

云容器引擎

常见问题

文档版本 01
发布日期 2025-09-15



版权所有 © 华为云计算技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI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。

目录

1 高频常见问题	1
2 计费类	3
2.1 云容器引擎 CCE 如何定价/收费?	3
2.2 CCE 集群的计费方式如何由按需改为包年包月?	3
2.3 CCE 创建的节点是否支持按需转包周期?	5
2.4 华为云支持哪几种开具发票模式?	7
2.5 CCE 是否支持余额不足提醒?	7
2.6 CCE 是否支持账户余额变动提醒?	7
2.7 包周期的 CCE 集群到期可以直接删除吗?	7
2.8 如何退订我的云容器引擎?	7
2.9 集群冻结/解冻/释放常见问题	9
3 集群	10
3.1 集群创建	10
3.1.1 CCE 集群创建失败的原因与解决方法?	10
3.1.2 集群的管理规模和控制节点的数量有关系吗?	11
3.1.3 CCE 集群创建时的根证书如何更新?	11
3.1.4 使用 CCE 需要关注哪些配额限制?	12
3.2 集群运行	13
3.2.1 当集群状态为“不可用”时,如何排查解决?	13
3.2.2 CCE 集群如何重置或重装?	15
3.2.3 如何确认已创建的集群是否为多控制节点模式?	15
3.2.4 是否可以直接连接 CCE 集群的控制节点?	15
3.2.5 CCE 集群删除之后相关数据能否再次找回?	15
3.2.6 为什么 CCE 集群界面的节点磁盘监控看起来不准确?	15
3.2.7 如何修改 CCE 集群名称?	16
3.2.8 控制台访问异常问题排查	17
3.3 集群删除	18
3.3.1 集群删除失败:安全组中存在残留资源	18
3.3.2 冻结或不可用的集群删除后如何清除残留资源	19
3.4 集群升级	22
3.4.1 CCE 集群升级时,升级集群插件失败如何排查解决?	22
3.4.2 CCE 集群升级时,ELB Ingress 与 ELB 配置不一致如何处理?	24

4 节点	32
4.1 节点异常问题排查	32
4.2 节点创建	47
4.2.1 CCE 集群新增节点时的问题与排查方法？	47
4.2.2 CCE 集群纳管节点时的常见问题及排查方法？	54
4.2.3 纳管节点时失败，报错“安装节点失败”如何解决？	55
4.2.4 CCE 支持等保三级认证吗？	56
4.3 节点运行	56
4.3.1 集群可用但节点状态为“不可用”如何解决？	56
4.3.2 CCE 集群中的节点无法远程登录，如何排查解决？	63
4.3.3 如何重置 CCE 集群中节点的密码？	63
4.3.4 如何收集 CCE 集群中节点的日志？	63
4.3.5 如何解决 yum update 升级操作系统导致的容器网络不可用问题？	65
4.3.6 Node 节点 vdb 盘受损，通过重置节点仍无法恢复节点？	65
4.3.7 CCE 集群节点中安装 kubelet 的端口主要有哪些？	67
4.3.8 如何配置 Pod 使用 GPU 节点的加速能力？	67
4.3.9 容器使用 SCSI 类型云硬盘偶现 IO 卡住如何解决？	67
4.3.10 docker 审计日志量过大影响磁盘 IO 如何解决？	68
4.3.11 thinpool 磁盘空间耗尽导致容器或节点异常时，如何解决？	69
4.3.12 CCE 节点上监听的端口列表	72
4.3.13 GPU 节点使用 nvidia 驱动启动容器排查思路	74
4.3.14 CCE 节点 NTP 时间不同步如何解决？	75
4.3.15 Containerd 节点业务容器标准输出日志写入过快导致节点数据盘使用率过高	75
4.3.16 为什么 kubectl top 命令查看节点内存使用超过 100%？	76
4.3.17 CCE 节点事件中一直出现“镜像回收失败”告警如何解决？	77
4.3.18 节点磁盘利用率高的问题排查	78
4.3.19 GPU 节点显示 GPU 卡不可用时，如何解决？	81
4.3.20 CCE AI 套件（NVIDIA GPU）插件升级后，GPU 节点事件中出现告警如何解决？	84
4.4 规格配置变更	85
4.4.1 如何变更 CCE 集群中的节点规格？	85
4.4.2 CCE 节点池内的节点变更规格后会有哪些影响？	86
4.4.3 CCE 节点变更规格后，为什么无法重新拉起或创建工作负载？	86
4.4.4 CCE 集群的节点可以更改 IP 吗？	87
4.5 操作系统问题说明	88
4.5.1 低版本内核的 CentOS 节点反复创删应用时偶现 cgroup kmem 泄露问题	88
4.5.2 CCE 集群 IPVS 转发模式下 conn_reuse_mode 问题说明	89
4.5.3 cgroup 统计资源异常导致 kubelet 驱逐 Pod	91
4.5.4 低版本内核的 CentOS 节点出现容器 OOM 时，偶现 ext4 文件系统卡死问题	92
4.5.5 IPVS 缺陷导致节点升级 CoreDNS 后出现概率性解析超时	92
4.5.6 节点 ARP 表项超过限制	93
4.5.7 EulerOS 2.9 内核缺陷导致虚拟机卡住	95
5 节点池	97

5.1 节点池异常状态排查.....	97
5.2 节点池一直在扩容中但“操作记录”里为何没有创建节点的记录？	98
5.3 节点池扩容失败.....	99
5.4 节点池批量扩缩容节点时，Kubernetes Event 事件存在部分缺失.....	101
5.5 云服务器无法纳管至节点池时如何修改云服务器配置.....	102
6 工作负载.....	106
6.1 工作负载异常问题排查.....	106
6.1.1 工作负载状态异常定位方法.....	106
6.1.2 工作负载异常：实例调度失败.....	110
6.1.3 工作负载异常：实例拉取镜像失败.....	119
6.1.4 工作负载异常：启动容器失败.....	127
6.1.5 工作负载异常：实例驱逐异常（ Evicted ）	136
6.1.6 工作负载异常：存储卷无法挂载或挂载超时.....	139
6.1.7 工作负载异常：一直处于创建中.....	142
6.1.8 工作负载异常：Pod 一直处于 Terminating 状态.....	143
6.1.9 工作负载异常：已停止.....	144
6.1.10 工作负载异常：GPU 节点部署服务时报错.....	145
6.1.11 工作负载异常：添加存储失败.....	146
6.1.12 工作负载异常：实例无法写入数据.....	147
6.1.13 工作负载异常：Init 容器启动失败.....	147
6.1.14 工作负载异常：OOM 问题.....	148
6.1.15 工作负载状态正常但未正常工作.....	149
6.1.16 挂载文件存储的节点，Pod 创建删除卡死.....	150
6.1.17 容器异常退出状态码.....	150
6.1.18 Java 容器内存虚高以及 OOM 问题定位.....	154
6.1.19 集群中大量 Pod 处于 UnexpectedAdmissionError 状态怎么办.....	159
6.1.20 Pod 实例异常，显示无法找到设备文件.....	160
6.2 容器设置.....	161
6.2.1 在什么场景下设置工作负载生命周期中的“停止前处理”？	161
6.2.2 在什么场景下，容器会被重建？	161
6.2.3 在同一个命名空间内访问指定容器的 FQDN 是什么？	162
6.2.4 健康检查探针（ Liveness、Readiness ）偶现检查失败？.....	162
6.2.5 如何设置容器 umask 值？	162
6.2.6 CCE 启动实例失败时的重试机制是怎样的？	163
6.3 监控日志.....	163
6.3.1 工作负载的“事件”保存多长时间？	164
6.3.2 容器监控的内存使用率与实际弹性伸缩现象不一致.....	164
6.4 调度策略.....	164
6.4.1 如何让多个 Pod 均匀部署到各个节点上？	164
6.4.2 如何避免节点上的某个容器被驱逐？	165
6.4.3 为什么 Pod 在节点不是均匀分布？	166
6.4.4 如何驱逐节点上的所有 Pod？	167

6.4.5 如何查看 Pod 是否使用 CPU 绑核？	168
6.4.6 节点关机后 Pod 不重新调度.....	168
6.4.7 如何避免非 GPU/NPU 负载调度到 GPU/NPU 节点？	170
6.4.8 为什么 Pod 调度不到某个节点上？	171
6.4.9 修改 kubelet 参数导致已驱逐的 Pod 被重新调度.....	171
6.4.10 根据 GPU/NPU 卡信息定位使用该卡的 Pod.....	171
6.4.11 节点标签更新导致的 Pod 容器退出问题.....	173
6.4.12 创建使用 GPU 虚拟化多卡均分调度功能的工作负载之后，大量 Pod 执行失败.....	174
6.5 其他.....	176
6.5.1 定时任务停止一段时间后，为何无法重新启动？	176
6.5.2 创建有状态负载时，实例间发现服务是指什么？	177
6.5.3 CCE 容器拉取私有镜像时报错“Auth is empty”	177
6.5.4 CCE 集群中工作负载镜像的拉取策略有哪些？	178
6.5.5 鲲鹏集群 Docker 容器挂载点被卸载.....	178
6.5.6 下载镜像缺少层如何解决？	179
6.5.7 容器内的文件权限和用户都是问号.....	179
6.5.8 如何导出容器中的文件？	181
7 网络管理.....	182
7.1 网络异常问题排查.....	182
7.1.1 工作负载网络异常时，如何定位排查？	182
7.1.2 负载均衡类型 Service 异常问题排查.....	184
7.1.3 集群内部无法使用 ELB 地址访问负载.....	188
7.1.4 集群外部访问 Ingress 异常.....	192
7.1.5 CCE 集群中域名解析失败.....	197
7.1.6 为什么访问部署的应用时浏览器返回 404 错误码？	199
7.1.7 为什么容器无法连接互联网？	199
7.1.8 VPC 的子网无法删除，怎么办？	200
7.1.9 CCE Turbo 集群如何修复出现故障的容器网卡？	200
7.1.10 节点无法连接互联网（公网），如何排查定位？	203
7.1.11 如何解决 VPC 网段与容器网络冲突的问题？	203
7.1.12 ELB 四层健康检查导致 java 报错：Connection reset by peer.....	204
7.1.13 Service 事件：Have no node to bind，如何排查？	204
7.1.14 为什么登录虚拟机 VNC 界面会间歇性出现 Dead loop on virtual device gw_11cbf51a, fix it urgently?	204
7.1.15 集群节点使用 networkpolicy 概率性出现 panic 问题.....	205
7.1.16 节点远程登录界面(VNC)打印较多 source ip_type 日志问题.....	207
7.1.17 使用 IE 浏览器访问 nginx-ingress 出现重定向 308 无法访问.....	208
7.1.18 NGINX Ingress 控制器插件升级导致集群内 Nginx 类型的 Ingress 路由访问异常.....	208
7.1.19 负载均衡型 Service 更新出现错误：Quota exceeded for resources: members_per_pool.....	210
7.1.20 ELB Ingress 出现告警：Invalid input for rules.....	211
7.1.21 为 ELB Ingress 配置了 HTTPS 证书后访问异常的原因有哪些？	211
7.1.22 Nginx Ingress 中默认开启 HSTS 会导致 HTTP 请求重定向到 HTTPS.....	212

7.2 网络规划.....	213
7.2.1 集群与虚拟私有云、子网的关系是怎样的？	213
7.2.2 如何查看虚拟私有云 VPC 的网段？	214
7.2.3 如何设置 CCE 集群中的 VPC 网段和子网网段？	215
7.2.4 如何设置 CCE 集群中的容器网段？	215
7.2.5 什么是云原生网络 2.0 网络模式，适用于什么场景？	216
7.2.6 什么是弹性网卡？	217
7.2.7 集群安全组规则配置.....	218
7.2.8 创建 CCE Turbo 集群时如何设置 IPv6 服务网段.....	226
7.2.9 CCE 集群的节点是否支持绑定多网卡？	228
7.3 安全加固.....	228
7.3.1 集群节点如何不暴露到公网？	228
7.3.2 如何配置集群的访问策略.....	228
7.3.3 如何获取 TLS 密钥证书？	229
7.3.4 如何批量修改集群 node 节点安全组？	231
7.4 网络指导.....	232
7.4.1 CCE 如何与其他服务进行内网通信？	232
7.4.2 使用 CCE 设置工作负载访问方式时，端口如何填写？	233
7.4.3 Ingress 中的 property 字段如何实现与社区 client-go 兼容？	235
7.4.4 服务加入 Istio 后，如何获取客户端真实源 IP？	237
7.4.5 为什么更换命名空间后无法创建 Ingress？	238
7.4.6 服务发布到 ELB，ELB 的后端为何会被自动删除？	238
7.4.7 如何使容器重启后所在容器 IP 仍保持不变？	239
7.4.8 如何确认网卡不被集群占用？	239
7.4.9 删除子网后如何删除安全组规则？	240
7.4.10 不同命名空间下的 Ingress 共用监听器时如何同步生效的证书？	241
7.4.11 如何确认监听器配置生效的 Ingress.....	244
7.4.12 如何扩展单个 CCE Turbo 集群能支持的最大网卡数？	247
8 存储管理.....	249
8.1 如何扩容容器的存储空间？	249
8.2 CCE 支持的存储在持久化和多节点挂载方面的有什么区别？	250
8.3 创建 CCE 节点时可以不添加数据盘吗？	251
8.4 EVS 存储卷的底层存储被删除或者过期后，数据是否可以恢复？	252
8.5 公网访问 CCE 部署的服务并上传 OBS，为何报错找不到 host？	252
8.6 Pod 接口 ExtendPathMode: PodUID 如何与社区 client-go 兼容？	253
8.7 创建存储卷失败如何解决？	255
8.8 CCE 容器云存储 PVC 能否感知底层存储故障？	255
8.9 通用文件系统（原 SFS 3.0 容量型）在 OS 中的挂载点修改属组及权限报错.....	256
8.10 无法使用 kubectl 命令删除 PV 或 PVC.....	256
8.11 删除挂载了云存储的 Pod 时提示 target is busy.....	257
8.12 无法自动创建包周期的云硬盘存储卷.....	257
8.13 误卸载存储池的磁盘后如何恢复.....	258

8.14 删除动态创建的 PVC 之后，底层存储依旧残留.....	262
8.15 已释放的云硬盘重新绑定 PV 后，在 Pod 中挂载失败.....	263
9 命名空间.....	264
9.1 一个集群最多能创建多少个命名空间？	264
9.2 命名空间因 APIService 对象访问失败无法删除.....	264
9.3 如何删除 Terminating 状态的命名空间？	265
10 模板插件.....	268
10.1 插件异常问题排查.....	268
10.2 集群安装 nginx-ingress 插件失败，一直处于创建中？	269
10.3 NPD 插件版本过低导致进程资源残留问题.....	271
10.4 模板格式不正确，无法删除模板实例？	271
10.5 CCE 是否支持 nginx-ingress？	273
10.6 插件安装失败，提示 The release name is already exist 如何解决？	273
10.7 创建或升级模板失败，提示 rendered manifests contain a resource that already exists.....	275
10.8 kube-prometheus-stack 插件实例调度失败如何解决？	276
10.9 上传模板失败如何解决？	278
10.10 如何根据集群规格调整插件配额？	279
10.11 NGINX Ingress 控制器插件处于 Unknown 状态时卸载残留.....	282
10.12 NGINX Ingress 控制器插件升级后无法使用 TLS v1.0 和 v1.1.....	282
10.13 NPU 驱动升级失败如何解决？	284
10.14 NPU 插件从 1.x.x 升级到 2.x.x 后 Pod 无法启动如何解决？	286
10.15 CCE AI 套件（NVIDIA GPU）插件版本升级或回滚后，如何对 GPU 节点进行排水？	287
10.16 通过 CCE AI 套件（NVIDIA GPU）插件安装的驱动，执行编解码任务时失败.....	289
10.17 EulerOS 2.9 系统安装 NVIDIA 驱动失败.....	290
10.18 Volcano 调度器插件升级后，Volcano Job（vcjob）资源无法正常运行.....	291
11 API&kubectl.....	294
11.1 用户访问集群 API Server 的方式哪些？	294
11.2 通过 API 或 kubectl 操作 CCE 集群，创建的资源是否能在控制台展示？	294
11.3 通过 kubectl 连接集群时，其配置文件 config 如何下载？	295
11.4 kubectl top node 命令为何报错.....	295
11.5 kubectl 使用报错：Error from server (Forbidden).....	295
12 域名 DNS.....	298
12.1 CCE 集群内域名解析失败，如何定位处理？	298
12.2 为什么 CCE 集群的容器无法通过 DNS 解析？	300
12.3 为什么修改子网 DNS 配置后，无法解析租户区域名？	301
12.4 解析外部域名很慢或超时，如何优化配置？	302
12.5 如何设置容器内的 DNS 策略？	302
12.6 如何解决 CoreDNS 使用废弃 API 问题？	303
13 镜像仓库.....	305
13.1 如何制作 Docker 镜像？如何解决拉取镜像慢的问题？	305

13.2 如何上传我的镜像到 CCE 中使用?	305
14 权限.....	306
14.1 能否只配置命名空间权限，不配置集群管理权限?	306
14.2 如果不配置集群管理权限的情况下，是否可以使用 API 呢?	306
14.3 如果不配置集群管理权限，是否可以使用 kubectl 命令呢?	307
14.4 IAM 用户无法使用调用 API.....	307
14.5 什么是 OBS 全局访问密钥？如何排查集群中是否使用全局访问密钥?	308
15 相关服务.....	310
15.1 云容器引擎（CCE）与云容器实例（CCI）的区别是什么?	310
15.2 云容器引擎（CCE）和应用管理与运维平台（ServiceStage）的区别是什么?	315

1 高频常见问题

集群管理

- [CCE集群创建失败的原因与解决方法？](#)
- [集群的管理规模和控制节点的数量有关系吗？](#)
- [当集群状态为“不可用”时，如何排查解决？](#)

节点及节点池

- [集群可用但节点状态为“不可用”如何解决？](#)
- [如何收集CCE集群中节点的日志？](#)
- [thinpool磁盘空间耗尽导致容器或节点异常时，如何解决？](#)
- [纳管节点时失败，报错“安装节点失败”如何解决？](#)

工作负载

- [工作负载状态异常定位方法](#)
- [工作负载异常：实例调度失败](#)
- [工作负载异常：实例拉取镜像失败](#)
- [工作负载异常：启动容器失败](#)
- [工作负载异常：Pod一直处于Terminating状态](#)
- [工作负载异常：实例驱逐异常（Evicted）](#)
- [容器异常退出状态码](#)
- [如何让多个Pod均匀部署到各个节点上？](#)
- [如何驱逐节点上的所有Pod？](#)

网络管理

- [集群安全组规则配置](#)
- [工作负载网络异常时，如何定位排查？](#)
- [为什么访问部署的应用时浏览器返回404错误码？](#)
- [节点无法连接互联网（公网），如何排查定位？](#)
- [解析外部域名很慢或超时，如何优化配置？](#)

存储管理

- [无法使用kubectl命令删除PV或PVC](#)
- [如何扩容容器的存储空间?](#)

API&kubectl

[kubectl使用报错: Error from server \(Forbidden\)](#)

镜像仓库

[如何制作Docker镜像? 如何解决拉取镜像慢的问题?](#)

2 计费类

2.1 云容器引擎 CCE 如何定价/收费？

计费模式

云容器引擎提供包年/包月、按需计费两种计费模式，以满足不同场景下的用户需求。关于计费模式的详细介绍请参见[计费模式概述](#)。

- 包年/包月是一种预付费模式，即先付费再使用，按照订单的购买周期进行结算，因此在购买之前，您必须确保账户余额充足。
- 按需计费是一种后付费模式，即先使用再付费，按照实际使用时长计费。

在购买集群或集群内资源后，如果发现当前计费模式无法满足业务需求，您还可以变更计费模式。详细介绍请参见[变更计费模式概述](#)。

计费项

云容器引擎的计费项由集群费用和其他云服务资源费用组成。了解每种计费项的计费因子、计费公式等信息，请参考[计费项](#)。

1. **集群：**控制节点资源费用，按照每个集群的类型（虚拟机或裸金属、控制节点数）、集群规模（最大支持的节点数）的差异收取不同的费用。
控制节点资源的价格目录请参见：[云容器引擎价格目录](#)。
2. **其他云服务资源：**集群所使用的IaaS基础设施费用，包括集群创建使用过程中自动创建或手动加入的相关资源，如云服务器、云硬盘、弹性IP/带宽、负载均衡等，价格参照相应产品价格表。
更多价格目录请参见：[产品价格详情](#)。

如需了解实际场景下的计费样例以及各计费项在不同计费模式下的费用计算过程，请参见[计费样例](#)。

2.2 CCE 集群的计费方式如何由按需改为包年包月？

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按周期计费的集群。

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”。

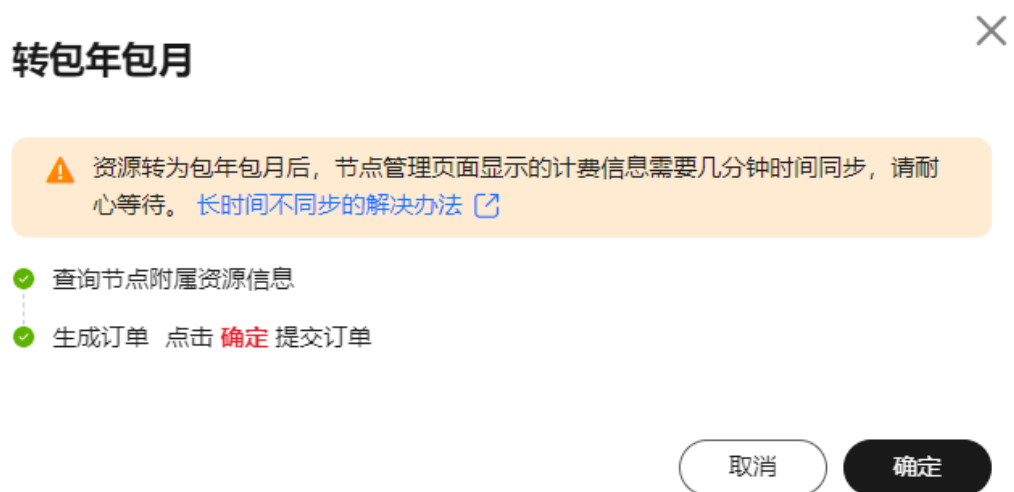
步骤3 在转包年包月页面中，选择需要转包年/包月的集群，您也可以同时选择需要转包年/包月的节点，单击“确定”。

图 2-1 按需集群转包年/包月



步骤4 等待资源查询并订单生成后，单击“确定”跳转至费用中心提交订单。

图 2-2 生成订单



步骤5 选择需要转包年包月的资源、包年包月时长及自动续费配置后，单击“去支付”完成支付即可。

图 2-3 按需转包年/包月配置



----结束

2.3 CCE 创建的节点是否支持按需转包周期？

当前在CCE中购买节点时支持“按需计费”和“包年/包月”（按周期）计费。

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、v1.23.9-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

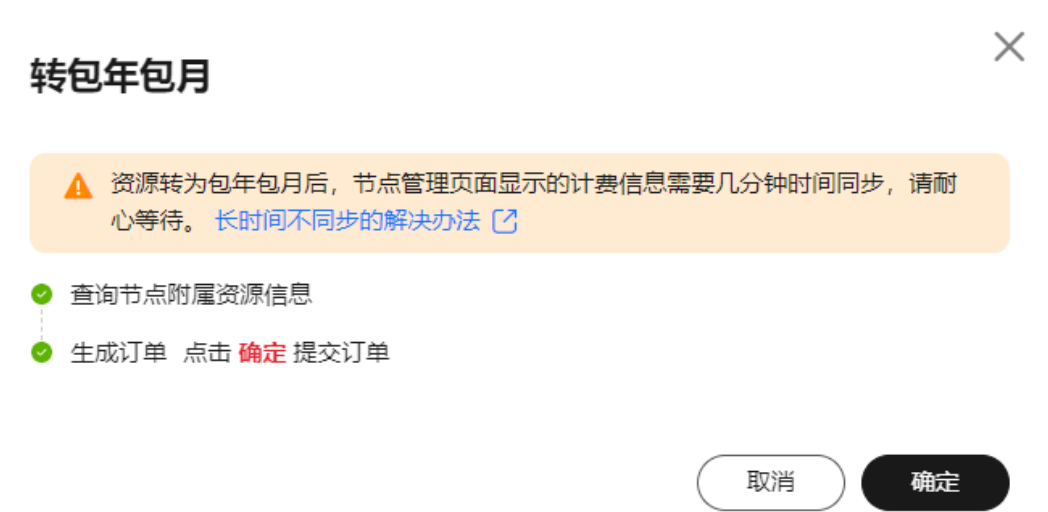
在左侧选择节点管理，在节点的操作列选择“更多 > 转包年包月”，单击“确定”。

图 2-4 按需节点转包年/包月



步骤3 等待资源查询并订单生成后，单击“确定”跳转至费用中心提交订单。

图 2-5 生成订单



步骤4 选择需要转包年包月的资源、包年包月时长及自动续费配置后，单击“去支付”完成支付即可。

图 2-6 按需转包年/包月配置



----结束

2.4 华为云支持哪几种开具发票模式？

华为云支持“按账期索取发票”和“按订单索取发票”模式。

您可在费用中心的[发票管理](#)开具发票。

2.5 CCE 是否支持余额不足提醒？

用户可在费用中心[总览](#)页面“可用额度”区域单击“设置”，设置“可用额度预警”后的开关，即可开通或关闭可用额度预警功能。单击“修改”，可以对预警阈值进行修改。

- 开通后，当可用额度（含现金余额、信用余额、通用代金券、现金券）的总金额低于预警阈值时，会每天给联系人发送短信和邮件提醒，最多连续提醒3天。
- 您可到消息中心“消息接收设置 > 财务信息 > 账户余额预警”中修改预警提醒的联系人信息。
- 企业主客户可批量为企业子客户设置预警阈值，具体操作请参见[批量设置子账号预警阈值](#)。

2.6 CCE 是否支持账户余额变动提醒？

系统会以邮件、短信形式给客户发送账户余额变动通知，包括账户余额调整、充值到账、客户在线充值等。

2.7 包周期的 CCE 集群到期可以直接删除吗？

CCE集群包周期到期后，您可以在备份好所有数据的情况下直接删除该集群。

如果到期后您仍没有续费或删除，系统会根据资源到期时间删除该集群，请及时续费并做好数据备份工作。

2.8 如何退订我的云容器引擎？

客户购买包周期资源后，支持客户退订包周期实例。退订资源实例包括资源续费部分和当前正在使用的部分，退订后资源将无法使用。退订资源实例需收取手续费。

注意事项

- 退订该实例是指退订续费部分和当前正在使用的部分，资源退订后将无法使用。
- 解决方案组合产品只支持整体退订。
- 订单中存在主从关系的资源，需分别退订。
- 资源退订，相关注意事项请参见[退订规则说明](#)。

操作步骤

⚠ 注意

- 在执行退订操作前，请确保将退订的云资源上的数据已完成备份或者迁移，退订完成后云资源将被删除，数据无法找回，请谨慎操作。
- 页面中间有关于5天无理由退订的已退订次数和剩余退订次数提示，请注意查看。

步骤1 进入“云服务退订”页面。

步骤2 单击“退订使用中的资源”页签。

步骤3 单个资源退订与批量退订可使用不同的操作方式：

- 退订单个资源：单击待退订资源所在行的“退订资源”。

图 2-7 退订单个资源



- 批量退订：在退订列表中勾选需要退订的资源，单击列表左上角的“退订资源”。

图 2-8 批量退订



步骤4 在“退订资源”页面中查看退订信息，确认无误后选择退订原因，单击“退订”。

----结束

2.9 集群冻结/解冻/释放常见问题

集群为什么被冻结

资源冻结的类型包括欠费冻结、违规冻结、公安冻结。

单击了解[资源冻结的类型、冻结后对续费、退订的影响](#)。

集群如何解冻

- 欠费冻结：用户可通过续费或充值来解冻资源，恢复集群正常使用。欠费冻结的集群允许续费、释放或删除；已经到期的包年/包月集群不能发起退订，未到期的包年/包月集群可以退订。
- 违规冻结：违规冻结的集群允许续费、释放或删除；已经到期的包年/包月集群不能发起退订，未到期的包年/包月集群可以退订。
- 公安冻结：已被公安冻结的集群允许续费，不允许释放或删除。被公安冻结的包年/包月集群不允许退订，在退订管理页面仍然显示，只是不能退订。

集群冻结、解冻对业务的影响

- 集群冻结时：资源将被限制访问和使用，会导致您的业务中断。
- 资源解冻时：资源将被解除限制。解冻后，集群将会处于“运行中”状态。例如，冻结前处于休眠状态的集群，解冻后也会处于“运行中”状态。

集群为什么被释放

客户购买产品后，如果没有及时的进行续费或充值，将进入宽限期。如宽限期满仍未续费或充值，将进入保留期。在保留期内资源将停止服务。保留期满仍未续费或充值，存储在云服务中的数据将被删除、云服务资源将被释放。请参见[资源停止服务或逾期释放说明](#)。

3 集群

3.1 集群创建

3.1.1 CCE 集群创建失败的原因与解决方法？

概述

本文主要介绍在CCE集群创建失败时，如何查找失败的原因，并解决问题。

详细信息

集群创建失败的原因包括：

1. ntpd没安装或者安装失败、k8s组件预校验不过、磁盘分区错误等，目前只能尝试重新创建，定位方法请参见[定位失败原因](#)。
2. 当前集群规模所需的底层资源不足，请选择其他规模的集群类型后重新创建集群。
3. 确认账号是否欠费：账号必须是未欠费状态才可以购买资源，包括使用代金券购买资源，详情请参见[账户总览](#)、[欠费还款](#)。

定位失败原因

您可以参考以下步骤，通过集群日志查看集群创建失败的报错信息，然后根据相应的解决方法解决问题：

- 步骤1** 登录CCE控制台，单击集群列表上方的“操作记录”查看具体的报错信息。
- 步骤2** 单击“操作记录”窗口中失败状态的报错信息。

图 3-1 查看操作详情

全部操作

全部状态

C

集群名称	操作类型	状态	操作时间
example	创建集群	● 执行中	2022/05/02 09:55:08 GMT+08:00

创建安全组

项目	开始时间	结束时间	状态
创建集群通信安全组规则	2022/05/02 09:55:08 GMT+08:00	2022/05/02 09:55:09 GMT+08:00	已完成
创建控制节点安全组规则	2022/05/02 09:55:08 GMT+08:00	2022/05/02 09:55:08 GMT+08:00	已完成
创建用户节点安全组规则	2022/05/02 09:55:08 GMT+08:00	2022/05/02 09:55:09 GMT+08:00	已完成

创建网络

创建控制节点网络	2022/05/02 09:55:08 GMT+08:00	2022/05/02 09:55:08 GMT+08:00	已完成
创建控制节点子网	2022/05/02 09:55:08 GMT+08:00	2022/05/02 09:55:12 GMT+08:00	已完成

Expected HTTP response code [200 201 202 203 204] when accessing [POST https://ecs-internal.cn-north-7.myhuaweicloud.com/v1/0524ea9c1a00d57e2fddc0190fc7dd97/cloudservers], but got 400 instead {"error":{"message":"The volume type[SSD] cannot be used with the specified flavor in the AZ [cn-north-7b].","code":"Ecs.0044"}}

创建集群

创建控制节点, 预计5分钟[0/3]	2022/05/02 09:55:12 GMT+08:00	--	失败
创建集群安全证书	--	--	未开始

步骤3 根据上一步获取的失败报错信息自行解决后，尝试重新创建集群。

----结束

3.1.2 集群的管理规模和控制节点的数量有关系吗？

在CCE集群中，集群的管理规模和控制节点数量没有明显的关联，它们是属于不同的维度的集群参数，具体说明如下：

- 集群管理规模：指集群支持管理的最大节点数。若选择50节点，表示当前集群最多可管理50个节点。
针对不同的集群规模，控制节点的规格不同，但控制节点的数量不受管理规模的影响。
- 控制节点的数量：主要影响集群的高可用性，选择多个控制节点可以提高集群的容灾能力。
例如，集群的3实例（高可用）模式将创建三个控制节点，在单个控制节点发生故障后集群可以继续使用，不影响业务功能。

3.1.3 CCE 集群创建时的根证书如何更新？

CCE集群根证书是Kubernetes认证的基础证书，华为云上的Kubernetes集群管理面托管在CCE管理平台上，证书也在CCE的管理平台上，不对用户开放，这个证书在平台上会定期维护，不会出现过期的情况。

X509证书在Kubernetes集群上也是默认开启的，更新平台会自动会维护更新。

获取集群证书

通过CCE控制台获取集群证书，使用该证书可以访问Kubernetes，详情请参见[获取集群证书](#)。

3.1.4 使用 CCE 需要关注哪些配额限制？

云容器引擎CCE配额只限制了集群个数，但是用户使用CCE时也会使用其他云服务，包括：弹性云服务器、云硬盘、虚拟私有云、弹性负载均衡、容器镜像服务等。

什么是配额？

为防止资源滥用，平台限制了各服务资源的配额，对用户的资源数量和容量做了限制。如您最多可以创建多少台弹性云服务器、多少块云硬盘。

如果当前资源配额限制无法满足使用需要，您可以申请扩大配额。

怎样查看我的配额？

1. 登录管理控制台。
2. 单击管理控制台左上角的 ，选择区域和项目。
3. 在页面右上角，选择“资源 > 我的配额”。
系统进入“服务配额”页面。

图 3-2 我的配额



4. 您可以在“服务配额”页面，查看各项资源的总配额、及使用情况。
如果当前配额不能满足业务要求，请单击“申请扩大配额”。

如何申请扩大配额？

1. 登录管理控制台。
2. 在页面右上角，选择“资源 > 我的配额”。
系统进入“服务配额”页面。

图 3-3 我的配额



3. 单击“申请扩大配额”。
4. 在“新建工单”页面提交工单，根据您的需求，填写相关参数。
其中，“问题描述”项请填写需要调整的内容和申请原因。
5. 填写完毕后，勾选协议并单击“提交”。

3.2 集群运行

3.2.1 当集群状态为“不可用”时，如何排查解决？

当集群状态显示为“不可用”时，请参照如下方式来排查解决。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- **排查项一：安全组是否被修改**
- **排查项二：集群是否过载**
- **排查项三：集群Secret落盘加密使用的KMS密钥是否有效**

如果以上排查思路仍无法解决您的问题，请[提交工单](#)寻找客服人员协助您进行定位。

排查项一：安全组是否被修改

步骤1 登录控制台，选择“服务列表 > 网络 > 虚拟私有云 VPC”，单击左侧导航栏的“访问控制 > 安全组”，找到集群控制节点的安全组。

控制节点安全组名称为：集群名称-cce-control-编号。

步骤2 单击安全组名称，进入详情页面，请确保集群控制节点的安全组规则的正确性。

安全组的详细说明请参见[集群安全组规则配置](#)。

----结束

排查项二：集群是否过载

问题现象

集群Master节点的使用率达到100%。

问题根因

在集群中同时创建大量资源，apiserver压力过大，导致Master节点过载出现OOM。

解决方案

您可以扩容集群管理规模，集群管理规模越大，控制节点规格越高、性能也更佳。操作详情请参见[变更集群规格](#)。

如果您出现以上集群过载的情况，您也可以提交工单以获取技术支持。

排查项三：集群 Secret 落盘加密使用的 KMS 密钥是否有效

问题现象

当出现集群不可用，您可以查看集群事件确认异常原因。



当集群事件中存在“KMS密钥状态异常”时，您需要确认该集群对应的使用的密钥状态是否被设置为“禁用”或“计划删除”。

事件

运维事件是指管理面事件，包含集群、节点的事件。事件保存时间为30天，30天后自动清除数据，查询时间段最多15天

导出

全部时间

选择属性筛选，或输入关键字搜索（在59条数据中进行搜索）

产生时间	事件名称	事件备注	错误信息
2024/09/12 11:02:40 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:59:31 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:56:31 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:53:34 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:50:31 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:47:29 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:44:39 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:41:23 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:38:28 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障
2024/09/12 10:35:33 GMT+0...	KMS密钥状态异常	KMS密钥状态异常	内部故障

总条数：59 +

10 < 1 2 3 4 5 6 ... >

解决方案

步骤1 登录数据加密服务 DEW控制台。

步骤2 在自定义密钥列表中，找到集群使用的KMS密钥。

- “计划删除”状态的密钥：单击操作列中的“取消删除”。如果密钥取消删除后，仍处于“禁用”状态，则还需取消禁用。
- “禁用”状态的密钥：单击操作列中的“启用”。

步骤3 密钥重新为“启用”状态后，集群将在大约在5-10分钟内自动恢复。

----结束

3.2.2 CCE 集群如何重置或重装？

CCE中的集群不能重置或重装，如确定集群无法使用，请[提交工单](#)或删除后重新购买集群。

CCE集群中的节点重置功能已上线，详情请参见[重置节点](#)。

3.2.3 如何确认已创建的集群是否为多控制节点模式？

登录CCE控制台，进入集群，在集群详情页面右侧查看控制节点数量：

- 3个节点即为多控制节点模式。
- 1个节点即为单控制节点模式。

须知

集群一旦创建，便无法更改控制节点数，需要重新创建集群才能调整。

3.2.4 是否可以直接连接 CCE 集群的控制节点？

CCE支持使用Kubectrl工具连接集群，具体请参见[通过Kubectrl连接集群](#)。

CCE不支持登录控制节点执行相关操作。

3.2.5 CCE 集群删除之后相关数据能否再次找回？

问题描述：

CCE集群删除之后相关数据能否再次找回？

问题解答：

集群删除之后，部署在集群上的工作负载也会同步删除，无法恢复，请慎重删除集群。

3.2.6 为什么 CCE 集群界面的节点磁盘监控看起来不准确？

问题描述：

CCE集群界面的某个节点磁盘监控高达80%以上，而进入云监控界面看到的磁盘使用率在40%不到。

后面在该节点上排查，发现有一个pvc磁盘使用达到了92%，将这个盘清理后，集群界面的磁盘使用率和云监控使用率一致了。

请问集群界面的节点监控是怎么样子的原理，是否只报最大磁盘使用率的数据呢？

问题解答：

CCE集群监控信息中，磁盘使用率为当前节点中使用率最高的硬盘的监控信息。

3.2.7 如何修改 CCE 集群名称？

集群创建完成后，支持修改集群名称。

步骤1 登录CCE控制台，单击集群名称进入集群“概览”页。

步骤2 在右侧，基本信息模块，单击集群名称后的 。

图 3-4 修改集群名称

基本信息

名称	 
集群 ID	 
类型	CCE Standard
集群版本	v1.30
补丁版本	v1.30.6-r10
集群状态	 运行中
集群规模	50 节点
集群时区	(UTC+08:00) Asia/Shanghai

步骤3 请在编辑名称页面输入新名称。集群名称长度范围为4-128个字符，以小写字母开头，支持小写字母、数字和中划线(-)，不能以中划线(-)结尾。

输入完成后，单击“保存”。保存后，集群名称将自动更新为修改后的名称。

📖 说明

- 集群名称不能与其他集群的名称或原名相同
- 修改名称后，CTS/RMS/EPS服务的记录会按照修改后的名称命名

----结束

3.2.8 控制台访问异常问题排查

访问容器组日志时报错

若您在查看容器组日志时遇到该问题，而集群其他资源均可正常访问，您可以按以下步骤排查。

1. 登录CCE控制台，单击集群名称进入集群。
2. 选择左侧导航栏的“工作负载”，查看工作负载容器组运行状态是否为“运行中”，若不是，请根据[工作负载状态异常定位方法](#)进行排查。
3. 在集群“概览”页面，在“网络信息”选择“节点默认安全组”，单击跳转到安全组页面，查看安全组入方向规则。
4. 检查所有安全组规则，确认是否开启了来自VPC私网、目标端口为TCP/10250的入方向访问，若未开放，请添加上述安全组规则。
5. 若以上均不符合，请提交工单咨询。

当前账号未被授予该操作所需的集群 RBAC 权限

问题现象

当您访问控制台时，出现报错“无任何命名空间的资源权限，请联系管理员添加权限”。



问题原因

您使用的账号未被授予当前操作所需的集群RBAC权限。

解决方案

1. 使用华为云账号或者具有管理员权限的账号登录IAM管理控制台，在左侧导航栏中选择“用户”。
2. 在IAM用户页签查找出现报错的用户名，单击用户名右侧的“授权”。
3. 选择相应的权限后，单击“确定”，提交授权。

当前账号未被授予该操作所需的 IAM 权限

问题现象

当您访问控制台时，出现报错“权限不足”，错误码：CCE.01403001。

问题原因

您使用的账号未被授予当前操作所需的IAM权限。

解决方案

1. 使用华为云账号或者具有IAM权限的账号登录IAM管理控制台。
2. 根据错误提示页面的信息为您的账号添加[CCE控制台依赖的权限](#)。详细授权操作，请参见[集群权限（IAM授权）](#)。

3.3 集群删除

3.3.1 集群删除失败：安全组中存在残留资源

CCE在删除集群时，会连接集群的kube-apiserver查询集群对接的周边资源信息，如Turbo集群对接的弹性网卡/弹性辅助网卡等，当CCE集群的状态为不可用，冻结，休眠等状态时，删除集群有可能会出现查询资源失败而导致集群删除失败的情况。

故障现象

删除集群失败，报错信息如下：

```
Expected HTTP response code [200 202 204 404] when accessing [DELETE https://vpc.***.com/v2.0/security-groups/46311976-7743-4c7c-8249-ccd293bcae91], but got 409 instead  
{  
  "code": "VPC.0602",  
  "message": "{  
    \"NeutronError\": {  
      \"message\": \"Security Group 46311976-7743-4c7c-8249-ccd293bcae91 in use.\",  
      \"type\": \"SecurityGroupInUse\",  
      \"detail\": \"\"  
    }  
  }  
}
```

问题根因

该场景引起的原因是集群关联的安全组中存在未删除的资源，该安全组无法删除，最终导致了集群删除失败。

操作步骤

步骤1 复制报错信息中的资源ID，进入到VPC服务的安全组界面，根据ID过滤安全组。



步骤2 单击进入安全组详情界面，选择关联实例页签。



查询该安全组关联的其他资源，例如服务器、弹性网卡实例、辅助弹性网卡实例等。您可以将残留的资源（辅助弹性网卡会自动删除）删除。



步骤3 以删除残留的弹性网卡为例，您需要前往弹性网卡界面将上一步查询到的网卡删除。

可以用ID过滤需要删除的弹性网卡，也可以通过集群ID的名称过滤需要删除的弹性网卡。



步骤4 清理完成后，前往安全组页面确认该安全组已经没有关联的实例，然后前往CCE控制台即可正常删除集群。

----结束

3.3.2 冻结或不可用的集群删除后如何清除残留资源

处于非运行状态（例如冻结、不可用状态）中的集群，由于无法获取集群中的PVC、Service、Ingress等资源，因此删除集群之后可能会残留网络及存储等资源，您需要前往资源所属服务手动删除。

弹性负载均衡资源

步骤1 前往弹性负载均衡控制台。

步骤2 通过集群使用的VPC ID进行过滤，得到该虚拟私有云下所有的弹性负载均衡实例。



步骤3 查看负载均衡实例下的监听器详情，描述中包含集群ID、Service ID等信息，说明该监听器由此集群创建。



步骤4 您可以根据上述信息将集群下残留的弹性负载均衡相关资源删除。

----结束

云硬盘资源

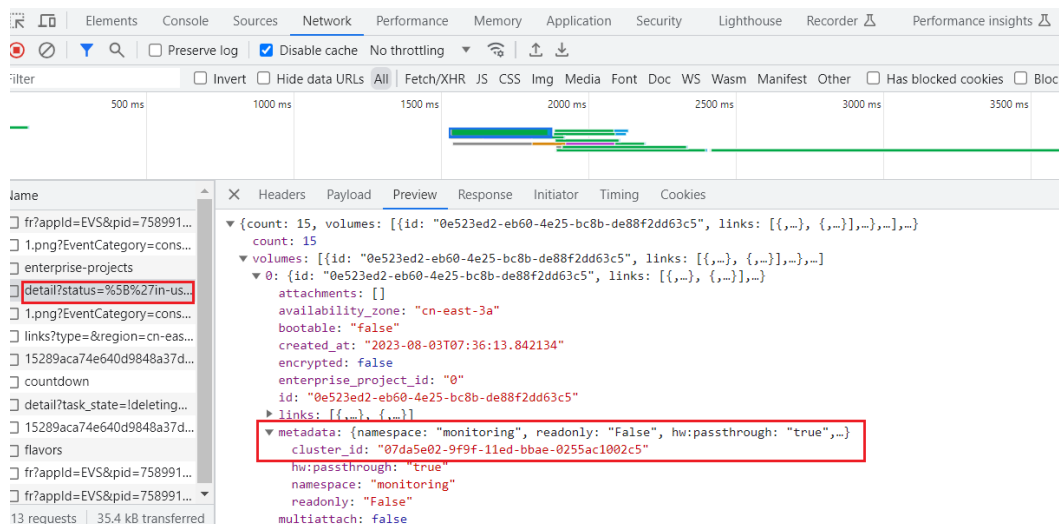
通过PVC动态创建方式创建的云硬盘名称格式为“pvc-{uid}”，且接口中的MetaData字段包含集群ID信息，您可以通过集群ID筛选出该集群中自动创建的云硬盘，根据需要进行删除。

步骤1 前往云硬盘控制台。

步骤2 通过名称“pvc-{uid}”进行过滤，得到所有由CCE自动创建的云硬盘实例。



步骤3 通过F12进入浏览器开发人员工具，查看detail接口中的MetaData字段包含集群ID信息，说明该云硬盘由此集群创建。



步骤4 您可以根据上述信息将集群下残留的云硬盘资源删除。

说明

删除后将无法恢复数据，请谨慎操作。

----结束

弹性文件服务资源

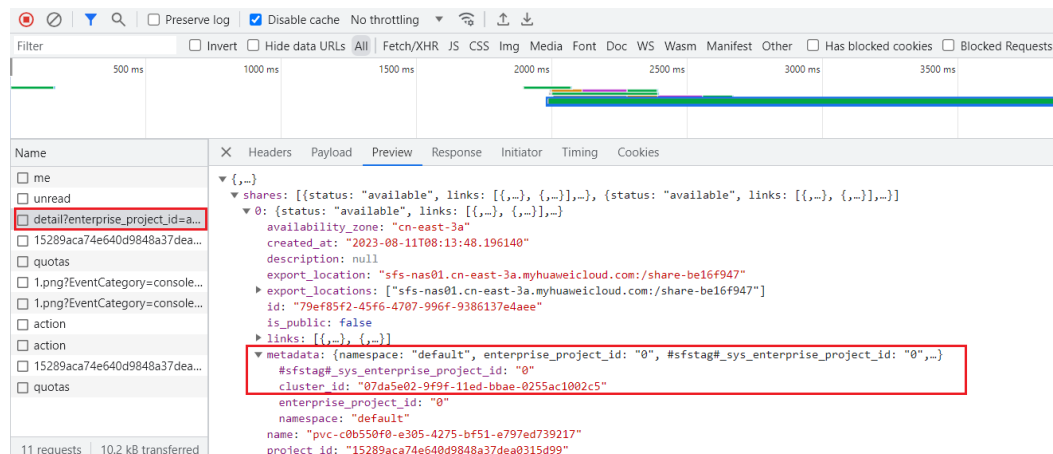
通过PVC动态创建方式创建的弹性文件服务容量型实例名称格式为“pvc-{uid}”，且接口中的MetaData字段包含集群ID信息，您可以通过集群ID筛选出该集群中自动创建的弹性文件服务容量型实例，根据需要进行删除。

步骤1 前往弹性文件服务控制台。

步骤2 通过名称“pvc-{uid}”进行过滤，得到所有由CCE自动创建的弹性文件实例。

名称	可用区	状态	协议类型	已用容量 (GiB)	最大容量 (GiB)	加密	挂载地址	操作
☐ pvc-c0b550f0-e305-4275-bf51-e797ed739217	可用区1	可用	NFS	0.00	1.00	否	15289aca74e640d9848a37dea0315d99	查看详情 更多
☐ pvc-c29056b4-...	可用区1	可用	NFS	0.00	自动扩容	否	15289aca74e640d9848a37dea0315d99	查看详情 更多

步骤3 通过F12进入浏览器开发人员工具，查看detail接口中的MetaData字段包含集群ID信息，说明该弹性文件实例由此集群创建。



步骤4 您可以根据上述信息将集群下残留的弹性文件资源删除。

说明

删除后将无法恢复数据，请谨慎操作。

----结束

3.4 集群升级

3.4.1 CCE 集群升级时，升级集群插件失败如何排查解决？

概述

本文主要介绍在CCE在升级集群时，如何查找插件升级失败的原因，并解决问题。

！

集群升级提醒，请及时处理故障

v1.19.16.05 — v1.21.0.0

38.30%

测试

① 升级前检查

② 备份

③ 配置升级包

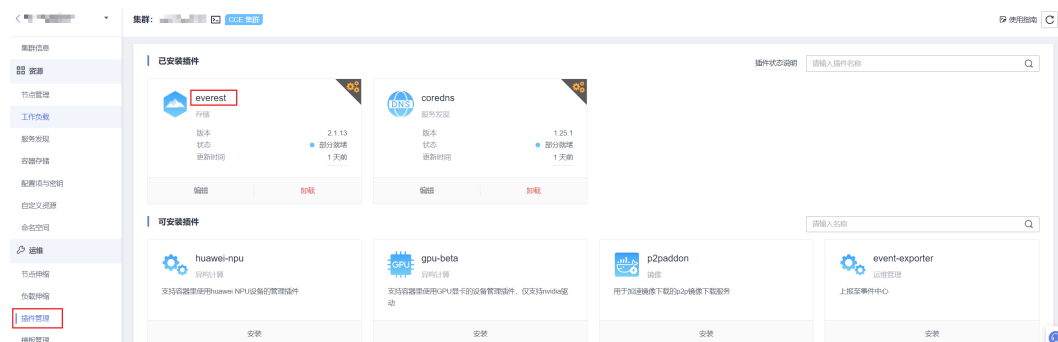
④ 升级验证

	详情	开始时间	结束时间	状态
升级前检查	升级前检查	2023/1/19 16:27:59 GMT+08:00 2023/1/19 16:28:21 GMT+08:00	2023/1/19 16:28:00 GMT+08:00 2023/1/19 16:28:21 GMT+08:00	成功
组件升级策略	📄 策略名称为策略ID 升级策略kubernetes 升级策略kubernetes	2023/1/19 16:28:22 GMT+08:00 2023/1/19 16:28:22 GMT+08:00	2023/1/19 16:28:20 GMT+08:00 2023/1/19 16:28:21 GMT+08:00	成功
控制节点升级	数据备份 升级前处理 升级策略执行点	— — —	— — —	等待中
用户节点升级	升级用户节点	—	—	等待中
升级后处理	升级后处理	—	—	等待中

操作步骤

步骤1 插件升级失败后，请优先进行重试。若重试不成功，则根据后续步骤排查问题。

步骤2 在升级界面显示失败后，请退出集群升级页面，前往“插件中心”界面查看插件的详细信息。针对异常的插件，单击插件名称查看详情。



步骤3 在插件运行实例的详情界面，单击“事件”查看异常实例的信息。

实例列表

批量删除

实例名称 请输入关键词

☐	实例名称	状态	命名空间	实例IP	所在节点	重启...	CPU 申请值/限制值/使用	内存 申请值/限制值/使用	创建时间	操作
☐	everest-csi-control	处理中	kube-system	--	10.18.26.90	0	0.25 Cores 0.25 Cores 0.00%	0.59 GiB 1.46 GiB 0.00%	17 分钟前	监控 事件 更多
☐	everest-csi-driver- 主机网络	运行中	kube-system	10.18.26.90	10.18.26.90	0	0.1 Cores 0.5 Cores 0.80%	300 MiB 300 MiB 25.10%	33 分钟前	监控 事件 更多
☐	everest-csi-driver- 主机网络	运行中	kube-system	10.18.174.197	10.18.174.197	0	0.1 Cores 0.5 Cores 0.80%	300 MiB 300 MiB 22.75%	33 分钟前	监控 事件 更多
☐	everest-csi-control	运行中	kube-system	172.16.0.3	10.18.174.197	0	0.25 Cores 0.25 Cores 0.80%	0.59 GiB 1.46 GiB 4.14%	50 分钟前	监控 事件 更多

步骤4 根据具体的异常信息进行相应处理，比如尝试删除未启动的实例让其重启等。

事件

注：事件保存时间为1小时，1小时后自动清除数据

开始日期 — 结束日期

请输入 K8s 事件搜索

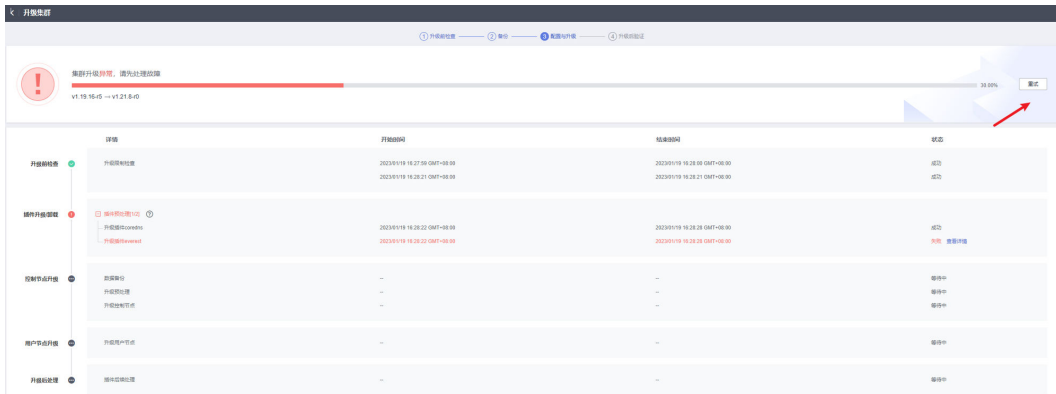
K8s 组件名	事件类型	发生次数	事件名称	K8s 事件	首次发生时间	最近发生时间
kubelet	警告	2	实例挂卷失败	Unable to attach or mount volumes: un...	2023/01/19 16:33:01 ...	2023/01/19 16:46:37 ...
kubelet	警告	16	实例挂卷失败	MountVolume.SetUp failed for volume "...	2023/01/19 16:28:44 ...	2023/01/19 16:45:06 ...
kubelet	警告	2	实例挂卷失败	Unable to attach or mount volumes: un...	2023/01/19 16:35:16 ...	2023/01/19 16:44:20 ...
kubelet	警告	3	实例挂卷失败	Unable to attach or mount volumes: un...	2023/01/19 16:37:30 ...	2023/01/19 16:42:04 ...
kubelet	警告	1	实例挂卷失败	Unable to attach or mount volumes: un...	2023/01/19 16:30:46 ...	2023/01/19 16:30:46 ...
--	正常	1	实例调度成功	Successfully assigned kube-system/ev...	2023/01/19 16:28:43 ...	2023/01/19 16:28:43 ...
--	警告	1	实例调度失败	0/2 nodes are available: 2 node(s) didn't...	2023/01/19 16:28:25 ...	2023/01/19 16:28:25 ...
--	警告	1	实例调度失败	0/2 nodes are available: 2 node(s) didn't...	2023/01/19 16:28:25 ...	2023/01/19 16:28:25 ...

步骤5 处理成功后，插件状态会变为运行中，需要保证所有插件状态都处于运行中。

已安装插件

everest 存储 版本 2.1.13 状态 运行中 更新时间 21 分钟前 编辑 卸载	coredns 服务发现 版本 1.25.1 状态 运行中 更新时间 21 分钟前 编辑 卸载

步骤6 此时进入集群升级界面，再次单击“重试”按钮即可。



----结束

3.4.2 CCE 集群升级时，ELB Ingress 与 ELB 配置不一致如何处理？

ELB 资源简介

在CCE集群中，使用ELB Ingress可以将外部流量路由到集群内的服务。Ingress中定义的一系列参数实际上是作用于弹性负载均衡器（ELB）的配置，以便更好地控制流量管理和分发。[表3-1](#)将详细介绍这些ELB配置与Ingress参数的对应关系。

表 3-1 ELB 配置与 ELB Ingress 中的对应关系

ELB配置		说明	ELB Ingress中的对应关系
负载均衡器（elb）	-	接受来自客户端的传入流量并将请求转发到一个或多个可用区中的后端服务器。	对应Ingress注解中的kubernetes.io/elb.id
监听器（listener）	-	监听器使用配置的协议和端口检查来自客户端的连接请求，并根据定义的分配策略和转发策略将请求转发到一个后端服务器组里的后端服务器。	对应Ingress注解中的kubernetes.io/elb.port，如果不填默认是80/443
转发策略（l7policy）	转发策略	ELB转发规则： 通过域名、路径进行转发。	转发规则中的域名和路径分别对应Ingress参数中的spec.rules[].host和spec.rules[].http.paths[].path配置
		ELB转发动作： 转发至后端服务器组。	后端服务器组对应Ingress参数中spec.rules[].http.paths[].backend.service填写的Service后端服务

ELB配置		说明	ELB Ingress中的对应关系
	高级转发策略	ELB转发规则：域名、路径、HTTP请求方法、HTTP请求头、查询字符串、网段。 ELB转发动作：转发至后端服务器组、重定向至监听器、重定向至URL、返回固定响应、重写、写入Header、删除Header、限速。	对应Ingress注解中kubernetes.io/elb.actions、kubernetes.io/elb.conditions等高级转发策略相关的注解。 详情请参见： 为ELB Ingress配置高级转发策略
后端服务器组（pool）	-	后端服务器组是一个或多个后端服务器的逻辑集合，用于将客户端的流量转发到一个或多个后端服务器，满足用户同时处理海量并发业务的需求。后端服务器可以是云服务器实例、辅助弹性网卡或IP地址。	对应Ingress参数中的spec.rules[].http.paths[].backend
后端服务器（member）	-	负载均衡器会将客户端的请求转发给后端服务器处理。例如，您可以添加ECS实例作为负载均衡器的后端服务器，监听器使用您配置的协议和端口检查来自客户端的连接请求，并根据您定义的分配策略将请求转发到后端服务器组里的后端服务器。	Ingress对应Service的endpoint

Ingress 与 ELB 配置不一致处理思路

集群升级前检查中，可能扫描出多个Ingress与ELB配置不一致项，并显示每个不一致项期望的配置。示例如下：

```
inconsistent: ingress(default/nginx) need create listener(HTTP/8000) of elb(4a090bf3-9d11-45dc-8d04-f9cf010ba2ec), the wanted configuration is
{"name":"k8s_HTTP_8000","protocol_port":8000,"protocol":"HTTP","loadbalancer_id":"4a090bf3-9d11-45dc-8d04-f9cf010ba2ec","sni_container_refs":null,"http2_enable":false}

inconsistent: ingress(default/nginx) need create pool(k8s_defaultnginx-80_HTTP-8000), the wanted configuration is
{"name":"k8s_defaultnginx-80_HTTP-8000","protocol":"HTTP","loadbalancer_id":"4a090bf3-9d11-45dc-8d04-f9cf010ba2ec","lb_algorithm":"ROUND_ROBIN","slow_start":{"enable":false,"duration":30}}

inconsistent: need create member(address: 172.16.0.54) of pool(k8s_defaultnginx-80_HTTP-8000), the wanted configuration is {"name":"2c2405d1abc17da48c70e9edc9a340fc","subnet_cidr_id":"77b9ad29-e30f-451b-92e7-949b83220b0f","address":"172.16.0.54","protocol_port":30838,"weight":6}

inconsistent: ingress(default/nginx) need create l7policy(k8s_defaultnginx_6666cd76) of listener(k8s_HTTP_8000/HTTP/8000), the wanted configuration is
{"name":"k8s_defaultnginx_6666cd76","listener_id":"k8s_HTTP_8000/HTTP/8000","action":"REDIRECT_TO_POOL","redirect_pool_id":"k8s_defaultnginx-80_HTTP-8000"}

inconsistent: ingress(default/nginx) need create rule of l7policy(k8s_defaultnginx_6666cd76), the wanted configuration is {"type":"PATH","compare_type":"STARTS_WITH","value":"/"}
```

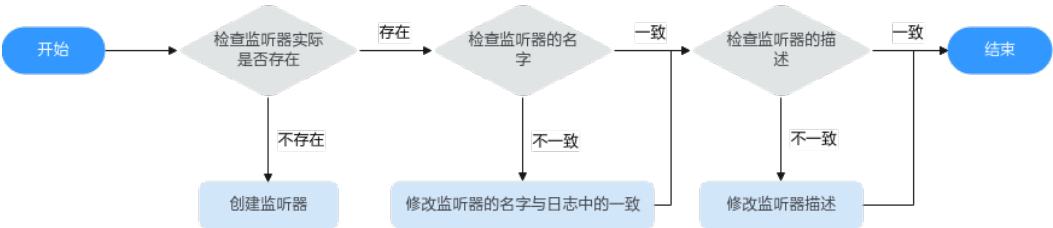
您可按照以下顺序依次处理：

序号	异常项	异常原因
1	监听器（listener）	• 监听器存在不一致-需要创建监听器 • 监听器存在不一致-需要更新监听器
2	转发策略（l7Policy）	• 转发策略存在不一致-需要创建转发策略 • 转发策略存在不一致-需要更新转发策略 • 转发策略存在不一致-需要删除转发策略
3	转发规则（rule）	转发规则存在不一致-需要创建/删除转发规则
4	后端服务器组（pool）	后端服务器组存在不一致-需要创建后端服务器组
5	后端服务器（member）	• 后端服务器存在不一致-需要创建后端服务器 • 后端服务器存在不一致-需要更新后端服务器 • 后端服务器存在不一致-需要删除后端服务器
6	健康检查（healthMonitor）	• 健康检查配置存在不一致-需要删除健康检查 • 健康检查配置存在不一致-需要更新健康检查

监听器存在不一致-需要创建监听器

inconsistent: ingress(xxx) need create listener ...

此问题出现在用户通过ELB控制台修改了监听器的名字、描述，或者删除了监听器。可在ELB控制台根据端口查看监听器是否存在，并进一步检查监听器名字与描述是否匹配。可按照以下步骤处理：



监听器的描述信息应为如下所示，其中cluster_id的值为本集群ID。

```
{"attention":"Auto-generated by CCE service, do not modify!","cluster_id":"5d4d44bd-0891-11f0-84d3-0255ac10003e"}
```

例如：

```
inconsistent: ingress(default/nginx) need create listener(HTTP/8000) of elb(4a090bf3-9d11-45dc-8d04-f9cf010ba2ec), the wanted configuration is
{"name":"k8s_HTTP_8000","protocol_port":8000,"protocol":"HTTP","loadbalancer_id":"4a090bf3-9d11-45dc-8d04-f9cf010ba2ec","sni_container_refs":null,"http2_enable":false}
```

表示default名称空间下的nginx这个ingress，需要在ELB（id：4a090bf3-9d11-45dc-8d04-f9cf010ba2ec）下创建端口为8000，协议为HTTP的监听器，监听器的名字为k8s_HTTP_8000（sni_container_refs为null，http2_enable为false表示无SNI与HTTP2配置），描述信息也需同时设置。



监听器存在不一致-需要更新监听器

inconsistent: ingress(yyy) need update listener ...

此问题出现在用户通过ELB控制台修改了监听器的配置，并未在CCE侧同步，可在ELB控制台查看该监听器的实际配置，并与期望配置项逐一对比，若存在不一致，则需将监听器的配置手动修改为CCE侧的期望配置，或者修改Ingress。

例如：

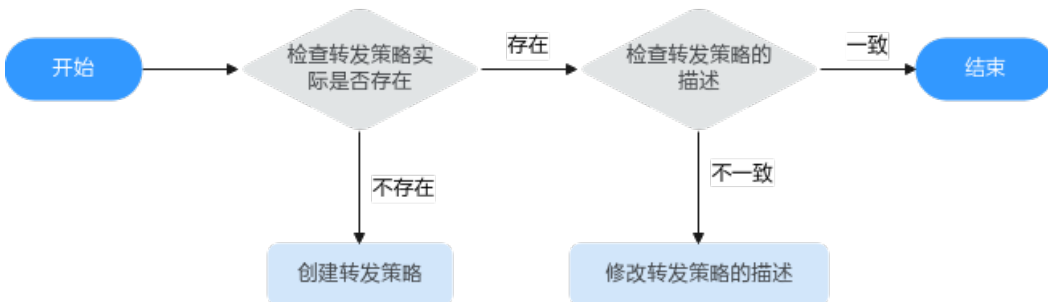
inconsistent: ingress(default/cce-elb-ingress) need update listener(5ca8a931-9d7a-465e-b503-08f88d528cd8), the wanted configuration is {"id":"5ca8a931-9d7a-465e-b503-08f88d528cd8","sni_container_refs":null}

the wanted configuration is后打印了监听器的期望配置，此处为sni_container_refs: null，可排查ELB侧监听器的SNI功能是否开启，而Ingress里没有配置。若Ingress里没有配置，则需在Ingress里加上SNI配置。操作步骤详情请参见[为ELB Ingress配置服务名称指示（SNI）](#)。

转发策略存在不一致-需要创建转发策略

inconsistent: ingress(yyy) need create l7policy(yyy) ...

此问题出现在用户通过ELB控制台修改了CCE创建的转发策略的描述，或者删除了转发策略。可在ELB控制台对应监听器下根据转发策略名字搜索其是否存在，并进一步检查转发策略的描述是否匹配。可按照以下步骤处理：



转发策略的描述信息如下所示，其中cluster_id的值为本集群ID，ingress_id为当前Ingress的UID。

```
{"attention":"Auto-generated by CCE service, do not modify!","cluster_id":"5d4d44bd-0891-11f0-84d3-0255ac10003e","ingress_id":"8a79a17b-f9ab-4431-95f6-0e1125093aac"}
```

需要注意的是，ELB控制台不会显示转发策略的描述信息，需从浏览器接口返回信息中查看。

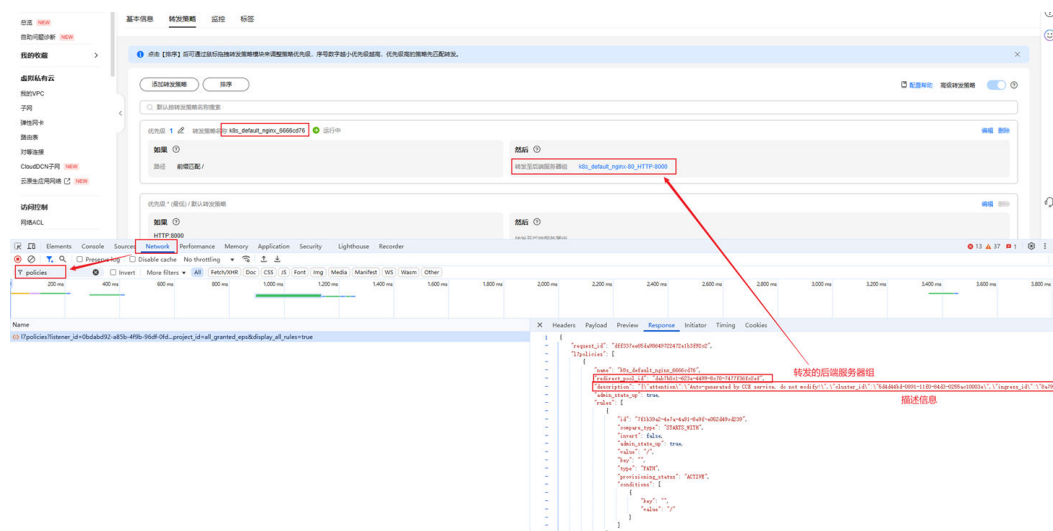
例如：

inconsistent: ingress(default/nginx) need create l7policy(k8s_defaultnginx_6666cd76) of listener(0bdabd92-a85b-4f9b-96df-0fde706ea028), the wanted configuration is


```
{"name":"k8s_default_nginx_6666cd76","listener_id":"0bdabd92-a85b-4f9b-96df-0fde706ea028","action":"REDIRECT_TO_POOL","redirect_pool_id":"k8s_default_nginx-80_HTTP-8000"}
```

表示default名称空间下nginx这个ingress，需要在监听器（id：0bdabd92-a85b-4f9b-96df-0fde706ea028）下创建名为k8s_default_nginx_6666cd76的转发策略，并转发到名为k8s_default_nginx-80_HTTP-8000的后端服务器组，描述信息也需同时设置。您可通过[创建转发策略](#)接口进行创建。

在浏览器接口返回信息中查看创建的转发策略。



转发策略存在不一致-需要更新转发策略

inconsistent: ingress(xxx) need update l7policy(xxx) of listener(xxx) ...

此问题出现在用户通过ELB控制台修改了CCE创建的转发策略的后端服务器组（redirect_pool_id）。例如：

inconsistent: ingress(default/nginx) need update l7policy(c4beb313-0a32-4add-9d0e-68b8f0f90e01) of listener(0bdabd92-a85b-4f9b-96df-0fde706ea028), the wanted configuration is {"id":"c4beb313-0a32-4add-9d0e-68b8f0f90e01","redirect_pool_id":"k8s_default_nginx-80_HTTP-8000","redirect_pools_config":[]}

表示default命名空间下nginx这个Ingress，需要在监听器（id：0bdabd92-a85b-4f9b-96df-0fde706ea028）下把转发策略（id：c4beb313-0a32-4add-9d0e-68b8f0f90e01）的重定向后端服务器组id（redirect_pool_id）修改为k8s_default_nginx-80_HTTP-8000这个后端服务器组的id，并删除重定向后端服务器组（redirect_pools_config）的配置。

转发策略存在不一致-需要删除转发策略

inconsistent: ingress(xxx) need delete l7policy(id: xxx) ...

此问题出现在用户通过ELB控制台在CCE创建的监听器下新建了转发策略。需根据转发策略id找到对应的转发策略，并将其删除。

例如：

inconsistent: need delete l7policy(id: f4eda0af-869a-47fb-a699-2f27f286b641)

可通过[查询转发策略详情](#)接口查询该转发策略位于哪个ELB监听器下，并将其删除。若您想保留该转发策略，可在Ingress里加上配置。

转发规则存在不一致-需要创建/删除转发规则

inconsistent: ingress(xxx) need create rule of l7policy(xxx) ...

此问题出现在用户通过ELB控制台修改了CCE创建的转发策略下的转发规则，通常是修改了类型、匹配方式、路径。由于CCE对于转发规则的更新方式为删除后重创，因此需要创建/删除转发规则的不一致问题通常会同时出现。

例如：

inconsistent: ingress(default/nginx) need create rule of l7policy(k8s_defaultnginx_6666cd76), the wanted configuration is {"type":"PATH","compare_type":"STARTS_WITH","value":"/"}

inconsistent: need delete rule(6cf87a3e-373e-4378-bf7a-16a709feee63) of l7policy(c4beb313-0a32-4add-9d0e-68b8f0f90e01)

表示default名称空间nginx这个ingress，需要在转发策略k8s_defaultnginx_6666cd76下创建类型为PATH，匹配方式为STARTS_WITH（前缀匹配），值为/的转发规则。同时需要在转发策略（id：c4beb313-0a32-4add-9d0e-68b8f0f90e01）下将转发规则（id：6cf87a3e-373e-4378-bf7a-16a709feee63）删除。



后端服务器组存在不一致-需要创建后端服务器组

inconsistent: need create pool(xxx)...

此问题出现在用户通过ELB控制台修改了CCE创建的转发策略的后端服务器组（redirect_pool_id），通常伴随着“need update l7policy ...”一起出现。可根据后端服务器组的名称、监听器查找其是否存在。若存在，则需修改转发策略的后端服务器组。

例如：

inconsistent: ingress(default/nginx) need create pool(k8s_defaultnginx-80_HTTP-8000), the wanted configuration is {"name":"k8s_defaultnginx-80_HTTP-8000","protocol":"HTTP","loadbalancer_id":"4a090bf3-9d11-45dc-8d04-f9cf010ba2ec","lb_algorithm":"ROUND_ROBIN","slow_start":{"enable":false,"duration":30}}

若不存在，则需按照打印出的配置项进行创建。

后端服务器存在不一致-需要创建后端服务器

inconsistent: need create member(address: xxx) of pool(xxx) ...

此问题出现在用户通过ELB控制台修改了CCE创建的后端服务器组中的后端服务器。

例如：

inconsistent: need create member(address: 172.16.0.54) of pool(dab7b0c1-623e-4499-8c70-7477f36fc2ef), the wanted configuration is {"name":"2c2405d1abc17da48c70e9edc9a340fc","subnet_cidr_id":"77b9ad29-e30f-451b-92e7-949b83220b0f","address":"172.16.0.54","protocol_port":30838,"weight":6}

需要在后端服务器（id：dab7b0c1-623e-4499-8c70-7477f36fc2ef）下添加IP地址为172.16.0.54，端口30838，权重为6，子网为77b9ad29-e30f-451b-92e7-949b83220b0f，名称为2c2405d1abc17da48c70e9edc9a340fc的后端服务器。ELB控制台无法填写名称，您可通过[创建后端服务器](#)接口进行创建。

后端服务器存在不一致-需要更新后端服务器

inconsistent: need update member(address: xxx) of pool(yyy) ...

此问题出现在用户通过ELB控制台修改了CCE创建的后端服务器组中的后端服务器的权重。

例如：

inconsistent: need update member(9a65b96c-891a-439e-a692-508c3e095095) of pool(dab7b0c1-623e-4499-8c70-7477f36fc2ef), the wanted configuration is {"id":"9a65b96c-891a-439e-a692-508c3e095095","name":"2c2405d1abc17da48c70e9edc9a340fc","weight":6}

需要在后端服务器组（id：dab7b0c1-623e-4499-8c70-7477f36fc2ef）中将后端服务器（id：9a65b96c-891a-439e-a692-508c3e095095）的权重修改为6。



后端服务器存在不一致-需要删除后端服务器

inconsistent: need delete member(address: xxx) of pool(yyy) ...

此问题出现在用户通过ELB控制台修改了CCE创建的后端服务器组中的后端服务器。

例如：

inconsistent: need delete members([d0891296-daf9-496b-9cac-ad1a1101b303]) of pool(dab7b0c1-623e-4499-8c70-7477f36fc2ef)

需要在后端服务器组（id：dab7b0c1-623e-4499-8c70-7477f36fc2ef）中将后端服务器（id：d0891296-daf9-496b-9cac-ad1a1101b303）删除。

健康检查配置存在不一致-需要删除健康检查

inconsistent: need delete healthMonitor(yyy) of pool(yyy) ...

此问题出现在用户通过ELB控制台开启了后端服务器组的健康检查，而未在Ingress里配置。

例如：

inconsistent: need delete healthMonitor(ef7ad2c3-bdda-4727-8a1c-de37eeb48b55) of pool(402dbff8-58b8-4a49-bb99-0dc5cf87c35b)

需要将后端服务器组（id：402dbff8-58b8-4a49-bb99-0dc5cf87c35b）上配置的健康检查删除（可通过[删除健康检查](#)接口进行删除），或者在对应的Ingress中开启健康检查。

健康检查配置存在不一致-需要更新健康检查

inconsistent: need update healthMonitor(yyy) of pool(yyy) ...

此问题出现在用户通过ELB控制台修改了后端服务器组的健康检查配置。

例如：

inconsistent: need update healthMonitor(9d438b19-4342-42af-a876-77d49bbc9447) of pool(402dbff8-58b8-4a49-bb99-0dc5cf87c35b), the wanted configuration is {"delay":5,"max_retries":3,"timeout":10,"type":"HTTP","url_path":"/","monitor_port":null,"admin_state_up":true}

需要将健康检查（ id: 9d438b19-4342-42af-a876-77d49bbc9447 ）的配置修改正确。

健康检查		配置健康检查	
健康检查	已开启	健康检查协议	HTTP
健康检查域名	使用或创建负载均衡的内网IP为域名	健康检查端口	使用或创建负载均衡监听端口
健康检查路径	/	检查间隔秒数	7
超时时间(秒)	10	健康检查正常阈值	3
健康检查异常阈值	3		

常见问题

- listener/l7policy/pools存在，为什么还会显示need create?
解决方案：
CCE管理的资源必须有description，您需要检查description是否存在。

4 节点

4.1 节点异常问题排查

排查思路

CCE支持[通过NPD插件排查](#)节点异常问题，您也可以参考[自主排查流程](#)进行排查。
若[通过NPD插件排查](#)和[自主排查流程](#)都无法解决问题，请提交工单排查。

通过 NPD 插件排查

CCE提供节点故障检测[NPD插件](#)，NPD插件从1.16.0版本开始增加了大量检查项，能对节点上各种资源和组件的状态检测，帮助发现节点故障。
强烈建议您安装该插件，如已安装请查看插件版本并升级到**1.16.0及以上**版本。
安装NPD插件后，当节点出现异常时，控制台上可以查看到指标异常。



您还可以在节点事件中查看到NPD上报的事件，根据事件信息可以定位故障。

事件

注：事件保存时间为1小时，1小时后自动清除数据

开始日期 - 结束日期

请输入 K8s 事件搜索

K8s 组件名	事件类型	发生次数	事件名称	K8s 事件	首次发生时间	最近发生时间
node-problem-detector	告警	1	异常事件	Node condition DiskReadonly is now: T...	2022/08/05 10:39:35 ...	2022/08/05 10:39:35 ...
node-problem-detector	告警	1	异常事件	Node condition CNIPProblem is now: Tru...	2022/08/05 10:39:30 ...	2022/08/05 10:39:30 ...
node-problem-detector	告警	1	异常事件	Node condition CRIPProblem is now: Tru...	2022/08/05 10:39:29 ...	2022/08/05 10:39:29 ...
node-problem-detector	告警	1	异常事件	Node condition ReadonlyFilesystem is ...	2022/08/05 10:39:13 ...	2022/08/05 10:39:13 ...
node-problem-detector	正常	1	正常事件	Started Cloud Container Engine Kubele...	2022/08/05 10:21:02 ...	2022/08/05 10:21:02 ...
node-problem-detector	正常	2	正常事件	Starting Docker Application Container E...	2022/08/05 10:20:58 ...	2022/08/05 10:21:02 ...

