容器服务

TKE 调度

腾讯云

【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 调度
作业调度
Default-scheduler 调度策略配置

最近更新时间：2025-01-02 14:36:42

背景

在 Kubernetes 集群中，资源调度的有效性直接影响资源利用效率。随着集群规模的增长和应用种类的多样化，单一调度策略往往难以满足所有需求。为了满足不同场景下的调度需求，Kubernetes 提供了灵活的调度策略选项。其中，Default-scheduler 作为系统内置的默认调度器，负责将 Pod 调度至集群节点。通过配置调度策略，可以综合考虑节点的资源可用性、亲和性与反亲和性规则以及其他调度限制，从而实现 Pod 的高效调度。

版本限制

仅支持 v1.20-1.28 的 Kubernetes 版本。具体的小版本要求如下：

- v1.20: v1.20.6-tke.20 及以上
- v1.22: v1.22.5-tke.5 及以上
- v1.24: v1.24.4-tke.3 及以上
- v1.26: v1.26.1-tke.1 及以上
- v1.28: v1.28.3-tke.1 及以上

默认调度策略

我们将开放以下七种调度策略（含紧凑优先调度），确保 Pod 能够根据业务需求和集群状态，被高效、合理地调度到最佳节点上，为您提供更全面的解决方案。具体配置方法请参阅后续使用说明。

说明：

- 在 v1.20 和 v1.22 版本中，默认启用 NodeResourcesLeastAllocated 调度策略，如果开启紧凑优先调度，即启用 NodeResourcesMostAllocated 调度策略，禁用 NodeResourcesLeastAllocated 策略。
- 在 v1.24 及以上版本中，NodeResourcesFit 策略默认值是 LeastAllocated，如果开启紧凑优先调度，则 type 自动变更为 MostAllocated。
- 如果 LeastAllocated 和 MostAllocated 同时启用，则两个策略同时失效。请避免这样的情况。

调度策略名称		描述	默认权重		配置范围	备注
v1.20和v1.22	v1.24及以上		v1.20和v1	v1.24及以上		

			.22			
MostRequestedPriority	NodeResourcesFit	优先选择资源请求量高的节点来部署 Pod，以提高集群资源利用率。	默认不开启	默认 type：LeastAllocated 默认值为 1	1–100	如果希望紧凑优先调度，权重建议设置为5。 不建议配置为0，可能引起集群装箱率低，利用率低。 启用时会禁用 LeastRequestedPriority，否则两者都不生效。
LeastRequestedPriority		优先选择资源请求量低的节点来部署 Pod，以平衡集群资源利用率。	1		1–100	不建议配置为0，可能导致集群的资源利用率不均衡。
InterPodAffinityPriority	InterPodAffinity	基于 Pod 间的亲和性及反亲和性规则进行调度。	1	2	1–100	不建议配置为0，可能导致 Pod 不及预期。
BalancedResourceAllocation	NodeResourcesBalancedAllocation	确保节点的 CPU 和内存资源使用保持均衡，避免某一类资源成为瓶颈。	1	1	1–100	不建议配置为0，可能导致资源使用不平衡，某些节点可能会单一资源上成为瓶颈，影响集群的整体性能。
EvenPodsSpreadPriority	PodTopologySpread	将一组相关联的 Pod 平均分散到不同的节点上，以提高应用程序的可靠性和容错能力。	2	2	1–100	不建议配置为0，可能无法完全满足将相关联的 Pod 分散到不同节点上的需求，导致 Pod不及预期。
NodeAffinityPriority	NodeAffinity	根据节点亲和性规则进行调度，确保 Pod 调度到符合亲和性规则的节点上。	1	2	1–100	不建议配置为0，可能导致 Pod 不及预期。

TaintTolerationPriority	TaintToleration	根据节点的污点和 Pod 的容忍度进行调度，确保 Pod 只调度到它们可以容忍的节点上。	1	3	1-100	不建议配置为0，可能违反节点的调度偏好。
ImageLocalityPriority	ImageLocality	优先将 Pod 调度到已经拥有该 Pod 所需镜像的节点上，从而避免从远程存储库拉取镜像，节省网络带宽和时间。	1	1	1-100	不建议配置为0，可能导致 Pod 不及预期。

操作步骤

进入配置页面

1. 登录 容器服务控制台，选择左侧导航栏中的 集群。
2. 在集群列表选择需要配置调度策略的集群，进入集群基本信息页。
3. 选择**节点管理 > Master&Etcd**，新增 kube-scheduler 调度策略，可看到当前调度策略配置情况，单击右上角**编辑**。

配置调度策略

在编辑页面中，可以修改 kube-scheduler 的调度策略。

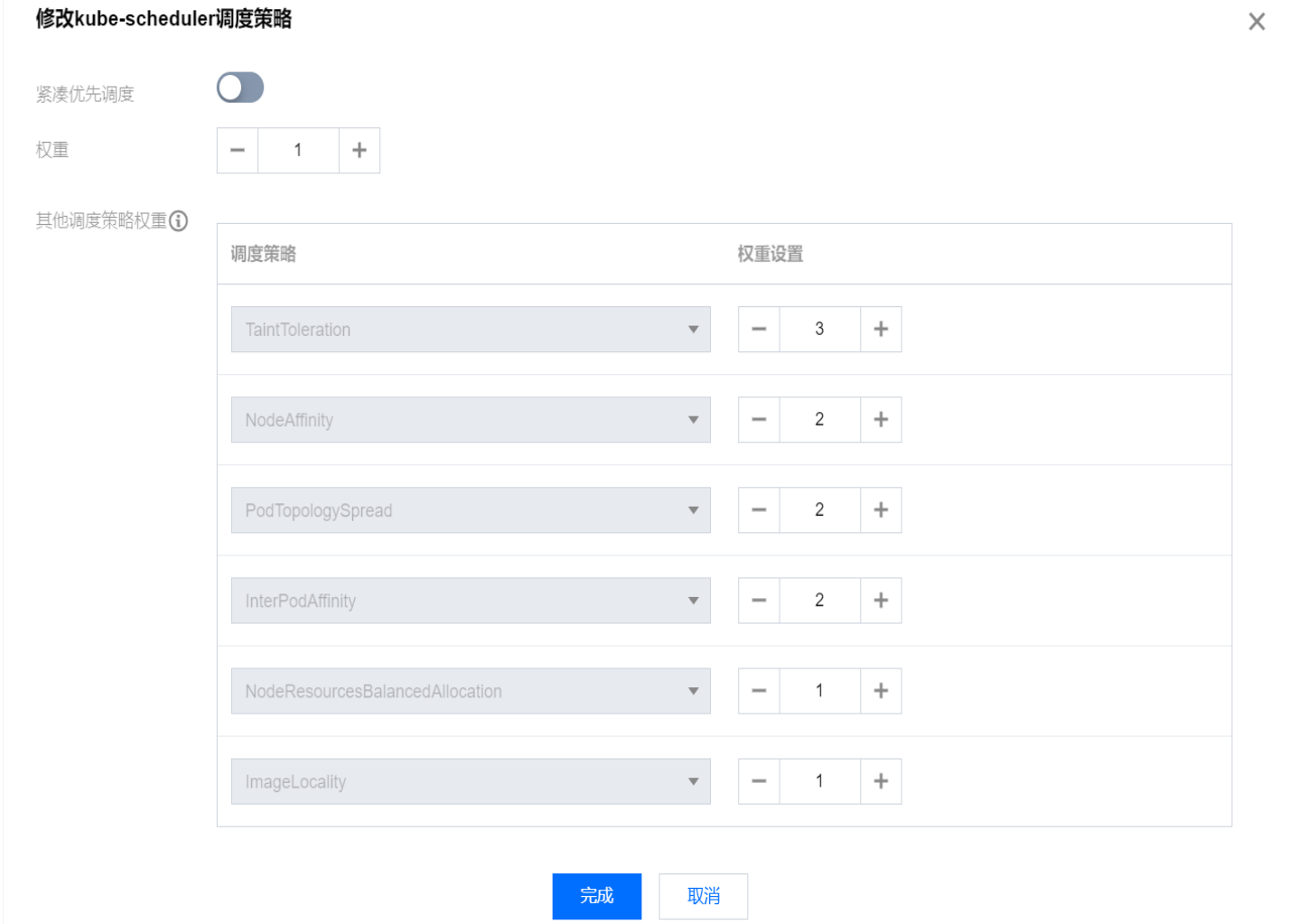
> **注意：**
> 1. 在配置调度策略时，请确保充分了解业务需求和集群资源状况，以避免不必要的资源浪费或性能下降。
> 2. 在调整调度策略时，建议先在测试环境中验证效果，再应用到生产环境。

紧凑优先调度设置

- 如果开启紧凑优先调度，NodeResourceFit 的 type 将自动变更为 MostAllocated，权重默认为1，可自定义权重1-100（若希望紧凑优先调度策略的优先级更高，建议设置为5）。
- 如果不开启紧凑优先调度，NodeResourceFit 的 type 默认为 LeastAllocated，权重默认为1，可自定义权重1-100。

其他调度策略权重设置

可自定义权重1-100。权重越高，表示该调度策略在调度决策中的优先级越高。


修改kube-scheduler调度策略
紧凑优先调度
权重
1
其他调度策略权重
调度策略
权重设置
TaintToleration
3
NodeAffinity
2
PodTopologySpread
2
InterPodAffinity
2
NodeResourcesBalancedAllocation
1
ImageLocality
1
完成
取消

Request 智能推荐

最近更新时间：2024-11-05 14:34:52

简介

组件介绍

Kubernetes 可以显著提升业务编排能力和资源利用率，但如果没有额外的支持，其提升效果会非常有限。根据 TKE 团队之前的统计数据（参见 Kubernetes 降本增效标准指南 | 容器化计算资源利用率现象剖析），TKE 节点的资源平均利用率仅为 14%左右，如下图所示：

Kubernetes 集群的资源利用率不高的主要原因是根据 Kubernetes 的资源调度逻辑，在创建 Kubernetes 工作负载时，通常需要为工作负载配置合适的资源 Request 和 Limit，表示对资源的占用和限制，其中对利用率影响最大的是 Request。为了防止自己的工作负载资源被其他工作负载占用，或者为了应对高峰流量时的资源消耗需求，用户习惯于为 Request 设置较大的数值。Request 和实际使用资源之间的差值是不能被其他工作负载使用的，因此造成了资源浪费。Request 数值设置不合理，导致了 Kubernetes 集群资源利用率低下。
容器服务 TKE 支持在集群中安装 Request 智能推荐组件。该组件可以为 Kubernetes 的 Workload 推荐容器级别的 Request/Limit 数值，从而减少资源浪费。该功能仅适用于标准集群，会对标准集群下所有的 Workload 定时更新推荐的 Request 配置。用户可以按需配置一键更新，但更新后 Pod 会重新调度到原生节点上。

部署在集群内的资源对象

开启集群的 Request 智能推荐，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
analytics.analysis.crane.io	CustomResourceDefinition	-	-
recommendations.analysis.crane.io	CustomResourceDefinition	-	-
crane-system	Namespace	-	-
housekeeper-default	Analytics	-	crane-system

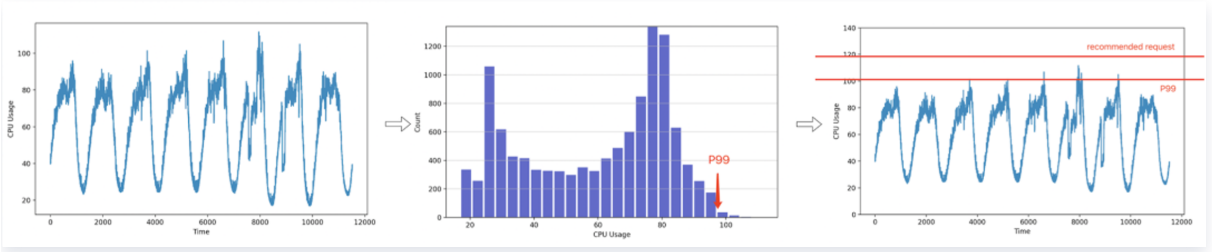
recommendation-config	ConfigMap	-	crane-system
craned	ClusterRole	-	-
craned	ClusterRoleBinding	-	-
craned	Service	-	crane-system
craned	ServiceAccount	-	crane-system
craned	Deployment	-	crane-system

功能说明

- 支持为 Deployment、StatefulSet、DaemonSet 中的每一个 Container 智能推荐合适的资源 Request/Limit。
- 支持一键更新：使用推荐值一键更新 Workload 中 Container 的资源值。
- 支持维持 Request/Limit 比例：推荐的 Request/Limit 会维持初始 Workload 中 Container 设置的 Reqeust/Limit 之间的比例，若 Limit 在创建 Workload 时没有设置，则不会推荐 Limit。
- 控制台的 Request 推荐一键更新能力会默认给工作负载加上 nodeSelector 的属性，Workload 在更新时， Pod 将只能调度到原生节点上，若原生节点资源不足，会引发 Pod 的 Pending。

Request 推荐原理

- 组件在 crane-system 命名空间下创建 Analytics CR 对象，默认覆盖集群中的 Kubernetes 原生工作负载（Deployment、StatefulSet），会分析工作负载最长 14 天的监控数据，12 小时更新一次推荐值。
- 然后根据 Analytics 生成集群内每个工作负载的 Recommendation CR 对象，用于存储推荐的数据。
- Recommendation CR 如果产生了推荐数据，就会把推荐数据写入到对应工作负载的 Annotation 里。



注意事项

环境要求

Kubernetes 版本：1.10+

节点要求

容器服务控制台中**一键更新 Workload Request** 功能会将工作负载迁移至 原生节点，若您的集群原生节点上资源不足，会导致 Pod 发生 Pending。

被控资源要求

- 支持 Deployment、StatefulSet。
- 不支持 Job、CronJob，不支持不是由 Workload 管理的 Pod。

推荐阈值

推荐最小值：单个容器推荐的 CPU 最小值是0.125核，即125m；内存的最小值是125Mi。

使用说明

安装组件

1. 登录 容器服务控制台 。
2. 在左侧选择 **TKE Insight > Node Map**。

> **说明：**
> 您也可以在 **TKE Insight > Workload Map** 中进行安装。

3. 在 Node Map 页面中，鼠标悬浮到页面下方某一个 Node 上，单击**详情**。
4. 在该 Node 的详情页的右上角，打开“Request 推荐” 开关，进行调度器的参数配置。

> **注意：**
> - 该功能是集群级别的全局开关，开启后，会自动分析工作负载历史的监控数据，推荐合适的 Request 数值。
> - 开启后非立即生效，为准确计算推荐值，需要分析该 Workload 的历史资源使用数据。
> - 不同的 Workload 的计算时间长度可能不一致，集群中不同的 Workload 之间互相可能会有影响。
> - 开启该功能后，对至少运行一天的 Workload 产生推荐数据。
> - 对于开启功能后新建的 Workload，一般情况下，也需要一天的时间才会产生 Workload 的推荐数据。
> - 建议工作负载稳定运行一段时间之后，再使用推荐值更新 Workload。

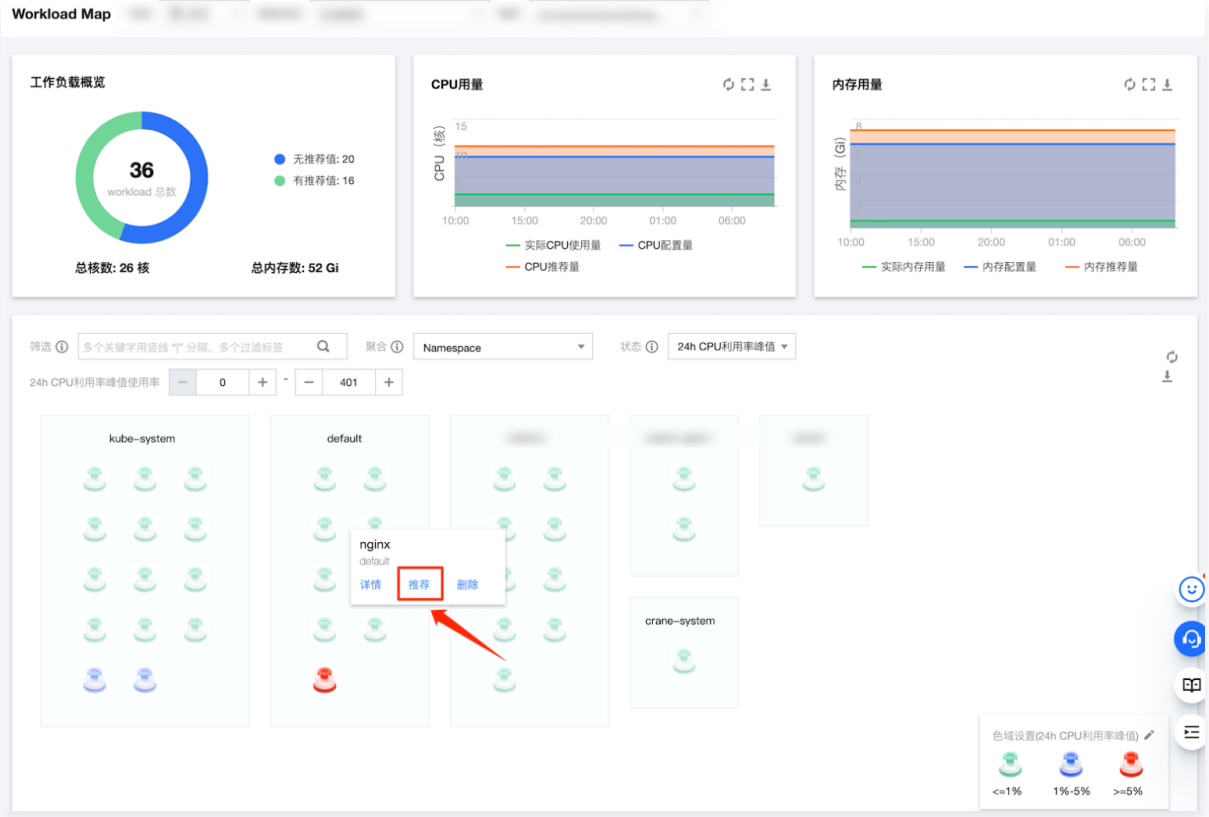
使用组件

1. 登录 容器服务控制台 。
2. 在左侧选择 **TKE Insight > Workload Map**。

> **说明：**

> Workload Map 主要通过可视化的页面展示工作负载的各项状态和指标，帮助用户了解当前工作负载的配置量和实际使用情况，辅助用户分析工作负载可能存在的问题。更多可参考文档 Workload Map 。

3. 在 Workload Map 页面，鼠标悬浮到页面下方某一个 Workload 上，单击**推荐**。



4. 在弹窗中，单击**确认**，即可使用推荐的 Request 数值更新原始 Workload 里面的数值。

> **说明：**
> 容器服务控制台中**一键更新 Workload Request** 功能会将工作负载迁移至 原生节点 ，若您的集群原生节点上资源不足，会导致 Pod 发生 Pending。

后台获取推荐数值

Request 智能推荐组件会将每个工作负载的推荐值保存在该工作负载的 YAML 里，您可以通过标准的 Kubernetes API 获取每个工作负载的推荐值，然后集成到业务的发布系统中。如下所示查看工作负载下每个容器的 Request 推荐量：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    analysis.crane.io/resource-recommendation: |
      containers:
      # 若一个 Pod 里有多个容器，每个容器都有 CPU 和 Memory 的 Request 的推荐值
      - containerName: nginx
```

