容器服务

TKE 标准集群指南

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

TKE 标准集群指南

TKE 标准集群概述

最近更新时间：2025-06-23 11:49:42

产品介绍

腾讯云容器服务（Tencent Kubernetes Engine，TKE）是高度可扩展的高性能容器管理服务，您可以在托管的云服务器实例集群上轻松运行应用程序。使用该服务，您将无需安装、运维、扩展您的集群管理基础设施，只需进行简单的 API 调用，便可启动和停止 Docker 应用程序，查询集群的完整状态，以及使用各种云服务。您可以根据资源需求和可用性要求在集群中安排容器的置放，满足业务或应用程序的特定要求。

腾讯云容器服务 TKE 基于原生 Kubernetes 提供以容器为核心的解决方案，解决用户开发、测试及运维过程的环境问题、帮助用户降低成本，提高效率。腾讯云容器服务 TKE 完全兼容原生 Kubernetes API，并扩展了腾讯云的云硬盘、负载均衡等 Kubernetes 插件，同时以腾讯云私有网络为基础，实现了高可靠、高性能的网络方案。

以下视频介绍了腾讯云容器服务 TKE，及其核心概念和使用流程：

观看视频

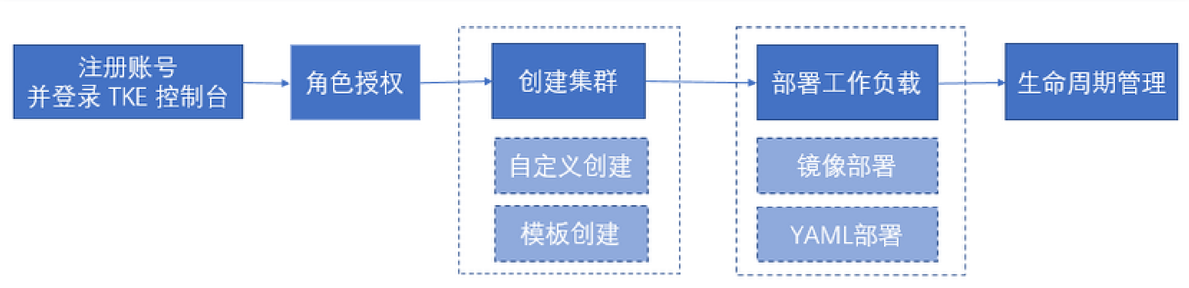
名词解释

使用腾讯云容器服务 TKE，会涉及到以下基本概念：

- **集群**：是指容器运行所需云资源的集合，包含了若干台云服务器、负载均衡器等云资源。
- **实例**：由相关的一个或多个容器构成一个实例（Pod），这些容器共享相同的存储和网络空间。
- **工作负载**：Kubernetes 资源对象，用于管理 实例（Pod）副本的创建、调度以及整个生命周期的自动控制。
- **Service**：由多个相同配置的实例（Pod）和访问这些实例（Pod）的规则组成的微服务。
- **Ingress**：Ingress 是用于将外部 HTTP（S）流量路由到服务（Service）的规则集合。
- **应用**：是指腾讯云容器服务 TKE 集成的 Helm 3.0 相关功能，为您提供创建 Helm chart、容器镜像、软件服务等各种产品和服务的能力。
- **镜像仓库**：用于存放 Docker 镜像，Docker 镜像用于部署容器服务。

使用流程

腾讯云容器服务 TKE 使用流程如下图所示：



1. 角色授权
   注册并登录 容器服务控制台，完成服务授权获取相关资源操作权限，即可开始使用容器服务产品。
2. 创建集群
3. 可自定义新建集群。
4. 部署工作负载
   支持使用镜像部署、YAML 文件编排两种方式部署工作负载，详情请参考 工作负载管理。
5. 完成工作负载创建后，通过监控、升级、伸缩等操作对 Pod 生命周期进行管理。

产品定价

腾讯云容器服务 TKE 针对不同规格的托管集群，会收取相应的集群管理费用，以及用户实际使用的云资源费用。关于收费模式和具体价格，请参阅 容器服务计费概述。

相关服务

- 通过购买若干个云服务器组成容器服务集群，容器运行在云服务器中。有关更多信息，请参阅 云服务器产品文档。
- 集群可以建立在私有网络下，集群内主机可以分配在不同可用区的子网下。有关更多信息，请参阅 私有网络产品文档。

- 可以使用负载均衡，自动分配横跨多个云服务实例的客户端请求流量，转发至主机内容器。有关更多信息，请参阅 负载均衡产品文档 。
- 监控容器服务集群和容器实例的运行统计数据，可使用腾讯云可观测平台。有关更多信息，请参阅 腾讯云可观测平台产品文档 。

购买 TKE 标准集群

计费概述

最近更新时间：2024-08-16 15:11:51

计费项

当您使用容器服务 TKE 时，产品费用由**集群管理费用**和**云产品资源费用**组成。

- **集群管理费用**
  集群管理费用**仅针对托管集群收取**。TKE 托管集群提供了高可用、高性能、可伸缩、高稳定的全托管控制面，简化集群的搭建和扩容等操作，让您可专注于容器化应用的开发，而无需关心集群的管理以及维护，TKE 针对不同规格的托管集群，会收取相应的集群管理费用。具体收费细则，可参考 集群管理费用 。
- **云产品资源费用**
  您在使用 TKE 过程中创建的其他的云产品资源（CVM、CBS、CLB 等），将按照各自云产品的计费方式进行收费。具体收费细则，可参考 云产品资源费用 。

集群管理费用

计费模式

容器服务计费方式主要为按量计费（后付费）。

计费项	计费模式	付款方式	计费单位
集群（个）	按量计费	购买时 冻结费用，每小时结算	元/小时

产品定价

注意：
- 具体单价可能因地域调整变化，请以控制台实际展示价格为准。
- 选择规格前请仔细阅读 购买说明 。
- 集群费用的计费周期自集群创建完成时开始，集群创建时间可在**容器服务控制台 > 集群基本信息**中查看。

集群规格	定价 （元/小时）
L5	0.13
L20	0.40
L50	0.73
L100	1.22
L200	2.55
L500	5.11
L1000	9.38
L3000	15.60
L5000	28.04

云产品资源费用

您在使用 TKE 过程中创建的其他的云产品资源（CVM、CBS、CLB 等），将按照各自云产品的计费方式进行收费。具体收费细则，可参考各产品的收费说明。

云产品	计费说明
云服务器 CVM	云服务器计费模式
云硬盘 CBS	云硬盘价格总览
负载均衡 CLB	负载均衡计费说明

> **注意：**
> 容器服务基于 Kubernetes 且为声明式服务，当不再需要容器服务创建的负载均衡 CLB、云硬盘 CBS 等 IaaS 服务资源时，请勿通过具体服务资源的管理界面删除，应该在容器服务控制台中删除相关服务资源，否则容器服务会重新创建被删除的服务资源，并继续扣除相关费用。例如您已在容器服务中创建负载均衡 CLB 服务资源，当在负载均衡控制台中删除该 CLB 实例时，容器服务会根据声明式 API 重新创建新的 CLB 实例。

购买说明

最近更新时间：2024-11-27 14:23:12

购买须知

> **说明：**
> 因为不遵循本购买须知导致的服务不可用，相应服务不可用时间不属于服务不可用的计算范畴，更多请参考 《容器服务服务等级协议》。

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

常见的**类 List 操作**如下（以 Pod 资源为例）：

- 带标签查询

```
kubectl get pod -l app=nginx
```

- 指定 namespace 查询

```
kubectl get pod -n default
```

- 查询整个集群的 pod 等

```
kubectl get pod --all-namespaces
```

- 通过 client-go 发起的 List 请求

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

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

推荐配置

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

例如，如果您计划在一个集群中部署50个节点，但是计划部署2000个 Pod, 则应该选用最大管理节点规模为100（而非50）的集群规格。

> **说明：**
> - 节点指 Kubernetes Node，包含节点类型有普通节点、原生节点。**注意：节点规模计数不包含超级节点。**
> - Pod 包括所有 Namespace 下，所有状态的 Pod，但不包括系统组件相关 Pod（cni-agent 等）。
> - CRD 不统计 TKE 系统组件产生的 CRD（如 VPC-CNI 相关组件产生的 CRD）。
> - **最大其他 K8s 资源数量**指除表格中单独列出的资源外，其他的 K8s 资源的最大数量。例如您购买了最大管理节点规模为L100的集群，为了保障集群的可用性，集群中的 ClusterRole、Service、Endpoint 等 K8s 资源的数量均不应该超过2500。
> - 建议每种资源类型的所有对象总和不应超过800MiB，每个资源对象大小不超过100KB。

集群规格	最大管理节点数量	最大 Pod 数量（推荐）	最大 replicasets 数量	最大 ConfigMap 数量（推荐）	最大 CRD 数量 / 最大其他 K8s 资源数量（推荐）
L5	5	150	900	128	150
L20	20	600	3600	256	600

L50	50	1500	9000	512	1250
L100	100	3000	18000	1024	2500
L200	200	6000	36000	2048	5000
L500	500	15000	90000	4096	10000
L1000	1000	30000	180000	6144	20000
L3000	3000	90000	540000	8192	50000
L5000	5000	150000	900000	10240	100000

欠费说明

最近更新时间：2023-05-17 15:40:17

欠费说明

从您的**账户欠费**（账户余额被扣为负值时刻）起：
- 24小时内，容器服务托管集群可继续使用且继续扣费。
- 24小时后，账户内托管集群将全部进入**隔离中**状态且停止扣费。集群被隔离后，API Server 无法访问，Node 节点上的业务不受影响。

> **说明**
> 如果您的账户于**收费缓冲期结束前**，即2022年4月1日上午10：00（北京时间）之前已经欠费，针对**2022年3月21日10:00（北京时间）前创建的容器服务 TKE 托管集群**, 腾讯云会在缓冲期结束后（2022年4月1日上午10：00之后）隔离集群，缓冲期内您仍可以正常使用。

欠费处理

托管集群将进入**隔离中**状态后，随着隔离时间的增长，腾讯云对集群进行如下处理：

> **注意**
> 以下说明仅列出了托管集群层面的欠费处理，集群中的 CVM、CLB 等资源的欠费处理逻辑请参考对应产品的欠费说明。

隔离后时间	说明
≤ 15天	若您的账户充值至余额大于0，恢复计费，集群将自动恢复至运行中状态。
	若您的账户余额尚未充值到大于0，则集群保持为隔离中状态。
> 15天	若您的账户仍欠费，集群会被删除，集群删除后不可恢复。如集群中仍有节点，所有节点会被移出集群，TKE 不会对节点做任何其他操作。集群被删除时，我们将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

地域和可用区

最近更新时间：2025-03-24 18:01:12

地域

简介

地域（Region）是指物理的数据中心的地理区域。腾讯云不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。
您可以查看下表或者通过 API 接口 查询地域列表 查看完整的地域列表。

相关特性

- 不同地域之间的网络完全隔离，不同地域之间的云产品**默认不能通过内网通信**。
- 不同地域之间的云产品，可以通过 公网 IP 访问 Internet 的方式进行通信。处于不同私有网络的云产品，可以通过 云联网 进行通信，此通信方式较为高速、稳定。
- 负载均衡 当前默认支持同地域流量转发，绑定本地域的云服务器。如果开通 跨地域绑定 功能，则可支持负载均衡跨地域绑定云服务器。

> **说明：**
> 针对地域为**深圳 / 上海金融专区**的金融行业监管要求定制的合规专区，具有高安全，高隔离性的特点。目前提供云服务器、金融数据库、Redis 存储、人脸识别等服务，已认证通过的金融行业客户可通过 在线咨询 申请使用专区。详见 金融专区介绍 。

可用区

简介

可用区（Zone）是指腾讯云在同一地域内电力和网络互相独立的物理数据中心。其目标是能够保证可用区间故障相互隔离（大型灾害或者大型电力故障除外），不出现故障扩散，使得用户的业务持续在线服务。通过启动独立可用区内的实例，用户可以保护应用程序不受单一位置故障的影响。
您可以通过 API 接口 查询可用区列表 查看完整的可用区列表。

相关特性

处于相同地域不同可用区，但在同一个私有网络下的云产品之间均通过内网互通，可以直接使用 内网 IP 访问。

> **说明：**
> 内网互通是指同一账户下的资源互通，不同账户的资源内网完全隔离。

中国

地域	可用区
华南地区（广州） ap-guangzhou	广州一区（已售罄） ap-guangzhou-1
	广州二区（已售罄） ap-guangzhou-2
	广州三区 （已售罄） ap-guangzhou-3
	广州四区 （已售罄） ap-guangzhou-4
	广州六区 ap-guangzhou-6
	广州七区 ap-guangzhou-7

华南地区（深圳金融） ap-shenzhen-fsi	深圳金融一区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shenzhen-fsi-1
	深圳金融二区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shenzhen-fsi-2
	深圳金融三区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shenzhen-fsi-3
华东地区（上海） ap-shanghai	上海一区（已售罄） ap-shanghai-1
	上海二区 ap-shanghai-2
	上海三区（已售罄） ap-shanghai-3
	上海四区（已售罄） ap-shanghai-4
	上海五区 ap-shanghai-5
	上海八区 ap-shanghai-8
华东地区（上海金融） ap-shanghai-fsi	上海金融一区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shanghai-fsi-1
	上海金融二区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shanghai-fsi-2
	上海金融三区（仅限金融机构和企业通过 在线咨询 申请开通） ap-shanghai-fsi-3
华东地区（南京） ap-nanjing	南京一区 ap-nanjing-1
	南京二区（已售罄） ap-nanjing-2
	南京三区 ap-nanjing-3
华北地区（北京） ap-beijing	北京一区（已售罄） ap-beijing-1
	北京二区（已售罄） ap-beijing-2
	北京三区 ap-beijing-3
	北京四区（已售罄） ap-beijing-4
	北京五区（已售罄） ap-beijing-5
	北京六区 ap-beijing-6
	北京七区 ap-beijing-7
西南地区（成都）	成都一区

ap-chengdu	ap-chengdu-1
	成都二区（已售罄） ap-chengdu-2
西南地区（重庆） ap-chongqing	重庆一区 ap-chongqing-1
港澳台地区（中国香港） ap-hongkong	香港一区（中国香港节点可用于覆盖港澳台地区） ap-hongkong-1
	香港二区（中国香港节点可用于覆盖港澳台地区） ap-hongkong-2
	香港三区（中国香港节点可用于覆盖港澳台地区） ap-hongkong-3

说明：
济南、杭州、福州、武汉、长沙、石家庄地域目前处于内测中，如需使用，请联系商务经理申请。

其他国家和地区

地域	可用区
亚太东南（新加坡） ap-singapore	新加坡一区（新加坡节点可用于覆盖亚太东南地区） ap-singapore-1
	新加坡二区（新加坡节点可用于覆盖亚太东南地区） ap-singapore-2
	新加坡三区（新加坡节点可用于覆盖亚太东南地区） ap-singapore-3
	新加坡四区（新加坡节点可用于覆盖亚太东南地区） ap-singapore-4
亚太东南（雅加达） ap-jakarta	雅加达一区（雅加达节点可用于覆盖亚太东南地区） ap-jakarta-1
	雅加达二区（雅加达节点可用于覆盖亚太东南地区） ap-jakarta-2
亚太东北（首尔） ap-seoul	首尔一区（首尔节点可用于覆盖亚太东北地区） ap-seoul-1
	首尔二区（首尔节点可用于覆盖亚太东北地区） ap-seoul-2
亚太东北（东京） ap-tokyo	东京一区（东京节点可用区覆盖亚太东北地区） ap-tokyo-1
	东京二区（东京节点可用区覆盖亚太东北地区） ap-tokyo-2
亚太东南（曼谷） ap-bangkok	曼谷一区 （曼谷节点用户覆盖亚太东南地区） ap-bangkok-1
	曼谷二区 （曼谷节点用户覆盖亚太东南地区） ap-bangkok-2
南美地区（圣保罗） sa-saopaulo	圣保罗一区（圣保罗节点可用于覆盖南美地区） sa-saopaulo-1
美国西部（硅谷） na-siliconvalley	硅谷一区（硅谷节点可用于覆盖美国西部） na-siliconvalley-1

	硅谷二区（硅谷节点可用于覆盖美国西部） na-siliconvalley-2
美国东部（弗吉尼亚） na-ashburn	弗吉尼亚一区 （弗吉尼亚节点用户覆盖美国东部地区） na-ashburn-1
	弗吉尼亚二区 （弗吉尼亚节点用户覆盖美国东部地区） na-ashburn-2
欧洲地区（法兰克福） eu-frankfurt	法兰克福一区（法兰克福节点可用于覆盖欧洲地区） eu-frankfurt-1
	法兰克福二区（法兰克福节点可用于覆盖欧洲地区） eu-frankfurt-2

如何选择地域和可用区

关于选择地域和可用区时，您需要考虑以下几个因素：

- 云服务器所在的地域、您以及您的目标用户所在的地理位置。
  建议您在购买云服务器时，选择最靠近您客户的地域，以降低访问时延、提高访问速度。
- 云服务器和其他云产品的关系。
  建议您在选择其他云产品时，尽量都在同个地域同个可用区，以便各云产品间可通过内网进行通信，降低访问时延、提高访问速度。
- 业务高可用和容灾考虑。
  即使在只有一个私有网络的场景下，建议您将业务至少部署在不同的可用区，以保证可用区间的故障隔离，实现跨可用区容灾。
- 不同可用区间可能会有网络的通信延迟，需要结合业务的实际需求进行评估，在高可用和低延迟之间找到最佳平衡点。
- 如果您需要访问其他国家和地区的主机，建议您选择其他国家和地区的云服务器进行访问。如果您在 中国 创建云服务器，访问 其他国家和地区的主机 会有较高的访问延迟，不建议您使用。

资源位置说明

这里说明腾讯云哪些资源是全球性的、哪些资源是区分地域不区分可用区的，以及哪些资源是基于可用区的。

资源	资源 ID 格式 <资源缩写>-8位数字及字符	类型	说明
账号	不限	全球唯一	用户可以使用同一个账号访问腾讯云全球各地资源。
SSH 密钥	skey-xxxxxxxx	全地域可用	用户可以使用 SSH 密钥绑定账号下任何地域的云服务器。
CVM 实例	ins-xxxxxxxx	只能在单地域的单个可用区下使用	用户只能在特定可用区下创建 CVM 实例。
自定义镜像	img-xxxxxxxx	单地域多可用区可用	用户可以创建实例的自定义镜像，并在同个地域的不同可用区下使用。需要在其他地域使用时请使用复制镜像功能将自定义镜像复制到其他地域下。
弹性 IP	eip-xxxxxxxx	单地域多可用区可用	弹性 IP 地址在某个地域下创建，并且只能与同一地域的实例相关联。
安全组	sg-xxxxxxxx	单地域多可用区可用	安全组在某个地域下创建，并且只能与同一地域的实例相关联。腾讯云为用户自动创建三条默认安全组。
云硬盘	disk-xxxxxxxx	只能在单地域的单个可用区下使用	用户只能在特定可用区下创建云硬盘，并且挂载在同一可用区的实例上。
快照	snap-xxxxxxxx	单地域多可用区可用	为某块云硬盘创建快照后，用户可在该地域下使用该快照进行其他操作（如创建云硬盘等）。
负载均衡	clb-xxxxxxxx	单地域多可用区可用	负载均衡可以绑定单地域下不同可用区的云服务器进行流量转发。
私有网络	vpc-xxxxxxxx	单地域多可用区可用	私有网络创建在某一地域下，可以在不同可用区下创建属于同一个私有网络的资源。

子网	subnet-xxxxxxxx	只能在单地域的单个可用区下使用	用户不能跨可用区创建子网。
路由表	rtb-xxxxxxxx	单地域多可用区可用	用户创建路由表时需要指定特定的私有网络，因此跟随私有网络的位置属性。

相关操作

将实例迁移到其他可用区

一个已经启动的实例是无法更改其可用区的，但是用户可以通过其他方法把实例迁移至其他可用区。迁移过程包括从原始实例创建自定义镜像、使用自定义镜像在新可用区中启动实例以及更新新实例的配置。

1. 创建当前实例的自定义镜像。更多信息，请参见 创建自定义镜像 。
2. 如果当前实例的 网络环境 为私有网络且需要在迁移后保留当前私有 IP 地址，用户可以先删除当前可用区中的子网，然后在新可用区中用与原始子网相同的 IP 地址范围创建子网。需要注意的是，不包含可用实例的子网才可以被删除。因此，应该将在当前子网中的所有实例移至新子网。
3. 使用刚创建的自定义镜像在新的可用区中创建一个新实例。用户可以选择与原始实例相同的实例类型及配置，也可以选择新的实例类型及配置。更多信息，请参见 创建实例 。
4. 如果原始实例已关联弹性 IP 地址，则将其与旧实例解除关联并与新实例相关联。更多信息，请参见 弹性 IP 。
5. （可选）若原有实例为 按量计费 类型，可选择销毁原始实例。更多信息，请参见 销毁实例 。若原有实例为 包年包月 类型，可选择等待其过期并回收。

将镜像复制到其他地域

用户启动实例、查看实例等动作都是区分地域属性的。若用户需要启动实例的镜像在本地域不存在，需要将镜像复制到本地域。更多信息，请参见 复制镜像 。

购买集群配额限制

最近更新时间：2025-06-13 09:23:42

容器服务的配额限制包括 TKE 配额限制，CVM 相关的配额限制以及托管集群的资源配额限制，详情如下：

TKE 配额限制

每个用户可购买的 TKE 配额默认如下，如果您需要更多的配额项数量，可通过 在线咨询 提出配额申请。

> **注意：**
> 2019年10月21日起，用户集群支持的最大节点配额统一调整为5000。

配额项	默认值	可查看入口	是否可提配额
单地域下集群	20	容器服务概览页右下方	是
单集群下节点	5000		
单地域下镜像命名空间	10		
单地域下镜像仓库	500		
单镜像下镜像版本	100		

CVM 配额限制

腾讯云容器服务所产生的云服务器需遵守云服务器的购买限制，详情请参见 云服务器购买约束。每个用户可购买的 CVM 配额默认如下，如果您需要更多的配额项数量，可通过 配额申请工单 提出配额申请。

配额项	默认值	可查看入口	是否可提配额
单可用区下包年包月服务器	150 台	CVM 概览页-各地域资源	是
单可用区下按量计费服务器	30 台或 60 台不等		

集群配置限制

> **说明：**
> 集群配置限制集群规模， 暂不支持修改。

配置项	地址范围	影响范围	可查看入口	是否可变更
VPC 网络-子网	用户自定义设置	该子网可添加的节点数	集群对应 VPC 子网列表页-可用 IP 数	• 不支持变更 • 可使用新子网
容器网段 CIDR	用户自定义设置	• 集群内节点上限 • 集群内 service 上限 • 每个节点 Pod 上限	集群基本信息页-容器网段	暂不支持变更

K8s 资源配额说明

> **说明：**
> 下列配额自2022年4月30日开始生效，该配额无法被移除。配额不足时，**您可通过升高集群规格提高各资源配额**。
> 如有特殊场景需要调整配额，可 提交工单 联系我们，并说明需要调整的原因。

如需检查此配额，您可以执行以下命令：

```
kubectl get resourcequota tke-default-quota -o yaml
```

如需查看给定命名空间的 `tke-default-quota` 对象，请添加 `--namespace` 选项以指定命名空间。

说明：
- 其他 K8s 资源限制指集群中除了 Pod，Node，ConfigMap 的其他 K8s 资源均不能超过该数值。 例如 L100 的集群，集群中的 ClusterRole，Service，Endpoint 等 K8s 资源的数量均不能超过 10000。
- CRD 数量限制指集群中所有 CRD 的总和不应该超过该限制。某类 CRD 的增多，会占用 CRD 配额，导致其他 CRD 能创建的数量变少。

集群规格	Pod 数量限制	ConfigMap 数量限制	CRD / 其他 K8s 资源数量限制
L5	600	256	1250
L20	1500	512	2500
L50	3000	1024	5000
L100	6000	2048	10000
L200	15000	4096	20000
L500	30000	6144	50000
L1000	90000	8192	100000
L3000	150000	10240	150000
L5000	200000	20480	200000

集群规格升配

1. 集群开启自动升配功能后，当集群控制面组件负载过高或者节点规模超过当前规格管理上限时，会自动升至更高规格。满足以下任意一项条件将触发集群自动升配（监控来源平于 metrics-server 获取的实时数据）：
   1.1 节点数（指 Kubernetes Node，包含节点类型有普通节点、原生节点）大于当前集群等级规格就触发升配。
   1.2 过去1天内，kube-apiserver OOM 重启次数大于2，或 kube-controller-manager/kube-scheduler OOM 重启次数大于3将触发升配。
   1.3 在过去2小时内，kube-apiserver OOM 重启次数大于1，且实时持续10分钟内负载大于70%将触发升配。
   1.4 kube-apiserver/kube-controller-manager/kube-scheduler 任意一组件连续10分钟以上，CPU 和内存实时负载持续95%将触发升配。
2. 升配过程中控制面（Master 节点）组件需要进行滚动更新，可能会有短暂中断，建议升配期间停止其他操作（如创建工作负载等）。
3. 集群升配后不会自动降配，您需手动下调集群规格。

针对命名空间分配配额

默认情况下，任意一个命名空间余量都相同（余量 = 当前集群等级的配额 - 整个集群已使用的量）。如果您在一个命名空间创建了资源，会导致余量减少，即其他命名空间的可使用量也会在一定时间后相应减少。
如果您有自定义分配比例的需求，可以在 `kube-system` 下创建一个 `tke-quota-config` 的 ConfigMap，指定**余量**在各个命名空间的分配比例。
例如：下面的例子代表在 `default` 的命名空间分配50%的余量，在 `kube-system` 的命令空间分配40%的余量，其余命名空间分配到10%的余量。如果设置的百分比之和超过100%，则 TKE 认为比例分配无效，会采用默认的分配策略。

```yaml
apiVersion: v1
data:
  default: "50"
  kube-system: "40"
kind: ConfigMap
metadata:
  name: tke-quota-config
  namespace: kube-system
```

容器服务节点公网 IP 说明

最近更新时间：2024-11-15 17:23:12

如果对业务安全有要求，不希望业务直接暴露到公网，同时又希望访问公网，您可以使用腾讯云 NAT 网关 。下文将介绍如何使用 NAT 网关来访问公网。

公网 IP

在默认的情况下，创建集群会为集群的节点分配公网 IP 。分配的公网 IP 将提供以下作用：

- 通过公网 IP 登录到集群的节点机器。
- 通过公网 IP 访问外网服务

NAT 网关

云服务器不绑定弹性公网 IP，所有访问 Internet 流量通过 NAT 网关转发。此种方案中，云服务器访问 Internet 的流量会通过内网转发至 NAT 网关，因而不会受云服务器购买时公网带宽的带宽上限限制，NAT 网关产生的网络流量费用也不会占用云服务器的公网带宽出口。通过 NAT 网关访问 Internet，您需要完成以下两个步骤：

步骤1：创建 NAT 网关

1. 登录 NAT 网关控制台 。
2. 选择地区和私有网络，单击**新建**。
3. 在 NAT 网关购买页中按需输入或确定相关参数，根据官网指引完成购买，具体详情请参见 创建与管理 NAT 网关 。

步骤2：配置相关子网所关联的路由表

> **说明：**
> 完成创建 NAT 网关后，您需要在私有网络控制台路由表页配置路由规则，以将子网流量指向 NAT 网关。

1. 单击左侧导航栏中的 路由表 。
2. 在路由表列表中，单击需要访问 Internet 的子网所关联的**路由表 ID/名称**，进入路由表详情页。
3. 在基本信息页面中，单击**新增路由策略**。
4. 在**新增路由**中，填写**目的端**，将**下一跳类型**选择为**公网 NAT 网关**，并将**下一跳**选择为步骤1创建的 **NAT 网关 ID**。

5. 单击**创建**。

完成以上配置后，关联此路由表的子网内的云服务器访问 Internet 的流量将指向 NAT 网关。

其他方案

方案1：使用弹性公网 IP

云服务器只绑定弹性公网 IP，不使用 NAT 网关。此种方案中，云服务器所有访问 Internet 流量通过弹性公网 IP 出去，会受到云服务器购买时公网带宽的带宽上限限制。访问公网产生的相关费用，根据云服务器网络计费模式而定。
使用方法：请参见 弹性公网 IP 操作指南 。

方案2：同时使用 NAT 网关和弹性公网 IP

云服务器同时使用 NAT 网关和弹性公网 IP。此种方案中，所有云服务器主动访问 Internet 的流量只通过内网转发至 NAT 网关，回包也经过 NAT 网关返回至云服务器。此部分流量不会受云服务器购买时公网带宽的带宽上限限制，NAT 网关产生的网络流量费用不会占用云服务器的公网带宽出口。如果来自 Internet 的流量主动访问云服务器的弹性公网 IP，则云服务器回包统一通过弹性公网 IP 返回，这样产生的公网出流量受到云服务器购买时公网带宽的带宽上限限制。访问公网产生的相关费用，根据云服务器网络计费模式而定。

> **注意：**
> 如果用户账号开通了带宽包共享带宽功能，则 NAT 网关产生的出流量按照带宽包整体结算（不再重复收取网络流量费），建议您限制 NAT 网关的出带宽，以避免因为 NAT 网关出带宽过高产生高额的带宽包费用。

容器服务安全组设置

最近更新时间：2023-10-13 17:58:51

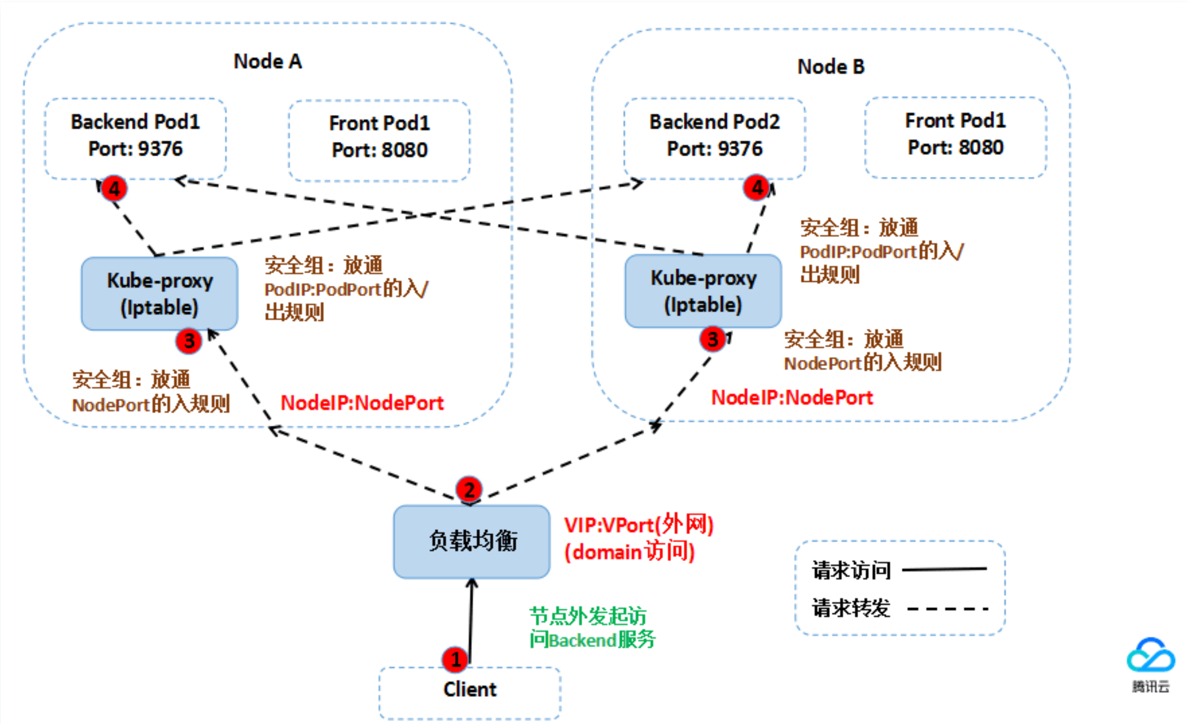
安全问题向来是一个大家非常关注的问题，腾讯云将安全性作为产品设计中的最高原则，严格要求产品做到安全隔离，容器服务同样非常看重这一点。腾讯云的基础网络可以提供充分的安全保障，容器服务选择了网络特性更丰富的 VPC 腾讯云私有网络 来作为容器服务的底层网络，本文档主要介绍容器服务下使用安全组的最佳实践，帮助大家选择安全组策略。

安全组

安全组是一种有状态的包过滤功能的虚拟防火墙，它用于设置单台或多台云服务器的网络访问控制，是腾讯云提供的重要的网络安全隔离手段。更多安全组的介绍请参见 安全组 。

容器服务选择安全组的原则

- 由于在容器集群中，服务实例采用分布式的方式进行部署，不同的服务实例分布在集群的节点上。建议同一个集群下的主机绑定同一个安全组，集群的安全组不添加其他云服务器。
- 安全组只对外开放最小权限。
- 需放通以下容器服务使用规则：
  - 放通容器实例网络和集群节点网络
    当服务访问到达主机节点后，会通过 Kube-proxy 模块设置的 iptable 规则将请求进行转发到服务的任意一个实例。由于服务的实例有可能在另外的节点上，这时会出现跨节点访问。例如访问的目的 IP有服务实例IP、集群中其它的节点 IP、节点上集群 cbr0 网桥的 IP。这就需要在对端节点上放通容器实例网络和集群节点网络访问。
  - 同一 VPC 不同集群互访的情况，需要放通对应集群的容器网络和节点网络。
  - 需要 SSH 登录节点的放通22端口。
  - 放通节点30000 - 32768端口。
    在访问路径中，需要通过负载均衡器将数据包转发到容器集群的 NodeIP:NodePort 上。其中 NodeIP 为集群中任意一节点的主机 IP ，而 NodePort 是在创建服务时容器集群为服务默认分配的，NodePort 的范围为30000 - 32768。
    下图以外网访问服务为例：



容器服务默认安全组规则

节点默认安全组规则

集群节点间的正常通信需要放通部分端口，为避免绑定无效安全组造成客户创建集群失败，容器服务为您提供了默认安全组配置规则。如下表：

⚠ 注意
若当前默认安全组不能满足业务需求，并且已创建绑定该安全组的集群时，您可参照 管理安全组规则 进行该集群安全组规则的查看、修改等操作。

入站规则

协议规则	端口	来源	策略	备注
ALL	ALL	容器网络 CIDR	允许	放通容器网络内 Pod 间通信
ALL	ALL	集群网络 CIDR	允许	放通集群网络内节点间通信
tcp	30000 - 32768	0.0.0.0/0	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）
udp	30000 - 32768	0.0.0.0/0	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）
icmp	-	0.0.0.0/0	允许	放通 ICMP 协议，支持 Ping 操作

出站规则

协议规则	端口	来源	策略
ALL	ALL	0.0.0.0/0	允许

ⓘ 说明
- 自定义出站规则时需放通节点网段和容器网段。
- 容器节点配置该规则，可满足不同的访问方式访问集群中服务。
- 集群中服务的访问方式，可参考 Service 管理 服务访问方式 。

独立集群 Master 默认安全组规则

创建独立集群时，会默认为 Master 机型绑定 TKE 默认安全组，降低集群创建后 Master 与 Node 无法正常通信及 Service 无法正常访问的风险。默认安全组配置规则如下表：

ⓘ 说明
创建安全组的权限继承至 TKE 服务角色，详情请参见 服务授权相关角色权限说明 。

入站规则

协议	端口	网段	策略	备注
ICMP	ALL	0.0.0.0/0	允许	支持 Ping 操作
TCP	30000 - 32768	集群网络 CIDR	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）
UDP	30000 - 32768	集群网络 CIDR	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）
TCP	60001,60002,10250,2380,2379,53,17443,50055,443,61678	集群网络 CIDR	允许	放通 API Server 通信
TCP	60001,60002,10250,2380,2379,53,17443	容器网络 CIDR	允许	放通 API Server 通信
TCP	30000 - 32768	容器网络 CIDR	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）
UDP	30000 - 32768	容器网络 CIDR	允许	放通 NodePort 访问（LoadBalancer 类型的 Service 需经过 NodePort 转发）

UDP	53	容器网络 CIDR	允许	放通 CoreDNS 通信
UDP	53	集群网络 CIDR	允许	放通 CoreDNS 通信

出站规则

协议规则	端口	来源	策略
ALL	ALL	0.0.0.0/0	允许

集群新增资源所属项目说明

最近更新时间：2023-09-04 14:39:01

总述

如需要通过分项目进行财务核算等，请先阅读以下内容：

1. 集群无项目属性，集群内云服务器、负载均衡器等资源有项目属性。
2. 集群新增资源所属项目：仅将新增到该集群下的资源归属到该项目下。

建议

1. 建议集群内的所有资源在同一个项目。
2. 如若需要集群内云服务器分布在不同的项目，请自行前往云服务器控制台迁移项目。操作详情见 调整项目配置 。

容器服务高危操作

最近更新时间：2024-07-22 17:43:41

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

集群

分类	高危操作	导致后果	误操作解决方案
master 及 etcd 节点	修改集群内节点安全组	可能导致 master 节点无法使用	按照官网推荐配置放通 安全组
	节点到期或被销毁	该 master 节点不可用	不可恢复
	重装操作系统	master 组件被删除	不可恢复
	自行升级 master 或者 etcd 组件版本	可能导致集群无法使用	回退到原始版本
	删除或格式化节点 /etc/kubernetes 等核心目录数据	该 master 节点不可用	不可恢复
	更改节点 IP	该 master 节点不可用	改回原 IP
	自行修改核心组件（etcd、kube-apiserver、docker 等）参数	可能导致 master 节点不可用	按照官网推荐配置参数
	自行更换 master 或 etcd 证书	可能导致集群不可用	不可恢复
worker 节点	修改集群内节点安全组	可能导致节点无法使用	按照官网推荐配置放通安全组
	调整节点实例规格	机器强制关机，节点不可用	节点移出再加入集群
	节点到期或被销毁	该节点不可用	不可恢复
	重装操作系统	节点组件被删除	节点移出再加入集群
	自行升级节点组件版本	可能导致节点无法使用	回退到原始版本
	更改节点 IP	节点不可用	节点移出再加入集群
	自行修改核心组件（etcd、kube-apiserver、docker 等）参数	可能导致节点不可用	按照官网推荐配置参数
	修改操作系统配置	可能导致节点不可用	尝试还原配置项或删除节点重新购买
其他	在 CAM 中执行权限变更或修改的操作	集群部分资源如负载均衡可能无法创建成功	恢复权限

网络与负载均衡

高危操作	导致后果	误操作解决方案
修改内核参数 net.ipv4.ip_forward=0	网络不通	修改内核参数为 net.ipv4.ip_forward=1
修改内核参数 net.ipv4.tcp_tw_recycle = 1	导致 nat 异常	修改内核参数 net.ipv4.tcp_tw_recycle = 0
节点安全组配置未放通容器 CIDR 的53端口 udp	集群内 DNS 无法正常工作	按照官网推荐配置放通安全组
修改或者删除 TKE 添加的 LB 的标签	购买新的 LB	恢复 LB 的标签
通过 LB 的控制台在 TKE 管理的 LB 创建自定义的监听器	所做修改被 TKE 侧重置	通过 service 的 yaml 来自动创建监听器
通过 LB 的控制台在 TKE 管理的 LB 绑定自定义的后端 rs		禁止手动绑定后端 rs

通过 LB 的控制台修改 TKE 管理的 LB 的证书		通过 ingress 的 yaml 来自动管理证书
通过 LB 的控制台修改 TKE 管理的 LB 监听器名称		禁止修改 TKE 管理的 LB 监听器名称

日志

高危操作	导致后果	误操作解决方案	备注
删除宿主机 /tmp/ccs-log-collector/pos 目录	日志重复采集	无	Pos 里面的文件记录了文件的采集位置
删除宿主机 /tmp/ccs-log-collector/buffer 目录	日志丢失	无	Buffer 里面是待消费的日志缓存文件

云硬盘

高危操作	导致后果	误操作解决方案
控制台手动解挂 CBS	Pod 写入报 io error	删掉 node上mount 目录，重新调度 Pod
节点上 umount 磁盘挂载路径	Pod 写入本地磁盘	重新 mount 对应目录到 Pod 中
节点上直接操作 CBS 块设备	Pod 写入本地磁盘	无

云上容器应用部署 Check List

最近更新时间：2024-11-26 11:28:42

简介

业务上云安全高效、稳定高可用是每一位涉云从业者的共同诉求。这一诉求实现的前提，离不开系统可用性、数据可靠性及运维稳定性三者的紧密配合。本文将从评估项目、影响说明及评估参考三个角度为您介绍云上容器应用部署的各个检查项，以便帮助您扫除上云障碍、顺利高效地将业务迁移到容器服务 TKE。

检查项

系统可用性

类别	评估项目	类型	影响说明	评估参考
集群	创建集群前，结合业务场景提前规划节点网络和容器网络，避免后续业务扩容受限。	网络规划	集群所在子网或容器网段较小，将可能导致集群实际支持的可用节点数少于业务所需容量。	• 网络规划 • 容器及节点网络设置
	创建集群前，提前梳理专线接入、对等连接、容器网段和子网网段等相关网段的规划，避免之后出现网段冲突，影响业务。	网络规划	简单组网场景按照页面提示配置集群相关网段，避免冲突；业务复杂组网场景，例如对等连接、专线接入、VPN 等，网络规划不当将影响整体业务正常互访。	VPC 连接
	创建集群时，会自动新建并绑定默认安全组，支持根据业务需求设置自定义安全组规则。	部署	安全组是重要的安全隔离手段，不当的安全策略配置可能会引起安全相关的隐患及服务连通性等问题。	容器服务安全组设置
	Containerd 和 Docker 作为 TKE 当前支持的运行时组件，有不同的适用场景。创建集群时，请根据业务场景选择合适的容器运行时（Container Runtime）组件。	部署	集群创建后，如修改运行时组件及版本，只对集群内无节点池归属的增量节点生效，不会影响存量节点。	如何选择 Containerd 和 Docker
	默认情况下，Kube-proxy 使用 iptables 来实现 Service 到 Pod 之间的负载均衡。创建集群时，支持快速开启 IPVS 来承接流量并实现负载均衡。	部署	当前支持在创建集群时开启 IPVS，之后对全集群生效且将不可关闭。	集群启用 IPVS
	创建集群时，根据业务场景选择合适的集群模式：独立集群、托管集群。	部署	托管集群的 Master 和 Etcd 不属于用户资源，由腾讯云技术团队集中管理和维护，用户无法修改 Master 和 Etcd 的部署规模和服务参数。如需修改，请选用独立部署模式集群。	• 集群概述 • 集群的托管模式说明
工作负载	创建工作负载时需设置 CPU 和内存的限制范围，提高业务的健壮性。	部署	同一个节点上部署多个应用，当未设置资源上下限的应用出现应用异常资源泄露问题时，将会导致其它应用分配不到资源而异常，且其监控信息将会出现误差。	设置工作负载的资源限制
	创建工作负载时可设置容器健康检查：“容器存活检查”和“容器就绪检查”。	可靠性	容器健康检查未配置，会导致用户业务出现异常时 Pod 无法感知，从而导致不会自动重启恢复业务，最终将会出现 Pod 状态正常，但 Pod 中的业务异常的现象。	服务健康检查设置
	创建服务时需要根据实际访问需求选择合适的访问方式，目前支持以下四种：提供公网访问、仅在集群内访问、VPC 内网访问及主机端口访问。	部署	选择不当的访问方式，可能造成服务内外部访问逻辑混乱和资源浪费。	Service 管理
	工作负载创建时，避免单 Pod 副本数设置，请根据自身业务合理设置节点调度策略。	可靠性	如设置单 Pod 副本数，当节点异常或实例异常会导致服务异常。为确保您的 Pod 能够调度成功，请确保您在设置调度规则后，节点有空余的资源用于容器的调度。	• 调整 Pod 数量 • 设置工作负载的调度规则

数据可靠性

类别	评估项目	类型	影响说明	评估参考
容器数据持久化	应用 Pod 数据存储，根据实际需求选择合适的数据卷类型。	可靠性	节点异常无法恢复时，存在本地磁盘中的数据无法恢复，而云存储此时可以提供极高的数据可靠性。	Volume 管理

运维稳定性

类别	评估项目	类型	影响说明	评估参考
工程	CVM、VPC、子网及 CBS 等资源配额是否满足客户需求。	部署	配额不足将会导致创建资源失败，对于配置了自动扩容的用户尤其需要保障所使用的云服务配额充足。	• 购买集群配额限制 • 配额限制
	集群的节点上不建议用户随意修改内核参数、系统配置、集群核心组件版本、安全组及 LB 相关参数等。	部署	可能会导致 TKE 集群功能异常或安装在节点上的 Kubernetes 组件异常，节点状态变成不可用，无法部署应用到此节点。	容器服务高危操作
主动运维	容器服务提供多维度的监控和告警功能，同时结合腾讯云可观测平台 TCOP 提供的基础资源监控，能保证更细的指标覆盖。配置监控告警，以便于异常时及时收到告警和故障定位。	监控	未配置监控告警，将无法建立容器集群性能的正常标准，在出现异常时无法及时收到告警，需要人工巡检环境。	• 查看监控数据 • 监控及告警指标列表

开源组件

最近更新时间：2024-02-29 15:09:21

tencentcloud-cloud-controller-manager

tencentcloud-cloud-controller-manager 是腾讯云容器服务的 Cloud Controller Manager 的实现。使用该组件，可以在通过腾讯云云服务器自建的 Kubernetes 集群上实现以下功能：

- nodecontroller：更新 Kubernetes node 相关的 addresses 信息。
- routecontroller：负责创建私有网络内 pod 网段内的路由。
- servicecontroller：当集群中创建了类型为负载均衡的 service 的时候，创建相应的负载均衡。

更多安装使用说明，可查看 GitHub tencentcloud-cloud-controller-manager 。

kubernetes-csi-tencentcloud

kubernetes-csi-tencentcloud 是腾讯云云硬盘服务的一个符合 CSI 标准的实现插件。使用该组件，可以在通过腾讯云云服务器自建的 Kubernetes 集群中使用云硬盘。

该插件适用于自建 Kubernetes 集群时使用云硬盘的场景，与容器服务集群自带的 `provisioner cloud.tencent.com/qcloud-cbs` 不同。

更多安装使用说明，可查看 GitHub kubernetes-csi-tencentcloud 。

权限管理
概述

最近更新时间：2023-05-17 15:40:18

如果您在腾讯云中使用到了容器服务（Tencent Kubernetes Engine，TKE），且该服务虽然由不同的人管理，但都统一使用您的云账号密钥，将存在以下问题：

- 您的密钥由多人共享，泄密风险高。
- 您无法限制其他人的访问权限，其他人误操作易造成安全风险。

为解决以上问题，您可以通过使用子账号来实现不同的人管理不同的业务。默认情况下，子账号没有使用 TKE 的权限，我们需要创建策略来允许子账号拥有他们所需要的权限。

简介

访问管理（Cloud Access Management，CAM）是腾讯云提供的一套 Web 服务，它主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。通过 CAM，您可以创建、管理和销毁用户（组），并通过身份管理和策略管理控制哪些人可以使用哪些腾讯云资源。

当您使用 CAM 的时候，可以将策略与一个用户或者一组用户关联起来，策略能够授权或者拒绝用户使用指定资源完成指定任务。有关 CAM 策略的更多相关基本信息，请参照 策略语法。有关 CAM 策略的更多相关使用信息，请参照 策略。

如果您不需要对子账户进行 CAM 相关资源的访问管理，您可以跳过此章节。跳过这些部分并不影响您对文档中其余部分的理解和使用。

入门

CAM 策略必须授权使用一个或多个 TKE 操作或者必须拒绝使用一个或多个 TKE 操作。同时还必须指定可以用于操作的资源（可以是全部资源，某些操作也可以是部分资源），策略还可以包含操作资源的条件。

TKE 部分 API 操作不支持资源级权限，意味着对于该类 API 操作，您不能在使用该类操作的时候指定某个具体的资源来使用，而必须要指定全部资源来使用。

服务授权相关角色权限说明

最近更新时间：2024-12-06 14:19:42

在使用腾讯云容器服务（Tencent Kubernetes Engines，TKE）的过程中，为了能够使用相关云资源，会遇到多种需要进行服务授权的场景。每种场景通常对应不同的角色所包含的预设策略，其中主要涉及到 `TKE_QCSRole` 和 `IPAMDofTKE_QCSRole` 两个角色。本文档接下来将分角色展示各个授权策略的详情、授权场景及授权步骤。

> **说明：**
> 本文档示例角色均不包含容器镜像仓库相关授权策略，容器镜像服务权限详情请参见 TKE 镜像仓库资源级权限设置。

TKE_QCSRole 角色

开通容器服务后，腾讯云会授予您的账户 `TKE_QCSRole` 角色的权限。该容器服务角色默认关联多个预设策略，为获取相关权限，需在特定的授权场景下执行对应的预设策略授权操作。操作完成之后，对应策略会出现在该角色的已授权策略列表中。`TKE_QCSRole` 角色关联的预设策略包含如下：

默认关联预设策略

- `QcloudAccessForTKERole`：容器服务对云资源的访问权限。
- `QcloudAccessForTKERoleInOpsManagement`：日志服务等运维管理。

其他关联预设策略

- `QcloudAccessForTKERoleInCreatingCFSStorageclass`：容器服务操作文件存储（CFS）权限，包含增删查文件存储文件系统、查询文件系统挂载点等。
- `QcloudCVMFinanceAccess`：云服务器财务权限。

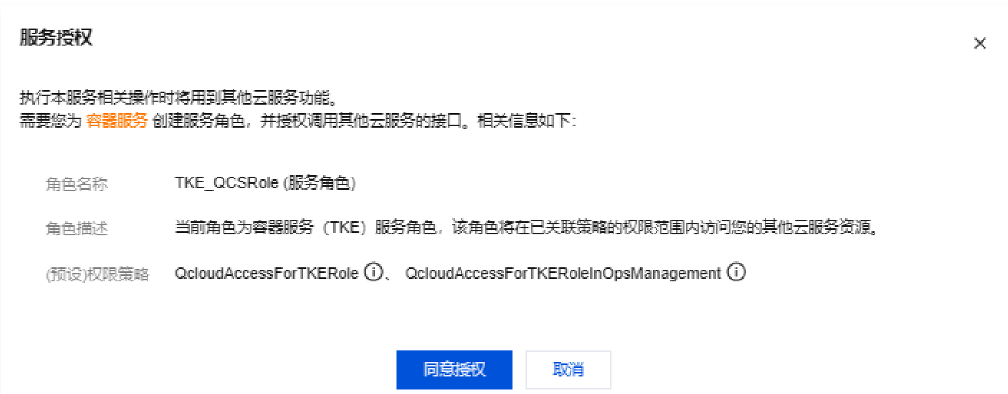
预设策略 QcloudAccessForTKERole

授权场景

当您已注册并登录腾讯云账号后，首次登录 容器服务控制台 时，需前往**访问管理**页面对当前账号授予腾讯云容器服务操作云服务器（CVM）、负载均衡（CLB）、云硬盘（CBS）等云资源的权限。

授权步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**，弹出**服务授权**窗口。
2. 单击**同意授权**，完成身份验证后即可成功授权。如下图所示：



权限内容

可前往访问管理平台查看预设策略 QcloudAccessForTKERole 授权访问周边云服务的权限内容。

预设策略 QcloudAccessForTKERoleInOpsManagement

授权场景

该策略默认关联 `TKE_QCSRole` 角色，开通容器服务并完成 `TKE_QCSRole` 角色授权后，即可获得包含日志在内的各种运维相关功能的权限。

授权步骤

该策略与 预设策略 QcloudAccessForTKERole 同时授权，无需额外操作。

权限内容

可前往访问管理平台查看预设策略 QcloudAccessForTKERoleInOpsManagement 授权访问 CLS 日志服务的权限内容。

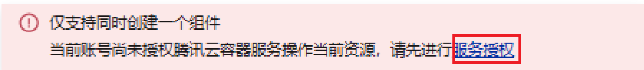
预设策略 QcloudAccessForTKERoleInCreatingCFSStorageclass

授权场景

使用腾讯云文件存储（CFS）扩展组件，能够帮助您在容器集群中使用文件存储。首次使用该插件时，需通过容器服务进行文件存储中文件系统等相关资源的授权操作。

授权步骤

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，选择地域及集群后，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在组件管理页面单击**新建**。
4. 在新建组件管理中，当扩展组件首次选择为 “CFS（腾讯云文件存储）” 时，单击页面下方的**服务授权**。如下图所示：



5. 在服务授权中，单击**同意授权**并完成身份验证即可成功授权。

权限内容

可前往访问管理平台查看预设策略 QcloudAccessForTKERoleInCreatingCFSStorageclass 授权访问 CFS 文件存储服务的权限内容。

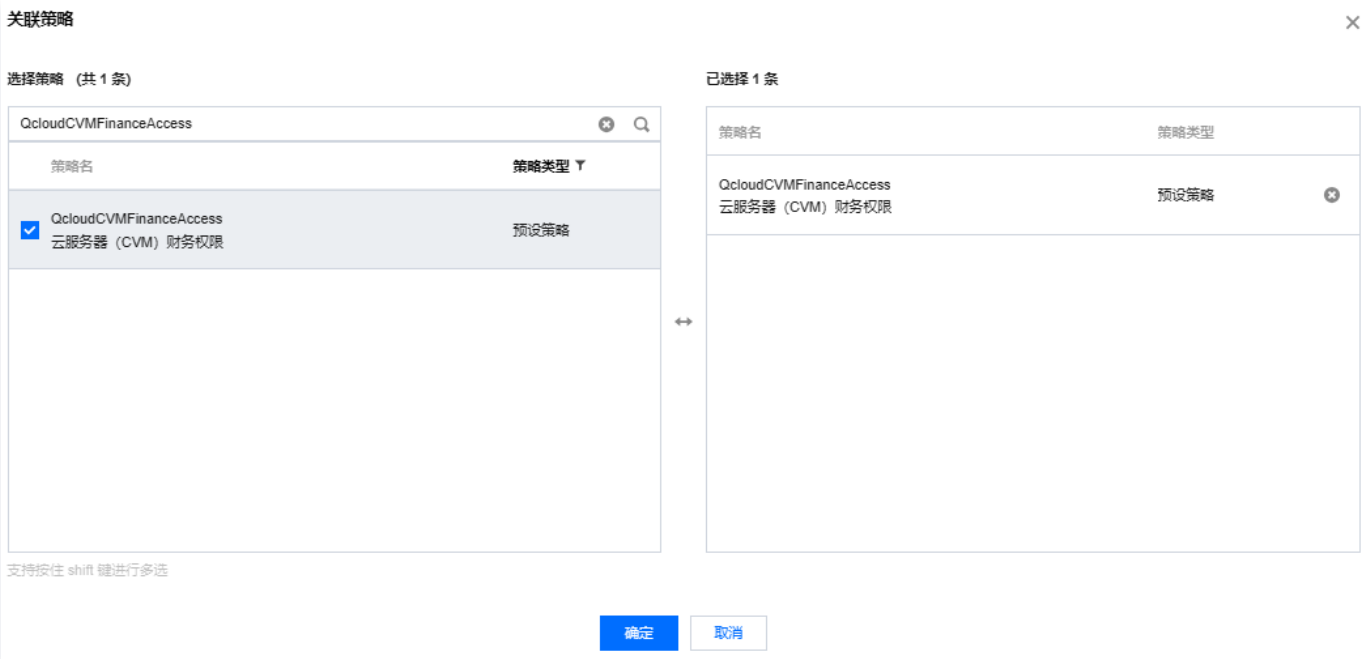
预设策略 QcloudCVMFinanceAccess

授权场景

当您需要购买包年包月云硬盘时，需要为角色 `TKE_QCSRole` 添加该策略以配置支付权限，否则可能会因为没有支付权限导致创建基于包年包月 storageclass 的 PVC 失败。

授权步骤

1. 登录访问管理控制台，选择左侧导航栏的 角色 。
2. 在**角色**列表页面中，单击 TKE_QCSRole 进入该角色管理页面。
3. 选择 TKE_QCSRole 页面中的**关联策略**，并在弹出的**风险提醒**窗口中进行确认。
4. 在关联策略中，找到 `QcloudCVMFinanceAccess` 策略并勾选。如下图所示：



5. 单击**确定**即可完成授权。

权限内容

可前往访问管理平台查看预设策略 QcloudCVMFinanceAccess 授权访问 云服务器财务权限的权限内容。

IPAMDofTKE_QCSRole 角色

`IPAMDofTKE_QCSRole` 角色为容器服务的 IPAMD 支持服务角色。被授予该角色的权限后，在本文描述的授权场景下需进行预设策略关联操作。完成操作后，以下策略会出现在该角色的已授权策略列表中：
`QcloudAccessForIPAMDofTKERole`：容器服务 IPAMD 支持（TKE IPAMD）对云资源的访问权限。

预设策略 QcloudAccessForIPAMDofTKERole

授权场景

在首次使用 VPC-CNI 网络模式创建集群时，需要首先对容器服务 IPAMD 支持（TKE IPAMD）对云资源的访问权限进行授权，以便能够正常使用 VPC-CNI 网络模式。

授权步骤

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 单击集群列表上方的新建。
3. 在创建集群页面，找到**网络配置** > **容器网络插件**，选择 **VPC-CNI** 时，单击**服务授权**。如下图所示：

4. 在服务授权中，单击**同意授权**并完成身份验证即可成功授权。

权限内容

可前往访问管理平台查看预设策略 QcloudAccessForIPAMDofTKERole 授权访问周边云服务的权限内容。

TCR 镜像仓库资源级权限设置

最近更新时间：2023-12-19 10:53:31

TKE 内置支持使用容器镜像服务 TCR 内镜像，并同时支持个人版及企业版。在使用镜像仓库时，可以按照命名空间、仓库颗粒度配置访问权限，方便企业灵活、精细管理权限。

具体设置请参考：

- 企业版：TCR 企业版权限管理指南
- 个人版：TCR 个人版权限管理指南

TKE 集群级权限控制
使用 TKE 预设策略授权

最近更新时间：2025-05-26 17:00:01

本文介绍腾讯云容器服务 TKE 的预设策略，及如何将子账号关联预设策略，授予子账号特定权限。您可参考文本并根据实际业务诉求进行配置。

TKE 预设策略

您可以使用以下预设策略为您的子账号授予相关权限：

策略	描述
QcloudTKEFullAccess	TKE 全读写访问权限，包括 TKE 及相关云服务器、负载均衡、私有网络、监控及用户组权限。
QcloudTKEInnerFullAccess	TKE 全部访问权限， TKE 涉及较多产品，建议您配置 QcloudTKEFullAccess 权限。
QcloudTKEReadOnlyAccess	TKE 只读访问权限。

以下预设策略是在您使用 TKE 服务时，授予 TKE 服务本身的权限。不建议为子账号关联以下预设策略：

策略	描述
QcloudAccessForIPAMDofTKERole	授予 TKE 服务弹性网卡相关权限。
QcloudAccessForIPAMDRoleInQcloudAllocateEIP	授予 TKE 服务弹性公网 IP 相关权限。
QcloudAccessForTKERole	授予 TKE 服务云服务器、标签、负载均衡、日志服务相关权限。
QcloudAccessForTKERoleInCreatingCFSStorageclass	授予 TKE 服务文件存储相关权限。
QcloudAccessForTKERoleInOpsManagement	该策略关联 TKE 服务角色（TKE_QCSRole），用于 TKE 访问其他云服务资源，包含日志服务等相关操作权限。

子账号关联预设策略

您可在创建子账号的“设置用户权限”步骤中，通过 直接关联 或 随组关联 方式，为该子账户关联预设策略。

直接关联

您可以直接为子账号关联策略以获取策略包含的权限。

1. 登录访问管理控制台，选择左侧导航栏中的**用户** > 用户列表 。
2. 在“用户列表”管理页面，选择需要设置权限的子账号所在行右侧的**授权**。
3. 在弹出的“关联策略”窗口中，勾选需授权的策略。
4. 单击**确定**即可。

随组关联

您可以将子账号添加至用户组，该子账号将自动获取该用户组所关联策略的权限。如需解除随组关联策略，仅需将子账号移出相应用户组即可。

1. 登录访问管理控制台，选择左侧导航栏中的**用户** > 用户列表 。
2. 在“用户列表”管理页面，选择需要设置权限的子账号所在行右侧的**更多操作** > **添加到组**。
3. 在弹出的“添加到组”窗口中，勾选需加入的用户组。
4. 单击**确定**即可。

登录子账号验证

登录 腾讯云容器服务控制台，验证可使用所授权策略对应功能，则表示子账号授权成功。

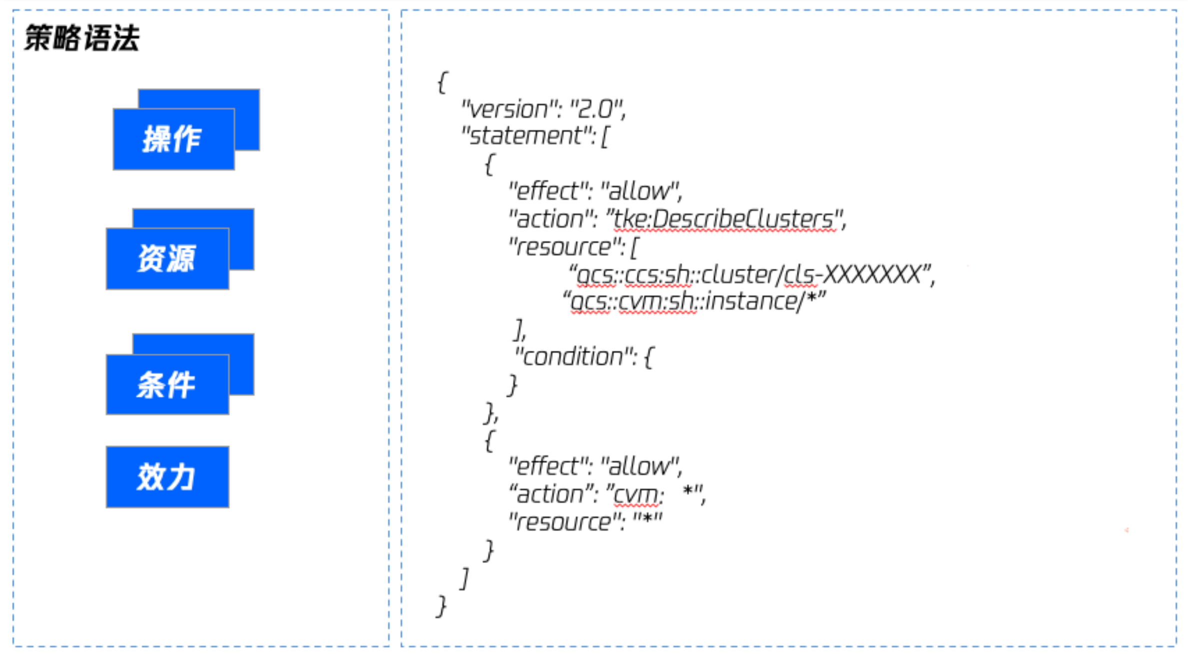
使用自定义策略授权

最近更新时间：2024-08-29 16:14:01

本文介绍如何自定义配置腾讯云容器服务 TKE 的自定义策略，授予子账号特定权限。您可参考文本并根据实际业务需求进行配置。

策略语法说明

策略语法结构如下图所示：



- **action**：表示接口。
- **resource**：表示资源。

> **说明：**
> 您可自行编写策略语法，或通过访问管理 CAM 策略生成器创建自定义策略。可结合以下示例进行自定义策略配置：
> - 配置子账号对单个 TKE 集群的管理权限
> - 通过标签为子账号配置批量集群的全读写权限

TKE 接口权限配置

本节提供了集群、节点模块的多个功能所包含的子功能、对应云 API 接口、间接调用接口、权限控制资源级别以及 Action 字段展示相关信息。

集群模块

功能接口对照表如下：

功能	包含子功能	对应云 API 接口	间接调用接口	权限控制资源级别	Action 字段
创建空集群	• Kubernetes 版本选择 • 运行时组件选择 • 选择 VPC 网络 • 设置容器网络 • 自定义镜像选择 • Ipvs 设置	tke:CreateCluster	cam:GetRole account:DescribeUserData account:DescribeWhiteList tag:GetTagKeys vpc:DescribeVpcEx cvm:DescribeImages	• 创建集群是接口级别权限控制 • 获取 VPC 列表，需要 VPC 的资源权限	"tke:CreateCluster", "cam:GetRole", "tag:GetTagKeys", "vpc:DescribeVpcEx", "cvm:DescribeImages"

使用已有 CVM 创建托管集群	• 创建空集群包含功能 • 将已有 CVM 作为 Node • 挂载安全组 • 挂载数据盘 • 开启自动调节		cvm:DescribeInstances vpc:DescribeSubnetEx cvm:DescribeSecurityGroups vpc:DescribeVpcEx cvm:DescribeImages cvm:ResetInstance cvm:DescribeKeyPairs	• 创建集群是接口级别权限控制 • 获取 CVM 列表，需要 CVM 的资源权限	"tke:CreateCluster", "cvm:DescribeInstances", "vpc:DescribeSubnetEx", "cvm:DescribeSecurityGroups", "vpc:DescribeVpcEx", "cvm:DescribeImages", "cvm:ResetInstance", "cvm:DescribeKeyPairs"
使用已有 CVM 创建独立集群	• 创建空集群包含功能 • 将已有 CVM 作为 Node • 将已有 CVM 作为 Master&ETCD • 挂载安全组 • 挂载数据盘 • 开启自动调节		cvm:DescribeInstances vpc:DescribeSubnetEx cvm:DescribeSecurityGroups vpc:DescribeVpcEx cvm:DescribeImages cvm:ResetInstance cvm:DescribeKeyPairs	• 创建集群是接口级别权限控制 • 获取 VPC 列表，需要 VPC 的资源权限 • 获取 CVM 列表，需要 CVM 的资源权限	"tke:CreateCluster", "cvm:DescribeInstances", "vpc:DescribeSubnetEx", "cvm:DescribeSecurityGroups", "vpc:DescribeVpcEx", "cvm:DescribeImages", "cvm:ResetInstance", "cvm:DescribeKeyPairs"
自动新建 CVM 创建托管集群	• 创建空集群包含功能 • 购买 CVM 作为 node • 挂载安全组 • 挂载数据盘 • 开启自动调节		cvm:DescribeSecurityGroups cvm:DescribeKeyPairs cvm:RunInstances vpc:DescribeSubnetEx vpc:DescribeVpcEx cvm:DescribeImages	• 创建集群是接口级别权限控制 • 获取 VPC 列表，需要 VPC 的资源权限	"cvm:DescribeSecurityGroups", "cvm:DescribeKeyPairs", "cvm:RunInstances", "vpc:DescribeSubnetEx", "vpc:DescribeVpcEx", "cvm:DescribeImages", "tke:CreateCluster"
自动新建 CVM 创建独立集群	• 创建空集群包含功能 • 购买 CVM 作为 Node • 购买 CVM 作为 Master&ETCD • 挂载安全组 • 挂载数据盘 • 开启自动调节		cvm:DescribeSecurityGroups cvm:DescribeKeyPairs cvm:RunInstancesvpc:DescribeSubnetEx vpc:DescribeVpcEx cvm:DescribeImages	• 创建集群是接口级别权限控制 • 获取 VPC 列表，需要 VPC 的资源权限	"cvm:DescribeSecurityGroups", "cvm:DescribeKeyPairs", "cvm:RunInstances", "vpc:DescribeSubnetEx", "vpc:DescribeVpcEx", "cvm:DescribeImages", "tke:CreateCluster"

查询集群列表	-	tke:DescribeClusters	-	获取集群列表，需要集群的资源权限	"tke:DescribeClusters"
显示集群凭证	-	tke:DescribeClusterSecurity	-	显示集群凭证，需要集群的资源权限	"tke:DescribeClusterSecurity"
开启/关闭集群内/外网访问地址	• 创建托管集群外网访问端口 • 创建集群访问端口 • 修改托管集群外网端口的安全策略 • 查询托管集群开启外网端口流程状态 • 删除托管集群外网访问端口 • 删除集群访问端口	tke:CreateClusterEndpointVip tke:CreateClusterEndpoint tke:ModifyClusterEndpointSP tke:DescribeClusterEndpointVipStatus tke:DescribeClusterEndpointStatus tke:DeleteClusterEndpointVip tke:DeleteClusterEndpoint	-	开启关闭集群访问，需要集群资源的权限	-
删除集群	-	tke:DeleteCluster	tke:DescribeClusterInstances tke:DescribeInstancesVersion tke:DescribeClusterStatus	删除集群，需要集群的资源权限	"tke:DescribeClusterInstances", "tke:DescribeInstancesVersion", "tke:DescribeClusterStatus", "tke:DeleteCluster"

节点模块

功能接口对照表如下：

功能	包含子功能	对应云 API 接口	间接调用接口	权限控制资源级别	Action 字段
添加已有节点	• 将已有节点加入到集群 • 重新设置数据盘 • 设置安全组	tke:AddExistedInstances	cvm:DescribeInstances vpc:DescribeSubnetEx cvm:DescribeSecurityGroups vpc:DescribeVpcEx cvm:DescribeImages cvm:ResetInstance cvm:DescribeKeyPairs cvm:ModifyInstancesAttribute tke:DescribeClusters	• 添加已有节点、需要对应集群的资源权限 • 获取 CVM 列表，需要 CVM 的资源权限	"cvm:DescribeInstances", "vpc:DescribeSubnetEx", "cvm:DescribeSecurityGroups", "vpc:DescribeVpcEx", "cvm:DescribeImages", "cvm:ResetInstance", "cvm:DescribeKeyPairs", "tke:DescribeClusters", "tke:AddExistedInstances"
新建节点	• 新建节点加入到集群 • 重新设置数据盘 • 设置安全组	tke:CreateClusterInstances	cvm:DescribeSecurityGroups cvm:DescribeKeyPairs cvm:RunInstances	新建节点、需要对应集群的资源权限	"cvm:DescribeSecurityGroups", "cvm:DescribeKeyPairs", "cvm:RunInstances", "vpc:DescribeSu

			vpc:DescribeSubnetEx vpc:DescribeVpcEx cvm:DescribeImages tke:DescribeClusters		bnetEx", "vpc:DescribeVpcEx", "cvm:DescribeImages", "tke:DescribeClusters"
节点列表	查看集群节点列表	tke:DescribeClusterInstances	cvm:DescribeInstances tke:DescribeClusters	• 查看节点列表需要对应集群的资源权限 • 获取 CVM 列表，需要 CVM 的资源权限	"cvm:DescribeInstances", "tke:DescribeClusters", "tke:DescribeClusterInstances"
移出节点	-	tke:DeleteClusterInstances	cvm:TerminateInstances tke:DescribeClusters	• 查看节点列表需要对应集群的资源权限 • 获取 CVM 列表，需要 CVM 的资源权限 • 删除节点，需要对应节点的销毁策略	"cvm:TerminateInstances", "tke:DescribeClusters", "tke:DeleteClusterInstances"

使用示例
通过标签为子账号配置批量集群的全读写权限

最近更新时间：2023-05-17 15:40:19

操作场景

您可以通过使用访问管理（Cloud Access Management，CAM）策略让用户拥有在容器服务（Tencent Kubernetes Engine，TKE）控制台中查看和使用特定资源的权限。本文档中的示例介绍如何通过控制台，为子账号授予指定标签集群的权限。

操作步骤

1. 在访问管理控制台的 策略 页面，单击左上角的**新建自定义策略**。
2. 在弹出的选择创建方式窗口中，单击**按标签授权**，进入按标签授权页面。
3. 在“可视化策略生成器”中添加服务与操作栏，补充以下信息，编辑一个授权声明。
    - 服务（必选）：选择容器服务 (tke)。
    - 操作（必选）：选择您要授权的操作。
4. 在选择标签栏，选择需要授权的标签信息，可添加多个标签。授权完成的子账号将对具有该标签键及标签值的资源拥有全读写权限。
5. 单击**下一步**，进入关联用户/用户组/角色页面。在关联用户/用户组/角色页面补充策略名称和描述信息。策略名称由控制台自动生成，默认为 "policygen"，后缀数字根据创建日期生成。您可进行自定义。
6. 对关联用户/用户组/角色快速授权。授权完成的子账号将对具有该标签键及标签值的资源拥有全读写权限。
    - **将此权限授权给用户**：按需勾选需授权的子账号。
    - **将此权限授权给用户组**：按需勾选需授权的子账号所在的用户组。
    - **将此权限授权给角色**：按需勾选需授权的子账号所在的角色。
7. 单击**完成**。

配置子账号对单个 TKE 集群的管理权限

最近更新时间：2023-09-08 19:13:02

操作场景

您可以通过使用访问管理（Cloud Access Management，CAM）策略让用户拥有在容器服务（Tencent Kubernetes Engine，TKE）控制台中查看和使用特定资源的权限。本文档中的示例指导您在控制台中配置单个集群的策略。

操作步骤

配置对单个集群全读写权限

1. 登录 CAM 控制台。
2. 在左侧导航栏中，单击 策略，进入策略管理页面。
3. 单击**新建自定义策略**，选择 “按策略语法创建” 方式。
4. 选择 “空白模板” 类型，单击**下一步**。
5. 自定义策略名称，将 “编辑策略内容” 替换为以下内容。

```
{
 "version": "2.0",
 "statement": [
     {
         "action": [
             "tke:*"
         ],
         "resource": [
             "qcs::tke:sh::cluster/cls-XXXXXXX",
             "qcs::cvm:sh::instance/*"
         ],
         "effect": "allow"
     },
     {
         "action": [
             "cvm:*"
         ],
         "resource": "*",
         "effect": "allow"
     },
     {
         "action": [
             "vpc:*"
         ],
         "resource": "*",
         "effect": "allow"
     },
     {
         "action": [
             "clb:*"
         ],
         "resource": "*",
         "effect": "allow"
     },
     {
         "action": [
             "monitor:*",
             "cam:ListUsersForGroup",
             "cam:ListGroups",
             "cam:GetGroup",
```

```
            "cam:GetRole"
        ],
        "resource": "*",
        "effect": "allow"
    }
 ]
}
```

6. 在 “编辑策略内容” 中，将 `qcs::tke:sh::cluster/cls-XXXXXXX` 修改为您想赋予权限的指定地域下的集群。
   例如，您需要为广州地域的 cls-69z7ek9l 集群赋予全读写的权限，将 `qcs::tke:sh::cluster/cls-XXXXXXX` 修改为 `"qcs::tke:gz::cluster/cls-69z7ek9l"`。

> **注意**
> 请替换成您想赋予权限的指定地域下的集群 ID。如果您需要允许子账号进行集群的扩缩容，还需要配置子账号用户支付权限。

7. 单击**创建策略**，即可完成对单个集群全读写权限的配置。

配置对单个集群只读权限

1. 登录 CAM 控制台 。
2. 在左侧导航栏中，单击 策略 ，进入策略管理页面。
3. 单击**新建自定义策略**，选择 “按策略语法创建” 方式。
4. 选择 “空白模板” 类型，单击**下一步**。
5. 自定义策略名称，将 “编辑策略内容” 替换为以下内容。

```
{
 "version": "2.0",
 "statement": [
    {
        "action": [
            "tke:Describe*",
            "tke:Check*"
        ],
        "resource": "qcs::tke:gz::cluster/cls-1xxxxxx",
        "effect": "allow"
    },
    {
        "action": [
            "cvm:Describe*",
            "cvm:Inquiry*"
        ],
        "resource": "*",
        "effect": "allow"
    },
    {
        "action": [
            "vpc:Describe*",
            "vpc:Inquiry*",
            "vpc:Get*"
        ],
        "resource": "*",
        "effect": "allow"
    },
    {
        "action": [
            "clb:Describe*"
        ],
        "resource": "*",
        "effect": "allow"
```

```
        },
        {
            "effect": "allow",
            "action": [
                "monitor:*",
                "cam:ListUsersForGroup",
                "cam:ListGroups",
                "cam:GetGroup",
                "cam:GetRole"
            ],
            "resource": "*"
        }
    ]
}
```

6. 在 “编辑策略内容” 中，将 `qcs::tke:gz::cluster/cls-1xxxxxx` 修改为您想赋予权限的指定地域下的集群。例如，您需要为北京地域的 cls-19a7dz9c 集群赋予只读的权限，将 `qcs::tke:gz::cluster/cls-1xxxxxx` 修改为 `qcs::tke:bj::cluster/cls-19a7dz9c` 。
7. 单击**创建策略**，即可完成对单个集群只读权限的配置。

配置子账号对 TKE 服务全读写或只读权限

最近更新时间：2024-10-25 17:53:41

操作场景

您可以通过使用访问管理（Cloud Access Management，CAM）策略让用户拥有在容器服务（Tencent Kubernetes Engine，TKE）控制台中查看和使用特定资源的权限。本文档中的示例指导您在控制台中配置部分权限的策略。

操作步骤

配置全读写权限

1. 登录访问管理控制台，选择左侧导航栏中的 策略 。
2. 在**策略**页面，选择 QcloudTKEFullAccess 策略行的**关联用户/组/角色**。如下图所示：

3. 在“关联用户/用户组/角色”弹窗中，勾选需对 TKE 服务拥有全读写权限的账号，单击**确定**，即可完成子账号对 TKE 服务全读写权限的配置。
4. 在**策略**页面，单击 QcloudCCRFullAccess 策略行的**关联用户/用户组/角色**。
5. 在“关联用户/用户组/角色”弹窗中，勾选需对镜像仓库拥有全读写权限的账号，并单击**确定**，即可完成子账号对镜像仓库全读写权限的配置。

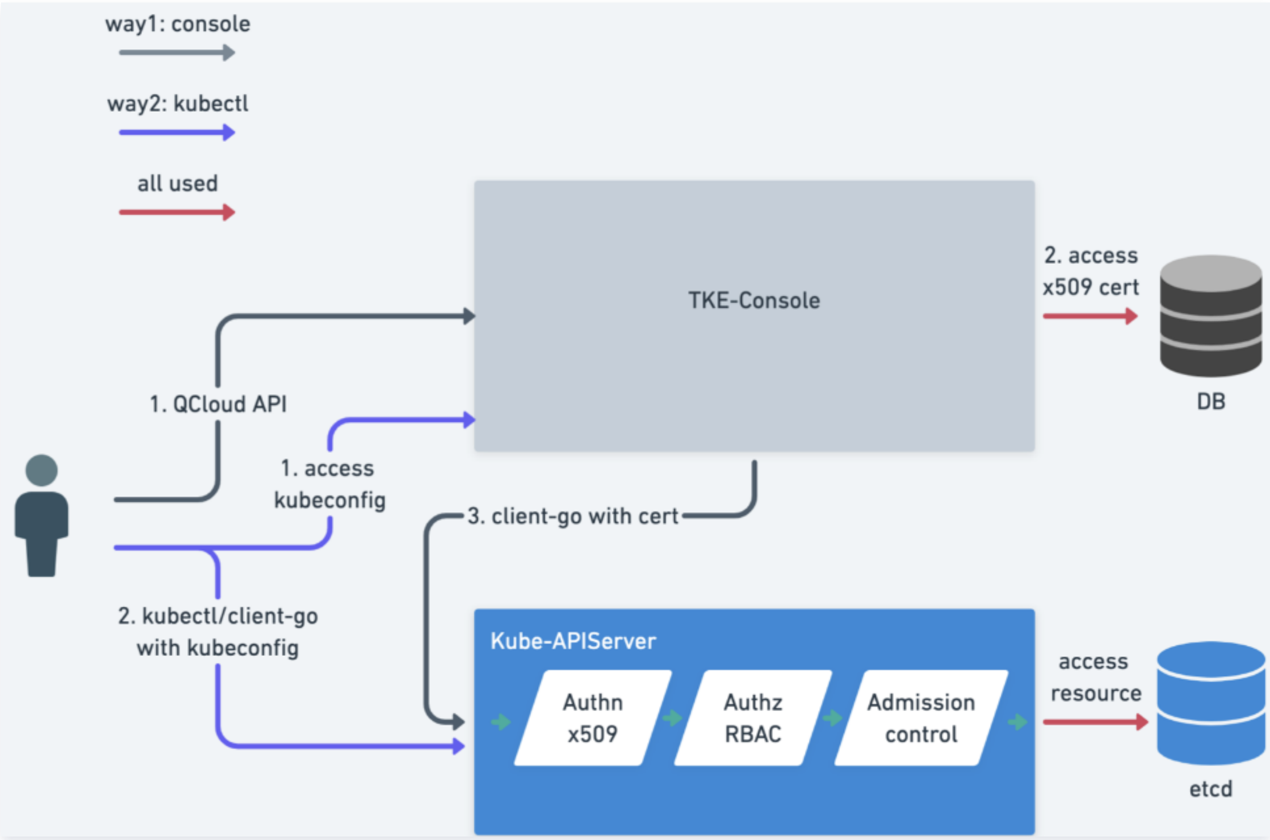
配置只读权限

1. 登录访问管理控制台，选择左侧导航栏中的 策略 。
2. 在**策略**页面，选择 QcloudTKEReadOnlyAccess 策略行的**关联用户/用户组/角色**。
3. 在“关联用户/用户组/角色”弹窗中，勾选需对 TKE 服务拥有只读权限的账号，并单击**确定**，即可完成子账号对 TKE 服务只读权限的配置。
4. 在**策略**页面，单击 QcloudCCRReadOnlyAccess 策略行的**关联用户/用户组/角色**。
5. 在“关联用户/用户组/角色”弹窗中，勾选需对镜像仓库拥有只读权限的账号，并单击**确定**，即可完成子账号对镜像仓库只读权限的配置。

TKE Kubernetes 对象级权限控制
概述

最近更新时间：2023-08-29 16:34:01

TKE 提供了对接 Kubernetes RBAC 的授权模式，便于对子账号进行细粒度的访问权限控制。该授权模式下，可通过容器服务控制台及 kubectl 两种方式进行集群内资源访问。如下图所示：



名词解释

RBAC（Role-Based Access Control）

基于角色的权限控制。通过角色关联用户、角色关联权限的方式间接赋予用户权限。
在 Kubernetes 中，RBAC 是通过 `rbac.authorization.k8s.io` API Group 实现的，即允许集群管理员通过 Kubernetes API 动态配置策略。

Role

用于定义某个命名空间的角色的权限。

ClusterRole

用于定义整个集群的角色的权限。

RoleBinding

将角色中定义的权限赋予一个或者一组用户，针对命名空间执行授权。

ClusterRoleBinding

将角色中定义的权限赋予一个或者一组用户，针对集群范围内的命名空间执行授权。
如需了解更多信息，请前往 Kubernetes 官方说明。

TKE Kubernetes 对象级别权限控制方案

认证方式

Kubernetes APIServer 支持丰富多样的认证策略，例如 x509 证书、bearer token、basic auth。其中，仅 bearer token 单个认证策略支持指定 known-token csv 文件的 bearer token、serviceaccount token、OIDC token、webhook token server 等多种 token 认证方式。
TKE 分析了实现复杂性及多种场景等因素，选择使用 x509 证书认证方式。其优势如下：

- 用户理解成本低。
- 对于存量集群无需进行复杂变更。
- 按照 User 及 Group 进行划分，后续扩展性好。

TKE 基于 x509 证书认证实现了以下功能：

- 每个子账号单独具备客户端证书，用于访问 Kubernetes APIServer。
- 当子账号在控制台访问 Kubernetes 资源时，后台默认使用该子账号的客户端证书去访问用户 Kubernetes APIServer。
- 支持子账号更新独有的客户端证书，防止凭证泄露。
- 支持主账号或使用集群 `tke:admin` 权限的账号进行查看、更新其他子账号的证书。

授权方式

Kubernetes 包含 RBAC 及 Webhook Server 两种主流授权模式。为给熟悉 Kubernetes 的用户提供一致性体验，并且需要与原生 Kubernetes 结合使用，TKE 选择使用 RBAC 模式。该模式提供了预设 Role 及 ClusterRole，用户只需要在集群内创建相应的 RoleBinding 和 ClusterRoleBinding 即可实现授权变更。其优势如下：

- 亲和有 Kubernetes 基础的用户。
- 复用 Kubernetes RBAC 能力，支持 Namespace 维度、APIGroup 维度及资源维度的多种 Verb 权限控制。
- 支持用户自定义策略。
- 支持管理用户自定义的扩展 API 资源。

TKE Kubernetes 对象级别权限控制功能

通过 TKE 提供的授权管理功能，您可以进行更细粒度的权限控制。例如，仅赋予某个子账号只读权限或仅赋予某个子账号下的某个命名空间读写权限等。可参考以下文档，对子账号进行更细粒度的权限控制：

- 使用预设身份授权
- 自定义策略授权

授权模式对比

最近更新时间：2024-09-11 18:07:31

腾讯云容器服务 TKE 目前存在新旧两种授权模式，旧的授权模式无法进行 Kubernetes 级别的授权管理，建议您升级集群管理的授权模式，以便能够对集群内 Kubernetes 资源进行细粒度的权限控制。

新旧模式对比

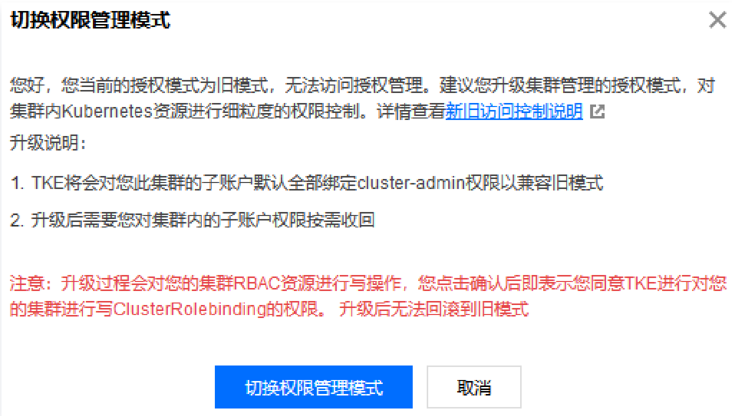
对比项	旧模式	新模式
Kubeconfig	admin token	子账号独立的 x509 证书
控制台访问集群资源	无细粒度权限，子账号具备全读写权限	对接 Kubernetes RBAC 资源控制

存量集群授权模式升级操作

升级授权模式

若使用旧授权模式的集群需要升级时，请参考以下操作步骤进行升级：

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在集群列表中，选择需升级的集群 ID。
3. 在集群详情页面中，选择左侧**授权管理** > **ClusterRole**。
4. 在 **ClusterRole** 中，单击 **RBAC 策略生成器**。
5. 在弹出的“切换权限管理模式”窗口中，单击**切换权限管理模式**即可进行授权模式升级。如下图所示：



为确保新旧模式的兼容性，升级过程中会进行如下操作：

5.1 创建默认预设管理员 ClusterRole： `tke:admin` 。

5.2 拉取子账号列表。

5.3 为每个子账号生成可用于 Kubernetes APIServer 认证的 x509 客户端证书。

5.4 为每个子账号都绑定 `tke:admin` 角色（确保和存量功能兼容）。

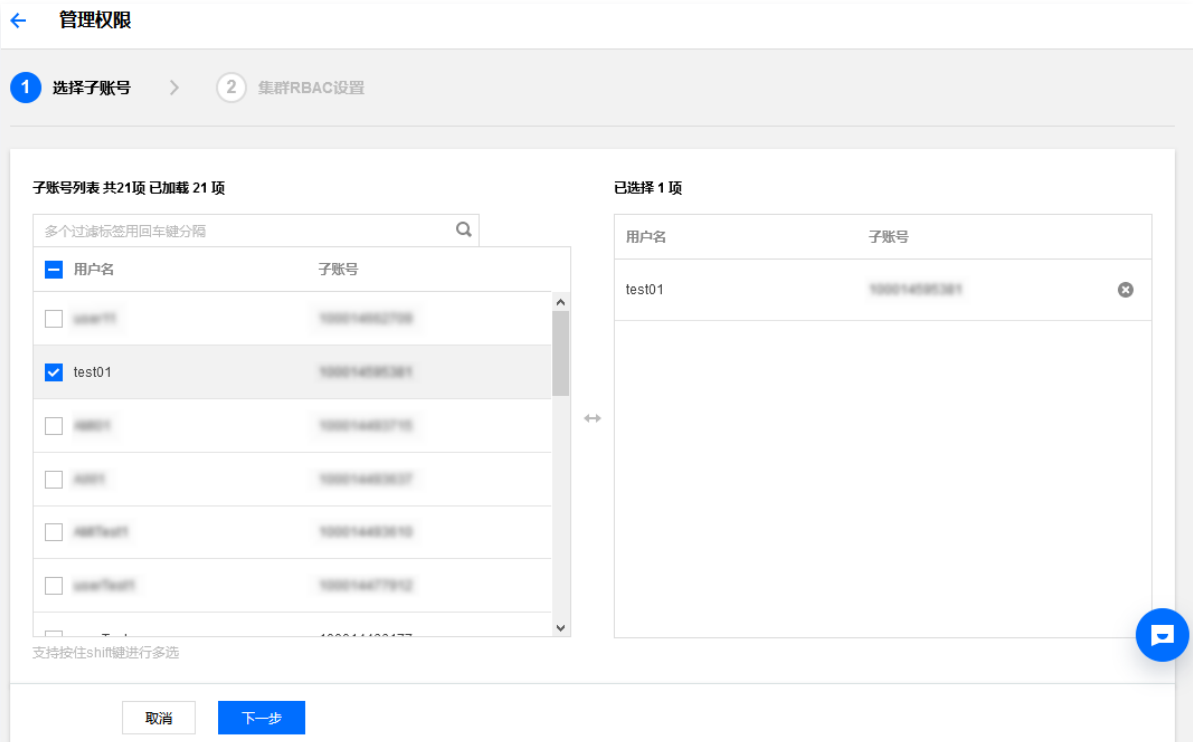
5.5 升级完毕。

回收子账号权限

集群授权模式升级完毕后，集群管理员（通常为主账号管理员或创建集群的运维人员）可按需对具有该集群权限的子账号进行权限回收操作，步骤如下：

1. 选择集群**授权管理**下的菜单项，在对应的管理页面中单击 **RBAC 策略生成器**。

2. 在管理权限页面的“选择子账号”步骤中，勾选需回收权限的子账号并单击**下一步**。如下图所示：



3. 在“集群 RBAC”步骤中，设置权限。例如，“权限设置”选择为命名空间“default”下的“只读用户”。如下图所示：

4. 单击**完成**即可完成回收操作。

确认子账号权限

当完成子账号回收操作后，您可通过以下步骤进行确认：

1. 选择左侧的**授权管理** > **ClusterRoleBinding**，进入 ClusterRoleBinding 管理页面。
2. 选择被回收权限的子账号名称，进入 YAML 文档页面。
   子账号默认为 `tke:admin` 权限，回收对应权限后，可在 YAML 文件中查看变更。如下图所示：

YAML


```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  annotations:
    cloud.tencent.com/tke-account-nickname: bxg
  creationTimestamp: "2020-07-08T12:59:05Z"
  labels:
    cloud.tencent.com/tke-account: "[redacted]"
  name: [redacted]-ClusterRole
  resourceVersion: "5838559579"
  selfLink: /apis/rbac.authorization.k8s.io/v1beta1/clusterrolebindings/[redacted]-ClusterRole
  uid: d43ef4ac-d68a-4e01-[redacted]
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tke:ro
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: [redacted]-1594205611
```


新授权模式相关问题

在新授权模式下创建的集群，谁具备管理员 admin 权限?

集群的创建者及主账号始终具备 `tke:admin` ClusterRole 的权限。

当前使用账号是否可控制自身权限?

目前不支持通过控制台操作当前使用账号权限，如需进行相关操作，可通过 kubectl 完成。

是否可以直接操作 ClusterRoleBinding 及 ClusterRole?

请勿直接对 ClusterRoleBinding 及 ClusterRole 进行修改或删除等操作。

客户端证书是如何创建的?

当您使用子账号通过控制台访问集群资源时，TKE 会获取该子账号的客户端证书。若未获取到证书，则会为该子账号创建客户端证书。

在访问管理 CAM 中删除了子账号，相关权限会自动回收吗?

不会自动回收，您可进入控制台“权限管理”页面，单击右上角“清理失效账户”进行权限回收。

如何授权其他账户“授权管理”的权限?

可使用默认管理员角色 `tke:admin` 进行“授权管理”的授权操作。

使用预设身份授权

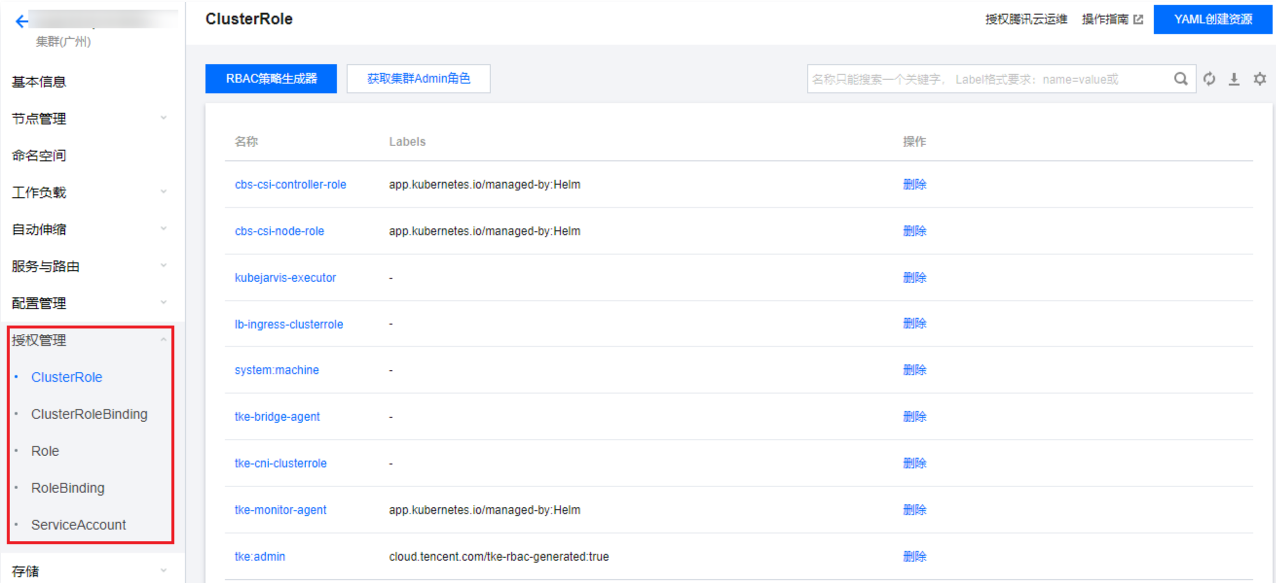
最近更新时间：2024-09-02 17:34:41

预设角色说明

腾讯云容器服务控制台通过 Kubernetes 原生的 RBAC 授权策略，针对子账号提供了细粒度的 Kubernetes 资源权限控制。同时提供了预设角色： Role 及 ClusterRole，详细说明如下：

Role 说明

容器服务控制台提供授权管理页，默认**主账号**及**集群创建者**具备管理员权限。可对其他拥有该集群 DescribeCluster Action 权限的子账号进行权限管理。如下图所示：



ClusterRole 说明

- **所有命名空间维度**：
  - **管理员（tke:admin）**：对所有命名空间下资源的读写权限，具备集群节点、存储卷、命名空间、配额的读写权限，可配置子账号的读写权限。
  - **运维人员（tke:ops）**：对所有命名空间下控制台可见资源的读写权限，具备集群节点、存储卷、命名空间、配额的读写权限。
  - **开发人员（tke:dev）**：对所有命名空间下控制台可见资源的读写权限。
  - **只读用户（tke:ro）**：对所有命名空间下控制台可见资源的只读权限。
  - **自定义**：用户自定义 ClusterRole。
- **指定命名空间维度**：
  - **开发人员（tke:ns:dev）**： 对所选命名空间下控制台可见资源的读写权限， 需要选择指定命名空间。
  - **只读用户（tke:ns:ro）**：对所选命名空间下控制台可见资源的只读权限， 需要选择指定命名空间。
- 所有预设的 ClusterRole 都将带有固定 label： `cloud.tencent.com/tke-rbac-generated: "true"` 。
- 所有预设的 ClusterRoleBinding 都带有固定的 annotations： `cloud.tencent.com/tke-account-nickname: yournickname` 及 label： `cloud.tencent.com/tke-account: "yourUIN"` 。

操作步骤

获取凭证

容器服务默认会为每个子账号创建独立的凭证，用户只需访问集群详情页或调用云 API 接口 DescribeClusterKubeconfig，即可获取当前使用账号的凭证信息 Kubeconfig 文件。

通过控制台获取步骤如下：

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。

3. 在**集群 APIServer 信息**模块中查看并下载 Kubeconfig 文件。如下图所示：

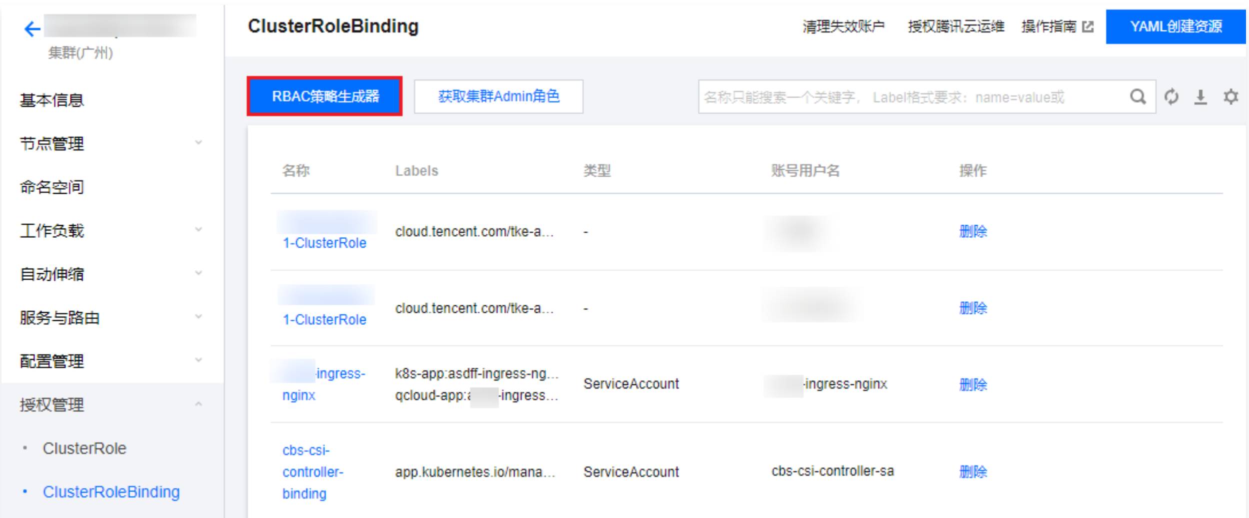
凭证管理

集群管理员可以访问凭证管理页，进行查看并更新所有账号下集群的凭证。详情请参见 更新子账号的 TKE 集群访问凭证 。

授权

> **说明：**
> 请联系集群管理员（主账号、集群创建者或拥有 admin role 的用户）进行授权。

1. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
2. 选择左侧**授权管理** > **ClusterRoleBinding**。
3. 在 **ClusterRoleBinding** 页面，单击 **RBAC 策略生成器**。如下图所示：



4. 在**新建 ClusterRoleBinding** 页面，勾选需授权的子账号，单击**下一步**。
5. 在**集群 RBAC 设置**步骤中，按照以下指引进行权限设置：
    - **Namespace 列表**：按需指定权限生效的 Namespace 范围。
    - **权限**：请参考界面中的“权限说明”，按需设置权限。

> **说明：**
> 您还可以单击**添加权限**，继续进行权限自定义设置。

鉴权

登录子账号，确认该账号已获得所授权限，则表示授权成功。

自定义策略授权

最近更新时间：2024-11-18 19:08:42

本文介绍如何通过自行编写 Kubernetes 的 ClusterRole 和 Role 以授予子账号特定权限，您可根据业务诉求进行对应操作。

策略语法说明

您可自行编写策略语法，或通过访问管理 CAM 策略生成器创建自定义策略。YAML 示例如下：

Role：命名空间维度

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: test-role
  namespace: default
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
```

ClusterRole：集群维度

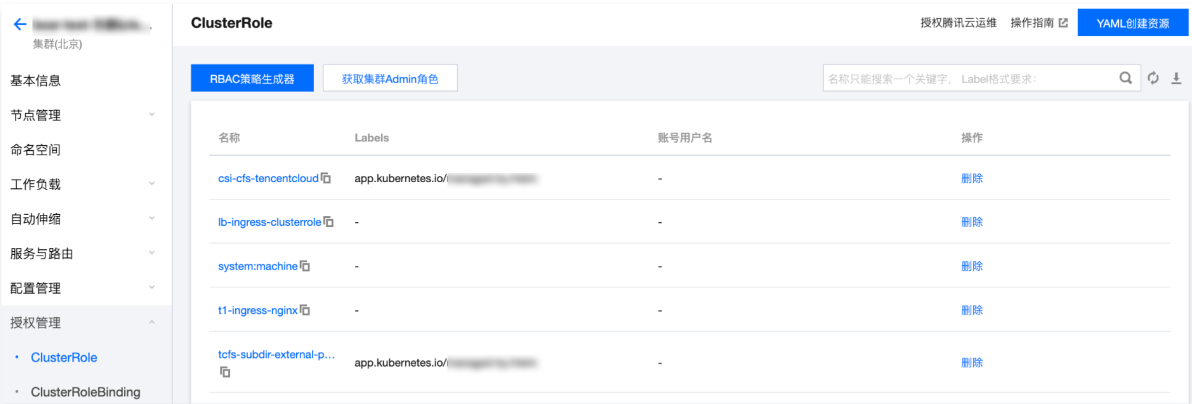
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: test-clusterrole
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
```

操作步骤

> **说明：**
> 该步骤以子账号绑定自定义 ClusterRole 为例，与绑定 Role 的步骤基本一致，您可结合实际需求进行操作。

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。

3. 选择左侧菜单栏中的**授权管理** > **ClusterRole**。如下图所示：



4. 在 **ClusterRole** 页面，单击右上角的 **YAML 创建资源**。
5. 在编辑界面输入自定义策略的 YAML 内容，单击**完成**即可创建 ClusterRole。
   该步骤以 ClusterRole：集群维度 YAML 为例，创建完成后，可在 **ClusterRole** 页面中查看自定义权限 “test-clusterrole”。
6. 在 **ClusterRoleBinding** 页面，单击 **RBAC 策略生成器**。
7. 在**新建 ClusterRoleBinding** 页面勾选需授权的子账号并单击**下一步**。
8. 进入**集群 RBAC** 页面，按照以下指引进行权限设置。如下图所示：

   - **Namespace 列表**：按需指定权限生效的 Namespace 范围。
   - **权限**：选择 “自定义” ，并单击**选择自定义权限**。按需在自定义权限列表中进行权限选择，本文以选择已创建的自定义权限 “test-clusterrole” 为例。

   > **说明**
   > 您还可以单击**添加权限**，继续进行权限自定义设置。

9. 单击**完成**即可完成授权操作。

参考资料

如需了解更多信息，可参考 Kubernetes 官方文档：使用 RBAC 授权 。

更新子账号的 TKE 集群访问凭证

最近更新时间：2024-09-02 17:34:41

访问凭证功能

腾讯云容器服务 TKE 基于 x509 证书认证实现了以下功能：

- 每个子账号均单独具备客户端证书，用于访问 Kubernetes APIServer。
- 在 TKE 新授权模式下，不同子账号在获取集群访问凭证时，即访问集群基本信息页面或调用云 API 接口 DescribeClusterKubeconfig 时，将会获取到子账号独有的 x509 客户端证书，该证书是使用每个集群的自签名 CA 进行签发的。
- 当子账号在控制台访问 Kubernetes 资源时，后台默认使用该子账号的客户端证书去访问用户 Kubernetes APIServer。
- 支持子账号更新独有的客户端证书，防止凭证泄露。
- 支持主账号或使用集群 `tke:admin` 权限的账号进行查看、更新其他子账号的证书。

操作步骤

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在集群列表中，选择目标集群 ID。
3. 在集群详情页面中，选择左侧的**基本信息**，在**集群 APIServer 信息**模块中单击 **Kubeconfig 权限管理**。
4. 在 **Kubeconfig 权限管理**中，按需勾选认证账号并单击**更新**即可。

集群管理
集群概述

最近更新时间：2025-04-30 16:05:12

集群基本信息

集群是指容器运行所需云资源的集合，包含若干台云服务器、负载均衡器等腾讯云资源。您可以在集群中运行您的应用程序。

集群架构

TKE 采用兼容标准的 Kubernetes 集群，包含以下组件：
- Master：用于管控集群的管理面节点。
- Etcd：用于保存集群的状态和元数据信息。
- Node：业务运行的工作节点。

集群类型

TKE 容器集群支持下述类型：

集群类型	描述
托管集群	Master、Etcd 腾讯云容器服务管理
独立集群（已停止新建）	Master、Etcd 采用用户自有主机搭建

集群类型详情可参见 集群模式说明。

集群生命周期

关于 TKE 集群的生命周期，请参见 集群生命周期。

集群相关操作

- 创建集群
- 更改集群操作系统
- 删除集群
- 集群扩缩容
- 连接集群
- 升级集群
- 集群启用 IPVS
- 选择容器网络模式
- 自定义 Kubernetes 组件启动参数
- 使用 KMS 进行 Kubernetes 数据加密

集群的托管模式说明

最近更新时间：2025-04-30 16:05:12

Master 托管模式

简介

腾讯云容器服务 TKE 提供 Master、Etcd 全部托管的 Kubernetes 集群管理服务。
该模式下，Kubernetes 集群的 Master 和 Etcd 会由腾讯云技术团队集中管理和维护。您只需要购置集群，运行负载所需的工作节点即可，不需要关心集群的管理和维护。

Master 托管模式注意事项

- 针对不同规格的托管集群，会收取相应的集群管理费用，以及用户实际使用的云资源（云服务器、持久化存储、负载均衡等）费用。关于收费模式和具体价格，请参阅 容器服务计费概述 。
- Master、Etcd 不属于用户资源，您在该模式下无法自主修改 Master 和 Etcd 的部署规模和服务参数。如果您有修改的需求，请使用 Master 独立部署模式 。
- 该模式下，即使您删除集群的全部工作节点，集群仍会不断尝试运行您未删除的工作负载和服务，导致在此过程中可能会产生费用。如果您决定终止集群服务和费用产生，请直接删除该集群。

Master 独立部署模式（已停止新建）

简介

腾讯云 TKE 也为您提供集群完全自主可控的 Master 独立部署模式。
选择该模式，Kubernetes 集群的 Master 和 Etcd 将会部署在您购置的 CVM 上。您拥有 Kubernetes 集群的所有管理和操作权限。

Master 独立部署模式注意事项

- 该模式仅适用于 Kubernetes 1.10.x 以上版本。
- 该模式下，Kubernetes 集群的 Master 和 Etcd 需要您额外购置资源部署。
- 如果您的集群规模较大，推荐选择高配机型。机型选择请参考：

集群规模	建议 Master 节点配置	建议节点数量
约100个节点	8核16GB SSD 系统盘	3台以上
约500个节点	16核32GB SSD 系统盘	3台以上
1000个节点以上	咨询我们	3台以上

购置限制说明

为了保证集群和服务的高可用性和提高集群性能，在独立部署模式下，设置以下限制：
- Master&Etcd 节点要求至少部署3台。
- Master&Etcd 节点需配置4核及以上的机型。
- Master&Etcd 节点选择 SSD 盘作为系统盘。

注意事项

为了保证集群的稳定性，以及发生异常后的恢复效率，建议如下：
在 Master 独立部署模式下：
- 请不要删除 Master 节点下支撑 Kubernetes 运行的核心组件。
- 请不要修改 Master 核心组件的配置参数。
- 请不要修改/删除集群内部的核心资源。
- 请不要修改/删除 Master 节点的相关证书文件（拓展名为 .crt，.key）。

非必要情况下：
- 请不要修改任何节点的 docker 版本。
- 请不要修改任何节点操作系统的 kernel、nfs-utils 等相关组件。

说明

- 核心组件：kube-APIserver，kube-scheduler，kube-controller-manager，tke-tools，systemd，cluster-contrainer-agent。
- 核心组件配置参数：kube-APIserver 参数，kube-scheduler 参数，kube-controller-manager 参数。
- 集群内部核心资源（包括但不限于）：hpa endpoint，master service account，kube-dns，auto-scaler，master cluster role，master cluster role binding。

如果您对以上建议有疑问，请 联系我们 。

集群生命周期

最近更新时间：2023-06-12 16:08:21

集群生命周期状态说明

状态	说明
创建中	集群正在创建，正在申请云资源。
规模调整中	集群的节点数量变更，添加节点或销毁节点中。
运行中	集群正常运行。
升级中	升级集群中。
删除中	集群在删除中。
异常	集群中存在异常，如节点网络不可达等。
隔离中	因为欠费超过24小时导致托管集群进入隔离中状态，停止扣除集群管理费用。

注意

容器服务基于 Kubernetes 且为声明式服务。如果您已在容器服务中创建负载均衡（CLB）、云硬盘（CBS）等 IaaS 资源，现在不再需要使用 CLB 和 CBS，请在 容器服务控制台 中删除对应的 Service 和 PersistentVolumeClaim 对象。如果您只在 CLB 控制台中删除 CLB 或者在 CBS 控制台中删除 CBS，容器服务会重新创建新的 CLB 和 CBS， 并继续扣除相关费用。

创建集群（新版）

最近更新时间：2025-05-21 14:34:42

本文介绍如何使用新版容器服务控制台快速创建标准 TKE 集群，您可在集群创建完毕后再创建 Worker Node。

前提条件

在创建集群前，您需要完成以下工作：

- 注册腾讯云账号，并完成 实名认证。
- 当您首次登录 容器服务控制台 时，需对当前账号授予腾讯云容器服务操作云服务器 CVM、负载均衡 CLB、云硬盘 CBS 等云资源的权限。详情请参见 服务授权。
- 如果要创建网络类型为私有网络的容器集群，需要在目标地域 创建一个私有网络，并且在私有网络下的目标可用区 创建一个子网。
- 集群创建过程中将使用私有网络、子网、安全组等多种资源。资源所在地域具备一定的配额限制，详情请参见 购买集群配额限制。

通过控制台创建集群

1. 登录 容器服务控制台，选择左侧导航栏中的 集群。
2. 单击集群列表上方的**新建**。
3. 在集群类型中，选择**标准集群**，单击**创建**。

1. （必填）填写集群信息

在**集群信息**页面，设置集群的基础配置和网络配置。

基础配置核心参数说明

参数名称	使用说明
集群名称	输入要创建的集群名称，不超过50个字符。
Master&Etcd 维护	• 平台管理：由腾讯云专业团队维护集群控制面组件，需支付一定的管理费用，详情请参见 集群管理费用 。 • 自行维护（已停止新建）：需用户自行购买、维护和管理控制面资源，平台不提供服务保障。
集群规格	根据业务实际情况选择合适的集群规格，详情请参见 如何选择集群规格 。集群规格可手动调整，或者通过自动升配能力自动调整。
所在地域	处在不同地域的云产品内网不通，购买后不能更换。建议选择靠近您客户的地域，以降低访问延时、提高下载速度。详情请参见 地域和可用区 。
Kubernetes 版本	推荐您选择最新版本，版本支持范围可参考 TKE Kubernetes Revision 版本历史，版本维护规则请参见 TKE Kubernetes 版本维护机制 。
运行时组件	推荐您选择 Containerd 支持的最新版本。（Kubernetes 1.24通过 Dockershim 对 Docker 的支持已移除，TKE 新建节点的容器运行时仅支持 Containerd 1.6.9。）
操作系统	TKE 提供了包含 TencentOS、Ubuntu、CentOS 在内的多款公共镜像，请根据实际情况进行选择，详情请参见 TKE 支持的公共镜像列表 。
腾讯云标签	为集群绑定标签后可实现资源的分类管理，在集群内创建的云资源会默认继承集群标签。

高级设置	• **节点 hostname 命名模式**：主机和节点命名规则，支持设置“自动命名（默认为内网 IP）”和“手动命名”。 • **设置 kubernetes 自定义组件参数**：设置和管理 Kubernetes 控制面组件（APIServer、KubeControllerManager、KubeScheduler）的参数。详情请参见 自定义 Kubernetes 组件启动参数 。 • **新增资源所属项目**：根据实际需求进行选择，新增的资源将会自动分配到该项目下。详情请参见 集群新增资源所属项目说明 。 • **删除保护**：开启后可阻止通过控制台或云 API 误删除本集群。 • **数据加密**：开启可对 ETCD 中存储的 Secret 数据进行加密。

网络配置核心参数说明

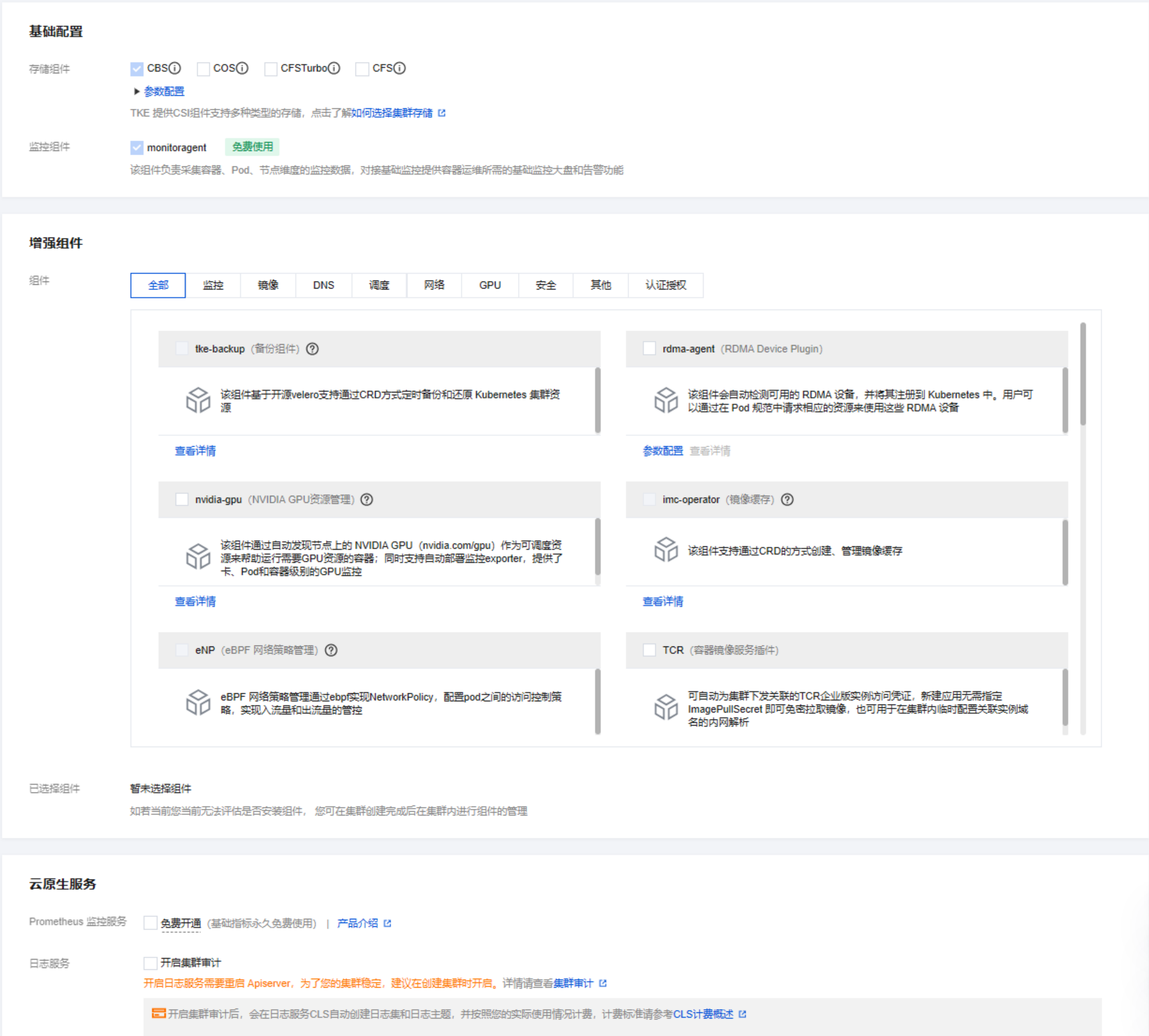
参数名称	使用说明
VPC 网络	为集群内节点选择一个私有网络，详情请参见 VPC 产品概述 。
容器网络插件	CNI (Container Network Interface) 网络插件是一种用于 Kubernetes 集群的网络解决方案，旨在提供灵活、可扩展和高性能的网络连接。CNI TKE 网络插件遵循 CNI 规范，允许 Kubernetes 集群与各种网络解决方案无缝集成，以满足不同的业务需求和性能要求。TKE 提供了3种网络 CNI 插件方案：VPC-CNI、Global Router 和 Cilium-Overlay，详情请参见 如何选择容器网络模式 。 **注意：** TKE 推荐用户在公有云场景下使用 “VPC-CNI” 网络方案，在注册节点场景下使用 “Cilium-Overlay” 网络方案。 • **VPC-CNI 网络方案** **TKE 默认推荐用户使用VPC-CNI 方案**。VPC-CNI 网络方案是 TKE 基于腾讯云 VPC 实现的容器网络接口 CNI，可以将 VPC 原生的弹性网卡直接分配给 Pod，实现 Pod 之间的网络互联，适用于对时延有较高需求的场景。该网络模式下，容器与节点分布在同一网络平面，容器 IP 为 IPAMD 组件分配的弹性网卡 IP。详情请参见 VPC-CNI 方案介绍 。 • **Global Router 网络方案** Global Router 网络方案是 TKE 基于腾讯云 VPC 的全局路由能力实现的容器网络接口 CNI。其中 Pod CIDR 网段独立于 VPC 的 CIDR 网段，不同节点的 Pod CIDR 信息会通过全局路由方式下发到 VPC，实现跨节点 Pod 的互访。详情请参见 Global Router 方案介绍 。 • **Cilium-Overlay 网络方案** Cilium-Overlay 网络方案是 TKE 基于 Cilium VXLan 实现的容器网络插件，实现分布式云场景中，注册节点添加到 TKE 集群的网络管理。详情请参见 Cilium-Overlay 方案介绍 。

高级设置	Kube-proxy 代理模式：可选择 iptables 或 ipvs。ipvs 适用于将在集群中运行大规模服务的场景，开启后不能关闭。详情请参见 集群启用 IPVS 。

2.（选填）填写组件配置

> **说明：**
> 所选组件需待节点资源就绪后才能正常部署，您也可以跳过当前步骤，等待集群创建初始化完毕后在集群内通过组件管理页安装，详情请参见 组件管理 。

在**组件配置**页面，设置集群的组件和云原生服务，如下图所示：

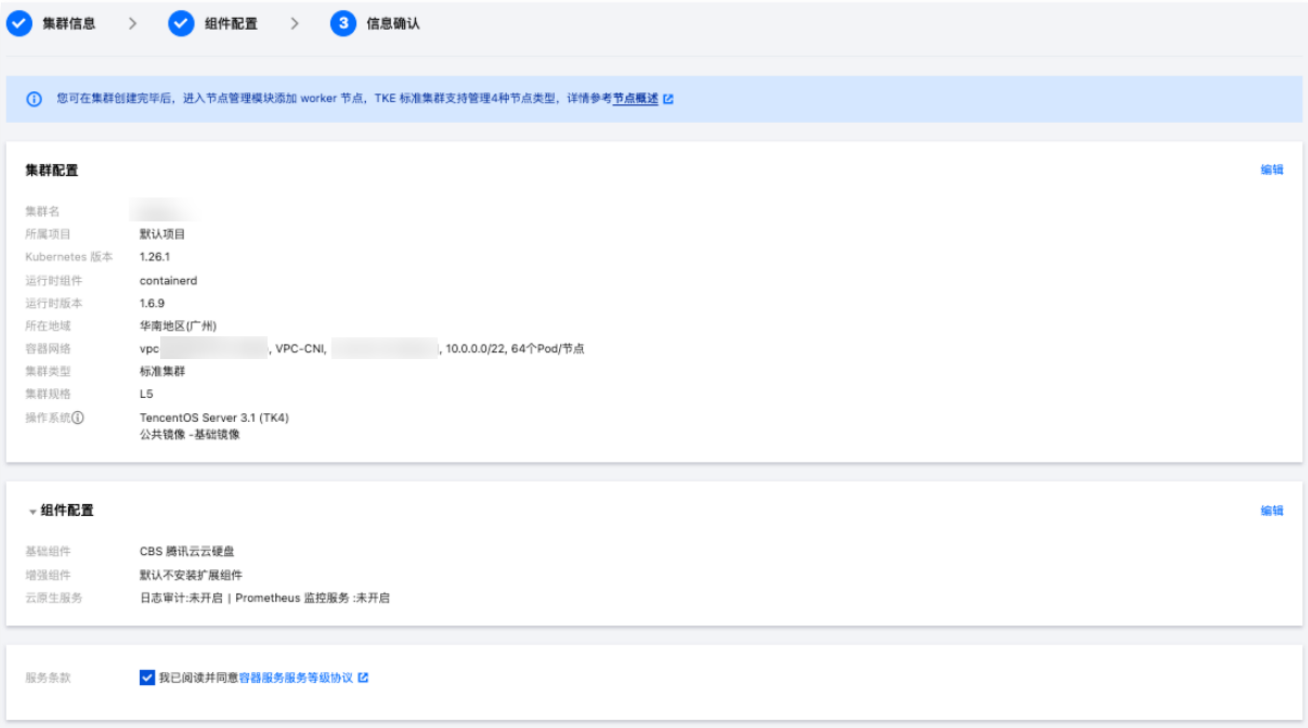


云原生服务参数说明

参数名称	使用说明
Prometheus 监控服务	开启后，您可以按照实际需求灵活配置数据采集规则，并按需要配置告警规则，配置完成后即可打开 Grafana 查看监控数据。详情请参见 Prometheus 监控服务 。
日志服务	选择是否开通集群审计服务。详情请参见 集群审计 。

3. 信息确认

在**信息确认**页面，确认集群的已选配置信息和费用，勾选**我已阅读并同意** 容器服务服务等级协议 。如下图所示：



单击**完成**即可创建一个不包含 Worker Node 的 TKE 标准集群。

相关操作

为集群添加 Worker 节点，详情请参见 节点概述 。

更改集群操作系统

最近更新时间：2023-09-18 16:52:31

操作系统说明

- 修改操作系统只影响后续新增的节点或重装的节点，对存量节点的操作系统无影响。
- 同一集群下节点使用不同版本操作系统，不会对集群功能产生影响。
- 同一脚本不一定适用于所有操作系统，建议您对节点进行脚本配置之后，验证该节点操作系统是否与此脚本相适配。
- 如需使用自定义镜像功能，请 在线咨询 申请。

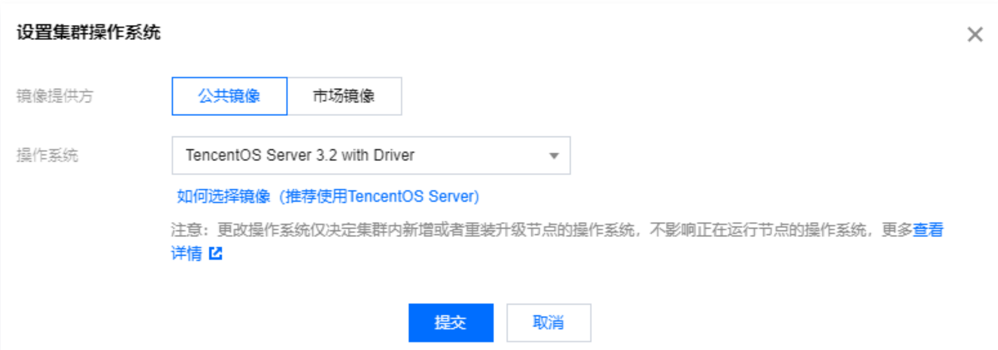
> **注意**
> 如果您需要使用自定义镜像功能，请使用容器服务提供的基础镜像来制作自定义镜像。详情请参见 自定义镜像说明 。

操作步骤

更改集群默认操作系统

> **说明**
> 进行集群默认操作系统更改操作之前，请仔细阅读 操作系统说明 以知悉相关风险。

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择目标集群所在行右侧的**查看集群凭证**，进入集群基本信息页。
3. 在**节点和网络信息**中，单击**默认操作系统**最右侧的。
4. 在**设置集群操作系统**中，选择镜像提供方并更改操作系统。如下图所示：



5. 单击**提交**。

删除集群

最近更新时间：2024-01-16 14:45:31

操作场景

本文介绍如何通过腾讯云容器服务控制台删除不再使用的集群，以免产生不必要的费用。删除集群界面支持展示集群内已有的全部资源，您可查看被销毁的资源，并按需选择是否保留部分资源，请确保您是在知晓操作风险的情况下进行删除操作。

> **注意：**
> - TKE 集群禁止级联删除：如果您的集群中仍然存在普通节点、原生节点、超级节点，请移除集群中的节点资源后再删除集群。
> - 集群删除过程针对开启删除保护的资源不会被删除（CLB、VPC 终端节点服务、CVM），仍会产生计费。如后续不再使用，请前往对应控制台进行处理。

操作步骤

关闭集群删除保护

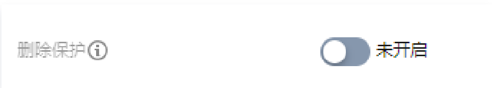
方式1

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表中，选择需删除集群所在行右侧的**更多 > 关闭集群删除保护**。如下图所示：

3. 在**关闭集群保护**弹窗中单击**确认**即可。

方式2

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表中，单击需删除集群名称，进入集群详情页。
3. 在集群基本信息页面中，关闭**删除保护**。如下图所示：

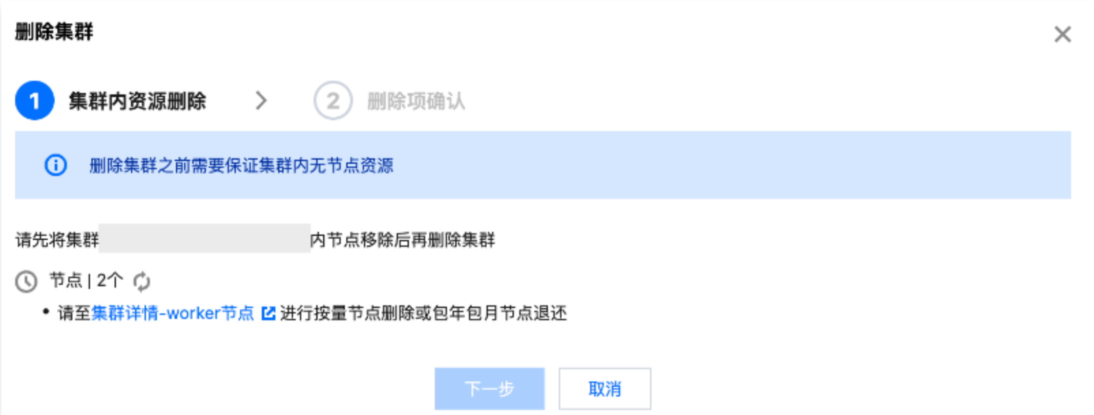


4. 在**关闭集群保护**弹窗中单击**确认**即可。

删除集群

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表中，选择需删除集群所在行右侧的**更多 > 删除**。如下图所示：

3. 如果您的集群中仍然存在普通节点、原生节点、超级节点，请移除集群中的节点资源后再删除集群。



4. 集群内无节点后，点击**下一步**进入删除项确认，在**删除集群**弹窗中，按需选择保留或删除该集群下已有资源。如下图所示：

5. 查阅集群删除操作风险提示，勾选“我已知晓以上信息并确认删除集群”。
6. 单击**确定**即可删除集群。

集群扩缩容

最近更新时间：2024-01-09 19:58:41

操作场景

本文档指导您对集群进行扩缩容，手动或自动处理应用对资源需求量的变化。TKE 支持以下三种扩缩容方法，您可结合实际情况进行选择：

- 手动添加/移出节点。
- 通过弹性伸缩自动添加/移出节点。
- 通过超级节点完成应用层的扩缩容，无需通过节点进行扩缩容。

前提条件

1. 已登录 容器服务控制台 。
2. 已 创建集群 。

操作步骤

手动添加/移出节点

您可通过新建节点或添加已有节点两种方式进行手动添加节点，实现集群的手动扩容。通过移出节点，实现集群的手动缩容。

新建节点

新建节点过程中，您可以在**新建普通节点**页面配置云服务器（CVM），对集群进行扩容。具体操作请参考 新建节点 。

添加已有节点

> **注意**
> - 当前仅支持添加同一 VPC 下的 CVM。
> - 添加已有节点到集群，会根据您的设置重装该 CVM 的操作系统。
> - 添加已有节点到集群，会迁移 CVM 所属项目到集群所设置的项目。
> - 有且仅有一块数据盘的节点加入到集群，可以选择设置数据盘挂载相关参数。

添加过程中，您可以在**添加已有节点**页面选择并配置需要添加到集群的 CVM，对集群进行扩容。具体操作请参考 添加已有 CVM 机器至集群管理 。

移出节点

请参考 移出节点 对集群进行缩容。

通过弹性伸缩自动添加/移出节点

弹性伸缩依赖社区组件 Cluster Autoscaler（CA），可以动态地调整集群的节点数量来满足业务的资源需求。更多弹性伸缩原理请参见 节点池概述 。

通过超级节点进行业务扩容

超级节点是一种调度能力，支持将标准 Kubernetes 集群中的 Pod 调度到集群服务器节点之外的资源中，实现资源不足时的动态资源补给。详情可参见 超级节点概述 。

常见问题

扩容缩容的相关问题可参见 扩容缩容相关 。

连接集群

最近更新时间：2025-05-29 15:18:11

操作场景

您可以通过 Kubernetes 命令行工具 Kubectl 从本地客户端机器连接到 TKE 集群。本文档指导您如何连接集群。

> **注意：**
> - 开启集群公网访问前，请确保您的集群内存在节点资源。
> - 如果使用子账号生成的 kubeconfig 文件无法访问资源，而主账号可以正常访问，通常是由于子账号缺少相应的权限。请为子账号授予所需的访问权限后再进行操作。具体授权方法请参考 获取凭证 。

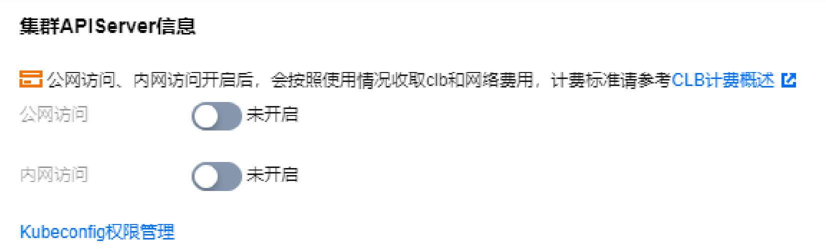
方案一：通过 Cloud Shell 连接集群

TKE 集成了腾讯云 Cloud Shell，您可以在腾讯云控制台上实现一键连接集群的能力，通过 kubectl 实现对集群的灵活操作。

操作步骤

步骤1：开启集群公网访问

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群所在地域，单击目标集群 ID，进入集群详情页。
3. 在**集群基本信息**页面，查看集群访问开启状态，如下图所示：



4. 单击开启公网访问。开启公网访问时，需配置相关参数，如：
   - **安全组**：开启公网访问后，会自动分配一个外网 CLB 作为访问端口。您可以通过安全组来配置来源授权，我们会将安全组绑定到外网 CLB 上，以达到访问控制的效果。集群访问代理的流量默认走443端口，请确认您选择的安全组内针对来源 IP 已放通443端口，以确保开启功能后集群可以正常访问。
   - **运营商类型、网络计费模式、带宽上限**：CLB 相关参数，详情请参见 CLB 创建指南，根据实际需求进行设置。
   - **访问方式**：选择公网域名后，您需要传入自定义域名，我们会为您传入的域名进行安全签名，您需要自行配置公网解析。选择 CLB 默认域名后，您无需再手动配置域名解析等操作。
5. 单击**保存**。确认公网访问已开启。

步骤2：使用 Cloud Shell 连接集群

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群所在地域，单击目标集群右侧的**更多 > 连接集群**，如下图所示：

3. 在控制台下方出现 Cloud Shell 入口，您可以直接在命令框里面输入 kubectl 指令，如下图所示：

方案二：通过本地计算机连接集群

前提条件

请安装 curl。

操作步骤

步骤1：安装 Kubectl 工具

1. 参考 Installing and Setting up kubectl，安装 Kubectl 工具。您可根据操作系统的类型，选择获取 Kubectl 工具的方式：

> **注意：**
> - 如果您已经安装 Kubectl 工具，请忽略本步骤。
> - 请根据实际需求，将命令行中的 “v1.18.4” 替换成业务所需的 Kubectl 版本。客户端的 Kubectl 与服务端的 Kubernetes 的最高版本需保持一致，您可以在**基本信息**的“集群信息”模块里查看 Kubernetes 版本。

Mac OS X 系统

执行以下命令，获取 Kubectl 工具：

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.18.4/bin/darwin/amd64/kubectl
```

Linux 系统

执行以下命令，获取 Kubectl 工具：

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.18.4/bin/linux/amd64/kubectl
```

Windows 系统

执行以下命令，获取 Kubectl 工具：

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/v1.18.4/bin/windows/amd64/kubectl.exe
```

2. 此步骤以 Linux 系统为例。执行以下命令，添加执行权限。

```
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

3. 执行以下命令，测试安装结果。

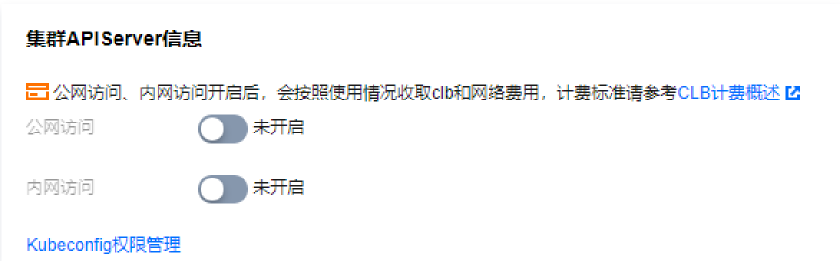
```
kubectl version
```

如若输出类似以下版本信息，即表示安装成功。

```
Client Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2",
GitCommit:"08e099554f3c31f6e6f07b448ab3ed78d0520507", GitTreeState:"clean", BuildDate:"2017-01-
12T04:57:25Z", GoVersion:"go1.7.4", Compiler:"gc", Platform:"linux/amd64"}
```

步骤2：开启集群访问

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页，选择集群所在地域，单击目标集群 ID，进入集群详情页。
3. 在集群基本信息页，查看集群访问开启状态，如下图所示：



开启公网访问

开启公网访问时，需配置相关参数：
- **安全组**：开启公网访问后，会自动分配一个外网 CLB 作为访问端口。您可以通过安全组来配置来源授权，我们会将安全组绑定到外网 CLB 上，以达到访问控制的效果。集群访问代理的流量默认走443端口，请确认您选择的安全组内针对来源 IP 已放通443端口，以确保开启功能后集群可以正常访问。
- **运营商类型、网络计费模式、带宽上限**：CLB 相关参数，详情请参见 CLB 创建指南，根据实际需求进行设置。
- **访问方式**：选择公网域名后，您需要传入自定义域名，我们会为您传入的域名进行安全签名，您需要自行配置公网解析。选择 CLB 默认域名后，您无需再手动配置域名解析等操作。

开启内网访问

开启内网访问时，需配置相关参数：
- **负载均衡器**：
  - 自动创建：自动创建CLB用于公网/内网访问Service，CLB 的生命周期由 TKE 管理。
  - 使用已有：使用当前未被TKE使用的应用型CLB用于公网/内网访问Service，请勿手动修改由TKE创建的CLB监听器。
- **LB所在子网**：默认不开启。开启内网访问时，需配置一个子网，开启成功后将在已配置的子网中分配 IP 地址。
- **访问方式**：选择内网域名后，您需要传入自定义域名，我们会为您传入的域名进行安全签名，您需要自行配置内网解析。选择内网 IP 后，我们会为您分配内网 IP 并安全签名。

步骤3：获取 KubeConfig

TKE 提供了两种 KubeConfig，分别用于公网访问和内网访问。开启集群访问后，即可按如下步骤获取对应的 KubeConfig：
1. 在**集群详情 > 基本信息**中，查看“集群 APIServer 信息”。
2. 在对应集群访问的开关下方，复制或下载 **KubeConfig**，或查看外网访问安全组、访问域名（开启访问时配置）、访问 IP。如下图所示：

步骤4：配置 KubeConfig 并访问 Kubernetes 集群

1. 根据实际情况进行集群凭据配置。
   配置前，请判断当前访问客户端是否已经配置过任何集群的访问凭证：
   - 否，即 `~/.kube/config` 文件内容为空，可直接复制已获取的 KubeConfig 访问凭证内容并粘贴入 `~/.kube/config` 中。若客户端无 `~/.kube/config` 文件，您可直接创建。
   - 是，您可下载已获取的 KubeConfig 至指定位置，并依次执行以下命令以合并多个集群的 config。

   ```
   KUBECONFIG=~/.kube/config:~/Downloads/cls-3jju4zdc-config kubectl config view --merge --flatten > ~/.kube/config
   ```

   > 说明：
   > 此处需要将文件下载到需要连接集群的客户端内，并且注意文件名。

   ```
   export KUBECONFIG=~/.kube/config
   ```

   其中，`~/Downloads/cls-3jju4zdc-config` 为本集群的 KubeConfig 的文件路径，请替换为下载至本地后的实际路径。

   > 注意：
   > 您需要修改集群的 kubeconfig 文件中的 user 和 name，以确保每个集群都有唯一的配置，这样才能保证切换集群后的正常访问。

2. 完成 KubeConfig 配置后，依次执行以下命令查看并切换 context 以访问本集群。

   ```
   kubectl config get-contexts
   ```

   > 说明：
   > 下面例子中的“cls-3jju4zdc-context-default”为上一步 `kubectl config get-contexts` 中查到的集群名称 NAME。

   ```
   kubectl config use-context cls-3jju4zdc-context-default
   ```

3. 执行以下命令，测试是否可正常访问集群。

   ```
   kubectl get node
   ```

   成功响应示例：

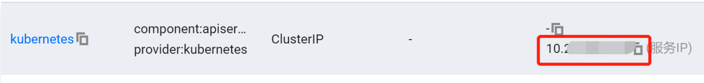
如果无法连接请查看是否已经开启公网访问或内网访问入口，并确保访问客户端在指定的网络环境内。

方案三：通过集群内节点连接集群

操作步骤

步骤1：获取 Kubernetes 的 service IP

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群所在地域，单击目标集群 ID，进入集群详情页。
3. 在集群详情页中，选择左侧的**服务与路由 > Service** 获取 default 命名空间下 Kubernetes 的 service IP，如下图所示：



> **说明：**
> Kubernetes service 是 ClusterIP 模式，仅适用于集群内访问。

步骤2：配置 KubeConfig 并访问 Kubernetes 集群

1. 调用云 API 接口 DescribeClusterKubeconfig 获取当前使用账号的凭证信息 Kubeconfig 文件。
2. 使用步骤1中获取的 kubernetes service ip，替换通过 API 获取的 Kubeconfig 中的 clusters.cluster.server 字段为 https://<IP>:443，并整理格式如下图所示：

3. 登录任意一台集群内节点，使用上步生成的 Kubeconfig 内容替换节点上 `~/.kube/config` 文件内容。
4. 执行以下命令， 测试是否可正常访问集群。

```
kubectl get node
```

成功响应式例：

相关说明

Kubectl 命令行介绍

Kubectl 是一个用于 Kubernetes 集群操作的命令行工具。本文涵盖 kubectl 语法、常见命令操作并提供常见示例。有关每个命令（包括所有主命令和子命令）的详细信息，请参见 kubectl 参考文档 或使用 `kubectl help` 命令查看详细帮助。

升级集群

最近更新时间：2025-07-04 17:54:12

操作场景

腾讯云容器服务 TKE 提供升级 Kubernetes 版本的功能，您可通过此功能对运行中的 Kubernetes 集群进行升级。升级的过程为：升级的前置检查、升级 Master 和升级 Node。

升级须知

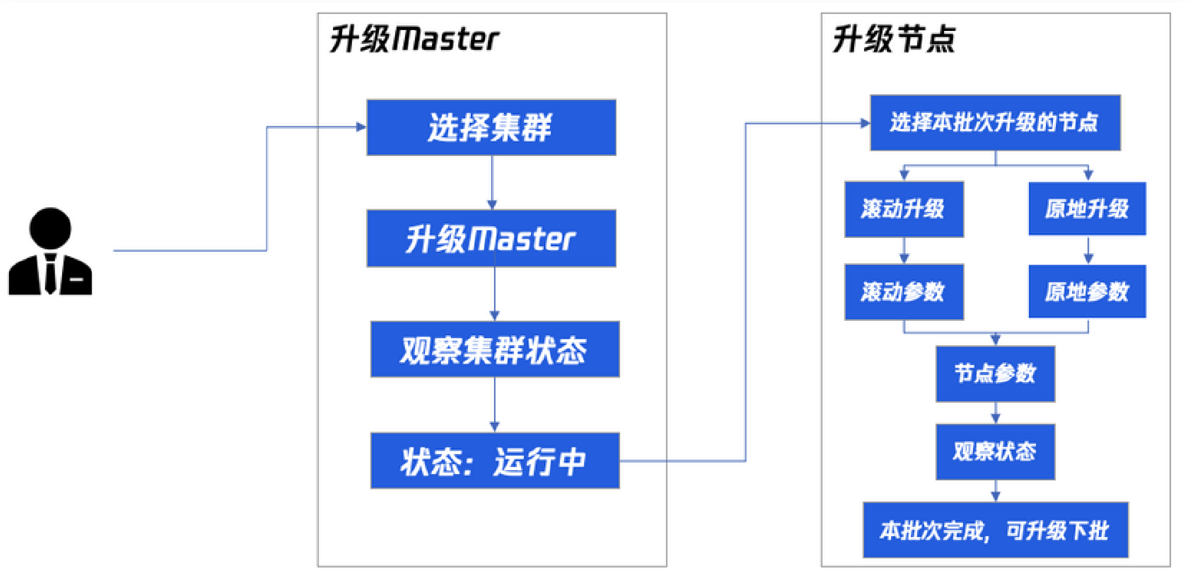
- **升级属于不可逆操作，请谨慎进行。**
- 在升级集群前，确保集群状态均为健康状态。若集群不正常，您可以自行修复，也可以通过 在线咨询 联系我们协助您进行修复。
- **升级顺序**：升级集群时，必须先完成 Master 版本升级，再尽快完成 Node 版本升级，且升级过程中不建议对集群进行任何操作。
- **仅支持向上升级 TKE 提供的最近 Kubernetes 版本**，不支持跨多个版本升级（例如1.8跳过1.10直接升级至1.12），且仅当集群内 Master 版本和 Node 版本一致时才可继续升级下一个版本。
- **CSI-CFS 插件不兼容问题**：关于 CSI 插件 COS CSI 和 CFS CSI，不同 Kubernetes 版本适配的 CSI 插件版本有以下差异，因此建议您：将集群升级到 TKE 1.14及以上版本时，在组件管理页面重新安装 CSI 插件（重建组件不影响已经在使用中的 COS 和 CFS 存储）。
  - Kubernetes 1.10 和 Kubernetes 1.12 版本适配的 CSI 插件版本是**0.3**。
  - Kubernetes 1.14 及以上版本适配的 CSI 插件版本是**1.0**。
- **HPA 失效问题**：在 Kubernetes 1.18版本之前，HPA 中所引用的 deployment 对象的 apiVersion 可能是 `extensions/v1beta1`，而 Kubernetes 1.18版本之后，deployment 的 apiVersion 只有 `apps/v1`，可能导致集群升级到 Kubernetes 1.18之后，HPA 会失效。如果您使用了 HPA 功能，建议在升级之前，执行如下命令，将 HPA 对象中的 apiVersion 切换到 `apps/v1`。

```
kubectl patch hpa test -p '{"spec":{"scaleTargetRef":{"apiVersion":"apps/v1"}}}'
```

- **Helm 应用失效问题**：每个应用支持的 Kubernetes 的版本不太相同，包括通过应用市场安装的应用或是通过第三方安装的应用。建议在升级集群前，查看已安装在集群里的应用列表，确认其支持的集群版本范围。有些应用本身对高版本的 Kubernetes 有适配，此时可能需要升级应用的版本。有些应用可能还没有对高版本的 Kubernetes 适配，此时请谨慎升级集群。
- **Nginx Ingress 版本问题**：extensions/v1beta1 和 networking.k8s.io/v1beta1 API 版本的 Ingress 不在 v1.22 版本中继续提供，详情请 查看文档。您集群里面 Nginx Ingress 的版本可能比较低，在升级 Kubernetes 版本到 v1.22 及以上版本时，在组件管理页面升级 Nginx Ingress 组件。

操作步骤

升级集群的两个步骤是 升级 Master Kubernetes 版本 和 升级 Node Kubernetes 版本。具体信息如下图所示：



升级 Master Kubernetes 版本

> **注意：**
> **目前已支持托管集群、独立集群 Master 版本升级**，且升级需要花费5 – 10分钟，在此期间您将无法操作您的集群。

Master 大版本与小版本升级说明

目前 Master 升级已支持**大版本升级**（例如从1.14升级到1.16）、**小版本升级**（例如从1.14.3升级到1.14.6，或者从v1.18.4-tke.5升级到v1.18.4-tke.6），强烈建议您升级前先查阅对应的功能发布记录：

- 在升级 Kubernetes 大版本之前，建议您查阅 TKE Kubernetes 大版本更新说明。
- 在升级 Kubernetes 小版本之前，建议您查阅 TKE Kubernetes Revision 版本历史。

> **说明：**
> - 当大版本升级（例如1.12升级到1.14），若您已设置自定义参数，需要您重新设置新版本的自定义参数。原参数不保留。详情可参见 自定义 Kubernetes 组件启动参数。
> - 当小版本升级时，您已设置的自定义参数会被保留，无需重新设置。
> - 集群 Master 升级后，apiVersion 会更新为当前 Kubernetes 版本支持的最新版本。

注意事项

- **升级前，请详细阅读 升级须知。**
- 1.7.8版本 TKE 集群，网络模式为 bridge，集群升级不会自动切换网络模式为 cni。
- 集群升级不会切换 kube-dns 为 core-dns。
- 创建集群时设置的部分特性（例如支持 ipvs），当集群 Master 版本升级到1.10和1.12后将不支持开通。
- 存量的集群升级后，若 Master 版本在1.10版本以上，Node 节点版本在1.8版本以下，PVC 功能将不可用。
- 升级 Master 完成后，建议您尽快升级节点版本。

Master 升级技术原理

Master 节点升级分为3个步骤：前置组件升级、Master 节点组件升级、后置组件升级。

- **升级前置操作**：将会升级前置依赖的组件，例如监控组件等，以防兼容性问题导致组件异常。
- **Master 节点组件升级**：将按组件顺序对所有 Master 的对应组件进行升级，所有 Master 的某个组件升级完成后再进行下一个组件的升级。
  TKE 会先升级 kube-apiserver，后升级 kube-controller-manager 和 kube-scheduler，最后升级 kubelet。具体步骤如下：
  - 重新生成 kube-apiserver 组件静态 Pod 对应的 YAML 文件内容。
  - 检查当前 kube-apiserver pod 是否健康，Kubernetes 版本是否正常。
  - 同理，依次升级 kube-controller-manager 和 kube-scheduler。
  - 升级 kubelet，并检查所在 Master 节点是否 ready。
- **升级后置操作**：
  - 按需升级后置依赖组件，如 kube-proxy（并将其滚动更新策略改为 on delete）、cluster-autoscaler 组件等。
  - 执行一些后置依赖组件相关的兼容性操作，防止兼容性问题导致组件异常。

Master 升级操作步骤

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择需进行 Master Kubernetes 版本升级的集群 ID，进入集群详情页。
3. 在**基本信息**页面的集群信息模块，单击 Master 版本右侧的**升级**。如下图所示：

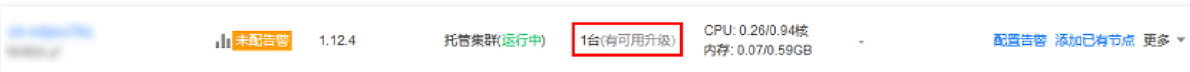
4. 在弹出窗口中单击**提交**，等待升级完成。
5. 您可以在集群管理页对应的集群状态处查看升级进度，也可以在升级进度弹窗中查看当前升级进展、Master 节点升级进度（托管集群不显示具体 Master 节点列表）、升级持续时间。如下图所示：

6. 

7. 该示例集群 Kubernetes 版本升级前 Master 版本为1.10.5，升级完成后为 Master 1.12.4。如下图所示：

升级 Node Kubernetes 版本

集群 Master Kubernetes 版本升级完成后，集群列表页将显示该集群节点有可用升级。如下图所示：



注意事项

- **升级前，请详细阅读** 升级须知 。
- 当 Node 节点处于运行中时，可进行升级操作。

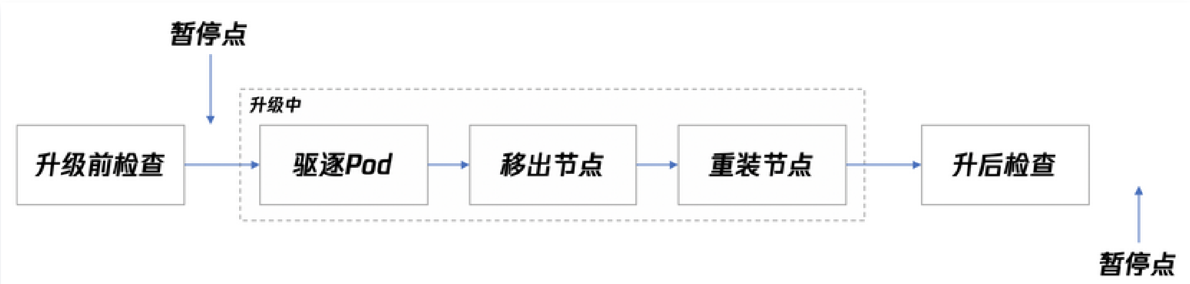
选择升级方式

升级 Node Kubernetes 版本支持 重装滚动升级 和 原地滚动升级 两种升级方式。您可按需选择：

- **重装滚动升级**：采用重装节点的方式升级节点版本。仅支持大版本升级，例如1.10可升级至1.12。
- **原地滚动升级**：原地不重装，仅替换 kubelet、kube-proxy 等组件。支持大版本、小版本升级，例如1.10可升级至1.12，1.14.3可升级至1.14.8。

重装滚动升级执行原理

基于重装的节点升级采用滚动升级的方式，同一时间只会对一个节点进行升级，只有当前节点升级成功才会进行下个节点的升级。如下图所示：



- **升级前检查**：对节点上的 Pod 进行驱逐前的检查。具体的升级前检查项如下：

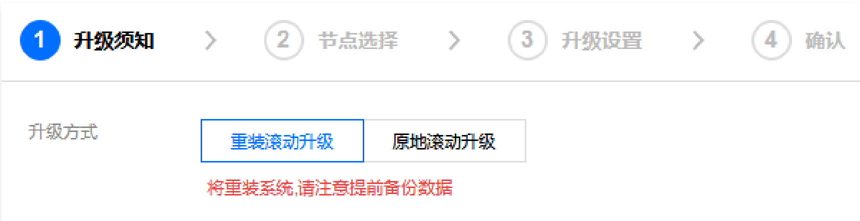
- ○ 统计该节点所有工作负载的 Pod 个数，若驱逐节点后，任何工作负载的 Pod 数目变为0 ，则检查不通过，不能进行升级。
- ○ 以下系统控制面工作负载将被忽略：
  - ○ l7-lb-controller
  - ○ cbs-provisioner
  - ○ hpa-metrics-server
  - ○ service-controller
  - ○ cluster-autoscaler
- **驱逐 Pod**：首先将节点标记为不可调度，随后驱逐或者删除节点上所有 Pod。
- **移出节点**：将节点从集群中移除。该步骤只进行基本的清理工作，不会删除节点在集群中的 Node 实例，所以节点的 label、taint 等属性都可保留。
- **重装节点**：重装节点的操作系统，并重新安装新版本 kubelet。
- **升级后检查**：检查节点是否 ready，是否为可调度的，并检查当前不可用 Pod 比例是否超过最大值。

重装滚动升级操作步骤（Node Kubernetes 版本）

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择需进行 Node Kubernetes 版本升级的集群 ID，进入集群详情页。
3. 在**基本信息**页面的集群信息模块，单击 Node Kubernetes 版本右侧的**升级**。如下图所示：

4. 在**升级须知**步骤中，选择升级方式为**重装滚动升级**，仔细阅读升级须知。勾选**我已阅读并同意上述技术条款**，并单击**下一步**。如下图所示：

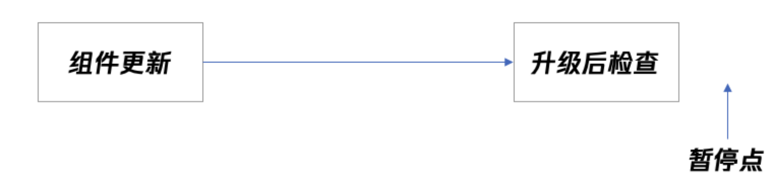
> **注意：**
> 该升级方式将重装系统，原有数据将会被清除，请注意提前备份数据。



5. 在**节点选择**步骤中，选择本批次需要升级的节点，并单击**下一步**。
6. 在**升级设置**步骤中，按需填写节点信息，并单击**下一步**。
7. 在**确认**步骤中，确认信息并单击**完成**即可开始升级。
8. 查看节点升级进度， 直至所有节点升级完成。

原地滚动升级执行原理

节点原地升级采用滚动升级的方式，同一时间只会对一个节点进行升级，只有当前节点升级成功才会进行下个节点的升级。原地升级目前已同时支持大版本升级以及大版本的不同小版本升级。如下图所示：



步骤描述如下：
- **组件更新**：替换和重启节点上的 kubelet 和二进制部署的 kube-proxy 组件。
  注：以 DaemonSet 形式部署的 kube-proxy 组件，会在节点升级完成后再更新。
- **升级后检查**：检查节点是否 ready，并检查当前不可用 Pod 比例是否超过最大值。

原地滚动升级操作步骤

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择需进行 Node Kubernetes 升级的集群 ID，进入集群详情页。
3. 在**基本信息**页面的集群信息模块，单击 Node Kubernetes 版本右侧的**升级**。
4. 在**升级须知**中，选择升级方式为“原地滚动升级”，仔细阅读升级须知。勾选**我已阅读并同意上述技术条款**，并单击**下一步**。如下图所示：

5. 在**节点选择**步骤中，选择本批次需要升级的节点，单击**下一步**。
6. 在**确认**步骤中，确认信息并单击**完成**即可开始升级。
7. 查看节点升级进度， 直至所有节点升级完成。

集群启用 IPVS

最近更新时间：2023-09-21 10:51:31

操作场景

默认情况下，Kube-proxy 使用 iptables 来实现 Service 到 Pod 之间的负载均衡。TKE 支持快速开启基于 IPVS 来承接流量并实现负载均衡。开启 IPVS 适用于大规模集群，可提供更好的可扩展性和性能。

注意事项

- 本功能仅在创建集群时开启，暂不支持对存量集群的修改。
- IPVS 开启针对全集群生效，建议不要手动修改集群内 IPVS 和 Iptables 混用。
- 集群开启 IPVS 后不可关闭。
- IPVS 仅针对 Kubernetes 版本1.10及以上的 TKE 集群生效。

操作步骤

1. 登录 容器服务控制台 。
2. 参考 创建集群，在**创建集群**页面中，将 **Kubernetes 版本**设置为高于1.10的 Kubernetes 版本。单击**高级设置**，开启 “ipvs 支持”。如下图所示：

3. 按照页面提示逐步操作，完成集群的创建。

自定义 Kubernetes 组件启动参数

最近更新时间：2024-10-29 16:09:22

操作场景

本文主要介绍如何在 TKE 集群中设置和管理 Kubernetes 控制面组件（APIServer、KubeControllerManager、KubeScheduler）的参数。

注意事项

- 如果当前开放的参数不满足您的业务需求，您可以 提交工单 进行咨询。在提交工单时，请提供账号 APPID、集群 ID、需要设置的组件和组件的参数。
- 集群大版本升级场景：由于 Kubernetes 跨版本升级后，启动参数可能存在不兼容的情况，升级前请确认是否存在参数废弃情况。

参数介绍

以下是用户可自定义配置的参数范围和 TKE 集群提供的默认值。请确保您修改的参数取值在系统支持的范围内。每个参数的具体含义可以从社区获取。

APIServer

参数名称	参数类型	默认值	取值范围	K8s 适用版本
--feature-gates	mapStringBool	-	-	1.14+
--goaway-chance	float	0	[0, 0.02]	1.18+

KubeControllerManager

参数名称	参数类型	默认值	取值范围	K8s 适用版本
--kube-api-burst	int	30	[30, 3000]	1.14+
--kube-api-qps	qps	20	[20, 5000]	1.14+
--node-monitor-grace-period	duration	40s	[40s, 1h]	1.14+
--node-startup-grace-period	duration	1m0s	[1m, 1h]	1.14+
--pod-eviction-timeout	duration	5m	[5m, 1h]	1.14-1.24
--cluster-signing-duration	duration	8760h0m0s	[8760h, 87600h]	1.22+
--concurrent-cron-job-syncs	int	5	[5, 20]	1.20+
--concurrent-deployment-syncs	int	5	[5, 20]	1.14+
--concurrent-endpoint-syncs	int	5	[5, 20]	1.14+
--concurrent-ephemeralvolume-syncs	int	5	[5, 20]	1.24+
--concurrent-gc-syncs	int	20	[20, 50]	1.14+
--concurrent-horizontal-pod-autoscaler-syncs	int	5	[5, 20]	1.26+
--concurrent-job-syncs	int	5	[5, 20]	1.14+
--concurrent-namespace-syncs	int	10	[10, 20]	1.14+
--concurrent-rc-syncs	int	5	[5, 20]	1.14+
--concurrent-replicaset-syncs	int	5	[5, 20]	1.14+

--concurrent-resource-quota-syncs	int	5	[5, 20]	1.14+
--concurrent-service-endpoint-syncs	int	5	[5, 20]	1.16+
--concurrent-service-syncs	int	1	[1, 5]	1.14+
--concurrent-serviceaccount-token-syncs	int	5	[5, 20]	1.14+
--concurrent-statefulset-syncs	int	5	[5, 20]	1.16+
--concurrent-ttl-after-finished-syncs	int	5	[5, 20]	1.14+
--endpoint-updates-batch-period	duration	0	[0s, 10s]	1.14+
--endpointslice-updates-batch-period	duration	0	[0s, 10s]	1.18+
--horizontal-pod-autoscaler-cpu-initialization-period	duration	5m0s	[1m,10m]	1.14+
--horizontal-pod-autoscaler-downscale-stabilization	duration	5m0s	[1m,10m]	1.14+
--horizontal-pod-autoscaler-initial-readiness-delay	duration	30s	[10s,1m]	1.14+
--horizontal-pod-autoscaler-sync-period	duration	15s	[5s, 1m]	1.14+
--horizontal-pod-autoscaler-tolerance	float	0.1	[0.01, 0.3]	1.14+
--large-cluster-size-threshold	int	50	[10, 100]	1.14+
--namespace-sync-period	duration	5m0s	[1m, 10m]	1.14+
--node-eviction-rate	float	0.1	[0.01, 0.1]	1.14+
--resource-quota-sync-period	duration	5m0s	[1m, 10m]	1.14+
--secondary-node-eviction-rate	float	0.01	[0.001, 0.01]	1.14+
--terminated-pod-gc-threshold	int	12500	[0, 12500]	1.14+
--unhealthy-zone-threshold	float	0.55	[0.55, 1]	1.14+
--feature-gates	mapStringBool	-	-	1.14+
-v, --v int	-	-	[1, 10]	1.14+

KubeScheduler

参数名称	参数类型	默认值	取值范围	K8s 适用版本
--evict-filter-toleration	duration	5m0s	[30s,240h]	1.18+
--hard-pod-affinity-symmetric-weight	int	1	[0,100]	1.14-1.20
--kube-api-burst	int	100	[30,3000]	1.14+

--kube-api-qps	float	50	[20,5000]	1.14+
--feature-gates	mapStringBool	-	-	1.14+
--feature-gates=EnableEvictHistory	-	-	-	1.14+
--feature-gates=EnableFitIPResource	-	-	-	1.14+
--feature-gates=EnableLocalReplicas	-	-	-	1.20、1.24+

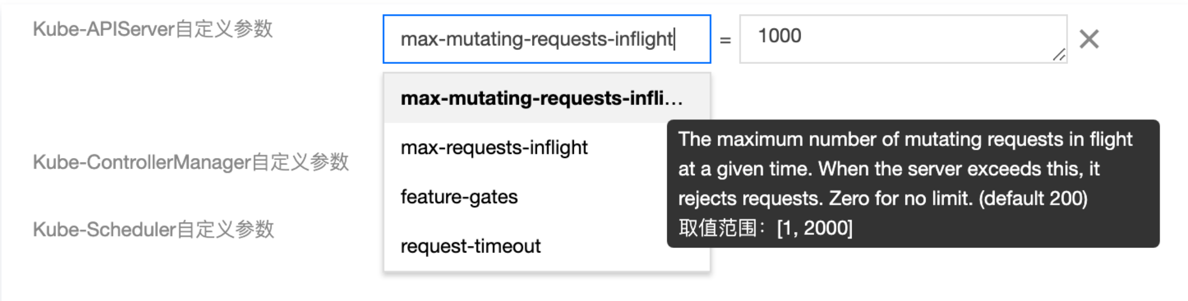
以下 feature-gates 参数在 APIServer、KubeControllerManager、KubeScheduler 组件中都支持设置：

feature-gates 名称	支持版本
StatefulSetStartOrdinal	1.26、1.28
TTLAfterFinished	1.14、1.16、1.18、1.20
MaxUnavailableStatefulSet	1.24、1.26、1.28
EphemeralContainers	1.16、1.18、1.20、1.22

操作场景

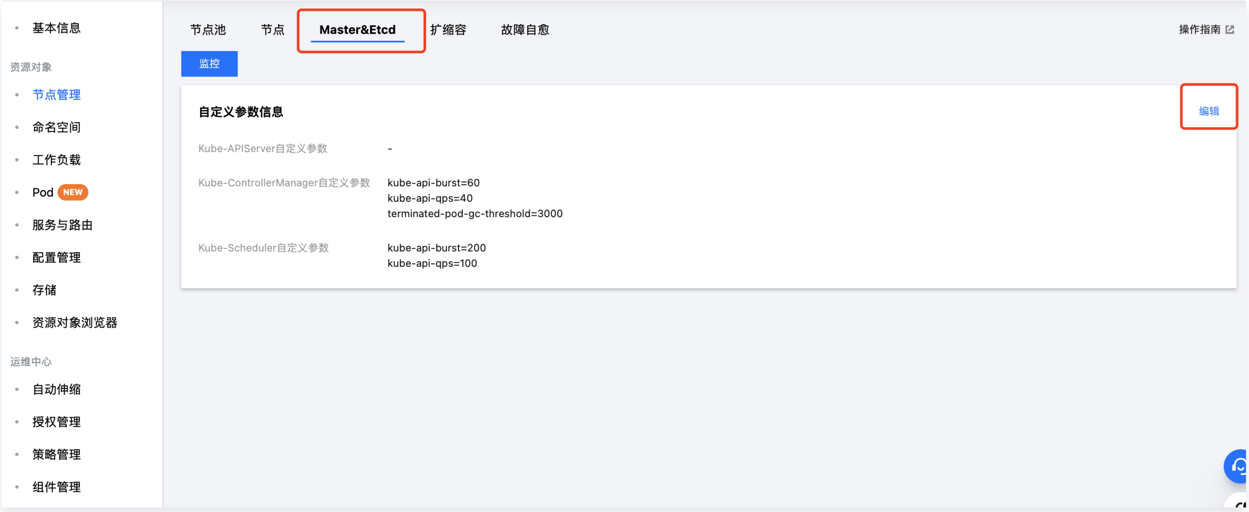
创建集群

1. 登录 腾讯云容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群页面，单击集群列表上方的**新建**。
3. 在**创建集群**页面，选择**高级设置 > 设置 kubernetes 自定义组件参数**。如下图所示：



已有集群

1. 登录 腾讯云容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群**页面，选择目标集群 ID，进入基本信息页。
3. 在左侧导航中选择**节点管理**，选择 **Master&Etcd**。
4. 单击**自定义参数**模块中的**编辑**来配置参数，如下图所示：



> **注意：**
> 控制面组件参数修改后需要滚动更新生效，为避免对业务产生影响，建议您在业务低峰期时间操作。

集群升级

1. 登录 腾讯云容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群**页面，选择目标集群 ID，进入基本信息页。
3. 在集群**基本信息**中，单击 Kubernetes 版本右侧的**升级**，同时设置 Kubernetes 组件启动参数，如下图所示：

> **注意：**
> 部分启动参数将随着 Kubernetes 版本的演进不再支持，建议您在升级前谨慎了解目标升级版本和当前参数是否存在废弃和不兼容的情况，更多信息可参考社区 Kubernetes Changelog 和 API 迁移指南 获取。

使用 KMS 进行 Kubernetes 数据加密

最近更新时间：2024-12-02 09:56:22

操作场景

在腾讯云 TKE 独立集群和托管集群中，使用密钥管理系统（Key Management Service，KMS）实现 Kubernetes 数据源加密，并提供丰富的密钥管理功能，针对 Kubernetes 集群中 Secret 提供强大的加密/解密能力。本文主要介绍如何通过 KMS 对 Kubernetes 集群进行数据加密。

基本概念

密钥管理系统 KMS

密钥管理系统（Key Management Service，KMS）是一款安全管理类服务，它使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。KMS 可以帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求，同时符合监管和合规要求。

数据加密

在 Kubernetes 中，可以使用 Secrets 对象来存储集群内部使用的各类敏感数据，例如数据库用户名、密码、证书、OAuth Token、SSH KEY 等，从而使得敏感信息和普通配置文件有效解耦。默认情况下，Secrets 存储在 etcd 中。
在腾讯云 TKE 标准集群的独立集群和托管集群中，您可以使用在 KMS 中创建的密钥来加密 Kubernetes Secrets 数据。KMS 加密过程基于 Kubernetes 提供的 KMS Encryption Provider 机制，使用信封加密的方式对存储在 etcd 中的 Kubernetes Secrets 数据进行自动加密和解密。Kubernetes Secrets 密钥加密和解密的过程如下：

- 当一个业务密钥需要通过 Kubernetes Secrets API 存储时，数据会首先被 API Server 生成的一个随机的数据加密密钥加密。然后，该数据密钥会被指定的 KMS 密钥加密为一个密文密钥，并存储在 etcd 中。
- 解密 Kubernetes Secrets 密钥时，系统会首先调用 KMS 的解密 OpenAPI 进行密文密钥的解密。然后，使用解密后的明文密钥对 Secrets 数据进行解密，并最终返回给用户。

前提条件

在使用腾讯云 KMS 加密 Kubernetes Secret 之前，请确保您已创建符合以下条件的容器服务独立集群或托管集群：

- Kubernetes 版本为1.18.0及以上。
- Etcd 版本为3.0及以上。

> **说明：**
> - Kubernetes 版本，您可以前往容器服务控制台 集群管理 页面，选择集群并查看**集群基本信息**。
> - Etcd 版本，需通过 Kubectl 连接 Kubernetes 集群进行查看。

使用限制及注意事项

- 使用腾讯云 KMS 加密 Kubernetes Secret 会产生计费。KMS 对 API 调用（以万次调用为单位）和用户上传密钥的托管会收取一定费用，关于 KMS 服务计费的详细说明，请参见 KMS 计费说明。
- 开启数据加密功能后，请勿通过 KMS 的 OpenAPI 或控制台禁用、删除或计划删除集群数据加解密时选择的密钥，否则会使集群 API Server 不可用，导致 Secret 和 ServiceAccount 等对象无法正常获取，影响业务应用的正常运行。

操作步骤

创建集群时开启数据加密

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击集群列表上方的**新建**。
3. 选择**标准集群**，单击**创建**。
4. 在**创建集群**页面，单击展开**高级设置**，单击 开启**数据加密**。
5. 开启**数据加密**后，选择地域和 KMS 密钥。如下图所示：

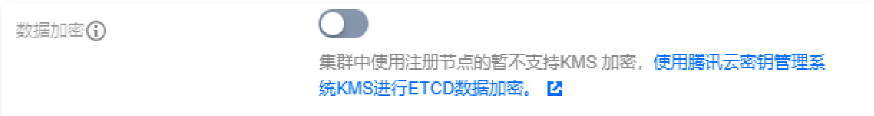
如果您尚未创建 KMS 密钥，请单击**新建 KMS 密钥**，前往 密钥管理系统（合规） 控制台创建密钥。具体操作请参见 创建密钥 。

6. 创建集群的其他配置项详情请参见 通过控制台创建集群 。配置结束后，单击**完成**。
7. 创建完成的集群将出现在 集群列表 中。您可单击集群 ID 进入集群详情页面，在集群**基本信息**中查看数据加密功能开启状态。

在已创建集群中开启数据加密

> **说明：**
> 如果您需要开启数据加密功能，请确保您当前登录的账号具有 CAM 权限，并且在该集群中具有 RBAC 的管理员或运维人员权限。更多信息请参见 TKE Kubernetes 对象级权限控制 。
> 如果您需要授权 TKE_QCSRole 角色，请确保您使用的是腾讯云账号（主账号）或拥有 CAM 管理权限的用户或角色。有关服务授权相关角色权限的说明，请参见 服务授权相关角色权限说明 。

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群**页面，选择需开启数据加密的集群 ID，进入集群详情页。
3. 在集群**基本信息**中，单击**数据加密**右侧的 ，开启数据加密。如下图所示：

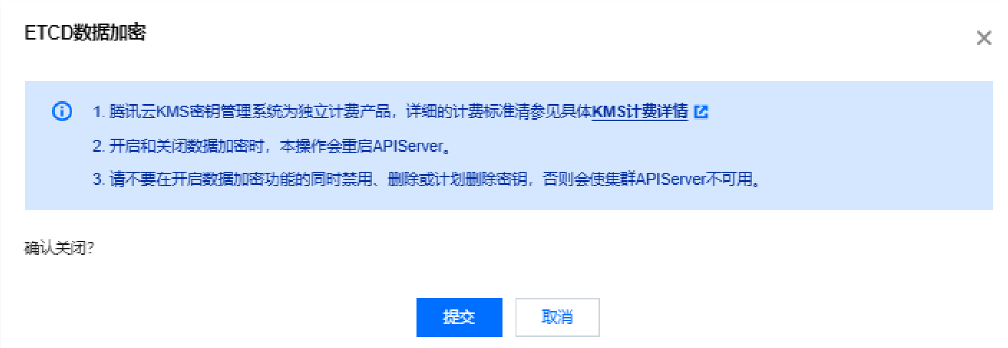


4. 在 ETCD **数据加密**中，选择地域和已有的 KMS 密钥，单击**提交**。如下图所示：

关闭数据加密

如果您不再需要使用数据加密功能，可在容器服务控制台关闭该功能。

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择需关闭数据加密的集群 ID，进入集群详情页。
3. 在集群**基本信息**中，单击**数据加密**右侧的 ，关闭数据加密。
4. 在 ETCD 数据加密确认窗口中，确认是否关闭，单击**提交**。如下图所示：


ETCD数据加密
1. 腾讯云KMS密钥管理系统为独立计费产品，详细的计费标准请参见具体KMS计费详情
2. 开启和关闭数据加密时，本操作会重启APIServer。
3. 请不要在开启数据加密功能的同时禁用、删除或计划删除密钥，否则会使集群APIServer不可用。
确认关闭？
提交
取消

镜像

镜像概述

最近更新时间：2024-11-27 18:11:42

概述

本文档介绍腾讯云容器服务 TKE 支持的镜像类型、对应的使用场景以及使用须知。

> **说明：**
> - TKE 支持的公共镜像需经过 Kubernetes 环境（如网络、存储、节点初始化、GPU 驱动等多个应用模块）的兼容性适配，并非和 CVM 服务完全对应。
> - TKE 仅针对支持的公共镜像提供 SLA 服务保障，镜像列表可参考下方表格。
> - 自定义镜像非标准操作环境，TKE 未经过兼容性适配，需要用户自行保证镜像在 Kubernetes 环境下的可用性。针对该类镜像，TKE 原则上不提供 SLA 服务和技术保障。

公共镜像

公共镜像是由腾讯云官方提供的镜像，包含基础操作系统和腾讯云提供的初始化组件，所有用户均可使用。

自定义镜像

自定义镜像由用户通过镜像制作功能制作，或通过镜像导入功能导入。仅创建者与共享者可以使用。自定义镜像属于非标准环境，腾讯云不提供官方支持及持续维护，具体使用方式请参见 自定义镜像说明 。

注意事项

- 操作系统有两个级别：**集群级别**和**节点池级别**。
  - 在集群内进行新增节点、添加已有节点、节点升级操作时，均会使用集群级别设置的操作系统。
  - 在节点池内部进行添加已有节点、节点扩容操作时，会使用节点池级别设置的操作系统。
- 修改操作系统只影响后续新增的节点或重装的节点，**对存量的节点操作系统无影响**。

TKE 支持的公共镜像列表

TKE 为您提供以下**公共镜像**，请根据实际情况进行选择，推荐您选择 Tencent OS Server 系列镜像。

镜像 ID	Os Name	控制台展示名	OS 类型	发布状态	备注
img-6n21msk1	tlinux4_x86_64_public	TencentOS Server 4 for x86_64	Tencent OS Server	内测中，请 提交工单 进行申请	内核版本：6.6.30
img-eb30mz89	tlinux3.1x86_64	TencentOS Server 3.1(TK4)	Tencent OS Server	全量发布（推荐）	内核版本：5.4.119
img-nannz3uj	tlinux2.4(tkernel4)x86_64_HCC	TencentOS Server 2.4 (TK4) HCC	Tencent OS Server	全量发布	内核版本：5.4.119
img-9axl1k53	tlinux2.4(tkernel4)x86_64	TencentOS Server 2.4(TK4)	Tencent OS Server	全量发布	内核版本：5.4.119
img-hdt9xxkt	tlinux2.4x86_64	TencentOS Server 2.4 曾用名：Tencent linux release 2.4 (Final)	Tlinux	全量发布	内核版本：4.14.105
img-487zeit5	ubuntu22.04x86_64	ubuntu22.04x86_64	Ubuntu	内测中，请 提交工单 进行申请	Ubuntu 22.04 公版内核

img-22trbn9x	ubuntu20.04x86_64	Ubuntu Server 20.04.1 LTS 64位	Ubuntu	全量发布	Ubuntu 20.04 公版内核
img-pi0ii46r	ubuntu18.04.1x86_64	Ubuntu Server 18.04 LTS 64位	Ubuntu	全量发布	Ubuntu 18.04.1 公版内核
img-n74hgdxx	centos8.2arm_64	CentOS 8.2(arm64)	CentOS	全量发布	Centos 8.2 公版内核
img-25szkc8t	centos8.0x86_64	CentOS 8.0 64位	CentOS	全量发布	Centos 8.0 公版内核
img-3la7wgnt	centos7.8.0_x64	CentOS 7.8 64位	CentOS	全量发布	Centos 7.8 公版内核
img-9qabwvbn	centos7.6.0_x64	CentOS 7.6 64位	CentOS	全量发布	Centos 7.6 公版内核
img-kp3mv36j	redhat8.6x86_64	RedHat Enterprise Linux 8.6 64位	RedHat	全量发布	RedHat 8.6 公版内核
img-0qhxz7dl	redhat7.9x86_64	RedHat Enterprise Linux 7.9 64位	RedHat	全量发布	RedHat 7.9 公版内核

常见问题

为什么在控制台无法看到上述镜像?

- 该镜像在 CVM 侧未全量开放，您可以先在 CVM 控制台中查看是否可以找到目标镜像。如果无法找到目标镜像，建议通过 提交工单 进行申请。
- 如果在创建实例时开启了某些网络特性（如注册节点、Dataplane v2 网络特性），这些特性可能对镜像选择有特定要求。
- 在普通节点场景下，部分机型对镜像的使用有特定要求。详情请参见 普通节点支持的 CVM 机型使用说明 。

自定义镜像说明

最近更新时间：2025-03-31 17:00:32

操作场景

本文档介绍如何使用腾讯云容器服务 TKE 提供的基础镜像进行自定义镜像制作。

> **注意：**
> 1. 基础镜像是指 TKE 支持的公共镜像，镜像列表请参见 镜像概述 。
> 2. 自定义镜像非标准操作环境，平台未经过兼容性适配，需要用户自行保证镜像在 Kubernetes 环境下的可用性。针对该类镜像，TKE 原则上不提供 SLA 服务和技术保障。

使用须知

- 目前仅支持同类型操作系统镜像的制作。例如，使用 CentOS 基础镜像制作 CentOS 类的自定义镜像。
- 如果您使用自定义镜像功能，**请使用 TKE 提供的基础镜像来制作自定义镜像**。
- 若 TKE 后期计划调整镜像逻辑，会提前**至少一周**通过站内信、短信、邮件的方式进行通知。镜像逻辑变化可能会导致原有自定义镜像新建节点失败，您需要重新制作自定义镜像。如集群有使用节点池，需调整节点池的镜像配置。
- 如需使用自定义镜像功能，请 提交工单 申请。

注意事项

制作自定义镜像前，请务必仔细阅读以下注意事项。自定义镜像属于非标准环境，腾讯云不提供官方支持及持续维护。
- 请勿随意修改 `/etc/fstab` 。
- **制作镜像之后请及时清理 `/var/lib/cloud` 目录**。
- 如果您在自定义镜像中预装了运行时组件，节点初始化无法正常进行，会直接报错。
- 您在制作自定义镜像前**务必通过 清理脚本 完成节点预清理**，避免当前机器存在历史脏数据而影响节点的正常初始化流程。**注意不要在集群已有节点内执行，否则会导致节点不可用。**执行命令如下：

```
curl --proto '=https' --tlsv1.2 -sSf https://mirrors.tencent.com/install/tke/clean-node.sh | bash
```

> **注意：**
> - 请勿使用集群中正在运行的普通节点（CVM实例）制作自定义镜像，如需使用请先从集群移出。
> - 请使用 TKE 提供的基础镜像来制作自定义镜像。

操作步骤

本文以使用基础镜像创建云服务器（CVM）为例。

1. 创建 CVM

1. 登录 云服务器控制台，选择**新建**进入云服务器购买页面。
2. 在**镜像**中选择容器服务基础镜像，这里以公共镜像 TencentOS Server 3.1 为例。如下图所示：

3. 其他选项设置请参考 新建 CVM 实例 。

2. 创建自定义镜像

1. 请参考 使用标准登录方式登录 Linux 实例（推荐） 登录 CVM。
2. 执行以下命令，新建 `test.txt` 文件。

```
vi test.txt
```

3. 按 i 进入编辑模式，并写入以下内容。

```
this is customer cvm images test
```

4. 按 Esc 并输入 `:wq` 退出并保存。
5. 请参考 创建自定义镜像 完成创建。

3. 使用自定义镜像

> **注意：**
> 您创建（共享）的自定义镜像所在地域需和 TKE 集群所在地域保持一致。

自定义镜像制作完成后，即可使用该镜像创建 TKE 集群。在**创建集群**页面的**镜像提供方**中选择**自定义镜像**，**操作系统**则选择已创建的自定义镜像。如下图所示：

其他选项设置请参考 创建集群 。

4. 验证自定义镜像

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 选择使用自定义镜像创建的集群 ID，进入集群详情页。
3. 在**节点管理**页面，选择**节点**，记录需要登录验证的节点 ID。
4. 登录 云服务器控制台 ，在搜索框中输入记录的节点 ID 并单击🔍，即可看到已创建的节点。如下图所示：

5. 请参考 使用标准登录方式登录 Linux 实例（推荐） ，登录节点。
6. 执行以下命令，验证自定义镜像。

```
cat test.txt
```

返回结果如下，则表示该节点已使用自定义镜像。

```
[root@VM_0_118_centos ~]# cat test.txt
this is customer cvm images test
[root@VM_0_118_centos ~]#
```

使用总结

- 制作自定义镜像必须使用 TKE 提供的基础镜像，没有满足此要求的自定义镜像在 TKE 控制台上不会显示。
- 若在自定义镜像中对 `/etc/resolv.conf` 文件设置文件保护（ `chattr +i /etc/resolv.conf` ），则会导致 `cloud-init` 失败。由于 TKE 依赖于 `cloud-init` 成功状态，最终会导致节点加入集群失败。
- 由于 `rc.local` 和 `container_cluster_agent` 无法保证执行顺序，会导致用户 `start_init.sh` 脚本拷贝的数据丢失。不建议您把 `start_init.sh` 放在 `rc.local` 中执行，建议在 `user-data` 中执行。
- 曾经制作过镜像的节点上如果保留了 `/var/lib/cloud` 目录，那么 `/var/lib/cloud/instances/${instance-id}/sem` 目录下的 `config_scripts_user` 文件会影响 `cloud-init` 服务的正常执行，会导致该节点加入 TKE 集群内时修改的节点主机名无法生效。
- 在自定义镜像中添加个人 yum 源时，若放置在不合适的目录下（例如 `/etc/yum.repos.d/` ），则会引起 `container-cluster-agent` 在执行 `yum install` 操作时报错，从而跳过该步骤，导致 agent 安装 yum 源失败。

常见问题

TKE 控制台看不到自定义镜像的可能原因：

1. 您选择的基础镜像在 CVM 控制台不可见，该场景需 提交工单 申请。
2. 您选择的基础镜像不在 TKE 支持的**公共镜像**列表内，镜像列表请参见 镜像概述 。
3. 您创建（共享）的自定义镜像所在地域和集群地域不一致。

Worker 节点介绍
节点概述

最近更新时间：2024-02-27 11:40:02

简介

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

节点分类

TKE 标准集群支持以下4种不同的节点类型，您可根据业务需要选择不同的节点类型：

节点类型	特点	适用场景
普通节点	适配腾讯云 CVM 数十种机型实例。 基于腾讯云弹性伸缩服务提供自动缩容功能。	用户对资源和运维的管控能力强，操作系统偏定制化。
原生节点	搭载 TKE Insight 可视化资源大盘，助力提升资源利用率。 专有调度器助力节点均衡负载、提升装箱、规整业务。 提供基础设施声明式 API，像管理 workload 一样管理节点。	业务有降本诉求，提升业务资源利用率，简化运维。
超级节点	先进的 Serverless 理念与技术，运维工作轻量化。 单 Pod 独占轻量虚拟机，强隔离无干扰，安全稳定。 秒级扩缩容，轻松应对弹性需求，保障业务降低成本。	弹性业务、隔离性要求高、轻量运维。
注册节点	IDC 资源接入云端管理，实现本地资源利旧。 云下云上资源混合调度部署，无需引入多集群管理。 支持日志、监控、事件、安全等云原生能力，享受一致运维体验。	云上云下资源统一管理。

节点相关操作

- 驱逐或封锁节点
- 设置节点的启动脚本
- 设置节点 Label
- 移出节点

节点池概述

最近更新时间：2024-07-29 10:54:51

为了帮助您高效管理 Kubernetes 集群中的节点，腾讯云容器服务 TKE 引入了节点池的概念。节点池是一组具有相似属性的节点的逻辑集合，一个集群中可以创建多个不同配置和类型的节点池。节点池的配置包含节点的资源属性和 Kubernetes 属性，如计费模式、规格、可用区、标签、污点等。通过节点池的基本功能，您可以方便快捷地创建、管理和销毁节点，从而实现更灵活和高效的资源管理。

应用场景

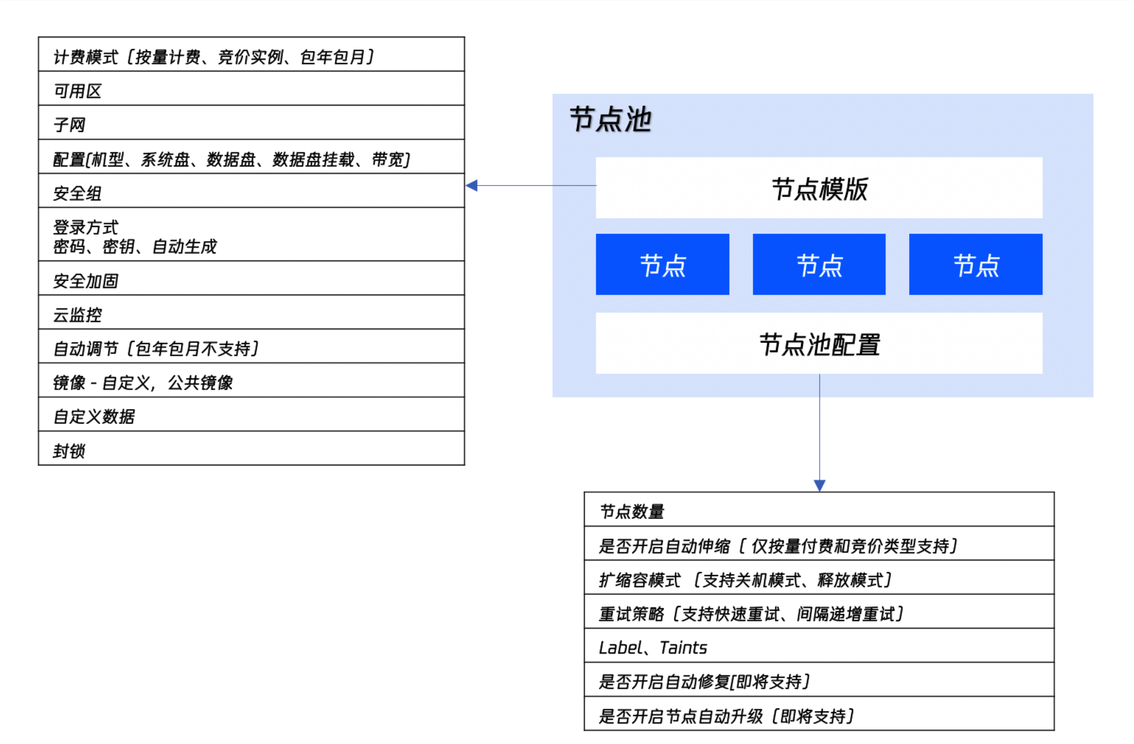
在需要使用大规模集群的业务场景中，我们推荐您使用节点池进行节点管理，以提高大规模集群的易用性。以下是节点池在各种大规模集群管理场景中的应用：

1. **批量管理**：节点选型经过资源、性能测算，将业务部署在一批配置相同的节点上。
2. **弹性伸缩**：节点池内的节点承载相同的业务，在业务高峰或低峰时，可以自动扩容或缩容机器。
3. **指定调度**：通过节点池标签，可以快速指定业务调度规则。
4. **物理隔离**：同集群下通过不同节点池区分业务资源和测试资源。

产品架构

通常情况下，节点池内的节点都具有以下相同的属性：

- **计费类型**：目前支持按量计费、竞价实例和包年包月。
- **操作系统**：节点的操作系统类型。
- **硬件规格**：包括 CPU、内存和 GPU 规格。
- **启动参数**：节点 Kubernetes 组件的启动参数。
- **启动脚本**：节点的自定义启动脚本。
- **标签和 Taint 设置**：节点 Kubernetes 的 Label 和 Taint 设置。



相关文档

节点池弹性伸缩原理

节点生命周期

最近更新时间：2024-02-27 11:40:02

节点生命周期状态说明

状态	说明
健康	节点正常运行，并连接上集群。
异常	节点运行异常，未连接上集群。
已封锁	节点已被封锁，不允许新的 Pod 调度到该节点。
驱逐中	节点正在驱逐 Pod 到其他节点。
其他状态	普通节点请参考 云服务器生命周期 。

节点资源预留说明

最近更新时间：2024-02-27 11:40:02

TKE 需要占用节点一定的资源来运行相关组件（例如：kubelet、kube-proxy、Runtime 等），因此会造成**节点资源总数与集群中可分配资源数**存在差异。本文介绍 TKE 集群中的节点资源预留策略和注意事项，以便在部署应用时合理设置 Pod 的请求资源量和限制资源量。

> **说明：**
> 本节节点资源预留的计算策略仅针对**普通节点**和**原生节点**生效。

节点可分配资源计算策略

计算公式

ALLOCATABLE = CAPACITY – RESERVED – EVICTION – THRESHOLD
默认情况下，除 RESERVED 外为100MiB，即 ALLOCATABLE = CAPACITY – RESERVED – 100MiB。

节点 CPU 预留规则

节点 CPU	预留规则	说明
1c ⩽ CPU ⩽ 4c	固定预留 0.1c	–
4c < CPU ⩽ 64c	4c 以下预留 0.1c，超过 4c 部分预留 2.5%	例如：CPU = 32c预留资源 = 0.1 + (32 – 4) * 2.5% = 0.8c
64c < CPU ⩽ 128c	4c 以下预留 0.1c，4c~64c 预留 2.5%，超过 64c 部分预留 1.25%	例如：CPU = 96c预留资源 = 0.1 + (64 – 4) * 2.5% + (96 – 64) * 1.25% = 2c
CPU > 128c	4c 以下预留 0.1c，4c~64c 预留 2.5%，64c~128c 预留 1.25%，超过 128c 部分预留 0.5%	例如：CPU = 196c预留资源 = 0.1 + (64 – 4) * 2.5% + (128 – 64) * 1.25% + (196 – 128) * 0.5%= 2.74c

节点内存预留规则

节点内存	预留规则	说明
1G ⩽ 内存 ⩽ 4G	固定预留 25%	例如：内存 = 2G预留资源 = 2 * 25% = 512MB
4G < 内存 ⩽ 8G	4G 以下预留 25%，超过 4G 部分预留 20%	例如：内存 = 8G预留资源 = 4 * 25% + (8 – 4) * 20% = 1843MB
8G < 内存 ⩽ 16G	4G 以下预留 25%，4G~8G 预留 20%，超过 8G 部分预留 10%	例如：内存 = 12G预留资源 = 4 * 25% + (8 – 4) * 20% + (12 – 8) * 10%= 2252MB
16G < 内存 ⩽ 128G	4G 以下预留 25%，4G~8G 预留 20%，8G~16G 预留 10%，超过 16G 部分预留 6%	例如：内存 = 32G预留资源 = 4 * 25% + (8 – 4) * 20% + (16 – 8) * 10% + (32 – 16) * 6% = 3645MB
内存 > 128G	4G 以下预留 25%，4G~8G 预留 20%，8G~16G 预留 10%，16G~128G 预留 6%，超过 128G 部分预留 2%	例如：内存 = 320G预留资源 = 4 * 25% + (8 – 4) * 20% + (16 – 8) * 10% + (128 – 16) * 6% + (320 – 128) * 2% = 13475MB

> **说明：**
> 用户可以通过自定义 kubelet 参数的方式来修改 `kube-reserved` 以达到修改节点预留资源的目的，建议给节点组件预留充足的 CPU 和内存资源来保证节点的稳定性。

查看节点可分配资源

检查集群中可用的节点可分配资源，请执行以下命令，并将 NODE_NAME 替换为您要检查的节点的名称。输出结果包含 `Capacity` 和 `Allocatable` 字段，并提供了针对 CPU、内存和临时存储的测量结果。

```
kubectl describe node NODE_NAME | grep Allocatable -B 7 -A 6
```

注意事项

- 该预留策略针对 k8s 1.16及以上版本、2022年6月24日后创建的新增节点自动生效，无需手动配置。
- 为保证业务稳定性，该预留策略不会对已有节点自动生效。因为该资源预留的计算方式可能会造成节点的可分配资源变少，对于资源水位较高的节点，可能会触发节点驱逐。
- 若您希望对已有节点应用该资源预留策略，您可以通过 容器服务控制台 将该节点在不直接销毁的情况下移出集群，然后添加已有节点，新添加的节点会默认执行该资源预留策略。

节点资源预留算法（新）

最近更新时间：2024-09-29 16:49:41

TKE 需要占用节点的一部分资源来运行相关组件（例如：kubelet、kube-proxy、Runtime 等），因此节点的总资源与集群中可分配的资源之间会存在差异。本文主要介绍 TKE 集群中 Kubernetes 版本在1.30及以上节点的新资源预留算法，帮助用户在部署应用时合理设置 Pod 的请求资源量和限制资源量。

适用范围

本算法适用于**集群 kubernetes 版本在1.30及以上的普通和原生节点。**

定义

整机使用资源 = 维持节点运行的系统进程占用资源 + Pod 运行占用资源

> **注意：**
> 本文主要讨论“维持节点运行的系统进程占用资源”的预留算法。

节点 CPU 预留规则

由于 CPU 是可压缩资源且存在部分 perCPU 的内核线程，我们尽量保持算法的一致性，并在大规格机型上提供更多的资源。更新后的算法如下：
- 第一个核心的6%。
- 下一个核心（最多2个核心）的1%。
- 接下来2个核心（最多4个核心）的0.5%。
- 4个以上任何核心数量的0.25%。

需注意，在小规格机器上创建过多的 Pod 可能会导致预留的 CPU 过少，从而影响系统稳定性。以下行为也可能会占用系统资源，可以使用自定义参数适当调整预留的资源情况：
- 容器打印过多日志（容器日志通过 pipe 交给 containerd，最终由 kubelet 压缩落盘）。
- exec probe 执行过于频繁（如每秒一次，exec probe 通过 runc 在容器中执行命令，因此可能会占用大量的 CPU）。
- 在节点上部署其他服务（非 Pod 管理的服务会占用节点预留资源，从而导致系统不稳定）。

新旧算法在 CPU 资源占用情况的对比如下：

CPU（核）	旧算法	新算法
1	0.1	0.06
2	0.1	0.07
4	0.1	0.08
8	0.2	0.09
16	0.4	0.11
32	0.8	0.15
64	1.6	0.23
128	2.4	0.39
256	3.04	0.71

节点 Mem 预留规则

Memory 是不可压缩资源，设置预留内存时应较为谨慎。经过测试内存和 Pod 数量、机器规格紧密相关，调整新算法为：`min(旧算法, 20MiB * Pod 数量 + 256MiB)`。注意在节点上部署其他服务也可能会占用节点预留资源，导致节点稳定性降低。如果需要部署非 Pod 管理的服务，建议调整预留资源。

新旧算法在 Mem 资源占用情况的对比如下：

内存大小（GiB）	旧算法	16个 pod（GiB）	32个 pod（GiB）	64个 pod（GiB）	128个 pod（GiB）	256个 pod（GiB）

1	0.25	0.25	0.25	0.25	0.25	0.25
2	0.5	0.5	0.5	0.5	0.5	0.5
4	1	0.58	0.9	1	1	1
8	1.8	0.58	0.9	1.54	1.8	1.8
16	2.6	0.58	0.9	1.54	2.6	2.6
32	3.56	0.58	0.9	1.54	2.82	3.56
64	5.48	0.58	0.9	1.54	2.82	5.38
128	9.32	0.58	0.9	1.54	2.82	5.38
256	11.88	0.58	0.9	1.54	2.82	5.38

常见问题

新旧算法有什么差异?

- 旧算法：根据机器规格为资源预留固定值，详情请参见 资源预留说明 。算法相对保守，预留资源量较大，用户可以给业务使用的资源量较低。
- 新算法：通过调整 Pod 数在不同规格机器上进行大规模压测获得资源预留公式，新算法在大规格机型上提供更多的资源给业务使用。

关于设置 kube-reserved 且不设置 system-reserved 的解释

kubelet 将 `capacity - kubeReserved - systemReserved  - evictionHard` 作为节点可分配资源，用来控制 Pod 可用到的资源量。按照文档，kubeReserved 可用来限制 kubelet 和运行时组件的资源使用量、systemReserved 可用来限制系统服务资源量，但生效的前提是需要开启 enforce-node-allocatable 并指定 kube-reserved-cgroup 和 system-reserved-cgroup ，这对系统的 cgroup 布局有额外要求。同时，社区官方和众多云厂商出于对整体稳定性考量，并不会启用这两种限制。因此 TKE 也不会对 kubelet 组件和系统组件进行资源限制，同样 reserved 的资源完全设置在 kube-reserved 上还是 kube-reserved 和 system-reserved 各占一部分实际的效果没有区别。

新算法是否支持对低版本节点生效?