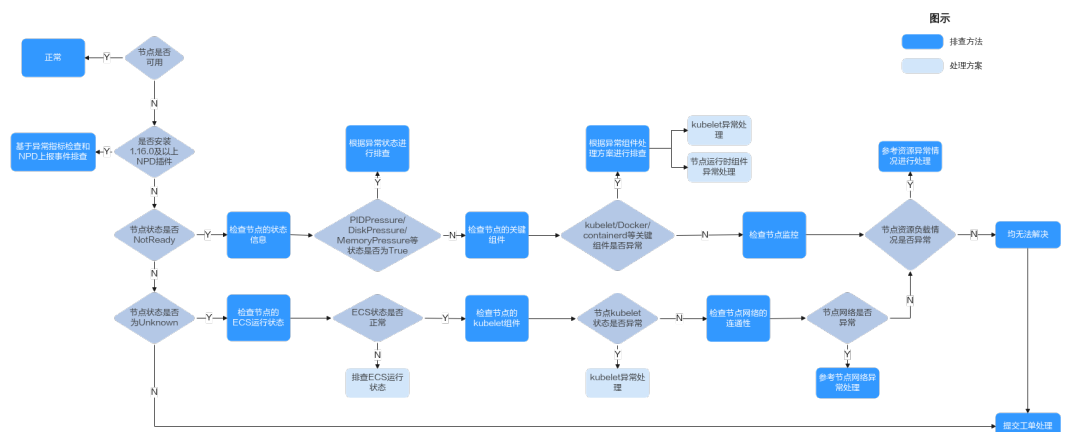
表 4-1 故障事件说明

故障事件	说明
OOMKilling	检查oom事件发生并上报。 处理建议： 排查项一：节点负载过高。
TaskHung	检查taskHung事件发生并上报
KernelOops	检查内核0指针panic错误
ConntrackFull	检查连接跟踪表是否满
FrequentKubeletRestart	检测kubelet频繁重启
FrequentDockerRestart	检测docker频繁重启
FrequentContainerdRestart	检测containerd频繁重启
CRIPProblem	检查容器CRI组件状态
KUBELETProblem	检查Kubelet状态
NTPProblem	检查ntp服务状态
PIDProblem	检查PID是否充足
FDProblem	检查文件句柄数是否充足
MemoryProblem	检查节点整体内存是否充足
CNIPProblem	检查容器CNI组件状态
KUBEPROXYProblem	检查Kube-proxy状态

故障事件	说明
ReadOnlyFilesystem	检查系统内核是否有Remount root filesystem read-only错误。 可能原因：用户在ECS侧误操作卸载数据盘、节点vdb盘被删除。 处理建议： ● 排查项六：检查磁盘是否异常 ● 排查项九：检查节点中的vdb盘是否被删除
DiskReadOnly	检查系统盘、docker盘、kubelet盘是否只读 可能原因：用户在ECS侧误操作卸载数据盘、节点vdb盘被删除。 处理建议： ● 排查项六：检查磁盘是否异常 ● 排查项九：检查节点中的vdb盘是否被删除
DiskProblem	检查磁盘使用量与关键逻辑磁盘挂载 检查系统盘、docker盘、kubelet盘磁盘使用率，检查docker盘、kubelet盘是否正常挂载在虚拟机上。
PIDPressure	检查PID是否充足。 处理建议：PID不足时可调整PID上限，请参见修改节点进程 ID数量上限kernel.pid_max。
MemoryPressure	检查容器可分配空间（allocable）内存是否充足
DiskPressure	检查kubelet盘和docker盘的磁盘使用量及inodes使用量。 处理建议：扩容数据盘，请参见节点磁盘扩容。

自主排查流程

图 4-1 诊断流程图



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”，切换至“节点”页签。
3. 单击节点操作栏中的“更多 > 查看YAML”，查看节点的Status字段。

节点状态为NotReady状态

- 查看节点的状态信息，是否存在PIDPressure、DiskPressure、MemoryPressure等节点状态是否为True。如果节点存在任一状态为True，则基于异常的关键词，查找相应的解决方案。
- 检查节点上的关键组件，及关键组件上的日志。节点上的关键组件为Kubelet及其运行时组件（ Docker/Containerd ），详细操作请参见[检查节点的关键组件](#)。
 - Kubelet组件
检查Kubelet的状态、日志等是否存在异常。若存在异常则参考[Kubelet异常处理](#)。
 - 运行时组件（ Docker/Containerd ）
 - 检查节点的运行时组件，若不确定节点的运行时组件具体是Docker还是Containerd，请登录CCE控制台，查看节点的运行时信息。
 - 如果存在异常，则参考[节点运行时异常处理-RuntimeOffline](#)。
 - NTP组件
 - 检查NTP服务的状态、日志、配置等是否存在异常。
 - 若NTP服务存在异常，则参考[NTP异常处理-NTPProblem](#)。
- 检查节点的监控，查看节点的CPU、内存、网络等资源负载情况是否存在异常。如果节点负载情况存在异常，请参考[节点MemoryPressure](#)等进行解决。

节点状态为Unknown状态

- 登录ECS界面，查看节点是否存在。
- 需确认节点是否处于运行中状态。
- 检查节点上的关键组件，及关键组件上的日志。节点上的关键组件为Kubelet及其运行时组件（ Docker/Containerd ），详细操作请参见[检查节点的关键组件](#)。
 - Kubelet组件
 - 检查Kubelet的状态、日志等是否存在异常。若存在异常则参考[Kubelet异常处理](#)。
- 检查节点的网络连通性。

常见问题排查方法

检查节点的状态

1. 登录CCE服务控制台。
2. 在界面中选择需要检查节点所在的集群。
3. 在集群列表页面单击“节点管理”，切换到“节点”一栏，查看不可用节点的状态。（如果已安装NPD 1.6.10及以上的插件版本，则在不可用下面会展示指标异常提示，将鼠标光标移动至上方即可看到具体的问题项，若未安装请参考排查项继续排查）。

检查节点监控

1. 登录CCE服务控制台。
2. 在界面中选择需要检查节点所在的集群。
3. 在集群列表页面单击“节点管理”，切换到“节点”一栏，在异常节点所在行单击“监控”。

单击“监控”页签顶部的“更多监控数据”，前往运维管理页面查看历史监控记录。当节点CPU和内存负载过高时，会导致节点网络时延过高，或系统OOM，最终展示为不可用。

检查节点事件

1. 登录CCE服务控制台。
2. 在界面中选择需要检查节点所在的集群。
3. 在集群列表页面单击“节点管理”，切换到“节点”一栏，在异常节点所在行单击“事件”。即可查看是否有节点异常的事件上报（需安装NPD插件）。

弹性云服务器是否删除或故障

1. 登录CCE服务控制台，进入集群，查看不可用节点的名称。
2. 进入ECS控制台，搜索该节点，查看对应的云服务器状态。
 - 若弹性云服务器状态为“已删除”：请在CCE中删除对应节点，再重新创建节点。
 - 若弹性云服务器状态为“关机”或“冻结”：请先恢复弹性云服务器，约3分钟后集群节点可自行恢复。
 - 若弹性云服务器出现故障：请先重启弹性云服务器，恢复故障。
 - 若弹性云服务器状态为“可用”：请参考[检查节点的关键组件](#)进行排查。

弹性云服务是否能够登录

1. 登录ECS控制台。
2. 确认ECS界面显示的节点名称与虚机内的名称是否一致，并且密码或者密钥能否登录。

云服务器信息

ID	e5fb4cf2-60cf-4761-b7ba-66a3837ca56d
名称	弹性云服务器名称
区域	华北-北京
可用区	可用区3
规格	通用计算增强型 2vCPUs 4GiB c7.large.2
镜像	CCE_images_HCE20-Node-24.7.0-test-amd64-ea3c58a9-20240727111... 公共镜像
虚拟私有云	vpc-django-1
全域弹性公网IP	-- 绑定

如果节点名称不一致，且密码和密钥均不能登录，说明ECS创建虚机时的cloudinit初始化问题，临时规避可以尝试重启节点，之后再向[ECS服务提交工单](#)确认根因。

节点的安全组检查

- 节点的安全组被修改导致异常
 - a. 登录VPC控制台，在左侧栏目树中单击“访问控制 > 安全组”，找到集群控制节点的安全组。
 - b. 控制节点安全组名称为：集群名称-cce-control-编号。您可以通过集群名称查找安全组，再进一步在名称中区分“-cce-control-”字样，即为本集群安全组。
 - c. 排查安全组中规则是否被修改，关于安全组的详细说明请参见[集群安全组规则配置](#)。
- 检查安全组规则中是否包含Master和Node互通的安全组策略
 - 请检查安全组规则中是否包含Master和Node互通的安全组策略。
 - 已有集群添加节点时，如果子网对应的VPC新增了扩展网段且子网是扩展网段，要在控制节点安全组（即集群名称-cce-control-随机数）中添加如下三条安全组规则（参考[表4-2](#)），以保证集群添加的节点功能可用（新建集群时如果VPC已经新增了扩展网段则不涉及此场景）。
 - 关于安全组的详细说明请参见[集群安全组规则配置](#)。

表 4-2 需新增的安全组规则

协议端口	类型	源地址
TCP: 8445	IPv4	新增的子网扩展网段
TCP: 9443	IPv4	新增的子网扩展网段
TCP: 5444	IPv4	新增的子网扩展网段

检查节点磁盘

- 新建节点会默认给节点绑定一个100G的运行时专用数据盘（您可以自己选择是否增加）。若数据盘卸载或损坏，会导致运行时异常，最终导致节点不可用。



- 检查节点挂载的数据盘是否已被卸载。若已卸载建议重新创建一个节点，并将不可用节点删除。（不建议您在ECS侧对CCE管理的节点进行操作，以避免出现意外的风险）

检查节点的关键组件

Kubelet

- 查看Kubelet状态

登录对应的异常节点，在节点上执行如下命令，查看Kubelet进程状态。

```
systemctl status kubelet
```

预期输出：

```
[root@192-168-53-14 paas]# systemctl status kubelet
● kubelet.service - Cloud Container Engine Kubelet Service
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2024-08-23 11:07:52 CST; 2 weeks 4 days ago
     Main PID: 9874 (srvkubelet)
        Tasks: 31 (limit: 49655)
       Memory: 119.4M
          CPU: 1d 21h 15min 12.559s
      CGroup: /reserved.slice/kubelet.service
              └─ 9874 /bin/bash /var/script/kubelet/srvkubelet start
                 └─ 9895 /usr/local/bin/kubelet --pod-infra-container-image=cce-pause:3.1 --root-dir=/mnt/paas/kuber

Notice: journal has been rotated since unit was started, output may be incomplete.
```

- 查看Kubelet日志

登录对应的异常节点，在节点上执行如下命令，查看Kubelet进程状态。

```
journalctl -u kubelet
```

运行时

- Docker

- 查看Docker状态

登录对应的异常节点，在节点上执行如下命令，查看docker的进程状态。

```
systemctl status docker
```

预期输出：

```
[root@192-168-53-14 paas]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Drop-In: /etc/systemd/system/docker.service.d
            └─ override.conf
   Active: active (running) since Thu 2024-09-12 15:01:48 CST; 5h 6min ago
     Docs: https://docs.docker.com
    Main PID: 4373 (dockerd)
        Tasks: 129 (limit: 49672)
       Memory: 2.4G
          CPU: 3min 31.735s
      CGroup: /reserved.slice/docker.service
              └─ 4373 /usr/bin/dockerd --live-restore --log-driver=json-file --userland-proxy=false --
                 └─ 4379 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
                    └─ 6068 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                       └─ 6075 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                          └─ 6076 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                             └─ 6111 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                └─ 6441 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                   └─ 6531 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                      └─ 6825 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                         └─ 7077 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                            └─ 7794 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                               └─ 7869 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
                                                  └─ 8042 containerd-shim -namespace moby -workdir /mnt/paas/runtime/containerd/daemon/io.
```

- 查看Docker日志

登录对应的异常节点，在节点上执行如下命令，可查看Docker的日志信息。

```
journalctl -u docker
```

- Containerd

- 查看Containerd状态

登录对应的异常节点，在节点上执行如下命令，查看Containerd进程状态。

```
systemctl status containerd
```

预期输出：

```
[root@ ~]# systemctl status containerd
● containerd.service - Cloud Container Engine Containerd Service
   Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2024-08-29 11:03:09 CST; 1 week 5 days ago
     Main PID: 3847 (containerd)
        Tasks: 128
       Memory: 1.3G
          CPU: 1h 50min 50.523s
      CGroup: /reserved.slice/containerd.service
              └─ 3847 /usr/local/bin/containerd
                  └─ 5299 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id a1e79e56c0ede6fc202da7184f4
                      └─ 1106143 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id c86aa5b1e05d8b8ceabbd39ffdc
                          └─ 1106186 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id 5ba483644413359f60319c68f7c
                              └─ 1185350 /opt/cni/bin/eni-neutron
                                  └─ 3739112 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id 38b6bb8c2a6a5545834c47452c9
                                      └─ 3739756 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id 27e5427373a6ecd0546dbb620e1
                                          └─ 3840273 /usr/local/bin/containerd-shim-runc-v2 --namespace k8s.io --id 1d96ab5316e51a64edd109bd61f

Notice: journal has been rotated since unit was started, output may be incomplete.
```

- 查看Containerd日志

登录对应的异常节点，在节点上执行如下命令，查看Containerd的日志。

```
journalctl -u containerd
```

- NTP组件

- 查看NTP服务状态是否正常

登录对应的异常节点，在节点上执行如下命令，查看Chronyd进程的状态。

```
systemctl status chronyd
```

预期输出：

```
[root@ ~]# systemctl status chronyd
● chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2024-09-06 15:49:40 CST; 4 days ago
     Docs: man:chronyd(8)
           man:chrony.conf(5)
   Process: 4127035 ExecStart=/usr/sbin/chronyd $OPTIONS (code=exited, status=0/SUCCESS)
    Main PID: 4127037 (chronyd)
         Tasks: 1 (Limit: 23124)
        Memory: 956.0K
           CPU: 88ms
      CGroup: /system.slice/chronyd.service
              └─ 4127037 /usr/sbin/chronyd

Notice: journal has been rotated since unit was started, output may be incomplete.
```

- 查看NTP服务日志

登录对应的异常节点，在节点上执行如下命令，查看NTP日志信息。

```
journalctl -u chronyd
```

节点的 DNS 地址配置错误

- 登录节点，在日志/var/log/cloud-init-output.log中查看是否有域名解析失败的报错。可以执行如下指令：

```
cat /var/log/cloud-init-output.log | grep resolv
```

如果回显结果包含如下类似内容则说明无法解析该域名。

```
Could not resolve host: xxx ; Unknown error
```

- 在节点上ping无法解析的域名，例如：

```
ping xxx
```

如果不能，则说明DNS无法解析该地址。请确认/etc/resolv.conf文件中的DNS地址与配置在VPC的子网上的DNS地址是否一致，通常是因为DNS地址配置错误导致无法正常解析域名。

排查是否为包周期节点退订

节点退订后，订单处理需要一定时间。在此期间节点将处于不可用状态，预计5~10分钟后自动清理该节点，无需做额外处理。

常见问题解决方案

节点 PIDPressure

问题根因

节点上的容器占用PID过多导致节点的PID不足。CCE默认留给分配Pod使用的可用PID数的百分比为10%。

问题现象

当节点的可用PID低于pid.available配置项时，则节点状态中PIDPressure为True，同时该节点上的容器被驱逐。关于节点驱逐，可参考社区文档[节点压力驱逐](#)。

解决方案

1. 执行如下命令，查看节点的最大PID数和拥有最多PID的进程。

```
sysctl kernel.pid_max #查看最大PID数  
ps -eLf|awk '{print $2}' | sort -rn| head -n 1 #查看拥有最多PID的进程
```

2. 执行如下命令，查看占用PID最多的前5个进程。

```
ps -eLT | awk '{print $4}' | sort | uniq -c | sort -k1 -g | tail -5
```

预期输出结果：

```
17 1211619  
18 3739112  
18 5299  
24 964  
25 3739756
```

第一列为进程占用的PID数，第二列为当前进程号，根据进程号找到对应进程和所属Pod，分析占用PID过多的原因并优化对应的代码。

3. 降低节点的负载。
4. 如需重启节点，可在ECS界面尝试重启异常节点。（**注意：重启节点可能会导致您的业务中断，请谨慎操作。**）

节点 MemoryPressure

问题根因

节点上的容器占用内存过多导致节点的内存不足。CCE默认节点可用内存值为100 Mi。

问题现象

- 当节点的可用内存低于memory.available配置项时，则节点状态中MemoryPressure为True，同时该节点上的容器被驱逐。关于节点驱逐，可参考社区文档[节点压力驱逐](#)。
- 当节点内存不足时，会有以下常见错误信息：
 - 节点状态中MemoryPressure为True。
 - 当节点上的容器被驱逐时：
 - 在被驱逐的容器事件中可看到关键字The node was low on resource: memory。
 - 在节点事件中可看到关键字attempting to reclaim memory。
 - 可能会导致系统OOM异常，当出现系统OOM异常时，节点事件中可看到关键字System OOM。

解决方案

- 通过节点的监控查看内存增长曲线，确认异常出现时间点，检查节点上的进程是否存在内存泄露现象。具体操作请参见[检查节点监控](#)。
- 降低节点上的负载。
- 如需重启节点，可在ECS界面尝试重启异常节点。（**注意：重启节点可能会导致您的业务中断，请谨慎操作。**）

节点 DiskPressure

问题根因

节点的根文件系统、镜像文件系统或容器文件系统上的可用磁盘空间和 inode 使用过多，已满足驱逐阈值，造成节点异常（nodefs.available、nodefs.inodesFree、imagefs.available、imagefs.inodesFree指标满足驱逐阈值均会造成DiskPressure）。CCE中配置的默认值如下表所示。

参数名称	描述	默认值
nodefs.available	Kubelet 使用的文件系统的可用容量的百分比	10%
nodefs.inodesFree	Kubelet 使用的文件系统的可用inodes数的百分比	5%
imagefs.available	容器运行时存放镜像等资源的文件系统的可用容量的百分比	10%
imagefs.inodesFree	容器运行时存放镜像等资源的文件系统的可用inodes数的百分比	5%

问题现象

- 当节点的可用磁盘空间低于imagefs.available配置项时，则节点状态中DiskPressure为True。
- 当可用磁盘空间低于nodefs.available配置项时，则该节点上的容器全部被驱逐。关于节点驱逐，可参考社区文档[节点压力驱逐](#)。
- 当磁盘空间不足时，通常会有以下常见错误信息：
 - 节点状态中DiskPressure为True。
 - 当触发镜像回收策略后，磁盘空间仍然不足以达到健康阈值（默认为80%），在节点事件中可看到关键字failed to garbage collect required amount of images。
 - 当节点上的容器被驱逐时：
 - 被驱逐的容器事件中可看到关键字The node was low on resource: [DiskPressure]。
 - 节点事件中可看到关键字attempting to reclaim ephemeral-storage或attempting to reclaim nodefs。

解决方案

- 通过节点的监控查看磁盘增长曲线，确认异常出现时间点，检查节点上的进程是否存在占用磁盘空间过多的现象。具体操作请参见[检查节点监控](#)。
- 若有大量文件在磁盘上未清理，请清理文件。
- 根据自身业务情况，限制Pod的ephemeral-storage资源配置。
- 建议使用云存储产品，避免使用HostPath数据卷。
- 扩容节点的磁盘。详细操作请参见[节点磁盘扩容](#)。
- 降低节点的负载。

Kubelet 异常处理

问题根因

通常是kubelet进程异常、kubelet配置有误导致。通常情况下，CCE已默认为kubelet配置了健康检查，因此配置有误导致起不来的可能性更大。

问题现象

Kubelet状态为inactive。

解决方案

1. 登录出现异常的节点，执行如下命令重启Kubelet。重启Kubelet不会影响运行中的容器。

```
systemctl restart kubelet
```
2. 输入以下指令查看Kubelet状态是否恢复正常。

```
systemctl status kubelet
```
3. 若kubelet重启后状态仍异常，请登录节点执行以下命令查看kubelet的日志。

```
journalctl -u kubelet
```

 - 若日志中有明确的错误信息，请根据发生错误时的关键字进行排查。
 - 若确认是kubelet配置异常，请在节点所属的节点池，单击“配置管理”，在kubelet组件配置栏，对kubelet配置进行修改。

节点运行时异常处理-RuntimeOffline

问题根因

通常Docker/Containerd的配置、进程异常等原因导致。

问题现象

- Docker组件
 - Docker的状态为inactive。
 - Docker的状态为active(running)，但是节点未正常工作，导致节点异常。此时通过docker ps 或者 docker exec等命令出现执行失败的现象。
 - 节点状态中的RuntimeOffline为True。
- Containerd组件
 - Containerd的状态为inactive。
 - 节点状态中的RuntimeOffline为True。

解决方案

1. 登录异常的节点。
2. 执行如下命令重启运行时。

```
# Docker
systemctl restart docker
# Containerd
systemctl restart containerd
```
3. 待命令执行完毕之后，查看运行时状态是否正常。

```
# Docker
systemctl status docker
# Containerd
systemctl status containerd
```
4. 若重启运行时，组件的状态仍然异常，请执行如下命令查看组件日志。

```
# Docker
journalctl -u docker
# containerd
journalctl -u containerd
```

NTP 异常处理-NTPPProblem

问题根因

NTP进程状态异常导致。

问题现象

- Chronyd状态为inactive。
- 节点状态中NTPPProblem为True。

解决方案

1. 登录异常节点，执行如下命令重启Chronyd。

```
systemctl restart chronyd
```
2. Chronyd重启之后，执行以下命令，查看Chronyd的状态是否恢复正常。

```
systemctl status chronyd
```
3. 若Chronyd重启之后状态仍然异常，请登录节点执行以下命令查看Chronyd日志。

```
journalctl -u chronyd
```

节点异常重启

问题根因

通常是节点负载异常等原因导致。

问题现象

节点重启的过程中，节点状态为NotReady。

解决方案

1. 执行如下命令，查看节点重启时间。

```
last reboot
```

预期输出：

```
[root@nodepool-50488-uy6bg ~]# last reboot
reboot    system boot  5.10.0-182.0.0.9 Thu Aug 29 11:02    still running
wtmp begins Sat Jul 27 11:21:16 2024
```

2. 查看节点的监控，根据节点重启时间排查出现异常的资源。具体操作请参见[检查节点监控](#)。

3. 查看节点的内核日志，根据重启时间排查异常日志。

节点网络异常

问题根因

通常是节点运行状态异常、安全组配置有误或网络负载过高导致。

问题现象

- 节点无法登录。
- 节点状态Unknown。

解决方案

- 若节点无法登录，请参考以下步骤排查：
 - 在ECS侧检查节点实例状态是否为运行中。
 - 参考是否为ECS的cloud-init执行失败导致。具体操作，可参考[弹性云服务是否能够登录](#)。
 - 检查节点的安全组配置。具体操作，可参考[节点的安全组检查](#)。
- 若节点的网络负载过高，请参考以下步骤进行排查：
 - 通过节点的监控查节点网络增长曲线，检查节点上的Pod是否存在占用网络带宽过多的现象。
 - 使用网络策略控制Pod的网络流量。具体操作，请参考配置[网络策略限制Pod访问的对象](#)。

节点 PLEG 异常-PLEG is not healthy

问题根因

Pod生命周期事件生成器PLEG（Pod Lifecycle Event Generator）会记录Pod生命周期中的各种事件，如容器的启动、终止等。PLEG is not healthy异常通常是由于节点上的运行时进程异常或者节点Systemd版本缺陷导致。

问题现象

- 节点状态NotReady
- 在Kubelet日志中，可看到如下日志。
skipping pod synchronization - PLEG is not healthy: pleg was last seen active 3m17.028393648s ago; threshold is 3m0s.

解决方案

- 依次重启节点关键组件Docker、Containerd、Kubelet，待完成重启之后查看节点是否恢复正常。
- 若节点关键组件重启后节点仍未恢复正常，可尝试重启异常节点。（**重启节点可能会导致您的业务中断，请谨慎操作**）

节点负载过高

问题根因

节点调度资源不足导致。

问题现象

当集群中的节点调度资源不足时，会导致Pod调度失败，出现以下常见错误信息（此处仅列举常见的资源）：

- 集群CPU资源不足：0/2 nodes are available: 2 Insufficient cpu
- 集群内存资源不足：0/2 nodes are available: 2 Insufficient memory
- 集群临时存储不足：0/2 nodes are available: 2 Insufficient ephemeral-storage

其中调度器判定节点资源不足的计算方式为：

- 集群节点CPU资源不足的判定方式：当前Pod请求的CPU资源总量 > （节点可分配的CPU资源总量-节点已分配的CPU资源总量）
- 集群节点内存资源不足的判定方式：当前Pod请求的内存资源总量 > （节点可分配的内存资源总量-节点已分配的内存资源总量）
- 集群节点临时存储不足的判定方式：当前Pod请求的临时存储总量 > （节点可分配的临时存储总量-节点已分配的临时存储总量）

如果当前Pod请求的资源总量大于节点可分配的资源总量减去节点已分配的资源总量，Pod就不会调度到当前节点。

执行以下命令，查看节点的资源分配信息：

```
kubectl describe node $nodeName
```

关注输出中，关于资源分配的部分：

```
Allocatable:
  cpu:          1930m
  ephemeral-storage: 94576560382
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  localssd:      0
  localvolume:   0
  memory:        2511096Ki
  pods:          20
Allocated resources:
 (Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests    Limits
-----
cpu                 1255m (65%) 4600m (238%)
memory              1945Mi (79%) 3876Mi (158%)
ephemeral-storage   0 (0%)      0 (0%)
hugepages-1Gi       0 (0%)      0 (0%)
hugepages-2Mi       0 (0%)      0 (0%)
localssd            0           0
localvolume         0           0
```

其中：

- Allocatable：表示本节点可分配的（CPU/内存/临时存储）资源总量。
- Allocated resources：表示本节点已经分配的（CPU/内存/临时存储）资源总量。

解决方案

当节点调度资源不足时，需降低节点负载，方法如下：

- 减少不必要的Pod，降低节点的负载。
- 根据自身业务情况，限制Pod的资源配置。
- 在集群中添加新的节点。

节点受限调度-存在 node.kubernetes.io/route-unreachable 污点

问题根因

VPC路由网络模式集群依赖网络基础设施的路由表打通容器网络，新创建的CCE节点支持网络隔离，网络未生效系统会自动添加node.kubernetes.io/route-unreachable污点，并在网络就绪后移除该污点。若遇到节点node.kubernetes.io/route-unreachable污点长时间未移除，说明节点存在网络连通性问题。

问题现象

新建节点长时间处于“受限调度”状态。

解决方案

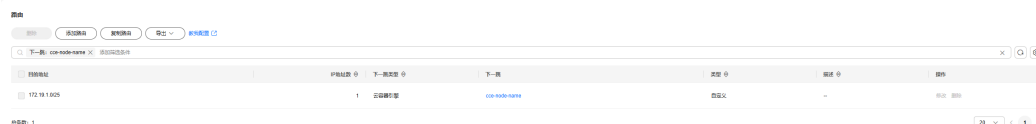
步骤1 若集群内存在其他正常节点，可通过ping命令检查跨节点容器之间的网络连通性。

创建一个测试容器，示例如下，其中{node_ip}为该异常节点的IP。

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - name: container-1
      image: nginx:latest
      imagePullPolicy: IfNotPresent
  imagePullSecrets:
    - name: default-secret
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - {node_ip}
  schedulerName: default-scheduler
  tolerations:
    - key: node.kubernetes.io/route-unreachable
      operator: Exists
      effect: NoSchedule
```

待容器启动后，登录另一正常节点，ping此容器的IP。若通信异常，进行下面第二步。

步骤2 到VPC路由表界面查看该节点路由是否已经添加成功，其中下一跳类型为“云容器引擎”，下一跳为该节点名称。如果路由表中存在该节点的路由仍无法连通网络，则可能是底层网络故障，请提网络工单处理。



步骤3 若通过上述检查都无法解决问题，请提交CCE工单进行排查。

---结束

节点不可用-发生 OOM

问题根因

节点上调度容器较多，资源耗尽导致节点OOM，主要发生在Docker容器引擎节点上。

问题现象

当集群中大量的容器调度到节点上，可能会导致节点OS的panic，节点显示为不可用。通过VNC登录节点可看到如下信息：

```
13513.736557] Out of memory: Killed process 929925 (runc:[2:INIT]) total-vm:482840kB, anon-rss:18624kB, file-rss:0kB, shmem-rss:3748kB
13750.287476] Out of memory: Killed process 946486 (icagent) total-vm:2486152kB, anon-rss:188804kB, file-rss:0kB, shmem-rss:0kB
13975.288781] Out of memory: Killed process 1887445 (icagent) total-vm:2548280kB, anon-rss:288724kB, file-rss:0kB, shmem-rss:0kB
13989.696681] Out of memory: Killed process 1844681 (runc:[2:INIT]) total-vm:556572kB, anon-rss:12060kB, file-rss:0kB, shmem-rss:3748kB
13991.585978] Out of memory: Killed process 1843529 (runc:[2:INIT]) total-vm:556316kB, anon-rss:18144kB, file-rss:0kB, shmem-rss:3768kB
14084.576616] Out of memory: Killed process 1843896 (runc:[2:INIT]) total-vm:556316kB, anon-rss:18140kB, file-rss:0kB, shmem-rss:3748kB
14021.768687] Out of memory: Killed process 1843817 (runc:[2:INIT]) total-vm:556572kB, anon-rss:18128kB, file-rss:0kB, shmem-rss:3736kB
14037.518461] Out of memory: Killed process 1843508 (runc:[2:INIT]) total-vm:556316kB, anon-rss:18128kB, file-rss:0kB, shmem-rss:3712kB
14043.899817] Out of memory: Killed process 1844436 (runc:[2:INIT]) total-vm:556316kB, anon-rss:18044kB, file-rss:0kB, shmem-rss:3732kB
14071.578923] Out of memory: Killed process 1845349 (runc:[2:INIT]) total-vm:556316kB, anon-rss:18028kB, file-rss:0kB, shmem-rss:3748kB
14082.348784] Out of memory: Killed process 1844358 (runc:[2:INIT]) total-vm:556572kB, anon-rss:18028kB, file-rss:0kB, shmem-rss:3692kB
14115.823985] Dead loop on virtual device gw_11cbf51a, fix it urgently!
14116.869973] Dead loop on virtual device gw_11cbf51a, fix it urgently!
14116.572981] Dead loop on virtual device gw_11cbf51a, fix it urgently!
14117.877956] Dead loop on virtual device gw_11cbf51a, fix it urgently!
14117.588965] Dead loop on virtual device gw_11cbf51a, fix it urgently!
14256.444963] Out of memory: Killed process 1865896 (icagent) total-vm:2851840kB, anon-rss:126944kB, file-rss:0kB, shmem-rss:0kB
14492.283423] Out of memory: Killed process 1158151 (runc:[2:INIT]) total-vm:556572kB, anon-rss:12924kB, file-rss:4kB, shmem-rss:3748kB
14496.262777] Out of memory: Killed process 1151198 (runc:[2:INIT]) total-vm:556316kB, anon-rss:12144kB, file-rss:0kB, shmem-rss:3732kB
```

执行以下命令，查看节点的事件：

```
kubectl describe node {nodeName}
```

关注输出中关于节点的异常事件：

Conditions:	Status	LastHeartbeatTime	LastTransitionTime	Reason	Message
NetworkUnavailable	False	Sat, 22 Feb 2025 16:07:07 +0800	Sat, 22 Feb 2025 16:07:07 +0800	YangtzeIsUp	Yangtze is running on this node
MemoryPressure	False	Sat, 22 Feb 2025 17:12:15 +0800	Sat, 22 Feb 2025 17:11:06 +0800	KubeletHasSufficientMemory	kubelet has sufficient memory available
DiskPressure	False	Sat, 22 Feb 2025 17:12:15 +0800	Sat, 22 Feb 2025 17:11:06 +0800	KubeletHasNoDiskPressure	kubelet has no disk pressure
PIDPressure	False	Sat, 22 Feb 2025 17:12:15 +0800	Sat, 22 Feb 2025 17:11:06 +0800	KubeletHasSufficientPID	kubelet has sufficient PID available
Ready	False	Sat, 22 Feb 2025 17:12:15 +0800	Sat, 22 Feb 2025 17:11:07 +0800	KubeletNotReady	PLEG is not healthy: pleg was last seen active 5m1.984571513s ago; threshold is 30s

解决方案

当节点资源不足时，需降低节点负载，方法如下：

- 故障节点通过节点重置恢复。
- 删除或减少不必要的Pod，降低节点的负载。
- 根据自身业务情况，限制Pod的资源配置。
- 在集群中添加新的节点。

4.2 节点创建

4.2.1 CCE 集群新增节点时的问题与排查方法？

注意事项

- 同一集群下的节点镜像保证一致，后续新建/添加/纳管节点时需注意。

- 新建节点时，数据盘如需分配用户空间，分配目录注意不要设置关键目录，例如：如需放到home下，建议设置为/home/test，不要直接写到/home/下。

📖 说明

请注意“挂载路径”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。

排查项一：提示子网可用 IP 不足

问题现象：

CCE集群中新增节点时无法添加新的节点，提示子网剩余可用IP不足。



警告

当前子网剩余可用IP 0 个，请更换子网或减少节点数量

确定

原因分析：

集群默认的节点子网网段较小，子网中的私有IP已用完，无法为节点分配新的私有IP。

解决方法：

- 场景一：VPC网段的IP未分配完**

您可以在创建节点时，在网络配置中选择一个新的节点子网。如果没有可用的节点子网，您可以前往VPC创建一个新的节点子网。详情请参见[为虚拟私有云创建新的子网](#)。

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。

虚拟私有云/节点子网

vpc-ipv6

subnet-f6c6123 (192.168.2.112/28)



子网可用IP数: 0

⚠️ 若子网默认的DNS服务器被修改，需确认自定义的DNS服务器可以解析OBS服务域名，否则无法创建节点

节点IP

自动分配

自定义

弹性公网IP

暂不使用

使用已有

自动创建



● 场景二：VPC网段的IP已分配完

如果整个VPC网段中的IP已分配完，您需要扩容VPC网段，然后创建新的节点子网。

- a. 登录控制台，在服务列表中单击“虚拟私有云 VPC”，在虚拟私有云列表中找到需要扩容的VPC，单击“操作”栏中的“编辑网段”。

如下：



- b. 添加扩展网段后，单击“确定”按钮。



- c. 在左侧导航栏中选择“子网”，单击“创建子网”，为集群所在VPC创建新的子网规划。

详情请参见[为虚拟私有云创建新的子网](#)。



d. 返回CCE新增节点页面，选择新的子网即可创建。

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。

虚拟私有云/子网 vpc-ipv6 subnet-f6c6123 (192.168.2.112/28) 子网可用IP数: 0

节点IP 自动分配 自定义

弹性公网IP 暂不使用 使用已有 自动创建 ?

说明

1. 扩容后原VPC内子网网段正常使用不受影响。如仍无法满足业务需求，可继续新增子网。
2. 同VPC下不同子网间也可以通过内网互信通信。

排查项二：提示弹性 IP 不足

问题现象：

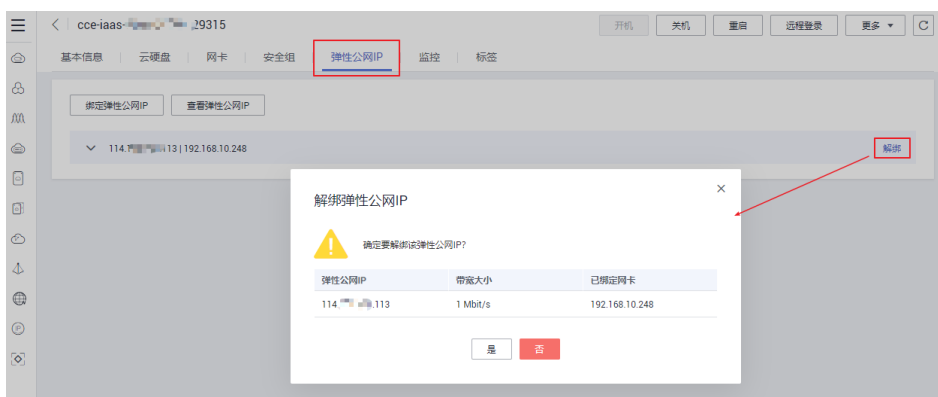
在CCE集群中新增节点时，在弹性公网IP处选择“自动创建”，但创建节点失败，提示弹性IP不足。

解决方法：

您可以有两种方法解决弹性IP不足的问题。

- **方法一：**解绑已绑定弹性IP的虚拟机，再重新添加节点。
 - a. 登录控制台。
 - b. 选择“计算> 弹性云服务 ECS ”。
 - c. 在弹性云服务器列表中，找到待解绑云服务器，单击云服务器名称。
 - d. 在打开的弹性云服务器详情页中，单击弹性公网IP页签，在公网IP列表中单击待解绑IP后的“解绑”，为该云服务器解绑弹性IP，单击“确定”。

图 4-2 解绑弹性公网 IP



e. 返回CCE控制台新增节点页面中，选择“使用已有”重新执行新增节点的操作。

- **方法二：**提高弹性IP的配额。

排查项三：节点安全组是否被修改或删除

问题现象：

在CCE集群中新增节点时创建失败。

解决方法：

您可单击集群名称，查看“概览”页面。在“网络信息”中单击“节点默认安全组”后的按钮，检查集群的节点默认安全组是否被删除，且安全组规则需要满足[集群安全组规则配置](#)。

如果您的账号下含有多个集群，需要统一管理节点的网络安全策略，您也可以指定自定义的安全组，具体操作方法及约束限制请参见[更改集群节点的默认安全组](#)。

网络信息

前往网络配置

网络模型	VPC 网络
VPC	vpc-django-1
子网	subnet-django2 （可用/总IP数: 73/251）
容器网段	172.18.0.0/16 添加
IPv4 服务网段 (IPv4 Service CIDR)	10.247.0.0/16
转发模式	iptables
节点默认安全组	编辑

排查项四：资源配额是否不足

问题现象：

在CCE集群中新增节点时，提示资源配额不足。

提交失败

您所需资源的配额不足，无法购买对应服务。

资源名称	所需配额	剩余配额
CPU(核)	2	1

取消

申请扩大配额

解决方法：

您可单击“申请扩大配额”，前往配额页面进行申请。

排查项五：内网 DNS 域名解析是否成功

创建节点过程中会使用域名方式从OBS下载软件包，需要能够使用云上内网DNS解析OBS域名，否则会导致创建不成功。为此，节点所在子网的DNS服务器地址需要配置为**内网DNS地址**，从而使得节点使用内网DNS。在创建子网时DNS默认配置为内网DNS，如果您修改过子网的DNS，请务必**确保子网下的DNS服务器可以解析OBS服务域名**，否则需要将DNS改成内网DNS。

确认并修改节点所在子网DNS的步骤如下：

- 步骤1** 登录**CCE控制台**，单击集群名称进入集群。
- 步骤2** 查看节点子网。
- 默认节点池：在左侧导航栏中选择“概览”，在“网络信息”中查看子网名称。

网络信息		前往网络配置
网络模型	容器隧道网络	
VPC	vpc-ipv6 🔗	
子网	subnet-831a (可用/总IP数: 21/251)	
容器网段	10.0.0.0/16	
IPv4 服务网段 (IPv4 Service CIDR)	10.247.0.0/16	
转发模式	iptables	
节点默认安全组	 cce-node-79x95 🔗 编辑	

Networking Configuration		View Network Settings
Network Model	Tunnel network	
VPC	vpc-ipv6	
Subnet	subnet-831a (Available/Total IP: 21/251)	
Container CIDR Block	10.0.0.0/16	
IPv4 Service CIDR Block	10.247.0.0/16	
Forwarding	iptables	
Default Node Security Group	cce-node-79x95 Edit	

- 自定义节点池：在左侧导航栏中选择“节点管理”，在“节点池”页签中单击节点池名称，查看节点池关联的子网名称，一个节点池可能同时关联多个子网。



步骤3 确认并修改节点子网的DNS。如果存在多个节点子网，需要逐一进行确认并修改。

1. 登录[VPC控制台](#)，在左侧导航栏中选择“子网”。
2. 在搜索栏中搜索节点所在的子网名称，并单击子网名称。
3. 确认并修改DNS服务器地址，建议添加默认的内网DNS地址，详细操作说明请参见[修改子网信息](#)。



----结束

4.2.2 CCE 集群纳管节点时的常见问题及排查方法？

概述

本文主要介绍纳管/添加已有的ECS实例到CCE集群的常见问题。

须知

- 纳管时，会将所选弹性云服务器操作系统重置为CCE提供的标准镜像，以确保节点的稳定性，请选择操作系统及重置后的登录方式。
- 所选弹性云服务器挂载的系统盘、数据盘都会在纳管时被格式化，请确保信息已备份。
- 纳管过程中，请勿在弹性云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：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。从v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0集群版本开始，待纳管节点无需和集群在同一企业项目下，纳管后节点的企业项目保持不变。
- 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。

排查步骤

您也可以参考以下步骤，通过集群日志查看节点纳管失败的报错信息，然后根据相应的解决方法解决问题：

步骤1 登录CCE控制台，单击集群列表上方的“操作记录”查看具体的报错信息。

步骤2 单击“操作记录”窗口中失败状态的报错信息。

步骤3 根据上一步获取的失败报错信息自行解决后，尝试重新纳管节点。

----结束

常见问题

纳管节点失败，提示已分区磁盘会被忽略，报错内容如下：

```
Install config-prepare failed: exit status 1, output: [ Mon Jul 17 14:26:10 CST 2023 ] start install config-prepare\nNAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT\nsda 8:0 0 40G 0 disk \n└─sda1 8:1 0 40G 0 part \nnsdb 8:16 0 100G 0 disk \n└─sdb1 8:17 0 100G 0 part disk /dev/sda has been partition, will skip this device\nRaw disk /dev/sdb has been partition, will skip this device\nwarning: selector can not match any evs volume
```

请为节点添加一块未分区的数据盘，且数据盘规格为20GiB及以上，即可解决上述问题。纳管完成后，将使用未分区的数据盘作为容器引擎及kubelet组件的存储空间，已分区的数据盘会被忽略不作任何操作，请根据需求自行处理。

4.2.3 纳管节点时失败，报错“安装节点失败”如何解决？

问题描述

节点纳管失败报错安装节点失败。

问题原因

登录节点，查看/var/paas/sys/log/baseagent/baseagent.log安装日志，发现如下报错：

```
net.core.somaxconn=32768
net.ipv4.tcp_max_syn_backlog=8096
PEERDNS=
failed because of no tenant.conf
10310 10:17:41.075997 6872 baseagent.go:330] install failed
E0310 10:17:41.076179 6872 install.go:181] Install Failed: Install Version(v1.13.7-r0) failed: Exec component plugins/config-prepare Install failed: exit status 1
, output: [ Tue Mar 10 10:17:35 CST 2020 ] start install plugins/config-prepare
net.ipv4.ip_forward = 1
net.ipv4.neigh.default.gc_thresh1 = 2048
net.ipv4.neigh.default.gc_thresh2 = 4096
net.ipv4.neigh.default.gc_thresh3 = 8192
net.ipv4.ip_forward=1
```

查看节点LVM设置，发现/dev/vdb没有创建LVM逻辑卷。

解决方案

手工创建逻辑卷：

```
pvccreate /dev/vdb  
vgcreate vgpaas /dev/vdb
```

然后在界面重置节点后节点状态正常。

4.2.4 CCE 支持等保三级认证吗？

云容器引擎CCE服务已通过等保三级认证，您可以在创建节点时进行安全加固，详情请参见[如何进行安全加固](#)。

但在您使用集群前，还需要充分理解云容器引擎的安全责任边界，华为云无法限制您在服务托管范围外的行为，您需要为这部分的行为承担安全责任。详情请参见[责任共担](#)。

如何进行安全加固

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”，在节点配置步骤中设置节点参数。

以下为开启安全加固关键参数设置，其余参数请根据需求设置。

1. 在“操作系统”中选择“Huawei Cloud EulerOS 2.0”。
2. 在“安全加固”设置中选择“等保加固”。

图 4-3 开启安全加固



步骤4 其余节点参数设置完成后，单击“下一步：规格确认”。

步骤5 确认完成后单击“提交”，开始创建节点。

----结束

4.3 节点运行

4.3.1 集群可用但节点状态为“不可用”如何解决？

当集群状态为“可用”，而集群中部分节点状态为“不可用”时，请参照本文提供的排查思路解决。

节点不可用检测机制说明

Kubernetes 节点发送的心跳确定每个节点的可用性，并在检测到故障时采取行动。检测的机制和间隔时间详细说明请参见[心跳](#)。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- [排查项一：节点负载过高](#)
- [排查项二：弹性云服务器是否删除或故障](#)
- [排查项三：弹性云服务器能否登录](#)
- [排查项四：安全组是否被修改](#)
- [排查项五：检查安全组规则中是否包含Master和Node互通的安全组策略](#)
- [排查项六：检查磁盘是否异常](#)
- [排查项七：内部组件是否正常](#)
- [排查项八：DNS地址配置错误](#)
- [排查项九：检查节点中的vdb盘是否被删除](#)
- [排查项十：排查Docker服务是否正常](#)
- [排查项十一：包周期节点退订](#)
- [排查项十二：节点证书未到生效时间](#)

排查项一：节点负载过高

问题描述：

集群中节点连接异常，多个节点报写入错误，业务未受影响。

问题定位：

步骤1 登录CCE控制台，进入集群，在不可用节点所在行单击“监控”。

步骤2 单击“监控”页签顶部的“查看更多”，前往运维管理页面查看历史监控记录。

当节点cpu和内存负载过高时，会导致节点网络时延过高，或系统OOM，最终展示为不可用。

----结束

解决方案：

1. 建议迁移业务，减少节点中的工作负载数量，并对工作负载设置资源上限，降低节点CPU或内存等资源负载。
2. 将集群中对应的cce节点进行数据清理。
3. 限制每个容器的CPU和内存限制配额值。
4. 对集群进行节点扩容。
5. 您也可以重启节点，请至ECS控制台对节点进行重启，重启方法请参见[如何重启弹性云服务器？](#)。

6. 增加节点，将高内存使用的业务容器分开部署。
7. 重置节点，详情请参见[重置节点](#)。

节点恢复为可用后，工作负载即可恢复正常。

排查项二：弹性云服务器是否删除或故障

步骤1 确认集群是否可用。

登录CCE控制台，确定集群是否可用。

- 若集群非可用状态，如错误等，请参见[当集群状态为“不可用”时，如何排查解决？](#)。
- 若集群状态为“运行中”，而集群中部分节点状态为“不可用”，请执行[步骤2](#)。

步骤2 登录ECS控制台，查看对应的弹性云服务器状态。

- 若弹性云服务器状态为“已删除”：请在CCE中删除对应节点，再重新创建节点。
- 若弹性云服务器状态为“关机”或“冻结”：请先恢复弹性云服务器，约3分钟后集群节点可自行恢复。
- 若弹性云服务器出现故障：请先重启弹性云服务器，恢复故障。
- 若弹性云服务器状态为“可用”：请参考[排查项七：内部组件是否正常](#)登录弹性云服务器进行本地故障排查。

----结束

排查项三：弹性云服务器能否登录

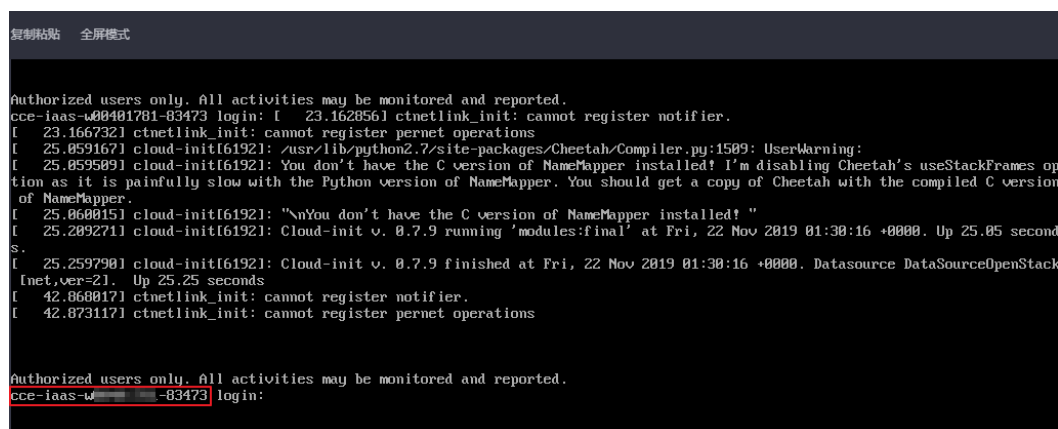
步骤1 登录ECS控制台。

步骤2 确认界面显示的节点名称与虚机内的节点名称是否一致，并且密码或者密钥能否登录。

图 4-4 确认界面显示的名称

名称	cce-iaas-w-83473
状态	运行中
ID	0cc359ab-f2bd-493c-99cb-21ef36c355fb
磁盘	3个
可用区	可用区1
所属订单	CS1911071719N00WJ
委托	-- 新建委托
企业项目	default
云服务器组	-- 新建云服务器组

图 4-5 确认虚拟机内的节点名称和能否登录



如果节点名称不一致，并且密码和密钥均不能登录，说明是ECS创建虚拟机时的cloudinit初始化问题，临时规避可以尝试重启节点，之后再提单给ECS确认问题根因。

----结束

排查项四：安全组是否被修改

登录VPC控制台，在左侧栏目树中单击“访问控制 > 安全组”，找到集群控制节点的安全组。

控制节点安全组名称为：集群名称-cce-**control**-编号。您可以通过**集群名称**查找安全组，再进一步在名称中区分“-cce-control-”字样，即为本集群安全组。

排查安全组中规则是否被修改，关于安全组的详细说明请参见[集群安全组规则配置](#)。

排查项五：检查安全组规则中是否包含 Master 和 Node 互通的安全组策略

请检查安全组规则中是否包含Master和Node互通的安全组策略。

已有集群添加节点时，如果子网对应的VPC新增了扩展网段且子网是扩展网段，要在控制节点安全组（即集群名称-cce-control-随机数）中添加如下三条安全组规则，以保证集群添加的节点功能可用（新建集群时如果VPC已经新增了扩展网段则不涉及此场景）。

关于安全组的详细说明请参见[集群安全组规则配置](#)。



排查项六：检查磁盘是否异常

新建节点会给节点绑定一个100G的docker专用数据盘。若数据盘卸载或损坏，会导致docker服务异常，最终导致节点不可用。

图 4-6 集群新建节点时的数据盘



请检查节点挂载的数据盘是否已被卸载。若已卸载请重新挂载数据盘，再重启节点，节点可恢复。

图 4-7 磁盘检查



排查项七：内部组件是否正常

1. 登录异常节点，查看以下关键组件是否正常运行。

- kubelet
- kube-proxy
- 网络组件：
 - yangtse：VPC网络模型或云原生网络2.0的集群使用该组件
 - canal：容器隧道网络模型的集群使用该组件
- 运行时组件：docker或containerd
- chronyd

查看组件运行状态的命令如下，例如查看kubelet组件运行状态：

```
systemctl status kubelet
```

其中kubelet为组件名称，请根据其他名称进行替换。

预期输出如下：

```
[root@node1 ~]# systemctl status kubelet
● kubelet.service - Cloud Container Engine Kubelet Service
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2025-05-28 16:20:02 CST; 34min ago
     Main PID: 4778 (srvcubelet)
        Tasks: 12 (limit: 23903)
      Memory: 43.2M
         CPU: 19.55%
    CGroup: /reserved.slice/kubelet.service
            └─ 4778 /bin/bash /usr/scripts/kubelet/srvcubelet start
               └─ 4845 /usr/local/bin/kubelet --bootstrap-kubeconfig=/opt/cloud/cc/kubernetes/kubelet/boot.conf --cert-dir=/opt/cloud/cc/kubernetes/kubelet/pki --rotate-certificates

May 28 16:20:02 node1 systemd[1]: Started Cloud Container Engine Kubelet Service.
May 28 16:20:02 node1 sh[4787]: DAEMON_ARGS=" --bootstrap-kubeconfig=/opt/cloud/cc/kubernetes/kubelet/boot.conf --cert-dir=/opt/cloud/cc/kubernetes/kubelet/pki --rotate-certificates"
May 28 16:20:02 node1 sh[4788]: [/usr/local/bin/kubelet] start a background script to retrieve bootstrap token
May 28 16:20:02 node1 su[4799]: (to paas) root on none
May 28 16:20:02 node1 su[4799]: pam_unix(su-l:session): session opened for user paas(uid=10000) by (uid=0)
May 28 16:20:02 node1 su[4799]: pam_unix(su-l:session): session closed for user paas
```

2. 若服务的组件状态不是Active，请重启对应的组件，重启命令请根据出错组件指定，如yangtse组件出错，则命令为：

```
systemctl restart yangtse
```

重启后再查看状态：

```
systemctl status yangtse
```

3. 如果组件异常但重启后依然无法恢复节点状态，请提交工单联系客服进行处理。

排查项八：DNS 地址配置错误

步骤1 登录节点，在日志/var/log/cloud-init-output.log中查看是否有域名解析失败相关的报错。

```
cat /var/log/cloud-init-output.log | grep resolv
```

如果回显包含如下内容则说明无法解析该域名。

Could not resolve host: test.obs.cn-north-4.myhuaweicloud.com; Unknown error

步骤2 在节点上ping上一步无法解析的域名，确认节点上能否解析此域名。

```
ping test.obs.cn-north-4.myhuaweicloud.com
```

- 如果不能，则说明DNS无法解析该地址。请确认/etc/resolv.conf文件中的DNS地址与配置在VPC的子网上的DNS地址是否一致，通常是由于此DNS地址配置错误，导致无法解析此域名。请修改VPC子网DNS为正确配置，然后重置节点。
- 如果能，则说明DNS地址配置没有问题，请排查其他问题。

----结束

排查项九：检查节点中的 vdb 盘是否被删除

如果节点中的vdb盘被删除，可参考[此章节内容](#)恢复节点。

排查项十：排查 Docker 服务是否正常

步骤1 执行以下命令确认docker服务是否正在运行：

```
systemctl status docker
```

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2021-02-03 16:07:02 CST; 1 day 23h ago
     Docs: https://docs.docker.com
    Main PID: 3673 (dockerd)
      Tasks: 46 (limit: 24004)
     Memory: 491.2M
    CGroup: /system.slice/docker.service
            └─3673 /usr/bin/dockerd --live-restore --log-opt max-size=50m --log-opt max-file=20 --log-driver=json-fil
              └─3680 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
                └─5961 containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v
                  └─6811 containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v

Warning: Journal has been rotated since unit was started. Log output is incomplete or unavailable.
```

若执行失败或服务状态非active，请确认docker运行失败原因，必要时可提交工单联系技术支持。

步骤2 执行以下命令检查当前节点上所有容器数量：

```
docker ps -a | wc -l
```

若命令卡死、执行时间过长或异常容器数过多（1000以上），请确认外部是否存在重复不断地创删负载现象，在大量容器频繁创删过程中有可能出现大量异常容器且难以及时清理。

在此场景下可考虑停止重复创删负载或采用更多的节点去分摊负载，一般等待一段时间后节点会恢复正常，必要情况可执行docker rm {container_id}手动清理异常容器。

----结束

排查项十一：包周期节点退订

节点退订后，订单处理需要一定时间。在此期间节点将处于不可用状态，预计5~10分钟后自动清理该节点。

排查项十二：节点证书未到生效时间

当集群所在地区出现夏令时与冬令时转换时，在夏令时与冬令时的重叠时间，例如在凌晨2点申请创建节点，转变为冬令时后，时间变为凌晨1点，可能导致节点不可用。

问题根因

节点内证书的生效时间在未来，而不是当前时间，在证书未生效时，会导致节点对kube-apiserver的请求被拒绝。

解决方案

登录节点，执行以下命令查看证书。

```
ll /opt/cloud/cce/kubernetes/kubelet/pki
```

```
total 20
-rw-r----- 1 root root 1448 Nov 12 17:11 kubelet-client-2024-11-12-17-11-49.pem
lrwxrwxrwx 1 root root   76 Nov 12 17:11 kubelet-client-current.pem -> /opt/cloud/cce/kubernetes/kubelet/pki/kubelet-client-2024-11-12-17-11-49.pem
-rw-r----- 1 root root 1472 Nov 12 17:11 kubelet-server-2024-11-12-17-11-52.pem
-rw-r----- 1 root root 1476 Nov 12 17:24 kubelet-server-2024-11-12-17-24-54.pem
lrwxrwxrwx 1 root root   76 Nov 12 17:24 kubelet-server-current.pem -> /opt/cloud/cce/kubernetes/kubelet/pki/kubelet-server-2024-11-12-17-24-54.pem
```

文件名称即为证书创建时间，请确认证书创建时间是否晚于当前时间。

如果证书生效的时间晚于当前时间，您可以采取以下措施之一解决：

- 删除节点重新创建。
- 等待证书生效时间到达，节点会自动变成可用状态。
- 联系运维人员更新节点证书。

4.3.2 CCE 集群中的节点无法远程登录，如何排查解决？

CCE创建节点成功后，无法ssh远程登录。ssh回显提示“所选的用户密钥未在远程主机上注册”，即root用户不能直接登录到节点。

出现上述问题的原因是CCE创建的节点安装了cloudinit，有默认的linux用户，并且该密钥也是用于linux。

解决方法

使用linux用户登录，使用`sudo su`命令切换到root用户。

4.3.3 如何重置 CCE 集群中节点的密码？

问题背景

在CCE中创建节点时，您选择了使用密钥对或者密码作为登录方式，当密钥对或密码丢失时，您可以登录ECS控制台对节点进行密码重置操作，重置密码后即可使用密码登录CCE服务中的节点。

操作步骤

- 步骤1** 登录ECS控制台。
- 步骤2** 在左侧弹性云服务器列表中，选择待操作节点对应的云服务器，单击后方操作列中的“更多 > 关机”。
- 步骤3** 待云服务器关机后，单击待操作节点后方操作列中的“更多 > 重置密码”，按照界面提示进行操作即可重置密码。
- 步骤4** 密码重置完成后，单击待操作节点后方操作列中的“更多 > 开机”，单击后方的“远程登录”即可通过密码登录该节点。

----结束

4.3.4 如何收集 CCE 集群中节点的日志？

节点日志路径

CCE节点日志文件如下表所示。

表 4-3 节点日志列表

日志名称	路径
kubelet日志	• v1.21及以上版本集群：/var/log/cce/kubernetes/kubelet.log • v1.19及以下版本集群：/var/paas/sys/log/kubernetes/kubelet.log
kube-proxy日志	• v1.21及以上版本集群：/var/log/cce/kubernetes/kube-proxy.log • v1.19及以下版本集群：/var/paas/sys/log/kubernetes/kube-proxy.log
yangtse日志（网络）	• v1.21及以上版本集群：/var/log/cce/yangtse • v1.19及以下版本集群：/var/paas/sys/log/yangtse
canal日志	• v1.21及以上版本集群：/var/log/cce/canal • v1.19及以下版本集群：/var/paas/sys/log/canal
系统日志	/var/log/messages
容器引擎日志	• docker节点：/var/lib/docker • containerd节点：/var/log/cce/containerd

表 4-4 插件日志列表

插件日志名称	路径
everest插件日志	• 2.1.41及以上版本插件： – everest-csi-driver：/var/log/cce/kubernetes – everest-csi-controller：/var/paas/sys/log/kubernetes • 2.1.41以下版本插件： – everest-csi-driver：/var/log/cce/everest-csi-driver – everest-csi-controller：/var/paas/sys/log/everest-csi-controller
npd插件日志	• 1.18.16及以上版本插件：/var/paas/sys/log/kubernetes • 1.18.16以下版本插件：/var/paas/sys/log/cceaddon-npd
cce-hpa-controller插件日志	• 1.3.12及以上版本插件：/var/paas/sys/log/kubernetes • 1.3.12以下版本插件：/var/paas/sys/log/ccehpa-controller

配置节点日志采集

您可以方便地使用CCE云原生日志采集插件采集节点日志并上报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](#)。

4.3.5 如何解决 yum update 升级操作系统导致的容器网络不可用问题？

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。

您可以通过如下方式手动恢复：

须知

当前该恢复方式仅针对EulerOS 2.2有效。

步骤1 root下执行如下脚本：

```
#!/bin/bash
function upgrade_kmod()
{
    openvswitch_mod_path=$(rpm -qal openvswitch-kmod)
    rpm_version=$(rpm -qal openvswitch-kmod|grep -w openvswitch|head -1|awk -F "/" '{print $4}')
    sys_version=$(cat /boot/grub2/grub.cfg | grep EulerOS|awk 'NR==1{print $3}' | sed 's/[()]//g')

    if [[ "${rpm_version}" != "${sys_version}" ]];then
        mkdir -p /lib/modules/"${sys_version}"/extra/openvswitch
        for path in ${openvswitch_mod_path[@]};do
            name=$(echo "$path" | awk -F "/" '{print $NF}')
            rm -f /lib/modules/"${sys_version}"/updates/"${name}"
            rm -f /lib/modules/"${sys_version}"/extra/openvswitch/"${name}"
            ln -s "${path}" /lib/modules/"${sys_version}"/extra/openvswitch/"${name}"
        done
        fi
        depmod ${sys_version}
    }
}
upgrade_kmod
```

步骤2 执行完成后，重启虚拟机。

----结束

相关链接

- [集群节点高危操作](#)

4.3.6 Node 节点 vdb 盘受损，通过重置节点仍无法恢复节点？

问题现象

客户node节点vdb盘受损，通过重置节点，无法恢复节点。

问题过程：

- 在一个正常的node节点上，删除lv，删除vg，节点不可用。
- 重置异常节点，重置过程中，报语法错误，而且节点不可用。

如下图：

```
vgcreate VG_new PV ...
create volume group error
, skip pause's work in case of failed dependency docker, skip fuxi's work in case of failed dependency docker, sk
work in case of failed dependency kubelet, skip kube-proxy's work in case of failed dependency config-prepare, sk
ork in case of failed dependency config-prepare, skip canal-agent's work in case of failed dependency fuxi, skip c
work in case of failed dependency config-prepare, skip docker's work in case of failed dependency config-prepare,
s work in case of failed dependency config-prepare]
10525 17:22:55.835605 7116 install.go:361 install failed
Install Failed: [Install config-prepare failed: exit status 1, output: [ Mon May 25 17:22:53 CST 2020 ] start inst
pare
success download the file
success download the file
success download the file
success download the file
success download the file
success download the file
success download the file
Checking device: /dev/vda
Raw disk /dev/vda has been partition, will skip this device
Checking device: /dev/vdb
Detected paas disk: /dev/vdb
Use to config lv(eg. docker(direct-lvm),kubelet,user)
No command with matching syntax recognised. Run 'vgcreate --help' for more information.
Correct command syntax is:
vgcreate VG_new PV ...
create volume group error
, skip pause's work in case of failed dependency docker, skip fuxi's work in case of failed dependency docker, sk
work in case of failed dependency kubelet, skip kube-proxy's work in case of failed dependency config-prepare, sk
ork in case of failed dependency config-prepare, skip canal-agent's work in case of failed dependency fuxi, skip c
work in case of failed dependency config-prepare, skip docker's work in case of failed dependency config-prepare,
s work in case of failed dependency config-prepare]
```

问题定位

node节点中vg被删除或者损坏无法识别，为了避免重置的时候误格式化用户的数据盘，需要先手动恢复vg，这样重置的时候就不会去格式化其余的数据盘。

解决方案

步骤1 登录节点。

步骤2 重新创建PV和VG，但是创建时报错：

```
root@host1:~# pvcreate /dev/vdb
Device /dev/vdb excluded by a filter
```

这是由于添加的磁盘是在另一个虚拟机中新建的，已经存在了分区表，当前虚拟机并不能识别磁盘的分区表，运行parted命令重做分区表，中途需要输入三次命令。

```
root@host1:~# parted /dev/vdb
GNU Parted 3.2
Using /dev/vdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel msdos
Warning: The existing disk label on /dev/vdb will be destroyed and all data on this disk will be lost. Do you
want to continue?
Yes/No? yes
(parted) quit
Information: You may need to update /etc/fstab.
```

再次运行pvcreate，当询问是否擦除dos签名时，输入y，就可以将磁盘创建为PV。

```
root@host1:~# pvcreate /dev/vdb
WARNING: dos signature detected on /dev/vdb at offset 510. Wipe it? [y/n]: y
Wiping dos signature on /dev/vdb.
Physical volume "/dev/vdb" successfully created
```

步骤3 创建VG。

判断该节点的docker盘，如果是/dev/vdb和/dev/vdc两个盘，则执行下面的命令：

```
root@host1:~# vgcreate vgpaas /dev/vdb /dev/vdc
```

如果只有/dev/vdb盘，则执行下面的命令：

```
root@host1:~# vgcreate vgpaas /dev/vdb
```

创建完成后，重置节点即可恢复。

----结束

4.3.7 CCE 集群节点中安装 kubelet 的端口主要有哪些？

CCE集群节点中安装kubelet的端口主要有如下几个：

- 10250 -port: kubelet服务与API Server通信的端口。
- 10248 -healthz-port: 健康检查服务的端口。
- 10255 -read-only-port: 只读端口，用于对外提供监控指标。

4.3.8 如何配置 Pod 使用 GPU 节点的加速能力？

问题描述

我已经购买了GPU节点，但运行速度还是很慢，请问如何配置Pod使用GPU节点的加速能力。

解答

方案1：

建议您将集群中GPU节点的不可调度的污点去掉，以便GPU插件驱动能够正常安装，同时您需要安装高版本的GPU驱动。

如果您的集群中有非GPU的容器，可以通过亲和、反亲和策略将这个容器不调度到GPU节点上。

方案2：

建议您安装高版本的GPU驱动，通过kubect更新GPU插件的配置，增加配置如下：

```
tolerations:  
- operator: "Exists"
```

增加该配置后，可以使GPU插件驱动能够正常安装到打了污点的GPU节点上。

4.3.9 容器使用 SCSI 类型云硬盘偶现 IO 卡住如何解决？

问题描述

容器使用SCSI类型的云硬盘存储，在CentOS节点上创建和删除容器触发磁盘频繁挂载卸载的场景，有概率会出现系统盘读写瞬时冲高，然后系统卡住的问题，影响节点正常工作。

出现该问题时，可在dmesg日志中观察到：

```
Attached SCSI disk  
task jdb2/xxx blocked for more than 120 seconds.
```

如下图红框所示：

```
1128163.173120] sd 2:0:0:0: [sda] Write Protect is off  
1128163.173457] sd 2:0:0:0: [sda] Mode Sense: 69 00 00 08  
1128163.173573] sd 2:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA  
1128163.176426] sd 2:0:0:0: [sda] Attached SCSI disk  
1128350.437941] INFO: task jdb2/dm-1-8:1604 blocked for more than 120 seconds.  
1128350.438267] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.  
1128350.438564] jdb2/dm-1-8    D ffff9ede7f8420e0    0 1604    2 0x00000000  
1128350.438829] Call Trace:  
1128350.439120] [] ? blk_mq_dispatch_rq_list+0x325/0x620  
1128350.439394] [] schedule+0x29/0x70
```


问题原理

BUS 0上热插PCI设备后，Linux内核会多次遍历挂载在BUS 0上的所有PCI-Bridge，且PCI-Bridge在被更新期间无法正常工作。在此期间，若设备使用的PCI-Bridge被更新，由于内核缺陷，该设备会认为PCI-Bridge异常，设备进入故障模式进而无法正常工作。如果此时前端正要写PCI配置空间让后端处理磁盘IO，那么这个写配置空间操作就会被剔除，导致后端接收不到通知去处理IO环上的新增请求，最终表现为前端IO卡住。

影响范围

对CentOS Linux内核3.10.0-1127.el7之前的版本有影响。

解决方法

通过[重置节点](#)将内核升级至高版本，具体请参见[重置节点](#)。

4.3.10 docker 审计日志量过大影响磁盘 IO 如何解决?

