百度智能云

BCI 文档

目录

功能发布记录

发布时间	功能概述
2025-03	支持 BCI自定义Condition 功能
2025-03	支持 忽略Sidecar容器的NotReady状态 功能
2025-03	支持 通过指定标签调度到BCI 功能
2025-03	支持 支持配置BCI-profile 功能
2025-03	支持 设置BCI Pod的故障处理策略 功能
2025-02	支持 自定义设置Pod的最大Pending时长 功能
2025-02	支持 设置容器终止消息 功能
2025-02	支持 设置容器时区 功能
2025-02	支持 配置资源规整时忽略特定容器 功能
2025-02	支持 配置NTP服务 功能
2025-02	支持 强制终止Sidecar容器并忽略容器退出码 功能

产品描述

什么是BCI？

百度智能云容器实例（Baidu Container Instance，即BCI）提供无服务器化的容器资源。您只需提供容器镜像及启动容器所需的配置参数，即可运行容器，而无需关心这些容器如何被调度部署到底层的物理服务器资源中。BCI服务将为您完成IaaS层资源的调度和运维工作，从而简化您对容器的使用流程，降低部署和维护成本。同时BCI只会对您创建容器时申请的资源计费，因此实现真正的按需付费。结合容器本身秒级启动的特性，BCI可以帮助您在百度智能云上构建灵活弹性且易于维护的大规模容器集群。

BCI的使用极为简单，您可以直接在BCI控制台中，或者通过BCI的Open API，发起创建BCI容器组的请求，并对您创建的容器组进行管理。如果需要对大规模的容器进行调度，也可以通过Virtual Kubelet将BCI无缝接入到CCE（百度云容器引擎服务）集群中，从而通过标准的Kubernetes API来进行BCI容器组的管理。

核心概念

容器组

容器组的概念类似于Kubernetes中的Pod，是一组由用户定义的容器集合。容器组中的容器将被调度到同一台宿主机中，共享

网络和存储资源。

容器组的生命周期由其中的所有容器决定，容器组可能出现以下几种状态之一：

- 挂起（pending）：创建容器组的请求已经发起，但有一个或者多个容器镜像尚未创建。挂起状态通常包括包括调度 Pod 、通过网络下载镜像和启动容器的时间。
- 运行中（Running）：容器组中所有容器都已经被创建完成，且至少有一个容器正在运行，或者正处于启动或重启状态。
- 成功（Succeeded）：容器组中的所有容器都被成功终止，并且不会再重启。
- 失败（Failed）：容器组中的所有容器都已终止了，并且至少有一个容器是因为失败终止。也就是说，容器以非0状态退出或者被系统终止。
- 未知（Unknown）：因为某些原因无法取得容器组的状态。

用户可以为容器组指定重启策略，当容器组中的容器终止时，容器组将根据重启策略判断是否重新启动该容器。重启策略包括：

- 总是重启：无论容器以什么状态终止，容器组都会重新启动该容器。
- 失败重启：只有容器因为失败而终止，即以非0状态退出或者被系统终止，容器组会重新启动该容器。
- 从不重启：无论容器以什么状态终止，容器组都不会重新启动容器。

网络

容器组中所有的容器共享相同的IP地址和端口号。用户可以在创建容器组时，为不同的容器指定使用的端口和协议。 容器组存在于用户的VPC中，自动获得VPC内的内网IP。在创建容器组时需要用户选择容器组所在的VPC和子网，以及网络安全组。 容器组也可以挂载EIP，获得访问公网的能力。EIP可以在容器组被创建出来之后挂载，也可以选择不为容器组挂载EIP，通过VPC中的NAT网关进行公网访问。

存储

容器组通过挂载存储卷获得存储能力，组内容器共享容器组中的所有存储卷。目前BCI容器组支持3种存储卷类型：

- NFS：NFS数据卷将网络文件系统挂载到容器组中，提供持久化存储，Server地址需要与容器组在相同VPC或者公网可访问，建议搭配百度云文件服务CFS使用。
- EmptyDir：EmptyDir提供一个容器组内所有容器均能访问的共享路径，当容器组被删除后，EmptyDir上的数据也会被一并删除，适用于容器组运行时的临时数据。
- ConfigFile：ConfigFile用于向容器组中传递配置信息，挂载配置文件后，容器可以直接通过对应路径读取文件内容。

镜像

Docker 镜像是容器应用打包的标准格式，用户需要在创建容器组时为组内每个容器指定镜像。镜像来源可以是DockerHub、百度智能云公共镜像或者用户上传到百度智能云镜像仓库中的自定义镜像。Docker镜像通过镜像地址和版本（tag）唯一标识。

如果在创建容器组时使用了私有镜像，那么需要用户提供镜像仓库对应的用户名和密码，否则创建容器组时将会拉取镜像失败。

产品特性

容器化

使用标准的Docker Image创建容器，支持容器组管理，支持容器重启策略、环境变量、配置文件及启动参数等容器组参数，兼容Kubernetes中的Pod模型。

Serverless

直接在百度智能云中启动容器，只需对容器占用的资源付费，完全无需管理运行容器的宿主机以及集群。

高隔离

BCI容器组基于kata-container技术启动容器，为容器提供虚机级别的高隔离，实现不同容器组之间CPU、内存、网络、磁盘的隔离，保障用户的业务和数据安全性。

灵活高效

相比传统的虚拟机，BCI容器组可以在秒级启动通网，并且快速进行复制。BCI的计费也以秒为单位，从而最大程度降低资源的闲置成本。

兼容Kubernetes

通过百度智能云定制的virtual-kubelet组件，BCI可以接入CCE Kubernetes集群，成为一个虚拟节点。从而通过标准的Kubernetes API进行BCI容器组的调度。

集成云上其它服务

BCI容器组可以直接与云上的网络、存储等服务集成，容器组直接加入VPC并获得内网IP地址，支持安全组控制访问规则，支持挂载EIP获取公网访问能力，支持挂载CFS进行持久化数据存储。

使用限制

BCI使用限制

您的账户必须通过实名认证，否则无法创建BCI实例

配额类型	默认配额
容器组数量	20
每个容器组可添加的数据卷总数	10
每个容器组可添加的NFS数据卷数量	5
每个容器组可添加的emptyDir数据卷数量	5
每个emptyDir磁盘空间大小	10 GB
每个容器组可添加的configFile数据卷数量	5
每个容器可添加的port数量	10
每个容器可添加的环境变量数量	10

产品定价

计费规则

采用按需计费的方式。

- 计费项：CPU、内存。
- 按秒计费，不足1秒钟按1秒钟计。
- 按小时扣费，即北京时间整点扣费并生成账单。出账单时间是当前计费周期结束后1小时内。例如，10:00-11:00的账单会在12:00之前生成，具体以系统出账时间为准。
- 购买实例前需保证账户无欠款，且保证账户余额和可用代金券总和大于或等于100元。

计费公式

费用=(容器组CPU核数×单价+容器组内存大小×单价)×容器组运行时间

注：运行时间指容器组开始创建到运行完成的时间，包括创建中和运行中状态，容器组进入成功或失败状态后即停止计费。

计费价格

关于计费具体信息请参见BCI价格详情。

操作指南

通过CCE使用BCI

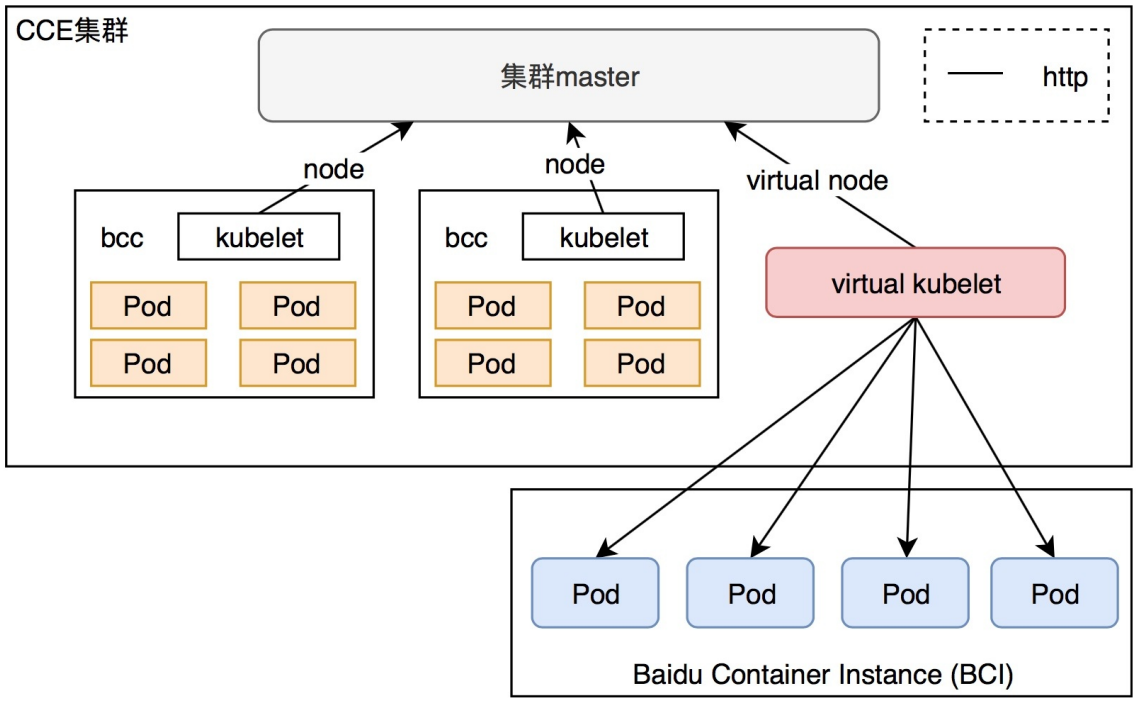
对接BCI

对接概述

BCI能为Kubernetes提供基础的容器Pod运行环境，但业务间的依赖、负载均衡、弹性伸缩、定期调度等能力依然需要Kubernetes来提供。本文为您介绍百度智能云容器引擎(Cloud Container Engine，即CCE)如何与BCI对接，使用BCI作为Pod的运行资源。

对接方式 BCI为Kubernetes提供一种层次化的解决方案：即BCI负责底层Pod容器资源的调度和管理工作，Kubernetes在BCI之上作为PaaS层来管理业务负载，例如管理Deployment、Service、StatefulSet、CronJob等。 基于Kubernetes社区的Virtual Kubelet（简称VK）技术，BCI可以以虚拟节点的形式接入到Kubernetes集群中，使得集群可以轻松获得极大的弹性能力，而不必受限于集群的节点计算容量。BCI在接管Pod容器底层基础设施的管理工作后，Kubernetes不再需要直接负责单个Pod的放置、启动等工作，也不再需要关心底层虚拟机的资源情况，通过BCI即可确保Pod需要的资源随时可用。

目前BCI已经无缝集成到百度智能云容器引擎(Cloud Container Engine，即CCE)中，您可以通过CCE快速体验BCI的容器运行能力。



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

如果您已经建立了CCE集群，可以通过部署虚拟节点（基于VK）的方式来使用BCI。有了虚拟节点后，当您的CCE集群需要扩容时，无需规划节点的计算容量，可以直接在虚拟节点下按需创建BCI，BCI与集群中真实节点上的Pod之间网络互通。建议您将长时间运行的业务负载的弹性流量部分调度至BCI，这可以缩短弹性扩容的时间，减少扩容成本，并充分利用已有资源。当业务流量下降后，您可以快速释放部署在BCI上的Pod，从而降低使用成本。

在CCE集群中，您需要手动部署虚拟节点（即VNode），才能创建BCI Pod。虚拟节点上的Pod均基于BCI运行在安全隔离的容器运行环境中，每个Pod对应一个BCI容器实例。更多信息，请参见CCE产品概述。

管理工具 通过VK将BCI以虚拟节点的方式接入Kubernetes集群后，您可以通过以下方式管理Kubernetes集群及BCI实例的运行情

况：

- 容器实例BCI控制台
  您可以通过弹性容器实例控制台查看BCI实例的运行情况。操作步骤如下：
  a. 登录**容器实例BCI控制台**。
  b. 在顶部菜单栏左上角处选择地域。
  c. 在**容器组**页面，您可以查看该地域下已经创建的BCI实例。

- 容器引擎CCE控制台
  您可以通过容器引擎CCE控制台来操作CCE集群，并查看BCI实例的运行情况。查看BCI实例的操作步骤如下：
  a. 登录**容器引擎CCE控制台**。
  b. 在左侧导航栏单击**集群管理**。
  c. 在集群列表中找到想要查看的集群，单击集群ID进入详情页面。
  d. 在左侧导航栏，选择**工作负载>容器组**。
  e. 在容器组页面，选择命名空间，您可以查看该命名空间下的BCI实例。

- kubectl客户端

您可以通过kubectl客户端在本地计算机来访问远端的Kubernetes集群，使用Kubectl命令来管理集群。具体操作，请参见通过kubectl连接Kubernetes集群

功能限制和说明 由于公有云安全性及虚拟节点带来的限制，BCI目前还不支持Kubernetes中HostPath、DaemonSet等功能，如下表所示。

不支持的功能	说明	推荐替代方案
HostPath	挂载本地宿主机文件到容器中	使用EmptyDir,CFS文件系统,NFS,BOS,暂不支持CDS云盘
HostNetwork	将宿主机端口映射到容器上	BCI将忽略HostNetwork字段，推荐使用type=LoadBalancer的负载均衡
DaemonSet	在容器所在宿主机上部署Static Pod	通过sidecar形式在Pod中部署多个镜像
Privileged权限	容器拥有privileged权限	去除业务逻辑特权依赖
type=NodePort的Service	将宿主机端口映射到容器上	使用type=LoadBalancer的负载均衡

通过CCE使用BCI时，请注意以下事项：

- 请先将容器镜像上传到容器镜像仓库中，便于镜像拉取。推荐您使用百度云容器镜像服务CCR，详细见镜像一节。
- 支持Deployment、ReplicaSet、Job、Cronjob、StatefulSet等常见负载，可以直接运行。
- 支持负载均衡，即配置type=LoadBalancer的Service。

使用流程 推荐使用：CCE托管版 + BCI 混合集群 1. 创建CCE托管集群 推荐创建CCE托管集群，操作说明详见：创建集群
您可以根据自身业务情况按需定义参数，其中以下几点需要特别注意：

- 集群类型请选择 『标准Kubermetes托管集群』
- Master配置请选择 『托管Master』

集群创建成功后，请增加2个BCC worker节点，用于运行集群系统组件，配置在2C8G起即可。

2. 在集群中添加虚拟节点 上述集群创建完毕后，通过 CCE 集群组件管理功能可直接安装虚拟节点组件，完成虚拟节点添加，详情请参考文档管理虚拟节点

3. 准备容器镜像 请先将容器镜像上传到容器镜像仓库中，便于镜像拉取。

- 推荐使用百度云容器镜像服务CCR。可参考文档使用CCR镜像仓库
- 若需使用第三方镜像仓库，可参考文档使用第三方镜像仓库

需要注意的是，您需要保证网络畅通，如果需要拉取公网镜像。则需要为BCI Pod配置公网访问，具体操作请参考文档连接公网

4. 创建BCI Pod 具体创建详细信息和步骤可参考如下文档

- BCI Pod概述
- 指定vCPU和内存创建Pod
- 多可用区创建实例
- 创建GPU实例

调度方式

- **手动调度Pod到BCI**， 详细说明见将Pod调度到VNode
- **使用混合调度插件调度到BCI**， 详细说明和步骤见混合调度

使用BCI功能 在Kubernetes集群中创建Pod到BCI时，为充分使用BCI提供的功能，在不改变Kubernetes语义的前提下，您可以根据需求为Pod添加Annotation。Annotation需添加到Pod级别的metadata中，支持的Annotation列表以及配置示例，请参见Pod Annotation。

将Pod调度到VNode

本文介绍如何将Pod调度到虚拟节点VNode。

概述 对于CCE集群，即混合使用普通节点和虚拟节点（VNode）的模式下，由于虚拟节点上默认配置了污点，用户需要为工作负载增加容忍以及调度策略，，您可以通过在Pod Spec配置nodeSelector和tolerations，将Pod调度到BCI。还可以通过标签将Pod调度到VNode上运行，详细请参考通过标签将Pod调度到VNode上运行。

配置示例 在Pod Spec中添加如下的nodeSelector和tolerations字段，将Pod调度到虚拟节点上。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    type: pod-perf-test
  name: nginx-test
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      type: pod-perf-test
  template:
    metadata:
      labels:
        type: pod-perf-test
    spec:
      containers:
      - image: registry.baidubce.com/qatest/nginx:1.23.0
        imagePullPolicy: IfNotPresent
        name: pod-perf-test
        resources:
          limits:
            cpu: 0.25
            memory: 512Mi
          requests:
            cpu: 0.25
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: Default
      nodeSelector:
        type: virtual-kubelet
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

查看是否调度成功,可查看到Pod所在节点在虚拟节点(bci-virtual-kubelet-0)上。

```
  $ kubectl get pod -o wide
NAME     READY     STATUS     RESTARTS   AGE     IP              NODE   NOMINATED NODE   READINESS    GATES
test-pod 1/1     Running     0          30d     192.168.128.53    bci-virtual-kubelet-0   <none>      <none>
```

混合调度

以多云的方式混合部署的客户，稳态资源一般用自建/公有云K8S集群IDC，弹性资源用BCI容器实例，混合调度插件负责无侵入协调客户负载在K8S集群节点及BCI分布。其主要场景是帮助客户将弹性负载弹性到云上容器实例。

策略	功能描述
bciOnly	工作负载容器强制调度到BCI
localOnly	工作负载强制调度到本地/公有云IDC
localprefer	当工作负载扩容时，当自建/公有云IDC有资源，优先调度到自建/公有云IDC节点；若自建/公有云IDC没资源，可调度到BCI。 当工作负载缩容时，根据配置，优先缩容删除优先级高的BCI或IDC的实例。还可限制本地IDC/BCI最多可部署的实例数。

1. 前提条件 用户在所使用的CCE集群中，已部署cce-virtual-kubelet Helm模板，如何部署详见文档 管理虚拟节点

2. 插件安装 当前只支持在CCE集群以命令行方式安装混合调度插件

```
###### 1. 添加 bci helm repo
helm repo add bci https://registry.baidubce.com/chartrepo/bci-online-public
###### 2. 安装 bci schedule profile 混合调度插件
helm install schedule-profile --version 0.2 bci/schedule-profile
```

3. 通过ScheduleProfile控制Pod在CCE集群节点和BCI之间的分布 用户可以通过创建ScheduleProfile资源，控制pod在本地IDC和BCI之间的分布情况，ScheduleProfile 具体定义如下

```
apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: test-schedule-profile
  namespace: default
spec:
  objectSelector:
    namespace:
      labels:
        type: hybrid
    pod:
      labels:
        type: hybrid
  strategy: localprefer
  distribution:
    local:
      maxNum: 20
      deletionPriority: 10
    bci:
      deletionPriority: 20
```

参数说明：

- objectSelector：控制调度策略覆盖的对象范围。当设置 namespace labels 时，匹配 labels 的所有 namespace 中的 pod，在该调度策略覆盖范围。当设置 pod labels 时，匹配 labels 的所有 pod，在该调度策略覆盖范围。
  - 当 namespace 或者 pod 的 labels 中有多个条目时，对应 namespace 或者 pod 需同时匹配所有 label，才在该调度策略覆盖范围；仅匹配单个 label 不在该调度策略覆盖范围。e.g.

```
objectSelector:
 namespace:
   labels:
     type: hybrid
     exp_enabled: false
```

此时，同时包含 type: hybrid 和 exp_enabled: false 的 namespace 中的 pod 在调度策略覆盖范围

  - namespace 和 pod 的 labels 同时存在时，匹配 namespace labels 的 namespace 中的 pod 及 匹配 pod labels 的所有 pod 都在该调度策略覆盖范围。
  - 当且仅当一个pod只对应一个ScheduleProfile时，这个pod在其调度策略覆盖范围内。若一个 pod 同时匹配多个 ScheduleProfile 或不匹配任何 ScheduleProfile，则这个 pod 不受任何 ScheduleProfile 影响。若需要通过多个 ScheduleProfile 管理不同的工作负载，建议通过配置 pod labels 进行区分，规避工作负载同时匹配多个 ScheduleProfile 的风险。
- strategy：调度策略选择，目前支持 localprefer/bciOnly/localOnly三种。
- distribution：仅localprefer策略可设置该字段。
  - local 描述本地IDC的部署策略，maxNum 限制本地IDC最多可部署的pod数目；deletionPriority 的作用是，当deployment缩容时，部署在本地IDC的pod被删除的优先级，值越大，越容易被删除。
  - bci 描述bci的部署策略，maxNum 限制bci最多可部署的pod数目，任何时刻都不会被打破；deletionPriority 的作用是，当deployment缩容时，部署在bci的pod被删除的优先级，值越大，越容易被删除。
  - bci 和 local 中的 maxNum 不能同时设置，maxNum取值范围 [0~int32]，0为不限制；deletionPriority取值范围[-100, 100]。 **4. 约束和说明**
- 本地自建IDC和BCI的最大部署pod数（maxNum）不能同时设置。
- 可以调整ScheduleProfile的覆盖范围（objectSelector），但是要注意，ScheduleProfile对存量已调度pod无影响，比如调大ScheduleProfile的覆盖范围后，新纳入覆盖范围的pod的删除优先级不会发生变化。
- deletionPriority 仅对 1.22 版本以上的 k8s 集群有效。若集群版本低于 1.22，工作负载缩容时，随机选择实例删除。
- 在deployment滚动升级场景下，推荐配置尽可能小的maxSurge值（如直接配置为0），避免出现升级时限制maxNum的区域实际部署数少于maxNum的现象。
- Pod只能关联一个ScheduleProfile，如果一个Pod的labels或所属namespace的labels存在于多个ScheduleProfile中，系统不会对该Pod做任何操作，效果类似于Pod没有关联的ScheduleProfile。
- ScheduleProfile的增删，可能导致Pod所属的ScheduleProfile变化，进而打破MaxNum等策略的限制。
- 若用户负载中明确指定了 nodeName，其关联的 pod 不受 ScheduleProfile 限制，可能打破 MaxNum 等策略的限制。
- 为了避免混合调度影响系统组件，对于需要开启混合调度的 namespace，必须打上开启混合调度的标签“hybrid-schedule = enabled”

5. 使用样例 使用profile配置管理集群内pod，通过labelSelector类方式关联profile和pod，并配置关联pod的分配策略，实现pod在自建/公有云K8S集群本地IDC和云上BCI的分配或数量限制。

本地突发负载上云场景

当工作负载扩容，本地资源不足或者达到设置的最大值时，将实例溢出到云端BCI容器实例，限制本地最多创建30个实例。工作负载缩容时，优先释放云端BCI容器实例。

为避免影响k8s系统组件，对需要开启混合调度功能的 namespace 打上 label，以 default namespace 为例

注：此处仅仅只是在某个名字空间开启混合调度功能，如果想让整个名字空间都被某个调度策略覆盖（受某个调度策略管理），仍然需要在 ScheduleProfile 的 objectSelector 中配置。

(1) 对需要开启混合调度功能的 namespace default打上开启混合调度功能标签

```
kubectl label namespace default hybrid-schedule=enabled
```

(2) 配置 ScheduleProfile

```
  apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: burst-to-cloud
  namespace: default
spec:
  objectSelector:
    pod:
      labels:
        type: hybrid
  strategy: localprefer
  distribution:
    local:
      maxNum: 30
      deletionPriority: 10
    bci:
      deletionPriority: 20
```

(3) 提交工作负载，以下为一个工作负载样例

```
  kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        type: hybrid
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          imagePullPolicy: IfNotPresent
      imagePullSecrets:
        - name: default-secret
```

本地强制负载弹性到容器实例场景

将工作负载强制调度到BCI容器实例时，可使用bciOnly策略。

(1) 对需要开启混合调度功能的 namespace 打上开启混合调度功能标签“hybrid-schedule=enabled”

```
  kubectl label namespace default hybrid-schedule=enabled
```

(2) 配置 ScheduleProfile

```
  apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: burst-to-cloud
  namespace: default
spec:
  objectSelector:
    pod:
      labels:
        type: hybrid
  strategy: bciOnly
```

6. 插件卸载

```
  helm uninstall schedule-profile
```

通过标签将Pod调度到VNode上运行

本文介绍如何通过指定 Pod 标签或命名空间标签将 Pod 调度到虚拟节点 VNode 上运行。

前提条件

- 已创建 CCE托管集群或 CCE独立集群，具体操作请参考 创建集群
- 在集群中安装 cce-virtual-kubelet 组件，具体操作请参考 部署cce-virtual-kubelet组件
- 可以通过 kubelet 连接 kubernetes 集群，具体操作请参考 通过kubelet连接集群

将 Pod 调度到 VNode 上运行

通过 Pod 标签将 Pod 调度到 VNode 上

创建 Pod 时添加标签 **baidubce.com/bci=true** ，将 Pod 调度到 VNode 上运行。示例如下：

```
  metadata:
  labels:
    baidubce.com/bci: "true"
```

查看是否调度成功，可查看到 Pod 被调度到虚拟节点 (bci-virtual-kubelet-0) 上。

```
  kubectl get pod -o wide
NAME       READY   STATUS     RESTARTS   AGE    IP              NODE                  NOMINATED NODE     READINESS
GATES
test-pod   1/1     Running    0          103s   172.254.129.54     bci-virtual-kubelet-0      <none>           <none>
```

通过命名空间标签将 Pod 调度到 VNode 上

在命名空间上添加标签 **baidubce.com/bci=true** 后，在该命名空间内创建 Pod，Pod 将被调度到 VNode 上。示例如下：

创建命名空间

```
  kubectl create ns vk
```

为 Pod 所在命名空间 **vk** 添加标签 **baidubce.com/bci=true**

```
  kubeclt label namespace vk baidubce.com/bci=true
```

创建 Pod 指定命名空间为 **vk**

```
metadata:
namespace: vk
```

查看是否调度成功，可查看到 Pod 被调度到虚拟节点 (bci-virtual-kubelet-0) 上。

```
kubectl -n vk get pod -o wide
NAME       READY      STATUS      RESTARTS    AGE        IP              NODE                    NOMINATED NODE
READINESS   GATES
test-pod    1/1        Running     0           13s        172.254.129.52     bci-virtual-kubelet-0      <none>           <none>
```

BCI自定义Condition

Pod status condition可以提供Pod状态和健康状况的信息。本文汇总了BCI Pod 的condition并给出说明。

reason	message	说明
Creating	Creating	创建请求已经提交，BCI pod正在创建中
AutoInstanceTypeMatch	The best matched instanceType for current instance is xc xGi	对提交的BCI Pod的资源规格进行规整。
NO_STOCK	Create BCI failed because the specified instance is out of stock	当前可用区的BCI资源库存不足。您可以使用多可用区的方式创建BCI Pod来提高创建成功率。

BCI Pod

BCI Pod概述

BCI能为Kubernetes提供基础的容器Pod运行环境，每个BCI实例相当于一个Pod。本文介绍BCI Pod的配置、创建方式和生命周期。

基本配置 基于Kubernetes社区的Virtual Kubelet技术，BCI支持以虚拟节点（VNode）的形式接入到Kubernetes集群中。一个BCI容器实例相当于一个Pod，包含以下几部分配置：

1. 规格： 规格包括vCPU、内存等配置，定义了BCI Pod的计算性能等。创建BCI Pod时，您可以指定BCI规格（直接指定vCPU和内存）来满足资源、增强网络能力等特殊需求。BCI当前支持的规格详见规格说明
2. 容器镜像：部署容器应用时，需要准备好容器镜像。容器镜像包含容器应用运行所需的程序、库文件、配置等。
   - 推荐使用百度云容器镜像服务CCR托管容器镜像，具体操作可参考文档使用CCR镜像仓库；您也可以使用第三方镜像仓库，具体操作可参考文档使用第三方镜像仓库；
   - 拉取镜像时，需要保证网络畅通，如镜像为公网镜像则需配置BCI容器能够访问公网，具体操作请参考文档连接公网
   - 若您对容器启动耗时有要求，推荐您使用镜像缓存功能来节约实例的启动耗时，具体操作可参考文档镜像缓存
3. 网络：一个BCI Pod将占用所属VPC下的交换机的一个弹性网卡资源，默认具备一个内网IP地址。如果需要连接公网，例如需要拉取公网镜像。则需要为BCI Pod绑定EIP，或者为所属VPC绑定NAT网关，更多信息详见如下文档
   - 配置公网访问，参见连接公网
   - 配置Pod访问集群内ClusterIP Service，参看配置BCI Pod访问集群内Service
   - 配置BCI Service集群外访问，参看通过Service访问BCI服务
   - 配置网络安全组，参看配置网络安全组
   - 配置自定义DNSconfig, 参看支持自定义DNSConfig

4. 存储：一个BCI Pod默认有20 GiB的临时存储空间，您可以根据需要增加临时存储空间。如果想要保留存储的文件，建议使用外挂数据卷作持久化存储。BCI支持挂载如下数据卷
    - EmptyDir， 详细说明见文档挂载EmptyDir数据卷
    - ConfigMap， 详细说明见文档挂载ConfigMap数据卷
    - CFS， 详细说明见文档挂载CFS文件存储
    - PFS， 详细说明见文档挂载PFS并行文件存储
    - BOS， 详细说明见文档挂载BOS数据卷

创建方式 创建方式概述 根据业务场景和使用场景，BCI Pod支持指定vCPU和内存的定义方式，对应计费模式如下：

创建方式	计费说明	相关文档
指定vCPU和内存	根据您创建时指定的vCPU和内存进行计费。对于不满足要求的vCPU和内存规格，系统将自动进行规整，并按自动规整后的规格进行计费。	指定vCPU和内存创建Pod
多可用区创建实例	应对可用区库存不足情况，您可以使用多可用区创建实例	多可用区创建实例)
创建GPU实例	BCI也支持创建GPU实例	创建GPU实例

> **说明** 更多关于BCI Pod计费的信息，请参见BCI实例计费。

调度方式

- **手动调度Pod到BCI**， 详细说明见将Pod调度到VNode。
- **使用混合调度插件调度到BCI**， 详细说明和步骤见混合调度。

优化使用成本 根据您的业务特征，在按量付费使用BCI的基础上，您还可以使用预留实例券来降低资源使用成本。

- 对于长时间运行的稳定业务负载，推荐使用预留实例券来抵扣BCI实例账单。具体操作，请参见使用预留实例券。

应对库存不足 BCI提供容器的云上运行资源，在大规模创建BCI Pod的场景下，您所在地域和可用区可能会存在指定资源售罄的情况，建议您使用多可用区的方式创建资源，保证创建成功率。具体操作，请参见：

- 多可用区创建实例

日志 BCI支持将日志持久化存储到百度日志服务BLS中，详细操作步骤见BLS日志采集方式

监控 BCI支持常用实例指标，您可以通过如下方式查看指标

- 从BCM（百度云监控）控制台查看，具体方式见查看实例监控指标。
- 通过VK查看获取使用指标，具体说明见通过VK获取实例监控指标。您还可根据文档配置指标收集。

运维 BCI提供运维能力，您可以通过使用coredump分析实例程序异常配置将coredump文件存入CFS中或者对象存储BOS中，用于分析程序异常。

BCI Pod生命周期

本文介绍BCI实例的生命周期状态，您可以根据实例状态，设计和实现符合您业务逻辑的处理逻辑。

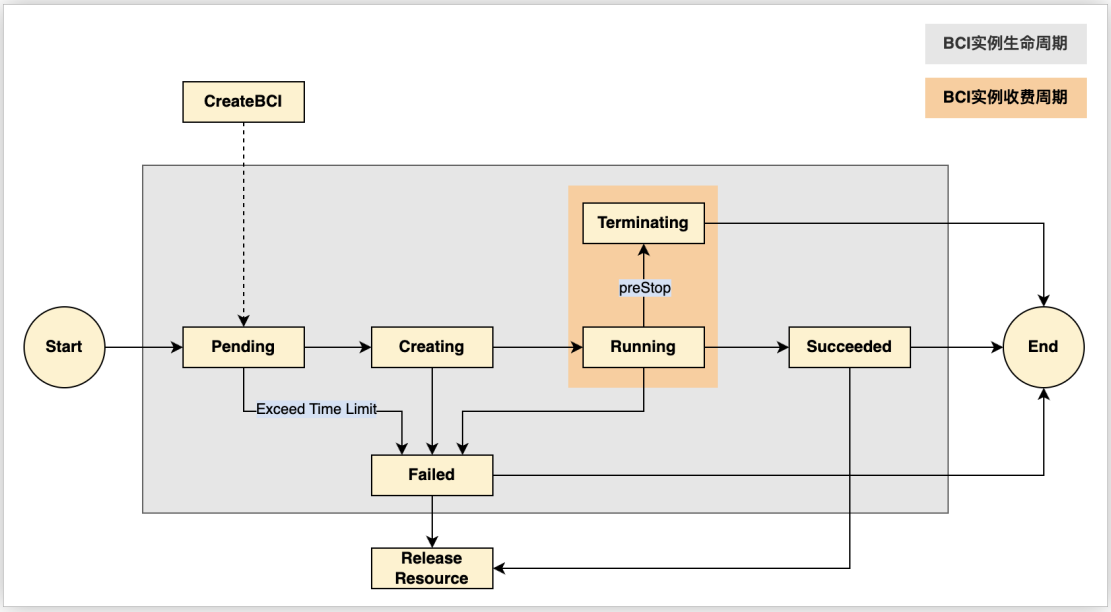
BCI实例状态

在实例的生命周期中，不同的阶段有其固有的状态，具体如下表所示：

BCI Pod状态	说明	对应Kubernetes Pod状态	是否收费
等待创建（Pending）	BCI Pod等待创建。	Pending	否
启动中（Creating）	BCI Pod中有一个或多个容器还在启动中，并且没有处于运行中的容器。	Pending	否
运行中（Running）	BCI Pod中所有容器均已经创建成功，并且至少有一个容器正在运行中。	Running	是
终止中（Terminating）	BCI Pod正在终止。对于运行中的实例，如果配置了preStop，则在删除实例时，Pod将进入Terminating状态。执行完preStop后，BCI Pod将自动删除。	Running	是
运行成功（Succeeded）	BCI Pod中所有容器均已运行成功终止，并且不会再重启。	Succeeded	否
创建/运行失败（Failed）	BCI Pod创建失败。 BCI Pod中所有容器均已运行终止，并且至少有一个容器是运行失败终止，即容器以非0状态退出或者被系统终止。	Failed	否

> **重要** BCI实例的重启策略仅决定实例内容器的行为，BCI实例不会被自动重启。

BCI实例的生命周期状态转换如下图所示：



> **说明**
>
> - 当BCI实例运行终止后，底层计算资源将会被回收，随实例一起创建的其它资源（例如EIP等）默认随实例一起释放。

容器状态

状态	说明
启动中（Waiting）	容器正在等待创建，还未开始运行。 一般在InitContainer运行时，应用容器会处于Waiting状态，直到InitContainer退出。
运行中（Running）	容器已经成功创建，并且正在运行。
运行终止（Terminated）	容器运行终止并退出，包括运行成功终止和运行失败终止。

指定vCPU和内存创建Pod

如果没有特殊的规格需求，推荐您指定vCPU和内存来创建BCI Pod（即BCI实例），系统会尝试使用多种BCC规格进行容器创建，以提供比BCC单规格更好的弹性和资源供应能力。

规格说明 创建BCI实例时，如果指定的vCPU和内存不符合要求，系统将自动按照BCI支持的规格进行规整。规整时将向最接近的BCI规格进行规整，同时需满足指定的vCPU和内存≤BCI规格的vCPU和内存。例如：创建BCI实例时，声明了7 vCPU，13 GiB内存，则实际创建的BCI实例为8 vCPU，16 GiB内存。

BCI支持的规格如下表所示

CPU/核	内存区间（GiB）
0.25	0.5、1、2
0.5	1、2、3、4
1	1、2、4、8
2	4、8、16
4	8、16、32
8	16、32
12	24、48
16	32、64
32	64、128

说明：目前BCI实例仅支持挂载一块弹性网卡，暂不支持多网卡能力。

配置说明 指定vCPU和内存创建BCI Pod时，支持以下方式：

- 指定Pod内容器的vCPU和内存：通过定义Containers的limits或requests来指定，建议使用limits。

如果您没有指定，或者同时指定了limits、requests，实际生效情况如下：

场景	Pod规格
全部未指定	按默认规格（1vCPU，2 GiB内存）
仅指定limits	汇总limits
仅指定requests	汇总requests
同时指定limits和requests	分别汇总limits和requests，按两者中较大值

说明：多个容器汇总limits、requests后的规格，系统会自动进行资源规整，并按规整后的规格进行计费，可通过kubectl describe po pxxx,查看事件

指定Pod内容器的vCPU和内存为Kubernetes默认方式。每个BCI Pod最多可以支持20个容器，每个容器的vCPU和内存规格可以自定义配置。

配置示例如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: testpod
  labels:
    app: testpod
spec:
  replicas: 10
  selector:
    matchLabels:
      app: testpod
  template:
    metadata:
      labels:
        app: testpod
    spec:
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Equal"
        value: "baidu"
        effect: "NoSchedule"
      containers:
      - name: gohttp
        image: registry.baidubce.com/qa-test/gohttp:v0.0.1
        imagePullPolicy: Always
        command:
          - sleep
          - "36000"
        resources:
          limits:
            cpu: 0.25
            memory: 1Gi
          requests:
            cpu: 0.25
            memory: 1Gi
```

创建GPU实例

本文介绍如何创建并使用BCI GPU实例。 **BCI GPU规格说明** BCI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。

BCI 规格名称	CPU	内存	GPU类型	显存	GPU卡数
bci.gna2.c8m36.1a10	8	36	Nvidia A10 PCIE	24*1	1
bci.gna2.c18m74.1a10	18	74	Nvidia A10 PCIE	24*1	1
bci.gna2.c30m118.2a10	30	118	Nvidia A10 PCIE	24*2	2
bci.gna2.c62m240.4a10	62	240	Nvidia A10 PCIE	24*4	4

创建实例 配置说明如下

- 指定GPU型号，需要在 Annotation 指定 GPU型号，请注意， Annotation需要配置在Pod Spec中，而不是Deployment Spec中。

```
  annotations:
  bci.virtual-kubelet.io/bci-gpu-type: "Nvidia A10 PCIE"
```

- 指定资源配置：GPU卡数、CPU和内存数量

```
  resources:
  limits:
    nvidia.com/gpu: 1 # GPU卡数
    cpu: 8 # CPU核数
    memory: 36Gi # MEM数量
  requests:
    nvidia.com/gpu: 1 # GPU卡数
    cpu: 8 # CPU核数
    memory: 36Gi # MEM数量
```

完整业务YAML示例

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: spot-deployment-test-gpu-wzy
  labels:
    run: ooo
spec:
  replicas: 1
  selector:
    matchLabels:
      run: ooo
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: ooo
      annotations:
        bci.virtual-kubelet.io/bci-gpu-type: "Nvidia A10 PCIE"
        bci.virtual-kubelet.io/bci-logical-zone: "zoneF" # 填写对应资源的可用区
        bci.virtual-kubelet.io/bci-subnet-id: "xxxxxx" # 子网需要和可用区对应
      name: spot-deployment-test-wzy-bid
    spec:
      volumes:
      - name: podinfo
        downwardAPI:
          defaultMode: 420
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.labels['mylabel']
            path: mylabel
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.annotations['myannotation']
            path: myannotation
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.labels
            path: labels
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.annotations
            path: annotations
```

```
    - path: workload_cpu_limit
      resourceFieldRef:
        containerName: ooo1
        divisor: 1m
        resource: limits.cpu
    - path: workload_cpu_request
      resourceFieldRef:
        containerName: ooo1
        divisor: 1m
        resource: requests.cpu
    - path: workload_mem_limit
      resourceFieldRef:
        containerName: ooo1
        divisor: 1Mi
        resource: limits.memory
    - path: workload_mem_request
      resourceFieldRef:
        containerName: ooo1
        divisor: 1Mi
        resource: requests.memory
nodeSelector:
  type: "virtual-kubelet"
tolerations:
- key: "virtual-kubelet.io/provider"
  operator: "Equal"
  value: "baidu"
  effect: "NoSchedule"
containers:
- image: hub.baidubce.com/cce/nginx-alpine-go
  name: ooo1
  env:
  - name: "MY_CPU_LIMIT"
    valueFrom:
      resourceFieldRef:
        containerName: ooo1
        resource: limits.cpu
  - name: "MY_CPU_REQ"
    valueFrom:
      resourceFieldRef:
        containerName: ooo1
        resource: requests.cpu
  - name: "MY_IP"
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: status.podIP
  volumeMounts:
  - name: podinfo
    mountPath: /etc/podinfo
  resources:
    limits:
      nvidia.com/gpu: 1 # GPU卡数
      cpu: 8 # CPU核数
      memory: 36Gi # MEM数量
    requests:
      nvidia.com/gpu: 1 # GPU卡数
      cpu: 8 # CPU核数
      memory: 36Gi # MEM数量
```

多可用区创建实例

当您在应对突发流量，进行业务的快速水平扩容时，或者启动大量实例进行Job任务处理时，可能会遇到可用区对应规格实例库存不足等特殊情况，从而导致实例创建失败，影响业务。此时，您可以采用指定多可用区的方式来创建实例，提高实例创建的成功率。 **前提条件** 已在要使用的专有网络VPC下创建多个不同可用区的子网。

- 关于如何创建子网，请参见私有网络 VPC - 子网。

背景信息 创建BCI实例时，可以通过指定多个子网来指定多个可用区，系统会随机把请求分散到所有指定的可用区中，来分散压力，如果在某一个可用区遇到没有库存的情况，会自动切换到下一个可用区继续尝试创建。
指定多可用区时，需注意以下限制：

- 指定的子网必须属于同一个VPC。
- 最多可以指定10个子网。
  配置说明 开启使用多可用区资源的步骤

1. 对于 job、deployment 等工作负载，只需要在 spec.template.metadata.annotations 中新加一个 annotation 即可。
2. annotation 的 key 为 "bci.virtual-kubelet.io/bci-subnet-ids"，value是多个子网以英文逗号连接的字符串，如需使用多个可用区，则需填对应多个可用区的子网。
   配置示例 例如想使用可用区 D & F 的资源，在可用区D、F分别创建了sbn-xxx1，sbn-xxx2的子网 deployment 或 job yaml 中最终效果样例:

```
spec:
  template:
    metadata:
      annotations:
        "bci.virtual-kubelet.io/bci-subnet-ids":"sbn-xxx1,sbn-xxx2"
```

BCI Pod Annotation

在Kubernetes集群中创建BCI类型的Pod（即BCI实例）时，为充分使用BCI提供的功能，在不改变Kubernetes语义的前提下，您可以根据需求为Pod添加Annotation。 本文为您介绍创建BCI Pod时支持添加的Annotation，以及BCI Pod调度完成后会追加的Annotation。

BCI Pod支持的Annotation 创建BCI Pod时，通过在Pod metadata中指定Annotation，可以为对应Pod开启特定的特性，或者覆盖虚拟节点上的原有配置，支持添加的Annotation如下：

> **注意：**
>
> - 列举的Annotation仅适用于创建到虚拟节点上的Pod，即BCI实例，调度到普通节点上的Pod不受这些Annotation影响。
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。
> - 配置网络、安全组和可用区相关Annotation时，需要保证相互之间的一致性：
> - 如子网与安全组应该同属一个VPC，子网与可用区参数需匹配等。
> - 如未在Pod中单独配置，则默认使用虚拟节点本身的配置。
> - 如通过Annotation指定子网参数，则必须同时通过Annotation指定可用区参数，可用区参数的值为子网所属可用区。

实例 | 功能及相关文档 | 参数 | 示例值 | 描述 | | --- | --- | --- | --- | | 配置BCI Pod所属可用区 | bci.virtual-kubelet.io/bci-logical-zone | "zoneF" | BCI实例所在可用区 | | 配置BCI Pod所属子网 | bci.virtual-kubelet.io/bci-subnet-id | "sbn-xxx" | BCI实例所属子网（子网短ID） | | 多可用区创建实例 | bci.virtual-kubelet.io/bci-subnet-ids | "sbn-xxx1,sbn-xxx2" | 支持指定多个可用区实现多

可用区功能 | | 自定义设置BCI Pod的hosts | bci.virtual-kubelet.io/pod-host-aliases | "[{"ip":"10.10.xx.xx","hostnames":["example.com"]},{"ip":"10.10.yy.yy","hostnames":["foo.com","bar.com"]}]" | 自定义设置BCI Pod的hosts（即/etc/hosts）。 | | 为BCI Pod绑定自定义标签 | bci.virtual-kubelet.io/resource-tag | "key1:value1,key2:value2" | 绑定的标签（Tag）字符串，标签键和标签值之间用半角冒号隔开，多个标签之间用半角逗号隔开。 | | 为BCI Pod配置延迟销毁时长(Job类型Pod) | bci.virtual-kubelet.io/pod-delay-release-duration-minute | "10" | 设置延迟销毁时长，只针对job任务类型Pod(即restartPolicy为never)生效 | | 为BCI Pod配置延时销毁时长(Succeeded状态的Pod) | bci.virtual-kubelet.io/pod-delay-release-succeeded | "true" | 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 | | 创建GPU类型实例 | bci.virtual-kubelet.io/bci-gpu-type | "Nvidia A10 PCIE" | 指定GPU卡型号 | | 设置BCI Pod的故障处理策略 | bci.virtual-kubelet.io/bci-fail-strategy | "fail-back" | BCI Pod的故障处理策略，取值如下：

- fail-back：失败自动恢复。即Pod创建失败后自动尝试重新创建。
- fail-over：失败转移。效果等同于fail-back。
- fail-fast：快速失败。即Pod创建失败后直接报错。| | 自定义设置BCI Pod的最大Pending时长 | bci.virtual-kubelet.io/bci-max-pending-minute | "30" | 自定义设置Pod对应BCI实例的最大Pending时长，超时后系统会自动终止实例。
取值范围为10~1440的整数，单位为分钟。默认为1小时。| | 配置资源规整时忽略特定容器 | bci.virtual-kubelet.io/bci-resource-ignore-containers | "container1,container2" | 指定资源规整时忽略的容器列表 | | 强制终止Sidecar容器并忽略容器退出码 | bci.virtual-kubelet.io/bci-ignore-exit-code-containers | "sidecar1,sidecar2" | 指定需要忽略状态码的Sidecar容器列表 | | 忽略Sidecar容器的NotReady状态 | bci.virtual-kubelet.io/bci-ignore-not-ready-containers | "sidecar1,sidecar2" | 指定需要忽略Not Ready的Sidecar容器列表 |

网络

功能及相关文档	参数	示例值	描述
配置BCI Pod所属安全组	bci.virtual-kubelet.io/bci-security-group-id	"g-xxx"	指定实例的安全组ID
配置BCI Pod是否自动创建EIP	bci.virtual-kubelet.io/bci-create-eip	"true"	是否自动创建并绑定EIP • true：自动创建EIP • false: 不自动创建EIP
配置BCI Pod EIP线路类型	bci.virtual-kubelet.io/bci-create-eip-route-type	"BGP"	设置EIP的线路类型。不指定时，默认为BGP类型。可选值： • BGP：标准型BGP • BGP_S：增强型BGP • ChinaMobile：中国移动 • ChinaUnicom：中国联通 • ChinaTelcom：中国电信
配置BCI Pod EIP带宽	bci.virtual-kubelet.io/bci-create-eip-bandwidth	"10"	设置EIP带宽。带宽能力和EIP线路类型相关。带宽区间： • BGP：1～200 Mbps • BGP_S：100～5000 Mbps • ChinaMobile：1～5000 Mbps • ChinaUnicom：1～5000 Mbps • ChinaTelcom：1～5000 Mbps
配置BCI Pod EIP计费方式	bci.virtual-kubelet.io/bci-create-eip-paymethod	"ByBandwidth"	设置EIP计费方式 • ByTraffic：流量 • ByBandwidth：带宽
为BCI Pod绑定已有的EIP	bci.virtual-kubelet.io/bci-eip-ip	"10.10.xx.xx"	绑定已有的EIP。且此EIP状态必须可用(Available)。限定条件：一个EIP只能成功绑定到一个BCI实例上，若使用deployment方式，副本数replicas需设置为1，否则可能导致创建失败。
配置BCI Pod访问集群内Service	bci.virtual-kubelet.io/kube-proxy-enabled	"true"	BCI实例访问K8S ClusterIP类型Service

容器配置 | 功能及相关文档 | 参数 | 示例值 | 描述 | | --- | --- | --- | --- | | 配置NTP服务 | bci.virtual-kubelet.io/bci-ntp-server | "10.0.xx.xx" | 指定NTP服务器的地址 |

运维

功能及相关文档	参数	示例值	描述
查看Core dump文件	bci.virtual-kubelet.io/core-pattern	"/tmp/cores"	Core dump文件保存目录