暂不支持。存量节点后续可以通过节点升级来体验新算法。

新增节点

最近更新时间：2024-02-27 11:40:02

操作场景

本文档指导您在集群内新增 Worker 节点，当前操作场景下新增的节点将通过节点池统一管理。

操作步骤

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在集群列表中，单击需要新增节点的集群 ID，进入集群的基本信息页面。
3. 在**节点管理** > **Worker 节点**页面，选择**节点池/节点**页签，单击**新建**。
4. 在选择节点类型页面，选择要创建的目标节点类型，如下图所示：

5. 单击**创建**，即可进入节点创建页面。

移出节点

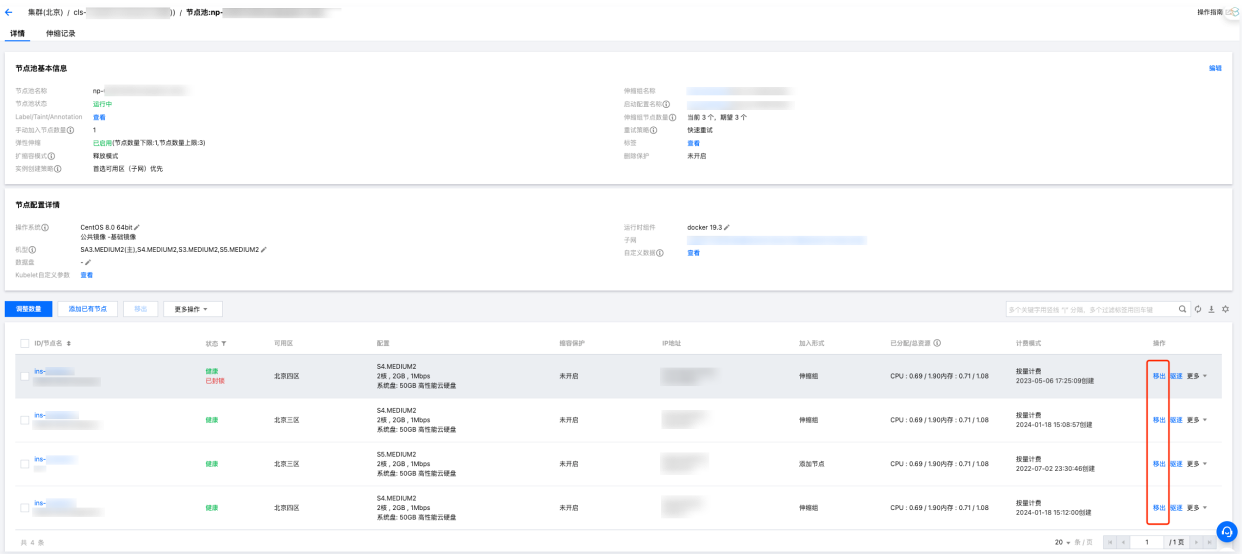
最近更新时间：2024-02-27 11:40:02

操作场景

本文档指导您移出集群下的 Worker 节点。

操作步骤

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在集群列表中，单击需要移出节点的集群 ID，进入集群的基本信息页面。
3. 在**节点管理** > **Worker 节点**页面，选择**节点池**页签，并选择目标节点池进入详情页。
4. 在节点池详情页的节点列表中，选择需要移出节点的节点行，单击**移出**。如下图所示：



5. 在弹窗中确认已选择的节点信息后，单击**确定**，即可完成移出。

驱逐或封锁节点

最近更新时间：2025-03-06 16:50:32

操作场景

本文档指导您如何驱逐或封锁节点。

操作步骤

封锁节点

封锁（cordon）节点后，将不接受新的 Pod 调度到该节点，您需要手动取消封锁的节点。封锁节点后，如果节点之前已被 CLB 绑定作为后端目标节点，节点将从目标节点列表中移除。封锁节点有以下两种方法：

方法一

新增节点 时，在云服务器配置页面，勾选 “开启封锁”。如下图所示：

方法二

1. 登录 容器服务控制台，选择左侧导航中的**集群**。
2. 在集群列表中，单击集群 ID，进入集群管理页面。
3. 选择**节点管理**，在节点页签中单击节点行右侧的**更多 > 封锁**。如下图所示：

4. 在弹出的对话框中，确认节点信息后，单击**确定**，即可完成封锁。

取消封锁节点

取消封锁（uncordon）节点后，将允许新的 Pod 调度到该节点。取消封锁有以下两种方法：

方法一

通过执行脚本的方式新增节点时，您可以在该脚本中添加取消封锁节点的命令，即可取消封锁。其示例如下：

```
#!/bin/sh
# your initialization script
echo "hello world!"
```

```
# If you set unschedulable when you create a node,
# after executing your initialization script,
# use the following command to make the node schedulable.
node=`ps -ef|grep kubelet|grep -oE 'hostname-override=\S+'|cut -d"=" -f2`
#echo ${node}
kubectl uncordon ${node} --kubeconfig=/root/.kube/config
```

`kubectl uncordon` 命令表示取消封锁节点。

方法二

1. 登录 容器服务控制台，选择左侧导航中的集群。
2. 在集群列表中，单击集群 ID，进入集群管理页面。
3. 选择**节点管理**，在节点页签中单击节点行右侧的**更多 > 取消封锁**。
4. 在弹出的对话框中，单击**确定**，即可完成取消封锁。

驱逐节点

概述

在节点上执行维护之前，您可以通过驱逐（drain）节点安全地从节点中逐出 Pod。节点驱逐后，自动将节点内的所有 Pod（不包含 DaemonSet 管理的 Pod）驱逐到集群内其他节点上，并将驱逐的节点设置为封锁状态。

> **注意：**
> 本地存储的 Pod 被驱逐后数据将丢失，请谨慎操作。

操作方法

1. 登录 容器服务控制台，选择左侧导航中的**集群**。
2. 在集群列表中，单击集群 ID，进入集群管理页面。
3. 选择**节点管理**，在节点页签中单击节点行右侧的**更多 > 驱逐**。如下图所示：

4. 在弹出的对话框中，单击**确定**，即可完成驱逐。

设置节点的启动脚本

最近更新时间：2024-03-04 16:24:32

操作场景

设置节点的启动脚本可以帮助您在节点 ready 之前，对您的节点进行初始化工作，即当节点启动的时候运行配置的脚本，如果一次购买多台云服务器，自定义数据会在所有的云服务器上运行。

使用限制

- 建议您不要通过启动脚本修改 TKE 节点上的 kubelet、kube-proxy、docker 等配置。
- 启动脚本执行失败不重试，需自行保证脚本的可执行性和重试机制。
- 脚本及其生成的日志文件可在节点的 `/usr/local/qcloud/tke/userscript` 路径查看。

操作步骤

您可以在以下两个场景设置节点的启动脚本：

创建集群或新增节点时

创建集群 及 新增节点 时，在**云服务器配置**或**新建节点**页面，单击**高级设置**，填写自定义数据，启动脚本。如下图所示：

添加已有节点时

添加已有节点 时，在**添加已有节点**页面，单击**高级设置**，填写自定义数据，启动脚本。

设置节点 Label

最近更新时间：2025-02-12 16:37:32

操作场景

本文档指导您设置节点 Label。

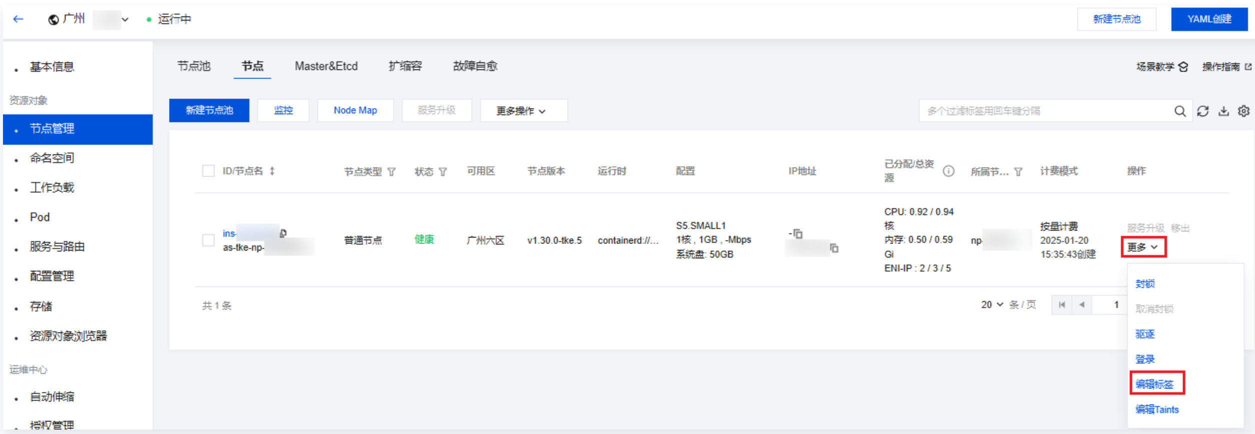
使用限制

- kubernetes 和 qcloud 相关标签禁用编辑和删除。
- kubernetes 和 qcloud 标签为保留键，不支持添加。
- 目前只支持为单个节点设置标签，不支持批量设置。

操作步骤

控制台设置节点 Label

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择**节点管理**，在节点页签中单击节点行右侧的**更多 > 编辑标签**。如下图所示：



4. 在**编辑 Label** 中，根据您的需求设置 Label，设置完成后单击**确定**。

kubectl 设置节点 Label

1. 安装 kubectl，并连接集群。操作详情请参考 通过 kubectl 连接集群 。
2. 执行以下命令，设置节点 Label。

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

3. 执行以下命令，查看节点 Label。

```
kubectl get nodes --show-labels
```

返回类似如下信息：

```
NAME           STATUS    ROLES     AGE       VERSION          LABELS
172.17.124.5   Ready     <none>    12d       v1.10.5-tke.3
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-
type=QCLOUD,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=sh,failure-
domain.beta.kubernetes.io/zone=200001,kubernetes.io/hostname=172.17.124.5
```

```
172.17.124.8    Ready     <none>    12d       v1.10.5-tke.3
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-
type=QCLOUD,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=sh,failure-
domain.beta.kubernetes.io/zone=200001,kubernetes.io/hostname=172.17.124.8
```

普通节点管理
普通节点支持的 CVM 机型

最近更新时间：2025-05-21 14:34:42

背景

为更好的提供容器服务，TKE 会针对普通节点所支持的机型实例做容器环境的可用性测试，测试内容主要覆盖容器网络模式、存储、公共镜像、节点初始化、GPU 驱动等多个应用模块，目前容器服务控制台支持创建的普通节点机型实例如下表所示。

> **说明：**
> 容器服务控制台创建普通节点所支持的机型并非和 CVM 控制台一一对应，若您的业务有新机型需要适配可 提交工单 申请。

普通节点支持的机型实例

实例类型	机型
标准型	S1、S2、S2ne、S3、S3ne、S4、S4m、S5、S5se、S5t、S6、S6t、SA1、SA2、SA2a、SA3、SK1、SN3ne、SR1、SW3a、SW3b、SW3ne、SA4、SA5、S8、SA9、S9、SA9e、S9e、S9pro
尊享型	RS2t、RS3t、RS4t、RS5t
计算型	C2、C3、C4、C5、C6、TC3、CN3
高I/O型	I1、I2、I3、I6t、IT2、IT3、IT3a、IT3b、IT3c、IT5、ITA5
内存型	M1、M2、M3、M4、M5、M6、M6ce、M6mp、M6p、MA2、MA3、MA4、MA5
高性能型	HCCG5v、HCCIC5、HCCPNV4h、HCCTG5v、HCCPNV4sne（HCCPNV4sn）、HCCPNV5v、HCCPNV5vp、HCCPNV5、HCCPNV5x、HCCPNV6
GPU 机型	GI1、GI3X、GN10S、GN10X、GN10Xp、GN6、GN6S、GN7、GN8、GNV4、GT4、PNV4、PNV4ne、PNV5、GC49、PNV5b、PNV5i、PNV6
大数据型	D1、D2、D3
裸金属	BMD2、BMD3、BMD3c、BMD3s、BMDA2、BMI5、BMIA2、BMIA2m、BMM5r、BMS4、BMSA2、BMSC4、BMM6i、BMTGC39me（BMGC39me）、BMG5e、BMG5n、BMG5i、BMG5t、BMGY5、BMGNV4、BMSA3、BMIA3、BMS5
其他	CHC、CN10X、BC1、BF、CH2

> **说明：**
> 1. VPC-CNI 集群的容器使用独立网卡功能仅支持部分机型，详情请参见 Pod 间独占网卡模式机型限制 。
> 2. 针对裸金属云服务器，使用前请前往 实例规格 确认机型配置，如是否支持弹性网卡挂载、是否支持云盘挂载。
>   2.1 不支持挂载弹性网卡的机型，VPC-CNI 网络模式的集群无法添加该类普通节点，您可选择 GR 模式。
>   2.2 不支持云盘挂载的机型，Pod 无法绑定 PVC。
> 3. HCC 高性能机型支持的公共镜像有 CentOS 7.6、Ubuntu18.04.1、TencentOS Server 2.4 (TK4)、TencentOS Server 2.4 (TK4) HCC（推荐）、TencentOS Server 3.1 (TK4) UEFI、Ubuntu Server 20.04 LTS (TK4) 64位 UEFI版。由于该类机型对操作系统要求较高，建议 提交工单 联系 CVM 售后根据具体机型提供推荐镜像。
> 4. ARM 类机型 SR1、SK1 仅支持 TencentOS Server 2.4 for ARM 64 (TK4)、CentOS 8.2(ARM) 镜像。
> 5. 海光机型 CH2 仅支持 TencentOS 系列镜像。
> 6. 消费级卡机型（如 GC49）需要自行安装驱动，您可在创建节点时指定驱动安装脚本或使用预装了驱动的自定义镜像，否则节点初始化可能会因检测不到驱动而失败。
> 7. Red Hat 镜像仅支持机型 SA2、S5、C3、C4。
> 8. **qGPU 功能仅针对 原生节点 开放**，目前支持 T4、v100、L20、L40、L40s 卡机型，详情请参见 使用 qGPU 。

节点池概述

最近更新时间：2025-06-05 18:40:32

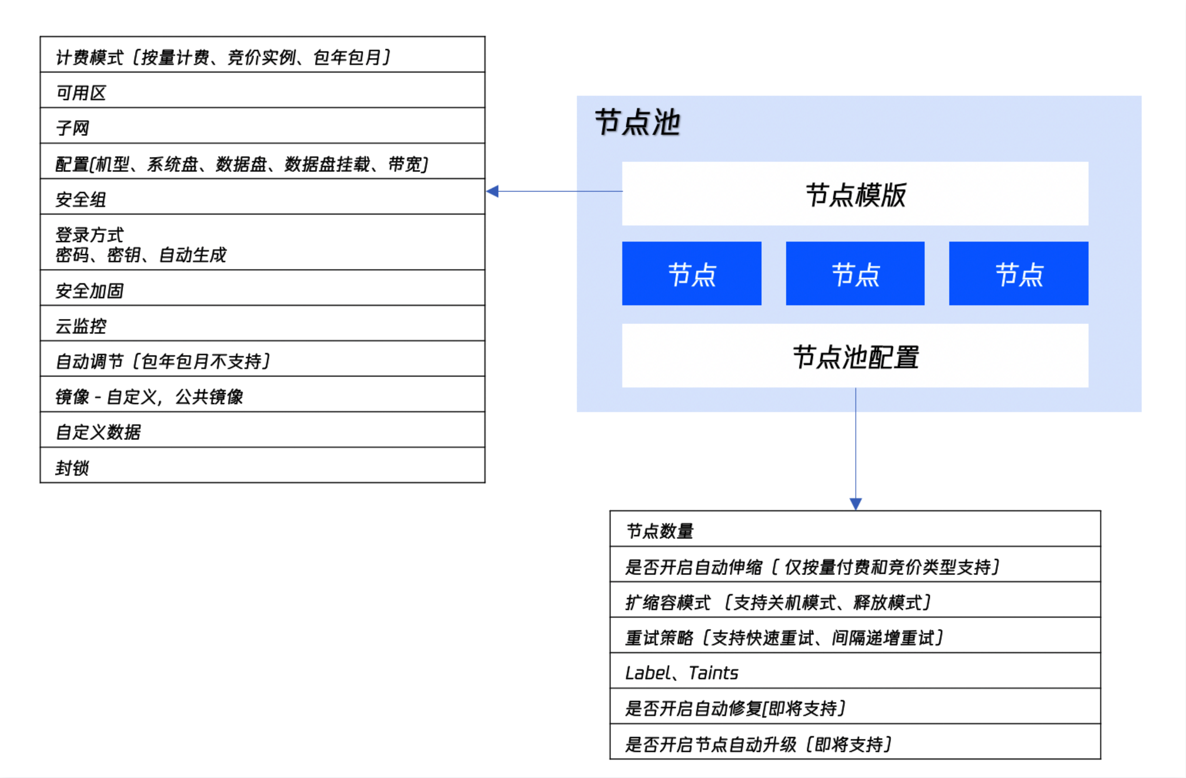
简介

为帮助您高效管理 Kubernetes 集群内节点，腾讯云容器服务 TKE 引入节点池概念。借助节点池基本功能，您可以方便快捷地创建、管理和销毁节点，以及实现节点的动态扩缩容：

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

产品架构

节点池整体架构图如下所示：



通常情况下，节点池内的节点均具有如下相同属性：

- 节点操作系统。
- 计费类型（目前支持按量计费、竞价实例和包年包月）。
- CPU/内存/GPU。
- 节点 Kubernetes 组件启动参数。
- 节点自定义启动脚本。
- 节点 Kubernetes Label 和 Taint 设置。

此外，TKE 将同时围绕节点池扩展以下功能：

- 支持用 CRD 管理节点池。
- 节点池级别每节点的 Pod 数上限。
- 节点池级别自动修复与自动升级。

应用场景

当业务需要使用大规模集群时，推荐您使用节点池进行节点管理，以提高大规模集群易用性。下表介绍了多种大规模集群管理场景，并分别展示节点池在每种场景下发挥的作用：

场景	作用

集群存在较多异构节点（机型配置不同）	通过节点池可规范节点分组管理。
集群需要频繁扩缩容节点	通过节点池可提高运维效率，降低人力成本。
集群内应用程序调度规则复杂	通过节点池标签可快速指定业务调度规则。
集群内节点日常维护	通过节点池可便捷操作 Kubernetes 版本升级、Docker 版本升级。

相关概念

TKE 的弹性伸缩实现是基于腾讯云弹性伸缩（AutoScaling）以及 Kubernetes 社区的 cluster-autoscaler 实现的。相关概念介绍：

- CA：cluster-autoscaler，社区开源组件，主要负责集群的弹性扩缩容。
- AS：AutoScaling，腾讯云弹性伸缩服务。
- ASG：AutoScaling Group，具体某个节点池（节点池依赖弹性伸缩服务提供的伸缩组，一个节点池对应一个伸缩组，您只需关心节点池）。
- ASA：AS activity，某次伸缩活动。
- ASC：AS config，AS 启动配置，即节点模板。

节点池内节点种类

为了满足不同场景下的需求，节点池内的节点可以分为两个类型。

> **说明：**
> 无特殊场景不推荐您使用添加已有节点功能，例如您没有新建节点的权限仅能通过添加已有节点来扩容集群，添加已有节点部分参数可能会与您定义的节点的模板不一致，将无法参与弹性伸缩。

节点类型	节点来源	是否支持弹性伸缩	从节点池移除方式	节点数目是否受**调整数量**影响
伸缩组内节点	弹性扩容或手动调整数量	是	弹性缩容或手动调整数量	是
伸缩组外节点	用户手动加入节点池	否	用户手动移除	否

节点池弹性伸缩原理

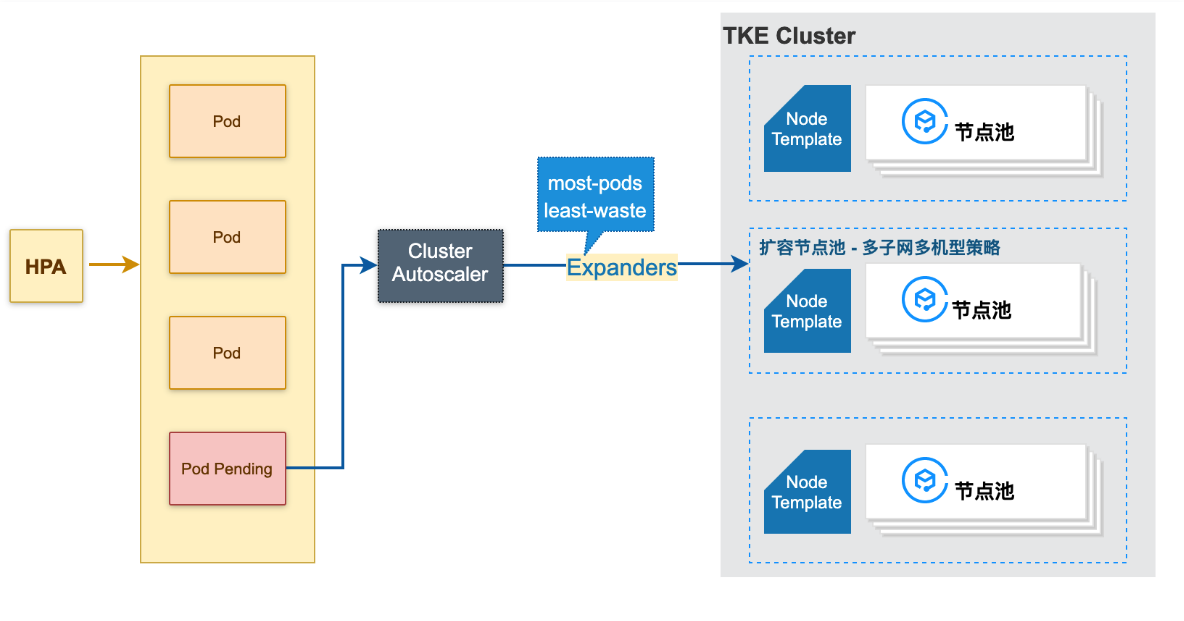
在您使用节点池弹性伸缩功能前，请阅读以下原理说明。
弹性伸缩相关参数配置请参见 Cluster Autoscaler 说明 。

节点池弹性扩容原理

1. 当集群中资源不足时（集群的计算/存储/网络等资源满足不了 Pod 的 request /亲和性规则），CA（Cluster Autoscaler）会监测到因无法调度而 Pending 的 Pod。
2. CA 根据每个节点池的节点模板进行调度判断，挑选合适的节点模板。
3. 若有多个模板合适，即有多个可扩的节点池备选，CA 会调用 **expanders** 从多个模板挑选最优模板并对对应节点池进行扩容。
4. 对指定节点池进行扩容 （根据多子网多机型策略），并且提供两种重试策略（可在创建节点池设置），在扩容失败时根据您设定的重试策略进行重试。

> **说明：**
> 对特定节点池扩容时，会根据您创建节点池设置的子网以及后续设置的多机型配置来进行扩容。一般情况下会**先保证多机型的策略，后保证多可用区/子网的策略。**
> 例如您配置了多机型 A、B，多子网1、2、3，会按照 A1、A2、A3、B1、B2、B3 进行尝试，如果A1售罄，会尝试 A2，而不是 B1。

节点池弹性扩容原理如下图所示：



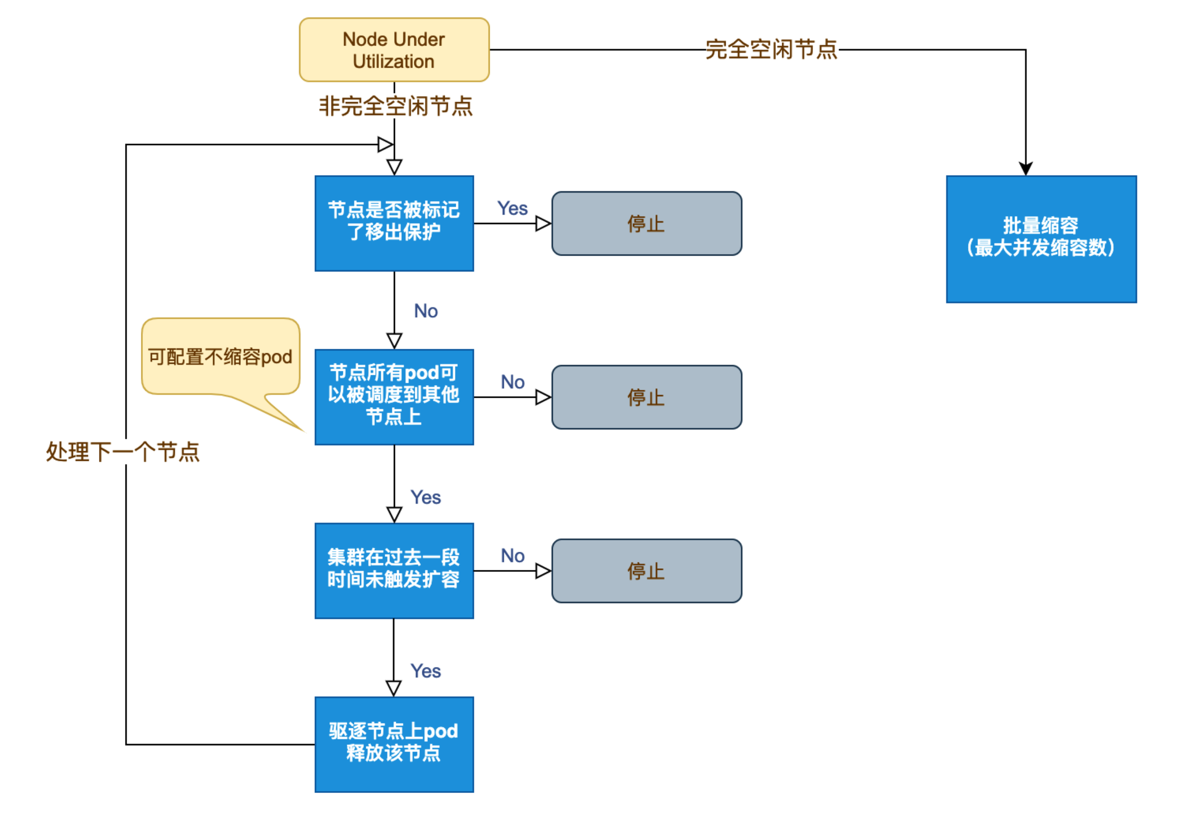
节点池弹性缩容原理

1. CA（Cluster Autoscaler）监测到分配率（即 Request 值，取 CPU 分配率和 MEM 分配率的最大值）低于设定的节点。计算分配率时，可以设置 Daemonset 类型不计入 Pod 占用资源。
2. CA 判断集群的状态是否可以触发缩容，需要满足如下要求：
    - 节点空闲时长要求（默认10分钟）。
    - 集群扩容缓冲时间要求（默认10分钟）。
3. CA 判断该节点是否符合缩容条件。您可以按需设置以下**不缩容条件**（满足条件的节点不会被 CA 缩容）：
    - 含有本地存储的节点。
    - 含有 Kube-system namespace 下非 DaemonSet 管理的 Pod 的节点。

    > **说明：**
    > 上述不缩容条件在集群维度生效，若您需要更细粒度的保护节点免于缩容，可以使用**缩容保护**功能。

4. CA 驱逐节点上的 Pod 后释放/关机节点（不会处理包年包月节点）。
    - 完全空闲节点可并发缩容（可设置最大并发缩容数）。
    - 非完全空闲节点逐个缩容。

节点池弹性缩容原理如下图所示：



功能点及注意事项

功能点	功能说明	注意事项
创建节点池	新增节点池	• 单个集群不建议超过20个节点池。 • 计费模式为节点池维度属性，包年包月节点池详情可参见 创建包年包月节点池说明。请勿将节点池内按量计费节点转成包年包月节点，建议您新建包年包月节点池。
删除节点池	• 删除节点池时可选择是否销毁节点池内节点。 • 无论是否销毁节点，节点都不会保留在集群内。	删除节点池时选择销毁节点，节点将不会保留，后续如需使用新节点可重新创建。
节点池开启弹性伸缩	开启弹性伸缩后，节点池内节点数量将随集群负载情况自动调整。	请勿在伸缩组控制台开启和关闭弹性伸缩。
节点池关闭弹性伸缩	关闭弹性伸缩后，节点池内节点数量不随集群负载情况自动调整。	
调整节点池节点数量	• 支持直接调整节点池内节点数量。 • 若减小节点数量，将按节点移出策略从伸缩组内缩容节点。请注意：该缩容动作由伸缩组执行，TKE 无法感知具体缩容节点，无提前驱逐/封锁动作。	• 开启弹性伸缩后，不建议手动调整节点池大小。 • 请勿在伸缩组控制台直接调整伸缩组期望实例数。 • 无特殊情况，请勿手动缩容节点池，请使用弹性缩容：弹性缩容时会首先将节点标记为不可调度，随后驱逐或者删除节点上所有 Pod 后再释放节点。
调整节点池配置	可修改节点池名称、操作系统、伸缩组节点数量范围、Kubernetes label 及 Taint。	修改 Label 和 Taint 会对节点池内节点全部生效，可能会引起 Pod 重新调度，请谨慎变更。

添加已有节点	• 可添加不属于集群的实例到节点池。要求如下：
	  ○ 实例与集群属于同一私有网络。
	  ○ 实例未被其他集群使用且实例与节点池配置相同机型、相同计费模式。
		无特殊情况时，不建议添加已有节点，推荐直接新建节点池。

	• 可添加集群内不属于任何节点池的节点，要求节点实例与节点池配置相同机型、相同计费模式。	
移出节点池内节点	支持移出节点池内任意节点，移出时节点可选择是否保留到集群。	请勿在伸缩组控制台往伸缩组内加入节点，可能会导致数据不一致的严重后果。
原伸缩组转换节点池	• 支持存量伸缩组切换为节点池。转化后，节点池完全继承原伸缩组的功能，该伸缩组信息将不再展示。 • 集群内存量所有伸缩组切换完成后，不再提供伸缩组入口。	操作不可逆，请熟悉节点池功能后再进行切换。

相关操作

您可以登录 容器服务控制台 并参考以下文档， 进行对应节点池操作：

- 创建节点池
- 查看节点池
- 调整节点池
- 删除节点池

创建节点池

最近更新时间：2024-12-06 14:19:42

操作场景

本文介绍如何通过容器服务控制台在集群中创建节点池，并提供了节点池相关操作，例如查看、管理及删除节点池。

前提条件

- 已了解 节点池基本概念 。
- 已 创建集群 。

说明事项

包年包月节点池

包年包月节点池里的节点为预付费机器，因此在弹性伸缩能力会有一定程度上的限制，说明如下：

- 包年包月节点池支持弹性扩容。
- 包年包月节点池不支持弹性缩容（弹性缩容具备一定的随机性，推荐您通过**手动移出**节点管理预付费机器）。
- 将包年包月节点池内节点移出节点池，可以选择是否保留在集群内，容器服务 TKE **不会为您销毁包年包月节点**，请将节点移出集群后自行前往云服务器控制台销毁，详情可参见 包年包月实例退费说明 。

操作步骤

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点池**。单击**新建**，并选择节点类型。
4. 在**节点启动配置**中，参考以下提示进行设置：

- **节点池名称**：自定义，可根据业务需求等信息进行命名，方便后续资源管理。
- **操作系统**：根据实际需求进行选择。该操作系统节点池维度生效，支持更改。更改后新操作系统只对节点池内增量节点生效，不会影响存量节点。
- **计费模式**：提供**按量计费**、**竞价计费**、**包年包月**三种计费模式，请根据实际需求进行选择。详情请参见 计费模式对比 。

> **注意：**
> 包年包月类型的节点池不支持弹性缩容，可通过手动调整节点池下节点数目来扩容。

- **支持网络**：系统将为集群内主机分配在节点网络地址范围内的 IP 地址。

> **注意：**
> 该选项为集群维度设置项，故不支持修改。

- **机型配置**：单击**请选择机型**，在机型配置页面根据以下信息按需选择：
  - **可用区**：启动配置里不包含可用区信息，该选项仅用于过滤所选可用区下可用实例类型。
  - **机型**：支持通过 CPU 核数、内存大小及实例类型进行筛选。详情请参见 实例规格 和 快速入门 Linux 云服务器 。
- **系统盘**：存储控制、调度云服务器运行的系统集合。支持查看所选机型的可选系统盘类型，请参考 云硬盘类型 并根据实际需求进行选择。
- **数据盘**：用于存储所有的用户数据。请根据以下指引进行设置。每种机型所对应的数据盘设置不尽相同，请参考以下表格进行设置：

机型	数据盘设置
标准型、内存型、计算型、GPU 机型	默认不勾选。若勾选，请根据实际情况进行云硬盘设置及格式化设置。
高 IO 型、大数据型	默认勾选且不可更改，支持对默认购买的本地盘进行自定义格式化设置。

批量型	默认勾选且支持取消勾选，勾选时仅支持购买默认本地盘，支持对默认本地盘进行自定义格式化设置。

> **说明：**
> 节点上挂载的首块数据盘（仅考虑云硬盘类型）格式化挂载路径默认为 /var/lib/container，此时系统将 “Kubelet/容器/镜像”的存储目录（包括 /var/lib/kubelet、 /var/lib/containerd、 /var/lib/docker）都默认挂载到该路径下。节点上的其他数据盘，您可以自定义该盘的挂载路径。

- **添加数据盘（可选）**：单击**添加数据盘**，并参考上表进行设置。
- **公网宽带**：默认勾选**分配免费公网 IP**，系统将免费分配公网 IP。支持按使用流量、按带宽计费两种模式，请参考 公网计费模式 根据实际情况进行选择，并进行网速自定义设置。
- **登录方式**：提供以下三种登录方式，请根据实际情况进行选择。
  - **立即关联密钥**：密钥对是通过算法生成的一对参数，是一种比常规密码更安全的登录云服务器的方式。详情请参见 SSH 密钥。
  - **SSH密钥**：该配置项仅在选择**立即关联密钥**登录方式时出现，在下拉框中选用已有密钥即可。若需新建，请参考 创建 SSH 密钥。
  - **自动生成密码**：自动生成的密码将通过 站内信 发送给您。
  - **设置密码**：请根据提示设置对应密码。
- **安全组**：默认为创建集群时所设置的安全组，可根据实际需要进行更换或添加。
- **支持子网**：请根据实际需求选择合适的可用子网。

> **说明：**
> 节点池默认的多子网扩容策略如下：当您配置了多个子网，节点池扩容时（手动扩容及弹性扩容）将按照子网列表的顺序，作为优先级来尝试创建节点，如果优先级最高的子网可以创建成功，则总在该子网创建。

- **数量**：对应期望实例数量，请根据实际需求进行设置。

> **注意：**
> 若节点池已开启自动伸缩，该数量将会随集群负载自动调整。

5. 在**运维功能设置**中，参考以下提示进行设置：

- **弹性伸缩**：默认勾选**开启**。
- **节点数量范围**：节点数量将在设定的节点范围内自动调节，不会超出该设定范围。
- **实例创建策略**：提供**首选可用区（子网）优先**和**多可用区（子网）打散**。弹性伸缩会在您首选的可用区优先执行扩缩容。若首选可用区无法扩缩容，才会在其他可用区进行扩缩容。
- **实例移出策略**：提供**优先移出最新实例**和**优先移出最旧实例**。节点池的移出策略决定了在减少节点数量时，哪些节点会被优先移除。
- **重试策略**：提供以下两种策略，请根据实际需求进行选择。
  - **快速重试**：立即重试，在较短时间内快速重试，连续失败超过一定次数（5次）后不再重试。

    - **间隔递增重试**：间隔递增重试，随着连续失败次数的增加，重试间隔逐渐增大，重试间隔从秒级到1天不等。
  - **扩缩容模式**：提供以下两种扩缩容模式，请根据实际需求进行选择。
    - **释放模式**：缩容时自动释放 Cluster AutoScaler 判断的空余节点， 扩容时自动创建新的节点加入到伸缩组。
    - **关机模式**： 扩容时优先对已关机的节点执行开机操作，节点数依旧不满足要求时再创建新的节点。缩容时将关机空余节点，若节点支持关机不收费则将不收取机型的费用，详情请参见 按量计费实例关机不收费说明，其余节点关机会继续收取费用。

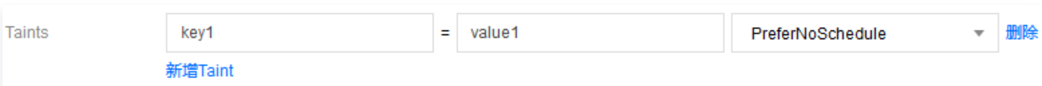
6. （可选）单击**更多设置**，查看或配置更多信息。如下图所示：

  - **删除保护**：开启后可阻止通过控制台或云 API 误删除节点池。
  - **封锁初始节点**：勾选**开启封锁**后，将不接受新的 Pod 调度到该节点，需要手动取消封锁的节点，或在自定义数据中执行 取消封锁命令，请按需设置。
  - **CAM 角色**：可为节点池的所有节点绑定相同的 CAM 角色，从而赋予节点该角色绑定的授权策略。详情请参见 管理实例角色 。
  - **容器目录**：勾选即可设置容器和镜像存储目录，建议存储到数据盘。例如 `/var/lib/docker` 。

- **安全加固**：默认免费开通 DDoS 防护、WAF 和云镜主机防护，详情请参见 T-Sec 主机安全官网页。
- **云监控**：默认免费开通云产品监控、分析和实施告警，安装组件获取主机监控指标，详情请参见 腾讯云可观测平台 TCOP。
- **腾讯云标签**：标签用于从不同维度对资源分类管理。如现有标签不符合您的要求，您亦可前往 标签控制台 统一管理标签。
- **Labels**：单击**新增**，即可进行 Label 自定义设置。该节点池下所创建的节点均将自动增加此处设置的 Label，可用于后续根据 Label 筛选、管理节点。
- **Taints**：节点属性，通常与 `Tolerations` 配合使用。此处可为节点池下的所有节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在的不符合条件的 Pod 也将会被驱逐。

> **说明：**
> Taints 内容一般由 `key`、`value` 及 `effect` 三个元素组成。其中 `effect` 可取值通常包含以下三种：
> - **PreferNoSchedule**：非强制性条件，尽量避免将 Pod 调度到设置了其不能容忍的 taint 的节点上。
> - **NoSchedule**：当节点上存在 taint 时，没有对应容忍的 Pod 一定不能被调度。
> - **NoExecute**：当节点上存在 taint 时，对于没有对应容忍的 Pod，不仅不会被调度到该节点上，该节点上已存在的 Pod 也会被驱逐。

以设置 Taints `key1=value1:PreferNoSchedule` 为例，控制台配置如下图所示：



- **置放群组**：勾选此选项后，您的节点池将具备置放群组的能力。您可以进一步选择合适的置放群组，以便将节点分散到不同的物理机、交换机或机架上。若当前没有合适的置放群组，您可以前往控制台 新建置放群组 新建一个置放群组。通过置放群组，您可以有效降低单点故障的风险，提升系统的整体容灾能力。节点的分散部署确保了在某台物理机或交换机出现故障时，其他节点仍能正常运行，从而保障业务的连续性和稳定性。

- **自定义脚本**：指定自定义数据来配置节点，即当节点启动后运行配置的脚本。需确保脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的 `/usr/local/qcloud/tke/userscript` 路径查看。

7. 单击**创建节点池**。

相关操作

节点池创建完成之后，您可参考以下操作指引进行后续的节点池管理：
- 查看节点池
- 调整节点池
- 删除节点池

查看节点池

最近更新时间：2025-02-27 17:55:12

操作场景

本文介绍如何通过容器服务控制台查看集群中已创建的节点池，并获取节点池的详细信息，以便后续对节点池进行管理。

前提条件

集群下已 创建节点池 。

操作步骤

查看节点池列表页

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点池**页签，查看已创建的节点池。每个节点池以卡片的形式进行展示，主要包含以下信息：
    - **节点池 ID（节点池名称）**：本例为 np-******，单击此 ID 可进入该节点池详情页，查看更多节点池相关信息。
    - **节点池状态**：本例为“运行中”，表示该节点池处于正常状态。
    - **节点池操作**：包含**编辑**、**克隆**、**更多**等，详情请参见 调整节点池配置 。
    - **该节点池下可用节点数/节点总数**：本例为“1可用/共1台”。
    - **主机型**：展示该节点池下所有节点的机型。
    - **计费模式**：展示该节点池下所有节点的计费模式，本例为“按量计费”，表示按照实例的实际使用量进行收费。更多计费详情请参见 计费模式 。
    - **弹性伸缩**：展示节点池是否开启弹性伸缩。
    - **操作系统**：展示该节点池的操作系统。
    - **节点池类型**：展示该节点池的节点池类型。

> **说明：**
> 当节点池较多时，可在该区域右上角的搜索框中输入节点池 ID 或节点池名称进行筛选。

查看单个节点池

1. 单击目标节点池 ID。如下图所示：

2. 进入该节点池详情页，查看节点池更多基本信息及节点信息。如下图所示：

相关操作

您可参考以下文档，了解更多节点池具备的功能：

- 创建节点池
- 调整节点池
- 删除节点池

调整节点池

最近更新时间：2025-07-01 14:42:22

操作场景

本文介绍如何通过容器服务控制台调整节点池配置，包括调整节点池全局配置、节点池配置、节点池下节点数量及启用或停用弹性伸缩、为节点设置缩容保护等操作。

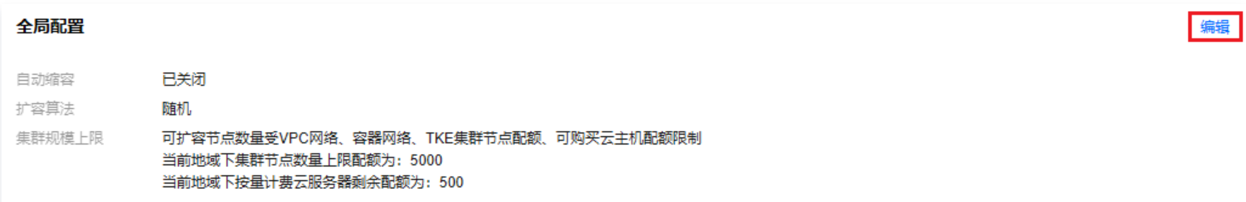
前提条件

- 已创建可用节点池。详情请参见 创建节点池 。
- 已进入节点池列表。详情请参见 查看节点池 。

操作步骤

调整节点池全局配置

1. 登录 容器服务控制台，单击左侧导航栏中的集群。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**扩缩容**。单击“全局配置”模块右上角的**编辑**。如下图所示：



4. 在**更新组件配置**中，参考以下信息进行设置。如下图所示：

主要参数信息如下：
- **自动缩容**：默认不勾选。开启自动缩容时，集群中节点空闲资源较多时将触发缩容。详情请参见 集群自动扩缩容说明 。
- **缩容配置**：该配置项仅在开启自动缩容时显示，请根据实际需求进行设置。
  - **最大并发缩容数**：该数值表示为可以同时进行缩容的节点数，此处默认为“10”，可按需自定义设置。

    > **注意：**
    > 此处只缩容完全空闲的空节点。如果节点上存在 Pod，则每次缩容最多一个节点。

  - **Pod 占用资源/可分配资源小于的值**：可设置 **Pod 占用资源/可分配资源**在占比小于设定值时开始判断缩容条件。占比值范围需确保在0 – 80之间。
  - **节点连续空闲**：可自定义设置节点连续空闲时间超过几分钟之后会被缩容。
  - **集群扩容**：可自定义设置集群首次判断缩容条件的时间点。
  - **不缩容节点**：请根据实际需求勾选以下配置项，确保不缩容以下特定类型的节点。
    - 含有本地存储 Pod 的节点。

- 含有 kube-system namespace 下非 DaemonSet 管理的 Pod 的节点。
- **扩容算法**：集群扩容时所依赖的算法准则，提供以下几种选择：
  - **随机**：有多个节点池时，随机选择一个节点池进行扩容。
  - **most-pods**：有多个节点池时，选择能调度更多 Pod 的节点池进行扩容。
  - **least-waste**：有多个节点池时，选择 Pod 调度后资源剩余更少的节点池进行扩容。
  - **priority**：该策略需要您在kube-system命名空间下创建相关 ConfigMap 来配置优先级策略，详情请参见 Cluster Autoscaler 说明 。

5. 单击**完成**，即可设置成功。

调整节点池配置

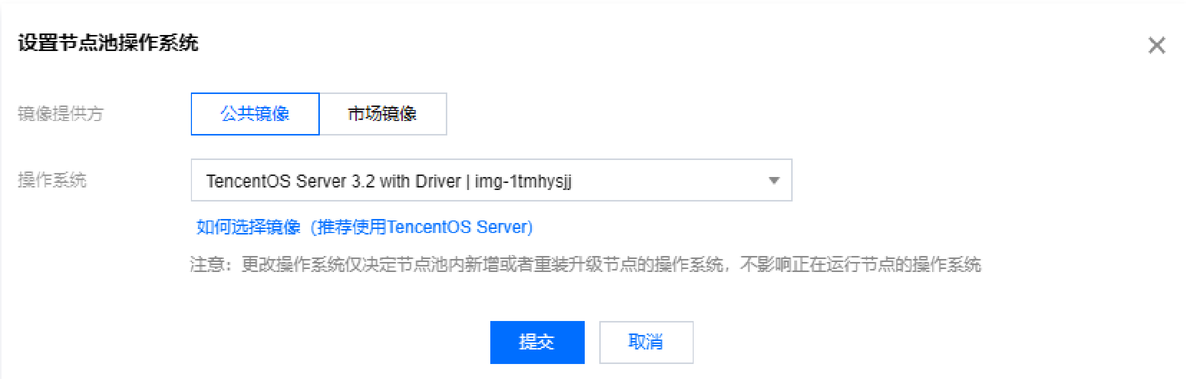
调整节点池操作系统、备选机型、容器运行时

1. 登录 容器服务控制台，单击左侧导航栏中的集群。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点池**。
4. 单击节点池 ID，进入节点池详情页。
5. 在节点配置详情页，可对节点池属性进行更改。如下图所示：

操作系统

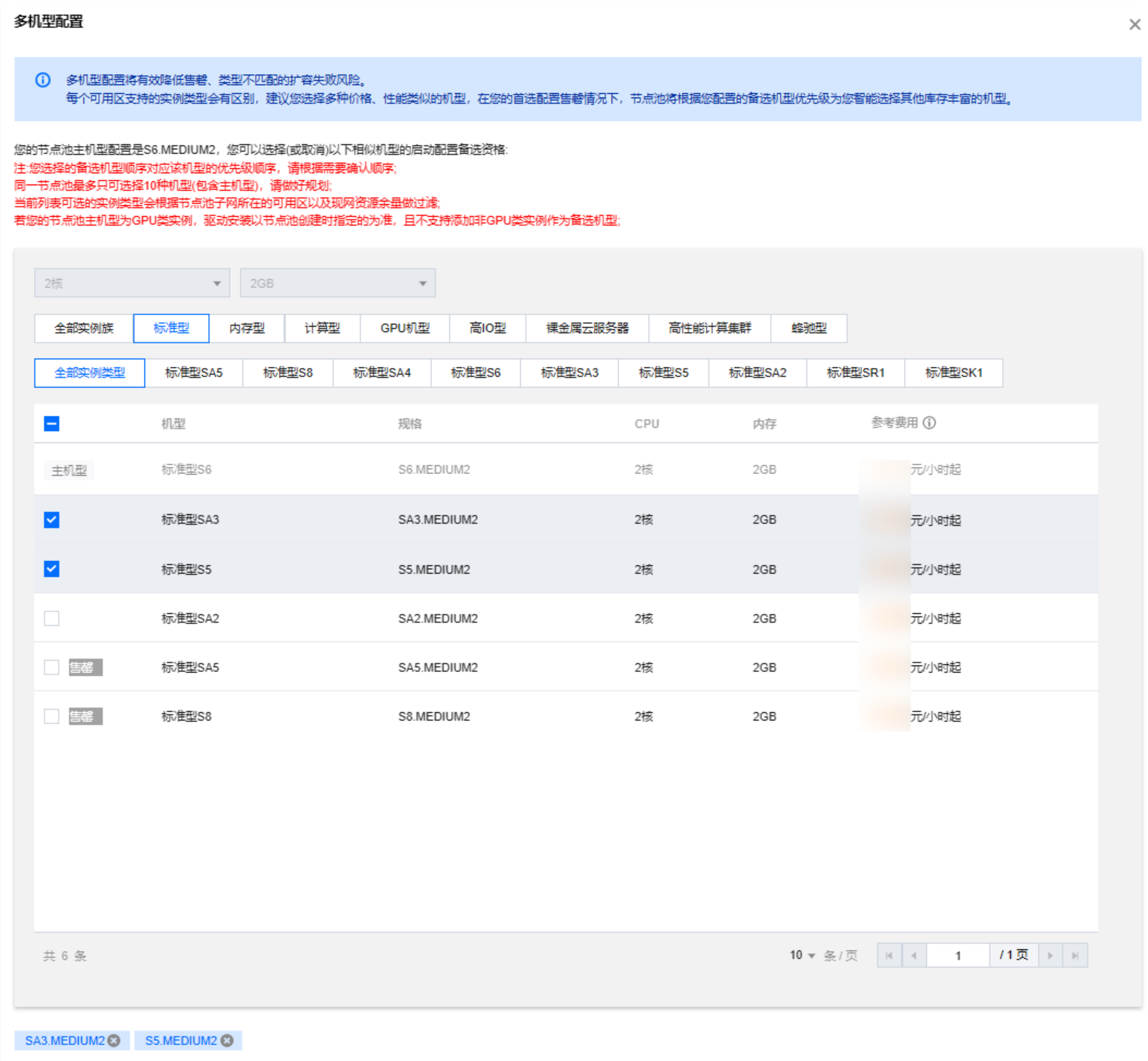
单击**操作系统**右侧的 ，即可更改节点池的操作系统。更改操作系统仅决定节点池内新增或者重装升级节点的操作系统，不影响正在运行节点的操作系统。

如下图所示：



机型

单击**机型**右侧的，即更改节点池的备选机型（主机型不可更改）。设置备选机型可有效降低由于主机型售罄导致扩容失败的风险。如下图所示：



- 备选机型顺序对应该机型的优先级顺序，请根据需要设置机型顺序，您可以通过弹窗最下方展示的机型顺序进行确认。
- 备选机型必须与主机型规格（CPU、内存、CPU 架构）相同。
- 同一节点池最多只可选择10种机型（包含主机型），请按需自行规划。

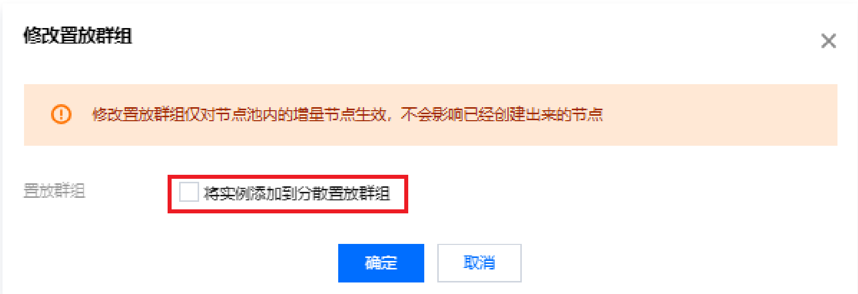
运行时组件

单击**运行时组件**右侧的，即可更改节点池的运行时组件以及版本，详情请参见 如何选择运行时组件 。如下图所示：

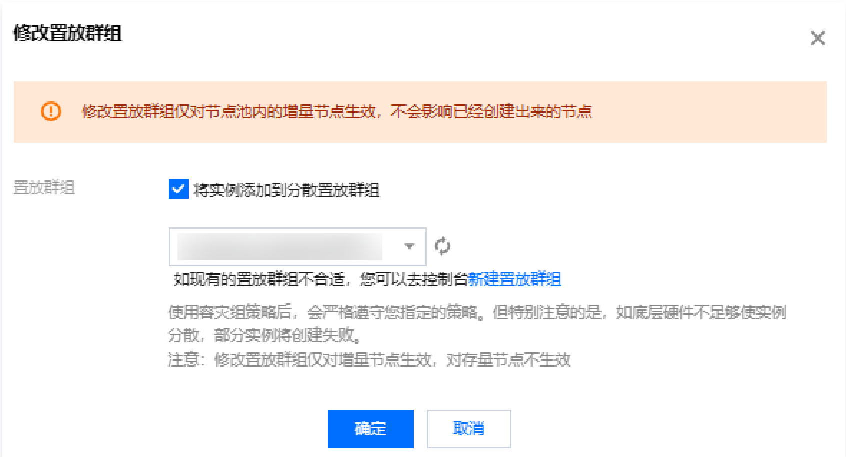
修改置放群组

1. 在节点配置详情页，单击**置放群组**右侧的 ，即可修改置放群组的配置。如下图所示：

2. 在**修改置放群组**中，若将该选项取消勾选，则该节点池创建出的节点将不具备置放群组的能力。如下图所示：



3. 勾选该选项，可指定新的置放群组策略，但新的置放群组策略仅对节点池内的增量节点生效，不会影响已经创建出来的节点。如下图所示：



4. 单击**确定**。

调整节点数量范围、Label、Taints

1. 单击目标节点池名片页右上角的**编辑**。如下图所示：

2. 在**调整节点池配置**页面，参考以下信息进行设置。如下图所示：

- **节点池名称**：自定义。可根据业务需求等信息进行命名，方便后续资源管理。
- **弹性伸缩**：根据实际需求进行勾选。
- **节点数量范围**：节点数量将在设定的节点范围内自动调节，不会超出该设定范围。

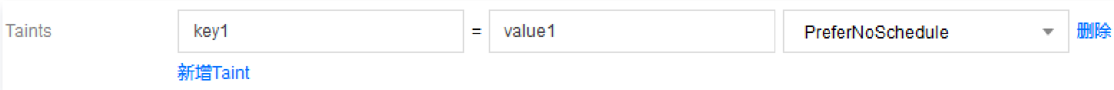
> ⚠ **注意：**

> 该数量范围的设置，将影响 调整节点池下节点数量 操作。例如，当前节点池的节点数量已达到该范围最大值时，节点数量将不再支持上调。

- **实例移出策略**：提供**优先移出最新实例**和**优先移出最旧实例**。节点池的移出策略决定了在减少节点数量时，哪些节点会被优先移除。
- **Label**：该节点池下所创建的节点将自动加上此处设置的 Label，方便后续根据 Label 筛选、管理节点。单击**新增 Label**，即可进行 Label 自定义设置。
- **Taints**：节点属性，通常与 `Tolerations` 配合使用。此处可为节点池下的所有节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在不符合条件的 Pod 也将会被驱逐。

> **说明：**
> Taints 内容一般由 `key` 、 `value` 及 `effect` 三个元素组成。其中 effect 可取值通常包含以下三种：
> - **PreferNoSchedule**：非强制性条件，尽量避免将 Pod 调度到设置了其不能容忍的 taint 的节点上。
> - **NoSchedule**：当节点上存在 taint 时，没有对应容忍的 Pod 一定不能被调度。
> - **NoExecute**：当节点上存在 taint 时，对于没有对应容忍的 Pod，不仅不会被调度到该节点上，该节点上已存在的 Pod 也会被驱逐。

以设置 Taints `key1=value1:PreferNoSchedule` 为例，控制台配置如下图所示：



3. 单击**确定**并等待更新完成即可。

调整节点池下节点数量

1. 单击目标节点池名片页右侧的**更多 > 调整数量**。
2. 在**调整数量**页面，按需调整节点数量，该数量必须落在设置的节点池数量范围内。

> **说明：**
> 节点池已开启弹性伸缩时，该数量将会随着集群工作负载自动调整，可能会存在最终实际的节点数量与数量调整时所设置的值不一致的问题。

3. 单击**确定**等待数量调整完成即可。

启用或停用弹性伸缩

> **说明：**
> - 执行启用/停用弹性伸缩操作时，仅建议在容器服务侧节点池处进行，以确保该状态能够同步至 Cluster-autoscaler。
> - 同一时间，控制台节点池弹性伸缩只能由一个角色控制，若已启用弹性伸缩，则不能手动调整实例数量；若要手动调整实例数量，则先关闭弹性伸缩。

1. 单击目标节点池名片页右上角的**更多**。如下图所示：

2. 结合实际情况选择**启用弹性伸缩**或者**停用弹性伸缩**，并在弹出的窗口中单击**确认**即可。

设置缩容保护

1. 在节点管理页面，选择**节点池**。
2. 单击节点池 ID，进入节点池详情页。
3. 在节点列表页面，选择节点右侧的**更多 > 开启缩容保护**。如下图所示：

4. 开启成功后，您可以在节点列表中查看节点缩容保护状态。如下图所示：

相关操作

您可参考以下文档，了解更多节点池功能及操作：

- 创建节点池
- 查看节点池
- 删除节点池

删除节点池

最近更新时间：2023-11-20 17:21:21

操作场景

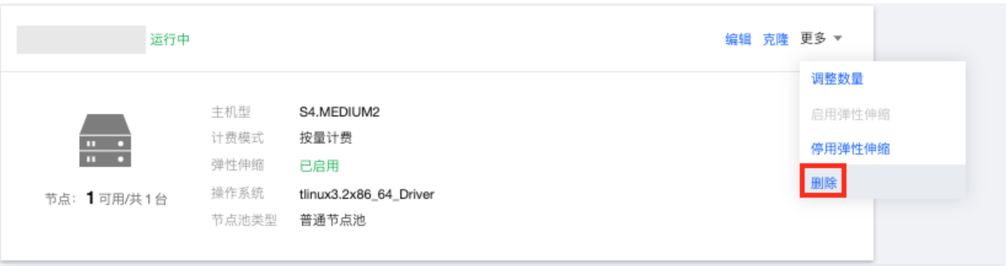
本文介绍如何通过容器服务控制台删除集群下已创建的节点池。您可参考本文删除不再使用的节点池，减少不必要的资源浪费。

前提条件

- 已创建可用节点池。详情请参见 创建节点池 。
- 已进入“节点池列表”页面。详情请参见 查看节点池 。

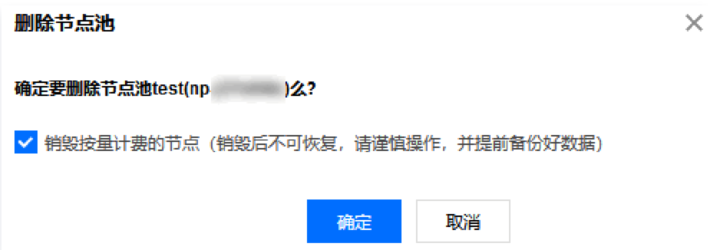
操作步骤

1. 选择目标节点池名片页右上角的**更多** > **删除**。如下图所示：



2. 在弹出的**删除节点池**窗口中，按需设置是否保留节点。如下图所示：

> **注意**
> - 默认勾选销毁按量计费的节点，可根据实际需求取消勾选。
> - 按量计费的节点销毁后不可恢复，请谨慎操作，并提前备份好数据。



3. 单击**确定**，等待删除成功即可。

相关操作

您可参考以下文档，了解节点池更多功能及操作：

- 创建节点池
- 查看节点池
- 调整节点池

查看节点池伸缩记录

最近更新时间：2025-02-27 17:55:12

操作场景

本文介绍如何查看节点池的伸缩记录，适用于以下场景：
- 您可以通过伸缩活动了解自己业务的流量变化，更有效的按需配置节点池。
- 您可以通过节点池内节点扩缩容活动来了解自己的花费来源，进行更高效的成本管理。
- 您可以了解扩缩容活动失败的原因（例如扩容时由于地域资源售罄导致扩容失败），进行风险管理。
- 您可以查看两个层级的伸缩记录：全局伸缩记录 和 特定节点池伸缩记录。

> **说明：**
> - 在存在多个节点池的情况下，CA（Cluster Autoscaler）负责选择合适的节点池进行扩缩容，全局伸缩记录可以从 CA 的 Event 得到。
> - 如果您只关心特定节点池的伸缩记录，不关心 CA 的行为，可进入节点池详情页查看该节点池的扩缩容活动记录。

前提条件

已创建可用节点池，详情请参见 创建节点池。

操作步骤

查看全局伸缩记录

社区开源组件 CA 会把任何一次扩缩活动的相关信息，以 Kubernetes event 的形式存储到特定的 Pod 或者 Node 下，但存在 Kubernetes events 资源默认后端只存储1小时的限制。若您想对节点池的扩缩记录进行查询及复盘，建议您开启集群的事件持久化功能，对 Kubernetes Events 进行持久存储。

开启事件日志

登录 腾讯云容器服务控制台，为集群开启事件日志，操作步骤请参见 事件日志。

查看事件

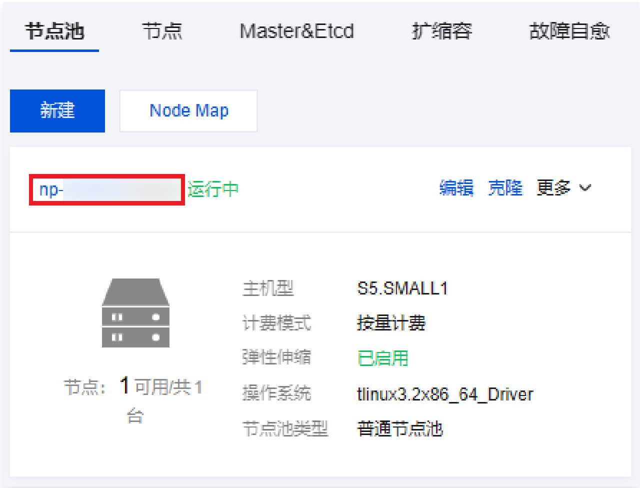
1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击事件日志右侧的**检索分析**。
4. 在**事件日志 > 全局检索**页面，在字段中选择 **event.source.component:cluster-autoscaler**，按需自行配置显示字段并进行检索分析。

检索指引

您可参考以下文档，查看更具体的扩缩容活动列表：
- CLS 检索语法
- CA FAQ
- CA 扩缩容 Event 的 Reason 字段可能有如下取值：TriggeredScaleUp、NotTriggerScaleUp、ScaledUpGroup、FailedToScaleUpGroup、ScaleDown、ScaleDownFailed、ScaleDownEmpty。详情请参见 字段详细介绍。

查看特定节点池伸缩记录

1. 登录 容器服务控制台，选择左侧导航中的**集群**。
2. 在集群列表页中，单击目标集群 ID，进入该集群详情页。
3. 选择左侧导航中的**节点管理**，在节点池页面中单击目标节点池 ID。如下图所示：



4. 进入该节点池详情页，选择顶端**伸缩记录**页签，即可查看伸缩记录。如下图所示：

伸缩记录展示字段如下：
- **活动 ID**：伸缩活动 ID。
- **状态**：伸缩活动的状态。
- **描述**：此次伸缩活动的描述，显示扩容机器数/缩容机器数。
- **活动起因**：触发此次伸缩活动的原因，例如“因匹配期望实例数”。
- **失败原因**：如果伸缩活动**状态**为失败，该栏会显示伸缩活动的失败原因。
- **开始时间**：伸缩活动开始的时间，精确到秒。
- **结束时间**：伸缩活动结束的时间，精确到秒。

相关操作

您可参考以下文档，了解节点池更多功能及操作：
- 创建节点池
- 查看节点池
- 调整节点池

新增游离普通节点

最近更新时间：2025-03-24 18:01:12

操作场景

若您不希望通过节点池管理 CVM 普通节点，可以通过以下方式为集群添加不归属节点池管理的普通节点。

- 新建离散的普通节点
- 添加已有 CVM 机器至集群管理

操作步骤

新建离散的普通节点

1. 登录 容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**页面，选择**节点**页签，单击**更多操作**，选择**新建普通节点**。
4. 在**新建节点管理**中，根据实际需求配置相关参数。如下图所示：

主要参数信息如下：

- **镜像提供方**：支持选择**公共镜像**。详情请参见 镜像。

- **计费模式**：提供**按量计费**和**包年包月**两种计费模式。详情请参见 计费模式 。
- **可用区**：该参数仅用来筛选可用区下可用的子网列表。
- **集群网络**：选择为本次新建节点分配 IP 的子网，单次创建节点操作只支持单子网。
- **机型配置**：单击**请选择机型**，在**机型配置**页面参考以下信息进行选择：
  - **机型**：支持通过 CPU 核数、内存大小及实例类型进行筛选。详情请参见 实例规格 和 快速入门 Linux 云服务器 。
  - **系统盘**：存储控制、调度云服务器运行的系统集合。支持查看所选机型的可选系统盘类型，请参考 云硬盘类型 并根据实际需求进行选择。
  - **数据盘**：用于存储所有的用户数据。
  - **公网带宽**：勾选后会为您分配公网 IP，默认公网带宽为1Mbps，您可以根据需求调整。
- **实例名称**：控制台显示的云服务器 CVM 实例名称，该属性受主机名命名模式限制。提供以下两种命名方式：
  - **自动命名**：主机名为自动命名模式，支持批量连续命名或指定模式串命名，最多输入60个字符。默认自动生成实例名，格式为 `tke_集群id_worker` 。
  - **手动命名**：主机名为手动命名模式，实例名称与主机名相同，无需重新配置。

  > **说明：**
  > 当集群参数“节点 hostname 命名模式”为“手动命名”时（可前往集群的基本信息页查看），新建游离节点才会展示“实例名称”的命名方式。

- **登录方式**：提供以下三种登录方式，请根据实际情况进行选择。
  - **立即关联密钥**：密钥对是通过算法生成的一对参数，是一种比常规密码更安全的登录云服务器的方式。详情请参见 SSH 密钥 。
  - **SSH密钥**：该配置项仅在选择**立即关联密钥**登录方式时出现，在下拉框中选用已有密钥即可。若需新建，请参考 创建 SSH 密钥 。
  - **自动生成密码**：自动生成的密码将通过 站内信 发送给您。
  - **设置密码**：请根据提示设置对应密码。
- **安全组**：默认为创建集群时所设置的安全组，可根据实际需要进行更换或添加。
- **云服务器数量**：创建实例数量，请根据实际需求进行设置。

5. （可选）单击**更多设置**，查看或配置更多信息。如下图所示：

- **忽略容器 IP 不足校验**：忽略后将会导致此节点 NotReady，并且只能调度 hostNetwork 的 pod 到此节点。
- **CAM 角色**：可为本批次创建的所有节点绑定相同的 CAM 角色，赋予节点该角色绑定的授权策略。详情请参见 管理实例角色 。
- **容器目录**：勾选即可设置容器和镜像存储目录，建议存储到数据盘。例如 `/var/lib/docker` 。
- **安全加固**：默认免费开通 DDoS 防护 和 主机安全基础版 。
- **云监控**：默认免费开通云产品监控、分析和实施告警，安装组件获取主机监控指标。详情请参见 腾讯云可观测平台 。
- **封锁初始节点**：勾选“开启封锁”后，将不接受新的 Pod 调度到该节点，需要手动取消封锁的节点，或在自定义数据中执行 取消封锁命令 ，请按需设置。
- **Label**：单击**新增**，即可进行 Label 自定义设置。可用于后续根据 Label 筛选、管理节点。
- **Taints**：为节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在不符合条件的 Pod 也将会被驱逐。
- **置放群组**：选择是否将实例添加到分散置放群组，详情请参见 置放群组 。
- **自定义脚本**：指定自定义数据来配置 Node，即当 Node 启动后运行配置的脚本，需要自行保证脚本的可重入及重试逻辑, 脚本及其生成的日志文件可在节点的 `/usr/local/qcloud/tke/userscript` 路径查看。

6. 单击**新建节点**。

添加已有 CVM 机器至集群管理

> **注意**
>
> - 当前仅支持添加同一 VPC 下的云服务器。

- 请勿添加公网网关 CVM 加入集群，该类型 CVM 重装加入集群后产生 DNS 异常，会导致该节点不可用。

1. 登录 容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**页面，选择**节点**页签，单击**更多操作**，选择**添加已有节点**。
4. 在**添加已有节点**页面，根据实际需求配置相关参数。
    - **选择节点**：勾选需要添加的节点。
    - **数据盘挂载**：
        - 勾选：将格式化挂载相关设置，需要填写设备名称，格式化系统（可以选择不格式化），挂载点。如果您想将 /dev/vdb 这块设备格式化成 ext4，并挂载到 /var/lib/docker 目录下，可以这样设置：设备名称 `/dev/vdb`，格式化系统 `ext4`，挂载点 `/var/lib/docker`。

            注意：
            - 若高 IO 型、高性能 HCC 类机型需要对 NVMe 类型数据盘进行挂载，建议通过 为数据盘设置文件系统卷标 的方式单独添加进集群，不要和其他机型一起操作。
            - 提前备份重要数据，如果已经自行格式化盘，则无需选择格式化系统，只需填写挂载点。
            - 您填写的格式化挂载设置会对本批次添加节点全部生效，请确认填写的设备名称，例如 /dev/vdb 符合您的预期（如果您对 CBS 做了热插拔等操作，设备名称可能会变化）。
            - 如果您对盘做了分区 /LVM，在设备名称处填写分区名 /LVM 名，配置对应的格式化挂载参数即可。
            - 如果您填写了错误的设备名称，系统会报错并终止节点初始化流程。
            - 如果您填写的挂载点不存在，系统会为您创建对应目录，不会报错。
        - 不勾选：不设置数据盘挂载选项，可手动或者使用脚本挂载。
    - **容器目录**：设置容器和镜像存储目录，建议存储到数据盘。
    - **镜像提供方**：支持选择**公共镜像**。详情请参见 镜像。
    - **操作系统**：操作系统为集群级别，您可以前往集群详情页进行修改，修改后新增或重装的节点将使用新的操作系统。
    - **登录方式**：
        - 设置密码：请根据提示设置对应密码。
        - 立即关联密钥：密钥对是通过一种算法生成的一对参数，是比常规密码更安全的登录云服务器的方式，具体详情可参阅 SSH 密钥。
        - 自动生成密码：自动生成的密码，该密码将通过站内信发送给您。
    - **安全组**：用于设置云服务器 CVM 的网络访问控制，请根据实际需求进行选择。您还可以单击**新建安全组**，放通其他端口。
    - **安全加固**：默认免费开通 DDoS 防护 和 主机安全基础版。
    - **云监控**：默认免费开通云产品监控、分析和实施告警，安装组件获取主机监控指标。详情请参见 腾讯云可观测平台。
5. 单击**完成**。

节点池 FAQ

最近更新时间：2025-03-19 14:15:42

本文主要介绍普通节点池使用过程中的常见问题。

节点池和伸缩组之间的关系是什么?

节点池是一组规格、配置、属性相似的节点集合，您可在 容器服务控制台 对这组节点进行批量运维，例如设置节点规格、Label、Taints、脚本等参数。同一个集群中，允许创建不同计费类型（按量付费、包年包月、竞价实例）的节点池。节点池的底层实现依赖云产品 弹性伸缩 AS（Auto Scaling），主要包含如下两个概念：

- **伸缩组**是遵循相同规则、面向同一场景的云服务器实例的集合。伸缩组定义了组内 CVM 实例数的最大值、最小值等属性。
- **启动配置**是自动创建云服务器的模板，其中包括云服务器实例类型、系统盘及数据盘类型和容量、密钥对、安全组等。

每个节点池对应一个伸缩组，每个伸缩组对应一个启动配置，您可在 容器服务控制台 节点池详情页中获取绑定的 AS “启动配置”和“伸缩组”链接。

节点池支持修改哪些参数?

> **警告：**
> 除了本文档中提到的可在 AS 控制台修改的参数外，不建议您调整其他参数。否则，可能会导致节点池的弹性伸缩功能无法正常使用。

<table>
<tr><th>参数项</th><th>推荐修改方式</th></tr>
<tr><td>节点池名称</td><td rowspan="4">TKE 控制台：节点池基本信息页可编辑。
注意：
节点池数量不可超过100。</td></tr>
<tr><td>弹性伸缩能力</td></tr>
<tr><td>节点数量范围</td></tr>
<tr><td>云标签、删除保护</td></tr>
<tr><td>Labels、Taints</td><td>TKE 控制台：节点池基本信息页可编辑，支持选择是否对存量节点应用本次修改。</td></tr>
<tr><td>操作系统</td><td rowspan="3">TKE 控制台：节点配置详情页可编辑，修改后仅对节点池下的新增节点生效。</td></tr>
<tr><td>运行时组件</td></tr>
<tr><td>数据盘</td></tr>
<tr><td>备选机型</td><td>TKE 控制台：节点配置详情页可编辑：
• 同一节点池最多只可选择10种机型（包含主机型），请做好规划。
• 当前列表可选的实例类型会根据节点池子网所在的可用区以及现网资源余量做过滤。
• 若您的节点池主机型为 GPU 类实例，驱动安装以节点池创建时指定的为准，且不支持添加非 GPU 类实例作为备选机型。</td></tr>
<tr><td>自定义数据</td><td>TKE 控制台：节点配置详情页可查看并编辑，修改后仅对节点池下的新增节点生效。
注意：
TKE 平台需在“自定义数据”中注入节点初始化 Agent 安装脚本，因此不推荐您在 AS 的启动配置里直接修改“自定义数据”，可能会影响节点正常加入集群。</td></tr>
<tr><td>安全组</td><td>AS 控制台：启动配置详情页可编辑，修改后仅对节点池下的新增节点生效。</td></tr>
<tr><td>实例名称（节点名）</td><td>AS 控制台：启动配置高级设置信息可编辑，修改后仅对节点池下的新增节点生效。</td></tr>
<tr><td>子网</td><td>AS 控制台：伸缩组详情页可编辑，修改后仅对节点池下的新增节点生效。</td></tr>
<tr><td>实例创建策略/重试策略</td><td>AS 控制台：伸缩组策略信息页可编辑，修改后对下一次伸缩活动生效。</td></tr>
</table>

不支持修改参数说明

参数项	影响描述

计费模式	节点池不支持修改计费模式，同时也**不推荐**您在 AS 的启动配置里修改“实例计费模式”。
支持网络（VPC）	节点池不支持修改 VPC，同时也**不推荐**您在 AS 的伸缩组详情页修改“支持网络”，否则节点可能会扩容失败。

说明：

- 每个节点池对应一个唯一的伸缩组和启动配置，启动配置不能绑定到其他伸缩组，否则节点池会删除失败。
- 通过 AS 控制台为伸缩组设置**告警触发策略**或**定时任务**而触发扩容的实例，无法被节点池感知，可能会影响集群组件 CA 的弹性伸缩判断，因此不建议在 AS 控制台修改伸缩组的其他参数。

节点初始化流程 FAQ

最近更新时间：2023-08-22 16:33:01

本文主要介绍普通节点初始化流程中的常见报错信息和潜在原因。

安装节点组件相关

描述：Cloud init failed: call common server failed, please confirm node network and security group is correctly.
说明：Cloud Init 执行失败，节点有特殊安全组或节点网络存在问题。

描述：TKE agent lost: please confirm CVM status is normal and network is connection.
说明：节点 agent 失联，可能是机器启动慢或启动失败，或节点有错误的网络、安全组设置、自定义镜像未清理 cloud init。

描述：Install package failed: command yum didn't complete in 60 seconds
说明：节点安装系统包超时，需要检查安全组配置、是否是自定义镜像修改过 yum/apt 源。

描述：Download resource failed: resource %s download failed, network xxx.
说明：检查是否为网络问题导致资源下载失败。

描述：Node check failed: failed to resolve address of common server, please confirm node network and security group is correctly.
说明：节点检查失败，agent server 无法访问。

描述：Node check failed: GPU inforom corrupted, please re-purchase or contact us
说明：节点检查失败，可能为 GPU inforom 损坏，并无法自修复。

描述：Start service failed: service %s start failed
说明：一般在使用自定义镜像场景出现，需检查镜像。

描述：Mount disk failed:
说明：挂盘失败，需根据具体情况分析。

等待节点注册相关

描述：Kubelet start failed: failed to run Kubelet: unknown policy: "xx"
举例：Kubelet start failed: failed to parse kubelet flag: invalid argument "Inplace" for "--feature-gates" flag: malformed pair, expect string=bool
说明：kubelet 参数设置导致启动失败。

GPU 相关

描述：GPU component not found: daemonset %s not found, please make sure the %s component is installed
说明：一般是 GPU Device Plugin 没安装，例如开启 qgpu、或者运行过程中把 Device Plugin 删掉。

描述：GPU component start failed: please ensure node has enough resources and daemonset %s has correct node selector
说明：GPU Device Plugin 没调度，可能是由于节点资源不够、Device Plugin 被修改等原因，需保证 Device Plugin 能调度到节点上。

描述：GPU component start failed: pod of %s is not running, please check node status and GPU driver
说明：GPU pod 启动失败，自定义 Device Plugin 有问题、节点使用自定义镜像、驱动安装有问题、nvidia runtime hook 有问题等。

描述：GPU resource register failed: gpu resource not found on node or in GPU device plugin
说明：GPU 资源注册失败，可能是节点或 pod 看不到 GPU 卡导致：如果在节点上执行 nvidia-smi 报错或看不到显卡，则是机型或驱动安装问题；如果在 Device Plugin 中执行 nvidia-smi 报错或看不到显卡，可能是 nvidia-runtime 或者自定义 Device Plugin 环境变量 NVIDIA_DRIVER_CAPABILITIES 设置有问题。

原生节点管理
原生节点概述

最近更新时间：2025-03-24 18:01:12

什么是原生节点?

原生节点是由腾讯云 TKE 容器服务团队面向 Kubernetes 环境推出的全新节点类型，依托腾讯云千万核容器运维的技术沉淀，为用户提供原生化、高稳定、快响应的 K8s 节点管理能力。

产品优势

搭载 FinOps 理念，助力云上资源成本优化

- 搭载 HouseKeeper 可视化资源大盘，助力提升节点资源利用率，实现云上降本增效。
- 负载 智能 Request 推荐，减少资源闲置。
- 提供 专有动态调度能力，支持如下特性：
  - 均衡负载：基于节点真实以及历史负载情况，让节点的资源负载更加均衡。
  - 提升装箱：放大节点 CPU/内存资源可调度量，将节点装箱率提升到100%以上。
  - 规整业务：通过设置目标利用率保证节点被持续调度，让业务资源部署更“集中”。

多维度节点管理能力，全方位降低运维负担

- K8s 运维新范式：提供基础设施声明式 API，像管理 workload 一样管理节点，可通过 kubernetes api、云 API、控制台三种方式管理节点。
- 自研智能运维系统：支持操作系统/运行时/k8s 层面的故障检测和自动升级，多维度助力用户降低运维负担。
- 结合腾讯云内部云原生技术实践，对操作系统、运行时、kubernetes 进行全方位参数调优和适配，节点初始化稳定性显著增强。

原生节点 VS 普通节点

原生节点包含普通节点的全部能力，且做了全方位增强：

模块	原生节点	普通节点
管理模式	节点管家模式：平台在资源管理和稳定性运维上提供能力辅助客户分析决策	Serverful 模式：用户分析、用户决策、用户执行
基础设施声明式管理	支持	不支持
Pod原地升降配	支持	不支持
内核参数调优等自定义配置入口	支持	不支持
节点故障自愈	自研操作系统、K8s 环境、运行时级别的 故障检测和自愈能力	社区 NPD（已停止维护）
调度器	原生节点专用调度器，支持虚拟放大可调度资源	社区 DynamicScheduler、DeScheduler
Request 智能推荐	支持 推荐和一键更新	不支持
可抢占 Job	支持	不支持
节点管理	支持内核 / Nameserver / Hosts 参数配置、前置脚本 / 后置脚本能力	不支持

计费模式

原生节点支持多种 CVM 实例类型，您可基于应用规模和业务特性选择合适的实例进行部署。容器服务对原生节点实例所消耗的资源（含 CPU、内存、GPU 和系统盘）按照实例类型和资源规格进行收费。
原生节点支持**按量计费**和**包年包月**两种计费模式。

计费模式	包年包月	按量计费

付款方式	预付费	后付费（购买时冻结费用，每小时结算）
计费单位	元/月	元/秒
单价	单价较低	单价较高
最少使用时长	至少一个月	按秒计费，按小时结算，随时购买随时释放
使用场景	适用于节点资源需求量长期稳定的成熟业务	适用于转码、大数据、电商抢购等周期性计算场景或开启自动伸缩（HPA）的潮汐型在线服务场景。

地域和可用区

您可在以下地域内使用原生节点。

中国

地域			地域简称
中国	公有云地域	北京	ap-beijing
		南京	ap-nanjing
		上海	ap-shanghai
		广州	ap-guangzhou
		成都	ap-chengdu
		重庆	ap-chongqing
		香港	ap-hongkong
		台北	ap-taipei
	金融云地域	深圳金融	ap-shenzhen-fsi
		上海金融	ap-shanghai-fsi
		北京金融	ap-beijing-fsi

其他国家和地区

地域			地域简称
亚太	公有云地域	新加坡	ap-singapore
		雅加达	ap-jakarta
		首尔	ap-seoul
		曼谷	ap-bangkok
		东京	ap-tokyo
北美		硅谷	na-siliconvalley
		弗吉尼亚	na-ashburn
南美		圣保罗	sa-saopaulo
欧洲		法兰克福	eu-frankfurt

原生节点相关操作

> **说明：**

> 为了更方便的对原生节点进行分组管理，推荐通过**节点池**创建并管理一组参数相同的原生节点。

- 新建原生节点：您可通过 控制台、Kubernetes API 、云 API 三种方式完成集群内原生节点的创建。
- 删除原生节点
- 故障自愈规则
- 声明式操作实践
- 原生节点扩缩容
- Pod 原地升降配
- Management 参数介绍

原生节点功能支持说明

最近更新时间：2025-06-25 14:27:12

参数	支持详情	描述
机型	• 标准型：S2、S4、S5、SA2、SA3、S6、SA5、SA4、S7、S8 • 计算型：C3、C4、C5、C6 • 内存型：MA2、M3、MA3、M5、M6、M8 • GPU 型：GN7、GNV4、PNV4、GN10X、GN10Xp、GT4、Pnv5i、Pnv5b、Pnv5、Pnv6、GC49d • 高 IO 型：IT5 • 裸金属：BMS5、HCCPNV4sne、HCCPNV4sn、HCCPNV5、HCCPNV5v、HCCPNV4ane、HCCPNV6、BMIA2m • 蜂驰型：BF1	控制台机型展示与所选可用区的资源剩余量相关，更多机型需求您可通过 提交工单 来寻求帮助。此外，原生节点池**支持配置规格相同的多种备选机型**，您可前往节点池详情页设置。
系统盘	高性能云盘、SSD 云盘、通用型 SSD、增强型 SSD	建议系统盘至少为100GB。
数据盘	高性能云盘、SSD 云盘、通用型 SSD、增强型 SSD	建议数据盘至少为50GB，默认不绑定数据盘。
公网带宽	支持绑定 EIP	默认不开启节点公网带宽，详情请参见 开启公网访问。
操作系统	TencentOS Server	依托腾讯云虚拟化平台，通过内核优化等技术为云原生资源隔离技术提供支撑，针对容器应用场景进行全面参数调优。
节点登录	SSH 登录	默认开启节点登录，您可在控制台为待登录节点开启并下发 SSH 密钥，详情请参见 开启 SSH 密钥登录。
GPU 驱动	450/470/515/525/535 驱动	节点池创建时，您可在机型选择下方勾选目标驱动。
运行时	仅支持 containerd	集群 1.16 及以上 Kubernetes 版本对应支持的 containerd 版本为 1.6.9。
Kubernetes	Kubernetes 大版本 ⩾ 1.16	部分 Kubernetes 版本对小版本号有要求，具体如下：v1.16.3-tke.28 及以上；v1.18.4-tke.26及以上； v1.20.6-tke.21及以上。
运维参数	支持 Kubelet、Kernel（内核）、Hosts、Nameservers 参数设置	详情请参见 Management 参数介绍。
初始化脚本	支持节点初始化前、后两个阶段	您可在创建节点池时提供初始化脚本。
节点自动伸缩	支持	详情请参见 原生节点扩缩容。
节点规格放大	支持，提供可扩展的原生节点专用调度器	详情请参见 专用调度器产品说明。
qGPU	支持	详情请参见 qGPU概述。
内存压缩	支持	详情请参见 内存压缩使用说明。
可抢占式 Job	支持	详情请参见 可抢占式 Job 功能说明。
QoSAgent	支持	提供 CPU 优先级、CPU Burst、内存/网络/磁盘 IO QoS 增强等多维精细调度能力，在提升集群资源利用率的同时，提供稳定性质量保障。详情请参见 精细调度。
故障自愈	支持	基于 TKE 自研智能化运维产品云探提供多维度故障自愈能力，全方位提升节点稳定性和运维效率，详情请参见 故障自愈规则。

原地升降配	支持	详情请参见 开启 Pod 原地升降配。

购买原生节点

原生节点产品定价

最近更新时间：2024-06-27 14:09:11

产品定价

> **说明：**
> 原生节点为容器服务 TKE 单独计费云产品，其定价与云服务器 CVM 存在差异，不同规格的计费信息以控制台展示为准。

TKE 原生节点支持多种 CVM 实例类型，您可基于应用规模和业务特性选择合适的实例进行部署。容器服务对原生节点实例所消耗的资源（含 CPU、内存、GPU 和系统盘）按照实例类型和资源规格进行收费。

计费公式：费用 = 原生节点实例资源配置单价 × 运行时间

计费项	描述
实例类型	当前原生节点支持的实例类型有，详细介绍可参考 CVM 实例： • 标准型：S2、S4、S5、SA2、SA3、S6、SA5、SA4、S7、S8 • 计算型：C3、C4、C5、C6 • 内存型：M3、MA3、M5、M6 • GPU 型：GN7、GNV4、PNV4、GN10X、GN10Xp、GT4 • 高 IO 型：IT5
资源规格	当前原生节点支持的资源及规格，以控制台实际展示为主： • CPU：最低支持2核，实际可选规格以控制台展示为准。 • 内存：最低支持2GB，实际可选规格以控制台展示为准。 • 系统盘：支持高性能和 SSD 两种系统盘类型，系统盘大小为50GB-1024GB。 • GPU：仅 GPU 型原生节点实例含 GPU 资源，仅支持整卡 GPU 规格。

计费模式

TKE 提供两种不同的原生节点实例购买模式：包年包月、按量计费，分别适用于不同场景下的用户需求，详情如下：

计费模式	包年包月	按量计费
付款方式	预付费	后付费（购买时冻结费用，每小时结算）
计费单位	元/月	元/秒
单价	单价较低	单价较高
最少使用时长	至少一个月	按秒计费，按小时结算，随时购买随时释放
使用场景	适用于节点资源需求量长期稳定的成熟业务。	适用于转码、大数据、电商抢购等周期性计算场景或开启自动伸缩（HPA）的潮汐型在线服务场景。

包年包月

计费公式：支付金额 = 申请的原生节点实例数 × 原生节点实例资源配置单价（包月） × 包月时长

包年包月是 TKE 原生节点实例的一种预付费模式，提前一次性支付一个月或多个月甚至多年的费用，价格相较于按量计费模式更低廉。在购买包月原生节点实例前，可先查看云账户中的可用余额，如果余额小于系统将扣除的金额，请先充值后再购买。

按量计费

计费公式：支付金额 = 申请的原生节点实例数 × 原生节点实例资源配置单价（按量） × 按量时长

开通按量计费原生节点实例时，会预先冻结该实例一个小时的资源配置单价费用（含 CPU、内存、GPU 和系统盘），并在每个整点（北京时间）进行一次结算，根据您在上一个小时的实际使用时长进行扣费。购买时实例单价按小时呈现，结算时按实际使用秒数计价，费用四舍五入到小数点后2位。计费的起点以每个原生节点实例资源购买成功的时间点为准，终点以每个节点销毁（删除）操作完成的时间点为准。实例销毁时，系统将会对冻结的费用进行解冻。

包年包月价格

> **说明：**
> - 以下仅列出了原生节点部分机型的价格，未列出的规格价格请参考控制台展示。
> - 下列价格仅包含 CPU、内存，不包含系统盘及数据盘费用。
> - 下列价格为全地域统一价。

机型	规格	核数	内存	单价（元/月）
S5	S5.2XLARGE16	8	16	896
	S5.8XLARGE128	32	128	5376
	S5.LARGE8	4	8	448
	S5.MEDIUM2	2	2	168
	S5.MEDIUM4	2	4	224
	S5.SMALL2	1	2	112
	S5.SMALL4	1	4	168
SA3	SA3.2XLARGE16	8	16	679.6288
	SA3.4XLARGE32	16	32	1359.2576
	SA3.LARGE16	4	16	510.1088
SA2	SA2.2XLARGE16	8	16	622.08
	SA2.2XLARGE32	8	32	933.12
	SA2.4XLARGE32	16	32	1244.16
	SA2.4XLARGE64	16	64	1866.24
	SA2.LARGE8	4	8	311.04
	SA2.MEDIUM2	3	2	155.52
	SA2.MEDIUM4	3	4	194.4
	SA2.MEDIUM8	3	8	272.16
	SA2.SMALL2	1	2	77.76

相关文档

- 欠费说明
- 包年包月实例退费说明

欠费说明

最近更新时间：2023-12-20 19:19:01

当您的账号发生欠费时，系统将为您推送欠费提醒，请在收到欠费通知后，及时前往控制台 充值中心 进行充值，以免影响您的业务。本文向您介绍原生节点欠费的相关说明。

预警说明

预警类型	说明
到期预警	集群中的资源会在到期前第7天内，将向用户推送到期预警消息。预警消息将通过邮件及短信的方式通知到腾讯云账户的创建者以及全局资源协作者、财务协作者。
欠费预警	集群中的资源到期当天及以后，将向用户推送欠费预警消息。预警消息将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

回收机制

说明：
- 停服：指资源已经对用户停止服务，用户不可继续使用；但数据仍未清除，用户可对资源进行恢复。
- 释放：指资源的数据被删除，无法恢复。

包年包月回收机制

- 原生节点资源到期前7天内，系统会给您发送续费提醒通知。
- 账户余额充足的情况下，若您已设置自动续费，设备在到期当日会执行自动续费。
- 若您的原生节点实例在到期前（包括到期当天）未进行续费，系统将在到期**24小时内**对其作停服并释放处理（节点资源将被销毁且数据不可恢复）。
- 若您在欠费后的24小时内完成手动续费，续费周期的起始时间为上一周期的到期日。

按量计费回收机制

- 从您的账户余额被扣为负值时刻起，原生节点实例在**2小时**内可继续使用且继续扣费，2小时后若您的账户余额未充值到>0，系统对该实例作停服并释放处理（节点资源将被销毁且数据不可恢复）。
- 若您在账户欠费后的2小时内完成账户余额已充值到>0，原生节点实例将正常按照按量计费规则进行扣费。

包年包月实例退费说明

最近更新时间：2023-08-25 14:26:21

为了更加方便您使用 TKE 原生节点模式，如果您在购买包年包月原生节点后有任何不满意，我们支持自助退货退款。每个主体可享受1次1台原生节点实例五天无理由退还，您支付的有效金额将返还至您的腾讯云账户。除此之外，您还可享受普通自助退还，扣除您已使用的费用，将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户的方式返还到您的账户。以上操作均可在容器服务 TKE 控制台自助进行，具体操作详情请参见 删除原生节点 。

五天无理由自助退还

TKE 原生节点遵守腾讯云 云服务退货说明，如果您在购买原生节点后有任何不满意，我们支持五天内无理由自助退还，具体规则如下：
- 每个主体下，包年包月预付费原生节点自**新购之日起五天之内（含五天）**，默认享受1台原生节点五天无理由退还。
- 符合五天无理由退还场景的订单，退款金额为购买时花费的全部消耗金额，包括现金账户金额、收益转入账户金额以及赠送账户金额。
- 如需退还参与活动购买的原生节点，且退款规则与活动规则有冲突，则以活动规则为准。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。

> **说明**
> - 抵扣的代金券不予以退还。
> - 退还金额将全部退还到腾讯云账号余额。

普通自助退还

普通自助退还将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

普通自助退还限制

不满足五天无理由退货退款规则的退货订单，退款策略如下：每个账户每年支持**199台**包年包月原生节点控制台自助退还，普通自助退还将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

普通自助退还规则

退款金额 = 当前有效订单金额 + 未开始订单金额 - 资源已使用价值
- **当前有效订单金额**：指生效中订单的付款金额，不包含折扣和代金券。
- **未开始订单金额**：将来生效订单的付款金额，不包含代金券。
- **资源已使用价值**：用户发起退费的当天，原生节点使用已满整月的，按已使用整月的包年包月价格及对应折扣进行扣除；**不满整月的，按已使用时长（单位精确至秒）的按量计费价格进行扣除**。

> **说明：**
> 退款金额 ⩽ 0，按0计算并清退资源。

原生节点包年包月实例自助退还说明

- 包年包月实例自助退还后，挂载在实例上**系统盘均随实例一起回收**，保存在这些存储上的数据将丢失。挂载在该实例上的数据盘会继续保留，数据不受影响。
- 包年包月实例自助退还后，实例上运行的服务彻底中断，相关资源彻底释放，**不支持续费恢复**。
- 若包年包月实例在创建时绑定了弹性公网 IP，则**实例销毁后 IP 资源随实例一起释放**。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。

原生节点自助退还计费规则示例

> **说明**
> 以下价格仅供示例，非官网实际价格。具体单价可能因活动或策略等调整变化，请以控制台实际单价为准。

案例背景

广州二区标准型 S2 1核1GB，10GB 高性能云硬盘（系统盘），不含带宽，85元/月，使用200元代金券，购买1年，包年享7折。
折扣金额为 85 × 12 × 0.7 = 714（元）
支付金额为 714 －200 = 514（元）

五天无理由退还场景

购买五天内发现不满意，想要退还，且为该主体首次退还。
退费金额 = 真实支付价514元

普通自助退还场景

案例1

购买五天内，且为该账户本年度首次退还。
退还现金金额 = 实际支付金额514元

案例2

购买五天内，且不是该账户本年度首次退还，总使用时长48小时。
退还现金金额 = 514 − 48 × 0.29 = 500.08（元）

> **说明：**
> 0.29元为同样配置按量计费实例的单价。

案例3

购买五天内，且不是该账户本年度首次退还，已使用时长48小时；其中又续费1年，包年官网7折，续费实际支付金额714元。
退还现金金额 = 514 − 48 × 0.29 + 714 = 1214.08（元）

> **说明：**
> 48 × 0.29为生效订单退款金额，714元为未开始订单金额。

按量计费实例转包年包月

最近更新时间：2025-04-08 15:04:12

操作场景

腾讯云容器服务 TKE 开放了原生节点按量计费转换成包年包月功能，您可以在原生节点控制台及云 API 进行转换操作。本文主要介绍在控制台进行按量计费实例转换成包年包月实例的功能操作。

转换规则

在原生节点控制台提供计费模式转换功能，具体规则如下：

- 转换以节点池维度进行操作，节点池内的原生节点将一次性完成按量到包月模式的切换。
- 按量计费节点转换包年包月时会产生一个续费订单，必须完成该订单的支付流程，计费方式的变更才能生效。
  若未支付或未成功支付，该订单可在您的 订单管理 页面查看和处理。
- 计费模式由按量计费转换为包年包月的原生节点不支持五天内无理由退还。
- 计费方式转换成功及支付成功后，节点会即刻按照包年包月计费，新包年包月节点的起始时间为转换成功时间。
- 在未成功支付前，不可对该节点池进行重复计费模式转换。
- 按量计费转包年包月功能支持同步转换节点和磁盘（包括系统盘和数据盘）的计费方式。

> **注意：**
> - 非按量计费实例不支持转换。
> - 按量计费转包年包月为不可逆操作。

操作限制

- 所有**节点状态为正常时**才能进行计费模式的转换操作。
- 节点池**在转换前需关闭弹性伸缩功能**才能进行计费模式的转换操作。

操作步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择**节点管理**，在**节点池**中选择目标节点池，进入节点池的详情页。
4. 在**节点启动配置信息**模块中修改计费模式。如下图所示：

5. 在弹出的**按量计费转包年包月**窗口中，根据实际需求，设置续费时长以及是否自动续费。如下图所示：

- 续费时长：选择转换为包年包月后的购买时长。
- 自动续费：根据您的需求，选择自动续费。
- 存储：节点系统盘和数据盘将一同转换为包年包月模式。

6. 勾选**已阅读并同意按量计费转包年包月规则**，单击**确定**。

常见问题

计费模式转换过程中，是否会影响业务的正常运行?

不会，计费模式转换仅涉及订单流程，不会对节点进行资源变更或操作。

为什么计费模式转换按钮不可点击?

为保障转换的一致性，转换操作需要满足上述操作限制。请确保所有节点的状态为正常，并关闭节点池的弹性伸缩功能。只有在满足这些限制条件后，计费模式转换按钮才会变为可点击状态。

原生节点生命周期

最近更新时间：2023-05-17 15:40:24

生命周期状态说明

状态	说明
健康	节点正常运行，并连接上集群。
异常	节点运行异常，未连接上集群。
创建中	节点正在创建，未连接上集群。创建中的节点完成 **购买机器、安装组件、节点注册** 动作后将正常连接集群。
驱逐中	节点正在驱逐 Pod 到其他节点。
重启中	节点正在重启，此时无法连接集群，不允许新的 Pod 调度到该节点。
已封锁	节点已被封锁，不允许新的 Pod 调度到该节点。

新建原生节点

最近更新时间：2024-11-29 17:26:43

本文向您介绍如何通过控制台和 YAML 创建原生节点。

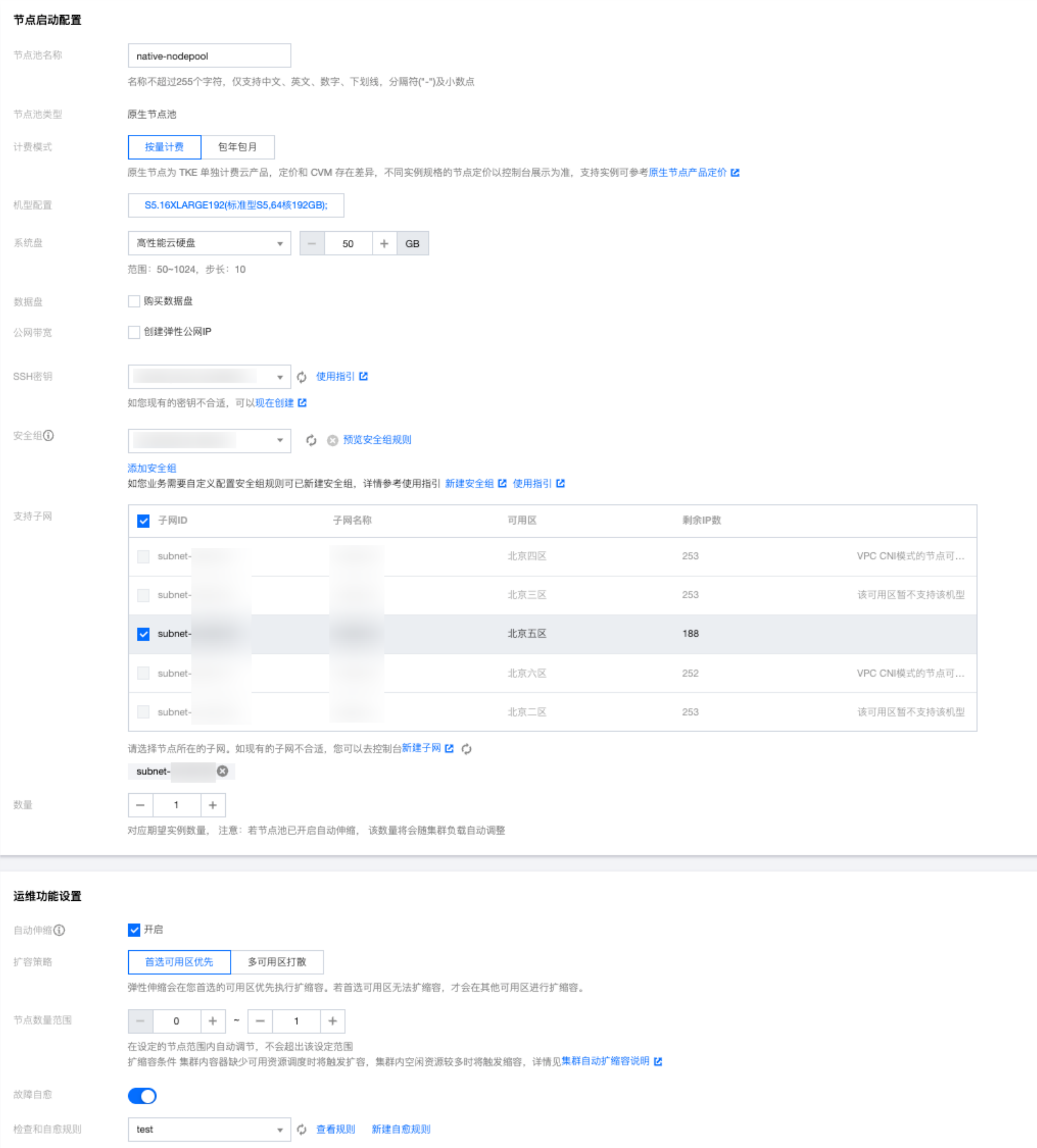
前提条件

- 已登录 容器服务控制台 。
- 已创建 TKE 标准集群。如未创建，请参考 快速创建一个标准集群 。

> **说明：**
> 原生节点仅支持通过**节点池**管理。

通过控制台创建

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理 > Worker 节点**，在**节点池**中单击**新建**。
4. 选择节点类型为**原生节点**，单击**创建**。
5. 在**新建**页面，根据 创建参数说明 提示进行设置，如下图所示：



6. （可选）单击**高级设置**，查看或配置更多信息，如下图所示：

7. 单击**创建节点池**即可。

通过 YAML 创建

原生节点池的 kubernetes 资源如下所示，YAML 字段详情请参见 创建参数说明：

```yaml
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
spec:
  autoRepair: false #故障自愈开关
  displayName: test
  healthCheckPolicyName:  #自愈规则名称
  instanceTypes: #机型规格
  - S5.MEDIUM2
  replicas: 1  #节点数
  scaling:  #自动扩缩容策略
    createPolicy: ZonePriority
    maxReplicas: 1
  subnetIDs:  #节点池子网
  - subnet-nnwwb64w
  template:
    metadata:
      annotations:
        node.tke.cloud.tencent.com/machine-cloud-tags: '[{"tagKey":"xxx","tagValue":"xxx"}]'  #腾讯云标签
    spec:
      displayName: tke-np-mpam3v4b-worker  #可自定义显示名称
      deletePolicy: Random #移出策略
      metadata:
        annotations:
          annotation-key1: annotation-value1  #自定义annotations
        labels:
          label-test-key: label-test-value    #自定义labels
      providerSpec:
        type: Native
        value:
```

```
        dataDisks:    #数据盘参数
        - deleteWithInstance: true
          diskID: ""
          diskSize: 50
          diskType: CloudPremium
          fileSystem: ext4
          mountTarget: /var/lib/containerd
        instanceChargeType: PostpaidByHour    #节点付费模式
        keyIDs:    #节点登录SSH参数
        - skey-xxx
        lifecycle:    #自定义脚本
          postInit: echo "after node init"
          preInit: echo "before node init"
        management:    #management参数设置，包含kubelet\kernel\nameserver\hostname
        securityGroupIDs:    #安全组配置
        - sg-xxxxx
        systemDisk:    #系统盘配置
          diskSize: 50
          diskType: CloudPremium
    runtimeRootDir: /var/lib/containerd
    taints:    #污点 非必填
    - effect: NoExecute
      key: taint-key2
      value: value2
type: Native
```

创建参数说明

启动配置

参数项	YAML 字段	说明
节点池类型（必填）	字段名：spec.type 字段值：Native	Native 代表原生节点池。
节点池名称（必填）	字段名：spec.displayname 字段值：demo-machineset（自定义）	自定义，可根据业务需求等信息进行命名，方便后续资源管理。
计费模式（必填）	字段名： spec.template.spec.providerSpec.value.instanceChargeType 字段值：PostpaidByHour（按量）/ PrepaidCharge（包月）	提供按量计费、包年包月两种计费模式，请根据实际需求进行选择。
机型配置（必填）	**机型：** • 字段名：spec.instanceTypes • 字段值：S2.MEDIUM4（可参考控制台获取其他机型规格） **系统盘：** • 字段名： spec.template.spec.providerSpec.value.systemDisk.diskSize/diskType • 字段值： diskSize: 50（支持自定义，大小需为10 的倍数，最小为 50G） diskType: CloudPremium / CloudSSD（系统盘类型，支持高性能 / SSD）	在“机型配置”窗口中参考以下信息按需选择： • 可用区：该选项用于过滤所选可用区下可用的实例类型。详情见 原生节点地域与可用区 。 • 机型：支持通过 CPU 核数、内存大小及实例类型进行筛选。 • 系统盘：存储控制、调度节点运行的系统集合，新建大小设置 > 100GB。

数据盘	字段名：spec.template.spec.providerSpec.value.dataDisks 字段值： • diskSize: 同系统盘 • diskType: 同系统盘 • fileSystem: ext3/ext4/xfs • mountTarget: /var/lib/containerd（挂载路径）	节点上挂载的首块数据盘（仅考虑云硬盘类型）格式化挂载路径默认为 /var/lib/container，此时系统将 “Kubelet/容器/镜像”的存储目录（/var/lib/kubelet、/var/lib/containerd、/var/lib/docker）都默认挂载到该路径下。节点上的其他数据盘，您可以自定义该盘的挂载路径。
公网带宽	字段名：spec.template.spec.providerSpec.value.internetAccessible 字段值：详情见 原生节点开启公网访问。	公网带宽：开启公网访问需绑定 EIP，详情见 原生节点开启公网访问。
主机名	展示字段：metadata.annotation key: "node.tke.cloud.tencent.com/hostname-pattern" value: "自定义"	操作系统内部的计算机名，默认为内网IP，节点 hostname 命名和主机名将保持一致。 命名规则： 1. 支持批量连续命名或指定模式串命名，要求 2-40 个字符，仅支持小写字母、数字、连字符 "-" 、点号 "."， 符号不能用于开头或结尾且不能连续使用。 2. 主机名组成形式为：自定义主机名(hostname) + “.节点池id” + ".{R:k}"，k 为本次活动内已经创建的实例数量。如：自定义为 “work{R:2}” ，则批量扩容主机名为 “work.节点池id.0” 、 “work.节点池id.1” 以此类推。 注意：当集群 “节点hostname命名模式” 为手动模式时，节点池创建时才会展示该字段。
ssh 密钥（必填）	字段名：spec.template.spec.providerSpec.value.keyIDs 字段值：skey-asxxxx（ssh 密钥 ID）	需通过 ssh 方式登录节点，若现有的密钥不合适，可以 创建密钥。
安全组（必填）	字段名：spec.template.spec.providerSpec.value.securityGroupIDs 字段值：sg-a7msxxx（安全组 ID）	默认为创建集群时所设置的安全组，若现有的安全组不合适，可以 创建安全组。
数量（必填）	字段名：spec.replicas 字段值：7（自定义）	对应节点池内维护的期望节点数量，请根据实际需求进行设置。例如，此处数量填写为5，则节点池将创建5台节点维护。
容器网络（必填）	字段名：spec.subnetIDs 字段值：subnet-i2ghxxxx（容器子网 ID）	请根据实际需求选择合适的可用子网。 1. 当您手动调节节点数量时，系统将按照子网列表顺序来尝试创建节点，如果顺序靠前的子网可以创建成功，则总在该子网创建。 2. 若节点池开启了自动伸缩，将根据您配置的扩容策略选择合适的子网进行创建节点。

运维功能

参数项	YAML 字段	说明
故障自愈	字段名：spec.autoRepair 字段值：true（开启）/ false（关闭）	可选项，推荐开启。该功能可实时检测原生节点上的各种异常情况并提供一定的自愈手段，包含：OS、Runtime、kubelet 异常等。
检查和自愈规则	字段名：spec.healthCheckPolicyName 字段值：test-all（绑定故障自愈 CR 名称）	您可为当前节点池选择不同的故障自愈规则，每个节点池支持绑定一个规则。
自动伸缩	字段名：spec.scaling	开启后 CA 组件对该类节点池进行自动伸缩。 备注：原生节点的自动伸缩功能由容器平台自研实现，普通节点的自动伸缩功能依赖云产品 弹性伸缩。

节点数量范围	字段名：spec.scaling.maxReplicas / minReplicas 字段值： maxReplicas: 7（自定义） minReplicas: 2（自定义）	节点池内的节点数量受限于该范围内的最小值/最大值，若节点池开启了自动伸缩，原生节点数量将在设定的范围内自动调节。
扩容策略	字段名：spec.scaling.createPolicy 字段值：ZonePriority（首选可用区优先）/ ZoneEquality（多可用区打散）	1. 首选可用区优先：弹性伸缩会在您首选的可用区优先执行扩缩容。若首选可用区无法扩缩容，才会在其他可用区进行扩缩容。 2. 多可用区打散：在伸缩组指定的多可用区（即指定多个子网）之间尽最大努力均匀分配节点实例。只有配置了多个子网时该策略才能生效。
移出策略	字段名：spec.deletePolicy 字段值：Random（随机）/Newest（优先移出最新实例）/Oldest（优先移出最旧实例）	1. 随机：缩容时系统会随机选择节点池中的节点进行移除。 2. 优先移出最新实例：缩容时创建时间最晚的节点会被首先选中并移除。 3. 优先移出最旧实例：缩容时创建时间最早的节点会被首先选中并移除。

高级设置

参数项	YAML 字段	说明
云标签	字段名：node.tke.cloud.tencent.com/machine-cloud-tags 字段值：[{"tagKey":"xxx","tagValue":"xxx"}]	支持通过云标签对节点分账，节点池设置的云标签对池内所有节点生效。 注意：原生节点池会默认继承集群标签，请确保此处云标签 key 的设置不会和集群设置的标签发生冲突。
Labels	字段名：spec.template.spec.metadata.labels 字段值： key1: “value1” （label 的 key/value 为自定义）	节点属性，方便对节点进行筛选和管理，该节点池下所创建的节点均将自动增加设置的 Label。
Taints	字段名：spec.template.spec.taints 字段值：effect: NoSchedule/PreferNoSchedule/NoExecute（填写 taints 类型）	节点属性，通常与 Tolerations 配合使用，确保不符合条件的 Pod 不能调度到该节点上，该节点池下所创建的节点均将自动增加设置的 Taints。
容器目录	字段名：spec.template.spec.runtimeRootDir 字段值：/var/lib/containerd	勾选即可设置容器和镜像存储目录，例如 `/var/lib/` 。
Management	字段名： spec.template.spec.providerSpec.value.management.**kubeletArgs/kernelArgs/hosts/nameservers** 字段值： 详情请参见 Management 参数介绍 。	支持自定义配置 **Kubelet** 、**内核**、**Hosts**、**Nameservers** 参数，详情请参见 Management 参数介绍 。
自定义脚本	字段名： spec.template.spec.providerSpec.value.lifecycle.preInit/postInit 字段值： preInit: "echo hello"（节点初始化前执行脚本，自定义） postInit: "echo world"（节点初始化后执行脚本，自定义）	指定自定义数据来配置节点，提供**节点初始化前**和**节点初始化后**两个设置阶段。需确保脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的 /usr/local/qcloud/tke/PreHook（PostInit）路径查看。

修改原生节点

最近更新时间：2024-01-30 14:46:11

本文向您介绍原生节点常见参数的修改方式：

功能项	修改位置及描述
云标签	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点池基本信息模块右上角的编辑 > 在编辑参数设置中修改腾讯云标签。 **生效范围**：存量和新增节点。
机型	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点启动配置信息模块 > 编辑机型。 **生效范围**：新增节点。 **注意：** 主机型不支持删除，您可添加规格相同的其他备选机型。 1. 同一节点池最多只可选择10种机型（包含主机型）。 2. 当前列表可选的实例类型会根据节点池子网所在的可用区以及现网资源余量做过滤。 3. 节点池主机型为 GPU 类实例，则不支持添加非 GPU 类实例作为备选机型。
系统盘	控制台暂不支持修改系统盘类型和容量，您可通过 kubectl 编辑节点池对应 machinesets 对象中的相关字段（详情请参见 创建参数说明 ），修改后仅针对新增节点生效。 **注意：**存量节点不支持修改系统盘类型和容量。
数据盘	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点启动配置信息模块 > 编辑数据盘。 **生效范围**：新增节点。
公网带宽	控制台暂不支持修改公网带宽绑定，您可通过 kubectl 编辑节点池对应 machinesets 对象中的相关字段（详情请参见 开启公网访问 ），修改后仅针对新增节点生效。 **注意：**存量节点不支持修改公网开启状态。
安全组	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点启动配置信息模块 > 编辑安全组。 **生效范围**：若您勾选了“存量更新”字段，则本次安全组更改（包括删除、更新、追加）会对节点池内全量节点（包括存量和增量）生效，请谨慎操作；否则只针对新增节点生效。
主机名	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点启动配置信息模块 > 编辑主机名。 **生效范围**：新增节点。 **注意：** 1. 主机命名模式默认和集群维度“节点 hostname” 命名模式保持一致。 2. 当集群维度“节点 hostname” 命名模式为自动命名时，当前参数不支持修改，默认以内网 IP 来命名主机和节点 hostname。 3. 当集群维度“节点 hostname” 命名模式为手动命名时，当前参数支持修改，您可在内网IP和自定义命名之间切换。
支持子网	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择节点启动配置信息模块 > 编辑子网。 **生效范围**：新增节点。
弹性伸缩	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择运维信息模块右上角的编辑 > 修改弹性伸缩。 **生效范围**：新增节点。 **注意：**使用原理和操作限制详情请参见 节点扩缩容 。
qGPU	**位置描述**：登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择运维信息模块右上角的编辑 > 修改 qGPU。 **生效范围**：新增节点。

	注意： qGPU 的开关能否开启和机型、驱动相关，详情请参见 使用qGPU 。
Label/Taint/Annotation	**位置描述：** 登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择参数设置模块 > 编辑 Label/Taint/Annotation。 **生效范围：** 若您勾选了“存量更新”字段，则本次对 Label/Annotation/Taint 的更改（包括删除、更新、追加）会对节点池内全量节点（包括存量和增量）生效；否则只针对新增节点生效。
Kubelet/Kernel/Nameserver/Host	**位置描述：** 登录 容器服务控制台 > 选择集群 > 选择节点管理 > 选择 Worker 节点 > 选择节点池页签 > 选择节点池 > 在详情页中选择参数设置模块 > 编辑 Management。 **生效范围：** 若您勾选了“存量更新”字段，本次对 Management 的更改（包括删除、更新、追加）会对节点池内全量节点（包括存量和增量）生效，请谨慎操作；否则只针对新增节点生效。 **注意：** 某些 Kubelet 参数对存量节点生效时会触发业务 Pod 重启，建议评估风险后谨慎操作，详情请参见 Management参数介绍 。

删除原生节点

最近更新时间：2024-06-12 10:32:42

本文向您介绍如何从节点池中删除原生节点，减少节点池管理的节点数目。

注意事项

- 原生节点按照节点池维度分组管理，不支持从节点池移出到集群。
- 原生节点**删除即销毁**，节点资源彻底释放，不支持恢复，建议您谨慎操作。

删除指定原生节点

按量计费类型

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理 > Worker 节点 > 节点池**，在节点池列表单击节点池 ID，进入节点池详情页。
4. 在节点列表中，选择目标节点单击**删除**，在弹窗中勾选“调整期望实例数”（勾选则代表删除当前节点后同时将节点期望数量 -1，否则节点池会通过继续扩容来维持当前期望实例数量）。

5. 单击**确定**，即可完成节点删除。

> **说明：**
> 按量计费的节点池，未指定具体节点删除，选择直接通过“调整数量”调小节点数量时，后台会**随机删除**多余节点副本。

包年包月类型

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理 > Worker 节点 > 节点池**，在节点池列表单击节点池 ID，进入节点池详情页。
4. 在节点列表中，单击**调整数量**，在弹窗中调小节点数量。在弹出的确认窗口和退费窗口中分别单击**确定**，即可完成包月节点删除。例如，当前节点池内的包月的原生节点数为5，您希望指定删除2个包月节点。首先需要调整数量为3，其次从当前节点列表中选择2个包月节点进行删除。
5. 在节点列表中，您也可以选择或批量选择待删除的包年包月节点，单击**删除**。

> **说明：**
> 包年包月类型节点因涉及到节点退费，因此无法像按量计费节点一样随机删除，需要您在删除前完成弹窗信息确认。
> 包年包月类型节点到期前退费会按照按量计费价格进行折算，请您谨慎操作。

节点关联资源处理

- 机器和系统盘：原生节点删除后，机器和系统盘资源将彻底销毁，不支持恢复，请您谨慎操作！

- EIP：如果您的节点在创建时开启了公网带宽，则绑定的 EIP 会在节点删除时一同销毁，不支持恢复，请您谨慎操作！
- 数据盘：如果您的节点在创建时绑定了数据盘，则会在节点删除时一同销毁，不支持恢复，建议在删除前提前备份数据！

服务升级指南

最近更新时间：2024-11-25 16:42:22

本文主要介绍如何将集群中的普通节点通过“服务升级”功能切换至原生节点，以便在不更换 CVM 机器资源的情况下享受 7*24 小时成本优化服务。

> **说明：**
> - 该功能仅适用于特定机型，包括 BMS5、HCCPNV5、HCCPNV5v、HCCPNV4sne、HCCPNV4ane 和 HCCPNV4sn。
> - 在执行升级操作之前，请务必谨慎操作，因为升级是不可逆的行为。

特性对比

参数项	普通节点	服务升级后的原生节点
收费说明	CVM 资源费用	CVM 资源费用 + 原生节点增值服务费
操作系统	公共或自定义镜像	原生节点专用镜像
Request 推荐	支持	支持，为 Kubernetes 的 Workload 智能推荐容器级别资源的 Request/Limit 数值进而减少资源浪费，详情请参见 Request 智能推荐 。
节点放大	不支持	支持，通过虚拟放大节点的容量提升资源利用率，详情请参见 原生节点专用调度器产品介绍 。
可抢占式 Job	不支持	支持，使用集群闲置资源，不占用集群/节点真实的剩余可调度量，详情请参见 可抢占式 Job 。
内存压缩	不支持	支持，通过将低频访问的内存换出来降低整机的内存消耗，详情请参见 内存压缩使用说明 。
在离线混部	不支持	支持，提供 CPU/内存/网络/磁盘 IO 隔离能力和优先级抢占能力来提升资源利用率，详情请参见 QoSAgent 。
qGPU	不支持	支持，支持多个容器共享 GPU 卡并支持容器间算力和显存精细隔离，详情请参见 qGPU 概述 。
原地升降配	不支持	支持，在不重启 Pod 的情况下修改 CPU/内存的 request/limit 值，详情请参见 Pod 原地升降配 。
故障自愈	不支持	支持，实时检测需要人为干预解决的持续性故障，详情请参见 故障自愈规则 。
内核/kubelet 运维参数管理	不支持	支持，详情请参见 Management 参数介绍 。
MachineSet/Machine 声明式管理	不支持	支持，详情请参见 声明式操作实践 。

操作步骤

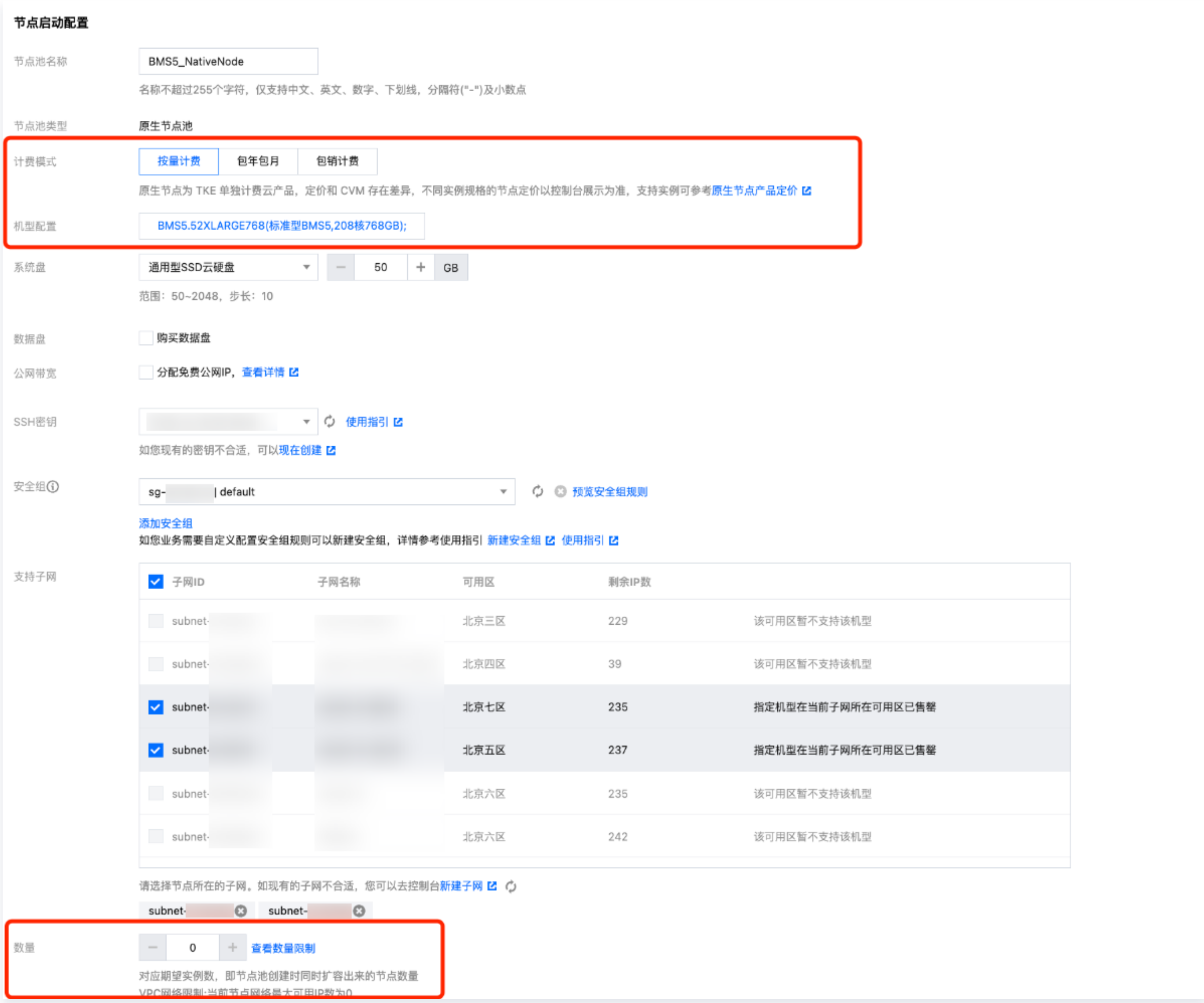
本文以按量计费的普通节点 BMS5 机型为例，介绍如何升级为原生节点管理。

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理 > Worker 节点**，进入节点管理页面。

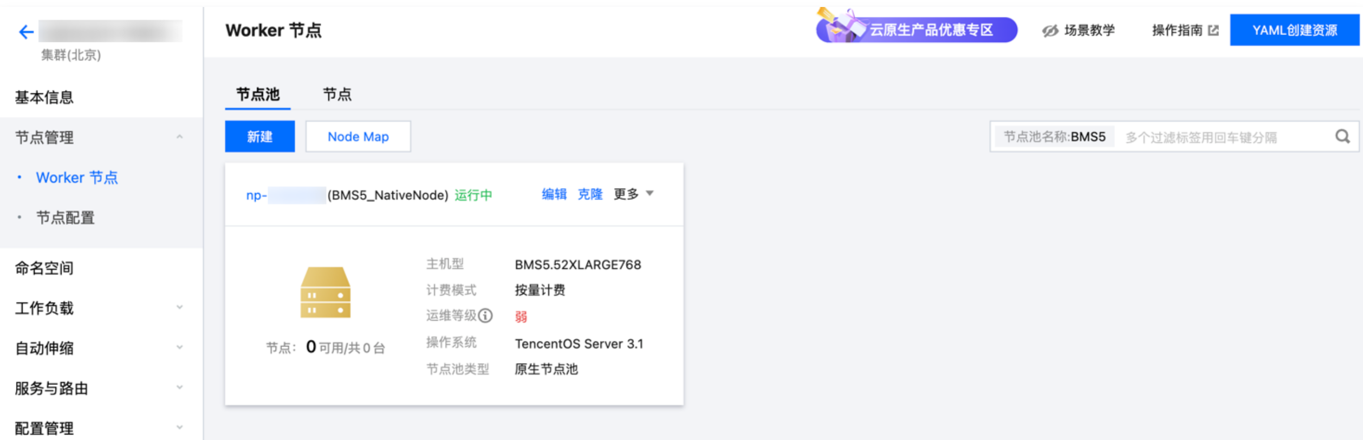
步骤1：创建原生节点池

原生节点通过节点池统一管理，服务升级前需要您先创建空（无实际机器资源）的原生节点池，方便在服务升级时将普通节点加入该池子管理。

1. 在 worker 节点页面点击**新建**按钮，在弹窗中选择节点类型为**“原生节点”**。
2. 在节点池创建页面填写相关参数，请特别注意以下参数的填写说明，其他参数的介绍请参见 新建原生节点 。
   - 计费模式：**原生节点池的计费模式不支持修改，节点池的计费模式务必与待升级普通节点的计费模式保持一致。**
   - 数量：数量为0代表仅创建资源数为空的节点池模板（无资源创建则不会产生任何费用），否则在创建节点池时会自动创建出指定数量的 CVM 资源（同时收取 CVM 资源费和原生节点增值服务费）。**若您仅计划将已有普通节点进行服务升级，推荐数量设置为0。**

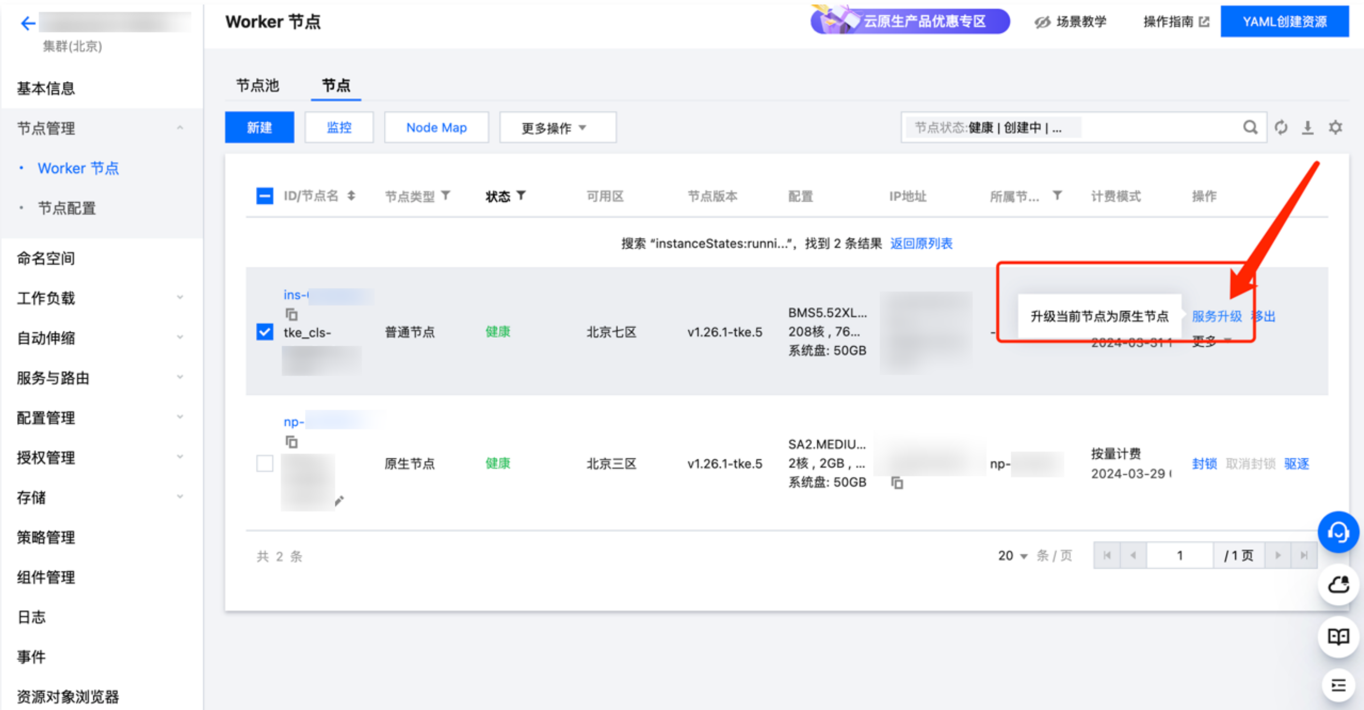


3. 单击**创建节点池**。如下图所示：



步骤2：选择目标普通节点

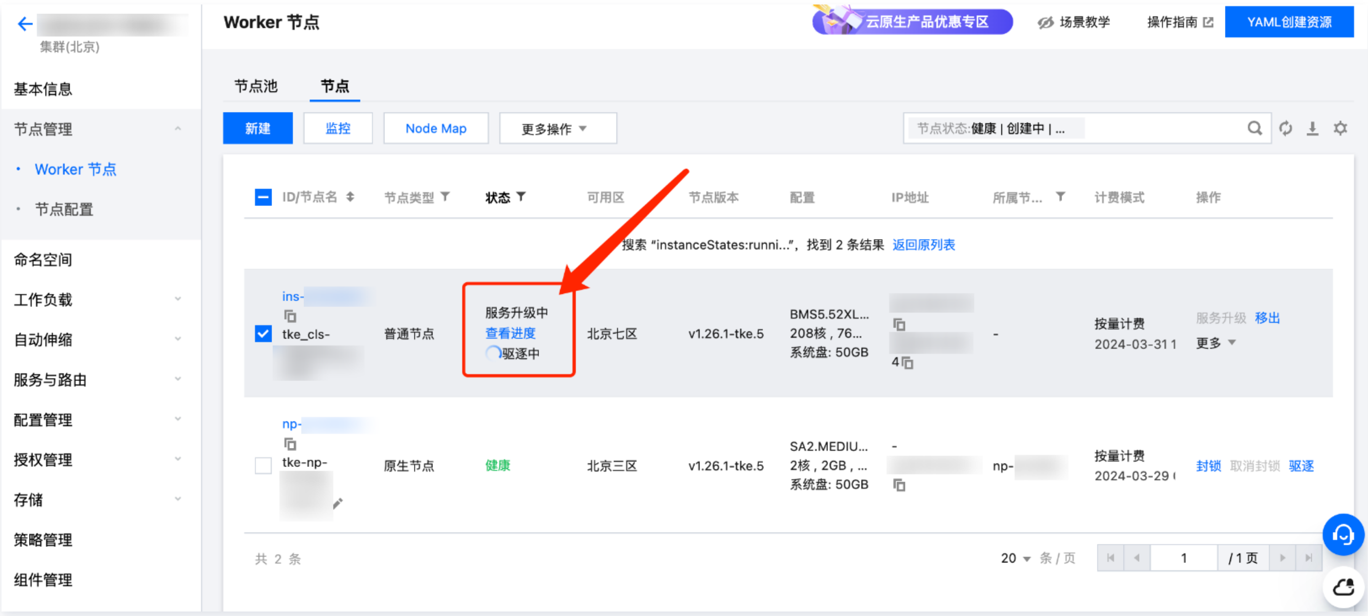
1. 进入节点列表页面。
2. 选择目标普通节点，在操作栏单击**服务升级**。如下图所示：



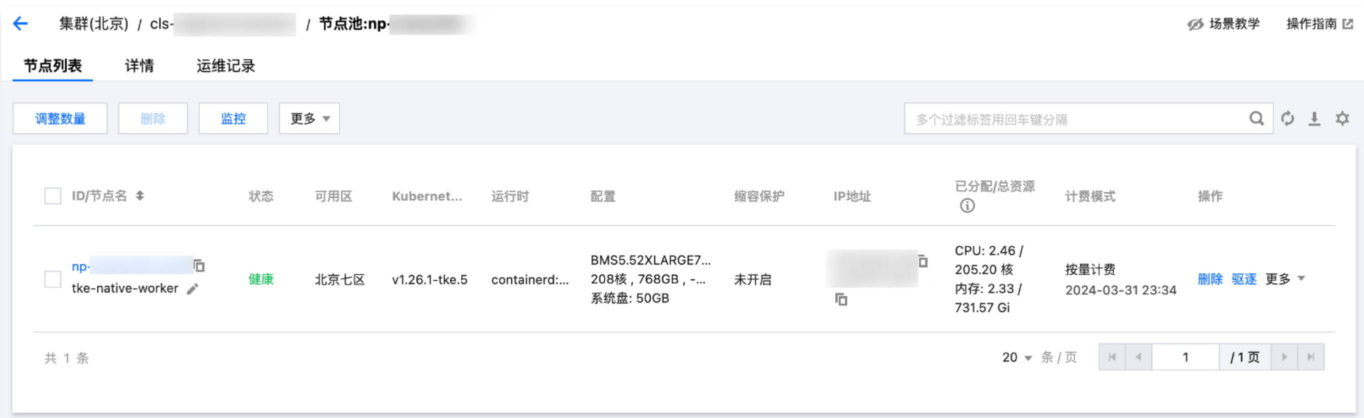
步骤3：服务升级

1. 在**服务升级**弹窗中的“信息确认”环节，选择待升级节点要加入的原生节点池，如下图所示：

   - 系统会校验该节点和目标节点池之间的计费模式和规格（对应机型的 CPU/内存）是否一致。
   - 每个节点会产生对应的增值服务费，计费模式和资源保持一致。
2. 单击**下一步**进入“服务升级”确认页，请特别注意以下参数。
   - **服务升级需重装节点镜像为原生节点专有镜像**，建议您提前封锁并排空节点。
   - 升级过程中将按照所选节点池的参数配置对节点初始化，**其中系统盘会重装，数据盘数据/公网 ip（若开启）/子网信息会被保留，安全组/ssh 密钥/kubelet 参数等高级参数会按照节点池填写值进行设置。**
   - 完成升级后您仍可在 CVM 控制台查看机器详情，为保障信息一致性，建议在 TKE 平台操作节点。
   - 服务升级流程预估15分钟左右，单击**确认升级**后可刷新页面在状态栏获取进度。
3. 单击**完成**进行服务升级，您可在节点列表的状态栏查看升级进度。如下图所示：



4. 升级完成后，您可在目标节点池中查看并管理该原生节点，此时节点具备所有增值功能。如下图所示：



故障自愈规则

最近更新时间：2024-02-18 10:05:21

功能概述

基础设施的不稳定性、环境的不确定性经常会引发不同维度的系统故障。为了将工作人员从繁重的运维事务中解放出来，腾讯云容器服务团队自研故障自愈功能来帮助运维人员快速定位问题，并通过预置平台运维经验，针对不同检测项提供最小化的自愈动作。该能力在 NPD Plus 组件的基础上进一步扩展，具体包含如下特性：

- 系统实时检测需要人为干预解决的持续性故障。
- 故障范围涵盖操作系统、K8s 环境、运行时等数十种检测项。
- 通过预置专家经验（执行修复脚本、重启组件）来对故障进行快速响应。

检测项介绍

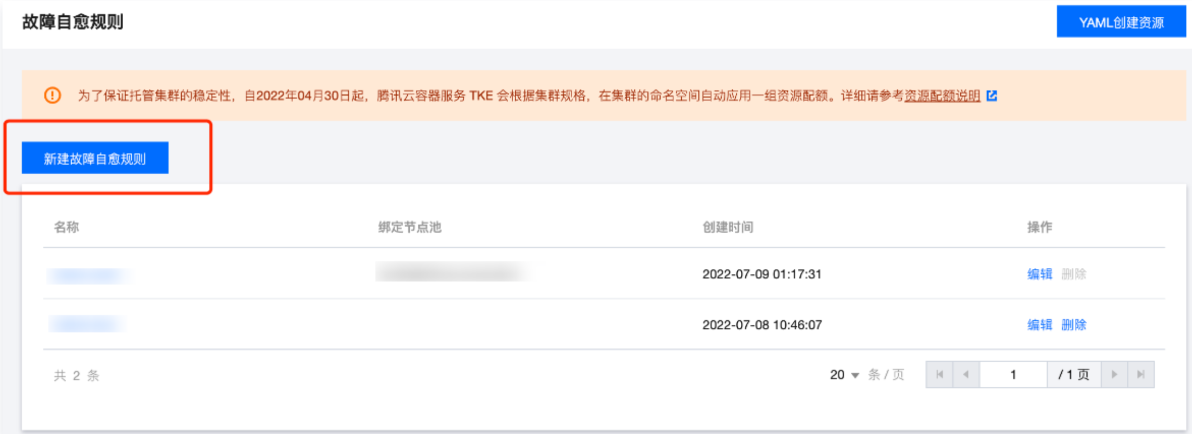
检测项	描述	风险等级	自愈动作
FDPressure	Too many open files（查看主机的文件描述符数量是否达到最大值的 90%）	low	-
RuntimeUnhealthy	List containerd task failed	low	RestartRuntime
KubeletUnhealthy	Call kubelet healthz failed	low	RestartKubelet
ReadonlyFilesystem	Filesystem is readonly	high	-
OOMKilling	Process has been oom-killed	high	-
TaskHung	Task blocked more then beyond the threshold	high	-
UnregisterNetDevice	Net device unregister	high	-
KernelOopsDivideError	Kernel oops with divide error	high	-
KernelOopsNULLPointer	Kernel oops with NULL pointer	high	-
Ext4Error	Ext4 filesystem error	high	-
Ext4Warning	Ext4 filesystem warning	high	-
IOError	IOError	high	-
MemoryError	MemoryError	high	-
DockerHung	Task blocked more then beyond the threshold	high	-
KubeletRestart	Kubelet restart	low	-

为节点开启故障自愈功能

通过控制台操作

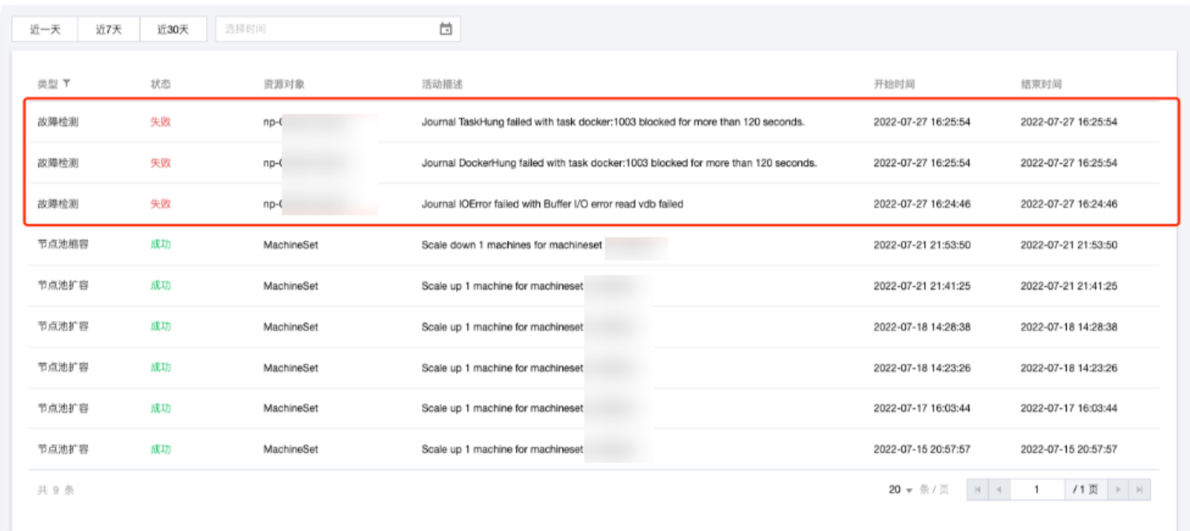
1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理** > **节点配置** > **故障自愈**，进入故障自愈规则页面。

4. 单击**新建故障自愈规则**，创建新的故障自愈规则。如下图所示：



5. 创建完成后，返回节点池列表页。
6. 单击节点池 ID，进入节点池详情页。
7. 在节点池详情页的“运维信息”模块，单击**编辑**为节点池开启故障自愈能力。如下图所示：

8. 开启后，可以在“运维记录”中查看实时故障检测详情，状态为“失败”则代表该检测项未通过。



通过 YAML 操作

1. 新建故障自愈规则

根据命令 `kubectl create -f demo-HealthCheckPolicy.yaml` 集群中创建自愈规则，YAML 配置如下：

```
apiVersion: config.tke.cloud.tencent.com/v1
kind: HealthCheckPolicy
metadata:
  name: test-all
  namespace: cls-xxxxxxxx（集群 id）
spec:
  machineSetSelector:
    matchLabels:
      key: fake-label
  rules:
  - action: RestartKubelet
    enabled: true
    name: FDPressure
  - action: RestartKubelet
    autoRepairEnabled: true
    enabled: true
    name: RuntimeUnhealthy
  - action: RestartKubelet
    autoRepairEnabled: true
    enabled: true
    name: KubeletUnhealthy
  - action: RestartKubelet
    enabled: true
    name: ReadonlyFilesystem
  - action: RestartKubelet
    enabled: true
    name: OOMKilling
  - action: RestartKubelet
    enabled: true
    name: TaskHung
  - action: RestartKubelet
    enabled: true
    name: UnregisterNetDevice
  - action: RestartKubelet
    enabled: true
    name: KernelOopsDivideError
  - action: RestartKubelet
    enabled: true
    name: KernelOopsNULLPointer
  - action: RestartKubelet
    enabled: true
    name: Ext4Error
  - action: RestartKubelet
    enabled: true
    name: Ext4Warning
  - action: RestartKubelet
    enabled: true
    name: IOError
  - action: RestartKubelet
    enabled: true
    name: MemoryError
  - action: RestartKubelet
    enabled: true
    name: DockerHung
  - action: RestartKubelet
    enabled: true
    name: KubeletRestart
```

```
```

2. 开启自愈开关

在 MachineSet 中指定字段 `healthCheckPolicyName: test-all`，YAML 配置如下：

```yaml
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
spec:
  type: Hosted
  displayName: demo-machineset
  replicas: 2
  autoRepair: true
  deletePolicy: Random
  healthCheckPolicyName: test-all
  instanceTypes:
  - C3.LARGE8
  subnetIDs:
  - subnet-xxxxxxxx
  - subnet-yyyyyyyy
......
```

声明式操作实践

最近更新时间：2024-08-23 16:54:21

kubectl 支持操作

CRD 类型	操作项
MachineSet	创建原生节点池 `kubectl create -f machineset-demo.yaml`
	查看原生节点池列表 `kubectl get machineset`
	查看原生节点池 YAML 详情 `kubectl describe ms machineset-name`
	删除原生节点池（包年包月节点池需要在控制台删除） `kubectl delete ms machineset-name`
	扩容原生节点池 `kubectl scale --replicas=3 machineset/machineset-name`
Machine	查看原生节点 `kubectl get machine`
	查看原生节点 YAML 详情 `kubectl describe ma machine-name`
	删除原生节点 `kubectl delete ma machine-name`
HealthCheckPolicy	创建故障检测自愈规则 `kubectl create -f demo-HealthCheckPolicy.yaml`
	查看故障自愈规则列表 `kubectl get HealthCheckPolicy`
	查看故障自愈规则 YAML 详情 `kubectl describe HealthCheckPolicy HealthCheckPolicy-name`
	删除故障自愈规则 `kubectl delete HealthCheckPolicy HealthCheckPolicy-name`

通过 YAML 使用 CRD

MachineSet

原生节点池参数填写可参考 原生节点创建参数说明。

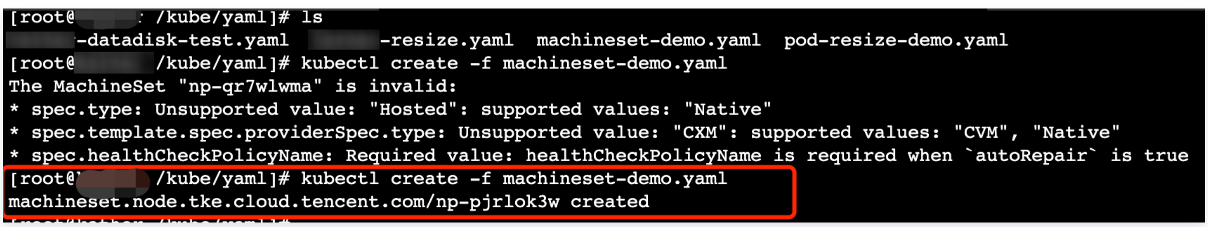
```
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
spec:
  autoRepair: false #故障自愈开关
  displayName: test
  healthCheckPolicyName:  #自愈规则名称
  instanceTypes: #机型规格
  - S5.MEDIUM2
  replicas: 1  #节点数
  scaling:  #自动扩缩容策略
```

```yaml
  createPolicy: ZonePriority
  maxReplicas: 1
subnetIDs:  #节点池子网
- subnet-nnwwb64w
template:
  metadata:
    annotations:
      node.tke.cloud.tencent.com/machine-cloud-tags: '[{"tagKey":"xxx","tagValue":"xxx"}]'  #腾讯云标签
  spec:
    displayName: tke-np-mpam3v4b-worker  #可自定义显示名称
    metadata:
      annotations:
        annotation-key1: annotation-value1  #自定义annotations
      labels:
        label-test-key: label-test-value    #自定义labels
    providerSpec:
      type: Native
      value:
        dataDisks:    #数据盘参数
        - deleteWithInstance: true
          diskID: ""
          diskSize: 50
          diskType: CloudPremium
          fileSystem: ext4
          mountTarget: /var/lib/containerd
        instanceChargeType: PostpaidByHour    #节点付费模式
        keyIDs:    #节点登录SSH参数
        - skey-xxx
        lifecycle:    #自定义脚本
          postInit: echo "after node init"
          preInit: echo "before node init"
        management:    #management参数设置，包含kubelet\kernel\nameserver\hostname
        securityGroupIDs:    #安全组配置
        - sg-xxxxx
        systemDisk:    #系统盘配置
          diskSize: 50
          diskType: CloudPremium
    runtimeRootDir: /var/lib/containerd
    taints:    #污点 非必填
    - effect: NoExecute
      key: taint-key2
      value: value2
type: Native
```

kubectl 操作 demo

MachineSet

1. 执行命令 `kubectl create -f machineset-demo.yaml` 根据上述 YAML 创建 MachineSet：



```
[root@[redacted] /kube/yaml]# ls
[redacted]-datadisk-test.yaml  [redacted]-resize.yaml  machineset-demo.yaml  pod-resize-demo.yaml
[root@[redacted] /kube/yaml]# kubectl create -f machineset-demo.yaml
The MachineSet "np-qr7wlwma" is invalid:
* spec.type: Unsupported value: "Hosted": supported values: "Native"
* spec.template.spec.providerSpec.type: Unsupported value: "CXM": supported values: "CVM", "Native"
* spec.healthCheckPolicyName: Required value: healthCheckPolicyName is required when `autoRepair` is true
[root@[redacted] /kube/yaml]# kubectl create -f machineset-demo.yaml
machineset.node.tke.cloud.tencent.com/np-pjrlok3w created
```

2. 根据命令 `kubectl get machineset` 查看 MachineSet np-pjrlok3w 状态。

```
[root@[redacted] /kube/yaml]# kubectl get machineset
NAME          TYPE     STATUS    READY   AVAILABLE   DISPLAYNAME       AGE
np-14024r66   Native   Running   2/2     2           [redacted]test    9m50s
np-pjrlok3w   Native   Running   0/1     0           [redacted]_yaml_test   3m22s
```

此时控制台上已出现对应节点池，节点在创建中：

3. 根据命令 `kubectl describe machineset np-pjrlok3w` 查看 MachineSet np-pjrlok3w 描述：

```
[root@[redacted] /kube/yaml]# kubectl describe ms np-pjrlok3w
Name:         np-pjrlok3w
Namespace:
Labels:       node.tke.cloud.tencent.com/appid=[redacted]
              node.tke.cloud.tencent.com/autoscaling-enabled=true
Annotations:  cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: 3
              cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: 0
              node.tke.cloud.tencent.com/direct-eni: 0
              node.tke.cloud.tencent.com/memoryGb: 2
              node.tke.cloud.tencent.com/route-eni: 9
              node.tke.cloud.tencent.com/vCPU: 2
API Version:  node.tke.cloud.tencent.com/v1beta1
Kind:         MachineSet
Metadata:
  Creation Timestamp:  2022-08-02T02:33:37Z
  Finalizers:
    node.tke.cloud.tencent.com/finalizer
  Generation:  2
  Managed Fields:
    API Version:  node.tke.cloud.tencent.com/v1beta1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
```

4. 根据命令 `kubectl scale --replicas=2 machineset/np-pjrlok3w` 执行节点池扩缩容：

```
Status:
  Fully Labeled Replicas:   1
  Kubelet Version:          1.20.6-tke.21
  Observed Generation:      2
  Replicas:                 1
  Runtime Version:          containerd-1.4.3
[root@[redacted] /kube/yaml]# kubectl scale --replicas=2 machineset/np-pjrlok3w
machineset.node.tke.cloud.tencent.com/np-pjrlok3w scaled
```

5. 根据命令 `kubectl delete ms np-pjrlok3w` 删除节点池。

```
[root@[redacted] /kube/yaml]# kubectl scale --replicas=2 machineset/np-pjrlok3w
machineset.node.tke.cloud.tencent.com/np-pjrlok3w scaled
[root@[redacted] /kube/yaml]# kubectl delete ms np-pjrlok3w
machineset.node.tke.cloud.tencent.com "np-pjrlok3w" deleted
```

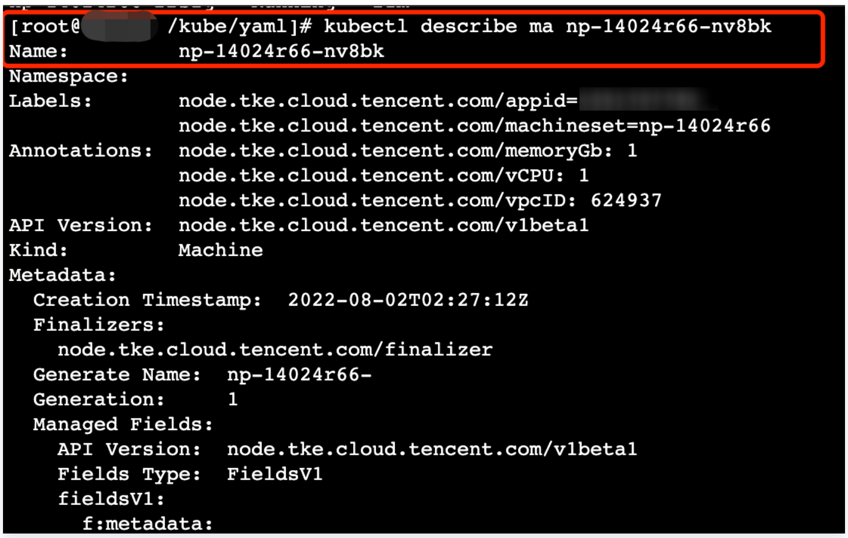
Machine

1. 根据命令 `kubectl get machine` 查看 Machine 列表：

```
[root@[redacted] /kube/yaml]# kubectl get ma
NAME                  STATUS    AGE
np-14024r66-nv8bk     Running   21m
np-14024r66-rrsfg     Running   21m
```

此时控制台上已存在对应节点：

2. 根据命令 `kubectl describe ma np-14024r66-nv8bk` 查看 Machine np-14024r66-nv8bk 描述：



3. 根据命令 `kubectl delete ma np-14024r66-nv8bk` 删除节点。

> **说明：**
> - 如果没有调整节点池期望节点数而直接删除节点，节点池会检测到实际节点数不足以满足声明式节点数量，然后会创建一个新节点并将其加入节点池。因此，我们推荐按照以下方式进行节点删除操作：
>   - 调整期望节点数：kubectl scale --replicas=1 machineset/np-xxxxx
>   - 删除对应节点：kubectl delete machine np-xxxxxx-dtjhd
> - 针对包年包月类型的节点，因为涉及到节点销毁退费，因此无法直接通过 kubectl 删除。

原生节点扩缩容

最近更新时间：2025-05-09 14:39:02

使用说明

原生节点的自动伸缩功能由容器平台自研实现，普通节点的自动伸缩功能依赖云产品 弹性伸缩 AS 。

- 原生节点池未开启自动伸缩：
  - 初始化节点数量取决于设置的**节点数**（字段名：replicas）。
  - 用户可以手动调整期望节点数，节点数上限受限于后台默认值 500 和容器子网 IP 数。
- 原生节点池开启自动伸缩：
  - 初始化节点数量取决于设置的**节点数**（字段名：replicas）。
  - 需要设置**节点数量范围**（字段名：minReplicas/maxReplicas），CA 调节当前节点池的节点数量受限于该范围。
  - 不支持用户手动调节期望实例数。

> **说明：**
> 同一时间，控制台节点池弹性伸缩只能由一个角色控制，若已启用弹性伸缩，则不能手动调整实例数量；若要手动调整实例数量，则先关闭弹性伸缩。

为节点开启自动伸缩功能

参数说明

功能项	字段名/值	描述
自动伸缩	字段名：spec.scaling	默认勾选开启，开启后 CA 组件对该类节点池进行自动伸缩。
节点数量范围	字段名：spec.scaling.maxReplicas/minReplicas 字段值：自定义	节点池内的节点数量受限于该范围内的最小值/最大值，若节点池开启了自动伸缩，原生节点数量将在设定的范围内自动调节。
扩容策略	字段名：spec.scaling.createPolicy 字段值： • ZonePriority（首选可用区优先） • ZoneEquality（多可用区打散）	• 首选可用区优先：弹性伸缩会在您首选的可用区优先执行扩缩容。若首选可用区无法扩缩容，才会在其他可用区进行扩缩容。 • 多可用区打散：在伸缩组指定的多可用区（即指定多个子网）之间尽最大努力均匀分配节点实例。只有配置了多个子网时该策略才能生效。

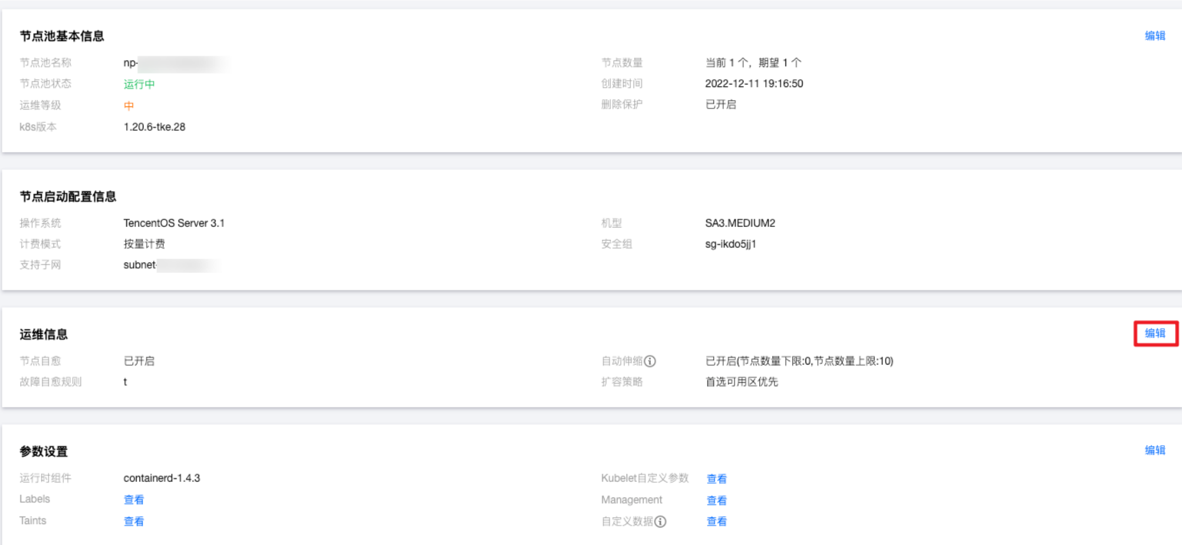
通过控制台操作

方式1：通过节点池创建页开启或关闭自动伸缩

1. 登录 容器服务控制台，在集群中新建节点池。操作详情见 新建原生节点 。
2. 在**新建**页面，勾选**开启自动伸缩**。如下图所示：

方式2：通过节点池详情页开启或关闭自动伸缩

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**中单击节点池 ID，进入节点池详情页。
4. 在节点池详情页中，单击运维信息右侧的**编辑**。如下图所示：



5. 勾选**开启弹性伸缩**。
6. 单击**确定**。

通过 YAML 操作

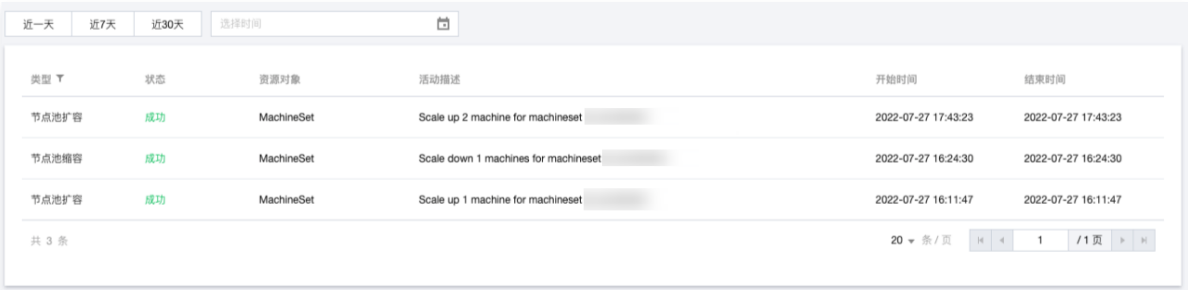
请根据参数介绍，在节点池 YAML 中填写 `scaling` 字段。

```
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
spec:
  type: Native
  displayName: mstest
  replicas: 2
  autoRepair: true
  deletePolicy: Random
  healthCheckPolicyName: test-all
  instanceTypes:
  - C3.LARGE8
  subnetIDs:
  - subnet-xxxxxxxx
  - subnet-yyyyyyyy
  scaling:
    createPolicy: ZonePriority
    minReplicas: 10
    maxReplicas: 100
  template:
    spec:
      displayName: mtest
      runtimeRootDir: /var/lib/containerd
      unschedulable: false
......
```

查看扩缩容记录

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。

3. 选择左侧菜单栏中的**节点管理**，在**节点池**中单击节点池 ID，进入节点池详情页。
4. 在运维记录页中，您可以查看节点扩缩容记录。如下图所示：



扩容原理介绍

本节将针对多机型、多子网条件下，举例说明原生节点的扩容原理。

场景1：开启弹性伸缩，扩容策略为“首选可用区优先”

算法：

1. 根据子网排列顺序，确定首选可用区。
2. 按照客户设置的机型排列顺序，依次选择机型进行扩容。每扩容一台机器后实时判断一次库存，确保机器尽量在首选可用区中扩容成功。

示例：

假设节点池设置了机型A/B（机型顺序为A优先，A库存5台，B库存3台），子网1/2/3（3个子网在不同可用区，子网1为首选），机型和子网的排列顺序在算法判断时有效，此时 CA 触发节点池扩容10台机器。后台判断流程如下：

2.1 根据子网顺序确定首选可用区所在子网为“子网1”。

2.2 按机型顺序优先选择A进行扩容，以此循环。

2.3 扩容8台节点后（5台A+3台B），此时没有资源可在子网1所中继续扩容，执行2.1步骤切换首选可用区为子网2。

场景2：开启弹性伸缩，扩容策略为“多可用区打散”

算法：

1. 根据节点池中存量节点在可用区的分布情况，确定每个可用区预计扩容的数量，使扩容后每个可用区分布的节点数量尽量一致。
2. 确定可用区后，按照客户设置的机型排列顺序，依次选择机型进行扩容。每扩容一台机器后实时判断一次库存，确保机器尽量在当前可用区中扩容成功。

示例：假设节点池设置了机型A/B（机型顺序为A优先，A库存5台，B库存3台），子网1/2/3（3个子网在不同可用区，子网1为首选），机型和子网的排列顺序在算法判断时有效，节点池中存在5台节点且部署在可用区1中，此时 CA 触发节点池扩容10台机器。后台判断流程如下：

2.1 根据存量节点的部署情况，预计要在可用区2和3中各扩容5台机器。

2.2 根据子网顺序确定当前操作的可用区子网，即“子网2”。

2.2.1 按机型顺序优先选择A进行扩容，以此循环。

2.2.2 可用区2扩容完毕后，执行2.2步骤切换当前待扩容的可用区子网为“子网3”。

场景3：关闭弹性伸缩，手动调高节点池数量

此时扩容策略默认认为“多可用区打散”，原理同场景2。

Pod 原地升降配

最近更新时间：2024-10-24 16:42:21

概述

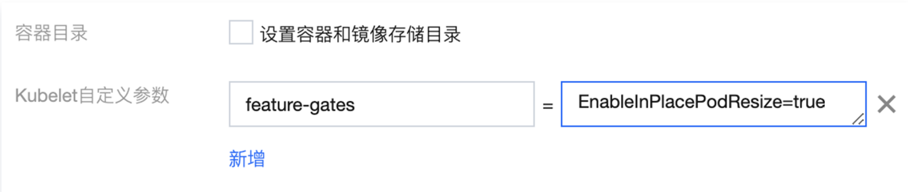
根据 Kubernetes 的设计规范，Pod 运行过程中若需要临时修改容器参数，只能更新 `PodSpec` 后重新提交，这种方式会触发 Pod 删除重建，很难满足业务侧应对流量突发时无损变配诉求。原生节点针对 Pod 的 CPU、内存提供原地升降配能力，通过对 API Server 和 Kubelet 进行升级改造，支持在不重启 Pod 的情况下修改 CPU、内存的 request/limit 值。本文主要介绍 Pod 资源原地更新功能的适用场景、工作原理和使用方式。

前提条件

- 集群支持 Kubernetes 版本在 1.16～1.26 之间的原生节点，需同时保证小版本满足以下条件：
  - kubernetes-v1.16.3-tke.30 及以上
  - kubernetes-v1.18.4-tke.28 及以上
  - kubernetes-v1.20.6-tke.24 及以上
  - kubernetes-v1.22.5-tke.15及以上
  - kubernetes-v1.24.4-tke.7 及以上
  - kubernetes-v1.26.1-tke.2 及以上

> **说明：**
> **由于和社区特性冲突，1.28、1.30版本暂未开放该功能**，如有相关需求可 提交工单 进行咨询。

- 在创建节点时设置自定义 kubelet 参数：**feature-gates** = **EnableInPlacePodResize=true**，如下图所示：



> **警告：**
> 已有节点增加当前 feature-gates 参数后会**触发节点上的 Pod 重启**，建议预先评估对业务的影响后再执行。

适用场景

1. 应对流量突发，保障业务稳定性

场景描述：动态修改 Pod 资源参数功能适用于临时性的调整，例如当 Pod 内存使用率逐渐升高，为避免触发 OOM（Out Of Memory）Killer，在不重启 Pod 的前提下提高内存的 Limit。
推荐动作：提升 CPU/ 内存的 limit 值。

2. 满足业务降本诉求，提高 CPU 利用率

场景描述：为保障线上应用的稳定性，管理员通常会预留相当数量的资源 Buffer 来应对上下游链路的负载波动，容器的 Request 配置会远高于其实际的资源利用率，导致集群资源利用率过低，造成大量资源浪费。
推荐动作：降低 CPU/ 内存的 request 值。

案例演示

验证场景

正在运行的业务 Pod，将其内存 limit 值由 128Mi 提升至 512Mi，修改后 limit 值生效且 Pod 不重建。

验证步骤

1. Kubectl 创建 pod-resize-demo.yaml 文件，YAML 内容如下所示。内存设定的 request 值为64Mi，limit 值为128Mi。

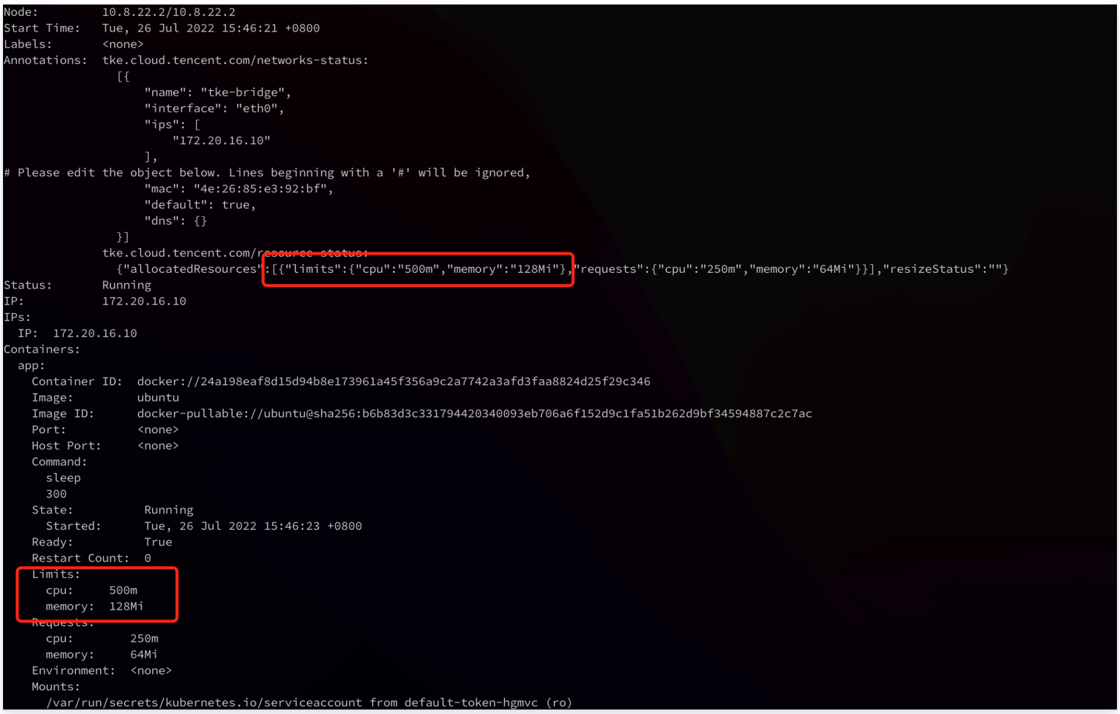
```
# Kubectl 命令:
kubectl apply -f pod-resize-demo.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo
  namespace: default
spec:
  containers:
  - name: app
    image: ubuntu
    command: ["sleep", "3600"]
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
      requests:
        memory: "64Mi"
        cpu: "250m"
```

2. 查看待变配 Pod 的 Resource 值。

```
# Kubectl 命令:
kubectl describe pod -n default demo
```

如下图所示，可变配 Pod 的 Annotation 中会有 **tke.cloud.tencent.com/resource-status** 字段，它标记了当前 Pod 实际使用资源和 Pod 的变配状态，Pod 的期望资源值会标记在每个 container 上。



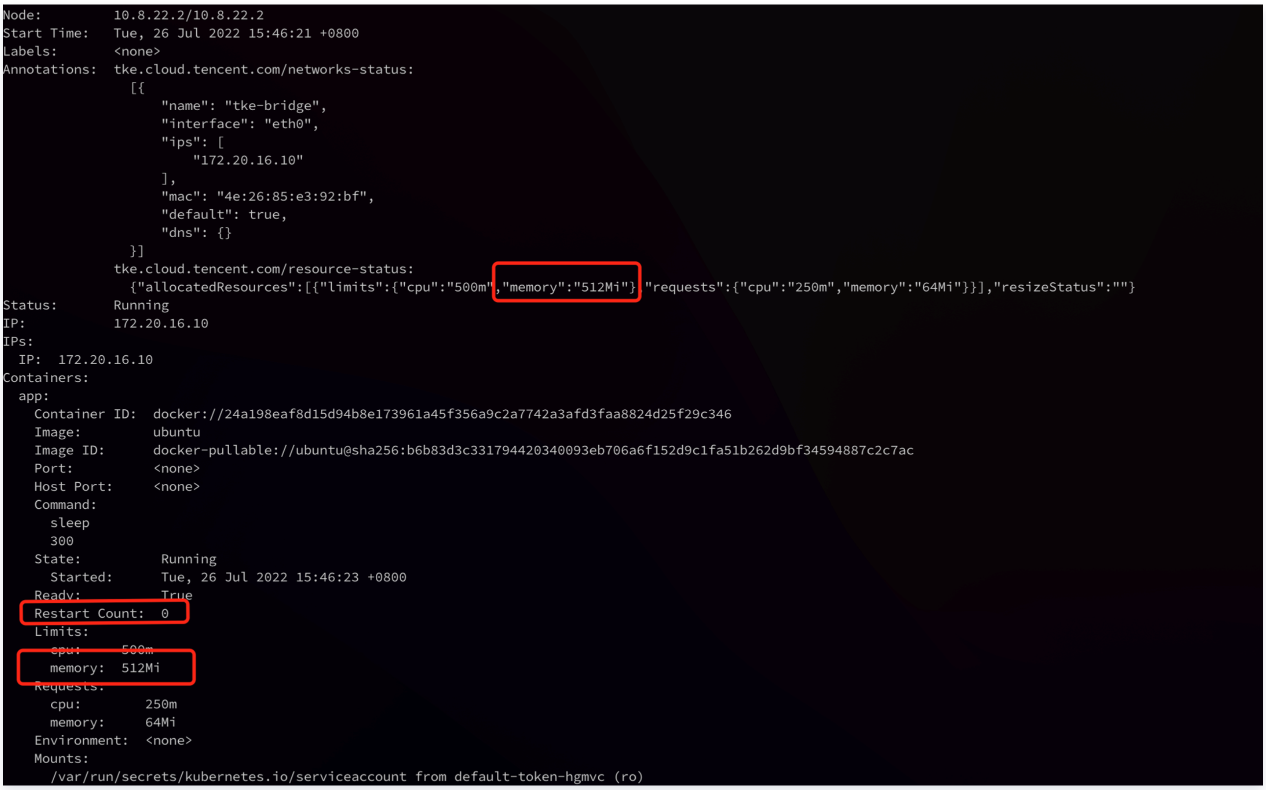
3. 更新 Pod 配置。
以提高 Pod 内存 Limit 值为例，Kubectl 修改字段 `pod.spec.containers.resources.limits.memory`（由 128Mi 提升至 512Mi）。

```
# Kubectl 命令:
kubectl edit pod -n default demo
```

4. 执行以下命令，查看 Pod 运行情况。

```
# Kubectl 命令:
kubectl describe pod -n default demo
```

如下图所示，Pod spec 中的资源和 Annotation 中的资源都变成了预期值“512Mi”，同时 Restart Count 为 0。



```
Node:         10.8.22.2/10.8.22.2
Start Time:   Tue, 26 Jul 2022 15:46:21 +0800
Labels:       <none>
Annotations:  tke.cloud.tencent.com/networks-status:
                [{
                    "name": "tke-bridge",
                    "interface": "eth0",
                    "ips": [
                        "172.20.16.10"
                    ],
                    "mac": "4e:26:85:e3:92:bf",
                    "default": true,
                    "dns": {}
                }]
              tke.cloud.tencent.com/resource-status:
                {"allocatedResources":[{"limits":{"cpu":"500m","memory":"512Mi"},"requests":{"cpu":"250m","memory":"64Mi"}}],"resizeStatus":""}
Status:       Running
IP:           172.20.16.10
IPs:
  IP:  172.20.16.10
Containers:
  app:
    Container ID:  docker://24a198eaf8d15d94b8e173961a45f356a9c2a7742a3afd3faa8824d25f29c346
    Image:         ubuntu
    Image ID:      docker-pullable://ubuntu@sha256:b6b83d3c331794420340093eb706a6f152d9c1fa51b262d9bf34594887c2c7ac
    Port:          <none>
    Host Port:     <none>
    Command:
      sleep
      300
    State:          Running
      Started:      Tue, 26 Jul 2022 15:46:23 +0800
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     500m
      memory:  512Mi
    Requests:
      cpu:        250m
      memory:     64Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-hgmvc (ro)
```

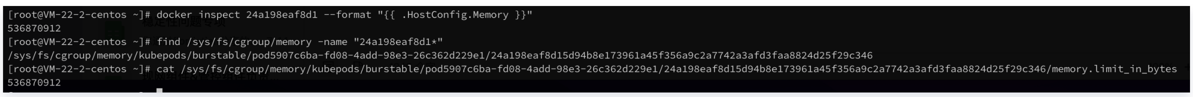
5. Pod 原地变配验证。
登录到部署了该Pod的服务器然后通过 docker 或 ctr 命令找到容器后，可以发现容器的元数据对于 memory 的限制已经被修改；同时，进入 memory cgroup 会发现对于内存的限制也被改成了期望值“512Mi”。

```
docker inspect <container-id> --format "{{ .HostConfig.Memory }}"
```

```
find /sys/fs/cgroup/memory -name "<container-id>*"
```

```
cat <container-memory-cgroup>/memory.limit_in_bytes
```

如下图所示：



```
[root@VM-22-2-centos ~]# docker inspect 24a198eaf8d1 --format "{{ .HostConfig.Memory }}"
536870912
[root@VM-22-2-centos ~]# find /sys/fs/cgroup/memory -name "24a198eaf8d1*"
/sys/fs/cgroup/memory/kubepods/burstable/pod5907c6ba-fd08-4add-98e3-26c362d229e1/24a198eaf8d15d94b8e173961a45f356a9c2a7742a3afd3faa8824d25f29c346
[root@VM-22-2-centos ~]# cat /sys/fs/cgroup/memory/kubepods/burstable/pod5907c6ba-fd08-4add-98e3-26c362d229e1/24a198eaf8d15d94b8e173961a45f356a9c2a7742a3afd3faa8824d25f29c346/memory.limit_in_bytes
536870912
```

工作原理

1. kubelet 将 Pod 当前的实际使用资源和变配状态以 json 格式存入 Annotation "tke.cloud.tencent.com/resource-status"。其中 resizeStatus 字段代表升降配状态，详情请参见 升降配状态。

```
Annotations: tke.cloud.tencent.com/resource-status:
  {"allocatedResources":[{"limits":{"cpu":"500m","memory":"128Mi"},"requests":
{"cpu":"250m","memory":"64Mi"}}],"resizeStatus":""}
```

2. `pod.spec.containers.resources` 中的资源代表期望资源，即期望分配给 Pod 的资源。当期望资源被修改后，kubelet 会尝试修改 Pod 的实际资源，并将最终结果写入 Annotation 中。

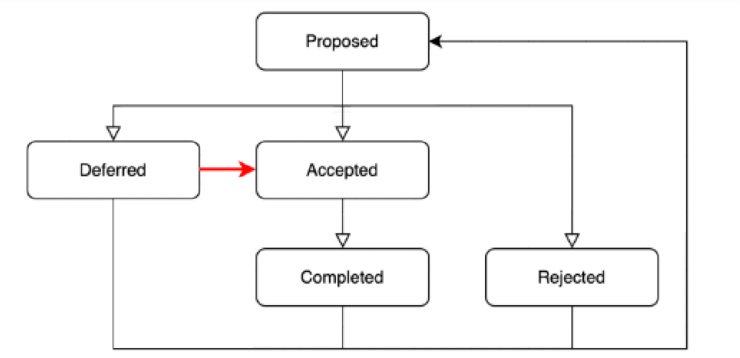
> **说明：**
> Pod 原地升降配的实现参考了社区 Kubernetes Enhancement Proposal 1287 。

升降配状态

社区在高版本中的 Pod.Status 中添加了一些字段展示变配操作的状态，该状态同时和 Kube Scheduler 配合来完成调度工作。TKE 原生节点将类似的字段放在 Pod 的 Annotation 中，其中包括 Pod 的真实资源以及当前升降配操作执行状态。

状态	描述	备注
Proposed	代表该 Pod 升降配的操作请求被提交。	-
Accepted	代表 Kubelet 发现 Pod 资源被修改，且节点上的资源足够 Admit 这个升降配后的 Pod。	-
Rejected	代表 Pod 升降配请求被驳回。	驳回原因：Pod 变配后 Request 的资源值大于节点的 Allocate 值。
Completed	代表 Pod 资源被成功修改，并变配后的资源设置在了容器上。	该状态下展示的实际值为空，即 "resizeStatus":""。
Deferred	代表由于某些问题当前升降配操作被推迟，推迟到 Pod 下次发生状态变化时再次触发变配。	可能出现的问题如下： • 当前节点资源不够：节点 Allocate 资源量 - 其他 Pod 占用资源量 < 升降配 Pod 要求资源量。 • 状态落盘失败。

执行状态如下图所示：



使用限制

为优先保障业务 Pod 运行的稳定性，需要对 Pod 原地升降配能力进行一些操作限制：

1. 只允许修改 Pod 的 CPU 和 Memory 资源。
2. 只有 `PodSpec.Nodename` 不为空的情况下才能修改 Pod 资源。
3. 资源修改范围：
   - Pod 内每个 Container 的 limit 值可以调高或者降低，降低CPU可能会导致业务降频，降低 Mem 可能失败（kubelet 会在随后的 syncLoop 中重试降低 Memory）。
   - Pod 内每个 Container 的 request 值可以调高 / 调低，但向上修改不能超过 Container 的 limit 值。
4. Container 未设置 request/limit 值场景：
   - 没有设置 limit 值的 Container 不允许设定新值。
   - 没有设置 request 值的 Container 向下变更配置时不允许低于100m。
5. 修改 request/limit 值不允许切换 QoS 类型，即不允许在 burstable/guaranteed 之间变化。升降配时需要同时修改 request 和 limit 以保证 QoS 不变。
   示例： [cpu-request: 30, cpu-limit: 50] 最多只能调整为 [cpu-request: 49, cpu-limit: 50]，禁止调整为 [cpu-request: 50, cpu-limit: 50]。

原生节点开启公网访问

最近更新时间：2024-12-06 14:19:42

> **说明：**
> 传统账户类型不支持为原生节点开启公网访问，详情请参见 账户类型说明，您可以 提交工单 进行升级。

为原生节点池开启公网访问

本节主要介绍如何通过控制台和 YAML 为节点绑定弹性公网 IP（Elastic IP，EIP）并开启公网访问。

注意事项

- 针对开启了公网访问的节点池，每个新增的原生节点将创建并绑定一个 EIP。
- EIP 和节点的生命周期保持一致，并随节点的销毁而销毁。
- 原生节点不针对 EIP 额外收费，EIP 收费模式请参考 弹性公网 IP 计费说明 。

为新增原生节点池开启公网访问

通过控制台

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧导航中**节点管理**，在**节点池**页面单击**新建**并选择**原生节点**。
4. 在**新建节点管理**页面，勾选**创建弹性公网 IP**，如下图所示：

5. 单击**创建节点池**即可。

通过 YAML

字段介绍

字段名称	字段值	含义
spec.template.spec.providerSpec.value.internetAccessible	addressType	• `EIP`：不填写则默认为常规 BGP IP，即普通 EIP。 • `HighQualityEIP`：精品 BGP IP，即精品 EIP。
	chargeType	计费模式： • `TrafficPostpaidByHour`：流量按小时后付费。 • `BandwidthPostpaidByHour`：带宽按小时后付费。 • `BandwidthPackage`：共享带宽包付费，需在 EIP 侧开通共享带宽包白名单。
	maxBandwidthOut	带宽上限，单位 Mbps。
	bandwidthPackageID	指定共享带宽包（请填写带宽包 ID）。

说明：
- 关于 EIP 类型介绍和注意事项详情请参考 申请 EIP 。
- 精品 EIP 目前账户类型仅支持**标准账户**，地域仅支持**中国香港**，计费模式仅支持**共享带宽包**。若无精品 BGP 带宽包，您可前往 私有网络控制台 > **共享带宽包**创建。

YAML 示例

```
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
metadata:
  name: np-ohh7gaek
spec:
  deletePolicy: Random
  displayName: HighQualityEIP-test
  instanceTypes:
  - SA2.MEDIUM2
  replicas: 1
  scaling:
    createPolicy: ZonePriority
    maxReplicas: 4
  subnetIDs:
  - subnet-xxxxxxx
  template:
    metadata:
      labels:
        node.tke.cloud.tencent.com/machineset: np-ohh7gaek
    spec:
      providerSpec:
        type: Native
        value:
          instanceChargeType: PostpaidByHour
          lifecycle: {}
          management:
            nameservers:
            - 183.60.83.19
            - 183.60.82.98
          metadata:
            creationTimestamp: null
          securityGroupIDs:
          - sg-5lxe2r2p
```

```
        systemDisk:
          diskSize: 50
          diskType: CloudPremium
        internetAccessible:
          chargeType: BandwidthPackage
          bandwidthPackageID: bwp-95xr2686
          maxBandwidthOut: 100
          addressType: HighQualityEIP
      runtimeRootDir: /var/lib/containerd
  type: Native
```

为已有节点池开启公网访问

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧导航中**节点管理**，在**节点池**页面单击节点池名称，进入**节点池详情页**。
4. 在**节点启动配置信息**中，单击弹性公网 IP 右侧的。如下图所示：

5. 在**设置网络配置**中，勾选**创建弹性公网 IP**。如下图所示：

> **说明：**
> 修改网络配置仅对节点池内的增量节点生效，不会影响已经创建出来的原生节点。

6. 单击**确定**。

为原生节点绑定 EIP

腾讯云原生节点支持绑定弹性公网 IP（EIP），使您的节点具备公网访问能力。通过此功能，您可以为节点池中的**存量**和**增量**节点配置公网 EIP，以满足不同场景下的网络需求。

注意事项

- **费用说明**：绑定和解绑 EIP 可能涉及额外费用，请参见 弹性公网 IP 计费说明 。
- **EIP 归属权**：解绑后的 EIP 归属权归用户所有，用户可以自行管理。
- **操作权限**：确保您具有足够的权限进行 EIP 绑定和解绑操作。

查询 EIP 状态

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧导航中**节点管理**，在**节点**页面单击节点名称，进入**节点详情页**。
4. 在主机信息中，查看 IP 地址，查询节点上绑定的 EIP 状态。如下图所示：

绑定 EIP

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧导航中**节点管理**，在**节点**页面单击节点右侧的**更多**，选择**绑定弹性公网 IP**。如下图所示：

4. 在**绑定弹性公网 IP** 中，选择要绑定的 EIP，单击**确定**。

解绑 EIP

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧导航中**节点管理**，在**节点**页面单击节点右侧的**更多**，选择**解绑弹性公网 IP**。
4. 确认解绑操作信息后，单击**确定**。

Management 参数介绍

最近更新时间：2025-06-17 15:04:41

功能概览

Management 参数为节点常用自定义配置提供了统一的入口，您可通过该入口为原生节点底层的内核参数 KernelArgs 进行调优，同时支持设置 Kubelet\Nameservers\Hosts 来满足业务部署的环境要求。

Management 参数项

参数项	描述
KubeletArgs	设置业务部署时需要自定义的 Kubelet 相关参数。
Nameservers	设置业务部署环境需要 DNS 服务器地址。
Hosts	设置业务部署环境所需要的 Hosts。
KernelArgs	设置内核参数对业务进行性能调优（该功能目前白名单开放，可 提交工单 申请）。

说明：

1. 已放开自定义配置的 Kubelet 参数和集群版本相关，若当前集群在控制台可选的 Kubelet 参数无法满足您的需求，请 提交工单 支持。
2. 为确保系统组件正常安装，原生节点默认注入腾讯云官方资料库地址 `nameserver = 183.60.83.19` 、 `nameserver = 183.60.82.98` 。

通过控制台操作

方式1：为新增节点池设置 Management 参数

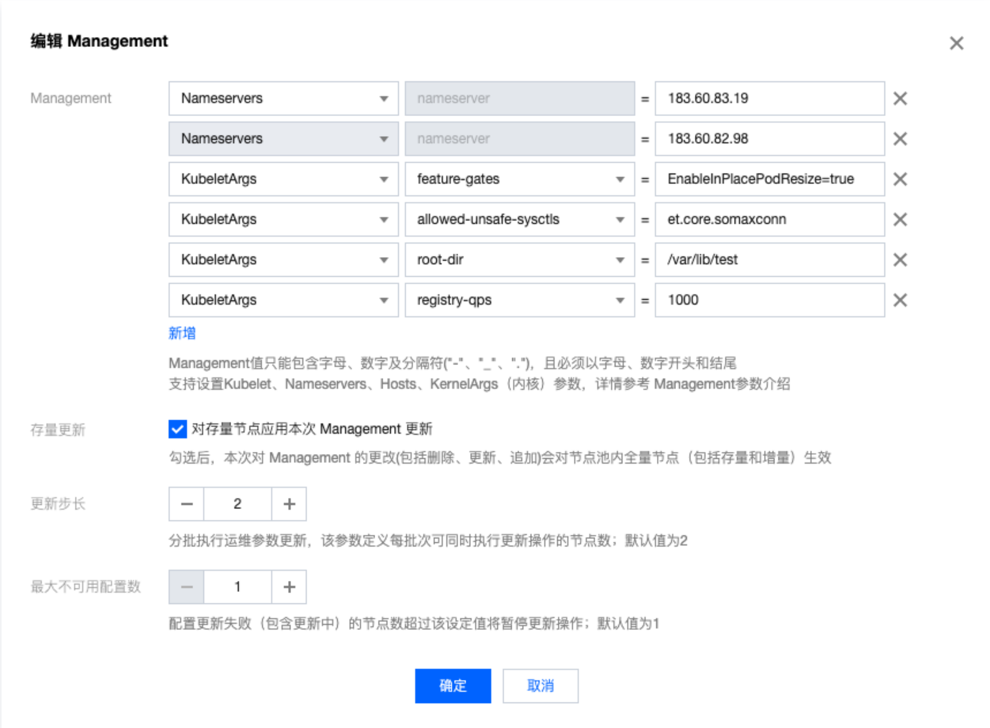
1. 登录 容器服务控制台，参考 新建原生节点 文档创建原生节点。
2. 在**新建**页面，单击**高级设置**，为节点设置 Management 参数。如下图所示：

3. 单击**创建节点池**。

方式2：为已有节点池设置 Management 参数

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理 > Worker 节点**，在**节点池**中单击节点池 ID，进入节点池详情页。
4. 在节点池详情页中，单击**参数设置 > Management > 编辑**，修改 Management 参数。如下图所示：

5. 您可通过勾选“存量更新”来设置本次参数修改是否针对节点池中的存量节点生效，勾选后本次对 Management 的更改（包括删除、更新、追加）会对节点池内全量节点（包括存量和增量）生效。如下图所示：



   - 更新步长：系统将分批执行运维参数更新操作，该参数定义每批次可同时执行更新的节点数。
   - 最大不可用配置数：更新失败（包含更新中）的节点超过该设定值时，系统将暂停更新操作。
6. 单击**确定**。您可在节点池的“运维记录”中查看更新进展和结果。

> **警告：**
> 某些 Kubelet 参数对存量节点生效时会触发业务 Pod 重启，例如 “feature-gates: EnableInPlacePodResize=true” 、 “kube-reserved: xxx” 、 “eviction-hard: xxx” 、 “max-pods: xxx” ，建议评估风险后谨慎操作。

通过 YAML 操作

```yaml
apiVersion: node.tke.cloud.tencent.com/v1beta1
kind: MachineSet
spec:
  autoRepair: false
  deletePolicy: Random
  displayName: xxxxxx
  healthCheckPolicyName: ""
  instanceTypes:
  - SA2.2XLARGE8
  replicas: 2
  scaling:
    createPolicy: ZoneEquality
```

```
  subnetIDs:
  - subnet-xxxxx
  - subnet-xxxxx
  template:
    spec:
      displayName: tke-np-bqclpywh-worker
      providerSpec:
        type: Native
        value:
          instanceChargeType: PostpaidByHour
          keyIDs:
          - skey-xxxxx
          lifecycle: {}
          management:
            kubeletArgs:
            - feature-gates=EnableInPlacePodResize=true
            - allowed-unsafe-sysctls=net.core.somaxconn
            - root-dir=/var/lib/test
            - registry-qps=1000
            hosts:
            - Hostnames:
              - static.fake.com
              IP: 192.168.2.42
            nameservers:
            - 183.60.83.19
            - 183.60.82.98
            kernelArgs:
            - kernel.pid_max=65535
            - fs.file-max=400000
            - net.ipv4.tcp_rmem="4096 12582912 16777216"
            - vm.max_map_count="65535"
          metadata:
            creationTimestamp: null
          securityGroupIDs:
          - sg-b3a93lhv
          systemDisk:
            diskSize: 50
            diskType: CloudPremium
      runtimeRootDir: /var/lib/containerd
  type: Native
```

KubeletArgs 参数说明

1. 不同账户、不同集群版本下支持配置的 kubelet 参数不完全一致，若当前参数不满足您的需求，您可以通过 在线客服 或 提交工单 与我们联系。
2. 以下参数不推荐修改，否则很大概率会影响节点上的业务正常运行：
    - container-runtime
    - container-runtime-endpoint
    - hostname-override
    - kubeconfig
    - root-dir

KernelArgs 参数说明

下面列出了支持调整的 OS 参数和接受的值。

套接字和网络优化

对于预期会处理大量并发会话的代理节点，您可以使用下面的 TCP 和网络选项调整。

编号	参数	默认值	允许的值 / 范围	参数类型	范围
1	"net.core.somaxconn"	32768	4096 – 3240000	int	The maximum length of the listening queue for each port in the system.
2	"net.ipv4.tcp_max_syn_backlog"	8096	1000 – 3240000	int	The maximum length of tcp SYN queue length.
3	"net.core.rps_sock_flow_entries"	8192	1024 – 536870912	int	The maximum size of hash table for RPS.
4	"net.core.rmem_max"	16777216	212992 – 134217728	int	The maximum size, in bytes, of the receive socket buffer.
5	"net.core.wmem_max"	16777216	212992 – 134217728	int	The maximum size, in bytes, of the send socket buffer.
6	"net.ipv4.tcp_rmem"	"4096 12582912 16777216"	1024 – 2147483647	string	The min/default/max size of tcp socket receive buffer.
7	"net.ipv4.tcp_wmem"	"4096 12582912 16777216"	1024 – 2147483647	string	The min/default/max size of tcp socket send buffer.
8	"net.ipv4.neigh.default.gc_thresh1"	2048	128 – 80000	int	The minimum number of entries that can be retained. If the number of entries is less than this value, the entries will not be recycled.
9	"net.ipv4.neigh.default.gc_thresh2"	4096	512 – 90000	int	When the number of entries exceeds this value, the GC will clear the entries longer than 5 seconds.
10	"net.ipv4.neigh.default.gc_thresh3"	8192	1024 – 100000	int	Maximum allowable number of non-permanent entries.
11	"net.ipv4.tcp_max_orphans"	32768	4096 – 2147483647	int	Maximal number of TCP sockets not attached to any user file handle, held by system. Increase this parameter properly to avoid the 'Out of socket memory' error when the load is high.
12	"net.ipv4.tcp_max_tw_buckets"	32768	4096 – 2147483647	int	Maximal number of timewait sockets held by system simultaneously. Increase this parameter properly to avoid ‘TCP: time wait bucket table overflow’ error.

文件句柄限制

在为大量流量提供服务时，所服务的流量通常来自大量本地文件。 您可以略微调整以下内核设置和内置限制，以便只占用部分系统内存来处理更大的量。

编号	参数	默认值	允许的值 / 范围	参数类型	范围
1	"fs.file-max"	3237991	8192 – 12000500	int	Limit on the total number of fd, including socket, in the entire system.
2	"fs.inotify.max_user_instances"	8192	1024 – 2147483647	int	Limit on the total number of inotify instances.

3	"fs.inotify.max_user_watches"	524288	781250 – 2097152	int	The total number of inotify watches is limited. Increase this parameter to avoid ‘Too many open files’ errors.

虚拟内存

以下设置可用于调整 Linux 内核虚拟内存 (VM) 子系统的操作以及向磁盘进行脏数据的 writeout。

编号	参数	默认值	允许的值 / 范围	参数类型	范围
1	"vm.max_map_count"	262144	65530 – 262144	int	The maximum number of memory map areas a process may have.

工作线程限制

编号	参数	默认值	允许的值 / 范围	参数类型	范围
1	"kernel.threads–max"	4194304	4096 – 4194304	int	The system–wide limit on the number of threads (tasks) that can be created on the system.
2	"kernel.pid_max"	4194304	4096 – 4194304	int	PIDs greater than this value are not allocated; thus, the value in this file also acts as a system–wide limit on the total number of processes and threads.

原生节点登录方式

最近更新时间：2025-05-26 17:00:01

操作场景

本文档介绍原生节点登录的方式，包括 SSH 密钥登录和 OrcaTerm 一键登录两种方式。

方式一：SSH 密钥登录

节点池维度

创建节点池时指定

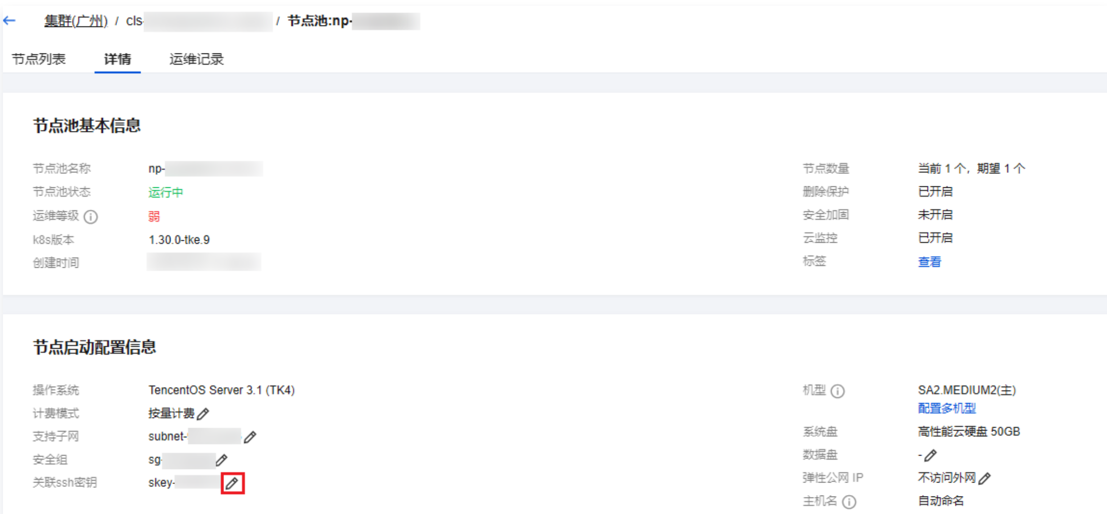
您可在创建时为节点池指定全局维度的 SSH 密钥，通过节点池扩容的节点默认会下发该密钥。

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**页面单击**新建**并选择**原生节点**。
4. 在**新建节点管理**页面，选择 **SSH 密钥**。
5. 单击**创建节点池**即可。

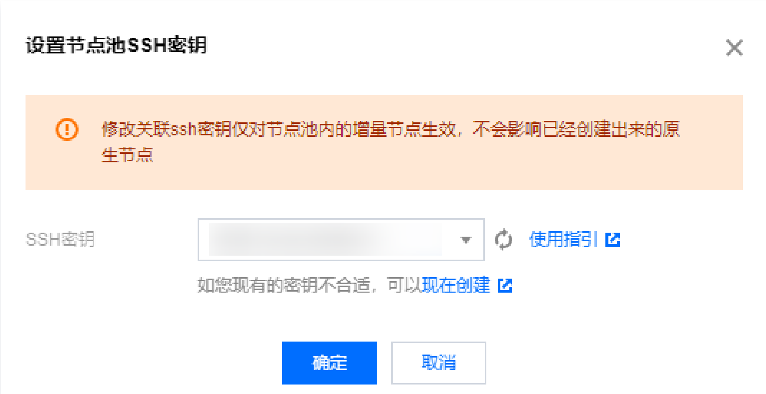
为已有节点池指定

您可在节点池详情页中找到启动配置模块，对关联 SSH 密钥进行修改。修改关联密钥仅对节点池内的增量节点生效，不会影响已经创建出来的原生节点。

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**页面单击**节点池 ID**，进入节点池详情页。
4. 在节点池详情页中，单击节点启动配置信息中**关联 ssh 密钥**右侧的 。如下图所示：



5. 在**设置节点池 SSH 密钥**页面，选择 SSH 密钥。如下图所示：



6. 单击**确定**。

节点维度

为单节点开启/修改节点登录

若您未设置节点池维度的 SSH 密钥，您可在节点操作中选择“节点登录”为目标节点开启登录，该操作同时支持修改已经下发到节点上的 SSH 密钥。

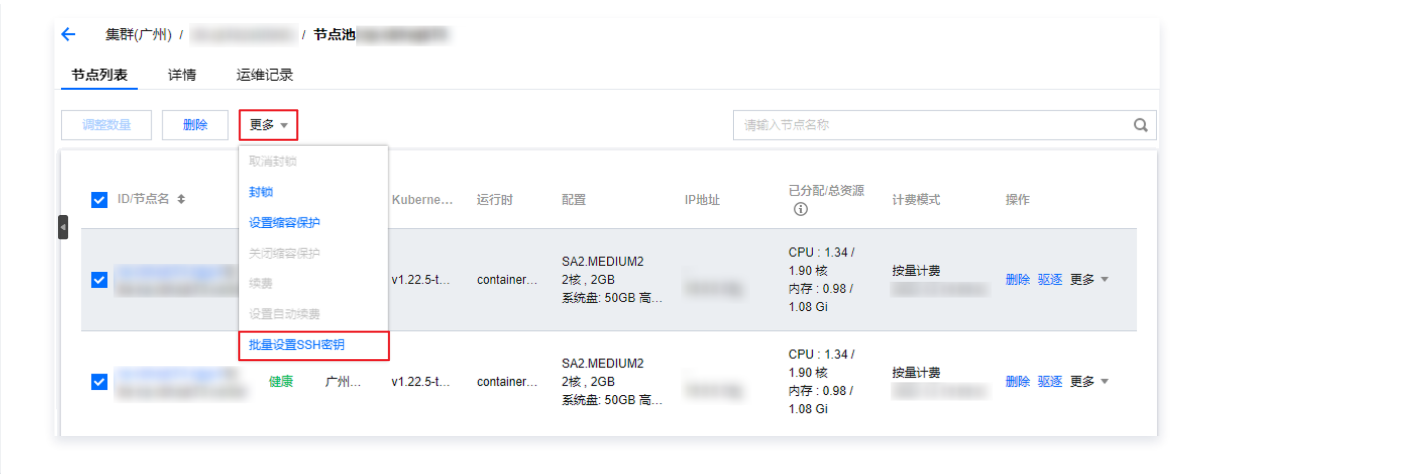
1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**页面单击**节点池 ID**，进入节点池详情页。
4. 在节点池节点列表页中，选择节点所在行右侧的**更多 > 密钥管理**。如下图所示：

5. 在密钥管理中单击**修改**。

为多节点批量开启/修改节点登录

在节点列表的更多操作中，您可在节点操作中选择“批量设置 SSH 密钥”为多个目标节点同时下发密钥。

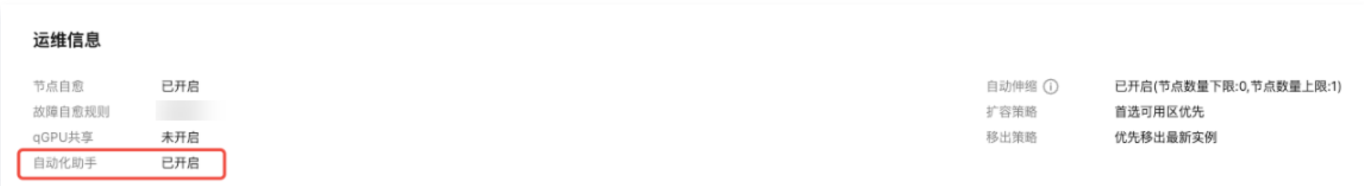
1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**页面单击**节点池 ID**，进入节点池详情页。
4. 在节点池节点列表页中，勾选多个节点，选择**更多 > 批量设置 SSH 密钥**。如下图所示：



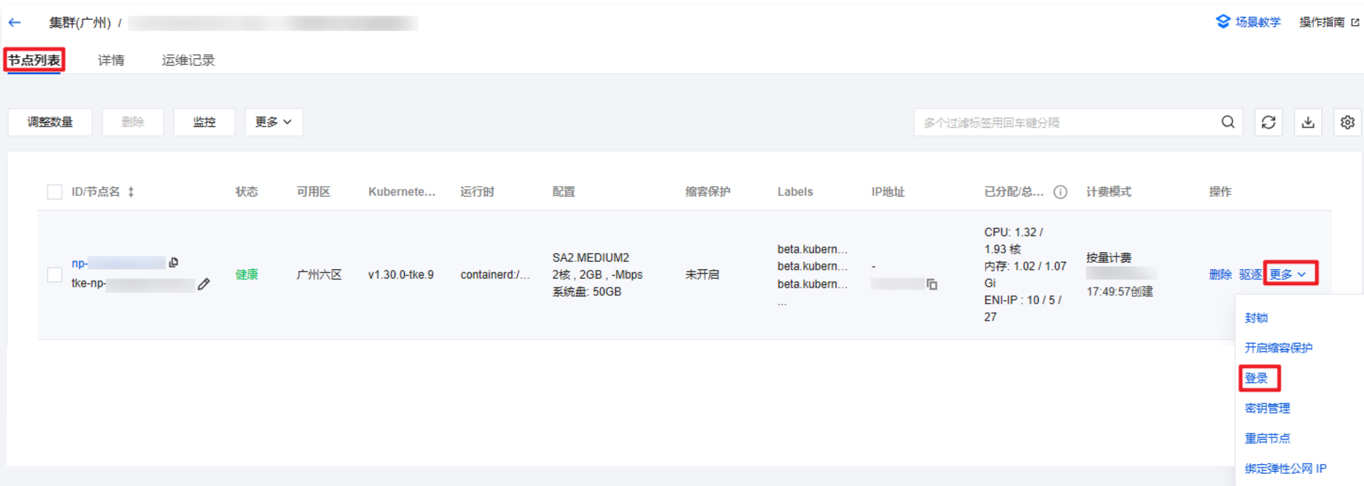
> **说明：**
> 1. 修改节点上下发的 SSH 密钥不会引起节点重启。
> 2. 首次开启/修改节点登录或修改节点下发的 SSH 密钥需要等待1-2分钟后生效。
> 3. 批量开启/修改多个节点登录时，建议单次操作控制在20个节点以内。
> 4. 批量开启/修改多个节点登录时，已经在节点上下发的密钥不会被覆盖。

方式二：OrcaTerm 一键登录

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理**，在**节点池**页面单击**节点池 ID**，进入节点池详情页。
4. 在节点池的**详情**页，**运维信息**下的**自动化助手**确保是**已开启**的状态。



5. 在节点池**节点列表**页中，选择节点所在行右侧的**更多 > 登录**。如下图所示：



6. OrcaTerm 登录详细操作请参见 OrcaTerm 多种登录方式 的免密连接、SSH 连接（密码与密钥验证）。

原生节点常见问题

最近更新时间：2024-12-02 09:56:22

原生节点是否支持标签分账?

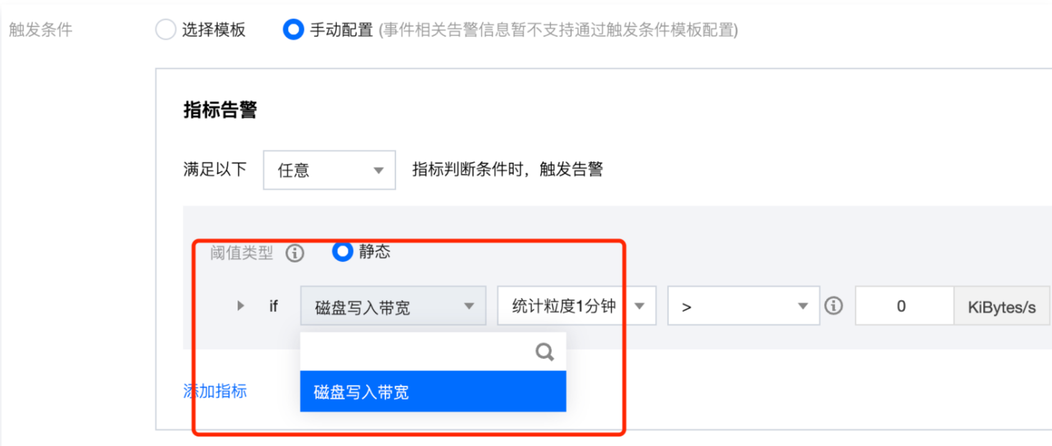
支持通过云标签对节点分账，节点池设置的云标签对池内所有节点生效。需注意：

1. 新建原生节点池时默认优先继承集群标签，请确保云标签 key 的设置不会和集群设置的标签发生冲突。
2. 集群标签发生更新时，会同步至原生节点池，进而同步至所有原生节点。

如何配置系统盘监控告警?

1. 登录 腾讯云可观测平台，选择**告警管理 > 策略管理**。
2. 单击**新建策略**，配置告警。
3. 在配置告警的页面中，监控类型选择**云产品监控**，策略类型选择**容器服务(2.0)/节点磁盘信息**或**容器服务(2.0)/节点磁盘 io**。
4. 依次选择**地域**、**集群**和**节点 ID**，设置目标原生节点为告警对象。如下图所示：

5. 选择**磁盘总量**、**磁盘使用率**、**磁盘写入带宽**作为告警指标。如下图所示：



6. 单击**下一步：配置告警通知**。告警通知配置详情请参见 通知模板 。
7. 单击**完成**。

如何在容器中查看容器本身的资源量?

背景

在普通容器中查看容器资源时（如 top 命令）会看到整机的资源，对于某些监控、排障，甚至某些业务（如根据 proc 下某些参数决定队列长度等逻辑）会产生困扰。原生节点镜像实现了控制容器资源可见性的能力，可以将 cgroupfs 挂载到主机目录下，创建容器时通过一次 bind mount 把此目录下的文件挂载到容器的 /proc 对应位置后，可以在容器中看到容器本身的资源量。即容器内通过 free、top、loadavg 等命令看到的就是容器内的值。

使用方式

需在 Pod 上设置如下 annotation：

annotation字段	含义
cloud.tencent.com/cgroupfs:"*"	代表 Pod 中所有的容器都应用 cgroupfs 的能力。
cloud.tencent.com/cgroupfs:"container1,container2"	代表 Pod 中仅容器1和容器2应用 cgroupfs 的能力。

> **说明：**
> 该特性要求 containerd 版本高于 1.4.3-tke.3 或 1.6.9-tke.3。

原生节点是否支持置放群组？

原生节点池支持置放群组功能，通过添加指定 label，可以实现节点的置放群组配置。

标签设置

置放群组级别 label：

- `node.tke.cloud.tencent.com/spread-placement-group-type`
- value 值可选：
  - `HOST`（物理机）
  - `SW`（交换机）
  - `RACK`（机架）
- 作用：创建新的置放群组供原生节点池使用。

置放群组 ID 标签：

- `node.tke.cloud.tencent.com/spread-placement-group-id`
- value 值：置放群组id
- 作用：将节点池中的新建节点加入到指定置放群组中。

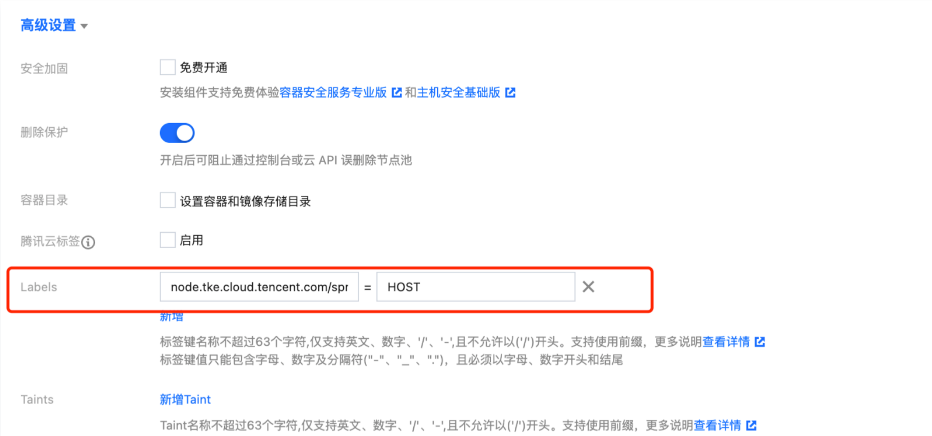
> **注意：**
> 以上两个标签只能二选一使用，设置后不能修改 value 值，也不能删除标签。

操作步骤

增量节点：

1. 在新建原生节点池页面中，展开**高级设置**。如下图所示：

2. 在 Label 选项中，填写置放群组级别 label 或置放群组 ID label。如下图所示：



存量节点：

添加指定 label 后，只会对新增节点生效，不会影响存量节点。

超级节点管理
超级节点概述

最近更新时间：2025-04-03 10:53:43

什么是超级节点?

