔chart，将CRD定义放入chart中，然后将所有使用该CRD的资源放到另一个 chart中进行管理。

7. **未通过helm创建的资源不强制update，releaes默认不强制升级**

   Helm v3强制升级逻辑变化，不再是升级失败后走删除重建，而是直接走put更新逻辑。因此当前CCE release升级默认使用非强制更新逻辑，无法通过Patch更新的资源将导致release升级失败。若环境存在同名资源且无Helm V3的归属标记app.kubernetes.io/managed-by: Helm，则会提示资源冲突。

适配方案：删除相关资源，并通过Helm创建。

8. **Release history数量限制更新**

   为避免release 历史版本无限增加，当前release升级默认只保留最近10个历史版本。

更多变化和详细说明请参见Helm官方文档

- Helm v2与Helm v3的区别：**https://v3.helm.sh/docs/faq/changes_since_helm2**
- Helm v2如何迁移到Helm v3：**https://helm.sh/docs/topics/v2_v3_migration**

15.4 通过 Helm v2 客户端部署应用

须知

CCE从2022年9月开始，各region将逐步切换至Helm v3。模板管理不再支持Helm v2版本的模板，若您在短期内不能切换至Helm v3，可通过Helm v2 客户端在后台管理v2版本的模板。

前提条件

在CCE中创建的Kubernetes集群已对接kubectl，具体请参见**使用kubectl连接集群**。

注意事项

CCE当前会尝试转换v2模板实例到v3模板实例。若在后台操作Helm v2模板实例，删除实例后，发现CCE 模板管理页面仍有实例信息，单击删除即可。

安装 Helm v2

本文以Helm v2.17.0为例进行演示。

如需选择其他合适的版本，请访问**https://github.com/helm/helm/releases**。

步骤1 在连接集群的虚拟机上下载Helm客户端。

```
wget https://get.helm.sh/helm-v2.17.0-linux-amd64.tar.gz
```

步骤2 解压Helm包。

```
tar -xzvf helm-v2.17.0-linux-amd64.tar.gz
```

步骤3 将helm复制到系统path路径下，以下为/usr/local/bin/helm。

```
mv linux-amd64/helm /usr/local/bin/helm
```

步骤4 因为Kubernetes APIServer开启了RBAC访问控制，所以需创建tiller使用的service account:tiller并给其分配cluster-admin这个集群内置的ClusterRole。按如下创建tiller的资源账号。

vim tiller-rbac.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
```



```
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

步骤5 部署tiller资源账号。

```
kubectl apply -f tiller-rbac.yaml
```

步骤6 初始化Helm, 部署tiller的Pod。

```
helm init --service-account tiller --skip-refresh
```

步骤7 查看状态。

```
kubectl get pod -n kube-system -l app=helm
```

回显如下

```
NAME                            READY   STATUS    RESTARTS   AGE
tiller-deploy-7b56c8dfb7-fxk5g   1/1     Running   1          23h
```

步骤8 查看helm版本。

```
# helm version
Client: &version.Version{SemVer:"v2.17.0", GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe",
GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.17.0", GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe",
GitTreeState:"clean"}
```

----结束

安装 Helm 模板 chart 包

CCE提供的模板不能满足要求时，可下载模板的chart包进行安装。

在https://github.com/helm/charts的stable目录中查找您需要的chart包，下载后将chart包上传至节点。

1. 下载并解压已获取的chart包，一般chart包格式为.zip。
   ```
   unzip chart.zip
   ```
2. 安装Helm模板。
   ```
   helm install aerospike/
   ```
3. 安装完成后，执行**helm list**查看已经安装的模板实例状态。

常见问题

- **执行Helm version时，提示如下错误信息：**
  ```
  Client:
  &version.Version{SemVer:"v2.17.0",
  GitCommit:"a690bad98af45b015bd3da1a41f6218b1a451dbe", GitTreeState:"clean"}
  E0718 11:46:10.132102    7023 portforward.go:332] an error occurred
  forwarding 41458 -> 44134: error forwarding port 44134 to pod
  d566b78f997eea6c4b1c0322b34ce8052c6c2001e8edff243647748464cd7919, uid : unable
  to do port forwarding: socat not found.
  Error: cannot connect to Tiller
  ```


出现上述问题，说明未安装socat，请执行如下命令安装socat。

```
yum install socat -y
```

- **在操作系统为EulerOS 2.9或Huawei Cloud EulerOS的节点执行yum install socat –y，如报如下错误：**

  No match for argument: socat

  Error: Unable to find a match: socat

  说明镜像未自带socat镜像，请手动下载rpm包，执行以下命令安装，其中rpm包名请根据实际情况进行替换：

  ```
  rpm -i socat-1.7.3.2-8.oe1.x86_64.rpm
  ```

  表 15-3 socat 镜像 rpm 包下载地址

操作系统	下载地址
EulerOS 2.9	• x86 • ARM
Huawei Cloud EulerOS 2.0	• x86 • ARM
Huawei Cloud EulerOS 1.1	x86

- **socat已安装，执行Helm version时，提示如下错误信息：**

  ```
  test@local:~/k8s/helm/test$ helm version
  Client: &version.Version{SemVer:"v3.3.0",
  GitCommit:"021cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
  Error: cannot connect to Tiller
  ```

  Helm模板从“.Kube/config”中读取配置证书和kubernetes进行通讯，出现上述错误信息说明kubectl配置有误，请重新对接kubectl，具体请参见**使用kubectl连接集群**。

- **对接云存储后，存储未创建成功。**

  出现上述问题可能是创建的pvc中annotation字段导致的，请修改模板名称后再次进行安装。

- **如果kubectl没有配置好，helm install时会出现如下报错：**

  ```
  [root@prometheus-57046 ~]# helm install prometheus/ --generate-name
  WARNING: This chart is deprecated
  Error: Kubernetes cluster unreachable: Get "http://localhost:8080/version?timeout=32s": dial tcp
  [::1]:8080: connect: connection refused
  ```

  解决办法：给节点配置kubeconfig，配置方法请参见**使用kubectl连接集群**。

15.5 通过 Helm v3 客户端部署应用

前提条件

- 在CCE中创建的Kubernetes集群已对接kubectl，具体请参见**使用kubectl连接集群**。
- 部署Helm时如果需要拉取公网镜像，请提前为节点绑定弹性公网IP。

安装 Helm v3

本文以Helm v3.3.0为例进行演示。

如需选择其他合适的版本，请访问**https://github.com/helm/helm/releases**。

步骤1 在连接集群的虚拟机上下载Helm客户端。

```
wget https://get.helm.sh/helm-v3.3.0-linux-amd64.tar.gz
```

步骤2 解压Helm包。

```
tar -xzvf helm-v3.3.0-linux-amd64.tar.gz
```

步骤3 将Helm复制到系统path路径下，以下为/usr/local/bin/helm。

```
mv linux-amd64/helm /usr/local/bin/helm
```

步骤4 查看Helm版本。

```
helm version
version.BuildInfo{Version:"v3.3.0", GitCommit:"e29ce2a54e96cd02ccfce88bee4f58bb6e2a28b6",
GitTreeState:"clean", GoVersion:"go1.13.4"}
```

----结束

安装 Helm 模板包

您可以使用Helm安装模板包（Chart），在使用Helm命令安装模板包前，您可能需要了解三大概念帮助您更好地使用Helm。

- 模板包（Chart）：模板包中含有Kubernetes应用的资源定义以及大量的配置文件。
- 仓库（Repository）：仓库是用于存放共享模板包的地方，您可以从仓库中下载模板包至本地安装，也可以选择直接在线安装。
- 实例（Release）：实例是Helm在Kubernetes集群中安装模板包后的运行结果。一个模板包通常可以在一个集群中安装多次，每次安装都会创建一个新的实例。以MySQL模板包为例，如果您想在集群中运行两个数据库，可以安装该模板包两次，每一个数据库都会拥有自己的release 和release name。

更多关于Helm命令的使用方法请参见**使用Helm**。

步骤1 从Helm官方推荐的仓库**Artifact Hub**中查找模板包，并配置Helm仓库。

```
helm repo add {repo_name} {repo_addr}
```

例如，以**WordPress模板包**为例：

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

步骤2 使用**helm install**命令安装模板包。

```
helm install {release_name} {chart_name} --set key1=val1
```

例如，以安装WordPress为例，**步骤1**添加的仓库中WordPress的模板包为bitnami/wordpress，并将实例自定义命名为my-wordpress，同时指定一些配置参数。

```
helm install my-wordpress bitnami/wordpress \
    --set mariadb.primary.persistence.enabled=true \
    --set mariadb.primary.persistence.storageClass=csi-disk \
    --set mariadb.primary.persistence.size=10Gi \
    --set persistence.enabled=false
```

您可以使用**helm show values *{chart_name}***命令查看模板可配置的选项。例如，查看WordPress的可配置项：

```
helm show values bitnami/wordpress
```


步骤3 查看已安装的模板实例。

```
helm list
```

----结束

常见问题

- **执行Helm version时，提示如下错误信息：**

```
Client:
&version.Version{SemVer:"v3.3.0",
GitCommit:"012cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
E0718 11:46:10.132102    7023 portforward.go:332] an error occurred
forwarding 41458 -> 44134: error forwarding port 44134 to pod
d566b78f997eea6c4b1c0322b34ce8052c6c2001e8edff243647748464cd7919, uid : unable
to do port forwarding: socat not found.
Error: cannot connect to Tiller
```

出现上述问题，说明未安装socat，请执行如下命令安装socat。

```
yum install socat -y
```

- **在操作系统为EulerOS 2.9或Huawei Cloud EulerOS的节点执行yum install socat –y，如报如下错误：**

```
No match for argument: socat
Error: Unable to find a match: socat
```

说明节点镜像未自带socat镜像，请手动下载rpm包后，执行以下命令安装，其中rpm包名请根据实际情况进行替换：

```
rpm -i socat-1.7.3.2-8.oe1.x86_64.rpm
```

表 15-4 socat 镜像 rpm 包下载地址

操作系统	下载地址
EulerOS 2.9	• x86 • ARM
Huawei Cloud EulerOS 2.0	• x86 • ARM
Huawei Cloud EulerOS 1.1	x86

- **socat已安装，执行Helm version时，提示如下错误信息：**

```
$ helm version
Client: &version.Version{SemVer:"v3.3.0",
GitCommit:"021cb0ac1a1b2f888144ef5a67b8dab6c2d45be6", GitTreeState:"clean"}
Error: cannot connect to Tiller
```

Helm模板从节点上的“.Kube/config”路径中读取配置证书和Kubernetes进行通讯，出现上述错误信息说明kubectl配置有误，请重新对接kubectl，具体请参见**使用kubectl连接集群**。

- **对接云存储后，存储未创建成功。**

出现上述问题可能是创建的PVC中annotation字段导致的，请修改模板名称后再次进行安装。

- **如果kubectl没有配置好，helm install时会出现如下报错：**

```
# helm install prometheus/ --generate-name
WARNING: This chart is deprecated
Error: Kubernetes cluster unreachable: Get "http://localhost:8080/version?timeout=32s": dial tcp
[::1]:8080: connect: connection refused
```


解决办法：给节点配置kubeconfig，配置方法请参见**使用kubectl连接集群**。

15.6 Helm v2 Release 转换成 Helm v3 Release

背景介绍

当前CCE已全面支持Helm v3版本，用户可通过本指南将已创建的v2 release转换成v3 release，从而更好地使用v3的特性。因Helm v3底层相对于Helm v2来说，一些功能已被弃用或重构，因此转换会有一定风险，需转换前进行模拟转换。

该指南参考社区文档：**https://github.com/helm/helm-2to3**

注意事项：

- Helm v2 release信息存储在configmap中，Helm v3 release信息存储在secrets中。
- 若用户通过前端console操作，在获取实例、更新实例等操作中CCE会自动尝试转换v2模板实例到v3模板实例。若用户仅在后台操作实例，需通过该指南进行转换操作。

转换流程（不使用 Helm v3 客户端）

步骤1 在CCE节点上下载helm 2to3 转换插件。

```
wget https://github.com/helm/helm-2to3/releases/download/v0.10.2/helm-2to3_0.10.2_linux_amd64.tar.gz
```

步骤2 解压插件包。

```
tar -xzvf helm-2to3_0.10.2_linux_amd64.tar.gz
```

步骤3 模拟转换。

以test-convert实例为例，执行以下命令进行转换的模拟。若出现以下提示，说明模拟转换成功。

```
# ./2to3 convert --dry-run --tiller-out-cluster -s configmaps test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
```

步骤4 执行正式转换。若出现以下提示，说明转换成功。

```
# ./2to3 convert --tiller-out-cluster -s configmaps test-convert
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" created.
[Helm 3] Release "test-convert" created.
Release "test-convert" was converted successfully from Helm v2 to Helm v3.
Note: The v2 release information still remains and should be removed to avoid conflicts with the migrated v3 release.
v2 release information should only be removed using `helm 2to3` cleanup and when all releases have been migrated over.
```

步骤5 转换完成后进行v2 release资源的清理，同样先进行模拟清理，成功后正式清理v2 release资源。

模拟清理：

```
# ./2to3 cleanup --dry-run --tiller-out-cluster -s configmaps --name test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
```



```
Run without --dry-run to take the actions described below:
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
```

正式清理：

```
# ./2to3 cleanup --tiller-out-cluster -s configmaps --name test-convert
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" d
```

----结束

转换流程（使用 Helm v3 客户端）

步骤1 安装Helm v3客户端，参见**安装Helm v3**。

步骤2 安装转换插件。

```
# helm plugin install https://github.com/helm/helm-2to3
Downloading and installing helm-2to3 v0.10.2 ...
https://github.com/helm/helm-2to3/releases/download/v0.10.2/helm-2to3_0.10.2_linux_amd64.tar.gz
Installed plugin: 2to3
```

步骤3 查看已安装的插件，确认插件已安装。

```
# helm plugin list
NAME VERSION DESCRIPTION
2to3  0.10.2      migrate and cleanup Helm v2 configuration and releases in-place to Helm v3
```

步骤4 模拟转换。

以test-convert实例为例，执行以下命令进行转换的模拟。若出现以下相关提示，说明模拟转换成功。

```
# helm 2to3 convert --dry-run --tiller-out-cluster -s configmaps test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
```

步骤5 执行正式转换。若出现以下提示，说明转换成功。

```
# helm 2to3 convert --tiller-out-cluster -s configmaps test-convert
Release "test-convert" will be converted from Helm v2 to Helm v3.
[Helm 3] Release "test-convert" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" will be created.
[Helm 3] ReleaseVersion "test-convert.v1" created.
[Helm 3] Release "test-convert" created.
Release "test-convert" was converted successfully from Helm v2 to Helm v3.
Note: The v2 release information still remains and should be removed to avoid conflicts with the migrated v3 release.
v2 release information should only be removed using `helm 2to3` cleanup and when all releases have been migrated over.
```

步骤6 正式转换成功后，用户可通过helm list查看已转换成功的模板实例。

```
# helm list
NAME            NAMESPACE   REVISION UPDATED                         STATUS      CHART           APP VERSION
test-convert    default     1        2022-08-29 06:56:28.166918487 +0000 UTC    deployed    test-helmold-1
```


步骤7 转换完成后进行V2 release资源的清理，同样先进行模拟清理，成功后正式清理V2 release资源。

模拟清理：

```
# helm 2to3 cleanup --dry-run --tiller-out-cluster -s configmaps --name test-convert
NOTE: This is in dry-run mode, the following actions will not be executed.
Run without --dry-run to take the actions described below:
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
```

正式清理：

```
# helm 2to3 cleanup --tiller-out-cluster -s configmaps --name test-convert
WARNING: "Release 'test-convert' Data" will be removed.

[Cleanup/confirm] Are you sure you want to cleanup Helm v2 data? [y/N]: y
Helm v2 data will be cleaned up.
[Helm 2] Release 'test-convert' will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" will be deleted.
[Helm 2] ReleaseVersion "test-convert.v1" deleted.
[Helm 2] Release 'test-convert' deleted.
Helm v2 data was cleaned up successfully.
```

----结束

16 权限

16.1 CCE 权限概述

CCE权限管理是在统一身份认证服务（IAM）与Kubernetes的角色访问控制（RBAC）的能力基础上，打造的细粒度权限管理功能，支持基于IAM的细粒度权限控制和IAM Token认证，支持集群级别、命名空间级别的权限控制，帮助用户便捷灵活的对租户下的IAM用户、用户组设定不同的操作权限。

如果您需要对CCE集群及相关资源进行精细的权限管理，例如限制不同部门的员工拥有部门内资源的细粒度权限，您可以使用CCE权限管理提供的增强能力进行多维度的权限管理。

本章节将介绍CCE权限管理机制及其涉及到的基本概念。如果当前账号已经能满足您的要求，您可以跳过本章节，不影响您使用CCE服务的其它功能。

CCE 支持的权限管理能力

CCE的权限管理包括“集群权限”和“命名空间权限”两种能力，能够从集群和命名空间层面对用户组或用户进行细粒度授权，具体解释如下：

- **集群权限：**是基于IAM系统策略的授权，可以通过用户组功能实现IAM用户的授权。用户组是用户的集合，通过集群权限设置可以让某些用户组操作集群（如创建/删除集群、节点、节点池、模板、插件等），而让某些用户组仅能查看集群。

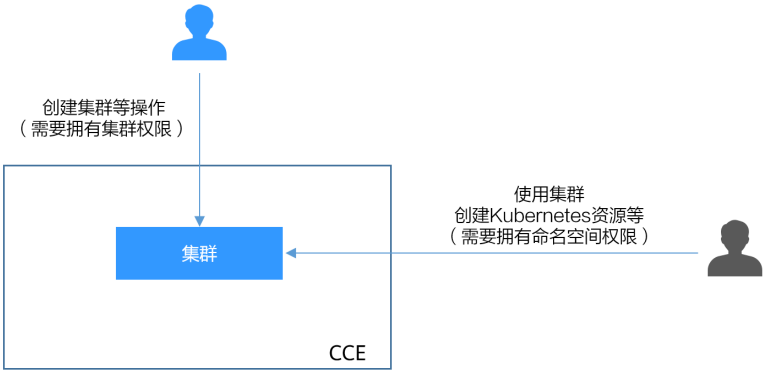
  集群权限涉及CCE非Kubernetes API，支持IAM细粒度策略、企业项目管理相关能力。

- **命名空间权限：**是基于Kubernetes RBAC（Role-Based Access Control，基于角色的访问控制）能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同Kubernetes资源的权限。同时CCE基于开源能力进行了增强，可以支持基于IAM用户或用户组粒度进行RBAC授权、IAM token直接访问API进行RBAC认证鉴权。

  命名空间权限涉及CCE Kubernetes API，基于Kubernetes RBAC能力进行增强，支持对接IAM用户/用户组进行授权和认证鉴权，但与IAM细粒度策略独立。

CCE的权限可以从使用的阶段分为两个阶段来看，第一个阶段是创建和管理集群的权限，也就是拥有创建/删除集群、节点等资源的权限。第二个阶段是使用集群Kubernetes资源（如工作负载、Service等）的权限。

图 16-1 权限示例图



清楚了集群权限和命名空间权限后，您就可以通过这两步授权，做到精细化的权限控制。

集群权限（IAM 授权）与命名空间权限（Kubernetes RBAC 授权）的关系

拥有不同集群权限（IAM授权）的用户，其拥有的命名空间权限（Kubernetes RBAC授权）不同。**表16-1**给出了不同用户拥有的命名空间权限详情。

表 16-1 不同用户拥有的命名空间权限

用户类型	1.13及以上版本的集群
拥有Tenant Administrator权限的用户（例如账号）	全部命名空间权限
拥有CCE Administrator权限的IAM用户	全部命名空间权限
拥有CCE FullAccess或者CCE ReadOnlyAccess权限的IAM用户	按Kubernetes RBAC授权
拥有Tenant Guest权限的IAM用户	按Kubernetes RBAC授权

kubectl 权限说明

您可以通过**kubectl访问集群**的Kubernetes资源，那kubectl拥有哪些Kubernetes资源的权限呢?

kubectl访问CCE集群是通过集群上生成的配置文件（kubeconfig.json）进行认证，kubeconfig.json文件内包含用户信息，CCE根据用户信息的权限判断kubectl有权限访问哪些Kubernetes资源。即哪个用户获取的kubeconfig.json文件，kubeconfig.json就拥有哪个用户的信息，这样使用kubectl访问时就拥有这个用户的权限。而用户拥有的权限就是**表16-1**所示的权限。

联邦用户支持说明

IAM支持基于SAML、OIDC协议的单点登录，如果您已经有自己的企业管理系统，同时您的用户需要使用您账号内的云服务资源，您可以使用IAM的身份提供商功能，实现用户使用企业管理系统账号单点登录，这一过程称之为联邦身份认证。

通过联邦身份认证访问的用户称为联邦用户，联邦用户相当于IAM用户。

联邦用户使用CCE时需要注意如下两点。

- 用户创建CCE集群时，会在集群中默认为该用户创建一个cluster-admin权限（管理员权限），联邦用户由于每次登录注销都会改变用户ID，所以在CCE控制台权限管理处，权限用户会显示已删除，请勿删除该权限，否则会导致鉴权失败。此种情况下建议在CCE为某个用户组创建cluster-admin权限，将联邦用户加入此用户组。
- 联邦用户不支持创建永久访问密钥AK/SK，在需要使用AK/SK的场景（如创建OBS类型PV/PVC时），只能由账号或是实体IAM用户创建密钥，共享给联邦用户。由于密钥表示用户所拥有的权限，因此建议由与联邦用户同在一个用户组的实体IAM用户创建并分享密钥。

IAM 支持的授权项

策略包含系统策略和自定义策略，如果系统策略不满足授权要求，管理员可以创建自定义策略，并通过给用户组授予自定义策略来进行精细的访问控制。策略支持的操作与API相对应，授权项列表说明如下：

- 权限：允许或拒绝某项操作。
- 对应API接口：自定义策略实际调用的API接口。
- 授权项：自定义策略中支持的Action，在自定义策略中的Action中写入授权项，可以实现授权项对应的权限功能。
- 依赖的授权项：部分Action存在对其他Action的依赖，需要将依赖的Action同时写入授权项，才能实现对应的权限功能。
- IAM项目(Project)/企业项目(Enterprise Project)：自定义策略的授权范围，包括IAM项目与企业项目。授权范围如果同时支持IAM项目和企业项目，表示此授权项对应的自定义策略，可以在IAM和企业管理两个服务中给用户组授权并生效。如果仅支持IAM项目，不支持企业项目，表示仅能在IAM中给用户组授权并生效，如果在企业管理中授权，则该自定义策略不生效。关于IAM项目与企业项目的区别，详情请参见：**IAM与企业管理的区别**。

说明

“√”表示支持，“x”表示暂不支持。

云容器引擎（CCE）支持的自定义策略授权项如下所示：

表 16-2 Cluster

权限	对应API接口	授权项（Action）	IAM项目(Project)	企业项目(Enterprise Project)
获取指定项目下的集群	GET /api/v3/projects/{project_id}/clusters	cce:cluster:list	√	√
获取指定的集群	GET /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:get	√	√
创建集群	POST /api/v3/projects/{project_id}/clusters	cce:cluster:create	√	√

权限	对应API接口	授权项（Action）	IAM项目(Project)	企业项目(Enterprise Project)
更新指定的集群	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:update	√	√
删除集群	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}	cce:cluster:delete	√	√
升级集群	POST /api/v2/projects/:projectid/clusters/:clusterid/upgrade	cce:cluster:upgrade	√	√
唤醒集群	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/operation/awake	cce:cluster:start	√	√
休眠集群	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/operation/hibernate	cce:cluster:stop	√	√
变更集群规格	POST /api/v2/projects/{project_id}/clusters/:clusterid/resize	cce:cluster:resize	√	√
获取集群证书	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/clustercert	cce:cluster:get	√	√

表 16-3 Node

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
获取集群下所有节点	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes	cce:node:list	√	√
获取指定的节点	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes/{node_id}	cce:node:get	√	√

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
创建节点	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes	cce:node:create	√	√ **说明** 使用企业项目授权创建节点需额外添加evs:quota:get的全局权限。
更新指定的节点	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes/{node_id}	cce:node:update	√	√
删除节点	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}/nodes/{node_id}	cce:node:delete	√	√

表 16-4 Job

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
获取任务信息	GET /api/v3/projects/{project_id}/jobs/{job_id}	cce:job:get	√	√
列出所有任务	GET /api/v2/projects/{project_id}/jobs	cce:job:list	√	√
删除所有任务或删除单个任务	DELETE /api/v2/projects/{project_id}/jobs DELETE /api/v2/projects/{project_id}/jobs/{job_id}	cce:job:delete	√	√

表 16-5 Nodepool

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
获取集群下所有节点池	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools	cce:nodepool:list	√	√
获取节点池	GET /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools/{nodepool_id}	cce:nodepool:get	√	√
创建节点池	POST /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools	cce:nodepool:create	√	√
更新节点池信息	PUT /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools/{nodepool_id}	cce:nodepool:update	√	√
删除节点池	DELETE /api/v3/projects/{project_id}/clusters/{cluster_id}/nodepools/{nodepool_id}	cce:nodepool:delete	√	√