BCI Pod追加的Annotation BCI Pod调度完成后会追加的Annotation如下表所示。您可以通过kubectl describe命令进行查询。

参数	示例值	描述
bci.virtual-kubelet.io/bci-subnet-id	sbn-xxx	BCI实例所属子网(短ID)
bci.virtual-kubelet.io/order-id	xxx	BCI实例对应的订单ID
bci.virtual-kubelet.io/pod-id	p-xxx	BCI实例短ID
bci.virtual-kubelet.io/bci-bound-eip-bandwidth	10	绑定的EIP实例带宽
bci.virtual-kubelet.io/bci-bound-eip-id	ip-xxx	绑定的EIP实例ID（短ID）
bci.virtual-kubelet.io/bci-bound-eip-ip	10.10.xx.xx	绑定的EIP实例IP地址
bci.virtual-kubelet.io/bci-bound-eip-paymethod	ByTraffic	绑定的EIP实例计费方式
bci.virtual-kubelet.io/bci-bound-eip-route-type	BGP	绑定的EIP实例线路类型

设置BCI Pod的故障处理策略

默认情况下，BCI Pod创建失败后，系统会自动重试尝试创建。如果您希望尽快得到创建结果以便及时处理故障，可以修改BCI Pod的故障处理策略。

配置说明 在虚拟节点上创建BCI Pod时，可能会因为库存不足等原因导致Pod创建失败，默认情况下，系统会自动进行重调度，尝试重新创建Pod。您可以通过添加bci.virtual-kubelet.io/bci-fail-strategy的Annotation来修改BCI Pod的故障处理策略，设置BCI Pod创建失败后是否尝试重新创建。

> **重要**
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

bci.virtual-kubelet.io/bci-fail-strategy的取值说明如下：

取值	说明	场景
fail-back	失败自动恢复。即Pod创建失败后自动尝试重新创建。此时，Pod会保持Pending状态，直到创建成功变为Running状态。	侧重成功率，能够接受Pod延迟交付。
fail-over	失败转移。效果等同于fail-back。	侧重成功率，能够接受Pod延迟交付。
fail-fast	快速失败。Pod创建失败后直接报错。Pod显示为Pending状态，由上层编排决定是否重试，或者把Pod创建调度到普通节点。	侧重效率，希望Pod快速交付，有完善的失败处理逻辑。

配置示例

```yaml
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
      annotations:
        bci.virtual-kubelet.io/bci-fail-strategy: "fail-fast"  #设置Pod故障处理策略，不再重新创建
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
      restartPolicy: Always
```

以上YAML示例中，BCI Pod的故障处理策略为fail-fast。如果Pod长时间Pending，您可以查看Pod status.reason。

- 如果Pod status.reason为ContainerInstanceScheduleFailed，则表示BCI调度失败。此时查看Pod status condition，通过ContainerInstanceCreated的reason和message可以确定具体原因，进而采取相应措施，例如修改指定的规格，设置多可用区等。
- 如果Pod status.reason为空（fail-fast一般不会出现该情况），可以查看Pod status condition，通过ContainerInstanceCreated的status确认调度状态。
  - 如果ContainerInstanceCreated为False，且reason不是Creating，则表示BCI调度还未成功，需要继续等待。 以库存不足创建BCI Pod失败为例，当Pod的故障处理策略为fail-fast时，Pod status condition为ContainerInstanceCreated的示例如下：

> **说明**
>
> 如果Pod的故障处理策略为fail-back，Pod创建失败后系统会自动尝试重调度。此时，Pod status.reason不会显示ContainerInstanceScheduleFailed，您也可以查看Pod status condition，通过ContainerInstanceCreated的reason和message确定当前调度周期内调度失败的原因。

```
{
  "conditions": [
    {
        "lastProbeTime": "2025-03-04T15:13:26Z",
        "lastTransitionTime": "2025-03-04T15:13:26Z",
        "message": "Create BCI failed because the specified instance is out of stock",
        "reason": "NoStock",
        "status": "False",
        "type": "ContainerInstanceCreated"
    }
  ],
  "message": "Create BCI failed because the specified instance is out of stock",
  "reason":"ContainerInstanceScheduleFailed",
  "phase": "Pending"
}
```

自定义设置BCI Pod的最大Pending时长

Pending状态的BCI实例（BCI Pod）是不计费的，对于长时间处于Pending状态的异常Pod，如果您没有及时处理，可能会对业务产生异常影响。默认情况下，BCI实例的最大Pending时长为1小时，您可以根据实际业务情况自定义设置最大Pending时长，系统会自动终止超时的BCI实例，可以在一定程度上规避因没有及时处理异常Pod而造成业务异常影响。

功能说明 每个BCI实例相当于一个Pod。创建BCI Pod时，当对应的BCI实例进入等待创建（Pending）阶段，如果出现镜像拉取失败、Volume挂载失败等问题，BCI实例会一直处于Pending状态，但不会计费，您需要及时处理这类异常的CI实例来避免对业务产生异常影响。默认情况下，对于Pending时长超出1小时的BCI实例，系统会自动终止。如果您对于时长有要求，可以自定义设置最大Pending时长。

配置说明 您可以在Pod metadata中添加 bci.virtual-kubelet.io/bci-max-pending-minute 的Annotation来自定义设置Pod对应BCI实例的最大Pending时长。该Annotation的相关说明如下：

- 该Annotation的取值范围为10~1440的整数，单位为分钟，即下限为10分钟，上限为1天。

> **重要**：请根据实际业务情况合理设置最大Pending时长。如果镜像过大且没有镜像缓存的情况下，创建Pod时需要一定的时长来拉取镜像，此时如果最大Pending时长设置过短，可能会导致Pod创建失败。

- 未添加该Annotation的情况下，默认的最大Pending时长为1小时。

配置示例 创建一个设置了最大Pending时长的BCI 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
      annotations:
        bci.virtual-kubelet.io/bci-max-pending-minute: "30"   # 设置最大Pending时长为30分钟
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

自定义设置BCI Pod的hosts

某些场景下，您可能需要自定义设置BCI Pod的hosts，例如拉取自建镜像仓库的镜像时，需要通过hosts明确镜像仓库的实际IP地址。本文介绍如何自定义设置BCI Pod级别的hosts（即/etc/hosts）。

配置说明 您可以通过bci.virtual-kubelet.io/pod-host-aliases 的Annotation自定义设置BCI Pod的hosts，支持传入多组IP和域名的映射关系，格式为：

```
  [{\"ip\":\"10.10.xx.xx\",\"hostnames\":[\"example.com\"]},{\"ip\":\"10.10.yy.yy\",\"hostnames\":
[\"foo.com\",\"bar.com\"]}]
```

> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例 例如拉取自建镜像仓库的镜像时，需要通过hosts明确镜像仓库的实际IP地址时，可参考以下YAML示例：

```
  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
      annotations:
        bci.virtual-kubelet.io/pod-host-aliases: "[{\"ip\":\"10.10.xx.xx\",\"hostnames\":[\"example.com\"]}]"
    spec:
      containers:
      - name: nginx
        image: example.com/test/nginx:1.7.9
        ports:
        - containerPort: 80
```

为BCI Pod绑定自定义标签

标签由一组键值对组成，可以用于标记BCI实例。您可以使用标签来分组管理BCI实例，便于筛选和批量操作。
本文介绍如何为BCI Pod绑定自定义标签（Tag），以便后续可以基于标签管理BCI Pod，例如基于标签进行费用分析。

背景信息 标签（Tag）是一组键值对，百度智能云提供标签管理功能，支持按照各种标准（如用途、所有者或项目）提供跨服务、跨区域对资源进行添加标签，从而快速分类和管理资源。

配置说明 您可以通过 bci.virtual-kubelet.io/resource-tag 的Annotation为BCI Pod绑定自定义标签，最多可以绑定10个标签。标签键和标签值之间用半角冒号隔开，多个标签之间用半角逗号隔开。

重要

- Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
- 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例如下：

```
  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
      annotations:
        bci.virtual-kubelet.io/resource-tag: "key1:value1,key2:value2"   #绑定标签
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

配置资源规整时忽略特定容器

创建BCI Pod（即BCI实例）时，如果指定的vCPU和内存规格不满足BCI规格要求，系统会在满足资源需求的同时自动向最接近的BCI规格进行资源规整，规整后的规格过大可能会造成一定程度的资源浪费。对于一些不影响业务的容器（例如Sidecar容器），可以为其设置对应的Annotation，实现资源规整时忽略该容器，以避免资源浪费，节约BCI使用成本。

功能说明 指定vCPU和内存创建BCI实例时，您可以自定义指定每个容器的vCPU和内存，但汇总到实例级别时，需满足BCI实例整体的vCPU和内存要求。如果实例级别没有配置vCPU和内存，则会汇总计算所有容器的规格之和，对于总和不满足BCI规格的情况，系统会自动进行资源规整。

配置说明 为BCI Pod设置以下Annotation来标记资源规整时忽略的容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-resource-ignore-containers: "container1,container2"
```

> **重要**
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例 按以下YAML创建Deployment，BCI Pod的规格为1 vCPU、2 GiB内存，即采用nginx容器的limits，忽略sidecar容器的limits。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
      annotations:
        bci.virtual-kubelet.io/bci-resource-ignore-containers: "sidecar"   # 声明资源规整时忽略的容器列表
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
      - name: sidecar
        image: sidecar:tag
        ports:
        - containerPort: 8080
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
      restartPolicy: Always
```

> **说明**
>
> 如果上述Yaml没有设置Annotation bci.virtual-kubelet.io/bci-resource-ignore-containers: "sidecar"，则汇总nginx容器和sidecar容器的资源需求为2 vCPU、4 GiB，创建该Deployment时，资源规整后实际创建的BCI Pod为2 vCPU、4 GiB 规格。

配置bci-profile

为了支持CCE Virtual Kubelet组件（VK）能动态热更新配置，BCI支持了bci-profile功能。

- 更新配置时无需重启VK。
- 对于新创建的BCI Pod，可以即时生效更新后的配置；对于存量BCI Pod，需要滚动发布后才能生效更新后的配置。

注意事项 使用该功能时，请确保集群中的VK（cce-virtual-kubelet组件）为最新版本。关于如何升级组件，请参见CCE Virtual Kubelet组件说明文档。

配置说明 创建Pod时，系统会读取kube-system命名空间下的bci-profile配置文件（名为bci-profile的ConfigMap），按其data配置来创建Pod。您可以通过kubectl get cm -n kube-system bci-profile -o yaml命令查看bci-profile的YAML。bci-profile的YAML模板如下：

```
  apiVersion: v1
kind: ConfigMap
metadata:
  name: bci-profile
  namespace: kube-system
data:
  config: |
    {
      "region": "gz",
      "clusterID": "example-cluster-id",
      "subnets": [
        {
          "subnetID": "subnet-12345",
          "logicalZone": "zone-a"
        },
        {
          "subnetID": "subnet-67890",
          "logicalZone": "zone-b"
        }
      ],
      "securityGroupID": "sg-abcdef",
      "enableReserveFailedPod": true,
      "enableForceInjectDNSConfig": false,
      "enableForceInjectDNSConfig": false,
      "disableAutoMatchImageCache": false,
      "dnsConfig": {
        "nameservers": [
          "169.254.20.10",
          "172.21.0.10"
        ],
        "searches": [
          "default.svc.cluster.local",
          "svc.cluster.local",
          "cluster.local"
        ],
        "options": [
          {
            "name": "ndots",
            "value": "3"
          },
          {
            "name": "attempts",
            "value": "2"
          },
          {
            "name": "timeout",
            "value": "1"
          }
        ]
      }
    }
```

修改bci-profile 方式一： 通过kubectl edit命令，注意配置内容需要满足json语法 kubectl edit configmap bci-profile -n kube-system

方式二：通过容器服务管理控制台

1. 登录容器服务管理控制台。

2. 在集群页面，找到目标集群，单击集群名称。

3. 在集群管理页的左侧导航栏，选择配置管理>配置项。

4. 选择命名空间为kube-system。

5. 找到bci-profile，单击YAML编辑。

更新配置项 data的config字段包含的子网、安全组等固定配置项，以json方式存储，您可以根据需要进行更新，更新后的配置可以即时生效（无需重启VK）。注意，如下配置为均为集群级别的配置项，即在创建BCI Pod时，如果没有额外配置，会采用bci-profile中的配置。 支持更新的配置项如下：

配置项	说明
subnets	BCI Pod运行的默认子网列表，需同时写出子网id和逻辑可用区；默认为VK组件安装时使用的子网参数
securityGroupID	BCI Pod所属安全组，默认为VK组件安装时参数,如 "securityGroupID": "sg-abcdef"
enableReserveFailedPod	是否自动保留失败的BCI Pod,默认为false
enableAutoInjectKubeProxy	是否自动注入kube-proxy到pod中，注入后可在BCI pod中访问k8s Cluster IP,默认为false
disableAutoMatchImageCache	是否禁用自动匹配镜像缓存，如果为true，则会禁用创建并且使用自动匹配镜像缓存，默认为false
enableForceInjectDNSConfig	是否强制注入自定义DNS配置，如果为true，则强制注入自定义DNS配置，默认为false
dnsConfig	BCI Pod配置的自定义DNS 配置，默认为null

子网示例：

```
  "subnets": [
  {
    "subnetID": "subnet-12345",
    "logicalZone": "zone-a"
  },
  {
    "subnetID": "subnet-67890",
    "logicalZone": "zone-b"
  }
 ],
```

dnsConfig配置示例：

```
    "dnsConfig": {
     "nameservers": [
      "169.254.20.10",
      "172.21.0.10"
     ],
     "searches": [
      "default.svc.cluster.local",
      "svc.cluster.local",
      "cluster.local"
     ],
     "options": [
      {
       "name": "ndots",
       "value": "3"
      },
      {
       "name": "attempts",
       "value": "2"
      },
      {
       "name": "timeout",
       "value": "1"
      }
     ]
    }
```

成本优化

预留实例券

预留实例券概述

预留实例券是一种抵扣券，可以抵扣按量付费实例（不含抢占式实例）的账单，也能够预留实例资源。相比包年包月实例，预留实例券与按量付费实例这种组合模式可以兼顾灵活性和成本。 **什么是预留实例券？**

- 当前预留实例券功能处于内测中，如需使用，请提交工单申请。

定义 预留实例券是一种抵扣券,可以抵扣相关配置的后付费BCI实例账单，支持1个月、1年、2年、3年购买周期。 预留实例券相比于后付费BCI实例单价更低且支持资源预留。另外，预留实例券支持多种付费方式（全预付、部分预付、零预付），您可根据财务情况灵活选择。 **功能** 预留实例券分为可用区级预留实例券和地域级预留实例券。

1. 预留实例券生效后将自动匹配可抵扣的后付费BCI实例账单；更多信息请查看预留实例券与实例的匹配。
2. 支持资源预留；

- 可用区级资源预留：
  可为所匹配的按量实例资源进行资源保留。
- 地域级资源预留：
  地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

3. 支持抵扣预测。
   预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。更多信息请查看可抵扣的实例。 **付费方式** 预留实例券支持3种付费方式：
4. 全预付：购买券时支付有效期内对应算力的总额，是3种付费方式中总额最低的；
5. 部分预付：购买券时支付有效期内对应算力的一部分费用，剩下的费用将在有效期内的每小时支付。总支付金额 = 半预付金额 + 每小时分期费用 * 购买周期内小时数；

6. 零预付：购买券时无需支付，费用将在有效期内的每小时支付。总支付金额 = 每小时分期费用 * 购买周期内小时数。

> **注意：**预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。
> 零预付根据使用情况自动开放，如果您需要使用，请提交工单申请。

相较于后付费模式，预留实例券可以有效的降低长时间运行实例的费用成本。

	预留实例券	后付费
定义	一种抵扣券，用来抵扣后付费BCI实例账单。	购买BCI时的一种付费方式（先使用后付费），分钟计费，小时出账。
特点	1. **价格优惠**：总成本比后付费实例低； 2. **资源保障**：支持资源预留，保证您创建成功与预留实例券匹配的实例。 3. **灵活抵扣**：实例券不与某个实例绑定，一张实例券可抵扣同规格的不同后付费BCI账单； 4. **灵活支付**：支持多种付费方式（全预付、部分预付、零预付），避免一次性支付带来的资金链压力。	1. 即用即买，随时释放； 2. 单价较高。

资源抵扣 预留实例券生效后将自动匹配满足条件的实例规格(后付费)进行抵扣。
其中抵扣是指相同实例规格的后付费BCI的账单，而不是后付费BCI的费用。

> **注意：**
>
> - 如果您希望对已有的BCI实例进行成本优化，可以筛选出指定规格创建的BCI实例，并根据规格情况购买对应的预留实例券。
> - 如果您还没有创建BCI实例，请根据业务需求选择合适的规格，然后购买对应的预留实例券，并指定规格创建BCI实例。
> - 如果您已经购买了预留实例券，则在创建BCI实例时，必须要指定预留实例券对应的规格，否则预留实例券无法抵扣。

到期自动购买预留实例券. 预留实例券支持自动续费。如果您是按月购买，则自动续费周期为 1 个月。按年购买，则自动续费周期为 1 年。 ![预留实例.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/BCI/%E9%A2%84%E7%95%99%E5%AE%9E%E4%BE%8B_71bb062.png)

购买预留实例券

本文介绍如何在BCI管理控制台购买预留实例券。

> 说明
> 预留实例券支持CPU和GPU实例规格。

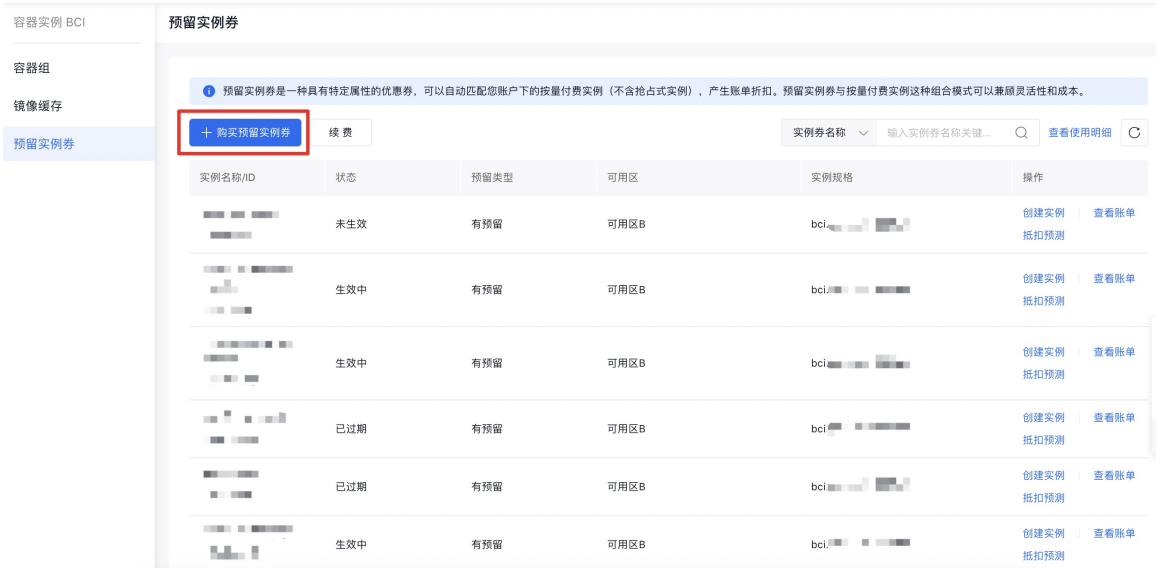
前提条件

- 购买预留实例券前，请确保您要匹配的按量付费实例符合预留实例券使用要求。详情请参见预留实例券概述
- 您无法手动管理预留实例券和按量付费实例的匹配状态，请确保您已了解预留实例券匹配规则。详情请参见预留实例券与实

例的匹配。

操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。
3. 单击**购买预留实例券**。



4. 配置地域信息。
   a. 选择资源预留类型。

> 可用区级资源预留：可为所匹配的按量实例资源进行资源保留。
> 地域级资源预留：地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。
> 详情请查看预留实例券与实例匹配。

   b. 选择地域。
   c. 选择可用区。

5. 配置实例信息。
   a. 选择实例规格。
   b. 选择付费类型。支持**全预付**、**部分预付**和**零预付**。

> 预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。

> 零预付根据使用情况自动开放，如果您需要使用，请提交工单申请。

6. 配置购买参数。
   a. 填写券名：可选。
   b. 填写购买数量。 可以同时匹配同规格按量付费实例的数量。

> 最多购买10台，如需购买更多，请提提交工单申请。

c. 选择预留实例券有效期。支持**1个月、1年、2年和3年**。

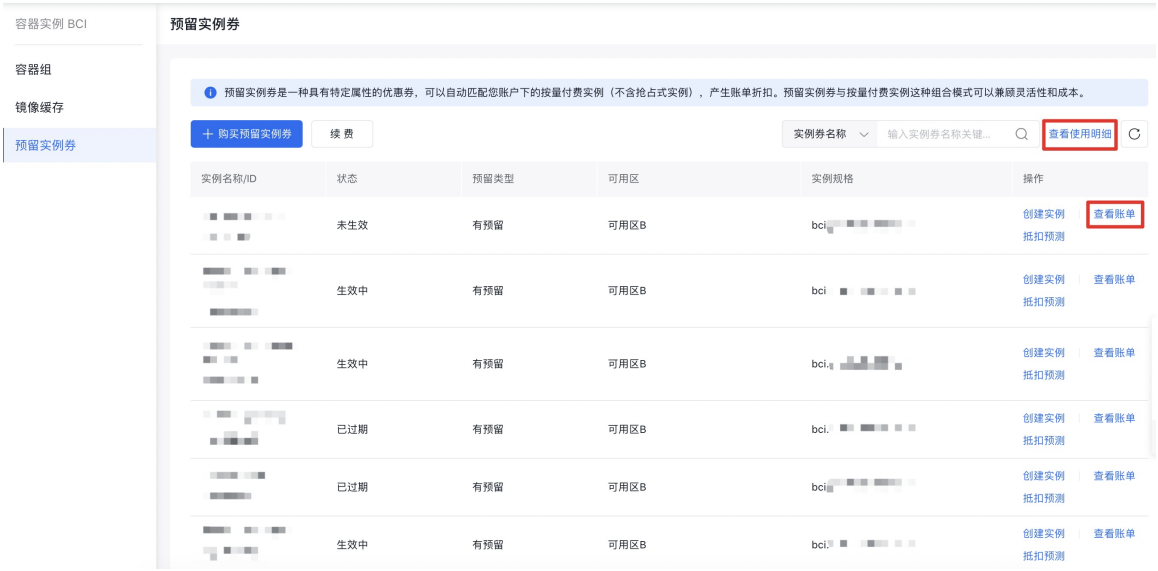
7. 选择是否启动自动续费。

> 按月购买，则自动续费周期为 1 个月。按年购买，则自动续费周期为 1 年。

8. 选择生效时间。目前支持**指定时间生效**。
9. 确认参数，然后单击**提交**。
10. 确认支付信息，然后单击**确认支付**。
    执行结果 您已经成功购买了一张预留实例券，成功匹配按量付费实例后即可抵扣按量付费实例账单。

查看预留实例券使用明细

您可以在预留实例券页面点击**查看使用明细**或**查看账单**，确认券的抵扣情况。



> 重要
> 在配置了多可用区创建BCI实例的场景下，创建的BCI实例可能会发布在多个可用区。如果您发现预留实例券未按预期进行抵扣，请检查预留实例券所在可用区和BCI实例可用区是否一致。

在**抵扣明细**页面，您可以查看预留实例券的抵扣情况。预留实例券的使用明细将记录每个出账周期（每小时）该预留实例券抵扣的实例信息。其中，抵扣时长对应的是计算力*小时（预留实例券按计算力进行抵扣，1个计算力可以理解为1vCPU）。

预留实例券与实例的匹配

购买预留实例券即代表承诺使用一定时长的按量付费实例，预留实例券匹配按量付费实例后才能抵扣账单。如果您的账号下暂时没有可以匹配的按量付费实例，预留实例券会闲置但继续计费。

本章节为您介绍预留实例券匹配按量付费实例的规则和示例。 **匹配规则** 您不能手动匹配预留实例券和按量付费实例。购买预留实例券后，在有效期内预留实例券将自动匹配满足条件的按量付费实例。匹配成功后，预留实例券每小时检查可抵扣的按量付费账单，并按券面的计算力抵扣账单。您可以使用抵扣预测功能查看预留实例券可以匹配的实例，具体操作请参见查看可抵扣的实例。

属性	地域和可用区	实例规格
可用区级预留实例券	只可匹配同一可用区中的按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 支持指定可用区的资源预留，在有效期内预留指定数量指定规格的实例，保证随时可以在指定可用区中成功创建按量付费实例。
地域级预留实例券	支持可用区灵活性，在指定地域中可以跨可用区匹配按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

可用区级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域和可用区相同。
- 实例规格和实例大小相同。

使用可用区级预留实例券的示例如下表所示。

示例场景	按量付费实例	可用区级预留实例券	抵扣效果
匹配成功	持有5台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：5台	预留实例券和5台按量付费实例完全匹配，1张预留实例券每小时抵扣5台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.c8m36.1a10实例，保证随时可以在华北-保定可用区B中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例 1台配置如下： 华北-北京，可用区A bci.c8m36.1a10 1台配置如下： 华北-保定，可用区B bci.c18m74.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：2台	导致匹配失败的原因如下： 1台实例可用区为华北-北京，可用区A； 1台实例规格为bci.c18m74.1a10； 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

地域级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域相同，支持跨可用区匹配实例。
- 实例规格和实例大小相同。

使用地域级预留实例券的示例如下表所示。

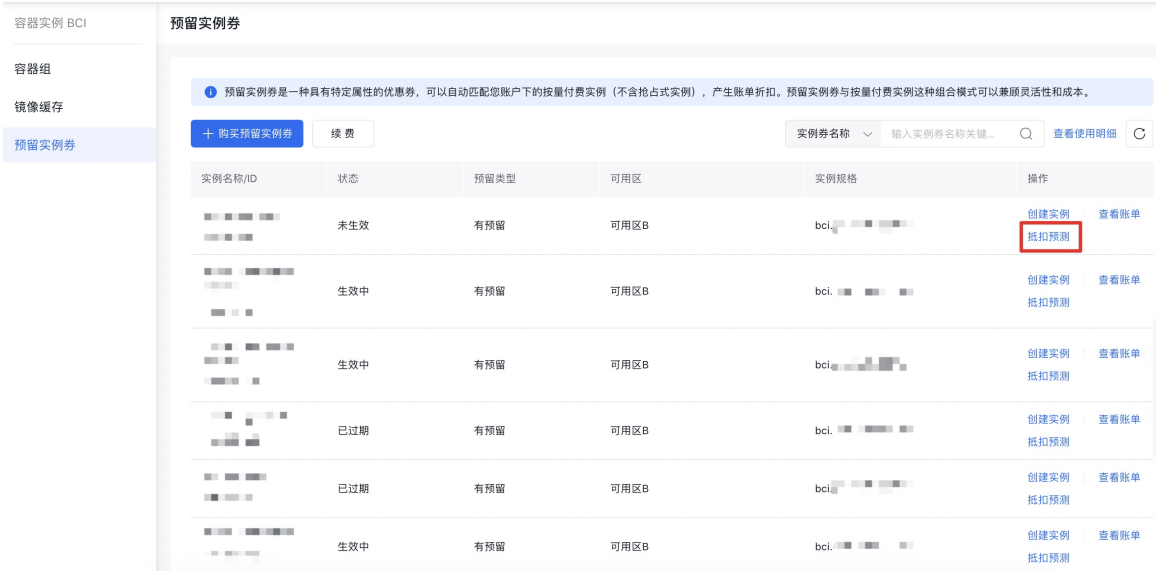
示例场景	按量付费实例	地域级预留实例券	抵扣效果
匹配成功	持有6台按量付费实例，配置如下： 3台配置如下： 华北-保定，可用区A bci.0.25c0.5m 3台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：6台	预留实例券和6台按量付费实例完全匹配，1张预留实例券每小时抵扣6台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.0.25c0.5m实例，保证随时可以在华北-保定地域中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 12台配置如下： 华北-保定，可用区A bci.0.25c0.5m 8台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例，配置如下： 华北-北京，可用区A bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：2台	导致匹配失败的原因如下： 2台实例可用区为华北-北京，可用区A 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

查看可抵扣的实例

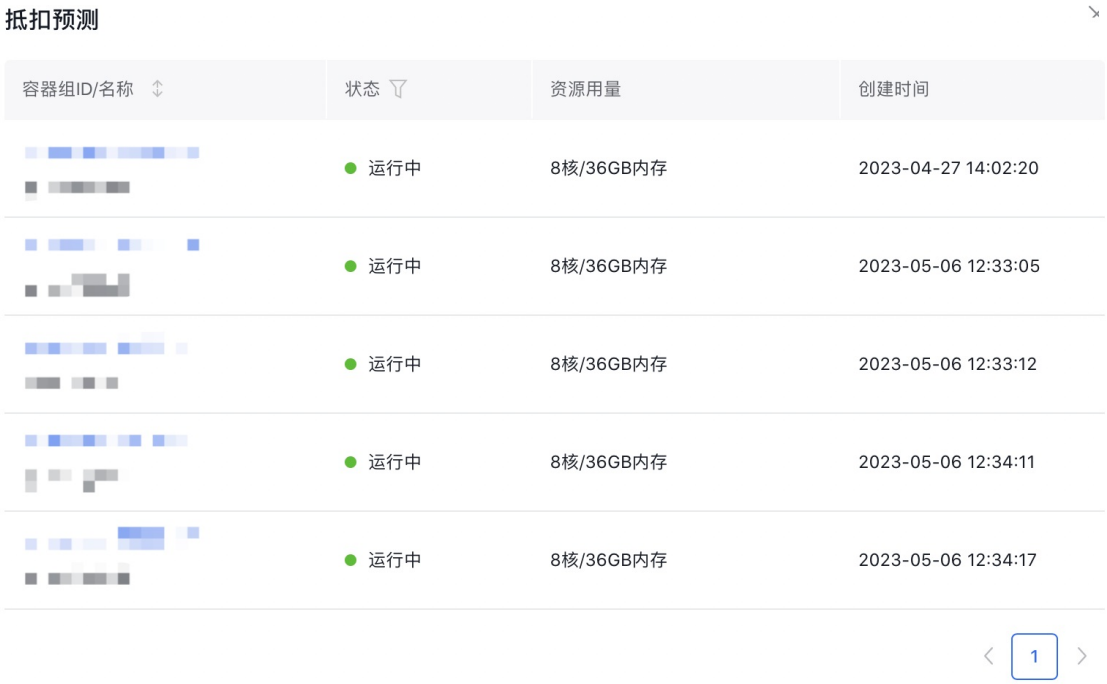
预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。通过抵扣预测功能查看到实例仅代表实例符合匹配要求，并不代表一定抵扣这些实例的账单，实际抵扣情况以账单为准。

操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。
3. 在**预留实例券**页面，在操作栏单击**抵扣预测**。



4. 在抵扣预测页面查看预留实例券可以匹配并抵扣账单的按量付费实例。



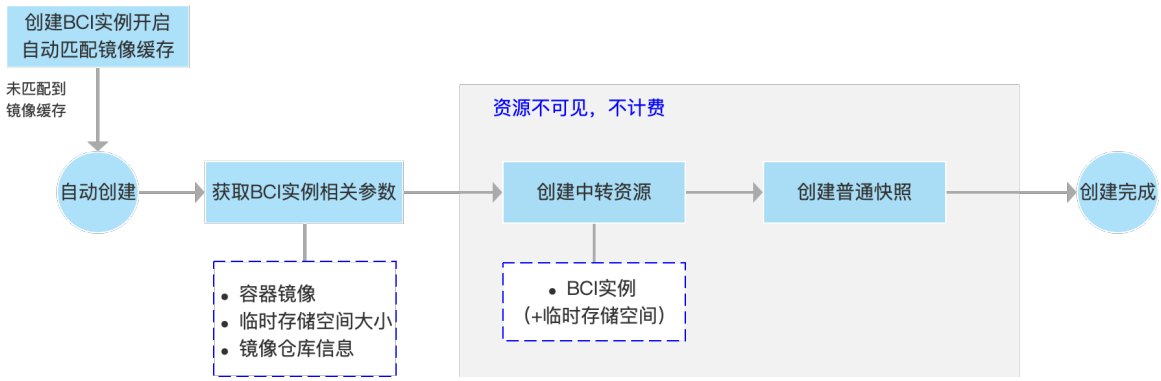
镜像

镜像缓存

使用镜像缓存（ImageCache）创建BCI实例可以加速拉取镜像，减少BCI实例的启动耗时。本文介绍镜像缓存的基本功能、创建和使用方式、以及计费说明等。 **功能简介** 在运行容器前，BCI需要先拉取您指定的容器镜像，但因网络和容器镜像大小等因素，镜像拉取耗时往往成了BCI实例启动的主要耗时。为加速实例的创建速度，BCI提供镜像缓存功能。您可以预先将需要使用的镜像制作成镜像缓存，然后基于该镜像缓存来创建BCI实例，避免或者减少镜像层的下载，从而提升实例的创建速度。

> **说明：**具体提升速度由BCI实例中使用的镜像个数、镜像大小和镜像仓库网络因素等决定。

创建方式 目前只支持自动方式创建镜像缓存。在创建BCI实例时，会为其自动匹配镜像缓存。在未匹配到镜像缓存的情况下，系统会在创建BCI实例的同时，自动创建一个镜像缓存。具体过程如下：



- 一个镜像缓存对应一份CCR快照，自动创建的镜像缓存快照由百度云BCI管理。注：该镜像缓存如果在30天内未使用，将被自动删除。
- 创建过程中，系统将创建一个BCI实例，并使用BCI实例自带的存储空间来中转创建镜像缓存对应的快照。创建完成后，该实例将被自动释放。 **使用方式** 创建BCI实例时，使用镜像缓存可以加快BCI实例的创建。目前只支持自动匹配镜像缓存方式：
- 自动匹配：自动匹配使用最优的镜像缓存。目前只支持完全匹配策略。
  - 完全匹配策略：即镜像名称及版本是否完全相同。
    - 注意1：镜像tag不能为latest，必须指定一个明确的tag。否则，镜像缓存不会生效。
    - 注意2：如果用户修改了镜像，必须发布一个新的tag。否则，使用的缓存会比较老。

注意事项

- 创建镜像缓存需要拉取容器镜像，因此创建时长由镜像个数、镜像大小、网络等多种因素决定。
- 自动创建镜像缓存时采用实例中所声明的容器镜像。
  - 如果镜像为私有镜像，则需要提供私有镜像仓库的访问凭证，包括地址、用户名和密码。
  - 如果镜像需要通过公网拉取（如Docker官方镜像），则需要配置EIP或者NAT来访问公网。更多信息，请参见 连接公网。
  - 如果镜像由于远程仓库超时等原因导致拉取失败，推荐您将镜像仓库和VPC打通，或者可以使用容器镜像服务CCR，将镜像上传至百度云仓库。

计费说明

- 创建镜像缓存
  - 自动创建镜像缓存，无需付费。
    如果您的镜像需要通过公网拉取，则会产生相应的公网流量费用。
- 使用镜像缓存
  - 使用自动创建的镜像缓存创建BCI实例时，如果镜像缓存大于20GiB，请提前发工单联系，以避免空间不足导致失败。
  - BCI后续将增加临时存储空间计费项，超过20GiB需付费，否则只需支付BCI实例费用。

使用CCR镜像仓库

背景信息

CCR镜像仓库分为：个人版，企业版，建议使用企业版镜像仓库

个人版：容器镜像服务个人版主要面向个人开发者或企业客户临时测试使用，提供基础的容器镜像托管、分发等能力，同时在资源配额上有一定的限制，不推荐。

企业版：容器镜像服务企业版主要面向企业用户业务生产使用，提供容器镜像、Helm Chart等云原生制品生命周期管理，支持

多地域、多场景的制品高效分发，同时企业版会不断完善其功能特性并持续更新，帮助企业在业务生产过程中降本增效。

更多信息，请参考容器镜像服务CCR。

前提条件 需要创建好CCR镜像仓库，此处以企业版镜像仓库为例。

创建命名空间：可创建公有命名空间或私有命名空间，私有命名空间需要登录密钥才可访问。

若使用私有命名空间，需要设置访问凭据。

使用CCR镜像仓库

操作步骤

1.确定CCR企业版的网络访问方式：

- 通过公网访问，配置公网访问控制
- 通过VPC内访问，配置私有网络访问控制

在配置好访问控制后，使用响应的镜像仓库地址创建BCI实例即可

2.（可选）如果您使用私用仓库，需要需要填写访问密钥，BCI只支持通过ImagePullSecrets字段指定访问密钥。

- 创建访问秘钥

```
kubectl create secret docker-registry <name> --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

- 在POD中使用密钥

```
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey
```

使用第三方镜像仓库

使用第三方镜像仓库

BCI支持使用第三方镜像仓库，例如：自建镜像仓库，dockerhub及其他云厂商等第三方镜像仓库。

> **注意：**
>
> 访问第三方镜像仓库会产生额外的公网流量费用，请确定BCI实例的EIP功能已打开

操作步骤

1.BCI支持HTTP、HTTPS协议，并已经默认跳过TLS CA认证。 2.（可选）如选择访问私有镜像仓库，需要使用secret进行登录：

- 创建secret

```
kubectl create secret docker-registry <name> --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

相关参数说明

创建Secrect 参数	说明
name	secret的名称
DOCKER_REGISTRY_SERVER	镜像仓库地址
DOCKER_USER	镜像仓库用户名
DOCKER_PASSWORD	镜像仓库登录密码
DOCKER_EMAIL	邮箱

- 使用secret示例

```
  apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1  #镜像地址
  imagePullSecrets:  #镜像仓库是public类型无需secret
    - name: myregistrykey  #创建的secret名称，为上面name的值
```

网络

连接公网

连接公网

如果您的BCI实例（即BCI Pod）有连接公网的需求，则需要配置NAT网关或者弹性公网IP，并支付相应的网络费用。

本文介绍如何为您的BCI实例绑定EIP，或者为BCI实例所属的VPC绑定NAT网关，以实现BCI实例与公网互通。

背景信息

为BCI实例配置公网服务时，支持以下两种方式：

方式	说明	费用
BCI实例绑定EIP	EIP是独立购买的可单独持有的公网IP地址，可以绑定到BCI实例上提供公网服务。具体详情请参见： 百度云EIP介绍。	EIP支持按固定带宽或者按使用流量计费。
BCI实例所属的VPC绑定NAT网关	NAT网关是可独立购买的网关产品，绑定EIP后，可以为关联VPC下的所有BCI实例提供公网服务。具体详情请参见：百度云NAT网关介绍。	NAT网关支持包年包月和按量付费。NAT网关需绑定EIP后才能具备公网能力，即除NAT网关费用外，您还需支付EIP费用。

您可以根据业务需要，选择合适的方式来配置公网服务：

- **示例场景一：单个BCI实例配置Nginx外网访问**

如果您有一个BCI实例用于部署Nginx服务，在创建实例时，您需要为该实例绑定EIP。当Nginx启动时，将自动暴露80端口到EIP。您可以通过EIP地址加端口的方式访问该BCI实例的Nginx服务。

- **示例场景二：多个BCI实例拉取Docker Hub镜像**

BCI默认不提供外部公网链路进行公网镜像的拉取。如果您有多个BCI实例需要从Docker Hub拉取镜像，您可以为BCI实例所属的VPC绑定NAT网关来实现公网访问，否则镜像将拉取失败。

> **注意：**
>
> 如果BCI实例已经绑定了EIP，则优先使用BCI实例绑定的EIP来访问公网，而不会使用NAT网关的SNAT功能访问公网。

操作指南

连接公网方式一：为BCI实例绑定EIP

创建BCI Pod时，您可以直接为Pod绑定EIP。支持在Pod metadata中添加Annotation来自动创建并绑定一个EIP。 相关Annotation如下：

Annotation	示例值	说明
bci.virtual-kubelet.io/bci-create-eip	"true"	是否自动创建并绑定EIP
bci.virtual-kubelet.io/bci-create-eip-route-type	"BGP"	设置EIP的线路类型。不指定时，默认为BGP类型。可选值： BGP：标准型BGP BGP_S：增强型BGP ChinaMobile：中国移动 ChinaUnicom：中国联通 ChinaTelcom：中国电信
bci.virtual-kubelet.io/bci-create-eip-bandwidth	"10"	设置EIP带宽。带宽能力和EIP线路类型相关。带宽区间： BGP：1～200 Mbps BGP_S：100～5000 Mbps 移动、联通、电信：1～5000 Mbps
bci.virtual-kubelet.io/bci-create-eip-paymethod	"ByBandwidth"	设置EIP计费方式： ByTraffic：流量 ByBandwidth：带宽
bci.virtual-kubelet.io/bci-eip-ip	"127.0.0.1"	绑定已有的EIP。且此EIP状态必须可用(Available)。 **限定条件**：一个EIP只能成功绑定到一个BCI实例上，若使用deployment方式，副本数replicas需设置为1，否则可能导致创建失败。

> **注意：**
>
> 如EIP的线路类型选择移动、联通、电信，需要先提交工单申请百度云EIP单线白名单，否则该BCI Pod会创建失败。
>
> 如EIP的线路类型选择增强BGP（BGP_S）且要按照流量付费，需要先提交工单申请百度云EIP计费白名单，否则该BCI Pod会创建失败。

配置示例：

- 示例一：使用默认参数自动创建EIP（默认线路类型为BGP，带宽为100Mbps）

```
 apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-create-eip": "true" # 自动创建并绑定EIP
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

- 示例二：自动创建EIP，并设置EIP带宽和线路类型

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-create-eip": "true"              # 自动创建并绑定EIP
    "bci.virtual-kubelet.io/bci-create-eip-route-type": "BGP_S"  # 设置EIP线路类型
    "bci.virtual-kubelet.io/bci-create-eip-bandwidth": "200"     # 设置EIP带宽
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

- 示例三：绑定已有的EIP

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-eip-ip": "127.0.0.1"    # 指定已经存在的EIP
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

查询BCI实例绑定的EIP信息

如果BCI实例已成功绑定EIP，则该实例yaml信息的Annotations中将包含以下字段：

Annotation	示例值	说明
bci.virtual-kubelet.io/bci-bound-eip-id	"ip-xxxx"	EIP实例id
bci.virtual-kubelet.io/bci-bound-eip-ip	"127.0.0.1"	EIP公网地址
bci.virtual-kubelet.io/bci-bound-eip-bandwidth	"100"	EIP公网带宽
bci.virtual-kubelet.io/bci-bound-eip-paymethod	"ByTraffic"	EIP计费方式
bci.virtual-kubelet.io/bci-bound-eip-route-type	"BGP"	EIP线路类型

连接公网方式二：为VPC绑定NAT网关

您可以在专有网络控制台为VPC绑定NAT网关，并为NAT网关绑定EIP，使其能够提供NAT代理（SNAT和DNAT）功能：

- SNAT功能：可以为VPC中没有公网IP的BCI实例提供访问公网的代理服务。
- DNAT功能：可以将NAT网关绑定的EIP映射给VPC中的BCI实例使用，使其能够面向公网提供服务。

操作步骤如下：

1. 登录私有网络VPC控制台。
2. 在顶部菜单栏，选择地域。
3. 在左侧菜单栏，选择“网络连接”->“NAT网关”，进入NAT网关页面，创建NAT网关。
    1. 单击创建NAT网关。
    2. 完成购买NAT网关相关的参数配置。配置时，“所在网络”选择BCI实例所属的VPC。更多信息，请参见NAT网关。
    3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。
4. 在弹性公网IP控制台，创建EIP。
    1. 在顶部菜单栏，选择地域后，单击“创建实例”。
    2. 完成购买EIP相关的参数配置。更多信息，请参见创建EIP实例。
    3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。
5. 绑定EIP与NAT网关。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“更多”->“绑定公网IP”。
    2. 在弹出的对话框中选择要绑定的EIP，然后单击确定。
6. 如果您的BCI实例需要访问公网，您需要为NAT网关创建SNAT条目，并在VPC中添加该NAT网关的路由。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置SNAT”。
    2. 单击“添加SNAT条目”。
    3. 配置SNAT相关条目的参数，然后单击确定。配置时，“源网段”选择BCI所属的子网。更多信息，请参见NAT网关的配置SNAT表。
    4. 在私有网络页面，单击要操作的VPC名称进入VPC详情页，单击“路由表”，然后单击“添加路由”并配置路由参数。更多信息，请参见路由表的添加路由。
7. 如果您的BCI实例需要面向公网提供服务，您需要为NAT网关创建DNAT条目。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置DNAT”。
    2. 单击“添加DNAT条目”。

3. 配置DNAT相关条目的参数，然后单击确定。更多信息，请参见NAT网关的配置DNAT表。

支持自定义DNSConfig

为了兼容原生kubernetes Pod使用节点默认NameServers能力，CCE+BCI提供vk支持自定义DNSConfig的方案。 **1. VirtualKubelet DNSConfig 介绍**

DNSConfig 功能介绍参考：https://kubernetes.io/zh-cn/docs/concepts/services-networking/dns-pod-service/
在 BCI 中 对 DNSPolicy 和 DNSConfig 支持情况如下：

DNS策略	说明	备注
Default	Pod 从运行所在的节点继承名称解析配置。	
ClusterFirst	与配置的集群域后缀不匹配的任何 DNS 查询（例如 "www.kubernetes.io"） 都会由 DNS 服务器转发到上游名称服务器。集群管理员可能配置了额外的存根域和上游 DNS 服务器。	
ClusterFirstWithHostNet	对于以 hostNetwork 方式运行的 Pod，应将其 DNS 策略显式设置为 "ClusterFirstWithHostNet"。否则，以 hostNetwork 方式和 "ClusterFirst" 策略运行的 Pod 将会做出回退至 "Default" 策略的行为。	在 BCI 中，不支持使用 HostNetwork；所以不支持该 DNS 策略；
None	此设置允许 Pod 忽略 Kubernetes 环境中的 DNS 设置。Pod 会使用其 dnsConfig 字段所提供的 DNS 设置。	

2. 使用限制

- kubernetes 集群版本 >= 1.18
- cce-virtual-kubelet 版本 >= 0.3.1 **3. 注意事项 建议用户自定义 Pod DNSConfig 时，NameServers 中建议仅包含一个 IP 地址。** 原因如下：
- Kubernetes 限制 DNSConfig 的 NameServers 中最多包含 3 个 IP；
- BCI 后端会默认在 Pod 的 DNSConfig 中注入一个 IP；
- 当需要在用户集群使用 Service 服务发现时，VirtualKubelet 也会默认在 DNSConfig 中注入一个 IP；

3.1 设置自定义DNSConfig的方法

- 通过CCE组件中心安装VK组件且cce-virtual-kubelet版本大于等于0.3.15，通过bci-profile设置自定义dns配置，详见[操作文档]；(https://cloud.baidu.com/doc/BCI/s/Vm5ux9u97)
- helm安装且cce-virtual-kubelet版本小于0.3.15，则直接修改helm charts中dnsConfig配置；

4. 使用场景 4.1 使用 DNSPolicy 为 Default 该场景下，按照 Kubernetes 行为，预期为：“Pod 从运行所在的节点继承名称解析配置”，即需要继承 **VKNode 的解析配置**。
在使用该功能之前，您需要先对 VirtualKubelet 完成配置，指定注册到 VKNode 中的 DNSConfig：

在 cce-virtual-kubelet 的 helm charts 中包含 dnsConfig 配置，默认为空：

```
dnsConfig: {}
```

你可以按照自己的期望配置，例如：

```
 dnsConfig:
  nameservers:
    - 169.254.20.10
    - 172.21.0.10
  options:
    - name: ndots
      value: "3"
    - name: attempts
      value: "2"
    - name: timeout
      value: "1"
  searches:
    - default.svc.cluster.local
    - svc.cluster.local
    - cluster.local
```

在创建 Pod 时，您需要将 dnsPolicy 指定为 *Default*，例如：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: default
  template:
    metadata:
      labels:
        app: default
    spec:
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Equal"
        value: "baidu"
        effect: "NoSchedule"
      containers:
      - name: main
        image: hub.baidubce.com/cce/nginx-alpine-go:latest
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
      dnsPolicy: Default # 指定 dnsPolicy 为 Default
```

此时 VirtualKubelet 将在 BCI 创建 Pod 时，继承您在 VirtualKubelet 中指定的 DNSConfig。

注意，当 DNSPolicy 为 Default 时，VirtualKubelet 不会自动为注入 DNSConfig。

4.2 使用 DNSPolicy 为 None 该场景下，按照 Kubernetes 行为，预期为：“Pod 会使用其 dnsConfig 字段所提供的 DNS 设置”。您无需对 VirtualKubelet 做额外配置。 则用户需要明确在 Pod 中填写dnsConfig：