超级节点是腾讯云全新升级的节点产品形态，向用户提供可用区级别的、支持自定义规格的节点能力，使用超级节点类似于使用一台超大规格的 CVM，资源管理和资源扩缩容都更简单，超级节点支持**包年包月**和**按量计费**两种计费模式。

- 包年包月模式下，由用户自定义节点规格，用户可包年包月购买自定义总规格的节点算力来实现预付费，包年包月模式的节点限制用户调度规模在 0.25C 至 8C 间、CPU 内存比值小于 1:4 的 Pod。包年包月模式适用于固定算力的在线常驻业务，包年包月的超级节点相较于普通节点单核价格更低，用户可将符合规则的 Pod 全部迁移至超级节点，来降低固定资源的单核成本。
- 按量计费模式下，用户无需指定节点规格，可弹性使用节点资源，使用后按节点内实际使用的 CPU 内存按量计费。按量计费模式适用于弹性算力场景，用户可添加按量计费的超级节点，在业务高峰时使用弹性资源，进一步降低集群资源成本。

部署在超级节点上的 Pod 具备与云服务器一致的安全隔离性，具备与部署在集群既有普通节点上的 Pod 一致的网络隔离性，但因超级节点上无单独的公网 IP，需要通过 NAT 或 绑定 EIP 的模式访问外网。

产品优势

超级节点相较于常规节点有如下产品优势：

成本节省

购买包年包月的超级节点，相较于购买普通节点单核价格更低。

- CPU、内存定价更低，详情请参考 超级节点价格说明 。
- 包年包月的超级节点规格上限更大，使 Pod 可调度域更广，避免了购买包年包月普通节点时边界资源的浪费。

按量计费的超级节点由于具备秒级弹性的优势及按需使用的产品形态，使其在弹性业务场景的成本节省方面也具有很大的优势。

- 按需使用，减少集群的资源 buffer。按量计费的超级节点是真正的按需使用，避免了碎片资源的产生，提升整体集群的资源利用率，通过减少资源 buffer，降低成本。
- 减少弹性资源的计费时长，节省成本。由于按量计费的超级节点是秒级扩容，瞬时缩容，因此会大幅降低在扩缩容过程中产生的计费成本。

管理简单

从在一个可用区需运维和管理多个节点变为一个可用区仅需管理 1 个节点，资源管理更容易。
包年包月超级节点支持随时升配，支持部分降配，像管理一个超规格的 CVM 节点一样，对节点进行升降配即可轻松实现常驻资源的扩缩容。弹性资源可通过添加按量计费的超级节点，弹性能力相较于常规节点池及伸缩组更快更高效，在更好满足弹性诉求的基础上，节省用户成本。
节点管理对比：

场景	常规节点	超级节点
购买	需详细规划不同节点类型、规格、数量，分多次购买。	仅需统计可用区下资源总规格并购买1个自定义规格的超级节点。
资源扩容	需要新添加节点或批量调整单个节点的规格。	• 长期扩容：升级超级节点配置，调整总规格大小。 • 短期扩容：使用按量计费超级节点，弹性部分默认调度至按量计费的超级节点。
资源缩容	需要退还节点并承担损失。	• 长期缩容：降级超级节点配置，调整总规格大小（单月支持降配1次），支持降配3个超级节点。 • 短期缩容：使用按量计费超级节点后无需处理。
节点管理	需管理多个节点。	只需按可用区维度管理单个节点。

弹性更快更高效

相比节点池及伸缩组，按量计费的超级节点的扩容、缩容流程简化了购买、初始化、退还服务器的流程，大幅提升了弹性的速度，尽可能降低在扩容流程中可能出现的失败，使得弹性更加的高效。

- 对于扩容，按量计费超级节点将 4~6 分钟的常规扩容流程缩短至秒级，扩容流程更高效。
- 对于缩容，按量计费超级节点的缩容流程短规避了 CA 流程、封锁流程及 Pod 驱逐流程，完全做到了无损缩容，瞬时缩容。

计费模式

超级节点支持按量计费、包年包月、预留券和竞价模式，详情请参见 超级节点计费模式说明 。

实例计费模式	包年包月	按量计费
付款方式	预付费	购买时 冻结费用， 每小时结算
计费单位	元/月	元/秒
单价	单价较低	单价较高
最少使用时长	至少使用一个月	按秒计费，按小时结算，随时购买随时释放
配置调整	支持升降配置	无规格限制
使用场景	适用于算力需求量长期稳定的成熟在线业务	适用于算力需求量瞬间大幅波动的场景

说明：

- 添加包年包月的超级节点，将按照节点总规格、CPU 内存单价及购买时长计算总价格进行预付费。
- 添加按量计费的超级节点，本身不收取任何费用，将根据实际调度到超级节点上的 Pod 资源来计费。后台根据工作负载申请的 CPU、GPU、内存数值以及工作负载的运行时间来核算具体费用，用户无需提前支付费用。

地域和可用区

用户可在以下可用区内使用包年包月的超级节点：

中国

地域	可用区
华南地区（广州） ap-guangzhou	广州三区 ap-guangzhou-3
	广州四区 ap-guangzhou-4
	广州六区 ap-guangzhou-6
	广州七区 ap-guangzhou-7
华东地区（上海） ap-shanghai	上海二区 ap-shanghai-2
	上海三区 ap-shanghai-3
	上海四区 ap-shanghai-4
	上海五区 ap-shanghai-5
华东地区（南京） ap-nanjing	南京一区 ap-nanjing-1
	南京二区 ap-nanjing-2
	南京三区 ap-nanjing-3
华北地区（北京）	北京三区

ap-beijing	ap-beijing-3
	北京四区 ap-beijing-4
	北京五区 ap-beijing-5
	北京六区 ap-beijing-6
	北京七区 ap-beijing-7

Kubernetes 版本

- 按量计费超级节点支持1.16及以上版本集群。
- 包年包月超级节点当前支持1.18、1.20、1.22、1.24、1.26、1.28、1.30版本集群。

超级节点可调度 Pod 说明

添加了超级节点的集群，依据不同的计费模式，支持调度不同的规格的 Pod。

包年包月的超级节点

- 支持调度 0.25C～8C 标准规格的 Pod（若为非标准规格，则自动向上转换成标准规格），规格请参考 超级节点可调度 Pod 说明 。
- 支持调度 CPU 内存比小于 1:4 的 Pod。

按量计费的超级节点

- 支持调度 0.25C~64C 标准规格的 Pod。
- 支持调度 CPU 内存比小于等于 1:8 的 Pod。
- 支持调度 GPU Pod。

详情请参考 超级节点可调度 Pod 说明 。

超级节点调度说明

包年包月的超级节点与包年包月的 TKE 常规节点平权调度。
添加了按量计费的超级节点的 TKE 集群，在包年包月的节点资源不足时，会自动调度到按量计费的超级节点上，若节点资源充足，会优先缩容按量计费超级节点上的 Pod。另外，也支持手动将 Pod 调度至超级节点。
详情请参考 调度至超级节点 。

应用场景

超级节点适用于全业务场景，包年包月模式适用于 Pod 规格在 1C 到 8C 的所有在线常驻业务，按量计费模式适用于弹性场景。

在线常驻业务使用超级节点

优势：低成本、易管理
相较于包年包月的普通节点，超级节点单核价格便宜约 20%。
对于在线常驻的服务，用户所需算力固定，若当前 Pod 规格大部分在 8C 以下，可使用超级节点的包年包月模式，基于 Pod 总规格按需购买包年包月的超级节点，将 1C~8C Pod 全部调度至超级节点，降低集群资源的单核价格，实现降本诉求。

弹性业务使用按量计费的超级节点

优势：低成本、高弹性
弹性业务场景下，用户使用按量计费的超级节点，可实现快速秒级扩容，轻松应对突发流量，通过减少预留的资源 buffer 降低成本。

- 高弹性：快速秒级扩容，轻松应对突发流量，业务流量下降后自动销毁 Pod，无损缩容。
- 低成本：减少集群预留 buffer，将集群的节点维护在资源利用率更高、使用和预留更合理的水平，节省成本。

购买超级节点
资源规格

最近更新时间：2025-06-05 18:30:12

概述

使用 TKE Serverless，您无需关心集群节点，但为了合理分配资源和准确核算，您需在部署工作负载时指定 Pod 申请的资源规格。腾讯云会根据您指定的规格为工作负载分配计算资源，也会根据此规格进行费用核算。
当您使用 Kubernetes API 或 Kubectl 创建 TKE Serverless 工作负载时，可以通过 Annotation 指定资源规格。如果不指定，TKE Serverless 会根据工作负载设置的容器 Request、Limit 等参数进行计算。详情请参考 指定资源规格 。

注意：
- 资源规格是 Pod 内容器可使用的最大资源量（含操作系统、容器运行时、日志采集等系统组件资源占用）。
- 目前支持分配的 CPU、GPU 规格见下表，分配时请勿超出所支持的规格。
- Pod 内所有 Container 设置的 Request 之和不可大于最大 Pod 规格。
- Pod 内任何 Container 设置的 Limit 不可大于最大 Pod 规格。

CPU 规格

TKE Serverless 在所有支持 CPU 资源类型的地域提供以下 CPU Pod 规格。TKE Serverless 提供一系列 CPU 选项，不同的 CPU 大小会对应不同的内存选择区间，请在创建工作负载时根据您的实际需求选择最合适规格，并进行资源分配。

Intel

CPU/核	内存区间/GiB	内存区间粒度/GiB
0.25	0.5、1、2	-
0.5	0.5、1、2、3、4	-
1	1 - 8	1
2	2、4 - 16	1
4	8 - 32	1
8	16 - 64	1
12	24 - 96	1
16	16 - 128	1
32	64、128、256	-
64	128、192、256、512	-

星星海 AMD

基于腾讯云自研星星海服务器，提供可靠、安全、稳定的高性能。详情请参见 云服务器标准型 SA2 介绍 。

CPU/核	内存区间/GiB	内存区间粒度/GiB
1	1 - 4	1
2	2 - 8	1
4	4 - 16,32	1
8	8 - 32,64	1
16	16 - 64,128	1

32	64、128、256	-
64	128、192、256	-

GPU 规格

TKE Serverless 提供以下型号 GPU Pod 规格，不同的 GPU 卡型号和大小会对应不同的 CPU、内存选项，请在创建工作负载时根据您的实际需求选择最合适规格，并进行资源分配。

> **注意：**
> 如果您需要通过 yaml 来创建、管理和使用 GPU 的工作负载，请参考如下 Annotation 信息和 Annotation 说明 。
> 1/4 NVIDIA T4和 1/2 NVIDIA T4的规格有更新，详情请参见 关于 Serverless 容器服务 T4 vGPU Pod 规格变化的公告 。

GPU 型号	GPU/卡	GPU 显存	支持驱动	CUDA 版本	CPU/核	内存/GiB	Annotation 指定 gpu-type
NVIDIA Tesla V100 - 1颗	1	1 * 32GB	470	11.4	8	40	eks.tke.cloud.tencent.com/gpu-type: V100
NVIDIA Tesla V100 - 2颗	2	2 * 32GB	470	11.4	18	80	eks.tke.cloud.tencent.com/gpu-type: V100
NVIDIA Tesla V100 - 4颗	4	4 * 32GB	470	11.4	36	160	eks.tke.cloud.tencent.com/gpu-type: V100
NVIDIA Tesla V100 - 8颗	8	8 * 32GB	470	11.4	72	320	eks.tke.cloud.tencent.com/gpu-type: V100
1/4 NVIDIA T4 - 1/4颗	1	4GB vGPU	450	11.0	4	16	eks.tke.cloud.tencent.com/gpu-type: 1/4*T4
1/2 NVIDIA T4 - 1/2颗	1	8GB vGPU	450	11.0	8	32	eks.tke.cloud.tencent.com/gpu-type: 1/2*T4
NVIDIA T4 - 1颗	1	1 * 16GB	470	11.4	8	32	eks.tke.cloud.tencent.com/gpu-type: T4
NVIDIA T4 - 1颗	1	1 * 16GB	470	11.4	20	80	eks.tke.cloud.tencent.com/gpu-type: T4
NVIDIA T4 - 1颗	1	1 * 16GB	470	11.4	32	128	eks.tke.cloud.tencent.com/gpu-type: T4
NVIDIA T4 - 2颗	2	2 * 16GB	470	11.4	40	160	eks.tke.cloud.tencent.com/gpu-type: T4
NVIDIA T4 - 4颗	4	4 * 16GB	470	11.4	80	320	eks.tke.cloud.tencent.com/gpu-type: T4
NVIDIA A10 - PNV4 - 1颗	1	1 * 24GB	470	11.4	28	116	eks.tke.cloud.tencent.com/gpu-type: A10*PNV4
NVIDIA A10 - PNV4 - 2颗	2	2 * 24GB	470	11.4	56	232	eks.tke.cloud.tencent.com/gpu-type: A10*PNV4
NVIDIA A10 - PNV4 - 4颗	4	4 * 24GB	470	11.4	112	466	eks.tke.cloud.tencent.com/gpu-type: A10*PNV4
NVIDIA A10 - PNV4 - 8颗	8	8 * 24GB	470	11.4	224	932	eks.tke.cloud.tencent.com/gpu-type: A10*PNV4
NVIDIA A10 - GNV4 - 1颗	1	24GB	470	11.4	12	44	eks.tke.cloud.tencent.com/gpu-type: A10*GNV4

NVIDIA A10 – GNV4v – 1/4颗	1	6GB vGPU	470	11.4	6	24	eks.tke.cloud.tencent.com/gpu-type: 1/4*A10*GNV4v
NVIDIA A10 – GNV4v – 1/2颗	1	12GB vGPU	470	11.4	14	58	eks.tke.cloud.tencent.com/gpu-type: 1/2*A10*GNV4v
NVIDIA A10 – GNV4v – 1颗	1	1 * 24GB	470	11.4	28	116	eks.tke.cloud.tencent.com/gpu-type: A10*GNV4v

超级节点价格说明
计费模式说明

最近更新时间：2025-06-09 17:21:12

超级节点的新版产品定价于2023年7月1日00:00开始生效。

计费模式说明

TKE Serverless 以超级节点维度承载资源，提供以下几种计费模式：按量计费、包年包月、预留券和竞价模式。下表列出了这些计费模式的区别：

计费模式	超级节点包年包月模式	超级节点按量计费模式	超级节点预留券模式	竞价模式
付款方式	预付费	购买时 冻结费用，每小时结算	预付费	购买时冻结费用，每小时结算
计费单位	元/月	元/秒	元/月	元/秒
单价	单价较低	单价较高	单价较低	单价最低，固定折扣比例，同规格按量计费模式的20%（2折）
最少使用时长	至少使用一个月	按秒计费，按小时结算，随时购买随时释放	至少使用一个月	按秒计费，按小时结算，随时购买随时释放
最小购买规格	10C20G	无限制	与 Pod 规格一致	无限制
使用场景	适用于算力需求量长期稳定的成熟在线小规格（8C内）业务	适用于算力需求量瞬间大幅波动的场景	适用于算力需求量长期稳定、需要保障资源的业务，推荐单个 Pod 超过8C时使用预留券模式	适用于短时长突发、周期任务，可切分的计算任务及无状态或者支持断点续传能力的计算任务。
计费模式维度	超级节点	Pod 维度	一张预留券对应抵扣一个Pod	Pod 维度

按量计费定价

在按量计费场景下，服务会根据用户选择的容器资源类型计算相关费用。计费公式为：**费用 = 相关计费项配置 × 资源单位时间价格 × 运行时间（精确到秒）**。

目前，超级节点支持的计费项配置详情请参见 资源规格。如果您想了解如何选择计费项配置，请参见 指定资源规格。

Intel Pod 按量计费

地域	计费项			
	CPU（元/核/小时）	内存（元/GiB/小时）	高性能云硬盘（元/GB/小时） 超过20GB部分收费	SSD 云硬盘（元/GB/小时） 超过20GB部分收费
上海、北京、广州、南京、重庆、成都、武汉、长沙、郑州、西安、沈阳、福州、合肥、济南、杭州、石家庄、上海金融、深圳金融、北京金融、上海自动驾驶云、日本、泰国、美国硅谷、弗吉尼亚、圣保罗	0.12	0.05	0.0009	0.0025
中国香港、中国台北、新加坡	0.12	0.06	0.0005	0.0024
首尔、法兰克福	0.126	0.063	0.0005	0.0024
雅加达	0.14	0.07	0.0005	0.0024

AMD Pod 按量计费

地域	计费项			
	CPU（元/核/小时）	内存（元/GiB/小时）	高性能云硬盘（元/GB/小时） 超过20GB部分收费	SSD 云硬盘（元/GB/小时） 超过20GB部分收费
上海、北京	0.065	0.032	0.0009	0.0025
重庆、成都、南京	0.055	0.032	0.0005	0.0024
广州、武汉、长沙、郑州、西安、沈阳、福州、合肥、济南、杭州、石家庄	0.061	0.032	0.0005	0.0024
上海金融、深圳金融、北京金融、上海自动驾驶云	0.099	0.049	0.0005	0.0024
中国香港、中国台北、日本、新加坡、泰国、弗吉尼亚、圣保罗、法兰克福	0.087	0.044	0.0005	0.0024
美国硅谷	0.072	0.036	0.0005	0.0024

GPU V100 Pod 按量计费

计量	价格（秒）	价格（小时）	备注
GPU	每卡 0.003193 元	每卡 11.5 元	同样适用于 vGPU，vGPU 以虚拟化系数（0.25、0.5）乘以此价格
CPU	每核 0.00005778 元	每核 0.208 元	–
内存	每 GiB 0.00003389 元	每 GiB 0.122 元	–

GPU T4 Pod 按量计费

计量	价格（秒）	价格（小时）	备注
GPU	每卡 0.001447 元	每卡 5.21 元	同样适用于 vGPU，vGPU 以虚拟化系数（0.25、0.5）乘以此价格
CPU	每核 0.00002411 元	每核 0.0868 元	–
内存	每 GiB 0.00002411 元	每 GiB 0.0868 元	–

GPU A10 Pod 按量计费

GNV4v

计量	价格（秒）	价格（小时）	备注
GPU	每卡 0.00170718 元	每卡 6.15 元	同样适用于 vGPU，vGPU 以虚拟化系数（0.25、0.5）乘以此价格
CPU	每核 0.0000185 元	每核 0.0666 元	–
内存	每 GiB 0.00000926 元	每 GiB 0.0333 元	–
license	每个 0.00012733 元	每个 0.458 元	–

GNV4

计量	价格（秒）	价格（小时）	备注
GPU	每卡 0.00170718 元	每卡 6.15 元	同样适用于 vGPU，vGPU 以虚拟化系数（0.25、0.5）乘以此价格

CPU	每核 0.00002894 元	每核 0.1042 元	-
内存	每 GiB 0.00001446 元	每 GiB 0.0521 元	-
license	每个 0.00012733 元	每个 0.458 元	-

PNV4

计量	价格（秒）	价格（小时）	备注
GPU	每卡 0.00170718 元	每卡 6.15 元	同样适用于 vGPU，vGPU 以虚拟化系数（0.25、0.5）乘以此价格
CPU	每核 0.0000185 元	每核 0.0666 元	-
内存	每 GiB 0.00000926 元	每 GiB 0.0333 元	-

按量计费 Pod 运行时间说明

按量计费 Pod 运行时间是指从 Pod 资源分配完成，准备拉取首个容器的镜像开始计算，到 Pod 运行终止结束的时间段。这段时间以秒为单位计算，作为 Pod 的计费时间。

包年包月定价

地域	可用区	计费项	
		CPU（元/核/月）	内存（元/GiB/月）
上海	上海四区、上海五区、上海七区	38	20
北京	北京七区	38	20
上海、北京、广州、南京	-	36	18
中国香港、中国台北	-	48	24
新加坡、法兰克福、弗吉尼亚、美国硅谷	-	46	23

说明：
- 未指定可用区通常指除了指定的可用区之外的其他可用区。
- 包年包月模式暂不支持 GPU 类型。
- 包年包月模式下支持购买的最小规格为10C20G。
- 包年包月模式仅抵扣8C以内规格的 Pod。
- 包年包月模式支持调度 intel 和 amd 类型的 Pod，如无明确指定将优先调度 intel 机型。
- 包年包月退费说明请参见 包年包月超级节点退费说明 。

计费示例

包年包月

示例 1

包年包月购买广州地域总规格为 200C400G 的超级节点 1 个月，总费用为 36 × 200 + 18 × 400 = 14400 元。该节点可累计调度算力总和为 200C400G 的符合包年包月超级节点规则的 Pod 资源。

按量计费

示例 1

某 Deployment 的 Pod 资源规格为 2 核 4GB，指定 intel，调度到广州六区，Pod 数固定为 2。假设该 Deployment 从启动到终止，共耗时 5 分钟，即要计算 300 秒的费用。则该 Deployment 的运行费用为 2 ×（2 × 0.00003334 + 4 × 0.00001389） × 300 = 0.073344 元，匹配优惠折扣后，最终计

费为 0.073344 × 0.675 = 0.0495072 元。

示例 2

某 CronJob 需要每次启动 10 个 Pod，每个 Pod 资源规格为 4 核 8GB，指定 intel，调度到上海五区，运行 10 分钟后结束。假设该 CronJob 每天执行 2 次，使用 TKE Serverless 服务托管该任务。则该任务每天收费为 2 × 10 ×（4 × 0.00003334 + 8 × 0.00001389） × 600 = 2.93376 元，匹配优惠折扣后，最终计费为 2.93376 元。

预留券定价

TKE Serverless 服务预留券是一种预付费产品。支持按月购买，购买后可以对同地域、同资源规格类型、同资源规格的容器（Serverless 集群 Pod 或容器实例）进行计算费用抵扣。详情请参见 预留券概述，定价请参见 预留券定价。

竞价模式

TKE Serverless 竞价模式是一种低成本资源购置模式，其核心特点为价格低于按量付费实例，在一些场景可以极大降低运行容器的成本，但资源可能会被腾讯云中断回收。您在使用竞价模式运行时仅需支付较少的费用，并运行到容器资源被回收为止。详情请参见 竞价模式说明。

包年包月超级节点退费说明

最近更新时间：2023-05-17 15:40:25

为了更加方便您使用超级节点，如果您在购买包年包月超级节点后有任何不满意，我们支持自助退货退款。每个主体可享受1次1个超级节点五天无理由退还，您支付的有效金额将返还至您的腾讯云账户。除此之外，您还可享受普通自助退还，扣除您已使用的费用，将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。您可在容器服务控制台自助进行。

五天无理由自助退还

TKE 超级节点遵守 云服务退货说明，若您在购买超级节点后有任何不满意，我们支持五天内无理由自助退还，具体策略如下：
- 每个主体下，包年包月预付费超级节点自**新购之日起五天之内（含五天）**，默认享受1个超级节点五天无理由退还。
- 符合五天无理由退还场景的订单，退款金额为购买时花费的全部消耗金额，包括现金账户金额、收益转入账户金额以及赠送账户金额。
- 如需退还参与活动购买的超级节点，且退款规则与活动规则有冲突，则以活动规则为准。
- 具体退款规则请参见 五天内无理由全额退款 。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。

普通自助退还

不满足五天无理由退货退款规则的退货订单，退款策略如下：
普通自助退还将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

普通自助退还限制

每个账户最多支持199个超级节点控制台自助退还。

普通自助退还规则

退款金额 = 当前有效订单金额 + 未开始订单金额 − 资源已使用价值
- 当前有效订单金额：指生效中订单的付款金额，不包含折扣和代金券。
- 未开始订单金额：将来生效订单的付款金额，不包含代金券。
- 资源已使用价值按照如下策略计算：已使用部分 = 已使用超级节点费用（用户发起退费的当天，超级节点使用已满整月的，按已使用整月的包年包月价格及对应的官网折扣进行扣除；不满整月的，按已使用时长内调度 Pod 的按量计费价格进行扣除。）
- 退款金额 <= 0，按0计算并清退资源。
- 抵扣的代金券不予以退还。
- 退还金额将按购买支付使用的现金和赠送金支付比例退还至用户的腾讯云账户。

超级节点自助退还后，超级节点的状态从生效中变为销毁中或已销毁，此时，不再产生该超级节点相关的费用，包月超级节点上也不可再调度 Pod。
包月超级节点自助退还后，对用户实际运行的 Pod 不产生影响，而是转换为按量计费模式。

升配费用规则

超级节点升配遵循按天补差价，升配费用 = 按月升配差价 × 升配月数 × 适用折扣。
- 按月升配差价：新老配置原价按月的单价差。
- 升配月数 = 升配天数 /(365/12)
- 升配的费用按天计算：升配天数 = 资源到期时间 − 当前时间
- 适用折扣：根据升配月数匹配官网适用折扣，其中折扣为官网生效的折扣。
- 升配不影响资源到期时间。
- 升配可以使用代金券抵扣费用。

降配费用规则

超级节点降配遵循按天退还差价，降配费用 = 剩余整月部分退还费用 + 剩余不足整月部分退还费用
- 剩余整月部分退还费用 = 按月降配差价 × 降配月数 × 适用折扣
- 剩余不足整月部分退还费用 = 按月降配差价 − 降配资源数 × 已使用时间 × 按量计费刊例价
- 升配不影响资源到期时间。
- 退款金额 <= 0，按0计算并清退资源。

购买预留券
预留券概述

最近更新时间：2023-05-31 17:56:18

TKE Serverless 服务预留券是一种预付费产品。支持按月购买，购买后可以对同地域、同资源规格类型、同资源规格的容器（Serverless 集群 Pod 或容器实例）进行计算费用抵扣。

预留券特点

- **单位时间价格便宜**
  预留券仅支持按月购买，在生效期内，每小时/秒的单位时间价格相比 TKE Serverless 服务按量计费模式更便宜，具体可参考 产品定价 。
- **生命周期独立**
  预留券和容器的生命周期互相独立，您可以先创建容器，再根据容器的资源规格购买匹配的预留券。也可以提前购买预留券，再创建与其资源规格一致的容器。
- **即时抵扣**
  成功购买预留券后，可在本账单周期（通常是一小时）内抵扣与之匹配的容器的费用。抵扣原则请参考 预留券使用说明 。
- **预保留资源**
  当指定可用区/超级节点时，可选择保留指定配置的资源，保证在此可用区/节点下创建出资源。
- **抵扣计算费用**
  抵扣计算费用，例如 CPU、内存费用，不包括网络、存储等其他资源或服务费用。

相关概念

资源规格类型

由于弹性容器本身不同算力资源（Intel CPU、AMD CPU、GPU）的价格不同，预留券也分为 CPU 通用、AMD 专用以及其他资源专用等类型。其中：

预留券类型	使用场景
CPU 通用类型	可以抵扣包括 Intel、AMD 型号 CPU Pod 产生的费用，并且会优先抵扣 Intel 型号。该类型适合您在混合使用 Intel、AMD 资源的时候采购。
AMD 专用类型	仅可以抵扣 AMD 型号 CPU Pod 产生的费用，但单位时长的费用更低。该类型适合您仅使用 AMD 资源的时候采购。

资源规格

资源规格是指预留券的配置，例如8核 CPU、16GiB内存等。为了保证预留券能适配抵扣容器费用，TKE Serverless 服务仅支持购买弹性容器支持规格的预留券，支持规格详情请参见 资源规格 。

抵扣范围

- 地域级：预留券只指定地域，不预留资源，可抵扣该地域下的全量 Pod。
- 可用区级：预留券勾选保留资源，并指定可用区，只可抵扣该可用区下的 Pod。
- 节点级：预留券勾选保留资源，并选择绑定到某个超级节点上，只可抵扣该节点上的 Pod。

计费公式

预留券支持预付费按月购买，将根据用户选择的预留券类型、资源规格以及购买时长，计算相关费用。计费公式如下：
费用 = 预留券配置 × 资源单位时间价格 × 购买时长
其中，

- 预留券支持配置详情请参考 资源规格 。
- 资源单位时间具体价格请参考 产品定价 。
- 购买时长单位为月，至少购买一个月。

适用场景

- 需要长期运行，负载稳定的业务。
- 需要长期运行，但需要弹性的业务，建议将最低保留容器负载转为预留券模式，动态扩缩的容器负载依然使用传统按量计费模式。

- 需要长期持有资源，保证资源供给的情况，建议选择可用区并预留资源。

预留券定价

最近更新时间：2025-03-21 10:13:22

Common 通用预留券

CPU 通用类型预留券可以优先抵扣 Intel CPU 容器的费用，详情请参见 预留券使用说明 。

> **说明：**
> 未指定可用区通常指除了指定的可用区之外的其他可用区。

地域	可用区	计费项	
		CPU（元/核/月）	内存（元/GiB/月）
上海、北京、广州、成都、重庆、南京	–	49	22
武汉、长沙、郑州、西安、沈阳、福州、合肥、济南、杭州、石家庄	–	49	24
上海金融、深圳金融、北京金融、上海自动驾驶云	–	72	36
中国香港	–	58	30
中国台北、首尔、日本	–	56	33
新加坡	–	56	33
泰国、美国硅谷、弗吉尼亚、圣保罗	–	50	25

AMD 预留券

AMD 专用类型预留券可以抵扣 AMD CPU 容器的费用，详情请参见 预留券使用说明 。

> **说明：**
> 未指定可用区通常指除了指定的可用区之外的其他可用区。

地域	可用区	计费项	
		CPU（元/核/月）	内存（元/GiB/月）
上海、北京	–	36	17
广州、成都、重庆、武汉、长沙、郑州、西安、沈阳、福州、合肥、济南、杭州、石家庄	–	35	17
南京	–	35	16
上海金融、深圳金融、北京金融	–	52	27
上海自动驾驶云	–	54	24
中国香港	–	45	23
首尔	–	47	23
新加坡、法兰克福	–	43	22
美国硅谷	–	36	18

GPU V100 预留券

GPU V100 预留券可以用于抵扣 V100 型号的 GPU 容器的费用，详情请参见 预留券使用说明 。

计量	价格（月）	价格（小时）
GPU	每卡 3310 元	约每卡 4.597 元
CPU	每核 60 元	约每核 0.083 元

内存	每 GiB 35 元	约每 GiB 0.069 元

GPU T4 预留券

GPU T4 预留券可以用于抵扣 T4 型号的 GPU 容器的费用，详情请参见 预留券使用说明 。

计量	价格（月）	价格（小时）
GPU	每卡 1500 元	约每卡 2.083 元
CPU	每核 25 元	约每核 0.035 元
内存	每 GiB 25 元	约每 GiB 0.035 元

GPU A10 预留券

GNV4v 预留券

计量	价格（月）	价格（小时）
GPU	每卡 2950 元	约每卡 4.097 元
CPU	每核 32 元	约每核 0.0444 元
内存	每 GiB 16 元	约每 GiB 0.0222 元
License	每个 220 元	约每个 0.3056 元

GNV4 预留券

计量	价格（月）	价格（小时）
GPU	每卡 2950 元	约每卡 4.097 元
CPU	每核 50 元	约每核 0.0694 元
内存	每 GiB 25 元	约每 GiB 0.0347 元
License	每个 220 元	约每个 0.3056 元

PNV4 预留券

计量	价格（月）	价格（小时）
GPU	每卡 2950 元	约每卡 4.097 元
CPU	每核 32 元	约每核 0.0444 元
内存	每 GiB 16 元	约每 GiB 0.0222 元

预留券使用说明

最近更新时间：2024-05-23 15:10:51

使用说明

预留券在生效时间内，尽量匹配并抵扣每个容器（Serverless 集群 Pod 或容器实例）结算周期产生的容器计算费用。

生效时间

预留券购买即时生效，并会抵扣当前结算周期账单。购买月数个自然月之后日期的23点59分59秒到期，到期后停止抵扣行为。
例如，在4月15日16点购买一个月预留券，则在5月15日23点59分59秒到期。

容器结算周期及预留券抵扣状态

容器结算周期为1小时，预留券生效时会默认覆盖本结算周期。
控制台抵扣状态显示为两个计费周期前的抵扣状态，并在整点更新。
示例
用户在13点30分购买预留券，可以在14点的抵扣周期抵扣13点到14点的按量计费 Pod，此时控制台预留券抵扣状态属于11点到12点这一计费周期；14点时控制台预留券抵扣状态更新为12点到13点的计费周期抵扣状态。

匹配抵扣原则

- **匹配抵扣范围**
  - 地域级预留券，可抵扣该地域下的全量 Pod。
  - 可用区级，只可抵扣该可用区下的 Pod。
  - 节点级预留券，只可抵扣该节点上的 Pod。
- **同地域**
  预留券与容器必须在同一地域才能进行费用抵扣。
  例如，广州地域的预留券不可抵扣上海地域的容器费用。
- **同资源规格类型**
  预留券与容器必须资源类型相同才能进行费用抵扣。
  例如，AMD 专用预留券不可抵扣 Intel CPU 容器费用。
- **同资源规格**
  预留券与容器必须配置相同才能进行费用抵扣。
  例如，8核16GiB 的预留券不可抵扣4核8GiB 的容器费用。
- **同时长**
  预留券最多在一个结算周期内抵扣本周期内（相同算力 × 结算周期时长）的容器费用。

计算费用

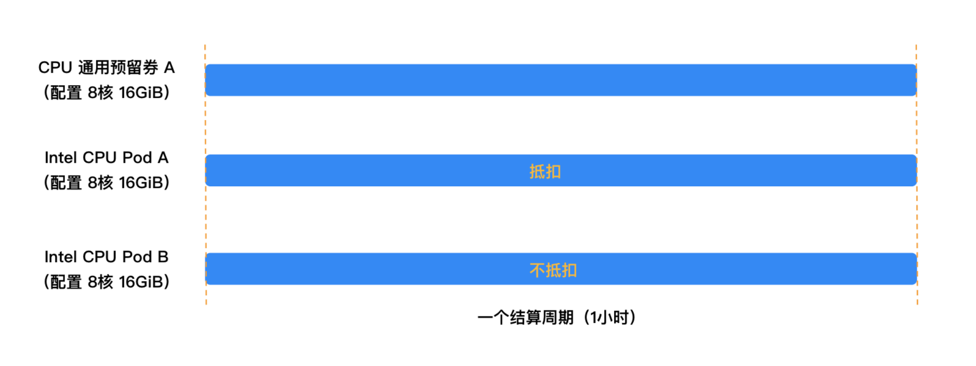
计算费用包括 CPU、内存、GPU 费用，不包括网络、存储等其他资源和服务费用。

匹配抵扣示例

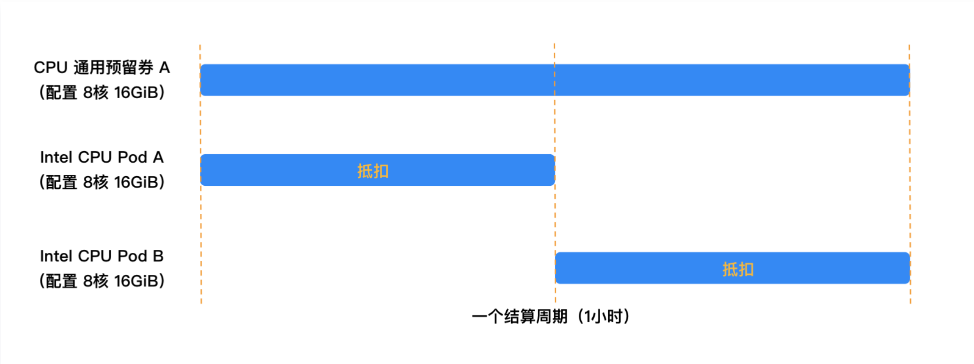
同时长匹配

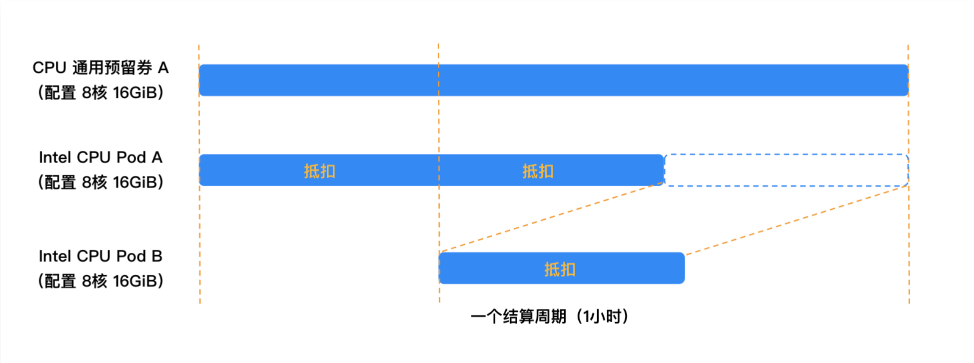
同样资源规格配置的预留券 A、Pod A、Pod B，根据结算周期内的两个容器的运行时长不同，存在以下三种抵扣情况：

- Pod A、Pod B 运行时长均覆盖整个结算周期，预留券 A 仅可抵扣其中一个容器的费用。如下图所示：

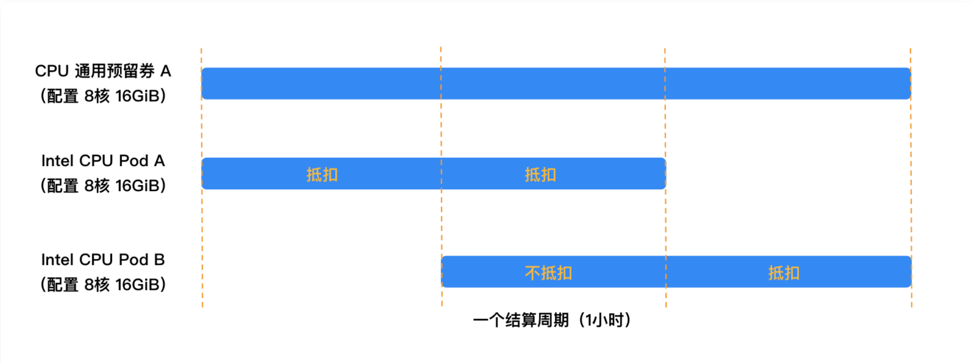


- Pod A、Pod B 运行时长总和正好覆盖整个结算周期，预留券 A 可抵扣两个容器的费用。如下图所示：



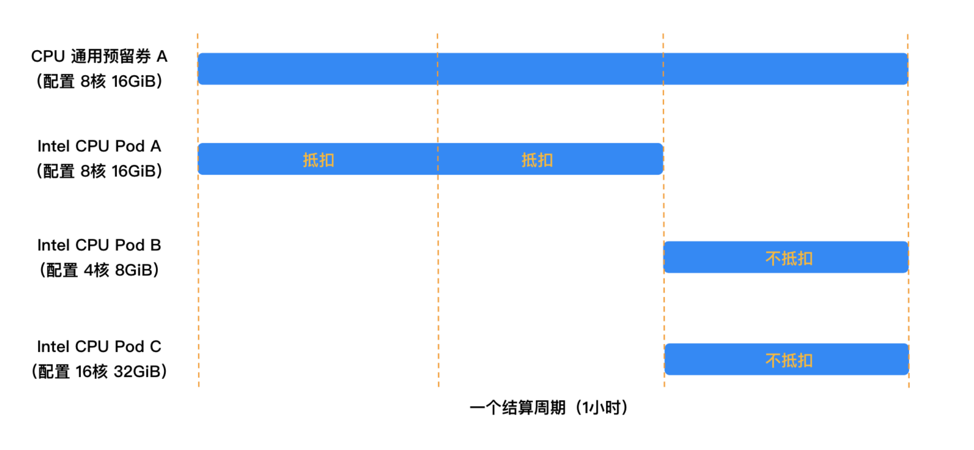


- Pod A、Pod B 运行时长总和超过整个结算周期，预留券 A 最多可抵扣该资源规格容器一个周期时长的费用。如下图所示：

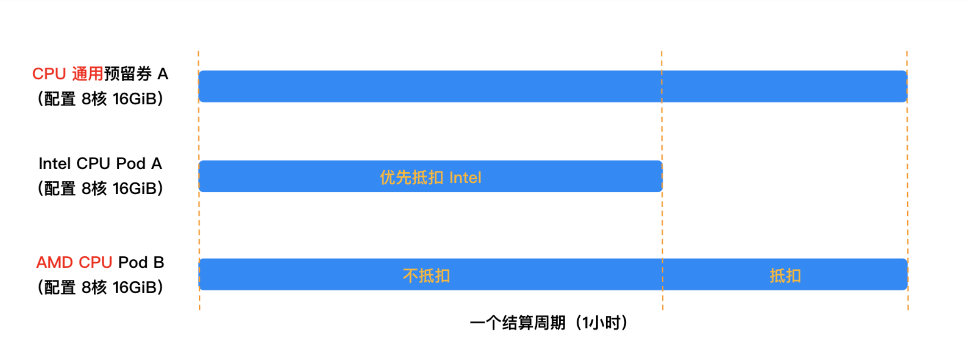


同资源规格匹配

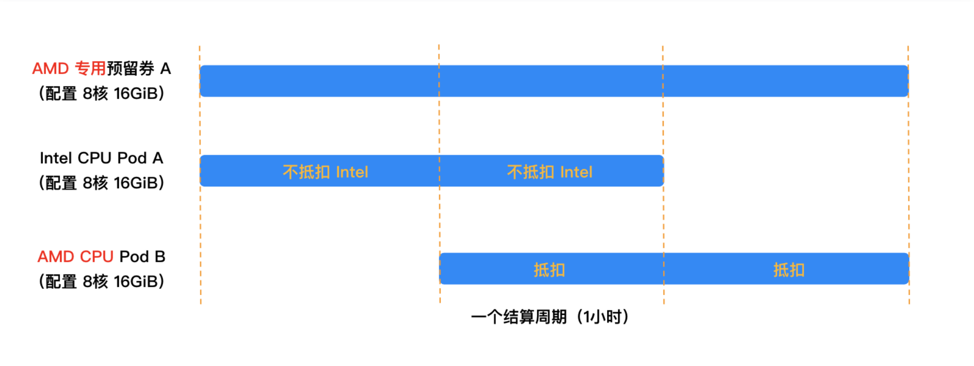
预留券 A 只能抵扣同样资源规格配置的 Pod A。如下图所示：



CPU 通用类型，优先抵扣 Intel



AMD 专用类型，仅抵扣 AMD



预留券管理

最近更新时间：2023-11-10 17:39:22

操作场景

本文介绍如何创建、管理 TKE Serverless 服务预留券，包括如何购买、退还预留券和查看预留券抵扣信息。

前提条件

在创建 TKE Serverless 服务预留券前，您需要完成以下工作：

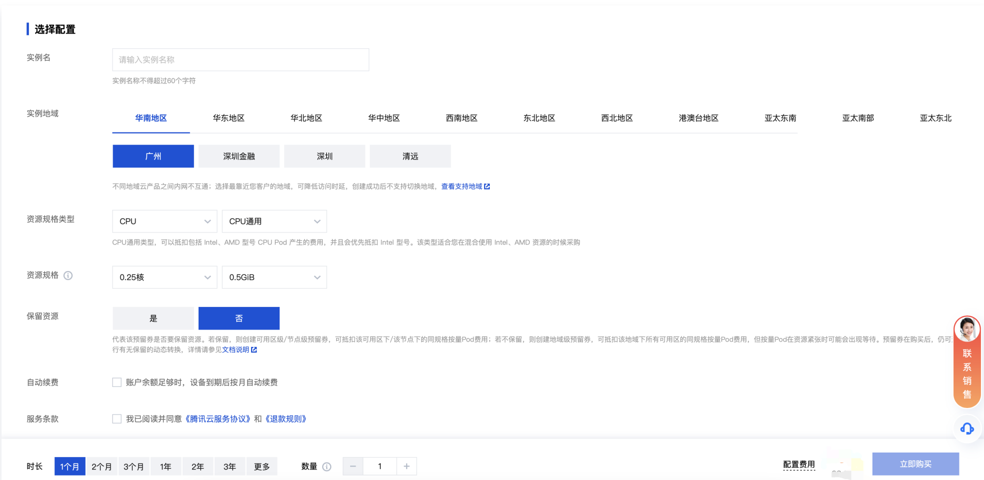
- 注册腾讯云账号，并完成 实名认证。
- 预留券是预付费产品，请确保账号余额充足。

操作步骤

创建预留券

1. 登录 容器服务控制台，选择左侧导航栏的**资源包管理 > 资源预留券**。
2. 在**预留券**列表页面，单击**新建**，进入 弹性容器服务预留券购买页 。如下图所示：

3. 在购买页中，参考以下信息购买预留券。如下图所示：



   - **实例名**：预留券的名称。不得超过60个字符。
   - **实例地域**：建议选择与您的客户最近的地域，可降低访问时延、提高访问速度。不同地域云产品之间内网不互通。详情可参见 查看支持地域 。
   - **资源规格类型**：
     - **CPU通用类型**：可以抵扣包括 Intel、AMD 型号 CPU Pod 产生的费用，并且会优先抵扣 Intel 型号。该类型适合您在混合使用 Intel、AMD 资源的时候采购。
     - **AMD专用类型**：仅可以抵扣 AMD 型号 CPU Pod 产生的费用，但单位时长的费用更低。该类型适合您仅使用 AMD 资源的时候采购。
   - **资源规格**：资源规格是容器（TKE Serverless 集群 Pod 或容器实例）可使用的最大资源量。预留券的资源规格需要和容器资源规格完全一致，以保证匹配抵扣和预留资源。
   - **保留资源（可选）**：代表该预留券是否要保留资源。若不勾选，则创建地域级预留券，可抵扣该地域下所有可用区的同规格按量 Pod 费用，不保留资源，按量 Pod 在资源紧张时可能会出现等待；若勾选，则创建可用区级/节点级预留券，可抵扣该可用区下/该节点下的同规格按量 Pod 费用，保留资源。（预留券在购买后，仍可进行有无保留资源的动态转换。）

- ○ **绑定超级节点（可选）：**代表该预留券是否要绑定到某个具体的超级节点，只可抵扣该节点上的 Pod。若勾选，则创建节点级预留券。如下图所示：

- ○ **数量：**一次购买数量，请尽量保证和希望抵扣的容器（TKE Serverless 集群 Pod 或容器实例）数量一致。

> **注意：**
> 单次购买数量上限为 300。

4. 选择购买时长，并勾选《腾讯云服务协议》。请在配置完成基本信息后确认已选规格及配置费用，计费详情请参见 产品定价 。
5. 单击**立即购买**购买已选配置的预留券。
6. 您可在**预留券**列表页查看已购买的预留券。

查看抵扣范围

在预留券概览页面，可查看预留券的抵扣范围，如下所示：

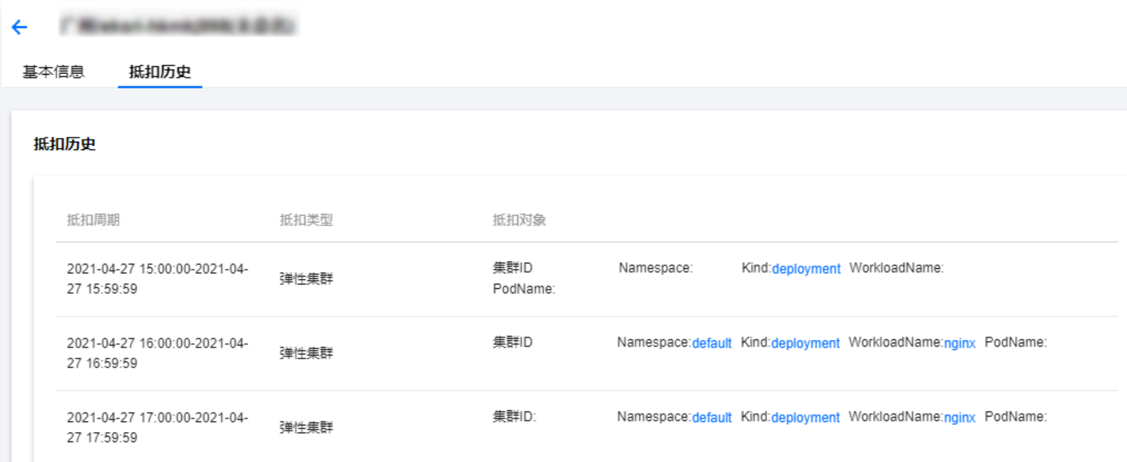
修改抵扣范围

当需要在地域级、可用区级、节点级预留券之间进行转换时，选择**更多 > 修改抵扣范围**，如下图所示：

在**修改以下预留券的抵扣范围页**中，选择想要修改的抵扣范围，单击**确定**，如下图所示：

查看预留券抵扣记录

1. 登录 容器服务控制台，选择左侧导航栏的**资源包管理 > 资源预留券**。
2. 在**预留券**列表页面，选择需要查看的预留券名称，进入预留券详情页。
3. 在**抵扣历史**页签中，查看该预留券的费用抵扣记录。如下图所示：



退还预留券

> **注意**
> 退还预留券属于费用有损的限制性操作，请勿随意操作。退费详情可参见 退费规则 。

1. 登录 容器服务控制台，选择左侧导航栏的**资源包管理 > 资源预留券**。
2. 在**预留券**列表页面，单击需要退还的预留券所在行右侧的**退还**。

3. 在退还预留券页中，单击**确定**即可退还当前所选预留券。如下图所示：

4. 您可在**预留券**列表页面查看到该预留券的状态为“已退还”，且不再进行费用抵扣和资源预留。

预留券退费说明

最近更新时间：2025-03-27 11:23:23

为了更加方便您使用 Serverless 容器服务预留券，如果您在购买预留券后有任何不满意，我们支持自助退货退款。您可在容器服务控制台自助进行，具体操作详情请参见 退还预留券 。

> **说明**
> - 每个账户可享受**1次1张**预留券的五天无理由退还，您支付的有效金额将返还至您的腾讯云账户。
> - 您可享受普通自助退还，扣除您已使用的费用，将按**购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户**的方式返还到您的账户。

五天无理由自助退还

弹性容器服务预留券遵守腾讯云 云服务退货说明，若您在购买预留券后有任何不满意，我们支持五天内无理由自助退还，具体策略如下：

- 弹性容器服务预留券**新购时长 ≤ 五天**，默认支持五天无理由退款。
- 对于每个主体，每个产品**只能退还1个实例**。例如，同一订单包含20张弹性容器服务预留券，最多只能退还其中1张预留券。
- 符合五天无理由退还场景的订单，退款金额为**购买时花费的全部消耗金额**，包括现金账户金额、收益转入账户金额以及赠送账户金额。
- 具体退款规则请参见 五天内无理由全额退款 。

普通自助退还

不满足五天无理由退货退款规则的退货订单，退款策略如下：
普通自助退还将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

普通自助退还限制

每个账户最多支持199张弹性容器服务预留券控制台自助退还。

普通自助退还规则

退费金额 = 实际支付金额 – (已使用时长 / 总时长) × 订单原价 × 订单折扣

- 已使用时长精确到天，不满一天按照一天计算。
- 退款金额 <= 0，按0计算并清退资源。

预留券自助退还计费规则示例

案例背景

在广州购买1张 CPU 通用型1核2GiB 资源规格的预留券。购买时长1年原价为1068元，包年享83折。购买时使用300元的代金券。
折后需付金额为 1068 × 0.83 = 886.44 (元)
实际支付金额为 886.44 – 300 = 586.44 (元)

五天无理由退还场景

案例

购买五天内发现不满意，想要退还，且该账户首次退还。
退费金额 = 实际支付金额 586.44 (元)

普通自助退还场景

案例 1

购买五天内，且该账户首次退还。
退还金额为 实际支付金额 586.44 (元)

案例 2

购买五天内，且不是该账户首次退还，总使用时长48小时，按照2天计算。
退还金额为 586.44 – (2 / 365) × 1068 × 0.83 = 581.58 (元)

案例 3

购买超过五天，总使用时长132小时，按照6天计算。

退还金额为 586.44 − (6 / 365) × 1068 × 0.83 = 571.87 (元)

案例 4

购买超过五天，总使用时长260天。

退还金额为 586.44 − (260 / 365) × 1068 × 0.83 < 0 (元)

由于购买时使用了代金券，而普通退还时不支持退还代金券，此时的退还金额为0，不建议进行退还操作。

竞价模式说明

最近更新时间：2025-06-09 17:21:12

竞价模式概述

TKE Serverless 竞价模式是一种低成本资源购置模式，其核心特点为价格低于按量付费实例，在一些场景可以极大降低运行容器的成本，但资源可能会被腾讯云中断回收。您在使用竞价模式运行时仅需支付较少的费用，并运行到容器资源被回收为止。

使用竞价模式时，您可以像使用普通按量计费资源一样在容器部署工作负载，且同样具备普通按量计费模式下的所有功能。

竞价模式策略

价格策略

目前 TKE Serverless 采用的竞价模式为**固定折扣比例（2折）**，即所有规格的竞价模式将以原规格 产品定价 的固定折扣出售。

> **注意：**
> 该折扣仅对 Pod 资源规格计费项（CPU、内存、GPU）生效。不包括网络带宽、网络流量、持久化存储等资源的费用。

回收中断机制

竞价模式下的容器会因为腾讯云计算资源池库存不足而产生回收中断。当库存不足时，会从已分配的竞价模式容器里随机回收，容器缓存数据不会保留。

回收中断时将会产生以下中断事件：

```
EVENT REASON : “SpotPodInterruption”
EVENT MESSAGE : “Spot pod was interrupted, it will be killed and re-created”
```

适用场景

适合短时长突发、周期任务

适用于不需要长期运行的突发性、周期性短时长工作负载。例如视频转码、视频渲染、服务压测、批量计算、爬虫等。

适合可切分的计算任务

适用于可以将长时间作业按作业对象切分为细粒度任务进行计算的系统。例如 EMR 等大数据套件。

适合无状态或者支持断点续传能力的计算任务

- 适用于将计算中间结果放到持久化存储上，可接受 Pod 被回收重启后继续运算的工作负载。
- 适用于支持自动负载均衡和服务发现的无状态工作负载，可接受 Pod 被回收重启的工作负载。

竞价模式开启

您可以通过在工作负载 YAML 中定义如下 Pod template annotation 方式，为工作负载开启竞价模式。

```
eks.tke.cloud.tencent.com/spot-pod: "true"
```

其他支持 Annotation 以及范例请参见 Annotation 说明 。

使用超级节点

新建超级节点

最近更新时间：2025-05-21 16:34:21

本文向您介绍如何通过容器服务控制台在集群内部署超级节点。

前提条件

- 请确保已经 创建集群 。
- 请确保集群类型为标准集群，推荐创建集群 Kubernetes 版本为1.28及以上版本，支持包年包月超级节点的集群版本为1.18、1.20、1.22、1.24、1.26、1.28、1.30。

操作步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击集群 ID，进入该集群详情页。
3. 选择左侧菜单栏中的**节点管理** ，在节点页面单击**新建节点池**。如下图所示：

4. 节点类型选择**超级节点**，单击**创建**，参考以下提示进行设置。如下图所示：

- **所属节点池**：支持**自动新建节点池**和**加入已有节点池**。选择自动新建节点池时将在创建超级节点时新建超级节点池；选择加入已有节点池可将新建的超级节点加入已有的节点池进行统一管理。超级节点所属的节点池支持加入不同计费模式，不同可用区和不同规格的超级节点。
- **节点池名称**：若自动新建节点池可对节点池进行命名管理。
- **超级节点配置**：

按量计费

- **可用区**：选择超级节点所在的可用区。
- **计费模式**：全地域支持按量计费模式。
- **容器网络**：为集群内容器分配在容器网络地址范围内的 IP 地址。集群内超级节点的 Pod 会直接占用 VPC 子网 IP（剩余 IP 数将限制调度至超级节点上的 Pod 数），请尽量选择 IP 数量充足且与其他产品使用无冲突的子网。超级节点上的 Pod 会直接运行在用户已指定的 VPC 网络上，每个 Pod 在生命周期内都会绑定一个指定 VPC 内的弹性网卡。您可前往 弹性网卡列表 查看 Pod 关联的网卡。

  按量计费模式下，支持选择多个子网，每个子网对应创建一个按量计费的超级节点，按量计费的超级节点创建时不计费，当实际创建工作负载后 Pod 调度至按量计费超级节点时开始按照 Pod 规格和实际运行时长计费。
- **节点名称**：可自动生成或自定义编辑名称，方便对节点进行命名管理。
- **腾讯云标签**：标签用于从不同维度对资源分类管理。如现有标签不符合您的要求，您亦可前往 标签控制台 统一管理标签。

- **预留券**：是一种预付费产品，用于抵扣按量计费产生的费用。支持按月购买，购买后可以对同地域、同资源规格类型、同资源规格的容器进行计算费用抵扣。

> **注意：**
> - 建议为容器网络配置多个可用区，这样您的工作负载在部署时会自动打散分布在多个可用区，可用性更高。
> - 请确保为容器网络分配 IP 充足的子网，避免创建大规模工作负载时因为 IP 资源耗尽无法创建 Pod。

包年包月

- **可用区**：选择超级节点所在的可用区。
- **计费模式**：北京、上海、广州、南京支持包年包月计费模式。
- **容器网络**：为集群内容器分配在容器网络地址范围内的 IP 地址。集群内超级节点的 Pod 会直接占用 VPC 子网 IP（剩余 IP 数将限制调度至超级节点上的 Pod 数），请尽量选择 IP 数量充足且与其他产品使用无冲突的子网。超级节点上的 Pod 会直接运行在用户已指定的 VPC 网络上，每个 Pod 在生命周期内都会绑定一个指定 VPC 内的弹性网卡。您可前往 弹性网卡列表 查看 Pod 关联的网卡。

  包年包月计费模式支持选择单个子网，如下图所示：

- **节点名称**：可自动生成或自定义编辑名称，方便对节点进行命名管理。
- **CPU 总规格**：可创建规格10-5000，总规格表示该包年包月超级节点上可调度的 Pod 的 CPU 总核数。
- **CPU 内存比**：可选择1:2、1:3、1:4规格的 CPU 内存比，这里指该包年包月超级节点上的 CPU 与内存总规格的比值，非超级节点上调度的 Pod 的 CPU 内存比，包年包月超级节点支持调度 CPU 内存比为1:1到1:4规格的 Pod。
- **购买时长**：包年包月超级节点的购买时长。
- **自动续费**：支持开启包月超级节点自动续费能力。
- **腾讯云标签**：标签用于从不同维度对资源分类管理。如现有标签不符合您的要求，您亦可前往 标签控制台 统一管理标签。

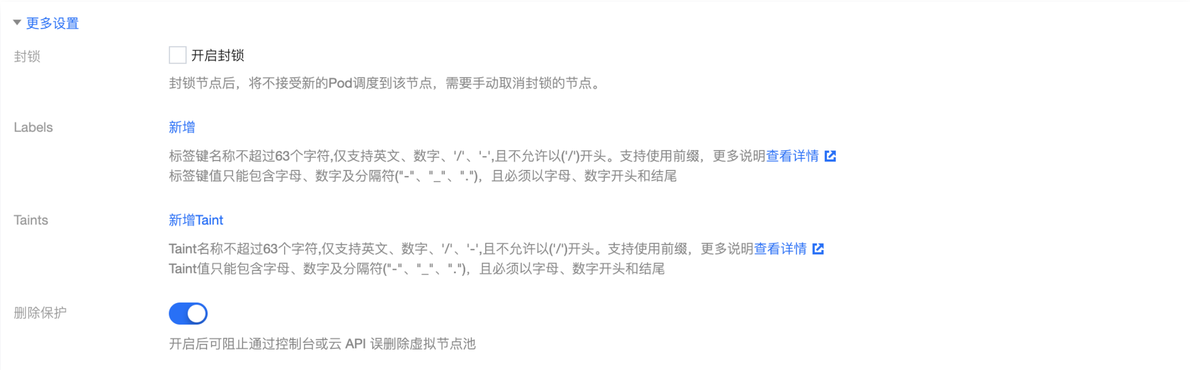
> **注意：**
> 建议为容器网络配置多个可用区，这样您的工作负载在部署时会自动打散分布在多个可用区，可用性更高。
> 请确保为容器网络分配 IP 充足的子网，避免创建大规模工作负载时因为 IP 资源耗尽无法创建 Pod。

  - **安全组配置**：详情见 容器服务安全组设置。

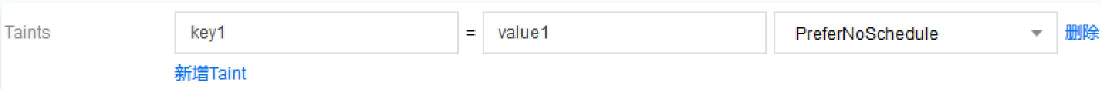
> **注意：**

- 超级节点配置的安全组会直接关联 Pod，请配置 Pod 工作所需的网络规则。例如，Pod 启用80端口提供服务，请放通入方向80端口的访问。
- 该安全组为调度到超级节点上 Pod 的默认绑定安全组，您可以在调度时指定其他安全组来覆盖默认安全组。

5. （可选）单击更多设置，查看或配置更多信息，如下图所示：



  - **封锁初始节点**：勾选**开启封锁**后，将不接受新的 Pod 调度到该节点，需要手动取消封锁的节点，或在自定义数据中执行 取消封锁命令，请按需设置。
  - **Labels**：单击**新增**，即可进行 Label 自定义设置。该节点池下所创建的超级节点均将自动增加此处设置的 Label，可用于后续根据 Label 筛选、管理超级节点。
  - **Taints**：节点属性，通常与 Tolerations 配合使用。此处可为节点池下的所有超级节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在的不符合条件的 Pod 也将会被驱逐。

> **说明：**
> Taints 内容一般由 `key` 、 `value` 及 `effect` 三个元素组成。其中 `effect` 可取值通常包含以下三种：
> - **PreferNoSchedule**：非强制性条件，尽量避免将 Pod 调度到设置了其不能容忍的 taint 的节点上。
> - **NoSchedule**：当节点上存在 taint 时，没有对应容忍的 Pod 一定不能被调度。
> - **NoExecute**：当节点上存在 taint 时，对于没有对应容忍的 Pod，不仅不会被调度到该节点上，该节点上已存在的 Pod 也会被驱逐。
>
> 以设置 Taints `key1=value1:PreferNoSchedule` 为例，控制台配置如下图所示：
>
> 
>

  - **删除保护**：开启后可阻止通过控制台或云 API 误删除超级节点池。

6. 单击**创建超级节点**。若选择了自动新建节点池，将同步创建纳管超级节点的超级节点池。

超级节点调度说明
超级节点可调度 Pod 说明

最近更新时间：2025-05-21 14:34:42

计费方式

调度到超级节点上的 Pod 支持预付费、后付费（按量计费、竞价）的两种计费模式。

支持超级节点的 Kubernetes 版本

- 按量计费超级节点支持 1.16 及以上版本集群。
- 包年包月超级节点当前支持 1.18、1.20、1.22、1.24、1.26、1.28、1.30版本的最高小版本集群。

超级节点上可调度的 Pod 规格

超级节点支持的 Pod 的规格配置是容器运行时可用资源和使用服务计费的依据，请务必了解超级节点 Pod 的资源规格配置。不同计费模式的超级节点支持的可调度 Pod 规格不同。

包年包月模式

- 支持调度 0.25C～8C 标准规格的 Pod。
- 支持调度 CPU 值大于等于1/4内存值的 Pod。
- 支持调度 intel 和 amd 类型的 Pod，如无明确指定将优先调度 intel 机型。若只指定 amd/intel，则不提供资源保障，且考虑到指定和不指定机型的定价一致，因此不建议指定机型去使用包月节点。

节点支持规格列表：

> **注意：**
> 若为非标准规格，则自动向上转换成标准规格。

CPU/核	内存区间/GiB	内存区间粒度/GiB
0.25	0.5、1、2	-
0.5	1、2、3、4	-
1	1 - 4	1
2	2 - 8	1
4	8 - 16	1
8	16 - 32	1

按量计费模式

- 支持调度0.25C～64C标准规格的 Pod（若为非标准规格，则自动向上转换成标准规格）。
- 支持调度 CPU 值大于1/8内存值的 Pod。

节点支持规格列表：

> **注意：**
> 若为非标准规格，则自动向上转换成标准规格。

CPU/核	内存区间/GiB	内存区间粒度/GiB
0.25	0.5、1、2	-
0.5	1、2、3、4	-

1	1 - 8	1
2	4 - 16	1
4	8 - 32	1
8	16 - 32	1
12	24 - 48	1
16	32 - 64	1
32	64、128、256	-
64	128、192、256、512	-

超级节点配置说明

Pod 临时存储

每个调度到超级节点上的 Pod，创建时会分配20GiB的临时镜像存储。

> **注意：**
> - 临时镜像存储将于 Pod 生命周期结束时删除，请勿用于存储重要数据。
> - 由于需存储镜像，实际可用空间小于20GiB。
> - 若需要扩容系统盘资源，可通过 Annotation 实现。
> - 重要数据、超大文件等推荐挂载 Volume 持久化存储。

Pod 网络

调度到超级节点上的 Pod 采用的是与云服务器、云数据库等云产品平级的 VPC 网络，每个 Pod 都会占用一个 VPC 子网 IP。
Pod 与 Pod、Pod 与其他同 VPC 云产品间可直接通过 VPC 网络通信，没有性能损耗。

Pod 隔离性

调度到超级节点上的 Pod 拥有与云服务器完全一致的安全隔离性。Pod 在腾讯云底层物理服务器上调度创建，创建时会通过虚拟化技术保证 Pod 间的资源隔离。

其他 Pod 特殊配置

调度到超级节点上的 Pod 可以通过在 yaml 中定义 `template annotation` 的方式，实现为 Pod 绑定安全组、分配资源、分配 EIP 等能力。配置方法见下表：

> **注意：**
> - 如果不指定安全组，则 Pod 会默认绑定节点池指定的安全组。请确保安全组的网络策略不影响该 Pod 正常工作，例如，Pod 启用 80 端口提供服务，请放通入方向80端口的访问。
> - 如需分配 CPU 资源，则必须同时填写 `cpu` 和 `mem` 2个 annotation，且数值必须符合 资源规格 中的 CPU 规格。
> - 如需通过 annotation 指定的方式分配 GPU 资源，则必须同时填写 `gpu-type` 及 `gpu-count` 2个 annotation，且数值必须符合 资源规格 中的 GPU 规格。

Annotation Key	Annotation Value 及描述	是否必填
eks.tke.cloud.tencent.com/security-group-id	工作负载默认绑定的安全组，请填写 安全组 ID：可填写多个，以 `,` 分割。例如 `sg-id1,sg-id2` 。网络策略按安全组顺序生效。	否。如不填写，则默认绑定节点池指定的安全组。 如填写，请确保同地域已存在该安全组 ID。
eks.tke.cloud.tencent.com/cpu	Pod 所需的 CPU 核数，请参考 资源规格 填写。默认单位为核，无需再次注明。	否。如填写，请确保为支持的规格，且需完整填写 `cpu` 和 `mem` 两个参数。

eks.tke.cloud.tencent.com/mem	Pod 所需的内存数量，请参考 资源规格 填写，需注明单位。例如，512Mi、0.5Gi、1Gi。	否。如填写，请确保为支持的规格，且需完整填写 `cpu` 和 `mem` 两个参数。
eks.tke.cloud.tencent.com/cpu-type	Pod 所需的 CPU 资源型号，目前支持型号如下：intel,amd。	否。如果不填写则默认不强制指定 CPU 类型，会根据 指定资源规格方法 尽量匹配最合适的规格，若匹配到的规格 Intel 和 amd 均支持，则优先选择 Intel。
eks.tke.cloud.tencent.com/gpu-type	Pod 所需的 GPU 资源型号，目前支持型号如下：V100、1/4 T4、1/2 T4、T4、A10，支持优先级顺序写法。如 "T4,V100" 表示优先创建 T4 资源 Pod，如果所选地域可用区 T4 资源不足，则会创建 V100 资源 Pod。各型号支持的具体配置请参考 资源规格 。	如需 GPU，则此项为必填项。填写时，请确保为支持的 GPU 型号，否则会报错。
eks.tke.cloud.tencent.com/gpu-count	Pod 所需的 GPU 数量，请参考 资源规格 填写，默认单位为卡，无需再次注明。	否。如填写，请确保为支持的规格。
eks.tke.cloud.tencent.com/retain-ip	Pod 固定 IP，value 填写 `"true"` 开启此特性，开启特性的 Pod，当 Pod 被销毁后，默认会保留这个 Pod 的 IP 24 小时。24 小时内 Pod 重建，还能使用该 IP。24 小时以后，该 IP 有可能被其他 Pod 抢占。**仅对 statefulset、rawpod 生效。**	否
eks.tke.cloud.tencent.com/retain-ip-hours	修改 Pod 固定 IP 的默认时长，value 填写数值，单位是小时。默认是 24 小时，最大可支持保留一年。**仅对 statefulset、rawpod 生效。**	否
eks.tke.cloud.tencent.com/eip-attributes	表明该 Workload 的 Pod 需要关联 EIP，值为 `""` 时表明采用 EIP 默认配置创建。`""` 内可填写 EIP 云 API 参数 json，实现自定义配置。例如 annotation 的值为 '{"InternetMaxBandwidthOut":2}' 即为使用 2M 的带宽。注意，非带宽上移的账号无法使用。	否
eks.tke.cloud.tencent.com/eip-claim-delete-policy	Pod 删除后，EIP 是否自动回收，"Never" 不回收，默认回收。该参数只有在指定 eks.tke.cloud.tencent.com/eip-attributes 时才生效。注意，非带宽上移的账号无法使用。	否
eks.tke.cloud.tencent.com/eip-id-list	如果工作负载为 StatefulSet，也可以使用指定已有 EIP 的方式，可指定多个，如 "eip-xx1,eip-xx2"。请注意，StatefulSet pod 的数量必须小于等于此 annotation 中指定 EIP Id 的数量，否则分配不到 EIP 的 pod 会处于 Pending 状态。注意，非带宽上移的账号无法使用。	否

示例请参考 Annotation 说明 。

默认配额

购买包年包月的超级节点时，将按照总规格分配默认配额，开通按量计费的超级节点，默认每个集群可将500个 Pod 调度到超级节点上。若您需要超过以上配额的资源，可填写提升配额申请，由腾讯云对您的实际需求进行评估，评估通过之后将为您提升配额。

申请提升配额操作指引

1. 请 提交工单，选择 **Serverless 容器服务-集群/容器实例配额申请-创建工单**，进入创建工单信息填写页面。
2. 在问题描述中填写"期望提升集群超级节点 Pod 配额"，注明目标地区及目标配额，并按照页面提示填写您可用的手机号等信息。
3. 填写完成后，单击**下一步-提交工单**即可。

Pod 限制说明

Service 限制

采用 GlobalRouter 网络模式 的集群 service 如果开启了 externaltrafficpolicy = local，流量不会转发到调度到超级节点上的 Pod。

Volume 限制

1. 支持以下类型的 Volume：EmptyDir、PVC、Secret、NFS、ConfigMap、Downward API、HostPath、GitRepo、ISCSI、Projected、CSI、Ephemeral。
2. 针对 PVC 类型的 Volume：
    - PV 类型：支持 NFS、CephFS、HostPath、静态 cbs 类型，不支持其他类型。
    - Storageclass 类型：支持用户自定义、cloud.tencent.com/qcloud-cbs、com.tencent.cloud.csi.cbs、com.tencent.cloud.csi.cfs，不支持其他类型。

GPU 限制

必须在 Annotation 中指定 gpu-type 字段，否则不支持调度到超级节点上；不同 type 的 GPU Pod 对应的 cpu、mem 规格是固定的，可不指定 cpu、mem 大小，若需要指定大小，则必须与 GPU Pod 支持的规格完全一致，否则将调度失败。

其他限制

- 没有任何服务器节点的空集群暂时无法正常使用超级节点功能。
- 指定了 hostIP 配置的 Pod 默认会把 Pod IP 作为 hostIP。
- 调度到超级节点上的 Pod 是强隔离的，如果开启了硬反亲和性特性，调度到超级节点上不会生效，会存在调度多个同一工作负载的 Pod 在同一个超级节点上的情况。
- tke-eni-ip-webhook 命名空间下的 Pod 不支持调度到超级节点。

调度 Pod 至超级节点

最近更新时间：2025-05-27 10:12:11

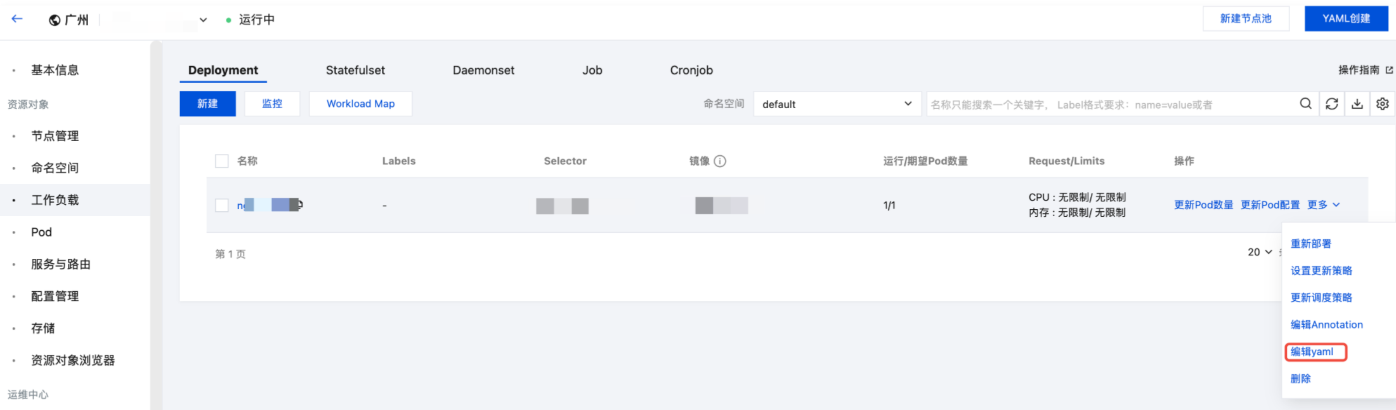
本篇文章主要介绍在容器服务 TKE 集群中，如何通过自动调度或手动调度 Pod 至超级节点。

自动调度

- 若集群配置了包年包月超级节点，在 Pod 符合超级节点调度规则的前提下，包年包月的超级节点与包年包月的常规节点平权调度。
- 若集群配置了按量计费超级节点，则当业务高峰且已有包年包月节点资源不足时，会自动调度 Pod 至超级节点，无需购买服务器；业务恢复平稳，自动释放在超级节点中的 Pod 资源，也无需再进行退还机器操作。
- 如果集群同时开启了 Cluster Autoscaler 和按量计费超级节点，则会尽量优先将 Pod 调度到按量计费的超级节点上，而非触发集群节点扩容。如果受调度限制影响，Pod 无法调度到超级节点上，则会依然正常触发集群节点扩容。而服务器节点资源充足时，会优先缩容按量计费超级节点上的 Pod。

手动调度

支持用户手动将 Pod 调度至超级节点，默认按量计费的超级节点会自动添加 Taints 以降低调度优先级，如需手动调度 Pod 到超级节点或指定超级节点调度，通常需要为 Pod 添加对应的 Tolerations。但并非所有的 Pod 均可以调度到超级节点上，详情请参见 超级节点调度说明 。为方便使用，您可以在 Pod Spec 中指定 nodeSelector。示例如下：




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      nodeSelector:
        node.kubernetes.io/instance-type: eklet
      containers:
      - name: my-container
        image: nginx
        ports:
        - containerPort: 80
```

容器服务 TKE 的管控组件会判断该 Pod 是否可以调度到超级节点，若不支持则不会调度到超级节点。
包年包月的超级节点当前仅支持调度指定规格和指定 CPU 内存比的 Pod，若不符合规则，则调度不会成功。调度至按量计费的超级节点时没有规格和 CPU 内存比的约束，只要按量节点所在的可用区有对应资源则可调度成功。

超级节点 Annotation 说明

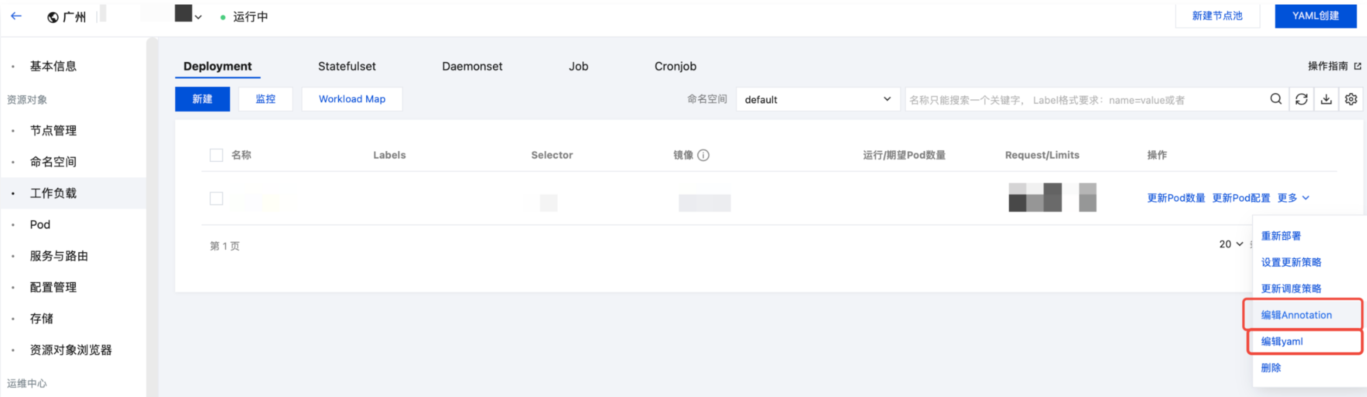
最近更新时间：2025-05-26 17:00:01

本文介绍超级节点特有的 Annotation 与示例，该 Annotation 针对 TKE 标准集群和 TKE Serverless 集群内超级节点上运行的 Pod 生效。

Annotation 使用方法

控制台操作步骤如下：

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表页中，单击**集群 ID**，进入该集群详情页。
3. 选择左侧导航中的**工作负载**，在右侧**操作 > 更多**中选择**编辑 Annotation/编辑 yaml**。如下图所示：



工作负载里添加 Pod 注解

本文所说明的注解均为在 Pod 级别添加注解，通常用户使用的是工作负载而不是裸 Pod。本文以在 Deployment 上添加 Pod 注解为例，需注意，工作负载里添加 Pod 注解是在 `.spec.template.metadata.annotations` 字段。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.tke.cloud.tencent.com/retain-ip: 'true' # 工作负载里添加 Pod 注解是在
.spec.template.metadata.annotations 字段
    spec:
      containers:
      - name: nginx
        image: nginx
```

全局配置

如果用户希望注解默认对集群里所有 Pod 生效，可以通过修改全局配置实现。需注意，全局配置在 kube-system 命名空间下名为 eks-config 的 configmap，如没有该资源则需要自行新建。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: eks-config
  namespace: kube-system
data:
  pod.annotations: |
    eks.tke.cloud.tencent.com/resolv-conf: |
      nameserver 183.60.83.19
```



```
eks.tke.cloud.tencent.com/host-sysctls: '[{"name": "net.core.rmem_max","value": "26214400"}]'
```

> **说明：**
> 添加在 Pod 上的注解优先级高于全局配置。

资源与规格

指定 CPU 与内存

超级节点上运行的 Pod 默认会根据 request 与 limit 自动计算出底层资源的规格，用户可以参考官方文档 指定资源规格，也可以通过给 Pod 添加注解去指定 Pod 需要的计算资源规格。示例如下：

```
eks.tke.cloud.tencent.com/cpu: '8'
eks.tke.cloud.tencent.com/mem: '16Gi' # 内存一定要以 Gi 为单位，以 G 为单位则会报参数错误
```

指定系统盘大小及类型

超级节点上运行的 Pod 默认免费提供 20GiB 高性能系统盘（不额外申明 Annotation 时的默认配置），系统盘生命周期与 Pod 的生命周期一致，可满足通用场景的系统盘资源诉求，若特殊场景下用户需要额外扩容系统盘大小或额外指定磁盘类型，支持通过 Annotation 来声明所需系统盘的大小及类型。注意，此时超过 20GiB 的部分将按照指定磁盘类型的按量计费的刊例价进行计费，计费详情请参见 云硬盘定价说明。系统盘大小及类型注解示例如下，可单独使用：

```
eks.tke.cloud.tencent.com/root-cbs-size: '50'  # 指定系统盘大小，超过20GiB额外按高性能云盘计费，最大1000GiB
eks.tke.cloud.tencent.com/root-cbs-type: 'CLOUD_SSD' # 指定系统盘类型为SSD，不额外指定时默认为高性能磁盘，超过20GiB额外按照SSD磁盘按量计费的刊例价进行计费。
```

规格自动升配

超级节点上运行的 Pod 会按照 request 与 limit 自动计算出匹配的底层资源规格。如果对应的规格底层无相应资源，则会创建失败并在事件中提示资源不足(insufficient resource)。如果用户接受使用更高规格的资源来创建 Pod，则可在 Pod 添加如下注解，开启自动升配，计费将按照升配后的规格计算。

```
eks.tke.cloud.tencent.com/spec-auto-upgrade: 'true' # 遇到资源不足时，开启自动升配，只按 CPU 规格向上升配一次
```

指定 GPU

如需指定 GPU，可在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/gpu-type: 'T4,V100' # 指定 GPU 型号，支持优先级顺序写法，若使用1/4卡的 T4 vGPU 则指定 GPU 型号为1/4*T4。
```

> **说明：**
> TKE Serverless 支持通过 Annotation 指定和通过 Request 及 Limit 计算两种声明方式指定资源规格。
> 如通过 Request 及 Limit 计算，声明 gpu-type 后我们将为您自动计算出合适的规格。
> 如通过 Annotation 指定，则需填写如下注解，注解中任一错误，都会导致创建失败，规格详情请参见 资源规格。
>
> ```
> eks.tke.cloud.tencent.com/gpu-type: 'V100'
> eks.tke.cloud.tencent.com/count: '1'
> eks.tke.cloud.tencent.com/cpu: '8'
> eks.tke.cloud.tencent.com/mem: '40'
> ```
>
> 建议您直接声明 gpu-type，通过 Request 及 Limit 计算指定资源。

指定 CPU 类型

如需指定 CPU 类型，可在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/cpu-type: 'amd,intel' # 表示优先创建 amd 资源 Pod，如果所选地域可用区 amd 资源不足，则会创建 intel 资源 Pod
```

开启竞价模式

如需开启 竞价模式，可在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/spot-pod: 'true'
```

IP 保留与 EIP

StatefulSet 固定 IP

使用固定 IP 的前提是工作负载为 StatefulSet，或者直接使用裸 Pod（需注意 Pod 名称不能改变），用户可在 Pod 级别添加注解启用固定 IP：

```
eks.tke.cloud.tencent.com/retain-ip: 'true' # 置为 true 启用固定 IP。
eks.tke.cloud.tencent.com/retain-ip-hours: '48' # 保留 IP 的最大时长（小时），Pod 销毁之后超过这个时长没有创建回来，IP 将被释放。
```

> **说明：**
> TKE 集群超级节点场景下的固定 IP，用法保持跟 TKE 一致，无需添加上述注解。

绑定 EIP

如需绑定 EIP，可在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/eip-attributes: '{"InternetMaxBandwidthOut":50, "InternetChargeType":"TRAFFIC_POSTPAID_BY_HOUR"}' # 值可以为空串，表示启用 EIP 并使用默认配置；也可以用创建 EIP 接口的 json 参数，详细参数列表参考 https://cloud.tencent.com/document/api/215/16699#2.-.E8.BE.93.E5.85.A5.E5.8F.82.E6.95.B0，本例中的参数表示 EIP 是按量付费，且带宽上限为 50M。
```

> **注意：**
> 非带宽上移的账号（传统账户），不支持绑定 EIP，若您无法确认账户类型，请参见 判断账户类型，若为非带宽上移账户，可 提交工单 进行账户升级。
> TKE 标准集群超级节点场景，无需添加上述注解，用法请参见 Pod 直接绑定弹性公网 IP 使用说明 。

StatefulSet 固定 EIP

如需在 StatefulSet 固定 EIP，可在 Pod 级别添加如下注解：

```
eks.tke.cloud.tencent.com/eip-attributes: '{}' # 启用 EIP 并使用默认配置。
eks.tke.cloud.tencent.com/eip-claim-delete-policy: 'Never' # Pod 删除后，EIP 是否自动回收，默认回收。使用 "Never" 不回收，即下次同名 Pod 创建出来仍然会绑定此 EIP，实现固定 EIP。
```

> **注意：**
> Deployment 类型的工作负载使用 `Never` 不回收策略时，Pod 删除后 EIP 不会回收，滚动更新后的 Pod 也不会使用原本的 EIP。

StatefulSet 使用已有 EIP

如需将已有的 EIP 绑定到 Pod 而不是自动创建，可以给 Pod 指定要绑定的 EIP 实例 ID。注解如下：

```
eks.tke.cloud.tencent.com/eip-id-list: 'eip-xx1,eip-xx2' # 这里指定已有的 EIP 实例列表，确保 StatefulSet 的 Pod 副本数小于等于这里的 EIP 实例数。
```

> ⚠ **注意：**
> 指定 EIP 场景下，请勿配置 eip-attributes 注解。

镜像与仓库

忽略证书校验

如果自建镜像仓库的证书是自签的，拉取镜像会失败 (ErrImagePull)，可以给 Pod 添加以下注解指定拉取此镜像仓库的镜像时不校验证书：

```
eks.tke.cloud.tencent.com/registry-insecure-skip-verify: 'harbor.example.com' # 也可以写多个，逗号隔开
```

使用 HTTP 协议

如果自建镜像仓库使用的 HTTP 而不是 HTTPS，拉取镜像会失败 (ErrImagePull)，可以给 Pod 添加以下注解指定拉取此镜像仓库的镜像时使用 HTTP 协议：

```
eks.tke.cloud.tencent.com/registry-http-endpoint: 'harbor.example.com' # 也可以写多个，逗号隔开
```

镜像复用

超级节点上默认会复用系统盘以加快启动速度，复用的是同一工作负载下相同可用区 Pod 且在缓存时间内（销毁后2小时内）的系统盘。如需复用不同工作负载的 Pod 的镜像，可以在不同工作负载的 Pod 上打上相同的 `cbs-reuse-key` 的注解：

```
eks.tke.cloud.tencent.com/cbs-reuse-key: 'image-name'
```

镜像缓存

超级节点上提供 镜像缓存能力，即提前创建好镜像缓存实例，自动将需要的镜像下载下来并创建云盘快照，后续创建 Pod 开启镜像缓存，这样就会根据镜像名自动匹配镜像缓存实例的快照，直接使用快照里面的镜像内容，避免重复下载，加快 Pod 启动速度。
启用镜像缓存的注解：

```
eks.tke.cloud.tencent.com/use-image-cache: 'auto'
```

指定镜像缓存创建出来的盘类型：

```
eks.tke.cloud.tencent.com/image-cache-disk-type: 'CLOUD_SSD'  # 指定镜像缓存创建出来的盘类型，可取值如下：
CLOUD_BASIC为普通云硬盘，CLOUD_PREMIUM为高性能云硬盘（默认），CLOUD_SSD为SSD云硬盘，CLOUD_HSSD为增强型SSD云硬盘，
CLOUD_TSSD为极速型SSD云硬盘
```

指定镜像缓存创建出来的盘的大小：

```
eks.tke.cloud.tencent.com/image-cache-disk-size: '50' # 指定镜像缓存创建出来的盘的大小，默认是镜像缓存创建时设置的大
小，可以调大，不能调小，最大支持1000GiB
```

用户也可以手动指定镜像缓存实例，不使用自动匹配：

```
eks.tke.cloud.tencent.com/use-image-cache: 'imc-xxx'
```

镜像缓存创建出来的数据盘，在 Pod 删除后会立即销毁。如果需额外保留镜像缓存创建的数据盘，可以通过注解指定保留时间。开启数据盘保留后，再次创建 Pod 的时候会直接复用保留中的盘，可节省数据再重新从快照回滚至新云硬盘的时间：

```
eks.tke.cloud.tencent.com/image-cache-disk-retain-minute: '10' # 设定镜像缓存创建的数据盘，在销毁 Pod 后保留 10
分钟
```

绑定安全组

超级节点默认会为 Pod 绑定同地域默认项目下的 `default` 安全组，也可以通过给 Pod 加注解绑定指定安全组：

```
eks.tke.cloud.tencent.com/security-group-id: 'sg-id1,sg-id2' # 填本地域存在的安全组 id，多个用逗号隔开，网络策略按安全组顺序生效，安全组默认最多只能绑定 2000 个 Pod，如需更多请提工单提升配额。
```

绑定角色

超级节点上默认会为 Pod 关联 `TKE_QCSLinkedRoleInEKSLog` 角色，授予 Pod 里日志采集组件上报日志的权限。用户也可通过注解为 Pod 关联其他 CAM 角色，方便从 Pod 内获取操作云上资源的权限。

```
eks.tke.cloud.tencent.com/role-name: 'TKE_QCSLinkedRoleInEKSLog'
```

设置宿主机内核参数

有些内核参数以网络命名空间隔离，无法在容器内查看或设置，超级节点内 Pod 虽然是独占虚拟机，但不代表就能在容器内直接设置宿主机上的内核参数。可以通过给 Pod 加注解来实现设置 Pod 宿主机内核参数：

```
eks.tke.cloud.tencent.com/host-sysctls: '[{"name": "net.core.rmem_max","value": "26214400"},{"name": "net.core.wmem_max","value": "26214400"},{"name": "net.core.rmem_default","value": "26214400"},{"name": "net.core.wmem_default","value": "26214400"}]'
```

加载内核模块

如需加载 TOA 内核模块，可在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/host-modprobe: 'toa'
```

自动重建自愈

超级节点所在集群虚拟机内的 agent 会上报心跳给控制面，如果上报超时（默认 5min），一般说明 Pod 内进程已经无法正常工作了，故障原因通常是高负载，这时集群默认会自动迁移虚拟机（对当前虚拟机关机并自动创建新虚拟机，让 Pod 迁移到新虚拟机里去运行），从而实现自愈。
如果用户不希望自动重建（如用于保留现场），可以在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/recreate-node-lost-pod: "false"
```

如果用户认为默认的 5min 上报超时时间过长，希望及时摘除异常 Pod 的流量，可以在 Pod 上添加如下注解来自定义心跳超时时间：

```
eks.tke.cloud.tencent.com/heartbeat-lost-period: 1m
```

磁盘清理

当超级节点内 Pod 磁盘空间紧张的时候，会自动触发清理流程以释放空间，通过 `df -h` 可以确认磁盘使用情况。
常见磁盘空间不足的原因有：
- 业务有大量临时输出。您可以通过 du 命令确认。
- 业务持有已删除的文件描述符，导致磁盘空间未释放。您可以通过 lsof 命令确认。

清理容器镜像

默认情况下，磁盘空间达 80% 以上会自动清理容器镜像来释放空间，如果已经没有容器镜像可以释放，则会报告如下事件：

```
failed to garbage collect required amount of images. Wanted to free 7980402688 bytes, but freed 0 bytes
```

如需自定义清理的阈值，可以在 Pod 上添加如下注解：

```
eks.tke.cloud.tencent.com/image-gc-high-threshold: '80' # 表示磁盘使用空间达 80% 触发清理容器镜像
```



```
eks.tke.cloud.tencent.com/image-gc-low-threshold: '75' # 触发清理容器镜像后，默认释放 5% (high-threshold - low-threshold) 的磁盘空间，便停止清理
eks.tke.cloud.tencent.com/image-gc-period: '3m' # 磁盘空间检查的间隔，默认 3 分钟
```

清理已退出的容器

如果业务原地升级过，或者容器异常退出过，已退出的容器仍会保留，直到磁盘空间达到 85% 时才会清理已退出的容器。清理阈值可以使用如下 Annotation 调整：

```
eks.tke.cloud.tencent.com/container-gc-threshold: "85"
```

如果已退出的容器不想被自动清理（例如需要退出的信息进一步排障），可以通过如下 Annotation 关闭容器的自动清理，但副作用是磁盘空间无法自动释放：

```
eks.tke.cloud.tencent.com/must-keep-last-container: "true"
```

重启磁盘用量高的 Pod

业务需要在容器的系统盘用量超过某个百分比后直接重启 Pod，可以通过 Annotation 配置：

```
eks.tke.cloud.tencent.com/pod-eviction-threshold: "85" # 此功能只在设置后开启，默认不开启
```

只重启 Pod，不会重建子机，退出和启动都会进行正常的 gracestop、prestop、健康检查。

> **注意：**
> 此特性上线时间为 2022-04-27，故在此时间前创建的 Pod，需要重建 Pod 来开启特性。

监控指标

9100 端口问题

超级节点上 的 pod 默认会通过 9100 端口对外暴露监控数据，用户可以执行以下命令访问 9100/metrics 获取数据：

- 获取全部指标：

```
curl -g "http://<pod-ip>:9100/metrics"
```

- 大集群建议去掉 ipvs 指标：

```
curl -g "http://<pod-ip>:9100/metrics?collect[]=ipvs"
```

如果业务本身直接监听了 9100 端口，将会有如下报错，提醒用户 9100 端口已经被使用：

```
listen() to 0.0.0.0:9100, backlog 511 failed (1: Operation not permitted)
```

您可以自定义暴露监控数据的端口号，避免与业务冲突。配置方式如下：

```
eks.tke.cloud.tencent.com/metrics-port: "9110"
```

> **说明：**
> 如果 Pod 带有公网 EIP，则需要设置安全组，注意 9100 端口问题，并放通需要的端口。

监控数据上报频率

超级节点上暴露的 cAdvisor 监控数据，默认 30 秒刷新一次，可以通过如下注解调整刷新频率：

```
eks.tke.cloud.tencent.com/cri-stats-interval: '30s'
```

自定义监控指标路径

监控数据默认通过 `/metrics` 路径暴露监控数据，无需改动。如有自定义的需求，可以用下面的注解：

```
eks.tke.cloud.tencent.com/custom-metrics-url: '/metrics'
```

设置 Pod 网卡名称

Pod 默认使用 `eth0` 网卡，如果需更改网卡名称，需要添加如下注解：

```
internal.eks.tke.cloud.tencent.com/pod-eth-idx: '1' # 设置网卡名为 eth1
```

自定义 DNS

Pod 所在宿主机默认使用 VPC 内的 DNS：183.60.83.19，183.60.82.98。如果需要更改宿主机上的 DNS，可以配置如下 annotation：

```
eks.tke.cloud.tencent.com/resolv-conf: |
  nameserver 4.4.4.4
  nameserver 8.8.8.8
```

注意：

1. 配置了自定义 DNS 后，则宿主机上的 DNS 以配置的为准。
2. 只对新增 Pod 有效。如果在 configmap kube-system/coredns 中配置了 forward . /etc/resolv.conf，则需重建 coredns Pod 才能生效此注解指定的 DNS为上游（重建 coredns Pod 可能影响业务，建议低峰期操作）。

日志采集到 kafka

超级节点支持通过开源的日志采集组件将日志采集到 kafka。如果采集的是容器内文件，或者采集时进行了多行合并，则需合理配置单行大小。默认设置的单行大小是 2M，超过 2M 则会丢弃该行日志。日志的单行大小可通过如下注解调整：

```
internal.eks.tke.cloud.tencent.com/tail-buffer-max-size: '2M' # 默认支持最大 2M 的单行日志
```

采集到 kafka 时，日志内容的字段为 `log`，如需将字段名配置为 `message`，则可通过如下注解调整：

```
eks.tke.cloud.tencent.com/log-key-as-message: 'true'
```

日志上报时会额外携带 Pod 的 metadata 信息，信息以 `string` 的格式放在 `kubernetes` 字段，如果需要将 `kubernetes` 字段设置成 `json` 格式，需配置如下注解：

```
eks.tke.cloud.tencent.com/filebeat-metadata-format: 'true'
```

延迟销毁

超级节点上运行的 Job 任务结束后，底层资源就会被销毁；相应的，运行时产生的临时输出，也会被销毁，此时执行 `kubectl logs <pod>` 会提示 `can not found connection to pod`。如果需要保留底层资源定位问题，可以设置延迟销毁底层资源：

```
eks.tke.cloud.tencent.com/reserve-sandbox-duration: '1m' # 开启延迟 1 分钟销毁的功能，Failed 状态的 Pod 最后一个容器退出后，额外保留底层资源 1 分钟
eks.tke.cloud.tencent.com/reserve-succeeded-sandbox: 'false' # 延迟销毁默认只对 Failed 状态的 Pod 生效，变更此字段可将延迟销毁也应用于 Succeeded 状态的 Pod
eks.tke.cloud.tencent.com/reserve-task-shorter-than: '5s' # 如果只关心运行时间较短的 Job，可以配置此参数，只有 Pod 内有任何一个容器的运行时间低于指定值，才会触发延迟销毁。默认不开启。
```


> ⚠ **注意：**
> 开启延迟销毁后，会将子机销毁的时间作为 pod 的结束时间对 pod 进行计费。

Service 规则同步

关闭 Service 规则同步

超级节点所在集群通过 IPVS 生成 K8s service 规则。 如果无需生成 service 规则，可通过如下注解关闭：

```
eks.tke.cloud.tencent.com/cluster-ip-switch: 'disable'
```

> ⚠ **注意：**
> 关闭 service 规则后，则集群内的服务发现都会失效。Pod 将无法使用 `ClusterFirst` 的 `dnsPolicy`，需改成其他类型，例如 `Default`。

等待 Service 规则同步

集群在 Pod 启动的同时同步 service 规则，如果需要在 Pod 启动前先等待 service 规则同步完成，则可以配置：

```
eks.tke.cloud.tencent.com/duration-to-wait-service-rules: '30s' # 启动 Pod 前先等待 service 规则同步完成，此处设置最多等待的时间为 30s
```

设置 IPVS 参数

IPVS 规则可通过如下注解控制：

```
eks.tke.cloud.tencent.com/ipvs-scheduler: 'sh' # 调度算法，sh 是 source hash，按源地址 hash 进行转发，更利于分布式全局的负载均衡。默认为 rr （轮询 round robin）
eks.tke.cloud.tencent.com/ipvs-sh-port: "true" # 按端口进行 source hash，仅在 ipvs-scheduler 为 sh 有效。
eks.tke.cloud.tencent.com/ipvs-sync-period: '30s' # 规则刷新的最大间隔，默认 30s 刷新一次。
eks.tke.cloud.tencent.com/ipvs-min-sync-period: '2s' # 规则刷新的最小间隔，默认 service 一有变更就刷新。调整此参数可避免刷新过于频繁。
```

设置集群内访问 CLB 的 VIP 时流量不走 IPVS：
适用于希望集群内访问不走 ipvs 而走 clb，通过给 service 上配置该 annotation 后，不会再生成对应的 ipvs 规则：

```
service.cloud.tencent.com/discard-loadbalancer-ip: 'true' # 该 annotation 配置在 service 上，无需重建 Pod 即可即时生效
```

自定义 Pod 底层的子机时区

超级节点上的 Pod 所在的子机默认为 UTC 时间，若需要调整子机时区为东8区，可添加如下 Annotation：

```
eks.tke.cloud.tencent.com/host-timezone: 'Asia/Shanghai' # 该 annotation 用于设置 Pod 时区为东8区
```

超级节点 Pod 支持云标签

超级节点上的 Pod 会默认继承集群、超级节点池、超级节点的云标签，也支持通过 Annotation 为 Pod 额外打上云标签，方便用户以更细的粒度对资源进行分账管理，对应 Annotation 如下：

```
eks.tke.cloud.tencent.com/resource-tag: '{"key0":"val0","key1":"val1"}' # 该 annotation 用于设置 Pod 云标签为 key0=val0,key1=val1
```

> ⚠ **注意：**

> 若集群维度、超级节点维度和 Pod 维度均有标签，则 Pod 维度会对所有标签进行合并，当出现相同 key 值不同 value 值时，Pod 维度云标签的生效顺序为 Pod > 超级节点 > 集群， 即当集群维度、超级节点维度和 Pod 维度的标签键值冲突的时候，以 Pod 维度的 value 值为准。

超级节点 Pod 支持获取 eks-id

超级节点中的 Pod 以 eks-id 作为资源的唯一 id，eks-id 应用于按量计费的 Pod 账单中，若用户需要使用 Pod 信息与账单中的 eks-id 进行对账，可通过如下 annotation 将 eks-id 写入 Pod 的 yaml 后进行获取。

```
eks.tke.cloud.tencent.com/eks-id: "true" # 该 annotation 用于设置 pod yaml 中是否展示 eks-id
```

指定资源规格

最近更新时间：2024-11-14 17:25:32

TKE Serverless 支持 通过 Annotation 指定 及 通过 Request、Limit 自动计算 两种方式，指定为 Pod 分配的资源上限。您可选择其中一种方式进行配置。

通过 Annotation 指定

TKE Serverless 支持在工作负载 yaml 中以添加 `template annotation` 的方式，显式的指定 Pod 资源规格。详情请参见 Annotation 说明 。

通过 Request、Limit 自动计算

TKE Serverless 支持对工作负载设置的 Request 及 Limit 进行计算，自动判断 Pod 运行所需的资源量。根据 Pod 资源类型的不同，其计算方法也略有差异。请参考 CPU Pod 规格计算方法 及 GPU Pod 规格计算方法，进一步了解如何通过 Request、Limit 自动计算指定资源规格。

> **注意：**
> - 如指定了工作负载 `template annotation` ，则以 Annotation 配置为准，不进行 Request 及 Limit 核算。
> - Request 及 Limit 的分配请参考 资源规格 中的 CPU、GPU 支持规格，若设定值与支持规格差别过大可能会导致某项资源分配超预期，造成资源浪费。
> - 无论如何设置 Request 及 Limit，其最终计算结果都会与 资源规格 进行匹配，且最终 Pod 分配的实际资源一定不会超出其中允许的规格。
> - 如 Pod 内所有容器都未设置 Request 及 Limit，则默认使用 Pod 规格为1核2GiB。
> - 如 Pod 内有容器设置了 Request 及 Limit，则分别按下述方式计算 Initcontainer 和业务 Container 值，最终取较大者来匹配 Pod 规格：
>   - Initcontainer 大小计算方式：按照单个 init 容器的 request/limit 判断各自对应的 CPU 内存值，取较大者，若存在单个 initcontainer 未填写 request/limit 值，取默认值1C2GiB。
>   - 业务 Container 大小计算方式：业务容器 request 的和值与业务容器 limit 的最大值中的较大值，若业务容器为空，则为1C2GiB。

CPU Pod 规格计算方法

步骤1：分别计算 Pod 的 CPU、Memory 的合计数值

合计数值分别为 **Pod 内所有容器的 Request 之和**、**Pod 内所有容器的 Limit 中的最大值**中的较大者。

步骤2：按以下情况匹配 Pod 资源规格

CPU 及 Memory 合计数值	Pod 资源选择规则
合计数值均为0	选择规格为1核2GiB。
任一合计数值为0	按非0项的合计数值进行最小匹配。例如，CPU 合计数值为0核，Memory 合计数值为8GiB，则在 Memory 为8GiB的允许规格中进行 CPU 最小匹配，最终选择规格为1核8GiB。
合计数值均不为0	与 资源规格 进行匹配。首先选择与 CPU 合计数值一致或相近的较大规格（A 规格），然后再选择与 Memory 的相近较大规格： • 如 Memory 合计数值 < A 规格的 Memory 区间最小值，则选择 A 规格的 Memory 区间的最小值。 • 如 Memory 合计数值 > A 规格的 Memory 区间最大值，则选择与 Memory 相近的较大规格（B 规格），并将 CPU 合计数改为 B 规格 CPU。 • 如 Memory 合计数值在 A 规格 Memory 区间之内，则选择最相近较大双数值。
任一合计数值超过允许的最大规格	出现错误，无法进行匹配。

示例

请结合以下示例，进一步了解 CPU Pod 规格计算方法：

示例1

```
resources:
```

```
    limits:
      cpu: "1"
      memory: 2Gi
    requests:
      cpu: "1"
      memory: 2Gi
```

结果：选择 Pod 规格为1核2GiB。

示例2

```
## container1
resources:
    limits:
      cpu: "4"
      memory: 4Gi
    requests:
      cpu: "2"
      memory: 4Gi
## container2
resources:
    limits:
      cpu: "1"
      memory: 2Gi
    requests:
      cpu: "1"
      memory: 2Gi
```

说明：
CPU 合计数值为：max((2+1),max(4,1)) = 4核
Memory 合计数值为：max((4+2),max(4,2)) = 6GiB
结果：
TKE Serverless 暂不支持4核6GiB的 Pod 规格，且6GiB小于 CPU 4核对应规格中 Memory 区间的最小值，则调整 Memory 为4核对应规格中 Memory 区间的最小值。最终选择 Pod 规格为4核8GiB。

GPU Pod 规格计算方法

说明：
- GPU 和 vGPU 的 Request 及 Limit 参数 `“nvidia.com/gpu”` 通常相等且仅支持整数。
- vGPU 可以看作是一种独立的 GPU 类型。例如 1/4*V100，是将一张 V100 GPU 卡 1/4 的算力虚拟为一张完整的卡进行分配，故在请求资源分配时依然应该是申请1卡 GPU，即 `“nvidia.com/GPU”=1` 。

步骤1：计算 Pod 的 GPU 的合计数值

GPU 合计数值为 **Pod 内所有容器的 Request 之和**。

步骤2：根据以下情况匹配 Pod 资源规格

CPU、Memory 及 GPU 合计数值	Pod 资源匹配规则
符合规格要求（例如1、2、4、8等）	首先选择与 GPU 合计数值一致或最相近的较大规格（A 规格），再按照 CPU Pod 规格计算方法 对 CPU 及 Memory 进行计算，得出 CPU 规格（B 规格）： • 如 A 规格的 CPU 及 Memory ≥ B 规格，则选择 A 规格 GPU。 • 如 A 规格的 CPU 及 Memory < B 规格，则选择与 B 规格的 CPU 及 Memory 最相近且较大的 GPU 规格（C 规格）。此时实际分配的 GPU 卡数会比实际所需更多，为了防止浪费，**尽量避免出现此情况**，请尽量调小 CPU 及 Memory 的请求数值。

任一合计数值如果超过允许的最大规格	出现错误，无法进行匹配。

示例

请结合以下示例，进一步了解 GPU Pod 规格计算方法：

示例1

```
## eks.tke.cloud.tencent.com/gpu-type: V100
resources:
    limits:
      cpu: "8"
      memory: 32Gi
      nvidia.com/gpu: "1"
    requests:
      cpu: "4"
      memory: 16Gi
      nvidia.com/gpu: "1"
```

说明：
GPU 合计数值为：1
CPU 合计数值为：max(4,8) = 8核
Memory 合计数值为：max(16,32) = 32GiB
结果：
8核32GiB小于 资源规格 中 V100 GPU 规格（1卡）对应的 CPU 及 Memory 规格（8核40GiB）。最终选择 Pod 规格为8核40GiB 1*V100。

示例2

```
## eks.tke.cloud.tencent.com/gpu-type: V100
## container1
resources:
    limits:
      cpu: "8"
      memory: 32Gi
      nvidia.com/gpu: "1"
    requests:
      cpu: "4"
      memory: 16Gi
      nvidia.com/gpu: "1"
## container2
resources:
    limits:
      cpu: "32"
      memory: 128Gi
      nvidia.com/gpu: "1"
    requests:
      cpu: "16"
      memory: 64Gi
      nvidia.com/gpu: "1"
```

说明：
GPU 合计数值为：1+1 = 2
CPU 合计数值为：max((4+16),max(8,32)) = 32核
Memory 合计数值为：max((16+64),max(32,128)) = 128GiB
结果：

32核128GiB大于 V100 GPU 规格（2卡）对应的 CPU 及 Memory 规格（18核80GiB），但小于 V100 GPU 规格（4卡）对应的 CPU 及 Memory 规格（36核160GiB）。
最终选择 Pod 规格为36核160GiB 4*V100，造成了2张 GPU 卡的资源浪费，请尽量避免出现这种情况。

超级节点上支持运行 DaemonSet

最近更新时间：2025-07-01 14:42:22

功能概述

由于超级节点没有实节点概念，因此在普通节点上运行的资源对象 DaemonSet 无法按照预期的方式运行，DaemonSet 所支持的某些系统层面的应用能力，如日志收集、资源监控等服务均无法在超级节点上统一支持。业内常规的解决方案为通过 sidecar 注入的方式实现 DaemonSet 相关的能力，但这带来了跟常规节点不一致的使用体验，且在功能上也是有损的，例如 sidecar 的更新将影响业务 Pod 的生命周期等，基于此，腾讯云全新推出了 DaemonSet Pod 注入方案用于在超级节点上运行 DaemonSet。作为业内唯一的在 Nodeless 架构中支持了 DaemonSet Pod 的运行，该方案拥有如下优势：
全兼容：完全兼容原生 DaemonSet Pod 的使用方式。
零侵入：DaemonSet Pod 与业务 Pod 拥有独立的生命周期，DaemonSet 的任何变更不会影响业务 Pod。
可观测：控制台支持对 DaemonSet Pod 的监控，支持查询 DaemonSet Pod 日志、事件等。

使用场景

- TKE 托管集群：在 TKE 托管集群中，如果您添加了超级节点，且超级节点上运行的 Pod 期望支持与普通节点一致的 DaemonSet 能力。
- TKE Serverless 集群：在 TKE Serverless 集群中，若期望运行 DaemonSet 资源。
- IDC 集群：在 IDC 集群中，如果您添加了超级节点，且超级节点上运行的 Pod 期望支持与 IDC 节点一致的 DaemonSet 能力。

前置准备

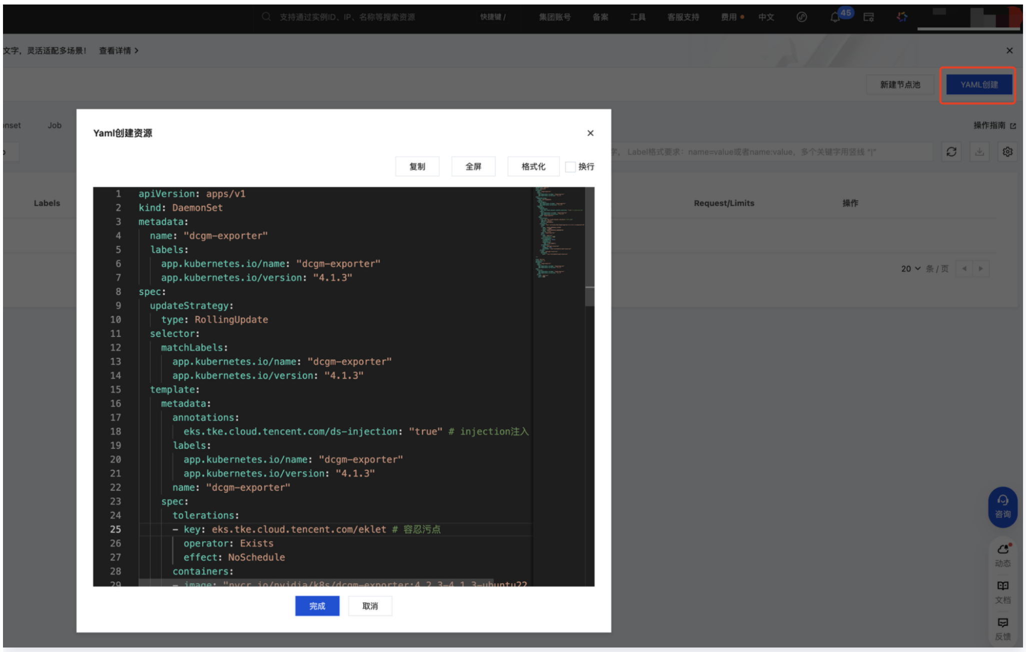
- 检查控制面 master 组件是否已升级到如下指定版本或更高版本：

Kubernetes 版本	控制面 master 组件版本要求	超级节点组件版本要求
v1.32	v1.32.0-tke.1 或更高版本	v2.16.10或更高版本
v1.30	v1.30.0-tke.1 或更高版本	v2.16.10或更高版本
v1.28	v1.28.3-tke.1或更高版本	v2.16.10或更高版本
v1.26	v1.26.1-tke.1 或更高版本	v2.16.10或更高版本
v1.24	v1.24.4-tke.4 或更高版本	v2.16.10或更高版本
v1.22	v1.22.5-tke.8 或更高版本	v2.16.10或更高版本
v1.20	v1.20.6-tke.30 或更高版本	v2.16.10或更高版本
v1.18	v1.18.4-tke.34 或更高版本	v2.16.10或更高版本
v1.16	v1.16.3-tke.33 或更高版本	v2.16.10或更高版本

- 检查超级节点的版本是否已升级到 v2.11.20 或以上。

使用方法

1. 创建 DaemonSet 并标记需要运行在超级节点的 DaemonSet



在 TKE 标准集群中，通过以下步骤配置即可在超级节点上运行 DaemonSet 负载，并且不会影响普通节点和原生节点上的 DaemonSet 负载正常运行。如下面文件所示，标记将要注入的 DaemonSet 为：`eks.tke.cloud.tencent.com/ds-injection: "true"`。对于按量计费超级节点，由于默认打了污点（key=eks.tke.cloud.tencent.com/eklet，effect=NoSchedule），因此需要容忍污点，才能支持将 DaemonSet Pod 创建上去。

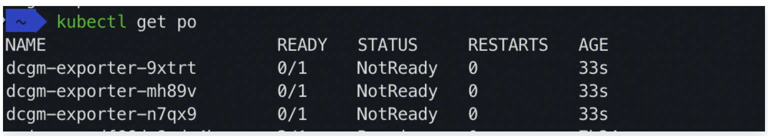
> **注意：**
> 调度时，DaemonSet 中定义的 resources 不会有效，并且也不会产生额外计费。

```
spec:
  template:
    metadata:
      annotations:
        eks.tke.cloud.tencent.com/ds-injection: "true"
    spec:
      tolerations:
      - key: eks.tke.cloud.tencent.com/eklet
        operator: Exists
        effect: NoSchedule
```

通过以上步骤，如果集群内存在超级节点，则可以通过运行 `kubectl get ds` 命令查看新的副本扩出情况，集群内有多少个超级节点则会展示多少个副本，如下图所示：

2. 启动 DaemonSet Pod

完成 DaemonSet 在超级节点的运行声明后，意味着当前超级节点上的 Pod 均会自动注入一个 DaemonSet Pod，后续启动的业务 Pod 也会被自动注入（kube-system 命名空间下的 Pod 除外），Pod 状态为 Injection 状态。



被注入的 Pod 本身的 YAML 配置不会发生任何变化，但 Container Status 里会新增注入的 DaemonSet Pod 容器的状态，以便更方便地观察 DaemonSet Pod 的状态，如下图所示：

```
containerStatuses:
- containerID: containerd://97923ccfd2f277128fe3b48ee9cd22a312950bdaf20d0609b5010e20e41de8f3
  image: nginx:1.20
  imageID: docker.io/library/nginx@sha256:38f8c1d9613f3f42e7969c3b1dd5c3277e635d4576713e6453c6193e66270a6d
  lastState: {}
  name: nginx
  ready: true
  restartCount: 0
  started: true
  state:
    running:
      startedAt: "2025-04-10T03:07:35Z"
- containerID: containerd://f38fe681ec3c4d33a85da444dd0ea24bd02456cf71c85672ef268b13eb1ed368
  image: docker.io/[redacted]/dcgm-exporter:4.2.0-4.1.0-ubuntu22.04
  imageID: docker.io/670017569/dcgm-exporter@sha256:0d85137d47b0d72735f121116111b9402a587714ef6418256507949980ca5093
  lastState: {}
  name: '[dcgm-exporter-mh89v]dcgm-exporter'
  ready: true
  restartCount: 0
  started: true
  state:
    running:
      startedAt: "2025-04-10T10:31:22Z"
```

DaemonSet Pod 容器的事件，例如拉镜像、启动、异常退出等信息，都会上报到被注入的 Pod 上：

```
Normal   Pulling  6m38s                  eklet  Pulling image "docker.io/[redacted]/dcgm-exporter:4.2.0-4.1.0-ubuntu22.04"
Normal   Pulled   6m33s                  eklet  Successfully pulled image "docker.io/[redacted]/dcgm-exporter:4.2.0-4.1.0-ubuntu22.04" in 5.639683379s
Normal   Killing  4m8s (x2 over 7h21m)   eklet  Stopping container httpd
Normal   Killing  3m12s                  eklet  Stopping container dcgm-exporter
Normal   Created  2m52s (x2 over 6m33s)  eklet  Created container dcgm-exporter
Normal   Started  2m52s (x2 over 6m33s)  eklet  Started container dcgm-exporter
Normal   Pulled   2m52s                  eklet  Container image "docker.io/[redacted]/dcgm-exporter:4.2.0-4.1.0-ubuntu22.04" already present on machine
```

3. 登录节点

在节点列表中找到一个普通节点，在右侧操作中，选择**更多 > 登录**，即可进入终端进行下一步操作。

4. 登录 DaemonSet Pod

在普通节点终端中使用 `kubectl exec` 命令可以正常登录原本 Pod 的容器。若要登录 DaemonSet Pod 的容器，则需要将 exec 指向被注入的 Pod，然后通过 `-c` 参数，指定要登录的 DaemonSet Pod 容器。命令如下：

```
kubectl exec -it <pod-name> -c <daemonset-container-name> -- /bin/bash
```

示例如下：

```
kubectl exec -it nginx-gw-df66dc8cd-4bmzx -c "[dcgm-exporter-mh89v]dcgm-exporter" -- /bin/bash
root@eks:/# exit
exit
```

> **注意：**
> 如果 DaemonSet Pod 的容器名称与原本 Pod 的容器名称相同，exec 命令会优先指向原本 Pod，此时无法登录 DaemonSet Pod 的容器。但是，DaemonSet Pod 的容器依旧在正常运行，为了 DaemonSet Pod 的容器可观测，建议将 DaemonSet Pod 的容器名称与 Pod 名称区分开。

5. 查看 DaemonSet Pod 日志

与使用 `kubectl exec` 命令类似，您可以使用 `kubectl logs` 命令结合 `-c` 参数来查看 DaemonSet Pod 容器的日志。命令如下：

```
kubectl logs <pod-name> -c <daemonset-container-name>
```

示例如下：

```
./kubectl-1.16 logs nginx-gw-df66dc8cd-4bmzx -c "[dcgm-exporter-mh89v]dcgm-exporter"
2025/04/10 10:31:22 maxprocs: Leaving GOMAXPROCS=2: CPU quota undefined
2025/04/10 10:31:22 INFO Starting dcgm-exporter Version=4.2.0-4.1.0
2025/04/10 10:31:22 INFO Attempting to initialize DCGM.
2025/04/10 10:31:22 INFO Initialized DCGM Fields module.
2025/04/10 10:31:22 INFO DCGM successfully initialized!
2025/04/10 10:31:22 INFO Attempting to initialize NVML library.
2025/04/10 10:31:22 ERROR Cannot init NVML library; err: ERROR_LIBRARY_NOT_FOUND
2025/04/10 10:31:22 INFO NVML provider successfully initialized!
2025/04/10 10:31:22 INFO Not collecting DCP metrics: This request is serviced by a module of DCGM that is not currently loaded
2025/04/10 10:31:22 INFO Falling back to metric file '/etc/dcgm-exporter/default-counters.csv'
2025/04/10 10:31:22 WARN Skipping line 20 ('DCGM_FI_PROF_GR_ENGINE_ACTIVE'): metric not enabled
2025/04/10 10:31:22 WARN Skipping line 21 ('DCGM_FI_PROF_PIPE_TENSOR_ACTIVE'): metric not enabled
2025/04/10 10:31:22 WARN Skipping line 22 ('DCGM_FI_PROF_DRAM_ACTIVE'): metric not enabled
2025/04/10 10:31:22 WARN Skipping line 23 ('DCGM_FI_PROF_PCIE_TX_BYTES'): metric not enabled
2025/04/10 10:31:22 WARN Skipping line 24 ('DCGM_FI_PROF_PCIE_RX_BYTES'): metric not enabled
2025/04/10 10:31:22 INFO Initializing system entities of type 'GPU'
2025/04/10 10:31:22 INFO Not collecting GPU metrics; error getting devices count: Cannot perform the requested operation because NVML doesn't exist on this system.
2025/04/10 10:31:22 INFO Initializing system entities of type 'NvSwitch'
2025/04/10 10:31:22 INFO Not collecting NvSwitch metrics; no switches to monitor
2025/04/10 10:31:22 INFO Initializing system entities of type 'NvLink'
2025/04/10 10:31:22 INFO Not collecting NvLink metrics; no switches to monitor
2025/04/10 10:31:22 INFO Initializing system entities of type 'CPU'
2025/04/10 10:31:22 INFO Not collecting CPU metrics; error retrieving DCGM CPU hierarchy: This request is serviced by a module of DCGM that is not currently loaded
2025/04/10 10:31:22 INFO Not collecting CPU Core metrics; error retrieving DCGM CPU hierarchy: This request is serviced by a module of DCGM that is not currently loaded
2025/04/10 10:31:22 INFO Kubernetes metrics collection enabled!
2025/04/10 10:31:22 INFO Starting webserver
2025/04/10 10:31:22 INFO Listening on address=[::]:9400
2025/04/10 10:31:22 INFO TLS is disabled. http2=false address=[::]:9400
```

请注意，由于 kubectl 1.18及以上版本的客户端，会在发起 logs 请求之前校验 Pod Spec 是否包含指定容器。如果不存在指定容器，则直接返回如下错误：

```
container xxx is not valid for pod xxx
```

出现此类情况时，您可以将 kubectl 版本切换到1.16版本，或者使用我们后续提供的发行版。以下是 kubectl 1.16 版本的下载链接：

```
curl -LO "https://dl.k8s.io/release/v1.16.0/bin/linux/amd64/kubectl"
```

6. 滚动更新

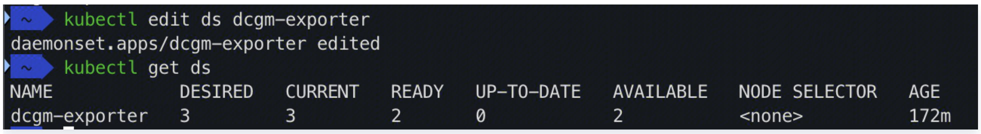
集群维度

在 TKE 标准集群维度，超级节点支持了 Kubernetes 原生能力一致的滚动更新能力，可以通过 updateStrategy 来设置 DaemonSet 的滚动更新规则。

```
spec:
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1  # 更新期间不可用 Pod 的最大数量，默认为1
```

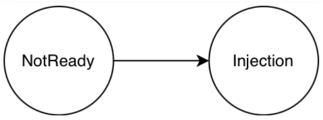
在 DaemonSet 配置发生更新后，超级节点会与普通节点或原生节点一样按照滚动更新策略进行更新。
修改 DaemonSet 配置后，通过 `kubectl get ds` 查看滚动更新状态。

```
kubectl get ds
```

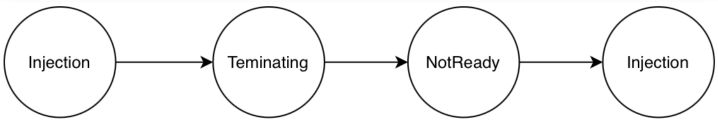


```
~ kubectl edit ds dcgm-exporter
daemonset.apps/dcgm-exporter edited
~ kubectl get ds
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
dcgm-exporter   3         3         2       0            2           <none>          172m
```

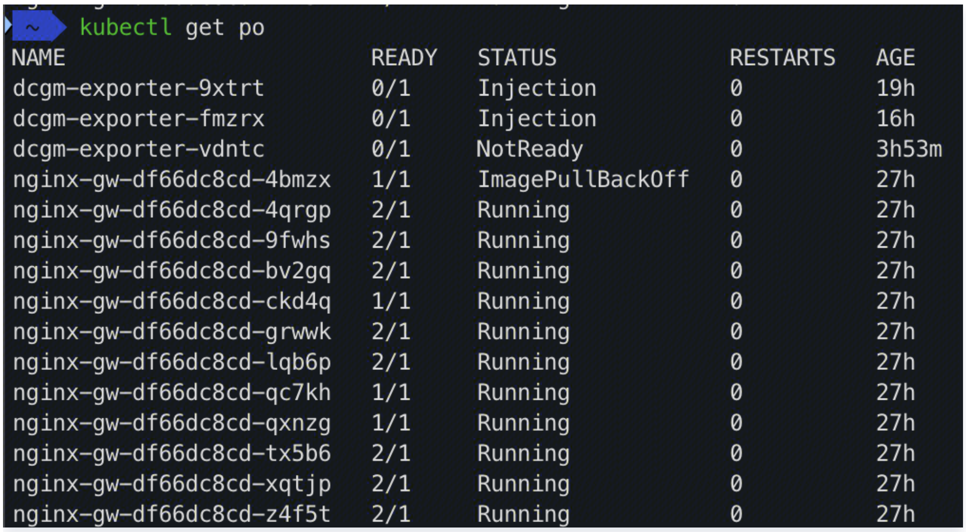
新建 DaemonSet 时，状态过程转移如下：



更新 DaemonSet 时，状态转移过程如下：



当发布更新出现问题时，会停止滚动：



```
~ kubectl get po
NAME                        READY   STATUS             RESTARTS   AGE
dcgm-exporter-9xtrt         0/1     Injection          0          19h
dcgm-exporter-fmzrx         0/1     Injection          0          16h
dcgm-exporter-vdntc         0/1     NotReady           0          3h53m
nginx-gw-df66dc8cd-4bmzx    1/1     ImagePullBackOff   0          27h
nginx-gw-df66dc8cd-4qrgp    2/1     Running            0          27h
nginx-gw-df66dc8cd-9fwhs    2/1     Running            0          27h
nginx-gw-df66dc8cd-bv2gq    2/1     Running            0          27h
nginx-gw-df66dc8cd-ckd4q    1/1     Running            0          27h
nginx-gw-df66dc8cd-grwwk    2/1     Running            0          27h
nginx-gw-df66dc8cd-lqb6p    2/1     Running            0          27h
nginx-gw-df66dc8cd-qc7kh    1/1     Running            0          27h
nginx-gw-df66dc8cd-qxnzg    1/1     Running            0          27h
nginx-gw-df66dc8cd-tx5b6    2/1     Running            0          27h
nginx-gw-df66dc8cd-xqtjp    2/1     Running            0          27h
nginx-gw-df66dc8cd-z4f5t    2/1     Running            0          27h
```

节点维度

在超级节点维度，某超级节点进入更新状态后，当且仅当该超级节点上的业务 Pod 都完成了 DaemonSet 的注入容器更新时，该超级节点所对应的虚拟 DaemonSet 状态才为 Ready，至此该节点的更新完成。

通过注解 `eks.tke.cloud.tencent.com/ds-inject-rate` 即可控制在超级节点内部的注入容器更新下发速率。

```
eks.tke.cloud.tencent.com/ds-inject-rate: "50%"
```

通过 `kubectl get po xxx -o yaml` 查看正在更新的超级节点所对应的更新状态信息聚合。

```
~ kubectl get po
NAME                       READY   STATUS      RESTARTS   AGE
dcgm-exporter-9xtrt        0/1     Injection   0          176m
dcgm-exporter-fmzrx        0/1     NotReady    0          60s
dcgm-exporter-mh89v        0/1     Injection   0          176m
nginx-gw-df66dc8cd-4bmzx   2/1     Running     0          10h
nginx-gw-df66dc8cd-4qrgp   2/1     Running     0          10h
nginx-gw-df66dc8cd-9fwhs   2/1     Running     0          10h
nginx-gw-df66dc8cd-bv2gq   2/1     Running     0          10h
nginx-gw-df66dc8cd-ckd4q   2/1     Running     0          10h
nginx-gw-df66dc8cd-grwwk   2/1     Running     0          10h
nginx-gw-df66dc8cd-lqb6p   1/1     Running     0          10h
nginx-gw-df66dc8cd-qc7kh   2/1     Running     0          10h
nginx-gw-df66dc8cd-qxnzg   2/1     Running     0          10h
nginx-gw-df66dc8cd-tx5b6   2/1     Running     0          10h
nginx-gw-df66dc8cd-xqtjp   2/1     Running     0          10h
nginx-gw-df66dc8cd-z4f5t   2/1     Running     0          10h
```

```
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2025-04-10T13:26:32Z"
    status: "True"
    type: PodScheduled
  - lastProbeTime: null
    lastTransitionTime: "2025-04-10T13:26:32Z"
    status: "False"
    type: Ready
  message: |
    desired : 4
    ready : 1
    notReadyPods[1/1]:
      default/nginx-gw-df66dc8cd-bv2gq[preupdate] : init
  phase: Pending
  qosClass: BestEffort
  reason: NotReady
```

暂停发布：

```
spec:
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 0   # 将更新期间不可用副本的最大数量设置为0即可暂停发布
```

发布回滚：

```
kubectl rollout history daemonset <ds_name>
```

```
✘ ~ kubectl rollout history daemonset dcgm-exporter
daemonset.apps/dcgm-exporter
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

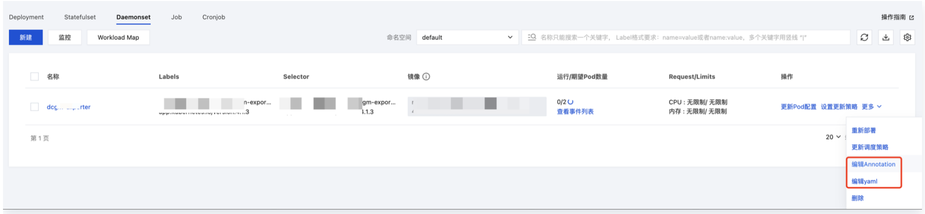
```
kubectl rollout history daemonset <ds_name>
```

```
~ kubectl rollout undo daemonset dcgm-exporter --to-revision=2
daemonset.apps/dcgm-exporter rolled back
```

7. 注入规则

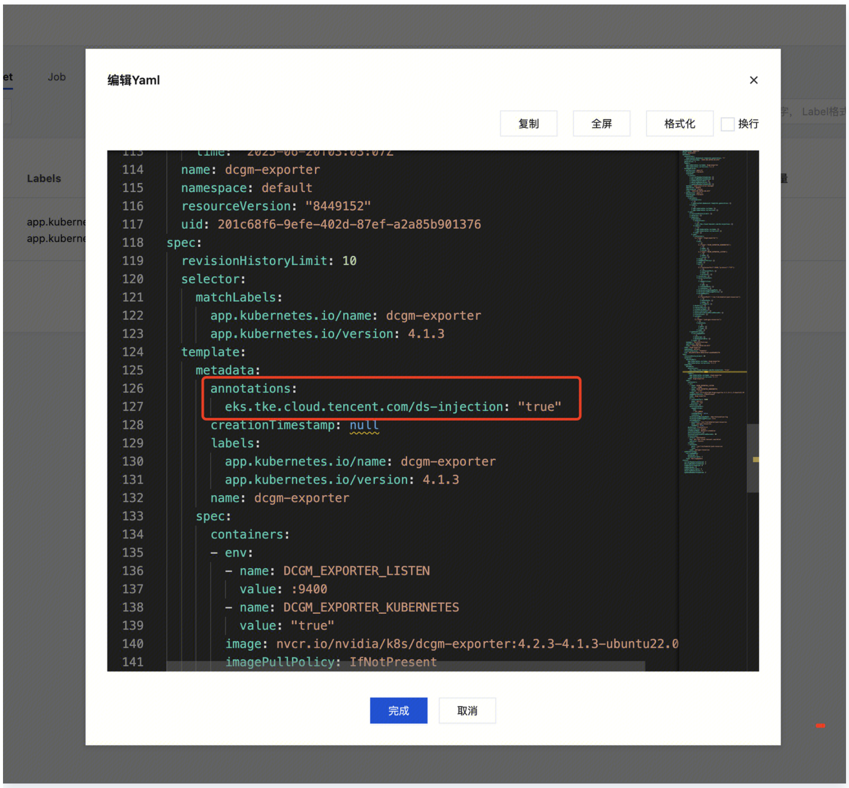
注入方式

首先找到需要编辑的 DaemonSet 工作负载。



在 DaemonSet 的工作负载上点击**编辑 Annotation** 可以进行对应注解的编辑。

也可以点击编辑 YAML 进行修改。



示例

若超级节点上声明启用了 DaemonSet Pod，则除 kube-system 命名空间下的 Pod 外，该超级节点上的其他 Pod，均会被注入 DaemonSet Pod。若您有特殊 Pod 不希望被注入 DaemonSet Pod，可以配置如下 annotation：

```
eks.tke.cloud.tencent.com/ds-injection: "false"
```

如果您希望整个命名空间下的所有 Pod 都不被注入 DaemonSet Pod，可以在命名空间上配置如下 annotation：


```
eks.tke.cloud.tencent.com/ds-injection: "false"
```

如果您希望 kube-system 命名空间下的 Pod 被注入 DaemonSet Pod，则需要在创建这些 Pod 时显式声明如下 annotation：

```
eks.tke.cloud.tencent.com/ds-injection: "true"
```

如果您希望按 Label 过滤需要注入的 DaemonSet，可以在 DaemonSet 上配合使用如下 annotation：

- 当与 pod 的某个 Label 匹配时注入：

```
eks.tke.cloud.tencent.com/ds-inject-by-label: "k1:v1,k2:v2"
```

- 当与 pod 的某个 Label 匹配时不注入：

```
eks.tke.cloud.tencent.com/ds-evict-by-label: "k1:v1,k2:v2"
```

> **注意：**
> 当两个 annotation 同时存在时，ds-evict-by-label 的优先级更高。

8. 特殊能力

调整 DaemonSet Pod 的启动顺序

如果注入的 DaemonSet Pod 需要先于业务 Pod 启动，可以给业务 Pod 配上如下 annotation ，让业务 Pod 晚于 DaemonSet Pod 启动：

```
eks.tke.cloud.tencent.com/start-after-ds: "true"
```

开放端口

注入的 DaemonSet Pod 若需要对外暴露端口，则需额外通过 annotation 声明，声明方式如下：

```
eks.tke.cloud.tencent.com/metrics-port: "9100,8080,3000-5000"
```

默认只暴露9100端口，用来提供监控数据 metrics，您可以在9100之后添加其他端口，支持范围 range 写法，多个端口之间用逗号分隔，最多支持 4 个逗号。请注意，可以变更9100端口，但不要删除，第一个端口默认用于提供监控数据 metrics。
从容器外访问 DaemonSet Pod 的端口时，访问的 IP 为业务 Pod 的 IP，端口为 DaemonSet Pod 的端口：

```
curl "http://<pod-ip>:<ds-port>/"
```

业务 Pod 与本地 DaemonSet Pod 通信

如果业务 Pod 需要主动访问注入的 DaemonSet Pod，则需要获取到注入的 DaemonSet Pod 使用的虚拟 IP。通过添加如下 annotation，业务 Pod 可以通过 env from hostIP，获取到 DaemonSet Pod 所使用的虚拟 IP：

```
eks.tke.cloud.tencent.com/env-host-ip: "true"
```

容器里使用类似 env from hostIP：

```
env:
- name: HOST_IP
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP
```

DaemonSet Pod 访问 kubelet 的 10250 端口

如果 DaemonSet Pod 要通过 kubelet 的接口获取本地 pod 的列表，可以给业务 pod 配上如下 annotation：

```
eks.tke.cloud.tencent.com/read-only-port: "10250"
```

此时 DaemonSet Pod 就可以通过 127.0.0.1:10250 访问到 pod 列表。

采集超级节点上的 Pod 日志

最近更新时间：2024-04-22 09:59:41

本文主要介绍 TKE 集群中调度至超级节点的 Pod 如何采集日志，包括：

- 采集日志至 CLS
- 采集日志至 Kafka

采集日志至 CLS

服务角色授权

在采集超级节点上的 Pod 日志至 CLS 之前，需要进行服务角色授权，以保证将日志正常上传到 CLS。
操作步骤如下：

1. 登录**访问管理控制台** > 角色 。
2. 在角色页面单击**新建角色**。
3. 在选择角色载体页面，选择**腾讯云产品服务** > **容器服务(tke)** > **容器服务-EKS日志采集**，并单击**下一步**。如下图所示：

4. 确认角色策略 QcloudAccessForTKELinkedRoleInEKSLog，单击**下一步**。
5. 配置角色标签，单击**下一步**。
6. 审阅角色策略，单击**完成**，即可完成为该账号配置该角色。

配置日志采集

服务角色授权完成后，需要开启 TKE 日志采集功能，并配置相应的日志采集规则。例如，指定工作负载采集和指定 pod labels 采集。详情请参见 使用 YAML 配置日志采集 。

采集日志至 Kafka

若需要采集超级节点上的 Pod 的日志至自建 Kafka 或者 CKafka，您可以在控制台配置相应的日志采集规则，或者自行配置 CRD，定义采集源及消费端，CRD 配置完成后，Pod 自带的采集器会依照规则进行日志采集。
CRD 具体配置如下所示：

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig                          ## 默认值
metadata:
  name: test                                ## CRD资源名，在集群内唯一
spec:
  kafkaDetail:
    brokers: xxxxxx        # 必填，broker地址，一般是域名:端口，多个地址以“,”分隔
    topic: xxxxxx          # 必填，topicID
    messageKey:            # 选填，指定pod字段作为key上传到指定分区
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    timestampKey:              #时间戳的key，默认是@timestamp
    timestampFormat:         #时间戳的格式，默认是double
  inputDetail:
```

```
    type: container_stdout                        ## 采集日志的类型，包括container_stdout（容器标准输出）、container_file（容器文件）

    containerStdout:                              ## 容器标准输出
      namespace: default                          ## 采集容器的kubernetes命名空间，如果不指定，代表所有命名空间
      allContainers: false                        ## 是否采集指定命名空间中的所有容器的标准输出
      container: xxx                              ## 采集日志的容器名，此处可填空
      includeLabels:                              ## 采集包含指定label的Pod
        k8s-app: xxx                              ## 只采pod标签中配置"k8s-app=xxx"的pod产生的日志，与workloads、allContainers=true不能同时指定
      workloads:                                  ## 要采集的容器的Pod所属的kubernetes workload
      - namespace: prod                           ## workload的命名空间
        name: sample-app                          ## workload的名字
        kind: deployment                          ## workload类型，支持deployment、daemonset、statefulset、job、cronjob
        container: xxx                            ## 要采集的容器名，如果填空，代表workload Pod中的所有容器

    containerFile:                                ## 容器内文件
      namespace: default                          ## 采集容器的kubernetes命名空间，必须指定一个命名空间
      container: xxx                              ## 采集日志的容器名，此处可填*
      includeLabels:                              ## 采集包含指定label的Pod
        k8s-app: xxx                              ## 只采pod标签中配置"k8s-app=xxx"的pod产生的日志，与workload不能同时指定
      workload:                                   ## 要采集的容器的Pod所属的kubernetes workload
        name: sample-app                          ## workload的名字
        kind: deployment                          ## workload类型，支持deployment、daemonset、statefulset、job、cronjob
      logPath: /opt/logs                          ## 日志文件夹，不支持通配符
      filePattern: app_*.log                      ## 日志文件名，支持通配符 * 和 ? ，* 表示匹配多个任意字符，? 表示匹配单个任意字符
```

镜像缓存

最近更新时间：2025-07-01 17:57:02

镜像缓存概述

使用镜像缓存可以在创建实例时加速拉取镜像，减少实例的启动耗时。该能力适用于 TKE Serverless Pod 和超级节点。本文主要介绍镜像缓存的工作原理、计费说明、创建和使用方式等。

工作原理

镜像缓存加速启动实例时会事先启动一个容器实例进行镜像拉取，该镜像存储在一个可自定义大小的数据盘中，同时将该数据盘作为云盘快照缓存，即镜像数据已被保存在快照中。在创建容器实例或 Pod 时选择自动匹配或手动匹配镜像缓存，会基于镜像快照创建相应数量的硬盘（数据盘），并直接挂载到实例上，避免镜像层下载，从而提升容器实例及 Pod 的创建速度。

镜像缓存的优势如下：

- 无时效限制，与镜像缓存（快照）的生命周期有关。
- 只需提前冷启动一个 Pod，创建快照后该 Pod 即被销毁。
- 无可用区限制，基于快照创建云盘时自动匹配可用区。
- 无工作负载的限制，相同地域中即可匹配。

计费说明

在创建镜像缓存时，会涉及到以下资源，相应的计费情况如下：

计费项	计费说明	计费文档
镜像缓存	创建镜像缓存时，需要运行一个2核4GiB Serverless pod 以拉取镜像。在镜像缓存创建完成后，该容器实例会自动释放并停止计费。	Serverless pod 计费
CBS 数据盘	创建镜像缓存时，需要绑定一个高性能数据盘存储镜像，该数据盘的大小支持自定义，默认为20G。在镜像缓存创建完成后，该数据盘会自动释放并停止计费。	云硬盘计费
快照	基于上述数据盘会创建一个快照，该快照的生命周期与镜像缓存的生命周期一致。快照按照使用时长及容量收费。	快照计费

使用镜像缓存时，会基于匹配的镜像缓存快照创建一个相同容量的高性能数据盘，并绑定到 Pod 上，因此除去创建 Pod 本身的费用，会额外收取**数据盘的费用**。数据盘的费用计算公式如下：

数据盘费用 = 容量 × 单价 × 实例运行时长

操作步骤

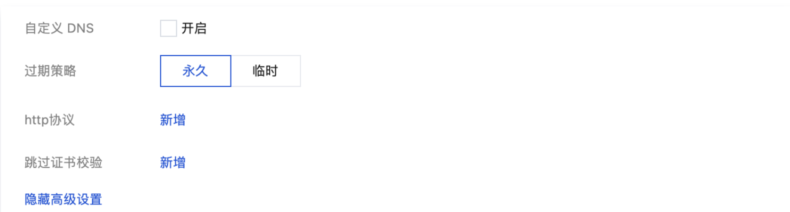
使用控制台创建镜像缓存

1. 登录 容器服务控制台 。
2. 选择左侧导航栏的**运维中心 > 镜像缓存**，在镜像缓存页面单击**新建实例**。
3. 在**创建镜像缓存**页面，配置相关参数。如下图所示：

- 实例名称：自定义。
- 所在地域：按需选择。
- 容器网络：为容器实例分配在容器网络地址范围内的 IP 地址。
- 安全组：安全组具有防火墙的功能，可限制容器实例的网络通信，默认为 default。
- 镜像：按需选择需要进行缓存的镜像及其版本。
- 镜像凭证：当选择 Dockerhub 及其他第三方镜像仓库的私有镜像时，必须填写镜像凭证，即仓库的名称、访问地址、用户名及密码。
- 缓存大小：该大小决定快照及之后创建实例时绑定的数据盘的默认大小。

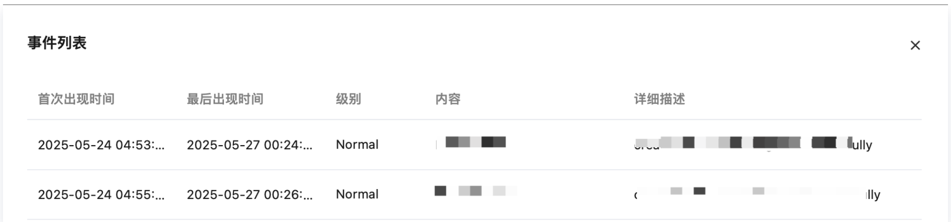
> **注意：**
> 镜像仓库中显示的镜像大小为压缩后的数据，制作镜像缓存有拉取镜像并展开的过程，若镜像过大或镜像压缩比过大将导致默认的20GB数据盘不够用，建议设置更大的数据盘空间。

- 绑定弹性公网 IP：若要访问外网，需要为该实例绑定一个弹性公网IP。
- 高级设置：



  - 自定义 DNS：开启后可自定义创建或更新镜像缓存时的 DNS server 配置。
  - 过期策略：选择镜像缓存的保留时间，默认为永久保留。
  - http协议：拉取指定镜像仓库的镜像时，使用 HTTP 协议。
  - 跳过证书校验：拉取指定镜像仓库的镜像时，不校验证书。

4. 单击**创建实例**。创建完成后可在镜像缓存的列表，单击操作列的**查看事件**查看创建进程。如下图所示：



使用 CRD 创建镜像缓存

若要使用 CRD 创建镜像缓存服务，需在集群内安装镜像缓存组件，安装后支持以 CRD+Controller 的模式，使用腾讯云镜像缓存服务，而不需要调用云 API 接口。操作步骤如下：

1. 登录 容器服务控制台 。
2. 在集群列表中单击 Serverless 集群 ID，进入集群详情页。
3. 在左侧导航栏中选择**组件管理**，单击**新建**。
4. 在**新建组件管理**页面选择 imc-operator（镜像缓存） 组件。如下图所示：

5. 单击**完成**。在组件管理列表页， 查看已安装的组件。
6. 编辑 YAML。

创建 ImageCache

示例：

```
apiVersion: eks.cloud.tencent.com/v1
kind: ImageCache
metadata:
  name: imagecache-sample
spec:
  images:
    - nginx
#  imageCacheSize: 30
  # TODO(user): Add fields here
```

带有更多参数的示例：

```
apiVersion: eks.cloud.tencent.com/v1
kind: ImageCache
metadata:
  name: imagecache-sample-more-para
spec:
  images:   # 用于制作镜像缓存的容器镜像列表
    - nginx
    - mysql
  imageCacheSize: 30 # 镜像缓存的大小
  retentionDays: 7 # 镜像缓存保留时间天数，过期将会自动清理，默认为0，永不过期
```

```
  imagePullSecrets: # 存放镜像仓库凭证的 Secret
    - imc-operator-system/qcloudregistrykey
  securityGroupIds: # 指定安全组
    - sg-maai9m2t
  subnetId: "subnet-rgkt80q9" # 指定制作镜像缓存时启动的容器实例所属子网 ID，未指定时，自动选择某个超级节点的子网
  # ExistedEipId: "eni-1cghhuuu" # 指定用来绑定容器实例的已有EIP
  autoCreateEip: true  # 是否为容器实例自动创建EIP，默认为false。若传true，则此参数和ExistedEipIds互斥
  autoCreateEipAttribute: # 自动创建EIP的可选参数。若传此参数，则会自动创建EIP
    deletePolicy: Never # 容器实例删除后，EIP是否释放
    InternetServiceProvider: BGP # EIP线路类型
    internetMaxBandwidthOut: 2 # EIP出带宽上限，单位：Mbps
  registrySkipVerifyList: # 拉取指定镜像仓库的镜像时，不校验证书
    - "harbor.example.com"
  registryHttpEndPointList: # 拉取指定镜像仓库的镜像时，使用 HTTP 协议
    - "harbor.example.com"
  resolveConfig: "nameserver 8.8.8.8" # 指定自定义制作镜像缓存过程中容器实例的宿主机上的 DNS自动创建EIP的可选参
数。若传此参数，则会自动创建EIP自动创建EIP的可选参数。若传此参数，则会自动创建EIP
```

上述示例中添加了参数说明，具体可参考 创建镜像缓存输入参数说明 。

查看 ImageCache

示例：

```
kubectl get imc
```

如有异常可查看 events：

```
kubectl describe imc xxx
```

使用已创建的镜像缓存

TKE 集群内的超级节点上的 Pod 可通过指定 Pod 的 Annotation 来使用镜像缓存，具体可参考超级节点 Annotation 说明 。

- 自动匹配：

```
eks.tke.cloud.tencent.com/use-image-cache: auto
```

- 手动指定：

```
eks.tke.cloud.tencent.com/use-image-cache: imc-xxx
```

Serverless 集群内创建 Pod 时，可在**高级配置**中，开启使用镜像缓存功能。镜像缓存支持**自动匹配**和**手动匹配**两种匹配方式，您可以根据自身需求选择合适的匹配方式。

自动匹配

在缓存匹配模式中选择自动匹配时，根据以下匹配策略，将自动匹配最优的镜像缓存。
- 镜像名称及版本完全相同，则可匹配。
- 镜像缓存大小，小容量优先匹配。
- 创建时间，创建时间晚的优先匹配。

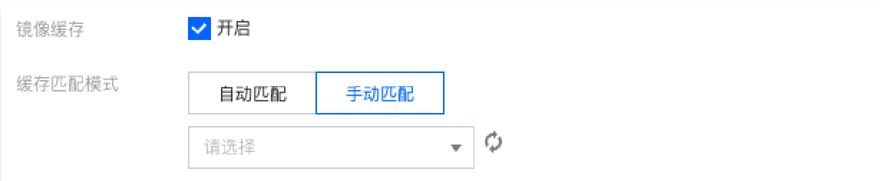
注意：

若双方都为 nginx: latest，依旧可匹配，但是由于创建时间不同，可能存在版本不一致的情况，因此，建议创建时镜像缓存及实例时，明确注明版本。
自动匹配时，无法匹配 TKE Serverless daemonset 的镜像，因为镜像缓存当前无法感知需要注入哪些 daemonset。

若没有匹配到对应的镜像缓存，则会自动正常拉取镜像。

手动匹配

在缓存匹配模式中选择**手动匹配**时，需要手动选择具体的镜像缓存。需注意，手动指定镜像缓存后，会直接按照该镜像缓存快照创建数据盘并绑定到实例上，但是若数据盘中并没有创建时填写的镜像（即手动指定了错误的镜像缓存），会重新在新创建的数据盘中拉取镜像。



匹配结果

可从创建实例的事件中，查看是否匹配成功。
若匹配成功，则显示以下事件：

内容	详细描述
Started	Started container container
Created	Created container container
Pulled	Container image "nginx" already present on machine
ImageCacheUsed	Image cache imc-7fwlzosl used. Disk disk-dz7qm9i8 attached

若未显示该事件，则表示没有匹配到合适的镜像缓存。
需要注意的是，若选择手动匹配，但是匹配的镜像缓存中并没有该镜像，会重新在新创建的数据盘中拉取镜像，并显示以下事件：

内容	详细描述
Started	Started container container
Created	Created container container
Pulled	Successfully pulled image "nginx" in 5.530971711s
Pulling	Pulling image "nginx"
ImageCacheUsed	Image cache imc-kcbee4nj used. Disk disk-d81vy7vw attached

超级节点常见问题

最近更新时间：2024-11-29 17:09:52

注册节点管理
注册节点概述

最近更新时间：2023-11-10 17:39:22

什么是注册节点?

注册节点（原第三方节点）是腾讯云容器服务团队针对混合云部署场景，全新升级的节点产品形态，允许用户将非腾讯云的主机，托管到容器服务 TKE 集群，由用户提供计算资源，容器服务 TKE 负责集群生命周期管理。

> **说明：**
> 注册节点现在支持专线版（通过专线 + 云联网连接）和公网版（通过 Internet 连接）两种产品模式，用户可以根据不同的场景按需选择使用。

> **注意：**
> TKE 新上线注册节点公网版产品，当前限制**2023年11月9日 12:00:00**之后的新建集群可以开启此能力，具体约束条件请参见 公网版约束条件 。
> 现有存量集群待后期产品规划，会逐步开放此功能。

使用场景

资源利旧

企业需要迁移上云，但在本地数据中心已经进行了投资，在 IDC 有存量的服务器资源（CPU 资源、GPU 资源）。可以通过注册节点的特性，将 IDC 主机资源添加到 TKE 公有云集群，确保在上云过程中存量服务器资源得到有效利用。

集群托管运维

免去在本地搭建、运维 K8s 集群的成本，由腾讯云统一运维管控，用户仅需要运维本地服务器即可。

混合部署调度

支持在单集群内同时调度注册节点与云上 CVM 节点，便于将云下业务拓展至云上，无需引入多集群管理。

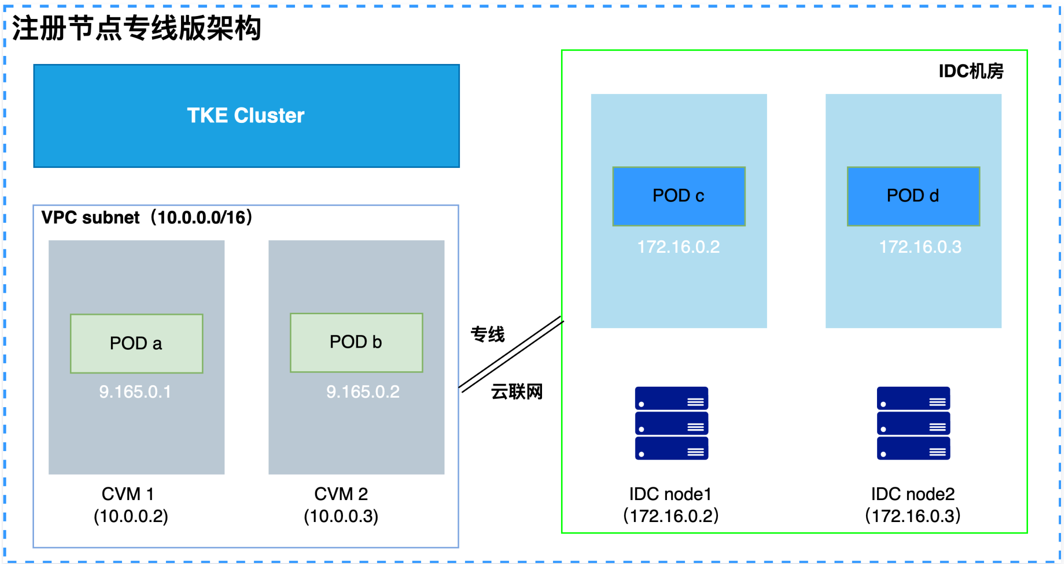
无缝集成云端服务

注册节点无缝集成腾讯云云原生相关服务，涵盖日志、监控、审计、存储、容器安全等云原生能力。

注册节点专线版

专线版产品架构

用户可以将自身 IDC 环境通过**专线 + 云联网**的方式和腾讯云 VPC 之间打通，然后通过内网将 IDC 节点接入到 TKE 集群，实现 IDC 节点和云上 CVM 的统一纳管。其架构图如下：



专线版约束条件

使用注册节点能力前，需要保证环境是满足约束要求的，否则无法正常使用注册节点产品功能。

- 操作系统约束：注册节点的操作系统必须使用 TencentOS Server 3.1 和 TencentOS Server 2.4(TK4)。
- 硬件约束（GPU）：仅支持 NVIDIA 系列显卡，包括：Volta（如 V100）、Turing（如 T4）、Ampere（如 A100、A10）。
- 云上云下连通：云联网+专线打通（暂不支持 VPN 方式）。
- TKE集群约束：**1.18及以上**版本，云上必须有至少一个 CVM 节点（暂不支持集群仅添加注册节点）。

专线版节点端口放通

为了保障混合云集群云上云下互通，需要在云上和 IDC 节点分别放通一系列端口。

- **云上节点**：云上节点使用满足 TKE 要求的安全组 容器服务安全组设置。如果 TKE 集群使用 **cilium overlay** 网络模式，则节点还需额外放通。
  - **入站规则**

协议	端口	来源	策略	备注
UDP	8472	集群网络 CIDR IDC 网络 CIDR	允许	放通集群节点 vxlan 通信

  - **出站规则**

协议	端口	目的	策略	备注
UDP	8472	集群网络 CIDR IDC 网络 CIDR	允许	放通集群节点 vxlan 通信

- **IDC 节点**：在节点上设置防火墙规则放通端口。
  - **入站规则**

协议	端口	来源	策略	备注
UDP	8472	集群网络 CIDR IDC 网络 CIDR	允许	放通集群节点 vxlan 通信
TCP	10250	集群网络 CIDR IDC 网络 CIDR	允许	放通 API Server 通信

  - **出站规则**

协议	端口	目的	策略	备注
UDP	8472	集群网络 CIDR IDC 网络 CIDR	允许	放通集群节点 vxlan 通信
TCP	80、443、9243、10250、60002	代理子网 SUBNET	允许	放通云上代理通信

网络模式

对于 TKE 集群不同网络类型，注册节点上 Pod 网络能力也存在一定的限制，说明如下：

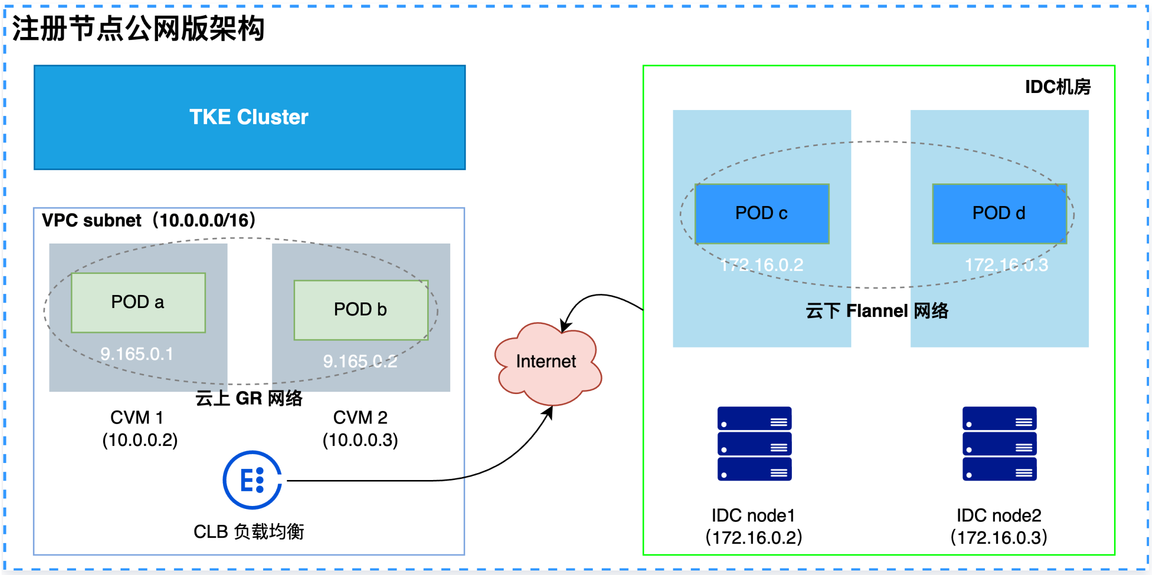
- GlobalRouter 和 VPC-CNI 独占网卡模式集群：云下 IDC 节点上 Pod 仅支持使用 hostNetwork 网络模式，此模式下云下 pod 只能通过 host 网络和云上互通。
- Cilium-Overlay 网络模式集群：TKE 团队特别为混合云推出的容器网络方案，云上云下 Pod 处于同一 overlay 网络平面，可以实现云上云下 pod 互通。

网络说明信息详见 网络模式。

注册节点公网版

公网版产品架构

如果用户因为某些客观因素，无法打通 IDC 和腾讯云之间的专线，同时用户又期望通过 TKE 管理 IDC 内节点，降低搭建和运维 K8s 的成本，可以使用注册节点公网版产品，通过 **Internet** 将 IDC 节点注册到 TKE 进行统一管理。其架构图如下：



注意：
和专线版不同，公网版由于只能通过 Internet 和 TKE 互通，默认情况下，云上 CVM 和云下 IDC 是两个完全隔离的分区，暂时无法支持云上节点和云下节点的 Pod 网络互通。因此，该产品模式建议用户将 IDC 机房节点作为一个独立的节点池进行管理以及业务调度，避免云上云下 Pod 互访（可以通过 Ingress 等其他方式实现云上云下业务互访）。

公网版约束条件

使用注册节点公网版前，需要保证环境是满足约束要求的，否则无法正常使用注册节点产品功能。
- 操作系统约束：注册节点的操作系统必须使用 TencentOS Server 3.1 和 TencentOS Server 2.4(TK4)。
- TKE集群约束：
  - K8s **1.20及以上版本**
  - “容器网络插件”需要选择 **Global Router**
  - 云上必须有至少一个 CVM 节点（暂不支持集群仅添加注册节点）。
- 网络约束：IDC 节点可以和腾讯云 CLB 互通，能够访问 CLB 的 TCP 443 和 9000 端口。
- 硬件约束（GPU）：当前暂时不支持 GPU 节点。

注意：
当前公网版只开放给创建时间在**2023年11月9日 12:00:00**之后的集群使用。现有存量集群待后续产品规划，会逐渐放开此能力限制。

公网版节点端口放通

为了保障 IDC 环境和 TKE 集群互通，需要在云上和 IDC 节点分别放通一系列端口。
- **云上节点**：云上节点使用满足 TKE 要求的安全组 容器服务安全组设置。
- **IDC 节点**：在节点上设置防火墙规则放通端口，同时设置规则允许访问公网镜像仓库。
  - **镜像仓库：保证在 IDC 节点可以访问 ccr.ccs.tencentyun.com 以及 superedge.tencentcloudcr.com**
  - **入站规则**

协议	端口	来源	策略	备注
UDP	8472	集群网络 CIDR IDC网络 CIDR	允许	放通集群节点 vxlan 通信
TCP	10250	集群网络 CIDR IDC网络 CIDR	允许	放通 API Server 通信

  - **出站规则**

协议	端口	目的	策略	备注

UDP	8472	集群网络 CIDR IDC网络 CIDR	允许	放通集群节点 vxlan 通信
TCP	443、9000	公网版对应的 CLB 地址	允许	放通访问云上 CLB 地址，提供节点注册和云边隧道服务

网络模式

对于注册节点公网版，云上和云下的网络天然隔离，因此云上节点会使用 GR 网络在云上节点之间实现 Pod 互通，云下节点会使用 Flannel网络在 IDC 节点之间实现 Pod 互通，而云上 Pod 和云下 Pod 之间默认隔离。

注册节点与云上节点能力对比

分类	能力	云上节点	注册节点（专线版）	注册节点（公网版）
节点管理	添加节点	✔	✔	✔
	移除节点	✔	✔	✔
	设置节点标签、污点	✔	✔	✔
	节点封锁、驱逐	✔	✔	✔
	通过节点池批量管理	✔	✔	✔
	Kubernetes 版本升级	✔	暂无	暂无
存储卷	本地存储（emptyDir、hostPath 等）	✔	✔	✔
	Kubernetes API（ConfigMap、Secret…）	✔	✔	✔
	腾讯云云硬盘 CBS	✔	无	无
	腾讯云文件存储 CFS	✔	✔	无
	腾讯云对象存储 COS	✔	✔	无
可观测性	支持 Prometheus 监控服务	✔	✔	暂无
	支持云产品监控	✔	无	暂无
	支持日志对接 CLS	✔	✔	暂无
	支持集群审计	✔	✔	✔
	支持事件存储	✔	✔	✔
流量接入	支持 ClusterIP 类型 Service	✔	✔	✔
	支持 NodePort 类型 Service	✔	✔	✔
	支持 LoadBalancer 类型 Service	✔	✔ 基于腾讯云负载均衡 CLB	无
	支持 CLB 类型的 Ingress	✔	✔	无
	支持 Nginx 类型的 Ingress	✔	✔	无
其他	支持 qGPU	✔	无	无

注册节点价格说明

最近更新时间：2023-11-10 17:39:22

注册节点公测期间，只收取 TKE 托管集群以及涉及到的其他云资源费用，**暂不收取注册节点管理费用**。

> **注意**
> 您需要对 TKE 集群管理费用以及实际使用的其他云资源进行付费，详细信息请参见 容器服务计费概述 。

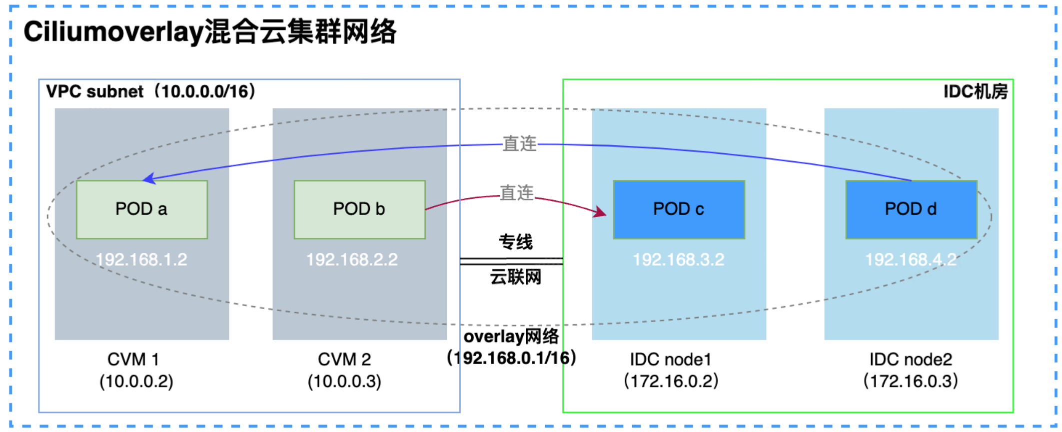
网络模式（专线版）

最近更新时间：2024-04-15 17:13:43

在专线场景下，添加了注册节点的集群，可能包含众多不同网络环境的计算节点，如 IDC 网络环境和公有云 VPC 网络环境的计算节点。在 TKE 不同网络类型场景下（GlobalRouter、VPC-CNI、Cilium-Overlay），IDC 注册节点上 Pod 的网络使用存在一定的限制。

Cilium-Overlay 网络模式

新建 TKE 集群管理云下 IDC 节点场景下，建议使用 Cilium-Overlay 网络模式。



模式介绍

Cilium-Overlay 网络模式是容器服务 TKE 基于 Cilium VXLan 实现的容器网络插件，在容器层面看到的是统一的网络平面，使得 Pod 无需感知其是运行在 IDC 的计算节点还是公有云的计算节点，实现分布式云场景中，外部节点注册到 TKE 集群的网络管理。

该网络模式特征如下：

- 云上节点和注册节点共用指定容器网段。
- 容器网段分配灵活，容器 IP 段不占用 VPC 的其他网段。
- 使用 Cilium VXLan 隧道封装协议构建 Overlay 网络。

关于 Cilium Overlay 更多介绍请参见 Cilium-Overlay 模式介绍 。

优势说明

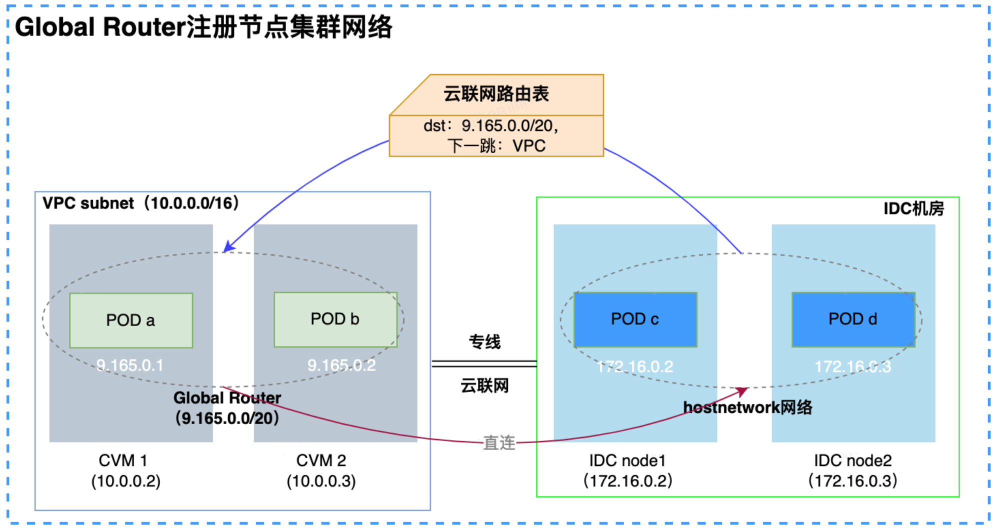
- 无需更改 IDC 基础网络设施，快速接入使用，避免网络配置变更对 IDC 已有业务带来影响。
- IDC 内 IP 资源不足，无法给业务容器分配 IDC 内的 IP，通过 Overlay 的网络模式，自定义容器网段，不占用 IDC 内部 IP 资源。

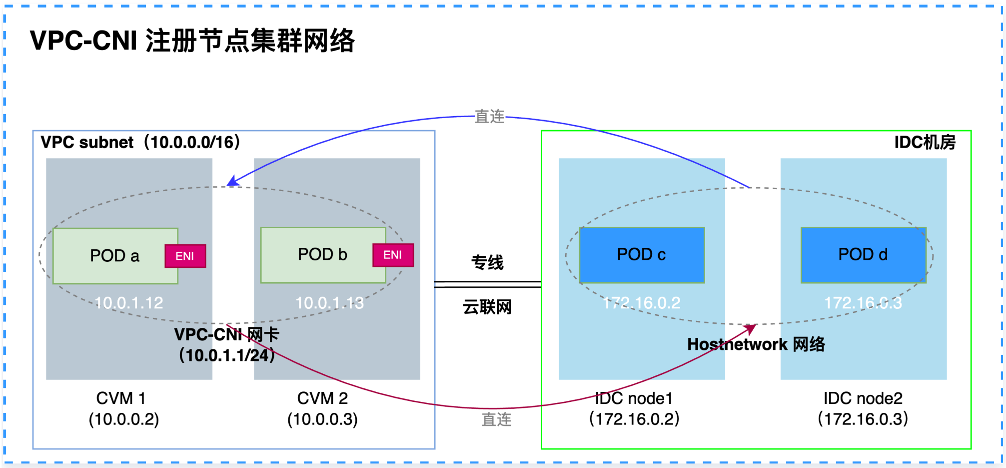
使用须知

- 使用 Cilium VXLan 隧道封装协议，有10%以内的性能损耗。
- Pod IP 在集群外不能直接访问。
- Pod 可以访问集群外节点，但是 Pod IP 会 SNAT 成 Pod 所在节点的 IP。
- 不支持固定 Pod IP。

> **警告：**
> 如果您的集群使用了 Cilium-Overlay 网络模式，请不要同时创建超级节点池，以免出现 Pod 互访问题。
> 原因：由于 Cilium-Overlay 模式下普通节点或者注册节点的 Pod 会通过 VXLan 进行封装报文，而超级节点池 Pod IP 会使用 VPC 的 IP，不会经过 VXLan 封装，因此这两个 Pod 之间无法直接互访。例如典型场景：超级节点 Pod 无法访问调度在普通节点的 CoreDNS 服务。

GlobalRouter 和 VPC-CNI 模式





模式说明

云上已有 Global Router 或 VPC-CNI 类型的 TKE 集群，将 IDC 节点加入存量集群中实现资源利用，IDC 上 Pod 仅支持使用 Hostnetwork 网络模式运行，复用 IDC 节点 Node 网络。

使用须知

Hostnetwork 复用主机网络，监听相同端口的 Pod 应避免调度在一台主机上（通过污点、亲和性等机制），否则会导致 Pod 无法进入 running 状态。

创建注册节点（专线版）

最近更新时间：2024-11-12 17:25:12

本文介绍如何使用注册节点专线版相关功能。

使用须知

在开始使用注册节点能力前，需要保证服务器和网络满足约束要求，详情请参见 约束条件和网络模式 。

操作步骤

1. 注册节点安装操作系统

目前注册节点的操作系统仅支持 TencentOS Server 3.1 和 TencentOS Server 2.4(TK4)。具体信息如下：

操作系统名称	说明
TencentOS Server 3.1	与 CentOS 8 用户态完全兼容，配套基于社区 5.4 LTS 内核深度优化的 tkernel4 版本。
TencentOS Server 2.4(TK4)	与 CentOS 7 用户态完全兼容，配套基于社区 5.4 LTS 内核深度优化的 tkernel4 版本。

> **说明：**
> TencentOS Server 是腾讯云针对云场景研发的 Linux 操作系统，提供特定的功能及性能优化，为云服务器实例中的应用程序提供更高的性能及更加安全可靠的运行环境。更多详细介绍，请参见 TencentOS Server 。

2. NVIDIA-SMI 驱动安装（仅限 GPU 节点）

1. 注册节点（专线版）节点池支持 GPU 节点池，需要用户自行安装 NVIDIA 驱动以及 CUDA。
2. NVIDIA 驱动下载安装请参见：
    - **NVIDIA Tesla 驱动**：安装 NVIDIA Tesla 驱动
    - **NVIDIA GRID 驱动**：安装 NVIDIA GRID 驱动
3. 安装 CUDA 请参见：安装 CUDA

3. 注册节点特性支持

对于 TKE 集群不同的网络类型，开启注册节点功能支持的方式有所不同，具体开启步骤参考如下：

3.1 Cilium-Overlay 网络模式集群

云上云下节点统一 Cilium Overlay 的网络，仅支持在网络模型为 Cilium Overlay 的集群中添加注册节点。您可以选择已有的 Cilium Overlay 集群或新建集群。

新建 Cilium Overlay 集群的操作步骤如下：

1. 登录 容器服务管理控制台，选择左侧导航中的**集群**。
2. 在集群管理页面，单击**新建**。具体操作请参见 创建集群 。
3. 在**集群信息**中，请参考如下信息进行填写：

- **容器网络插件**：此处需选择 Cilium Overlay。
- **子网选择**：TKE 会在上述子网内创建代理弹性网卡，用于代理注册节点访问云上资源。

4. Cilium Overlay 集群创建完毕后，默认开启注册节点的特性，无需您手动开启。

3.2 GlobalRouter 和 VPC-CNI 模式集群

在这两种网络模式下，需要用户手动开启注册节点（专线版）特性能力，开启方式如下：

1. 登录 容器服务控制台，选择左侧导航中的**集群**。
2. 在集群管理页面，单击集群 ID，进入集群基本信息页面。
3. 在**节点和网络信息**中，单击注册节点能力右侧的，开启能力。
4. 在开启注册节点能力对话框中，请根据以下信息进行设置：

- **专线连接**：选择**开启支持**。
- **子网选择**：选择代理访问子网 subnet 网段。

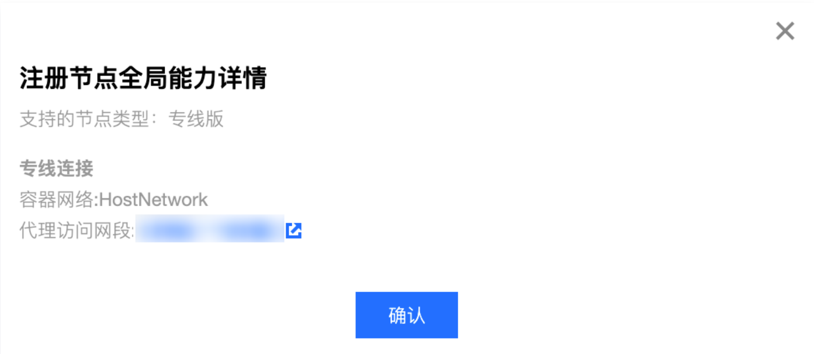
- ○ **代理访问网段**：TKE 会在上述子网内创建代理弹性网卡，用于代理注册节点访问云上资源。代理访问网段的详情解释，请参见 注册节点代理介绍 。
- ○ **容器网络**：目前默认只支持 HostNetwork。

> ⚠ **注意：**
> GR 和 VPC-CNI 模式下，注册节点上的 Pod 仅支持 HostNetwork 网络模式，网络说明请参见 网络模式 。

5. 单击**确认开启**，开启成功后，节点和网络信息如下图所示：

> ⚠ **注意：**
> 为保证 coredns、ipamd、csi-cbs-controller、l7-lb-controller、csi-cbs-node 组件的正常运行，需要确保组件运行在云上节点，因此开启过程中会对组件增加亲和性调度属性并且重建组件的 pod，会产生如下影响：
> 1、ipamd 组件重建过程中，pod 创建会因为分配不到 ip 而 暂时 pending，待 ipamd 组件恢复后 pod 创建会自动恢复正常。
> 2、采用了 ipvs 转发模式的情况下，配置的 ipvs-udp-timeout 过大会导致业务访问 dns 失败，具体处理方式请参见 配置会话保持 。

6. 单击“支持的节点类型：专线版”右侧的**详情**，可以查看专线版开启细节。如下图所示：



4. 添加注册节点

创建注册节点池（专线版）

> ⓘ **说明：**
> 注册节点（专线版）仅支持通过节点池管理，同时专线版支持 GPU 节点，CPU 节点和 GPU 节点在不同的节点池进行管理。

1. 登录 容器服务控制台 ，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择目标集群 ID，进入集群的基本信息页面。
3. 在**节点管理** > **Worker 节点**页面，选择**节点池**，单击**新建**。
4. 选择节点类型为**注册节点**，单击**创建**。在新建节点池页面，参考以下提示进行设置。

- **网络类型**：这里因为只开启了“专线连接”，因此选择默认“专线连接”即可；如果同时开启了“专线连接”和“公网连接”，这里选择“专线连接”即可。
- **节点类型**：选择线下 IDC 节点资源类型，CPU 节点和 GPU 节点在不同的节点池进行异构管理。
- **节点池名称**：自定义，可根据业务需求等信息进行命名，方便后续资源管理。
- **节点池类型**：默认为**注册节点池**类型。
- **容器网络**：默认为 **HostNetwork** 网络。
- **容器目录**：勾选即可设置容器和镜像存储目录，建议存储到数据盘。例如 `/var/lib/docker` 。
- **运行时组件**：容器运行时组件，当前支持 **docker** 和 **containerd**。
- **运行时版本**：容器运行时组件的版本。
- **封锁初始节点**：勾选**开启封锁**后，将不接受新的 Pod 调度到该节点，需要手动取消封锁的节点，或在自定义数据中执行 取消封锁命令，请按需设置。
- **Label**：单击**新增 Label**，即可进行 Label 自定义设置。该节点池下所创建的节点均将自动增加此处设置的 Label，可用于后续根据 Label 筛选、管理第三方节点。

- **Taints**：节点属性，通常与 `Tolerations` 配合使用。此处可为节点池下的所有节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在的不符合条件的 Pod 也将会被驱逐。
  Taints 内容一般由 `key` 、 `value` 及 `effect` 三个元素组成。其中 `effect` 可取值通常包含以下三种：
  - **PreferNoSchedule**：非强制性条件，尽量避免将 Pod 调度到设置了其不能容忍的 taint 的节点上。
  - **NoSchedule**：当节点上存在 taint 时，没有对应容忍的 Pod 一定不能被调度。
  - **NoExecute**：当节点上存在 taint 时，对于没有对应容忍的 Pod，不仅不会被调度到该节点上，该节点上已存在的 Pod 也会被驱逐。以设置 Taints `key1=value1: PreferNoSchedule` 为例，控制台配置如下图所示：

- **Annotations**：单击**新增**，即可进行 Annotations 自定义设置。
- **Management**：单击**新增**，即可进行 Management 自定义设置。
- **Kubelet自定义参数**：单击**新增**，即可对注册节点上的 kubelet 的参数进行自定义设置。
- **自定义数据**：指定自定义数据来配置节点，即当节点启动后运行配置的脚本。需确保脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的 `/usr/local/qcloud/tke/userscript` 路径查看。

5. 单击**创建节点池**，创建成功后，会显示节点池信息如下：

添加专线版注册节点

成功创建注册节点池后，此时节点池内还没有节点，请参考以下步骤添加注册节点：

1. 在节点池名片页面，单击目标节点池 ID。
2. 进入该节点池详情页，单击**新建节点**，获取导入节点的脚本。
3. 在弹出的对话框中，选择节点初始化资源的下载方式，复制或下载脚本。如下图所示：

- 公网：默认选中，IDC 节点通过公网直接下载安装脚本文件（文件大小31KB）。
- 内网：用户 IDC 节点无法访问公网时，经由专线访问内网来下载安装脚本文件。

4. 在您的机器上执行脚本。

> **注意：**
> 脚本下载链接1小时后过期。因为脚本通过 COS 下载，所以您需要确保 IDC 节点能够通过内网/外网访问 COS。

5. 执行如下命令，完成节点添加：

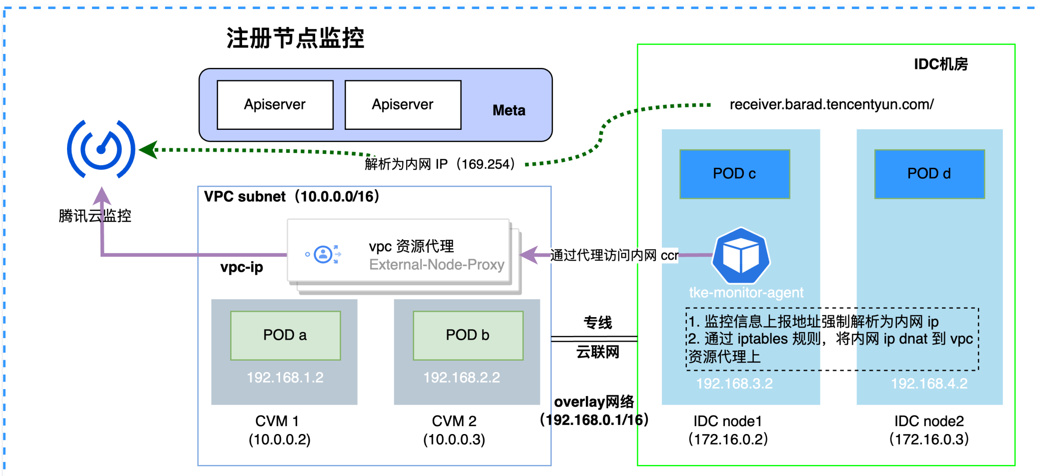
```
./add2tkectl-cls-m57oxxxp-np-xxxx install
```

> **说明：**
> 如外部节点上安装有相关 docker、containerd 组件而添加失败的情况，可以先执行以下清理的指令，再进行添加。
>
> ```
> ./add2tkectl-cls-m57oxxxp-np-xxxx clear
> ```

附录

注册节点代理介绍

开启注册节点专线版时，需要用户选择子网创建代理网卡后，作为代理提供给用户节点访问云上资源，其原理如下图：



- 开启注册节点专线版后，会在用户选择的 vpc 以及 subnet 下创建上图中的 **“vpc 资源代理”**，此代理服务会占用用户选择的子网下的一个 vpc IP 地址，用于和用户 IDC 节点通信。
- 用户 IDC 节点通过 iptables 规则，会将 ccr 镜像的访问地址，以及监控 agent 上报监控信息的目标地址，都 dnat 到 “vpc 资源代理”的 IP，另外还会代理云上认证服务等其他各种类型服务地址。

创建注册节点（公网版）

最近更新时间：2024-11-15 11:03:02

本文介绍如何使用注册节点公网版相关功能。

使用须知

在开始使用注册节点能力前，需要保证服务器和网络满足约束要求，详情请参见 约束条件和网络模式 。

> **注意：**
> 由于腾讯云 CLB 服务当前已经默认域名化，详情请参见 域名化公网负载均衡上线公告；同时将不允许用户直接使用平台侧提供的默认域名进行访问，详情请参见 域名化公网负载均衡域名升级公告 。因此，如果您需要使用注册节点公网版，需要处理域名相关工作，请参见 配置负载均衡的转发域名 。若您需要自行在腾讯云注册自定义域名（或者将域名迁移到腾讯云），同时添加 CNAME 解析到 CLB 的默认域名，详情请参见以下操作步骤。

前提条件

注册节点安装操作系统

目前注册节点的操作系统仅支持 TencentOS Server 3.1 和 TencentOS Server 2.4(TK4)。具体信息如下：

操作系统名称	说明
TencentOS Server 3.1	与 CentOS 8 用户态完全兼容，配套基于社区 5.4 LTS 内核深度优化的 tkernel4 版本。
TencentOS Server 2.4(TK4)	与 CentOS 7 用户态完全兼容，配套基于社区 5.4 LTS 内核深度优化的 tkernel4 版本。

> **说明：**
> TencentOS Server 是腾讯云针对云的场景研发的 Linux 操作系统，提供特定的功能及性能优化，为云服务器实例中的应用程序提供更高的性能及更加安全可靠的运行环境。更多详细介绍，请参见 TencentOS Server 。

操作步骤

步骤1：创建 TKE 标准集群

1. 登录 容器服务控制台，通过控制台创建标准集群，详情请参见 创建集群（新版） 。
2. 当前只支持 GR 模式集群开启注册节点公网版，在创建集群时，您需要在**网络配置 > 容器网络插件**中选择 GlobalRouter 模式。

> 🔔 **警告：**
> 注册节点（公网版）的容器网络配置创建好后，不支持修改。
> 请合理规划好容器网络段，和单节点 Pod 数量上限值，确保当前容器网络配置下，集群最多容纳的节点数量满足业务需求。
> 如果超过网络规划承载的节点数，需要通过重建集群或者重建节点的方式进行业务迁移后方可扩容。

步骤2：添加普通节点

注册节点公网版需要腾讯云普通节点做网络代理，用于连接注册节点和集群 api-server 之间的网络访问。您可以参考以下步骤添加普通节点：

1. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
2. 在**节点管理 > 节点**中，选择**更多操作 > 新建普通节点**。如下图所示：

3. 在**新建节点管理**中，请参考以下提示进行设置：

推荐配置：
- 机型：规格大于2核4G。
- 云服务器数量：2。
- 公网带宽：≥10Mbps。

必须配置：
- 安全组：安全组入站规则放通 TCP 和 UDP 协议 0.0.0.0/0 来源的 30000 - 32768 端口地址段，用于 LoadBalancer 类型的 Service 经过 NodePort 转发。详情请参见 容器服务安全组设置 。

步骤3：开启注册节点公网版

> **说明：**
> 根据 使用须知，需要首先准备好对外服务的 CLB 自定义域名，本文档示例为：**test.example.io**。参考 配置负载均衡的转发域名 文档，您需要在腾讯云上购买此域名，或者将域名从其他服务商迁移到腾讯云，同时您还需要将此域名进行备案，后续才可以正常进行配置访问。

开启方式如下：

1. 在集群管理页面，单击集群 ID，进入集群基本信息页面。
2. 在**节点和网络信息**中，单击注册节点能力右侧的，开启能力。
3. 在开启注册节点能力对话框中，在 “公网连接” 处选择**开启支持**。如下图所示：

> **注意：**
> 这里需要填入您确认好的自定义域名，本文档使用 **test.example.io** 来进行示例。TKE 会默认将此域名进行签名，保证外网节点可以通过此域名访问到云上集群控制面。

4. 单击**确认开启**，开启成功后，节点和网络信息如下图所示：

5. 单击**支持的节点类型：公网版**右侧的**详情**，可以查看公网版开启细节，**公网连接地址**即为 CLB 配置的默认访问域名，如下图所示：

6. 在腾讯云云解析 DNS 中配置 CNAME 解析，保证自定义域名 test.example.io 可以CNAME 到上述 公网连接地址，如下图所示：

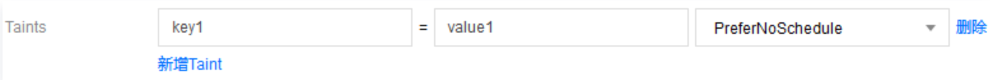
步骤4：创建注册节点池（公网版）

> **说明：**
> 注册节点（公网版）仅支持通过节点池管理，同时公网版当前暂时只支持 CPU 节点。

1. 登录 容器服务控制台 ，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，选择目标集群 ID，进入集群的基本信息页面。
3. 在**节点管理** > **Worker 节点**页面，选择**节点池**，单击**新建**。
4. 选择节点类型为**注册节点**，单击**创建**。在新建节点池页面，参考以下提示进行设置。

- **网络类型**：这里因为只开启了公网连接，因此选择默认“公网连接”即可；如果同时开启了“专线连接”和“公网连接”，这里选择“公网连接”即可。
- **节点类型**：注册节点公网版当前暂时只支持 CPU 节点。
- **节点池名称**：自定义，可根据业务需求等信息进行命名，方便后续资源管理。
- **节点池类型**：默认为**注册节点池**类型。

- **容器目录**：勾选即可设置容器和镜像存储目录，建议存储到数据盘。例如 `/var/lib/docker` 。
- **运行时组件**：容器运行时组件，当前支持 **docker** 和 **containerd**。
- **运行时版本**：容器运行时组件的版本。
- **封锁初始节点**：勾选**开启封锁**后，将不接受新的 Pod 调度到该节点，需要手动取消封锁的节点，或在自定义数据中执行 取消封锁命令，请按需设置。
- **Label**：单击**新增 Label**，即可进行 Label 自定义设置。该节点池下所创建的节点均将自动增加此处设置的 Label，可用于后续根据 Label 筛选、管理第三方节点。
- **Taints**：节点属性，通常与 `Tolerations` 配合使用。此处可为节点池下的所有节点设置 Taints，确保不符合条件的 Pod 不能够调度到这些节点上，且这些节点上已存在的不符合条件的 Pod 也将会被驱逐。
  Taints 内容一般由 `key` 、 `value` 及 `effect` 三个元素组成。其中 `effect` 可取值通常包含以下三种：
  - **PreferNoSchedule**：非强制性条件，尽量避免将 Pod 调度到设置了其不能容忍的 taint 的节点上。
  - **NoSchedule**：当节点上存在 taint 时，没有对应容忍的 Pod 一定不能被调度。
  - **NoExecute**：当节点上存在 taint 时，对于没有对应容忍的 Pod，不仅不会被调度到该节点上，该节点上已存在的 Pod 也会被驱逐。以设置 Taints `key1=value1: PreferNoSchedule` 为例，控制台配置如下图所示：



- **Annotations**：单击**新增**，即可进行 Annotations 自定义设置。
- **自定义数据**：指定自定义数据来配置节点，即当节点启动后运行配置的脚本。需确保脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的 `/usr/local/qcloud/tke/userscript` 路径查看。

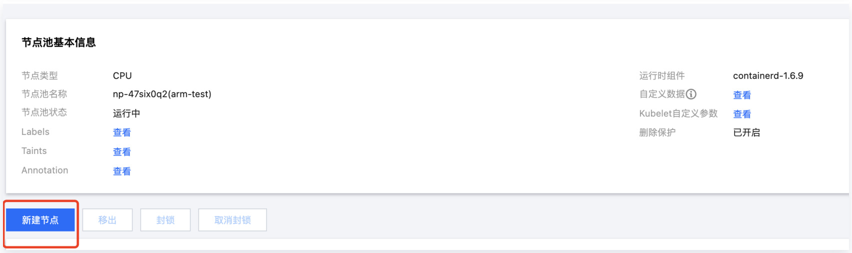
5. 单击**创建节点池**，创建成功后，会显示节点池信息如下：

步骤5：添加公网版注册节点

成功创建注册节点池（公网版）后，此时节点池内还没有节点，请参考以下步骤添加注册节点：

1. 在节点池名片页面，单击目标节点池 ID。

2. 进入该节点池详情页，单击**新建节点**，获取导入节点的脚本。



3. 在**初始化脚本**中，选择节点初始化资源的下载方式，复制或下载脚本。如下图所示：

> **注意：**
> 注册节点（公网版）当前只支持通过“公网”获取，因此必须保证接入的节点能够访问此公网链接。同时此下载链接有效期为1小时，请在有效期内使用该链接下载注册脚本。

4. 在您的机器上执行命令，下载 edgectl 脚本，并执行如下命令，完成节点添加：

```
#注册前环境检查
./edgectl check

#执行注册
./edgectl install -n [nodename] -i [flannel-eth]
#nodename 指定接入节点在集群中的名称
#flannel-eth 指定接入节点的主网卡，flannel 会基于此网卡实现 pod 跨节点互通
```

> **说明：**
> 如注册节点上安装有相关 docker、containerd 组件而添加失败的情况，可以先执行以下清理的指令，再进行添加。
>
> ```
> ./edgectl clear
> ```

最佳实践

注册节点公网版开启后会默认在 edge-system 命名空间下部署 tunnel-proxy 的 Pod。该 Pod 内包含 api-proxy 容器，用于作为注册节点和 API Server 的访问代理。为了尽量少占用用户资源，默认的 req/limit 设置如下：

```yaml
      image: ccr.ccs.tencentyun.com/tkeimages/apiserver-proxy:v1.4.5
      imagePullPolicy: IfNotPresent
      name: kubernetes-proxy
      ports:
      - containerPort: 443
        name: https
        protocol: TCP
      resources:
        limits:
          cpu: 200m
          memory: 400Mi
        requests:
          cpu: 10m
          memory: 20Mi
```

如果用户需要添加的节点数超过5个，建议在云上购买大规格机器，推荐为3台4c/8g的机器，并修改 api-proxy 容器的 req/limit 配置。

```yaml
      image: ccr.ccs.tencentyun.com/tkeimages/apiserver-proxy:v1.4.5
      imagePullPolicy: IfNotPresent
      name: kubernetes-proxy
      ports:
      - containerPort: 443
        name: https
        protocol: TCP
      resources:
        limits:
          cpu: “1”
          memory: 2Gi
        requests:
          cpu: “1”
          memory: 2Gi
```

```yaml
    - image: ccr.ccs.tencentyun.com/tkeimages/openresty:v2.0.0
      imagePullPolicy: IfNotPresent
      name: tunnel-proxy
      ports:
      - containerPort: 9000
        name: https
        protocol: TCP
      resources:
        limits:
          cpu: “1”
          memory: 2Gi
        requests:
          cpu: "1"
          memory: 2Gi
```

编辑注册节点池

最近更新时间：2023-11-10 17:39:22

本文介绍如何编辑注册节点池。

操作步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**列表页面，选择目标集群 ID，进入该集群**基本信息**页面。
3. 选择左侧菜单栏中的**节点管理** > **Worker 节点**，进入节点池列表页面。

4. 选择计划修改的类型为**注册节点**的节点池，单击**编辑**。
5. 在**调整注册节点池配置**弹窗中，支持修改节点池名称、Labels、Taints 等信息。如下图所示：

注意：

Labels、Taints 相关的修改会在节点池中的所有节点上生效，如果有特殊调度策略请务必谨慎操作。

移除注册节点

最近更新时间：2023-11-10 17:39:22

本文介绍如何移除注册节点。

操作步骤

您可以通过删除注册节点池的方式批量移除注册节点，也可以指定节点池中的节点进行移除。

删除注册节点池

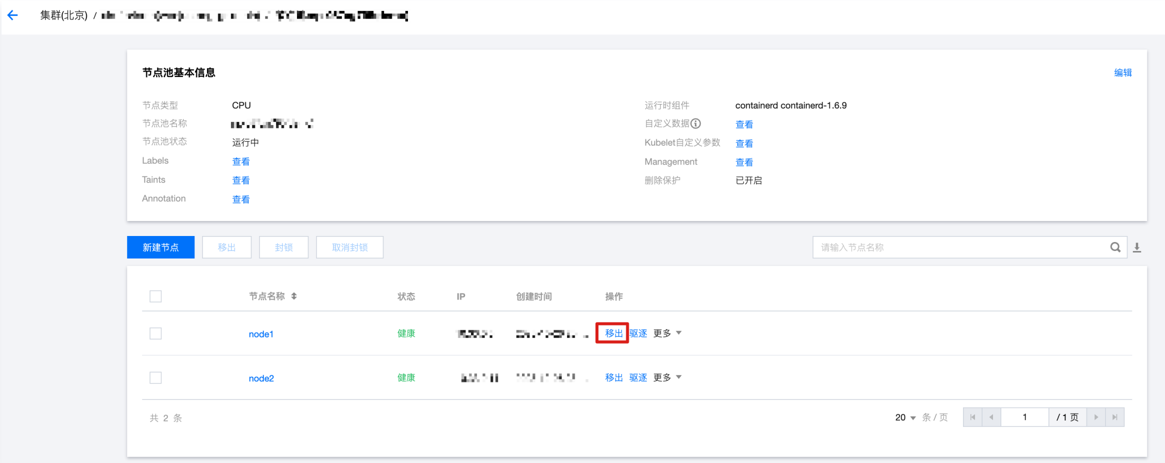
1. 登录容器服务控制台，选择左侧导航栏中的 **集群** 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理 > Worker 节点**中，选择**节点池**。
4. 在节点池卡片中，选择节点池名片页右上角的**更多 > 关闭删除保护**。

5. 在节点池卡片中，选择节点池名片页右上角的**更多 > 删除**。

6. 确认节点池信息，并单击**确定**。

删除指定注册节点

1. 登录容器服务控制台，选择左侧导航栏中的 **集群** 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理 > Worker 节点**中，选择**节点池**。
4. 在节点池名片页面，单击目标节点所在的**节点池 ID**。
5. 选择需要删除的注册节点，单击**移出**。



注意

- 删除注册节点后，注册节点仅会被移出集群，不会清除节点上运行的 Pod，请在节点上使用注册脚本的 clear 命令进行清除。
- 删除节点后为了保证安全，建议重装节点或者执行以下命令删除节点上 kube-apiserver 访问配置：

```
rm -rf /etc/kubernetes $HOME/.kube
```

流量接入

最近更新时间：2023-07-12 17:02:01

本文主要介绍如何使用腾讯负载均衡 CLB，实现注册节点的四层以及七层的流量接入。

使用前提

- 集群已经 创建注册节点 。
- 使用腾讯云 CLB 实现注册节点流量接入的能力，基于 CLB 混合云部署特性实现，请确保已经开通此特性。更多信息请参考 混合云部署 。

Service

Kubernetes Service 资源抽象了访问一组 Pod 的策略，屏蔽了后端实例的动态变化和对多实例的负载均衡，用于管理集群中基于四层网络的服务访问。
Kubernetes 的 ServiceTypes 允许指定 Service 类型，目前主要支持 `ClusterIP` 、 `NodePort` 、 `LoadBalancer` 三种类型。ClusterIP 和 NodePort 类型的 Service，在不同云服务商或是自建集群中的行为表现通常情况下相同。而 `LoadBalancer` 类型的 Service，由于使用了云服务商的负载均衡进行服务暴露，云服务商会围绕其负载均衡的能力提供不同的额外功能。
这里介绍如何基于腾讯云 CLB，使用 LoadBalancer 类型的 Service 实现注册节点的四层流量接入。

使用方式

创建 Loadbalancer Service 时，为 Service 添加相应的 Annotation：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/hybrid-type: CCN
    service.cloud.tencent.com/snat-pro-info: '{"snatIps": [{"subnetId":"subnet-
12345678","ip":"192.168.0.1"}]}'
  name: test
  .......
```

Annotation 说明

此处仅介绍与注册节点流量接入相关的参数：

- `service.cloud.tencent.com/hybrid-type`
  通过该 Annotation 开启分布式云场景中注册节点流量接入的特性，需要指定流量转发的方式，当前仅支持云联网。参数有效值：CCN。
- `service.cloud.tencent.com/snat-pro-info`
  通过该 Annoation 为 CLB 配置 SnatIP，否则 CLB 后端无法绑定 IDC IP。需要指定 SnatIP 的所在子网以及 IP，其中 IP 非必填，如不填写则从指定的子网中默认分配。

> **说明**
> 关于 SNAT IP 的说明如下：
> - SNAT IP 主要用于混合云部署中将请求转发至 IDC 内服务器的场景，使用负载均衡绑定云联网打通的 IDC 内 IP 时，必须分配 SNAT IP. SNAT IP 是您的 VPC 的内网 IP。
> - 单个 CLB 实例最多支持配置 10 个 SNAT IP。
> - 单个 CLB 实例的单个规则配置单个 SNAT IP，绑定单个后端服务后的连接数最大是 5.5 万个，若增加 SNAT IP 或增加后端服务时，连接数等比例增加。例如 1 个 CLB 实例配置了 2 个 SNAT IP，后端绑定了 10 个端口，此时该 CLB 实例总的连接数是：2 x 10 x 5.5万 = 110万个。您可以根据连接数来评估 SNAT IP 的分配个数。
> - **删除 SNAT IP 时，该 SNAT IP 上的连接会全部断开，请谨慎操作。**

使用示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
```

```
    service.cloud.tencent.com/hybrid-type: CCN
    service.cloud.tencent.com/snat-pro-info: '{"snatIps": [{"subnetId":"xxxxxxxx"}]}'
  name: nginx-svc
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    k8s-app: nginx
  sessionAffinity: None
  externalTrafficPolicy: Local
  type: LoadBalancer
```

说明：
- 这里如不指定 `service.cloud.tencent.com/hybrid-type` 和 `service.cloud.tencent.com/snat-pro-info`，则 CLB 仅会挂载云上节点。
- 建议使用 Service Local 模式，将 `ExternalTrafficPolicy` 设置为 `Local`，避免流量通过 NAT 在节点间的转发设置，最大程度减少流量跨专线穿行。关于 Service Local 模式，请参考 Service Local 模式。

Ingress

Ingress 是允许访问到集群内 Service 规则的集合，您可以通过配置转发规则，实现不同 URL 可以访问到集群内不同的 Service。为了使 Ingress 资源正常工作，集群需运行 Ingress Controller，容器服务在集群内默认启用了基于腾讯云负载均衡器 CLB 实现的 TKE Ingress Controller。
这里介绍如何基于腾讯云负载均衡 CLB，使用 CLB 类型的 Ingress 实现注册节点的七层流量接入。

使用方式

创建 CLB 类型的 Ingress 时，为 Ingress 添加相应的 Annotation：

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: qcloud
    ingress.cloud.tencent.com/hybrid-type: CCN
    ingress.cloud.tencent.com/snat-pro-info: '{"snatIps": [{"subnetId":"subnet-
12345678","ip":"192.168.0.1"}]}'
  name: test-ingress
  ........
```

Annotation 说明

- `kubernetes.io/ingress.class`
  配置 Ingress 类型，这里我们使用 CLB 类型的 Ingress，有效值为 **qcloud**。
- `ingress.cloud.tencent.com/hybrid-type`
  通过该 Annotation 开启分布式云场景中注册节点流量接入的特性，需要指定流量转发的方式，当前仅支持云联网。参数有效值：**CCN**。
- `ingress.cloud.tencent.com/snat-pro-info`
  通过该 Annoation 为 CLB 配置 SnatIP，否则 CLB 后端无法绑定 IDC IP。需要指定 SnatIP 的所在子网以及 IP，其中 IP 非必填，如不填写则从指定的子网中默认分配。

使用示例

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: qcloud
    ingress.cloud.tencent.com/hybrid-type: CCN
    ingress.cloud.tencent.com/snat-pro-info: '{"snatIps": [{"subnetId":"subnet-xxxxxx"}]}'
  name: nginx-ingress
```

```yaml
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: nginx-svc
          servicePort: 80
        path: /
        pathType: ImplementationSpecific
```

注册节点常见问题

最近更新时间：2023-11-10 17:39:22

集群类

- 注册节点特性是否支持独立部署集群?
- 使用注册节点特性，为什么要求集群内必须要存在云上节点?

节点类

- 注册节点和云上节点在能力上有哪些差异?
- 注册节点支持哪些操作系统?
- 节点上由于 docker、containerd 相关软件导致添加节点失败，如何处理?

网络、流量接入类

- 注册节点的容器如何对外暴露服务?

运维类

- 注册节点的日志如何接入日志服务 CLS?
- 注册节点如何接入 Prometheus 监控服务?
- Cilium-Overlay 模式下如何使用 webhook?

内存压缩
使用说明

最近更新时间：2025-03-13 14:22:52

本文为您介绍如何基于原生节点开启并启用内存压缩能力。

环境准备

内存压缩功能要求原生节点镜像的内核更新至最新版本（5.4.241-19-0017），可通过以下方式实现：

新增原生节点

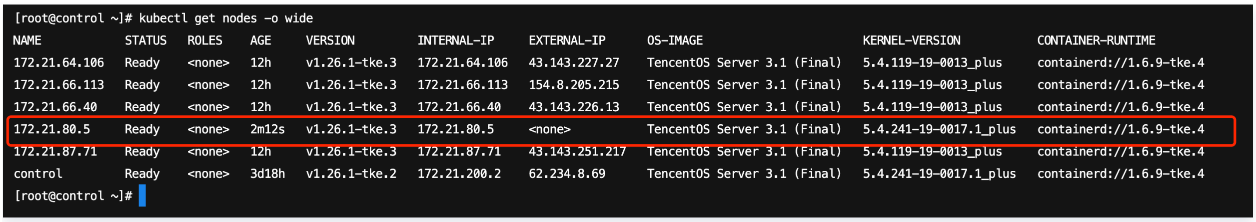
1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在**节点管理**中，选择**节点池**页签，单击**新建**。
4. 选择**原生节点**，参考 新建原生节点，完成节点池配置，该节点池下新增的原生节点将默认安装最新内核版本（5.4.241-19-0017）的镜像。

存量原生节点

若您创建了较早版本的原生节点，内核版本展示为 5.4.119-19-0013 的镜像还不支持内存压缩特性，可通过 RPM 包更新获取该特性，您可以通过 在线客服 或 提交工单 与我们联系，我们将协助您进行升级。

内核版本验证

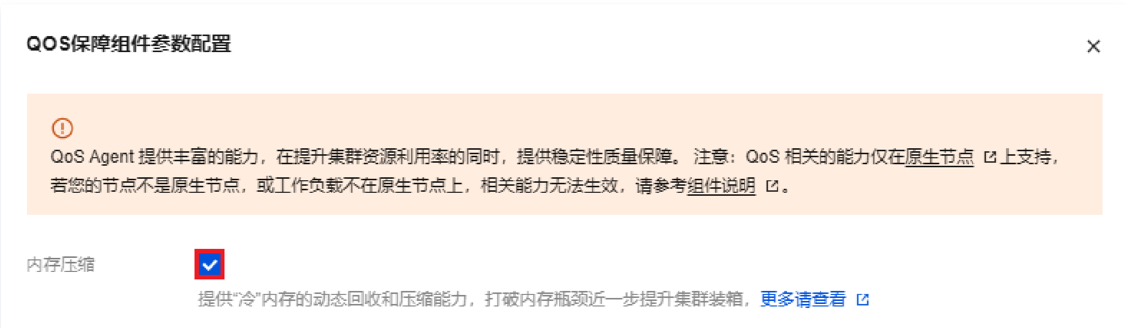
可通过执行命令 `kubectl get nodes -o wide` 查看节点的 KERNEL-VERSION 已经为最新内核版本 5.4.241-19-0017.1_plus。



```
[root@control ~]# kubectl get nodes -o wide
NAME            STATUS   ROLES    AGE     VERSION          INTERNAL-IP     EXTERNAL-IP      OS-IMAGE                       KERNEL-VERSION             CONTAINER-RUNTIME
172.21.64.106   Ready    <none>   12h     v1.26.1-tke.3    172.21.64.106   43.143.227.27    TencentOS Server 3.1 (Final)   5.4.119-19-0013_plus       containerd://1.6.9-tke.4
172.21.66.113   Ready    <none>   12h     v1.26.1-tke.3    172.21.66.113   154.8.205.215    TencentOS Server 3.1 (Final)   5.4.119-19-0013_plus       containerd://1.6.9-tke.4
172.21.66.40    Ready    <none>   12h     v1.26.1-tke.3    172.21.66.40    43.143.226.13    TencentOS Server 3.1 (Final)   5.4.119-19-0013_plus       containerd://1.6.9-tke.4
172.21.80.5     Ready    <none>   2m12s   v1.26.1-tke.3    172.21.80.5     <none>           TencentOS Server 3.1 (Final)   5.4.241-19-0017.1_plus     containerd://1.6.9-tke.4
172.21.87.71    Ready    <none>   12h     v1.26.1-tke.3    172.21.87.71    43.143.251.217   TencentOS Server 3.1 (Final)   5.4.119-19-0013_plus       containerd://1.6.9-tke.4
control         Ready    <none>   3d18h   v1.26.1-tke.2    172.21.200.2    62.234.8.69      TencentOS Server 3.1 (Final)   5.4.241-19-0017.1_plus     containerd://1.6.9-tke.4
[root@control ~]#
```

安装 QosAgent 组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 QoSAgent，同时在参数配置中勾选**内存压缩**，如下图所示：



5. 单击**确定**。
6. 在**新建组件管理**页面单击**完成**即可安装组件。

> **说明：**
> QosAgent 组件版本在1.1.5及以上才支持内存压缩能力，若您的集群已经安装过该组件，需要执行如下步骤：
> 1. 在集群的**组件管理**页面中，找到部署成功的 QoSAgent，单击右侧的**升级**。
> 2. 升级后单击**更新配置**，勾选**内存压缩**。

3. 单击**完成**。

选择节点开启压缩开关

为方便灰度验证，QosAgent 不会默认给所有原生节点都打开内存压缩所依赖的内核配置，您还需要通过 **NodeQOS** 来指定哪些节点可以开启压缩能力。

部署 NodeQOS 对象

> **说明：**
> 安装完成 QosAgent 组件后，系统会默认在集群中部署一个名称为 `compress-node` 的 NodeQOS 资源，您可直接编辑修改。

1. **部署 NodeQOS 对象**。通过 `spec.selector.matchLables` 指定在哪些节点上开启压缩开关，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: NodeQOS
metadata:
  name: compress-node
spec:
  selector:
    matchLabels:
      compression: enable
  memoryCompression:
    enable: true
```

2. **给节点打 lable 将 node 和 NodeQOS 关联**。操作步骤如下：
   2.1 登录 容器服务控制台，在左侧导航栏中选择**集群**。
   2.2 在集群列表中，单击目标集群 ID，进入集群详情页。
   2.3 在**节点管理**中，选择**节点池**页签，单击节点池卡片的**编辑**。
   2.4 在**调整节点池配置**页面中，修改 lable 并勾选**对存量节点应用本次更新**，示例中 lable 为 `compression: enable`。
   2.5 单击**确定**。

生效验证

当节点开启内存压缩后，您可以使用以下命令获取节点的 YAML 配置，通过 node 的 annotation 确认内存压缩是否正确开启。示例如下：

```
kubectl get node <nodename> -o yaml | grep "gocrane.io/memory-compression"
```

登录节点后依次检查 zram、swap、内核参数，确认内存压缩被正确开启，示例如下：

```
# 确认zram设备初始化
# zramctl
NAME       ALGORITHM DISKSIZE DATA COMPR TOTAL STREAMS MOUNTPOINT
/dev/zram0 lzo-rle       3.6G   4K   74B   12K       2 [SWAP]

# 确认设置到swap
# free -h
              total        used        free      shared  buff/cache   available
Mem:          3.6Gi       441Mi       134Mi       5.0Mi       3.0Gi       2.9Gi
Swap:         3.6Gi       0.0Ki       3.6Gi

# sysctl vm.force_swappiness
vm.force_swappiness = 1
```

选择业务启用内存压缩

部署 PodQOS 对象

1. 部署 PodQOS 对象。通过 `spec.lableSelector.matchLables` 指定在哪些 Pod 上启用内存压缩，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: memorycompression
spec:
  labelSelector:
    matchLabels:
      compression: enable
  resourceQOS:
    memoryQOS:
      memoryCompression:
        compressionLevel: 1
        enable: true
```

> **说明：**
> compressionLevel 代表压缩等级，取值范围1-4，对应算法 lz4、lzo-rle、lz4hc、zstd，对应压缩率由低到高，性能损失由小增大。

2. 创建工作负载和 PodQOS 中 labelSelector 相匹配，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: memory-stress
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: memory-stress
  template:
    metadata:
      labels:
        app: memory-stress
        compression: enable
    spec:
      containers:
        - command:
            - bash
            - -c
            - "apt update && apt install -yq stress && stress --vm-keep --vm 2 --vm-hang 0"
          image: ccr.ccs.tencentyun.com/ccs-dev/ubuntu-base:20.04
          name: memory-stress
          resources:
            limits:
              cpu: 500m
              memory: 1Gi
            requests:
              cpu: 100m
              memory: 100M
      restartPolicy: Always
```

> **注意：**
> Pod 中所有容器必须设置 memory limit。

生效验证

可通过 Pod annotation(gocrane.io/memory-compression)、进程内存使用情况、zram 或 swap 使用情况、cgroup 内存使用情况来确认 Pod 正确开启了内存压缩。

```
# qos agent 会给开启内存压缩的pod设置annotation
kubectl get pods -l app=memory-stress -o jsonpath="{.items[0].metadata.annotations.gocrane\.io\/memory-compression}"

# zramctl
NAME       ALGORITHM DISKSIZE  DATA  COMPR TOTAL STREAMS MOUNTPOINT
/dev/zram0 lzo-rle         3.6G  163M 913.9K  1.5M        2 [SWAP]

# free -h
              total        used        free      shared  buff/cache   available
Mem:          3.6Gi       1.4Gi       562Mi       5.0Mi       1.7Gi       1.9Gi
Swap:         3.6Gi       163Mi       3.4Gi

# 查看 cgroup （一般为/sys/fs/cgroup/memory/路径下）中的 memory.zram.{raw_in_bytes,usage_in_bytes} 可以看到pod有多少内存进行了压缩、压缩后内存大小，计算差值即为节约内存
cat memory.zram.{raw_in_bytes,usage_in_bytes}
170659840
934001

# 计算差值即为节约内存大小，例子中节约了170Mi内存
cat memory.zram.{raw_in_bytes,usage_in_bytes} | awk 'NR==1{raw=$1} NR==2{compressed=$2} END{ print raw - compress }'
170659840
```

压缩监控

最近更新时间：2023-12-26 14:39:41

QosAgent 在端口 8084 上提供了一系列指标，用于监控节点和 Pod 内存压缩情况，以及内存和 CPU 的压力情况。用户可以自行配置 Prometheus 和 Grafana 来进行监控。此外，我们还提供了 Grafana 的监控面板模板，方便业务快速查看内存压缩效果（请 提交工单 咨询获取）。

关键指标介绍

对象	指标名称	含义
Pod	pod_pressure_total	Pod 级别 PSI，可以显示每个 Pod 由于缺少 CPU、内存、IO 等资源而产生的等待时长情况。
	pod_memory_info	Pod 的内存情况，包括对 Pod 和容器的以下内存指标进行统计：RSS、匿名内存、文件页、活跃内存和非活跃内存。
	pod_memory_page_fault_info	Pod 的缺页情况（包含文件页、匿名页缺页情况，major、minor 缺页情况）。
	pod_memory_oom_kill	Pod 的 OOM 统计。
Node	node_pressure_total	节点 CPU、IO 和 memory 的 PSI 指标（表明是否某种资源受到限制）。
	node_memory_page_fault_distance	refault 频率，表明“热”页面被换出去的情况。
	node_memory_page_fault_major	发生磁盘读取的缺页次数。
	node_disk_io_time_seconds_total	节点磁盘 IO 总时间（通过 zram0 设备的指标，可以观察到换出、换入到 zram0 的情况）。
	node_disk_read_bytes_total	磁盘、zram0 设备 IO 读带宽。
	node_disk_reads_completed_total	磁盘、zram0 设备 IO 读取次数（间接表明了内存压缩导致匿名内存缺页的情况）。
	node_disk_writes_completed(time_seconds)_total	磁盘/zram0 设备写入次数、总耗时。
	node_memory_oom_kill	磁盘、zram0 设备的写入情况。

哪些业务可以被压缩

支持采集业务“冷”页占比作为预估可压缩值，“冷”页包含冷匿名页和冷文件页。根据预估的压缩值和业务属性，可以判断哪些业务可以进行压缩，以及预估压缩量。

- Workingset Saved：kubelet 视角观察 “Inactive anon” 的节约值。
- Memory Saved：监控视角观察 “Inactive anon + Inactive File” 节约值。

节约多少内存量

每个 Pod 节约的内存大小 = zram 压缩前的大小 − zram 压缩后的大小

内存回收是否准确

观察 “node_memory_page_fault_major” 和 PSI 指标，node_memory_page_fault_major 和 Memory PSI 指标低代表回收较为准确。

业务是否稳定

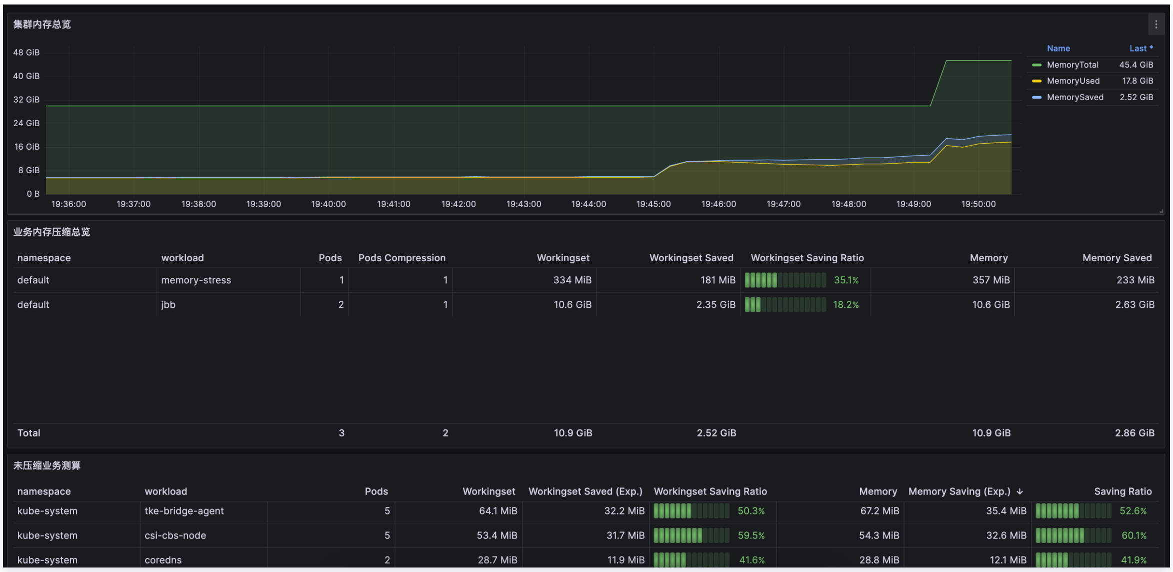
可关注 Pod/Node 的 OOM 次数、PSI、Zram0 设备的 IO（如 “node_disk_read_bytes_total ”、“node_disk_reads_completed_total ”、“node_disk_writes_completed(time_seconds)_total”）变化情况，OOM 次数、PSI、Zram0 IO 升高均代表不够稳定。

对接腾讯云 Prometheus 监控

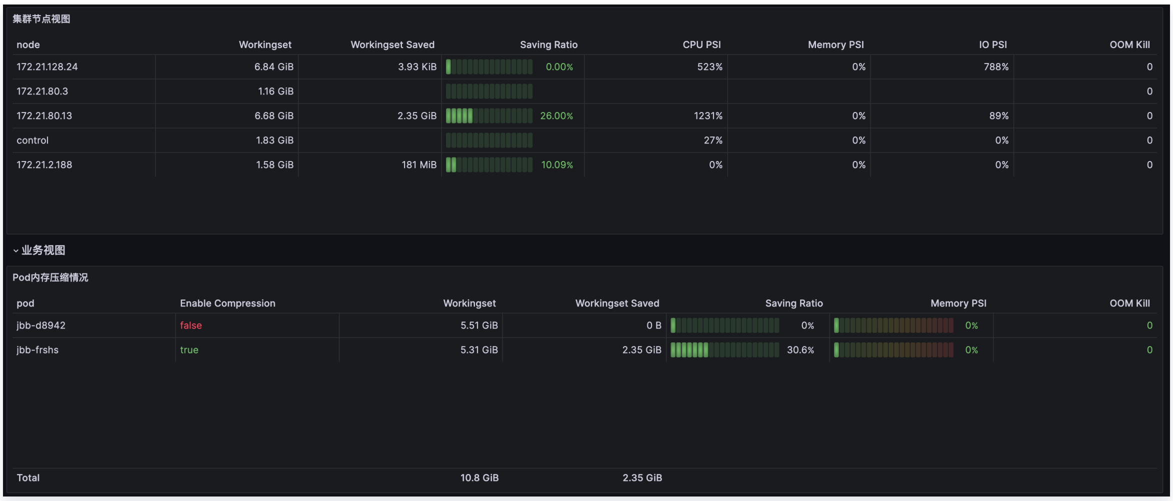
1. 登录 Prometheus 监控服务控制台。
2. 在 Prometheus 实例列表中，单击新建的**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中单击**数据采集**。

4. 在集成容器服务页面单击**关联集群**，将集群和 Prometheus 实例关联，详情见 关联集群 。
5. 在集群列表中单击集群右侧的**数据采集配置**，选择**新建自定义监控**，并填写配置信息。如下图所示：

6. 单击**确定**。
7. 在 grafana 中导入如下两个面板：
   - 集群维度面板，请 提交工单 获取，如下图所示：



   - 节点维度面板，请 提交工单 获取。如下图所示：


集群节点视图
node | Workingset | Workingset Saved | Saving Ratio | CPU PSI | Memory PSI | IO PSI | OOM Kill
172.21.128.24 | 6.84 GiB | 3.93 KiB | 0.00% | 523% | 0% | 788% | 0
172.21.80.3 | 1.16 GiB | | | | | | 0
172.21.80.13 | 6.68 GiB | 2.35 GiB | 26.00% | 1231% | 0% | 89% | 0
control | 1.83 GiB | | | 27% | 0% | 0% | 0
172.21.2.188 | 1.58 GiB | 181 MiB | 10.09% | 0% | 0% | 0% | 0
业务视图
Pod内存压缩情况
pod | Enable Compression | Workingset | Workingset Saved | Saving Ratio | Memory PSI | OOM Kill
jbb-d8942 | false | 5.51 GiB | 0 B | 0% | 0% | 0
jbb-frshs | true | 5.31 GiB | 2.35 GiB | 30.6% | 0% | 0
Total | 10.8 GiB | 2.35 GiB


CPU Usage
Memory Usage
jbb-d8942
jbb-frshs
8.34 GiB
5.58 GiB
Memory Saved
Memory Pressure
0 B
2.41 GiB
2%
0%
CPU Pressure
IO Pressure

GPU 共享

qGPU 概述

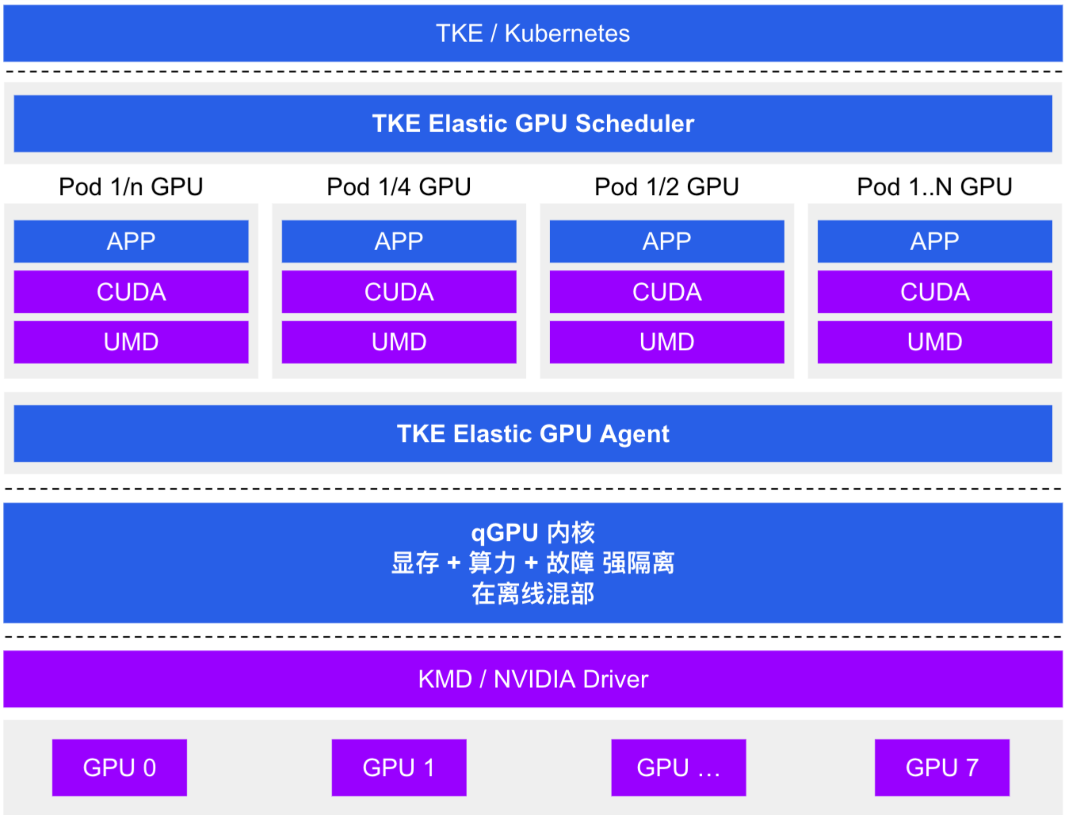
最近更新时间：2024-11-26 11:28:42

容器服务 GPU 虚拟化

腾讯云 Tencent Kubernetes Engine qGPU 服务（以下简称 TKE qGPU）是腾讯云针对 原生节点 推出的 GPU 容器虚拟化产品，支持多个容器共享 GPU 卡并支持容器间算力和显存精细隔离，同时提供业界唯一的在离线混部能力，在精细切分 GPU 资源的基础上，在最大程度保证业务稳定的前提下，提高 GPU 使用率，帮助客户大幅度节约 GPU 资源成本。qGPU 依托 TKE 对外开源的 Elastic GPU 框架，可实现对 GPU 算力与显存的细粒度调度，并支持多容器共享 GPU 与多容器跨 GPU 资源分配。同时依赖底层强大的 qGPU 隔离技术，可做到 GPU 显存和算力的强隔离，在通过共享使用 GPU 的同时，尽量保证业务性能与资源不受干扰。

> **说明：**
> qGPU 功能仅针对 TKE 原生节点开放，其他节点类型不提供有效性服务保障。

方案框架图



产品优势

- **灵活性：**精细配置 GPU 算力占比和显存大小。
- **强隔离：**支持显存和算力的严格隔离。
- **在离线：**支持业界唯一在离线混部能力，GPU 利用率压榨到最大化。
- **覆盖度：**支持主流架构 Volta（如 V100 等）、Turing（如 T4 等）、Ampere（如 A100、A10 等）。
- **云原生：**支持标准 Kubernetes 和 NVIDIA Docker。
- **兼容性：**业务不重编、CUDA 库不替换、业务无感。

使用 qGPU

最近更新时间：2024-10-25 17:53:41

使用须知

支持的 Kubernetes 版本	TKE 版本 ≥ v1.14.x
支持的节点类型	仅支持 原生节点，原生节点搭载 FinOps 理念，配合 qGPU 使用可全面提升 GPU/CPU 资源利用率。
支持的 GPU 卡架构	支持 V100 (Gn10x)、T4 (Gn7)、A100 (GT4)、A10 (Pnv4)、L20 (Pnv5b)、L40 (Pnv5i)、L40s (Pnv5) 卡。
支持的驱动版本	• 450.102.04 • 470.82.01 • 515.65.01 • 525.105.17 • 535.154.05
共享粒度	每个 qGPU 最小分配1G显存，精度单位是1G。算力最小分配5（代表一张卡的5%），最大100（代表一张卡），精度单位是5（即5、10、15、20...100）。
整卡分配	开启了 qGPU 能力的节点可按照 tke.cloud.tencent.com/qgpu-core: 100 | 200 | ...（N * 100，N 是整卡个数）的方式分配整卡。建议通过 TKE 的节点池能力来区分 NVIDIA 分配方式或转换到 qGPU 使用方式。
个数限制	一个 GPU 上最多可创建16个 qGPU 设备。建议按照容器申请的显存大小确定单个 GPU 卡可共享部署的 qGPU 个数。

注意：

如果您升级了 TKE 集群的 Kubernetes Master 版本，请注意以下事项：

- 对于托管集群，您无需重新设置本插件。
- 对于独立集群（Master 自维护），Master 版本升级会重置 Master 上所有组件的配置，这将影响到 qgpu-scheduler 插件作为 Scheduler Extender 的配置。因此，您需要先卸载 qGPU 插件，然后再重新安装。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	Namespace
qgpu-manager	DaemonSet	每 GPU 节点一个 Memory: 300M, CPU:0.2	kube-system
qgpu-manager	ClusterRole	-	-
qgpu-manager	ServiceAccount	-	kube-system
qgpu-manager	ClusterRoleBinding	-	kube-system
qgpu-scheduler	Deployment	单一副本 Memory: 800M, CPU:1	kube-system
qgpu-scheduler	ClusterRole	-	-
qgpu-scheduler	ClusterRoleBinding	-	kube-system
qgpu-scheduler	ServiceAccount	-	kube-system
qgpu-scheduler	Service	-	kube-system

qGPU 权限

说明：

权限场景章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考权限定义章节。

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要安装 qgpu ko 内核文件，创建、管理和删除 qgpu 设备，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
跟踪 pod 的状态变化，获取 pod 信息，以及在 pod 删除时清理 qgpu 设备等资源。	pods	get/list/watch
跟踪 node 的状态变化，获取 node 信息，并根据 gpu 卡驱动和版本信息以及 qgpu 版本信息给 nodes 增加 label。	nodes	get/list/watch/update
qgpu-scheduler 是基于 kubernetes 调度器 extender 机制开发的针对 qgpu 资源的扩展调度器，**需要的权限与社区其他调度类组件**（如 volcano）相同，包括跟踪和获取 pods 信息，需要把调度结果更新到 pod 的 label 和 annotation，跟踪和获取 node 信息，跟踪获取配置的 configmap，创建调度事件。	pods	get/list/update/patch
	nodes	get/list/watch
	configmaps	get/list/watch
	events	create/patch
gpu.elasticgpu.io 是 qgpu 的记录 gpu 资源信息的自定义 CRD 资源（**该功能已废弃，但为了兼容旧版本，资源定义需要保留**），由 qgpu-manager 及 qgpu-scheduler 管理，需要增删改查所有权限。	gpu.elasticgpu.io 及 gpu.elasticgpu.io/status	所有权限