表 16-6 Chart

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
更新模板	PUT /v2/charts/{id}	cce:chart:update	√	×
上传模板	POST /v2/charts	cce:chart:upload	√	×
列出所有模板	GET /v2/charts	cce:chart:list	√	×
获取模板信息	GET /v2/charts/{id}	cce:chart:get	√	×
删除模板	DELETE /v2/charts/{id}	cce:chart:delete	√	×

表 16-7 Release

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
更新升级模板实例	PUT /v2/releases/{name}	cce:release:update	√	√
列出所有模板实例	GET /v2/releases	cce:release:list	√	√
创建模板实例	POST /v2/releases	cce:release:create	√	√
获取模板实例信息	GET /v2/releases/{name}	cce:release:get	√	√
删除模板实例	DELETE /v2/releases/{name}	cce:release:delete	√	√

表 16-8 Storage

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
创建PersistentVolumeClaim	POST /api/v1/namespaces/{namespace}/cloudpersistentvolumeclaims	cce:storage:create	√	√
删除PersistentVolumeClaim	DELETE /api/v1/namespaces/{namespace}/cloudpersistentvolumeclaims/{name}	cce:storage:delete	√	√
列出所有磁盘	GET /storage/api/v1/namespaces/{namespace}/listvolumes	cce:storage:list	√	√

表 16-9 Addon

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
创建插件实例	POST /api/v3/addons	cce:addonInstance:create	√	√
获取插件实例	GET /api/v3/addons/{id}?cluster_id={cluster_id}	cce:addonInstance:get	√	√
列出所有插件实例	GET /api/v3/addons?cluster_id={cluster_id}	cce:addonInstance:list	√	√
删除插件实例	DELETE /api/v3/addons/{id}?cluster_id={cluster_id}	cce:addonInstance:delete	√	√
更新升级插件实例	PUT /api/v3/addons/{id}	cce:addonInstance:update	√	√

表 16-10 Quota

权限	对应API接口	授权项	IAM项目(Project)	企业项目(Enterprise Project)
查询配额详情	GET /api/v3/projects/{project_id}/quotas	cce:quota:get	√	√

16.2 集群权限（IAM 授权）

CCE集群权限是基于IAM**系统策略**和**自定义策略**的授权，可以通过用户组功能实现IAM用户的授权。

⚠ 注意

- 集群权限仅针对与集群相关的资源（如集群、节点等）有效，您必须确保同时配置了**命名空间权限**，才能有操作Kubernetes资源（如工作负载、Service等）的权限。
- 使用CCE控制台查看集群时，显示情况依赖于命名空间权限的设置情况，如果没有设置命名空间权限，则无法查看集群下的资源，详情请参见**CCE控制台的权限依赖**。

前提条件

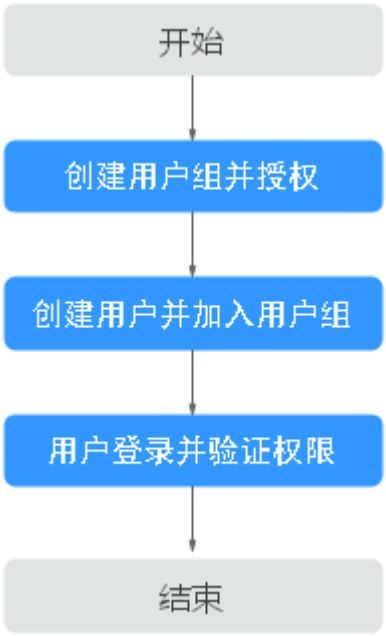
- 给用户组授权之前，请您了解用户组可以添加的CCE系统策略，并结合实际需求进行选择，CCE支持的系统策略及策略间的对比，请参见**CCE系统权限**。若您需要对除CCE之外的其它服务授权，IAM支持服务的所有策略请参见**系统权限**。
- 拥有Security Administrator（IAM除切换角色外所有权限）权限的用户（如账号默认拥有此权限），才能看见CCE控制台权限管理页面当前用户组及用户组所拥有的权限。

配置说明

CCE控制台“权限管理 > 集群权限”页面中创建用户组和具体权限设置均是跳转到IAM控制台进行具体操作，设置完后在集群权限页面能看到用户组所拥有的权限。本章节描述操作直接以IAM中操作为主，不重复介绍在CCE控制台如何跳转。

示例流程

图 16-2 给用户授予 CCE 权限流程



1. **创建用户组并授权**。
   在IAM控制台创建用户组，并授予CCE权限，例如CCE ReadOnlyAccess。

   说明

   CCE服务按区域部署，在IAM控制台授予CCE权限时请选择“区域级项目”。
2. **创建用户并加入用户组**。
   在IAM控制台创建用户，并将其加入**1**中创建的用户组。

须知

通过IAM用户使用CCE时，该IAM用户需要同时支持“编程访问”和“管理控制台访问”的访问方式。

3. 用户登录并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择云容器引擎，进入CCE主界面尝试购买集群，如果无法无法成功操作（假设当前权限仅包含CCE ReadOnlyAccess），表示“CCE ReadOnlyAccess”已生效。
   - 在“服务列表”中选择除云容器引擎外（假设当前策略仅包含CCE ReadOnlyAccess）的任一服务，若提示权限不足，表示“CCE ReadOnlyAccess”已生效。

系统角色

角色是IAM最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。

IAM中预置的CCE系统角色为**CCE Administrator**，给用户组授予该系统角色权限时，必须同时勾选该角色依赖的其他策略才会生效，例如Tenant Guest、Server Administrator、ELB Administrator、OBS Administrator、SFS Administrator、SWR Admin、APM FullAccess。了解更多角色依赖关系，请参考：系统权限。

系统策略

IAM中预置的CCE系统策略当前包含**CCE FullAccess**和**CCE ReadOnlyAccess**两种策略：

- **CCE FullAccess**：系统策略，CCE服务集群相关资源的普通操作权限，不包括集群（启用Kubernetes RBAC鉴权）的命名空间权限，不包括委托授权、生成集群证书等管理员角色的特权操作。
- **CCE ReadOnlyAccess**：系统策略，CCE服务集群相关资源的只读权限，不包括集群（启用Kubernetes RBAC鉴权）的命名空间权限。

说明

购买包周期集群、节点时，需要为用户添加自定义策略，额外配置费用中心服务的支付相关权限，如bss:*:*。

表 16-11 CCE FullAccess 策略主要权限

操作（Action）	Action详情	说明
cce:*:*	cce:cluster:create	创建集群
	cce:cluster:delete	删除集群
	cce:cluster:update	更新集群，如后续允许集群支持RBAC，调度参数更新等

操作（Action）	Action详情	说明
	cce:cluster:upgrade	升级集群
	cce:cluster:start	唤醒集群
	cce:cluster:stop	休眠集群
	cce:cluster:list	查询集群列表
	cce:cluster:get	查询集群详情
	cce:node:create	添加节点
	cce:node:delete	删除节点/批量删除节点
	cce:node:update	更新节点，如更新节点名称
	cce:node:get	查询节点详情
	cce:node:list	查询节点列表
	cce:nodepool:create	创建节点池
	cce:nodepool:delete	删除节点池
	cce:nodepool:update	更新节点池信息
	cce:nodepool:get	获取节点池
	cce:nodepool:list	列出集群的所有节点池
	cce:release:create	创建模板实例
	cce:release:delete	删除模板实例
	cce:release:update	更新升级模板实例
	cce:job:list	查询任务列表（集群层面的job）
	cce:job:delete	删除任务/批量删除任务（集群层面的job）
	cce:job:get	查询任务详情（集群层面的job）
	cce:storage:create	创建存储
	cce:storage:delete	删除存储
	cce:storage:list	列出所有磁盘
	cce:addonInstance:create	创建插件实例
	cce:addonInstance:delete	删除插件实例
	cce:addonInstance:update	更新升级插件实例

操作（Action）	Action详情	说明
	cce:addonInstance:get	获取插件实例
	cce:addonTemplate:get	获取插件模板
	cce:addonInstance:list	列出所有插件实例
	cce:addonTemplate:list	列出所有插件模板
	cce:chart:list	列出所有模板
	cce:chart:delete	删除模板
	cce:chart:update	更新模板
	cce:chart:upload	上传模板
	cce:chart:get	获取模板信息
	cce:release:get	获取模板实例信息
	cce:release:list	列出所有模板实例
	cce:userAuthorization:get	获取CCE用户授权
	cce:userAuthorization:create	创建CCE用户授权
ecs:*:*	-	ECS（弹性云服务器）服务的所有权限。
evs:*:*	-	EVS（云硬盘）的所有权限。 可以将云硬盘挂载到云服务器，并可以随时扩容云硬盘容量
vpc:*:*	-	VPC（虚拟私有云，包含二代ELB）的所有权限。 创建的集群需要运行在虚拟私有云中，创建命名空间时，需要创建或关联VPC，创建在命名空间的容器都运行在VPC之内。
bms:*:get*	-	BMS（裸金属服务器）所有资源详情的查看权限。
bms:*:list*	-	BMS（裸金属服务器）所有资源列表的查看权限。
ims:*:get*	-	IMS（镜像服务）所有资源详情的查看权限。
ims:*:list*	-	IMS（镜像服务）所有资源列表的查看权限。

操作（Action）	Action详情	说明
elb:*:get	-	ELB（弹性负载均衡）所有资源详情的查看权限。
elb:*:list	-	ELB（弹性负载均衡）所有资源列表的查看权限。
nat:*:get	-	NAT网关服务所有资源详情的查看权限。
nat:*:list	-	NAT网关服务所有资源列表的查看权限。
sfs:*:get*	-	SFS（弹性文件存储）所有资源详情的查看权限。
sfs:shares:ShareAction	-	SFS（弹性文件存储）资源的扩容共享。
sfsturbo:*:get*	-	SFS Turbo（极速弹性文件存储）服务所有资源详情的查看权限。
sfsturbo:shares:ShareAction	-	SFS Turbo（极速弹性文件存储）资源的扩容共享。
tms:resourceTags:list	-	TMS（标签管理服务）资源标签列表查看权限。
kps:domainKeypairs:list	-	DEW（数据加密服务）账号密钥对的SSH密钥列表查看权限。
kps:domainKeypairs:get	-	DEW（数据加密服务）账号密钥对的SSH密钥详情查看权限。
kms:cmk:get	-	DEW（数据加密服务）查看密钥信息权限。
kms:cmk:list	-	DEW（数据加密服务）查看密钥列表权限。
aom:*:get	-	AOM（应用运维管理）资源详情的查看权限。
aom:*:list	-	AOM（应用运维管理）资源列表的查看权限。
aom:autoScalingRule:*	-	AOM（应用运维管理）自动扩缩容规则的所有操作权限。
apm:icmgr:*	-	APM（应用性能管理服务）操作ICAgent权限。
lts:*:*	-	LTS（云日志服务）的所有权限。
smn:*:*	-	SMN（消息通知服务）的所有权限。

表 16-12 CCE ReadOnlyAccess 策略主要权限

操作（Action）	操作（Action）	说明
cce:*:get	cce:cluster:get	查询集群详情
	cce:node:get	查询节点详情
	cce:job:get	查询任务详情（集群层面的job）
	cce:addonInstance:get	获取插件实例
	cce:addonTemplate:get	获取插件模板
	cce:chart:get	获取模板信息
	cce:nodepool:get	获取节点池
	cce:release:get	获取模板实例信息
	cce:userAuthorization:get	获取CCE用户授权
cce:*:list	cce:cluster:list	查询集群列表
	cce:node:list	查询节点列表
	cce:job:list	查询任务列表（集群层面的job）
	cce:addonInstance:list	列出所有插件实例
	cce:addonTemplate:list	列出所有插件模板
	cce:chart:list	列出所有模板
	cce:nodepool:list	列出集群的所有节点池
	cce:release:list	列出所有模板实例
	cce:storage:list	列出所有磁盘
cce:kubernetes:*	-	操作所有Kubernetes资源，具体权限请在**命名空间权限**中配置。
ecs:*:get	-	ECS（弹性云服务器）所有资源详情的查看权限。 CCE中的一个节点就是具有多个云硬盘的一台弹性云服务器
ecs:*:list	-	ECS（弹性云服务器）所有资源列表的查看权限。
bms:*:get*	-	BMS（裸金属服务器）所有资源详情的查看权限。

操作（Action）	操作（Action）	说明
bms:*:list	-	BMS（裸金属服务器）所有资源列表的查看权限。
ims:*:get*	-	IMS（镜像服务）所有资源详情的查看权限。
ims:*:list*	-	IMS（镜像服务）所有资源列表的查看权限。
evs:*:get	-	EVS（云硬盘）所有资源详情的查看权限。可以将云硬盘挂载到云服务器，并可以随时扩容云硬盘容量
evs:*:list	-	EVS（云硬盘）所有资源列表的查看权限。
evs:*:count	-	-
vpc:*:get	-	VPC（虚拟私有云）所有资源详情的查看权限。 创建的集群需要运行在虚拟私有云中，创建命名空间时，需要创建或关联VPC，创建在命名空间的容器都运行在VPC之内
vpc:*:list	-	VPC（虚拟私有云）所有资源列表的查看权限。
elb:*:get	-	ELB（弹性负载均衡）所有资源详情的查看权限。
elb:*:list	-	ELB（弹性负载均衡）所有资源列表的查看权限。
nat:*:get	-	NAT网关服务所有资源详情的查看权限。
nat:*:list	-	NAT网关服务所有资源列表的查看权限。
sfs:*:get*	-	SFS（弹性文件存储）所有资源详情的查看权限。
sfs:shares:ShareAction	-	SFS（弹性文件存储）资源的扩容共享。
sfsturbo:*:get*	-	SFS Turbo（极速弹性文件存储）服务所有资源详情的查看权限。
sfsturbo:shares:ShareAction	-	SFS Turbo（极速弹性文件存储）资源的扩容共享。
tms:resourceTags:list	-	TMS（标签管理服务）资源标签列表查看权限。
kps:domainKeypairs:list	-	DEW（数据加密服务）账号密钥对的SSH密钥列表查看权限。

操作（Action）	操作（Action）	说明
kps:domainKeypairs:get	-	DEW（数据加密服务）账号密钥对的SSH密钥详情查看权限。
kms:cmk:get	-	DEW（数据加密服务）查看密钥信息权限。
kms:cmk:list	-	DEW（数据加密服务）查看密钥列表权限。
aom:*:get	-	AOM（应用运维管理）服务所有资源详情的查看权限。
aom:*:list	-	AOM（应用运维管理）服务所有资源列表的查看权限。
aom:autoScalingRule:*	-	AOM（应用运维管理）服务自动扩缩容规则的所有操作权限。
lts:*:get	-	LTS（云日志服务）的所有资源详情的查看权限。
lts:*:list	-	LTS（云日志服务）的所有资源列表的查看权限。
smn:*:get	-	SMN（消息通知服务）所有资源详情的查看权限。
smn:*:list	-	SMN（消息通知服务）所有资源列表的查看权限。

自定义策略

如果系统预置的CCE策略，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考**权限策略和授权项**。

目前支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。本章为您介绍常用的CCE自定义策略样例。

CCE自定义策略样例：

- 示例1：创建一个名称为“test”的集群

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cce:cluster:create"
```

```
        ]
      }
    ]
}
```

- 示例2：拒绝用户删除节点

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循**Deny优先原则**。

  如果您给用户授予CCEFullAccess的系统策略，但不希望用户拥有CCEFullAccess中定义的删除节点权限（cce:node:delete），您可以创建一条相同Action的自定义策略，并将自定义策略的Effect设置为Deny，然后同时将CCEFullAccess和拒绝策略授予用户，根据Deny优先原则，则用户可以对CCE执行除了删除节点外的所有操作。拒绝策略示例如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "cce:node:delete"
            ]
        }
    ]
}
```

- 示例3：多个授权项策略

  一个自定义策略中可以包含多个授权项，且除了可以包含本服务的授权项外，还可以包含其他服务的授权项，可以包含的其他服务必须跟本服务同属性，即都是项目级服务或都是全局级服务。多个授权语句策略描述如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "ecs:cloudServers:resize",
                "ecs:cloudServers:delete",
                "ecs:cloudServers:delete",
                "ims:images:list",
                "ims:serverImages:create"
            ],
            "Effect": "Allow"
        }
    ]
}
```

CCE 集群权限与企业项目

CCE支持以集群为粒度，基于企业项目维度进行资源管理以及权限分配。

如下事项需特别注意：

- IAM项目是基于资源的物理隔离进行管理，而企业项目则是提供资源的全局逻辑分组，更符合企业实际场景，并且支持基于企业项目维度的IAM策略管理，因此推荐您使用企业项目。详细信息请参见**如何创建企业项目**。
- IAM项目与企业项目共存时，IAM将优先匹配IAM项目策略、未决则匹配企业项目策略。
- CCE集群基于已有基础资源（VPC）创建集群、节点时，请确保IAM用户在已有资源的企业项目下有相关权限，否则可能导致集群或者节点创建失败。
- 当资源不支持企业项目时，为企业项目授予该资源的权限将不会生效。

是否支持企业项目	资源名称	说明
支持企业项目的资源	cluster	集群
	node	节点
	nodepool	节点池
	job	任务
	tag	集群标签
	addonInstance	插件实例
	release	Helm版本
	storage	存储资源
不支持企业项目的资源	quota	集群配额
	chart	模板
	addonTemplate	插件模板

CCE 集群权限与 IAM RBAC

CCE兼容IAM传统的系统角色进行权限管理，建议您切换使用IAM的细粒度策略，避免设置过于复杂或不必要的权限管理场景。

CCE当前支持的角色如下：

- IAM的基础角色：
  - te_admin（Tenant Administrator）：可以调用除IAM外所有服务的所有API。
  - readonly（Tenant Guest）：可以调用除IAM外所有服务的只读权限的API。
- CCE的自定义管理员角色：CCE Administrator。

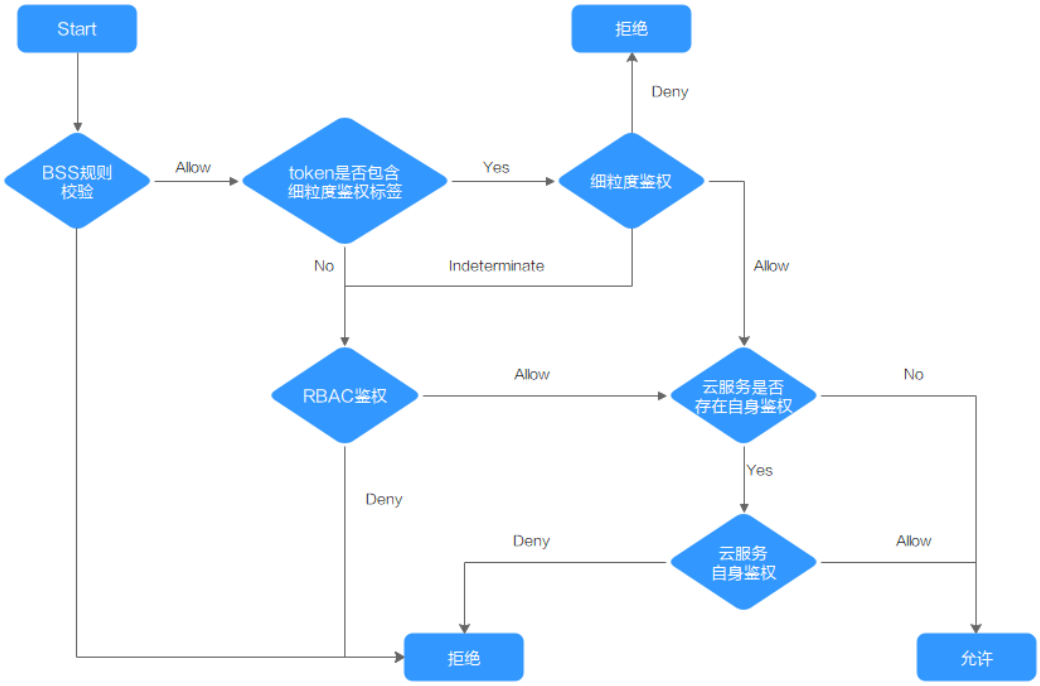
说明

如果用户有Tenant Administrator或者CCE Administrator的系统角色，则此用户拥有Kubernetes RBAC的cluster-admin权限，在集群创建后不可移除。

如果用户为集群创建者，则默认被授权Kubernetes RBAC的cluster-admin权限，此项权限可以在集群创建后被手动移除:

- 方式1：权限管理 - 命名空间权限 - 移除cluster-creator。
- 方式2：通过API或者kubectl删除资源，ClusterRoleBinding：cluster-creator。

RBAC与IAM策略共存时，CCE开放API或Console操作的后端鉴权逻辑如下：



16.3 命名空间权限（Kubernetes RBAC 授权）

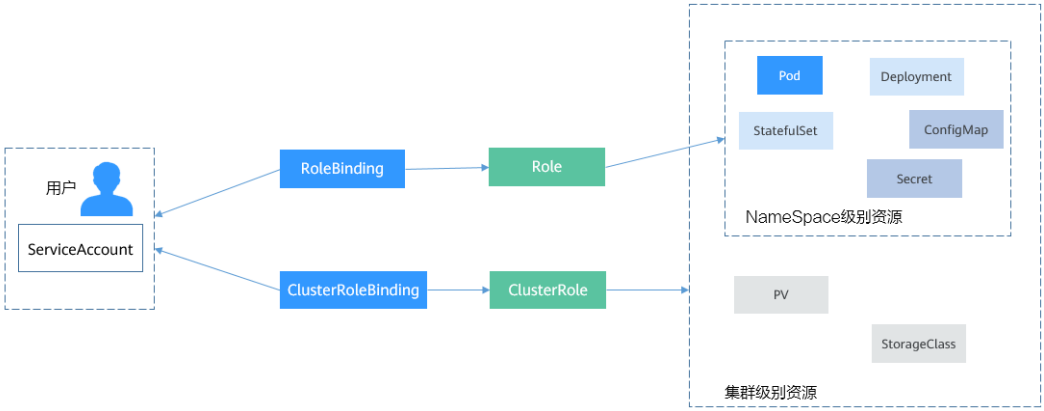
命名空间权限（kubernetes RBAC 授权）

命名空间权限是基于Kubernetes RBAC能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同Kubernetes资源的权限。Kubernetes RBAC API定义了四种类型：Role、ClusterRole、RoleBinding与ClusterRoleBinding，这四种类型之间的关系和简要说明如下：

- Role：角色，其实是定义一组对Kubernetes资源（命名空间级别）的访问规则。
- RoleBinding：角色绑定，定义了用户和角色的关系。
- ClusterRole：集群角色，其实是定义一组对Kubernetes资源（集群级别，包含全部命名空间）的访问规则。
- ClusterRoleBinding：集群角色绑定，定义了用户和集群角色的关系。

Role和ClusterRole指定了可以对哪些资源做哪些动作，RoleBinding和ClusterRoleBinding将角色绑定到特定的用户、用户组或ServiceAccount上。如下图所示。

图 16-3 角色绑定



在CCE控制台可以授予用户或用户组命名空间权限，可以对某一个命名空间或全部命名空间授权，CCE控制台默认提供如下ClusterRole。

- view（只读权限）：对全部或所选命名空间下大多数资源的只读权限。
- edit（开发权限）：对全部或所选命名空间下多数资源的读写权限。当配置在全部命名空间时能力与运维权限一致。
- admin（运维权限）：对全部命名空间下大多数资源的读写权限，对节点、存储卷，命名空间和配额管理的只读权限。
- cluster-admin（管理员权限）：对全部命名空间下所有资源的读写权限。
- drainage-editor：节点排水操作权限，可执行节点排水。
- drainage-viewer：节点排水只读权限，仅可查看节点排水状态，无法执行节点排水。
- geip-editor-role：该权限用于在集群中使用全域弹性公网IP资源，仅对资源geips具有读写权限。
- icagent-clusterRole：该权限用于生成Kubernetes事件日志上报到LTS。

除了使用上述常用的ClusterRole外，您还可以通过定义Role和RoleBinding来进一步对全局资源（如Node、PersistentVolumes、CustomResourceDefinitions等）和命名空间中不同类别资源（如Pod、Deployment、Service等）的增删改查权限进行配置，从而做到更加精细化的权限控制。

集群权限（IAM 授权）与命名空间权限（Kubernetes RBAC 授权）的关系

拥有不同集群权限（IAM授权）的用户，其拥有的命名空间权限（Kubernetes RBAC授权）不同。表16-13给出了不同用户拥有的命名空间权限详情。

表 16-13 不同用户拥有的命名空间权限

用户类型	1.13及以上版本的集群
拥有Tenant Administrator权限的用户（例如账号）	全部命名空间权限
拥有CCE Administrator权限的IAM用户	全部命名空间权限

用户类型	1.13及以上版本的集群
拥有CCE FullAccess或者CCE ReadOnlyAccess权限的IAM用户	按Kubernetes RBAC授权
拥有Tenant Guest权限的IAM用户	按Kubernetes RBAC授权

注意事项

- 任何用户创建集群后，CCE会自动为该用户添加该集群的所有命名空间的cluster-admin权限，也就是说该用户允许对集群以及所有命名空间中的全部资源进行完全控制。联邦用户由于每次登录注销都会改变用户ID，所以权限用户会显示已删除，此情况下请勿删除该权限，否则会导致鉴权失败。此种情况下建议在CCE为某个用户组创建cluster-admin权限，将联邦用户加入此用户组。
- 拥有Security Administrator（IAM除切换角色外所有权限）权限的用户（如账号所在的admin用户组默认拥有此权限），才能在CCE控制台命名空间权限页面进行授权操作。

配置命名空间权限（控制台）

CCE中的命名空间权限是基于Kubernetes RBAC能力的授权，通过权限设置可以让不同的用户或用户组拥有操作不同Kubernetes资源的权限。

步骤1 登录CCE控制台，在左侧导航栏中选择“权限管理”。

步骤2 在右边下拉列表中选择要添加权限的集群。

步骤3 在右上角单击“添加权限”，进入添加权限页面。

步骤4 在添加权限页面，确认集群名称，选择该集群下要授权使用的命名空间，例如选择“全部命名空间”，选择要授权的用户或用户组，再选择具体权限。

说明

对于没有IAM权限的用户，给其他用户和用户组配置权限时，无法选择用户和用户组，此时支持填写用户ID或用户组ID进行配置。

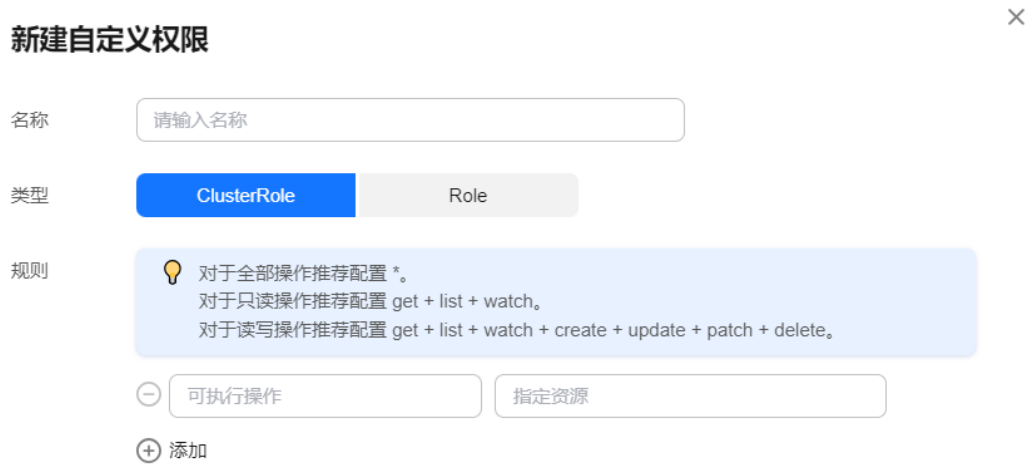
图 16-4 配置命名空间权限

其中自定义权限可以根据需要自定义，选择自定义权限后，在自定义权限一行右侧单击新建自定义权限，在弹出的窗口中填写名称并选择规则。创建完成后，在添加权限的自定义权限下拉框中可以选择。

自定义权限分为ClusterRole或Role两类，ClusterRole或Role均包含一组代表相关权限的规则，详情请参见**使用RBAC鉴权**。

- ClusterRole：ClusterRole是一个集群级别的资源，可设置集群的访问权限。
- Role：Role用于在某个命名空间内设置访问权限。当创建Role时，必须指定该Role所属的命名空间。

图 16-5 自定义权限



步骤5 单击“确定”。

----结束

自定义命名空间权限（kubectl）

说明

kubectl访问CCE集群是通过集群上生成的配置文件（kubeconfig.json）进行认证，kubeconfig.json文件内包含用户信息，CCE根据用户信息的权限判断kubectl有权限访问哪些Kubernetes资源。即哪个用户获取的kubeconfig.json文件，kubeconfig.json就拥有哪个用户的信息，这样使用kubectl访问时就拥有这个用户的权限。而用户拥有的权限就是**集群权限（IAM授权）与命名空间权限（Kubernetes RBAC授权）的关系**所示的权限。

除了使用cluster-admin、admin、edit、view这4个最常用的clusterrole外，您还可以通过定义Role和RoleBinding来进一步对命名空间中不同类别资源（如Pod、Deployment、Service等）的增删改查权限进行配置，从而做到更加精细化的权限控制。

Role的定义非常简单，指定namespace，然后就是rules规则。如下面示例中的规则就是允许对default命名空间下的Pod进行GET、LIST操作。

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default                    # 命名空间
  name: role-example
rules:
- apiGroups: [""]
```



```
resources: ["pods"]                    # 可以访问pod
verbs: ["get", "list"]                 # 可以执行GET、LIST操作
```