```
        target:
          cpu: 125m
          memory: 125Mi #若这里缺少单位，显示的是字符串"58243235"，省略的单位是byte
```

注意：
组件本身不会推荐 Limit，在控制台使用 Request 推荐值更新 Workload 时，会维持该 Workload Request 和 Limit 的比值以保证 QoS 不会发生变化。您如果在后台获取到 Request 推荐值，可以作为参考更新原始 Workload 的资源配置量。

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
记录 pod 的 oom 记录	pod	get/list/watch
根据 node 查找空闲节点并进行推荐	node	get/list/watch
需要更新 workload 的 annotations 记录推荐相关的信息	workload	get/list/watch
需要将异常信息以事件进行记录	event	create/patch/update
监听推荐相关资源的变更，进行资源推荐	analysis.crane.io	所有权限

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: craned
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - pods
  - nodes
  verbs:
  - get
  - list
```

```
  - watch
- apiGroups:
  - analysis.crane.io
  resources:
  - "*"
  verbs:
  - "*"
- apiGroups:
  - apps
  resources:
  - daemonsets
  - deployments
  - deployments/scale
  - statefulsets
  - statefulsets/scale
  verbs:
  - get
  - list
  - watch
- apiGroups:
    - apps
  resources:
    - daemonsets/status
    - deployments/status
    - deployments/scale
    - statefulsets/status
    - statefulsets/scale
  verbs:
    - update
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - '*'
- apiGroups:
  - autoscaling.crane.io
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - events
  verbs:
```

```
  - create
  - patch
  - update
- apiGroups:
  - prediction.crane.io
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: craned
  namespace: crane-system
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - secrets
    verbs:
      - create
  - apiGroups:
      - ""
    resourceNames:
      - craned
    resources:
      - configmaps
    verbs:
      - get
      - patch
      - update
  - apiGroups:
      - ""
    resourceNames:
      - clusters-secret-store
    resources:
      - secrets
    verbs:
      - get
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs:
```

```
    - get
    - patch
    - update
    - create
```

可抢占式 Job

最近更新时间：2024-10-29 16:09:22

简介

在 Kubernetes 集群中，因为容器化的方式，可以在一个节点上运行更多的业务，但这样也会引发业务之间资源的竞争。此时会使用 Request 来保证 Pod 申请资源的最小量，以防止在发生资源竞争时，没有足够的资源可用。但是这部分通过 Request 申请的资源量被永久“占用”，无法被别的业务使用，在流量波谷时，会造成较大浪费。因此 TKE 推出了可抢占式 Job，该类型 Job 使用的资源是集群中的闲置资源，不占用集群/节点真实的剩余可调度量，在发生资源竞争时，该部分资源会被优先回收，保证正常使用 Request 资源的业务的稳定性。

部署在集群内的资源对象

说明：
使用可抢占式 Job 需要首先在集群中安装 QoSAgent 组件并开启 CPU 使用优先级，我们会自动为所有可抢占式 Job 分配最低的优先级（优先级7），在发生资源抢占时被绝对抢占，该组件在集群中部署的资源对象请参见下方表格。

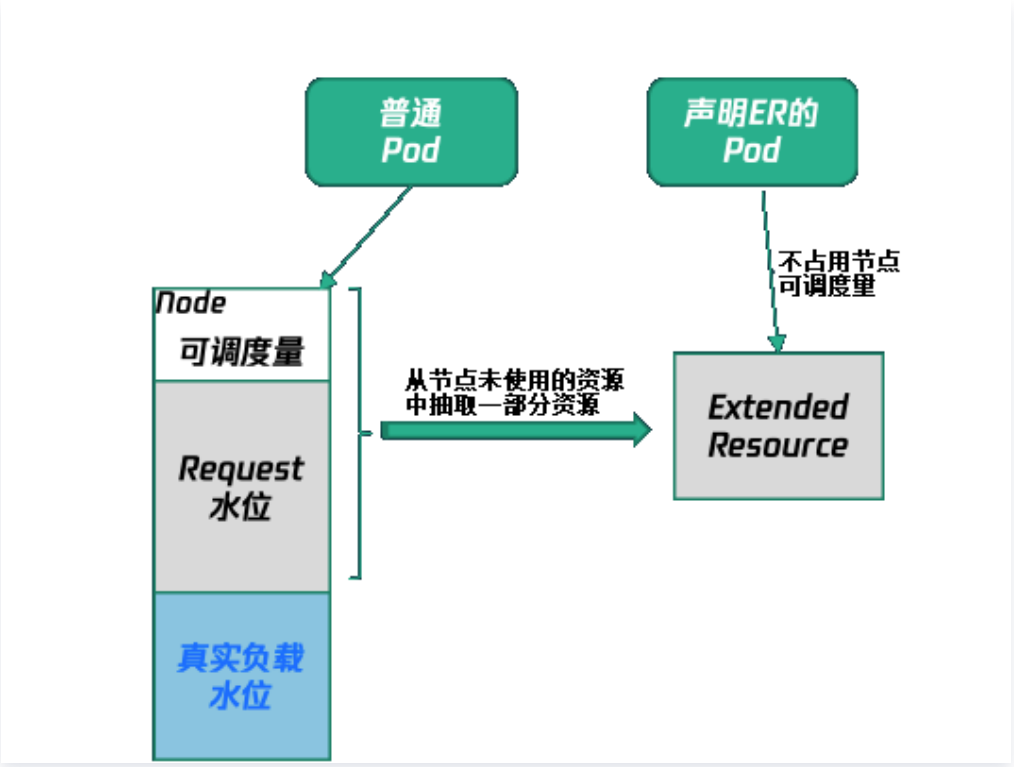
Kubernetes 对象名称	类型	默认占用资源	所属 Namespaces
avoidanceactions.ensurance.crane.io	CustomResourceDefinition	-	-
nodeqoss.ensurance.crane.io	CustomResourceDefinition	-	-
podqoss.ensurance.crane.io	CustomResourceDefinition	-	-
timeseriespredictions.prediction.crane.io	CustomResourceDefinition	-	-
kube-system	Namespace	-	-
all-be-pods	PodQOS	-	kube-system
qos-agent	ClusterRole	-	-
qos-agent	ClusterRoleBinding	-	-
crane-agent	Service	-	kube-system
qos-agent	ServiceAccount	-	kube-system
qos-agent	Daemonset	-	kube-system

功能说明

- 因为只有原生节点才有闲置资源，所以该功能仅支持在 原生节点 中使用。
- 支持为 Job、CronJob 分配集群中的闲置资源。
- 该部分闲置资源不占用集群中的剩余可调度资源。
- 当发生资源竞争时，闲置资源会被优先回收，因此被称之为可抢占式 Job。
- 节点和集群的闲置资源数值是动态变化的，根据实际的负载动态变化。当集群/节点闲置资源小于 Job 对闲置资源申请时，导致 Pod 会 Pending。

Request 推荐原理

- 闲置资源的分配方式是根据 Kubernetes 原生的 Extended Resource 实现。通过 Extended Resource 回收节点的剩余可用资源形成共享资源池，共享资源池不占用实际集群/节点的 CPU/RAM 可调度资源量。
- 如下图所示，QoS Agent 组件将集群中所有没有被真正使用的资源中抽取一部分资源，根据节点历史负载画像情况预测出安全的 Extended Resource 数值，该部分资源可以被声明使用 Extended Resource 的 Job/CronJob 使用。



使用可抢占式 Job

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. 选择左侧菜单栏中的**组件管理**，进入组件列表页面。
4. 在组件列表页面中选择**新建**，并在新建组件页面中勾选 QoS Agent。

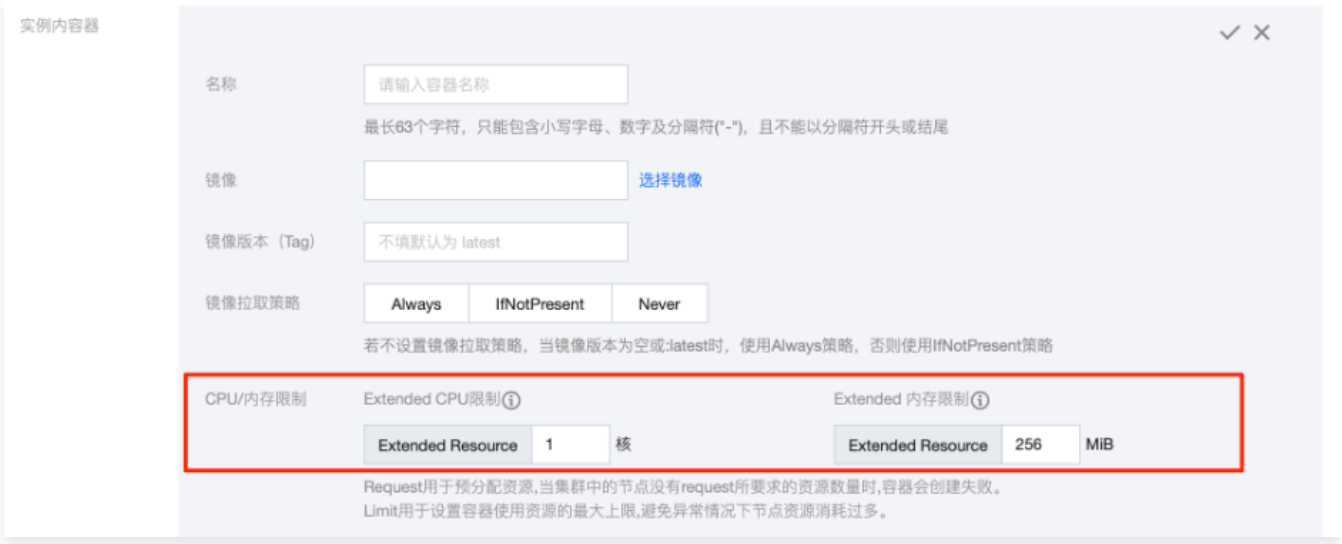
5. 单击**完成**即可安装组件。

部署可抢占式 Job

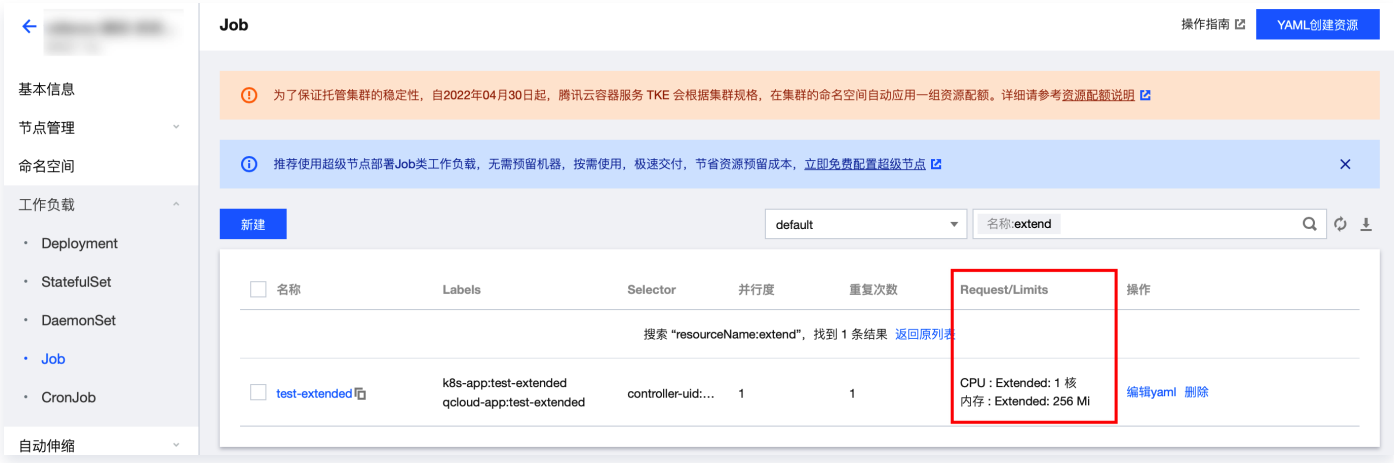
1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在集群管理页面单击目标集群 ID，进入集群详情页。
3. 选择**工作负载 > Job/CronJob**，在 Job/CronJob 资源列表页面，单击**新建**。
4. 在新建组件页面中开启**可抢占**功能。如下图所示：

开启可抢占的能力之后，Job/CronJob 的资源输入方式发生了变化，无需输入 Request/Limit 变为了 Extended CPU 和 Extended 内存：

- **Extended CPU**：申请原生节点中的空闲 CPU 资源。Extended CPU 限制只能是整数。
- **Extended 内存**：申请原生节点中的空闲内存资源。Extended 内存限制只能是整数。如下图所示：



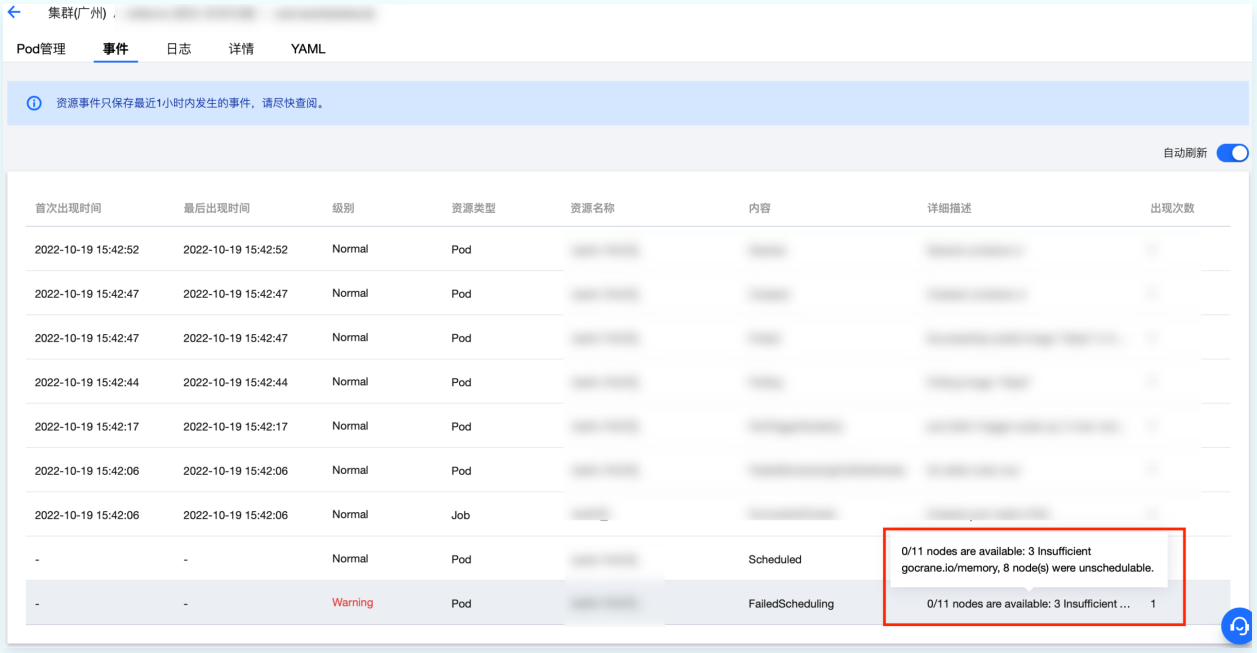
5. 通过声明使用 Extended Resource 的闲置资源创建出来的可抢占类型的 Job：



注意：

若您没有原生节点，或原生节点没有足够多的闲置资源，则可能有如下报错："Insufficient gocrane.io/memory"，表示没有足够的闲置内存资源；如果是"Insufficient gocrane.io/cpu"，表示没有足够的闲置 CPU 资源。此时需要增加 原生节点 数量。

报错信息如下图所示：



原生节点专用调度器
产品介绍

最近更新时间：2024-10-17 18:10:52

简介

组件介绍

Kubernetes 的调度逻辑为**按照 Pod 的 Request 进行调度**。节点上的可调度资源会被 Pod 的 Request 量占用，且无法腾挪。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。

部署在集群内的 Kubernetes 对象

Kubernetes 对象名称	类型	请求资源	所属 Namespace
crane-scheduler-controller	Deployment	每个实例 CPU: 200m Memory：200Mi，共计1个实例	kube-system
crane-descheduler	Deployment	每个实例 CPU: 200m Memory：200Mi，共计1个实例	kube-system
crane-scheduler	Deployment	每个实例 CPU: 200m Memory：200Mi，共计3个实例	kube-system
crane-scheduler-controller	Service	-	kube-system
crane-scheduler	Service	-	kube-system
crane-scheduler	ClusterRole	-	kube-system
crane-descheduler	ClusterRole	-	kube-system
crane-scheduler	ClusterRoleBinding	-	kube-system
crane-descheduler	ClusterRoleBinding	-	kube-system
crane-scheduler-policy	ConfigMap	-	kube-system
crane-descheduler-policy	ConfigMap	-	kube-system

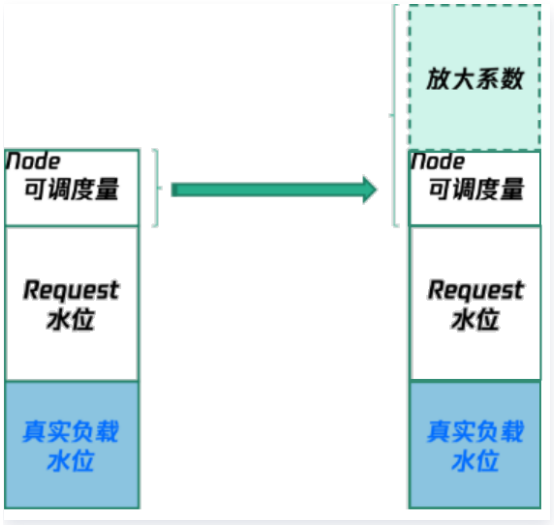
ClusterNodeResourcePolicy	CRD	-	-
CraneSchedulerConfiguration	CRD	-	-
NodeResourcePolicy	CRD	-	-
crane-scheduler-controller-mutating-webhook	MutatingWebhookConfiguration	-	-

应用场景

场景1：解决节点装箱率高但利用率低的问题

> **说明：**
> 基本概念如下：
> **装箱率**：节点上所有 Pod 的 Request 之和除以节点实际的规格。
> **利用率**：节点上所有 Pod 的真实用量之和除以节点实际的规格。

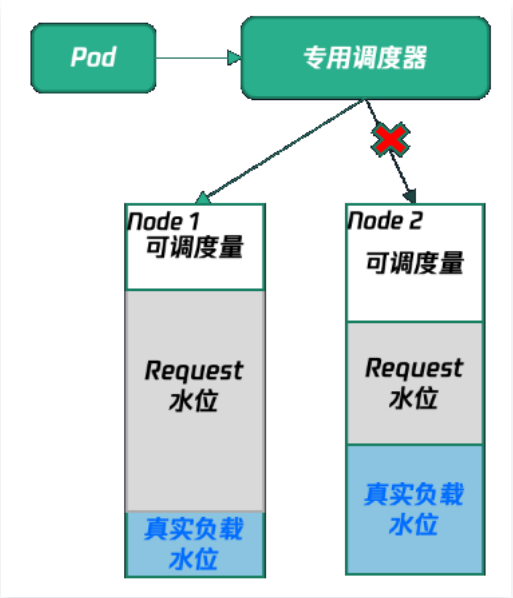
Kubernetes 原生调度器是基于 Pod Request 资源进行调度，因此，即使此时节点上的真实用量很低，但节点上所有 Pod 的 Request 之和接近节点实际规格，也无法调度新的 Pod 上来，造成较大的资源浪费。并且，业务通常为了保证自己服务的稳定性，倾向申请过量的资源，即较大的 Request，导致节点资源被占用，无法腾挪。此时节点的装箱率很高，但实际资源利用率较低。
此时可以使用原生节点专用调度器，虚拟放大节点的 CPU 和内存的规格，使得节点可调度资源被虚拟放大，可以调度进更多的 Pod。示例图如下所示：