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: qgpu-manager
rules:
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - update
      - get
      - list
      - watch
  - apiGroups:
      - "elasticgpu.io"
    resources:
      - gpus
      - gpus/status
    verbs:
      - '*'

---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: qgpu-scheduler
```

```
rules:
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - update
      - patch
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - bindings
      - pods/binding
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "elasticgpu.io"
    resources:
      - gpus
      - gpus/status
    verbs:
      - '*'
```

操作步骤

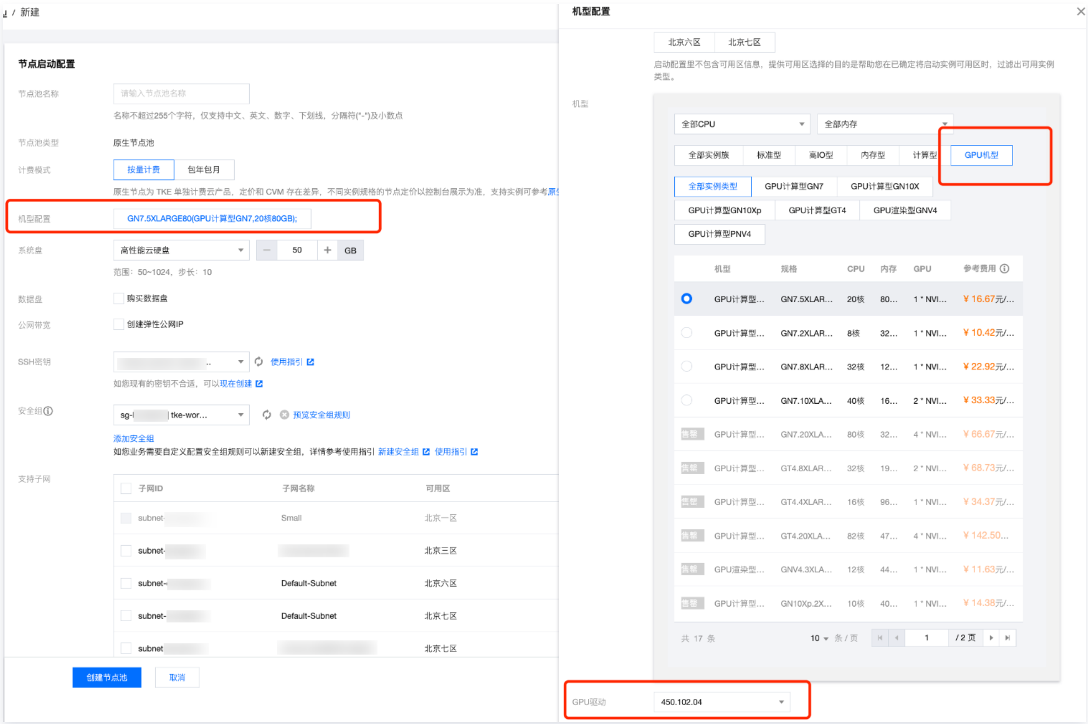
步骤1：安装 qGPU 调度组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 QGPU（GPU 隔离组件）。
5. 单击**参数配置**，设置 qgpu-scheduler 的调度策略。

  ○ **Spread**：多个 Pod 会分散在不同节点、不同显卡上，优先选择资源剩余量较多的节点，适用于高可用场景，避免把同一个应用的副本放到同一个设备上。
  ○ **Binpack**：多个 Pod 会优先使用同一个节点，适用于提高 GPU 利用率的场景。
6. 单击**完成**即可创建组件。安装成功后，需要为集群准备 GPU 资源。

步骤2：准备 GPU 资源并开启 qGPU 共享

1. 在集群列表中，单击目标集群 ID，进入集群详情页。
2. 在**节点管理 > Worker节点**中，选择**节点池**页签，单击**新建**。
3. 选择**原生节点**，单击**创建**。
4. 在**新建**页面，选择对应 GPU 机型并选择 qgpu 支持的驱动版本，如下图所示：



5. 在**运维功能设置**中，单击 **qGPU 共享**右侧开关。开关开启后，**节点池中所有新增 GPU 原生节点**默认开启 GPU 共享能力。您可以通过 Label 控制是否开启隔离能力。
6. 在**高级设置 > Labels** 中，通过节点池的高级配置来设置 Label，指定 qGPU 隔离策略：

  ○ **Label 键**：`tke.cloud.tencent.com/qgpu-schedule-policy`
  ○ **Label 值**：`fixed-share`（Label value 可填写全称或者缩写，更多取值可参考下方表格）
  当前 qGPU 支持以下隔离策略：

Label 值	缩写	英文名	中文名	含义
best-effort（默认值）	be	Best Effort	争抢模式	默认值。各个 Pods 不限制算力，只要卡上有剩余算力就可使用。 如果一共启动 N 个 Pods，每个 Pod 负载都很重，则最终结果就是 1/N 的算力。
fixed-share	fs	Fixed Share	固定配额	每个 Pod 有固定的算力配额，无法超过固定配额，即使 GPU 还有空闲算力。
burst-share	bs	Guaranteed Share with Burst	保证配额加弹性能力	调度器保证每个 Pod 有保底的算力配额，但只要 GPU 还有空闲算力，就可被 Pod 使用。例如，当 GPU 有空闲算力时（没有分配给其他 Pod），Pod 可以使用超过它的配额的算力。注意，当它所占用的这部分空闲算力再次被分配出去时，Pod 会回退到它的算力配额。

7. 单击**创建节点池**。

步骤3：给应用分配共享 GPU 资源

通过给容器器设置 qGPU 对应资源可以允许 Pod 使用 qGPU，您可以通过控制台或者 YAML 方式为应用分配 GPU 资源。

> **说明：**
> - 如果应用需要使用整数卡资源，只需填写卡数，无需填写显存（自动使用分配的 GPU 卡上全部显存）。
> - 如果应用需要使用小数卡资源（即和其他应用共享同一张卡），需要同时填写卡数和显存。

通过控制台设置

1. 在集群的左侧导航栏选择**工作负载**，在任意工作负载对象类型页面单击**新建**。本文以 Deployment 为例。
2. 在**新建** Deployment 页面，选择**实例内容器**，并填写 GPU 相关资源，如下图所示：

通过 YAML 设置

通过 YAML 来设置相关 qGPU 资源：

```
spec:
  containers:
    resources:
      limits:
        tke.cloud.tencent.com/qgpu-memory: "5"
        tke.cloud.tencent.com/qgpu-core: "30"
      requests:
        tke.cloud.tencent.com/qgpu-memory: "5"
        tke.cloud.tencent.com/qgpu-core: "30"
```

其中：
- requests 和 limits 中和 qGPU 相关的资源值必须一致（根据 K8S 的规则，可以省略掉 requests 中对 qGPU 的设置，这种情况下 requests 会被自动设置为和 limits 相同的值）。
- tke.cloud.tencent.com/qgpu-memory 表示容器申请的显存（单位G），**整数分配，不支持小数**。
- tke.cloud.tencent.com/qgpu-core 代表容器申请的算力，每个 GPU 卡可以提供100%算力，**qgpu-core 的设置应该小于100**，设置值超过剩余算力比例值，则设置失败，设置后容器可以得到一张 GPU 卡 n% 的算力。

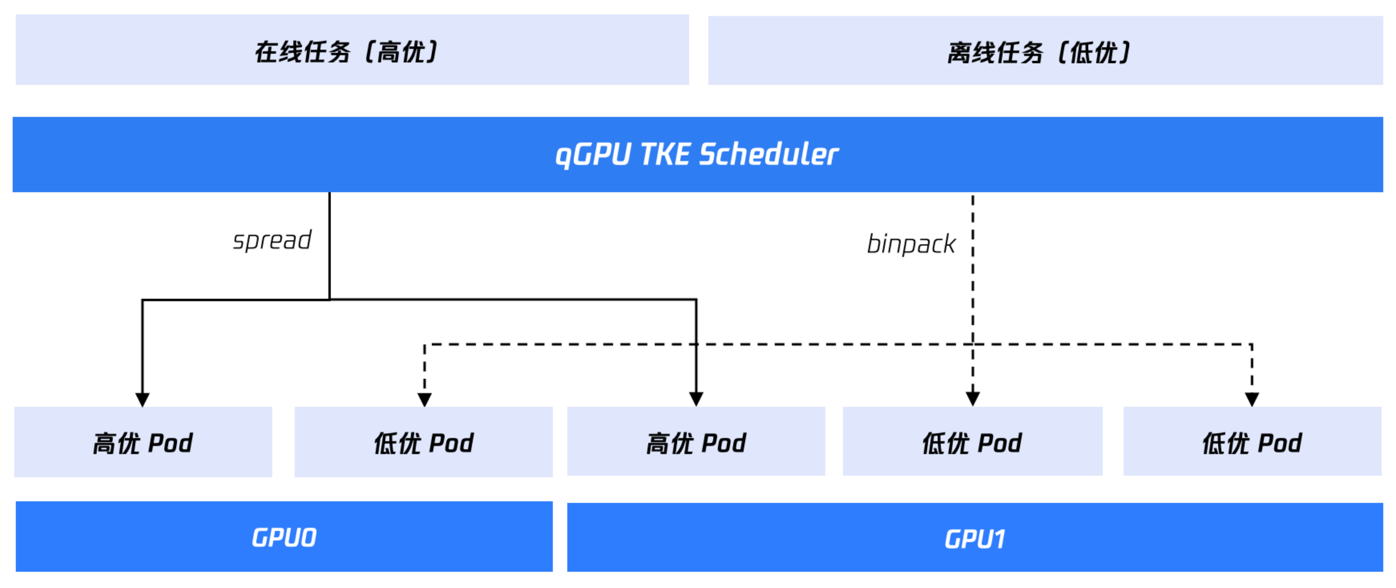
qGPU 离在线混部
qGPU 离在线混部说明

最近更新时间：2023-08-11 17:21:11

功能概述

通常场景下，qGPU Pod 会公平地使用物理 GPU 资源，qGPU 内核驱动为各任务分配等价的 GPU 时间片。但不同 GPU 计算任务的运行特点和重要性会有很大差异，导致对 GPU 资源的使用和要求不同。如实时推理对 GPU 资源比较敏感，要求低延迟，在使用时需要尽快拿到 GPU 资源进行计算，但它对 GPU 资源的使用率通常不高。模型训练对 GPU 资源使用量较大，但对延迟敏感度较低，可以忍受一定时间的抑制。

在这种背景下，腾讯云推出了 qGPU 离在线混部功能。qGPU 离在线混部是腾讯云创新性推出的 GPU 离在线混部调度技术，它支持在线（高优）任务和离线（低优）任务同时混合部署在同一张 GPU 卡上，在内核与驱动层面，实现了低优100%使用闲置算力及高优100%抢占。依赖 qGPU 离在线混部调度技术，可将用户的 GPU 资源做进一步压榨，将 GPU 利用率提升到100%，把 GPU 使用成本降到最低。



功能优势

qGPU 离在线混部实现了对 GPU 算力资源的两个“100%”控制，通过创新性的技术做到了对 GPU 算力的极限压榨：

- 100%使用高优闲置算力：高优任务闲时，低优任务可100%使用 GPU 算力资源。
- 100%抢占低优占用算力：高优任务忙时，可100%抢占低优任务所使用 GPU 算力资源。

典型场景

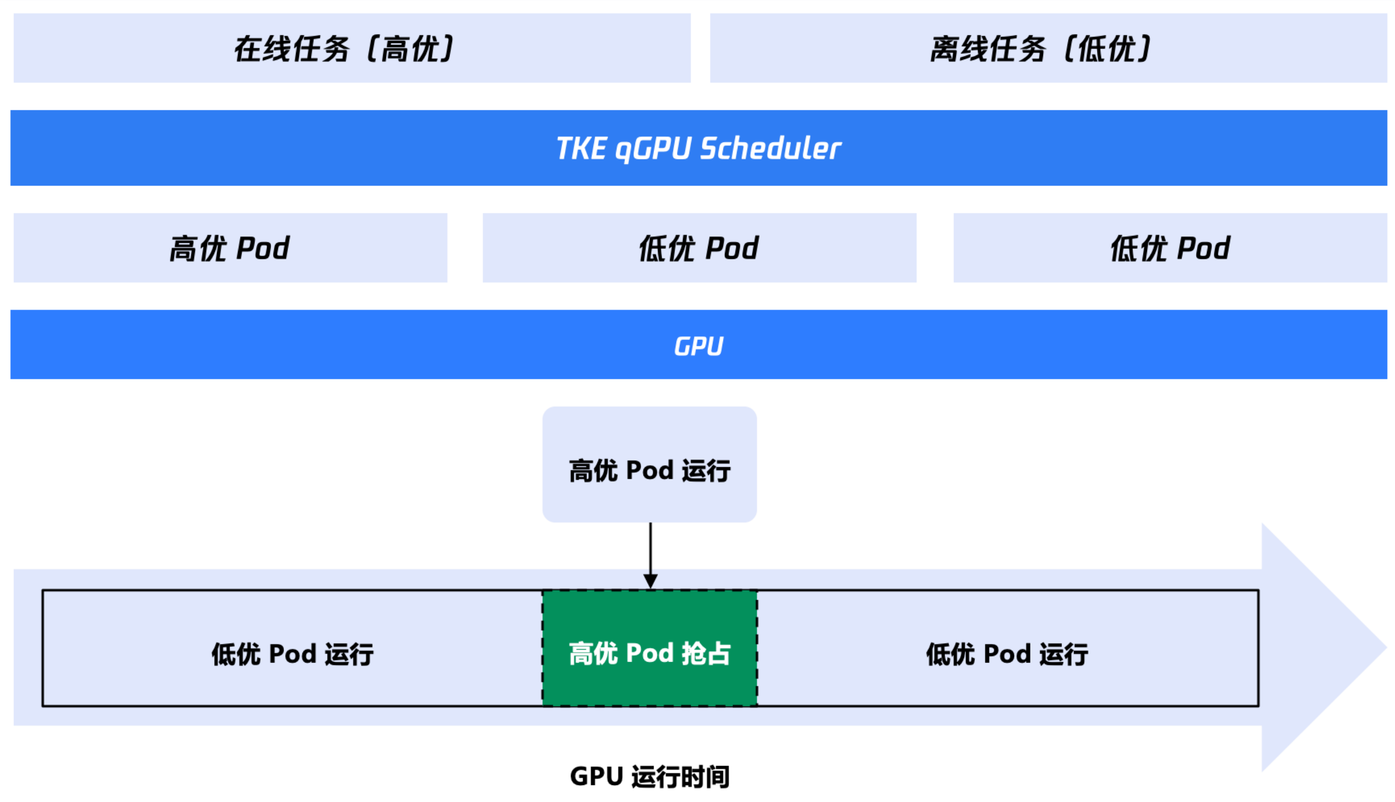
在线推理与离线推理混部

搜索 / 推荐等推理任务用于支持线上服务，对 GPU 算力实时性要求高。数据预处理等推理任务用于支持线下数据清洗和处理，对 GPU 算力实时性要求较低。通过将线上推理业务设置为高优任务，线下推理业务设置成低优任务，混合部署在同一张 GPU 卡上。

在线推理与离线训练混部

实时推理对 GPU 算力可用性要求高，资源使用较少。模型训练对 GPU 算力使用较多，敏感度要求较低。通过将推理设置为高优任务，训练设置为低优任务，混合部署在同一张 GPU 卡上。

技术原理



通过 TKE 集群提供的离在线调度策略可以开启 qGPU 离在线混部能力，帮助在线任务（高优）和离线任务（低优）更高效的共享使用物理 GPU 资源。qGPU 离在线混部技术主要包含两个功能：

功能1：低优 Pod 可以100%使用闲置算力

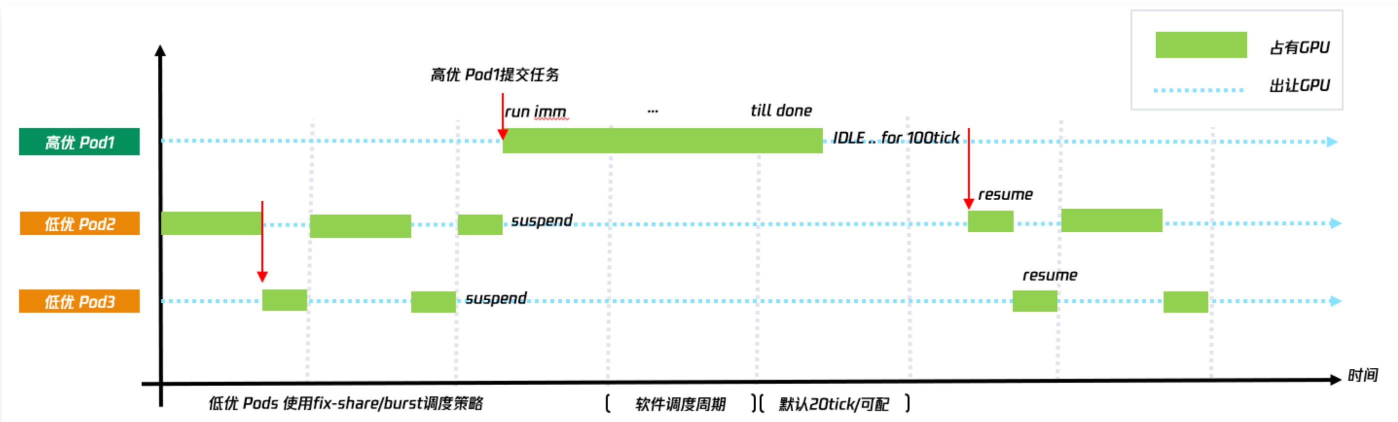
低优 Pod 在调度到节点 GPU 上后，如果 GPU 算力没有被高优 Pod 占用，低优 Pod 可以完全使用 GPU 算力。多个低优 Pod 共享 GPU 算力会受到 qGPU policy 策略控制。多个高优 Pod 之间不受具体 policy 控制，会是争抢模式。

功能2：高优 Pod 可以100%抢占低优 Pod

qGPU 离在线混部提供了一种优先级抢占能力，可以保证高优 Pod 在忙时能立刻、完全使用 GPU 算力资源，这是通过一种优先级抢占调度策略实现的。我们在 qGPU 驱动层实现了这种绝对抢占能力：

首先，qGPU 驱动可以感知高优 Pod 对 GPU 算力的需求。高优 Pod 一旦提交涉及 GPU 算力的计算任务，qGPU 驱动会在第一时间将算力全部提供给高优 Pod 使用，响应时间被控制在1ms以内。当高优 Pod 无任务运行时，驱动会在100ms后释放所占用算力，并重新分配给离线 Pod 使用。

其次，qGPU 驱动可以支持计算任务的暂停和继续。当高优 Pod 有计算任务运行时，原有占用 GPU 的低优 Pod 会立刻被暂停，将 GPU 算力让出，给高优 Pod 使用。当高优 Pod 任务结束，低优 Pod 会随即被唤醒，按照中断点继续计算。各优先级计算任务运行的时序图如下所示：



调度策略

在普通 qGPU 节点中，用户可以通过设置 policy 影响不同 Pod qGPU 在同一张卡上的调度策略。离在线混部功能中，policy 只会对低优 Pod 的调度产生影响。

- 低优 Pod
  当前高优 Pod 处于休眠状态，低优 Pod 在运行，低优 Pod 之间仍会按照 policy 策略调度。当高优 Pod 开始使用 GPU 算力，所有低优 Pod 会立刻被暂停，直到高优 Pod 计算任务结束，低优任务会重新按照 policy 策略继续运行。
- 高优 Pod
  高优 Pod 在有计算任务后会立即抢占 GPU 算力，高优 Pod 与低优 Pod 间是绝对抢占关系，不受具体 policy 影响。多个高优 Pods 之间的 GPU 算力分配是一种争抢模式，不受具体 policy 策略控制。

使用 qGPU 离在线混部

最近更新时间：2024-06-27 14:09:11

本文介绍如何使用 qGPU 离在线混部能力。

步骤1：前置准备

1. 在使用 qGPU 及 qGPU 在离线混部之前，您需要拥有一个集群。若您没有集群，请参见 创建 TKE 集群。
2. 依次执行使用 qGPU 的 步骤1：安装 qGPU 调度组件、步骤2：开启集群 qGPU 共享 和 步骤3：准备 GPU 资源。

步骤2：重建组件

> **注意：**
> 请确保完成下方重建动作后再调度应用到节点上，否则离在线混部功能无法正常工作。
> 如果您是先创建好了工作负载再创建节点，建议您在创建节点池时勾选封锁节点，完成下方重建步骤后再解除封锁，避免在重建过程中调度应用到节点上。

1. 节点创建完成后，需要重建节点上的 qgpu-manager 的 Pod 以及 qgpu-scheduler Pod 以使配置生效，命令如下：

```
$ kubectl delete pod qgpu-manager-6mcrk -n kube-system
$ kubectl delete pod -n kube-system -l app=qgpu-scheduler
```

2. 查看节点是否有 qGPU 低优算力资源，执行 `kubectl describe node` 命令，检查是否有如下资源：

```
tke.cloud.tencent.com/qgpu-core-greedy:  100
tke.cloud.tencent.com/qgpu-memory:       15
```

步骤3：配置业务

离线 Pod

通过 `tke.cloud.tencent.com/app-class: offline` 标识是一个离线 Pod，通过 `tke.cloud.tencent.com/qgpu-core-greedy` 申请离线算力，需要注意的是，离线 Pod 不支持多卡，需要同时设置低优算力与显存：

- 低优算力：tke.cloud.tencent.com/qgpu-core-greedy，单位为百分比。
- 显存：tke.cloud.tencent.com/qgpu-memory，单位为 GB。

```
apiVersion: v1
kind: Pod
metadata:
  name: xx
  annotations:
    tke.cloud.tencent.com/app-class: offline # 低优标识
spec:
  containers:
  - name: offline-container
    image: xx
    resources:
      requests:
        tke.cloud.tencent.com/qgpu-core-greedy: 10 # 离线算力
        tke.cloud.tencent.com/qgpu-memory: 5
      limits:
        tke.cloud.tencent.com/qgpu-core-greedy: 10 # 离线算力
        tke.cloud.tencent.com/qgpu-memory: 5
```

在线 Pod

通过 `tke.cloud.tencent.com/app-class: online` 标识是一个在线 Pod，不需要申请算力，只需要申请显存。

> **说明：**
> 显存是静态切分的，高优 Pod 并不会抢占低优 Pod 的显存资源。

```
apiVersion: v1
kind: Pod
annotations:
  tke.cloud.tencent.com/app-class: online
spec:
  containers:
  - name: online-container
    resources:
      requests:
        tke.cloud.tencent.com/qgpu-memory: xx
```

普通 Pod

没有 `tke.cloud.tencent.com/app-class` 这个 Annotation，普通 Pod 支持多卡。

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: common-container
    resources:
      requests:
        tke.cloud.tencent.com/qgpu-core: xx
        tke.cloud.tencent.com/qgpu-memory: xx
```


Kubernetes 对象管理

概述

最近更新时间：2024-12-12 16:31:12

对象管理说明

您可以通过控制台直接操作原生 Kubernetes 对象，例如 Deployment、DaemonSet 等。
Kubernetes 对象是集群中持久实体，用来承载集群内运行的业务。不同的 Kubernetes 对象可以表达不同的含义：

- 正在运行的应用程序
- 应用程序可用的资源
- 应用程序关联的策略等

您可以直接通过 容器服务控制台 或者 Kubernetes API 使用 Kubernetes 的对象，例如 Kubectl。

对象分类

Kubernetes 常用对象主要分为以下类型：

对象分类		对象说明	对象管理操作
工作负载	Deployment	用于管理指定调度规则的 Pod。	Deployment 管理
	StatefulSet	管理应用程序的工作负载 API 对象，且该应用程序为有状态的应用程序。	StatefulSet 管理
	DaemonSet	确保所有或部分节点上运行 Pod，例如日志采集程序。	DaemonSet 管理
	Job	一个 Job 创建一个或多个 Pod，直至运行结束。	Job 管理
	CronJob	定时运行的 Job 任务。	CronJob 管理
服务	Service	提供 Pod 访问的 Kubernetes 对象，可以根据业务需求定义不同类型。	Service 管理
	Ingress	管理集群中 Services 的外部访问的 Kubernetes 对象。	Ingress 管理
配置	ConfigMap	用于保存配置信息。	ConfigMap 管理
	Secret	用于保存敏感信息，例如密码、令牌等。	Secret 管理
存储	Volume	可以存储容器访问相关的数据。	存储管理
	Persistent Volumes（PV）	Kubernetes 集群中配置的一块存储。	
	Persistent Volumes Claim（PVC）	请求存储的声明。如果把 PV 比作 Pod，那么 PVC 相当于工作负载。	
	StorageClass	用于描述存储的类型。 创建 PVC 时，通过 StorageClass 创建指定类型的存储，即存储的模板。	

Kubernetes 对象还包括 Namespaces、HPA、Resource Quotas 等数十种，您可以根据业务需要使用不同的 Kubernetes 对象。不同版本的 Kubernetes 可使用的对象也不相同，更多说明可登录 Kubernetes 官方网站 查询。

资源限制

TKE 使用 ResourceQuota/tke-default-quota 对所有**托管集群**进行资源限制，相关限制说明请参见 K8s 资源配额说明 。

Namespaces

最近更新时间：2023-11-23 16:53:21

Namespaces 是 Kubernetes 在同一个集群中进行逻辑环境划分的对象， 您可以通过 Namespaces 进行管理多个团队多个项目的划分。在 Namespaces 下，Kubernetes 对象的名称必须唯一。您可以通过资源配额进行可用资源的分配，还可以进行不同 Namespaces 网络的访问控制。

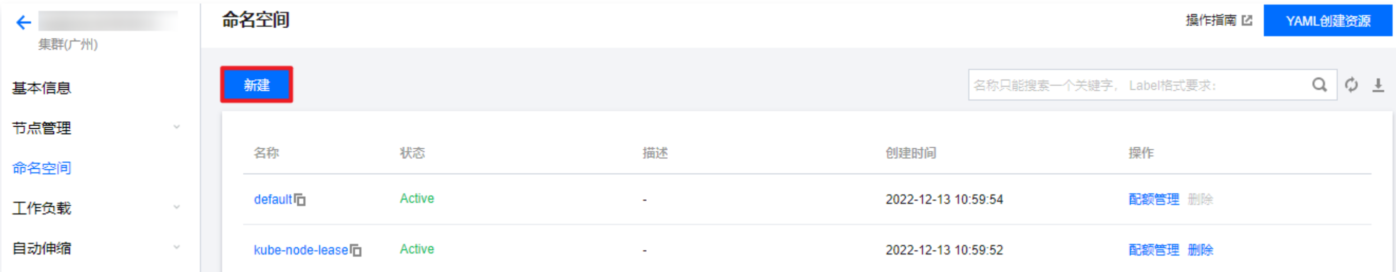
使用方法

通过控制台使用

容器服务控制台提供 Namespaces 的增删改查功能。操作步骤如下：

创建命名空间

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群列表中单击集群 ID，进入集群详情页。
3. 选择**命名空间**，在命名空间页面单击**新建**。如下图所示：



4. 在新建命名空间页面，设置命名空间名称。如下图所示：

> **说明：**
> 新建命名空间不再自动下发镜像访问凭证，可在新建命名空间后，前往 集群-配置管理-Secret 页面 创建镜像仓库密钥并下发至命名空间内。

5. 单击**创建Namespace**。完成创建后，您可以在命名空间页面进行查看。

设置资源配额和限制

1. 在命名空间页面，单击目标命名空间右侧操作列的**配额管理**。

2. 在**资源配额与限制**中，您可快速设置计算资源限制、存储资源限制和其他资源限制。为命名空间配置资源配额，如下图所示：

3. 在**编辑 LimitRange** 中，您可快速设置默认资源限制。为该命名空间下的容器设置默认资源限制，如下图所示：

> **说明**
> 对命名空间设置 CPU/内存 limit/request 配额后，创建工作负载时，必须指定 CPU/内存 limit/request，或为命名空间配置 LimitRange，更多请参见 ResourceQuotas 。

4. 编辑完成后单击**确定**即可。

删除命名空间

1. 在命名空间页面，单击目标命名空间右侧操作列的**删除**。
2. 在**删除资源**弹窗中，确认资源信息，单击**确定**可删除对应的命名空间，且该命名空间下的资源对象也会被删除。

通过 Kubectl 使用

通过 Kubectl 使用：更多详情可参见 Kubernetes 官网文档 。

通过 ResourceQuota 设置 Namespaces 资源的使用配额

一个命名空间下可以拥有多个 ResourceQuota 资源，每个 ResourceQuota 可以设置每个 Namespace 资源的使用约束。可以设置 Namespaces 资源的使用约束如下：

- 计算资源的配额，例如 CPU、内存。
- 存储资源的配额，例如请求存储的总存储。
- Kubernetes 对象的计数，例如 Deployment 个数配额。

不同的 Kubernetes 版本，ResourceQuota 支持的配额设置略有差异，更多详情可参见 Kubernetes ResourceQuota 官方文档 。
ResourceQuota 的示例如下所示：

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: default
spec:
  hard:
    configmaps: "10"  ## 最多10个 ConfigMap
    replicationcontrollers: "20" ## 最多20个 replicationcontroller
    secrets: "10" ## 最多10个 secret
    services: "10" ## 最多10个 service
    services.loadbalancers: "2"  ## 最多2个 Loadbanlacer 模式的 service
    cpu: "1000" ## 该 Namespaces 下最多使用1000个 CPU 的资源
    memory: 200Gi ## 该 Namespaces 下最多使用200Gi的内存
```

通过 NetWorkPolicy 设置 Namespaces 网络的访问控制

Network Policy 是 K8s 提供的一种资源，用于定义基于 Pod 的网络隔离策略。不仅可以限制 Namespaces， 还可以控制 Pod 与 Pod 之间的网络访问控制，即控制一组 Pod 是否可以与其它组 Pod，以及其它 network endpoints 进行通信。
在集群内部署 NetworkPolicy Controller，并通过 NetworkPolicy 实现 Namespaces 之间的网络控制的操作详情可参见 使用 Network Policy 进行网络访问控制 。

工作负载
Deployment 管理

最近更新时间：2024-07-30 17:06:21

简介

Deployment 声明了 Pod 的模板和控制 Pod 的运行策略，适用于部署无状态的应用程序。您可以根据业务需求，对 Deployment 中运行的 Pod 的副本数、调度策略、更新策略等进行声明。

Deployment 控制台操作指引

创建 Deployment

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 单击需要创建 Deployment 的集群 ID，进入集群的基本信息页面。
3. 在左侧导航栏选择**工作负载** > **Deployment**。
4. 单击**新建**，进入**新建 Deployment** 页面。根据实际需求，设置 Deployment 参数。关键参数信息如下：
    - **名称**：自定义工作负载名称。
    - **命名空间**：根据实际需求进行选择。
    - **Labels**：一个键 - 值对（Key-Value），用于对资源进行分类管理。
    - **数据卷（选填）**：为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储 NFS、配置文件、PVC，还需挂载到容器的指定路径中。
    - **实例内容器**：根据实际需求，为 Deployment 的一个 Pod 设置一个或多个不同的容器。
        - **名称**：自定义。
        - **镜像**：根据实际需求进行选择。
        - **镜像版本（Tag）**：根据实际需求进行填写。
        - **镜像拉取策略**：提供以下3种策略，请按需选择。
          若不设置镜像拉取策略，当镜像版本为空或 `latest` 时，使用 Always 策略，否则使用 IfNotPresent 策略。
            - **Always**：总是从远程拉取该镜像。
            - **IfNotPresent**：默认使用本地镜像，若本地无该镜像则远程拉取该镜像。
            - **Never**：只使用本地镜像，若本地没有该镜像将报异常。
        - **环境变量**：设置容器中的变量。
        - **CPU/内存限制**：可根据 Kubernetes 资源限制 进行设置 CPU 和内存的限制范围，提高业务的健壮性。
        - **GPU 资源**：配置该工作负载使用的最少 GPU 资源。
        - **高级设置**：可设置**工作目录**、**运行命令**、**运行参数**、**容器健康检查**和**特权级**等参数。
    - **实例数量**：根据实际需求选择调节方式，设置实例数量。
        - **手动调节**：设定实例数量，可单击“+”或“-”控制实例数量。
        - **自动调节**：满足任一设定条件，则自动调节实例（pod）数目。详情请参见 自动伸缩。
    - **镜像访问凭证**：默认无需配置访问凭证，工作负载会基于镜像地址中的域名，自动匹配命名空间下已有的访问凭证。也支持手动指定一个或多个访问凭证，以及新建访问凭证。
5. 单击**创建 Deployment**，完成创建。如下图所示：
   当运行数量 = 期望数量时，即表示 Deployment 下的所有 Pod 已创建完成。

更新 Deployment

更新 YAML

1. 登录容器服务控制台，选择左侧导航栏中的 集群。

2. 单击集群 ID，进入集群的基本信息页面。
3. 在左侧导航栏选择**工作负载** > **Deployment**，在需要更新 YAML 的 Deployment 行中，单击**更多** > **编辑 yaml**。
4. 在**编辑 Yaml** 页面中进行编辑，单击**完成**。

更新 Pod 配置

1. 在集群管理页面，单击集群 ID，进入集群的基本信息页面。
2. 在左侧导航栏选择**工作负载** > **Deployment**，在需要更新 Pod 配置的 Deployment 行中，单击**更新 Pod 配置**。如下图所示：

3. 在**更新 Pod 配置**页面，根据实际需求修改更新方式，设置参数。
4. 单击**更新 Pod 配置**即可。

回滚 Deployment

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击集群 ID，进入集群的基本信息页面。
3. 在左侧导航栏选择**工作负载** > **Deployment**，单击需要回滚的 Deployment 名称，进入 Deployment 信息页面。
4. 选择**修订历史**页签，在需要回滚的版本行中，单击**回滚**。如下图所示：

5. 在弹出的**回滚资源**提示框中，单击**确定**即可完成回滚。

调整 Pod 数量

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击集群 ID，进入集群的基本信息页面。
3. 在左侧导航栏选择**工作负载** > **Deployment**，在需要调整 Pod 数量的 Deployment 行中，单击**更新 Pod 数量**。如下图所示：

4. 在**更新 Pod 数量**页面，根据实际需求调整 Pod 数量，单击**更新实例数量**即可完成调整。

Kubectl 操作 Deployment 指引

YAML 示例

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
```

```
  labels:
    app: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-deployment
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

说明：

- **kind**：标识 Deployment 资源类型。
- **metadata**：Deployment 的名称、Namespace、Label 等基本信息。
- **metadata.annotations**：对 Deployment 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- **spec.replicas**：Deployment 管理的 Pod 数量。
- **spec.selector**：Deployment 管理 Selector 选中的 Pod 的 Label。
- **spec.template**：Deployment 管理的 Pod 的详细模板配置。

更多参数详情可查看 Kubernetes Deployment 官方文档 。

Kubectl 创建 Deployment

1. 参考 YAML 示例，准备 Deployment YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参见 通过 Kubectl 连接集群 。
3. 执行以下命令，使用准备好的 YAML 文件创建 Deployment：

```
kubectl create -f Deployment YAML 文件名称
```

   例如，使用一个文件名为 `nginx.yaml` 的 Deployment YAML 文件，则执行以下命令：

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

4. 执行以下命令，验证 Deployment 是否创建成功：

```
kubectl get deployments
```

   返回类似以下信息，即表示创建成功。

```
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
first-workload   1         1         1            0           6h
ng               1         1         1            1           42m
```

Kubectl 更新 Deployment

通过 Kubectl 更新 Deployment 有以下三种方法：

> **说明：**
> 方法一和方法二均支持 **Recreate** 和 **RollingUpdate** 两种更新策略。
> - Recreate 更新策略为先销毁全部 Pod，再重新创建 Deployment。

- RollingUpdate 更新策略为滚动更新策略，逐个更新 Deployment 的 Pod。RollingUpdate 还支持暂停、设置更新时间间隔等。

方法一

执行以下命令，更新 Deployment。

```
kubectl edit  deployment/[name]
```

此方法适用于简单的调试验证，不建议在生产环境中直接使用。您可以通过此方法更新任意的 Deployment 参数。

方法二

执行以下命令，更新指定容器的镜像。

```
kubectl set image deployment/[name] [containerName]=[image:tag]
```

建议保持 Deployment 的其他参数不变，业务更新时，仅更新容器镜像。

方法三

执行以下命令，滚动更新指定资源。

```
kubectl rolling-update [NAME] -f FILE
```

Kubectl 回滚 Deployment

1. 执行以下命令，查看 Deployment 的更新历史。
```
kubectl rollout history deployment/[name]
```
2. 执行以下命令，查看指定版本详情。
```
kubectl rollout history deployment/[name] --revision=[REVISION]
```
3. 执行以下命令，回滚到前一个版本。
```
kubectl rollout undo deployment/[name]
```
如需指定回滚版本号，可执行以下命令。
```
kubectl rollout undo deployment/[name] --to-revision=[REVISION]
```

Kubectl 调整 Pod 数量

手动更新 Pod 数量

执行以下命令，手动更新 Pod 数量。

```
kubectl scale deployment [NAME] --replicas=[NUMBER]
```

自动更新 Pod 数量

前提条件
开启集群中的 HPA 功能。您在容器服务 TKE 中创建的集群默认开启 HPA 功能。
操作步骤
执行以下命令，设置 Deployment 的自动扩缩容。

```
kubectl autoscale deployment [NAME] --min=10 --max=15 --cpu-percent=80
```

Kubectl 删除 Deployment

执行以下命令，删除 Deployment。

```
kubectl delete deployment [NAME]
```

StatefulSet 管理

最近更新时间：2024-11-25 16:42:22

简介

StatefulSet 主要用于管理有状态的应用，可以创建具有持久性标识符的 Pod。Pod 迁移或销毁重启后，标识符仍会保留。在需要持久化存储时，您可以通过标识符对存储卷进行一一对应。如果应用程序不需要持久的标识符，建议您使用 Deployment 部署应用程序。

StatefulSet 控制台操作指引

创建 StatefulSet

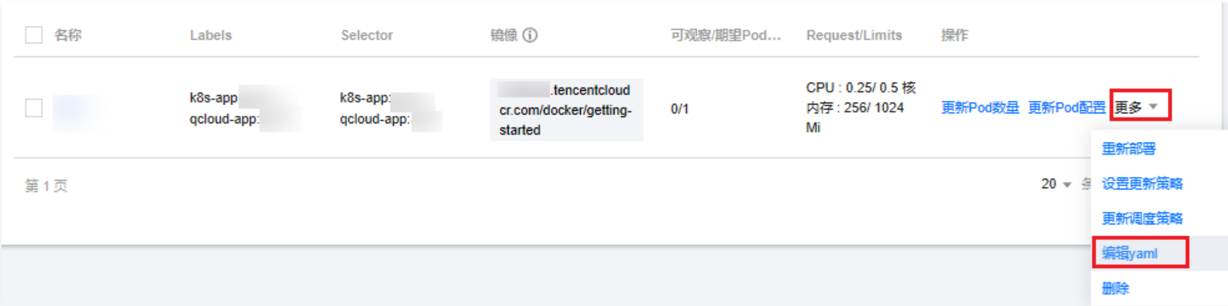
1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击需要创建 StatefulSet 的集群 ID，进入待创建 StatefulSet 的集群管理页面。
3. 选择**工作负载** > **StatefulSet**，进入 StatefulSet 管理页面。如下图所示：

4. 单击**新建**，在**新建工作负载**页面，根据实际需求，设置 StatefulSet 参数。关键参数信息如下：
    - **名称**：自定义工作负载名称。
    - **命名空间**：根据实际需求选择。
    - **Labels**：一个键值对（Key-Value），用于资源分类管理。
    - **数据卷（选填）**：为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储 NFS、配置文件、PVC，还需挂载到容器的指定路径中。
    - **实例内容器**：根据实际需求，为 StatefulSet 的一个 Pod 设置一个或多个不同的容器。
        - **名称**：自定义。
        - **镜像**：根据实际需求进行选择。
        - **镜像版本（Tag）**：根据实际需求进行填写。
        - **镜像拉取策略**：提供以下几种策略，请按需选择。若不设置镜像拉取策略，当镜像版本为空或 `latest` 时，使用 Always 策略，否则使用 IfNotPresent 策略。
            - **Always**：总是从远程拉取该镜像。
            - **IfNotPresent**：默认使用本地镜像，若本地无该镜像则远程拉取该镜像。
            - **Never**：只使用本地镜像，若本地没有该镜像将报异常。
        - **环境变量**：设置容器中的变量。
        - **CPU/内存限制**：可根据 Kubernetes 资源限制 进行设置 CPU 和内存的限制范围，提高业务的健壮性。
        - **GPU 资源**：配置该工作负载使用的最少 GPU 资源。
        - **高级设置**：可设置**工作目录**、**运行命令**、**运行参数**、**容器健康检查**和**特权级**等参数。
    - **实例数量**：根据实际需求选择调节方式，设置实例数量。
        - **手动调节**：设定实例数量，可单击“+”或“-”控制实例数量。
        - **自动调节**：满足任一设定条件，则自动调节实例（pod）数目。详情请参见 自动伸缩 。
    - **镜像访问凭证**：默认无需配置访问凭证，工作负载会基于镜像地址中的域名，自动匹配命名空间下已有的访问凭证。也支持手动指定一个或多个访问凭证，以及新建访问凭证。
    - **节点调度策略**：可根据调度规则，将 Pod 调度到符合预期的 Label 的节点中。
    - **访问设置**：根据实际需求，设置 Service 参数。详情见 服务访问方式 。
5. 单击**创建 StatefulSet**，完成创建。

更新 StatefulSet

更新 YAML

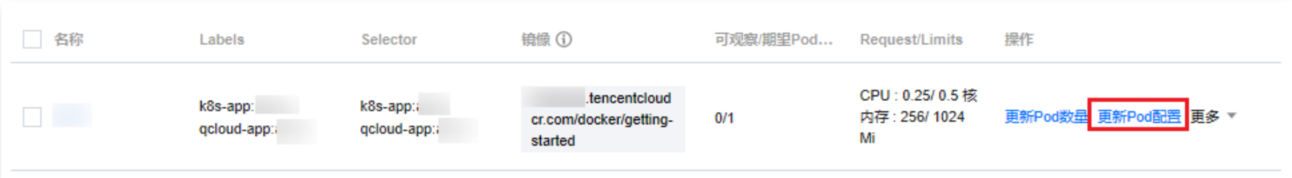
1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击集群 ID，进入集群的基本信息页面。
3. 选择**工作负载** > **StatefulSet**，在需要更新 YAML 的 StatefulSet 行中，选择**更多** > **编辑 yaml**。如下图所示：



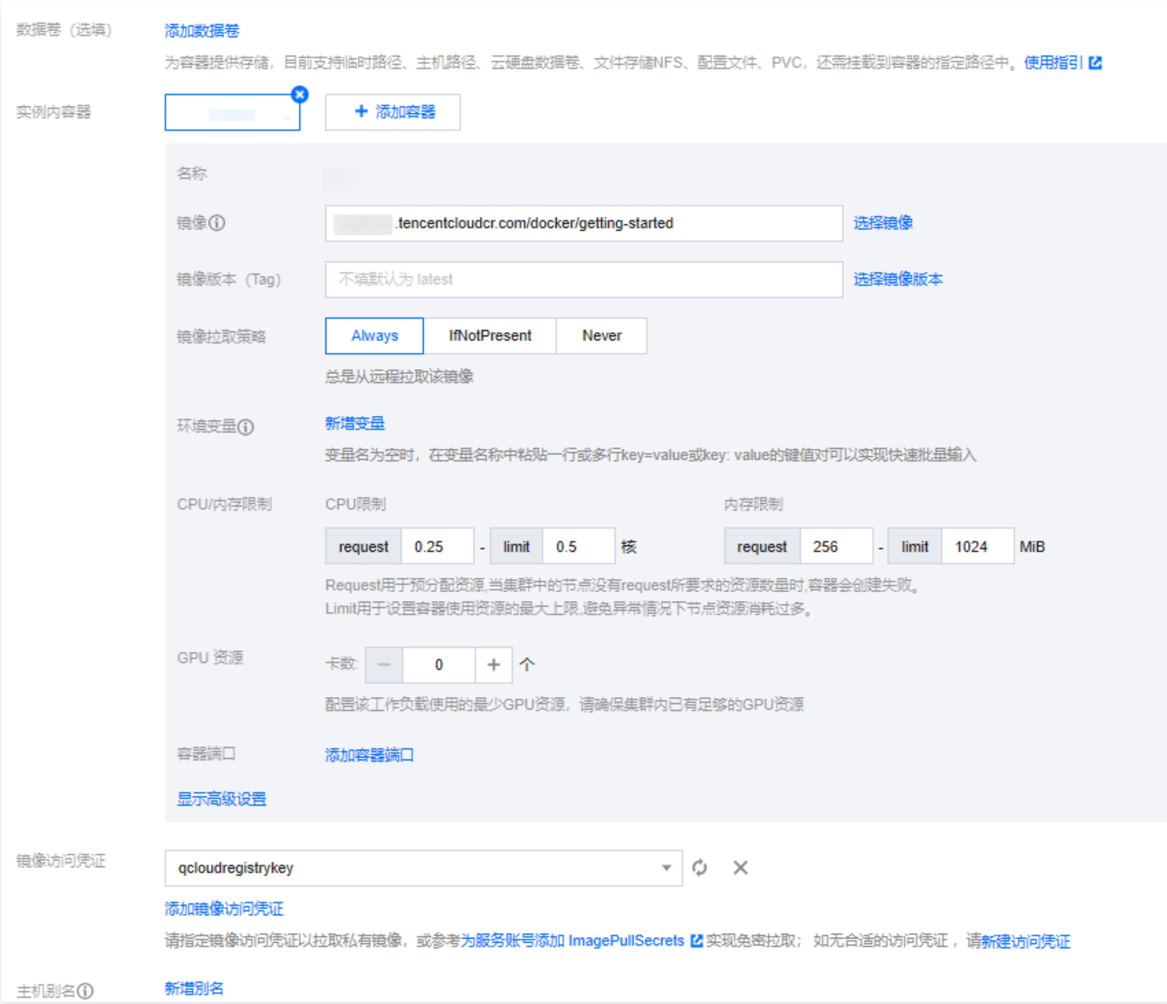
4. 在**编辑 Yaml** 页面中进行编辑，单击**完成**。

更新 Pod 配置

1. 在集群管理页面，单击集群 ID，进入集群的基本信息页面。
2. 选择**工作负载** > **StatefulSet**，在需要更新 Pod 配置的 StatefulSet 行中，单击**更新 Pod 配置**。如下图所示：



3. 在**更新 Pod 配置**页面，根据实际需求修改更新方式，设置参数。如下图所示：



4. 单击**完成**，即可更新 Pod 配置。

Kubectl 操作 StatefulSet 指引

YAML 示例

```
apiVersion: v1
kind: Service  ## 创建一个 Headless Service，用于控制网络域
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet ### 创建一个 Nginx的StatefulSet
metadata:
  name: web
  namespace: default
spec:
  selector:
```

```
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "cbs"
      resources:
        requests:
          storage: 10Gi
```

- **kind**：标识 StatefulSet 资源类型。
- **metadata**：StatefulSet 的名称、Label 等基本信息。
- **metadata.annotations**：对 StatefulSet 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- **spec.template**：StatefulSet 管理的 Pod 的详细模板配置。
- **spec.volumeClaimTemplates**：提供创建 PVC&PV 的模板。

更多参数详情可查看 Kubernetes StatefulSet 官方文档 。

创建 StatefulSet

1. 参考 YAML 示例，准备 StatefulSet YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参考 通过 Kubectl 连接集群 。
3. 执行以下命令创建 StatefulSet：

   ```
   kubectl create -f <StatefulSet YAML 文件名称>
   ```

   例如，创建一个文件名为 web.yaml 的 StatefulSet YAML 文件，则执行以下命令：

   ```
   kubectl create -f web.yaml
   ```

4. 执行以下命令，验证创建是否成功。

   ```
   kubectl get statefulset
   ```

   返回类似以下信息，即表示创建成功。

   ```
   NAME    READY   AGE
   web     2/2     37s
   ```

更新 StatefulSet

执行以下命令，查看 StatefulSet 的更新策略类型。

```
kubectl get statefulset/<statefulset-name> -o go-template='{{.spec.updateStrategy.type}}{{"\n"}}'
```

StatefulSet 有以下两种更新策略类型：
- OnDelete：默认更新策略。该更新策略在更新 StatefulSet 后，需手动删除旧的 StatefulSet Pod 才会创建新的 StatefulSet Pod。
- RollingUpdate：支持 Kubernetes 1.7或更高版本。该更新策略在更新 StatefulSet 模板后，旧的 StatefulSet Pod 将被终止，并且以滚动更新方式创建新的 StatefulSet Pod（Kubernetes 1.7或更高版本）。

方法一

执行以下命令，更新 StatefulSet。

```
kubectl edit statefulset/<name>
```

此方法适用于简单的调试验证，不建议在生产环境中直接使用。您可以通过此方法更新任意的 StatefulSet 参数。

方法二

执行以下命令，更新指定容器的镜像。

```
kubectl patch statefulset <NAME> --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/image", "value":"<newImage>"}]'
```

建议保持 StatefulSet 的其他参数不变，业务更新时，仅更新容器镜像。如果更新的 StatefulSet 是滚动更新方式的策略，可执行以下命令查看更新状态：

```
kubectl rollout status sts/<StatefulSet-name>
```

删除 StatefulSet

执行以下命令，删除 StatefulSet。

```
kubectl delete statefulset <name> --cascade=false
```

`--cascade=false` 参数表示 Kubernetes 仅删除 StatefulSet，且不删除任何 Pod。如需删除 Pod，则执行以下命令：

```
kubectl delete statefulset <name>
```

更多 StatefulSet 相关操作可查看 Kubernetes官方指引。

DaemonSet 管理

最近更新时间：2023-11-28 14:52:32

简介

DaemonSet 主要用于部署常驻集群内的后台程序，例如节点的日志采集。DaemonSet 保证在所有或部分节点上均运行指定的 Pod。 新节点添加到集群内时，也会有自动部署 Pod；节点被移除集群后，Pod 将自动回收。

调度说明

若配置了 Pod 的 nodeSelector 或 affinity 参数，DaemonSet 管理的 Pod 将按照指定的调度规则调度。若未配置 Pod 的 nodeSelector 或 affinity 参数，则将在所有的节点上部署 Pod。

DaemonSet 控制台操作指引

创建 DaemonSet

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **DaemonSet** 页面，单击**新建**。如下图所示：

4. 在**新建 DaemonSet** 页面，根据实际需求，设置 DaemonSet 参数。关键参数信息如下：
    - **名称**：输入自定义名称。
    - **命名空间**：根据实际需求进行选择。
    - **Labels**：一个键 - 值对（Key-Value），用于对资源进行分类管理。
    - **数据卷（选填）**：为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储 NFS、配置文件、PVC，还需挂载到容器的指定路径中。
    - **实例内容器**：根据实际需求，为 DaemonSet 的一个 Pod 设置一个或多个不同的容器。
        - **名称**：自定义。
        - **镜像**：根据实际需求进行选择。
        - **镜像版本（Tag）**：根据实际需求进行填写。
        - **镜像拉取策略**：提供以下3种策略，请按需选择。
          若不设置镜像拉取策略，当镜像版本为空或 `latest` 时，使用 Always 策略，否则使用 IfNotPresent 策略。
            - **Always**：总是从远程拉取该镜像。
            - **IfNotPresent**：默认使用本地镜像，若本地无该镜像则远程拉取该镜像。
            - **Never**：只使用本地镜像，若本地没有该镜像将报异常。
        - **CPU/内存限制**：可根据 Kubernetes 资源限制 进行设置 CPU 和内存的限制范围，提高业务的健壮性。
        - **GPU 资源**：配置该工作负载使用的最少 GPU 资源。
        - 高级设置：可设置 **“工作目录”**，**“运行命令”**，**“运行参数”**，**“容器健康检查”**，**“特权级”**等参数。
    - **镜像访问凭证**：容器镜像默认私有，在创建工作负载时，需选择实例对应的镜像访问凭证。
    - **节点调度策略**：可根据调度规则，将 Pod 调度到符合预期的 Label 的节点中。
5. 单击**创建 DaemonSet**，完成创建。

更新 DaemonSet

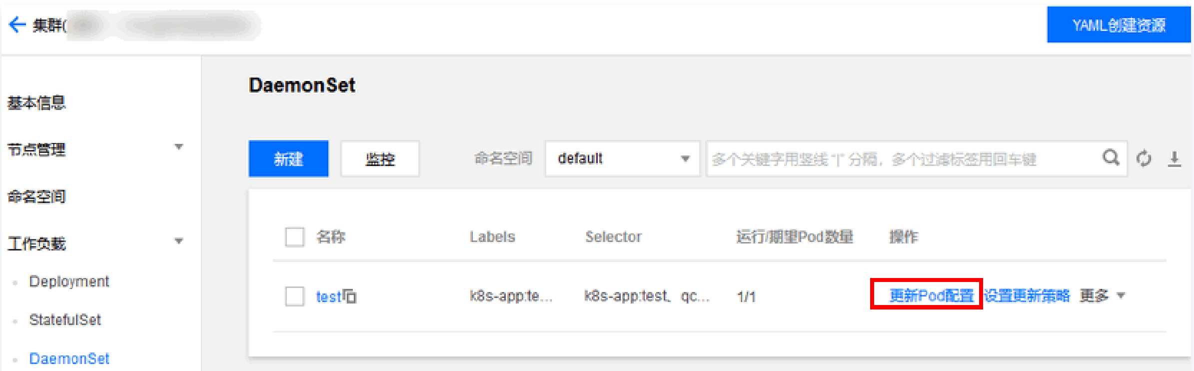
更新 YAML

1. 登录容器服务控制台，选择左侧导航栏中的 **集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **DaemonSet** 页面，选择实例右侧的**更多** > **编辑 yaml**。
4. 在编辑 Yaml 页面，更新 **yaml** 后单击**完成**。

更新 Pod 配置

> **说明**
> 仅在 Kubernetes 1.6或更高版本中支持 DaemonSet 滚动更新功能。

1. 登录容器服务控制台，选择左侧导航栏中的 **集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **DaemonSet** 页面，选择实例右侧的**更新 Pod 配置**。如下图所示：



4. 在**更新 Pod 配置**页面，根据实际需求修改更新方式，设置参数。
5. 单击**完成**，即可更新 Pod 配置。

Kubectl 操作 DaemonSet 指引

YAML 示例

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: k8s.gcr.io/fluentd-elasticsearch:1.20
        resources:
          limits:
```



```
          memory: 200Mi
        requests:
          cpu: 100m
          memory: 200Mi
      volumeMounts:
      - name: varlog
        mountPath: /var/log
      - name: varlibdockercontainers
        mountPath: /var/lib/docker/containers
        readOnly: true
    terminationGracePeriodSeconds: 30
    volumes:
    - name: varlog
      hostPath:
        path: /var/log
    - name: varlibdockercontainers
      hostPath:
        path: /var/lib/docker/containers
```

> **注意**
> 以上 YAML 示例引用于 Kubernetes Documentation DaemonSet，创建时可能存在容器镜像拉取失败的情况，仅用于本文介绍 DaemonSet 的组成。

- **kind**：标识 DaemonSet 资源类型。
- **metadata**：DaemonSet 的名称、Label 等基本信息。
- **metadata.annotations**：DaemonSet 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- **spec.template**：DaemonSet 管理的 Pod 的详细模板配置。

更多可查看 Kubernetes DaemonSet 官方文档 。

Kubectl 创建 DaemonSet

1. 参考 YAML 示例，准备 DaemonSet YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参考 通过 Kubectl 连接集群 。
3. 执行以下命令，创建 DaemonSet YAML 文件。

```
kubectl create -f [DaemonSet YAML 文件名称]
```

例如，创建一个文件名为 fluentd-elasticsearch.yaml 的 DaemonSet YAML 文件，则执行以下命令：

```
kubectl create -f fluentd-elasticsearch.yaml
```

4. 执行以下命令，验证创建是否成功。

```
kubectl get DaemonSet
```

返回类似以下信息，即表示创建成功。

```
NAME      DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR       AGE
frontend  0         0         0         0            0           app=frontend-node   16d
```

Kubectl 更新 DaemonSet

执行以下命令，查看 DaemonSet 的更新策略类型。

```
kubectl get ds/<daemonset-name> -o go-template='{{.spec.updateStrategy.type}}{{"\n"}}'
```

DaemonSet 有以下两种更新策略类型：

- OnDelete：默认更新策略。该更新策略在更新 DaemonSet 后，需手动删除旧的 DaemonSet Pod 才会创建新的 DaemonSet Pod。
- RollingUpdate：支持 Kubernetes 1.6或更高版本。该更新策略在更新 DaemonSet 模板后，旧的 DaemonSet Pod 将被终止，并且以滚动更新方式创建新的 DaemonSet Pod。

方法一

执行以下命令，更新 DaemonSet。

```
kubectl edit DaemonSet/[name]
```

此方法适用于简单的调试验证，不建议在生产环境中直接使用。您可以通过此方法更新任意的 DaemonSet 参数。

方法二

执行以下命令，更新指定容器的镜像。

```
kubectl set image ds/[daemonset-name] [container-name]=[container-new-image]
```

建议保持 DaemonSet 的其他参数不变，业务更新时，仅更新容器镜像。

Kubectl 回滚 DaemonSet

1. 执行以下命令，查看 DaemonSet 的更新历史。
```
kubectl rollout history daemonset/[name]
```
2. 执行以下命令，查看指定版本详情。
```
kubectl rollout history daemonset/[name] --revision=[REVISION]
```
3. 执行以下命令，回滚到前一个版本。
```
kubectl rollout undo daemonset/[name]
```
如需指定回滚版本号，可执行以下命令。
```
kubectl rollout undo daemonset/[name] --to-revision=[REVISION]
```

Kubectl 删除 DaemonSet

执行以下命令，删除 DaemonSet。

```
kubectl delete DaemonSet [NAME]
```

CronJob 管理

最近更新时间：2024-10-29 16:09:22

简介

一个 CronJob 对象类似于 crontab（cron table）文件中的一行，它根据指定的预定计划周期性地运行一个 Job。

CronJob 控制台操作指引

创建 CronJob

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **CronJob** 页面，单击**新建**。
4. 在**新建工作负载**页面，根据实际需求，设置 CronJob 参数。关键参数信息如下：
    - **名称**：自定义。
    - **命名空间**：根据实际需求进行选择。
    - **Labels**：一个键值对（Key-Value），用于对资源进行分类管理。
    - **定时规则**：根据业务需求选择任务的定期执行策略。
    - **保留成功 Job 数**：对应 `.spec.successfulJobsHistoryLimit`，详情见 Jobs History Limits。
    - **保留失败 Job 数**：对应 `.spec.failedJobsHistoryLimit`，详情见 Jobs History Limits。
    - **Job 设置**：
        - **重复次数**：Job 管理的 Pod 需要重复执行的次数。
        - **并行度**：Job 并行执行的 Pod 数量。
        - **失败重启策略**：Pod下容器异常退出后的重启策略。
            - Never：不重启容器，直至 Pod 下所有容器退出。
            - OnFailure：Pod 继续运行，容器将重新启动。
    - **数据卷（选填）**：为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储 NFS、配置文件、PVC，还需挂载到容器的指定路径中。
    - **实例内容器**：根据实际需求，为 CronJob 的一个 Pod 设置一个或多个不同的容器。
        - **名称**：自定义。
        - **镜像**：根据实际需求进行选择。
        - **镜像版本**：根据实际需求进行填写。
        - **镜像拉取策略**：提供以下3种策略，请按需选择。若不设置镜像拉取策略，当镜像版本为空或 `latest` 时，使用 Always 策略，否则使用 IfNotPresent 策略。
            - **Always**：总是从远程拉取该镜像。
            - **IfNotPresent**：默认使用本地镜像，若本地无该镜像则远程拉取该镜像。
            - **Never**：只使用本地镜像，若本地没有该镜像将报异常。
        - **CPU/内存限制**：可根据 Kubernetes 资源限制 进行设置 CPU 和内存的限制范围，提高业务的健壮性。
        - **GPU 资源**：配置该工作负载使用的最少 GPU 资源。
        - **高级设置**：可设置**工作目录**、**运行命令**、**运行参数**、**容器健康检查**和**特权级**等参数。
    - **镜像访问凭证**：容器镜像默认私有，在创建工作负载时，需选择实例对应的镜像访问凭证。
    - **节点调度策略**：可根据调度规则，将 Pod 调度到符合预期的 Label 的节点中。
5. 单击**创建 CronJob**，完成创建。

查看 CronJob 状态

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **CronJob** 页面，单击需要查看状态的 CronJob 名称，即可查看 CronJob 详情。

Kubectl 操作 CronJob 指引

YAML 示例

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

- kind：标识 CronJob 资源类型。
- metadata：CronJob 的名称、Label等基本信息。
- metadata.annotations：对 CronJob 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- spec.schedule：CronJob 执行的 Cron 的策略。
- spec.jobTemplate：Cron 执行的 Job 模板。

创建 CronJob

方法一

1. 参考 YAML 示例，准备 CronJob YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参考 通过 Kubectl 连接集群 。
3. 执行以下命令，创建 CronJob YAML 文件。

```
kubectl create -f <CronJob YAML 文件名称>
```

例如，创建一个文件名为 cronjob.yaml 的 CronJob YAML 文件，则执行以下命令：

```
kubectl create -f cronjob.yaml
```

方法二

1. 通过执行 `kubectl run` 命令，快速创建一个 CronJob。
例如，快速创建一个不需要写完整配置信息的 CronJob，则执行以下命令：

```
kubectl run hello --schedule="*/1 * * * *" --restart=OnFailure --image=busybox -- /bin/sh -c "date;
echo Hello"
```

2. 执行以下命令，验证创建是否成功。

```
kubectl get cronjob [NAME]
```

返回类似以下信息，即表示创建成功。

```
NAME      SCHEDULE     SUSPEND   ACTIVE    LAST SCHEDULE   AGE
```

```
cronjob   * * * * *   False     0         <none>          15s
```

删除 CronJob

注意：

- 执行此删除命令前，请确认是否存在正在创建的 Job，否则执行该命令将终止正在创建的 Job。
- 执行此删除命令时，已创建的 Job 和已完成的 Job 均不会被终止或删除。
- 如需删除 CronJob 创建的 Job，请手动删除。

执行以下命令，删除 CronJob。

```
kubectl delete cronjob [NAME]
```

Job 管理

最近更新时间：2024-05-11 14:29:11

简介

Job 控制器会创建 1-N 个 Pod，这些 Pod 按照运行规则运行，直至运行结束。Job 可用于批量计算、数据分析等场景。通过设置重复执行次数、并行度、重启策略等满足业务诉求。
Job 执行完成后，不再创建新的 Pod，也不会删除 Pod，您可在 “日志” 中查看已完成的 Pod 的日志。如果您删除了 Job，Job 创建的 Pod 也会同时被删除，将无法查看该 Job 创建的 Pod 的日志。

Job 控制台操作指引

创建 Job

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **Job** 页面，单击**新建**。如下图所示：

4. 在**新建 Job** 页面，根据实际需求，设置 Job 参数。关键参数信息如下：
    - **名称**：自定义。
    - **命名空间**：根据实际需求进行选择。
    - **Labels**：一个键 - 值对（Key-Value），用于对资源进行分类管理。
    - **Job 设置**：根据实际需求，为 Job 的一个 Pod 设置一个或多个不同的容器。
        - **重复次数**：设置 Job 管理的 Pod 需要重复执行的次数。
        - **并行度**：设置 Job 并行执行的 Pod 数量。
        - **失败重启策略**：设置 Pod 下容器异常退出后的重启策略。
            - **选择 Never**：不重启容器，直至 Pod 下所有容器退出。
            - **选择 OnFailure**：Pod 继续运行，容器将重新启动。
    - **可抢占**：可抢占类型的 Job 类工作负载，不会占用节点实际的可调度资源，但在发生资源竞争时，资源会被优先回收，建议集群中低优先级的业务开启该功能。当前功能仅在原生节点上使用，若原生节点的 Extended Resource 资源不足，会引发 Pod Pending，详情请参加 可抢占式 Job。
    - **数据卷（选填）**：为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储 NFS、配置文件、PVC，还需挂载到容器的指定路径中。

    - **实例内容器**：根据实际需求，为 Job 的一个 Pod 设置一个或多个不同的容器。
        - **名称**：自定义。
        - **镜像**：根据实际需求进行选择。
        - **镜像版本（Tag）**：根据实际需求进行填写。
        - **镜像拉取策略**：提供以下3种策略，请按需选择。若不设置镜像拉取策略，当镜像版本为空或 `latest` 时，使用 Always 策略，否则使用 IfNotPresent 策略。
            - **Always**：总是从远程拉取该镜像。
            - **IfNotPresent**：默认使用本地镜像，若本地无该镜像则远程拉取该镜像。
            - **Never**：只使用本地镜像，若本地没有该镜像将报异常。
        - **CPU/内存限制**：可根据 Kubernetes 资源限制 进行设置 CPU 和内存的限制范围，提高业务的健壮性。
        - **GPU 资源**：配置该工作负载使用的最少 GPU 资源。

    - **高级设置**：可设置 “工作目录”， “运行命令”， “运行参数”， “容器健康检查”， “特权级”等参数。
  - **镜像访问凭证**：容器镜像默认私有，在创建工作负载时，需选择实例对应的镜像访问凭证。
  - **节点调度策略**：可根据调度规则，将 Pod 调度到符合预期的 Label 的节点中。
5. 单击**创建 Job**，完成创建。

查看 Job 状态

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载** > **Job** 页面，选择需要查看状态的 Job 名称，即可查看 Job 详情。

删除 Job

Job 执行完成后，不再创建新的 Pod，也不会删除 Pod，您可在日志中查看已完成的 Pod 的日志。如果您删除了 Job，Job 创建的 Pod 也会同时被删除，将查看不到该 Job 创建的 Pod 的日志。

Kubectl 操作 Job 指引

YAML 示例

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  completions: 2
  parallelism: 2
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

说明：

- kind：标识 Job 资源类型。
- metadata：Job 的名称、Label 等基本信息。
- metadata.annotations：Job 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- spec.completions：Job 管理的 Pod 重复执行次数。
- spec.parallelism：Job 并行执行的 Pod 数。
- spec.template：Job 管理的 Pod 的详细模板配置。

创建 Job

1. 参考 YAML 示例，准备 Job YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参见 通过 Kubectl 连接集群 。
3. 创建 Job YAML 文件。

   ```
   kubectl create -f Job YAML 文件名称
   ```

   例如，创建一个文件名为 pi.yaml 的 Job YAML 文件，则执行以下命令：

   ```
   kubectl create -f pi.yaml
   ```

4. 执行以下命令，验证创建是否成功。

```
kubectl get job
```

返回类似以下信息，即表示创建成功。

```
[root@VM-0-7-centos ~]# kubectl get job
NAME          COMPLETIONS   DURATION   AGE
conjob-       0/1           177m       177m
jobtest       0/1           2m39s      2m39s
pi            0/2           5s         5s
```

删除 Job

执行以下命令，删除 Job。

```
kubectl delete job [NAME]
```

设置工作负载的资源限制

最近更新时间：2023-05-17 15:40:28

请求（Request）与限制（Limit）

Request：容器使用的最小资源需求，作为容器调度时资源分配的判断依赖。只有当节点上可分配资源量 >= 容器资源请求数时才允许将容器调度到该节点。但 Request 参数不限制容器的最大可使用资源值。
Limit： 容器能使用的资源最大值。
更多 **Limit** 和 **Request** 参数介绍，单击 查看详情 。

CPU 限制说明

CPU 资源允许设置 CPU 请求和 CPU 限制的资源量，以核（U）为单位，允许为小数。

> **注意**
> - CPU Request 作为调度时的依据，在创建时为该容器在节点上分配 CPU 使用资源，称为 “已分配 CPU” 资源。
> - CPU Limit 限制容器 CPU 资源的上限，不设置表示不做限制（CPU Limit >= CPU Request）。

内存限制说明

内存资源只允许限制容器最大可使用内存量。以 MiB 为单位，允许为小数。

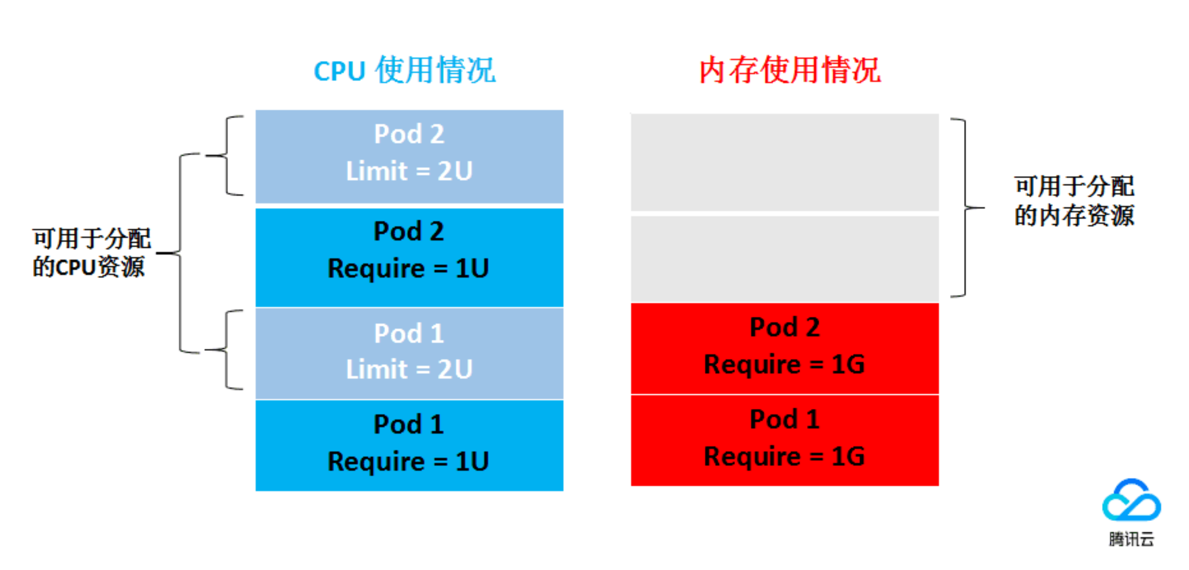
> **注意**
> - 内存 Request 作为调度时的依据，在创建时为该容器在节点上分配内存，称为 “已分配内存” 资源。
> - 内存资源为不可伸缩资源。当节点上所有容器使用内存均超量时，存在 OOM（Out Of Memory，即内存溢出）的风险。不设置 Limit 时，容器可以使用节点所有可使用资源，会导致其它容器的资源被占用，且该类型的容器所在的 Pod 容易被驱逐，不建议使用。建议 Limit = Request。

CPU 使用量和 CPU 使用率

- CPU 使用量为绝对值，表示实际使用的 CPU 的物理核数，CPU 资源请求和 CPU 资源限制的判断依据都是 CPU 使用量。
- CPU 使用率为相对值，表示 CPU 的使用量与 CPU 单核的比值（或者与节点上总 CPU 核数的比值）。

使用示例

一个简单的示例说明 Request 和 Limit 的作用，测试集群包括1个 4U4G 的节点、已经部署的两个 Pod（Pod1，Pod2），每个 Pod 的资源设置为（CPU Request，CPU Limit，Memory Request，Memory Limit）=（1U，2U，1G，1G）。（1.0G = 1000MiB）
节点上 CPU 和内存的资源使用情况如下图所示：



已经分配的 CPU 资源为：1U（分配 Pod1）+ 1U（分配 Pod2）= 2U，剩余可以分配的 CPU 资源为2U。

已经分配的内存资源为：1G（分配 Pod1）+ 1G（分配 Pod2）= 2G，剩余可以分配的内存资源为2G。
所以该节点可以再部署一个（CPU Request， Memory Request）=(2U，2G)的 Pod 部署，或者部署2个（CPU Request，Memory Request）=（1U，1G）的 Pod 部署。
在资源限制方面，每个 Pod1 和 Pod2 使用资源的上限为（2U，1G），即在资源空闲的情况下，Pod 使用 CPU 的量最大能达到2U。

服务资源限制推荐

TKE 会根据您当前容器镜像的历史负载来推荐 Request 与 Limit 值，使用推荐值会保证您的容器更加平稳的运行，减小出现异常的概率。
推荐算法：
我们首先会取出过去7天当前容器镜像分钟级别负载，并辅以百分位统计第95%的值来最终确定推荐的 Request，Limit 为 Request 的2倍。

```
Request = Percentile(实际负载[7d],0.95)
Limit = Request * 2
```

如果当前的样本数量（实际负载）不满足推荐计算的数量要求，我们会相应的扩大样本取值范围，尝试重新计算。例如，去掉镜像 tag，namespace，serviceName 等筛选条件。若经过多次计算后同样未能得到有效值，则推荐值为空。
推荐值为空：
在使用过程中，您会发现有部分值暂无推荐的情况，可能由于以下几点造成：
1. 当前数据并不满足计算的需求，我们需要待计算的样本数量（实际负载）大于1440个，即有一天的数据。
2. 推荐值小于您当前容器已经配置的 Request 或者 Limit。

> **注意**
> - 由于推荐值是根据历史负载来计算的，原则上，容器镜像运行真实业务的时间越长，推荐的值越准确。
> - 使用推荐值创建服务，可能会因为集群资源不足造成容器无法调度成功。在保存时，须确认当前集群的剩余资源。
> - 推荐值是建议值，您可以根据自己业务的实际情况做相应的调整。

相关文档

容器的 Request 及 Limit 需根据服务类型、需求及场景进行灵活设置。详情可参见 设置 Request 与 Limit。

设置工作负载的调度规则

最近更新时间：2023-11-28 14:52:32

概述

通过设置工作负载中高级设置的调度规则，指定该工作负载下的 Pod 在集群内进行调度。存在以下应用场景：
- 将 Pod 运行在指定的节点上。
- 将 Pod 运行在某一作用域（作用域可以是可用区、机型等属性）的节点上。

使用方法

前置条件

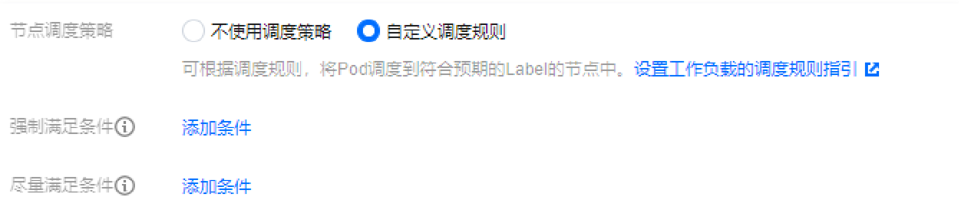
- 设置工作负载高级设置中的调度规则，且集群的 Kubernetes 版本必须是1.7以上的版本。
- 为确保您的 Pod 能够调度成功，请确保您设置的调度规则完成后，节点有空余的资源用于容器的调度。
- 使用自定义调度功能时，需要为节点设置对应 Label。详情请参见 设置节点 Label 。

设置调度规则

本文以通过容器服务控制台创建 Deployment 为例：
1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择需创建服务的集群 ID，进入集群的基本信息页面。
3. 在**工作负载 > Deployment**页面，单击**新建**。
4. 在**新建 Deployment** 页面，您可以根据实际需求，选择节点调度策略。支持选择**不使用调度策略**和**自定义调度规则**。

自定义调度规则

可自定义实例（Pod）调度规则，匹配实例标签。



自定义调度规则包含以下两种模式：
- **强制满足要求条件**：调度期间如果满足亲和性条件，则调度到对应 Node。如果没有节点满足条件，则调度失败。
- **尽量满足要求条件**：调度期间如果满足亲和性条件，则调度到对应 Node。如果没有节点满足条件，则随机调度到任意节点。

自定义调度规则均可以添加多条调度规则，各规则操作符的含义如下：

操作符	含义
In	Label 的 value 在列表中。
NotIn	Label 的 value 不在列表中。
Exists	Label 的 key 存在。
DoesNotExits	Label 的 key 不存在。
Gt	Label 的值大于列表值（字符串匹配）。
Lt	Label 的值小于列表值（字符串匹配）。

原理介绍

服务的调度规则主要通过下发 Yaml 到 Kubernetes 集群， Kubernetes 的 Affinity and anti-affinity 机制会使得 Pod 按一定规则进行调度。更多 Kubernetes 的 Affinity and anti-affinity 机制介绍请 查看详情 。

设置工作负载的健康检查

最近更新时间：2025-06-04 18:05:02

腾讯云容器集群内核基于 Kubernetes。Kubernetes 支持对容器进行周期性探测，并根据探测结果判断容器的健康状态，执行额外的操作。

健康检查类别

健康检查分为以下类别：

- **容器存活检查**：用于检测容器是否存活，类似于执行 ps 命令检查进程是否存在。如果容器的存活检查失败，集群会对该容器执行重启操作。如果容器的存活检查成功，则不执行任何操作。
- **容器就绪检查**：用于检测容器是否准备好开始处理用户请求。例如，程序的启动时间较长时，需要加载磁盘数据或者要依赖外部的某个模块启动完成才能提供服务。此时，可通过容器就绪检查方式检查程序进程，确认程序是否启动完成。如果容器的就绪检查失败，集群会屏蔽请求访问该容器。如果容器的就绪检查成功，则会开放对该容器的访问。

健康检查方式

TCP 端口探测

TCP 端口探测的原理如下：
对于提供 TCP 通信服务的容器，集群周期性地对该容器建立 TCP 连接。如果连接成功，证明探测成功，否则探测失败。选择 TCP 端口探测方式，必须指定容器监听的端口。
例如，一个 redis 容器，它的服务端口是6379。我们对该容器配置了 TCP 端口探测，并指定探测端口为6379，那么集群会周期性地对该容器的6379端口发起 TCP 连接。如果连接成功，证明检查成功，否则检查失败。

HTTP 请求探测

HTTP 请求探测是针对于提供 HTTP/HTTPS 服务的容器，并集群周期性地对该容器发起 HTTP/HTTPS GET 请求。如果 HTTP/HTTPS response 返回码属于200 - 399范围，证明探测成功，否则探测失败。使用 HTTP 请求探测必须指定容器监听的端口和 HTTP/HTTPS 的请求路径。
例如，提供 HTTP 服务的容器，服务端口为 80，HTTP 检查路径为 `/health-check`，那么集群会周期性地对容器发起 `GET http://containerIP:80/health-check` 请求。

执行命令检查

执行命令检查是一种强大的检查方式，该方式要求用户指定一个容器内的可执行命令，集群会周期性地在容器内执行该命令。如果命令的返回结果是0，检查成功，否则检查失败。
对于 TCP 端口探测 和 HTTP 请求探测，都可以通过执行命令检查的方式来替代：

- 对于 TCP 端口探测，可以写一个程序对容器的端口进行 connect。如果 connect 成功，脚本返回0，否则返回-1。
- 对于 HTTP 请求探测，可以编写一个脚本对容器进行 wget 并检查 response 的返回码。例如，`wget http://127.0.0.1:80/health-check`。如果返回码在200 - 399的范围，脚本返回0，否则返回-1。

注意事项

- 必须将需要执行的程序放在容器的镜像中，否则会因找不到程序而执行失败。
- 若执行的命令是一个 shell 脚本，则不能直接指定脚本作为执行命令，需要加上脚本的解释器。例如，脚本是 `/data/scripts/health_check.sh`，那么使用执行命令检查时，指定的程序应为：

```
sh
```

```
/data/scripts/health_check.sh
```

GRPC 检查

该检测项使用 GRPC 健康检查协议（grpc.health.v1.Health），通过调用服务的 Check 方法，根据响应状态判断健康程度。kubelet 通过 GRPC 调用容器端点的 Health/Check 方法，服务返回 SERVING 状态代表成功，否则失败。该检测项的使用对 K8s 集群版本有要求，建议保持在1.24及以上，更多功能细节请参见 社区介绍。

操作步骤

设置步骤以通过容器服务控制台创建 Deployment 为例：

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择需创建服务的集群 ID，进入集群的基本信息页面。
3. 在**工作负载 > Deployment** 页面，单击**新建**。Deployment 参数设置请参见 Deployment 管理 。
4. 在**新建工作负载**页面，单击**实例内容器**中的**显示高级设置**。
5. 在**容器健康检查**中，以选择**存活检查**为例，设置以下参数。
   - **检查方法**：选择“执行命令检查”。
   - **执行命令**：输入以下内容。

```
sh
```

```
/data/scripts/health_check.sh
```

   其余公共参数如下：
   - **启动延时**：单位秒。指定容器启动后，多久开始探测。例如，启动延时设置为5，那么健康检查将在容器启动5秒后开始。
   - **间隔时间**：单位秒。指定健康检查的频率。例如，间隔时间设置成10，那么集群会每隔10s检查一次。
   - **响应超时**：单位秒。指定健康探测的超时时间。对应到 TCP 端口探测、HTTP 请求探测、执行命令检查三种方式，分别表示 TCP 连接超时时间、HTTP 请求响应超时时间以及执行命令的超时时间。
   - **健康阈值**：单位次。指定健康检查连续成功多少次后，才判定容器是健康的。例如，健康阈值设置成3，则说明只有满足连续3次探测都成功，才认为容器是健康的。

   > **注意：**
   > 如果健康检查的类型为存活检查，那么健康阈值只能是1，用户设置成其它值将被视为无效。

   - **不健康阈值**：单位次。指定健康检查连续失败多少次后，才判定容器是不健康的。例如，不健康阈值设置成3，则说明只有满足连续3次都探测失败，才认为容器是不健康的。
6. 单击**创建 Deployment**。

设置工作负载的运行命令和参数

最近更新时间：2024-10-17 15:40:29

概述

创建工作负载时，通常通过镜像来指定实例中容器所运行的进程。在默认的情况下，镜像会运行默认的命令，如果您需要运行一个特定的命令或重写镜像的默认值，您需要使用到以下三个设置：

- 工作目录（workingDir）：指定当前的工作目录。
- 运行命令（command）：控制镜像运行的实际命令。
- 命令参数（args）：传递给运行命令的参数。

工作目录说明

WorkingDir 即指定当前的工作目录。如果不存在，则自动创建。如果没有指定，则使用容器运行时的默认值。如果镜像中如果没指定 WORKDIR，且在控制台未指定，则 workingDir 默认为 “/”。

命令和参数的使用

如何将 docker run 命令适配到腾讯云容器服务，请参见 docker run 参数适配。
Docker 的镜像拥有存储镜像信息的相关元数据，如果不提供运行命令和参数，容器将会运行镜像制作时提供的默认的命令和参数。Docker 原生定义的字段为 “Entrypoint” 和 “CMD”。详情可查看 Docker 的 Entrypoint 说明 和 CMD 说明。
如果您在创建服务时，填写了容器的运行命令和参数，容器服务将会覆盖镜像构建时的默认命令（即 “Entrypoint” 和 “CMD”）。其规则如下：

镜像 Entrypoint	镜像 CMD	容器的运行命令	容器的运行参数	最终执行
[ls]	[/home]	未设置	未设置	[ls / home]
[ls]	[/home]	[cd]	未设置	[cd]
[ls]	[/home]	未设置	[/data]	[ls / data]
[ls]	[/home]	[cd]	[/data]	[cd / data]

注意

- Docker entrypoint 对应容器服务控制台上的运行命令，Docker run 的 CMD 参数对应容器服务控制台上的运行参数。当有多个运行参数时，需在容器服务的运行参数中输入参数，且每个参数单独一行。
- 通过 容器服务控制台 设置容器运行命令和参数的示例请参考 Command 和 Args。

使用 TCR 企业版实例内容器镜像创建工作负载

最近更新时间：2023-12-20 19:19:01

操作场景

腾讯云容器镜像服务（Tencent Container Registry，TCR）企业版面向具有严格数据安全及合规性要求、业务分布在多个地域、集群规模庞大的企业级容器客户，提供企业级的独享镜像安全托管服务。相较于个人版服务，企业版支持容器镜像安全扫描、跨地域自动同步、Helm Chart 托管、网络访问控制等特性，详情请参见 容器镜像服务 。

本文介绍如何在容器服务 TKE 中，使用容器镜像服务 TCR 内托管的私有镜像进行应用部署。

前提条件

在使用 TCR 内托管的私有镜像进行应用部署前，您需要完成以下准备工作：

- 已在 容器镜像服务 创建企业版实例。如尚未创建，请参考 创建企业版实例 完成创建。
- 如果使用子账号进行操作，请参考 企业版授权方案示例 提前为子账号授予对应实例的操作权限。

操作步骤

准备容器镜像

创建命名空间

新建的 TCR 企业版实例内无默认命名空间，且无法通过推送镜像自动创建。请参考 创建命名空间 按需完成创建。

建议命名空间名使用项目或团队名，本文以 `docker` 为例。创建成功后如下图所示：

创建镜像仓库（可选）

容器镜像托管在具体的镜像仓库内，请参考 创建镜像仓库 按需完成创建。镜像仓库名称请设置为期望部署的容器镜像名称，本文以 `getting-started` 为例。创建成功后如下图所示：

> **说明**
> 通过 docker cli 或其他镜像工具，例如 jenkins 推送镜像至企业版实例内时，若镜像仓库不存在，将会自动创建，无需提前手动创建。

推送容器镜像

您可通过 docker cli 或其他镜像构建工具，例如 jenkins 推送镜像至指定镜像仓库内，本文以 docker cli 为例。此步骤需要您使用一台安装有 Docker 的云服务器或物理机，并确保访问的客户端已在 配置网络访问策略 定义的公网或内网允许访问范围内。

1. 参考 获取实例访问凭证 获取登录指令，并进行 Docker Login。
2. 登录成功后，您可在本地构建新的容器镜像或从 DockerHub 上获取一个公开镜像用于测试。
   本文以 DockerHub 官方的 Nginx 最新镜像为例，在命令行工具中依次执行以下指令，即可推送该镜像。请将 demo-tcr、docker 及 getting-started 依次替换为您实际创建的实例名称、命名空间名称及镜像仓库名。

```
docker tag getting-started:latest demo-tcr.tencentcloudcr.com/docker/getting-started:latest
```

```
docker push demo-tcr.tencentcloudcr.com/docker/getting-started:latest
```

推送成功后，即可前往控制台的 镜像仓库 页面，选择仓库名进入详情页面查看。

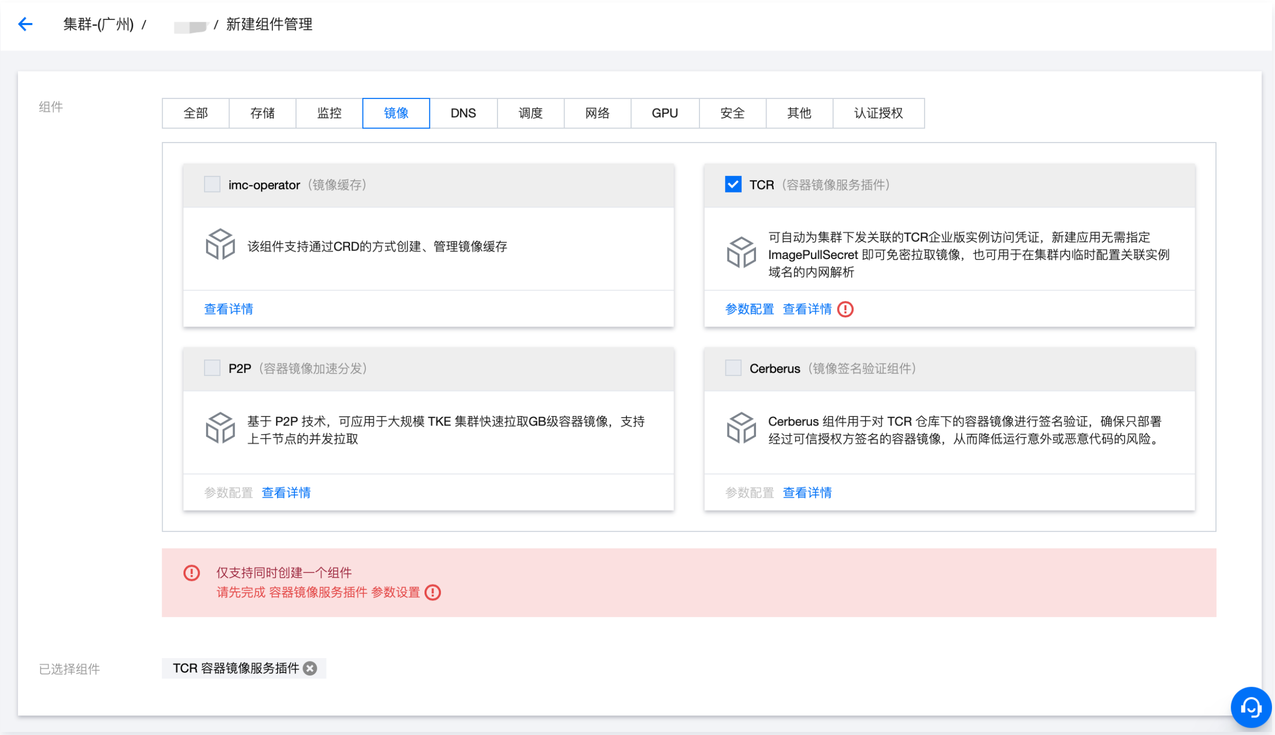
配置 TKE 集群访问 TCR 实例

TCR 企业版实例支持网络访问控制，默认拒绝全部来源的外部访问。您可根据 TKE 集群的网络配置，选择通过公网或内网访问指定实例，拉取容器镜像。若 TKE 集群与 TCR 实例部署在同一地域，建议通过内网访问方式拉取容器镜像，可提升拉取速度，并节约公网流量成本。

使用 TCR 扩展组件进行快速配置（推荐）

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群管理**页面，选择集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 TCR 组件。如下图所示：

> **说明**
> 当前 TCR 组件暂只支持 K8S 版本 ≥ 1.12 的集群，如集群版本暂不支持，请采用手动配置方式，或升级集群版本。



- 单击**查看详情**了解组件功能及配置说明。
- 单击**参数配置**开始配置组件。

5. 在 **TCR 组件参数设置**页面，参考**查看详情**中介绍的组件配置方式，配置相关参数。如下图所示：

- **关联实例**：选择与集群同地域的 TCR 实例。
- **免密拉取配置**：可采用默认配置。
- **内网访问配置**：可选功能，在 TCR 实例接入集群所在 VPC 并开启自动解析后，集群内节点可内网访问 TCR 实例，无需使用本功能。由于 TCR 侧自动解析功能依赖于 PrivateDNS，若当前集群所在地暂未支持 PrivateDNS 产品，可使用本配置实现内网访问。如内网访问链路中未展示为“链路正常”，请参考 内网访问控制，配置 TCR 实例与 TKE 集群所在私有网络 VPC 的内网链路。

6. 单击**确定**返回组件选择界面。
7. 在组件选择界面单击**完成**，开始为集群安装 TCR 扩展组件。

8. 组件安装完成后，集群将具备内网免密拉取该关联实例内镜像的能力，无需额外配置。如下图所示：

手动配置内网访问及访问凭证

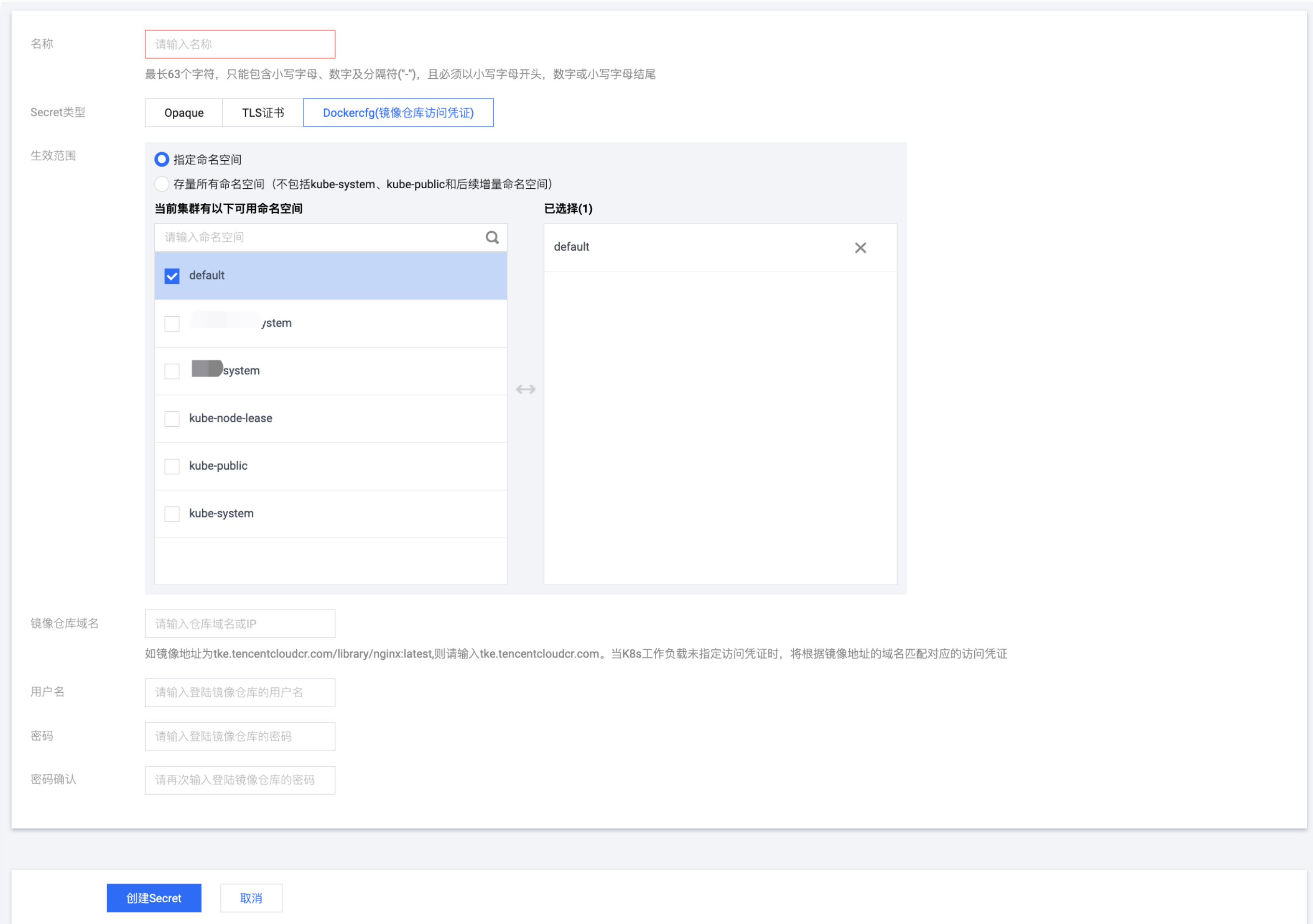
1. 配置内网访问

1. 参考 内网访问控制，配置 TCR 实例与 TKE 集群所在私有网络 VPC 的内网链路，并开启自动解析。
2. 如当前 TCR 实例所在地域暂不支持开启自动解析，可在 TKE 集群中直接配置 TCR 实例的域名解析。请根据您的实际情况，选择以下方案：
   - **创建集群时配置节点 Host**
     在创建 TKE 集群的“云服务器配置”步骤中，选择**高级设置**并在“节点启动配置”中输入如下内容：
     ```
     echo '172.21.17.69 demo.tencentcloudcr.com' >> /etc/hosts
     ```
   - **为已有集群配置节点 Host**
     登录集群各个节点，并执行以下命令：
     ```
     echo '172.21.17.69 demo.tencentcloudcr.com' >> /etc/hosts
     ```
     `172.21.17.69` 及 `demo.tencentcloudcr.com` 请替换为您实际使用的内网解析 IP 及 TCR 实例域名。

2. 配置访问凭证

1. 参考 TCR – 服务级账号管理，获取服务级账号的用户名及密码。
2. 在集群详情页，选择左侧的**配置管理** > **Secret**，进入 Secret 页面。

3. 在 Secret 页面单击**新建**进入**新建 Secret** 页面，参考以下信息下发访问凭证。如下图所示：



主要参数信息如下：
- **Secret 类型**：选择 Dockercfg (镜像仓库访问凭证)。
- **生效范围**：勾选需下发凭证的命名空间。
- **镜像仓库域名**：填写 TCR 实例的访问域名。
- **用户名、密码**：填写 步骤1 已获取的用户名及密码。

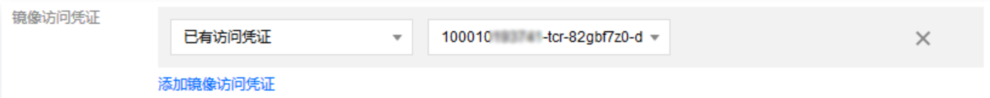
4. 单击**创建 Secret** 即可完成下发。

使用 TCR 实例内容器镜像创建工作负载

1. 在集群详情页面，选择左侧**工作负载** > **Deployment**。
2. 在 Deployment 页面单击**新建**。
3. 在**新建 Deployment** 页面，参考以下信息创建工作负载。
主要参数信息如下，其他参数请按需设置：
   - **命名空间**：选择已下发访问凭证的命名空间。
   - **实例内容器**：
     - **镜像**：单击**选择镜像**，并在**选择镜像**页面，选择**容器镜像服务企业版**，再根据需要选择地域、实例和镜像仓库。如下图所示：

- **镜像版本**：选择好镜像后，单击**选择镜像版本**，在**选择镜像版本**页面，根据需要选择该镜像仓库的某个版本。若不选择则默认为 latest。

- **镜像访问凭证**：
  - 集群已安装 TCR 扩展组件：无需配置。
  - 集群未安装 TCR 扩展组件：选择**添加镜像访问凭证**，并选择 配置访问凭证 步骤中已下发的访问凭证。如下图所示：



4. 完成其他参数设置后，单击**创建 Deployment** 后查看该工作负载的部署进度。
   部署成功后，可在 Deployment 页面查看该工作负载的“运行/期望Pod数量”为“1/1”。

配置
ConfigMap 管理

最近更新时间：2023-11-28 15:41:36

简介

通过 ConfigMap 您可以将配置和运行的镜像进行解耦，使得应用程序有更强的移植性。ConfigMap 是有 key-value 类型的键值对，您可以通过控制台的 Kubectl 工具创建对应的 ConfigMap 对象，也可以通过挂载数据卷、环境变量或在容器的运行命令中使用 ConfigMap。

通过控制台

创建 ConfigMap

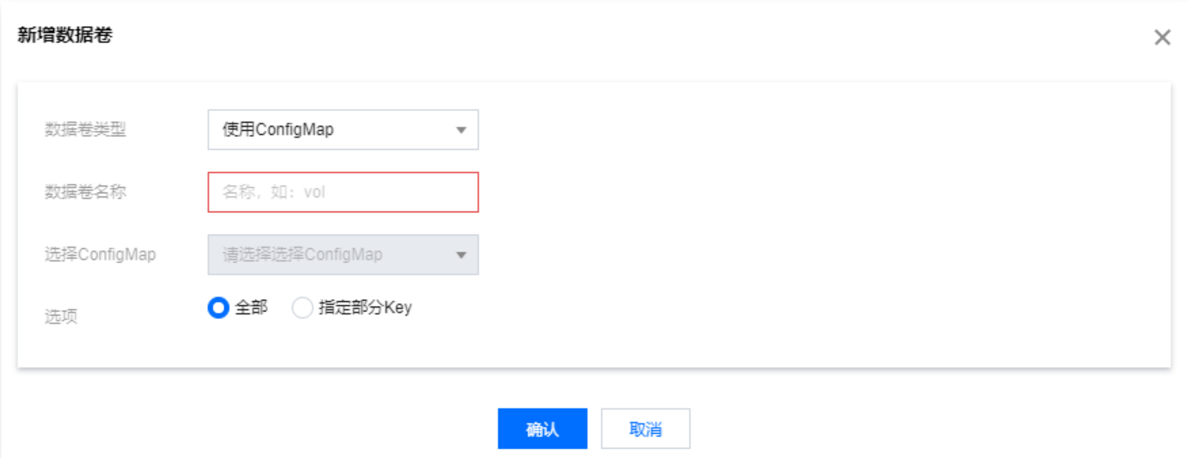
1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 选择**配置管理** > **ConfigMap**，在 ConfigMap 信息页面单击**新建**。
4. 在新建 ConfigMap 页面，根据实际需求，设置 ConfigMap 参数。关键参数信息如下：
    - 名称：自定义。
    - 命名空间：根据实际需求进行选择命名空间类型，定义变量名和变量值。
    - 内容：添加变量名和变量值。
5. 单击**创建 ConfigMap**，完成创建。

使用 ConfigMap

方式1：数据卷使用 ConfigMap 类型

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载**中，任意选择 Workload 类型，进入对应的信息页面。例如，选择**工作负载** > **DaemonSet**，在 DaemonSet 信息页面单击**新建**。
4. 在**新建 DaemonSet** 页面，根据页面信息，设置工作负载名、命名空间等信息。并在数据卷中，单击**添加数据卷**。如下图所示：

5. 在**新增数据卷**中，参考以下信息配置挂载点，并单击**确认**。如下图所示：



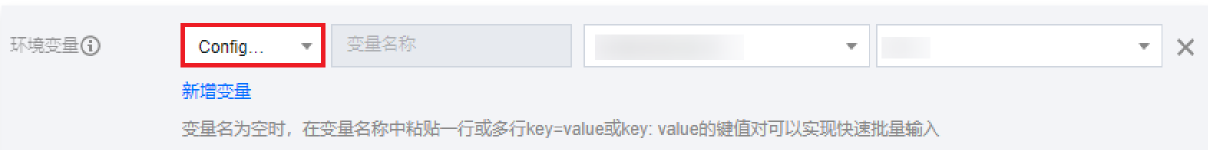
   - **数据卷类型**：选择 “使用ConfigMap” 方式。
   - **数据卷名称**：自定义名称。
   - **选择ConfigMap**：根据实际需求进行选择。
   - **选项**：提供 “全部” 和 “指定部分Key” 两种选择。
     - **Items**：当选择 “指定部分Key” 选项时，可以通过添加 item 向特定路径挂载，如挂载点是 /data/config，文件名是 filename，最终该键值对的值会存储在 /data/config/filename 下。

6. 单击**创建 DaemonSet**，完成创建。

方式2：环境变量中使用 ConfigMap 类型

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载**中，任意选择 Workload 类型，进入对应的信息页面。例如，选择**工作负载** > **DaemonSet**，在 DaemonSet 信息页面单击**新建**。
4. 在**新建 DaemonSet** 页面，根据页面信息，设置工作负载名、命名空间等信息。并在**实例内容器**的环境变量中，单击**新增变量**。如下图所示：

5. 选择 “ConfigMap” 环境变量方式，并根据实际需求选择资源。如下图所示：



6. 单击**创建 DaemonSet**，完成创建。

更新 ConfigMap

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，选择集群 ID，进入集群的基本信息页面。
3. 选择**配置管理** > **ConfigMap**，在 ConfigMap 信息页面，单击需要更新的 ConfigMap 行右侧的**更新配置**，如下图所示：

4. 在**更新配置**页面，编辑 key-value 类型的键值对，单击**更新 ConfigMap**。

通过 Kubectl

YAML 示例

```
apiVersion: v1
data:
  key1: value1
  key2: value2
  key3: value3
kind: ConfigMap
metadata:
  name: test-config
  namespace: default
```

说明：

- **data**：ConfigMap 的数据，以 key-value 形式呈现。
- **kind**：标识 ConfigMap 资源类型。
- **metadata**：ConfigMap 的名称、Label 等基本信息。
- **metadata.annotations**：ConfigMap 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。

创建 ConfigMap

方式1：通过 YAML 示例文件方式创建

1. 参考 YAML 示例，准备 ConfigMap YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参考 通过 Kubectl 连接集群。
3. 执行以下命令，创建 ConfigMap YAML 文件。


```
kubectl create -f ConfigMap YAML 文件名称
```

例如，创建一个文件名为 web.yaml 的 ConfigMap YAML 文件，则执行以下命令：

```
kubectl create -f web.yaml
```

4. 执行以下命令，验证创建是否成功。

```
kubectl get configmap
```

返回类似以下信息，即表示创建成功。

```
NAME           DATA      AGE
test           2         39d
test-config    3         18d
```

方式2：通过执行命令方式创建

执行以下命令，在目录中创建 ConfigMap。

```
kubectl create configmap <map-name> <data-source>
```

- <map-name>：表示 ConfigMap 的名字。
- <data-source>：表示目录、文件或者字面值。

更多参数详情可参见 Kubernetes configMap 官方文档 。

使用 ConfigMap

方式1：数据卷使用 ConfigMap 类型

YAML 示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx1
spec:
  containers:
  - name: nginx
    image: nginx:latest
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: test-config ## 设置 ConfigMap 来源
      ## items:  ## 设置指定 ConfigMap 的 Key 挂载
      ## - key: key1  ## 选择指定 Key
      ##   path: keys ## 挂载到指定的子路径
  restartPolicy: Never
```

方式2：环境变量中使用 ConfigMap 类型

YAML 示例如下：

```
apiVersion: v1
```

```yaml
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    env:
    - name: key1
      valueFrom:
        configMapKeyRef:
          name: test-config ## 设置来源 ConfigMap 文件名
          key: test-config.key1  ## 设置该环境变量的 Value 来源项
  restartPolicy: Never
```

Secret 管理

最近更新时间：2023-11-01 16:44:01

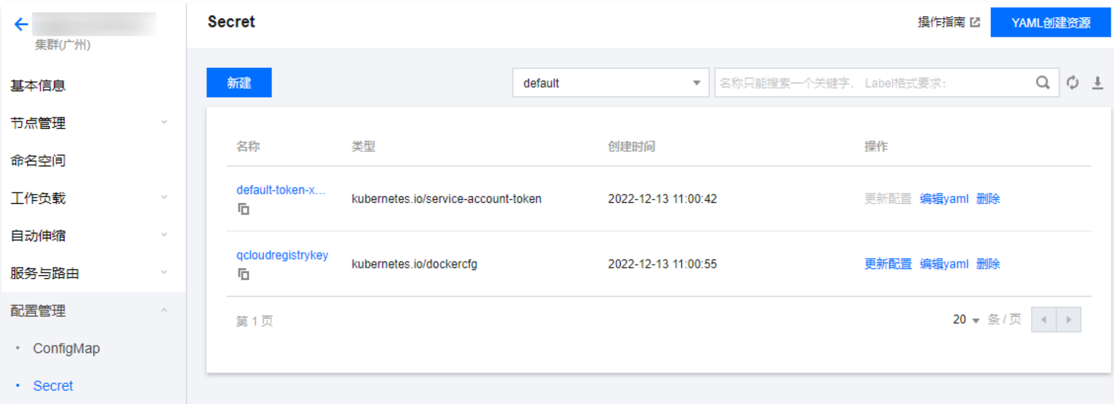
简介

Secret 可用于存储密码、令牌、密钥等敏感信息，降低直接对外暴露的风险。Secret 是 key-value 类型的键值对，您可以通过控制台的 Kubectl 工具创建对应的 Secret 对象，也可以通过挂载数据卷、环境变量或在容器的运行命令中使用 Secret。

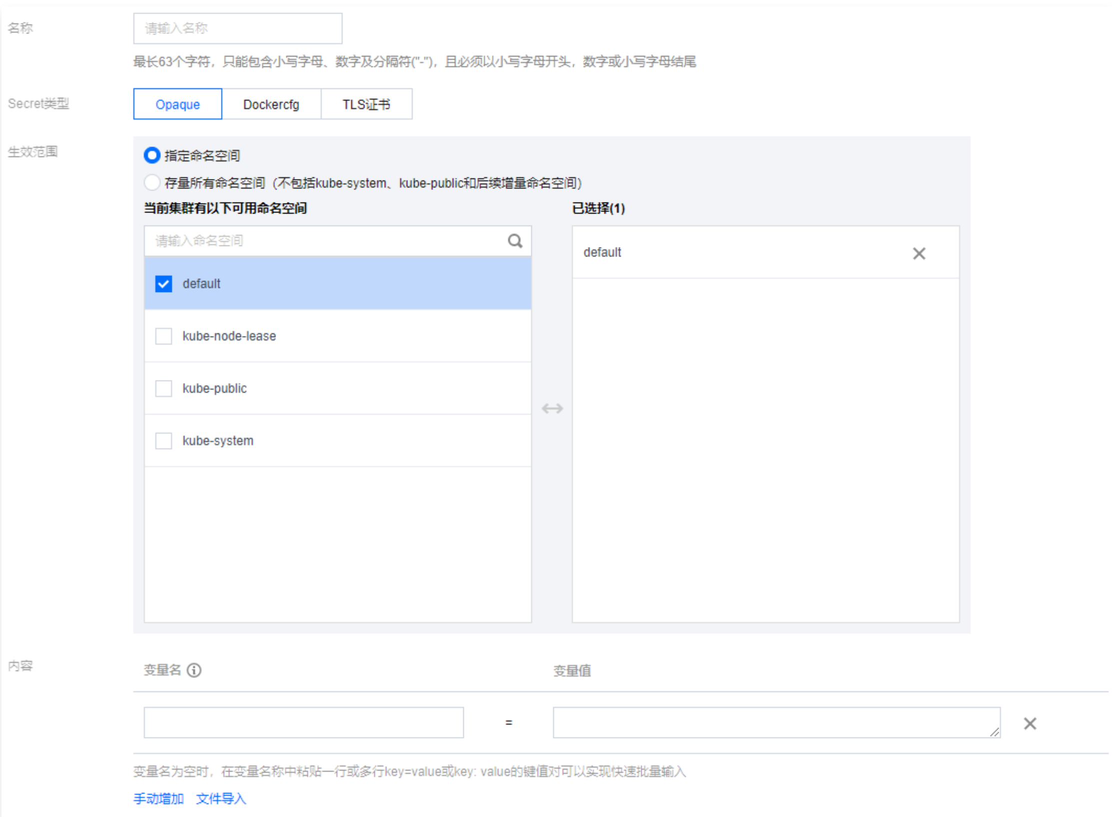
通过控制台

创建 Secret

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 选择需要创建 Secret 的集群 ID，进入集群管理页面。
3. 选择左侧导航栏中的**配置管理** > **Secret**，进入 Secret 信息页面。如下图所示：



4. 单击**新建**，在**新建 Secret** 页面，根据实际需求，进行如下参数设置。如下图所示：



  - **名称**：请输入自定义名称。
  - **Secret 类型**：提供 Opaque、Dockercfg 和 TLS 证书类型，请根据实际需求进行选择。

        - **Opaque**：适用于保存密钥证书和配置文件，Value 将以 base64 格式编码。
        - **Dockercfg**：适用于保存私有 Docker Registry 的认证信息。
        - **TLS 证书**：适用于保存 TLS 证书和私钥。
    - **生效范围**：提供以下两种范围，请根据实际需求进行选择。
        - **存量所有命名空间**：不包括 kube-system、kube-public 和后续增量命名空间。
        - **指定命名空间**：支持选择当前集群下一个或多个可用命名空间。
    - **内容**：根据不同的 Secret 类型，进行配置。
        - 当 Secret 类型为 **Opaque** 时：根据实际需求，设置变量名和变量值。
        - 当 Secret 类型为 **Dockercfg** 时：
            - 仓库域名：请根据实际需求输入域名或 IP。
            - 用户名：请根据实际需求输入第三方仓库的用户名。
            - 密码：请根据实际需求设置第三方仓库的登录密码。

            > **说明**
            > 如果本次为首次登录系统，则会新建用户，相关信息写入 `~/.dockercfg` 文件中。

        - 当 Secret 类型为 **TLS 证书** 时：添加证书和私钥。
5. 单击**创建 Secret**，即可完成创建。

使用 Secret

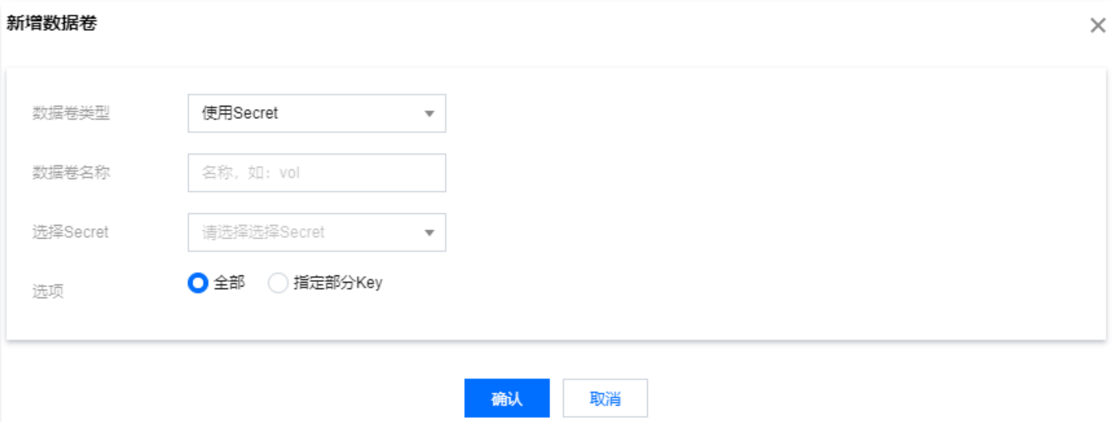
方式一：数据卷使用 Secret 类型

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 选择需要部署 Workload 的集群 ID，进入集群管理页面。
3. 在**工作负载**下，任意选择 Workload 类型，进入对应的信息页面。
   例如，选择**工作负载** > **DaemonSet**，进入 DaemonSet 信息页面。如下图所示：

4. 单击**新建**，进入**新建** **DaemonSet** 页面。

5. 根据页面信息，设置工作负载名、命名空间等信息。并在 “数据卷” 中，单击**添加数据卷**。如下图所示：

6. 选择**使用 Secret** 方式，填写名称，并单击**选择Secret**。如下图所示：



- **选择 Secret**：根据实际需求进行。
- **选项**：提供**全部**和**指定部分 Key**两种选择。
  - **Items**：当选择**指定部分 Key**选项时，可以通过添加 Item 向特定路径挂载，如挂载点是 `/data/config`，子路径是 `dev`，最终会存储在 `/data/config/dev` 下。

7. 单击**创建 DaemonSet**，完成创建。

方式二：环境变量中使用 Secret 类型

1. 登录容器服务控制台，选择左侧导航栏中的 **集群**。
2. 选择需要部署 Workload 的集群 ID，进入集群管理页面。
3. 在**工作负载**下，任意选择 Workload 类型，进入对应的信息页面。
   例如，选择**工作负载** > **DaemonSet**，进入 DaemonSet 信息页面。
4. 单击**新建**，进入**新建 DaemonSet** 页面。
5. 根据页面信息，设置工作负载名、命名空间等信息。并在**实例内容器**的“环境变量”中，选择 **Secret** 环境变量方式，并根据实际需求选择资源。如下图所示：

6. 单击**创建 DaemonSet**，完成创建。

方法三：使用第三方镜像仓库时引用

1. 登录容器服务控制台，选择左侧导航栏中的 **集群**。
2. 选择需要部署 Workload 的集群 ID，进入集群管理页面。

3. 在**工作负载**下，任意选择 Workload 类型，进入对应的信息页面。
   例如，选择**工作负载** > **DaemonSet**，进入 DaemonSet 信息页面。
4. 单击**新建**，进入**新建 DaemonSet** 页面。
5. 根据页面信息，设置工作负载名、命名空间等信息。请根据实际情况选择**镜像访问凭证**。
6. 单击**创建 DaemonSet**，完成创建。

更新 Secret

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 选择需要更新 YAML 的集群 ID，进入集群管理页面。
3. 选择**配置管理** > **Secret**，进入 Secret 信息页面。
4. 在需要更新 YAML 的 Secret 行中，单击**编辑 YAML**。
5. 在**编辑 YAML** 页面，编辑 YAML，并单击**完成**。

> **说明**
> 如需修改 key-values，则编辑 YAML 中 data 的参数值，并单击**完成**即可完成更新。

通过 Kubectl

创建 Secret

方式一：通过指定文件创建 Secret

1. 依次执行以下命令，获取 Pod 的用户名和密码。

```
echo -n 'username' > ./username.txt
echo -n 'password' > ./password.txt
```

2. 执行 Kubectl 命令，创建 Secret。

```
kubectl create secret generic test-secret --from-file=./username.txt --from-file=./password.txt
secret "test-secret" created
```

3. 执行以下命令，查看 Secret 详情。

```
kubectl describe secrets/test-secret
```

方式二：YAML 文件手动创建

> **说明**
> 通过 YAML 手动创建 Secret，需提前将 Secret 的 data 进行 Base64 编码。

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
type: Opaque
data:
  username: dXNlcm5hbWU=  ## 由echo -n 'username' | base64生成
  password: cGFzc3dvcmQ=  ## 由echo -n 'password' | base64生成
```

使用 Secret

方式一：数据卷使用 Secret 类型

YAML 示例如下：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/config
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret  ## 设置 secret 来源
      items:  ## 设置指定 secret 的 Key 挂载
      - key: username  ## 选择指定 Key
        path: group/user  ## 挂载到指定的子路径
        mode: 256  ## 设置文件权限
  restartPolicy: Never
```

方式二：环境变量中使用 Secret 类型

YAML 示例如下：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: test-secret ## 设置来源 Secret 文件名
          key: username  ## 设置该环境变量的 Value 来源项
  restartPolicy: Never
```

方法三：使用第三方镜像仓库时引用

YAML 示例如下：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
  imagePullSecrets:
  - name: test-secret ## 设置来源 Secret 文件名
  restartPolicy: Never
```

工作负载伸缩
水平伸缩（HPA）
水平伸缩基本操作

最近更新时间：2024-10-29 15:56:52

操作场景

实例（Pod）水平自动扩缩容功能（Horizontal Pod Autoscaler，HPA）可以根据目标实例 CPU 利用率的平均值等指标自动扩展、缩减服务的 Pod 数量。本文介绍如何通过腾讯云容器服务控制台实现 Pod 自动扩缩容。

工作原理

HPA 后台组件会每隔15秒向腾讯云可观测平台拉取容器和 Pod 的监控指标，然后根据当前指标数据、当前副本数和该指标目标值进行计算，计算所得结果作为服务的期望副本数。当期望副本数与当前副本数不一致时，HPA 会触发 Deployment 进行 Pod 副本数量调整，从而达到自动伸缩的目的。
以 CPU 利用率为例，假设当前有2个实例， 平均 CPU 利用率（当前指标数据）为90%，自动伸缩设置的目标 CPU 为60%， 则自动调整实例数量为：90% × 2 / 60% = 3个。

> **注意：**
> 如果用户设置了多个弹性伸缩指标，HPA 会依据各个指标，分别计算出目标副本数，取最大值进行扩缩容操作。

注意事项

- 当指标类型选择为 **CPU 利用率（占 Request）**时，必须为容器设置 CPU Request。
- 策略指标目标设置合理，例如设置70%给容器和应用，预留30%的余量。
- 保持 Pod 和 Node 健康（避免 Pod 频繁重建）。
- 保证用户请求的负载均衡稳定运行。
- HPA 在计算目标副本数时会有一个10%的波动因子。如果在波动范围内，HPA 并不会调整副本数目。
- 如果服务对应的 Deployment.spec.replicas 值为0，HPA 将不起作用。
- 如果对单个 Deployment 同时绑定多个 HPA ，则创建的 HPA 会同时生效，会造成工作负载的副本重复扩缩。

前提条件

- 已 注册腾讯云账户 。
- 已登录 腾讯云容器服务控制台 。
- 已创建集群。操作详情请参见 创建集群 。

操作步骤

开启自动扩缩容

可以通过以下方式开启自动扩缩容。

通过设置实例数量调节

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**工作负载 > Deployment** 页面，单击**新建**。
4. 在**新建工作负载**页面，设置实例数量为**自动调节**。如下图所示：

- **触发策略**：自动伸缩功能依赖的策略指标。详情请参见 自动伸缩指标说明 。
- **实例范围**：请根据实际需求进行选择，实例数量会在设定的范围内自动调节，不会超出该设定范围。

通过新建自动伸缩组

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**自动伸缩 > HorizontalPodAutoscaler** 页面，单击**新建**。
4. 在**新建自动伸缩**页面，根据以下提示，进行 HPA 配置。如下图所示：

   - **名称**：输入要创建的自动伸缩组的名称。
   - **命名空间**：请根据实际需求进行选择。
   - **工作负载类型**：请根据实际需求进行选择。
   - **关联工作负载**：不能为空，请根据实际需求进行选择。
   - **触发策略**：自动伸缩功能依赖的策略指标，详情请参见 自动伸缩指标说明 。
   - **实例范围**：请根据实际需求进行选择，实例数量会在设定的范围内自动调节，不会超出该设定范围。
5. 单击**创建** HPA。

更新自动扩缩容规则

可以通过以下方式更新服务自动扩缩容规则。

通过更新 Pod 数量

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。

3. 在**工作负载 > Deployment** 页面，单击实例右侧的**更新 Pod 数量**。

4. 在**更新 Pod 数量**页面，选择**自动调节**，并根据实际需求进行设置。如下图所示：

5. 单击**更新实例数量**。

通过修改 HPA 配置

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**自动伸缩 > HorizontalPodAutoscaler** 页面，单击实例右侧的**修改配置**。如下图所示：

4. 在**更新配置**页面，根据实际需求进行设置。如下图所示：

5. 单击**更新** HPA。

指标类型

相关指标和类型请参见 水平自动伸缩指标说明 。

水平自动伸缩指标说明

最近更新时间：2024-10-29 15:56:52

实例（Pod）自动扩缩容功能（Horizontal Pod Autoscaler，HPA）可以根据目标实例 CPU 利用率的平均值等指标自动扩展、缩减服务的 Pod 数量。自动扩缩容时，可供在控制台进行设置的触发指标类型包括 CPU 指标、内存、硬盘、网络和 GPU 相关指标。此外，这些指标还可以在您通过 YAML 文件创建和编辑 HPA 时使用，本文将给出配置 YAML 文件示例。

自动伸缩指标

自动伸缩指标详情如下表所示：

> **说明：**
> 其中 `metricName` 中的变量本身有单位，即表中所示默认单位，该单位在编写 YAML 文件时可忽略。

CPU 指标

指标名称（控制台）	单位（控制台）	备注	type	metricName	默认单位
CPU 使用量	核	Pod 的 CPU 使用量	Pods	k8s_pod_cpu_core_used	核
CPU 利用率（占节点）	%	Pod 的 CPU 使用量占节点总量之比	Pods	k8s_pod_rate_cpu_core_used_node	%
CPU 利用率（占 Request）	%	Pod 的 CPU 使用量和 Pod 中容器设置的 Request 值之比	Pods	k8s_pod_rate_cpu_core_used_request	%
CPU 利用率（占 Limit）	%	Pod 的 CPU 使用量和 Pod 中容器设置的 Limit 之和的比例	Pods	k8s_pod_rate_cpu_core_used_limit	%

硬盘

指标名称（控制台）	单位（控制台）	备注	type	metricName	默认单位
硬盘写流量	KB/s	Pod 的硬盘写速率	Pods	k8s_pod_fs_write_bytes	B/s
硬盘读流量	KB/s	Pod 的硬盘读速率	Pods	k8s_pod_fs_read_bytes	B/s
硬盘读 IOPS	次/s	Pod 从硬盘读取数据的 IO 次数	Pods	k8s_pod_fs_read_times	次/s
硬盘写 IOPS	次/s	Pod 把数据写入硬盘的 IO 次数	Pods	k8s_pod_fs_write_times	次/s

网络

指标名称（控制台）	单位（控制台）	备注	type	metricName	默认单位
网络入带宽	Mbps	单 Pod 下所有容器的入方向带宽之和	Pods	k8s_pod_network_receive_bytes_bw	bps
网络出带宽	Mbps	单 Pod 下所有容器的出方向带宽之和	Pods	k8s_pod_network_transmit_bytes_bw	bps

网络入流量	KB	单 Pod 下所有容器的入方向流量之和	Pods	k8s_pod_network_receive_bytes	B
网络出流量	KB	单 Pod 下所有容器的出方向流量之和	Pods	k8s_pod_network_transmit_bytes	B
网络入包量	个/s	单 Pod 下所有容器的入方向包数之和	Pods	k8s_pod_network_receive_packets	个/s
网络出包量	个/s	单 Pod 下所有容器的出方向包数之和	Pods	k8s_pod_network_transmit_packets	个/s

内存

指标名称（控制台）	单位（控制台）	备注	type	metricName	默认单位
内存使用量	MiB	Pod 内存使用量	Pods	k8s_pod_mem_usage_bytes	B
内存使用量（不包含 Cache）	MiB	Pod 内存使用，不包含 Cache	Pods	k8s_pod_mem_no_cache_bytes	B
内存利用率（占节点）	%	Pod 内存使用占 node 的比例	Pods	k8s_pod_rate_mem_usage_node	%
内存利用率（占节点，不包含 Cache）	%	Pod 内存使用占 node 的比例，不含 Cache	Pods	k8s_pod_rate_mem_no_cache_node	%
内存利用率（占 Request）	%	Pod 内存使用占 Request 的比例	Pods	k8s_pod_rate_mem_usage_request	%
内存利用率（占 Request，不包含 Cache）	%	Pod 内存使用占 Request 的比例，不含 Cache	Pods	k8s_pod_rate_mem_no_cache_request	%
内存利用率（占 Limit）	%	Pod 内存使用占 Limit 的比例	Pods	k8s_pod_rate_mem_usage_limit	%
内存利用率（占 Limit，不包含 Cache）	%	Pod 内存使用占 Limit 的比例，不含 Cache	Pods	k8s_pod_rate_mem_no_cache_limit	%

GPU

指标名称（控制台）	单位（控制台）	备注	type	metricName	默认单位
GPU 使用量	CUDA Core	Pod GPU 使用量	Pods	k8s_pod_gpu_used	CUDA Core
GPU 申请量	CUDA Core	Pod GPU 申请量	Pods	k8s_pod_gpu_request	CUDA Core
GPU 利用率（占 Request）	%	GPU 使用占 Request 的比例	Pods	k8s_pod_rate_gpu_used_request	%
GPU 利用率（占节点）	%	GPU 使用占 node 的比例	Pods	k8s_pod_rate_gpu_used_node	%

GPU memory 使用量	MiB	Pod GPU memory 使用量	Pods	k8s_pod_gpu_memory_used_bytes	B
GPU memory 申请量	MiB	Pod GPU memory 申请量	Pods	k8s_pod_gpu_memory_request_bytes	B
GPU memory 利用率（占 Request）	%	GPU memory 使用占 Request 的比例	Pods	k8s_pod_rate_gpu_memory_used_request	%
GPU memory 利用率（占节点）	%	GPU memory 使用占 node 的比例	Pods	k8s_pod_rate_gpu_memory_used_node	%

通过 YAML 创建和编辑 HPA

您可以通过 YAML 文件创建和编辑 HPA。以下为配置文件的示例，该文件定义了一条名称为 example 的 HPA，CPU 使用量为1时触发 HPA ，实例范围为1－2。

> **注意：**
> TKE 同样兼容原生的 Resource 类型。

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: example
  namespace: default
  labels:
    qcloud-app: example
spec:
  minReplicas: 1
  maxReplicas: 2
  metrics:
  - type: Pods  # 支持使用 Resource
    pods:
      metric:
        name: k8s_pod_cpu_core_used
      target:
        averageValue: "1"
        type: AverageValue
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: nginx
```

定时水平伸缩（HPC）

最近更新时间：2024-10-29 15:56:52

概述

实例（Pod）定时水平扩缩容功能（Horizontal Pod CronScaler，HPC）根据预设的时间点，定时扩容、缩减服务的 Pod 数量。

组件安装和使用流程

请参考 HPC 说明 了解详细的安装和使用流程。

HPC 和 HPA 的冲突解决

当 HPA（Horizontal Pod Autoscaler）和 HPC 同时作用于一个工作负载（workload）时，可能会出现规则冲突。具体表现为，workload 会按照 HPC 设定的时间进行扩缩容，同时也会根据 HPA 设定的当前使用率规则再次调整实例数。为避免冲突，建议 HPC 直接作用于 HPA，如下图所示：

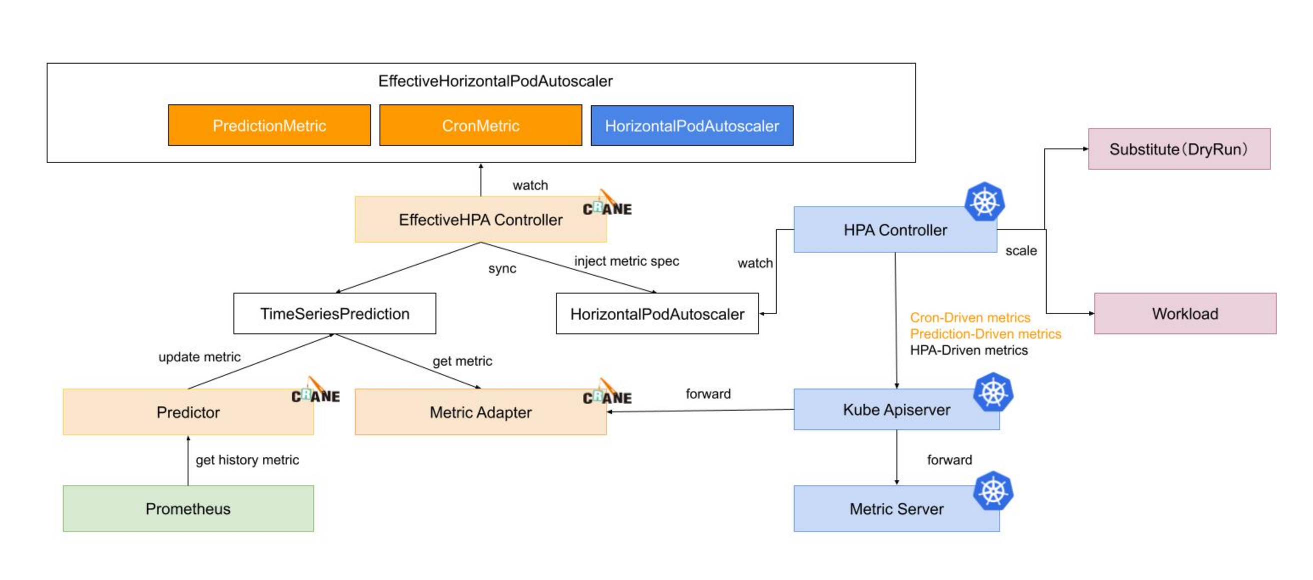
预测水平伸缩（EHPA）

最近更新时间：2024-10-29 15:56:52

简介

EffectiveHorizontalPodAutoscaler（简称 EHPA）是 Crane 提供的弹性伸缩产品，基于社区 HPA 进行底层弹性控制，支持更丰富的弹性触发策略（预测、观测、周期），让弹性更加高效，并保障了服务的质量。详情请参见 使用 EHPA 实现基于流量预测的弹性 。TKE 将该功能进行产品化包装，用户可以在控制台上一键安装配置 craned 组件，创建 EHPA，便捷实现基于流量预测的弹性伸缩。

工作原理

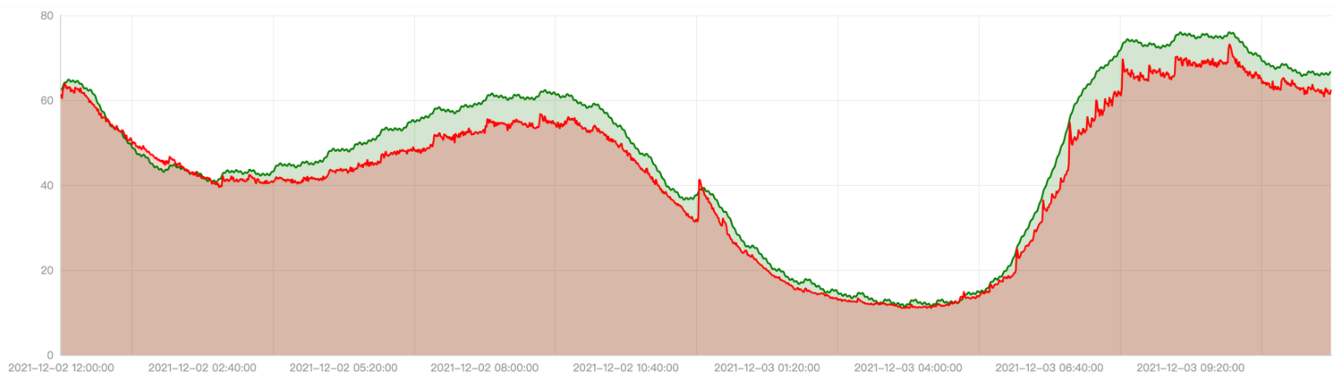


EHPA 的工作原理如下：

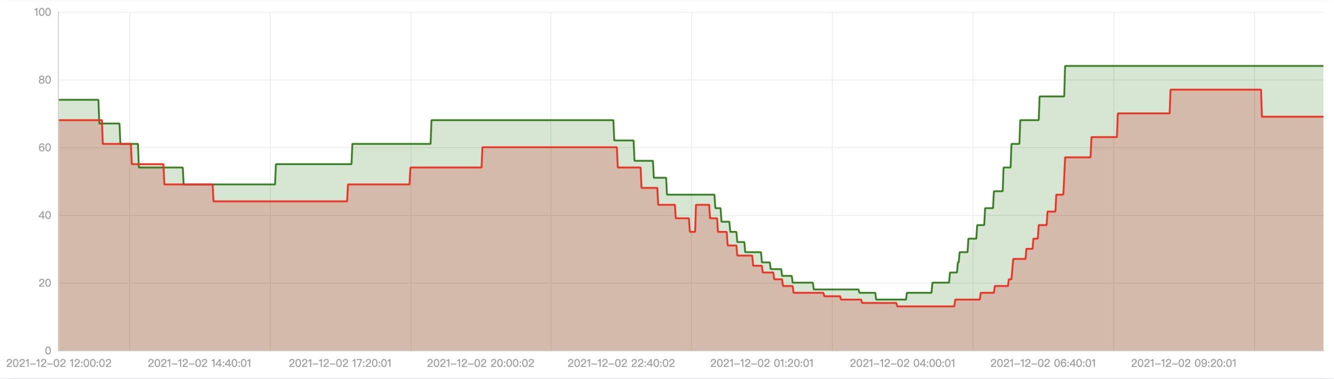
1. EffectiveHPAController 创建 HorizontalPodAutoscaler 和 TimeSeriesPrediction 对象。
2. PredictionCore 从 Prometheus 获取历史指标，通过预测算法计算，将结果记录到 TimeSeriesPrediction。
3. HPAController 通过 metric client 从 KubeApiServer 读取指标数据。
4. KubeApiServer 将请求路由到 Crane 的 MetricAdapter。
5. HPAController 计算所有指标返回的结果，得到最终的弹性副本推荐。
6. HPAController 调用 scale API 对目标应用进行扩/缩容。

预期效果

通过一个生产环境的客户案例介绍 EHPA 的实际效果。我们将生产数据在预发环境重放，对比使用 EHPA 和社区 HPA 的弹性效果。
下图的红线是应用在一天内的实际 CPU 使用量曲线，可以看到在8点，12点，晚上8点时是使用高峰。绿线是 EHPA 预测的 CPU 使用量。



下图是对应的自动弹性的副本数曲线，红线是社区 HPA 的副本数曲线，绿线是 EHPA 的副本数曲线。



EHPA 具有以下优势：

- 在流量高峰来临前提前扩容。
- 当流量先降后立刻升时，不进行无效缩容。
- 相比 HPA，弹性次数更少但更高效。

安装和配置

EHPAController 集成在 craned 中，同时依赖 Prometheus 组件获取历史指标。需要在组件管理中安装或升级最新版本的 craned，开启 EHPA 配置，并设置 Prometheus 访问地址和用户名、密码。

操作步骤如下：

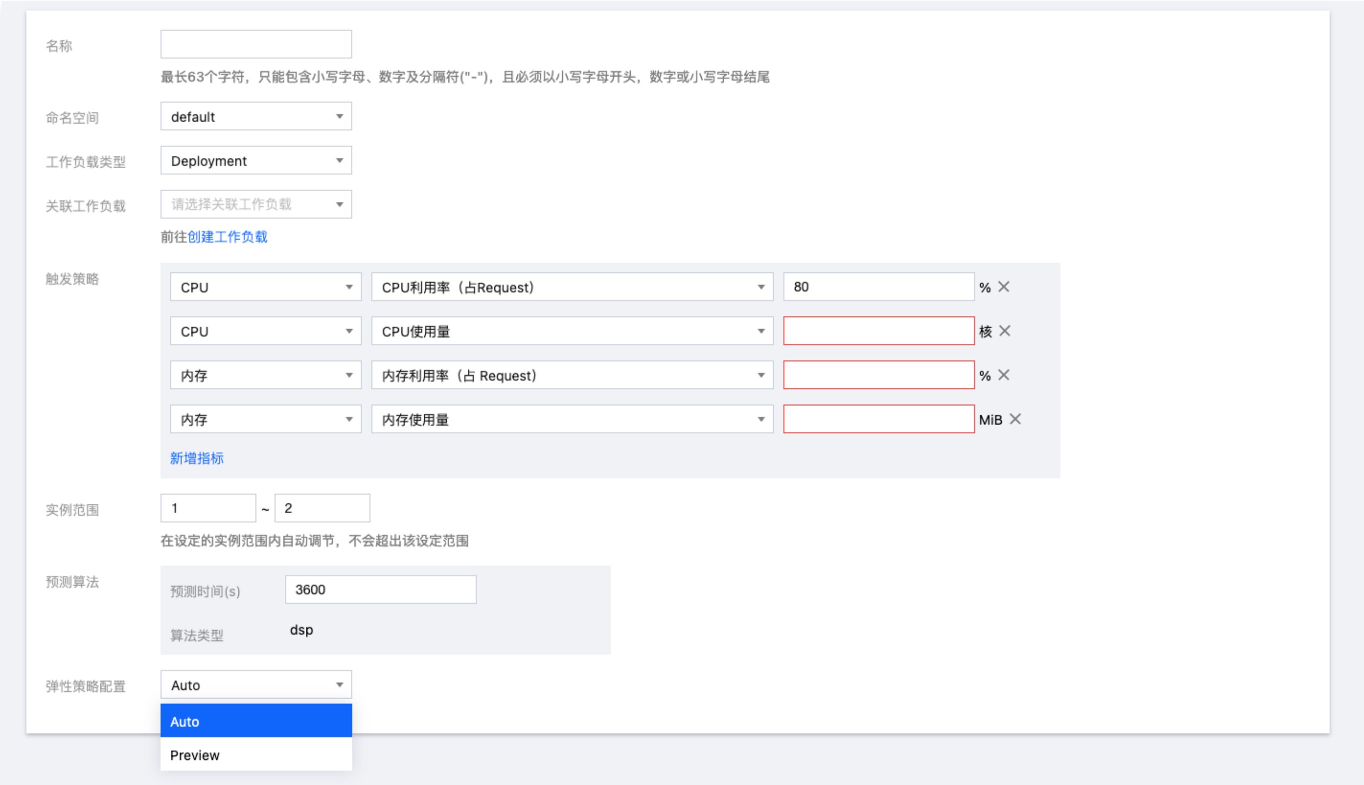
1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 选择左侧导航栏中的**运维中心** > **组件管理**。
4. 升级 craned 组件至最新版本，选择**更新配置**；或安装 craned 组件，选择**参数配置**。
5. 勾选**是否开启 ehpa**，设置 Prometheus 访问地址、用户名和密码，单击**确定**。如下图所示：

创建 EHPA

控制台创建

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 选择**运维中心** > **自动伸缩**，在 **EffectiveHorizontalPodAutoscaler** 页面单击**新建**。
4. 触发策略可选：
    - CPU：CPU 利用率（占 Request）、CPU 使用量。
    - 内存：内存利用率（占 Request）、内存使用量。
5. 弹性预测配置可选：
    - Auto：Auto 策略下 EHPA 会自动执行弹性行为。默认 EHPA 的策略是 Auto。在这个模式下 EHPA 会创建一个社区的 HPA 对象并自动接管它的生命周期。我们不建议用户修改或者控制这个底层的 HPA 对象，当 EHPA 被删除时，底层的 HPA 对象也会一并删除。

- Preview：Preview 策略提供了一种让 EHPA 不自动执行弹性的能力。用户可以通过 EHPA 的 desiredReplicas 字段观测 EHPA 计算出的副本数。用户可以随时在两个模式间切换，当用户切换到 Preview 模式时，可以通过 spec.specificReplicas 调整应用的副本数，如果 `spec.specificReplicas` 为空，则不会对应用执行弹性，但是依然会执行副本数的计算。



Yaml 创建

您也可以通过编写 Yaml 文件来创建 EHPA。以下是一个示例：

```
apiVersion: autoscaling.crane.io/v1alpha1
kind: EffectiveHorizontalPodAutoscaler
metadata:
  name: test
  namespace: default
spec:
  maxReplicas: 10
  minReplicas: 1
  metrics:
  - resource:
      name: cpu
      target:
        averageValue: "2"
        type: AverageValue
    type: Resource
  prediction:
    predictionAlgorithm:
      algorithmType: dsp
      dsp:
        estimators: {}
        historyLength: 7d
        sampleInterval: 60s
    predictionWindowSeconds: 3600
  scaleStrategy: Auto
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```
    name: php-apache
```

Service 管理

概述

最近更新时间：2024-10-21 14:57:11

Service 基本概念

用户在 Kubernetes 中可以部署各种容器，其中一部分是通过 HTTP、HTTPS 协议对外提供七层网络服务，另一部分是通过 TCP、UDP 协议提供四层网络服务。而 Kubernetes 定义的 Service 资源就是用来管理集群中四层网络的服务访问。
Kubernetes 的 ServiceTypes 允许指定 Service 类型，默认为 ClusterIP 类型。ServiceTypes 的可取值以及行为描述如下：

可取值	说明
ClusterIP	通过集群的内部 IP 暴露服务。当您的服务只需要在集群内部被访问时，请使用该类型。该类型为默认的 ServiceType。
NodePort	通过每个集群节点上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，该 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可从集群的外部访问该 NodePort 服务。除了测试以及非生产环境以外，不推荐在生产环境中直接通过集群节点对外甚至公网提供服务。从安全上考虑，使用该类型会直接暴露集群节点，容易受到攻击。通常认为集群节点是动态的、可伸缩的，使用该类型使得对外提供服务的地址和集群节点产生了耦合。
LoadBalancer	使用腾讯云的负载均衡器，可以向公网或者内网暴露服务。负载均衡器可以路由到 NodePort 服务，或直接转发到处于 VPC-CNI 网络条件下的容器中。

ClusterIP 和 NodePort 类型的 Service，在不同云服务商或是自建集群中的行为表现通常情况下相同。而 LoadBalancer 类型的 Service，由于使用了云服务商的负载均衡进行服务暴露，云服务商会围绕其负载均衡的能力提供不同的额外功能。例如，控制负载均衡的网络类型，后端绑定的权重调节等，详情请参见 Service 功能文档。

服务访问方式

根据上述 `ServiceTypes` 定义，您可以使用腾讯云容器服务 TKE 提供的以下四种服务访问方式：

访问方式	Service 类型	说明
公网	LoadBalancer	使用 Service 的 LoadBalancer 模式，公网 IP 可直接访问到后端的 Pod，适用于 Web 前台类的服务。 创建完成后的服务在集群外可通过负载均衡域名或 IP + 服务端口访问服务，集群内可通过服务名 + 服务端口访问服务。 注意：腾讯云负载均衡（Cloud Load Balancer）实例已于2023年03月06日升级了架构，升级后公网负载均衡以**域名**的方式提供服务。**VIP** 随业务请求动态变化，控制台不再展示 VIP 地址。请参见 域名化公网负载均衡上线公告。 • 新注册的腾讯云用户默认使用升级后的域名化负载均衡。 • 存量用户可以选择继续使用原有的负载均衡，不受升级影响。如果您需要升级负载均衡服务，则需要同时升级腾讯云产品 CLB 以及 TKE，否则 TKE 中的所有公网类型的 Service/Ingress 同步将可能受到影响。CLB 升级操作详情请参见 域名化负载均衡升级指南；TKE 升级 Service/Ingress 组件版本，请通过 在线咨询 联系我们。
VPC 内网	LoadBalancer	使用 Service 的 LoadBalancer 模式，指定注解 `service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: subnet-xxxxxxxx`，即可通过内网 IP 直接访问到后端的 Pod。 创建完成后的服务在集群外可通过负载均衡域名或 IP + 服务端口访问服务，集群内可通过服务名 + 服务端口访问服务。
主机端口访问	NodePort	提供一个主机端口映射到容器的访问方式，支持 TCP、UDP、Ingress。可用于业务定制上层 LB 转发到 Node。 创建完成后的服务可以通过云服务器 IP + 主机端口访问服务。
仅集群内访问	ClusterIP	使用 Service 的 ClusterIP 模式，自动分配 Service 网段中的 IP，用于集群内访问。数据库类等服务如 MySQL 可以选择集群内访问，以保证服务网络隔离。 创建完成后的服务可以通过服务名 + 服务端口访问服务。

负载均衡相关概念

Service 工作原理

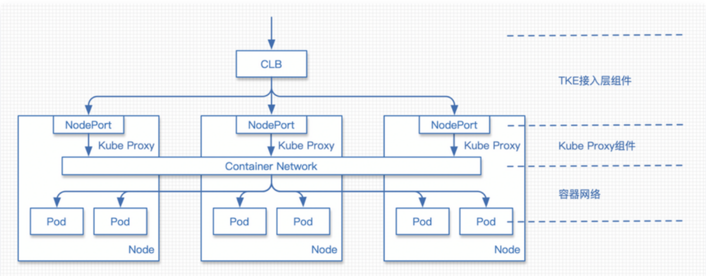
腾讯云容器集群中的 Service Controller 组件负责用户 Service 资源的同步。当用户创建、修改或删除 Service 资源时、集群节点或 Service Endpoints 出现变化时、组件容器发生飘移重启时，组件都会对用户的 Service 资源进行同步。
Service Controller 会依照用户 Service 资源的描述创建对应的负载均衡资源，并对监听器及其后端进行配置。当用户删除集群 Service 资源时，也会回收对应负载均衡资源。

Service 生命周期管理

Service 对外服务的能力依赖于负载均衡所提供的资源，服务资源管理也是 Service 的重要工作之一。Service 在资源的生命周期管理中会使用以下标签：
- `tke-createdBy-flag = yes`：标识该资源是由容器服务创建。
- `tke-clusterId = <ClusterId>`：标识该资源被哪一个 Cluster 所使用的。
- `tke-lifecycle-owner = tke|user`：标识 CLB 的控制权是 TKE 还是用户，如果值是 user，删除 Service 时不会删除负载均衡。

> **说明：**
> - `tke-createdBy-flag` 和 `tke-clusterId` 这两个标签对用户只读，请不要修改或删除这些标签，否则可能导致资源泄漏。
> - 若用户使用了已有负载均衡，则 Service 仅会使用该负载均衡，而不会删除该负载均衡。
> - 若用户将 `tke-lifecycle-owner` 置为 user，或者使用 私有连接，则删除 Service 时，不会删除该负载均衡。

当 LoadBalancer 类型的 Service 集群资源被创建时，对应负载均衡的生命周期就开始了。直到 Service 资源被删除或是负载均衡被重建时，负载均衡的生命周期就结束了。在此期间负载均衡会持续根据 Service 资源的描述进行同步。**当用户切换 Service 的网络访问时，例如公网 > Cluster IP、VPC 内网 > Cluster IP、Cluster IP 切换成使用的已有负载均衡，此类操作都会涉及到负载均衡的删除或销毁。**LoadBalancer 类型 Service 工作原理如下图所示：



Service 注意事项

- **请勿在负载均衡控制台手动删除 Service 所关联的 CLB 实例，也不能手动修改 CLB 的配置，一切 CLB 相关操作都应在 TKE 侧进行。**
- Service 有一个字段：`.spec.externalTrafficPolicy`。kube-proxy 基于 spec.internalTrafficPolicy 的设置来过滤路由的目标服务端点。当它的值设为 `Local` 时，只会选择节点本地的服务端点。当它的值设为 `Cluster` 或缺省时，Kubernetes 会选择所有的服务端点。更多请参见 Kubernetes 文档。**如果 Service 使用了 `Local` 方式，当 Pod 从 TKE 节点调度到超级节点，或者从超级节点调度到 TKE 节点的时候会出现断流，因为 Service 仅会选择本地的服务端点。**

Service 功能

Service 相关操作及功能如下，您可参考以下文档进一步了解：
- Service 基本功能
- Service 负载均衡配置
- Service 使用已有 CLB
- Service 后端选择
- Service 跨域绑定
- Service 优雅停机
- Pod 优雅删除
- 使用 LoadBalancer 直连 Pod 模式 Service
- 多 Service 复用 CLB
- Service 扩展协议
- Service Annotation 说明

参考资料

您也可以参考开源文档 Kubernetes Service，了解关于 Service 的更多信息。

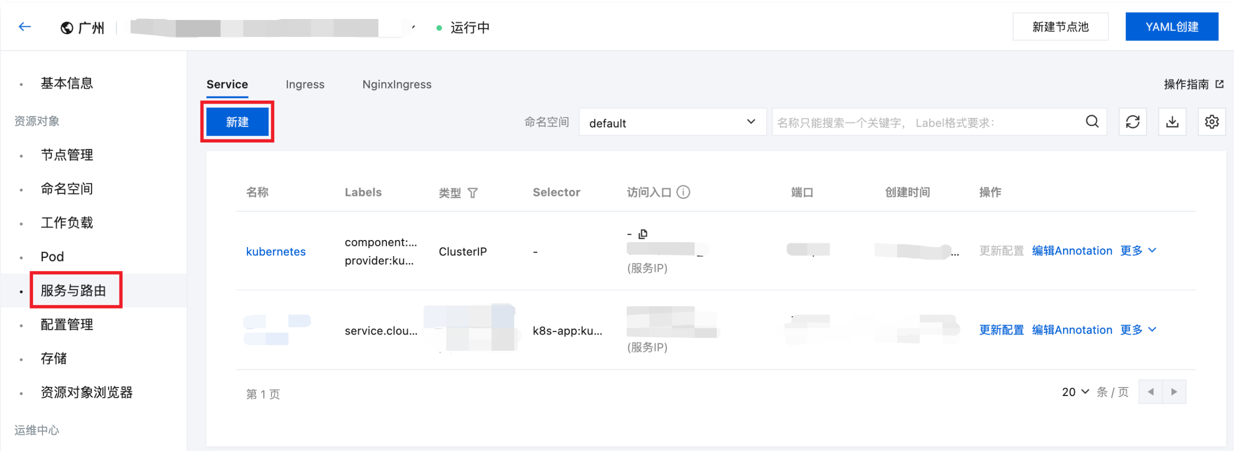
Service 基本功能

最近更新时间：2025-06-06 16:00:12

控制台操作指引

创建 Service

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击需要创建 Service 的集群 ID，进入集群基本信息页。
3. 选择**服务与路由 > Service**，在 **Service** 页面单击**新建**。如下图所示：



4. 在**新建服务与路由**页面，根据实际需求，设置 Service 参数。关键参数信息如下：
   - **服务名称**：自定义。
   - **命名空间**：根据实际需求进行选择。
   - **访问设置**：请参考 服务访问方式说明 进行设置。
   - （选填）**高级设置**：
     - **External TrafficPolicy**：
       - Cluster：默认均衡转发到工作负载的所有 Pod。
       - Local：能够保留来源 IP，并可以保证公网、VPC 内网访问（LoadBalancer）和主机端口访问（NodePort）模式下流量仅在本节点转发。Local 转发使部分没有业务 Pod 存在的节点健康检查失败，可能存在流量不均衡的转发的风险。

         > **说明：**
         > 如果 Service 使用了 Local 方式，当 Pod 从 TKE 节点调度到超级节点，或者从超级节点调度到 TKE 节点的时候会出现断流，因为 Service 仅会选择本地的服务端点。

     - **Session Affinity**：如果要确保来自特定客户端的连接每次都传递给同一个 Pod，您可以通过设置 Service 的 `.spec.sessionAffinity` 为 ClientIP 来设置基于客户端 IP 地址的会话亲和性（默认为 None）。
   - **Workload 绑定**：引用一个存量的 Workload，或自定义标签，该 Service 会根据自定义的标签选择拥有这些标签的 Workload。

     > **说明：**
     > - 如需使用已有负载均衡器，请参考 使用已有 CLB 。
     > - 由于4层 CLB 仅限制 **CLB VIP + 监听器协议 + 后端 RS VIP + 后端 RS 端口4元组唯一**，且未包含 CLB 监控端口。因此不支持 CLB 监听端口不同，协议及 RS 相同的场景。容器服务也不支持同一个业务对外开放相同协议的不同端口。

5. 单击**创建 Service**，完成创建。

更新 Service

> **说明：**

为防止 Service 在切换不同的 CLB 时发生资源异常问题，例如：CLB 可能会脱离 TKE 的管控导致资源泄漏，或无法创建出对应的 CLB 导致服务中断，TKE 对 Service 的生命周期的变更做了如下限制：

1. 禁止服务访问方式中的公网 LB 访问与内网 LB 访问之间互相切换。
2. 选择公网 LB 访问时，禁止切换当前 VPC 的可用区，以及其他 VPC；禁止在负载均衡器中的自动创建与使用已有之间互相切换。
3. 选择内网 LB 访问时，禁止在负载均衡器中的自动创建与使用已有之间互相切换，禁止切换 LB 所在子网。
4. 选择使用已有时，禁止切换成其他已有 CLB。
5. 禁止传统型和应用型 CLB 之间的切换。

若您尝试切换以上动作，前台会做限制，后台会拦截。若您有以上切换诉求，可以尝试：

1. 先将服务访问方式切换至仅在集群内访问或主机端口访问。
2. 再配置成目标的参数配置。

有关详细信息，请参见 公告 。

更新配置

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击集群 ID，进入集群基本信息页。
3. 选择**服务与路由** > **Service**，在 **Service** 页面单击 Service 所在行右侧的**更新配置**。如下图所示：

4. 在**更新访问方式**页面，根据实际需求进行访问设置。
5. 设置完成后，单击**更新访问方式**即可。

编辑 YAML

1. 选择**服务与路由** > **Service**，在 **Service** 页面单击 Service 所在行右侧的**更多** > **编辑 yaml**。
2. 在**编辑 Yaml** 页面，根据实际需求编辑 YAML 后单击**完成**即可。

删除 Service

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击集群 ID，进入集群基本信息页。
3. 选择**服务与路由** > **Service**，在 **Service** 页面单击 Service 所在行右侧的**更多** > **删除**。如下图所示：

Kubectl 操作 Service 指引

YAML 示例

```
kind: Service
apiVersion: v1
metadata:
  ## annotations:
  ##   service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: subnet-xxxxxxxx ##若是创建内网访问的
Service 需指定该条 annotation
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
  type: LoadBalancer
```

说明：
- **kind**：标识 Service 资源类型。
- **metadata**：Service 的名称、Label 等基本信息。
- **metadata.annotations**：Service 的额外说明，可通过该参数设置腾讯云容器服务的额外增强能力。
- **spec.selector**：该 Service 会根据这里标签选择器里的标签，选择拥有这些标签的 Workload。
- **spec.type**：标识 Service 的被访问形式。
  - **ClusterIP**：在集群内部公开服务，可用于集群内部访问。
  - **NodePort**：使用节点的端口映射到后端 Service，集群外可以通过节点 `IP:NodePort` 访问。
  - **LoadBalancer**：使用腾讯云提供的负载均衡器公开服务，默认创建公网负载均衡，指定 annotations 可创建内网负载均衡。
    - 默认用户可以创建的内网或外网的 CLB 数量分别是100个，如果您需要使用的数量超过100时，可通过 在线咨询 提升负载均衡 CLB 的配额。
    - Service 和 CLB 之间配置的管理和同步是由以 CLB ID 为名字的 LoadBalancerResource 类型的资源对象，请勿对该 CRD 进行任何操作，否则容易导致 Service 失效。
  - **ExternalName**：将服务映射到 DNS，仅适用于 kube-dns1.7及更高版本。

创建 Service

1. 参考 YAML 示例，准备 Service YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参见 通过 Kubectl 连接集群 。
3. 执行以下命令，创建 Service YAML 文件。

```
kubectl create -f Service YAML 文件名称
```

例如，创建一个文件名为 `my-service.yaml` 的 Service YAML 文件，则执行以下命令：

```
kubectl create -f my-service.yaml
```

4. 执行以下命令，验证创建是否成功。

```
kubectl get services
```

返回类似以下信息，即表示创建成功。

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   172.16.255.1   <none>        443/TCP   38d
```

更新 Service

方法1

执行以下命令，更新 Service。

```
kubectl edit service/[name]
```

方法2

1. 手动删除旧的 Service。
2. 执行以下命令，重新创建 Service。

```
kubectl create/apply -f [YAML FILE]
```

删除 Service

执行以下命令，删除 Service。

```
kubectl delete service [NAME]
```

注意事项

对于 LoadBalancer 类型的 Service，TKE 在对账过程中会给该 Service 加上一个随机注解 `service.cloud.tencent.com/client-token`，该注解用于实现创建 LoadBalancer 的幂等控制。
通常情况下该注解和一个具体的 LoadBalancer 一一对应，如果用户需要复制 Service 的内容用来重新创建一个 LoadBalancer，在 apply yaml 前，**用户需要在复制的 yaml 中删除掉该注解**，否则可能造成创建异常，甚至后续可能造成误删除老的 LoadBalancer。

Service 负载均衡配置

最近更新时间：2024-11-29 17:26:43

TkeServiceConfig

TkeServiceConfig 是腾讯云容器服务提供的自定义资源 CRD，通过 TkeServiceConfig 能够帮助您更灵活的配置 LoadBalancer 类型的 Service，及管理其中负载均衡的各种配置。

使用场景

Service YAML 的语义无法定义的负载均衡的参数和功能，可以通过 TkeServiceConfig 进行配置。

配置说明

使用 TkeServiceConfig 能够帮您快速进行负载均衡器的配置。通过 Service 注解 `service.cloud.tencent.com/tke-service-config:<config-name>`，您可以指定目标配置并应用到 Service 中。

> **注意：**
> TkeServiceConfig 资源需要与 Service 处于同一命名空间。

TkeServiceConfig 并不会帮您直接配置并修改协议和端口，您需要在配置中描述协议和端口以便指定配置下发的监听器。在一个 TkeServiceConfig 中可以声明多组监听器配置，目前主要针对负载均衡的健康检查以及对后端访问提供配置。

通过指定协议和端口，配置能够被准确的下发到对应监听器：

- `spec.loadBalancer.l4Listeners.protocol`：四层协议
- `spec.loadBalancer.l4Listeners.port`：监听端口

Service 与 TkeServiceConfig 关联行为

1. 创建 Loadbalancer 模式 Service 时，设置注解 `service.cloud.tencent.com/tke-service-config-auto: "true"`，将自动创建 <ServiceName>-auto-service-config。您也可以通过 **service.cloud.tencent.com/tke-service-config:<config-name>** 直接指定您自行创建的 TkeServiceConfig。两个注解不可同时使用，且手动指定的 `<config-name>` 不能以 `-auto-service-config` 与 `-auto-ingress-config` 为后缀。
2. 其中自动创建的 TkeServiceConfig 存在以下同步行为：
    - 用户修改 Service 默认的 TkeServiceConfig，TkeServiceConfig 内容同样会被应用到负载均衡（手动创建的 TkeServiceConfig 也遵循此行为）。
    - 更新 Service 资源时，新增若干四层监听器时，如果该监听器或转发规则没有对应的 TkeServiceConfig 配置片段。Service-Controller 将主动添加 TkeServiceConfig 对应片段。
    - 删除若干四层监听器时，Service-controller 组件将主动删除 TkeServiceConfig 对应片段。
    - 删除 Service 资源时，联级删除该 TkeServiceConfig。
3. 您也可以参考下列 TkeServiceConfig 完整配置参考自行创建需要的 CLB 配置，Service 通过注解：**service.cloud.tencent.com/tke-service-config:<config-name>** 引用该配置。
4. 其中您手动创建的 TkeServiceConfig 存在以下同步行为：
    - 当用户在 Service 中添加配置注解时，负载均衡将会立即进行设置同步。
    - 当用户在 Service 中删除配置注解时，负载均衡将会保持不变。
    - 修改 TkeServiceConfig 配置时，引用该配置 Service 的负载均衡将会根据新的 TkeServiceConfig 进行设置同步。
    - Service 的监听器未找到对应配置时，该监听器将不会进行修改。
    - Service 的监听器找到对应配置时，若配置中没有声明的属性，该监听器将不会进行修改。

完整配置参考

```
apiVersion: cloud.tencent.com/v1alpha1
kind: TkeServiceConfig
metadata:
  name: sample # 配置的名称
  namespace: default # 配置的命名空间
spec:
```

```
  loadBalancer:
    l4Listeners: # 四层规则配置，适用于Service的监听器配置。
    - protocol: TCP # 协议端口锚定Service的四层规则。必填，枚举值：TCP|UDP|TCP_SSL|QUIC。
      port: 80 # 必填，可选值：1~65535。
      deregisterTargetRst: true # 选填，布尔值。双向 RST 开关，建议非直连类型 Service 启用，非直连 Service 的连接会经过 NodePort 中转，不启用双向 RST 可能导致下线集群节点后造成业务中断。
      session: # 会话保持相关配置。选填
        enable: true # 是否开启会话保持。必填，布尔值
        sessionExpireTime: 100 # 会话保持的时间。选填，默认值：30，可选值：30~3600，单位：秒。
      healthCheck: # 健康检查相关配置。选填
        enable: true # 是否开启健康检查。必填，布尔值
        checkType: "TCP" # 健康检查类型。选填，枚举值：TCP|HTTP|CUSTOM（仅适用于TCP/UDP监听器，其中UDP监听器只支持CUSTOM；如果使用自定义健康检查功能，则必传）。
        checkPort: 80 # 健康检查端口，选填。默认为后端服务的端口，除非您希望指定特定端口，否则建议留空。（仅适用于TCP/UDP监听器）。
        intervalTime: 10 # 健康检查探测间隔时间。选填，默认值：5，可选值：5~300，单位：秒。
        healthNum: 2 # 健康阈值，表示当连续探测几次健康则表示该转发正常。选填，默认值：3，可选值：2~10，单位：次。
        unHealthNum: 3 # 不健康阈值，表示当连续探测几次健康则表示该转发异常。选填，默认值：3，可选值：2~10，单位：次。
        timeout: 10 # 健康检查的响应超时时间，响应超时时间要小于检查间隔时间。选填，默认值：2，可选值：2~60，单位：秒。
        httpCode: 31 # 健康检查状态码，选填，默认值：31，可选值：1~31。仅适用于HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式。 1 表示探测后返回值 1xx 代表健康，2 表示返回 2xx 代表健康，4 表示返回 3xx 代表健康，8 表示返回 4xx 代表健康，16 表示返回 5xx 代表健康。若希望多种返回码都可代表健康，则将相应的值相加。
        httpCheckPath: "/" # 健康检查路径，选填。仅适用于HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式。
        httpCheckDomain: "" # 健康检查域名，选填。默认为七层规则域名（仅适用于HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式）。
        httpCheckMethod: "HEAD" # 健康检查方法（仅适用于HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式），默认值：HEAD，可选值HEAD或GET。
        httpVersion: "HTTP/1.1" # 自定义探测相关参数。健康检查协议CheckType的值取HTTP时，必传此字段，代表后端服务的HTTP版本：HTTP/1.0、HTTP/1.1；（仅适用于TCP监听器）。
        sourceIpType: 0 # 健康检查探测来源。0(VIP为源IP) 1(100.64为源IP)。对于域名化clb默认值为1且只能为1，对于非域名化的clb默认值不一定，可在clb控制台配置页面看能否看到VIP探测方式，如能看到默认值为0，否则为1，更多详情查看https://cloud.tencent.com/document/product/214/86666
      scheduler: WRR # 请求转发方式配置。WRR、LEAST_CONN 分别表示按权重轮询、最小连接数。选填，枚举值：WRR|LEAST_CONN。
      proxyProtocol:
        enable: false # 启用 ProxyProtocol，仅对 TCP_SSL 和 QUIC 协议有效，CLB 申请开通白名单后可用，默认关闭。
```

示例

Deployment 示例：jetty-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: jetty
  name: jetty-deployment
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: jetty
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: jetty
    spec:
      containers:
      - image: jetty:9.4.27-jre11
        imagePullPolicy: IfNotPresent
        name: jetty
        ports:
        - containerPort: 80
          protocol: TCP
        - containerPort: 443
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

Service 示例：jetty-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/tke-service-config: jetty-service-config
    # 指定已有的 tke-service-config
    # service.cloud.tencent.com/tke-service-config-auto: "true"
    # 自动创建 tke-service-config
  name: jetty-service
  namespace: default
spec:
  ports:
  - name: tcp-80-80
    port: 80
    protocol: TCP
    targetPort: 80
  - name: tcp-443-443
    port: 443
    protocol: TCP
    targetPort: 443
  selector:
    app: jetty
  type: LoadBalancer
```

该示例中包含以下配置：

- Service 为公网 LoadBalancer 类型。声明了两个 TCP 服务，一个在80端口，一个在443端口。
- 使用了 `jetty-service-config` 负载均衡配置。

TkeServiceConfig 示例：jetty-service-config.yaml

```
apiVersion: cloud.tencent.com/v1alpha1
kind: TkeServiceConfig
metadata:
```

```
  name: jetty-service-config
  namespace: default
spec:
  loadBalancer:
    l4Listeners:
    - protocol: TCP
      port: 80
      deregisterTargetRst: true
      healthCheck:
        enable: false
    - protocol: TCP
      port: 443
      session:
        enable: true
        sessionExpireTime: 3600
      healthCheck:
        enable: true
        intervalTime: 10
        healthNum: 2
        unHealthNum: 2
        timeout: 5
      scheduler: WRR
```

该示例中包含以下配置：
名称为 `jetty-service-config` 。且在四层监听器配置中，声明了以下两段配置：

1. 80端口的 TCP 监听器将会被配置。关闭健康检查。
2. 443端口的 TCP 监听器将会被配置。
    - 打开健康检查，健康检查间隔调整为10s，健康阈值2次，不健康阈值2次，超时5s。
    - 打开会话保持功能，会话保持的超时时间设置为3600s。
    - 转发策略配置为：按权重轮询。

kubectl 配置命令

```
$ kubectl apply -f jetty-deployment.yaml
$ kubectl apply -f jetty-service.yaml
$ kubectl apply -f jetty-service-config.yaml

$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
jetty-deployment-8694c44b4c-cxscn   1/1     Running   0          8m8s
jetty-deployment-8694c44b4c-mk285   1/1     Running   0          8m8s
jetty-deployment-8694c44b4c-rjrtm   1/1     Running   0          8m8s

$ kubectl get service jetty-service
NAME    TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)                       AGE
jetty   LoadBalancer   10.127.255.209   150.158.220.237   80:31338/TCP,443:32373/TCP    2m47s

# 获取TkeServiceConfig配置列表
$ kubectl get tkeserviceconfigs.cloud.tencent.com
NAME                   AGE
jetty-service-config   52s

# 更新修改TkeServiceConfig配置
$ kubectl edit tkeserviceconfigs.cloud.tencent.com jetty-service-config
tkeserviceconfig.cloud.tencent.com/jetty-service-config edited
```

Service 使用已有 CLB

最近更新时间：2025-05-16 11:06:22

腾讯云容器服务 TKE 具备通过 `service.kubernetes.io/tke-existed-lbid: <LoadBalanceId>` 注解实现使用已有负载均衡的功能，您可使用该注解指定集群 Service 资源关联的负载均衡实例。还提供了 **Service 负载均衡复用**功能，即指定多个 Service 使用同一个已有负载均衡，您可参考本文进行设置。

使用已有负载均衡的同步行为

- 使用已有负载均衡时，指定 Service 的网络类型的注解不生效。
- 当 Service 不再使用已有负载均衡时，该 Service 描述的对应监听器会删除，该负载均衡将保留。
  删除监听器时，会校验监听器名称是否被修改。如果用户修改监听器名称，则认为该监听器可能由用户创建，不进行主动删除。
- 如果 Service 目前正在使用自动创建的负载均衡，那么给它添加使用已有负载均衡的注解，会使得当前负载均衡的生命周期结束并释放，Service 的配置将会与该负载均衡进行同步。反之，如果删除 Service 正在使用的已有负载均衡的注解，Service Controller 组件将会为该 Service 创建负载均衡并进行同步。

使用已有负载均衡同步腾讯云标签行为

- 默认情况下，Service 创建的 CLB 均会配置 `tke-createdBy-flag = yes` 标签，Service 会在销毁时删除对应资源。若使用已有 CLB，则不会配置该标签，Service 销毁时也不会删除对应资源。
- 所有 Service 均会配置 `tke-clusterId =` 标签，若 ClusterId 正确，则 Service 会在销毁时删除对应标签。
- 于2020年8月17日起创建的集群，将默认关闭多个 Service 复用相同 CLB 的功能。该日期前后集群内 Service 创建的 CLB 标签配置规则变更情况及详细信息，请参见 多 Service 复用 CLB。

注意事项

- 指定使用的负载均衡需和集群处于同一 VPC。
- 请确保您的容器业务不和云服务器 CVM 业务共用一个负载均衡。
- 不支持您在负载均衡控制台操作 TKE 所管理负载均衡的监听器和后端绑定的服务器，您的更改会被 TKE 的自动同步所覆盖。
- 使用已有的负载均衡时：
  - Service Controller 将不负责该已有负载均衡的释放与回收。
  - 仅支持使用通过负载均衡控制台创建的负载均衡器，不支持复用由 TKE 自动创建的负载均衡，**会破坏其他 Service 负载均衡的生命周期管理（例如，可能导致复用的 Service 被删除后，CLB 也随之删除）**。
- **复用**负载均衡时：
  - 不支持跨集群复用负载均衡。
  - 您需要使用**复用**功能时，建议有明确的监听器端口管理，否则负载均衡在多个 Service 的使用下，会出现管理混乱。
  - 复用负载均衡的端口冲突时，将会被拒绝。如果在修改中出现冲突，那么出现冲突的监听器后端同步无法确保正确。
  - 复用负载均衡的 Service 不支持开启 Local 访问（传统型负载均衡限制）。
  - 删除 Service，则复用负载均衡绑定的后端云服务器需要自行解绑，同时会保留一个 tag `tke-clusterId: cls-xxxx`，需自行清理。

Service 示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/tke-existed-lbid: lb-6swtxxxx
  name: nginx-service
spec:
  ports:
    - name: 80-80-no
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

```
  type: LoadBalancer
```

> **说明**
> - `service.kubernetes.io/tke-existed-lbid: lb-6swtxxxx` 注解表示该 Service 将使用已有负载均衡进行配置。
> - 请注意 Service 的类型，需设置为 `LoadBalancer` 类型。

使用场景 示例

使用包年包月的负载均衡对外提供服务

Service Controller 组件管理负载均衡生命周期时，仅支持购买按量计费的负载均衡资源。当用户需要长时间使用负载均衡时，包年包月计费模式在价格上有一定的优势。在此类场景下，用户就可以独立购买和管理负载均衡，再通过注解控制 Service 使用已有负载均衡，并将负载均衡的生命周期管理从 Service Controller 组件中剥离。

在同一端口暴露 TCP 和 UDP 服务

Kubernetes 官方在 Service 的设计中具有限制：一个 Service 下暴露的多个端口协议必须相同。有许多游戏场景下的用户，有在同一个端口同时暴露 TCP 和 UDP 服务的需求，腾讯云负载均衡服务支持在同一个端口上同时监听 UDP 和 TCP 协议，此需求可以通过 Service 负载均衡复用来解决。
例如以下 Service 配置，`game-service` 被描述为两个 Service 资源，描述的内容除了监听的协议以外基本相同。两个 Service 都通过注解指定使用已有负载均衡 `lb-6swtxxxx`。通过 kubectl 将以上资源应用到集群中，就可以实现在同一个负载均衡的端口上暴露多种协议的目的。

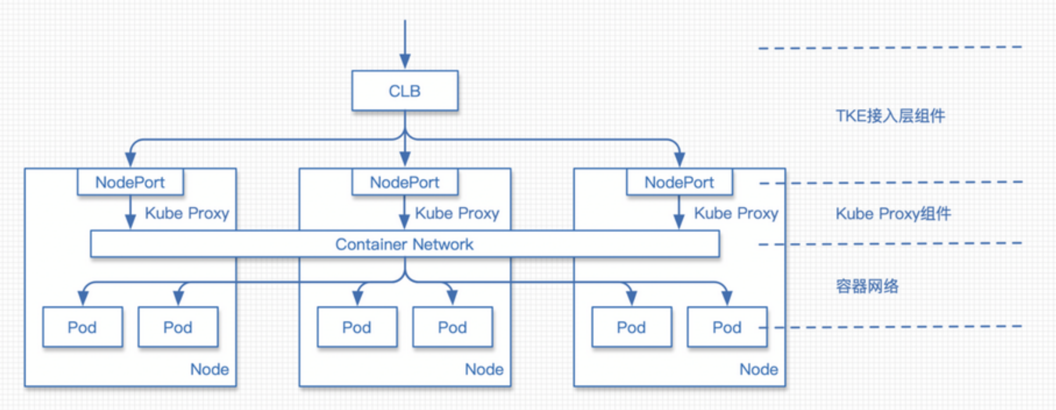
```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/tke-existed-lbid: lb-6swtxxxx
  name: game-service-a
spec:
  ports:
    - name: 80-80-tcp
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: game
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/tke-existed-lbid: lb-6swtxxxx
  name: game-service-b
spec:
  ports:
    - name: 80-80-udp
      port: 80
      protocol: UDP
      targetPort: 80
  selector:
    app: game
  type: LoadBalancer
```

Service 后端选择

最近更新时间：2023-11-16 15:56:42

默认后端选择

默认情况下，Service 会配置负载均衡的后端到集群节点的 NodePort，如下图 TKE 接入层组件部分。此方案具有非常高的容错性，流量从负载均衡到任何一个 NodePort 之后，NodePort 会再一次随机选择一个 Pod 将流量转发过去。同时这也是 Kubernetes 官方提出的最基础的网络接入层方案。如下图所示：



`TKE Service Controller` 默认不会将以下节点作为负载均衡后端：
- Master 节点（不允许 Master 节点参与网络接入层的负载）。
- 节点状态为 NotReady（节点不健康）。

> **注意**
> `TKE Service Controller` 可以绑定状态为 Unschedulable 的节点。Unschedulable 的节点也可以作为流量的入口，因为流量进入到节点之后，会再做一层容器网络里的流量转发，流量在 Unschedulable 的节点里面不会被丢弃，如上图所示。

指定接入层后端

对于一些规模很大的集群，Service 管理的负载均衡会挂载几乎所有集群节点的 NodePort 作为后端。此场景存在以下问题：
- 负载均衡的后端数量有数量限制。
- 负载均衡会对每一个 NodePort 进行健康检查，所有健康检查都会请求到后端的工作负载上。

此类问题可通过以下方式进行解决：

在一些大规模集群的场景中，用户可以通过 `service.kubernetes.io/qcloud-loadbalancer-backends-label` 指定一部分节点进行绑定。
`service.kubernetes.io/qcloud-loadbalancer-backends-label` 的内容是一个标签选择器，用户可以通过在集群节点上标记 Label，然后在 Service 中通过该注解描述的标签选择器，选择匹配的节点进行绑定。这个同步会持续进行，当节点发生变化导致其被选择或是不再被选择时，Service Controller 会对应添加或删除负载均衡上的对应后端。详情请参见 Kubernetes 标签与选择器。

注意事项

- 当 `service.kubernetes.io/qcloud-loadbalancer-backends-label` 的选择器没有选取到任何节点的时候，服务的后端将会被排空，会使得服务中断。使用此功能时，需要对集群节点的 Label 有一定的管理。
- 新增符合要求的节点或变更存量节点也会触发 controller 更新。

使用场景

大规模集群下的测试应用

在一个大规模集群下，部署一个仅包含一两个 Pod 的测试应用。通过 Service 进行服务暴露时，负载均衡将对所有的后端 NodePort 进行健康检查，此健康检查的请求量对测试应用有很大影响。此时可以在集群中通过 Label 指定一小部分节点作为后端，缓解健康检查带来的压力。详情请参见 关于健康检查探测频率过高的说明。

示例

```
apiVersion: v1
kind: Service
```

```
metadata:
  annotations:
    service.kubernetes.io/qcloud-loadbalancer-backends-label: "group=access-layer"
  name: nginx-service
spec:
  ports:
    - name: 80-80-no
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

该示例包含以下配置：
- 描述了一个公网类型负载均衡的服务暴露。
- `service.kubernetes.io/qcloud-loadbalancer-backends-label` 注解声明了后端选择器，仅支持集群节点上有 `group=access-layer` Label 的节点才会作为这个负载均衡的后端。

Service Local 模式

Kubernetes 提供了 Service 特性 `ExternalTrafficPolicy`。当 `ExternalTrafficPolicy` 设置为 Local 时，可以避免流量通过 NAT 在节点间的转发，减少了 NAT 操作也使得源 IP 得到了保留。NodePort 仅会将流量转发到当前节点的 Pod。Local 模式特点如下：

优点：

- 避免了 NAT 与节点间转发带来的性能损失。
- 为服务端保留了请求来源 IP。

缺点：

没有工作负载的节点，NodePort 将无法提供服务。

注意事项

- Service Local 模式本身就具有节点选择能力，所以：
  - 请不要在 Service Local 模式下，使用指定接入层后端的能力。
  - 请不要在 Service Local 模式下，在 Node 上使用 `node.kubernetes.io/exclude-from-external-load-balancers` Label驱逐后端节点。
- 负载均衡的同步是需要时间的。当 Local 类型的服务工作负载数量很少时，工作负载的飘移或滚动更新会很快。此时后端如未来得及同步，后端的服务可能会出现不可用的情况。
- 仅适用于处理低流量、低负载的业务，不建议在生产环境中使用。

示例：Service 开启 Local 转发（externalTrafficPolicy: Local）

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  externalTrafficPolicy: Local
  ports:
    - name: 80-80-no
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Local 默认后端选择

默认情况下，当 Service 开启 Local 模式之后，仍会按默认方式挂载几乎所有节点的 NodePort 作为后端。负载均衡会根据健康检查的结果，避免流量进入没有工作负载的后端节点。为了避免这些没有工作负载的后端被绑定，用户可以通过 `service.kubernetes.io/local-svc-only-bind-node-with-pod: "true"` 注解，在 Local 模式下指定绑定有工作负载节点作为后端。更多信息请参考 Kubernetes Service Local。

示例：Service 开启 Local 转发并开启 Local 绑定

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/local-svc-only-bind-node-with-pod: "true"
  name: nginx-service
spec:
  externalTrafficPolicy: Local
  ports:
    - name: 80-80-no
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

由于 Local 模式下，进入节点的请求流量不会在节点间转发。所以当节点上的工作负载数量不一致的时候，同样的后端权重可能会使得每一个节点上的负载不平均。此时用户可以通过 `service.cloud.tencent.com/local-svc-weighted-balance: "true"` 进行加权平衡。使用此注解时，NodePort 后端的权重将由节点上工作负载的数量决定，从而避免不同节点上工作负载数量不同带来的负载不均的问题。其中，**Local 加权平衡必须和 Local 绑定同时使用**。示例如下：

示例：Service 开启 Local 转发，并开启 Local 绑定与 Local 加权平衡

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/local-svc-only-bind-node-with-pod: "true"
    service.cloud.tencent.com/local-svc-weighted-balance: "true"
  name: nginx-service
spec:
  externalTrafficPolicy: Local
  ports:
    - name: 80-80-no
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Service 跨 VPC 绑定

最近更新时间：2025-04-24 16:34:12

简介

使用公网 CLB 型 Service 时，默认是在当前集群所在 VPC 内的随机可用区生成 CLB，目前 TKE 的公网 CLB Service 已支持指定可用区，包括其他地域的可用区。本文将为您介绍如何通过控制台和 YAML 两种方式为 CLB Service 跨 VPC 绑定和指定可用区。

应用场景

- 需要支持 CLB 的跨地域接入或跨 VPC 接入，即 CLB 所在的 VPC 和当前集群所在的 VPC 不在同一 VPC 内。
- 需要指定 CLB 的可用区以实现资源的统一管理。

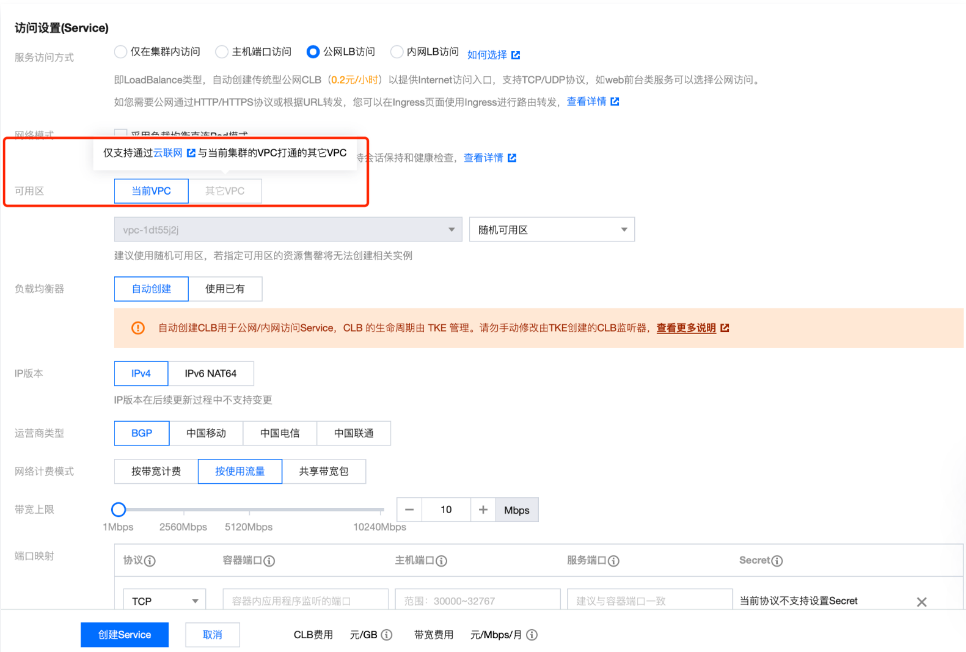
> **说明：**
> 1. 跨 VPC 绑定仅支持“带宽上移账户”。若您无法确定账户类型，请参见 判断账户类型 。
> 2. 如需使用非本集群所在 VPC 的 CLB，需先通过 云联网 打通当前集群 VPC 和 CLB 所在的 VPC。
>   2.1 云联网需要提前规划各个地域 VPC 的网段，不能出现冲突。否则冲突的路由规则将不会生效，导致数据面无法转发。
>   2.2 集群所在 VPC 不能同时加入多个云联网中，否则会出现路由不唯一的情况，导致数据面无法转发。
> 3. 在确保 VPC 已经打通之后，请 在线咨询 申请使用该功能。
> 4. 以下 YAML 中，需要您输入地域 ID，您可以通过 地域和可用区 查看地域 ID。

操作步骤

公网 CLB Service 跨 VPC 绑定和指定可用区支持通过控制台和 YAML 两种方式进行操作，操作步骤如下：

控制台方式

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面单击需要创建 Service 的集群 ID，进入集群管理页面。
3. 选择**服务与路由** > **Service**，进入 Service 页面，并单击**新建**。
4. 在**新建 Service** 页面中配置相关可用区规则。配置规则说明如下：
   - **服务访问方式**：选择“公网 LB 访问”。



   - **可用区**：选择“其他 VPC”。

> 注意：
> 仅支持通过 云联网 将当前集群的 VPC 打通的其它 VPC。

5. 其他参数按照 Service 创建时的配置即可。

YAML 方式

> 说明：
> 1. 如需使用非本集群所在 VPC 的 CLB，需先通过 云联网 打通当前集群 VPC 和 CLB 所在的 VPC。
> 2. 在确保 VPC 已经打通之后，请 在线咨询 申请使用该功能。

示例1

如果仅需指定本集群所在 VPC 的可用区，例如集群的 VPC 在广州地域，CLB Service 需要指定广州一区的 CLB，可以在 Service 的 YAML 中添加如下 annotation：

```
service.kubernetes.io/service.extensiveParameters: '{"ZoneId":"ap-guangzhou-1"}'
```

示例2

如需使用非本集群所在 VPC 内的 CLB，可以在 Service 的 YAML 中添加如下 annotation：

```
service.cloud.tencent.com/cross-region-id: "ap-guangzhou"
service.cloud.tencent.com/cross-vpc-id: "vpc-646vhcjj"
```

> 注意：
> 如果您还需要指定可用区，需要再添加示例1中的 annotation。

示例3

选择已有负载均衡进行异地接入，示例如下：

```
service.cloud.tencent.com/cross-region-id: "ap-guangzhou"
service.kubernetes.io/tke-existed-lbid: "lb-342wppll"
```

示例4

annotation 在 Service YAML 中的写法如下所示：

```
# 创建异地接入的负载均衡
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/cross-region-id: "ap-chongqing"
    service.cloud.tencent.com/cross-vpc-id: "vpc-mjekzyps"
  name: echo-server-service
  namespace: default
spec:
  ......
---
# 用户复用其他地域负载均衡的场景
```

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/cross-region-id: "ap-chongqing"
    service.kubernetes.io/tke-existed-lbid: "lb-o8ugf2wb"
  name: echo-server-service
  namespace: default
spec:
  ......
```

完整 Service Annotation 说明请参见 Service Annotation 说明 文档。

Service 优雅停机

最近更新时间：2024-12-04 16:34:51

简介

基于接入层直连 Pod 的场景，当后端进行滚动更新或后端 Pod 被删除时，如果直接将 Pod 从 LB 的后端摘除，则无法处理 Pod 已接收但还未处理的请求。特别是长链接的场景，例如会议业务，如果直接更新或删除工作负载的 Pod，此时会议会直接中断。

应用场景

- 更新工作负载时，Pod 的优雅退出，使客户端不会感受到更新时产生的抖动和错误。
- 当 Pod 需要被删除时，Pod 能够处理完已接收到的请求，此时入流量关闭，但出流量仍能走通。直到处理完所有已有请求和 Pod 真正删除时，出入流量才进行关闭。

> **注意：**
> 1. 仅针对 直连场景 生效，请检查您的集群是否支持直连模式。
> 2. 如果是多 Service 复用相同的 CLB，且同时更新多个 Service 对应的后端服务，可能导致优雅停机功能受影响，详情见 多 Service 复用 CLB 中的说明事项。

操作步骤

步骤1：使用 Annotation 标明使用优雅停机

以下为使用 Annotation 标明使用优雅停机示例，完整 Service Annotation 说明可参见 Service Annotation 说明 。

```
kind: Service
apiVersion: v1
metadata:
  annotations:
    service.cloud.tencent.com/direct-access: "true" ## 开启直连 Pod 模式
    service.cloud.tencent.com/enable-grace-shutdown: "true"  # 表示使用优雅停机
  name: my-service
spec:
  selector:
    app: MyApp
```

步骤2：使用 preStop 和 terminationGracePeriodSeconds

步骤2为在需要优雅停机的工作负载里配合使用 preStop 和 terminationGracePeriodSeconds。

容器终止流程

以下为容器在 Kubernetes 环境中的终止流程：

1. Pod 被删除，此时 Pod 里有 DeletionTimestamp，且状态置为 Terminating。此时调整 CLB 到该 Pod 的权重为 0。
2. kube-proxy 更新转发规则，将 Pod 从 service 的 endpoint 列表中摘除掉，新的流量不再转发到该 Pod。
3. 如果 Pod 配置了 preStop Hook ，将会执行。
4. kubelet 将对 Pod 中各个 container 发送 SIGTERM 信号，以通知容器进程开始优雅停止。
5. 等待容器进程完全停止，如果在 terminationGracePeriodSeconds 内 (默认30s) 还未完全停止，将发送 SIGKILL 信号强制停止进程。
6. 所有容器进程终止，清理 Pod 资源。

具体操作步骤

1. **使用 preStop**
   要实现优雅终止，务必在业务代码里处理 SIGTERM 信号。主要逻辑是不接受新的流量进入，继续处理存量流量，所有连接全部断开才退出，了解更多可参见 示例 。

若您的业务代码中未处理 SIGTERM 信号，或者您无法控制使用的第三方库或系统来增加优雅终止的逻辑，也可以尝试为 Pod 配置 preStop，在其实现优雅终止的逻辑，示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
          - /clean.sh
...
```

更多关于 preStop 的配置请参见 Kubernetes API 文档。
在某些极端情况下，Pod 被删除的一小段时间内，仍然可能有新连接被转发过来，因为 kubelet 与 kube-proxy 同时 watch 到 Pod 被删除，kubelet 有可能在 kube-proxy 同步完规则前就已经停止容器，这时可能导致一些新的连接被转发到正在删除的 Pod，而通常情况下，当应用受到 SIGTERM 后都不再接受新连接，只保持存量连接继续处理，因此可能导致 Pod 删除的瞬间部分请求失败。
针对上述情况，可以利用 preStop 先 sleep 短暂时间，等待 kube-proxy 完成规则同步再开始停止容器内进程。示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
          - sleep
          - 5s
```

2. **使用 terminationGracePeriodSeconds 调整优雅时长**
如果需要优雅终止时间较长（preStop + 业务进程停止可能超过30s），可根据实际情况自定义 terminationGracePeriodSeconds，避免过早的被 SIGKILL 停止，示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: grace-demo
spec:
  terminationGracePeriodSeconds: 60 # 优雅停机默认30s，您可以设置更长的时间
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
```



```
          - sleep
          - 5s
...
```

相关能力

优雅停机只是在 Pod 删除时，才把 CLB 后端的权重置为 0。若 Pod 在运行的过程中，出现了不健康的情况，此时将该后端的权重置为 0，可以减少服务不可用的风险。
您可以使用 Annotation： `service.cloud.tencent.com/enable-grace-shutdown-tkex: "true"` 实现这样优雅退出的能力。
该 Annotation 会根据 Endpoint 对象中 endpoints 是否 not-ready，将 not-ready 的 CLB 后端权重置为 0。

相关文档

故障处理：Nginx Ingress Controller 后端解绑不优雅的问题

Node 优雅下线

最近更新时间：2024-12-17 16:21:22

简介

在接入层非直连 Pod 的场景中，Node 作为实际的中间转发者，将流量转发到 Pod。如果直接删除 Node，在大规模场景下（例如多个 Service 共享一个 CLB，且下线的 Node 数量较多），Node 的删除速度可能会快于 CLB 解绑后端服务（RS）的速度。在这种情况下，Node 上的转发组件已经被移除，但 Node 仍然作为后端服务绑定在 CLB 上。此时，CLB 会将流量转发给这些已经失去转发能力的 Node。特别是在长连接场景下，例如会议业务或 RPC 长连接等，这一过程可能会持续到长连接超时为止。

应用场景

- 节点下线时，Node 优雅退出，使长连接在 Node 被销毁前解绑并收到 RST，及时重新建立连接。
- 当节点下线时，保证节点下线动作在所有的 CLB 解绑该 Node 之后。

> **注意：**
> - 仅针对 “非直连场景” 生效。
> - 在 TKE 层的 Node 被删除前，不能在云服务器控制台删除实际对应的机器。
> - 该功能目前存在不支持的场景，详情请参见 生效范围 。
> - 长连接业务，需要能够感知 RST（Connection Reset），及时重新建立连接。
> - 开启该功能后，接入层组件会为所有作为 CLB 后端的 Node 打上保护性 Finalizer，这一过程需要 10 分钟时间，在这段时间内不要操作缩容。

支持版本

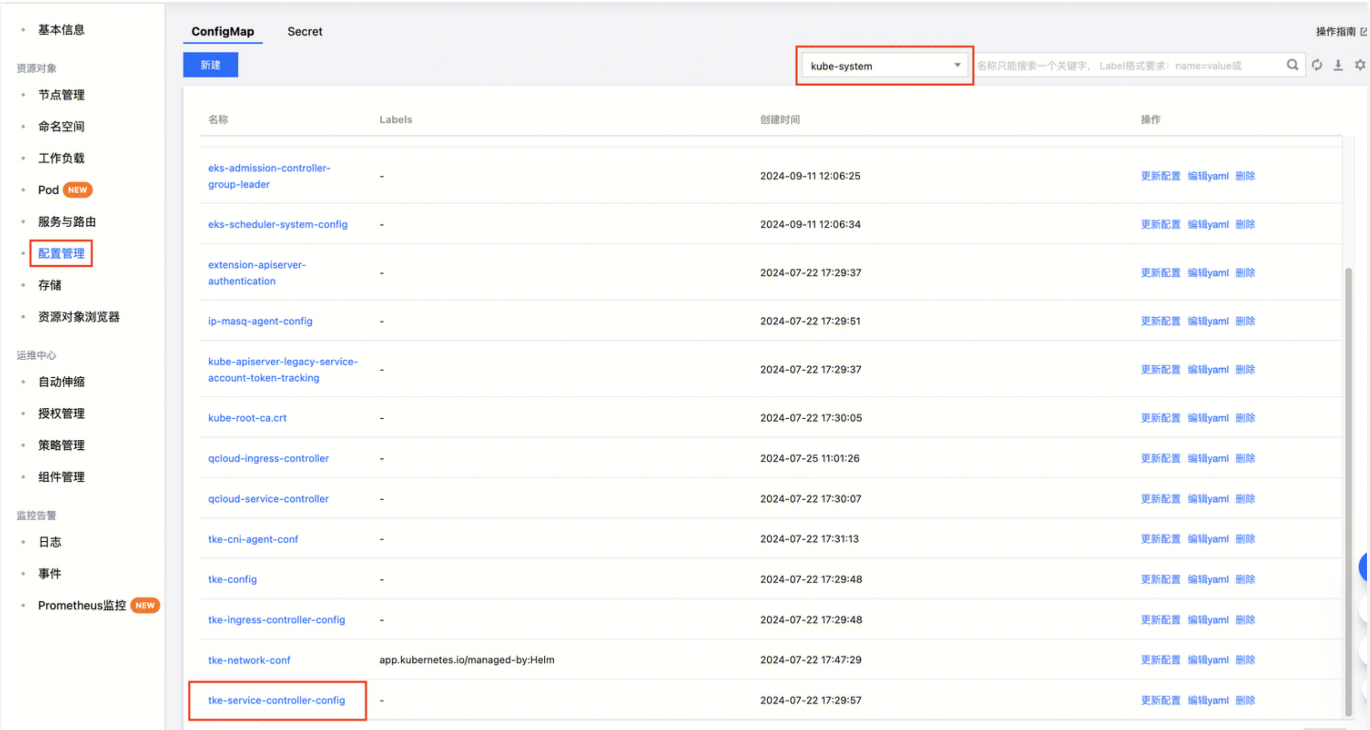
节点优雅下线接入层需要的最低版本：
- service-controller（针对 Service 场景）：v2.4.2
- ingress-controller（针对 Ingress 场景）：v2.4.2

> **说明：**
> 您可以分别通过查看 kube-system 下名为 tke-service-controller-config 和 tke-ingress-controller-config 的 ConfigMap 中的 VERSION 字段的值来确认 service-controller 和 ingress-controller 版本。

操作步骤

方式1：通过控制台

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，单击集群 ID，进入集群基本信息页。
3. 选择**配置管理**，在 ConfigMap 中，找到 kube-system 命名空间下的名称为 tke-service-controller-config 的 ConfigMap 资源。



4. 单击**更新配置**，找到 `EnableNodeGracefulDeletion` 开关变量，把 `EnableNodeGracefulDeletion` 对应的变量值置为 true。

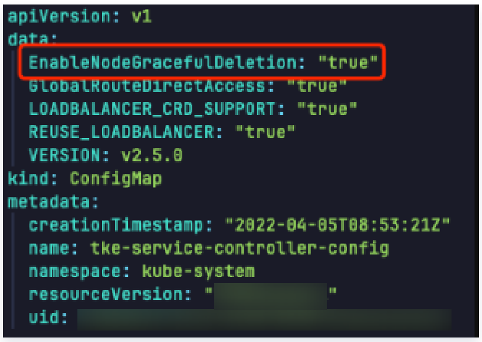
方式2：通过命令行

1. 查看 `kube-system` 命名空间下的 `tke-service-controller-config` ConfigMap 资源：

```
kubectl -n kube-system get configmap tke-service-controller-config -o yaml
```

2. 修改 tke-service-controller-config 这个 ConfigMap，将 `EnableNodeGraceDeletion` 置为 true：

```
kubectl -n kube-system edit configmap tke-service-controller-config
```



生效范围

节点类型	操作方式	是否支持
普通节点池	直接删除节点池	No
	单击调整数量	No
	选中一批节点，单击移出	Yes
	自动扩缩容	Yes
普通节点	单击移出	Yes
原生节点池	所有操作	No

说明：
超级节点不存在“非直连”模式，不需要支持该能力。

Pod 优雅删除

最近更新时间：2024-12-26 10:26:32

操作背景

在 CLB 直连场景下，当用户发起工作负载滚动更新时，会同时触发两个并行的工作流：

1. Service/Ingress 控制器调整 CLB 后端 Pod 的权重为0，并在后续摘除该 Pod。
2. 运行时（kubelet）终止并删除 Pod。

在某些极端场景，比如有多个 Service 复用同一个 CLB，或者一个 Service 下有非常多的 Pod，可能出现某些 Pod 已经被运行时删除了，但 Service 控制器还没有完成 CLB 上该 Pod 的权重调整，进而导致流量还会发往不存在的 Pod IP，导致业务异常。

应用场景

TKE Service/Ingress 的优雅删除特性，会将上述两个并行的工作流，协调为串行，也就是先让 Service/Ingress 控制器完成 CLB 后端 Pod 权重调整为0，再让运行时（kubelet）删除该 Pod。

> 注意：
> 1. 仅适用于 直连场景 。
> 2. 对节点 kubelet 版本有要求，具体版本请参见 TKE Kubernetes Revision 版本历史，确保集群节点版本支持 kubelet 支持 pod 删除保护功能。
> 3. 该功能对 TKE Service/Ingress 组件版本有要求，如需使用请 提交工单 咨询，检查集群是否支持该功能。

操作步骤

步骤1：确保集群节点版本和 Service 组件版本支持该功能

- kubelet 版本：参考 TKE Kubernetes Revision 版本历史，确保集群节点版本支持 kubelet 的 Pod 删除保护功能。
- Service/Ingress 组件版本：请 提交工单 咨询确认版本。
- 如果是超级节点，请 提交工单 咨询确认。

步骤2：使用注解对指定的 Service/Ingress 开启 Pod 优雅删除

Service YAML 配置示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: anyserver
  annotations:
    service.cloud.tencent.com/direct-access: "true"
    service.cloud.tencent.com/enable-grace-deletion: "true" # 表示开启 Pod 优雅删除
```

Ingress YAML 配置示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: anyserver
  annotations:
    ingress.cloud.tencent.com/direct-access: "true"
    ingress.cloud.tencent.com/enable-grace-deletion: "true"
```

使用 LoadBalancer 直连 Pod 模式 Service

最近更新时间：2024-11-07 16:54:32

操作场景

原生 LoadBalancer 模式 Service 可自动创建负载均衡 CLB，并通过集群的 Nodeport 转发至集群内，再通过 iptables 或 ipvs 进行二次转发。该模式下的 Service 能满足大部分使用场景，但在以下场景中更推荐使用**直连 Pod 模式 Service**：

- 有获取来源 IP 需求时（非直连模式必须另外开启 Local 转发）。
- 要求具备更高转发性能时（非直连模式下 CLB 和 Service 本身存在两层 CLB，性能有一定损失）。
- 需使用完整的健康检查和会话保持到 Pod 层级时（非直连模式下 CLB 和 Service 本身存在两层 CLB，健康检查及会话保持功能较难配置）。

> **说明：**
> - 若您的集群是 Serverless 集群，则默认为直连 Pod 模式，您无需任何操作。
> - 当前 GlobalRouter 和 VPC-CNI 容器网络模式均支持直连 Pod 模式，您可以在 集群列表 中单击集群 ID 进入集群详情页面，在集群的“基本信息”页面中查看当前集群使用的网络插件。

容器网络模式为 VPC-CNI

使用限制

- 集群 Kubernetes 版本需要高于 1.12。
- 集群网络模式必须开启 VPC-CNI 弹性网卡模式。
- 直连模式 Service 使用的工作负载需使用 VPC-CNI 弹性网卡模式。
- 仅支持带宽上移账号，如若当前账户是传统账号类型（带宽非上移），可参见 账户类型升级说明 。
- 默认 CLB 的后端数量会有限制，具体限制数量与 CLB 所在地域有关，默认限制通常是100或200个，如果您绑定的工作负载的副本数超过，可通过 在线咨询 提升负载均衡 CLB 的配额。
- 满足 CLB 本身绑定弹性网卡的功能限制，详情请参见 绑定弹性网卡 。
- 开启直连 Pod 模式的工作负载更新时，将会根据 CLB 的健康检查状态进行滚动更新，会对更新速度造成一定影响。

> **注意：**
> 从 NodePort 接入方案迁移至直连时，需要注意以下事项：
> - 确认弹性网卡工作负载的安全组配置，确认出入流量放通。
> - 确认来源 IP 的变化对业务没有影响。（直连避免了 FullNAT 转发，来源 IP 不再是节点 IP，而是请求来源 IP）。
> - 确认工作负载当前没有处于滚动更新中的状态。
> - 如果出现数据面访问的不预期，请及时参考操作步骤进行回滚。

操作步骤

控制台操作指引

1. 登录 容器服务控制台 。
2. 参考 控制台创建 Service 步骤，进入**新建 Service** 页面，根据实际需求设置 Service 参数。
   其中，部分关键参数信息需进行如下设置，如下图所示：

- **服务访问方式**：选择为**公网 LB 访问**或**内网 LB 访问**。
- **网络模式**：勾选**采用负载均衡直连 Pod 模式**。（取消勾选则为默认的 NodePort 接入方式）
- **Workload 绑定**：选择**引用 Workload**。

3. 单击**创建 Service**，完成创建。

YAML 操作指引

直连 Pod 模式 Service 的 YAML 配置与普通 Service YAML 配置相同，示例中的 annotation 即代表是否开启直连 Pod 模式。

```
kind: Service
apiVersion: v1
metadata:
  annotations:
    service.cloud.tencent.com/direct-access: "true" ##开启直连 Pod 模式（删除注解或"false"，为关闭直连）
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
  type: LoadBalancer
```

annotation 扩展

负载均衡 CLB 的相关配置可参见 TkeServiceConfig 介绍。其中相关 annotation 配置如下：

```
service.cloud.tencent.com/tke-service-config: [tke-service-configName]
```

注意事项

如何保证滚动更新时的可用性保证

Kubernetes 官方提供的一个特性 ReadinessGate，主要是用来控制 Pod 的状态，集群版本需高于1.12。默认情况下，Pod 有以下 Condition：PodScheduled、Initialized、ContainersReady，当这几个状态都 Ready 的时候，Pod Ready 的 Condition 就通过了。但是在云原生场景下，Pod

的状态可能需要参考其他状态。ReadinessGate 提供了这样一个机制，允许为 Pod 的状态判断添加一个栅栏，由第三方来进行判断与控制。这样 Pod 的状态就和第三方关联起来了。

直连模式滚动更新的变化

当用户开始为应用做滚动更新的时候，Kubernetes 会根据更新策略进行滚动更新。但其判断一批 Pod 启动的标识仅包括 Pod 自身的状态，并不会考虑该 Pod 在负载均衡上是否配置健康检查且通过。如在接入层组件高负载时，不能及时对此类 Pod 进行及时调度，则滚动更新成功的 Pod 可能并没有正在对外提供服务，从而导致服务的中断。
为了关联滚动更新和负载均衡的后端状态，TKE 接入层组件引入了 Kubernetes 1.12中引入的新特性 `ReadinessGate`。TKE 接入层组件仅在确认后端绑定成功并且健康检查通过时，通过配置 `ReadinessGate` 的状态来使 Pod 达到 Ready 的状态，从而推动整个工作负载的滚动更新。

> **注意：**
> `ReadinessGate` 仅负责流量就绪检查，和 `Endpoints` 的职责是不一样的，所以 Pod 启动就绪之后的健康状态不会在 ReadinessGate Status 中体现。

在集群中使用 ReadinessGate

Kubernetes 集群提供了服务注册的机制，只需要将您的服务以 `MutatingWebhookConfigurations` 资源的形式注册至集群即可。集群会在 Pod 创建的时候按照配置的回调路径进行通知，此时可对 Pod 进行创建前的操作，即给 Pod 加上 `ReadinessGate`。需注意此回调过程必须是 HTTPS，即需要在 `MutatingWebhookConfigurations` 中配置签发请求的 CA，并在服务端配置该 CA 签发的证书。

ReadinessGate 机制的灾难恢复

用户集群中的服务注册或证书有可能被用户删除，虽然这些系统组件资源不应该被用户修改或破坏。在用户对集群的探索或是误操作下，这类问题会不可避免的出现。因此接入层组件在启动时会检查以上资源的完整性，在完整性受到破坏时会重建以上资源，加强系统的鲁棒性。详情可参见 Kubernetes Pods ReadinessGate 特性。

容器网络模式为 GlobalRouter

使用限制

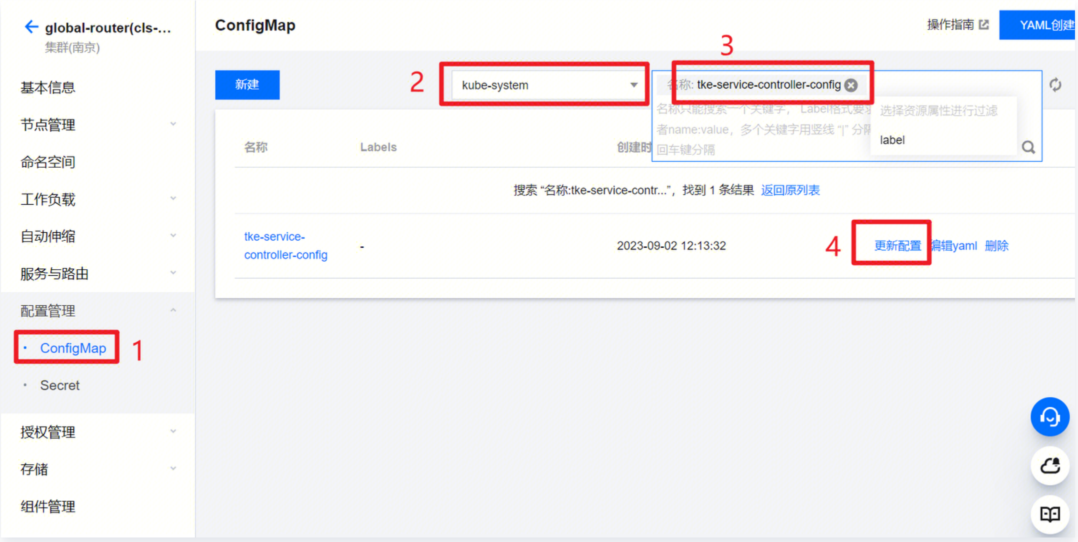
- 单个工作负载仅能运行在一种网络模式下，您可选择弹性网卡直连或 GlobalRoute 直连。
- 仅支持带宽上移账号，如若当前账户是传统账号类型（带宽非上移），可参见 账户类型升级说明。
- 默认 CLB 的后端数量会有限制，具体限制数量与 CLB 所在地域有关，默认限制通常是100或200个，如果您绑定的工作负载的副本数超过数量限制，可通过 在线咨询 提升负载均衡 CLB 的配额。
- 使用 CLB 直连 Pod，需注意网络链路受云服务器的安全组限制，确认安全组配置是否放开对应的协议和端口，**需要开启 CVM 上工作负载对应的端口**。
- 开启直连后，默认将启用 ReadinessGate 就绪检查，将会在 Pod 滚动更新时检查来自负载均衡的流量是否正常，需要为业务方配置正确的健康检查配置，详情可参见 TkeServiceConfig 介绍。
- 不支持黑石机型 BMG5n.24XLARGE384，如果有使用该机型，确保启用直连的 Pod 不被调度到该机型的节点。

步骤1: 开启直连配置

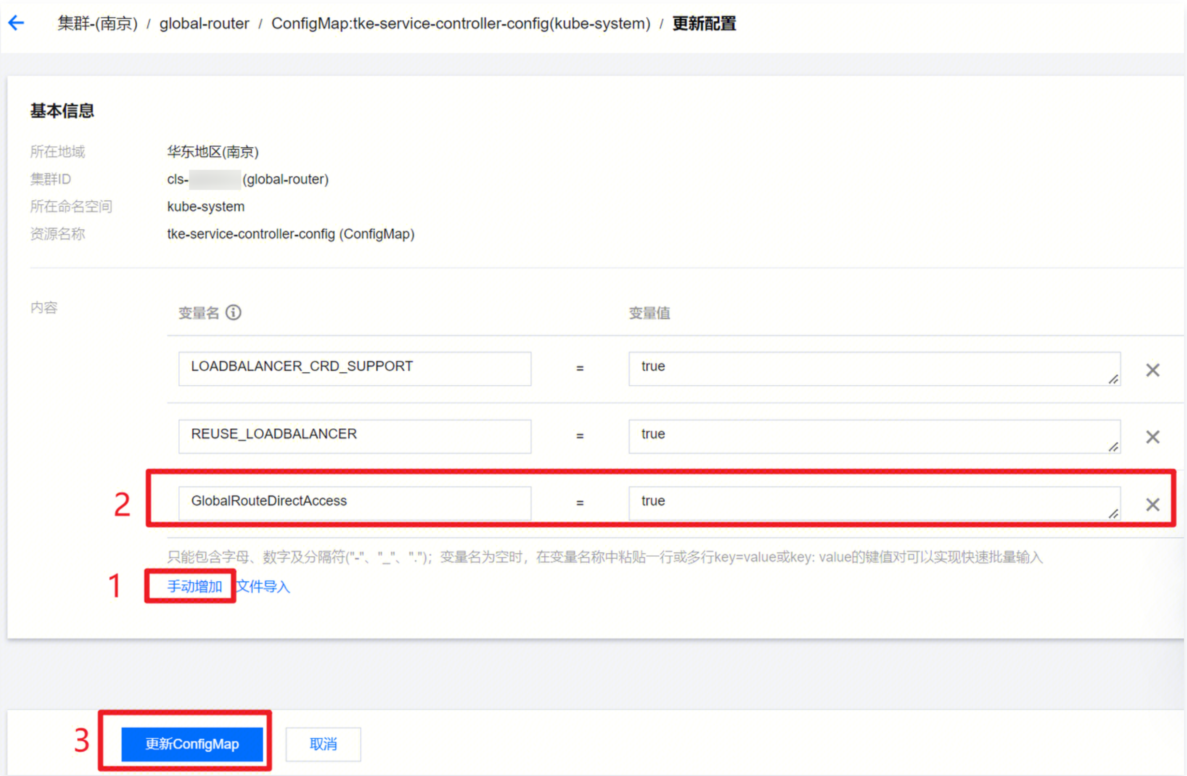
对于 GlobalRouter 集群，需要先开启“集群”维度的直连能力，再开启 “service” 维度的直连开关。
在 `kube-system/tke-service-controller-config` ConfigMap 中新增 `GlobalRouteDirectAccess: "true"` 以开启 GlobalRoute 直连能力。

控制台操作指引

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在**配置管理 > ConfigMap** 页面，命名空间选择 `kube-system`，搜索 `tke-service-controller-config`，单击**更新配置**。如下图所示：



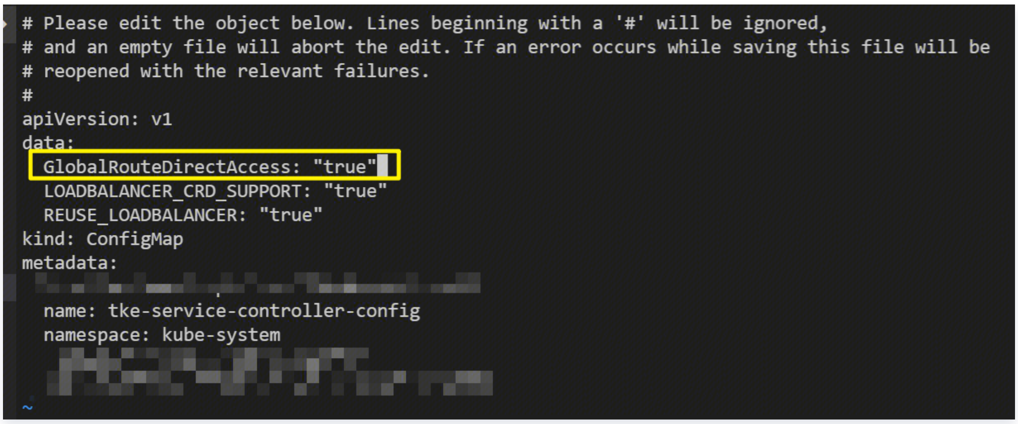
4. 在**更新配置**页面，选择**手动增加**，变量名填写为 `GlobalRouteDirectAccess` ，变量值填写为 `true` （注意没有引号）。如下图所示：



5. 单击**更新 ConfigMap**。

YAML 操作指引

1. 登录 容器服务控制台，在左侧导航栏中选择集群。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在配置管理 > ConfigMap 页面，单击 `tke-service-controller-config` 右侧的**编辑 yaml**。或通过命令行执行 `kubectl edit -n kube-system cm tke-service-controller-config` 进入 yaml 编辑模式。
4. 在 `data` 下新增行 `GlobalRouteDirectAccess: "true"` （注意缩进2个控制），保存即可。如下图所示：