- apiGroups表示资源所在的API分组。
- resources表示可以操作哪些资源：pods表示可以操作Pod，其他Kubernetes的资源如deployments、configmaps等都可以操作
- verbs表示可以执行的操作：get表示查询一个Pod，list表示查询所有Pod。您还可以使用create（创建），update（更新），delete（删除）等操作词。

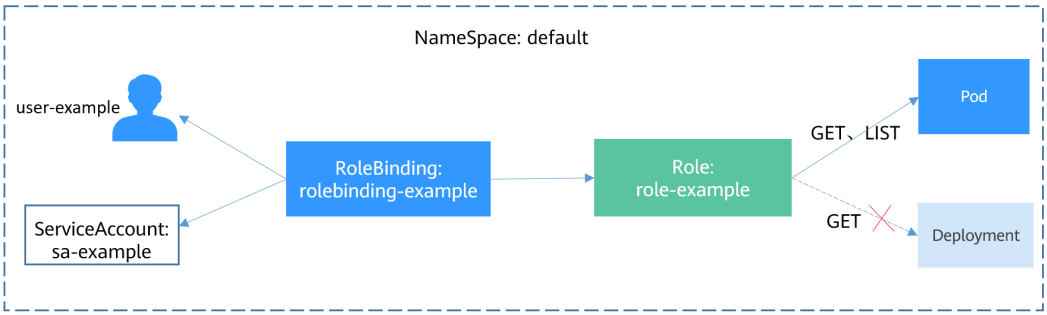
详细的类型和操作请参见**使用 RBAC 鉴权**。

有了Role之后，就可以将Role与具体的用户绑定起来，实现这个的就是RoleBinding了。如下所示。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: RoleBinding-example
  namespace: default
  annotations:
    CCE.com/IAM: 'true'
roleRef:
  kind: Role
  name: role-example
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: User
  name: 0c97ac3cb280f4d91fa7c0096739e1f8    # user-example的用户ID
  apiGroup: rbac.authorization.k8s.io
```

这里的subjects就是将Role与IAM用户绑定起来，从而使得IAM用户获取role-example这个Role里面定义的权限，如下图所示。

图 16-6 RoleBinding 绑定 Role 和用户



subjects下用户的类型还可以是用户组，这样配置可以对用户组下所有用户生效。

```
subjects:
- kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7    # 用户组ID
  apiGroup: rbac.authorization.k8s.io
```

使用IAM用户user-example连接集群，获取Pod信息，发现可获取到Pod的信息。

```
# kubectl get pod
NAME                                  READY   STATUS    RESTARTS   AGE
deployment-389584-2-6f6bd4c574-2n9rk  1/1     Running   0          4d7h
deployment-389584-2-6f6bd4c574-7s5qw  1/1     Running   0          4d7h
deployment-3895841-746b97b455-86g77   1/1     Running   0          4d7h
deployment-3895841-746b97b455-twvpn   1/1     Running   0          4d7h
nginx-658dff48ff-7rkph                1/1     Running   0          4d9h
nginx-658dff48ff-njdhj                1/1     Running   0          4d9h
```

```
# kubectl get pod nginx-658dff48ff-7rkph
NAME                     READY   STATUS    RESTARTS   AGE
nginx-658dff48ff-7rkph   1/1     Running   0          4d9h
```

然后查看Deployment和Service，发现没有权限；再查询kube-system命名空间下的Pod信息，发现也没有权限。这就说明IAM用户user-example仅拥有default这个命名空间下GET和LIST Pod的权限，与前面定义的没有偏差。

```
# kubectl get deploy
Error from server (Forbidden): deployments.apps is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "deployments" in API group "apps" in the namespace "default"
# kubectl get svc
Error from server (Forbidden): services is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "services" in API group "" in the namespace "default"
# kubectl get pod --namespace=kube-system
Error from server (Forbidden): pods is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "pods" in API group "" in the namespace "kube-system"
```

示例：授予集群管理员权限（cluster-admin）

集群全部权限可以使用cluster-admin权限，cluster-admin包含集群所有资源的权限。

图 16-7 授予集群管理员权限（cluster-admin）

如果使用kubectl查看可以看到创建了一个ClusterRoleBinding，将cluster-admin和cce-role-group这个用户组绑定了起来。

```
# kubectl get clusterrolebinding
NAME                                                        ROLE                        AGE
clusterrole_cluster-admin_group0c96fad22880f32a3f84c009862af6f7   ClusterRole/cluster-admin      61s

# kubectl get clusterrolebinding clusterrole_cluster-admin_group0c96fad22880f32a3f84c009862af6f7 -oyaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    CCE.com/IAM: "true"
  creationTimestamp: "2021-06-23T09:15:22Z"
  name: clusterrole_cluster-admin_group0c96fad22880f32a3f84c009862af6f7
  resourceVersion: "36659058"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/clusterrole_cluster-admin_group0c96fad22880f32a3f84c009862af6f7
  uid: d6cd43e9-b4ca-4b56-bc52-e36346fc1320
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
```

```
  kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7
```

使用被授予用户连接集群，如果能正常查询PV、StorageClass的信息，则说明权限配置正常。

```
# kubectl get pv
No resources found
# kubectl get sc
NAME                PROVISIONER                    RECLAIMPOLICY    VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION    AGE
csi-disk            everest-csi-provisioner        Delete           Immediate              true                   75d
csi-disk-topology   everest-csi-provisioner        Delete           WaitForFirstConsumer   true                   75d
csi-nas             everest-csi-provisioner        Delete           Immediate              true                   75d
csi-obs             everest-csi-provisioner        Delete           Immediate              false                  75d
csi-sfsturbo        everest-csi-provisioner        Delete           Immediate              true                   75d
```

示例：授予命名空间运维权限（admin）

admin权限拥有命名空间大多数资源的读写权限，您可以授予用户/用户组全部命名空间admin权限。

图 16-8 授予全部命名空间运维权限（admin）

如果使用kubectl查看可以看到创建了一个RoleBinding，将admin和cce-role-group这个用户组绑定了起来。

```
# kubectl get rolebinding
NAME                                                    ROLE                AGE
clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7   ClusterRole/admin   18s
# kubectl get rolebinding clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7 -oyaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    CCE.com/IAM: "true"
  creationTimestamp: "2021-06-24T01:30:08Z"
  name: clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7
  resourceVersion: "36963685"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/
clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7
  uid: 6c6f46a6-8584-47da-83f5-9eef1f7b75d6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7
```

使用被授予用户连接集群，如果能正常查询PV、StorageClass的信息，但无法创建命名空间，则说明权限配置正常。

```
# kubectl get pv
No resources found
# kubectl get sc
NAME                PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
csi-disk            everest-csi-provisioner        Delete          Immediate              true                   75d
csi-disk-topology   everest-csi-provisioner        Delete          WaitForFirstConsumer   true                   75d
csi-nas             everest-csi-provisioner        Delete          Immediate              true                   75d
csi-obs             everest-csi-provisioner        Delete          Immediate              false                  75d
csi-sfsturbo        everest-csi-provisioner        Delete          Immediate              true                   75d
# kubectl apply -f namespaces.yaml
Error from server (Forbidden): namespaces is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot
create resource "namespaces" in API group "" at the cluster scope
```

示例：授予命名空间开发权限（edit）

edit权限拥有命名空间大多数资源的读写权限，您可以授予用户/用户组全部命名空间edit权限。

图 16-9 授予 default 命名空间开发权限（edit）

如果使用kubectl查看可以看到创建了一个RoleBinding，将edit和cce-role-group这个用户组绑定了起来，且权限范围是default这个命名空间。

```
# kubectl get rolebinding
NAME                                                ROLE                AGE
clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7   ClusterRole/admin   18s
# kubectl get rolebinding clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7 -oyaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    CCE.com/IAM: "true"
  creationTimestamp: "2021-06-24T01:30:08Z"
  name: clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7
  namespace: default
  resourceVersion: "36963685"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/
clusterrole_admin_group0c96fad22880f32a3f84c009862af6f7
  uid: 6c6f46a6-8584-47da-83f5-9eef1f7b75d6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
- apiGroup: rbac.authorization.k8s.io
```

```
  kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7
```

使用被授予用户连接集群，您会发现可以查询和创建default命名空间的资源，但无法查询kube-system命名空间资源，也无法查询集群级别的资源。

```
# kubectl get pod
NAME                READY   STATUS    RESTARTS   AGE
test-568d96f4f8-brdrp   1/1     Running   0          33m
test-568d96f4f8-cgjqp   1/1     Running   0          33m
# kubectl get pod -nkube-system
Error from server (Forbidden): pods is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "pods" in API group "" in the namespace "kube-system"
# kubectl get pv
Error from server (Forbidden): persistentvolumes is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "persistentvolumes" in API group "" at the cluster scope
```

示例：授予命名空间只读权限（view）

view权限拥有命名空间查看权限，您可以给某个或全部命名空间授权。

图 16-10 授予 default 命名空间只读权限（view）

如果使用kubectl查看可以看到创建了一个RoleBinding，将view和cce-role-group这个用户组绑定了起来，且权限范围是default这个命名空间。

```
# kubectl get rolebinding
NAME                                                ROLE               AGE
clusterrole_view_group0c96fad22880f32a3f84c009862af6f7   ClusterRole/view   7s

# kubectl get rolebinding clusterrole_view_group0c96fad22880f32a3f84c009862af6f7 -oyaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    CCE.com/IAM: "true"
  creationTimestamp: "2021-06-24T01:36:53Z"
  name: clusterrole_view_group0c96fad22880f32a3f84c009862af6f7
  namespace: default
  resourceVersion: "36965800"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/
clusterrole_view_group0c96fad22880f32a3f84c009862af6f7
  uid: b86e2507-e735-494c-be55-c41a0c4ef0dd
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: 0c96fad22880f32a3f84c009862af6f7
```

使用被授予用户连接集群，您会发现可以查询default命名空间的资源，但无法创建资源。