场景2：节点水位线的设置

节点的水位线设置用于保障节点的稳定性的同时，设置节点的目标利用率：

- 调度时的水位控制：设定原生节点目标资源利用率，保障稳定性。Pod 调度时，高于该水位的节点将不被选中。并且，满足水位线的节点中，如下图所示，优先选择真实负载水位较低的节点，使得集群节点的利用率的分布更加均衡。



- 运行时的水位控制：设定当前原生节点目标资源利用率，保障稳定性。节点运行时，高于该水位的节点可能引发驱逐。因为驱逐本身是高危动作，需注意下述事项。

注意事项

1. 为避免驱逐关键的 Pod，该功能默认不驱逐 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：

```
descheduler.alpha.kubernetes.io/evictable: 'true'
```

2. 建议为集群开启事件持久化，以便更好地监控组件异常以及故障定位。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。
3. 驱逐动作对节点的要求：集群需要有3个及以上的低负载原生节点，其中低负载定义是 Node 负载小于运行时的水位控制。
4. 节点维度过滤后，开始驱逐 Node 上 Workload，此时为 Workload 级别的过滤，需要工作负载的副本数大于等于2或者是 Workload spec replicas 的一半及以上。
5. 在 Pod 维度上，如果 Pod 的负载大于节点的驱逐水位，则不可驱逐，以避免驱逐到其他节点造成其他节点负载过高。

场景3：指定命名空间下的 Pod 在下次调度时只调度到原生节点上

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。原生节点支持节点规格放大，Request 推荐等能力，因此为了充分利用原生节点的优势，建议您将工作负载调度到原生节点上。在开启原生节点调度器时，您可以选择命名空间，指定命名空间下的 Pod 在下次调度时只调度到原生节点上。

> ⚠ 注意

> 若此时原生节点资源不足，会导致 Pod Pending。

限制条件

- 该功能仅支持原生节点，更多请参考 原生节点概述 。
- 需要保证 Kubernetes 版本为 v1.22.5-tke.8，v1.20.6-tke.24，v1.18.4-tke.28，v1.16.3-tke.30 及以上。集群版本升级请参考 升级集群 。

风险控制

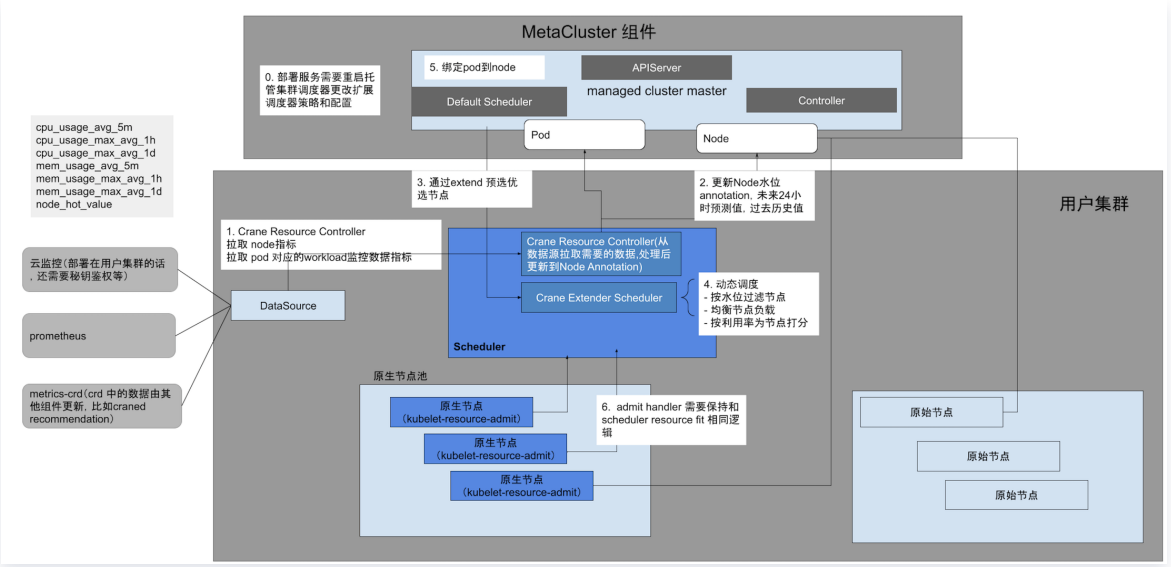
- 该组件卸载后，只会删除原生节点专用调度器有关调度逻辑，不会对原生 Kube-Scheduler 的调度功能有任何影响。原生节点上存量的 Pod 因为已经调度，不受影响。但原生节点上 kubelet 重启时，可能会引发 Pod 的驱逐，因为此时可能原生节点上 Pod 的 Request 之和可能大于原生真实的规格。
- 当调低放大系数时，原生节点上存量的 Pod 因为已经调度，不受影响。但原生节点上 kubelet 重启时，可能会引发 Pod 的驱逐，因为此时可能原生节点上 Pod 的 Request 之和可能大于原生节点当前放大后的规格。
- 用户看到 Kubernetes 集群中的 Node 资源和对应的 CVM 节点资源不一致。
- 未来可能会发生负载过高、稳定性问题。
- 节点规格放大以后，节点 kubelet 层和资源 QoS 相关模块可能受到影响，例如 kubelet 绑核，原来4核的节点被当做8核调度，Pod 绑核可能受到影响。

节点规格放大原理

用户设定节点规格放大系数以后，后台部署 Crane Resource Controller，该 Controller 会获取该系数，并不断的校验 Node 的系数是否和用户指定的系数一致。详细步骤如下：

1. 用户提交 ClusterNodeResourcePolicy CRD，通过 Label Selector 来指定需要放大的节点，Label Selector 选中的节点，将统一采用节点放大的参数。
2. Crane Resource Controller 会读取 CRD 中指定的静态超卖参数，为每个节点 Patch Node Annotation，Controller 会不停的将放大系数记录到 Node 的 Annotation 中， Reconcile Loop 保持最终一致性。
3. Kubelet 在上报节点资源的时候，请求会被 Controller 的 Webhook 拦截，并将其 Capacity 和 Allocatable 进行虚拟放大，得到新的 Capacity 和 Allocatable。

4. 用户调度器就会通过新 Capacity 和 Allocatable 调度，从而调度更多的 Pod。



组件权限说明

Crane 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。

权限场景

功能	涉及对象	涉及操作权限
需要跟踪节点的更新及变化，获取节点利用率。	nodes	get/watch/list
跟踪pod的更新及变化，根据集群内pod最近调度情况决定节点调度优先级。	pods/namespaces	get/watch/list
需要更新节点利用率到节点资源，实现调度及查询逻辑的解耦。	nodes/status	patch
需要支持多副本保障组件可用性 。	leases	create/get/update
需要跟踪 configmap 的更新及变化，实现指定 pod 调度到原生节点的功能。	configmap	get/list/watch

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: crane-scheduler
rules:
```

```
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  - namespaces
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - nodes/status
  verbs:
  - patch
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - extensions
  - apps
  resources:
  - deployments/scale
  verbs:
  - get
  - update
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - create
  - get
  - update
- apiGroups:
  - "scheduling.crane.io"
  resources:
  - clusternoderesourcepolicies
  - noderesourcepolicies
  - craneschedulerconfigurations
```

```
  verbs:
  - get
  - list
  - watch
  - update
  - create
  - patch
```

Crane descheduler 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
需要跟踪节点的更新及变化，获取节点利用率。	nodes	get/watch/list
跟踪 pod 的更新及变化，根据集群内 pod 信息决定优先驱逐的 pod。	pods	get/watch/list
驱逐 pod。	pods/eviction	create
需要得知 pod 所在的 workload ready 的个数是否占总需求数的一半及以上，来决定是否驱逐 pod。	replicasets/deployments/statefulsets/statefulsetpluses/job	get
驱逐 pod 时需要上报事件。	create	events

权限定义

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: crane-descheduler
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "watch", "list"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
  - apiGroups: [""]
    resources: ["nodes/status"]
```

```
    verbs: ["patch"]
  - apiGroups: [""]
    resources: ["pods/eviction"]
    verbs: ["create"]
  - apiGroups: ["*"]
    resources: ["replicasets"]
    verbs: ["get"]
  - apiGroups: ["*"]
    resources: ["deployments"]
    verbs: ["get"]
  - apiGroups: ["apps"]
    resources: ["statefulsets"]
    verbs: ["get"]
  - apiGroups: ["platform.stke"]
    resources: ["statefulsetpluses"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create"]
  - apiGroups: ["*"]
    resources: ["jobs"]
    verbs: ["get"]
  - apiGroups: [ "coordination.k8s.io" ]
    resources: [ "leases" ]
    verbs: [ "get", "create", "update" ]
```

使用说明

最近更新时间：2025-06-18 09:17:41

本文向您介绍如何使用原生节点专用调度器。

安装原生节点专用调度器

1. 登录 容器服务控制台 。
2. 在左侧选择 **TKE Insight > Node Map**。
3. 在 **Node Map** 页面中，鼠标悬浮到页面下方某一个 Node 上，单击**详情**。
4. 在该 Node 的详情页的右上角，打开“原生节点专用调度器”开关。

> **注意：**
> 该能力为全局能力，即一个集群只需要开启一次即可，其他节点详情页里的开关会同步开启/关闭。该组件仅针对 原生节点 生效，若原生节点资源不足，容易引发 Pod Pending。

使用原生节点专用调度器

1. 设置节点放大系数和水位线

1. 登录 容器服务控制台 。
2. 在左侧选择 **TKE Insight > Node Map**。
3. 在 **Node Map** 页面中，鼠标悬浮到页面下方某一个需要进行节点放大的原生节点上，单击**详情**。
4. 在该原生节点的详情页的右上角，单击原生节点专用调度器右侧的**编辑**。
5. 在原生节点专用调度器编辑页面，设置节点放大系数和水位线。如下图所示：

字段名称	描述
规格放大	虚拟放大原生节点规格，调度更多的 Pod，让当前节点装箱率突破100%。 **注意：** 当前节点规格放大系数仅会虚拟放大当前 原生节点 的规格，该功能有一定的风险：若节点上调度过多的 Pod，且 Pod 真正使用的总资源量大于节点规格，会引发 Pod 的驱逐和重新调度。
静态放大系数-CPU	支持输入1～3之间的数字，保留小数点后一位。
静态放大系数-内存	支持输入1～2之间的数字，保留小数点后一位。

调度时水位控制	设定当前原生节点目标资源利用率，保障稳定性。Pod 调度时，默认近5分钟平均利用率和1小时最大利用率高于该水位的节点将不被选中。 调度时水位同时默认作为： 1. 驱逐停止水位线：节点利用率在达到运行时水位，发生驱逐后，驱逐到的目标水位线。 例如：运行时水位为80，调度时水位为60，则节点利用率达到80后，开始驱逐，会按照可驱逐的 Pod 利用率高低，依次驱逐，直到水位线到60以下。 2. 低负载节点水位线：节点利用率在达到运行时水位，发生驱逐时，需要判断是否**至少有三个原生节点**的利用率低于这几个节点上的**调度时水位线**。 例如：有5个原生节点，每个原生节点的配置都是运行时水位为80，调度时水位为60，则节点利用率达到80后，开始判断是否要驱逐，要找可以容纳 Pod 的低负载原生节点，要求至少有三个原生节点的利用率低于60。 **⚠ 注意：** 如果不同的原生节点调度水位不一样，需要判断自己原生节点上的利用率和调度时水位。
调度时 CPU 目标利用率	支持输入0～100之间的整数。
调度时内存目标利用率	支持输入0～100之间的整数。
运行时水位控制	设定当前原生节点目标资源利用率，保障稳定性。节点运行时，近5分钟平均利用率高于该水位的节点可能引发驱逐，前提需要在指定工作负载上标识可驱逐的标签。 **⚠ 注意：** 1. 默认不驱逐业务 Pod。为避免驱逐关键的 Pod，该功能默认不驱逐 业务 Pod。对于可以驱逐的 Pod，用户需要在这些 Pod 所属 workload 上做标识，组件才会在触发水位线时做驱逐操作。例如：statefulset、deployment 等对象设置可驱逐 annotation： `descheduler.alpha.kubernetes.io/evictable: 'true'` 2. 需要有足够多的低负载节点才会驱逐。为保证 Pod 在驱逐后有节点可以容纳，调度器在驱逐时默认要求**至少有三个原生节点**的利用率低于这几个节点上的**调度时水位线**。因此，如果集群中存在很多原生节点，但是开启水位线的原生节点数量小于三个，或者低于**调度时水位线**的原生节点少于三个，则驱逐无法执行。 3. 驱逐策略在 cranescheduler 1.5.0版本之后支持设置，详细操作请参见 调度策略配置 。
运行时 CPU 目标利用率	支持输入0～100之间的整数。注意：运行时 CPU 目标利用率要大于调度时 CPU 目标利用率。

运行时内存目标利用率	支持输入0～100之间的整数。注意：运行时内存目标利用率要大于调度时内存目标利用率。
停止驱逐水位控制	原生节点调度器 v1.4.0版本及后续版本，支持控制台可视化配置停止驱逐水位。表示当某节点高负载时，会根据 Pod 近5分钟利用率驱逐能驱逐的 Pod，直到节点预期利用率到停止驱逐水位。
停止驱逐 CPU 目标利用率	支持输入0～100之间的整数。注意：运行时 CPU 目标利用率要大于调度时 CPU 目标利用率。
停止驱逐内存目标利用率	支持输入0～100之间的整数。注意：运行时 CPU 目标利用率要大于调度时 CPU 目标利用率。

2. 节点动态放大能力说明

为了解决因客户配置过于激进的放大系数导致的稳定性问题，并防止节点被过度放大，系统将选择节点目标利用率与节点当前利用率和静态放大率的较小值作为动态放大系数。原生节点调度器 v1.4.0 及后续版本支持节点动态放大功能，可根据需要启用。

节点动态放大配置

在原生节点专用调度器编辑页面，开启动态放大，设置节点 CPU 和内存目标利用率。

节点动态放大计算方法

1. 完成节点静态放大系数的配置 maxRatio 和动态放大系数目标利用率 targetUsage 配置。
2. 获取近7天节点峰值利用率 maxUsage。

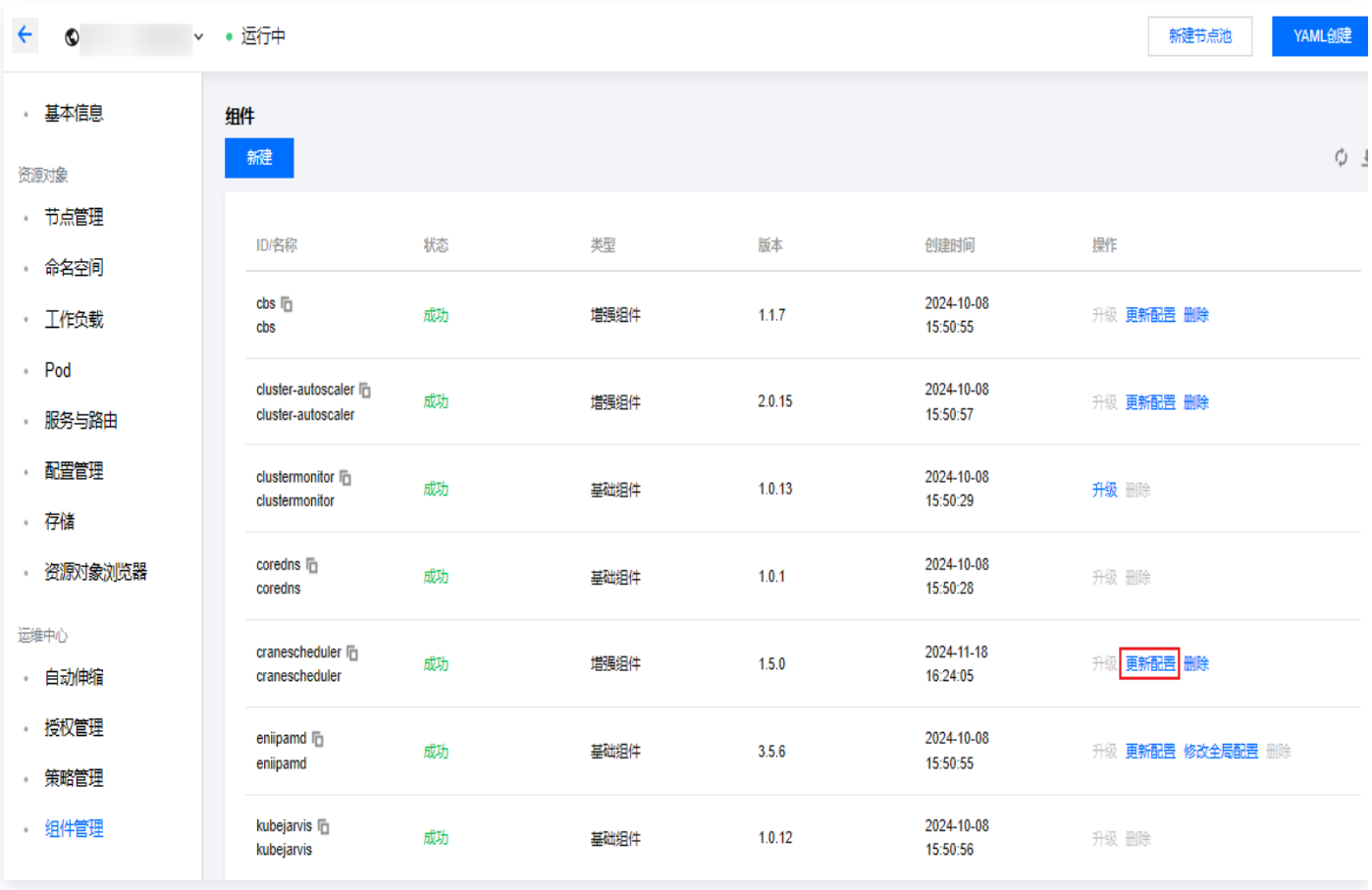
3. 动态放大系数为 targetUsage/maxUsage 和 maxRatio 中的较小值，且不会小于节点目前的真实装箱率。

3. 开启负载感知调度

为解决非 CNRP 选中的原生节点调度优先级低的问题，原生节点调度器 v1.5.0及后续版本支持开启负载感知调度。开启后，集群内的所有原生节点都将按照负载感知调度，即使节点未被 CNRP 管理，也能基于实际负载进行调度。该功能默认关闭，用户可手动在组件管理开启。

开启操作步骤如下：

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择**组件管理**，在组件列表页面中选择 “cranescheduler（原生节点专用调度器）”，单击**更新配置**。如下图所示：



4. 在更新组件配置中，开启负载感知调度。如下图所示：

4. 开启 Pod 用量扣减调度

为保证集群调度的稳定性，避免节点在临近调度水位时依旧调度高 request 的 Pod，原生节点调度器 v1.3及后续版本支持 “Pod 用量扣减”调度。开启后，调度器的预选和优选阶段将扣减 Pod 的资源使用量，计算是否满足预选规则，并参与优选评分。

Pod 资源使用量计算方式如下：

1. 若集群开启 Request 智能推荐，Pod 已有推荐值，将直接按照该推荐值进行负载扣减。
2. 若 Pod 所属 workload 暂无推荐值，则直接按照 Pod 设置的 request 值扣减。
3. 若 Pod 本身未配置推荐值，则按照默认值 CPU:100m，Mem:200Mi 扣减。