```
  apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 192.0.2.1 # 这是一个示例
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
    options:
      - name: ndots
        value: "2"
```

此时 VirtualKubelet 将在 BCI 创建 Pod 时，将直接使用您在 Pod 中填写的 DNSConfig。

注意，当 DNSPolicy 为 None 时，VirtualKubelet 不会自动为注入 DNSConfig。

4.3 使用 DNSPolicy 为 ClusterFirst 该场景下，按照 Kubernetes 行为，预期为：“转发到上游名称服务器”。
当前在 BCI 场景下与标准 Kubernetes 有区别，需要在 Pod 中添加 annotation bci.virtual-kubelet.io/kube-proxy-enabled: "true"，才会默认注入 DNSConfig，用于解析集群中的域名。
参考：https://cloud.baidu.com/doc/CCE/s/Qkhfxcry7
示例 Pod：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
spec:
  selector:
    matchLabels:
      app: busybox
  replicas: 1
  template:
    metadata:
      labels:
        app: busybox
      annotations:
        bci.virtual-kubelet.io/kube-proxy-enabled: "true" # 开启 kube-proxy
    spec:
      containers:
      - name: busybox
        image: busybox
        command: [ "/bin/sh", "-c", "sleep 3600" ]
```

VirtualKubelet 在创建 Pod 时，将注入如下 DNSConfig，用于实现 集群内 的服务发现：

```
dnsConfig:
nameservers:
  - ${coredns clusterip}
options:
  - name: ndots
    value: "5"
searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
```

通过Service访问BCI服务

概述 如果您的BCI实例（即BCI Pod）需要通过BLB被外网访问，那么您需要在您的CCE 集群创建LoadBalancer类型的Service，详情见: 通过YAML创建LoadBalancer_Service

操作示例 当用户创建类型是LoadBalancer的Service，默认情况下，CCE集群会联动地创建与之对应的BLB，并为此BLB绑定EIP。

假设创建如下的nginx服务，将该服务调度到BCI上，同时创建一个与该服务对应的k8s LoadBalancer类型的Service。 示例所用nginx.yaml，内容如下：

```
  kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  type: LoadBalancer
  ports:
  - name: nginx-port
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Equal"
        value: "baidu"
        effect: "NoSchedule"
```

1. 向CCE集群提交YAML

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

2. 查询该nginx Service绑定的EIP

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

如查询得到，CLUSTER-IP字段为集群内ClusterIP, EXTERNAL-IP为EIP

3. 查询该nginx Service绑定的BLB id。

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

如查询所得，其lb-xxxxxx即为此Service的BLB的id。

4. 通过EIP访问服务。

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

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

- Cluster模式的LoadBalancer Service，当其收到数据包后，负载均衡器将数据包发往集群的某个节点上，进而节点二次转发数据包到集群的某个Pod上。目标Pod的宿主节点和转发数据包的节点可能不是同一个；这种情况数据包的源IP地址会丢失。
- LB直连Pod模式的LB Service，当其收到数据包后，负载均衡器直接将数据包发往各个Pod。相比前两种模式，这种模式减少了一次节点转发操作；这种情况数据包的源IP地址不会丢失。

Cluster模式

如果希望使用Cluster模式的Service，在创建Service时，您可以显式指定externalTrafficPolicy: Cluster，因该字段默认值为Cluster，您也可不指定指定，显式指定该字段如以下例子所示：

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

LB直连Pod模式 如果希望使用LB直连Pod模式的Service时，在创建Service时，应添加Annotationservice.beta.kubernetes.io/cce-load-balancer-backend-type: "eni"。如以下例子所示： 注：一个BLB默认最多可挂载Pod数,默认值为200，也可自定义设置，详见设置BLB默认挂载的最大后端数

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

配置BCI Pod所属安全组

概述 安全组是一种虚拟防火墙，具备状态检测和数据包过滤能力，用于在云端划分安全域。通过添加安全组规则，您可以控制安全组内BCI Pod（即BCI实例）的入流量和出流量。

安全组介绍 安全组是一个逻辑上的分组，由同一VPC内具有相同安全保护需求并相互信任的实例组成。通过添加安全组规则，安全组可以允许或拒绝安全组内BCI实例对公网或者私网的访问，以及管理是否放行来自公网或私网的访问请求。

安全组分为普通安全组和企业安全组。如果您对整体规模和运维效率有较高需求，建议您使用企业安全组。相比普通安全组，企业安全组大幅提升了组内支持容纳的实例数量，简化了规则配置方式。更多关于两种安全组的差异信息和具体操作，请参考安全组

> 注意：一个安全组可管理同一个VPC内多个BCI实例

指定安全组 在CCE集群中创建BCI Pod时，BCI Pod默认会加入到cce-virtual-kubelet组件安装时配置的安全组中。如果有特殊需求，您也可以为某些BCI Pod指定其他安全组。 **操作前提** 操作前，请先创建安全组，具体操作，请参见文档创建安全组。

> 创建成功的BCI Pod不支持修改所属安全组。如果想要变更安全组，需要重新创建BCI Pod。

BCI Pod配置 对于有特殊需求的某些BCI Pod，可以添加bci.virtual-kubelet.io/bci-security-group-id的Annotation来指定安全组。配置要求如下：

- 支持指定一个或多个安全组，最多可以指定5个安全组。
- 指定的安全组必须属于同一VPC。
- 指定的安全组的类型必须相同，即同为普通安全组或同为企业安全组。

> Annotation 请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，已成功的BCI Pod只能重建来生效

配置如下：

```
  apiVersion: v1
kind: Pod
metadata:
  name: test
  labels:
    app: nginx
  annotations:
    "bci.virtual-kubelet.io/bci-security-group-id": "g-xxxx" #配置安全组
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - name: sun-nginx-host
    imagePullPolicy: IfNotPresent
    image: hub.baidubce.com/jpaas-public/nginx-alpine-go:latest
    resources:
      limits:
        cpu: 0.5
        memory: 1Gi
      requests:
        cpu: 0.5
        memory: 1Gi
```

添加安全组规则 对于安全组内的BCI实例，您可以添加安全组规则来控制其出入流量。例如：

- 当BCI实例需要与所在安全组之外的网络进行通信时，您可以添加允许访问的安全组规则，实现网络互通。
- 在运行BCI实例的过程中，发现部分请求来源有恶意攻击行为时，您可以添加拒绝访问的安全组规则，实现网络隔离。

关于如何添加安全组规则，请参见编辑安全组

配置BCI Pod访问集群内Service

概述

BCI支持从BCI实例中访问K8S ClusterIP类型Service。如果您需要在BCI实例中访问集群中ClusterIP, 您需要在Pod Spec Yaml中添加如下Annotation，用于开启功能:

`bci.virtual-kubelet.io/kube-proxy-enabled: true`

> Annotation 请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，已成功的BCI Pod只能重建来生效。

配置示例 以创建一个简单的Nginx为例，Nginx的Pod 会调度到VNode上：

```
  kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - name: nginx-port
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-bci
  labels:
    app: nginx
  annotations:
    "bci.virtual-kubelet.io/kube-proxy-enabled": "true" #添加这个annotation
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - name: sun-nginx-host
    imagePullPolicy: IfNotPresent
    image: hub.baidubce.com/jpaas-public/nginx-alpine-go:latest
    resources:
      limits:
        cpu: 0.5
        memory: 1Gi
      requests:
        cpu: 0.5
        memory: 1Gi
```

存储

挂载CFS文件存储

挂载CFS文件存储

文件存储CFS(Cloud File Storage)是百度智能云提供的安全、可扩展的文件存储服务。通过标准的文件访问协议，为云上的计算资源提供无限扩展、高可靠、全球共享的文件存储能力。本文为您介绍挂载CFS文件存储。更多CFS相关信息，请参见CFS说明。

前提条件

请确保您已经创建CFS并已获得CFS挂载地址（CFS Server）。

> **注意：**
>
> - CFS为共享存储，一个CFS可以挂载到多个BCI实例上。此时，如果多个BCI同时修改相同数据，请进行同步与冲突保护。
> - 在删除所有使用此挂载点的BCI实例前，请勿删除CFS挂载点，否则可能会造成操作系统无响应。

配置示例

CFS Volume

```
 apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: registry.k8s.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /my-cfs-data
      name: test-volume
  volumes:
  - name: test-volume
    nfs:
      server: my-cfs-server.example.com #此为cfs挂载点
      path: / #此为server端目录，cfs中为根目录不可变更
      readOnly: true
```

CFS PVC

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Mi
  volumeName: cfs
```

CFS PV

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: cfs
spec:
 capacity:
  storage: 1Mi
 accessModes:
  - ReadWriteMany
 nfs:
  server: cfs-server.default.svc.cluster.local  #此为cfs挂载点
  path: "/" #此为server端目录，cfs中为根目录不可变更
 mountOptions: #当前不支持指定挂载选项
  - cfsvers=4.2
```

使用CFS PVC的Deployment

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: cfs-web
spec:
 replicas: 2
 selector:
  matchLabels:
   role: web-frontend
 template:
  metadata:
   labels:
    role: web-frontend
  spec:
   containers:
   - name: web
    image: nginx
    ports:
     - name: web
      containerPort: 80
    volumeMounts:
        # name must match the volume name below
        - name: cfs
          mountPath: "/usr/share/nginx/html"
   volumes:
   - name: cfs
    persistentVolumeClaim:
     claimName: cfs
```

挂载EmptyDir数据卷

挂载EmptyDir数据卷

本文介绍如何挂载EmptyDir数据卷。EmptyDir数据卷是一个空的目录，用于临时存放数据，便于容器之间共享数据。

注意事项

EmptyDir为临时存储，当BCI实例删除或重启时，EmptyDir数据卷中保存的数据均会被清空。

> **提示：**
>
> 注意：当前暂不支持指定临时存储空间大小，不支持基于内存的临时存储。

操作步骤

1.声明数据卷

通过Volume相关参数声明数据卷时，需要先明确Volume的名称和类型。再根据Volume.N.Type的取值，进一步配置该类型数据卷的相关参数。

名称	类型	示例值	描述
Volume.N.Name	String	emptydir-demo	数据卷名称
Volume.N.Type	String	emptyDir	取值为EmptyDirVolume，表示创建一个EmptyDir类型的数据卷

2.挂载数据卷

声明数据卷后，可以通过VolumeMount相关参数将数据卷挂载到容器中。

名称	类型	示例值	描述
Container.N.VolumeMount.N.Name	String	test-volume	要挂载到容器的数据卷的名称，对应Volume.N.Name的值
Container.N.VolumeMount.N.MountPath	String	/usr/share	挂载目录。容器挂载目录下的内容会被数据卷的内容直接覆盖，请准确填写
Container.N.VolumeMount.N.ReadOnly	Boolean	false	挂载目录是否只读。默认为false。

3.配置示例

使用vk创建pod参数示例如下

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: registry.k8s.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache  #容器内挂载路径
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir:{} #默认为文件型，使用节点的存储空间#
```

挂载ConfigMap数据卷

挂载ConfigMap数据卷

本文介绍如何挂载ConfigMap数据卷。ConfigMap数据卷是一个配置文件，用于向BCI实例注入配置数据，具体详见create-a-configmap;同时BCI已支持ConfigMap配置文件的动态更新，即在CCE集群内更新ConfigMap配置文件内容，BCI Pod使用的该配置文件会自动更新。

配置示例

1.声明configMap

```
  apiVersion: v1
kind: ConfigMap
metadata:
  name: game-config
  namespace: default
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
```

2.使用vk提交pod

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: config-file-test
  namespace: default
spec:
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: game-config
```

挂载PFS并行文件存储

PFS 概述

并行文件存储服务PFS (Parallel Filesystem Service)，是百度智能云提供的完全托管、简单可扩展的并行文件存储系统，针对高性能计算场景提供亚毫秒级的访问能力和高IOPS的数据读写请求能力。同时，百度智能云PFS提供简单、易操作的接口，免去部署、维护费用的同时，最大化提升您的业务效率。更多信息参见PFS说明。

> 注意：
> 挂载PFS并行文件存储功能正在内测中，如需使用可提交工单申请。

提前准备

创建PFS实例

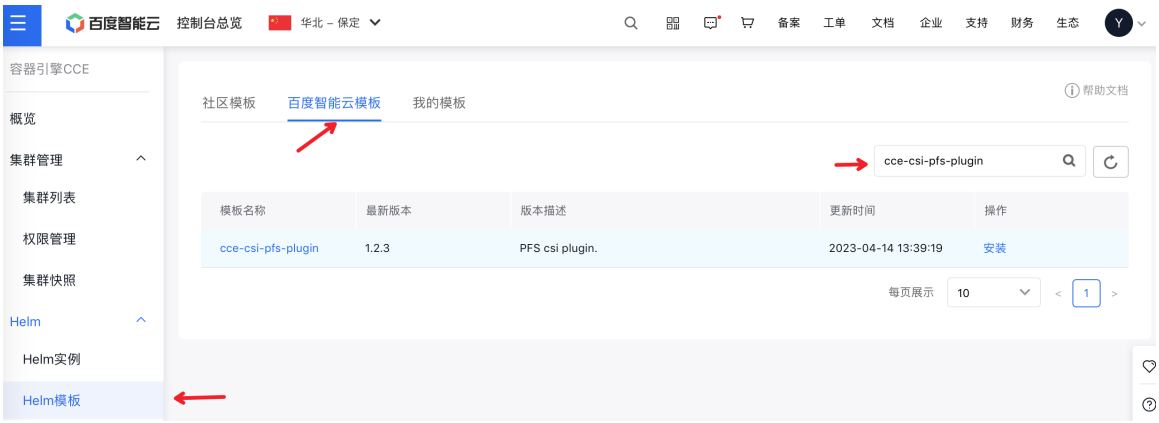
1. 创建PFS，操作步骤请参考创建并行文件系统。

> 注意：
> PFS实例须与容器集群处于同一VPC内。

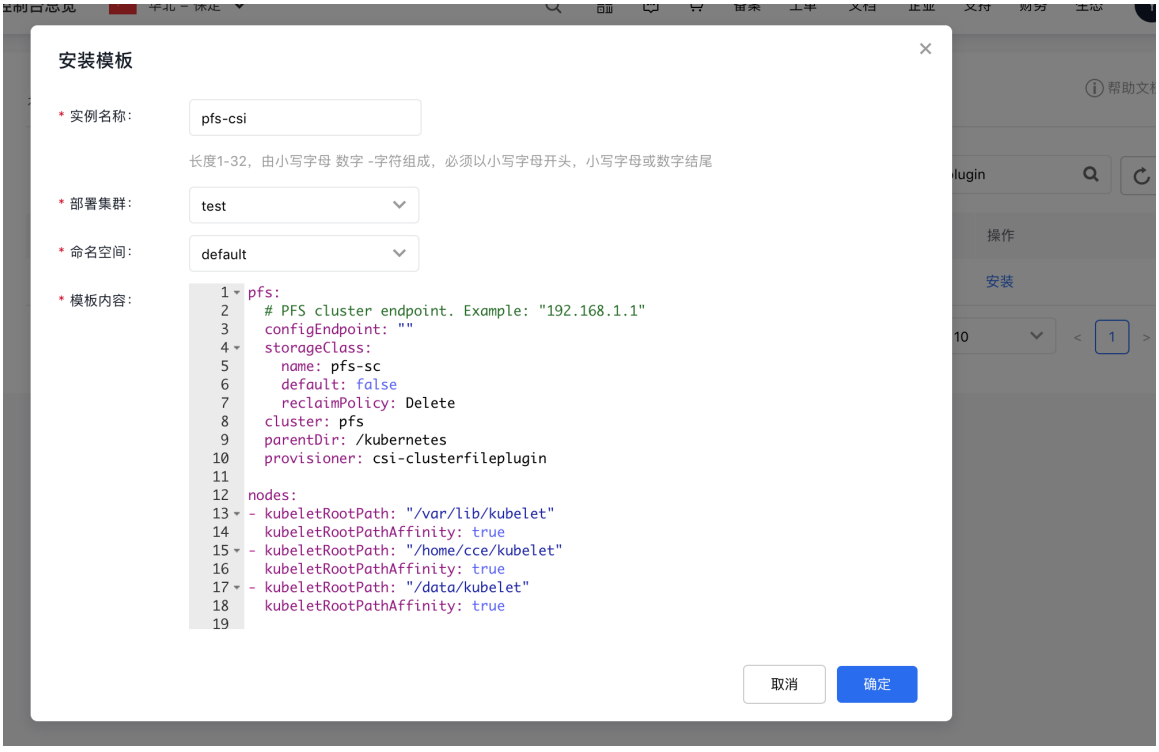
2. 从页面上获取PFS连接地址，以备后续部署存储插件时使用。

部署存储插件

3. 依次选择：容器引擎 CCE -> Helm 模版 -> 百度智能云模版，通过模版名称 cce-csi-pfs-plugin 搜索模版。



4. 点击安装并填写相应的参数。



- 实例名称：插件实例名称，例：pfs-csi；
- 部署集群：选择需要部署 PFS CSI 插件的容器集群；
- 命名空间：管理实例的 helm 元数据的命名空间，例：kube-system；
- pfs.configEndpoint: 填写从PFS页面上获取的PFS连接地址；
- pfs.storageClass: 创建的StorageClass属性
  - name: storageClass名称
  - default: 是否作为集群默认StorageClass
  - reclaimPolicy: PV回收策略，支持Delete或Retain。
- pfs.cluster: 动态创建的PV在parentDir下的子路径。例如对于多个不同的CCE集群挂载同一个PFS实例的场景，可以在不同CCE集群中安装时指定为不同的值，这样各集群动态创建的PV就在不同的目录下。
- pfs.parentDir: 代表CSI有权限读写的子路径，需要使用静态PV挂载的路径必须在这个路径之内，动态创建的PV则会在这个路径下的cluster子目录内自动创建新的路径。可以填 /（根目录）或者其它目录，默认为 /kubernetes。挂载前需确认该路径已创建，并且已经对CCE集群机器授权。

> Q1：如何创建parentDir？
> A1：在PFS所在VPC内找一台可登录的BCC/BBC机器进行目录创建操作，仅第一次在该PFS实例中使用到该目录时操作即可。创建操作可以通过在主机上挂载PFS实例后，手动在挂载点内创建目录的方式实现。
> Q2：parentDir如何对CCE集群授权？
> A2：对CCE集群授权请提交工单人工处理，提交工单时请按需提供如下信息：
>
> - 工单标题：CCE使用PFS，parentDir对CCE集群授权
> - PFS实例所在地域：如 “北京”
> - PFS实例ID：如 “pfs-xxx”
> - parentDir：如 “/kubernetes”

- pfs.provisioner: 动态创建时使用的Provisioner名称。
- nodes：如果部署集群的节点的时候，指定了 kubelet 的数据目录，需要填写具体使用的数据目录到该列表，否则保持默认即可；
  pfs配置项补充示例：

```
pfs:
parentDir:"/kubernetes"
cluster:"pfs"
```

如果安装时填写上述配置，则动态创建的PV在pfs中的路径为/kubernetes/pfs/pvc-xxxx-xxxx-xxxx-xxxx；静态PV在pfs中的路径必须位于/kubernetes路径下。
完成配置项填写后，点击确定即可安装。
检查插件状态 通过kubectl命令行工具，检查插件Pod状态都为Running，StorageClass 已正确创建。

```
 $ kubectl get pod -n kube-system | grep pfs
$ kubectl get sc pfs-sc
```

如果遇到插件Pod无法正常运行的情况，请提交产品类型为CCE的工单人工处理，提交工单时类型选择集群存储问题，并请提供

如下信息

- CCE集群所在地域；
- CCE集群ID（cce-开头）；
- PFS文件系统ID（pfs-开头）；
- 具体的问题描述，如PFS插件Pod状态异常/storageClass未创建。

配置示例

动态挂载

1. 创建PVC，注意storageClassName需要指定为上述安装插件时填写的storageClass名称。

```
 ###### 按需指定PVC存储空间
$ cat >pfs-pvc.yaml <<EOF
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-pvc-pfs
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: pfs-sc
  resources:
    requests:
      storage: 50Gi # 用户指定PVC存储空间
EOF
$ kubectl create -f pfs-pvc.yaml
```

2. 检查PVC状态为Bound。

```
 $ kubectl get pvc csi-pvc-pfs
NAME            STATUS    VOLUME               CAPACITY   ACCESS MODES   STORAGECLASS   AGE
csi-pvc-pfs     Bound     pvc-1ab36e4d1d2711e9   50Gi       RWX            pfs-sc         4s
```

3. 在 Pod 中挂载PVC，要求 Pod 和 PVC 在同一个 namespace 下。

```
 apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
  namespace: default
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: pfs-pvc
        mountPath: "/pfs-volume"
  volumes:
    - name: pfs-pvc
      persistentVolumeClaim:
        claimName: csi-pvc-pfs
```

静态挂载

1. 创建静态PV和PVC。
   通过 pv.spec.csi.volumeAttributes.path 指定需要挂载的路径，该路径为相对于安装插件时填写的parentDir路径的相对路径。

> - 注意storageClassName需要指定为上述安装插件时填写的storageClass不同的名称，防止被动态创建流程接管。不要求该storageClass在集群中存在，只要PV和PVC的storageClassName一致即可，否则PV和PVC无法互相绑定。
> - 例如：按照下述示例yaml创建，则该PV在pfs中的路径为 /kubernetes/exist/some/dir。如果对应的路径不存在，在挂载PV时会自动创建该路径。

```
 apiVersion: v1
kind: PersistentVolume
metadata:
  name: static-pv-pfs
spec:
  accessModes:
  - ReadWriteMany
  storageClassName: pfs-static-sc
  capacity:
    storage: 100Gi
  csi:
    driver: csi-clusterfileplugin
    volumeHandle: data-id
    volumeAttributes:
      # the path is the related path to parentDir
      path: exist/some/dir

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: static-pvc-pfs
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: pfs-static-sc
  resources:
    requests:
      storage: 100Gi
```

2. 检查PVC状态为Bound，并且与对应的PV绑定。

```
 $ kubectl get pvc
NAME           STATUS   VOLUME        CAPACITY   ACCESS MODES   STORAGECLASS    AGE
static-pvc-pfs   Bound    static-pv-pfs   100Gi      RWX           pfs-static-sc   10s
```

3. 在 Pod 中挂载PVC，要求 Pod 和 PVC 在同一个 namespace 下。

```
  apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
  namespace: default
  labels:
    app: test-pvc-pod
spec:
  containers:
  - name: test-pvc-pod
    image: nginx
    volumeMounts:
      - name: pfs-pvc
        mountPath: "/pfs-volume"
  volumes:
    - name: pfs-pvc
      persistentVolumeClaim:
        claimName: static-pvc-pfs
```

挂载BOS数据卷

BOS概述 百度对象存储 BOS (Baidu Object Storage) 提供稳定、安全、高效以及高扩展存储服务，支持单文件最大 48.8 TB 的文本、多媒体、二进制等任何类型的数据存储。本文介绍如何挂载BOS数据卷。更多信息，请参考：BOS说明 **提前准备 创建 BOS Bucket** 操作步骤请参考创建bucket

> **注意：**请注意创建的BOS Bucket与CCE集群应该处于同一地域，暂不支持跨地域挂载BOS

部署存储插件

1. 登录百度智能云官网，并进入管理控制台。
2. 选择“产品服务 > 容器 > 容器引擎 CCE”，单击进入容器引擎管理控制台。
3. 单击左侧导航栏中的 集群管理 > 集群列表 。
4. 在集群列表页面中，单击目标集群名称进入集群详情页面。
5. 在集群详情页面单击 组件管理 。
6. 在组件管理列表中选择存储 CCE CSI BOS Plugin 组件单击“安装”。

7. 点击安装并填写相应的参数

配置示例 静态PV/PVC方式挂载BOS 注意：由于CSI bosplugin 依赖 bosfs，在Pod中配置livenessProbe检查挂载点状态可以避免容器无法感知 bosfs 异常重启后的挂载点失效的问题，强烈建议添加该配置，具体配置项参考在Pod内挂载PVC中的例子。

1. 在集群中创建 AK/SK 的 secret，用以访问BOS存储。

```
kubectl create secret generic csi-bos-secret \
--from-literal=ak=<Your AK> \
--from-literal=sk=<Your SK>
```

关于 AK/SK 的更多信息参考：如何获取 AK 和 SK

2. 在集群中创建PV和PVC资源
   使用kubectl，执行kubectl create -f bos-pv.yaml完成PV的创建
   对应的bos-pv.yaml文件如下所示：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-bos
  namespace: "default"
spec:
  accessModes:
  - ReadWriteOnce
  - ReadOnlyMany
  capacity:
    storage: 5Gi
  storageClassName: csi-bos
  csi:
    driver: "csi-bosplugin"
    volumeHandle: "v-XXXXXX"
    nodePublishSecretRef:
      name: "csi-bos-secret"
      namespace: "default"
  mountOptions:
  - "-o meta_expires=0 -o preload_blocks=100"
  persistentVolumeReclaimPolicy: Retain
```

注意事项及参数说明：

* yaml中volumeHandle：对应的是BOS的 bucketName, 支持挂载 BOS bucket子目录 ， 如: bucketName/dirName

* nodePublishSecretRef：填写步骤1中的 secret 名
* mountOptions: 由于 BOS bucket 挂载依赖 Bosfs，因此支持通过 mountOptions 指定 Bosfs 的启动参数，支持的参数详情见[BOS参数说明](https://cloud.baidu.com/doc/BOS/s/Kjwvyqg72)，常用参数和说明如下:
    * meta_expires: meta缓存过期时间，单位为秒。不设置该参数代表永不过期。上述PV yaml示例中该参数被设置为0，代表不缓存。缓存过期时间过短可能影响写入性能，过长可能导致挂载点内看不到其他客户端写入同一bucket的数据。同一bucket一写多读场景建议设置为0，纯写入场景建议设置为3600，单点读写场景可以不设置。
    * preload_blocks: 指定预读blocks数, 单次预读数据为: preload_blocks*1MB。默认0, 如进行大文件下载(顺序读), 建议配置, 若进行随机读, 则不建议配置。
* BOS 支持一写多读，对应的 accessMode 只支持 ReadWriteOnce + ReadOnlyMany

创建PV后，输入kubectl get pv可以看见一个available状态的PV，如下所示：

```
 $ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM     STORAGECLASS   REASON    AGE
bos-pv   5Gi        RWO,ROX        Retain           Available             csi-bos                  3s
```

3. 建立一个能够与该PV绑定的PVC
   使用kubectl，执行kubectl create -f bos-pvc.yaml完成PVC的创建
   对应的bos-pvc.yaml文件如下所示：

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: bos-pvc
spec:
 accessModes:
 - ReadWriteOnce
 - ReadOnlyMany
 resources:
   requests:
     storage: 5Gi
 storageClassName: csi-bos
```

注意：yaml中storageClassName字段用于和 PV关联，建议填写，如果集群中使用多类存储系统的 PV

绑定前，PVC为pending状态

```
 $ kubectl get pvc
NAME     STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
bos-pvc  Pending                                       csi-bos        2s
```

绑定后，PV和PVC状态变为Bound

```
 $ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM             STORAGECLASS   REASON    AGE
bos-pv   5Gi        RWX            Retain           Bound     default/bos-pvc                  36s
$ kubectl get pvc
NAME     STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
bos-pvc  Bound     bos-pv    5Gi        RWO,ROX        csi-bos        1m
```

有关PV和PVC的更多设置和字段说明，见k8s官方文档

4. 在Pod内挂载PVC
   在Pod spec内指定相应的PVC名称即可，使用kubectl，执行kubectl create -f demo-bos-pod.yaml完成pod的创建
   对应的demo-bos-pod.yaml文件如下所示：

```
  apiVersion: v1
kind: Pod
metadata:
  name: nginx01
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx01
    volumeMounts:
    - mountPath: /var/lib/www/html
      name: bos-pvc
    - mountPath: /var/lib/www/html000
      name: bos-pvc
      readOnly: true
    livenessProbe:
      exec:
        command:
        - stat
        - /var/lib/www/html
  volumes:
  - name: bos-pvc
    persistentVolumeClaim:
      claimName: bos-pvc
      readOnly: false
```

Pod创建后，可以读写容器内的/var/lib/www/html路径来访问相应的BOS存储上的内容, 同时该路径支持读写，/var/lib/www/html000支持只读。

注意：由于CSI bosplugin 依赖 bosfs，在Pod中配置livenessProbe可以避免容器无法感知 bosfs 异常重启后的挂载点失效的问题。

同时，支持在其余机器上挂载只读盘，kubectl create -f demo-bos-pod1.yaml创建一个包含只读 bos bucket的 pod

```
 apiVersion: v1
kind: Pod
metadata:
 name: nginx01-bbaa
spec:
 containers:
 - image: nginx
   imagePullPolicy: Always
   name: nginx01
   terminationMessagePath: /dev/termination-log
   terminationMessagePolicy: File
   volumeMounts:
   - mountPath: /var/lib/www/html000
     name: bos-pvc
     readOnly: true
   livenessProbe:
     exec:
       command:
       - stat
       - /var/lib/www/html000
 volumes:
 - name: bos-pvc
   persistentVolumeClaim:
     claimName: bos-pvc
     readOnly: true
```

5. 释放PV和PVC资源

   完成存储资源的使用后，可以释放PVC和PV资源

   使用以下命令可以释放PVC

```
$ kubectl delete -f  bos-pvc.yaml
```

释放PVC后，原来与之绑定的PV状态会变为Release，如下所示：

```
 NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM           STORAGECLASS   REASON   AGE
bos-pv   5Gi        RWO,ROX        Retain           Released   default/bos-pvc   csi-bos        16m
```

输入以下指令释放PV资源

```
$ kubectl delete -f  bos-pv.yaml
```

挂载容器标准输出日志

概述

本文介绍如何通过挂载 stdout flexVolume，实现将容器的标准输出日志以 root 权限挂载到 BCI Pod 内。

配置示例 stdout volume 定义

```
 volumes:
- name: stdout  # volume 名称可自定义
 flexVolume:
   driver: "k8s/sidecar-stdout"
```

Pod 示例

```
  apiVersion: v1
kind: Pod
metadata:
  name: stdout-test
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - 'i=0; while true; do echo "{\"time\":\"$(date -u +"%FT%T.999Z")\",\"INFO\":$i}"; i=$((i+1)); sleep 10; done'
    image: registry.baidubce.com/glen-centos/centos:centos6
    imagePullPolicy: Always
    name: stdout-test
    resources:
      limits:
        cpu: 250m
        memory: 0.5Gi
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
      - name: stdout
        mountPath: /stdout   # 可自定义路径。如果同时对标准输出进行日志采集，需要确保挂载路径是 /stdout
  volumes:
    - name: stdout
      flexVolume:
        driver: "k8s/sidecar-stdout"
  nodeSelector:
    type: virtual-kubelet
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoSchedule
    key: virtual-kubelet.io/provider
    operator: Equal
    value: baidu
```

部署 Pod 并查看挂载结果

```
  kubectl apply -f stdout-test.yaml
kubectl exec -it stdout-test -- /bin/bash
ls -al /stdout
ls -al /stdout/stdout-test
```

```
[root@stdout-test /]# ls -al /stdout
total 12
drwxr-x--- 3 root root 4096 May 13 08:00 .
drwxr-xr-x 1 root root 4096 May 13 08:00 ..
drwxr-xr-x 2 root root 4096 May 13 08:00 stdout-test
[root@stdout-test /]# ls -al /stdout/stdout-test/
total 12
drwxr-xr-x 2 root root 4096 May 13 08:00 .
drwxr-x--- 3 root root 4096 May 13 08:00 ..
-rw-r----- 1 root root 2080 May 13 08:04 0.log
```

您可以使用 修改实例密码接口 为指定实例重置密码。

容器配置

设置容器启动命令和参数

设置容器启动命令和参数

BCI实例（即BCI Pod）通过容器镜像中的预设参数来启动容器。如果您在构建镜像时没有设置启动命令和参数，或者想要变更启动命令和参数，可以在创建BCI Pod时设置。本文介绍如何为容器设置启动时要执行的命令和参数。

功能说明

如果您想覆盖镜像中设置的启动默认值，包括工作目录、启动命令和参数，可以通过以下参数进行配置：

- 工作目录镜像构建时，通过WORKDIR可以指定容器的工作目录， 容器启动时执行的命令会在该目录下执行。更多信息，请参见WORKDIR。创建BCI实例时，通过配置BCI实例中容器的工作目录(WorkingDir)，可以覆盖WORKDIR。

> **注意：**
>
> 如果镜像里未指定WORKDIR，且创建BCI实例也未配置工作目录，则工作目录默认为根目录。 如果指定的工作目录不存在，系统将自动创建。

- 启动命令和参数镜像构建时，通过ENTRYPOINT和CMD可以指定启动容器后要执行的命令和参数。更多信息，请参见ENTRYPOINT和CMD。创建BCI实例时，通过配置BCI实例中容器的启动命令（Command）和参数（Arg），可以覆盖ENTRYPOINT和CMD。具体生效规则如下：

镜像ENTRYPOINT	镜像CMD	容器command	容器Arg	最终执行命令	说明
ls	/root/data	未设置	未设置	ls /root/data	容器的Command和Arg均未设置，则执行镜像ENTRYPOINT和CMD
ls	/root/data	cd	未设置	cd	容器设置了Command，未设置Arg，则只执行Command，忽略镜像ENTRYPOINT和CMD
ls	/root/data	未设置	/home/work	ls /home/work	容器设置了Arg，未设置Command，则执行镜像ENTRYPOINT和容器Arg
ls	/root/data	cd	/home/work	cd /home/work	同时设置了Command和Arg，则执行容器Command和Arg

> **注意：**
>
> 启动命令必须为容器镜像支持的命令，否则会导致容器启动失败。

配置示例

您可以通过容器的workingDir、command和args字段来设置工作目录、启动命令和参数。更多信息，请参见为容器设置启动时要执行的命令和参数。

配置示例如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
  labels:
    app: test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx-test
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: hub.baidubce.com/cce/nginx-alpine-go
        workingDir: /work
        command: ["printenv"]
        args: ["HOSTNAME", "KUBERNETES_PORT"]
        ports:
        - containerPort: 80
```

使用探针对容器进行健康检查

使用探针对容器进行健康检查

Kubernetes中，容器的健康检查由kubelet定期执行，kubelet通过存活探针和业务探针来检查容器的状态和运行情况。当前BCI支持探针如下：

探针	说明	使用场景
应用存活探针（Liveness Probe）	用于检查容器是否正常运行。如果检查成功，则表示容器正常运行。如果检查失败，系统会根据配置的容器重启策略进行相应的处理。如果未配置该探针，则默认容器一直正常运行。	* 当应用程序处于运行状态但无法进行进一步操作时，Liveness Probe将捕获到deadlock，重启对应的容器，使得应用程序在存在bug的情况下依然能够运行。另外，长时间运行的应用程序最终可能会转换到broken状态，此时除了重新启动，无法恢复。Liveness Probe可以检测并补救这种情况。
应用业务探针（Readiness Probe）	用于检查容器是否已经就绪，可以为请求提供服务。如果检查成功，则表示容器已经准备就绪，可以接收业务请求。如果检查失败，则表示容器没有准备就绪，系统将停止向该容器发送任何请求，直至重新检查成功。	如果应用程序暂时无法对外部流量提供服务，例如应用程序需要在启动期间加载大量数据或配置文件，此时，如果不想终止应用程序，也不想向其发送请求，可以通过Readiness Probe来检测和缓解这种情况。
应用启动探针(StartupProbe Probe)	用于检查容器是否启动成功，有时候，会有一些现有的应用在启动时需要较长的初始化时间。 要这种情况下，若要不影响对死锁作出快速响应的探测，设置存活探测参数是要技巧的。	比如服务A启动时间很慢，需要60s。这个时候如果还是用存活探针就会进入死循环，因为当存活探针开始探测时，服务并没有起来，发现探测失败就会触发restartPolicy。此时简单调大存活探针的initialDelay时间不灵活且可能带来更多问题，因此可以通过应用启动探针来环境这种情况。

配置示例

您可以通过容器的livenessProbe和readinessProbe字段来设置Liveness Probe或者Readiness Probe，配置详情见配置存活、就绪和启动探针

1.配置应用存活探针（Liveness Probe）

```
 apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: liveness-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    livenessProbe:
      exec:
        command:
          - /bin/sh
          - '-c'
          - sleep 1 && exit 0
      failureThreshold: 3
      initialDelaySeconds: 5
      periodSeconds: 30
      successThreshold: 1
      timeoutSeconds: 10
```

2.配置应用业务探针（Readiness Probe）

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: readiness-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    readinessProbe:
      exec:
        command:
          - /bin/sh
          - '-c'
          - sleep 1 && exit 0
      failureThreshold: 3
      initialDelaySeconds: 5
      periodSeconds: 30
      successThreshold: 1
      timeoutSeconds: 10
```

3.配置应用启动探针(StartupProbe Probe)

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: startup-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    startupProbe:
      exec:
        command:
          - /bin/sh
          - '-c'
          - sleep 1 && exit 0
      failureThreshold: 3
      initialDelaySeconds: 5
      periodSeconds: 30
      successThreshold: 1
      timeoutSeconds: 10
```

在容器内获取元数据

在容器内获取元数据 当前仅支持在CCE集群中获取元数据

通过 Downward API 访问元数据

Kubernetes Downward API提供了以下两种方式：

- 环境变量（Environment Variables）用于单个变量，可以将Pod信息直接注入容器内部。
- Volume挂载（Volume Files）可以将Pod信息生成为文件，直接挂载到容器内部。 目前BCI已经支持了Downward API的大部分常用字段，下文将为您介绍使用方式。

1. 环境变量方式

您可以通过Downward API将Pod的名称、命名空间、IP等信息注入到容器的环境变量中。通过环境变量可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称
metadata.namespace	Pod命名空间
metadata.uid	Pod的UID
metadata.labels['']	Pod的标签值
metadata.annotations['']	Pod的注解值
spec.serviceAccountName	Pod服务账号名称
spec.nodeName	节点名称
status.podIP	节点IP
limits.cpu	容器的 CPU 限制值
requests.cpu	容器的 CPU 请求值
limits.memory	容器的内存限制值
requests.memory	容器的内存请求值

> **注意：**
>
> 暂不支持的字段如下：
>
> - status.hostIP
> - resource: limits.ephemeral-storage
> - resource: requests.ephemeral-storage

配置示例：

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: env-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
```

```
  tolerations:
  - effect: NoSchedule
    key: virtual-kubelet.io/provider
    operator: Equal
    value: baidu
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
 containers:
 - image: hub.baidubce.com/cce/nginx-alpine-go
  imagePullPolicy: IfNotPresent
  name: c01
  workingDir: /work
  ports:
  - containerPort: 8080
    protocol: TCP
  resources:
    limits:
      cpu: 250m
      memory: 512Mi
    requests:
      cpu: 250m
      memory: 512Mi
  env:
    - name: MY_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: MY_ENV
      value: "test"
    - name: METADATA_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: METADATA_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: METADATA_UID
      valueFrom:
        fieldRef:
          fieldPath: metadata.uid
    - name: METADATA_LABELS
      valueFrom:
        fieldRef:
          fieldPath: metadata.labels['mylabel']
    - name: METADATA_ANNOTATIONS_REGION
      valueFrom:
        fieldRef:
          fieldPath: metadata.annotations['myannotation']
    - name: MY_CPU_LIMIT
      valueFrom:
        resourceFieldRef:
          containerName: c01
          resource: limits.cpu
    - name: MY_CPU_REQUEST
      valueFrom:
        resourceFieldRef:
```

```
            containerName: c01
            resource: requests.cpu
      - name: MY_MEM_LIMIT
        valueFrom:
          resourceFieldRef:
            containerName: c01
            resource: limits.memory
      - name: MY_MEM_REQUEST
        valueFrom:
          resourceFieldRef:
            containerName: c01
            resource: requests.memory
```

2. Volume挂载方式

您可以通过Downward API将Pod的Label、Annotation等信息通过Volume挂载到容器的某个文件中。通过Volume挂载可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称
metadata.namespace	Pod命名空间
metadata.uid	Pod的UID
metadata.labels['']	Pod的标签值
metadata.annotations['']	Pod的注解值
metadata.labels	Pod的所有标签
metadata.annotations	Pod的所有注解
limits.cpu	容器的 CPU 限制值
requests.cpu	容器的 CPU 请求值
limits.memory	容器的内存限制值
requests.memory	容器的内存请求值

配置示例：

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: volume-test
  namespace: default
spec:
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
```

```
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
containers:
- image: hub.baidubce.com/cce/nginx-alpine-go
  imagePullPolicy: IfNotPresent
  name: c01
  workingDir: /work
  ports:
  - containerPort: 8080
    protocol: TCP
  resources:
    limits:
      cpu: 250m
      memory: 512Mi
    requests:
      cpu: 250m
      memory: 512Mi
  volumeMounts:
    - name: podinfo
      mountPath: /etc/podinfo
volumes:
- name: podinfo
  downwardAPI:
    items:
      - path: "metadata.name"
        fieldRef:
          fieldPath: metadata.name
      - path: "metadata.namespace"
        fieldRef:
          fieldPath: metadata.namespace
      - path: "metadata.uid"
        fieldRef:
          fieldPath: metadata.uid
      - path: "mylabel"
        fieldRef:
          fieldPath: metadata.labels['mylabel']
      - path: "myannotation"
        fieldRef:
          fieldPath: metadata.annotations['myannotation']
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "annotations"
        fieldRef:
          fieldPath: metadata.annotations
      - path: "workload_cpu_limit"
        resourceFieldRef:
          containerName: c01
          resource: limits.cpu
          divisor: 1m
      - path: "workload_cpu_request"
        resourceFieldRef:
          containerName: c01
          resource: requests.cpu
          divisor: 1m
      - path: "workload_mem_limit"
        resourceFieldRef:
          containerName: c01
```

```
      containerName: c01
      resource: limits.memory
      divisor: 1Mi
  - path: "workload_mem_request"
    resourceFieldRef:
      containerName: c01
      resource: requests.memory
      divisor: 1Mi
```

容器生命周期回调

容器生命周期回调

本文介绍如何使用BCI容器生命周期回调框架，即preStop和postStart函数触发业务自定义代码逻辑执行，详情请见container-lifecycle-hooks

> **注意：**
>
> BCI目前只支持exec形式的hook，httpGet和tcpSocket类型的hook会被忽略。

postStart

这个回调在容器被创建之后立即被执行。 但是不能保证回调会在容器入口点（ENTRYPOINT）之前执行。 没有参数传递给处理程序。

pod示例如下：

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: poststart-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh","-c","echo postStart hook > /tmp/termination-log"]
```

preStop

在容器因API请求或者管理事件（诸如存活态探针、启动探针失败、资源抢占、资源竞争等） 而被终止之前，此回调会被调用。如果容器已经处于已终止或者已完成状态，则对preStop回调的调用将失败。在用来停止容器的TERM信号被发出之前，回调必须执行结束。Pod的终止宽限周期在PreStop回调被执行之前即开始计数，所以无论回调函数的执行结果如何，容器最终都会在Pod的终止宽限期内被终止。没有参数会被传递给处理程序。

pod示例如下：

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: prestop-test
  namespace: default
spec:
  enableServiceLinks: false
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    lifecycle:
      preStop:
        exec:
          command: ["/bin/sh","-c","echo preStop hook > /tmp/termination-log && sleep 30"]
```

设置容器终止消息

设置容器终止消息

在Kubernetes中，terminationMessagePath和terminationMessagePolicy用于指定容器终止消息的来源路径和策略。本文介绍如何设置BCI容器的terminationMessagePath和terminationMessagePolicy字段，实现自定义设置容器终止消息。

配置说明 Kubernetes通过terminationMessagePath和terminationMessagePolicy管理容器终止消息。

字段	说明
terminationMessagePath	用于设置容器终止的消息来源。即当容器退出时，Kubernetes 从容器的terminationMessagePath字段中指定的终止消息文件中检索终止消息。默认值为 /dev/termination-log。 通过自定义设置terminationMessagePath，可以使得Kubernetes在容器运行成功或失败时，使用指定文件中的内容来填充容器的终止消息。终止消息内容最大为4 KB。
terminationMessagePolicy	用于设置容器终止消息的策略。取值为： • File（默认）：仅从终止消息文件中检索终止消息。 • FallbackToLogsOnError：在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一部分内容来作为终止消息。

> **说明**：
> Pod内所有容器的终止信息大小之和最大为12 KB。当总和超过12 KB时，Kubernetes的状态管理器会对其加以限制。例如：Pod内有4个InitContainer和8个应用Container，则状态管理器会限制每个容器的终止信息最大为1 KB，即截取每个Container终止信息的前1 KB。

操作指南

在以下示例中，配置了terminationMessagePath字段为：/tmp/termination-log，则容器将把终止消息写入/tmp/termination-log给Kubernetes接收。

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: termination-test
  namespace: default
spec:
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    terminationMessagePath: "/tmp/termination-log"
```

此外，您还可以设置容器的terminationMessagePolicy字段，进一步自定义容器终止消息。该字段默认值为：File，即仅从终止消息文件中检索终止消息。您可以根据需要设置为：FallbackToLogsOnError，即在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一部分内容来作为终止消息。

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: volume-test
  namespace: default
spec:
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    terminationMessagePath: "/tmp/termination-log"
    terminationMessagePolicy: "FallbackToLogsOnError"
```

设置容器时区

本文介绍如何为容器配置时区，以此来保证容器中的时间与所处环境的时间一致，避免时区错误导致的时间一致性和准确性等问题。

配置示例

1. 创建一个ConfigMap，导入/usr/share/zoneinfo/目录下您需要的时区

```
kubectl create configmap tz --from-file=/usr/share/zoneinfo/Asia/Shanghai
```

2. 创建配置时区的应用

```
kubectl apply -f timezone-demo.yaml
```

timezone-demo.yaml内容示例如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    name: pod-test
  name: pod-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      name: pod-test
  template:
    metadata:
      creationTimestamp: null
      labels:
        name: pod-test
    spec:
      containers:
      - image: registry.baidubce.com/glen-centos/centos:centos7
        imagePullPolicy: IfNotPresent
        name: pod-test
        resources:
          limits:
            cpu: 250m
            memory: 500Mi
          requests:
            cpu: 250m
            memory: 500Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - name: tz
          mountPath: /etc/localtime   # 挂载路径
          subPath: Shanghai  # 请根据您的ConfigMap替换
      dnsPolicy: Default
      volumes:
      - name: tz
        configMap:   # 挂载ConfigMap
          name: Shanghai
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
      nodeSelector:
        type: virtual-kubelet
```

验证结果

1. 提交上述yaml，登录到容器内，验证时区是否设置成功。命令提交前根据实际Pod名称替换。

```
kubectl exec -it <pod-name> -- sh
```

2. 查询容器时区

```
date -R
```

如果返回的时间与设置的时区信息对应，则表示设置成功。返回示例如下：

```
Thu, 13 Feb 2025 18:00:11 +0800
```

强制终止Sidecar容器并忽略容器退出码

当您采用Sidecar容器的形式实现类似DaemonSet的效果时，可能会出现Job类Pod无法运行完成的情况，此时可以通过设置BCI Pod Annotation指定需要忽略容器退出码的容器，以保证Job可以正常运行完成。

功能说明 在BCI场景下，由于虚拟节点的限制，BCI不支持Kubernetes的DaemonSet功能。此时部分需要使用DaemonSet的场景可以采用为BCI Pod添加Sidecar容器的形式来实现类似效果，但该方式在RestartPolicy配置为OnFailure和Never时，会影响BCI Pod的生命周期。例如：运行Job类任务时，为Job添加Filebeat Sidecar容器后，由于业务容器退出后，Filebeat容器会继续运行，会导致Job始终无法达到终态，无法运行完成。

针对上述场景，BCI支持通过Annotation指定需要忽略状态码的Sidecar容器列表：

- 忽略容器退出码

  当业务容器正常退出后，由于BCI强制终止Sidecar容器的运行时，Sidecar容器的退出码为非0（非0表示容器运行失败终止），会导致Job最终的状态为Failed，此时可以通过设置Annotation的方式，标记Sidecar容器忽略容器退出码，强制将该容器置为运行成功终止状态，保证Job最终的状态为Succeeded。

配置说明 为BCI Pod设置以下Annotation来标记忽略退出码的Sidecar容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-ignore-exit-code-containers: "sidecar1,sidecar2"
```