问题描述

部分集群版本的存量节点docker审计日志量较大，由于操作系统内核缺陷，会低概率出现IO卡住。该问题可通过优化审计日志规则，降低问题出现的概率。

影响范围

受影响的集群版本:

- v1.15.11-r1
- v.1.17.9-r0

须知

- 只需对已有节点进行修复，新建节点默认无此问题。
- 升级过程需要重启auditd组件。

检查方法

步骤1 以root用户登录node节点。

步骤2 执行以下命令检查当前节点是否存在该问题:

```
auditctl -l | grep "/var/lib/docker -p rwx -k docker"
```

如果有类似下图的回显，则说明存在该问题，需要进行修复，如果无回显则说明节点不受此问题影响。

```
[root@localhost ~]# auditctl -l | grep "/var/lib/docker -p rwx -k docker"
```

----结束

解决方法

步骤1 以root用户登录node节点。

步骤2 执行如下命令：

```
sed -i "/\var/lib/docker -k docker/d" /etc/audit/rules.d/docker.rules  
service auditd restart  
----结束
```

验证方法

执行以下命令检查问题是否修复：

```
auditctl -l | grep "/var/lib/docker -p rwx -k docker"
```

如果无回显，则说明问题已修复。

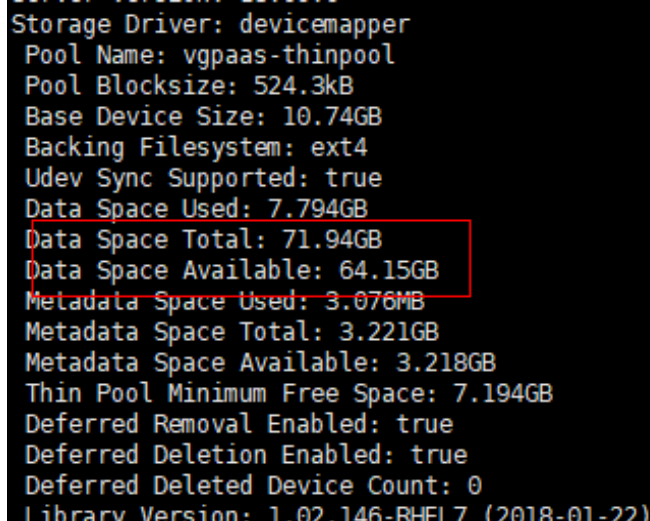
4.3.11 thinpool 磁盘空间耗尽导致容器或节点异常时，如何解决？

问题描述

当节点上的thinpool磁盘空间接近写满时，概率性出现以下异常：

在容器内创建文件或目录失败、容器内文件系统只读、节点被标记disk-pressure污点及节点不可用状态等。

用户可手动在节点上执行docker info查看当前thinpool空间使用及剩余量信息，从而定位该问题。如下图：



```
Storage Driver: devicemapper  
Pool Name: vgpaas-thinpool  
Pool Blocksize: 524.3kB  
Base Device Size: 10.74GB  
Backing Filesystem: ext4  
Udev Sync Supported: true  
Data Space Used: 7.794GB  
Data Space Total: 71.94GB  
Data Space Available: 64.15GB  
Metadata Space Used: 3.076MB  
Metadata Space Total: 3.221GB  
Metadata Space Available: 3.218GB  
Thin Pool Minimum Free Space: 7.194GB  
Deferred Removal Enabled: true  
Deferred Deletion Enabled: true  
Deferred Deleted Device Count: 0  
Library Version: 1.02.146-BHEL 7 (2018-01-22)
```

问题原理

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

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

解决方案

当节点已出现thinpool空间耗尽时，可将部分业务迁移至其他节点实现业务快速恢复。但对于此类问题，建议采用以下方案从根因上解决问题：

方案1：

合理规划业务分布及数据面磁盘空间，避免和减少出现**业务容器数*basesize > 节点thinpool空间大小**场景。如需对thinpool空间进行扩容，请参考以下步骤：

步骤1 在EVS控制台扩容数据盘。详情请参见[扩容云硬盘容量](#)。

在EVS控制台扩容成功后，仅扩大了云硬盘的存储容量，还需要执行后续步骤扩容逻辑卷和文件系统。

步骤2 登录CCE控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤3 登录目标节点。

步骤4 使用lsblk命令查看节点块设备信息。

这里存在两种情况，根据容器存储Rootfs而不同。

Overlayfs：没有单独划分thinpool，在dockersys空间下统一存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0    0  50G  0 disk
└─sda1       8:1    0  50G  0 part /
sdb          8:16   0 150G  0 disk # 数据盘已扩容至150G，存在50G空间仍未分配
└─vgpaas-dockersys 253:0   0  90G  0 lvm  /var/lib/containerd
   └─vgpaas-kubernetes 253:1   0  10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

将新增的磁盘容量加到容器引擎使用的dockersys逻辑卷上。

a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/sdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/sdb
```

回显如下：

```
Physical volume "/dev/sdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <90.00 GiB (23039 extents) to 140.00 GiB (35840 extents).
Logical volume vgpaas/dockersys successfully resized.
```

c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/containerd; on-line resizing required
old_desc_blocks = 12, new_desc_blocks = 18
The filesystem on /dev/vgpaas/dockersys is now 36700160 blocks long.
```

3. 检查是否扩容成功。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    0   50G  0 disk
└─sda1               8:1    0   50G  0 part /
sdb                  8:16   0  150G  0 disk
├─vgpaas-dockersys  253:0    0  140G  0 lvm  /var/lib/containerd
└─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

Devicemapper：单独划分了thinpool存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                  8:0    0   50G  0 disk
└─vda1               8:1    0   50G  0 part /
vdb                  8:16   0  200G  0 disk
├─vgpaas-dockersys  253:0    0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta 253:1    0    3G  0 lvm
├─vgpaas-thinpool      253:3    0   67G  0 lvm          # thinpool空间
├─...
├─vgpaas-thinpool_tdata 253:2    0   67G  0 lvm
├─vgpaas-thinpool      253:3    0   67G  0 lvm
├─...
└─vgpaas-kubernetes  253:4    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

选项一：将新增的磁盘容量加到thinpool盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/vdb为thinpool空间所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/thinpool为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/thinpool
```

回显如下：

```
Size of logical volume vgpaas/thinpool changed from <67.00 GiB (23039 extents) to <167.00 GiB (48639 extents).
Logical volume vgpaas/thinpool successfully resized.
```

- c. 由于thinpool未挂载到设备，因此无需调整文件系统的大小。
- d. 检查是否扩容成功。使用lsblk命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到thinpool盘，则表示扩容成功。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                  8:0    0   50G  0 disk
└─vda1               8:1    0   50G  0 part /
vdb                  8:16   0  200G  0 disk
├─vgpaas-dockersys  253:0    0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta 253:1    0    3G  0 lvm
├─vgpaas-thinpool      253:3    0  167G  0 lvm          # 扩容后的thinpool空间
├─...
├─vgpaas-thinpool_tdata 253:2    0   67G  0 lvm
├─vgpaas-thinpool      253:3    0   67G  0 lvm
├─...
└─vgpaas-kubernetes  253:4    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

选项二：将新增的磁盘容量加到dockersys盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/vdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <18.00 GiB (4607 extents) to <118.00 GiB (30208 extents).
Logical volume vgpaas/dockersys successfully resized.
```

- c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/docker; on-line resizing required
old_desc_blocks = 3, new_desc_blocks = 15
The filesystem on /dev/vgpaas/dockersys is now 30932992 blocks long.
```

- d. 检查是否扩容成功。使用lsblk命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到dockersys盘，则表示扩容成功。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                 8:0    0   50G  0 disk
├─vda1                             8:1    0   50G  0 part /
└─vdb                               8:16   0  200G  0 disk
   └─vgpaas-dockersys              253:0    0  118G  0 lvm  /var/lib/docker # 扩容后的
      dockersys盘
         └─vgpaas-thinpool_tmeta    253:1    0    3G  0 lvm
            └─vgpaas-thinpool      253:3    0   67G  0 lvm
               ...
         └─vgpaas-thinpool_tdata    253:2    0   67G  0 lvm
            └─vgpaas-thinpool      253:3    0   67G  0 lvm
               ...
         └─vgpaas-kubernetes        253:4    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

----结束

方案2：

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。

方案3：

使用overlayfs存储模式的操作系统，可将业务部署在此类节点上，避免容器内创删文件后占用的磁盘空间不立即释放问题。

4.3.12 CCE 节点上监听的端口列表

表 4-5 Node 节点监听端口

目的端口	协议	端口说明
10248	TCP	kubelet健康检查端口

目的端口	协议	端口说明
10250	TCP	kubelet服务端端口，提供节点上工作负载的监控信息和容器的访问通道
10255	TCP	kubelet只读端口，提供节点上工作负载的监控信息
动态端口（与宿主机限制的范围有关，比如 内核参数 net.ipv4.ip_local_port_range）	TCP	kubelet 随机监听一个端口，与CRI Shim通信获取Exec URL
10249	TCP	kube-proxy metric端口，提供kube-proxy组件的监控信息
10256	TCP	kube-proxy健康检查端口
动态端口（32768~65535）	TCP	docker exec等功能的websocket监听端口
动态端口（32768~65535）	TCP	containerd exec等功能的websocket监听端口
28001	TCP	icagent本地侦听端口，接受节点 syslog日志
28002	TCP	icagent健康检查端口
20101	TCP	yangtse-agent/canal-agent健康检查端口(容器隧道网络模式涉及)
20104	TCP	yangtse-agent/canal-agent的 metric端口，提供组件的监控信息(容器隧道网络模式涉及)
3125	TCP	everest-csi-driver侦听健康检查端口
3126	TCP	everest-csi-driver pprof端口
19900	TCP	node-problem-detector 健康检查 server端口
19901	TCP	node-problem-detector对接普罗采集监控数据端口
4789	UDP	ovs侦听端口，容器网络vxlan报文的传输通道(容器隧道网络模式涉及)
4789	UDIPv6	ovs侦听端口，容器网络vxlan报文的传输通道(容器隧道网络模式涉及)

目的端口	协议	端口说明
动态端口 30000~32767	TCP	kube-proxy侦听端口，做4层负载均衡。K8s会给NodePort和Loadbalancer类型的服务分配一个随机端口，默认范围在30000~32767
动态端口 30000~32767	UDP	kube-proxy侦听端口，做4层负载均衡。K8s会给NodePort和Loadbalancer类型的服务分配一个随机端口，默认范围在30000~32767
123	UDP	ntpd侦听端口，负责时间同步
20202	TCP	PodLB侦听端口，做7层负载均衡，负责转发容器镜像拉取请求

4.3.13 GPU 节点使用 nvidia 驱动启动容器排查思路

集群中的节点是否有资源调度失败的事件？

问题现象：

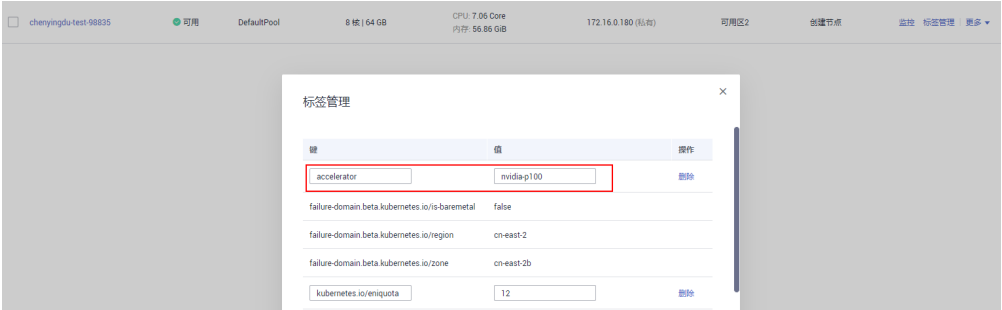
节点运行正常且有GPU资源，但报如下失败信息：

0/9 nodes are available: 9 insufficient nvidia.com/gpu

排查思路：

1. 确认节点标签是否已经打上nvidia资源。

```
root@chenyingdu-test-98835 ~]# kubectl get node --show-labels
NAME              STATUS    ROLES    AGE   VERSION   LABELS
172.16.0.180      Ready    <none>   6h26m v1.13.10-r1-CCE2.0.28.B001 accelerator=nvidia-p100 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/is-baremetal=false,failure-domain.beta.kubernetes.io/region=,failure-domain.beta.kubernetes.io/zone=,kubernetes.io/availablezone=,kubernetes.io/eniquote=12,kubernetes.io/hostname=172.16.0.180,node.kubernetes.io/subnetid=4883a3c2-f09f-412d-bd3a-5a2092c5833a,os.architecture=amd64,os.name=EulerOS_2.0_SP5,os.version=3.10.0-862.14.1.2.h249.eulerosv2r7.x86_64
root@chenyingdu-test-98835 ~]#
```



2. 查看nvidia驱动运行是否正常。

到插件运行所在的节点上，查看驱动的安装日志，路径如下所示：
/opt/cloud/cce/nvidia/nvidia_installer.log

查看nvidia容器标准输出日志：

过滤容器id

```
docker ps -a | grep nvidia
```

查看日志

```
docker logs 容器id
```

业务上报 nvidia 版本和 cuda 版本不匹配？

容器中查看cuda的版本，执行如下命令：

```
cat /usr/local/cuda/version.txt
```

然后查看容器所在节点的nvidia驱动版本支持的cuda版本范围，是否包含容器中的cuda版本。

相关链接

[工作负载异常：GPU节点部署服务时报错](#)

4.3.14 CCE 节点 NTP 时间不同步如何解决？

问题现象

节点上的ntpd在长时间无法连接ntpserver等特殊场景下，可能导致偏移量过大，无法自动恢复。

问题检测

CCE节点故障检测插件（npd）中已包含节点时间同步检查项，您可以在集群中安装该插件进行检测。详情请参见[CCE节点故障检测](#)。

问题根因

EulerOS和CentOS类型的节点存在由NTP引起的已知问题，其他类型的节点不涉及该问题。

说明

上述问题在v1.19.16-r7、v1.21.9-r10、v1.23.7-r10版本的集群中被修复。

解决方法

- 若您的集群版本为v1.19.16-r7、v1.21.9-r10、v1.23.7-r10及以上，该版本的节点已经切换至chronyd时间同步，请重置节点即可修复该问题。
- 若您的集群版本不满足要求，建议您将集群升级至v1.19.16-r7、v1.21.9-r10、v1.23.7-r10及以上版本，再重置节点。

4.3.15 Containerd 节点业务容器标准输出日志写入过快导致节点数据盘使用率过高

问题现象

Containerd节点上业务容器标准输出不断写入大量日志，导致/var/lib/containerd目录占用空间不断增长，同时节点上容器创删速度变慢，进一步出现磁盘使用率过高、Pod驱逐、节点异常等现象。

问题根因

对于使用Containerd运行时的节点上业务容器，若日志输出方式采用容器标准输出，其日志转储由节点上kubelet组件完成，除负责业务容器标准输出日志转储外，kubelet还负责节点上所有容器生命周期的维护操作。

若节点上业务容器过多，业务容器标准日志输出过快，会导致kubelet持续高负荷运行，超过一定限度会出现日志转储，进而日志累积导致磁盘使用率过高。同时因为kubelet高负荷运行此时节点上容器创删等操作也会受影响。

解决方法

一般场景下，以8U16G节点、数据盘大小为100G为例，建议单容器的日志标准输出速率不超过512KB/s，节点上所有容器总体日志标准速率建议不超过5MB/s。若确实存在大量日志输出场景，可考虑以下方式优化：

- 避免日志输出过多的容器调度在同一个节点。例如给此类应用配置Pod间反亲和，或减少单节点的Pod数量上限。
- 考虑单独挂盘，如用户创建节点时挂载额外用户数据盘或应用动态挂载存储等等，然后将业务日志输出到额外挂载盘中的文件。

4.3.16 为什么 kubectl top 命令查看节点内存使用超过 100%？

问题现象

从界面上看节点内存使用率并不是很高，但使用**kubelet top node**查看节点内存使用率已超过100%。

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
192.168.0.243	79m	4%	2357Mi	109%

问题根因

出现该问题的原因是**kubectl top node**是调用kubelet的metrics API来获取数据的，因此看到的是节点上已使用的资源总和除以可分配的所有资源。

社区issue链接：<https://github.com/kubernetes/kubernetes/issues/86499>。

场景示例

例如，某节点的参数可通过**kubectl describe node**命令查询，示例如下：

```
...
Capacity:
  cpu:                2
  ephemeral-storage:  51286496Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  localssd:           0
  localvolume:        0
  memory:             4030180Ki
  pods:               40
Allocatable:
  cpu:                1960m
  ephemeral-storage:  47265634636
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  localssd:           0
  localvolume:        0
```

```
memory:      2213604Ki
pods:        40
...
```

- 节点内存总量：即Capacity.memory字段，为4030180Ki。
- 节点可分配量：即Allocatable.memory字段，为2213604Ki。
- 节点已使用量：可通过以下命令获取，本示例中为2413824Ki。
`kubectl get --raw /apis/metrics.k8s.io/v1beta1/nodes/`

回显如下：

```
{
  "kind": "NodeMetricsList",
  "apiVersion": "metrics.k8s.io/v1beta1",
  "metadata": {},
  "items": [
    {
      ...
      "timestamp": "2023-08-15T14:09:38Z",
      "window": "1m0.177s",
      "usage": {
        "cpu": "78528126n",
        "memory": "2413824Ki"
      }
    }
  ]
}
```

则使用**kubectl top node**查看节点内存使用率：

节点内存使用率 = 节点已使用量 / 节点可分配 = 2413824Ki / 2213604Ki = 109%

实际节点内存使用率：

实际节点内存利用率 = 节点已使用量 / 节点内存总量 = 2413824Ki / 4030180Ki = 59.9%

4.3.17 CCE 节点事件中一直出现“镜像回收失败”告警如何解决？

问题现象

节点事件中，重复出现“镜像回收失败”告警，告警示例如下：

```
wanted to free xx bytes, but freed xx bytes space with errors in image deletion: rpc error: code = Unknown
desc = Error response from daemon: conflict: unable to remove repository reference "imageName:tag"
(must force) - container 966fce58d9b8 is using its referenced image 50a7aa6fa56a
```

例如上述告警中提到的容器ID(966fce58d9b8)已经停止运行了，但没有完全被删除。

问题根因

kubelet根据imageGCHighThresholdPercent和imageGCLowThresholdPercent两个配置参数定期回收未在使用中的镜像。如果在节点上使用docker或crictl命令行启动容器，那么在容器停止后，它将处于退出状态，但并未完全删除，这意味着该容器仍然引用着容器镜像。由于kubelet无法感知到非Pod产生的容器，也就无法感知到该容器镜像被引用，因此当kubelet尝试删除容器镜像时，容器运行时会因为容器镜像仍处于被引用的状态而拦截kubelet的删除行为，导致kubelet在定期回收镜像过程中出现失败。

解决方法

登录节点上执行以下命令，过滤出告警提示的容器，确认是否处于exited状态。其中{containerId}需要替换为告警中提到的容器ID。

- 节点使用docker运行时：
`docker ps -a | grep {containerId}`
- 节点使用containerd运行时：
`crictl ps -a | grep {containerId}`

如果您确认容器已不再使用，请执行以下命令，彻底清除处于exited状态的容器。其中{containerId}需要替换为告警中提到的容器ID。

- 节点使用docker运行时：
`docker rm {containerId}`
- 节点使用containerd运行时：
`crictl rm {containerId}`

删除问题容器后，kubelet下次回收镜像可以正常进行。

4.3.18 节点磁盘利用率高的问题排查

问题现象

节点磁盘利用率高，执行df -h命令，输出中存在使用率超过90%的场景：

```
f215b6a889e39e38/mounts/shm overlay 561G 479G 56G 90% /var/lib/docker/overlay2/16fa6f47f3d7102f2632d061b8d1ecc1726a087c85697e
4c8be2565e553d/merged shm 64M 0 64M 0% /var/lib/docker/containers/313f3a1eec9172548f32ac93922fd1ba030716dd783b
b8f5e2665bbe201b/mounts/shm overlay 561G 479G 56G 90% /var/lib/docker/overlay2/6ceec438739e275e9b96eccf7c5e4fd7f6629d0251ba5c
8ffca21562a130/merged overlay 561G 479G 56G 90% /var/lib/docker/overlay2/14907a11830afad5e55f7acd439b2c05458ac47565b9a5
b94acbc28a7007/merged overlay 561G 479G 56G 90% /var/lib/docker/overlay2/3a3050496a556d1770bbc098d8f24581075835a820c6c5
35327e532298d9/merged overlay 561G 479G 56G 90% /var/lib/docker/overlay2/c3e5e10945d542c6b7c8ccdca3c27767c4ee74a34c707
```

问题根因

以下原因均可能导致节点磁盘利用率偏高：

- 场景一：磁盘上存在大文件。
- 场景二：由于容器挂载hostPath或者emptyDir的，大量容器数据写入主机导致。
- 场景三：容器自身log日志打印，日志未设置保存天数，导致日志过大。

解决方法

- 场景一：排查是否存在大文件

使用find命令来查找大文件。例如，要查找所有大于1GB的文件，可以使用以下命令：

```
find / -type f -size +1G
```

这个命令从根目录/开始查找所有大小超过 1GB 的文件（包括隐藏文件）。

示例如下：

- 查找当前文件夹下超过10k的文件

```
]# find . -type f -size +10k
./log/.test-3.log
./log/test-2.log
```

- 查找根目录下的超过100M的文件

```
]# find / -type f -size +100M
/var/log/paas/kubernetes/ingress-controller.log
/var/log/paas/kubernetes/kube-apiserver.log
```

- 场景二：查询容器挂载路径

查询Pod挂载方式

```
kubectl describe pods | grep -Pz Volumes
```

输出示例如下：

```
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-pp5rs (
Volumes:
vol-174589709548581872:
  Type:          HostPath (bare host directory volume)
  Path:          /root
  HostPathType:
kube-api-access-pp5rs:
```

如果容器挂载hostPath或者emptyDir，则需进一步排查挂载路径文件夹里的文件大小。

- 场景三：查询日志存储配置

- docker:

执行以下命令：

```
cat /etc/docker/daemon.json
```

输出示例：

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

- containerd:

查看kubelet配置文件：

```
cat /opt/cloud/cce/kubernetes/kubelet/kubelet_config.yaml
```

输出示例：

```
containerLogMaxSize: 50Mi
containerLogMaxFiles: 20
```

使用建议

1. 清理无用的文件。

参考命令如下：

- 列出当前目录下的所有文件

```
ls -la
```

- 删除无用的文件

```
rm -f <file_name>
```

2. 对于需要写入大量数据的Pod，建议使用云存储。详情请参见[存储概述](#)。
3. docker、containerd的日志文件配置maxSize和maxFile。



注意

更新配置需要重启docker服务或kubelet服务，请谨慎操作。

containerd容器引擎的节点池

- 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”。
- 在侧边栏滑出的“配置管理”窗口中，在“kubelet组件配置”中找到容器的日志文件个数上限（**container-log-max-files**）、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**）参数配置。
- 在参数输入框中输入新的数值。

配置管理（testcx节点池） ×

是否让kubelet每次仅拉取一个镜像（serialize-image-pulls）
☐

每秒钟可以执行的镜像仓库拉取操作限值（registry-pull-qps）
- 5 +

突发性镜像拉取的上限值（registry-burst）
- 10 +

容器的日志文件个数上限（container-log-max-files） ⓘ
- 20 +

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） ⓘ
- 50 + Mi

镜像垃圾回收上限百分比（image-gc-high-threshold） ⓘ
- 80 + %

镜像垃圾回收下限百分比（image-gc-low-threshold） ⓘ
- 70 + %

- 保存配置。

docker容器引擎的节点池

- 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”。
- 在侧边栏滑出的“配置管理”窗口中，在“容器引擎Docker配置”中找到容器日志文件轮换生成新文件的最大值（**max-size**）、容器的日志文件个数上限（**max-file**）参数配置。
- 在参数输入框中输入新的数值。

配置管理 (000581011-118-nadepool-55811节点池)

Native Umask (native-umask)
normal

Base Size (docker-base-size)
10G

不安全的镜像源地址 (insecure-registry)
false

容器core文件的大小限制 (limitcore)
5368709120

容器内句柄数限制 (default-ulimit-nofile)
{soft} {hard}

镜像拉取超时时间 (image-pull-progress-timeout)
1 分 0 秒

单次拉取镜像层的最大并发数 (max-concurrent-downloads) ⓘ
- 3 +

容器日志文件轮换生成新文件的最大值 (max-size) ⓘ
- 50 + m

容器的日志文件个数上限 (max-file) ⓘ
- 20 +

- f. 保存配置。
4. 定期清理节点上没有使用的镜像。
- 使用containerd容器引擎的节点：
 - i. 查看节点上的本地镜像。
cricctl images -v
 - ii. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
cricctl rmi {镜像ID}
 - 使用docker容器引擎的节点：
 - i. 查看节点上的本地镜像。
docker images
 - ii. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
docker rmi {镜像ID}

4.3.19 GPU 节点显示 GPU 卡不可用时，如何解决？

问题现象

GPU节点显示GPU卡不可用，且可能原因如下：

- GPU插件未就绪或状态异常
- 节点驱动未就绪
- GPU卡异常

图 4-8 GPU 卡不可用



解决方法

针对上述问题，建议首先检查驱动程序是否存在问题，随后排查CCE AI套件（NVIDIA GPU）插件的device-plugin组件问题，最后再排查GPU设备问题。具体步骤如下：

驱动问题排查

步骤1 检查nvidia-driver-installer Pod实例的状态。

登录CCE控制台，单击对应集群名称进入集群概览页。在左侧导航栏中，单击“节点管理”，右侧切换至“节点”页签。在异常节点的操作列单击“更多 > 实例列表”，检查异常节点中是否存在运行中的nvidia-driver-installer Pod实例。nvidia-driver-installer Pod实例可能情况如下：

- Pod实例存在且为运行中，则为正常，请跳转至步骤2进一步排查驱动是否安装成功。
- Pod实例存在但长时间未切换至运行中，则需要检查该Pod的事件是否有异常报错，根据报错信息进行处理。如果无法解决，请提交工单联系技术支持人员。

不同操作系统中，nvidia-driver-installer Pod实例的名称有所差异，具体如下：

表 4-6 nvidia-driver-installer Pod 名称

操作系统	nvidia-driver-installer Pod实例名称
Huawei Cloud EulerOS 2.0	hce20-nvidia-driver-installer
Ubuntu	ubuntu22-nvidia-driver-installer
其他	nvidia-driver-installer

步骤2 检查GPU驱动是否安装成功。

- 在节点列表中，单击异常节点名称，“前往弹性云服务器ECS详情”弹窗中单击“确定”。在异常节点页面中，右上角单击“远程登录”。
- 检查驱动安装目录。
 - 执行以下命令，进入驱动安装目录，确定驱动安装目录是否存在。若驱动安装目录不存在，则请跳过该步骤，进入步骤3排查驱动安装过程。

```
cd <驱动安装目录>
```

驱动安装目录与CCE AI套件（NVIDIA GPU）插件版本有关，具体如下：

- CCE AI套件（NVIDIA GPU）插件版本大于2.0.0时，驱动安装目录为/usr/local/nvidia。
 - CCE AI套件（NVIDIA GPU）插件版本小于2.0.0时，驱动安装目录为/opt/cloud/cce/nvidia。
- b. 在驱动安装目录中，执行以下命令，查看目录下的全部文件。

```
ls -l
```

正常文件目录如下，其中nvidia.run为驱动文件，nvidia-installer.log是NVIDIA驱动自身输出的安装日志，nvidia-uninstall.log（目录中不一定存在）为NVIDIA驱动自身输出的卸载日志。除nvidia-uninstall.log外，如果文件不全，请跳转至步骤3进一步排查驱动安装过程。

```
total 367012
drwxr-xr-x 2 root root      4096 Jun  5 19:34 bin
drwxr-xr-x 2 root root      4096 Jun  5 19:34 bin-mount
drwxr-xr-x 3 root root      4096 Jun  5 19:33 bin-workdir
drwxr-xr-x 2 root root      4096 Jun  5 19:34 drivers
drwxr-xr-x 3 root root      4096 Jun  5 19:33 drivers-workdir
drwxr-xr-x 4 root root      4096 Jun  5 19:34 lib64
drwxr-xr-x 3 root root      4096 Jun  5 19:33 lib64-workdir
-rw-r----- 1 root root 46236 Jun  5 19:34 nvidia-installer.log
-rw-r----- 1 root root 375725620 Jun  5 19:33 nvidia.run
-rw-r----- 1 root root    887 Jun  5 19:34 nvidia-uninstall.log
drwxr-xr-x 4 root root      4096 Jun  5 19:05 share
drwxr-xr-x 2 root root      4096 Jun  6 17:08 xgpu-kmod
```

- c. 执行以下命令，进入nvidia的bin目录，检查nvidia-smi是否正常。当插件版本小于2.0.0时，路径请替换为/opt/cloud/cce/nvidia/bin。

```
cd /usr/local/nvidia/bin
./nvidia-smi
```

若未显示以下信息，请跳转至步骤3进一步排查驱动安装过程。

```
Mon Jun  9 11:28:13 2025
+-----+
| NVIDIA-SMI 570.124.06                Driver Version: 570.124.06      CUDA Version: 12.8     |
+-----+-----+-----+-----+-----+-----+
| 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   42C    P0               32W / 165W |  1MiB / 24576MiB |      0%      Default |
|                               |                      | Disabled |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               |
|  GPU   GI    CI          PID    Type   Process name                  GPU Memory |
|          ID    ID              |              |           Usage            |
+-----+-----+-----+-----+-----+
| No running processes found              |
+-----+
```

步骤3 检查节点驱动安装日志，确认驱动安装过程是否存在问题。

执行以下命令，查看nvidia-driver-installer容器的日志。当插件版本小于2.0.0时，路径请替换为/opt/cloud/cce/nvidia/nvidia-installer.log。


```
cat /usr/local/nvidia/nvidia-installer.log
```

回显结果中存在如下信息，则说明安装过程正常，否则说明存在异常。如果异常，请[提交工单](#)联系技术支持人员。

```
...  
> Installation of the NVIDIA Accelerated Graphics Driver for xxx (version: x.x.x) is now complete.
```

----结束

device-plugin 组件问题排查

device-plugin组件是CCE集群中用于报告硬件资源状态的组件。在GPU场景中，节点可用的GPU资源由kube-system命名空间中的nvidia-gpu-device-plugin上报。若节点资源上报不准确或设备挂载存在问题，建议优先排查device-plugin。

执行以下命令，**检查device-plugin组件状态**。

```
kubectl get po -A -o wide | grep nvidia
```

- 若异常节点对应的device-plugin Pod实例为**非Running状态**，请[提交工单](#)联系技术支持人员。
- 若异常节点对应的device-plugin Pod实例为**Running状态**，请执行以下命令，进一步检查device-plugin组件日志，确认是否存在报错信息。

```
kubectl logs -n kube-system nvidia-gpu-device-plugin-9xmhr
```

若回显结果显示 “**gpu driver wasn't ready. will re-check**”，请进入[步骤2](#)，检查驱动安装目录下的/usr/local/nvidia/bin/nvidia-smi或/opt/cloud/cce/nvidia/bin/nvidia-smi文件是否存在。

```
...  
I0527 11:29:06.420714 3336959 nvidia_gpu.go:76] device-plugin started  
I0527 11:29:06.521884 3336959 nodeinformer.go:124] "nodeInformer started"  
I0527 11:29:06.521964 3336959 nvidia_gpu.go:262] "gpu driver wasn't ready. will re-check in %s"  
5s="(MISSING)"  
I0527 11:29:11.524882 3336959 nvidia_gpu.go:262] "gpu driver wasn't ready. will re-check in %s"  
5s="(MISSING)"  
...
```

GPU 设备问题排查

请参考[GPU故障处理](#)进行排查。

4.3.20 CCE AI 套件（NVIDIA GPU）插件升级后，GPU 节点事件中 出现告警如何解决？

问题现象

CCE AI套件（NVIDIA GPU）插件升级后，查看GPU节点事件时，存在以下告警信息：

- **告警一：**
事件名称：XGPUKmodNeedUpgrade
K8S事件：GPU serverid: xxx, info: XGPU kmod on node xx.xx.xx.xx needs upgrade
- **告警二：**
事件名称：XGPUKmodAbnormal
K8S事件：XGPU kmod on node %s is abnormal

问题根因

- **告警一：**在CCE AI套件（NVIDIA GPU）插件升级前，未提前清空GPU节点中的GPU虚拟化负载，导致插件升级过程中跳过了xgpu kmod升级，造成xgpu kmod版本与插件版本不匹配，产生告警。
- **告警二：**CCE AI套件（NVIDIA GPU）插件升级过程中，xgpu kmod升级故障。

上述告警对现有业务无影响，但可能导致插件新特性或缺陷修复无法生效，建议及时处理以保障功能完整性。

解决方案

请依次排空GPU节点中的GPU虚拟化负载，并重启节点对应的nvidia-gpu-device-plugin组件，具体操作请参见[CCE AI套件（NVIDIA GPU）插件版本升级或回滚后，如何对GPU节点进行排水？](#)。

4.4 规格配置变更

4.4.1 如何变更 CCE 集群中的节点规格？

约束与限制

- 节点池中的节点在ECS侧变更规格后，可能导致节点池弹性伸缩问题，详情请参见[CCE节点池内的节点变更规格后会有哪些影响？](#)。
- CCE Turbo集群中的部分规格节点仅支持在CCE中创建，无法在ECS控制台变更规格，此种情况下调用ECS API变更规格，具体请参见[变更云服务器规格](#)。

操作方法

注意

如果需要变更规格的节点是纳管到集群中的，可将节点从CCE集群中移除后再变更节点规格，避免影响业务。

- 步骤1** 登录CCE控制台，进入集群，在左侧选择“节点管理”，在右侧单击节点名称，跳转到弹性云服务器详情页。
- 步骤2** 在弹性云服务器详情页中，单击右上角的“关机”，关机完成后单击“更多 > 变更规格”。
- 步骤3** 在“云服务器变更规格”页面中根据业务需求选择相应的规格，单击“提交”完成节点规格的变更，返回弹性云服务器列表页，将该云服务器执行“开机”操作。
- 步骤4** 登录CCE控制台，进入集群，在节点管理列表中找到该节点，并单击操作栏中的“同步云服务器”，同步后即可看到节点规格已与弹性云服务器中变更的规格一致。

----结束

常见问题

配置了CPU管理策略绑核的节点，在变更规格后，可能会无法重新拉起或创建工作负载。如发生此种情况请参见[CCE节点变更规格后，为什么无法重新拉起或创建工作负载？](#)解决。

4.4.2 CCE 节点池内的节点变更规格后会有哪些影响？

问题背景

在ECS侧变更CCE节点池内节点的规格，前往CCE控制台同步云服务器状态，导致节点规格与节点池中设置的规格不一致。

问题影响

节点变更规格后，由于CPU、内存、网卡配额（可用IP地址）等节点参数发生变化，可能会导致该节点所在的节点池弹性伸缩行为与预期不符。

例如，节点进行规格变更后，增加CPU和内存（从2U4G变更4U8G）。

- 节点池扩容时，将根据节点池的节点模板信息计算资源，而ECS侧变更规格导致节点的规格与节点池设定的规格不一致，导致当前集群的CPU和内存使用量计算存在偏差，使扩容时节点池的资源总数可以部分超出CPU/内存的扩容上限。
- 节点池缩容时，如果缩容已变更规格的节点，将导致实际缩容的CPU/内存数（4U8G）大于预期缩容的CPU/内存数（2U4G），使得被缩容的CPU/内存资源过多。

解决方案

不建议您变更节点池中节点的规格，您可以使用更新节点池功能为节点池添加其他规格的节点，然后等待业务调度至新节点后，将原节点缩容。

- 步骤1** 登录CCE控制台，进入集群，在左侧选择“节点管理”。
- 步骤2** 找到目标节点池，单击“更新”。
- 步骤3** 在“节点规格”中勾选新的规格，单击“下一步 规格确认”，然后提交更新。
- 步骤4** 节点池配置更新后，单击节点池名称后的“扩缩容”。
- 步骤5** 在弹出的“节点池扩缩容”窗口中，选择需要扩容的节点规格并增加扩容节点数，然后单击“确定”。
- 步骤6** 切换至“节点”页签，找到目标节点单击“更多 > 节点排水”，安全驱逐节点上的业务Pod。
- 步骤7** 等业务Pod调度到新节点后，单击节点池名称后的“扩缩容”，选择需要缩容的节点规格并设置缩容节点数，然后单击“确定”。

----结束

4.4.3 CCE 节点变更规格后，为什么无法重新拉起或创建工作负载？

问题背景

kubelet启动参数中默认将CPU Manager的策略设置为static，允许为节点上具有某些资源特征的pod赋予增强的CPU亲和性和独占性。用户如果直接在ECS控制台对CCE节

点变更规格，会由于变更前后CPU信息不匹配，导致节点上的负载无法重新拉起，也无法创建新负载。

更多信息请参见[Kubernetes控制节点上的CPU管理策略](#)。

影响范围

开启了CPU管理策略的集群。

解决方案

步骤1 登录CCE节点（弹性云服务器）并删除cpu_manager_state文件。

删除命令示例如下：

```
rm -rf /mnt/paas/kubernetes/kubelet/cpu_manager_state
```

步骤2 重启节点或重启kubelet，重启kubelet的方法如下：

```
systemctl restart kubelet
```

步骤3 此时重新拉起或创建工作负载，已可成功执行。

----结束

4.4.4 CCE 集群的节点可以更改 IP 吗？

- 不支持修改私网IP：当前CCE的集群中把节点私网IP作为kubernetes node名称，kubernetes node名称不支持修改，修改后会导致节点不可用及容器网络异常等故障，所以不支持更换节点私网IP。
- 支持修改公网IP：节点上的公网IP可以在ECS控制台更换。

修改节点私网 IP 后如何恢复

节点私网IP修改后，会导致节点不可用。这时您需要将节点的私网IP修改回原来使用的IP。

步骤1 在CCE控制台，查看节点详情，找到该节点之前使用的IP和子网。

图 4-9 节点私网 IP 地址和所在子网

节点 节点池

标签与污点管理 更多

节点名称

节点名称	状态	所属...	节点配置	IP地址	容器组	CPU	内存	运行时版本
					(已分配/...	申请/限制	申请/限制	OS版本
☐ example2-75620...	运行中 可调度	DefaultP...	可用区3 c7.xlarge.2 4vCPUs 8GiB	10.1.0.55 (私有)	6 / 110	39.54% 75.26%	38.8% 74.81%	docker://18.9.0 EulerOS 2.0 (S...

步骤2 登录ECS控制台，找到节点，先关机，然后进入节点详情页，在弹性网卡页签修改私网IP。注意此时要选择对应的子网。

图 4-10 修改私有 IP



图 4-11 修改私有 IP

修改私有IP

云服务器	turbo-43970-ngo9l
虚拟私有云	vpc-cce
当前私有IP地址	10.100.0.99
★ 子网	subnet-node(10.100.0.0/24) 查看已有子网
私有IP地址	10.100.0.5 查看已使用IP地址
确定 取消	

步骤3 修改完成后再次开机。

----结束

4.5 操作系统问题说明

4.5.1 低版本内核的 CentOS 节点反复创删应用时偶现 cgroup kmem 泄露问题

故障现象

CentOS 7.6节点内核低于3.10.0-1062.12.1.el7.x86_64的场景下（主要为1.17.9版本集群），反复创建应用时出现cgroup kmem泄露，导致节点内存有空余，但是无法创建新的Pod，并提示报错Cannot allocate memory。

问题根因

在反复创建应用时会创建的临时memory cgroup，但在应用删除时，内核已经删除了cgroup（/sys/fs/cgroup/memory下对应的cgroup目录已经删除），但在内核中没有释放cssid，导致内核认为的cgroup的数量实际数量不一致，残留的cgroup达到节点上限后，导致该节点无法继续新建Pod。

解决方法

- 该问题可以通过可以在内核层全局使用 “cgroup.memory=nokmem” 参数关闭 kmem使用防止发生泄漏。
- 1.17集群版本已停止维护，修复该问题建议升级至1.19及以上集群版本，并通过节点重置为最新版本的操作系统修复该问题，确保内核版本高于 3.10.0-1062.12.1.el7.x86_64。

4.5.2 CCE 集群 IPVS 转发模式下 conn_reuse_mode 问题说明

问题说明

对于节点内核版本小于5.9的场景，CCE集群在IPVS模式下，通过Service方式访问集群内部服务，偶现1秒延时或者后端业务升级后访问Service失败的情况，引起该问题的主要原因为社区IPVS连接复用Bug。

IPVS 连接复用参数说明

IPVS对端口的复用策略主要由内核参数net.ipv4.vs.conn_reuse_mode决定。

- 当net.ipv4.vs.conn_reuse_mode=0时，IPVS不会对新连接进行重新负载，而是复用之前的负载结果，将新连接转发到原来的RS（IPVS的后端）上。
- 当net.ipv4.vs.conn_reuse_mode=1时，IPVS则会对新连接进行重新负载。

IPVS 连接复用参数带来的问题

• 问题1

当net.ipv4.vs.conn_reuse_mode=0时，对于端口复用的连接，IPVS不会主动进行新的调度，也并不会触发结束连接或DROP操作，新连接的数据包会被直接转发到之前使用的后端pod。如果此时后端pod已经被删除或重建就会出现异常，根据当前的实现逻辑，高并发访问Service场景下，不断有端口复用的连接请求发来，旧的转发连接不会被kube-proxy删除，导致访问Service失败。

• 问题2

当net.ipv4.vs.conn_reuse_mode=1时，高并发场景下发生源端口与之前链接重复的情况，不会复用链接，而是会重新进行调度。根据ip_vs_in()的处理逻辑，当开启了net.ipv4.vs.conntrack时，这种情况会先DROP掉第一个SYN包，导致SYN的重传，有1秒延迟。导致性能下降。

社区当前行为及在 CCE 集群中影响

节点上net.ipv4.vs.conn_reuse_mode的初始默认值为1，但社区kube-proxy会对该参数进行重新设置：

集群版本	kube-proxy行为	对CCE集群的影响
1.17及以下	kube-proxy默认将net.ipv4.vs.conn_reuse_mode设置为0，详情请参见 fix IPVS low throughput issue 。	如果CCE 1.17及以下版本的集群使用IPVS作为服务转发模式，所有节点net.ipv4.vs.conn_reuse_mode参数都会被kube-proxy默认设置为0，因此将存在 问题1 ，即端口复用下的RS无法移除问题。

集群版本	kube-proxy行为	对CCE集群的影响
1.19及以上	kube-proxy会根据内核的版本进行选择，详情请参见ipvs: only attempt setting of sysctlconnreuse on supported kernels。 - 内核版本大于4.1：将net.ipv4.vs.conn_reuse_mode设置为0。 - 其他情况：保持系统原有的默认值（即net.ipv4.vs.conn_reuse_mode=1）。 说明 由于Linux 5.9内核已修复该问题，从Kubernetes 1.22版本开始，对于5.9及以上的内核，kube-proxy不再修改net.ipv4.vs.conn_reuse_mode参数，详情请参见Don't set sysctl net.ipv4.vs.conn_reuse_mode for kernels >=5.9。	在CCE 1.19.16-r0及以上版本集群中，使用IPVS作为服务转发模式的情况下，由于节点内核版本不同，不同操作系统情况如下： - 当节点的OS版本为EulerOS 2.5和CentOS 7.6时， - 当集群版本在v1.19.8-r0到v1.19.16-r60（不包括v1.19.16-r60）、v1.21.1-r0到v1.21.11-r20（不包括v1.21.11-r20）、v1.23.1-r0到v1.23.10-r0（不包括v1.23.10-r0）、v1.25.1-r0到v1.25.5-r0（不包括v1.25.5-r0）、v1.27.1-r0到v1.27.2-r0（不包括v1.27.2-r0）的范围内时，OS内核版本低于4.1，因此kube-proxy会保持系统原有的默认值net.ipv4.vs.conn_reuse_mode=1，将存在问题2，即高并发场景存在1秒延时。 - 其他版本集群：设置net.ipv4.vs.conn_reuse_mode=0，存在问题1，即端口复用下的RS无法移除问题。 - 当节点的OS版本为Ubuntu 18.04时，内核版本高于4.1，因此kube-proxy会设置net.ipv4.vs.conn_reuse_mode=0，存在问题1，即端口复用下的RS无法移除问题。 - 当节点的OS版本为EulerOS 2.9时，低版本内核情况下kube-proxy设置net.ipv4.vs.conn_reuse_mode=0会引发问题1，将内核版本升级至指定版本即可修复该问题，参见修复计划。 - 当节点的OS版本为Huawei Cloud EulerOS 2.0、Ubuntu 22.04时，内核版本大于5.9，操作系统已修复该问题。

建议

请您对该问题的影响进行评估，如果上述问题会对您的业务产生影响建议您采取以下措施：

- 使用不涉及该问题的操作系统，如Huawei Cloud EulerOS 2.0、Ubuntu 22.04。对于EulerOS 2.9的节点，新建节点不涉及该问题，存量低版内核的节点需要升级至已修复版本，详情请参见[修复计划](#)。
- 使用服务转发模式为iptables的集群。

修复计划

如果您使用EulerOS 2.9的节点，请确认节点内核版本是否满足以下要求。如节点内核版本过低，可以选择重置节点或者重新创建节点解决该问题。

已修复的内核版本如下：

- x86: 4.18.0-147.5.1.6.h686.eulerosv2r9.x86_64
- ARM: 4.19.90-vhulk2103.1.0.h584.eulerosv2r9.aarch64

K8s社区issue: <https://github.com/kubernetes/kubernetes/issues/81775>

4.5.3 cgroup 统计资源异常导致 kubelet 驱逐 Pod

故障现象

ARM架构节点上，cgroup统计资源异常导致kubelet驱逐Pod，节点无法正常使用。

kubelet一直在驱逐pod，把容器全终止之后还是认为内存不足。

```
0000 CSI: m=41145882.1730751 (duration: 4.000000000s). Error: UnregisterPlugin error: dial failed at socket /mnt/paas/kubernetes/kubelet/dial socket /mnt/paas/kubernetes/kubelet/plugins_registry/disk.csi.everest.io-reg.sock, err: context deadline exceeded"
10621 14:33:26.820449 5176 setters.go:74] Using node IP: "192.168.160.181"
10621 14:33:27.866390 5176 eviction_manager.go:395] eviction manager: attempting to reclaim memory
10621 14:33:27.866453 5176 eviction_manager.go:406] eviction manager: must evict pod(s) to reclaim memory
10621 14:33:27.866466 5176 eviction_manager.go:417] eviction manager: eviction thresholds have been met, but no pods are active to evict
10621 14:33:36.826267 5176 setters.go:74] Using node IP: "192.168.160.181"
10621 14:33:37.953876 5176 eviction_manager.go:395] eviction manager: attempting to reclaim memory
10621 14:33:37.953941 5176 eviction_manager.go:406] eviction manager: must evict pod(s) to reclaim memory
10621 14:33:37.953954 5176 eviction_manager.go:417] eviction manager: eviction thresholds have been met, but no pods are active to evict
10621 14:33:46.830638 5176 setters.go:74] Using node IP: "192.168.160.181"
10621 14:33:48.041573 5176 eviction_manager.go:395] eviction manager: attempting to reclaim memory
10621 14:33:48.041639 5176 eviction_manager.go:406] eviction manager: must evict pod(s) to reclaim memory
10621 14:33:48.041654 5176 eviction_manager.go:417] eviction manager: eviction thresholds have been met, but no pods are active to evict
10621 14:33:56.842191 5176 setters.go:74] Using node IP: "192.168.160.181"
10621 14:33:58.129728 5176 eviction_manager.go:395] eviction manager: attempting to reclaim memory
10621 14:33:58.129794 5176 eviction_manager.go:406] eviction manager: must evict pod(s) to reclaim memory
10621 14:33:58.129809 5176 eviction_manager.go:417] eviction manager: eviction thresholds have been met, but no pods are active to evict
10621 14:34:06.846538 5176 setters.go:74] Using node IP: "192.168.160.181"
10621 14:34:08.217755 5176 eviction_manager.go:395] eviction manager: attempting to reclaim memory
10621 14:34:08.217832 5176 eviction_manager.go:406] eviction manager: must evict pod(s) to reclaim memory
10621 14:34:08.217845 5176 eviction_manager.go:417] eviction manager: eviction thresholds have been met, but no pods are active to evict
```

此时实际资源使用正常。

```
top - 14:34:46 up 135 days, 22:42, 2 users, load average: 0.09, 0.17, 0.17
Tasks: 222 total, 1 running, 221 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 1.4 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 22.9/15509.0 [|||||||||||||||||]
MiB Swap: 0.0/0.0 [
```

查看/sys/fs/cgroup/memory目录下cgroup的usage_in_bytes统计值有问题，与实际不符。

```
# cd /sys/fs/cgroup/memory
# cat memory.usage_in_bytes
17618837504
```

问题根因

ARM架构节点的EulerOS 2.8和EulerOS 2.9操作系统内核存在Bug，会触发kubelet驱逐Pod导致业务不可用。

📖 说明

该问题在以下版本中已被修复：

- EulerOS 2.8: 内核版本kernel-4.19.36-vhulk1907.1.0.h1252.eulerosv2r8.aarch64
- EulerOS 2.9: 内核版本kernel-4.19.90-vhulk2103.1.0.h819.eulerosv2r9.aarch64

解决方法

- 若您的集群版本为1.19.16-r0、1.21.7-r0、1.23.5-r0、1.25.1-r0及以上，请将节点重置为最新版本的操作系统即可修复该问题。
- 若您的集群版本不满足要求，请将集群升级到上述指定的版本后，再将节点重置为最新版本的操作系统。

4.5.4 低版本内核的 CentOS 节点出现容器 OOM 时，偶现 ext4 文件系统卡死问题

故障现象

CentOS 7.6节点内核低于3.10.0-1160.66.1.el7.x86_64的场景下，节点上容器出现OOM后，可能遇到节点上所有容器无法访问，docker、jdb等相关进程处于D状态，节点重启后恢复。

```
[<ffffffff99986832>] ? mutex_lock+0x12/0x2f
[<ffffffff9944d243>] do_sync_write+0x93/0xc8
[<ffffffff9944dd38>] vfs_write+0xc8/0x1f8
[<ffffffff9944eb8f>] Sys_write+0x7f/0xf8
[<ffffffff99994f92>] system call fastpath+0x25/0x2a
INFO: task dockerd:4393 blocked for more than 120 seconds.
"echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this
dockerd      D ffff8b4c      0  4393      1 0x00000000
Call Trace:
[<ffffffff99987dd9>] schedule+0x29/0x78
[<ffffffffc8350885>] wait_transaction_locked+0x85/0xd8 [jbd2]
[<ffffffff992c6f80>] ? wake_up_atomic_t+0x38/0x38
[<ffffffffc8350378>] add_transaction_credits+0x278/0x318 [jbd2]
```

问题根因

业务容器内存使用超过容器的内存限制量时，触发cgroup OOM，被系统内核终止。容器cgroup OOM在CentOS 7会偶现触发ext4文件系统卡死，ext4/jbd2会因为死锁而永远挂起。在文件系统上执行I/O的所有任务都将受到影响。

解决方法

- 临时解决方案：该问题触发后可以通过重启节点临时恢复。
- 长久解决方案：
 - 若您的集群版本为1.19.16-r0、1.21.7-r0、1.23.5-r0、1.25.1-r0及以上，请将节点重置为最新版本的操作系统即可修复该问题。
 - 若您的集群版本不满足要求，请将集群升级到上述指定的版本后，再将节点重置为最新版本的操作系统。

4.5.5 IPVS 缺陷导致节点上升级 CoreDNS 后出现概率性解析超时

故障现象

在集群使用IPVS转发的场景下，节点上升级CoreDNS后，可能出现概率性丢包，导致域名解析失败。

问题根因

该问题由IPVS缺陷导致，社区已在IPVS v5.9-rc1版本中修复该问题，详情请参见[ipvs: queue delayed work to expire no destination connections if expire_nodest_conn=1](#)。

使用Ubuntu 22.04或Huawei Cloud EulerOS 2.0操作系统的节点上不存在此问题，CentOS/Ubuntu18.04/EulerOS 2.5/EulerOS 2.9（低版本内核）/Huawei Cloud EulerOS 1.1操作系统则存在此问题。

解决方法

- 考虑采用NodeLocal DNSCache缓存方案，可以容忍IPVS丢包，具体操作请参见[使用NodeLocal DNSCache提升DNS性能](#)。
- 使用不受影响的操作系统，如Huawei Cloud EulerOS 2.0、Ubuntu 22.04。
- 当您的节点操作系统为EulerOS 2.9时，请确认节点内核版本是否满足以下要求。如节点内核版本过低，可通过重置节点进行修复；如节点内核已满足以下要求，则不存在上述问题，无需进行修复。
 - X86节点：内核版本为4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64及以上
 - ARM节点：内核版本为4.19.90-vhulk2103.1.0.h990.eulerosv2r9.aarch64及以上

4.5.6 节点 ARP 表项超过限制

问题现象

ARP缓存超限，容器网络的访问出现异常，例如coredns域名解析概率失败。

问题根因

出现该问题的原因是节点上容器缓存的ARP表项超过限制。

问题定位

- 在节点操作系统内核为4.3及以上时，dmesg日志中会有显性的打印neighbor table overflow字样。详情请参见社区链接：[link](#)。

```
# dmesg -T
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:55 2023] neighbour: arp_cache: neighbor table overflow!
[Tue May 30 18:35:58 2023] print_fib4_table_status: 7 callbacks suppressed
[Tue May 30 18:35:59 2023] print_fib4_table_status: 23 callbacks suppressed
[Tue May 30 18:36:00 2023] print_fib4_table_status: 16 callbacks suppressed
[Tue May 30 18:36:03 2023] print_fib4_table_status: 7 callbacks suppressed
[Tue May 30 18:36:04 2023] print_fib4_table_status: 17 callbacks suppressed
[Tue May 30 18:37:38 2023] net_ratelimit: 7966 callbacks suppressed
[Tue May 30 18:37:38 2023] neighbour: arp_cache: neighbor table overflow!
```
- 在节点操作系统内核低于4.3时，不会显示打印，若有callbacks suppressed字样，则也有可能是ARP表超限导致。

```
[Wed Jun 14 21:08:58 2023] net_ratelimit: 198 callbacks suppressed
[Wed Jun 14 21:09:05 2023] net_ratelimit: 11 callbacks suppressed
[Wed Jun 14 21:12:35 2023] nr_pdlush_threads exported in /proc is scheduled for removal
[Wed Jun 14 21:39:03 2023] net_ratelimit: 337 callbacks suppressed
[Wed Jun 14 21:39:10 2023] net_ratelimit: 236 callbacks suppressed
[Wed Jun 14 22:09:18 2023] net_ratelimit: 53 callbacks suppressed
[Wed Jun 14 22:14:04 2023] net_ratelimit: 266 callbacks suppressed
[Wed Jun 14 22:14:10 2023] net_ratelimit: 350 callbacks suppressed
[Wed Jun 14 22:15:28 2023] net_ratelimit: 81 callbacks suppressed
[Wed Jun 14 22:34:12 2023] net_ratelimit: 178 callbacks suppressed
[Wed Jun 14 22:34:19 2023] net_ratelimit: 18 callbacks suppressed
[Wed Jun 14 22:39:17 2023] net_ratelimit: 95 callbacks suppressed
[Wed Jun 14 22:44:24 2023] net_ratelimit: 135 callbacks suppressed
[Wed Jun 14 22:51:43 2023] ip_finish_output2: No header cache and no neighbour!
```

解决办法

节点最大可允许的非永久表项数量由内核参数`net.ipv4.neigh.default.gc_thresh3`确定，此内核参数非namespace隔离，节点和节点上运行容器会共用ARP表项大小。容器场景下，该参数推荐设置为163790。

此内核参数计算公式如下：

- CCE Turbo集群&容器隧道网络模型集群：
$$\text{net.ipv4.neigh.default.gc\_thresh3} = \text{单节点容器个数} \times \text{容器子网可用IP数} \quad (\text{Turbo 集群多容器子网情况下，取容器子网可用IP数较大值})$$

举例：当前容器子网为192.168.0.1/20，可用地址为4096个，单节点容器部署密度最大为35个，那么`net.ipv4.neigh.default.gc_thresh3`建议配置为 $4096 \times 35 = 143360$ 个。
- VPC网络模式集群：
$$\text{net.ipv4.neigh.default.gc\_thresh3} = (\text{单节点容器个数})^2$$

举例：当前节点容器小网段掩码为25，则单节点容器可用地址为个128个，那么`net.ipv4.neigh.default.gc_thresh3`建议配置为 $128 \times 128 = 16384$ 个。

说明

以上计算公式均为最极端场景：

1. 节点上的所有容器都会主动连接容器关联子网的所有IP地址，如网关类容器需要连接集群内所有其他容器。
2. 容器关联的子网所有的IP地址都被容器使用。

步骤1 在88-k8s.conf中，将`net.ipv4.neigh.default.gc_thresh3`的值修改为163790。

```
vi /etc/sysctl.d/88-k8s.conf
```

说明

`net.ipv4.neigh.default.gc_thresh1`和`net.ipv4.neigh.default.gc_thresh2`参数禁止修改。

步骤2 重新加载配置文件。

```
sysctl -p /etc/sysctl.d/88-k8s.conf
```

步骤3 再次查看配置是否最终生效。

```
sysctl -a | grep gc_thresh3
```

```
[root@xxxxxxxxx-turbo-readinessgate-08342-r05vt ~]# sysctl -a | grep gc_thresh3
net.ipv4.neigh.default.gc_thresh3 = 163790
net.ipv6.neigh.default.gc_thresh3 = 1024
```

----结束

4.5.7 EulerOS 2.9 内核缺陷导致虚拟机卡住

故障现象

EulerOS 2.9节点上，由于内核存在调度相关的社区问题，有低概率会触发死锁，表现为虚拟机卡住。

影响范围

- x86内核版本：4.18.0-147.5.1.6.h1152.eulerosv2r9.x86_64
- arm内核版本：4.19.90-vhulk2103.1.0.h1144.eulerosv2r9.aarch64

问题根因

EulerOS 4.18版本内核存在调度相关问题，使用CPU cgroup场景下，设置cfs bandwidth，并触发CPU带宽管控，会概率性触发warn级别告警打印，该流程会持有调度的rq锁，跟其他进程发生死锁（x86_64下为ABBA锁，aarch64下为AA锁）。

解决方法

您可以修改配置文件中的kernel.printk参数值进行修复。kernel.printk参数用于控制内核日志信息的输出级别和方式。

步骤1 检查配置文件中kernel.printk参数的当前配置。

```
grep "kernel.printk" /etc/sysctl.conf
```

回显中kernel.printk参数值为“7 4 1 7”，如下：

```
[root@ ~]# grep "kernel.printk" /etc/sysctl.conf
kernel.printk=7 4 1 7
```

步骤2 删除kernel.printk配置。

```
sed -i '/^kernel.printk/d' /etc/sysctl.conf
```

步骤3 确认配置文件是否修改成功，执行以下命令无回显。

```
grep "kernel.printk" /etc/sysctl.conf
```

步骤4 重新配置kernel.printk参数。

x86_64版本：

1. 执行以下命令：

```
sysctl -w kernel.printk="4 4 1 7"
```

```
[root@localhost ~]# sysctl -w kernel.printk="4 4 1 7"
kernel.printk = 4 4 1 7
```

2. 检查修改是否成功，执行以下命令：

```
sysctl -a | grep kernel.printk
```

确认kernel.printk参数为“4 4 1 7”。

```
[root@localhost ~]# sysctl -a | grep kernel.printk
kernel.printk = 4 4 1 7
```

arm版本：

1. 执行以下命令：

```
sysctl -w kernel.printk="1 4 1 7"
```

```
[root@~]# sysctl -w kernel.printk="1 4 1 7"
kernel.printk = 1 4 1 7
```

2. 检查修改是否成功，执行以下命令：

```
sysctl -a | grep kernel.printk
```

确认kernel.printk参数为“1 4 1 7”。

```
[root@~]# sysctl -a | grep kernel.printk
kernel.printk = 1 4 1 7
```

----结束

5 节点池

5.1 节点池异常状态排查

排查思路

请根据具体节点池异常状态确定具体问题原因，如[表5-1](#)所示。

表 5-1 节点池异常

节点池异常状态	说明	解决方案
错误 Error	节点池删除失败	重试删除节点池操作，如果节点池仍旧无法删除，请提交工单帮助删除错误节点池。
配额不足 QuotaInsufficient	用户配额不足导致节点池无法扩容	请提交工单申请扩大账号配额。
资源售罄 SoldOut	底层资源不足	更新节点池配置，选择其他可用资源。
配置异常 ConfigurationInvalid	云服务器组不存在（ServerGroupNotExist）： 节点池绑定的云服务器组不存在，可能由于用户手动删除了云服务器组导致。	1. 请登录CCE控制台，在左侧导航栏中单击“节点管理”，并单击节点池名称，在“总览”页签中单击“展开高级配置”，查看所属云服务器组。 2. 登录ECS控制台，在左侧导航栏中单击“弹性云服务器 > 云服务器组”，确认云服务器组是否存在。 3. 如果云服务器组已经不存在，请登录CCE控制台，在左侧导航栏中单击“节点管理”，找到目标节点池单击“更新”，在“高级配置”中解绑云服务器组或者切换云服务器组。

节点池异常状态	说明	解决方案
云服务器组配额不足 InstanceAboveServerGroup	云服务器超过云服务器组上限	1. 请登录CCE控制台，在左侧导航栏中单击“节点管理”，并单击节点池名称，在“总览”页签中单击“展开高级配置”，查看所属云服务器组。 2. 登录ECS控制台，在左侧导航栏中单击“弹性云服务器 > 云服务器组”，检查云服务器组配额。 3. 如果云服务器组配额不足，请登录CCE控制台，在左侧导航栏中单击“节点管理”，找到目标节点池单击“更新”，在“高级配置”中解绑云服务器组或者切换云服务器组。
容量预留不足 InvalidCapacityReservation	容量预留错误	更新节点池，选择其他容量预留规格。
节点池/伸缩组的期望节点数已达到弹性扩容最大值 MaxNodeCountReached	当前节点池或者伸缩组内期望节点数已达到当前弹性扩容最大值。	● 节点池/伸缩组内期望节点数达到当前规格弹性扩容最大值时，节点池或者伸缩组无法触发弹性扩容，节点池手动扩容则不受影响，您可以通过手动扩容添加解决。 ● 如需解决该异常，请在“节点管理>节点池”页面中单击节点池的“弹性伸缩”按钮，并修改“节点数范围”。如果关闭节点池或者伸缩组的“弹性伸缩”开关，也可以解决该异常。

5.2 节点池一直在扩容中但“操作记录”里为何没有创建节点的记录？

问题现象

节点池的状态一直处于“扩容中”，但是“操作记录”里面没有看到有对应创建节点的记录。

原因排查：

检查如下问题并修复：

- 租户是否欠费。
- 查看节点池配置的规格是否资源不足。
- 租户的ECS或内存配额是否不足。
- 如果一次创建节点太多，可能会出现租户的ECS容量校验不过的情况发生。

解决方案：

- 若租户已经欠费，请尽快续费。
- 若ECS节点资源不足，使用其他规格节点替代。
- 若ECS或内存配额不足，请扩大配额。
- 若ECS容量校验不通过，请重新校验。

5.3 节点池扩容失败

排查思路

请根据节点池扩容失败的具体事件信息确定问题原因，如表5-2所示。

表 5-2 节点池扩容失败

事件信息	问题原因	解决方案
...call fsp to query keypair fail, error code : Ecs.0314, reason is : the keypair *** does not match the user_id ***...	该问题可能由以下原因引起： • 创建节点池时使用的密钥对被删除。 • 创建节点池时使用的密钥对为私有密钥对，其他用户无法使用该密钥对创建节点。	无法获取节点池使用的密钥对
{"error": {"message": "encrypted key id [***] is invalid.", "code": "Ecs.0912"}}	该问题可能由以下原因引起： • 创建节点池输入的KMS密钥ID不存在。 • 创建节点池输入的KMS密钥ID为他人密钥，但他人未给您授权。	KMS密钥ID非法
Security group [****] not found	该问题可能存在以下两种情况： • 节点池配置了自定义安全组，但是该安全组被删除，导致节点扩容失败。 • 节点池未配置自定义安全组，且集群默认安全组被删除，导致扩容失败。	节点池指定的安全组被删除
The Enterprise Project ID [****] not exist	出现该问题的原因是节点池设定的企业项目已经在EPS服务中删除。	节点池指定的企业项目被删除

无法获取节点池使用的密钥对

当扩容节点池失败时，事件中包含Ecs.0314错误，表明无法查询到节点池使用的密钥对，导致创建云服务器失败。


```
...call fsp to query keypair fail, error code : Ecs.0314, reason is : the keypair *** does not match the user_id  
***  
...
```

该问题可能由以下原因引起：

- 原因一：创建节点池时使用的密钥对被删除。
- 原因二：用户使用私有密钥对创建节点池，而其他用户无法使用该私有密钥对创建节点，导致节点池扩容失败。

解决方案：

- 对于原因一引起的扩容失败，您可以创建一个新的密钥对，并使用该密钥对创建新的节点池。
- 对于原因二引起的扩容失败，该节点池只能通过私有密钥对的创建者进行扩容。您也可以使用其他密钥对创建一个新的节点池。

KMS 密钥 ID 非法

当扩容节点池失败时，事件中包含Ecs.0912错误：

```
{"error":{"message":"encrypted key id [***] is invalid.,"code":"Ecs.0912"}}
```

该问题可能由以下原因引起：

- 原因一：创建节点池输入的KMS密钥ID不存在。
- 原因二：创建节点池输入的KMS密钥ID为他人密钥，但他人未给您授权。

解决方案：

- 对于原因一引起的扩容失败，确保您输入密钥ID存在。
- 对于原因二引起的扩容失败，请使用已给您授权的共享密钥ID。

节点池指定的安全组被删除

当扩容节点池失败时，事件中包含创建节点失败的错误，错误信息如下：

```
Security group [****] not found
```

该问题可能存在以下两种情况：

- 情况一：节点池配置了自定义安全组，但是该安全组被删除，导致节点扩容失败。
- 情况二：节点池未配置自定义安全组，且集群默认安全组被删除，导致扩容失败。

解决方案：

- 情况一：通过更新节点池接口，更新customSecurityGroups字段中指定的安全组。详情请参见[更新指定节点池](#)。
- 情况二：您需要登录CCE控制台，在集群的“配置中心”页面修改“节点默认安全组”。新增的节点安全组需要满足集群端口通信规则，详情请参见[集群安全组规则配置](#)。

节点池指定的企业项目被删除

当扩容节点池失败时，事件中包含创建节点失败的错误，错误信息如下：

The Enterprise Project ID [*****] not exist

出现该问题的原因是节点池设定的企业项目已经在EPS服务中删除。

解决方案：

- 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 找到扩容失败的节点池，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。
 - 步骤4 重新选择一个节点池企业项目。
 - 步骤5 配置完成后，单击“确定”。
- 结束

5.4 节点池批量扩缩容节点时，Kubernetes Event 事件存在部分缺失

问题现象

节点池批量扩缩容节点时，Kubernetes Event事件存在部分缺失。
例如，集群中批量缩容10个节点，CCE打印了10次“删除节点”事件，但是Kubernetes仅打印了4次“缩容空闲节点启动”的Event事件。

2023/12/20 11:33:44 GMT+08:00		●	删除节点192.168.10.198成功[id:6c3234f5-9ee7-11e...	--
2023/12/20 11:33:44 GMT+08:00		●	删除节点192.168.10.125成功[id:6c322e6f-9ee7-11e...	--
2023/12/20 11:33:44 GMT+08:00		●	删除节点192.168.10.133成功[id:6c323761-9ee7-11e...	--
2023/12/20 11:33:44 GMT+08:00		●	删除节点192.168.10.39成功[id:6c322fc1-9ee7-11ee...	--
2023/12/20 11:33:44 GMT+08:00		●	删除节点192.168.10.201成功[id:6c32326a-9ee7-11...	--
2023/12/20 11:33:30 GMT+08:00		●	删除节点192.168.10.220成功[id:6c32295c-9ee7-11...	--
2023/12/20 11:33:30 GMT+08:00		●	删除节点192.168.10.60成功[id:6c323102-9ee7-11e...	--
2023/12/20 11:33:30 GMT+08:00		●	删除节点192.168.10.228成功[id:6c32361c-9ee7-11...	--
2023/12/20 11:33:30 GMT+08:00		●	删除节点192.168.10.76成功[id:6c322d0d-9ee7-11e...	--
2023/12/20 11:33:30 GMT+08:00		●	删除节点192.168.10.17成功[id:6c32338d-9ee7-11e...	--
2023/12/20 11:32:06 GMT+08:00		●	缩容空闲节点启动	--
2023/12/20 11:32:06 GMT+08:00		●	缩容空闲节点启动	--
2023/12/20 11:32:06 GMT+08:00		●	缩容空闲节点启动	--
2023/12/20 11:32:06 GMT+08:00		●	缩容空闲节点启动	--

问题根因

出现该问题的原因是Kubernetes在处理Event事件时，为了后端服务etcd的可用性，会对事件进行限流、聚合、计数的预处理，因此Kubernetes Event事件并非100%打印，在遇到大批量打印相同事件时，可能会出现上述问题。

以上逻辑通过Kubernetes源码中的[EventCorrelate](#)方法实现，您可以查看社区的[设计方案](#)了解详情。

该问题为Kubernetes设计机制导致，因此您可以无需关注。

5.5 云服务器无法纳管至节点池时如何修改云服务器配置

云服务器纳管至节点池时，由于以下原因导致无法纳管，您可通过修改配置进行纳管。

无法纳管原因	解决方案	操作指导
规格不一致	将云服务器规格修改成节点池中包含的规格。	修改云服务器的规格
虚拟私有云和子网不一致	将云服务器所在的虚拟私有云和子网修改成节点池相同的虚拟私有云和子网。	修改云服务器的虚拟私有云和子网
计费模式不一致	将云服务器的计费模式修改成与节点池的计费模式一致。	修改云服务器的计费模式
数据盘不一致	将云服务器的数据盘配置修改成与节点池的数据盘配置一致。	修改云服务器的数据盘
企业项目不一致	将云服务器的企业项目修改成与节点池的企业项目一致。	修改云服务器的企业项目
云服务器组不一致	将云服务器的云服务器组修改成与节点池的云服务器组一致。	修改云服务器的云服务器组

修改云服务器的规格

说明

待纳管云服务器规格需修改成节点池中包含的规格。

更多操作指导请参见[ECS变更规格通用操作](#)。

- 步骤1** 登录ECS控制台。
- 步骤2** 单击目标云服务器名称，在弹性云服务器详情页中，单击右上角的“关机”，关机完成后单击“更多 > 变更规格”。
- 步骤3** 在“云服务器变更规格”页面中根据业务需求选择相应的规格，单击“提交”完成节点规格的变更。
- 步骤4** 返回弹性云服务器列表页，将该云服务器执行“开机”操作。

----结束

修改云服务器的虚拟私有云和子网

说明

待纳管的云服务器所在VPC和子网需修改成节点池相同的VPC和子网。

更多操作指导请参见[ECS切换虚拟私有云](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器“操作”列下的“更多 > 网络/安全组 > 切换VPC”。

步骤3 设置切换VPC参数。

- 虚拟私有云：选择需要切换的VPC。
- 子网：选择需要切换的子网。
- 私有IP地址：根据需求选择自动分配或使用已有IP。

步骤4 单击“确定”，等待云服务器切换完成。

----结束

修改云服务器的计费模式

说明

待纳管云服务器的计费模式需修改成和节点池相同的计费模式。

按需转包年/包月

更多操作指导请参见[ECS按需转包年/包月](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器“操作”列下的“更多 > 转包年/包月”。

步骤3 单击“确定”，跳转至费用中心的“按需转包年/包月”页面。

步骤4 选择弹性云服务器的购买时长，判断是否勾选“自动续费”，确认预计到期时间和配置费用后单击“去支付”。

步骤5 进入支付页面，选择支付方式，确认付款，支付订单后即可完成按需转包年/包月。

----结束

包年/包月转按需

更多操作指导请参见[ECS包年/包月转按需](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器“操作”列下的“更多 > 即时转按需”。

步骤3 单击“确定”，跳转至费用中心的“包年/包月转按需”页面。

步骤4 根据界面提示，勾选需要转按需的资源。

步骤5 确认退款信息后，单击“转按需”。

步骤6 再次根据界面信息确认要转按需的资源后，单击“确定”。

----结束

修改云服务器的数据盘

说明

待纳管云服务器的数据盘数量、大小、类型需修改成和节点池的数据盘配置相同。

数据盘数量

更多操作指导请参见[ECS新增磁盘](#)或[ECS在线卸载磁盘](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器名称，进入弹性云服务器详情页。

步骤3 选择“云硬盘”页签。

- 如果待纳管节点的数据盘数量少于节点池配置中的数据盘数量，则需新增磁盘。单击“新增磁盘”，设置新添加云硬盘的参数信息。云硬盘的参数信息配置请参考[购买云硬盘](#)。

须知

配置新增磁盘时，磁盘规格及大小需和节点池中的磁盘配置保持一致，且高级配置中需勾选“SCSI”。

- 如果待纳管节点的数据盘数量多于节点池配置中的数据盘数量，则需卸载磁盘：单击待卸载磁盘所在行的“卸载”，卸载云硬盘。

----结束

数据盘大小

更多操作指导请参见[ECS扩容云硬盘](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器名称，进入弹性云服务器详情页。

步骤3 切换至“云硬盘”页签，单击待扩容云硬盘右侧的“扩容”，系统跳转至云硬盘控制台的“扩容磁盘”页面。

步骤4 根据界面提示，设置“目标容量”。

步骤5 设置完成后，单击“下一步”并根据界面提示完成订单提交。

----结束

数据盘类型

更多操作指导请参见[变更云硬盘类型](#)。

步骤1 登录ECS控制台。

步骤2 单击目标云服务器名称，进入弹性云服务器详情页。

步骤3 切换至“云硬盘”页签，单击待扩容云硬盘右侧的“磁盘变配”。

步骤4 根据界面提示，设置“磁盘类型”。

步骤5 设置完成后，单击“提交”。

----结束

修改云服务器的企业项目

说明

待纳管云服务器的企业项目需修改成和节点池相同的企业项目。

更多操作指导请参见[迁出企业项目资源](#)。

步骤1 登录华为云控制台。

步骤2 单击页面右上方的“企业 > 项目管理”。

步骤3 在企业项目管理页面，选择企业项目，单击操作列的“查看资源”。

步骤4 勾选待迁出资源，单击“迁出”。

步骤5 选择“ECS关联迁出”，其关联的资源将自动同时迁出。

步骤6 选择要迁入的企业项目，单击“确定”。

----结束

修改云服务器的云服务器组

说明

待纳管云服务器的云服务器组需修改成和节点池相同的云服务器组。

更多操作指导请参见[ECS管理云服务器组](#)。

步骤1 登录ECS控制台。

步骤2 在左侧导航树中，选择“弹性云服务器 > 云服务器组”。

步骤3 单击“操作”列下的“添加云服务器”。

步骤4 在“添加云服务器”页面，选择待添加的弹性云服务器。

步骤5 单击“确定”，将弹性云服务器加入云服务器组。

----结束

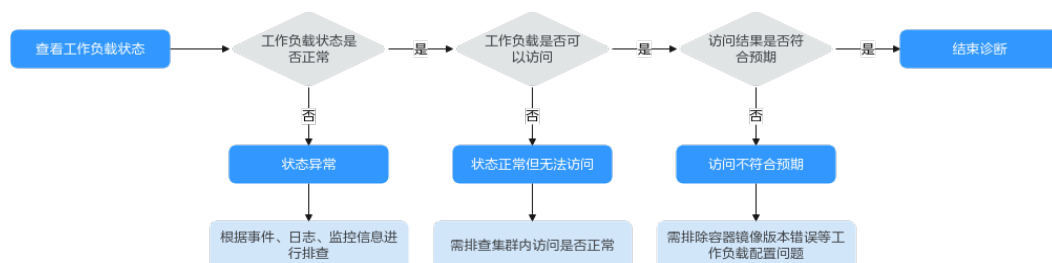
6 工作负载

6.1 工作负载异常问题排查

6.1.1 工作负载状态异常定位方法

工作负载状态异常时，建议先查看Pod的事件以便于确定导致异常的初步原因，再针对性解决问题。

定位流程



工作负载状态异常定位步骤如下：

步骤1 查看Pod状态是否正常

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 在页面左上角选择命名空间，找到对应的工作负载，查看其状态。
 - 如果工作负载状态为“未就绪”，可通过查看Pod的事件等信息确定异常原因，详情请参见[Pod事件查看方法](#)。根据事件，参考[Pod常见异常问题](#)查找异常的解决方案。
 - 如果工作负载状态为“处理中”，一般为过程中的状态，请耐心等待。
 - 如果工作负载状态为“运行中”，一般无需处理。如果出现状态正常但无法访问的情况，则需要进一步排查集群内访问是否正常。

步骤2 集群内部是否可以正常访问

您可以在CCE控制台界面或者使用kubectl命令查找Pod的IP，然后登录到集群内的节点或容器中，使用curl命令等方法手动调用接口，查看结果是否符合预期。

如果容器IP+端口不能访问，建议登录到业务容器内使用“127.0.0.1+端口”进行排查。

登录容器的操作步骤请参见[登录容器的方法](#)。

步骤3 访问结果是否符合预期

如果集群内可以正常访问工作负载，但访问结果不符合预期，则需要进一步排查工作负载配置问题，例如镜像版本、环境变量是否正确。详情请参见[工作负载状态正常但未正常工作](#)。

----结束

Pod 常见异常问题

实例状态	问题描述	处理措施
Pending	实例调度失败	请参考 工作负载异常：实例调度失败
Pending	实例挂卷失败	请参考 工作负载异常：存储卷无法挂载或挂载超时
Pending	添加存储失败	请参考 工作负载异常：添加存储失败
FailedPullImage ImagePullBackOff	拉取镜像失败 重新拉取镜像失败	请参考 工作负载异常：实例拉取镜像失败
CreateContainerError CrashLoopBackOff	启动容器失败 重新启动容器失败	请参考 工作负载异常：启动容器失败
Evicted	实例状态为“Evicted”， pod不断被驱逐	请参考 工作负载异常：实例驱逐异常（Evicted）
Creating	实例状态一直为“创建中”	请参考 工作负载异常：一直处于创建中
Terminating	实例状态一直为“结束中”	请参考 工作负载异常：Pod一直处于Terminating状态
Stopped	实例状态为“已停止”	请参考 工作负载异常：已停止
Running	实例状态为“运行中”	Pod运行正常则无需处理。如果未正常工作，请参考 工作负载状态正常但未正常工作
Init:N/M	Init容器启动失败	请参考 工作负载异常：Init容器启动失败

实例状态	问题描述	处理措施
Init:Error	Init容器启动失败	请参考 工作负载异常：Init容器启动失败
Init:CrashLoopBackOff	Init容器不断重启失败	请参考 工作负载异常：Init容器启动失败

Pod 状态检查方法

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 单击指定工作负载名称，查看工作负载中实例（Pod）的运行状态。



Pod 配置查看方法

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 单击指定工作负载名称，在工作负载实例列表中，单击操作列的“更多 > 查看 YAML”。



Pod 事件查看方法

方式一

在CCE控制台中单击工作负载名称，前往“工作负载详情”页面，找到处于异常状态的实例，单击操作栏中的“事件”进行查看。

图 6-1 查看 Pod 事件



方式二

Pod的事件可以使用kubectl命令查看：

```
kubectl describe pod {pod-name}
```

回显如下：

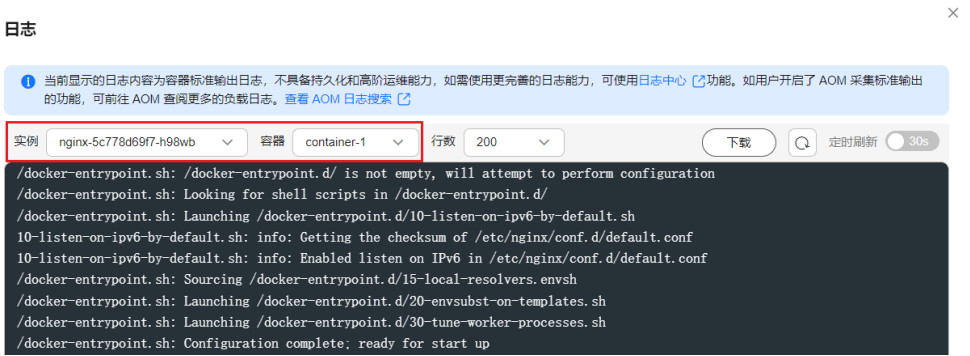
```
...
Events:
  Type     Reason            Age   From          Message
  ----     -
  Warning   FailedScheduling  49s   default-scheduler  0/2 nodes are available: 2 Insufficient cpu.
  Warning   FailedScheduling  49s   default-scheduler  0/2 nodes are available: 2 Insufficient cpu.
```

容器日志查看方法

- 1. 登录CCE控制台。
- 2. 单击集群名称进入集群，在左侧选择“工作负载”。
- 3. 单击工作负载操作列的“日志”。



- 4. 您可以在日志上方切换实例和容器。



Pod 监控检查方法

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 单击工作负载操作列的“监控”，即可查看Pod的CPU、内存、网络I/O等监控大盘。



登录容器的方法

您可以通过kubect命令登录容器，详情请参见[登录容器实例](#)。

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 单击指定工作负载名称，在工作负载实例列表中，单击操作列的“更多 > 远程登录”。



6.1.2 工作负载异常：实例调度失败

问题定位

当Pod状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。事件查看方法请参见[工作负载状态异常定位方法](#)。

排查思路

根据具体事件信息确定具体问题原因，如[表6-1](#)所示。

表 6-1 实例调度失败

事件信息	问题原因与解决方案
no nodes available to schedule pods.	集群中没有可用的节点。 排查项一：集群内是否无可用节点
0/2 nodes are available: 2 Insufficient cpu. 0/2 nodes are available: 2 Insufficient memory.	节点资源（CPU、内存）不足。 排查项二：节点资源（CPU、内存等）是否充足
0/2 nodes are available: 1 node(s) didn't match node selector, 1 node(s) didn't match pod affinity rules, 1 node(s) didn't match pod affinity/anti-affinity.	节点与Pod亲和性配置互斥，没有满足Pod要求的节点。 排查项三：检查工作负载的亲和性配置
0/2 nodes are available: 2 node(s) had volume node affinity conflict.	Pod挂载云硬盘存储卷与节点不在同一个可用区。 排查项四：挂载的存储卷与节点是否处于同一可用区
0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate.	节点存在污点Taints，而Pod不能容忍这些污点，所以不可调度。 排查项五：检查Pod污点容忍情况
0/7 nodes are available: 7 Insufficient ephemeral-storage.	节点临时存储不足。 排查项六：检查临时卷使用量
0/1 nodes are available: 1 everest driver not found at node	节点上everest-csi-driver不在running状态。 排查项七：检查everest插件是否工作正常
Failed to create pod sandbox: ... Create more free space in thin pool or use dm.min_free_space option to change behavior	节点thinpool空间不足。 排查项八：检查节点thinpool空间是否充足
0/1 nodes are available: 1 Too many pods.	该节点调度的Pod超出上限。 检查项九：检查节点上调度的Pod是否过多
UnexpectedAdmissionError Allocate failed due to not enough cpus available to satisfy request, which is unexpected.	由于已知社区问题导致kubelet静态绑核异常。 排查项十：kubelet静态绑核异常

排查项一：集群内是否无可用节点

登录CCE控制台，检查节点状态是否为可用。或使用如下命令查看节点状态是否为Ready。

```
$ kubectl get node
NAME                STATUS    ROLES    AGE   VERSION
192.168.0.37        Ready    <none>   21d   v1.19.10-r1.0.0-source-121-gb9675686c54267
192.168.0.71        Ready    <none>   21d   v1.19.10-r1.0.0-source-121-gb9675686c54267
```

如果状态都为不可用（Not Ready），则说明集群中无可用节点。

解决方案：

- 新增节点，若工作负载未设置亲和策略，pod将自动迁移至新增的可用节点，确保业务正常。
- 排查不可用节点问题并修复，排查修复方法请参见[集群可用但节点状态为“不可用”如何解决？](#)。
- 重置不可用的节点，详情请参见[重置节点](#)。

排查项二：节点资源（CPU、内存等）是否充足

0/2 nodes are available: 2 Insufficient cpu. CPU不足。

0/2 nodes are available: 2 Insufficient memory. 内存不足。

当“实例资源的申请量”超过了“实例所在节点的可分配资源总量”时，节点无法满足实例所需资源要求导致调度失败。



如果节点可分配资源小于Pod的申请量，则节点无法满足实例所需资源要求导致调度失败。

解决方案：

资源不足的情况主要解决办法是扩容，建议在集群中增加节点数量。

排查项三：检查工作负载的亲和性配置

当亲和性配置出现如下互斥情况时，也会导致实例调度失败：

例如：

workload1、workload2设置了工作负载间的反亲和，如workload1部署在Node1，workload2部署在Node2。

workload3部署上线时，既希望与workload2亲和，又希望可以部署在不同节点如Node1上，这就造成了工作负载亲和与节点亲和间的互斥，导致最终工作负载部署失败。

0/2 nodes are available: 1 node(s) didn't match **node selector**, 1 node(s) didn't match **pod affinity rules**, 1 node(s) didn't match **pod affinity/anti-affinity**.

- node selector** 表示节点亲和不满足。
- pod affinity rules** 表示Pod亲和不满足。

- **pod affinity/anti-affinity** 表示Pod亲和/反亲和不满足。

解决方案：

- 在设置“工作负载间的亲和性”和“工作负载和节点的亲和性”时，需确保不要出现互斥情况，否则工作负载会部署失败。
- 若工作负载配置了节点亲和性，需确保亲和的节点标签中supportContainer设置为true，否则会导致pod无法调动到节点上，查看事件提示如下错误信息：
No nodes are available that match all of the following predicates: MatchNode Selector, NodeNotSupportsContainer
节点标签为false时将会调度失败。

排查项四：挂载的存储卷与节点是否处于同一可用区

0/2 nodes are available: 2 node(s) had volume node affinity conflict. 存储卷与节点之间存在亲和性冲突，导致无法调度。

这是因为云硬盘不能跨可用区挂载到节点。例如云硬盘存储卷在可用区1，节点在可用区2，则会导致无法调度。

CCE中创建云硬盘存储卷，默认带有亲和性设置，如下所示：

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pvc-c29bfac7-efa3-40e6-b8d6-229d8a5372ac
spec:
  ...
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: failure-domain.beta.kubernetes.io/zone
              operator: In
              values:
                - cn-east-3a
```

解决方案：

重新创建存储卷，可用区选择与节点同一分区，或重新创建工作负载，存储卷选择自动分配。

排查项五：检查 Pod 污点容忍情况

0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate. 是因为节点打上了污点，不允许Pod调度到节点上。

查看节点的上污点的情况。如下则说明节点上存在污点。

```
$ kubectl describe node 192.168.0.37
Name:          192.168.0.37
...
Taints:        key1=value1:NoSchedule
...
```

在某些情况下，系统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。
- node.kubernetes.io/unreachable：节点控制器访问不到节点。
- 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, 此情况下您可通过[节点磁盘空间不足](#)的方案进行解决。
- node.kubernetes.io/pid-pressure: 节点的 PID 压力, 此情况下您可通过[修改节点进程 ID数量上限kernel.pid_max](#)进行解决。
- 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初始化这个节点后, kubelet将删除这个污点。

解决方案:

要想把Pod调度到这个节点上, 有两种方法:

- 若该污点为用户自行添加, 可考虑删除节点上的污点。若该污点为[系统自动添加](#), 解决相应问题后污点会自动删除。
- Pod的定义中容忍这个污点, 如下所示。详细内容请参见[污点和容忍](#)。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:alpine
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
```

排查项六: 检查临时卷使用量

0/7 nodes are available: 7 Insufficient ephemeral-storage. 节点临时存储不足。

检查Pod是否限制了临时卷的大小, 如下所示, 当应用程序需要使用的量超过节点已有容量时会导致无法调度, 修改临时卷限制或扩容节点磁盘可解决此问题。

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
  resources:
    requests:
      ephemeral-storage: "2Gi"
    limits:
      ephemeral-storage: "4Gi"
  volumeMounts:
  - name: ephemeral
    mountPath: "/tmp"
  volumes:
  - name: ephemeral
    emptyDir: {}
```

您可以通过**kubectl describe node**命令查询节点临时卷的容量 (Capacity) 和可使用量 (Allocatable), 并可查询节点已分配的临时卷申请值和限制值。