```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  GlobalRouteDirectAccess: "true"
  LOADBALANCER_CRD_SUPPORT: "true"
  REUSE_LOADBALANCER: "true"
kind: ConfigMap
metadata:
  [redacted]
  name: tke-service-controller-config
  namespace: kube-system
  [redacted]
~
```

步骤2: Service 或 Ingress 中开启直连模式

2.1 Service 开启直连模式

您可以通过以下方式开启 Service 的直连模式：

方式1：

1. 登录 容器服务控制台，在左侧导航栏中选择集群。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在**服务与路由 > Service** 页面，单击待配置的 Service 右侧的**编辑 yaml**。

方式2：

通过命令行工具使用 `kubectl edit service -n <命名空间> <service名>` 进入编辑模式。

示例：

在 metadata.annotations 中新增以下注解： `service.cloud.tencent.com/direct-access: "true"`，表示开启 Service 的直连模式。如果未添加该注解或注解值为 "false"，则表示关闭直连模式。

直连 Pod 模式 Service 的其他 YAML 配置与普通 Service YAML 配置相同。

```
kind: Service
apiVersion: v1
metadata:
  annotations:
    service.cloud.tencent.com/direct-access: "true" ##开启直连 Pod 模式
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
  type: LoadBalancer
```

Annotation 扩展：

有关负载均衡 CLB 的配置，请参见 TkeServiceConfig 介绍。其中相关 Annotation 配置如下：

```
service.cloud.tencent.com/tke-service-config: [tke-service-configName]
```

2.2 Ingress 开启直连模式

对于 Ingress 需要使用 `ingress.cloud.tencent.com/direct-access: "true"` 注解来开启直连模式。

除了注解名称不同外，其余步骤与 开启 Service 的直连模式 相同。

多 Service 复用 CLB

最近更新时间：2025-01-07 10:01:34

操作场景

您可通过多个 Service 复用相同负载均衡器 CLB 的能力，来支持在同一个 VIP 同时暴露 TCP 及 UDP 的相同端口。

> **注意：**
> 其他场景下均不建议使用多个 Service 复用相同的 CLB。

说明事项

- **于2025年1月2日起创建的 TKE 集群（Master 托管），默认开启 Service 复用。**
- **于2020年8月17日至2025年1月1日创建的 TKE 集群，默认关闭多 Service 复用相同 CLB 的功能。**
  您可通过 在线咨询 联系我们开启需要使用多个 Service 复用相同 CLB 的功能。
  如果您的集群是 TKE Serverless 集群，集群默认已开启了 CLB 复用能力，但需要注意以下内容：
  1.1 用于复用的 CLB 必须为用户手动购买，而非 Serverless 集群自动购买。Serverless 集群自动购买的 CLB 在复用时会报错，是为了保护复用 CLB 的 Service 的 CLB 不被 Serverless 集群回收。
  1.2 CLB 购买成功后，需要在 Service 里添加两个 Annotation：
    - service.kubernetes.io/qcloud-share-existed-lb:"true"
    - service.kubernetes.io/tke-existed-lbid:lb-xxx
- Service 和 CLB 之间配置的管理和同步是由以 CLB ID 为名字的 LoadBalancerResource 类型的资源对象，请勿对该 CRD 进行任何操作，否则容易导致 Service 失效。
- **在业务发版时（滚动更新），请避免同时对这些 Service 关联的后端服务进行发版，因为同步这些复用相同 CLB 的 Service 的操作是串行的（保证同步结果的准确性），如果多个 Service 后端同时在更新，部分后端的流量摘除可能就不及时，容易造成流量异常，同时 优雅停机 的能力也不能真正生效。**

使用限制

- 多个 Service 复用相同 CLB 时，Service 的端口不能重复。
- 在 Service 复用场景下，单个负载均衡管理的监听器数量由 CLB 的 TOTAL_LISTENER_QUOTA 限制，更多请 查看文档。
- 在 Service 复用场景下，只能使用用户自行创建的负载均衡。因为容器服务 TKE 集群创建的负载均衡在被复用的情况下，负载均衡资源可能因为无法释放而导致泄漏。

  > **注意：**
  > 负载均衡资源被复用后，该 CLB 的生命周期将不由 TKE 侧控制，需要自行管理，请谨慎操作。

操作步骤

1. 参考 创建负载均衡实例，创建集群所在 VPC 下的公网或内网类型的负载均衡。
2. 参考 创建 Deployment 或 创建 Service，创建 LoadBalancer 类型的 Service，选择**使用已有**负载均衡，并选择 步骤1 中创建的负载均衡实例。如下图所示：

3. 重复步骤2，即可完成通过多个 Service 复用相同负载均衡器 CLB。

Service 扩展协议

最近更新时间：2024-08-28 09:50:11

Service 默认支持的协议

Service 是 Kubernetes 暴露应用程序到集群外的一种机制与抽象，您可以通过 Service 访问集群内的应用程序。

> **注意：**
> - 在 直连场景 下接入时，使用扩展协议没有任何限制，支持 TCP 和 UDP 协议混用。
> - 在非直连场景下，ClusterIP 和 NodePort 模式支持混用。但是，对于 LoadBalancer 类型的 Service，社区目前仅支持同类型协议的使用。
> - 当 LoadBalancer 声明为 TCP 时，端口可以使用扩展协议的能力，将负载均衡的协议变更为 TCP_SSL、HTTP 或 HTTPS。
> - 当 LoadBalancer 声明为 UDP 时，端口可以使用扩展协议的能力，将负载均衡的协议变更为 QUIC。

TKE 扩展 Service 转发协议

在原生的 Service 支持的协议规则上，存在部分场景需要在 Service 上同时支持 TCP 和 UDP 混合，且需 Service 能够支持 TCP SSL、HTTP、HTTPS 协议。TKE 针对 LoadBalancer 模式扩展了更多协议的支持。

前置说明

- 扩展协议仅对 LoadBalancer 模式的 Service 生效。
- 扩展协议通过注解 Annotation 的形式描述协议与端口的关系。
- 扩展协议与注解 Annotation 关系如下：
  - 当扩展协议注解中没有覆盖 Service Spec 中描述的端口时，Service Spec 按照用户描述配置。
  - 当扩展协议注解中描述的端口在 Service Spec 中不存在时，忽略该配置。
  - 当扩展协议注解中描述的端口在 Service Spec 中存在时，覆盖用户在 Service Spec 中声明的协议配置。

注解名称

service.cloud.tencent.com/specify-protocol

扩展协议注解示例

TCP_SSL 示例

```
{"80":{"protocol":["TCP_SSL"],"tls":"cert-secret"}}
```

HTTP 示例

```
{"80":{"protocol":["HTTP"],"hosts":{"a.tencent.com":{},"b.tencent.com":{}}}}
```

HTTPS 示例

```
{"80":{"protocol":["HTTPS"],"hosts":{"a.tencent.com":{"tls":"cert-secret-a"},"b.tencent.com":
{"tls":"cert-secret-b"}}}}
```


TCP/UDP混合示例

```
{"80":{"protocol":["TCP","UDP"]}} # 仅直连模式支持，详情见
https://cloud.tencent.com/document/product/457/41897
```

混合示例

```
{"80":{"protocol":["TCP_SSL","UDP"],"tls":"cert-secret"}} # 仅直连模式支持，详情见
https://cloud.tencent.com/document/product/457/41897
```

QUIC

```
{"80":{"protocol":["QUIC"],"tls":"cert-secret"}}
```

注意：
TCP_SSL 和 HTTPS 中的字段 `cert-secret`，表示使用该协议需要指定一个证书，证书是 Opaque 类型的 Secret，Secret 的 Key 为 qcloud_cert_id，Value 是证书 ID。详情见 Ingress 证书配置。

扩展协议使用说明

扩展协议YAML使用说明

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/specify-protocol: '{"80":{"protocol":["TCP_SSL"],"tls":"cert-secret"}}'
# 若要使用别的协议，修改该键值对的值为上述内容
  name: test
   ....
```

扩展协议控制台使用说明

- 在创建 Service 时，若以**公网 LB** 或**内网 LB** 的形式暴露服务，非 直连模式 情况下，端口映射中，仅支持 TCP 和 TCP SSL 一起使用。如下图所示：

- 当 Service 为**仅在集群内访问**（ClusterIP）或**主机端口访问**（NodePort）模式时，支持任意协议混用。
- 直连模式，支持任意协议混用。

案例说明

原生 Service 不支持协议混用，TKE 经过特殊改造后，在 直连场景 中支持混合协议的使用。
需注意的是，YAML 中仍使用相同的协议，但可以通过 Annotation 明确每个端口的协议类型。如下示例展示了 80 端口使用 TCP 协议，8080 端口使用 UDP 协议。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/direct-access: "true"  #TKE Serverless 集群默认是直连模式，TKE 集群请务必先参照文档开启直连模式。
    service.cloud.tencent.com/specify-protocol: '{"80":{"protocol":["TCP"]},"8080":{"protocol":["UDP"]}}' # 指定 80 端口 TCP 协议，8080 端口 UDP 协议。
  name: nginx
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: tcp-80-80
    nodePort: 32150
    port: 80
    protocol: TCP
    targetPort: 80
  - name: udp-8080-8080
    nodePort: 31082
    port: 8080
    protocol: TCP # 注意，因为 Kubernetes Service Controller 限制，只能使用同类型协议。
    targetPort: 8080
  selector:
    k8s-app: nginx
    qcloud-app: nginx
```

```
    sessionAffinity: None
    type: LoadBalancer
```

Service Annotation 说明

最近更新时间：2025-05-26 17:00:01

说明

查看当前集群 service-controller 版本：

```
kubectl -n kube-system get cm tke-service-controller-config -o jsonpath='{.data.VERSION}'
```

列表

负载均衡 ID

名称： service.kubernetes.io/loadbalance-id
说明：
只读注解，提供当前 Service 引用的负载均衡 LoadBalanceId。您可以在腾讯云 CLB 控制台查看与集群在同一 VPC 下的 CLB 实例 ID。
类型： string
版本要求： >=v1.0.0

复用已有负载均衡

名称： service.kubernetes.io/tke-existed-lbid
说明：
使用已存在的 CLB，需注意不同使用方式对腾讯云标签的影响。
详情参考 Service 使用已有 CLB 。
类型： string
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/tke-existed-lbid: 'lb-6swtxxxx'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

创建内网负载均衡

名称： service.kubernetes.io/qcloud-loadbalancer-internal-subnetid
说明：
在指定子网创建内网负载均衡。
类型： string
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: 'subnet-xxx'
  name: example
spec:
```

```
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

指定付费类型

名称： service.kubernetes.io/qcloud-loadbalancer-internet-charge-type
说明：
负载均衡的付费类型，当前仅在创建时支持配置，创建后不支持修改付费类型，创建后修改本注解无效。
指定创建负载均衡时，负载均衡的付费类型。请配合 service.kubernetes.io/qcloud-loadbalancer-internet-max-bandwidth-out 注解一起使用。
可选值：

- BANDWIDTH_POSTPAID_BY_HOUR：按带宽按小时后计费
- TRAFFIC_POSTPAID_BY_HOUR：按流量按小时后计费

类型： enum
版本要求： >=v1.0.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/qcloud-loadbalancer-internet-charge-type: 'BANDWIDTH_POSTPAID_BY_HOUR'
    service.kubernetes.io/qcloud-loadbalancer-internet-max-bandwidth-out: '10'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

指定公网带宽上限

名称： service.kubernetes.io/qcloud-loadbalancer-internet-max-bandwidth-out
说明：
CLB 带宽设置，当前仅在创建时支持配置，创建后不支持修改带宽，创建后修改本注解无效。
指定创建负载均衡时，负载均衡的最大出带宽，仅对公网属性的 LB 生效。需配合 service.kubernetes.io/qcloud-loadbalancer-internet-charge-type 注解一起使用。
值域：[1, 2048] Mbps。
类型： int
版本要求： >=v1.0.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/qcloud-loadbalancer-internet-max-bandwidth-out: '10'
```

```
    service.kubernetes.io/qcloud-loadbalancer-internet-charge-type: 'TRAFFIC_POSTPAID_BY_HOUR'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

负载均衡拓展参数

名称： service.kubernetes.io/service.extensiveParameters
说明：
指定 CLB 创建时的参数，当前仅在创建时支持配置，创建后不支持修改，创建后修改本注解无效。
详情参考 创建负载均衡实例 为创建负载均衡追加自定义参数。
类型： json
版本要求： >=v1.0.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/service.extensiveParameters: '{"AddressIPVersion":"IPV4","InternetAccessible":
{"InternetChargeType":"TRAFFIC_POSTPAID_BY_HOUR","InternetMaxBandwidthOut":10}}'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

安全组默认放通

名称： service.cloud.tencent.com/pass-to-target
说明：
是否启用 CLB 安全组默认放通的能力，CLB 和 CVM 之间默认放通，来自 CLB 的流量只需通过 CLB 上安全组的校验。
注意：
- 请查看 CLB 使用安全组的 使用限制 。
- 通常需要配合绑定安全组的能力。对应 Annotation 为：service.cloud.tencent.com/security-groups。
- 对于 Service 使用已有 CLB 的场景，若多个 Service 声明了不同的放通配置，会有逻辑冲突的问题。

类型： bool
版本要求： >=v1.8.3
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
```

```yaml
    service.cloud.tencent.com/pass-to-target: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

绑定/解绑安全组

名称： service.cloud.tencent.com/security-groups
说明：
为 CLB 绑定安全组，单个 CLB 最多可绑定5个安全组。
注意：
- 请查看 CLB 使用安全组的 使用限制 。
- 通常需要配合安全组默认放通的能力，CLB 和 CVM 之间默认放通，来自 CLB 的流量只需通过 CLB 上安全组的校验。对应 Annotation 为：service.cloud.tencent.com/pass-to-target。
- 对于 Service 使用已有 CLB 的场景，若多个 Service 声明了不同的安全组，会有逻辑冲突的问题。

类型： string
版本要求： >=v1.8.3
使用示例：
- **绑定安全组：**

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/security-groups: 'sg-xxxxx,sg-yyyyy,sg-zzzzz'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

- **解绑部分安全组：**

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/security-groups: 'sg-xxxxx'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
```

```yaml
  selector:
    app: example
  type: LoadBalancer
```

- **解绑全部安全组：**

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/security-groups: ''
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

修改保护

名称： service.cloud.tencent.com/modification-protection
说明：
配置 CLB 修改保护，开启后将无法通过 CLB 控制台或 API 修改负载均衡实例的属性。
类型： bool
版本要求： >=v1.7.3
使用示例：

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/modification-protection: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

监听器拓展参数

名称： service.kubernetes.io/service.listenerParameters
说明：
指定 CLB 创建监听器时的参数，当前仅在创建时支持配置，创建后不支持修改，创建后修改本注解无效。
类型： json
版本要求： >=v1.1.0

使用拓展配置

名称： service.cloud.tencent.com/tke-service-config

说明：
通过 tke-service-config 配置负载均衡 CLB。
详情参考 Service 负载均衡配置 。
类型： string
版本要求： >=v1.3.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/tke-service-config: 'config-name'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

使用自动生成拓展配置

名称： service.cloud.tencent.com/tke-service-config-auto
说明：
通过该注解可自动创建 TkeServiceConfig，然后可按需修改配置。
详情参考 Service 与 TkeServiceConfig 关联行为 。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/tke-service-config-auto: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

使用拓展协议

名称： service.cloud.tencent.com/specify-protocol
说明：
支持通过注解为指定的监听端口配置 TCP、UDP、TCP SSL、HTTP、HTTPS。
详情参考 Service 扩展协议 。
类型： json

版本要求： >=v1.5.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/specify-protocol: '{"80":{"protocol":["HTTPS"],"hosts":{"a.tencent.com":
{"tls":"cert-secret-a"},"b.tencent.com":{"tls":"cert-secret-b"}}}}'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

开启防止回环

名称： service.cloud.tencent.com/prevent-loopback
说明：
是否开启防止回环，开启后将修改健康探测源模式，采用100.64.0.0/10网段作为探测源。
详情参考 健康检查探测标识 。
类型： bool
版本要求： >=v1.7.3
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/prevent-loopback: 'false'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

开启直连

名称： service.cloud.tencent.com/direct-access
说明：
是否开启直连，负载均衡绑定 pod IP。
详情参考 使用 LoadBalancer 直连 Pod 模式 Service 。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

指定后端标签

名称： service.kubernetes.io/qcloud-loadbalancer-backends-label
说明：
指定标签设置负载均衡后端绑定的节点。
详情参考 指定接入层后端 。
类型： string
版本要求： >=v1.0.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.kubernetes.io/qcloud-loadbalancer-backends-label: 'group=access-layer'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

自定义后端权重

名称： service.cloud.tencent.com/lb-rs-weight
说明：
用来支持自定义后端权重，可以设定默认权重以及有状态服务的权重。
类型： json
版本要求： >=v1.6.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
```

```
    service.cloud.tencent.com/lb-rs-weight: '{"defaultWeight":10,"groups":[{"key":
{"proto":"TCP","port":80},"statefulSets":[{"name":"ti-ai-gateway-gz-shared-st-v1","weights":
[{"weight":0,"podIndexes":[0]}]},{"name":"ti-ai-gateway-gz-shared-st-v2","weights":
[{"weight":10,"podIndexes":[0]}]}]}]}'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

跳过 readiness-gate

名称： service.cloud.tencent.com/readiness-gate-skip
说明：
开启后，控制器健康检查时，将跳过 readiness-gate 的检查。
类型： bool
版本要求： >=v1.8.1
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/readiness-gate-skip: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

优雅停机

名称： service.cloud.tencent.com/enable-grace-shutdown
说明：
支持 CLB 直连模式的优雅停机。Pod 被删除，此时 Pod 里有 DeletionTimestamp，且状态置为 Terminating。此时调整 CLB 到该 Pod 的权重为 0。
仅在直连模式下支持，需要配合使用 service.cloud.tencent.com/direct-access。
v2.2.0 版本开始废弃，默认开启。
详情参考 Service 优雅停机 。
类型： bool
版本要求： >=v1.5.0
使用示例：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/enable-grace-shutdown: 'true'
```

```yaml
    service.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

优雅停机 tkex

名称： service.cloud.tencent.com/enable-grace-shutdown-tkex

说明：

支持 CLB 直连模式的优雅退出。Endpoint 对象中 endpoints 是否 not-ready，将 not-ready 的 CLB 后端权重置为 0。

仅在直连模式下支持，需要配合使用 service.cloud.tencent.com/direct-access。

v2.2.0 版本开始废弃，默认开启。

详情参考 Service 优雅停机 。

类型： bool

版本要求： >=v1.5.0

使用示例：

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.cloud.tencent.com/enable-grace-shutdown-tkex: 'true'
    service.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

优雅删除

名称： service.cloud.tencent.com/enable-grace-deletion

说明：

开启后，运行时等待控制器将所有 Pod 的权重调整为 0 后，再删除后端。

仅在直连模式下支持，需要配合使用 service.cloud.tencent.com/direct-access。

详情参考 容器服务 Pod 优雅删除 。

类型： bool

版本要求： >=v2.4.0

使用示例：

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
```

```yaml
    service.cloud.tencent.com/enable-grace-deletion: 'true'
    service.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: example
  type: LoadBalancer
```

Ingress 管理
Ingress Controllers 说明

最近更新时间：2024-07-30 17:06:21

各类型 Ingress Controllers 介绍

应用型 CLB

应用型 CLB 是基于腾讯云负载均衡器 CLB 实现的 TKE Ingress Controller，可以配置实现不同 URL 访问到集群内不同的 Service。CLB 直接将流量通过 NodePort 转发至 Pod（CLB 直连 Pod 时直接转发到 Pod），一条 Ingress 配置绑定一个 CLB 实例（IP），适合仅需做简单路由管理，对 IP 地址收敛不敏感的场景。详情可参见 CLB 类型 Ingress。

Nginx Ingress Controller（已废弃）

Nginx Ingress Controller 是基于腾讯云负载均衡器 CLB 和 Nginx 反向代理（容器化部署在集群内）的 Ingress Controller，通过 Annotations 扩展了原生 Kubernetes Ingress 的功能。CLB 后增加了一层代理（nginx），适合对接入层路由管理有更多诉求，及有 IP 地址收敛诉求的场景。详情可参见 Nginx 类型 Ingress。

各类型 Ingress Controllers 功能对比

模块	功能	应用型 CLB	Nginx Ingress Controller（已废弃）
流量管理	支持协议	http，https	http，https，http2，grpc，tcp，udp
	IP 管理	一条 Ingress 规则对应一个 IP（CLB）	多条 Ingress 规则对应一个 IP（CLB），IP 地址收敛
	特征路由	host，URL	更多特征支持：header、cookie 等
	流量行为	不支持	支持，重定向，重写等
	地域感知负载均衡	不支持	不支持
应用访问寻址	服务发现	单 Kubernetes 集群	单 Kubernetes 集群
安全	SSL 配置	支持	支持
	认证授权	不支持	支持
可观测性	监控指标	支持（需要在 CLB 中查看）	支持（云原生监控）
	调用追踪	不支持	不支持
	组件运维	关联 CLB 已托管，仅需集群内运行 TKE Ingress Controller	需集群内运行 Nginx Ingress Controller（控制面 + 数据面）

CLB 类型 Ingress
概述

最近更新时间：2025-05-07 17:12:42

Service 提供了基于四层网络的集群内容器服务的暴露能力，Service 暴露类型（例如 ClusterIP、NodePort 或 LoadBalancer）均基于四层网络服务的访问入口，缺少基于七层网络的负载均衡、SSL 或基于名称的虚拟主机等七层网络能力。Ingress 提供七层网络下 HTTP、 HTTPS 协议服务的暴露，及七层网络下的常见能力。

Ingress 基本概念

Ingress 是允许访问到集群内 Service 规则的集合，您可以通过配置转发规则，实现不同 URL 可以访问到集群内不同的 Service。为了使 Ingress 资源正常工作，集群需运行 Ingress Controller，容器服务在集群内默认启用了基于腾讯云负载均衡器实现的 TKE Ingress Controller。

Ingress 生命周期管理

Ingress 对外服务的能力依赖于负载均衡所提供的资源，因此服务资源管理也是 Ingress 的重要工作之一。Ingress 在资源的生命周期管理上会使用以下标签：

标签	描述
tke-createdBy-flag = yes	• 标识该资源是容器服务创建，拥有该标签的 Ingress 会在销毁时删除对应资源。 • 如果没有该标签，Ingress 会在销毁时，仅删除负载均衡内的监听器资源，而不删除负载均衡自身。
tke-clusterId = <clusterId>	• 标识该资源被哪一个 Cluster 所使用。 • Ingress 会在销毁时，删除对应标签（ClusterId 需正确）。
tke-lb-ingress-uuid = <Ingress UUID>	• 标识该资源被哪一个 Ingress 所使用。 • Ingress 目前不支持复用，当用户指定 Ingress 使用已有负载均衡时，标签的值若不正确会被拒绝。 • Ingress 会在销毁时，删除对应标签（Ingress UUID 需正确）。
tke-lifecycle-owner = <tke|user>	标识该 Ingress 对应 CLB 的生命周期管理权在 TKE 还是在用户。如果值为 tke，删除 Ingress 时会自动删除 CLB；如果值为 user，删除 Ingress 时不会删除 CLB。

> **说明：**
> 除 `tke-lifecycle-owner` 标签外，其他标签均对用户只读，请勿修改或删除上述标签，否则可能导致资源泄漏。
> 若用户使用了已有负载均衡，则 Ingress 仅会使用该负载均衡，而不会删除该负载均衡。
> 若负载均衡是 Ingress 自动创建的，删除 Ingress 默认会自动删除负载均衡，若希望保留负载均衡，可将 tke-lifecycle-owner 设置为 user（需要 kube-system 中 l7-lb-controller 容器镜像版本为 v2.4.0 及以上）。

Ingress Controller 使用方法

除了腾讯云服务提供的 TKE Ingress Controller 以外，Kubernetes 社区还有各种类型的第三方 Ingress Controller，这些 Ingress 控制器均为完成服务的七层网络暴露。Kubernetes 社区基本支持使用 `kubernetes.io/ingress.class` 注解和 `spec.ingressClassName` 字段用于区分各种 Ingress 控制器，以确定当前 Ingress 资源应被哪一个控制器处理。TKE Ingress Controller 也支持使用该注解，具体规则及使用建议如下：

- 当 Ingress 资源没有描述注解 `kubernetes.io/ingress.class` 和 `spec.ingressClassName` 字段时，TKE Ingress Controller 会管理当前 Ingress 资源。
- 当 Ingress 资源有注解 `kubernetes.io/ingress.class` 或 `spec.ingressClassName` 字段且值为 `qcloud` 时，TKE Ingress Controller 会管理当前 Ingress 资源。
- 当 Ingress 资源修改注解 `kubernetes.io/ingress.class` 或 `spec.ingressClassName` 字段的内容时，TKE Ingress Controller 会根据注解内容将其纳入或脱离管理范围，其操作会涉及到资源的创建与释放。
- 当您确认完全不需要使用 TKE Ingress Controller 时，可以将集群中的 Deployment（kube-system:l7-lb-controller）的工作副本数量调整为0，从而关闭 TKE Ingress Controller 功能。

  > **说明：**
  > - 关闭该功能前，请确保集群中没有被 TKE Ingress Controller 管理的 Ingress 资源，避免出现负载均衡资源释放失败的情况。
  > - 若用户在负载均衡上面开启了**删除保护**，或者使用**私有连接**，则删除 Service 时，不会删除该负载均衡。

Ingress 注意事项

请勿在负载均衡控制台手动删除 Ingress 所关联的 CLB 实例，也不能手动修改 CLB 的配置，CLB 相关操作都应在 TKE 侧进行。
如果您已在负载均衡控制台手动删除 Ingress 所关联的 CLB 实例，请参见 常见问题 进行恢复。

Ingress 相关操作

Ingress 相关操作及功能如下，您可参考以下文档进一步了解：
- Ingress 基本功能
- Ingress 使用已有 CLB
- Ingress 使用 TkeServiceConfig 配置 CLB
- Ingress 跨 VPC 绑定
- Ingress 重定向
- Ingress 混合使用 HTTP 及 HTTPS 协议
- Ingress 优雅停机
- Ingress 证书配置
- Ingress Annotation 说明

Ingress 基本功能

最近更新时间：2024-09-09 17:31:51

简介

Ingress 是允许访问到集群内 Service 的规则的集合，您可以通过配置转发规则，实现不同 URL 访问集群内不同的 Service。
为了使 Ingress 资源正常工作，集群必须运行 Ingress-controller。TKE 服务在集群内默认启用了基于腾讯云负载均衡器实现的 `l7-lb-controller`，支持 HTTP、HTTPS，同时也支持在集群内自建其他 Ingress 控制器，您可以根据您的业务需要选择不同的 Ingress 类型。

注意事项

- 腾讯云负载均衡（Cloud Load Balancer）实例已于2023年03月06日升级了架构，升级后公网负载均衡以域名的方式提供服务。VIP 随业务请求动态变化，控制台不再展示 VIP 地址。请参见 域名化公网负载均衡上线公告 。
  - 新注册的腾讯云用户默认使用升级后的域名化负载均衡。
  - 存量用户可以选择继续使用原有的负载均衡，不受升级影响。如果您需要升级负载均衡服务，则需要同时升级腾讯云产品 CLB 以及 TKE，否则 TKE 中的所有公网类型的 Service/Ingress 同步将可能受到影响。CLB 升级操作详情请参见 域名化负载均衡升级指南；TKE 升级 Service/Ingress 组件版本，请通过 在线咨询 联系我们。
- Ingress apiVersion 支持情况：
  - extensions/v1beta1 和 networking.k8s.io/v1beta1 API 版本的 Ingress 不在 v1.22 版本中继续提供。即仅在 TKE Kubernetes 版本 ≤ 1.20 版本中使用。
  - networking.k8s.io/v1 API 从 v1.19（TKE 场景只支持偶数版本，因此是从 TKE 的 v1.20）版本开始可用。即仅在 TKE Kubernetes 版本 ≥ 1.20 版本中使用。
  - 更多信息请参见 Kubernetes 文档 。
- 确保您的容器业务不和 CVM 业务共用一个 CLB。
- 不支持您在 CLB 控制台操作 TKE 管理的 CLB 的监听器、转发路径、证书和后端绑定的服务器，您的更改会被 TKE 自动覆盖。
- 使用已有的 CLB 时：
  - 只能使用通过 CLB 控制台创建的负载均衡器，不支持复用由 TKE 自动创建的 CLB。
  - 不支持多个 Ingress 复用 CLB。
  - 不支持 Ingress 和 Service 共用 CLB。
  - 删除 Ingress 后，复用 CLB 绑定的后端云服务器需要自行解绑，同时会保留一个 `tag tke-clusterId: cls-xxxx`，需自行清理。
  - 默认 CLB 的转发规则的限制是50个，如果您 Ingress 的转发规则超过50时，可通过 在线咨询 提升负载均衡 CLB 的配额。
  - Ingress 和 CLB 之间配置的管理和同步是由以 CLB ID 为名字的 LoadBalancerResource 类型的资源对象，请勿对该 CRD 进行任何操作，否则容易导致 Ingress 失效。

Ingress 控制台操作指引

创建 Ingress

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群管理页面，单击需要创建 Ingress 的集群 ID。
3. 在集群详情页，选择**服务与路由** > **Ingress**。
4. 在 Ingress 页面，单击**新建**。
5. 在**新建 Ingress** 页面，根据实际需求，设置 Ingress 参数。关键参数信息如下：
   - Ingress 名称：自定义。
   - 命名空间：根据实际需求进行选择。
   - Ingress 类型：根据实际需求进行选择，详情请参见 各类型 Ingress Controllers 功能对比 。
   - 网络类型：默认为 “公网”，请根据实际需求进行选择。
   - 负载均衡器：可自动创建或使用已有 CLB。
   - IP 版本：提供 IPv4 和 IPv6 NAT64 两种版本，请根据实际需求进行选择。
   - 运营商类型：支持 BGP（多线）运营商类型。
   - 网络计费模式：网络计费模式分为按带宽计费、按使用流量、共享带宽包。详情请参见 负载均衡计费模式 。
   - 实例规格：支持共享型实例和性能容量型实例。

- 转发配置：协议默认为 Http，请根据实际情况进行选择。如果协议选择 Https 则需绑定服务器证书，以保证访问安全。如下图所示：

证书详情请参见 SSL 证书格式要求及格式转换说明 。

6. 单击**创建Ingress**，完成创建。

更新 Ingress

更新 YAML

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群管理页面，单击集群 ID，进入待更新 YAML 的集群管理页面。
3. 在集群详情页，选择**服务与路由** > **Ingress**。
4. 在需要更新 YAML 的 Ingress 行中，单击**编辑 yaml**。
5. 对 YAML 进行编辑，单击**完成**。

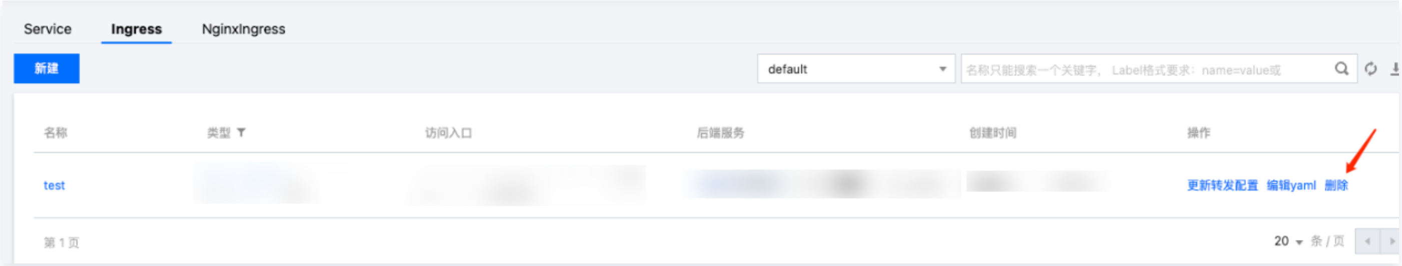
更新转发规则

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群管理页面，单击集群 ID，进入待更新 YAML 的集群管理页面。
3. 在集群详情页，选择**服务与路由** > **Ingress**。
4. 在需要更新 YAML 的 Ingress 行中，单击**更新转发配置**。
5. 在**更新转发配置**页面，根据实际需求，修改转发配置。如下图所示：

6. 单击**更新转发配置**，即可完成更新。

删除 Ingress

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群管理页面，单击需要删除 Ingress 的集群 ID。
3. 在集群详情页，选择**服务与路由** > **Ingress**。
4. 找到需要删除的 Ingress，单击**删除**。

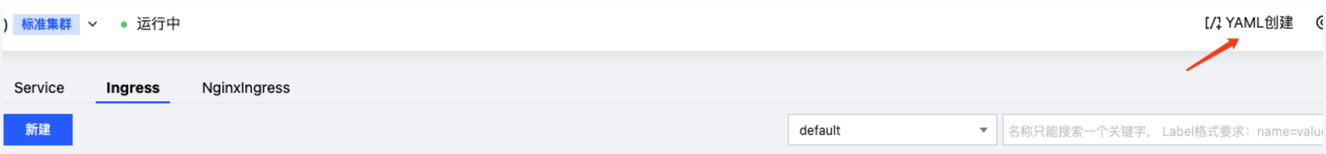


重建 Ingress

如果 CLB 被误删、或者修改了某些需要更换 CLB 的注解（如修改子网）时，需要重建 Ingress 才能生效，下面是重建 Ingress 的操作方法。

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群管理页面，单击需要重建 Ingress 的集群 ID。
3. 在集群详情页，选择**服务与路由** > **Ingress**。
4. 找到需要重建的 Ingress，单击**编辑 yaml**。

5. 复制 yaml 内容。
6. 删除 Ingress。
7. 单击右上角的 **YAML创建**。



8. 粘贴 Ingress 的 yaml 内容，移除自动生成的一些字段，如status、managedFields、creationTimestamp、finalizers、generation、resourceVersion、uid 等。
9. 单击**完成**。

Kubectl 操作 Ingress 指引

YAML 示例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: qcloud ## 可选值：qcloud（CLB类型ingress），nginx（nginx-ingress）,traefik
    ## kubernetes.io/ingress.existLbId: lb-xxxxxxxx   ##指定使用已有负载均衡器创建公网/内网访问的Ingress
    ## kubernetes.io/ingress.subnetId: subnet-xxxxxxxx  ##若是创建CLB类型内网ingress需指定该条annotation
  name: my-ingress
  namespace: default
spec:
  rules:
    - host: localhost
      http:
        paths:
          - backend:
              service:
                name: non-service
```

```
                port:
                  number: 65535
            path: /
            pathType: ImplementationSpecific
```

- kind：标识 Ingress 资源类型。
- metadata：Ingress 的名称、Label 等基本信息。
- metadata.annotations：Ingress 的额外说明，可通过该参数设置腾讯云 TKE 的额外增强能力。
- spec.rules：Ingress 的转发规则，配置该规则可实现简单路由服务、基于域名的简单扇出路由、简单路由默认域名、配置安全的路由服务等。

annotations: 使用已有负载均衡器创建公网/内网访问的 Ingress

如果您已有的应用型 CLB 为空闲状态，需要提供给 TKE 创建的 Ingress 使用，或期望在集群内使用相同的 CLB ，您可以通过以下 annotations 进行设置：

> **说明：**
> 请了解 注意事项 后开始使用。

```
metadata:
  annotations:
    kubernetes.io/ingress.existLbId: lb-6swtxxxx
```

annotations: 创建 CLB 类型内网 Ingress

如果您需要使用内网负载均衡，可以通过以下 annotations 进行设置：

```
metadata:
  annotations:
    kubernetes.io/ingress.subnetId: subnet-xxxxxxxx
```

说明事项

如果您使用的是 **IP 带宽包**账号，在创建公网访问方式的服务时需要指定以下两个 annotations 项：

- `kubernetes.io/ingress.internetChargeType` 公网带宽计费方式，可选值有：
  - TRAFFIC_POSTPAID_BY_HOUR（按使用流量计费）
  - BANDWIDTH_POSTPAID_BY_HOUR（按带宽计费）
- `kubernetes.io/ingress.internetMaxBandwidthOut` 带宽上限，范围：[1,2000] Mbps。
  例如：

  ```
  metadata:
    annotations:
      kubernetes.io/ingress.internetChargeType: TRAFFIC_POSTPAID_BY_HOUR
      kubernetes.io/ingress.internetMaxBandwidthOut: "10"
  ```

创建 Ingress

1. 参考 YAML 示例，准备 Ingress YAML 文件。
2. 安装 Kubectl，并连接集群。操作详情请参考 通过 Kubectl 连接集群。
3. 执行以下命令，创建 Ingress YAML 文件。

   ```
   kubectl create -f Ingress YAML 文件名称
   ```

   例如，创建一个文件名为 my-ingress.yaml 的 Ingress YAML 文件，则执行以下命令：

   ```
   kubectl create -f my-ingress.yaml
   ```

4. 执行以下命令，验证创建是否成功。

```
kubectl get ingress
```

返回类似以下信息，即表示创建成功。

```
NAME         CLASS    HOSTS       ADDRESS      PORTS   AGE
my-ingress   <none>   localhost                80      4s
```

更新 Ingress

方法一

执行以下命令，更新 Ingress。

```
kubectl edit  ingress/[name]
```

方法二

1. 手动删除旧的 Ingress。
2. 执行以下命令，重新创建 Ingress。

```
kubectl create/apply
```

删除 Ingress

执行以下命令，删除 Ingress。

```
kubectl delete ingress/[name]
```

Ingress 使用已有 CLB

最近更新时间：2025-03-24 18:01:12

腾讯云容器服务 TKE 具备通过 `kubernetes.io/ingress.existLbId: <LoadBalanceId>` 注解使用已有负载均衡的功能，您可使用该注解指定 Ingress 关联的负载均衡实例。

注意事项

- 请确保您的容器业务不与云服务器 CVM 业务共用一个负载均衡资源。
- 不支持在负载均衡控制台操作 `Ingress Controller` 管理的负载均衡监听器以及后端绑定的服务器，更改会被 `Ingress Controller` 自动覆盖。
- 使用已有负载均衡时：
  - 不支持多个 Ingress 复用同一个负载均衡。
  - 指定负载均衡的已有端口，不能和 ingress 上定义的端口有冲突。通常 ingress 会创建 443 和 80 端口，因此指定的负载均衡不能有这两个端口，如果有的话需要先删除。
  - 仅支持使用通过负载均衡控制台创建的负载均衡器，不支持使用由 `Service Controller` 自动创建和管理的负载均衡，即 Service 和 Ingress 不能混用同一个负载均衡。
  - `Ingress Controller` 不负责负载均衡的资源管理，即在 Ingress 资源删除时，负载均衡资源不会被删除回收。

使用场景

使用包年包月的负载均衡对外提供服务

`Ingress Controller` 管理负载均衡生命周期时，仅支持购买按量计费的资源。但由于包年包月的负载均衡在价格上有一定的优势，用户需要长时间使用负载均衡时，通常会优先选择购买包年包月负载均衡。

在此类场景下，用户就可以独立购买和管理负载均衡。通过注解控制 Ingress 使用已有负载均衡，将负载均衡的生命周期管理从 `Ingress Controller` 中剥离。示例如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.existLbId: lb-xxxxxxxx
  name: nginx-ingress
spec:
  rules:
    - http:
        paths:
          - backend:
              service:
                name: nginx-service
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
```

`kubernetes.io/ingress.existLbId: lb-xxxxxxxx` 注解表明了该 Ingress 将使用已有负载均衡 `lb-xxxxxxxx` 进行 Ingress 服务配置。

Ingress 使用 TkeServiceConfig 配置 CLB

最近更新时间：2025-03-31 17:00:33

TkeServiceConfig

TkeServiceConfig 是腾讯云容器服务 TKE 提供的自定义资源 CRD，通过 TkeServiceConfig 能够帮助您更灵活的进行 Ingress 管理负载均衡的各种配置。

使用场景

Ingress YAML 的语义无法定义的负载均衡参数和功能，可以通过 TkeServiceConfig 来配置。

配置说明

使用 TkeServiceConfig 能够帮您快速进行负载均衡器的配置。通过 Ingress 注解 **ingress.cloud.tencent.com/tke-service-config:<config-name>**，您可以指定目标配置应用到 Ingress 中。

> **注意：**
> TkeServiceConfig 资源需要和 Ingress 处于同一命名空间。

TkeServiceConfig 不会帮您配置并修改协议、端口、域名以及转发路径，您需要在配置中描述协议、端口、域名还有转发路径以便指定配置下发的转发规则。每个七层的监听器下可有多个域名，每个域名下可有多个转发路径。因此，在一个 `TkeServiceConfig` 中可以声明多组域名、转发规则配置，目前主要针对负载均衡的健康检查以及对后端访问提供配置。

- 通过指定协议和端口，配置能够被准确地下发到对应监听器：
  - `spec.loadBalancer.l7Listeners.protocol`：七层协议
  - `spec.loadBalancer.l7Listeners.port`：监听端口
- 通过指定协议、端口、域名以及访问路径，可以配置转发规则级别的配置。例如，后端健康检查、负载均衡方式。
  - `spec.loadBalancer.l7Listeners.protocol`：七层协议
  - `spec.loadBalancer.l7Listeners.port`：监听端口
  - `spec.loadBalancer.l7Listeners.domains[].domain`：域名
  - `spec.loadBalancer.l7Listeners.domains[].rules[].url`：转发路径
  - `spec.loadBalancer.l7listeners.protocol.domain.rules.url.forwardType`：指定后端协议，支持 HTTP/HTTPS/GRPC。
    - 后端协议是指 CLB 与后端服务之间的协议：后端协议选择 HTTP 时，后端服务需部署 HTTP 服务。后端协议选中 HTTPS 时，后端服务需部署 HTTPS 服务，HTTPS 服务的加解密会让后端服务消耗更多资源。更多请查看 CLB 配置 HTTPS 监听器。

> **说明：**
> 当您的域名配置为默认值，即公网或内网 VIP 时，可以通过 domain 填空值的方式进行配置。

Ingress 与 TkeServiceConfig 关联行为

1. 创建 Ingress 时，设置 **ingress.cloud.tencent.com/tke-service-config-auto: "true";**，将自动创建 <IngressName>-auto-ingress-config。您也可以通过 **ingress.cloud.tencent.com/tke-service-config:<config-name>** 直接指定您自行创建的 TkeServiceConfig。两个注解不可同时使用。
2. 您为 Service\Ingress 使用的自定义配置，名称不能以 `-auto-service-config` 与 `-auto-ingress-config` 为后缀。
3. 其中自动创建的 TkeServiceConfig 存在以下同步行为：
   - 更新 Ingress 资源时，新增若干7层转发规则，如果该转发规则没有对应的 TkeServiceConfig 配置片段。Ingress-Controller 将主动添加 TkeServiceConfig 对应片段。
   - 删除若干7层转发规则时，Ingress-Controller 组件将主动删除 TkeServiceConfig 对应片段。
   - 删除 Ingress 资源时，联级删除该 TkeServiceConfig。
   - 用户修改 Ingress 默认的 TkeServiceConfig，TkeServiceConfig 内容同样会被应用到负载均衡。
4. 您也可以参考下列 TkeServiceConfig 完整配置参考，自行创建需要的 CLB 配置，Service 通过注解 **ingress.cloud.tencent.com/tke-service-config:<config-name>** 引用该配置。
5. 其中您手动创建的 TkeServiceConfig 存在以下同步行为：
   - 当用户在 Ingress 中使用配置注解时，负载均衡将会即刻进行设置同步。

- 当用户在 Ingress 中删除配置注解时，负载均衡将会保持不变。
- 修改 TkeServiceConfig 配置时，引用该配置的 Ingress 的负载均衡将会根据新的 TkeServiceConfig 进行设置同步。
- 当 Ingress 的监听器没有找到对应配置时，该监听器将不会进行修改。
- Ingress 的监听器找到对应配置时，若配置中没有声明的属性，该监听器将不会进行修改。

示例

Deployment 示例：jetty-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: jetty
  name: jetty-deployment
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: jetty
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: jetty
    spec:
      containers:
      - image: jetty:9.4.27-jre11
        imagePullPolicy: IfNotPresent
        name: jetty
        ports:
        - containerPort: 80
          protocol: TCP
        - containerPort: 443
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

Service 示例：jetty-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: jetty-service
  namespace: default
```

```
spec:
  ports:
  - name: tcp-80-80
    port: 80
    protocol: TCP
    targetPort: 80
  - name: tcp-443-443
    port: 443
    protocol: TCP
    targetPort: 443
  selector:
    app: jetty
  type: NodePort
```

该示例包含以下配置：

Service 的 NodePort 类型，声明了两个 TCP 服务。一个在80端口，一个在443端口。

Ingress：jetty-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.rule-mix: "true"
    kubernetes.io/ingress.http-rules: '[{"path":"/health","backend":{"serviceName":"jetty-
service","servicePort":"80"}}]'
    kubernetes.io/ingress.https-rules: '[{"path":"/","backend":{"serviceName":"jetty-
service","servicePort":"443","host":"sample.tencent.com"}}]'
    ingress.cloud.tencent.com/tke-service-config: jetty-ingress-config
    # 指定已有的 tke-service-config
    # ingress.cloud.tencent.com/tke-service-config-auto: "true"
    # 自动创建 tke-service-config
  name: jetty-ingress
  namespace: default
spec:
  rules:
    - http:
        paths:
          - backend:
              service:
                name: jetty-service
                port:
                  number: 80
            path: /health
            pathType: ImplementationSpecific
    - host: "sample.tencent.com"
      http:
        paths:
          - backend:
              service:
                name: jetty-service
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
  tls:
    - secretName: jetty-cert-secret
```

该示例包含以下配置：

- 使用了混合协议，使用默认域名（公网 IP）暴露了一个 HTTP 服务，使用 `sample.tencent.com` 域名暴露了一个 HTTPS 服务。

- HTTP 服务的转发路径是 `/health`，HTTPS 服务的转发路径是 `/`。
- 使用了 `jetty-ingress-config` 负载均衡配置。

TkeServiceConfig 示例：jetty-ingress-config.yaml

```
apiVersion: cloud.tencent.com/v1alpha1
kind: TkeServiceConfig
metadata:
  name: jetty-ingress-config
  namespace: default
spec:
  loadBalancer:
    l7Listeners:
    - protocol: HTTP
      port: 80
      domains:
      - domain: ""      # domain为空表示使用VIP作为域名
        rules:
        - url: "/health"
          forwardType: HTTP # 指定后端协议为 HTTP，目前支持 HTTP/HTTPS/GRPC。
          healthCheck:
            enable: false
    - protocol: HTTPS
      port: 443
      defaultServer: "sample.tencent.com" # 默认域名
      keepaliveEnable: 1                  # 监听器开启长连接（非keepalive白名单用户，请勿声明该字段）
      domains:
      - domain: "sample.tencent.com"
        http2: true # 启用 HTTP 2.0
        rules:
        - url: "/"
          forwardType: HTTPS # 指定后端协议为 HTTPS，目前支持 HTTP/HTTPS/GRPC。
          session:
            enable: true
            sessionExpireTime: 3600
          healthCheck:
            enable: true
            intervalTime: 10 # intervalTime 要大于 timeout，否则会出错
            timeout: 5 # timeout 要小于 intervalTime，否则会出错
            healthNum: 2
            unHealthNum: 2
            httpCheckPath: "/checkHealth"
            httpCheckDomain: "sample.tencent.com" #注意：健康检查必须使用固定域名进行探测，如果您在.spec.loadBalancer.l7Listeners.protocol.domains.domain 里填写的是泛域名，一定要使用 httpCheckDomain 字段明确具体需要健康检查的域名，否则泛域名不支持健康检查。
            httpCheckMethod: HEAD
            httpCode: 31 # 可选值：1~31，默认 31。 1 表示探测后返回值 1xx 代表健康，2 表示返回 2xx 代表健康，4 表示返回 3xx 代表健康，8 表示返回 4xx 代表健康，16 表示返回 5xx 代表健康。若希望多种返回码都可代表健康，则将相应的值相加。
            sourceIpType: 0 # 可选值：0或1，设定健康检查源ip。0 表示负载均衡VIP，1 表示 100.64.0.0/10 网段ip。对于域名化clb默认值为1且只能为1，对于非域名化的clb默认值不一定，可在clb控制台配置页面看能否看到VIP探测方式，如能看到默认值为0，否则为1，更多详情查看 https://cloud.tencent.com/document/product/214/86666。
            checkType: "HTTPS" # 可选值：HTTP，HTTPS 或 TCP，默认 HTTP。2024.06之后新建的集群支持改字段，存量集群如有需要可提工单升级后台组件来支持。当 forwardType 为 HTTPS 时，checkType 只能为 TCP 或 HTTPS。
          scheduler: WRR # 可选值：WRR、LEAST_CONN、IP_HASH
```

该示例包含以下配置：

该 TkeServiceConfig 名称为 `jetty-ingress-config`。且在七层监听器配置中，声明了两段配置：

1. 80端口的 HTTP 监听器将会被配置，其中包含域名配置，是默认域名对应负载均衡的 VIP。
   `/health` 路径下的健康检查被关闭了。

2. 443端口的 HTTPS 监听器将会被配置。其中包含域名配置，域名是 `sample.tencent.com` 。该域名下仅描述了一个转发路径为 `/` 的转发规则配置，其中配置包含以下内容：
   - 打开健康检查，健康检查间隔调整为10s，健康阈值2次，不健康阈值2次。通过 HEAD 请求进行健康检查，检查路径为 `/checkHealth` ，检查域名为 `sample.tencent.com` 。
   - 打开会话保持功能，会话保持的超时时间设置为3600s。
   - 转发策略配置为：按权重轮询。

kubectl 配置命令

```
$ kubectl apply -f jetty-deployment.yaml
$ kubectl apply -f jetty-service.yaml
$ kubectl apply -f jetty-ingress.yaml
$ kubectl apply -f jetty-ingress-config.yaml

$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
jetty-deployment-8694c44b4c-cxscn   1/1     Running   0          8m8s
jetty-deployment-8694c44b4c-mk285   1/1     Running   0          8m8s
jetty-deployment-8694c44b4c-rjrtm   1/1     Running   0          8m8s

# 获取TkeServiceConfig配置列表
$ kubectl get tkeserviceconfigs.cloud.tencent.com
NAME                    AGE
jetty-ingress-config    52s

# 更新修改TkeServiceConfig配置
$ kubectl edit tkeserviceconfigs.cloud.tencent.com jetty-ingress-config
tkeserviceconfigs.cloud.tencent.com/jetty-ingress-config edited
```

Ingress 跨 VPC 绑定

最近更新时间：2025-04-24 16:34:12

简介

使用 CLB 型 Ingress 时，默认是在当前集群所在 VPC 内的随机可用区生成 CLB。目前 TKE 的 CLB Ingress 已支持指定可用区，包括其他地域的可用区。本文将为您介绍如何通过控制台和 YAML 两种方式为 CLB Ingress 跨 VPC 绑定和指定可用区。

应用场景

- 需要支持 CLB 的跨地域接入或跨 VPC 接入，即 CLB 所在的 VPC 和当前集群所在的 VPC 不在同一 VPC 内。
- 需要指定 CLB 的可用区以实现资源的统一管理。

说明：

1. 如需使用非本集群所在 VPC 的 CLB，需先通过 云联网 打通当前集群 VPC 和 CLB 所在的 VPC。
    1.1 云联网需要提前规划各个地域 VPC 的网段，不能出现冲突。否则冲突的路由规则将不会生效，导致数据面无法转发。
    1.2 集群所在 VPC 不能同时加入多个云联网中，否则会出现路由不唯一的情况，导致数据面无法转发。
    1.3 VPC 需要与 TKE 集群在同一个账号下（不支持跨账号的 VPC）。
2. 在确保 VPC 已经打通之后，请 在线咨询 申请使用该功能。
3. 以下 YAML 中，需要您输入地域 ID ，您可以通过 地域和可用区 查看地域 ID。

操作步骤

CLB Ingress 跨 VPC 绑定和指定可用区支持通过控制台和 YAML 两种方式进行操作，操作步骤如下：

控制台方式

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 在**服务与路由** > **Ingress** 中，单击**新建**。如下图所示：

4. 在**新建** Ingress 页面中配置相关可用区规则。配置规则说明如下：
    - **当前 VPC**：使用本集群所在 VPC 内的 CLB，建议使用随机可用区，若指定可用区的资源售罄将无法创建相关实例。
    - **其它 VPC**：仅支持通过 云联网 与当前集群的 VPC 打通的其他 VPC。建议使用随机可用区，若指定可用区的资源售罄将无法创建相关实例。

YAML 方式

> **说明：**
> 1. 如需使用非本集群所在 VPC 的 CLB，需先通过 云联网 打通当前集群 VPC 和 CLB 所在的 VPC。
> 2. 在确保 VPC 已经打通之后，请 在线咨询 申请使用该功能。

示例1

如果仅需要指定本集群所在 VPC 的可用区，例如集群的 VPC 在广州地域，CLB Ingress 需要指定广州一区的 CLB，可以在 Ingress 的 YAML 中添加如下 annotation：

```
kubernetes.io/ingress.extensiveParameters: '{"ZoneId":"ap-guangzhou-1"}'
```

示例2

如需使用非本集群所在 VPC 内的 CLB，需先添加如下两条 annotation：

```
ingress.cloud.tencent.com/cross-region-id:
ingress.cloud.tencent.com/cross-vpc-id:
```

具体示例如下：

- 创建异地接入的负载均衡，需先添加如下两条 annotation：

```
ingress.cloud.tencent.com/cross-region-id: "ap-guangzhou"
ingress.cloud.tencent.com/cross-vpc-id: "vpc-646vhcjj"
```

- 选择已有负载均衡进行异地接入，添加如下两条 annotation：

```
ingress.cloud.tencent.com/cross-region-id: "ap-guangzhou"
kubernetes.io/ingress.existLbId: "lb-342wppll"
```

> **注意：**
> 若您还需指定可用区，则需要再添加示例1中的 annotation。

完整 Ingress Annotation 说明请参见 Ingress Annotation 说明 。

Ingress 重定向

最近更新时间：2024-01-02 10:53:02

简介

域名重定向，指当用户通过浏览器访问某个 URL 时，Web 服务器被设置自动跳转到另外一个 URL。

应用场景

- 网站支持 HTTP 和 HTTPS，例如 `http://tencent.com` 和 `https://tencent.com` 需要访问到同一个 Web 服务。
- 网站更换过域名，例如 `https://tengxun.com` 更换为 `https://tencent.com`，两个域名访问到同一个 Web 服务。
- 网站部分内容做过调整，原始 URL 已经无法访问，可以重定向到一个新的提供服务的 URL。

> **注意**
> - 当用户使用重定向后，将会多出如下一条注解，该注解表明 Ingress 的转发规则由 TKE 管理，后期不能被删除和修改，否则将和 CLB 侧设置的重定向规则冲突。
> ```
> ingress.cloud.tencent.com/rewrite-support: "true"
> ```
> - 假设用字母表示域名地址，若 A 已经重定向至 B，则：
>   - A 不能再重定向至 C（除非先删除旧的重定向关系，再建立新的重定向关系）。
>   - B 不能重定向至任何其他地址。
>   - A 不能重定向到 A。

重定向方式

重定向有如下两种方式：
- **自动重定向**：用户需要先创建出一个 `HTTPS:443` 监听器，并在其下创建转发规则。通过调用本接口，系统会自动创建出一个 `HTTP:80` 监听器（如果之前不存在），并在其下创建转发规则，与 `HTTPS:443` 监听器下的域名等各种配置对应。
- **手动重定向**：用户手动配置原访问地址和重定向地址，系统自动将原访问地址的请求重定向至对应路径的目的地址。同一域名下可以配置多条路径作为重定向策略，实现 HTTP 和 HTTPS 之间请求的自动跳转。

注意事项

- 若您没有 TKE Ingress 重定向注解声明，会兼容原有不管理重定向规则的逻辑，即：您可以在负载均衡 CLB 的控制台里面配置重定向规则，TKE Ingress 不处理用户在 CLB 控制台配置的这些重定向规则。
- 若您没有 TKE Ingress 重定向注解声明，因为 CLB 的重定向保护限制，如果转发配置 A 重定向到转发配置 B，此时无法直接删除转发配置 B，必须先删掉该重定向规则，才能删除转发配置 B。
- 若您使用 TKE Ingress 重定向注解声明，CLB 下所有重定向规则都是由 TKE Ingress 管理，所有重定向规则仅在 TKE Ingress 里面的相关 Annotation 里面生效，此时用户在 CLB 控制台如果修改重定向配置，最终会被 TKE Ingress 里配置的重定向规则覆盖。
- 当前基于 CLB 的 TKE ingress 暂不支持 nginx-ingress 的 `rewrite-target` 功能，无法重写路径。

操作步骤

Ingress 支持通过控制台和 YAML 两种方式进行重定向，具体步骤如下：

通过控制台

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**服务与路由** > **Ingress**页面，单击**新建**。如下图所示：

4. 在**新建 Ingress** 页面中配置相关重定向规则。配置规则说明如下：
   - **无**：不使用重定向规则。
   - **手动**：会在**转发配置**下方出现一栏**重定向转发配置**。
     - **转发配置**里面填写的方式和普通 Ingress 的转发配置一样，后端是某个服务。
     - **重定向转发配置**里面填写的方式和普通 Ingress 的转发配置一样，但后端是某个“转发配置”里的某条路径。

   - **自动**：仅对**转发配置**里的协议为 “HTTPS” 的路径生效，都将自动生成一个 “HTTP” 的路径，路径完全一样，只有协议不一样。 “HTTP” 的路径的转发规则自动重定向到 “HTTPS” 的路径。

通过 YAML

自动重定向：HTTP 重定向到 HTTPS

> **注意**
> 仅对 HTTPS 协议的转发规则生效。

在 Ingress YAML 中配置如下注解：

```
ingress.cloud.tencent.com/rewrite-support: "true"  # 表示允许重定向
ingress.cloud.tencent.com/auto-rewrite: "true"     # 表示自动重定向
```

配置该注解之后，转发路径中的所有 HTTPS 监听器中存在的七层转发规则都将被对应到 HTTP 监听器中作为重定向规则。域名与路径都保持一致。

手动重定向

手动重定向需要增加一个 annotation，修改一个 annotation：
- 增加的 annotation：

```
ingress.cloud.tencent.com/rewrite-support: "true"  # 表示允许重定向
```

- 修改的 annotation：

```
# 原注解 格式：
{
  "host": "<domain>",
  "path": "<path>",
  "backend": {
    "serviceName": "<service name>",
    "servicePort": "<service port>"
  }
}

# 新注解格式：
{
  "host": "<domain>",
  "path": "<path>",
  "backend": {
    "serviceName": "<service name>",
    "servicePort": "<service port>"
  },
  "rewrite": {
    "port": "<rewrite-port>",
    "host": "<rewrite-domain>",
    "path": "<rewrite-path>"
  }
}
```

示例

某服务之前通过 `121.4.25.44/path2` 进行访问，在发布新版本之后，期望通过 `121.4.25.44/v2/path2` 进行访问。可以进行如下修改：
1. 增加一条 annotation：

```
ingress.cloud.tencent.com/rewrite-support: "true"   # 允许重定向
```

2. 修改原 annotation：

```
# 将 /v1/path1 替换为 /path1 80端口；把 /v2/path2 替换为 /path2 80端口。注意：host 可以省略
kubernetes.io/ingress.http-rules: '
[
  {
```

```
      "path": "/path1",
      "backend": {
        "serviceName": "path1",
        "servicePort": "80"
      }
    },
    {
      "path": "/path2",
      "backend": {
        "serviceName": "path2",
        "servicePort": "80"
      }
    },
    {
      "path": "/v1/path1",
      "rewrite": {
        "port": 80,
        "path": "/path1"
      }
    },
    {
      "path": "/v2/path2",
      "rewrite": {
        "port": 80,
        "path": "/path2"
      }
    }
  ]'
```

修改后的 YAML：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    description: test
    ingress.cloud.tencent.com/rewrite-support: "true"
    kubernetes.io/ingress.class: qcloud
    kubernetes.io/ingress.http-rules: '[{"path":"/path1","backend":
{"serviceName":"path1","servicePort":"80"}},{"path":"/path2","backend":
{"serviceName":"path2","servicePort":"80"}},{"path":"/v1/path1","rewrite":
{"port":80,"path":"/path1"}},{"path":"/v2/path2","rewrite":{"port":80,"path":"/path2"}}]'
    kubernetes.io/ingress.https-rules: "null"
    kubernetes.io/ingress.rule-mix: "true"
  name: test
  namespace: default
spec:
  rules:
    - http:
        paths:
          - backend:
              service:
                name: path1
                port:
                  number: 80
            path: /path1
            pathType: ImplementationSpecific
    - http:
        paths:
          - backend:
```

```
              service:
                name: path2
                port:
                  number: 80
            path: /path2
            pathType: ImplementationSpecific
status:
  loadBalancer:
    ingress:
      - ip: 121.4.25.44
```

完整 Ingress Annotation 说明请参见 Ingress Annotation 说明 文档。

Ingress 混合使用 HTTP 及 HTTPS 协议

最近更新时间：2024-09-02 17:34:41

混合规则

默认场景下，当 Ingress 中不配置 TLS 时，服务将以 HTTP 协议的方式对外暴露。当 Ingress 配置 TLS 时，服务将以 HTTPS 协议的方式对外暴露。Ingress 描述的服务只能以其中一种协议暴露服务，基于此规则的局限性，腾讯云容器服务 TKE 提供了混合协议的支持。
用户需要同时暴露 HTTP 及 HTTPS 服务时，只需参考本文，开启混合协议并配置所有的转发规则到 `kubernetes.io/ingress.http-rules` 及 `kubernetes.io/ingress.https-rules` 注解中即可。

规则格式

`kubernetes.io/ingress.http-rules` 及 `kubernetes.io/ingress.https-rules` 的规则格式是一个 `Json Array` 。每个对象的格式如下：

```
{
  "host": "<domain>",
  "path": "<path>",
  "backend": {
    "serviceName": "<service name>",
    "servicePort": "<service port>"
  }
}
```

混合规则配置步骤

`TKE Ingress Controller` 支持混合配置 `HTTP` 及 `HTTPS` 规则，步骤如下：

1. **开启混合规则**在 Ingress 中添加注解 `kubernetes.io/ingress.rule-mix` ，并设置为 true。
2. **规则匹配**将 Ingress 中的每条转发规则与 `kubernetes.io/ingress.http-rules` 及 `kubernetes.io/ingress.https-rules` 进行匹配，并添加到对应规则集中。若 Ingress 注解中未找到对应规则，则默认添加到 HTTPS 规则集中。
3. **校验匹配项**匹配时请注意校验 Host、Path、ServiceName 及 ServicePort，其中 Host 默认为 `VIP` 、Path 默认为 `/` 。

> **注意：**
> IPv6 的负载均衡没有 IPv4 地址，不具备提供默认域名的功能。

示例

Ingress 示例：sample-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.http-rules: '[{"host":"www.tencent.com","path":"/","backend":
{"serviceName":"sample-service","servicePort":"80"}}]'
    kubernetes.io/ingress.https-rules: '[{"host":"www.tencent.com","path":"/","backend":
{"serviceName":"sample-service","servicePort":"80"}}]'
    kubernetes.io/ingress.rule-mix: "true"
  name: sample-ingress
  namespace: default
spec:
  rules:
    - host: www.tencent.com
      http:
        paths:
          - backend:
              service:
                name: sample-service
```

```
              port:
                number: 80
          path: /
          pathType: ImplementationSpecific
  tls:
    - secretName: tencent-com-cert
```

该示例包含以下配置：

- 描述了默认证书，证书 ID 应该存在于名为 `tencent-com-cert` 的 Secret 资源中。
- 开启了混合协议，并在 `kubernetes.io/ingress.http-rules` 及 `kubernetes.io/ingress.https-rules` 中都描述了 `ingress.spec.rule` 中描述的转发规则。

此时负载均衡会同时在 HTTP、HTTPS 中配置转发规则对外暴露服务。

Ingress 优雅停机

最近更新时间：2023-12-28 14:56:21

简介

基于接入层直连 Pod 的场景，当后端进行滚动更新或后端 Pod 被删除时，如果直接将 Pod 从 LB 的后端摘除，则无法处理 Pod 已接收但还未处理的请求。
特别是长链接的场景，例如会议业务，如果直接更新或删除工作负载的 Pod，此时会议会直接中断。

应用场景

> **注意**
> 仅针对 直连场景 生效，请检查您的集群是否支持直连模式。

- 更新工作负载时，Pod 的优雅退出，使客户端不会感受到更新时产生的抖动和错误。
- 当需要删除 Pod 时，Pod 能够处理完已接收到的请求。此时，入流量关闭，但出流量仍然可以正常传输。直到处理完所有已有请求和 Pod 真正删除时，出入流量才进行关闭。

操作步骤

步骤1：使用 Annotation 标明使用优雅停机

以下为使用 Annotation 标明使用优雅停机示例，完整 Ingress Annotation 说明可参见 Ingress Annotation 说明 文档。

```
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
  annotations:
    ingress.cloud.tencent.com/direct-access: "true" ## 开启直连 Pod 模式
    ingress.cloud.tencent.com/enable-grace-shutdown: "true" # 表示使用优雅停机
  name: my-Ingress
spec:
...
```

步骤2：使用 preStop 和 terminationGracePeriodSeconds

步骤2为在需要优雅停机的工作负载里配合使用 preStop 和 terminationGracePeriodSeconds。

容器终止流程

以下为容器在 Kubernetes 环境中的终止流程：

1. Pod 被删除，此时 Pod 里有 DeletionTimestamp，且状态置为 Terminating。此时调整 CLB 到该 Pod 的权重为 0。
2. kube-proxy 更新转发规则，将 Pod 从 Ingress 的 endpoint 列表中摘除掉。
3. 如果 Pod 配置了 preStop Hook ，将会执行。
4. kubelet 将对 Pod 中各个 container 发送 SIGTERM 信号，以通知容器进程开始优雅停止。
5. 等待容器进程完全停止，如果在 terminationGracePeriodSeconds 内 (默认30s) 还未完全停止，将发送 SIGKILL 信号强制停止进程。
6. 所有容器进程终止，清理 Pod 资源。

具体操作步骤

1. **使用 preStop**
   要实现优雅终止，务必在业务代码里处理 SIGTERM 信号。主要逻辑是不接受新的流量进入，继续处理存量流量，所有连接全部断开才退出。了解更多可参见 示例 。
   若您的业务代码中未处理 SIGTERM 信号，或者您无法控制使用的第三方库或系统来增加优雅终止的逻辑，也可以尝试为 Pod 配置 preStop，在其实现优雅终止的逻辑，示例如下：

   ```
   apiVersion: v1
   kind: Pod
   metadata:
   ```

```
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
          - /clean.sh
```

更多关于 preStop 的配置请参见 Kubernetes API 文档。

在某些极端情况下，Pod 被删除的一小段时间内，仍然可能有新连接被转发过来，因为 kubelet 与 kube-proxy 同时 watch 到 Pod 被删除，kubelet 有可能在 kube-proxy 同步完规则前就已停止容器，这时可能导致一些新的连接被转发到正在删除的 Pod，而通常情况下，当应用收到 SIGTERM 后都不再接受新连接，只保持存量连接继续处理，因此可能导致 Pod 删除的瞬间部分请求失败。

针对上述情况，可以利用 preStop 先 sleep 短暂时间，等待 kube-proxy 完成规则同步再开始停止容器内进程。示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
          - sleep
          - 5s
```

2. **使用 terminationGracePeriodSeconds 调整优雅时长**

   如果需要的优雅终止时间比较长 (preStop + 业务进程停止可能超过 30s)，可根据实际情况自定义 terminationGracePeriodSeconds，避免过早的被 SIGKILL 停止，示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: grace-demo
spec:
  terminationGracePeriodSeconds: 60 # 优雅停机默认30s，您可以设置更长的时间
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      preStop:
        exec:
          command:
          - sleep
          - 5s
```

相关能力

优雅停机只是在 Pod 删除时，才把 CLB 后端的权重置为 0。若 Pod 在运行的过程中，出现了不健康的情况，此时将该后端的权重置为 0，可以减少服务不可用的风险。

您可以使用 Annotation： `ingress.cloud.tencent.com/enable-grace-shutdown-tkex: "true"` 实现这样优雅退出的能力。

该 Annotation 会根据 Endpoint 对象中 endpoints 是否 not-ready，将 not-ready 的 CLB 后端权重置为 0。

Ingress 证书配置

最近更新时间：2025-05-21 14:34:42

操作场景

本文档介绍 Ingress 证书使用相关的内容，您可在以下场景中进行 Ingress 证书配置：
- 创建 Ingress 选用 HTTPS 监听协议时，选用合适的服务器证书能够确保访问安全。
- 为所有的 HTTPS 域名绑定同一个证书，简化配置 Ingress 下所有 HTTPS 规则的证书，使更新操作更加便捷。
- 为不同的域名绑定不同的证书，改善服务器与客户端 SSL/TLS。

注意事项

- 需提前创建需配置的证书，详情请参见 通过控制台新建服务器证书 。
- 需使用 Secret 形式来设置 Ingress 证书，其内容包含证书 ID。
- 若您需要更换证书，建议在证书平台新建一个证书，然后更新 Secret 的证书 ID。因为集群中组件的同步会以 Secret 的声明为准，若您直接在其他证书服务、负载均衡服务上更新的证书，将会被 Secret 里的内容还原。
- Secret 证书资源需和 Ingress 资源放置在同一个 Namespace 下。
- 通常情况下，在创建 Ingress 时，不会复用 Secret 关联的证书资源。但仍支持在创建 Ingress 复用 Secret 关联的证书资源，更新 Secret 时，会同步更新所有引用该 Secret 的 Ingress 的证书。
- 为域名增加匹配证书后，将同步开启负载均衡 CLB SNI 功能（不支持关闭）。若删除证书对应的域名，则该证书将默认匹配 Ingress 所对应的 HTTPS 域名。
- 传统型负载均衡不支持基于域名和 URL 的转发，由传统型负载均衡创建的 Ingress 不支持配置多证书。

示例

TKE 支持通过 Ingress 中的 `spec.tls` 的字段，为 Ingress 创建的 CLB HTTPS 监听器配置证书。其中，secretName 为包含腾讯云证书 ID 的 Kubernetes Secret 资源。示例如下：

Ingress

通过 YAML 创建：

```
spec:
    tls:
    - hosts:
      - www.abc.com
      secretName: secret-tls-2
```

Secret

通过 YAML 创建

```
apiVersion: v1
stringData:
    qcloud_cert_id: Xxxxxxxx ## 配置证书 ID 为 Xxxxxxxx
    qcloud_ca_cert_id: Xxxxxxxx ## 配置证书 ID 为 Xxxxxxxx。仅配置双向证书时需要，见下面的“注意”。
kind: Secret
metadata:
    name: tencent-com-cert
    namespace: default
type: Opaque
```

通过控制台创建

您可以通过**容器服务控制台**创建，操作详情可参考 创建 Secret 。
在**新建 Secret** 页面，Secret 主要参数配置如下：
- **名称**：自定义，本文以 cos-secret 为例。
- **Secret 类型**：选择 **Opaque**，该类型适用于保存密钥证书和配置文件，Value 将以 Base64 格式编码。
- **生效范围**：按需选择，需确保与 Ingress 在同一 Namespace 下。
- **内容**：变量名设置为 `qcloud_cert_id` ，变量值配置为服务器证书所对应的证书 ID。

注意：
若您需要配置双向证书，则 Secret 除了要添加“服务器证书”外，还需要添加“客户端 CA 证书”。此时该 Secret 还需要额外添加一个键值对：变量名为： `qcloud_ca_cert_id` ，变量值配置为“客户端CA证书”所对应的证书 ID。

Ingress 证书配置行为

- 仅配置单个 `spec.secretName` 且未配置 hosts 的情况下，将会为所有的 HTTPS 的转发规则配置该证书。示例如下：

```
spec:
    tls:
    - secretName: secret-tls
```

- 支持配置一级泛域名统配。 示例如下：

```
spec:
  tls:
    - hosts:
        - "*.abc.com"
      secretName: secret-tls
```

- 若同时配置证书与泛域名证书，将优先选择一个证书。 示例如下， `www.abc.com` 将会使用 `secret-tls-2` 中描述的证书。

```
spec:
  tls:
    - hosts:
        - "*.abc.com"
      secretName: secret-tls-1
    - hosts:
        - www.abc.com
      secretName: secret-tls-2
```

- 对已使用多个证书的 Ingress 进行更新时，TKE Ingress controller 将进行以下行为判断：
  - HTTPS 的 rules.host 无任何匹配时，若判断不通过，则不能提交更新。
  - HTTPS 的 rules.host 匹配中单个 TLS 时，可提交更新，并为该 host 配置对 Secret 中对应的证书。
  - 修改 TLS 的 SecretName 时仅校验 SecretName 的存在性，而不校验 Secret 内容，Secret 存在即可提交更新。

  注意：
  请确保 Secret 中证书 ID 符合要求。

操作步骤

通过控制台新建服务器证书

说明：
若您已具备需配置的证书，则请跳过此步骤。

1. 登录 SSL 证书控制台，选择左侧导航栏中的 我的证书 。
2. 根据自身需求选择**购买证书**、**申请免费证书**或**上传证书**的方式来创建证书。

创建使用证书的 Ingress 对象

注意事项：

TLS 配置域名与证书的对应关系如下：

- 可以使用一级泛域名统配。
- 若域名匹配中多个不同的证书，将随机选择一个证书，不建议相同域名使用不同证书。
- 需为所有 HTTPS 域名配置证书，否则会创建不通过。

操作步骤：

参考 创建 Ingress 完成 Ingress 新建，其中监听端口勾选 **Https:443**。

修改证书

注意事项：

- 如果您需要修改证书， 请确认所有使用该证书的 Ingress。如用户的多个 Ingress 配置使用同一个 Secret 资源，那么这些 Ingress 对应 CLB 的证书会同步变更。
- 证书需要通过修改 Secret 进行修改， Secret 内容中包含您使用的腾讯云证书的 ID。
- TKE 会将用户配置在 Secret 中的证书同步到 CLB， 请勿在 CLB 或证书平台直接修改 TKE ingress 关联的 CLB 的证书，否则可能导致修改被覆盖掉。

操作步骤：

1. 执行以下命令，使用默认编辑器打开需修改的 Secret。其中， `[secret-name]` 需更换为需修改的 Secret 的名称。
```
kubectl edit secrets [secret-name]
```
2. 修改 Secret 资源，将 `qcloud_cert_id` 的值修改为新的证书 ID。
与创建 Secret 相同，修改 Secret 证书 ID 需要进行 Base64 编码，请根据实际需求选择 Base64 手动编码或者指定 `stringData` 进行 Base64 自动编码。

更新 Ingress 对象

通过控制台更新

1. 登录 腾讯云容器服务控制台 ，选择左侧导航栏中的**集群**。
2. 在**集群**页面，选择集群 ID，进入集群详情页。
3. 在**服务与路由** > **Ingress** 中，单击目标 Ingress 所在行右侧的**更新转发配置**。如下图所示：

4. 在**更新转发配置**页面，根据实际情况进行转发配置规则更新。
5. 单击**更新转发配置**即可完成更新操作。

通过 YAML 更新

执行以下命令，使用默认编辑器打开需修改的 ingress，修改 yaml 文件并保存即可完成更新操作。

```
kubectl edit ingress <ingressname> -n <namespaces>
```

Ingress 配置双算法证书

最近更新时间：2025-01-07 10:49:12

简介

TKE Ingress 支持为一个域名配置一个证书，详情请参见 Ingress 证书配置。同时，也支持为一个域名配置两个不同算法类型的证书，如 ECC 和 RSA。本文将介绍其具体使用方法。

支持版本

配置双证书的 ingress-controller 组件版本最低要求为 v2.5.0。

> **说明：**
> 您可以通过 提交工单 确认组件版本是否满足要求，也可以通过查看 `kube-system` 命名空间下 `tke-service-controller-config` 和 `tke-ingress-controller-config` 的 ConfigMap 中的 `VERSION` 字段值来确认 service-controller 和 ingress-controller 版本。

注意事项

- Secret 最多支持两个不同算法类型的证书。
- Secret 的类型必须为 `Opaque`。
- 使用双算法证书的 Secret，暂不支持在 SSL 控制台自动更新，用户需要手动在 Secret 中替换新的证书 ID，TKE 会将新的证书 ID 同步到 CLB。
- 使用双算法证书的 Secret 暂时只支持配置给 Ingress，不支持特殊协议监听器。
- Secret 需与 Ingress 在同一个命名空间。

操作步骤

1. 创建 Opaque 类型 Secret

方式1：使用 stringData 区域的键值对，value 会自动 BASE64 加密

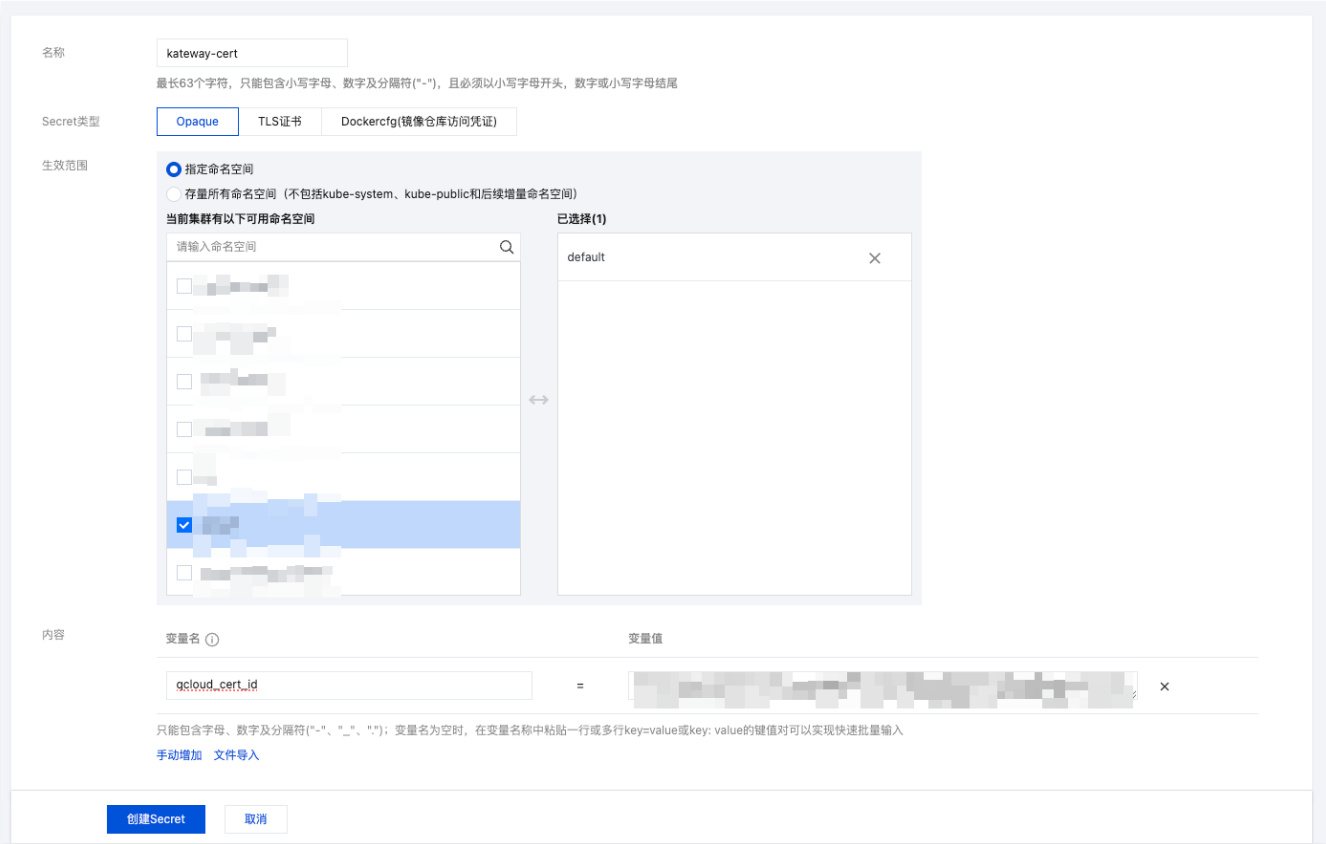
```
apiVersion: v1
stringData:
  qcloud_cert_id: CertID1,CertID2 # 填写证书ID，用逗号（英文）作为分隔符
kind: Secret
metadata:
  name: kateway-cert
type: Opaque
```

方式2：用户手动对证书 ID 进行 BASE64 加密

```
apiVersion: v1
data:
  qcloud_cert_id: (base64) CertID1,CertID2 # 提前对 CertID1,CertID2 BASE64加密
kind: Secret
metadata:
  name: kateway-cert
type: Opaque
```

方式3：在控制台创建 Secret（无需手动 BASE64 加密）

通过控制台创建 Secret，操作步骤请参见 创建 Secret。



2. 部署工作负载

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
```

3. 创建 NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
    service: nginx
spec:
```

```
  ports:
  - name: http
    port: 80
    targetPort: 80
  selector:
    app: nginx
  type: NodePort
```

4. 创建 Ingress，域名关联 Secret

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
spec:
  rules:
  - host: <domain>
    http:
      paths:
      - backend:
          service:
            name: nginx
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts: # 有 hosts 会默认开启 sni，这样证书会配置到 rule 级别
    - <domain>
    secretName: <secret>
```

说明：

- `<domain>` 替换成您的域名。
- `<secret>` 替换成您的 Secret。

Ingress Annotation 说明

最近更新时间：2025-06-11 10:01:22

说明

查看当前集群 ingress-controller 版本：
`kubectl -n kube-system get cm tke-ingress-controller-config -o jsonpath='{.data.VERSION}'`

列表

指定 ingress 类型

名称： kubernetes.io/ingress.class
说明：
指定 ingress 类型。如果没有指定，则认为是 qcloud，由 TKE 识别为 CLB 类型。
类型： string
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: 'qcloud'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

负载均衡 ID

名称： kubernetes.io/ingress.qcloud-loadbalance-id
说明：
只读注解，组件提供当前 Ingress 引用的负载均衡 LoadBalanceId。
类型： string
版本要求： >=v1.0.0

复用已有负载均衡

名称： kubernetes.io/ingress.existLbId
说明：
使用已存在的 CLB，需注意不同使用方式对腾讯云标签的影响。
详情参考 容器服务 Ingress 使用已有 CLB 。
类型： string
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.existLbId: 'lb-6swtxxxx'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

创建内网负载均衡

名称： kubernetes.io/ingress.subnetId
说明：
在指定子网创建内网负载均衡。
类型： enum
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.subnetId: 'subnet-xxx'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

指定付费类型

名称： kubernetes.io/ingress.internetChargeType
说明：
负载均衡的付费类型，当前仅在创建时支持配置，创建后不支持修改付费类型，创建后修改本注解无效。
指定创建负载均衡时，负载均衡的付费类型。请配合 kubernetes.io/ingress.internetMaxBandwidthOut 注解一起使用。
可选值：
- BANDWIDTH_POSTPAID_BY_HOUR：按带宽按小时后计费
- TRAFFIC_POSTPAID_BY_HOUR：按流量按小时后计费

类型： enum
版本要求： >=v1.0.0

使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.internetChargeType: 'BANDWIDTH_POSTPAID_BY_HOUR'
    kubernetes.io/ingress.internetMaxBandwidthOut: '10'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

指定公网带宽上限

名称： kubernetes.io/ingress.internetMaxBandwidthOut
说明：
CLB 带宽设置，当前仅在创建时支持配置，创建后不支持修改带宽，创建后修改本注解无效。
指定创建负载均衡时，负载均衡的最大出带宽，仅对公网属性的 LB 生效。需配合 kubernetes.io/ingress.internetChargeType 注解一起使用。
值域： [1, 2048] Mbps。
类型： int
版本要求： >=v1.0.0
使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.internetMaxBandwidthOut: '10'
    kubernetes.io/ingress.internetChargeType: 'TRAFFIC_POSTPAID_BY_HOUR'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

负载均衡拓展参数

名称： kubernetes.io/ingress.extensiveParameters
说明：

指定 CLB 创建时的参数，当前仅在创建时支持配置，创建后不支持修改，创建后修改本注解无效。
详情参考 创建负载均衡实例 为创建负载均衡追加自定义参数。
类型： json
版本要求： >=v1.0.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.extensiveParameters: '{"VipIsp":"CTCC"}'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

安全组默认放通

名称： ingress.cloud.tencent.com/pass-to-target
说明：
是否启用 CLB 安全组默认放通的能力，CLB 和 CVM 之间默认放通，来自 CLB 的流量只需通过 CLB 上安全组的校验。
注意：
- 请查看 CLB 使用安全组的 使用限制 。
- 通常需要配合绑定安全组的能力。对应 Annotation 为：ingress.cloud.tencent.com/security-groups。

类型： bool
版本要求： >=v1.8.3
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/pass-to-target: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

绑定/解绑安全组

名称： ingress.cloud.tencent.com/security-groups
说明：
为 CLB 绑定安全组，单个 CLB 最多可绑定5个安全组。
注意：

- 请查看 CLB 使用安全组的 使用限制 。
- 通常需要配合安全组默认放通的能力，CLB 和 CVM 之间默认放通，来自 CLB 的流量只需通过 CLB 上安全组的校验。对应 Annotation 为：ingress.cloud.tencent.com/pass-to-target。
- 对于 Ingress 使用已有 CLB 场景，不支持用该注解为 CLB 配置安全组。

类型： string
版本要求： >=v1.8.3
使用示例：

- **绑定安全组：**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/security-groups: 'sg-xxxxx,sg-yyyyy,sg-zzzzz'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

- **解绑部分安全组：**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/security-groups: 'sg-xxxxx'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

- **解绑全部安全组（>=v2.5.2）：**

```
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/security-groups: ''
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

修改保护

名称： ingress.cloud.tencent.com/modification-protection

说明：

配置 CLB 修改保护，开启后将无法通过 CLB 控制台或 API 修改负载均衡实例的属性。

类型： bool

版本要求： >=v1.7.3

使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/modification-protection: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

自定义监听端口

名称： ingress.cloud.tencent.com/listen-ports

说明：

自定义 Ingress 的监听端口。

默认情况下，HTTP 为 80，HTTPS 为 443，通过该 Annotation 可以自定义 Ingress 对应的 CLB 所监听的端口，支持分别为 HTTP 和 HTTPS 协议自定义一个或多个端口。

注意：

如需使用自动重定向 (ingress.cloud.tencent.com/auto-rewrite: "true")，仅在 HTTP：80 以及 HTTPS：443 均存在时生效。

类型： json

版本要求： >=v2.4.1

使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/listen-ports: '[{"HTTP": 80}, {"HTTPS": 443}, {"HTTP": 8080}, {"HTTPS":
8443}]'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

使用拓展配置

名称： ingress.cloud.tencent.com/tke-service-config
说明：
通过 tke-service-config 配置负载均衡 CLB。
详情参考 容器服务 Ingress 使用 TkeServiceConfig 配置 CLB 。
类型： string
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/tke-service-config: 'config-name'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

使用自动生成拓展配置

名称： ingress.cloud.tencent.com/tke-service-config-auto
说明：
通过该注解可自动创建 TkeServiceConfig，然后可按需修改配置。
详情参考 容器服务 Ingress 使用 TkeServiceConfig 配置 CLB 。
类型： bool

版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/tke-service-config-auto: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

自动重定向

名称： ingress.cloud.tencent.com/auto-rewrite
说明：
为 HTTP 端口提供自动重定向能力，所有在 HTTPS 端口声明的转发规则都会创建对应的重定向规则。
详情参考 容器服务 Ingress 重定向 。
注意：
如果之前已启用重定向功能，但现在需要关闭重定向功能，请显式将该注解设置为 "false"，而不是直接删除注解。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/auto-rewrite: 'true'
    ingress.cloud.tencent.com/rewrite-support: "true"
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

支持重定向

名称： ingress.cloud.tencent.com/rewrite-support

说明：
开启重定向支持，配合 kubernetes.io/ingress.http-rules、kubernetes.io/ingress.https-rules 实现手动重定向能力。
注意：
如果之前已启用重定向功能，但现在需要关闭重定向功能，请显式将该注解设置为 "false"，而不是直接删除注解。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/rewrite-support: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

混合协议

名称： kubernetes.io/ingress.rule-mix
说明：
支持配置混合协议，支持转发路径同时在 HTTP 和 HTTPS 上进行转发。支持手动配置重定向规则。
详情参考 Ingress 混合使用 HTTP 及 HTTPS 协议。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.rule-mix: 'true'
    kubernetes.io/ingress.http-rules: '[{"host":"kateway.tke-demo.cn","path":"/both","backend":
{"serviceName":"anyserver","servicePort":"7000"}},{"host":"kateway.tke-demo.cn","path":"/only-
http","backend":{"serviceName":"anyserver","servicePort":"7000"}}]'
    kubernetes.io/ingress.https-rules: '[{"host":"kateway.tke-demo.cn","path":"/both","backend":
{"serviceName":"anyserver","servicePort":"7000"}}]'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
```

```
        path: /
        pathType: ImplementationSpecific
```

配置 http 转发规则

名称： kubernetes.io/ingress.http-rules
说明：
配置 HTTP 转发规则，支持手动配置重定向规则。
详情参考 Ingress 混合使用 HTTP 及 HTTPS 协议。
类型： json
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.http-rules: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

配置 https 转发规则

名称： kubernetes.io/ingress.https-rules
说明：
配置 HTTPS 转发规则，支持手动配置重定向规则。
详情参考 Ingress 混合使用 HTTP 及 HTTPS 协议。
类型： json
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.https-rules: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
```

```
              number: 80
        path: /
        pathType: ImplementationSpecific
```

开启直连

名称： ingress.cloud.tencent.com/direct-access
说明：
支持七层直连用户负载均衡。需要注意在各种不同的网络下，直连接入的服务依赖。
详情参考 使用 LoadBalancer 直连 Pod 模式 Service 。
类型： bool
版本要求： >=v1.3.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

优雅停机

名称： ingress.cloud.tencent.com/enable-grace-shutdown
说明：
支持 CLB 直连模式的优雅停机。Pod 被删除，此时 Pod 里有 DeletionTimestamp，且状态置为 Terminating。此时调整 CLB 到该 Pod 的权重为 0。
仅在直连模式下支持，需要配合使用 ingress.cloud.tencent.com/direct-access。
v2.2.0 版本开始废弃，默认开启。
详情参考 Ingress 优雅停机 。
类型： bool
版本要求： >=v1.5.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/enable-grace-shutdown: 'true'
    ingress.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
```

```
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

优雅停机 tkex

名称： ingress.cloud.tencent.com/enable-grace-shutdown-tkex
说明：
支持 CLB 直连模式的优雅退出。Endpoint 对象中 endpoints 是否 not-ready，将 not-ready 的 CLB 后端权重置为 0。
仅在直连模式下支持，需要配合使用 ingress.cloud.tencent.com/direct-access。
v2.2.0 版本开始废弃，默认开启。
详情参考 Ingress 优雅停机 。
类型： bool
版本要求： >=v1.5.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/enable-grace-shutdown-tkex: 'true'
    ingress.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

优雅删除

名称： ingress.cloud.tencent.com/enable-grace-deletion
说明：
开启后，运行时等待控制器将所有 Pod 的权重调整为 0 后，再删除后端。
仅在直连模式下支持，需要配合使用 ingress.cloud.tencent.com/direct-access。
详情参考 容器服务 Pod 优雅删除 。
类型： bool
版本要求： >=v2.4.0
使用示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/enable-grace-deletion: 'true'
```

```yaml
    ingress.cloud.tencent.com/direct-access: 'true'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

自定义后端权重

名称： ingress.cloud.tencent.com/lb-rs-weight
说明：
用来支持自定义后端权重，可以设定默认权重以及有状态服务的权重。
类型： json
版本要求： >=v1.6.0
使用示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.cloud.tencent.com/lb-rs-weight: '{"defaultWeight":10,"groups":[{"key":
{"proto":"TCP","port":80},"statefulSets":[{"name":"ti-ai-gateway-gz-shared-st-v1","weights":
[{"weight":0,"podIndexes":[0]}]},{"name":"ti-ai-gateway-gz-shared-st-v2","weights":
[{"weight":10,"podIndexes":[0]}]}]}]}'
  name: example
spec:
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: example
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

Nginx 类型 Ingress（停止更新）
概述

最近更新时间：2024-12-30 14:17:42

> **注意：**
> NginxIngress 扩展组件已停止更新，详情请参见 NginxIngress 扩展组件停止更新公告 。

NginxIngress 介绍

Nginx 可以用作反向代理、负载平衡器和 HTTP 缓存。
NginxIngress 是使用 Nginx 作为反向代理和负载平衡器的 Kubernetes Ingress 控制器。您可以部署 NginxIngress 组件，在集群中使用 NginxIngress。容器服务 TKE 提供了产品化的能力，帮助您在集群内安装和使用 NginxIngress。

NginxIngress 名词解释

- **NginxIngress 组件**：在 TKE 中使用 NginxIngress 的入口，您可以在集群的组件页面一键安装部署 NginxIngress。
- **NginxIngress 实例**：一个集群中可部署多个 NginxIngress（例如一个用于公网，一个用于内网）。在 Kubernetes 中对应一个 CRD，创建一个 NginxIngress 实例会在集群中自动创建 Nginx-ingress-controller、service、configmap 等 Kubernetes 资源。
- **Nginx-ingress-controller**：实际 Nginx 负载，同时 controller 会 watch kubernetes ingress 对象的变化更新在集群中，Nginx 负载的转发配置即 `nginx.conf` 文件。

NginxIngress 相关操作

您可以参考以下文档进一步了解 NginxIngress 的相关操作及功能：
- NginxIngress 安装
- 使用 NginxIngress 对象接入集群外部流量
- NginxIngress 日志配置

NginxIngress 相关链接

- Github 首页：https://github.com/kubernetes/ingress-nginx
- 官网文档：https://kubernetes.github.io/ingress-nginx/

关于可用性保障

NginxIngress 是开源社区组件，TKE 仅提供安装，不提供可用性保障，需用户自行保障。

安装 NginxIngress 实例

最近更新时间：2024-11-07 16:54:32

> **注意：**
> NginxIngress 扩展组件已停止更新，详情请参见 NginxIngress 扩展组件停止更新公告 。

安装 NginxIngress 组件

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在**组件管理**页面，单击**新建**。
4. 在**新建组件管理**页面中勾选 NginxIngress。
5. 单击**完成**即可安装组件。安装完成后，您可以在**组件管理**中查看组件详情。

注意事项

腾讯云负载均衡（Cloud Load Balancer）实例已于2023年03月06日升级了架构，升级后公网负载均衡以域名的方式提供服务。VIP 随业务请求动态变化，控制台不再展示 VIP 地址。请参见 域名化公网负载均衡上线公告 。

- 新注册的腾讯云用户默认使用升级后的域名化负载均衡。
- 存量用户可以选择继续使用原有的负载均衡，不受升级影响。如果您需要升级负载均衡服务，则需要同时升级腾讯云产品 CLB 以及 TKE，否则 TKE 中的所有公网类型的 Service/Ingress 同步将可能受到影响。CLB 升级操作详情请参见 域名化负载均衡升级指南；TKE 升级 Service/Ingress 组件版本，请通过 在线咨询 联系我们。

安装方式

您可以根据不同的业务场景需求，使用以下几种安装方案在容器服务 TKE 中安装 NginxIngress。

方案1：通过 DaemonSet 形式在指定节点池部署（推荐）

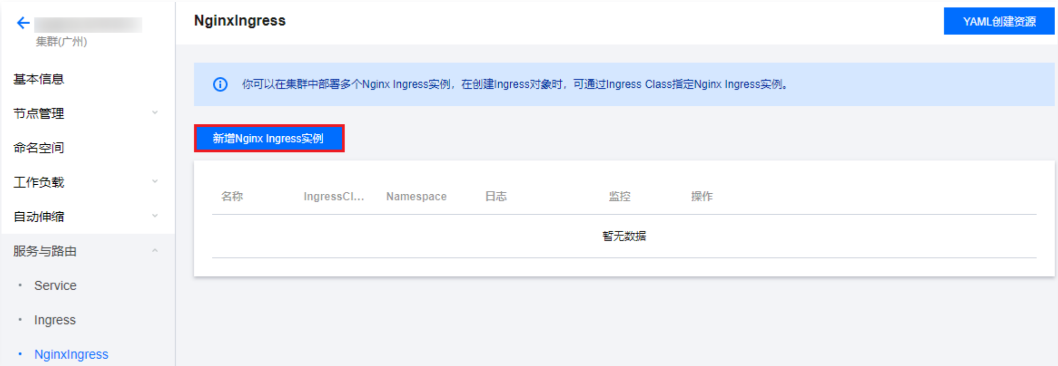
Nginx 作为关键的流量接入网关，不建议您将 Nginx 与其他业务部署在相同的节点内。推荐您使用指定的节点池来部署 NginxIngress。部署架构如下图所示：

安装步骤

> **注意：**
> - 使用此安装方式，您可以完整享有节点池快速扩缩容的能力，后续您只要调整节点池的数量，即可扩缩容 Nginx 的副本。
> - DaemonSet 方式无法选择**注册节点池**。

1. 准备用于部署 NginxIngress 的节点池，同时设置污点 taint（防止其他 Pod 调度到该节点池）。部署节点池详情可参见 节点池相关说明 。
2. 在集群中 安装 NginxIngress 组件 。

3. 在集群信息页中，选择**服务与路由 > NginxIngress**，单击**新增 NginxIngress 实例**（一个集群内可以同时存在多个 Nginx）。



4. 在**新建 NginxIngress** 中，选择部署选项中的**指定节点池 DaemonSet 部署**，并按需设置其他参数。如下图所示：

- **节点池**：配置节点池。
- **Nginx 配置**：Request 需设置比节点池的机型配置小（节点本身有资源预留）。Limit 可不设置。
- **镜像版本说明**：

Kubernetes 版本范围	NginxIngress 组件支持安装的版本	Nginx 实例支持的镜像版本
<=1.18	1.1.0、1.2.0、1.3.0、1.3.1、1.4.1、1.5.1	v0.41.0、v0.49.3
	1.0.0	v0.41.0
1.20	1.1.0、1.2.0、1.3.0、1.3.1、1.4.1、1.5.1	v1.1.3
	1.0.0	v0.41.0
1.22	1.1.0、1.2.0、1.3.0、1.3.1、1.4.1、1.5.1	v1.1.3
1.24	1.1.0、1.2.0、1.3.0、1.3.1、1.4.1、1.5.1	v1.1.3、v1.6.4
1.26	1.3.0、1.3.1、1.4.1、1.5.1	v1.1.3、v1.6.4、v1.9.5
1.28	1.5.1	v1.9.5

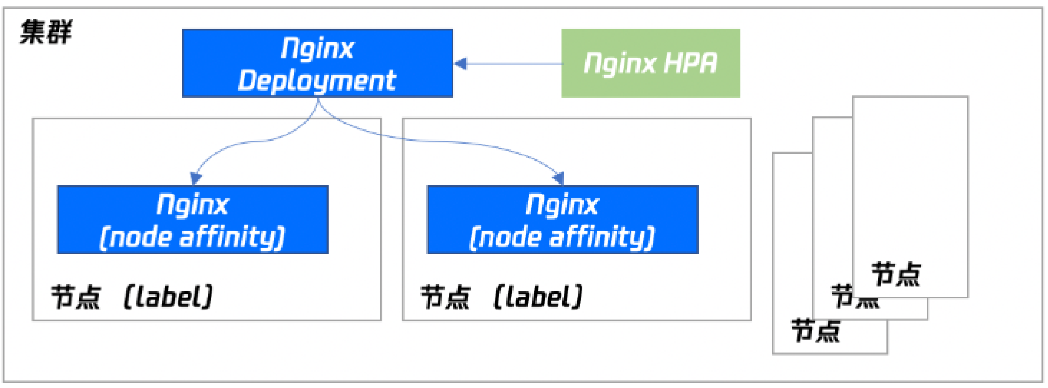
说明：

1. **组件使用 EIP 说明**：NginxIngress 组件 1.0.0 和 1.1.0 版本依赖腾讯云弹性公网 IP 服务（EIP），在 v1.2.0 版本中组件不再依赖 EIP，如果您有 EIP 使用限制的需求，建议您升级 NginxIngress 组件。组件的升级不影响存量的 NginxIngress 实例，对业务访问无影响，不影响数据安全。
2. **升级说明**：Nginx 实例的版本说明请参见 Ingress-nginx 文档。升级集群请参见 升级集群操作步骤。升级 NginxIngress 组件请参见 组件升级操作步骤。

5. 单击**确定**即可完成安装。

方案2：通过 Deployment + HPA 形式并指定调度规则部署

使用 Deployment + HPA 的形式部署 NginxIngress，您可以根据业务需要配置污点和容忍将 Nginx 和业务 Pod 分散部署。同时搭配 HPA，可设置 Nginx 根据 CPU / 内存等指标进行弹性伸缩。部署架构如下图所示：



安装步骤

1. 准备用于部署 NginxIngress 的节点池，同时设置污点 taint（防止其他 Pod 调度到该节点池）。部署节点池详情可参见 节点池相关说明 。
2. 在集群中 安装 NginxIngress 组件 。
3. 在集群信息页中，选择**服务与路由 > NginxIngress**，单击**新增 NginxIngress 实例**（一个集群内可以同时存在多个 Nginx）。
4. 在**新建 NginxIngress** 中，选择部署选项中的**自定义 Deployment+HPA 部署**，并按需设置其他参数。如下图所示：

- **自动伸缩**：可以选择关闭 nginx 实例的自动伸缩。

> **说明：**
> - 当您手动删除 NginxIngress 实例对应的 HPA 对象，平台 ApiServer 滚动更新或者 Nginx controller pod 重启，系统内部会再次对账，重新下发 HPA 对象。新下发的 HPA 可能会伸缩 Nginx controller pod 数量，对业务造成影响。
> - 当您需要关闭 HPA 时请通过此开关关闭 Nginx 实例的 HPA 功能。

- **Nginx 配置**：Request 需设置比节点池的机型配置小（节点本身有资源预留）。Limit 可不设置。
- **节点调度策略**：需自行指定。

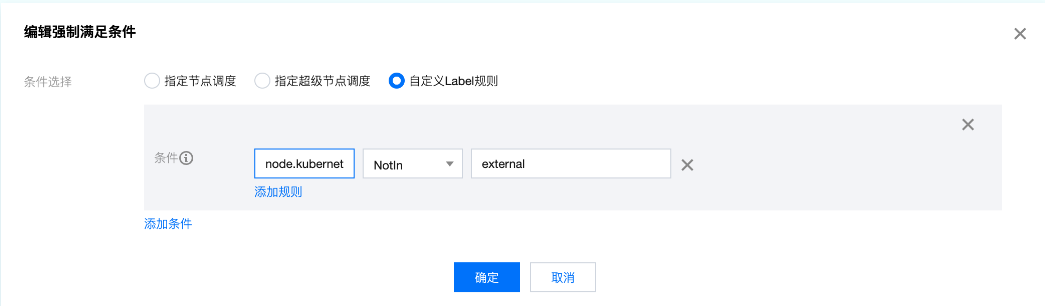
> **注意：**
> 如果您开启了“注册节点能力”，请参考下面的方式，防止 NginxIngress 实例部署到注册节点上。
>
> ```
> spec:
> ```

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: node.kubernetes.io/instance-type
          operator: NotIn
          values:
          - external
```

操作步骤如下：

1. 在**节点调度策略**中，选择自定义调度规则。
2. 在**强制满足条件**中，单击**添加条件**。如下图所示：

3. 在**编辑强制满足条件**页面，选择**自定义 Label 规则**，添加调度策略 node.kubernetes.io/instance-type NotIn external。如下图所示：



4. 单击**确定**。

- **镜像版本说明：**
  - Kubernetes 为 1.20 及以下版本的集群，NginxIngress 组件版本为 1.0.0，Nginx 实例的镜像版本只能选择 v41.0。
  - Kubernetes 为 1.20 及以下版本的集群，NginxIngress 组件版本为 1.1.0，Nginx 实例的镜像版本只能选择 v41.0，v49.3。
  - Kubernetes 大于等于 1.22 版本的集群，NginxIngress 组件版本只支持 1.1.0，Nginx 实例的镜像版本只能选择 v1.1.3。

> **说明：**
> - 版本说明：如果您需要了解 Nginx 实例的版本信息，请参见 Ingress-nginx 文档 。
> - 升级集群：如果您需要升级集群，请参见 升级集群操作步骤 。
> - 升级 NginxIngress 组件：如果您需要升级 NginxIngress 组件，请参见 组件升级操作步骤 。

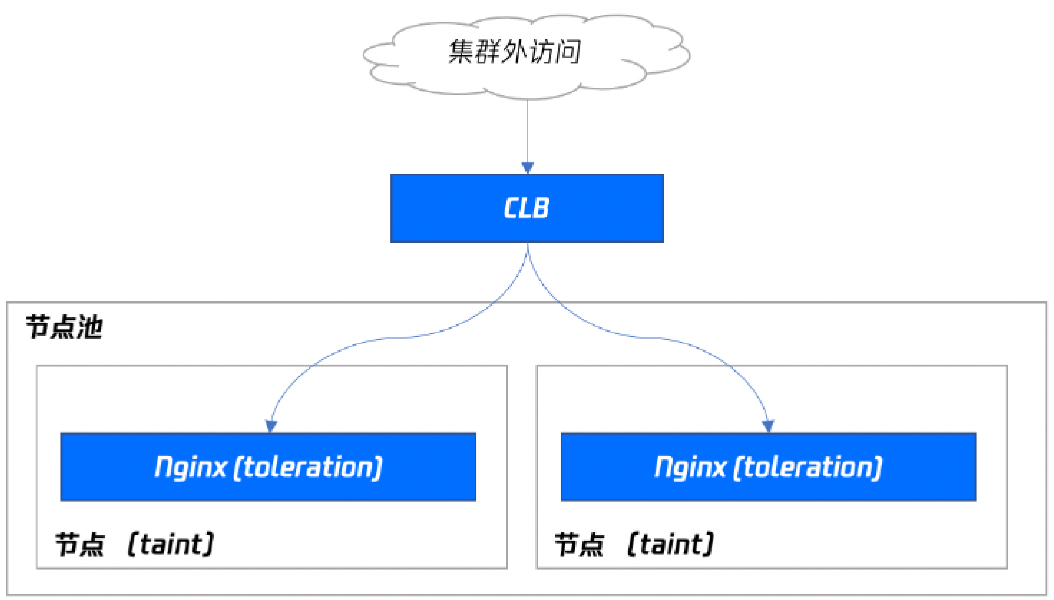
5. 单击**确定**即可完成安装。

方案3：Nginx 前端接入 LB 部署方式

仅部署 Nginx 在集群内将无法接收外部流量，还需配置 Nginx 的前端 LB。TKE 现已提供产品化的安装能力，您也可以根据业务需要选择不同的部署模式。

VPC-CNI 模式集群使用 CLB 直通 Nginx 的 Service(推荐)

如果您的集群是 VPC-CNI 模式的集群，推荐您使用 CLB 直通 Nginx 的 Serivce。下图为以节点池部署的负载示例。



当前方案性能好，而且不需要手动维护 CLB，是最理想的方案。该方案需要集群支持 VPC-CNI，如果您的集群已配置 VPC-CNI 网络插件，或者已配置 Global Router 网络插件并开启了 VPC-CNI 的支持（两种模式混用），建议使用此方案。

Globalrouter 模式集群使用普通 Loadbalancer 模式的 Service

如果您的集群不支持 VPC-CNI 模式网络，您也可以通过常规的 Loadbalancer 模式 Service 接入流量。当前 TKE 上 LoadBalancer 类型的 Service 默认实现是基于 NodePort，CLB 会绑定各节点的 NodePort 作为后端 RS，将流量转发到节点的 NodePort，然后节点上再通过 iptables 或 ipvs 将请求路由到 Service 对应的后端 Pod。这种方案是最简单的方案，但流量会经过一层 NodePort，会多一层转发。可能存在以下问题：

- 转发路径较长，流量到了 NodePort 还会再经过 k8s 内部负载均衡，通过 iptables 或 ipvs 转发到 Nginx，会增加一点网络耗时。
- 经过 NodePort 必然发生 SNAT，如果流量过于集中容易导致源端口耗尽或者 conntrack 插入冲突导致丢包，引发部分流量异常。
- 每个节点的 NodePort 也充当一个负载均衡器，CLB 如果绑定大量节点的 NodePort，负载均衡的状态会分散在每个节点上，容器导致全局负载不均。
- CLB 会对 NodePort 进行健康探测，探测包最终会被转发到 NginxIngress 的 Pod，如果 CLB 绑定的节点多，NginxIngress 的 Pod 少，会导致探测包对 NginxIngress 造成较大的压力。

使用 HostNetwork + LB 模式

控制台暂不支持，您可以手动修改 Nginx 工作负载的 Yaml 配置网络模式为 HostNetwork，手动创建 CLB 绑定 Nginx 暴露的节点端口。
需要注意使用 hostNetwork 时，为避免端口监听冲突，NginxIngress 的 Pod 不能被调度到同一节点。

TKE 安装 NginxIngress 默认参数

设置 NginxIngress 参数

您可以在 NginxIngress 组件详情页，Nginx 参数 tab 中选择 NginxIngress 实例进行 YAML 编辑。

> **注意：**
> 默认情况下配置参数不会重启 Nginx，生效时间有细微延迟。

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 在**服务与路由 > NginxIngress** 页面，单击实例右侧的**查看 YAML**。
4. 在 NginxIngress 实例详情页，选择 **Nginx 配置**页签，并单击**编辑 YAML**。
5. 编辑 YAML 后，单击**完成**即可配置参数。

配置参数示例

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: alpha-ingress-nginx-controller
  namespace: kube-system
data:
  access-log-path: /var/log/nginx/nginx_access.log
  error-log-path: /var/log/nginx/nginx_error.log
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request" $status
$body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time [$proxy_upstream_name]
[$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_length] [$upstream_response_time]
[$upstream_status] $req_id
  keep-alive-requests: "10000"
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
```

注意：

- 请勿修改 `access-log-path` 、 `error-log-path` 、 `log-format-upstream` 。若修改则会对 CLS 日志采集造成影响。
- 若您需要根据业务配置不同的参数，请参见 官方文档 。

使用 NginxIngress 对象接入集群外部流量

最近更新时间：2024-11-07 16:54:32

> **注意：**
> NginxIngress 扩展组件已停止更新，详情请参见 NginxIngress 扩展组件停止更新公告 。

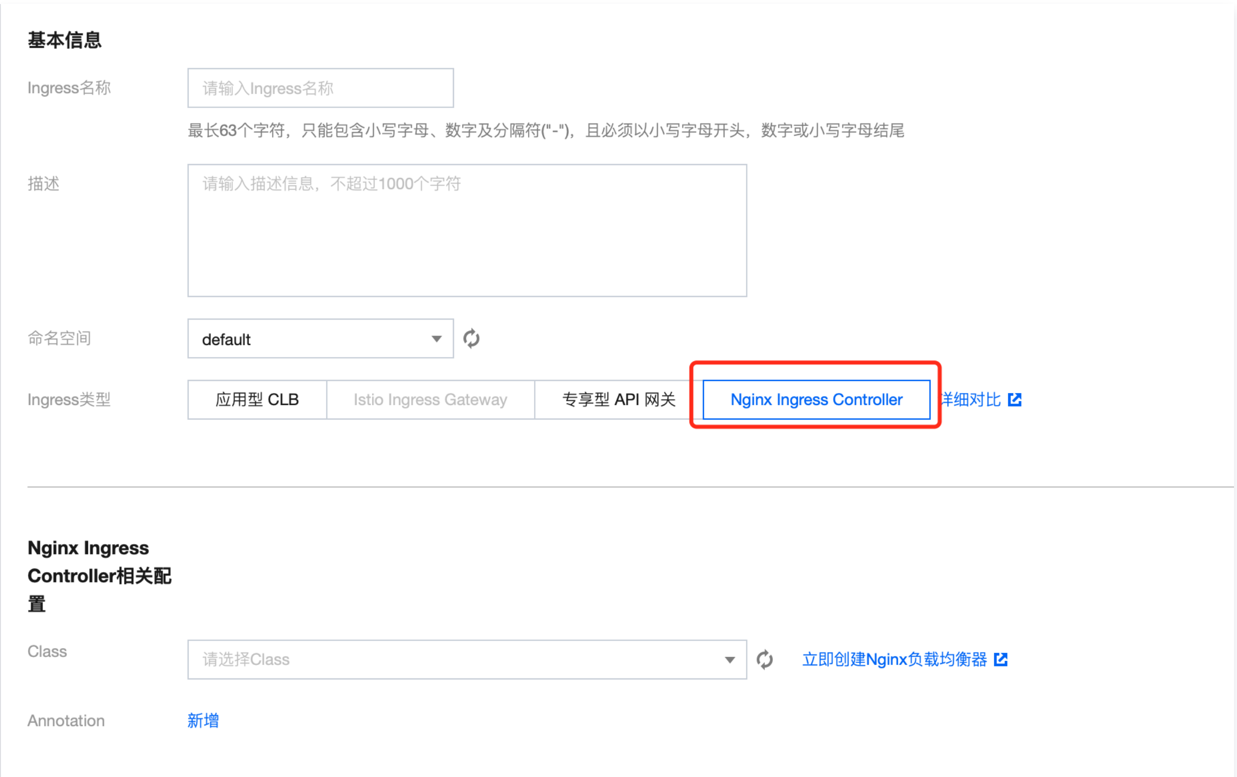
前提条件

- 已登录 容器服务控制台 。
- 集群内已 部署 NginxIngress 组件 。
- 已安装并创建业务需要的 NginxIngress 实例。

使用方法

NginxIngress 控制台操作指引

1. 登录 容器服务控制台 ，在左侧导航栏中单击**集群**。
2. 进入集群管理页面，单击已安装 NginxIngress 组件的集群 ID，进入集群详情页。
3. 选择**服务与路由** > **Ingress**，进入 Ingress 信息页面。
4. 单击**新建**，进入新建 Ingress 页面。
5. 根据实际需求，设置 Ingress 参数。如下图所示：



   - Ingress 类型：选择 **Nginx Ingress Controller**。
   - Class：选择一个 NginxIngress 实例，若无则单击右侧的**立即创建 Nginx 负载均衡器**。
   - 转发规则：需自行设置。
6. 单击**创建 Ingress**。

Kubectl 操作 NginxIngress 指引

在 Kubernetes 中引入 IngressClass 资源和 ingressClassName 字段之前，Ingress 类由 Ingress 中的 `kubernetes.io/ingress.class` 注解指定。
示例如下：

```
metadata:
  name:
  annotations:
    kubernetes.io/ingress.class: "nginx-public". ## 对应 TKE 集群 NginxIngress 组件中的 NginxIngress 实例名称
```

相关操作

为 Nginx 类型 Ingress 对象可配置注解，详情可参见 官方文档 。

NginxIngress 对象使用模型

当多个 Ingress 对象作用于一个 Nginx 实体时：

- 按 CreationTimestamp 字段对 Ingress 规则排序，即先按旧规则。
- 如果在多个 Ingress 中为同一主机定义了相同路径，则最早的规则将获胜。
- 如果多个 Ingress 包含同一主机的 TLS 部分，则最早的规则将获胜。
- 如果多个 Ingress 定义了一个影响 Server 块配置的注释，则最早的规则将获胜。
- 按每个 hostname 创建 NGINX Server。
- 如果多个 Ingress 为同一 host 定义了不同的路径，则 ingress-controller 合并这些定义。
- 多个 Ingress 可以定义不同的注释。这些定义在 Ingress 之间不共享。
- Ingress 的注释将应用于 Ingress 中的所有路径。

触发更新 nginx.conf 机制

以下内容描述了需要重新加载 nginx.conf 的情况：

- 创建新的 Ingress 对象。
- 为 Ingress 添加新的 TLS。
- Ingress 注解的更改不仅影响上游配置，而且影响更大。例如 load-balance 注释不需要重新加载。
- 为 Ingress 添加/删除路径。
- 删除 Ingress、Ingress 的 Service、Secret。
- Ingress 关联的对象状态不可知，例如 Service 或 Secret。
- 更新 Secret。

NginxIngress 日志配置

最近更新时间：2024-11-07 16:54:32

⚠ **注意：**
NginxIngress 扩展组件已停止更新，详情请参见 NginxIngress 扩展组件停止更新公告 。

容器服务 TKE 通过集成日志服务 CLS，提供了全套完整的产品化能力，实现 NginxIngress 日志采集、消费能力。

NginxIngress 日志基础

Nginx Controller 需要搜集以下日志并提供给用户：

- **Nginx Controller 日志**：重要。控制面日志，记录了 Nginx Controller 控制面的修改。主要用于控制面排障，例如用户错误配置 Ingress 模板导致同步未进行等。
- **AccessLog 日志**：重要。用户数据面日志，记录了用户的七层请求相关信息。主要用于提供给用户进行数据分析、审计、业务排障等。
- **ErrorLog 日志**：一般。Nginx 的内部错误日志。

默认配置下，AccessLog 和 Nginx Controller 日志会混合到标准输出流，日志采集将遇到困难。本文向您介绍如何对日志路径进行区分后分别收集日志。

前提条件

已在容器服务控制台的 功能管理 中开启日志采集，详情参见 开启日志采集 。

TKE NginxIngress 采集日志

采集日志步骤

1. 为目标集群 安装 NginxIngress 组件。
2. 在**服务与路由 > NginxIngress**中，选择已安装的实例名称，进入组件详情页。如下图所示：

3. 在**运维**标签页，选择日志配置右侧的**重新设置**。如下图所示：

4. 在弹出的窗口中选择指定的日志集，如不指定将创建新的日志集。如下图所示：

5. 单击**立即启用**即可完成日志采集配置。

> **注意：**
> 日志服务具体计费规则和收费标准请参见 CLS 计费概述 。

采集日志指标

采集日志的指标如下所示：

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: nginx-ingress-test
  resourceVersion: "7169042"
  selfLink: /apis/cls.cloud.tencent.com/v1/logconfigs/nginx-ingress-test
  uid: 67c96f86-4160-****-****-f6faf8d544dc
spec:
  clsDetail:
    extractRule:
      beginningRegex: (\S+)\s-\s(\S+)\s\[(\S+)\]\s(\S+)\s\"(\w+)\s(\S+)\s([^\"]+)\"\s(\S+)\s(\S+)\s\"
([^"]*)\"\s\"([^"]*)\"\s(\S+)\s(\S+)\s\[([^\]]*)\]\s\[([^\]]*)\]\s\[([^\]]*)\]\s\[([^\]]*)\]\s\
[([^\]]*)\]\s\[([^\]]*)\]\s(\S+)
      keys:
      - remote_addr
      - remote_user
      - time_local
      - timestamp
      - method
      - url
      - version
      - status
      - body_bytes_sent
      - http_referer
      - http_user_agent
      - request_length
      - request_time
      - proxy_upstream_name
      - proxy_alternative_upstream_name
      - upstream_addr
      - upstream_response_length
      - upstream_response_time
      - upstream_status
```

```
        - req_id
        logRegex: (\S+)\s-\s(\S+)\s\[(\S+)\]\s(\S+)\s\"(\w+)\s(\S+)\s([^\"]+)\"\s(\S+)\s(\S+)\s\"
([^"]*)\"\s\"([^"]*)\"\s(\S+)\s(\S+)\s\[([^\]]*)\]\s\[([^\]]*)\]\s\[([^\]]*)\]\s\[([^\]]*)\]\s\
[([^\]]*)\]\s\[([^\]]*)\]\s(\S+)
      logType: fullregex_log
      topicId: 56766bad-368e-****-****-ed77ebcdefa8
    inputDetail:
      containerFile:
        container: controller
        filePattern: nginx_access.log
        logPath: /var/log/nginx
        namespace: default
        workload:
          kind: deployment
          name: nginx-ingress-nginx-controller
      type: container_file
```

NginxIngress 日志仪表盘

TKE NginxIngress 开启日志采集功能后，您可以在 CLS 控制台使用预置的 Nginx 仪表盘对日志进行分析，详情请参见 Nginx 访问日志分析。也可从 NginxIngress 实例页面快捷跳转：

相关文档

如果需要自定义日志采集规则和索引，请参考 NginxIngress 自定义日志。

通过 Terraform 安装 Nginx 插件和实例

最近更新时间：2024-11-07 16:54:32

⚠ **注意：**
NginxIngress 扩展组件已停止更新，详情请参见 NginxIngress 扩展组件停止更新公告 。

前言

本文示例使用的环境信息如下：
- TKE 集群 Kubernetes 版本：v1.26.1
- 安装 Nginx 插件版本：v1.4.1
- 安装 Nginx 实例版本：v1.6.4

步骤1：安装 Terraform

您可以通过以下命令下载并安装 Terraform：

```
wget https://releases.hashicorp.com/terraform/1.6.6/terraform_1.6.6_linux_amd64.zip
```

v1.4.6版本 Release 地址为 https://releases.hashicorp.com/terraform/1.6.6/，您可以根据系统选择对应安装包。

步骤2：在集群中安装 Nginx Addon

Nginx Addon 插件是一个 Nginx 的安装管理工具。首先安装 Addon 插件，然后再使用插件安装 Nginx 实例，具体操作请参见 腾讯云Terraform 应用指南 。
provider.tf 示例文件如下：

```
# 腾讯云 provider
terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
      version = "1.81.61"
    }
  }
}

# 腾讯云 相关信息（更换密钥对 "secret_id"、"secret_key"）
provider "tencentcloud" {
    secret_id  = "********"
    secret_key = "********"
    region     = "ap-shanghai"
}

# 安装Nginx插件（更换集群ID "cluster_id"）
resource "tencentcloud_kubernetes_addon_attachment" "addon_ingressnginx" {
  cluster_id   = "cls-xxxxxxxx"
  name = "ingressnginx"
  request_body = "{\"kind\":\"App\",\"spec\":{\"chart\":
{\"chartName\":\"ingressnginx\",\"chartVersion\":\"1.4.1\"}}}"
}
```

步骤3：声明式安装 Nginx 实例

有关 Kubernetes Provider 的更多配置信息，请参见 官方文档 。
Nginx 实例的相关配置可以根据需要进行修改。
- IngressClass 配置（示例中使用的是 demo）

- HPA 配置
- requests/limits 配置

provider.tf 示例文件如下：

```
provider "kubernetes" {
  config_path = "~/.kube/config"
}

resource "kubernetes_manifest" "nginxingress_demo" {
  manifest = {
    "apiVersion" = "cloud.tencent.com/v1alpha1"
    "kind"       = "NginxIngress"
    "metadata" = {
      "name" = "demo"
    }
    "spec" = {
      "ingressClass" = "demo"
      "service" = {
        "annotation" = {
          "service.kubernetes.io/service.extensiveParameters" = "{\"InternetAccessible\":
{\"InternetChargeType\":\"TRAFFIC_POSTPAID_BY_HOUR\",\"InternetMaxBandwidthOut\":10}}"
        }
        "type" = "LoadBalancer"
      }
      "workLoad" = {
        "hpa" = {
          "enable"      = true
          "maxReplicas" = 2
          "metrics" = [
            {
              "pods" = {
                "metricName"         = "k8s_pod_rate_cpu_core_used_limit"
                "targetAverageValue" = "80"
              }
              "type" = "Pods"
            },
          ]
          "minReplicas" = 1
        }
        "template" = {
          "affinity" = {}
          "container" = {
            "image" = "ccr.ccs.tencentyun.com/tkeimages/nginx-ingress-controller:v1.6.4"
            "resources" = {
              "limits" = {
                "cpu"    = "0.5"
                "memory" = "1024Mi"
              }
              "requests" = {
                "cpu"    = "0.25"
                "memory" = "256Mi"
              }
            }
          }
        }
        "type" = "deployment"
      }
    }
  }
```

```
}
```

存储管理

概述

最近更新时间：2023-05-17 15:40:33

集群的存储管理是保存业务数据的重要组件。目前，腾讯云容器服务（Tencent Kubernetes Engine，TKE）支持多种类型的存储。

存储类型

存储类型	说明	使用方法
腾讯云硬盘（CBS）	CBS 提供数据块级别的持久性存储，通常用作需要频繁更新、细粒度更新的数据（如文件系统、数据库等）的主存储设备，具有高可用、高可靠和高性能的特点。	TKE 支持通过创建 PV/PVC，并为工作负载挂载动（静）态数据卷的方式使用云硬盘 CBS。详情请参见 使用云硬盘 CBS 。
腾讯云文件存储（CFS）	CFS 提供了标准的 NFS 及 CIFS/SMB 文件系统访问协议，为多个 CVM 实例或其他计算服务提供共享的数据源，支持弹性容量和性能的扩展，是一种高可用、高可靠的分布式文件系统，适合于大数据分析、媒体处理和内容管理等场景。	TKE 支持通过创建 PV/PVC，并为工作负载挂载动（静）态数据卷的方式使用文件存储 CFS。详情请参见 使用文件存储 CFS 。
腾讯云对象存储（COS）	COS 是腾讯云提供的一种存储海量文件的分布式存储服务，通过 COS 可以进行多格式文件的上传、下载和管理。	TKE 支持通过创建 PV/PVC，并为工作负载挂载静态数据卷的方式使用对象存储 COS。详情请参见 使用对象存储 COS 。
其他类型	-	在创建工作负载时，TKE 还支持使用以下类型的本地存储，如使用主机路径、NFS 盘、配置项（ConfigMap）、密钥（Secret）等。详情请参见 使用其他存储卷 。

> **说明**
> 建议使用云存储服务，否则当节点异常无法恢复时，本地存储的数据同样不能恢复。

相关概念

- **PersistentVolume（PV）**：集群内的存储资源。PV 独立于 Pod 的生命周期，可根据不同的 StorageClass 类型创建不同类型的 PV。
- **PersistentVolumeClaim（PVC）**：集群内的存储请求。例如，PV 是 Pod 使用的节点资源，PVC 则声明使用 PV 资源。当 PV 资源不足时，PVC 可动态创建 PV。

使用对象存储 COS

最近更新时间：2024-10-24 10:54:52

操作场景

腾讯云容器服务 TKE 支持通过创建 PersistentVolume（PV）和 PersistentVolumeClaim（PVC），并为工作负载挂载数据卷的方式使用腾讯云对象存储 COS。本文介绍如何在 TKE 集群中为工作负载挂载对象存储。

准备工作

1. 安装对象存储扩展组件

> **说明：**
> 若您的集群已安装 COS-CSI 扩展组件，则请跳过此步骤。

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在**组件管理**页面中单击**新建**。
4. 在**新建组件管理**页面，勾选 **COS（腾讯云对象存储）**组件。
5. 单击**完成**。

2. 创建访问密钥

> **注意：**
> 为避免主账号密钥泄露造成您的云上资产损失，建议您参照 安全设置策略 停止使用主账号登录控制台或者使用主账号密钥访问云 API，并使用已授予相关管理权限的子账号/协作者进行相关资源操作。
> 本文以已授予访问管理相关权限的子用户创建或查看访问密钥为例，关于如何创建子用户并实现访问管理权限请参考文档 自定义创建子用户 。

1. 使用子账号用户登录 访问管理控制台，选择左侧导航中的**访问密钥 > API 密钥管理**。
2. 在 **API 密钥管理**页面，单击**新建密钥**等待新建完成即可。

> **说明：**
> - 一个子用户最多可以创建两个 API 密钥。
> - API 密钥是构建腾讯云 API 请求的重要凭证，为了您的财产和服务安全，请妥善保存和定期更换密钥。当您更换密钥后，请及时删除旧密钥。

3. 创建存储桶

登录 对象存储控制台 并创建一个存储桶，操作详情请参见 创建存储桶 。创建完成后，在存储桶列表中进行查看。

4. 获取存储桶子目录

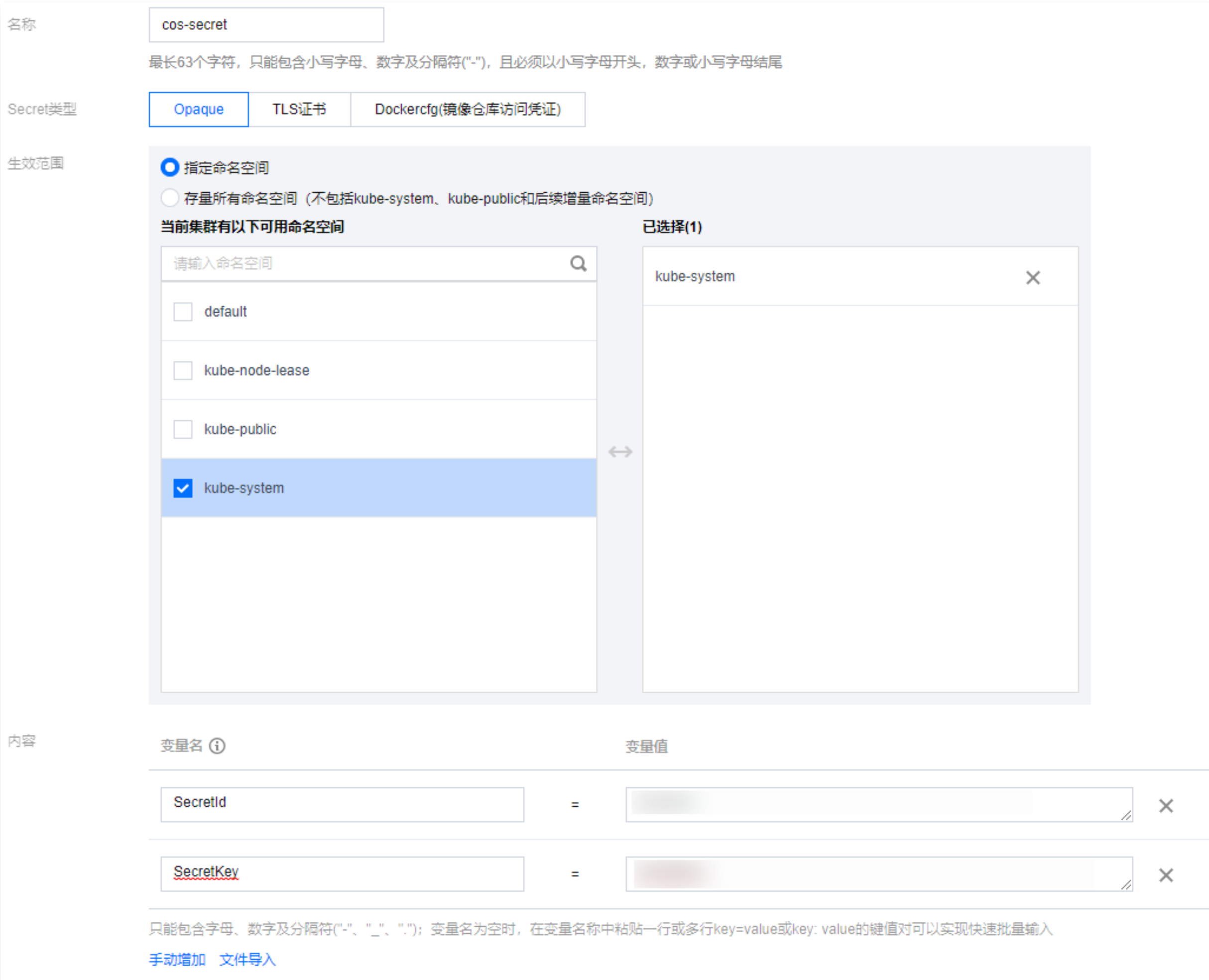
1. 在**存储桶列表**页，单击已创建的存储桶名称，进入该存储桶的详情页。
2. 选择左侧导航中的**文件列表**，在文件列表中选择需要挂载的子文件夹，进入该文件夹详情页。例如，在页面右上角获取子目录路径 `/costest` 。如下图所示：

操作步骤

通过控制台使用对象存储

步骤1：创建可以访问对象存储的 Secret

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**配置管理** > **Secret**，在 Secret 页面中单击**新建**。
4. 在**新建 Secret** 页面，根据以下信息进行设置。如下图所示：



- **名称**：自定义，本文以 `cos-secret` 为例。
- **Secret 类型**：选择 **Opaque**，该类型适用于保存密钥证书和配置文件，Value 将以 Base64 格式编码。
- **生效范围**：选择**指定命名空间**，请确保 Secret 创建在 `kube-system` 命名空间下。
- **内容**：此处用于设置 Secret 访问存储桶（Bucket）所需的访问密钥，需包含变量名 `SecretId` 和 `SecretKey` 及其分别所对应的变量值。请参考 创建访问密钥 完成创建，并前往 API 密钥管理 页面获取访问密钥。

5. 单击**创建 Secret** 即可。

步骤2：创建支持 COS-CSI 静态配置的 PV

> **注意：**
> 本步骤需使用存储桶，若当前地域无可用存储桶，则请参考 创建存储桶 进行创建。

1. 在目标集群详情页面，选择左侧菜单栏中的**存储** > **PersistentVolume**，在 **PersistentVolume** 页面单击**新建**。
2. 在**新建 PersistentVolume** 页面，参考以下信息创建 PV。如下图所示：

3. 主要参数信息如下：
  - **来源设置**：选择**静态创建**。
  - **名称**：自定义，本文以 `cos-pv` 为例。
  - **Provisioner**：选择为**对象存储 COS**。
  - **读写权限**：对象存储仅支持多机读写。

> **说明：**
> - 单机读写：当前仅支持云硬盘同时挂载到一台机器上，因此只能处理单机器的数据读写。
> - 多机读写：文件存储/对象存储支持同时挂载到多台机器，可以处理多机器的数据读写。

  - **Secret**：选择已在 步骤1 创建的 Secret，本文以 `cos-secret` 为例（请确保 Secret 创建在 `kube-system` 命名空间下）。
  - **存储桶列表**：用于保存对象存储中的对象，按需选择可用存储桶即可。
  - **存储桶子目录**：填写已在 获取存储桶子目录 中获取的存储桶子目录，本文以 `/costest` 为例。

> **注意：**
> - COSFS 挂载方式，若填写的子目录不存在，则系统将为您自动创建。
> - GooseFS-Lite 挂载方式，填写的子目录需已存在，请前往对象存储控制台提前新建。

  - **域名**：展示为默认域名，您可以使用该域名对存储桶进行访问。
  - **挂载选项**：
    - COSFS 工具支持将存储桶挂载到本地，挂载后可直接操作对象存储中的对象，此项用于设置相关限制条件。本例中挂载选项 `-oensure_diskfree=20480` 表示当缓存文件所在磁盘剩余空间不足 20480MB 时，COSFS 将启动失败。
    - GooseFS-Lite 工具支持将存储桶挂载到本地，像使用本地文件系统一样直接操作腾讯云对象存储中的对象，相比于 COSFS 工具，GooseFS-Lite 可提供更高的大文件读写速度，不受本地磁盘的性能限制，建议容器内存大于2G时使用。配置包括 core_site、goosefs_lite。

      core_site 常用参数如下，更多参数可参见 Hadoop-COS 文档。

属性键	说明	默认值	必填项
fs.cosn.useHttps	配置是否使用 HTTPS 作为与 COS 后端的传输协议。	true	否
fs.cosn.upload.part.size	分块上传的每个 part size 的大小。由于 COS 的分块上传最多只能支持10000块，因此需要预估最大可能使用到的单文件大小。例如，part size 为8MB时，最大能够支持78GB的单文件上传。 part size 最大可以支持到2GB，即单文件最大可支持19TB。	8388608（8MB）	否
fs.cosn.upload_thread_pool	文件流式上传到 COS 时，并发上传的线程数目。	32	否
fs.cosn.read.ahead.block.size	预读块的大小。	1048576（1MB）	否
fs.cosn.read.ahead.queue.size	预读队列的长度。	6	否
fs.cosn.trsf.fs.ofs.tmp.cache.dir	元数据加速存储桶的临时文件目录。	无	元数据加速桶必填
fs.cosn.trsf.fs.ofs.user.appid	元数据加速存储桶的 Appid。	无	元数据加速桶必填
fs.cosn.trsf.fs.ofs.bucket.region	元数据加速存储桶所在的地域，如 ap-shanghai，ap-beijing。	无	元数据加速桶必填

goosefs_lite 常用参数如下：

属性键	说明	默认值	必填项
goosefs.fuse.list.entries.cache.enabled	是否开启客户端 List 缓存	true	否
goosefs.fuse.list.entries.cache.max.size	客户端 List 最大缓存的条目数，单位：条	100000	否
goosefs.fuse.list.entries.cache.max.expiration.time	客户端 List 缓存的有效时间，单位：ms	15000	否
goosefs.fuse.async.release.wait_time.max.ms	open 和 rename 操作的文件正在被写入时，等待写入操作完成的时间，单位：ms	5000	否
goosefs.fuse.umount.timeout	卸载文件系统时，等待未完成操作的时间，单位：ms	120000	否

> **说明：**
> 不同的挂载项请以空格进行间隔，更多挂载选项请参见 常用挂载选项文档 。

4. 单击**创建 PersistentVolume** 即可。

步骤3：创建 PVC 绑定 PV

> **注意：**
> 请勿绑定状态为 Bound 的 PV。

1. 在目标集群详情页，选择左侧菜单栏中的**存储 > PersistentVolumeClaim**，在 **PersistentVolumeClaim** 页面单击**新建**。
2. 在**新建 PersistentVolumeClaim** 页面，参考以下信息创建 PVC。如下图所示：

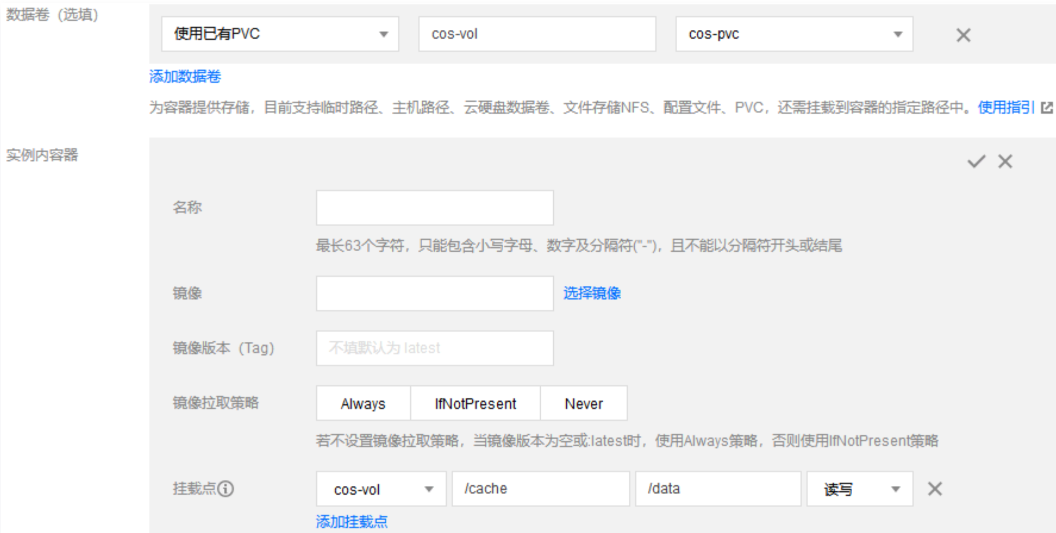
- **名称**：自定义，本文以 `cos-pvc` 为例。
- **命名空间**：选择为 `kube-system` 。
- **Provisioner**：选择**对象存储 COS**。
- **读写权限**：对象存储仅支持多机读写。
- **PersistentVolume**：选择在 步骤2 中已创建的 PV，本文以 `cos-pv` 为例。

3. 单击**创建 PersistentVolumeClaim** 即可。

步骤4：创建 Pod 使用的 PVC

> **说明：**
> 本步骤以创建工作负载 Deployment 为例。

1. 在目标集群详情页，选择左侧菜单栏中的**工作负载** > **Deployment**，在 Deployment 页面单击**新建**。
2. 在**新建 Deployment** 页面，参考 创建 Deployment 进行创建，并设置数据卷挂载。如下图所示：



- **数据卷（选填）**：
  - **挂载方式**：选择**使用已有 PVC**。
  - **数据卷名称**：自定义，本文以 `cos-vol` 为例。
  - **选择 PVC**：选择已在 步骤3 中创建的 PVC，本文以选择 `cos-pvc` 为例。
- **实例内容器**：单击**添加挂载点**，进行挂载点设置。
  - **数据卷**：选择为该步骤中所添加的数据卷 “cos-vol” 。
  - **目标路径**：填写目标路径，本文以 `/cache` 为例。
  - **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。例如， `./data` 或 `data` 。

3. 单击**创建 Deployment** 即可。

通过 YAML 文件使用对象存储

创建可以访问对象存储的 Secret

可通过 YAML 创建可以访问对象存储的 Secret，模板如下：

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: cos-secret
  # Replaced by your secret namespace.
  namespace: kube-system
data:
  # Replaced by your temporary secret file content. You can generate a temporary secret key with these
docs:
  # Note: The value must be encoded by base64.
  SecretId: VWVEJxRk5Fb0JGbDA4M...(base64 encode)
  SecretKey: Qa3p4ZTVCMFlQek...(base64 encode)
```

创建支持 COS-CSI 动态配置的 PV

可通过 YAML 创建 PV 以支持 COS-CSI 动态配置，模板如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cos-pv
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cosfs
    nodePublishSecretRef:
      name: cos-secret
      namespace: kube-system
    volumeAttributes:
      # Replaced by the url of your region.
      url: http://cos.ap-XXX.myqcloud.com
      # Replaced by the bucket name you want to use.
      bucket: XXX-1251707795
      # You can specify sub-directory of bucket in cosfs command in here.
      path: /costest
       # You can specify any other options used by the cosfs command in here.
    # additional_args: "-oallow_other"# Specify a unique volumeHandle like bucket name.(this value must
different from other pv's volumeHandle)
    volumeHandle: cos-pv
  persistentVolumeReclaimPolicy: Retain
  volumeMode: Filesystem
```

创建 PVC 绑定 PV

可通过 YAML 创建绑定上述 PV 的 PVC，模板如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cos-pvc
spec:
  accessModes:
```

```yaml
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  # You can specify the pv name manually or just let kubernetes to bind the pv and pvc.
  volumeMode: Filesystem
  volumeName: cos-pv
  # Currently cos only supports static provisioning, the StorageClass name should be empty.
  storageClassName: ""
```

创建 Pod 使用 PVC

可通过 YAML 创建 Pod，模板如下：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-cos
spec:
  containers:
  - name: pod-cos
    command: ["tail", "-f", "/etc/hosts"]
    image: "centos:latest"
    volumeMounts:
    - mountPath: /data
      name: cos
    resources:
      requests:
        memory: "128Mi"
        cpu: "0.1"
  volumes:
  - name: cos
    persistentVolumeClaim:
      # Replaced by your pvc name.
      claimName: cos-pvc
```

相关信息

更多关于如何使用对象存储的信息请参见 README_COSFS.md。

使用数据加速器 GooseFS

最近更新时间：2025-02-26 16:53:52

操作场景

腾讯云容器服务 TKE 支持通过创建 PersistentVolume（PV）和 PersistentVolumeClaim（PVC），并为工作负载挂载数据卷的方式使用腾讯云数据加速器 GooseFS。本文介绍如何在 TKE 集群中为工作负载挂载对象存储。

准备工作

1. 安装数据加速器扩展组件

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在**组件管理**页面中单击**新建**。
4. 在**新建组件管理**页面，勾选 **GooseFS（腾讯云数据加速器）**组件。
5. 单击**完成**。

2. 创建数据加速器

登录 数据加速器控制台 并创建一个 GooseFS 集群，操作详情请参见 创建 GooseFS 集群。创建完成后，在实例列表中进行查看。

操作步骤

通过控制台使用数据加速器

步骤1：创建支持 GooseFS-CSI 静态配置的 PV

> **注意：**
> 本步骤需使用数据加速器，若当前地域无可用 GooseFS 集群，则请参考 创建 GooseFS 集群 进行创建。

1. 在目标集群详情页面，选择左侧导航中的**存储**，在 **PersistentVolume** 页面单击**新建**。
2. 在**新建存储**页面，参考以下信息创建 PV。如下图所示：

主要参数信息如下：
- **来源设置**：选择**静态创建**。
- **名称**：自定义，本文以 `goosefs-pv` 为例。

- **Provisioner**：选择为**数据加速 GooseFS**。
- **读写权限**：数据加速器仅支持多机读写。

> **说明：**
> - 单机读写：当前仅支持云硬盘同时挂载到一台机器上，因此只能处理单机器的数据读写。
> - 多机读写：文件存储/对象存储/数据加速器支持同时挂载到多台机器，可以处理多机器的数据读写。

- **GooseFS 集群**：按需选择可用 GooseFS 即可。
- **GooseFS 命名空间**：填写已在 GooseFS 集群中创建的命名空间，本文以 goosefs-test 为例。

> **注意：**
> 填写的 GooseFS 命名空间需已存在，请前往数据加速器控制台提前新建。

- **命名空间子路径**：填写已在 GooseFS 集群中命名空间的子路径。
- **GooseFS FUSE**：

属性键	说明	默认值	必填项
-Xms4g	jvm 最小内存	4g	否
-Xmx8g	jvm 最大内存	8g	否
-XX:MaxDirectMemorySize	堆外最大内存	8g	否
-XX:+UseG1GC	jvm gc 算法	g1	否
-Dgoosefs.fuse.web.enabled	是否开启监控	false	否
-Dgoosefs.fuse.web.port	监控的端口	9213	否

> **注意：**
> - 不同的挂载项请以空格进行间隔。
> - 开启监控的情况 (goosefs.fuse.web.enabled=true)，需保证设置的端口 (goosefs.fuse.web.port) 唯一。

- **挂载选项**：GooseFS 工具支持将存储桶挂载到本地，挂载后可直接操作对象存储中的对象，此项用于设置相关限制条件。本例中挂载选项 allow_other 表示允许其他用户访问；direct_io 表示启用直接 I/O。更多参数可参见 FUSE 挂载参数列表 文档。

3. 单击**创建 PersistentVolume** 即可。

步骤2：创建 PVC 绑定 PV

> **注意：**
> 请勿绑定状态为 Bound 的 PV。

1. 在目标集群详情页，选择左侧导航中的**存储**，在 **PersistentVolumeClaim** 页面单击**新建**。
2. 在**新建存储**页面，参考以下信息创建 PVC。如下图所示：

- **名称**：自定义，本文以 `goosefs-pvc` 为例。
- **命名空间**：选择为 `kube-system` 。
- **Provisioner**：选择数据加速器 GooseFS。
- **读写权限**：数据加速器仅支持多机读写。
- **PersistentVolume**：选择在 步骤1 中已创建的 PV，本文以 `goosefs-pv` 为例。

3. 单击**创建 PersistentVolumeClaim** 即可。

步骤3：创建 Pod 使用的 PVC

> **说明：**
> 本步骤以创建工作负载 Deployment 为例。

1. 在目标集群详情页，选择左侧导航中的**工作负载**，在 **Deployment** 页面单击**新建**。
2. 在**新建 Deployment** 页面，参考 创建 Deployment 进行创建，并设置数据卷挂载。如下图所示：

- **数据卷（选填）**：
  - **挂载方式**：选择**使用已有 PVC**。
  - **数据卷名称**：自定义，本文以 `goosefs` 为例。
  - **选择 PVC**：选择已在 步骤2 中创建的 PVC，本文以选择 `goosefs-pvc` 为例。
- **实例内容器**：单击**添加挂载点**，进行挂载点设置。
  - **数据卷**：选择为该步骤中所添加的数据卷 “goosefs”。
  - **目标路径**：填写目标路径。
  - **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。

3. 单击**创建 Deployment** 即可。

通过 YAML 文件使用数据加速器

创建支持 GooseFS-CSI 动态配置的 PV

可通过 YAML 创建 PV 以支持 GooseFS-CSI 动态配置，模板如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: goosefs-pv
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.goosefs
    volumeAttributes:
      goosefsPath: /borgerli-1251707795
      javaOptions: -
Dgoosefs.master.embedded.journal.addresses=172.21.112.141:9202,172.21.112.133:9202,172.21.112.30:9202
        -Xms4g -Xmx8g -XX:MaxDirectMemorySize=8g -XX:+UseG1GC -Dgoosefs.fuse.web.enabled=true
        -Dgoosefs.fuse.web.port=9213
    volumeHandle: goosefs-pv
  mountOptions:
  - allow_other
  - direct_io
  persistentVolumeReclaimPolicy: Retain
  volumeMode: Filesystem
```

创建 PVC 绑定 PV

可通过 YAML 创建绑定上述 PV 的 PVC，模板如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: goosefs-pvc
  namespace: kube-system
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: ""
  volumeMode: Filesystem
  volumeName: goosefs-pv
status:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
```

创建 Pod 使用 PVC

可通过 YAML 创建 Pod，模板如下：

```
apiVersion: v1
kind: Pod
```

```yaml
metadata:
  name: pod-goosefs
spec:
  containers:
  - name: pod-goosefs
    command: ["tail", "-f", "/etc/hosts"]
    image: "centos:latest"
    volumeMounts:
    - mountPath: /data
      name: goosefs
    resources:
      requests:
        memory: "128Mi"
        cpu: "0.1"
  volumes:
  - name: goosefs
    persistentVolumeClaim:
      # Replaced by your pvc name.
      claimName: goosefs-pvc
```

使用建议

同一个 GooseFS 集群使用同一个 PVC

业务 Pod 部署到节点后，goosefs csi 会创建一个 fuse pod 负责挂载，会为每一个 volumeHandle 创建一个 fuse pod。fuse pod 使用的资源比较多，默认请求 2GB 内存，建议相同 GooseFS 集群的业务使用同一个 PVC。运行业务 Pod 的节点，只运行一个 fuse pod，最小化资源使用。

fuse pod 优先级

fuse pod 使用的资源比较多，默认请求 2GB 内存。存在以下两个问题：

- 在节点资源紧张时，无法调度；
- 运行后，节点资源紧张，有可能被驱逐，导致存量业务受影响。

以上两个问题的解决方案，可以为 fuse pod 加上 priorityClassName: system-node-critical。但同时会引起第三个问题：在调度时如果节点内存不足，有可能驱逐其他 Pod，影响业务。需要根据使用场景进行配置。

```yaml
apiVersion: v1
data:
  podtemplate: |
    kind: Pod
    apiVersion: v1
    metadata:
      name: goosefs-fuse
      labels:
        app: goosefs-fuse
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/metrics/prometheus/'
        prometheus.io/port: '9213'
    spec:
      priorityClassName: system-node-critical
      hostNetwork: false
      dnsPolicy: ClusterFirstWithHostNet
      securityContext:
        runAsUser: 0
        runAsGroup: 0
        fsGroup: 0
      containers:
        - name: goosefs-fuse
          image: ccr.ccs.tencentyun.com/tkeimages/csi-tencentcloud-goosefs-fuse:v1.0.2
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              memory: "16Gi"
            requests:
              memory: "2Gi"
          command: [ "/entrypoint.sh" ]
```

调整 fuse pod 资源需求

可以修改 configmap -nkube-system goosefs-csi-fuse-config，调整 podtemplate 的资源配置：

```yaml
apiVersion: v1
data:
  podtemplate: |
    kind: Pod
    apiVersion: v1
    metadata:
      name: goosefs-fuse
      labels:
        app: goosefs-fuse
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/metrics/prometheus/'
        prometheus.io/port: '9213'
    spec:
      priorityClassName: system-node-critical
      hostNetwork: false
      dnsPolicy: ClusterFirstWithHostNet
      securityContext:
        runAsUser: 0
        runAsGroup: 0
        fsGroup: 0
      containers:
        - name: goosefs-fuse
          image: ccr.ccs.tencentyun.com/tkeimages/csi-tencentcloud-goosefs-fuse:v1.0.2
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              memory: "16Gi"
            requests:
              memory: "2Gi"
          command: [ "/entrypoint.sh" ]
```

使用文件存储 CFS
文件存储使用说明

最近更新时间：2023-11-22 16:48:31

操作场景

腾讯云容器服务 TKE 支持通过创建 PV/PVC，并为工作负载挂载数据卷的方式使用腾讯云文件存储 CFS。本文介绍如何通过以下两种方式在集群中为工作负载挂载文件存储：

- 方式1：动态创建文件存储
- 方式2：使用已有的文件存储

准备工作

安装文件存储扩展组件

> **说明**
> 若您的集群已安装 CFS-CSI 的扩展组件，则请跳过此步骤。

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 CFS（**腾讯云文件存储**）。
5. 单击**完成**即可创建组件。

操作步骤

动态创建文件存储

当您需要动态创建文件存储时，可以按照以下步骤进行操作：

1. 创建文件存储类型的 StorageClass，定义需使用的文件存储模板。
2. 通过 StorageClass 创建 PVC，进一步定义所需的文件存储参数。
3. 创建工作负载数据卷时选择已创建的 PVC，并设置容器挂载点。
   详细操作步骤请参见 StorageClass 管理文件存储模板 。

使用已有的文件存储

当您需要使用已有文件存储时，可以按照以下步骤进行操作：

1. 通过已有的文件存储创建 PV。
2. 创建 PVC 时设置与上述创建的 PV 相同的 StorageClass 和容量。
3. 创建工作负载时，选择上述 PVC。
   详细操作步骤请参见 PV 和 PVC 管理文件存储 。

相关信息

更多关于如何使用文件存储的信息请参见 README_CFS.md 。

StorageClass 管理文件存储模板

最近更新时间：2025-01-25 17:16:32

操作场景

集群管理员可使用 StorageClass 为容器服务集群定义不同的存储类型。容器服务已默认提供块存储类型的 StorageClass，您可通过 StorageClass 配合 PersistentVolumeClaim 动态创建需要的存储资源。
本文介绍通过控制台、Kubectl 两种方式创建文件存储 CFS 类型的 StorageClass，自定义文件存储使用所需的模板。

准备工作

1. 安装文件存储扩展组件

若您的集群已安装 CFS-CSI 的扩展组件，请跳过此步骤。若未安装，详细步骤请参见 安装文件存储扩展组件 。

2. 创建子网

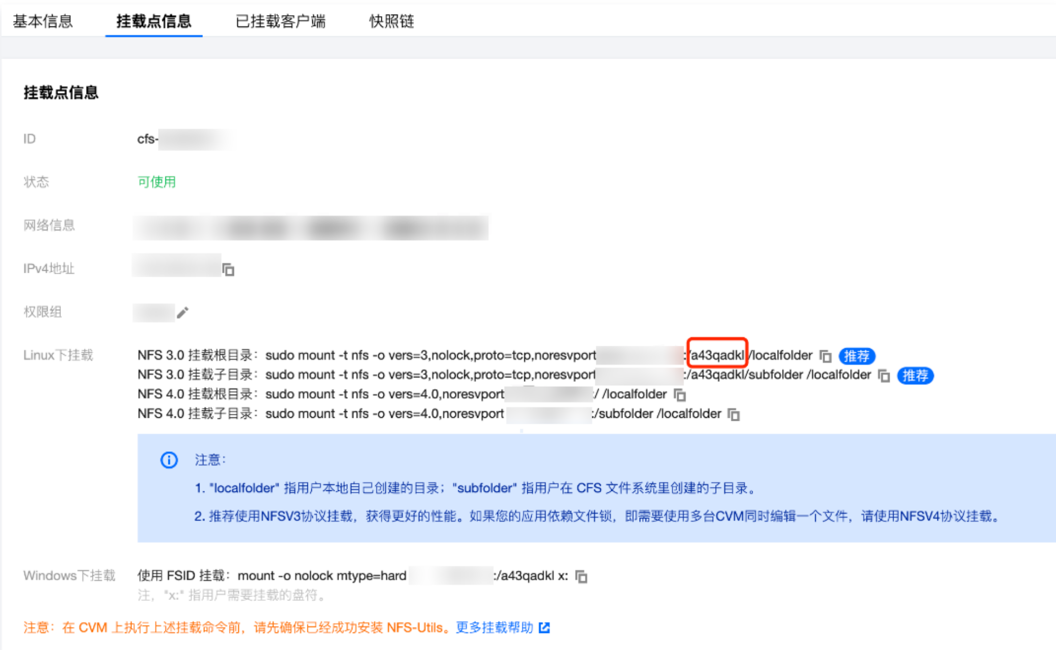
创建 StorageClass 过程中，需设置文件存储归属子网，为确保文件存储所处私有网络下每一个可用区均拥有合适子网，建议您提前进行子网创建。若无子网，详细步骤请参见 创建子网 。

3. 创建权限组并添加权限组规则

创建 StorageClass 过程中，需为文件系统配置权限组，为确保具备合适的权限组，建议您提前进行权限组创建。若无权限组，详细步骤请参见 创建权限组和添加权限组规则 。

4. 获取文件系统 FSID

1. 在 文件系统控制台，单击需获取 FSID 的文件系统 ID，进入该文件系统详情页。
2. 选择**挂载点信息**页签，从 “Linux 下挂载” 获取该文件系统的 FSID。如下图所示， `a43qadkl` 为该文件系统的 FSID。



> **说明：**
> 为了获取更好的稳定性，在通过 YAML 创建 PV 并使用 NFSV3 协议挂载时，需要指定待挂载文件系统对应的 FSID。

控制台操作指引

创建 StorageClass

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。

3. 选择左侧菜单栏中的**存储 > StorageClass**，在 StorageClass 页面单击**新建**。
4. 在**新建存储**页面，配置 StorageClass 参数。如下图所示：

配置项	描述
名称	填写 StorageClass 的名称。本文以 cfs-storageclass 为例。
地域	默认为集群所在地域。
Provisioner	Provisioner 支持选择云硬盘 CBS(CSI)、文件存储 CFS 或文件存储 CFS turbo。此处选择文件存储 CFS。
实例创建模式	提供**创建新实例**和**共享实例**两种模式。 • 创建新实例：挂载时每个 PVC 默认创建一个 CFS 实例。 • 共享实例：挂载时每个 PVC 将共享同一 CFS 实例的不同子目录，共享的 CFS 实例及子目录由系统自动创建。 **说明：** CFS-CSI 组件自 v1.0.1 版本开始支持共享存储实例功能，请及时升级组件版本，使用说明如下： • 共享实例类型的 StorageClass 回收策略限制为“保留”。 • 通过该 StorageClass 初次动态创建 PVC 时会默认创建一个 CFS 实例，并在该实例下创建子目录实现 PVC 之间的挂载隔离。 • 每个共享实例类型 StorageClass 创建的 CFS 实例不同，建议您妥善控制数量。
可用区	选择当前地域下支持使用文件存储的可用区。每个地域下不同可用区所适用的存储类型不完全一致，请参考 可用地域 进行选择。
CFS 归属子网	设置当前可用区下文件系统的所属子网范围。
存储类型	文件存储提供**标准存储**和**性能存储**两种类型的文件系统，每个地域下不同可用区所适用的存储类型不完全一致，请结合控制台实际情况进行选择。 • 标准存储：低成本、大容量，适用于成本敏感及大容量的业务。例如数据备份、文件共享、日志存储等场景。 • 性能存储：高吞吐、高 IOPS，适用于 IO 密集型工作负载。例如高性能计算、媒资渲染、机器学习、DevOps、办公 OA 等场景。

文件服务协议	默认为 NFS 协议，允许透明访问服务器上的文件和文件系统。
协议版本	推荐使用 NFS v3 协议挂载获得更好的性能。如果您的应用依赖文件锁（即需要使用多台 CVM 同时编辑一个文件）请使用 NFS v4 协议挂载。
权限组	为文件系统配置权限组，便于进一步管理与文件系统处于同一网络下的来访客户端的访问权限及读写权限。请根据实际需求选择合适的权限组，如不具备，请前往 权限组 页面进行创建。
回收策略	提供**删除**和**保留**两种回收策略，出于数据安全考虑，推荐使用**保留**回收策略。 • 删除：通过 PVC 动态创建的 PV，在 PVC 销毁时，与其绑定的 PV 和存储实例也会自动销毁。 • 保留：通过 PVC 动态创建的 PV，在 PVC 销毁时，与其绑定的 PV 和存储实例会被保留。
标签	选择 CFS 实例需要绑定的云标签。该标签将由 StorageClass 动态创建的 CFS 实例自动继承，StorageClass 创建后其绑定的标签参数不支持修改。如现有标签不符合您的要求，请前往 标签控制台 操作。

5. 单击**创建 StorageClass** 即可。

使用指定 StorageClass 创建 PVC

1. 在集群列表中，单击目标集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储 > PersistentVolumeClaim**，在 PersistentVolumeClaim 信息页面单击**新建**。
3. 在**新建 PersistentVolumeClaim** 页面，配置 PVC 关键参数。如下图所示：

配置项	描述
名称	填写 PersistentVolumeClaim 的名称。本文以 cfs-pvc 为例。
命名空间	命名空间用来划分集群资源。此处选择 default。
Provisioner	选择文件存储 CFS。
读写权限	文件存储仅支持多机读写。
StorageClass	按需指定 StorageClass。本文选择**指定 StorageClass**，以在 创建 StorageClass 步骤中创建的 cfs-storageclass 为例。 **说明：** • PVC 和 PV 会绑定在同一个 StorageClass 下。 • **不指定 StorageClass** 意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。
PersistVolume	按需指定 PersistentVolume。本文选择**不指定 PersistentVolume**。 **说明：**

	• 系统首先会筛选当前集群内是否存在符合绑定规则的 PV，若没有则根据 PVC 和所选 StorageClass 的参数动态创建 PV 与之绑定。 • 系统不允许在不指定 StorageClass 的情况下同时选择不指定 PersistVolume。 • 关于**不指定 PersistVolume** 的详细介绍，请参见 查看 PV 和 PVC 的绑定规则。

4. 单击**创建 PersistentVolumeClaim**。

创建 Workload 使用 PVC 数据卷

> **说明：**
> 该步骤以创建工作负载 Deployment 为例。

1. 在集群列表中，单击目标集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**工作负载 > Deployment**，在 Deployment 页面单击**新建**。
3. 在**新建工作负载**页面，参考 创建 Deployment 进行创建，并参考以下信息进行数据卷挂载。如下图所示：

   ○ **数据卷（选填）**：
      ○ **数据卷类型**：选择“使用已有 PVC”。
      ○ **数据卷名称**：自定义，本文以 `cfs-vol` 为例。
      ○ **选择 PVC**：选择在步骤 创建 PVC 中已创建的 “cfs-pvc”。
   ○ **实例内容器**：单击**添加挂载点**，进行挂载点设置。
      ○ **数据卷**：选择该步骤中已添加的数据卷“cfs-vol”。
      ○ **目标路径**：填写目标路径，本文以 `/cache` 为例。
      ○ **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。例如， `/data` 或 `/test.txt` 。
4. 单击**创建 Deployment**，完成创建。

> **注意：**
> 如使用 CFS 的 PVC 挂载模式，数据卷支持挂载到多台 Node 主机上。

Kubectl 操作指引

创建 StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cfs
parameters:
  pgroupid: xxxxxxxx
  storagetype: SD
```

```
  #subdir-share: "true"
  vpcid: vpc-xxxxxxxx
  subnetid: subnet-xxxxxxxx
  vers: "3"
  resourcetags: ""
  zone: ap-guangzhou-3
provisioner: com.tencent.cloud.csi.cfs  #共享实例：provisioner: com.tencent.cloud.csi.tcfs.xxx(storageClass
的name)
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

parameters 支持参数如下：

参数	是否可选	描述
zone	否	设置文件存储所在的地域。
pgroupid	否	设置文件存储所归属的权限组。
storagetype	否	默认为标准存储 SD，可取值及描述如下： • SD：标准型存储 • HP：性能存储
subdir-share	是	填写则代表 StorageClass 的实例创建模式为共享实例。
vpcid	是	创建的文件存储所在的私有网络 ID。
subnetid	是	创建的文件存储所在的子网 ID。
vers	是	插件连接文件系统时所使用的协议版本，动态生成的 PV 会继承该参数，目前支持的版本有 "3" 和 "4"。
resourcetags	是	文件系统云标签，生成的文件系统上会打上对应腾讯云标签，多个标签由英文逗号隔开，例如 "a:b,c:d"。

创建 PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cfs
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: cfs
  volumeMode: Filesystem
  volumeName: XXX  #动态创建无需填写，静态创建需要在该字段中指定 pv 实例 id
```

参数	是否可选	描述
spec.accessModes	否	cfs 存储支持多读多写
spec.resources.requests.storage	是	无实际意义，具体存储大小只与文件系统种类有关。

说明：

1. CFS 文件存储系统支持根据文件容量大小自动扩展文件系统存储容量，扩展过程不会中断请求和应用。默认创建的 CFS 实例容量大小为 10Gi，容量上限与产品类型相关，详情根据请参考 系统限制 。
2. 通过 PVC 动态创建的 PV 将自动继承 StorageClass 中设定的参数，该参数由存储插件自动生成。

PV 和 PVC 管理文件存储

最近更新时间：2025-07-01 14:42:22

操作场景

腾讯云容器服务支持通过创建 PV/PVC，并在创建工作负载添加数据卷时使用已有 PVC，实现通过 PV 和 PVC 管理文件系统。

> **注意：**
> 不同地域所支持的文件存储能力有一定差异，请按需选择。详情请参见 文件存储类型和性能规格 。

准备工作

安装文件存储扩展组件

> **说明：**
> 若您的集群已安装 CFS-CSI 的扩展组件，则请跳过此步骤。

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 CFS（**腾讯云文件存储**）。
5. 单击**完成**即可创建组件。

通过控制台创建 StorageClass

由于静态创建文件存储类型的 PV 时，需要绑定同类型可用 StorageClass，请参考 通过控制台创建 StorageClass 完成创建。

创建文件存储

1. 登录 文件存储控制台，在左侧导航栏单击**文件系统**，进入文件系统列表页面。
2. 单击**创建**。可根据实际需要进行灵活选择，文件类型可参见 存储类型及性能规格 。
3. 在详细设置中，参考以下信息填写：

字段	含义
文件系统名称	制定文件系统的自定义名称。本文以 cfs-test 为例。
地域	选择所需要创建 CFS 文件系统的地域，需确保与集群在同一地域。
可用区	选择所需要创建 CFS 文件系统的可用区。
文件协议	选择文件系统的协议类型，NFS 或 SMB 。其中，NFS 协议更适合于 Linux/Unix 客户端，CIFS/SMB 协议更适合于 Windows 客户端。Turbo 系列仅支持私有客户端使用，无法选择文件系统协议。
数据源	支持使用快照创建文件系统。
选择网络	指定文件系统所在的 VPC 网络，该网络内的实例可挂载访问文件系统。
权限组	每个文件系统必须绑定一个权限组，权限组规定了一组可来访白名单及读、写操作权限。
存储量	仅 Turbo 系列需填写存储量，因 Turbo 系列为独占集群，所以有最小的集群规模和扩容步长的限制。其中 Turbo 标准型的初始最小集群为40TiB，扩容步长为20TiB。Turbo 性能型的初始最小集群为20TiB，扩容步长为10TiB。
云联网	仅 Turbo 系列需填写云联网，需选择一个现有云联网或新创建一个。详细用途可参见 云联网介绍 。
网段	仅 Turbo 系列需填写网段，此项的目的是为 Turbo 相关组件预留IP地址段。需保证所选网段和希望与 Turbo 做信息通讯的云上其他实例的地址段不冲突，同时为保证 IP 地址的数量，掩码范围需在16-24位之间。示例：10.0.0.0/24
定期快照	为所购文件系统设置定期快照。
标签	• 若您已拥有标签，可以在此处为新建文件系统添加标签。

- 若还未拥有标签，请先到 标签控制台 创建所需要的标签，然后再为文件系统绑定标签。或者您也可以在文件系统创建完成后，再为文件系统添加标签。

4. 单击**立即创建**，等待创建成功即可。

获取文件系统子目录

1. 当文件系统创建完毕后，回到文件系统列表。
2. 单击已创建的文件系统名称，进入文件系统基本信息界面。
3. 单击**挂载点信息**，即可查看和获取 Linux 下挂载的该文件系统子目录路径 `/subfolder` 。如下图所示：

   - `localfolder` ：指用户本地自己创建的目录。
   - `subfolder` ：指用户在文件存储的文件系统里创建的子目录，则该文件系统子目录路径即为 `/subfolder` 。

获取文件系统 FSID

> **说明：**
> 为了获取更好的稳定性，在使用 NFSv3 协议挂载时，需要指定待挂载文件系统对应的 FSID。

1. 在 文件系统控制台，单击需获取 FSID 的文件系统 ID，进入该文件系统详情页。

2. 单击**挂载点信息**，从 “Linux 下挂载” 获取该文件系统的 FSID。如下图所示，`a43qadkl` 为该文件系统的 FSID。

操作步骤

静态创建 PV

> **说明：**
> 静态创建 PV 适用于已有存量的文件存储，并在集群内使用的场景。

1. 登录容器服务控制台，选择左侧导航栏中的 **集群**。
2. 在集群列表中，单击集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**存储** > **PersistentVolume**，在 PersistentVolume 页面单击**新建**。
4. 在**新建 PersistentVolume** 页面中，配置 PV 关键参数。
   - **来源设置**：选择**静态创建**。
   - **名称**：自定义，本文以 `cfs-pv` 为例。
   - **Provisioner**：选择**文件存储 CFS**。
   - **读写权限**：文件存储仅支持多机读写。
   - **StorageClass**：按需选择合适的 StorageClass。本文以选择在 通过控制台创建 StorageClass 步骤中创建的 `cfs-storageclass` 为例。

   > **说明：**
   > - PVC 和 PV 会绑定在同一个 StorageClass 下。
   > - 不指定 StorageClass 意味着该 PV 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。

   - **选择 CFS**：需确保文件存储与当前集群处于同一私有网络下，本文以选择在 创建文件存储 步骤中创建的 `cfs-test` 为例。
   - **CFS 子目录**：填写已在步骤 获取文件系统子目录 中获取的文件系统子路径，本文以 `/subfolder` 为例。
5. 单击**创建 PersistentVolume**，即可完成创建。

创建 PVC

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储** > **PersistentVolumeClaim**，在 PersistentVolumeClaim 页面单击**新建**。
3. 在**新建 PersistentVolumeClaim** 页面中，配置 PVC 关键参数。
   - **名称**：自定义，本文以 `cfs-pvc` 为例。

- 命名空间：选择 “default”。
- Provisioner：选择**文件存储 CFS**。
- **读写权限**：文件存储仅支持多机读写。
- **StorageClass**：按需选择合适的 StorageClass。本文以选择在 通过控制台创建 StorageClass 步骤中创建的 `cfs-storageclass` 为例。

> **说明：**
> - PVC 和 PV 会绑定在同一个 StorageClass 下。
> - 不指定意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。

- **PersistentVolume**：按需指定 PersistentVolume，本文选择以在 静态创建 PV 步骤中创建的 `cfs-pv` 为例。

> **说明：**
> - 只有与指定的 StorageClass 相同并且状态为 Available 和 Released 的 PV 为可选状态，如果当前集群内没有满足条件的 PV 可选，请选择“不指定”PersistentVolume。
> - 如果选择的 PV 状态为 Released，还需手动删除该 PV 对应 YAML 配置文件中的 `claimRef` 字段，该 PV 才能顺利与 PVC 绑定。详情请参见 查看 PV 和 PVC 的绑定规则 。

4. 选择**创建 PersistentVolumeClaim**，即可完成创建。

创建 Workload 使用 PVC 数据卷

> **说明：**
> 该步骤以创建工作负载 Deployment 为例。

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**工作负载** > **Deployment**，在 Deployment 页面单击**新建**。
3. 在新建 Deployment 页面中，配置工作负载参数，参数详情请参见 创建 Deployment 。
4. 参考以下信息进行数据卷挂载。如下图所示：

- **数据卷（选填）**：
  - **数据卷类型**：选择“使用已有 PVC”。
  - **数据卷名称**：自定义，本文以 `cfs-vol` 为例。
  - **选择 PVC**：选择在步骤 创建 PVC 中已创建的 “cfs-pvc”。
- **实例内容器**：单击**添加挂载点**，进行挂载点设置。
  - **数据卷**：选择该步骤中已添加的数据卷 “cfs-vol”。
  - **目标路径**：填写目标路径，本文以 `/cache` 为例。
  - **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。例如， `/data` 或 `/test.txt` 。

5. 单击**创建 Deployment**，完成创建。

> **注意：**
> 如使用 CFS 的 PVC 挂载模式，数据卷支持挂载到多台 Node 主机上。

kubectl 操作指引

创建 PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cfs
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cfs
    volumeAttributes:
      fsid: XXXXXX
      host: 192.168.XX.XX
      path: /
      vers: "3"
    volumeHandle: cfs
  persistentVolumeReclaimPolicy: Retain
  storageClassName: XXX
  volumeMode: Filesystem
```

参数	是否可选	描述
fsid	是	文件系统 fsid（非文件系统 id），可在文件系统挂载点信息中查看。
host	是	文件系统 ip 地址，可在文件系统挂载点信息中查看。
path	是	文件系统子目录，挂载后 workload 将无法访问到该子目录的上层目录。
vers	是	插件连接文件系统时所使用的协议版本，目前支持的版本有 "3" 和 "4"。

> **注意：**
> - 如果您在静态 PV 的 YAML 中指定协议版本为 `vers: "3"`，则还需要指定待挂载文件系统的 fsid 参数（获取方式请参考 获取文件系统 FSID），否则会存在挂载失败的情况；`vers: "4"` 则无需指定 fsid。
> - volumeHandle：一个字符串值，唯一标识来自 CSI 卷插件的 CreateVolume 调用返回的卷名。

使用云硬盘 CBS
云硬盘使用说明

最近更新时间：2023-05-17 15:40:34

操作场景

腾讯云容器服务 TKE 支持通过创建 PV/PVC，并为工作负载挂载数据卷的方式使用云硬盘 CBS。本文介绍如何通过以下两种方式在集群中为工作负载挂载云硬盘：

> **说明**
> 通过 PV 和 PVC 使用云硬盘 CBS 时，一个云硬盘仅支持创建一个 PV，同时只能被一个集群节点挂载。

- 方式1: 动态创建云硬盘
- 方式2: 使用已有的云硬盘

操作步骤

动态创建云硬盘

动态创建云硬盘时，通常包含以下几个步骤：

1. 创建云硬盘类型的 StorageClass，定义需使用的云硬盘模板。

> **说明**
> - 容器服务默认提供名称为 cbs 的 StorageClass。配置为：高性能云硬盘、随机选择可用区、按量计费。
> - 您可按需自行定义 StorageClass。

2. 通过 StorageClass 创建 PVC， 进一步定义所需的云硬盘参数。
3. 创建工作负载数据卷时选择已创建的 PVC，并设置容器挂载点。
   详细操作步骤请参见 StorageClass 管理云硬盘模板 。

使用已有的云硬盘

可通过以下步骤使用已有云硬盘：

1. 使用已有云硬盘创建 PV。
2. 创建 PVC 时，设置与已有 PV 相同的 StorageClass 和容量。
3. 创建工作负载时，选择 PVC。
   详细操作步骤请参见 PV 和 PVC 管理云硬盘 。

StorageClass 管理云硬盘模板

最近更新时间：2025-03-27 11:23:23

集群管理员可使用 StorageClass 为容器服务集群定义不同的存储类型。容器服务已默认提供块存储类型的 StorageClass，您可通过 StorageClass 配合 PersistentVolumeClaim 动态创建需要的存储资源。本文介绍通过控制台、Kubectl 两种方式创建云硬盘 CBS 类型的 StorageClass，自定义云硬盘使用所需的模板。

控制台操作指引

创建 StorageClass

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**存储**，在 StorageClass 页面单击**新建**。
4. 在**新建存储**页面中，配置 StorageClass 参数。如下图所示：

配置项	描述
名称	自定义，本文以 cbs-test 为例。
地域	默认为集群所在地域。
Provisioner	此处选择**云硬盘 CBS(CSI)**。

可用区	表示当前地域下支持使用云硬盘的可用区，请按需选择。
云盘类型	通常提供**高性能云硬盘**、**SSD 云硬盘**、**增强型 SSD 云硬盘**和**通用型 SSD 云硬盘**四种类型，不同可用区下提供情况有一定差异，详情请参见 云硬盘类型说明 并结合控制台提示进行选择。
额外性能	增强型 SSD 云硬盘可开启额外性能配置能力以获得更高性能。详情请参见 增强型 SSD 云硬盘额外性能 。 **注意：** • CBS 组件大于等于1.1.2支持。 • 额外性能最大吞吐量不大于650MB/s。
云盘计费模式	提供**按量计费**和**包年包月**两种计费模式，不同计费模式所支持的回收策略不同，请参考以下信息进行选择： • 按量计费：一种弹性计费模式，支持随时开通/销毁实例，按实例的实际使用量付费。支持删除和保留的回收策略。 • 包年包月：一种预付费模式，提前一次性支付一个月的存储费用，支持按月自动续费。仅支持保留的回收策略。 **注意：** • 如需购买包年包月云硬盘，则需前往 角色 页面，为 TKE_QCSRole 角色添加策略 `QcloudCVMFinanceAccess` 配置支付权限，否则可能会因支付权限问题导致创建基于包年包月 StorageClass 的 PVC 失败。 • 仅计费模式为包年包月的云硬盘可执行续费操作，自动续费功能默认按月续费。用户可前往所创建的PVC详情页，打开/关闭自动续费功能。更多计费信息参见 云硬盘计费问题 。
卷绑定模式	提供立即绑定和等待调度两种卷绑定模式，不同模式所支持的卷绑定策略不同，请参考以下信息进行选择： • 立即绑定：通过该 storageclass 创建的 PVC 将直接进行 PV 的绑定和分配。 • 等待调度：通过该 storageclass 创建的 PVC 将延迟与 PV 的绑定和分配，直到使用该 PVC 的 Pod 被创建。
标签	可按需给实例增加标签，用于云资源进行分类、搜索和聚合。 更多信息请参见 标签 。
加密	云硬盘加密能力能够最大限度保护您的数据安全，同时业务和应用程序无需做任何额外改动。 **注意：** CBS 组件大于等于1.1.10支持。 • 密钥材料来源：默认使用 KMS 为云产品创建的密钥。也可使用自定义密钥。 • 加密类型：支持一代加密、二代加密。 **注意：** S5/SA2及更早机型支持一代加密，此后机型支持二代加密。
定期备份	设置定期备份可有效保护数据安全，备份数据将产生额外费用，详情请见 快照概述 。 **说明：** 容器服务默认提供的 default-policy 备份策略的配置包括：执行备份的日期、执行备份的时间点和备份保留的时长。
在线扩容	选择是否启用在线扩容，详情见 在线扩容云硬盘 。
回收策略	云盘的回收策略，通常提供删除和保留两种回收策略，具体选择情况与所选计费模式相关。出于数据安全考虑，推荐使用保留回收策略。

5. 单击**创建 StorageClass**。

使用指定 StorageClass 创建 PVC

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧导航中的**存储**，在 PersistentVolumeClaim 页面单击**新建**。
3. 在**新建存储**页面中，配置 PVC 关键参数。如下图所示：

配置项	描述
名称	自定义，本文以 cbs-pvc 为例。
命名空间	命名空间用来划分集群资源。此处选择 default。
Provisioner	选择**云硬盘 CBS(CSI)**。
读写权限	云硬盘仅支持单机读写。 **说明：** • 单机读写：当前仅支持云硬盘同时挂载到一台机器上，因此只能处理单机器的数据读写。 • 多机读写：文件存储/对象存储支持同时挂载到多台机器，可以处理多机器的数据读写。
是否指定StorageClass	选择**指定 StorageClass**。本文选择已在 创建 StorageClass 步骤中创建的 cbs-test 为例。 **说明：** • PVC 和 PV 会绑定在同一个 StorageClass 下。 • **不指定 StorageClass** 意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。
StorageClass	选择上述步骤创建的 StorageClass。
是否指定 PersistVolume	本文以不指定 PersistentVolume 为例。 **说明：** • 系统首先会筛选当前集群内是否存在符合绑定规则的 PV，若没有则根据 PVC 和所选 StorageClass 的参数动态创建 PV 与之绑定。 • 系统不允许在不指定 StorageClass 的情况下同时选择不指定 PersistVolume。 • 关于**不指定 PersistVolume** 的详细介绍，请参见 查看 PV 和 PVC 的绑定规则。
云盘类型	根据所选的 StorageClass 展示所选的云盘类型为**高性能云硬盘**、**SSD 云硬盘**和**增强型 SSD 云硬盘**。
容量	在不指定 PersistentVolume 时，需提供期望的云硬盘容量（云硬盘大小必须为10的倍数。高性能云硬盘最小为10GB；SSD 和增强型 SSD 云硬盘最小为20GB）。
费用	根据上述参数计算创建对应云盘的所需费用，详情参考 计费模式 。

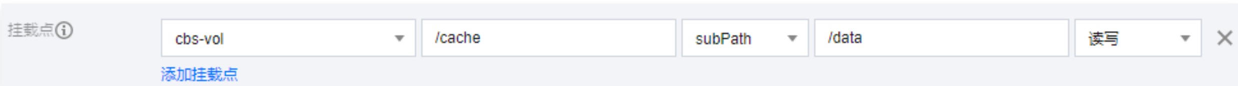
4. 单击**创建 PersistentVolumeClaim**。

创建 StatefulSet 挂载 PVC 类型数据卷

> **说明：**
> 该步骤以创建工作负载 StatefulSet 为例。

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧导航中的**工作负载**，在 StatefulSet 页面单击**新建**。
3. 在新建存储页面中，配置工作负载参数，参数详情请参见 创建 StatefulSet。根据实际需求，数据卷选择**使用已有 PVC**，并选择上述已创建的 PVC。如下图所示：

   - **数据卷（选填）**：
     - **挂载方式**：选择“使用已有 PVC”。
     - **数据卷名称**：自定义，本文以 `cbs-vol` 为例。
     - **选择 PVC**：选择已有 PVC，本文以选择在 使用指定 StorageClass 创建 PVC 步骤中创建的 `cbs-pvc` 为例。
4. 在实例内容器中添加挂载点。如下图所示：



   - **实例内容器**：单击**添加挂载点**，进行挂载点设置。
     - **数据卷**：选择该步骤中已添加的数据卷 “cbs-vol”。
     - **目标路径**：填写目标路径，本文以 `/cache` 为例。
     - **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。例如， `/data` 或 `/test.txt` 。
5. 单击**创建 StatefulSet**。

> **注意：**
> 如使用 CBS 的 PVC 挂载模式，则数据卷只能挂载到一台 Node 主机上。

Kubectl 操作指引

您可参考本文提供的示例模板，使用 Kubectl 进行 StorageClass 创建操作。

创建 StorageClass

以下 YAML 文件示例为集群内默认存在 name 为 cbs 的 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  # annotations:
  #   storageclass.beta.kubernetes.io/is-default-class: "true"
  #   如果有这一条，则会成为 default-class，创建 PVC 时不指定类型则自动使用此类型
    name: cloud-premium

# 安装了 CBS-CSI 组件的TKE集群请填写 provisioner 为 com.tencent.cloud.csi.cbs
# 未安装 CBS-CSI 组件请填写 provisioner 为 cloud.tencent.com/qcloud-cbs （该能力在1.20及以后版本废弃）
provisioner: com.tencent.cloud.csi.cbs

parameters:
    diskType: CLOUD_PREMIUM
    diskChargeType: POSTPAID_BY_HOUR
    aspid: asp-123
reclaimPolicy: Retain
```

```
volumeBindingMode: WaitForFirstConsumer
```

支持参数如下表：

参数	描述
diskType	包括 CLOUD_PREMIUM（高性能云硬盘）、CLOUD_SSD（SSD 云硬盘）、CLOUD_HSSD（增强型 SSD 云硬盘）和 CLOUD_BSSD（通用型 SSD 云硬盘）。
diskZone	用于指定可用区。如果指定，则云硬盘将创建到此可用区。如果不指定，则拉取所有 Node 的可用区信息，进行随机选取。腾讯云各地域标识符请参见 地域和可用区 。
diskChargeType	云硬盘的计费模式，默认设置为 POSTPAID_BY_HOUR 模式，即按量计费，支持 Retain 保留和 Delete 删除策略，Retain 仅在高于1.8的集群版本生效。还可设置为 PREPAID 模式，即包年包月，仅支持 Retain 保留策略。
volumeBindingMode	卷绑定模式，支持 Immediate（立即绑定）和 WaitForFirstConsumer（延迟调度）。
reclaimPolicy	回收策略，支持 Delete（删除）和 Retain（保留）。
aspid	指定快照 ID，创建云硬盘后自动绑定此快照策略，绑定失败不影响创建。

创建多实例 StatefulSet

使用云硬盘创建多实例 StatefulSet，YAML 文件示例如下：

> **说明：**
> 资源对象的 apiVersion 可能因为您集群的 Kubernetes 版本不同而不同，您可通过 `kubectl api-versions` 命令查看当前资源对象的 apiVersion。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:   # 自动创建pvc，进而自动创建pv
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: cloud-premium
      resources:
```

```
        requests:
          storage: 20Gi
```

PV 和 PVC 管理云硬盘

最近更新时间：2025-07-01 14:42:22

操作场景

腾讯云容器服务 TKE 支持通过创建 PV/PVC，并在创建工作负载添加数据卷时使用已有 PVC，实现通过 PV 和 PVC 管理云硬盘。本文介绍如何通过控制台、kubectl 两种方式实现 PV 和 PVC 管理云硬盘。

> **注意：**
> - 云硬盘不支持跨可用区挂载。若挂载云硬盘类型 PV 的 Pod 迁移至其他可用区，将会导致挂载失败。
> - 容器服务控制台不支持云硬盘扩容，可前往 云硬盘控制台 进行扩容操作。详情请参见 扩容云硬盘 。

操作步骤

控制台操作指引

通过控制台创建 StorageClass

由于静态创建云硬盘类型的 PV 时，需要绑定同类型可用 StorageClass，请参考 创建 StorageClass 完成创建。

静态创建 PV

> **说明：**
> 静态创建 PV 适用于已有存量云盘，并在集群内使用的场景。

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**存储** > **PersistentVolume**，在 PersistentVolume 页面单击**新建**。如下图所示：

4. 在**新建 PersistentVolume** 页面，参考以下信息进行创建。如下图所示：

主要参数信息如下：
- **来源设置**：选择**静态创建**。
- **名称**：自定义，本文以 `cbs-pv` 为例。
- **Provisioner**：选择**云硬盘CBS**。
- **读写权限**：云硬盘仅支持单机读写。
- **StorageClass**：按需选择合适的 StorageClass。本文以选择在 通过控制台创建 StorageClass 步骤中创建的 `cbs-test` 为例。

> **说明：**
> - PVC 和 PV 会绑定在同一个 StorageClass 下。
> - 不指定意味着该 PV 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。

- **云盘**：选择已经创建好的云硬盘。
- **文件系统**：默认为 ext4。

5. 单击**创建 PersistentVolume** 即可完成创建。

创建 PVC

1. 在集群详情页，选择左侧菜单栏中的**存储** > **PersistentVolumeClaim**，在 PersistentVolumeClaim 页面单击**新建**。如下图所示：

2. 在**新建 PersistentVolumeClaim** 页面，参考以下信息进行创建。如下图所示：

主要参数信息如下：
- **名称**：自定义，本文以 `cbs-pvc` 为例。
- **命名空间**：选择 “default”。
- **Provisioner**：选择**云硬盘CBS**。
- **读写权限**：云硬盘只支持单机读写。
- **StorageClass**：按需选择合适的 StorageClass。本文以选择在 通过控制台创建 StorageClass 步骤中创建的 `cbs-test` 为例。

> **说明：**
> - PVC 和 PV 会绑定在同一个 StorageClass 下。
> - 不指定意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。

- **PersistentVolume**：按需指定 PersistentVolume，本文选择以在 静态创建PV 步骤中创建的 `cbs-pv` 为例。

> **说明：**
> - 只有与指定的 StorageClass 相同并且状态为 Available 和 Released 的 PV 为可选状态，如果当前集群内没有满足条件的 PV 可选，请选择“不指定”PersistentVolume。
> - 如果选择的 PV 状态为 Released，还需手动删除该 PV 对应 YAML 配置文件中的 `claimRef` 字段，该 PV 才能顺利与 PVC 绑定。详情请参见 查看 PV 和 PVC 的绑定规则 。

3. 单击**创建PersistentVolumeClaim**，即可完成创建。

创建 Workload 使用 PVC 数据卷

> **说明：**
> 该步骤以创建工作负载 Deployment 为例。

1. 在集群详情页，选择左侧菜单栏中的**工作负载** > **Deployment**。

2. 单击**新建**，在**新建 Deployment** 页面，参考 创建 Deployment 进行创建，并参考以下信息进行数据卷挂载。如下图所示：

- **数据卷（选填）**：
  - **数据卷类型**：选择“使用已有PVC”。
  - **数据卷名称**：自定义，本文以 `cbs-vol` 为例。
  - **选择 PVC**：选择在步骤 创建 PVC 中已创建的 “cbs-pvc”。
- **实例内容器**：单击**添加挂载点**，进行挂载点设置。
  - **数据卷**：选择该步骤中已添加的数据卷 “cbs-vol”。
  - **目标路径**：填写目标路径，本文以 `/cache` 为例。
  - **挂载子路径**：仅挂载选中数据卷中的子路径或单一文件。例如， `/data` 或 `/test.txt` 。

3. 单击**创建 Deployment** 即可完成创建。

> **注意：**
> 如使用 CBS 的 PVC 挂载模式，则数据卷只能挂载到一台 Node 主机上。

kubectl 操作指引

您可通过以下 YAML 示例文件，使用 kubectl 进行创建操作。

（可选）创建 PV

可以通过已有云硬盘创建 PV，也可以直接 创建 PVC ，系统将自动创建对应的 PV。YAML 文件示例如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cbs-test
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cbs
    fsType: ext4
    readOnly: false
    volumeHandle: disk-xxx # 指定已有的CBS id
  storageClassName: cbs
```

创建 PVC

若未 创建 PV，则在创建 PVC 时，系统将自动创建对应的 PV。YAML 文件示例如下：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nginx-pv-claim
spec:
  storageClassName: cbs
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

- 云硬盘大小必须为10的倍数。
- 高性能云硬盘最小为10GB，SSD 和增强型 SSD 云硬盘最小为20GB，详情见 创建云硬盘 。

使用 PVC

可通过创建 Deployment 使用 PVC 数据卷。YAML 示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      qcloud-app: nginx-deployment
  template:
    metadata:
      labels:
        qcloud-app: nginx-deployment
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        volumeMounts:
        - mountPath: "/opt/"
          name: pvc-test
      volumes:
      - name: pvc-test
        persistentVolumeClaim:
          claimName: nginx-pv-claim # 已经创建好的 PVC
```


其他存储卷使用说明

最近更新时间：2024-08-28 16:51:01

简介

数据卷类型

数据卷类型	描述
使用临时路径	/
使用主机路径	将容器所在宿主机的文件目录挂载到容器的指定路径中（即对应 Kubernetes 的 HostPath）。您可以根据业务需求，不设置源路径（即对应 Kubernetes 的 EmptyDir）。如果不设置源路径，系统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 适用于容器的临时存储。
使用 NFS 盘	只需填写 NFS 路径，您可以使用腾讯云的 文件存储 CFS，也可使用自建的文件存储 NFS。使用 NFS 数据卷适用于多读多写的持久化存储，也适用于大数据分析、媒体处理、内容管理等场景。
使用已有 PersistentVolumeClaim	使用已有 PersistentVolumeClaim 声明工作负载的存储，自动分配或新建 PersistentVolume 挂载到对应的 Pod 下。主要适用于 StatefulSet 创建的有状态应用。
使用 ConfigMap	ConfigMap 以文件系统的形式挂载到 Pod 上，支持自定义 ConfigMap 条目挂载到特定的路径。更多详情请参见 ConfigMap 管理。
使用 Secret	Secret 以文件系统的形式挂载到 Pod 上，支持自定义 Secret 条目挂载到特定的路径。更多详情请参见 Secret 管理。

数据卷的注意事项

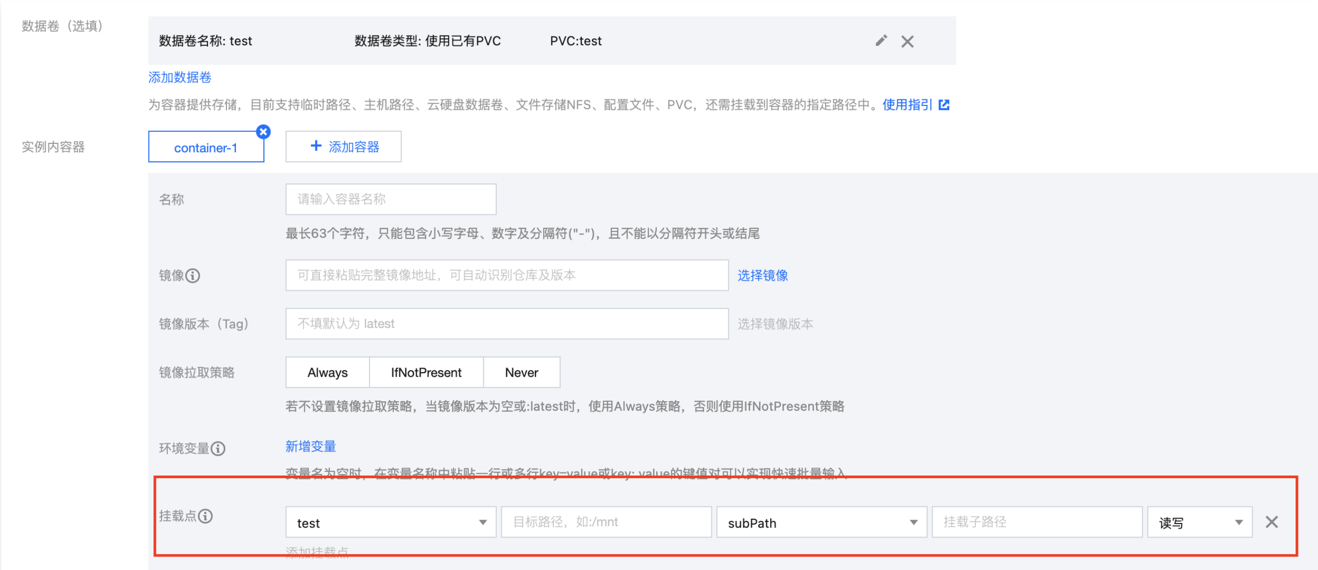
- 创建数据卷后，需在**实例内容器**模块设置容器的挂载点。
- 同一个服务下，数据卷的名称和容器设置的挂载点不能重复。
- 本地硬盘数据卷源路径为空时，系统将分配 `/var/lib/kubelet/pods/pod_name/volumes/kubernetes.io~empty-dir` 临时目录，且使用临时的数据卷生命周期与实例的生命周期保持一致。
- 数据卷挂载未设置权限，默认设置为读写权限。

Volume 控制台操作指引

创建工作负载挂载数据卷

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在**工作负载**下，任意选择 Workload 类型，进入对应的信息页面。
   例如，选择**工作负载** > **DaemonSet**，在 DaemonSet 信息页面单击**新建**。如下图所示：

4. 在**新建 DaemonSet** 页面，根据页面信息，设置工作负载名、命名空间等信息。
5. 在“数据卷（选填）”中，单击**添加数据卷**。根据实际需求，选择数据卷的存储方式，本文以**使用已有 PVC** 为例。
6. 在“实例内容器”的**挂载点**配置挂载点。如下图所示：
   在 步骤5 中选择**添加数据卷**后，才可进行挂载点配置。



7. 其余选项请按需设置，并单击**创建 DaemonSet** 即可完成创建。

各类数据卷挂载配置

该表展示了不同数据卷的使用细节，**当您在创建工作负载时，并选择“添加数据卷”后**，可参照以下内容进行数据卷的添加以及挂载点的设置：

数据卷			挂载点		
类型	名称	其他	目标路径	挂载子路径	读写权限
临时路径		/			
主机路径		设置主机路径。 • 主机路径：该路径不能为空，例如当该容器需要访问 Docker 时主机路径可设置为 `/var/lib/docker` 。 • 检查类型：TKE 为您提供 NoChecks、DirectoryOrCreate 等多种检查类型，请仔细查阅控制台上每种类型介绍，并根据实际需求进行选择。			
NFS 盘	自定义	NFS 路径：填写文件系统 CFS 或自建 NFS 地址。 • 如需创建文件系统，请参见 创建文件系统及挂载点 。 • NFS 路径示例如10.0.0.161:/。该路径可登录 文件系统控制台，单击目标文件系统 ID，在**挂载点信息**页签的 “Linux 下挂载目录”中获取。	请根据实际需求进行填写，示例如 `/cache` 。	仅挂载选中数据卷中的子路径或单一文件，示例如 `/data` 或 `/test.txt` 。	请根据实际需求进行选择。 • 只读：只允许读取该容器路径数据卷，数据修改只允许在宿主机上操作。 • 读写：允许读取以及将修改保存到该容器路径数据卷。
已有 PVC		请选择 PVC：根据实际需求进行选择。			
ConfigMap		• 选择 ConfigMap：根据实际需求进行。 • 选项：提供“全部”和“指定部分Key”两种选择。 • Items：当选择“指定部分Key” 选项时，可以通过添加 item 向特定路径挂载，如挂载点是 `/data/config` ，子路径是 `dev` ，最终会存储在 `/data/config/dev` 下。			
Secret					

Kubectl 操作 Volume 指引

仅提供示例文件，您可直接通过 Kubectl 进行创建操作。

Pod 挂载 Volume YAML 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

- **spec.volumes**：设置数据卷名称、类型、数据卷的参数。
  - **spec.volumes.emptyDir**：设置临时路径。
  - **spec.volumes.hostPath**：设置主机路径。
  - **spec.volumes.nfs**：设置 NFS 盘。
  - **spec.volumes.persistentVolumeClaim**：设置已有 PersistentVolumeClaim
- **spec.volumeClaimTemplates**：若使用该声明，将根据内容自动创建 PersistentVolumeClaim 和 PersistentVolume。
- **spec.containers.volumeMounts**：填写数据卷的挂载点。

PV 和 PVC 的绑定规则

最近更新时间：2024-11-05 14:34:52

PV 状态介绍

PV 状态	描述
Avaliable	创建好的 PV 在没有和 PVC 绑定的时候处于 Available 状态。
Bound	当一个 PVC 与 PV 绑定之后，PV 就会进入 Bound 的状态。
Released	一个回收策略为 Retain 的 PV，当其绑定的 PVC 被删除，该 PV 会由 Bound 状态转变为 Released 状态。 **注意：**Released 状态的 PV 需要手动删除 YAML 配置文件中的 claimRef 字段才能与 PVC 成功绑定。

PVC 状态介绍

PVC 状态	描述
Pending	没有满足条件的 PV 能与 PVC 绑定时，PVC 将处于 Pending 状态。
Bound	当一个 PV 与 PVC 绑定之后，PVC 会进入 Bound 的状态。

绑定规则

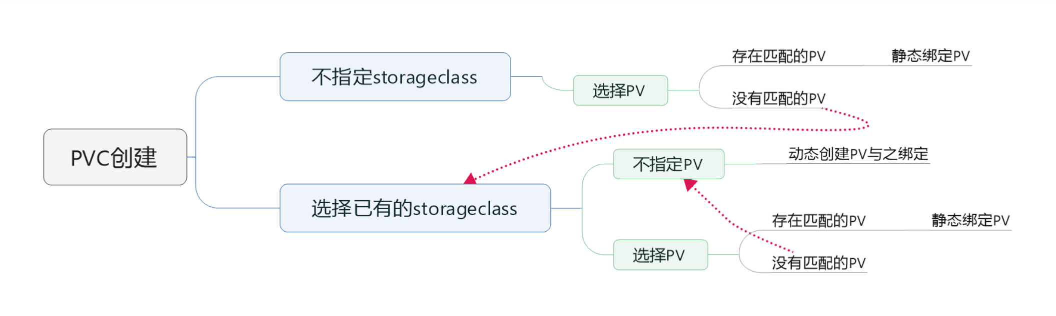
当 PVC 绑定 PV 时，需考虑以下参数来筛选当前集群内是否存在满足条件的 PV。

参数	描述
VolumeMode	主要定义 volume 是文件系统（FileSystem）类型还是块（Block）类型，PV 与 PVC 的 VolumeMode 标签必须相匹配。
Storageclass	PV 与 PVC 的 storageclass 类名必须相同（或同时为空）。
AccessMode	主要定义 volume 的访问模式，PV 与 PVC 的 AccessMode 必须相同。
Size	主要定义 volume 的存储容量，PVC 中声明的容量必须小于等于 PV，如果存在多个满足条件的 PV，则选择最小的 PV 与 PVC 绑定。

> **说明：**
> PVC 创建后，系统会根据上述参数筛选满足条件的 PV 进行绑定。如果当前集群内的 PV 资源不足，系统会动态创建一个满足绑定条件的 PV 与 PVC 进行绑定。

StorageClass 的选择和 PV/PVC 的绑定关系

容器服务 TKE 的平台操作中，StorageClass 的选择与 PV/PVC 之间的绑定关系见下图：



策略管理

最近更新时间：2025-06-11 10:01:22

简介

原生 Kubernetes 存在级联删除机制，删除一个资源时会自动删除与之相关的其他资源，例如删除 Namespace 时会自动删除 Namespace 下所有的 Pod、Service、ConfigMap 等关联资源，可能导致业务故障。
K8S 自身的脆弱性会导致在生产环境存在稳定性和安全的风险，例如用户配置的镜像拉取来源没有限制，可能导致镜像被篡改；缺少容器级别的安全隔离策略，可能导致容器发生越权等行为。
容器服务（Tencent Kubernetes Engine，TKE）提供“策略管理”能力，通过系统预置策略或者用户自主开启策略的方式，防止误删除引起业务故障，支持通过对计算资源、网络资源进行策略的加固，提升 TKE 集群的稳定性和安全性。

策略说明

策略分类

- **删除防护**：包含集群删除保护和集群内资源删除保护。在删除 TKE 集群内的各类资源时，如果资源正在使用中，或者存在和其他资源之间的引用关系，则不允许删除。
- **策略管控**：用于约束和规范 K8S 集群内的资源配置。例如：要求 Pod 具有 Readiness 或 Liveness Probe；限制容器镜像来源必须在指定的列表内等。
- **安全加固**：包含 PSP（PodSecurityPolicy）相关策略。TKE 提供基于 gatekeeper 的安全防护能力。例如，禁止创建特权容器，约束 PodSecurityPolicy 中的 hostPath、hostPID 和 hostIPC 字段等。

支持边界

支持1.16及以上 K8S 版本的 TKE 标准集群和 TKE Serverless 集群，暂不支持注册集群和边缘集群。

策略类型

- **基线策略**：TKE 内置的策略，包含保护集群基础设施资源不被误删除等策略。
- **优选策略**：TKE 最佳实践形成的标准和规范，对用户的各项配置进行校验，使得用户请求符合约束。用户可根据实际情况开启策略，并设置策略运行模式。
- **可选策略**：官方 OPA gatekeeper 策略库支持的策略，包含替代 PSP 能力的策略和通用策略。用户可以根据实际情况，自行创建策略实例，具体创建流程请参见 新建策略实例。

策略库

> **说明：**
> 为最大程度防范集群资源误删现象，TKE 策略管理系统计划于2025年6月5日后新建集群针对“存在 pod 的命名空间不允许删除”、“存在 cr 的 crd 不允许删除”、“非封锁状态的 Node 不允许删除”、“CoreDNS 组件删除保护”、“PV 处于绑定状态则不允许删除”开启默认 deny 策略，您可前往**集群详情-策略管理**模块进行查看和修改。

基线策略

策略分类	策略名称	策略描述	拦截对象	运行模式
删除保护	存在节点的集群不允许删除	集群中存在任意节点（普通节点、原生节点、注册节点），需先下线节点后方可删除。	集群	默认 deny

优选策略

策略分类	策略名称	策略描述	拦截对象	运行模式
删除保护	存在 pod 的命名空间不允许删除	命名空间内如果存在 pod，需先清除 pod 后方可删除 namespace。	Namespace	默认 dryrun
删除保护	存在 cr 的 crd 不允许删除	crd 定义的 apiversion 下如果有创建 cr 资源，则清空 cr 后方可删除 crd。	CRD	默认 dryrun
删除保护	非封锁状态的 Node 不允许删除	ready 状态的节点处于非封锁状态时不允许直接删除。	Node	默认不创建策略实例

删除保护	CoreDNS 组件删除保护	禁止删除 CoreDNS 组件的 Service、ConfigMap 和 Deployment。	Deployment、Service、ConfigMap	默认不创建策略实例
删除保护	资源删除保护	存在指定 Label 的资源（Service、Ingress、Deployment、StatefulSet）不允许被删除。	Service、Ingress、Deployment、StatefulSet	默认不创建策略实例
删除保护	PV 处于绑定状态则不允许删除	PersistentVolume 如果处于 Bound 状态，则不允许被删除。	PV	默认不创建策略实例
策略管控	禁止挂载指定的 volume 类型	将可挂载的 volume 类型限制为用户指定的类型。	Pod	默认不创建策略实例
策略管控	禁止镜像拉取策略使用 Always	禁止容器使用 Always 镜像拉取策略，减少对镜像仓库的访问。	Pod	默认不创建策略实例
策略管控	容器镜像来源限制	只允许从指定的镜像仓库拉取镜像。	Pod	默认不创建策略实例
策略管控	禁止未知的 DaemonSet 部署	只允许部署指定的 DaemonSet。	DaemonSet	默认不创建策略实例
策略管控	工作负载镜像版本升级策略管控	限制 Deployment 和 DaemonSet 只能在配置的镜像列表中升级。	Deployment、DaemonSet	默认不创建策略实例
策略管控	ServiceAccount 权限管控	禁止 ServiceAccount 绑定较大权限的 Role 和 ClusterRole，提升集群安全性。	ServiceAccount	默认不创建策略实例
策略管控	不允许 Service 为 ClusterIP 类型	禁止创建 ClusterIP 类型的 Service 或将 Service 由其他类型更新为 ClusterIP 类型。	Service	默认不创建策略实例
策略管控	禁止公网访问	禁止通过创建公网类型的 Service 或 Ingress 的方式将后端服务暴露到公网。	Service、Ingress	默认不创建策略实例
策略管控	弹性网卡资源配置限制	限制跨租户弹性网卡必须配置 Request 资源。	Pod	默认不创建策略实例

可选策略

策略分类	策略名称	策略描述	拦截对象
策略管控	tkeblockvolumemountpath	禁止容器挂载指定的目录。	pods
策略管控	k8sallowedrepos	容器镜像必须以指定字符串列表中的字符串开头。	pods
策略管控	k8spspautomountserviceaccounttokenpod	约束容器不能设置 automountServiceAccountToken 为 true。	pods
策略管控	k8sblockendpointeditdefaultrole	默认情况下，许多 Kubernetes 都预定义了一个名为 system:aggregate-to-edit 的 ClusterRole，k8sblockendpointeditdefaultrole 策略定义禁止该 ClusterRole 对 Endpoints 进行 create、patch 和 update 操作。	clusterroles
策略管控	k8sblockloadbalancer	不允许 Service 为 LoadBalancer 类型。	services
策略管控	k8sblocknodeport	不允许 Service 为 NodePort 类型。	services
策略管控	k8sblockwildcardingress	禁止 ingress 配置空白或通配符类型的 hostname。	ingresses
策略管控	k8scontainerlimits	限制容器必须设置 CPU 和内存 Limit，并且小于设定的最大值。	pods