```
# kubectl get pod
NAME                 READY   STATUS    RESTARTS   AGE
test-568d96f4f8-brdrp   1/1     Running   0          40m
test-568d96f4f8-cgjqp   1/1     Running   0          40m
# kubectl run -i --tty --image tutum/dnsutils dnsutils --restart=Never --rm /bin/sh
Error from server (Forbidden): pods is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot create resource "pods" in API group "" in the namespace "default"
```

示例：授予某类 Kubernetes 资源权限

上面几个示例都是集群全部资源（cluster-admin）、命名空间全部资源（admin、view），也可以对某类Kubernetes资源授权，如Pod、Deployment、Service这些资源，具体请参见**自定义命名空间权限（kubectl）**。

16.4 示例：某部门权限设计及配置

概述

随着容器技术的快速发展，原有的分布式任务调度模式正在被基于Kubernetes的技术架构所取代。云容器引擎（Cloud Container Engine，简称CCE）是高度可扩展的、高性能的企业级Kubernetes集群，支持社区原生应用和工具。借助云容器引擎，您可以在云上轻松部署、管理和扩展容器化应用程序，快速高效的将微服务部署在云端。

为方便企业中的管理人员对集群中的资源权限进行管理，CCE后台提供了多种维度的细粒度权限策略和管理方式。CCE的权限管理包括“集群权限”和“命名空间权限”两种能力，分别从集群和命名空间两个层面对用户组或用户进行细粒度授权，具体解释如下：

- **集群权限：**是基于IAM系统策略的授权，可以让用户组拥有“集群管理”、“节点管理”、“节点池管理”、“模板市场”、“插件管理”权限。
- **命名空间权限：**是基于Kubernetes RBAC能力的授权，可以让用户或用户组拥有Kubernetes资源的权限，如“工作负载”、“网络管理”、“存储管理”、“命名空间”等的权限。

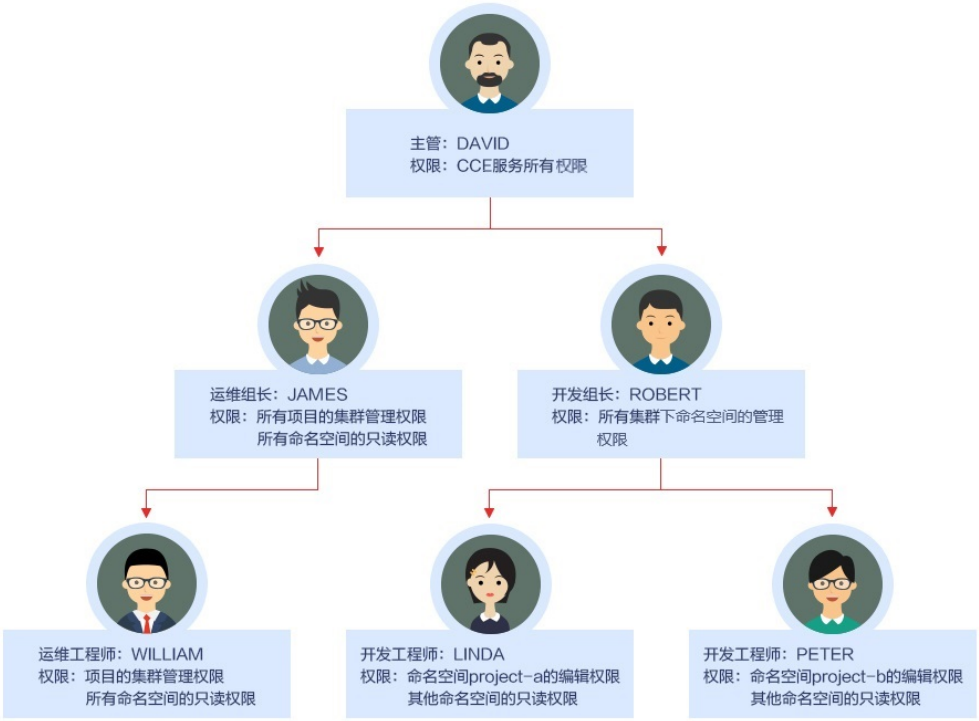
基于IAM系统策略的“集群权限”与基于Kubernetes RBAC能力的“命名空间权限”，两者是完全独立的，互不影响，但要配合使用。同时，为用户组设置的权限将作用于用户组下的全部用户。当给用户或用户组添加多个权限时，多个权限会同时生效（取并集）。

权限设计

下面以一个公司为例进行介绍。

通常一个公司中有多个部门或项目，每个部门又有多个成员，所以在配置权限前需要先进行详细设计，并在设置权限之前提前为每个成员创建用户名，便于后续对用户进行用户组归属和权限设置。

下图为某公司某部门的组织架构图和相关人员的权限设计，本文将按照该设计对每个角色的权限设置进行演示：



主管：DAVID

用户“DAVID”为该公司某部门的主管，根据权限设计需要为其配置CCE服务的所有权限（包括集群权限和命名空间权限），因此需要在统一身份认证服务 IAM中单独为DAVID创建用户组“cce-admin”，并配置所有项目的权限：“CCE Administrator”，这样主管DAVID的权限就配置好了。

说明

CCE Administrator：CCE的管理员权限，拥有该服务的所有权限，不需要再赋予其他权限。

CCE FullAccess、CCE ReadOnlyAccess：CCE的集群管理权限，仅针对与集群相关的资源（如集群、节点）有效，您必须确保同时配置了“命名空间权限”，才能有操作Kubernetes资源（如工作负载、Service等）的权限。

图 16-11 为主管 DAVID 所在的用户组授权

运维组长：JAMES

用户“JAMES”为该部门的运维组长，需要设置所有项目的集群权限和所有命名空间的只读权限。

在统一身份认证服务 IAM中先为用户“JAMES”单独创建并加入用户组“cce-sre”，然后为用户组“cce-sre”配置所有项目的集群权限：“CCE FullAccess”，用户组“cce-sre”便拥有了所有项目的集群管理权限，接下来还需要为其设置命名空间的只读权限。

图 16-12 为运维组长 JAMES 所在的用户组授权

为所有组长和工程师添加所有集群和命名空间的只读权限

在统一身份认证服务 IAM中再创建一个只读用户组“read_only”，然后将相关用户都添加到此用户组中。

- 两个开发工程师虽然不需要配置集群的管理权限，但也需要查看CCE控制台，因此需要有集群的只读权限才能满足需求。
- 运维工程师需要某区域集群的管理权限，为方便管理，这里先为其赋予集群的只读权限。
- 运维组长已经拥有了所有集群的管理权限，为方便管理，也可以将其添加到“read_only”用户组中，为其赋予集群的只读权限。

将JAMES、ROBERT、WILLIAM、LINDA、PETER五个用户都添加到用户组“read_only”中。

图 16-13 将相关用户添加到“read_only”用户组中

接下来为用户组“read_only”赋予集群的只读权限。

图 16-14 为用户组赋予集群的只读权限

然后返回CCE控制台，为这五个用户所在的用户组“read_only”增加命名空间的只读权限，单击左侧栏目树中的“权限管理”，为用户组“read_only”逐个赋予所有集群的只读权限。

图 16-15 为用户组赋予命名空间的只读权限

设置完成后，运维组长“JAMES”就拥有了所有项目的集群管理权限和所有命名空间的只读权限，而开发组长“ROBERT”、运维工程师“WILLIAM”以及两位开发工程师“LINDA”和“PRTER”则拥有了所有集群和命名空间的只读权限。

开发组长：ROBERT

用户“ROBERT”作为开发组的组长，虽然在上一步中已经为其设置了所有集群和命名空间的只读权限，但显然还不够，还需要为其设置所有命名空间的管理权限。

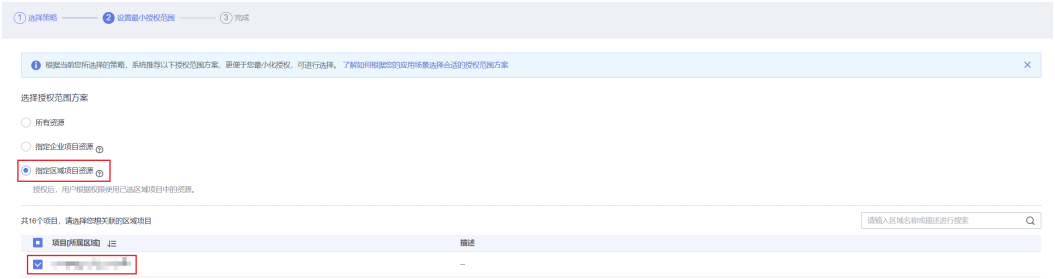
因此需要再单独为其赋予所有集群下全部命名空间的管理员权限。

图 16-16 为用户 ROBERT 赋予命名空间的管理权限

运维工程师：WILLIAM

运维工程师“WILLIAM”虽然也有了所有集群和命名空间的只读权限，但还需要在统一身份认证服务 IAM中为其设置区域的集群管理权限，因此单独为其创建一个用户组“cce-sre-b4”，然后配置区域项目的“CCE FullAccess”。

图 16-17 为用户 WILLIAM 所在的用户组配置北京四项目的集群管理权限



由于之前已经为其设置过所有命名空间的只读权限，所以运维工程师“WILLIAM”现在就拥有了区域的集群管理权限和所有命名空间的只读权限。

开发工程师：LINDA、PETER

“LINDA”和“PETER”是开发工程师，由于前面已经在用户组“read-only”中为两位工程师配置了集群和命名空间的只读权限，这里只需要再另外配置相应命名空间的编辑权限即可。

图 16-18 为开发工程师分别配置命名空间的编辑权限

至此，该部门的所有权限就设置完成了。

16.5 CCE 控制台的权限依赖

CCE对其他云服务有诸多依赖关系，因此在您开启IAM授权后，在CCE Console控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，详细说明如下：

- 依赖服务的权限配置均基于您已设置了IAM授权的CCE FullAccess或CCE ReadOnlyAccess策略权限，详细设置方法请参见**集群权限（IAM授权）**。
- 集群显示情况依赖于命名空间权限的设置情况，如果没有设置命名空间权限，则无法查看集群下的资源。

- 如果您设置了全部命名空间的view权限，则可以查看到对应集群的全部命名空间下的资源，但密钥（Secret）除外，密钥（Secret）需要在命名空间权限下设置admin或者edit权限才能查看。
- 如果您设置的是单一命名空间的view权限，则看到的只能是指定命名空间下的资源。

依赖服务的权限设置

如果IAM用户需要在CCE Console控制台拥有相应功能的查看或使用权限，请确认已经对该用户所在的用户组设置了CCE Administrator、CCE FullAccess或CCE ReadOnlyAccess策略的集群权限，再按如下表16-14增加依赖服务的角色或策略。

说明

企业项目能够实现企业不同项目间资源的分组和管理，重在资源隔离，而IAM可以实现细粒度授权，因此强烈推荐您使用IAM实现权限管理。

若您使用企业项目设置子用户权限，会有如下功能限制：

- 在CCE控制台，集群监控获取AOM监控的接口暂不支持企业项目，因此企业项目子用户将无法查看监控相关数据。
- 在CCE控制台，由于创建节点时的密钥对查询接口不支持企业项目，因此企业项目子用户将无法使用“密钥对”登录方式，您可以选择使用“密码”登录方式。
- 在CCE控制台，由于创建模板时不支持企业项目，因此企业项目子用户将无法使用模板管理。
- 在CCE控制台，由于云硬盘查询接口不支持企业项目，因此企业项目子用户将无法使用已有云硬盘创建PV。如需使用，需要为IAM用户添加evs:volumes:get的细粒度权限。

CCE支持细粒度的权限设置，但有如下限制说明：

- AOM不支持资源级别细粒度：当通过IAM集群资源细粒度设置特定资源操作权限之后，IAM用户在CCE控制台的总览界面查看集群监控时，将显示非细粒度关联集群的监控信息。

表 16-14 CCE Console 中依赖服务的角色或策略

Console控制台功能	依赖服务	需配置角色/策略
集群信息总览	应用运维管理 AOM	• IAM用户设置了CCE Administrator权限后，需要增加AOM FullAccess权限后才能访问总览中的数据图表。 • 支持设置了IAM ReadOnlyAccess和CCE FullAccess或CCE ReadOnlyAccess权限的IAM用户直接访问总览中的数据图表。

Console控制台功能	依赖服务	需配置角色/策略
工作负载	弹性负载均衡 ELB 应用性能管理 APM 应用运维管理 AOM NAT网关 NAT 对象存储服务 OBS 弹性文件服务 SFS	正常创建工作负载时不依赖其他服务的权限。 ● 如果需要创建ELB类型的服务，需要设置ELB FullAccess或者ELB Administrator权限，以及VPC Administrator权限。 ● 如果需要使用Java探针，需要设置AOM FullAccess和APM FullAccess权限。 ● 如果需要NAT网关类型的服务，需要设置NAT Gateway Administrator权限。 ● 如果使用对象存储，需要全局设置OBS Administrator权限。 **说明** 由于缓存的存在，对用户、用户组以及企业项目授予OBS相关的RBAC策略后，大概需要等待13分钟RBAC策略才能生效；授予OBS相关的系统策略后，大概需要等待5分钟系统策略能生效。 ● 如果使用文件存储，需要设置SFS FullAccess权限。
集群管理	应用运维管理 AOM 费用中心 BSS	● 如果需要弹性扩容权限，需要设置AOM FullAccess权限。 ● 如果需要转包周期，需要设置BSS Administrator权限。
节点管理	弹性云服务器 ECS	当IAM用户权限为CCE Administrator时，如果创建和删除节点，需要配置ECS FullAccess或ECS Administrator权限，以及VPC Administrator权限。
服务	弹性负载均衡 ELB NAT网关 NAT	正常创建时不依赖其他服务的权限。 ● 如果需要创建ELB类型的服务，需要设置ELB FullAccess或者ELB Administrator权限，以及VPC Administrator权限。 ● 如果需要NAT网关类型的服务，需要设置NAT Administrator权限。

Console控制台功能	依赖服务	需配置角色/策略
存储	对象存储服务 OBS 弹性文件服务 SFS 极速文件存储 SFS Turbo	• 如果使用对象存储，需要全局设置OBS Administrator权限。 **说明** 由于缓存的存在，对用户、用户组以及企业项目授予OBS相关的RBAC策略后，大概需要等待13分钟RBAC策略才能生效；授予OBS相关的系统策略后，大概需要等待5分钟系统策略能生效。 • 如果使用文件存储，需要设置SFS FullAccess权限。 • 如果使用极速文件存储，需要设置SFS Turbo FullAccess权限 导入存储的功能需要设置CCE Administrator权限。
命名空间	/	无需其他依赖权限。
模板市场	/	当前仅支持账号、设置了CCE Administrator权限的IAM用户访问。
插件中心	/	支持账号、设置了CCE Administrator、CCE FullAccess或CCE ReadOnlyAccess等权限的IAM用户访问本功能。
权限管理	/	• 支持账号访问。 • 支持设置了CCE Administrator和Security Administrator（全局级策略）权限的IAM用户访问。 • 支持设置了CCE FullAccess或CCE ReadOnlyAccess权限的IAM用户访问，同时还需要拥有命名空间的**管理员权限（cluster-admin）**。
配置与密钥	/	• 配置项（ConfigMap）无需其他依赖权限。 • 密钥（Secret）需要在命名空间权限下设置cluster-admin、admin或者edit权限才能查看，依赖服务需要添加DEW KeypairFullAccess或者DEW KeypairReadOnlyAccess权限。
帮助中心	/	无需其他依赖权限。
其他服务跳转	容器镜像服务 SWR 云日志服务 LTS 多云容器平台 MCP	为便于您快速进入CCE相关服务的控制台，在CCE控制台增加了其他服务的跳转链接，CCE默认没有这些服务的全部权限，如果IAM用户需要查看或使用其功能，请按照该服务的权限策略说明设置相应的权限策略。

16.6 ServiceAccount Token 安全性提升说明

Kubernetes 1.21以前版本的集群中，Pod中获取Token的形式是通过挂载ServiceAccount的Secret来获取Token，这种方式获得的Token是永久的。该方式在1.21及以上的版本中不再推荐使用，并且根据社区版本迭代策略，在1.25及以上版本的集群中，ServiceAccount将不会自动创建对应的Secret。

Kubernetes 1.21及以上版本的集群中，直接使用**TokenRequest** API获得Token，并使用投射卷（Projected Volume）挂载到Pod中。使用这种方法获得的Token具有固定的生命周期（默认有效期为1小时），在到达有效期之前，Kubelet会刷新该Token，保证Pod始终拥有有效的Token，并且当挂载的Pod被删除时这些Token将自动失效。该方式通过**BoundServiceAccountTokenVolume**特性实现，能够提升服务账号（ServiceAccount）Token的安全性，Kubernetes 1.21及以上版本的集群中会默认开启。

为了帮助用户平滑过渡，社区默认将Token有效时间延长为1年，1年后Token失效，不具备证书reload能力的client将无法访问APIServer，建议使用低版本Client的用户尽快升级至高版本，否则业务将存在故障风险。

如果用户使用版本过低的Kubernetes客户端（Client），由于低版本Client并不具备证书轮转能力，会存在证书轮转失效的风险。K8s社区默认具有证书轮转能力的Client版本如下：

- Go: >= v0.15.7
- Python: >= v12.0.0
- Java: >= v9.0.0
- Javascript: >= v0.10.3
- Ruby: master branch
- Haskell: v0.3.0.0
- C#: >= 7.0.5

官方说明请参见：**https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/1205-bound-service-account-tokens**

说明

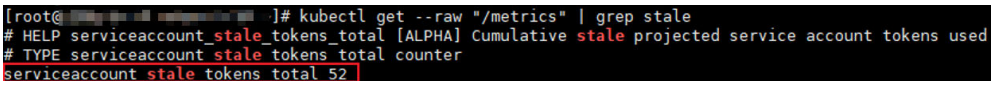
如果您在业务中需要一个永不过期的Token，您也可以选择**手动管理ServiceAccount的Secret**。尽管存在手动创建永久ServiceAccount Token的机制，但还是推荐使用**TokenRequest**的方式使用短期的Token，以提高安全性。

排查方案

CCE提供以下排查方式供用户参考（CCE 1.21及以上版本的集群均涉及）：

1. 排查集群中使用的插件版本。
   - 若用户集群中有使用2.23.34及以下版本Prometheus 插件，则需升级至2.23.34以上版本。
   - 若用户集群中有使用1.15.0及以下版本npd插件，则需升级至最新版本。
2. 通过kubectl连接集群，并通过**kubectl get --raw "/metrics" | grep stale**查询，可以看到一个名为serviceaccount_stale_tokens_total的指标。

如果该值大于0，则表示当前集群可能存在某些负载正在使用过低的client-go版本情况，此时请您排查自己部署的应用中是否有该情况出现。如果存在，则尽快将client-go版本升级至社区指定的版本之上（至少不低于CCE集群的两个大版本，如部署在1.23集群上的应用需要使用1.19版本以上的Kubernetes依赖库）。



16.7 系统委托说明

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

- 计算类服务

  CCE集群创建节点时会关联创建云服务器，因此需要获取访问弹性云服务器、裸金属服务器的权限。

- 存储类服务

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

- 网络类服务

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

- 容器与监控类服务

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

当您同意授权后，CCE将在IAM中自动创建账号委托，将账号内的其他资源操作权限委托给华为云CCE服务进行操作。关于资源委托详情，您可参考**委托**进行了解。

CCE自动创建的委托如下：

- **cce_admin_trust**：具有除IAM管理外的全部云服务管理员权限，用于调用CCE依赖的其他云服务资源。
- **cce_cluster_agency**：仅包含CCE组件依赖的云服务资源操作权限，用于生成CCE集群中组件使用的临时访问凭证。

cce_admin_trust 委托说明

cce_admin_trust委托具有Tenant Administrator权限。Tenant Administrator拥有除IAM管理外的全部云服务管理员权限，用于对CCE所依赖的其他云服务资源进行调用，例如创建集群、节点等场景，且该授权仅在当前区域生效。

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

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

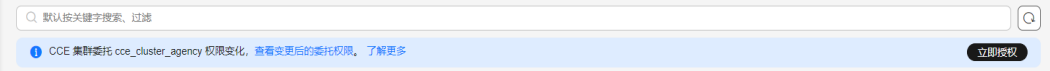
cce_cluster_agency 委托说明

cce_cluster_agency委托没有Tenant Administrator系统角色，只包含CCE组件需要的云服务资源操作权限，用于生成CCE集群中组件使用的临时访问凭证。在集群中自动创建其他相关云服务的资源时将使用该委托权限，例如创建Ingress、创建动态存储卷等场景。

说明

- cce_cluster_agency委托仅支持1.21及以上版本新建的集群。
- 创建cce_cluster_agency委托时将会自动创建名为“CCE cluster policies”的自定义策略，请勿删除该策略。

若当前cce_cluster_agency委托的权限与CCE期望的权限不同时，控制台会提示权限变化，需要您重新授权。



以下场景中，可能会出现cce_cluster_agency委托重新授权：

- CCE组件依赖的权限可能会随版本变动而发生变化。例如新增组件需要依赖新的权限，CCE将会更新期望的权限列表，此时需要您重新为cce_cluster_agency委托授权。
- 当您手动修改了cce_cluster_agency委托的权限时，该委托中拥有的权限与CCE期望的权限不相同，此时也会出现重新授权的提示。若您重新进行授权，该委托中手动修改的权限可能会失效。

17 配置中心

17.1 集群配置概览

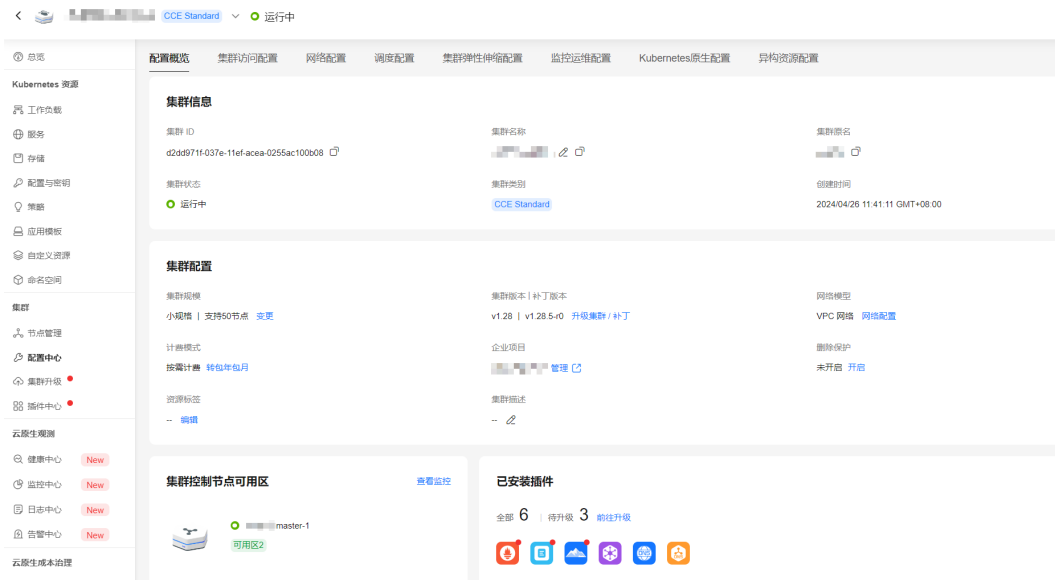
集群配置中心为您提供集群基础配置的概况及对应的修改入口，包含**集群信息**、**集群配置**、**集群控制节点可用区**和**已安装插件**多维度的信息概况。

功能入口

步骤1 登录CCE控制台，单击集群名称进入集群详情页。

步骤2 在左侧导航栏中选择“配置中心”，单击“配置概览”页签。

图 17-1 配置概览



----结束

集群信息

集群信息包括多个维度：

- **集群ID**：集群资源唯一标识，创建成功后自动生成，可用于API接口调用等场景。
- **集群名称**：集群创建成功后可以单击 进行修改。
- **集群原名**：修改集群名称后显示的集群原名，设置其他集群的名称时和该集群的原名同样不可以重复。
- **集群状态**：当前集群的运行状态，详情请参见**集群生命周期**。
- **集群类别**：显示当前集群为CCE Standard集群或CCE Turbo集群，两者差异请参见集群类型对比。
- **集群创建时间**：显示集群创建的时间，CCE以该时间作为计费依据。

集群配置

集群创建完成后，可修改的基本配置如下：

- **集群规模**：集群规模为集群管理的最大节点数量，请根据实际业务需求进行调整，详情请参见**变更集群规格**。
- **集群版本 | 补丁版本**：显示当前集群对应的Kubernetes版本号以及CCE的补丁版本号。
- **网络模型**：显示当前集群的网络模型，不支持修改。关于各网络模型介绍，请参见**容器网络**
- **计费模式**：显示当前集群的计费模式。按需计费模式支持转包周期计费，详情请参见**按需计费集群转包周期**。
- **企业项目**：显示集群所属的企业项目。了解更多企业项目相关信息，请查看**企业管理**。
- **删除保护**：开启删除保护后，需通过虚拟MFA、手机短信或邮箱等再次确认当前操作。请前往“IAM 服务 > 安全设置 > 敏感操作”开启操作保护。
- **资源标签**：对资源进行自定义标记，实现资源的分类。
- **集群描述**：添加自定义的集群描述，支持200个英文字符。
- **禁止集群删除**：防止通过控制台或API误删除集群，开启后将禁止删除或退订集群。

集群控制节点可用区

您可查看集群控制节点数量，如果需要查看控制节点资源使用率等数据，请单击右上角“查看监控”，前往**监控中心**页面查看。

已安装插件

您可查看集群中已安装的插件，当集群中存在可以升级的插件时，请单击“前往升级”，在插件中心页面进行查看。

17.2 集群访问配置

访问方式

- **kubectl**：您需要先下载kubectl以及kubeconfig配置文件，完成配置后，即可以使用kubectl访问Kubernetes集群。详情请参见**通过kubectl连接集群**。

- **公网地址**：为Kubernetes集群的API Server绑定弹性公网IP。配置完成后，集群API Server将具有公网访问能力。

  说明

  - 绑定弹性公网IP后，集群存在公网访问风险，建议为控制节点安全组加固5443端口的入方向规则。详情请参见**如何配置集群的访问策略**。
  - 此操作将会短暂重启 kube-apiserver 并更新集群访问证书（kubeconfig），请避免在此期间操作集群。

- **自定义 SAN**：主题备用名称（SAN）允许将多种值（包括IP地址、域名等）与证书关联，在集群访问证书中签入自定义SAN后，可以通过SAN定义的域名或IP访问集群。详情请参见**通过自定义域名访问集群**。

  说明

  此操作将会短暂重启 kube-apiserver 并更新集群访问证书（kubeconfig），请避免在此期间操作集群。

认证鉴权

CCE支持下载X509证书，证书中包含client.key、client.crt、ca.crt三个文件，请妥善保管您的证书，不要泄露。

如需使用证书访问集群，请参考**通过X509证书连接集群**。

服务端请求处理配置

表 17-1 服务端请求处理配置参数说明

名称	参数	说明	取值
修改类API请求最大并发数	max-mutating-requests-inflight	修改类请求的最大并发数。 当服务器收到的请求数超过此值时，它会拒绝请求。 0表示无限制。该参数与集群规模相关，不建议修改。	从v1.21版本开始不再支持手动配置，根据集群规格自动配置如下： • 50和200节点：200 • 1000节点：500 • 2000节点：1000

名称	参数	说明	取值
非修改类API请求最大并发数	max-requests-inflight	非修改类请求的最大并发数。当服务器收到的请求数超过此值时，它会拒绝请求。 0表示无限制。该参数与集群规模相关，不建议修改。	从v1.21版本开始不再支持手动配置，根据集群规格自动配置如下： • 50和200节点：400 • 1000节点：1000 • 2000节点：2000
请求超时时间	request-timeout	kube-apiserver组件的默认请求超时时间，请谨慎修改此参数，确保取值合理性，以避免频繁出现接口超时或其他异常。 该参数仅v1.19.16-r30、v1.21.10-r10、v1.23.8-r10、v1.25.3-r10及以上版本集群支持。	默认： 1m0s 取值范围： min>=1s max<=1h
开启过载防护	support-overload	集群过载控制开关，开启后将根据控制节点的资源压力，动态调整请求并发量，维护控制节点和集群的可靠性。详情请参见**开启集群过载控制**。 该参数仅v1.23及以上版本集群支持。 **说明** 开启过载防护功能不意味着绝对不会过载，极端场景如短时内请求量急剧冲高超出过载调整反应速度时，仍可能有过载现象出现，建议您针对集群访问行为进行主动管控，避免此类极端场景。	-

17.3 网络配置

网络配置支持为您的集群配置节点默认安全组，扩展容器网段等。

集群网络配置

表 17-2 集群网络配置参数说明

参数名称	参数说明
虚拟私有云	显示集群所在虚拟私有云。 虚拟私有云（Virtual Private Cloud，简称VPC）可以为云服务器、云容器、云数据库等资源构建隔离的、用户自主配置和管理的虚拟网络环境。您可以自由配置VPC内的IP地址段、子网、安全组等子服务，也可以申请弹性带宽和弹性公网IP搭建业务系统。
虚拟私有云网段	显示虚拟私有云网段。 如需扩展虚拟私有云网段，请参见**扩展集群VPC网段**。
默认节点子网	显示集群的节点子网。 子网是用来管理弹性云服务器网络平面的一个网络，可以提供IP地址管理、DNS服务，子网内的弹性云服务器IP地址都属于该子网。 默认情况下，同一个VPC的所有子网内的弹性云服务器均可以进行通信，不同VPC的弹性云服务器不能进行通信。 不同VPC的弹性云服务器可通过VPC创建对等连接通信。
默认节点子网 | IPv4网段	显示集群的节点子网网段。
容器网络模型	显示集群的容器网络模型，集群创建成功后，网络模型不可更改。不同网络模型对比请参见**容器网络模型对比**。
节点默认安全组	显示集群节点默认安全组。您可以选择自定义的安全组作为集群默认的节点安全组，但是需要注意放通指定端口来保证正常通信。 如需要自定义配置安全组规则，修改后的安全组只作用于新创建的节点和新纳管的节点，存量节点需要手动修改节点安全组规则。

参数名称	参数说明
访问集群外地址时保留原有Pod IP的网段（VPC网络模型的集群支持）	VPC网络集群中默认将10.0.0.0/8、172.16.0.0/12、192.168.0.0/16这三个VPC私有网段视为集群私有网段。如果集群所在VPC使用了扩展网段，创建、重置节点等操作也会将扩展网段添加到集群私有网段中。 在Pod中发起访问请求时，如果Pod访问的目的地址在集群私有网段范围内，则节点不会将Pod IP进行网络地址转换，可以借由上层VPC直接将报文送达至目的地址，即直接使用Pod IP与集群私有网络地址进行通信。 v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本支持该配置。详情请参见**在VPC网络集群中访问集群外地址时使用Pod IP作为客户端源IP**。 **说明** 如果需要保证节点能正常访问跨节点的Pod，该参数中设置的网段必须包含节点的子网网段。 同理，如果同VPC下的其他ECS节点需要能正常访问Pod IP，该参数中设置的网段必须包含ECS所在子网网段。
Pod访问元数据（CCE Turbo集群支持）	是否允许集群中的Pod访问宿主机元数据，例如可用区、企业项目ID等信息，详情请参见**弹性云服务器元数据类型**。 v1.23.13-r0、v1.25.8-r0、v1.27.5-r0、v1.28.3-r0及以上版本支持该配置。 • 如果Pod在开关开启状态下创建，则该Pod能否访问元数据取决于开关状态。 • 如果Pod在开关关闭状态下创建，或是在历史版本集群中创建，则无论开关状态如何，Pod均无法访问元数据。如需访问元数据，需要在开关开启状态下重建Pod。

服务配置

表 17-3 服务配置参数说明

参数名称	参数说明
服务转发模式	显示集群的转发模式，集群创建完成后，服务转发模式不可修改。当前支持IPVS和iptables两种转发模式，具体请参见**iptables与IPVS如何选择**。
服务网段	集群中的每个Service都有自己的地址，在CCE上创建集群时，可以指定Service的地址段（即服务网段）。服务网段不能和子网网段重合，而且服务网段也不能和容器网段重叠。服务网段只在集群内使用，不能在集群外使用。

参数名称	参数说明
服务端口范围配置	NodePort端口范围，默认范围为30000-32767，支持的修改范围为20106-32767。修改后需前往安全组页面同步修改节点安全组30000-32767的TCP/UDP端口范围，否则除默认端口外的其他端口将无法被外部访问。 **说明** 端口号小于20106会和系统组件的健康检查端口冲突，引发集群不可用；端口号高于32767会和操作系统的随机端口冲突，影响性能。

容器网段配置（VPC 网络模型集群支持）

当创建集群时设置的容器网段太小，无法满足业务扩容需求时，您通过扩展集群容器网段的方法来解决，详情请参见**扩展集群容器网段**。

说明

- 仅支持v1.19及以上版本的“VPC网络”模型集群。
- 容器网段添加后无法删除，请谨慎操作。

自定义容器网络配置（CCE Turbo 集群支持）

如您期望不同的命名空间或工作负载使用不同的子网网段或安全组，可创建一条策略，将子网/安全组信息和命名空间或工作负载进行绑定，详情请参见**使用容器网络配置为命名空间/工作负载绑定子网及安全组**。

- 容器绑定子网：Pod IP会被约束在特定的网段中，不同命名空间或工作负载之间可实现网络隔离。
- 容器绑定安全组：支持为同一命名空间或工作负载的Pod设置安全组规则，实现自定义访问策略。

说明

仅CCE Turbo集群支持该配置。v1.23.17-r0、v1.25.12-r0、v1.27.9-r0、v1.28.7-r0、v1.29.3-r0及以上版本的集群支持删除容器子网。

集群容器网络预热配置（CCE Turbo 集群支持）

CCE Turbo集群会为每个Pod分配（申请并绑定）一张弹性网卡或辅助弹性网卡。容器场景下Pod支持极速弹性，而网卡创建绑定需要一定时间，影响了大规模批创场景下的容器启动速度。系统默认提供了容器网卡动态预热的能力，在尽可能提高IP的资源利用率的前提下，加快Pod的启动速度。集群预热配置为您的集群设置全局的预热策略，集群节点默认会根据集群预热配置选项进行容器网卡的预热。如您期望为一组节点设置独立的预热策略，建议您配置节点池预热。

说明

仅CCE Turbo集群支持该配置。

是否启用容器网络全预热：

- 开启：开启容器网络全预热后，您的集群节点会预热申请节点规格上限的网卡数，如s7.large.2机型的节点辅助弹性网卡上限是16个，则系统会动态预热出16个辅助弹性网卡。

- 关闭：不启用容器网络全预热时，您可以自行定义预热参数。

表 17-4 容器网卡动态预热参数

容器网卡动态预热参数	默认值	参数说明	配置建议
节点最少绑定容器网卡数(nic-minimum-target)	10	保障节点最少有多少张容器网卡绑定在节点上。 参数值需为正整数。例如10，表示节点最少有10张容器网卡绑定在节点上。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	建议配置为大部分节点平时日常运行的Pod数。
节点预热容器网卡上限检查值(nic-maximum-target)	0	当节点绑定的容器网卡数超过节点预热容器网卡上限检查值(nic-maximum-target)，不再主动预热容器网卡。 当该参数大于等于节点最少绑定容器网卡数(nic-minimum-target)时，则开启预热容器网卡上限值检查；反之，则关闭预热容器网卡上限值检查。 参数值需为正整数。例如0，表示关闭预热容器网卡上限值检查。当超过节点的容器网卡配额时，后台取值为节点的容器网卡配额。	建议配置为大部分节点平时最多运行的Pod数。
节点动态预热容器网卡数(nic-warm-target)	2	保障节点至少预热的容器网卡数，只支持数值配置。 当 节点动态预热容器网卡数(nic-warm-target) + 节点当前绑定的容器网卡数 大于 节点预热容器网卡上限检查值(nic-maximum-target) 时，只会预热nic-maximum-target与节点当前绑定的容器网卡数的差值。	建议配置为大部分节点日常10s内会瞬时弹性扩容的Pod数。
节点预热容器网卡回收阈值(nic-max-above-warm-target)	2	只有当 节点上空闲的容器网卡数 - 节点动态预热容器网卡数(nic-warm-target) 大于此阈值 时，才会触发预热容器网卡的解绑回收。只支持数值配置。 • 调大此值会减慢空闲容器网卡的回收，加快Pod的启动速度，但会降低IP地址的利用率，特别是在IP地址紧张的场景，**请谨慎调大**。 • 调小此值会加快空闲容器网卡的回收，提高IP地址的利用率，但在瞬时大量Pod激增的场景，部分Pod启动会稍微变慢。	建议配置为大部分节点日常在分钟级时间范围内会频繁弹性扩容缩容的Pod数 - 大部分节点日常10s内会瞬时弹性扩容的Pod数。

17.4 调度配置

为您提供kube-scheduler基础配置信息，并提供Volcano作为容器调度器的高级调度能力配置，您可以在此开启装箱策略、基于优先级的调度与抢占、AI任务性能增强、异构资源管理等高级调度能力，提升集群资源利用率，为您节约成本。

设置集群默认调度器

默认调度器 (default-scheduler)

Kubernetes调度器可以发现集群中新创建且尚未被调度到节点上的Pod，并负责将未调度的Pod指派到一个合适的节点上运行。在同一个集群中可以使用多个不同的调度器，kube-scheduler调度器是Kubernetes社区提供的集群默认调度器，CCE同时还支持增强的Volcano调度器，提供了高性能任务调度引擎、高性能异构芯片管理、高性能任务运行管理等通用计算能力。

您可以选择将kube-scheduler调度器和Volcano调度器配合使用，也可以单独使用kube-scheduler调度器或Volcano调度器。

表 17-5 集群默认调度器

调度器名称	说明	调度器配置
kube-scheduler调度器	提供社区原生调度器标准调度能力。 设置kube-scheduler调度器为默认调度器时，如果集群中同时安装**Volcano调度器**，将默认启用Volcano增强能力，为您提供资源利用率优化、AI任务性能增强、异构资源管理等高级调度能力，提升集群资源利用率，节约使用成本。此时，集群中的普通工作负载调度任务任由kube-scheduler调度器执行，仅部分指定的工作负载由Volcano调度器执行，详情请参见**使用Volcano调度工作负载**。	kube-scheduler调度器配置： • **调度器性能配置** • **业务优先级保障调度** 安装Volcano调度器后的增强配置： • **资源利用率优化调度（Volcano调度器支持）** • **AI任务性能增强调度（Volcano调度器支持）** • **异构资源调度（Volcano调度器支持）**

调度器名称	说明	调度器配置
Volcano调度器（v1.27及以上版本支持）	设置Volcano调度器为默认调度器后，Volcano调度器将替换kube-scheduler调度器，集群中的工作负载任务调度均由Volcano调度器执行。 Volcano兼容kube-scheduler调度能力，并提供增量调度能力。使用该调度器时，请先安装Volcano调度器插件，详情请参见**Volcano调度器**。	Volcano调度器增强配置： • **业务优先级保障调度** • **资源利用率优化调度（Volcano调度器支持）** • **AI任务性能增强调度（Volcano调度器支持）** • **异构资源调度（Volcano调度器支持）**

Kube-scheduler 调度器

kube-scheduler 提供社区原生调度器标准调度能力。

启用volcano增强能力：需安装Volcano 调度器插件。开启后为您提供资源利用率优化、AI任务性能增强、异构资源管理等高级调度能力，提升集群资源利用率，节约使用成本。

Volcano调度器增强配置：

- **业务优先级保障调度**
- **资源利用率优化调度（Volcano调度器支持）**
- **AI任务性能增强调度（Volcano调度器支持）**
- **异构资源调度（Volcano调度器支持）**

调度器性能配置

说明

仅kube-scheduler调度器支持该配置。

表 17-6 调度器性能配置参数说明

名称	参数	说明	取值
调度器访问kube-apiserver的QPS	kube-api-qps	与kube-apiserver通信的QPS，即每秒查询率。	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200

名称	参数	说明	取值
调度器访问kube-apiserver的突发流量上限	kube-api-burst	与kube-apiserver通信的突发流量上限。	• 集群规格为1000节点以下时，默认值100 • 集群规格为1000节点及以上时，默认值200

业务优先级保障调度

基于优先级调度

基础调度能力，不支持关闭，调度器会优先保障高优先级业务运行，但不会主动驱逐已运行的低优先级业务。详情请参见**优先级调度与抢占**。

基于优先级抢占调度（Volcano调度器支持）

启用该能力后，集群资源不足时，调度器主动驱逐低优先级业务，保障高优先级业务正常调度。详情请参见**优先级调度与抢占**。

说明

- 集群默认调度器设置为Volcano调度器时支持该配置。
- 基于优先级抢占调度能力与Pod延迟创建能力不可同时开启。

资源利用率优化调度（Volcano 调度器支持）

装箱策略（Binpack）

启用该能力后，调度器优先选择具有最多请求资源的节点，减少各节点资源碎片，提高集群整体资源利用率。详情请参见**装箱调度（Binpack）**。

装箱策略整体权重和内部各资源维度的打分权重设置如**表17-7**。

表 17-7 装箱策略权重配置

名称	说明	默认值
装箱调度策略权重	增大该权重值，可提高装箱策略在整体调度中的影响力。	10
CPU权重	增大该权重值，优先提高集群CPU利用率。	1
内存权重	增大该权重值，优先提高集群Memory利用率。	1
自定义资源类型	指定Pod请求的其他自定义资源类型，例如nvidia.com/gpu。增大该权重值，优先提高指定资源的利用率。	-

负载感知调度（Usage）

负载感知调度通过云原生监控插件（kube-prometheus-stack）获取各节点 CPU、内存的真实负载数据，根据用户指定的周期计算各节点的负载平均值，优先调度任务至真实负载较低的节点，实现节点负载均衡。详情请参见**负载感知调度**。

AI 任务性能增强调度（Volcano 调度器支持）

公平调度（DRF）

DRF（Dominant Resource Fairness）是主资源公平调度策略，可以支持多种类型资源的公平分配，应用于大批量提交AI训练和大数据作业场景。DRF调度算法优先考虑集群中业务的吞吐量，适用单次AI训练、单次大数据计算以及查询等批处理小业务场景。

启用公平调度（DRF）后，可增强集群业务的吞吐量，提高业务运行性能。详情请参见**公平调度（DRF）**。

组调度（Gang）

Gang调度策略满足了调度过程中的“All or nothing”的调度需求，避免Pod的任意调度导致集群资源的浪费，应用于AI、大数据等多任务协作场景。

启用组调度（Gang）后，可以解决分布式训练任务之间的资源忙等待和死锁等痛点问题，大幅度提升整体训练性能。详情请参见**组调度（Gang）**。

异构资源调度（Volcano 调度器支持）

支持GPU资源调度

使用该能力时，集群中需要同时安装**CCE AI套件（NVIDIA GPU）**。启用该能力后，可使用GPU资源运行AI训练作业，调度器提供GPU整卡调度和GPU共享调度能力，提高GPU资源利用率。

支持NPU资源调度

使用该能力时，集群中需要同时安装**CCE AI套件（Ascend NPU）**。启用该能力后，可使用使用NPU资源运行AI训练作业，调度器提供NPU拓扑感知调度能力，提高训练作业执行效率。

17.5 集群弹性伸缩配置

弹性扩容配置

CCE集群弹性引擎将综合判断整集群的资源情况，当微服务负载高（CPU/内存使用率过高）时水平扩容，增加Pod的数量以降低负载。

节点扩容条件

- 负载无法调度时自动扩容：集群中存在负载实例无法调度时，尝试自动扩容已开启弹性伸缩的节点池。若Pod已经设置亲和某个节点，则不会自动扩容节点。

  该功能可以和HPA策略配合使用，具体请参见**使用HPA+CA实现工作负载和节点联动弹性伸缩**。
- 自定义节点弹性策略开关：根据**节点弹性策略**自动扩容节点池，默认开启。

节点扩容资源上限

- 节点数量：扩容时，集群下所有节点数量的上限，超过该值时将不再扩容。默认为集群管理规模的节点上限。
- CPU核数：扩容时，集群下所有节点CPU核数之和的上限，超过该值时将不再扩容。默认不限。
- 内存（GiB）：扩容时，集群下所有节点内存之和的上限，超过该值时将不再扩容。默认不限。

说明

统计节点、CPU和内存总数时，包含自定义节点池的不可用节点，但是不包含默认节点池中的不可用节点。

节点池扩容优先级

节点池列表可通过拖拽调整扩容优先级。

弹性缩容配置

CCE集群弹性引擎将综合判断整集群的资源情况，在满足负载迁移后能够正常调度运行的前提下，自动筛选节点来进行缩容。