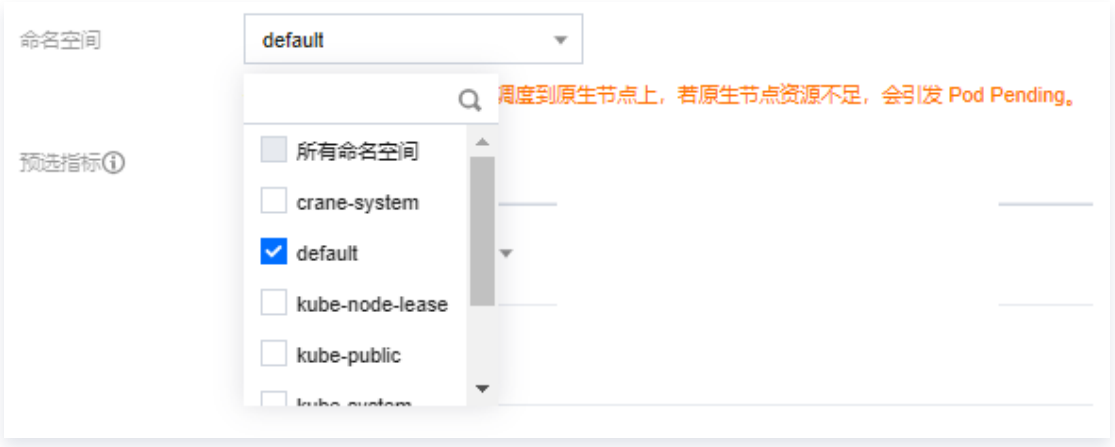
开启操作步骤如下：

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择**组件管理**，在组件列表页面中选择 “cranescheduler（原生节点专用调度器）”，单击**更新配置**。
4. 在更新组件配置中，开启 Pod 用量扣减。如下图所示：

5. 设置指定命名空间的 Pod 调度至原生节点

除了设置当前原生节点上放大系数和水位控制参数之外，原生节点专用调度器还支持设置指定命名空间下的 Pod 调度到原生节点上，帮助用户自动化迁移 Pod，充分利用原生节点的优势。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**集群管理**页面，单击目标集群 ID，进入集群详情页。
3. 选择**组件管理**，在组件列表页面中选择 “cranescheduler（原生节点专用调度器）”，单击**更新配置**。
4. 选择命名空间。指定命名空间下的 Pod 只能调度到原生节点上，若 原生节点 资源不足，会引发 Pod Pending。



节点规格放大能力 YAML 字段说明

若指定了节点的 CPU 和内存的放大系数，您可以查看原生节点上和放大系数相关的 Annotation：`expansion.scheduling.crane.io/cpu`，`expansion.scheduling.crane.io/memory`，示例如下：

```
kubectl describe node 10.8.22.108
```

```
...
Annotations:        expansion.scheduling.crane.io/cpu: 1.5   # CPU 放大系数
                    expansion.scheduling.crane.io/memory: 1.2    # 内存放大系数
...
Allocatable:
  cpu:                1930m # 该节点原始可调度资源量
  ephemeral-storage:  47498714648
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             1333120Ki
  pods:               253
...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests         Limits
  --------           --------         ------
  cpu                960m (49%)       8100m (419%)   # 该节点 Request 和 Limit
占用量
  memory             644465536 (47%)  7791050368 (570%)
  ephemeral-storage  0 (0%)           0 (0%)
  hugepages-1Gi      0 (0%)           0 (0%)
  hugepages-2Mi      0 (0%)           0 (0%)
...
```

说明：

- 当前节点原始 CPU 可调度量：1930m
- 节点上所有 Pod 的总 CPU Request 申请量：960m
- 正常情况下，该节点只能最多调度 1930m-960m=970m CPU
- 但实际上该节点 CPU 的可调度量已经被虚拟放大成：1930m*1.5=2895m；实际剩余 CPU 可调度资源为：2895-960 =1935m

此时，创建一个工作负载，只有一个 Pod，CPU 申请量为 1500m，如果没有节点放大的能力，则无法调度到该节点。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: test-scheduler
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
```

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector: # 指定节点调度
        kubernetes.io/hostname: 10.8.20.108 # 指定使用样例中的原生节点
      containers:
      - name: nginx
        image: nginx:1.14.2
        resources:
          requests:
            cpu: 1500m  # 申请量大于放大之前的节点可调度量，但有小于放大之后的节点可调度
量
        ports:
        - containerPort: 80
```

该工作负载创建成功：

```
kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
test-scheduler      1/1     1            1           2m32s
```

再次检查节点的资源占用情况：

```
kubectl describe node 10.8.22.108

...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                2460m (127%)      8100m (419%)  # 该节点 Request 和 Limit
占用量。可以看到，Request 总和超过了节点原始可调度量，节点规格放大成功。
  memory             644465536 (47%)   7791050368 (570%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
```

1. 自定义设置节点放大系数和水位线

> ⚠ **注意：**

> 自定义设置节点放大系数和水位线十分灵活，可以选择一个节点或节点池、一批节点或节点池、甚至整个集群配置相同的节点规格放大系数、水位线。
> 您可以创建多个配置文件作用到不同的节点上。若一个节点被多个配置文件声明，且不同的文件声明的参数不一样，仅创建时间最新的配置文件生效。

除控制台的操作外，原生节点专用调度器的参数配置也支持通过 YAML 配置，每个配置文件保存在名为 clusternoderesourcepolicies（简称：CNRP）的资源对象里，您可以直接创建该类型的资源对象。需要注意的是不能控制台和 YAML 创建的 CNRP 作用同一个节点，或者多个 YAML 创建的 CNRP 作用同一个节点。如果有多个 CNRP 作用同一个节点，节点放大系数将以最新创建的 CNRP 定义的生效。

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name 不能重名
spec:
  applyMode: annotation  # 默认值，暂不支持其它数值
  nodeSelector:  # 用来选取一组相同配置的节点，比如下面是用来选取一个原生节点，它的 ID 是 np-88888888-55555。也可以使用一批节点共有的标签，实现批量节点的选择。
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      # evictLoadThreshold 是原生节点运行时水位控制，设定当前这批原生节点驱逐资源利用率，保障稳定性。Pod 在原生节点上运行时，高于该水位的原生节点可能发生驱逐。为避免驱逐关键的 Pod，该功能默认不驱逐 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: descheduler.alpha.kubernetes.io/evictable: 'true'。
      # evictLoadThreshold 要大于 targetLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      resourceExpansionStrategyType: static # 默认值，暂不支持其它数值
      staticResourceExpansion:
        ratios: # 节点规格放大系数，设定当前这批原生节点的放大系数
          cpu: "3" # 节点 CPU 的放大系数。建议不要设置得过大，否则可能有稳定风险，控制台限制最大数值为3
          memory: "2" # 节点内存的放大系数。建议不要设置得过大，否则可能有稳定风险，控制台限制最大数值为2
      # targetLoadThreshold 是原生节点调度时水位控制，设定当前这批原生节点目标资源利用率，保障稳定性。Pod 调度时，高于该水位的原生节点将不被选中。
      # targetLoadThreshold 要小于 evictLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      targetLoadThreshold:
```

```
      percents:
        cpu: 70
        memory: 70
```

您可以修改或者创建该类型的资源对象。如下图所示，您可以在控制台上查看存量的 clusternoderesourcepolicies（简称：CNRP）。

1. 登录容器服务控制台，选择左侧导航栏中的 集群 。
2. 单击需要查看原生节点专用调度器的集群 ID，进入资源对象浏览器页面。在左上角的方框里搜索 clusternoderesourcepolicies（简称：CNRP），如下图所示：

3. 如果您是通过控制台页面给原生节点配置的调度参数。这些文件的命名方式为：housekeeper-policy-np-88888888-55555，其中 np-88888888 为该原生节点的节点池ID，np-88888888-55555 为该原生节点的节点 ID。

设置节点动态放大

原生节点调度器v1.4.0版本以后，支持节点动态放大，参数也可后台 YAML 配置，对应的 ClusterNodeResourcePolicy 示例如下：

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name 不能重名
spec:
  applyMode: annotation  # 默认值，暂不支持其它数值
  nodeSelector:  # 用来选取一组相同配置的节点，比如下面是用来选取一个原生节点，它的 ID
是 np-88888888-55555。也可以使用一批节点共有的标签，实现批量节点的选择。
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      resourceExpansionStrategyType: auto # 开启节点动态放大
      autoResourceExpansion:
        crontab: 0 * * * 0-6 #设置节点最大利用率每一小时刷新一次，如无特殊需求请勿修改
此参数
        decayLife: 168h #设置获取近7天的节点最大利用率，如无特殊需求请勿修改次参数
```

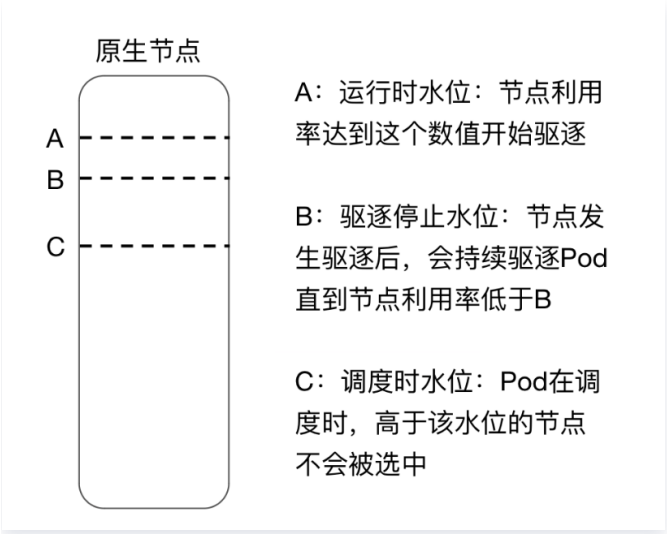
```
        # 设置节点动态放大的具体参数
        maxRatios:
          cpu: "2"
          memory: "1.5"
        minRatios:
          cpu: "1.0"
          memory: "1.0"
        targetLoadThreshold:
          percents:
            cpu: 50
            memory: 70
      # evictLoadThreshold 是原生节点运行时水位控制，设定当前这批原生节点驱逐资源利用率，保障稳定性。Pod 在原生节点上运行时，高于该水位的原生节点可能发生驱逐。为避免驱逐关键的 Pod，该功能默认不驱逐 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: descheduler.alpha.kubernetes.io/evictable: 'true'。
      # evictLoadThreshold 要大于 targetLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      # targetLoadThreshold 是原生节点调度时水位控制，设定当前这批原生节点目标资源利用率，保障稳定性。Pod 调度时，高于该水位的原生节点将不被选中。
      # targetLoadThreshold 要小于 evictLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      targetLoadThreshold:
        percents:
          cpu: 70
          memory: 70
```

2. 自定义驱逐停止水位线

上文中提到，当前调度时的水位还有两个额外功能：驱逐停止水位线，低负载节点水位线。但是因为这样的设计，导致了功能绑定的问题，例如：

1. 如果用户想在业务驱逐时，尽量少驱逐一些 Pod，这样就需要调大调度时水位。调大调度时水位会导致节点上调度更多的 Pod，驱逐的概率提升；而又因为驱逐的 Pod 数量变少了，因此节点可能会发生频繁的调度和驱逐，节点不稳定性增加。
2. 如果用户想在业务驱逐时，尽量多驱逐一些 Pod，这样就需要调小调度时水位。调小调度时水位会导致节点上调度更少的 Pod，驱逐的概率降低；而又因为驱逐的 Pod 数量变多了，因此节点可能利用率急剧变化，节点不稳定性也会增加。且因为调度时水位较低，因此利用率无法提升，资源浪费明显。

因此，在新的 Crane Scheduler 中，TKE 推出了第三条水位线：驱逐停止水位线。驱逐停止水位线就包含了调度时的水位的两个额外功能：驱逐停止水位线，低负载节点水位线。



注意：
该功能要求 Crane Scheduler 至少在 1.1.10 版本以上，可参考 文档 查看版本和升级。

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name 不能重名
spec:
  applyMode: annotation  # 默认值，暂不支持其它数值
  nodeSelector:  # 用来选取一组相同配置的节点，比如下面是用来选取一个原生节点，它的 ID 是 np-88888888-55555。也可以使用一批节点共有的标签，实现批量节点的选择。
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      # evictLoadThreshold 是原生节点运行时水位控制，设定当前这批原生节点驱逐资源利用率，保障稳定性。Pod 在原生节点上运行时，高于该水位的原生节点可能发生驱逐。为避免驱逐关键的 Pod，该功能默认不驱逐 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：descheduler.alpha.kubernetes.io/evictable: 'true'。
      # evictLoadThreshold 要大于 targetLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      ##################
      ## 驱逐停止水位线 ##
      # 数值要低于运行时水位 evictLoadThreshold
      evictTargetLoadThreshold:
```