返回示例如下:

```
...
Capacity:
  cpu:          4
  ephemeral-storage: 61607776Ki
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  localssd:     0
  localvolume:  0
  memory:       7614352Ki
  pods:         40
Allocatable:
  cpu:          3920m
  ephemeral-storage: 56777726268
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  localssd:     0
  localvolume:  0
  memory:       6180752Ki
  pods:         40
...
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests          Limits
-----
cpu                 1605m (40%)      6530m (166%)
memory              2625Mi (43%)    5612Mi (92%)
ephemeral-storage  0 (0%)          0 (0%)
hugepages-1Gi      0 (0%)          0 (0%)
hugepages-2Mi      0 (0%)          0 (0%)
localssd            0                0
localvolume         0                0
Events:             <none>
```

排查项七：检查 everest 插件是否工作正常

0/1 nodes are available: 1 everest driver not found at node. 集群everest插件的everest-csi-driver 在节点上未正常启动。

检查kube-system命名空间下名为everest-csi-driver的守护进程，查看对应Pod是否正常启动，若未正常启动，删除该Pod，守护进程会重新拉起该Pod。

排查项八：检查节点 thinpool 空间是否充足

节点在创建时会绑定一个供kubelet及容器引擎使用的专用数据盘，详情请参见[数据盘空间分配说明](#)。若数据盘空间不足，将导致实例无法正常创建。

方案一：清理镜像

您可以执行以下步骤清理未使用的镜像：

- 使用containerd容器引擎的节点：
 - a. 查看节点上的本地镜像。
`crictl images -v`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`crictl rmi {镜像ID}`
- 使用docker容器引擎的节点：
 - a. 查看节点上的本地镜像。
`docker images`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`docker rmi {镜像ID}`

📖 说明

请勿删除cce-pause等系统镜像，否则可能导致无法正常创建容器。

方案二：扩容磁盘

扩容磁盘的操作步骤如下：

步骤1 在EVS控制台扩容数据盘。详情请参见[扩容云硬盘容量](#)。

在EVS控制台扩容成功后，仅扩大了云硬盘的存储容量，还需要执行后续步骤扩容逻辑卷和文件系统。

步骤2 登录CCE控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤3 登录目标节点。

步骤4 使用lsblk命令查看节点块设备信息。

这里存在两种情况，根据容器存储Rootfs而不同。

Overlayfs：没有单独划分thinpool，在dockersys空间下统一存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    0   50G  0 disk
└─sda1               8:1    0   50G  0 part /
sdb                  8:16    0  150G  0 disk # 数据盘已扩容至150G，存在50G空间仍未分配
└─vgpaas-dockersys 253:0    0   90G  0 lvm  /var/lib/containerd
   └─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

将新增的磁盘容量加到容器引擎使用的dockersys逻辑卷上。

a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/sdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/sdb
```

回显如下：

```
Physical volume "/dev/sdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <90.00 GiB (23039 extents) to 140.00
GiB (35840 extents).
Logical volume vgpaas/dockersys successfully resized.
```

c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/containerd; on-line resizing required
old_desc_blocks = 12, new_desc_blocks = 18
The filesystem on /dev/vgpaas/dockersys is now 36700160 blocks long.
```

3. 检查是否扩容成功。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    0   50G  0 disk
```

```
└─sda1      8:1  0  50G  0 part /
sdb         8:16  0 150G  0 disk
└─vgpaas-dockersys 253:0  0 140G  0 lvm /var/lib/containerd
   └─vgpaas-kubernetes 253:1  0 10G  0 lvm /mnt/paas/kubernetes/kubelet
```

Devicemapper: 单独划分了thinpool存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                8:0    0   50G  0 disk
└─vda1                             8:1    0   50G  0 part /
vdb                                8:16    0 200G  0 disk
├─vgpaas-dockersys                 253:0    0  18G  0 lvm /var/lib/docker
├─vgpaas-thinpool_tmeta            253:1    0   3G  0 lvm
├─└─vgpaas-thinpool                253:3    0  67G  0 lvm          # thinpool空间
│   ...
├─vgpaas-thinpool_tdata            253:2    0  67G  0 lvm
├─└─vgpaas-thinpool                253:3    0  67G  0 lvm
│   ...
└─vgpaas-kubernetes               253:4    0  10G  0 lvm /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

选项一：将新增的磁盘容量加到thinpool盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/vdb为thinpool空间所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/thinpool为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/thinpool
```

回显如下：

```
Size of logical volume vgpaas/thinpool changed from <67.00 GiB (23039 extents) to <167.00
GiB (48639 extents).
Logical volume vgpaas/thinpool successfully resized.
```

- c. 由于thinpool未挂载到设备，因此无需调整文件系统的大小。
- d. 检查是否扩容成功。使用lsblk命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到thinpool盘，则表示扩容成功。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                8:0    0   50G  0 disk
└─vda1                             8:1    0   50G  0 part /
vdb                                8:16    0 200G  0 disk
├─vgpaas-dockersys                 253:0    0  18G  0 lvm /var/lib/docker
├─vgpaas-thinpool_tmeta            253:1    0   3G  0 lvm
├─└─vgpaas-thinpool                253:3    0 167G  0 lvm          # 扩容后的thinpool空间
│   ...
├─vgpaas-thinpool_tdata            253:2    0  67G  0 lvm
├─└─vgpaas-thinpool                253:3    0  67G  0 lvm
│   ...
└─vgpaas-kubernetes               253:4    0  10G  0 lvm /mnt/paas/kubernetes/kubelet
```

选项二：将新增的磁盘容量加到dockersys盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/vdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <18.00 GiB (4607 extents) to <118.00 GiB (30208 extents).
Logical volume vgpaas/dockersys successfully resized.
```

- c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/docker; on-line resizing required
old_desc_blocks = 3, new_desc_blocks = 15
The filesystem on /dev/vgpaas/dockersys is now 30932992 blocks long.
```

- d. 检查是否扩容成功。使用lsblk命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到dockersys盘，则表示扩容成功。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                  8:0    0   50G  0 disk
├─vda1                              8:1    0   50G  0 part /
└─vgdb                              8:16   0  200G  0 disk
   └─vgpaas-dockersys              253:0   0  118G  0 lvm  /var/lib/docker # 扩容后的
      dockersys盘
         └─vgpaas-thinpool_tmeta    253:1   0    3G  0 lvm
            └─vgpaas-thinpool      253:3   0   67G  0 lvm
               ...
         └─vgpaas-thinpool_tdata    253:2   0   67G  0 lvm
            └─vgpaas-thinpool      253:3   0   67G  0 lvm
               ...
         └─vgpaas-kubernetes        253:4   0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

----结束

检查项九：检查节点上调度的 Pod 是否过多

0/1 nodes are available: 1 Too many pods.表示节点上调度的Pod过多，超出可调度的最大实例数。

创建节点时，在“高级配置”中可选择设置“最大实例数”参数，设置节点上可以正常运行的容器 Pod 的数目上限。该数值的默认值随节点规格浮动，您也可以手动设置。

图 6-2 最大实例数



您可以在“节点管理”页面，查看节点的“Pod(已分配/可用地址总数/总额度)”参数列，检查节点已调度的容器是否达到上限。若已达到上限，可通过添加节点或修改最大实例数的方式解决。

您可通过以下方式修改“最大实例数”参数：

- 默认节点池中的节点：通过重置节点时修改“最大实例数”。
- 自定义节点池中的节点：可修改节点池配置参数中的max-pods参数。详情请参见[节点池配置管理](#)。

图 6-3 查看容器数



排查项十：kubelet 静态绑核异常

当kubelet开启静态绑核（cpuManagerPolicy设置为static），且Pod的QoS类型为Guaranteed时，如果该Pod同时配置了整数CPU Request值的初始化容器（init container），且初始化容器的CPU Request值与主容器的CPU Request值不一样，会有概率泄漏导致Pod调度失败，出现UnexpectedAdmissionError报错。

社区相关issue：<https://github.com/kubernetes/kubernetes/issues/112228>

解决方案：

设置init容器CPU为小数，不绑定核心，但需保证Request值等于Limit值。

例如：主容器：{"limits":{"cpu":"7","memory":"60G"},"requests":{"cpu":"7","memory":"60G"}}，init容器{"limits":{"cpu":"6.9","memory":"60G"},"requests":{"cpu":"6.9","memory":"60G"}}

6.1.3 工作负载异常：实例拉取镜像失败

问题定位

当工作负载状态显示“实例未就绪：Back-off pulling image "xxxxx"”，该状态下工作负载实例K8s事件名称为“实例拉取镜像失败”或“重新拉取镜像失败”。查看K8s事件的方法请参见[Pod事件查看方法](#)。

排查思路

根据具体事件信息确定具体问题原因，如[表6-2](#)所示。

表 6-2 实例拉取镜像失败

事件信息	问题原因与解决方案
Failed to pull image "xxx": rpc error: code = Unknown desc = Error response from daemon: Get xxx: denied: You may not login yet	没有登录镜像仓库，无法拉取镜像。 排查项一：kubect创建工作负载时未指定imagePullSecret
Failed to pull image "nginx:v1.1": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-1.docker.io: no such host	镜像地址配置有误找不到镜像导致失败。 排查项二：填写的镜像地址错误（使用第三方镜像时） 排查项三：使用错误的密钥（使用第三方镜像时）
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "nginx-6dc48bf8b6-l8xrw": Error response from daemon: mkdir xxxxx: no space left on device	磁盘空间不足。 排查项四：节点磁盘空间不足
Failed to pull image "xxx": rpc error: code = Unknown desc = error pulling image configuration: xxx x509: certificate signed by unknown authority	从第三方仓库下载镜像时，第三方仓库使用了非知名或者不安全的证书。 排查项五：远程镜像仓库使用非知名或不安全的证书
Failed to pull image "XXX": rpc error: code = Unknown desc = context canceled	镜像体积过大。 排查项六：镜像过大导致失败
Failed to pull image "docker.io/bitnami/nginx:1.22.0-debian-11-r3": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)	排查项七：无法连接镜像仓库
ERROR: toomanyrequests: Too Many Requests. 或 you have reached your pull rate limit, you may increase the limit by authenticating an upgrading	由于拉取镜像次数达到上限而被限速。 排查项八：拉取公共镜像达上限

排查项一：kubectl 创建工作负载时未指定 imagePullSecret

当工作负载状态异常并显示“实例拉取镜像失败”的K8s事件时，请排查yaml文件中是否存在**imagePullSecrets**字段。

排查事项：

- 当Pull SWR容器镜像仓库的镜像时，name参数值需固定为default-secret。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx
          imagePullPolicy: Always
          name: nginx
          imagePullSecrets:
            - name: default-secret
```

- Pull第三方镜像仓库的镜像时，需设置为创建的secret名称。

kubectl创建工作负载拉取第三方镜像时，需指定的imagePullSecret字段，name表示pull镜像时的secret名称，创建密钥的方法请参见[使用kubectl创建第三方镜像仓库的密钥](#)。

排查项二：填写的镜像地址错误（使用第三方镜像时）

CCE支持拉取第三方镜像仓库中的镜像来创建工作负载。

在填写第三方镜像的地址时，请参照要求的格式来填写。镜像地址格式为：domainname/organization/imagename:tag，若没标注版本号则默认版本号为latest。

- 若是私有仓库，请填写domainname/organization/imagename:tag。
- 若是docker开源仓库，请填写name:version，例如nginx:latest。

图 6-4 第三方镜像

容器配置

容器信息

容器 - 1

基本信息

容器名称 container-1

镜像名称 nginx:latest [了解详情](#) [更换镜像](#)

生命周期

健康检查

环境变量

数据存储

安全设置

容器日志

CPU配额 申请 0.25 cores; 限制 0.25 cores ②

GPU配额 暂未安装GPU插件，无法使用此功能。 [点此安装插件](#)

特权容器 ☐ ②

镜像地址配置有误找不到镜像导致失败，Kubernetes Event中提示如下信息：

```
Failed to pull image "nginx:v1.1": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-1.docker.io: no such host
```

解决方案：

可编辑yaml修改镜像地址，也可在工作负载详情页面更新升级页签单击更换镜像。

排查项三：使用错误的密钥（使用第三方镜像时）

通常第三方镜像仓库都必须经过认证（账号密码）才可以访问，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必须先创建镜像仓库的密钥。

解决方案：

若您的密钥错误将会导致镜像拉取失败，请重新获取密钥。

创建密钥的方法请参见[使用kubectl创建第三方镜像仓库的密钥](#)。

排查项四：节点磁盘空间不足

当k8s事件中包含以下信息，表明节点上用于存储镜像的磁盘空间已满，会导致重新拉取镜像失败。您可以通过清理镜像或扩容磁盘解决该问题。

```
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "nginx-6dc48bf8b6-l8xrw": Error response from daemon: mkdir xxxxx: no space left on device
```

您可以执行以下命令，确认节点上存储镜像的磁盘空间：

```
lvs
```

```
[root@zhouxu-20650 ~]# lvs
LV          VG      Attr      LSize   Pool Origin         Data%  Meta%   Move Log Cpy%Sync Convert
kubernetes  vgpaas  -wi-ao--- <10.00g
thinpool    vgpaas  twi-aot--- 84.00g
[root@zhouxu-20650 ~]#
```

方案一：清理镜像

您可以执行以下步骤清理未使用的镜像：

- 使用containerd容器引擎的节点：
 - a. 查看节点上的本地镜像。
`crictl images -v`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`crictl rmi {镜像ID}`
- 使用docker容器引擎的节点：
 - a. 查看节点上的本地镜像。
`docker images`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`docker rmi {镜像ID}`

📖 说明

请勿删除cce-pause等系统镜像，否则可能导致无法正常创建容器。

方案二：扩容磁盘

扩容磁盘的操作步骤如下：

步骤1 在EVS控制台扩容数据盘。详情请参见[扩容云硬盘容量](#)。

在EVS控制台扩容成功后，仅扩大了云硬盘的存储容量，还需要执行后续步骤扩容逻辑卷和文件系统。

步骤2 登录CCE控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤3 登录目标节点。

步骤4 使用lsblk命令查看节点块设备信息。

这里存在两种情况，根据容器存储Rootfs而不同。

Overlayfs：没有单独划分thinpool，在dockersys空间下统一存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    0   50G  0 disk
└─sda1                8:1    0   50G  0 part /
sdb                  8:16    0  150G  0 disk # 数据盘已扩容至150G，存在50G空间仍未分配
└─vgpaas-dockersys 253:0    0   90G  0 lvm  /var/lib/containerd
   └─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

将新增的磁盘容量加到容器引擎使用的dockersys逻辑卷上。

a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/sdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/sdb
```

回显如下：

```
Physical volume "/dev/sdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <90.00 GiB (23039 extents) to 140.00
GiB (35840 extents).
Logical volume vgpaas/dockersys successfully resized.
```

c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/containerd; on-line resizing required
old_desc_blocks = 12, new_desc_blocks = 18
The filesystem on /dev/vgpaas/dockersys is now 36700160 blocks long.
```

3. 检查是否扩容成功。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    0   50G  0 disk
└─sda1                8:1    0   50G  0 part /
sdb                  8:16    0  150G  0 disk
└─vgpaas-dockersys 253:0    0  140G  0 lvm  /var/lib/containerd
   └─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

Devicemapper：单独划分了thinpool存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                  8:0    0   50G  0 disk
```



```
└─vda1          8:1  0  50G  0 part /
vdb             8:16 0 200G 0 disk
├─vgpaas-dockersys 253:0 0 18G 0 lvm /var/lib/docker
├─vgpaas-thinpool_tmeta 253:1 0 3G 0 lvm
├─└─vgpaas-thinpool 253:3 0 67G 0 lvm # thinpool空间
├─...
├─vgpaas-thinpool_tdata 253:2 0 67G 0 lvm
├─└─vgpaas-thinpool 253:3 0 67G 0 lvm
├─...
└─vgpaas-kubernetes 253:4 0 10G 0 lvm /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

选项一：将新增的磁盘容量加到thinpool盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中`/dev/vdb`为thinpool空间所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中`vgpaas/thinpool`为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/thinpool
```

回显如下：

```
Size of logical volume vgpaas/thinpool changed from <67.00 GiB (23039 extents) to <167.00 GiB (48639 extents).
Logical volume vgpaas/thinpool successfully resized.
```

- c. 由于thinpool未挂载到设备，因此无需调整文件系统的大小。

- d. 检查是否扩容成功。使用`lsblk`命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到thinpool盘，则表示扩容成功。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                8:0    0  50G  0 disk
└─vda1                             8:1    0  50G  0 part /
vdb                                8:16    0 200G  0 disk
├─vgpaas-dockersys                 253:0    0  18G  0 lvm /var/lib/docker
├─vgpaas-thinpool_tmeta            253:1    0   3G  0 lvm
├─└─vgpaas-thinpool                253:3    0 167G  0 lvm # 扩容后的thinpool空间
├─...
├─vgpaas-thinpool_tdata            253:2    0  67G  0 lvm
├─└─vgpaas-thinpool                253:3    0  67G  0 lvm
├─...
└─vgpaas-kubernetes               253:4    0  10G  0 lvm /mnt/paas/kubernetes/kubelet
```

选项二：将新增的磁盘容量加到dockersys盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中`/dev/vdb`为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中`vgpaas/dockersys`为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <18.00 GiB (4607 extents) to <118.00 GiB (30208 extents).
Logical volume vgpaas/dockersys successfully resized.
```

- c. 调整文件系统的大小。其中`/dev/vgpaas/dockersys`为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/docker; on-line resizing required
old_desc_blocks = 3, new_desc_blocks = 15
The filesystem on /dev/vgpaas/dockersys is now 30932992 blocks long.
```

- d. 检查是否扩容成功。使用`lsblk`命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到`dockersys`盘，则表示扩容成功。

```
# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda                                  8:0    0   50G  0 disk
├─vda1                              8:1    0   50G  0 part /
└─vgdb                              8:16   0  200G  0 disk
   └─vgpaas-dockersys                253:0   0  118G  0 lvm  /var/lib/docker # 扩容后的
      dockersys盘
         └─vgpaas-thinpool_tmeta      253:1   0    3G  0 lvm
            └─vgpaas-thinpool        253:3   0   67G  0 lvm
               ...
         └─vgpaas-thinpool_tdata      253:2   0   67G  0 lvm
            └─vgpaas-thinpool        253:3   0   67G  0 lvm
               ...
         └─vgpaas-kubernetes          253:4   0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

----结束

排查项五：远程镜像仓库使用非知名或不安全的证书

从第三方仓库下载镜像时，若第三方仓库使用了非知名或者不安全的证书，节点上会拉取镜像失败，Pod事件列表中有“实例拉取镜像失败”事件，报错原因为“x509: certificate signed by unknown authority”。

说明

当前EulerOS 2.9镜像中有进行安全增强，移除系统中部分非安全或过期知名证书配置，部分第三方镜像在其他类型节点上未报错，在EulerOS 2.9系统报此错误属正常现象，也可通过下述解决方案进行处理。

解决方案：

步骤1 确认报错unknown authority的第三方镜像服务器地址和端口。

从“实例拉取镜像失败”事件信息中能够直接看到报错的第三方镜像服务器地址和端口，如上图中错误信息为：

```
Failed to pull image "bitnami/redis-cluster:latest": rpc error: code = Unknown desc = error pulling image configuration: Get https://production.cloudflare.docker.com/registry-v2/docker/registry/v2/blobs/sha256/e8/e83853f03a2e792614e7c1e6de75d63e2d6d633b4e7c39b9d700792ee50f7b56/data?verify=1636972064-AQbl5RActnuddZV%2F3EShZwnqOe8%3D: x509: certificate signed by unknown authority
```

对应的第三方镜像服务器地址为 `production.cloudflare.docker.com`，端口为https默认端口 `443`。

步骤2 在需要下载第三方镜像的节点上加载第三方镜像服务器的根证书。

EulerOS, CentOS节点执行如下命令，`{server_url}:{server_port}`需替换成步骤1中地址和端口，如 `production.cloudflare.docker.com:443`。

若节点的容器引擎为containerd，最后一步“`systemctl restart docker`”命令替换为“`systemctl restart containerd`”。

```
openssl s_client -showcerts -connect {server_url}:{server_port} < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /etc/pki/ca-trust/source/anchors/tmp_ca.crt
```

```
update-ca-trust
systemctl restart docker
```

ubuntu节点执行如下命令。

```
openssl s_client -showcerts -connect {server_url}:{server_port} < /dev/null | sed -ne '/-BEGIN
CERTIFICATE-/,/-END CERTIFICATE-/p' > /usr/local/share/ca-certificates/tmp_ca.crt
update-ca-trust
systemctl restart docker
```

----结束

排查项六：镜像过大导致失败

Pod事件列表中有“实例拉取镜像失败”事件，报错原因如下。这可能是镜像较大导致的情况。

```
Failed to pull image "XXX": rpc error: code = Unknown desc = context canceled
```

登录节点使用**docker pull**命令手动下拉镜像，镜像拉取成功。

问题根因：

Kubernetes默认存在拉取镜像超时时间，如果一定时间内镜像下载没有任何进度更新，下载动作就会取消。在节点性能较差或镜像较大时，可能出现镜像无法成功下载，负载启动失败的现象。

解决方案：

- 方案一（推荐）：
 - a. 登录节点手动下载镜像。
 - Containerd节点：

```
crictl pull <image-address>
```
 - Docker节点：

```
docker pull <image-address>
```
 - b. 创建负载时，确认负载的镜像拉取策略imagePullPolicy为IfNotPresent（默认策略配置），此时负载会使用已拉取到本地的镜像。
- 方案二（仅支持v1.25及以上版本的集群）：修改节点池的配置参数。DefaultPool节点池中的节点不支持修改该参数。
 - a. 登录CCE控制台。
 - b. 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。
 - c. 单击节点池名称后的“更多 > 配置管理”。
 - d. 在侧边栏滑出的“配置管理”窗口中，修改“容器引擎Docker/Containerd配置”的image-pull-progress-timeout参数。该参数用于设置镜像拉取的超时时长。
 - e. 单击“确定”，完成配置操作。

排查项七：无法连接镜像仓库

问题现象

创建工作负载时报如下错误。

```
Failed to pull image "docker.io/bitnami/nginx:1.22.0-debian-11-r3": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

问题原因

无法连接镜像仓库，网络不通。SWR仅支持直接拉取Docker官方的镜像，其他仓库的镜像需要连接公网。

解决方案：

- 方案一：给需要下载镜像的节点绑定公网IP。
- 方案二：先将镜像上传到SWR，然后从SWR拉取镜像。

排查项八：拉取公共镜像达上限

问题现象

创建工作负载时报如下错误。

```
ERROR: toomanyrequests: Too Many Requests.
```

或

```
you have reached your pull rate limit, you may increase the limit by authenticating an upgrading: https://www.docker.com/increase-rate-limits.
```

问题原因

DockerHub对用户拉取容器镜像请求设定了上限，详情请参见[Understanding Docker Hub Rate Limiting](#)。

解决方案：

将常用的镜像上传到SWR，然后从SWR拉取镜像。

6.1.4 工作负载异常：启动容器失败

问题定位

工作负载详情中，若事件中提示“启动容器失败”，请按照如下方式来初步排查原因：

步骤1 登录异常工作负载所在的节点。

步骤2 查看工作负载实例非正常退出的容器ID。

如果节点为docker，请执行docker命令：

```
docker ps -a | grep $podName
```

如果节点为containerd，请执行containerd命令：

```
crictl ps -a | grep $podName
```

步骤3 查看退出容器的错误日志。

如果节点为docker，请执行docker命令：

```
docker logs $containerID
```

如果节点为containerd，请执行containerd命令：

```
crictl logs $containerID
```

根据日志提示修复工作负载本身的问题。

步骤4 查看操作系统的错误日志，例如查询日志中是否存在OOM信息。

```
cat /var/log/messages | grep $containerID | grep oom
```

根据日志判断是否触发了系统OOM。

----结束

排查思路

根据具体事件信息确定具体问题原因，如表6-3所示。

表 6-3 容器启动失败

日志或事件信息	问题原因	排查与解决方案
Pod日志中存在exit(0)	容器中无进程。	请调试容器是否能正常运行，详情请参见 容器中无持续运行的进程（退出码：0） 。
- Kubernetes事件信息： Liveness probe failed: Get http... - Pod日志中存在 exit(137)	健康检查执行失败。	检查Pod中所配置的容器健康检查（Liveness Probe）策略是否符合预期，详情请参见 健康检查执行失败（退出码：137） 。
- Kubernetes事件信息如下： Thin Pool has 15991 free data blocks which is less than minimum required 16383 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior - Pod日志中存在no left space	磁盘空间不足。	您需要扩容或清理磁盘空间，详情请参见 容器所在磁盘空间不足 。
Pod日志中存在OOM字眼	Pod内存不足。	检查Pod的资源配置是否正确，详情请参见 容器资源配置过小 。
Pod日志中存在Address already in use	Pod中容器端口冲突。	检查Pod是否出现容器端口冲突，详情请参见 同一Pod中container端口冲突 。

日志或事件信息	问题原因	排查与解决方案
Kubernetes事件信息如下： Error: failed to start container "filebeat": Error response from daemon: OCI runtime create failed: container_linux.go:330: starting container process caused "process_linux.go:381: container init caused \"setenv: invalid argument\"": unknown	负载中挂载了Secret，Secret对应的值没有进行base64加密。	解决方案详情请参见 工作负载挂载的密钥值不符合要求 。
Kubernetes事件信息如下： the failed container exited with ExitCode: 255	可能是在ARM架构的节点上运行x86的容器镜像。	解决方案详情参见 容器镜像版本与节点架构不符 。
Kubernetes事件信息如下： the failed container exited with ExitCode: 141	containerd版本与tail版本不兼容导致。	解决方案详情参见 容器启动命令中执行tail -f xx退出（退出码：141） 。
Kubernetes事件信息如下： Created container init-pinpoint	Java探针的版本不兼容导致。	解决方案详情参见 JAVA探针的版本不兼容 。
其他Pod日志	请结合业务进行排查。	排查方案请参见 业务配置问题排查 。

容器中无持续运行的进程（退出码：0）

步骤1 登录异常工作负载所在的节点。

步骤2 查看容器状态。

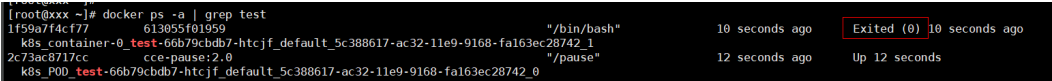
如果节点为docker，请执行docker命令：

```
docker ps -a | grep $podName
```

如果节点为containerd，请执行containerd命令：

```
crictl ps -a | grep $podName
```

如下图所示：



当容器中无持续运行的进程时，会出现exit(0)的状态码，此时说明容器中无进程。

----结束

健康检查执行失败（退出码：137）

工作负载配置的健康检查会定时检查业务，异常情况下pod会报实例不健康的事件且pod一直重启失败。

工作负载若配置liveness型（工作负载存活探针）健康检查，当健康检查失败次数超过阈值时，会重启实例中的容器。在工作负载详情页面查看事件，若K8s事件中出现“Liveness probe failed: Get http...”时，表示健康检查失败。

解决方案：

请在工作负载详情页中，切换至“容器管理”页签，核查容器的“健康检查”配置信息，排查健康检查策略是否合理或业务是否已异常。

容器所在磁盘空间不足

如下磁盘为创建节点时选择的docker专用盘分出来的thinpool盘，以root用户执行lvs命令可以查看当前磁盘的使用量。

Thin Pool has 15991 free data blocks which is less than minimum required 16383 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior



LV	VG	Attr	LSIZE	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync	Convert
dockersys	vgpaas	-wi-ao----	<18.00g									
kubernetes	vgpaas	-wi-ao----	<18.00g									
thinpool	vgpaas	twi-aot---	67.00g			98.84	1.32					

解决方案：

方案一：清理镜像

您可以执行以下步骤清理未使用的镜像：

- 使用containerd容器引擎的节点：
 - a. 查看节点上的本地镜像。
`crictl images -v`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`crictl rmi {镜像ID}`
- 使用docker容器引擎的节点：
 - a. 查看节点上的本地镜像。
`docker images`
 - b. 确认镜像无需使用，并通过镜像ID删除无需使用的镜像。
`docker rmi {镜像ID}`

说明

请勿删除cce-pause等系统镜像，否则可能导致无法正常创建容器。

方案二：扩容磁盘

扩容磁盘的操作步骤如下：

步骤1 在EVS控制台扩容数据盘。详情请参见[扩容云硬盘容量](#)。

在EVS控制台扩容成功后，仅扩大了云硬盘的存储容量，还需要执行后续步骤扩容逻辑卷和文件系统。

步骤2 登录CCE控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤3 登录目标节点。

步骤4 使用lsblk命令查看节点块设备信息。

这里存在两种情况，根据容器存储Rootfs而不同。

Overlaysfs：没有单独划分thinpool，在dockersys空间下统一存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0  0   50G  0 disk
└─sda1       8:1  0   50G  0 part /
sdb          8:16  0  150G  0 disk  # 数据盘已扩容至150G，存在50G空间仍未分配
├─vgpaas-dockersys 253:0  0   90G  0 lvm  /var/lib/containerd
└─vgpaas-kubernetes 253:1  0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

将新增的磁盘容量加到容器引擎使用的dockersys逻辑卷上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中/dev/sdb为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/sdb
```

回显如下：

```
Physical volume "/dev/sdb" changed
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中vgpaas/dockersys为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <90.00 GiB (23039 extents) to 140.00 GiB (35840 extents).
Logical volume vgpaas/dockersys successfully resized.
```

- c. 调整文件系统的大小。其中/dev/vgpaas/dockersys为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/containerd; on-line resizing required
old_desc_blocks = 12, new_desc_blocks = 18
The filesystem on /dev/vgpaas/dockersys is now 36700160 blocks long.
```

3. 检查是否扩容成功。

```
# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0  0   50G  0 disk
└─sda1       8:1  0   50G  0 part /
sdb          8:16  0  150G  0 disk
├─vgpaas-dockersys 253:0  0  140G  0 lvm  /var/lib/containerd
└─vgpaas-kubernetes 253:1  0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

Devicemapper：单独划分了thinpool存储镜像相关数据。

1. 查看设备的磁盘和分区大小。

```
# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda          8:0  0   50G  0 disk
└─vda1       8:1  0   50G  0 part /
vdb          8:16  0  200G  0 disk
├─vgpaas-dockersys 253:0  0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta 253:1  0    3G  0 lvm
├─vgpaas-thinpool 253:3  0   67G  0 lvm  # thinpool空间
├─...
├─vgpaas-thinpool_tdata 253:2  0   67G  0 lvm
├─vgpaas-thinpool 253:3  0   67G  0 lvm
├─...
└─vgpaas-kubernetes 253:4  0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

2. 扩容磁盘。

选项一：将新增的磁盘容量加到thinpool盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中`/dev/vdb`为thinpool空间所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed  
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中`vgpaas/thinpool`为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/thinpool
```

回显如下：

```
Size of logical volume vgpaas/thinpool changed from <67.00 GiB (23039 extents) to <167.00  
GiB (48639 extents).  
Logical volume vgpaas/thinpool successfully resized.
```

- c. 由于thinpool未挂载到设备，因此无需调整文件系统的大小。
- d. 检查是否扩容成功。使用`lsblk`命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到thinpool盘，则表示扩容成功。

```
# lsblk  
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT  
vda                                8:0    0   50G  0 disk  
├─vda1                             8:1    0   50G  0 part /  
└─vdb                              8:16   0  200G  0 disk  
    ├─vgpaas-dockersys             253:0    0   18G  0 lvm  /var/lib/docker  
    ├─vgpaas-thinpool_tmeta         253:1    0    3G  0 lvm  
    │ └─vgpaas-thinpool            253:3    0  167G  0 lvm      # 扩容后的thinpool空间  
    │ ...  
    ├─vgpaas-thinpool_tdata         253:2    0   67G  0 lvm  
    └─vgpaas-thinpool              253:3    0   67G  0 lvm  
    ...  
    └─vgpaas-kubernetes            253:4    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

选项二：将新增的磁盘容量加到dockersys盘上。

- a. 扩容物理卷PV，让LVM识别EVS新增的容量。其中`/dev/vdb`为dockersys逻辑卷所在的物理卷。

```
pvresize /dev/vdb
```

回显如下：

```
Physical volume "/dev/vdb" changed  
1 physical volume(s) resized or updated / 0 physical volume(s) not resized
```

- b. 将空闲容量100%扩容到逻辑卷LV。其中`vgpaas/dockersys`为容器引擎使用的逻辑卷。

```
lvextend -l+100%FREE -n vgpaas/dockersys
```

回显如下：

```
Size of logical volume vgpaas/dockersys changed from <18.00 GiB (4607 extents) to <118.00  
GiB (30208 extents).  
Logical volume vgpaas/dockersys successfully resized.
```

- c. 调整文件系统的大小。其中`/dev/vgpaas/dockersys`为容器引擎的文件系统路径。

```
resize2fs /dev/vgpaas/dockersys
```

回显如下：

```
Filesystem at /dev/vgpaas/dockersys is mounted on /var/lib/docker; on-line resizing required  
old_desc_blocks = 3, new_desc_blocks = 15  
The filesystem on /dev/vgpaas/dockersys is now 30932992 blocks long.
```

- d. 检查是否扩容成功。使用`lsblk`命令查看设备的磁盘和分区大小，若新增的磁盘容量已经加到dockersys盘，则表示扩容成功。

```
# lsblk  
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT  
vda                                8:0    0   50G  0 disk  
├─vda1                             8:1    0   50G  0 part /
```

```
vdb                8:16  0 200G 0 disk
├─vgpaas-dockersys 253:0  0 118G 0 lvm /var/lib/docker # 扩容后的
dockersys盘
├─vgpaas-thinpool_tmeta 253:1  0 3G 0 lvm
├─vgpaas-thinpool 253:3  0 67G 0 lvm
├─...
├─vgpaas-thinpool_tdata 253:2  0 67G 0 lvm
├─vgpaas-thinpool 253:3  0 67G 0 lvm
├─...
└─vgpaas-kubernetes 253:4  0 10G 0 lvm /mnt/paas/kubernetes/kubelet
```

----结束

容器资源配置过小

事件详情中有OOM字样。并且，在日志中也会有记录：

```
cat /var/log/messages | grep 96feb0a425d6 | grep oom
```

```
[root@xxx ~]# cat /var/log/messages | grep 96feb0a425d6 | grep oom
2019-07-22T11:57:49.441756+08:00 xxx dockerd: time="2019-07-22T11:57:49.440755329+08:00" level=info msg=event OOMKilled=true containe
rID=96feb0a425d6669f8f062cf3a6096868617a10711334f6d5bce4a6ee6eadc82d module=libcontainerd namespace=moby topic=/tasks/oom
2019-07-22T11:59:55.828162+08:00 xxx [/bin/bash]: [2019-07-22T11:57:49.441756+08:00 xxx dockerd: time="2019-07-22T11:57:49.440755329+
08:00" level=info msg=event OOMKilled=true containerID=96feb0a425d6669f8f062cf3a6096868617a10711334f6d5bce4a6ee6eadc82d module=libcon
tainerd namespace=moby topic=/tasks/oom] return code=[127], execute failed by [root(uid=0)] from [pts/0 (192.168.0.7)]
2019-07-22T12:01:47.621029+08:00 xxx [/bin/bash]: [cat /var/log/messages | grep 96feb0a425d6 | grep oom] return code=[0], execute suc
cess by [root(uid=0)] from [pts/0 (192.168.0.7)]
[root@xxx ~]#
```

创建工作负载时，设置的限制资源若小于实际所需资源，会触发系统OOM，并导致容器异常退出。

同一 Pod 中 container 端口冲突

步骤1 登录异常工作负载所在的节点。

步骤2 查看工作负载实例非正常退出的容器ID。

如果节点为docker，请执行docker命令：

```
docker ps -a | grep $podName
```

如果节点为containerd，请执行containerd命令：

```
crictl ps -a | grep $podName
```

步骤3 查看退出容器的错误日志。

如果节点为docker，请执行docker命令：

```
docker logs $containerID
```

如果节点为containerd，请执行containerd命令：

```
crictl logs $containerID
```

根据日志提示修复工作负载本身的问题。如下图所示，即同一Pod中的container端口冲突导致容器启动失败。

图 6-5 container 冲突导致容器启动失败

```
[root@k8s-master ~]# docker ps -a|grep test2
aebc17c4d66c          94818572c4ef          "nginx -g 'daemon ..." 8 seconds ago
ExitCode: 1          Exited (1) 5 seconds ago
lt_38892324-94b7-11e9-aa5f-fa163e07fc60_3
0c43d629292e          nginx
Up About a minute
lt_38892324-94b7-11e9-aa5f-fa163e07fc60_0
3484b34393ce          cfe-pause:11.23.1
Up About a minute
lt_38892324-94b7-11e9-aa5f-fa163e07fc60_0
k8s_CONTAINER_1_test2-65dbb945d6-xh9n2_default
k8s_CONTAINER_0_test2-65dbb945d6-xh9n2_default
k8s_POD_test2-65dbb945d6-xh9n2_default_38892324-94b7-11e9-aa5f-fa163e07fc60_0

[root@k8s-master ~]# docker logs aebc17c4d66c
2019/06/22 06:31:29 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2019/06/22 06:31:29 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2019/06/22 06:31:29 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2019/06/22 06:31:29 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2019/06/22 06:31:29 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2019/06/22 06:31:29 [emerg] 1#1: still could not bind()
nginx: [emerg] still could not bind()
```

----结束

解决方案:

配置正确的容器端口，确保端口不冲突，然后重新创建工作负载。

如果Pod使用主机网络（即配置**hostNetwork: true**）也可能会出现容器端口冲突，这是因为Pod内的容器会直接与宿主机共享网络接口和端口空间，多个使用相同端口的Pod无法运行在同一节点上。

工作负载挂载的密钥值不符合要求

事件详情中出现以下错误:

```
Error: failed to start container "filebeat": Error response from daemon: OCI runtime create failed: container_linux.go:330: starting container process caused "process_linux.go:381: container init caused \"setenv: invalid argument\\\": unknown"
```

出现以上问题的根因是由于工作负载中挂载了Secret，但Secret对应的值没有进行base64加密。

解决方案:

通过控制台创建Secret，Secret对应的值会自动进行base64加密。

如果通过YAML进行创建，需要手动对密钥值进行base64加密:

```
echo -n "待编码内容" | base64
```

容器镜像版本与节点架构不符

在ARM架构的节点上创建工作负载时未使用正确的镜像版本，使用正确的镜像版本即可解决该问题。

如果您需要构建x86和ARM双架构镜像，请参考[CCE中使用x86和ARM双架构镜像](#)。

容器启动命令中执行 **tail -f xx** 退出（退出码：141）

Kubernetes事件为:

```
the failed container exited with ExitCode: 141
```

问题根因：

containerd历史版本和容器镜像中tail>=8.28版本不兼容，导致tail -f 命令退出，退出码为141。

短期解决方案：

将启动参数中的tail -f xx修改为sleep 2 && tail -f xx，重新创建工作负载。

长期解决方案：

- 将集群升级到v1.25.16-r20、v1.27.16-r20、v1.28.15-r10、v1.29.10-r10、v1.30.6-r10、v1.31.4-r0版本及以上。
- 重置节点，将容器引擎切换为docker。

JAVA 探针的版本不兼容

Kubernetes事件为：

```
Created container init-pinpoint
```

解决方案：

1. 在创建工作负载时，在“高级配置”步骤下的“性能管理配置”项目下，JAVA探针版本请选择最新的版本（例如：1.0.36），不要选择latest版本。
2. 如果工作负载创建时的JAVA探针版本已经选择了latest版本，可以更新工作负载配置，将JAVA探针版本请选择为最新的版本（例如：1.0.36）。

业务配置问题排查

请检查工作负载启动命令是否正确执行，或工作负载本身bug导致容器不断重启。

步骤1 登录异常工作负载所在的节点。

步骤2 查看工作负载实例非正常退出的容器ID。

如果节点为docker，请执行docker命令：

```
docker ps -a | grep $podName
```

如果节点为containerd，请执行containerd命令：

```
crictl ps -a | grep $podName
```

步骤3 查看退出容器的错误日志。

如果节点为docker，请执行docker命令：

```
docker logs $containerID
```

如果节点为containerd，请执行containerd命令：

```
crictl logs $containerID
```

注意：这里的containerID为已退出的容器的ID

图 6-6 容器启动命令配置不正确

```
[root@dcba-ha-11638 ~]# docker ps -a | grep nginx
cf0352f617f9      3f8a4339aadd      "/bin/bash /tmp/test." 2 minutes ago
      Exited (127) 2 minutes ago                                k8s_container-0_nginx-267
0177225-kt929_test_d6402ef7-4e0f-11e8-b4f7-fa163e74044e_5
c2176ce394a1      cfe-pause:3.7.6    "/pause"                5 minutes ago
      Up 5 minutes                                              k8s_POD_nginx-2670177225-
kt929_test_d6402ef7-4e0f-11e8-b4f7-fa163e74044e_0
[root@dcba-ha-11638 ~]# docker logs cf035
/bin/bash: /tmp/test.sh: No such file or directory
[root@dcba-ha-11638 ~]#
```

如上图所示，容器配置的启动命令不正确导致容器启动失败。其他错误请根据日志提示修复工作负载本身的BUG问题。

----结束

解决方案：

重新创建工作负载，并配置正确的启动命令。

6.1.5 工作负载异常：实例驱逐异常（Evicted）

驱逐原理

当节点出现异常时，为了保证工作负载的可用性，Kubernetes会通过驱逐机制（Eviction）将该节点上的Pod调离异常节点。

目前Kubernetes中存在两种Eviction机制，分别由**kube-controller-manager**和**kubelet**实现。

- **kube-controller-manager实现的驱逐**

kube-controller-manager主要由多个控制器构成，而驱逐的功能主要由node controller这个控制器实现，它会周期性检查所有节点状态，当节点处于NotReady状态超过一段时间后，驱逐该节点上所有Pod。

kube-controller-manager提供了以下启动参数控制驱逐：

- **pod-eviction-timeout**：即当节点宕机时间超过一定的时间间隔后，开始驱逐宕机节点上的Pod，默认为5min。
- **node-eviction-rate**：每秒需要排空的节点数量，默认为0.1，即每10s从一个节点驱逐Pod。
- **secondary-node-eviction-rate**：第二档的排空节点的速率。当集群中宕机节点过多时，排空节点的速率会降低至第二档，默认为0.01。
- **unhealthy-zone-threshold**：可用区的不健康阈值，默认为0.55，即当该可用区中节点宕机数目超过55%时，认为该可用区不健康。
- **large-cluster-size-threshold**：集群的大规模阈值，默认为50，当集群节点数量超过该阈值时认为集群属于大规模集群。大规模集群的可用区节点宕机数目超过55%时，则将排空节点速率降为0.01；假如是小规模集群，则将速率直接降为0，即停止驱逐节点上的Pod。

- **kubelet的eviction机制**

如果节点处于资源压力，那么kubelet就会执行驱逐策略。驱逐会考虑Pod的优先级，资源使用和资源申请。当优先级相同时，资源使用/资源申请最大的Pod会被首先驱逐。

kube-controller-manager的驱逐机制是粗粒度的，即驱逐一个节点上的所有Pod，而kubelet则是细粒度的，它驱逐的是节点上的某些Pod。此类驱逐会周期性检查本节点内存、磁盘等资源，当资源不足时，按照优先级驱逐部分Pod。关于Pod驱逐优先级，请参见[kubelet驱逐时Pod的选择](#)。

驱逐阈值分为软驱逐条件（Soft Eviction Thresholds）和硬驱逐条件（Hard Eviction Thresholds）两种机制，如下：

- **软驱逐条件**：当节点的内存/磁盘空间达到一定的阈值后，kubelet不会马上回收资源，如果改善到低于阈值就不进行驱逐，若这段时间一直高于阈值就进行驱逐。

您可以通过以下参数配置软驱逐条件：

- **eviction-soft:** 软驱逐阈值设置。当节点**驱逐信号**满足一定阈值时，例如memory.available<1.5Gi时，kubelet不会立即执行Pod驱逐，而会等待eviction-soft-grace-period时间，假如该时间过后，依然还是达到了软驱逐阈值，则触发一次Pod驱逐。
- **eviction-soft-grace-period:** 当达到软驱逐阈值时，允许Pod优雅终止的时间，即软驱逐宽限期，软驱逐信号与驱逐处理之间的时间差。
- **eviction-max-pod-grace-period:** 最大驱逐pod宽限期，停止信号与kill之间的时间差。
- **硬驱逐条件:** 硬驱逐机制则简单得多，一旦达到阈值，直接把Pod从本地驱逐。

您可以通过以下参数配置硬驱逐条件：

eviction-hard: 硬驱逐阈值设置。当节点**驱逐信号**满足一定阈值时，例如memory.available<1Gi，即当节点可用内存低于1Gi时，会立即触发一次Pod驱逐。

kubelet 具有以下默认硬驱逐条件：

- memory.available<100Mi
- nodefs.available<10%
- imagefs.available<15%
- nodefs.inodesFree<5%（Linux 节点）

除此之外，kubelet还提供了其他的驱逐参数：

- **eviction-pressure-transition-period:** 驱逐等待时间。当出现节点压力驱逐时，节点需要等待一定的时间，才会被设置为DiskPressure或者MemoryPressure，然后开启Pod驱逐，该时间默认为5分钟。该参数可以防止在某些情况下，节点在软驱逐条件上下振荡而出现错误的驱逐决策。
- **eviction-minimum-reclaim:** 表示每一次驱逐必须至少回收多少资源。该参数可以避免在某些情况下，驱逐Pod只会回收少量的资源，导致kubelet反复触发多次驱逐。

问题定位

若节点故障时，实例未被驱逐，请先按照如下方法进行问题定位。

使用如下命令发现很多pod的状态为Evicted：

```
kubectl get pods
```

在节点的kubelet日志中会记录Evicted相关内容，搜索方法可参考如下命令：

```
cat /var/log/cce/kubernetes/kubelet.log | grep -i Evicted -C3
```

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- **排查项一：节点是否存在资源压力**

- **排查项二：是否在实例上设置了tolerations**
- **排查项三：是否满足停止驱逐实例的条件**
- **排查项四：容器与节点上的“资源分配量”是否一致**
- **排查项五：工作负载实例不断失败并重新部署**

排查项一：节点是否存在资源压力

当满足硬性或软性驱逐条件时，即存在资源压力时，kubelet会根据驱逐信号将节点设置为相应的**节点状况**，并为节点打上对应的污点。请通过以下步骤查看节点是否存在对应的污点。

```
$ kubectl describe node 192.168.0.37
Name:          192.168.0.37
...
Taints:        key1=value1:NoSchedule
...
```

表 6-4 存在资源压力的节点状况及解决方案

节点状况	节点污点	驱逐信号	描述	解决方案
MemoryPressure	node.kubernetes.io/memory-pressure	memory.available	节点上的可用内存已满足驱逐条件。	您可以扩容节点规格，详情请参见 如何变更CCE集群中的节点规格？ 。
DiskPressure	node.kubernetes.io/disk-pressure	nodefs.available、nodefs.inodesFree、imagefs.available 或 imagefs.inodesFree	节点的根文件系统或镜像文件系统上的可用磁盘空间和 inode 已满足驱逐条件。	您可以扩容节点磁盘空间，详情请参见 存储扩容 。
PIDPressure	node.kubernetes.io/pid-pressure	pid.available	节点上的可用进程标识符已低于驱逐条件。	您可以修改节点进程ID上限，详情请参见 修改节点进程 ID数量上限 kernel.pid_max 。

排查项二：是否在实例上设置了 tolerations

通过kubectl工具或单击对应工作负载后的“更多 > 编辑YAML”，检查工作负载上是不是设置了容忍度，具体请参见[污点和容忍度](#)。

排查项三：是否满足停止驱逐实例的条件

若属于小规格的集群（集群节点数小于50个节点），如果故障的节点大于总节点数的55%，实例的驱逐将会被暂停。此情况下Kubernetes将不再尝试驱逐故障节点的工作负载，具体请参见[节点驱逐速率限制](#)。

排查项四：容器与节点上的“资源分配量”是否一致

容器被驱逐后还会频繁调度到原节点。

问题原因：

节点驱逐容器是根据节点的“资源使用率”进行判断；容器的调度规则是根据节点上的“资源分配量”进行判断。由于判断标准不同，所以可能会出现被驱逐后又再次被调度到原节点的情况。

解决方案：

遇到此类问题时，请合理分配各容器的资源分配量即可解决。

排查项五：工作负载实例不断失败并重新部署

工作负载实例出现不断失败，不断重新部署的情况。

问题分析：

pod驱逐后，如果新调度到的节点也有驱逐情况，就会再次被驱逐；甚至出现pod不断被驱逐的情况。

如果是由kube-controller-manager触发的驱逐，会留下一个状态为Terminating的pod；直到容器所在节点状态恢复后，pod才会自动删除。如果节点已经删除或者其他原因导致的无法恢复，可以使用“强制删除”删除pod。

如果是由kubelet触发的驱逐，会留下一个状态为Evicted的pod，此pod只是方便后期定位的记录，可以直接删除。

解决方案：

使用如下命令删除旧驱赶的遗留：

```
kubectrl get pods -n <namespace> | grep Evicted | awk '{print $1}' | xargs kubectrl delete pod -n <namespace>
```

<namespace>为命名空间名称，请根据需要指定。

参考

[Kubelet does not delete evicted pods](#)

提交工单

如果上述方法均不能解决您的疑问，请[提交工单](#)寻求更多帮助。

6.1.6 工作负载异常：存储卷无法挂载或挂载超时

排查思路

- [EVS存储卷挂载异常](#)

- [SFS Turbo存储卷挂载异常](#)
- [SFS存储卷挂载异常](#)
- [存储卷挂载超时](#)

EVS 存储卷挂载异常

问题现象	问题根因	解决方案
在有状态工作负载上挂载EVS存储卷，但无法挂载卷并超时。	节点在 可用区1 ，而要挂载的磁盘在 可用区2 ，导致无法挂载而超时。	在同一可用区内创建磁盘再挂载后即可正常。
创建Pod失败，并出现“添加存储失败”的事件，事件信息如下： Multi-Attach error for volume "pvc-62a7a7d9-9dc8-42a2-8366-0f5ef9db5b60" Volume is already used by pod(s) testttt-7b774658cb-lc98h	查看Deployment的副本数是否大于1。 Deployment中使用EVS存储卷时，副本数只能为1。若用户在后台指定Deployment的实例数为2以上，此时CCE并不会限制Deployment的创建。但若这些实例Pod被调度到不同的节点，则会有部分Pod因为其要使用的EVS无法被挂载到节点，导致Pod无法启动成功。	使用EVS的Deployment的副本数指定为1，或使用其他类型存储卷。
创建Pod失败，出现以下类似信息： MountVolume.MountDevice failed for volume "pvc-08178474-c58c-4820-a828-14437d46ba6f" : rpc error: code = Internal desc = [09060def-afd0-11ec-9664-fa163eef47d0] /dev/sda has file system, but it is detected to be damaged	磁盘文件系统损坏。	1. 在EVS中对磁盘进行备份，具体操作步骤请参见创建云硬盘备份。 2. 在节点上执行如下命令修复文件系统。 fsck -y {盘符} 若执行命令以上命令的过程中，出现“has unsupported feature(s): xxx”这类错误，如下所示： <pre>[root@reptimedb-audit-8jwml ~]# fsck -y /dev/sdb fsck from util-linux 2.23.2 e2fsck 1.42.9 (28-Dec-2013) /dev/sdb has unsupported feature(s): metadata_csum e2fsck: Get a newer version of e2fsck!</pre>说明节点fsck与文件系统不兼容，建议先升级fsck，再尝试执行以上命令。

SFS Turbo 存储卷挂载异常

问题现象	问题根因	解决方案
- 普通容器场景，Pod状态为“处理中”，且Pod的事件中有如下内容： MountVolume.SetUp failed for volume {pv name}... - 安全容器场景，Pod状态为“实例异常”，且Pod的事件中有如下内容： mount {sfsturbo共享地址} to xxx failed	- PV中共享地址不正确。 - Pod所在节点与需挂载的SFS Turbo文件系统间的网络不通。	- 检查PV中共享地址是否正确。 获取PV的YAML，查看spec.csi.volumeAttributes下的everest.io/share-export-location字段值即为共享地址，正确的共享地址是指定的SFS Turbo文件系统的共享路径。 <pre>kubectl get pv {pv name} -o jsonpath='{.spec.csi.volumeAttributes.everest.io/share-export-location}'</pre> 若指定了子路径，则要求子路径格式正确，且是存在的子目录，例如192.168.135.24:/a/b/c。 - 检查Pod所在节点与需挂载的SFS Turbo文件系统间的网络是否打通。 在节点上执行如下指令，可测试SFS Turbo是否可以挂载。 <pre>mount -t nfs -o vers=3,nolock,noresvport {sfsturbo共享地址} /tmp</pre>

SFS 存储卷挂载异常

问题现象	问题根因	解决方案
Pod状态为“处理中”，且Pod的事件中存在“实例挂卷失败”的告警，具体事件如下： MountVolume.SetUp failed for volume {pv name}... please check whether the volumeAttributes of related PersistentVolume of the volume is correct and whether it can be mounted.	工作负载中使用了通用文件系统（原SFS 3.0 容量型），但未创建通用文件系统使用的VPCEP，导致文件系统无法访问。	在集群所在VPC创建一个VPC终端节点，配置VPC终端节点的方法请参见 配置VPC终端节点 。

存储卷挂载超时

如果您挂载的存储中内容太多，同时又配置了以下几条配置，最终会由于逐个修改文件权限，而导致挂载时间过长。

问题定位：

- Securitycontext字段中是否包含runAsuser/fsGroup。securityContext是kubernetes中的字段，即安全上下文，它用于定义Pod或Container的权限和访问控制设置。
- 启动命令中是否包含ls、chmod、chown等查询或修改文件权限的操作。

解决建议：

请根据您的业务需求，判断是否需要修改。

6.1.7 工作负载异常：一直处于创建中

问题描述

节点上的工作负载一直处于创建中。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- **排查项一：cce-pause镜像是否被误删除**
- **排查项二：集群开启CPU管理策略后变更节点规格**

排查项一：cce-pause 镜像是否被误删除

问题现象

创建工作负载时报如下错误，显示无法创建sandbox，原因是拉取cce-pause:3.1镜像失败。

```
Failed to create pod sandbox: rpc error: code = Unknown desc = failed to get sandbox image "cce-pause:3.1": failed to pull image "cce-pause:3.1": failed to pull and unpack image "docker.io/library/cce-pause:3.1": failed to resolve reference "docker.io/library/cce-pause:3.1": pulling from host **** failed with status code [manifests 3.1]: 400 Bad Request
```

问题原因

该镜像为创建节点时添加的系统镜像，如果手动误删除该镜像，会导致工作负载Pod一直无法创建。

解决方案：

步骤1 登录该问题节点。

步骤2 手动解压节点上的cce-pause镜像安装包。

```
tar -xzf /opt/cloud/cce/package/node-package/pause-*.tgz
```

步骤3 导入镜像。

- Docker节点：
`docker load -i ./pause/package/image/cce-pause-*.tar`
- Containerd节点：
`ctr -n k8s.io images import --all-platforms ./pause/package/image/cce-pause-*.tar`

步骤4 镜像导入成功后，即可正常工作负载。

----结束

排查项二：集群开启 CPU 管理策略后变更节点规格

集群开启CPU管理策略（绑核）时，kubelet启动参数中会将CPU Manager的策略设置为static，允许为节点上具有某些资源特征的pod赋予增强的CPU亲和性和独占性。用户如果直接在ECS控制台对CCE节点变更规格，会由于变更前后CPU信息不匹配，导致节点上的负载无法重新拉起，也无法创建新负载。

步骤1 登录CCE节点（弹性云服务器）并删除cpu_manager_state文件。

删除命令示例如下：

```
rm -rf /mnt/paas/kubernetes/kubelet/cpu_manager_state
```

步骤2 重启节点或重启kubelet，重启kubelet的方法如下：

```
systemctl restart kubelet
```

此时重新拉起或创建工作负载，已可成功执行。

解决方式链接：[CCE节点变更规格后，为什么无法重新拉起或创建工作负载？](#)

----结束

6.1.8 工作负载异常：Pod 一直处于 Terminating 状态

问题描述

查询某个命名空间下的工作负载时，偶现部分Pod（实例）一直处于Terminating 状态。

例如，查询aos命名空间下的Pod：

```
#kubectl get pod -n aos
NAME                                READY   STATUS    RESTARTS   AGE
aos-apiserver-5f8f5b5585-s9l92     1/1     Terminating    0          3d1h
aos-cmdbserver-789bf5b497-6rwrq    1/1     Running         0          3d1h
aos-controller-545d78bs8d-vm6j9    1/1     Running         3          3d1h
```

通过**kubectl delete pods <podname> -n <namespace>** 命令始终无法将其删除：

```
kubectl delete pods aos-apiserver-5f8f5b5585-s9l92 -n aos
```

问题根因

Pod出现Terminating 状态的原因可能有多种，以下是一些常见的情况：

- **节点异常**：在节点处于“不可用”状态时，CCE会迁移节点上的容器实例，并将节点上运行的Pod置为Terminating状态。
待节点恢复后，处于Terminating状态的Pod会自动删除。
- **容器无响应**：如果Pod中的容器在终止过程中没有响应SIGTERM信号，则可能导致Pod卡在Terminating状态。

- **Pod中存在未处理完的请求或资源占用**：如果Pod中存在长时间运行的进程没有结束，则可能导致Pod无法被正常终止，进入Terminating状态。
- **Pod存在Finalizers**：Finalizers是一种允许在删除资源之前清理资源的机制。如果Pod有Finalizers，并且相关的清理操作被卡住或没有响应，则Pod将保持在Terminating状态。
- **Pod设置了terminationGracePeriodSeconds优雅退出时间**：Pod设置优雅退出时间后，结束Pod时会进入Terminating状态，等待容器优雅退出后将会自动删除。

解决方法

说明

强制删除Pod可能会产生数据不一致、业务容器异常退出等风险，尤其是强制删除StatefulSet的Pod，请您合理评估可能存在的业务风险后执行该操作。详情请参见[强制删除StatefulSet中的Pod](#)。

无论各种方式生成的Pod，均可以使用如下命令强制删除：

```
kubectl delete pod <pod> -n <namespace> --grace-period=0 --force
```

因此对于上面的Pod，只要执行如下命令即可删除：

```
kubectl delete pod aos-apiserver-5f8f5b5585-s9l92 -n aos --grace-period=0 --force
```

6.1.9 工作负载异常：已停止

问题现象

工作负载的状态为“已停止”。

问题原因：

工作负载的yaml中的metadata.enable字段为false，导致工作负载被停止，Pod被删除导致工作负载处于已停止状态，如下图所示：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
  namespace: default
  selfLink: /apis/apps/v1/namespaces/default/deployments/test
  uid: b130db9f-9306-11e9-a2a9-fa163eaff9f7
  resourceVersion: '7314771'
  generation: 1
  creationTimestamp: '2019-06-20T02:54:16Z'
  labels:
    appgroup: ''
  annotations:
    deployment.kubernetes.io/revision: '1'
    description: ''
  enable: false
spec:
```

解决方案

将enable字段删除或者将false修改为true。

6.1.10 工作负载异常：GPU 节点部署服务时报错

问题现象

在CCE集群的GPU节点上部署服务出现如下问题：

1. 容器无法查看显存。
2. 部署了7个GPU服务，有2个是能正常访问的，其他启动时都有报错。
 - 2个是能正常访问的CUDA版本分别是10.1和10.0
 - 其他服务CUDA版本也在这2个范围内
3. 在GPU服务容器中发现一些新增的文件core.*，在以前的部署中没有出现过。

问题定位

1. GPU插件的驱动版本较低，单独下载驱动安装后正常。
2. 工作负载中未声明需要gpu资源。

建议方案

节点安装了gpu-beta (gpu-device-plugin) 插件后，会自动安装nvidia-smi命令行工具。引起部署GPU服务报错通常是由于nvidia驱动安装失败，请排查nvidia驱动是否下载成功。

- GPU节点：
 - 插件版本为2.0.0以下时，请执行以下命令：

```
cd /opt/cloud/cce/nvidia/bin && ./nvidia-smi
```
 - 插件版本为2.0.0及以上时，请执行以下命令：

```
cd /usr/local/nvidia/bin && ./nvidia-smi
```
- 容器：
 - 当集群版本在1.27及以下时，请执行以下命令：

```
cd /usr/local/nvidia/bin && ./nvidia-smi
```
 - 当集群版本在1.28及以上时，请执行以下命令：

```
cd /usr/bin && ./nvidia-smi
```

若能正常返回GPU信息，说明设备可用，插件安装成功。

如果驱动地址填写错误，需要将插件卸载后重新安装，并配置正确的地址。

说明

nvidia驱动建议放在OBS桶里，并设置为公共读。

相关链接

- [GPU节点使用nvidia驱动启动容器排查思路](#)
- [GPU插件安装](#)

6.1.11 工作负载异常：添加存储失败

问题现象

实例一直处于创建中，事件中存在“添加存储失败”的告警，事件信息如下所示：

```
AttachVolume.Attach failed for volume "pvc-***" : rpc error: code = Internal desc = [***][disk.csi.everest.io] attaching volume *** to node *** failed: failed to send request of attaching disk(id=***) to node(id=***): error statuscode 400 for posting request, response is {"badRequest": {"message": "Maximum number of scsi disk exceeded", "code": 400}}, request is {"volumeAttachment": {"volumeId": "****", "device": "", "id": "", "serverId": "", "bus": "", "pciAddress": "", "VolumeWwn": "", "VolumeMultiAttach": false, "VolumeMetadata": null}}, url is: .....
```

问题原因

该告警事件说明节点上绑定的云硬盘已达上限，挂载云硬盘的工作负载实例调度到该节点后，无法继续挂载云硬盘，导致工作负载无法正常运行。

例如，假设节点可挂载的云硬盘上限为20，除去节点上已挂载的1块系统盘和1块数据盘后，节点剩余可挂载的云硬盘数量为18块。若该节点通过ECS控制台自定义挂载了2块裸盘用于创建本地存储池，则实际上仅能挂载16块数据盘。该场景下，如果该节点上调度18个挂载1块云硬盘的工作负载实例，就会出现2个工作负载实例由于节点上无法再挂载云硬盘，出现以上错误。

解决方案

当CCE 容器存储 (Everest)插件版本为2.3.11及以上时，插件提供 `number_of_reserved_disks`参数，该参数可以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。注意，修改该参数将对集群中所有节点生效。

设置 `number_of_reserved_disks`参数后，节点剩余可挂载的云硬盘数量计算如下：

节点剩余挂盘数 = 节点可挂载的云硬盘上限值 - `number_of_reserved_disks`

参数配置

请参考使用指南进行参数配置

```
{
  "cluster_id": "4e5fe06b-902c-11ee-9283-0255ac100043",
  "cluster_name": "test-api",
  "csi_attacher_detach_worker_threads": "60",
  "csi_attacher_worker_threads": "60",
  "default_vpc_id": "05e72ac9-556f-4017-9fe6-b3ed4caa1b05",
  "disable_auto_mount_secret": false,
  "enable_node_attacher": false,
  "flow_control": {},
  "number_of_reserved_disks": "6",
  "over_subscription": "80",
  "project_id": "b6315dd3d0ff4be5b31a963256794989",
  "volume_attaching_flow_ctrl": "0"
}
```

例如，假设节点可挂载的云硬盘上限为20，设置 `number_of_reserved_disks`参数为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。该场景下，如果集

群中需要调度18个挂载1块云硬盘的工作负载实例，则该节点最多可调度14个实例，剩余4个工作负载实例会调度到其他节点，不会出现节点存储无法挂载的情况。

6.1.12 工作负载异常：实例无法写入数据

Pod 事件

Pod所在的节点文件系统损坏，新建的Pod无法成功在/var/lib/kubelet/device-plugins/.xxxxx写入数据，Pod通常会出现以下类似事件：

Message: Pod Update Plugin resources failed due to failed to write checkpoint file
"kubelet_internal_checkpoint": open /var/lib/kubelet/device-plugins/.xxxxxx: read-only file system, which is unexpected.

```
[root@10-0-0-213 pass]# kubectl describe pod trunlport-test1-d84dc649-zxfpk
Name:          trunlport-test1-d84dc649-zxfpk
Namespace:     default
Priority:       0
Node:          10.0.0.77
Start Time:    Sat, 20 Feb 2021 16:43:35 +0800
app=trunlport-test1
pod-template-hash=d84dc649
version=v1
Annotations:   kubernetes.io/psp: psp-global
               metrics.alpha.kubernetes.io/custom-endpoints: [{"api":"","path":"","port":"","names":""}]
Status:        Failed
Reason:        UnexpectedAdmissionError
Message:       Pod Update plugin resources failed due to failed to write checkpoint file "kubernetes_internal_checkpoint": open /var/lib/kubelet/device-plugins/.762032416: read-only file system, which is unexpected.
IP:            10.0.0.10
IPs:           <none>
Controlled By: ReplicaSet/trunlport-test1-d84dc649
Containers:
  container-0:
    Image:      100.125.4.7:20202/cce-test/tomcat:latest
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:      250m
      memory:   512Mi
    Requests:
      cpu:      250m
      memory:   512Mi
    Environment:
```

此类异常Pod仅为异常记录，并不实际占用系统资源。

排查步骤

导致文件系统异常的原因有很多，例如物理控制节点的异常开关机。此类异常Pod并不影响正常业务，当系统文件未能恢复，出现大量异常Pod时，可采取以下步骤进行规避排查：

步骤1 执行以下命令，将该Node标记为不可调度，并将已有Pod驱逐到其他节点。

```
kubectl drain <node-name>
```

步骤2 等待Pod调度到其他节点后，排查文件系统异常的原因，并进行恢复或规避。

步骤3 执行以下命令，取消节点不可调度标记。

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

----结束

异常 Pod 清理

- 本服务kubelet的GC回收机制与社区保持一致，在清除Pod的Owner（例如Deployment）后，异常Pod也会随之清理。
- 通过kubelet命令，删除有异常记录的Pod。

6.1.13 工作负载异常：Init 容器启动失败

问题现象

- Pod的状态为Init:N/M。

- Pod的状态为Init:Error。
- Pod的状态为Init:CrashLoopBackOff。

问题原因

- Pod运行状态为Init:N/M，说明该Pod包含M个Init容器，其中N个已经启动完成，但仍有M-N个Init容器未启动成功。
- Pod运行状态为Init:Error，说明Pod中的Init容器启动失败。
- Pod运行状态为Init:CrashLoopBackOff，说明Pod中的Init容器启动失败并处于反复重启状态。

解决方案

1. 查看Pod的事件，确认当前Pod中未启动的Init容器是否存在异常。具体操作，请参见[Pod事件查看方法](#)。
2. 查看Pod中未启动的Init容器的日志，通过日志内容排查问题。具体操作，请参见[容器日志查看方法](#)。
3. 查看Pod的配置，确认未启动的Init容器配置是否正常。具体操作，请参见[Pod配置查看方法](#)。关于Init容器的更多信息，请参见[调试Init容器](#)。

6.1.14 工作负载异常：OOM 问题

问题现象

若因OOM被终止的进程为容器的阻塞进程，可能会导致容器异常重启。

问题原因

容器不允许使用超过其限制的内存，超过后容器可能会被终止，触发OOM（Out Of Memory）事件，导致容器异常退出。关于OOM事件，可以参考[为容器和Pod分配内存资源](#)。

解决方案

1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“工作负载”。
3. 单击工作负载操作列的“监控”，即可查看Pod的CPU、内存、网络I/O等监控大盘。



4. 通过Pod内存监控查看内存增长曲线，确定异常出现时间。
5. 根据监控、内存增长时间点、日志、进程名等信息，排查Pod内对应进程是否存在内存泄漏。
 - 若OOM是进程内存泄漏导致，请您自行排查泄露原因。

- 若进程运行状态正常，则根据实际运行需要，适当增大Pod的内存限制，建议Pod的内存实际使用量不超过内存限制值的80%。具体操作，请参见[设置容器规格](#)。

6.1.15 工作负载状态正常但未正常工作

问题现象

Pod已经处于Running状态（查看Pod状态）但未正常工作，或者访问结果不符合预期。

问题原因

可能是您的部署描述YAML文件（例如 Pod、Deployment、StatefulSet等）存在错误。例如：

- 镜像版本未更新。您可能未使用正确的镜像版本，或者是旧版镜像和新版镜像重名均为latest版本，而节点上已存在旧版镜像，但工作负载的imagePullPolicy设置为IfNotPresent，导致未重新拉取新版镜像仍然运行旧版镜像。
- 配置中的环境变量错误。例如将command拼写成commnd，仍然能够使用该YAML文件创建工作负载，但是容器在运行时，却不会使用您原本期望的命令，而是执行了镜像中的EntryPoint默认命令。

解决方案

1. 查看Pod的配置，确定Pod中容器的配置是否符合预期。具体操作，请参见[Pod配置查看方法](#)。
2. 排查环境变量中的某一个Key是否存在拼写错误时，您可以参考以下方法。以command拼写成commnd为例，说明拼写问题排查方法。
 - a. 在执行**kubectl apply -f**命令前，尝试为其添加--validate，然后执行**kubectl apply --validate -f XXX.yaml**命令。如果您将command拼写成commnd，将看到错误信息

```
I0805 10:43:25.129850 46757 schema.go:126] unknown field: commnd
I0805 10:43:25.129973 46757 schema.go:129] this may be a false alarm, see https://github.com/kubernetes/kubernetes/issues/6842/pods/mypod
```
 - b. 执行以下命令，将输出结果的pod.yaml文件与您创建Pod使用的文件进行对比。

```
kubectl get pods/$mypod -o yaml > mypod.yaml
```

*\$mypod*为异常Pod的名称，您可以通过**kubectl get pods**命令查看。

 - mypod.yaml文件比您创建时所使用的Pod文件多几行，说明已创建的Pod符合预期。
 - 如果mypod.yaml缺失创建Pod使用文件的部分代码行，说明您创建Pod使用的文件存在拼写问题。
3. 查看Pod的日志，通过日志内容排查问题。具体操作，请参见[容器日志查看方法](#)。
4. 可通过终端进入容器查看容器内的本地文件是否符合预期。具体操作，请参见[登录容器的方法](#)。

6.1.16 挂载文件存储的节点，Pod 创建删除卡死

故障现象

在挂载文件存储（SFS或SFS Turbo）的节点上，删除Pod卡在“结束中”，创建Pod卡在“创建中”。

可能原因

- 后端文件存储被删除，导致无法访问挂载点。
- 节点与文件存储间网络异常，导致无法访问挂载点。

解决方案

步骤1 登录挂载文件存储的节点，执行如下命令找到文件存储挂载路径。

findmnt

挂载点路径示例：/mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

步骤2 执行如下命令尝试进入文件存储文件夹。

cd /mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

如果不能正确进入，则说明文件存储被删除或文件存储与节点间网络异常。

步骤3 执行umount -l 命令解除挂载。

umount -l /mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

步骤4 重启kubelet。

systemctl restart kubelet

----结束

问题根因

该问题常见于文件存储挂载模式为hard的场景，在hard模式下，所有访问挂载点的进程都会Hang住，直到访问成功。使用soft模式挂载可以避免该情况，具体请参见[设置挂载参数](#)。

6.1.17 容器异常退出状态码

当容器启动失败或终止时，K8s事件中将会打印容器异常退出状态码（Exit Code）来报告容器异常的原因。本文将介绍如何通过事件中打印的Exit Code进一步定位容器异常的根本原因。

查看容器异常退出状态码

您可使用kubectl连接集群，并通过以下命令查询Pod详细状态：

```
kubectl describe pod {pod name}
```

在返回结果中的Exit Code字段即为程序上次退出时的状态码，该值不为0即表示程序异常退出，可根据退出状态码进一步分析异常原因。

```
Containers:
  container-1:
    Container ID: ...
    Image: ...
    Image ID: ...
    Ports: ...
    Host Ports: ...
    Args: ...
    State: Running
      Started: Sat, 28 Jan 2023 09:06:53 +0000
    Last State: Terminated
      Reason: Error
      Exit Code: 255
    Started: Sat, 28 Jan 2023 09:01:33 +0000
    Finished: Sat, 28 Jan 2023 09:05:11 +0000
    Ready: True
    Restart Count: 1
```

退出状态码说明

容器退出状态码的范围为0~255之间：

- 0表示正常退出。
- 一般由于程序自身原因导致的异常退出，状态码区间在1~128。在特殊场景下，程序也可以使用129~255区间的状态码。
- 由于外界中断导致程序退出时，状态码区间在129~255。当操作系统给程序发送[中断信号](#)时，程序退出时的状态码为操作系统的中断信号值加128。例如，SIGKILL（强制停止）的中断信号值为9，那么程序退出状态码则为 $9+128 = 137$ 。
- 当指定的状态码不在0~255范围内时（例如使用`exit(-1)`），将自动转化至0~255范围内。

当指定的状态码为正数，转换公式如下：

```
code % 256
```

当指定的状态码为负数，转换公式如下：

```
256 - (|code| % 256)
```

更多退出状态码说明请参见[Exit Codes With Special Meanings](#)。

常见的容器异常退出状态码

表 6-5 常见的容器异常退出状态码

容器异常退出状态码	名称	含义
0	正常退出	表示容器正常退出。该状态码不一定表示存在异常，例如，当容器内没有进程时，也会出现退出状态码为0的场景。
1	一般程序错误	引起该异常状态的原因较多，大多由于程序自身错误导致，需要进一步通过容器日志定位原因。例如在ARM节点上运行X86镜像时，会出现该错误。

容器异常退出状态码	名称	含义
125	容器未能运行	发生这种情况的常见原因有如下几种： - 命令中使用了未定义的flag，例如docker run --abcd。 - 镜像中用户定义的命令在本机权限不足。 - 容器引擎与宿主机操作系统或硬件不兼容。
126	命令调用错误	镜像中调用的命令无法执行，例如文件权限不足或文件不可执行。
127	找不到文件或目录	无法找到镜像中指定的文件或目录。
128	无效的退出参数	容器退出但未提供有效的退出代码，可能的原因有多种，需要进一步定位原因。例如containerd节点上运行的应用尝试调用docker命令。
137	立即终止（SIGKILL）	表示程序被SIGKILL信号终止，常见的原因有如下几种： - Pod中容器内存达到了其资源限制（resources.limits）。例如，内存溢出（OOM）会导致cgroup强制停止该容器。 - 运行容器的节点本身资源不足（OOM），则节点内核会选择停止一些进程来释放内存，可能会导致容器被终止。 - 容器健康检查失败，kubelet会停止该容器。 - 其他外部进程强制停止容器，例如恶意脚本。
139	分段错误（SIGSEGV）	表示容器收到了来自操作系统的SIGSEGV信号，由于容器试图访问无权限的内存位置引起。
143	优雅终止（SIGTERM）	表示容器在主机指示后正确关闭。一般来说，退出码143不需要进行故障排除。
255	状态码超出范围	表示容器退出状态码超出范围。例如，可能是设置异常退出使用exit(-1)导致的，而-1将会自动转换成255。 出现该异常时无法判断原因，需要进一步通过容器日志定位原因。

Linux 标准中断信号

您可以使用**kill -l**命令查看Linux操作系统中信号以及对应的数值。

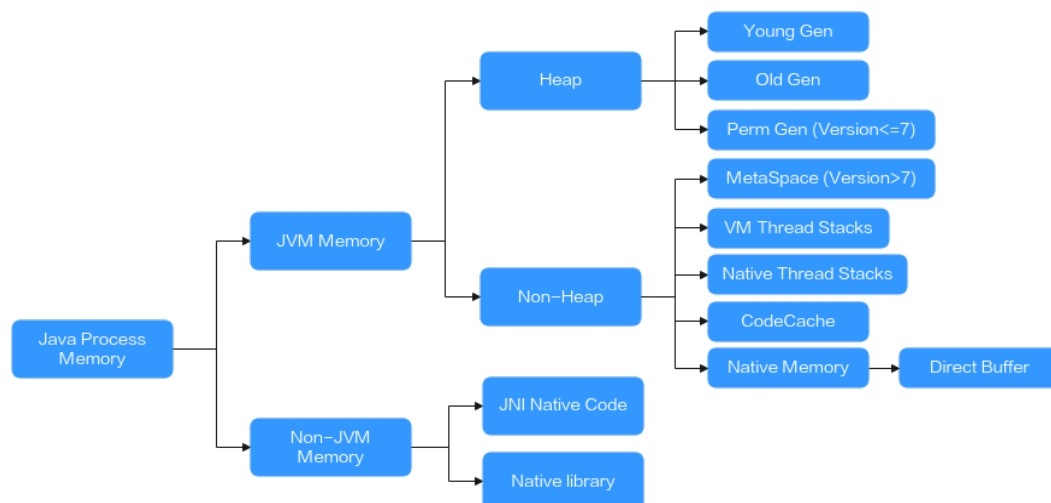
表 6-6 常用的 Linux 标准中断信号

信号 (Signal)	状态码 (Value)	动作 (Action)	描述 (Commit)
SIGHUP	1	Term	用户终端连接（正常或非正常）结束时发出
SIGINT	2	Term	程序终止信号，通常由终端发出中断指令，例如键盘输入Ctrl+C
SIGQUIT	3	Core	和SIGINT类似，由终端发出退出指令，通常是键盘输入Ctrl+\来控制
SIGILL	4	Core	非法指令，通常是因为可执行文件本身出现错误
SIGABRT	6	Core	调用abort函数时产生的信号，进程会非正常结束
SIGFPE	8	Core	发生浮点运算错误，出现除数为0等其它算术错误时也会产生
SIGKILL	9	Term	终止任何进程
SIGSEGV	11	Core	试图访问无权限的内存位置
SIGPIPE	13	Term	管道断开信号
SIGALRM	14	Term	时钟定时信号
SIGTERM	15	Term	进程结束信号，通常是程序自行正常退出
SIGUSR1	10	Term	用户在应用程序中自行定义的信号
SIGUSR2	12	Term	用户在应用程序中自行定义的信号
SIGCHLD	17	Ign	子进程结束或中断时产生该信号
SIGCONT	18	Cont	让一个暂停（stopped）的进程继续执行
SIGSTOP	19	Stop	暂停进程的执行
SIGTSTP	20	Stop	停止进程的运行
SIGTTIN	21	Stop	后台进程从终端读取输入值
SIGTTOU	22	Stop	后台进程从终端读取输出值

6.1.18 Java 容器内存虚高以及 OOM 问题定位

Java 进程内存相关背景

Java堆内存（Heap Memory）是Java虚拟机（JVM）管理的主要内存区域，但将Java堆内存限制参数Xmx并不能当作进程内存限制参数使用，将容器内存限制设置为和Xmx一样大，可能会导致OOM。除了堆内存，JVM还管理其他类型的内存，Java进程的内存占用情况可以简略总结为下图：



Java应用程序的内存使用情况包括JVM内存和非JVM内存。

- **JVM内存（JVM Memory）**：这是JVM管理的内存部分，可以进一步分为堆（Heap）和非堆（Non-Heap）内存。
 - 堆内存（Heap Memory）：堆是OOM故障最主要的发生区域，它是内存区域中最大的一块区域，被所有线程共享，存储着几乎所有的实例对象、数组。所有的对象实例以及数组都要在堆上分配。Java堆也是垃圾收集器管理的主要区域，因此很多时候也被称为GC堆。从内存回收的角度来看，由于现在收集器基本都采用分代收集算法，所以Java堆中还可以细分为新生代（Young Gen）和老年代（Old Gen）。
 - 非堆内存（Non-Heap Memory）：这部分内存不用于存储对象实例，而是用于存储类的元数据、线程信息等。
 - 元空间（MetaSpace）：在Java 8中，永久代（PermGen）被元空间代替，元空间的本质和永久代类似，都是对JVM规范中方法区的实现，不过元空间和永久代最大的差别在于元空间并不在虚拟机中，而是使用的本地内存，元空间的大小只受本地内存限制。
 - Java虚拟机线程栈（VM Thread Stacks）：对于每一个线程，JVM都会在线程被创建的时候，创建一个单独的栈。线程是私有的，除了本地方法以外，Java方法都是通过Java虚拟机栈来实现调用和执行过程的。
 - 本地方法栈（Native Thread Stacks）：与虚拟机栈所发挥的作用是非常相似的，它们之间的区别不过是虚拟机栈为虚拟机执行Java方法（也就是字节码）服务，而本地方法栈则为虚拟机使用到的本地方法服务。
 - 代码缓存（CodeCache）：JVM代码缓存是JVM将其字节码存储为本机代码的区域。一般情况下不关心这部分内存区域，如果这块区域OOM了，在日志里面就会看到`java.lang.OutOfMemoryErrorcode cache`。

- Native Memory: 是指在JVM堆内存 (heap memory) 以外的内存, 也会被叫做堆外内存。但它仍然属于这个Java程序的进程内存。通俗的说就是JVM管不到的原生内存。常见的是Java调用汇编/C/C++的时候, 汇编/C/C++那部分所占用的内存。
- 非JVM内存 (Non-JVM Memory) : 这是JVM之外的内存, 包括本地库和JNI本地代码。

须知

Java容器过程中, 尤其要注意-Xmx或者--XX:MaxRAMPercentage限制的是Heap空间, 而不是整个Java进程的空间。比如配置--XX:MaxRAMPercentage=70%, 只能限制最大Heap内存为70%*总内存, 但是由于非Heap内存也会占用内存空间, 最后导致整个Java进程内存可能超过预期。

容器内存指标说明

很多用户在虚拟机上部署Java业务时, 因为整个虚拟机都给整个Java业务使用, 所以很少关注Java业务整体内存使用问题; 而在通过容器化提高调度密度 (最终提高整体资源使用率) 的过程中, 绝大部分用户都会配置Memory Limit来控制单个容器 (每个容器都是通过内核cgroup限制资源上限) 的内存使用上限, 防止某个容器使用过多内存从而影响本节点其他容器。由于内存是不可压缩资源, 一旦超过Limit限制则会触发操作系统OOM导致容器重启, 所以业务容器化后需要关注容器内存使用率指标。

- 操作系统以memory_limit_in_bytes值作为某cgroup可使用内存上限, 当某个cgroup内存使用总量memory_usage_in_bytes即将超过memory_limit_in_bytes时, 操作系统将尝试回收内存, 如果可以回收且回收后低于memory_limit_in_bytes, 则不会触发OOM, 否则则触发OOM。
memory_usage_in_bytes计算方式如下:

memory_usage_in_bytes = 匿名内存 (inactive_anon + active_anon) + file 内存 (inactive_file+active_file)

- active_anon: 活跃的LRU列表中匿名和swap缓存的字节数, 包括tmpfs。
- inactive_anon: 不活跃的LRU列表中匿名和swap缓存的字节数, 包括tmpfs。
- active_file: 活跃LRU列表中文件支持的 (file-backed) 的内存字节数。
- inactive_file: 不活跃列表中文件支持的 (file-backed) 的内存字节数。

由于Kubernetes暂时不支持容器中开启swap内存, 所以可以简单理解容器业务中匿名内存不可回收, file内存 (主要包括应用自己管理的内存映射文件mmap、文件读写缓存、二进制、动态链接库等) 可以被回收。当触发回收时优先将inactive_file对应的文件内容写回磁盘后回收对应内存 (由于active_file列表中为最近访问过文件的内存, 所以尽量不回收active_file对应文件内存) 。

- 云原生监控均以容器WorkingSet指标作为容器内存指标:

WorkingSet内存 = 匿名内存 (inactive_anon + active_anon) + active_file

Java 容器内存虚高定位思路

通过以下配置示例分析Java进程在容器中的使用场景:


```
command:
- java
- '-Dapp.name=pv-insurance'
- '-XX:InitialRAMPercentage=30.0'
- '-XX:MinRAMPercentage=30.0'
- '-XX:MaxRAMPercentage=70.0'
- '-XX:MaxMetaspaceSize=512m'
- '-Xss256k'
- '-XX:+UseContainerSupport'
- '-XX:-OmitStackTraceInFastThrow'
- '-XX:+HeapDumpOnOutOfMemoryError'
- '-XX:HeapDumpPath=./dump'
- '-XX:+PrintGCDetails'
- '-XX:+UseG1GC'
- '-Xloggc:./logs/gc.log'
- '-verbose:gc -XX:+PrintGCDateStamps'
- '-Djava.awt.headless=true'
- '-Djava.security.egd=file:/dev/./urandom'
- '-Djava.io.tmpdir=/opt/deploy/temp'
- '-DtraceId.save=true'
- '-Dlog_dir=./logs'
```

在配置示例中，可以看到以下关键参数：

- -XX:MaxRAMPercentage=70.0：表示最大堆内存为容器内存限制的70%
- -XX:MaxMetaspaceSize=512m：表示元空间内存占用触发FGC的阈值为512M
- -Xss256k：表示为每个线程分配256k的内存
- -XX:+UseG1GC：表示使用G1 GC作为垃圾收集器

在该示例中，容器内存Limit为6G，假设容器内存实际使用率偏高，可通过以下步骤进行排查：

1. 确认是WorkingSet内存高还是真实内存高

登录目标容器，运行以下命令查看内存统计信息：

```
cat /sys/fs/cgroup/memory/memory.stat
```

查看total_cache（缓存内存量）、total_rss（当前应用进程实际使用内存量）、total_inactive_file（不活跃文件内存使用量）。

则WorkingSet内存 = total_cache + total_rss - total_inactive_file

假设memory.stat文件内容如下：

```
rss 1048576
cache 524288
inactive_file 262144
```

则WorkingSet内存为：

```
WorkingSet = 1048576 + (524288 - 262144) = 1310720 bytes
```

由于标准云原生监控都是采集WorkingSet内存，当看到容器内存高时，先确认容器WorkingSet内存和匿名内存使用率，如果active_file内存过高，结合[容器内存指标说明](#)，需要分析代码是否频繁读写文件、读取大文件或者有内存映射文件mmap。

如果WorkingSet和匿名内存基本上重合，则重点需要分析JVM内存构成、GC等判断影响。

2. 分析Java业务内存分配和GC

- a. 进入业务容器查看Java最大实际堆内存大小

进入容器业务容器使用-XshowSettings:vm -version选项查看JVM的设置，确认Java进程heap最大堆内存配置是否符合预期。

本例中容器内存Limit为6G，Java启动参数-XX:MaxRAMPercentage=70%，则最大heap内存为4.2G时符合预期。

- b. 分析查看GC日志，根据上述启动参数-Xloggc:…/logs/gc.log查看GC日志。由于Java业务内存主要由JVM回收，通过分析GC日志主要是jvm是否回收是符合预期（比如是否有fullGC等问题）。
- c. 通过Java自带的Native Memory Tracking（NMT）分析内存JVM内存占用情况

- i. 首先需要在Java进程的启动参数中添加-**XX:NativeMemoryTracking=detail**

Java 启动时先打开 NativeMemoryTracking，默认是关闭的。注意不要在生产环境长期开启，有性能损失

```
java -XX:NativeMemoryTracking=detail -jar
```

- ii. 然后执行命令 **jcmd pid VM.native_memory summary**，其中pid为指定进程号，查看内存分配信息。

查看详情

```
jcmd pid VM.native_memory summary
```

```
Total: reserved=6562952KB, committed=5458596KB
-   Java Heap (reserved=4485248KB, committed=4485248KB)
    (mmap: reserved=4485248KB, committed=4485248KB)
-   Class (reserved=1278953KB, committed=262845KB)
    (classes #33311)
    (malloc=7145KB #89788)
    (mmap: reserved=1271888KB, committed=254988KB)
-   Thread (reserved=185951KB, committed=185951KB)
    (thread #184)
    (stack: reserved=185892KB, committed=185892KB)
    (malloc=643KB #1898)
    (arena=215KB #364)
-   Code (reserved=282867KB, committed=284825KB)
    (malloc=33267KB #41888)
    (mmap: reserved=249688KB, committed=178756KB)
-   GC (reserved=242325KB, committed=242325KB)
    (malloc=46881KB #138846)
    (mmap: reserved=196244KB, committed=196244KB)
-   Compiler (reserved=699KB, committed=699KB)
    (malloc=588KB #2678)
    (arena=151KB #6)
-   Internal (reserved=188623KB, committed=188623KB)
    (malloc=188591KB #127982)
    (mmap: reserved=32KB, committed=32KB)
-   Symbol (reserved=37885KB, committed=37885KB)
    (malloc=33289KB #351145)
    (arena=5876KB #1)
-   Native Memory Tracking (reserved=12486KB, committed=12486KB)
    (malloc=552KB #7898)
    (tracking overhead=11853KB)
```

Java进程所占的内存项有：Java Heap、Class、Thread、Code等对应内存空间。其中Total committed内存大约为：5.2G，但是Java Heap committed内存就占了大约4.2G，因此Pod使用率高主要还是因为Java堆内存使用较多。

- d. **heap dump分析**

通过jmap获取Java进程的dump内存数据，分析dump后的bin文件，是否符合预期。

执行命令**jmap -dump:format=b,file=/xxx/dump.hprof pid**（注意加live参数会触发GC，命令如下**jmap -dump:live,format=b,file=dump.hprof pid**）

通过分析dump bin文件，可以看到在Java进程中未被GC回收的内存数据有数据库查询的大对象、大数组等信息。该部分为堆内存数据，如果不触发Full

GC不会进行内存回收。建议部分SQL语句可以进行优化，非必要字段不进行查询。

综上所述分析和大规模测试过程结果可以得出以下结论：

1. Java业务在压测过程中 `-XX:MaxRAMPercentage`配置为70%生效，对heap内存上限设置在4G左右时业务压测均正常无问题，heap内存主要存储缓存数据库查询的结果，可以被JVM主动回收。
2. Java业务在压测过程中no-heap内存诉求大约在1G左右。
3. 因此云原生监控内存高的主要原因为：`-XX:MaxRAMPercentage`配置为70%，容器Limit配置为6G，上述配置导致Java总内存约为heap内存4.2G+non-heap内存1G共5.2G，因此云原生监控内存使用率约为 $5.2/6=87\%$ 。

问题处理：

建议调整Java进程启动参数将`-XX:MaxRAMPercentage`参数由70.0%设置为50.0%，同时调整`memory.Limits`参数由6G扩大至8G，如此保证heap内存达到最大诉求的同时，给non-heap内存预留足够空间。

附录：JVM 参数配置说明

在Java进程可以通过一些启动参数可以堆栈内存的使用。下面将对JDK8以后版本中的常用参数进行介绍。

Java堆栈内存相关参数

Java堆内存是Java虚拟机所管理的内存中最大的一块，是所有线程共享的一块内存区域，在虚拟机启动时创建。Java堆内存区域唯一目的就是存放对象实例，几乎所有的对象实例和数组都在这里分配内存。

- `-Xms`：设置最小堆内存，并初始化堆内存大小。单位可以是k/K、m/M、g/G，最小值为1M；例如`-Xms6m`。
- `-Xmx`：设置最大堆内存。一般建议与`-Xms`保持一致，防止动态扩展。
- `-Xmn`：设置年轻代内存最大允许大小，并初始化年轻代内存大小。官方建议该区域大小一般为堆内存的1/2-1/4之间。设置太小会导致频繁发生Minor GC，设置过大只有full GC才会生效，fullGC完成时间一般较久。例如`-Xmn2G`。
- `-XX:NewSize`：设置年轻代内存初始大小。
- `-XX:MaxNewSize`：设置年轻代最大内存大小。
- `-XX:MetaspaceSize`：设置元空间的大小（使用的是本地内存），超出时将触发FGC。`MetaspaceSize`表示使用过程中触发GC的阈值。
- `-XX:MaxMetaspaceSize`：设置元空间的最大值（使用的是本地内存）。例如`-XX:MaxMetaspaceSize=256m`。
- `-Xss`：设置线程栈的大小。Linux/x64平台下默认值是1024kb。例如`-Xss1024k`。

GC相关参数

这些参数控制JVM如何执行垃圾回收策略：

- `-XX:+UseG1GC`：启用G1（garbage first）算法的垃圾回收策略。对于堆内存分配大小大于6G，且GC延迟要求低的应用程序建议使用G1收集器。
- `-XX:+PrintGCDetails`：在每次发生GC时打印GC详细信息，默认关闭该参数。
- `-XX:+PrintGCDateStamps`：在每次发生GC时打印日期戳。

- -Xloggc: gc.log: GC日志文件的输出路径。

OOM相关参数

- -XX:+HeapDumpOnOutOfMemoryError: 当JVM抛出 java.lang.OutOfMemoryError异常时, 将heap转储到物理文件中。
- -XX:HeapDumpPath=/var/log/xxx/xx.hprof: 写入dump文件的路径, 使用.hprof格式。

容器场景下相关参数

因为容器场景下, 容器的资源分配是动态可调的, 如果每次调整完容器的资源配额, 然后再手动设置最大堆内存, 比较繁琐。可以通过以下参数动态调整最大堆内存大小。

- -XX:+UseContainerSupport: 默认启用容器支持。JVM能够自动进行容器平台检测, 能够确定容器中运行的Java进程可用的内存量和CPU处理器。
- -XX:MaxRAMPercentage=xxx: 设置Java堆内存的最大百分比。例如, 设置为-XX:MaxRAMPercentage=50, 意味着Java堆内存最大不超过容器可用内存的50%。适用于容器场景下资源限制量动态变化, 动态调节最大堆内存大小, 等同于-Xmx参数。
- -XX:MinRAMPercentage=xxx: 设置Java堆内存的最小百分比, 当容器的内存资源限制较低时, 确保堆内存的最低分配。当容器Memory资源Limits大小为250MB及以下时, 使用该参数计算最大堆内存。

6.1.19 集群中大量 Pod 处于 UnexpectedAdmissionError 状态怎么办

问题描述

查询集群中的Pod时, 发现大量Pod实例处于UnexpectedAdmissionError状态。例如, 执行kubectl get pod -A时, 回显结果如下:

NAME	READY	STATUS	RESTARTS	AGE
aos-apiserver-5f8f5b5585-s9l92	0/1	UnexpectedAdmissionError	0	3d1h
aos-cmdbserver-789bf5b497-6rwrq	0/1	UnexpectedAdmissionError	0	3d1h
aos-controller-545d78bs8d-vm6j9	0/1	UnexpectedAdmissionError	3	3d1h
...				

问题根因

Pod被调度器调度到某个节点上后, 节点资源不满足Pod申请的资源 (CPU、Memory、异构资源) 等, 导致该Pod会被Kubelet拒绝, 进入终态Failed状态。

解决方案

处于UnexpectedAdmissionError状态的Pod不会被立即清理。当集群中处于终态 (包括Completed和UnexpectedAdmissionError状态) 的Pod数量超过1000时, 系统将进行集中清理。终态Pod占用的资源已被释放, 查询时仅会显示其状态记录, 便于后续问题排查和定位。若无需保留此类Pod, 可直接手动删除。

6.1.20 Pod 实例异常，显示无法找到设备文件

问题现象

Pod实例创建失败，报错信息如下：

```
Error: failed to generate container "af736..." spec: failed to apply OCI options: lstat /dev/davinci4: no such file or directory
```

同时，执行以下命令，查看PCIE总线的数量。

```
lspci | grep -i accelerator | wc -l
```

当回显结果小于预期数目时，可以判断发生了PCIE断链。

问题原因

昇腾Snt9设备发生PCIE断链后，设备驱动未能上报准确的掉卡信息，导致任务仍被调度至断链的卡。

您可以执行以下命令，进一步验证PCIE是否断链：

```
npu-smi info -m
```

回显结果如下：

NPU ID	Chip ID	Chip Logic ID	Chip Name
0	0	0	Ascend xxx
0	1	-	Mcu
1	0	1	Ascend xxx
1	1	-	Mcu
2	0	2	Ascend xxx
2	1	-	Mcu
3	0	3	Ascend xxx
3	1	-	Mcu
5	0	4	Ascend xxx
5	1	-	Mcu
6	0	5	Ascend xxx
6	1	-	Mcu
7	0	6	Ascend xxx
7	1	-	Mcu

由NPU ID可知，昇腾Snt9设备中的第4张丢失，即发生PCIE断链。在此过程中，设备驱动对NPU卡的Chip Logic ID进行了重排，Chip Logic ID 7丢失。CCE AI套件（Ascend NPU）插件上报信息时只会更新Chip Logic ID，不会更新Chip Logic ID与NPU ID的对应关系，因此Kubernetes会认为原有的Chip Logic ID 7对应的第7张NPU卡发生故障，无法参与调度，而原Chip Logic ID 4所对应的第4张NPU卡依然被调度器识别为可用资源。

解决方案

升级Snt9设备的驱动版本到24.1.rc2及以上版本即可。新版本的驱动不会在PCIE断链后进行Chip Logic ID重排，可以上报正确的掉卡情况。

升级驱动后，执行以下命令，验证以上结论。

```
npu-smi info -m
```

回显结果如下，可以发现在PCIE断链后，新版本驱动在PCIE断链后没有对Chip Logic ID进行重排。

NPU ID	Chip ID	Chip Logic ID	Chip Name
-1	0	-	Mcu

0	0	0	Ascend xxx
0	1	-	Mcu
1	0	1	Ascend xxx
1	1	-	Mcu
2	0	2	Ascend xxx
2	1	-	Mcu
3	0	3	Ascend xxx
3	1	-	Mcu
5	0	5	Ascend xxx
5	1	-	Mcu
6	0	6	Ascend xxx
6	1	-	Mcu
7	0	7	Ascend xxx
7	1	-	Mcu

6.2 容器设置

6.2.1 在什么场景下设置工作负载生命周期中的“停止前处理”？

问题描述：

在什么场景下设置工作负载生命周期中的“停止前处理”？

问题解答：

服务的业务处理时间较长，在升级时，需要先等Pod中的业务处理完，才能kill该Pod，以保证业务不中断的场景。

6.2.2 在什么场景下，容器会被重建？

问题描述

在什么场景下，容器会被重建？

可能原因

容器重建是指容器被销毁并重新创建一个新实例。容器重建可能由多种原因触发，以下是一些常见的场景：

表 6-7 容器重建的常见场景

常见场景	说明
容器崩溃或异常终止	当运行中的容器因软件错误、资源耗尽或其他意外情况崩溃时，为了确保服务的连续性，系统会自动重建容器以迅速恢复服务。
手动删除某容器	若手动删除了某个正在运行的容器，容器编排工具会根据定义的部署策略重新调度并重建该容器，以保持预期的实例数量。
Pod优先级抢占	当高优先级的Pod需要资源时，Kubernetes可能会驱逐低优先级的Pod，低优先级Pod可能会被重新调度和启动。
配置更新	更新Deployment或StatefulSet的相关配置（如镜像版本、环境变量、数据存储等）时，通常会触发滚动更新，进一步导致现有容器被逐步销毁并重新创建。

常见场景	说明
节点资源不足	当节点资源（如内存、CPU等）不足时，集群可能会驱逐部分Pod并将其调度到其他资源充足节点，从而触发容器重建。
节点重启或故障	若由于某些原因重启节点，节点上的容器可能会被销毁并在其他可用节点上重建。当节点发生故障时，集群会检测到该节点不可用，该节点上的容器将在其他可用节点上进行重建。

使用[数据存储](#)（如挂载云硬盘、文件系统等）可以有效解决容器重建导致的数据丢失问题，确保重要数据实现持久化存储并能够在容器重建后继续使用。

6.2.3 在同一个命名空间内访问指定容器的 FQDN 是什么？

问题背景

客户询问在创建负载时指定部署的容器名称、pod名称、namespace名称，在同一个命名空间内访问该容器的FQDN是什么？

全限定域名：FQDN，即Fully Qualified Domain Name，同时带有主机名和域名的名称。（通过符号“.”）

例如：主机名是bigserver，域名是mycompany.com，那么FQDN就是：bigserver.mycompany.com。

问题建议

方案一：发布服务使用域名发现，需要提前预置好主机名和命名空间，服务发现使用域名的方式，注册的服务的域名为：服务名.命名空间.svc.cluster.local。这种使用有限制，注册中心部署必须容器化部署。

方案二：容器部署使用主机网络部署，然后亲和到集群的某一个节点，这样可以明确知道容器的服务地址（就是节点的地址），注册的地址为：服务所在节点IP，这种方案可以满足注册中心利用VM部署，缺陷是使用主机网络效率没有容器网络高。

6.2.4 健康检查探针（Liveness、Readiness）偶现检查失败？

健康检查探针偶现检测失败，是由于容器内的业务故障所导致，您需要优先定位自身业务问题。

常见情况有：

- 业务处理时间长，导致返回超时。
- tomcat建链和等待耗费时间太长（连接数、线程数等），导致返回超时。
- 容器所在节点，磁盘IO等性能达到瓶颈，导致业务处理超时。

6.2.5 如何设置容器 umask 值？

问题描述

tailf /dev/null的方式启动容器，然后手动执行启动脚本的方式得到的目录的权限是700，而不加tailf由Kubernetes自行启动的方式得到的目录权限却是751。

解决方案

这个问题是因为两种方式设置的umask值不一样，所以创建出来的目录权限不相同。

umask值用于为用户新创建的文件和目录设置缺省权限。如果umask的值设置过小，会使群组用户或其他用户的权限过大，给系统带来安全威胁。因此设置所有用户默认的umask值为0077，即用户创建的目录默认权限为700，文件的默认权限为600。

可以在启动脚本里面增加如下内容实现创建出来的目录权限为700：

1. 分别在/etc/bashrc文件和/etc/profile.d/目录下的所有文件中加入“umask 0077”。

2. 执行如下命令：

```
echo "umask 0077" >> $FILE
```

说明

FILE为具体的文件名，例如：echo “umask 0077” >> /etc/bashrc

3. 设置/etc/bashrc文件和/etc/profile.d/目录下所有文件的属主为：root，群组为：root。

4. 执行如下命令：

```
chown root.root $FILE
```

6.2.6 CCE 启动实例失败时的重试机制是怎样的？

CCE是基于原生Kubernetes的云容器引擎服务，完全兼容Kubernetes社区原生版本，与社区最新版本保持紧密同步，完全兼容Kubernetes API和Kubectl。

在Kubernetes中，Pod的spec中包含一个restartPolicy字段，其取值包括：Always、OnFailure和Never，默认值为：Always。

- Always：当容器失效时，由kubelet自动重启该容器。
- OnFailure：当容器终止运行且退出不为0时（正常退出），由kubelet自动重启该容器。
- Never：不论容器运行状态如何，kubelet都不会重启该容器。

restartPolicy适用于Pod中的所有容器。

restartPolicy仅针对同一节点上kubelet的容器重启动作。当Pod中的容器退出时，kubelet 会按指数回退方式计算重启的延迟（10s、20s、40s...），其最长延迟为5分钟。一旦某容器执行了10分钟并且没有出现问题，kubelet对该容器的重启回退计时器执行重置操作。

每种控制器对Pod的重启策略要求如下：

- Replication Controller（RC）和DaemonSet：必须设置为Always，需要保证该容器的持续运行。
- Job：OnFailure或Never，确保容器执行完成后不再重启。

6.3 监控日志

6.3.1 工作负载的“事件”保存多长时间？

在1.7.3-r12、1.9.2-r3及以上版本的集群中，工作负载的“事件”信息保存时间为1个小时，1小时后自动清除数据。

在1.7.3-r12之前更老的集群版本中，保存时间为24小时。

6.3.2 容器监控的内存使用率与实际弹性伸缩现象不一致

问题现象

容器监控的内存使用率与实际弹性伸缩现象不一致，例如容器内存使用率在界面上显示为40%左右，而HPA设置缩容阈值为70%，但界面上显示的内存使用率低于HPA阈值后并没有发生缩容。

问题根因

界面上显示的容器内存使用率与HPA弹性伸缩的内存使用率在计算方式上存在差异：

- 界面上显示的容器内存使用率计算方式为：**container_memory_rss**/内存Limit
container_memory_rss（即Resident Set Size，RSS）包含了部分可能并不活跃或未被有效利用的内存部分。
- HPA对于内存使用率弹性伸缩的计算方式为：
container_memory_working_set_bytes/内存Request
container_memory_working_set_bytes（即Working Set Size，WSS）的计算方式如下：

在Pod中执行`cat /sys/fs/cgroup/memory/memory.stat`，得到total_cache（缓存内存量）、total_rss（当前应用进程实际使用内存量）、total_inactive_file（不活跃文件内存使用量）。

$$WSS = total\_cache + total\_rss - total\_inactive\_file$$

如果您的应用存在以下情况，均可能导致HPA的扩容行为与预期不符，出现界面上显示的内存使用率低于HPA缩容阈值后并没有发生缩容，或者界面上显示的内存使用率未高于HPA扩容阈值但发生扩容等现象。

- 应用缓存占用非常高，WSS明显大于RSS，导致界面上显示的容器内存使用率小于HPA计算的内存使用率。
- Limit与Request配置差异较大时，Request明显小于Limit，导致界面上显示的容器内存使用率小于HPA计算的内存使用率。

6.4 调度策略

6.4.1 如何让多个 Pod 均匀部署到各个节点上？

Kubernetes中kube-scheduler组件负责Pod的调度，对每一个新创建的 Pod 或者是未被调度的 Pod，kube-scheduler 会选择一个最优的节点去运行这个 Pod。kube-scheduler 给一个 Pod 做调度选择包含过滤和打分两个步骤。过滤阶段会将所有满足 Pod 调度需求的节点选出来，在打分阶段 kube-scheduler 会给每一个可调度节点进行优先级打分，最后kube-scheduler 会将 Pod 调度到得分最高的节点上，如果存在多个得分最高的节点，kube-scheduler 会从中随机选取一个。

打分优先级中节点调度均衡（BalancedResourceAllocation）只是其中一项，还有其他打分会导致分布不均匀。详细的调度说明请参见[Kubernetes 调度器](#)和[调度策略](#)。

想要让多个Pod尽可能的均匀分布在各个节点上，可以考虑使用工作负载反亲和特性，让Pod之间尽量“互斥”，这样就能尽量均匀的分布在各节点上。

示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      affinity:
        podAntiAffinity:
          # 工作负载反亲和
          preferredDuringSchedulingIgnoredDuringExecution:
            # 尽量满足如下条件
            - 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:
          labelSelector:
            # 选择Pod的标签，与工作负载本身反亲和
            matchExpressions:
              - key: app
                operator: In
                values:
                  - nginx
          namespaces:
            - default
          topologyKey: kubernetes.io/hostname # 在节点上起作用
      imagePullSecrets:
        - name: default-secret
```

6.4.2 如何避免节点上的某个容器被驱逐？

问题背景

在工作负载调度时可能会发生一个节点上的两个容器之间互相争资源的情况，最终导致kubelet将其全部驱逐。那么能不能设定策略让其中一个服务一直保留？如何设定？

问题建议

Kubelet会按照下面的标准对Pod的驱逐行为进行评判：

- 根据服务质量：即[BestEffort](#)、[Burstable](#)、[Guaranteed](#)。
- 根据Pod调度请求的被耗尽资源的消耗量。

接下来，Pod按照下面的顺序进行驱逐（QOS）：

BestEffort -> Burstable -> Guaranteed

- BestEffort类型的Pod：系统用完了全部内存时，该类型Pod会最先被终止。
- Burstable类型的Pod：系统用完了全部内存，且没有BestEffort容器可以终止时，该类型Pod会被终止。
- Guaranteed类型的Pod：系统用完了全部内存、且没有Burstable与BestEffort容器可以终止时，该类型的Pod会被终止。

说明

- 如果Pod进程因使用超过预先设定的限制值而非Node资源紧张情况，系统倾向于在其原来所在的机器上重启该容器。
- 如果资源充足，可将QoS Pod类型均设置为Guaranteed。用计算资源换业务性能和稳定性，减少排查问题时间和成本。
- 如果想更好的提高资源利用率，业务服务可以设置为Guaranteed，而其他服务根据重要程度可分别设置为Burstable或BestEffort，例如filebeat。

6.4.3 为什么 Pod 在节点不是均匀分布？

Kubernetes 中的 Pod 调度原理

Kubernetes中kube-scheduler组件负责Pod的调度，对每一个新创建的 Pod 或者是未被调度的 Pod，kube-scheduler 会选择一个最优的节点去运行这个 Pod。kube-scheduler 给一个 Pod 做调度选择包含过滤和打分两个步骤。过滤阶段会将所有满足 Pod 调度需求的节点选出来，在打分阶段 kube-scheduler 会给每一个可调度节点进行优先级打分，最后kube-scheduler 会将 Pod 调度到得分最高的节点上，如果存在多个得分最高的节点，kube-scheduler 会从中随机选取一个。

打分优先级中节点调度均衡（BalancedResourceAllocation）只是其中一项，还有其他打分会导致分布不均匀。详细的调度说明请参见[Kubernetes 调度器](#)和[调度策略](#)。

为什么 Pod 数量在节点上分布不均匀

- 资源需求：不同节点的资源配置可能不同，例如CPU、内存大小，导致Pod中定义的Request值无法被满足。即使节点实际负载很低，也无法调度到该节点。
- 自定义调度策略：Pod可能根据自定义的亲亲和性和反亲和性策略进行调度，导致Pod在节点上分布不均匀。
- 节点污点和容忍度：节点存在某些污点，未设置容忍度的Pod无法调度到该节点上运行。
- 部分工作负载特性导致：工作负载可能具有特殊的分布约束，例如工作负载挂载某个可用区的云硬盘时只能调度到相同可用区的节点上。
- 节点特殊资源：部分Pod可能请求特殊的资源类型，例如GPU等资源，调度器只能将其调度到GPU类型的节点上。
- 节点健康状况：节点的健康状况和状态可能影响调度决策，不健康的节点可能不会调度新的Pod。

为什么 Pod 实际负载在节点上分布不均匀

kube-scheduler调度器在分配Pod时不会考虑应用的实际负载，如果应用负载不均匀可能导致某些节点的负载较高，而其他节点的负载较低。

Volcano调度器支持使用负载感知调度功能，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。详情请参见[负载感知调度](#)。

6.4.4 如何驱逐节点上的所有 Pod？

您可使用**kubectl drain**命令从节点安全地逐出所有Pod。

📖 说明

默认情况下，**kubectl drain**命令会保留某些系统级Pod不被驱逐，例如everest-csi-driver。

步骤1 使用kubectl连接集群。

步骤2 查看集群中的节点。

```
kubectl get node
```

步骤3 选择一个节点，查看节点上存在的所有Pod。

```
kubectl get pod --all-namespaces -owide --field-selector spec.nodeName=192.168.0.160
```

驱逐前该节点上的Pod如下：

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED NODE	READINESS GATES				
default	nginx-5bcc57c74b-lgcvh	1/1	Running	0	7m25s	10.0.0.140
192.168.0.160	<none>	<none>				
kube-system	coredns-6fcd88c4c-97p6s	1/1	Running	0	3h16m	10.0.0.138
192.168.0.160	<none>	<none>				
kube-system	everest-csi-controller-56796f47cc-99dtm	1/1	Running	0	3h16m	10.0.0.139
192.168.0.160	<none>	<none>				
kube-system	everest-csi-driver-dpfzl	2/2	Running	2	12d	192.168.0.160
192.168.0.160	<none>	<none>				
kube-system	icagent-tpfpv	1/1	Running	1	12d	192.168.0.160
192.168.0.160	<none>	<none>				

步骤4 驱逐该节点上的所有Pod。

```
kubectl drain 192.168.0.160
```

如果节点上存在绑定了本地存储的Pod或是一些守护进程集管理的Pod，将提示“error: unable to drain node "192.168.0.160" due to error: cannot delete DaemonSet-managed Pods...”。驱逐命令将不会生效，您可在上述命令后面添加如下参数进行强制驱逐：

- **--delete-emptydir-data**：强制驱逐节点上绑定了本地存储的Pod，例如coredns。
- **--ignore-daemonsets**：忽略节点上的守护进程集Pod，例如everest-csi-driver。

示例中节点上存在绑定本地存储的Pod和守护进程集Pod，因此驱逐命令如下：

```
kubectl drain 192.168.0.160 --delete-emptydir-data --ignore-daemonsets
```

步骤5 驱逐成功后，该节点被自动标记为不可调度，即该节点将会被打上**node.kubernetes.io/unschedulable = : NoSchedule**的污点。

驱逐后该节点上的Pod如下，节点上仅保留了不可驱逐的系统级Pod。

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READINESS GATES						
kube-system	everest-csi-driver-dpfzl	2/2	Running	2	12d	192.168.0.160	192.168.0.160
<none>	<none>						
kube-system	icagent-tpfpv	1/1	Running	1	12d	192.168.0.160	192.168.0.160
<none>	<none>						

----结束

相关操作

kubectl的drain、cordon和uncordon操作：

- drain：从节点安全地逐出所有Pod，并将该节点标记为不可调度。
- cordon：将节点标记为不可调度，即该节点将会被打上`node.kubernetes.io/unschedulable = : NoSchedule`的污点。
- uncordon：将节点标记为可调度。

更多说明请参考[kubectl文档](#)。

6.4.5 如何查看 Pod 是否使用 CPU 绑核？

以4U8G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":"static","defaultCpuSet":"0,2-3","entries":{"c1fcd22d-8a83-4aef-a27a-4c037e482b16":  
{"container-1":"1"},"checksum":"1500530529"}
```

policyName字段值为static代表策略设置成功。

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/pod{pod uid}/{容器id}/cpuset.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}'
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/podc1fcd22d-8a83-4aef-  
a27a-4c037e482b16/5cb15f5f429e4496172bef05994477caa96e0ca468563208695c1ad5cc141e0/  
cpuset.cpus
```

回显如下，表示绑定1号CPU。

```
1
```

----结束

6.4.6 节点关机后 Pod 不重新调度

问题现象

节点关机后，节点上的Pod仍然显示running状态。通过**kubectl describe pod <pod-name>**命令查询Pod最新事件为：

```
Warning NodeNotReady 17s node-controller Node is not ready
```

问题原因

节点关机后，系统会自动给节点添加污点，比如：

- node.kubernetes.io/unreachable:NoExecute
- node.cloudprovider.kubernetes.io/shutdown:NoSchedule
- node.kubernetes.io/unreachable:NoSchedule
- node.kubernetes.io/not-ready:NoExecute

当Pod对这些污点存在容忍策略时，Pod不会进行重新调度，因此需要检查Pod对污点的容忍策略。

解决方案

通过查询Pod或者工作负载的yaml，查看容忍策略。一般情况下，工作负载的容忍度设置由以下字段组成：

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
```

或者：

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

当上述容忍度的设置填写错误时，可能会出现调度问题。例如以下的容忍策略：

```
tolerations:
- operator: "Exists"
```

上述例子中只填写了operator参数为Exists（此时容忍度不能指定value参数）。

- 当一个容忍度的operator参数为Exists但key为空时，表示这个容忍度与任意的key、value和effect都匹配，即这个容忍度能容忍任何污点。
- 如果effect为空但键名key已填写，则表示与所有键名key的效果相匹配。

关于Kubernetes容忍度的详细说明，请参见[污点和容忍度](#)。

因此，需要修改工作负载的yaml，还原tolerations为默认配置如下：

```
tolerations:
- key: node.kubernetes.io/not-ready
  operator: Exists
  effect: NoExecute
  tolerationSeconds: 300
- key: node.kubernetes.io/unreachable
  operator: Exists
  effect: NoExecute
  tolerationSeconds: 300
```

上述默认的容忍度表示Pod可以在拥有以上污点的节点上运行300s，然后会被驱逐。

6.4.7 如何避免非 GPU/NPU 负载调度到 GPU/NPU 节点？

问题现象

当集群中存在GPU/NPU节点和普通节点混合使用的场景时，普通工作负载也可以调度到GPU/NPU节点上，可能出现GPU/NPU资源未充分利用的情况。

问题原因

由于GPU/NPU节点同样提供CPU、内存资源，在一般情况下，即使工作负载未声明使用GPU/NPU资源，调度器也会根据打分机制将工作负载调度到GPU/NPU节点运行，于是可能会出现GPU/NPU节点的CPU、内存资源被普通工作负载占用的情况，导致GPU/NPU资源闲置。

解决方案

在使用GPU/NPU节点时，可以为其添加污点，并通过工作负载容忍度设置，避免非GPU/NPU工作负载调度到GPU/NPU节点上。

- GPU/NPU工作负载：添加指定污点的容忍度，可以调度至GPU/NPU节点。
- 普通工作负载：未添加指定污点的容忍度，无法调度至GPU/NPU节点。

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。
- 步骤2 在左侧列表中选择“节点管理”，勾选GPU/NPU节点，并单击“标签与污点管理”。
- 步骤3 单击“新增批量操作”，为GPU/NPU节点添加污点。

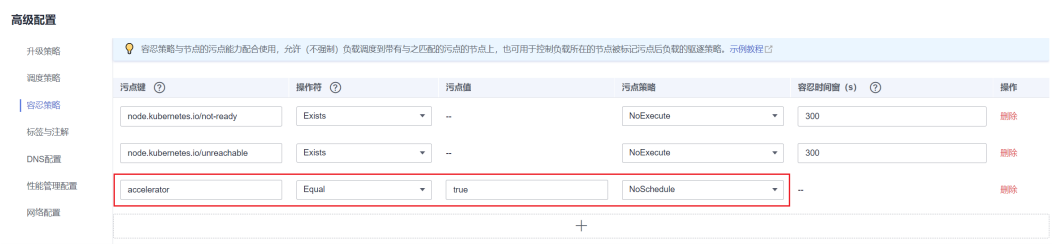
选择“污点(Taints)”，并填写键值与污点效果。示例中，为GPU/NPU节点添加 accelerator=true:NoSchedule的污点。

图 6-7 添加污点



- 步骤4 创建GPU/NPU工作负载时，在高级配置中，手动添加容忍策略，容忍该污点。

图 6-8 容忍策略



步骤5 普通工作负载创建时，无需添加容忍策略。由于未容忍该污点，则不会被调度到GPU/NPU节点。

----结束

6.4.8 为什么 Pod 调度不到某个节点上？

步骤1 请排查节点和docker是否正常，排查方法请参见[排查项七：内部组件是否正常](#)。

步骤2 如果节点和docker正常，而pod调度不到节点上，请确认pod是否做了亲和，排查方法请参见[排查项三：检查工作负载的亲和性配置](#)。

步骤3 如果节点上的资源不足，导致节点调度不上，请扩容或者新增节点。

----结束

6.4.9 修改 kubelet 参数导致已驱逐的 Pod 被重新调度

问题现象

如果节点存在Memory/Disk/PID Pressure的情况，节点会被添加系统污点。此时修改节点池kubelet组件配置参数或者重启节点kubelet后，该污点会被临时删除，可能会导致由于节点资源压力而触发驱逐的节点重新加入调度器计算流程中，Pod重新调度到该节点上，如果节点资源压力未缓解，之后节点会再次进入驱逐流程。

问题根因

当前kubelet上报Memory/Disk/PID Pressure情况（心跳）都依赖于eviction manager的检测。心跳上报和eviction manager的检测分别由两个协程并发执行。正常情况下，如果先执行eviction manager的检测，再执行心跳上报，则kubelet可以上报正确的磁盘情况，并不会删除污点。异常情况下，如果先执行心跳上报，再执行eviction manager的检测，则kubelet将会删除上一次的污点。

解决方案

对于该问题您可以无需处理，节点在一段时间后会自动驱逐Pod。

6.4.10 根据 GPU/NPU 卡信息定位使用该卡的 Pod

在CCE中使用GPU/NPU卡时，无法直接获取到使用该卡的Pod。您可以根据GPU/NPU卡的信息，通过kubectl命令行操作筛选Pod，以便在GPU/NPU卡故障时能够及时将Pod驱逐。

前提条件

- 已创建CCE集群，且配置了kubectl命令行工具。详细操作可参考[通过kubectl连接集群](#)。
- 集群上安装了CCE AI套件（GPU/NPU），可参考[CCE AI套件（NVIDIA GPU）](#)和[CCE AI套件（Ascend NPU）](#)。其中NPU驱动版本应大于23.0。

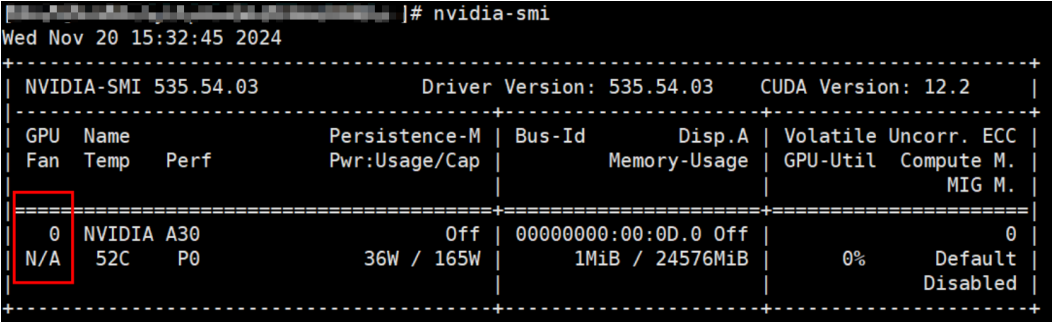
操作步骤

您可以在集群节点上查询GPU/NPU卡的信息，然后使用kubectl搜索到使用该卡的Pod。

GPU 场景

- 步骤1** 登录CCE控制台，在左侧导航栏中选择“节点管理”，切换至“节点”页签，查看GPU节点的IP。本文中以192.168.0.106为例。
- 步骤2** 登录GPU节点，通过以下命令查看GPU卡的信息。

```
nvidia-smi
```



The screenshot shows the output of the `nvidia-smi` command. It displays the NVIDIA-SMI version (535.54.03), Driver Version (535.54.03), and CUDA Version (12.2). Below this, a table lists GPU information. GPU 0 is highlighted with a red box. The table columns include GPU Name, Persistence-M, Bus-Id, Disp.A, Volatile Uncorr. ECC, Fan, Temp, Perf, Pwr:Usage/Cap, Memory-Usage, GPU-Util, Compute M., and MIG M.

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	52C	P0	36W / 165W	1MiB / 24576MiB	0%	Default	Disabled

可以看到该机器上存在1张卡GPU0。本文以GPU0为例，定位使用这张卡的Pod。

- 步骤3** 根据节点IP（即192.168.0.106）和设备号（即第0张卡）定位使用了该卡的Pod。

```
kubectl get pods --all-namespaces -o jsonpath='{range .items[?(@.spec.nodeName=="192.168.0.106")]}{.metadata.namespace}{"\t"}{.metadata.name}{"\t"}{.metadata.annotations}{"\n"}{end}' | grep nvidia0 | awk '{print $1, $2}'
```

该命令查找了192.168.0.106节点上所有的Pod，并且查找了annotation中存在nvidia0（代表第0张卡）信息的Pod。最终输出了Pod所在的命名空间和Pod名称。查询结果如下：

```
default k8-job-rhblr
```

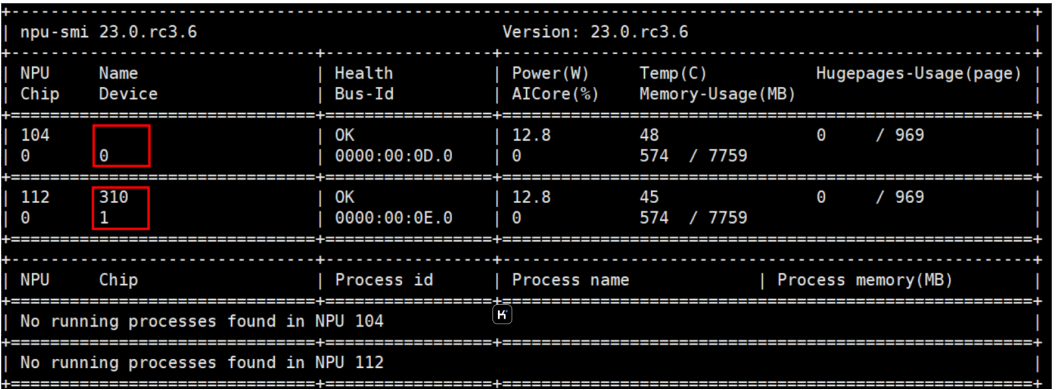
代表default命名空间下名为k8-job-rhblr的Pod使用了192.168.0.106节点上的第0张卡。

----结束

NPU 场景

- 步骤1** 登录CCE控制台，在左侧导航栏中选择“节点管理”，切换至“节点”页签，查看NPU节点的IP。本文中以192.168.0.138为例。
- 步骤2** 登录NPU节点，通过以下命令查看NPU卡的信息。

```
npu-smi info
```



The screenshot shows the output of the `npu-smi info` command. It displays the npu-smi version (23.0.rc3.6) and a table of NPU information. Two NPUs are listed: 104 and 112. GPU 0 is highlighted with a red box. The table columns include NPU Name, Health, Power(W), Temp(C), Hugepages-Usage(page), Chip, Device, Bus-Id, AICore(%), Memory-Usage(MB), Process id, Process name, and Process memory(MB).

NPU	Name	Health	Power(W)	Temp(C)	Hugepages-Usage(page)	Chip	Device	Bus-Id	AICore(%)	Memory-Usage(MB)	Process id	Process name	Process memory(MB)
104	0	OK	12.8	48	0 / 969	0	0	0000:00:0D:0	0	574 / 7759			
112	310	OK	12.8	45	0 / 969	1	1	0000:00:0E:0	0	574 / 7759			

可以看到该机器上存在两张卡device0和device1。本文以device0为例，定位使用了这张卡的Pod。

步骤3 根据节点IP（即192.168.0.138）和设备号（即第0张卡）定位使用了该卡的Pod。

```
kubectl get pods --all-namespaces -o jsonpath='{range .items[?(@.spec.nodeName=="192.168.0.138")]}{.metadata.namespace}{\t}{.metadata.name}{\t}{.metadata.annotations}{\n}{end}' | grep Ascend310-0 | awk '{print $1, $2}'
```

该命令查找了192.168.0.138节点上所有的Pod，并且查找了annotation中存在Ascend310-0（代表NPU卡的第0张卡）信息的Pod。最终输出了Pod所在的命名空间和Pod名称。查询结果如下：

```
[root@ ~]# kubectl get pods --all-namespaces -o jsonpath='{range .items[?(@.spec.nodeName=="192.168.0.138")]}{.metadata.namespace}{\t}{.metadata.name}{\t}{.metadata.annotations}{\n}{end}' | grep Ascend310-0 | awk '{print $1, $2}'
default test-564f996c77-fws6z
```

代表default命名空间下名为test-564f996c77-fws6z的Pod使用了192.168.0.138节点上的第0张卡。

说明

- 在使用其他卡时，Ascend310-0的Ascend310应该修改为对应卡名称。
- NPU驱动版本应大于等于23.0。

----结束

6.4.11 节点标签更新导致的 Pod 容器退出问题

问题现象

工作负载通过节点标签调度到节点运行后，如果更新节点标签，然后修改节点的kubelet配置参数导致kubelet组件重启。kubelet组件重启后约30s，查看工作负载状态，会出现异常事件“Predicate NodeAffinity failed”，原Pod状态变成Completed，并且新Pod将会重新调度。如果不存在其他符合指定标签的节点，新Pod也会调度失败。

问题原因

工作负载设置指定标签亲和后，Pod只能调度到存在指定标签的节点上。如果调度成功后修改节点标签，Pod仍可以在原节点上运行，但是重启kubelet后，kubelet会检查一次节点上的Pod亲和策略是否与节点匹配，不匹配的话会将原Pod状态设置为Completed，并会新建一个Pod重新调度。

该问题根因是Kubernetes的原生机制，为了保证Pod可以稳定运行，建议保持已运行Pod的亲和策略中指定的标签与所在节点的标签一致，避免kubelet重启后由于亲和策略检查不通过导致Pod容器退出，或者Pod重新部署后由于亲和策略不匹配节点标签导致Pod无法在原节点启动。

解决方案

- 如果您仅需要新增节点或节点池的K8S标签，建议不要删除节点或节点池上原有的K8S标签，使节点保持满足节点上已运行Pod的亲和策略。
- 如果您需要修改或删除某节点或某节点池的K8S标签，且修改后的K8S标签不满足该节点上已运行Pod的亲和策略，请根据以下步骤排查节点上是否存在配置亲和性调度的Pod，否则可能会导致Pod被重新调度。
 - 查找节点上的工作负载是否配置亲和策略。

- 如果节点为默认节点池（DefaultPool）中的节点，请执行以下命令，其中“节点IP”为实际的节点IP。

```
nodeIP=节点IP && kubectl get pod --all-namespaces -o=custom-columns=nodeIP:'status.hostIP,nodeAffinity:'spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | sed '1d' | grep $nodeIP | awk '{print substr($0, index($0,$2))}' | grep matchExpressions | uniq
```

- 如果输出为空，则说明节点中当前运行的负载没有配置亲和策略，不涉及该问题。
- 如果输出不为空，例如下图所示，则说明节点中存在配置了亲和tag1=value1标签的工作负载。

```
[root@192-18-0-12 paas]# nodeIP='192.18.0.22' && kubectl get pod --all-namespaces -o=custom-columns=nodeIP:'status.hostIP,nodeAffinity:'spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | sed '1d' | grep -E $nodeIP | awk '{print substr($0, index($0,$2))}' | grep matchExpressions | uniq
```

- 如果节点为自定义节点池中的节点，请执行以下命令，其中“节点池名称”为实际的节点池名称。

```
nodeIPs=$(kubectl get nodes -l cce.cloud.com/cce-nodepool=节点池名称 -o=custom-columns=IP:'metadata.annotations.alpha.kubernetes.io/provided-node-ip' | sed '1d' | tr '\n' '|' | sed 's/|/$//') && kubectl get pod --all-namespaces -o=custom-columns=nodeIP:'status.hostIP,nodeAffinity:'spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | sed '1d' | grep -E $nodeIPs | awk '{print substr($0, index($0,$2))}' | grep matchExpressions | uniq
```

- 如果输出为空，则说明节点池中当前运行的负载没有配置亲和策略，不涉及该问题。
- 如果输出不为空，例如下图所示，则说明节点池中存在配置了亲和tag1=value1标签的工作负载。

```
[root@192-168-0-217 paas]# nodeIPs=$(kubectl get nodes -l cce.cloud.com/cce-nodepool=节点池名称 -o=custom-columns=IP:'metadata.annotations.alpha.kubernetes.io/provided-node-ip' | sed '1d' | tr '\n' '|' | sed 's/|/$//') && kubectl get pod --all-namespaces -o=custom-columns=nodeIP:'status.hostIP,nodeAffinity:'spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | sed '1d' | grep -E $nodeIPs | awk '{print substr($0, index($0,$2))}' | grep matchExpressions | uniq
```

- b. 查询亲和策略关联的所有负载，其中“亲和策略”为上一步中查询出的亲和策略。

```
match='亲和策略' && kubectl get deployment,statefulset,daemonset,job,cronjob --all-namespaces -o=custom-columns=kind:'kind',name:'metadata.namespace',namespace:'metadata.name',nodeAffinity:'spec.template.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | grep -F "$match" | awk '{print "kind:" $1 ",namespace:" $2 ",name:" $3}'
```

如下图所示，查询出配置了tag1=value1标签亲和策略的工作负载。

```
[root@192-168-0-217 paas]# match='[map[matchExpressions:[map[key:tag1 operator:In values:[value1]]]]]' && kubectl get deployment --all-namespaces -o=custom-columns=kind:'kind',name:'metadata.namespace',namespace:'metadata.name',nodeAffinity:'spec.template.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms' | grep -F "$match" | awk '{print "kind:" $1 ",namespace:" $2 ",name:" $3}'
```

- c. 请谨慎评估该工作负载重启对业务的影响，再修改节点或节点池标签。

您也可以修改该工作负载的亲和策略，将其亲和调度到其他节点，详情请参见[设置节点亲和调度（nodeAffinity）](#)。等待Pod重新正常运行后，再修改该节点标签。

6.4.12 创建使用 GPU 虚拟化多卡均分调度功能的工作负载之后，大量 Pod 执行失败

问题现象

创建使用GPU虚拟化多卡均分调度功能的工作负载之后，大量的GPU Pod执行失败。具体示例如下：

```
kubectl get pods
```

回显结果如下：

NAME	READY	STATUS	RESTARTS	AGE
test-586cf9464c-54g4s	0/1	UnexpectedAdmissionError	0	57s
test-586cf9464c-58n6d	0/1	UnexpectedAdmissionError	0	10s
test1-689cf9462f-5bzcw	0/1	UnexpectedAdmissionError	0	58s
...				

问题原因

使用GPU虚拟化多卡均分调度功能，集群版本与CCE AI套件（NVIDIA GPU）插件版本需要适配，具体如下：

- 插件版本在2.1.41及以上时：集群版本在1.27.16-r20、1.28.15-r10、1.29.10-r10、1.30.6-r10、v1.31.4-r0及以上。
- 插件版本在2.7.57及以上时：集群版本在1.28.15-r10、1.29.10-r10、1.30.6-r10、v1.31.4-r0及以上。

当集群版本不兼容时，若节点的GPU虚拟化资源未耗尽，则Pod可以正常调度。但在资源耗尽后，当前版本的kubelet无法正确处理多卡均分调度请求，导致持续调度失败的Pod堆积，引发内存泄漏，最终造成批量Pod执行失败。

解决方案

- 如果需要使用GPU虚拟化多卡均分调度功能，则需要先将执行失败的工作负载删除，再将集群升级至适配版本，升级的具体步骤请参见[集群升级流程](#)。集群升级成功后，重新创建对应工作负载，即可正常调度。
- 如果不需要使用GPU虚拟化多卡均分调度功能，则需要依次重启对应节点的kubelet组件，以恢复其性能。具体步骤如下：

- a. 执行以下命令，查看问题Pod的所在节点，记录所有节点IP，后续需要根据节点IP依次恢复节点kubelet组件。

```
kubectrl get pod -l volcano.sh/gpu-num -owide
```

回显结果如下：

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
test-586cf9464c-54g4s	0/1	UnexpectedAdmissionError	0	5m57s		<none>
11.84.252.4						
test-586cf9464c-58n6d	0/1	UnexpectedAdmissionError	0	5m10s	172.19.0.24	
11.84.252.4						
test-586cf9464c-5bzcw	0/1	UnexpectedAdmissionError	0	5m58s		<none>
11.84.252.4						
test-586cf9464c-6bb5d	0/1	UnexpectedAdmissionError	0	6m15s		<none>
11.84.252.4						
test-586cf9464c-6r2bq	0/1	UnexpectedAdmissionError	0	5m11s		<none>
11.84.252.4						
test-586cf9464c-6rcpl	0/1	UnexpectedAdmissionError	0	6m11s	172.19.0.21	
11.84.252.4						
...						

- b. 执行以下命令，删除使用GPU虚拟化多卡均分调度的工作负载，deployment请替换为对应的工作负载类型，test请替换成对应的工作负载名称。

```
kubectrl delete deployment test # 删除无状态工作负载test
```

回显结果如下：

```
deployment.apps/test deleted
```

- c. 依次登录a中涉及的节点，执行以下命令，重启kubelet服务。

```
systemctl restart kubelet
```

在回显中输入节点密码，无错误提示则说明重启成功。重启成功后，之前执行失败的非多卡均分调度的Pod将正常调度。

6.5 其他

6.5.1 定时任务停止一段时间后，为何无法重新启动？

定时任务在运行过程中，如果被暂停，再次被开启时，控制器会检查上一次调度的时间点到现在所错过了调度次数。如果错过的调度次数超过100次，那么它就不会启动这个任务并记录这个错误，详情请参考[CronJob限制](#)。

Cannot determine if job needs to be started. Too many missed start time (> 100). Set or decrease .spec.startingDeadlineSeconds or check clock skew.

例如，假设一个CronJob被设置为从08:30:00开始每隔1分钟创建一个新的Job，且startingDeadlineSeconds字段未被设置。如果CronJob控制器从08:29:00到10:21:00终止运行，则该Job将不会启动，因为从08:29:00到10:21:00超过了100分钟，即错过的调度次数超过了100（示例中一个调度周期为1分钟）。

但如果设置了startingDeadlineSeconds字段，则控制器会统计从startingDeadlineSeconds设置的值到现在的时间，计算期间错过了多少次Job。例如，如果startingDeadlineSeconds是 200，则控制器会统计在过去200秒中错过了多少次Job。此时如果CronJob控制器同样在08:29:00到10:21:00时间段终止运行，则Job仍将从10:22:00开始，因为最近200秒中仅错过了3个调度（示例中一个调度周期为1分钟）。

解决方法

如果想要解决这个问题，可以在定时任务的CronJob中配置参数：startingDeadlineSeconds。该参数只能使用kubectl命令，或者通过API接口进行创建或修改。

YAML示例如下：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  startingDeadlineSeconds: 200
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox:1.28
              imagePullPolicy: IfNotPresent
              command:
                - /bin/sh
                - -c
                - date; echo Hello
          restartPolicy: OnFailure
```

如果重新创建CronJob，也可以临时规避这个限制。

6.5.2 创建有状态负载时，实例间发现服务是指什么？

云容器引擎的实例间发现服务，在原生Kubernetes中称之为Headless Service。Headless Service也是一种Service，但是会在YAML中定义spec.clusterIP: None，也就是不需要Cluster IP的Service。

Headless Service 和普通 Service 的区别

- 普通Service：
一个Service可能对应多个EndPoint（Pod），client访问的是Cluster IP，通过iptables或IPVS规则转到Real Server，从而达到负载均衡的效果。例如：Service有2个EndPoint，但是DNS查询时只会返回Service的地址，具体client访问的是哪个Real Server，是由iptables或IPVS规则来决定的，客户端无法自行选择访问指定的EndPoint。
- Headless Service：
访问Headless Service时，DNS查询会如实的返回每个真实的EndPoint（Pod的IP地址）。对应到每一个Endpoints，即每一个Pod，都会有对应的DNS域名；这样Pod之间就可以互相访问，达到实例间发现和访问的效果。

Headless Service 使用场景

当某个工作负载的多个Pod之间没有任何区别时，可以使用普通Service，利用集群kube-proxy实现Service的负载均衡，例如常见的无状态应用Nginx。

但是某些应用场景下，工作负载的各个实例间存在不同的角色区别，比如Redis集群，每个Redis实例都是不同的，它们之间存在主从关系，并且需要相互通信。这种情况下，使用普通Service无法通过Cluster IP来保证访问到某个指定的实例，因此需要设置Headless Service直接访问Pod的真实IP地址，实现Pod间互相访问。

Headless Service一般结合StatefulSet来部署有状态的应用，比如Redis集群、MySQL集群等。

6.5.3 CCE 容器拉取私有镜像时报错 “Auth is empty”

问题描述

在CCE的控制台界面中为已经创建的工作负载更换镜像，选择我上传的镜像，容器在拉取镜像时报错 “Auth is empty, only accept X-Auth-Token or Authorization”。

```
Failed to pull image "IP地址:端口号/magicdoom/tidb-operator:latest": rpc error: code = Unknown desc = Error response from daemon: Get https://IP地址:端口号/v2/magicdoom/tidb-operator/manifests/latest: error parsing HTTP 400 response body: json: cannot unmarshal number into Go struct field Error.code of type errcode.ErrorCode: "{\n\"errors\": [{\n\"code\": 400,\n\"message\": \"Auth is empty, only accept X-Auth-Token or Authorization.\"}]}\""
```

解答

您可以通过CCE控制台界面选择私有镜像创建应用，此时CCE会自动带上该secret，升级时不会出现该问题。

您通过API创建应用时，在deployment中带入该secret也可以在升级时避免该问题。

```
imagePullSecrets:  
- name: default-secret
```


6.5.4 CCE 集群中工作负载镜像的拉取策略有哪些？

容器在启动运行前，需要镜像。镜像的存储位置可能会在本地，也可能在远程镜像仓库中。

Kubernetes配置文件中的imagePullPolicy属性是用于描述镜像的拉取策略的，如下：

- Always：总是拉取镜像。
imagePullPolicy: Always
- IfNotPresent：本地有则使用本地镜像，不拉取。
imagePullPolicy: IfNotPresent
- Never：只使用本地镜像，从不拉取，即使本地没有。
imagePullPolicy: Never

说明如下：

1. 如果设置为Always，则每次容器启动或者重启时，都会从远程仓库拉取镜像。
如果省略imagePullPolicy，策略默认为Always。
2. 如果设置为IfNotPresent，有以下两种情况：
 - a. 当本地不存在所需的镜像时，会从远程仓库中拉取。
 - b. 如果需要的镜像和本地镜像内容相同，只不过重新打了tag。此tag镜像本地不存在，而远程仓库存在此tag镜像。这种情况下，Kubernetes并不会拉取新的镜像。

6.5.5 鲲鹏集群 Docker 容器挂载点被卸载

故障现象

鲲鹏集群Docker容器挂载点被卸载。

问题根因

鲲鹏集群节点为EulerOS 2.8系统时，如果在Docker服务文件中配置了MountFlags=shared字段，会因为systemd特性的原因导致容器挂载点被卸载。

解决方法

修改Docker服务文件，删除MountFlags=shared字段，重启Docker。

步骤1 登录节点。

步骤2 执行如下命令，删除配置文件中MountFlags=shared字段，然后保存。

vi /usr/lib/systemd/system/docker.service

```
[Service]
MountFlags=shared
Type=notify
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
Environment=GOTRACEBACK=crash
```

步骤3 重启Docker。

`systemctl restart docker`

----结束

6.5.6 下载镜像缺少层如何解决？

故障现象

在使用containerd容器引擎场景下，拉取镜像到节点时，概率性缺少镜像层，导致工作负载容器创建失败。

```
Events:
Type      Reason      Age      From      Message
----      -
Normal    Scheduled   54s      default-scheduler    Successfully assigned cattle-prometheus/prometheus-server-6c69469cf4-nfs7f to 10.14.11.139
Normal    SuccessfulMountVolume  55s      kubelet    Successfully mounted volume for pod "prometheus-server-6c69469cf4-nfs7f_cattle-prometheus(48ac202a-649a-429c-91ca-573dbaabcb72)"
Normal    SuccessfulUpdateSecurityGroup  52s      yangtse-controller    Successfully updated security group to "eb607f89-6fd6-431a-b901-e80758b5198c"
Normal    Pulled      8s (vs over 31s)  kubelet    Container image "100.125.0.29:2020/registry.cn-hangzhou.aliyuncs.com/buysbox1.29.2f already present on machine
Warning   FailedCreate 7s (vs over 39s)  kubelet    Error: failed to create container: error unpacking image: failed to extract layer sha256:f9f9e4e62f0689cd752390e14ade48b0ec6f2488a05af5ab2f9cca54c299d: failed to get reader from content store: content digest sha256:8c5a7d1afbc602695fc32c9b445743cec5ff32053ea389ea9bd8773b7068105: not found
```

问题根因

docker v1.10 之前支持mediaType 为 application/octet-stream 的layer，而containerd不支持application/octet-stream，导致没有拉取。

解决方法

有如下两种方式可解决该问题。

- 使用高版本Docker (>= docker v1.11) 重新打包镜像。
- 手动下载镜像
 - a. 登录节点。
 - b. 执行如下命令手动下载镜像。

```
ctr -n k8s.io images pull --user u:p images
```
 - c. 使用新下载的镜像重新创建工作负载。

6.5.7 容器内的文件权限和用户都是问号

问题现象

节点操作系统为CentOS 7.6或EulerOS 2.5时，如果使用“Debian GNU/Linux 11 (bullseye)”内核为基础镜像的容器，会出现容器内的文件权限和用户异常。


```
[root@ ]# docker run -it debian:11 bash
root@a6b8fa7fcdea:/# ls -al
ls: cannot access 'dev': Operation not permitted
ls: cannot access 'root': Operation not permitted
ls: cannot access 'run': Operation not permitted
ls: cannot access 'lib': Operation not permitted
ls: cannot access 'mnt': Operation not permitted
ls: cannot access '.': Operation not permitted
ls: cannot access 'tmp': Operation not permitted
ls: cannot access 'proc': Operation not permitted
ls: cannot access 'bin': Operation not permitted
ls: cannot access 'srv': Operation not permitted
ls: cannot access 'sys': Operation not permitted
ls: cannot access 'var': Operation not permitted
ls: cannot access 'etc': Operation not permitted
ls: cannot access 'media': Operation not permitted
ls: cannot access 'usr': Operation not permitted
ls: cannot access 'sbin': Operation not permitted
ls: cannot access 'home': Operation not permitted
ls: cannot access 'boot': Operation not permitted
ls: cannot access 'lib64': Operation not permitted
ls: cannot access '..': Operation not permitted
ls: cannot access 'opt': Operation not permitted
ls: cannot access '.dockerenv': Operation not permitted
total 0
d????????? ? ? ? ?      ? .
d????????? ? ? ? ?      ? ..
-????????? ? ? ? ?      ? .dockerenv
d????????? ? ? ? ?      ? bin
d????????? ? ? ? ?      ? boot
d????????? ? ? ? ?      ? dev
d????????? ? ? ? ?      ? etc
d????????? ? ? ? ?      ? home
d????????? ? ? ? ?      ? lib
d????????? ? ? ? ?      ? lib64
d????????? ? ? ? ?      ? media
d????????? ? ? ? ?      ? mnt
d????????? ? ? ? ?      ? opt
d????????? ? ? ? ?      ? proc
d????????? ? ? ? ?      ? root
d????????? ? ? ? ?      ? run
d????????? ? ? ? ?      ? sbin
```

问题影响

容器内文件权限及用户异常。

解决方案

CCE提供以下两种解决方案，您可根据实际情况选取：

- 建议业务容器的基础镜像使用Debian 9或者Debian 10。
- 建议节点操作系统使用EulerOS 2.9或者Ubuntu18.04。

6.5.8 如何导出容器中的文件？

在开发和生产环境中，可能存在很多场景需要将容器中的文件导出到宿主机，例如导出日志文件进行分析、导出配置文件进行备份、导出构建产物等等。本文将为您介绍如何将容器中的文件导出。

使用 kubectl 命令

您可以使用**kubectl cp**命令导出容器中的文件。假设您的Pod名为pod_name，位于default命名空间中，要从Pod的/path/to/file路径复制文件到本地的/local/path路径，执行以下命令：

```
kubectl cp default/pod_name:/path/to/file /local/path
```

更多操作详情请参见[kubectl cp](#)。

使用卷（Volume）

如果您需要频繁地在容器和宿主机之间交换文件，您可以使用卷（Volume）将宿主机的目录挂载到容器中。这样，容器中的文件可以直接在宿主机上访问和修改。

以下是创建一个Pod并挂载卷的示例，文件pod.yaml中定义了一个Pod并挂载宿主机的目录/local/path到容器的/path/to/mount路径。

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: example-container
      image: nginx
      volumeMounts:
        - mountPath: /path/to/mount
          name: example-volume
  volumes:
    - name: example-volume
      hostPath:
        path: /local/path
      imagePullSecrets:
        - name: default-secret
```

使用kubectl命令创建Pod：

```
kubectl apply -f pod.yaml
```

这样，容器中的/path/to/mount目录将直接映射到宿主机的/local/path目录，您可以在宿主机上直接访问和修改这些文件。

7 网络管理

7.1 网络异常问题排查

7.1.1 工作负载网络异常时，如何定位排查？

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- **排查项一：容器+容器端口**
- **排查项二：节点IP+节点端口**
- **排查项三：负载均衡IP+端口**
- **排查项四：NAT网关+端口**
- **排查项五：检查容器所在节点安全组是否放通**

排查项一：容器+容器端口

在CCE控制台界面或者使用kubectl命令查找pod的IP，然后登录到集群内的节点或容器中，使用curl命令等方法手动调用接口，查看结果是否符合预期。

如果容器IP+端口不能访问，建议登录到业务容器内使用“127.0.0.1+端口”进行排查。

常见问题：

1. 容器端口配置错误（容器内未监听访问端口）。
2. URL不存在（容器内无相关路径）。
3. 服务异常（容器内的业务BUG）。
4. 检查集群网络内核组件是否异常（容器隧道网络模型：openswitch内核组件；VPC网络模型：ipvlan内核组件）。

排查项二：节点 IP+节点端口

只有发布为节点访问（NodePort）或负载均衡（LoadBalancer）的服务才能通过节点 IP+节点端口进行访问。

- **节点访问（NodePort）类型：**
节点的访问端口就是节点对外发布的端口。
- **负载均衡（LoadBalancer）类型：**
负载均衡的节点端口通过“编辑YAML”可以查看。

如下图所示：

nodePort: 30637为节点对外暴露的端口。**targetPort: 80**为Pod对外暴露的端口。**port: 123**为服务对外暴露的端口，负载均衡类型的服务同时使用该端口配置ELB的监听器。

```
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 123
      targetPort: 80
      nodePort: 30637
```

找到节点端口（nodePort）后，使用容器所在节点的IP地址+端口进行访问，并查看结果是否符合预期。

常见问题：

1. 节点的入方向对业务端口未放通。
2. 节点配置了自定义路由，并且配置错误。
3. pod的label与service的label不匹配（kubectl或API创建）。

排查项三：负载均衡 IP+端口

如果使用负载均衡IP+端口不能访问，但节点IP+端口可以访问。

请排查：

- 相关端口或URL的后端服务器组是否符合预期。
- 节点上的安全组是否对ELB暴露了相关的协议或端口。
- 四层ELB的健康检查是否开启（未开启的话，请开启）。
- 七层ELB的访问方式中使用的证书是否过期。

常见问题：

1. 发布四层ELB时，如果客户在界面未开启健康检查，ELB可能会将流量转发到异常的节点。
2. UDP协议的访问，需要放通节点的ICMP协议。
3. pod的label与service的label不匹配（kubectl或API创建）。

排查项四：NAT 网关+端口

配置在NAT后端的服务器，通常不配置EIP，不然可能会出现网络丢包等异常。

排查项五：检查容器所在节点安全组是否放通

用户可单击服务列表中的“网络 > 虚拟私有云 VPC”，在网络控制台单击“访问控制 > 安全组”，找到CCE集群对应的安全组规则进行修改和加固。

- CCE集群：
Node节点的安全组名称是：**{集群名}-cce-node-{随机字符}**。
- CCE Turbo集群：
Node节点的安全组名称是：**{集群名}-cce-node-{随机字符}**。
容器关联的安全组名称是：**{集群名}-cce-eni-{随机字符}**。

请排查：

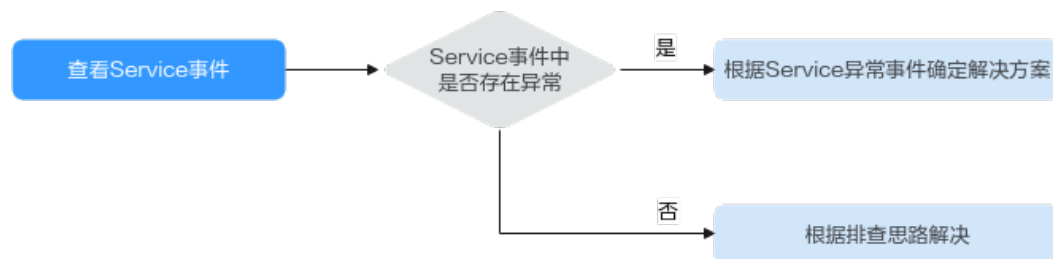
- 从集群外访问集群内负载时，来访者的IP地址、端口、协议需在集群安全组的入方向规则中开放。
- 集群内的工作负载访问外部时，访问的地址、端口、协议需在集群安全组的出方向规则中开放。

更多安全组配置信息请参见[集群安全组规则配置](#)。

7.1.2 负载均衡类型 Service 异常问题排查

负载均衡类型Service通过ELB负载均衡对外提供服务，创建负载均衡类型Service时，支持选择已有的ELB实例或自动创建ELB实例。若选择已有的ELB实例，CCE集群会为Service配置ELB监听器、后端服务器组等资源；若选择自动创建ELB实例，CCE除了会为Service配置ELB监听器、后端服务器组等资源，还会负责ELB实例的自动创建。本文将介绍负载均衡型Service的异常问题排查流程。

排查流程



1. 登录CCE控制台。
2. 单击集群名称进入集群，在左侧选择“服务”。
3. 单击对应Service操作栏中的“事件”，查看Service是否存在异常事件。
 - 如果Service存在异常事件，可通过根据事件，参考[根据Service异常事件排查](#)查找异常的解决方案。
 - 如果Service不存在异常事件，则一般为访问类问题或周边服务配置引入的问题，可参考[根据常见问题对比排查](#)查找异常的解决方案。

根据 Service 异常事件排查

不同异常事件的解决方法如下表所示：

报错信息	说明	解决方案
Quota exceeded for resources: loadbalancer	负载均衡器配额不足，无法为Service自动创建负载均衡器。	- 默认配额为100，请申请提升配额。关于如何查询和提升配额，请参见如何申请扩大配额？。 - 不同Service可以共用同一个负载均衡器，请评估是否可以复用。
Quota exceeded for resources: listener	监听器配额不足，无法为Service创建ELB监听器。	默认配额为200，请申请提升配额。关于如何查询和提升配额，请参见 如何申请扩大配额？ 。
can't find backend pod with service	Service无后端pod，请确认Service是否已关联Pod且Pod是否正常运行。	- 若未关联Pod，请将Service关联至对应的后端Pod。 - 若关联的Pod运行异常，请解决Pod异常的问题，请参见工作负载状态异常定位方法。
failed to ensure load balancer: Not Found	Service关联的负载均衡器不存在。	登录ELB控制台，在对应的region，根据Service的annotations中kubernetes.io/elb.id值，搜索对应的负载均衡器。如果确认对应的负载均衡器不存在，则按需重建相应的Service，选择其他已有的负载均衡器或选择自动创建负载均衡器。
Update loadbalancer of service([ServiceName]/[Namespace]) error: listener is empty	Service关联的负载均衡器的监听器可能被误删。	登录ELB控制台，在对应的region，根据Service的annotations中kubernetes.io/elb.id值，搜索对应的负载均衡器，并根据端口确认监听器是否存在。如果确认负载均衡器的指定监听器不存在，则按需重新创建相应的Service。

报错信息	说明	解决方案
Failed to CreateListener : request failed: {"error_msg":"Load Balancer [ELB id] already has a listener with protocol_port of [端口号].","error_code":"ELB.8907","request_id":"xxx"}, status code: 409	监听器端口冲突，ELB侧已有该端口的监听器。	更新Service，设置其他可用的端口号。
Failed to create member : {"error_msg":"Vpc [集群VPC id] of member's subnet_cidr [期望监听器所属的子网] and vpc [ELB VPC id] of loadbalancer [ELB id] mismatch","error_code":"ELB.8902","request_id":"xxx"}, status code: 400	创建Service时，关联了已有的负载均衡器，但负载均衡器和集群不在同一个VPC内。	请确保您选择的负载均衡器和集群在同一个VPC内。
Failed to CreateListener : request failed: {"error_msg":"Loadbalancer [ELB id] has no flavor of type L7_elastic and cannot create listeners of type L7.","error_code":"ELB.8907","request_id":"xxx"}, status code: 409	网络型ELB无法创建七层监听器。	按需准确选择对应规格的负载均衡器实例。 如果需要创建HTTP/HTTPS类型的七层监听器，请使用包含应用型规格的负载均衡器实例。
Failed to CreateListener : request failed: {"error_msg":"Loadbalancer [ELB id] has only flavor of type l7 and cannot create listeners of type l4.","error_code":"ELB.8907","request_id":"xxx"}, status code: 409	应用型ELB无法创建四层监听器。	按需准确选择对应规格的负载均衡器实例。 如果需要创建TCP/UDP类型的四层监听器，请使用包含网络型规格的负载均衡器实例。

根据常见问题对比排查

对于无Service异常事件的异常问题，请参考下表进行排查：

问题类别	问题现象	解决方案
ELB访问类	ELB负载不均	由于负载均衡配置的后端分配策略设置不合理导致ELB负载不均，详情请参见 ELB负载不均 。
	集群内无法访问ELB	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，详情请参见 集群内部无法使用ELB地址访问负载 。
	集群外无法访问ELB	- 可能是由于ELB配置异常或后端服务器存在异常，详情请参见集群外无法访问ELB。 - 如果是Turbo集群独享型ELB访问不通，请检查ENI安全组并放通源地址网段，安全组规则配置详情请参见集群安全组规则配置。
ELB配置类	ELB配置被修改	请参见 ELB配置为什么会被修改 。
	ELB的后端被自动删除	请参见 服务发布到ELB，ELB的后端为何会被自动删除？ 。
ELB删除类	Service删除后ELB未删除	在不同的场景下，ELB的自动删除策略不同，详情请参见 ELB删除策略 。

ELB 负载不均

问题原因

负载均衡配置的后端分配策略设置不合理。

解决方案

- externalTrafficPolicy为Local的Service需要将ELB后端分配策略设置为加权轮询算法，即为Service添加注解kubernetes.io/elb.lb-algorithm: ROUND_ROBIN。
- 如果业务为长连接，则需要将ELB后端分配策略设置为加权最少连接算法，即为Service添加注解kubernetes.io/elb.lb-algorithm: LEAST_CONNECTIONS。

集群外无法访问 ELB

问题原因

ELB配置异常或后端服务器存在异常。

排查步骤

- 执行以下命令，查看Service事件信息，并处理异常事件。具体操作，请参见[根据Service异常事件排查](#)。

```
kubectl -n {your-namespace} describe svc {your-svc-name}
```


2. 检查相关端口或URL的后端服务器组是否符合预期。

如果后端服务器组为空，请检查业务Pod是否正常运行。若关联的Pod运行异常，请解决Pod异常的问题，请参见[工作负载状态异常定位方法](#)。

ELB 配置为什么会被修改

问题原因

CCE集群会在一定条件下根据Service的配置自动更新ELB配置。您额外在ELB控制台上修改的配置均存在被覆盖的风险。

解决方案

建议您使用Annotation配置负载均衡类型的服务对应的ELB。关于注解配置方式的更多信息，请参见[使用Annotation配置负载均衡类型的服务](#)。

须知

请勿在ELB控制台上手动修改CCE创建并维护的ELB的任何配置，否则有配置丢失的风险，造成Service不可访问。

ELB 删除策略

Service删除时对于ELB的自动删除策略如下：

- 对于Service自动创建的ELB：
 - 如果该ELB未被其他Service使用，则删除原Service时系统会自动删除对应的ELB。
 - 如果其他Service也关联了该ELB，则删除原Service时不会删除对应的ELB。
- Service选择已有的ELB时，删除Service时，不会删除原有的ELB。

7.1.3 集群内部无法使用 ELB 地址访问负载

问题现象

在集群内部（节点上或容器中），使用ELB地址无法访问。

问题原因

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。

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
			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
			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
			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控制器插件对接独享型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

解决办法

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":"traffic","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。

📖 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": "traffic", "bandwidth_size": 5, "bandwidth_sharetype": "PER", "eip_t
ype": "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
```

7.1.4 集群外部访问 Ingress 异常

Ingress基于七层的HTTP和HTTPS协议进行转发，是集群流量的入口，可以通过域名和路径对访问做到更细粒度的划分。集群在添加Ingress后，可能会出现无法正常访问的情况，本文提供添加Ingress失败或无法正常访问的通用排查思路，帮助您找到问题所在。在进行Ingress问题排查前，请您阅读以下几点须知，并进行自检。

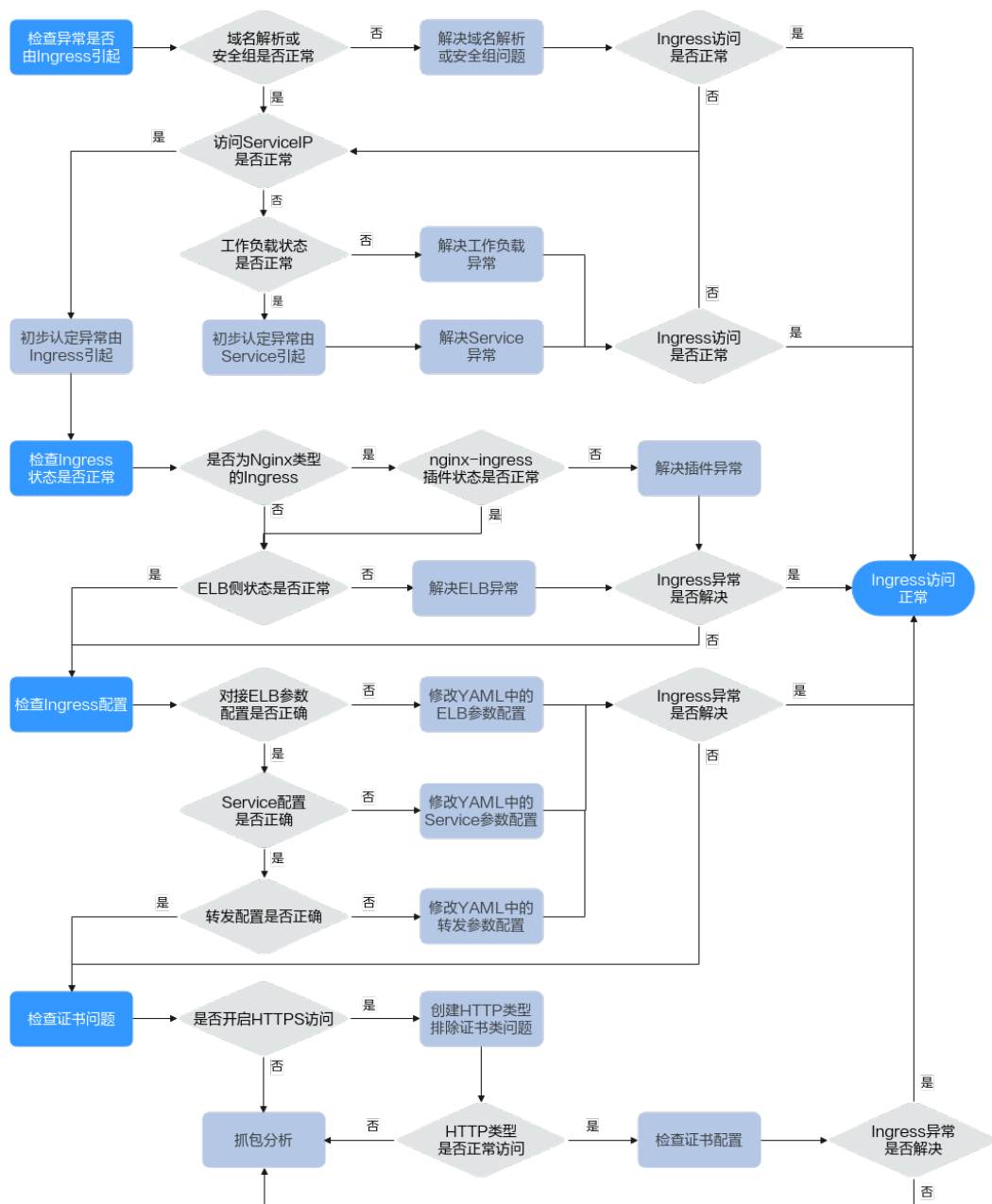
须知

- 如创建Ingress过程中指定了host地址，将无法通过IP访问。
- 请检查集群的节点安全组，确认30000-32767范围内的业务端口在入方向对所有网段放开。

排查思路

本文提供了Ingress外部访问异常排查全景图，如[图7-1](#)所示，帮助您由浅入深地排查Ingress访问异常问题。

图 7-1 Ingress 访问异常排查全景图



1. 检查异常是否由Ingress引起。

首先需要确认问题是否由Ingress导致的，因此需要确保外部域名解析正常、安全组规则正确，且Ingress对应的Service和工作负载工作正常。

2. 检查Ingress状态是否正常。

在Service和工作负载都正常的情况下，需要保证Ingress依赖的ELB状态正常。如果是Nginx型的Ingress，还需要保证nginx-ingress插件的状态是正常的。

3. 检查Ingress配置是否正确。

如果以上排查结果都正常，说明可能是Ingress的配置出现问题。

- 检查对接ELB参数是否填写正确。
- 检查Service参数是否填写正确。

- 检查转发配置参数是否填写正确。
- 4. **检查证书问题。**
如果Ingress开启了HTTPS访问，还需要排除证书配置错误的问题。您可使用相同ELB创建一个HTTP协议的Ingress访问，如HTTP协议下访问正常，则说明HTTPS协议证书可能存在问题。
- 5. 如果以上排查均无效果，请进行抓包分析，或提交工单寻求帮助。

检查异常是否由 Ingress 引起

您需要确认访问异常是否由Ingress引起，当存在域名解析异常、安全组规则错误、Service异常或工作负载本身异常时，都可能会引起Ingress访问不通。

以下排查顺序遵从由外到内的规则：

步骤1 检查域名解析或安全组规则是否正常，排除集群外部可能存在的问题。

1. 执行以下命令检查域名在权威DNS的解析是否生效。
`nslookup -qt=类型 域名 权威DNS地址`
2. 检查集群节点安全组规则，确认30000-32767范围内的业务端口在入方向对所有网段放开。如需对安全组进行加固，详情请参见[集群安全组规则配置](#)。

☐	优先级 ?	策略 ?	协议端口 ?	类型	源地址 ?
☐	1	允许	UDP : 30000-32767	IPv4	0.0.0.0/0 ?
☐	1	允许	TCP : 30000-32767	IPv4	0.0.0.0/0 ?