> 重要
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例

1. 编写Job的YAML配置文件，然后使用该YAML文件创建Job

   ```
   kubectl apply -f test-sidecar-job.yaml
   ```

   test-sidecar-job.yaml的内容示例如下，表示创建一个Job，Job内包含两个容器，c1为业务容器，c2为Sidecar容器，并且添加了Annotation声明忽略Sidecar容器退出码。

```
 apiVersion: batch/v1
kind: Job
metadata:
 labels:
   app: test
 name: test
 namespace: default
spec:
 template:
   metadata:
     annotations:
       bci.virtual-kubelet.io/bci-ignore-exit-code-containers: "c2"
     labels:
       app: test
     name: test
   spec:
     containers:
     - name: c1
       image: registry.baidubce.com/glen-centos/centos:centos6
       command: ["control.sh", "start"]
       imagePullPolicy: Always
       resources:
         limits:
           cpu: 1
           memory: 2Gi
     - name: c2
       image: registry.baidubce.com/glen-centos/centos:centos6
       imagePullPolicy: Always
       command: ["/bin/sh", "-c", "sleep 120"]
       resources:
         limits:
           cpu: 250m
           memory: 512Mi
     nodeSelector:
       type: virtual-kubelet
     restartPolicy: Never
     tolerations:
     - effect: NoSchedule
       key: virtual-kubelet.io/provider
       operator: Equal
       value: baidu
```

2. 查看Job详情和对应的Pod详情，观察效果

   - 确认Job已经运行完成，且状态为Succeeded

     ```
     kubectl describe job <job-name>
     ```

     示例如下：

   - 查看Sidecar容器详情，确认实际的退出码和相关信息

     ```
     kubectl describe pod <pod-name>
     ```

     示例如下：

配置NTP服务

在时间敏感的业务场景，使用BCI部署容器应用时，您可以配置NTP服务来确保Pod内容器的时间同步准确，从而规避因时间不准确导致的问题，保证数据准确性和业务正常运行。 **配置说明** 创建BCI Pod时，可以为Pod添加`bci.virtual-kubelet.io/bci-ntp-server`的Annotation来指定NTP服务器的地址，使得Pod内的容器能与NTP服务进行时间同步，从而保证时间准确性。

> Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
>
> 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。
>
> Annotation `bci.virtual-kubelet.io/bci-ntp-server`对应的值，只支持指定一个ip

配置示例

1. 创建配置NTP服务的应用。

```
kubectl create -f set-ntp.yaml
```

yaml内容示例如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: centos6-vk
  name: centos6-vk
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: centos6-vk
  template:
    metadata:
      annotations:
        bci.virtual-kubelet.io/bci-ntp-server: "10.0.xx.xx" # 指定NTP服务器的IP地址
      labels:
        app: centos6-vk
      name: centos6-vk
    spec:
      containers:
      - command:
        - /bin/sh
        - -c
        - sleep 3600000
        image: registry.baidubce.com/glen-centos/centos:centos6
        imagePullPolicy: Always
        name: centos-test
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
      nodeSelector:
        type: virtual-kubelet
      restartPolicy: Always
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

2. 时间同步以sidecar的形式集成在了提交的pod内，该sidecar对用户不可见，需要登录pod所在的机器查看

```
kubectl exec -it ${bci-podname} -c ntp-sidecar -- sh
```

3. 查询容器的时间来源

```
sh-4.2# chronyc sources
```

输出结果：

```
210 Number of sources = 1
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* 10.0.xx.xx                10   6    37    38   +517ns[ +221us] +/-  205us
```

忽略Sidecar容器的NotReady状态

当您采用Sidecar容器的形式实现类似DaemonSet的效果时，如果Sidecar容器的状态为NotReady，会导致Pod状态为NotReady。某些场景下，如果您不希望Sidecar容器的状态影响整个Pod状态，可以通过设置BCI Pod Annotation，设置需要忽略的Sidecar容器的NotReady状态的容器列表，保证Pod状态不受Sidecar容器状态的影响。

功能说明 在BCI场景下，由于虚拟节点的限制，BCI不支持Kubernetes的DaemonSet功能。此时部分需要使用DaemonSet的场景可以采用为BCI Pod添加Sidecar容器的形式来实现类似效果。但添加Sidecar容器后，如果Sidecar容器的状态为NotReady，会导致Pod状态为NotReady。在某些场景下，您可能会不希望Sidecar容器状态影响整个Pod状态，例如：使用Sidecar容器用于收集日志，但日志容器出现问题，不应该影响业务容器对外提供服务。

针对上述场景，BCI支持了忽略容器NotReady状态的功能。如果您不希望某一容器的状态影响整个Pod状态，可以为其添加忽略容器状态的环境变量。添加后，当该容器出现NotReady状态时，也不会影响Pod进入Ready状态。

配置说明 为BCI Pod设置以下Annotation来标记忽略退出码的Sidecar容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-ignore-not-ready-containers: "sidecar1,sidecar2"
```

> 重要
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例

1. 编写Deployment的YAML配置文件，然后使用该YAML文件创建deployment

   ```
   kubectl apply -f test-sidecar-ignore.yaml
   ```

   test-sidecar-ignore.yaml的内容示例如下，表示创建一个deployment, 内包含两个容器，c1为业务容器，c2为Sidecar容器，并且添加了Annotation声明忽略Sidecar容器NotReady的状态。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-ignore
  labels:
    app: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
      annotations:
        bci.virtual-kubelet.io/bci-ignore-not-ready-containers: "c2"
    spec:
      containers:
      - name: c1
        image: registry.baidubce.com/glen-centos/centos:centos6
        command: ["/bin/sh", "-c", "sleep 999"]
      - name: c2
        image: registry.baidubce.com/glen-centos/centos:centos6
      - name: c3
        image: registry.baidubce.com/glen-centos/centos:centos6
      nodeSelector:
        type: virtual-kubelet
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

2. 查看Deployment对应的Pod详情，查看Pod的Condition观察效果，可见ContainersReady和Ready类型的condition的状态均为True。

   ```bash
   kubectl get pod <pod-name> -o yaml
   ```

conditon示例如下：

```
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:52Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:14:00Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:14:00Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:51Z"
    status: "True"
    type: PodScheduled
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:43Z"
    status: "True"
    type: Creating
```

日志

BLS日志采集使用方式

BCI容器组支持采集业务容器内的日志文件、标准输出，并推送到BLS(百度云日志服务)。

BCI日志采集架构

上图是BCI日志采集的架构图，如需采集日志，需要创建三类关键对象

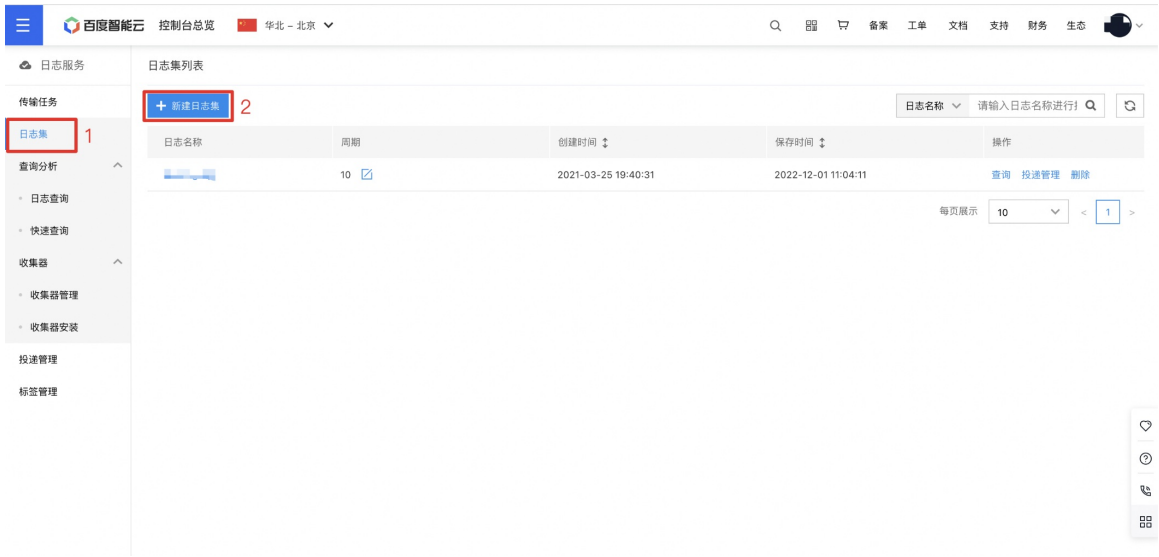
- 存储端：存放用户日志的存储产品，由**用户**负责创建。目前日志采集支持的存储产品有：BLS日志集、百度云Kafka、BOS、BES。
- 采集端：采集物理机/容器内日志文件的Agent组件，一般一个物理机/容器组内部署一个，由**BCI**负责创建。
- 传输任务：实际满足用户的日志采集需求，关键参数包含：**采集任务名称、收集器列表、日志文件路径、日志存储端**。传输任务可以由**用户**创建，也可以由**BCI**创建。 **使用流程 创建存储产品（存储端）** 注：如果已有存储产品，可跳过此步骤。
  用户可选择使用如下任一种云产品作为存储端，详细使用方案如下
- BLS日志集使用文档
- 百度云Kafka使用文档

- BOS使用文档

以 BLS 日志集为例，我们的创建流程如下

1. 登录百度智能云官网，点击右上角的“管理控制台”，快速进入控制台界面。

2. 选择“产品服务>日志服务BLS”，进入『日志集』页面

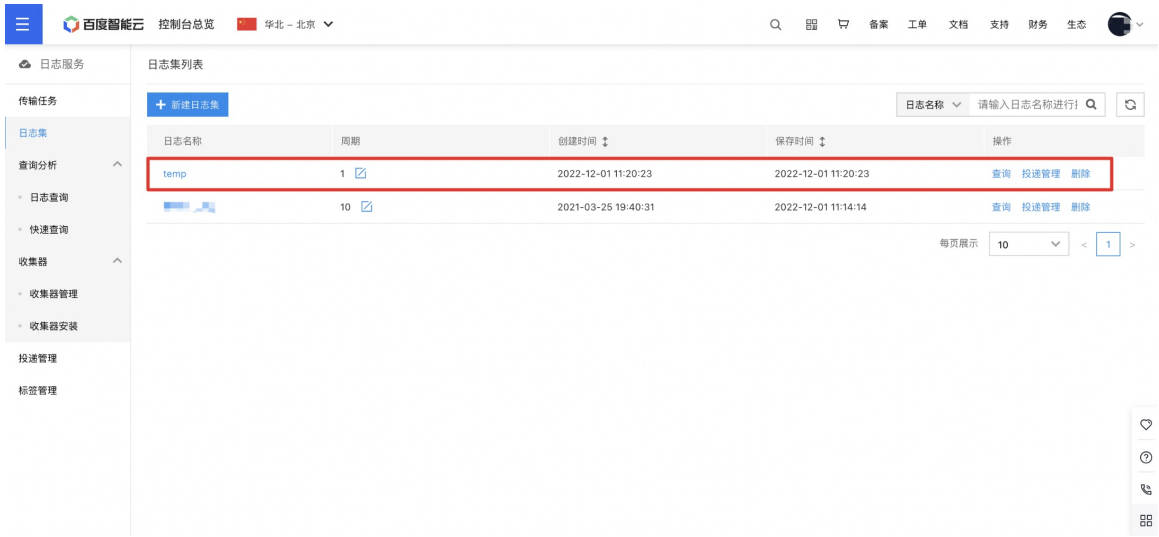


3. 点击『新建日志集』，弹出新建日志集页面，填写配置信息，包括：

   a. 名称：设置日志集名称，1-128位字符、数字、英文和符号，符号仅限：_-.。名称创建后不支持修改，并且同一region下名称具有唯一性。

   b. 存储周期：支持1～180天范围的存储周期。如需更大存储周期，请提交 BLS 工单。

4. 日志集名称为『temp』，完成配置后点击『确定』，完成日志集的创建。



BCI创建传输任务 目前支持通过vk方式支持使用BLS日志采集。支持复用已经存在的BLS任务，复用时当前填写的任务配置需和BLS远端配置一致。 **VK使用** 通过vk使用bls日志采集需要在业务容器中设置日志采集任务参数为环境变量，并设置volumeMounts和volume参数。 **1.容器内设置环境变量：** {key} 表示日志采集类型，internal表示采集容器内的日志文件，stdout表示采集容器的标准输出，不支持填写其他内容。

{index}用于区分同一个日志采集任务下的配置，取值为正整数，同一pod内{index}不可重复。

一个容器内支持多个internal类型的bls任务采集任务，仅支持一个stdout类型的bls日志采集任务。

环境变量key名	是否必选	类型	含义	输入限制
bls_task_{key}_{index}_name	是	string	日志采集任务名（支持复用已经存在的bls任务名）	不能包含空格，长度为1-64字符；同一用户名空间下不能重复。
bls_task_{key}_{index}_logStore	是	string	日志集(需用户手动创建)	1-128位字符、数字、英文和符号，符号仅限：_-.
bls_task_{key}_{index}_ttl	是	int	日志采集有效文件时间 单位：天	正整数
bls_task_{key}_{index}_rateLimit	是	int	日志投递速率	单位：MB/s， 速率范围在1-100之间
bls_task_{key}_{index}_matchedPattern	是	string	日志匹配表达式，符合规则的日志，将被采集	无
bls_task_{key}_index_srcDir	是	string	日志采集源目录	容器输出填写stdout，容器内日志填写目录

2.容器内设置volumeMounts和volume 每添加一个bls日志采集任务，需要在yaml文件中添加对应的volumeMounts参数和volume参数。

- **internal 类型**
  volumeMounts参数：name应与环境变量bls_task_internal_{index}_name对应的value一致，mountPath应该与日志采集任务对应的srdDir路径一致。

```
volumeMounts:
 - mountPath: /var/log
   name: emptydir1
```

volume参数：name应与环境变量bls_task_internal_{index}_name对应的value一致，类型为： emptyDir。

```
volumes:
 - name: emptydir1
   emptyDir: {}
```

- **stdout类型**

volumeMounts参数：

```
volumeMounts:
- mountPath: /stdout
  name: sidecar-stdout
```

volume参数：

```
 volumes:
- name: sidecar-stdout
  flexVolume:
    driver: k8s/sidecar-stdout
```

注意：多个标准输出日志存储只需要指定一次sidecar-stdout volume，未指定stdout日志采集，则无需指定该volume。

- **使用示例**

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

```
metadata:
  labels:
    app: test-bls-log
  name: test-bls-log
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-bls-log
  template:
    metadata:
      labels:
        app: test-bls-log
    spec:
      containers:
        - name: nginx
          image: registry.baidubce.com/qa-test/wynginx:7.0.0
          imagePullPolicy: Always
          env:
            - name: bls_task_internal_index_name       # 注：index用来区分不同的采集任务，如
bls_task_internal_1_name,bls_task_stdout_2_name。
              value: {{bls_task_internal_index_name}}
            - name: bls_task_internal_index_logStore
              value: {{bls_task_internal_index_logStore}}
            - name: bls_task_internal_index_ttl
              value: {{bls_task_internal_index_ttl}}
            - name: bls_task_internal_index_rateLimit
              value: {{bls_task_internal_index_rateLimit}}
            - name: bls_task_internal_index_matchedPattern
              value: {{bls_task_internal_index_matchedPattern}}
            - name: bls_task_internal_index_srcDir
              value: {{bls_task_internal_index_srcDir}}
            - name: bls_task_stdout_index_name
              value: {{bls_task_stdout_index_name}}
            - name: bls_task_stdout_index_logStore
              value: {{bls_task_stdout_index_logStore}}
            - name: bls_task_stdout_index_ttl
              value: {{bls_task_stdout_index_ttl}}
            - name: bls_task_stdout_index_rateLimit
              value: {{bls_task_stdout_index_rateLimit}}
            - name: bls_task_stdout_index_matchedPattern
              value: {{bls_task_stdout_index_matchedPattern}}
            - name: bls_task_stdout_index_srcDir
              value: {{bls_task_stdout_index_srcDir}} # 标准输出填写stdout即可
          volumeMounts:
            - mountPath: {{bls_task_internal_index_srcDir}}
              name: {{emptyDirName}} # name需与volume name一致
            - mountPath: /stdout  # 标准输出的挂载路径和名称不可修改
              name: sidecar-stdout
      volumes:
        - name: {{emptyDirName}}
          emptyDir: {}
        - name: sidecar-stdout  # 有标准输出的采集任务需要填写，否则不要填写。
          flexVolume:
            driver: k8s/sidecar-stdout
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
        - key: "virtual-kubelet.io/provider"
          operator: "Equal"
          value: "baidu"
          effect: "NoSchedule"
```

监控

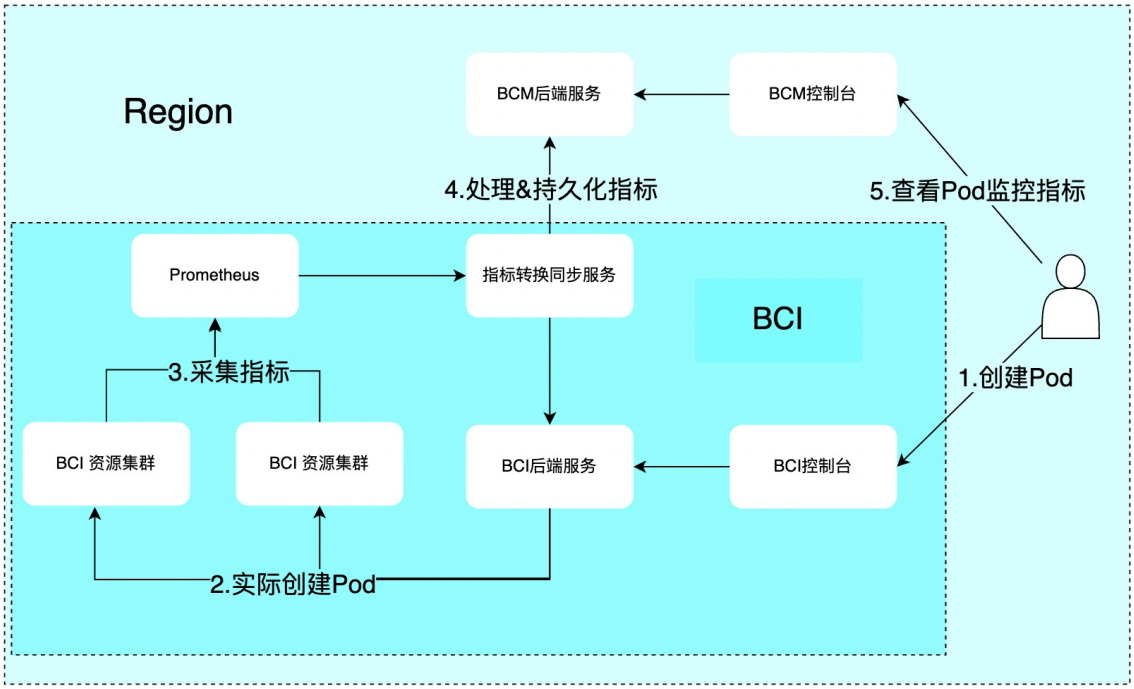
查看实例监控指标

BCI支持自动采集实例的CPU、内存、网络和磁盘等相关监控指标，并可通过BCM（百度云监控）进行查看。

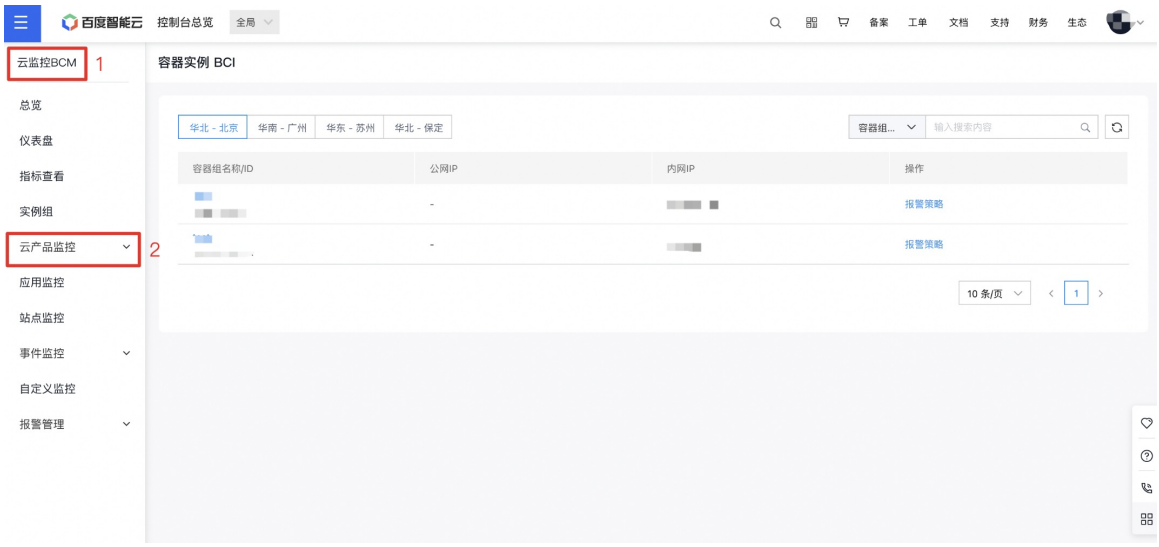
监控指标概述 BCI支持查看的监控指标如下： **CPU**

- CPU使用率：指标采集周期内，平均CPU使用率（单位：%）。CPU处于非IDLE状态即使用，CPU使用率上限为申请的CPU核数*100%。例如：如果申请了1.5 vCPU，且在指标采集窗口期内有50%的时间处于使用状态，则CPU使用率为75%。 **内存**
- 内存用量：指标采集时，正在使用的内存字节数（单位：字节）。使用的内存字节数不包含Page Cache占用的内存空间。 **网络**
- 网络接收量：指标采集周期内，每秒平均接收的网络数据比特数（单位：比特/秒）。
- 网络发送量：指标采集周期内，每秒平均发送的网络数据比特数（单位：比特/秒）。
- 网络接收包：指标采集周期内，每秒平均接收的网络IP包个数（单位：个/秒）。
- 网络发送包：指标采集周期内，每秒平均发送的网络IP包个数（单位：个/秒）。 **磁盘**
- 磁盘读取量：指标采集周期内，每秒平均从文件系统读取的字节数（单位：字节/秒）。指标名称中的磁盘，泛指以磁盘为代表的文件系统，从内存文件系统读取的数据量也会算入指标值中。下述磁盘相关指标的命名都有此含义。
- 磁盘写入量：指标采集周期内，每秒平均写到文件系统的字节数（单位：字节/秒）。
- 磁盘读取次数（暂未采集）：指标采集周期内，每秒平均发起文件系统读请求的次数（单位：次/秒）。
- 磁盘写入次数（暂未采集）：指标采集周期内，每秒平均发起文件系统写请求的次数（单位：次/秒）。

实例监控架构



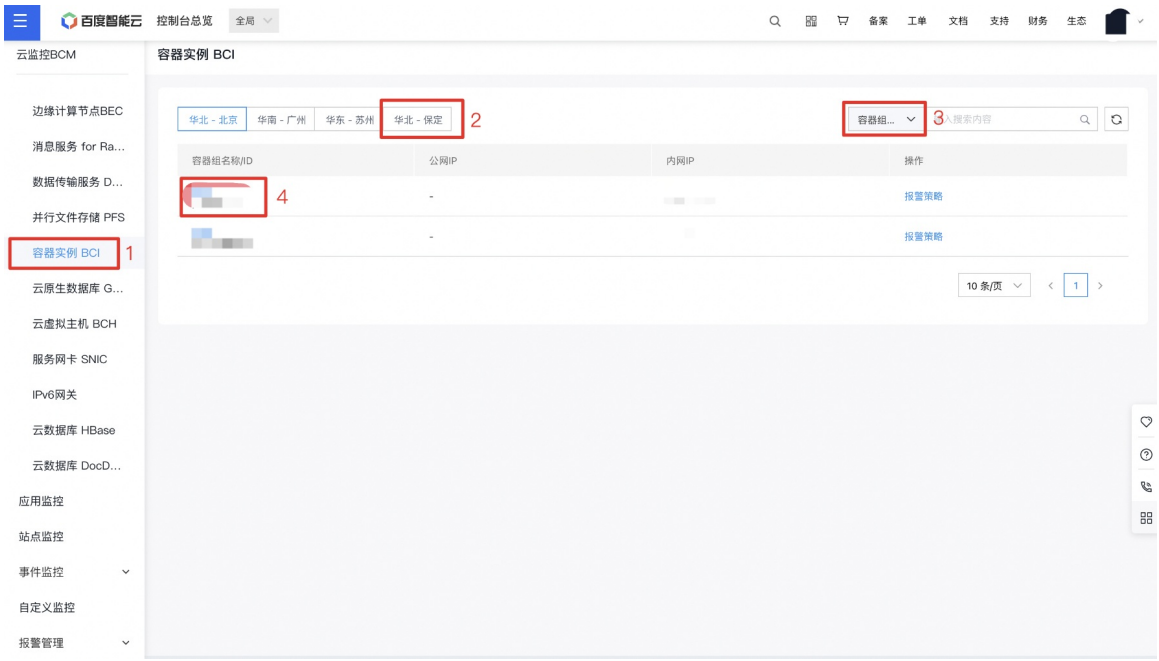
BCM查看监控指标 进入BCM产品页面



1.进入BCM产品页面

2.点击展开左侧『云产品监控』

选择容器实例 BCI



1.展开『云产品监控』后，选择『容器实例 BCI』

2.选择容器实例所在的 Region

3.选择查看容器组粒度的监控指标，或者容器粒度监控指标

4.点击要查看的容器组/容器

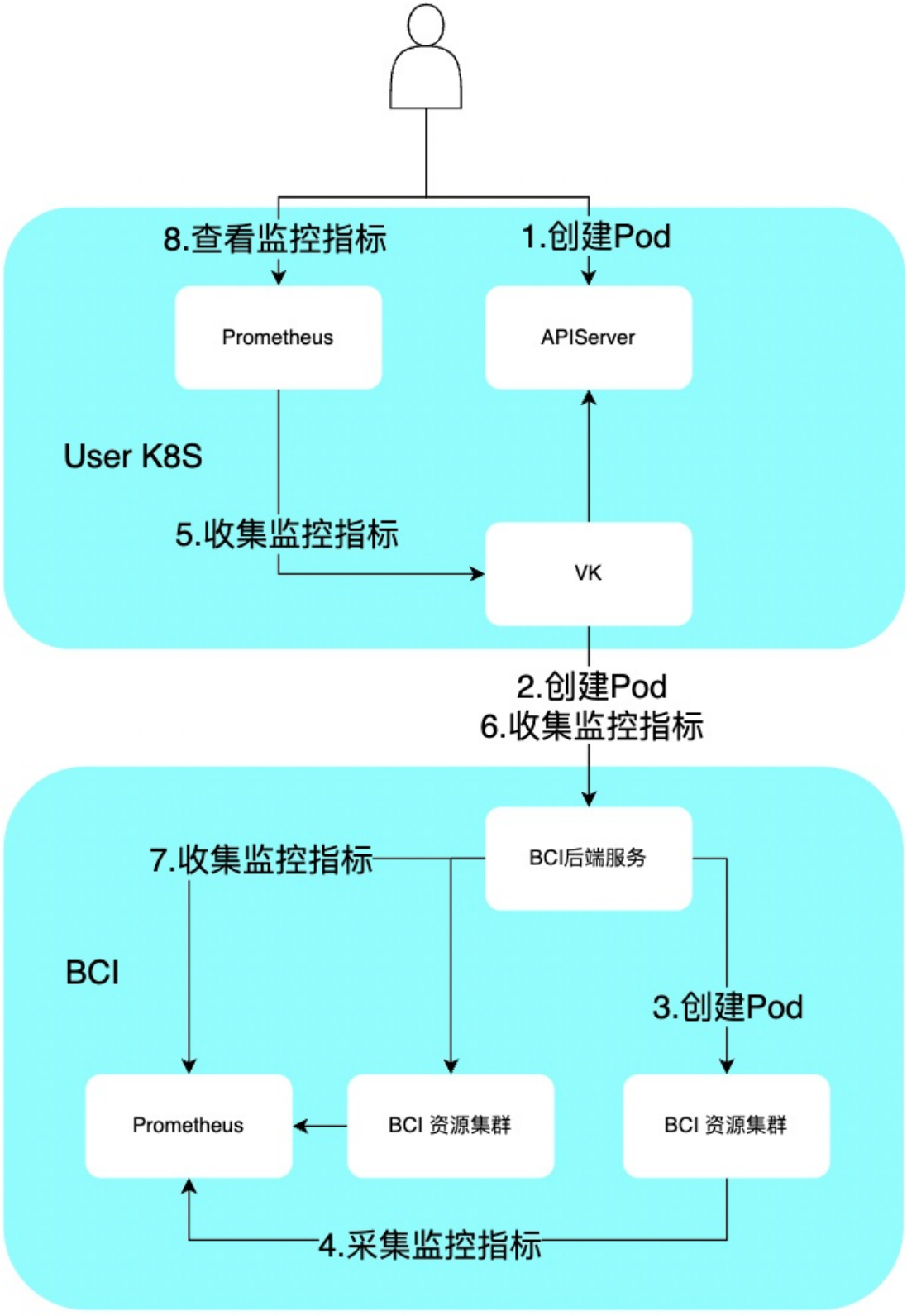
更多信息，请参考 云监控BCM操作指南。

通过VK获取实例监控指标

除了通过百度云监控BCM查看 BCI 实例监控指标外，BCI 还支持通过 VK 查看 BCI 实例监控指标。 通过 VK 查看监控指标的方式与 Kubelet 类似，本文将分三部分介绍：

1. 监控指标来源
2. 支持查看的指标
3. 查看指标的方式

监控指标来源 BCI实例监控指标的流向如下图所示



支持查看的指标 通过VK提供的监控指标是prometheus指标格式，与BCM上看到的指标有较大的差异。vk 提供的监控指标格式如下

```
 ###### 指标的帮助说明
###### 指标名称 指标类型
指标名称{标签，标签，...} 指标取值 指标时间戳
指标名称{标签，标签，...} 指标取值 指标时间戳

###### 指标的帮助说明
###### 指标名称 指标类型
指标名称{标签，标签，...} 指标取值 指标时间戳
指标名称{标签，标签，...} 指标取值 指标时间戳
```

指标分为多类，每一类指标的指标名称和指标类型是都一样的，每一类指标又可分为多条（指标名称一样，标签不一样）。
每条指标都包含指标名称、标签集合、指标取值、指标时间戳四个部分。
如下图示例说明：

支持的指标： CPU

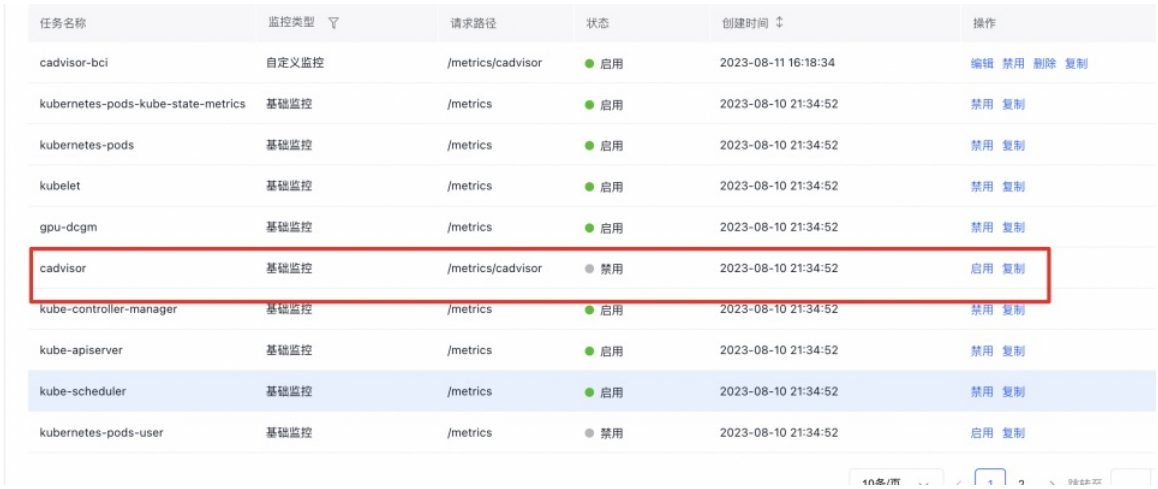
- container_cpu_usage_seconds_total：容器创建以来，累计使用cpu的总时间（单位：秒）。 **内存**
- container_memory_working_set_bytes：容器正在使用的内存字节数（单位：字节）。使用的内存字节数不包含Page Cache占用的内存空间 **磁盘**
- container_fs_reads_bytes_total：容器创建以来，累计从文件系统读取的字节数（单位：字节）。
- container_fs_writes_bytes_total：容器创建以来，累计写入到文件系统的字节数（单位：字节）。 **网络**
- container_network_receive_bytes_total：容器创建以来，累计接收的网络数据字节数（单位：字节）。
- container_network_receive_packets_total：容器创建以来，累计接收的网络数据IP包数（单位：个）。
- container_network_transmit_bytes_total：容器创建以来，累计发送的网络数据字节数（单位：字节）。
- container_network_transmit_packets_total：容器创建以来，累计发送的网络数据IP包数（单位：个）。 **GPU指标** 目前BCI支持的GPU指标如下：

指标名称	类型	描述	示例值
DCGM_FI_DEV_SM_CLOCK	gauge	# HELP DCGM_FI_DEV_SM_CLOCK SM clock frequency (in MHz).	1695
DCGM_FI_DEV_MEM_CLOCK	gauge	# HELP DCGM_FI_DEV_MEM_CLOCK Memory clock frequency (in MHz).	6250
DCGM_FI_DEV_GPU_TEMP	gauge	# HELP DCGM_FI_DEV_GPU_TEMP GPU temperature (in C).	56
DCGM_FI_DEV_POWER_USAGE	gauge	# HELP DCGM_FI_DEV_POWER_USAGE Power draw (in W).	75.738000
DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION	counter	# HELP DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION Total energy consumption since boot (in mJ).	164108202242
		# HELP DCGM_FI_DEV_PCIE_REPLAY_COUNTER	

DCGM_FI_DEV_PCIE_REPLAY_COUNTER	counter	Total number of PCIe retries.	0
DCGM_FI_DEV_GPU_UTIL	gauge	# HELP DCGM_FI_DEV_GPU_UTIL GPU utilization (in %).	0
DCGM_FI_DEV_MEM_COPY_UTIL	gauge	# HELP DCGM_FI_DEV_MEM_COPY_UTIL Memory utilization (in %).	0
DCGM_FI_DEV_ENC_UTIL	gauge	# HELP DCGM_FI_DEV_ENC_UTIL Encoder utilization (in %).	0
DCGM_FI_DEV_DEC_UTIL	gauge	# HELP DCGM_FI_DEV_DEC_UTIL Decoder utilization (in %).	0
DCGM_FI_DEV_XID_ERRORS	gauge	# HELP DCGM_FI_DEV_XID_ERRORS Value of the last XID error encountered.	0
DCGM_FI_DEV_FB_FREE	gauge	# HELP DCGM_FI_DEV_FB_FREE Framebuffer memory free (in MiB).	5774
DCGM_FI_DEV_FB_USED	gauge	# HELP DCGM_FI_DEV_FB_USED Framebuffer memory used (in MiB).	16957
DCGM_FI_DEV_NVLINK_BANDWIDTH_TOTAL	counter	# HELP DCGM_FI_DEV_NVLINK_BANDWIDTH_TOTAL Total number of NVLink bandwidth counters for all lanes	0
DCGM_FI_DEV_VGPU_LICENSE_STATUS	gauge	# HELP DCGM_FI_DEV_VGPU_LICENSE_STATUS vGPU License status	0
DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS	counter	# HELP DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS Number of remapped rows for uncorrectable errors	0
DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS	counter	# HELP DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS Number of remapped rows for correctable errors	0
DCGM_FI_DEV_ROW_REMAP_FAILURE	gauge	# HELP DCGM_FI_DEV_ROW_REMAP_FAILURE Whether remapping of rows has failed	0
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE	gauge	# HELP DCGM_FI_PROF_PIPE_TENSOR_ACTIVE Ratio of cycles the tensor (HMMA) pipe is active (in %).	0.037385
DCGM_FI_PROF_GR_ENGINE_ACTIVE	gauge	# HELP DCGM_FI_PROF_GR_ENGINE_ACTIVE Ratio of time the graphics engine is active (in %).	0.130759
DCGM_FI_PROF_SM_ACTIVE	gauge	# HELP DCGM_FI_PROF_SM_ACTIVE The ratio of cycles an SM has at least 1 warp assigned (in %).	0.110320
DCGM_FI_PROF_SM_OCCUPANCY	gauge	# HELP DCGM_FI_PROF_SM_OCCUPANCY The ratio of number of warps resident on an SM (in %).	0.045643
DCGM_FI_PROF_DRAM_ACTIVE	gauge	# HELP DCGM_FI_PROF_DRAM_ACTIVE Ratio of cycles the device memory interface is active sending or receiving data (in %).	0.065867
DCGM_FI_PROF_PCIE_TX_BYTES	counter	# HELP DCGM_FI_PROF_PCIE_TX_BYTES The number of bytes of active pcie tx data including both header and payload.	64522980
		# HELP DCGM_FI_PROF_PCIE_RX_BYTES The	

DCGM_FI_PROF_PCIE_RX_BYTES	counter	number of bytes of active pcie rx data including both header and payload.	8436715

如您使用CCE的prometheus 监控，在VK的cadvisor接口访问获取指标，需要将CCE原始的cadvisor指标采集任务关闭，因为原始的cadvisor指标只采集的免费的指标；



将cadvisor指标拷贝一份，添加想要采集的DCGM指标，配置如下:

```
 job_name: 'cadvisor-gpu'
scheme: https
tls_config:
  ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  insecure_skip_verify: true
bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
metrics_path: /metrics/cadvisor
kubernetes_sd_configs:
  - role: node
relabel_configs:
  - action: labelmap
    regex: __meta_kubernetes_node_label_(.+)
  - source_labels: [ instance ]
    action: replace
    target_label: node
metric_relabel_configs:
  - source_labels: [__name__]
    action: keep
    regex: (machine_(cpu_cores|memory_bytes)|container_(cpu_usage_seconds_total|fs_limit_bytes|fs_reads_bytes_total|
       fs_usage_bytes|fs_writes_bytes_total|memory_working_set_bytes|network_receive_bytes_total|
       network_receive_packets_dropped_total|network_receive_packets_total|network_transmit_bytes_total|

network_transmit_packets_dropped_total|network_transmit_packets_total)|DCGM_(FI_DEV_GPU_UTIL|FI_DEV_GPU_TEMP|FI_D
)
```

支持的标签：

- container：容器名称
- id：容器复杂ID
- image：容器镜像地址
- name：容器简单ID
- namespace：Pod所在的名字空间
- pod：容器所在的Pod

- device：指标所属的块设备，此标签只在 container_fs_xxx 等几个指标中存在
- interface：指标所属的网卡设备，此标签只在 container_network_xxx 等几个指标中存在

> **注意**：使用了镜像加速功能的容器，其监控指标中的 image 并非用户原始镜像，而是 {ACCELERATE_PREFIX}/transfer/{ACCOUNT_ID}/{USER_ORIGINAL_IMAGE}_accelerate {ACCELERATE_PREFIX} 镜像加速添加的前缀 {ACCOUNT_ID} 用户的账户ID {USER_ORIGINAL_IMAGE} 用户原始镜像地址

查看指标的方式 通过 VK 查看 BCI 监控指标 通过 VK 查看 BCI 监控指标的方式，与通过 Kubelet 查看 cAdvisor 指标的方式一样，通过如下接口查看

```
 Host:  {KubeletServer}
Port:  10250
URI:   GET /metrics/cadvisor
Param：无
```

通过 vk 查看BCI实例监控指标，示例如下

```
 ./kubectl --kubeconfig={kubeconfig_path} get --raw "/api/v1/nodes/{vk_bci_kubelet_name}/proxy/metrics/cadvisor"
// {vk_bci_kubelet_name} vk 的 nodeName
```

通过 Prometheus 采集并查看 BCI 监控指标 通过 Prometheus 采集存储 BCI实例监控指标，简单配置（prometheus.yml）如下

```
  global:
   scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
   evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
   # scrape_timeout is set to the global default (10s).

 ###### Alertmanager configuration
 alerting:
   alertmanagers:
     - static_configs:
         - targets:
           # - alertmanager:9093

 ###### Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
 rule_files:
   # - "first_rules.yml"
   # - "second_rules.yml"

 ###### 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: "prometheus"
     static_configs:
       - targets: ["localhost:9090"]
   - job_name: "cadvisor"
     static_configs:
     - targets: ['47.92.228.38:10250']  // 此处 47.92.228.38 是 vk 因特网可访问的 ip，10250 是 vk 的 https server 端口
   (注意不是 http)
     scheme: https
     metrics_path: /metrics/cadvisor
     tls_config:
       insecure_skip_verify: true    // 作为一个demo示例，关闭 tls 协议
     authorization:
       credentials_file: /home/work/prometheus/token  // 可以正常访问 node server 的 bearer token
     relabel_configs:
       - action: labelmap
         regex: __meta_kubernetes_node_label_(.+)
```

- targets：vk 的 ip:port，注意该 ip 是需要能够被 prometheus 访问的，port 是 kubelet 对外暴露 /metrics/cadvisor 接口的端口（一般是10250）
- credentials_file：有访问 node 权限的 service account 的 token；
  启动命令

```
./prometheus --config.file="prometheus.yml"
```

运维

使用coredump分析实例程序异常

功能概述

coredump是指在程序运行过程中发生异常终止或崩溃时，操作系统将程序的内存内容转储到一个特殊的文件（即coredump文件）中，以便于后续的调试和分析。BCI容器实例支持例开启coredump运维任务，以便在容器异常终止时可以查看分析coredump生成的文件，从而定位问题原因，修复程序异常。 BCI支持将coredump文件存入CFS中或者对象存储BOS中。

使用流程

BCI默认关闭coredump，避免磁盘占用过多而导致业务不可用。您可以根据需要选择以下方式开启coredump运维任务。

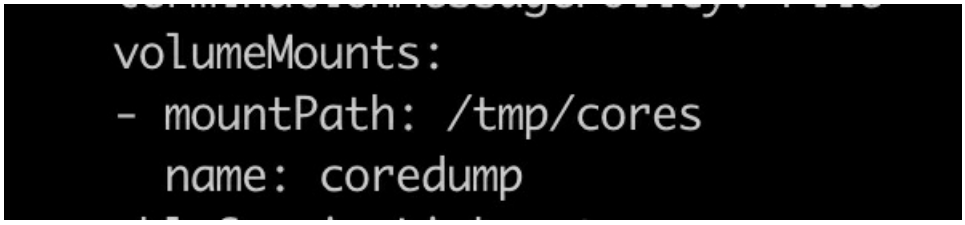
方式一：支持指定Pod Annotation创建Pod开启coredump运维任务 此种形式开启的coredump运维任务在容器运行异常终止或者退出时，触发coredump生成的core文件，该core文件保存在容器/tmp/cores目录中，该目录实际由同地域CFS挂载。

方式二：支持手动开启coredump运维任务 手动通过OpenAPI开启coredump后，BCI将生成一个coredump运维任务,在容器运行异常终止或者退出时，触发coredump生成的core文件将自动保存到对象存储BOS中。

指定Pod Annotation创建Pod开启coredump运维任务 操作步骤如下：

1. 创建BCI 容器实例， 需要注意如下两点：

- 指定并添加如下Pod CorePattern Annotation。需要保证完全一致 bci.virtual-kubelet.io/core-pattern: /tmp/cores
- Pod的远程存储需挂载在容器内/tmp/cores路径，示例如下。



示例example如下：

```
  apiVersion: v1
kind: PersistentVolume
metadata:
  name: coredump-pv-cfs
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  mountOptions:
  - nfsvers=4.2
  nfs:
    path: /  #这里指定远程存储cfs内部路径
    server: cfs-xxxxxxxxx.cfs.gz.baidubce.com #这里指定远程存储cfs地址
  persistentVolumeReclaimPolicy: Retain
  volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: coredump-pvc-cfs
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeMode: Filesystem
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: core-dump-deployment
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 2
```

```
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: core-dump-deployment
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      annotations:
        bci.virtual-kubelet.io/core-pattern: /tmp/cores ## 必须指定这个annotation, 必须完全一致不能更改。
        myannotation: "myannotation"
      labels:
        app: core-dump-deployment
        mylabel: "mylabel"
    spec:
      containers:
      - image: hub.baidubce.com/cce/nginx-alpine-go
        imagePullPolicy: IfNotPresent
        name: c01
        workingDir: /work
        ports:
        - containerPort: 8080
          protocol: TCP
        resources:
          limits:
            cpu: 0.25
            memory: 512Mi
          requests:
            cpu: 0.25
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /tmp/cores  ##这里需要将远程存储挂载到容器内/tmp/cores路径，该路径不支持自定义
          name: coredump
      enableServiceLinks: true
      nodeName: bci-virtual-kubelet-0
      nodeSelector:
        type: virtual-kubelet
      preemptionPolicy: PreemptLowerPriority
      priority: 0
      dnsPolicy: ClusterFirst
      enableServiceLinks: false
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      serviceAccount: default
      serviceAccountName: default
      terminationGracePeriodSeconds: 30
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
      - effect: NoExecute
        key: node.kubernetes.io/not-ready
        operator: Exists
        tolerationSeconds: 300
      - effect: NoExecute
```

```
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    volumes:
    - name: coredump
      persistentVolumeClaim:
        claimName: coredump-pvc-cfs
```

2. 触发coredump。连接BCI实例，在容器内执行sleep 100命令后按Ctrl+\键，可触发coredump，生成的core文件将自动保存到CFS中。

3. 查看并下载core文件。可登陆CFS相关路径获取core文件

手动开启coredump运维任务 当前BCI仅支持通过OpenAPI开启运维任务。步骤如下

1. 确保BCI实例成功创建并处于Running状态，获取到podID

2. 为BCI实例开启coredump运维任务。调用OpenAPI创建coredump运维任务，接口需指定目标BCI实例，然后将OpsType设为coredump，OpsValue设为enable，设置保存coredump文件的BOS bucket地址，即可开启coredump运维任务。一个注意事项是指定BCI实例时，需确保目标BCI实例在创建时没有设置CorePattern Annotation。

```
 POST http://{{endpoint_bci}}/v2/opstask

示例request body:
{
  "podId":"pod-xxxx",
  "opsType":"coredump",
  "opsValue":"enable",
  "bucket":"xxxbucket"
}
```

3. 触发coredump。连接BCI实例，在容器内执行sleep 100命令后按Ctrl+\键，可触发coredump，生成的core文件将自动保存到BOS中。

4. 下载Core文件。调用OpenAPI查询运维任务接口，指定BCI实例podId和运维任务类型OpsType为coredump查询参数，查看运维任务的结果，从返回信息的result中，可以获取core文件是否已保存到BOS中，可到BOS相关bucket下载core文件。