节点缩容条件

- 节点资源条件：当集群节点资源的Request值（CPU和内存需同时满足）连续一段时间（默认10min）低于一定百分比（默认50%）时，会触发集群缩容操作。
- 节点状态条件：节点处于不可用状态下超过一定时间会被自动回收，默认为20分钟。
- 缩容例外场景：节点满足以下例外场景时，即使节点资源或状态满足缩容条件，不会被CCE集群弹性引擎自动缩容。
  a. 集群其它节点资源不足时将不会触发非完全空闲节点缩容。
  b. 节点开启缩容保护时将不会触发节点缩容。如需开启或关闭节点缩容保护，请前往“节点管理 > 节点”页面，单击节点操作列的“更多 > 开启/关闭节点缩容保护”按钮操作。
  c. 节点上存在指定不缩容标记的Pod时，该节点将不会被缩容。
  d. 节点上的部分容器存在可靠性等配置策略时，将有可能不会自动缩容。
  e. 节点上存在kube-system命名空间下的非DaemonSet类容器时，该节点将不会被缩容。
  f. （可选）节点上如果存在已运行的容器由第三方Pod Controller进行管理，则该节点不会被缩容。第三方Pod Controller是指除Kubernetes原生的工作负载（如Deployment、StatefulSet等）外的自定义工作负载，可通过**自定义资源CRD**进行创建。

节点缩容策略

表 17-8 节点缩容策略配置

名称	说明	默认值
缩容并发数	最多支持多少个空闲节点同时缩容。 缩容并发数只针对完全空闲节点，完全空闲节点可实现并发缩容。非完全空闲节点则只能逐个缩容。 **说明** 节点在缩容的时候，若节点上的Pod不需要驱逐（DaemonSet的Pod认为不需要驱逐），则认为该节点为完全空闲节点，否则认为该节点为非完全空闲。	10
检查周期	节点被判定不可移除后能再次启动检查的时间间隔。	5min
冷却时间	集群触发弹性缩容后，再次启动缩容评估的冷却时间。 **说明** 集群中如果同时存在自动扩容和自动缩容的场景，建议配置该参数为0min，避免由于部分节点池持续扩容或者扩容失败重试而阻塞整体缩容节点行为，导致非预期的节点资源浪费。	10min
	集群触发弹性扩容后，再次启动缩容评估的冷却时间。	10min
	集群触发弹性缩容失败后，再次启动缩容评估的冷却时间。	3min

17.6 监控运维配置

CCE为您提供监控应用及资源的能力，支持采集各项指标及事件等数据以分析应用健康状态，您可以通过“配置中心 > 监控运维配置”统一调整监控运维参数。

您需要**开通监控中心**，以使用监控运维配置的所有功能。

监控配置

采集配置

- 系统预置采集：可视化管理云原生监控插件的监控采集任务。详情请参见**管理监控采集任务**。
- ServiceMonitor：定义针对Service的自定义指标采集策略，详情请参见**管理监控采集任务**。
  关于ServiceMonitor的创建方式请参见**配置Service Monitor监控自定义指标**。
- PodMonitor：定义针对Pod的自定义指标采集策略，详情请参见**管理监控采集任务**。
  关于PodMonitor的创建方式请参见**配置Pod Monitor监控自定义指标**。
- Targets：指标采集目标的状态。详情请参见**管理监控采集任务**。
- 默认采集周期：云原生监控插件的指标采集周期，默认为15秒。
- 自定义指标采集策略：开启后，云原生监控插件支持采集应用的自定义指标。您需要按照Prometheus规范在应用中提供自定义指标，并且暴露自定义指标的接

口，然后在集群中使用该应用镜像部署工作负载，Prometheus会通过采集配置对这些指标进行采集。详情请参见**使用云原生监控插件监控自定义指标**。

对接AOM监控服务

AOM实例是应用运维管理服务（AOM）推出的Prometheus监控功能。启用后指标会上报到您选择的AOM实例，其中容器基础指标免费，其他指标按需收费。关于免费指标详情请参见**基础指标-容器指标**。

对接第三方监控平台

开启后，支持将普罗数据上报至第三方监控平台，您需要提前获取第三方监控平台的数据上报地址及身份认证凭据。详情请参见**CCE云原生监控插件对接第三方监控平台**。

日志配置

采集配置

CCE可以帮助您快速采集 Kubernetes 集群的容器日志，包括容器标准输出、容器内日志文件、节点日志以及Kubernetes事件，并可快速进行日志查询与分析。

须知

日志上报LTS会创建名为k8s-logs-{clusterId}的默认日志组，并收取相关的费用。LTS收费标准请参见**价格计算器**。

日志类型	日志	LTS日志流名称	开启状态	参考文档
容器日志	容器标准输出	stdout-{clusterId}	开通业务日志采集需安装**云原生日志采集插件**。	**通过云原生日志采集插件采集容器日志**
Kubernetes事件	Kubernetes事件	event-{clusterId}	开通业务日志采集需安装**云原生日志采集插件**。	**采集Kubernetes事件**
Kubernetes审计日志	Kubernetes审计日志	audit-{clusterId}	支持单独配置开关	**采集Kubernetes审计日志**
控制面组件日志	kube-apiserver日志	kube-apiserver-{clusterId}	支持单独配置开关	**采集控制面组件日志**
	kube-controller-manager日志	kube-controller-manager-{clusterId}	支持单独配置开关	

日志类型	日志	LTS日志流名称	开启状态	参考文档
	kube-scheduler日志	kube-scheduler-{clusterId}	支持单独配置开关	

Kubernetes事件上报至AOM

集群中安装**云原生日志采集插件**后，Kubernetes事件默认上报至LTS，您可以通过该配置将Kubernetes事件上报至AOM。

- 异常事件上报：默认开启，会将所有异常事件上报至AOM。您可以单击“配置黑名单”，将不需要上报的事件添加至黑名单进行管理，其中“事件名称”可通过**CCE事件列表**查询。
- 普通事件上报：开启后，会将普通事件上报至AOM，系统默认配置了部分需要上报的普通事件。如果您需要自定义上报的事件，可以单击“配置白名单”，将需要上报添加至白名单进行管理，其中“事件名称”可通过**CCE事件列表**查询。

17.7 Kubernetes 原生配置

为您提供典型的原生配置选项，您可以在此设置kube-apiserver、kube-controller等社区原生管理组件的配置，为您的集群在海量场景下提供最佳的云原生体验。

集群服务器配置（kube-apiserver）

容器故障迁移默认容忍周期

容器故障迁移默认容忍周期配置默认对集群中所有的容器生效，您也可以为指定Pod进行差异化容忍配置，此时将以Pod配置的容忍时长为准。

说明

请合理设置容忍时间配置，否则可能出现以下问题：

- 配置过小：在网络抖动等短时故障场景下，容器可能会频繁迁移而影响业务。
- 配置过大：在节点故障时，容器可能长时间无法迁移，导致业务受损。

表 17-9 容器故障迁移默认容忍周期配置参数说明

名称	参数	说明	取值
容器迁移对节点不可用状态的容忍时间	default-not-ready-toleration-seconds	表示节点处于NotReady状态下的容忍时间。当节点出现异常，变为不可用状态时，容器将在该容忍时间后自动驱逐，默认为300s。	默认：300s

名称	参数	说明	取值
容器迁移对节点无法访问状态的容忍时间	default-unreachable-toleration-seconds	表示节点处于unreachable状态下的容忍时间。当环境出现异常，例如节点无法访问（如节点网络异常）时，容器将在该容忍时间后自动驱逐，默认为300s。	默认：300s

准入控制器插件配置

Kubernetes支持开启一些准入控制插件，可以在集群对Kubernetes API对象（如Pods、Services、Deployments等）进行修改操作前，进行相应的约束与控制。

说明

v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。

表 17-10 准入控制器插件配置参数说明

名称	参数	说明	取值
节点限制插件	enable-admission-plugin-node-restriction	节点限制插件限制了节点的 kubelet 只能操作当前节点的对象，增强了在高安全要求或多租户场景下的隔离性。详细信息请参考官方文档。	开启/关闭
Pod节点选择器插件	enable-admission-plugin-pod-node-selector	Pod节点选择器插件允许集群管理员通过命名空间注释设置默认节点选择器，帮助约束 Pod 可以运行的节点，并简化配置。	开启/关闭
Pod容忍度限制插件	enable-admission-plugin-pod-toleration-restriction	Pod容忍度限制插件允许通过命名空间设置 Pod 的容忍度的默认值和限制，为集群管理者提供了对 Pod 调度的精细控制，以保护关键资源。	开启/关闭

服务账号令牌卷投射

kubelet可以将ServiceAccount令牌投射到Pod中。您可以指定令牌的期望属性，例如API受众。当Pod或ServiceAccount被删除时，该令牌也将对API无效。详细信息请参考官方文档。

说明

v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。

表 17-11 服务账号令牌卷投射配置参数说明

名称	参数	说明	取值
API 受众	api-audiences	为ServiceAccount令牌定义其受众。Kubernetes 用于服务账户令牌的身份验证组件，会验证API请求中使用的令牌是否指定了合法的受众。 配置建议：根据集群服务间通信的需求，精确配置受众列表。此举确保服务账户令牌仅在授权的服务间进行认证使用，提升安全性。 **说明** 不正确的配置可能导致服务间认证通信失败，或令牌的验证过程出现错误。	默认值："https://kubernetes.default.svc.cluster.local" 支持配置多个值，用英文逗号隔开。
服务账户令牌发行者	service-account-issuer	指定发行服务账户令牌的实体标识符。这是在服务账户令牌的负载（Payload）中的 'iss' 字段所标识的值。 配置建议：请确保所配置的发行者（Issuer）URL在集群内部可被访问，并且可被集群内部的认证系统所信任。 **说明** 若设置了一个不可信或无法访问的发行者 URL，可能会导致基于服务账户的认证流程失败。	默认值："https://kubernetes.default.svc.cluster.local" 支持配置多个值，用英文逗号隔开。

集群控制器配置（kube-controller-manager）

控制器公共配置

- **控制器性能配置**：用于设置控制器访问kube-api-server的性能参数配置。

  说明

  请合理设置控制器性能配置，否则可能出现以下问题：
  - 配置过小：可能会触发客户端限流，对控制器性能产生影响。
  - 配置过大：可能会导致kube-apiserver过载。

表 17-12 控制器性能配置参数说明

名称	参数	说明	取值
控制器访问kube-apiserver的QPS	kube-api-qps	与kube-apiserver通信的QPS，即每秒查询率。	• 集群规格为1000节点以下时，默认值为100 • 集群规格为1000节点及以上时，默认值为200
控制器访问kube-apiserver的突发流量上限	kube-api-burst	与kube-apiserver通信的突发流量上限。	• 集群规格为1000节点以下时，默认值为100 • 集群规格为1000节点及以上时，默认值为200

- **资源对象处理并发配置**：允许同时同步的资源对象的数量。配置数量越大，管理响应越快，但 CPU（和网络）负载也越高。

  说明

  请合理设置资源对象处理并发配置，否则可能出现以下问题：
  - 配置过小：可能导致管理器处理响应慢。
  - 配置过大：会对集群管控面造成压力，产生过载风险。

表 17-13 资源对象处理并发配置参数说明

名称	参数	说明	取值
Deployment	concurrent-deployment-syncs	可以并发同步的Deployment对象个数。数值越大意味着对Deployment的响应越及时，同时也意味着更大的CPU（和网络带宽）压力。	默认：5
Endpoint	concurrent-endpoint-syncs	可以并发同步的Endpoints对象个数。数值越大意味着更新Endpoints越快，同时也意味着更大的CPU（和网络）压力。	默认：5

名称	参数	说明	取值
GC回收	concurrent-gc-syncs	可以并发同步的垃圾收集（Garbage Collector）工作线程个数。	默认：20
Job	concurrent-job-syncs	可以并发同步的Job对象个数。较大的数值意味着对Job的响应越及时，不过也意味着更多的CPU（和网络）占用。	默认：5
CronJob	concurrent-cron-job-syncs	可以并发同步的CronJob对象个数。较大的数值意味着对CronJob的响应越及时，不过也意味着更多的CPU（和网络）占用。	默认：5
Namespace	concurrent-namespace-syncs	可以并发同步的Namespace对象个数。较大的数值意味着对Namespace的响应越及时，不过也意味着更多的CPU（和网络）占用。	默认：10
Replicaset	concurrent-replicaset-syncs	可以并发同步的ReplicaSet个数。数值越大，副本管理的响应速度越快，同时也意味着更多的CPU（和网络）占用。	默认：5
ResourceQuota	concurrent-resource-quota-syncs	可以并发同步的ResourceQuota对象个数。数值越大，配额管理的响应速度越快，不过对CPU（和网络）的占用也越高。	默认：5
Servicepace	concurrent-service-syncs	可以并发同步的Service对象个数。数值越大，服务管理的响应速度越快，不过对CPU（和网络）的占用也越高。	默认：10
ServiceAccountToken	concurrent-serviceaccount-token-syncs	可以并发同步的服务账号令牌对象个数。数值越大，令牌生成的速度越快，不过对CPU（和网络）的占用也越高。	默认：5
TTLAfterFinished	concurrent-ttl-after-finished-syncs	可以并发同步的ttl-after-finished-controller线程个数。	默认：5

名称	参数	说明	取值
RC	concurrent_rc_syncs	可以并发同步的副本控制器对象个数。数值越大，副本管理操作越快，不过对CPU（和网络）的占用也越高。 **说明** 该参数仅在v1.19及以下版本集群中使用。	默认：5
RC	concurrent-rc-syncs	可以并发同步的副本控制器对象个数。数值越大，副本管理操作越快，不过对CPU（和网络）的占用也越高。 **说明** 该参数仅在v1.21至v1.23版本集群中使用。v1.25版本后，该参数弃用（正式弃用版本为v1.25.3-r0）。	默认：5
HPA并发处理数	concurrent-horizontal-pod-autoscaler-syncs	允许并发执行的HPA弹性伸缩数量。 数值越大，HPA弹性伸缩响应越快，不过对CPU（和网络）的占用也越高。 该参数仅v1.27及以上版本集群支持。	默认：5 取值范围为1-50

节点生命周期控制器（node-lifecycle-controller）配置

说明

v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上版本的集群支持该参数。

表 17-14 负载弹性伸缩控制器配置参数说明

名称	参数	说明	取值
可用区亚健康阈值	unhealthy-zone-threshold	当可用区故障节点规模达到指定比例时被认定为不健康，针对不健康的区域，故障节点业务的迁移频率会降级，避免规模故障场景下大规模迁移操作产生更坏的影响。 **说明** 比例配置过大可能导致区域在规模故障场景下仍尝试执行大规模迁移动作，导致集群过载等风险。	默认：0.55

名称	参数	说明	取值
节点迁移速率	node-eviction-rate	当某区域健康时，在节点故障的情况下每秒删除Pod的节点数。该值默认设置为0.1，代表每10秒钟内至多从一个节点驱逐Pod。 **说明** 结合集群规模合理设置，建议按比例折算后每批迁移Pod数量不超过300。 迁移速率设置过大可能引入集群过载风险，同时每批迁移重调度的Pod过多，大量Pod无法及时调度，影响整体故障恢复时间。	默认：0.1
次级节点迁移速率	secondary-node-eviction-rate	当某区域不健康时，在节点故障的情况下每秒删除Pod的节点数。该值默认设置为0.01，代表每100秒钟内至多从一个节点驱逐Pod。 **说明** 配合node-eviction-rate设置，一般建议设置为node-eviction-rate的十分之一。 区域亚健康场景迁移速率设置过大无实际意义，且可能引入集群过载风险。	默认：0.01
大规模集群大小阈值	large-cluster-size-threshold	集群内节点数量大于此参数时，集群被判断为大规模集群。 配置建议：在拥有大量节点的集群中，适当增加此阈值可以帮助提高控制器的性能和响应速度。对于规模较小的集群，保持默认值即可。在调整此参数时，建议先在测试环境中验证其对性能的影响，然后再在生产环境中应用。 **说明** 被视为大型集群时，kube-controller-manager 会进行特定配置调整。这些配置用来优化大规模集群性能。因此阈值如果过低，规模小的集群用上的大集群的配置，反而降低性能。	默认：50

负载弹性伸缩控制器（horizontal-pod-autoscaler-controller）配置

表 17-15 负载弹性伸缩控制器配置参数说明

名称	参数	说明	取值
Pod水平伸缩同步的周期	horizontal-pod-autoscaler-sync-period	水平Pod扩缩器对Pod进行弹性伸缩的周期。配置越小弹性伸缩器反应越及时，同时CPU负载也越高。 **说明** 请合理设置该参数，周期配置过长可能导致控制器处理响应慢；周期配置过短则会对集群管控面造成压力，产生过载风险。	默认：15s
Pod水平伸缩容忍度	horizontal-pod-autoscaler-tolerance	该配置影响控制器对伸缩策略相关指标反应的灵敏程度，当配置为0时，指标达到策略阈值时立即触发弹性。 配置建议：如业务资源占用随时间的“突刺”特征明显，建议保留一定的容忍度值，避免因业务短时资源占用飚高导致预期之外的弹性行为。	默认：0.1
HPA CPU初始化期间	horizontal-pod-autoscaler-cpu-initialization-period	这一时段定义了纳入HPA计算的CPU使用数据仅来源于已经达到就绪状态并完成了最近一次指标采集的Pods。它的目的是在Pod启动初期过滤掉不稳定的CPU使用数据，进而防止基于瞬时峰值做出错误的扩缩容决策。 配置建议：如果您观察到Pods在启动阶段的CPU使用率波动导致HPA做出错误的扩展决策，增大此值可以提供一个CPU使用率稳定化的缓冲期。 **说明** 请合理设置该参数，设置值过低可能导致基于CPU峰值做出过度反应的扩容；而设置得过高则可能在实际需要扩容时造成延迟反应。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认：5分钟

名称	参数	说明	取值
HPA 初始就绪状态延迟	horizontal-pod-autoscaler-initial-readiness-delay	在CPU初始化期之后，此时间段允许HPA以一个较宽松的标准筛选CPU度量数据。也就是说，这段时间内，即使Pods的就绪状态有所变化，HPA也会考虑它们的CPU使用数据进行扩缩容。这有助于在Pod状态频繁变化时，确保CPU使用数据被持续追踪。 配置建议：如果Pods在启动后的就绪状态发生波动，并且您需要避免此波动导致HPA的误判，适当增加此值可以使HPA得到更全面的CPU使用数据。 **说明** 请合理设置该参数，值设置过低可能会在Pod刚进入就绪状态时，因CPU数据波动导致不恰当的扩容行为；而设置过高则可能导致在需要快速反应时HPA无法立即做出决策。 v1.23.16-r0、v1.25.11-r0、v1.27.8-r0、v1.28.6-r0、v1.29.2-r0及以上版本的集群支持该参数。	默认：30s

Pod回收控制器（pod-garbage-collector-controller）配置

表 17-16 Pod 回收控制器配置参数说明

名称	参数	说明	取值
终止状态Pod触发回收的数量阈值	terminated-pod-gc-threshold	在Pod GC开始删除终止状态（terminated）的Pod之前，系统允许存在终止状态的Pod数量。 **说明** 请合理设置该参数，配置过大时，集群中可能存在大量终止状态的Pod，影响相关List查询请求性能，产生集群过载风险。	默认：1000 取值范围为10-12500

资源配额控制器（resource-quota-controller）配置

说明

在高并发场景下（如批量创建Pod），配额管理机制可能导致部分请求因冲突而失败，除非必要不建议启用该功能。如启用，请确保请求客户端具备重试机制。

表 17-17 资源配额控制器配置参数说明

名称	参数	说明	取值
启用资源配额管理	enable-resource-quota	通过配额管理功能，用户可以对命名空间或相关维度下的各类负载（Deployment、Pod等）数量以及资源（CPU、Memory）上限进行控制。命名空间通过ResourceQuota对象进行配额限制。 • 关闭（false）：不自动创建ResourceQuota对象。 • 开启（true）：自动创建ResourceQuota对象。ResourceQuota的默认取值请参见**设置资源配额及限制**。	默认：关闭（false）

17.8 异构资源配置

GPU 配置

GPU虚拟化

CCE GPU虚拟化采用自研xGPU虚拟化技术，能够动态对GPU设备显存与算力进行划分，单个GPU卡最多虚拟化成20个GPU虚拟设备。相对于静态分配来说，虚拟化的方案更加灵活，最大程度保证业务稳定的前提下，可以完全由用户自己定义使用的GPU量，提高GPU利用率。详情请参见**GPU虚拟化概述**。

GPU驱动配置

- **集群默认驱动**：集群中GPU节点默认使用的GPU驱动版本。如果选择“自定义驱动链接地址”，则需填写Nvidia驱动的下载链接，详情请参见**获取驱动链接-公网地址**。
- **节点池自定义驱动**：若您不希望集群中的所有GPU节点使用相同的驱动，CCE支持以节点池为单位安装不同的GPU驱动。配置节点池自定义驱动后，节点池中节点优先使用当前节点池自定义驱动，未指定驱动的节点将使用集群默认驱动。

  说明

  - 系统将根据节点池指定的驱动版本进行安装，仅对节点池新建节点生效。
  - 更新驱动版本后，新建节点直接生效，存量节点需重启节点生效。
  - 安装2.7.2及以上版本的GPU插件时，支持以节点池级别配置XGPU虚拟化开关。

NPU 配置

当不开启驱动选择时，无法根据用户诉求指定驱动版本，无法依靠插件进行驱动维护。如从控制台创建NPU节点，控制台会自动补充NPU驱动（用户无法指定版本和类型）安装命令，并在安装完成后自动重启节点；如通过API或其他方式创建节点则需要用户在“安装后执行脚本”中添加驱动安装命令。

开启驱动选择后，NPU插件启动时将自动根据对应机型的驱动配置安装驱动，驱动维护更灵活。推荐使用默认的驱动版本，您也可以选择“自定义驱动”并填写完整的驱动地址。

支持的NPU卡类型和对应的操作系统规格如下：

NPU卡类型	支持的操作系统
D310	EulerOS 2.5 x86、CentOS 7.6 x86、EulerOS 2.9 x86、EulerOS 2.8 arm

18 存储管理-Flexvolume（已弃用）

18.1 存储 Flexvolume 概述

容器存储是为容器工作负载提供存储的组件，支持多种类型的存储，同一个工作负载（pod)可以使用任意数量的存储。

云容器引擎CCE的容器存储功能基于Kubernetes存储系统，并深度融合云存储服务，完全兼容Kubernetes原生的存储服务，例如EmptyDir、HostPath、Secret、ConfigMap等存储。

1.13及以下版本的CCE基于Kubernetes社区Flexvolume容器存储接口（**storage-driver**）实现了云存储服务接入能力，目前该插件已经不是官方建议的存储扩展方式，**在1.15及以上版本的CCE集群中默认安装CSI容器存储插件（everest）**，详情参见**存储概述**。

说明

- Kubernetes1.13版本之前的CCE集群不支持端到端容器存储扩容功能，PVC容量与存储容量不一致。
- 在**v1.13及以下版本**的集群中，当存储功能有升级或者BUG修复时，用户无需升级集群或新建集群来升级存储功能，仅需安装或升级storage-driver插件。

约束与限制

- 在CCE所创的集群中，Kubernetes v1.15.11版本是Flexvolume插件（storage-driver）被CSI插件（**everest**）兼容接管的过渡版本，v1.17及之后版本的集群中将彻底去除对Flexvolume插件（**storage-driver**）的支持，请您将Flexvolume插件的使用切换到CSI插件上。
- CSI插件（**everest**）兼容接管Flexvolume插件（storage-driver）后，原有功能保持不变，但请注意不要新建Flexvolume插件（storage-driver）的存储，否则将导致部分存储功能异常。

如何判断集群的存储插件模式

步骤1 登录CCE控制台。

步骤2 在控制台左侧栏目树中，单击“插件中心”。

步骤3 在右侧的插件管理列表中，单击“插件实例”页签。

步骤4 在插件实例页面下，选择右上方的集群后，可以看到创建该集群时默认安装的存储插件。

----结束

CSI 和 Flexvolume 存储插件的区别

表 18-1 CSI 与 Flexvolume

Kubernetes插件方案	CCE插件名称	插件特性	使用推荐
CSI	everest	CSI插件是kubernetes社区推荐的存储插件机制。CCE发布的kubernetes1.15版本及以上版本默认安装CSI插件everest，并用于对接块存储、文件存储、对象存储、极速文件存储等Iaas存储服务。 everest插件包含两部分： • everest-csi-controller：提供存储卷的创建、删除、扩容、云盘快照等功能； • everest-csi-driver：提供存储卷在node上的挂载、卸载、格式化等功能。 详情请参见**everest**	针对**1.15及以上**版本的集群，在创建时将默认安装CSI插件（**everest**）。CCE会跟随社区持续更新CSI插件的各种能力。
Flexvolume	storage-driver	Flexvolume插件是kubernetes社区早期实现的存储卷插件机制。自CCE上线伊始，提供的就是Flexvolume数据卷服务。CCE发布的kubernetes 1.13及以下版本安装的插件是“storage-driver”，并用于对接块存储、文件存储、对象存储、极速文件存储等Iaas存储服务。 详情请参见**storage-driver**	针对已经创建的**1.13及以下**版本的集群，仍然使用已经安装的Flexvolume存储插件（**storage-driver**），CCE已停止更新该插件，您可以**升级集群版本**。

说明

- 不支持CSI和Flexvolume插件在同一个集群中使用。
- 不支持将v1.13及以下版本集群的Flexvolume插件转变到CSI插件，v1.13版本的集群可以通过升级集群版本切换为CSI插件，详情请参见**集群升级路径**。

18.2 1.15 集群如何从 Flexvolume 存储类型迁移到 CSI Everest 存储类型

在v1.15.11-r1之后版本的集群中，CSI Everest插件已接管fuxi Flexvolume（即storage-driver插件）容器存储的所有功能，建议将对fuxi Flexvolume的使用切换CSI Everest上。

迁移的主要原理是通过创建静态PV的形式关联原有底层存储，并创建新的PVC关联该新建的静态PV，之后应用升级挂载这个新的PVC到原有挂载路径，实现存储卷迁移。

警告

迁移时会造成服务断服，请合理规划迁移时间，并做好相关备份。

操作步骤

步骤1 数据备份（可选，主要防止异常情况下数据丢失）。

步骤2 根据FlexVolume格式的PV，准备CSI格式的PV的yaml文件关联已有存储。

执行如下命令，配置名为“pv-example.yaml”的创建PV的yaml文件。

touch pv-example.yaml

vi pv-example.yaml

云硬盘存储卷PV的配置示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: <zone name>
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
  name: pv-evs-example
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  csi:
    driver: disk.csi.everest.io
    fsType: ext4
    volumeAttributes:
      everest.io/disk-mode: SCSI
      everest.io/disk-volume-type: SAS
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: 0992dbda-6340-470e-a74e-4f0db288ed82
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-disk
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-2 云硬盘存储卷 PV 配置参数说明

参数	描述
failure-domain.beta.kubernetes.io/region	云硬盘所在region，可参考FlexVolume PV的相同字段。
failure-domain.beta.kubernetes.io/zone	云硬盘所在可用区，可参考FlexVolume PV的相同字段。
name	PV资源的名称，集群下唯一。
storage	云硬盘的容量，单位为Gi。可参考FlexVolume PV的spec.capacity.storage。
driver	挂载依赖的存储驱动，EVS云硬盘配置为“disk.csi.everest.io”。
volumeHandle	云硬盘的volumeID，可参考FlexVolume PV的spec.flexVolume.options.volumeID。
everest.io/disk-mode	云硬盘磁盘模式，可参考FlexVolume PV的spec.flexVolume.options.disk-mode。
everest.io/disk-volume-type	云硬盘类型，当前支持高I/O（SAS）、超高I/O（SSD）。可参考FlexVolume PV的spec.storageClassName对应的sc中的parameters."kubernetes.io/volumetype"。
storageClassName	存储卷动态供应关联的K8s storage class名称；云硬盘需使用“csi-disk”。

文件存储卷PV配置示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-sfs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: nas.csi.everest.io
    fsType: nfs
    volumeAttributes:
      everest.io/share-export-location: sfs-nas01.ap-southeast-1.myhuaweicloud.com:/share-436304e8
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: 682f00bb-ace0-41d8-9b3e-913c9aa6b695
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-nas
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-3 文件存储卷 PV 配置参数说明

参数	描述
name	PV资源的名称，集群下唯一。
storage	文件存储的大小，单位为Gi。可参考FlexVolume PV的spec.capacity.storage。
driver	挂载依赖的存储驱动，文件存储配置为“nas.csi.everest.io”。
everest.io/share-export-location	文件存储的共享路径。可参考FlexVolume PV的spec.flexVolume.options.deviceMountPath。
volumeHandle	文件存储的ID。可参考FlexVolume PV的spec.flexVolume.options.volumeID。
storageClassName	K8s storage class名称；需配置为"csi-nas"。

对象存储卷PV配置示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  csi:
    driver: obs.csi.everest.io
    fsType: s3fs
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: ap-southeast-1
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: obs-normal-static-pv
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-obs
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-4 对象存储卷 PV 配置参数说明

参数	描述
name	PV资源的名称，集群下唯一。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。
driver	挂载依赖的存储驱动，对象存储配置为“obs.csi.everest.io”。

参数	描述
fsType	文件类型，支持“obsfs”与“s3fs”，取值为s3fs时创建是obs对象桶，配套使用s3fs挂载；取值为obsfs时创建的是obs并行文件系统，配套使用obsfs挂载。可参考FlexVolume PV的spec.flexVolume.options.posix的对应关系：true（obsfs）、false/空值（s3fs）。
everest.io/obs-volume-type	存储类型，包括STANDARD（标准桶）、WARM（低频访问桶）。可参考FlexVolume PV的spec.flexVolume.options.storage_class的对应关系：standard（标准桶）、standard_ia（低频访问桶）。
everest.io/region	对象存储所在的region。可参考FlexVolume PV的spec.flexVolume.options.region。
volumeHandle	对象存储的桶名称。可参考FlexVolume PV的spec.flexVolume.options.volumeID。
storageClassName	K8s storage class名称；需配置为"csi-obs”。

极速文件存储卷PV配置示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-efs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: sfsturbo.csi.everest.io
    fsType: nfs
    volumeAttributes:
      everest.io/share-export-location: 192.168.0.169:/
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: 8962a2a2-a583-4b7f-bb74-fe76712d8414
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-sfsturbo
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-5 极速文件存储卷 PV 配置参数说明

参数	描述
name	PV资源的名称，集群下唯一。
storage	文件存储的大小。可参考FlexVolume PV的spec.capacity.storage。
driver	挂载依赖的存储驱动，极速文件存储配置为“sfsturbo.csi.everest.io”。

参数	描述
everest.io/share-export-location	极速文件存储的共享路径。可参考FlexVolume PV的spec.flexVolume.options.deviceMountPath。
volumeHandle	极速文件存储的ID。可参考FlexVolume PV的spec.flexVolume.options.volumeID。
storageClassName	指定K8s storage class名称；极速文件存储卷需配置为"csi-sfsturbo”。

步骤3 根据FlexVolume格式的PVC，准备CSI格式的PVC的yaml文件关联上述步骤准备的静态PV。

执行如下命令，配置名为“pvc-example.yaml”的创建PVC的yaml文件。

touch pvc-example.yaml

vi pvc-example.yaml

云硬盘存储卷PVC的配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: <zone name>
  annotations:
    everest.io/disk-volume-type: SAS
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName:  pv-evs-example
  storageClassName: csi-disk
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-6 云硬盘存储卷 PVC 配置参数说明

参数	描述
failure-domain.beta.kubernetes.io/region	集群所在region。可参考FlexVolume PVC的相同字段。
failure-domain.beta.kubernetes.io/zone	EVS云硬盘所在可用区。可参考FlexVolume PVC的相同字段。
everest.io/disk-volume-type	云硬盘存储类型，支持SAS、SSD。和上述步骤的PV保持一致。

参数	描述
name	PVC资源名称，同namespace下唯一。保证在namespace下唯一即可。（若PVC是由有状态应用动态创建，则保持和FlexVolume PVC的name一致）。
namespace	PVC资源命名空间。可参考FlexVolume PVC的相同字段。
storage	PVC申请容量，必须和已有PV的storage大小保持一致。
volumeName	PV的名称。使用上述步骤的静态PV的名称。
storageClassName	指定K8s storage class名称；云硬盘需使用“csi-disk”。

文件存储卷PVC配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-sfs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-nas
  volumeName: pv-sfs-example
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-7 文件存储卷 PVC 配置参数说明

参数	描述
name	PVC资源名称，同namespace下唯一。保证在namespace下唯一即可。（若PVC是由有状态应用动态创建，则保持和FlexVolume PVC的name一致）。
namespace	PVC资源命名空间。可参考FlexVolume PVC的相同字段。
storage	存储容量，单位Gi，必须和已有pv的storage大小保持一致。
storageClassName	需配置为"csi-nas"。
volumeName	PV的名称。参考上述步骤的静态PV的名称。

对象存储卷PVC配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
```



```
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: s3fs
  name: pvc-obs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
  volumeName: pv-obs-example
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-8 对象存储卷 PVC 配置参数说明

参数	描述
everest.io/obs-volume-type	obs存储类型；当前支持标准（STANDARD）和低频（WARM）两种存储类型。和上述步骤的PV保持一致。
csi.storage.k8s.io/fstype	指定文件类型，支持“obsfs”与“s3fs”。与上述步骤中静态obs存储的PV的fsType保持一致。
name	PVC资源名称，同namespace下唯一。保证在namespace下唯一即可。（若PVC是由有状态应用动态创建，则保持和FlexVolume PVC的name一致）。
namespace	PVC资源命名空间。可参考FlexVolume PVC的相同字段。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。
storageClassName	K8s storage class名称；需配置为"csi-obs”。
volumeName	PV的名称。参考上述步骤创建的静态PV的名称。

极速文件存储卷PVC配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-efs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-sfsturbo
  volumeName: pv-efs-example
```