步骤2 检查Service是否可以正常访问容器内业务，检查集群内部可能存在的问题。

您可通过在集群中新建Pod并通过ClusterIP访问Service的方式进行检查，如您的服务为NodePort类型，也可通过EIP:Port使用互联网访问服务来验证。

1. 通过kubectl连接集群，查询集群内服务。

```
# kubectl get svc
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
kubernetes ClusterIP  10.247.0.1    <none>       443/TCP    34m
nginx     ClusterIP  10.247.138.227 <none>       80/TCP     30m
```
2. 创建一个Pod并登录到容器内。
`kubectl run -i --tty --image nginx:alpine test --rm /bin/sh`
3. 使用curl命令访问Service的ClusterIP:Port，检验集群内服务是否可访问。
`curl 10.247.138.227:80`
如Service可正常访问，则说明后端工作负载状态正常，初步可认定异常由Ingress引起，请参照[检查Ingress状态是否正常](#)继续排查。
如Service访问异常，则需继续排查工作负载状态，确定异常原因。

步骤3 检查工作负载状态是否正常。

如工作负载正常，但Service无法正常访问，则说明异常可能是由于Service引起。请检查Service的配置，例如容器端口是否正确填写为容器内业务开放端口。

如工作负载正常，但访问结果不符合预期，请排查容器内运行的业务代码。

----结束

检查 Ingress 状态是否正常

CCE支持两种类型的Ingress，其中Nginx类型的Ingress Controller由社区开源的插件提供，需要在集群中安装插件自行运维；而ELB型的Ingress Controller运行在master节点上，由专门的华为云团队负责运维，无需用户保证。

步骤1 如果您使用Nginx类型的Ingress，需要在集群中安装nginx-ingress插件。如果您使用ELB型的Ingress，则无需检查此步骤。

前往“插件中心”查看nginx-ingress插件的状态为“运行中”。请保证集群下节点资源充足，若资源不足，插件实例将无法调度。

步骤2 前往ELB控制台检查ELB的状态。

- **ELB型Ingress**
访问端口可自定义，请检查ELB侧创建的监听器和后端服务器组未被删除或修改。
建议您在创建ELB型Ingress时通过控制台选择自动创建ELB，并且不要对自动创建的ELB进行修改，能够有效避免ELB侧导致的Ingress异常。
- **Nginx型Ingress**
访问端口固定为80和443，不支持自定义端口。安装nginx-ingress插件将同时占用80和443端口，切勿删除，否则只能通过重装插件解决。

您也可以通过错误码来简单判断故障是否由于ELB引起，如下图页面所示，说明该异常大概率由ELB侧故障引起，需要重点排查。

404 Not Found

ELB

----结束

检查 Ingress 配置是否正确

如果以上排查项都正常，需要考虑是否由于参数设置引起异常。由于使用kubectl创建时需要填写的参数较多易出错，建议您使用控制台创建，根据可视界面按需设置参数，自动过滤不符合要求的负载均衡及Service，能够有效避免出现关键参数格式错误或缺失的问题。

请您根据以下思路进行逐一排查Ingress配置：

- **检查对接ELB参数是否正确**

由于ELB通过annotations字段下的参数进行定义，但是K8s在创建资源时并不会对annotations字段参数进行校验，如果出现关键参数错误或缺失，Ingress资源也可被创建，但无法正常访问。

以下是出现频率较高的问题，供您参考：

- 对接的目标ELB未与集群处于同一VPC下。
- 添加ELB型Ingress时对接已有ELB，annotations中关键字段kubernetes.io/elb.id、kubernetes.io/elb.ip、kubernetes.io/ingress.class、kubernetes.io/elb.port缺失。

- 添加Nginx型Ingress时，**未安装nginx-ingress插件**导致无ELB连接。
- 添加Nginx型Ingress时，**kubernetes.io/elb.port**参数不支持自定义端口，使用HTTP协议固定为80，HTTPS协议固定为443。
- **检查Service配置是否正确**
 - 检查Ingress对接的Service类型是否正确，Ingress支持的Service如下。

表 7-1 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	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	支持

表 7-2 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

- 检查Service的访问端口号是否正确，此处Service的访问端口号（ port字段 ）需区别于容器端口号（ targetPort字段 ）。
- **检查转发配置参数是否填写正确**
 - 添加的URL转发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。

 说明

使用Nginx型的Ingress Controller时，您可以通过在annotations字段添加rewrite注释进行重定向，将业务内不存在的path路径进行重写，避免访问路径不存在的错误，详情请参见[Rewrite](#)。

- 创建Ingress时指定了域名（ host ），将无法通过IP访问。

检查证书问题

CCE的Ingress密钥证书类型为IngressTLS或kubernetes.io/tls，若证书类型不正确，创建的Ingress将无法在ELB侧建立监听器，导致Ingress访问异常。

步骤1 首先去除YAML中关于HTTPS的参数，尝试创建HTTP类型的Ingress是否可正常访问。

如HTTP访问正常，则考虑HTTPS密钥证书是否存在问题。

步骤2 排除密钥类型错误。检查密钥类型是否为IngressTLS或kubernetes.io/tls类型。

```
kubectl get secret
```

回显如下：

NAME	TYPE	DATA	AGE
ingress	IngressTLS	2	36m

步骤3 创建测试证书，排除证书问题。

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out tls.crt -subj "/"  
CN={YOUR_HOST}/O={YOUR_HOST}"
```

步骤4 使用测试的私钥证书tls.key和tls.crt创建正确类型的密钥，并重新创建HTTPS类型的Ingress测试是否可以正常访问。本文以创建IngressTLS类型密钥为例。

通过kubectl方式创建时，创建IngressTLS类型密钥的示例如下：

```
kind: Secret  
apiVersion: v1  
type: IngressTLS  
metadata:  
  name: ingress  
  namespace: default  
data:  
  tls.crt: LS0tLS1CRU*****FURS0tLS0t  
  tls.key: LS0tLS1CRU*****VZLS0tLS0=
```

说明

此处tls.crt和tls.key为示例，请获取真实密钥进行替换。tls.crt和tls.key的值为Base64加密后的内容。

----结束

7.1.5 CCE 集群中域名解析失败

问题现象

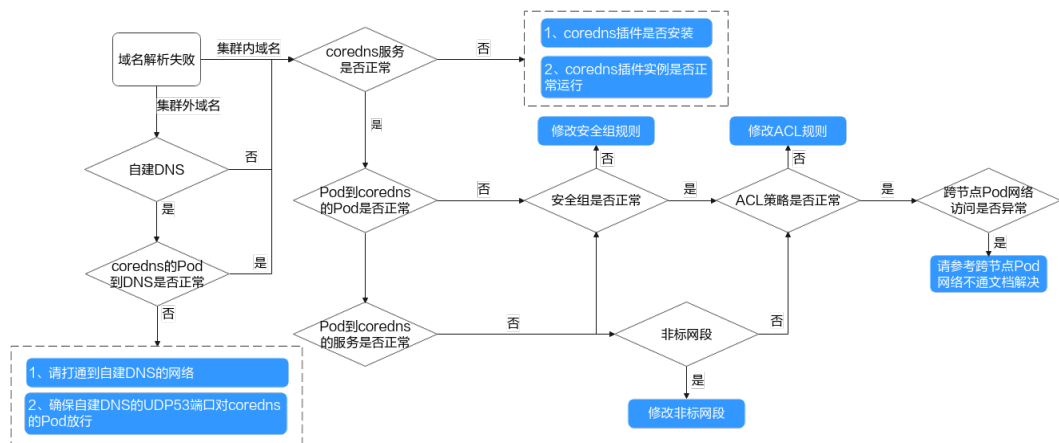
CCE集群中域名解析失败。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频原因往低频原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

图 7-2 域名解析失败排查思路



当遇到域名解析失败的问题时，首先需要判断是集群内域名还是集群外域名解析失败。

- 集群内域名：

- a. 确认coredns插件是否安装，coredns服务是否正常运行。
- b. 其他Pod到coredns的Pod网络是否正常，其他Pod到coredns的服务是否网络正常，如网络不正常：
 - i. 安全组是否正常
 - ii. ACL是否正常
 - iii. 是否跨节点Pod网络是否正常，如果跨节点Pod网络不通则需要确认以下问题是否存在：
 - 1) 修改了节点内核
 - 2) 安全组和ACL策略未放通
 - 3) VPC路由表是否正常
 - 4) 节点上的iptables规则是否正常
 - 5) 内核其他参数
 - 6) 非标网段（CCE不推荐使用非标网段，使用非标网段可能引入集群网络不通等风险）

- 集群外域名：

- a. 确定是否为自建DNS（容器如果未走coredns或者节点DNS非本region云解析的地址均属自建）
- b. coredns到自建DNS网络是否正常，工作负载到自建DNS的网络是否正常，如不正常：
 - i. 请打通到自建DNS网络
 - ii. 请确保DNS的UDP53端口放行，需要对Pod网段安全组&ACL放通如下策略：
 - 节点网段到节点网段
 - 节点网段到容器网段
 - 容器网段到节点网段
 - 容器网段到容器网段

7.1.6 为什么访问部署的应用时浏览器返回 404 错误码？

CCE服务本身在浏览器中访问应用时不会返回任何的错误码，请优先排查自身业务。

404 Not Found

如果404的返回如下图所示，说明这个返回码是ELB返回的，说明ELB找不到相关的转发策略。请排查相关的转发规则等。

图 7-3 404:ALB



The image shows a large, bold, black text "404 Not Found" centered on a white background. Below it, in a smaller, blue font, is the text "ALB".

ALB

如果404的返回如下图所示，说明这个返回码是由nginx（客户业务）返回，请排查客户自身业务问题。

图 7-4 404:nginx/1.**.*



The image shows a large, bold, black text "404 Not Found" centered on a white background. Below it, in a smaller, blue font, is the text "nginx/1.14.0".

nginx/1.14.0

7.1.7 为什么容器无法连接互联网？

当容器无法连接互联网时，首先需要排查容器所在节点能否连接互联网。其次，需要查看容器的网络配置是否正确，例如DNS配置是否可以正常解析域名。

排查项一：节点能否连接互联网

步骤1 登录ECS控制台。

步骤2 查看节点对应的弹性云服务器是否已绑定弹性IP或者配置NAT网关。

如**图7-5**，若弹性IP一栏有IP地址，表示已绑定弹性IP；若没有，请为弹性云服务器绑定弹性IP。

图 7-5 节点是否已绑定弹性 IP



----结束

排查项二：节点是否配置网络 ACL

步骤1 登录VPC控制台。

步骤2 单击左侧导航栏的“访问控制 > 网络ACL”。

步骤3 排查节点所在集群的子网是否配置了网络ACL，并限制了外部访问。

----结束

排查项三：检查容器 DNS 配置

在容器中执行`cat /etc/resolv.conf`命令，查看DNS配置。示例如下：

```
nameserver 10.247.x.x
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

若nameserver设置为10.247.x.x说明DNS对接到集群的CoreDNS，需要确保集群CoreDNS工作负载运行正常。如果是其他IP地址，则表示采用云上DNS或者用户自建的DNS，请您自行确保解析正常。

更多域名解析异常的排查思路，请参见[CCE集群中域名解析失败](#)。

7.1.8 VPC 的子网无法删除，怎么办？

VPC的子网无法删除可能是因为您在CCE的集群中使用了该VPC的子网，因此需要在CCE界面删除相应的集群后，再删除VPC的子网。

须知

- 删除集群会将集群内的节点以及运行的工作负载和服务都销毁，请谨慎操作。
- 不建议在ECS界面删除CCE集群中的节点。

7.1.9 CCE Turbo 集群如何修复出现故障的容器网卡？

CCE Turbo集群采用的是云原生网络2.0网络模型，每个Pod创建时都会分配一张网卡，您可以根据Pod上的Event事件进行对应的处理恢复：

- [Pod创建报错exit status -1](#)
- [Pod创建报错timed out waiting for the condition \[arping timeout\]](#)
- [Pod创建报错timed out waiting for the condition \[vlan is exist\]](#)

- [Pod创建报错timed out waiting for the condition \[port unavailable\]](#)
- [Pod创建报错no eni bound to pod](#)
- [Pod正常创建并启动成功，但网络异常](#)

Pod 创建报错 exit status -1

此报错表明容器网络插件为该Pod设置相关的网络配置错误，根因一般为节点过载或单节点大批量并发创建Pod时执行相关的ip或iptables命令报错，出现此Event事件无需任何处理，kubelet会自动重试，最终会正常拉起Pod。如果Pod持续报错且最终无法拉起请提交工单联系运维人员进行处理。

Pod 创建报错 timed out waiting for the condition [arping timeout]

此报错表明该Pod分配到的网卡无法arping通Pod的子网网关，根因一般为VPC底层网络故障，10分钟后如果VPC底层网络还未恢复，CCE会对此Pod重新分配新的网卡。如果您想尽快恢复，请重建Pod。如果Pod持续报错请提交工单联系运维人员进行处理。

Pod 创建报错 timed out waiting for the condition [vlan is exist]

此报错是由于节点上已经存在相同vlan号的vlan子接口，导致无法在节点上创建辅助弹性网卡的vlan子接口，请确保您没有自行创建vlan子接口。如果您想尽快恢复，请重建Pod。如果Pod持续报错请提交工单联系运维人员进行处理。

Pod 创建报错 timed out waiting for the condition [port unavailable]

此报错表明节点上无法查找到分配给此Pod的弹性网卡，根因一般为BMS底层网络故障。如果您想尽快恢复，请重建Pod。如果Pod持续报错请提交工单联系运维人员进行处理。

Pod 创建报错 no eni bound to pod

此报错表明该Pod没有分配到网卡，需要您进一步查看Pod上的其他Event事件做进一步的辅助定位：

- **报错[insufficient IP addresses in subnets]**

此报错是由于Pod使用的容器网络配置的子网的IP已耗尽或您没有配置子网。您可以参考以下步骤进行解决：

 - a. 登录[CCE控制台](#)，单击集群名称进入集群。
 - b. 查看“网络信息”，确认您配置的默认容器子网信息，确认是否还有足够的IP。

网络信息		前往网络配置
网络模型	云原生网络2.0	
DataPlane V2	关闭	
VPC	vpc- 	
子网	subnet-cd83 (可用/总IP数: 95/251)	
默认容器子网	subnet- 	(可用/总IP数: 135/251)
全部展开 		
编辑		

- c. 如果已无可用IP，可单击“编辑”，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。

• **报错[insufficient NICs on node]**

此报错是由于节点上绑定的网卡数已经达到此节点规格的最大配额，根因一般为您给此节点额外绑定了弹性网卡，导致额外占用了节点弹性网卡的配额。如您需要额外绑定弹性网卡，请联动修改节点池的maxPods配置。配置详情请参见[修改节点池配置](#)。

• **报错[insufficient secgroup referred by nic]**

此报错是由于Pod使用的安全组已经超过最大可关联的网卡数限制。由于节点上预热的网卡也会使用default-network上配置的安全组，请合理配置容器网卡动态预热，避免节点上预热过多的网卡，进而占用安全组关联的网卡数配额。详情请参见[CCE Turbo配置容器网卡动态预热](#)。

如果您当前集群的Pod规模数确实比较大，会超过安全组最大可关联的网卡数限制，请为不同的节点池配置不同的安全组以提升规模。配置详情请参见[修改节点池配置](#)。

• **报错Security group xxx does not exist**

此报错是由于Pod使用的容器网络配置里的安全组ID不存在。您可以参考以下步骤进行解决：

- 登录[CCE控制台](#)，单击集群名称进入集群。
- 在“配置中心”页面，切换至“网络配置”页签。
- 在“容器网络配置”中，更新安全组。
- 更新完成后，重建Pod即可恢复。

Pod 正常创建并启动成功，但网络异常

如果Pod正确创建并启动成功，表明创建过程网络一切正常，但后续网络不通，可能有以下原因：

- Pod对应的安全组或ACL禁止了您的网络访问，请确保您的安全组规则或ACL配置正确。
- 可在容器内尝试arping Pod的子网网关，命令如下：

```
arping -I eth0 $Pod的子网网关
```

如果无法arping通，大概率Pod在节点上对应的容器网络配置被误删除或者VPC底层网络异常。如果您想尽快恢复，请重建Pod，也可提交工单联系运维人员进行处理。

- 新旧节点IP复用触发社区已知的[issue](#)。CCE Turbo集群中DaemonSet类型的Pod的创建时间如果早于对应新Node节点的创建时间，可能导致分配给Pod的网卡又被删除，请重建Pod恢复。

7.1.10 节点无法连接互联网（公网），如何排查定位？

当节点无法连接互联网时，请参照如下方法排查。

排查项一：节点是否绑定弹性 IP

登录ECS控制台，查看节点对应的弹性云服务器是否已绑定弹性IP。

若弹性IP一栏有IP地址，表示已绑定弹性IP。若没有，请为弹性云服务器绑定弹性IP。

图 7-6 节点是否已绑定弹性 IP



排查项二：节点是否配置网络 ACL

登录VPC控制台，单击左侧导航栏的“访问控制 > 网络ACL”。排查节点所在集群的子网是否配置了网络ACL，并限制了外部访问。

7.1.11 如何解决 VPC 网段与容器网络冲突的问题？

在集群创建页面，若“容器网段”配置与“VPC网段”冲突，界面会提示“该网段与VPC网段有冲突，请重新选择”，重新调整“容器网段”即可。

图 7-7 网段冲突提示



7.1.12 ELB 四层健康检查导致 java 报错：Connection reset by peer

完整错误信息

```
java.io.IOException: Connection reset by peer
at sun.nio.ch.FileDispatcherImpl.read0(Native Method)
at sun.nio.ch.SocketDispatcher.read(SocketDispatcher.java:39)
at sun.nio.ch.IOUtil.readIntoNativeBuffer(IOUtil.java:223)
at sun.nio.ch.IOUtil.read(IOUtil.java:197)
at sun.nio.ch.SocketChannelImpl.read(SocketChannelImpl.java:380)
at com.wanyu.smarthome.gateway.EquipmentSocketServer.handleReadEx(EquipmentSocketServer.java:245)
at com.wanyu.smarthome.gateway.EquipmentSocketServer.run(EquipmentSocketServer.java:115)
```

分析结果

使用Java NIO建立Socket服务端，当客户端意外关闭的情况，不是发送指定指令通知服务器退出，就会产生此错误。

TCP 健康检查的机制

1. ELB节点根据健康检查配置，向后端服务器（IP+健康检查端口）发送TCP SYN报文。
2. 后端服务器收到请求报文后，如果相应的端口已经被正常监听，则会返回SYN+ACK报文。
3. 如果在超时时间内没有收到后端服务器的SYN+ACK报文，则判定健康检查失败，然后发送RST报文给后端服务器中断TCP连接。
4. 如果在超时时间内收到了SYN+ACK报文，则发送ACK给后端服务器，判定健康检查成功，并发送RST报文给后端服务器中断TCP连接。

注意

正常的TCP三次握手后，会进行数据传输，但是在健康检查时会发送RST中断建立的TCP连接。该实现方式可能会导致后端服务器中的应用认为TCP连接异常退出，并打印错误信息，如“Connection reset by peer”。

这种错误是合理范围内的，无法避免的，不必关心它。

7.1.13 Service 事件：Have no node to bind，如何排查？

步骤1 登录CCE控制台，进入集群，在左侧导航栏选择“服务”。

步骤2 在service列表里确认此服务是否有关联的工作负载，或关联的工作负载的相关实例是否正常。

----结束

7.1.14 为什么登录虚拟机 VNC 界面会间歇性出现 Dead loop on virtual device gw_11cbf51a, fix it urgently?

问题现象

VPC网络模式的集群，登录虚拟机出现 Dead loop on virtual device gw_11cbf51a, fix it urgently，如图：

```
[7520230.908741] Dead loop on virtual device gw_11cbf51a, fix it urgently!  
[7764908.323899] Dead loop on virtual device gw_11cbf51a, fix it urgently!  
[7876345.412678] Dead loop on virtual device gw_11cbf51a, fix it urgently!  
[7886952.430199] Dead loop on virtual device gw_11cbf51a, fix it urgently!  
[8053806.787694] Dead loop on virtual device gw_11cbf51a, fix it urgently!
```

原因定位

VPC网络模式的集群采用了linux开源社区的ipvlan模块实现容器网络通信，这一日志打印与ipvlan的设计实现有关，ipvlan L2E模式它优先进行二层转发，再进行三层转发。

场景还原：

假定有业务Pod A，它持续对外提供服务，不断被同节点访问收发报文，通过本机k8s service经过容器gw接口进行访问，或者同属本节点的Pod间直接互相访问。在升级、扩容，或者其他原因导致的退出场景，容器A已停止运行，对应的网络资源被回收。此时同节点的报文仍持续尝试在往容器A的IP发送报文。内核中的ipvlan模块首先尝试根据目的IP来二层转发这些报文，但是由于Pod A已无法找到该IP所属的网卡，ipvlan模块判断它有可能是个外部报文，因此尝试进行三层转发，根据路由规则匹配上了gw口，因此gw口又收到此报文，再经由ipvlan模块转发，如此循环。内核中的dev_queue_xmit函数检测到重复进入发包过程达10次，报文被丢弃同时打印该日志。

发起访问端的在报文丢失后一般会进行几次退避重试，因此在这种场景下会连续打印几次条日志，直到发起访问端的容器内ARP老化或业务自身终止访问。

跨节点容器间通信，由于目的IP及源IP不属于同个节点级专属子网（注意此子网与VPC子网概念不同），报文不会重复走到此业务流程因此，不会触发此问题。

同集群不同节点间的Pod通过Cluster模式的NodePort来访问除外，它会被SNAT成被访问端容器gw接口的IP，因此也有可能触发此日志打印。

问题影响

被访问端容器正常运行时不会有影响。容器被销毁时，有一定影响，但影响较小，重复进入发包过程10次，然后被丢包，这个过程在内核中处理十分迅速，对性能影响可以忽略。

对于ARP老化或业务自身不再重试，或新容器会被拉起，容器service服务报文经过kubeproxy重定向到新业务。

开源现状

目前开源社区ipvlan L2E模式仍存在此问题，已向开源社区反馈，待确认更优方案。

解决方法

打印Dead loop问题本身无需解决。

但推荐服务Pod使用优雅退出，在业务真正终止服务前，优先将Pod设置为删除中的状态，待业务处理完毕请求后再退出。

7.1.15 集群节点使用 networkpolicy 概率性出现 panic 问题

问题场景

集群版本：v1.15.6-r1版本

集群类型：CCE集群

网络模式：容器隧道网络模式

节点操作系统：CentOS 7.6

上述集群的用户配置使用networkpolicy后，由于节点上canal-agent网络组件与CentOS 7.6内核存在不兼容，概率性导致CentOS 7.6的节点panic。

受影响范围

以下三条为必要条件，若有一条不满足，则您不会受到此问题的影响：

- v1.15.6-r1容器隧道网络模式集群。
- canal-agent组件版本为1.0.RC10.1230.B005或更低版本的CentOS 7.6节点（简单的判断方法为2021年2月23日及之前创建的节点）。
- 计划使用或已配置使用networkpolicy规则。

排查方法

快速排查方法（适用于节点为按需计费类型）

若您的节点为按需计费类型，可从cce-console上查看节点创建时间，对创建于2021年2月24日及之后的新建CentOS 7.6节点已无该问题。

准确排查方法（通用）

若您的集群版本为v1.15.6-r1，网络模式为容器隧道网络，节点操作系统为CentOS 7.6，而canal-agent组件版本为1.0.RC10.1230.B005.sp1或更高版本，则无此问题；若低于此版本（例如1.0.RC10.1230.B005、1.0.RC10.1230.B003、1.0.RC10.1230.B002），则建议您重置/删除对应节点后再使用networkpolicy。

执行以下步骤查询节点网络组件版本：

步骤1 准备可执行kubectl的节点。

步骤2 执行如下命令查询存量CentOS节点列表：

```
for node_item in $(kubectl get nodes --no-headers | awk '{print $1}'); do kubectl get node ${node_item} -o yaml | grep CentOS >/dev/null; if [[ "$?" == "0" ]];then echo "${node_item} is CentOS node";fi;done
```

回显如图：

```
10.0.50.187 is CentOS node
10.0.50.220 is CentOS node
10.0.50.43 is CentOS node
```

步骤3 假定用户CentOS节点的ip为10.0.50.187（请替换为实际IP），执行下述命令查看canal-agent版本：

```
kubectl get packageversions.version.cce.io 10.0.50.187 -o yaml | grep -A 1 canal-agent
```

回显如图：

```
- name: canal-agent
  version: 1.0.RC10.1230.B005.sp1
```

----结束

解决办法

如果您希望继续使用该节点资源，建议重置所属集群中的CentOS 7.6节点，以升级节点上网络组件到最新版本，具体操作请参考[重置节点](#)。

如果您希望删除该隐患节点后重新购买，具体操作请参考[删除节点](#)、[购买节点](#)。

7.1.16 节点远程登录界面(VNC)打印较多 source ip_type 日志问题

问题场景

集群版本：v1.15.6-r1版本

集群类型：CCE集群

网络模式：VPC网络

节点操作系统：CentOS 7.6

上述节点的容器进行容器间通信时，由于容器网络组件在VNC界面打印较多source ip_type或者not ipvlan but in own host日志，导致影响节点VNC界面使用体验及高负载场景下的容器网络性能。现象如下：

```
[ 3840.916433] =====source ip_type 2, ipv4 10.0.0.128, mac fa:16:3e:57:f2:8f
[ 3840.916527] =====source ip_type 2, ipv4 10.0.0.129, mac fa:16:3e:57:f2:8f
[ 3840.916736] =====source ip_type 2, ipv4 10.0.0.129, mac fa:16:3e:57:f2:8f

[16739.000551] =====not ipvlan but in own host, mac_src=fa:16:3e:34:23:93 mac_dst=ff:ff:ff:ff:ff:ff
[16740.000968] =====not ipvlan but in own host, mac_src=fa:16:3e:34:23:93 mac_dst=ff:ff:ff:ff:ff:ff
```

排查方法

1、快速排查方法

适用于节点为按需计费类型，若您的节点为该类型，可从cce-console上查看节点创建时间，2021年2月24日及之后创建的CentOS 7.6节点无该问题。

2、准确排查方法（通用）

您可以执行下述步骤排查节点是否受此问题影响：

步骤1 以root用户登录CCE集群节点。

步骤2 执行下述命令排查是否为隐患节点：

```
ETH0_IP=$(ip addr show eth0 | grep "inet " | head -n 1 | awk '{print $2}' | awk -F '/' '{print $1}');arping -w 0.2 -c 1 -I gw_11cbf51a 1.1.1.1 >/dev/null 2>&1; echo ;dmesg -T | grep -E "!==not ipvlan but in own host! ==source ip_type" 1>/dev/null 2>&1; if [[ "$?" == "0" ]];then echo "WARNING, node ${ETH0_IP} is affected"; else echo "node ${ETH0_IP} works well"; fi;
```

说明

参考命令中1.1.1.1为示例IP，该IP仅用于触发发送ARP报文。可直接使用，也可替换为任意合法IP。

步骤3 若回显为如下则表示节点存在隐患（10.2.0.35为示例节点eth0网卡IP，具体以实际回显为准）：

```
WARNING, node 10.2.0.35 is affected
```

若回显如下则表示节点无此问题：

```
node 10.2.0.35 works well
```

----结束

解决办法

如果您希望继续使用该节点资源，建议重置所属集群中的CentOS 7.6节点，以升级节点上网络组件到最新版本，具体操作请参考[重置节点](#)。

如果您希望删除该隐患节点后重新购买，具体操作请参考[删除节点](#)、[购买节点](#)。

7.1.17 使用 IE 浏览器访问 nginx-ingress 出现重定向 308 无法访问

问题现象

NGINX Ingress控制器从较老的版本升级后，使用IE浏览器无法访问已有的服务，状态码显示为308。

问题根因

NGINX Ingress控制器在升级后默认的永久重定向状态码从301变成了308，而部分老版本的IE浏览器不支持308重定向，因此出现无法访问的问题。

NGINX Ingress控制器社区issue：<https://github.com/kubernetes/ingress-nginx/issues/1825>

解决方法

您在创建Ingress时，可以通过“`nginx.ingress.kubernetes.io/permanent-redirect-code`”注解指定永久重定向的状态码为301。

示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/permanent-redirect-code: '301'
...
```

7.1.18 NGINX Ingress 控制器插件升级导致集群内 Nginx 类型的 Ingress 路由访问异常

问题现象

集群中存在未指定Ingress类型（`annotations`中未添加`kubernetes.io/ingress.class: nginx`）的Nginx Ingress路由，NGINX Ingress控制器插件从1.x版本升级至2.x版本后，服务中断。

问题自检

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。

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:' -B 1
```

- 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@192-168-0-31 ~]# 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集群可通过managedFields机制确认。

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

```
[root@192-168-0-31 ~]# 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'
```

```
[root@192-168-0-31 ~]# kubectl logs -nkube-system cceaddon-nginx-ingress-controller-545db6b4f7-bv74t | grep 'updating Ingress status'
8 status.go:281] "updating Ingress status" namespace="default" ingress="test" currentValue=[] newV
alue=[{"IP: 192.168.0.31 Hostname: Ports: []"} {"IP: 192.168.0.31 Hostname: Ports: []"}]
```

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

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。

问题根因

NGINX Ingress控制器插件基于开源社区Nginx Ingress Controller的模板与镜像。

对于社区较老版本的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。详情请参见[社区代码](#)。

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。详情请参见[社区代码](#)。

社区相关PR链接为：<https://github.com/kubernetes/ingress-nginx/pull/7341>

目前有两种方式指定Ingress类别：

- 通过annotations指定，为Ingress资源添加annotations（`kubernetes.io/ingress.class: nginx`）。
- 通过spec指定，`.spec.ingressClassName`字段配置为`nginx`。但需要配套具有IngressClass资源。

示例如下：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
    ...
status:
  loadBalancer: {}
```

7.1.19 负载均衡型 Service 更新出现错误：Quota exceeded for resources: members_per_pool

问题现象

负载均衡型Service更新时出现错误，信息如下：

(combined from similar events):Details:Update member of listener/pool(dc9098a3-e004-4e60-ac6c-44a9a04bd8f8/539490e1-51c2-4c09-b4df-10730f77e35f) error: Failed to create member : (error_msg:"Quota exceeded for resources: members_per_pool","error_code":"ELB.8905","request_id":"e064fd46211318ff57f455b29c07c821"),status code: 409

事件					
注：事件保存时间为1小时，1小时后自动清除数据					
K8s 组件名	事件类型	发生次数	事件名称		
hws-cloudprovider	告警	218	更新负载均衡失败	(combined from similar events): Details: U...	2023/12/11 16:34:43 G... 2023/12/12 10:03:38 G...
service-controller	正常	221	正常事件	Ensuring load balancer	2023/12/11 16:30:52 G... 2023/12/12 10:03:38 G...

排查项一：后端服务器数量是否达到配额上限

ELB单后端服务器组可添加的后端服务器数量默认配额上限值为500。在CCE Turbo集群中使用独享型ELB创建Service时，Service关联的每个Pod实例都会在ELB侧创建一个后端服务器，当后端服务器数量超过配额上限时，会出现以上错误。

解决方法：根据业务需求对负载均衡器的后端服务器进行合理规划。

- [如何查询配额详情](#)
- [如何查询配额使用详情](#)

排查项二：后端服务器健康检查是否有异常

在Service更新ELB后端服务器的过程中，为保证流量不中断，会先增加后端服务器，在确保新加的后端服务器可用后，再删除原先的后端服务器。

但在后端服务器配额已满的情况下，无法再新增后端服务器，会出现以上超出配额上限的错误，此时会直接更新已有的后端服务器。如果在更新Service时配置出现错误，导致更新后的后端服务器全部健康检查失败，在保证流量不中断的逻辑下，Service此时并不会删除原先正常的后端服务器，最终导致只有部分后端服务器更新为错误配置，其余后端服务器仍保持原先配置。

解决方法：在后端服务器配额已满的情况下，更新Service时需要配置正确的健康检查协议、端口，查看健康检查是否成功。

7.1.20 ELB Ingress 出现告警：Invalid input for rules

问题现象

创建或更新ELB Ingress时出现告警，信息如下：

```
Update elb(****) listener(****)error: status_code: 400, resp_body:{"error_msg":"Invalid input for rules.  
Reason: the number of condition for per policy must be no larger than  
10.", "error_code":"ELB.8902", "request_id": "*****"}
```

解决方案

该告警中的错误码ELB.8902表示请求参数输入有误，详情请参见[弹性负载均衡错误码](#)。您需要根据问题原因修改正确的参数配置。

问题现象	问题根因	解决方案
告警信息中包含以下内容： Reason: the number of condition for per policy must be no larger than 10.	由于ELB的API限制，使用高级转发规则时，每一条转发策略中设置的域名、路径、HTTP请求方法、HTTP请求头、查询字符串、网段、Cookie所有类型的转发规则下条件总数最多为10（同一个规则中的每个参数值均视为1个条件）。 例如，在一条转发策略中已设置1个域名转发规则、1个路径转发规则。如果继续添加“查询字符串”类型的转发规则，则最多只能添加8个字符串值。 高级转发规则详情请参见 为ELB Ingress配置高级转发规则 。	修改Ingress配置，设置转发策略中的规则数量不超过上限。

7.1.21 为 ELB Ingress 配置了 HTTPS 证书后访问异常的原因有哪些？

为ELB Ingress配置了HTTPS证书后，如果证书配置出现以下问题，可能导致访问异常，您可以参考表格中的原因进行排查。

访问异常原因	问题现象	解决方案
证书已过期	通过curl命令测试时报错信息如下： SSL certificate problem: certificate has expired	请及时替换新证书。
客户端使用了不配套的HTTPS证书链验证ELB Ingress侧配置的HTTPS证书	通过curl命令测试时报错信息如下： SSL certificate problem: unable to get local issuer certificate	请确保客户端中的HTTPS证书链与ELB Ingress中配置的证书配套。
创建证书时，未指定域名	通过curl命令测试时报错信息如下： SSL: unable to obtain common name from peer certificate	请在创建证书时指定域名。
访问的域名和HTTPS证书的域名不一致	通过curl命令测试时报错信息如下： SSL: certificate subject name 'example.com' does not match target host name 'test.com'	请为Ingress配置和域名相符的证书。

说明

您可以通过以下命令查看证书的过期时间、域名等信息，其中ca.crt为证书路径。

```
openssl x509 -in ca.crt -subject -noout -text
```

更新证书操作

- 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。

7.1.22 Nginx Ingress 中默认开启 HSTS 会导致 HTTP 请求重定向到 HTTPS

问题现象

Nginx Ingress中没有配置HTTP重定向至HTTPS，`nginx.ingress.kubernetes.io/ssl-redirect`设置为false，但是访问HTTPS之后再访问HTTP会被重定向成HTTPS。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: 200m
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
```

问题根因

NGINX Ingress控制器默认开启HTTP Strict Transport Security (HSTS) 配置，当客户端初次访问开启HSTS的服务端时，服务端会在返回客户端时携带HSTS的响应头，例如：

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

此时浏览器会记录这个HSTS响应头，并记住该域名在接下来一段时间内（指定的max-age值）只能通过HTTPS访问。当客户端下次通过HTTP访问同一域名时，浏览器会将请求直接重定向到HTTPS。

解决方案

如果客户端不希望支持HTTP自动转到HTTPS协议时，您可以在NGINX Ingress控制器插件的配置中关闭HSTS，关于HSTS的更多说明请参见[HSTS](#)。

- 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“管理”。
- 步骤2** 找到Ingress对应的控制器实例，单击“编辑”。
- 步骤3** 在Nginx配置参数中选择“YAML配置”，并将hsts参数设置为false。

nginx 配置参数

界面化配置

YAML配置

配置参数范围请参考[社区文档](#)。不在范围内的参数配置不会生效。
配置参数的值必须为字符串格式，否则无法安装成功。

```
1 {
2   "allow-backend-server-header": "false",
3   "enable-underscores-in-headers": "false",
4   "generate-request-id": "true",
5   "hsts": "false",
6   "ignore-invalid-headers": "true",
7   "keep-alive-requests": "100",
8   "max-worker-connections": "65536",
9   "proxy-body-size": "20m",
10  "proxy-connect-timeout": "10",
11  "reuse-port": "true",
12  "server-tokens": "false",
13  "ssl-redirect": "false",
14  "upstream-keepalive-connections": "320",
15  "upstream-keepalive-timeout": "900",
16  "worker-cpu-affinity": "auto"
17 }
```

- 步骤4** 单击“确定”，等待插件更新完成。

说明

客户端HSTS默认是有缓存的，当关闭NGINX Ingress控制器的HSTS后，请您清理客户端缓存。

----结束

7.2 网络规划

7.2.1 集群与虚拟私有云、子网的关系是怎样的？

“虚拟私有云”类似家庭生活中路由器管理192.168.0.0/16的私有局域网，是为用户在云上构建的一个私有网络，是弹性云服务器、负载均衡、中间件等工作的基本网络环境。根据实际业务需要可以设置不同规模的网络，一般可为10.0.0.0/8~24，172.16.0.0/12~24，192.168.0.0/16~24，其中最大的网络10.0.0.0/8的A类地址网络。

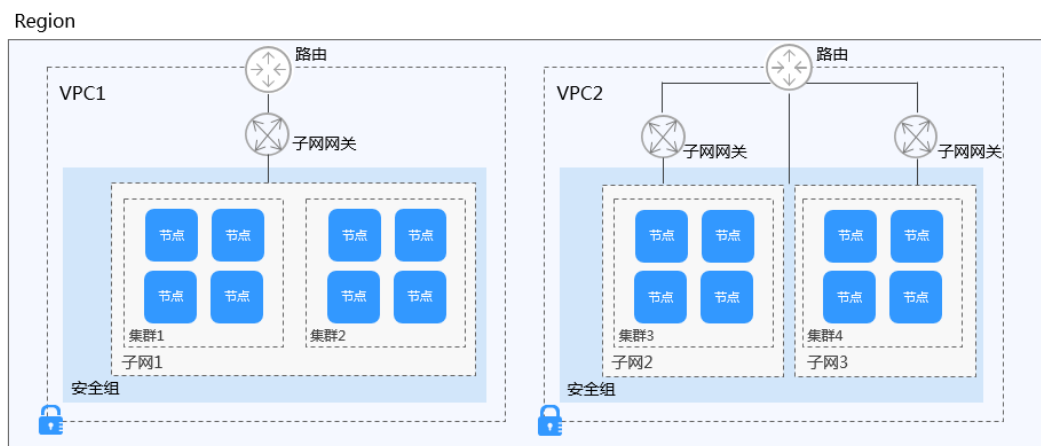
子网是虚拟私有云中的一个子集，可以将虚拟私有云划分为一个个子网，每个子网之间可以通过安全组控制其之间能否互通，保证子网之间可以相互隔离，用户可以将不同业务部署在不同的子网内。

集群是同一个VPC中一个或多个弹性云服务器或裸金属服务器（又称：节点）通过相关技术组合而成的计算机群体，为容器运行提供了计算资源池。

如图7-8，同一个region下可以有多个虚拟私有云（图中以VPC表示）。虚拟私有云由一个个子网组成，子网与子网之间的网络交互通过子网网关完成，而集群就是建立在某个子网中。因此，存在以下三种场景：

- 不同集群可以创建在不同的虚拟私有云中。
- 不同集群可以创建在同一个子网中。
- 不同集群可以创建在不同的子网中。

图 7-8 集群与 VPC、Subnet 的关系



相关文档

集群网络地址段规划实践

7.2.2 如何查看虚拟私有云 VPC 的网段？

在“虚拟私有云”页面，可查看虚拟私有云的“名称/ID”和“VPC网段”。用户可以调整已创建的VPC或通过重新创建VPC调整网段。

图 7-9 查看 VPC 网段

虚拟私有云 ⓘ 创建虚拟私有云

您还可以创建2个虚拟私有云。

名称/ID	状态	VPC网段	子网个数	操作
vpc-webmobile 0c8ecb72-df79-4a4f-b530-405510d402f3	正常	192.168.0.0/16	1个	修改 删除
vpc-demo 1664dda5-7181-4562-9c68-1047d10c2afc	正常	192.168.0.0/16	1个	修改 删除
vpc-b98f 291d42e2-c2a3-45d8-a621-543f3ae6609b	正常	192.168.0.0/16	1个	修改 删除

7.2.3 如何设置 CCE 集群中的 VPC 网段和子网网段？

VPC中的子网网段一旦创建，便无法更改。创建虚拟私有云时，请预留一定的VPC网段和子网网段资源，避免后续无法扩容。

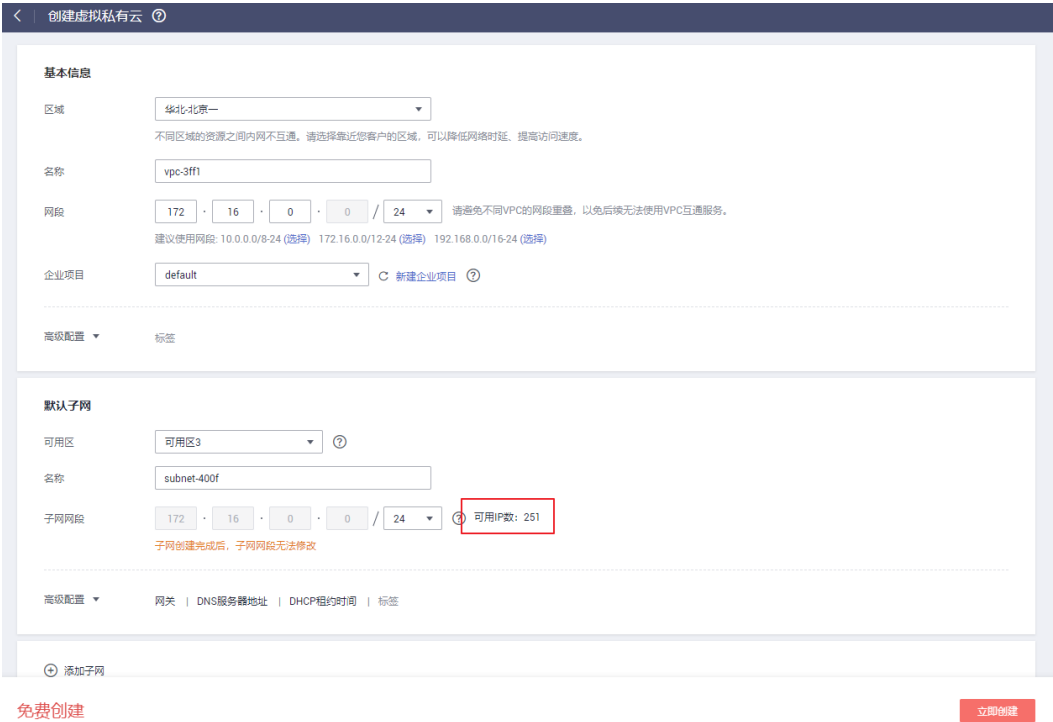
子网网段可在“创建虚拟私有云”页面的“子网配置 > 子网网段”中进行设置。在设置选项下可查看到“可用IP数”。

容器网段掩码如果设置不合适，会导致集群实际可用的节点较少。

例如：

- 节点规模为1000，子网可以选择192.168.0.0/20，支持约4090个节点。
- VPC选择192.168.0.0/16，子网选择192.168.0.0/25，则可支持122个节点。若创建200个节点集群，则实际只能添加122个节点（包括控制节点）。

图 7-10 查看“可用 IP 数”



相关文档

[集群网络地址段规划实践](#)

7.2.4 如何设置 CCE 集群中的容器网段？

进入CCE控制台，在创建集群时进行“容器网段”设置。

当前可供选择的容器网段为10.0.0.0/8~18，172.16.0.0/16~18，192.168.0.0/16~18。

集群创建完成后，如需添加容器网段，可前往集群信息页面，单击“添加容器网段”进行添加。

须知

- 使用“容器隧道网络”模型的集群暂不支持添加容器网段。
- 容器网段添加后无法删除，请谨慎操作。
- 服务网段默认为10.247.0.0/16，容器网段不能选择此网段。

The screenshot displays the CCE (Cloud Container Engine) management interface. On the left is a sidebar with navigation options: 集群信息 (Cluster Information), 资源 (Resources), and 运维 (Operations). The main panel shows the details for a specific cluster, categorized into '基本信息' (Basic Information) and '网络信息' (Network Information).

Cluster Details:

- Cluster Name:** [Redacted]
- Cluster ID:** 4c57febc-9090-11ed-bb66-0255ac1000c4
- Cluster Type:** CCE 集群
- Cluster Version:** v1.21
- Patch Version:** v1.21.6-r0
- Cluster Status:** 运行中 (Running)
- Cluster Management Scale:** 50 节点
- Creation Time:** 2023/01/10 10:41:34 GMT+08:00
- Enterprise Project:** default

Network Configuration:

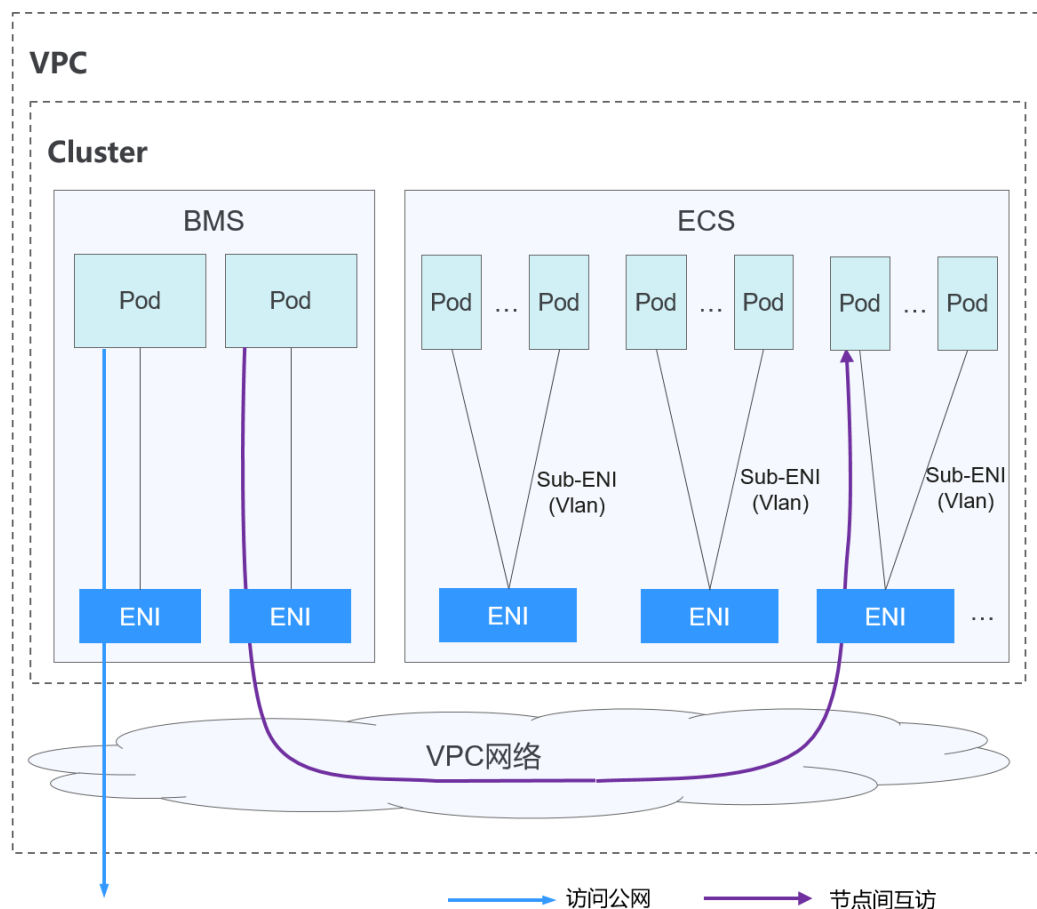
- Network Model:** VPC 网络
- VPC:** vpc-cce
- Subnet:** subnet-cce
- Container Network:** 172.16.0.0/16
- Add Container Network:** 添加容器网段 (highlighted with a red box)
- Service Network:** 10.247.0.0/16
- Forwarding Mode:** iptables
- Node Default Security Group:** cce-node-u04rv

7.2.5 什么是云原生网络 2.0 网络模式，适用于什么场景？

云原生网络 2.0 是什么

云原生网络2.0是新一代容器网络模型，深度整合了虚拟私有云VPC的原生弹性网卡（Elastic Network Interface，简称ENI）能力，采用VPC网段分配容器地址，支持ELB直通容器，享有高性能。

图 7-11 云原生网络 2.0



约束与限制

仅CCE Turbo集群支持使用云原生网络2.0。

适用场景

- 性能要求高，需要使用VPC其他网络能力的场景：由于云原生网络2.0直接使用的VPC网络，性能与VPC网络的性能几乎一致，所以适用于对带宽、时延要求极高的业务场景，比如：线上直播、电商优惠等。
- 大规模组网：云原生网络2.0当前最大可支持2000个ECS节点，10万个容器。

7.2.6 什么是弹性网卡？

弹性网卡即虚拟网卡，您可以通过创建并配置弹性网卡，并将其附加到您的云服务器实例（包括弹性云服务器和裸金属服务器）上，实现灵活、高可用的网络方案配置。

弹性网卡类型

- 主弹性网卡：在创建云服务器实例时，随实例默认创建的弹性网卡称作主弹性网卡。主弹性网卡无法与实例进行解绑。
- 扩展弹性网卡：您可以创建扩展弹性网卡，将其附加到云服务器实例上，您也可以将其从实例上进行解绑。每台实例可附加的扩展弹性网卡数量由实例规格决定。

应用场景

- 灵活迁移
通过将弹性网卡从云服务器实例解绑后再绑定到另外一台服务器实例，保留已绑定私网IP、弹性公网IP和安全组策略，无需重新配置关联关系，将故障实例上的业务流量快速迁移到备用实例，实现服务快速恢复。
- 业务分离管理
可以为服务器实例配置多个分属于同一VPC内不同子网的弹性网卡，特定网卡分别承载云服务器实例的内网、外网、管理网流量。针对子网可独立设置访问安全控制策略与路由策略，弹性网卡也可配置独立安全组策略，从而实现网络隔离与业务流量分离。

使用限制

- 云服务器实例与扩展弹性网卡必须在同一VPC，可以分属于不同安全组。
- 主弹性网卡不能解绑服务器。
- 云服务器可附加的扩展弹性网卡数量由云服务器实例规格决定。

7.2.7 集群安全组规则配置

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

用户可根据安全需求，登录[VPC控制台](#)，在左侧导航栏单击“访问控制 > 安全组”，找到集群对应的安全组规则进行修改和加固。

如集群在创建时需要指定节点安全组，请参考集群自动创建的默认安全组规则放通指定端口，以保证集群中的正常网络通信。

不同网络模型的默认安全组规则如下：

- [VPC网络模型安全组规则](#)
- [容器隧道网络模型安全组规则](#)
- [云原生网络2.0（CCE Turbo集群）安全组规则](#)

须知

- 安全组规则的**修改和删除可能会影响集群的正常运行**，请谨慎操作。如需修改安全组规则，请尽量避免对CCE运行依赖的端口规则进行修改。
- 在集群中添加新的安全组规则时，需要**确保新规则与原有规则不会发生冲突**，否则可能导致原有规则失效，影响集群正常运行。

VPC 网络模型安全组规则

Node节点安全组

集群自动创建的Node节点安全组名称为**{集群名}-cce-node-{随机ID}**，默认端口说明请参见[表7-3](#)。

表 7-3 VPC 网络模型 Node 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	UDP：全部	VPC网段	Node节点之间互访、Node节点与Master节点互访。	不建议修改	如果修改会影响集群的正常运行。
	TCP：全部				
	ICMP：全部	Master节点安全组	Master节点访问Node节点。	不建议修改	如果修改会影响集群的正常运行。
	TCP：30000-32767	所有IP地址（0.0.0.0/0）	集群NodePort服务默认访问端口范围。	可修改	端口需对VPC网段、容器网段和ELB的网段放通。
	UDP：30000-32767				
	全部	容器网段	允许集群中的容器访问节点。	不建议修改	如果修改会影响集群的正常运行。
	全部	Node节点安全组	限制Node节点安全组外的访问，但对于Node节点安全组中的实例互相访问不做限制。	不建议修改	如果修改会影响集群的正常运行。
	TCP：22	所有IP地址（0.0.0.0/0）	允许SSH远程连接Linux弹性云服务器。	建议修改	建议修改成只允许固定IP或IP地址段访问。
出方向规则	全部	所有IP地址（0.0.0.0/0）	默认全部放通，通常情况下不建议修改。	可修改	如需加固出方向规则，请注意指定端口需要放通，详情请参见 安全组出方向规则加固建议 。

Master节点安全组

集群自动创建的Master节点安全组名称为{集群名}-cce-control-{随机ID}，默认端口说明请参见表7-4。

表 7-4 VPC 网络模型 Master 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	TCP: 5444	VPC网段	kube-apiserver服务端端口，提供K8s资源的生命周期管理。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 5444	容器网段			
	TCP: 9443	VPC网段	Node节点网络插件访问Master节点。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 5443	所有IP地址（0.0.0.0/0）	kube-apiserver的HAProxy负载均衡端口，负责将请求转发到kube-apiserver后端实例。	建议修改	端口需保留对VPC网段、容器网段和托管网格控制面网段放通。 说明 如果您需要使用CloudShell功能，请保留5443端口对198.19.0.0/16网段放通，否则将无法访问集群。
	TCP: 8445	VPC网段	Node节点存储插件访问Master节点。	不建议修改	如果修改会影响集群的正常运行。
	全部	Master节点安全组	限制Master节点安全组外的访问，但对于Master节点安全组中的实例互相访问不做限制。	不建议修改	如果修改会影响集群的正常运行。
出方向规则	全部	所有IP地址（0.0.0.0/0）	默认全部放通。	不建议修改	如果修改会影响集群的正常运行。

容器隧道网络模型安全组规则

Node节点安全组

集群自动创建的Node节点安全组名称为{集群名}-cce-node-{随机ID}，默认端口说明请参见表7-5。

表 7-5 容器隧道网络模型 Node 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	UDP: 4789	所有IP地址 (0.0.0.0/0)	容器间网络互访。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 10250	Master节点网段	Master节点主动访问Node节点的kubelet（如执行 kubectl exec {pod}）。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 30000-32767	所有IP地址 (0.0.0.0/0)	集群NodePort服务默认访问端口范围。	可修改	端口需对VPC网段、容器网段和ELB的网段放通。
	UDP: 30000-32767				
	TCP: 22	所有IP地址 (0.0.0.0/0)	允许SSH远程连接Linux弹性云服务器。	建议修改	建议修改成只允许固定IP或IP地址段访问。
	全部	Node节点安全组	限制Node节点安全组外的访问，但对于Node节点安全组中的实例互相访问不做限制。	不建议修改	如果修改会影响集群的正常运行。
出方向规则	全部	所有IP地址 (0.0.0.0/0)	默认全部放通，通常情况下不建议修改。	可修改	如需加固出方向规则，请注意指定端口需要放通，详情请参见 安全组出方向规则加固建议 。

Master节点安全组

集群自动创建的Master节点安全组名称为{集群名}-cce-control-{随机ID}，默认端口说明请参见[表7-6](#)。

表 7-6 容器隧道网络模型 Master 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	UDP: 4789	所有IP地址 (0.0.0.0/0)	容器间网络互访。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 5444	VPC网段	kube-apiserver服务端端口，提供K8s资源的生命周期管理。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 5444	容器网段			
	TCP: 9443	VPC网段	Node节点网络插件访问Master节点。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 5443	所有IP地址 (0.0.0.0/0)	kube-apiserver的HAProxy负载均衡端口，负责将请求转发到kube-apiserver后端实例。	建议修改	端口需保留对VPC网段、容器网段和托管网格控制面网段放通。 说明 如果您需要使用CloudShell功能，请保留5443端口对198.19.0.0/16网段放通，否则将无法访问集群。
	TCP: 8445	VPC网段	Node节点存储插件访问Master节点。	不建议修改	如果修改会影响集群的正常运行。
	全部	Master节点安全组	限制Master节点安全组外的访问，但对于Master节点安全组中的实例互相访问不做限制。	不建议修改	如果修改会影响集群的正常运行。
出方向规则	全部	所有IP地址 (0.0.0.0/0)	默认全部放通。	不建议修改	如果修改会影响集群的正常运行。

云原生网络 2.0（CCE Turbo 集群）安全组规则

Node节点安全组

集群自动创建的Node节点安全组名称为{集群名}-cce-node-{随机ID}，默认端口说明请参见表7-7。

表 7-7 CCE Turbo 集群 Node 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	TCP: 10250	Master节点网段	Master节点主动访问Node节点的kubelet（如执行kubectl exec {pod}）。	不建议修改	如果修改会影响集群的正常运行。
	TCP: 30000-32767	所有IP地址（0.0.0.0/0）	集群NodePort服务默认访问端口范围。	可修改	端口需对VPC网段、容器网段和ELB的网段放通。
	UDP: 30000-32767				
	TCP: 22	所有IP地址（0.0.0.0/0）	允许SSH远程连接Linux弹性云服务器。	建议修改	建议修改成只允许固定IP或IP地址段访问。
	全部	Node节点安全组	限制Node节点安全组外的访问，但对于Node节点安全组中的实例互相访问不做限制。	不建议修改	如果修改会影响集群的正常运行。
	全部	容器子网网段	允许集群中的容器访问节点。	不建议修改	如果修改会影响集群的正常运行。
出方向规则	全部	所有IP地址（0.0.0.0/0）	默认全部放通，通常情况下不建议修改。	可修改	如需加固出方向规则，请注意指定端口需要放通，详情请参见 安全组出方向规则加固建议 。

Master节点安全组

集群自动创建的Master节点安全组名称为{集群名}-cce-control-{随机ID}，默认端口说明请参见表7-8。

表 7-8 CCE Turbo 集群 Master 节点安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	TCP: 5444	所有IP地址 (0.0.0.0/0)	kube-apiserver服 务端口，提供K8s资 源的生命周期管 理。	不建 议修 改	如果修改会影响集 群的正常运行。
	TCP: 5444	VPC网段		不建 议修 改	如果修改会影响集 群的正常运行。
	TCP: 9443	VPC网段	Node节点网络插件 访问Master节点。	不建 议修 改	如果修改会影响集 群的正常运行。
	TCP: 5443	所有IP地址 (0.0.0.0/0)	kube-apiserver的 HAProxy负载均衡 端口，负责将请求 转发到kube- apiserver后端实 例。	建议 修改	端口需保留对VPC 网段、容器网段和 托管网格控制面网 段放通。 说明 如果您需要使用 CloudShell功能， 请保留5443端口对 198.19.0.0/16网段 放通，否则将无法 访问集群。
	TCP: 8445	VPC网段	Node节点存储插件 访问Master节点。	不建 议修 改	如果修改会影响集 群的正常运行。
	全部	Master节 点安全组	限制Master节点安 全组外的访问，但 对于Master节点安 全组中的实例互相 访问不做限制。	不建 议修 改	如果修改会影响集 群的正常运行。
	全部	容器子网 网段	属于容器子网网段 的源地址需全部放 通。	不建 议修 改	如果修改会影响集 群的正常运行。
出方向规则	全部	所有IP地址 (0.0.0.0/0)	默认全部放通。	不建 议修 改	如果修改会影响集 群的正常运行。

ENI安全组

CCE Turbo集群会额外创建名为{集群名}-cce-eni-{随机ID}的安全组，默认为集群中的容器绑定该安全组，默认端口说明请参见表7-9。

表 7-9 ENI 安全组默认端口说明

方向	端口	默认源地址	说明	修改建议	修改后影响
入方向规则	全部	ENI安全组	允许集群中的容器互相访问。	不建议修改	如果修改会影响集群的正常运行。
		VPC网段	允许集群VPC中的实例访问容器。	不建议修改	如果修改会影响集群的正常运行。
出方向规则	全部	所有IP地址 (0.0.0.0/0)	默认全部放通。	不建议修改	如果修改会影响集群的正常运行。

安全组出方向规则加固建议

对于**出方向规则**，CCE创建的安全组默认全部放通，通常情况下不建议修改。如需加固出方向规则，请注意如下端口需要放通。

表 7-10 Node 节点安全组出方向规则最小范围

端口	放通地址段	说明
TCP: 53	子网的DNS服务器	用于域名解析。
UDP: 53		
TCP: 5353	容器网段	用于CoreDNS域名解析。
UDP: 5353		
UDP: 4789 (仅容器隧道网络模型的集群需要)	所有IP地址	容器间网络互访。
TCP: 5443	Master节点网段	kube-apiserver的HAProxy负载均衡端口，负责将请求转发到kube-apiserver后端实例。
TCP: 5444	VPC网段、容器网段	kube-apiserver服务端口，提供K8s资源的生命周期管理。
TCP: 6443	Master节点网段	-
TCP: 8445	VPC网段	Node节点存储插件访问Master节点。
TCP: 9443	VPC网段	Node节点网络插件访问Master节点。

端口	放通地址段	说明
所有端口	198.19.128.0/17网段	访问VPCEP服务。
UDP: 123	100.125.0.0/16网段	Node节点访问内网NTP服务器端口。
TCP: 443	100.125.0.0/16网段	Node节点访问内网OBS端口用于拉取安装包。
TCP: 6443	100.125.0.0/16网段	Node节点上报节点安装成功。

7.2.8 创建 CCE Turbo 集群时如何设置 IPv6 服务网段

问题背景

当您需要创建一个IPv4/IPv6双栈的CCE Turbo集群时，需要设置IPv6服务网段，该网段默认值为fc00::/112，包含了65536个IPv6服务地址。如果您需要自定义服务网段，您可参考本文进行设置。

IPv6 介绍

IPv6地址

IPv6地址采用128位二进制表示，是IPv4地址长度的4倍。因此IPv4地址的十进制格式不再适用，IPv6采用了十六进制来表示，将128位二进制数转换为32位十六进制数，每4位十六进制数（不区分大小写）为一组，每组以冒号“:”隔开，可以分为8组。

IPv6地址存在多种省略写法：

- 0位省略：如果每个冒号分组中存在以0开头的，则可以将0位省略，多个0连续时可省略多个。例如以下IPv6地址均是等价的。
 - ff01:0d28:03ee:0000:0000:0000:0c23
 - ff01:d28:3ee:0000:0000:0000:c23
 - ff01:d28:3ee:0:0:0:c23
- 双冒号省略：如果以十六进制表示的IPv6地址中间依然存在很多个全为0的分组，可以把连续全为0的分组压缩成双冒号“::”。但为保证唯一性，这种压缩方式只能使用一次，即一个IPv6地址中只能出现一次双冒号“::”。

例如：

双冒号省略前	双冒号省略后
ff01:d28:3ee:0:0:0:c23	ff01:d28:3ee::c23
0:0:0:0:0:0:1	::1
0:0:0:0:0:0:0	::

IPv6地址段

IPv6地址段通常采用CIDR（无类别域间路由选择）表示法，通常用斜杠（/）后跟一个数字表示，即格式为“IPv6地址/前缀长度”。此处前缀长度与IPv4地址段的掩码作用类似，用数字来表示网络部分所占用的二进制位数，可将IPv6地址分为网络地址和主机地址两部分。而前缀长度指定了网络部分占用的位数，剩余位数则是主机地址部分，可以更加方便和灵活地表示不同的地址段。

例如，fc00:d28::/32表示一个前缀长度为32位的IPv6地址段，则在该网段中分配地址时，前32位（以二进制计算，此处即为fc00:d28）为网络地址，后96位则为可用的主机地址。

IPv6 服务网段使用约束

在设置集群服务网段时，需要首先考虑以下有如下使用约束：

- IPv6服务网段必须属于fc00::/8网段内。
该地址属于本地唯一地址（ULA）网段。ULA拥有固定前缀：fc00::/7，其中包括fc00::/8和fd00::/8两个范围，类似于IPv4的专用网络地址10.0.0.0/8、172.16.0.0/12和192.168.0.0/16，相当于私有IP网段，仅能够在本地网络使用。
- 前缀范围112-120，您可以通过调整前缀数值，调整地址个数，地址数最多有65536个。

IPv6 服务网段示例

根据约束，本文中提供一个包含8192个地址的IPv6网段设置示例，供您参考。

步骤1 根据地址数需求设定前缀长度，且前缀范围为112-120。

本例中，需要8192个地址数，8192个地址需要13位二进制数表示，而IPv6地址总长度为128位二进制数，则该IPv6网段的前缀长度为128-13=115，表示前115位可用于区分网络地址段，后13位用于表示8192个主机地址。

根据以上计算方式，可得出前缀范围为112-120的IPv6地址段所包含的地址数量如下：

IPv6地址段的前缀长度	包含的地址数量
112	65536
113	32768
114	16384
115	8192
116	4096
117	2048
118	1024
119	512
120	256

步骤2 设置IPv6网络地址，且网络地址必须属于fc00::/8网段内。

本例中，确定前缀长度为115，由于网络地址必须属于fc00::/8网段内，因此前8位二进制数是固定的。可修改的网络地址范围是第9位至第115位，第116位至第128位则属于主机地址。

将IPv6地址写成二进制形式，则根据以上条件：

- 网络地址必须属于fc00::/8网段，因此二进制中的前8位不可修改，否则将不属于fc00::/8网段，固定为1111 1110，对应十六进制为fc。
- 包含8192个地址数的网段前缀长度设置为115，因此二进制中后13位用于表示主机地址，固定全为0。

具体示例如下，其二进制中标红部分是不可修改的。

```
二进制： 1111 1100 **** ** ... ***0 0000 0000 0000/115
          | | | | | | | |
十六进制： f c x x ... y 0 0 0/115
```

其中x为任意十六进制数；而y对应的4位二进制数最后一位固定为0，因此十六进制数y可选范围为0、2、4、6、8、a、c、e。

----结束

7.2.9 CCE 集群的节点是否支持绑定多网卡？

CCE集群的节点不支持绑定多网卡，请勿手动绑定多网卡，否则会对集群的网络访问造成影响。

7.3 安全加固

7.3.1 集群节点如何不暴露到公网？

问题描述：

集群节点如何不暴露到公网？

问题解决：

- 如果不需要访问集群节点的22端口，可在安全组规则中禁用22端口的访问。
- 如非必须，集群节点不建议绑定EIP。

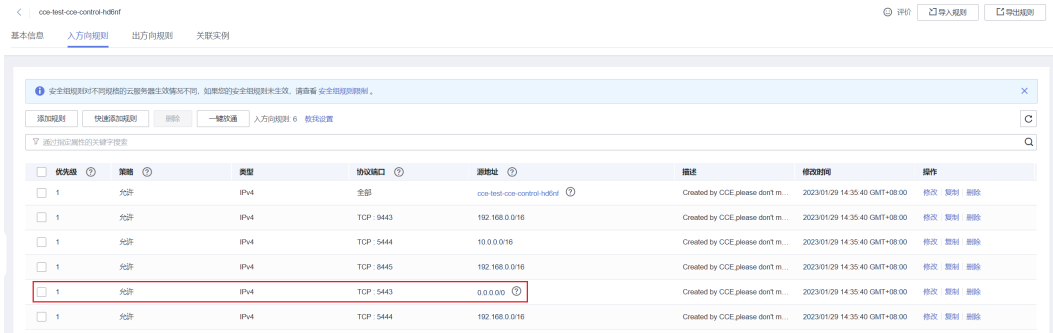
如有远程登录集群节点的需求，推荐使用华为云堡垒机服务作为中转连接集群节点。

7.3.2 如何配置集群的访问策略

为集群绑定公网API Server地址后，建议修改控制节点5443端口的安全组规则，加固集群的访问控制策略。

- 步骤1** 登录[CCE控制台](#)，单击集群名称进入集群，在“概览”页面找到“集群ID”并复制。
- 步骤2** 登录[VPC控制台](#)，在左侧导航栏中选择“访问控制 > 安全组”。
- 步骤3** 在筛选栏中，选择筛选条件为“描述”，并粘贴集群ID进行筛选。
- 步骤4** 筛选结果中将会包含多个安全组，找到控制节点的安全组（以[cce集群名称]-cce-control开头），单击“配置规则”。

步骤5 修改入方向的5443端口规则，单击“修改”。

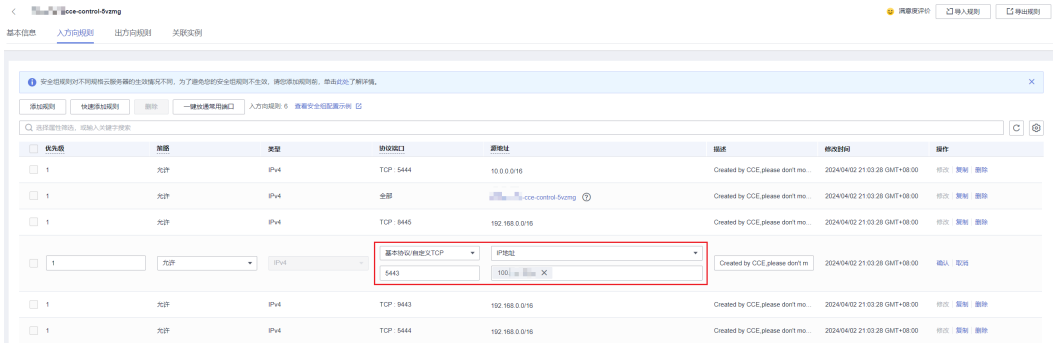


步骤6 根据需求修改允许访问的源地址。例如，客户端需要访问API Server的IP为100.*.*，则可添加5443端口入方向规则的源地址为100.*.*。

说明

除客户端IP外，端口还需保留对VPC网段、容器网段和托管网路控制面网段放通，以保证集群内部可访问API Server。

如果您需要使用CloudShell功能，请保留5443端口对198.19.0.0/16网段放通，否则将无法访问集群。



步骤7 修改完成后单击“确认”。

----结束

7.3.3 如何获取 TLS 密钥证书？

场景

当您的Ingress需要使用HTTPS协议时，创建Ingress时必须配置IngressTLS或kubernetes.io/tls类型的密钥。

以创建IngressTLS密钥证书为例。如图7-12：

图 7-12 创建密钥

创建密钥

名称

test

命名空间

default

描述

请输入描述信息

0/255

密钥类型

IngressTLS

存放7层负载均衡服务所需的证书

密钥数据

★ 证书文件 ?