```
 GET http://{{endpoint_bci}}/v2/opsrecord?podId=xxx&opsType=coredump

示例reponse
{
  "result": [
    {
      "opsType": "coredump",
      "opsStatus": "success",
      "storageType": "bos",
      "~~~~storageContent": "please goto bos page to download file core.p-ysappvy3.sh.8.1713786425 .",
      "createTime": 1713786437000
    }
  ]
}
```

常见问题

创建错误 创建BCI Pod后，Pod状态变为ProviderFailed 该状态代表BCI实例创建失败。 可以通过kubectl describe po <pod名称> --namespace <命名空间>查看具体失败原因。

创建BCI Pod后，实例状态长时间处于Creating 一般为BCI订单异常导致，您可以通过查看Pod annotation中的bci.virtual-

kubelet.io/order-id字段查询到BCI订单ID，再到控制台订单页面查询对应的订单失败原因。

接入方式 BCI接入方式有哪些？ 百度容器引擎CCE、本地自建K8S集群、BCIConsole、OpenAPI和SDK。

Job任务类BCI实例如何收费？ Job、CronJob等任务类容器在运行完成后，BCI实例会进入运行成功（Succeeded）或者运行失败（Failed）状态。此时，BCI实例不论是否删除，都不再进行计费。

规格 为什么指定的cpu/memory和实际分配的不一致？ 因为您指定的cpu和memory不是标准规格，系统将自动按照BCI支持的规格进行规整，BCI标准实例规格请参见指定vCPU和内存创建Pod。

镜像 BCI实例是否支持私有镜像? 支持；支持使用百度云镜像仓库搭建镜像仓库，也支持您自己搭建的镜像仓库，请参见使用第三方镜像仓库

BCI实例是否支持镜像缓存？ 支持，详见镜像缓存

网络 BCI实例是否支持修改子网和安全组 不支持；如果您需要更改子网或安全组，需要重新创建BCI实例。

如何从外网访问BCI实例 如果您的BCI实例需要访问外网，或者被外网访问，您需要为BCI实例绑定EIP，或者为实例所属的VPC绑定NAT网关；请参见连接公网

BCI实例是否支持端口映射 不支持；在同一个VPC网络中，您可以直接通过BCI实例的IP+Port进行访问（默认会将容器的端口开放）

存储 BCI实例是否可以和BCC共享CFS文件存储 支持；为BCI实例和BCC实例挂载同一CFS文件系统，即可实现共享数据，请参见挂载CFS文件存储，详细说明见文档挂载CFS文件存储 **BCI实例是否支持数据持久化** 支持；针对一些在BCI实例运行期间对磁盘有较高的IO需求、同时有大量的临时数据需要存放的业务，建议采用外置的数据卷来外挂存储；当前支持：CFS、PFS和BOS

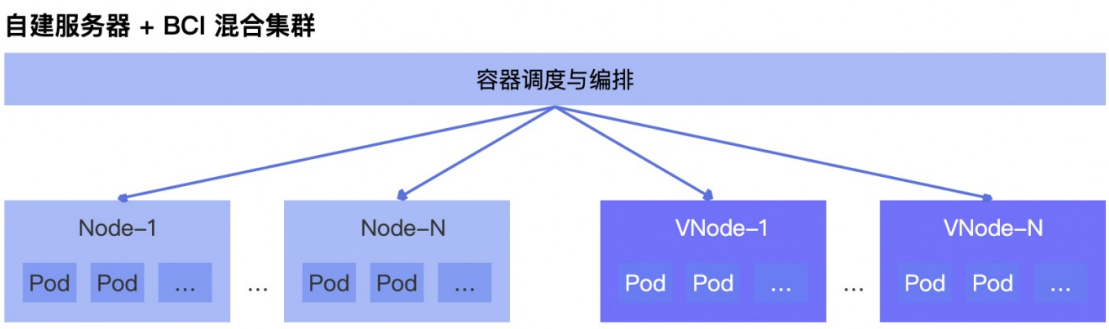
- CFS， 详细说明见文档挂载CFS文件存储
- PFS， 详细说明见文档挂载PFS并行文件存储
- BOS， 详细说明见文档挂载BOS数据卷

通过自建集群使用BCI

对接BCI

对接概述

本文为您介绍自建Kubernetes集群如何与BCI对接，以及如何使用BCI。 BCI能为Kubernetes提供基础的容器Pod运行环境，但业务间的依赖、负载均衡、弹性伸缩、定期调度等能力依然需要Kubernetes来提供。 **对接方式** BCI负责底层Pod容器资源的调度和管理工作，Kubernetes在BCI之上作为PaaS层来管理业务负载，例如管理Deployment、Service、StatefulSet、CronJob等。BCI在接管Pod容器底层基础设施的管理工作后，Kubernetes不再需要直接负责单个Pod的放置、启动等工作，也不再需要关心底层虚拟机的资源情况，通过BCI即可确保Pod需要的资源随时可用。 **使用场景** 对于长时间运行的业务负载，您可以将此类负载的弹性流量部分调度至BCI，缩短弹性扩容的时间，减少弹性部分的扩容成本，并尽可能充分利用已有资源。当业务流量下降后，可以快速释放部署在BCI上的Pod，从而降低您的使用成本。 **混合集群架构** 如果您在本地IDC，或者百度云的BCC上自建了Kubernetes集群，则可以通过部署虚拟节点（VNode）的方式来使用BCI。VNode对标原生kubernetes节点，内置了virtual-kubelet、kube-proxy等组件，兼容原生kubernetes节点API。 当有Pod调度到VNode上时，VNode会自动创建并管理底层的BCI资源。 在VNode上运行的每个Pod都对应一个BCI实例，架构如下图所示：



功能限制 由于公有云安全性及虚拟节点带来的限制，BCI目前还不支持Kubernetes中HostPath、DaemonSet等功能，如下表所示。

不支持的功能	说明	推荐替代方案
HostPath	挂载本地宿主机文件到容器中	使用emptyDir、云盘或者CFS文件系统
HostNetwork	将宿主机端口映射到容器上	BCI将忽略HostNetwork字段，推荐使用type=LoadBalancer的负载均衡
DaemonSet	在容器所在宿主机上部署Static Pod	通过sidecar形式在Pod中部署多个镜像
Privileged权限	容器拥有privileged权限	去除业务逻辑特权依赖
type=NodePort的Service	将宿主机端口映射到容器上	使用type=LoadBalancer的负载均衡

调度方式 对于混合使用普通节点和虚拟节点的Kubernetes集群，您可以根据需要将Pod调度到VNode，以BCI来运行。主要方式如下：

- 手动调度通过配置nodeSelector和tolerations、指定nodeName的方式，可以手动将Pod调度到VNode。具体操作，请参见通过标签将Pod调度到VNode上运行。

使用BCI功能 在Kubernetes集群中创建Pod到BCI时，为充分使用BCI提供的功能，在不改变Kubernetes语义的前提下，您可以根据需求为Pod添加Annotation。Annotation需添加到Pod级别的metadata中，支持的Annotation列表以及配置示例，请参见BCI Pod Annotation。

自建Kubernetes集群对接VNode

如果您在线下IDC或者百度云BCC上自建了Kubernetes集群，您需要在集群中部署虚拟节点（VNode）来使用BCI。本文为您介绍自建的Kubernetes集群如何对接VNode。

前提条件

- 已部署好Kubernetes集群，且集群的版本属于1.18~1.22版本。
- Vnode需要占用您集群资源，预计是1 vCPU、2 GiB内存，请提前在集群中准备好资源。
- 如果您的Kubernetes集群部署在线下IDC或者其他云厂商，请确保已通过专线或者VPN网关等方式打通IDC和云上网络。

准备工作 操作前，请准备创建虚拟节点所需的参数信息，并了解虚拟节点所需的权限信息。需要准备的参数如下表所示。

参数	描述	操作
地域（region）	地域指的是物理的数据中心。请根据您以及您目标用户所在的地理位置，资源价格等因素选择合适的地域。 不同地域云产品之间内网不互通；建议选择最靠近您目标用户的地域，以降低访问延时，创建成功后 **不支持** 更换地域	您可以通过容器实例BCI控制台获取所支持的地域信息。
私有网络（VPC）	私有网络(Virtual private Cloud，VPC) 是用户自定义的虚拟网络，，不同的专有网络之间逻辑上彻底隔离。 更多信息，请参考 私有网络 VPC。	您可以在私有网络控制台的创建并查看专有网络。
子网（subnet）	子网是 VPC 内的用户可定义的IP地址范围，在私有网络VPC中创建BCI及其相关资源时，需要指定子网（subnet）。 更多信息，请参考 私有网络 VPC - 子网。	您可以在私有网络子网控制台的创建并查看子网，根据已选的VPC来选择对应的子网。
安全组（securityGroup）	安全组是一种虚拟防火墙，可以控制组内资源的进出流量，从而提高网络安全性。 更多信息，请参考 私有网络 VPC - 安全组。	您可以在私有网络安全组控制台的创建并查看安全组，根据已选的VPC来选择对应的安全组。
AK/SK	AK（Access Key ID）/SK（Secret Access Key）用于对用户的调用行为进行鉴权和认证，相当于百度智能云API专用的用户名及密码。 更多信息，请参考 AK/SK简介。	您可以在管理控制台获取AK/SK。
BCI API 服务域名	BCI对外暴露的正式OpenAPI的Endpoint	您可以在 BCI线上文档，根据不同的地域获取不同OpenAPI地址。

步骤一：签发证书 需要签发virtual-kubelet（VK）使用的双向客户端证书用于API端点认证，以及kube-proxy sidecar使用的证书（可选）。
签发证书依赖cfssl工具，集群CA证书和CA Key，集群以及CA Config。

1. 准备访问集群的 hosts 列表：

- 集群的公网访问地址（如有）：可以直接从 kubeconfig 文件中，server 字段获取
- apiserver 的集群内访问地址：

```
kubectl get svc kubernetes
```

完整的 Hosts 列表：

```
"hosts": [
  "${公网访问地址}"（如有），
  "${集群内访问地址}",
  "*.cluster.local",
  "127.0.0.1"
]
```

2. 签发VK客户端证书：

```
###### 准备自建集群的CA证书及配置文件，一般在k8s master节点的 /etc/kubernetes/pki 目录下，获取的相关文件如下:
$ ls
ca-config.json  ca-key.pem  ca.pem

###### 设置 CA_PATH 环境变量，将存放CA证书的目录设置为环境变量 ${CA_PATH}
$ export CA_PATH=`pwd`

###### 设置 CA_PROFILE 环境变量，将CA配置文件中定义的证书签发策略名保存到环境变量中
$ export CA_PROFILE=$(jq -r '.signing.profiles | to_entries[0].key' ca.config)

###### 创建vk证书签发的csr文件
$ cat >vk-csr.json <<EOF
{
  "CN": "system:node",
  "hosts": [
    "xx.xx.xx.xx", #用户集群的公网访问 IP，如192.168.0.17
    "yy.yy.yy.yy", #用户集群的集群内访问 IP，如192.168.18.4。可通过kubectl cluster-info获取
    "*.cluster.local",
    "127.0.0.1"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "system:node",
      "OU": "cloudnative"
    }
  ]
}
EOF

###### 使用cfssl生成证书
$ cfssl gencert -ca=${CA_PATH}/ca.pem -ca-key=${CA_PATH}/ca-key.pem -config=${CA_PATH}/ca-config.json -profile=${CA_PROFILE} vk-csr.json | cfssljson -bare vk

###### 证书文件
$ ls
vk.csr  vk-csr.json  vk-key.pem  vk.pem
```

签发kube-proxy sidecar证书（可选）：

```
###### 设置 CA_PATH 以及 CA_PROFILE 变量，方法同上
$ ls ${CA_PATH}
ca-config.json  ca-key.pem  ca.pem

###### 创建kube-proxy证书签发的csr文件
$ cat >kube-proxy-csr.json <<EOF
{
  "CN": "system:kube-proxy",
  "hosts": [
   "xx.xx.xx.xx", #用户集群的公网访问 IP，如192.168.0.17
   "yy.yy.yy.yy", #用户集群的集群内访问 IP，如192.168.18.4。可通过kubectl cluster-info获取
   "*.cluster.local",
   "127.0.0.1"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "system:kube-proxy",
      "OU": "cloudnative"
    }
  ]
}
EOF

###### 使用cfssl生成证书
$ cfssl gencert -ca=${CA_PATH}/ca.pem -ca-key=${CA_PATH}/ca-key.pem -config=${CA_PATH}/ca-config.json -
profile=${CA_PROFILE} kube-proxy-csr.json | cfssljson -bare kube-proxy

###### 证书文件
$ ls
kube-proxy.csr  kube-proxy-csr.json  kube-proxy-key.pem  kube-proxy.pem
```

将生成的vk.pem, vk-key.pem, kube-proxy.pem, kube-proxy-key.pem置于同一目录下，生成证书secret：

```
 $ ls
kube-proxy.csr  kube-proxy-csr.json  kube-proxy-key.pem  kube-proxy.pem  vk.csr  vk-csr.json  vk-key.pem  vk.pem

###### 需指定namespace为kube-system
$ kubectl -n kube-system create secret generic vk-certs --from-file ./
```

步骤二：创建AK/SK secret 您可以在管理控制台获取AK/SK，然后将AK/SK保存到用户自建集群的secret中。

```
###### 需指定namespace为kube-system
$ kubectl -n kube-system create secret generic vk-aksk \
 --from-literal=ak=<Your AK> \
 --from-literal=sk=<Your SK>
```

步骤三：部署virtual-kubelet

```
 $ helm repo add bce-public https://kubernetes-charts.baidubce.com/public
###### 需指定namespace为kube-system
$ helm -n kube-system install vk -f values.yaml bce-public/cce-virtual-kubelet
```

参考values.yaml如下

```
  ###### Default values for bci-virtual-kubelet.
###### This is a YAML-formatted file.
###### Declare variables to be passed into your templates.

###### Virtual kubelet image. Keep the tag empty to use the default tag.
images:
  virtualKubelet: # Virtual kubelet image. Keep the tag empty to use the default tag.
    repository: "registry.baidubce.com/cce-plugin-pro/bci-virtual-kubelet"
    tag: "bciv2"
  kubectl: # Kubectl image. Keep the tag empty to use the default tag.
    repository: "registry.baidubce.com/cce-plugin-pro/kubectl"
    tag: ""

###### Maximum log backups for virtual kubelet. Each log file will consume 256MiB disk space.
maximumLogBackups: "4"
###### 必填项，地域简称. 取值项: bj,su,gz,bd,hkg,fwh
region: ""

###### 必填项，用户自建集群的cluster id. 需要保证唯一性。
clusterID: ""

###### Attributes of BCI pods running on the virtual node.
bci:
  # Subnet list, specify one subnet at least.
  # If multiple subnets are specified, one subnet will be randomly chosen for each pod creation attempt.
  # 必填项，subnets
  subnets:
  - zone: ""
    subnetID: ""
  - zone: ""
    subnetID: ""
  # 必填项，securityGroupID，Security group applied on pod.
  securityGroupID: ""

###### Attributes of virtual node.
virtualNode:
  # Virtual node count to create, default 1.
  nodeCount: 1
  # The virtual nodes will be named as ${nodeNamePrefix}-0, ${nodeNamePrefix}-1, ...
  nodeNamePrefix: "bci-virtual-kubelet"
  # CPU capacity.
  cpuCapacity: "1000"
  # Memory capacity.
  memoryCapacity: "4Ti"
  # Pods capacity.
  podsCapacity: "1000"
  # IPs capacity.
  ipsCapacity: "1000"
  # ENIs capacity.
  enisCapacity: "1000"
  # Ephemeral-storage capacity.
  ephemeralStorageCapacity: "40Ti"
  # Enable NodeLease or not. NodeLease is only available for k8s version > 1.14
  enableNodeLease: true
  # Enable pod cache or not.
  enablePodCache: true

###### Keep the endpoints empty to use the default values. Override them if needed.
endpoints:
  # 必填项，BCI 在目标地域 OpenAPI Endpoint
```

```
bci: ""
```

安装参数说明（带有*号的参数为用户必填项）：

参数	描述	获取方式	示例
region*	地域	填写所在地域的英文缩写，仅支持开放BCI产品的地域	bj
clusterID*	集群id，用于区分不同k8s集群的标识，会在BCI页面显示	保证对于不同集群不重复即可，长度不可超过20个字符	k8s-xxxxxxxx
bci.subnets*	创建BCI的子网信息，需要填写子网id和子网所属可用区。至少填写一个子网；指定多个子网时，每次创建会从子网列表中随机选择一个。	从私有网络VPC-子网页面获取	- subnetID: sbn-4syxw5hsuc7e zone: zoneC
bci.securityGroupID*	安全组ID	从私有网络VPC-安全组页面获取	g-f90242zt83dd
images.virtualKubelet.repository	virtual-kubelet镜像repository，需要覆盖默认值时填写	-	registry.baidubce.com/cce-plugin-pro/bci-virtual-kubelet
images.virtualKubelet.tag*	virtual-kubelet镜像tag	-	bciv2
images.kubectl.repository	kubectl镜像repository，需要覆盖默认值时填写	-	registry.baidubce.com/cce-plugin-pro/kubectl
maximumLogBackups	virtual-kubelet组件日志文件保留，每份日志文件将占用所在节点/var/log下256Mi空间	-	"4"
virtualNode.nodeCount	需要创建的虚拟节点数量，默认为1	-	1
virtualNode.nodeNamePrefix	虚拟节点名称前缀，虚拟节点将按照 <前缀>-0 ， <前缀>-1 ，... 依次命名	-	bci-virtual-kubelet
virtualNode.cpuCapacity	虚拟节点CPU Capacity	-	"1000"
virtualNode.memoryCapacity	虚拟节点Memory Capacity	-	"4Ti"
virtualNode.podsCapacity	虚拟节点Pods Capacity	-	"1000"
virtualNode.ephemeralStorageCapacity	虚拟节点Ephemeral Storage Capacity	-	"40Ti"
virtualNode.enableNodeLease	是否开启Node Lease，K8S 1.14版本以上建议开启	-	true
virtualNode.enablePodCache	是否开启Pod Cache，可以提升Pod并发创建性能，无特殊需求建议开启	-	true
endpoints.apiserver	BCI中kube-proxy sidecar访问K8S APIServer的端点，不用可以留空	-	"https://192.168.0.1:6443"
endpoints.bci	BCI服务Endpoint，默认为空则使用默认值，需要覆盖时填写	-	"bci.bj.baidubce.com"

安装成功后，在集群中执行kubectl get node，看到对应的虚拟节点创建即可。

步骤四：配置反亲和性策略 由于VNode不是真实节点，因此无法运行DaemonSet。创建VNode后，您需要修改kube-proxy的DaemonSet，配置nodeAffinity来禁止DaemonSet调度到VNode。相关配置如下：

1. 执行以下命令修改DaemonSet配置。

2. 配置nodeAffinity。在spec>template>spec下添加以下YAML：

```
affinity:
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: type
        operator: NotInvalues:
        - virtual-kubelet
```

将Pod调度到VNode

本文介绍如何将Pod调度到VNode。 对于自建的Kubernetes集群，即混合使用普通节点和虚拟节点（VNode）的模式下，您可以通过配置nodeSelector和tolerations，将Pod调度到VNode上，以BCI来运行。

概述 VNode对标原生kubernetes节点，在混合使用普通节点和VNode的模式下，支持通过配置nodeSelector和tolerations的方式，将Pod调度到VNode。

配置示例 在Pod Spec中添加如下的nodeSelector和tolerations字段，将Pod调度到虚拟节点上。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    type: pod-perf-test
  name: nginx-test
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      type: pod-perf-test
  template:
    metadata:
      labels:
        type: pod-perf-test
    spec:
      containers:
      - image: registry.baidubce.com/qatest/nginx:1.23.0
        imagePullPolicy: IfNotPresent
        name: pod-perf-test
        resources:
          limits:
            cpu: 0.25
            memory: 512Mi
          requests:
            cpu: 0.25
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: Default
      nodeSelector:
        type: virtual-kubelet
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

调度到虚拟节点上的Pod会以BCI Pod的形式启动。

混合调度

以多云的方式混合部署的客户，稳态资源一般用自建/公有云K8S集群IDC，弹性资源用BCI容器实例，混合调度插件负责无侵入协调客户负载在K8S集群节点及BCI分布。其主要场景是帮助客户将弹性负载弹性到云上容器实例。
基于该插件，支持如下3种调度策略：

策略	功能描述
bciOnly	工作负载容器强制调度到BCI
localOnly	工作负载强制调度到本地/公有云IDC
localprefer	当工作负载扩容时，当自建/公有云IDC有资源，优先调度到自建/公有云IDC节点；若自建/公有云IDC没资源，可调度到BCI。 当工作负载缩容时，根据配置，优先缩容缩容优先级高的BCI或IDC的实例。还可限制本地IDC/BCI最多可部署的实例数。

1. 前提条件 用户在所使用的 k8s 集群中，已部署 virtual kubelet，如何部署详见
https://cloud.baidu.com/doc/BCI/s/slbdd8fue **2. 插件安装**

```
###### 1. 添加 bci helm repo
helm repo add bci https://registry.baidubce.com/chartrepo/bci-online-public
###### 2. 安装 bci schedule profile 混合调度插件
helm install schedule-profile --version 0.2 bci/schedule-profile
```

3. 通过 ScheduleProfile 控制 pod在k8s集群和BCI之间的分布 用户可以通过创建ScheduleProfile资源，控制pod在本地IDC和BCI之间的分布情况，ScheduleProfile 具体定义如下

```
apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: test-schedule-profile
  namespace: default
spec:
  objectSelector:
    namespace:
      labels:
        type: hybrid
    pod:
      labels:
        type: hybrid
  strategy: localprefer
  distribution:
    local:
      maxNum: 20
      deletionPriority: 10
    bci:
      deletionPriority: 20
```

参数说明：

- objectSelector：控制调度策略覆盖的对象范围。当设置 namespace labels 时，匹配 labels 的所有 namespace 中的 pod，在该调度策略覆盖范围。当设置 pod labels 时，匹配 labels 的所有 pod，在该调度策略覆盖范围。
  - 当 namespace 或者 pod 的 labels 中有多个条目时，对应 namespace 或者 pod 需同时匹配所有 label，才在该调度策略覆盖范围；仅匹配单个 label 不在该调度策略覆盖范围。e.g.

    ```
    objectSelector:
     namespace:
       labels:
         type: hybrid
         exp_enabled: false
    ```
    此时，同时包含 type: hybrid 和 exp_enabled: false 的 namespace 中的 pod 在调度策略覆盖范围
  - namespace 和 pod 的 labels 同时存在时，匹配 namespace labels 的 namespace 中的 pod 及 匹配 pod labels 的所有 pod 都在该调度策略覆盖范围。
  - 当且仅当一个pod只对应一个ScheduleProfile时，这个pod在其调度策略覆盖范围内。若一个 pod 同时匹配多个 ScheduleProfile 或不匹配任何 ScheduleProfile，则这个 pod 不受任何 ScheduleProfile 影响。若需要通过多个 ScheduleProfile 管理不同的工作负载，建议通过配置 pod labels 进行区分，规避工作负载同时匹配多个 ScheduleProfile 的风险。
- strategy：调度策略选择，目前支持 localprefer/bciOnly/localOnly三种。
- distribution：仅localprefer策略可设置该字段。
  - local 描述本地IDC的部署策略，maxNum 限制本地IDC最多可部署的pod数目；deletionPriority 的作用是，当

deployment缩容时，部署在本地IDC的pod被删除的优先级，值越大，越容易被删除。

- bci 描述bci的部署策略，maxNum 限制bci最多可部署的pod数目，任何时刻都不会被打破；deletionPriority 的作用是，当deployment缩容时，部署在bci的pod被删除的优先级，值越大，越容易被删除。
- bci 和 local 中的 maxNum 不能同时设置，maxNum取值范围 [0~int32]，0为不限制；deletionPriority取值范围[-100, 100]。 **4. 约束和说明**

- localprefer策略本地自建IDC和BCI的最大部署pod数（maxNum）不能同时设置。
- 可以调整ScheduleProfile的覆盖范围（objectSelector），但是要注意，ScheduleProfile对存量已调度pod无影响，比如调大ScheduleProfile的覆盖范围后，新纳入覆盖范围的pod的删除优先级不会发生变化。
- deletionPriority 仅对 1.22 版本以上的 k8s 集群有效。若集群版本低于 1.22，工作负载缩容时，随机选择实例删除。
- 在deployment滚动升级场景下，推荐配置尽可能小的maxSurge值（如直接配置为0），避免出现升级时限制maxNum的区域实际部署数少于maxNum的现象。
- Pod只能关联一个ScheduleProfile，如果一个Pod的labels或所属namespace的labels存在于多个ScheduleProfile中，系统不会对该Pod做任何操作，效果类似于Pod没有关联的ScheduleProfile。
- ScheduleProfile的增删，可能导致Pod所属的ScheduleProfile变化，进而打破MaxNum等策略的限制。
- 若用户负载中明确指定了 nodeName，其关联的 pod 不受 ScheduleProfile 限制，可能打破 MaxNum 等策略的限制。
- 为了避免混合调度影响系统组件，对于需要开启混合调度的 namespace，必须打上开启混合调度的标签“hybrid-schedule = enabled”

5. 使用样例 使用profile配置管理集群内pod，通过labelSelector类方式关联profile和pod，并配置关联pod的分配策略，实现pod在自建/公有云K8S集群IDC和云上BCI的分配或数量限制。

本地突发负载弹性上容器实例场景

当工作负载扩容，本地资源不足或者达到设置的最大值时，将实例溢出到云端BCI容器实例，限制本地最多创建30个实例。工作负载缩容时，优先释放云端BCI容器实例。工作负载缩容时，优先释放云端实例。

为避免影响k8s系统组件，对需要开启混合调度功能的 namespace 打上 label，以 default namespace 为例

注：此处仅仅只是在某个名字空间开启混合调度功能，如果想让整个名字空间都被某个调度策略覆盖（受某个调度策略管理），仍然需要在 ScheduleProfile 的 objectSelector 中配置。

(1)对需要开启混合调度功能的 namespace 打上开启混合调度功能标签“hybrid-schedule=enabled”

```
kubectl label namespace default hybrid-schedule=enabled
```

(2)配置 ScheduleProfile

```
  apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: burst-to-cloud
  namespace: default
spec:
  objectSelector:
    pod:
      labels:
        type: hybrid
  strategy: localprefer
  distribution:
    local:
      maxNum: 3
      deletionPriority: 10
    bci:
      deletionPriority: 20
```

(3) 提交工作负载，以下为一个工作负载样例

```
  kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        type: hybrid
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          imagePullPolicy: IfNotPresent
      imagePullSecrets:
        - name: default-secret
```

本地强制负载弹性到容器实例场景

将工作负载强制调度到BCI容器实例时，可使用bciOnly策略。

(1)对需要开启混合调度功能的 namespace 打上开启混合调度功能标签“hybrid-schedule=enabled”

```
  kubectl label namespace default hybrid-schedule=enabled
```

(2)配置 ScheduleProfile

```
apiVersion: scheduling.bci.cloud.baidu.com/v1
kind: ScheduleProfile
metadata:
  name: burst-to-cloud
  namespace: default
spec:
  objectSelector:
    pod:
      labels:
        type: hybrid
  strategy: bciOnly
```

6. 插件卸载

```
helm uninstall schedule-profile
```

通过标签将Pod调度到VNode上运行

本文介绍如何通过指定 Pod 标签或命名空间标签将 Pod 调度到虚拟节点 VNode 上运行。

前提条件

- 已部署好 kubernetes 集群，且集群版本为 1.20 及以上
- 在自建集群中安装 cce-virtual-kubelet 组件，具体操作请参考 自建集群对接VNode
- 可以通过 kubelet 连接 kubernetes 集群，具体操作请参考 通过kubelet连接集群

将 Pod 调度到 VNode 上运行

通过 Pod 标签将 Pod 调度到 VNode 上

创建 Pod 时添加标签 **baidubce.com/bci=true** ，将 Pod 调度到 VNode 上运行。示例如下：

```
metadata:
  labels:
    baidubce.com/bci: "true"
```

查看是否调度成功，可查看到 Pod 被调度到虚拟节点 (bci-virtual-kubelet-0) 上。

```
kubectl get pod -o wide
NAME       READY      STATUS     RESTARTS    AGE     IP              NODE                  NOMINATED NODE
READINESS   GATES
test-pod   1/1        Running    0           103s    172.254.129.54    bci-virtual-kubelet-0     <none>           <none>
```

通过命名空间标签将 Pod 调度到 VNode 上

在命名空间上添加标签 **baidubce.com/bci=true** 后，在该命名空间内创建 Pod，Pod 将被调度到 VNode 上。示例如下：

创建命名空间

```
kubectl create ns vk
```

为 Pod 所在命名空间 **vk** 添加标签 **baidubce.com/bci=true**

```
kubeclt label namespace vk baidubce.com/bci=true
```

创建 Pod 指定命名空间为 **vk**

```
metadata:
namespace: vk
```

查看是否调度成功，可查看到 Pod 被调度到虚拟节点 (bci-virtual-kubelet-0) 上。

```
kubectl -n vk get pod -o wide
NAME       READY   STATUS    RESTARTS   AGE    IP             NODE                 NOMINATED NODE   READINESS
GATES
test-pod   1/1     Running   0          13s    172.254.129.52  bci-virtual-kubelet-0   <none>           <none>
```

BCI自定义condition

Pod status condition可以提供Pod状态和健康状况的信息。本文汇总了BCI Pod 的condition并给出说明。

reason	message	说明
Creating	Creating	创建请求已经提交，BCI pod正在创建中
AutoInstanceTypeMatch	The best matched instanceType for current instance is xc xGi	对提交的BCI Pod的资源规格进行规整。
NO_STOCK	Create BCI failed because the specified instance is out of stock	当前可用区的BCI资源库存不足。您可以使用多可用区的方式创建BCI Pod来提高创建成功率。

BCI Pod

BCI Pod概述

BCI能为Kubernetes提供基础的容器Pod运行环境，每个BCI实例相当于一个Pod。本文介绍BCI Pod的配置、创建方式和生命周期。

基本配置 基于Kubernetes社区的Virtual Kubelet技术，BCI支持以虚拟节点（VK）的形式接入到Kubernetes集群中。一个BCI实例相当于一个Pod，包含以下几部分配置：

1. 规格：规格包括vCPU、内存等配置，定义了BCI Pod的计算性能等。创建BCI Pod时，您可以指定BCI规格（直接指定vCPU和内存）来满足GPU、增强网络能力等特殊需求。
2. 容器镜像：部署容器应用时，需要准备好容器镜像。容器镜像包含容器应用运行所需的程序、库文件、配置等。拉取镜像时，需要保证网络畅通，推荐您使用镜像缓存功能来节约实例的启动耗时。
3. 网络：一个BCI Pod将占用所属VPC下的交换机的一个弹性网卡资源，默认具备一个内网IP地址。如果需要连接公网，例如需要拉取公网镜像。则需要为BCI Pod绑定EIP，或者为所属VPC绑定NAT网关。
4. 存储：一个BCI Pod默认有 20 GiB的临时存储空间，您可以根据需要增加临时存储空间。如果想要保留存储的文件，建议使用外挂数据卷，支持挂载CFS、EmptyDir和ConfigMap数据卷。
   创建方式 创建方式概述 根据业务场景和使用场景，BCI Pod支持指定vCPU和内存的定义方式，对应计费模式如下：

创建方式	计费说明	相关文档
指定vCPU和内存	根据您创建时指定的vCPU和内存进行计费。对于不满足要求的vCPU和内存规格，系统将自动进行规整，并按自动规整后的规格进行计费。	指定vCPU和内存创建Pod

> **说明** 更多关于BCI Pod计费的信息，请参见BCI实例计费

优化使用成本 根据您的业务特征，在按量付费使用BCI的基础上，您还可以使用预留实例券来降低资源使用成本。

- 对于长时间运行的稳定业务负载，推荐使用预留实例券来抵扣BCI实例账单。具体操作，请参见使用预留实例券。

应对库存不足 BCI提供容器的云上运行资源，在大规模创建BCI Pod的场景下，您所在地域和可用区可能会存在指定资源售罄的情况，建议您使用多可用区的方式创建资源，保证创建成功率。具体操作，请参见：

- 多可用区创建实例

BCI Pod生命周期

本文介绍BCI实例的生命周期状态，您可以根据实例状态，设计和实现符合您业务逻辑的处理逻辑。

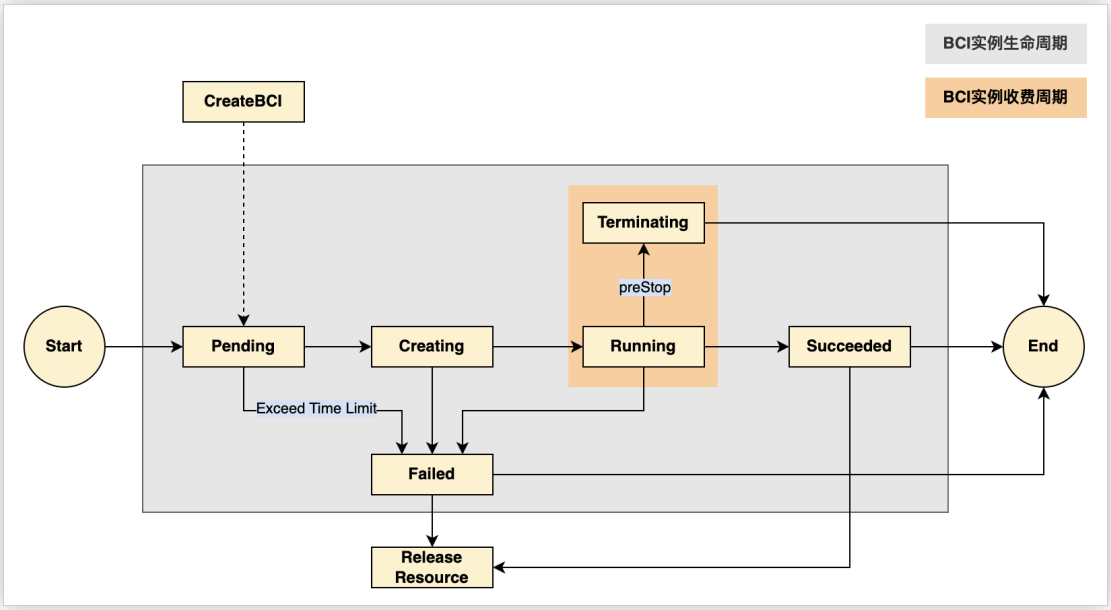
BCI实例状态

在实例的生命周期中，不同的阶段有其固有的状态，具体如下表所示：

BCI Pod状态	说明	对应Kubernetes Pod状态	是否收费
等待创建（Pending）	BCI Pod等待创建。	Pending	否
启动中（Creating）	BCI Pod中有一个或多个容器还在启动中，并且没有处于运行中的容器。	Pending	否
运行中（Running）	BCI Pod中所有容器均已经创建成功，并且至少有一个容器正在运行中。	Running	是
终止中（Terminating）	BCI Pod正在终止。对于运行中的实例，如果配置了preStop，则在删除实例时，Pod将进入Terminating状态。执行完preStop后，BCI Pod将自动删除。	Running	是
运行成功（Succeeded）	BCI Pod中所有容器均已运行成功终止，并且不会再重启。	Succeeded	否
创建/运行失败（Failed）	BCI Pod创建失败。 BCI Pod中所有容器均已运行终止，并且至少有一个容器是运行失败终止，即容器以非0状态退出或者被系统终止。	Failed	否

> **重要** BCI实例的重启策略仅决定实例内容器的行为，BCI实例不会被自动重启。

BCI实例的生命周期状态转换如下图所示：



> **说明**
>
> - 当BCI实例运行终止后，底层计算资源将会被回收，随实例一起创建的其它资源（例如EIP等）默认随实例一起释放。

容器状态

状态	说明
启动中（Waiting）	容器正在等待创建，还未开始运行。 一般在InitContainer运行时，应用容器会处于Waiting状态，直到InitContainer退出。
运行中（Running）	容器已经成功创建，并且正在运行。
运行终止（Terminated）	容器运行终止并退出，包括运行成功终止和运行失败终止。

指定vCPU和内存创建Pod

如果没有特殊的规格需求，推荐您指定vCPU和内存来创建BCI Pod（即BCI实例），系统会尝试使用多种BCC规格进行容器创建，以提供比BCC单规格更好的弹性和资源供应能力。 **规格说明** 创建BCI实例时，如果指定的vCPU和内存不符合要求，系统将自动按照BCI支持的规格进行规整。规整时将向最接近的BCI规格进行规整，同时需满足指定的vCPU和内存≤BCI规格的vCPU和内存。例如：创建BCI实例时，声明了7 vCPU，13 GiB内存，则实际创建的BCI实例为8 vCPU，16 GiB内存。

BCI支持的规格如下表所示

CPU/核	内存区间（GiB）
0.25	0.5、1、2
0.5	1、2、3、4
1	1 、2、4、8
2	4 、8、16
4	8 、16、32
8	16 、32
12	24 、 48
16	32、64
32	64、128

> **说明：**
> 1、32c限定白名单可用，如需32c资源请提交工单申请白名单。
> 2、目前BCI实例仅支持挂载一块弹性网卡，暂不支持多网卡能力。

配置说明 指定vCPU和内存创建BCI Pod时，支持以下方式：

- 指定Pod内容器的vCPU和内存：通过定义Containers的limits或requests来指定，建议使用limits。

如果您没有指定，或者同时指定了limits、requests，实际生效情况如下：

场景	Pod规格
全部未指定	按默认规格（1vCPU，2 GiB内存）
仅指定limits	汇总limits
仅指定requests	汇总requests
同时指定limits和requests	分别汇总limits和requests，按两者中较大值

说明：如果汇总limits、requests后的规格，系统会自动进行资源规整，并按规整后的规格进行计费。
指定Pod内容器的vCPU和内存为Kubernetes默认方式。每个BCI Pod最多可以支持20个容器，每个容器的vCPU和内存规格可以自定义配置。

配置示例如下:

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: testpod
  labels:
    app: testpod
spec:
  replicas: 10
  selector:
    matchLabels:
      app: testpod
  template:
    metadata:
      labels:
        app: testpod
    spec:
      containers:
      - name: gohttp
        image: registry.baidubce.com/qa-test/gohttp:v0.0.1
        imagePullPolicy: Always
        command:
          - sleep
          - "36000"
        resources:
          limits:
            cpu: 0.25
            memory: 1Gi
          requests:
            cpu: 0.25
            memory: 1Gi
```

创建GPU实例

本文介绍如何创建并使用BCI GPU实例。 **BCI GPU规格说明** BCI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。

BCI 规格名称	CPU	内存	GPU类型	显存	GPU卡数
bci.gna2.c8m36.1a10	8	36	Nvidia A10 PCIE	24*1	1
bci.gna2.c18m74.1a10	18	74	Nvidia A10 PCIE	24*1	1
bci.gna2.c30m118.2a10	30	118	Nvidia A10 PCIE	24*2	2
bci.gna2.c62m240.4a10	62	240	Nvidia A10 PCIE	24*4	4

创建实例 配置说明如下

- 指定GPU型号，需要在 Annotation 指定 GPU型号，请注意， Annotation需要配置在Pod Spec中，而不是Deployment Spec中。

```
annotations:
bci.virtual-kubelet.io/bci-gpu-type: "Nvidia A10 PCIE"
```

- 指定资源配置：GPU卡数、CPU和内存数量

```
resources:
limits:
  nvidia.com/gpu: 1 # GPU卡数
  cpu: 8 # CPU核数
  memory: 36Gi # MEM数量
requests:
  nvidia.com/gpu: 1 # GPU卡数
  cpu: 8 # CPU核数
  memory: 36Gi # MEM数量
```

完整业务YAML示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spot-deployment-test-gpu-wzy
  labels:
    run: ooo
spec:
  replicas: 1
  selector:
    matchLabels:
      run: ooo
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: ooo
      annotations:
        bci.virtual-kubelet.io/bci-gpu-type: "Nvidia A10 PCIE"
        bci.virtual-kubelet.io/bci-logical-zone: "zoneF" # 填写对应资源的可用区
        bci.virtual-kubelet.io/bci-subnet-id: "xxxxxx" # 子网需要和可用区对应
      name: spot-deployment-test-wzy-bid
    spec:
      volumes:
```

```
  volumes:
  - name: podinfo
    downwardAPI:
      defaultMode: 420
      items:
      - fieldRef:
          apiVersion: v1
          fieldPath: metadata.labels['mylabel']
        path: mylabel
      - fieldRef:
          apiVersion: v1
          fieldPath: metadata.annotations['myannotation']
        path: myannotation
      - fieldRef:
          apiVersion: v1
          fieldPath: metadata.labels
        path: labels
      - fieldRef:
          apiVersion: v1
          fieldPath: metadata.annotations
        path: annotations
      - path: workload_cpu_limit
        resourceFieldRef:
          containerName: ooo1
          divisor: 1m
          resource: limits.cpu
      - path: workload_cpu_request
        resourceFieldRef:
          containerName: ooo1
          divisor: 1m
          resource: requests.cpu
      - path: workload_mem_limit
        resourceFieldRef:
          containerName: ooo1
          divisor: 1Mi
          resource: limits.memory
      - path: workload_mem_request
        resourceFieldRef:
          containerName: ooo1
          divisor: 1Mi
          resource: requests.memory
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: ooo1
    env:
    - name: "MY_CPU_LIMIT"
      valueFrom:
        resourceFieldRef:
          containerName: ooo1
          resource: limits.cpu
    - name: "MY_CPU_REQ"
      valueFrom:
        resourceFieldRef:
          containerName: ooo1
          resource: requests.cpu
    - name: "MY_IP"
```

```
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.podIP
      volumeMounts:
      - name: podinfo
        mountPath: /etc/podinfo
      resources:
        limits:
          nvidia.com/gpu: 1 # GPU卡数
          cpu: 8 # CPU核数
          memory: 36Gi # MEM数量
        requests:
          nvidia.com/gpu: 1 # GPU卡数
          cpu: 8 # CPU核数
          memory: 36Gi # MEM数量
```

多可用区创建实例

当您在应对突发流量，进行业务的快速水平扩容时，或者启动大量实例进行Job任务处理时，可能会遇到可用区对应规格实例库存不足等特殊情况，从而导致实例创建失败，影响业务。此时，您可以采用指定多可用区的方式来创建实例，提高实例创建的成功率。 **前提条件** 已在要使用的专有网络VPC下创建多个不同可用区的子网。

- 关于如何创建子网，请参见私有网络 VPC - 子网。

背景信息 创建BCI实例时，可以通过指定多个子网来指定多个可用区，系统会随机把请求分散到所有指定的可用区中，来分散压力，如果在某一个可用区遇到没有库存的情况，会自动切换到下一个可用区继续尝试创建。
指定多可用区时，需注意以下限制：

- 指定的子网必须属于同一个VPC。
- 最多可以指定10个子网。
  配置说明 开启使用多可用区资源的步骤

1. 对于 job、deployment 等工作负载，只需要在 spec.template.metadata.annotations 中新加一个 annotation 即可。
2. annotation 的 key 为 "bci.virtual-kubelet.io/bci-subnet-ids" ，value是多个子网以英文逗号连接的字符串，如需使用多个可用区，则需填对应多个可用区的子网。
   配置示例 例如想使用可用区 D & F 的资源，在可用区D、F分别创建了sbn-xxx1，sbn-xxx2的子网 deployment 或 job yaml 中最终效果样例:

```
spec:
  template:
    metadata:
      annotations:
        "bci.virtual-kubelet.io/bci-subnet-ids":"sbn-xxx1,sbn-xxx2"
```

BCI Pod Annotation

在Kubernetes集群中创建BCI类型的Pod（即BCI实例）时，为充分使用BCI提供的功能，在不改变Kubernetes语义的前提下，您可以根据需求为Pod添加Annotation。 本文为您介绍创建BCI Pod时支持添加的Annotation，以及BCI Pod调度完成后会追加的Annotation。

BCI Pod支持的Annotation 创建BCI Pod时，通过在Pod metadata中指定Annotation，可以为对应Pod开启特定的特性，或者覆盖虚拟节点上的原有配置，支持添加的Annotation如下：

> **注意：**

- 列举的Annotation仅适用于创建到虚拟节点上的Pod，即BCI实例，调度到普通节点上的Pod不受这些Annotation影响。
- Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
- 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。
- 配置网络、安全组和可用区相关Annotation时，需要保证相互之间的一致性：
- 如子网与安全组应该同属一个VPC，子网与可用区参数需匹配等。
- 如未在Pod中单独配置，则默认使用虚拟节点本身的配置。
- 如通过Annotation指定子网参数，则必须同时通过Annotation指定可用区参数，可用区参数的值为子网所属可用区。

实例 | 功能及相关文档 | 参数 | 示例值 | 描述 | | --- | --- | --- | --- | | 配置BCI Pod所属可用区 | bci.virtual-kubelet.io/bci-logical-zone | "zoneF" | BCI实例所在可用区 | | 配置BCI Pod所属子网 | bci.virtual-kubelet.io/bci-subnet-id | "sbn-xxx" | BCI实例所属子网（子网短ID） | | 多可用区创建实例 | bci.virtual-kubelet.io/bci-subnet-ids | "sbn-xxx1,sbn-xxx2" | 支持指定多个可用区实现多可用区功能 | | 自定义设置BCI Pod的hosts | bci.virtual-kubelet.io/pod-host-aliases | "[{"ip":"10.10.xx.xx","hostnames":["example.com"]},{"ip":"10.10.yy.yy","hostnames":["foo.com","bar.com"]}]" | 自定义设置BCI Pod的hosts（即/etc/hosts）。 | | 为BCI Pod绑定自定义标签 | bci.virtual-kubelet.io/resource-tag | "key1:value1,key2:value2" | 绑定的标签（Tag）字符串，标签键和标签值之间用半角冒号隔开，多个标签之间用半角逗号隔开。 | | 为BCI Pod配置延迟销毁时长(Job类型Pod) | bci.virtual-kubelet.io/pod-delay-release-duration-minute | "10" | 设置延迟销毁时长，只针对job任务类型Pod(即restartPolicy为never)生效 | | 为BCI Pod配置延时销毁时长(Succeeded状态的Pod) | bci.virtual-kubelet.io/pod-delay-release-succeeded | "true" | 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 | | 创建GPU类型实例 | bci.virtual-kubelet.io/bci-gpu-type | "Nvidia A10 PCI"E | 指定GPU卡型号 | | 设置BCI Pod的故障处理策略 | bci.virtual-kubelet.io/bci-fail-strategy | "fail-back" | BCI Pod的故障处理策略，取值如下：

- fail-back：失败自动恢复。即Pod创建失败后自动尝试重新创建。
- fail-over：失败转移。效果等同于fail-back。
- fail-fast：快速失败。即Pod创建失败后直接报错。| | 自定义设置BCI Pod的最大Pending时长 | bci.virtual-kubelet.io/bci-max-pending-minute | "30" | 自定义设置Pod对应BCI实例的最大Pending时长，超时后系统会自动终止实例。
取值范围为10~1440的整数，单位为分钟。默认为1小时。| | 配置资源规整时忽略特定容器 | bci.virtual-kubelet.io/bci-resource-ignore-containers | "container1,container2" | 指定资源规整时忽略的容器列表 | | 强制终止Sidecar容器并忽略容器退出码 | bci.virtual-kubelet.io/bci-ignore-exit-code-containers | "sidecar1,sidecar2" | 指定需要忽略状态码的Sidecar容器列表 | | 忽略Sidecar容器的NotReady状态 | bci.virtual-kubelet.io/bci-ignore-not-ready-containers | "sidecar1,sidecar2" | 指定需要忽略Not Ready的Sidecar容器列表 |

网络

功能及相关文档	参数	示例值	描述
配置BCI Pod所属安全组	bci.virtual-kubelet.io/bci-security-group-id	"g-xxx"	指定实例的安全组ID
配置BCI Pod是否自动创建EIP	bci.virtual-kubelet.io/bci-create-eip	"true"	是否自动创建并绑定EIP • true：自动创建EIP • false: 不自动创建EIP
配置BCI Pod EIP线路类型	bci.virtual-kubelet.io/bci-create-eip-route-type	"BGP"	设置EIP的线路类型。不指定时，默认为BGP类型。可选值： • BGP：标准型BGP • BGP_S：增强型BGP • ChinaMobile：中国移动 • ChinaUnicom：中国联通 • ChinaTelcom：中国电信
配置BCI Pod EIP带宽	bci.virtual-kubelet.io/bci-create-eip-bandwidth	"10"	设置EIP带宽。带宽能力和EIP线路类型相关。带宽区间： • BGP：1～200 Mbps • BGP_S：100～5000 Mbps • ChinaMobile：1～5000 Mbps • ChinaUnicom：1～5000 Mbps • ChinaTelcom：1～5000 Mbps
配置BCI Pod EIP计费方式	bci.virtual-kubelet.io/bci-create-eip-paymethod	"ByBandwidth"	设置EIP计费方式 • ByTraffic：流量 • ByBandwidth：带宽
为BCI Pod绑定已有的EIP	bci.virtual-kubelet.io/bci-eip-ip	"10.10.xx.xx"	绑定已有的EIP。且此EIP状态必须可用(Available)。限定条件：一个EIP只能成功绑定到一个BCI实例上，若使用deployment方式，副本数replicas需设置为1，否则可能导致创建失败。
配置BCI Pod访问集群内Service	bci.virtual-kubelet.io/kube-proxy-enabled	"true"	BCI实例访问K8S ClusterIP类型Service

容器配置 | 功能及相关文档 | 参数 | 示例值 | 描述 | | --- | --- | --- | --- | | 配置NTP服务 | bci.virtual-kubelet.io/bci-ntp-server | "10.0.xx.xx" | 指定NTP服务器的地址 |

运维

功能及相关文档	参数	示例值	描述
查看Core dump文件	bci.virtual-kubelet.io/core-pattern	"/tmp/cores"	Core dump文件保存目录

BCI Pod追加的Annotation BCI Pod调度完成后会追加的Annotation如下表所示。您可以通过kubectl describe命令进行查询。

参数	示例值	描述
bci.virtual-kubelet.io/bci-subnet-id	sbn-xxx	BCI实例所属子网(短ID)
bci.virtual-kubelet.io/order-id	xxx	BCI实例对应的订单ID
bci.virtual-kubelet.io/pod-id	p-xxx	BCI实例短ID
bci.virtual-kubelet.io/bci-bound-eip-bandwidth	10	绑定的EIP实例带宽
bci.virtual-kubelet.io/bci-bound-eip-id	ip-xxx	绑定的EIP实例ID（短ID）
bci.virtual-kubelet.io/bci-bound-eip-ip	10.10.xx.xx	绑定的EIP实例IP地址
bci.virtual-kubelet.io/bci-bound-eip-paymethod	ByTraffic	绑定的EIP实例计费方式
bci.virtual-kubelet.io/bci-bound-eip-route-type	BGP	绑定的EIP实例线路类型

设置BCI Pod的故障处理策略

默认情况下，BCI Pod创建失败后，系统会自动重试尝试创建。如果您希望尽快得到创建结果以便及时处理故障，可以修改BCI Pod的故障处理策略。

配置说明 在虚拟节点上创建BCI Pod时，可能会因为库存不足等原因导致Pod创建失败，默认情况下，系统会自动进行重调度，尝试重新创建Pod。您可以通过添加bci.virtual-kubelet.io/bci-fail-strategy的Annotation来修改BCI Pod的故障处理策略，设置BCI Pod创建失败后是否尝试重新创建。

> **重要**
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

bci.virtual-kubelet.io/bci-fail-strategy的取值说明如下：

取值	说明	场景
fail-back	失败自动恢复。即Pod创建失败后自动尝试重新创建。此时，Pod会保持Pending状态，直到创建成功变为Running状态。	侧重成功率，能够接受Pod延迟交付。
fail-over	失败转移。效果等同于fail-back。	侧重成功率，能够接受Pod延迟交付。
fail-fast	快速失败。Pod创建失败后直接报错。Pod显示为Pending状态，由上层编排决定是否重试，或者把Pod创建调度到普通节点。	侧重效率，希望Pod快速交付，有完善的失败处理逻辑。

配置示例