策略管控	k8scontainerrequests	限制 CPU 和内存的 Request 必须设置且小于配置的最大值。	pods
策略管控	k8scontainerratios	限制 CPU 和内存的 Request 与 Limit 的最大比率。	pods
策略管控	k8srequiredresources	必须配置内存的 Limit，CPU 和内存的 Request。	pods
策略管控	k8sdisallowanonymous	不允许将白名单以外的 ClusterRole 和 Role 关联到 system:anonymous User 和 system:unauthenticated Group。	rolebindings clusterrolebindings
策略管控	k8sdisallowedtags	约束容器镜像 tag。	pods
策略管控	k8sexternalips	限制服务 externalIP 仅为允许的 IP 地址列表。	services
策略管控	k8simagedigests	容器镜像必须包含 digest。	pods
策略管控	noupdateserviceaccount	拒绝白名单外的资源更新 ServiceAccount。	replicationcontrollers replicasets deployments statefulsets daemonsets cronjobs
策略管控	k8sreplicalimits	要求具有 “spec.replicas” 字段的对象（Deployments、ReplicaSets等）在定义的范围内。	deployments
策略管控	k8srequiredannotations	要求资源包含指定的 annotations，其值与提供的正则表达式匹配。	services
策略管控	k8srequiredlabels	要求资源包含指定的标签，其值与提供的正则表达式匹配。	namespaces
策略管控	k8srequiredprobes	要求 Pod 具有 Readiness 或 Liveness Probe。	pods
安全加固	k8spspallowprivilegeescalationcontainer	约束 PodSecurityPolicy 中的 “allowPrivilegeEscalation” 字段为 false。	pods
安全加固	k8spspapparmor	约束 AppArmor 字段列表。	pods
安全加固	k8spspcapabilities	限制 PodSecurityPolicy 中的 “allowedCapabilities” 和 “requiredDropCapabilities” 字段。	pods
安全加固	k8spspflexvolumes	约束 PodSecurityPolicy 中的 allowedFlexVolumes 字段类型。	pods
安全加固	k8spspforbiddensysctls	约束 PodSecurityPolicy 中的 “sysctls” 字段不能使用的 name。	pods
安全加固	k8spspfsgroup	控制 PodSecurityPolicy 中的 “fsGroup” 字段在限制范围内。	pods
安全加固	k8spsphostfilesystem	约束 PodSecurityPolicy 中的 “hostPath” 字段的参数。	pods
安全加固	k8spsphostnamespace	限制 PodSecurityPolicy 中的 “hostPID” 和 “hostIPC” 字段。	pods
安全加固	k8spsphostnetworkingports	约束 PodSecurityPolicy 中的 “hostNetwork” 和 “hostPorts” 字段。	pods
安全加固	k8spspprivilegedcontainer	禁止 PodSecurityPolicy 中的 “privileged” 字段为 true。	pods
安全加固	k8spspprocmount	约束 PodSecurityPolicy 中的 “allowedProcMountTypes” 字段。	pods
安全加固	k8spspreadonlyrootfilesystem	约束 PodSecurityPolicy 中的 “readOnlyRootFilesystem”	pods

		字段。	
安全加固	k8spspseccomp	约束 PodSecurityPolicy 上的 “seccomp.security.alpha.kubernetes.io/allowedProfileNames” 注解。	pods
安全加固	k8spspselinuxv2	约束 Pod 定义 SELinux 配置的允许列表。	pods
安全加固	k8spspallowedusers	约束 PodSecurityPolicy 中的runAsUser、runAsGroup、supplementalGroups 和 fsGroup 字段。	pods
安全加固	k8spspvolumetypes	约束 PodSecurityPolicy 中的 “volumes” 字段类型。	pods

操作说明

开启/关闭策略

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面，选择目标集群 ID，进入集群的基本信息页面。
3. 在左侧导航中选择**策略管理**，进入策略管理页面选择策略，单击**开启/关闭**。关闭策略需要二次确认，开启则不需要。如下图所示：

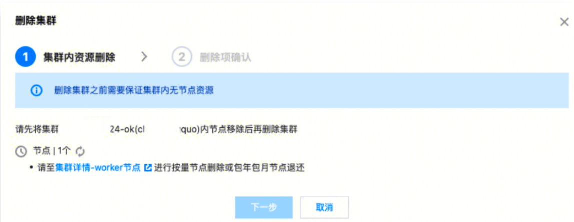
验证策略效果

以集群删除策略为例，创建 TKE 标准集群，验证集群在存在节点情况下删除请求是否会被拦截。

1. 创建有节点的 TKE 标准集群，详细步骤请参见 创建集群 。
2. 发起删除集群请求。

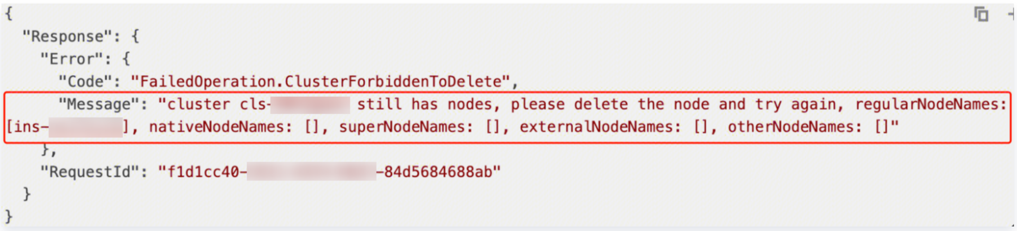
通过控制台删除

1. 删除集群，详细步骤请参见 删除集群 。
2. 窗口提示需要先清空节点后，方可继续删除集群。如下图所示：



调用云 API 删除

1. 调用云 API 删除，调用方式请参见 API 文档 删除集群 。
2. 删除集群接口调用失败，错误信息返回中包含集群中存在的节点清单。如下图所示：



```
{
  "Response": {
    "Error": {
      "Code": "FailedOperation.ClusterForbiddenToDelete",
      "Message": "cluster cls-[redacted] still has nodes, please delete the node and try again, regularNodeNames: [ins-[redacted]], nativeNodeNames: [], superNodeNames: [], externalNodeNames: [], otherNodeNames: []"
    },
    "RequestId": "f1d1cc40-[redacted]-84d5684688ab"
  }
}
```

3. 在**策略管理**页面，单击关联事件的数字，查看拦截事件信息。如下图所示：

拦截时间	资源类型	资源名称/ID	事件信息
2024-01-09 16:23:21	集群	mo[redacted].24-ok cls-[redacted]quo	adminsion weebhook cls-[redacted] delete blocked

共 1 条　20 条 / 页　1　/ 1 页

新建策略实例

以禁止创建特权容器为例，演示如何新建策略实例。

- 策略名称：k8spspprivilegedcontainer
- 策略类型：K8sPSPPrivilegedContainer
- 策略描述：禁止 Pod securityContext 中的 privileged 字段为 true。
- 生效资源类型：Pod

需要修改的参数如下：

- namespaces 可选参数：表示策略作用生效的命名空间。
  - 不填该字段或者字段取值为空时，表示所有命名空间都生效。
  - 支持前缀匹配，例如 namespaces: ["kube-*"] 匹配 "kube-system" 和 "kube-public"。
- excludedNamespaces 可选参数：表示策略豁免生效的命名空间，在该列表中的命名空间不会生效此条策略。
  - 不填该字段或者字段取值为空时，表示没有豁免的命名空间。
  - 支持前缀匹配，例如 excludedNamespaces: ["kube-*"] 匹配 "kube-system" 和 "kube-public"。
- exemptInitContainers 自定义参数：布尔值。含义：是否允许 initContainer 使用特权容器。
  - 部分业务的 initContainer 以特权容器的方式运行，执行类似 iptables 规则下发等操作。
  - 创建策略实例时，默认会允许 initContainer 使用特权容器。

策略实例 YAML：

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivilegedContainer
metadata:
  name: psp-privileged-container
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces: []
    excludedNamespaces: ["kube-system"]
  parameters:
    exemptInitContainers: true
```

测试策略是否生效的 Pod YAML 如下，直接 apply YAML，会被策略拦截，则表明策略已生效。

```yaml
apiVersion: v1
kind: Pod
metadata:
```

```
  name: privileged-pod
spec:
  containers:
  - name: privileged-container
    image: nginx
    securityContext:
      privileged: true
  initContainers:
  - name: privileged-init-container
    image: busybox
    command: ['sh', '-c', 'echo Hello, Kubernetes!']
    securityContext:
      privileged: true
```

创建一个特权容器的 Pod，预期输出：

```
Error from server (Forbidden): error when creating "pod.yaml": admission webhook
"validation.gatekeeper.sh" denied the request: [psp-privileged-container] Privileged container is not
allowed: privileged-container, securityContext: {"privileged": true}, Pod name: privileged-pod
```

应用与组件功能管理说明

最近更新时间：2023-11-20 18:02:54

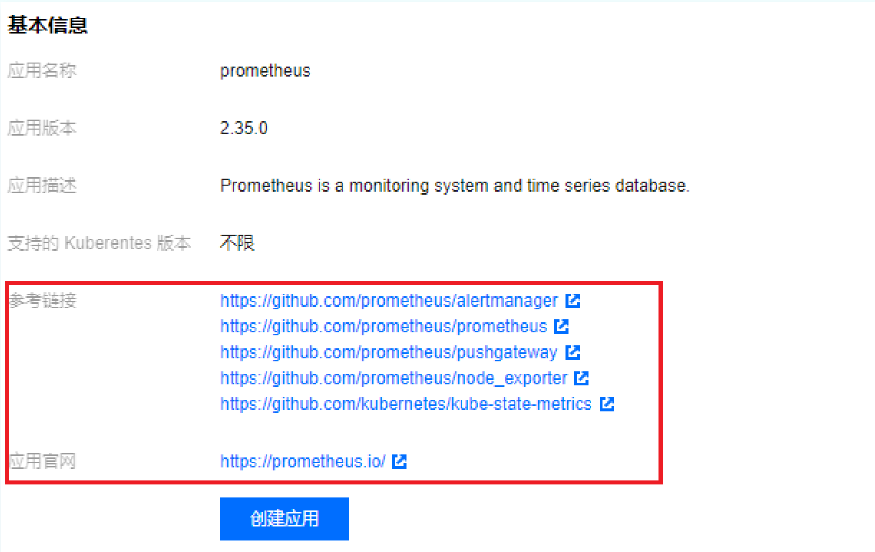
组件管理说明

腾讯云容器服务 TKE 提供多种组件，以丰富集群功能，增强集群性能，提升整体稳定性。腾讯云提供三种不同类型的组件：

组件类型	组件说明	相关文档
系统组件	用户使用集群默认的必备核心组件：例如 CBS-CSI，IPAMD，监控，日志等，若这些组件异常将可能导致集群故障。	-
增强组件	增强组件即扩展组件，是腾讯云容器服务 TKE 提供的扩展功能包，您可以根据业务诉求选择部署所需的扩展组件。	扩展组件
应用市场	应用市场是腾讯云容器服务 TKE 集成的 Helm 3.0 相关功能，为您提供创建 Helm Chart、容器镜像、软件服务等各种产品和服务的能力。	应用市场

注意

- **免责声明**：针对应用市场中的应用，腾讯云会保障用户在应用支持的集群类型和 Kubernetes 版本中正常安装应用。除此之外的部分（如运行过程中遇到的应用问题；因为自定义配置的修改导致应用异常；应用不支持指定的集群类型和 Kubernetes 版本）由用户自行负责。
- 腾讯云售后团队向您提供的应用市场里的应用只针对有经验的系统管理员或其他相关 IT 人员，同时腾讯云不提供这些应用的调试及建议实施的 SLA 保障。您在使用应用中如遇到任何问题，请到应用详情里的参考链接和应用官网反馈。如下图所示：



- 腾讯云容器服务（Tencent Kubernetes Engine，TKE）提供的服务等级协议请参见 腾讯云容器服务服务等级协议。

三种不同类型组件管理方式如下：

组件类型	包含范围	服务支持力度	升级方式
系统组件（默认安装，无法删除）	CBS-CSI，IPAMD，监控，日志等	TKE 会优先保障系统组件稳定性，会在后台及时更新和修复涉及到安全和兼容的问题。	某些拥有特殊功能的版本，TKE 不会后台自动更新，您可以根据实际需要自行更新，详情见 组件版本维护说明。
增强组件（用户自定义安装）	完整列表请 查看	TKE 会保障增强组件稳定性，会在后台及时更新和发布修复涉及到安全和兼容问题。每个增强组件有支持的版本列表，若您没有及时更新，组件可能会失效。	某些拥有特殊功能的版本，TKE 不会后台自动更新，您可以根据实际需要自行更新，详情见 组件版本维护说明。
应用市场（用户自定义安装）	完整列表请 查看	TKE 仅保障应用在支持的集群类型和 Kubernetes 版本里安装部署。	提供应用更新升级的方式，会不定期推送应用的新 Chart，操作详情见 更新应用。

功能管理说明

针对部分功能，TKE 会在功能的文档和控制台上标识：“预览版”，表示该功能为抢先体验版，不在 腾讯云容器服务服务等级协议 保障范围内。

组件管理

扩展组件概述

最近更新时间：2025-03-06 16:50:32

扩展组件是腾讯云容器服务 TKE 提供的扩展功能包，您可以根据业务诉求选择部署所需的扩展组件。扩展组件可帮助您管理集群的 Kubernetes 组件，包括组件部署、升级、更新配置和卸载等。

扩展组件类型

扩展组件分为基础组件和增强组件两种类型。

基础组件

基础组件是 TKE 功能依赖的软件包。例如，负载均衡组件 Service-controller、CLB-ingress-controller 及容器网络插件 tke-cni-agent 等。

> **说明：**
> - 基础组件的升级、配置管理将由 TKE 统一进行管理维护，不建议您修改基础组件。
> - 基础组件的更新发布动态将通过邮件、短信等形式进行通知。

增强组件

增强组件是 TKE 提供的非必需部署的组件，您可以通过部署增强组件来使用 TKE 支持的增强功能，增强组件类型如下表所示：

组件名称	使用场景	组件介绍
OOMGuard 内存溢出守护	监控	该组件在用户态降低了由于 cgroup 内存回收失败而产生的各种内核故障的发生几率。
NodeProblemDetector Plus 节点异常检测 Plus	监控	该组件可以实时检测节点上的各种异常情况，并将检测结果报告给 kube-apiserver。
NodeLocalDNSCache 本地 DNS 缓存组件	DNS	该组件通过在集群节点上作为 DaemonSet 运行 DNS 缓存代理来提高集群 DNS 性能。
DNSAutoscaler DNS 水平伸缩组件	DNS	该组件通过 deployment 获取集群的节点数和核数，并可以根据预设的伸缩策略，自动水平伸缩 DNS 的副本数。
COS-CSI 腾讯云对象存储	存储	该组件实现了 CSI 接口，可帮助容器集群使用腾讯云对象存储。
CFS-CSI 腾讯云文件存储	存储	该组件实现了 CSI 接口，可帮助容器集群使用腾讯云文件存储。
CBS-CSI 腾讯云硬盘存储	存储	该组件实现了 CSI 接口，支持 TKE 集群通过控制台快捷选择存储类型，并创建对应块存储云硬盘类型的 PV 和 PVC。
TCR 容器镜像服务插件	镜像	该组件自动为集群配置指定 TCR 实例的域名内网解析及集群专属访问凭证，可用于内网，免密拉取容器镜像。
Cerberus 镜像签名验证组件	镜像	该组件用于对 TCR 仓库下的容器镜像进行签名验证，确保只部署经过可信授权方签名的容器镜像，从而降低运行意外或恶意代码的风险。
Dynamic Scheduler 动态调度组件	调度	Dynamic Scheduler 是容器服务 TKE 基于 Kubernetes 原生 Kube-scheduler Extender 机制实现的动态调度器插件，可基于 Node 真实负载进行预选和优选。安装该组件后可以有效避免原生调度器基于 request 和 limit 调度机制带来的节点负载不均问题。
Descheduler 重调度组件	调度	在 TKE 集群中安装该插件后，该插件会和 Kube-scheduler 协同生效，实时监控集群中高负载节点并驱逐低优先级 Pod。建议您搭配 TKE Dynamic Scheduler（动态调度器扩展组件）一起使用，多维度保障集群负载均衡。
NetworkPolicy Controller 网络策略控制器组件	其他	Network Policy 是 Kubernetes 提供的一种资源，本组件提供了针对该资源的 Controller 实现。

Nginx-Ingress 社区 Ingress 组件	其他	Nginx 可以用作反向代理、负载平衡器和 HTTP 缓存。Nginx-ingress 组件是使用 Nginx 作为反向代理和负载平衡器的 Kubernetes 的 Ingress 控制器。
HPC 定时修改副本数	其他	HPC（HorizontalPodCronscaler）是一种可以对 K8S workload 副本数进行定时修改的自研组件，配合 HPC CRD 使用，最小支持秒级的定时任务。
QoSAgent 说明 QOS 保障组件	其他	QoS Agent 是腾讯云基于服务质量增强的扩展组件。提供丰富的能力，在提升集群资源利用率的同时，提供稳定性质量保障。
CraneScheduler 说明 原生节点专用调度器	其他	原生节点专用调度器是容器服务 TKE 基于 Kubernetes 原生 Kube-scheduler Extender 机制实现的调度器插件，可以虚拟放大节点的容量，用来解决节点资源都被占用，但本身利用率很低的问题。
Craned 说明 Request 推荐组件	其他	Request 智能推荐可以为 Kubernetes 的 Workload 推荐容器级别资源的 Request/Limit 数值，减少资源浪费。
PodIdentityWebhook 说明 Pod 身份注入器	其他	借助凭据管理系统（SSM）和腾讯云访问控制管理（CAM）来简化访问腾讯云数据库的整个过程，从而消除验证腾讯云数据库用户名和密钥存在的安全风险；同时凭据管理系统（SSM）定时轮转访问凭证的特性，间接解决人为操作所带来的负担。
tke-log-agent 日志采集组件	日志采集	日志组件是当用户开启容器服务内的日志服务时，腾讯云日志服务部署于用户集群内每个标准节点上的组件，用于采集容器服务产生的业务日志并写入腾讯云上的消费端，目前支持写入 CLS 和 Kafka。
GooseFS-CSI 说明 数据加速器组件	存储	该组件实现了 CSI 接口，可帮助容器集群使用腾讯云数据加速器。

组件的生命周期管理

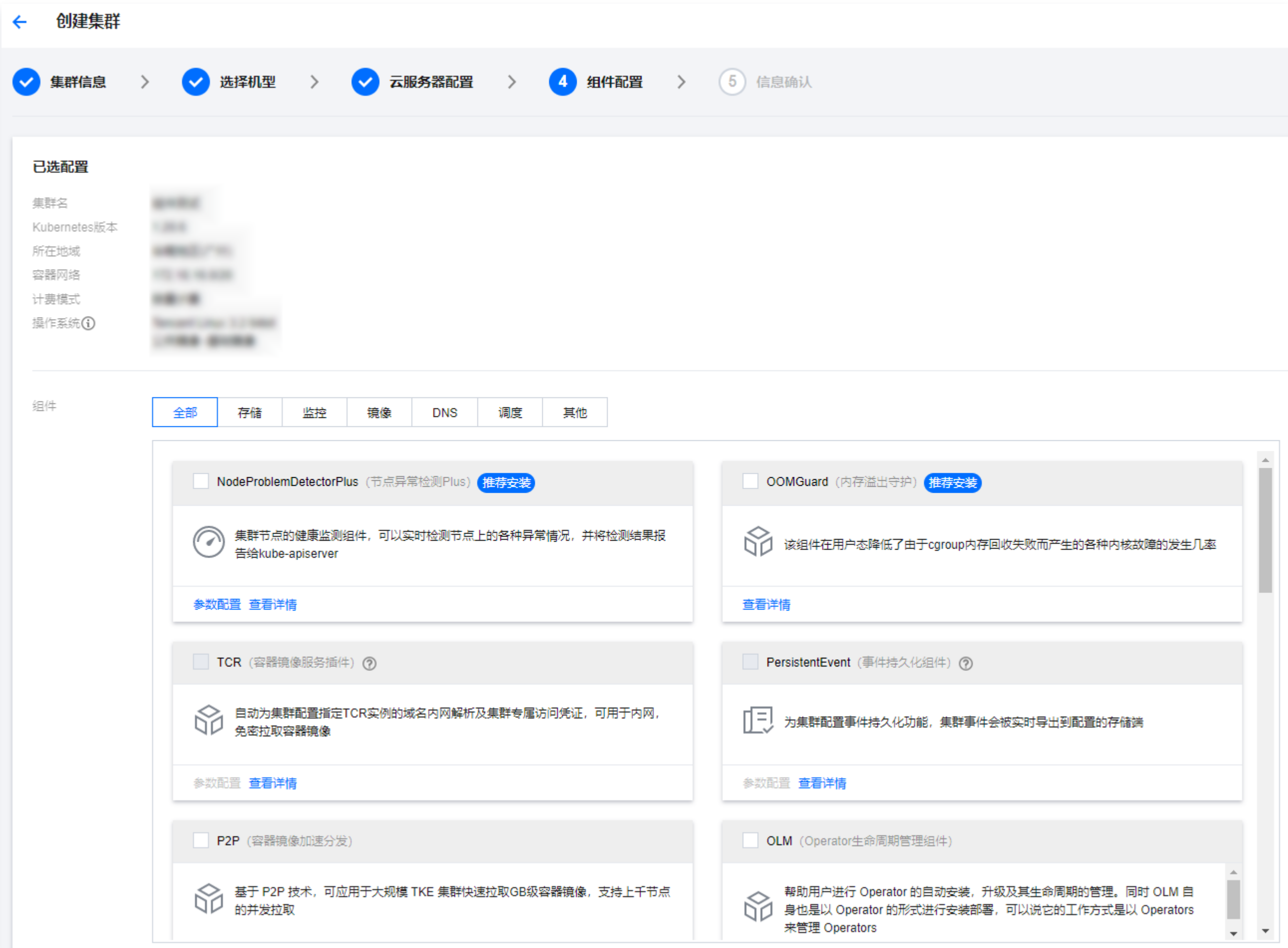
最近更新时间：2025-06-25 14:27:12

组件安装

您可以通过集群创建页或通过组件管理页安装增强组件。

通过集群创建页安装

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在**集群**页面，选择地域，单击集群列表上方的**新建**。
3. 选择集群类型，单击**创建**。在**创建集群**页面，依次填写集群的**集群信息**、**选择机型**、**云服务器配置**及**组件配置**。如下图所示：



4. 您可以根据业务部署情况，按需选择合适的组件安装。单击每个组件卡片的**查看详情**可以查看该组件的介绍，部分组件需要您先完成**参数配置**。

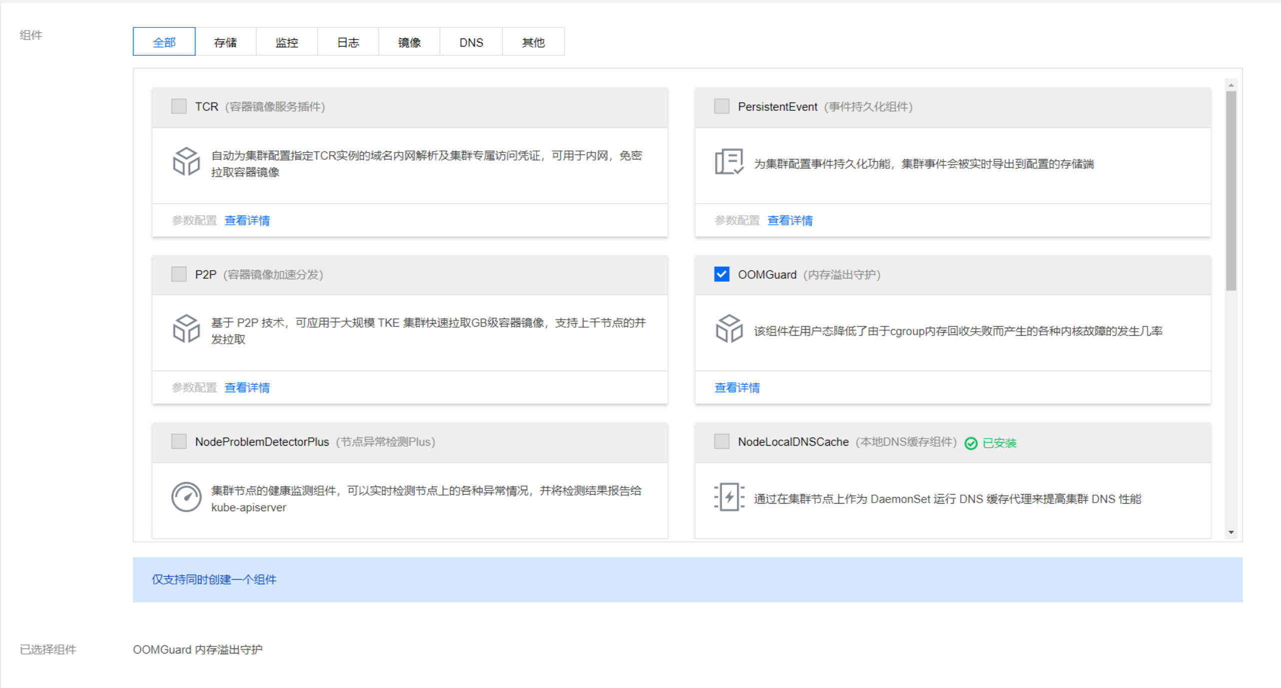
> **说明：**
> - 组件安装为集群创建的非关键路径，安装失败不会影响集群的创建。
> - 组件安装需要占用集群的一定资源，不同组件的资源占用情况不同，单击**查看详情**查看每个组件的详细信息。

5. 单击**下一步**，检查并确认集群配置信息。
6. 单击**完成**，即可完成创建。

通过组件管理页安装

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。

4. 在**新建组件管理**页面中勾选组件。如下图所示：



5. 单击**完成**。

组件卸载

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击需要删除组件所在行右侧的**删除**，如下图所示：

4. 在删除资源页面，单击**确定**即可完成组件卸载。

组件升级

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击需要升级组件所在行右侧的**升级**。
4. 在组件升级页面，单击**确定**即可完成升级。

> **说明：**
> 各组件版本更新详情，请参考 增强组件版本维护说明 。

Cerberus 说明

最近更新时间：2024-01-29 15:18:51

组件介绍

Cerberus 组件支持对签名镜像进行可信验证，确保在 TKE 集群中只部署可信授权方签名的容器镜像，降低在容器环境中的镜像安全风险。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
cerberus-webhook	Deployment	每个实例 CPU 0.5C，Memory 0.5G，共1个实例	tcr-verification
cerberus-webhook	Service	-	tcr-verification
cerberus-webhook	Configmap	-	tcr-verification
cerberus-webhook	Secret	-	tcr-verification
cerberus-webhook	validatingwebhookconfigurations	-	-
cerberus-rolebinding	ClusterRoleBinding	-	-
cerberus-role	ClusterRole	-	-
cerberus-controller	ServiceAccount	-	-

限制条件

- TKE 集群版本 = 1.18。
- 开通并使用 TCR 高级版 容器镜像签名 功能后，才可在 TKE 集群中为已加签的镜像进行验签。
- 验签需要提前授权容器服务角色拥有使用 TCR 接口的权限。

组件权限说明

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要支持镜像验签功能，所以需要 tcr.tencentcloudcr.com/verifications 的 create、 delete、get、list、patch、update、watch 权限。
- 不涉及 secret 等资源的敏感权限。

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: cerberus-role
rules:
  - apiGroups:
      - tcr.tencentcloudcr.com
    resources:
      - verifications
    verbs:
      - create
      - delete
      - get
      - list
      - patch
      - update
```

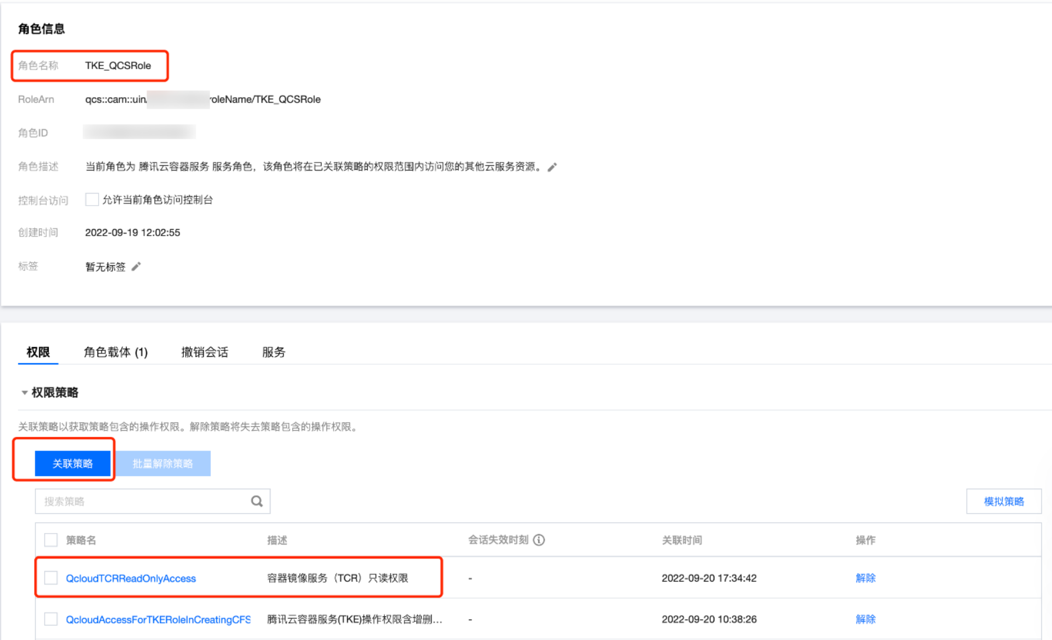
```
      - watch
  - apiGroups:
      - tcr.tencentcloudcr.com
    resources:
      - verifications/finalizers
    verbs:
      - update
  - apiGroups:
      - tcr.tencentcloudcr.com
    resources:
      - verifications/status
    verbs:
      - get
      - patch
      - update
```

使用方法

授权容器服务使用 TCR 查询接口

为让 TKE 服务可以读取到您账号下的签名信息，需要在您的账号配置如下策略。

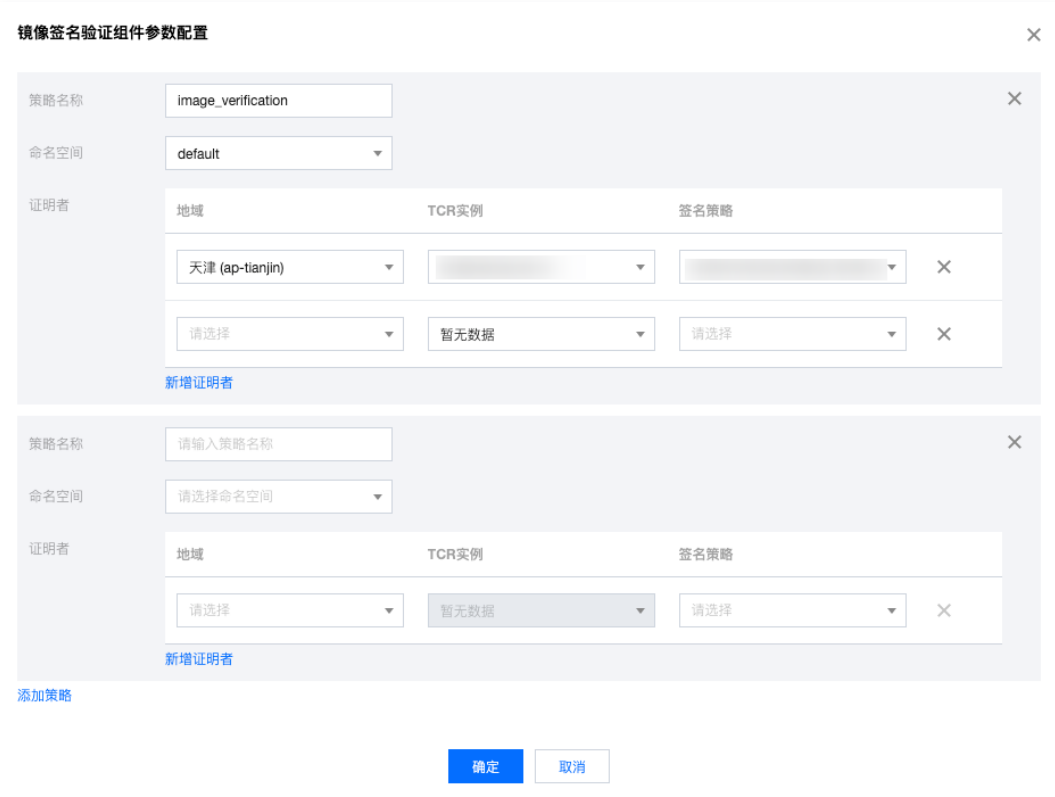
1. 登录 访问管理控制台 。
2. 在角色页面，单击 **TCR_QCSRole**。
3. 在 TCR_QCSRole 角色详情页，关联预设策略 **QcloudTCRReadOnlyAccess**，如下图所示：



安装组件并配置验签策略

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。

4. 在**新建组件管理**页面中勾选 Cerberus 组件并进行签名验证策略的参数配置，说明如下：



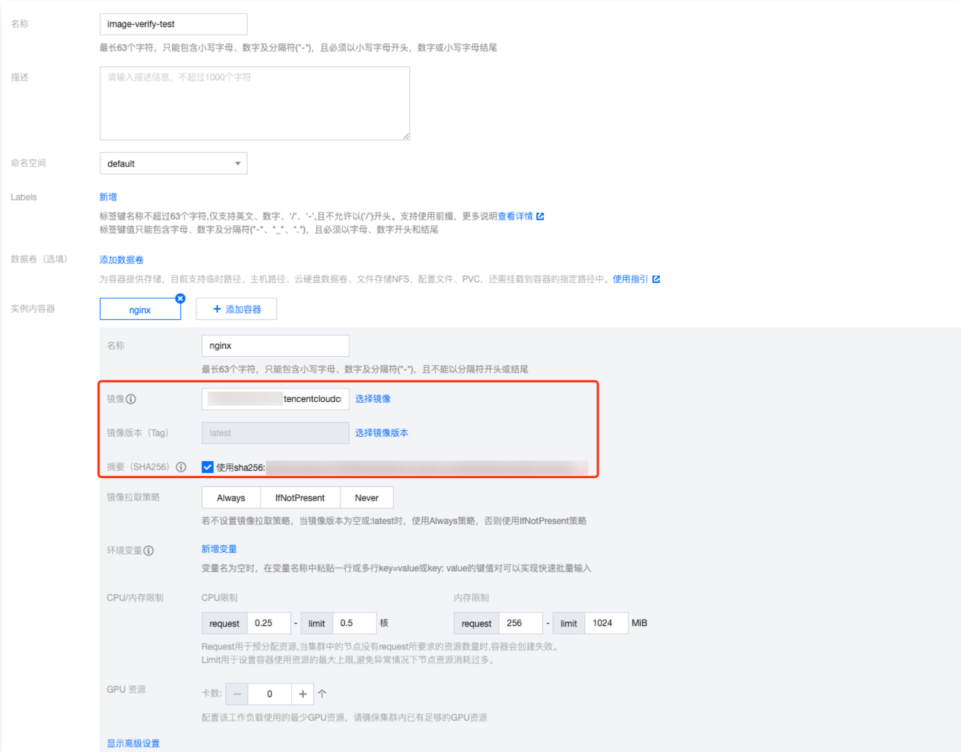
  - **策略名称**：请为验签策略设置合理名称。
  - **命名空间**：请选择当前验签策略生效的 Kubernetes 命名空间，每个验签策略仅对一个命名空间生效。
  - **证明者**：证明者将定位到镜像加签时所采用的 KMS 密钥，可通过该字段确保镜像签名来源的可信性。
    - **地域**：选择 TCR 镜像仓库实例所在的地域，和 TKE 集群所在地域无强制关联性。
    - **TCR 实例**：选择 TCR 镜像仓库实例。
    - **签名策略**：选择已经创建的签名策略。签名策略和镜像签名时使用的 KMS 密钥相对应，支持选择多个签名策略。
5. 单击**确定**完成组件创建。

> **说明**
> - 组件创建时则验签策略生效，未通过签名验证的镜像将部署失败。为保障 TKE 基础服务的正常运行，请不要为 TKE 系统组件所在的命名空间（如 kube-system）设置验签策略。
> - 若您需要为集群中相同 Kubernetes 命名空间绑定不同 TCR 仓库实例下的签名策略，可在当前验签策略中**新增证明者**；若您需要为集群的多个 kubernetes 命名空间都设置镜像签名验证，可直接**添加策略**。

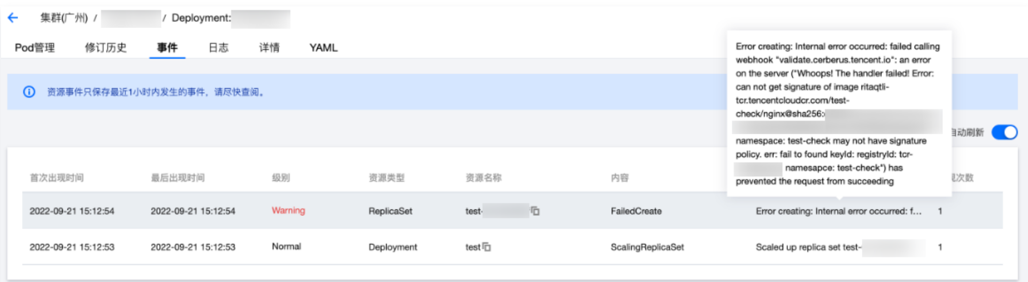
镜像签名验证

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**工作负载**，单击**新建**。

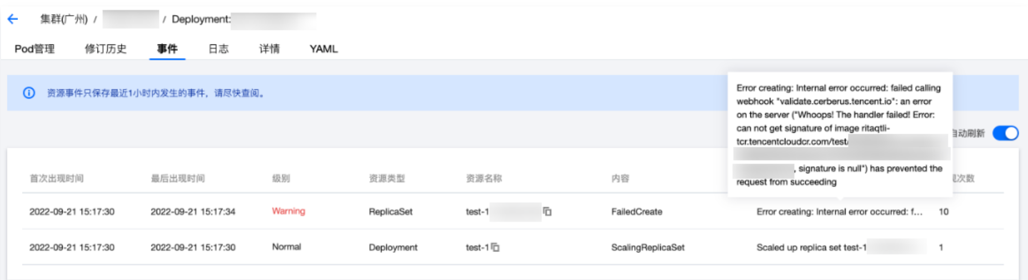
4. 在新建工作负载页，配置**实例内容器 > 镜像**时，选择待部署的已经加签的容器镜像及镜像版本（Tag），同时勾选**摘要（SHA256）**，如下图所示：



5. 单击**完成**创建工作负载部署。若签名验证通过，则 Pod 将正常部署；若签名验证不通过，则 Pod 部署将被阻断，相关 event 信息展示如下图所示：
    - 若拉取镜像所在 TCR 仓库实例下未开启镜像签名特性，则报错如下：



    - 若拉取镜像没有签名信息，则报错如下：



    - 若拉取镜像的签名信息验证不通过，则报错如下：

> **说明：**
> 镜像签名验证目前只支持使用 digest 格式的镜像，否则会验证不通过。

CBS-CSI 说明
CBS-CSI 简介

最近更新时间：2024-01-30 14:46:11

操作场景

CBS-CSI 组件 支持 TKE 集群通过控制台快捷选择存储类型，并创建对应块存储云硬盘类型的 PV 和 PVC。本文提供 CBS-CSI 组件功能特性等说明并介绍几种常见示例用法。

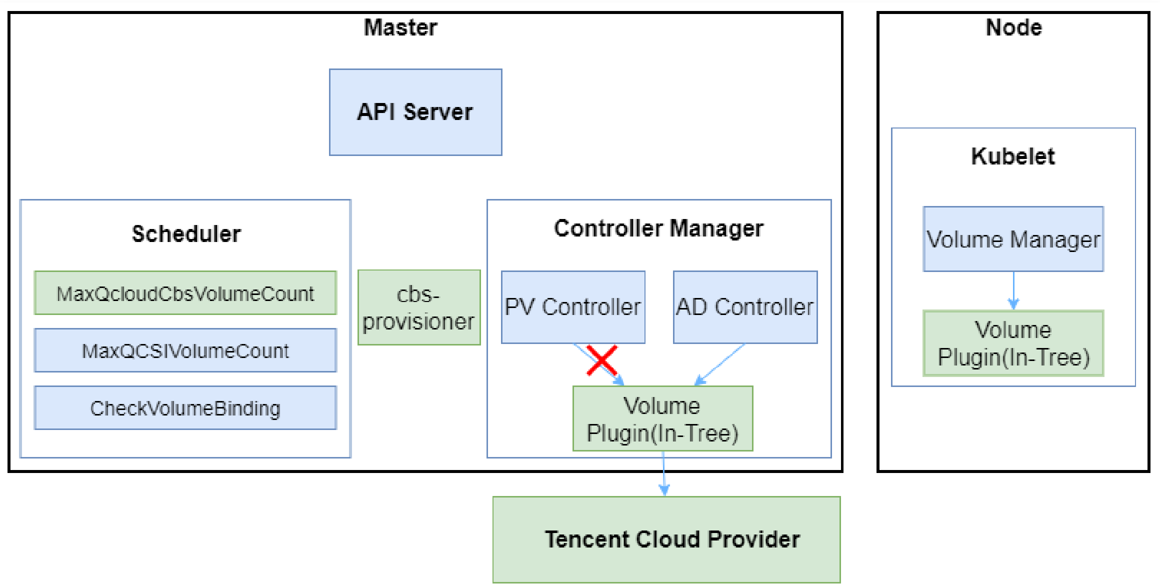
功能特性

功能	说明
静态数据卷	支持手动创建 Volume、PV 对象及 PVC 对象。
动态数据卷	支持通过 StorageClass 配置、创建和删除 Volume 及 PV 对象。
存储拓扑感知	云硬盘不支持跨可用区挂载，在多可用区集群中，CBS-CSI 组件将先调度 Pod，后调度 Node 的 zone 创建 Volume。
调度器感知节点 maxAttachLimit	腾讯云单个云服务器上默认最多挂载20块云硬盘，调度器调度 Pod 时将过滤超过最大可挂载云硬盘数量的节点。
卷在线扩容	支持通过修改 PVC 容量字段，实现在线扩容（仅支持云硬盘类型）。
卷快照和恢复	支持通过快照创建数据卷。

组件说明

CBS-CSI 组件在集群内部署后，包含以下组件：
- DaemonSet：每个 Node 提供一个 DaemonSet，简称为 NodePlugin。由 CBS-CSI Driver 和 node-driver-registrar 两个容器组成，负责向节点注册 Driver，并提供挂载能力。
- StatefulSet 和 Deployment：简称为 Controller。由 Driver 和多个 Sidecar（external-provisioner、external-attacher、external-resizer、external-snapshotter、snapshot-controller）一起构成，提供创删卷、attach、detach、扩容、快照等能力。

示例图如下：



限制条件

- TKE 集群版本 ≥ 1.14
- 使用 CBS-CSI 组件后，才可在 TKE 集群中为云硬盘在线扩容和创建快照。

- 已经使用 QcloudCbs（In-Tree 插件）的 TKE 集群，可以继续正常使用。（后续将通过 Volume Migration 统一到 CBS-CSI）
- CBS-CSI 组件版本 ≥ 1.0.4，支持原生节点。

cbs-csi 权限

> **说明：**
> **权限场景**章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考**权限定义**章节。

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机 /var/lib/kubelet 相关目录到容器来完成 volume 的 mount/umount，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
获取 node 资源中 providerID 来感知节点最大可挂盘数量	node	get/list
根据 pvc/pv 等信息完成盘的创建和删除	pv/pvc/storageclasses/csinode	get/list/watch/create/update/patch/delete
根据 volumeattachments 资源对象来完成盘的挂载和卸载	volumeattachments/volumesnapshotclasses	create/get/list/watch/update/delete
对盘进行扩容快照	pod/volumesnapshotclasses/volumesnapshots/configmap	get/list/watch

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cbs-csi-controller-role
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "update", "patch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims/status"]
    verbs: ["update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["csinodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
```



```
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
  - apiGroups: ["csi.storage.k8s.io"]
    resources: ["csinodeinfos"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments", "volumeattachments/status"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotcontents"]
    verbs: ["create", "get", "list", "watch", "update", "delete"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    verbs: ["create", "list", "watch", "delete"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotcontents/status"]
    verbs: ["update"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshots/status"]
    verbs: ["update"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "update", "patch", "create", "delete"]
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cbs-csi-node-role
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list"]
```

使用示例

- 通过 CBS-CSI 避免云硬盘跨可用区挂载
- 在线扩容云硬盘
- 创建快照和使用快照来恢复卷

通过 CBS-CSI 避免云硬盘跨可用区挂载

最近更新时间：2024-09-29 16:49:41

操作场景

云硬盘不支持跨可用区挂载到节点，在跨可用区的集群环境中，推荐通过 CBS-CSI **拓扑感知**特性来避免跨可用区挂载问题。

实现原理

拓扑感知调度需要多个 Kubernetes 组件配合完成，包括 Scheduler、PV controller、external-provisioner。具体流程如下：

1. PV controller 观察 PVC 对象，检查 Storageclass 的 VolumeBindingMode 是否为 **WaitForFirstConsumer**，如是，则不会立即处理该 PVC 的创建事件，等待 Scheduler 处理。
2. Scheduler 调度 Pod 后，会将 nodeName 以 annotation 的方式加入到 PVC 对象上 `volume.kubernetes.io/selected-node: 10.0.0.72`。
3. PV controller 获取到 PVC 对象的更新事件后，将开始处理 annotation（`volume.kubernetes.io/selected-node`），根据 nodeName 获取 Node 对象，传入到 external-provisioner 中。
4. external-provisioner 根据传过来的 Node 对象的 label 获取可用区（`failure-domain.beta.kubernetes.io/zone`）后在对应可用区创建 PV，达到和 Pod 相同可用区的效果，避免云硬盘和 Node 在不同可用区而无法挂载问题。

前提条件

- 已安装1.14或以上版本的 TKE 集群。
- 已将 CBS-CSI 或 In-Tree 组件更新为最新版本。

操作步骤

使用以下 YAML，在 Storageclass 中设置 volumeBindingMode 为 **WaitForFirstConsumer**。示例如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cbs-topo
parameters:
  type: cbs
provisioner: com.tencent.cloud.csi.cbs
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

> **说明：**
> CBS-CSI 和 In-Tree 组件均支持该操作。

在线扩容云硬盘

最近更新时间：2024-12-30 14:17:42

操作场景

TKE 支持在线扩容 PV、对应的云硬盘及文件系统，即不需要重启 Pod 即可完成扩容。为确保文件系统的稳定性，建议在云硬盘文件系统处于未挂载状态时进行操作。

前提条件

- 已创建1.16或以上版本的 TKE 集群。
- 已将 CBS-CSI 更新为最新版本。
- （可选）为避免扩容失败导致数据丢失，可以在扩容前 使用快照备份数据。
- 1.20以下集群非 CBS-CSI 类型的 PV 不支持在线扩容。

操作步骤

创建允许扩容的 StorageClass

方式一：控制台创建允许扩容的 StorageClass

1. 选择左侧菜单栏中的**存储 > StorageClass**，在 StorageClass 页面单击**新建**。如下图所示：

2. 在新建 StorageClass 页面中，勾选**启用在线扩容**。

方式二：使用 YAML 创建允许扩容的 StorageClass

使用以下 YAML 创建允许扩容的 StorageClass，在 Storageclass 中设置 `allowVolumeExpansion` 为 `true` 。示例如下：

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cbs-csi
parameters:
  diskType: CLOUD_PREMIUM
provisioner: com.tencent.cloud.csi.cbs
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

在线扩容

提供以下两种扩容方式：

扩容方式	说明
重启 Pod 的情况下在线扩容	待扩容的云硬盘文件系统未被挂载，能够避免扩容出错以及方式2存在的问题。**推荐使用该方式进行扩容**。
不重启 Pod 的情况下在线扩容	在节点上挂载着待扩容的云硬盘文件系统，如果存在 I/O 进程，将可能出现文件系统扩容错误。

重启Pod情况下在线扩容

1. 执行以下命令，确认扩容前 PV 和文件系统状态。示例如下，PV 和文件系统大小均为30G：

```
$ kubectl exec ivantestweb-0 df /usr/share/nginx/html
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdd        30832548 44992  30771172   1% /usr/share/nginx/html
$ kubectl get pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                          STORAGECLASS   REASON   AGE
pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c   30Gi       RWO            Delete           Bound
default/www1-ivantestweb-0   cbs-csi                 20h
```

说明：
PV 名称：pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c，查询方式：
1. 登录 容器服务控制台，在集群管理页面，单击目标集群 ID，进入集群详情页。
2. 选择左侧导航中的**存储**，在 PersistentVolume 页面查看名称。

2. 执行以下命令，为 PV 对象打上一个非法 zone 标签，旨在下一步重启 Pod 后，使 Pod 无法调度到某个节点上。示例如下：

```
$ kubectl label pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c failure-
domain.beta.kubernetes.io/zone=nozone
```

3. 执行以下命令重启 Pod，重启后由于 Pod 对应的 PV 的标签表明的是非法 zone，Pod 将处于 Pending 状态。示例如下：

```
$ kubectl delete pod ivantestweb-0
$ kubectl get pod ivantestweb-0
NAME            READY   STATUS    RESTARTS   AGE
ivantestweb-0   0/1     Pending   0          25s
$ kubectl describe pod ivantestweb-0
Events:
Type     Reason             Age                  From                 Message
----     ------             ----                 ----                 -------
```

```
Warning  FailedScheduling  40s (x3 over 2m3s)  default-scheduler  0/1 nodes are available: 1
node(s) had no available volume zone.
```

4. 执行以下命令，修改 PVC 对象中的容量，将容量扩容至40G。示例如下：

```
kubectl patch pvc www1-ivantestweb-0 -p '{"spec":{"resources":{"requests":{"storage":"40Gi"}}}}'
```

> **注意：**
> - 设置的扩容后容量不允许小于当前容量。
> - 扩容后的 PVC 对象容量的大小必须为10的倍数，不同云硬盘类型所支持的存储容量规格可参考说明 云硬盘类型 。

5. 执行以下命令，去除 PV 对象之前打上的标签， 标签去除之后 Pod 即可调度成功。示例如下：

```
$ kubectl label pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c failure-domain.beta.kubernetes.io/zone-
persistentvolume/pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c unlabeled
```

6. 执行以下命令，可以查看到 Pod 状态为 Running、对应的 PV 和文件系统扩容成功，从30G扩容到40G。示例如下：

```
$ kubectl get pod ivantestweb-0
NAME            READY   STATUS    RESTARTS   AGE
ivantestweb-0   1/1     Running   0          17m
$ kubectl get pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                        STORAGECLASS   REASON   AGE
pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c   40Gi       RWO            Delete           Bound
default/www1-ivantestweb-0   cbs-csi                 20h
$ kubectl get pvc www1-ivantestweb-0
NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
www1-ivantestweb-0   Bound    pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c   40Gi       RWO
cbs-csi        20h
$ kubectl exec ivantestweb-0 df /usr/share/nginx/html
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdd        41153760 49032  41088344   1% /usr/share/nginx/html
```

不重启Pod情况下在线扩容

1. 执行以下命令，确认扩容前 PV 和文件系统状态。示例如下，PV 和文件系统大小均为20G：

```
$ kubectl exec ivantestweb-0 df /usr/share/nginx/html
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdd        20511312 45036  20449892   1% /usr/share/nginx/html
$ kubectl get pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                        STORAGECLASS   REASON   AGE
pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c   20Gi       RWO            Delete           Bound
default/www1-ivantestweb-0   cbs-csi                 20h
```

2. 执行以下命令，修改 PVC 对象中的容量，将容量扩容至30G。示例如下：

```
$ kubectl patch pvc www1-ivantestweb-0 -p '{"spec":{"resources":{"requests":{"storage":"30Gi"}}}}'
```

> **注意：**

扩容后的PVC对象容量的大小必须为10的倍数，不同硬盘类型所支持的存储容量规格可参考说明 云硬盘类型 。

3. 执行以下命令，可以查看到 PV 和文件系统已扩容至30G。示例如下：

```
$ kubectl exec ivantestweb-0 df /usr/share/nginx/html
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vdd        30832548 44992  30771172   1% /usr/share/nginx/html
$ kubectl get pv pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                            STORAGECLASS   REASON   AGE
pvc-e193201e-6f6d-48cf-b96d-ccc09225cf9c   30Gi       RWO            Delete           Bound
default/www1-ivantestweb-0   cbs-csi                 20h
```

创建快照和使用快照来恢复卷

最近更新时间：2025-07-01 14:42:22

操作场景

如需为 PVC 数据盘创建快照来备份数据，或者将备份的快照数据恢复到新的 PVC 中，可以通过 CBS-CSI 插件来实现，本文将介绍如何利用 CBS-CSI 插件实现 PVC 的数据备份与恢复。

前提条件

- 已创建1.18或以上版本的 TKE 集群。
- 已安装最新版的 CBS-CSI 组件。

操作步骤

备份PVC

创建 VolumeSnapshotClass

1. 使用以下 YAML，创建 VolumeSnapshotClass 对象。示例如下：

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: cbs-snapclass
driver: com.tencent.cloud.csi.cbs
deletionPolicy: Delete
```

2. 执行以下命令查看 VolumeSnapshotClass 是否创建成功。示例如下：

```
$ kubectl get volumesnapshotclass
NAME            DRIVER                         DELETIONPOLICY    AGE
cbs-snapclass   com.tencent.cloud.csi.cbs     Delete            17m
```

创建 PVC 快照 VolumeSnapshot

1. 本文以 `new-snapshot-demo` 快照名为例，使用以下 YAML 创建 VolumeSnapshot 对象。示例如下：

> **说明：**
> 请将以下代码中的 PVC 名称 csi-pvc 替换为您集群下具体要创建快照的 PVC 名称。

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: new-snapshot-demo
spec:
  volumeSnapshotClassName: cbs-snapclass
  source:
    persistentVolumeClaimName: csi-pvc
```

2. 执行以下命令，查看 Volumesnapshot 和 Volumesnapshotcontent 对象是否创建成功，若 `READYTOUSE` 为 true，则创建成功。示例如下：

```
$ kubectl get volumesnapshot
NAME                 READYTOUSE   SOURCEPVC            SOURCESNAPSHOTCONTENT   RESTORESIZE
SNAPSHOTCLASS    SNAPSHOTCONTENT                                    CREATIONTIME   AGE
new-snapshot-demo   true         www1-ivantestweb-0                           10Gi          cbs-
snapclass   snapcontent-ea11a797-d438-4410-ae21-41d9147fe610   22m            22m
```

```
$ kubectl get volumesnapshotcontent
NAME                                                      READYTOUSE   RESTORESIZE   DELETIONPOLICY   DRIVER
VOLUMESNAPSHOTCLASS   VOLUMESNAPSHOT      AGE
snapcontent-ea11a797-d438-4410-ae21-41d9147fe610   true         10737418240   Delete
com.tencent.cloud.csi.cbs   cbs-snapclass          new-snapshot-demo   22m
```

3. 执行以下命令，可以获取 Volumesnapshotcontent 对象的快照 ID，字段是 `status.snapshotHandle`（如下为 snap-e406fc9m），可以根据快照 ID 在 云服务控制台 > 快照列表 确认快照是否存在。示例如下：

```
$ kubectl get volumesnapshotcontent snapcontent-ea11a797-d438-4410-ae21-41d9147fe610 -oyaml
```

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotContent
metadata:
  creationTimestamp: "2020-11-04T08:58:39Z"
  finalizers:
  - snapshot.storage.kubernetes.io/volumesnapshotcontent-bound-protection
  name: snapcontent-ea11a797-d438-4410-ae21-41d9147fe610
  resourceVersion: "471437790"
  selfLink: /apis/snapshot.storage.k8s.io/v1beta1/volumesnapshotcontents/snapcontent-ea11a797-d438-
4410-ae21-41d9147fe610
  uid: 70d0390b-79b8-4276-aa79-a32e3bdef3d6
spec:
  deletionPolicy: Delete
  driver: com.tencent.cloud.csi.cbs
  source:
    volumeHandle: disk-7z32tin5
  volumeSnapshotClassName: cbs-snapclass
  volumeSnapshotRef:
    apiVersion: snapshot.storage.k8s.io/v1beta1
    kind: VolumeSnapshot
    name: new-snapshot-demo
    namespace: default
    resourceVersion: "471418661"
    uid: ea11a797-d438-4410-ae21-41d9147fe610
status:
  creationTime: 1604480319000000000
  readyToUse: true
  restoreSize: 10737418240
  snapshotHandle: snap-e406fc9m
```

从快照恢复数据到新 pvc

1. 本文以上述 步骤 中创建的 VolumeSnapshot 的对象名为 `new-snapshot-demo` 为例，使用以下 YAML 从快照恢复卷。示例如下：

> **说明：**
> 请将以下代码中的 StorageClass 名称 cbs-csi 替换为您集群对应的 StorageClass 名称。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restore-test
spec:
  storageClassName: cbs-csi
  dataSource:
    name: new-snapshot-demo
    kind: VolumeSnapshot
```

```
    apiGroup: snapshot.storage.k8s.io
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

2. 执行以下命令，查看恢复的 PVC 已成功创建，从 PV 中可以查看到对应的 diskid（如下为 disk-gahz1kw1）。示例如下：

```
$ kubectl get pvc restore-test
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
restore-test   Bound    pvc-80b98084-29a3-4a38-a96c-2f284042cf4f   10Gi       RWO            cbs-csi
97s
```

```
$ kubectl get pv pvc-80b98084-29a3-4a38-a96c-2f284042cf4f -oyaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: com.tencent.cloud.csi.cbs
  creationTimestamp: "2020-11-04T12:08:25Z"
  finalizers:
  - kubernetes.io/pv-protection
  name: pvc-80b98084-29a3-4a38-a96c-2f284042cf4f
  resourceVersion: "474676883"
  selfLink: /api/v1/persistentvolumes/pvc-80b98084-29a3-4a38-a96c-2f284042cf4f
  uid: 5321df93-5f21-4895-bafc-71538d50293a
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: restore-test
    namespace: default
    resourceVersion: "474675088"
    uid: 80b98084-29a3-4a38-a96c-2f284042cf4f
  csi:
    driver: com.tencent.cloud.csi.cbs
    fsType: ext4
    volumeAttributes:
      diskType: CLOUD_PREMIUM
      storage.kubernetes.io/csiProvisionerIdentity: 1604478835151-8081-com.tencent.cloud.csi.cbs
    volumeHandle: disk-gahz1kw1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: topology.com.tencent.cloud.csi.cbs/zone
          operator: In
          values:
          - ap-beijing-2
  persistentVolumeReclaimPolicy: Delete
  storageClassName: cbs-csi
```



```
  volumeMode: Filesystem
status:
  phase: Bound
```

说明：

如果 StorageClass 使用了拓扑感知（先调度 Pod 再创建 PV），即指定 `volumeBindingMode: WaitForFirstConsumer`，则需要先部署 Pod（需挂载 PVC）才会触发创建 PV（从快照创建新的 CBS 并与 PV 绑定）。

CFS-CSI 说明

最近更新时间：2025-03-31 17:00:33

简介

组件介绍

CFS-CSI 组件 实现 CSI 的接口，可帮助您在容器集群中使用腾讯云文件存储。

> **注意：**
> 1.12集群需要修改 kubelet 配置，增加 `--feature-gates=KubeletPluginsWatcher=false` 参数。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
csi-provisioner-cfsplugin	StatefulSet	-	kube-system
csi-nodeplugin-cfsplugin	DaemonSet	-	kube-system
csi-provisioner-cfsplugin	Service	1C2G	kube-system

使用场景

文件存储 CFS 提供了可扩展的共享文件存储服务，可与腾讯云 CVM、容器服务 TKE、批量计算等服务搭配使用。CFS 提供了标准的 NFS 及 CIFS/SMB 文件系统访问协议，为多个 CVM 实例或其他计算服务提供共享的数据源，支持弹性容量和性能的扩展，现有应用无需修改即可挂载使用，是一种高可用、高可靠的分布式文件系统，适合于大数据分析、媒体处理和内容管理等场景。

CFS 接入简单，您无需调整自身业务结构，或者是进行复杂的配置。只需三步即可完成文件系统的接入和使用：创建文件系统，启动服务器上文件系统客户端，挂载创建的文件系统。通过 CFS-CSI 扩展组件，您可以快速在容器集群中通过标准原生 Kubernetes 使用 CFS，详情请参见 CFS 使用场景 。

限制条件

- CFS 自身限制可参见 CFS 系统限制 。
- 在 TKE 中使用 CFS，需要在集群内安装该扩展组件，这将占用一定的系统资源。

cfs-csi 权限

> **说明：**
> **权限场景**章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考**权限定义**章节。

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机/var/lib/kubelet相关目录到容器来完成volume的mount/umount，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
需要支持动态创建 cfs 实例	persistentvolumeclaims/persistentvolumes	所有操作
	storageclasses	get/list/watch
支持共享模式下的 cfs 实例	tcfs	get/list/watch/create/update/delete/patch
	deployment	get/list/watch/create/update/delete
	node	get/list

权限定义

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: csi-cfs-controller-role
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list"]
  - apiGroups: [""]
    resources: ["services", "events", "configmaps", "endpoints"]
    verbs: ["get","list","create","update","patch","delete"]
  - apiGroups: [""]
    resources: ["services/status", "events/status"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete", "update"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update", "patch", "create"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments", "volumeattachments/status"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "update", "patch", "create"]
  - apiGroups: ["extensions"]
    resources: ["ingresses/status"]
    verbs: ["get"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get", "list", "delete", "update", "create", "watch"]
  - apiGroups: ["apps"]
    resources: ["deployments/status"]
    verbs: ["get"]
  - apiGroups: ["tcfsoperator.k8s.io"]
    resources: ["tcfs", "tcfs/status"]
    verbs: ["get", "list", "watch", "create", "delete", "update", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tcfs-subdir-external-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
```

```
    resources: ["events"]
    verbs: ["create", "update", "patch"]
```

操作步骤

安装并设置 CFS 扩展组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件页面单击**新建**。
4. 在**新建组件管理**页面中勾选 CFS。
5. 单击**完成**即可创建组件。

创建 CFS 类型 StorageClass

1. 在集群列表中，单击使用 CFS 的集群 ID，进入集群详情页。
2. 在左侧导航栏中选择**存储**，在 StorageClass 中单击**新建**。
3. 在**新建存储**页面，根据实际需求，创建 CFS 类型的 StorageClass。如下图所示：

4. 单击**创建 StorageClass**，完成创建。

创建 PersistentVolumeClaim

1. 在集群列表中，单击使用 CFS 的集群 ID，进入集群详情页。
2. 在左侧导航栏中选择**存储**，在 PersistentVolumeClaim 中单击**新建**。
3. 在**新建存储**页面，根据实际需求，创建 CFS 类型 PersistentVolumeClaim，选择上述步骤创建的 StorageClass。
4. 单击**创建 PersistentVolumeClaim**，完成创建。

创建工作负载

1. 在集群列表中，单击使用 CFS 的集群 ID，进入集群详情页。
2. 在左侧导航栏中选择**工作负载**，在 Deployment 中单击**新建**。
3. 在**新建工作负载**页面，根据实际需求，数据卷选择**使用已有 PVC**，并选择上述已创建的 PVC。
4. 挂载到容器的指定路径后，单击**创建 Deployment** 完成创建。

CFSTURBO-CSI 说明

最近更新时间：2024-01-30 14:46:11

组件介绍

Kubernetes-csi-tencentcloud CFSTURBO 插件实现 CSI 的接口，可帮助您在容器集群中使用腾讯云 CFS Turbo 文件存储。

使用场景

文件存储 CFS Turbo 适用于大规模吞吐型和混合负载型业务，它提供了私有协议的挂载方式，单个客户端的性能可达到存储集群的性能水平。您可以将其与腾讯云云服务器 CVM、容器服务 TKE 和批量计算等服务搭配使用。
CFS Turbo 接入简单，您无需调整现有业务结构或进行复杂的配置。只需三个步骤即可完成文件系统的接入和使用：创建文件系统、启动服务器上的文件系统客户端，然后挂载所创建的文件系统。通过 CFSTURBO-CSI 扩展组件，您可以快速在容器集群中通过标准原生 Kubernetes 使用 CFS Turbo。

限制条件

- 使用 CFSTURBO-CSI 扩展组件需要 Kubernetes 版本大于等于1.14。
- CFS Turbo 自身限制请参见 CFS 系统限制。
- 在 TKE 中使用 CFS Turbo，需要在集群内安装该扩展组件，这将占用一定的系统资源。

cfsturbo-csi 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机 /var/lib/kubelet 相关目录到容器来完成 volume 的 mount/umount，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
支持动态创建/删除 cfsturbo 子目录类型 pv	persistentvolumeclaims/persistentvolumes/storageclasses	get/list/watch/create/delete/update
	node	get/list/

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cfsturbo-csi-controller-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete", "update"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
```

操作步骤

安装组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 CFSTurbo。
5. 单击**完成**即可创建组件。

指定 StroageClass

步骤1：创建 CFS Turbo 类型 StroageClass

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储** > **StorageClass**，在 StorageClass 页面单击**新建**。
3. 在**新建 StorageClass** 页面中，配置 StorageClass 参数。如下图所示：

配置项	描述
名称	填写 StorageClass 的名称。
地域	默认为集群所在地域。
Provisioner	此处选择文件存储 CFS Turbo。
CFS Turbo	此处选择已创建的 CFS Turbo。如果您没有合适的 CFS Turbo，请前往文件存储控制台新建，详情请参见 创建文件系统及挂载点 。
回收策略	提供删除和保留两种回收策略，出于数据安全考虑，推荐使用保留回收策略。 • 删除：通过 PVC 动态创建的 PV，在 PVC 销毁时，与其绑定的 PV 和存储实例也会自动销毁。 • 保留：通过 PVC 动态创建的 PV，在 PVC 销毁时，与其绑定的 PV 和存储实例会被保留。

4. 单击**创建 StorageClass**，完成创建。

步骤2：创建 PersistentVolumeClaim

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储** > **PersistentVolumeClaim**，在 PersistentVolumeClaim 页面单击**新建**。
3. 在**新建 PersistentVolumeClaim** 页面中，配置 PVC 关键参数。

配置项	描述
名称	填写 PersistentVolumeClaim 的名称。
命名空间	命名空间用来划分集群资源。此处选择 default。
Provisioner	选择文件存储 CFS Turbo。
读写权限	文件存储仅支持多机读写。

是否指定StorageClass	选择**指定 StorageClass**。 **说明：** • PVC 和 PV 会绑定在同一个 StorageClass 下。 • **不指定 StorageClass** 意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。
StorageClass	选择上述步骤创建的 StorageClass。
是否指定 PersistVolume	按需指定 PersistentVolume。 **说明：** • 系统首先会筛选当前集群内是否存在符合绑定规则的 PV，若没有则根据 PVC 和所选 StorageClass 的参数动态创建 PV 与之绑定。 • 系统不允许在不指定 StorageClass 的情况下同时选择不指定 PersistVolume。 • 关于**不指定 PersistVolume** 的详细介绍，请参见 查看 PV 和 PVC 的绑定规则。

4. 单击**创建 PersistentVolumeClaim**。

步骤3：创建工作负载

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**工作负载** > **Deployment**，在 Deployment 页面单击**新建**。
3. 在新建 Deployment 页面中，配置工作负载参数，参数详情请参见 创建 Deployment。根据实际需求，数据卷选择**使用已有PVC**，并选择上述已创建的 PVC。
4. 挂载到容器的指定路径后，单击**创建 Deployment**。

不指定 StroageClass

步骤1：创建 PersistentVolume

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储** > **PersistentVolume**，在 PersistentVolume 页面单击**新建**。
3. 在**新建 PersistentVolume** 页面中，配置 PV 关键参数。如下图所示：

配置项	描述
来源设置	选择**静态创建**。

名称	填写 PersistentVolume 的名称。
Provisioner	选择文件存储 CFS Turbo。
读写权限	文件存储仅支持多机读写。
是否指定 StorageClass	选择**不指定 StorageClass**。
CFS Turbo	此处选择已创建的 CFS Turbo。如果您没有合适的 CFS Turbo，请前往文件存储控制台新建，详情请参见 创建文件系统及挂载点 。
CFS Turbo根目录	根据 CFS Turbo 挂载点信息中的挂载根目录进行填写。
CFS Turbo子目录	根据 CFS Turbo 挂载点信息中的挂载子目录进行填写。

4. 单击**创建PersistentVolume**，完成创建。

步骤2：创建 PersistentVolumeClaim

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**存储** > **PersistentVolumeClaim**，在 PersistentVolumeClaim 页面单击**新建**。
3. 在**新建 PersistentVolumeClaim** 页面中，配置 PVC 关键参数。

配置项	描述
名称	填写 PersistentVolumeClaim 的名称。
命名空间	命名空间用来划分集群资源。此处选择 default。
Provisioner	选择文件存储 CFS Turbo。
读写权限	文件存储仅支持多机读写。
是否指定 StorageClass	选择**不指定 StorageClass**。 **说明：** • PVC 和 PV 会绑定在同一个 StorageClass 下。 • **不指定 StorageClass** 意味着该 PVC 对应的 StorageClass 取值为空，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。
StorageClass	选择上述步骤创建的 StorageClass。
是否指定 PersistVolume	按需指定 PersistentVolume。 **说明：** • 系统首先会筛选当前集群内是否存在符合绑定规则的 PV，若没有则根据 PVC 和所选 StorageClass 的参数动态创建 PV 与之绑定。 • 系统不允许在不指定 StorageClass 的情况下同时选择不指定 PersistVolume。 • 关于**不指定 PersistVolume** 的详细介绍，请参见 查看 PV 和 PVC 的绑定规则。

4. 单击**创建 PersistentVolumeClaim**。

步骤3：创建工作负载

1. 在集群列表中，单击集群 ID，进入集群详情页。
2. 选择左侧菜单栏中的**工作负载** > **Deployment**，在 Deployment 页面单击**新建**。
3. 在新建 Deployment 页面中，配置工作负载参数，参数详情请参见 创建 Deployment 。根据实际需求，数据卷选择**使用已有 PVC**，并选择上述已创建的 PVC。
4. 挂载到容器的指定路径后，单击**创建 Deployment**。

COS-CSI 说明

最近更新时间：2024-08-28 16:51:01

简介

组件介绍

Kubernetes-csi-tencentcloud COS 插件实现 CSI 的接口，可帮助您在容器集群中使用腾讯云对象存储 COS。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
csi-coslauncher	DaemonSet	-	kube-system
csi-cosplugin	DaemonSet	-	kube-system

使用场景

对象存储（Cloud Object Storage，COS）是腾讯云提供的一种存储海量文件的分布式存储服务，用户可通过网络随时存储和查看数据。腾讯云 COS 使所有用户都能使用具备高扩展性、低成本、可靠和安全的数据存储服务。
通过 COS-CSI 扩展组件，您可以快速的在容器集群中通过标准原生 Kubernetes 以 COSFS 的形式使用 COS，详情请参见 COSFS 工具介绍。

限制条件

- 支持 Kubernetes 1.10 以上版本的集群。
- Kubernetes 1.12 版本的集群需要增加 kubelet 配置：`--feature-gates=KubeletPluginsWatcher=false`。
- COSFS 本身限制，详情请参见 COSFS 局限性。
- 在 TKE 中使用 COS，需要在集群内安装该扩展组件，将占用一定的系统资源。

cos-csi 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机 /var/lib/kubelet 相关目录到容器来完成 volume 的 mount/umount，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
支持 lite 方式下的 cos 桶挂载	PersistentVolume	get/watch/list/update
	pod	get/create/delete/update
在 lite 挂载方式下存储相关 cos 配置	configmap	get/create/delete/update

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: csi-cos-tencentcloud
rules:
  - apiGroups: [""]
    resources: ["events", "persistentvolumes"]
    verbs: ["get", "watch", "update", "list"]
  - apiGroups: [""]
    resources: ["pods", "configmaps"]
    verbs: ["get", "create", "delete", "update"]
```

使用方法

安装 COS 扩展组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 COS。
5. 单击**完成**即可创建组件。

使用对象存储 COS

您可在 TKE 集群中为工作负载挂载对象存储，详情请参见 使用对象存储 COS 。

Cluster Autoscaler 说明

最近更新时间：2025-06-06 16:00:12

简介

组件介绍

Cluster Autoscaler（简称 CA）组件基于模拟调度算法为集群提供节点自动扩缩容能力，支持在资源不足时扩容新节点，在资源闲置时缩容旧节点。

说明：
- 该组件需搭配节点池一起使用，现已支持原生节点、普通节点，不支持超级节点。
- 使用该能力需确保节点池已开启**弹性伸缩**。

部署在集群内的 Kubernetes 对象

kubernetes 对象名称	类型	资源量	Namespaces
cluster-autoscaler	PodDisruptionBudget	-	kube-system
cluster-autoscaler	ServiceAccount	-	kube-system
cluster-autoscaler	Secret	-	kube-system
cluster-autoscaler	ClusterRole	-	-
cluster-autoscaler	ClusterRoleBinding	-	-
cluster-autoscaler	Role	-	kube-system
cluster-autoscaler	RoleBinding	-	kube-system
cluster-autoscaler	Service	-	kube-system
cluster-autoscaler	Deployment	0.5C1G（针对新建）	kube-system

使用场景

- 当集群中出现因资源不足而无法调度的实例（Pod）时，自动触发节点扩容，通过模拟调度选择合适的节点类型，为您减少人力成本。
- 当满足节点空闲缩容条件时，自动触发节点缩容，为您节约资源成本。

限制条件

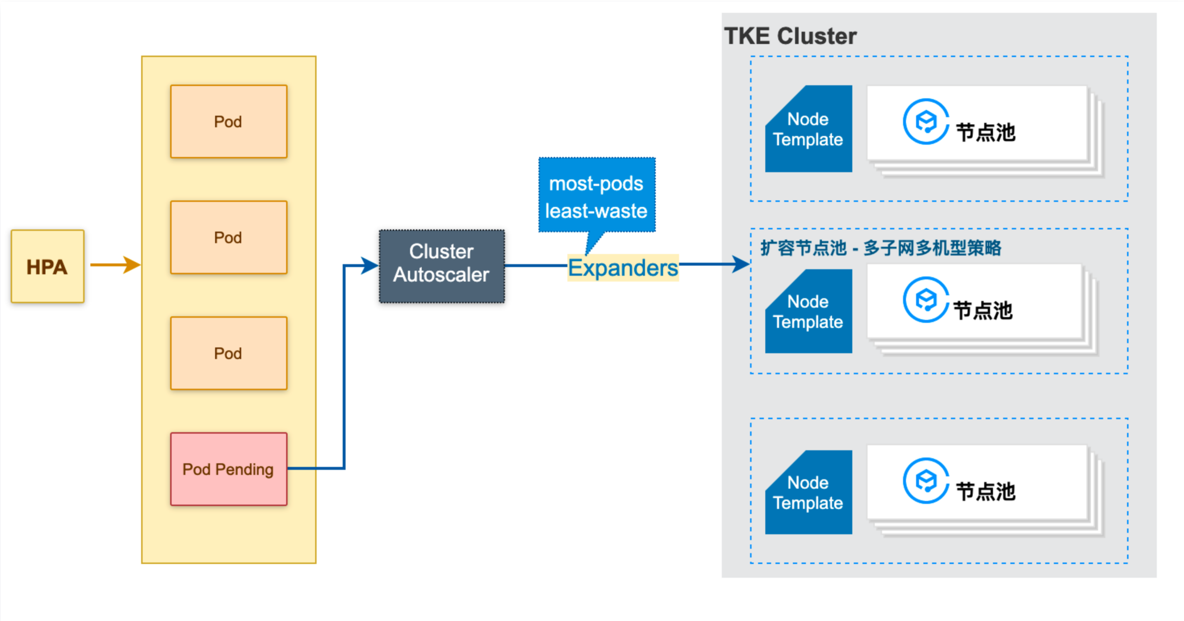
k8s 集群版本 >= 1.16

组件原理

扩容原理

1. 当集群中资源不足时（集群的计算/存储/网络等资源满足不了 Pod 的 Request /亲和性规则），CA 会监测到因无法调度而 Pending 的 Pod。
2. CA 根据每个节点池的节点模板进行调度判断，挑选合适的节点模板。

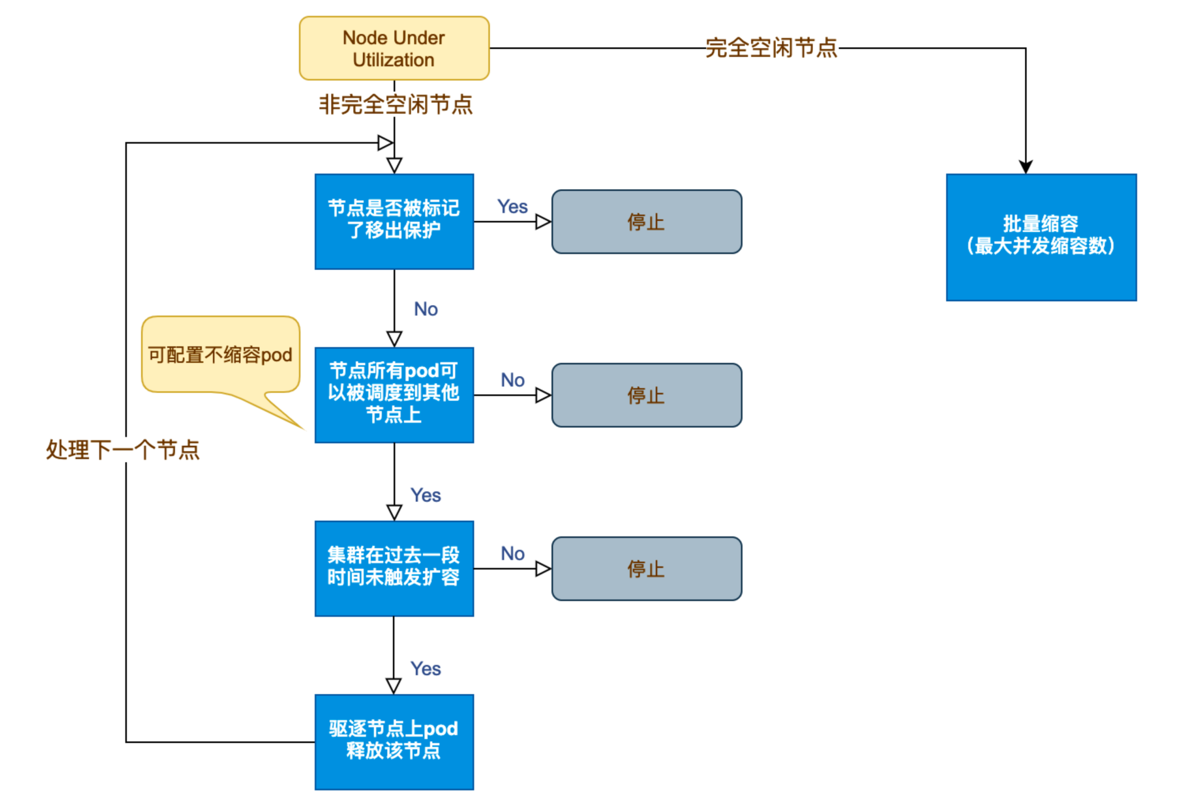
3. 若有多个模板合适，即有多个可扩的节点池备选，CA 会调用 expanders 从多个模板挑选最优模板并对对应节点池进行扩容。



缩容原理

1. CA（Cluster Autoscaler）监测到分配率（即 Request 值，取 CPU 分配率和 MEM 分配率的最大值）低于设定的节点。计算分配率时，可以设置 Daemonset 类型不计入 Pod 占用资源。
2. CA 判断集群的状态是否可以触发缩容，需要满足如下要求：
   - 节点空闲时长要求（默认10分钟）。
   - 集群扩容缓冲时间要求（默认10分钟）。
3. CA 判断该节点是否符合缩容条件。您可以按需设置以下不缩容条件（满足条件的节点不会被 CA 缩容）：
   - 含有本地存储的节点。
   - 含有 Kube-system namespace 下非 DaemonSet 管理的 Pod 的节点。
4. CA 驱逐节点上的 Pod 后释放/关机节点。
   - 完全空闲节点可并发缩容（可设置最大并发缩容数）。
   - 非完全空闲节点逐个缩容。

> **注意：**
> 包年包月类型的普通节点和原生节点不支持自动缩容。



参数说明

模块	功能项	参数值介绍
扩容	扩容算法	• **随机（默认）**：如果可扩容节点池有多个，从中任意选择一个节点池进行扩容。 • **most-pods**：如果可扩容节点池有多个，从中选择运行 Pod 数量最多的节点池进行扩容。 • **least-waste**：如果可扩容节点池有多个，从中选择一个资源浪费最少的节点池进行扩容。 • **priority**：如果可扩容节点池有多个，会按照您自定义的 ConfigMap（详情参考下方），选择优先级高的节点池进行扩容（该特性仅支持原生节点池，配置后普通节点池不参与扩容）。
缩容	最大并发缩容数	发起缩容时，同时支持缩容的节点数量。 **说明**：只缩容符合完全空闲的空节点；如果存在 Pod, 每次缩容最多一个节点。
	缩容条件	阈值：Pod 占用资源/可分配资源百分比小于 x%时开始判断缩容条件。
		触发时延：节点连续空闲 x 分钟后被缩容。
		静默时间：集群扩容 x 分钟开始判断缩容条件。
	不缩容节点	含有本地存储 Pod 的节点（本地存储包括 hostPath 和 emptyDir）。 含有 kube-system namespace 下非 DaemonSet 管理的 Pod 的节点。

自定义 ConfigMap 使用 priority 扩容算法

> **说明：**
> - 该特性仅支持原生节点池，对普通节点池不生效。
> - 优先级取值1~100，必须为正整数。
> - 一个节点池 ID 属于且只属于一个优先级。
> - 如果节点池 ID 没有配置在 ConfigMap 中，即使满足扩容需求，也会由于优先级未配置而不扩容。

示例如下：

```
apiVersion: v1
data:
  priorities: |-
    100:
      - np-15wmakan      #np-15wmakan（节点池id）优先级为100。
    50:
      - np-9r9rh7kp      #np-9r9rh7kp（节点池id）优先级为50。
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander      #名称需固定为 cluster-autoscaler-priority-expander
  namespace: kube-system
```

相关链接

- 创建原生节点
- 创建普通节点

DeScheduler 说明

最近更新时间：2025-07-01 14:42:22

温馨提示
感谢您对腾讯云容器服务 TKE 的认可与信赖。为了提供更优质的服务和更强大的调度能力，我们计划于2023年11月30日下线扩展组件里的调度器 DynamicScheduler 和 DeScheduler 。
请注意，自下线日期起，您将无法安装这两个新组件。对于已经安装的组件，不会受到影响，但将无法进行更新。
存量的 DeScheduler 调度器卸载方式请参见 组件卸载 。调度器卸载后，对集群本身调度不会产生影响，会默认使用原生的 Kubernetes 调度器。
为了减少这次变更对您的影响，我们推荐您使用腾讯云 TKE 原生节点专用调度器 。这个调度器完全覆盖了 DynamicScheduler 和 DeScheduler 的功能，无需依赖 Prometheus 监控，提供更加便捷稳定的调度增强能力，并且支持节点虚拟放大能力让节点装箱率超过100%。
我们对由此给您带来的不便表示歉意，并感谢您对我们产品的支持和理解。如果您在这个过程中遇到任何问题，或者需要我们的帮助，您可以咨询 在线客服 或 提交工单 来与我们联系。

简介

组件介绍

DeScheduler 是容器服务 TKE 基于 Kubernetes 原生社区 DeScheduler 实现的一个基于 Node 真实负载进行重调度的插件。在 TKE 集群中安装该插件后，该插件会和 kube-scheduler 协同生效，实时监控集群中高负载节点并驱逐低优先级 Pod。建议您搭配 TKE Dynamic Scheduler（动态调度器扩展组件）一起使用，多维度保障集群负载均衡。该插件依赖 Prometheus 监控组件以及相关规则配置，建议您安装插件之前仔细阅读 依赖部署，以免插件无法正常工作。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
descheduler	Deployment	每个实例 CPU:200m，Memory:200Mi，共1个实例	kube-system
descheduler	ClusterRole	-	kube-system
descheduler	ClusterRoleBinding	-	kube-system
descheduler	ServiceAccount	-	kube-system
descheduler-policy	ConfigMap	-	kube-system
probe-prometheus	ConfigMap	-	kube-system

使用场景

DeScheduler 通过重调度来解决集群现有节点上不合理的运行方式。社区版本 DeScheduler 中提出的策略基于 APIServer 中的数据实现，并未基于节点真实负载。因此可以增加对于节点的监控，基于真实负载进行重调度调整。
容器服务 TKE 自研的 ReduceHighLoadNode 策略依赖 Prometheus 和 node_exporter 监控数据，根据节点 CPU 利用率、内存利用率、网络 IO、system loadavg 等指标进行 Pod 驱逐重调度，防止出现节点极端负载的情况。DeScheduler 的 ReduceHighLoadNode 与 TKE 自研的 Dynamic Scheduler 基于节点真实负载进行调度的策略需配合使用。

注意事项

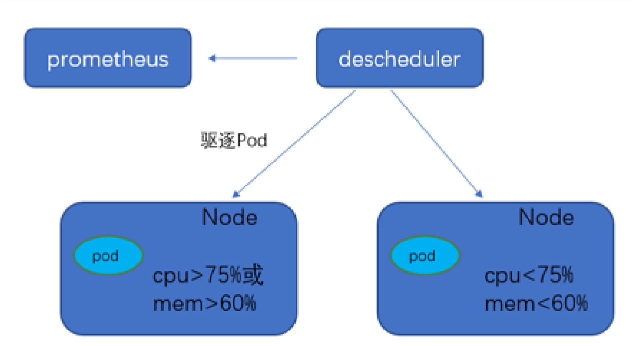
- Kubernetes 版本 ⩾ v1.10.x
- 在特定场景下，某些 Pod 会被重复调度到需要重调度的节点上，从而引发 Pod 被重复驱逐。此时可以根据实际场景改变 Pod 可调度的节点，或者将 Pod 标记为不可驱逐。
- 该组件已对接容器服务 TKE 的监控告警体系。
- 建议您为集群开启事件持久化，以便更好的监控组件异常以及故障定位。Descheduler 驱逐 Pod 时会产生对应事件，可根据 reason 为 “Descheduled” 类型的事件观察 Pod 是否被重复驱逐。
- 为避免 DeScheduler 驱逐关键的 Pod，设计的算法默认不驱逐 Pod。对于可以驱逐的 Pod，用户需要显示给判断 Pod 所属 workload。例如，StatefulSet、Deployment 等对象设置可驱逐 annotation。
- 驱逐的前提：为防止被驱逐的 Pod 没有资源使用，集群至少包含5个节点，需要有4个及以上的节点负载低于**目标利用率**才可以驱逐。
- 驱逐属于高危操作，请注意节点亲和性、污点相关配置，以及 Pod 本身对节点的要求选择，防止出现驱逐后无节点可调度情况。

- 驱逐大量 Pod，将导致服务不可用。Kubernetes 原生提供 PDB 对象用于防止驱逐接口造成的 workload 不可用 Pod 过多，但需要用户创建该 PDB 配置。容器服务 TKE 自研的 DeScheduler 组件加入了兜底措施，即调用驱逐接口前，判断 workload 准备的 Pod 数是否大于副本数一半，否则不调用驱逐接口。

组件原理

DeScheduler 基于 社区版本 DeScheduler 的重调度思想，定期扫描各个节点上的运行 Pod，发现不符合策略条件的进行驱逐以进行重调度。社区版本 DeScheduler 已提供部分策略，策略基于 APIServer 中的数据，例如 `LowNodeUtilization` 策略依赖的是 Pod 的 request 和 limit 数据，这类数据能够有效均衡集群资源分配、防止出现资源碎片。但社区策略缺少节点真实资源占用的支持，例如节点 A 和 B 分配出去的资源一致，由于 Pod 实际运行情况，CPU 消耗型和内存消耗型不同，峰谷期不同造成两个节点的负载差别巨大。

因此，腾讯云 TKE 推出 DeScheduler，底层依赖对节点真实负载的监控进行重调度。通过 Prometheus 拿到集群 Node 的负载统计信息，根据用户设置的负载阈值，定期执行策略里面的检查规则，驱逐高负载节点上的 Pod。



组件参数说明

Prometheus 数据查询地址

> **注意：**
> 为确保组件可以拉取到所需的监控数据、调度策略生效，请按照 依赖部署 > **Prometheus 文件配置**步骤配置监控数据采集规则。

- 如果使用自建 Prometheus，直接填入数据查询 URL（HTTP/HTTPS）即可。
- 如果使用托管 Prometheus，选择托管实例 ID 即可，系统会自动解析实例对应的数据查询 URL。

利用率阈值和目标利用率

> **注意：**
> 负载阈值参数已设置默认值，如您无额外需求，可直接采用。

过去5分钟内，节点的 CPU 平均利用率或者内存平均使用率超过设定阈值，DeScheduler 会判断节点为高负载节点，执行 Pod 驱逐逻辑，并尽量通过 Pod 重调度使节点负载降到目标利用率以下。

操作步骤

依赖部署

DeScheduler 组件依赖于 Node 当前和过去一段时间的真实负载情况来进行调度决策，需要通过 Prometheus 等监控组件获取系统 Node 真实负载信息。在使用 DeScheduler 组件之前，您可以采用自建 Prometheus 监控或采用 TKE 云原生监控。

自建 Prometheus 监控服务

部署 node-exporter 和 Prometheus

通过 node-exporter 实现对于 Node 指标的监控，您可按需部署 node-exporter 和 Prometheus。

聚合规则配置

在 node-exporter 获取节点监控数据后，需要通过 Prometheus 对原始的 node-exporter 中采集数据进行聚合计算。为获取 DeScheduler 所需要的 `cpu_usage_avg_5m` 、 `mem_usage_avg_5m` 等指标，需要在 Prometheus 的 rules 规则中进行配置。示例如下：

```
groups:
  - name: cpu_mem_usage_active
    interval: 30s
    rules:
    - record: mem_usage_active
      expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: cpu-usage-1m
    interval: 1m
    rules:
    - record: cpu_usage_avg_5m
      expr: 100 - (avg by (instance) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
  - name: mem-usage-1m
    interval: 1m
    rules:
    - record: mem_usage_avg_5m
      expr: avg_over_time(mem_usage_active[5m])
```

注意：
当您使用 TKE 提供的 DynamicScheduler 时，需在 Prometheus 配置获取 Node 监控数据的聚合规则。DynamicScheduler 聚合规则与 DeScheduler 聚合规则有部分重合，但并不完全一样，请您在配置规则时不要互相覆盖。同时使用 DynamicScheduler 和 DeScheduler 时应该配置如下规则：

```
groups:
  - name: cpu_mem_usage_active
    interval: 30s
    rules:
    - record: mem_usage_active
      expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: mem-usage-1m
    interval: 1m
    rules:
    - record: mem_usage_avg_5m
      expr: avg_over_time(mem_usage_active[5m])
  - name: mem-usage-5m
    interval: 5m
    rules:
    - record: mem_usage_max_avg_1h
      expr: max_over_time(mem_usage_avg_5m[1h])
    - record: mem_usage_max_avg_1d
      expr: max_over_time(mem_usage_avg_5m[1d])
  - name: cpu-usage-1m
    interval: 1m
    rules:
    - record: cpu_usage_avg_5m
      expr: 100 - (avg by (instance) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
  - name: cpu-usage-5m
    interval: 5m
    rules:
    - record: cpu_usage_max_avg_1h
      expr: max_over_time(cpu_usage_avg_5m[1h])
    - record: cpu_usage_max_avg_1d
      expr: max_over_time(cpu_usage_avg_5m[1d])
```

Prometheus 文件配置

1. 上述定义了 DeScheduler 所需要的指标计算的 rules，需要将 rules 配置到 Prometheus 中，参考一般的 Prometheus 配置文件。示例如下：

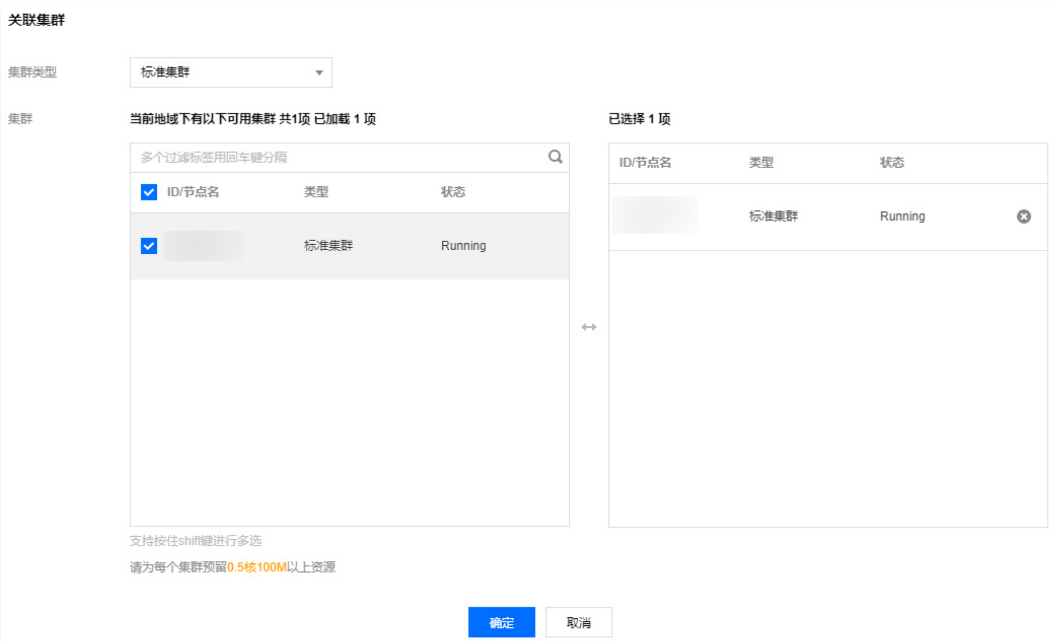
```
global:
   evaluation_interval: 30s
   scrape_interval: 30s
   external_labels:
rule_files:
- /etc/prometheus/rules/*.yml # /etc/prometheus/rules/*.yml 是定义的 rules 文件
```

2. 将 rules 配置复制到一个文件（例如 de-scheduler.yaml），文件放到上述 Prometheus 容器的 `/etc/prometheus/rules/` 下。
3. 重新加载 Prometheus server，即可从 Prometheus 中获取到动态调度器需要的指标。

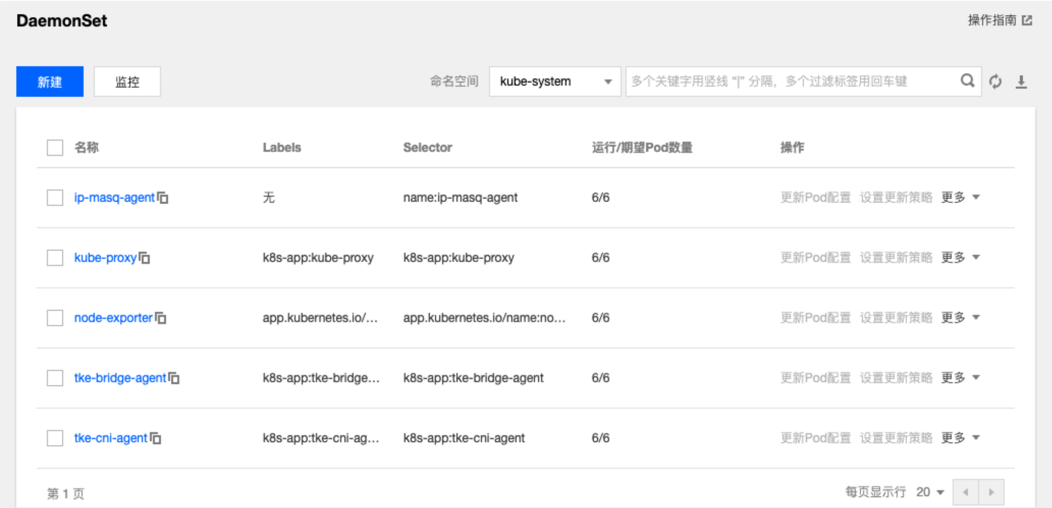
> **说明：**
> 通常情况下，上述 Prometheus 配置文件和 rules 配置文件都是通过 ConfigMap 存储，再挂载到 Prometheus server 容器，因此修改相应的 configmap 即可。

Prometheus 监控服务

1. 登录容器服务控制台，在左侧菜单栏中选择 Prometheus 监控，进入 Prometheus 监控页面。
2. 创建与 Cluster 处于同一 VPC 下的 Prometheus 实例，并 关联集群。如下图所示：



3. 与原生托管集群关联后，可以在用户集群查看到每个节点都已安装 node-exporter。如下图所示：



4. 设置 Prometheus 聚合规则，具体规则内容与上述 自建 Prometheus 监控服务 中的“聚合规则配置”相同。规则保存后立即生效，无需重新加载 server。

安装组件

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在**组件列表**页面中单击**新建**。
4. 在**新建组件**页面中勾选 DeScheduler（重调度器），单击**参数配置**，按照 参数说明 填写组件所需参数。
5. 单击**完成**即可创建组件。安装成功后，DeScheduler 即可正常运行，无需进行额外配置。
6. 若您需要驱逐 workload（例如 StatefulSet、Deployment 等对象），可以设置 annotation 如下：

```
descheduler.alpha.kubernetes.io/evictable: 'true'
```

DNSAutoscaler 说明

最近更新时间：2024-08-26 17:25:41

简介

组件介绍

DNSAutoscaler 是 DNS 自动水平伸缩组件，可通过一个 deployment 获取集群的节点数和核数，根据预设的伸缩策略，自动水平伸缩 DNS 的副本数。目前的伸缩模式分为两种，分别是 Linear 线性模式 和 Ladder 阶梯模式 。

Linear Mode

ConfigMap 配置示例如下：

```
data:
  linear: |-
    {
      "coresPerReplica": 2,
      "nodesPerReplica": 1,
      "min": 1,
      "max": 100,
      "preventSinglePointFailure": true
    }
```

目标副本计算公式：
replicas = max(ceil(cores × 1/coresPerReplica) , ceil(nodes × 1/nodesPerReplica))
replicas = min(replicas, max)
replicas = max(replicas, min)

Ladder Mode

ConfigMap 配置示例如下：

```
data:
  ladder: |-
    {
      "coresToReplicas":
      [
        [ 1, 1 ],
        [ 64, 3 ],
        [ 512, 5 ],
        [ 1024, 7 ],
        [ 2048, 10 ],
        [ 4096, 15 ]
      ],
      "nodesToReplicas":
      [
        [ 1, 1 ],
        [ 2, 2 ]
      ]
    }
```

目标副本计算：
假设 100nodes/400cores 的集群中，按照上述配置，nodesToReplicas 取2（100node 大于 2node，向下取整），coresToReplicas 取3（400core 落在 64core 和 512core 之间，向下取整），二者取较大值3，最终 replica 为3。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace

tke-dns-autoscaler	Deployment	每节点20mCPU，10Mi内存	kube-system
dns-autoscaler	ConfigMap	-	kube-system
tke-dns-autoscale	ServiceAccount	-	kube-system
tke-dns-autoscaler	ClusterRole	-	kube-system
tke-dns-autoscaler	ClusterRoleBinding	-	kube-system

限制条件

- 仅在 1.8 版本以上的 Kubernetes 集群支持。
- 集群中的 dns server 的工作负载为 deployment/coredns。

特别说明

CoreDNS 的水平伸缩可能导致部分副本在一段时间内不可用，强烈建议安装该组件前，进行相关的优化配置，最大程度保证 DNS 服务的可用性，详情可参见 配置 CoreDNS 平滑升级 。

组件权限说明

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
需要监听集群内 node 资源的变化。	node	list/watch
修改 deployment 部署的 coredns 副本数。	replicationcontrollers/scale、deployments/scale 和 replicasets/scale	get/update
获取 configmap 中参数配置。在没有配置参数的情况下，会创建默认参数的 configmap。	configmap	get/create

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tke-dns-autoscaler
rules:
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - replicationcontrollers/scale
    verbs:
      - get
      - update
  - apiGroups:
      - extensions
      - apps
    resources:
```

```
      - deployments/scale
      - replicasets/scale
    verbs:
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - create
```

使用方法

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件**页面中勾选 DNSAutoscaler。该组件默认伸缩配置策略如下：

```
data:
  ladder: |-
    {
      "coresToReplicas":
      [
        [ 1, 1 ],
        [ 128, 3 ],
        [ 512, 4 ]
      ],
      "nodesToReplicas":
      [
        [ 1, 1 ],
        [ 2, 2 ]
      ]
    }
```

扩展组件创建成功后，可以通过修改 kube-system 命名空间下的 `configmap/tke-dns-autoscaler` 来变更配置。详细配置请参见 官方文档。
5. 单击**完成**即可创建组件。

DynamicScheduler 说明

最近更新时间：2024-10-29 16:09:22

温馨提示

感谢您对腾讯云容器服务 TKE 的认可与信赖。为了提供更优质的服务和更强大的调度能力，我们计划于2023年11月30日下线扩展组件里的调度器 DynamicScheduler 和 DeScheduler。

请注意，自下线日期起，您将无法安装这两个新组件。对于已经安装的组件，不会受到影响，但将无法进行更新。

存量的 Dynamic Scheduler 调度器卸载方式请参见 组件卸载。调度器卸载后，对集群本身调度不会产生影响，会默认使用原生的 Kubernetes 调度器。

为了减少这次变更对您的影响，我们推荐您使用腾讯云 TKE 原生节点专用调度器。这个调度器完全覆盖了 DynamicScheduler 和 DeScheduler 的功能，无需依赖 Prometheus 监控，提供更加便捷稳定的调度增强能力，并且支持节点虚拟放大能力让节点装箱率超过100%。

我们对由此给您带来的不便表示歉意，并感谢您对我们产品的支持和理解。如果您在这个过程中遇到任何问题，或者需要我们的帮助，您可以咨询 在线客服 或 提交工单 来与我们联系。

简介

组件介绍

Dynamic Scheduler 是容器服务 TKE 基于 Kubernetes 原生 Kube-scheduler Extender 机制实现的动态调度器插件，可基于 Node 真实负载进行预选和优选。在 TKE 集群中安装该插件后，该插件将与 Kube-scheduler 协同生效，有效避免原生调度器基于 request 和 limit 调度机制带来的节点负载不均问题。

该组件依赖 Prometheus 监控组件以及相关规则配置，可参见本文 依赖部署 进行操作，避免遇到插件无法正常工作的情况。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
node-annotator	Deployment	每个实例 CPU:100m，Memory:100Mi，共1个实例	kube-system
dynamic-scheduler	Deployment	每个实例 CPU:400m，Memory:200Mi，共3个实例	kube-system
dynamic-scheduler	Service	-	kube-system
node-annotator	ClusterRole	-	kube-system
node-annotator	ClusterRoleBinding	-	kube-system
node-annotator	ServiceAccount	-	kube-system
dynamic-scheduler-policy	ConfigMap	-	kube-system
restart-kube-scheduler	ConfigMap	-	kube-system
probe-prometheus	ConfigMap	-	kube-system

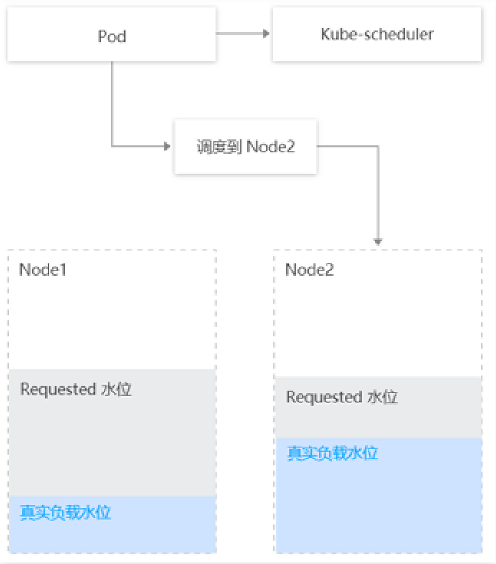
应用场景

集群负载不均

Kubernetes 原生调度器大部分基于 Pod Request 资源进行调度，并不具备根据 Node 当前和过去一段时间的真实负载情况进行相关调度的决策，因此可能会导致如下问题：

集群内部分节点的剩余可调度资源较多（根据节点上运行的 Pod 的 request 和 limit 计算出的值）但真实负载却比较高，而另外节点的剩余可调度资源比较少但真实负载却比较低，此时 Kube-scheduler 会优先将 Pod 调度到剩余资源比较多的节点上（根据 LeastRequestedPriority 策略）。

如下图所示，Kube-Scheduler 会将 Pod 调度到 Node2 上，但明显调度到 Node1（真实负载水位更低）是更优的选择。



防止调度热点

为防止低负载的节点被持续调度 Pod，Dynamic Scheduler 支持设置防调度热点策略（统计节点过去几分钟调度 Pod 的数量，并相应减小节点在优选阶段的评分）。

当前采取策略如下：

- 如果节点在过去1分钟调度了超过2个 Pod，则优选评分减去1分。
- 如果节点在过去5分钟调度了超过5个 Pod，则优选评分减去1分。

风险控制

- 该组件已对接 TKE 的监控告警体系。
- 推荐您为集群开启事件持久化，以便更好的监控组件异常以及故障定位。
- 该组件卸载后，只会删除动态调度器有关调度逻辑，不会对原生 Kube-Scheduler 的调度功能有任何影响。

限制条件

- TKE 版本建议 ⩾ v1.10.x
- 如果需要升级 Kubernetes master 版本：
  - 对于托管集群无需再次设置本插件。
  - 对于独立集群，master 版本升级会重置 master 上所有组件的配置，从而影响到 Dynamic Scheduler 插件作为 Scheduler Extender 的配置，因此 Dynamic Scheduler 插件需要卸载后再重新安装。

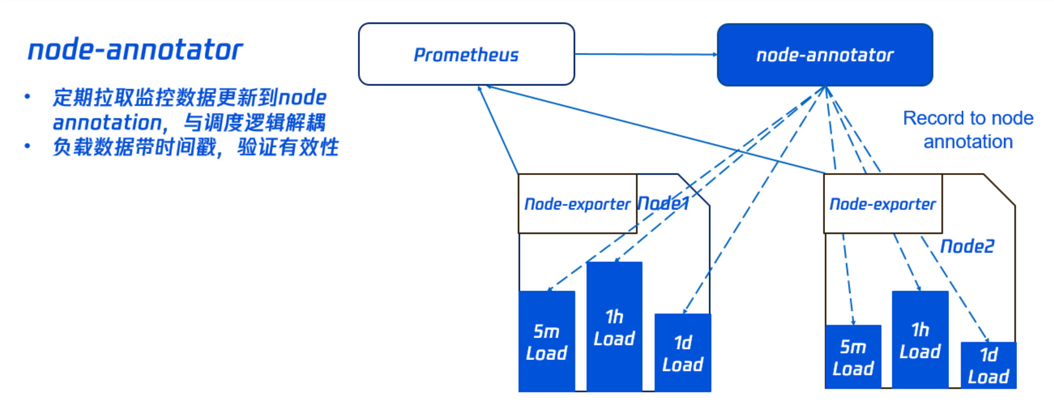
组件原理

动态调度器基于 scheduler extender 扩展机制，从 Prometheus 监控数据中获取节点负载数据，开发基于节点实际负载的调度策略，在调度预选和优选阶段进行干预，优先将 Pod 调度到低负载节点上。该组件由 node-annotator 和 Dynamic-scheduler 构成。

node-annotator

node-annotator 组件负责定期从监控中拉取节点负载 metric，同步到节点的 annotation。如下图所示：

> **注意：**
> 删除组件后，node-annotator 生成的 annotation 并不会被自动清除。您可根据需要手动清除。

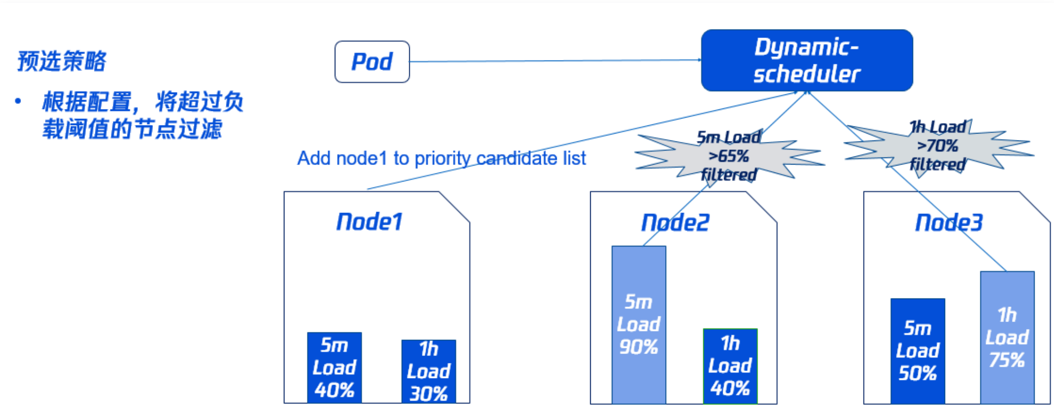


Dynamic-scheduler

Dynamic-scheduler 是一个 scheduler-extender，根据 node annotation 负载数据，在节点预选和优选中进行过滤和评分计算。

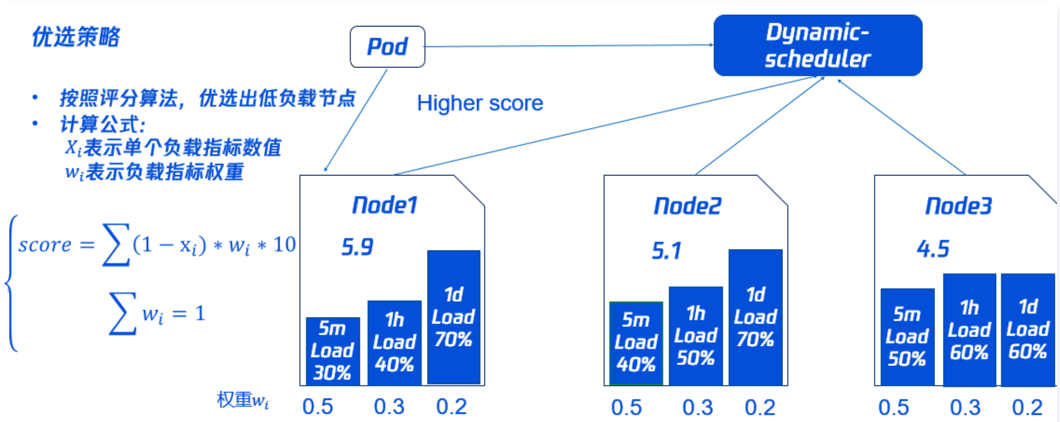
预选策略

为了避免 Pod 调度到高负载的 Node 上，需要先通过预选过滤部分高负载的 Node（其中过滤策略和比例可以动态配置，具体请参见本文 组件参数说明 ）。
如下图所示，Node2 过去5分钟的负载，Node3 过去1小时的负载均超过对应的阈值，因此不会参与接下来的优选阶段。



优选策略

同时为了使集群各节点的负载尽量均衡，Dynamic-scheduler 会根据 Node 负载数据进行打分，负载越低打分越高。
如下图所示，Node1 的打分最高将会被优先调度（其中打分策略和权重可以动态配置，具体请参见本文 组件参数说明 ）。



组件参数说明

Prometheus 数据查询地址

注意：

- 为确保组件可以拉取到所需的监控数据、调度策略生效，请按照 依赖部署 > Prometheus 规则配置 步骤配置监控数据采集规则。
- 预选和优选参数已设置默认值，如您无额外需求，可直接采用。

- 如果使用自建 Prometheus，直接填入数据查询 URL（HTTP/HTTPS）即可。
- 如果使用托管 Prometheus，选择托管实例 ID 即可，系统会自动解析实例对应的数据查询 URL。

预选参数

预选参数默认值	说明
5分钟平均 CPU 利用率阈值	节点过去5分钟**平均** CPU 利用率超过设定阈值，不会调度 Pod 到该节点上。
1小时最大 CPU 利用率阈值	节点过去1小时**最大** CPU 利用率超过设定阈值，不会调度 Pod 到该节点上。
5分钟平均**内存**利用率阈值	节点过去5分钟**平均**内存利用率超过设定阈值，不会调度 Pod 到该节点上。
1小时最大**内存**利用率阈值	节点过去1小时**最大**内存利用率超过设定阈值，不会调度 Pod 到该节点上。

优选参数

优选参数默认值	说明
5分钟平均 CPU 利用率权重	该权重越大，过去5分钟节点**平均** CPU 利用率对节点的评分影响越大。
1小时最大 CPU 利用率权重	该权重越大，过去1小时节点**最大** CPU 利用率对节点的评分影响越大。
1天最大 CPU 利用率权重	该权重越大，过去1天内节点**最大** CPU 利用率对节点的评分影响越大。
5分钟平均**内存**利用率权重	该权重越大，过去5分钟节点**平均**内存利用率对节点的评分影响越大。
1小时最大**内存**利用率权重	该权重越大，过去1小时节点**最大**内存利用率对节点的评分影响越大。
1天最大**内存**利用率权重	该权重越大，过去1天内节点**最大**内存利用率对节点的评分影响越大。

操作步骤

依赖部署

Dynamic Scheduler 动态调度器依赖于 Node 当前和过去一段时间的真实负载情况来进行调度决策，需通过 Prometheus 等监控组件获取系统 Node 真实负载信息。在使用动态调度器之前，需要部署 Prometheus 等监控组件。在容器服务 TKE 中，您可按需选择采用自建的 Prometheus 监控服务或采用 TKE 推出的云原生监控。

自建Prometheus监控服务

部署 node-exporter 和 prometheus

通过 node-exporter 实现对 Node 指标的监控，用户可以根据业务需求部署 node-exporter 和 prometheus。

聚合规则配置

在 node-exporter 获取节点监控数据后，需要通过 Prometheus 对原始的 node-exporter 采集数据进行聚合计算。为了获取动态调度器中需要的 `cpu_usage_avg_5m`、`cpu_usage_max_avg_1h`、`cpu_usage_max_avg_1d`、`mem_usage_avg_5m`、`mem_usage_max _avg_1h`、`mem_usage_max_avg_1d` 等指标，需要在 Prometheus 的 rules 规则进行如下配置：

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-record
spec:
  groups:
    - name: cpu_mem_usage_active
      interval: 30s
```

```
    rules:
      - record: cpu_usage_active
        expr: 100 - (avg by (instance) (irate(node_cpu_seconds_total{mode="idle"}[30s])) * 100)
      - record: mem_usage_active
        expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
  - name: cpu-usage-5m
    interval: 5m
    rules:
      - record: cpu_usage_max_avg_1h
        expr: max_over_time(cpu_usage_avg_5m[1h])
      - record: cpu_usage_max_avg_1d
        expr: max_over_time(cpu_usage_avg_5m[1d])
  - name: cpu-usage-1m
    interval: 1m
    rules:
      - record: cpu_usage_avg_5m
        expr: avg_over_time(cpu_usage_active[5m])
  - name: mem-usage-5m
    interval: 5m
    rules:
      - record: mem_usage_max_avg_1h
        expr: max_over_time(mem_usage_avg_5m[1h])
      - record: mem_usage_max_avg_1d
        expr: max_over_time(mem_usage_avg_5m[1d])
  - name: mem-usage-1m
    interval: 1m
    rules:
      - record: mem_usage_avg_5m
        expr: avg_over_time(mem_usage_active[5m])
```

Prometheus 文件配置

1. 上述定义了动态调度器所需要的指标计算的 rules，需要将 rules 配置到 Prometheus 中，参考一般的 Prometheus 配置文件。示例如下：

```
global:
  evaluation_interval: 30s
  scrape_interval: 30s
  external_labels:
rule_files:
- /etc/prometheus/rules/*.yml # /etc/prometheus/rules/*.yml就是定义的rules文件
```

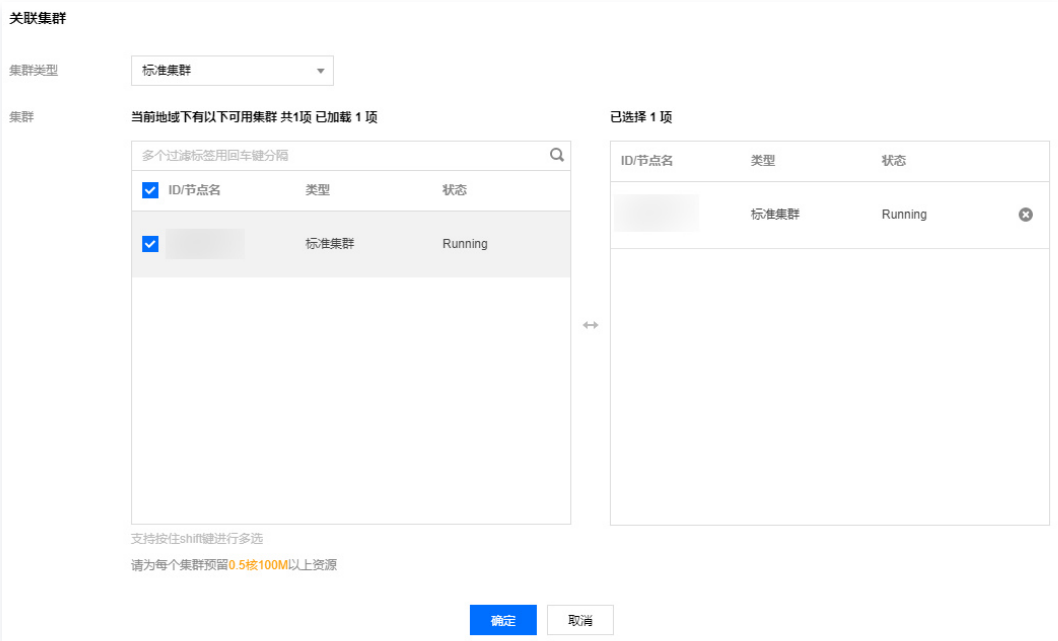
2. 将 rules 配置复制到一个文件（例如 dynamic-scheduler.yaml），文件放到上述 prometheus 容器的 `/etc/prometheus/rules/` 目录下。
3. 加载 Prometheus server，即可从 Prometheus 获取到动态调度器需要的指标。

> **说明：**
> 通常情况下，上述 Prometheus 配置文件和 rules 配置文件都是通过 configmap 存储，再挂载到 Prometheus server 容器，因此修改相应的 configmap 即可。

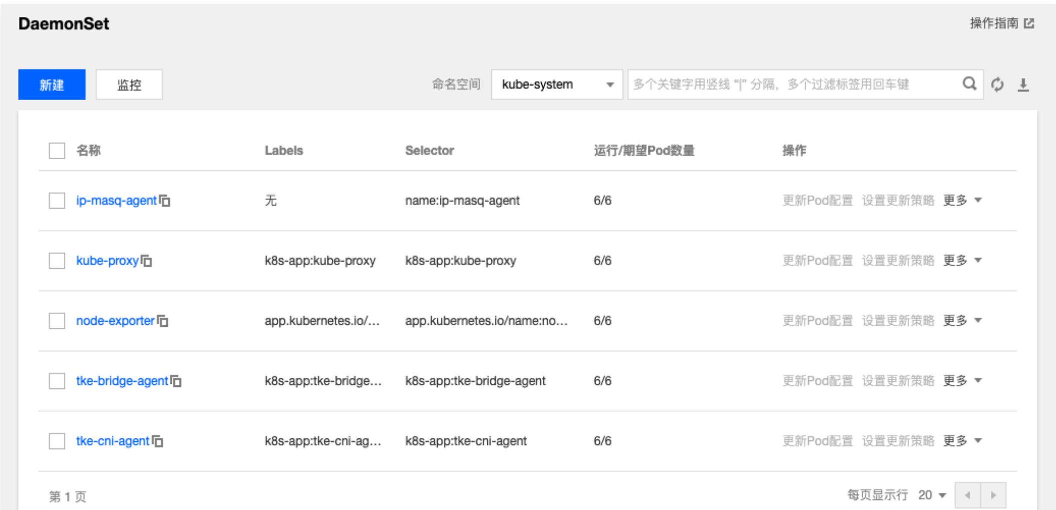
Prometheus 监控服务

1. 登录容器服务控制台，在左侧导航中选择 Prometheus 监控。

2. 在 Prometheus 监控页面，单击**新建**，创建与 Cluster 处于同一 VPC 下的 Prometheus 实例，并 关联集群。如下图所示：



3. 与原生托管集群关联后，可以在用户集群查看到每个节点都已安装 node-exporter。如下图所示：



4. 设置 Prometheus 聚合规则。代码示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-record
spec:
  groups:
    - name: cpu_mem_usage_active
      interval: 30s
      rules:
        - record: cpu_usage_active
          expr: 100 - (avg by (instance) (irate(node_cpu_seconds_total{mode="idle"}[30s])) * 100)
        - record: mem_usage_active
          expr: 100*(1-node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes)
    - name: cpu-usage-5m
      interval: 5m
      rules:
        - record: cpu_usage_max_avg_1h
```

```
            expr: max_over_time(cpu_usage_avg_5m[1h])
          - record: cpu_usage_max_avg_1d
            expr: max_over_time(cpu_usage_avg_5m[1d])
      - name: cpu-usage-1m
        interval: 1m
        rules:
          - record: cpu_usage_avg_5m
            expr: avg_over_time(cpu_usage_active[5m])
      - name: mem-usage-5m
        interval: 5m
        rules:
          - record: mem_usage_max_avg_1h
            expr: max_over_time(mem_usage_avg_5m[1h])
          - record: mem_usage_max_avg_1d
            expr: max_over_time(mem_usage_avg_5m[1d])
      - name: mem-usage-1m
        interval: 1m
        rules:
          - record: mem_usage_avg_5m
            expr: avg_over_time(mem_usage_active[5m])
```

安装组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 **DynamicScheduler**（动态调度器插件）。
5. 单击**参数配置**，按照 参数说明 填写组件所需参数。
6. 单击**完成**即可创建组件。安装成功后，Dynamic Scheduler 即可正常运行，无需进行额外配置。

eNP（ebpf Network Policies）使用说明

最近更新时间：2025-02-28 15:51:43

简介

Network Policies

如果您希望在 IP 地址或端口层面控制 TCP、UDP 和 SCTP 等协议的网络流量，可以为集群中的特定应用使用 Kubernetes 网络策略（NetworkPolicy）。NetworkPolicy 是一种以应用为中心的策略机制，用于精细化地定义 Pod 如何与网络中的其他实体进行通信，特别适用于与 Pod 相关的网络连接场景。

您可以通过以下三种标识符来指定允许与 Pod 通信的实体：

1. 其他已被允许的 Pod（注意：Pod 无法阻止对自身的访问）。
2. 被允许的命名空间。
3. IP 块（注意：无论 Pod 或节点的 IP 地址是什么，Pod 与其所在节点之间的通信总是被允许）。

在定义基于 Pod 或命名空间的 NetworkPolicy 时，可以使用 标签选择器（Label Selector）来决定哪些流量可进入或离开符合该选择器条件的 Pod；而对于基于 IP 地址的 NetworkPolicy，则通过指定 IP 块（CIDR 范围）来进行相应的流量控制。

eNP

eNP（ebpf Network Policies）组件是一种基于 ebpf 技术的 NetworkPolicy 实现。它通过在 Pod 所在宿主机的 veth 的 tc 上挂载 ebpf 程序，在网卡处理报文时，根据五元组（源地址、源端口、目的地址、目的端口以及传输协议协议）去匹配 NetworkPolicy 的规则，判断该报文是放通还是丢弃。

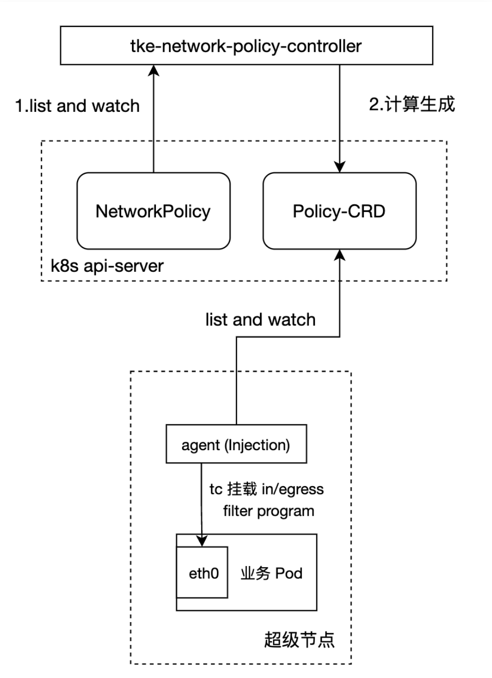
架构

限制

1. 支持的 OS 版本为 TencentOS Server 3.1 以及 TencentOS Server 3.2。
2. Kubernetes 1.20 及以上版本。
3. 网络插件为共享网卡 VPC-CNI。
4. 当前仅支持超级节点。
5. 仅支持 IPv4 集群，暂不支持 IPv4/IPv6 双栈集群。

组件架构

eNP 组件架构示意图如下：



1. eNP 组件引入一个 CRD（Custom Resource Definition），用来存储 NetworkPolicy 关联的实体。

2. tke-network-policy-controller 以 deployment 形式部署在控制面，其 list and watch NetworkPolicy 资源并为每个 Pod 计算生成 Policy-CRD。
3. tke-network-policy-agent 通过 daemonset 注入 的方式在每个业务 Pod 的 network namespace 内的默认网卡（eth0）上挂载 filter ebpf 程序。
4. agent 根据 Policy-CRD 在 ebpf map 中维护放通规则，挂载的 ebpf 程序根据放通 ebpf map 中的规则去放通或者丢弃某个报文。

eNP 组件 e2e 测试情况

运行 Kubernetes 社区的 NetworkPolicy 的 e2e 测试（release-1.26），该版本的测试项目来自两个测试集合，不同集合的网络模型不同，但两个集合之间的 测试项目有重叠，这里将 network_policy.go 内的测试项目标记为 network_policy，network_legacy.go 内的测试项目标记为 network_legacy，通过的 case 为：

编号	Feature	来源
1	should support a 'default-deny-ingress' policy [Feature:NetworkPolicy]	network_legacy
2	should support a 'default-deny-all' policy [Feature:NetworkPolicy]	network_legacy
3	should enforce policy to allow traffic from pods within server namespace based on PodSelector [Feature:NetworkPolicy]	network_legacy
4	should enforce policy to allow traffic only from a different namespace, based on NamespaceSelector [Feature:NetworkPolicy]	network_legacy
5	should enforce policy based on PodSelector with MatchExpressions[Feature:NetworkPolicy]	network_legacy
6	should enforce policy based on NamespaceSelector with MatchExpressions[Feature:NetworkPolicy]	network_legacy
7	should enforce policy based on PodSelector or NamespaceSelector [Feature:NetworkPolicy]	network_legacy
8	should enforce policy based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_legacy
9	should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_legacy
10	should enforce policy based on Ports [Feature:NetworkPolicy]	network_legacy
11	should enforce multiple, stacked policies with overlapping podSelectors [Feature:NetworkPolicy]	network_legacy
12	should support allow-all policy [Feature:NetworkPolicy]	network_legacy
13	should allow ingress access on one named port [Feature:NetworkPolicy]	network_legacy
14	should allow ingress access from namespace on one named port [Feature:NetworkPolicy]	network_legacy
15	should allow egress access on one named port [Feature:NetworkPolicy]	network_legacy
16	should enforce updated policy [Feature:NetworkPolicy]	network_legacy
17	should allow ingress access from updated namespace [Feature:NetworkPolicy]	network_legacy

18	should allow ingress access from updated pod [Feature:NetworkPolicy]	network_legacy
19	should deny ingress access to updated pod [Feature:NetworkPolicy]	network_legacy
20	should work with Ingress,Egress specified together [Feature:NetworkPolicy]	network_legacy
21	should enforce egress policy allowing traffic to a server in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_legacy
22	should enforce multiple ingress policies with ingress allow-all policy taking precedence [Feature:NetworkPolicy]	network_legacy
23	should enforce multiple egress policies with egress allow-all policy taking precedence [Feature:NetworkPolicy]	network_legacy
24	should stop enforcing policies after they are deleted [Feature:NetworkPolicy]	network_legacy
25	should enforce policies to check ingress and egress policies can be controlled independently based on PodSelector [Feature:NetworkPolicy]	network_legacy
26	should not allow access by TCP when a policy specifies only SCTP [Feature:NetworkPolicy]	network_legacy
27	should support a 'default-deny' policy [Feature:NetworkPolicy]	network_legacy
28	should enforce policy based on Ports [Feature:NetworkPolicy]	network_legacy
29	should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_legacy
30	should support a 'default-deny-ingress' policy [Feature:NetworkPolicy]	network_policy
31	should support a 'default-deny-all' policy [Feature:NetworkPolicy]	network_policy
32	should enforce policy to allow traffic from pods within server namespace based on PodSelector [Feature:NetworkPolicy]	network_policy
33	should enforce policy to allow ingress traffic for a target [Feature:NetworkPolicy]	network_policy
34	should enforce policy to allow ingress traffic from pods in all namespaces [Feature:NetworkPolicy]	network_policy
35	should enforce policy to allow traffic only from a different namespace, based on NamespaceSelector [Feature:NetworkPolicy]	network_policy
36	should enforce policy based on PodSelector with MatchExpressions[Feature:NetworkPolicy]	network_policy
37	should enforce policy based on NamespaceSelector with MatchExpressions[Feature:NetworkPolicy]	network_policy
38	should enforce policy based on PodSelector or NamespaceSelector [Feature:NetworkPolicy]	network_policy
39	should enforce policy based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_policy

40	should enforce policy based on Multiple PodSelectors and NamespaceSelectors [Feature:NetworkPolicy]	network_policy
41	should enforce policy based on any PodSelectors [Feature:NetworkPolicy]	network_policy
42	should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_policy
43	should enforce policy based on Ports [Feature:NetworkPolicy]	network_policy
44	should enforce multiple, stacked policies with overlapping podSelectors [Feature:NetworkPolicy]	network_policy
45	should support allow-all policy [Feature:NetworkPolicy]	network_policy
46	should allow ingress access on one named port [Feature:NetworkPolicy]	network_policy
47	should allow ingress access from namespace on one named port [Feature:NetworkPolicy]	network_policy
48	should enforce updated policy [Feature:NetworkPolicy]	network_policy
49	should allow ingress access from updated namespace [Feature:NetworkPolicy]	network_policy
50	should allow ingress access from updated pod [Feature:NetworkPolicy]	network_policy
51	should deny ingress from pods on other namespaces [Feature:NetworkPolicy]	network_policy
52	should deny ingress access to updated pod [Feature:NetworkPolicy]	network_policy
53	should deny egress from pods based on PodSelector [Feature:NetworkPolicy]	network_policy
54	should deny egress from all pods in a namespace [Feature:NetworkPolicy]	network_policy
55	should work with Ingress, Egress specified together [Feature:NetworkPolicy]	network_policy
56	should support denying of egress traffic on the client side (even if the server explicitly allows this traffic) [Feature:NetworkPolicy]	network_policy
57	should enforce egress policy allowing traffic to a server in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_policy
58	should enforce ingress policy allowing any port traffic to a server on a specific protocol [Feature:NetworkPolicy] [Feature:UDP]	network_policy
59	should enforce multiple ingress policies with ingress allow-all policy taking precedence [Feature:NetworkPolicy]	network_policy
60	should enforce multiple egress policies with egress allow-all policy taking precedence [Feature:NetworkPolicy]	network_policy
61	should stop enforcing policies after they are deleted [Feature:NetworkPolicy]	network_policy

62	should enforce policies to check ingress and egress policies can be controlled independently based on PodSelector [Feature:NetworkPolicy]	network_policy
63	should not mistakenly treat 'protocol: SCTP' as 'protocol: TCP', even if the plugin doesn't support SCTP [Feature:NetworkPolicy]	network_policy
64	should properly isolate pods that are selected by a policy allowing SCTP, even if the plugin doesn't support SCTP [Feature:NetworkPolicy]	network_policy
65	should not allow access by TCP when a policy specifies only UDP [Feature:NetworkPolicy]	network_policy
66	should enforce policy to allow traffic based on NamespaceSelector with MatchLabels using default ns label [Feature:NetworkPolicy]	network_policy
67	should enforce policy based on NamespaceSelector with MatchExpressions using default ns label [Feature:NetworkPolicy]	network_policy
68	should support a 'default-deny-ingress' policy [Feature:NetworkPolicy]	network_policy
69	should enforce policy based on Ports [Feature:NetworkPolicy]	network_policy
70	should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_policy
71	should support a 'default-deny-ingress' policy [Feature:NetworkPolicy]	network_policy
72	should enforce policy based on Ports [Feature:NetworkPolicy]	network_policy
73	should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector [Feature:NetworkPolicy]	network_policy

目前不支持的场景为：

1. 当 Pod 为某个 Service 的后端时，通过 egress 放通对该 Pod 访问，如果通过 Service IP 对该 Pod 进行访问会被拒绝（与原有 Network Policy 组件不同，原有的组件通过 Service IP 访问时可以访问到该 Pod）。
2. 不支持 named port 类型的规则。

具体的 e2e 测试 case 如下：

case	现象	原因
should enforce except clause while egress access to server in CIDR block	当在 NetworkPolicy 中放通某个 Pod 的 IP CIDR 时，不能放通 Pod 相关的 Service IP。	由于底层 ebpf 在 Pod 网卡 egress 处实现，此时报文的 Service IP 还未被 iptables/ipvs DNAT 为 Pod IP，因此会被挂载的 ebpf 程序丢弃。
should allow egress access to server in CIDR block		
should ensure an IP overlapping both IPBlock.CIDR and IPBlock.Except is allowed		
should allow egress access on one named port	egress 的 named port 不生效（暂不支持）	如果 egress 的 peer 是某个外网 IP 地址，那么 named port 没有源头获取，所以暂不支持。

GooseFS-CSI 说明

最近更新时间：2025-02-26 16:53:52

简介

组件介绍

Kubernetes-csi-tencentcloud GooseFS 插件实现 CSI 的接口，可帮助您在容器集群使用腾讯云数据加速器。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
goosefs-csi-controller	Deployment	0.02核CPU，220MB内存	kube-system
goosefs-csi-nodeplugin	DaemonSet	0.06核CPU，270MB内存	kube-system
csi-goosefs-controller	ServiceAccount	-	kube-system
csi-goosefs-controller	ClusterRole	-	-
csi-goosefs-controller	ClusterRoleBinding	-	-
csi-goosefs-node	ServiceAccount	-	kube-system
csi-goosefs-node	ClusterRole	-	-
csi-goosefs-node	ClusterRoleBinding	-	-
goosefs-csi-fuse-config	ConfigMap	-	kube-system
com.tencent.cloud.csi.goosefs	CSIDriver	-	-

使用场景

数据加速器（Data Accelerator Goose FileSystem，GooseFS）是由腾讯云推出的多协议、高性能、大吞吐的数据加速服务。GooseFS 可以为上层计算应用提供统一的命名空间和访问协议，方便用户在不同的存储系统管理和流转数据。依托对象存储作为数据存储底座的海量存储、低成本优势，GooseFS 不仅可以加速海量数据分析、机器学习、人工智能等业务访问存储的性能，还可以帮助业务实现冷热数据自动分层，平衡业务架构的性能表现和成本支出。
通过 GooseFS-CSI 扩展组件，您可以快速的在容器集群中使用GooseFS。

限制条件

- 支持 Kubernetes 1.14 以上版本的集群。
- 支持 GooseFS 可用地域。
- 在 TKE 中使用 GooseFS，需要在集群内安装该扩展组件，将占用一定的系统资源。

goosefs-csi 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机 `/var/lib/kubelet` 相关目录到容器来完成 volume 的 mount/umount，因此需要开启特权级容器。

权限场景

goosefs-csi-controller

功能	涉及对象	涉及操作权限
根据 pvc/pv 等信息完成盘的创建和删除。	nodes/csinodes/storageclasses/volumesnapshots/volumesnapshotcontents	get/watch/list
	PersistentVolumeClaim	get/watch/list/update

	PersistentVolume	get/watch/list/create/delete
通过 kubernetes lease 机制支持 controller 多副本。	leases	get/list/watch/create/update/patch/delete

goosefs-csi-node

功能	涉及对象	涉及操作权限
启动和删除进行 goosefs fuse 挂载的伴随 pod	pods	create/delete

权限定义

goosefs-csi-controller

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: csi-goosefs-controller
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotcontents"]
    verbs: ["get", "list"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["csinodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

goosefs-csi-node

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: csi-goosefs-node
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create", "delete"]
```

使用方法

安装 GooseFS 扩展组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 GooseFS。
5. 单击**完成**即可创建组件。

使用数据加速器 GooseFS

您可在 TKE 集群中为工作负载挂载数据加速器，详情请参见 使用数据加速器 GooseFS 。

HPC 说明

最近更新时间：2024-12-12 16:23:22

简介

组件介绍

HPC（HorizontalPodCronscaler）是一种可以对 K8S workload 副本数进行定时修改的自研组件，配合 HPC CRD 使用，最小支持秒级的定时任务。

组件功能

- 支持设置“实例范围”（关联对象为 HPA）或“目标实例数量”（关联对象为 deployment 和 statefulset）。
- 支持开关“例外时间”。例外时间的最小配置粒度是日期，支持设置多条。
- 支持设置定时任务是否只执行一次。

部署在集群内的 Kubernetes 对象

在集群内部署 HPC，将在集群内部署以下 Kubernetes 对象：

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
horizontalpodcronscalers.autoscaling.cloud.tencent.com	CustomResourceDefinition	-	-
hpc-leader-election-role	Role	-	kube-system
hpc-leader-election-rolebinding	RoleBinding	-	kube-system
hpc-manager-role	ClusterRole	-	-
hpc-manager-rolebinding	ClusterRoleBinding	-	-
cronhpa-controller-manager-metrics-service	Service	-	kube-system
hpc-manager	ServiceAccount	-	kube-system
tke-hpc-controller	Deployment	100mCPU/pod、100Mi/pod	kube-system

限制条件

环境要求

> **说明：**
> 您在创建集群时选择1.12.4以上版本集群，无需修改任何参数，开箱可用。

- 仅支持1.12版本以上的 Kubernetes。
- 需设置 kube-apiserver 的启动参数：`--feature-gates=CustomResourceSubresources=true`

节点要求

- HPC 组件默认挂载主机的时区将作为定时任务的参考时间，因此要求节点存在 `/etc/localtime` 文件。
- HPC 默认安装2个 HPC Pod 在不同节点，因此节点数推荐为2个及以上。

被控资源要求

在创建 HPC 资源时，被控制的 workload（K8S 资源）需要存在于集群中。

组件权限说明

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
监听 horizontalpodcronscalers 的变动	horizontalpodcronscalers	create/delete/get/list/patch/watch
需要修改 deployments/statefulsets 的 replicas	deployments/statefulsets	get/list/patch/watch
修改 horizontalpodautoscalers 的 minReplicas/maxReplicas	horizontalpodautoscalers	get/list/patch/watc
同步 HPC 定时任务执行的 events	events	create/patch

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: hpc-manager-role
rules:
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - get
  - list
  - patch
  - watch
- apiGroups:
  - apps
  resources:
  - statefulsets
  verbs:
  - get
  - list
  - patch
  - watch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - get
  - list
  - patch
  - watch
- apiGroups:
  - autoscaling.cloud.tencent.com
  resources:
  - horizontalpodcronscalers
  verbs:
```

```
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - autoscaling.cloud.tencent.com
  resources:
  - horizontalpodcronscalers/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  resourceNames:
  - horizontalpodcronscalers.autoscaling.cloud.tencent.com
  verbs:
  - get
  - list
  - delete
  - watch
```

操作步骤

安装 HPC

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 HPC。
5. 单击**完成**即可创建组件。

HPC 支持 Openkruise 的高级 workload 类型

HPC 组件自v1.0.8版本之后，支持 OpenKruise 的高级 Workload 类型，操作步骤如下：

1. 在集群中安装 OpenKruise 组件，请参考官方文档 OpenKruise 安装指南 。
2. 创建关联 OpenKruise 高级工作负载类型的 HPC。
   关联 CloneSet：

```
apiVersion: autoscaling.cloud.tencent.com/v1
kind: HorizontalPodCronscaler
metadata:
  name: hpc-cloneset
  namespace: default
spec:
  scaleTarget:
    apiVersion: apps.kruise.io/v1alpha1
    kind: CloneSet
    name: nginx-deployment
    namespace: default
  crons:
  - name: "scale-down"
    excludeDates:
      - "* * * 15 11 *"
```

```
      - "* * * * * 5"
    schedule: "30 */1 * * * *"
    targetSize: 1
  - name: "scale-up"
    excludeDates:
      - "* * * 15 11 *"
      - "* * * * * 5"
    schedule: "0 */1 * * * *"
    targetSize: 3
```

关联 Advanced Statefulset：

```
apiVersion: autoscaling.cloud.tencent.com/v1
kind: HorizontalPodCronscaler
metadata:
  name: hpc-ad
  namespace: default
spec:
  scaleTarget:
    apiVersion: apps.kruise.io/v1beta1
    kind: Statefulset
    name: nginx-deployment
    namespace: default
  crons:
  - name: "scale-down"
    excludeDates:
      - "* * * 15 11 *"
      - "* * * * * 5"
    schedule: "30 */1 * * * *"
    targetSize: 1
  - name: "scale-up"
    excludeDates:
      - "* * * 15 11 *"
      - "* * * * * 5"
    schedule: "0 */1 * * * *"
    targetSize: 3
```

创建并使用 HPC 工作负载示例

创建关联 Deployment 的定时任务资源

示例如下：

```
apiVersion: autoscaling.cloud.tencent.com/v1
kind: HorizontalPodCronscaler
metadata:
  name: hpc-deployment
  namespace: default
spec:
  scaleTarget:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment
    namespace: default
  crons:
  - name: "scale-down"
    excludeDates:
      - "* * * 15 11 *"
      - "* * * * * 5"
    schedule: "30 */1 * * * *"
```

```
    targetSize: 1
  - name: "scale-up"
    excludeDates:
      - "* * * 15 11 *"
      - "* * * * * 5"
    schedule: "0 */1 * * * *"
    targetSize: 3
```

创建关联 StatefulSet 的定时任务资源

示例如下：

```
apiVersion: autoscaling.cloud.tencent.com/v1
kind: HorizontalPodCronscaler
metadata:
  name: hpc-statefulset
  namespace: default
spec:
  scaleTarget:
    apiVersion: apps/v1
    kind: Statefulset
    name: nginx-statefulset
    namespace: default
  crons:
  - name: "scale-down"
    excludeDates:
      - "* * * 15 11 *"
    schedule: "0 */2 * * * *"
    targetSize: 1
  - name: "scale-up"
    excludeDates:
      - "* * * 15 11 *"
    schedule: "30 */2 * * * *"
    targetSize: 4
```

创建关联 HPA 的定时任务资源

示例如下：

```
apiVersion: autoscaling.cloud.tencent.com/v1
kind: HorizontalPodCronscaler
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: hpc-hpa
spec:
  scaleTarget:
    apiVersion: autoscaling/v1
    kind: HorizontalPodAutoscaler
    name:  nginx-hpa
    namespace: default
  crons:
  - name: "scale-up"
    schedule: "30 */1 * * * *"
    minSize: 2
    maxSize: 6
  - name: "scale-down"
    schedule: "0 */1 * * * *"
    minSize: 1
```

```
    maxSize: 5
```

定时时间设置参考

字段名称	是否必选	允许值范围	允许的特殊字符
Seconds	是	0 – 59	* / , –
Minutes	是	0 – 59	* / , –
Hours	是	0 – 23	* / , –
Day of month	是	1 – 31	* / , – ?
Month	是	1 – 12 或 JAN – DEC	* / , –
Day of week	是	0 – 6 或 SUN – SAT	* / , – ?

注意：
当定时时间设置同时包含星期几和月份日期时，若时间匹配其中任一设置，命令都会执行。例如，设置 `"30 4 1,15 * 5"` 将导致命令在每个月的1日和15日的凌晨4:30以及每个星期五运行。更多定时时间设置格式可参考 crontab 规范 。

Network Policy 说明

最近更新时间：2024-01-29 15:18:51

简介

组件介绍

Network Policy 是 Kubernetes 提供的一种资源，用于定义基于 Pod 的网络隔离策略。它描述了一组 Pod 是否可以与其他组 Pod，以及其他 Network Entities 进行通信。本组件提供了针对该资源的 Controller 实现。如果您希望在 IP 地址或端口层面（OSI 第3层或第4层）控制特定应用的网络流量，则可考虑使用本组件。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
networkpolicy	DaemonSet	每个实例CPU:250m，Memory:250Mi	kube-system
networkpolicy	ClusterRole	-	kube-system
networkpolicy	ClusterRoleBinding	-	kube-system
networkpolicy	ServiceAccount	-	kube-system

组件权限说明

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要获取集群内的 namespaces、pods、services、nodes、endpoints 和 networkpolicies，所以需要 list/get/watch 权限。

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: networkpolicy
rules:
  - apiGroups:
      - ""
    resources:
      - namespaces
      - pods
      - services
      - nodes
      - endpoints
    verbs:
      - list
      - get
      - watch
  - apiGroups:
      - "networking.k8s.io"
    resources:
      - networkpolicies
    verbs:
      - list
      - get
      - watch
  - apiGroups:
      - extensions
    resources:
      - networkpolicies
```



```
    verbs:
      - get
      - list
      - watch
```

操作步骤

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 NetworkPolicy。NetworkPolicy 详细配置可参见 Network Policy 最佳实践 。
5. 单击**完成**即可创建组件。

Nginx-ingress 说明

最近更新时间：2024-12-30 14:17:42

简介

组件介绍

Nginx 可以用作反向代理、负载平衡器和 HTTP 缓存。Nginx-ingress 组件是使用 Nginx 作为反向代理和负载平衡器的 Kubernetes 的 Ingress 控制器。您可以部署 Nginx-ingress 组件，在集群中使用 Nginx-ingress。

部署在集群内的 Kubernetes 对象

在集群内部署 Nginx-ingress Add-on，将在集群内部署以下 Kubernetes 对象：

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
tke-ingress-nginx-controller-operator	Deployment	0.13核 CPU，128MB内存	kube-system
ingress-nginx-controller	Deployment/DaementSet	0.1核 CPU	kube-system
ingress-nginx-controller-hpa	HPA	-	kube-system

在集群内创建 Nginx ingress 实例后会在集群内部部署以下 Kubernetes 对象：

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
nginx-ingress	Service	-	自定义设置
nginx-ingress	Configmap	-	自定义设置

前提条件

- Kubernetes 版本建议在1.16版本及以上。
- 建议您使用 TKE 节点池功能。
- 建议您使用 腾讯云日志服务 CLS。

关于可用性保障

NginxIngress 是开源社区组件，TKE 仅提供安装，不提供可用性保障，需用户自行运维。

使用方法

- Nginx-ingress 概述
- Nginx-ingress 安装
- 使用 Nginx-ingress 对象接入集群外部流量
- Nginx-ingress 日志配置

NodeLocalDNSCache 说明

最近更新时间：2025-01-09 15:23:53

简介

组件介绍

NodeLocal DNSCache 通过在集群节点上作为 DaemonSet 运行 DNS 缓存代理来提高集群 DNS 性能。在当今的体系结构中，处于 ClusterFirst DNS 模式的 Pod 可以连接到 kube-dns serviceIP 进行 DNS 查询。通过 kube-proxy 添加的 iptables 规则将其转换为 kube-dns/CoreDNS 端点。借助此新架构，Pods 将可以访问在同一节点上运行的 DNS 缓存代理，从而避免了 iptables DNAT 规则和连接跟踪。本地缓存代理将查询 kube-dns 服务以获取集群主机名的缓存缺失（默认为 cluster.local 后缀）。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
node-local-dns	DaemonSet	每节点50mCPU，5Mi内存	kube-system
kube-dns-upstream	Service	-	kube-system
node-local-dns	ServiceAccount	-	kube-system
node-local-dns	Configmap	-	kube-system

限制条件

- 仅支持 1.14 版本以上的 kubernetes 版本。
- VPC-CNI 同时支持 kube-proxy 的 iptables 和 ipvs 模式，GlobalRouter 仅支持 kube-proxy 的 iptables 模式，ipvs 模式下需要更改 kubelet 参数，详情请参见 官方文档 。
- 集群创建后没有调整过 dns 服务对应工作负载的相关 name 和 label，检查集群 kube-system 命名空间中存在以下 dns 服务的相关工作负载：
  - service/kube-dns
  - deployment/kube-dns 或者 deployment/coredns，且存在 k8s-app: kube-dns 的 label
- IPVS 的独立集群，需要确保 add-pod-eni-ip-limit-webhook ClusterRole 具备以下权限：

```
- apiGroups:
  - ""
  resources:
  - configmaps
  - secrets
  - namespaces
  - services
  verbs:
  - list
  - watch
  - get
  - create
  - update
  - delete
  - patch
```

- IPVS 的独立集群和托管集群，都需要确保 tke-eni-ip-webhook Namespace 下的 add-pod-eni-ip-limit-webhook Deployment 镜像版本大于等于 v0.0.6。

推荐配置

当安装 NodeLocal DNSCache 后，推荐为 CoreDNS 增加如下配置：

```
        template ANY HINFO . {
            rcode NXDOMAIN
```

```
        }
        forward . /etc/resolv.conf {
            prefer_udp
        }
```

操作步骤

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 NodeLocalDNSCache。NodeLocalDNSCache 详细配置可参见 官方文档 。
5. 单击**完成**即可创建组件。

Release Log

版本号	适用集群	镜像地址	变更时间	变更内容	变更影响
1.0.1	适用于 Kubernetes 1.14及以上版本的 TKE 托管集群、独立集群。	ccr.ccs.tencentyun.com/tcrimages/k8s-dns-node-cache:1.22.28	2024年12月25日	更新 NodeLocalDNS 的镜像版本至1.22.28。 更新 Request\Limit 的配置项支持。	此次升级不会对业务造成影响。
1.0.0	适用于 Kubernetes 1.14及以上版本的 TKE 托管集群、独立集群。	ccr.ccs.tencentyun.com/tcrimages/k8s-dns-node-cache:1.15.13	2021年11月26日	新增 NodeLocalDNS 插件 NodeLocalDNS 的镜像版本1.15.13。	-

NodeProblemDetectorPlus 说明

最近更新时间：2024-11-05 14:34:52

简介

组件介绍

Node-Problem-Detector-Plus 是 Kubernetes 集群节点的健康监测组件。在容器服务 TKE 环境中以 DaemonSet 方式运行，帮助用户实时检测节点上的各种异常情况，并将检测结果报告给上游的 Kube-apiserver。

部署在集群内的 Kubernetes 对象

kubernetes 对象名称	类型	资源量	Namespaces
node-problem-detector	DaemonSet	0.1C200M	kube-system
node-problem-detector	ServiceAccount	-	kube-system
node-problem-detector	ClusterRole	-	-
node-problem-detector	ClusterRoleBinding	-	-

使用场景

使用 Node-Problem-Detector-Plus 组件可以监控节点的工作状态，包括内核死锁、OOM、系统线程数压力、系统文件描述符压力等指标，通过 Node Condition 和 Event 的形式上报给 Apiserver。
您可以通过检测相应的指标，提前预知节点的资源压力，可以在节点开始驱逐 Pod 之前手动释放或扩容节点资源压力，防止 Kubernetes 进行资源回收或节点不可用可能带来的损失。

限制条件

在集群中使用 NPD，需要在集群内安装该扩展组件，NPD 容器将被限制使用0.5核 CPU，80M内存的系统资源。

组件权限说明

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
需要在节点遇到故障时上报故障信息，需要修改 node 的 condition	nodestatus	patch
需要发送 event 通知集群	event	create/patch/update

权限定义

```
rules:
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - nodes/status
  verbs:
  - patch
```

```
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
  - update
```

使用方法

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 **Node-Problem-Detector-Plus**。
5. 单击**完成**即可创建组件。组件安装成功后，您的集群中会有对应的 node-problem-detector 资源，在 Node 的 Condition 中也会增加相应的条目。

附录

Node Conditions

安装 NPD 插件后，会在节点中增加以下特定的 Conditions：

Condition Type	默认值	描述
ReadonlyFilesystem	False	文件系统是否只读
FDPressure	False	查看主机的文件描述符数量是否达到最大值的80%
FrequentKubeletRestart	False	Kubelet 是否在20Min内重启超过5次
CorruptDockerOverlay2	False	DockerImage 是否存在问题
KubeletProblem	False	Kubelet service 是否 Running
KernelDeadlock	False	内核是否存在死锁
FrequentDockerRestart	False	Docker 是否在20Min内重启超过5次
FrequentContainerdRestart	False	Containerd 是否在20Min内重启超过5次
DockerdProblem	False	Docker service 是否 Running（若节点运行时为 Containerd，则一直为 False）
ContainerdProblem	False	Containerd service 是否 Running（若节点运行时为 Docker，则一直为 False）
ThreadPressure	False	系统目前线程数是否达到最大值的90%
NetworkUnavailable	False	NTP service 是否 Running
SerfFailed	False	分布式检测节点网络健康状态
CustomPIDPressure	False	当前节点 PID 数已经达到最大值的90%
NFConntrackPressure	False	当前节点 conntrack 表用量已经达到了总量的90%

nvidia-gpu 说明

最近更新时间：2024-03-26 15:44:31

简介

组件介绍

该组件通过自动发现节点上的 NVIDIA GPU（nvidia.com/gpu）作为可调度资源来帮助运行需要 GPU 资源的容器；同时支持自动部署监控 exporter，提供了卡、Pod 和容器级别的 GPU 监控。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
nvidia-device-plugin-daemonset	DaemonSet	0.1核CPU，100MB内存	kube-system
nvidia-gpu-exporter	DaemonSet	0.1核CPU，100MB内存	kube-system
nvidia-gpu-exporter	ServiceAccount	-	kube-system
nvidia-gpu-exporter	ClusterRole	-	-
nvidia-gpu-exporter	ClusterRoleBinding	-	-
nvidia-gpu-exporter	Service	-	kube-system

限制条件

支持 Kubernetes 1.16及以上版本的集群。

组件权限

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
需要跟踪及获取节点信息，以确定显存单位。	Nodes	get, list, watch
需要跟踪及获取 Pod 信息，以确定需要收集指标的 GPU Pod 信息。	Pods	get, list, watch

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nvidia-gpu-exporter
rules:
  - apiGroups:
      - ""
    resources:
      - pods
      - nodes
    verbs:
      - get
      - list
      - watch
```

使用方法

安装

- 若您在集群中创建了 GPU 节点且未开启 qgpu 功能，平台将自动安装该组件，您可前往集群的**组件管理**页面管理。
- 若您的集群已经存在 GPU 节点且希望能采集到 GPU 相关的基础监控数据，您可以按照以下步骤进行安装：

  1.1 登录 容器服务控制台，在左侧导航栏中选择**集群**。

  1.2 在集群列表中，单击目标集群 ID，进入集群详情页。

  1.3 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。

  1.4 在**新建组件管理**页面中勾选 nvidia-gpu。

  1.5 单击**完成**即可创建组件。

> **注意：**
> 已经存在 GPU 节点的集群在**首次安装**本组件时，需检查集群中是否已存在 DaemonSet `nvidia-device-plugin-daemonset` 。如果存在，请先删除该 DaemonSet，然后再安装本组件 nvidia-gpu。删除 DaemonSet 不影响存量已运行的 GPU 业务 Pod，安装期间新建的 GPU Pod 会在安装完成后自动运行。

查看监控

您可通过控制台提供的基础监控查看**卡/Pod/容器**相关的 GPU 监控，详情请参见 查看监控数据 。

> **注意：**
> 组件版本限制：GPU 监控查看需要您将组件 monitor-agent 升级至1.3.11及以上。组件详情请参见 tke-monitor-agent 说明 。

监控指标

GPU 卡相关指标

GPU 卡的指标形式为：

```
gpu_core_usage{card="0",node="10.0.66.4"} 0
```

> **说明：**
> “card” 表示 GPU 的序号，“node” 表示所在节点。

gpu_xxx	GPU 指标
gpu_count	GPU 卡的数量。
gpu_mem_per_card	单张 GPU 卡的显存大小（单位为 MiB）。请注意，这里取第一张显卡的显存大小，一般情况下多张卡时每张卡的显存大小一致。但如果某些卡未开启 ECC，可能导致显存大小不一致。建议所有卡都应该打开 ECC。
gpu_core_usage	GPU 实际使用的算力（单位1%，使用全部算力时，值为100）。
gpu_mem_usage	GPU 实际使用的显存（单位为 MiB）。
gpu_core_utilization_percentage	GPU 算力使用率。
gpu_mem_utilization_percentage	GPU 显存使用率。
gpu_mem_unit_mb	每份 qgpu-memory 的显存大小（单位为 MiB），通常为 1024 或 128。
gpu_enc_utilization_percentage	GPU 编码资源使用率。
gpu_dec_utilization_percentage	GPU 解码资源使用率。

Pod 相关指标

Pod 的指标形式为：

```
pod_core_usage{namespace="default",node="10.0.66.4",pod="7a2fa737-eef1-4801-8937-493d7efb16b7"} 0
```

> **说明：**
> “namespace” 表示 Pod 所在的 namespace，“node” 表示 Pod 所在节点，“pod” 表示 Pod 的名称。

pod_xxx	pod 指标
pod_core_usage	pod 实际使用的算力（单位1%，使用全部算力时，值为100）。
pod_mem_usage	pod 实际使用的显存（单位为 MiB）。
pod_core_utilization_percentage	pod 实际使用的算力占申请算力的百分比。
pod_mem_utilization_percentage	pod 实际使用的显存占申请显存的百分比。
pod_core_occupy_node_percentage	pod 实际使用的算力占节点总算力百分比。
pod_mem_occupy_node_percentage	pod 实际使用的显存占节点总显存百分比。
pod_core_request	pod 申请的算力（单位1%）。
pod_mem_request	pod 申请的显存（单位为 MiB）。
pod_enc_utilization_percentage	pod GPU 编码资源使用率。
pod_dec_utilization_percentage	pod GPU 解码资源使用率。

容器相关指标

容器的指标形式为：

```
container_gpu_utilization{container="cuda",namespace="default",node="10.0.66.4",pod="cuda"} 0
```

> **说明：**
> “container” 表示容器名称，“namespace” 表示容器所在的 namespace，“node” 表示容器所在节点，“pod” 表示容器所在的 Pod 的名称。

container_xxx	容器指标
container_gpu_utilization	容器实际使用的算力（单位1%）。
container_gpu_memory_total	容器实际使用的显存（单位为 MiB）。
container_core_utilization_percentage	容器实际使用的算力占申请算力的百分比。
container_mem_utilization_percentage	容器实际使用的显存占申请显存的百分比。
container_request_gpu_memory	容器申请的显存（单位为 MiB）。
container_request_gpu_utilization	容器申请的算力（单位1%）。
container_enc_utilization_percentagek8s_container_enc_util	容器 GPU 编码资源使用率。
container_dec_utilization_percentagek8s_container_dec_util	容器 GPU 解码资源使用率。

OOMGuard 说明

最近更新时间：2025-04-08 15:55:22

简介

> **说明：**
> 该组件在用户态降低了由于 cgroup 内存回收失败而产生的各种内核故障的发生几率，仅适用于解决操作系统版本为 CentOS 7.2/7.6 的原生内核缺陷，其他镜像版本无需安装。

组件介绍

内存溢出（Out of Memory，OOM）是指应用系统中存在无法回收的内存或使用的内存过多，最终使得程序运行要用到的内存大于能提供的最大内存。当 cgroup 内存不足时，Linux 内核会触发 cgroup OOM 来选择一些进程终止，以便能回收一些内存从而尽量继续保持系统继续运行。但 Linux 内核（尤其是 3.10等低版本内核）对 cgroup OOM 的处理存在很多问题，频繁的 cgroup OOM 经常会带来节点故障（例如卡死、重启或进程异常但无法终止）的情况。
OOMGuard 是容器服务 TKE 提供的在用户态处理容器 cgroup OOM 的组件。当 cgroup OOM 情况出现时，在系统内核终止相关容器进程之前，OOMGuard 组件会直接在用户空间终止超过内存限制的容器，从而减少了在内核态回收内存失败而触发各种节点故障的概率。
在触发阈值进行 OOM 之前，OOMGuard 会先通过写入 `memory.force_empty` 触发相关 cgroup 的内存回收，如果 `memory.stat` 显示还有较多 cache，则不会触发后续处理策略。在 cgroup OOM 终止掉容器后，会向 Kubernetes 上报 `OomGuardKillContainer` 事件，可以通过 `kubectl get event` 命令进行查看。

原理介绍

核心思想是在发生内核 cgroup OOM kill 之前，在用户空间终止掉超限的容器， 减少走到内核 cgroup 内存回收失败后的代码分支从而触发各种内核故障的机会。
oom-guard 会给 memory cgroup 设置 threshold notify，接受内核的通知。详情见 threshold notify。

示例

假如一个 pod 设置的 memory limit 是1000M，oom-guard 会根据配置参数计算出 margin。

```
margin = 1000M * margin_ratio = 20M //缺省 margin_ratio 是 0.02
```

另外 margin 最小不小于 min_margin（缺省1M），最大不大于 max_margin（缺省为50M）。如果超出范围，则取 min_margin 或 max_margin。然后计算 threshold：

```
threshold = limit - margin //即 1000M - 20M = 980M
```

把980M作为阈值设置给内核。当这个 pod 的内存使用量达到980M时，oom-guard 会收到内核的通知。
在触发阈值之前，oom-guard 会先通过 memory.force_empty 触发相关 cgroup 的内存回收。另外，如果触发阈值时，相关 cgroup 的 memory.stat 显示还有较多 cache，则不会触发后续处理策略，这样当 cgroup 内存达到 limit 时，内核还是会触发 cgroup OOM。

达到阈值后的处理策略

通过 `--policy` 参数来控制处理策略。目前有以下三个策略，缺省策略是 container。

策略	描述
process	采用跟内核 cgroup OOM killer 相同的策略，在该 cgroup 内部，选择一个 oom_score 得分最高的进程终止。通过 oom-guard 发送 SIGKILL 来终止掉进程。
container	在该 cgroup 下选择一个 docker 容器，终止掉整个容器。
noop	只记录日志，并不采取任何措施。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
oomguard	ServiceAccount	-	kube-system

system:oomguard	ClusterRoleBinding	-	-
oom-guard	DaemonSet	0.02核 CPU，120MB内存	kube-system

使用场景

应用于节点内存压力比较大，业务容器经常发生 OOM 导致节点故障的 Kubernetes 集群。

限制条件

- 没有修改 containerd 服务 socket 路径，保持 TKE 的默认路径：
    - docker 运行时： `/run/docker/containerd/docker-containerd.sock`
    - containerd 运行时： `/run/containerd/containerd.sock`
- 没有修改 cgroup 内存子系统挂载点，保持默认挂载点： `/sys/fs/cgroup/memory`

组件权限说明

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- OOM guard 需要在出现 OOM 时通过 event 发送 OOM 的情况，因此需要 event 的 create/patch/update 权限。

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:oomguard
rules:
- apiGroups:
  - ''
  resources:
  - 'events'
  verbs:
  - create
  - patch
  - update
```

使用方法

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 OOMGuard。
5. 单击**完成**即可安装组件。

rdma-agent 说明

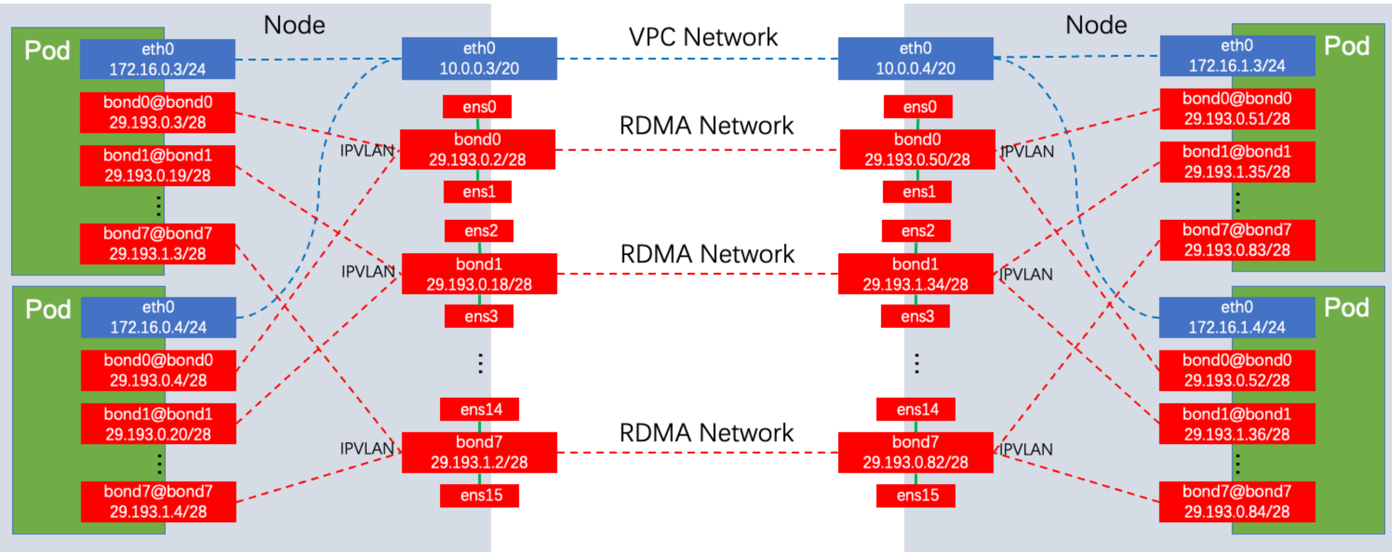
最近更新时间：2025-03-10 17:11:32

简介

rdma-agent 是腾讯云容器服务 TKE 提供的网络组件，该组件会自动检测节点可用的 RDMA 设备，并注册到 Kubernetes 中。该组件支持 RDMA 网络虚拟化，能够将 RDMA 设备分配绑定到 Kubernetes Pod 当中，从而支持 Pod 使用 RDMA 设备的高性能数据传输能力。

网络架构

rdma-agent 支持通过 IPVLAN 实现 RDMA 网络虚拟化，其网络架构如下：



上图为 2 个节点 4 个 Kubernetes Pod 的图例，在图中：

- 每个节点有 1 个主网卡和 8 个 RDMA Bond 网卡。
- 每个 Pod 分配了 1 个名为 eth0 的普通网卡，可访问普通 VPC 网络。
- 每个 Pod 还分配了 8 个 IPVLAN 虚拟化出的 RDMA Bond 网卡，每个网卡与节点的其中一个 Bond 网卡对应，且有独立的 IP 地址，可访问 RDMA 互联网络。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
tke-rdma-agent-conf	ConfigMap	-	kube-system
tke-rdma-shared-agent	DaemonSet	0.01核CPU，100MB内存	kube-system

使用限制

- 支持 Kubernetes 1.20及以上版本的集群。
- 仅支持部署在相关有 RDMA 设备的实例上。

组件部署

前置检查

> **注意：**
> - 确保 RDMA netns 为 shared 模式，如果不是 shared，组件不能正常工作。
> - 确保 RDMA 机器加入的 高性能计算集群（THCC） 已经支持 /28 及以上的网段。

1. 执行以下命令，查看节点当前 rdma netns 是否为 shared：

```
/opt/mellanox/iproute2/sbin/rdma system show
```

输出应为：

```
netns shared
```

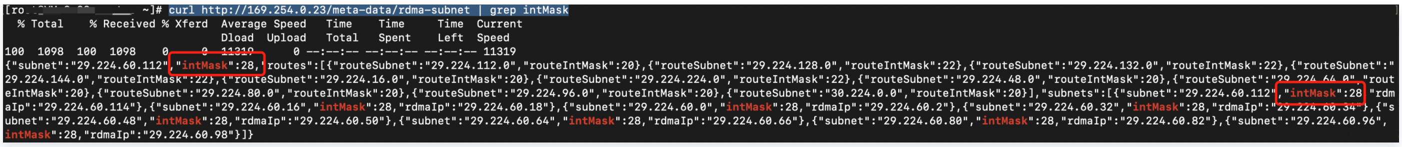
2. 如果输出为 exclusive，则需要执行以下命令修改为 shared：

```
rm /etc/modprobe.d/ib_core.conf
/opt/mellanox/iproute2/sbin/rdma system set netns shared
```

如果命令执行失败，则需要重启节点。

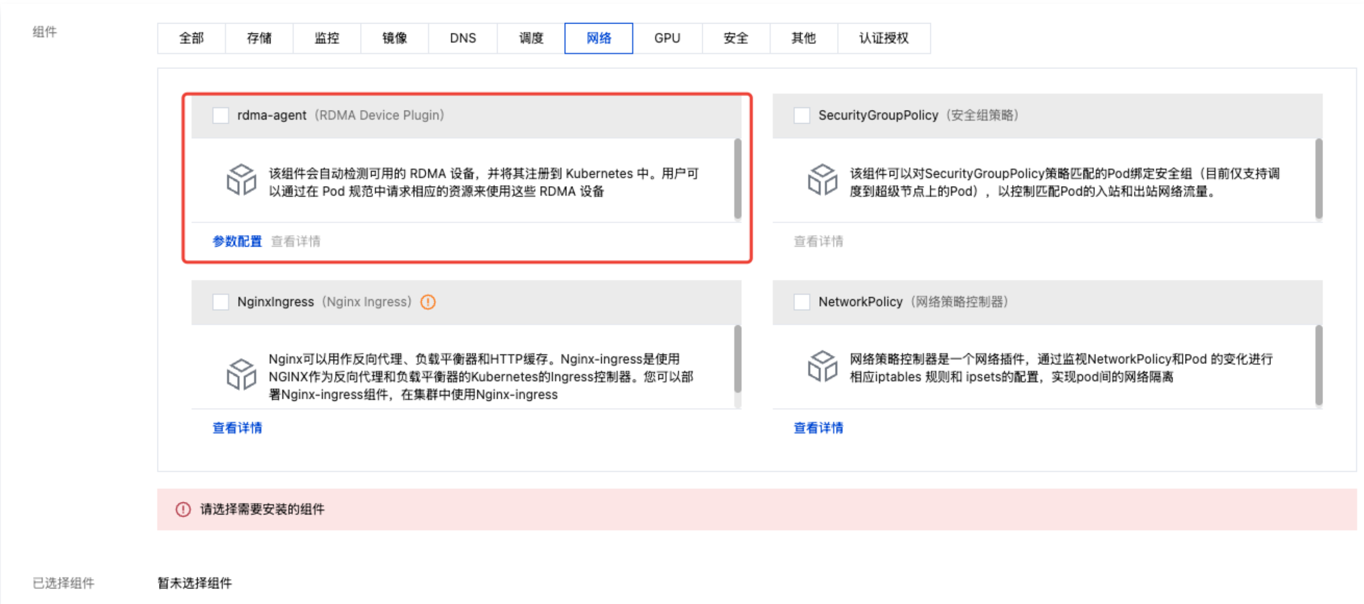
3. 确保 RDMA 机器加入的 THCC 集群已经支持 /28 及以上的网段，在节点上访问 meta-data 接口查询 RDMA 网卡支持的网段，确保其中 intMask 都为 /28 及以上，否则无法支持 IPVLAN 网络虚拟化能力（仅能支持 HostNetwork 的方式使用）。

```
curl http://169.254.0.23/meta-data/rdma-subnet | grep intMask
```



部署安装

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 **rdma-agent**。如下图所示：



5. 参数配置请参见 组件参数配置说明 。
6. 单击**完成**即可创建组件。

部署节点设置 label

1. 在 容器服务控制台 的集群详情页中，选择**节点管理 > 节点池**，单击**编辑**修改节点池中节点的 label。如下图所示：

2. 在编辑参数设置中，新增或修改节点池中节点的 label，默认需添加 `feature.node.kubernetes.io/tke-shared-rdma=true`：

3. 添加、修改完以后，也可勾选“对存量节点应用本次 Label/Annotation/Taint 更新”，从而应用到存量节点。

   存量节点也可通过以下命令给节点添加 label，从而部署 rdma-agent 组件：

```
kubectl label node 10.226.0.19 feature.node.kubernetes.io/tke-shared-rdma=true
```

组件参数配置说明

参数配置项	描述
CpuRequest	组件在每个节点上运行时需要占用的 CPU，默认为 10m。
CpuLimit	组件在每个节点上运行时最大占用的 CPU，默认为 50m。
MemoryRequest	组件在每个节点上运行时需要占用的内存，默认为 100Mi。
MemoryLimit	组件在每个节点上运行时最大占用的内存，默认为 500Mi。
KubeletRootDir	组件运行所在节点的 kubelet 的根目录，默认为 /var/lib/kubelet，根目录修改时需同步修改组件的这个配置。
NodeSelector	组件所需运行节点的 label 键值对，默认配置是 `feature.node.kubernetes.io/tke-shared-rdma=true` ，若需自定义部署的节点 label，则需要修改这个配置。

使用方法

使用方法请参见 在 TKE 上运行基于 NCCL 的 RDMA 分布式训练任务 。

TCR 说明

最近更新时间：2024-09-02 17:34:41

简介

组件介绍

TCR Addon 是容器镜像服务 TCR 推出的容器镜像内网免密拉取的官方组件。在容器服务 TKE 集群中安装该组件后，集群节点可通过内网拉取企业版实例内容器镜像，且无需在集群资源 YAML 中显式配置 ImagePullSecret。可提高 TKE 集群内镜像拉取速度，简化镜像配置流程。

> **说明：**
> - TKE 集群需为 v1.10.x 及以上版本。建议在v1.12.x 及以上版本中使用本组件。
> - Kubernetes 的 `controller manager` 组件的启动参数需要包含 `authentication-kubeconfig` 和 `authorization-kubeconfig`（TKE v.12.x 默认启用）。

部署在集群内的 Kubernetes 对象

名称	类型	资源量	Namespace
tcr-assistant-system	Namespace	1	-
tcr-assistant-manager-role	ClusterRole	1	-
tcr-assistant-manager-rolebinding	ClusterRoleBinding	1	-
tcr-assistant-leader-election-role	Role	1	tcr-assistant-system
tcr-assistant-leader-election-rolebinding	RoleBinding	1	tcr-assistant-system
tcr-assistant-webhook-server-cert	Secret	1	tcr-assistant-system
tcr-assistant-webhook-service	Service	1	tcr-assistant-system
tcr-assistant-validating-webhook-configuration	ValidatingWebhookConfiguration	1	tcr-assistant-system
imagepullsecrets.tcr.tencentcloudcr.com	CustomResourceDefinition	1	tcr-assistant-system
tcr.ips*	ImagePullSecret CRD	(2-3)	tcr-assistant-system
tcr.ips*	Secret	(2-3)*{Namespace No.}	tcr-assistant-system
tcr-assistant-controller-manager	Deployment	1	tcr-assistant-system
updater-config	ConfigMap	1	tcr-assistant-system
hosts-updater	DaemonSet	{Node No.}	tcr-assistant-system

组件资源用量

组件	资源用量	实例数量
tcr-assistant-controller-manager	CPU：500m memory：512Mi	1
hosts-updater	CPU：100m memory：100Mi	工作节点数

使用场景

免密拉取镜像

Kubernetes 集群拉取私有镜像需要创建访问凭证 Secret 资源，并配置资源 YAML 中的 ImagePullSecret 属性，显式指定已创建的 Secret。整体配置流程较为繁琐，且会因未配置 ImagePullSecret 或指定错误 Secret 而造成镜像拉取失败。
为解决以上问题，可在集群中安装 TCR 组件，组件将自动获取指定的 TCR 企业版实例的访问凭证，并下发至 TKE 集群指定命名空间内。在使用 YAML 创建或更新资源时，无需配置 ImagePullSecret，集群将会自动使用已下发的访问凭证拉取 TCR 企业版内镜像。

内网拉取镜像

组件将自动创建 DaemonSet 工作负载 host-updater，用于更新集群节点的Host配置，解析当前关联实例域名至已建立的内网访问链路专用内网 IP 上。请注意，本配置仅用于测试场景配置，建议直接使用 TCR 提供的内网链路自动解析功能，或直接使用 PrivateDNS 产品进行私有域解析配置，也可使用自建 DNS 服务自行管理解析。

限制条件

针对免密拉取镜像使用场景：
- 用户需要具有指定的 TCR 企业版实例的获取访问凭证的权限，即 CreateInstanceToken 接口调用权限。建议具有 TCR 管理员权限的用户进行此组件的配置。
- 安装组件并生效后，请避免在资源 YAML 中重复指定 ImagePullSecret，从而造成节点使用错误的镜像拉取访问凭证，引起拉取失败。

组件权限说明

> **说明：**
> **权限场景**章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考**权限定义**章节。

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
需要/支持免密拉取镜像功能，即主动帮客户管理镜像凭证（secret）。	Secret	watch、create、update、patch、delete

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: tcr-assistant-leader-election-role
  namespace: tcr-assistant-system
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
      - delete
  - apiGroups:
      - ""
    resources:
      - configmaps/status
    verbs:
      - get
```

```yaml
      - update
      - patch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: tcr-assistant-manager-role
  namespace: tcr-assistant-system
rules:
  - apiGroups:
      - ""
    resources:
      - secrets
    verbs:
      - create
      - delete
      - patch
      - update
      - watch
  - apiGroups:
      - admissionregistration.k8s.io
    resources:
      - validatingwebhookconfigurations
    verbs:
      - create
      - get
      - patch
  - apiGroups:
      - certificates.k8s.io
    resources:
      - certificatesigningrequests
    verbs:
      - create
      - delete
      - get
  - apiGroups:
      - certificates.k8s.io
    resources:
      - certificatesigningrequests/approval
    verbs:
      - update
  - apiGroups: ["certificates.k8s.io"]
    resources:
      - "signers"
    #   # resourceNames:
    #   #   # Support legacy versions, before signerName was added
    #   #   - "kubernetes.io/legacy-unknown"
    verbs:
      - approve
  - apiGroups:
      - ""
    resources:
      - namespaces
    verbs:
```

```
    - get
    - list
    - watch
- apiGroups:
    - ""
  resources:
    - namespaces/status
  verbs:
    - get
- apiGroups:
    - ""
  resources:
    - serviceaccounts
  verbs:
    - get
    - list
    - patch
    - update
    - watch
- apiGroups:
    - ""
  resources:
    - serviceaccounts/status
  verbs:
    - get
    - patch
    - update
- apiGroups:
    - tcr.tencentcloudcr.com
  resources:
    - imagepullsecrets
  verbs:
    - create
    - delete
    - get
    - list
    - patch
    - update
    - watch
- apiGroups:
    - tcr.tencentcloudcr.com
  resources:
    - imagepullsecrets/status
  verbs:
    - get
    - patch
    - update
```

操作步骤

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面单击**新建**。
4. 在**新建组件管理**页面中勾选 TCR，单击**参数配置**。
5. 在 **TCR 组件参数设置**页面，参考以下信息进行配置：
    - 选择关联实例：选择当前登录账户下已有的 TCR 企业版实例，并确认当前登录用户具有创建实例长期访问凭证的权限。如果需要新建企业版实例，请在当前集群所在地域内新建。
    - 配置免密拉取（默认启用）：可选自动下发当前操作用户的访问凭证，或指定用户名及密码，可选配置免密拉取生效的命名空间及 ServiceAccount。建议均使用默认配置，避免新建命名空间后无法使用该功能。

- 配置内网解析（高级功能）：确认集群与关联 TCR 实例已建立内网访问链路，并启用内网解析功能。请注意，本配置仅用于测试场景配置，建议直接使用 TCR 提供的内网链路自动解析功能，或直接使用 PrivateDNS 产品进行私有域解析配置，也可使用自建 DNS 服务自行管理解析。

6. 单击**完成**即可创建组件。创建组件完成后，如需修改组件相关配置，请删除组件并重新配置及安装。

> **注意：**
> 删除组件将不会同时删除自动创建的专属访问凭证，可前往容器镜像服务控制台手动禁用或删除。

原理说明

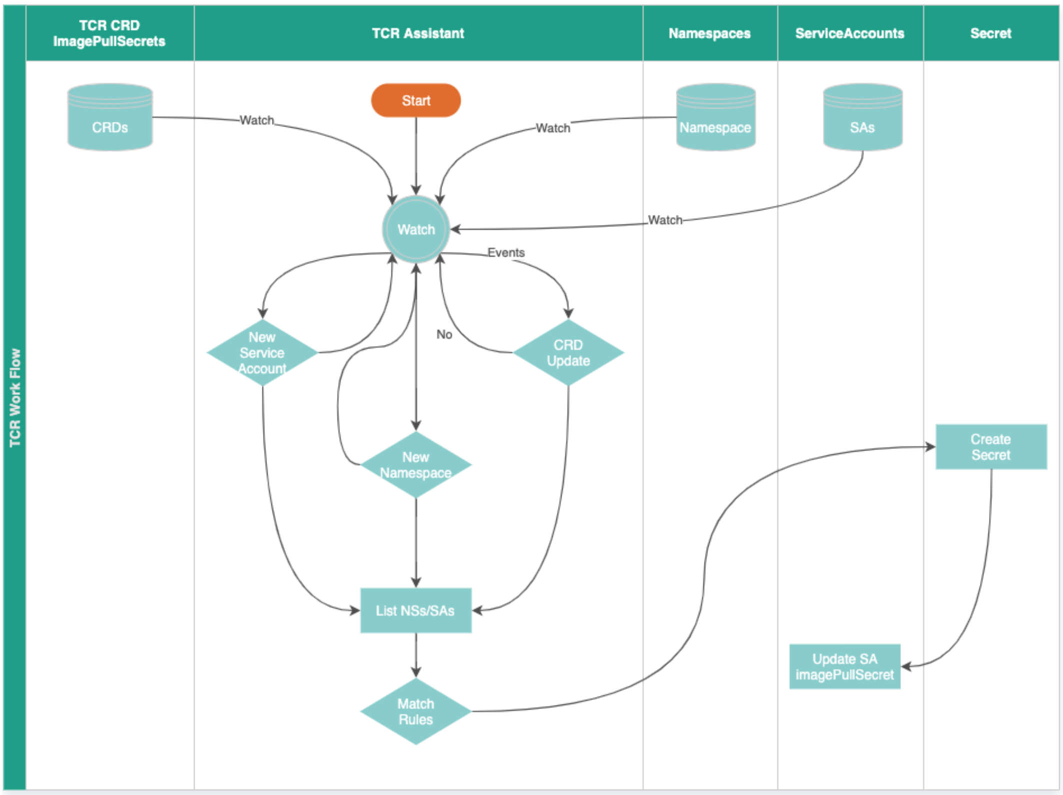
概述

TCR Assistant 用于帮助用户自动部署 k8s `imagePullSecret` 到任意 `Namespace`，并关联到该空间下的 `ServiceAccount`。在用户创建的工作负载当中**没有明确指定** `imagePullSecret` 和 `serviceAccount` 的情况下，k8s 会尝试从当前命名空间下名为 `default` 的 `ServiceAccount` 资源中查找、匹配合适的 `imagePullSecret`。

术语表

Name	别名	描述
ImagePullSecret	ips, ipss	TCR Assistant 定义的 CRD。用于存储镜像仓库用户名与密钥，分发目标 `Namespace` 和目标 `ServiceAccount`。

实现原理



TCR Assistant 是一个典型的 k8s Operator。在部署 TCR Assistant 时，我们会在目标 k8s 集群当中创建 CRD 对象：`imagepullsecrets.tcr.tencentcloudcr.com`。该 CRD 的 kind 为 `ImagePullSecret`，版本是 `tcr.tencentcloudcr.com/v1`，缩写为 `ips` 或者 `ipss`。

TCR Assistant 通过持续观察（watch）k8s 集群的 `Namespace` 和 `ServiceAccount` 资源，并在这些资源发生变更的时候，检查资源变化是否匹配 `ImagePullSecret` 中设定的规则来自动的为用户部署拉取**私有镜像仓库**所需要的 Secret 资源。程序通常部署在 k8s 集群内，使用 `in cluster` 模式访问 k8s master API。

创建 CRD 资源

程序部署完成后，不会在目标 K8s 集群部署任何的 TCR 镜像拉取密钥。此时，需要我们使用 kubectl 或者通过 Client Go 新建 `ImagePullSecret` 资源。

```
# 新建 ImagePullSecret 资源
$ kubectl create -f allinone/imagepullsecret-sample.yaml

imagepullsecret.tcr.tencentcloudcr.com/imagepullsecret-sample created
```

`ImagePullSecret` 资源示例文件（allinone/imagepullsecret-sample.yaml）：

```
apiVersion: tcr.tencentcloudcr.com/v1
kind: ImagePullSecret
metadata:
  name: imagepullsecret-sample
spec:
  namespaces: "*"
  serviceAccounts: "*"
  docker:
    username: "100012345678"
    password: tcr.jwt.token
    server: fanjiankong-bj.tencentcloudcr.com
```

`ImagePullSecret` spec 字段解释如下表：

字段	作用	注释
namespaces	`NameSpace` 匹配规则	`*` 或者空字符表示匹配任意；要匹配任意多个 `NameSpace` 则使用 `,` 分隔多个资源名称，**注意**：不支持任何表达式，需要明确填写资源名称。
serviceAccounts	`serviceAccounts` 匹配规则	`*` 或者空字符表示匹配任意；要匹配任意多个 `ServiceAccount` 则使用 `,` 分隔多个资源名称，**注意**：不支持任何表达式，需要明确填写资源名称。
docker.server	镜像仓库域名	仅填写仓库域名。
docker.username	镜像仓库用户名	请确保用户在镜像仓库拥有足够的访问权限。
docker.password	镜像仓库用户名所对应的密码	-

创建完成后，我们可以使用以下命令观察 TCR Assistant 执行结果：

```
# 列出 ImagePullSecret 信息
$ kubectl get ipss
NAME                      NAMESPACES   SERVICE-ACCOUNTS   SECRETS-DESIRED   SECRETS-SUCCESS
imagepullsecret-sample    *            *                  10                10

# 查看详细信息
$  kubectl describe ipss
Name:         imagepullsecret-sample
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  tcr.tencentcloudcr.com/v1
Kind:         ImagePullSecret
Metadata:
  Creation Timestamp:  2021-12-01T06:47:34Z
  Generation:          1
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         2021-12-01T06:47:34Z
    API Version:  tcr.tencentcloudcr.com/v1
    Manager:         manager
    Operation:       Update
```

```
    Time:               2021-12-01T06:47:38Z
  Resource Version:   30389349
  UID:                2109f384-240b-405c-9ce8-73ce938a7c2f
Spec:
  Docker:
    Password:         tcr.jwt.token
    Server:           fanjiankong-bj.tencentcloudcr.com
    Username:         100012345678
  Namespaces:         *
  Service Accounts:   *
Status:
  S As Desired:  47
  S As Success:  1
  Secret Update Successful:
    Namespaced Name:  kube-public/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  devtools/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  demo/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  kube-system/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  tcr-assistant-system/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  kube-node-lease/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  cert-manager/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  default/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:36Z
    Namespaced Name:  afm/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:37Z
    Namespaced Name:  lens-metrics/tcr.ipsimagepullsecret-sample
    Updated At:       2021-12-01T06:47:37Z
  Secrets Desired:    10
  Secrets Success:    10
  Service Accounts Modify Successful:
    Namespaced Name:  default/default
    Updated At:       2021-12-01T06:47:38Z
Events:               <none>
```

注意：

如果需要更新 TCR Assistant 部署的 `Secret` 资源，无需删除重建 `ImagePullSecret` 资源，只需要编辑其中 `docker.username` 和 `docker.password` 字段即可生效。例如：

```
$ kubectl edit ipss imagepullsecret-sample
```

Namespace 变更

TCR Assistant 在观察到有新的 k8s `Namespace` 资源创建后，会首先检查名称是否和 `ImagePullSecret` 资源中的 `namespaces` 字段匹配。如果资源名称不匹配跳过后续流程；资源名称匹配的情况下，会调用 k8s API 创建 `Secret` 资源，并添加 `Secret` 资源名称到 `ServiceAccount` 资源的 `imagePullSecrets` 字段当中。示例如下：

```
# 查看 newns 下自动部署的 Secret
$ kubectl get secrets -n newns
NAME                            TYPE                                  DATA   AGE
tcr.ipsimagepullsecret-sample   kubernetes.io/dockerconfigjson        1      7m2s
```

```
default-token-nb5vw              kubernetes.io/service-account-token   3      7m2s

# 查看 newns 下自动关联到 ServiceAccount 资源 default 中的 Secret
$ kubectl get serviceaccounts default -o yaml -n newns
apiVersion: v1
imagePullSecrets:
- name: tcr.ipsimagepullsecret-sample
kind: ServiceAccount
metadata:
  creationTimestamp: "2021-12-01T07:09:56Z"
  name: default
  namespace: newns
  resourceVersion: "30392461"
  uid: 7bc67144-3685-4666-ba41-b1447bbbaa38
secrets:
- name: default-token-nb5vw
```

ServiceAccount 变更

TCR Assistant 在观察到有新的 k8s `ServiceAccount` 资源创建后，会首先检查名称是否和 `ImagePullSecret` 资源中的 `serviceAccounts` 字段匹配。如果资源名称**不匹配**跳过后续流程；资源名称匹配的情况下，会调用 k8s API 创建或更新 `Secret` 资源，并添加 `Secret` 资源名称到 `ServiceAccount` 资源的 `imagePullSecrets` 字段当中。示例如下：

```
# 在 newns 新建 ServiceAccount 资源
$ kubectl create sa kung -n newns
serviceaccount/kung created

# 查看 newns 下自动关联到新建 ServiceAccount 资源 kung 中的 Secret
$ kubectl get serviceaccounts kung -o yaml -n newns
apiVersion: v1
imagePullSecrets:
- name: tcr.ipsimagepullsecret-sample
kind: ServiceAccount
metadata:
  creationTimestamp: "2021-12-01T07:19:12Z"
  name: kung
  namespace: newns
  resourceVersion: "30393760"
  uid: e236829e-d88e-4feb-9e80-5e4a40f2aea2
secrets:
- name: kung-token-fljt8
```

tke-event-collector 说明

最近更新时间：2024-04-03 15:53:22

组件介绍

tke-event-collector 通过监听集群事件，并上报到 CLS 以实现集群事件持久化。该组件以 Deployment 托管于容器服务。

使用场景

使用 tke-event-collector 可以采集集群事件，并上报到 CLS。用户可以在 容器服务控制台 中选择**日志管理 > 事件日志**，在事件检索页面查看事件总览和异常事件聚合检索，也可以在全局检索中根据实际需要对事件进行检索。

使用方法

1. 登录 容器服务控制台，在左侧导航栏中选择**运维功能管理**。
2. 在**运维功能管理**页面，单击集群右侧的**设置。**
3. 在**设置功能**窗口中，单击事件存储右侧的**编辑**，并勾选**开启事件存储。**
4. 选择**日志集**和**日志主题**，单击**确定**，即可开启事件存储功能。
5. 开启事件存储后，在左侧导航栏选择**日志管理 > 事件日志**，可查看事件仪表盘及检索日志。

tke-karpenter 说明

最近更新时间：2025-06-06 16:00:12

组件介绍

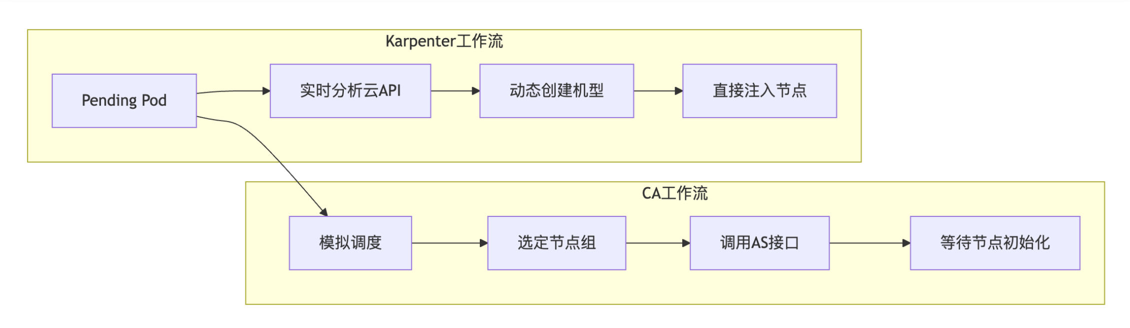
Karpenter 是一款开源的 Kubernetes 工作节点生命周期动态管理组件，能够根据集群中未调度的 Pod 自动扩展节点。当在 Kubernetes 中创建一个 Pod 时，Kubernetes 调度器负责将该 Pod 调度到集群中的一个节点上；如果没有可用的节点可以容纳该 Pod，它将保持在 Pending 状态。该组件可以监控集群中的 Pending Pod，并通过创建新节点来扩展集群资源以满足 Pod 调度需求。相比于 Cluster Autoscaler，Karpenter 具有调度速度快、调度更灵活、资源利用率高等优势。该组件主要服务于 TKE 集群中使用 Karpenter 相关功能的用户。

与传统 Cluster Autoscaler（CA）相比，Karpenter 通过革命性的动态资源感知架构，解决了企业在云原生环境中长期面临的三大核心痛点：

1. 响应延迟高：CA 复杂的节点组匹配、弹性伸缩组接口调用、节点初始化等环节导致扩容过程耗时较长，在突发流量场景易引发服务中断。
2. 调度精度低：因 CA 依赖静态节点组预设，无法动态感知 Pod 的拓扑约束要求，导致扩容随机性大，资源利用率不足，多可用区调度问题频发。
3. 运维负担重：源于 CA 需为不同机型/AZ 单独维护节点池，千级节点集群常需管理多个节点池，需手动维护多规格节点池，机型售罄、模板不一致等问题导致运维成本高。

Karpenter 作为腾讯云 TKE 新一代弹性引擎，通过动态资源感知 + 声明式 API 架构实现突破性创新：

1. 解决速度慢：Karpenter 绕过节点组和弹性伸缩组，通过实时调用腾讯云 API 直接创建定制化节点，大幅简化扩容流程，秒级完成资源供给。
2. 解决精度低：基于动态资源感知引擎，实时分析库存/成本/拓扑数据，自动选择最优机型与可用区，实现跨 AZ 调度偏差显著降低，资源利用率显著提高。
3. 解决运维重：单 NodePool 声明覆盖全量机型/AZ，无需预定义节点组，运维对象从“节点池数量”降维至“资源策略数量”。

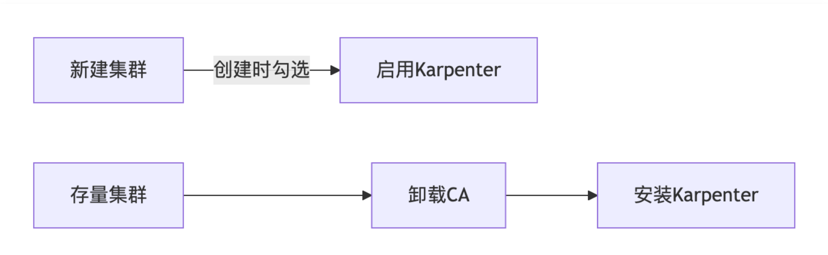


使用限制

- 该组件要求集群的 Kubernetes 版本 ⩾ 1.22。
- 该组件目前仅支持管理按量计费的原生节点。
- 该组件目前不支持 GPU 相关资源。
- 若您的集群已经使用了节点池提供的 Cluster Autoscaler (CA) 弹性伸缩能力，您需要关闭 CA 后再使用本组件，否则会存在功能冲突。

> **注意：**
> TKE 集群节点的 ENI 网卡辅助 IP 为动态分配，并对应节点的 `status.allocatable.tke.cloud.tencent.com/eni-ip` 值。当 Pod 因为 ENI IP 不足而处于 Pending 状态时，网络组件会动态调整该值，但该过程中存在一定延迟，导致组件在检测到 Pod Pending 时会迅速启动新节点。强烈建议您在新建集群时开启固定 IP 网络模式（在 Terraform 中需配置 `is_non_static_ip_mode = false`），这样节点的 `status.allocatable.tke.cloud.tencent.com/eni-ip` 值会固定，可避免不必要的节点弹出。

使用方法



存量集群迁移方案

步骤1：卸载 CA 组件

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，选择目标集群，进入集群详情页。
3. 选择**组件管理**，单击 cluster-autoscaler 右侧的**删除**。
4. 在删除资源弹窗中，单击**确定**。删除组件后，系统自动冻结所有节点池伸缩能力。

步骤2：安装 Karpenter

1. 在组件管理中，单击**新建**。
2. 在新建组件管理中，勾选 **Karpenter**。
3. 单击**完成**。

> **说明：**
> Karpenter 安装完成后，系统自动解除 CA 关联，确保互斥生效。

步骤3：部署 NodePool 资源

自定义资源 NodePool 可以为创建的节点以及在这些节点上运行的 Pod 设置约束，例如：
1. 定义污点以限制可以在扩容节点上运行的 Pod。
2. 定义节点所在的可用区、实例类型、实例规格和计费模式。
3. 定义所管理节点的资源限制。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: test
  annotations:
    kubernetes.io/description: "NodePool to restrict the number of cpus provisioned to 10"
spec:
  # Disruption section which describes the ways in which Karpenter can disrupt and replace Nodes
  # Configuration in this section constrains how aggressive Karpenter can be with performing operations
  # like rolling Nodes due to them hitting their maximum lifetime (expiry) or scaling down nodes to
reduce cluster cost
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m
    budgets:
    - nodes: 10%
  template:
    spec:
    # Requirements that constrain the parameters of provisioned nodes.
    # These requirements are combined with pod.spec.topologySpreadConstraints,
pod.spec.affinity.nodeAffinity, pod.spec.affinity.podAffinity, and pod.spec.nodeSelector rules.
    # Operators { In, NotIn, Exists, DoesNotExist, Gt, and Lt } are supported.
    # https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#operators
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.k8s.tke/instance-family
          operator: In
          values: ["S5","SA2"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
```

```
        # - key: node.kubernetes.io/instance-type
        #   operator: In
        #   values: ["S5.MEDIUM2", "S5.MEDIUM4"]
        - key: "karpenter.k8s.tke/instance-cpu"
          operator: Gt
          values: ["1"]
        # - key: "karpenter.k8s.tke/instance-memory-gb"
        #   operator: Gt
        #   values: ["3"]
      # References the Cloud Provider's NodeClass resource
      nodeClassRef:
        group: karpenter.k8s.tke
        kind: TKEMachineNodeClass
        name: default
  # Resource limits constrain the total size of the pool.
  # Limits prevent Karpenter from creating new instances once the limit is exceeded.
  limits:
    cpu: 10
```

说明：

1. 常用标签选择器。

Selector	Description
topology.kubernetes.io/zone	实例所在可用区，如 "900001"
kubernetes.io/arch	实例架构，目前仅支持 "amd64"
kubernetes.io/os	操作系统，目前仅支持 "linux"
karpenter.k8s.tke/instance-family	机型实例族，如 "S5","SA2"
karpenter.sh/capacity-type	实例计费模式，如 "on-demand"
node.kubernetes.io/instance-type	实例规格，如 "S5.MEDIUM2"
karpenter.k8s.tke/instance-cpu	实例的 CPU 数量，如 4
karpenter.k8s.tke/instance-memory-gb	实例的 Memory 大小，如 8

2. 如何设置 Topology Label?

   tke-karpenter 通过 ID 方式设置 Topology Label，如 `topology.kubernetes.io/zone: "900001"`，同时使用 `topology.com.tencent.cloud.csi.cbs/zone` 的标签值（如 `topology.com.tencent.cloud.csi.cbs/zone: ap-singapore-1`）和实际可用区描述做对应。您可以使用命令 `describe tmnc Namexxx` 查看您子网的 zone 和 zone ID。

3. 禁用 Drift 能力。

   我们认为这是一个危险的功能，因此 tke-karpenter 尚未实现 drift 能力。如果您修改了 TKEMachineNodeClass，则不会替换现有的旧节点（nodeclaim）。您的修改只会影响新节点的创建；如果您修改了 nodepool，并且现有 nodeclaim 的标签与 nodepool 要求不兼容，则将替换旧节点。例如，旧节点声明的标签为：`karpenter.k8s.tke/instance-cpu: 2`。但 nodepool 的要求已修改为：

```
template:
    spec:
      requirements:
        - key: "karpenter.k8s.tke/instance-cpu"
          operator: Gt
          values: ["2"]
```

   由于 `karpenter.k8s.tke/instance-cpu: 2` 不是 Gt 2，因此 nodeclaim 将被替换。如果您想忽略 Drifted 中断，您应该在节点池中添加以下中断设置：

```
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m
    budgets:
    - nodes: "0"
      reasons: [Drifted]
    - nodes: 10%
```

您可参考 社区介绍 获取更多操作说明。

4. 建议 expireAfter 设置为 Never。

```
expireAfter: 720h | Never
```

`spec.template.spec.expireAfter` 字段定义了节点在被移除之前可以在集群中存活的时间，减少长时间运行的节点会产生的如文件碎片或系统进程的内存泄漏等问题。由于该参数会导致节点定期销毁重建，为避免重建过程中对业务产生影响，您可以选择设置字符串值 “Never” 来完全禁用过期。

步骤4：部署 TKEMachineNodeClass 资源

NodeClass 支持配置 TKE 节点的相关参数，如子网、系统盘、安全组、节点登录方式 SSH 密钥等。每个 NodePool 必须通过 `spec.template.spec.nodeClassRef` 引用 TKEMachineNodeClass，多个 NodePool 可能指向同一个 TKEMachineNodeClass。

```
apiVersion: karpenter.k8s.tke/v1beta1
kind: TKEMachineNodeClass
metadata:
  name: default
  annotations:
    kubernetes.io/description: "General purpose TKEMachineNodeClass"
spec:
  ## using kubectl explain tmnc.spec.internetAccessible to check how to use internetAccessible filed.
  # internetAccessible:
  #   chargeType: TrafficPostpaidByHour
  #   maxBandwidthOut: 2
  ## using kubectl explain tmnc.spec.systemDisk to check how to use systemDisk filed.
  # systemDisk:
  #   size: 60
  #   type: CloudSSD
  ## using kubectl explain tmnc.spec.dataDisks to check how to use systemDisk filed.
  # dataDisks:
  # - mountTarget: /var/lib/container
  #   size: 100
  #   type: CloudPremium
  #   fileSystem: ext4
  subnetSelectorTerms:
    # repalce your tag which is already existed in https://console.cloud.tencent.com/tag/taglist
    - tags:
        karpenter.sh/discovery: cls-xxx
    # - id: subnet-xxx
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: cls-xxx
    # - id: sg-xxx
  sshKeySelectorTerms:
    - tags:
        karpenter.sh/discovery: cls-xxx
    # - id: skey-xxx
```

步骤5：部署工作负载

您可以通过扩展应用程序 Pod 的数量，并观察它们处于 Pending 状态时 Karpenter 如何创建新节点。常用命令如下：

```
# Get nodepool
kubectl get nodepool
# Get nodeclaim
kubectl get nodeclaim
# Get TKEMachineNodeClass
kubectl get tmnc
# Check your cloud resources has been synced to nodeclass
kubectl describe tmnc default
```

以下是 `kubectl describe tmnc default` 命令的示例输出：

```
Status:
  Conditions:
    Last Transition Time:  2024-08-21T09:17:26Z
    Message:
    Reason:                Ready
    Status:                True
    Type:                  Ready
  Security Groups:
    Id:  sg-xxx
  Ssh Keys:
    Id:  skey-xxx
    Id:  skey-xxx
    Id:  skey-xxx
  Subnets:
    Id:       subnet-xxx
    Zone:     ap-singapore-1
    Zone ID:  900001
    Id:       subnet-xxx
    Zone:     ap-singapore-4
    Zone ID:  900004
```

增量集群安装方案

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 单击**新建**。新建集群步骤请参见 创建集群 。
3. 在创建集群 > 组件配置，勾选 **Karpenter**。

> **注意：**
> 如果选择部署了 Karpenter，CA 将不会被部署。

4. 集群创建完成后，请参考 存量集群迁移方案中步骤3-5 部署 NodePool 资源、TKEMachineNodeClass 资源及工作负载。

tke-log-agent 说明

最近更新时间：2024-08-28 10:28:31

简介

组件介绍

tke-log-agent 是 Kubernetes 集群日志采集组件，用户可非侵入式采集容器标准输出日志、容器内日志以及节点日志。

部署在集群内的资源对象

Kubernetes 对象名称	类型	资源量	Namespace
tke-log-agent	Daemonset	0.21C126M	kube-system
cls-provisioner	Deployment	0.1C64M	kube-system
logconfigs.cls.cloud.tencent.com	CustomResourceDefinition	-	-
cls-provisioner	ClusterRole	-	-
cls-provisioner	ClusterRoleBinding	-	-
cls-provisioner	ServiceAccount	-	kube-system
tke-log-agent	ClusterRole	-	-
tke-log-agent	ClusterRoleBinding	-	-
tke-log-agent	ServiceAccount	-	kube-system

使用场景

- 独立集群开启审计日志采集时，默认安装 tke-log-agent 并采集 apiserver 审计日志。
- 通过采集规则采集容器标准输出日志、容器内日志、节点日志。

组件原理

1. cls-provisioner 监听到用户创建了采集规则后，根据采集规则的配置信息，生成 CLS 侧采集配置同步到 CLS 侧服务端。
2. tke-log-agent 根据采集规则，映射日志目录到统一目录下。
3. loglistener 同步 CLS 服务端采集配置，并根据采集配置采集日志上报到 CLS 侧，详情请参见 LogListener 。

组件权限说明

> **说明：**
> **权限场景**章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考**权限定义**章节。

log-agent 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 只有开启了日志采集的标准集群会部署该组件，其他类型集群不会部署。
- 需要在主机目录下读写 metadata 文件，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
监听日志采集规则的变动	logconfig/logconfigpro	watch/patch/get

获取节点的 runtime 类型	node	list/watch/get
采集标准输出日志/容器内日志时需要采集特定 namespace 下的 pod 日志	namespace/pod	list/watch/get
采集容器内日志采集时需要获取容器日志的实际存储路径	PV/PVC	list/watch/get
	SC	get
采集 workload 相关日志	工作负载	list/watch/get

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tke-log-agent
rules:
  - apiGroups: ["cls.cloud.tencent.com"]
    resources: ["logconfigs","logconfigpros"]
    verbs: ["list", "watch", "patch","get"]
  - apiGroups: [""]
    resources: ["pods", "namespaces", "nodes", "persistentvolumeclaims","configmaps","persistentvolumes"]
    verbs: ["list", "watch", "get"]
  - apiGroups: ["apps"]
    resources: ["daemonsets","replicasets","deployments","statefulsets"]
    verbs: ["list", "watch", "get"]
  - apiGroups: ["batch"]
    resources: ["jobs","cronjobs"]
    verbs: ["list", "watch", "get"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get"]
```

cls-provisioner 权限

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
把 log config 的规则内容同步到 CLS 侧	logconfig	list/watch/patch/update

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cls-provisioner
rules:
- apiGroups:
  - cls.cloud.tencent.com
  resources:
  - logconfigs
  verbs:
  - list
  - watch
  - patch
```

```
    - update
  - apiGroups:
    - '*'
    resources:
    - events
    - configmaps
    verbs:
    - create
    - patch
    - update
```

相关链接

- 开启日志采集
- 通过控制台配置日志采集
- 通过 YAML 配置日志采集
- 使用 CRD 采集日志到 Kafka

tke-monitor-agent 说明

最近更新时间：2024-01-29 15:18:51

组件介绍

为了提升容器服务基础监控及告警服务的稳定性，腾讯云升级了基础监控服务架构。新版基础监控会在用户集群的 kube-system 命名空间下部署一个 DaemonSet，名称为 tke-monitor-agent，并创建对应的认证授权 K8s 资源对象 ClusterRole、ServiceAccount、ClusterRoleBinding，名称均为 tke-monitor-agent。

组件作用

该组件会采集每个节点上容器、Pod、节点、以及官方组件的监控数据，该数据源用于控制台基础监控指标展示、指标告警和基于基础指标的 HPA 服务。部署该组件，可极大程度改善之前因基础监控运行不稳定导致的监控数据无法正常获取的问题，获得更稳定的监控、告警及 HPA 服务。

组件影响

- 部署该组件不会影响集群服务的正常运行。
- 如果您的**节点资源分配不合理**或者**节点负载过高、节点资源不够**，部署基础监控组件时可能会导致监控组件 DaemonSet tke-monitor-agent 对应的 Pod 处于 **Pending**、**Evicted**、**OOMKilled**、**CrashLoopBackOff** 状态，这属于正常现象。对于 DaemonSet tke-monitor-agent 对应 Pod 出现的意外状态描述如下：
    - **Pending** 状态：表示集群的节点上没有足够的资源进行 Pod 的调度，尝试将 DaemonSet tke-monitor-agent 的资源申请量设置为0，可将 Pod 调度上去（详情见 Pod 处于 pending 状态的排错指南）。
    - **Evicted** 状态：DaemonSet tke-monitor-agent 的 Pod 如果处于此状态，可能是您的节点资源不够或者节点本身负载就已经过高，可通过如下方式去查看具体的原因，并进行排查和解决：
        - 执行 `kubectl describe pod -n kube-system <podName>`，通过 Message 字段的描述信息来查看具体被驱逐的原因。
        - 执行 `kubectl describe pod -n kube-system <podName>`，通过 Events 字段描述的信息来查看具体被驱逐的原因。
    - **CrashLoopBackOff** 或者 **OOMKilled** 状态：可以通过 `kubectl describe pod -n kube-system <podName>` 查看是否为 OOM，如果是，可以通过提升 memory limits 的数值解决，limits 值最多不超过100M，如果设置为100M仍然出现 OOM，请 提交工单 来寻求帮助。
    - **ContainerCreating** 状态：执行命令 `kubectl describe pod -n  kube-system <pod 名称>`，查看 Events 字段。若显示如下内容：
    ```
    Failed to create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "<pod 名称>": Error response from daemon: Failed to set projid for /data/docker/overlay2/xxx-init: no space left on device
    ```
    ，则表明容器数据盘已满，清理节点上数据盘后即可恢复。

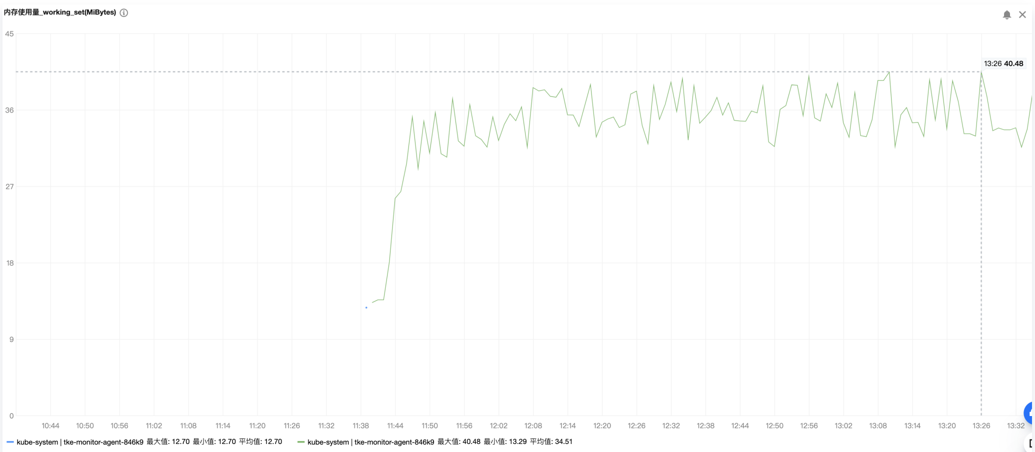
    > **说明：**
    > 如果以上描述未解决您的疑问，请 提交工单 来寻求帮助。

- 监控组件 DaemonSet（名称为 tke-monitor-agent）所管理的每个 Pod 的资源耗费情况和节点上运行的 Pod 数量和容器数量成正相关，下图为压测示例，内存和 CPU 占用量均很小。
  数据规模
  节点上有220个 Pod，每个 Pod 有3个容器。
  资源消耗

内存（峰值）	CPU（峰值）
40MiB 左右	0.01C

  CPU 使用量压测结果如下：

内存使用量压测结果如下：



组件权限说明

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
需要采集集群中 pod 数量和 pod 相关信息	replicasets、deployments和pods	list/watch
通过访问节点上 kubelet 的 /metrics 端口获取 cadvisor 的指标信息	nodes、nodes/proxy、nodes/metrics	list/watch/get
和 cluster-monitor 传递指标数据	services	list/watch
上报指标到 hpa-metrics-server	custommetrics	update

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tke-monitor-agent
rules:
```

```yaml
  - apiGroups: ["apps"]
    resources: ["replicasets"]
    verbs: ["list", "watch"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["list", "watch"]
  - apiGroups: [""]
    resources: ["nodes", "nodes/proxy", "nodes/metrics"]
    verbs: ["list", "watch", "get"]
  - apiGroups: [""]
    resources: ["services"]
    verbs: ["list", "watch"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["list", "watch"]
  - apiGroups: ["monitor.tencent.io"]
    resources: ["custommetrics"]
    verbs: ["update"]
```