加粗标红字段需要重点关注，其中参数说明如下：

表 18-9 极速文件存储卷 PVC 配置参数说明

参数	描述
name	PVC资源名称，同namespace下唯一。保证在namespace下唯一即可。（若PVC是由有状态应用动态创建，则保持和FlexVolume PVC的name一致）。
namespace	PVC资源命名空间。可参考FlexVolume PVC的相同字段。
storageClassName	指定K8s storage class名称；需配置为"csi-sfsturbo”。
storage	存储容量，单位Gi，必须和已有pv的storage大小保持一致。
volumeName	PV的名称。参考上述步骤创建的静态PV的名称。

步骤4 应用升级替换成新的PVC。

无状态应用

1. 通过kubectl create -f的形式创建pv和pvc。

   kubectl create -f pv-example.yaml

   kubectl create -f pvc-example.yaml

   说明

   命令中的yaml名称是示例，请以实际步骤**步骤2**和步骤**步骤3**创建的pv和pvc的yaml名字为准。

2. 进入应用更新升级界面：更新升级 - 高级设置 - 数据存储 - 云存储。

3. 卸载老存储，同时添加CSI格式的PVC的云存储，容器内挂载路径和以前保持一致，实现存储迁移。
4. 单击提交，确认后升级生效。
5. 等待pod running。

升级使用已有存储的有状态应用

1. 通过kubectl create -f的形式创建pv和pvc

   kubectl create -f pv-example.yaml

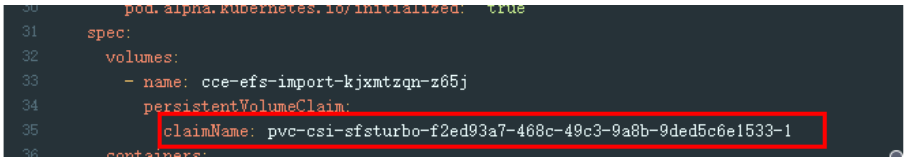
   kubectl create -f pvc-example.yaml

   说明

   命令中的yaml名称是示例，请以实际步骤**步骤2**和步骤**步骤3**创建的pv和pvc的yaml名字为准。

2. 通过kubectl edit的方式修改有状态应使用新建的PVC。

   kubectl edit sts sts-example -n xxx



说明

命令中的sts-example为待升级的有状态应用的名称，请以实际为准。xxx指代有状态应用所在的命名空间。

3. 等待pod running。

说明

当前界面暂未提供有状态应用添加新的云存储，因此升级替换成新PVC需要通过后台kubectl命令实现。

升级使用动态分配存储的有状态应用

1. 备份当前有状态应用使用的flexVolume格式的PV和PVC。

   kubectl get pvc xxx -n {namespaces} -oyaml > pvc-backup.yaml

   kubectl get pv xxx -n {namespaces} -oyaml > pv-backup.yaml

2. 将应用的实例数修改成0。
3. 在存储界面解关联有状态应用使用的flexVolume格式的PVC。
4. 通过kubectl create -f的形式创建pv和pvc。

   kubectl create -f pv-example.yaml

   kubectl create -f pvc-example.yaml

   说明

   命令中的yaml名称是示例，请以实际步骤**步骤2**和步骤**步骤3**创建的pv和pvc的yaml名字为准。

5. 将应用的实例数恢复，等待pod running。

说明

有状态应用动态创建存储是通过volumeClaimTemplates机制实现，而该字段K8s无法修改，因此无法通过更换新PVC的方式实现数据迁移。

volumeClaimTemplates的PVC命名格式是固定的，当符合命名格式的PVC已经存在的时候则直接使用该PVC。

因此需要先将原有PVC解除关联之后，创建同名的CSI格式的PVC来实现存储迁移。

6. 迁移完成，但是如果不重建有状态应用，扩容时仍是FlexVolume格式的PVC（按需操作）。

- 获取当前有状态应用yaml：

kubectl get sts xxx -n {namespaces} -oyaml > sts.yaml

- 备份当前有状态应用yaml：

cp sts.yaml sts-backup.yaml

- 修改有状态应用yaml中volumeClaimTemplates的定义：

vi sts.yaml

云硬盘存储卷volumeClaimTemplates的配置示例如下：

```
volumeClaimTemplates:
  - metadata:
      name: pvc-161070049798261342
      namespace: default
      creationTimestamp: null
      annotations:
        everest.io/disk-volume-type: SAS
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
      storageClassName: csi-disk
```

其中参数和上述步骤创建的云硬盘存储卷的PVC保持一致。

文件存储卷volumeClaimTemplates配置示例如下：

```
volumeClaimTemplates:
  - metadata:
      name: pvc-161063441560279697
      namespace: default
      creationTimestamp: null
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 10Gi
      storageClassName: csi-nas
```

其中参数和上述步骤创建的文件存储卷PVC保持一致。

对象存储卷PVC配置示例如下：

```
volumeClaimTemplates:
  - metadata:
      name: pvc-161070100417416148
      namespace: default
      creationTimestamp: null
      annotations:
        csi.storage.k8s.io/fstype: s3fs
        everest.io/obs-volume-type: STANDARD
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 1Gi
      storageClassName: csi-obs
```

其中参数和上述步骤创建的对象存储卷PVC保持一致。

- 删除原有状态应用：

kubectl delete sts xxx -n {namespaces}

- 创建新的有状态应用

kubectl create -f sts.yaml

步骤5 检查业务功能。

1. 检查业务功能是否正常。

2. 检查数据是否丢失。

说明

若功能或数据检查异常需要回退，请执行步骤**步骤4**，选择FlexVolume格式的PVC并单击提交升级。

步骤6 卸载FlexVolume格式的PVC。

检查正常，存储管理界面执行解关联操作。

也可以后台通过kubectl指令删除Flexvolume格式的PVC和PV。

注意

在删除之前需要修改PV的回收策略persistentVolumeReclaimPolicy为Retain，否则底层存储会被回收。

在存储迁移执行前已完成集群升级可能会导致无法删除PV，可以去除PV的保护字段finalizers来实现PV删除

kubectl patch pv {pv_name} -p '{"metadata":{"finalizers":null}}'