```yaml
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
   app: nginx
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     name: nginx
     labels:
       app: nginx
     annotations:
       bci.virtual-kubelet.io/bci-fail-strategy: "fail-fast"  #设置Pod故障处理策略，不再重新创建
   spec:
     containers:
     - name: nginx
       image: nginx:1.7.9
       ports:
       - containerPort: 80
       resources:
         limits:
           cpu: "1"
           memory: "2Gi"
     restartPolicy: Always
```

以上YAML示例中，BCI Pod的故障处理策略为fail-fast。如果Pod长时间Pending，您可以查看Pod status.reason。

- 如果Pod status.reason为ContainerInstanceScheduleFailed，则表示BCI调度失败。此时查看Pod status condition，通过ContainerInstanceCreated的reason和message可以确定具体原因，进而采取相应措施，例如修改指定的规格，设置多可用区等。
- 如果Pod status.reason为空（fail-fast一般不会出现该情况），可以查看Pod status condition，通过ContainerInstanceCreated的status确认调度状态。
  - 如果ContainerInstanceCreated为False，且reason不是Creating，则表示BCI调度还未成功，需要继续等待。 以库存不足创建BCI Pod失败为例，当Pod的故障处理策略为fail-fast时，Pod status condition为ContainerInstanceCreated的示例如下：

> **说明**
>
> 如果Pod的故障处理策略为fail-back，Pod创建失败后系统会自动尝试重调度。此时，Pod status.reason不会显示ContainerInstanceScheduleFailed，您也可以查看Pod status condition，通过ContainerInstanceCreated的reason和message确定当前调度周期内调度失败的原因。

```
{
  "conditions": [
    {
        "lastProbeTime": "2025-03-04T15:13:26Z",
        "lastTransitionTime": "2025-03-04T15:13:26Z",
        "message": "Create BCI failed because the specified instance is out of stock",
        "reason": "NoStock",
        "status": "False",
        "type": "ContainerInstanceCreated"
    }
  ],
  "message": "Create BCI failed because the specified instance is out of stock",
  "reason":"ContainerInstanceScheduleFailed",
  "phase": "Pending"
}
```

自定义设置BCI Pod的最大Pending时长

Pending状态的BCI实例（BCI Pod）是不计费的，对于长时间处于Pending状态的异常Pod，如果您没有及时处理，可能会对业务产生异常影响。默认情况下，BCI实例的最大Pending时长为1小时，您可以根据实际业务情况自定义设置最大Pending时长，系统会自动终止超时的BCI实例，可以在一定程度上规避因没有及时处理异常Pod而造成业务异常影响。

功能说明 每个BCI实例相当于一个Pod。创建BCI Pod时，当对应的BCI实例进入等待创建（Pending）阶段，如果出现镜像拉取失败、Volume挂载失败等问题，BCI实例会一直处于Pending状态，但不会计费，您需要及时处理这类异常的CI实例来避免对业务产生异常影响。默认情况下，对于Pending时长超出1小时的BCI实例，系统会自动终止。如果您对于时长有要求，可以自定义设置最大Pending时长。

配置说明 您可以在Pod metadata中添加 bci.virtual-kubelet.io/bci-max-pending-minute 的Annotation来自定义设置Pod对应BCI实例的最大Pending时长。该Annotation的相关说明如下：

- 该Annotation的取值范围为10~1440的整数，单位为分钟，即下限为10分钟，上限为1天。

> **重要**：请根据实际业务情况合理设置最大Pending时长。如果镜像过大且没有镜像缓存的情况下，创建Pod时需要一定的时长来拉取镜像，此时如果最大Pending时长设置过短，可能会导致Pod创建失败。

- 未添加该Annotation的情况下，默认的最大Pending时长为1小时。

配置示例 创建一个设置了最大Pending时长的BCI 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
      annotations:
        bci.virtual-kubelet.io/bci-max-pending-minute: "30"   # 设置最大Pending时长为30分钟
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

自定义设置BCI Pod的hosts

某些场景下，您可能需要自定义设置BCI Pod的hosts，例如拉取自建镜像仓库的镜像时，需要通过hosts明确镜像仓库的实际IP地址。本文介绍如何自定义设置BCI Pod级别的hosts（即/etc/hosts）。

配置说明 您可以通过bci.virtual-kubelet.io/pod-host-aliases 的Annotation自定义设置BCI Pod的hosts，支持传入多组IP和域名的映射关系，格式为：

```
  [{\"ip\":\"10.10.xx.xx\",\"hostnames\":[\"example.com\"]},{\"ip\":\"10.10.yy.yy\",\"hostnames\":
[\"foo.com\",\"bar.com\"]}]
```

> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例 例如拉取自建镜像仓库的镜像时，需要通过hosts明确镜像仓库的实际IP地址时，可参考以下YAML示例：

```
  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
      annotations:
        bci.virtual-kubelet.io/pod-host-aliases: "[{\"ip\":\"10.10.xx.xx\",\"hostnames\":[\"example.com\"]}]"
    spec:
      containers:
      - name: nginx
        image: example.com/test/nginx:1.7.9
        ports:
        - containerPort: 80
```

为BCI Pod绑定自定义标签

标签由一组键值对组成，可以用于标记BCI实例。您可以使用标签来分组管理BCI实例，便于筛选和批量操作。
本文介绍如何为BCI Pod绑定自定义标签（Tag），以便后续可以基于标签管理BCI Pod，例如基于标签进行费用分析。

背景信息 标签（Tag）是一组键值对，百度智能云提供标签管理功能，支持按照各种标准（如用途、所有者或项目）提供跨服务、跨区域对资源进行添加标签，从而快速分类和管理资源。

配置说明 您可以通过 bci.virtual-kubelet.io/resource-tag 的Annotation为BCI Pod绑定自定义标签，最多可以绑定10个标签。标签键和标签值之间用半角冒号隔开，多个标签之间用半角逗号隔开。

重要

- Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
- 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例如下：

```
  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
      annotations:
        bci.virtual-kubelet.io/resource-tag: "key1:value1,key2:value2"   #绑定标签
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

配置资源规整时忽略特定容器

创建BCI Pod（即BCI实例）时，如果指定的vCPU和内存规格不满足BCI规格要求，系统会在满足资源需求的同时自动向最接近的BCI规格进行资源规整，规整后的规格过大可能会造成一定程度的资源浪费。对于一些不影响业务的容器（例如Sidecar容器），可以为其设置对应的Annotation，实现资源规整时忽略该容器，以避免资源浪费，节约BCI使用成本。

功能说明 指定vCPU和内存创建BCI实例时，您可以自定义指定每个容器的vCPU和内存，但汇总到实例级别时，需满足BCI实例整体的vCPU和内存要求。如果实例级别没有配置vCPU和内存，则会汇总计算所有容器的规格之和，对于总和不满足BCI规格的情况，系统会自动进行资源规整。

配置说明 为BCI Pod设置以下Annotation来标记资源规整时忽略的容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-resource-ignore-containers: "container1,container2"
```

> **重要**
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例 按以下YAML创建Deployment，BCI Pod的规格为1 vCPU、2 GiB内存，即采用nginx容器的limits，忽略sidecar容器的limits。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
      annotations:
        bci.virtual-kubelet.io/bci-resource-ignore-containers: "sidecar"   # 声明资源规整时忽略的容器列表
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
      - name: sidecar
        image: sidecar:tag
        ports:
        - containerPort: 8080
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
      restartPolicy: Always
```

> **说明**
>
> 如果上述Yaml没有设置Annotation bci.virtual-kubelet.io/bci-resource-ignore-containers: "sidecar"，则汇总nginx容器和sidecar容器的资源需求为2 vCPU、4 GiB，创建该Deployment时，资源规整后实际创建的BCI Pod为2 vCPU、4 GiB 规格。

预留实例券

预留实例券概述

预留实例券是一种抵扣券，可以抵扣按量付费实例（不含抢占式实例）的账单，也能够预留实例资源。相比包年包月实例，预留实例券与按量付费实例这种组合模式可以兼顾灵活性和成本。

什么是预留实例券？

> 当前预留实例券功能处于内测中，如需使用，请提交工单申请。

定义 预留实例券是一种抵扣券,可以抵扣相关配置的后付费BCI实例账单，支持1个月、1年、2年、3年购买周期。 预留实例券相比于后付费BCI实例单价更低且支持资源预留。另外，预留实例券支持多种付费方式（全预付、部分预付、零预付），您可根据财务情况灵活选择。

功能 预留实例券分为可用区级预留实例券和地域级预留实例券。

1. 预留实例券生效后将自动匹配可抵扣的后付费BCI实例账单。更多信息请查看预留实例券与实例的匹配。
2. 支持资源预留。

> 可用区级资源预留：
> 可为所匹配的按量实例资源进行资源保留。
> 地域级资源预留：
> 地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

3. 支持抵扣预测。
   预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。更多信息请查看可抵扣的实例。 **付费方式** 预留实例券支持3种付费方式：
4. **全预付**：购买券时支付有效期内对应算力的总额，是3种付费方式中总额最低的；
5. **部分预付**：购买券时支付有效期内对应算力的一部分费用，剩下的费用将在有效期内的每小时支付。总支付金额=半预付金额+每小时分期费用*购买周期内小时数；
6. **零预付**：购买券时无需支付，费用将在有效期内的每小时支付。总支付金额=每小时分期费用*购买周期内小时数。

> 预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。
> 零预付根据使用情况自动开放，如果您需要使用，请提交工单申请。

相较于后付费模式，预留实例券可以有效的降低长时间运行实例的费用成本。

	预留实例券	**后付费**
定义	一种抵扣券，用来抵扣后付费BCI实例账单。	购买BCI时的一种付费方式（先使用后付费），分钟计费，小时出账。
特点	1. **价格优惠**：总成本比后付费实例低； 2. **资源保障**：支持资源预留，保证您创建成功与预留实例券匹配的实例； 3. **灵活抵扣**：实例券不与某个实例绑定，一张实例券可抵扣同规格的不同后付费BCI账单； 4. **灵活支付**：支持多种付费方式（全预付、部分预付、零预付），避免一次性支付带来的资金链压力。	1.即用即买，随时释放； 2.单价较高

资源抵扣 预留实例券生效后将自动匹配满足条件的实例规格(后付费)进行抵扣。
其中抵扣是指相同实例规格的后付费BCI的账单，而不是后付费BCI的费用。

> - 如果您希望对已有的BCI实例进行成本优化，可以筛选出指定规格创建的BCI实例，并根据规格情况购买对应的预留实例券。

- 如果您还没有创建BCI实例，请根据业务需求选择合适的规格，然后购买对应的预留实例券，并指定规格创建BCI实例。
- 如果您已经购买了预留实例券，则在创建BCI实例时，必须要指定预留实例券对应的规格，否则预留实例券无法抵扣。

到期自动购买预留实例券 预留实例券支持自动续费。如果您是按月购买，则自动续费周期为1个月。按年购买，则自动续费周期为1年。

购买预留实例券

本文介绍如何在BCI管理控制台购买预留实例券。

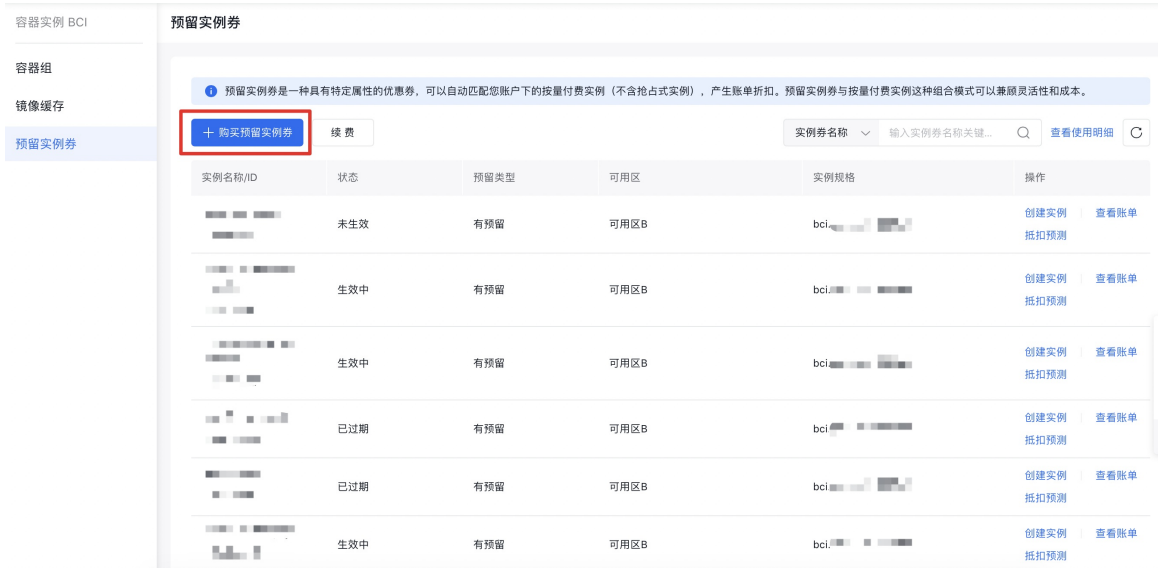
> 说明
> 预留实例券支持CPU和GPU实例规格。

前提条件

- 购买预留实例券前，请确保您要匹配的按量付费实例符合预留实例券使用要求。详情请参见预留实例券概述
- 您无法手动管理预留实例券和按量付费实例的匹配状态，请确保您已了解预留实例券匹配规则。详情请参见预留实例券与实例的匹配。

操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。
3. 单击**购买预留实例券**。



4. 配置地域信息。
   a. 选择资源预留类型。

> 可用区级资源预留： 可为所匹配的按量实例资源进行资源保留。
> 地域级资源预留： 地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。
> 详情请查看预留实例券与实例匹配。

b. 选择地域。
c. 选择可用区。

5. 配置实例信息。
   a. 选择实例规格。
   b. 选择付费类型。支持**全预付**、**部分预付**和**零预付**。

> 预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。
> 零预付根据使用情况自动开放，如果您需要使用，请提交工单申请。

6. 配置购买参数。
   a. 填写券名：可选。
   b. 填写购买数量。 可以同时匹配同规格按量付费实例的数量。

> 最多购买10台，如需购买更多，请提提交工单申请。

c. 选择预留实例券有效期。支持**1个月、1年、2年和3年**。

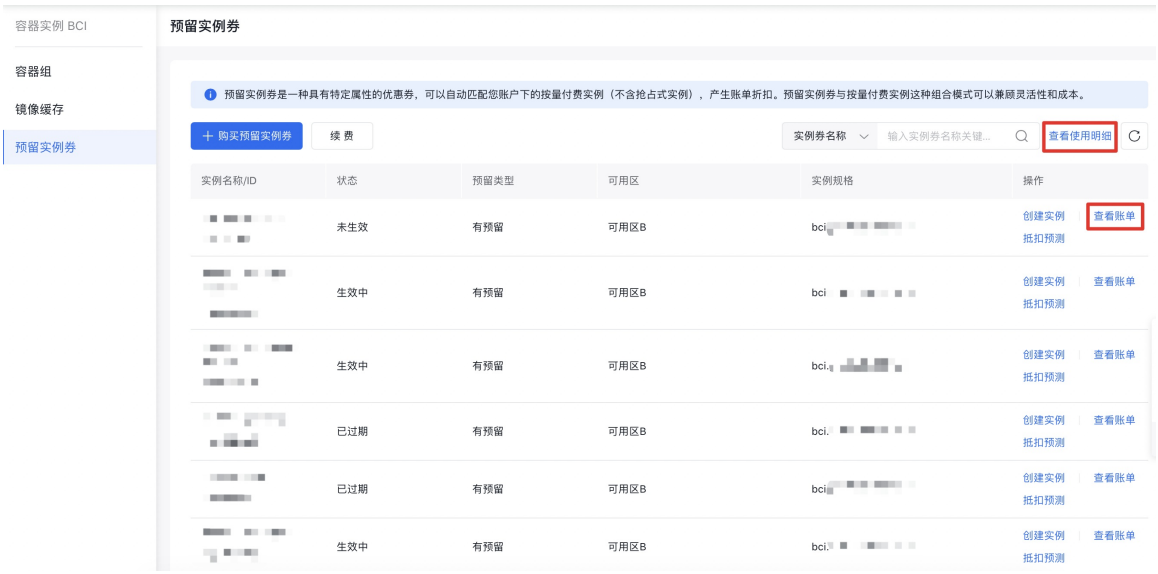
7. 选择是否启动自动续费。

> 按月购买，则自动续费周期为 1 个月。按年购买，则自动续费周期为 1 年。

8. 选择生效时间。目前支持**指定时间生效**。
9. 确认参数，然后单击**提交**。
10. 确认支付信息，然后单击**确认支付**。
    执行结果 您已经成功购买了一张预留实例券，成功匹配按量付费实例后即可抵扣按量付费实例账单。

查看预留实例券使用明细

您可以在预留实例券页面点击**查看使用明细**或**查看账单**，确认券的抵扣情况。



> 重要
> 在配置了多可用区创建BCI实例的场景下，创建的BCI实例可能会发布在多个可用区。如果您发现预留实例券未按预期进行抵扣，请检查预留实例券所在可用区和BCI实例可用区是否一致。

在**抵扣明细**页面，您可以查看预留实例券的抵扣情况。预留实例券的使用明细将记录每个出账周期（每小时）该预留实例券抵扣的实例信息。其中，抵扣时长对应的是计算力*小时（预留实例券按计算力进行抵扣，1个计算力可以理解为1vCPU）。

预留实例券与实例的匹配

购买预留实例券即代表承诺使用一定时长的按量付费实例，预留实例券匹配按量付费实例后才能抵扣账单。如果您的账号下暂时没有可以匹配的按量付费实例，预留实例券会闲置但继续计费。

本章节为您介绍预留实例券匹配按量付费实例的规则和示例。 **匹配规则** 您不能手动匹配预留实例券和按量付费实例。购买预留实例券后，在有效期内预留实例券将自动匹配满足条件的按量付费实例。匹配成功后，预留实例券每小时检查可抵扣的按量付费账单，并按券面的计算力抵扣账单。您可以使用抵扣预测功能查看预留实例券可以匹配的实例，具体操作请参见查看可抵扣的实例。

属性	地域和可用区	实例规格
可用区级预留实例券	只可匹配同一可用区中的按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 支持指定可用区的资源预留，在有效期内预留指定数量指定规格的实例，保证随时可以在指定可用区中成功创建按量付费实例。
地域级预留实例券	支持可用区灵活性，在指定地域中可以跨可用区匹配按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

可用区级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域和可用区相同。
- 实例规格和实例大小相同。

使用可用区级预留实例券的示例如下表所示。

示例场景	按量付费实例	可用区级预留实例券	抵扣效果
匹配成功	持有5台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：5台	预留实例券和5台按量付费实例完全匹配，1张预留实例券每小时抵扣5台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.c8m36.1a10实例，保证随时可以在华北-保定可用区B中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例 1台配置如下： 华北-北京，可用区A bci.c8m36.1a10 1台配置如下： 华北-保定，可用区B bci.c18m74.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：2台	导致匹配失败的原因如下： 1台实例可用区为华北-北京，可用区A； 1台实例规格为bci.c18m74.1a10； 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

地域级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域相同，支持跨可用区匹配实例。
- 实例规格和实例大小相同。

使用地域级预留实例券的示例如下表所示。

示例场景	按量付费实例	地域级预留实例券	抵扣效果
匹配成功	持有6台按量付费实例，配置如下： 3台配置如下： 华北-保定，可用区A bci.0.25c0.5m 3台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：6台	预留实例券和6台按量付费实例完全匹配，1张预留实例券每小时抵扣6台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.0.25c0.5m实例，保证随时可以在华北-保定地域中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 12台配置如下： 华北-保定，可用区A bci.0.25c0.5m 8台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例，配置如下： 华北-北京，可用区A bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：2台	导致匹配失败的原因如下： 2台实例可用区为华北-北京，可用区A 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

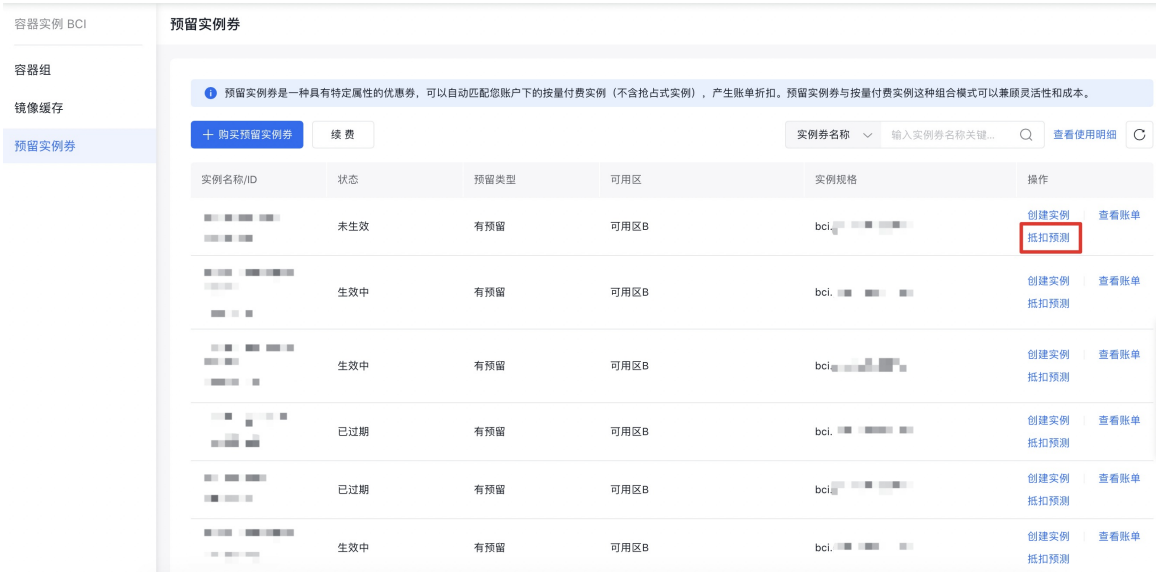
查看可抵扣的实例

预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。通过抵扣预测功能查看到实例仅代表实例符合匹配要求，并不代表一定抵扣这些实例的账单，实际抵扣情况以账单为准。

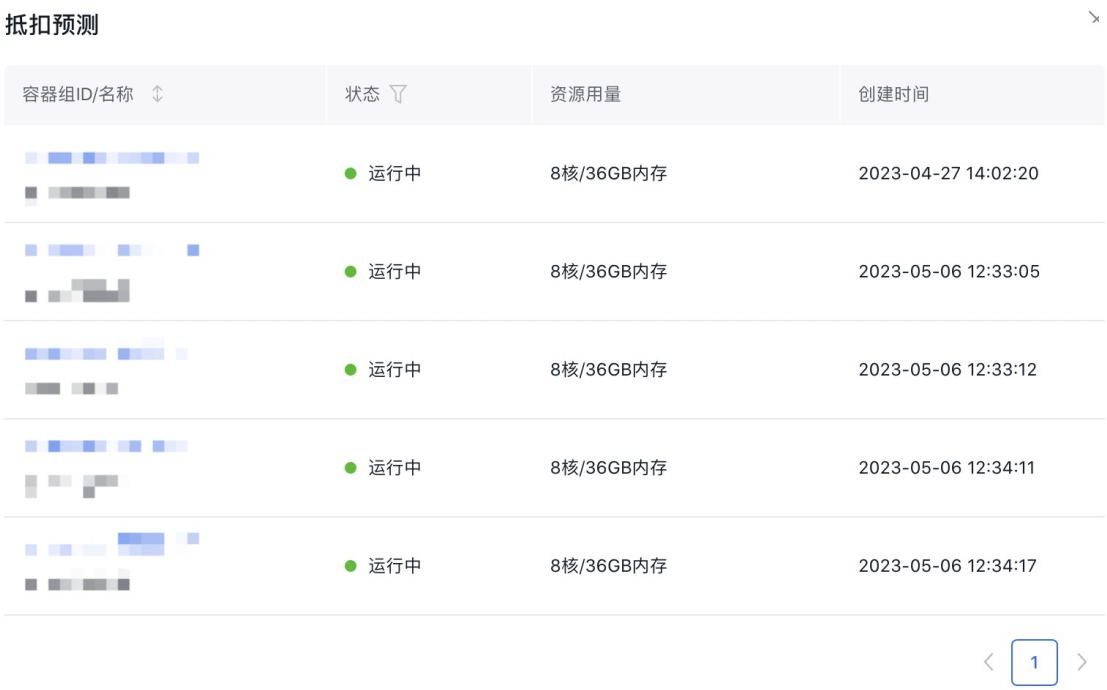
操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。

3. 在**预留实例券**页面，在操作栏单击**抵扣预测**。



4. 在抵扣预测页面查看预留实例券可以匹配并抵扣账单的按量付费实例。



镜像管理

使用CCR镜像仓库

背景信息 CCR镜像仓库分为：个人版，企业版，建议使用企业版镜像仓库

个人版：容器镜像服务个人版主要面向个人开发者或企业客户临时测试使用，提供基础的容器镜像托管、分发等能力，同时在资源配额上有一定的限制，不推荐。

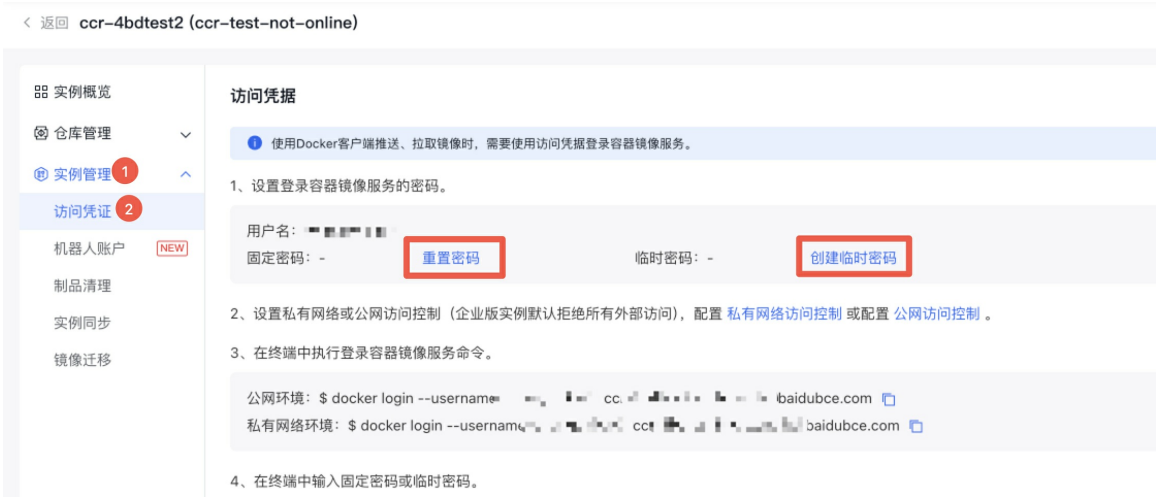
企业版：容器镜像服务企业版主要面向企业用户业务生产使用，提供容器镜像、Helm Chart等云原生制品生命周期管理，支持多地域、多场景的制品高效分发，同时企业版会不断完善其功能特性并持续更新，帮助企业在业务生产过程中降本增效。

更多信息，请参考容器镜像服务CCR。

前提条件 需要创建好CCR镜像仓库，此处以企业版镜像仓库为例。

创建命名空间：可创建公有命名空间或私有命名空间，私有命名空间需要登录密钥才可访问。

若使用私有命名空间，需要设置访问凭据。



使用CCR镜像仓库

操作步骤

1.确定CCR企业版的网络访问方式：

- 通过公网访问，配置公网访问控制
- 通过VPC内访问，配置私有网络访问控制

在配置好访问控制后，使用响应的镜像仓库地址创建BCI实例即可

2.（可选）如果您使用私用仓库，需要需要填写访问密钥，BCI只支持通过ImagePullSecrets字段指定访问密钥。

- 创建访问秘钥

```
kubectl create secret docker-registry <name> --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

- 在POD中使用密钥

```
  apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey
```

使用第三方镜像仓库

使用第三方镜像仓库

BCI支持使用第三方镜像仓库，例如：自建镜像仓库，dockerhub及其他云厂商等第三方镜像仓库。

> **注意：**
>
> 访问第三方镜像仓库会产生额外的公网流量费用，请确定BCI实例的EIP功能已打开

操作步骤

1. BCI支持HTTP、HTTPS协议，并已经默认跳过TLS CA认证。

2. （可选）如选择访问私有镜像仓库，需要使用secret进行登录：

- 创建secret

```
  kubectl create secret docker-registry <name> --docker-server=DOCKER_REGISTRY_SERVER --docker-
username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

相关参数说明

创建Secrect 参数	说明
name	secret的名称
DOCKER_REGISTRY_SERVER	镜像仓库地址
DOCKER_USER	镜像仓库用户名
DOCKER_PASSWORD	镜像仓库登录密码
DOCKER_EMAIL	邮箱

- 使用secret示例

```
  apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1  #镜像地址
  imagePullSecrets:  #镜像仓库是public类型无需secret
    - name: myregistrykey  #创建的secret名称，为上面name的值
```

使用镜像缓存

镜像缓存概述

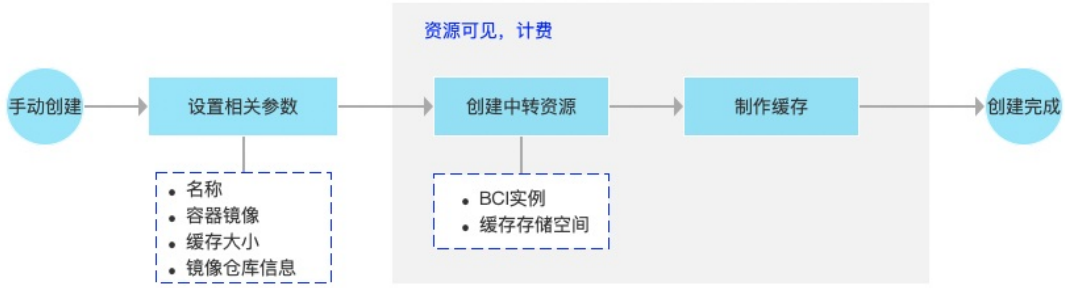
使用镜像缓存（ImageCache）创建BCI实例可以加速拉取镜像，减少BCI实例的启动耗时。本文介绍镜像缓存的基本功能、创建方式和镜像缓存状态查询。

功能简介 在运行容器前，BCI需要先拉取您指定的容器镜像，但因网络和容器镜像大小等因素，镜像拉取耗时往往成了BCI实例启动的主要耗时。为加速实例的创建速度，BCI提供镜像缓存功能。您可以预先将需要使用的镜像制作成缓存快照，然后基于该快照来创建BCI实例，避免或者减少镜像层的下载，从而提升实例的创建速度。

具体提升速度由BCI实例中使用的镜像个数、镜像大小和镜像仓库网络因素等决定。

创建方式 镜像缓存的创建分为 **手动创建** 和 **自动创建** 两种方式：
推荐使用自动创建的镜像缓存，可以节约使用成本。但对于首次创建就需要加速创建BCI实例的场景，需提前手动创建镜像缓存。

- 手动创建
  您可以根据需要，自行设置名称、大小等参数来创建镜像缓存。过程如下：



每个BCI实例默认有20 GiB的临时存储空间可用于创建镜像缓存，如果您的镜像大小超出了20 GiB，需要额外声明临时存储空间。

手动创建和自动创建的镜像缓存在名称、使用成本方面均有差异，对比如下表所示。

对比项	手动创建	自动创建
镜像缓存名称	可以自定义设置名称。	名称由系统自动生成，格式为 `auto-create-for-实例ID` 。
缓存大小	默认为20 GiB，如超过20GiB需要发工单调整配额。	
保留时长	由百度云BCI管理保留时长。如果在30天内未使用，将被自动删除。	
淘汰策略	当数量达到配额限制时，系统会遵循LRU规则，自动删除最不常用的镜像缓存。	
使用成本	收取创建费用和使用费用。	仅收取使用费用，不收取创建费用。

使用方式 创建BCI实例时，使用镜像缓存可以加快BCI实例的创建。目前只支持自动匹配镜像缓存方式：

- 自动匹配：自动匹配使用最优的镜像缓存。目前只支持完全匹配策略。
- 完全匹配策略：即镜像名称及版本是否完全相同。 *注意1：镜像tag不能为latest，必须指定一个明确的tag。否则，镜像缓存不会生效。* 注意2：如果用户修改了镜像，必须发布一个新的tag。否则，使用的缓存会比较老。 **注意事项**
- 创建镜像缓存需要拉取容器镜像，因此创建时长由镜像个数、镜像大小、网络等多种因素决定。
- 自动创建镜像缓存时采用实例中所声明的容器镜像。
  - 如果镜像为私有镜像，则需要提供私有镜像仓库的访问凭证，包括地址、用户名和密码。
  - 如果镜像需要通过公网拉取（如Docker官方镜像），则需要配置EIP或者NAT来访问公网。更多信息，请参见 连接公网。
  - 如果镜像由于远程仓库超时等原因导致拉取失败，推荐您将镜像仓库和VPC打通，或者可以使用容器镜像服务CCR，

将镜像上传至百度云仓库。

计费说明

- 创建镜像缓存
  - 自动创建镜像缓存，无需付费。
    如果您的镜像需要通过公网拉取，则会产生相应的公网流量费用。
- 使用镜像缓存
  - 使用自动创建的镜像缓存创建BCI实例时，如果镜像缓存大于20GiB，请提前发工单联系，以避免空间不足导致失败。
  - BCI后续将增加临时存储空间计费项，超过20GiB需付费，否则只需支付BCI实例费用。

查询镜像缓存

查询镜像缓存 镜像缓存状态查询 创建镜像缓存后，您可以在镜像缓存页面查看创建结果。
状态列显示创建进度，当镜像缓存状态为 **创建成功** 时，可以使用该镜像缓存。

镜像缓存ID/名称	状态	镜像信息	创建时间	最近匹配时间
2e1be1eb99e946c3a543ec5a4eaa7d3940adcb8335294f9935e1ae4ffc8e86d8	创建成功 (100%)		2023-04-18 19:03:48	2023-04-18 19:04:22
2e1be1eb99e946c3a543ec5a4eaa7d39c38fe0d216de7ca1e8c19871b05da9cd	创建成功 (100%)		2023-04-18 19:25:07	2023-04-18 19:26:09
2e1be1eb99e946c3a543ec5a4eaa7d392d0e9f431ca7251866386333359aaa367	创建成功 (100%)		2023-04-18 19:30:17	2023-04-18 19:37:30
2e1be1eb99e946c3a543ec5a4eaa7d39faf9d193a03ed5d7ce24d43a0f55e218	创建成功 (100%)		2023-04-18 19:45:07	2023-04-18 19:46:09
2e1be1eb99e946c3a543ec5a4eaa7d395561695e79ea52e026cd7c5c435141bd	创建失败(0%)		2023-04-19 10:22:54	2023-04-19 18:24:09

网络管理

连接公网

连接公网

如果您的BCI实例（即BCI Pod）有连接公网的需求，则需要配置NAT网关或者弹性公网IP，并支付相应的网络费用。

本文介绍如何为您的BCI实例绑定EIP，或者为BCI实例所属的VPC绑定NAT网关，以实现BCI实例与公网互通。

背景信息

为BCI实例配置公网服务时，支持以下两种方式：

方式	说明	费用
BCI实例绑定EIP	EIP是独立购买的可单独持有的公网IP地址，可以绑定到BCI实例上提供公网服务。具体详情请参见：百度云EIP介绍。	EIP支持按固定带宽或者按使用流量计费。
BCI实例所属的VPC绑定NAT网关	NAT网关是可独立购买的网关产品，绑定EIP后，可以为关联VPC下的所有BCI实例提供公网服务。具体详情请参见：百度云NAT网关介绍。	NAT网关支持包年包月和按量付费。NAT网关需绑定EIP后才能具备公网能力，即除NAT网关费用外，您还需支付EIP费用。

您可以根据业务需要，选择合适的方式来配置公网服务：

- **示例场景一：单个BCI实例配置Nginx外网访问**

如果您有一个BCI实例用于部署Nginx服务，在创建实例时，您需要为该实例绑定EIP。当Nginx启动时，将自动暴露80端口到EIP。您可以通过EIP地址加端口的方式访问该BCI实例的Nginx服务。

- **示例场景二：多个BCI实例拉取Docker Hub镜像**

BCI默认不提供外部公网链路进行公网镜像的拉取。如果您有多个BCI实例需要从Docker Hub拉取镜像，您可以为BCI实例所属

的VPC绑定NAT网关来实现公网访问，否则镜像将拉取失败。

> **注意：**
>
> 如果BCI实例已经绑定了EIP，则优先使用BCI实例绑定的EIP来访问公网，而不会使用NAT网关的SNAT功能访问公网。

操作指南

连接公网方式一：为BCI实例绑定EIP

创建BCI Pod时，您可以直接为Pod绑定EIP。支持在Pod metadata中添加Annotation来自动创建并绑定一个EIP。 相关Annotation如下：

Annotation	示例值	说明
bci.virtual-kubelet.io/bci-create-eip	"true"	是否自动创建并绑定EIP
bci.virtual-kubelet.io/bci-create-eip-route-type	"BGP"	设置EIP的线路类型。不指定时，默认为BGP类型。可选值： BGP：标准型BGP BGP_S：增强型BGP ChinaMobile：中国移动 ChinaUnicom：中国联通 ChinaTelcom：中国电信
bci.virtual-kubelet.io/bci-create-eip-bandwidth	"10"	设置EIP带宽。带宽能力和EIP线路类型相关。带宽区间： BGP：1～200 Mbps BGP_S：100～5000 Mbps 移动、联通、电信：1～5000 Mbps
bci.virtual-kubelet.io/bci-create-eip-paymethod	"ByBandwidth"	设置EIP计费方式： ByTraffic：流量 ByBandwidth：带宽
bci.virtual-kubelet.io/bci-eip-ip	"127.0.0.1"	绑定已有的EIP。且此EIP状态必须可用(Available)。 **限定条件**：一个EIP只能成功绑定到一个BCI实例上，若使用deployment方式，副本数replicas需设置为1，否则可能导致创建失败。

> **注意：**
>
> 如EIP的线路类型选择移动、联通、电信，需要先提交工单申请百度云EIP单线白名单，否则该BCI Pod会创建失败。
>
> 如EIP的线路类型选择增强BGP（BGP_S）且要按照流量付费，需要先提交工单申请百度云EIP计费白名单，否则该BCI Pod会创建失败。

配置示例：

- 示例一：使用默认参数自动创建EIP（默认线路类型为BGP，带宽为100Mbps）

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-create-eip": "true"  # 自动创建并绑定EIP
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

- 示例二：自动创建EIP，并设置EIP带宽和线路类型

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-create-eip": "true"              # 自动创建并绑定EIP
    "bci.virtual-kubelet.io/bci-create-eip-route-type": "BGP_S"  # 设置EIP线路类型
    "bci.virtual-kubelet.io/bci-create-eip-bandwidth": "200"     # 设置EIP带宽
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

- 示例三：绑定已有的EIP

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    "bci.virtual-kubelet.io/bci-eip-ip": "127.0.0.1"    # 指定已经存在的EIP
  name: test
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    name: nginx
    resources:
      requests:
        cpu: 0.5
        memory: 1Gi
  restartPolicy: Always
```

查询BCI实例绑定的EIP信息

如果BCI实例已成功绑定EIP，则该实例yaml信息的Annotations中将包含以下字段：

Annotation	示例值	说明
bci.virtual-kubelet.io/bci-bound-eip-id	"ip-xxxx"	EIP实例id
bci.virtual-kubelet.io/bci-bound-eip-ip	"127.0.0.1"	EIP公网地址
bci.virtual-kubelet.io/bci-bound-eip-bandwidth	"100"	EIP公网带宽
bci.virtual-kubelet.io/bci-bound-eip-paymethod	"ByTraffic"	EIP计费方式
bci.virtual-kubelet.io/bci-bound-eip-route-type	"BGP"	EIP线路类型

连接公网方式二：为VPC绑定NAT网关

您可以在专有网络控制台为VPC绑定NAT网关，并为NAT网关绑定EIP，使其能够提供NAT代理（SNAT和DNAT）功能：

- SNAT功能：可以为VPC中没有公网IP的BCI实例提供访问公网的代理服务。
- DNAT功能：可以将NAT网关绑定的EIP映射给VPC中的BCI实例使用，使其能够面向公网提供服务。

操作步骤如下：

1. 登录私有网络VPC控制台。
2. 在顶部菜单栏，选择地域。
3. 在左侧菜单栏，选择“网络连接”->“NAT网关”，进入NAT网关页面，创建NAT网关。
    1. 单击创建NAT网关。
    2. 完成购买NAT网关相关的参数配置。配置时，“所在网络”选择BCI实例所属的VPC。更多信息，请参见NAT网关。
    3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。
4. 在弹性公网IP控制台，创建EIP。
    1. 在顶部菜单栏，选择地域后，单击“创建实例”。

2. 完成购买EIP相关的参数配置。更多信息，请参见创建EIP实例。
3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。

5. 绑定EIP与NAT网关。
   1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“更多”->“绑定公网IP”。
   2. 在弹出的对话框中选择要绑定的EIP，然后单击确定。
6. 如果您的BCI实例需要访问公网，您需要为NAT网关创建SNAT条目，并在VPC中添加该NAT网关的路由。
   1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置SNAT”。
   2. 单击“添加SNAT条目”。
   3. 配置SNAT相关条目的参数，然后单击确定。配置时，“源网段”选择BCI所属的子网。更多信息，请参见NAT网关的配置SNAT表。
   4. 在私有网络页面，单击要操作的VPC名称进入VPC详情页，单击“路由表”，然后单击“添加路由”并配置路由参数。更多信息，请参见路由表的添加路由。
7. 如果您的BCI实例需要面向公网提供服务，您需要为NAT网关创建DNAT条目。
   1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置DNAT”。
   2. 单击“添加DNAT条目”。
   3. 配置DNAT相关条目的参数，然后单击确定。更多信息，请参见NAT网关的配置DNAT表。

支持自定义DNSConfig

为了兼容原生kubernetes Pod使用节点默认NameServers能力，CCE+BCI提供vk支持自定义DNSConfig的方案。 **1. VirtualKubelet DNSConfig 介绍**

DNSConfig 功能介绍参考：https://kubernetes.io/zh-cn/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-policy
在 BCI 中 对 DNSPolicy 和 DNSConfig 支持情况如下：

DNS策略	说明	备注
Default	Pod 从运行所在的节点继承名称解析配置。	
ClusterFirst	与配置的集群域后缀不匹配的任何 DNS 查询（例如 "www.kubernetes.io"） 都会由 DNS 服务器转发到上游名称服务器。集群管理员可能配置了额外的存根域和上游 DNS 服务器。	
ClusterFirstWithHostNet	对于以 hostNetwork 方式运行的 Pod，应将其 DNS 策略显式设置为 "ClusterFirstWithHostNet"。否则，以 hostNetwork 方式和 "ClusterFirst" 策略运行的 Pod 将会做出回退至 "Default" 策略的行为。	在 BCI 中，不支持使用 HostNetwork；所以不支持该 DNS 策略；
None	此设置允许 Pod 忽略 Kubernetes 环境中的 DNS 设置。Pod 会使用其 dnsConfig 字段所提供的 DNS 设置。	

2. 使用限制

- kubernetes 集群版本 >= 1.18
- VirtualKubelet 版本 >= 0.3.1 **3. 注意事项 建议用户自定义 Pod DNSConfig 时，NameServers 中建议仅包含一个 IP 地址。** 原因如下：
- Kubernetes 限制 DNSConfig 的 NameServers 中最多包含 3 个 IP；

- BCI 后端会默认在 Pod 的 DNSConfig 中注入一个 IP；
- 当 DNSConfig 为 ClusterFirst（需要使用您自己k8s集群的CoreDNS/Kube-DNS）并且开启kube-proxy，VirtualKubelet 会默认将集群的域名服务器 clusterIP 注入，占用一个 IP 地址；

4. 使用场景 4.1 使用 DNSPolicy 为 Default 该场景下，按照 Kubernetes 行为，预期为："Pod 从运行所在的节点继承名称解析配置"，即需要继承 **VKNode 的解析配置**。
在使用该功能之前，您需要先对 VirtualKubelet 完成配置，指定注册到 VKNode 中的 DNSConfig：

在 cce-virtual-kubelet 的 helm charts 中包含 dnsConfig 配置，默认为空：

```
dnsConfig: {}
```

您可以按照自己的期望配置，例如：

```
dnsConfig:
nameservers:
  - 169.254.20.10 # 这是一个示例，建议您只配置一个域名服务器ip，最多不超过两个
options:
  - name: ndots
    value: "3"
  - name: attempts
    value: "2"
  - name: timeout
    value: "1"
searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
```

在创建 Pod 时，您需要将 dnsPolicy 指定为 *Default* ，例如：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: default
  template:
    metadata:
      labels:
        app: default
    spec:
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Equal"
        value: "baidu"
        effect: "NoSchedule"
      containers:
      - name: main
        image: hub.baidubce.com/cce/nginx-alpine-go:latest
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
      dnsPolicy: Default # 指定 dnsPolicy 为 Default
```

此时 VirtualKubelet 将在 BCI 创建 Pod 时，继承您在 VirtualKubelet 中指定的 DNSConfig。

4.2 使用 DNSPolicy 为 None 该场景下，按照 Kubernetes 行为，预期为：“Pod 会使用其 dnsConfig 字段所提供的 DNS 设置”。您无需对 VirtualKubelet 做额外配置，需要在 Pod 的配置中填写dnsConfig字段：

```
  apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 192.0.2.1 # 这是一个示例，最多填写三个域名服务器ip
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
    options:
      - name: ndots
        value: "2"
```

此时 VirtualKubelet 将在 BCI 创建 Pod 时，将直接使用您在 Pod 中填写的 DNSConfig。

4.3 使用 DNSPolicy 为 ClusterFirst 该场景下，按照 Kubernetes 行为，预期为：“转发到上游名称服务器”。
在 BCI 场景下与标准 Kubernetes 有区别，需要在 Pod 中添加 annotation `bci.virtual-kubelet.io/kube-proxy-enabled: "true"`，VirtualKubelet 才会将集群的域名服务器 clusterIP 注入 DNSConfig。
参考：https://cloud.baidu.com/doc/CCE/s/Qkhfxcry7
示例 Pod：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: busybox
spec:
 selector:
  matchLabels:
   app: busybox
 replicas: 1
 template:
  metadata:
   labels:
    app: busybox
   annotations:
    bci.virtual-kubelet.io/kube-proxy-enabled: "true" # 开启 kube-proxy
  spec:
   dnsPolicy: "ClusterFirst"
   containers:
   - name: busybox
    image: busybox
    command: [ "/bin/sh", "-c", "sleep 3600" ]
```

VirtualKubelet 在创建 Pod 时，将注入如下 DNSConfig，用于实现 集群内 的服务发现：