UserGroupAccessControl 说明

最近更新时间：2023-09-08 19:13:04

简介

组件介绍

UserGroupAccessControl 用户组访问控制组件，支持将 Kubernetes RBAC 权限管理机制对接腾讯云 CAM 用户组，便于对子账号进行细粒度的访问权限控制。

部署在集群内的 Kubernetes 对象

kubernetes 对象名称	类型	资源量	Namespaces
user-group-access-control	ServiceAccount	-	kube-system
user-group-access-control	ClusterRole	-	kube-system
user-group-access-control	ClusterRoleBinding	-	kube-system
user-group-access-control	Service	-	kube-system
user-group-access-control	ConfigMap	-	kube-system
user-group-access-control	Deployment	0.5C1G（针对新建）	kube-system

使用场景

CAM 用户组是多个相同职能的用户（子账号）的集合，可以实现批量的授权、设置订阅消息等功能，本组件适用于需要在 TKE 标准集群中为职能相同的子账号快速设置相同的 Kubernetes 对象访问权限场景。

限制条件

- 该功能仅支持 TKE 托管集群。
- K8S 集群版本 ≥ 1.16。

操作步骤

> **说明：**
> 如果您需要使用 UserGroupAccessControl 组件，请通过 提交工单 申请开通。

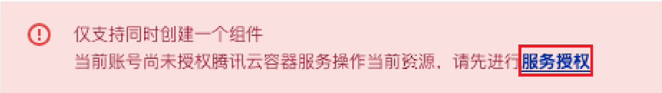
步骤1：新建用户组

您需要在访问管理 CAM 服务中完成用户组创建，操作详情请参见 新建用户组。若您已有用户组，可跳过此步骤。

步骤2：安装组件

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**，在**组件管理**页面中单击**新建**。
4. 在**新建组件**页面，选择**认证授权**模块，勾选 **UserGroupAccessControl 组件**。如下图所示：

5. 单击**服务授权**。为让容器服务可以读取到您账号下的用户组信息，需要您在组件安装时完成角色 “TKE_QCSRole” 关联预设策略 “QcloudAccessForTKERoleInGroupsForUser” 的引导授权。如下图所示：

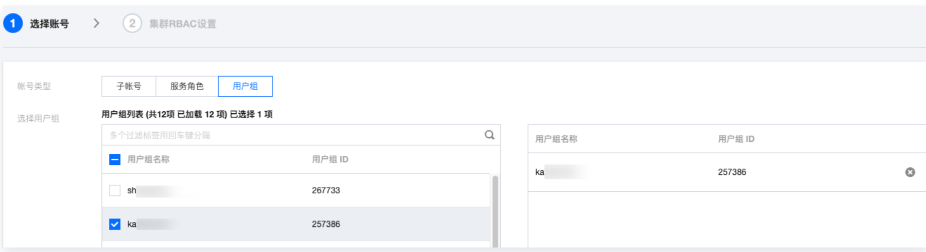


在**服务授权**页面中，确认角色名称和授权策略，单击**同意授权**。

6. 返回**新建组件**页面，单击**完成**。组件创建完成后，您可以在**组件管理**页面查看组件详情。

步骤3：为用户组创建角色并绑定策略

1. 选择左侧导航栏中的**授权管理 > ClusterRole**，在 ClusterRole 页面单击 **RBAC 策略生成器**。
2. 在**新建 ClusterRole** 中，账号类型选择**用户组**，并勾选用户组。如下图所示：



3. 单击**下一步**。在**集群 RBAC 设置**中，为指定用户组设置 Kubernetes 对象资源权限。
4. 单击**完成**。

步骤4：查看角色绑定策略

选择左侧导航栏中的**授权管理 > ClusterRoleBinding**，在 ClusterRoleBinding 中查看以用户组 ID 命名开头的权限策略信息。如下图所示：

> **说明：**
> 若您需要变更腾讯云资源的操作权限，如组内子账号迁移、增加/删除云产品的操作权限，您仅需在 CAM 侧的用户组进行修改，权限变更的结果会在您创建的角色绑定策略中即时生效。操作详情请参见 为用户组添加/移除用户 。

应用管理

概述

最近更新时间：2023-05-17 15:40:38

应用功能是指腾讯云容器服务（Tencent Kubernetes Engine，TKE）集成的 Helm 3.0 相关功能，为您提供创建 helm chart、容器镜像、软件服务等各种产品和服务的能力。已创建的应用将在您指定的集群中运行，为您带来相应的能力。

应用相关操作

- 应用管理
- 本地 Helm 客户端连接集群

使用应用

最近更新时间：2025-05-26 17:00:01

本文介绍如何通过容器服务控制台对应用进行创建、更新、回滚、删除操作。

说明事项

应用管理仅支持 Kubernetes 1.8 版本以上集群。

操作步骤

创建应用

1. 登录容器服务控制台，选择左侧导航栏中的 应用市场 。
2. 在**应用管理**页面，选择集群及地域，并单击**新建**。
3. 在**新建应用**页面，参考以下信息设置应用的基本信息。如下图所示：

主要参数信息如下：
- **应用名**：自定义应用名称。
- **来源**：可选择**应用市场**和**第三方来源**。详细配置见下表：

来源	配置项
应用市场	根据集群类型、应用场景进行 Chart 筛选。选择适用的应用包及 Chart 版本，并可编辑参数。
第三方来源	• Chart 地址：支持 Helm 官方或自建 Helm Repo 仓库。注意必须设置为以 http 开头 .tgz 结尾的参数值。本文示例为 http://139.199.162.50/test/nginx-0.1.0.tgz。 • 类型：提供**公有**及**私有**两种类型，请按需选择。 • 参数：按需进行参数编辑。

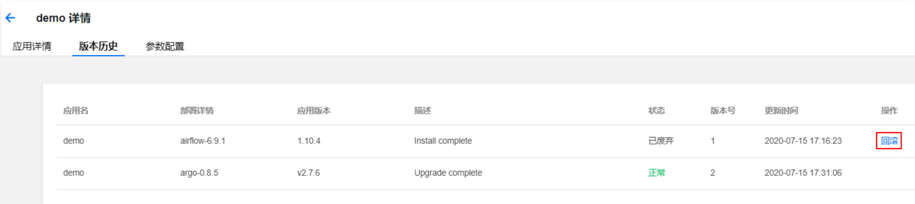
4. 单击**完成**即可创建应用。

更新应用

1. 前往 容器服务控制台，选择左侧导航栏中的**应用市场**。
2. 在**应用管理**页面中，选择应用所在行右侧的**更新应用**。
3. 在**更新应用**页面，按需进行关键信息配置，并单击**完成**。

回滚应用

1. 前往 容器服务控制台，选择左侧导航栏中的**应用市场**。
2. 在**应用管理**页面中，选择需要更新的应用，进入该应用详情页面。
3. 在应用详情页中，选择**版本历史**页签，单击需回滚版本所在行右侧的**回滚**。如下图所示：



4. 在**回滚应用**页面，单击**确认**即可。如下图所示：

删除应用

1. 前往 容器服务控制台，选择左侧导航栏中的**应用市场**。
2. 在**应用管理**页面中，选择应用所在行右侧的**删除**。
3. 在**删除应用**页面，单击**确定**即可。

本地 Helm 客户端连接集群

最近更新时间：2024-08-23 16:20:41

操作场景

本文档指导您通过本地 Helm 客户端连接集群。

操作步骤

下载 Helm 客户端

依次执行以下命令，下载 Helm 客户端。关于安装 Helm 的更多信息，请参见 Installing Helm。

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
```

```
chmod 700 get_helm.sh
```

```
./get_helm.sh
```

配置 Helm Chart 仓库（可选）

1. 执行以下命令，配置 Kubernetes 官方仓库。

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

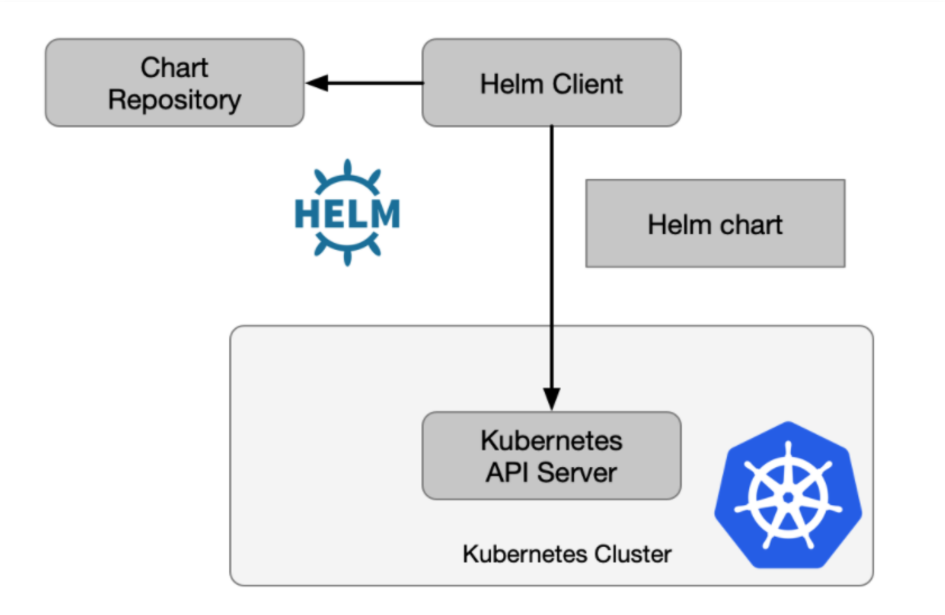
2. 执行以下命令，配置腾讯云应用市场。

```
helm repo add tkemarket https://market-tke.tencentcloudcr.com/chartrepo/opensource-stable
```

3. 配置 TCR 私有 Helm 仓库。

连接集群

Helm v3对比 Helm v2已移除 Tiller 组件，Helm 客户端可直接连接集群的 ApiServer，应用相关的版本数据直接存储在 Kubernetes 中。如下图所示：



Helm Client 使用 TKE 生成的客户端证书访问集群，具体操作步骤如下：

1. 通过容器服务控制台或 API 获取可用公网或内网访问的 Kubeconfig。

2. 连接目标集群可参考以下两种方式：
    - 使用上述获取的 kubeconfig，对 Helm Client 所在机器的 kubectl config use-context 进行配置。
    - 执行以下命令，通过指定参数的形式访问目标集群。

```
helm  install ....  --kubeconfig [kubeconfig所在路径]
```

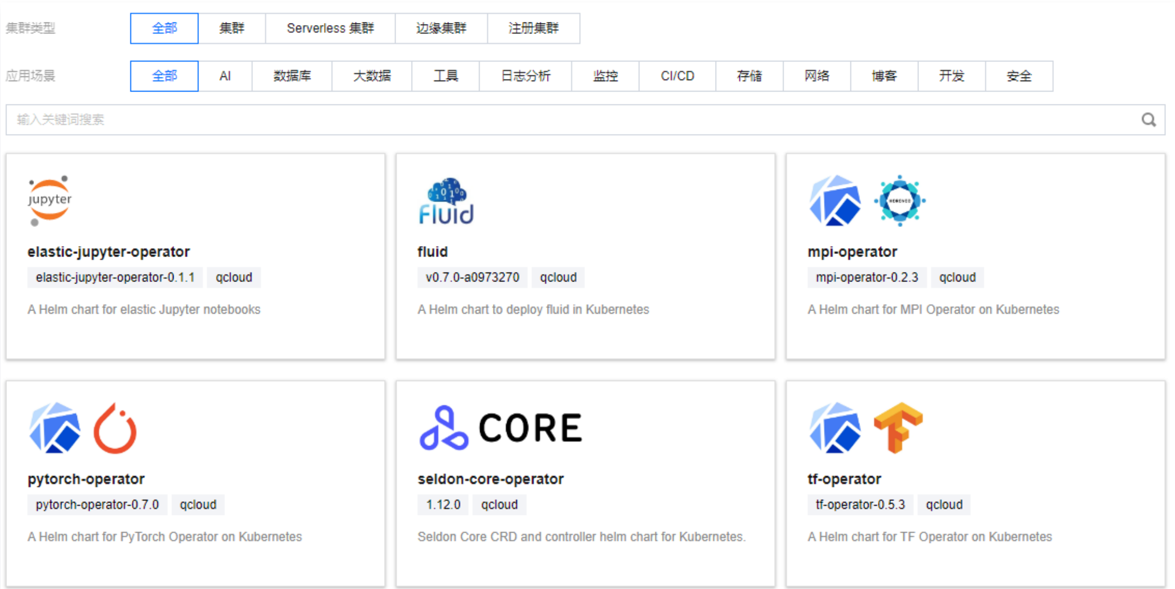
应用市场

最近更新时间：2023-11-28 15:41:37

腾讯云容器服务（Tencent Kubernetes Engine，TKE）应用市场按照集群类型、应用场景等分类方式，为您提供多种产品和服务。例如 helm chart、容器镜像、软件服务等。本文介绍如何通过容器服务控制台中的应用市场，快速完成应用创建。

查看应用

1. 登录 容器服务控制台，选择左侧导航栏中的**应用市场**。
2. 在**应用市场**页面中，可进行如下操作：
    - 筛选应用：可按照集群类型、应用场景或输入关键词进行应用筛选。
        - 集群类型：包含集群、 Serverless 集群、边缘集群和注册集群。
        - 应用场景：包含 AI、数据库、大数据、工具、日志分析、监控、CI/CD、存储、网络、博客、开发、安全。
    - 查看应用：单击需要查看的应用包，即可前往该应用详情页。



创建应用

1. 登录 容器服务控制台，选择左侧导航栏中的**应用市场**。
2. 在**应用市场**页面中按需选择应用包，并进入该应用详情页。
3. 在应用详情页中，单击基本信息模块中的**创建应用**。

4. 在**创建应用**页面中，按需配置并创建应用。如下图所示：

4.单击**创建**。

网络管理
容器网络概述

最近更新时间：2024-07-25 10:09:41

腾讯云容器服务 TKE 集成 Kubernetes 网络、腾讯云 VPC 以及 CLB 等各种云服务能力，提供高效稳定的容器网络。本文主要介绍 TKE 提供的 CNI 能力，您可以通过了解这些 CNI 网络方案，更合理地为您的应用选择合适的 CNI 网络插件。

容器网络 CNI

CNI (Container Network Interface) 网络插件是一种用于 Kubernetes 集群的网络解决方案，旨在提供灵活、可扩展和高性能的网络连接。CNI TKE 网络插件遵循 CNI 规范，允许 Kubernetes 集群与各种网络解决方案无缝集成，以满足不同的业务需求和性能要求。
网络插件需满足的以下约束：

- 每个 Pod 都有独立的 IP 地址，节点上的 Pod 可以不通过 NAT 和集群内的所有 Pod 通信。
- 节点上的代理服务（例如 Kubelet）可以和本节点上的所有 Pod 通信。

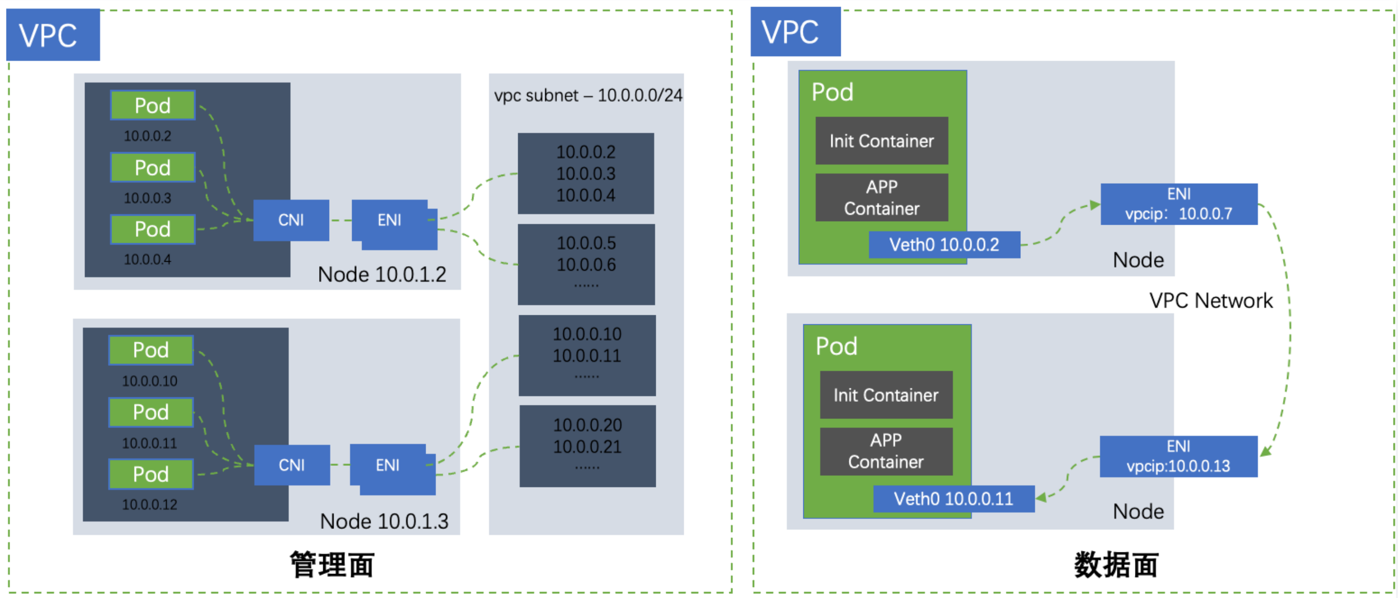
TKE 提供了3种网络 CNI 插件方案：**VPC-CNI**、**Global Router** 和 **Cilium-Overlay**，下文会详细介绍这三种网络方案。

VPC-CNI 网络方案介绍

VPC-CNI 是基于腾讯云 VPC 专有网络实现的容器网络 CNI 插件，可以将 VPC 的弹性网卡直接分配给 Pod，实现 Pod 之间的互联互通。这种方案充分复用 VPC 的云上网络资源，容器与节点分布在同一网络平面，Pod IP 为集群 IPAMD 组件分配的弹性网卡 IP，由于不需要经过节点上的网桥设备，不需要使用 VxLAN 等隧道技术封装报文，在网络性能、可观测性、限流、隔离等多层面都提供更好的支持，更加适合公有云场景。
TKE 推荐将 VPC-CNI 作为集群默认的网络方案。VPC-CNI 模式提供**共享网卡**和**独占网卡**两种模式，各自适用于不同的场景，您可以根据业务需求选择不同的网络模式。

- 共享网卡模式：多个 Pod 共享一张弹性网卡，IPAMD 组件为弹性网卡申请多个 IP 并绑定到不同的 Pod，此模式可以降低对弹性网卡资源的需求，并提升节点上 Pod 的部署密度。共享网卡模式可支持 固定 Pod IP 能力。详情请参见 多 Pod 共享网卡模式。
- 独立网卡模式：每个 Pod 会绑定独立的弹性网卡，此模式适合对网络性能有较高需求（高吞吐低延迟）的场景。但是会受机型影响，不同节点可使用的弹性网卡数量有限，单节点 Pod 密度更低。详情请参见 Pod 独占网卡模式。

VPC-CNI 方案的基本原理图如下所示：



优势特性

- 容器网络和 VPC 子网处于同一网络平面，可以使用 VPC 相应的云上网络的能力，例如 EIP 和安全组等。
- 无需按节点预分配 CIDR 地址段，避免浪费 CIDR 内的 IP 地址。
- 数据面转发无需网桥，网络转发性能更高，大约提升10%，适用于对网络性能要求较高的场景。
- 支持 Pod 固定 IP，适用于依赖容器固定 IP 的场景。

适用场景

- 时延敏感型：
  - 中间件与微服务：较低的网络延迟，可以提升大规模中间件以及微服务部署的网络性能。
  - 在线游戏、视频直播类应用：较高的网络吞吐和较低的网络延迟，同时可以使用 VPC 的云上网络资源，能够更好地支持网络密集型业务。
- 传统架构迁移：传统架构迁移到容器平台需要保证 IP 不变及针对 IP 做安全策略限制。
- 需要为容器单独绑定独立的安全组策略，使用独立网卡模式绑定安全组。

注意事项

- Pod 需要从 VPC 子网中分配 IP，建议该子网独立分配给 Pod 使用，不要与其他云上资源共用（如云服务器、负载均衡等）。
- 集群内的节点需要和容器子网处于相同可用区。
- 节点上可调度的Pod 数量受限于节点所支持弹性网卡能绑定 IP 的最大数量和弹性网卡数量。配置越高的机器可支持的弹性网卡数量越多，具体数量可以查看节点的 Allocatable 配置。

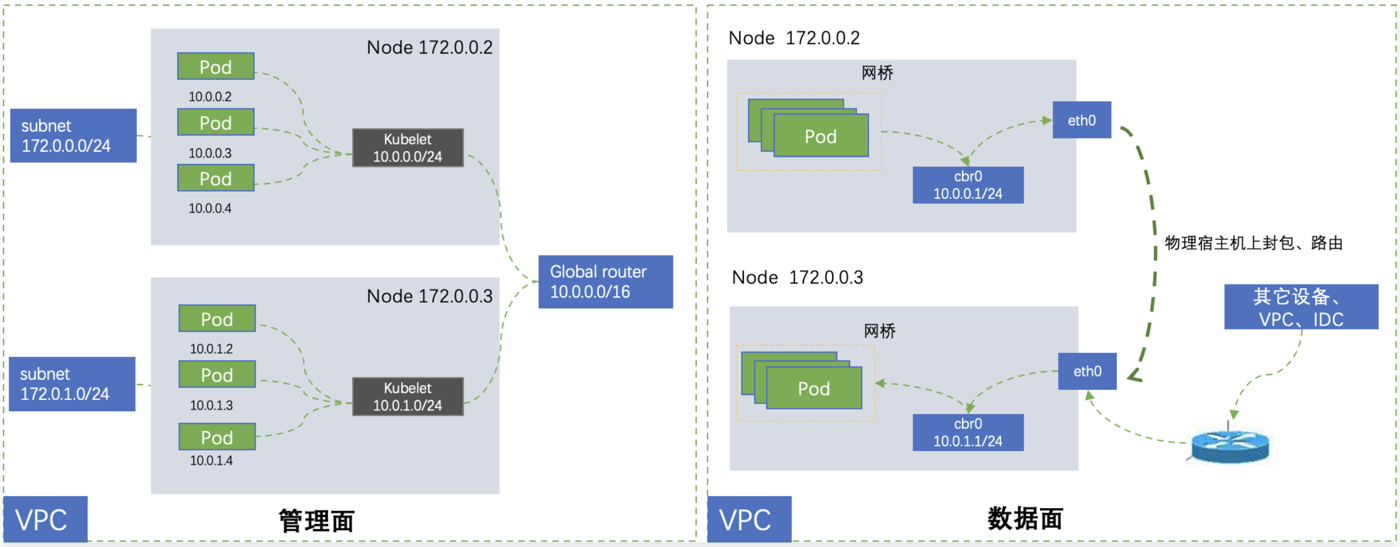
Global Router 网络方案介绍

Global Router 方案原理

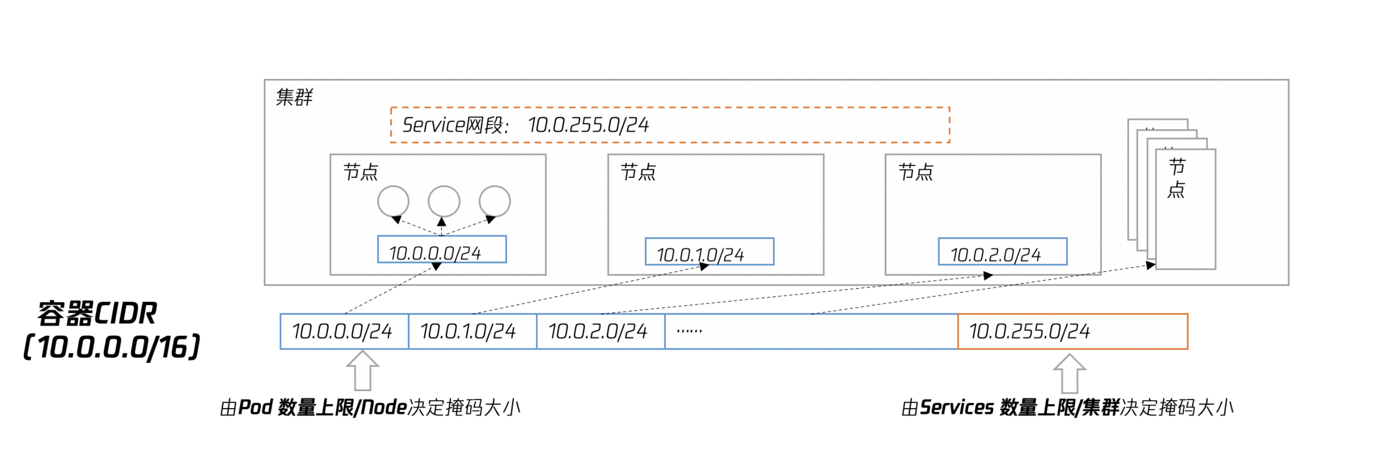
Global Router 网络方案是 TKE 基于腾讯云网络全局路由能力实现的容器网络 CNI。集群会为每个节点分配一个 Pod CIDR，用于给该节点上的 Pod 分配 IP 地址，Pod CIDR 独立于 VPC CIDR，每个节点上的 Pod 具有独立且不重叠的 IP 地址。不同节点的 Pod CIDR 信息会通过全局路由方式下发到 VPC，实现跨节点 Pod 的互访。
该网络模式特征包含以下几点：

- 容器 Pod CIDR 段不占用 VPC CIDR 段，并且会预分配给节点，容器 IP 从节点预分配的 Pod CIDR 网段中获取。
- 容器路由会直接发布给 VPC，跨节点 Pod 之间互访直接通过全局路由转发。
- 无需使用 VxLan 封装报文。

Global Router 方案的基本原理图如下所示：



容器 IP 分配机制



- 集群的每一个节点会根据设置的集群 Pod CIDR，以及“单节点 Pod 数量上限”计算出每个节点需要分配的 Pod CIDR 网段，用于该节点下 Pod 的 IP 地址分配。
- 集群的 Service 网段会根据用户设置的“集群内Service数量上限”选用集群 Pod CIDR 中最后一段CIDR 地址用于 Service 的 IP 地址分配。
- 节点释放后，该节点使用的容器 Pod CIDR 网段也会释放回 IP 段池。
- 扩容节点按顺序循环选择容器 Pod CIDR 大段中可用的 IP 网段。

优势特性以及适用场景

- 使用简单，Pod 启动速度较快。
- 适合规模相对固定，对于 IP 分配和网络性能没有特殊需求的简单业务场景。

注意事项

- 集群 VPC CIDR 和集群 Pod CIDR 网段不能重叠。
- 同一 VPC 内，不同集群的 Pod CIDR 网段不能重叠。
- 容器网络和 VPC 路由重叠时，优先在容器网络内转发。
- 不支持固定 Pod IP。

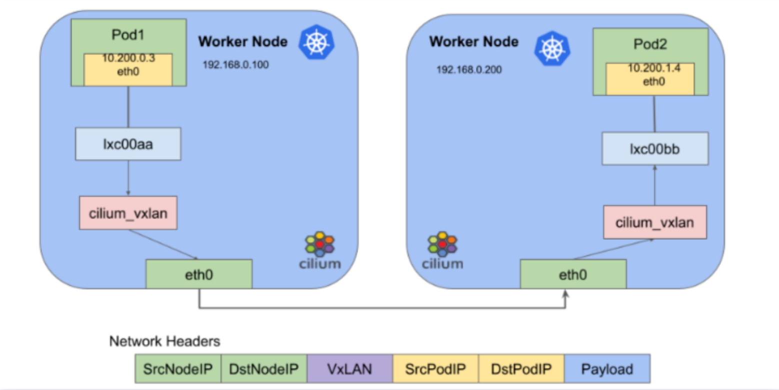
Cilium-Overlay 网络方案介绍

> **注意：**
> 由于 Cilium-Overlay 网络方案仅支持注册节点场景，当前不支持只存在云上节点场景，只有云上节点的情况请使用推荐的 VPC-CNI 网络方案。

Cilium-Overlay 网络方案是容器服务 TKE 基于 Cilium VXLan 实现的容器网络插件，实现分布式云场景中，用户云下节点添加到 TKE 集群时，云上云下 Pod 之间互通的需求。该网络模式特征如下：

- 云上节点和云下节点共用指定的容器网段。
- 容器网段分配灵活，容器 IP 段不占用 VPC 的其他网段。
- 使用 Cilium VXLan 隧道封装协议构建 Overlay 网络。

云上 VPC 网络和注册节点 IDC 网络通过云联网互通后，跨节点 Pod 访问原理如下图所示：



使用限制

- 使用 Cilium VXLan 隧道封装协议，有10%以内的性能损耗。
- Pod IP 在集群外不能直接访问。
- 需从指定子网获取2个 IP 创建内网负载均衡，满足 IDC 中第三方节点访问 APIServer 和云上公共服务。
- 集群网络和容器网络网段不能重叠。
- 不支持固定 Pod IP。

容器集群网络方案选型

最近更新时间：2024-05-24 15:17:01

容器网络 CNI

容器网络接口（CNI，Container Network Interface）是一种用于配置容器网络的插件化标准。它定义了一组规范，可以与容器编排系统（如 Kubernetes 等）无缝集成，实现容器在多个主机之间的互相通信。CNI 提供基于IP的虚拟网络、IP 分配等功能，可以满足各种复杂场景下对容器 Pod 对网络的需求。

TKE 容器产品提供 “VPC-CNI”、“Global Router” 以及 “Cilium-Overlay” 三种 CNI 网络方案。

> **注意：**
> TKE 推荐用户在公有云场景下使用 “VPC-CNI” 网络方案，在注册节点场景下使用 “Cilium-Overlay” 网络方案。

VPC-CNI 网络方案

VPC-CNI 网络方案是 TKE 基于腾讯云 VPC 实现的容器网络接口 CNI，可以将 VPC 原生的弹性网卡直接分配给 Pod，实现 Pod 之间的网络互联，适用于对时延有较高需求的场景。该网络模式下，容器与节点分布在同一网络平面，容器 IP 为 IPAMD 组件分配的弹性网卡 IP。详情请参见 VPC-CNI 方案介绍。

Global Router 网络方案

Global Router 网络方案是 TKE 基于腾讯云 VPC 的全局路由能力实现的容器网络接口 CNI。其中 Pod CIDR 网段独立于 VPC 的 CIDR 网段，不同节点的 Pod CIDR 信息会通过全局路由方式下发到 VPC，实现跨节点 Pod 的互访。详情请参见 Global Router 方案介绍。

Cilium-Overlay 网络方案

Cilium-Overlay 网络方案是 TKE 基于 Cilium VXLan 实现的容器网络插件，实现分布式云场景中，注册节点添加到 TKE 集群的网络管理。详情请参见 Cilium-Overlay 方案介绍。

TKE 网络方案对比

网络方案	VPC-CNI（推荐）	Global Router	Cilium-Overlay
基本介绍	• Pod 网络地址从 VPC子网地址中划分，建议独占一个子网。 • 可以使用 VPC 云上网络能力（EIP/CLB/安全组等）。	• Pod 网段独立于 VPC 的子网网段。 • 跨节点 Pod 之间互访直接通过全局路由转发。	• 容器网络与节点网络不在同一平面。 • 容器网络是基于节点网络的 Overlay 网络。
优势特点	• 无需按节点分配 CIDR 地址段，IP 地址不浪费。 • 数据面转发减少网桥，网络转发性能更高。 • 支持 Pod 固定 IP，适用于需要依赖容器固定 IP 的场景。	容器网段独立于 VPC 网段，使用简单，Pod 启动速度较快。	• 容器网段地址独立于 VPC 网段，地址充裕、扩展性强。 • Pod 网络之间通过 VxLan 封装报文，适合云上云下节点互通，兼容性更好。
使用场景	• 对网络时延有较高要求的场景。 • 传统架构迁移到容器平台，依赖容器有固定 IP 的场景。 • 适合有特殊网络安全策略的业务，例如数据库服务下需要特殊安全组策略。	• 业务场景简单，对于 IP 分配和网络性能没有特殊需求。 • 节点不支持弹性网卡场景。	仅适用于注册节点场景。
使用限制	• 容器网络与节点网络属于同一个 VPC，IP 地址资源有限。 • 节点内容器数量受弹性网卡和弹性网卡可分配 IP 数量的限制。 • 固定 IP 模式不支持 固定IP Pod 跨可用区调度。	• 容器网络地址与节点网络地址不可冲突。 • 专线、对等连接及云联网等互通场景需要额外配置。 • 不支持固定 Pod IP。	• Pod IP 在集群外不能直接访问。 • 需从指定子网获取2个 IP 创建内网负载均衡，满足 IDC 中注册节点访问 APIServer 和云上公共服务。 • 不支持固定 Pod IP。
是否支持 CLB 直通	支持，详情请参见 CLB 直通 Pod Service	支持（需提交白名单申请），详情请参见 CLB 直通 Pod Service	不支持

是否支持固定ip	支持	不支持	不支持
IPv4/IPv6双栈	支持	不支持	不支持
指定子网分配ip	支持	不支持	不支持
扩容 Pod 网段	支持	支持（暂未产品化）	不支持
设置 Pod 安全组	支持	不支持	不支持
Pod绑定EIP	支持	不支持	不支持
Pod 访问公网	支持（NAT、EIP）	支持（IP 伪装）	支持

容器集群网络规划

最近更新时间：2024-06-12 10:32:42

介绍

在 TKE 中创建集群时，您需要根据具体的业务情况进行集群、VPC、子网以及节点可用区的规划，本文将介绍 VPC 环境下集群内各个地址的作用，以及如何进行集群网络规划。

基本概念

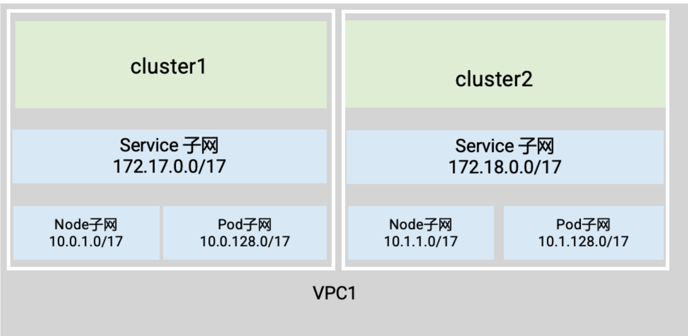
- Node 子网（节点网段）：在 TKE 集群中，worker 节点的 IP 分配的网段。
- Pod 子网（容器网段）：在 TKE 集群中，pod 的 IP 分配的网段。
- Service 子网（service 网段）：在 TKE 集群，集群内 ClusterIP 或者 Loadbalancer 类型的 service 相应的 IP 网段。

集群与 vpc 规划

在使用 TKE 容器服务时，您首先需要根据业务的需要进行 VPC 的规划，具体可以分为两种情况。

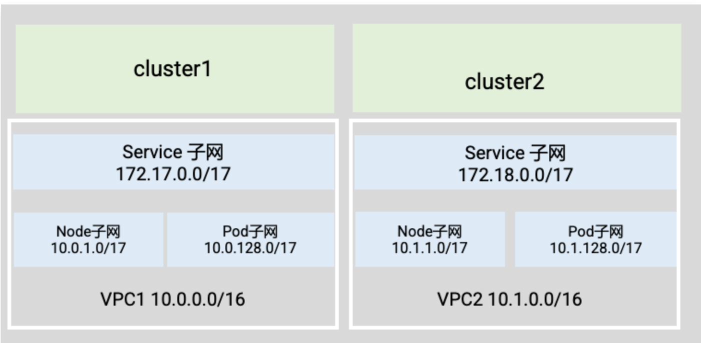
多个 TKE 集群使用同一个 vpc

- 在创建 TKE 集群时，需要规划 Pod 子网，Node 子网，Service 子网。
- 需要保证每个 TKE 集群的 Pod 子网，Node 子网不重合，且不和其他 TKE 集群重合。
- Service 子网不能和 Node 子网以及 Pod 子网重合，Service 子网只能在集群内使用，不能在集群外使用。
- 每个 TKE 集群中的节点将从集群的 Node 子网中分配 IP，Pod 从集群的 Pod 子网中分配 IP，同一个 VPC 内的 TKE 集群的 Pod 和 Node 可以和其他集群的 Pod 以及 Node 互通。



多个 TKE 集群使用不同的 VPC

在创建 TKE 集群时，为不同的 TKE 集群选择不同的 VPC，在单个 VPC 内规划 Node 子网，Pod 子网，和 Service子网时，需要保证三个子网不重合。当需要多个 VPC 的集群互通时可用通过云联网实现多个 VPC 打通，不同集群的 Node 子网和 Pod 子网不可以重合。



VPC-CNI 模式单集群子网与可用区规划

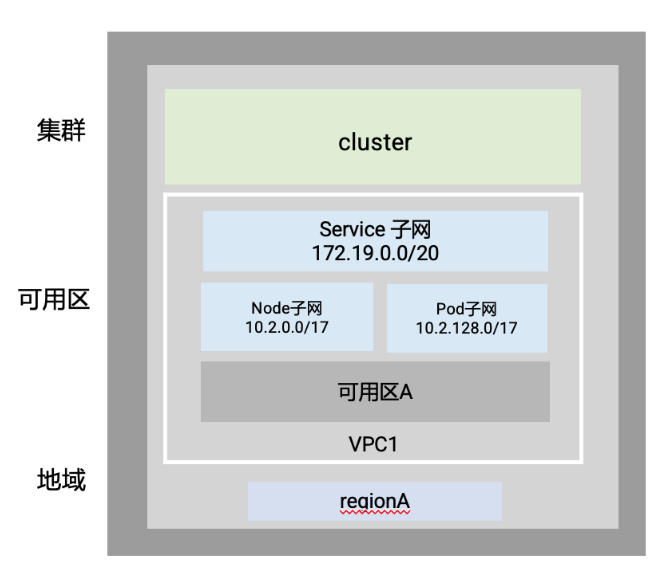
在创建单个 TKE 集群时，集群内的所有节点必须在一个 VPC 内。
- 规划 Pod 子网，所有的 Pod IP 从 Pod 子网中分配，当单个 Pod 子网 IP 不足时，可以通过添加子网来解决。

- 规划 Node 子网，由于 VPC-CNI 模式下同节点内主网卡和辅助网卡的 IP 必须在一个可用区的约束，当创建包含多个可用区 worker 节点的集群时，需要添加按需规划多个可用区的 Node 子网。
- 规划 Service 子网，用于集群内通过 ClusterIP 方式进行服务的访问，如果有多集群互通的需求，建议不同集群的 Service 子网不冲突。

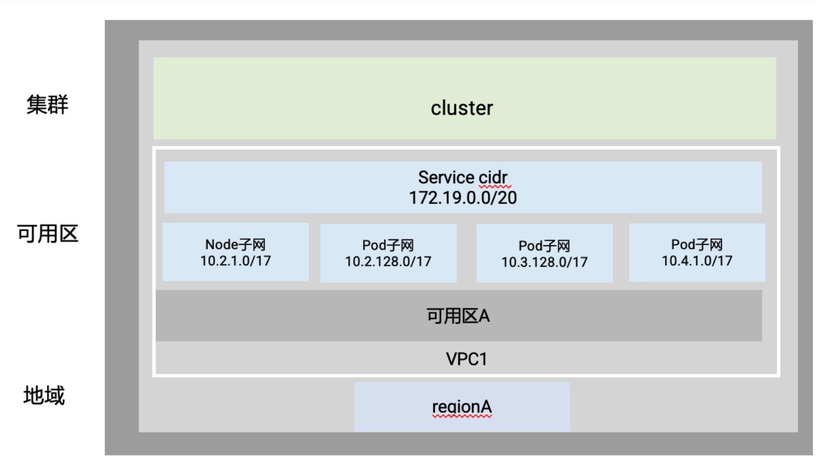
在创建 TKE 集群时可以按需规划包含单可用区节点的集群和包含多可用区节点的集群。

单可用区集群网络规划

创建单个可用区节点的 TKE 集群时，需要选择要创建集群的地域，VPC 信息，并且选择 Pod 子网和 Service 子网，且 Servcie 子网和 Pod 子网不重合，在添加节点时只能添加 Pod 子网所在可用区的节点。



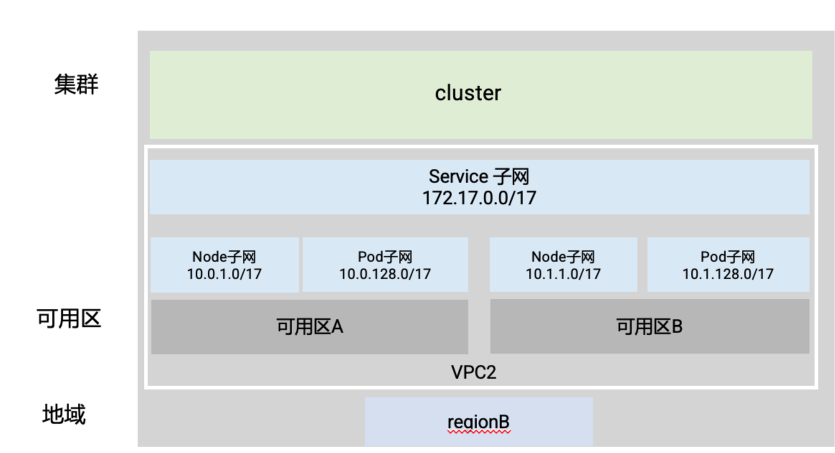
当 Pod 子网容量不足时，可以通过添加 Pod 子网来进行扩容。



多可用区集群网络规划

创建多个可用区节点的 TKE 集群时，需要选择要创建集群的地域，VPC 信息，并且选择多个可用区的 Pod 子网和 Servcie子网，且 Servcie 子网和 Pod 子网不重合，在添加节点时只能添加 Pod 子网所在可用区的节点。

如果需要新增包含其他可用区的节点时，需要先规划新增可用区的 Pod 子网和 Node 子网，并在集群中添加包含该可用区的 Pod 子网，之后在该可用区中添加 worker 节点。

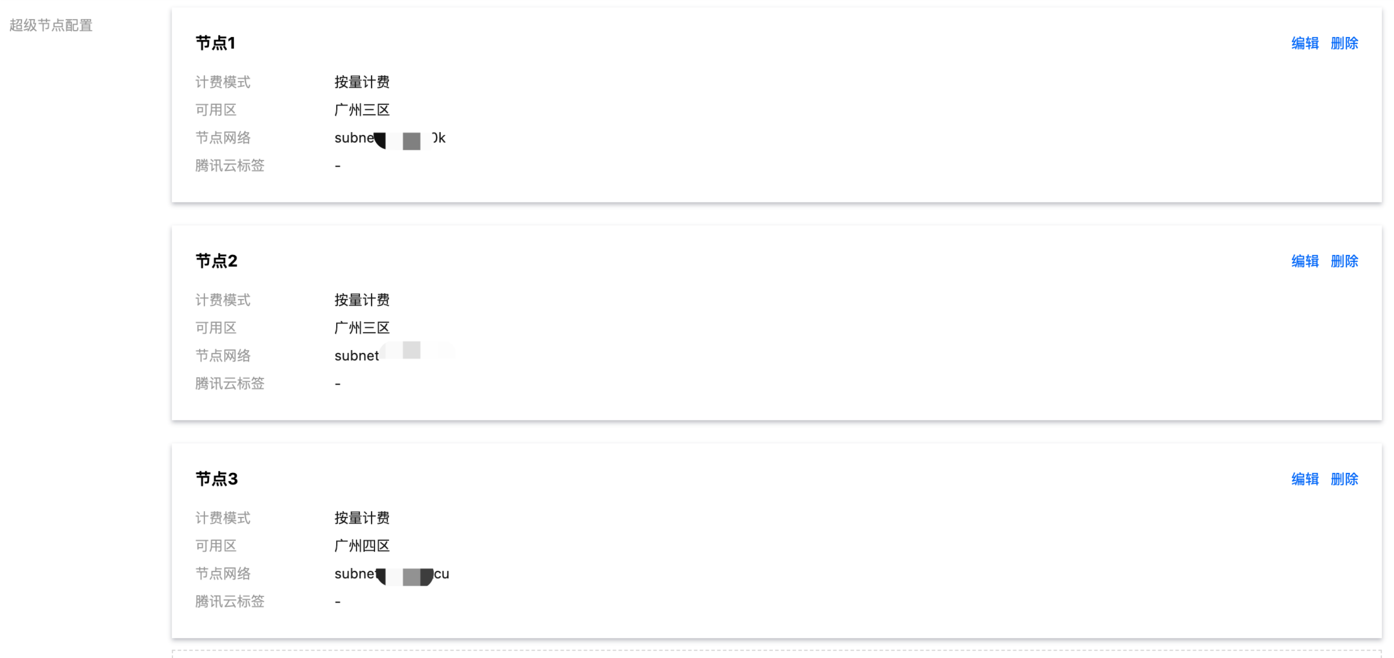


TKE 集群中添加超级节点池

用户可以在 TKE 集群中添加超级节点池来使用 Serverless Pod，添加超级节点池时，需要先确定要添加的可用区，并选择 Pod 子网，Serverless Pod 将从该子网分配。

当超级节点需要多个子网，请添加多个节点池并选择不同的子网，超级节点的 pod 从相应的节点池的子网中分配 ip。

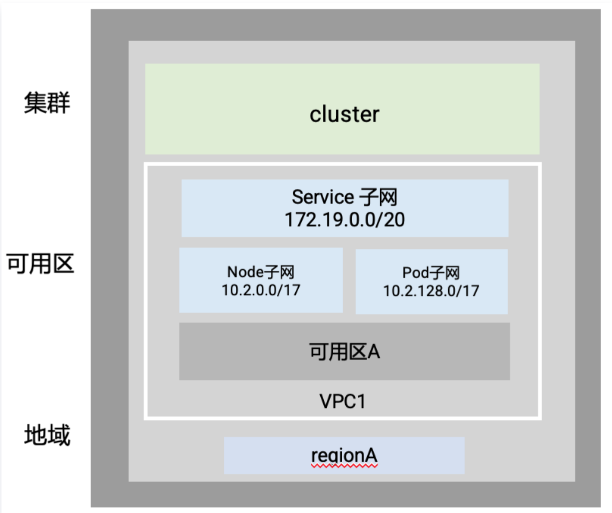
当需要使用多个可用区的 Serverless Pod 时，可以新建多个超级节点并选择不同的可用区来满足。



集群规划实践

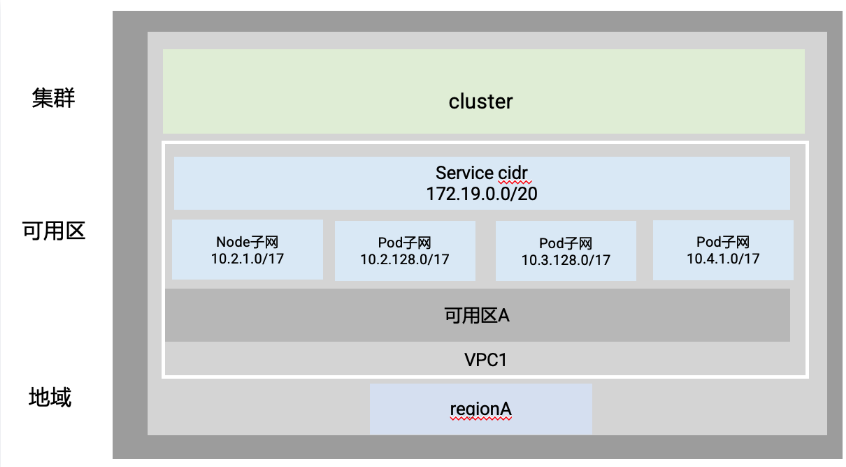
- 单可用区单子网集群

  用户在地域A vpc1 10.2.0.0/16 内单独划出两个17位的子网10.2.0.0/17和10.2.128.0/17 分别作为 Node 子网和 Pod 子网，同时规划172.19.0.0/20子网作为Service 子网。



- 单可用区多子网集群

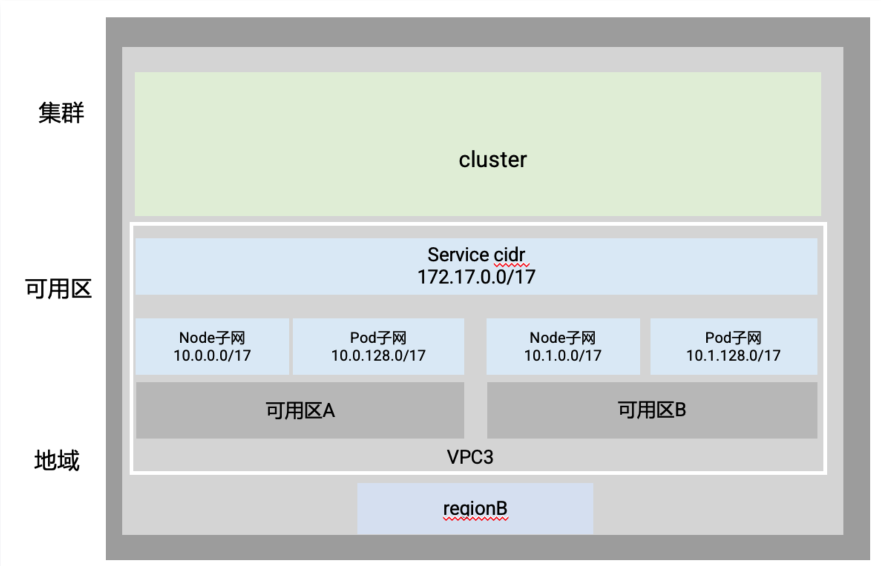
  用户在单集群 Pod IP 分配不足的情况下时，可以添加多个 Pod 子网，具体的集群形态如图：



- 多可用区集群

用户在地域B构建多可用区集群，其中部分节点从可用区A购买，且规划 Node 子网为10.0.0.0/17，Pod 子网为10.0.128.0/17, 另外一部分节点从可用区B购买，其中 Node 子网为10.1.0.0/17，Pod 子网为10.1.128.0/17，Service 子网为 172.17.0.0/17。

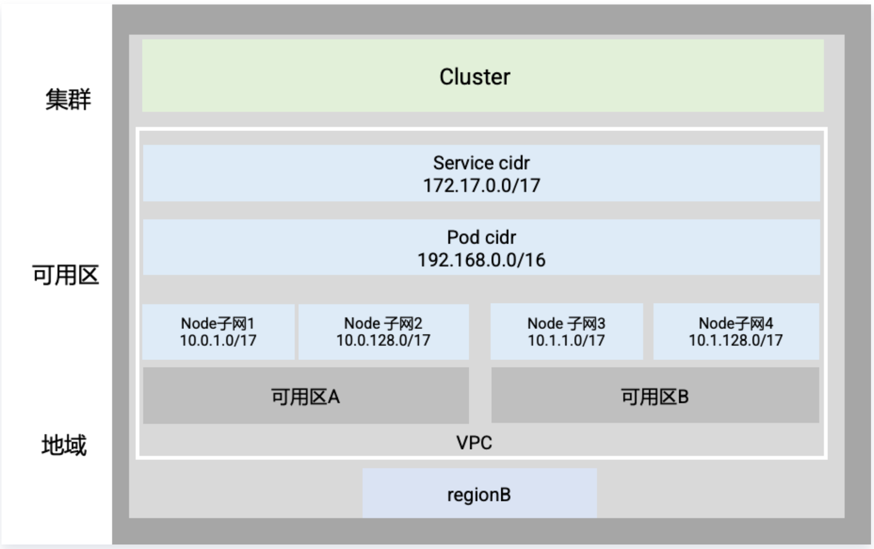
> **注意：**
> 由于 VPC 主网卡和辅助网卡必须在一个可用区的限制，可用区 A 节点的 pod 不能使用可用区 B 的 pod 子网，只能从可用区 A 的 Pod 子网中分配 IP。



集群类型	适用场景	优点	缺点
单可用区单集群	单元化业务部署，将微服务上下游部署在一个可用区，减少跨可用区部署的延迟。	集群规划简单，ip 分配不冲突，减少跨可用区访问延迟。	无法利用单集群拓扑打散特性，业务可自行做多集群单元化部署。
多可用区单集群	利用集群节点的 az 分布特性将 workload 打散在不同的可用区。	可以利用多 az 打散特性进行多 az 容灾。	跨可用区访问时会出现1-2ms延迟。
单可用区单集群（TKE 集群共享 Node 子网和 Pod 子网）	不推荐	-	容易出现 IP 抢占导致分配延迟和失败。

Global Router 模式集群网络规划

用户在创建 Global Router 模式的集群时，需要进行 Node 子网，Pod 子网和 Service 子网的规划。
Global Router 模式下 Pod 子网内 IP 和该 VPC 内其他子网互通，所以需要保证 Pod 子网不与 VPC 内其他子网重合，同时不能与 Servcie 子网重合。
确认单节点上 Pod 的数量上限，每个节点的容器子网会按照 Pod 子网/Pod 数量上限来预分配一个网段，之后节点上的 Pod 将从该预分配的网段分配 IP。


集群
可用区
地域
Cluster
Service cidr
172.17.0.0/17
Pod cidr
192.168.0.0/16
Node子网1
10.0.1.0/17
Node 子网2
10.0.128.0/17
Node 子网3
10.1.1.0/17
Node子网4
10.1.128.0/17
可用区A
可用区B
VPC
regionB

GlobalRouter 模式
GlobalRouter 模式介绍

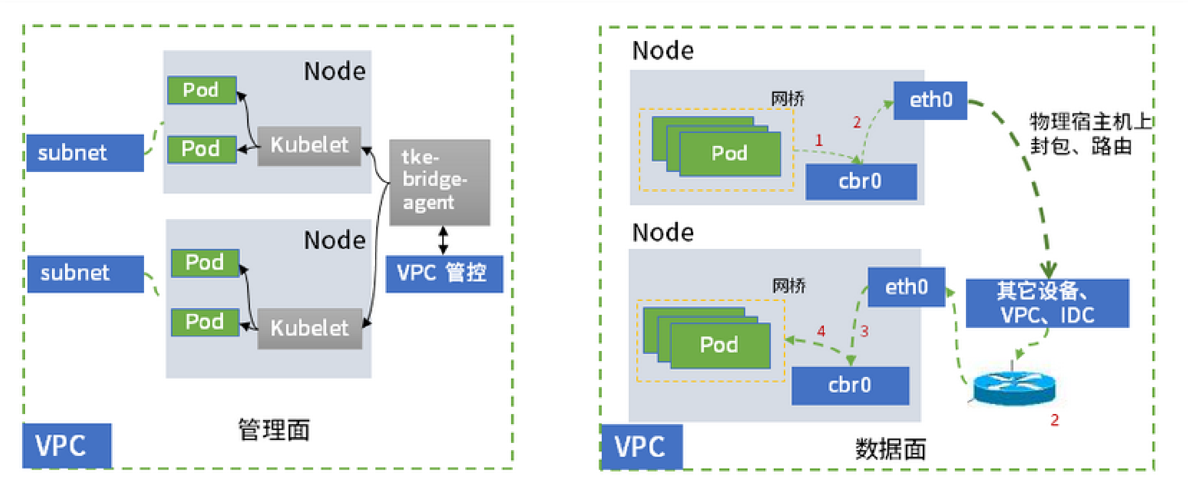
最近更新时间：2024-09-03 17:12:51

使用原理

GlobalRouter 网络模式是容器服务 TKE 基于底层私有网络 VPC 的全局路由能力，实现了容器网络和 VPC 互访的路由策略。该网络模式特征包含以下几点：

- 容器路由直接通过 VPC。
- 容器与节点分布在同一网络平面。
- 容器网段分配灵活，容器 IP 段不占用 VPC 的其他网段。

GlobalRouter 网络模式适用于常规场景，可与标准 Kubernetes 功能无缝使用。使用原理图如下所示：



使用限制

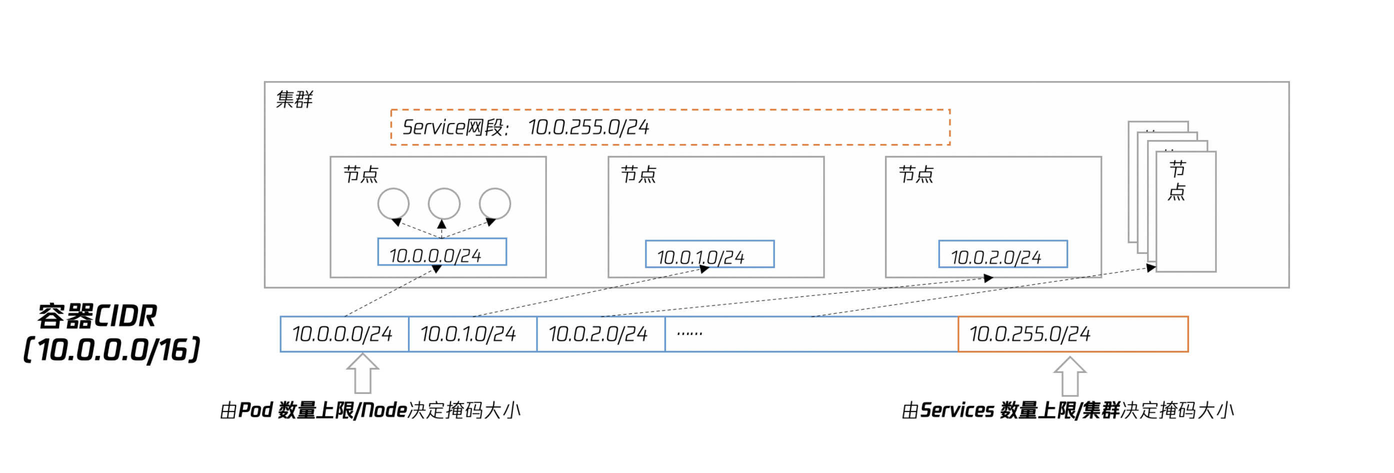
- 集群网络和容器网络网段不能重叠。
- 同一 VPC 内，不同集群的容器网络网段不能重叠。
- 容器网络和 VPC 路由重叠时，优先在容器网络内转发。
- 不支持固定 Pod IP。

容器 IP 分配机制

容器网络名词介绍请参见 基本概念 。

Pod IP 分配

工作原理如下图所示：



说明：

- 集群的每一个节点会使用容器 CIDR 中的指定大小的网段用于该节点下 Pod 的 IP 地址分配。
- 集群的 Service 网段会选用容器 CIDR 中最后一段指定大小的网段用于 Service 的 IP 地址分配。
- 节点释放后，使用的容器网段也会释放回 IP 段池。
- 扩容节点自动按顺序循环选择容器 CIDR 大段中可用的 IP 段。

同地域及跨地域 GlobalRouter 模式集群间互通

最近更新时间：2024-11-21 14:51:12

操作场景

在云计算环境中，用户通常会在不同地域或同一地域内部署多个集群，以满足业务的高可用性和灾备需求。然而，不同地域或同一地域内的集群之间默认情况下是无法直接通信的，为了解决这一问题，可以使用对等连接（Peering Connection）服务。对等连接是一种大带宽、高质量的云上资源互通服务，能够打通腾讯云上的资源通信链路。通过对等连接，您可以实现同地域和跨地域的不同集群互通，从而解决跨集群网络互通的问题。

前提条件

- 本文档以已创建集群并已添加节点为例。若未创建，请参考 创建集群 进行创建。
- 参考 创建对等连接通信 建立对等连接。请先确认对等连接已成功建立，且子机间能互通。若对等连接建立有问题，请排查**控制台路由表项、CVM 安全组、子网 ACL** 的设置是否有问题。

操作步骤

> **说明：**
> 如需实现跨地域集群间互通，请在执行完以下操作步骤后 在线咨询 申请打通容器路由，实现容器间互通。

获取容器的基本信息

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击需要设置集群间互通的集群 ID/名称，进入该集群的详情页面。例如，进入 A 集群的**基本信息**页面。
3. 记录基本信息中 A 集群**所在地域**、**节点网络**和**容器网络**的信息。
4. 返回集群管理页面，进入另一个集群（B 集群）的基本信息页面，并记录基本信息中 B 集群的**所在地域**、**节点网络**和**容器网络**的信息。

配置路由表

1. 登录私有网络控制台，选择左侧导航栏中的 对等连接 。
2. 在对等连接管理页面，记录对等连接的 **ID/名称**。如下图所示：

3. 选择左侧导航栏中的 子网，进入子网管理页面。
4. 单击对等连接本端指定子网的关联路由表。如下图所示：

5. 在关联路由表的**基本信息**页面，单击**新增路由策略**。
6. 在**新增路由**页面，设置路由信息。主要参数信息如下：
    - 目的端：输入 B 集群容器的网段。

- ○ 下一跳类型：选择**对等连接**。
- ○ 下一跳：选择已建立的对等连接。

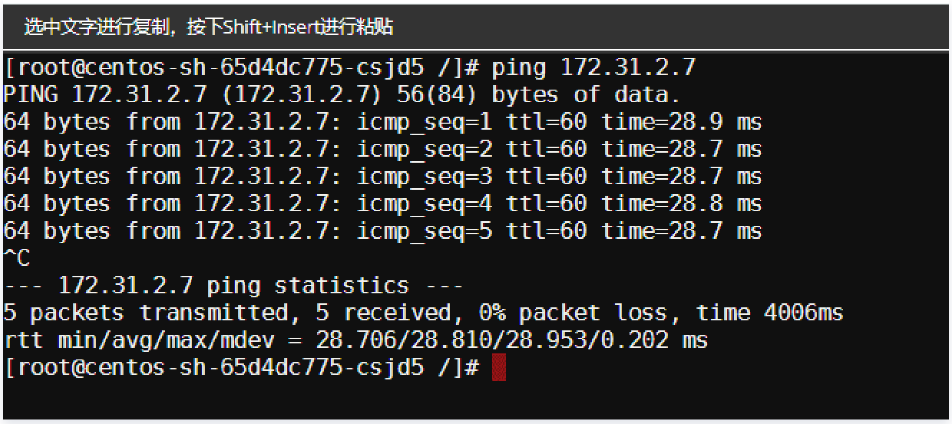
7. 单击**创建**，完成本端路由表的配置。
8. 重复执行 步骤3 – 步骤7，完成对端路由表的配置。

预期结果

- **同地域集群**：通过上述操作可直接实现容器之间的互通。
- **跨地域集群**：对等连接建立成功后，请 在线咨询 打通容器路由，实现容器之间的互通。

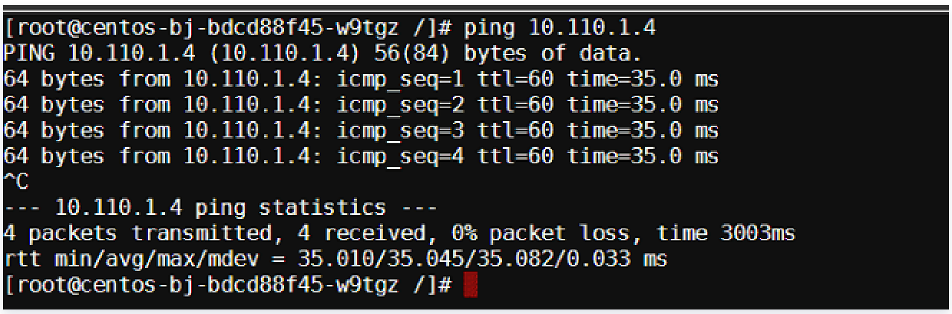
请参考 远程终端基本操作 登录容器，并按照以下步骤进行容器间的访问，验证容器间是否互通：

1. 登录集群 A 的容器，并在集群 A 的容器中访问集群 B 的容器。如下图所示：



```
选中文字进行复制，按下Shift+Insert进行粘贴
[root@centos-sh-65d4dc775-csjd5 /]# ping 172.31.2.7
PING 172.31.2.7 (172.31.2.7) 56(84) bytes of data.
64 bytes from 172.31.2.7: icmp_seq=1 ttl=60 time=28.9 ms
64 bytes from 172.31.2.7: icmp_seq=2 ttl=60 time=28.7 ms
64 bytes from 172.31.2.7: icmp_seq=3 ttl=60 time=28.7 ms
64 bytes from 172.31.2.7: icmp_seq=4 ttl=60 time=28.8 ms
64 bytes from 172.31.2.7: icmp_seq=5 ttl=60 time=28.7 ms
^C
--- 172.31.2.7 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 28.706/28.810/28.953/0.202 ms
[root@centos-sh-65d4dc775-csjd5 /]#
```

2. 登录集群 B 的容器，并在集群 B 的容器中访问集群 A 的容器。如下图所示：



```
[root@centos-bj-bdcd88f45-w9tgz /]# ping 10.110.1.4
PING 10.110.1.4 (10.110.1.4) 56(84) bytes of data.
64 bytes from 10.110.1.4: icmp_seq=1 ttl=60 time=35.0 ms
64 bytes from 10.110.1.4: icmp_seq=2 ttl=60 time=35.0 ms
64 bytes from 10.110.1.4: icmp_seq=3 ttl=60 time=35.0 ms
64 bytes from 10.110.1.4: icmp_seq=4 ttl=60 time=35.0 ms
^C
--- 10.110.1.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 35.010/35.045/35.082/0.033 ms
[root@centos-bj-bdcd88f45-w9tgz /]#
```


GlobalRouter 模式集群与 IDC 互通

最近更新时间：2023-07-13 17:59:11

操作场景

目前容器集群与用户 IDC 互通主要通过两种方式：**专线**和 **IPsec VPN**。

> **注意**
> - 本文档以已创建集群并已添加节点为例。关于如何创建集群，您可以参考 创建集群 进行创建。
> - 请先确保容器服务所在的 VPC 和您 IDC 机房已通过专线或 VPN 成功连接。若通道未连接，您可以参考 VPN 通道未连通如何处理 。

操作步骤

通过专线方式互通

1. 参考 申请物理专线，申请物理专线。
2. 参考 申请通道，申请通道。
3. 参考 创建专线网关，创建云联网类型专线网关。
4. 验证容器节点与 IDC 互通。

> **注意**
> 执行此步骤时，请保证容器节点与 IDC 互通，验证通过。

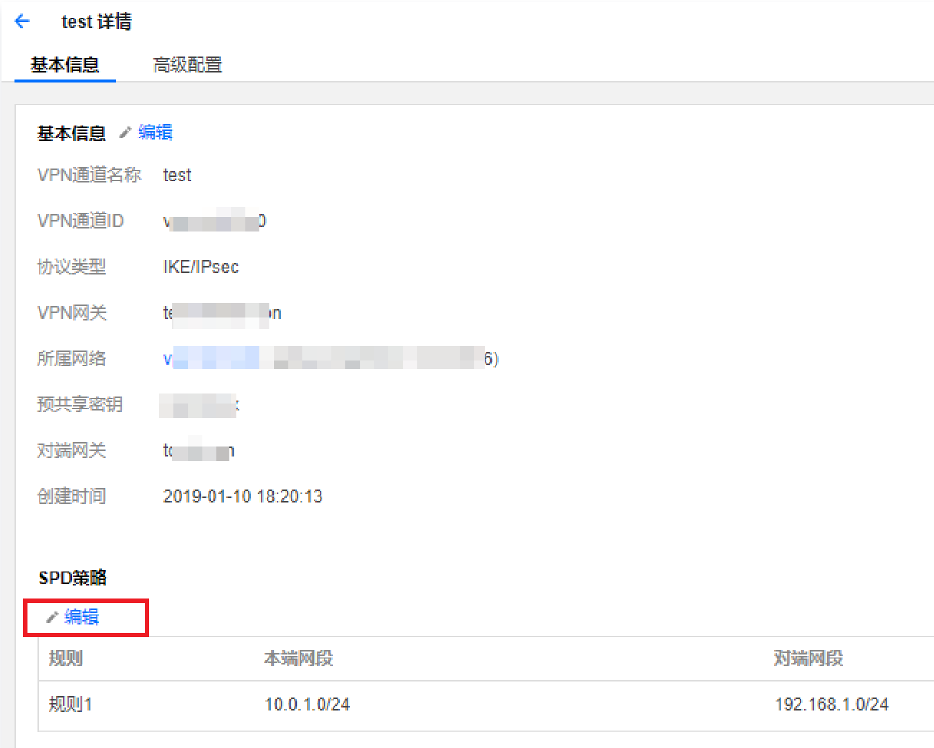
5. 准备地域，appID，集群 ID，vpcID，专线网关 ID 信息，提交工单 打通容器网络。
6. 根据 IDC 使用的协议类型，选择操作方式。
   - 若 IDC 使用的是 BGP 协议，容器网段路由将自动同步。
   - 若是其他协议，需在 IDC 内配置访问容器网段下一跳路由到专线网关。
7. 验证容器与 IDC 互通。

通过 VPN 方式互通

配置 SPD 策略

1. 登录 私有网络控制台 。
2. 在左侧导航栏中，单击 **VPN链接** > VPN通道，进入VPN 通道管理页面。

3. 在 VPN 通道的详情页面，单击 “SPD策略” 栏下的**编辑**，添加容器网段。如下图所示：



4. 单击**保存**。
5. 重复执行步骤3 – 步骤4，配置对端 VPN 通道的 SPD 策略。

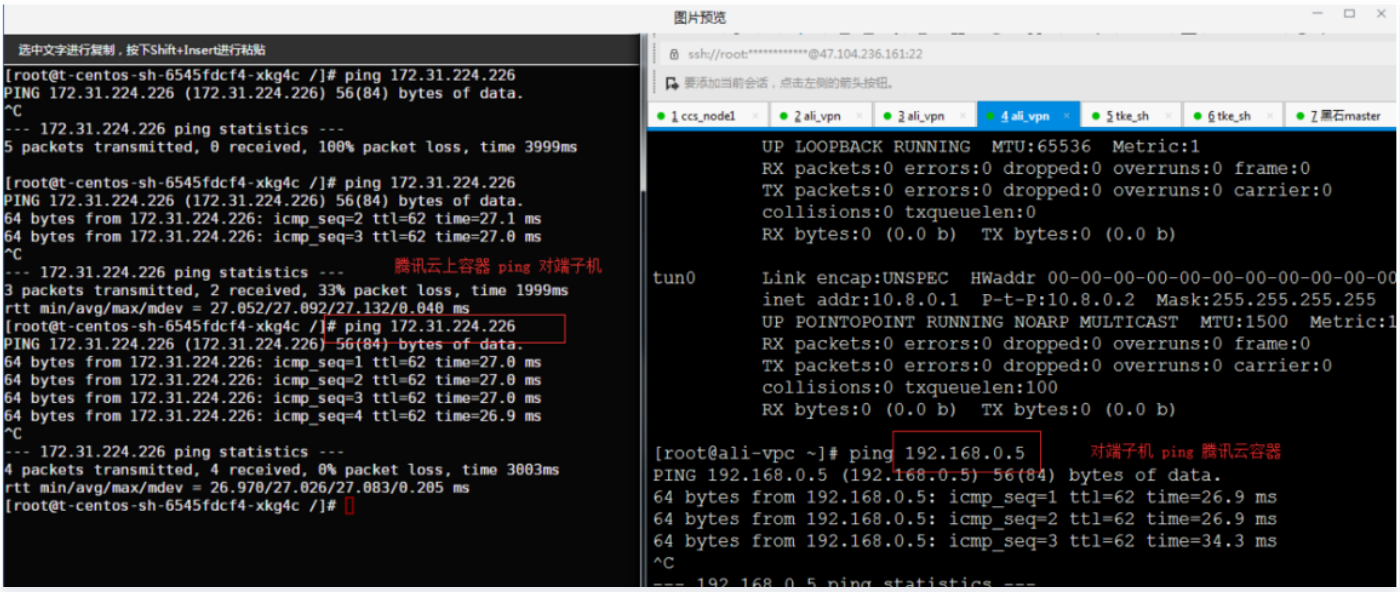
添加容器网段

> **注意**
> 一个子网只能绑定一个路由表，若关联多个路由表，将被替换成最后一个绑定的路由表。

1. 在左侧导航栏中，单击 路由表，进入路由表管理页面。
2. 找到 设置同地域集群间互通 或者 设置跨地域集群间互通 时配置的路由表，单击该路由表的 ID/名称，进入路由表的详情页面。
3. 单击**+新增路由策略**，追加容器网段。
4. 选择**关联子网**页签，单击**新建关联子网**，关联子机所在的子网。
5. 重复执行步骤2 – 步骤4，在您对端的路由设备上，添加腾讯云容器所在网段。

预期结果

容器和对端子机可以互通。如下图所示：



容器间与 VPN 对端子机已经实现互通。

> **说明**
> 如需云上容器与 IDC 机房通过 IPsec VPN 互通，主要需要设置 **SPD 策略**和**路由表**。

注册 GlobalRouter 模式集群到云联网

最近更新时间：2025-06-04 18:05:02

云联网说明

云联网（Cloud Connect Network，CCN）为您提供云上私有网络间（Virtual Private Cloud，VPC）、VPC 与本地数据中心间互联的服务。您可以将 VPC 和专线网关实例加入云联网，以实现单点接入、全网资源互通，轻松构建简单、智能、安全、灵活的混合云及全球互联网络。

操作场景

您可以将已有容器集群注册到云联网，云联网可将容器网络归纳至管理范围中。当容器网络完成注册后，您可以在云联网侧启用或关闭容器网络的网段路由，实现容器集群与云联网内的资源互通。

> **注意：**
> 当容器集群注册到云联网后，该网段与云联网实例中已有路由不冲突时开启，冲突时默认关闭。

前提条件

- 集群所在 VPC 已在云联网中，云联网相关操作请参见 云联网操作总览。
- 评估集群容器网络的网段是否与云联网网内其他资源网段冲突。

操作步骤

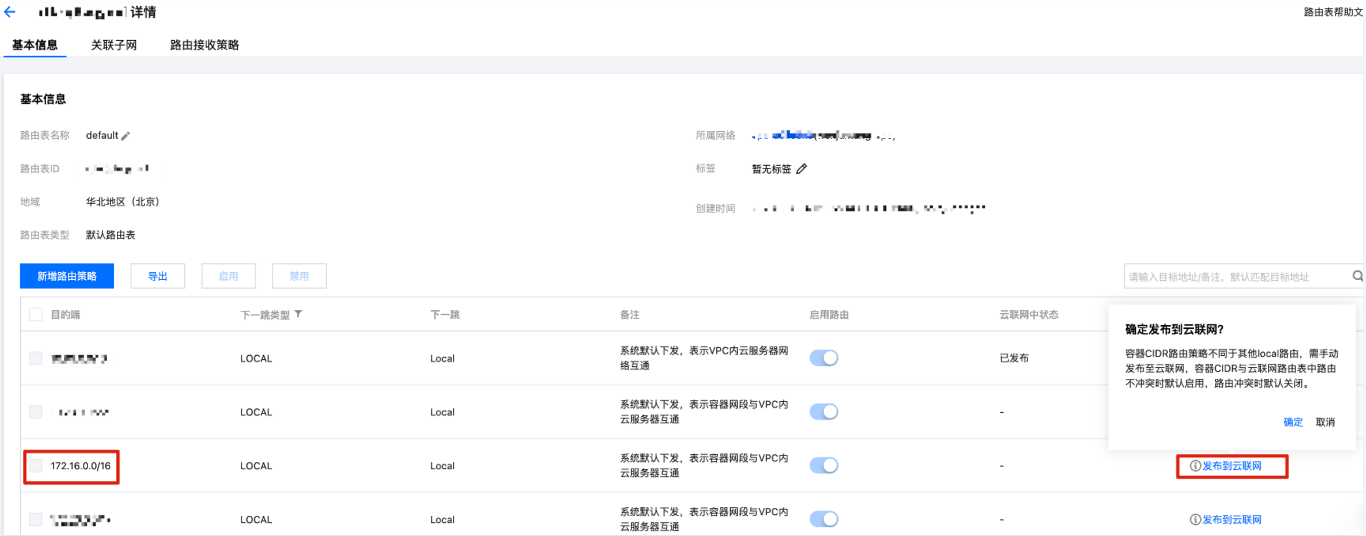
注册容器网络至对应云联网

1. 登录容器服务控制台，单击左侧导航栏中的 集群。
2. 进入集群管理页面，选择需要进行云联网注册的集群 ID。
3. 单击左侧的**基本信息**进入集群基础信息页面，单击**通过路由表发布到云联网**，跳转到 VPC 的路由表配置中，将 GR 集群的 CIDR 发布到云联网。

   > **注意：**
   > 此步骤仅是将容器网段注册到云联网，路由是否生效，需要在云联网侧控制。

4. 进入 VPC 的路由表配置页面，选择容器所在的 CIDR 网段，单击**发布到云联网**，单击**确定**。



查看容器网段路由

1. 登录私有网络控制台，单击左侧导航栏中的 云联网 进入云联网管理页面。
2. 单击集群 VPC 关联的云联网所在行右侧的**管理实例**，进入云联网实例管理页面。如下图所示：

3. 在云联网实例管理页面中，单击**路由表**页签，查看容器网段路由启动情况。如下图所示：

> **说明：**
> - 若网段不冲突，则路由默认启动。网段冲突，则路由默认关闭。
> - 路由冲突原则请参见 路由限制，如需启动冲突路由，请参见 启用路由 。

4. 可开始测试容器集群与云联网其他资源的互通性。

VPC-CNI 模式
VPC-CNI 模式介绍

最近更新时间：2023-05-17 15:40:39

使用原理

VPC-CNI 模式是容器服务 TKE 基于 CNI 和 VPC 弹性网卡实现的容器网络能力，适用于对时延有较高要求的场景。该网络模式下，容器与节点分布在同一网络平面，容器 IP 为 IPAMD 组件所分配的弹性网卡 IP。

其中 VPC-CNI 模式分为共享网卡模式和独占网卡模式，两种网络模式适用于不同的场景。您可以根据业务需要选择不同的网络模式。

- 共享网卡模式：Pod 共享一张弹性网卡，IPAMD 组件为弹性网卡申请多个 IP 给到不同的 Pod。可固定 Pod IP，详情请参见 固定 IP 模式使用说明 。
- 独占网卡模式：每个 Pod 有独立的弹性网卡，性能更高。受机型影响，不同节点可使用的弹性网卡数量有限，单节点 Pod 密度更低。

使用限制

- 当前 VPC-CNI 模式的子网不建议与其他云上资源共用（如云服务器、负载均衡等）。
- 集群内的节点需要和子网处于相同可用区，如果节点可用区与容器子网不在相同可用区，Pod 将无法调度。
- 节点上可调度的 VPC-CNI 模式的 Pod 数量受限于节点所支持插入弹性网卡能绑定 IP 的最大数量和网卡数量。配置越高的机器可插入的弹性网卡数量越多，可以通过查看节点的 Allocatable 来确认。

应用场景

相比 Global Router，VPC-CNI 具有以下优势及适用场景：

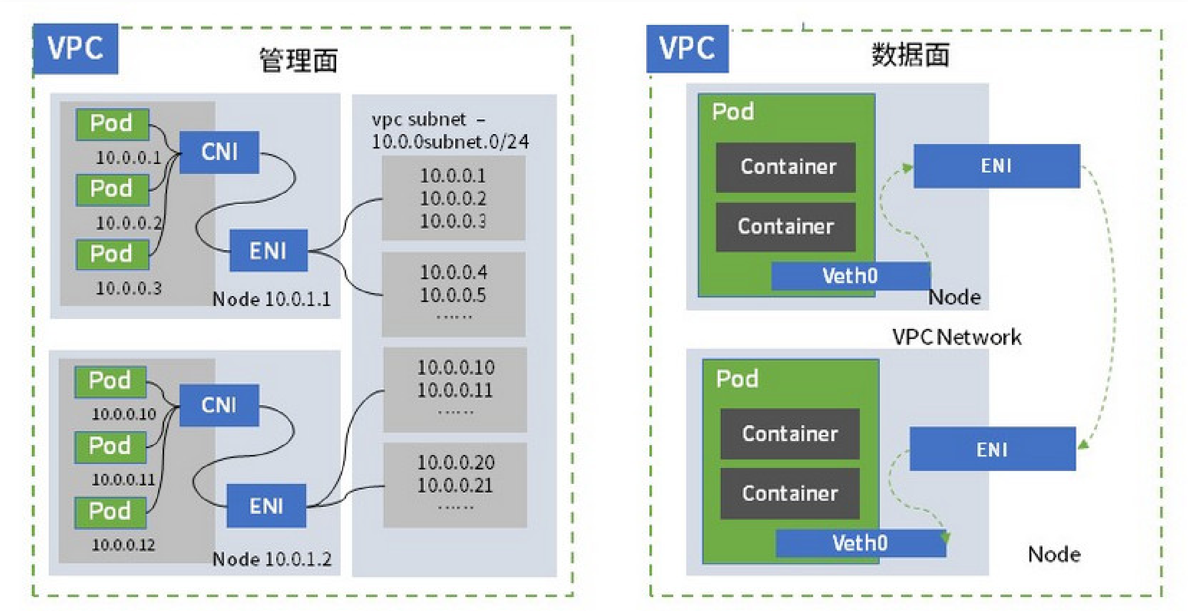
- 少了一层网桥，网络转发性能更高，大约提升10%，适用于对网络时延要求较高的场景。
- 支持 Pod 固定 IP，适用于依赖容器固定 IP 的场景。例如，传统架构迁移到容器平台及针对 IP 做安全策略限制。
- 支持 LB 直通 Pod。

多 Pod 共享网卡模式

最近更新时间：2025-07-01 14:42:22

使用原理

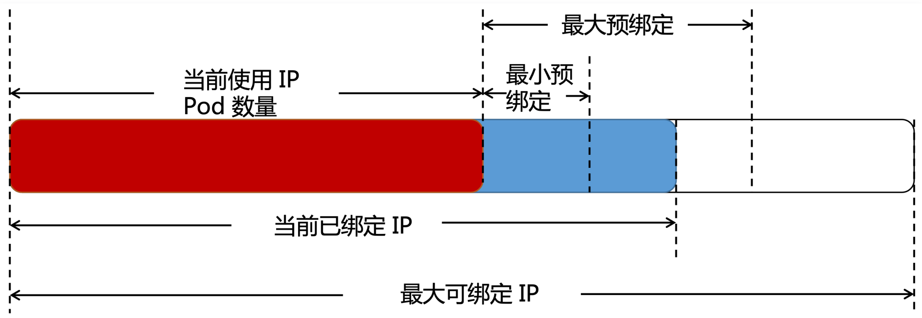
VPC-CNI 多 Pod 共享网卡模式使用原理图如下所示：



- 集群网络是用户的 VPC，节点和容器子网属于该 VPC。
- 容器子网可以选择多个 VPC 内的子网。
- 可设置是否开启固定 IP。您可参考 固定 IP 模式使用说明 。

IP 地址管理原理

非固定 IP 模式



- TKE 组件在每个节点维护一个可弹性伸缩的 IP 池。已绑定的 IP 数量将被维持在 **Pod 数量 + 最小预绑定数量**及 **Pod 数量 + 最大预绑定数量**之间：
  - 当**已绑定数量 < Pod 数量 + 最小预绑定数量**时，会绑定 IP 使得**已绑定数量 = Pod 数量 + 最小预绑定数量**。
  - 当**已绑定数量 > Pod 数量 + 最大预绑定数量**时，会定时释放IP（约2分钟一次），直到**已绑定数量 = Pod 数量 + 最大预绑定数量**。
  - 当**最大可绑定数量 < 当前已绑定数量**时，会直接释放多余的空闲 IP，使得**已绑定数量 = 最大可绑定数量**。
- 共享网卡的 Pod 创建时，从节点可用 IP 池中随机分配一个可用 IP。
- 共享网卡的 Pod 销毁时，IP 释放回节点的 IP 池，留给下一个 Pod 使用，不会在 VPC 侧释放（删除）。
- IP 和网卡的分配和释放目前基于**最少网卡原则**，即保证使用的弹性网卡尽量的少：
  - **IP 分配给 Pod**：优先分配已分配 IP 数量最多的网卡上的 IP
  - **IP 释放**：优先释放已分配 IP 数量最少的网卡上的 IP
  - **新网卡绑定**：若当前已绑定网卡 IP 配额用尽或网卡所在的子网 IP 用完，则申请新网卡绑定 IP。
  - **网卡释放**：若已绑定网卡的辅助 IP 都已解绑，且不再需要新增 IP，则解绑并删除网卡。

- 节点会注册扩展资源 `tke.cloud.tencent.com/eni-ip`，资源的可分配数(Allocatable) 为实际的已绑定 IP 资源数，总量(Capacity) 为节点可绑定的 IP 资源上限。因此，当 Pod 调度到某节点失败时，说明节点的 IP 已用尽。
- 新网卡的子网选择：新网卡优先选择可用 IP 最多的子网。
- 各节点最大可绑定 IP = 最大绑定网卡数 * 单网卡可绑定 IP 数。
- 当前**最小预绑定数量**和**最大预绑定数量**的默认值为5。

固定 IP 模式

- TKE 网络组件维护一个集群维度的可用 IP 池。
- 集群每新增一个节点，不会提前绑定任何辅助 IP 和弹性网卡，IP 完全**按需分配**。
- 新建一个使用 VPC-CNI 模式的 Pod 时，IPAMD 组件会在其所在节点寻找一张可用网卡分配 IP，网卡分配原则基于**最少网卡**，即优先分配已绑定 IP 数量最多的网卡。
- 若已绑定网卡都已绑满 IP，则尝试新建绑定网卡再分配 IP，网卡所在子网优先选择可用 IP 最多的子网。
- 没有设置固定 IP 注解的 Pod 销毁时，IP 地址回归集群的可用 IP 池，并触发网卡解绑 IP，IP 地址将释放回 VPC 子网内。
- 固定 IP 的 Pod 的 IP 销毁后会在 VPC 内保留，保证下一次创建同名 Pod 的时候仍使用这个 IP。
- 节点删除时，将释放网卡占用的 IP 资源。
- 多容器子网的情况下，网卡优先分配到可用 IP 数量最多的子网内，若没有完全满足需求的子网，则节点绑定网卡失败。

多网卡数据面原理

当节点绑定了多张网卡时，Pod 发出的网络包遵循策略路由转发到对应的网卡上：

- 在节点上执行 `ip link` 可看到节点所有的网络设备信息，通过弹性网卡的 MAC 地址比对，可知道其中弹性网卡对应的网络设备。一般情况下，`eth0` 为主网卡，`eth1`、`eth2` 等为辅助弹性网卡：

```
[root@VM-4-196-tlinux ~]# ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:23:98:4a brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 20:90:6f:cf:6a:b9 brd ff:ff:ff:ff:ff:ff
4: eni4b1d9e7dd9b@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether da:53:24:6c:af:e4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
5: eni43625ff3fd0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 46:17:d2:59:ff:d9 brd ff:ff:ff:ff:ff:ff link-netnsid 1
6: eni206b44a5853@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 0e:0d:c6:3b:70:99 brd ff:ff:ff:ff:ff:ff link-netnsid 2
7: enia8afb31cbdb@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 12:1d:32:9d:8f:9f brd ff:ff:ff:ff:ff:ff link-netnsid 3
8: eni2435af570f1@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 42:a2:64:9a:3f:01 brd ff:ff:ff:ff:ff:ff link-netnsid 4
9: eni4ab50a1bd0b@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 5a:3e:c6:3e:5c:51 brd ff:ff:ff:ff:ff:ff link-netnsid 5
10: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 20:90:6f:97:cf:45 brd ff:ff:ff:ff:ff:ff
11: eni0a1cc6fca61@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 6a:4d:57:25:45:3c brd ff:ff:ff:ff:ff:ff link-netnsid 6
```

- 在节点上执行 `ip rule` 可看到策略路由表的信息，TKE 网络组件通过弹性网卡的 `<link index>+2000` 得到路由表号，绑定了对应网卡 IP 的 Pod 网络包都将转发到该路由表，如此例中，eth1 对应的路由表即为 2003，eth2 对应的路由表即为 2010：

```
[root@VM-4-196-tlinux ~]# ip rule
0:      from all lookup local
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
512:    from all to [redacted] lookup main
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2003
1536:   from [redacted] lookup 2010
1536:   from [redacted] lookup 2010
1536:   from [redacted] lookup 2010
32766:  from all lookup main
32767:  from all lookup default
```

- 对应的路由表则设置了到对应网卡的默认路由，节点上执行 `ip route show table <id>` 可查看：

```
[root@VM-4-196-tlinux ~]# ip route show table 2003
default via [redacted] dev eth1 onlink
[root@VM-4-196-tlinux ~]# ip route show table 2010
default via [redacted] dev eth2 onlink
```

而欲发送给 Pod 的网络包到达节点时，同样遵循策略路由，直接通过主路由表发送给 Pod 的 Veth 网卡。

使用方法

使用 VPC-CNI 需要确保 `rp_filter` 处于关闭状态。可参考以下代码示例：

```
sysctl -w net.ipv4.conf.all.rp_filter=0
# 假设 eth0 为主网卡
sysctl -w net.ipv4.conf.eth0.rp_filter=0
```

> **注意：**
> `tke-eni-agent` 组件自动设置节点的内核参数。若您自己有维护内核参数且打开 `rp_filter`，则会导致网络不通。

开启 VPC-CNI

创建集群时开启 VPC-CNI

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，单击集群列表上方的**新建**。
3. 在**创建集群 > 网络配置**页面，在容器网络插件中选择 **VPC-CNI**。

> **说明：**
> 默认情况下，VPC-CNI 模式**不支持固定 Pod IP 能力**，且该能力仅支持在 创建集群 时设置。如需为集群开启支持固定 Pod IP，请参见 固定 IP 模式使用说明 。

为已有集群开启 VPC-CNI

创建集群时选择 Global Router 网络插件，后续在集群基本信息页面开启 VPC-CNI 模式（两种默认混用）。
1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。

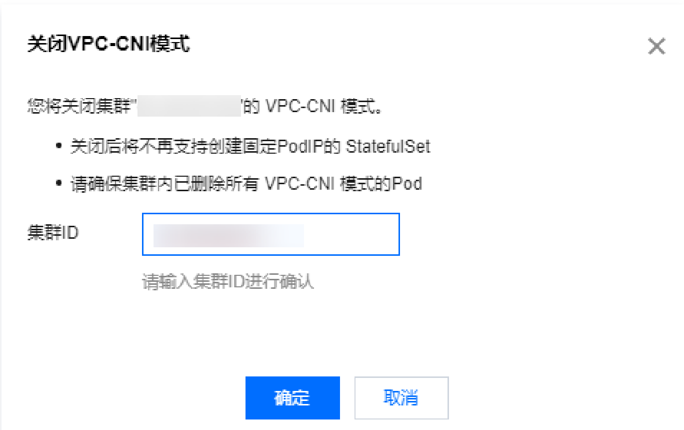
3. 在集群的**基础信息**页面，查看节点和网络信息，在 VPC-CNI 模式中单击**开启**。
4. 在弹出窗口中选择是否支持固定 IP，并选择子网。如下图所示：

⚠ **注意：**
- 针对固定 IP 场景，启用 VPC-CNI 后需要设置 IP 回收策略，即设置 Pod 销毁后需要退还 IP 的时长。
- 非固定 IP 的 Pod 销毁后可立即释放 IP（非释放回 VPC，释放回容器管理的 IP 池），不受此设置的影响。

5. 单击**提交**，即可为已有集群开启 VPC-CNI。

关闭 VPC-CNI

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。
3. 在集群的**基本信息**页面，查看节点和网络信息，在 VPC-CNI 模式中单击关闭。
4. 在弹出窗口中输入集群 ID 进行确认，单击**确定**，即可关闭 VPC-CNI。



Pod 间独占网卡模式

最近更新时间：2025-04-24 15:49:12

Pod 间独占网卡模式在原有 VPC-CNI 模式单网卡多 IP 模式的基础上，进阶为容器直接独享使用弹性网卡。无缝对接腾讯云私有网络产品的全部功能，同时在性能上做了极大的提升。

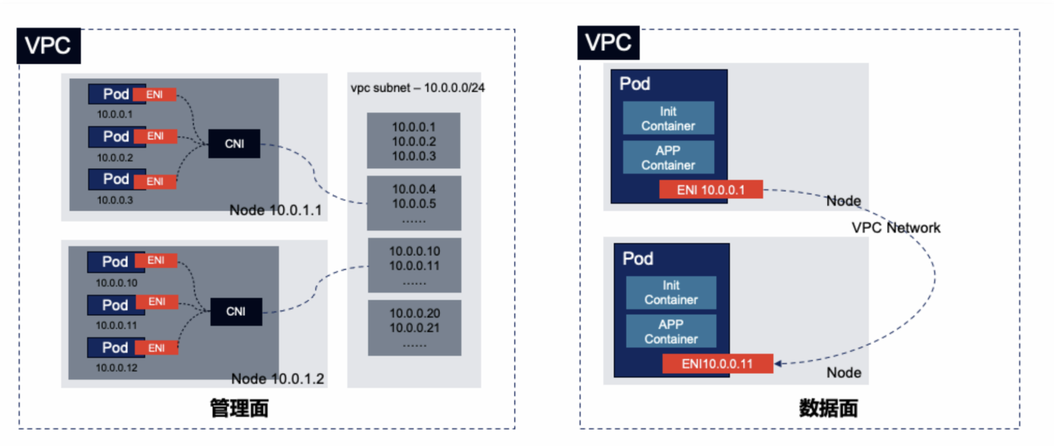
> **注意：**
> 目前该功能正处于内测阶段，您可以 提交工单 申请开通使用。

功能简介

新一代 VPC-CNI 模式的网络方案中，能够在原有的网络能力中额外增加以下能力：
- 支持 Pod 绑定 EIP/NAT，不再依赖节点的外网访问能力，无须做 SNAT，可以满足爬虫，视频会议等高并发，高带宽外网访问场景。
- 支持基于 Pod 名称的固定 IP，Pod 调度重启后仍能保证 IP 不变。
- 支持 CLB 直通 Pod，不再经过 NodePort 转发，提升转发性能并拥有统一的负载均衡视图。

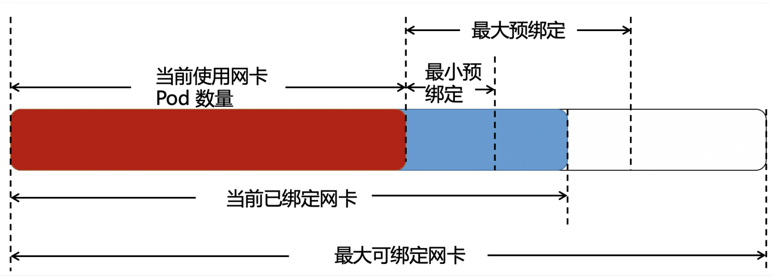
实现方式

新一代方案在原有 VPC-CNI 模式的基础上扩展，依托于弹性网卡，将绑定到节点的弹性网卡通过 CNI 配置到容器网络命名空间，实现容器直接独享使用弹性网卡。实现原理如下图所示：



IP 地址管理原理

非固定 IP 模式



- TKE 组件在每个节点维护一个可弹性伸缩的网卡池。已绑定的网卡数量将被维持在 **Pod 数量 + 最小预绑定数量**及 **Pod 数量 + 最大预绑定数量**之间：
  - 当已绑定数量 < Pod 数量 + 最小预绑定数量时，会绑定网卡使得已绑定数量 = Pod 数量 + 最小预绑定数量。
  - 当已绑定数量 > Pod 数量 + 最大预绑定数量时，会定时释放1个网卡（约2分钟一次），直到已绑定数量 = Pod 数量 + 最大预绑定数量。
  - 当最大可绑定网卡数量 < 当前已绑定数量时，会直接释放多余的空闲网卡，使得已绑定数量 = 最大可绑定数量。
- 独占网卡的 Pod 创建时，从节点可用网卡池中随机分配一个可用网卡。
- 独占网卡的 Pod 销毁时，网卡释放回节点的网卡池，留给下一个 Pod 使用，不会在 VPC 侧释放（删除）。
- 节点删除时，将释放（删除）所有已绑定的网卡。
- 多容器子网的情况下，网卡优先分配到可用 IP 数量最多的子网内。

固定 IP 模式

- TKE 不会为每个节点维护网卡池，网卡不会预绑定到节点上。
- 独占网卡的 Pod 创建时，直接绑定一张网卡到节点上，给这个 Pod 使用。
- 非固定 IP 的独占网卡 Pod 销毁时，直接在 VPC 侧删除释放该 Pod 使用的网卡，固定 IP 的 Pod 销毁时，网卡仅做解绑，不会删除释放。
- 节点删除时，将释放（删除）所有已绑定的网卡。
- 多容器子网的情况下，网卡优先分配到可用 IP 数量最多的子网内。

功能限制

- 仅支持部分机型使用该网络模式。具体列表如下：
  - S5
  - SA2
  - IT5
  - ITA4
  - SA3
  - M5
  - SA5
  - S8
  - ITA5
  - BMS5.52XLARGE768
  - BMIA2.48XLARGE512
  - BMSA2.48XLARGE512
  - BMM5c.52XLARGE3072
  - BMDA2.48XLARGE512
- 节点上运行的独立网卡方案的 Pod 数量限制受到机型可绑定弹性网卡数量的影响。其最大数量为最大可绑定弹性网卡数量 - 1，详情请参见 VPC-CNI 模式 Pod 数量限制。
- 仅支持新集群，存量容器服务 TKE 集群暂不支持变更网络方案。
- 有 VPC-CNI 模式的统一限制：
  - 需要为容器专门规划子网，子网不建议其他云上资源共用（如云服务器、负载均衡等）。
  - 集群内的节点需要和子网处于相同可用区，如果节点可用区与容器子网不在相同可用区，Pod 将无法调度。

固定 IP 模式使用说明

固定 IP 使用方法

最近更新时间：2025-03-21 10:13:22

使用场景

适用于依赖容器固定 IP 的场景。例如，传统架构迁移到容器平台及针对 IP 做安全策略限制。对 IP 无限制的业务不推荐您使用固定 IP 模式。

能力和限制

- 支持 Pod 销毁 IP 保留，Pod 迁移 IP 不变，从而实现固定 IP。
- 支持多子网，但不支持跨子网调度固定 IP 的 Pod，因此固定 IP 的 Pod 不支持跨可用区调度。
- 固定 IP 的 Pod 销毁后，其 IP 只在本集群范围内保留。
- 固定 IP 的 Pod 支持在普通节点、原生节点和超级节点间任意迁移，且能保持 IP 不变。
- VPC-CNI 网络模式集群创建之后，暂不支持修改开启/关闭固定 IP 模式。
- GlobalRouter 模式集群需先关闭当前 VPC-CNI 模式，再打开实现固定 IP 模式的变更。

使用方法

您可以通过以下步骤使用固定 IP：

1. 为集群开启固定 IP 模式。
2. 创建固定 Pod IP 类型工作负载。

为集群开启固定 IP 模式

您可以通过以下两种方式为集群开启固定 IP 模式：

- 创建集群选择固定 IP 模式的 VPC-CNI。
- 为 GlobalRouter 模式附加固定 IP VPC-CNI 模式。

创建集群选择固定 IP 模式的 VPC-CNI

> **说明：**
> 使用此方式启用 VPC-CNI，通过控制台或通过 yaml 创建工作负载，Pod 均默认使用弹性网卡。

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，单击集群列表上方的**新建**。
3. 在**创建集群**页面，选择容器网络插件为 **VPC-CNI**，并勾选“开启支持”固定 Pod IP 即可。如下图所示：

为 GlobalRouter 模式附加固定 IP VPC-CNI 模式

> **说明：**
> - 为 GlobalRouter 模式附加固定 IP VPC-CNI 模式即创建集群时选择 Global Router 网络插件，后续在集群基本信息页面开启 VPC-CNI 模式（两种模式默认混用）。
> - 使用此方式启用 VPC-CNI，Pod 默认不使用弹性网卡。

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，选择需开启 VPC-CNI 的集群 ID，进入集群详情页。

3. 在**集群详情**页面，选择左侧**基本信息**。在节点和网络信息模块中，找到 VPC-CNI 字段，单击**开启**。
4. 在弹出窗口中勾选“开启支持”固定 Pod IP，确认 IP 回收策略，并选择子网。如下图所示：

> **注意：**
> - 针对固定 IP 场景，启用 VPC-CNI 后需要设置 IP 回收策略，即设置 Pod 销毁后需要退还 IP 的时长。
> - 非固定 IP 的 Pod 销毁后可立即释放 IP（非释放回 VPC，释放回容器管理的 IP 池），不受此设置的影响。

5. 单击**提交**，即可完成为已有集群开启 VPC-CNI。

创建固定 Pod IP 类型工作负载

集群开启固定 IP 模式后，如果您存在业务需要在容器服务 TKE 中部署，并存在固定 Pod IP 的需求，您需要使用 Pod 重建名称不变的工作负载，常见的为 Kubernetes 原生提供的 StatefulSets。TKE 提供扩展 StatefulSet 固定 IP 的能力，该类型的 StatefulSet 创建的 Pod 将通过弹性网卡分配真实的 VPC 内的 IP 地址。容器服务 TKE VPC-CNI 的插件负责 IP 分配，当 Pod 重启或迁移，可实现 IP 地址不变。
您可以通过创建固定 IP 类型 StatefulSet 来满足以下场景：
- 通过来源 IP 授权。
- 基于 IP 做流程审核。
- 基于 Pod IP 做日志查询等。

> **注意：**
> 固定 IP 类型 StatefulSet 存在使用限制，仅支持 StatefulSet 生命周期内固定 IP。

您可通过以下两种方法创建固定 IP：

通过控制台

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 选择需要使用固定 IP 模式的集群 ID 名称，进入该集群的管理页面。
3. 选择**工作负载** > **StatefulSet**，进入 **StatefulSet** 的集群管理页面。

4. 单击**新建**。在新建 StatefulSet 中勾选**网络模式 > 使用 VPC-CNI 模式**，并开启**固定 Pod IP**。如下图所示：

- IP 地址范围：目前仅支持随机。
- 固定 Pod IP：选择**开启**。

通过 Yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    k8s-app: busybox
  name: busybox
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      k8s-app: busybox
      qcloud-app: busybox
  serviceName: ""
  template:
    metadata:
      annotations:
        tke.cloud.tencent.com/networks: "tke-route-eni"
        tke.cloud.tencent.com/vpc-ip-claim-delete-policy: Never
      creationTimestamp: null
      labels:
        k8s-app: busybox
        qcloud-app: busybox
    spec:
      containers:
      - args:
        - "10000000000"
        command:
        - sleep
        image: busybox
        imagePullPolicy: Always
        name: busybox
        resources:
          limits:
            tke.cloud.tencent.com/eni-ip: "1"
          requests:
            tke.cloud.tencent.com/eni-ip: "1"
```

- spec.template.annotations：`tke.cloud.tencent.com/networks: "tke-route-eni"` 表明 Pod 使用共享网卡的 VPC-CNI 模式，如果使用的是独立网卡的 VPC-CNI 模式，请将值修改成 `"tke-direct-eni"` 。**该注解仅在 GlobalRouter 附加固定 IP VPC-CNI 模式的集群中需要添加，创建集群时就选择 VPC-CNI 模式的集群中无需添加**。
- spec.template.annotations：创建 VPC-CNI 模式的 Pod，您需要设置 annotations，即 `tke.cloud.tencent.com/vpc-ip-claim-delete-policy` ，默认是 “Immediate” ，Pod 销毁后，关联的 IP 同时被销毁。如需固定 IP，则

需设置成 “Never”，Pod 销毁后 IP 也将会保留，那么下一次同名的 Pod 拉起后，会使用之前的 IP。

- spec.template.spec.containers.0.resources：创建共享网卡的 VPC-CNI 模式的 Pod，需要添加 requests 和 limits 限制，即 `tke.cloud.tencent.com/eni-ip` 。如果是独立网卡的 VPC-CNI 模式，则添加 `tke.cloud.tencent.com/direct-eni` 。该资源只需给 Pod 的第一个容器添加即可，其他容器无需添加。此外，该资源默认情况下会自动添加，您只需声明 `tke.cloud.tencent.com/networks` 注解即可。

固定 IP 相关特性

最近更新时间：2023-12-07 17:11:54

固定 IP 的保留和回收

固定 IP 模式下，创建使用 VPC-CNI 模式的 Pod 以后，网络组件会为该 Pod 在同 namespace 下创建同名的 CRD 对象 `VpcIPClaim` 。该对象描述 Pod 对 IP 的需求。网络组件随后会根据这个对象创建 CRD 对象 `VpcIP` ，并关联对应的 `VpcIPClaim` 。 `VpcIP` 以实际的 IP 地址为名，表示实际的 IP 地址占用。
您可以通过以下命令查看集群使用的容器子网内 IP 的使用情况：

```
kubectl get vip
```

对于非固定 IP 的 Pod，其 Pod 销毁后 `VpcIPClaim` 也会被销毁， `VpcIP` 随之销毁回收。而对于固定 IP 的 Pod，其 Pod 销毁后 `VpcIPClaim` 仍然保留， `VpcIP` 也因此保留。同名的 Pod 启动后会使用同名的 `VpcIPClaim` 关联的 `VpcIP` ，从而实现 IP 地址保留。
由于网络组件在集群范围内分配 IP 时会依据 `VpcIP` 信息找寻可用 IP，因此固定 IP 的地址若不使用需要及时回收（目前默认策略是永不回收），否则会导致 IP 浪费而无 IP 可用。本文介绍过期回收、手动回收及级联回收的 IP 回收方法。

过期回收

新建集群

在 创建集群 页面，容器网络插件选择**VPC-CNI**模式并且勾选**开启支持**固定Pod IP 支持，如下图所示：

在高级设置中设置 IP 回收策略，可以设置 Pod 销毁后多少秒回收保留的固定 IP。如下图所示：

存量集群

对于**存量集群**，也可支持变更：

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧**组件管理**，在组件管理页面中，找到 eniipamd 组件，选择**更新配置**。

4. 在更新配置页面，填写固定 IP 回收策略里的过期时间，并单击**完成**。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

- 修改现存的 tke-eni-ipamd deployment：`kubectl edit deploy tke-eni-ipamd -n kube-system`。
- 执行以下命令，在 `spec.template.spec.containers[0].args` 中加入/修改启动参数。

```
- --claim-expired-duration=1h # 可填写不小于 5m 的任意值
```

手动回收

对于急需回收的 IP 地址，需要先确定需回收的 IP 被哪个 Pod 占用，找到对应的 Pod 的名称空间和名称，执行以下命令通过手动回收：

> **注意**
> 需保证回收的 IP 对应的 Pod 已经销毁，否则会导致该 Pod 网络不可用。

```
kubectl delete vipc <podname> -n <namespace>
```

级联回收

目前的固定 IP 与 Pod 强绑定，而与具体的 Workload 无关（例如 deployment、statefulset 等）。Pod 销毁后，固定 IP 不确定何时回收。TKE 现已实现删除 Pod 所属的 Workload 后即刻删除固定 IP。
以下步骤介绍如何开启级联回收：

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。

3. 选择左侧**组件管理**，在组件管理页面中，找到 **eniipamd** 组件，选择**更新配置**。

4. 在**更新组件配置**页面，勾选**级联回收**，并单击**完成**。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

1. 修改现存的 tke-eni-ipamd deployment：`kubectl edit deploy tke-eni-ipamd -n kube-system`。
2. 执行以下命令，在 `spec.template.spec.containers[0].args` 中加入启动参数：

```
- --enable-ownerref
```

修改后，ipamd 会自动重启并生效。生效后，增量 Workload 可实现级联删除固定 IP，存量 Workload 暂不能支持。

相关问题

节点不能分配到弹性网卡，无法正常调度 Pod（共享网卡模式）

当节点加入到集群后，ipamd 会尝试从和节点相同可用区的子网（配置给 ipamd 的子网）中为节点绑定一个弹性网卡，如果 ipamd 异常或者没有给 ipamd 配置和节点相同可用区的子网，ipamd 将无法给节点分配辅助网卡。此外，如果当前 VPC 使用的辅助网卡数目超过上限，则无法给节点分配辅助网卡。
执行以下命令，确认问题原因：

```
kubectl get event
```

- event 中显示 ENILimit，则是配额问题，可以通过为 VPC 调大弹性网卡数目配额来解决问题。
- 查看集群内节点所在可用区的容器子网 IP 是否充足，如已耗尽则补充同可用区容器子网可解决。

节点不能分配到弹性网卡，提示弹性网卡数量超出限制

现象

节点配置的弹性网卡无法绑定，nec 关联的 vip attach 失败。查看 nec 则看到节点关联的 nec status 为空。
执行以下代码可查看 nec：

```
kubectl get nec -o yaml
```

当节点关联的 nec status 为空时返回结果如下图所示：

```
- apiVersion: networking.tke.cloud.tencent.com/v1
  kind: NodeENIConfig
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: 
        {"apiVersion":"networking.tke.cloud.tencent.com/
esourceVersion":"20649","selfLink":"/apis/networking.tke
2"},"status":{}}
    creationTimestamp: "2020-06-22T13:11:34Z"
    finalizers:
    - tke.cloud.tencent.com/nec
    generation: 2
    name: 9.131.155.254
    resourceVersion: "25339"
    selfLink: /apis/networking.tke.cloud.tencent.com/v1/
    uid: [redacted]
  spec:
    maxENI: 7
    maxIPPerENI: 13
    providerID: [redacted]
    zone: ap-hongkong-2
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

执行以下代码查看 nec 关联的 VIP：

```
kubectl get vip -oyaml
```

若命令返回成功则报错 VIP 状态为 Attaching，报错信息如下图所示：

```
  kind: VpcIP
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"networking.tke.cloud.tencent.com/v1","kind":"VpcIP","metadata":{"annotatio
m/created-by-ipamd":"yes"},"name":"9.208.15.9","resourceVersion":"23949","selfLink":"/apis/netwo
.cloud.tencent.com/v1","kind":"NodeENIConfig","name":"9.131.155.177","resourceVersion":"20645","
TransitionTime":"2020-06-22T13:11:34Z","message":"create eni: failed to create eni: [TencentClou
d","status":"False","type":"VpcIPAttached"}],"phase":"Attaching"}}
      tke.cloud.tencent.com/max-secondary-ip: "13"
    creationTimestamp: "2020-06-22T13:11:34Z"
    generation: 412
    labels:
      tke.cloud.tencent.com/created-by-ipamd: "yes"
    name: 9.208.15.9
    resourceVersion: "250800"
    selfLink: /apis/networking.tke.cloud.tencent.com/v1/vpcips/9.208.15.9
    uid: [redacted]
  spec:
    necRef:
      apiVersion: networking.tke.cloud.tencent.com/v1
      kind: NodeENIConfig
      name: 9.131.155.177
      resourceVersion: "20645"
      uid: [redacted]
    type: Node
  status:
    conditions:
    - attempts: 410
      lastProbeTime: "2020-06-23T02:42:41Z"
      lastTransitionTime: "2020-06-22T13:11:34Z"
      message: 'create eni: failed to create eni: [TencentCloudSDKError] Code=LimitExceeded,
        Message=`NetworkInterface`数量达到上限。, RequestId=[redacted]'
      reason: AttachFailed
      status: "False"
      type: VpcIPAttached
    phase: Attaching
kind: List
metadata:
  resourceVersion: ""
```

解决方案

目前腾讯云弹性网卡限制一个 VPC 下面最多绑定1000个弹性网卡。您可 在线咨询 申请提高配额，配额按地域生效。

非固定 IP 模式使用说明

最近更新时间：2023-11-01 16:44:01

使用场景

适用于不依赖容器固定 IP 的场景。例如，可部署多副本的无状态服务，无状态离线业务等。

能力和限制

- 支持节点维护可用的网卡/ IP 池，从而支持 Pod 大规模快速重建。
- 支持预绑定策略，从而一定范围内支持 Pod 快速扩容。
- 支持弹性伸缩网卡/ IP，从而可避免 IP 浪费，提高 IP 利用率。
- 预绑定值不可为0，即暂不能支持完全按需分配，节点数过多可能会造成 IP 浪费。

IP 地址管理原理

TKE 组件在每个节点维护一个可弹性伸缩的独占网卡/IP 池。已绑定的独占网卡/IP 数量将被维持在 **Pod 数量 + 最小预绑定数量**及 **Pod 数量 + 最大预绑定数量**之间。

- 当**已绑定数量 < Pod 数量 + 最小预绑定数量**时，会绑定独占网卡/IP 使得**已绑定数量 = Pod 数量 + 最小预绑定数量**。
- 当**已绑定数量 > Pod 数量 + 最大预绑定数量**时，会定时释放独占网卡/IP（约2分钟一次），直到**已绑定数量 = Pod 数量 + 最大预绑定数量**。
- 当**最大可绑定数量 < 当前已绑定数量**时，会直接释放多余的空闲独占网卡/IP，使得**已绑定数量 = 最大可绑定数量**。

使用方法

启用非固定 IP

创建集群选择非固定 IP 模式的 VPC-CNI：集群创建时不勾选**固定Pod IP** 选项。

支持快释放

默认情况，非固定 IP 模式管理的网卡/IP 池采用慢释放策略，默认是2分钟只释放1个多余的网卡/IP，若用户需要更高效的利用 IP，则需要开启快释放，快释放模式下，每2分钟会检查一次网卡/IP 池，释放多余的网卡/IP，直到空闲网卡/IP 数等于最大预绑定值。开启方式如下：

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面找到 **eniipamd** 组件，选择**更新配置**。如下图所示：

4. 在更新配置页面，勾选**快释放**。如下图所示：

5. 单击**完成**。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

- 修改现存的 tke-eni-agent daemonset：`kubectl edit ds tke-eni-agent -n kube-system`。
- 在 `spec.template.spec.containers[0].args` 中加入以下启动参数开启快释放。修改后，agent 会滚动更新生效特性。

```
- --enable-quick-release
```

指定某节点预绑定数量

可通过修改节点对应的 CRD `NEC` 的注解来指定该节点 eni-ip 预绑定的数量，相关的注解为：

```
# 共享网卡模式指定最小预绑定值
"tke.cloud.tencent.com/route-eni-ip-min-warm-target"
# 共享网卡模式指定最大预绑定值
"tke.cloud.tencent.com/route-eni-ip-max-warm-target"
# 独占网卡模式指定最小预绑定值
"tke.cloud.tencent.com/direct-eni-min-warm-target"
# 独占网卡模式指定最大预绑定值
"tke.cloud.tencent.com/direct-eni-max-warm-target"
```

修改方法如下：

```
# 示例, 修改节点 <nodeName> 的最小预绑定 ip 值为1
kubectl annotate nec <nodeName> "tke.cloud.tencent.com/route-eni-ip-min-warm-target"="1" --overwrite
# 示例, 修改节点 <nodeName> 的最大预绑定 ip 值为3
kubectl annotate nec <nodeName> "tke.cloud.tencent.com/route-eni-ip-max-warm-target"="3" --overwrite
```

- 修改后即触发动态预绑定的检查，如果预绑定数量不满足期望，会绑定足够网卡/IP。反之则会解绑网卡/IP。
- 修改时这两个注解必须同时存在，且满足：`0 <= 最小预绑定 <= 最大预绑定`，否则修改失败。

指定某节点最大绑定数量

可通过修改节点对应的 CRD `nec` 的注解来指定该节点网卡/IP 最大绑定的数量，可指定最大的网卡数和单网卡绑定的 IP 数，相关的注解为：

```
# 共享网卡模式指定最大网卡数
kubectl annotate nec <nodeName> "tke.cloud.tencent.com/route-eni-max-attach"="1" --overwrite
# 共享网卡模式指定单网卡绑定的 IP 数
kubectl annotate nec <nodeName> "tke.cloud.tencent.com/max-ip-per-route-eni"="9" --overwrite
# 独占网卡模式指定最大独占网卡数
kubectl annotate nec <nodeName> "tke.cloud.tencent.com/direct-eni-max-attach"="5" --overwrite
```

修改时需保证修改值大于等于节点当前正在使用的网卡/IP 数量，否则修改失败。
修改后即触发动态预绑定的检查，如果 `已绑定数量 > 最大可绑定值` ，则会解绑网卡/IP，使 `已绑定数量 = 最大可绑定值` 。

指定默认预绑定数量

可通过修改默认预绑定数量影响**新增节点的预绑定数量**，存量节点的预绑定数量修改请参考 指定某节点预绑定数量 。

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面找到 eniipamd 组件，选择**更新配置**。如下图所示：

4. 在更新配置页面，填写欲设置的共享网卡/独占网卡模式预绑定默认值。如下图所示：

5. 单击**完成**。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

- 修改现存的 tke-eni-ipamd deployment： `kubectl edit deploy tke-eni-ipamd -n kube-system` 。
- 在 `spec.template.spec.containers[0].args` 中加入以下启动参数修改默认预绑定值。修改后，ipamd 会自动重启并生效。默认值只影响新增的节点：

```
# 共享网卡模式最小预绑定默认值，默认值为 5
- --ip-min-warm-target=3
# 共享网卡模式最大预绑定默认值，默认值为 5
- --ip-max-warm-target=3
# 独占网卡模式最小预绑定默认值，默认值为 1
- --eni-min-warm-target=3
# 独占网卡模式最大预绑定默认值，默认值为 1
- --eni-max-warm-target=3
```


VPC-CNI 模式与其他云资源、IDC 互通

最近更新时间：2024-11-21 14:51:12

VPC-CNI 模式和容器网络属于 VPC 可管理的网段，因此可以直接通过 VPC 的产品功能配置实现与其他云资源、IDC 资源的互通。
腾讯云为您提供丰富的解决方案，可以满足以下需求：

- VPC 内的云服务器、数据库等实例连接公网（Internet）。
- 连接其他 VPC 内实例。
- 与本地数据中心（IDC）互联。

私有网络 VPC 及其多种连接方式详情可参见 VPC 连接方案概述。在 VPC 连接方案概述中，提供了多种连接方式：

- 如何通过普通公网 IP、弹性公网 IP、NAT 网关、负载均衡等实现访问公网（Internet），详情请参见 连接公网。
- 如何通过对等连接、云联网实现连接不同的 VPC，详情请参见 连接其它 VPC。
- 如何通过 VPN 连接、专线接入、云联网实现 VPC 与本地数据中心的通信，详情请参见 连接本地数据中心。

通过这些解决方案，您可以灵活地实现 VPC 内外资源的互通，满足不同业务场景的需求。

VPC-CNI 模式安全组使用说明

最近更新时间：2024-05-13 11:54:01

您可以通过下述方式为 VPC-CNI 模式创建的弹性网卡绑定指定的安全组。

前提条件

- IPAMD 组件版本在 v3.2.0+（可通过镜像 tag 查看）。
- IPAMD 组件启动了安全组能力，启动参数：`--enable-security-groups`（默认未启用）。
- 目前仅支持多 Pod 共享网卡模式。

IPAMD 组件开启安全组特性

> **注意：**
> 默认情况下，集群创建后，ipamd 不会开启安全组能力，此时绑定的弹性网卡未绑定安全组；如果用户对 Pod 安全特性要求较高，建议用户创建集群后及时开启安全组特性，并绑定所需安全组。

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在**集群管理**页面，选择需开启安全组的集群 ID，进入集群详情页。
3. 在集群详情页面，选择左侧**组件管理**，在组件管理页面中，单击 eniipamd 组件右侧的**更新配置**。

4. 在更新配置页面，勾选**安全组**。如果希望继承主网卡安全组，则不指定安全组，否则需指定安全组。

5. 单击**完成**。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

- 修改现存的 tke-eni-ipamd deployment：

```
kubectl edit deploy tke-eni-ipamd -n kube-system
```

- 执行以下命令，在 `spec.template.spec.containers[0].args` 中加入启动参数。
  修改后，ipamd 会自动重启并生效。
  生效后，存量节点上的辅助弹性网卡没有关联安全组的会按以下策略绑定安全组，如果绑定了也会与设置的安全组强同步，除非之前已开启特性，节点安全组已设置。增量节点的弹性网卡则都会绑定以下安全组。


```
- --enable-security-groups
# 如果希望默认继承自主网卡/实例的安全组，则不添加 security-groups 参数
- --security-groups=sg-xxxxxxxx,sg-xxxxxxxx
```

存量节点同步网卡安全组设置的方法

如果想让已设置安全组的存量节点也生效，需要手动禁用安全组，再开启来达到同步。以下为存量节点的同步方法：

1. 给节点加上注解清空并禁用节点的弹性网卡绑定安全组,添加后，节点的存量弹性网卡会解绑所有安全组：

```
kubectl annotate node <nodeName> --overwrite tke.cloud.tencent.com/disable-node-eni-security-groups="yes"
```

2. (第一步执行完后等待2-5s)重新设置为 no 后，则可以重新绑定以上策略配置的安全组：

```
kubectl annotate node <nodeName> --overwrite tke.cloud.tencent.com/disable-node-eni-security-groups="no"
```

功能逻辑

- 若未设置启动参数 `--security-groups`，或者其值为空，则各节点安全组继承自节点实例绑定的安全组（主网卡绑定的安全组）。若特性开启后，节点实例安全组（主网卡安全组）发生变化，辅助网卡的安全组不会进行同步感知，需禁用节点安全组再开启，来达到同步。操作方法请参见 存量节点同步网卡安全组设置的方法。
- 特性开启以后，如果设置了 `--security-groups`，则各节点安全组设置为该安全组集合。
- 特性开启以后，如果变更 `--security-groups` 参数，增量节点安全组设置会与全局参数同步，存量节点安全组设置不会改变，若需同步存量节点安全组设置，则需禁用节点安全组再开启，来达到同步。操作方法请参见 存量节点同步网卡安全组设置的方法。
- 安全组设置的优先级与节点安全组设置的顺序一致，若继承自主网卡，则与主网卡保持一致。
- 执行以下命令可查看节点安全组。其中 `spec.securityGroups` 域包含了节点安全组信息。

  ```
  kubectl get nec <nodeName> -oyaml
  ```

  执行以下命令可修改节点安全组，修改后即刻生效。

  ```
  kubectl edit nec <nodeName>
  ```

- 特性开启以后，存量网卡如果没绑定安全组，则会绑定节点安全组。存量网卡的安全组会与节点安全组强同步，保证与设置的节点安全组保持一致。增量网卡都会绑定节点安全组。

Pod 直接绑定弹性公网 IP 使用说明

最近更新时间：2025-05-23 11:07:32

您可以通过下述方式为 VPC-CNI 模式的 Pod 直接绑定弹性公网 IP（EIP）。

前提条件和限制

- IPAMD 使用的角色策略被授权了 EIP 相关的接口权限。
- 目前 VPC-CNI 独占网卡非固定 IP 模式暂不支持 EIP 功能（v3.3.9及之后版本可支持）。
- 当前集群删除时暂不支持回收该集群自动创建的 EIP。
- Pod 不可以配置 hostNetwork: true。

IPAMD 组件角色添加 EIP 接口访问权限

1. 登录 访问管理控制台，选择左侧的**角色**。
2. 在**访问管理控制台** > 角色 中搜索 IPAMD 组件的相关角色 `IPAMDofTKE_QCSRole`，单击角色名称进入角色详情页面。
3. 在权限设置中，单击**关联策略**。
4. 在弹出的关联策略窗口中，在搜索框中搜索 `QcloudAccessForIPAMDRoleInQcloudAllocateEIP`，然后勾选已创建的预设策略 `QcloudAccessForIPAMDRoleInQcloudAllocateEIP`。单击**确定**，完成为 IPAMD 组件角色添加 EIP 接口访问权限操作。该策略包含了 IPAMD 组件操作弹性公网 IP 所需的所有权限。

> **注意：**
> 通过 自动新建 EIP 或 指定 EIP 的方式成功为 Pod 绑定 EIP 后，还需按照 确保主动外访网络流量走 EIP 进行配置，才能使 Pod 能够正常使用 EIP 的能力。

自动新建 EIP

如需自动关联 EIP，可参考以下 Yaml 示例：

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    k8s-app: busybox
  name: busybox
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: busybox
      qcloud-app: busybox
  serviceName: ""
  template:
    metadata:
      annotations:
        tke.cloud.tencent.com/networks: "tke-route-eni"
        tke.cloud.tencent.com/eip-attributes: '{"Bandwidth":"100","ISP":"BGP"}'
        tke.cloud.tencent.com/eip-claim-delete-policy: "Never"
      creationTimestamp: null
      labels:
        k8s-app: busybox
        qcloud-app: busybox
    spec:
      containers:
      - args:
        - "10000000000"
```

```
        command:
        - sleep
        image: busybox
        imagePullPolicy: Always
        name: busybox
        resources:
          limits:
            tke.cloud.tencent.com/eni-ip: "1"
            tke.cloud.tencent.com/eip: "1"
          requests:
            tke.cloud.tencent.com/eni-ip: "1"
            tke.cloud.tencent.com/eip: "1"
```

- `spec.template.annotations`：`tke.cloud.tencent.com/eip-attributes: '{"Bandwidth":"100","ISP":"BGP"}'` 表明该 Workload 的 Pod 需要自动关联 EIP，且 EIP 的带宽是 100 Mbps，线路类型是 BGP。
- `spec.template.annotations`：`tke.cloud.tencent.com/eip-claim-delete-policy: "Never"` 表明 Workload 的 Pod 的 EIP 也需要固定，Pod 销毁后不能变更。若不需要固定，则不添加该注解。
- 对于需要关联 EIP 的 Pod，您需要在 `spec.template.spec.containers.0.resources` 中添加 requests 和 limits 限制，即 `tke.cloud.tencent.com/eip`，以确保调度器将 Pod 调度到仍有可用 EIP 资源的节点上。

关键配置说明

- 各节点可绑定的 EIP 资源受到相关配额限制和云服务器的绑定数量限制，详情可参见 EIP使用限制。
  各节点可绑定的最大 EIP 数量为**云服务器绑定数量** – 1。
- `tke.cloud.tencent.com/eip-attributes: '{"Bandwidth":"100","ISP":"BGP"}'`：当前只支持配置带宽和线路类型两个参数。`ISP` 参数可配置为 `BGP`、`CMCC`、`CTCC`、`CUCC`，分别对应普通线路 BGP IP、静态单线 IP（网络运营商中国移动、中国电信、中国联通）。若不填写，则默认值为 100Mbps 和 BGP。
- 当前自动申请的 EIP 绑定后不收取 IP 资源费用，访问公网网络默认计费方式为**流量按小时后付费**，详情见 EIP 计费概述。

指定 EIP

如需自动关联指定 EIP，可参考以下 Yaml 示例：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    k8s-app: busybox
  name: busybox
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: busybox
      qcloud-app: busybox
  serviceName: ""
  template:
    metadata:
      annotations:
        tke.cloud.tencent.com/networks: "tke-route-eni"
        tke.cloud.tencent.com/eip-id-list: "eip-xxx1,eip-xxx2"
      creationTimestamp: null
      labels:
        k8s-app: busybox
        qcloud-app: busybox
    spec:
      containers:
      - args:
        - "10000000000"
```

```
        command:
        - sleep
        image: busybox
        imagePullPolicy: Always
        name: busybox
        resources:
          limits:
            tke.cloud.tencent.com/eni-ip: "1"
            tke.cloud.tencent.com/eip: "1"
          requests:
            tke.cloud.tencent.com/eni-ip: "1"
            tke.cloud.tencent.com/eip: "1"
```

- `tke.cloud.tencent.com/eip-id-list: "eip-xxx1,eip-xxx2"` 表明该 Workload 的 Pod 需要自动关联指定 EIP，且第一个副本使用 eipID 为 `eip-xxx1` 的 EIP，第二个副本使用 eipID 为 `eip-xxx2` 的 EIP。当前解析指定策略：Pod 按照其名字末尾的编号依次选用注解中的 EIP，若名字末尾无编号（如 deployment 类型），则随机选用，冲突时只能有一个 Pod 关联成功。建议无编号的 Pod 只指定单个 EIP。
- 对于需要关联 EIP 的 Pod，您需要在 `spec.template.spec.containers.0.resources` 中添加 requests 和 limits 限制，即 `tke.cloud.tencent.com/eip`，以确保调度器将 Pod 调度到仍有可用 EIP 资源的节点上。

确保主动外访网络流量走 EIP

当前集群内默认部署了 `ip-masq-agent` 组件，该组件默认会对集群内 Pod 的主动外访流量以所在节点的地址做 SNAT。此外，如果 VPC 内配置了 NAT 网关，则其对 Pod 的主动外访流量也有影响。因此，如需让 Pod 的主动外访流量走其关联的 EIP，则需修改相关配置和路由策略以达到效果。

去除集群内 SNAT

需修改集群内的 SNAT 规则来让关联 EIP 的 Pod 的主动外访流量不被做 SNAT：

```
kubectl -n kube-system edit cm ip-masq-agent-config
```

在 `data.config` 字段中加入键为 `NonMasqueradeSrcCIDRs` 的新字段，值为已关联 EIP 的 Pod 的**内网 IP** 网段列表，如 IP 为 `172.16.0.2`，则要填写 `172.16.0.2/32`。

> **注意：**
> NonMasqueradeSrcCIDRs 字段对 ip-masq-agent 版本有要求，**v2.6.2**版本以上才支持相应参数配置；如果环境不满足，请升级 ip-masq-agent 的版本。

代码示例如下：

```
apiVersion: v1
data:
  config: '{"NonMasqueradeCIDRs":["172.16.0.0/16","10.67.0.0/16"],"NonMasqueradeSrcCIDRs":
["172.16.0.2/32"],"MasqLinkLocal":true,"ResyncInterval":"1m0s","MasqLinkLocalIPv6":false}'
kind: ConfigMap
metadata:
  name: ip-masq-agent-config
  namespace: kube-system
```

填写后保存退出即生效，该配置会在一分钟内同步热更新。
该字段的作用是网段内的 Pod 主动外访流量不再做节点地址的 SNAT，如果填写较大的网段，则网段内的 Pod 也会不再做 SNAT，请慎重填写。

调整 NAT 网关和 EIP 的优先级

如果集群所在 VPC 内配置了 NAT 网关，请参考文档 调整 NAT 网关和 EIP 的优先级 确保配置正确（查询路由表时要查询 Pod 所在子网关联的路由表），否则 Pod 的主动外访流量可能优先走 NAT 网关，而非 EIP。

EIP 的保留和回收

Pod 启用自动关联 EIP 特性后，网络组件会为该 Pod 在同 namespace 下创建同名的 CRD 对象 `EIPClaim`。该对象描述 Pod 对 EIP 的需求。

对于非固定 EIP 的 Pod，其 Pod 销毁后 `EIPClaim` 也会被销毁，Pod 关联的 EIP 随之销毁回收。而对于固定 EIP 的 Pod，其 Pod 销毁后 `EIPClaim` 仍然保留，EIP 也因此保留。**同名**的 Pod 启动后会使用同名的 `EIPClaim` 关联的 EIP，从而实现 EIP 保留。

下面介绍三种回收 EIP 的方法：过期回收、手动回收及级联回收。

过期回收

新建集群

在 创建集群 页面，容器网络插件选择 **VPC-CNI** 模式并且勾选**开启支持**固定 Pod IP 支持，如下图所示：

在高级设置中设置 IP 回收策略，可以设置 Pod 销毁后多少秒回收保留的固定 IP。如下图所示：

存量集群

对于**存量集群**，也可支持变更：

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在“集群管理”页面，选择需设置过期时间的集群 ID，进入集群详情页。
3. 在集群详情页面，选择左侧**组件管理**。
4. 在组件管理页面中，找到eniipamd组件，选择**更新配置**。

5. 在更新配置页面，填写固定IP回收策略里的过期时间，并点击完成。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

- 修改现存的 tke-eni-ipamd deployment：`kubectl edit deploy tke-eni-ipamd -n kube-system`。
- 执行以下命令，在 `spec.template.spec.containers[0].args` 中加入/修改启动参数。

```
- --claim-expired-duration=1h # 可填写不小于 5m 的任意值
```

手动回收

对于急需回收的 EIP，找到对应的 Pod 的名称空间和名称，执行以下命令通过手动回收：

> **注意：**
> 需保证回收的 EIP 对应的 Pod 已经销毁，否则会再次触发关联绑定 EIP。

```
kubectl delete eipc <podname> -n <namespace>
```

级联回收

目前的固定 EIP 与 Pod 强绑定，而与具体的 Workload 无关（例如 deployment、statefulset 等）。Pod 销毁后，固定 EIP 不确定何时回收。TKE 现已实现删除 Pod 所属的 Workload 后即刻删除固定 EIP。**要求 IPAMD 组件版本在 v3.3.9+（可通过镜像 tag 查看）**。
以下步骤介绍如何开启级联回收：

tke-eni-ipamd 组件版本 >= v3.5.0

1. 登录 容器服务控制台，单击左侧导航栏中**集群**。
2. 在“集群管理”页面，选择需开启级联回收的集群 ID，进入集群详情页。
3. 在集群详情页面，选择左侧**组件管理**。
4. 在组件管理页面中，找到**eniipamd**组件，选择**更新配置**。

5. 在更新配置页面，勾选**级联回收**，并点击完成。

tke-eni-ipamd 组件版本 < v3.5.0 或组件管理中无 eniipamd 组件

1. 修改现存的 tke-eni-ipamd deployment：`kubectl edit deploy tke-eni-ipamd -n kube-system`。
2. 执行以下命令，在 `spec.template.spec.containers[0].args` 中加入启动参数：

```
- --enable-ownerref
```

修改后，ipamd 会自动重启并生效。生效后，增量 Workload 可实现级联删除固定 EIP，存量 Workload 暂不能支持。

配置 EIP 公网地址到 Pod 内的网卡上

可利用 downwardAPI 和 Volumes 映射 Pod 绑定的 EIP 地址到 Pod 内的网卡上，从而在 Pod 内获取 EIP 公网地址，可参考如下 Yaml 进行配置：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    tke.cloud.tencent.com/eip-attributes: ''
  labels:
    app: busybox-eip
  name: busybox-eip
spec:
  initContainers:
  - name: get-eip
    image: busybox
    imagePullPolicy: IfNotPresent
    command: ["sh", "-c"]
    args:
    - while true; do
        if [[ -e /etc/podinfo/eip ]]; then
            ip=$(cat /etc/podinfo/eip);
            echo "$ip\n";
            if [ $ip ];then
                echo "found eip ${ip}, set it to eth0";
                ip addr add $ip dev eth0;
                break;
            else
                echo "wait eip associated";
            fi;
        else
            echo "eip file not found";
        fi;
        sleep 1;
      done;
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /etc/podinfo
```

```
      name: podinfo
  containers:
  - command: ["sleep", "2300000"]
    image: busybox
    imagePullPolicy: IfNotPresent
    name: busybox
    resources:
      requests:
        tke.cloud.tencent.com/eip: "1"
      limits:
        tke.cloud.tencent.com/eip: "1"
  volumes:
     - name: podinfo
       downwardAPI:
         items:
           - path: "eip"
             fieldRef:
               fieldPath: metadata.annotations['tke.cloud.tencent.com/eip-public-ip']
```

- pod 绑定 EIP 成功后，公网地址会写到 pod 注解 `tke.cloud.tencent.com/eip-public-ip` 中，通过 downwardAPI 和 volumes 可挂载到 pod 内。
- 初始化容器 initContainer 执行脚本代码，轮询 EIP 公网地址文件直到文件存在，并将地址配置到 `eth0` 网卡上。

VPC-CNI（eniipamd）组件介绍

最近更新时间：2025-07-04 17:54:12

VPC-CNI 组件（组件管理中名为 eniipamd）包含3个 kubernetes 集群组件，分别是 `tke-eni-agent` 、`tke-eni-ipamd` 和 `tke-eni-ip-scheduler` 。

tke-eni-agent

以 `daemonset` 形式部署在集群中的每个节点上，职责：

- 拷贝 `tke-route-eni` 和 `tke-eni-ipamc` 等 CNI 插件到节点 CNI 执行文件目录（默认为 `/opt/cni/bin` ）。
- 在 CNI 配置目录（默认为 `/etc/cni/net.d/` ）生成 CNI 配置文件。
- 设置节点策略路由和弹性网卡。
- Pod IP 分配/释放的 GRPC Server。
- 定期进行 IP 垃圾回收，回收 Pod 已不在节点上的 IP。
- 通过 kubernetes 的 device-plugin 机制 设置网卡和 IP 的扩展资源。

tke-eni-ipamd

以 `deployment` 形式部署在集群中的特定节点或 master 上，职责：

- 创建管理 CRD 资源（nec, vipc, vip, veni）。
- 非固定 IP 模式下，依据节点需求和状态创建/绑定/解绑/删除弹性网卡，分配/释放弹性网卡 IP。
- 固定 IP 模式下，依据 Pod 需求和状态创建/绑定/解绑/删除弹性网卡，分配/释放弹性网卡 IP。
- 节点弹性网卡安全组管理。
- 依据 Pod 需求创建/绑定/解绑/删除弹性公网 IP。

tke-eni-ip-scheduler

以 `deployment` 形式部署在集群中的特定节点或 master 上，仅固定 IP 模式会部署，为调度扩展插件，职责：

- 多子网情况下，需要让已固定 IP 的 Pod 调度到指定子网的节点。
- 固定 IP 模式下，判断 Pod 调度的节点对应子网 IP 是否充足。

组件权限说明

> **说明：**
> **权限场景**章节中仅列举了组件核心功能涉及到的相关权限，完整权限列表请参考**权限定义**章节。

tke-eni-agent 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要修改网络相关的内核参数，如 net.ipv4.ip_forward，net.ipv4.rp_filter等，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
分配 IP 过程中，需要获取 pod 和 node 相关信息。	pods、namespaces、nodes	get/list/watch
获取网络配置信息。	configmaps	get/list/watch
管理 node 的相关网络扩展资源，如 tke.cloud.tencent.com/eni-ip 等。	nodes/status	get/list/watch/patch
通过自定义对象获取分配 IP、网卡等网络配置信息，并与 eni-ipamd 组件配合工作。	networking.tke.cloud.tencent.com groups	get/list/watch/delete/update
通过 events 暴露组件的工作状态，节点网络的相关变更信息。	events	get/list/watch/create/update/patch

权限定义

```
kind: ClusterRole
metadata:
  name: tke-eni-agent
rules:
- apiGroups: [""]
  resources:
  - pods
  - namespaces
  - nodes
  - configmaps
  verbs: ["list", "watch", "get"]
- apiGroups: [""]
  resources:
  - nodes/status
  verbs: ["list", "watch", "get", "patch"]
- apiGroups: ["networking.tke.cloud.tencent.com"]
  resources:
    - underlayips
    - nodeeniconfigs
    - vpcipclaims
    - vpcips
    - vpcenis
  verbs: ["get", "list", "watch", "delete", "update"]
- apiGroups: [""]
  resources:
    - events
  verbs: ["list", "watch", "get", "update", "patch", "create"]
```

tke-eni-ipamd 权限

权限说明

该组件权限是当前功能实现的最小权限依赖。

权限场景

功能	涉及对象	涉及操作权限
分配 IP 的过程中，需要获取 Pod 和 Node 的相关信息。	pods、namespaces、nodes、nodes/status	get/list/watch
给超级节点的 Pod 分配 IP 的过程中，需要将分配信息更新到 Pod 的注解中。	pods	update/patch
全局路由工作模式下，需要将分配给节点的 podCIDR 写到 nodes 对象上，同时与节点自动扩缩容配合工作时，需要更新 nodes 的 conditions 和 taints。	nodes、nodes/status	update/patch
多副本运行功能基于 LeaderElection 实现，LeaderElection 需要 configmaps 或 endpoints 的相关读写权限，同时运行信息通过 events 暴露。	configmaps、endpoints、events	get/list/watch/create/update/patch
固定 IP 的 Pod 销毁时，需要获取所属的 workload 信息来判断是否需要释放固定 IP。	statefulsets、deployments	get/list/watch
使用自定义对象来管理相关网络资源（弹性网卡、IP、安全组等）。	customresourcedefinitions	create/update/get
	networking.tke.cloud.tencent.com apiGroups	get/list/watch/create/update/patch/delete

需要获取原生节点的相关信息。	node.tke.cloud.tencent.com apiGroups	get/list/watch
注册节点相关能力需要与 cilium 组件配合工作。	cilium.io apiGroups	get/list/watch/create/update/patch/delete

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: tke-eni-ipamd
rules:
- apiGroups: [""]
  resources:
  - pods
  - namespaces
  - nodes
  - nodes/status
  verbs: ["list", "watch", "get", "patch", "update"]
- apiGroups: [""]
  resources:
  - configmaps
  - endpoints
  - events
  verbs: ["get", "list", "watch", "update", "create", "patch"]
- apiGroups: ["apps", "extensions"]
  resources:
  - statefulsets
  - deployments
  verbs: ["list", "watch", "get"]
- apiGroups: ["apiextensions.k8s.io"]
  resources:
  - customresourcedefinitions
  verbs: ["create", "update", "get"]
- apiGroups: ["networking.tke.cloud.tencent.com"]
  resources:
  - staticipconfigs
  - underlayips
  - nodeeniconfigs
  - vpcipclaims
  - vpcips
  - eipclaims
  - vpcenis
  verbs: ["create", "update", "delete", "get", "list", "watch", "patch"]
- apiGroups: ["node.tke.cloud.tencent.com"]
  resources:
  - machines
  verbs: ["get", "list", "watch"]
- apiGroups: [ "cilium.io" ]
  resources:
  - ciliumnodes
  - ciliumnodes/status
  - ciliumnodes/finalizers
  verbs: [ "create", "update", "delete", "get", "list", "watch", "patch" ]
```

tke-eni-ip-scheduler 权限

权限说明

- 该组件权限是当前功能实现的最小权限依赖。
- 需要挂载主机 /var/lib/kubelet 相关目录到容器来完成 volume 的 mount/umount，所以需要开启特权级容器。

权限场景

功能	涉及对象	涉及操作权限
需要扩展 bindVerb，以解决 Pod 并发绑定时 IP 分配冲突的问题。	pods/binding	get/list/watch/create/update/patch
多副本运行功能基于 LeaderElection 实现，LeaderElection 需要 configmaps 或 endpoints 的相关读写权限，同时运行信息通过 events 暴露。	configmaps、endpoints、events	get/list/watch/create/update/patch
扩展调度时需要获取 pod 和 nodes 的相关信息。	pods、namespaces、nodes、nodes/status	get/list/watch
扩展调度时需要与组件自定义对象进行交互，从而实现 IP 的完整分配，解决 IP 分配冲突的问题。	networking.tke.cloud.tencent.com groups	get/list/watch/update

权限定义

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: tke-eni-ip-scheduler
rules:
  - apiGroups: [""]
    resources:
      - pods/binding
    verbs: ["get", "list", "watch", "update", "create", "patch"]
  - apiGroups: [""]
    resources:
      - ["configmaps", "endpoints", "events"]
    verbs: ["get", "list", "watch", "update", "create", "patch"]
  - apiGroups: [""]
    resources:
      - ["pods", "namespaces", "nodes", "nodes/status"]
    verbs: ["list", "watch", "get"]
  - apiGroups: ["networking.tke.cloud.tencent.com"]
    resources:
      - ["nodeeniconfigs", "vpcipclaims", "vpcips"]
    verbs: ["get", "list", "watch", "update"]
```

VPC-CNI 模式 Pod 数量限制

最近更新时间：2024-09-12 11:05:31

本文说明 VPC-CNI 各网络模式 Pod 数量默认限制。

共享网卡模式 Pod 数量限制

共享网卡的 Pod 数量受限于节点可绑定的网卡数量和单网卡可绑定的 IP 数量，默认情况下，多网卡的**单节点 PodIP 数量上限 = 最大可绑定辅助网卡数 * 单网卡可绑定辅助 IP 数**，而单网卡的**单节点 PodIP 数量上限 = 单网卡可绑定辅助 IP 数**。默认情况详见下表：

CPU 核数	1	2-6	8-10	>=12
最大可绑定辅助弹性网卡	1	3	5	7
单网卡可绑定辅助 IP 数	5	9	19	29
单节点 Pod IP 上限（多网卡）	5	27	95	203
单节点 Pod IP 上限（单网卡）	5	9	19	29

注意：
支持多网卡的组件版本（非固定 IP 模式 ≥ v3.3，固定 IP 模式 ≥ v3.4）。

各机型可绑定的网卡数量和单网卡可绑定的 IP 数量略有差异， 详情见 弹性网卡使用限制 。

独占网卡模式 Pod 数量限制

独占网卡的 Pod 数量只受限于节点可绑定的网卡数量，同时只支持 S5、SA2、IT5、SA3 等部分机型，默认情况详见下表（未涉及机型请参考 弹性网卡使用限制 弹性网卡配额）：

CPU核数 机型	1	2	4	>=8	>=128
S5	4	9	19	39	23
SA2	4	9	19	39	23
IT5	4	9	19	39	23
SA3	4	9	15	15	15

Cilium-Overlay 模式
Cilium-Overlay 模式介绍

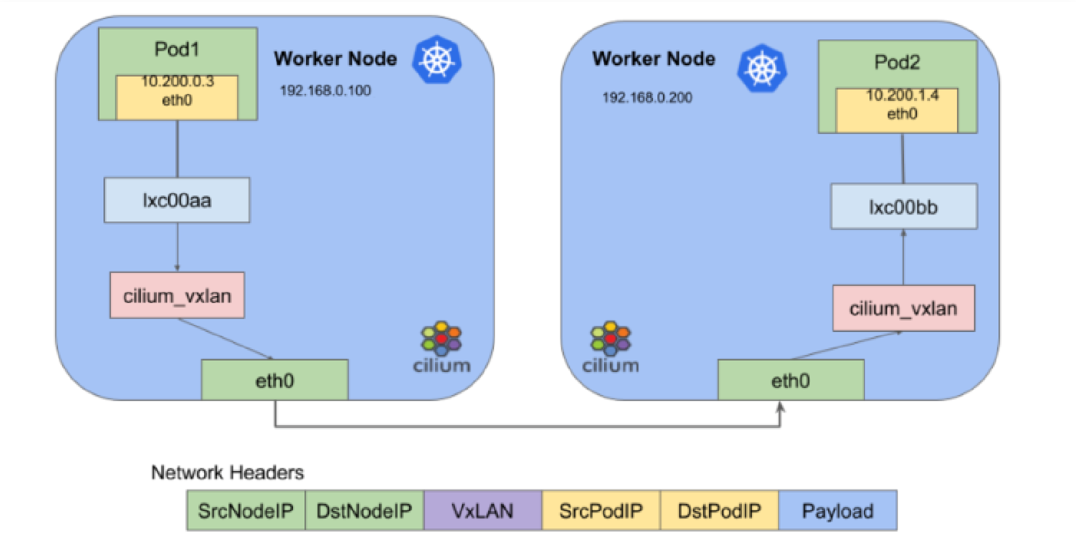
最近更新时间：2024-11-29 17:26:43

使用原理

Cilium-Overlay 网络模式是容器服务 TKE 基于 Cilium VXLan 实现的容器网络插件，实现分布式云场景中，第三方节点添加到 TKE 集群的网络管理。该网络模式特征如下：

- 云上节点和第三方节点共用指定的容器网段。
- 容器网段分配灵活，容器 IP 段不占用 VPC 的其他网段。
- 使用 Cilium VXLan 隧道封装协议构建 Overlay 网络。

云上 VPC 网络和第三方节点 IDC 网络通过云联网互通后，跨节点 Pod 访问原理如下图所示：



> **说明：**
> - 由于 Cilium-Overlay 模式存在性能损耗，因此此模式只支持分布式云中第三方节点场景，不支持只存在云上节点场景，分布式云中第三方节点详情可参见 注册节点概述 。
> - Cillium 模式不支持 Webhook。

使用限制

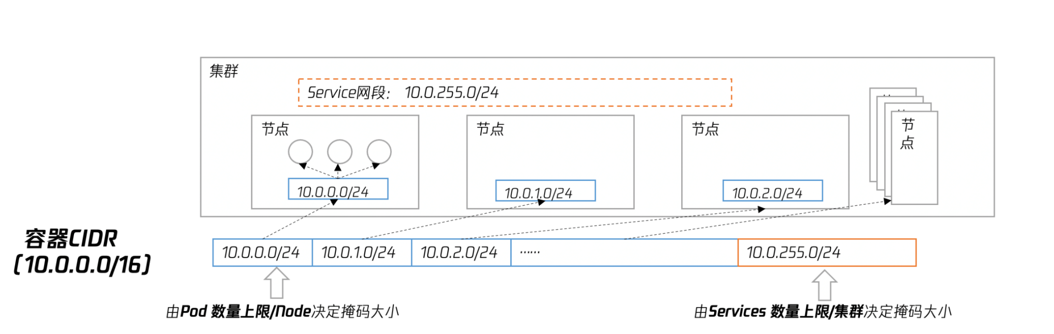
- 使用 Cilium VXLan 隧道封装协议，有10%以内的性能损耗。
- Pod IP 在集群外不能直接访问。
- 需从指定子网获取 2 个 IP 创建内网负载均衡，满足 IDC 中第三方节点访问 APIServer 和云上公共服务。
- 集群网络和容器网络网段不能重叠。
- 不支持固定 Pod IP。

容器 IP 分配机制

容器网络名词介绍和数量计算可参见 容器网络说明 。

Pod IP 分配

工作原理如下图所示：



- 集群中节点包括云上节点和第三方节点。
- 集群的每一个节点会使用容器 CIDR 中的指定大小的网段用于该节点下 Pod 的 IP 地址分配。
- 集群的 Service 网段会选用容器 CIDR 中最后一段指定大小的网段用于 Service 的 IP 地址分配。
- 节点释放后，使用的容器网段也会释放回 IP 段池。
- 扩容节点自动按顺序循环选择容器 CIDR 大段中可用的 IP 段。

Dataplane V2 转发模式

Dataplane V2 介绍

最近更新时间：2024-11-29 17:09:52

TKE 推出新一代 Kubernetes 网络数据平面——Dataplane V2。与之前版本相比，Dataplane V2 具有以下优势：

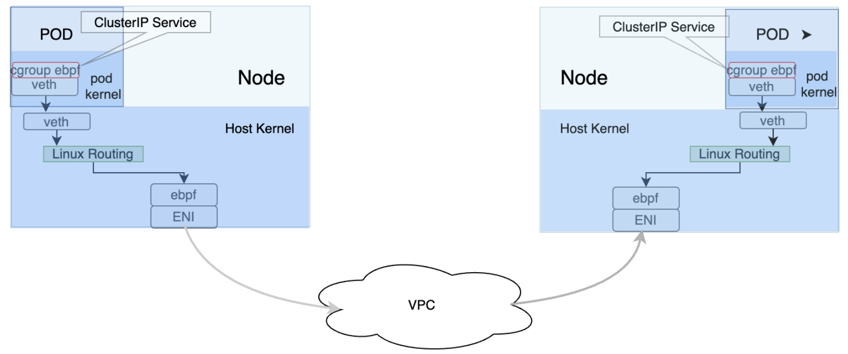
- 解决了 kube-proxy 在 iptables 模式下大量 Service 情况下的控制面性能问题。
- 原生支持 NetworkPolicy，无需额外安装 NetworkPolicy 插件。
- 支持部署 Hubble，增强集群网络的可观测性。

原理

TKE Dataplane V2 基于 eBPF 实现了东西向的 Service（包括 ClusterIP 和 NodePort）和 NetworkPolicy。当数据包从 Pod 网卡发出或到达 TKE 节点时，内核中的 eBPF 程序会决定如何路由和处理数据包。

Service 实现

TKE Dataplane V2 采用 eBPF 替换了原有的 iptables 和 IPVS 转发模式。当 Service 创建、删除等变更时，节点上会更新相应的 eBPF 转发逻辑；后续报文转发时，会判断报文是否是 Service 相关 IP，并进行对应的转发处理。如下图所示：

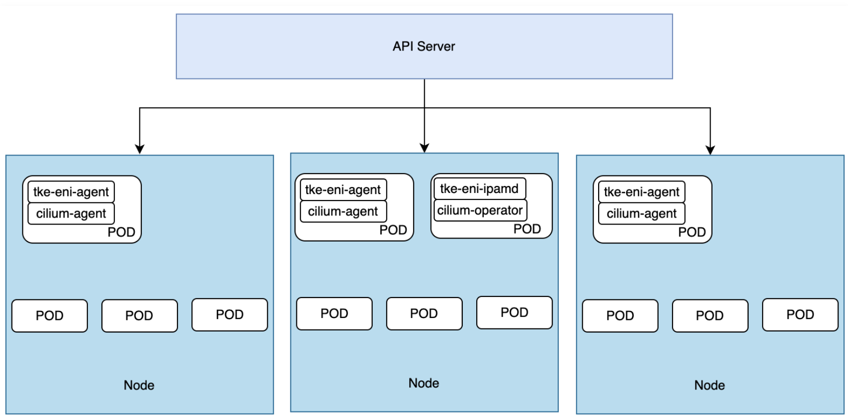


NetworkPolicy 实现

使用 eBPF 取代原有的 iptables，实现 Pod 的网络策略。不会在宿主机上产生大量的 iptables 规则，降低网络策略对性能的影响。

部署方式

Dataplane V2 基于 Cilium 实现了 ClusterIP 和 NodePort 类型的 Service，完全替代了 kube-proxy。为简化 Cilium 的管理，我们将 cilium-agent 容器融入了 TKE 网络插件 tke-eni-agent，将 cilium-operator 容器融入了 TKE 网络控制器 tke-eni-ipamd。如下图所示：



方案优势

相较于 kube-proxy IPVS 模式

- **数据面**：Dataplane V2 的转发性能比 IPVS 可以提高15%-20%。

	1service		1000service		5000 service		10000 service	
	4核	8核	4核	8核	4核	8核	4核	8核
ipvs	18300	25700	18000	25100	17400	24000	-	-
Iptables	26500	28700	18900	25600	16000	23000	-	-
dataplabev2	21000	28500	21000	28500	21000	28500	21000	28500

说明：
以上所有指标使用以下命令进行测试：

```
./wrk -t16 -c2400 -d20s -H "Connection: close" --latency http://clusterip:80/
```

- **控制面**：在 Service 变更频繁或后端变更频繁的场景下，kube-proxy IPVS 模式下的 kube-proxy 进程会长时间占用超过一个 CPU 核心，时长和受影响的 Service 成正比。例如，500个 Service 受影响时，需要持续大约25秒。而在 Dataplane V2 情况下，基本无明显感知。

相较于 kube-proxy iptables 模式

- 当 Service 数量达到5000个时，控制面性能会急剧下降，甚至导致系统不可用。而 Dataplane V2 即使在 Service 规模超过1万时，性能基本不受影响。
- iptables 方案下，转发性能随着 Service 数量的增加而下降；Dataplane V2 的转发性能与 Service 数量基本无关联。

	1service		1000service		5000 service		10000 service	
	4核	8核	4核	8核	4核	8核	4核	8核
ipvs	18300	25700	18000	25100	17400	24000	-	-
Iptables	26500	28700	18900	25600	16000	23000	-	-
dataplabev2	21000	28500	21000	28500	21000	28500	21000	28500

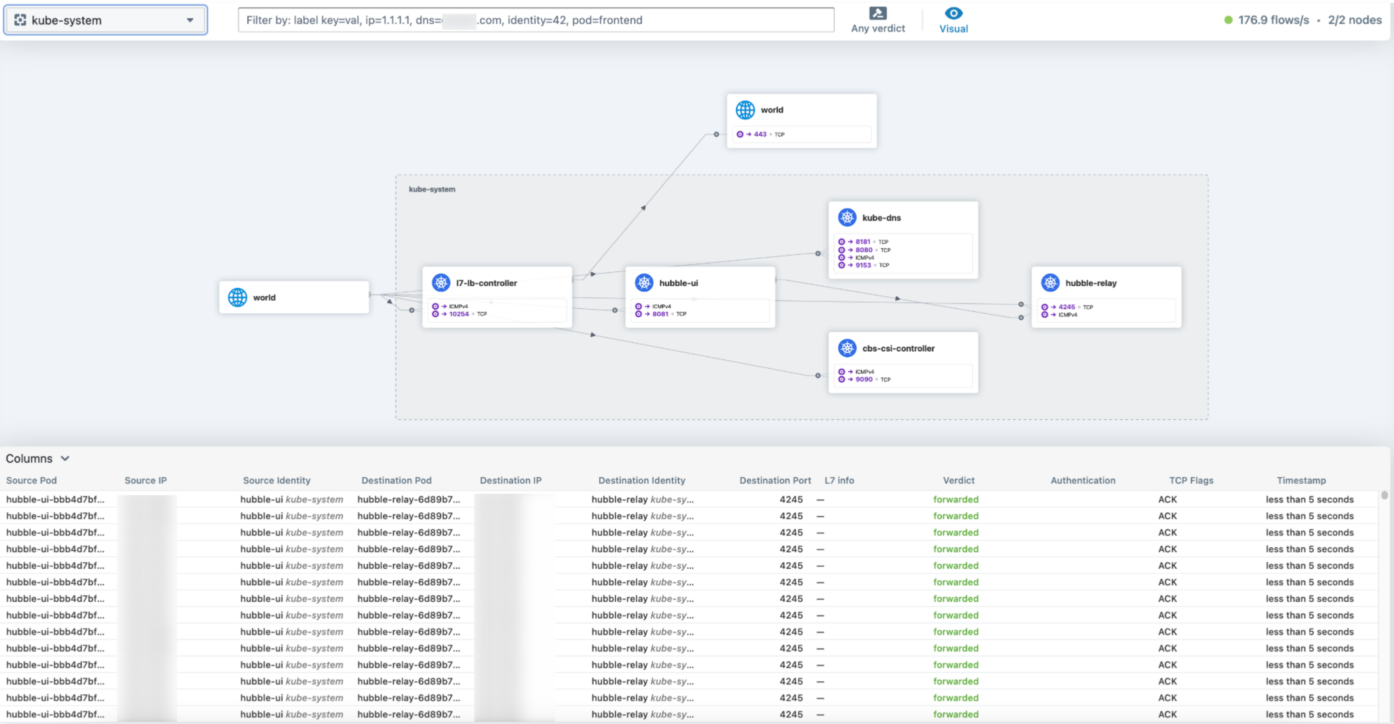
- iptables 方案在 Service 数量较大时，周期对账会占用一定的 CPU 资源。Dataplane V2 无需对账机制，可节省节点 CPU 资源。

安全方面

Dataplane V2 原生支持 NetworkPolicy，无需额外部署新的组件，有效节省了节点的内存和 CPU 资源。

可观测性增强

相比于 iptables/IPVS，eBPF 程序可以更方便地暴露 Service 访问和 NetworkPolicy 裁决信息，结合 Kubernetes 元数据，提升了网络的可观测性。在 Dataplane V2 集群中支持部署 Cilium Hubble，可以对容器网络流量进行可视化展示，实现网络架构和业务拓扑关系的可观测性。


kube-system
Filter by: label key=val, ip=1.1.1.1, dns= .com, identity=42, pod=frontend
Any verdict
Visual
176.9 flows/s • 2/2 nodes
world
kube-system
kube-dns
l7-lb-controller
hubble-ui
hubble-relay
world
cbs-csi-controller
Columns
Source Pod
Source IP
Source Identity
Destination Pod
Destination IP
Destination Identity
Destination Port
L7 info
Verdict
Authentication
TCP Flags
Timestamp
hubble-ui-bbb4d7bf...
hubble-ui kube-system
hubble-relay-6d89b7...
hubble-relay kube-sy...
4245
forwarded
ACK
less than 5 seconds

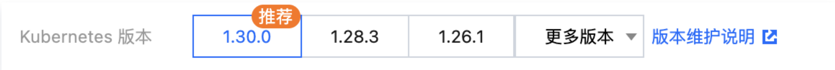
使用 Dataplane V2

最近更新时间：2024-12-06 14:19:43

本文向您介绍如何通过控制台启用 Dataplane V2。

操作步骤

1. 登录 容器服务控制台 。
2. 创建集群。参考 创建集群，在创建集群页面中，选择 Kubernetes 版本为 v1.24 及以上的集群。如下图所示：



3. 选择操作系统。选择 TencentOS Server 3.1 (TK4) 内核。如下图所示：

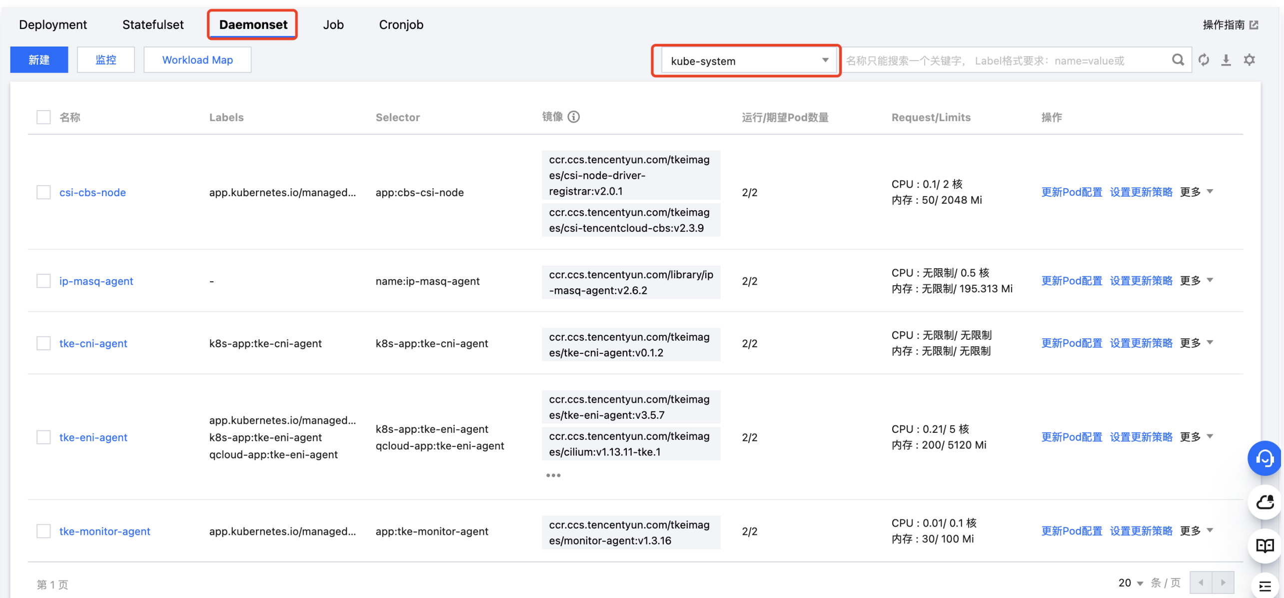
> **注意：**
> - 操作系统仅支持 TencentOS Server 3.1/3.2。启用 Dataplane V2 特性后，集群内不支持添加其他版本操作系统的节点。
> - 暂不支持 NodeLocalDNSCache。
> - 目前默认限制节点数量为500，如有需要，可通过 提交工单 提升规格。

4. 单击**高级设置**，开启 “Dataplane V2”。如下图所示：

5. 按照页面提示逐步操作，完成集群创建。

> **说明：**
> 使用 Dataplane V2 转发模式后，集群中将不再安装 kube-proxy 组件。

在控制台的**工作负载 > Daemonset** 界面，kube-proxy 组件将不再显示。



可观测性
容器服务可观测体系概述

最近更新时间：2025-06-04 18:05:02

容器服务可观测架构概述

可观测性（Observability）是指通过系统输出的外部信息（如日志、指标、追踪数据等）来推断系统内部状态的能力。它强调在无需侵入系统或预定义监控规则的情况下，仍能实时理解系统的运行状态、诊断问题并分析未知行为。
Kubernetes 的可观测性体系是一套针对容器化、动态、分布式环境的监控与诊断框架，旨在通过多维度数据（日志、指标）的采集、关联和分析，实现对集群、节点、Pod、容器及应用的全面状态感知。
本文介绍腾讯云容器服务 TKE 可观测能力，以帮助您对容器服务可观测性能力有一个全局的了解。以下分别是指标、日志、链路追踪的可观测性能力介绍。

指标可观测性

指标可观测性是容器化环境监控的核心支柱，它通过采集、存储和分析随时间变化的数值型数据（Metrics），提供对系统运行状态、资源利用率和应用性能的量化洞察。指标能够量化 TKE 集群整体的资源利用率、调度效率、控制平面健康状态以及分布式系统的协同性能，旨在确保集群高效稳定运行。

可观测能力	说明	参考文档
云监控	容器服务 TKE 新建的集群默认安装并集成云监控。可从容器服务控制台上开箱即用的获取集群、容器、Pod 等维度的基础指标。	查看监控数据
腾讯云 Prometheus 监控服务	Prometheus 监控服务（TencentCloud Managed Service for Prometheus，TMP）是基于开源 Prometheus 构建的高可用、全托管的服务，与容器服务（TKE）高度集成，兼容开源生态丰富多样的应用组件，结合腾讯云可观测平台的告警功能和 Prometheus Alertmanager 能力，提供多种开箱即用的预置监控大盘，为您提供免搭建的高效运维能力，减少开发及运维成本。	Prometheus 监控
自建 Prometheus 监控	提供自建 Prometheus 监控 TKE 集群的方案。	自建 Prometheus 监控 TKE 集群

日志可观测性

日志可观测性是容器化环境中记录、分析和诊断离散事件的核心能力。它通过捕获系统、应用和组件生成的文本流，提供上下文丰富的运行证据，是排查复杂问题的“终极武器”。
Kubernetes 事件日志是集群内部的审计日志，记录了从节点、Pod、服务到控制器等核心对象的关键状态变更和异常信息。它提供了理解集群内部运作机制和诊断系统级问题的黄金线索，是 TKE 可观测性的重要组成部分。

可观测能力	说明	参考文档
日志监控	应用日志监控不仅是故障排查的“最后一道防线”，更是理解系统行为、优化性能和保障安全的核心手段。通过腾讯云日志服务提供的一站式日志服务平台，提供了从日志采集、日志存储到日志检索，图表分析、监控告警、日志投递等多项服务，团队可以实现从被动响应到主动预防的转变，为业务稳定性和用户体验提供坚实基础。	采集容器日志到 CLS
事件监控	事件监控是 Kubernetes 从事件角度出发的另一种监控方式，可以弥补资源监控在实时性、准确性和场景上的缺陷。开发者可以通过获取事件，实时诊断集群的异常与问题。推荐使用腾讯云日志服务 CLS（Cloud Log Service）产品实现事件的监控。	事件日志

链路追踪可观测性

链路追踪可观测性旨在通过采集、分析和可视化应用的运行时数据，帮助开发者快速定位性能瓶颈、异常根因，并优化用户体验。

可观测能力	说明	参考文档
腾讯云应用性能监控 APM	腾讯云应用性能监控 APM 是一款应用性能管理平台，基于实时的多语言应用探针全量采集技术，为您提供分布式应用性能分析和故障自检能力，全方位保障系统的可用性和稳定性。通过调用链分析、接口分析、数据库调用监控等能力协助您在复杂的业务系统快速定位性能问题，降低 MTTR（平均故障恢复时间）。实时了解并追踪应用性能情况，提升用户体验。	应用性能管理

相关文档

- 指标监控，请参见 基础资源监控 、 监控及告警指标列表 、 应用性能监控 、 查看 TKE 集群控制面组件监控 。
- 日志采集，请参见 日志采集 、 事件日志 、 审计日志 。
- Prometheus 监控，请参见 Prometheus 监控服务 、 腾讯云 Prometheus 一键关联监控容器服务 。

日志管理

日志采集概述

最近更新时间：2025-04-17 17:36:02

操作场景

日志采集功能是容器服务 TKE 为用户提供的集群内日志采集工具，可以将集群内服务或集群节点特定路径文件的日志发送至 腾讯云日志服务 CLS 、消息队列 CKafka 。日志采集功能适用于需要对 Kubernetes 集群内服务日志进行存储和分析的用户。
日志采集功能需要为每个集群手动开启并配置采集规则。日志采集功能开启后，日志采集 Agent 会在集群内以 DaemonSet 的形式运行，并根据用户通过日志采集规则配置的采集源、CLS 日志主题和日志解析方式，从采集源进行日志采集，将日志内容发送到日志消费端。您可参考以下步骤为集群开启日志采集功能。

前提条件

- 请在开启前保证集群节点上有足够资源。开启日志采集功能会占用您集群的部分资源。
  - 占用 CPU 资源：0.11 – 1.1核，日志量过大时可根据情况自行调大。
  - 占用内存资源：24 – 560MB，日志量过大时可根据情况自行调大。
  - 日志长度限制：单条512K，如超过会截断。
- 若使用日志采集功能，请确认 Kubernetes 集群内节点能够访问日志消费端。且以下日志采集功能仅支持 Kubernetes 1.10及以上版本集群。

基本概念

日志采集 Agent

TKE 用于采集日志信息的 Agent，采用 Loglistener，在集群内以 DaemonSet 的方式运行。

日志规则

用户可以使用日志规则指定日志的采集源、日志主题、日志解析方式和配置过滤器。
- 日志采集 Agent 会监测日志采集规则的变化，变化的规则会在最多10s内生效。
- 多条日志采集规则不会创建多个 DaemonSet，但过多的日志采集规则会使得日志采集 Agent 占用的资源增加。

日志源

包含指定容器标准输出、容器内文件以及节点文件。
- 在采集容器标准输出日志时，用户可选择所有容器、或指定工作负载和指定 Pod Labels 内的容器服务日志作为日志的采集源。
- 在采集容器文件路径日志时，用户可指定工作负载或 Pod Labels 内容器的文件路径日志作为采集源。
- 在采集节点文件路径日志时，用户可设定日志的采集源为节点文件路径日志。

消费端

用户选择日志服务 CLS 的日志集和日志主题作为消费端。

提取模式

日志采集 Agent 支持将采集到的日志以单行全文、多行全文、单行 – 完全正则、多行 – 完全正则、JSON、分隔符、组合解析的形式发送至用户指定的日志主题。

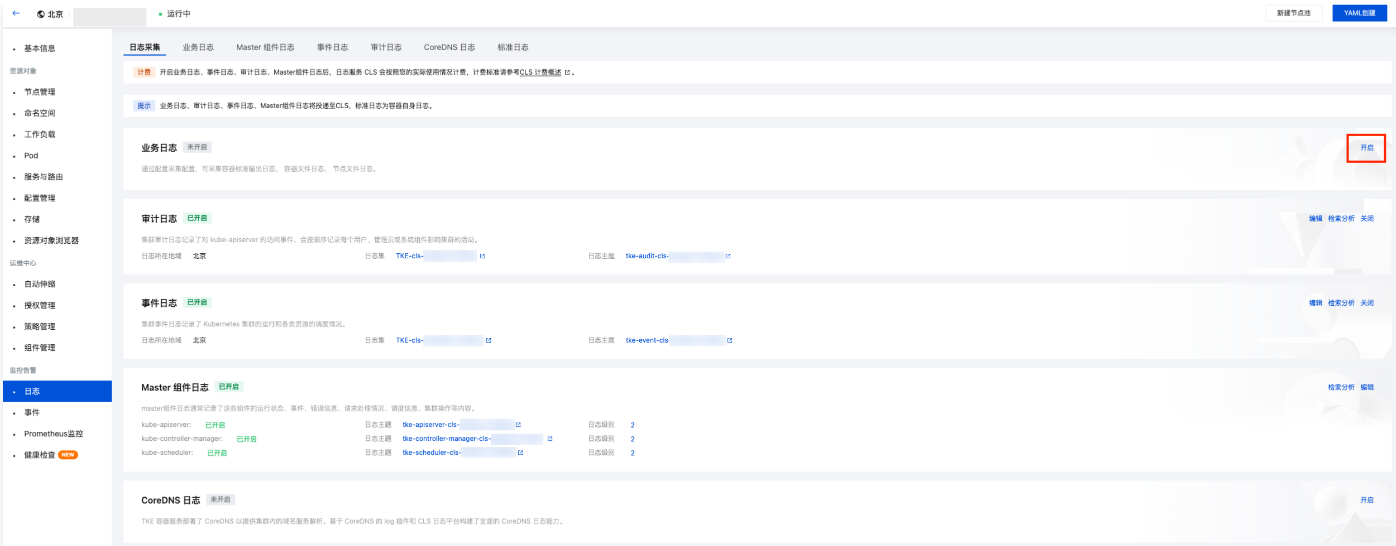
过滤器

开启过滤器后可以根据用户指定的规则采集部分日志，key 支持完全匹配，过滤规则支持正则匹配，如仅采集 `ErrorCode = 404` 的日志。

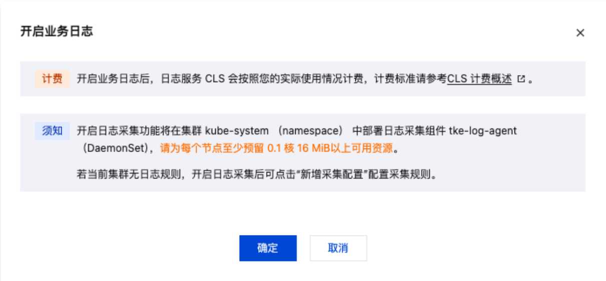
操作步骤

开启日志采集

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击**业务日志**右侧的**开启**。如下图所示：



4. 在开启业务日志中，单击**确定**。如下图所示：



5. 开启后，您可以在日志采集页查看日志采集状态。如下图所示：

关闭日志采集

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击**业务日志**右侧的**关闭**。如下图所示：

4. 在关闭业务日志采集中，勾选“我已知悉关闭须知，确认关闭当前集群业务日志采集”，单击**确定**。如下图所示：

采集容器日志到 CLS

最近更新时间：2025-04-24 15:49:12

本文将介绍如何在容器服务控制台配置日志采集规则并投递到 腾讯云日志服务 CLS 。

使用限制

配置日志采集规则前，需了解 Loglistener 采集数据时在文件采集、配置、资源、性能、错误处理等方面的能力与限制，以及相关使用说明。详情请参见 LogListener 限制说明 。

操作步骤

创建日志采集规则

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击**业务日志**右侧的**新增采集配置**。如下图所示：

4. 在**新建日志采集规则**页面中配置日志服务消费端，在**消费端 > 类型**中选择 CLS。如下图所示：

- 收集规则名称：您可以自定义日志收集规则名称。
- 日志所在地域：CLS 支持跨地域日志投递，您可通过单击**修改**选择日志投递的目标地域。
- 日志集：根据日志所在地域展示您已经创建的日志集，若现有日志集不合适，您可以在 日志服务控制台 新建日志集。操作详情见 创建日志集 。
- 日志主题：选择日志集下对应的日志主题，支持“自动创建”和“选择已有”日志主题两种模式。

- 高级设置：
  - 默认元数据：CLS 默认将元数据 `pod_name` 、 `namespace` 、 `container_name` 设为索引用于日志检索。
  - 自定义元数据：支持自定义元数据及日志索引。

> **说明：**
> - CLS 不支持国内和境外地域之间跨地域日志投递；针对 CLS 未开通日志服务的地域仅支持就近投递，如深圳集群采集的容器日志仅支持投递到广州，天津集群采集的容器日志仅支持投递到北京；详情可通过控制台查看。
> - 一个日志主题目前仅支持配置一类日志，即日志、审计、事件不可采用同一个 topic，会产生覆盖情况，请确保所选日志主题没有被其他采集配置占用；若日志集下已存在500个日志主题，则不能新建日志主题。
> - 自定义元数据和基于元数据开启索引创建后不能修改，您可前往 日志服务控制台 修改。

5. 选择采集类型并配置日志源，目前采集类型支持**容器标准输出**、**容器文件路径**和**节点文件路径**。

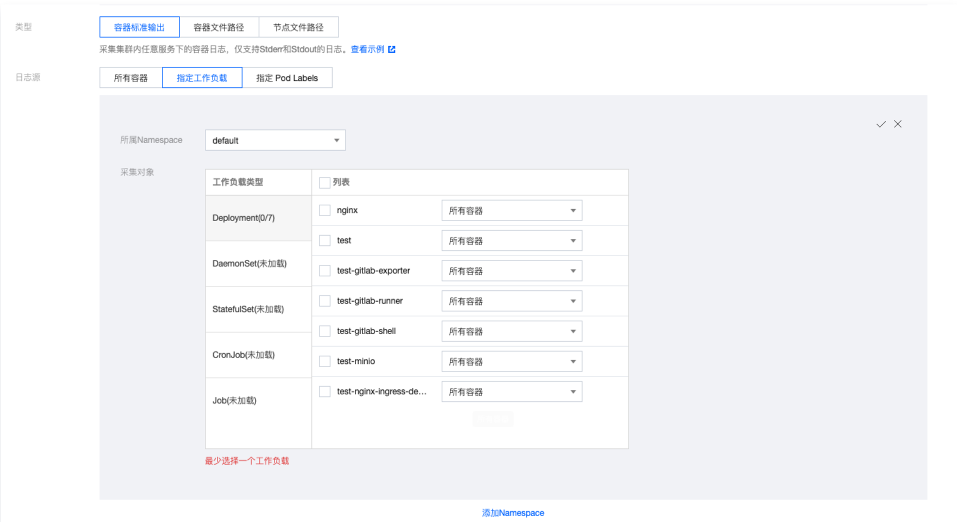
容器标准输出日志

> **说明：**
> 采集集群内任意服务下的**容器日志**，仅支持 Stderr 和 Stdout 的日志。

日志源支持**所有容器**、**指定工作负载**、**指定 Pod Labels** 三种类型。如下图所示：

- **所有容器**

- **指定工作负载**



- **指定 Pod Labels**

容器内文件日志

> **说明：**
> 采集集群内指定容器内的**文件**日志。

- 日志源支持**指定工作负载、指定 Pod Labels** 两种类型。
- 采集文件路径支持文件路径和通配规则，例如当容器文件路径为 `/opt/logs/*.log`，可以指定采集路径为 `/opt/logs`，文件名为 `*.log`。如下图所示：

注意：

“容器文件路径” **不能为符号链接或硬链接**，否则会导致软链接的实际路径在采集器的容器内不存在，采集日志失败。

节点文件日志

说明：

采集集群内指定**节点**路径的文件。

- 采集路径支持以文件路径和通配规则的方式填写，例如当需要采集所有文件路径形式为 `/opt/logs/service1/*.log`，`/opt/logs/service2/*.log`，可以指定采集路径的文件夹为 `/opt/logs/service*`，文件名为 `*.log`。
- 您可根据实际需求自定义添加 Key-Value 形式的 metadata，metadata 将会添加到日志记录中。

注意：

- “节点文件路径” **不能为软链接或硬链接**，否则会导致软链接的实际路径在采集器不存在，采集日志失败。
- 一个节点日志文件只能被一个日志主题采集。

说明：

- 请确保 Pod 的 label 配置正确，即 Pod 的 label，而非工作负载的 label。您可以按照以下步骤查询：

  1.1 登录 容器服务控制台，选择左侧导航栏中的**集群**。

  1.2 在**集群管理**页面，选择集群 ID，进入集群的基本信息页面。

  1.3 在**工作负载**中，选择具体的工作负载名称，进入工作负载详情页。

  1.4 在 **YAML 页签**中，查看 template 下的 labels。如下图所示：

- 对于“容器的标准输出”及“容器内文件”（不包含“节点文件路径”即 hostPath 挂载），除了原始的日志内容， 还会带上容器或 kubernetes 相关的元数据（例如：产生日志的容器 ID）一起上报到 CLS，方便用户查看日志时追溯来源或根据容器标识、特征（例如：容器名、labels）进行检索。
  容器或 kubernetes 相关的元数据请参考下方表格：

字段名	含义
container_id	日志所属的容器 ID。
container_name	日志所属的容器名称。
image_name	日志所属容器的镜像名称 IP。
namespace	日志所属 pod 的 namespace。
pod_uid	日志所属 pod 的 UID。
pod_name	日志所属 pod 的名字。
pod_ip	日志所属 pod 的IP地址。
cluster_id	日志所属的集群 ID
pod_label_{label name}	日志所属 pod 的 label（例如一个 pod 带有两个 label：app=nginx，env=prod，则在上传的日志中会附带两个 metadata：pod_label_app:nginx，pod_label_env:prod）。

6. 配置采集策略。您可以选择**全量**或者**增量**。
   - 全量：全量采集指从日志文件的开头开始采集。
   - 增量：增量采集指从距离文件末尾1M处开始采集（若日志文件小于1M，则等同于全量采集）。

> **注意：**
>
> **全量采集场景**
> - 采集规则生效延时：为确保采集到 Pod 启动过程中的所有关键日志，请配置全量采集策略。全量采集策略能够在采集规则完全生效前，无差别地收集所有日志数据，有效避免日志丢失现象。
>
> **增量采集场景**
> - 采集路径为云硬盘 CBS：采集路径为云硬盘时建议配置增量采集策略，以免 Pod 重建产生重采。
>
> **异常场景**
> - 采集路径为文件存储 CFS：配置全量采集会造成重采，且多个 Pod 可能同时上报相同日志，造成日志服务重复计费，建议日志源不要使用 CFS。

7. 单击**下一步**，选择日志解析方式。
   - 编码模式：支持 **UTF-8** 和 **GBK**。

- 提取模式：支持多种类型的提取模式，详情如下：

解析模式	说明	相关文档
单行全文	一条日志仅包含一行的内容，以换行符 \n 作为一条日志的结束标记，每条日志将被解析为键值为 CONTENT 的一行完全字符串，开启索引后可通过全文检索搜索日志内容。日志时间以采集时间为准。	单行全文格式
多行全文	指一条完整的日志跨占多行，采用首行正则的方式进行匹配，当某行日志匹配上预先设置的正则表达式，就认为是一条日志的开头，而下一个行首出现作为该条日志的结束标识符，也会设置一个默认的键值 CONTENT，日志时间以采集时间为准。支持自动生成正则表达式。	多行全文格式
单行 – 完全正则	指将一条完整日志按正则方式提取多个 key-value 的日志解析模式，您需先输入日志样例，其次输入自定义正则表达式，系统将根据正则表达式里的捕获组提取对应的 key-value。支持自动生成正则表达式。	单行 – 完全正则格式
多行 – 完全正则	适用于日志文本中一条完整的日志数据跨占多行（例如 Java 程序日志），可按正则表达式提取为多个 key-value 键值的日志解析模式，您需先输入日志样例，其次输入自定义正则表达式，系统将根据正则表达式里的捕获组提取对应的 key-value。支持自动生成正则表达式。	多行–完全正则格式
JSON	JSON 格式日志会自动提取首层的 key 作为对应字段名，首层的 value 作为对应的字段值，以该方式将整条日志进行结构化处理，每条完整的日志以换行符 \n 为结束标识符。	JSON 格式
分隔符	指一条日志数据可以根据指定的分隔符将整条日志进行结构化处理，每条完整的日志以换行符 \n 为结束标识符。日志服务在进行分隔符格式日志处理时，您需要为每个分开的字段定义唯一的 key，无效字段即无需采集的字段可填空，不支持所有字段均为空。	分隔符格式
组合解析	当您的日志结构太过复杂，涉及多种解析模式，单种解析模式（如 Nginx 模式、完整正则模式、JSON 模式等）无法满足日志解析需求时，您可以使用 Loglistener 组合解析格式解析日志，此模式支持用户在控制台输入代码（json 格式）用来定义日志解析的流水线逻辑。您可添加一个或多个 Loglistener 插件处理配置，Loglistener 会根据处理配置顺序逐一执行。	组合解析格式

- 过滤器：LogListener 仅采集符合过滤器规则的日志，Key 支持完全匹配，过滤规则支持正则匹配，如仅采集 `ErrorCode = 404` 的日志。您可以根据需求开启过滤器并配置规则。

> **说明：**
> 一个日志主题目前仅支持一个采集配置，请保证选用该日志主题的所有容器的日志都可以接受采用所选的日志解析方式。若在同一日志主题下新建了不同的采集配置，旧的采集配置会被覆盖。

8. 单击**完成**，完成投递到 CLS 的容器日志采集规则创建。

更新日志规则

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击**采集规则**右侧的**编辑**。如下图所示：

4. 根据需求更新相应配置，单击**完成**。

> **注意：**
> 日志集和日志主题不可更新。

相关文档

您不仅可以使用控制台配置日志采集，还可以通过自定义资源（CustomResourceDefinitions，CRD）的方式配置日志采集。详情见 使用 CRD 配置日志采集。

使用 CRD 配置日志采集
使用 CRD 进行日志采集配置指导

最近更新时间：2025-02-21 10:23:23

操作场景

您不仅可以 使用控制台配置日志采集，还可通过自定义资源（CustomResourceDefinitions，CRD）的方式配置日志采集。CRD 支持采集容器标准输出、容器文件和主机文件，支持多种日志采集格式。支持投递到 CLS 和 CKafka 等不同消费端。

前提条件

已在容器服务控制台的 运维功能管理 中开启日志采集，操作详情请参见 开启日志采集 。

采集器限制

Kafka采集器限制

项目	能力与限制
采集规则	日志规则最大能够同时采集监听约256个文件，如果当前采集超出此限制，请 提交工单 。
单条日志大小	• 标准输出模式下单行最大长度为16KB，超过此长度会导致日志截断。 • 容器文件和节点文件模式下： 　○ 日志单行最大长度小于256KB时，单日志规则最大能够同时监听采集约80个文件。 　○ 日志单行最大长度大于256KB且小于2MB时，单日志规则最大能够同时监听采集约10个文件。 　○ 日志单行最大长度不能超过2MB。如果当前采集超出此限制，请 提交工单 。
标签等采集方式	如果使用标签等方式批量采集文件，需要确保当前日志规则下所有被采集容器的目标路径均存在，否则当前日志规则会出现采集失败的情况。
处理能力	• Kafka 采集日志开启 Metadata 与关闭时采集能力基本持平，同一个日志规则的所有被采集文件共享吞吐能力。 • 单日志规则单文件情况下，每行日志大小为25KB时，采集器单行文本提取模式处理能力约为40MB/s，采集器 JSON 提取模式处理能力约为4.5MB/s。

CRD 介绍

结构总览

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig                      ## 默认值
metadata:
  name: test                         ## CRD资源名，在集群内唯一
spec:
  clsDetail:                         ## 投递到CLS的配置
    ...
  inputDetail:                       ## 采集数据源配置
    ...
  kafkaDetail:                       ## 投递到 ckafka 或者自建kafka配置
    ...
status:                              ## CRD资源状态
  status: ""
  code: ""                           ## 调用接口出错时，接口返回的错误码
  reason: ""                         ## 出错原因
```

clsDetail 字段说明

> ⚠ 注意：

> topic 指定后不允许修改。

```
  clsDetail:
    ## 自动创建日志主题，需要同时指定日志集和主题的name
    logsetName: test                    ## CLS日志集的name，若无该name的日志集，会自动创建，若有，会在该日志集下创建
日志主题
    topicName: test                     ## CLS日志主题的name，若无该name的日志主题，会自动创建

    # 选择已有日志集日志主题， 如果指定了日志集未指定日志主题，则会自动创建一个日志主题
    logsetId: xxxxxx-xx-xx-xx-xxxxxxxx  ## CLS日志集的ID，日志集需要在CLS中提前创建
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx   ## CLS日志主题的ID，日志主题需要在CLS中提前创建，且没有被其它采集配置占用

    logType: json_log                   ## 日志采集格式，json_log代表 json 格式，delimiter_log代表分隔符格式，
minimalist_log代表单行全文格式，multiline_log代表多行全文格式，fullregex_log代表完全正则格式，
multiline_fullregex_log代表多行正则格式。默认为minimalist_log
    logFormat: xxx                      ## 日志格式化方式
    period: 30                          ## 生命周期，单位天，可取值范围1~3600。取值为3640时代表永久保存
    partitionCount:                     ## Integer 类型，日志主题分区个数。默认创建1个，最大支持创建10个分区。
    tags:                               ## 标签描述列表，通过指定该参数可以同时绑定标签到相应的日志主题。最大支持9个标
签键值对，同一个资源只能绑定到同一个标签键下。
    - key: xxx                          ## 标签key
      value: xxx                        ## 标签value
    autoSplit: false                    ## boolean 类型，是否开启自动分裂，默认值为true
    maxSplitPartitions:
    storageType: hot                    ## 日志主题的存储类型，可选值 hot（标准存储），cold（低频存储）；默认为hot。
    excludePaths:                       ## 采集黑名单路径列表
    - excludeType: File                        ## 类型，选填File或Path
      value: /xx/xx/xx/xx.log           ## type 对应的值
    indexs:                             ## 创建 topic 时可自定义索引方式和字段
    - indexName:                        ## 需要配置键值或者元字段索引的字段，元字段Key无需额外添加__TAG__.前缀，与上
传日志时对应的字段Key一致即可，腾讯云控制台展示时将自动添加__TAG__.前缀
      indexType:                        ## 字段类型，目前支持的类型有：long、text、double
      tokenizer:                        ## 字段的分词符，其中的每个字符代表一个分词符；仅支持英文符号及\n\t\r；long及
double类型字段需为空；text类型字段推荐使用 @&?|#()='",;:<>[]{}/ \n\t\r\ 作为分词符；
      sqlFlag:                          ## boolean 字段是否开启分析功能
      containZH:                        ## boolean 是否包含中文
    region: ap-xxx                      ## topic 所在地域，用于跨地域投递
    userDefineRule: xxxxxx              ## 用户自定义采集规则，Json格式序列化的字符串
    extractRule: {}                     ## 提取、过滤规则。 如果设置了ExtractRule，则必须设置LogType
```

inputDetail 字段说明

```
  inputDetail:
    type: container_stdout    ## 采集日志的类型，包括container_stdout（容器标准输出）、container_file（容器文件）、
host_file（主机文件）

    containerStdout:         ## 容器标准输出
      namespace: default     ## 采集容器的kubernetes命名空间。支持多个命名空间，如果有多个命名空间使用","分隔，如：
default,namespace。 如果不指定，代表所有命名空间。注意：与 excludeNamespace 不能同时指定
      excludeNamespace: nm1,nm2    ## 排除采集容器的kubernetes命名空间。支持多个命名空间，如果有多个命名空间使用","分
隔，如：nm1,nm2。 如果不指定，代表所有命名空间。 注意：与 namespace 不能同时指定
      nsLabelSelector: environment in (production),tier in (frontend) ## 根据命名空间label 筛选符合的
namespace
      allContainers: false        ## 是否采集指定命名空间中的所有容器的标准输出。注意:allContainers=true 时不能同时指
定 workload，includeLabels 和 excludeLabels
      container: xxx              ## 采集日志的容器名，为空时，代表采集所有符合容器的日志名。
      excludeLabels:  ## 采集不包含指定label的Pod，与workload，namespace 和 excludeNamespace 不能同时指定
        key2: value2  ## 支持匹配同一个key下多个value值的pod，例填写environment = production,qa表示当key为
environment，value值为production或qa时，均会被排除，注意输入多个value值时请使用逗号隔开。如果同时指定了
```

```
includeLabels，则匹配与 includeLabels 交集的pod

      includeLabels:  ## 采集包含指定label的Pod，与workload，namespace 和 excludeNamespace 不能同时指定
        key: value1   ## 收集规则收集的日志会带上metadata，并上报到消费端。支持匹配同一个key下多个value值的pod，例填写environment = production,qa表示当key为environment，value值为production或qa时，均会被匹配，注意输入多个value值时请使用逗号隔开。 如果同时指定了 excludeLabels，则匹配与 excludeLabels 交集的pod

      metadataLabels:           ## 指定具体哪些pod label被当作元数据采集，如果不指定，则采集所有pod label为元数据
      - label1
      customLabels:             ## 用户自定义metadata
        label: l1

      workloads:
      - container: xxx     ## 要采集的容器名，如果不指定，代表workload Pod中的所有容器
        kind: deployment   ## workload类型，支持deployment、daemonset、statefulset、job、cronjob
        name: sample-app   ## workload的名字
        namespace: prod    ## workload的命名空间

    containerFile:  ## 容器内文件
      namespace: default      ## 采集容器的kubernetes命名空间，必须指定一个命名空间
      excludeNamespace: nm1,nm2    ## 排除采集容器的kubernetes命名空间。支持多个命名空间，如果有多个命名空间使用","分隔，如：nm1,nm2。 如果不指定，代表所有命名空间。 注意：与 namespace 不能同时指定
      nsLabelSelector: environment in (production),tier in (frontend) ## 根据命名空间label 筛选符合的namespace
      container: xxx          ## 采集日志的容器名，为 * 时，代表采集所有符合容器的日志名
      logPath: /var/logs      ## 日志文件夹，不支持通配符
      filePattern: app_*.log  ## 日志文件名，支持通配符 * 和 ? ，* 表示匹配多个任意字符，? 表示匹配单个任意字符
      customLabels:   ## 用户自定义metadata
        key: value
      excludeLabels:  ## 采集不包含指定label的Pod，与workload不能同时指定
        key2: value2  ## 支持匹配同一个key下多个value值的pod，例填写environment = production,qa表示当key为environment，value值为production或qa时，均会被排除，注意输入多个value值时请使用逗号隔开。如果同时指定了includeLabels，则匹配与 includeLabels 交集的pod

      includeLabels:  ## 采集包含指定label的Pod，与workload不能同时指定
        key: value1   ## 收集规则收集的日志会带上metadata，并上报到消费端。支持匹配同一个key下多个value值的pod，例填写environment = production,qa表示当key为environment，value值为production或qa时，均会被匹配，注意输入多个value值时请使用逗号隔开。 如果同时指定了 excludeLabels，则匹配与 excludeLabels 交集的pod
      metadataLabels:         ## 指定具体哪些pod label被当作元数据采集，如果不指定，则采集所有pod label为元数据
      - label1                ## pod label
      workload:
        container: xxx        ## 要采集的容器器名，如果不指定，代表workload Pod中的所有容器
        name: sample-app      ## workload的名字

    hostFile:                 ## 节点文件路径
      filePattern: '*.log'    ## 日志文件名，支持通配符 * 和 ? ，* 表示匹配多个任意字符，? 表示匹配单个任意字符
      logPath: /tmp/logs      ## 日志文件夹，不支持通配符
      customLabels:           ## 用户自定义metadata
        label1: v1
```

extractRule 对象说明

名称	类型	必填项	描述
timeKey	String	否	时间字段的 key 名字，time_key 和 time_format 必须成对出现。
timeFormat	String	否	时间字段的格式，参考 C 语言的 strftime 函数对于时间的格式说明输出参数。
delimiter	String	否	分隔符类型日志的分隔符，只有 log_type 为 delimiter_log 时有效。
logRegex	String	否	整条日志匹配规则，只有 log_type 为 fullregex_log 时有效。

beginningRegex	String	否	行首匹配规则，只有 log_type 为 multiline_log 或 fullregex_log 时有效。
unMatchUpload	String	否	解析失败日志是否上传，true 表示上传，false 表示不上传。
unMatchedKey	String	否	失败日志的 key。
backtracking	String	否	增量采集模式下的回溯数据量，默认-1（全量采集），0 表示增量。
keys	Array of String	否	取的每个字段的 key 名字，为空的 key 代表丢弃这个字段，只有 log_type 为 delimiter_log 时有效，json_log 的日志使用 json 本身的 key。
filterKeys	Array of String	否	需要过滤日志的 key，与 FilterRegex 按下标进行对应。
filterRegex	Array of String	否	需要过滤日志的 key 对应的 regex，与 FilterKeys 按下标进行对应。
isGBK	String	否	是否为 Gbk 编码。0: 否，1: 是。 **注意**：此字段可能返回 null，表示取不到有效值。
jsonStandard	String	否	是否为标准 json。0: 否，1: 是。 **注意**：此字段可能返回 null，表示取不到有效值。

kafkaDetail 字段说明

```
kafkaDetail:
  brokers: x.x.x.x:p     ## 必填，broker地址，一般是域名:端口，多个地址以“,”分隔
  topic: test
  kafkaType: CKafka      ## kafka 类型, CKafka - ckafka, SelfBuildKafka - 自建kafka
  instanceId: xxxx       ## 当 kafkaType = CKafka, 设置ckafka实例 id
  logType: minimalist_log ## kafka 日志解析类型，"minimalist_log" 或 "" 单行全文，"multiline_log" 多行全文，"json" json 格式
  timestampFormat: xxx    ## 时间戳的格式，默认是double
  timestampKey: xxx       ## 时间戳的key值，默认是"@timestamp"
metadata:
  formatType: default    ## metadata 格式。 "default" 默认格式（与 eks kafka 采集器相同），"filebeat" filebeat 格式，"fluent-bit" fluent-bit 格式
  messageKey:             ## 支持指定一个Key，将日志投递到指定分区。默认不开启，日期随机投放；开启后带有同样Key的日志，将投递到相同的分区里。支持选择Pod字段作为Key，以Pod name为例，请选择Field>metadata.name
    value: Field         ## 必填，topicID
    valueFrom:
      fieldRef:
        fieldPath: metadata.name ##  当key为Field时可选 metadata.name,metadata.namespace,spec.nodeName,spec.serviceAccountName
```

status 字段说明

status	说明
状态为空	初始状态
Synced	采集配置处理成功
Stale	采集配置处理失败

CRD 示例

配置容器标准输出 CRD 示例

所有容器

指定命名空间

```yaml
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: "test"
spec:
  clsDetail:
    .......
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx
  inputDetail:
    containerStdout:
      allContainers: true
      namespace: default,kube-public
    type: container_stdout
```

排除命名空间

```yaml
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: "test"
spec:
  clsDetail:
    ........
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx
  inputDetail:
    containerStdout:
      allContainers: true
      excludeNamespace: kube-system,kube-node-lease
    type: container_stdout
```

指定工作负载

```yaml
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: "test"
spec:
  clsDetail:
    ......
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx
  inputDetail:
    containerStdout:
      allContainers: false
      workloads:
      - container: prod
        kind: deployment
        name: sample-app
        namespace: kube-system
    type: container_stdout
```


指定 Pod Labels

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: test
spec:
  clsDetail:
    ......
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx
  inputDetail:
    containerStdout:
      container: prod
      excludeLabels:
        key2: v2
      includeLabels:
        key1: v1
      namespace: default,kube-system
    type: container_stdout
```

配置容器文件路径 CRD 示例

指定工作负载

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: test
spec:
  clsDetail:
    .......
    topicId: xxxx-xx-xx-xx-xxxx
  inputDetail:
    containerFile:
      container: prod
      filePattern: '*.log'
      logPath: /tmp/logs
      namespace: kube-system
      workload:
        kind: deployment
        name: sample-app
    type: container_file
```

指定 Pod Labels

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: test
spec:
  clsDetail:
```

```
      .......
      topicId: xxxx-xx-xx-xx-xxxx
    inputDetail:
      containerFile:
        container: prod
        filePattern: '*.log'
        includeLabels:
          key1: v1
        excludeLabels:
          key2: v2
        logPath: /tmp/logs
        namespace: default,kube-public
      type: container_file
```

配置节点文件路径 CRD 示例

```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  creationTimestamp: "2022-03-13T12:48:49Z"
  generation: 4
  name: test
  resourceVersion: "11729531"
  selfLink: /apis/cls.cloud.tencent.com/v1/logconfigs/test
  uid: 233f4b72-cfef-4a43-abb8-e4d033185097
spec:
  clsDetail:
    .......
    topicId: xxxx-xx-xx-xx-xxxx
  inputDetail:
    hostFile:
      customLabels:
        testmetadata: v1
      filePattern: '*.log'
      logPath: /var/logs
    type: host_file
```

LogConfig json 格式说明

最近更新时间：2024-10-12 18:17:41

LogConfig CR 字段说明

```
{
  "kind": "cls.cloud.tencent.com/v1",
  "apiVersion": "LogConfig",
  "metadata": {
    "name": "configName"  // 采集规则名
  },
  "spec": {
    "inputDetail": {...}, // 日志采集源定义
    "clsDetail": {...},   // 采集到CLS的配置
    "kafkaDetail": {...}  // 采集到 Kafka 的配置
  }
}
```

spec 内容说明

采集规则 spec 部分主要定义了日志的采集源以及日志输出端的配置信息。

> **注意：**
> clsDetail 和 kafkaDetail 在同一个采集规则中只能配置一种。

字段	是否必填	说明
inputDetail	是	定义采集源。
clsDetail	否	CLS 采集器配置，该配置信息将由日志组件 cls-provisioner 同步到 CLS 侧。
kafkaDetail	否	Kafka 采集器配置，定义日志采集格式等信息。

inputDetail 说明

```
{
  "type": "host_file",
  "containerStdout": {},
  "containerFile": {},
  "hostFile": {}
}
```

字段	是否必填	类型	说明
type	是	String	采集类型，取值如下： • container_file：容器内文件采集，该类型对应 `containerFile` 配置项。 • container_stdout：采集容器标准输出。 • host_file：采集节点日志，超级节点不支持该配置。
containerStdout	否	Object	容器标准输出采集配置。配置采集容器标准输出时，需指定采集类型为 container_stdout。
containerFile	否	Object	容器内路径采集。配置采集容器标准输出时，需指定采集类型为 container_file。
hostFile	否	Object	节点路径采集。配置采集容器标准输出时，需指定采集类型为 container_file。

containerStdout 对象结构说明

```
{
  "includeLabels": {
    "label1": "value1"
  },
  "excludeLabels": {
    "label": "value"
  },
  "metadataLabels": [
    "label1"
  ],
  "metadataContainer": [
    "namespace",
    "pod_name",
    "pod_ip",
    "pod_uid",
    "container_id",
    "container_name",
    "image_name",
    "cluster_id"
  ],
  "container": "container",
  "workloads": [
    {
      "container": "kubernetes-proxy",
      "kind": "deployment",
      "name": "kubernetes-proxy",
      "namespace": "default"
    },
    {
      "container": "testlog11",
      "kind": "deployment",
      "name": "testlog1",
      "namespace": "default"
    }
  ],
  "customLabels": {
    "xx": "xx"
  }
}
```

字段	是否必填	类型	说明
includeLabels	否	Map	采集包含指定 label:value 的 pod，不能与 workloads 同时配置。
excludeLabels	否	Map	排除包含指定 label:value 的 pod，不能与 workloads 同时配置。
metadataLabels	否	Array	定义将 pod 的哪些 label 作为元数据上报。不指定则全部上报。
metadataContainer	否	Array	定义将 pod、container 的哪些信息作为元数据上报。不指定则全部上报。
customLabels	否	Map	用户自定义元数据。
container	否	String	指定采集哪些容器的日志。不指定或配置 `*`，则采集命中 pod 的所有容器的标准输出。不能与 workloads 同时配置。
workloads	否	Array	指定需要采集哪些 workload 的标准输出，同时指定采集的容器名称。

workload 对象说明

字段	是否必填	类型	说明

container	否	String	需要采集的容器名称。不指定，则采集所有容器标准输出。
kind	是	String	workload 类型。
name	是	String	workload 名称。
namespace	是	String	workload 所在 namespace。

containerFile 对象结构说明

```
{
  "logPath": "/var/logs",
  "filePattern": "*.log",
  "filePaths": [
    {
      "path": "/var/logs",
      "file": "*.log"
    }
  ],
  "customLabels": {
    "key": "value1"
  },
  "namespace": "default",
  "nsLabelSelector": "",
  "excludeNamespace": "xxx,xx",
  "includeLabels": {
    "label": "value"
  },
  "excludeLabels": {
    "exLabel": "exvalue"
  },
  "metadataLabels": [
    "xx"
  ],
  "metadataContainer": [
    "namespace",
    "pod_name",
    "pod_ip",
    "pod_uid",
    "container_id",
    "container_name",
    "image_name",
    "cluster_id"
  ],
  "container": "xxx",
  "workload": {
    "name": "xxx",
    "kind": "deployment"
  }
}
```

字段	是否必填	类型	说明
logPath	否	String	采集路径。不可与 filePaths 同时配置。
filePattern	否	String	预期采集文件匹配模式，不可与 filePaths 同时配置。通配符支持 `*` 和 `?` 。
filePaths	是	Array	配置预期采集容器内的路径及文件匹配模式，可配置多个采集路径。
includeLabels	否	Map	采集包含指定 label:value 的 pod，不能与 workload 同时配置。

excludeLabels	否	Map	排除包含指定 label:value 的 pod，不能与 workload 同时配置。
metadataLabels	否	Array	定义将 pod 的哪些 label 作为元数据上报。不指定则全部上报。
metadataContainer	否	Array	定义将 pod、container 的哪些信息作为元数据上报。不指定则全部上报。
customLabels	否	Map	用户自定义元数据。
container	否	String	指定采集哪些容器的日志。
workload	否	Object	指定需要采集哪些 workload 的标准输出，同时指定采集的容器名称。

workload 字段说明

字段	是否必填	类型	说明
kind	是	String	workload 类型。
name	是	String	workload 名称。

filePath 对象字段说明

字段	是否必填	类型	说明
path	是	String	预期采集的日志所在路径。
file	是	String	预期采集的日志文件名称匹配模式，通配符支持 `*` 和 `?` 。

hostFile 对象结构说明

```
{
  "logPath": "/var/logs",
  "filePattern": "*.log",
  "filePaths": [
    {
      "path": "/var/log",
      "file": "*.log"
    }
  ],
  "customLabels": {
    "key": "value"
  }
}
```

字段	是否必填	类型	说明
logPath	否	String	采集路径。不可与 filePaths 同时配置。
filePattern	否	String	预期采集文件匹配模式，不可与 filePaths 同时配置。通配符支持 `*` 和 `?` 。
filePaths	否	Array	配置预期采集容器内的路径及文件匹配模式，可配置多个采集路径。
customLabels	否	Map	用户自定义元数据。

clsDetail 对象结构说明

```
{
  "topicId": "",
  "topicName": "",
  "logsetId": "",
  "logsetName": "",
```

```
  "logFormat": "",
  "logType": "",
  "hotPeriod": 7,
  "period": 30,
  "partitionCount": 1,
  "tags": [
    {
      "key": "key",
      "value": "value"
    }
  ],
  "autoSplit": false,
  "maxSplitPartitions": 50,
  "storageType": "hot",
  "extractRule": {...},
  "excludePaths": [
    {
      "excludeType": "File",
      "value": "/var/log/1.log"
    },
    {
      "excludeType": "Path",
      "value": "/var/log1/"
    }
  ],
  "fullTextIndex": {...},
  "indexs": [
    {...}
  ],
  "indexStatus": "off",
  "region": "ap-xxx",
  "advancedConfig": {...}
}
```

字段名	是否必填	类型	说明
topicId	否	String	日志主题 ID。
topicName	否	String	日志主题名称。当 topicId 为空时，指定日志主题名称，可根据日志主题名称自动拉取日志主题 ID，或者自动创建日志主题。
logsetId	否	String	日志集 ID。
logsetName	否	String	日志集名称。当 logsetId 为空时，指定日志集名称，可根据日志集名称自动拉取日志集 ID，或者自动创建日志集。
region	否	String	日志主题所在 region。默认为集群所在 region。
logType	否	String	日志类型。可选值：json_log（json 格式）、delimiter_log（分隔符提取）、minimalist_log（单行全文格式）、multiline_log（多行全文格式）、fullregex_log（完全正则格式）。默认为 minimalist_log。
logFormat	否	String	日志格式化。
storageType	否	String	日志主题存储类型（仅在自动创建 topic 时生效）。可选值：hot（标准存储）、cold（低频存储）。默认 hot。
hotPeriod	否	Number	日志沉降开关（仅在自动创建 topic 时生效）。0：关闭日志沉降，非 0：开启日志沉降后标准存储的天数。HotPeriod 需要大于等于 7，且小于 Period。仅在 StorageType 为 hot 时生效。
period	否	Number	日志存储周期（单位天）（仅在自动创建 topic 时生效）。标准存储取值范围 1~3600，低频存储取值范围 7~3600 天。取值为 3640 时代表永久保存。默认 30 天。

partitionCount	否	Number	日志主题分区个数（仅在自动创建 topic 时生效）。默认创建 1 个，最大支持创建 10 个分区。
tags	否	Array	标签列表（仅在自动创建 topic 时生效）。
autoSplit	否	Bool	是否开启自动分裂，“false” 或 "true"（仅在自动创建 topic 时生效）。
maxSplitPartitions	否	Number	开启自动分裂后，每个日志主题能够容许的最大分区数，默认为 50 个（仅在自动创建 topic 时生效）。
extractRule	否	Object	采集规则相关配置。
excludePaths	否	Array	采集黑路径配置。
fullTextIndex	否	Object	全文索引配置（仅在自动创建 topic 时生效）。
indexs	否	Array	索引配置（仅在自动创建 topic 时生效）。
indexStatus	否	String	索引状态（仅在自动创建 topic 时生效）。
autoIndex	否	String	是否开启自动索引（仅在自动创建 topic 时生效）。
advancedConfig	否	Object	高级采集配置。

tag 对象结构说明

```
    {
      "key": "key",
      "value": "value"
    }
```

字段	类型	是否必填	说明
key	String	是	标签名称。
value	String	是	标签值。

extractRule 对象结构说明

```
{
  "timeKey": "xxx",
  "timeFormat": "xxx",
  "delimiter": "xxx",
  "logRegex": "xxx",
  "beginningRegex": "xxx",
  "keys": [
    "xx"
  ],
  "filterKeys": [
    "xx"
  ],
  "filterRegex": [
    "xx"
  ],
  "unMatchUpload": "true",
  "unMatchedKey": "parseFailed",
  "backtracking": "-1",
  "isGBK": "true",
  "jsonStandard": "true",
  "advancedFilters": [
    {
      "key": "level",
```

```
        "rule": 0,
        "value": "info"
      }
    ]
  }
```