```
        percents:
          cpu: 75
          memory: 75
      resourceExpansionStrategyType: static # 默认值，暂不支持其它数值
      staticResourceExpansion:
        ratios: # 节点规格放大系数，设定当前这批原生节点的放大系数
          cpu: "3" # 节点 CPU 的放大系数。建议不要设置得过大，否则可能有稳定风险，控制台限制最大数值为3
          memory: "2" # 节点内存的放大系数。建议不要设置得过大，否则可能有稳定风险，控制台限制最大数值为2
      # targetLoadThreshold 是原生节点调度时水位控制，设定当前这批原生节点目标资源利用率，保障稳定性。Pod 调度时，高于该水位的原生节点将不被选中。
      # targetLoadThreshold 要小于 evictLoadThreshold，否则驱逐后，节点可能持续被调度新的 Pod 导致抖动
      targetLoadThreshold:
        percents:
          cpu: 70
          memory: 70
```

调度策略配置

最近更新时间：2024-11-18 19:08:42

功能概述

调度策略配置是原生节点专用调度器（Crane-scheduler）的一项功能，旨在帮助用户根据业务需求和集群资源状况，灵活地调整 Pod 的调度策略。通过配置调度策略，用户可以优化集群资源利用率，提高应用性能和可用性。

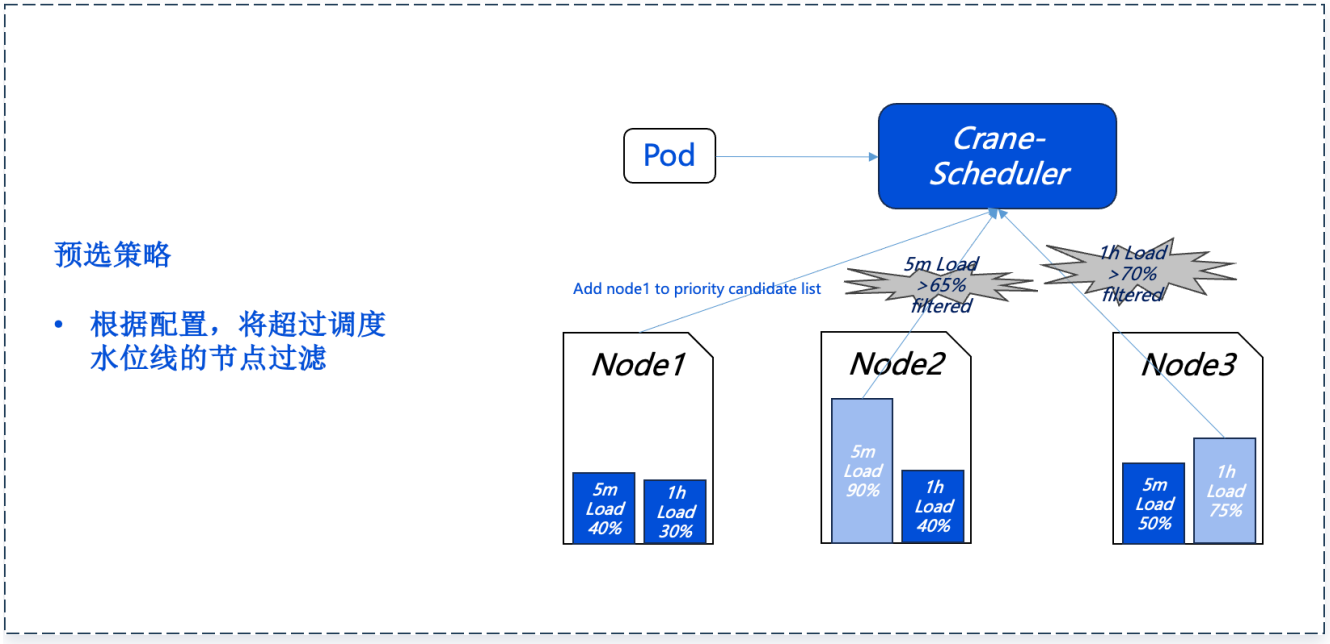
版本限制

仅支持 v1.4.1 及以上的 Crane-scheduler 版本。重调度规则配置需要1.5.0及以上的 Crane-scheduler 组件版本。

调度策略说明

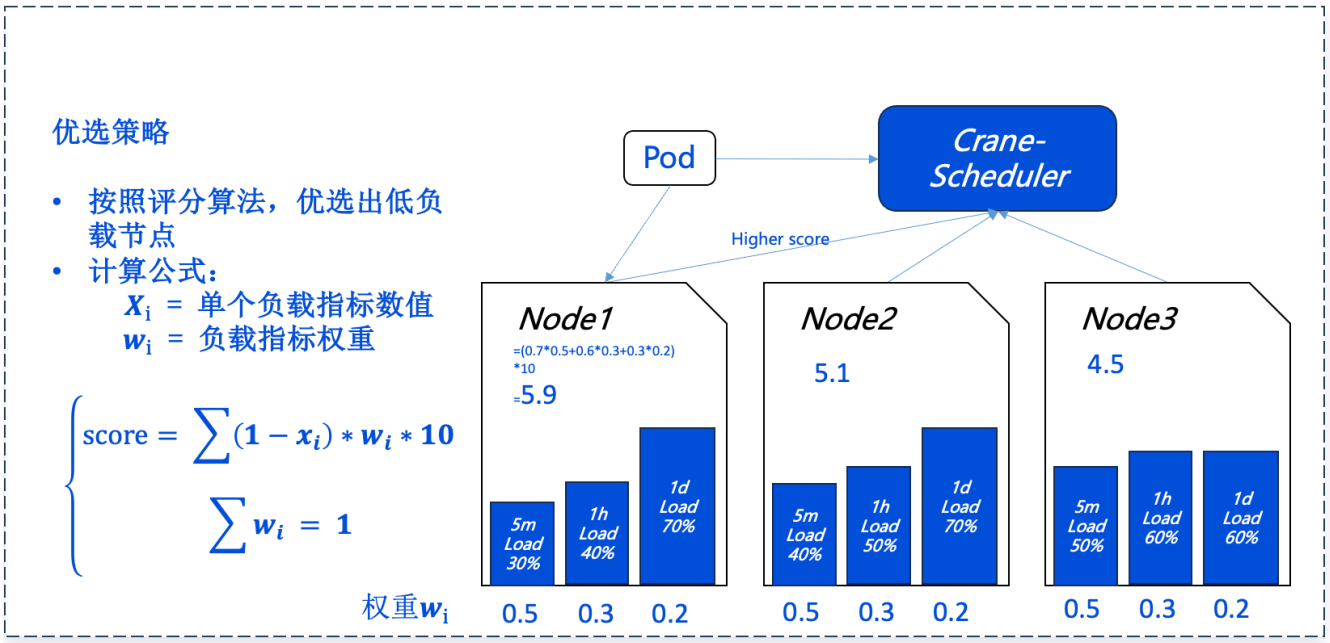
预选策略

为了避免 Pod 被调度至高负载的节点（Node），我们首先采用预选策略把高负载的 Node 过滤掉。预选策略可以根据节点的实际负载情况动态配置过滤阈值和比例。具体配置方法请参见 后续参数设置说明。
以图示为例，Node2 在过去5分钟内的负载以及 Node3 在过去1小时内的负载均超出了预设的阈值，因此它们将不会进入下一阶段的优选过程。



优选策略

优选策略旨在从符合预选策略条件的节点（Node）中挑选出最佳节点进行 Pod 调度。为了实现集群内各节点负载的均衡分布，Crane-scheduler 会依据 Node 的实时负载数据为其评分，其中负载越低的节点获得的评分越高。评分策略及其权重系数均可根据实际需求进行动态调整，详细的配置方法请参阅后续参数配置说明。
参考下图，我们可以看到 Node1 因其较低的负载而获得了最高的评分，因此它将被系统优先调度以运行 Pod。

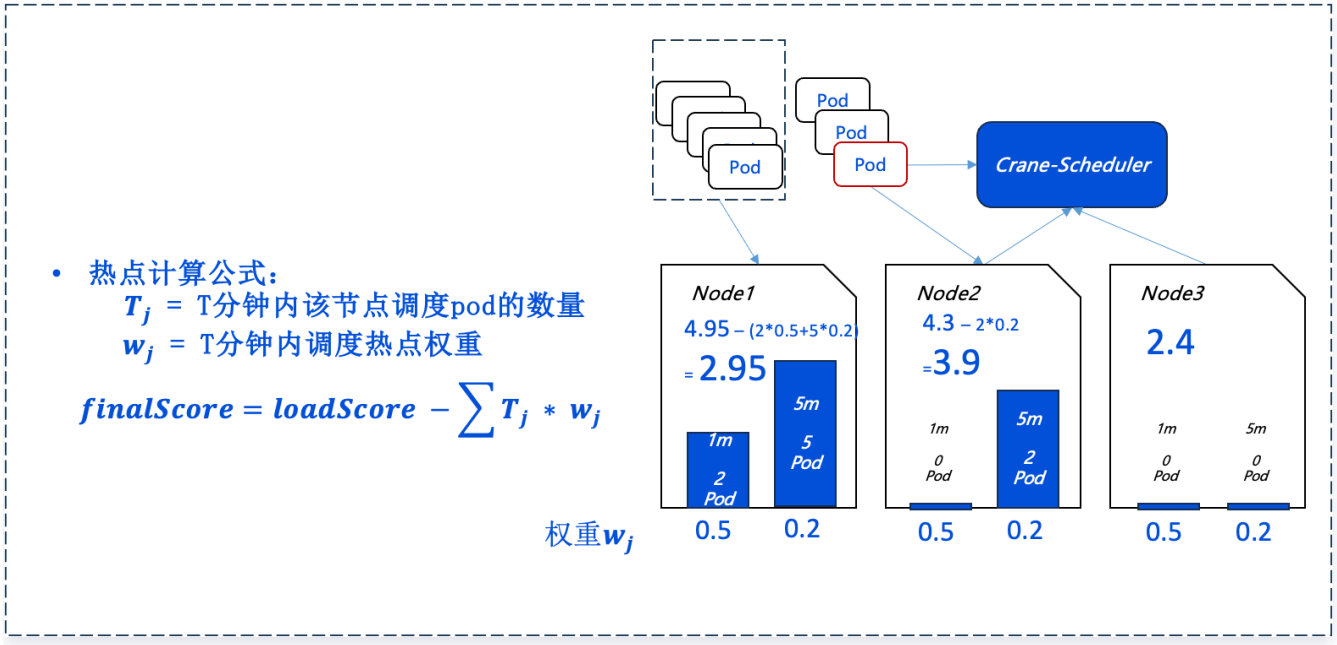


调度热点

为了避免低负载节点因持续调度大量 Pod 而变成新的负载热点，Crane-scheduler 引入了调度热点策略。该策略通过统计节点在过去一段时间内调度的 Pod 数量，并据此调整节点在优选阶段的评分。

- **节点得分计算公式：** 节点得分 = 优选策略评分 − 热点值
- **当前策略细节如下：**
  - 若节点在过去一分钟内调度的 Pod 数量超过2个，则优选评分减1分。
  - 若节点在过去五分钟内调度的 Pod 数量超过5个，则优选评分同样减1分。

通过实施这一策略，Crane-scheduler 能够确保调度决策更加全面，既考虑了节点的即时负载情况，也考虑了节点近期的调度历史，从而实现更为均衡的集群负载分布。



参数说明

预选参数

> **说明：**
> - 预选参数默认值，如您无额外需求，可直接采用。
> - 预选指标默认参数的阈值是原生节点专用调度器的调度时水位控制参数设置，详情请参见 使用说明 。

预选指标默认参数	描述
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

> **说明：**
> 优选参数默认值，如您无额外需求，可直接采用。

优选指标	默认参数	描述
5分钟平均 CPU 利用率权重	0.2	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	0.3	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	0.5	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	0.2	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	0.3	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	0.5	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。

热点参数

> **说明：**
> 热点参数默认值，如您无额外需求，可直接采用。

热点指标	默认参数	描述
近5分钟调度 Pod 数	10.0	该权重越大，近5分钟节点调度 Pod 数对节点的评分影响越大。
近1分钟调度 Pod 数	20.0	该权重越大，近1分钟节点调度 Pod 数对节点的评分影响越大。

操作步骤

开始使用

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 CranesScheduler 的操作列单击**更新配置**，进入更新组件配置页面。

配置调度策略

> **注意：**
> 1. 在配置调度策略时，请确保充分了解业务需求和集群资源状况，以避免不必要的资源浪费或性能下降。
> 2. 在调整调度策略时，建议先在测试环境中验证效果，再应用到生产环境。

预选指标配置

- **预选指标作用：**用于节点初步筛选，仅当该节点对应指标低于设定的调度阈值时，方可通过预选。
- **指标数量限制：**最多可配置4条预选指标，且指标不可重复。您也可以选择不配置预选指标。

优选指标配置

- **优选指标作用：**用于在通过预选后的节点中进一步筛选，显示默认权重参数，支持自定义权重设置。
- **调度原则：**Pod 将优先调度至资源利用率较低的节点。
- **指标数量限制：**最多可配置6条优选指标，且指标不可重复。您也可以选择不配置优选指标。

热点设置

- **热点指标作用：**考虑节点近期调度 Pod 的数量，以便更全面地评估节点的负载状况。
- **推荐权重设置：**近5分钟调度 Pod 数权重建议设为10，近1分钟调度 Pod 数权重建议设为20。
- **指标数量限制：**最多可配置2条热点指标，且指标不可重复。您也可以选择不配置热点指标。

配置重调度策略

- 驱逐间隔：系统将按照设定的驱逐间隔自动扫描集群中符合驱逐条件的 Pod。若实际执行驱逐操作的时间超过驱逐间隔，系统将在驱逐完成后立即开始下一次扫描。默认为180s。
- 高负载打散配置：系统默认将选取 **最小节点数** 与 **非超级节点数 × 最小低负载节点百分比** 中的较大数值，作为判断高负载打散的限制标准。最小节点数默认配置为3，最小低负载节点百分比默认为0.01。

自定义资源优先级（PlacementPolicy）

最近更新时间：2025-06-18 09:17:41

概述

PlacementPolicy 是 TKE（Tencent Kubernetes Engine）提供的一种智能调度策略，旨在帮助用户在 Kubernetes 集群中更灵活地管理 Pod 的调度行为。通过 PlacementPolicy，用户可以根据业务需求自定义资源的优先级，实现扩容时优先使用高优先级节点，缩容时优先释放低优先级节点，从而在降本增效与业务稳定性之间找到最佳平衡点。该功能重点解决了原生 Kubernetes 调度器存在的以下痛点：

客户痛点

1. **业务场景与调度能力错配**

   1.1 K8s 原生调度以资源供需匹配为核心，缺乏对业务价值和业务语义（成本/性能/高可用）的抽象表达。

   1.2 企业实际需求中的“业务目标→调度策略→资源分配”传导链断裂。

2. **云资源特性与调度策略割裂**

   云厂商 IaaS 层产品化节点特性未被纳入调度决策体系，例如：

   - 节点类型：原生节点、超级节点、普通节点、注册节点等。
   - 节点计费特性：包年包月、按量计费等。

3. **配置复杂度与用户体验矛盾**

   3.1 K8s 原生调度器提供原子化调度 API，需要用户根据业务语义自行组合使用，要求用户具备调度器内部实现知识，例如：污点、拓扑等。

   3.2 K8s 原生调度策略变更的运维成本与业务连续性存在冲突，关注点未分离，调度策略变更对 workload 存在侵入，需要重建 Pod。

核心特性

- **智能成本优化**

  在业务扩容或重建时，PlacementPolicy 会优先使用低成本资源。在业务缩容时，优先释放高成本资源上的业务，进一步降低运营成本，提升资源利用效率。

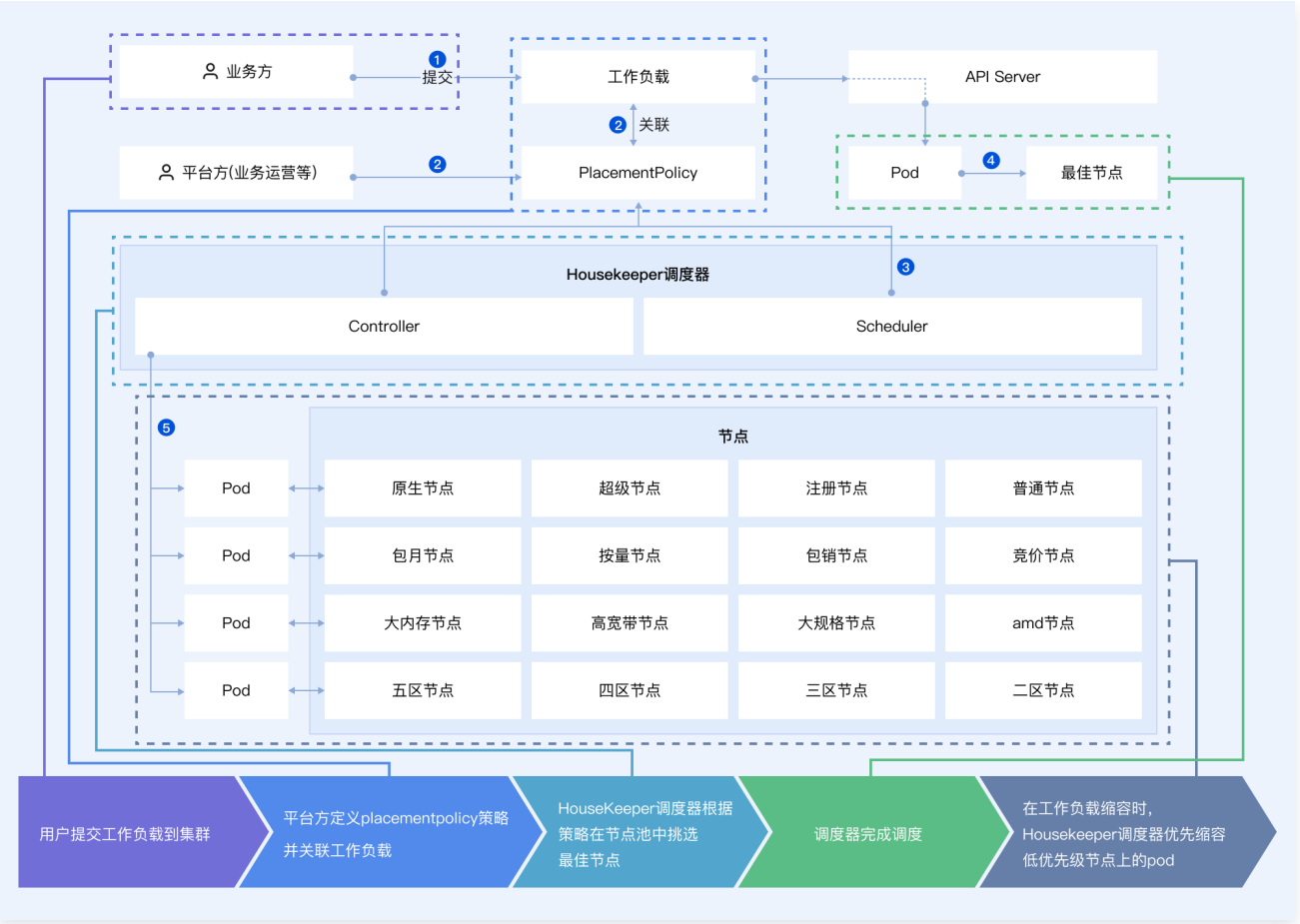
- **高可用性保障**

  针对高可用性要求严格的业务（如核心交易系统、关键基础设施等），PlacementPolicy 能够在确保业务多可用区平均分布的同时，兼顾成本最优。

- **灵活调整调度规则**

  PlacementPolicy 允许用户在保证存量业务无损的前提下，仅针对新增或重建的 Pod 动态调整调度规则，避免对整个应用进行重建，从而降低运维复杂性和风险。

功能流程



1. **提交工作负载**：用户的业务方提交工作负载到集群。
2. **定义 PlacementPolicy**：用户的平台方在集群中定义 PlacementPolicy，并将工作负载与之关联。
3. **PlacementPolicy 调度**：当工作负载扩容或者重建等需要调度场景时，TKE Crane 调度器根据 PlacementPolicy 定义，在节点池里挑选最佳节点。
4. **Pod Bind**：TKE Crane 调度器向 API Server 提交 Pod 的调度结果，进行绑定。
5. **控制 Pod 缩容顺序**：TKE Crane 调度器根据 PlacementPolicy 定义，按需为 Pod 打上 pod-deletion-cost，控制 Pod 缩容的顺序。当工作负载缩容时，低优先级节点上的 Pod 会被优先销毁。

前提条件

1. 已创建 TKE 容器集群，并升级至目标版本。
2. 已安装 CraneScheduler 调度器，并升级至目标版本。CraneScheduler 组件的安装及升级操作请参见 组件的生命周期管理 。

3. 版本要求：
   - TKE 集群版本要求：1.22 及以上版本。
   - CraneScheduler 调度器版本要求：1.6.0及以上版本。

使用限制

- PlacementPolicy 支持的 TKE 节点类型：普通节点、原生节点、超级节点。
- PlacementPolicy 支持的工作负载类型：
  - Pod 扩容场景：支持 TKE 全部类型的工作负载
  - Pod 缩容场景：仅支持 Deployment 类型的工作负载。
- 当开启节点池自动扩缩容能力时，并且您同时设置了 “DoNotSchedule” 和 “max” 字段，存在无法扩容节点的可能性。建议您开启节点池自动扩缩容能力时，尽量避免设置 “DoNotSchedule” 和 “max” 字段。PlacementPolicy 和 CA 的联动能力将持续优化。
- 本功能与 pod-deletion-cost 冲突，不能同时使用。关于 pod-deletion-cost 的更多信息，请查看 pod-deletion-cost 。
- 按量计费模式的超级节点的优先级配置，需要先取消超级节点上的 taint，或为 Pod 添加忽略 taint 的 toleration。

```
kubectl taint node $SuperNodeName eks.tke.cloud.tencent.com/eklet-
```

使用方式

> **注意：**
> PlacementPolicy 是全局调度策略，优先级高于负载调度等其他策略。

创建 PlacementPolicy 定义弹性资源优先级

```
apiVersion: scheduling.crane.io/v1alpha1
kind: PlacementPolicy
metadata:
  name: testpolicy
```

```
spec:
  targets:
    podSelectors:
    - matchExpressions:
      - key: app
        operator: In
        values:
        - nginx
    namespaces:
      include: #默认为空，空值代表选择全部。该值不能与exclude冲突。
      exclude: #默认为空，空值代表不排除任何namespace
    workloadRefs:
    - name: nginx
      namespace: default
      kind: Deployment
      apiVersion: apps/v1
      matchExpressions:
      - key: testkey
        operator: In
        values:
        - testvalue
  nodeGroups:
    - name:  nativegroups
      nodeType: native
      nodeSelectorTerms:
      - matchExpressions:
        - key: node.tke.cloud.tencent.com/instance-charge-type
          operator: In
          values:
          - PrepaidCharge
      nodeResourceFitStrategy:
        type: MostAllocated
        resources:
        - name: cpu
          weight: 1
        - name: memory
          weight: 1
      whenUnsatisfiable: ScheduleAnyway
      priority: 100
      max: 4
    - name:  supergroups
      nodeType: super
      whenUnsatisfiable: ScheduleAnyway
      priority: 0
```

字段说明：

- **targets**：选择该策略的生效范围，支持按 pod、workload、namespace 三种维度进行筛选。targets下的所有字段取交集。若全为空，表示选择集群中所有非 DaemonSet pod。
  - **podSelectors**：可选，通过 pod label 选择 pod，多个 matchExpressions 条件取并集。
    - matchExpressions：matchExpressions 若指定了多个key，需同时满足。
  - **namespaces**：可选，选择特定命名空间下的 namespace。
    - include: 默认为空，空值代表选择全部。该值不能与 exclude 冲突。支持数组。
    - exclude: 默认为空，空值代表不排除任何 namespace。支持数组。
  - **workloadRefs**：可选，选择特定的 workload 。支持指定 workload name、namespace、类型等条件筛选。
    - name：可选，为空表示不限制。
    - namespace：可选，为空表示不限制。
    - kind：可选，为空表示不限制，默认为非 DaemonSet pod，支持 Deployment，Statefulset，Job 等。
    - apiVersion：可选，为空表示不限制。
    - matchExpressions：可选，为空表示不限制，该字段仅支持 Deployment。判断 Deployment 的 label 是否满足 matchExpressions。
- **nodeGroups**：必选，指定被调度的资源分组和优先级。
  - **name**：必选，调度资源组名称。
  - **nodeType**：必选，指定该组的节点类型，可选值为 native, super, general；native 代表原生节点，super 代表超级节点，general 代表普通节点；一个调度资源组仅支持配置1个节点类型。
  - **nodeSelectorTerms**：可选，支持 nodeSelectorTerms 的方式选择节点，目前仅支持 matchExpressions 的表述方式，多个 matchExpressions 满足任一即可。
    - matchExpressions：matchExpressions 若指定了多个 key，需同时满足。
  - **nodeResourceFitStrategy**：可选，定义该节点组里根据节点资源情况计算节点的分数。不设置则表示默认不开启。它包含以下子字段：
    - type：必选，指定评分策略类型，可选值支持：LeastAllocated 和 MostAllocated；LeastAllocated 代表优先选择已分配资源较少的节点，MostAllocated 代表优先选择已分配资源较多的节点。
    - resources：可选，指定参与节点资源评分的资源名称（如 cpu、memory）以及每种资源在评分计算中的权重。由一个列表组成，每个元素包含以下两个字段：
      - name：资源名称（字符串）。可以是标准资源（如 cpu、memory），也可以是扩展资源（如 nvidia.com/gpu）。
      - weight：权重值（整数）。表示该资源在评分计算公式中的相对重要性，范围通常为 1 到 100。
  - **whenUnsatisfiable**：可选，定义无法满足该节点组约束时的行为。可选值支持：DoNotSchedule 和 ScheduleAnyway。默认为 ScheduleAnyway。
    - DoNotSchedule：如果无法满足该节点组约束，Pod 将保持在 `Pending` 状态，直到有足够的资源满足约束。

- ScheduleAnyway：即使无法满足节点组约束，Kubernetes 调度器仍然会尝试调度该 Pod 到不满足该约束的节点上。
  - priority：ScheduleAnyway 的子属性，指定该组的优先级，范围[0,1000]。
- **max**：可选，该 Pod 所在的 workload 最多有N个副本在这个节点组，取值范围[0,1000]。该字段可用于高可用场景，如：指定 workload 在某个 zone 上最大副本数。也常用于降本场景，如：指定稳态资源池最大副本数，超出后调度至弹性节点池。

作用范围

- 命中 PlacementPolicy 策略的新建工作负载，将立刻生效。
- 命中 PlacementPolicy 策略的存量工作负载，对于新扩容的 Pod 将立刻生效；或者工作负载重建后全部生效。
- 当一个对象匹配多个 PlacementPolicy 策略时，优先级以创建时间最新的为准。

使用场景示例

场景一：降本场景（基于节点池的调度策略）

假设您在集群中使用了原生节点，并划分了两个节点池 pool1 和 pool2。出于业务考虑，您希望 Pod 优先调度到 pool1 节点池，资源不足时，再调度至 pool2 节点池。

步骤1：创建 PlacementPolicy 定义弹性资源优先级