```
dnsConfig:
 nameservers:
  - ${coredns clusterip}
 options:
  - name: ndots
    value: "5"
 searches:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local
```

配置BCI Pod所属安全组

概述 安全组是一种虚拟防火墙，具备状态检测和数据包过滤能力，用于在云端划分安全域。通过添加安全组规则，您可以控制安全组内BCI Pod（即BCI实例）的入流量和出流量。

安全组介绍 安全组是一个逻辑上的分组，由同一VPC内具有相同安全保护需求并相互信任的实例组成。通过添加安全组规则，安全组可以允许或拒绝安全组内BCI实例对公网或者私网的访问，以及管理是否放行来自公网或私网的访问请求。

安全组分为普通安全组和企业安全组。如果您对整体规模和运维效率有较高需求，建议您使用企业安全组。相比普通安全组，企业安全组大幅提升了组内支持容纳的实例数量，简化了规则配置方式。更多关于两种安全组的差异信息和具体操作，请参考安全组

> 注意：一个安全组可管理同一个VPC内多个BCI实例

指定安全组 在CCE集群中创建BCI Pod时，BCI Pod默认会加入到cce-virtual-kubelet组件安装时配置的安全组中。如果有特殊需求，您也可以为某些BCI Pod指定其他安全组。 **操作前提** 操作前，请先创建安全组，具体操作，请参见文档创建安全组。

> 创建成功的BCI Pod不支持修改所属安全组。如果想要变更安全组，需要重新创建BCI Pod。

BCI Pod配置 对于有特殊需求的某些BCI Pod，可以添加bci.virtual-kubelet.io/bci-security-group-id的Annotation来指定安全组。配置要求如下：

- 支持指定一个或多个安全组，最多可以指定5个安全组。
- 指定的安全组必须属于同一VPC。
- 指定的安全组的类型必须相同，即同为普通安全组或同为企业安全组。

> Annotation 请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，已成功的BCI Pod只能重建来生效

配置如下：

```
 apiVersion: v1
kind: Pod
metadata:
  name: test
  labels:
    app: nginx
  annotations:
    "bci.virtual-kubelet.io/bci-security-group-id": "g-xxxx" #配置安全组
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - name: sun-nginx-host
    imagePullPolicy: IfNotPresent
    image: hub.baidubce.com/jpaas-public/nginx-alpine-go:latest
    resources:
      limits:
        cpu: 0.5
        memory: 1Gi
      requests:
        cpu: 0.5
        memory: 1Gi
```

添加安全组规则 对于安全组内的BCI实例，您可以添加安全组规则来控制其出入流量。例如：

- 当BCI实例需要与所在安全组之外的网络进行通信时，您可以添加允许访问的安全组规则，实现网络互通。
- 在运行BCI实例的过程中，发现部分请求来源有恶意攻击行为时，您可以添加拒绝访问的安全组规则，实现网络隔离。

关于如何添加安全组规则，请参见编辑安全组

配置BCI Pod访问集群内Service

概述

BCI支持从BCI实例中访问K8S ClusterIP类型Service。如果您需要在BCI实例中访问集群中ClusterIP, 您需要在Pod Spec Yaml中添加如下Annotation，用于开启功能:

`bci.virtual-kubelet.io/kube-proxy-enabled: true`

> Annotation 请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，已成功的BCI Pod只能重建来生效。

配置示例 以创建一个简单的Nginx为例，Nginx的Pod 会调度到VNode上：

```
 kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - name: nginx-port
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-bci
  labels:
    app: nginx
  annotations:
    "bci.virtual-kubelet.io/kube-proxy-enabled": "true" #添加这个annotation
spec:
  nodeSelector:
    type: "virtual-kubelet"
  tolerations:
  - key: "virtual-kubelet.io/provider"
    operator: "Equal"
    value: "baidu"
    effect: "NoSchedule"
  containers:
  - name: sun-nginx-host
    imagePullPolicy: IfNotPresent
    image: hub.baidubce.com/jpaas-public/nginx-alpine-go:latest
    resources:
      limits:
        cpu: 0.5
        memory: 1Gi
      requests:
        cpu: 0.5
        memory: 1Gi
```

存储管理

挂载CFS文件存储

挂载CFS文件存储

文件存储CFS(Cloud File Storage)是百度智能云提供的安全、可扩展的文件存储服务。通过标准的文件访问协议，为云上的计算资源提供无限扩展、高可靠、全球共享的文件存储能力。本文为您介绍挂载CFS文件存储。更多CFS相关信息，请参见CFS说明。

前提条件

请确保您已经创建CFS并已获得CFS挂载地址（CFS Server）。

> **注意：**
>
> - CFS为共享存储，一个CFS可以挂载到多个BCI实例上。此时，如果多个BCI同时修改相同数据，请进行同步与冲突保护。
> - 在删除所有使用此挂载点的BCI实例前，请勿删除CFS挂载点，否则可能会造成操作系统无响应。

配置示例

CFS Volume

```
 apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: registry.k8s.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /my-cfs-data
      name: test-volume
  volumes:
  - name: test-volume
    nfs:
      server: my-cfs-server.example.com #此为cfs挂载点
      path: / #此为server端目录，cfs中为根目录不可变更
      readOnly: true
```

CFS PVC

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Mi
  volumeName: cfs
```

CFS PV

```
 apiVersion: v1
kind: PersistentVolume
metadata:
 name: cfs
spec:
 capacity:
  storage: 1Mi
 accessModes:
  - ReadWriteMany
 nfs:
  server: cfs-server.default.svc.cluster.local  #此为cfs挂载点
  path: "/" #此为server端目录，cfs中为根目录不可变更
 mountOptions: #当前不支持指定挂载选项
  - cfsvers=4.2
```

使用CFS PVC的Deployment

```
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: cfs-web
spec:
 replicas: 2
 selector:
  matchLabels:
   role: web-frontend
 template:
  metadata:
   labels:
    role: web-frontend
  spec:
   containers:
   - name: web
    image: nginx
    ports:
     - name: web
      containerPort: 80
    volumeMounts:
        # name must match the volume name below
        - name: cfs
          mountPath: "/usr/share/nginx/html"
   volumes:
   - name: cfs
    persistentVolumeClaim:
     claimName: cfs
```

挂载EmptyDir数据卷

挂载EmptyDir数据卷

本文介绍如何挂载EmptyDir数据卷。EmptyDir数据卷是一个空的目录，用于临时存放数据，便于容器之间共享数据。

注意事项

EmptyDir为临时存储，当BCI实例删除或重启时，EmptyDir数据卷中保存的数据均会被清空。

> **提示：**
>
> 注意：当前暂不支持指定临时存储空间大小，不支持基于内存的临时存储。

操作步骤

1.声明数据卷

可在 Pod 中 Volume 字段对 emptyDir 类型的数据卷进行声明。

```
  volumes:
- name: cache-volume
  emptyDir: {}
```

2.挂载数据卷

声明数据卷后，可以通过VolumeMount相关参数将数据卷挂载到容器中。

```
 volumeMounts:
- mountPath: /cache
  name: cache-volume
```

3.完整配置示例

使用vk创建pod参数示例如下

```
 apiVersion: v1
kind: Pod
metadata:
 name: test-pd
spec:
 containers:
 - image: registry.k8s.io/test-webserver
   name: test-container
   volumeMounts:
   - mountPath: /cache  #容器内挂载路径
     name: cache-volume
 volumes:
 - name: cache-volume
   emptyDir:{} #默认为文件型，使用节点的存储空间#
```

挂载ConfigMap数据卷

挂载ConfigMap数据卷

本文介绍如何挂载ConfigMap数据卷。ConfigMap数据卷是一个配置文件，用于向BCI实例注入配置数据，具体详见create-a-configmap，同时BCI已支持ConfigMap配置文件的动态更新，即在K8S集群内更新ConfigMap配置文件内容，BCI Pod使用的该配置文件会自动更新。

配置示例

1.声明configMap

```
  apiVersion: v1
kind: ConfigMap
metadata:
  name: game-config
  namespace: default
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
```

2.使用vk提交pod

```
  apiVersion: v1
kind: Pod
metadata:
  annotations:
    myannotation: "myannotation"
  labels:
    app: bci-test-vk
    mylabel: "mylabel"
  name: config-file-test
  namespace: default
spec:
  nodeSelector:
    type: virtual-kubelet
  tolerations:
    - effect: NoSchedule
      key: virtual-kubelet.io/provider
      operator: Equal
      value: baidu
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
  containers:
  - image: hub.baidubce.com/cce/nginx-alpine-go
    imagePullPolicy: IfNotPresent
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: game-config
```

3.查看pod中已挂载的ConfigMap

查看挂载目录中文件
~# kubectl exec -it config-file-test -- ls /etc/config
game.properties
ui.properties

查看挂载目录中文件内容
~# kubectl exec -it config-file-test -- cat /etc/config/game.properties
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30

挂载PFS并行文件存储

PFS 概述
并行文件存储服务PFS (Parallel Filesystem Service)，是百度智能云提供的完全托管、简单可扩展的并行文件存储系统，针对高性能计算场景提供亚毫秒级的访问能力和高IOPS的数据读写请求能力。同时，百度智能云PFS提供简单、易操作的接口，免去部署、维护费用的同时，最大化提升您的业务效率。更多信息参见[PFS说明](https://cloud.baidu.com/product/pfs.html)。
> 注意：
> 挂载PFS并行文件存储功能正在内测中，如需使用可[提交工单](https://ticket.bce.baidu.com/?_=1681454984002/#/ticket/create?productId=189)申请。

提前准备
创建PFS实例
1. 创建PFS，操作步骤请参考[创建并行文件系统](https://cloud.baidu.com/doc/PFS/s/fks8xeoud)。
> 注意：
> PFS实例须与容器集群处于同一VPC内。
2. 从页面上获取PFS连接地址，以备后续部署存储插件时使用。
![PFS1.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/BCI/PFS1_1b9a98c.png)

配置示例
动态挂载
1. 创建PVC，注意storageClassName需要指定为上述安装插件时填写的storageClass名称。

按需指定PVC存储空间

$ cat >pfs-pvc.yaml <<EOF kind: PersistentVolumeClaim apiVersion: v1 metadata: name: csi-pvc-pfs namespace: default spec: accessModes: - ReadWriteMany storageClassName: pfs-sc resources: requests: storage: 50Gi # 用户指定PVC存储空间 EOF $ kubectl create -f pfs-pvc.yaml

2. 检查PVC状态为Bound。

$ kubectl get pvc csi-pvc-pfs NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE csi-pvc-pfs Bound pvc-1ab36e4d1d2711e9 50Gi RWX pfs-sc 4s

3. 在 Pod 中挂载PVC，要求 Pod 和 PVC 在同一个 namespace 下。

apiVersion: v1 kind: Pod metadata: name: test-pvc-pod namespace: default labels: app: test-pvc-pod spec: containers:

- name: test-pvc-pod image: nginx volumeMounts: - name: pfs-pvc mountPath: "/pfs-volume" volumes:
    - name: pfs-pvc persistentVolumeClaim: claimName: csi-pvc-pfs

静态挂载
1. 创建静态PV和PVC。
通过 pv.spec.csi.volumeAttributes.path 指定需要挂载的路径，该路径为相对于安装插件时填写的parentDir路径的相对路径。
> * 注意storageClassName需要指定为上述安装插件时填写的storageClass不同的名称，防止被动态创建流程接管。不要求该storageClass在集群中存在，只要PV和PVC的storageClassName一致即可，否则PV和PVC无法互相绑定。
> * 例如，按照下述示例yaml创建，则该PV在pfs中的路径为 /kubernetes/exist/some/dir。如果对应的路径不存在，在挂载PV时会自动创建该路径。

apiVersion: v1 kind: PersistentVolume metadata: name: static-pv-pfs spec: accessModes:

- ReadWriteMany storageClassName: pfs-static-sc capacity: storage: 100Gi csi: driver: csi-clusterfileplugin volumeHandle: data-id volumeAttributes: # the path is the related path to parentDir path: exist/some/dir

---

kind: PersistentVolumeClaim apiVersion: v1 metadata: name: static-pvc-pfs namespace: default spec: accessModes: - ReadWriteMany storageClassName: pfs-static-sc resources: requests: storage: 100Gi

2. 检查PVC状态为Bound，并且与对应的PV绑定。

$ kubectl get pvc NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE static-pvc-pfs Bound static-pv-pfs 100Gi RWX pfs-static-sc 10s

3. 在 Pod 中挂载PVC，要求 Pod 和 PVC 在同一个 namespace 下。

apiVersion: v1 kind: Pod metadata: name: test-pvc-pod namespace: default labels: app: test-pvc-pod spec: containers:

- name: test-pvc-pod image: nginx volumeMounts: - name: pfs-pvc mountPath: "/pfs-volume" volumes:
    - name: pfs-pvc persistentVolumeClaim: claimName: static-pvc-pfs

容器配置

设置容器启动命令和参数

设置容器启动命令和参数

BCI实例（即BCI Pod）通过容器镜像中的预设参数来启动容器。如果您在构建镜像时没有设置启动命令和参数，或者想要变更启动命令和参数，可以在创建BCI Pod时设置。本文介绍如何为容器设置启动时要执行的命令和参数。

功能说明

如果您想覆盖镜像中设置的启动默认值，包括工作目录、启动命令和参数，可以通过以下参数进行配置：
* 工作目录镜像构建时，通过WORKDIR可以指定容器的工作目录， 容器启动时执行的命令会在该目录下执行。更多信息，请参见[WORKDIR](https://docs.docker.com/engine/reference/builder/#workdir)。创建BCI实例时，通过配置BCI实例中容器的工作目录(WorkingDir)，可以覆盖WORKDIR。

> **注意：**
>
> 如果镜像里未指定WORKDIR，且创建BCI实例也未配置工作目录，则工作目录默认为根目录。
> 如果指定的工作目录不存在，系统将自动创建。
>

* 启动命令和参数镜像构建时，通过ENTRYPOINT和CMD可以指定启动容器后要执行的命令和参数。更多信息，请参见ENTRYPOINT和CMD。创建BCI实例时，通过配置BCI实例中容器的启动命令（Command）和参数（Arg），可以覆盖[ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint)和[CMD](https://docs.docker.com/engine/reference/builder/#cmd)。具体生效规则如下：

镜像ENTRYPOINT	镜像CMD	容器command	容器Arg	最终执行命令	说明
ls	/root/data	未设置	未设置	ls /root/data	容器的Command和Arg均未设置，则执行镜像ENTRYPOINT和CMD
ls	/root/data	cd	未设置	cd	容器设置了Command，未设置Arg，则只执行Command，忽略镜像ENTRYPOINT和CMD
ls	/root/data	未设置	/home/work	ls /home/work	容器设置了Arg，未设置Command，则执行镜像ENTRYPOINT和容器Arg
ls	/root/data	cd	/home/work	cd /home/work	同时设置了Command和Arg，则执行容器Command和Arg

> **注意：**
>
> 启动命令必须为容器镜像支持的命令，否则会导致容器启动失败。
>

配置示例

您可以通过容器的workingDir、command和args字段来设置工作目录、启动命令和参数。更多信息，请参见[为容器设置启动时要执行的命令和参数](https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/)。

配置示例如下：

apiVersion: apps/v1 kind: Deployment metadata: name: test labels: app: test spec: replicas: 2 selector: matchLabels: app: nginx template: metadata: name: nginx-test labels: app: nginx spec: containers: - name: nginx image: hub.baidubce.com/cce/nginx-alpine-go workingDir: /work command: ["printenv"] args: ["HOSTNAME", "KUBERNETES_PORT"] ports: - containerPort: 80

使用探针对容器进行健康检查

使用探针对容器进行健康检查

Kubernetes中，容器的健康检查由kubelet定期执行，kubelet通过存活探针和业务探针来检查容器的状态和运行情况。当前BCI支持探针如下：

探针	说明	使用场景
应用存活探针（ Liveness Probe）	用于检查容器是否正常运行。如果检查成功，则表示容器正常运行。如果检查失败，系统会根据配置的容器重启策略进行相应的处理。如果未配置该探针，则默认容器一直正常运行。	* 当应用程序处于运行状态但无法进行进一步操作时，Liveness Probe将捕获到deadlock，重启对应的容器，使得应用程序在存在bug的情况下依然能够运行。另外，长时间运行的应用程序最终可能会转换到broken状态，此时除了重新启动，无法恢复。Liveness Probe可以检测并补救这种情况。
应用业务探针（Readiness Probe）	用于检查容器是否已经就绪，可以为请求提供服务。如果检查成功，则表示容器已经准备就绪，可以接收业务请求。如果检查失败，则表示容器没有准备就绪，系统将停止向该容器发送任何请求，直至重新检查成功。	如果应用程序暂时无法对外部流量提供服务，例如应用程序需要在启动期间加载大量数据或配置文件，此时，如果不想终止应用程序，也不想向其发送请求，可以通过Readiness Probe来检测和缓解这种情况。
应用启动探针 (StartupProbe Probe)	用于检查容器是否启动成功，有时候，会有一些现有的应用在启动时需要较长的初始化时间。 要这种情况下，若要不影响对死锁作出快速响应的探测，设置存活探测参数是要技巧的。	比如服务A启动时间很慢，需要60s。这个时候如果还是用存活探针就会进入死循环，因为当存活探针开始探测时，服务并没有起来，发现探测失败就会触发restartPolicy。此时简单调大存活探针的initialDelay时间不灵活且可能带来更多问题，因此可以通过应用启动探针来环境这种情况。

配置示例

您可以通过容器的livenessProbe和readinessProbe字段来设置Liveness Probe或者Readiness Probe，配置详情见[配置存活、就绪和启动探针](https://kubernetes.io/zh-cn/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/)

1.配置应用存活探针（ Liveness Probe）

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: liveness-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi livenessProbe: exec: command: - /bin/sh - '-c' - sleep 1 && exit 0 failureThreshold: 3 initialDelaySeconds: 5 periodSeconds: 30 successThreshold: 1 timeoutSeconds: 10

**
2.配置应用业务探针（Readiness Probe）**

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: readiness-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key:

node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi readinessProbe: exec: command: - /bin/sh - '-c' - sleep 1 && exit 0 failureThreshold: 3 initialDelaySeconds: 5 periodSeconds: 30 successThreshold: 1 timeoutSeconds: 10

3.配置应用启动探针(StartupProbe Probe)

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: startup-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi startupProbe: exec: command: - /bin/sh - '-c' - sleep 1 && exit 0 failureThreshold: 3 initialDelaySeconds: 5 periodSeconds: 30 successThreshold: 1 timeoutSeconds: 10

在容器内获取元数据

在容器内获取元数据
`当前仅支持在CCE集群中获取元数据`

通过 Downward API 访问元数据

Kubernetes Downward API提供了以下两种方式：

* 环境变量（Environment variables）用于单个变量，可以将Pod信息直接注入容器内部。

* Volume挂载（Volume Files）可以将Pod信息生成为文件，直接挂载到容器内部。
目前BCI已经支持了Downward API的大部分常用字段，下文将为您介绍使用方式。

1.环境变量方式

您可以通过Downward API将Pod的名称、命名空间、IP等信息注入到容器的环境变量中。通过环境变量可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称
metadata.namespace	Pod命名空间
metadata.uid	Pod的UID
metadata.labels['<KEY>']	Pod的标签值
metadata.annotations['<KEY>']	Pod的注解值
spec.serviceAccountName	Pod服务账号名称
spec.nodeName	节点名称
status.podIP	节点IP
limits.cpu	容器的 CPU 限制值
requests.cpu	容器的 CPU 请求值
limits.memory	容器的内存限制值
requests.memory	容器的内存请求值

> **注意：**
>
> 暂不支持的字段如下：
> * status.hostIP
> * resource: limits.ephemeral-storage
> * resource: requests.ephemeral-storage
>

配置示例：

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: env-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi env:

- name: MY_POD_IP valueFrom: fieldRef: fieldPath: status.podIP
- name: MY_ENV value: "test"
- name: METADATA_NAME valueFrom: fieldRef: fieldPath: metadata.name
- name: METADATA_NAMESPACE valueFrom: fieldRef: fieldPath: metadata.namespace
- name: METADATA_UID valueFrom: fieldRef: fieldPath: metadata.uid
- name: METADATA_LABELS valueFrom: fieldRef: fieldPath: metadata.labels['mylabel']
- name: METADATA_ANNOTATIONS_REGION valueFrom: fieldRef: fieldPath: metadata.annotations['myannotation']
- name: MY_CPU_LIMIT valueFrom: resourceFieldRef: containerName: c01 resource: limits.cpu
- name: MY_CPU_REQUEST valueFrom: resourceFieldRef: containerName: c01 resource: requests.cpu
- name: MY_MEM_LIMIT valueFrom: resourceFieldRef: containerName: c01 resource: limits.memory
- name: MY_MEM_REQUEST valueFrom: resourceFieldRef: containerName: c01 resource: requests.memory

2.Volume挂载方式

您可以通过Downward API将Pod的Label、Annotation等信息通过Volume挂载到容器的某个文件中。通过Volume挂载可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称
metadata.namespace	Pod命名空间
metadata.uid	Pod的UID
metadata.labels['<KEY>']	Pod的标签值
metadata.annotations['<KEY>']	Pod的注解值
metadata.labels	Pod的所有标签
metadata.annotations	Pod的所有注解
limits.cpu	容器的 CPU 限制值
requests.cpu	容器的 CPU 请求值
limits.memory	容器的内存限制值
requests.memory	容器的内存请求值

配置示例：

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: volume-test namespace: default spec: nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
  - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi volumeMounts: - name: podinfo mountPath: /etc/podinfo volumes:
- name: podinfo downwardAPI: items: - path: "metadata.name" fieldRef: fieldPath: metadata.name - path: "metadata.namespace" fieldRef: fieldPath: metadata.namespace - path: "metadata.uid" fieldRef: fieldPath: metadata.uid - path: "mylabel" fieldRef: fieldPath: metadata.labels['mylabel'] - path: "myannotation" fieldRef: fieldPath: metadata.annotations['myannotation'] - path: "labels" fieldRef: fieldPath: metadata.labels - path: "annotations" fieldRef: fieldPath: metadata.annotations - path: "workload_cpu_limit" resourceFieldRef: containerName: c01 resource: limits.cpu

divisor: 1m - path: "workload_cpu_request" resourceFieldRef: containerName: c01 resource: requests.cpu divisor: 1m - path: "workload_mem_limit" resourceFieldRef: containerName: c01 resource: limits.memory divisor: 1Mi - path: "workload_mem_request" resourceFieldRef: containerName: c01 resource: requests.memory divisor: 1Mi

容器生命周期回调

容器生命周期回调

本文介绍如何使用BCI容器生命周期回调框架，即preStop和postStart函数触发业务自定义代码逻辑执行，详情请见[container-lifecycle-hooks](https://kubernetes.io/zh-cn/docs/concepts/containers/container-lifecycle-hooks/)

> **注意：**
>
> BCI目前只支持exec形式的hook，httpGet和tcpSocket类型的hook会被忽略。
>

postStart

这个回调在容器被创建之后立即被执行。 但是不能保证回调会在容器入口点（ENTRYPOINT）之前执行。 没有参数传递给处理程序。

pod示例如下：

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: poststart-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi lifecycle: postStart: exec: command: ["/bin/sh","-c","echo postStart hook > /tmp/termination-log"]

preStop

在容器因API请求或者管理事件（诸如存活态探针、启动探针失败、资源抢占、资源竞争等） 而被终止之前，此回调会被调用。如果容器已经处于已终止或者已完成状态，则对preStop回调的调用将失败。在用来停止容器的TERM信号被发出之前，回调必须执行结束。Pod的终止宽限周期在PreStop回调被执行之前即开始计数，所以无论回调函数的执行结果如何，容器最终都会在Pod的终止宽限期内被终止。没有参数会被传递给处理程序。

pod示例如下：

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: prestop-test namespace: default spec: enableServiceLinks: false nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi lifecycle: preStop: exec: command: ["/bin/sh","-c","echo preStop hook > /tmp/termination-log && sleep 30"]

设置容器终止消息

设置容器终止消息

在Kubernetes中，terminationMessagePath和terminationMessagePolicy用于指定容器终止消息的来源路径和策略。本文介绍如何设置BCI容器的terminationMessagePath和terminationMessagePolicy字段，实现自定义设置容器终止消息。

配置说明
Kubernetes通过terminationMessagePath和terminationMessagePolicy管理容器终止消息。

字段	说明
terminationMessagePath	用于设置容器终止的消息来源。即当容器退出时，Kubernetes 从容器的terminationMessagePath字段中指定的终止消息文件中检索终止消息。默认值为 /dev/termination-log。 通过自定义设置terminationMessagePath，可以使得Kubernetes在容器运行成功或失败时，使用指定文件中的内容来填充容器的终止消息。终止消息内容最大为4 KB。
terminationMessagePolicy	用于设置容器终止消息的策略。取值为： - File（默认）：仅从终止消息文件中检索终止消息。 - FallbackToLogsOnError：在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一部分内容来作为终止消息。

> **说明**：
Pod内所有容器的终止信息大小之和最大为12 KB。当总和超过12 KB时，Kubernetes的状态管理器会对其加以限制。例如：Pod内有4个InitContainer和8个应用Container，则状态管理器会限制每个容器的终止信息最大为1 KB，即截取每个Container终止信息的前1 KB。

操作指南

在以下示例中，配置了terminationMessagePath字段为：/tmp/termination-log，则容器将把终止消息写入/tmp/termination-log给Kubernetes接收。

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: termination-test namespace: default spec: nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:
    - containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi terminationMessagePath: "/tmp/termination-log"

此外，您还可以设置容器的terminationMessagePolicy字段，进一步自定义容器终止消息。该字段默认值为：File，即仅从终止消息文件中检索终止消息。您可以根据需要设置为：FallbackToLogsOnError，即在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一部分内容来作为终止消息。

apiVersion: v1 kind: Pod metadata: annotations: myannotation: "myannotation" labels: app: bci-test-vk mylabel: "mylabel" name: volume-test namespace: default spec: nodeSelector: type: virtual-kubelet tolerations: - effect: NoSchedule key: virtual-kubelet.io/provider operator: Equal value: baidu - effect: NoExecute key: node.kubernetes.io/not-ready operator: Exists tolerationSeconds: 300 - effect: NoExecute key: node.kubernetes.io/unreachable operator: Exists tolerationSeconds: 300 containers:

- image: hub.baidubce.com/cce/nginx-alpine-go imagePullPolicy: IfNotPresent name: c01 workingDir: /work ports:

- containerPort: 8080 protocol: TCP resources: limits: cpu: 250m memory: 512Mi requests: cpu: 250m memory: 512Mi terminationMessagePath: "/tmp/termination-log" terminationMessagePolicy: "FallbackToLogsOnError"

设置容器时区

本文介绍如何为容器配置时区，以此来保证容器中的时间与所处环境的时间一致，避免时区错误导致的时间一致性和准确性等问题。

配置示例
1. 创建一个ConfigMap，导入/usr/share/zoneinfo/目录下您需要的时区

```yaml
kubectl create configmap tz --from-file=/usr/share/zoneinfo/Asia/Shanghai
```

2. 创建配置时区的应用

```
kubectl apply -f timezone-demo.yaml
```

timezone-demo.yaml内容示例如下：

```yaml
  apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    name: pod-test
  name: pod-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      name: pod-test
  template:
    metadata:
      creationTimestamp: null
      labels:
        name: pod-test
    spec:
      containers:
      - image: registry.baidubce.com/glen-centos/centos:centos7
        imagePullPolicy: IfNotPresent
        name: pod-test
        resources:
          limits:
            cpu: 250m
            memory: 500Mi
          requests:
            cpu: 250m
            memory: 500Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - name: tz
          mountPath: /etc/localtime   # 挂载路径
          subPath: Shanghai  # 请根据您的ConfigMap替换
      dnsPolicy: Default
      volumes:
      - name: tz
        configMap:   # 挂载ConfigMap
          name: Shanghai
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
      nodeSelector:
        type: virtual-kubelet
```

验证结果

1. 提交上述yaml，登录到容器内，验证时区是否设置成功。命令提交前根据实际Pod名称替换。

```
kubectl exec -it <pod-name> -- sh
```

2. 查询容器时区

```
date -R
```

如果返回的时间与设置的时区信息对应，则表示设置成功。返回示例如下：

```
Thu, 13 Feb 2025 18:00:11 +0800
```

强制终止Sidecar容器并忽略容器退出码

当您采用Sidecar容器的形式实现类似DaemonSet的效果时，可能会出现Job类Pod无法运行完成的情况，此时可以通过设置BCI Pod Annotation指定需要忽略容器退出码的容器，以保证Job可以正常运行完成。

功能说明 在BCI场景下，由于虚拟节点的限制，BCI不支持Kubernetes的DaemonSet功能。此时部分需要使用DaemonSet的场景可以采用为BCI Pod添加Sidecar容器的形式来实现类似效果，但该方式在RestartPolicy配置为OnFailure和Never时，会影响BCI Pod的生命周期。例如：运行Job类任务时，为Job添加Filebeat Sidecar容器后，由于业务容器退出后，Filebeat容器会继续运行，会导致Job始终无法达到终态，无法运行完成。

针对上述场景，BCI支持通过Annotation指定需要忽略状态码的Sidecar容器列表：

- 忽略容器退出码

  当业务容器正常退出后，由于BCI强制终止Sidecar容器的运行时，Sidecar容器的退出码为非0（非0表示容器运行失败终止），会导致Job最终的状态为Failed，此时可以通过设置Annotation的方式，标记Sidecar容器忽略容器退出码，强制将该容器置为运行成功终止状态，保证Job最终的状态为Succeeded。

配置说明 为BCI Pod设置以下Annotation来标记忽略退出码的Sidecar容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-ignore-exit-code-containers: "sidecar1,sidecar2"
```

> 重要
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例

1. 编写Job的YAML配置文件，然后使用该YAML文件创建Job

   ```
   kubectl apply -f test-sidecar-job.yaml
   ```

   test-sidecar-job.yaml的内容示例如下，表示创建一个Job，Job内包含两个容器，c1为业务容器，c2为Sidecar容器，并且添加了Annotation声明忽略Sidecar容器退出码。

```
 apiVersion: batch/v1
kind: Job
metadata:
 labels:
   app: test
 name: test
 namespace: default
spec:
 template:
   metadata:
     annotations:
       bci.virtual-kubelet.io/bci-ignore-exit-code-containers: "c2"
     labels:
       app: test
     name: test
   spec:
     containers:
     - name: c1
       image: registry.baidubce.com/glen-centos/centos:centos6
       command: ["control.sh", "start"]
       imagePullPolicy: Always
       resources:
         limits:
           cpu: 1
           memory: 2Gi
     - name: c2
       image: registry.baidubce.com/glen-centos/centos:centos6
       imagePullPolicy: Always
       command: ["/bin/sh", "-c", "sleep 120"]
       resources:
         limits:
           cpu: 250m
           memory: 512Mi
     nodeSelector:
       type: virtual-kubelet
     restartPolicy: Never
     tolerations:
     - effect: NoSchedule
       key: virtual-kubelet.io/provider
       operator: Equal
       value: baidu
```

2. 查看Job详情和对应的Pod详情，观察效果

   - 确认Job已经运行完成，且状态为Succeeded

     ```
     kubectl describe job <job-name>
     ```

     示例如下：

   - 查看Sidecar容器详情，确认实际的退出码和相关信息

     ```
     kubectl describe pod <pod-name>
     ```

     示例如下：

配置NTP服务

在时间敏感的业务场景，使用BCI部署容器应用时，您可以配置NTP服务来确保Pod内容器的时间同步准确，从而规避因时间不准确导致的问题，保证数据准确性和业务正常运行。 **配置说明** 创建BCI Pod时，可以为Pod添加`bci.virtual-kubelet.io/bci-ntp-server`的Annotation来指定NTP服务器的地址，使得Pod内的容器能与NTP服务进行时间同步，从而保证时间准确性。

> Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
>
> 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。
>
> Annotation `bci.virtual-kubelet.io/bci-ntp-server`对应的值，只支持指定一个ip

配置示例

1. 创建配置NTP服务的应用。

```
kubectl create -f set-ntp.yaml
```

yaml内容示例如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: centos6-vk
  name: centos6-vk
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: centos6-vk
  template:
    metadata:
      annotations:
        bci.virtual-kubelet.io/bci-ntp-server: "10.0.xx.xx" # 指定NTP服务器的IP地址
      labels:
        app: centos6-vk
      name: centos6-vk
    spec:
      containers:
      - command:
        - /bin/sh
        - -c
        - sleep 3600000
        image: registry.baidubce.com/glen-centos/centos:centos6
        imagePullPolicy: Always
        name: centos-test
        resources:
          limits:
            cpu: 250m
            memory: 512Mi
      nodeSelector:
        type: virtual-kubelet
      restartPolicy: Always
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

2. 时间同步以sidecar的形式集成在了提交的pod内，该sidecar对用户不可见，需要登录pod所在的机器查看

```
kubectl exec -it ${bci-podname} -c ntp-sidecar -- sh
```

3. 查询容器的时间来源

```
sh-4.2# chronyc sources
```

输出结果：

```
210 Number of sources = 1
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* 10.0.xx.xx                10   6    37    38   +517ns[ +221us] +/-  205us
```

忽略Sidecar容器的NotReady状态

当您采用Sidecar容器的形式实现类似DaemonSet的效果时，如果Sidecar容器的状态为NotReady，会导致Pod状态为NotReady。某些场景下，如果您不希望Sidecar容器的状态影响整个Pod状态，可以通过设置BCI Pod Annotation，设置需要忽略的Sidecar容器的NotReady状态的容器列表，保证Pod状态不受Sidecar容器状态的影响。

功能说明 在BCI场景下，由于虚拟节点的限制，BCI不支持Kubernetes的DaemonSet功能。此时部分需要使用DaemonSet的场景可以采用为BCI Pod添加Sidecar容器的形式来实现类似效果。但添加Sidecar容器后，如果Sidecar容器的状态为NotReady，会导致Pod状态为NotReady。在某些场景下，您可能会不希望Sidecar容器状态影响整个Pod状态，例如：使用Sidecar容器用于收集日志，但日志容器出现问题，不应该影响业务容器对外提供服务。

针对上述场景，BCI支持了忽略容器NotReady状态的功能。如果您不希望某一容器的状态影响整个Pod状态，可以为其添加忽略容器状态的环境变量。添加后，当该容器出现NotReady状态时，也不会影响Pod进入Ready状态。

配置说明 为BCI Pod设置以下Annotation来标记忽略退出码的Sidecar容器列表，容器名称之间用半角逗号隔开：

```
bci.virtual-kubelet.io/bci-ignore-not-ready-containers: "sidecar1,sidecar2"
```

> 重要
>
> - Annotation请添加在Pod的metadata下，例如：创建Deployment时，Annotation需添加在spec>template>metadata下。
> - 仅支持在创建BCI Pod时添加BCI相关Annotation来生效BCI功能，更新BCI Pod时添加或者修改BCI相关Annotation均不会生效。

配置示例

1. 编写Deployment的YAML配置文件，然后使用该YAML文件创建deployment

   ```
   kubectl apply -f test-sidecar-ignore.yaml
   ```

   test-sidecar-ignore.yaml的内容示例如下，表示创建一个deployment, 内包含两个容器，c1为业务容器，c2为Sidecar容器，并且添加了Annotation声明忽略Sidecar容器NotReady的状态。

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-ignore
  labels:
    app: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
      annotations:
        bci.virtual-kubelet.io/bci-ignore-not-ready-containers: "c2"
    spec:
      containers:
      - name: c1
        image: registry.baidubce.com/glen-centos/centos:centos6
        command: ["/bin/sh", "-c", "sleep 999"]
      - name: c2
        image: registry.baidubce.com/glen-centos/centos:centos6
      - name: c3
        image: registry.baidubce.com/glen-centos/centos:centos6
      nodeSelector:
        type: virtual-kubelet
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      tolerations:
      - effect: NoSchedule
        key: virtual-kubelet.io/provider
        operator: Equal
        value: baidu
```

2. 查看Deployment对应的Pod详情，查看Pod的Condition观察效果，可见ContainersReady和Ready类型的condition的状态均为True。

    ```bash
    kubectl get pod <pod-name> -o yaml
    ```

conditon示例如下：

```
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:52Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:14:00Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:14:00Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:51Z"
    status: "True"
    type: PodScheduled
  - lastProbeTime: null
    lastTransitionTime: "2025-03-20T05:10:43Z"
    status: "True"
    type: Creating
```

日志

bls日志采集使用方式

BCI容器组支持采集业务容器内的日志文件、标准输出，并推送到BLS(百度云日志服务)。

BCI日志采集架构

上图是BCI日志采集的架构图，如需采集日志，需要创建三类关键对象

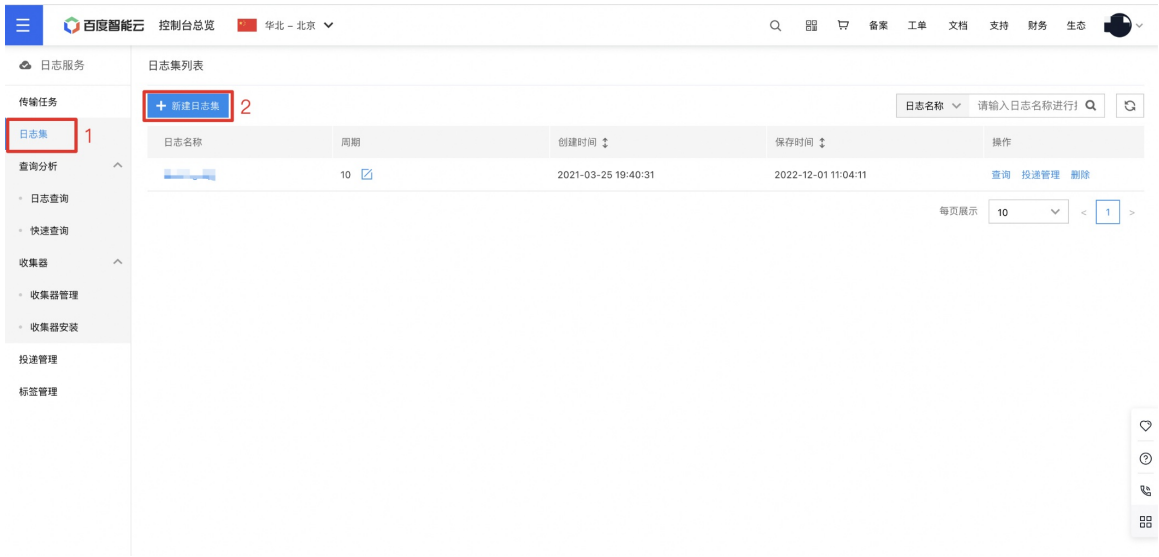
- 存储端：存放用户日志的存储产品，由**用户**负责创建。目前日志采集支持的存储产品有：BLS日志集、百度云Kafka、BOS、BES。
- 采集端：采集物理机/容器内日志文件的Agent组件，一般一个物理机/容器组内部署一个，由**BCI**负责创建。
- 传输任务：实际满足用户的日志采集需求，关键参数包含：**采集任务名称、收集器列表、日志文件路径、日志存储端**。传输任务可以由**用户**创建，也可以由**BCI**创建。 **使用流程 创建存储产品（存储端）** 注：如果已有存储产品，可跳过此步骤。
  用户可选择使用如下任一种云产品作为存储端，详细使用方案如下
- BLS日志集使用文档
- 百度云Kafka使用文档

- BOS使用文档

以 BLS 日志集为例，我们的创建流程如下

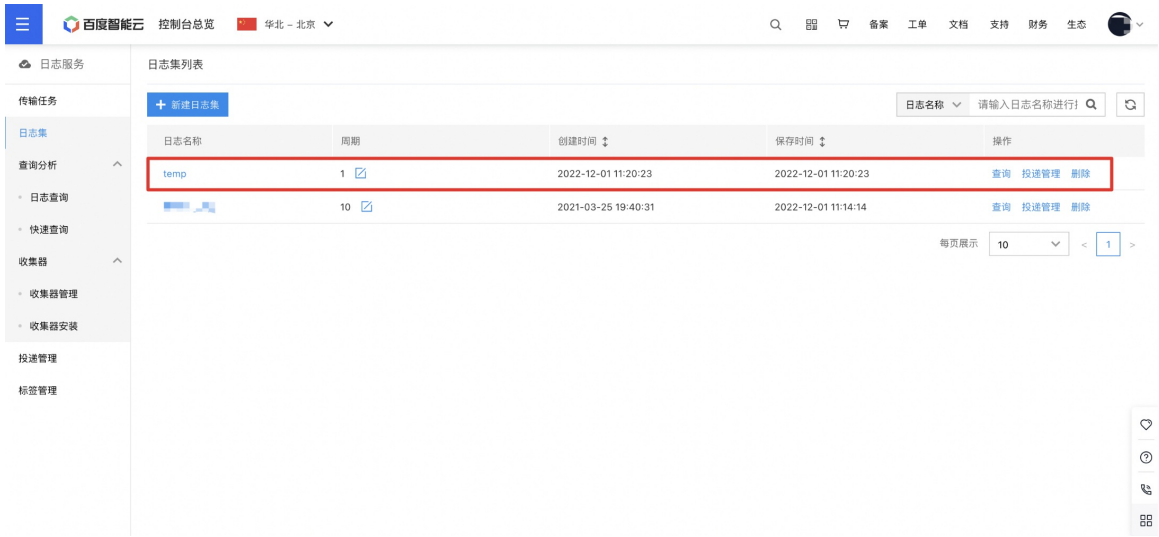
1. 登录百度智能云官网，点击右上角的“管理控制台”，快速进入控制台界面。

2. 选择“产品服务>日志服务BLS”，进入『日志集』页面



3. 点击『新建日志集』，弹出新建日志集页面，填写配置信息，包括：
   a. 名称：设置日志集名称，1-128位字符、数字、英文和符号，符号仅限：_-.。名称创建后不支持修改，并且同一region下名称具有唯一性。
   b. 存储周期：支持1～180天范围的存储周期。如需更大存储周期，请提交 BLS 工单。

4. 日志集名称为『temp』，完成配置后点击『确定』，完成日志集的创建。



BCI创建传输任务 目前支持通过vk方式支持使用BLS日志采集。支持复用已经存在的BLS任务，复用时当前填写的任务配置需和BLS远端配置一致。 **VK使用** 通过vk使用bls日志采集需要在业务容器中设置日志采集任务参数为环境变量，并设置volumeMounts和volume参数。 **1.容器内设置环境变量：** {key} 表示日志采集类型，internal表示采集容器内的日志文件，stdout表示采集容器的标准输出，不支持填写其他内容。
{index}用于区分同一个日志采集任务下的配置，取值为正整数，同一pod内{index}不可重复。
一个容器内支持多个internal类型的bls任务采集任务，仅支持一个stdout类型的bls日志采集任务。

环境变量key名	是否必选	类型	含义	输入限制
bls_task_{key}_{index}_name	是	string	日志采集任务名（支持复用已经存在的bls任务名）	不能包含空格，长度为1-64字符；同一用户名空间下不能重复。
bls_task_{key}_{index}_logStore	是	string	日志集(需用户手动创建)	1-128位字符、数字、英文和符号，符号仅限：_-.
bls_task_{key}_{index}_ttl	是	int	日志采集有效文件时间 单位：天	正整数
bls_task_{key}_{index}_rateLimit	是	int	日志投递速率	单位：MB/s， 速率范围在1-100之间
bls_task_{key}_{index}_matchedPattern	是	string	日志匹配表达式，符合规则的日志，将被采集	无
bls_task_{key}_index_srcDir	是	string	日志采集源目录	容器输出填写stdout，容器内日志填写目录

2.容器内设置volumeMounts和volume 每添加一个bls日志采集任务，需要在yaml文件中添加对应的volumeMounts参数和volume参数。

- **internal 类型**

  volumeMounts参数:

  name应与环境变量bls_task_internal_{index}_name对应的value一致，mountPath应该与日志采集任务对应的srdDir路径一致。

```
volumeMounts:
 - mountPath: /var/log
   name: emptydir1
```

volume参数:

name应与环境变量bls_task_internal_{index}_name对应的value一致，类型为： emptyDir。

```
volumes:
 - name: emptydir1
   emptyDir: {}
```

- **stdout类型**

  volumeMounts参数：

```
volumeMounts:
 - mountPath: /stdout
   name: sidecar-stdout
```

volume参数：

```
 volumes:
- name: sidecar-stdout
  flexVolume:
    driver: k8s/sidecar-stdout
```

注意：多个标准输出日志存储只需要指定一次sidecar-stdout volume，未指定stdout日志采集，则无需指定该volume。

- **使用示例**

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  labels:
    app: test-bls-log
  name: test-bls-log
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-bls-log
  template:
    metadata:
      labels:
        app: test-bls-log
    spec:
      containers:
        - name: nginx
          image: registry.baidubce.com/qa-test/wynginx:7.0.0
          imagePullPolicy: Always
          env:
            - name: bls_task_internal_index_name       # 注：index用来区分不同的采集任务，如
bls_task_internal_1_name,bls_task_stdout_2_name。
              value: {{bls_task_internal_index_name}}
            - name: bls_task_internal_index_logStore
              value: {{bls_task_internal_index_logStore}}
            - name: bls_task_internal_index_ttl
              value: {{bls_task_internal_index_ttl}}
            - name: bls_task_internal_index_rateLimit
              value: {{bls_task_internal_index_rateLimit}}
            - name: bls_task_internal_index_matchedPattern
              value: {{bls_task_internal_index_matchedPattern}}
            - name: bls_task_internal_index_srcDir
              value: {{bls_task_internal_index_srcDir}}
            - name: bls_task_stdout_index_name
              value: {{bls_task_stdout_index_name}}
            - name: bls_task_stdout_index_logStore
              value: {{bls_task_stdout_index_logStore}}
            - name: bls_task_stdout_index_ttl
              value: {{bls_task_stdout_index_ttl}}
            - name: bls_task_stdout_index_rateLimit
              value: {{bls_task_stdout_index_rateLimit}}
            - name: bls_task_stdout_index_matchedPattern
              value: {{bls_task_stdout_index_matchedPattern}}
            - name: bls_task_stdout_index_srcDir
              value: {{bls_task_stdout_index_srcDir}} # 标准输出填写stdout即可
          volumeMounts:
            - mountPath: {{bls_task_internal_index_srcDir}}
              name: {{emptyDirName}} # name需与volume name一致
            - mountPath: /stdout  # 标准输出的挂载路径和名称不可修改
              name: sidecar-stdout
      volumes:
        - name: {{emptyDirName}}
          emptyDir: {}
        - name: sidecar-stdout  # 有标准输出的采集任务需要填写，否则不要填写。
          flexVolume:
            driver: k8s/sidecar-stdout
      nodeSelector:
        type: "virtual-kubelet"
      tolerations:
        - key: "virtual-kubelet.io/provider"
          operator: "Equal"
          value: "baidu"
```