字段名	是否必填	类型	说明
timeKey	否	String	时间字段的 key 名字，timeKey 和 timeFormat 必须成对出现。
timeFormat	否	String	时间字段的格式，参考 C 语言的 strftime 函数对于时间的格式说明输出参数。
delimiter	否	String	分隔符类型日志的分隔符，只有 logType 为 delimiter_log 时有效。
logRegex	否	String	整条日志匹配规则，只有 logType 为 fullregex_log 时有效。
beginningRegex	否	String	行首匹配规则，只有 logType 为 multiline_log 或 fullregex_log 时有效。
keys	否	Array	取的每个字段的 key 名字，为空的 key 代表丢弃这个字段，只有 logType 为 delimiter_log 时有效，json_log 的日志使用 json 本身的 key。
filterKeys	否	Array	需要过滤日志的 key，与 filterRegex 按下标进行对应。
filterRegex	否	Array	需要过滤日志的 key 对应的 regex，与 filterKeys 按下标进行对应。
unMatchUpload	否	String	解析失败日志是否上传，true 表示上传，false 表示不上传。
unMatchedKey	否	String	失败日志的 key。
backtracking	否	String	增量采集模式下的回溯数据量，默认 -1（全量采集），0 表示增量。
isGBK	否	String	是否为 GBK 编码。可选值：`"false"`：否；`"true"`：是。注意：此字段可能返回 null，表示取不到有效值。
jsonStandard	否	String	是否为标准 JSON。可选值：`"false"`：否；`"true"`：是。注意：此字段可能返回 null，表示取不到有效值。
advancedFilters	否	Array	高级过滤规则，该字段仅适用于 v1.1.15 及以上的采集组件版本，低于 v1.1.15 请使用 filterKeys 与 filterRegex。

advancedFilters 对象结构说明

```
    {
        "key": "level",  // 需要过滤的key
        "rule": 0,       // 过滤规则 0(等于), 1(字段存在), 2(字段不存在), 3(不等于)
        "value": "info" // 需要过滤的值value。 其中当rule为1或2时, value无需指定
    }
```

excludePath 对象结构说明

```
    {
      "excludeType": "File",
      "value": "/var/log/1.log"
    }
```

字段	类型	是否必填	说明
excludeType	String	是	排除类型。取值 File、Path。
excludeType	String	是	排除的日志路径或者路径。

fullTextIndex 对象结构说明

```
{
        "caseSensitive": false,
        "containZH": false,
        "status": "on",
        "tokenizer": "@&()='\",;:<>[]{}\/ \n\t\r"
    }
```

字段名	类型	是否必填	说明
caseSensitive	bool	否	是否大小写敏感。
containZH	bool	否	是否包含中文。
status	String	否	全文索引开关。如果不设置，则会默认打开全文索引；如果设置为 on 打开，不设置其他参数，则 caseSensitive 和 tokenizer 都会是默认值；如果设置为 off 关闭，则不打开全文索引。
tokenizer	String	否	全文索引的分词符。如果全文索引开关打开，则必须设置 tokenizer。可以设置为 "@&()='",;:<>[]{}/ \n\t\r"，为默认设置。

index 对象结构说明

```
    {
      "indexName": "xxx",
      "indexType": "text",
      "tokenizer": "@&()='\",;:<>[]{}/ \n\t\r",
      "sqlFlag": true,
      "containZH": false
    }
```

字段名	类型	是否必填	说明
indexName	String	是	索引名称。
indexType	String	是	索引类型。可选值：long、text、double。
tokenizer	String	否	字段分词符。
sqlFlag	bool	否	字段是否开启分析功能。
containZH	bool	否	是否包含中文。

advancedConfig 对象结构说明

```
{
"ClsAgentMaxDepth": 1,
"ClsAgentFileTimeout": 60,
"ClsAgentParseFailMerge": false,
}
```

字段名	类型	是否必填	说明
ClsAgentMaxDepth	Number	否	当前配置采集时下钻的深度。
ClsAgentFileTimeout	Number	否	当前配置在采集文件时，当文件在指定时间未写入日志时，采集器释放文件句柄。
ClsAgentParseFailMerge	bool	否	采集器解析文件失败后是否合并日志上报。

kafkaDetail 对象结构说明

```
{
  "brokers": "127.0.0.1:9092",
  "topic": "test_log",
  "timestampKey": "@timestamp",
  "timestampFormat": "double",
  "extractRule": {
    "beginningRegex": "xxxx"
  },
  "logType": "minimalist_log",
  "metadata": {
    "formatType": "json"
  }
}
```

字段名	类型	是否必填	说明
brokers	String	是	Kafka brokers。
topic	String	是	Kafka topic。
timestampKey	String	否	时间戳的 key，默认 @timestamp。
timestampFormat	String	否	时间戳格式。可选值：double、iso8601。默认 double。
extractRule.beginningRegex	String	否	行首匹配规则。logType 为 multiline_log 时，该项需要配置。
logType	String	否	日志提取模式。默认单行全文。可选值：minimalist_log（单行全文）、multiline_log（多行全文）、json（json 格式）。
metadata.formatType	String	否	元数据格式。可选值：fluent-bit（fluent-bit 原生格式）、fluent-bit-string（字符串类型）。

日志采集源配置示例

容器标准输出

采集 default 命名空间中的所有容器的标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com/v1",
    "kind": "LogConfig",
    "metadata": {
        "name": "test"
    },
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": true,
                "namespace": "default"
            },
            "type": "container_stdout"
        },
        ...
    }
}
```

采集 production 命名空间中属于 ingress-gateway deployment 的 pod 中的所有容器的标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com\/v1",
    "kind": "LogConfig",
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": false,
                "workloads": [
                    {
                        "kind": "deployment",
                        "name": "ingress-gateway",
                        "namespace": "production"
                    }
                ]
            },
            "type": "container_stdout"
        },
        ...
    }
}
```

采集 production 命名空间中 pod 标签中包含 k8s-app=nginx 的 pod 中的所有容器的标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com\/v1",
    "kind": "LogConfig",
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": false,
                "includeLabels": {
                    "k8s-app": "nginx"
                },
                "namespace": "production"
            },
            "type": "container_stdout"
        },
        ...
    }
}
```

采集 production 命名空间中属于 ingress-gateway 工作负载的 pod 中，名称为 ingress 的容器的标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com\/v1",
    "kind": "LogConfig",
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": false,
                "includeLabels": {
                    "k8s-app": "nginx"
                },
                "namespace": "production"
            },
            "type": "container_stdout"
        },
        ...
```

```
    }
}
```

容器文件采集

采集 production 命名空间中属于 ingress-gateway deployment 的 pod 中的 nginx 容器中 /data/nginx/log/ 路径下名为 access.log 的文件：

```
{
    "apiVersion": "cls.cloud.tencent.com/v1",
    "kind": "LogConfig",
    "metadata": {
        "name": "test"
    },
    "spec": {
        "inputDetail": {
            "containerFile": {
                "container": "nginx",
                "filePaths": [
                    {
                        "file": "access.log",
                        "path": "/data/nginx/log"
                    }
                ],
                "namespace": "production",
                "workload": {
                    "kind": "deployment",
                    "name": "ingress-gateway"
                }
            },
            "type": "container_file"
        },
      ...
    }
}
```

采集 production 命名空间中 pod 标签包含 k8s-app=ingress-gateway 的 pod 中的 nginx 容器中 /data/nginx/log/ 路径下名为 access.log 的文件：

```
{
    "apiVersion": "cls.cloud.tencent.com/v1",
    "kind": "LogConfig",
    "spec": {
        "inputDetail": {
            "containerFile": {
                "container": "nginx",
                "filePaths": [
                    {
                        "file": "access.log",
                        "path": "/data/nginx/log"
                    }
                ],
                "includeLabels": {
                    "k8s-app": "ingress-gateway"
                },
                "namespace": "production"
            },
            "type": "container_file"
        }
    }
```

```
}
```

主机文件采集

采集主机 /data/ 路径下所有 .log 文件：

```
{
    "apiVersion": "cls.cloud.tencent.com/v1",
    "kind": "LogConfig",
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": false,
                "includeLabels": {
                    "k8s-app": "nginx"
                },
                "namespace": "production"
            },
            "type": "container_stdout"
        },
      ...
    }
}
```

完整示例

投递到 CLS 的日志采集规则，采集 default 命名空间下 deployment/testlog1 的标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com\/v1",
    "kind": "LogConfig",
    "metadata": {
        "name": "123"
    },
    "spec": {
        "clsDetail": {
            "extractRule": {
                "backtracking": "0",
                "isGBK": "false",
                "jsonStandard": "false",
                "unMatchUpload": "false"
            },
            "indexs": [
                {
                    "indexName": "namespace"
                },
                {
                    "indexName": "pod_name"
                },
                {
                    "indexName": "container_name"
                }
            ],
            "logFormat": "default",
            "logType": "minimalist_log",
            "maxSplitPartitions": 0,
            "region": "ap-chengdu",
            "storageType": "",
            "topicId": "c26b66bd-617e-4923-bea0-test"
        },
```

```
        "inputDetail": {
            "containerStdout": {
                "metadataContainer": [
                    "namespace",
                    "pod_name",
                    "pod_ip",
                    "pod_uid",
                    "container_id",
                    "container_name",
                    "image_name",
                    "cluster_id"
                ],
                "nsLabelSelector": "",
                "workloads": [
                    {
                        "kind": "deployment",
                        "name": "testlog1",
                        "namespace": "default"
                    }
                ]
            },
            "type": "container_stdout"
        }
    }
}
```

投递到 Kafka 的采集规则，采集 default 命名空间下的所有 pod 标准输出：

```
{
    "apiVersion": "cls.cloud.tencent.com\/v1",
    "kind": "LogConfig",
    "metadata": {
        "name": "321"
    },
    "spec": {
        "inputDetail": {
            "containerStdout": {
                "allContainers": true,
                "namespace": "default",
                "nsLabelSelector": ""
            },
            "type": "container_stdout"
        },
        "kafkaDetail": {
            "brokers": "127.0.0.1:9092",
            "extractRule": {},
            "instanceId": "",
            "kafkaType": "SelfBuildKafka",
            "logType": "minimalist_log",
            "messageKey": {
                "value": "",
                "valueFrom": {
                    "fieldRef": {
                        "fieldPath": ""
                    }
                }
            },
            "metadata": {},
            "timestampFormat": "double",
            "timestampKey": "",
```

```
            "topic": "test"
        }
    }
}
```

日志组件版本升级

最近更新时间：2023-05-17 15:40:42

操作场景

容器服务运维中心提供日志组件版本升级的功能，若您已开启日志采集，腾讯云容器服务当前支持您在容器服务控制台的**运维功能管理**中，查看当前组件版本和进行组件版本的手动升级操作。

升级须知

- 升级属于**不可逆**操作。
- 仅支持向上升级容器服务提供的组件版本，默认升级至当前最新版本。
- 升级时，控制台将自动升级配套的 loglistener 版本和 provisioner 版本，并且将自动更新用户集群内的 CRD 资源，以便获得最新的日志功能。
- 版本详情请查看 日志组件版本迭代记录 。

操作步骤

1. 登录 容器服务控制台，单击左侧导航栏中**运维功能管理**。
2. 在“运维功能管理”页面中，在集群列表上方的选择地域和集群类型。若您的集群开启了日志采集功能，并且组件为可更新状态，控制台会提示“组件可升级”，如下图所示：

3. 选择您的集群，单击“设置”进入设置功能页面，并在“日志采集”栏，单击**编辑**。
4. 在“日志采集”详情中单击**升级组件**完成日志组件升级。如下图所示：

审计管理
审计日志

最近更新时间：2025-02-17 10:22:33

简介

审计日志是基于 Kubernetes Audit 对 kube-apiserver 产生的可配置策略的 JSON 结构日志的记录存储及检索功能。本功能记录了对 kube-apiserver 的访问事件，按顺序记录每个用户、管理员或系统组件影响集群的活动。

功能优势

审计日志功能提供了一种区别于 metrics 的集群观测维度。开启审计日志后，Kubernetes 可以记录每一次对集群的操作。每一条审计日志是一个 JSON 格式的结构化记录，包括元数据（metadata）、请求内容（requestObject）和响应内容（responseObject）三个部分。其中元数据（包含了请求的上下文信息，例如谁发起的请求、从哪里发起的、访问的 URI 等信息）一定会存在，请求和响应内容是否存在取决于审计级别。通过日志可以了解到以下内容：

- 集群里发生的活动。
- 活动的发生时间及发生对象。
- 活动的触发时间、触发位置及观察点。
- 活动的结果以及后续处理行为。

阅读审计日志

```
{
  "kind":"Event",
  "apiVersion":"audit.k8s.io/v1",
  "level":"RequestResponse",
  "auditID":0a4376d5-307a-4e16-a049-24e017******,
  "stage":"ResponseComplete",
  // 发生了什么
  "requestURI":"/apis/apps/v1/namespaces/default/deployments",
  "verb":"create",
  // 谁发起的
  "user":{
    "username":"admin",
      "uid":"admin",
      "groups":[
        "system:masters",
        "system:authenticated"
      ]
  },
  // 从哪里发起
  "sourceIPs":[
    "10.0.6.68"
  ],
  "userAgent":"kubectl/v1.16.3 (linux/amd64) kubernetes/ald64d8",
  // 发生了什么
  "objectRef":{
    "resource":"deployments",
    "namespace":"default",
    "name":"nginx-deployment",
    "apiGroup":"apps",
    "apiVersion":"v1"
  },
  // 结果是什么
  "responseStatus":{
    "metadata":{
    },
    "code":201
```

```
  },
  // 请求及返回具体信息
  "requestObject":Object{...},
  "responseObject":Object{...},
  // 什么时候开始/结束
  "requestReceivedTimestamp":"2020-04-10T10:47:34.315746Z",
  "stageTimestamp":"2020-04-10T10:47:34.328942Z",
  // 请求被接收/拒绝的原因是什么
  "annotations":{
    "authorization.k8s.io/decision":"allow",
    "authorization.k8s.io/reason":""
  }
}
```

TKE 审计日志策略

审计级别（level）

和一般日志不同，Kubernetes 审计日志的级别类似于 verbose 配置，用来标示记录信息的详细程度。一共有4个级别，可参考以下表格内容：

参数	说明
None	不记录。
Metadata	记录请求的元数据（例如用户、时间、资源、操作等），不包括请求和响应的消息体。
Request	除了元数据外，还包括请求消息体，不包括响应消息体。
RequestResponse	记录所有信息，包括元数据以及请求、响应的消息体。

审计阶段（stage）

记录日志可以发生在不同的阶段，参考以下表格内容：

参数	说明
RequestReceived	一收到请求就记录。
ResponseStarted	返回消息头发送完毕后记录，只针对 watch 之类的长连接请求。
ResponseComplete	返回消息全部发送完毕后记录。
Panic	内部服务器出错，请求未完成。

TKE 审计策略

TKE 默认在收到请求时记录审计日志，且大部分的操作会记录 RequestResponse 级别的审计日志。但也会存在如下情况：
- get、list 和 watch 会记录 Request 级别的日志。
- 针对 secrets 资源、configmaps 资源或 tokenreviews 资源的请求会在 Metadata 级别记录。

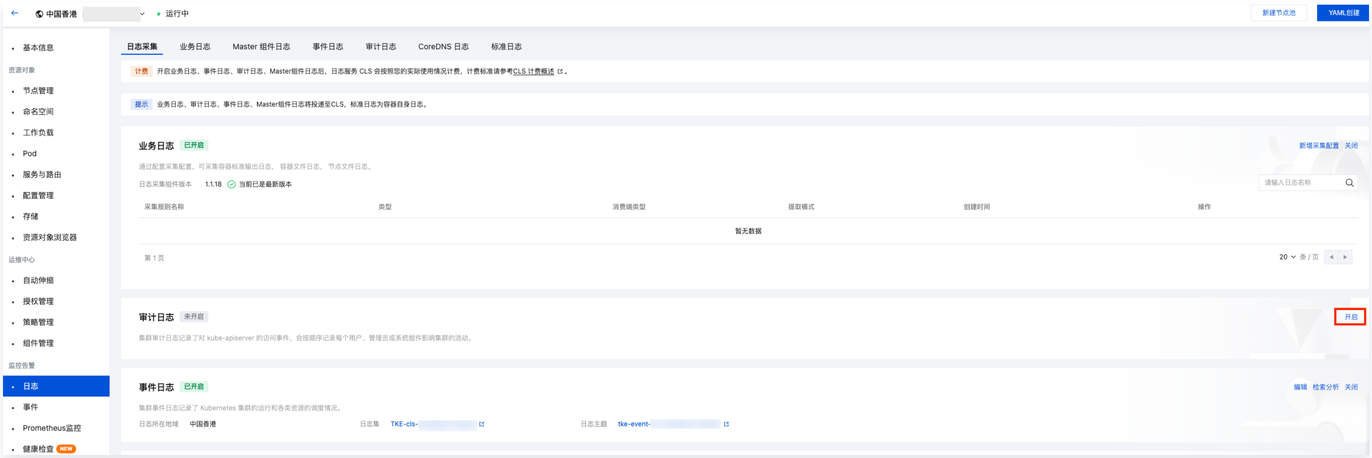
以下请求将不会进行记录日志：
- `system:kube-proxy` 发出的监视 endpoints 资源、services 资源或 services/status 资源的请求。
- `system:unsecured` 发出的针对 kube-system 命名空间中 configmaps 资源的 get 请求。
- kubelet 发出的针对 nodes 资源或 nodes/status 资源的 get 请求。
- `system:nodes` 组中的任何身份发出的针对 nodes 资源或 nodes/status 资源的 get 请求。
- `system:kube-controller-manager`、`system:kube-scheduler` 或 `system:serviceaccount:endpoint-controller` 发出的针对 kube-system 命名空间中 endpoints 资源的 get 和 update 请求。
- `system:apiserver` 发出的针对 namespaces 资源、namespaces/status 资源或 namespaces/finalize 资源的 get 请求。
- 对与 `/healthz*`、`/version` 或 `/swagger*` 匹配的网址发出的请求。

操作步骤

开启审计日志

> ⚠ **注意：**
> - 开启审计日志功能需要重启 kube-apiserver，建议不要频繁开关。
> - 独立集群会占用 Master 节点约1Gib本地存储，请保证 Master 节点存储充足。

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击**审计日志**右侧的**开启**。如下图所示：



4. 在开启审计日志中，选择日志所在地域，并配置日志集和日志主题。如下图所示：

5. 单击**确定**，即可开启审计日志。

审计仪表盘

最近更新时间：2024-09-03 17:12:51

操作场景

容器服务 TKE 为用户提供了开箱即用的审计仪表盘。在集群开启集群审计功能后，TKE 将自动为该集群配置审计总览、节点操作总览、K8S 对象操作概览、聚合检索仪表盘。还支持用户自定义配置过滤项，同时内置 CLS 的全局检索，方便用户观测和检索各类集群操作，以便于及时发现和定位问题。

功能介绍

审计检索中配置了五个大盘，分别是审计总览、节点操作概览、K8S 对象操作概览、聚合检索、全局检索。请按照以下步骤进入**审计日志**页面，开始使用对应功能：

1. 登录 容器服务控制台 。
2. 开启集群审计功能，操作详情请参见 开启集群审计 。
3. 选择左侧导航栏中的**日志管理 > 审计日志**，进入审计日志页面。

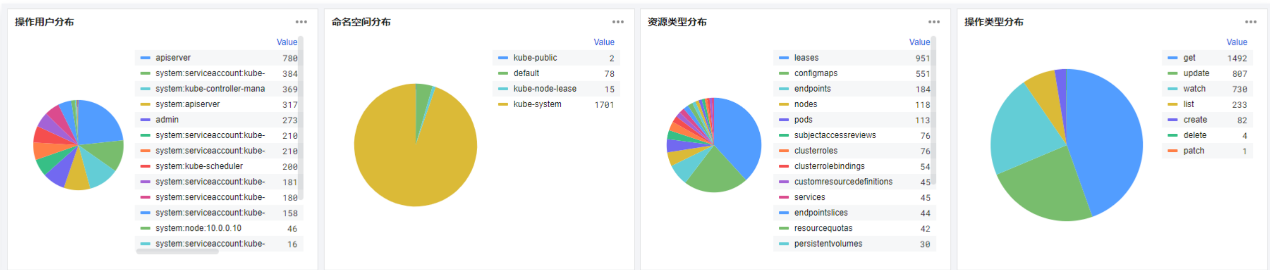
审计总览

当您想观测整个集群 APIserver 操作时，可在**审计总览**页面设置过滤条件，查看核心审计日志的汇总统计信息，并展示一个周期内的数据对比。例如，核心审计日志的统计数、分布情况、重要操作趋势等。如下图所示：

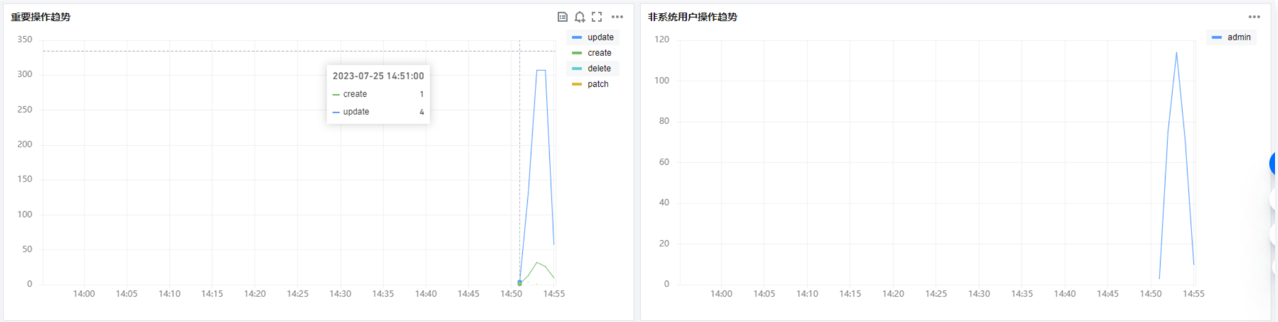
您还可在该页面中查看更多统计信息，如下所示：

- 核心审计日志的统计数仪表盘：

- 分布情况仪表盘：

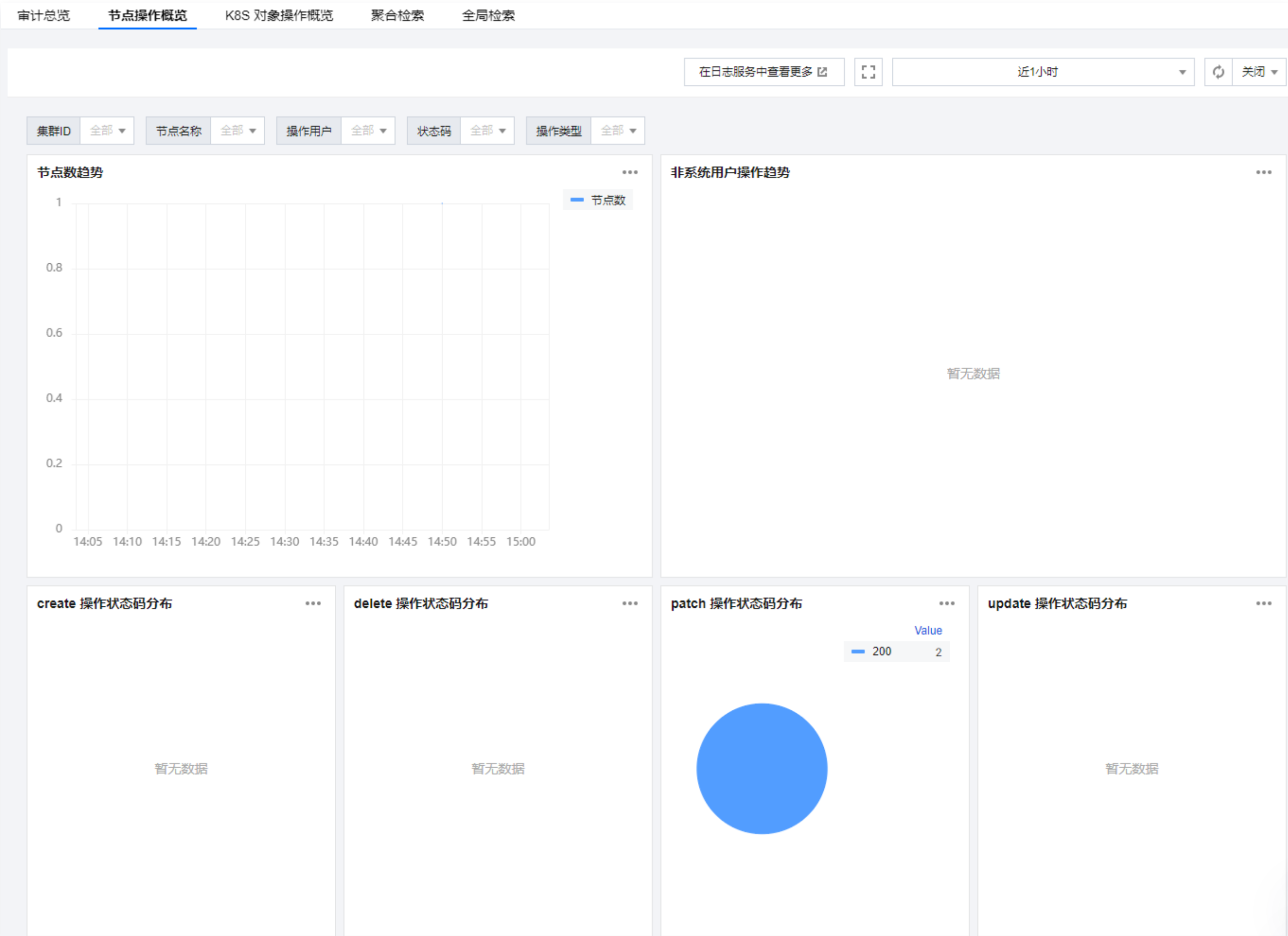


- 重要操作趋势仪表盘：



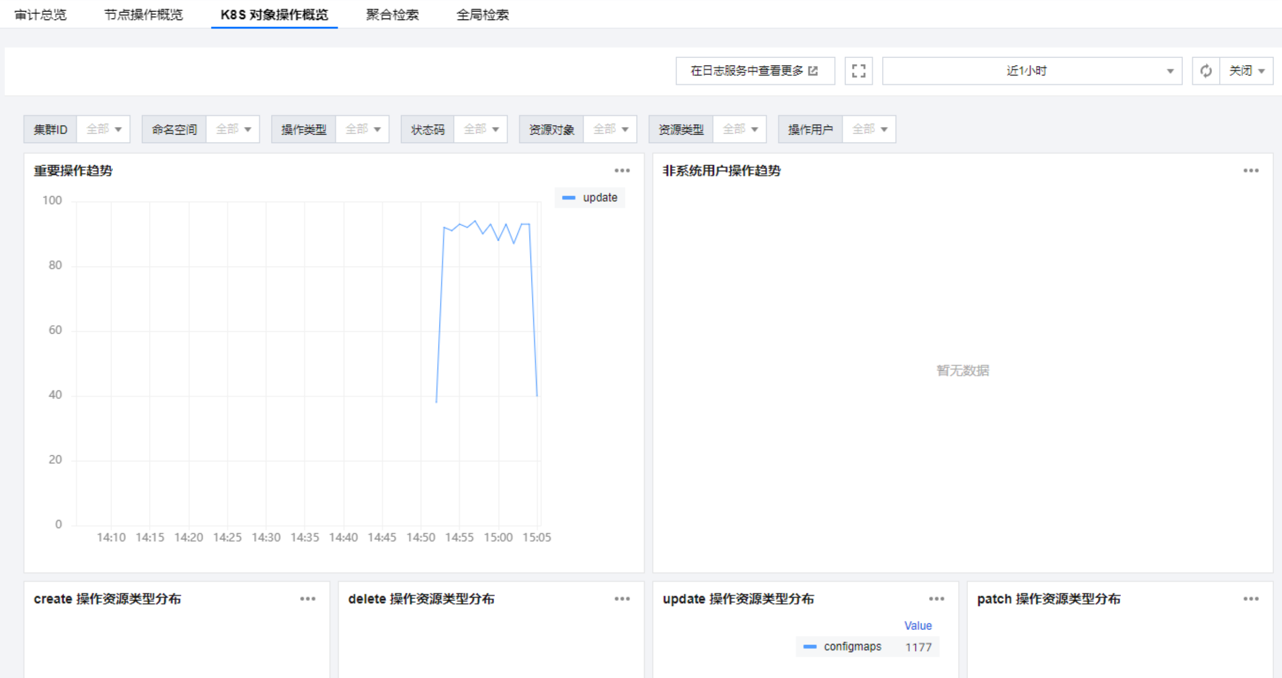
节点操作概览

当您需要排查节点相关问题时，可在**节点操作概览**页面设置过滤条件，查看各类节点操作相关的仪表，包括 create、delete、patch、update、封锁、驱逐等。如下图所示：



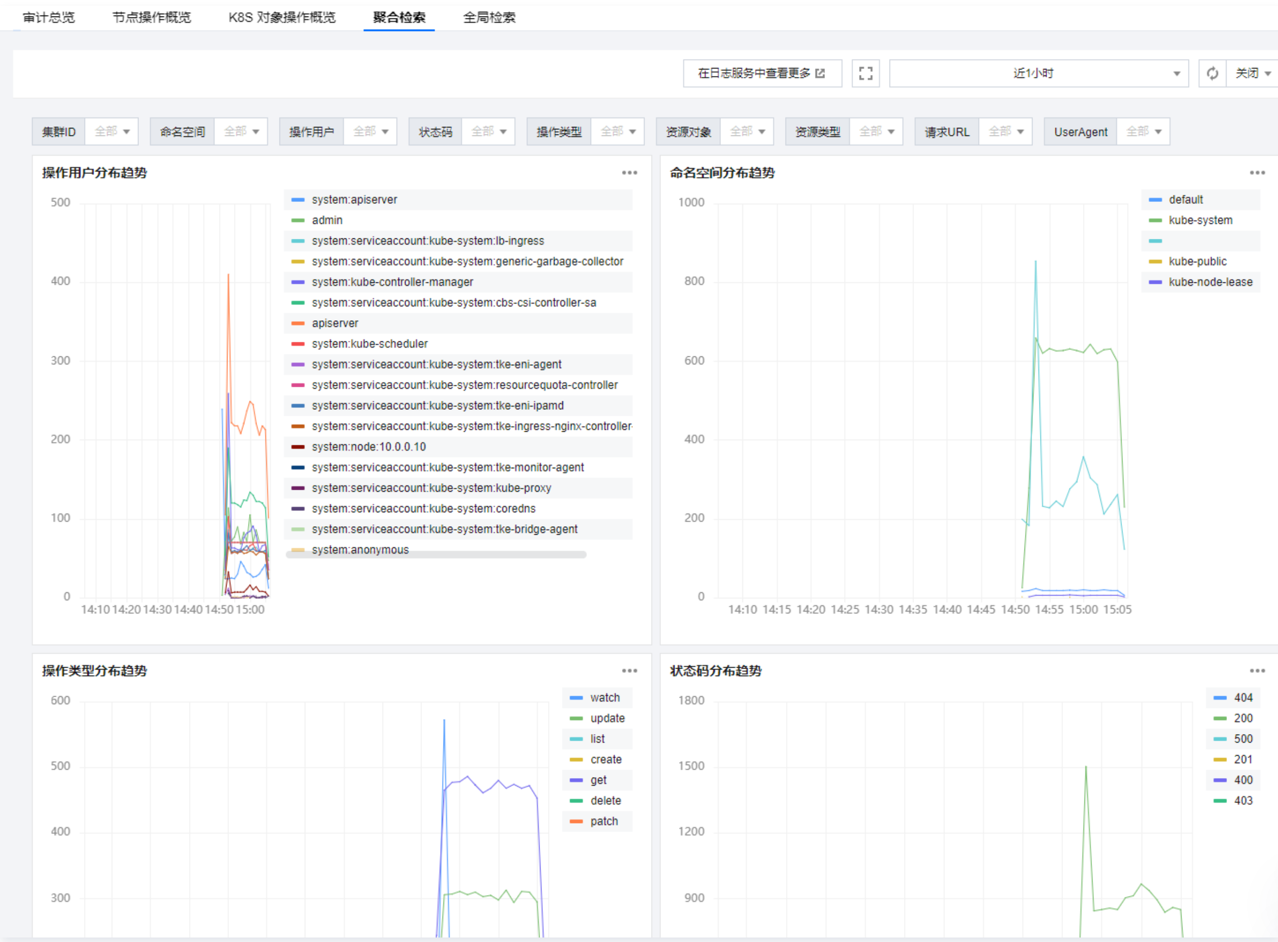
K8S 对象操作概览

当您需要排查 K8S 对象（例如某个工作负载）的相关问题时，可在 **K8S 对象操作概览**页面设置过滤条件，查看各类 K8S 对象的操作概览、对应的操作用户、相应的审计日志列表，以查找更多的细节。如下图所示：



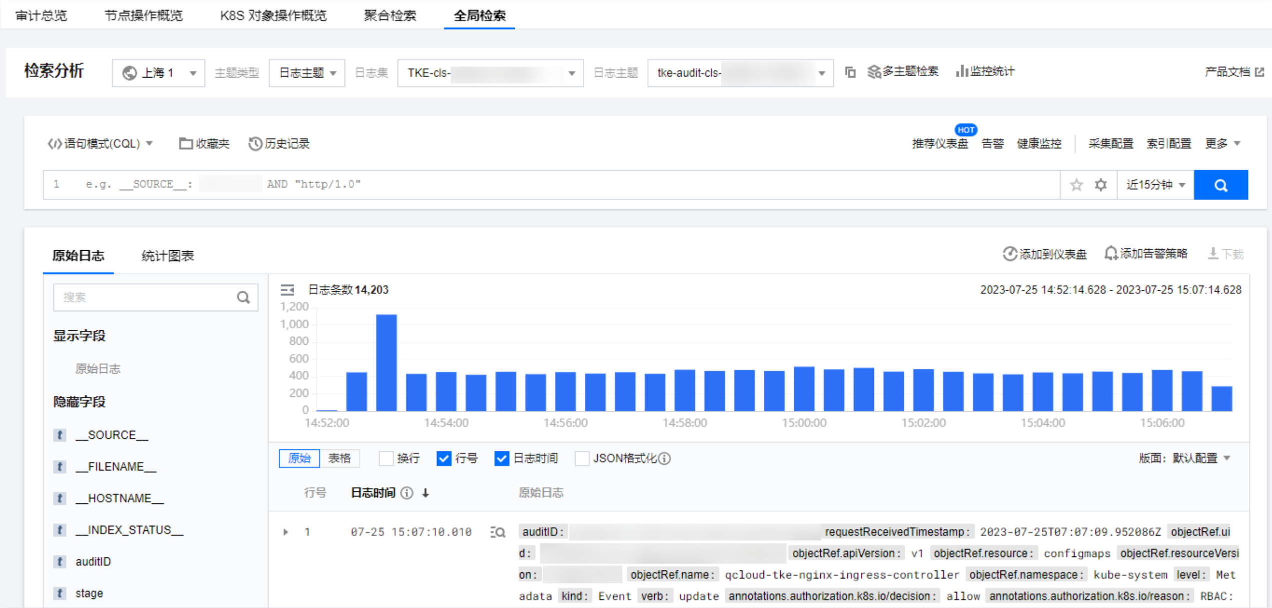
聚合检索

当您想观测某个维度下审计日志的分布趋势，可在**聚合检索**页面设置过滤条件，查看各类重要操作的时序图。维度包括操作用户、命名空间、操作类型、状态码、资源类型以及相应的审计日志列表。如下图所示：



全局检索

全局检索仪表盘中内嵌了 CLS 的检索分析页面，方便用户在容器服务控制台也能快速检索全部审计日志。如下图所示：



基于仪表盘配置告警

您可以通过以上预设的仪表盘配置告警，达到您所设置的条件则触发告警。操作详情如下：

1. 单击需要配置告警的仪表盘右侧的**添加到监控告警**。
2. 在 [日志服务控制台]() 中，选择**监控告警 > 告警策略**，新建告警策略。详情可参见 [配置告警策略]() 。

事件管理
事件日志

最近更新时间：2025-02-21 10:23:23

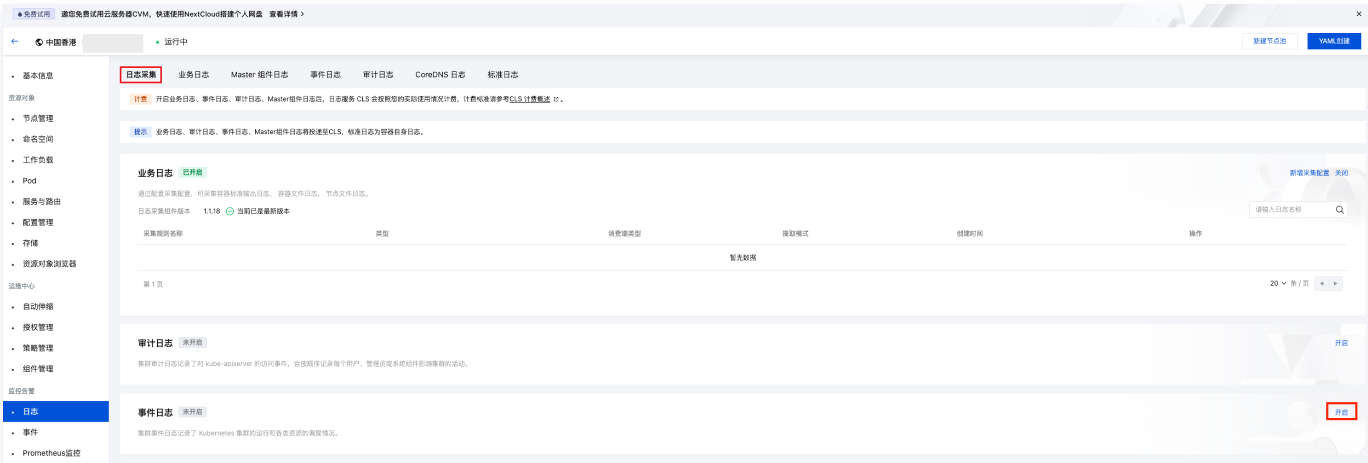
操作场景

Kubernetes Events 包括了 Kubernetes 集群的运行和各类资源的调度情况，对维护人员日常观察资源的变更以及定位问题均有帮助。TKE 支持为您的所有集群配置事件持久化功能，开启本功能后，集群事件会实时导出到配置的存储端。TKE 还支持使用腾讯云提供的 PAAS 服务或开源软件对事件流水进行检索。本文档指导您如何开启集群事件持久化存储。

操作步骤

开启事件日志

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击事件日志右侧的**开启**。如下图所示：



4. 在**开启事件日志**对话框中，选择日志所在地域，并配置日志集和日志主题。如下图所示：

   - 日志所在地域：CLS 支持跨地域日志投递，您可通过单击修改选择日志投递的目标地域。
   - 日志集：根据日志所在地域展示您已经创建的日志集，若现有日志集不合适，您可以在 日志服务控制台 新建日志集。操作详情见 创建日志集 。

   > **注意：**
   > 每个日志集下最多可以创建500个日志主题及指标主题。

   - 日志主题：选择日志集下对应的日志主题，支持“自动创建”和“选择已有”日志主题两种模式。
5. 单击**确定**，即可开启事件日志。

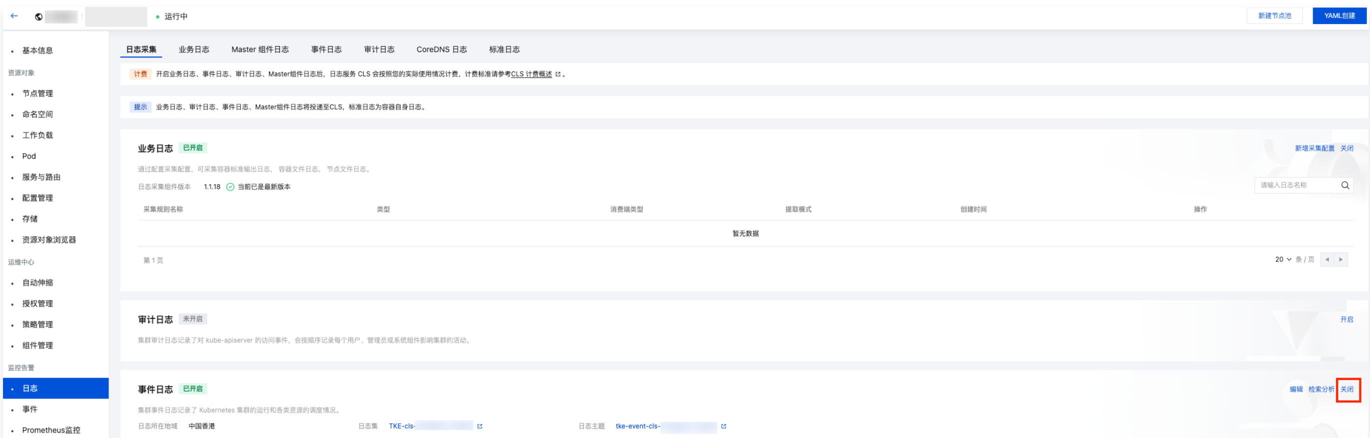
更新日志集或日志主题

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击事件日志右侧的**编辑**。
4. 在编辑事件日志中，重新选择日志集和日志主题。如下图所示：

5. 单击**确定**即可更新日志集和日志主题。

关闭事件日志

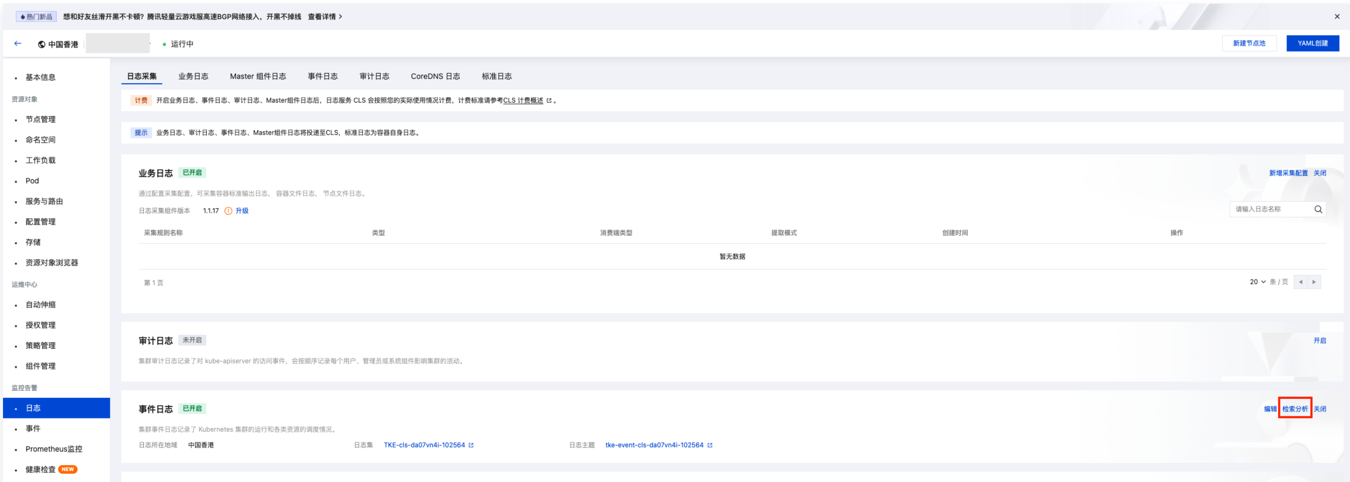
1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击事件日志右侧的**关闭**。如下图所示：



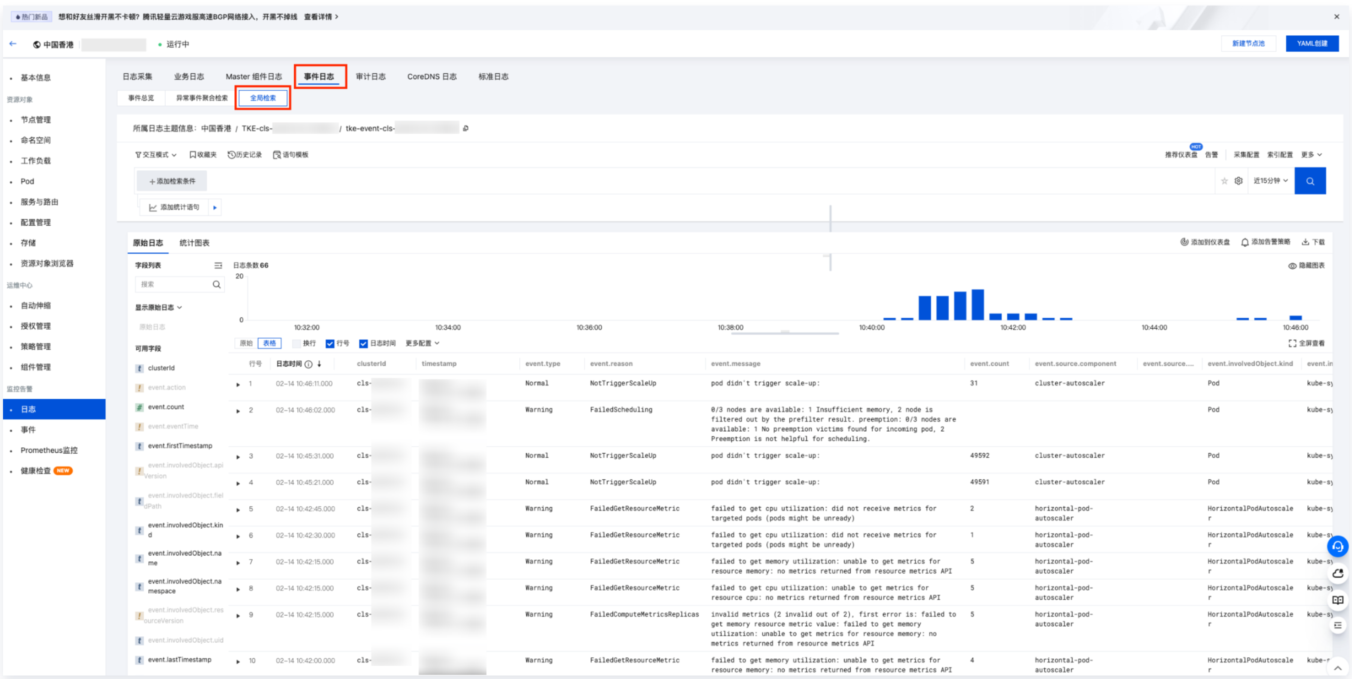
4. 在关闭事件日志页面，单击**确定**，即可关闭事件日志。如果不再需要事件日志，可勾选删除日志数据。如下图所示：

查看事件

1. 登录 [容器服务控制台]，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**日志**，在日志采集页签中单击事件日志右侧的**检索分析**。如下图所示：



4. 在**事件日志**页面，单击**全局检索**，按需自行配置显示字段并进行检索分析。



> **说明：**
> 当您开启事件日志时，将默认为您的 Topic 开启索引。详情请参见 日志检索分析 。

事件仪表盘

最近更新时间：2025-02-17 10:22:33

操作场景

容器服务 TKE 为用户提供了开箱即用的事件仪表盘。在集群开启事件日志功能后，TKE 将自动为集群配置各类事件总览大盘和异常事件的聚合检索分析仪表盘。还支持用户自定义配置过滤项，同时内置 CLS 的事件全局检索，实现在容器服务控制台全面观测、查找、分析、定位问题的能力。

功能介绍

事件检索中配置了三个大盘，分别是“事件总览”、“异常事件聚合检索”、“全局检索”。请按照以下步骤进入“事件检索”页面，开始使用对应功能：

1. 登录 容器服务控制台。
2. 开启“事件日志功能”，详情请参见 事件日志。
3. 在左侧导航栏中，选择**运维中心 > 日志管理**，在日志规则页面选择地域和集群类型。
4. 在**事件日志**页签查看集群事件详情。

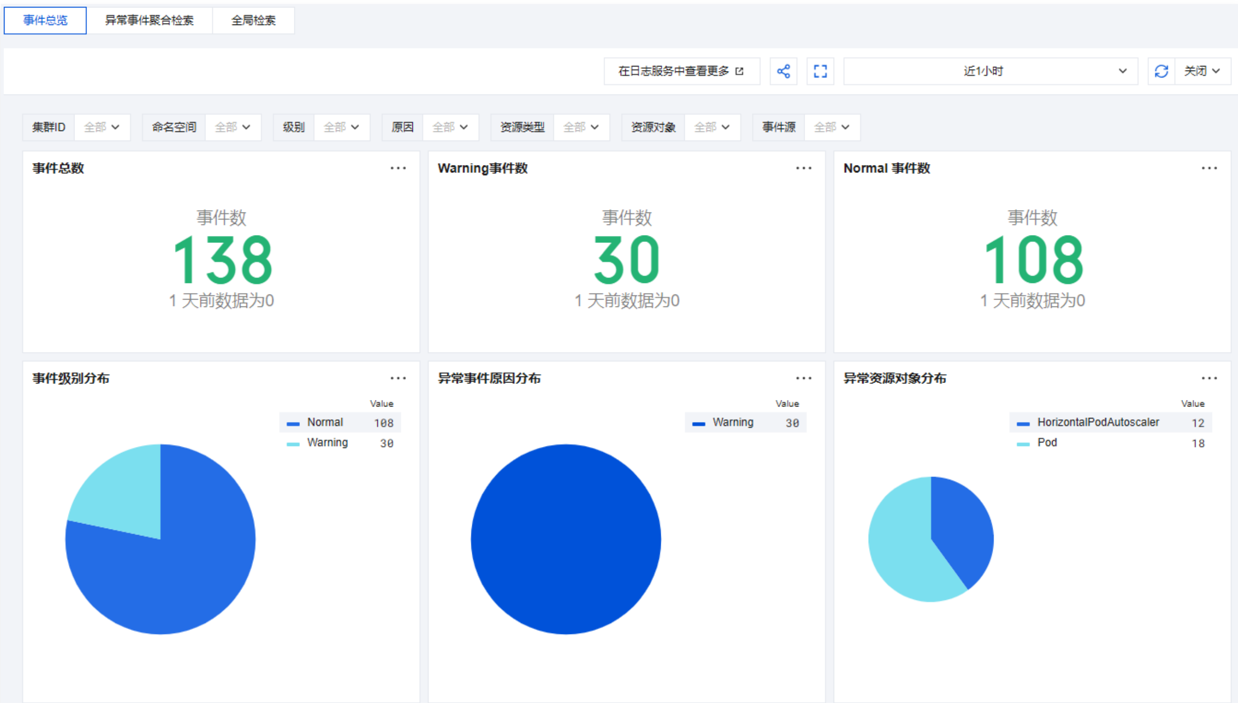
事件总览

在**事件日志 > 事件总览**页面，可根据集群 ID、命名空间、级别、原因、资源类型、资源对象、事件源等维度过滤事件，查看核心事件的汇总统计信息，并展示一个周期内的数据对比。例如，事件总数及分布情况、节点异常、Pod OOM、重要事件趋势等仪表盘以及异常 TOP 事件列表。

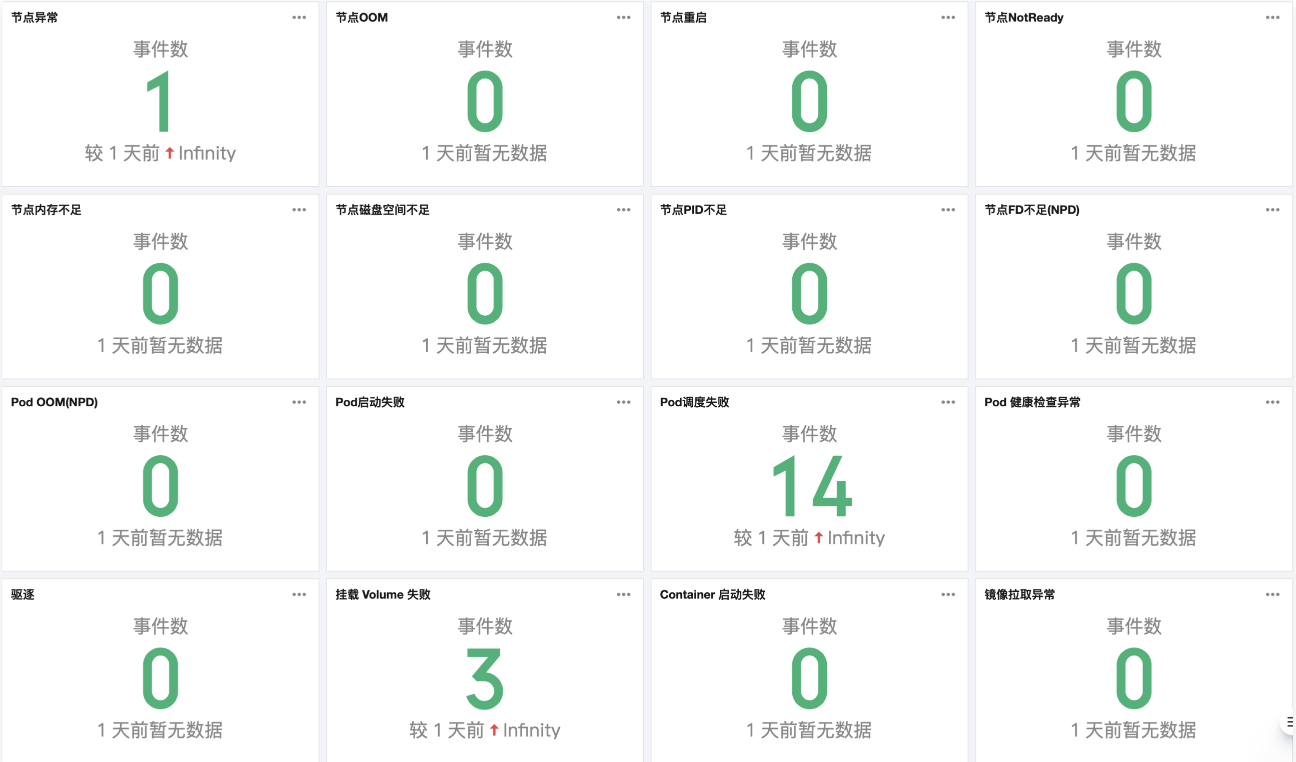
在过滤项中，您可根据自己需求进行个性化配置。如下图所示：

您还可在该页面中查看更多统计信息，如下所示：

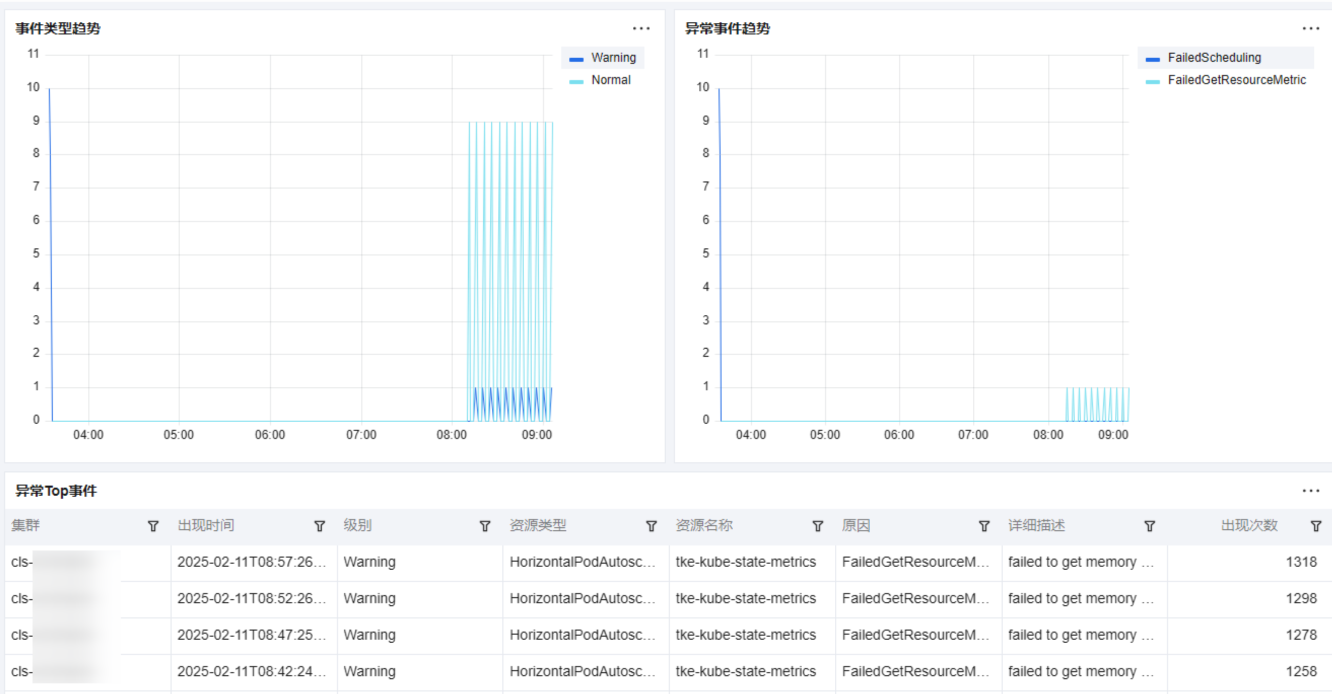
- 事件总数及级别分布情况，异常事件的原因、对象分布情况检索如下图所示：



- 各类常见事件的汇总信息检索如下图所示：

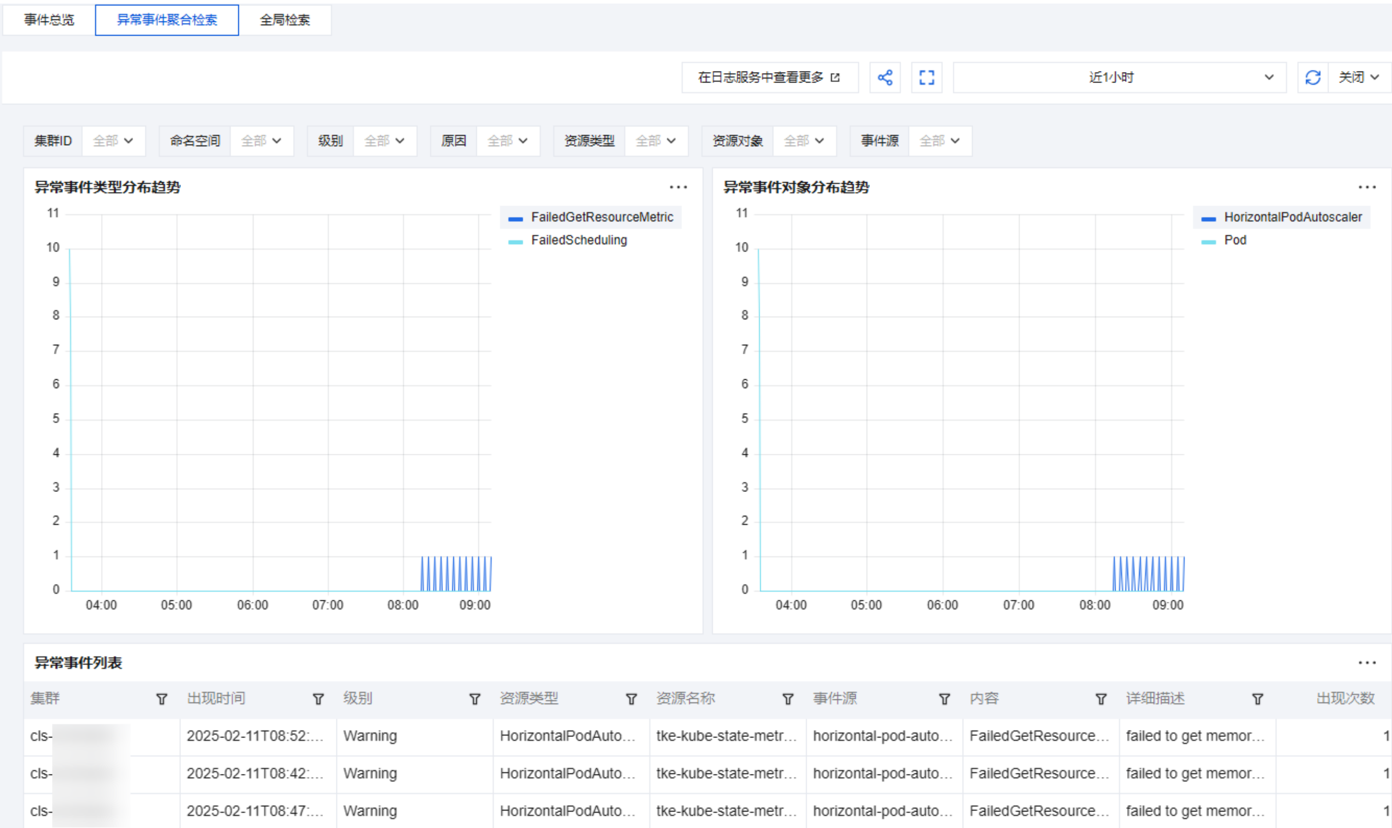


- 事件趋势及异常 TOP 事件列表检索如下图所示：



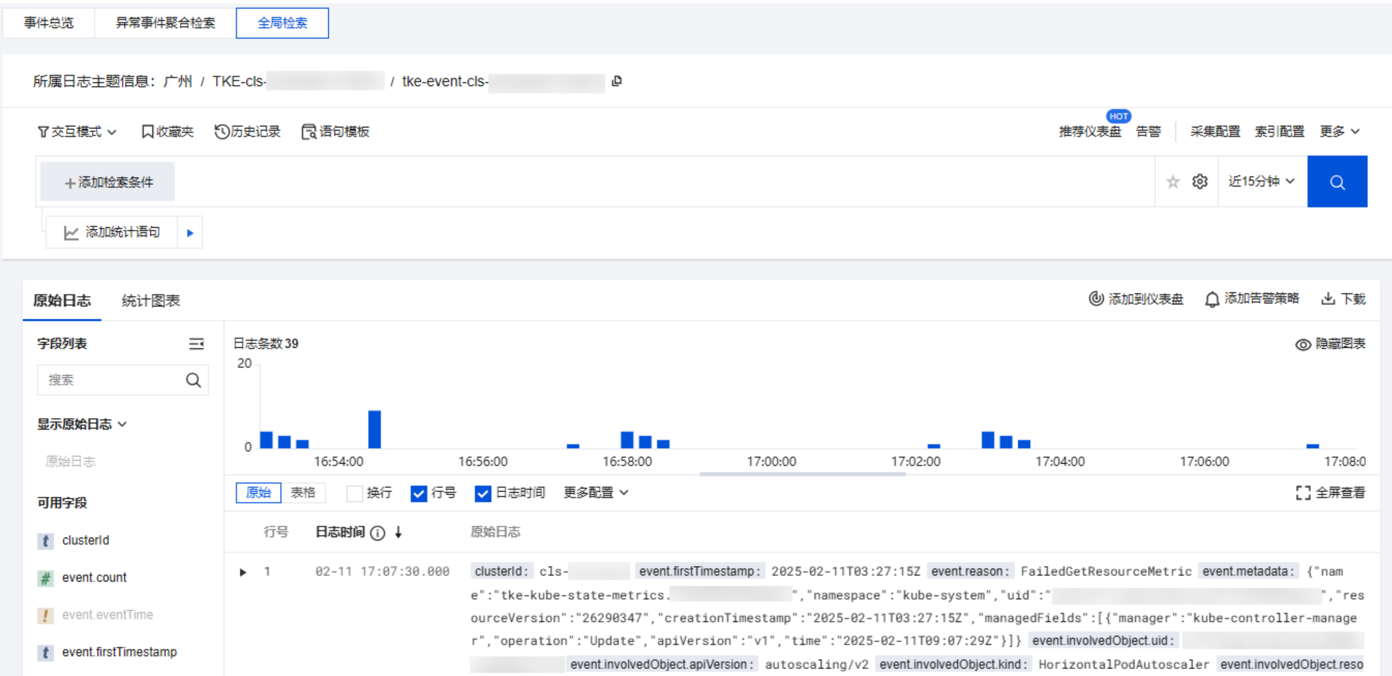
异常事件聚合检索

在**异常事件聚合检索**页面设置过滤条件，查看某个时间段内各类异常事件的 reason 和 object 分布趋势。在趋势下方展示了可供搜索的异常事件列表，帮助您快速定位到问题。如下图所示：



全局检索

全局检索仪表盘中内嵌了 CLS 的检索分析页面，方便用户在容器服务控制台也能快速检索全部事件。如下图所示：



基于仪表盘配置告警

您可以通过以上预设的仪表盘配置告警，达到您所设置的条件则触发告警。操作详情如下：

1. 单击需要配置告警的仪表盘右侧的**添加到监控告警**。如下图所示：

2. 在 日志服务控制台 中，选择**监控告警 > 告警策略**，新建告警策略。详情请参见 配置告警策略 。

监控管理
基础监控与告警
监控告警概述

最近更新时间：2023-05-10 17:09:56

概述

腾讯云容器服务 TKE 提供集群、节点、工作负载、Pod、Container 5个层面的监控数据收集和展示功能。良好的监控环境为腾讯云容器服务高可靠性、高可用性和高性能提供重要保证。通过告警配置您可以为不同资源收集不同维度的监控数据，方便掌握资源的使用状况，轻松定位故障。
收集监控数据有助于您建立容器集群性能的正常标准。通过在不同时间、不同负载条件下测量容器集群的性能并收集历史监控数据，您可以较为清楚地了解容器集群和服务运行时的正常性能，并能快速根据当前监控数据判断服务运行时是否处于异常状态，及时找出解决问题的方法。例如，您可以监控服务的 CPU 利用率、内存使用率和磁盘 I/O。

监控

容器服务的监控功能使用指引请参见 查看监控数据 。
目前覆盖的监控指标请参见 监控及告警指标列表 。

告警

为了方便您及时发现容器服务的异常状况，以保证您业务的稳定性和可靠性。建议您为所有生产集群配置必要告警，告警配置指引请参见 设置告警 。
目前覆盖的告警指标请参见 监控及告警指标列表 。

相关说明

容器服务提供的监控和告警功能主要覆盖 Kubernetes 对象的核心指标或事件，请结合 腾讯云可观测平台 提供的基础资源监控（如云服务器、块存储、负载均衡等）使用，以保证更细的指标覆盖。
若腾讯云容器服务提供的基础监控能力无法满足您的诉求，您可以使用腾讯云推出的 Prometheus 监控 服务。Prometheus 监控致力于提供轻量、稳定、高可用的服务。保留原生 Prometheus 的特性，支持采集自定义指标，支持多集群监控，支持千万级指标上报，提供基于 Grafana 的优秀可视化能力和默认面板，提供稳定的多渠道的告警能力，无侵入式架构几乎不占用您的集群资源，高度自由化的配置方式助您构建云原生场景下最适合自己的监控平台。具体操作请参见 腾讯云 Prometheus 一键关联监控容器服务 。

查看监控数据

最近更新时间：2025-04-17 17:36:02

操作场景

腾讯云容器服务默认为所有集群提供基础监控功能，您可以通过以下方式查看容器服务的监控数据。

前提条件

已登录容器服务控制台，并进入 集群 的管理页面。

操作步骤

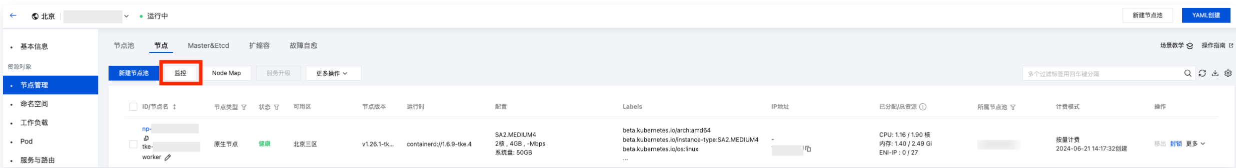
查看集群指标

在需要查看监控数据的集群行中，单击 ，即可查看该集群监控信息页面。如下图所示：

查看节点指标

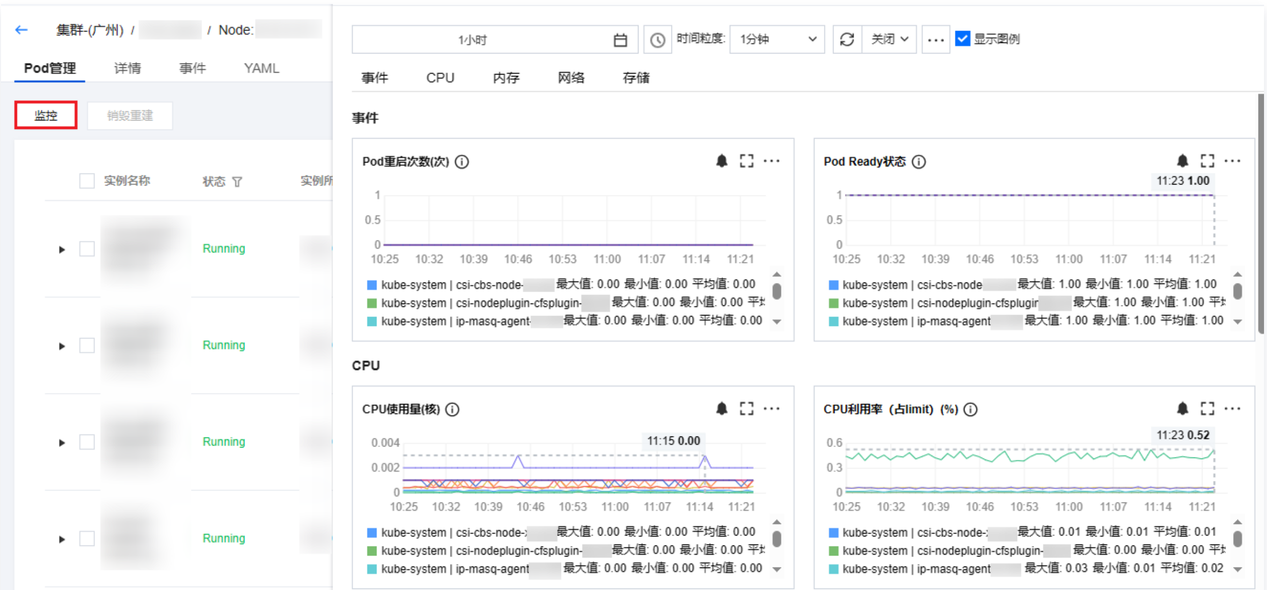
您可以通过以下操作查看节点的监控信息。

1. 选择集群 ID，进入该集群的基本信息页。
2. 在**节点管理**页面，选择**节点**页签，单击**监控**即可查看节点的监控信息。如下图所示：



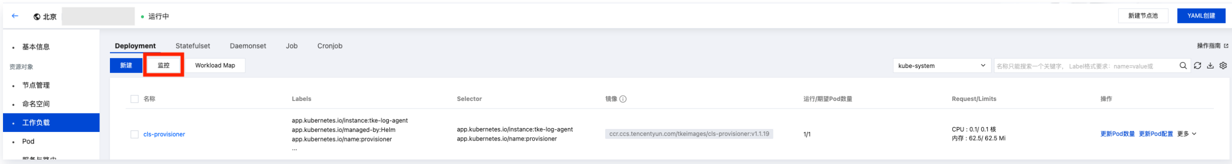
查看节点内 Pod 指标

1. 选择集群 ID，进入该集群的基本信息页。
2. 在**节点管理**页面，选择节点，进入节点详情页。
3. 在 **Pod 管理**页签中单击**监控**，即可查看到该节点内 Pod 的监控指标曲线图。如下图所示：



查看工作负载指标

1. 选择集群 ID，进入该集群的基本信息页。
2. 选择**工作负载** > **任意类型工作负载**。例如，选择 **Deployment**，进入 Deployment 管理页面。
3. 单击**监控**，即可查看该工作负载的监控信息。如下图所示：

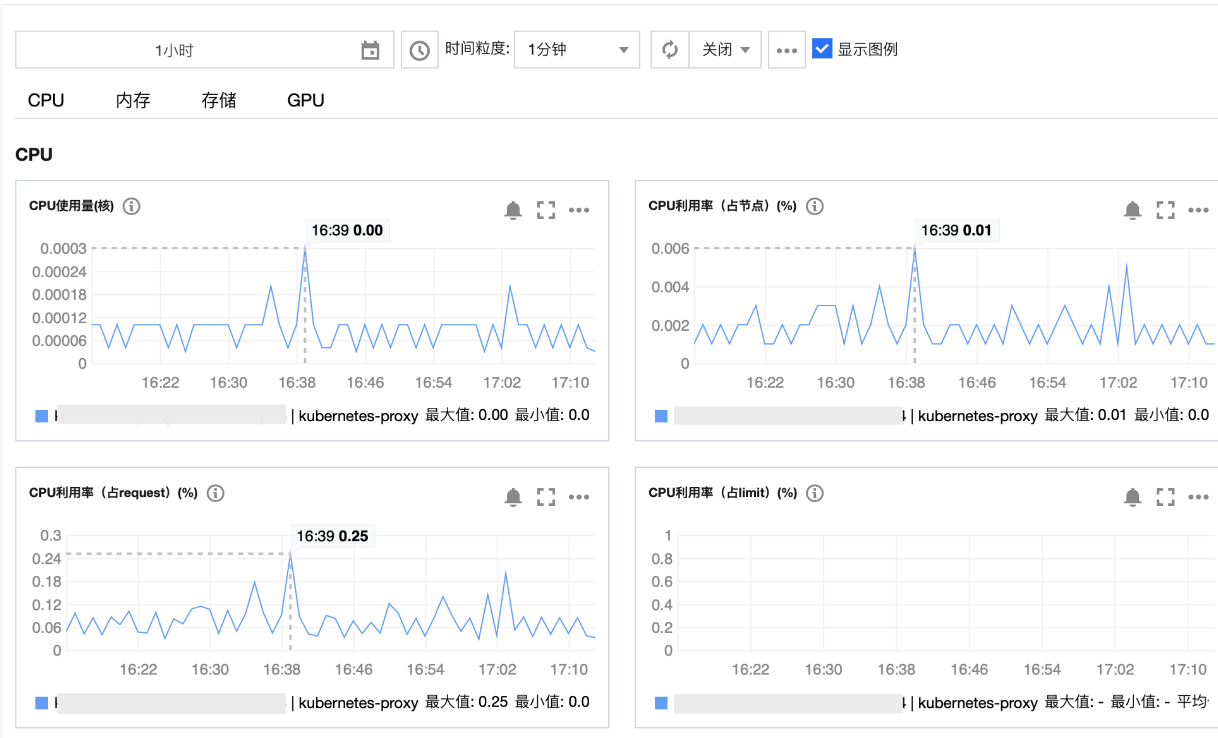


查看工作负载内 Pod 指标

1. 选择需要查看的集群 ID，进入该集群的基本信息页。
2. 选择**工作负载** > **任意类型工作负载**。例如，选择 **Deployment**，进入 Deployment 管理页面。
3. 选择工作负载名称，在该工作负载的 **Pod 管理**页签中单击**监控**，即可查看该工作负载内所有 Pod 的监控指标对比图。如下图所示：

查看 Pod 内 Container 指标

1. 选择 集群 ID，进入该集群的管理页面。
2. 选择**工作负载** > **任意类型工作负载**。例如，选择 **Deployment**，进入 Deployment 管理页面。
3. 选择工作负载名称，在该工作负载的 **Pod 管理**页签中，单击实例名称前的，即可查看该 Pod 的 Container 信息。
4. 单击，即可查看该 Pod 内 Container 的监控指标对比图。如下图所示：



监控及告警指标列表

最近更新时间：2024-12-12 09:57:22

目前容器服务提供了以下维度的监控告警指标，所有指标均为统计周期内的**平均值**。

集群监控及告警指标

指标	单位	说明
Pod 数量	个	集群中 Pod 个数
Node 数量	个	集群中 Node 个数
CPU 总配置	核	集群的 CPU 总配置量
CPU 使用量	核	集群的 CPU 使用量
CPU 利用率	%	集群的 CPU 利用率
CPU 使用量（弹性容器）	核	弹性容器的 CPU 使用量（若使用节点池的虚拟节点）
块设备读取大小	Mbytes	集群硬盘的使用总量
块设备读取次数	次	集群硬盘读取总次数
块设备写入大小	Mbytes	集群硬盘写入数据量
块设备写入次数	次	集群硬盘写入总次数
内存总和	Gbytes	集群内存总量
内存使用量	Mbytes	集群内存使用量总和
内存利用率	%	集群内存利用率
内存使用量（弹性容器）	Mbytes	弹性容器的内存使用量（若使用节点池的虚拟节点）
内存使用量（弹性容器，不含 Cache）	Mbytes	弹性容器的内存（不含 Cache）使用量（若使用节点池虚拟节点）
网络入流量	Mbytes	集群网络入流量
网络带宽	Mbps	集群网络带宽
网络入包量	个/s	集群网络入包量
网络出流量	Mbytes	集群网络出流量
网络出包量	个/s	集群网络出包量
GPU 内存总量	Gbytes	集群 GPU 内存总量
GPU 内存使用量	Mbytes	集群 GPU 内存总使用量
GPU 总量	卡	集群 GPU 总量
GPU 使用量	卡	集群整体的 CPU 利用率
显存利用率	%	GPU 显存利用率
GPU 利用率	%	集群 GPU 利用率

Master&Etcd 和普通节点监控及告警指标

指标	单位	说明
Pod 重启次数	次	节点内所有 Pod 的重启次数之和
Node 状态	-	节点的状态，正常或异常
CPU 利用率	%	节点内所有 Pod 的 CPU 使用量占节点总量之比
CPU 分配量	核	节点内所有 Pod 的 CPU 分配量总和

内存利用率	%	节点内所有 Pod 的工作集内存使用量占节点总量之比
内存分配量	Mbps	节点内所有 Pod 的内存分配量总和
内网入带宽	Mbps	节点内所有 Pod 的内网入方向带宽之和
内网出带宽	Mbps	节点内所有 Pod 的内网出方向带宽之和
外网入带宽	Mbps	节点内所有 Pod 的外网入方向带宽之和
外网出带宽	Mbps	节点内所有 Pod 的外网出方向带宽之和
TCP 连接数	个	节点保持的 TCP 连接数
GPU 使用量	卡	节点内所有 Pod 的 GPU 使用量之和
GPU 内存使用量	Mbps	节点内所有 Pod 的 GPU 内存使用量之和
GPU 内存利用率	%	节点内所有 Pod 的 GPU 内存使用量占节点 GPU 内存总量之比
GPU 利用率	%	节点内所有 Pod 的 GPU 使用量占节点 GPU 总量之比
Node 的 eni- IP 分配量	个	Node 的弹性网卡上已分配的 IP 数量
Node 的 direct-eni 分配量	个	Node 的 direct-eni 上已分配的 IP 数量
GlobalRouter模式集群中节点 Pod CIDR 已经分配的 IP 数	个	GlobalRouter 模式的 K8S 集群中，一个节点的 Pod CIDR 中已分配的 IP 个数
GlobalRouter模式集群中节点可以分配的 IP 数	个	GlobalRouter 模式的 K8S 集群中，一个节点中总共可分配的 IP 个数

集群节点更详细的指标监控及告警请参考 云服务器监控 和 创建告警策略 。
集群节点数据盘更详细的指标监控及告警请参考 云硬盘监控 和 创建告警策略 。

工作负载监控及告警指标

指标	单位	说明
工作负载异常	-	工作负载是否为异常状态，非0即为异常
Pod 数量	个	工作负载内所有 Pod 的数量和
Pod 重启次数	次	工作负载内所有 Pod 的重启次数之和
CPU 使用量	核	工作负载内所有 Pod 的 CPU 使用量
CPU 利用率	%	工作负载内所有 Pod 的 CPU 使用量占总量之比
内存使用量	Mbytes	工作负载内所有 Pod 的内存使用量之和
内存使用量（不含 Cache）	Mbytes	工作负载内所有 Pod 的内存使用量（不含 Cache）之和
内存使用量（working_set）	Mbytes	工作负载内所有 Pod 的工作集内存使用量之和
内存利用率	%	工作负载内所有 Pod 的内存使用量占总量之比
内存利用率（不含 Cache）	%	工作负载内所有 Pod 的内存使用量（不含 Cache）占所有 Pod 内存总量之比
内存利用率（working_set）	%	工作负载内所有 Pod 的工作集内存使用量占总量之比
网络入带宽	bps	工作负载内所有 Pod 的入方向带宽之和
网络出带宽	bps	工作负载内所有 Pod 的出方向带宽之和
网络入流量	B	工作负载内所有 Pod 的入方向流量之和
网络出流量	B	工作负载内所有 Pod 的出方向流量之和

网络入包量	个/s	工作负载内所有 Pod 的入方向包数之和
网络出包量	个/s	工作负载内所有 Pod 的出方向包数之和
块设备读取大小	Mbytes	工作负载内所有 Pod 的块设备读取大小之和
块设备读取次数	次	工作负载内所有 Pod 的块设备读取次数之和
块设备写入大小	Mbytes	工作负载内所有 Pod 的块设备写入大小之和
块设备写入次数	次	工作负载内所有 Pod 的块设备写入次数之和
GPU 使用量	卡	工作负载内所有 Pod 的 GPU 使用量之和
GPU 内存使用量	Mbps	工作负载内所有 Pod 的 GPU 内存使用量之和
GPU 内存利用率	%	工作负载内所有 Pod 的 GPU 内存使用量与 GPU 内存总量之比
GPU 利用率	%	工作负载内所有 Pod 的 GPU 使用量与 GPU 总量之比

如果工作负载对集群外部提供服务，绑定的 Service 更详细的网络监控指标请参考 负载均衡监控 。

Pod 监控及告警指标

指标	单位	说明
Pod 重启次数	次	Pod 的重启次数
异常状态	-	Pod 的状态，正常或异常
CPU 使用量	核	Pod 的 CPU 使用量
CPU 利用率（占节点）	%	Pod 的 CPU 使用量占节点总量之比
CPU 利用率（占 Request）	%	Pod 下设置了 Request 容器的 CPU 使用量之和与设置的 Request 值之和的比值（没设置 Request 的容器不参与计算）
CPU 利用率（占 Limit）	%	Pod 下设置了 Limit 容器的 CPU 使用量之和与设置的 Limit 之和的比值（没设置 Limit 的容器不参与计算）
内存使用量	Mbytes	Pod 中 Container 的内存使用量（含缓存）之和（来源：container_memory_usage_bytes）
内存使用量（不包含 Cache）	Mbytes	Pod 中 Container 的内存使用量（不含缓存）之和（来源：container_memory_usage_bytes - container_memory_cache）
内存使用量（working_set）	Mbytes	Pod 中 Container 的工作集内存使用量（来源：container_memory_working_set_bytes）
内存利用率（占节点）	%	Pod 中 Container 的内存使用量（含缓存）占节点总量之比
内存利用率（占节点，不包含 Cache）	%	Pod 中 Container 的内存使用量（不含缓存）占节点总量之比
内存利用率（占节点，working_set）	%	Pod 中 Container 的工作集内存使用量占节点总量之比
内存利用率（占 Request）	%	Pod 下设置了 Request 容器的内存使用量之和与设置的 Request 值之和的比值（没设置 Request 的容器不参与计算）
内存利用率（占 Request，不包含 Cache）	%	Pod 下设置了 Request 容器的内存使用量（不含缓存）之和与设置的 Request 值之和的比值（没设置 Request 的容器不参与计算）
内存利用率（占 Request，working_set）	%	Pod 下设置了 Request 容器的工作集内存使用量之和与设置的 Request 值之和的比值（没设置 Request 的容器不参与计算）
内存利用率（占 Limit）	%	Pod 下设置了 Limit 容器的内存使用量之和与设置的 Limit 值之和的比值（没设置 Limit 的容器不参与计算）

内存利用率（占 Limit，不包含 Cache）	%	Pod 下设置了 Limit 容器的内存使用量（不含缓存）之和与设置的 Limit 值之和的比值（没设置 Request 的容器不参与计算）
内存利用率（占 Limit，working_set）	%	Pod 下设置了 Limit 容器的工作集内存使用量之和与设置的 Limit 值之和的比值（没设置 Limit 的容器不参与计算）
网络入带宽	Mbps	Pod 的入方向带宽之和
网络出带宽	Mbps	Pod 的出方向带宽之和
网络入流量	Mbytes	Pod 的入方向流量之和
网络出流量	Mbytes	Pod 的出方向流量之和
网络入包量	个/s	Pod 的入方向包数之和
网络出包量	个/s	Pod 的出方向包数之和
块设备读取大小	Mbytes	Pod 的块设备读取大小
块设备读取次数	次	Pod 的块设备读取次数
块设备写入大小	Mbytes	Pod 的块设备写入大小
块设备写入次数	次	Pod 的块设备写入次数
rootfs使用量	字节	Pod 中 rootfs 使用量
GPU 申请量	卡	Pod 中 GPU 申请量
GPU 内存利用率（占节点）	%	Pod 中 GPU 内存使用量占节点 GPU 内存总量之比
GPU 内存利用率（占 request）	%	Pod 中 GPU 内存使用量占 GPU 内存申请量之比
GPU 利用率（占节点）	%	Pod 中 GPU 使用量占节点 GPU 总量之比
GPU 利用率（占 request）	%	Pod 中 GPU 使用量占 GPU 申请量之比
GPU 内存申请量	Mbytes	Pod 中 GPU 内存申请量
GPU 内存使用量	Mbytes	Pod 中 GPU 内存使用量
GPU 使用量	卡	Pod 中 GPU 使用量
GPU 显存使用率	%	Pod 中 GPU 显存使用量占显存总量的百分比
GPU 编码资源使用率	%	Pod 中 GPU 编码资源使用率
GPU 解码资源使用率	%	Pod 中 GPU 解码资源使用率
GPU 流处理器使用率	%	Pod 中 GPU 流处理器使用率

Container 监控及告警指标

指标	单位	说明
CPU 使用量	核	Container 的 CPU 使用量
CPU 利用率（占节点）	%	Container 的 CPU 使用量占节点总量之比
CPU 利用率（占 Request）	%	Container 的 CPU 使用量和设置的 Request 值之比
CPU 利用率（占 Limit）	%	Container 的 CPU 使用量和设置的 Limit 值之比
内存使用量	Mbytes	Container 的内存使用量，含缓存（来源：container_memory_usage_bytes）

内存使用量（不包含 Cache）	Mbytes	Container 的内存使用量，不含缓存（来源：container_memory_usage_bytes − container_memory_cache）
内存使用量（working_set）	Mbytes	Container 的工作集内存使用量（来源：container_memory_working_set_bytes）
内存利用率（占节点）	%	Container 的内存使用量（含缓存）占节点总量之比
内存利用率（占节点，不包含 Cache）	%	Container 的内存使用量（不含缓存）占节点总量之比
内存利用率（占节点，working_set）	%	Container 的工作集内存使用量占节点总量之比
内存利用率（占 Request）	%	Container 的内存使用量和设置的 Request 值之比
内存利用率（占 Request，不包含 Cache）	%	Container 的内存使用量（不含缓存）和设置的 Request 值之比
内存利用率（占 Request，working_set）	%	Container 的工作集内存使用量与设置的 Request 值之比
内存利用率（占 Limit）	%	Container 的内存使用量和设置的 Limit 值之比
内存利用率（占 Limit，不包含 Cache）	%	Container 的内存使用量（不含缓存）和设置的 Limit 值之比
内存利用率（占 Limit，working_set）	%	Container 的工作集内存使用量与设置的 Limit 值之比
块设备读带宽	B/s	Container 从硬盘读取数据的吞吐量
块设备写带宽	B/s	Container 把数据写入硬盘的吞吐量
块设备读 IOPS	次/s	Container 从硬盘读取数据的 IO 次数
块设备写 IOPS	次/s	Container 把数据写入硬盘的 IO 次数

GPU 监控指标获取

最近更新时间：2024-08-28 16:51:01

组件介绍

TKE 开发了 `elastic-gpu-exporter` 组件，用于获取 GPU 相关的监控指标。主要包含以下内容：

- GPU 卡利用率
- Pod / 容器 GPU 资源利用率

部署方式

`elastic-gpu-exporter` 通过 DaemonSet 的方式部署到集群中。

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: elastic-gpu-exporter
  namespace: kube-system
  labels:
    app: elastic-gpu-exporter
spec:
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      name: gpu-manager-ds
      app: nano-gpu-exporter
  template:
    metadata:
      name: elastic-gpu-exporter
      labels:
        name: gpu-manager-ds
        app: nano-gpu-exporter
    spec:
      nodeSelector:
        qgpu-device-enable: enable
      serviceAccount: elastic-gpu-exporter
      hostNetwork: true
      hostPID: true
      hostIPC: true
      containers:
        - image: ccr.ccs.tencentyun.com/tkeimages/elastic-gpu-exporter:v1.0.8
          imagePullPolicy: Always
          args:
            - --node=$(NODE_NAME)
          env:
            - name: "PORT"
              value: "5678"
            - name: "NODE_NAME"
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          name: elastic-gpu-exporter
          securityContext:
            capabilities:
              add: ["SYS_ADMIN"]
          volumeMounts:
            - name: cgroup
              readOnly: true
```

```
              mountPath: "/host/sys"
      volumes:
        - name: cgroup
          hostPath:
            type: Directory
            path: "/sys"
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: elastic-gpu-exporter
rules:
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - update
      - patch
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - bindings
      - pods/binding
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - watch
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: elastic-gpu-exporter
  namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: elastic-gpu-exporter
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: elastic-gpu-exporter
subjects:
  - kind: ServiceAccount
    name: elastic-gpu-exporter
    namespace: kube-system
---
apiVersion: v1
kind: Service
metadata:
  name: elastic-gpu-exporter
  namespace: kube-system
  annotations:
    prometheus.io/scrape: "true"
  labels:
    kubernetes.io/cluster-service: "true"
spec:
  clusterIP: None
  ports:
    - name: elastic-gpu-exporter
      port: 5678
      protocol: TCP
      targetPort: 5678
  selector:
    app: nano-gpu-exporter
```

查看运行状态

部署后，集群中生成一个 elastic-gpu-exporter 的 DaemonSet：

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
elastic-gpu-exporter   1         1         1       1            1           <none>          3m36s
```

符合条件的节点上会有一个运行的 elastic-gpu-exporter 的 Pod：

```
NAME                         READY   STATUS    RESTARTS   AGE
elastic-gpu-exporter-dblqm   1/1     Running   0          6s
```

获取监控指标

elastic-gpu-exporter 的服务运行所在节点会输出到 `/metrics` 路径下，因此也可以通过如下命令获取监控指标：

```
$ curl NodeIP:5678/metrics
```

GPU 卡相关指标

gpu_xxx	GPU 指标
gpu_core_usage	GPU 实际使用的算力
gpu_mem_usage	GPU 实际使用的显存
gpu_core_utilization_percentage	GPU 算力使用率

gpu_mem_utilization_percentage	GPU 显存使用率

GPU 卡的指标形式为：

```
gpu_core_usage{card="0",node="10.0.66.4"} 0
```

说明：
“card” 表示 GPU 的序号，“node” 表示所在节点。

Pod 相关指标

pod_xxx	pod 指标
pod_core_usage	pod 实际使用的算力
pod_mem_usage	pod 实际使用的显存
pod_core_utilization_percentage	pod 实际使用的算力占申请算力的百分比
pod_mem_utilization_percentage	pod 实际使用的显存占申请显存的百分比
pod_core_occupy_node_percentage	pod 实际使用的算力占节点总算力百分比
pod_mem_occupy_node_percentage	pod 实际使用的显存占节点总显存百分比
pod_core_request	pod 申请的算力
pod_mem_request	pod 申请的显存

Pod 的指标形式为：

```
pod_core_usage{namespace="default",node="10.0.66.4",pod="7a2fa737-eef1-4801-8937-493d7efb16b7"} 0
```

说明：
“namespace” 表示 Pod 所在的 namespace，“node” 表示 Pod 所在节点，“pod” 表示 Pod 的名称。

容器相关指标

container_xxx	容器指标
container_gpu_utilization	容器实际使用的算力
container_gpu_memory_total	容器实际使用的显存
container_core_utilization_percentage	容器实际使用的算力占申请算力的百分比
container_mem_utilization_percentage	容器实际使用的显存占申请显存的百分比
container_request_gpu_memory	容器申请的显存
container_request_gpu_utilization	容器申请的算力

容器的指标形式为：

```
container_gpu_utilization{container="cuda",namespace="default",node="10.0.66.4",pod="cuda"} 0
```

说明：
“container” 表示容器名称，“namespace” 表示容器所在的 namespace，“node” 表示容器所在节点，“pod” 表示容器所在的 Pod 的名称。

控制面组件监控
查看 TKE 集群控制面组件监控

最近更新时间：2025-02-27 17:55:12

使用场景

腾讯云容器服务 TKE 是一种托管的 Kubernetes 服务，TKE 自动管理 Kubernetes 控制平面节点的可用性和可扩展性，这些节点负责调度容器、管理应用程序可用性、存储集群数据和其他关键任务。但某些工作负载行为或配置可能会增加控制平面的负载，从而导致性能下降。在这些情况下，访问关键控制面指标和监控图表使集群管理员能够快速检测和排除集群上运行的工作负载的问题。例如，资源不足的工作节点可能会妨碍调度程序调度新 Pod 的能力。为了及时检测这些新出现的调度问题，集群管理员需要访问调度程序指标以查看待处理的 Pod 并及时收到通知。此外，控制面指标数量庞大，即使是经验丰富的管理员也面临选择最佳监控指标和创建有效监控图表的挑战。

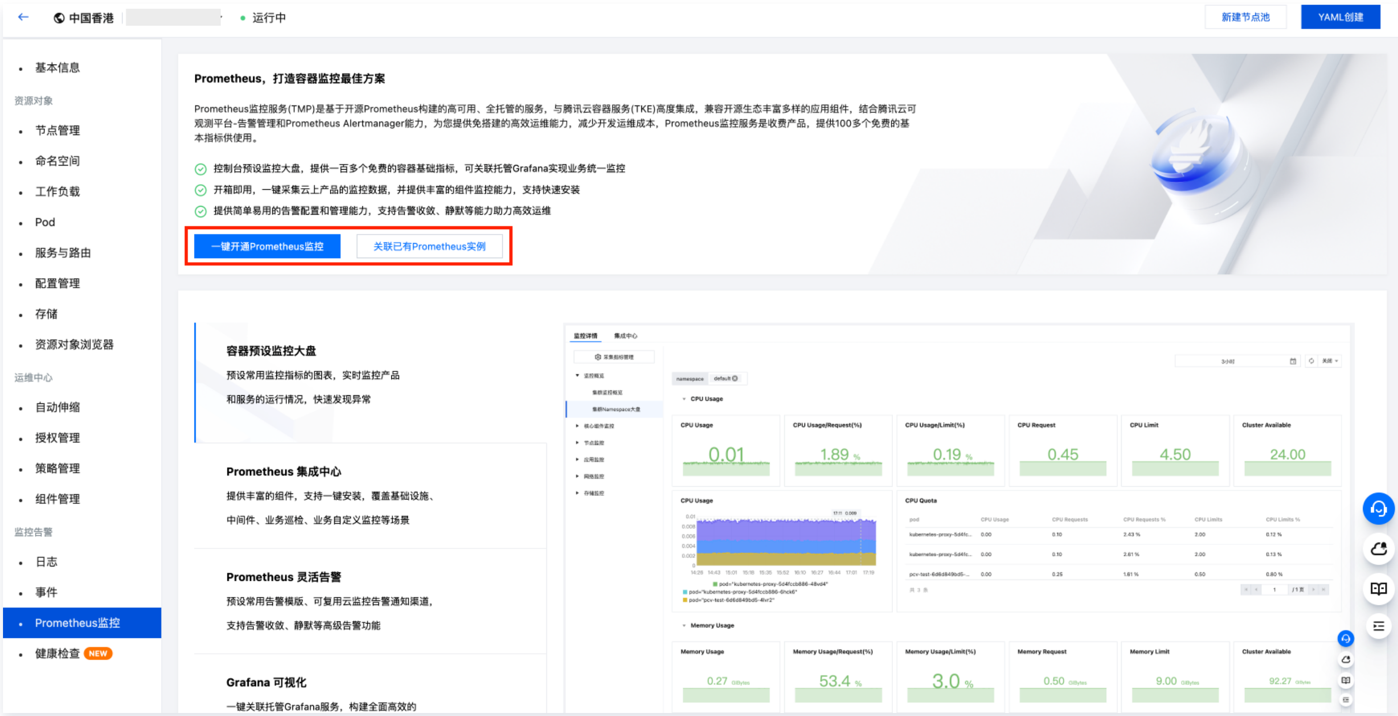
TKE 提供了集群控制面组件的监控能力，使集群管理员能够查看 Kubernetes 集群控制面的性能，快速检测、排除故障并修复问题。TKE 集群通过 Prometheus 监控提供监控图表。

前提条件

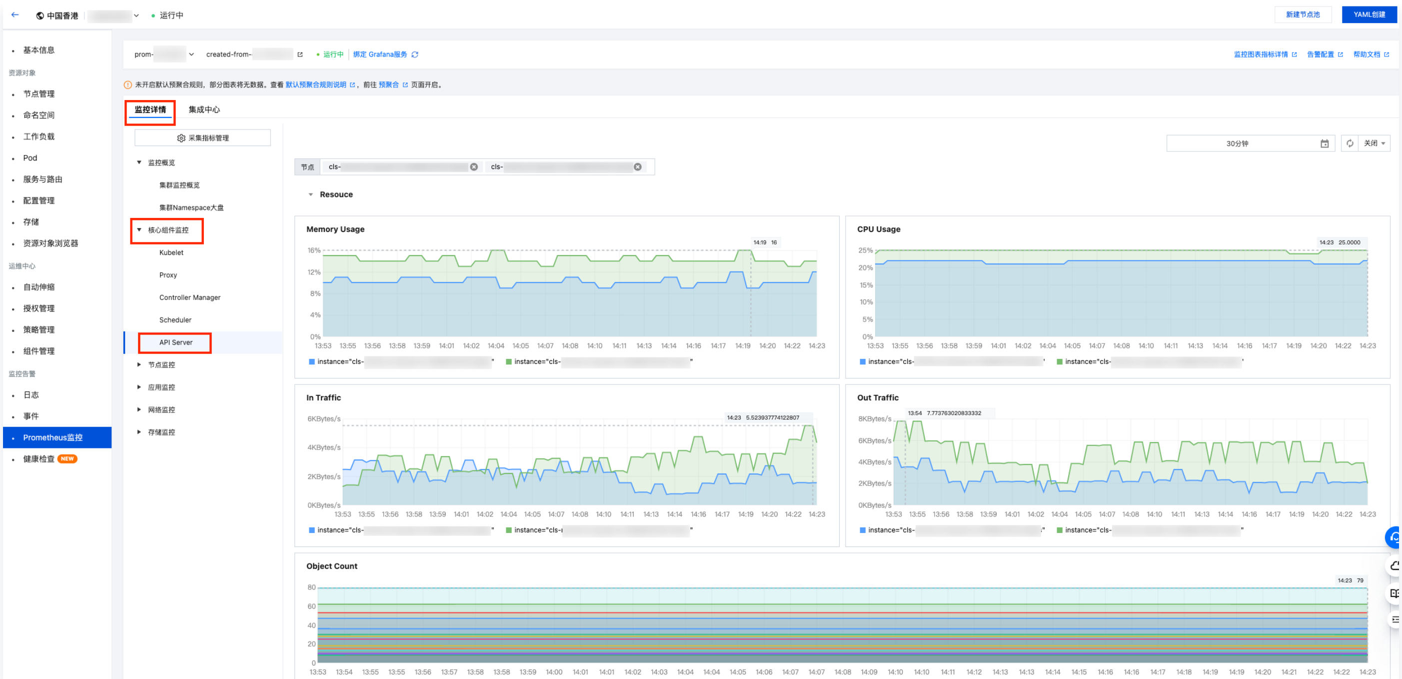
- 已将监控组件 clustermonitor 升级到 1.3.2 版本以上。
- 已开通 Prometheus 监控服务并关联实例。操作详情请参见 腾讯云 Prometheus 一键关联监控容器服务 。

操作步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群ID，进入集群详情页。
3. 选择左侧导航中的 **Prometheus 监控**。
4. 可通过**一键开通 Prometheus 监控**、**关联已有 Prometheus 实例**开启 Prometheus 监控。



5. 在 Prometheus 监控页面中，单击监控详情，单击**核心组件监控**。



在该页面，您可以查看 API Server、Kube Scheduler、Controller Manager 监控大盘。

相关文档

- kube-apiserver 组件：指标列表、对应监控大盘的使用说明，请参见 kube-apiserver 组件指标说明。
- kube-scheduler 组件：指标列表、对应监控大盘的使用说明，请参见 kube-scheduler 组件指标说明。
- kube-controller-manager 组件：指标列表、对应监控大盘的使用说明，请参见 kube-controller-manager组件指标说明。
- 如何基于用户自建的 Prometheus，采集 TKE 集群的控制面组件监控 API Server、Scheduler、KCM 指标配置说明及推荐的告警配置，请参见 用户自建 Prometheus 采集控制面监控和告警。

kube-apiserver 组件指标说明

最近更新时间：2025-04-23 15:05:02

前提条件

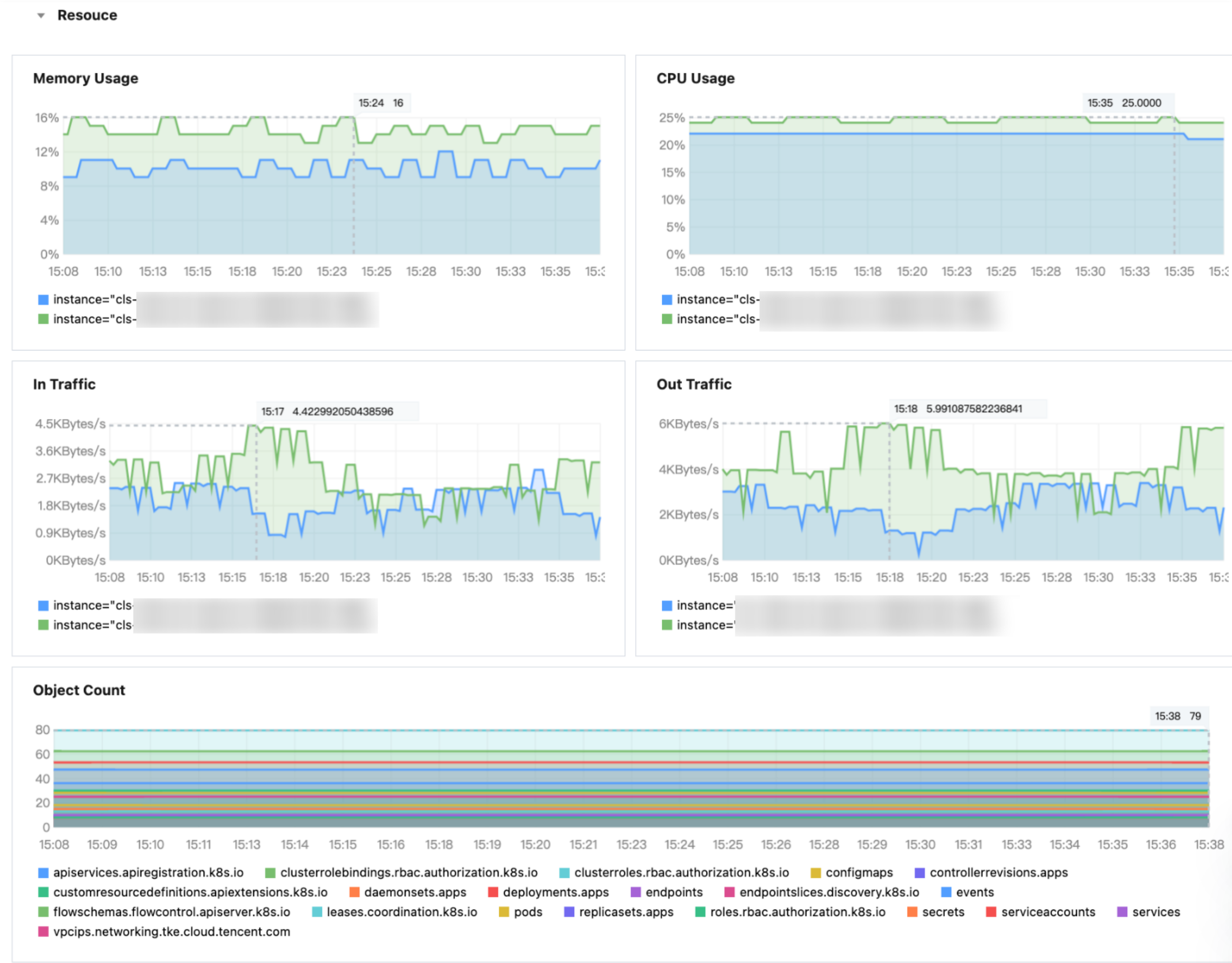
请参见 查看 TKE 集群控制面组件监控。

指标列表

功能分类	指标	类型	说明
请求概览	apiserver_request_duration_seconds_bucket	Histogram	该指标用于统计 APIServer 客户端对 API Server 不同请求的时延分布。 请求的维度包括： • Verb：请求的类型，例如 GET、POST、PUT、DELETE 等。 • Group：API 组，即相关 API 接口的集合，用于扩展 Kubernetes API。 • Version：API 版本，例如v1、v1beta1等。 • Resource：请求针对的资源类型，例如 Pod、Service、Lease 等。 • Subresource：资源的子资源，例如 Pod 详细信息、Pod 日志等。 • Scope：请求的范围，例如命名空间维度的资源（Namespace-scope）或集群维度的资源（Cluster-scope）。 • Component：发起请求的组件的名称，例如 kube-controller-manager、kube-scheduler、cloud-controller-manager 等。 • Client：发起请求的客户端，可能是内部组件或外部服务。 API ServerHistogram 的 Bucket 阈值为{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5, 1.75, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 60}。单位：秒。
请求概览	apiserver_request_total	Counter	对 API Server 不同请求的计数。请求的维度包括 Verb、Group、Version、Resource、Scope、Component、HTTP contentType、HTTP code（响应的HTTP状态码）和 Client。
资源请求	apiserver_request_count	Counter	API Server 接收到的请求总数。
资源分析	apiserver_storage_objects	Gauge	资源对象数量。
请求概览	apiserver_watch_events_total	Counter	资源对象总量。按照资源类型分类，例如 nodes，pods 等。
请求概览	apiserver_current_inflight_requests	Gauge	API Server 当前处理的请求数量。请求包括两种： • ReadOnly：这类请求不会改变集群的状态，通常为读取资源的操作，例如获取 Pods 列表、查询节点状态等。 • Mutating：这类请求会改变集群的状态，通常为创建、更新或删除资源的操作，例如新建 Pod、更新 Service 配置等。
资源分析	pod_core_usage	Gauge	CPU 使用率。单位：%。
资源分析	pod_mem_usage	Gauge	内存使用率。单位：%。
资源分析	container_network_receive_bytes_total	Counter	网络入流量。单位：Bit。 说明：原指标 k8s_pod_network_receive_bytes_bw 需收费，因此进行指标名称映射为免费指标。
资源分析	container_network_transmit_packets_total	Counter	网络出流量。单位：Bit。 说明：原指标 k8s_pod_network_transmit_bytes_bw 需收费，因此进行指标名称映射为免费指标。

资源分析

监控指标展示



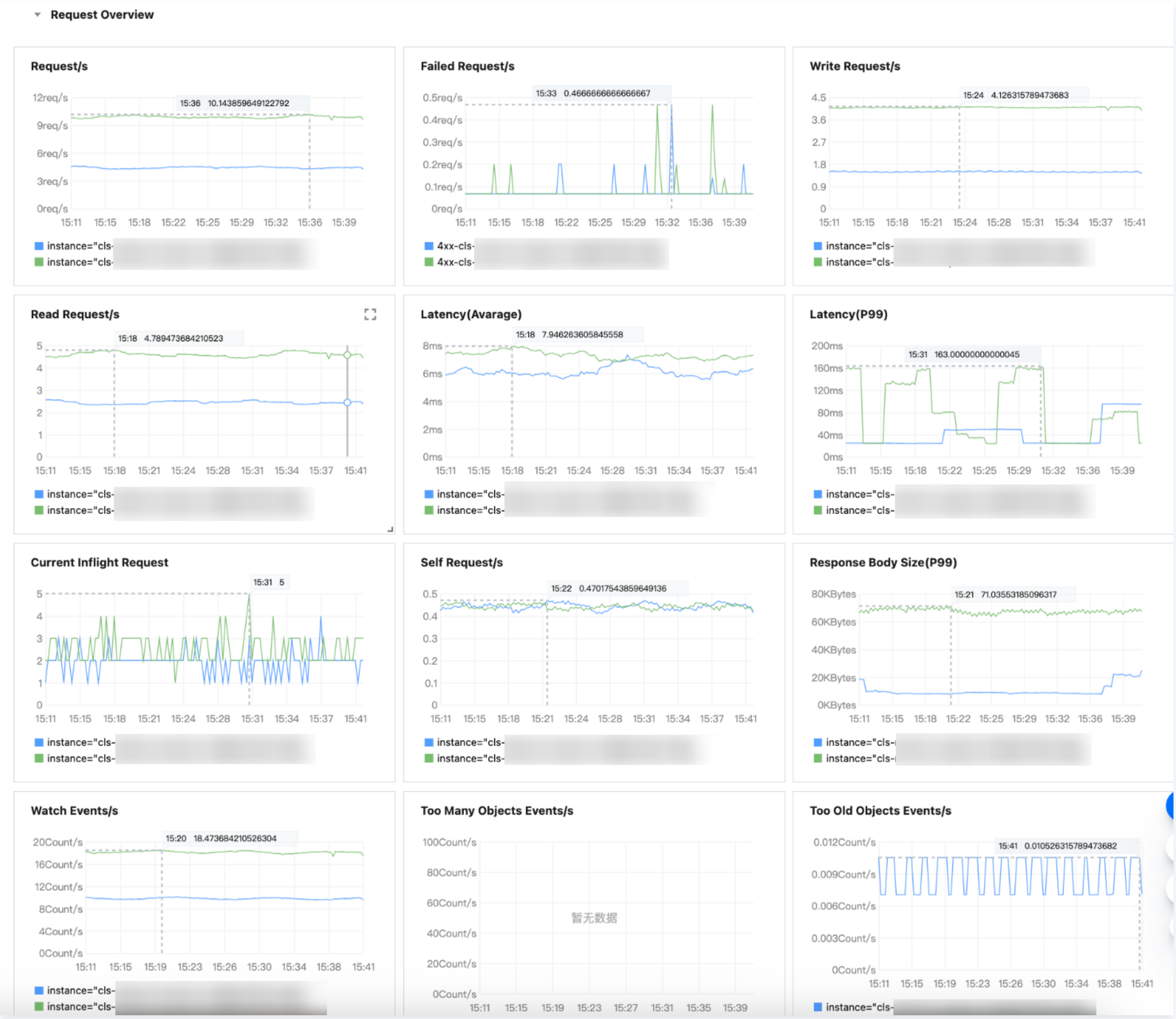
功能说明

指标	指标名称	使用指标	指标说明
Memory Usage	内存使用率	pod_mem_usage	API Server 的内存使用率。单位：%。
Cpu Usage	CPU 使用率	pod_core_usage	API Server 的 CPU 使用率。单位：%。
In Traffic	入流量	container_network_receive_bytes_total	API Server 的网络入流量。单位：Bit/s。
Out Traffic	出流量	container_network_transmit_packets_total	API Server 的网络出流量。单位：Bit/s。
Object Count	etcd 资源对象数	apiserver_storage_objects etcd_object_counts	资源对象数量。 • 当 TKE 为1.22及以上版本时，指标名字为 apiserver_storage_objects。 • 当 TKE 为1.22及以下版本时，指标名字为 etcd_object_counts。 说明：

			由于兼容性问题，1.22版本中 apiserver_storage_objects 名称和 etcd_object_counts 名称均存在。

请求概览

监控指标展示



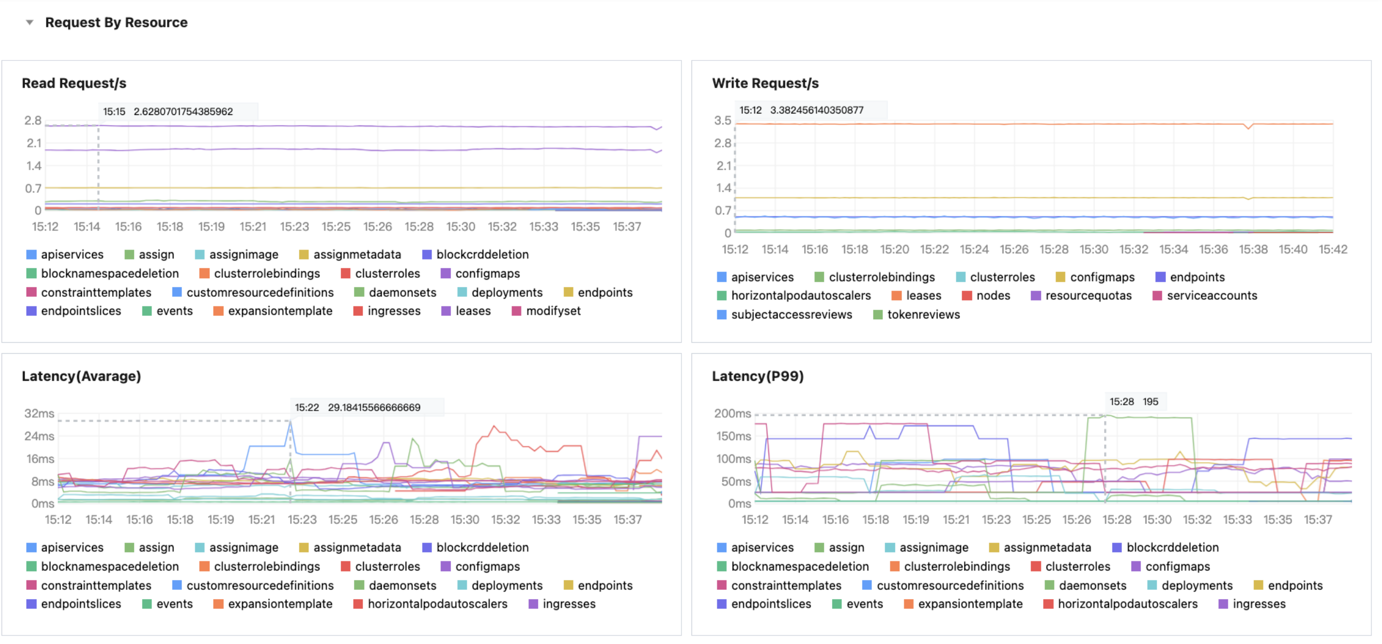
功能说明

指标	指标名称	使用指标	指标说明
Request/s	apiserver 每秒处理请求总量	apiserver_request_count apiserver_request_total	apiserver 每秒处理请求总量。单位：req/s。
Failed Request/s	apiserver 每秒返回失败请求数量	apiserver_request_count apiserver_request_total	apiserver 每秒返回失败请求数量。单位：req/s。
Write Request/s	apiserver 每秒写请求数量	apiserver_request_count apiserver_request_total	apiserver 每秒写请求数量。单位：req/s。

Read Request/s	apiserver 每秒读请求数量	apiserver_request_count apiserver_request_total	apiserver 每秒读请求数量。单位：req/s。
Latency(Average)	apiserver 访问平均延时	apiserver_request_latencies_summary_sum apiserver_request_latencies_summary_count	apiserver访问平均延时。单：毫秒。
Latency(P99)	apiserver 请求时延 P99	apiserver_request_duration_seconds_bucket	统计APIServer客户端对API Server不同请求的时延分布。单位：毫秒。
Current Inflight Request	apiserver Inflight Request	apiserver_current_inflight_requests	API Server当前处理的请求数量。
Self Request/s	apiserver selfrequest QPS	apiserver_selfrequest_total	apiserver selfrequest QPS。
Response Body Size(P99)	apiserver 回包大小 P99	apiserver_response_sizes_bucket	apiserver回包大小P99。单位：Bytes。
Watch Events/s	apiserver watch 事件推送QPS	apiserver_watch_events_total	资源对象总量。按照资源类型分类，例如 nodes，pods 等。单位Count/s。
Too Many Objects Events/s	list_too_many_objects_events_total	list_too_many_objects_events_total	list_too_many_objects 事件数量。单位Count/s。
Too Old Objects Events/s	watch_too_old_objects_events_total	watch_too_old_objects_events_total	watch_too_old_objects 事件数量。单位Count/s。

资源请求

监控指标展示



功能说明

指标	指标名称	使用指标	指标说明
Read Request/s	apiserver 每秒读请求数量	apiserver_request_count apiserver_request_total	apiserver 每秒读请求数量（按资源分类）。单位：req/s。

Write Request/s	apiserver 每秒写请求数量	apiserver_request_count apiserver_request_total	apiserver 每秒写请求数量（按资源分类）。单位：req/s。
Latency(Average)	apiserver 访问平均延时	apiserver_request_duration_seconds_sum apiserver_request_duration_seconds_count	apiserver 访问平均延时（按资源分类）。单位：毫秒。
Latency(P99)	apiserver 请求时延 P99	apiserver_request_duration_seconds_bucket	apiserver 请求时延P99（按资源分类）。单位：毫秒。

kube-scheduler 组件指标说明

最近更新时间：2025-03-26 11:18:42

前提条件

请参见 查看 TKE 集群控制面组件监控 。

指标列表

功能分类	指标	类型	说明
请求分析	scheduler_pending_pods	Gauge	Pending Pod 的数量。队列种类如下： • unschedulable：表示不可调度的 Pod 数量。 • backoff：表示 backoffQ 的 Pod 数量。 • active：表示 activeQ 的 Pod 数量。
请求分析	scheduler_pod_scheduling_attempts_bucket	Histogram	调度器在尝试将 Pod 调度到节点时的调度尝试次数。bucket 阈值为{1、2、4、8、16}。
资源分析	pod_core_usage	Gauge	CPU 使用率。单位：%。
资源分析	pod_mem_usage	Gauge	内存使用率。单位：%。
请求分析	rest_client_requests_total	Counter	从状态值（Status Code）、方法（Method）和主机（Host）维度分析 HTTP 请求数。
请求分析	rest_client_request_duration_seconds_bucket	Histogram	从方法（Verb）和 URL 维度分析 HTTP 请求时延。bucket 阈值{0.001,0.002,0.004,0.008,0.016,0.032,0.064,0.128,0.256,0.512}。单位：秒。
资源分析	container_network_receive_bytes_total	Counter	网络入流量。单位：Bit。 说明：原指标 k8s_pod_network_receive_bytes_bw 需收费，因此进行指标名称映射为免费指标。
资源分析	container_network_transmit_packets_total	Counter	网络出流量。单位：Bit。 说明：原指标 k8s_pod_network_transmit_bytes_bw 需收费，因此进行指标名称映射为免费指标。

资源分析

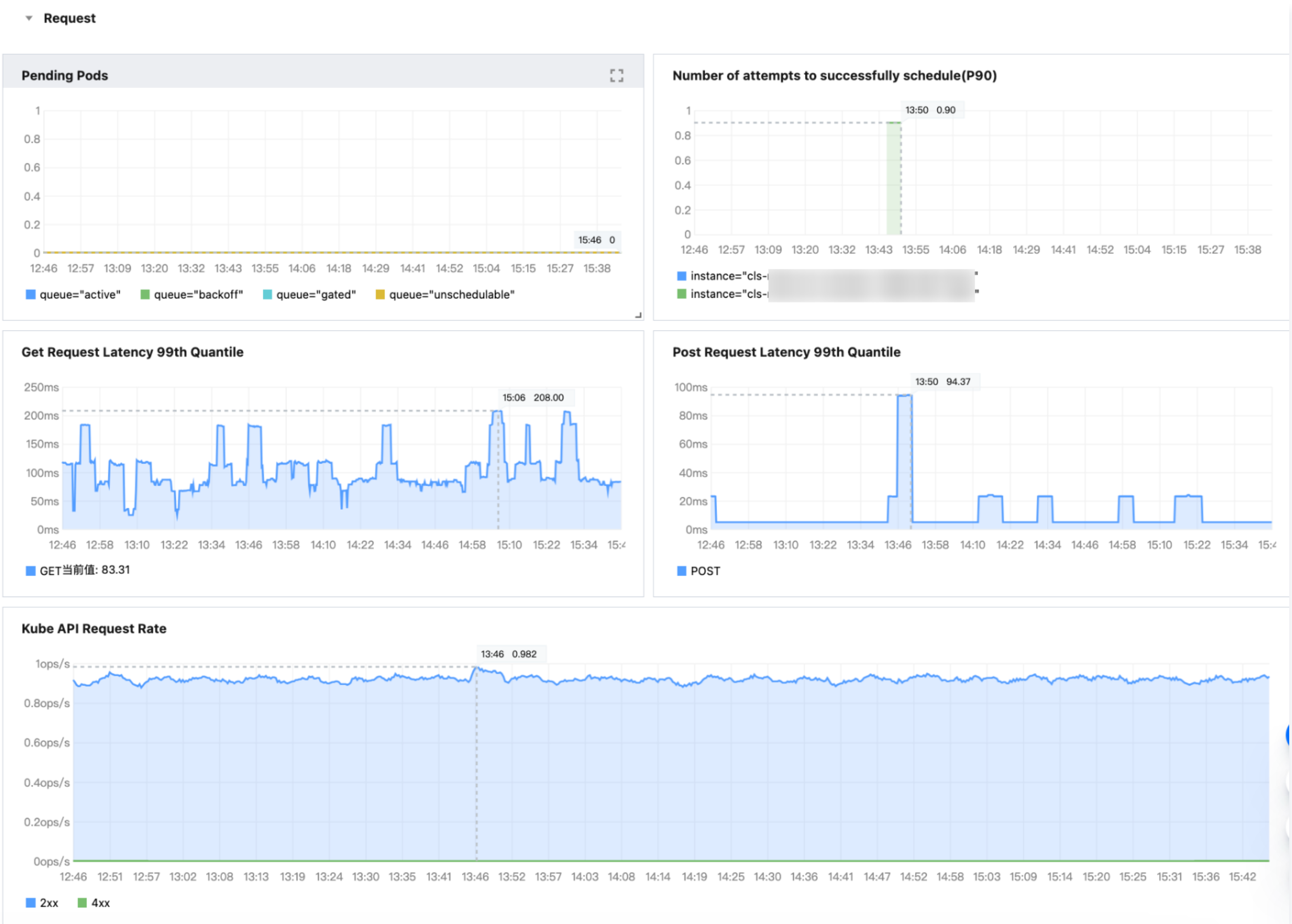
监控指标展示

功能说明

指标	指标名称	使用指标	指标说明
Memory Usage	内存使用率	pod_mem_usage	Scheduler 的内存使用率。单位：%。
Cpu Usage	CPU使用率	pod_core_usage	Scheduler 的 CPU 使用率。单位：%。
In Traffic	入流量	container_network_receive_bytes_total	Scheduler 的网络入流量。单位：Bit/s。
Out Traffic	出流量	container_network_transmit_packets_total	Scheduler 的网络出流量。单位：Bit/s。

请求分析

监控指标展示



功能说明

指标	指标名称	使用指标	指标说明
Pending Pods	Scheduler Pending Pods	scheduler_pending_pods	Pending Pod 的数量。
Number of attempts to successfully schedule(P90)	Scheduler 尝试成功调度 Pod 次数	scheduler_pod_scheduling_attempts_bucket	调度器在尝试将 Pod 调度到节点时的调度尝试次数。
Get Request Latency 99th Quantile	Get 请求P99延时	rest_client_request_duration_seconds_bucket	Get 请求P99延时。单位：毫秒。
Post Request Latency 99th Quantile	Post 请求P99延时	rest_client_request_duration_seconds_bucket	Post 请求P99延时。单位：毫秒。
Kube API Request Rate	Kube API 请求 QPS	rest_client_requests_total	从状态值（Status Code）、方法（Method）和主机（Host）维度分析 HTTP 请求数。单位：ops/s

kube-controller-manager 组件指标说明

最近更新时间：2025-03-27 11:23:23

前提条件

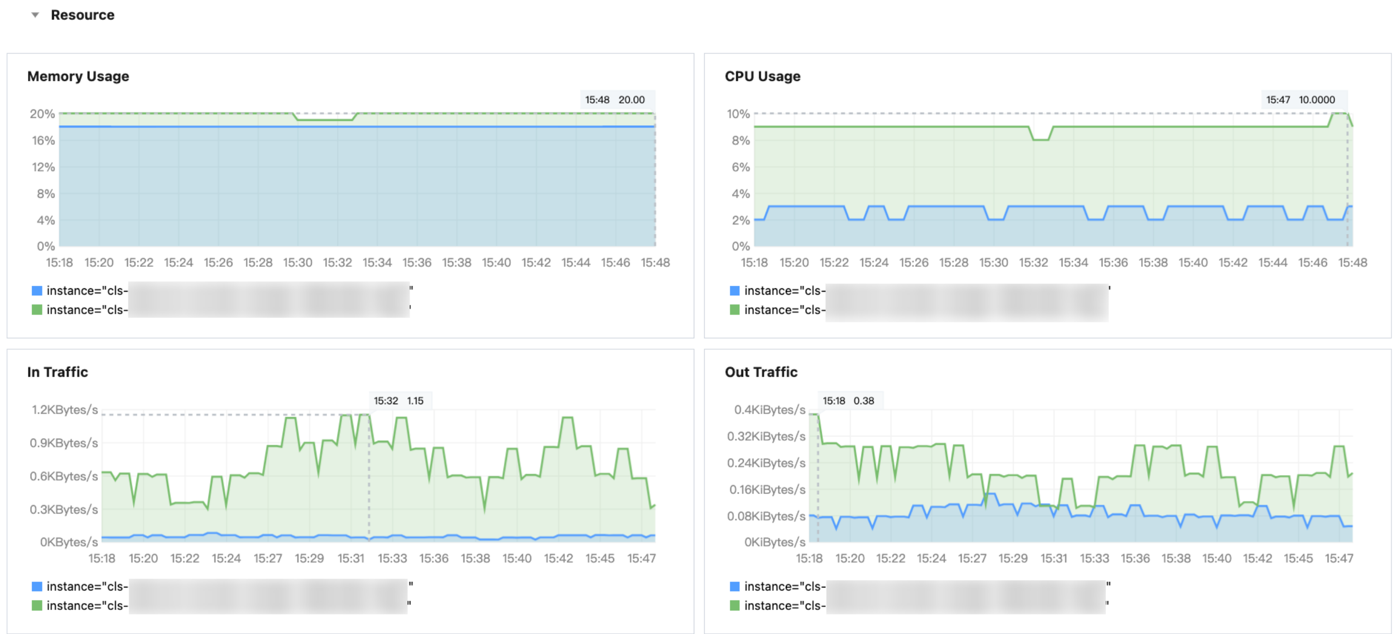
请参见 查看 TKE 集群控制面组件监控 。

指标列表

功能分类	指标	类型	说明
请求分析	workqueue_adds_total	Counter	Workqueue 处理的新增事件（Adds）数量。
请求分析	workqueue_depth	Gauge	Workqueue 当前队列深度。如果队列深度长时间保持在较高水平，表明 Controller 不能及时处理队列中的任务，导致任务堆积。
请求分析	workqueue_queue_duration_seconds_bucket	Histogram	任务在 Workqueue 中存在的时长。Bucket 阈值为{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10}。单位：秒。
资源分析	pod_core_usage	Gauge	CPU 使用率。 单位：%。
资源分析	pod_mem_usage	Gauge	内存使用率。 单位：%。
请求分析	rest_client_requests_total	Counter	从状态值（Status Code）、方法（Method）和主机（Host）维度分析 HTTP 请求次数。
请求分析	rest_client_request_duration_seconds_bucket	Histogram	从方法（Verb）和 URL 维度分析 HTTP 请求时延。
资源分析	container_network_receive_bytes_total	Counter	网络入流量。单位：Bit。 说明：原指标 k8s_pod_network_receive_bytes_bw 需收费，因此进行指标名称映射为免费指标。
资源分析	container_network_transmit_packets_total	Counter	网络出流量。单位：Bit。 说明：原指标 k8s_pod_network_transmit_bytes_bw 需收费，因此进行指标名称映射为免费指标。

资源分析

监控指标展示

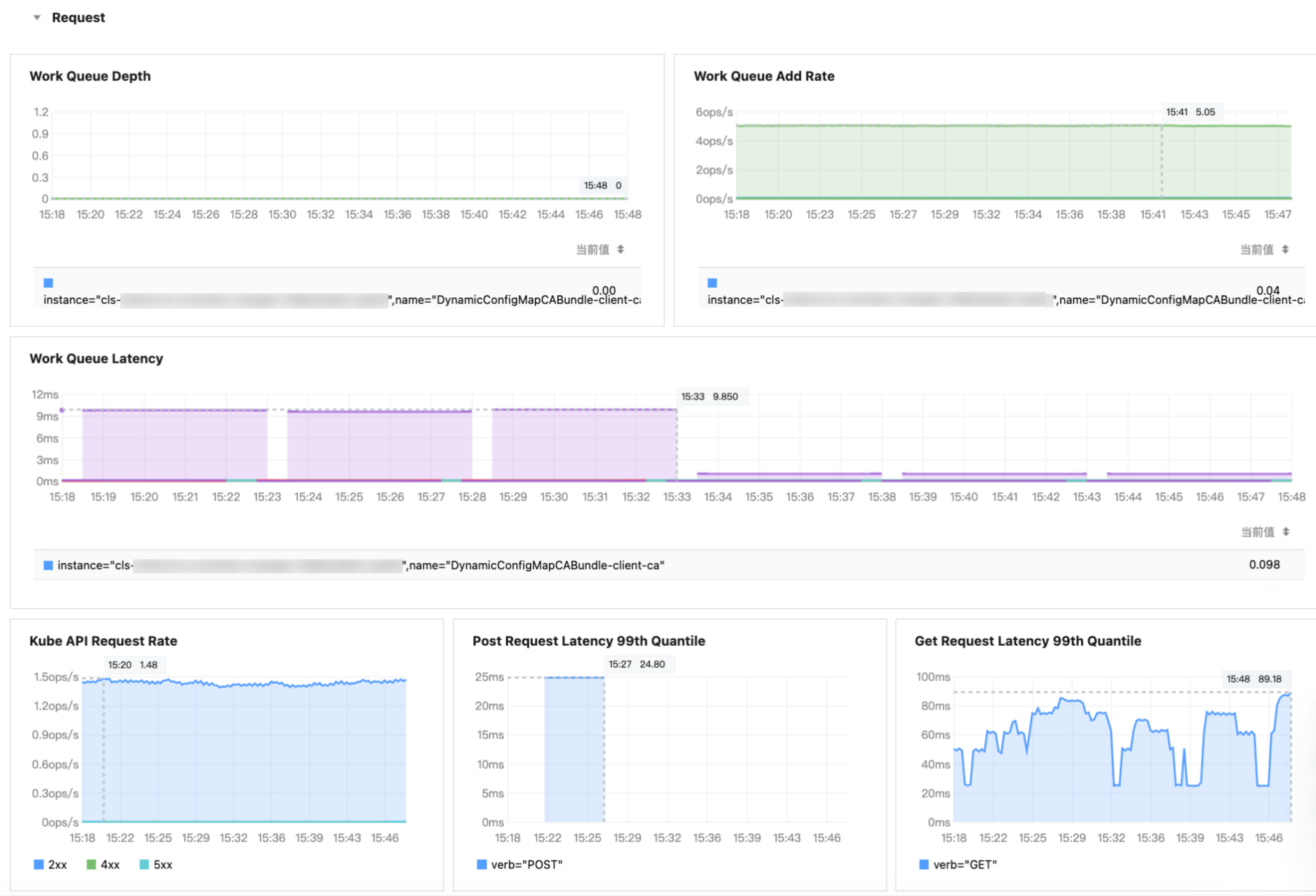


功能说明

指标	指标名称	使用指标	指标说明
Memory Usage	内存使用率	pod_mem_usage	Controller Manager 的内存使用率。单位：%。
Cpu Usage	CPU 使用率	pod_core_usage	Controller Manager 的 CPU 使用率。单位：%。
In Traffic	入流量	container_network_receive_bytes_total	Controller Manager 的网络入流量。单位：Bit/s。
Out Traffic	出流量	container_network_transmit_packets_total	Controller Manager 的网络出流量。单位：Bit/s。

请求分析

监控指标展示



功能说明

指标	指标名称	使用指标	指标说明
Work Queue Depth	Work Queue深度	workqueue_depth	Workqueue 深度在单位时间内的变化。
Work Queue Add Rate	Work Queue 增长率	workqueue_adds_total	Workqueue 在单位时间内新增事件（Adds）的数量。单位：ops/s。
Work Queue Latency	Work Queue 延迟	workqueue_queue_duration_seconds_bucket	事件在 Work Queue 中存在的时长。单位：毫秒。
Kube API Request Rate	Kube API 请求QPS	rest_client_requests_total	从状态值（Status Code）、方法（Method）和主机（Host）维度分析 HTTP 请求数。单位：ops/s。
Post Request Latency 99th Quantile	Post 请求P99延时	rest_client_request_duration_seconds_bucket	Post 请求P99延时。单位：毫秒。
Get Request Latency 99th Quantile	Get 请求P99延时	rest_client_request_duration_seconds_bucket	Get 请求P99延时。单位：毫秒。

用户自建 Prometheus 采集控制面监控

最近更新时间：2025-03-19 14:15:42

本文介绍如何基于用户自建的 Prometheus，采集 TKE 托管集群的控制平面组件监控 APIServer、Scheduler、KCM 指标配置。

前提条件

- 自建的 Prometheus 能够访问 TKE 集群的 API Server。
- 自建的 Prometheus 可以部署在 TKE 集群内，也可以部署在 TKE 集群外。

Prometheus 采集配置

使用自建的 Prometheus 采集 TKE 集群控制面核心组件指标时，首先需要在 Prometheus 的配置文件 prometheus.yaml 中配置指标采集 Job。配置文件格式如下：

```
global:
  scrape_interval:     15s # By default, scrape targets every 15 seconds.


# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: kube-apiserver
    ......

  - job_name: kube-controller-manager
    ......

  - job_name: kube-scheduler
    ......
```

其中，每个核心组件对应一个 Job 配置，具体配置可参见对应核心组件的指标列表。社区 Prometheus 配置 Prometheus.yaml 方法，请参见 Configuration 。

TKE 集群内监控采集

内监控是将 Prometheus 部署在待监控的 TKE 集群内的监控形式。

kube-apiserver

关于 kube-apiserver 组件的更多信息，请参见 kube-apiserver 组件指标说明 。

```
scrape_configs:
- job_name: kube-apiserver
  honor_timestamps: true
  params:
    component:
    - apiserver
  scrape_interval: 15s
  metrics_path: /master/metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  relabel_configs:
  - source_labels: [job]
    separator: ;
    regex: (.*)
    target_label: __tmp_prometheus_job_name
    replacement: $1
```

```
    action: replace
  - source_labels: [__meta_kubernetes_service_label_app_kubernetes_io_managed_by,
      __meta_kubernetes_service_labelpresent_app_kubernetes_io_managed_by]
    separator: ;
    regex: (Helm);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_app,
__meta_kubernetes_service_labelpresent_label_qcloud_app]
    separator: ;
    regex: (cluster-monitor);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_service,
__meta_kubernetes_service_labelpresent_label_qcloud_service]
    separator: ;
    regex: (master-metrics-service);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_port_name]
    separator: ;
    regex: http-metrics
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
    separator: ;
    regex: Node;(.*)
    target_label: node
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
    separator: ;
    regex: Pod;(.*)
    target_label: pod
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_namespace]
    separator: ;
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_service_name]
    separator: ;
    regex: (.*)
    target_label: service
    replacement: $1
    action: replace
  - separator: ;
    regex: (.*)
    target_label: endpoint
    replacement: http-metrics
    action: replace
  metric_relabel_configs:
  - source_labels: [pod]
    separator: ;
    regex: (.*)
    target_label: instance
    replacement: $1
```



```
    action: replace
  kubernetes_sd_configs:
  - role: endpoints
    kubeconfig_file: ""
    follow_redirects: true
    enable_http2: true
    namespaces:
      own_namespace: false
      names:
      - kube-system
```

kube-scheduler

关于 kube-scheduler 组件的更多信息，请参见 kube-scheduler 组件指标说明 。

```
scrape_configs:
- job_name: kube-scheduler
  honor_labels: true
  honor_timestamps: true
  params:
    component:
    - scheduler
  scrape_interval: 30s
  scrape_timeout: 10s
  metrics_path: /master/metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  relabel_configs:
  - source_labels: [job]
    separator: ;
    regex: (.*)
    target_label: __tmp_prometheus_job_name
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_service_label_app_kubernetes_io_managed_by,
      __meta_kubernetes_service_labelpresent_app_kubernetes_io_managed_by]
    separator: ;
    regex: (Helm);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_app,
__meta_kubernetes_service_labelpresent_label_qcloud_app]
    separator: ;
    regex: (cluster-monitor);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_service,
__meta_kubernetes_service_labelpresent_label_qcloud_service]
    separator: ;
    regex: (master-metrics-service);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_port_name]
    separator: ;
    regex: http-metrics
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
```



```
    separator: ;
    regex: Node;(.*)
    target_label: node
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
    separator: ;
    regex: Pod;(.*)
    target_label: pod
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_namespace]
    separator: ;
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_service_name]
    separator: ;
    regex: (.*)
    target_label: service
    replacement: $1
    action: replace
  - separator: ;
    regex: (.*)
    target_label: endpoint
    replacement: http-metrics
    action: replace
  metric_relabel_configs:
  - source_labels: [pod]
    separator: ;
    regex: (.*)
    target_label: instance
    replacement: $1
    action: replace
  kubernetes_sd_configs:
  - role: endpoints
    kubeconfig_file: ""
    follow_redirects: true
    enable_http2: true
    namespaces:
      own_namespace: false
      names:
      - kube-system
```

kube-controller-manager

关于 kube-controller-manager 组件的更多信息，请参见 kube-controller-manager 组件指标说明 。

```
scrape_configs:
- job_name: kube-controller-manager
  honor_labels: true
  honor_timestamps: true
  params:
    component:
    - controller-manager
  scrape_interval: 30s
  scrape_timeout: 10s
  metrics_path: /master/metrics
```

```
  scheme: http
  follow_redirects: true
  enable_http2: true
  relabel_configs:
  - source_labels: [job]
    separator: ;
    regex: (.*)
    target_label: __tmp_prometheus_job_name
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_service_label_app_kubernetes_io_managed_by,
      __meta_kubernetes_service_labelpresent_app_kubernetes_io_managed_by]
    separator: ;
    regex: (Helm);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_app,
__meta_kubernetes_service_labelpresent_label_qcloud_app]
    separator: ;
    regex: (cluster-monitor);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_service_label_label_qcloud_service,
__meta_kubernetes_service_labelpresent_label_qcloud_service]
    separator: ;
    regex: (master-metrics-service);true
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_port_name]
    separator: ;
    regex: http-metrics
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
    separator: ;
    regex: Node;(.*)
    target_label: node
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_endpoint_address_target_kind,
__meta_kubernetes_endpoint_address_target_name]
    separator: ;
    regex: Pod;(.*)
    target_label: pod
    replacement: ${1}
    action: replace
  - source_labels: [__meta_kubernetes_namespace]
    separator: ;
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_service_name]
    separator: ;
    regex: (.*)
    target_label: service
    replacement: $1
    action: replace
  - separator: ;
    regex: (.*)
```

```
    target_label: endpoint
    replacement: http-metrics
    action: replace
  metric_relabel_configs:
  - source_labels: [pod]
    separator: ;
    regex: (.*)
    target_label: instance
    replacement: $1
    action: replace
  kubernetes_sd_configs:
  - role: endpoints
    kubeconfig_file: ""
    follow_redirects: true
    enable_http2: true
    namespaces:
      own_namespace: false
      names:
      - kube-system
```

通过 ServiceMonitor 进行采集

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: master-metrics-exporter
  namespace: kube-system
spec:
  endpoints:
  - honorLabels: true
    interval: 15s
    metricRelabelings:
    - action: replace
      sourceLabels:
      - pod
      targetLabel: instance
    params:
      component:
      - apiserver
    path: /master/metrics
    port: http-metrics
    relabelings:
    - action: replace
      replacement: kube-apiserver
      targetLabel: job
  - honorLabels: true
    interval: 15s
    metricRelabelings:
    - action: replace
      sourceLabels:
      - pod
      targetLabel: instance
    params:
      component:
      - controller-manager
    path: /master/metrics
    port: http-metrics
    relabelings:
    - action: replace
      replacement: kube-controller-manager
```

```yaml
      targetLabel: job
  - honorLabels: true
    interval: 15s
    metricRelabelings:
    - action: replace
      sourceLabels:
      - pod
      targetLabel: instance
    params:
      component:
      - scheduler
    path: /master/metrics
    port: http-metrics
    relabelings:
    - action: replace
      replacement: kube-scheduler
      targetLabel: job
  namespaceSelector: {}
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: Helm
      label_qcloud_app: cluster-monitor
      label_qcloud_service: master-metrics-service
```

TKE 集群外监控采集

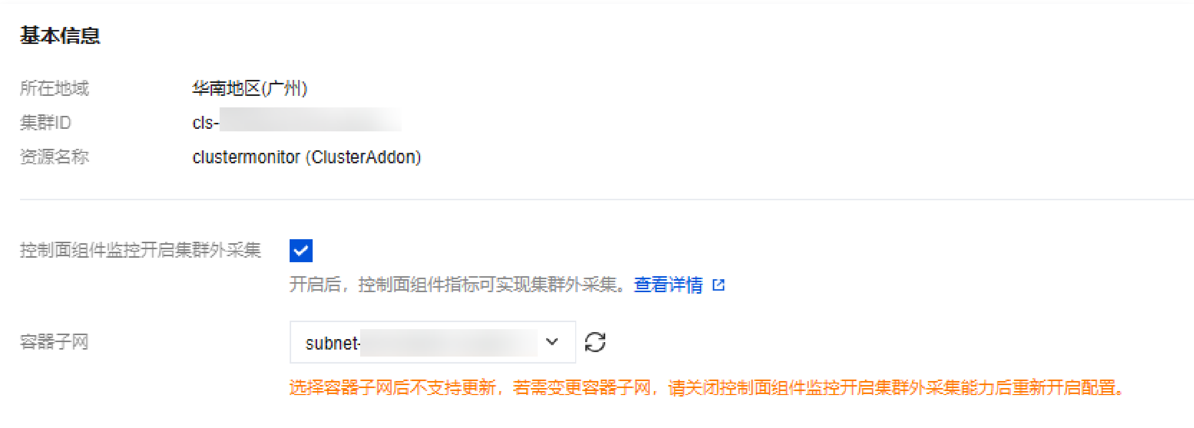
本文以同 VPC 下集群外采集为示例。

前提条件

被采集集群已开启公网或内网访问。

开启跨集群指标采集

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**组件管理**。
4. 单击 clustermonitor 组件的**更新配置**。
5. 在更新组件配置页面，**勾选开启**控制面组件监控开启集群外采集，选择容器子网。



> **说明：**
> 选择容器子网后不支持更新，若需变更容器子网，请关闭控制面组件监控开启集群外采集能力后重新开启配置。

6. 单击**完成**。

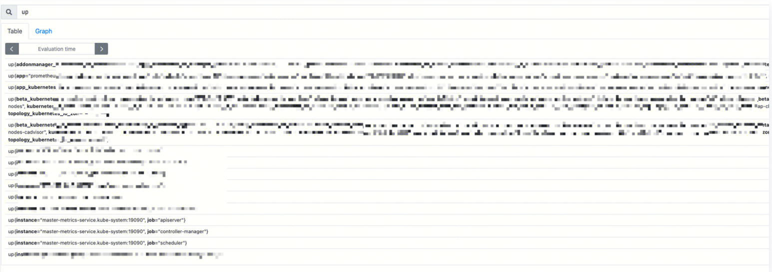
采集配置

具体操作请参见 Configuration 和 Monitoring kubernetes with prometheus from outside of k8s cluster，与集群内配置相比，主要修改 kubernetes_sd_configs 配置，配置方式如下：

```
scrape_configs:
- job_name: out-tke-apiserver
  kubernetes_sd_configs:
  - role: endpoints
    kubeconfig_file: ""
    follow_redirects: true
    enable_http2: true
    namespaces:
      names:
      - kube-system
    namespaces:
      names:
      - kube-system
    api_server: 'https://<KUBERNETES URL>'
    tls_config:
      ca_file: /etc/prometheus/kubernetes-ca.crt
    bearer_token: '<SERVICE ACCOUNT BEARER TOKEN>'
  scrape_interval: 30s
```

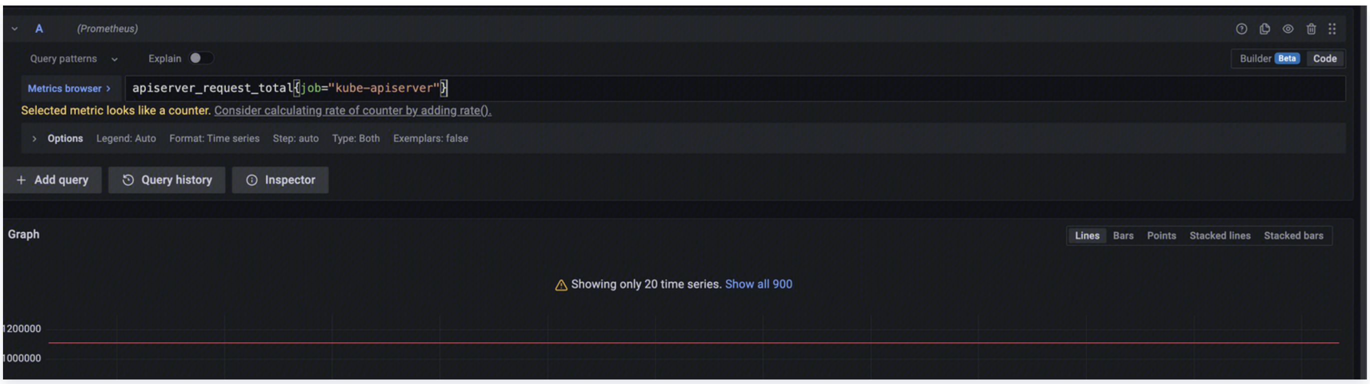
验证

1. 登录自建的 Prometheus 控制台，切换到 Graph 页面。
2. 输入 `up`，查看是否全部控制面组件都可以显示。



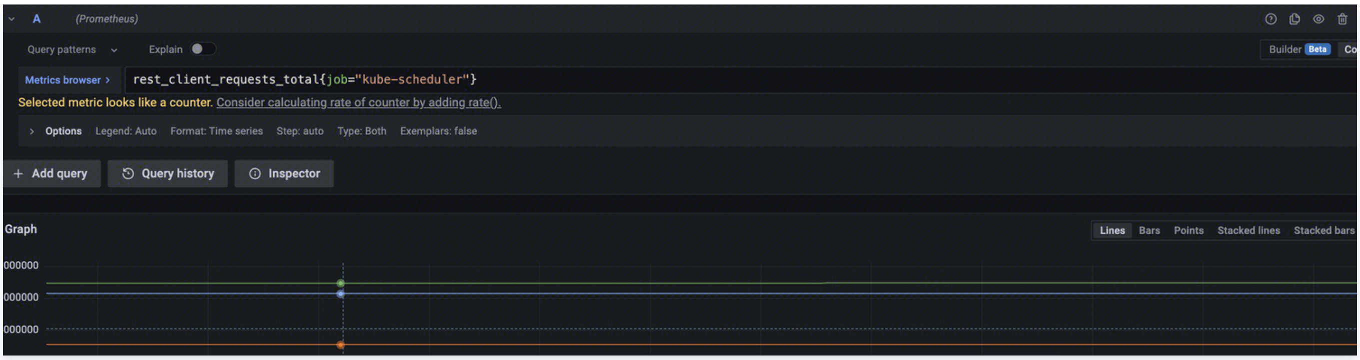
3. 输入以下指令，验证 API Server 是否正常：

```
apiserver_request_total{job="apiserver"}
```



4. 输入以下指令，验证 Scheduler 是否正常：

```
rest_client_requests_total{job="scheduler"}
```



5. 输入以下指令，验证 Controller Manager 是否正常：

```
rest_client_requests_total{job="controller-manager"}
```

应用性能管理

最近更新时间：2025-05-21 14:34:42

背景

随着企业越来越多地采用容器化和微服务架构，应用的复杂性显著增加，性能管理变得尤为重要。在 Kubernetes 中，应用性能管理（APM）涉及监控、分析和优化应用的资源使用、响应时间和可用性。有效的性能管理可以提高应用响应速度、优化资源利用率、减少停机时间、提升用户体验。

前提条件

登录 腾讯云可观测平台 控制台，创建业务系统管理。详情请参见 新建业务系统 。

接入方案选型

APM 支持 OpenTelemetry、Skywalking 等多种上报协议。OpenTelemetry 社区活跃度高，技术更迭迅速，广泛兼容主流编程语言、组件与框架，请优先选择 OpenTelemetry 方案，关于 OpenTelemetry 的更多信息请参考 OpenTelemetry 官方网站 。
对于部署在 Kubernetes 的应用，请优先选择 Operator 模式自动接入，节省安装探针和修改配置方面的工作量，关于 Operator 模式的更多信息请参考 安装 tencent-opentelemetry-operator 。

接入方式

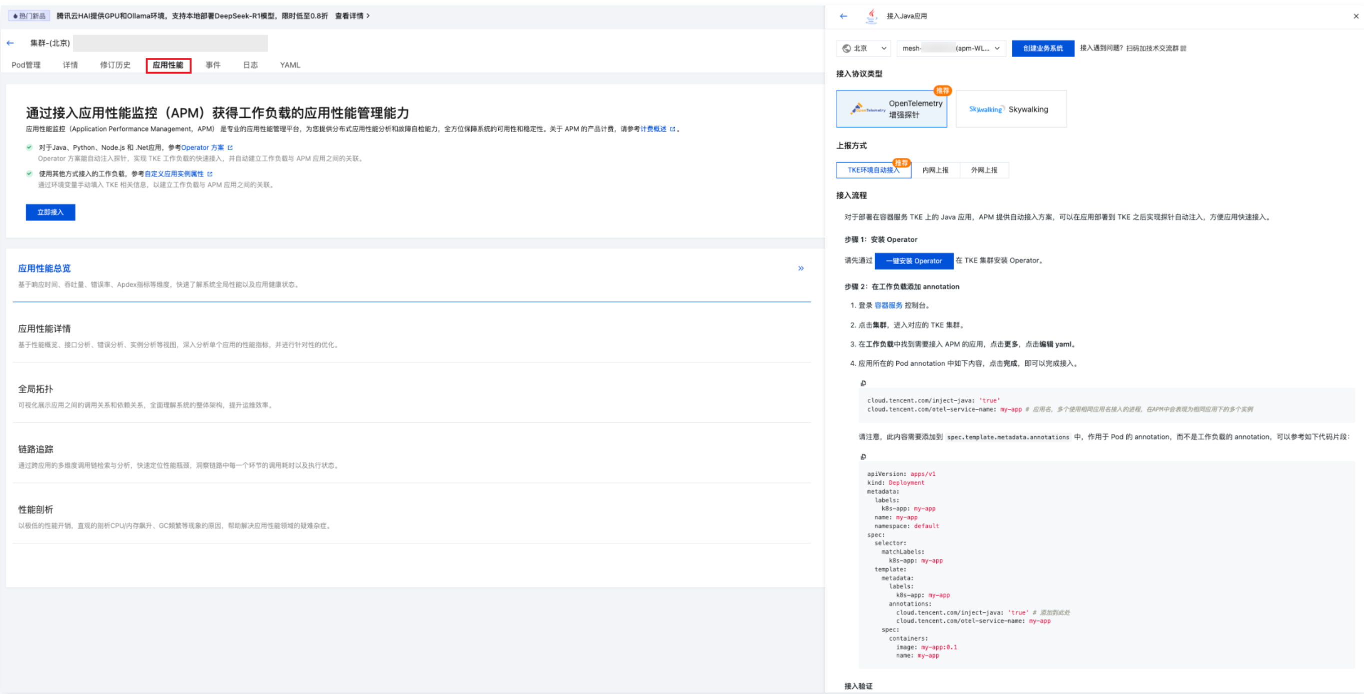
不同编程语言的推荐接入方案请参见 接入方式 。

Operator 方案

tencent-opentelemetry-operator 是腾讯云基于社区 opentelemetry-operator 构建的探针接入方案。由于实现了 Kubernetes 环境下探针自动注入机制，免去了探针部署和配置的工作量，让应用接入变得更加简单。目前 tencent-opentelemetry-operator 支持 Java、Python、node.js、.Net 应用的快速接入，应用程序部署到腾讯云容器服务 TKE 以后，只需要在工作负载的 YAML 文件中添加2行 annotation，就能完成整个接入流程，无需在开发态涉及任何代码的修改。

快速接入应用性能监控（APM）

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群**页面，单击目标集群 ID，进入集群详情页。
3. 选择左侧导航中的**工作负载**。
4. 单击进入需要监控的工作负载，选择**应用性能**，单击**立即接入**，并选择应用类型快速接入应用性能监控。



5. 单击**一键安装 Operator**，在 TKE 集群安装 Operator。

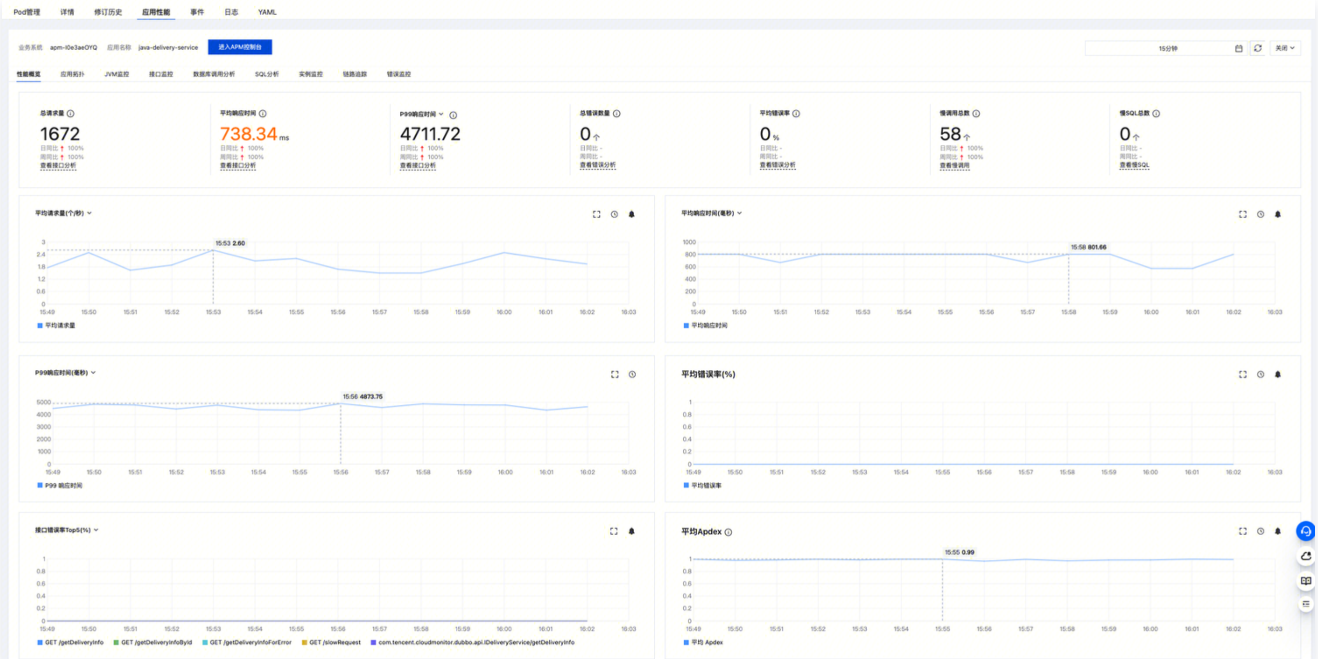
6. 在**一键安装 Operator** 页面，通过下拉框选择上报地域、默认业务系统、TKE 集群等信息，确认后单击**安装**。

> **说明：**
> 上报地域为 APM 业务系统所在的地域，每个 TKE 集群只能指定唯一的上报地域，建议将上报地域与 TKE 集群所在的地域保持一致。

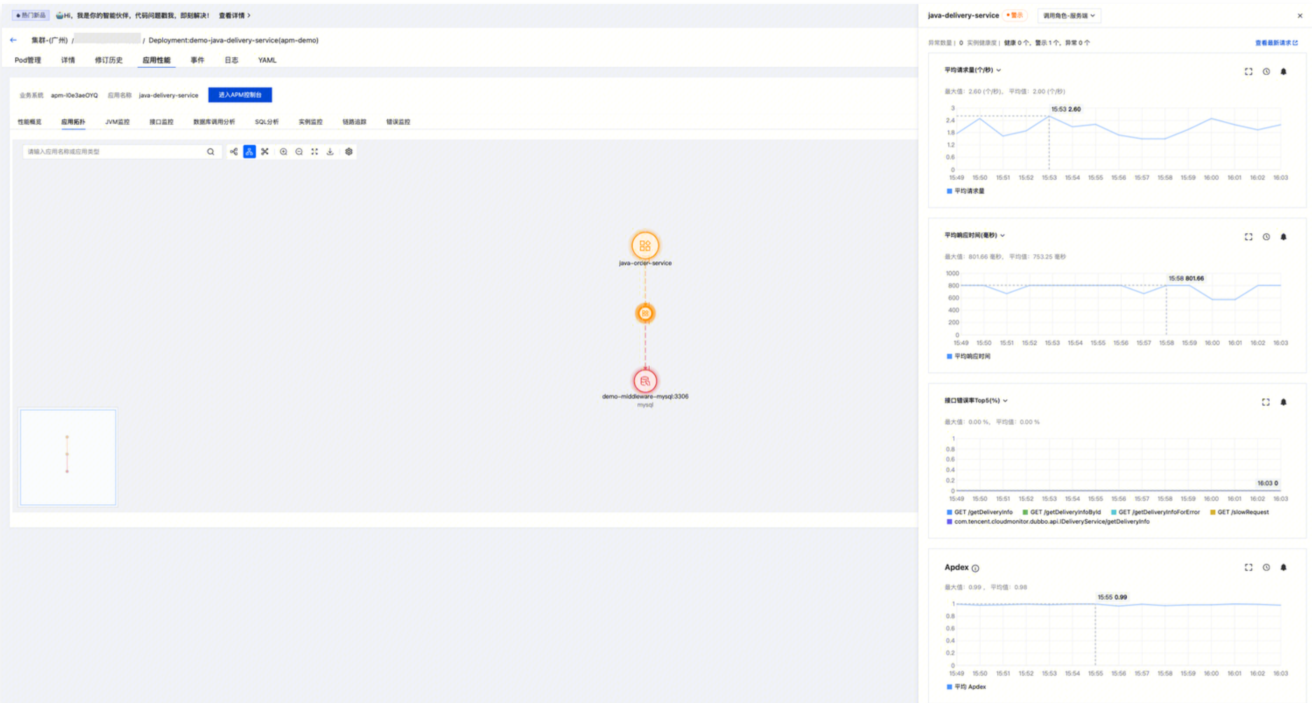
7. 根据应用类型参考以下文档进行后续的操作步骤。

应用类型	OpenTelemetry 方案
Go	通过 OpenTelemetry-Go 接入 Go 应用（推荐）
Java	K8s 环境自动接入 Java 应用 （推荐）
Python	K8s 环境自动接入 Python 应用 （推荐）
PHP	通过 OpenTelemetry-PHP 接入 PHP 应用（推荐）
Node.js	K8s 环境自动接入 Node.js 应用 （推荐） 通过 OpenTelemetry-JS 方案接入 Node.js 应用（推荐）
.Net	K8s 环境自动接入 .NET 应用 （推荐）
其他语言	接入其他语言编写的应用

8. 以 Java 应用为例，可查看丰富的应用指标：
   - 性能概览

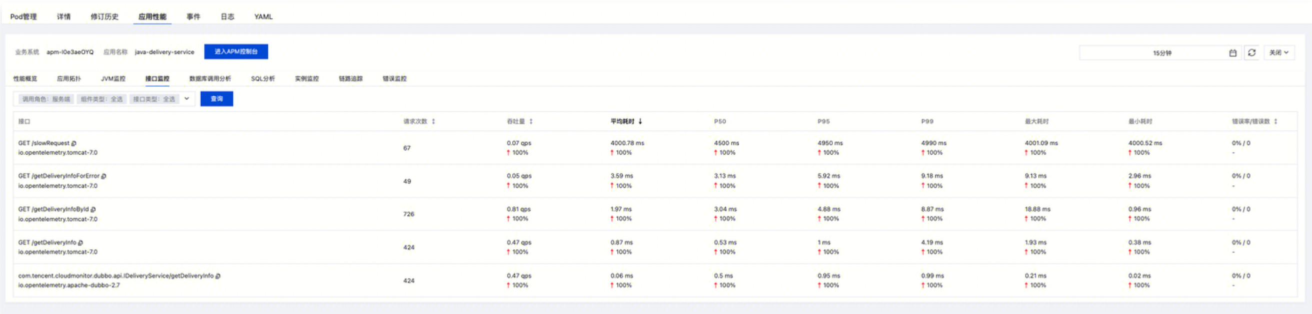


- 应用拓扑

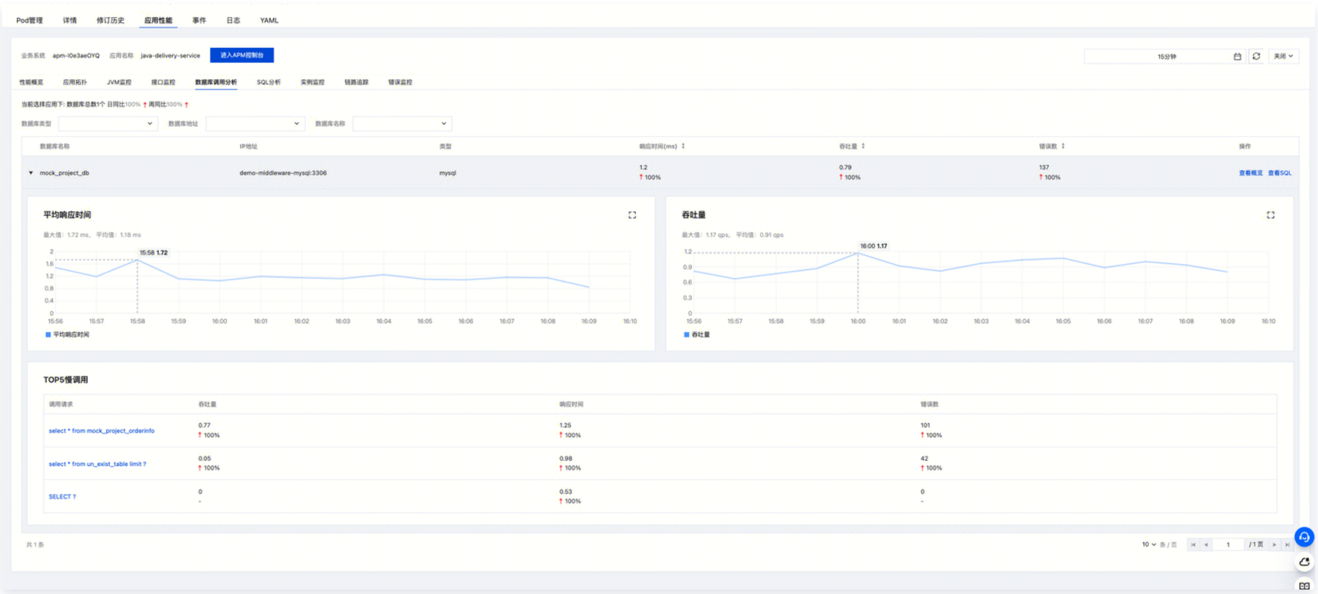


- JVM 监控

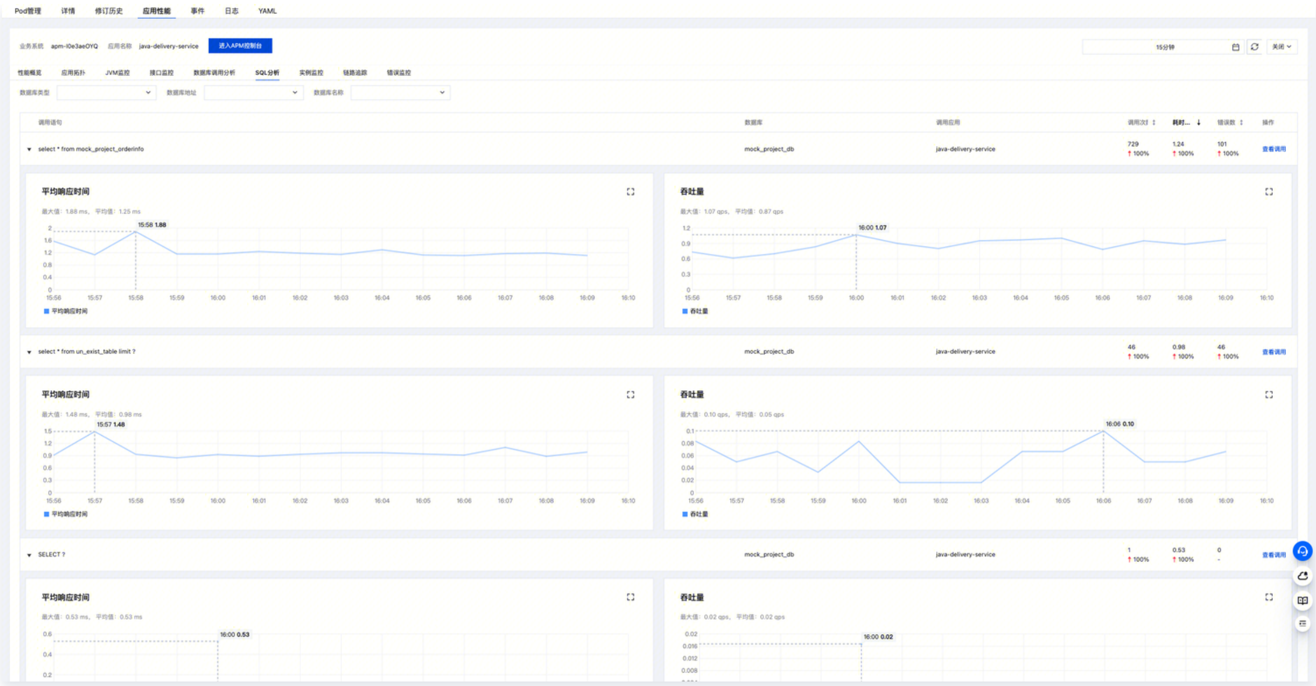
- 接口监控



- 数据库调用分析

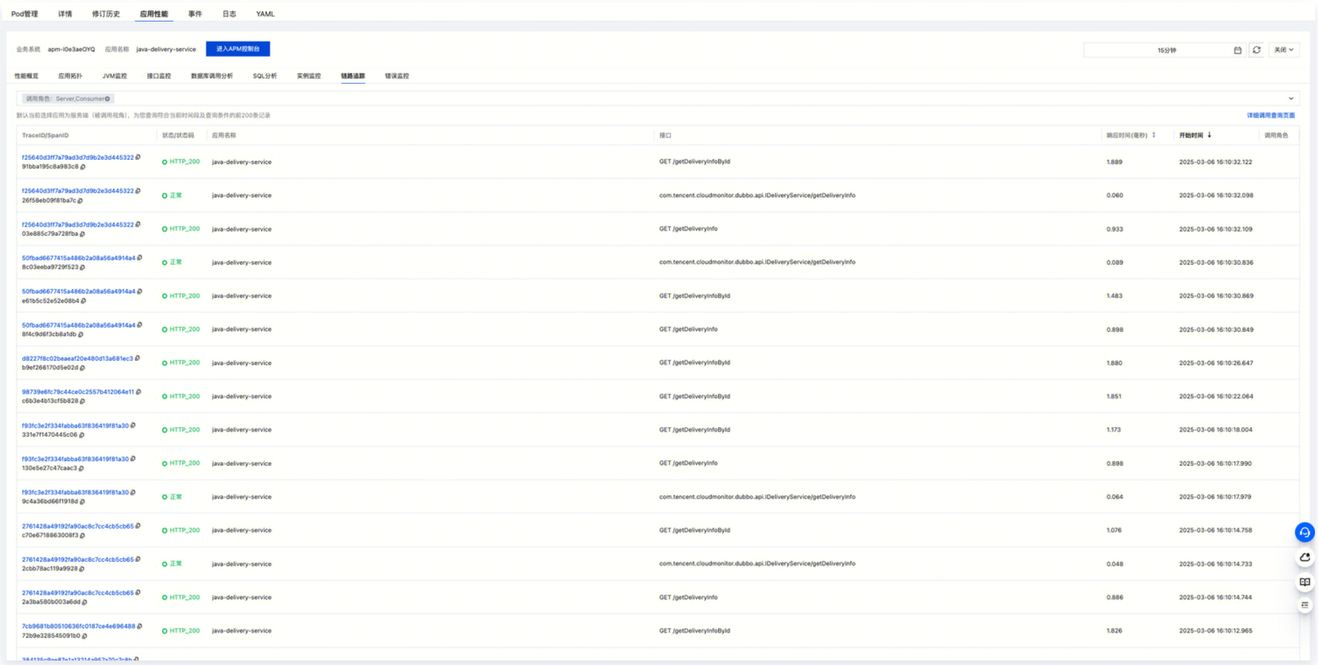


- SQL 分析

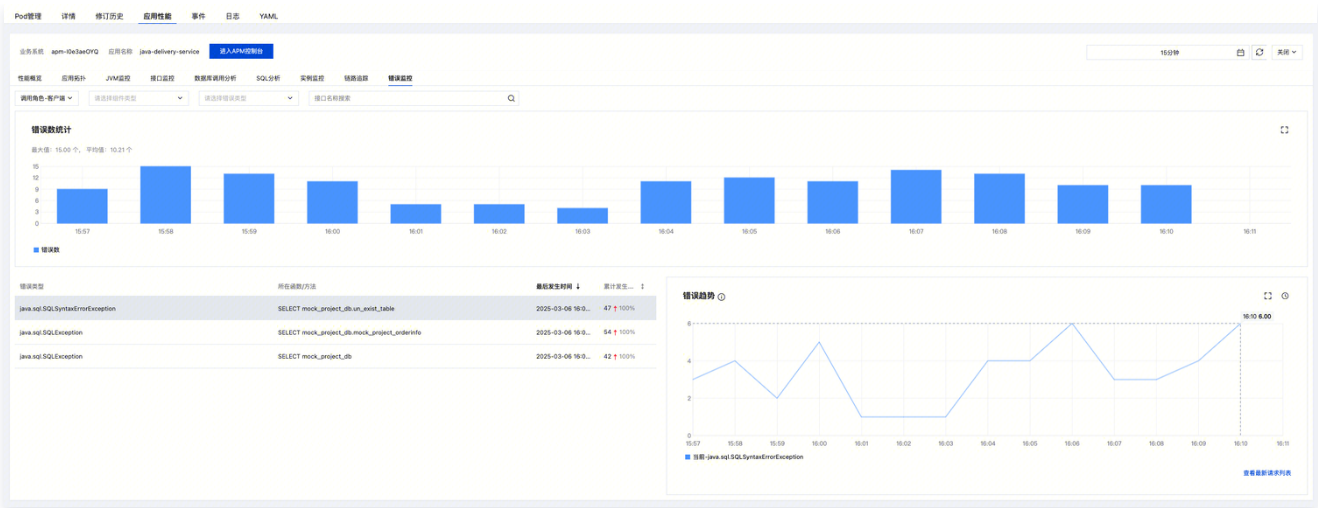


- 实例监控

- 链路追踪

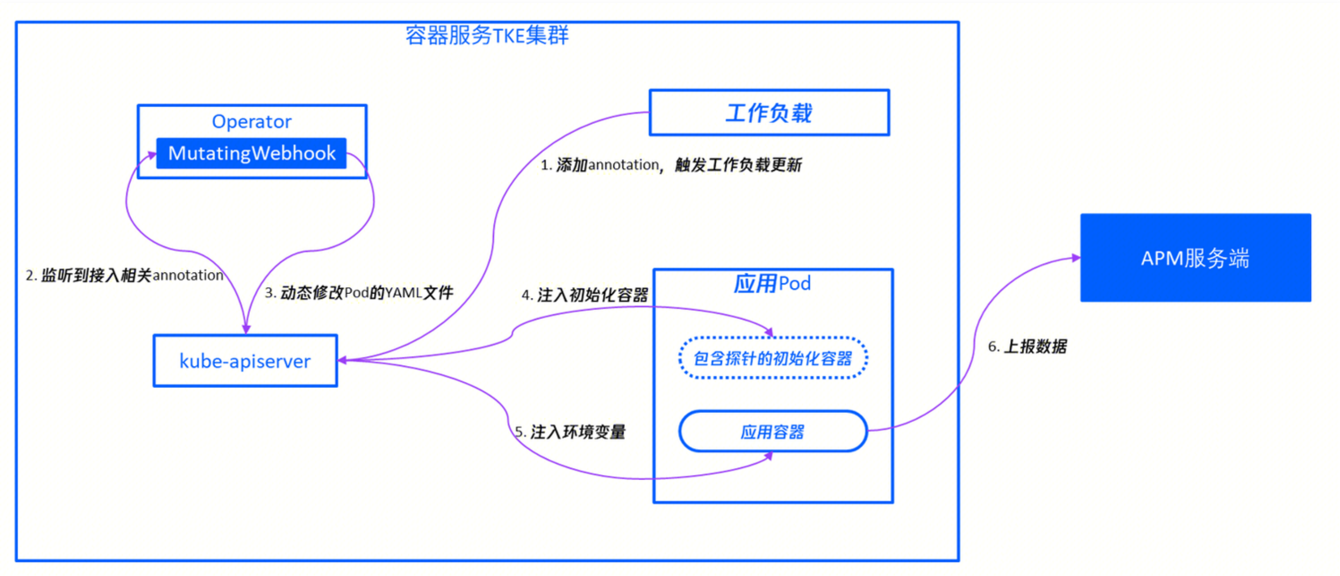


- 错误监控



接入原理

Operator 方案利用了 Kubernetes 提供的 Dynamic Admission Control 机制实现探针的注入，可以参考以下流程了解 Operator 方案的接入原理：



1. 用户对工作负载添加 annotation，API Server 将基于工作负载的发布策略启动工作负载的更新。
2. 在 Operator 的相关组件中，包含一个具备 Dynamic Admission Control 能力的工作负载，通过 Mutating Webhook 的形式在容器服务集群中运行，这个组件可以从 API Server 监听工作负载的变化。
3. Operator 监听到应用工作负载添加了接入相关的 annotation 之后，会通过 API Server 对即将创建的应用 Pod 进行动态修改，修改的内容通过 Pod 的 YAML 描述表现出来。
4. Operator 会在应用 Pod 中注入一个初始化容器，在容器本地路径中包含了探针文件。
5. Operator 还会在应用容器中注入一些环境变量，这些环境变量可以对应用进程进行配置，实现挂载探针的动作。以 Java 应用为例，Operator 会对应用容器添加 JAVA_TOOL_OPTIONS 环境变量，其中包括 -javaagent 参数，指向由初始化容器引入的探针文件本地路径，使 Java 进程启动的时候执行挂载探针操作。
6. 探针通过字节码增强机制自动采集监控数据，并向 APM 服务端上报。

接入优势

- **简化 Operator 安装流程**：安装部署流程，不但上架了 TKE 应用市场，还能在 APM 控制台实现 Operator 一键安装。
- **通过封装减少人力**：对接入过程中需要用到的一系列配置项进行了封装，不再需要手工填写，包括 APM 接入点的地址，业务系统 token 等。
- **优化证书管理机制**：解决了 Operator 安装过程中有可能遇到的各类兼容性问题。
- **降级处理**：当 APM 平台的任何一个组件出现故障的时候，整个接入机制都以不影响业务正常运行为前提，在必要的时候对探针注入和监控数据上报逻辑行降级处理，确保业务的稳定运行。

自定义埋点

在探针接入方案的基础上，Operator 模式通过探针注入进一步简化了应用接入 APM 的流程。tencent-opentelemetry-operator 注入的探针来自 OpenTelemetry 生态，对主流的开发框架与类库实现了运行态自动埋点。对大多数用户而言，应用在接入成功后即可完成监控数据上报，实现分布式链路追踪，不需要修改任何代码。但在某些特殊的场景中，如果探针的自动埋点机制不能满足用户需求，用户可以引入 OpenTelemetry API，在自动埋点的基础上，通过修改应用代码增加自定义埋点。例如下述场景中，可能需要引入自定义埋点：

- 应用引入的开发框架与类库没有在开发者中被广泛使用，或者版本很低。
- 应用引入的开发框架与类库由用户自己封装，或者来自于闭源商业化定制开发。
- 对于某些核心的自定义方法，需要在链路中追加方法级的埋点。

对于 tencent-opentelemetry-operator 注入的探针，可以参考 OpenTelemetry 方案支持的组件和框架 了解自动埋点的范围。自定义埋点代码的具体编写方式，可以参考如下 OpenTelemetry 的官方文档：

应用	参考链接
Java	https://opentelemetry.io/docs/languages/java/instrumentation/
Python	https://opentelemetry.io/docs/languages/python/instrumentation/
Node.js	https://opentelemetry.io/docs/languages/js/instrumentation/

以 Java 语言为例，如果我们需要对某个自定义方法在链路中追加方法级埋点，可以参考以下步骤实现。

1. 引入 OpenTelemetry API 依赖：

```
<dependencies>
    <!-- 其他依赖 -->
    <dependency>
        <groupId>io.opentelemetry</groupId>
        <artifactId>opentelemetry-api</artifactId>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.opentelemetry</groupId>
            <artifactId>opentelemetry-bom</artifactId>
            <version>1.9.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

2. 通过如下代码，在链路中增加 doTask() 的方法级埋点。

```
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.StatusCode;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.context.Scope;

// Trace 对象可以在业务方法中获取，或者通过参数传入业务方法
public void doTask(Tracer tracer) {
    // 创建一个 Span
    Span span = tracer.spanBuilder("doTask").startSpan();
    // 在 Span 中添加一些 Attributes
    span.setAttribute("RequestId", "5fc92ff1-8ca8-45f4-8013-24b4b5257666");
    // 将此 Span 设置为当前的Span
    try (Scope scope = span.makeCurrent()) {
        doSubTask1();
        doSubTask2();
    } catch (Throwable t) {
        // 处理异常，异常信息将记录到 Span 的对应事件中
```

```
            span.recordException(t);
            span.setStatus(StatusCode.ERROR);
            throw t;
        } finally {
            // 结束 Span
            span.end();
        }
    }
```

3. 前往 腾讯云可观测平台控制台，在**应用性能监控 > 调用查询**中，找到相关的调用链，单击 Span ID 后，链路详情页面中，即可查到通过自定义埋点新增的 Span。

健康检查

最近更新时间：2023-05-17 15:40:41

操作场景

集群健康检查功能是腾讯云容器服务（Tencent Kubernetes Engine，TKE）为集群提供检查各个资源状态及运行情况的服务，检查报告将详细展示组件、节点、工作负载的状态和配置的检查内容。若出现异常项，可进行异常详情描述，并自动分析异常级别、异常原因、异常影响和修复建议等。

> **注意**
> 在健康检查过程中，您的集群内会自动新建 namespace tke-cluster-inspection，并安装一个 Daemonset 进行节点信息采集，检查结束后均会被自动删除。

主要检查项目

检查类别	检查项	检查内容	仅独立集群
资源状态	kube-apiserver 的状态	检测组件是否正在运行，如果组件以 Pod 形式运行，则检测其24小时内是否重启过。	是
	kube-scheduler 的状态		是
	kube-controller-manager 的状态		是
	etcd 的状态		是
	kubelet 的状态		否
	kube-proxy 的状态		否
	dockerd 的状态		否
	master 节点的状态	检测节点状态是否 Ready 且无其他异常情况，如内存不足，磁盘不足等。	是
	worker 节点的状态	检测节点状态是否 Ready 且无其他异常情况，如内存不足，磁盘不足等。	否
	各个工作负载的状态	检测工作负载当前可用 Pod 数是否符合其期望目标 Pod 数。	否
运行情况	kube-apiserver 的参数配置	根据 master 节点配置检测以下参数： max-requests-inflight：给定时间内运行的非变更类请求的最大值。 max-mutating-requests-inflight：给定时间内运行的变更类请求的最大值。	是
	kube-scheduler 的参数配置	根据 master 节点配置检测以下参数： kube-api-qps：请求 kube-apiserver 使用的 QPS。 kube-api-burst：和 kube-apiserver 通信的时候最大 burst 值。	是
	kube-controller-manager 的参数配置	根据 master 节点配置检测以下参数： kube-api-qps：请求 kube-apiserver 使用的 QPS。 kube-api-burst：和 kube-apiserver 通信的时候最大 burst 值。	是
	etcd 的参数配置	根据 master 节点配置检测以下参数：quota-backend-bytes：存储大小。	是
	master 节点的配置合理性	检测当前 master 节点配置是否足以支撑当前的集群规模。	是
	node 高可用	检测目前集群是否是单节点集群；检测当前集群节点是否支持多可用区容灾。即当一个可用区不可用后，其他可用区的资源总和是否足以支撑当前集群业务规模。	否
	工作负载的 Request 和 Limit 配置	检测工作负载是否有未设置资源限制的容器，配置资源限制有益于完善资源规划、Pod 调度、集群可用性等。	否
	工作负载的反亲和性配置	检测工作负载是否配置了亲和性或者反亲和性，配置反亲和性有助于提高业务的高可用性。	否
	工作负载的 PDB 配置	检测工作负载是否配置了 PDB，配置 PDB 可避免您的业务因驱逐操作而不可用。	否

	工作负载的健康检查配置	检测工作负载是否配置了健康检查，配置健康检查有助于发现业务异常。	否
	HPA-IP 配置	当前集群剩余的 Pod IP 数目是否满足 HPA 扩容的最大数。	否

操作步骤

1. 登录 容器服务控制台，选择左侧导航栏中的**运维中心** > **健康检查**。
2. 进入“健康检查”页面，选择需要健康检查的集群，并为其选择合适的检查方式。 健康检查的三种方式分别为批量检查、立即检查和自动检查。
    - **批量检查**：适用于同时检查多个集群。
    - **立即检查**：适用于只检查一个集群。
    - **自动检查**：适用于需要周期性检查的集群。选择需要周期检查的集群，单击**自动检查**。如下图所示：

在“自动检查设置”弹窗中，可根据您的需求设置开启状态、检查周期和时刻。如下图所示：

3. 选择好检查方式之后，等待检查完成，可查看检查进度。如下图所示：

4. 检查完成后，可单击**查看结果**查看检查报告。如下图所示：

在检查报告页面，选择**资源状态**和**运行情况**分别查看资源状态和异常情况，单击**检查内容**可展示具体的检查内容，单击**异常**可查看异常级别、异常描述、异常原因、异常影响和修复建议。如下图所示：

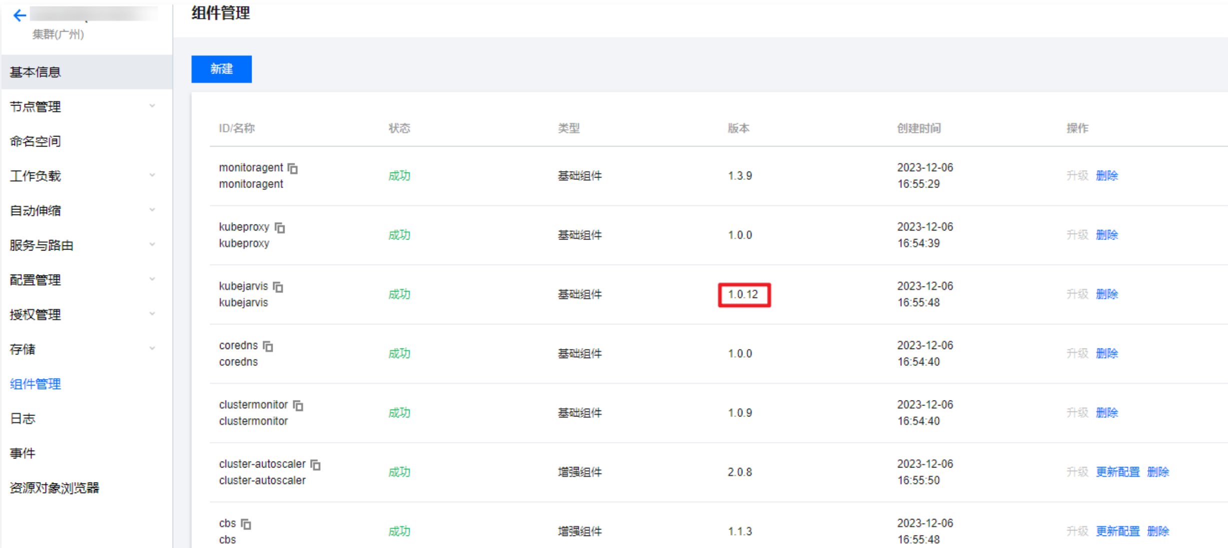
风险推送

最近更新时间：2023-12-14 18:22:32

容器服务 TKE 现在可以通过与 腾讯云事件总线 EB 对接，支持自定义推送规则的配置。TKE 检测到的风险可以及时通过短信、站内信等方式推送给您，您可以及时处理相关风险。

相关组件配置

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 在**组件管理**页面，检查 kubejarvis 版本是否大于或等于1.0.8。如下图所示：



> **说明：**
> 如果版本低于1.0.8，请进行组件升级。

4. 运行命令 `kubectl edit kjc default` ，并在 exporters 下添加以下配置：

```
exporters:
- config: null
  name: my-eb-test
  requirements:
  - diagnostic: hosting-apiserver-etcd-object-counts-check
    levels:
    - warn
    - risk
    - serious
  type: eb
```

推送规则配置

1. 如果您是第一次使用事件总线功能，请先参考文档 开通事件总线 。
2. 登录 腾讯云可观测平台 。
3. 在**事件总线 > 事件规则**页面，地域选择广州，事件集选择 default，单击**新建事件规则**。
4. 配置需要推送的事件。以集群资源数量超过推荐值/最大配额为例，配置如下：
   - 规则名称：按需求配置。
   - 事件匹配：
     - 事件模式：选择**表单模式**。
     - 云服务类型：容器服务。

- 事件类型：以“集群资源数量超过推荐值”和“集群当前资源数量超过了最大配额”为例。

5. 单击**下一步**，配置通知相关参数：

- **触发方式**：选择“消息推送”。
- **消息模板**：选择“通用通知模板”。
- **告警内容**：按需选择。
- **通知方式**：选择“渠道推送”。
- **接收对象**：您希望接受推送的账户。
- **接收渠道**：按需选择。

> **注意：**
> - 使用限制：短信消息限制500字，电话限制350字。如果您的实例名等字段过长，可能导致消息因长度超限而发送失败，建议您同时配置多个渠道。
> - 跨境进行接口回调可能存在因网络不稳定导致回调失败的情况，请您谨慎选择。

6. 单击**完成**。用户收到通知后，通知内容中会显示超限的资源名称、当前数量、推荐值，以及最大配额数量。如下图所示：

云服务 容器服务 事件通知

尊敬的腾讯云用户，您好!

您的腾讯云账号（账号 ID：332*******，昵称：[redacted] 云服务产品 容器服务 事件 tke:ErrorEvent:ApiserverEtcdObjectCountsWarn 已经产生，请及时关注并处理。

事件类型：tke:ErrorEvent:ApiserverEtcdObjectCountsWarn

产品类型：容器服务

资源 ID：uuid: | Name:hosting-apiserver-etcd-object-counts | resourceName:pods | currentValue:161 | softNum:150 | hardNum:600 | instance: cls-

资源地域：ap-

事件产生时间：2023-06-30 17:40:14 (UTC+08:00)

查看更多详情，请登录腾讯云「事件总线」产品控制台查看与管理。

进入事件总线控制台

备份中心
概述

最近更新时间：2024-06-27 14:09:11

腾讯云容器服务 TKE 备份中心为容器化应用的备份、恢复与迁移提供了一体化解决方案，目前已支持 TKE 标准集群资源对象的备份与恢复。本文主要介绍备份中心的使用场景及核心组件。

使用场景

- **备份恢复**：当集群或命名空间下的所有资源被误删除时，可以通过备份数据快速恢复业务。
- **业务合规**：配合安全部门定期拉取备份数据进行业务审计。

核心组件

组件名称	描述
tke-backup	备份组件，部署在用户集群中，基于开源工具 Velero 支持通过 CRD 方式定时备份和还原 Kubernetes 集群资源。

说明：
1. 跨集群备份恢复能力要求组件版本在 1.1.0及以上，建议您及时更新。
2. 当前仅支持 kubernetes 资源对象的恢复，不支持云硬盘 CBS、负载均衡 CLB 等云资源的恢复。

部署在集群内的 Kubernetes 对象

kubernetes 对象名称	类型	资源量	Namespaces
tke-backup	Deployment	至少需要0.1核 CPU 和256MB内存	tke-backup
tke-backup	Service	-	tke-backup
tke-backup	backupstoragelocation	-	-
tke-backup	backup	-	-
tke-backup	restores	-	-

资源类型

TKE 自定义的备份相关 CRD 资源，描述如下：

资源名称	描述
Backup	指定资源对象的备份策略。创建 Backup 资源会启动备份过程，删除 Backup 资源不会关联删除已存储在备份仓库 COS 的底层数据。
BackupSchedule	指定资源对象在特定时间点的备份策略，负责定时产生 Backup 资源对象。
Restore	将备份信息恢复至 TKE 目标集群中。创建 Restore 资源会启动恢复过程。删除 Restore 资源不会产生其他影响，只会从恢复列表中移除恢复操作的记录。

操作步骤

1. 登录 容器服务控制台，创建备份仓库，详情请参见 创建备份仓库。
2. 为目标集群创建备份或定时备份策略，详情请参见 备份管理。
3. 根据备份数据恢复集群中的指定资源对象，详情请参见 恢复管理。

备份仓库

最近更新时间：2024-06-27 14:09:11

操作场景

腾讯云容器服务 TKE 备份中心为业务应用的备份和恢复提供了产品化解决方案，本文介绍如何创建备份仓库来存储备份数据。

前提条件

1. 登录 对象存储控制台 新建 COS 存储桶作为备份仓库的底层存储。容器服务角色采用最小化授权方式访问您的 COS 存储，存储桶的命名必须以 “tke-backup” 开头。具体操作步骤请参见 创建存储桶 。
2. 完成对 COS 对象读写操作的授权。在使用备份仓库之前，在控制台根据提示将策略 QcloudAccessForTKERoleInCOSObject 授权给角色 TKE_QCSRole。

> **说明：**
> 对象存储 COS 计费方式详情请参见 对象存储计费概述 。

操作步骤

1. 登录 容器服务控制台，在左侧导航栏中选择**运维中心 > 备份中心**。
2. 进入备份仓库页面，单击**创建**。
3. 在创建备份仓库页面，填写仓库基本信息，如下图所示：

   - **仓库名称**：自定义备份仓库的名称
   - **COS 地域**：选择对象存储所在地域
   - **存储桶列表**：存储桶命名需要以 “tke-backup” 开头， 如当前 COS 存储桶不适合，可前往 对象存储控制台 新建。
4. 单击**确定**完成创建。

> **说明：**
> - 不同地域的 TKE 集群可使用相同的备份仓库存储数据，但国内和国外地域之间的数据不可同时存储在一个仓库中。
> - 删除仓库时，关联了本仓库的备份对象将无法正常执行恢复操作，请谨慎处理。
> - 删除仓库时，底层存储资源不会被删除，您可前往 对象存储控制台 进一步操作。

备份管理

最近更新时间：2025-04-17 17:36:02

操作场景

腾讯云容器服务 TKE 备份中心为业务应用的备份和恢复提供了产品化解决方案，本文介绍如何针对目标集群创建备份任务和定时备份策略。

前提条件

> **说明：**
> 若您之前在集群中已安装社区开源备份组件如 velero，需要提前卸载，否则会影响 TKE 备份组件的正常安装。

在目标集群中安装 tke-backup 备份组件。您可以前往集群中的**组件管理**模块进行操作，具体操作步骤请参见 组件安装 。

操作步骤

创建备份

1. 登录 容器服务控制台 。
2. 在左侧导航中选择**运维中心 > 备份中心**，在**备份管理**中单击**创建备份**。
3. 在**创建备份任务**页面，依次填写备份信息，如下图所示：

相关字段介绍如下：

- 备份名称：请遵循控制台的提示校验规则填写备份任务的名称。
- 备份类型：
  - 立即备份：根据您筛选的业务即时创建 Backup 备份任务并执行备份操作。
  - 定时备份：创建资源对象 BackupSchedule，该对象会根据您设置的规则定时创建 Backup 备份任务。
- 备份仓库：选择已经创建好的备份仓库。
- 命名空间：选择需要备份的命名空间，代表备份您选择的命名空间下的所有应用。
- 备份有效期：备份数据的保留时长，过期后数据将被删除且无法恢复。
- 高级设置：
  - 排除命名空间：若您在**命名空间**选项处勾选了“全选”，可通过该字段快速过滤不需要备份的命名空间。
  - 备份对象：仅备份您指定的 Kubernetes 资源对象，“全选”则代表备份筛选命名空间下的所有资源对象。
  - 排除备份对象：若您在**备份对象**选项处勾选了“全选”，可通过该字段快速过滤不需要备份的资源对象。
  - 指定标签：根据您指定的标签进一步筛选资源对象，仅备份目标命名空间下带有该标签的应用。

4. 单击**确定**完成创建。

> **说明：**
> 目前支持的 Kubernetes 资源对象的备份范围包括 Deployment、StatefulSet、DaemonSet、Job、CronJob、ConfigMap 和 Secret 等。

查看备份

您可在**备份管理**页面查看**备份列表**和**定时备份**列表。

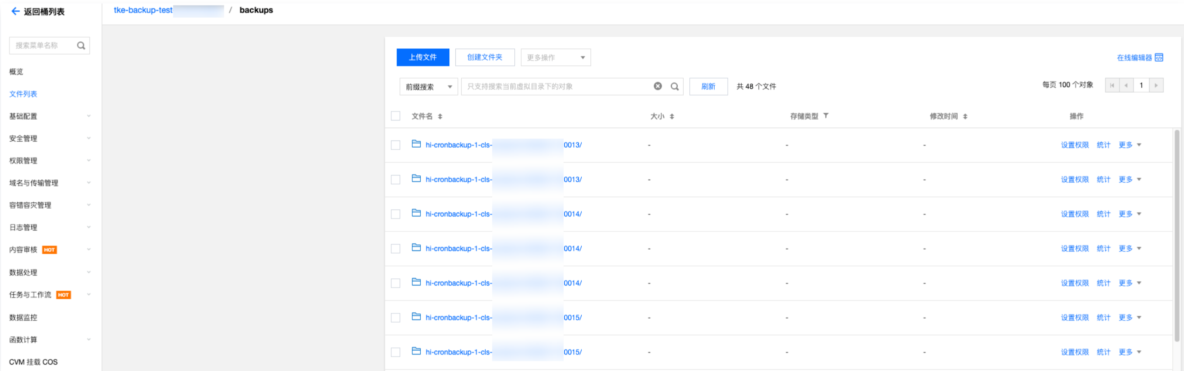
检查备份状态

状态	描述

初始化中	创建 Backup 资源对象。
执行中	执行备份任务。
完成	备份操作已完成。
部分失败	备份出现部分资源对象成功、部分失败情况，可在控制台通过查看 YAML 中的 status 字段获取成功的对象数量，失败的原因等。
失败	备份执行失败，可在控制台或通过 YAML 的 status 字段查看失败原因。

查看备份内容

您可前往 对象存储控制台 查看存储的备份数据，每个备份任务对应在 COS 的命名方式为 “备份名称-集群名称-年月日时分秒”。



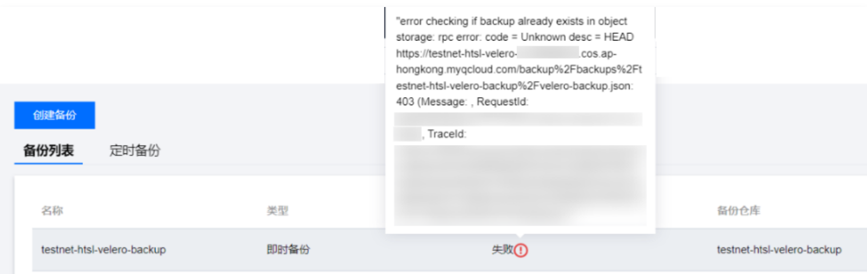
常见问题

删除备份资源时，对象存储中的数据是否会一起删除?

不会，删除备份资源时您存储在 COS 的数据会被保留。如果需要删除这些数据，您需要前往对象存储控制台进行手动操作。

创建“立即备份”任务失败，提示 “error checking if backup already exists in object storage: xxx”？

这可能是因为在历史上，您创建过一个同名的备份任务，但该任务删除后，存储在 COS 的数据还在，导致备份失败。您可以尝试修改备份名称或删除同名的备份数据来解决这个问题。



恢复管理

最近更新时间：2025-04-11 09:18:22

操作场景

腾讯云容器服务 TKE 备份中心为业务应用的备份和恢复提供了产品化解决方案，本文介绍如何针对已经创建了备份任务的目标集群进行恢复操作。

前提条件

源集群中已经创建了备份任务。

> **说明：**
> 假设您计划将A集群备份的业务恢复至B集群，为方便理解，此时A集群称为源集群，B集群称为恢复集群。

操作限制

1. 跨集群备份恢复能力要求目标集群中安装的组件版本在 1.1.0及以上，建议您及时更新。
2. 当前仅支持 kubernetes 资源对象的恢复，不支持云硬盘 CBS、负载均衡 CLB 等云资源的恢复。

操作步骤

创建恢复任务

1. 登录 容器服务控制台 。
2. 在左侧导航中选择**运维中心 > 备份中心**，在**恢复管理**中单击**创建恢复任务**。
3. 在**创建恢复任务**页面，依次填写恢复信息，如下图所示：

相关字段介绍如下：
   - 任务名称：请遵循控制台的提示校验规则填写恢复任务的名称。
   - 备份仓库：选择已经创建好的备份仓库，需要根据仓库过滤出源集群的备份数据。
   - 选择备份：选择待恢复的备份数据，**支持选择当前或其他集群（即源集群）创建的备份任务**作为恢复依据。
   - 恢复命名空间：用来恢复在源集群备份数据命名空间下找到的相关应用。
     - 所有命名空间：恢复在备份数据中找到的所有命名空间下的资源对象，您可通过“排除”选项快速过滤。
     - 指定命名空间：从备份数据中选择特定命名空间恢复资源。
   - 冲突处理：
     - 不覆盖（推荐）：若恢复集群的命名空间中存在同名的备份资源时，则当前恢复任务不会覆盖已有资源。
     - 更新：若恢复集群的命名空间中存在同名的备份资源时，则当前恢复任务会尝试对已有资源更新覆盖。
4. 单击**确定**，创建恢复任务资源 Restore 并执行恢复操作。

> **说明：**
> - 恢复任务无法保证100%成功。
> - 删除备份任务不会产生其他影响，也不会删除 COS 中存储的备份数据，只会从恢复列表中移除恢复操作的记录。

查看恢复状态

状态	描述
初始化中	创建 Restore 资源对象。
执行中	执行恢复任务。
完成	恢复操作已完成。
部分失败	恢复出现部分资源对象成功、部分失败情况，可在控制台通过查看 YAML 中的 status 字段获取成功的对象数量，失败的原因等。
失败	恢复执行失败，可在控制台或通过 YAML 的 status 字段查看失败原因。

跨集群备份恢复说明

1. 恢复操作时，源集群和恢复集群的网络模式（如 VPC-CNI、GR）需保持一致。
2. 暂不支持恢复集群中的云存储资源，如 CBS/CFS/COS，涉及存储相关的 Pod 可能会由于找不到存储资源而 Pending。
3. 如果将业务从独立集群恢复至托管集群，可能会由于部分系统组件部署模式的差异而造成恢复任务失败。为提高恢复成功率，建议您在恢复时优先选择“指定命名空间”进行少量多次操作。
4. kubernetes 不同版本之间会存在不兼容的变更（如参数废弃、apiserver 版本变更），为提高恢复成功率，恢复集群的 Kubernetes 版本与源集群尽量保持相同或相邻大版本，如源集群1.18可恢复至目标集群1.18/1.20。
5. 建议在执行恢复动作前预先检查恢复集群的资源情况，资源不足时可能会造成 Pod Pending。
6. 源集群和目标集群不在相同地域下需确保网络联通性，否则会造成镜像拉取失败造成 Pod Pending。
7. 若您的 Service 资源绑定的 CLB 开启了删除保护（即 CLB 没有随 Service 资源一同被删除），在 Service 资源恢复时可复用原有的 CLB 实现业务恢复。

备份恢复实践

最近更新时间：2024-06-27 14:09:11

操作场景

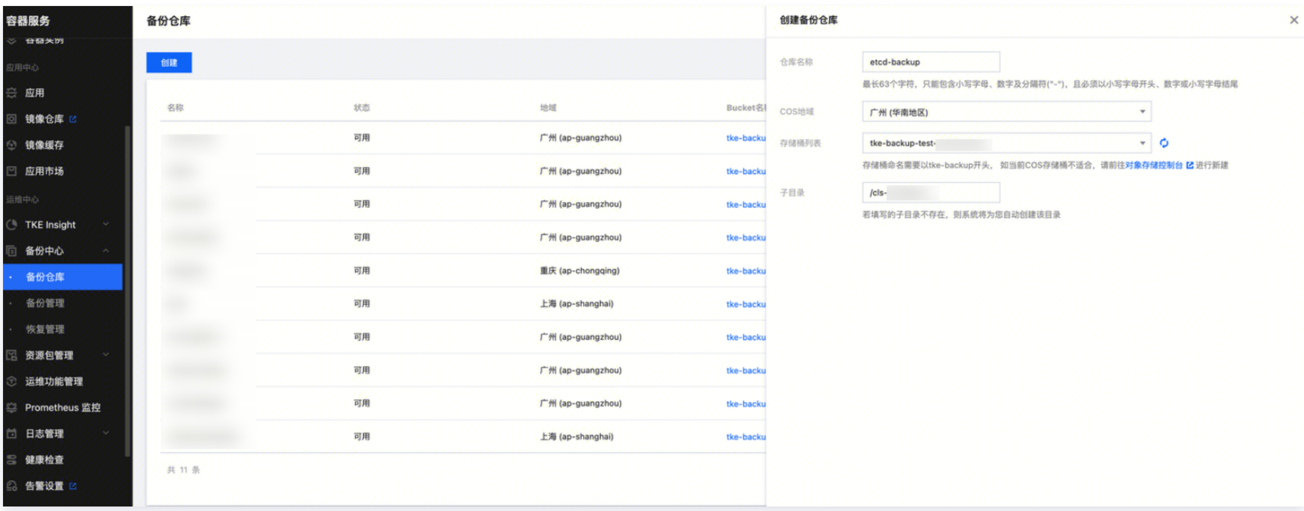
本文档以误删除集群中的 “service-controller” 资源为例，为您介绍如何进行备份和恢复操作。

操作详情

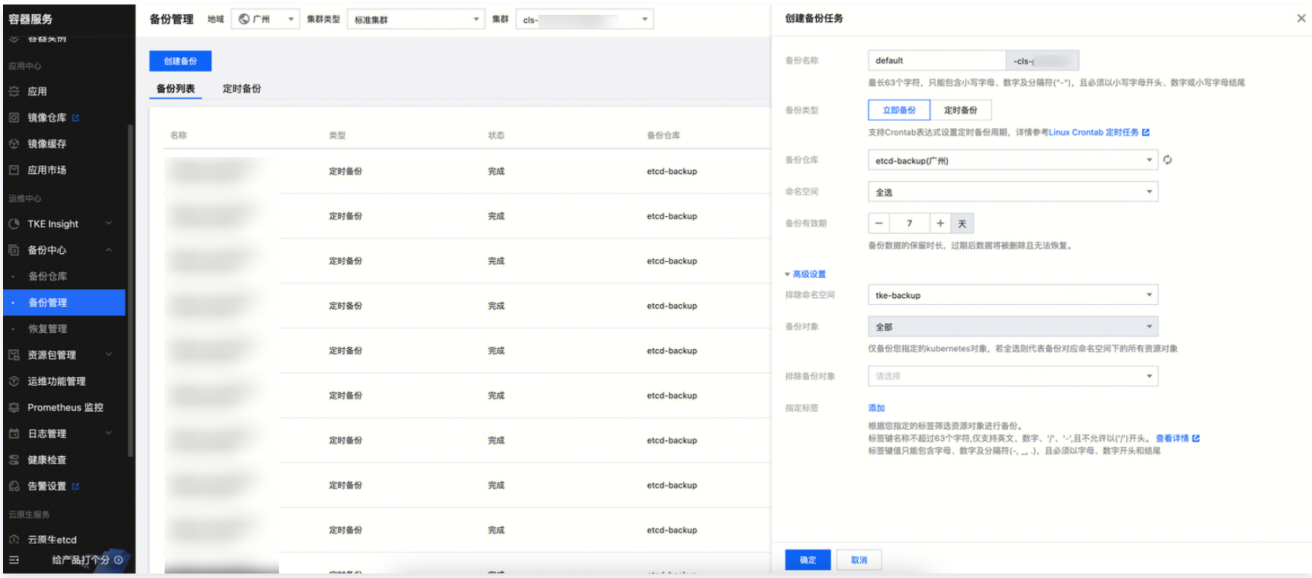
1. 开启周期性备份

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群组件管理中安装 “tke-backup” 组件，并确保组件状态显示为“成功”。组件安装步骤详情请参见 通过组件管理页安装组件 。

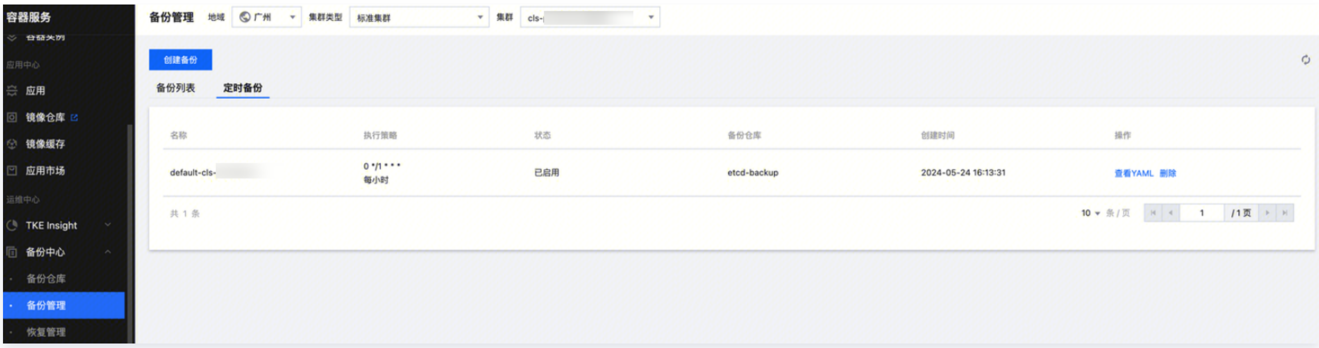
3. 在**备份中心 > 备份仓库**页面，单击**创建**，新建并完成仓库配置，操作详情请参见 创建备份仓库 。



4. 在**备份管理**页面为集群创建周期性备份，操作详情请参见 创建备份 。



5. 在定时备份中确认备份情况，如下图所示：



6. 在备份列表中确认状态是否正常。

2. 模拟误删除

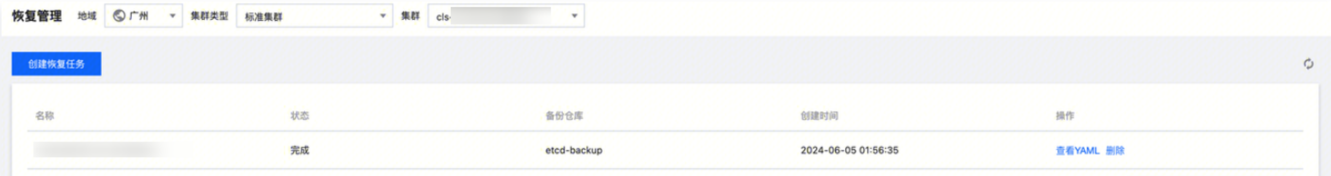
模拟误删集群 kube-system 命名空间下的 service-controller 资源对象，此时集群中新增 service 对象出现异常。

```
[[redacted]@VM-46-151-centos ~]$ k delete deploy -n kube-system   service-controller
deployment.apps "service-controller" deleted
[[redacted]@VM-46-151-centos ~]$ █
```

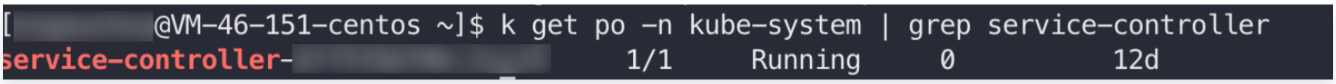
3. 完成恢复流程

1. 在**备份中心 > 恢复管理**页面，选择集群并在当前集群下创建恢复任务，操作详情请参见 创建恢复任务 。
   - 由于 service-controller 部署在 kube-system 命名空间下，我们选择了对应的命名空间进行恢复。
   - 在冲突处理中，我们选择“不覆盖”，这意味着除 service-controller 之外，其他在 kube-system 命名空间下运行的同名资源对象在本次恢复中不会被备份数据所更新。

2. 在**备份中心 > 恢复管理**页面，确认恢复任务的执行状态。



3. 后台查看 service-controller 对应 Pod 的执行情况是否运转正常，业务是否已经恢复。



容器登录方式

最近更新时间：2025-06-04 18:05:02

通过 OrcaTerm 一键登录容器

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 在**集群**页面，单击集群名称，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点**页签。
4. 在节点列表，选择节点右侧的**更多 > 登录**。

5. 在 OrcaTerm 页面，通过**身份验证**即可完成登录并进入容器。

通过 SSH 登录容器

如果您的容器已安装 SSH 服务端，可以通过 SSH 登录容器。

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点**。
4. 在节点列表中，单击**节点 ID**，进入 **Pod 管理**页面。
5. 在实例列表中，获取实例 IP 地址。如下图所示：

6. 登录集群内任意节点，详情请参见 登录到云服务器 。
7. 通过 SSH 登录到容器。

通过容器所在节点登录到容器

1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 在**集群**页面，选择集群 ID，进入集群的基本信息页面。
3. 在**节点管理**中，选择**节点**。
4. 在节点列表中，单击**节点 ID**，进入 **Pod 管理**页面。
5. 在实例列表中，获取容器所在节点 IP 地址，容器 ID。如下图所示：

6. 登录到节点，详情请参见 登录到云服务器 。
7. 通过 `docker ps` 命令查看需登录的容器。

```
[root@VM_88_88_centos ~]# docker ps | grep 75b3b15af61a
75b3b15af61a       nginx:latest       "nginx -g 'daemon off"   About a minute ago   Up About a minute
k8s_worid.e8b44cc_worid-24bn2_default_81a59654-aa14-11e6-8a18-52540093c40b_42c0b746
```

8. 通过 `docker exec` 命令登录到容器。

```
[root@VM_0_60_centos ~]# docker ps | grep 75b3b15af61a
75b3b15af61a        nginx:latest                                 "nginx -g 'daemon off"   2 minutes ago
Up 2 minutes                            k8s_worid.e8b44cc_worid-24bn2_default_81a59654-aa14-11e6-8a18-
52540093c40b_6b389dd2
[root@VM_0_60_centos ~]# docker exec -it 75b3b15af61a /bin/bash
root@worid-24bn2:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