-----BEGIN CERTIFICATE-----
MIIF/DCCBOSgAwIBAgIQB1uyWgQ9RnmTCsnlcmxLsTANBgkqhkiG
9w0BAQsFADBu
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSWW

上传 样例参考

★ 私钥文件 ?

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAf/ro4qzQjwLHdCEbD2kW9aoxllCZ6JuD0NrvS
1Ze3pnYcBr
Pskxi65ffOILdDsW9BhKJ41Kj6aA3szFEyRzVf3wJ01RVyeGgLNPeV6

上传 样例参考

标签

键 = 值

确认添加

创建密钥时密钥数据中上传的证书文件和私钥文件必须是配套的，不然会出现无效的情况。

解决方法

一般情况下，您需要从证书提供商处获取有效的合法证书。如果您需要在测试环境下使用，您可以自建证书和私钥，方法如下：

说明

自建的证书通常只适用于测试场景，使用时界面会提示证书不合法，影响正常访问，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。

1. 自己生成tls.key。
openssl genrsa -out tls.key 2048
将在当前目录生成一个tls.key的私钥。

2. 用此私钥去签发生成自己的证书。
openssl req -new -x509 -key tls.key -out tls.crt -subj /C=CN/ST=*****/O=Devops/CN=example.com -days 3650
生成的私钥格式必须为：
-----BEGIN RSA PRIVATE KEY-----
.....
-----END RSA PRIVATE KEY-----
生成的证书格式必须为：
-----BEGIN CERTIFICATE-----
.....
-----END CERTIFICATE-----

文档版本 01 (2025-09-15)

版权所有 © 华为云计算技术有限公司

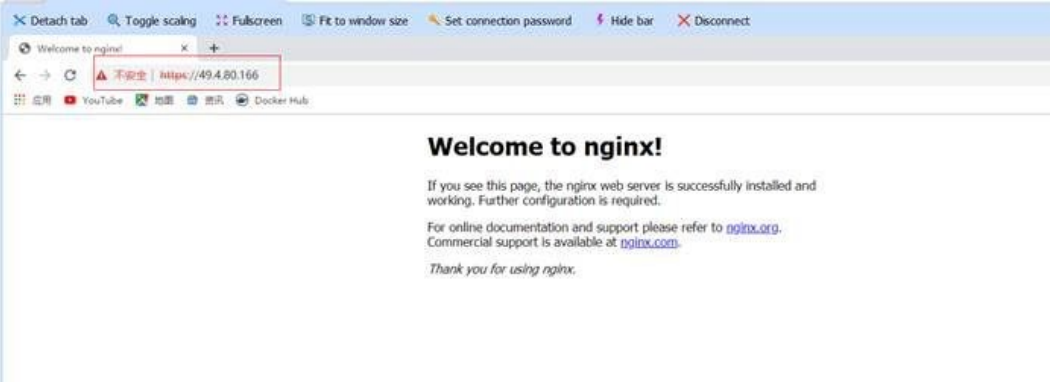
230

3. 导入证书。
- 新建TLS密钥时，对应位置导入证书及私钥文件即可。

验证

通过浏览器访问Ingress地址可以正常访问，但因为是自己签发的证书和密钥，所以CA不认可，显示不安全。

图 7-13 验证结果



7.3.4 如何批量修改集群 node 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](#)。

操作步骤

- 步骤1 登录VPC控制台，并在左上角选择区域和项目。
- 步骤2 在左侧导航树选择“访问控制 > 安全组”。
- 步骤3 在安全组界面，单击操作列的“管理实例”。
- 步骤4 在“服务器”页签，并单击“添加”。



- 步骤5 勾选需要加入安全组的服务器，单击“确定”。您也可以通过服务器的名称、ID、私有IP地址、状态、企业项目或标签进行筛选。

通过修改左下角的单页最大显示条数，您可至多一次性添加20台服务器至安全组中。

说明

加入新的安全组后，节点仍保留原安全组。如需移除，请单击原安全组的“管理实例”按钮，并勾选其中的节点服务器进行移除。

添加服务器

安全组

选择服务器

云服务器

裸金属服务器

通过指定属性的关键字搜索

名称	状态	私有IP地址	IPv6地址	企业项目
ID	关机	192.168.0.70	--	default
私有IP地址	关机	192.168.0.85	--	default
状态	关机	192.168.48.116	--	default
企业项目	关机	192.168.0.160	--	default
标签	运行中	192.168.0.38	--	default

5 总条数: 12 < 1 2 3 >

已选服务器
您还没选择服务器。

----结束

7.4 网络指导

7.4.1 CCE 如何与其他服务进行内网通信？

与CCE进行内网通信的华为云常见服务有：RDS、DMS、Kafka、RabbitMQ、VPN、ModelArts等，有如下两种场景：

- 在同一个VPC网络下，CCE节点可以与此VPC下的所有服务进行互通。CCE的容器与其他服务通信时，需要关注对端是否开启了容器网段的入方向的安全组规则（此限制只针对于CCE的集群为VPC网络模式）。
- 如果不在同一个VPC，可以使用“对等连接”或VPN的方式打通两个VPC，不过需要注意的是两个VPC的网段及容器网段不能重复。并且，对端VPC或小网需要配置回程路由（此限制只针对于CCE的集群为VPC网络模式），配置方法请参见[对等连接](#)。

须知

- 此逻辑针对于华为云所有服务均有效。
- “容器隧道网络”的集群，天然支持各服务间内网通信，不需要另外配置。
- “VPC网络”模型的集群需要注意的事项：
 1. 对端看到的来源IP为容器IP。
 2. 容器在VPC内的节点上互通，是通过CCE添加的自定义路由规则实现的。
 3. CCE中的容器访问其他服务时，需要关注对端（目的端）是否开启了容器网段入方向的安全组规则或防火墙。具体请参见[安全组配置示例](#)。
 4. 如果使用VPN或“对等连接”等方式打通了小网通信，需要在中间路上和目的地，都需要在对等连接添加容器网段的路由。
- “云原生网络2.0”模型的集群需要根据业务诉求放通容器关联的安全组，容器关联的默认安全组名称为{集群名}-cce-eni-{随机ID}，具体请参见[云原生网络2.0（CCE Turbo集群）安全组规则](#)。

7.4.2 使用 CCE 设置工作负载访问方式时，端口如何填写？

CCE支持工作负载的内部互访和被互联网访问两种方式。

- 内部访问：包括集群内访问（通过集群虚拟IP）和节点访问（通过节点私有IP）两种方式。

表 7-11 内部访问类型说明

内部访问类型	说明	端口如何填写
集群内访问 （通过集群虚拟IP）	用于“工作负载之间的互访”，例如某个后端工作负载需要和前端工作负载互访，就需要选择该类型来实现内部互访。 集群虚拟IP，即Cluster IP，是在工作负载设置此访问类型后，就会自动分配一个可用的Cluster IP。	• 容器端口：指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。 • 服务端口：指该容器工作负载发布为服务后，所设定的服务端口号，请填写1-65535之间的整数值。在内部工作负载互访时，将通过“Cluster IP:访问端口”来访问。

内部访问类型	说明	端口如何填写
节点访问（通过节点私有IP）	可以通过“节点IP:节点端口”的形式访问工作负载。若该节点绑定了弹性IP，则外网就可以访问该工作负载。	- 容器端口：指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。 - 服务端口：指该容器工作负载发布为服务后，所设定的服务端口号，请填写1-65535之间的整数值。 - 节点端口：指容器映射到节点上的端口。配置完成后，系统会在用户所在项目的所有节点上打开一个真实的端口号。访问工作负载时可以通过“节点IP:节点端口”来访问工作负载。 如无特殊需求，选择“自动生成”即可，系统会自动分配访问端口号。若选择“指定端口”，请填写30000-32767之间的整数，且确保集群内该值唯一。

- 外部访问：包括节点访问（通过弹性IP）、负载均衡和DNAT网关三种方式。

表 7-12 外部访问类型说明

外部访问类型	说明	端口如何填写
节点访问（通过弹性IP）	为节点绑定弹性IP，访问工作负载时，通过“节点弹性IP:节点端口”的形式访问工作负载。工作负载可被公网访问。	- 容器端口：指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。 - 服务端口：指该容器工作负载发布为服务后，所设定的服务端口号，请填写1-65535之间的整数值。 - 节点端口：指容器映射到节点上的端口。配置完成后，系统会在用户所在项目的所有节点上打开一个真实的端口号。访问工作负载时可以通过“节点IP:节点端口”来访问工作负载。 如无特殊需求，选择“自动生成”即可，系统会自动分配访问端口号。若选择“指定端口”，请填写30000-32767之间的整数，且确保集群内该值唯一。

外部访问类型	说明	端口如何填写
负载均衡	负载均衡通过将访问流量自动分发到多台节点，扩展工作负载系统对外的服务能力，实现更水平的工作负载程序容错性能。 您需要提前创建负载均衡实例，并在CCE访问类型中选择负载均衡实例。	- 容器端口：指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。 - 服务端口：代表负载均衡上注册的对外端口，请填写1-65535之间的整数值。外部用户使用“ELB的VIP:服务端口”访问工作负载。
DNAT网关	NAT网关提供网络地址转换服务，使多个云服务器可以共享弹性公网IP。 您需要提前创建公网NAT网关实例。	- 容器端口：指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。 - 服务端口：代表NAT网关上注册的对外端口，请填写1-65535之间的整数值。系统会自动创建DNAT规则，外部用户使用“网关的弹性IP:服务端口”访问工作负载。

7.4.3 Ingress 中的 property 字段如何实现与社区 client-go 兼容？

使用场景

社区Ingress结构体中没有property属性，导致用户使用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提供了如下解决方案。

解决方案

在使用client-go创建Ingress实例时，在annotation中做如下声明：

```
kubernetes.io/ingress.property: '[{"host":"test.com","path":"/test","matchmode":"STARTS_WITH"},  
{"host":"test.com","path":"/dw","matchmode":"EQUAL_TO"}]
```

匹配规则：用户调用CCE的Kubernetes接口创建Ingress时，会试图匹配Ingress rules中的host和path两个字段，如果和annotation中的host和path一致，则会在这条path下面注入property属性，示例如下：

```
kind: Ingress  
apiVersion: extensions/v1beta1  
metadata:  
  name: test  
  namespace: default  
  resourceVersion: '2904229'  
  generation: 1  
  labels:  
    isExternal: 'true'  
    zone: data  
  annotations:  
    kubernetes.io/ingress.class: cce  
    kubernetes.io/ingress.property: '[{"host":"test.com","path":"/test","matchmode":"STARTS_WITH"},  
{"Path":"/dw","MatchMode":"EQUAL_TO"}]'  
spec:
```



```
rules:
  - host: test.com
    http:
      paths:
        - path: /ss
          backend:
            serviceName: zlh-test
            servicePort: 80
        - path: /dw
          backend:
            serviceName: zlh-test
            servicePort: 80
```

转换后的格式如下：

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: test
  namespace: default
  resourceVersion: '2904229'
  generation: 1
  labels:
    isExternal: 'true'
    zone: data
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/ingress.property: '[{"host":"test.com","path":"/ss","matchmode":"STARTS_WITH"},
{"host":"","path":"/dw","matchmode":"EQUAL_TO"}]'
spec:
  rules:
    - host: test.com
      http:
        paths:
          - path: /ss
            backend:
              serviceName: zlh-test
              servicePort: 80
          - path: /dw
            backend:
              serviceName: zlh-test
              servicePort: 80
      property:
        ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
```

表 7-13 关键参数说明

参数	参数类型	描述
host	String	域名配置。 若不配置，会自动匹配path。
path	String	匹配路径。
ingress.beta.kubernetes.io/url-match-mode	String	路由匹配策略，取值如下： ● REGEX：正则匹配 ● STARTS_WITH：前缀匹配 ● EQUAL_TO：精确匹配

相关链接

[七层负载均衡（Ingress）](#)

7.4.4 服务加入 Istio 后，如何获取客户端真实源 IP？

问题现象

服务启用Istio后，访问日志中无法获取到客户端源IP。

解决方案

本文以绑定ELB类型Service的nginx应用为例，详细步骤如下：

步骤1 ELB侧开启获取客户端IP

说明

独享型ELB默认开启源地址透传功能，无需手动开启。

1. 在管理控制台左上角单击  图标，选择区域和项目。
2. 选择“服务列表 > 网络 > 弹性负载均衡 ELB”。
3. 在“弹性负载均衡器”界面，单击需要操作的负载均衡名称。
4. 切换到“监听器”页签，单击需要修改的监听器名称右侧的“编辑”按钮。如果存在修改保护，请在监听器基本信息页面中关闭修改保护后重试。
5. 开启“获取客户端IP”开关。

图 7-14 开启开关



步骤2 更新服务所关联的网关

1. 登录CCE控制台，单击集群名称进入集群，在左侧选择“服务发现”。
2. 在“服务发现”页面，切换至istio-system命名空间，并更新服务所关联的网关服务。



3. 将istio-system命名空间下自动生成的Service改为节点级别。



步骤3 验证获取源IP

1. 使用kubectl连接集群。
2. 查看nginx应用日志。
`kubectl logs <pod_name>`

本示例中，nginx应用获取到的源IP如下：

```
2023/04/11 16:56:18 [notice] 181: using the "epoll" event method
2023/04/11 16:56:18 [notice] 181: nginx/1.25.4
2023/04/11 16:56:18 [notice] 181: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/04/11 16:56:18 [notice] 181: OS: Linux 4.18.0-147.el1.elrepo.x86_64
2023/04/11 16:56:18 [notice] 181: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/04/11 16:56:18 [notice] 181: start worker processes
2023/04/11 16:56:18 [notice] 181: start worker process 30
2023/04/11 16:56:18 [notice] 181: start worker process 31
2023/04/11 16:56:18 [notice] 181: start worker process 32
2023/04/11 16:56:18 [notice] 181: start worker process 33
192.0.0.6 - - [11/Apr/2023:16:56:11 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.34" "-"
192.0.0.6 - - [11/Apr/2023:16:56:13 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.34" "-"
192.0.0.6 - - [11/Apr/2023:16:56:14 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.34" "-"
192.0.0.6 - - [11/Apr/2023:16:56:15 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.34" "-"
192.0.0.6 - - [11/Apr/2023:16:56:16 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.34" "-"
```

----结束

7.4.5 为什么更换命名空间后无法创建 Ingress?

问题描述

在default命名空间下可以正常创建ingress，但在其他命名空间（如:ns）下创建时不能创建成功。

原因分析

创建弹性负载均衡ELB后，使用default命名空间创建80端口的http监听，在CCE中只允许在本命名空间下创建同一端口的其它ingress（实际转发策略可根据域名、service来区分）；所以出现客户侧在其它命名空间无法创建相同端口的ingress的情况（会提示端口冲突）。

解决方法

可以使用yaml创建，端口冲突只有前台限制，后台未限制。

7.4.6 服务发布到 ELB，ELB 的后端为何会被自动删除？

问题描述：

服务发布到ELB，工作负载已正常，但服务的pod端口未及时发布出来，ELB里的后端会被自动删除。

问题解答：

1. 创建ELB时候，如果ELB健康检查失败，后端服务器组会删除，而且后续服务正常以后也不会添加。如果是更新已有的SVC时则不会删除。

2. 添加删除节点的时候，由于集群状态的改变，可能会引起集群内的Node访问方式的改变，为保证服务正常运行，所以ELB会进行一次刷新操作，这个过程类似于更新ELB。

修复建议：

优化应用，加快应用的启动速度。

7.4.7 如何使容器重启后所在容器 IP 仍保持不变？

单节点场景

如果集群下仅有1个节点时，要使容器重启后所在容器IP保持不变，需在工作负载中配置主机网络，在工作负载的yaml中的spec.spec.下加入hostNetwork: true字段。

多节点场景

如果集群下有多个节点时，除进行以上操作外，还需要设置节点的亲和策略，但工作负载创建后实例运行数不得超过亲和的节点数。

完成效果

进行如上设置并在工作负载运行后，工作负载实例ip与节点ip将保持一致，重启工作负载后ip也将保持不变。

7.4.8 如何确认网卡不被集群占用？

操作场景

在CCE Turbo集群中，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。

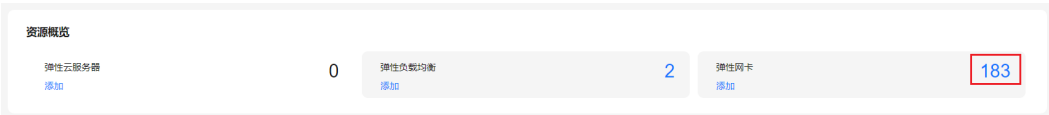
删除集群容器子网属于高危操作，您需要确保当前集群正在使用的网卡中没有网卡属于该子网，包含Pod正在使用和集群预热的网卡。

操作步骤

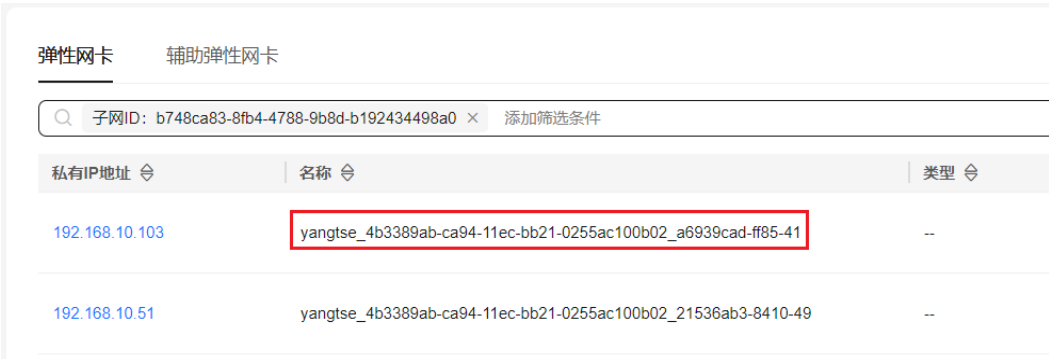
- 步骤1** 登录CCE控制台，单击集群列表中的集群名称。
- 步骤2** 在左侧导航栏中选择“配置中心”，切换至“网络配置”页签。
- 步骤3** 查看“容器网络配置”，以default-network（默认容器子网）为例，复制容器子网的“网络ID”。



- 步骤4** 登录VPC控制台，在左侧导航栏中选择“子网”，并根据容器子网的“网络ID”进行过滤，找到对应的子网。
- 步骤5** 单击进入子网，选择“基本信息”页签，在“资源概览”中单击进入弹性网卡，查看该子网下的“弹性网卡”和“辅助弹性网卡”。



- 步骤6** 查看网卡“名称”或者“描述”，如果其中包含当前集群的ID，表示网卡被集群占用。您可以在CCE控制台的集群“概览”页中复制集群ID。
- 如果需要清理集群内使用的子网网卡，需要提交工单处理。



----结束

7.4.9 删除子网后如何删除安全组规则？

操作场景

在CCE Turbo集群中，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。

删除子网后，CCE默认生成的节点级的安全组不会自动清理待删除子网的安全组规则，需要手动清理。

操作步骤

- 步骤1 登录CCE控制台，单击集群列表中的集群名称。
- 步骤2 在左侧导航栏中选择“配置中心”，切换至“网络配置”页签。
- 步骤3 查看“容器网络配置”，以default-network（默认容器子网）为例，复制容器子网的“IPv4网段”。



- 步骤4 在左侧导航栏中选择“概览”，查看“网络信息”，单击“节点默认安全组”，跳转到安全组页面。
- 步骤5 选择“入方向规则”，并根据源地址过滤容器子网网段，找到对应的安全组规则。



- 步骤6 单击“删除”，删除关联的子网安全组规则。

----结束

7.4.10 不同命名空间下的 Ingress 共用监听器时如何同步生效的证书？

问题背景

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。

但是不同命名空间下的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。

例如，两个Ingress配置如下：

Ingress名称	ingress1	ingress2

命名空间	namespace1	namespace2
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
证书来源	TLS密钥	TLS密钥
TLS密钥对应secret	namespace1/secret1	namespace2/secret2
实际生效的证书	namespace1/secret1	namespace1/secret1

问题现象

在同一个集群中，在namespace1创建ingress1，在namespace2创建ingress2，两个Ingress对接同一个监听器，使用了TLS密钥类型的证书。

由于ingress1创建时间早于ingress2，所以实际生效证书以ingress1为准。但ingress2所在命名空间不是namespace1，无法读到secret1的配置，在ingress2的配置页面将出现以下提示。

更新路由

名称

ingress2

命名空间

namespace2

负载均衡器

testnotdel

我已阅读《负载均衡使用须知》

监听器配置

前端协议

HTTPS

对外端口

443

访问控制

未配置

证书来源

TLS 密钥

服务器证书

当前监听器已配置服务器证书，无法修改，证书对应 secret_id 为 40a0773d-b874-4739-a490-3565f1ffa689。[监听器详情](#)

解决方案

由于每一个TLS密钥对应ELB的一个证书，且密钥内容相同，ELB证书信息的获取是使用CCE委托权限，获取不受命名空间约束。因此可以通过修改ingress1的证书来源为服务器证书，并配置证书为TLS密钥对应的ELB证书，在ingress2的配置修改页面可以看到实际生效的服务器证书。

- 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ingress1路由对应的负载均衡器链接跳转到ELB页面。

- 步骤3** 切换至“监听器”页签，根据ingress1配置的端口，在负载均衡器页面找到对应的监听器，并单击监听器名称进入详情。
- 步骤4** 在监听器详情页面，找到服务器证书并记录。



- 步骤5** 在CCE控制台中，单击ingress1的“更多 > 更新”，选择证书来源为ELB服务器证书，并配置为上一步骤查询到的服务器证书，单击“确定”更新Ingress配置。
- ingress1的证书来源由TLS密钥更换为服务器证书，且更换前后密钥内容一致，实际生效的配置在更换前后不变。



- 步骤6** 切换至namespace2，单击ingress2的“更多 > 更新”，单击服务器证书下方的“同步”按钮，并单击“确定”更新Ingress配置。



- 步骤7** 更新完成后ingress2的配置显示正常。
- 结束

7.4.11 如何确认监听器配置生效的 Ingress

CCE支持将多个Ingress对接到同一个ELB的监听器，并创建不同的转发策略。由于监听器配置参数通过annotation方式承载，因此可能存在同一个监听器配置在多个Ingress上有不同配置参数的场景。本文为您介绍如何确认监听器配置生效Ingress。

如何确认监听器配置生效的 Ingress

由于对接同监听器的路由（Ingress）均可配置监听器参数，因此CCE采取首个创建的Ingress（简称首路由）上的annotation监听器配置（除SNI证书外）。首路由为通过Ingress的metadata.createTimestamp字段进行升序排列后的第一条路由。

- v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0、v1.29.1-r0及以上版本集群将首路由信息写入了annotation，可以查看已有ingress的annotation中的kubernetes.io/elb.listener-master-ingress确认：

```
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: ingress-first
5    namespace: default
6    uid: 43b57afc-7f55-4310-ac1a-ac8afdd3d5fd
7    resourceVersion: '1568102'
8    generation: 1
9    creationTimestamp: '2024-09-09T02:31:07Z'
10  annotations:
11    kubernetes.io/elb.class: performance
12    kubernetes.io/elb.id: be56202a-c2cb-40d5-900e-d7a007a4b054
13    kubernetes.io/elb.listener-master-ingress: default/ingress-first
14    kubernetes.io/elb.port: 443
15    kubernetes.io/elb.tls-certificate-ids: 87e311e965db421ca806c151368c01ca,8f47921346e74aa58ba38660127e5967
16    kubernetes.io/elb.tls-ciphers-policy: tls-1-2-fs
17  managedFields:
```

- v1.21.15-r0、v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0、v1.29.1-r0以下的老版本集群需通过kubectl命令，获取关联同一ELB监听器下的ingress，并按时间顺序进行升序排列，第一条即为首路由。

快速查询命令如下，请替换想要查询的ELB ID和端口。

```
elb_id=${1}
elb_port=${2}
kubectl get ingress --all-namespaces --sort-by='metadata.creationTimestamp' -o=custom-columns=Name:'metadata.name',Namespace:'metadata.namespace',elbID:'metadata.annotations.kubernetes.io/elb.id',elbPort:'metadata.annotations.kubernetes.io/elb.port',elbPorts:'metadata.annotations.kubernetes.io/elb.listen-ports' | awk 'NR==1 {print; next} {if ($5 != "<none>") $4 = "<none>"; print} | grep -E "^Name|${elb_id}" | grep -E "^Name|${elb_port}" | awk '{printf "%-30s %-30s %-38s %-10s %-10s\n", $1, $2, $3, $4, $5}'
```

输出如下：第一列为路由名称，第二列为路由命名空间，第三列为ELB ID，第四列为监听器端口号，第五列为多监听器端口号。（设置有多监听器端口号时，替代监听器端口号生效）

Name	Namespace	elbID	elbPort	elbPorts
ingress-first	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-second	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-third	test	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>

监听器证书配置及更新说明

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：由Secret承载证书内容，证书内容在CCE侧维护，并自动在ELB侧进行证书的创建、更新或删除。配置在Ingress的spec.tls字段下。
- 使用ELB服务中的证书：直接使用ELB服务中创建的证书，证书内容在ELB侧维护。配置在Ingress的annotation字段下。

ELB服务器证书将证书维护在ELB侧，无需将证书内容导入Secret，实现了跨命名空间配置的统一。因此建议通过ELB服务器证书方式来为Ingress配置证书。

ELB服务器证书支持集群版本：v1.19.16-r2、v1.21.5-r0、v1.23.3-r0。

如果您的Ingress使用TLS密钥方式创建监听器服务器证书，请按照如下步骤进行证书更新。

1. 查看[如何获取首路由](#)中的快速查询命令，获取对接同一监听器的所有路由。
回显示例如下：

Name	Namespace	elbID	elbPort	elbPorts
ingress-first	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-second	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-third	test	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>

2. 获取首路由中证书配置信息。

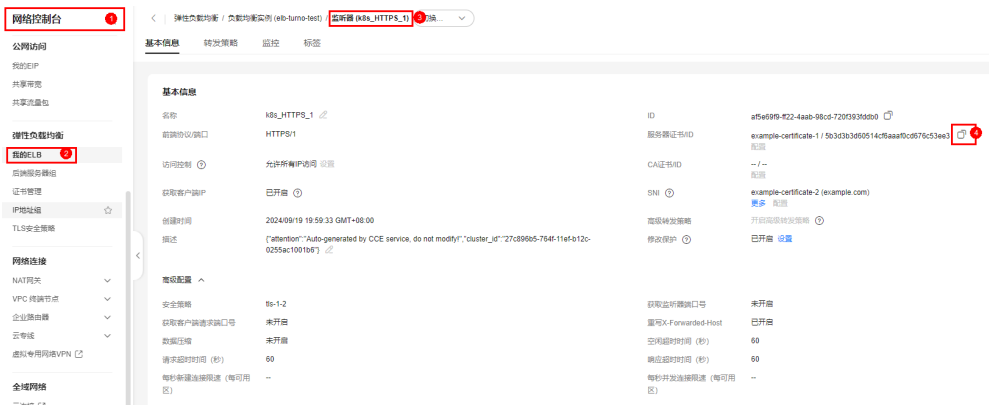
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-first
  namespace: default
...
spec:
  tls:
    - secretName: default-ns-secret-1
    - hosts:
        - 'example.com'
      secretName: default-ns-secret-2
...
```

3. 更新首路由中配置的default-ns-secret-1和default-ns-secret-2密钥。
4. 更新完密钥后，登录网络控制台，在左侧导航栏选择“弹性负载均衡 > 证书管理”，通过查看证书的更新时间确认服务器证书更新是否成功。



如果您的Ingress使用ELB方式创建监听器服务器证书，请在ELB侧直接修改证书内容。

1. 获取ELB监听器的服务器证书ID。
 - a. 登录网络控制台，在左侧导航栏选择“弹性负载均衡 > 我的ELB”，单击ELB名称进入ELB。
 - b. 选择“监听器”页签，单击监听器名称进入 监听器。
 - c. 在“基本信息”页签，获取服务器证书ID。



2. 在左侧导航栏选择“弹性负载均衡 > 证书管理”，通过获取的服务器证书ID查询证书，在操作列修改服务器证书。



首路由删除对监听器配置影响

监听器配置仅能在首路由配置上生效（SNI证书例外），首路由删除后，将顺延选取未删除的ingress中创建时间最早的成为首路由，并以新的首路由上的监听器配置进行更新。因此若新老首路由的监听器配置不一致，可能导致ELB侧非预期更新，请确认即将成为首路由的监听器配置是否与原首路由一致或是否符合预期。

- 您可以在控制台页面中进行监听器配置同步，步骤如下：
 - 登录CCE控制台，单击集群名称进入集群。
 - 选择左侧导航栏的“服务”，在右侧选择“路由”页签，单击对应路由“更多 > 更新”选项。
 - 当路由监听器配置与ELB不一致时，将提供同步选项。单击“点击同步”，即可自动同步服务器证书。

说明

在同步服务器证书和SNI证书时，如果当前路由使用“TLS密钥”，将被更改为ELB服务器证书。若集群为不支持ELB服务器证书的版本（低于v1.19.16-r2、v1.21.5-r0、v1.23.3-r0），则需[通过YAML手动同步](#)。

更新路由

名称 ingress-second

命名空间 default

负载均衡器 cce-lb-3fa3384b-341a-43e8-a765-06661ccta5af

☐ 我已阅读《负载均衡使用须知》

监听器配置

前端协议 HTTPS

对外端口 445

⚠ 当前监听器配置由default命名空间下的ingress-first路由维护，当前路由监听器配置与首路由存在差异，需手动同步首路由配置。 [点击同步](#)

访问控制 继承ELB已有配置

证书来源 ELB 服务器证书

服务器证书 example-certificate-2 [查看 ELB 证书](#)

SNI --请选择--

安全策略 安全策略 TLS-1-2

后端协议 HTTP

- d. 单击“确定”，下发更新配置。

- 通过YAML手动同步
 - a. 查看[如何获取首路由](#)中的快速查询命令，获取对接同一监听器的所有路由。

回显示例如下：

Name	Namespace	elbID	elbPort	elbPorts
ingress-first	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-second	default	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>
ingress-third	test	be56202a-c2cb-40d5-900e-d7a007a4b054	443	<none>

- b. 假设即将删除ingress-first和ingress-second，则需要将ingress-first的监听器配置同步至ingress-third的annotation中。

若监听器服务器证书是通过TLS密钥方式创建，需要将记录在ingress的spec.tls下的配置信息同步至ingress-third中。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-first
  namespace: default
...
spec:
  tls:
    - secretName: default-ns-secret-1
    - hosts:
      - 'example.com'
      secretName: default-ns-secret-2
...
```

7.4.12 如何扩展单个 CCE Turbo 集群能支持的最大网卡数？

CCE Turbo采用的是云原生网络2.0网络模式，Pod直接使用是从VPC分配出来的网卡，每个网卡默认关联该集群的[ENI安全组](#)。由于单个安全组关联的网卡数存在上限，默认值为5万。如果不做任何配置，假设没有预热的容器网卡，单个CCE Turbo集群最多只能运行5万个使用容器网络的Pod。

您可以使用以下方法扩展单集群能支持的最大网卡数：

安全组拆分方案	方案说明	约束&限制	实施方案
使用多个安全组横向分隔业务	将不同的安全组关联到不同的工作负载上，使用同一个安全组的Pod数需控制在5万以内。	● 预热的容器网卡不支持绑定目标的安全组。 ● 请合理规划安全组规则，确保不同工作负载之间必要的连通性。	● 使用安全组策略为工作负载绑定安全组 ● 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

安全组拆分方案	方案说明	约束&限制	实施方案
使用多个安全组纵向分隔业务	将不同的安全组关联到不同的节点池上，同一个节点池上创建的容器网卡使用对应的安全组（包括预热的容器网卡），同一个节点池上创建的容器网卡总数需控制在5万以内。 计算公式： 节点池最大节点数 * 对应机型的最大辅助弹性网卡数 < 5万	• 预热的容器网卡支持绑定目标的安全组。 • 请合理规划安全组规则，确保不同工作负载之间必要的连通性。	使用节点池配置为节点池上的Pod绑定默认的安全组

8 存储管理

8.1 如何扩容容器的存储空间？

使用场景

容器默认大小为10G，当容器中产生数据较多时，容易导致容器存储空间不足，可以通过此方法来扩容。

解决方案

- 步骤1** 登录CCE控制台，单击集群列表中的集群名称。
- 步骤2** 在左侧导航栏中选择“节点管理”。
- 步骤3** 切换至“节点”页签，选择集群中的节点，单击操作列中的“更多 > 重置节点”。

须知

重置节点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4** 重新配置节点参数。

如需对容器存储空间进行调整，请重点关注以下配置。

存储配置：单击数据盘后方的“展开高级配置”可进行如下设置：

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。

更多关于容器存储空间分配的内容，请参考[数据盘空间分配说明](#)。

- 步骤5** 重置节点后登录该节点，查看容器容量是否已扩容。容器存储Rootfs不同回显结果也不同，具体如下。

- Overlayfs：没有单独划分thinpool，在dockersys空间下统一存储镜像相关数据。执行以下代码，查看容器容量是否扩容成功。

```
kubectrl exec -it pod_name -- /bin/sh
df -h
```

回显如下，可以看到overlay容量从10G扩容到15G，说明扩容成功。

```
Filesystem      Size  Used Avail Use% Mounted on
overlay         15G  104K   15G   1% /
tmpfs           64M    0   64M   0% /dev
tmpfs           3.6G    0   3.6G   0% /sys/fs/cgroup
/dev/mapper/vgpaas-share 98G  4.0G   89G   5% /etc/hosts
...
```

- Devicemapper：单独划分了thinpool存储镜像相关数据。执行以下代码，查看容器容量是否扩容成功。

```
kubectrl exec -it pod_name -- /bin/sh
df -h
```

回显如下，可以看到thinpool容量从10G扩容到15G，说明扩容成功。

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vgpaas-thinpool-snap-84 15G  232M   15G   2% /
tmpfs           64M    0   64M   0% /dev
tmpfs           3.6G    0   3.6G   0% /sys/fs/cgroup
/dev/mapper/vgpaas-kubernetes 11G  41M   11G   1% /etc/hosts
/dev/mapper/vgpaas-dockersys 20G  1.1G   18G   6% /etc/hostname
...
```

----结束

8.2 CCE 支持的存储在持久化和多节点挂载方面的有什么区别？

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。

当前云容器引擎CCE支持本地磁盘存储、云硬盘存储卷、文件存储卷、对象存储卷和极速文件存储卷。

各类存储的区别和对比如下：

表 8-1 各类存储的区别和对比

存储类型	持久化存储	伴随容器自动迁移	多节点挂载
本地磁盘存储	支持	不支持	不支持
云硬盘存储卷（EVS）	支持	支持	不支持
对象存储卷（OBS）	支持	支持	支持，可由多个节点或工作负载共享
文件存储卷（SFS）	支持	支持	支持，可由多个节点或工作负载共享

存储类型	持久化存储	伴随容器自动迁移	多节点挂载
极速文件存储卷（SFS Turbo）	支持	支持	支持，可由多个节点或工作负载共享
专属存储（DSS）	支持	支持	不支持

CCE 存储类型选择

创建工作负载时，可以使用以下类型的存储。建议将工作负载pod数据存储在云存储上。若存储在本地磁盘上，节点异常无法恢复时，本地磁盘中的数据也将无法恢复。

- 本地硬盘：将容器所在宿主机的文件目录挂载到容器的指定路径中（对应Kubernetes的HostPath），也可以不填写源路径（对应Kubernetes的EmptyDir），不填写时将分配主机的临时目录挂载到容器的挂载点，指定源路径的本地硬盘数据卷适用于将数据持久化存储到容器所在宿主机，EmptyDir（不填写源路径）适用于容器的临时存储。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。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。
- 云硬盘存储卷：CCE支持将EVS创建的云硬盘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。这种存储方式适用于需要永久化保存的数据。
- 文件存储卷：CCE支持创建SFS存储卷并挂载到容器的某一路径下，也可以使用底层SFS服务创建的文件存储卷，SFS存储卷适用于多读多写的持久化存储，适用于多种工作负载场景，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场景。
- 对象存储卷：CCE支持创建OBS对象存储卷并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。
- 极速文件存储卷：CCE支持创建SFS Turbo极速文件存储卷并挂载到容器的某一路径下，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。
- 专属存储（DSS）：CCE支持创建DSS存储卷并挂载到容器的某一路径下。DSS提供独享的物理存储资源，能够满足高性能、低延迟的存储需求，适合于高性能计算、在线分析处理以及混合负载等应用场景。

8.3 创建 CCE 节点时可以不添加数据盘吗？

- 如果“系统组件存储”选择“系统盘”，可以不添加数据盘。
- 如果“系统组件存储”选择“数据盘”，数据盘是必须要的。

新建节点会给节点绑定一个供kubelet及容器引擎使用的专用数据盘，详情请参见[数据盘空间分配说明](#)。CCE数据盘默认使用LVM（Logical Volume Manager）进行磁盘管理，开启后您可以通过空间分配调整数据盘中不同资源的空间占比，具体请参见[LVM简介](#)。

若数据盘卸载或损坏，会导致容器引擎服务异常，最终导致节点不可用。

8.4 EVS 存储卷的底层存储被删除或者过期后，数据是否可以恢复？

- 当底层云硬盘被删除时，可能存在以下场景：
 - 若您已为原PVC创建快照，则可直接通过快照恢复数据，具体请参见[使用快照创建PVC](#)。
 - 若您已开启云硬盘回收站功能，则可以通过回收站找回，具体请参见[恢复回收站中的云硬盘](#)。
 - 若不满足以上场景，则原数据无法找回。
- 当底层云硬盘的计费类型为包年/包月且过期时，可能存在以下场景：
 - 若云硬盘尚在保留期，则续费后即可继续使用。
 - 若云硬盘因欠费而被释放，且已为原PVC创建快照，则可直接通过快照恢复数据，具体请参见[使用快照创建PVC](#)。
 - 若云硬盘因欠费而被释放，且未提前创建快照，则原数据无法找回。

8.5 公网访问 CCE 部署的服务并上传 OBS，为何报错找不到 host？

线下机器访问CCE部署的服务并上传OBS，报错找不到host，报错截图如下：

Time	message
February 22nd 2020, 18:50:27.521	com.obs.services.exception.ObsException: OBS service Error Message. Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com
February 22nd 2020, 18:50:27.521	18:50:27.520 [XNIO-1 task-16] ERROR c.h.f.c.provider.ExceptionProvider - OBS service Error Message. Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com
February 22nd 2020, 18:50:27.298	18:50:27.298 [XNIO-1 task-9] ERROR c.h.f.c.provider.ExceptionProvider - OBS service Error Message. Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com
February 22nd 2020, 18:50:27.298	com.obs.services.exception.ObsException: OBS service Error Message. Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com
February 22nd 2020, 18:50:27.275	18:50:27.274 [XNIO-1 task-9] WARN c.o.s.internal.RestStorageService - com.obs.services.internal.ServiceException: Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com HEAD 'https://obs.oss-cn-sz-001.obs.amazonaws.com/obs-it-problem-management-media-test?apiversion' on Host 'obs.oss-cn-sz-001.obs.amazonaws.com'
February 22nd 2020, 18:50:27.275	com.obs.services.internal.ServiceException: Request Error : java.net.UnknownHostException: obs.oss-cn-sz-001.obs.amazonaws.com
February 22nd 2020, 18:50:27.275	2020-02-22 18:50:27 274 com.obs.services.internal.RestStorageService handleThrowable 205 com.obs.services.internal.ServiceException: Request Error : java.net.UnknownHostException:

问题定位

服务收到http请求之后，向OBS传输文件，这些报文都会经过Proxy。

传输文件总量很大的话，会消耗很多资源，目前proxy分配内存128M，在压测场景下，损耗非常大，最终导致请求失败。

目前压测所有流量都经过Proxy，业务量大就要加大分配资源。

解决方法

1. 传文件涉及大量报文复制，会占用内存，建议把Proxy内存根据实际场景调高后再进行访问和上传。
2. 可以考虑把该服务从网格内移除出去，因为这里的Proxy只是转发包，并没有做其他事情，如果是通过Ingress Gateway走进来的话，这个服务的灰度发布功能是不受影响的。

8.6 Pod 接口 ExtendPathMode: PodUID 如何与社区 client-go 兼容?

使用场景

社区Pod结构体中没有ExtendPathMode，用户使用client-go调用创建pod或deployment的API接口时，创建的pod中没有ExtendPathMode。为了与社区的client-go兼容，CCE提供了如下解决方案。

解决方案

须知

- 创建pod时，在pod的annotation中需增加**kubernetes.io/extend-path-mode**。
- 创建deployment时，需要在template中的annotation增加**kubernetes.io/extend-path-mode**。

如下为创建pod的yaml示例，在annotation中添加**kubernetes.io/extend-path-mode**关键字后，完全匹配到containername，name，mountpath三个字段，则会在volumeMount中增加对应的**extendpathmode**：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-8b59d5884-96vdz
  generateName: test-8b59d5884-
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/test-8b59d5884-96vdz
  labels:
    app: test
    pod-template-hash: 8b59d5884
  annotations:
    kubernetes.io/extend-path-mode:
'["containername":"container-0","name":"vol-156738843032165499","mountpath":"/tmp","extendpathmode":"PodUID"]'
  metrics.alpha.kubernetes.io/custom-endpoints: '[{"api":"","path":"","port":"","names":""}]'
ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: test-8b59d5884
    uid: 2633020b-cd23-11e9-8f83-fa163e592534
    controller: true
    blockOwnerDeletion: true
spec:
  volumes:
    - name: vol-156738843032165499
      hostPath:
        path: /tmp
```

```
type: "  
- name: default-token-4s959  
secret:  
  secretName: default-token-4s959  
  defaultMode: 420  
containers:  
- name: container-0  
  image: 'nginx:latest'  
  env:  
    - name: PAAS_APP_NAME  
      value: test  
    - name: PAAS_NAMESPACE  
      value: default  
    - name: PAAS_PROJECT_ID  
      value: b6315dd3d0ff4be5b31a963256794989  
resources:  
  limits:  
    cpu: 250m  
    memory: 512Mi  
  requests:  
    cpu: 250m  
    memory: 512Mi  
volumeMounts:  
- name: vol-156738843032165499  
  mountPath: /tmp  
  extendPathMode: PodUID  
- name: default-token-4s959  
  readOnly: true  
  mountPath: /var/run/secrets/kubernetes.io/serviceaccount  
terminationMessagePath: /dev/termination-log  
terminationMessagePolicy: File  
imagePullPolicy: Always  
restartPolicy: Always  
terminationGracePeriodSeconds: 30  
dnsPolicy: ClusterFirst  
serviceAccountName: default  
serviceAccount: default  
nodeName: 192.168.0.24  
securityContext: {}  
imagePullSecrets:  
  - name: default-secret  
  - name: default-secret  
affinity: {}  
schedulerName: default-scheduler  
tolerations:  
  - key: node.kubernetes.io/not-ready  
    operator: Exists  
    effect: NoExecute  
    tolerationSeconds: 300  
  - key: node.kubernetes.io/unreachable  
    operator: Exists  
    effect: NoExecute  
    tolerationSeconds: 300  
priority: 0  
dnsConfig:  
  options:  
    - name: timeout  
      value: "  
    - name: ndots  
      value: '5'  
    - name: single-request-reopen  
enableServiceLinks: true
```

表 8-2 关键参数说明

参数	参数类型	描述
containername	String	容器名称。

参数	参数类型	描述
name	String	volume的名称。
mountpath	String	挂载路径
extendpathmode	String	将在已创建的“卷目录/子目录”中增加一个三级目录，便于更方便获取单个Pod输出的文件。 支持如下五种类型，详情请参考 容器日志 。 • 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。

8.7 创建存储卷失败如何解决？

现象描述

创建PV或PVC失败，在事件中看到如下信息。

```
{"message": "Your account is suspended and resources can not be used.", "code": 403}
```

问题根因

事件信息表示账号被停用或没有权限，请检查账号状态是否正常。

如账号正常请查看该用户的命名空间权限，您需要拥有该命名空间的开发权限、运维权限或管理员权限之一，或者包含PVC/PV读写操作的自定义权限。详情请参见[配置命名空间权限（控制台）](#)。

8.8 CCE 容器云存储 PVC 能否感知底层存储故障？

CCE PVC按照社区逻辑实现，PVC本身的定义是存储声明，与底层存储解耦，不负责感知底层存储细节，因此没有感知底层存储故障的能力。

云监控服务CES 具备查看云服务监控指标的能力：云监控服务基于云服务自身的服务属性，已经内置了详细全面的监控指标。当用户在云平台上开通云服务后，系统会根据服务类型自动关联该服务的监控指标，帮助用户实时掌握云服务的各项性能指标，精确掌握云服务的运行情况。

建议有存储故障感知诉求的用户配套云监控服务CES的云服务监控能力使用，实现对底层存储的监控和告警通知。

8.9 通用文件系统（原 SFS 3.0 容量型）在 OS 中的挂载点修改属组及权限报错

现象描述

将通用文件系统（原SFS 3.0 容量型）挂载到OS中某个目录后，该目录成为通用文件系统（原SFS 3.0 容量型）的挂载点，使用chown和chmod命令尝试修改挂载点的属组或权限，会遇到以下报错：

```
chown: changing ownership of '***': Operation not permitted
```

或

```
chmod: changing permissions of '***': Operation not permitted
```

问题根因

通用文件系统（原SFS 3.0 容量型）在OS中的挂载点的属组及权限不支持修改，请勿执行此类操作。

8.10 无法使用 kubectl 命令删除 PV 或 PVC

现象描述

无法使用kubectl delete命令直接删除已有的PV或PVC，删除后会一直处于Terminating状态。

问题根因

Kubernetes为了防止误删除PV和PVC导致数据丢失，存在数据保护机制，无法使用delete命令直接删除。

解决方案

执行以下命令，先解除保护机制，再删除PV或PVC。

如果已经使用kubectl delete命令删除PV或PVC，会一直处在Terminating状态，在执行下面patch命令后会直接删除，无需重复执行kubectl delete命令。

- PV
kubectl patch pv <pv-name> -p '{"metadata":{"finalizers":null}}'
kubectl delete pv <pv-name>
- PVC
kubectl patch pvc <pvc-name> -p '{"metadata":{"finalizers":null}}'
kubectl delete pvc <pvc-name>

8.11 删除挂载了云存储的 Pod 时提示 target is busy

现象描述

在删除Pod时一直处于Terminating状态，通过节点上的/var/log/cce/kubernetes/kubelet.log路径查询kubelet日志，出现以下错误提示：

```
...umount failed: exit status 32...Output: umount: <mount-path>: target is busy
```

问题根因

出现以上问题的原因是宿主主机上有其他进程正在使用该设备。

解决方法

您需要登录到Pod所在宿主主机上查找正在使用该设备的进程，并终止对应的进程。

步骤1 登录Pod所在节点。

步骤2 执行以下命令，找到对应挂载路径下的云存储设备，其中<mount-path>为错误信息中显示的挂载路径。

```
mount | grep <mount-path>
```

回显示例如下：

```
/dev/sdatest on <mount-path> type ext4 (rw,relatime)
```

步骤3 执行以下命令，找到使用块设备的进程ID。

```
fuser -mv /dev/sdatest
```

步骤4 终止对应的进程。

```
fuser -kmv /dev/sdatest
```

进程终止后，云存储会被自动卸载，Pod会被继续删除。

----结束

8.12 无法自动创建包周期的云硬盘存储卷

问题现象

创建包周期的云硬盘存储卷时，无法为cce_cluster_agency委托添加支付权限。

说明

当集群版本为v1.23.14-r0、v1.25.9-r0、v1.27.6-r0、v1.28.4-r0及以上时支持动态创建包周期的云硬盘存储卷，且集群中需安装2.4.16及以上版本的Everest插件。

问题原因

cce_cluster_agency委托是CCE的系统委托，其中包含CCE组件需要的云服务资源操作权限，但不包含支付权限，详情请参见[系统委托说明](#)。在创建包周期的云硬盘存储卷时，要求包含支付权限，因此需要为cce_cluster_agency委托中添加bss:order:pay权限。

解决方案

您可以创建一个自定义策略，为该策略添加bss.order.pay权限，然后将该策略授权给cce_cluster_agency委托。

步骤1 创建自定义策略。

1. 登录IAM控制台，在左侧导航栏中选择“权限管理>权限”，单击右上方的“创建自定义策略”。
2. 配置自定义策略参数。详情请参见[创建自定义策略](#)。
 - 策略名称：设置为CCE Subscribe Operator。
 - 策略配置方式：选择“JSON视图”。
 - 策略内容：设置策略内容如下。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "bss:order:pay"
      ]
    }
  ]
}
```

3. 单击“确定”。

步骤2 将自定义策略授权给cce_cluster_agency委托。

1. 登录IAM控制台，在左侧导航栏中选择“委托”。
2. 找到名为cce_cluster_agency的委托，并单击“授权”。
3. 搜索CCE Subscribe Operator自定义策略并勾选，单击“下一步”。
4. 根据需求选择授权范围方案，默认选择“所有资源”。
5. 单击“确定”。

步骤3 返回CCE控制台，重新尝试创建包周期的云硬盘存储卷，该问题已解决。

----结束

8.13 误卸载存储池的磁盘后如何恢复

存储池是Everest创建的一种Custom Resource，资源为nodelocalvolumes，该资源在正常情况下不建议手动操作。Everest每分钟会扫描空闲磁盘，并检查已添加进存储池的磁盘是否正常。

Everest使用LVM进行存储池管理，本地持久卷和本地临时卷均为LVM中的一个卷组（Volume Group，VG）：

- 本地持久卷使用的卷组名称为：vg-everest-localvolume-persistent
- 本地临时卷使用的卷组名称为：vg-everest-localvolume-ephemeral

本文以本地持久卷举例，修复本地临时卷时，使用相应的卷组即可。

须知

本文提供的指导仅能恢复误卸载磁盘后存储池页面显示“不可用”状态，修复后可重新导入持久卷或临时卷，但无法恢复原有数据。

问题现象

当用户错误卸载存储池的磁盘时，会导致该节点存储池状态不可用。

节点名称	状态	已挂载/可挂载	写入模式	持久卷容量	临时卷容量	创建时间	操作
192.168.1.137	不可用	2 / 3	持久卷：线性 临时卷：--	89.9922 GiB 可用，共 99.9922 GiB	未导入	5天前	导入持久卷 导入临时卷

问题定位

使用kubect命令查看nodelocalvolumes资源：

```
kubectl get nodelocalvolumes.localvolume.everest.io -n kube-system 192.168.1.137 -o yaml
```

看到状态已列出错误原因为/dev/vde is lost：

```
...
status:
  lastUpdateTime: "2024-07-16T07:13:55Z"
  message: the device 6eef2886-f5ad-4b3f-a:/dev/vde is lost
  phase: Unavailable
totalDisks:
- capacity: 100Gi
  name: /dev/vdb
  uuid: 3511993c-61e6-4faa-a
usedDisks:
- totalSize: 102392Mi
  type: persistent
  usedSize: 10Gi
  volume-group: vg-everest-localvolume-persistent
volumes:
- capacity: 50Gi
  name: /dev/vdd
  pv-uuid: Jo9Uur-evVi-RLWM-yaln-J6rz-3QCo-aHpvwB
  uuid: 40b8b92b-5852-4a97-9
- capacity: 50Gi
  name: /dev/vde
  pv-uuid: ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo
  uuid: 6eef2886-f5ad-4b3f-a
```

在节点上查看LVM资源均会产生告警。

- 运行vgdisplay命令告警如下：

```
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
WARNING: VG vg-everest-localvolume-persistent is missing PV ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo (last written to /dev/vde).
--- Volume group ---
VG Name          vg-everest-localvolume-persistent
System ID
Format           lvm2
Metadata Areas   1
Metadata Sequence No 3
VG Access        read/write
VG Status        resizable
MAX LV           0
Cur LV          1
Open LV          1
Max PV           0
```



```
Cur PV      2
Act PV      1
VG Size     99.99 GiB
PE Size     4.00 MiB
Total PE    25598
Alloc PE / Size 2560 / 10.00 GiB
Free PE / Size 23038 / 89.99 GiB
VG UUID     31LHdA-yZPV-M7JX-ttwK-aynz-lyxY-usp22p
```

● 运行**lvdisplay**命令告警如下：

```
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
WARNING: VG vg-everest-localvolume-persistent is missing PV ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo (last written to /dev/vde).
--- Logical volume ---
LV Path                /dev/vg-everest-localvolume-persistent/pvc-fad963ae-f168-4e6c-a060-088b68ee503e
LV Name                pvc-fad963ae-f168-4e6c-a060-088b68ee503e
VG Name                vg-everest-localvolume-persistent
LV UUID                NzZvao-peoH-q4Zr-YTs3-WgqX-vxt5-W20Og2
LV Write Access        read/write
LV Creation host, time , 2024-07-16 12:03:20 +0800
LV Status              available
# open                 1
LV Size                10.00 GiB
Current LE             2560
Segments               1
Allocation              inherit
Read ahead sectors     auto
- currently set to     256
Block device           252:1
```

● 运行**pvdisplay**命令告警如下：

```
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
WARNING: VG vg-everest-localvolume-persistent is missing PV ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo (last written to /dev/vde).
--- Physical volume ---
PV Name                /dev/vdd
VG Name                vg-everest-localvolume-persistent
PV Size                50.00 GiB / not usable 4.00 MiB
Allocatable            yes
PE Size                4.00 MiB
Total PE               12799
Free PE                10239
Allocated PE           2560
PV UUID                Jo9Uur-evVi-RLWM-yaln-J6rz-3QCo-aHpwwB

--- Physical volume ---
PV Name                [unknown]
VG Name                vg-everest-localvolume-persistent
PV Size                50.00 GiB / not usable 4.00 MiB
Allocatable            yes
PE Size                4.00 MiB
Total PE               12799
Free PE                12799
Allocated PE           0
PV UUID                ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo
```

● 运行**pvs**命令告警如下：

```
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
WARNING: VG vg-everest-localvolume-persistent is missing PV ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo (last written to /dev/vde).
PV      VG      Fmt Attr PSize  PFree
/dev/vdc vgpaas    lvm2 a-- <50.00g  0
/dev/vdd vg-everest-localvolume-persistent lvm2 a-- <50.00g <40.00g
[unknown] vg-everest-localvolume-persistent lvm2 a-m <50.00g <50.00g
```

解决方案

步骤1 恢复nodelocalvolumes资源。

```
kubectl edit nodelocalvolumes.localvolume.everest.io -n kube-system 192.168.1.137
```

修改上述资源，删除 `spec.volumes.type: persistent` 中已丢失的磁盘，同时删除资源中整个 `status` 字段。

步骤2 移除VG中相应PV。

说明

VG信息会存在于相应的磁盘上：

- 当多块磁盘组成的VG缺少一部分时，会显示PV丢失。
- 当VG中仅有的磁盘被拔出时，`vgdisplay`将无法显示相应VG，通过`vgdisplay`无法查看到`vg-everest-localvolume-persistent`或`vg-everest-localvolume-ephemeral`时当前步骤可以跳过。

通过以下命令移除本地持久卷VG中所有已丢失的PV。其中，`vg-everest-localvolume-persistent`为本地持久卷的VG名称，若修复本地临时卷则VG名称为`vg-everest-localvolume-ephemeral`。

```
vgreduce --removemissing vg-everest-localvolume-persistent
```

回显如下：

```
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
WARNING: VG vg-everest-localvolume-persistent is missing PV ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo (last written to /dev/vde).
WARNING: Couldn't find device with uuid ZxA9kY-5C28-96Z9-ZjOE-dCrc-yTgp-DOhUHo.
Wrote out consistent volume group vg-everest-localvolume-persistent.
```

再次使用 `vgdisplay` 可以看到显示正常。

```
vgdisplay vg-everest-localvolume-persistent
```

回显如下：

```
--- Volume group ---
VG Name                vg-everest-localvolume-persistent
System ID
Format                 lvm2
Metadata Areas         1
Metadata Sequence No   4
VG Access               read/write
VG Status               resizable
MAX LV                 0
Cur LV                 1
Open LV                 1
Max PV                  0
Cur PV                 1
Act PV                  1
VG Size                 <50.00 GiB
PE Size                 4.00 MiB
Total PE                12799
Alloc PE / Size         2560 / 10.00 GiB
Free PE / Size          10239 / <40.00 GiB
VG UUID                 31LHdA-yZPV-M7JX-ttwK-aynz-lyxY-usp22p
```

步骤3 重启相应节点的everest-csi-driver。

1. 查看everest-csi-driver的Pod名称。

```
kubectl get pod -A -owide
```

回显如下：

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
IP	NODE	NOMINATED NODE	READINESS GATES		
kube-system	everest-csi-driver-7clbg	1/1	Running	0	5d4h
192.168.1.38	192.168.1.38	<none>	<none>		
kube-system	everest-csi-driver-jvj9f	1/1	Running	0	5d4h
192.168.1.137	192.168.1.137	<none>	<none>		

- 删除192.168.1.137节点上的Pod。

```
kubectl delete pod -n kube-system everest-csi-driver-jvj9f
```
- 此时查看 `nodeLocalvolumes` 状态已正常，可继续导入持久卷。

导出							
Q 选择属性筛选，或输入关键字搜索							
节点名称	状态	已挂载/可挂载	写入模式	持久卷容量	临时卷容量	创建时间	操作
192.168.1.137	正常	1/2	持久卷: 线性 临时卷: --	38 9961 GB 可用, 共 48 9961 GB	未导入	5天前	导入持久卷 导入临时卷

步骤4 重启节点。

某些情况下如NPD可能依然会认为节点存在指标异常导致节点不可用，因此通过重启节点解决此类问题。

----结束

8.14 删除动态创建的 PVC 之后，底层存储依旧残留

问题现象

删除集群中动态创建的PVC，PVC使用的StorageClass中`reclaimPolicy`为Delete模式，但删除PVC时底层存储却没有被同步删除。

触发场景

- 同时删除PVC和与其绑定的PV，会出现底层存储没有被同步删除的情况。
- 在删除PVC前，尝试直接删除PV，但由于PV被PVC绑定而受到保护无法直接删除。然后再删除PVC，就会出现底层存储没有被同步删除的情况。

问题根因

在开源csi-provisioner模块业务逻辑中，常规情况下删除动态创建的PVC，会先删除PVC，待PVC资源删除成功后，将PV状态更新为Released。csi-provisioner会监听到PV更新事件，开始执行删底层卷的流程，等底层卷删除成功后，再下发删除PV的请求，这样即完成了“PVC-底层卷-PV”的完整删除链。

异常操作过程中，未删除PVC的情况下直接删除PV，但由于PV上有`kubernetes.io/pv-protection`这个finalizer，无法立即删除，但会给PV加上`deletionTimestamp`。随后删除PVC，PVC资源删除成功后，PV状态更新为Released，csi-provisioner监听到PV更新事件，但由于PV上有`deletionTimestamp`，csi-provisioner认为不需要删除底层卷，于是跳过删底层卷的流程，直接开始删除PV，这样PVC和PV被成功删除，但是底层卷残留。关于此问题的逻辑代码请参见[controller](#)。

解决方案

- 对于已残留的底层存储，请通过手动删除的方式进行清理。
- 对于未删除的动态创建PVC，请直接删除PVC，其绑定的PV和底层存储会被自动删除，无需手动删除。

8.15 已释放的云硬盘重新绑定 PV 后，在 Pod 中挂载失败

问题现象

在Pod中挂载云硬盘存储卷时，报错信息中出现“wrong fs type, bad option, bad superblock on /dev/sda, missing codepage or helper program, or other error”，具体示例如下：

图 8-1 报错信息示例

```
(combined from similar events): MountVolume.MountDevice failed for volume "pvc-8d0644c5-980e-428c-8bdb-2f095de2e1d8" : rpc error: code = Internal desc = [b7d0a0dc-754b-4971-bc2e-e5948721f706] failed to format "/dev/sda" to mount it to  
"/mnt/paas/kubernetes/kubelet/plugins/kubernetes.io/csi/disk.csi.everest.io/c7889f5ff1d1b96a631128505b07a324a19dc2a2c314c48b733052/globalmount": mount failed: exit status 32 Mounting command: mount Mounting arguments: ext4 -o defaults /dev/sde  
/mnt/paas/kubernetes/kubelet/plugins/kubernetes.io/csi/disk.csi.everest.io/c7889f5ff1d1b96a631128505b07a324a19dc2a2c314c48b733052/globalmount Output: mount:  
/mnt/paas/kubernetes/kubelet/plugins/kubernetes.io/csi/disk.csi.everest.io/c7889f5ff1d1b96a631128505b07a324a19dc2a2c314c48b733052/globalmount: wrong fs type, bad option, bad superblock on /dev/sde, missing codepage or other error.
```

问题根因

云硬盘的文件系统类型与首次绑定的PV文件系统类型（fstype）一致。当已释放的云硬盘重新绑定PV时，要求现有PV的fstype与原PV的fstype一致。当fstype不一致时，系统将会检测出现有PV的fstype与磁盘实际不符，进而导致挂载失败。

解决方案

建议重新创建云硬盘用于后续使用。

9 命名空间

9.1 一个集群最多能创建多少个命名空间？

在Kubernetes集群中并没有明确的硬性限制来规定一个集群中可以创建的命名空间数量，但在实际使用过程中，可创建命名空间的数量会受到一些其他因素的限制。

- 资源限制

对于CCE集群来说，集群的规模越大，集群控制平面节点的资源也会相对变大，能够支持的命名空间数量相对也会更多。但每个命名空间都会消耗一定的控制平面资源用于存储元数据，因此大量的命名空间可能会导致集群控制平面的资源紧张，影响集群稳定性。

- 性能限制

对于集群控制平面来说，大量的命名空间会增加API服务器的负载，降低其响应速度。例如，在查询所有命名空间的资源时需要遍历更多的数据，导致集群性能下降。

因此，在实际使用过程中，建议根据需求合理创建命名空间，您也可以为每个命名空间设置合理的资源配额和限制，以确保集群的稳定性和性能。

9.2 命名空间因 APIService 对象访问失败无法删除

问题现象

删除命名空间时，命名空间一直处“删除中”状态，无法删除。查看命名空间yaml配置，status中有报错“DiscoveryFailed”，示例如下：

```
75 - kubernetes
76 status:
77   phase: Terminating
78   conditions:
79     - type: NamespaceDeletionDiscoveryFailure
80       status: 'True'
81       lastTransitionTime: '2022-07-04T13:44:55Z'
82       reason: DiscoveryFailed
83       message: 'Discovery failed for some groups, 1 failing: unable to retrieve the complete list of server
84 APIs: metrics.k8s.io/v1beta1: the server is currently unable to handle the request'
85     - type: NamespaceDeletionGroupVersionParsingFailure
86       status: 'False'
```

上图中报错信息为：Discovery failed for some groups, 1 failing: unable to retrieve the complete list of server APIs: metrics.k8s.io/v1beta1: the server is currently unable to handle the request

表示当前删除命名空间动作阻塞在kube-apiserver访问metrics.k8s.io/v1beta1接口的APIService资源对象。

问题根因

当集群中存在APIService对象时，删除命名空间会先访问APIService对象，若APIService资源无法正常访问，会阻塞命名空间删除。除用户创建的APIService对象资源外，CCE集群部分插件也会自动创建APIService资源，如metrics-server、prometheus插件。

说明

APIService使用介绍请参考：<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/>

解决方法

可以采用如下两种方法解决：

- 修复报错信息中的APIService对象，使其能够正常访问，如果是插件中的APIService，请确保插件的Pod正常运行。
- 删除报错信息中的APIService对象，如果是插件中的APIService，可从页面卸载该插件。

9.3 如何删除 Terminating 状态的命名空间？

Kubernetes中namespace有两种常见的状态，即Active和Terminating状态。当对应的命名空间下还存在运行的资源，但该命名空间被删除时才会出现Terminating状态，这种情况下只要等待Kubernetes本身将命名空间下的资源回收后，该命名空间将会被系统自动删除。

但是在某些情况下，即使命名空间下没有运行的资源，但依然无法删除Terminating状态的命名空间的情况，它会一直处于Terminating状态下。

解决这个问题的步骤为：

步骤1 查看命名空间详情。

```
$ kubectl get ns | grep rdb
rdbms          Terminating  6d21h

$ kubectl get ns rdbms -o yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Namespace","metadata":{"annotations":{},"name":"rdbms"}}
  creationTimestamp: "2020-05-07T15:19:43Z"
  deletionTimestamp: "2020-05-07T15:33:23Z"
  name: rdbms
  resourceVersion: "84553454"
  selfLink: /api/v1/namespaces/rdbms
  uid: 457788ddf-53d7-4hde-afa3-1fertg21ewe1
spec:
```

```
finalizers:
- kubernetes
status:
phase: Terminating
```

步骤2 查看该命名空间下的资源。

```
# 查看集群中可以使用命名空间隔离的资源
$ kubectl api-resources -o name --verbs=list --namespaced | xargs -n 1 kubectl get --show-kind --ignore-not-found -n rdbms
```

发现rdbms命名空间下并无资源占用。

步骤3 尝试对命名空间进行删除。

直接删除命名空间rdbms。

```
$ kubectl delete ns rdbms
Error from server (Conflict): Operation cannot be fulfilled on namespaces "rdbms": The system is ensuring all content is removed from this namespace. Upon completion, this namespace will automatically be purged by the system.
```

删除操作未能完成，并提示系统会在确定命名空间中无资源后自动删除该命名空间。

步骤4 使用强制删除。

```
$ kubectl delete ns rdbms --force --grace-period=0
warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. The resource may continue to run on the cluster indefinitely.
Error from server (Conflict): Operation cannot be fulfilled on namespaces "rdbms": The system is ensuring all content is removed from this namespace. Upon completion, this namespace will automatically be purged by the system.
```

依然无法删除该命名空间。

步骤5 大多数情况下，命名空间下的资源无法强制删除，您可以使用原生接口进行删除。

获取namespace的详情信息。

```
$ kubectl get ns rdbms -o json > rdbms.json
```

查看namespace定义的json配置，编辑json文件并删除掉spec部分。

```
$ cat rdbms.json
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "annotations": {
      "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Namespace\",\"metadata\":{\"annotations\":{},\"name\":\"rdbms\"}}\n"
    },
    "creationTimestamp": "2019-10-14T12:17:44Z",
    "deletionTimestamp": "2019-10-14T12:30:27Z",
    "name": "rdbms",
    "resourceVersion": "8844754",
    "selfLink": "/api/v1/namespaces/rdbms",
    "uid": "29067ddf-56d7-4cce-afa3-1fbdbb221ab1"
  },
  "spec": {
    "finalizers": [
      "kubernetes"
    ]
  },
  "status": {
    "phase": "Terminating"
  }
}
```

执行接口PUT请求更新后，命名空间将自动删除。

```
$ curl --cacert /root/ca.crt --cert /root/client.crt --key /root/client.key -k -H "Content-Type:application/json" -X PUT --data-binary @rdbms.json https://x.x.x.x:5443/api/v1/namespaces/rdbms/finalize
```

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "rdbms",
    "selfLink": "/api/v1/namespaces/rdbms/finalize",
    "uid": "29067ddf-56d7-4cce-afa3-1fbdbb221ab1",
    "resourceVersion": "8844754",
    "creationTimestamp": "2019-10-14T12:17:44Z",
    "deletionTimestamp": "2019-10-14T12:30:27Z",
    "annotations": {
      "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Namespace\", \"metadata\":{\"annotations\":{},\"name\":\"rdbms\"}}\\n"
    }
  },
  "spec": {
  },
  "status": {
    "phase": "Terminating"
  }
}
```

如果仍然无法删除命名空间，请查看metadata部分是否存在finalizers字段，如果存在，需要通过如下命令进入命名空间后删除该字段：

```
kubectl edit ns rdbms
```

说明

- 集群证书获取方法请参见[获取集群证书](#)。
- <https://x.x.x.x:5443>为连接集群的地址。您可以登录[CCE控制台](#)，进入集群，查看连接信息的内网地址进行获取。

步骤6 再次查看namespace确认已经被删除。

```
$ kubectl get ns | grep rdb
```

----结束

10 模板插件

10.1 插件异常问题排查

在插件安装、升级、更改配置等过程中出现异常问题时，控制台通常会提示相应的错误码。您可以根据错误码查找对应的问题，查看问题原因和解决方案。本文介绍常见错误码及其问题原因和解决方案。

资源冲突

问题现象

安装插件时，出现“内部错误”，错误码为CCE.03500001。

问题原因

出现“内部错误”时，错误信息内会提示具体的错误原因，如 ClusterRole `\gatekeeper-manager-role\` in namespace `\` exists and cannot be imported into the current release 表示集群内已经创建了对应的ClusterRole资源，但是该资源不是由插件管理的。

解决方案

手动使用kubectl删除非插件管理的冲突的资源后，重试插件的安装。

安装超时

问题现象

安装/升级插件时，提示安装失败，错误信息显示 timed out。

```
Release "*****" failed: failed pre-install: timed out waiting for the condition
```

问题原因

插件的Pod没有到达就绪状态，导致插件无法完成安装/升级。

解决方案

在“概览”页面查看“K8S事件”，确认Pod没有就绪的原因。

常见原因	解决方案
原因一：Pod无法调度	对应事件内容：FailedScheduling 原因详述：集群中的节点无法满足Pod的调度要求，可能由以下一种或多种原因导致。可以通过事件的详细描述确定具体原因。 • 集群节点可用的CPU、内存资源不足，无法满足插件Pod的资源需求，对应事件描述中存在Insufficient memory或Insufficient cpu等字样。 • 节点存在插件Pod未容忍的污点，对应事件描述中存在the pod didn't tolerate等字样。 • 节点数量过少，无法满足插件Pod的反亲和性要求，对应事件描述中存在didn't match pod anti-affinity rules等字样。 解决方案：可以参考以下方法满足插件Pod的调度要求，处理完成之后再次尝试插件安装操作。 • 检查节点的污点，删除不必要的污点。具体操作，请参见管理节点污点。 • 合理分配容器资源。具体操作，请参见合理分配容器计算资源。 • 在集群中添加新的节点。具体操作，请参见创建节点池。
原因二：Pod创建失败	参考 工作负载状态异常定位方法 解决创建失败的问题。

插件资源不存在

问题现象

更新或升级插件时，提示更新失败，报 404 错误。

```
update release failed: update release failed {"error":{"message":"Get release failed: get v3 release by cluster failed, error: release: not found","code":"SVCSTG.CCECAM.4040204"}}, 404
```

问题原因

插件相关资源查询异常，无法直接更新或升级，可能由于相关资源被手工修改或删除导致。

解决方案

请先卸载该插件，然后重新安装最新版本。

10.2 集群安装 nginx-ingress 插件失败，一直处于创建中？

问题背景

客户已经购买并搭建了CCE集群，希望在公网上可以访问到CCE上部署的应用服务，目前最高效的方式是在ingress资源上注册该应用的Service路径，从而满足要求。

但客户安装ingress插件后，插件状态一直显示“创建中”，nginx-ingress-controller的pod一直处于pending状态。

解决方案

nginx限制的内存资源不足导致无法启动，取消限制后正常。

场景模拟

步骤1 新集群3个节点，规格 6cpu，12G内存，每个节点2U4G。

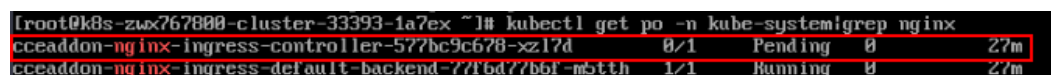
步骤2 单击nginx-ingress插件安装，选择规格2核2G。

步骤3 nginx-ingress deployment安装成功，但是nginx-ingress-controller安装失败。

图 10-1 一直处于创建中



图 10-2 安装失败



步骤4 错误显示资源不足。

```
status:
  phase: Pending
  conditions:
    - type: PodScheduled
      status: 'False'
      lastProbeTime: '2020-02-13T01:20:57Z'
      lastTransitionTime: '2020-02-12T08:50:21Z'
      reason: Unschedulable
      message: '0/3 nodes are available: 3 Insufficient cpu, 3 Insufficient memory.'
  qosClass: Guaranteed
```

步骤5 添加节点资源为4U8G后，nginx-ingress安装正常。

----结束

问题原因

最初建立的集群中各节点的基本配置为2U4G，且各节点上有kubelet，kube-proxy及docker等相关程序占用系统资源，导致节点可用资源低于2000m，无法满足nginx-ingress插件的要求，从而无法安装。

建议方案

请重新购买节点，节点要求（>=4U8G）。

10.3 NPD 插件版本过低导致进程资源残留问题

问题描述

在节点负载压力比较大的场景下，可能存在NPD进程资源残留的问题。

问题现象

登录到CCE集群的ECS节点，查询存在大量npd进程。

```
pass 32769 16574 0 Oct20 ? 00:00:00 [tail] <defunct>
root 32768 16574 0 Oct20 ? 00:00:00 [sudo] <defunct>
root 32768 16574 0 Oct20 ? 00:00:00 [sudo] <defunct>
pass 32768 16574 0 Oct20 ? 00:00:00 [tail] <defunct>
pass 32769 16574 0 Oct13 ? 00:00:00 [grep] <defunct>
[root@mcn-imagekit-oneal-inference-t4-gpu-2-89-44 ~]# nc
bash: fork: retry: No child processes
bash: fork: retry: No child processes
[root@mcn-imagekit-oneal-inference-t4-gpu-2-89-44 ~]# ls -l -p 16574
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node-prob 16574 pass ctd DIR 252,9 4896 2051 /
node-prob 16574 pass rtd DIR 252,9 4896 2051 /
node-prob 16574 pass txt REG 252,9 56274852 25180170 /var/pass/node-problem-detector/node-problem-detector
node-prob 16574 pass mem REG 0,20 108063296 159589846 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 134217728 1590491938 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1359727646 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1343967823 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1327784289 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1311338304 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1261544036 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1244428975 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1227480734 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 109051904 1211843384 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1195578084 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 109051904 1180118994 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1165479957 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 109051904 1147393413 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1130671264 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 134217728 1580819808 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 109051904 1134680888 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 134217728 1567372072 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 117440512 1465667748 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
node-prob 16574 pass mem REG 0,20 109051904 1513039458 /run/log/Journal/6d14d648c8d4fabab6db1b1412959c/system073dfe9d514304ac3f0508736d07-00000000000a78-000b34934459033-Journal
```

解决方案

升级CCE节点故障检测（简称NPD）插件至最新版本。

步骤1 登录CCE控制台，进入集群，在左侧导航栏中选择“插件中心”，单击CCE节点故障检测下的“升级”。

说明

如果CCE节点故障检测插件版本已经为1.13.6及以上版本，则不需要进行升级操作。

步骤2 在升级插件页面配置插件参数，然后单击“确定”即可升级CCE节点故障检测插件。

----结束

10.4 模板格式不正确，无法删除模板实例？

问题现象

若上传的模板中包含不正确或者不兼容的资源，会导致安装模板失败，类似下图：



此时模板实例无法正常工作。如果您尝试在界面上删除，可能会出现deletion failed的报错，模板实例仍在列表中：

我的模板		模板实例					
批量编辑		集群:		全部命名空间	全部模板	按名称搜索	
☐	模板实例名称	☐	执行状态	☐	最新事件	☐	命名空间
☐	模板来源	☐	模板属性	☐	更新时间		
☐	aosredis	☒	升级失败	deletion failed with 1 error(s): unable to decode "": Object 'Kind' is missing ...		default	aosredis
						我的模板	2021/01/15 16:22:39 GMT+

解决方法

您可以使用kubect命令删除残留的模板实例。

说明

删除残留的模板实例无法从根本上解决该问题。为避免该问题再次发生，建议您及时更新模板中资源的apiVersion版本，保证资源apiVersion与Kubernetes版本匹配。

安装模板时，模板中的一些资源可能已经创建成功，因此首先要手动删除这些资源。确保残留的资源删除后，需删除模板实例。

若为 helm v2 的实例，在kube-system命名空间下查询模板实例对应的配置项（ConfigMap），例如：

```
[paas@192-168-0-40 ~]$ kubectl -n kube-system get cm
NAME                                DATA  AGE
9a37566a.cce.io                    0      25d
aosredis.v1                         1      55s
cceaddon-coredns.v1                 1      25d
cceaddon-everest.v1                 1      17d
cceaddon-metrics-server.v1          1      25d
cceaddon-npd-custom-config           0      25d
cceaddon-npd.v1                     1      25d
cceaddon-prometheus.v1              1      25h
cluster-autoscaler-status            1      8d
cluster-versions                    1      25d
coredns                             1      25d
extension-apiserver-authentication  6      25d
ingress-controller-leader-nginx      0      22d
[paas@192-168-0-40 ~]$
```

删除该配置项，此时模板实例即删除成功：

```
[paas@192-168-0-40 ~]$ kubectl -n kube-system delete cm aosredis.v1
configmap "aosredis.v1" deleted
[paas@192-168-0-40 ~]$
```

若为helm v3 的实例，在实例所在命名空间下查询模板实例对应的密钥（Secret），例如：

```
[root@cce-1717-vpc-node2 ~]# kubectl -n default get secret
NAME                                TYPE                                DATA  AGE
default-secret                     kubernetes.io/dockerconfigjson     1      21h
default-token-978pv                 kubernetes.io/service-account-token 3      21h
paas.elb                             cfe/secure-opaque                  3      21h
sh.helm.release.v1.test-nginx.v1    helm.sh/release.v1                  1      139m
[root@cce-1717-vpc-node2 ~]#
```

删除该密钥，此时模板实例即删除成功：