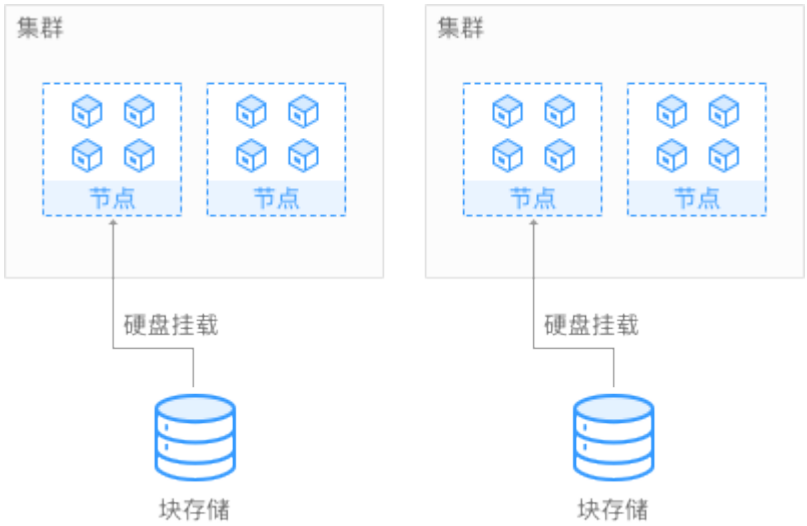
----结束

18.3 云硬盘存储卷

18.3.1 云硬盘存储卷概述

为满足数据持久化的需求，CCE支持将云硬盘（EVS）创建的存储卷挂载到容器的某一路径下，当容器迁移时，挂载的云硬盘将一同迁移。通过云硬盘，可以将存储系统的远端文件目录挂载到容器中，数据卷中的数据将被永久保存，即使删除了容器，数据卷中的数据依然保存在存储系统中。

图 18-1 CCE 挂载云硬盘存储卷



使用说明

- **使用便捷：**您可以像使用传统服务器硬盘一样，对挂载到服务器上的块存储（硬盘）做格式化、创建文件系统等操作。
- **数据不共享：**每台服务器使用独立的块存储（硬盘），多服务器之间数据隔离。
- **私有网络：**数据访问必须在数据中心内部网络中。
- **容量性能：**单卷容量有限（TB级），但性能极佳（IO读写时延ms级）。
- **使用限制：**不支持导入分区过或者具有非ext4文件系统的云硬盘。
- **应用场景：**主要面向HPC高性能计算、企业核心集群应用、企业应用系统和开发测试等。适用于供单实例部署的无状态负载（Deployment）和普通任务（Job），以及有状态工作负载（StatefulSet）的每个实例独占式使用。因为云硬盘属于非共享存储，不能同时被多个节点挂载，若两个Pod配置了使用同一个云硬盘，当这两个Pod被调度到不同的节点时，必然有一个Pod会因为无法挂载云硬盘导致无法成功启动。

18.3.2 使用 kubectl 自动创建云硬盘

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“pvc-evs-auto-example.yaml”的创建PVC的yaml文件。

touch pvc-evs-auto-example.yaml

vi pvc-evs-auto-example.yaml

1.9、1.11、1.13版本集群，yaml文件配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-auto-example
  namespace: default
  annotations:
    volume.beta.kubernetes.io/storage-class: sas
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: ap-southeast-1a
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

表 18-10 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	云硬盘类型，小写。

参数	描述
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
storage	存储容量，单位为Gi。
accessModes	指定读写模式，显示volume实际具有的访问模式。 支持配置“ReadWriteMany”（共享卷）与“ReadWriteOnce”（非共享卷）

步骤3 执行如下命令创建PVC。

kubectl create -f pvc-evs-auto-example.yaml

命令执行完成后，会在集群所在分区创建EVS云硬盘，您可以在“存储管理 > 云硬盘存储卷”中查看该云硬盘，也可以在EVS的控制台根据卷名称查看该硬盘。

----结束

18.3.3 使用 kubectl 对接已有云硬盘

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 登录EVS控制台，创建一个EVS云硬盘，记录云硬盘的VolumeID、容量和磁盘类型。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建两个yaml文件，用于创建PersistentVolume（PV）、PersistentVolumeClaim（PVC），假设文件名分别为**pv-evs-example.yaml**、**pvc-evs-example.yaml**。

touch pv-evs-example.yaml pvc-evs-example.yaml

K8s集群版本（K8s version）	说明	yaml示例
1.11.7 ≤ K8s version ≤ 1.13	1.11.7以上及1.13版本集群	请参见**1.11.7~1.13 yaml文件配置示例**
1.11 ≤ K8s version < 1.11.7	1.11.7之前的1.11版本集群	请参见**1.11~1.11.7 yaml文件配置示例**
K8s version = 1.9	1.9版本集群	请参见**1.9 yaml文件配置示例**

1.11.7 ≤ K8s version ≤ 1.13（1.11.7以上及1.13版本集群）

- **PV yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone:  ap-southeast-1a
  annotations:
    pv.kubernetes.io/provisioned-by: flexvolume-huawei.com/fuxivol
  name: pv-evs-example
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc-evs-example
    namespace: default
  flexVolume:
    driver: huawei.com/fuxivol
    fsType: ext4
    options:
      disk-mode: SCSI
      fsType: ext4
      volumeID: 0992dbda-6340-470e-a74e-4f0db288ed82
  persistentVolumeReclaimPolicy: Delete
  storageClassName: sas
```

表 18-11 关键参数说明

参数	描述
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
storage	云硬盘的容量，单位为Gi。
storageClassName	云硬盘类型，当前支持高I/O（SAS）、超高I/O（SSD）。
driver	挂载依赖的存储驱动。 EVS云硬盘配置为“huawei.com/fuxivol”。
volumeID	云硬盘的volumeID。 **获取方法：**在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“云硬盘存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。

参数	描述
disk-mode	云硬盘磁盘模式，取值可以是VBD和SCSI。 v1.11.7之前的CCE集群，该字段无需填写，默认都是VBD。 v1.11.7+以及v1.13的Linux x86 CCE集群要求该字段值必须存在，且基于PVC触发动态创建的都是EVS SCSI模式的卷，因此这里静态PV形式优先选用SCSI模式的云硬盘；同时支持升级后的老集群中VBD卷能够继续正常使用。
spec.claimRef.apiVersion	固定值"v1"。
spec.claimRef.kind	固定值"PersistentVolumeClaim"。
spec.claimRef.name	pvc名称；与下一步创建的pvc的name一致。
spec.claimRef.namespace	pvc的namespace；与下一步创建的pvc的namespace一致；

- **PVC yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: sas
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxivol
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: ap-southeast-1a
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-evs-example
```

表 18-12 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	存储类型，必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxivol。
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	EVS云硬盘所在可用区，必须和工作负载规划的可用区保持一致。

参数	描述
storage	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
volumeName	PV的名称。

1.11 ≤ K8s version < 1.11.7（1.11.7之前的1.11版本集群）

- **PV yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone:
  name: pv-evs-example
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  flexVolume:
    driver: huawei.com/fuxivol
    fsType: ext4
    options:
      fsType: ext4
      volumeID: 0992dbda-6340-470e-a74e-4f0db288ed82
  persistentVolumeReclaimPolicy: Delete
  storageClassName: sas
```

表 18-13 关键参数说明

参数	描述
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
storage	云硬盘的容量，单位为Gi。
storageClassName	云硬盘类型，当前支持高I/O（SAS）、超高I/O（SSD）。
driver	挂载依赖的存储驱动。 EVS云硬盘配置为“huawei.com/fuxivol”。
volumeID	云硬盘的volumeID。 **获取方法：**在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“云硬盘存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。

参数	描述
disk-mode	云硬盘磁盘模式，取值可以是VBD和SCSI。 v1.11.7之前的CCE集群，该字段无需填写，默认都是VBD。 v1.11.7+以及v1.13的Linux x86 CCE集群要求该字段值必须存在，且基于PVC触发动态创建的都是EVS SCSI模式的卷，因此这里静态PV形式优先选用SCSI模式的云硬盘；同时支持升级后的老集群中VBD卷能够继续正常使用。

- **PVC yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: sas
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxivol
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: ap-southeast-1a
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-evs-example
```

表 18-14 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	存储类型，支持sas，ssd。必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxivol。
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	EVS云硬盘所在可用区，必须和工作负载规划的可用区保持一致。
storage	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
volumeName	PV的名称。

K8s version = 1.9（1.9版本集群）

- **PV yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone:
  name: pv-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  flexVolume:
    driver: huawei.com/fuxivol
    fsType: ext4
    options:
      fsType: ext4
      kubernetes.io/namespace: default
      volumeID: 0992dbda-6340-470e-a74e-4f0db288ed82
  persistentVolumeReclaimPolicy: Delete
  storageClassName: sas
```

表 18-15 关键参数说明

参数	描述
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	创建云硬盘所在的可用区，必须和工作负载规划的可用区保持一致。
storage	云硬盘的容量，单位为Gi。
storageClassName	云硬盘类型，当前支持高I/O（SAS）、超高I/O（SSD）。
driver	挂载依赖的存储驱动。 EVS云硬盘配置为“huawei.com/fuxivol”。
volumeID	云硬盘的volumeID。 **获取方法：**在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“云硬盘存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。
disk-mode	云硬盘磁盘模式，取值可以是VBD和SCSI。 v1.11.7之前的CCE集群，该字段无需填写，默认都是VBD。 v1.11.7+以及v1.13的Linux x86 CCE集群要求该字段值必须存在，且基于PVC触发动态创建的都是EVS SCSI模式的卷，因此这里静态PV形式优先选用SCSI模式的云硬盘；同时支持升级后的老集群中VBD卷能够继续正常使用。

- **PVC yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: sas
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxivol
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone:
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-evs-example
  volumeNamespace: default
```

表 18-16 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	存储类型，必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxivol。
failure-domain.beta.kubernetes.io/region	集群所在的region。
failure-domain.beta.kubernetes.io/zone	EVS云硬盘所在可用区，必须和工作负载规划的可用区保持一致。
storage	PVC申请容量，单位为Gi。 必须和已有PV的storage大小保持一致。
volumeName	PV的名称。

步骤4 创建PV。

kubectl create -f pv-evs-example.yaml

步骤5 创建PVC。

kubectl create -f pvc-evs-example.yaml

执行成功后，可以在“资源管理 > 存储管理”的云硬盘存储中查看创建的PVC，也可以在EVS页面根据名称查看EVS云硬盘。

步骤6 （可选）增加集群关联的metadata，确保在删除节点或集群时避免删除已挂载的静态PV关联的EVS盘。

⚠ 注意

若不执行本步骤或创建静态PV/PVC时没有执行过本步骤，请务必确保删除节点前，提前将静态PV关联的云硬盘从节点上解关联。

1. 获取租户Token，详情请参见**获取用户Token** 。
2. 获取EVS访问地址EVS_ENDPOINT，详情请参见**区域和终端节点**。
3. 给EVS静态PV关联的EVS盘补充集群关联的metadata。

```
curl -X POST ${EVS_ENDPOINT}/v2/${project_id}/volumes/${volume_id}/metadata --insecure \
  -d '{"metadata":{"cluster_id": "${cluster_id}", "namespace": "${pvc_namespace}"}}' \
  -H 'Accept:application/json' -H 'Content-Type:application/json;charset=utf8' \
  -H 'X-Auth-Token:${TOKEN}'
```

表 18-17 关键参数说明

参数	描述
EVS_ENDPOINT	EVS访问地址，配置为**b**中获取的值。
project_id	项目ID。
volume_id	关联EVS盘的ID，配置为待创建静态PV中的volume_id，也可在EVS控制台，单击待导入的云硬盘名称，在磁盘详情界面的“概览信息”中获取ID的值，如**图18-2**。
cluster_id	待创建EVS PV的集群ID。在CCE控制台单击“资源管理 > 集群管理”。单击待关联集群的名称，在集群详情页面，即可获取集群ID，如**图18-3**。
pvc_namespace	待绑定PVC的namespace名称。
TOKEN	用户Token，配置为**a**中获取的值。

图 18-2 获取磁盘 ID

图 18-3 获取 cluster_id

例如，执行如下命令：

```
curl -X POST https://evs.ap-southeast-1.myhuaweicloud.com:443/v2/060576866680d5762f52c0150e726aa7/volumes/69c9619d-174c-4c41-837e-31b892604e14/metadata --insecure \
  -d '{"metadata":{"cluster_id": "71e8277e-80c7-11ea-925c-0255ac100442", "namespace": "default"}}' \
  -H 'Accept:application/json' -H 'Content-Type:application/json;charset=utf8' \
  -H 'X-Auth-Token:MIIPe******IsIm1ldG
```

请求执行完成后，执行如下命令，可查看EVS盘是否已关联集群的metadata。

```
curl -X GET ${EVS_ENDPOINT}/v2/${project_id}/volumes/${volume_id}/metadata --insecure \
  -H 'X-Auth-Token:${TOKEN}'
```

例如，执行如下命令：

```
curl -X GET https://evs.ap-southeast-1.myhuaweicloud.com/v2/060576866680d5762f52c0150e726aa7/volumes/69c9619d-174c-4c41-837e-31b892604e14/metadata --insecure \
  -H 'X-Auth-Token:MIIPeAYJ***9t1c31ASaQ=='
```

在回显中就可以看到该EVS盘当前的metadata。

```
{
  "metadata": {
     "namespace": "default",
     "cluster_id": "71e8277e-80c7-11ea-925c-0255ac100442",
     "hw:passthrough": "true"
  }
}
```

----结束

18.3.4 使用 kubectl 部署带云硬盘存储卷的工作负载

操作场景

云硬盘创建或导入CCE后，可以在工作负载中挂载云硬盘。

须知

云硬盘不支持跨可用区挂载。在挂载前，您可以使用 **kubectl get pvc** 命令查询当前集群所在分区下可用PVC。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“evs-deployment-example.yaml”的创建无状态工作负载的yaml文件。

touch evs-deployment-example.yaml

vi evs-deployment-example.yaml

在无状态工作负载中基于pvc共享式使用云硬盘存储示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: evs-deployment-example
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: evs-deployment-example
  template:
    metadata:
      labels:
        app: evs-deployment-example
    spec:
      containers:
      - image: nginx
        name: container-0
        volumeMounts:
        - mountPath: /tmp
          name: pvc-evs-example
      imagePullSecrets:
        - name: default-secret
      restartPolicy: Always
      volumes:
      - name: pvc-evs-example
        persistentVolumeClaim:
          claimName: pvc-evs-auto-example
```

表 18-18 关键参数说明

前置路径	参数	描述
spec.template.spec.containers.volumeMounts	name	容器内挂载卷的名称。
spec.template.spec.containers.volumeMounts	mountPath	容器内挂载路径，示例中挂载到“/tmp”路径。
spec.template.spec.volumes	name	卷的名称。
spec.template.spec.volumes.persistentVolumeClaim	claimName	已有PVC名称。

说明

“spec.template.spec.containers.volumeMounts.name ”和“spec.template.spec.volumes.name”有映射关系，必须保持一致。

在有状态工作负载中基于PVCTemplate独占式使用云硬盘存储。

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: deploy-evs-sas-in
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deploy-evs-sata-in
  template:
    metadata:
      labels:
        app: deploy-evs-sata-in
        failure-domain.beta.kubernetes.io/region: ap-southeast-1
        failure-domain.beta.kubernetes.io/zone: ap-southeast-1a
    spec:
      containers:
        - name: container-0
          image: 'nginx:1.12-alpine-perl'
          volumeMounts:
            - name: bs-sas-mountoptionpvc
              mountPath: /tmp
      imagePullSecrets:
        - name: default-secret
  volumeClaimTemplates:
    - metadata:
        name: bs-sas-mountoptionpvc
        annotations:
          volume.beta.kubernetes.io/storage-class: sas
          volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxivol
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
  serviceName: wwww
```

表 18-19 关键参数说明

前置路径	参数	描述
metadata	name	创建的工作负载名称。
spec.template.spec.containers	image	工作负载的镜像。
spec.template.spec.containers.volumeMount	mountPath	容器内挂载路径，示例中挂载到“/tmp”路径。
spec	serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。

说明

“spec.template.spec.containers.volumeMounts.name ”和
“spec.volumeClaimTemplates.metadata.name”有映射关系，必须保持一致。

步骤3 执行如下命令创建Pod。

kubectl create -f evs-deployment-example.yaml

创建完成后，登录CCE控制台，在左侧导航栏中选择“存储管理 > 云硬盘存储卷”。单击PVC名称，在PVC详情页面可查看云硬盘和PVC的绑定关系。

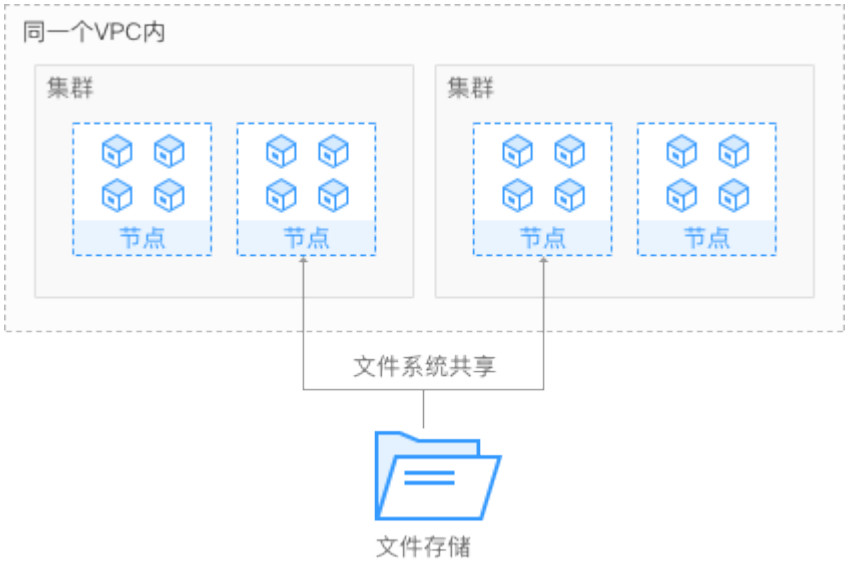
----结束

18.4 极速文件存储卷

18.4.1 极速文件存储卷概述

CCE支持将极速文件存储（SFS Turbo）创建的存储卷挂载到容器的某一路径下，以满足数据持久化的需求，极速文件存储具有按需申请，快速供给，弹性扩展，方便灵活等特点，适用于DevOps、容器微服务、企业办公等应用场景。

图 18-4 CCE 挂载极速文件存储卷



使用说明

- **符合标准文件协议：**用户可以将文件系统挂载给服务器，像使用本地文件目录一样。
- **数据共享：**多台服务器可挂载相同的文件系统，数据可以共享操作和访问。
- **私有网络：**数据访问必须在数据中心内部网络中。
- **安全隔离：**直接使用云上现有IAAS服务构建独享的云文件存储，为租户提供数据隔离保护和IOPS性能保障。
- **应用场景：**适用于多读多写（ReadWriteMany）场景下的各种工作负载（Deployment/StatefulSet）、守护进程集（DaemonSet）和普通任务（Job）使用，主要面向高性能网站、日志存储、DevOps、企业办公等场景。

18.4.2 使用 kubectl 对接已有极速文件存储卷

操作场景

CCE支持使用已有的极速文件存储来创建PersistentVolume，创建成功后，通过创建相应的PersistentVolumeClaim来绑定当前的PersistentVolume使用。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 登录SFS控制台，创建一个文件存储，记录文件存储的ID、共享路径和容量。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建两个yaml文件，用于创建PersistentVolume（PV）、PersistentVolumeClaim（PVC）。假设文件名分别为**pv-efs-example.yaml**、**pvc-efs-example.yaml**。

touch pv-efs-example.yaml pvc-efs-example.yaml

- **PV yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-efs-example
  annotations:
    pv.kubernetes.io/provisioned-by: flexvolume-huawei.com/fuxiefs
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 100Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc-efs-example
    namespace: default
  flexVolume:
    driver: huawei.com/fuxiefs
    fsType: efs
    options:
      deviceMountPath: <your_deviceMountPath>  #您的极速文件存储共享路径
      fsType: efs
      volumeID: 8962a2a2-a583-4b7f-bb74-fe76712d8414
  persistentVolumeReclaimPolicy: Delete
  storageClassName: efs-standard
```

表 18-20 关键参数说明

参数	描述
driver	挂载依赖的存储驱动，极速文件存储配置为“huawei.com/fuxiefs”。
deviceMountPath	极速文件存储的共享路径。

参数	描述
volumeID	极速文件存储的ID。 获取方法：在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“极速文件存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。
storage	文件存储的大小。
storageClassName	极速文件存储支持的卷类型，当前支持efs-standard、efs-performance（目前SFS Turbo不支持动态创建，所以此参数后续没有使用）。
spec.claimRef.apiVersion	固定值"v1"。
spec.claimRef.kind	固定值"PersistentVolumeClaim"。
spec.claimRef.name	与下一步创建的pvc的name一致。
spec.claimRef.namespace	与下一步创建的pvc的namespace一致。

- **PVC yaml文件配置示例如下：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: efs-standard
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxiefs
  name: pvc-efs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  volumeName: pv-efs-example
```

表 18-21 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	文件存储支持的读写方式，支持efs-standard、efs-performance。必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxiefs。
storage	存储容量，单位Gi，必须和已有pv的storage大小保持一致。
volumeName	PV的名称。

说明

极速文件存储所在VPC，子网必须与工作负载规划部署的ECS虚拟机的VPC保持一致，安全组开放入方向端口(111、445、2049、2051、20048)。

步骤4 创建PV。

kubectl create -f pv-efs-example.yaml

步骤5 创建PVC。

kubectl create -f pvc-efs-example.yaml

----结束

18.4.3 使用 kubectl 部署带极速文件存储卷的无状态工作负载

操作场景

极速文件存储创建或导入CCE后，可以在工作负载中挂载极速文件存储。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“efs-deployment-example.yaml”的创建deployment的yaml文件。

touch efs-deployment-example.yaml

vi efs-deployment-example.yaml

在无状态工作负载中基于pvc共享式使用极速文件存储示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: efs-deployment-example                    # 工作负载名称
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: efs-deployment-example
  template:
    metadata:
      labels:
        app: efs-deployment-example
    spec:
      containers:
      - image: nginx
        name: container-0
        volumeMounts:
        - mountPath: /tmp                         # 挂载路径
          name: pvc-efs-example
      restartPolicy: Always
      imagePullSecrets:
        - name: default-secret
```



```
  volumes:
  - name: pvc-efs-example
    persistentVolumeClaim:
      claimName: pvc-sfs-auto-example              # 挂载PVC
```

表 18-22 关键参数说明

参数	描述
name	为创建的无状态工作负载名称。
app	为无状态工作负载名称。
mountPath	为容器内挂载路径，此处示例中为“/tmp”。

说明

“spec.template.spec.containers.volumeMounts.name”和“spec.template.spec.volumes.name”有映射关系，必须保持一致。

步骤3 执行如下命令创建Pod。

kubectl create -f efs-deployment-example.yaml

创建完成后，在CCE界面“存储管理 > 极速文件存储卷”中单击PVC名称，在PVC详情页面可查看极速文件存储服务和PVC的绑定关系。

----结束

18.4.4 使用 kubectl 部署带极速文件存储卷的有状态工作负载

操作场景

CCE支持使用已有的极速文件存储（SFS Turbo），创建有状态工作负载（StatefulSet）。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 参照创建文件存储卷中操作创建极速文件存储卷，记录极速文件存储卷名称。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建一个文件，用于创建工作负载。假设文件名为**efs-statefulset-example.yaml**。

touch efs-statefulset-example.yaml

vi efs-statefulset-example.yaml

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
```



```
  name: efs-statefulset-example
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: efs-statefulset-example
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        pod.alpha.kubernetes.io/initialized: 'true'
      labels:
        app: efs-statefulset-example
    spec:
      containers:
        - image: 'nginx:1.0.0'
          name: container-0
          resources:
            requests: {}
            limits: {}
          env:
            - name: PAAS_APP_NAME
              value: efs-statefulset-example
            - name: PAAS_NAMESPACE
              value: default
            - name: PAAS_PROJECT_ID
              value: b18296881cc34f929baa8b9e95abf88b
          volumeMounts:
            - name: efs-statefulset-example
              mountPath: /tmp
              readOnly: false
              subPath: ''
      imagePullSecrets:
        - name: default-secret
      terminationGracePeriodSeconds: 30
      volumes:
        - persistentVolumeClaim:
            claimName: cce-efs-import-jnr481gm-3y5o
          name: efs-statefulset-example
      affinity: {}
      tolerations:
        - key: node.kubernetes.io/not-ready
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300
        - key: node.kubernetes.io/unreachable
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300
  podManagementPolicy: OrderedReady
  serviceName: test
  updateStrategy:
    type: RollingUpdate
```

表 18-23 关键参数说明

参数	描述
replicas	实例数。
name	新建工作负载的名称。
image	新建工作负载使用的镜像。
mountPath	容器内挂载路径。

参数	描述
serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。
claimName	已有PVC名称。

说明

spec.template.spec.containers.volumeMounts.name和spec.template.spec.volumes.name有映射关系，必须保持一致。

步骤4 创建有状态工作负载。

kubectl create -f efs-statefulset-example.yaml

----结束

18.5 对象存储卷

18.5.1 对象存储卷概述

为满足数据持久化的需求，CCE支持将对象存储服务（OBS）创建的存储卷挂载到容器的某一路径下，对象存储适用于云工作负载、数据分析、内容分析和热点对象等场景。

图 18-5 CCE 挂载对象存储卷

约束限制

安全容器不支持使用对象存储卷。

OBS限制单用户创建100个桶，但是CCE使用OBS桶为单个工作负载挂载一个桶，当工作负载数量较多时，容易导致桶数量超过限制，OBS桶无法创建。建议此种场景下直接通过OBS的API或SDK使用OBS，不在工作负载中挂载OBS桶。