```
apiVersion: scheduling.crane.io/v1alpha1
kind: PlacementPolicy
metadata:
  name: testpolicy
spec:
  targets:
    namespaces:
      include:
        - default
    podSelectors:
      - matchExpressions:
        - key: app
          operator: In
          values:
           - nginx
  nodeGroups:
    - name:  pool1
      nodeType: native
      nodeSelectorTerms:
        - matchExpressions:
          - key: node.tke.cloud.tencent.com/machineset
            operator: In
            values:
            - np-9559uqj3
```

```
      whenUnsatisfiable: ScheduleAnyway
      priority: 100
    - name:  pool2
      nodeType: native
      nodeSelectorTerms:
        - matchExpressions:
          - key: node.tke.cloud.tencent.com/machineset
            operator: In
            values:
            - np-eaw5t0en
      whenUnsatisfiable: ScheduleAnyway
      priority: 0
```

步骤2：部署业务（如 nginx）

> **注意：**
> 应用需要满足 PlacementPolicy 中目标 targets 的要求，如在上述场景中：label 指定 key 为 app，value 为 nginx。

例如，使用以下 YAML 内容创建 Deployment，部署2个 Pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - env:
        - name: PATH
          value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        - name: NGINX_VERSION
          value: 1.27.4
        - name: NJS_VERSION
```

```
          value: 0.8.9
        - name: NJS_RELEASE
          value: 1~bookworm
        - name: PKG_RELEASE
          value: 1~bookworm
        - name: DYNPKG_RELEASE
          value: 1~bookworm
        image: ccr.ccs.tencentyun.com/halewang/nginx:1.27.4
        name: nginx
        resources:
          limits:
            cpu: 2
          requests:
            cpu: 2
```

执行以下命令，创建应用 Nginx。

```
kubectl apply -f nginx.yaml
```

执行以下命令，查看部署结果。

```
kubectl get pods -o wide
```

预期输出：
Pod 优先调度到 pool1 下的节点（10.2.2.8和10.2.2.4节点属于节点池 pool1）。

```
j[redacted] kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE        NOMINATED NODE   READINESS GATES
nginx-f4876dbb6-9xbbw    1/1     Running   0          3m21s   10.2.2.29    10.2.2.8    <none>           <none>
nginx-f4876dbb6-cnjxm    1/1     Running   0          3m21s   10.2.2.48    10.2.2.4    <none>           <none>
```

步骤3：扩容您的业务，确认 PlacementPolicy 已生效

将工作负载副本数从 2 调整至 4。当 pool1 节点池中的资源不足时，调度至次优先级的节点池 pool2。
其他确认方式请参见 排查手段 部分。

场景二：降本场景（基于计费方式的调度策略）

您仅使用了原生节点，资源池中有两种计费类型的资源：按量计费和包年包月的计费方式，您希望 Pod 优先调度至包年包月的节点上，不满足后再调度至按量计费的节点上。缩容时，优先缩容按量计费的节点，最终实现成本最优的效果。

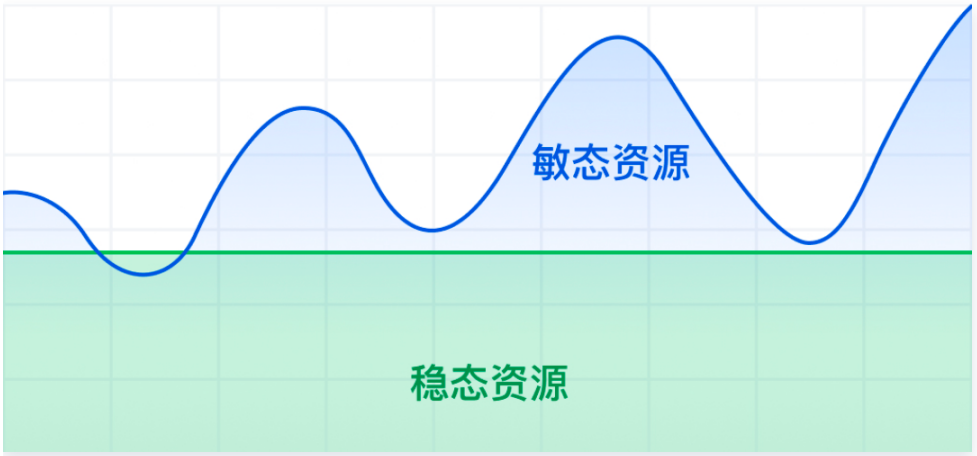
```
apiVersion: scheduling.crane.io/v1alpha1
kind: PlacementPolicy
metadata:
  name: chargetypepolicy
spec:
  targets:
```

```
    namespaces:
      include:
        - default
  nodeGroups:
    - name:  prepaid #包月计费
      nodeType: native
      nodeSelectorTerms:
        - matchExpressions:
          - key: node.tke.cloud.tencent.com/instance-charge-type
            operator: In
            values:
            - PrepaidCharge
      whenUnsatisfiable: ScheduleAnyway
      priority: 100
    - name:  postpaid #按量计费
      nodeType: native
      nodeSelectorTerms:
        - matchExpressions:
          - key: node.tke.cloud.tencent.com/instance-charge-type
            operator: In
            values:
            - PostpaidByHour
      whenUnsatisfiable: ScheduleAnyway
      priority: 0
```

其他步骤如场景一，不再赘述。

场景三：流量洪峰/潮汐场景（基于节点类型的调度策略）

通常您会维持基于原生节点的稳态资源池，用于您日常的资源运维。在流量洪峰的场景下，会调度至弹性资源池，进行快速资源扩容。您可以设置资源池的优先级，优先调度至稳定资源池，有流量洪峰的敏态场景下，调度至弹性资源池。



例如：
您命名空间 pro 下的应用具有流量洪峰的特性，您希望：

- **日常情况下**：Pod 优先调度至原生节点类型的稳态资源池。由于您对资源池的用途进行了划分，您希望该业务均调度至 id 为：np-isagw4ni，np-manr12ry 的资源池上。
- **突发流量的情况下**：pod 调度至超级节点类型的弹性资源池，ID 为 np-flr86fpw。

```
apiVersion: scheduling.crane.io/v1alpha1
kind: PlacementPolicy
metadata:
  name: testpolicy
spec:
  namespaces:
    include:
      - pro
  nodeGroups:
    - name:  native_node_pool # 稳态资源池
      nodeType: native
      nodeSelectorTerms: # 如对原生节点资源池有特定要求，可设置node pool id属性
        - matchExpressions:
          - key: node.tke.cloud.tencent.com/machineset
            operator: In
            values:
            - np-isagw4ni
            - np-manr12ry
      whenUnsatisfiable: ScheduleAnyway
      priority: 100
    - name:  super_node_pool # 敏态资源池
      nodeType: super
      nodeSelectorTerms:
        - matchExpressions:
          - key: tke.cloud.tencent.com/nodepool-id
            operator: In
            values:
            - np-flr86fpw
      whenUnsatisfiable: ScheduleAnyway
      priority: 0
```

其他步骤如场景一，不再赘述。

> ⚠ **注意：**
> 上述仅为示例，实际的稳态资源节点类型和节点池以您实际业务使用为准。如您同时使用多种节点类型，请根据实际诉求配置 priority。

排查手段

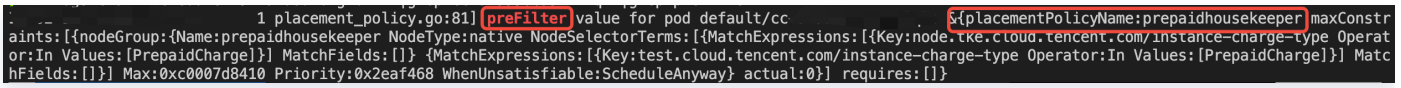
如何得知调度规则已在调度过程中生效?

可以从 crane-scheduler 的日志中找到相关逻辑：

1. **获取 Pod 生效的预选策略：**

```
kubectl -nkube-system logs $craneSchedulerPodName | grep preScore
```

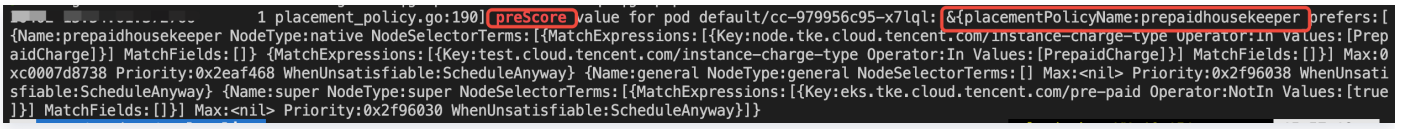
如：PlacementPolicyName 为 prepaidhousekeeper。



2. **获取 Pod 生效的优选策略：**

```
kubectl -nkube-system logs $craneSchedulerPodName | grep preScore
```

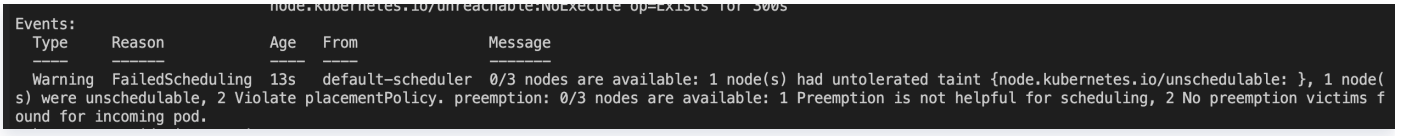
如：PlacementPolicyName 为 prepaidhousekeeper。



3. **确认 Pod 无法调度的原因是否是因为 PlacementPolicy 的拦截：**

```
kubectl describe po $pendingPodName
```

关注输出中的关键词：Violate placementpolicy。



精细调度
QoSAgent

最近更新时间：2024-12-25 15:45:11

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。

> **注意：**
> QoS 相关的能力仅支持在 原生节点 上使用，若您的节点不是原生节点，或工作负载不在原生节点上，相关能力无法生效。

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
avoidanceactions.ensurance.crane.io	CustomResourceDefinition	-	-
nodeqoss.ensurance.crane.io	CustomResourceDefinition	-	-
podqoss.ensurance.crane.io	CustomResourceDefinition	-	-
timeseriespredictions.prediction.crane.io	CustomResourceDefinition	-	-
kube-system	Namespace	-	-
all-be-pods	PodQOS	-	kube-system
qos-agent	ClusterRole	-	-
qos-agent	ClusterRoleBinding	-	-
crane-agent	Service	-	kube-system
qos-agent	ServiceAccount	-	kube-system
qos-agent	Daemonset	-	kube-system

功能说明

功能	说明

CPU 使用优先级	CPU 使用优先级的功能可以通过对工作负载设置优先级，保证高优先级业务在发生资源竞争时的资源供给量，并压制低优先级业务。详情见 CPU 使用优先级 。
CPU Burst	CPU Burst 可以临时给延迟敏感型应用提供超过 Limit 数量的资源，保证其稳定性。详情见 CPU Burst 。
CPU 超线程隔离	避免高优先级容器线程的 L2 Cache 受到运行在同一个 CPU 物理核上的低优先级线程的影响。详情见 CPU 超线程隔离 。
内存 QoS 增强	全方位提升内存表现，以及灵活限制容器对内存的使用。详情见 内存 QoS 增强 。
网络 QoS 增强	全方位提升网络表现，以及灵活限制容器对网络的使用。详情见 网络 QoS 增强 。
磁盘 IO QoS 增强	全方位提升磁盘表现，以及灵活限制容器对磁盘的使用。详情见 磁盘 IO QoS 增强 。

QoS Agent 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。

权限场景

功能	涉及对象	涉及操作权限
读取 podqos、nodeqos、时间序列等配置。	podqoss / nodeqoss / avoidanceactions	get/list/watch/update
查看当前节点的 pod 信息。	pod	get/list/watch
根据 Podqos 开启隔离能力 / 修改 node resource 以增加离线资源。	pod status	update/patch
给 node 添加污点。	node	get/list/watch/update
根据隔离开启情况、资源干扰情况发送 event。	event	所有权限

权限定义

```
rules:
 - apiGroups:
     - ""
   resources:
```

```
      - pods
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - pods/status
    verbs:
      - update
      - patch
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
      - list
      - watch
      - update
  - apiGroups:
      - ""
    resources:
      - nodes/status
      - nodes/finalizers
    verbs:
      - update
      - patch
  - apiGroups:
      - ""
    resources:
      - pods/eviction
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
```

```
      - events
    verbs:
      - "*"
  - apiGroups:
      - "ensurance.crane.io"
    resources:
      - podqoss
      - nodeqoss
      - avoidanceactions
    verbs:
      - get
      - list
      - watch
      - update
  - apiGroups:
      - "prediction.crane.io"
    resources:
      - timeseriespredictions
      - timeseriespredictions/finalizers
    verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
  - apiGroups:
      - "topology.crane.io"
    resources:
      - "noderesourcetopologies"
    verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
```

部署方式

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 **QoS Agent**。
5. 单击**完成**即可安装组件。

> **注意：**
> 在部署完成之后，因为集群的 cgroup 驱动可能不同，需要您手动选择对应的驱动。操作方式如下：
> 1. 在集群里的**扩展组件**里，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
> 2. 在修改 QoS Agent 的组件配置页面，选择 cgroupDrive 选项右侧的下拉框，选择与自己集群匹配的 cgroupDrive。
> 3. 单击**完成**。

常见问题

如何确认自己集群的 cgroupDrive?

集群的 cgroupDrive 只可能是 cgroupfs 或 systemd。确认方式如下：
首先查看集群的运行时，可以在集群的“基本信息”页里的“运行时组件”判断当前集群是 docker 还是 containerd。
如果运行时是 docker 的话，在集群的任意节点上执行 `docker info` 查看 `Cgroup Driver` 的字段内容。
如果运行时是 containerd，在集群的任意节点的 /etc/containerd/config.toml 文件里，如果字段：`SystemdCgroup = true`，代表是 systemd，否则是 cgroup。

如何选择作用的业务或者节点?

支持通过 `label` 或者 `scope` 选择某种资源对象。

> **注意：**
> 当同时存在下面两种 selector 的时候，会取“与”，也即满足所有条件。

labelSelector

labelSelector 通过关联资源对象的 label 对资源对象进行筛选，常用的使用方式是，业务侧在指定的工作负载上打上特定的标签，并将该标签提供给运维侧，运维在创建 PodQOS 时通过 labelSelector 字段关联该标签，即可赋予不同的业务不同的 QoS 能力。

scopeSelector

scopeSelector 由多个 MatchExpressions 组成，这些 MatchExpressions 之间是“与”的关系，
MatchExpressions 中有三个字段，分别是 ScopeName，Operator 和 ScopeName 对应的 Values；
其中 ScopeName 包括 QOSClass，Priority，Namespace 三种；

- QOSClass 是指希望关联具有特定的 QOSClass 的 Workload，Values 可以填：Guaranteed，Burstable，BestEffort 中的一种或多种；
- Priority 是指希望关联具有特定的 Priority 的 workload，Values 可以填特定的 priority 数值，如["1000", "2000-3000"]，支持 priority 范围；
- Namespace 是指希望关联特定的 Namespace 的 Workload，Values 可以填一个或多个。

Operator 包含两种，分别是 In 和 NotIn，不填默认为 In。
如下述示例，表示将满足 app-type=offline 的 BestEffortPod，CPU 优先级设置为7：

```yaml
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: offline-task
spec:
  allowedActions:
  - eviction
  resourceQOS:
    cpuQOS:
      cpuPriority: 7
  scopeSelector:
    matchExpressions:
    - operator: In
      scopeName: QOSClass
      values:
      - BestEffort
  labelSelector:
    matchLabels:
      app-type: offline
```

CPU 使用优先级

最近更新时间：2024-10-29 15:56:52

功能介绍

部署服务过程中，通常会将一类业务进程部署到某台服务器上，服务器的 CPU 利用率取决于业务进程的压力情况，一般情况下，服务器的整体 CPU 利用率往往比较低。为了提高服务器 CPU 的利用率，可以在服务器上混合部署一些优先级较低的任务，这样可以显著提高整机的 CPU 利用率，降低硬件成本。这里定义混部到机器上的优先级较低的任务称之为“离线任务”或是“低优先级任务”。
混合部署解决了 CPU 利用率低的问题，但是会导致混布任务之间的 CPU 争抢，从而使得原先的重要业务指标（运行时长，延迟等）受到影响。为了解决这个问题，CPU QoS 引入了绝对抢占机制，允许重要业务进程在可运行的时候无条件抢占不重要业务进程，保证在提高整机 CPU 利用率的同时，重要业务进程不受影响。

基本原理

用户可以通过给任务/容器设定不同的优先级，来指定内核对任务/容器的调度行为。例如优先级为0的容器和优先级为7的容器同时运行在同一个 CPU 上时，优先级为0的容器只要可以运行，总是会无条件抢占优先级为7的容器，直至高优先级容器进程运行完成或是阻塞休眠，低优先级容器中的进程才会被调度执行。
QoS Agent 支持用户设置8个不同级别（0-7）的优先级，优先级为0-6之间的容器则不存在绝对抢占的关系，按照 CPU 的使用比例进行资源的划分和限流。两个业务，一个优先级设置成0，另一个设置成6，之间不会因为优先级划分资源。划分资源只是在节点资源发生竞争时，不同的 Pod 按照 Request 的资源申请量进行比例划分，详情可参见 文档 。
将一些离线作业优先级设置为7，在提高机器利用率的同时，保证高优和延迟敏感的在线业务不受离线业务的影响。
不同优先级之间 CPU QoS 行为如下：

<table>
<tr><th></th><th colspan="6">0 1 2 3 4 5 6</th><th>7</th></tr>
<tr><td>0</td><td colspan="6" rowspan="7">CPU比例划分
限流</td><td rowspan="7">绝对
抢占</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td><td colspan="6">让出CPU</td><td>CPU比例划分
限流</td></tr>
</table>