```
[root@cce-1717-vpc-node2 ~]# kubectl -n default delete secret sh.helm.release.v1.test-nginx.v1
secret "sh.helm.release.v1.test-nginx.v1" deleted
[root@cce-1717-vpc-node2 ~]#
```

注：若用户通过前端console操作，在获取实例、更新实例等操作中CCE会自动尝试转换原v2模板实例到v3模板实例。在密钥中存储release信息，原配置项中release信息不会删除。建议用户在配置项和密钥中均查询并删除该实例。

10.5 CCE 是否支持 nginx-ingress?

nginx-ingress 简介

nginx-ingress是比较热门的ingress-controller，作为反向代理将外部流量导入到集群内部，将Kubernetes内部的Service暴露给外部，在Ingress对象中通过域名匹配Service，这样就可以直接通过域名访问到集群内部的服务。

nginx-ingress由ingress-controller和nginx组件组成：

- ingress-controller：负责监听Kubernetes的Ingress对象，更新nginx配置。

📖 说明

Ingress的具体说明，请参见<https://kubernetes.io/docs/concepts/services-networking/ingress/>。

- nginx：负责请求负载均衡，支持7层转发能力。

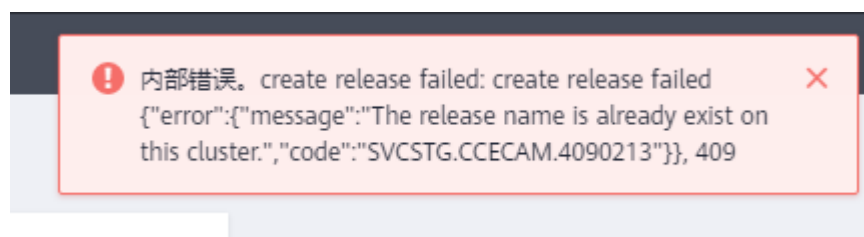
安装方法

目前CCE在插件市场中已提供nginx-ingress插件的一键式安装，也可以在安装界面展开参数进行设置。

10.6 插件安装失败，提示 The release name is already exist 如何解决?

问题现象

当安装插件失败，返回 **The release name is already exist** 错误。



问题原因

当安装插件返回**The release name is already exist**错误时，表示kubernetes集群中有残留该插件release记录，一般由于集群etcd做过备份恢复或者该插件之前安装删除异常导致。

解决方案

通过kubectl对接集群，手动清理该插件release对应的secret及configmap。以下以清理autoscaler插件release为示例。

步骤1 配置kubectl对接集群后，执行以下命令查看插件相关的release的secret列表。

kubectl get secret -A |grep cceaddon

```
[root@cce-123-vpc-node2 ~]# kubectl get secret -nkube-system |grep cceaddon
sh.helm.release.v1.cceaddon-autoscaler.v1      helm.sh/release.v1      1      61s
sh.helm.release.v1.cceaddon-autoscaler.v2      helm.sh/release.v1      1      47s
sh.helm.release.v1.cceaddon-coredns.v1         helm.sh/release.v1      1      6h2m
sh.helm.release.v1.cceaddon-everest.v1         helm.sh/release.v1      1      6h2m
[root@cce-123-vpc-node2 ~]#
```

插件release的secret名称为"sh.helm.release.v1.cceaddon-{插件名称}.v*"格式，可能有多个版本，删除时多个版本同时删除。

步骤2 执行删除release secret命令。

如删除上图中的autoscaler插件对应的release secret

```
kubectl delete secret sh.helm.release.v1.cceaddon-autoscaler.v1
sh.helm.release.v1.cceaddon-autoscaler.v2 -nkube-system
```

```
[root@cce-123-vpc-node2 ~]# kubectl delete secret sh.helm.release.v1.cceaddon-autoscaler.v1 sh.helm.release.v1.cceaddon-autoscaler.v2 -nkube-system
secret "sh.helm.release.v1.cceaddon-autoscaler.v1" deleted
secret "sh.helm.release.v1.cceaddon-autoscaler.v2" deleted
[root@cce-123-vpc-node2 ~]#
```

步骤3 若该插件为helm v2时创建，cce会在查看插件列表及插件详情等操作中自动将configmap中的v2 release转换至secret中的v3 release，原configmap中的v2 release不会删除。可执行以下命令查看插件相关的release的configmap列表。

kubectl get configmap -A | grep cceaddon

```
cluster-autoscaler-th-config      1      7d10h
[paas@192-168-0-64 ~]$ kubectl get configmap -nkube-system | grep cceaddon
cceaddon-autoscaler.v1            1      7d10h
cceaddon-autoscaler.v2            1      52m
cceaddon-coredns.v1               1      14d
cceaddon-everest.v1               1      14d
[paas@192-168-0-64 ~]$
```

插件release的configmap名称为"cceaddon-{插件名称}.v*"格式，可能有多个版本，删除时多个版本同时删除。

步骤4 执行删除release configmap命令。

如删除上图中的autoscaler插件对应的release configmap

```
kubectl delete configmap cceaddon-autoscaler.v1 cceaddon-autoscaler.v2 -
nkube-system
```

```
[paas@192-168-0-64 ~]$ kubectl delete configmap cceaddon-autoscaler.v1 cceaddon-autoscaler.v2 -nkube-system
configmap "cceaddon-autoscaler.v1" deleted
configmap "cceaddon-autoscaler.v2" deleted
[paas@192-168-0-64 ~]$
```

⚠ 注意

删除kube-system下资源属高风险操作，请确保命令正确后再执行，以免出现误删资源。

步骤5 在CCE控制台安装插件，然后再卸载保证之前的残留的插件资源清理干净，卸载完成后再进行第二次安装插件，安装成功即可。

📖 说明

第一次安装插件时可能因之前的插件残留资源而导致安装后插件状态异常，属正常现象，这时在控制台卸载插件能保证这些残留资源清理干净，再次安装插件能正常运行。

----结束

10.7 创建或升级模板失败，提示 rendered manifests contain a resource that already exists

问题现象

创建或升级模板失败，提示 “Create release by helm failed: rendered manifests contain a resource that already exists. Unable to continue with install: ...” 的错误。

由于CCE插件通过模板进行安装，因此安装或升级插件时，同样可能出现上述相同错误。

问题原因

该报错内容说明集群中已存在与模板/插件相关的残留资源，通常是由于以下原因导致：

- 模板资源（如 release）从后台被误删或异常删除。
- 模板安装的命名空间被直接删除，导致资源残留。
- 集群中存在与模板组件同名的资源且无helm v3的归属标记 “app.kubernetes.io/managed-by: Helm”。

解决方案

通过kubectl对接集群，手动删除相关报错资源，并重新安装模板或插件。

步骤1 查看报错内容，确认产生冲突的资源。请您关注 “Unable to continue with install:” 后的信息。

例如：

```
create release failed: create release failed {"error":{"message":"Create release by helm failed:rendered manifests contain a resource that already exists. Unable to continue with install: ClusterRole \"cceaddon-ingress\" in namespace \"\" exists and cannot be imported into the current release: invalid ownership metadata; annotation validation error: key \"meta.helm.sh/release-namespace\" must equal \"kube-system\": current value is \"example\"","code":"SVCSTG.CCECAM.5000208"}}
```

- **冲突资源**：ClusterRole，名称为 cceaddon-ingress，命名空间为空（ClusterRole 是非命名空间级别资源）。
- **冲突字段**：metadata.annotations.meta.helm.sh/release-namespace。
 - **预期值**：kube-system
 - **实际值**：example

步骤2 执行以下kubectl命令删除集群中的冲突资源。此处仅为示例，请根据实际报错信息中的资源进行删除。

```
kubectl delete clusterRole cceaddon-ingress
```


步骤3 解决资源冲突后，尝试重新安装模板或插件。若还有冲突提示，请重复以上步骤。

----结束

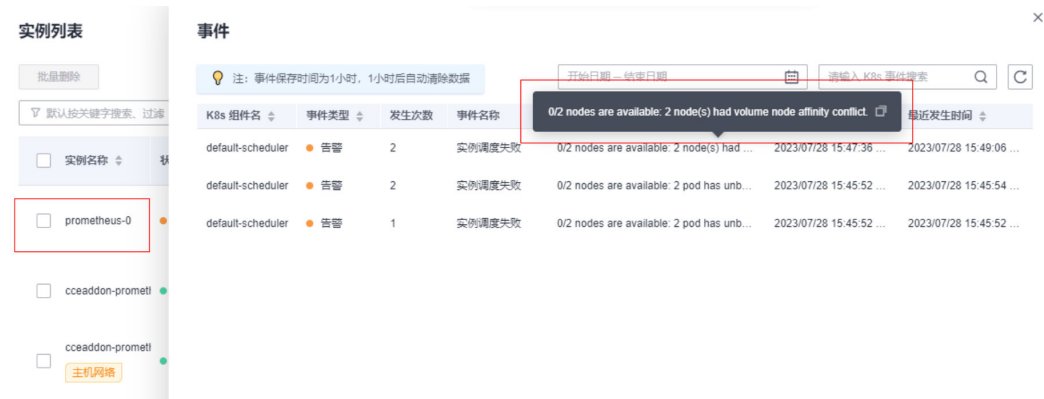
10.8 kube-prometheus-stack 插件实例调度失败如何解决？

问题现象

安装kube-prometheus-stack插件时，插件状态一直处于“部分就绪”，查看插件的prometheus实例事件中提示“0/x nodes are available: x node(s) had volume node affinity conflict.”。

如果选择安装grafana组件，也可能出现同样的问题。

图 10-3 插件实例调度失败



问题原因

当出现以上报错内容，说明集群中已存在prometheus实例需要的存储卷，但该存储卷对应的云硬盘未满足与节点同一可用区的要求，导致调度失败。这可能是由于集群并非首次安装kube-prometheus-stack插件引起。

- 首次安装kube-prometheus-stack插件时，prometheus实例会延迟绑定云硬盘类型的存储卷（PVC名为pvc-prometheus-server-0），创建该云硬盘时可用区会自动与实例运行的节点所在可用区保持一致。例如实例运行的节点可用区为可用区1，则该云硬盘的可用区也为可用区1。
- 当集群中卸载kube-prometheus-stack插件时，prometheus实例绑定的存储卷不会被一起删除，保留了已有的监控数据。当再次安装插件时，集群的节点可能已经进行过删除新建，如果集群中不存在可用区1的节点，则会出现该问题导致prometheus实例无法运行。

如果grafana实例出现同样的问题，也可能是由于上述原因引起。

解决方案

查看集群中插件遗留的存储卷对应的云硬盘可用区，并在集群中添加该可用区的节点。

- 步骤1** 登录CCE控制台，单击集群名称进入集群。
- 步骤2** 在左侧列表中选择“容器存储”，切换至“存储卷”页签，单击PVC（名为pvc-prometheus-server-0）对应的卷名称跳转至云硬盘详情页面。

图 10-4 存储卷



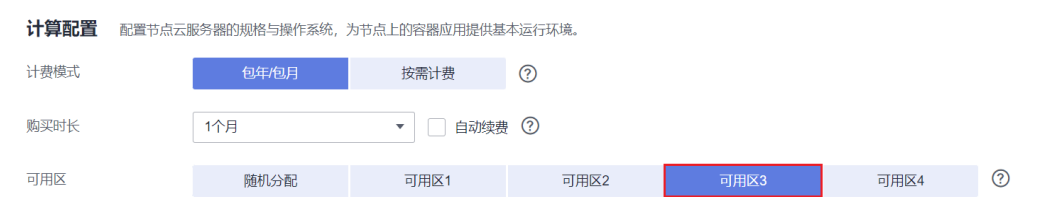
- 步骤3** 在基本信息中查看云硬盘的可用区。

图 10-5 云硬盘详情



- 步骤4** 在CCE控制台左侧列表中选择“节点管理”，单击“创建节点”，创建一个该可用区的节点。

图 10-6 创建指定可用区的节点



- 步骤5** 节点创建完成后，工作负载调度器会自动尝试重新调度。

----结束

10.9 上传模板失败如何解决？

问题现象

上传模板时出现“请求失败，请稍后重试”的错误，错误码为 **SVCSTG.CCECAM.4000121**，错误信息提示“Package name and version must be valid and same with chart name and version!”。

图 10-7 上传模板失败



问题原因

当出现以上报错内容时，说明模板Chart.yaml文件中的name和version字段和模板包名称不一致。

📖 说明

如果您需要自定义模板包的名称和版本，需要同步修改Chart.yaml文件中的name和version字段。

解决方案

步骤1 查看模板Chart.yaml文件中的name和version字段。

例如，nginx-ingress的模板包中Chart.yaml文件如下，并定义模板包名称为**newer-nginx-ingress**，版本为**4.4.2**：

```
annotations:
  artifacthub.io/changes: |
    - Adding support for disabling liveness and readiness probes to the Helm chart
    - add:(admission-webhooks) ability to set securityContext
    - Updated Helm chart to use the fullname for the electionID if not specified
    - Rename controller-webhooks-networkpolicy.yaml
  artifacthub.io/prerelease: "false"
apiVersion: v2
appVersion: 1.5.1
description: Ingress controller for Kubernetes using NGINX as a reverse proxy and
  load balancer
home: https://github.com/kubernetes/ingress-nginx
icon: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Nginx_logo.svg/500px-
  Nginx_logo.svg.png
keywords:
  - ingress
```

```
- nginx
kubeVersion: '>=1.20.0-0'
maintainers:
- name: rikatz
- name: strongjz
- name: tao12345666333
name: newer-nginx-ingress
sources:
- https://github.com/kubernetes/ingress-nginx
version: 4.4.2
```

步骤2 根据Chart.yaml修改模板包名称，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newer-nginx-ingress-4.4.2.tgz。

步骤3 重新上传该模板。

----结束

10.10 如何根据集群规格调整插件配额？

当您的集群规格调整后，可能需要根据集群规格相应地调整插件资源配额，以确保插件实例能够正常运行。例如，如果您将集群规格从50节点调整为200节点或以上，则需要增加插件CPU、内存配额，防止插件实例因需要调度过多的节点而出现OOM等异常。因此，在调整集群规格后，请您同时考虑调整插件资源配额。

CoreDNS 域名解析

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。

表 10-1 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

CCE 容器存储（Everest）

集群规格调整后，Everest插件规格需要根据集群的规模和PVC数量进行自定义调整。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10-2](#)。

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 10-2 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

CCE 集群弹性引擎

CCE集群弹性引擎插件可根据Pod资源运行的节点负载，自动调整集群中的节点数量。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。

表 10-3 CCE 集群弹性引擎插件配额建议

节点数量	实例数	CPU申请值	CPU限制值	内存申请值	内存限制值
50	2	1000m	1000m	1000Mi	1000Mi
200	2	4000m	4000m	2000Mi	2000Mi
1000	2	8000m	8000m	8000Mi	8000Mi
2000	2	8000m	8000m	8000Mi	8000Mi

Volcano 调度器

集群规格调整后，Volcano调度器所需的资源需要根据集群的规模进行自定义调整。

- 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10-4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10-4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 10-4 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
700/7w	4000	5500	7500	8500

其他插件

除上述插件外，其他插件也可能因为集群规模调整而出现分配资源不足的情况，如您发现插件实例CPU或内存使用率明显增加，甚至出现OOM或无法运行的状况，请根据情况调整资源配额。

例如云原生监控插件占用的资源与集群中的容器数量相关，当集群规模调整后，容器数量可能同步增加，需要适当调大插件实例的资源配额。

10.11 NGINX Ingress 控制器插件处于 Unknown 状态时卸载残留

问题现象

NGINX Ingress控制器插件处于Unknown状态时，卸载插件会出现组件残留。

NGINX Ingress控制器插件涉及的K8s资源：

- 命名空间级别资源：secret、configmap、deployment、service、role、rolebinding、lease、serviceAccount、job
- 集群级别资源：clusterRole、clusterRoleBinding、ingressClass、validatingWebhookConfiguration

解决方案

步骤1 使用kubectl连接集群。

步骤2 查找NGINX Ingress相关资源。

```
className="nginx"
namespace="kube-system"
className=`if [[ ${className} == "nginx" ]]; then echo ""; else echo "-${className}";fi`
kubectl get -n ${namespace} secret sh.helm.release.v1.cceaddon-nginx-ingress${className}.v1 cceaddon-
nginx-ingress${className}-admission
kubectl get -n ${namespace} cm cceaddon-nginx-ingress${className}-controller
kubectl get -n ${namespace} deploy cceaddon-nginx-ingress${className}-controller cceaddon-nginx-ingress
${className}-default-backend
kubectl get -n ${namespace} svc cceaddon-nginx-ingress${className}-controller-admission cceaddon-nginx-
ingress${className}-default-backend cceaddon-nginx-ingress${className}-controller
kubectl get -n ${namespace} role cceaddon-nginx-ingress${className}
kubectl get -n ${namespace} rolebinding cceaddon-nginx-ingress${className}
kubectl get -n ${namespace} lease ingress-controller-leader${className}
kubectl get -n ${namespace} serviceAccount cceaddon-nginx-ingress${className}
kubectl get clusterRole cceaddon-nginx-ingress${className}
kubectl get clusterRoleBinding cceaddon-nginx-ingress${className}
kubectl get ingressClass ${className}
kubectl get ValidatingWebhookConfiguration cceaddon-nginx-ingress${className}-admission
```

其中className为控制器名称，namespace为安装NGINX Ingress控制器的命名空间。

步骤3 如果集群中存在上述资源，请手动删除残留资源。

----结束

10.12 NGINX Ingress 控制器插件升级后无法使用 TLS v1.0 和 v1.1

问题现象

NGINX Ingress控制器插件升级至2.3.3及以上版本后，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。

```
[root@~]# curl -I --tls-max 1.1 -kv https://192.168.0.141:443
* Trying 192.168.0.141:443...
* Connected to 192.168.0.141 (192.168.0.141) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
* CAfile: /etc/pki/tls/certs/ca-bundle.crt
* CApath: none
* TLSv1.1 (OUT), TLS handshake, Client hello (1):
* TLSv1.1 (IN), TLS alert, protocol version (582):
* error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version
* Closing connection 0
curl: (35) error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version
```

解决方法

2.3.3及以上版本的NGINX Ingress默认仅支持TLS v1.2及v1.3版本，如果需要使用更多TLS版本，您可以在NGINX Ingress控制器插件配置的ssl-ciphers参数中添加@SECLEVEL=0字段，以启用对更多TLS版本的支持。更多详情请参见[TLS/HTTPS](#)。

⚠ 注意

为确保数据传输安全，强烈建议避免启用TLS 1.0/1.1。这些协议已被弃用，存在安全风险，可能导致数据泄露或被攻击者利用。建议升级到TLS 1.2或更高版本，以确保通信安全。

- 步骤1** 登录CCE控制台，进入集群，在左侧导航栏中选择“插件中心”，单击NGINX Ingress控制器下的“管理”。
- 步骤2** 单击对应控制器实例的“编辑”按钮。
- 步骤3** 在“nginx 配置参数”中添加以下配置。

```
{
  "ssl-ciphers": "@SECLEVEL=0 ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA",
  "ssl-protocols": "TLSv1 TLSv1.1 TLSv1.2 TLSv1.3"
}
```

- 步骤4** 单击“确定”。
- 步骤5** 重新使用TLS v1.1进行访问，响应正常。


```
[root@... ]# curl -I --tls-max 1.1 -kv https://192.168.0.141:443
* Trying 192.168.0.141:443...
* Connected to 192.168.0.141 (192.168.0.141) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
* CAfile: /etc/pki/tls/certs/ca-bundle.crt
* CApath: none
* TLSv1.1 (OUT), TLS handshake, Client hello (1):
* TLSv1.1 (IN), TLS handshake, Server hello (2):
* TLSv1.1 (IN), TLS handshake, Certificate (11):
* TLSv1.1 (IN), TLS handshake, Server key exchange (12):
* TLSv1.1 (IN), TLS handshake, Server finished (14):
* TLSv1.1 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.1 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.1 (OUT), TLS handshake, Finished (20):
* TLSv1.1 (IN), TLS handshake, Finished (20):
```

----结束

10.13 NPU 驱动升级失败如何解决？

在NPU驱动升级过程中，可能因版本兼容性或环境配置等问题导致命令报错。本文整理了NPU驱动升级过程中的典型报错场景及其解决方案，您可根据实际报错代码或错误描述快速解决问题。

报错场景一：查询 NPU 驱动信息时，报错-8001

- 执行代码：
npu-smi info
- 报错信息：
dcmi module initialize failed. ret is -8001
- 解决方案：提前在[文档中心](#)下载对应的驱动软件包，再利用./{product name}-npu-driver_x.x.x_linux-{arch}.run--upgrade命令升级驱动，具体示例如下：
./A300-3010-npu-driver_24.1.rc3.1_linux-aarch64.run --upgrade

若回显结果如下，则说明升级成功。

```
Verifying archive integrity... 100% SHA256 checksums are OK. All good.
Uncompressing ASCEND DRIVER RUN PACKAGE 100%
[Driver] [2025-04-17 09:39:16] [INFO]Start time: 2025-04-17 09:39:16
[Driver] [2025-04-17 09:39:16] [INFO]LogFile: /var/log/ascend_seclog/ascend_install.log
[Driver] [2025-04-17 09:39:16] [INFO]OperationLogFile: /var/log/ascend_seclog/operation.log
[Driver] [2025-04-17 09:39:16] [INFO]base version is 24.1.rc3.1.
[Driver] [2025-04-17 09:39:16] [WARNING]Do not power off or restart the system during the
installation/upgrade
[Driver] [2025-04-17 09:39:16] [INFO]set username and usergroup, HwHiAiUser:HwHiAiUser
[Driver] [2025-04-17 09:39:18] [INFO]device hot reset start
/usr/local/Ascend/driver/script/device_hot_reset.sh: line 269: echo: write error: No such device
[Driver] [2025-04-17 09:39:23] [INFO]driver install type: Direct
[Driver] [2025-04-17 09:39:23] [INFO]upgradePercentage:10%
[Driver] [2025-04-17 09:39:29] [INFO]upgradePercentage:30%
[Driver] [2025-04-17 09:39:29] [INFO]upgradePercentage:40%
[Driver] [2025-04-17 09:39:30] [INFO]upgradePercentage:90%
[Driver] [2025-04-17 09:39:33] [INFO]upgradePercentage:100%
[Driver] [2025-04-17 09:39:34] [INFO]Driver package upgraded finished!
[Driver] [2025-04-17 09:39:34] [WARNING]Kernel modules can not be removed, reboot needed for
installation/upgrade to take effect!
[Driver] [2025-04-17 09:39:34] [INFO]End time: 2025-04-17 09:39:3
```

报错场景二：执行 upgrade 命令时，报错 0x0094

- 执行代码：

```
./A300-3010-npu-driver_24.1.rc3.1_linux-x86_64.run --upgrade
```
- 报错信息：
Verifying archive integrity... 100% SHA256 checksums are OK. All good.
Uncompressing ASCEND DRIVER RUN PACKAGE 100%
[Driver] [2025-04-17 09:43:09] [ERROR]ERR_NO:0x0094;ERR_DES:Operation failed, An all-in-one RUN package is found in /usr/local/HiAI, which needs to be uninstalled before proceeding.
- 解决方案：请手动卸载驱动后，重新执行upgrade命令，具体示例如下：

```
./A300-3010-npu-driver_24.1.rc3.1_linux-x86_64.run --uninstall
```

回显结果如下，则说明卸载成功。

```
Verifying archive integrity... 100% SHA256 checksums are OK. All good.  
Uncompressing ASCEND DRIVER RUN PACKAGE 100%  
[Driver] [2025-04-22 15:18:20] [INFO]Start time: 2025-04-22 15:18:20  
[Driver] [2025-04-22 15:18:20] [INFO]LogFile: /var/log/ascend_seclog/ascend_install.log  
[Driver] [2025-04-22 15:18:20] [INFO]OperationLogFile: /var/log/ascend_seclog/operation.log  
[Driver] [2025-04-22 15:18:20] [INFO]base version is 24.1.rc3.1.  
[Driver] [2025-04-22 15:18:45] [INFO]Driver package uninstalled successfully! Reboot needed for  
uninstallation to take effect!  
[Driver] [2025-04-22 15:18:45] [INFO]End time: 2025-04-22 15:18:45
```

报错场景三：执行 install 命令时，报错 0x0091

- 执行代码：

```
./A300-3010-npu-driver_24.1.rc3.1_linux-aarch64.run --install-for-all --full
```
- 报错信息：
Verifying archive integrity... 100% SHA256 checksums are OK. All good.
Uncompressing ASCEND DRIVER RUN PACKAGE 100%
[Driver] [2025-04-17 09:18:42] [INFO]Start time: 2025-04-17 09:18:42
[Driver] [2025-04-17 09:18:42] [INFO]LogFile: /var/log/ascend_seclog/ascend_install.log
[Driver] [2025-04-17 09:18:42] [INFO]OperationLogFile: /var/log/ascend_seclog/operation.log
[Driver] [2025-04-17 09:18:42] [WARNING]Do not power off or restart the system during the
installation/upgrade
[Driver] [2025-04-17 09:18:42] [ERROR]ERR_NO:0x0091;ERR_DES:HwHiAiUser not exists! Please
add HwHiAiUser
[Driver] [2025-04-17 09:18:42] [INFO]End time: 2025-04-17 09:18:42
- 解决方案：请手动添加名为HwHiAiUser的账户，添加后重新执行install命令。执行以下命令，创建名为HwHiAiUser的账户，作为昇腾NPU AI加速平台的默认系统账户，用于安全地管理和运行NPU相关服务及应用程序。

```
adduser HwHiAiUser
```

报错场景四：执行 install 命令时，提醒 reboot

- 执行代码：

```
./A300-3010-npu-driver_24.1.rc3.1_linux-aarch64.run --install-for-all --full
```
- 存在以下两种报错信息：
[Driver] [2025-04-17 10:09:30] [WARNING]Kernel modules can not be removed, **reboot** needed for
installation/upgrade to take effect!
[Driver] [2025-04-17 10:18:47] [INFO]Driver package installed successfully! **Reboot** needed for
installation/upgrade to take effect!
- 解决方案：此时驱动已安装成功，您需要重启节点才可生效。建议在重启节点前进行排水操作，避免对当前节点的现存业务产生影响，具体操作请参见[节点排水](#)。

10.14 NPU 插件从 1.x.x 升级到 2.x.x 后 Pod 无法启动如何解决？

问题现象

NPU插件从1.x.x版本升级到2.x.x版本之后，按照原先的配置部署业务，业务的Pod一直处于Pending状态无法启动。

通过**kubecttl describe pod**命令查询，发现该Pod存在以下报错导致创建容器失败。

Error response from daemon: Duplicate mount point: /usr/local/bin/npu-smi

```
started: false
state:
  waiting:
    message: 'Error response from daemon: Duplicate mount point: /usr/local/bin/npu-smi'
    reason: CreateContainerError
ostIP: 10.94.11.187
hase: Pending
odIP: 10.94.17.129
```

问题原因

由于1.x.x版本的插件不具备给业务容器自动挂载驱动和npu-smi工具的能力，用户需要业务手动挂载主机上的驱动和npu-smi的对应目录。但是2.x.x版本的插件具有给业务容器挂载驱动和npu-smi的能力，而1.x.x版本插件中手动挂载的npu-smi目录与插件自动挂载的npu-smi目录冲突，导致了*Duplicate mount point: /usr/local/bin/npu-smi*的错误。

2.x.x版本的NPU插件会自动为业务容器挂载以下目录：

/usr/local/Ascend/driver/lib64/common
/usr/local/Ascend/driver/lib64/driver
/usr/local/bin/npu-smi

NPU插件为业务容器挂载驱动与npu-smi随版本的情况变化，如下表所示：

类型		1.x.x版本的NPU插件	2.0.0 ~ 2.1.6版本的NPU插件	2.1.7 ~ 最新版本的NPU插件
310系列卡	驱动版本 < 23.0.rc0	您需要在业务容器中手动挂载驱动以及npu-smi	您需要在业务容器中手动挂载驱动以及npu-smi	您需要在业务容器中手动挂载驱动以及npu-smi
	驱动版本 >= 23.0.rc0	您需要在业务容器中手动挂载驱动以及npu-smi	插件可以在业务容器中自动挂载驱动，但是无法挂载npu-smi	插件可以在业务容器中自动挂载驱动以及npu-smi

解决方案

使用2.x.x版本的NPU插件时，您可以在业务的YAML中去掉自行挂载驱动和npu-smi目录的字段，由NPU插件自动挂载。

如果您需要手动挂载，可以将挂载路径/usr/local/bin/npu-smi修改为/usr/local/sbin/npu-smi，解决路径冲突即可。

10.15 CCE AI 套件（NVIDIA GPU）插件版本升级或回滚后，如何对 GPU 节点进行排水？

问题现象

在GPU节点上存在GPU虚拟化负载时，对CCE AI套件（NVIDIA GPU）插件进行版本升级或回滚，可能导致部分组件（GPU虚拟化和运行时组件）出现升级或回滚失败的情况。此时，为了保证插件的正常使用，需要对GPU节点进行排水操作，以排空节点中的GPU虚拟化负载。建议采取滚动排水策略，即每次仅对一个或少量GPU节点进行排水，避免大规模排水对业务产生影响。

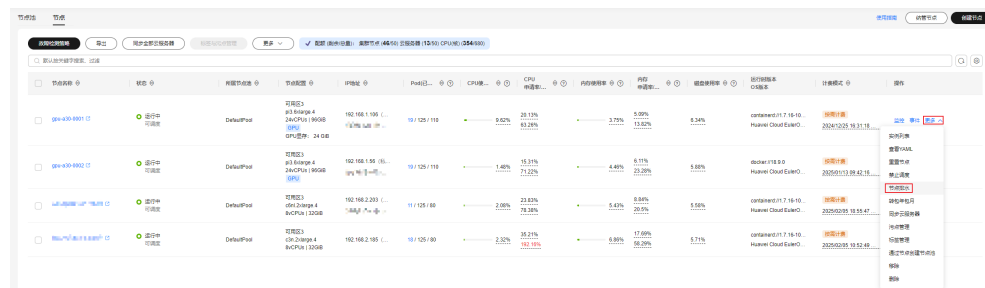
解决方法

在对某个GPU节点进行排水时，请在其他节点中预留足够的GPU资源，以满足节点排空过程中的Pod调度需求，防止资源不足导致Pod调度失败，从而影响业务运行。

步骤1 登录CCE控制台，单击对应集群名称，进入集群“概览”页。

步骤2 在左侧导航栏中，单击“集群 > 节点管理”。在右侧页面，切换至“节点”页签。在目标GPU节点的操作列，单击“更多 > 节点排水”。

图 10-8 节点排水



步骤3 在“节点排水”的弹窗中，单击“确定”。当节点中存在挂载emptyDir卷的Pod或不受controller管理的Pod时，请开启强制排水。

图 10-9 配置排水信息

节点排水

❗ 驱逐节点上所有的Pod（守护进程集 DaemonSet 创建的除外），如负载未设置PDB，Pod重调度期间负载功能可能无法正常使用。驱逐Pod可能触发流控，如遇报错可稍后重试。

超时时间（秒）

0

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

强制排水开启后，将驱逐有emptyDir卷的Pod和删除不受controller管理的Pod

节点名称	所属节点池	备注
gpu-l2-0003	DefaultPool	--

取消

确定

步骤4 在该GPU节点的操作列，单击“更多 > 实例列表”。在“nvidia-gpu-device-plugin-xxx”实例的操作列，单击“更多 > 删除”。在“删除容器组”弹窗中，单击“是”。

此时，“nvidia-gpu-device-plugin-xxx”实例状态为“实例异常”，待其状态恢复为运行中且节点状态列出现“排水成功”时，则说明当前GPU节点已排空GPU虚拟化负载。

图 10-10 删除 nvidia-gpu-device-plugin-xxx 实例

实例列表

批量删除

导出

Q 选择属性筛选，或输入关键字搜索实例名称

实例名称	状态	命名空间	实例ID	所在节点池	重启次数	CPU申请...	内存申请...	创建时间	操作
☐ icagent-rnvt 主机网络	运行中	kube-	192....	192.168.1.1f	0	-- 0.43%	-- 0.17%	2天前	监控 事件 更多
☐ node-exporter-d... 主机网络	运行中	moniti	192....	192.168.1.1f	0	0.2 Cores 0.5 Cores 0.10%	100 MiB 1 GiB 0.93%	2天前	监控 事件 更多
☒ nvidia-gpu-devic... 主机网络	运行中	kube-	192....	192.168.1.1f	0	0.05 Cores 2 Cores 0.28%	30 MiB 500 MiB 10.34%	3天前	监控 事件 更多
☐ hce20-nvidia-dri... 主机网络	运行中	kube-	192....	192.168.1.1f	0	-- 0.00%	-- 0.01%	6天前	监...
☐ cceaddon-npd-... 主机网络	运行中	kube-	192....	192.168.1.1f	0	0.03 Cores 0.3 Cores 4.46%	100 MiB 150 MiB 30.19%	22天前	监...
☐ everest-csi-drive... 主机网络	运行中	kube-	192....	192.168.1.1f	0	0.3 Cores 0.3 Cores 0.35%	300 MiB 300 MiB 22.36%	22天前	监控 事件 更多

日志

容器列表

查看YAML

远程登录

删除

总条数：6

10

<

1

>

步骤5 在该GPU节点的操作列单击“更多 > 开启调度”，从而恢复GPU节点的正常使用。重复以上操作，直至所有GPU节点完成排水。

----结束

10.16 通过 CCE AI 套件（NVIDIA GPU）插件安装的驱动，执行编解码任务时失败

问题现象

在使用以下版本的CCE AI套件（NVIDIA GPU）插件安装驱动后，执行编解码任务时可能失败，并报错“Cuda Encode error:”。

- 集群版本为v1.27及以下时，插件版本在2.2.1以下。
- 集群版本为v1.28及以上时，插件版本在2.8.1以下。

问题原因

上述问题可能是当前Pod所在的GPU节点中缺少10_nvidia.json文件造成的。您可以通过以下步骤进行排查：

步骤1 登录对应的GPU节点。

步骤2 执行以下命令，查看/usr/share/glvnd/egl_vendor.d/10_nvidia.json是否存在。

```
ls /usr/share/glvnd/egl_vendor.d/10_nvidia.json
```

回显结果如下，则说明不存在。

```
ls: cannot access '/usr/share/glvnd/egl_vendor.d/10_nvidia.json': No such file or directory
```

----结束

解决方法

如果对应的GPU节点中缺少10_nvidia.json文件，则需要先将CCE AI套件（NVIDIA GPU）插件版本升级至2.8.1或2.2.1及以上，再重启对应的节点，以重置驱动。对GPU节点进行重启操作之前，建议对其进行排水操作，避免对业务产生影响。

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，并单击“升级”。在“升级插件”页面上方，单击“插件版本”，选择需要升级的版本。

所有参数配置完成后，在右下角单击“确定”。待插件状态由“升级中”变为“运行中”，则说明插件升级成功。

步骤3 依次进行以下操作，重启目标GPU节点。重启GPU节点之前建议排空节点中的Pod，详情请参见[节点排水](#)。在对某个GPU节点进行排水时，请在其他节点中预留足够的GPU资源，以满足节点排空过程中的Pod调度需求，防止资源不足导致Pod调度失败，从而影响业务运行。

1. 在集群左侧导航栏中，单击“节点管理”，切换至“节点”页签，找到目标GPU节点。

- 单击GPU节点名称，跳转到ECS界面，在右上角单击重启。GPU节点完成重启后，需要等待5~10分钟，等待驱动重置。

步骤4 登录节点检查/usr/share/glvnd/egl_vendor.d/10_nvidia.json是否存在。

```
ls /usr/share/glvnd/egl_vendor.d/10_nvidia.json
```

如果文件存在，将输出路径。

----结束

10.17 EulerOS 2.9 系统安装 NVIDIA 驱动失败

问题现象

在EulerOS 2.9系统中，手动安装GPU驱动时，即未通过CCE AI套件（NVIDIA GPU）插件安装驱动，出现错误，且错误提示如下：

```
ERROR: Unable to find the kernel source tree for the currently running kernel. Please make sure you have installed the kernel source files for your kernel and that they are properly configured; on Red Hat Linux systems, for example, be sure you have the 'kernel-source' or 'kernel-devel' RPM installed. If you know the correct kernel source files are installed, you may specify the kernel source path with the '--kernel-source-path' command line option.  
...
```

问题定位

当安装GPU驱动出错时，可以通过以下命令查询驱动日志，从而确定报错原因。

```
cat /var/log/nvidia-installer.log
```

若回显中存在以下内容，则说明未安装GPU驱动的相关依赖项。

```
...  
Cannot generate ORC metadata for CONFIG_UNWINDER_ORC=y, please install libelf-dev, libelf-devel or elfutils-libelf-devel  
...
```

解决方法

您可以采取两种方式解决上述问题：

- 通过CCE AI套件（NVIDIA GPU）插件重新安装GPU驱动，具体请参见[安装CCE AI套件（NVIDIA GPU）插件](#)。
- 先手动安装缺少的依赖项，后重新安装GPU驱动，具体步骤如下：
 - 执行以下命令，安装[问题定位](#)中缺少的依赖项。

```
yum install -y gcc gcc-c++ perl make elfutils-libelf-devel libX11 libXext binutils
```

回显结果如下，则说明依赖项安装成功。

```
...  
Complete!
```
 - 执行以下命令，重新安装驱动。

```
./NVIDIA-Linux-x86_64-535.54.03.run --silent --accept-license
```

 - 若回显结果中无ERROR，则说明驱动成功。例如：

```
Verifying archive integrity... OK  
...  
WARNING: This NVIDIA driver package includes Vulkan components, but no Vulkan ICD loader was detected on this system. The NVIDIA Vulkan ICD will not function without the
```


loader. Most distributions package the Vulkan loader; try installing the "vulkan-loader", "vulkan-icd-loader", "libvulkan" package.

- 若回显中出现如下错误，则需要在安装命令中加入 “--no-drm” 。
ERROR: Unable to load the kernel module 'nvidia-drm.ko'. ...
ERROR: The nvidia-drm kernel module failed to load. This kernel module is required for the proper operation of DRM-KMS. If you do not need to use DRM-KMS, you can try to install this driver package again with the '--no-drm' option.
...
请执行以下命令，重新安装驱动。
`./NVIDIA-Linux-x86_64-535.54.03.run --silent --accept-license --no-drm`

- c. 执行以下命令，验证驱动nvidia-smi是否可以正常使用。
`nvidia-smi`
回显结果如下，则说明nvidia-smi可以正常使用。

图 10-11 回显结果

```
Tue Mar 25 16:29:19 2025
```

NVIDIA-SMI 535.54.03			Driver Version: 535.54.03			CUDA Version: 12.2		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute	M.
							MIG	M.
0	Tesla P4	P0	Off	00000000:00:0D:0	Off	1%	Default	0
N/A	30C		23W / 75W	0MiB / 7680MiB			N/A	

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	
	ID	ID				Usage	
No running processes found							

10.18 Volcano 调度器插件升级后，Volcano Job（vcjob）资源无法正常运行

问题现象

Volcano调度器插件由1.4.7及以下版本升级到1.4.7以后版本时，出现新建的vcjob资源无法正常运行的现象。同时，API Server日志报错信息和volcano-admission组件报错信息分别如下。

API Server的日志报错信息：

```
...
W0318 14:57:51.376736 13 dispatcher.go:142] Failed calling webhook, failing open
validatejob.volcano.sh: failed calling webhook "validatejob.volcano.sh": failed to call webhook: Post "https://volcano-admission-service.kube-system.svc:443/jobs/validate?timeout=30s": EOF
E0318 14:57:51.376768 13 dispatcher.go:149] failed calling webhook "validatejob.volcano.sh": failed to call webhook: Post "https://volcano-admission-service.kube-system.svc:443/jobs/validate?timeout=30s": EOF
...
```

volcano-admission组件报错信息：

```
... no kind "AdmissionReview" is registered for version "admission.k8s.io/v1beta1" ...
```

问题原因

主要原因在于新旧版本的“webhooks.admissionReviewVersions”字段信息不兼容。

在旧版本（1.4.7及以下）中，MutatingWebhookConfiguration和ValidatingWebhookConfiguration资源对象的webhooks.admissionReviewVersions字段值为“v1beta1”，而在新版本（1.4.7以上）中，该字段的值为“v1”。在升级插件时，如果使用了高可用模式，当扩容volcano-admission的副本数时，旧版本的ReplicaSet会启动一个旧版本的“Pod”。如果该Pod未及时销毁，它将强制覆盖MutatingWebhookConfiguration和ValidatingWebhookConfiguration资源中的配置，将“webhooks.admissionReviewVersions”字段重置为“v1beta1”。此时，Kubernetes无法识别该字段信息，从而导致创建的vcjob资源无法正常运行。

具体的MutatingWebhookConfiguration和ValidatingWebhookConfiguration资源对象请参见[表10-5](#)。

表 10-5 涉及的资源对象

资源类型	资源名称
MutatingWebhookConfiguration	volcano-admission-service-jobs-mutate
	volcano-admission-service-podgroups-mutate
	volcano-admission-service-queues-mutate
	volcano-admission-service-pods-mutate
ValidatingWebhookConfiguration	volcano-admission-service-jobs-validate
	volcano-admission-service-pods-validate
	volcano-admission-service-queues-validate

解决方法

请将下列资源对象的“webhooks.admissionReviewVersions”字段信息由“v1beta1”修改为“v1”后，再尝试重新创建vcjob资源。

步骤1 请依次执行**kubectl edit**命令，依次修改[表10-5](#)中字段信息。以“volcano-admission-service-jobs-mutate”为例，介绍如何修改“webhooks.admissionReviewVersions”字段信息。

执行以下命令，修改“volcano-admission-service-jobs-mutate”的Yaml文件。

```
kubectl edit MutatingWebhookConfiguration volcano-admission-service-jobs-mutate
```

在Yaml中，按*Enter*键编辑文件内容，将“webhooks.admissionReviewVersions”由“v1beta1”修改为“v1”。

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  annotations:
    meta.helm.sh/release-name: cceaddon-volcano
    meta.helm.sh/release-namespace: kube-system
    creationTimestamp: "2025-02-25T08:11:52Z"
    generation: 2
  labels:
    app.kubernetes.io/managed-by: Helm
    release: cceaddon-volcano
    name: volcano-admission-service-jobs-mutate
    resourceVersion: "252406"
    uid: 7e9bdaaf-1b6c-4975-a171-ada8456c12e5
webhooks:
```

```
- admissionReviewVersions:
- v1beta1 # 将该内容修改为v1
...
```

修改完成后，按**Esc**键退出编辑，输入**:wq**保存。

步骤2 表10-5中对象的“webhooks.admissionReviewVersions”字段都修改完成后完成，执行以下命令删除未成功运行的vcjob资源。

```
kubectl delete vcjob -n namespace vcjob_name
```

回显结果如下：

```
vcjob vcjob_name deleted
```

步骤3 执行以下命令，重新创建vcjob资源。

```
kubectl create -f vcjob.yaml # 请替换为创建vcjob资源的YAML文件
```

回显结果如下：

```
job.batch.volcano.sh/vcjob_name created
```

步骤4 执行以下命令，查看vcjob资源是否成功创建。

```
kubectl get vcjob_name -n namespace
```

回显结果如下，当STATUS为Running时，则说明vcjob资源创建成功。

NAME	STATUS	MINAVAILABLE	RUNNINGS	AGE
vcjob_name	Running	1	2m30s	

----结束

问题规避措施

在插件升级之前，您可以通过以下措施对以上问题进行规避：

- 从Volcano调度器插件版本≤1.4.7升级到版本>1.4.7且≤1.13.7：在“升级插件”页面，请先将“插件规格”设置为“系统预置规格 > 单实例”或“自定义规格 > 实例数设置1”，然后进行升级。插件升级完成后，在“插件中心”中找到“Volcano调度器”插件，单击“编辑”。在“编辑插件”页面，将“插件规格”修改为所需规格即可。
- 从Volcano调度器插件版本≤1.4.7升级到版本>1.13.7：在“升级插件”页面，请先将“插件规格”切换至“自定义规格”，并将“volcano-admission”组件的副本数设置为1，然后进行升级。插件升级完成后，在“插件中心”中找到“Volcano调度器”插件，单击“编辑”。在“编辑插件”页面，将“插件规格”修改为所需规格即可。

11 API&kubectl

11.1 用户访问集群 API Server 的方式哪些？

当前CCE提供两种访问集群API Server的方式：

- 集群API方式：（推荐）集群API需要使用证书认证访问。直接连接集群API Server，适合大规模调用。
- API网关方式：API网关采用token方式认证，需要使用账号信息获取token。适合小规模调用场景，大规模调用时可能会触发API网关流控。

详情请参见[使用Kubernetes API](#)。

11.2 通过 API 或 kubectl 操作 CCE 集群，创建的资源是否能在控制台展示？

在CCE控制台，暂时不支持显示的kubernetes资源有：**DaemonSet**、**ReplicationController**、**ReplicaSets**、**Endpoints**等。

若需要查询这些资源，请通过kubectl命令进行查询。

此外，Deployment、Statefulset、Service和Pod资源需满足以下条件，才能在控制台显示：

- **Deployment和Statefulset**：标签中必须至少有一个标签是以"app"为key的。
- **Pod**：只有创建了无状态工作负载（Deployment）和有状态工作负载（StatefulSet）后，对应Pod实例才会在工作负载详情页的“实例列表”页签中显示。
- **Service**：Service当前在无状态工作负载（Deployment）和有状态工作负载（StatefulSet）详情页的“访问方式”页签中显示。

此处的显示需要Service与工作负载有一定的关联：

- a. 工作负载中的一个标签必须是以"app"为key。
- b. Service的标签和工作负载的标签保持一致。

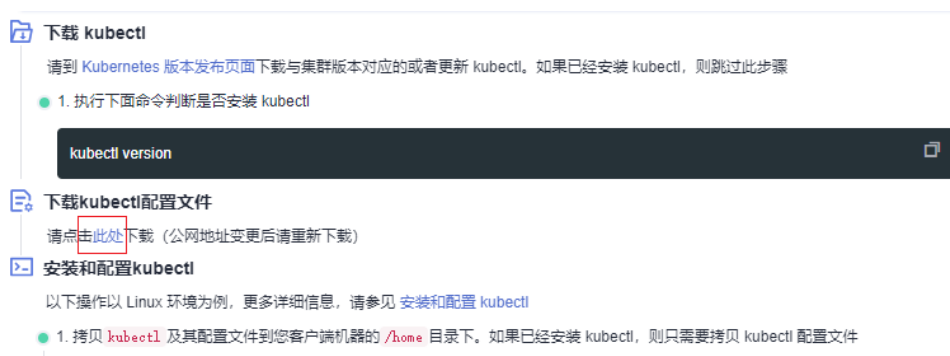
11.3 通过 kubectl 连接集群时，其配置文件 config 如何下载？

步骤1 登录CCE控制台，单击需要连接的集群名称，进入“集群信息”页面。

步骤2 在“连接信息”版块中查看kubectl的连接方式。

步骤3 在弹出的窗口中可以下载kubectl配置文件kubeconfig.json。

图 11-1 下载 kubeconfig.json



----结束

11.4 kubectl top node 命令为何报错

故障现象

执行kubectl top node命令报错Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)

可能原因

执行kubectl时出现Error from server (ServiceUnavailable)时，表示未能连接到集群，需要检查kubectl到集群Master节点的网络是否能够连通。

解决方法

- 如果是在集群外部执行kubectl，请检查集群是否绑定公网IP，如已绑定，请重新下载kubeconfig文件配置，然后重新执行kubectl命令。
- 如果是在集群内节点上执行kubectl，请检查节点的安全组，是否放通Node节点与Master节点TCP/UDP互访，安全组的详细说明请参见[集群安全组规则配置](#)

11.5 kubectl 使用报错：Error from server (Forbidden)

故障现象

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"
```

问题根因

用户没有操作该Kubernetes资源的权限。

解决方法

给该用户授权Kubernetes权限，具体方法如下。

- 步骤1 登录[CCE控制台](#)。
- 步骤2 在左侧导航栏中选择“权限管理”，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 11-2 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 11-3 自定义权限

新建自定义权限

名称

请输入名称

类型

ClusterRole

Role

规则

对于全部操作推荐配置 *。

对于只读操作推荐配置 get + list + watch。

对于读写操作推荐配置 get + list + watch + create + update + patch + delete。

可执行操作

指定资源

添加

步骤5 单击“确定”。

----结束

12 域名 DNS

12.1 CCE 集群内域名解析失败，如何定位处理？

排查项一：检查是否已安装 CoreDNS 插件

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“插件中心”，确认异常的集群是否已安装CoreDNS插件。

步骤3 如果未安装，请安装。详情请参见[为什么CCE集群的容器无法通过DNS解析？](#)

----结束

排查项二：检查 CoreDNS 实例是否已到达性能瓶颈

CoreDNS所能提供的域名解析QPS与CPU消耗呈正相关，如遇QPS较高的场景，需要根据QPS的量级调整CoreDNS实例规格。集群超过100节点时，推荐使用NodeLocal DNSCache提升DNS性能，详情请参见[使用NodeLocal DNSCache提升DNS性能](#)。

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“插件中心”，确认CoreDNS插件状态为“运行中”。

步骤3 单击CoreDNS插件名称，查看插件实例列表。

步骤4 单击CoreDNS实例的“监控”按钮，查看实例CPU、内存使用率。

如实例已达性能瓶颈，则需调整CoreDNS插件规格。

1. 单击CoreDNS插件的“编辑”按钮，进入插件详情页。
2. 在“规格配置”下配置CoreDNS参数规格。您可以根据业务需求选择CoreDNS所能提供的域名解析QPS。
3. 您也可以选择自定义规格，通过选择不同的副本数、CPU配额和内存配额，来定制集群的CoreDNS参数规格。

规格配置

插件规格

系统预置规格 自定义规格

配置清单

部署组件数量	预估资源最少占用量
1	CPU 8000m, 内存 8192Mi

组件名称	部署方式	副本数 (个)	CPU配额	内存配额	组件说明
coredns	Deployment	- 4 +	申请 2000 m	申请 2048 Mi	CoreDNS是一款能够按需组合, 灵活配置的...
			限制 2000 m	限制 2048 Mi	

请保证集群下节点资源充足, 若资源不足, 插件实例无法调度。

4. 单击“确定”，完成配置下发。

----结束

排查项三：解析外部域名很慢或超时

如果域名解析失败率低于1/10000，请参考[解析外部域名很慢或超时，如何优化配置？](#)进行参数优化，或在业务中增加重试。

排查项四：概率性出现 UnknownHostException

集群中的业务请求到外部域名服务器时发生域名解析错误，概率性出现UnknownHostException。UnknownHostException是一个常见的异常，发生该异常时优先检查域名是否存在问题或键入错误。

您可根据以下步骤进行排查：

步骤1 仔细检查主机名是否正确，检查域名的拼写并删除多余的空格。

步骤2 检查DNS设置。在运行应用程序之前，通过**ping hostname**命令确保DNS服务器已启动并正在运行。如果主机名是新的，则需要等待一段时间才能访问DNS服务器。

步骤3 检查CoreDNS实例的CPU、内存使用率监控，确认是否已到达性能瓶颈，具体操作步骤请参见[排查项二：检查CoreDNS实例是否已到达性能瓶颈](#)。

步骤4 检查CoreDNS是否有发生限流，如果触发限流可能出现部分请求处理时间延长，需要调整CoreDNS插件规格。

登录CoreDNS Pod所在节点，查看以下内容：

```
cat /sys/fs/cgroup/cpu/kubepods/pod<pod_uid>/<coredns容器id>/cpu.stat
```

- <pod uid>为CoreDNS的Pod UID，可通过以下命令获取：
kubectrl get po <pod name> -nkube-system -ojsonpath='{.metadata.uid}'{"\n"}

以上命令中的<pod name>需要是在当前节点上运行的CoreDNS Pod名称。

- <coredns容器id>需要是完整的容器ID，可通过以下命令获取：

docker节点：

```
docker ps --no-trunc | grep coredns | awk '{print $1}'
```

containerd节点：

```
crictl ps --no-trunc | grep coredns | awk '{print $1}'
```

完整的命令示例如下：


```
cat /sys/fs/cgroup/cpu/kubepods/  
pod27f58662-3979-448e-8f57-09b62bd24ea6/6aa98c323f43d689ac47190bc84cf4fadd23bd8dd25307f773df2  
5003ef0eef0/cpu.stat
```

请关注以下指标：

- `nr_throttled`：被限流次数。
- `throttled_time`：被限流的总时间长度（纳秒）。

---结束

如果检查后无上述问题，可采用下方优化策略。

优化策略：

1. 修改CoreDNS的缓存时间
2. 配置存根域
3. 修改ndots

说明

- **增加coredns的缓存时间**：有利于同一个域名的第N次解析，减少级联DNS的请求数量。
- **配置存根域**：有利于减少DNS请求链路。

修改方式：

1. 修改CoreDNS缓存时间及配置存根域
修改方法请参见[为CoreDNS配置存根域](#)。

修改完成后重启CoreDNS。

2. 修改ndots

修改方法请参见[解析外部域名很慢或超时，如何优化配置？](#)。

示例：

```
dnsConfig:  
  options:  
    - name: timeout  
      value: '2'  
    - name: ndots  
      value: '5'  
    - name: single-request-reopen
```

建议值修改成：2

12.2 为什么 CCE 集群的容器无法通过 DNS 解析？

问题描述

某客户在DNS服务中做内网解析，将自有的域名绑定到DNS服务中的内网域名中，并绑定到特定的VPC中，发现本VPC内的节点（ECS）可以正常解析内网域名的记录，而VPC内的容器则无法解析。

适用场景

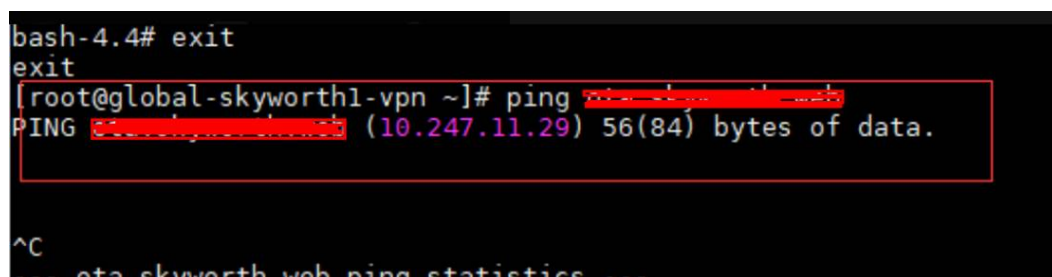
VPC内的容器无法进行正常DNS解析的情况。

解决方案

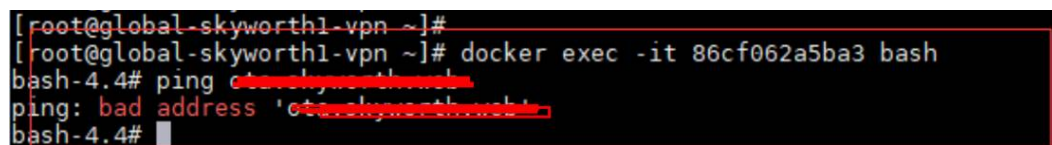
由于本案例涉及的是内网域名解析，按照内网域名解析的规则，需要确保VPC内的子网DNS设置成的云上DNS，具体以内网DNS服务的控制台界面提示为准。



本案例的子网中已经完成该设置，其中用户可以在该VPC子网内的节点（ECS）进行域名解析，也说明已完成该设置，如下图：



但是在容器内进行解析却提示bad address无法解析域名返回地址，如下图：



登录CCE控制台查看该集群的插件安装情况。

如果已安装插件列表中没有coredns插件，可能是用户卸载了该插件等原因导致。

安装coredns插件，并添加相应的域名及对应的DNS服务地址，即可进行域名解析。

12.3 为什么修改子网 DNS 配置后，无法解析租户区域名？

问题描述

用户集群子网DNS配置，增加了DNS服务器配置，如114.114.114.114，该域名无法解析租户区域名。

根因分析

CCE会将用户的子网DNS信息配置到node节点上，coredns插件中也是使用该配置信息，因此会导致用户在节点容器内解析域名会偶发失败的状况。

解决方案

建议您通过修改coredns插件的存根域更新用户集群子网DNS配置，修改方法请参见[CoreDNS配置存根域](#)。

12.4 解析外部域名很慢或超时，如何优化配置？

工作负载的容器内的resolv.conf文件，示例如下：

```
root@test-5dffdddf95-vpt4m:/# cat /etc/resolv.conf
nameserver 10.247.3.10
search istio.svc.cluster.local svc.cluster.local cluster.local
options ndots:5 single-request-reopen timeout:2
```

其中：

- **nameserver**：DNS服务器的IP地址，此处为coredns的ClusterIP。
- **search**：域名的搜索列表，此处为Kubernetes的常用后缀。
- **ndots**：“.”的个数小于它的域名，会优先使用search进行解析。
- **timeout**：超时时间。
- **single-request-reopen**：发送A类型请求和AAAA类型请求使用不同的源端口。

在界面创建工作负载时，以上几项配置默认都会创建，具体参数如下：

```
dnsConfig:
  options:
    - name: timeout
      value: '2'
    - name: ndots
      value: '5'
    - name: single-request-reopen
```

以上参数可以根据业务需要进行优化或修改。

场景一：解析外部域名慢

优化方案：

1. 如果此工作负载不需要访问集群内的k8s服务，可以参考[如何设置容器内的DNS策略？](#)。
2. 如果此工作服务访问其他的k8s服务时，使用的域名中“.”的个数小于2，可以将ndots参数设置为2。

场景二：解析外部域名超时

优化方案：

1. 通常业务内的超时时间要大于timeout * attempts的时间。
2. 如果解析此域名通常要超过2s，可以将timeout改大。

12.5 如何设置容器内的 DNS 策略？

CCE支持通过dnsPolicy标记每个Pod配置不同的DNS策略：

- **None**：表示空的DNS设置，这种方式一般用于想要自定义DNS配置的场景，而且，往往需要和dnsConfig配合一起使用达到自定义DNS的目的。

- **Default:** 从运行所在的节点继承名称解析配置。即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）。
- **ClusterFirst:** 这种方式表示 Pod 内的 DNS 使用集群中配置的 DNS 服务，简单来说，就是使用 Kubernetes 中 kubedns 或 coredns 服务进行域名解析。如果解析不成功，才会使用宿主机的 DNS 配置进行解析。

如果未明确指定 dnsPolicy，则默认使用 “ClusterFirst”：

- 如果将 dnsPolicy 设置为 “Default”，则名称解析配置将从运行 pod 的工作节点继承。
- 如果将 dnsPolicy 设置为 “ClusterFirst”，则 DNS 查询将发送到 kube-dns 服务。
对于以配置的集群域后缀为根的域的查询将由 kube-dns 服务应答。所有其他查询（例如，www.kubernetes.io）将被转发到从节点继承的上游名称服务器。在此功能之前，通常通过使用自定义解析程序替换上游 DNS 来引入存根域。但是，这导致自定义解析程序本身成为 DNS 解析的关键路径，其中可伸缩性和可用性可能导致集群丢失 DNS 功能。此特性允许用户在不接管整个解析路径的情况下引入自定义解析。

如果某个工作负载不需要使用集群内的 coredns，可以使用 kubectl 命令或 API 将此策略设置为 dnsPolicy: Default。

12.6 如何解决 CoreDNS 使用废弃 API 问题？

问题描述

在 CCE 集群中，如果 CoreDNS 在 1.19 集群版本中启动，集群升级到 1.25 版本或 1.27 版本时，CoreDNS 可能仍在监听 v1beta1 版本的 EndpointSlice 资源。由于 v1beta1 版本的 EndpointSlice 资源在 CCE 1.25 集群版本中被废弃，导致 CoreDNS 无法获取到最新的 Service/Endpoint 变更事件，导致以下影响：

- Service 域名解析失败：使用 Service 进行服务发现的业务，新建的 Service 域名无法正常解析。
- Headless Service 解析异常：使用 Headless Service 进行服务发现的业务，当工作负载变更后，Pod 重建，CoreDNS 无法解析到新的 Pod IP 地址，返回缓存中旧的 Pod IP 地址，影响业务访问。

根因分析

在 CCE 1.19 集群中，CoreDNS 使用的是社区 1.8.4 版本，该版本 CoreDNS 在启动时会优先选择 v1 版本的 EndpointSlice 资源，如果不存在 v1 版本则会使用 v1beta1 版本的 EndpointSlice 资源，由于 Kubernetes 1.19 版本中只有 v1beta1 版本的 EndpointSlice 资源，所以 CoreDNS 通过访问 v1beta1 版本的 EndpointSlice 资源获取 Service/Endpoint 变更事件。Kubernetes 在 1.25 版本中移除了 v1beta1 版本的 EndpointSlice API，所以当集群版本升级到 1.25 后，CoreDNS 继续监听被移除的 v1beta1 版本的 EndpointSlice 资源，导致无法获取 Service/Endpoint 事件。

您可以查看 CoreDNS 日志，操作方式请参见[容器日志查看方法](#)，如果日志中出现以下告警，表明 CoreDNS 正在监听已废弃的 API：

```
[ERROR] plugin/kubernetes: k8s.io/client-go@v0.26.1/tools/cache/reflector.go:169: Failed to watch *v1beta1.EndpointSlice: failed to list *v1beta1.EndpointSlice: the server could not find the requested resource
```

```
[INFO] plugin/kubernetes: k8s.io/client-go@v0.26.1/tools/cache/reflector.go:169: failed to list *v1beta1.EndpointSlice: the server could not find the requested resource
```

解决方案

重启CoreDNS Pod，让CoreDNS监听v1版本的EndpointSlice资源。

⚠ 注意

重启CoreDNS Pod可能会对集群内的域名解析服务造成短暂的波动，建议您在业务低峰期操作。

您可以通过以下流程重建CoreDNS Pod：

1. 登录CCE控制台，选择一个集群，在左侧导航栏中单击“工作负载”，“命名空间”选择kube-system。
2. 选择coredns负载，单击“重新部署”。



3. 等待coredns负载重新运行即可。

13 镜像仓库

13.1 如何制作 Docker 镜像？如何解决拉取镜像慢的问题？

Docker 镜像制作

关于如何通过Dockerfile定制一个简单的Web应用程序的Docker镜像，请参见[Docker 基础知识](#)或[如何制作Docker镜像？](#)

拉取镜像加速

由于运营商网络问题可能导致公共镜像仓库中的镜像拉取速度缓慢，您可将常用的镜像上传至容器镜像服务SWR，提高镜像拉取速度。

相关说明

容器镜像服务（Software Repository for Container，SWR）是一种支持容器镜像全生命周期管理的服务，提供简单易用、安全可靠的镜像管理功能，帮助用户快速部署容器化服务。容器镜像服务提供的镜像仓库是用于存储、管理docker容器镜像的场所，可以让使用人员轻松存储、管理、部署docker容器镜像。

容器镜像服务相关问题汇总

https://support.huaweicloud.com/swr_faq/swr_faq_0001.html

13.2 如何上传我的镜像到 CCE 中使用？

镜像的管理是由容器镜像服务（SoftWare Repository）提供的，当前容器镜像服务提供如下上传镜像的方法：

- [客户端上传镜像](#)
- [页面上传镜像](#)

如您需要将Harbor镜像仓库平滑地迁移到容器镜像服务，请参考[跨云Harbor同步镜像至华为云SWR](#)。

14 权限

14.1 能否只配置命名空间权限，不配置集群管理权限？

命名空间权限和集群管理权限是相互独立又相互补充的两个权限体系：

- 命名空间权限：作用于集群内部，用于管理集群资源操作（如创建工作负载等）。
- 集群管理（IAM）权限：云服务层面的权限，用于管理CCE集群与周边资源（如VPC、ELB、ECS等）的操作。

对于IAM Admin用户组的管理员用户来说，可以为IAM子用户授予集群管理权限（如CCE Administrator、CCE FullAccess等），也可以在CCE控制台授予某个集群的命名空间权限。但由于CCE控制台界面权限是由IAM系统策略进行判断，如果IAM子用户未配置集群管理（IAM）权限，该子用户将无法进入CCE控制台。

如果您无需使用CCE控制台，只使用kubectl命令操作集群中的资源，则不受集群管理（IAM）权限的影响，您只需要获取具有命名空间权限的配置文件（kubeconfig），详情请参考[如果不配置集群管理权限，是否可以使用kubectl命令呢？](#)。集群配置文件在传递过程中可能存在泄露风险，应在实际使用中注意。

14.2 如果不配置集群管理权限的情况下，是否可以使用 API 呢？

CCE提供的API可以分为云服务接口和集群接口：

- 云服务接口：支持操作云服务层面的基础设施（如创建节点），也可以调用集群层面的资源（如创建工作负载）。
使用云服务接口时，必须配置集群管理（IAM）权限。
- 集群接口：直接通过Kubernetes原生API Server来调用集群层面的资源（如创建工作负载），但不支持操作云服务层面的基础设施（如创建节点）。
使用集群接口时，无需配置集群管理（IAM）权限，仅需在调用集群接口时带上集群证书。但是，集群证书需要有集群管理（IAM）权限的用户进行[下载](#)，在证书传递过程中可能存在泄露风险，应在实际使用中注意。

14.3 如果不配置集群管理权限，是否可以使用 kubectl 命令呢？

使用kubectl命令无需经过IAM认证，因此理论上不配置集群管理（IAM）权限是可以使用kubectl命令的。但前提是需要获取具有命名空间权限的kubectl配置文件（kubeconfig），以下场景认证文件传递过程中均存在安全泄露风险，应在实际使用中注意。

- 场景一

如果某IAM子用户先配置了集群管理权限和命名空间权限，然后在界面下载 kubeconfig 认证文件。后面再删除集群管理权限（保留命名空间权限），依然可以使用kubectl来操作Kubernetes集群。因此如需彻底删除用户权限，必须同时删除该用户的集群管理权限和命名空间权限。

- 场景二

如果某IAM用户拥有一定范围的集群管理权限和命名空间权限，然后在界面下载 kubeconfig 认证文件。此时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中就拥有哪个用户的认证信息，任何人都可以通过这个kubeconfig文件访问集群。

14.4 IAM 用户无法使用调用 API

问题现象

使用IAM用户调用API时，出现以下报错：

```
"code":403,"message": "This user only supports console access, not programmatic access."
```

该错误表示IAM用户没有编程访问权限。

解决方案

- 步骤1** 请联系主账号管理员，登录[统一身份认证服务](#)。
- 步骤2** 找到需要修改的IAM用户，单击用户名称。
- 步骤3** 修改“访问方式”，同时勾选“编程访问”和“管理控制台访问”。

图 14-1 修改 IAM 用户访问方式



- 步骤4** 单击“确定”。

----结束

14.5 什么是 OBS 全局访问密钥？如何排查集群中是否使用全局访问密钥？

在CCE集群中创建OBS类型的PVC时，需要选择一个访问密钥（AK/SK）。OBS的访问密钥可分为以下两种：

- 自定义访问密钥（推荐使用）：使用自定义访问密钥时，PVC的yaml中带有csi.storage.k8s.io/node-publish-secret-namespace和csi.storage.k8s.io/node-publish-secret-name两个annotation，分别表示集群中存放密钥的secret命名空间和secret名称。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: jiooij
  namespace: default
  uid: 43c04970-4951-4f31-942f-6290858a8f98
  resourceVersion: '1348742'
  creationTimestamp: '2024-12-24T07:46:10Z'
  annotations:
    csi.storage.k8s.io/fstype: obsfs
    csi.storage.k8s.io/node-publish-secret-name: aksk-www
    csi.storage.k8s.io/node-publish-secret-namespace: default
    everest.io/enterprise-project-id: '0'
    everest.io/obs-volume-type: standard
    pv.kubernetes.io/bind-completed: 'yes'
```

- 全局访问密钥（仅存量已上传过的用户支持使用，已不推荐）：使用全局访问密钥时，PVC的yaml中不会带有csi.storage.k8s.io/node-publish-secret-namespace和csi.storage.k8s.io/node-publish-secret-name两个annotation，且全局访问密钥的secret固定在kube-system命名空间下，名称为paas.longask。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: sadfdsaf
  namespace: default
  uid: 73675783-76b6-4d09-94fb-aec17d409b4b
  resourceVersion: '1347883'
  creationTimestamp: '2024-12-24T07:42:29Z'
  annotations:
    csi.storage.k8s.io/fstype: obsfs
    everest.io/obs-volume-type: standard
    pv.kubernetes.io/bind-completed: 'yes'
    pv.kubernetes.io/bound-by-controller: 'yes'
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    volume.kubernetes.io/storage-provisioner: everest-csi-provisioner
  finalizers:
    - kubernetes.io/pvc-protection
```

说明

全局访问密钥（paas.longaksk）是项目维度的，使用全局访问密钥后，该项目下的每个集群默认都会创建一个相同的全局访问密钥。由于这种机制可能导致安全性和管理上的复杂性，因此不推荐使用全局访问密钥。

如果用户没有使用全局访问密钥的需求，则可以在集群的“配置中心”中关闭全局访问密钥，操作后CCE会删除kube-system命名空间下的全局访问密钥paas.longaksk。如果后续有使用需求可以在配置中心再次开启。



排查集群中是否使用全局访问密钥

- 通过控制台页面排查

在CCE集群的存储页面，选择全部命名空间，在存储卷声明页签下，添加筛选条件“存储配置”为“obs”，并查看筛选结果列表的存储配置列。

如果访问密钥显示为空，说明使用的是全局访问密钥。如果访问密钥显示不为空，说明使用的是自定义访问密钥。



- 使用kubectl命令行排查

```
kubectl get pvc --all-namespaces -o=custom-columns=Name:'metadata.name',Namespace:'metadata.namespace',secretName:'metadata.annotation.scsi.storage.k8s.io/node-publish-secret-name' | grep \<none>
```

通过以上命令，可以查询出集群中正在使用全局访问密钥的PVC，输出第一列为PVC的名字，第二列为PVC所在的命名空间。

```
obs-test default <none>
```

如果返回为空，说明集群中没有正在使用全局访问密钥的PVC。

15 相关服务

15.1 云容器引擎（CCE）与云容器实例（CCI）的区别是什么？

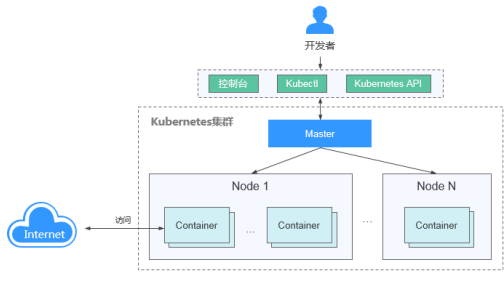
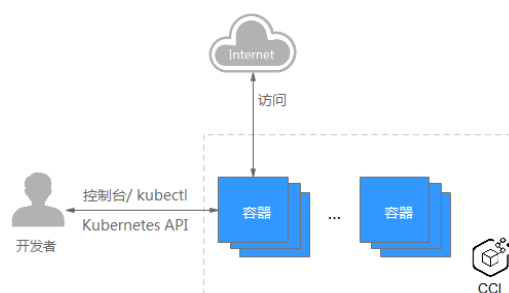
基本介绍

表 15-1 CCE 和 CCI 基本介绍

云容器引擎CCE	云容器实例CCI
云容器引擎（Cloud Container Engine，简称CCE）提供高度可扩展的、高性能的企业级Kubernetes集群，支持运行Docker容器，提供了Kubernetes集群管理、容器应用全生命周期管理、应用服务网格、Helm应用模板、插件管理、应用调度、监控与运维等容器全栈能力，为您提供一站式容器平台服务。借助云容器引擎，您可以在华为云上轻松部署、管理和扩展容器化应用程序。 详细介绍请查看什么是云容器引擎。	云容器实例（Cloud Container Instance，CCI）服务提供Serverless Container（无服务器容器）引擎，让您无需创建和管理服务器集群即可直接运行容器。通过CCI您只需要管理运行在Kubernetes上的容器化业务，无需管理集群和服务器即可在CCI上快速创建和运行容器负载，使容器应用零运维，使企业聚焦业务核心，为企业提供了Serverless化全新一代的体验和选择。 而Serverless是一种架构理念，是指不用创建和管理服务器、不用担心服务器的运行状态（服务器是否在工作等），只需动态申请应用需要的资源，把服务器留给专门的维护人员管理和维护，进而专注于应用开发，提升应用开发效率、节约企业IT成本。传统上使用Kubernetes运行容器，首先需要创建运行容器的Kubernetes服务器集群，然后再创建容器负载。 详细介绍请查看什么是云容器实例。

创建方式

表 15-2 创建方式不同

云容器引擎CCE	云容器实例CCI
CCE是基于Kubernetes的托管式容器管理服务，可以提供原生Kubernetes体验，可以一键创建原生Kubernetes集群，与社区能力基本一致。 使用CCE，您需要创建集群和节点，简单、低成本、高可用，无需管理Master节点。 	CCI提供 Serverless Container引擎，在华为云上部署容器时，您不需要购买和管理ECS，可以直接在华为云上运行容器和Pod，为您省去底层ECS的运维和管理工作。 使用CCI，您无需创建集群，无需创建和管理Master节点及Work节点，可直接启动应用程序。 

收费方式

表 15-3 收费方式不同

维度	云容器引擎CCE	云容器实例CCI
定价	CCE在使用过程中会创建相关资源（如节点、带宽等），您需要为这些资源付费。	CCI实例资源包含CPU、内存、GPU等，根据使用的实际实例资源规格进行计费。
计费方式	支持按需计费、包年/包月两种计费模式	支持按需计费、购买套餐包两种计费模式
最小计价单位	按小时计费	按秒计费，以小时为出账周期

图 15-1 CCE 和 CCI 收费方式区别



应用场景

表 15-4 应用场景不同

云容器引擎CCE	云容器实例CCI
适用所有场景，一般运行大规模长期稳定的应用，例如： • 电商 • 业务中台 • IT系统	适用于有明显的波峰波谷特征的场景，灵活申请资源，提高资源利用率。例如： • 批量计算 • 高性能计算 • 突发扩容 • CI/CD测试

图 15-2 CCE 和 CCI 应用场景区别

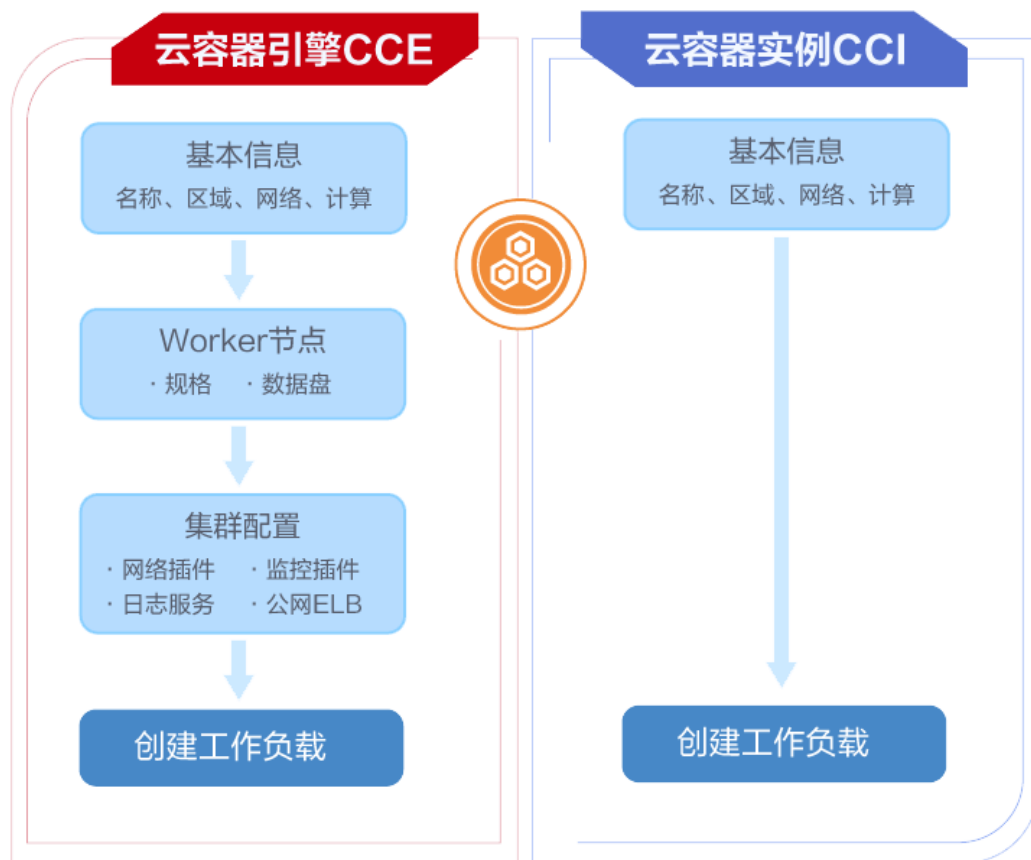


集群创建

表 15-5 创建方式不同

云容器引擎CCE	云容器实例CCI
云容器引擎使用流程如下： 1. 创建集群 配置名称、区域、网络等基本信息。 2. 创建节点 选择节点规格、数据盘大小等配置。 3. 集群配置 安装各类集群插件，如网络、监控、日志等。 4. 创建工作负载	云容器实例使用流程如下： 1. 创建命名空间 配置名称、区域、网络等基本信息。 2. 创建工作负载

图 15-3 CCE 和 CCI 创建集群区别



CCE 与 CCI 两者的配合

通过安装CCE突发弹性引擎（对接CCI）插件，可以在短时高负载场景时，将部署在CCE上的无状态工作负载（Deployment）、有状态工作负载（StatefulSet）、普通任务（Job）三种资源类型的容器实例（Pod），弹性创建到华为云云容器实例CCI服务上，以减少集群扩容带来的消耗。

具体功能如下：

- 支持容器实例实现秒级弹性伸缩：在集群资源不足时，无需新增节点，CCE突发弹性引擎（对接CCI）插件将自动为您在云容器实例CCI侧创建容器实例，减少运维成本。
- 无缝对接华为云容器镜像服务SWR，支持使用公用镜像和私有镜像。
- 支持CCI容器实例的事件同步、监控、日志、exec、查看状态等操作。
- 支持查看虚拟弹性节点的节点容量信息。
- 支持CCE和CCI两侧实例的Service网络互通。

详情请参见[CCE容器实例弹性伸缩到CCI服务](#)。

15.2 云容器引擎（CCE）和应用管理与运维平台（ServiceStage）的区别是什么？

对于使用者而言，云容器引擎关注的重点是Pod的部署，应用管理与运维平台关注的是服务的使用。

对于技术实现来看，应用管理与运维平台是对云容器引擎的再一次封装。

基础概念

云容器引擎（CCE）

云容器引擎（Cloud Container Engine，简称CCE）提供高度可扩展的、高性能的企业级Kubernetes集群，支持运行Docker容器，提供了Kubernetes集群管理、容器应用全生命周期管理、应用服务网格、Helm应用模板、插件管理、应用调度、监控与运维等容器全栈能力，为您提供一站式容器平台服务。借助云容器引擎，您可以在华为云上轻松部署、管理和扩展容器化应用程序。

应用管理与运维平台（ServiceStage）

ServiceStage应用管理与运维平台是一个应用托管和微服务管理平台，可以帮助企业简化部署、监控、运维和治理等应用生命周期管理工作。ServiceStage面向企业提供微服务、移动和Web类应用开发的全栈解决方案，帮助您的各类应用轻松上云，聚焦业务创新，帮助企业数字化快速转型。