存储类别

对象存储提供了三种存储类别：标准存储、低频访问存储、归档存储，从而满足客户业务对存储性能、成本的不同诉求。

- 标准存储：访问时延低和吞吐量高，因而适用于有大量热点文件（平均一个月多次）或小文件（小于1MB），且需要频繁访问数据的业务场景，例如：大数据、移动应用、热点视频、社交图片等场景。
- 低频访问存储：适用于不频繁访问（平均一年少于12次）但在需要时也要求快速访问数据的业务场景，例如：文件同步/共享、企业备份等场景。与标准存储相比，低频访问存储有相同的数据持久性、吞吐量以及访问时延，且成本较低，但是可用性略低于标准存储。
- 归档存储：适用于很少访问（平均一年访问一次）数据的业务场景，例如：数据归档、长期备份等场景。归档存储安全、持久且成本极低，可以用来替代磁带库。为了保持成本低廉，数据取回时间可能长达数分钟到数小时不等。

使用说明

- **标准接口：**具备标准Http Restful API接口，用户必须通过编程或第三方工具访问对象存储。
- **数据共享：**服务器、嵌入式设备、IOT设备等所有调用相同路径，均可访问共享的对象存储数据。
- **公共/私有网络：**对象存储数据允许在公网访问，满足互联网应用需求。
- **容量与性能：**容量无限制，性能较高（IO读写时延10ms级）。
- **应用场景：**适用于（基于OBS界面、OBS工具、OBS SDK等）的一次上传共享多读（ReadOnlyMany）的各种工作负载（Deployment/StatefulSet）和普通任务（Job）使用，主要面向大数据分析、静态网站托管、在线视频点播、基因测序、智能视频监控、备份归档、企业云盘（网盘）等场景。

相关参考

CCE支持挂载第三方租户的OBS桶，包含OBS并行文件系统（优先）和OBS对象桶，使用方法请参见**挂载第三方租户的对象存储**。

18.5.2 使用 kubectl 自动创建对象存储

操作场景

动态使用OBS可以自动创建并挂载所期望的OBS对象存储，目前支持标准、低频两种类型的桶，分别对应obs-standard、obs-standard-ia。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“pvc-obs-auto-example.yaml”的创建PVC的yaml文件。

touch pvc-obs-auto-example.yaml

vi pvc-obs-auto-example.yaml

yaml示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: obs-standard  # 对象存储桶类型，当前支持标准（obs-standard）和低频（obs-standard-ia）
  name: pvc-obs-auto-example  # PVC名称
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi  # 存储容量，单位为Gi，对OBS桶来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi
```

表 18-24 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	桶类型，当前支持标准（obs-standard）和低频（obs-standard-ia）两种桶。
name	创建的PVC名称。
accessModes	只支持ReadWriteMany，不支持ReadWriteOnly。
storage	存储容量，单位为Gi，对OBS桶来说，此处仅为校验需要（不能为空和0），设置的大小不起作用，此处设定为固定值1Gi。

步骤3 执行如下命令创建PVC。

kubectl create -f pvc-obs-auto-example.yaml

命令执行完成后会在集群所在VPC内创建一个对象存储桶，您可以在“存储管理 > 对象存储卷”中单击桶名称查看该桶，也可以在OBS的控制台查看该桶。

----结束

18.5.3 使用 kubectl 对接已有对象存储

操作场景

CCE支持使用已有的对象存储来创建PersistentVolume，并通过创建对应PersistentVolumeClaim绑定当前PersistentVolume使用。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 登录OBS控制台，创建对象存储桶，记录桶名称和存储类型。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建两个yaml文件，用于创建PersistentVolume（PV）、PersistentVolumeClaim（PVC），假设文件名为**pv-obs-example.yaml**、**pvc-obs-example.yaml**。

touch pv-obs-example.yaml pvc-obs-example.yaml

K8s集群版本（K8s version）	说明	yaml示例
1.11 ≤ K8s version ≤ 1.13	1.11以上及1.13版本集群	请参见**1.11~1.13 yaml文件配置示例**
K8s version = 1.9	1.9版本集群	请参见**1.9 yaml文件配置示例**

1.11 ≤ K8s version ≤ 1.13（1.11以上及1.13版本集群）

- **PV yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: flexvolume-huawei.com/fuxiobs
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: pvc-obs-example
    namespace: default
  flexVolume:
    driver: huawei.com/fuxiobs
    fsType: obs
    options:
      fsType: obs
      region: ap-southeast-1
      storage_class: STANDARD
      volumeID: test-obs
  persistentVolumeReclaimPolicy: Delete
  storageClassName: obs-standard
```

表 18-25 关键参数说明

参数	描述
driver	挂载依赖的存储驱动，对象存储配置为“huawei.com/fuxiobs”。
storage_class	存储类型，包括STANDARD（标准桶）、STANDARD_IA（低频访问桶）。
region	集群所在的region。

参数	描述
volumeID	对象存储的桶名称。 获取方法：在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“对象存储卷”页签下单击PVC的名称，在PVC详情页的“PV详情”页签下复制“PV名称”后的内容即可。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。
storageClassName	对象存储支持的存储类型，包括obs-standard（标准）、obs-standard-ia（低频）。
spec.claimRef.apiVersion	固定值"v1"。
spec.claimRef.kind	固定值"PersistentVolumeClaim"。
spec.claimRef.name	与下一步创建的pvc的name一致。
spec.claimRef.namespace	与下一步创建的pvc的namespace一致。

- **PVC yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: obs-standard
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxiobs
  name: pvc-obs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeName: pv-obs-example
```

表 18-26 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	对象存储支持的存储类型，包括obs-standard、obs-standard-ia。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxiobs。
volumeName	PV的名称。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。

K8s version = 1.9（1.9版本集群）

- **PV yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  flexVolume:
    driver: huawei.com/fuxiobs
    fsType: obs
    options:
      fsType: obs
      kubernetes.io/namespace: default
      region: ap-southeast-1
      storage_class: STANDARD
      volumeID: test-obs
  persistentVolumeReclaimPolicy: Delete
  storageClassName: obs-standard
```

表 18-27 关键参数说明

参数	描述
driver	挂载依赖的存储驱动，对象存储配置为“huawei.com/fuxiobs”。
storage_class	存储类型，包括STANDARD（标准桶）、STANDARD_IA（低频访问桶）。
region	集群所在的region。
volumeID	对象存储的桶名称。 获取方法：在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“对象存储卷”页签下单击PVC的名称，在PVC详情页的“PV详情”页签下复制“PV名称”后的内容即可。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。
storageClassName	对象存储支持的存储类型，包括obs-standard（标准）、obs-standard-ia（低频）。

- **PVC yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: obs-standard
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxiobs
  name: pvc-obs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeName: pv-obs-example
  volumeNamespace: default
```


表 18-28 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	对象存储支持的存储类型，包括obs-standard、obs-standard-ia。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxiobs。
volumeName	PV的名称。
storage	存储容量，单位为Gi。此处配置为固定值1Gi。

步骤4 创建PV。

kubectl create -f pv-obs-example.yaml

步骤5 创建PVC。

kubectl create -f pvc-obs-example.yaml

----结束

18.5.4 使用 kubectl 部署带对象存储卷的无状态工作负载

操作场景

对象存储卷创建或导入CCE后，可以在工作负载中挂载对象存储卷。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“obs-deployment-example.yaml”的创建Pod的yaml文件。

touch obs-deployment-example.yaml

vi obs-deployment-example.yaml

在无状态工作负载中基于pvc共享式使用对象存储示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: obs-deployment-example                  # 工作负载名称
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: obs-deployment-example
  template:
    metadata:
      labels:
```



```
      app: obs-deployment-example
  spec:
    containers:
    - image: nginx
      name: container-0
      volumeMounts:
      - mountPath: /tmp                    # 挂载路径
        name: pvc-obs-example
    restartPolicy: Always
    imagePullSecrets:
      - name: default-secret
    volumes:
    - name: pvc-obs-example
      persistentVolumeClaim:
        claimName: pvc-obs-auto-example       # PVC名称
```

表 18-29 关键参数说明

参数	描述
name	创建的Pod名称。
app	Pod工作负载名称。
mountPath	容器内挂载路径。

说明

“spec.template.spec.containers.volumeMounts.name”和“spec.template.spec.volumes.name”有映射关系，必须保持一致。

在有状态工作负载中基于PVCTemplate独占式使用对象存储。

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: deploy-obs-standard-in
  namespace: default
  generation: 1
  labels:
    appgroup: ''
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deploy-obs-standard-in
  template:
    metadata:
      labels:
        app: deploy-obs-standard-in
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        pod.alpha.kubernetes.io/initialized: 'true'
    spec:
      containers:
        - name: container-0
          image: 'nginx:1.12-alpine-perl'
          env:
            - name: PAAS_APP_NAME
              value: deploy-obs-standard-in
            - name: PAAS_NAMESPACE
              value: default
            - name: PAAS_PROJECT_ID
```



```
            value: a2cd8e998dca42e98a41f596c636dbda
        resources: {}
        volumeMounts:
          - name: obs-bs-standard-mountoptionpvc
            mountPath: /tmp
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        imagePullPolicy: IfNotPresent
    restartPolicy: Always
    terminationGracePeriodSeconds: 30
    dnsPolicy: ClusterFirst
    securityContext: {}
    imagePullSecrets:
      - name: default-secret
    affinity: {}
    schedulerName: default-scheduler
volumeClaimTemplates:
  - metadata:
      name: obs-bs-standard-mountoptionpvc
      annotations:
        volume.beta.kubernetes.io/storage-class: obs-standard
        volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxiobs
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 1Gi
serviceName: wwww
podManagementPolicy: OrderedReady
updateStrategy:
  type: RollingUpdate
revisionHistoryLimit: 10
```

表 18-30 关键参数说明

参数	描述
name	创建的工作负载名称。
image	工作负载的镜像。
mountPath	容器内挂载路径，示例中挂载到“/tmp”路径。
serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。

说明

“spec.template.spec.containers.volumeMounts.name ”和
“spec.volumeClaimTemplates.metadata.name”有映射关系，必须保持一致。

步骤3 执行如下命令创建Pod。

kubectl create -f obs-deployment-example.yaml

创建完成后，在CCE界面“存储管理 > 对象存储卷”中单击PVC名称，在PVC详情页面可查看对象存储服务和PVC的绑定关系。

----结束

18.5.5 使用 kubectl 部署带对象存储卷的有状态工作负载

操作场景

CCE支持使用已有的对象存储卷（PersistentVolumeClaim），创建有状态工作负载（StatefulSet）。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 参照**使用kubectl自动创建对象存储**中操作创建对象存储卷，并获取PVC名称。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建一个YAML文件，用于创建工作负载。假设文件名为**obs-statefulset-example.yaml**。

touch obs-statefulset-example.yaml

vi obs-statefulset-example.yaml

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: obs-statefulset-example
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: obs-statefulset-example
  serviceName: qwqq
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        pod.alpha.kubernetes.io/initialized: "true"
      creationTimestamp: null
      labels:
        app: obs-statefulset-example
    spec:
      affinity: {}
      containers:
        image: nginx:latest
        imagePullPolicy: Always
        name: container-0
        volumeMounts:
        - mountPath: /tmp
          name: pvc-obs-example
      imagePullSecrets:
      - name: default-secret
      volumes:
        - name: pvc-obs-example
          persistentVolumeClaim:
            claimName: cce-obs-demo
```


表 18-31 关键参数说明

参数	描述
replicas	实例数。
name	新建工作负载的名称。
image	新建工作负载使用的镜像。
mountPath	容器内挂载路径。
serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。
claimName	已有PVC名称。

步骤4 创建有状态工作负载。

kubectl create -f obs-statefulset-example.yaml

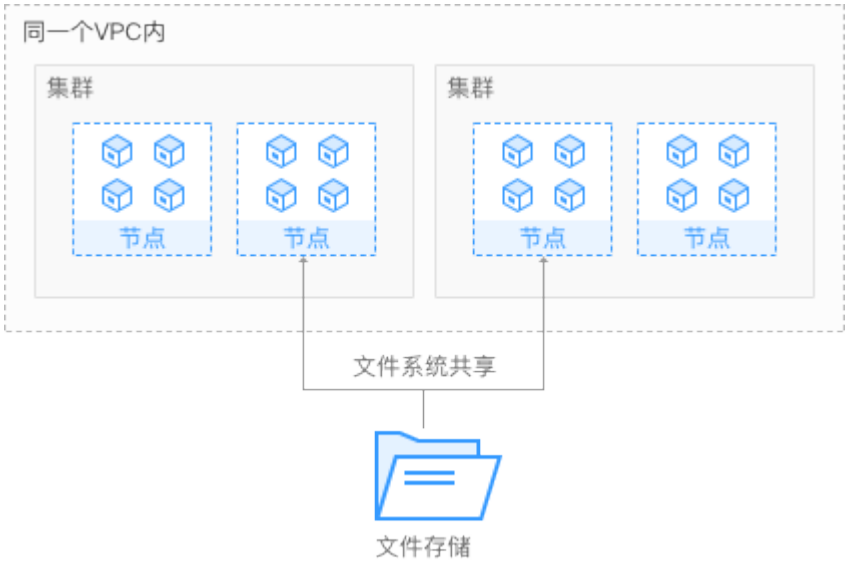
----结束

18.6 文件存储卷

18.6.1 文件存储卷概述

CCE支持将弹性文件存储（SFS）创建的存储卷挂载到容器的某一路径下，以满足数据持久化需求，SFS存储卷适用于多读多写的持久化存储，适用场景包括：媒体处理、内容管理、大数据分析和分析工作负载程序等。

图 18-6 CCE 挂载文件存储卷



使用说明

- **符合标准文件协议：**用户可以将文件系统挂载给服务器，像使用本地文件目录一样。
- **数据共享：**多台服务器可挂载相同的文件系统，数据可以共享操作和访问。
- **私有网络：**数据访问必须在数据中心内部网络中。
- **容量与性能：**单文件系统容量较高（PB级），性能极佳（IO读写时延ms级）。
- **应用场景：**适用于多读多写（ReadWriteMany）场景下的各种工作负载（Deployment/StatefulSet）和普通任务（Job）使用，主要面向高性能计算、媒体处理、内容管理和Web服务、大数据和分析应用程序等场景。

详情请参见**弹性文件服务产品介绍**。

18.6.2 使用 kubectl 自动创建文件存储

说明

当前SFS文件存储处于售罄状态，暂时无法使用存储类自动创建PVC。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“pvc-sfs-auto-example.yaml”的创建PVC的yaml文件。

touch pvc-sfs-auto-example.yaml

vi pvc-sfs-auto-example.yaml

yaml文件配置示例如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: nfs-rw
  name: pvc-sfs-auto-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

表 18-32 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	文件存储类型，当前支持标准文件协议类型（nfs-rw）。
name	创建的PVC名称。

参数	描述
accessModes	只支持ReadWriteMany，不支持ReadWriteOnly。
storage	存储容量，单位为Gi。

步骤3 执行如下命令创建PVC。

kubectl create -f pvc-sfs-auto-example.yaml

命令执行完成后会在集群所在VPC内创建一个文件存储，您可以在“存储管理 > 文件存储卷”中查看该文件系统，也可以在SFS的控制台查看该文件系统。

----结束

18.6.3 使用 kubectl 对接已有文件存储

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 登录SFS控制台，创建一个文件存储，记录文件存储的ID、共享路径和容量。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建两个yaml文件，用于创建PersistentVolume（PV）、PersistentVolumeClaim（PVC）。假设文件名分别为**pv-sfs-example.yaml**、**pvc-sfs-example.yaml**。

touch pv-sfs-example.yaml pvc-sfs-example.yaml

K8s集群版本（K8s version）	说明	yaml示例
1.11 ≤ K8s version < 1.13	1.11以上及1.13版本集群	请参见**1.11~1.13 yaml文件配置示例**
K8s version = 1.9	1.9版本集群	请参见**1.9 yaml文件配置示例**

1.11 ≤ K8s version ≤ 1.13（1.11以上及1.13版本集群）

- **PV yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-sfs-example
  annotations:
    pv.kubernetes.io/provisioned-by: flexvolume-huawei.com/fuxinfs
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
```



```
claimRef:
  apiVersion: v1
  kind: PersistentVolumeClaim
  name: pvc-sfs-example
  namespace: default
flexVolume:
  driver: huawei.com/fuxinfs
  fsType: nfs
  options:
    deviceMountPath: <your_deviceMountPath>  #您的文件存储共享路径
    fsType: nfs
    volumeID: f6976f9e-2493-419b-97ca-d7816008d91c
persistentVolumeReclaimPolicy: Delete
storageClassName: nfs-rw
```

表 18-33 关键参数说明

参数	描述
driver	挂载依赖的存储驱动，文件存储配置为“huawei.com/fuxinfs”。
deviceMountPath	文件存储的共享路径。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，在弹性文件服务列表中可以看到“挂载地址”列，即为文件存储的共享路径，如图18-7。
volumeID	文件存储的ID。 获取方法：在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“文件存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。
storage	文件存储的大小。
storageClassName	文件存储支持的读写方式，当前支持nfs-rw、nfs-ro。
spec.claimRef.apiVersion	固定值"v1"。
spec.claimRef.kind	固定值"PersistentVolumeClaim"。
spec.claimRef.name	与下一步创建的pvc的name一致。
spec.claimRef.namespace	pvc的namespace；是下一步创建的pvc的namespace一致。

- **PVC yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: nfs-rw
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxinfs
  name: pvc-sfs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
```



```
    storage: 10Gi
volumeName: pv-sfs-example
```

表 18-34 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	文件存储支持的读写方式，支持nfs-rw 、nfs-ro。必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxinfs。
storage	存储容量，单位Gi，必须和已有pv的storage大小保持一致。
volumeName	PV的名称。

K8s version = 1.9（1.9版本集群）

- **PV yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-sfs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  flexVolume:
    driver: huawei.com/fuxinfs
    fsType: nfs
    options:
      deviceMountPath: <your_deviceMountPath>  #您的文件存储共享路径
      fsType: nfs
      kubernetes.io/namespace: default
      volumeID: f6976f9e-2493-419b-97ca-d7816008d91c
  persistentVolumeReclaimPolicy: Delete
  storageClassName: nfs-rw
```

表 18-35 关键参数说明

参数	描述
driver	挂载依赖的存储驱动，文件存储配置为“huawei.com/fuxinfs”。
deviceMountPath	文件存储的共享路径。 获取方法：在CCE控制台，单击顶部的“服务列表 > 存储 > 弹性文件服务”，在弹性文件服务列表中可以看到“挂载地址”列，即为文件存储的共享路径，如图18-7。
volumeID	文件存储的ID。 获取方法：在CCE控制台，单击左侧栏目树中的“资源管理-存储管理”，在“文件存储卷”页签下单击PVC的名称，在PVC详情页中复制“PVC UID”后的内容即可。

参数	描述
storage	文件存储的大小。
storageClassName	文件存储支持的读写方式，当前支持nfs-rw、nfs-ro。

- **PVC yaml文件配置示例：**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-class: nfs-rw
    volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxinfs
  name: pvc-sfs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-sfs-example
  volumeNamespace: default
```

表 18-36 关键参数说明

参数	描述
volume.beta.kubernetes.io/storage-class	文件存储支持的读写方式，支持nfs-rw 、nfs-ro。必须和已有PV保持一致。
volume.beta.kubernetes.io/storage-provisioner	必须使用flexvolume-huawei.com/fuxinfs。
storage	存储容量，单位Gi，必须和已有pv的storage大小保持一致。
volumeName	PV的名称。

图 18-7 弹性文件存储-共享路径

说明

文件存储所在VPC必须与工作负载规划部署的ECS虚拟机的VPC保持一致。

步骤4 创建PV。

kubectl create -f pv-sfs-example.yaml

步骤5 创建PVC。

kubectl create -f pvc-sfs-example.yaml

----结束

18.6.4 使用 kubectl 部署带文件存储卷的无状态工作负载

操作场景

文件存储卷创建或导入CCE后，可以在工作负载中挂载文件存储卷。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤2 执行如下命令，配置名为“sfs-deployment-example.yaml”的创建Pod的yaml文件。

touch sfs-deployment-example.yaml

vi sfs-deployment-example.yaml

在无状态工作负载中基于pvc共享式使用文件存储示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sfs-deployment-example                  # 工作负载名称
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sfs-deployment-example
  template:
    metadata:
      labels:
        app: sfs-deployment-example
    spec:
      containers:
      - image: nginx
        name: container-0
        volumeMounts:
        - mountPath: /tmp                       # 挂载路径
          name: pvc-sfs-example
      imagePullSecrets:
        - name: default-secret
      restartPolicy: Always
      volumes:
      - name: pvc-sfs-example
        persistentVolumeClaim:
          claimName: pvc-sfs-auto-example       # 挂载PVC
```

表 18-37 关键参数说明

前置路径	参数	描述
metadata	name	创建的Pod名称。

前置路径	参数	描述
spec.template.spec.containers.volumeMounts	mountPath	容器内挂载路径，此处示例中为“/tmp”。
spec.template.spec.volumes.persistentVolumeClaim	claimName	已有PVC名称。

说明

“spec.template.spec.containers.volumeMounts.name”和“spec.template.spec.volumes.name”有映射关系，必须保持一致。

在有状态工作负载中基于PVCTemplate独占式使用文件存储。

说明

当前SFS文件存储处于售罄状态，暂时不支持PVCTemplate独占式使用文件存储。

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: deploy-sfs-nfs-rw-in
  namespace: default
  labels:
    appgroup: ''
spec:
  replicas: 2
  selector:
    matchLabels:
      app: deploy-sfs-nfs-rw-in
  template:
    metadata:
      labels:
        app: deploy-sfs-nfs-rw-in
    spec:
      containers:
        - name: container-0
          image: 'nginx:1.12-alpine-perl'
          volumeMounts:
            - name: bs-nfs-rw-mountoptionpvc
              mountPath: /aaa
      imagePullSecrets:
        - name: default-secret
  volumeClaimTemplates:
    - metadata:
        name: bs-nfs-rw-mountoptionpvc
        annotations:
          volume.beta.kubernetes.io/storage-class: nfs-rw
          volume.beta.kubernetes.io/storage-provisioner: flexvolume-huawei.com/fuxinfs
      spec:
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 1Gi
  serviceName: wwww
```


表 18-38 关键参数说明

前置路径	参数	描述
metadata	name	创建的工作负载名称。
spec.template.spec.containers	image	工作负载的镜像。
spec.template.spec.containers.volumeMount	mountPath	容器内挂载路径，此处示例中为“/tmp”。
spec	serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。

说明

“spec.template.spec.containers.volumeMounts.name ”和“spec.volumeClaimTemplates.metadata.name”有映射关系，必须保持一致。

步骤3 执行如下命令创建Pod。

kubectl create -f sfs-deployment-example.yaml

创建完成后，登录CCE控制台，在左侧导航栏中选择“存储管理 > 文件存储卷”。单击PVC名称，在PVC详情页面可查看文件存储服务和PVC的绑定关系。

----结束

18.6.5 使用 kubectl 部署带文件存储卷的有状态工作负载

操作场景

CCE支持使用已有的文件存储（PersistentVolumeClaim），创建有状态工作负载（StatefulSet）。

约束与限制

如下配置示例适用于Kubernetes 1.13及以下版本的集群。

操作步骤

步骤1 参照**使用kubectl自动创建文件存储**中操作创建文件存储卷，记录文件存储卷名称。

步骤2 请参见**通过kubectl连接集群**，使用kubectl连接集群。

步骤3 新建一个YAML文件，用于创建工作负载。假设文件名为**sfs-statefulset-example.yaml**。

touch sfs-statefulset-example.yaml

vi sfs-statefulset-example.yaml

yaml示例如下：

```
apiVersion: apps/v1
kind: StatefulSet
```



```
metadata:
  name: sfs-statefulset-example
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sfs-statefulset-example
  serviceName: qwqq
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
        pod.alpha.kubernetes.io/initialized: "true"
      labels:
        app: sfs-statefulset-example
    spec:
      affinity: {}
      containers:
      - image: nginx:latest
        name: container-0
        volumeMounts:
        - mountPath: /tmp
          name: pvc-sfs-example
      imagePullSecrets:
      - name: default-secret
      volumes:
        - name: pvc-sfs-example
          persistentVolumeClaim:
            claimName: cce-sfs-demo
```

表 18-39 关键参数说明

前置路径	参数	描述
spec	replicas	实例数。
metadata	name	新建工作负载的名称。
spec.template.spec.containers	image	新建工作负载使用的镜像。
spec.template.spec.containers.volumeMounts	mountPath	容器内挂载路径。
spec	serviceName	工作负载对应的服务，服务创建过程请参见**创建有状态负载（StatefulSet）**。
spec.template.spec.volumes.persistentVolumeClaim	claimName	已有PVC名称。

说明

spec.template.spec.containers.volumeMounts.name和spec.template.spec.volumes.name有映射关系，必须保持一致。

步骤4 创建有状态工作负载。

kubectl create -f sfs-statefulset-example .yaml

----结束