使用方式

1. 部署 QoS Agent 。

2. 在集群里的**组件管理**页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **CPU 使用优先级**。
4. 单击**完成**。
5. 部署离线业务。
6. 部署关联离线业务的 PodQOS 对象，选择需要降低优先级的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: low
spec:
  labelSelector:
    matchLabels:
      k8s-app: low # 选择需要降低优先级的业务的 Label
  resourceQOS:
    cpuQOS:
      cpuPriority: 7 # 将指定业务的 cpuPriority 设置为7，7表示优先级最低，发生资源
竞争时会被优先抢占
```

CPU Burst

最近更新时间：2024-12-25 14:30:12

功能介绍

部分业务使用 Quota 限制 cgroup 占用 CPU 的份额，在有的场景下，cgroup 会达到 Quota 的限制，此时会导致业务的性能受到影响。CPU Burst，可以让 cgroup 中的进程在达到 Quota 限制时，可以超出一定的 Quota 限额，从而达到提高业务性能的目的。

基本原理

因为 cgroup 配置一定的 Quota 比例之后，并不一定每个周期都100%的使用了 Quota 配额，当有的周期没有使用完 Quota 时，将对应的 Quota 收集起来，在该 cgroup 消耗完 Quota 时，能继续使用之前收集的 Quota，CPU Burst 功能就是利用之前周期未使用的 Quota，这样长时间来看，该 cgroup 是并没有超过整体的 Quota 的。能在需要更多的 CPU 时间的某些周期里边，突破 Quota 的限制，达到提升业务性能的目的。

使用方式

1. 部署 QoS Agent。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **CPU Burst**。
4. 单击**完成**。

部署 PodQOS 对象，选择作用的 Pod 和 burstQuota，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: burst
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql # 作用的 Pod
  resourceQOS:
    cpuQOS:
      cpuBurst:
        burstQuota: "40.0" # burstQuota 表示该 Pod 能 Burst 到的最大核数
```

最佳实践

部署业务：

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  labels:
    k8s-app: mysql
    qcloud-app: mysql
  name: mysql
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: mysql
      qcloud-app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        k8s-app: mysql
        qcloud-app: mysql
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: mysql
        image: mysql:5.6.51
        imagePullPolicy: Always
        name: mysql
        resources:
          limits:
            cpu: "2"
            memory: 8Gi
          requests:
            cpu: "2"
            memory: 8Gi
        securityContext:
          privileged: false
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: data
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
```

```
      restartPolicy: Always
      hostNetwork: true
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:
      - hostPath:
          path: /data2/mysql
          type: DirectoryOrCreate
        name: data
```

```
# 登录 mysql，IP 为高优 Pod 的 IP
mysql -u root -h $IP  -pmysql

# 创建对应的数据库
CREATE DATABASE sbtest

# 在主机上执行
sysbench oltp_read_only --tables=10 --table_size=3000000 --mysql-host=$IP --
mysql-user=root --mysql-password=mysql --threads=10 --mysql-db=sbtest prepare
```

```
# 另一台主机上执行如下命令进行压测，mysql-host是高优pod的ip
# 1. 长期压力
sysbench oltp_read_only --tables=10 --table_size=3000000 --mysql-host=$IP --
mysql-user=root --mysql-password=mysql --mysql-db=sbtest --time=300 --
threads=2 --report-interval=1 run

# 2. 突发压力
#!/bin/bash
while [ 1 ]
do
    sysbench oltp_read_only --tables=10 --table_size=3000000 --mysql-host=$IP
--mysql-user=root --mysql-password=mysql --mysql-db=sbtest --time=1 --
events=800 --threads=20 --report-interval=1 run
    sleep 10
done

# 压测结束会获取 Latency 信息
```

开启 Burst：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
```

```
metadata:
  name: burst
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql
  resourceQOS:
    cpuQOS:
      cpuBurst:
        burstQuota: "40.0"
```

查看 Pod annotation， burstQuota 设置成功：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    tke.cloud.tencent.com/burst-qos: '{"quota":40}'
    tke.cloud.tencent.com/cpu-qos: '{"priority":null,"containers-priority":null}'
    tke.cloud.tencent.com/qos-blkio: '{"weight":0,"read-iops":0,"write-iops":0,"read-bps":0,"write-bps":0}'
  creationTimestamp: "2022-08-10T07:42:13Z"
  generateName: mysql-7c8c7f64d7-
  labels:
    k8s-app: mysql
    pod-template-hash: 7c8c7f64d7
    qcloud-app: mysql
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:generateName: {}
        f:labels:
          .: {}
          f:k8s-app: {}
          f:pod-template-hash: {}
          f:qcloud-app: {}
        f:ownerReferences:
          .: {}
          k:{"uid":"3a513adc-966d-4768-a48d-48aa028f2533"}:
            .: {}
```

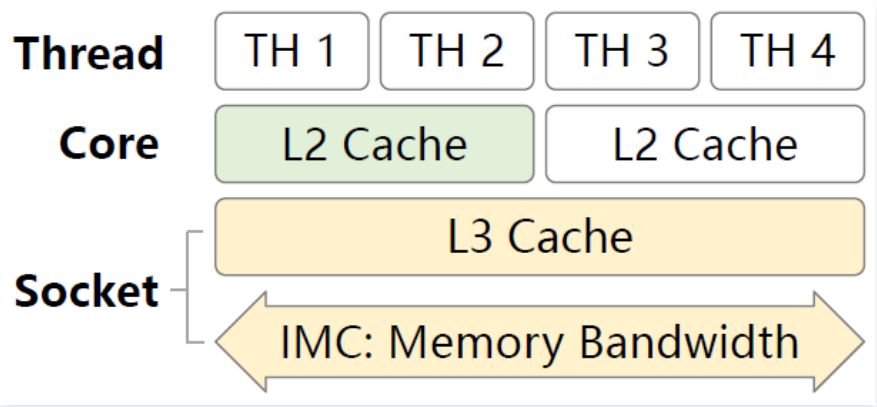
再次用上文提到的长期压力和突发压力进行压测，获取 Latency 信息。

CPU 超线程隔离

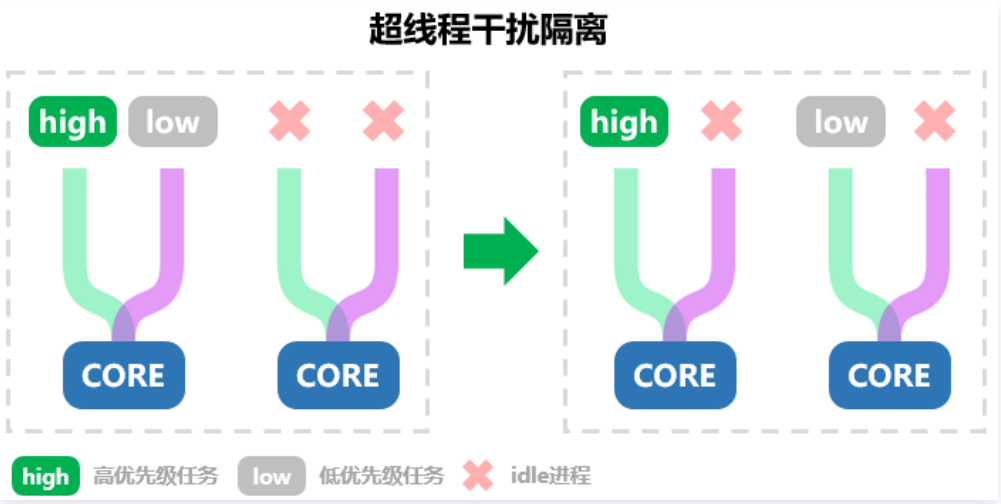
最近更新时间：2024-12-25 16:27:11

功能介绍

常见的 SMP 处理器存在多个层级，包含线程、物理核、处理器等结构，在开启超线程的情况下，一个物理核上一般会包含2个线程，且同核上的2个线程共享 L2 Cache。



当高优先级容器与低优先级容器同时运行时，可能会出现高优先级容器的线程与低优先级容器的线程在同一个物理核甚至在同一个超线程 CPU 上执行的情况，在这种情况下，虽然 CPU 使用优先级 能保证高优先级容器线程总能抢占低优先级容器的线程，但是只要低优先级容器线程运行，就会占用物理核上共享的 L2 Cache，导致高优先级容器线程的 L2 Cache 受到影响。



为了避免高优先级容器线程的 L2 Cache 受到运行在同一个物理核上的低优先级线程的影响，QoS Agent 引入了超线程隔离机制。在处理器资源富余的情况下，保证高优先级容器线程所在物理核上没有低优先级容器线程的干扰。
在不同场景下的超线程干扰隔离策略：

Thread 1	Thread 2	Action
高优先级	高优先级	不采取任何行为
高优先级	低优先级	将 sibling 上的低优先级任务限流并限制负载均衡时低优先级任务迁移到 sibling 上

低优先级	低优先级	尝试拉取其它核上的高优先级任务

使用方式

1. 部署 QoS Agent 。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **CPU 超线程隔离**。
4. 勾选 **CPU 使用优先级**，用于标识高优先级业务。
5. 单击**完成**。
6. 部署业务。
7. 部署关联该业务的 PodQOS 对象，选择需要使用超线程隔离的 Workload 的 Label，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: ht-1
spec:
  labelSelector:
    matchLabels:
      k8s-app: memcached # 选择需要降低优先级的业务的 Label
  resourceQOS:
    cpuQOS:
      cpuPriority: 0 # 标识为高优先级业务
      htIsolation:
        enable: true # 开启 CPU 超线程隔离能力。enable 可取 true/false，代表针对
PodQOS 关联到的业务是否开启超线程隔离
```

应用启动时 CPU 突增

最近更新时间：2024-12-25 10:31:21

简介

在通常情况下，应用程序在发布或重启过程中可能会出现 CPU 使用率波动增高，甚至可以消耗掉集群所有资源的现象。这主要是因为在应用程序启动时，Java虚拟机（JVM）需要重新进行类加载和对象初始化操作，导致 CPU 在整个过程中承担更多的编译任务。
在 Kubernetes 的场景中，应用程序都是通过 Pod 中的容器部署，每个容器都可以设置 Request 和 Limit 来控制资源使用的下限和上限。如果 Limit 设置的过低，就容易导致上述应用启动缓慢，甚至因为资源不够无法启动；如果 Limit 设置得过高，那后续在应用的运行过程中，又容易使用过量资源，与其他应用形成资源竞争关系。

功能说明

使用方式

设置应用的启动时间，以及 Limit 扩大的倍数，在应用启动的初始阶段增加应用使用资源的上限。

注意事项

- 业务启动时资源突增的能力依赖原生节点的内核，该功能仅支持在 原生节点 中使用。
- 在使用该功能之前，需要提前安装 QoS Agent。确保 QoS Agent 组件版本在1.1.4及以上。

功能原理

在 Kubernetes 中，资源限制（Limit）是通过 CPU cgroup 控制模块中的 cpu.cfs_period_us 和 cpu.cfs_quota_us 两个配置来实现的。Kubernetes 会为容器的 cgroup 配置这两个信息。通过调整 Pod 所在节点的 cgroup 中 cpu.cfs_quota_us 的数值，实现临时资源突增，从而增加应用在启动阶段的资源使用上限。

使用方式

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 **QoS Agent**。
5. 单击**完成**即可安装组件。

开启应用启动时资源突增能力

1. 在**组件管理**页面，查看 QoS Agent 组件版本。确保 QoS Agent 组件版本在1.1.4及以上。如果版本低于此要求，请单击组件右侧的**升级**进行升级。如下图所示：

2. 通过 kubectl 创建 PodQoS CRD 对象，将其应用到需要启动突增的工作负载上。示例代码如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: exceed
spec:
  labelSelector:
    matchLabels:
      app: 0kusa3 # 匹配到需要启动突增的工作负载上的标签
  resourceQOS:
    cpuQOS:
      limitExceed:
        time: 5     # 应用启动多长时间需要突增 Limit。单位：mins
        value: "1.2"     # Limit 突增的倍数
```

3. 创建工作负载。示例代码如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: 0kusa3     # 和 PodQoS 对象中的 matchLabels 要保持一致
spec:
  replicas: 3
  selector:
    matchLabels:
      app: 0kusa3
  template:
    metadata:
      labels:
        app: 0kusa3
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

4. 登录 Pod 所在节点，在节点上查看容器对应的 cfs_quota 数值。

路径：

```
/sys/fs/cgroup/cpu/kubepods/burstble/pod<id>/<containerid>/cpu.cfs_quota_us
```

示例：

```
/sys/fs/cgroup/cpu/kubepods/burstable/pod44ad741a-cbe6-48a2-a5b7-
b6920df2fe40/645a0a88f23c2aca4b207ec2c3d83bbe760a5f2a4db812938042df5cb5054e
18/cpu.cfs_quota_us
```

其中：

- pod<id>：<id>来自 Pod YAML 中的 UID 字段。
- <containerid>：来自 Pod YAML 中的 containerID 字段。

在创建 PodQoS 之前，cpu.cfs_quota_us 的值为 cpu.cfs_period_us * Limit，例如：100000 * 0.5核 = 50000。

```
[[root@VM-21-7-centos 645a0a88f23c2aca4b207ec2c3d83bbe760a5f2a4db812938042df5cb5054e18]# cat cpu.cfs_quota_us
50000
```

在创建 PodQoS 之后，当 Pod 重建后的前 5 分钟内，新的 cpu.cfs_quota_us 值为旧值乘以 1.2，即 Limit 被放大了。

> **注意：**
> Pod 重建后 Pod ID 会变化。

```
[[root@VM-21-7-centos 197dbd54d23a17ebded1dea9c21e644c2a1bb17b26a727a8701848d612aa179b]# cat cpu.cfs_quota_us
60000
```

内存精细调度

最近更新时间：2024-12-25 19:13:05

内存精细调度能力提供了一系列功能，保证业务内存方面的服务质量保证。全方位提升内存表现，以及灵活限制容器对内存的使用。

功能一：异步回收

功能介绍

内存异步回收允许容器内部设置一个阈值，超过该阈值则会触发后台异步回收，保证对应 memcg 内使用维持一定量的空闲内存；对后续的内存分配提供保障，减少其进入直接内存回收的次数。

使用方式

1. 部署 QoS Agent 。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **内存 QoS 增强**。
4. 单击**完成**。
5. 部署业务。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: low # 选择作用的业务 Label
  resourceQOS:
    memoryQOS:
      memAsyncReclaim:
        asyncRatio: 90 # asyncRatio代表异步回收水位线，当 cgroup 中内存用量超过这个比例开始回收，取值[0-100]，建议设置90以上；默认为0，表示关闭
        asyncDistanceFactor: 200 # asyncDistanceFactor 控制每次触发异步回收的时候，尝试回收的页面总数，默认为1。取值范围为[1, 150000]
```

功能二：全局水位分级

功能介绍

内存全局水位分级是指针对不同优先级的 cgroup，拥有不同的全局内存水位线；高优先级容器拥有更低的水位线，同等条件下更容易获得内存；低优先级的容器拥有更高的水位线，同等条件下更容易触发回收，进入直接回收流程。

使用方式

1. 部署 QoS Agent。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **内存 QoS 增强**。
4. 单击**完成**。
5. 部署业务。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: test
spec:
  labelSelector:
    matchLabels:
      k8s-app: low # 选择作用的业务 Label
  resourceQOS:
    memoryQOS:
      memWatermark:
        watermarkRatio: 50 # watermarkRatio 取值范围为[-75,75]；负值表示降低水
位，主要针对在线容器；正值表示抬升水位；主要针对离线容器；
```

针对离线业务创建 PodQOS 对象：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql
  resourceQOS:
    memoryQOS:
      memWatermark:
        watermarkRatio: -50
```

功能三：pagecache limit

功能介绍

进行容器级别的 pagecache 隔离。

使用方式

1. 部署 QoS Agent。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **内存 QoS 增强**。
4. 单击**完成**。
5. 部署业务。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql-pi-000006 # 选择作用的业务 Label
  resourceQOS:
    memoryQOS:
      memPageCacheLimit:
        pageCacheMaxRatio: 20 # pageCacheMaxRatio 代表 pagecache 占用内存限额的最大比例，基于当前 memory 的限制值，所以如果要使用这个特性，limits 中必须有 memory 的限制。比如 Pod 内存限制10GB，pageCacheMaxRatio 占20%，就是限制 pagecache 最多使用2GB。
        pageCacheReclaimRatio: 5 # pageCacheReclaimRatio 代表 pagecache 超额后的回收比例，具体是指占 pagecache 最多使用量的比例。
```

网络精细调度

最近更新时间：2024-12-25 10:44:15

网络精细调度能力提供了一系列功能，保证业务网络方面的服务质量保证。全方位提升网络表现，以及灵活限制容器对网络的使用。

功能一：出入方向限速

功能介绍

限制某个容器的入、出带宽。

使用方式

1. 部署 QoS Agent。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选**网络 QoS 增强**。
4. 单击**完成**。
5. 部署业务。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # 选择需要降低优先级的业务的 Label
  resourceQOS:
    netIOQOS:
      netIOLimits:
        rxBps: 50 # rxBps 代表入方向的最大带宽，单位为 Mbps，0表示无限制，最大可以输入13位的整数
        txBps: 50 # txBps 代表出方向的最大带宽，单位为 Mbps，0表示无限制，最大可以输入13位的整数
```

功能二：优先级（带宽绝对抢占）

功能介绍

用户为不同的容器设置不同的优先级，按照优先级分配网口的带宽资源。以三档优先级为例说明网络入带宽的调度策略，可扩展到更多档优先级：

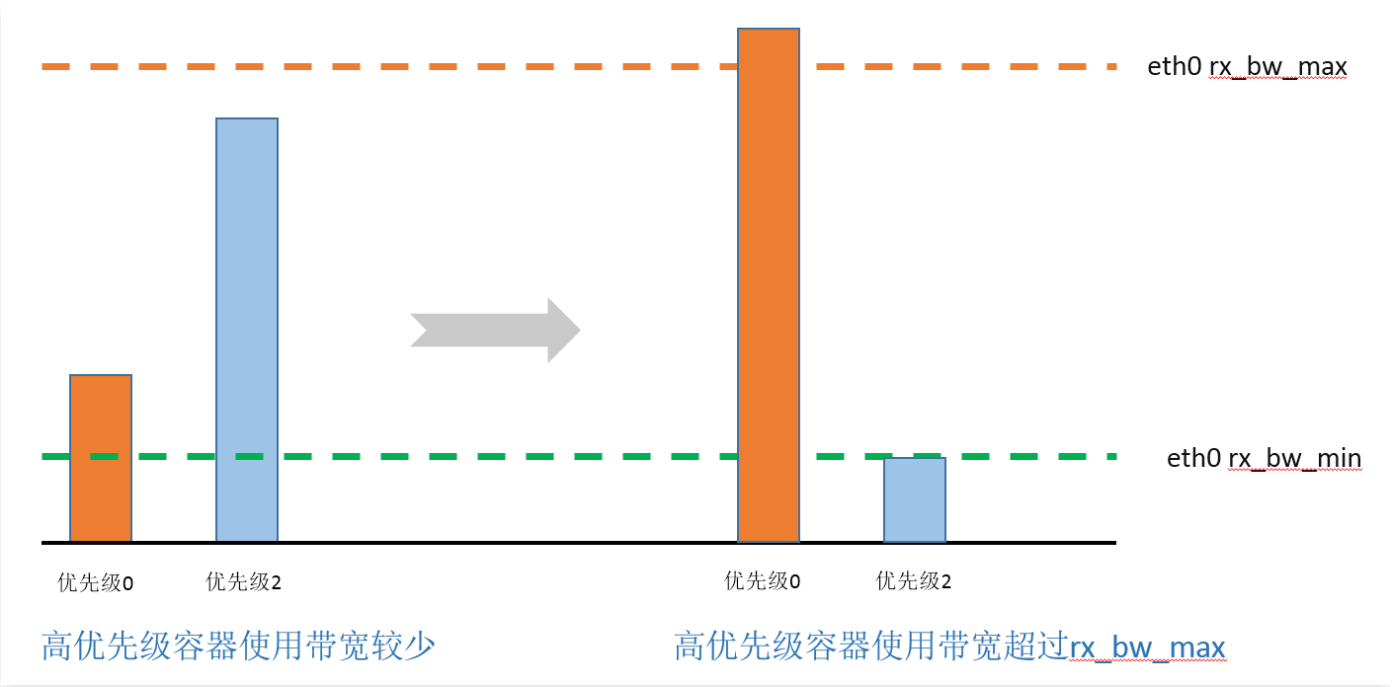
- 最高优先级（优先级为0）的容器不受网络 QoS 限制，可任意使用带宽资源。

- 所有较低优先级（优先级为1、2）容器使用的带宽之和小于 rx_bps_max 减最高优先级带宽时，每个容器可以使用的带宽不受限制，并都可以超过 rx_bps_min。
- 所有较低优先级（优先级为1、2）容器使用的带宽之和大于 rx_bps_max 时，最多可以使用 rx_bps_max 减较高优先级容器的总流量的差值，如果差值小于 rx_bps_min，则保障低优先级容器至少可使用 rx_bps_min 的带宽。
- 最高优先级（优先级为0）容器的带宽可以突破 rx_bps_max/tx_bps_max，较低优先级容器不能突破此参数。

用户场景

场景一

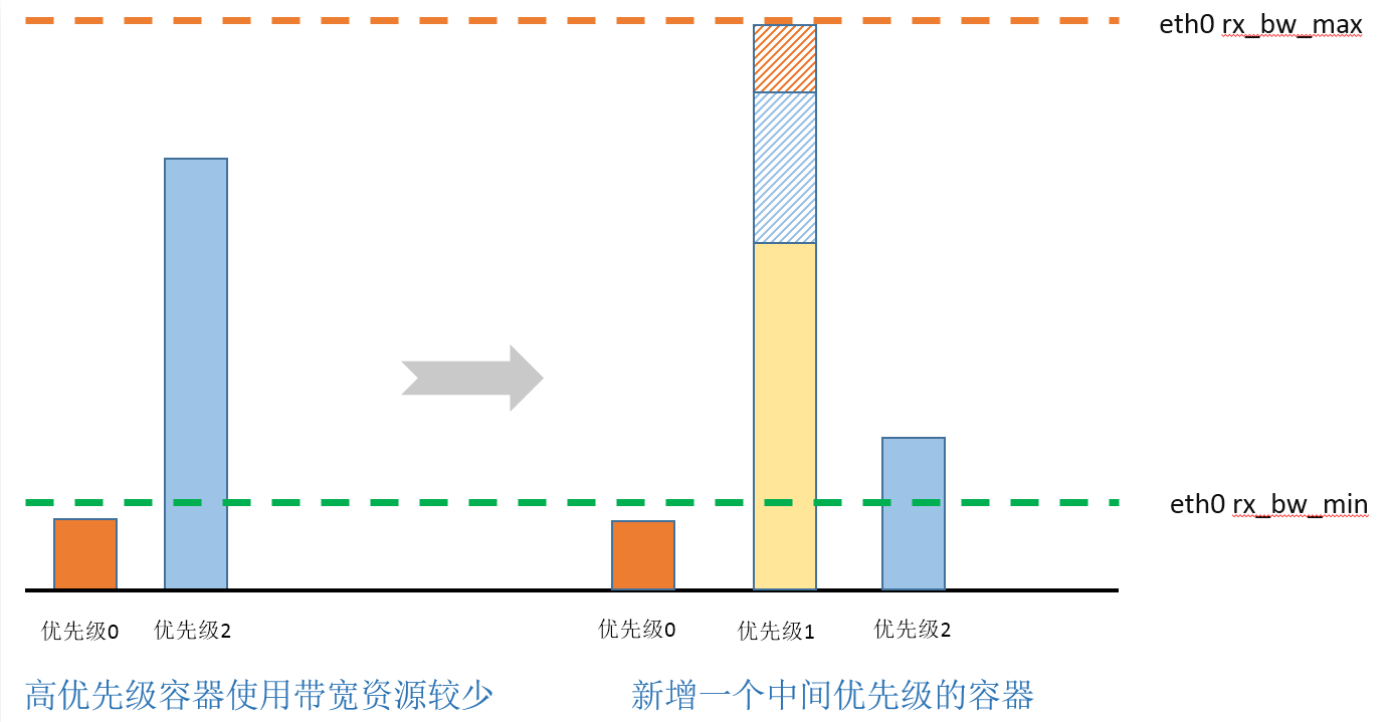
- 高优先级容器使用带宽较低时，将空闲带宽分配给低优先级容器。
- 高优先级容器使用带宽较高，超过 rx_bw_max 时，只给低优先级容器分配最低保障带宽。



场景二

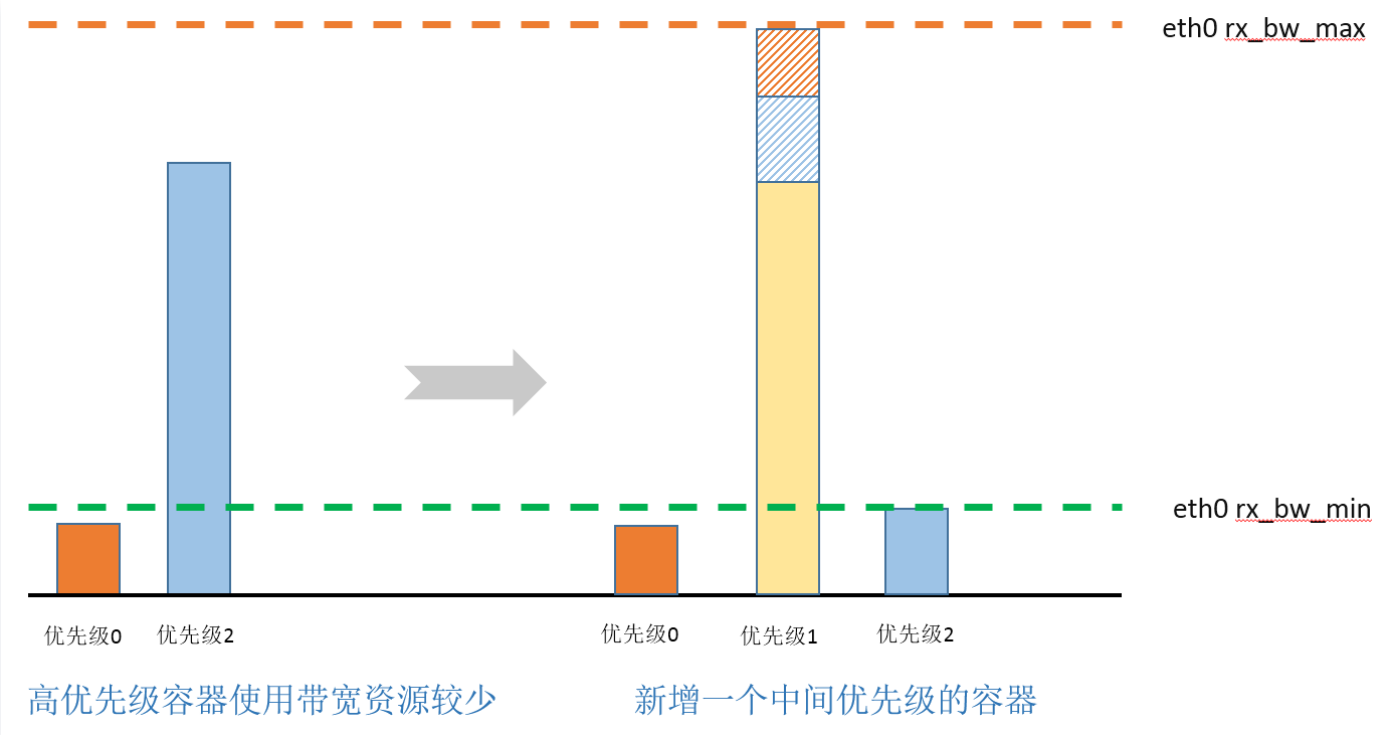
- 高优先级容器使用带宽较低时，将空闲带宽分配给低优先级容器。

- 新增一个中间优先级的容器时，抢占相对较低优先级容器的带宽。



场景三

- 高优先级容器使用带宽较低时，将空闲带宽分配给低优先级容器。
- 新增一个中间优先级的容器时，抢占相对较低优先级容器的带宽。最多能抢占的带宽要减去较高优先级容器所用带宽，还要减去较低优先级容器最低保障带宽。



使用方式

1. 部署 QoS Agent。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **网络 QoS 增强**。
4. 单击**完成**。
5. 部署业务 A。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # 选择业务的 Label
  resourceQOS:
    netIOQOS:
      netIOPriority: 6 # 选择网络优先级，0-7，总共8级
```

7. 部署业务 B。
8. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: b
spec:
  labelSelector:
    matchLabels:
      k8s-app: b # 选择业务的 Label
  resourceQOS:
    netIOQOS:
      netIOPriority: 7 # 选择网络优先级，0-7，总共8级
```

该功能需要搭配 NodeQOS 中的节点带宽限制联合使用，配合 NodeQOS 指定 rxBpsMin，rxBpsMax，txBpsMin，txBpsMax。

- rxBpsMin：per-priority 级别的入方向最低保障带宽。为该级别的容器所共享。目前不同优先级的入方向最低保障带宽相同。优先级0除外。
- txBpsMin：per-priority 级别的出方向最低保障带宽。为该级别的容器所共享。目前不同优先级的出方向最低保障带宽相同。优先级0除外。
- rxBpsMax：网口的最大入带宽。
- txBpsMax：网口的最大出带宽。

> **注意**
> 单位为Mbps。

```
apiVersion: ensurance.crane.io/v1alpha1
kind: NodeQOS
metadata:
  name: total
spec:
  netLimits:
    rxBpsMin: 10
    rxBpsMax: 10240
    txBpsMin: 10
    txBpsMax: 10240
```

功能三：端口白名单

功能介绍

为了防止低优先级容器被饿死，有两种机制：

1. per-priority 的最低保障带宽：由该优先级的容器所共享。
2. 端口白名单机制：用户可以将容器中的某个本地端口、或对端端口添加到白名单，这个端口的流量将不被限流。通常用于保护特定协议的控制报文。

使用方式

1. 部署 QoS Agent 。
2. 在集群里的“扩展组件”页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选 **网络 QoS 增强**。
4. 单击**完成**。
5. 部署业务。
6. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # 选择业务的 Label
  resourceQOS:
    netIOQOS:
```

```
      whitelistPorts: # 代表业务a的本地端口 5201 和远端端口 5205 被设置为白名单端
口，不做限流。lports,rports 默认值都为0，取值范围为0 ~ 13位整数
        lports: 5201
        rports: 5205
```

磁盘 IO 精细调度

最近更新时间：2024-12-25 09:48:53

磁盘 IO 精细调度能力提供了一系列功能，保证业务磁盘方面的服务质量保证。灵活限制容器对磁盘传输的使用量。

使用限制

1. 部署 QoS Agent 。
2. 在集群里的**组件管理**页面，找到部署成功的 QoS Agent，单击右侧的**更新配置**。
3. 在修改 QoS Agent 的组件配置页面，勾选**磁盘 IO QoS 增强**。
4. 输入参数指定限制的磁盘名称，如下图所示，可输入多个：

> **注意：**
>
> 1. 磁盘名要符合实际设备名称，获取节点磁盘名称的方式请参见 如何确定节点的磁盘名 。
> 2. 在控制台输入时，每次输入一个磁盘设备的名称，请单击**添加**，增加新的磁盘名称。请勿一次性输入多个磁盘名称。
> 3. 如果节点上缺少对应的磁盘名，则磁盘 IO 精细调度无法生效。
> 4. 非原生不支持该功能，请确保集群中有原生节点，并且配置了磁盘 IO 限制的 Pod 调度到了原生节点上。

5. 单击**完成**。

功能1：磁盘 IOPS 限制（direct IO + buffer IO）

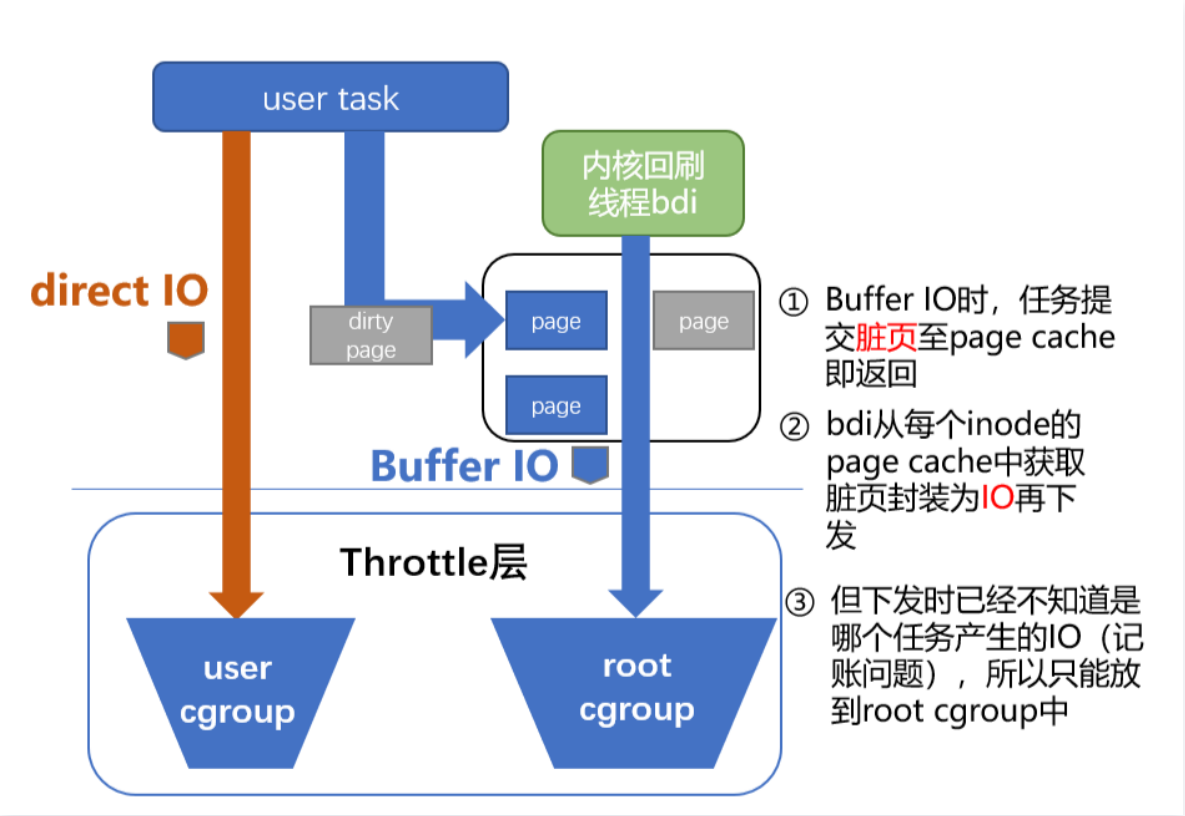
1. 根据上述使用限制部署组件、打开相关开关、输入相关磁盘名称。
2. 部署业务。
3. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
```

```
    matchLabels:
      k8s-app: a # 选择业务的 Label
  resourceQOS:
    diskIOQOS:
      diskIOLimit:
        readIOps: 1024 # readIOps 代表限制 Pod 读 IO 的量，单位为 IOPS/s
        writeIOps: 1024 # writeIOps 代表限制 Pod 写 IO 的量，单位为 IOPS/s
```

针对 buffer IO 的限速机制

由于低版本内核或 cgroup v1 对 Buffer IO 无法有效限速，有可能造成 Buffer IO 干扰业务正常 IO（数据库场景经常使用direct IO）。TKE 基于 cgroup v1 提供了 Buffer IO 限速功能。基于 cgroup v2 的 Buffer IO 限速和上游内核保持一致。
cgroup v1 无法支持限速的主要原因在于异步刷脏页的时候，内核无法得知这个 IO 应该提交给哪个 blkio cgroup。示例如下：



为了解决这个记账问题，TKE 将 page cache 所属的 cgroup 和对应的 blkio cgroup 进行绑定。之后异步刷盘时，内核线程便可以确定目标 blkio cgroup。

功能2：磁盘 BPS 限制（direct IO + buffer IO）

1. 根据上述使用限制部署组件、打开相关开关、输入相关磁盘名称。
2. 部署业务。
3. 部署关联该业务的 PodQOS 对象，选择需要作用的业务，示例如下：

```yaml
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a
  resourceQOS:
    diskIOQOS:
      diskIOLimit:
        readBps: 1048576 # readBPS 限制 Pod 读带宽，单位为 mbps
        writeBps: 1048576 # writeBPS 限制 Pod 写带宽，单位为 mbps
```

常见问题

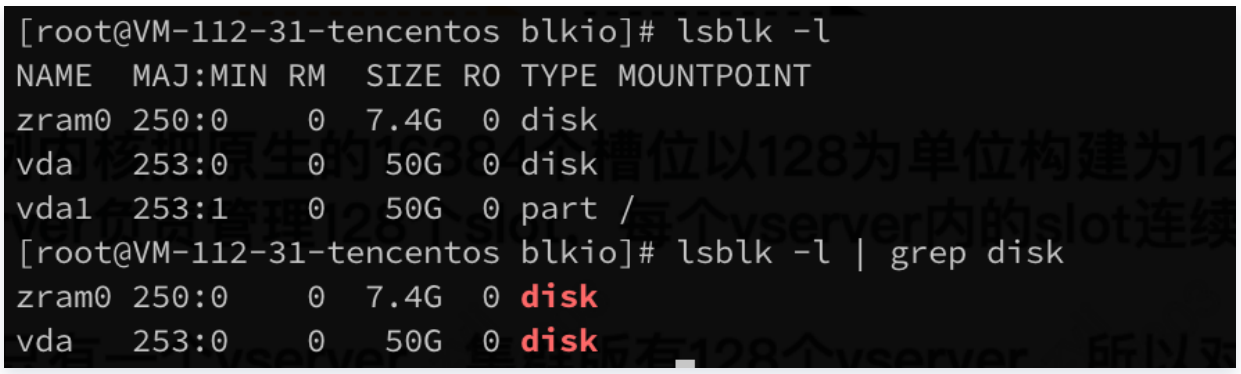
如何确定节点的磁盘名?

1. 登录需要开启磁盘 IO 的原生节点，操作详情请参见 原生节点开启 SSH 密钥登录 。
2. 执行以下命令，以获取所有的磁盘设备：

```
lsblk -l | grep disk
```

执行结果如下图所示：

> **注意：**
> 1. 下图中的 zram0 是用于虚拟内存压缩的模块，不是磁盘名。
> 2. 下图中的 vda1 是一个 part 分区块，也不是磁盘名。
> 3. 下方示例仅展示了一个磁盘，其磁盘名为：vda。



社区 BFQ 问题可能导致原生节点内核 panic

建议：升级 QoS Agent 版本到1.1.2以上，以避免由于 Linux 社区内核 BFQ 模块的 bug 导致原生节点内核发生 panic。相关问题如下：

- kernel/git/stable/linux.git - Linux kernel stable tree
- Re: [PATCH 0/5 v2] bfq: Avoid use-after-free when moving processes between cgroups - Jan Kara

查询方案：您可以通过在节点上执行命令 `cat /sys/block/[磁盘名称]/queue/scheduler` 来确认调度策略，如果输出包含 `[bfq]` 字段，则表示已启用 BFQ，需要进行更改。

修改方案：将调度策略更改为 mq-deadline。例如，如果磁盘名称为 vda，则可以使用以下方式进行修改：

```
echo mq-deadline > /sys/block/vda/queue/scheduler
```