```
        effect: "NoSchedule"
```

监控

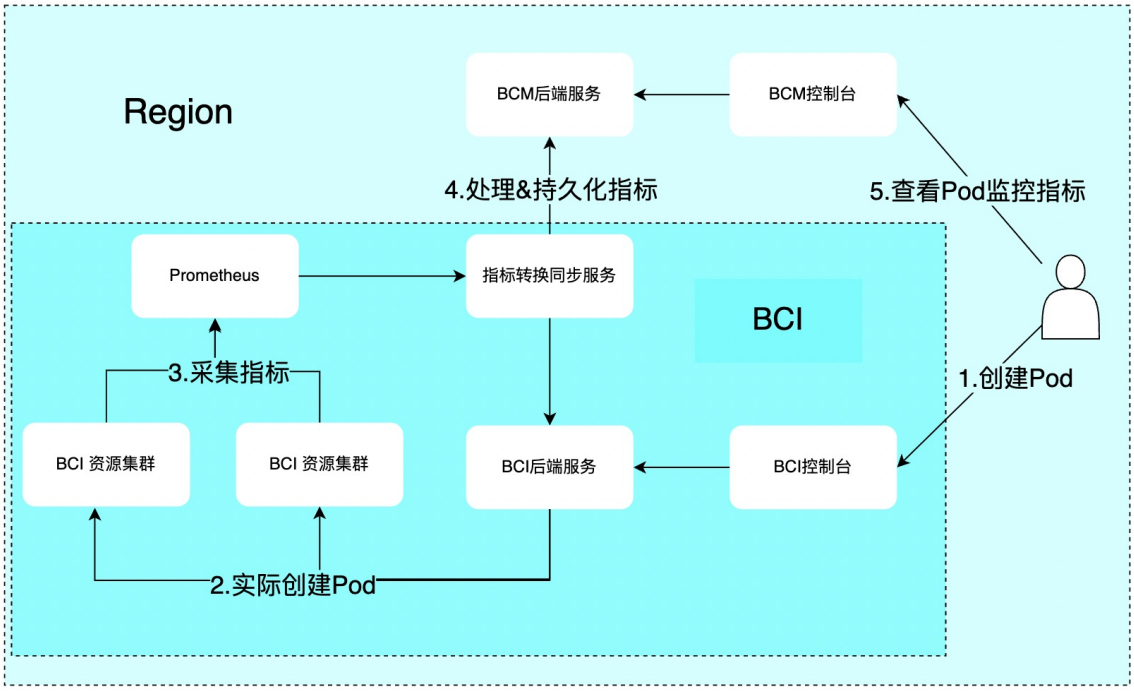
查看实例监控指标

BCI支持自动采集实例的CPU、内存、网络和磁盘等相关监控指标，并可通过BCM（百度云监控）进行查看。

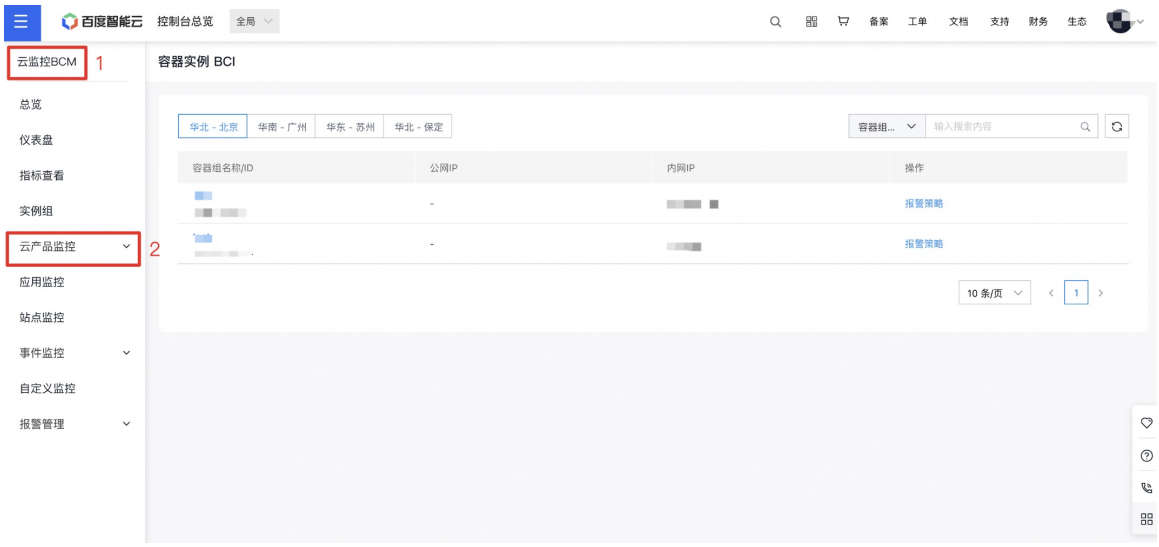
监控指标概述 BCI支持查看的监控指标如下： **CPU**

- CPU使用率：指标采集周期内，平均CPU使用率（单位：%）。CPU处于非IDLE状态即使用，CPU使用率上限为申请的CPU核数*100%。例如：如果申请了1.5 vCPU，且在指标采集窗口期内有50%的时间处于使用状态，则CPU使用率为75%。 **内存**
- 内存用量：指标采集时，正在使用的内存字节数（单位：字节）。使用的内存字节数不包含Page Cache占用的内存空间。 **网络**
- 网络接收量：指标采集周期内，每秒平均接收的网络数据比特数（单位：比特/秒）。
- 网络发送量：指标采集周期内，每秒平均发送的网络数据比特数（单位：比特/秒）。
- 网络接收包：指标采集周期内，每秒平均接收的网络IP包个数（单位：个/秒）。
- 网络发送包：指标采集周期内，每秒平均发送的网络IP包个数（单位：个/秒）。 **磁盘**
- 磁盘读取量：指标采集周期内，每秒平均从文件系统读取的字节数（单位：字节/秒）。指标名称中的磁盘，泛指以磁盘为代表的文件系统，从内存文件系统读取的数据量也会算入指标值中。下述磁盘相关指标的命名都有此含义。
- 磁盘写入量：指标采集周期内，每秒平均写到文件系统的字节数（单位：字节/秒）。
- 磁盘读取次数（暂未采集）：指标采集周期内，每秒平均发起文件系统读请求的次数（单位：次/秒）。
- 磁盘写入次数（暂未采集）：指标采集周期内，每秒平均发起文件系统写请求的次数（单位：次/秒）。

实例监控架构



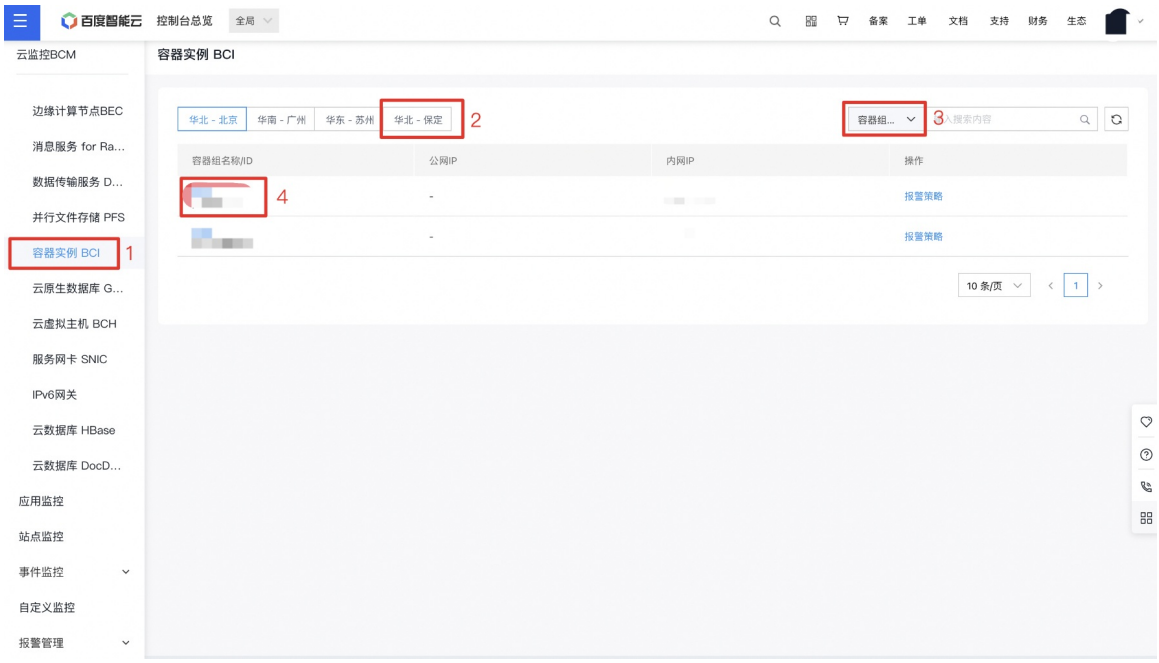
BCM查看监控指标 进入BCM产品页面



1. 进入BCM产品页面

2. 点击展开左侧『云产品监控』

选择容器实例 BCI



1. 展开『云产品监控』后，选择『容器实例 BCI』

2. 选择容器实例所在的 Region

3. 选择查看容器组粒度的监控指标，或者容器粒度监控指标

4. 点击要查看的容器组/容器

更多信息，请参考 云监控BCM操作指南。

通过VK获取实例监控指标

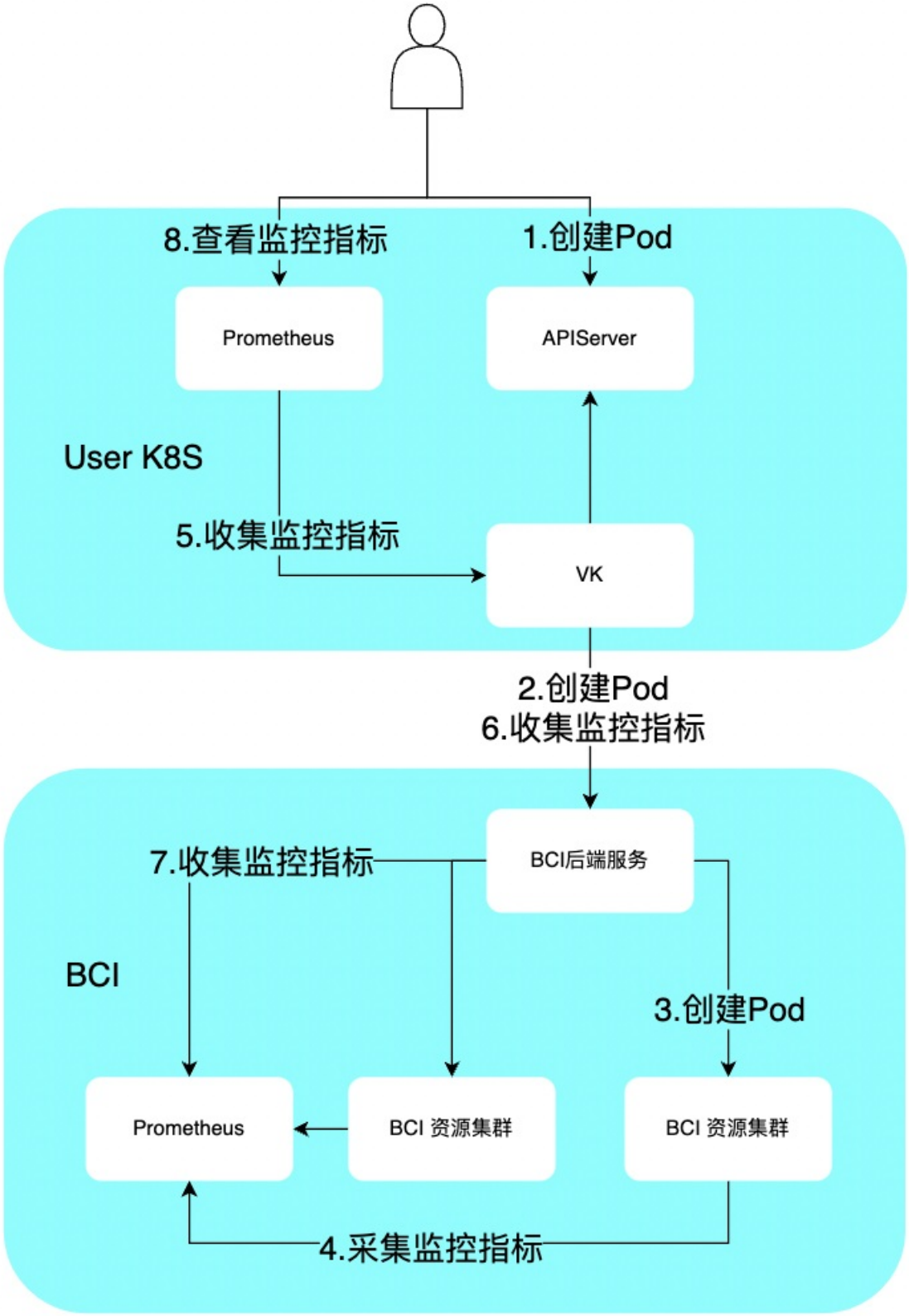
除了通过百度云监控BCM查看 BCI 实例监控指标外，BCI 还支持通过 VK 查看 BCI 实例监控指标。 通过 VK 查看监控指标的方式与 Kubelet 类似，本文将分三部分介绍：

1. 监控指标来源

2. 支持查看的指标

3. 查看指标的方式

监控指标来源 BCI实例监控指标的流向如下图所示



支持查看的指标 通过VK提供的监控指标是prometheus指标格式，与BCM上看到的指标有较大的差异。vk 提供的监控指标格式如下

```
 ###### 指标的帮助说明
###### 指标名称 指标类型
指标名称{标签，标签，...} 指标取值 指标时间戳
指标名称{标签，标签，...} 指标取值 指标时间戳

###### 指标的帮助说明
###### 指标名称 指标类型
指标名称{标签，标签，...} 指标取值 指标时间戳
指标名称{标签，标签，...} 指标取值 指标时间戳
```

指标分为多类，每一类指标的指标名称和指标类型是都一样的，每一类指标又可分为多条（指标名称一样，标签不一样）。
每条指标都包含指标名称、标签集合、指标取值、指标时间戳四个部分。
如下图示例说明：

支持的指标： CPU

- container_cpu_usage_seconds_total：容器创建以来，累计使用cpu的总时间（单位：秒）。 **内存**
- container_memory_working_set_bytes：容器正在使用的内存字节数（单位：字节）。使用的内存字节数不包含Page Cache占用的内存空间 **磁盘**
- container_fs_reads_bytes_total：容器创建以来，累计从文件系统读取的字节数（单位：字节）。
- container_fs_writes_bytes_total：容器创建以来，累计写入到文件系统的字节数（单位：字节）。 **网络**
- container_network_receive_bytes_total：容器创建以来，累计接收的网络数据字节数（单位：字节）。
- container_network_receive_packets_total：容器创建以来，累计接收的网络数据IP包数（单位：个）。
- container_network_transmit_bytes_total：容器创建以来，累计发送的网络数据字节数（单位：字节）。
- container_network_transmit_packets_total：容器创建以来，累计发送的网络数据IP包数（单位：个）。 **支持的标签：**
- container：容器名称
- id：容器复杂ID
- image：容器镜像地址
- name：容器简单ID
- namespace：Pod所在的名字空间
- pod：容器所在的Pod
- device：指标所属的块设备，此标签只在 container_fs_xxx 等几个指标中存在
- interface：指标所属的网卡设备，此标签只在 container_network_xxx 等几个指标中存在

> **注意**：使用了镜像加速功能的容器，其监控指标中的 image 并非用户原始镜像，而是 {ACCELERATE_PREFIX}/transfer/{ACCOUNT_ID}/{USER_ORIGINAL_IMAGE}_accelerate {ACCELERATE_PREFIX} 镜像加速添加

> 的前缀 {ACCOUNT_ID} 用户的账户ID {USER_ORIGINAL_IMAGE} 用户原始镜像地址

查看指标的方式 通过 VK 查看BCI监控指标的方式，与通过 Kubelet 查看 cAdvisor 指标的方式一样，通过如下接口查看

```
 Host:  {KubeletServer}
Port:  10250
URI:   GET /metrics/cadvisor
Param：无
```

通过 vk 查看BCI实例监控指标，示例如下

```
 ./kubectl --kubeconfig={kubeconfig_path} get --raw "/api/v1/nodes/{vk_bci_kubelet_name}/proxy/metrics/cadvisor"
// {vk_bci_kubelet_name} vk 的 nodeName
```

通过 Prometheus 采集存储 BCI实例监控指标，简单配置如下

```
 scrape_configs:
 - job_name: 'kubernetes-kubelet'
   metrics_path: /metrics/cadvisor
   static_configs:
   - targets: ['{vk_bci_kubelet_ip}:10250']

// {vk_bci_kubelet_ip} vk 的 ip
```

运维

使用coredump分析实例程序异常

功能概述

coredump是指在程序运行过程中发生异常终止或崩溃时，操作系统将程序的内存内容转储到一个特殊的文件（即coredump文件）中，以便于后续的调试和分析。BCI容器实例支持例开启coredump运维任务，以便在容器异常终止时可以查看分析coredump生成的文件，从而定位问题原因，修复程序异常。 BCI支持将coredump文件存入CFS中或者对象存储BOS中。

使用流程

BCI默认关闭coredump，避免磁盘占用过多而导致业务不可用。您可以根据需要选择以下方式开启coredump运维任务。

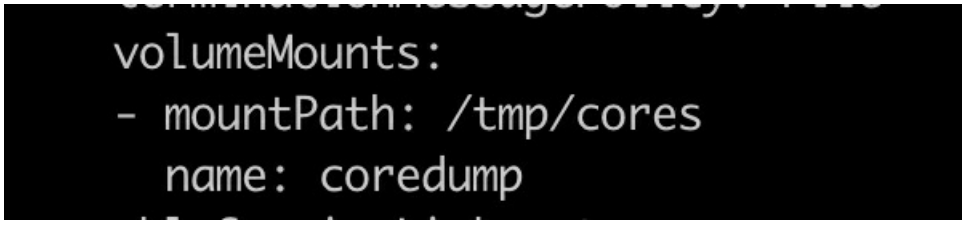
方式一：支持指定Pod Annotation创建Pod开启coredump运维任务 此种形式开启的coredump运维任务在容器运行异常终止或者退出时，触发coredump生成的core文件，该core文件保存在容器/tmp/cores目录中，该目录实际由同地域CFS挂载。

方式二：支持手动开启coredump运维任务 手动通过OpenAPI开启coredump后，BCI将生成一个coredump运维任务,在容器运行异常终止或者退出时，触发coredump生成的core文件将自动保存到对象存储BOS中。

指定Pod Annotation创建Pod开启coredump运维任务 操作步骤如下：

1. 创建BCI 容器实例， 需要注意如下两点：

- 指定并添加如下Pod CorePattern Annotation。需要保证完全一致 bci.virtual-kubelet.io/core-pattern: /tmp/cores
- Pod的远程存储需挂载在容器内/tmp/cores路径，示例如下。



示例example如下：

```
  apiVersion: v1
kind: PersistentVolume
metadata:
  name: coredump-pv-cfs
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  mountOptions:
  - nfsvers=4.2
  nfs:
    path: /  #这里指定远程存储cfs内部路径
    server: cfs-xxxxxxxxx.cfs.gz.baidubce.com #这里指定远程存储cfs地址
  persistentVolumeReclaimPolicy: Retain
  volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: coredump-pvc-cfs
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeMode: Filesystem
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: core-dump-deployment
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: core-dump-deployment
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      annotations:
        bci.virtual-kubelet.io/core-pattern: /tmp/cores ## 必须指定这个annotation, 必须完全一致不能更改。
        myannotation: "myannotation"
      labels:
        app: core-dump-deployment
        mylabel: "mylabel"
    spec:
      containers:
      - image: hub.baidubce.com/cce/nginx-alpine-go
        imagePullPolicy: IfNotPresent
```

```
    name: c01
    workingDir: /work
    ports:
    - containerPort: 8080
      protocol: TCP
    resources:
      limits:
        cpu: 0.25
        memory: 512Mi
      requests:
        cpu: 0.25
        memory: 512Mi
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /tmp/cores  ##这里需要将远程存储挂载到容器内/tmp/cores路径，该路径不支持自定义
      name: coredump
  enableServiceLinks: true
  nodeName: bci-virtual-kubelet-0
  nodeSelector:
    type: virtual-kubelet
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  dnsPolicy: ClusterFirst
  enableServiceLinks: false
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoSchedule
    key: virtual-kubelet.io/provider
    operator: Equal
    value: baidu
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: coredump
    persistentVolumeClaim:
      claimName: coredump-pvc-cfs
```

2. 触发coredump。连接BCI实例，在容器内执行sleep 100命令后按Ctrl+\键，可触发coredump，生成的core文件将自动保存到CFS中。

3. 查看并下载core文件。可登陆CFS相关路径获取core文件

手动开启coredump运维任务 当前BCI仅支持通过OpenAPI开启运维任务。步骤如下

1. 确保BCI实例成功创建并处于Running状态，获取到podID

2. 为BCI实例开启coredump运维任务。调用OpenAPI创建coredump运维任务，接口需指定目标BCI实例，然后将OpsType设为coredump，OpsValue设为enable，设置保存coredump文件的BOS bucket地址，即可开启coredump运维任务。一个注意事项是指定BCI实例时，需确保目标BCI实例在创建时没有设置CorePattern Annotation。

```
 POST http://{{endpoint_bci}}/v2/opstask

示例request body:
{
  "podId":"pod-xxxx",
  "opsType":"coredump",
  "opsValue":"enable",
  "bucket":"xxxbucket"
}
```

3. 触发coredump。连接BCI实例，在容器内执行sleep 100命令后按Ctrl+\键，可触发coredump，生成的core文件将自动保存到BOS中。

4. 下载Core文件。调用OpenAPI查询运维任务接口，指定BCI实例podId和运维任务类型OpsType为coredump查询参数，查看运维任务的结果，从返回信息的result中，可以获取core文件是否已保存到BOS中，可到BOS相关bucket下载core文件。

```
 GET http://{{endpoint_bci}}/v2/opsrecord?podId=xxx&opsType=coredump

示例reponse
{
  "result": [
    {
      "opsType": "coredump",
      "opsStatus": "success",
      "storageType": "bos",
      "~~~~storageContent": "please goto bos page to download file core.p-ysappvy3.sh.8.1713786425 .",
      "createTime": 1713786437000
    }
  ]
}
```

通过BCI控制台和API使用BCI

BCI Pod

BCI Pod概述

BCI能为Kubernetes提供基础的容器Pod运行环境，每个BCI实例对应一个容器组，由vCPU、内存、网络等基础组件组成，用于运行一个或多个容器。本文介绍BCI实例的基本配置、创建方式等。

基本配置 BCI实例包含实例规格、容器镜像、网络、存储等基础组件，您可以方便地定制、更改实例的配置。您对该BCI实例拥有完全的控制权，不需要进行底层服务器的管理和配置操作，只需要提供打包好的容器镜像，即可运行容器。
一个BCI实例相当于一个Pod，包含以下几部分配置：

- 实例规格 规格包括vCPU、内存等配置，定义了BCI Pod的计算性能等。创建BCI Pod时，您可以指定BCI规格（直接指定vCPU和内存）来满足GPU、增强网络能力等特殊需求。
- 容器镜像 一个BCI Pod由一个或者多个容器组成。部署容器应用时，需要准备好容器镜像。容器镜像包含容器应用运行所需的程序、库文件、配置等。拉取镜像时，需要保证网络畅通，推荐您使用镜像缓存功能来节约实例的启动耗时。
- 网络 一个BCI Pod将占用所属VPC下的交换机的一个弹性网卡资源，默认具备一个内网IP地址。如果需要连接公网，例如需要拉取公网镜像。则需要为BCI Pod绑定EIP，或者为所属VPC绑定NAT网关。
- 存储 一个BCI Pod默认有20 GiB的临时存储空间，您可以根据需要增加临时存储空间。如果想要保留存储的文件，建议使用外挂数据卷，支持挂载CFS、EmptyDir和ConfigMap数据卷。
  创建方式 创建方式概述 根据业务场景和使用场景，BCI Pod支持指定vCPU和内存的定义方式，对应计费模式如下。

创建方式	计费说明	相关文档
指定vCPU和内存	根据您创建时指定的vCPU和内存进行计费。对于不满足要求的vCPU和内存规格，系统将自动进行规整，并按自动规整后的规格进行计费。	指定vCPU和内存创建Pod

说明 更多关于BCI Pod计费的信息，请参见 BCI实例计费

优化使用成本 根据您的业务特征，在按量付费使用BCI的基础上，您还可以结合使用抢占式实例、预留实例券来降低资源使用成本。

- 对于无状态且可容错的业务负载，您可以使用抢占式实例。具体操作，请参见创建抢占式实例。
- 对于长时间运行的稳定业务负载，推荐使用预留实例券来抵扣BCI实例账单。

您可以根据使用的BCI实例情况选择合适的方式。具体操作，请参见使用预留实例券。

BCI Pod生命周期

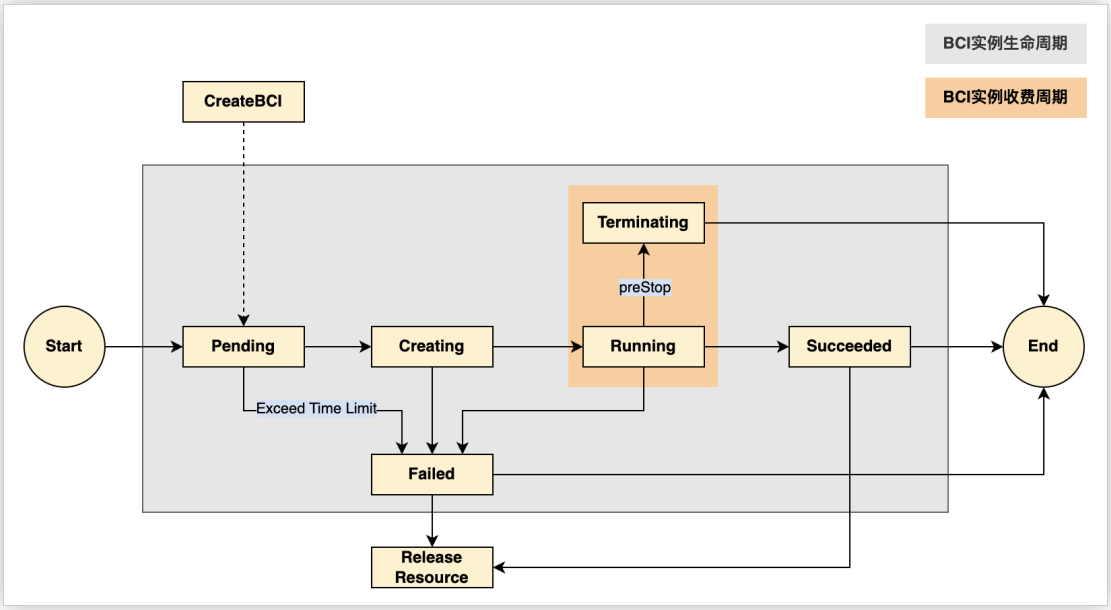
本文介绍BCI实例的生命周期状态，您可以根据实例状态，设计和实现符合您业务逻辑的处理逻辑。

BCI实例状态 在实例的生命周期中，不同的阶段有其固有的状态，具体如下表所示：

BCI Pod状态	说明	对应Kubernetes Pod状态	是否收费
等待创建（Pending）	BCI Pod等待创建。	Pending	否
启动中（Creating）	BCI Pod中有一个或多个容器还在启动中，并且没有处于运行中的容器。	Pending	否
运行中（Running）	BCI Pod中所有容器均已经创建成功，并且至少有一个容器正在运行中。	Running	是
终止中（Terminating）	BCI Pod正在终止。对于运行中的实例，如果配置了preStop，则在删除实例时，Pod将进入Terminating状态。执行完preStop后，BCI Pod将自动删除。	Running	是
运行成功（Succeeded）	BCI Pod中所有容器均已运行成功终止，并且不会再重启。	Succeeded	否
创建/运行失败（Failed）	BCI Pod创建失败。 BCI Pod中所有容器均已运行终止，并且至少有一个容器是运行失败终止，即容器以非0状态退出或者被系统终止。	Failed	否

> **重要** BCI实例的重启策略仅决定实例内容器的行为，BCI实例不会被自动重启。

BCI实例的生命周期状态转换如下图所示：



> **说明**
>
> - 当BCI实例运行终止后，底层计算资源将会被回收，随实例一起创建的其它资源（例如EIP等）默认随实例一起释放。

容器状态

状态	说明
启动中（Waiting）	容器正在等待创建，还未开始运行。 一般在InitContainer运行时，应用容器会处于Waiting状态，直到InitContainer退出。
运行中（Running）	容器已经成功创建，并且正在运行。
运行终止（Terminated）	容器运行终止并退出，包括运行成功终止和运行失败终止。

通过控制台使用BCI

创建BCI实例

前提条件 请确保您已完成以下准备工作：

- 已开通弹性容器实例服务。
- 已创建专有网络VPC和子网。
  关于如何创建专有网络和子网，请参见创建专有网络和创建子网。 **操作步骤**

1. 登录**容器实例BCI控制台**。
2. 进入容器组列表页面，点击创建容器组。

3. 配置基本信息。

- 选择付费方式、地域及可用区。

此处付费方式默认为**后付费**。

> 后付费服务将根据使用情况从账户余额中扣除，请保证有足够的金额。

- 配置基础信息。

  a. 输入容器组名称。

  b. 选择重启策略。

  重启策略将决定容器在退出后是否被重启。默认为**总是重启**，即当容器失效时，将自动重启该容器。

  c. 选择网络类型。

  选择自定义网络及对应子网，若该网络中没有子网，请先选择创建子网或者选择其他私有网络。

  d. 选择安全组。

  安全组是一种虚拟防火墙，具备状态检测和数据包过滤功能，用于设置网络访问控制。

- 选择弹性资源。

  默认为暂不需要。如需公网访问请购买弹性公网IP，或绑定已有弹性公网IP。

4. 配置容器。

- 存储卷。

  NFS:NFS数据卷将网络文件系统挂载到容器组中，提供持久化存储，Server地址需要与容器组在相同VPC或者公网可访问，建议搭配百度云文件服务CFS使用。

  EmptyDir: EmptyDir提供一个容器组内所有容器均能访问的共享路径，将容器组被删除后，EmptyDir上的数据也会被一并删除，适用于容器组运行时的临时数据。

  ConfigFile: ConfigFile用于向容器组中传递配置信息，挂载配置文件后，容器可以直接通过对应路径读取文件内容。

- 容器配置。

  a. 输入容器名称。

  当您需要购买多个容器时，点击添加容器或末尾新增，将按照数量自动在名称后增加有序后缀。例如：容器名称默认为container1，若购买数量为3台，则三台实例的名称分别为：container1、container2、container3。

  b. 输入镜像地址。

  您可以输入指定镜像地址或从仓库中选择镜像。

  c. 输入镜像版本。

  d. 选择容器组的vCPU。

  e. 选择容器组的内存。

  如需设置更多，请点击高级设置。

> 如果有容器使用了私有镜像，请您务必添加对应镜像仓库的访问凭据（仓库地址需包含完整域名和命名空间）。

- 选择标签。
  您可以按各种标准（如用途、所有者或项目）对资源进行分类管理（每个标签包含键和值两部分）。
- 选择购买数量。
  您可以在最下方查看配置详情，以便选择购买数量。

5. 确认订单。
   确认容器实例BCI配置信息以及实际支付金额，然后单击提交。开通成功后，您可以在**容器组**页面查看该实例详情，在**订单管理**页面查看订单明细。

查看BCI实例的详情和事件

本文介绍如何查看BCI实例的详情和事件，有助于您更为全面地了解BCI实例的信息。

操作步骤

1. 登录**容器实例BCI控制台**。
2. 在顶部菜单栏左上角处选择区域。
3. 在**容器组列表**页面，找到您想要查看的实例，单击ID。
4. 容器组详情页面。

- 查看基本信息。
  可以查看BCI实例所属区域、ID、状态、所在网络/子网等基本信息。

- 查看数据卷。

  可以查看所用数据卷名称、类型及配置，如需更多可点击下载配置文件。

  数据卷

数据卷名称	数据卷类型	数据卷配置	
default	configFile	Path: Path: Path:	下载配置文件 下载配置文件 下载配置文件
	configFile	Path:	下载配置文件

- 查看容器。

  点击容器名，可以查看该容器的配置信息。

  容器

  容器名称： 状态： Running 镜像地址：

  镜像版本： CPU： 0.25核 内存： 0.5GB

  工作目录： - 启动命令： /bin/sh 启动参数：

  环境变量：

  端口协议：

  数据卷： [ConfigFile]

5. 查看容器组事件。

   可以在事件列表页面查看事件名称、类型、详情及创建时间。

   事件列表 ⓘ 帮助文档

事件名称	类型	事件详情	事件时间
通知事件	Started	Started container	2023-03-20 09:40:12
通知事件	Created	Created container	2023-03-20 09:40:12
通知事件	Pulled	Container image	2023-03-20 09:40:12

   如果您的容器出现异常，您还可以通过查看日志和监控来排查问题。

连接BCI实例

在完成容器部署后，您可以通过容器实例BCI控制台，使用WebSSH连接BCI实例中的容器，执行命令进行调试。本文介绍如何连接BCI实例。

前提条件

实例中的容器处于**运行中**状态。

操作步骤

1. 登录**容器实例BCI控制台**。
2. 在顶部菜单栏左上角处选择区域。
3. 在**容器组列表**页面，选择您想要调试的实例，单击**WebSSH**。

容器组ID/名称	状态	资源用量	公网IP/带宽	内网IP	重启策略	所属集群ID	标签	创建时间	操作
	运行中	0.25核/0.5GB内存	-/-Mbps		总是重启		-	2023-03-20 10:06:13	详情 删除 WebSSH

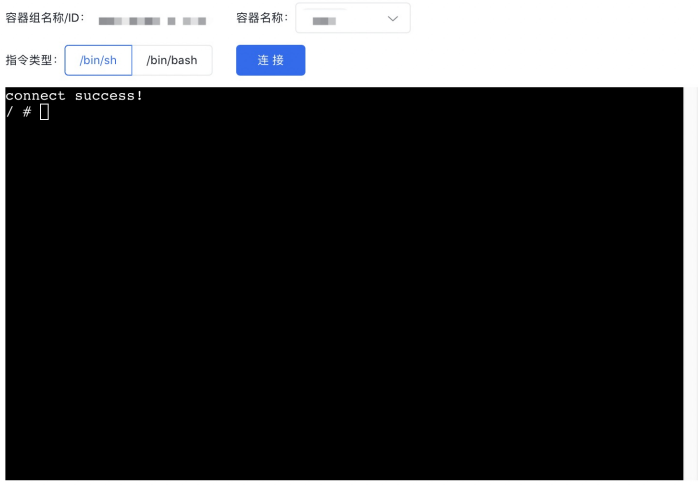
4. 在弹出的WebSSH页面选择容器、指令类型等，然后单击**连接**。

- 选择容器时，仅支持选择**运行中**的容器。

- 指令类型支持常用的/bin/sh和/bin/bash指令。

> 系统默认选择当前可用实例，您也可以切换选择其他实例； 系统默认选择/bin/sh指令类型进行连接，您也可以切换选择其他指令类型。

5. 根据需要输入并执行命令。



删除BCI实例

您可以在容器实例BCI控制台对不再使用的BCI实例，进行删除。

操作步骤

1. 登录**容器实例BCI控制台**。
2. 在顶部菜单栏左上角处选择区域。
3. 在**容器组列表**页面，选择您想要删除的BCI实例。您可以选择以下方式进行删除：

- 单击想要删除的实例对应操作栏中**删除**；
- 选择多个BCI实例，单击顶部**删除**。

容器组列表

4. 确认删除。
   在弹出的对话框中，单击**确定**。

> 由于容器组释放后不可恢复，该容器组的所有日志和事件将被清空，EmptyDir和ConfigFile也将被删除，存放在NFS中的数据不受影响，请您确认是否继续删除操作。

成本优化

预留实例券

预留实例券概述

预留实例券是一种抵扣券，可以抵扣按量付费实例（不含抢占式实例）的账单，也能够预留实例资源。相比包年包月实例，预留实例券与按量付费实例这种组合模式可以兼顾灵活性和成本。 **什么是预留实例券？**

- 当前预留实例券功能处于内测中，如需使用，请提交工单申请。

定义 预留实例券是一种抵扣券,可以抵扣相关配置的后付费BCI实例账单，支持1个月、1年、2年、3年购买周期。 预留实例券相比于后付费BCI实例单价更低且支持资源预留。另外，预留实例券支持多种付费方式（全预付、部分预付），您可根据财务情况灵活选择。 **功能** 预留实例券分为可用区级预留实例券和地域级预留实例券。

1. 预留实例券生效后将自动匹配可抵扣的后付费BCI实例账单；更多信息请查看预留实例券与实例的匹配。
2. 支持资源预留；

- 可用区级资源预留：
  可为所匹配的按量实例资源进行资源保留。
- 地域级资源预留：
  地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

3. 支持抵扣预测。
   预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。更多信息请查看可抵扣的实例。 **付费方式** 预留实例券支持3种付费方式：
4. 全预付：购买券时支付有效期内对应算力的总额，是3种付费方式中总额最低的；
5. 部分预付：购买券时支付有效期内对应算力的一部分费用，剩下的费用将在有效期内的每小时支付。总支付金额 = 半预付金额 + 每小时分期费用 * 购买周期内小时数；

> **注意：**预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。

相较于后付费模式，预留实例券可以有效的降低长时间运行实例的费用成本。

	预留实例券	后付费
定义	一种抵扣券，用来抵扣后付费BCI实例账单。	购买BCI时的一种付费方式（先使用后付费），分钟计费，小时出账。
特点	1. **价格优惠**：总成本比后付费实例低； 2. **资源保障**：支持资源预留，保证您创建成功与预留实例券匹配的实例。 3. **灵活抵扣**：实例券不与某个实例绑定，一张实例券可抵扣同规格的不同后付费BCI账单； 4. **灵活支付**：支持多种付费方式（全预付、部分预付），避免一次性支付带来的资金链压力。	1. 即用即买，随时释放； 2. 单价较高。

资源抵扣 预留实例券生效后将自动匹配满足条件的实例规格(后付费)进行抵扣。
其中抵扣是指相同实例规格的后付费BCI的账单，而不是后付费BCI的费用。

注意：

- 如果您希望对已有的BCI实例进行成本优化，可以筛选出指定规格创建的BCI实例，并根据规格情况购买对应的预留实例券。
- 如果您还没有创建BCI实例，请根据业务需求选择合适的规格，然后购买对应的预留实例券，并指定规格创建BCI实例。
- 如果您已经购买了预留实例券，则在创建BCI实例时，必须要指定预留实例券对应的规格，否则预留实例券无法抵扣。

到期自动购买预留实例券. 预留实例券支持自动续费。如果您是按月购买，则自动续费周期为 1 个月。按年购买，则自动续费周期为 1 年。 ![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/BCI/image_35d8991.png)

购买预留实例券

本文介绍如何在BCI管理控制台购买预留实例券。

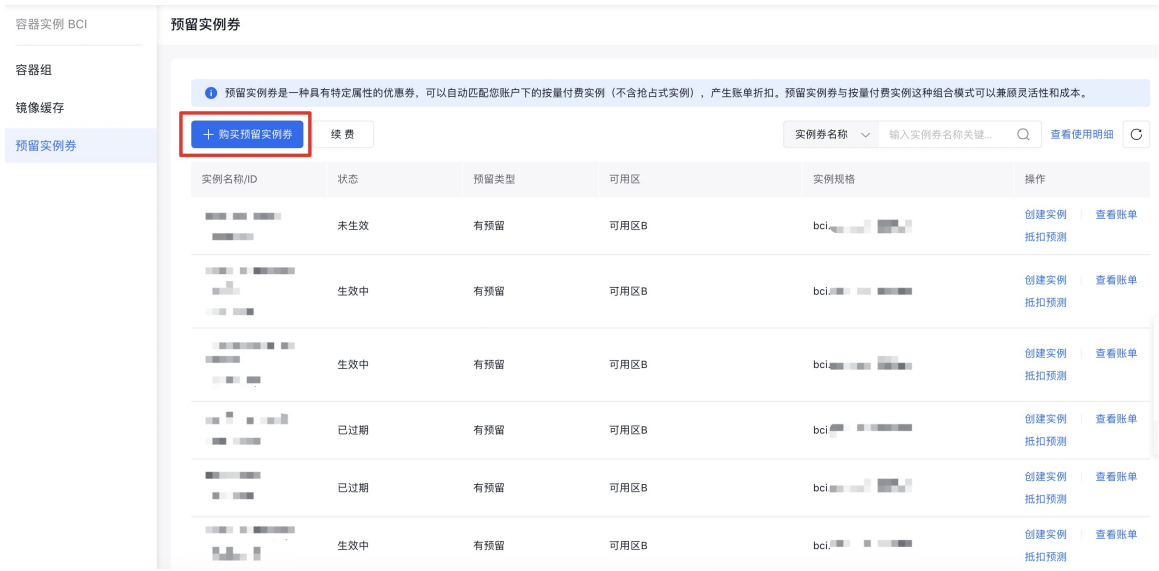
> **说明**
> 预留实例券支持CPU和GPU实例规格。

前提条件

- 购买预留实例券前，请确保您要匹配的按量付费实例符合预留实例券使用要求。详情请参见预留实例券概述。
- 您无法手动管理预留实例券和按量付费实例的匹配状态，请确保您已了解预留实例券匹配规则。详情请参见预留实例券与实例匹配。

操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。
3. 单击**购买预留实例券**。



4. 配置地域信息。
   a. 选择资源预留类型。

> **可用区级资源预留：**
> 可为所匹配的按量实例资源进行资源保留。
> **地域级资源预留：**
> 地域级的预留实例券，不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。
> 详情请查看预留实例券与实例匹配。

   b. 选择地域。
   c. 选择可用区。

5. 配置实例信息。
   a. 选择实例规格。
   b. 选择付费类型。支持**全预付**、**部分预付**和**零预付**。

> 预付费用一次性支付，预留费用每小时扣费收取，无论预留实例券能否匹配到按量实例，在有效期内您都需要按付款类型支付费用，选择全预付可以节省更多成本。
> 零预付根据使用情况自动开放，如果您需要使用，请提交工单申请。

6. 配置购买参数。
   a. 填写券名：可选。
   b. 填写购买数量。 可以同时匹配同规格按量付费实例的数量。

> 最多购买10台，如需购买更多，请提提交工单申请。

c. 选择预留实例券有效期。支持**1个月、1年、2年和3年**。

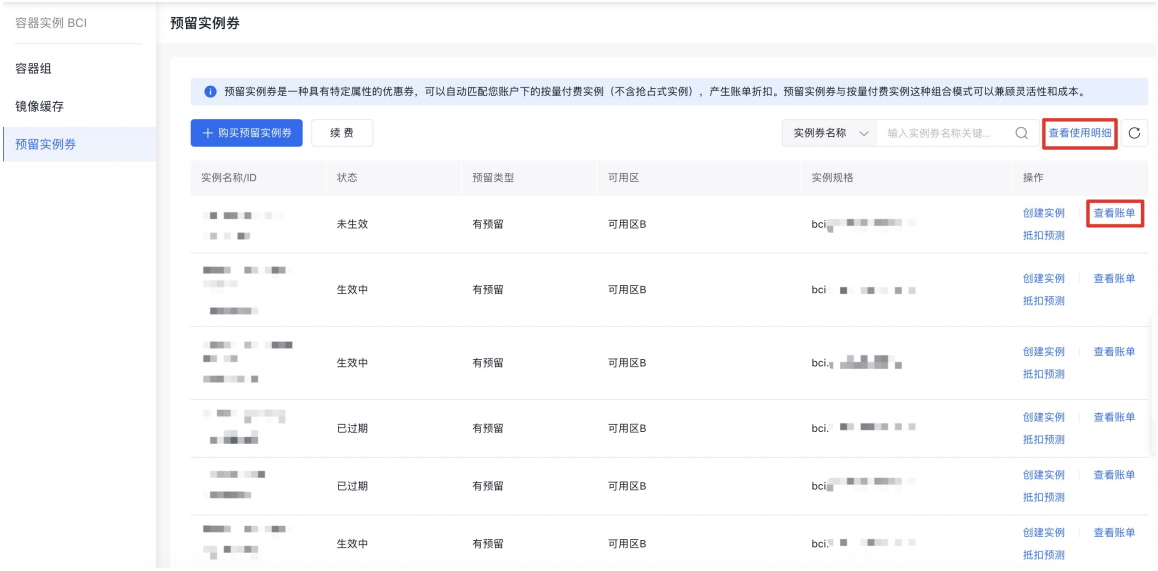
7. 选择是否启动自动续费。

> 按月购买，则自动续费周期为 1 个月。按年购买，则自动续费周期为 1 年。

8. 选择生效时间。目前支持**指定时间生效**。
9. 确认参数，然后单击**提交**。
10. 确认支付信息，然后单击**确认支付**。
    执行结果 您已经成功购买了一张预留实例券，成功匹配按量付费实例后即可抵扣按量付费实例账单。

查看预留实例券使用明细

您可以在预留实例券页面点击**查看使用明细**或**查看账单**，确认券的抵扣情况。



> 重要
> 在配置了多可用区创建BCI实例的场景下，创建的BCI实例可能会发布在多个可用区。如果您发现预留实例券未按预期进行抵扣，请检查预留实例券所在可用区和BCI实例可用区是否一致。

在**抵扣明细**页面，您可以查看预留实例券的抵扣情况。预留实例券的使用明细将记录每个出账周期（每小时）该预留实例券抵扣的实例信息。其中，抵扣时长对应的是计算力*小时（预留实例券按计算力进行抵扣，1个计算力可以理解为1vCPU）。

预留实例券与实例的匹配

购买预留实例券即代表承诺使用一定时长的按量付费实例，预留实例券匹配按量付费实例后才能抵扣账单。如果您的账号下暂时没有可以匹配的按量付费实例，预留实例券会闲置但继续计费。

本章节为您介绍预留实例券匹配按量付费实例的规则和示例。 **匹配规则** 您不能手动匹配预留实例券和按量付费实例。购买预留实例券后，在有效期内预留实例券将自动匹配满足条件的按量付费实例。匹配成功后，预留实例券每小时检查可抵扣的按量付费账单，并按券面的计算力抵扣账单。您可以使用抵扣预测功能查看预留实例券可以匹配的实例，具体操作请参见查看可抵扣的实例。

属性	地域和可用区	实例规格
可用区级预留实例券	只可匹配同一可用区中的按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 支持指定可用区的资源预留，在有效期内预留指定数量指定规格的实例，保证随时可以在指定可用区中成功创建按量付费实例。
地域级预留实例券	支持可用区灵活性，在指定地域中可以跨可用区匹配按量付费实例。	实例大小灵活性和资源预留支持情况如下： 1. 只可匹配实例大小相同的按量付费实例。 2. 不提供指定可用区的库存预留，但能保证您在整个地域创建成功与预留实例券匹配的实例。

可用区级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域和可用区相同。
- 实例规格和实例大小相同。

使用可用区级预留实例券的示例如下表所示。

示例场景	按量付费实例	可用区级预留实例券	抵扣效果
匹配成功	持有5台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：5台	预留实例券和5台按量付费实例完全匹配，1张预留实例券每小时抵扣5台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.c8m36.1a10实例，保证随时可以在华北-保定可用区B中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 华北-保定，可用区B bci.c8m36.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例 1台配置如下： 华北-北京，可用区A bci.c8m36.1a10 1台配置如下： 华北-保定，可用区B bci.c18m74.1a10	持有1张已生效的可用区级预留实例券，属性如下： 华北-保定，可用区B bci.c8m36.1a10 实例数量：2台	导致匹配失败的原因如下： 1台实例可用区为华北-北京，可用区A； 1台实例规格为bci.c18m74.1a10； 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

地域级预留实例券示例 根据匹配要求，待匹配的可用区级预留实例券和按量付费实例必须满足以下条件：

- 地域相同，支持跨可用区匹配实例。
- 实例规格和实例大小相同。

使用地域级预留实例券的示例如下表所示。

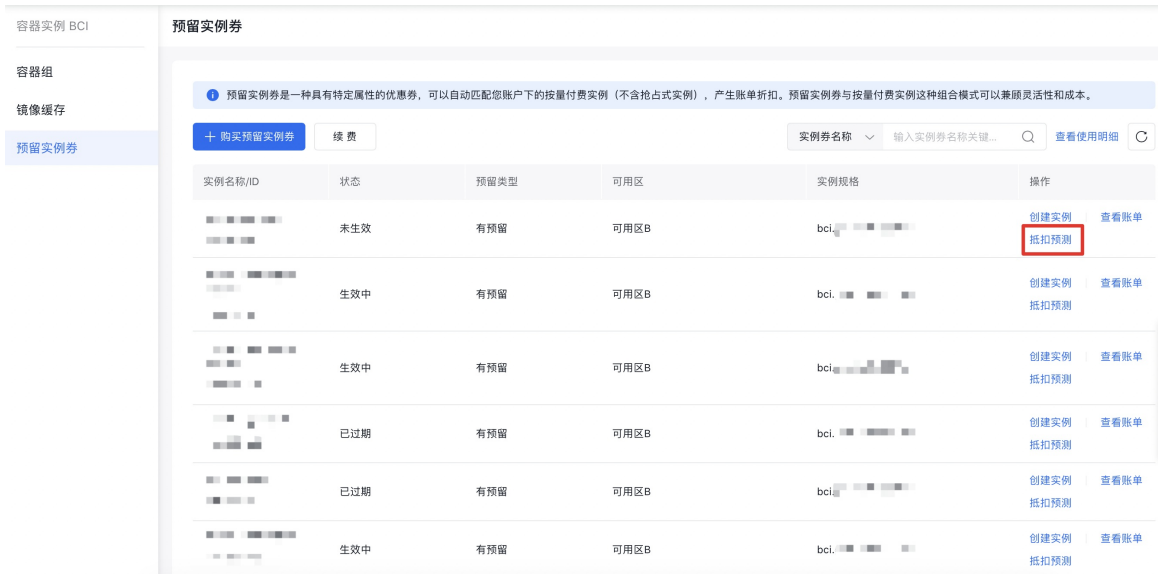
示例场景	按量付费实例	地域级预留实例券	抵扣效果
匹配成功	持有6台按量付费实例，配置如下： 3台配置如下： 华北-保定，可用区A bci.0.25c0.5m 3台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：6台	预留实例券和6台按量付费实例完全匹配，1张预留实例券每小时抵扣6台按量付费实例100%的账单。
无实例资源预留	未持有按量付费实例	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券闲置并继续计费。但会在有效期内为您预留10台bci.0.25c0.5m实例，保证随时可以在华北-保定地域中成功创建按量付费实例。
部分匹配成功	持有20台按量付费实例，配置如下： 12台配置如下： 华北-保定，可用区A bci.0.25c0.5m 8台配置如下： 华北-保定，可用区B bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：10台	预留实例券和10台按量付费实例匹配，1张预留实例券每小时抵扣20台按量付费实例50%的账单（随机抵扣其中10台实例的账单）。
匹配失败	持有2台按量付费实例，配置如下： 华北-北京，可用区A bci.0.25c0.5m	持有1张已生效的地域级预留实例券，属性如下： 华北-保定，跨可用区 bci.0.25c0.5m 实例数量：2台	导致匹配失败的原因如下： 2台实例可用区为华北-北京，可用区A 因此，预留实例券闲置并继续计费，按量付费实例的账单通过账户额度结算。

查看可抵扣的实例

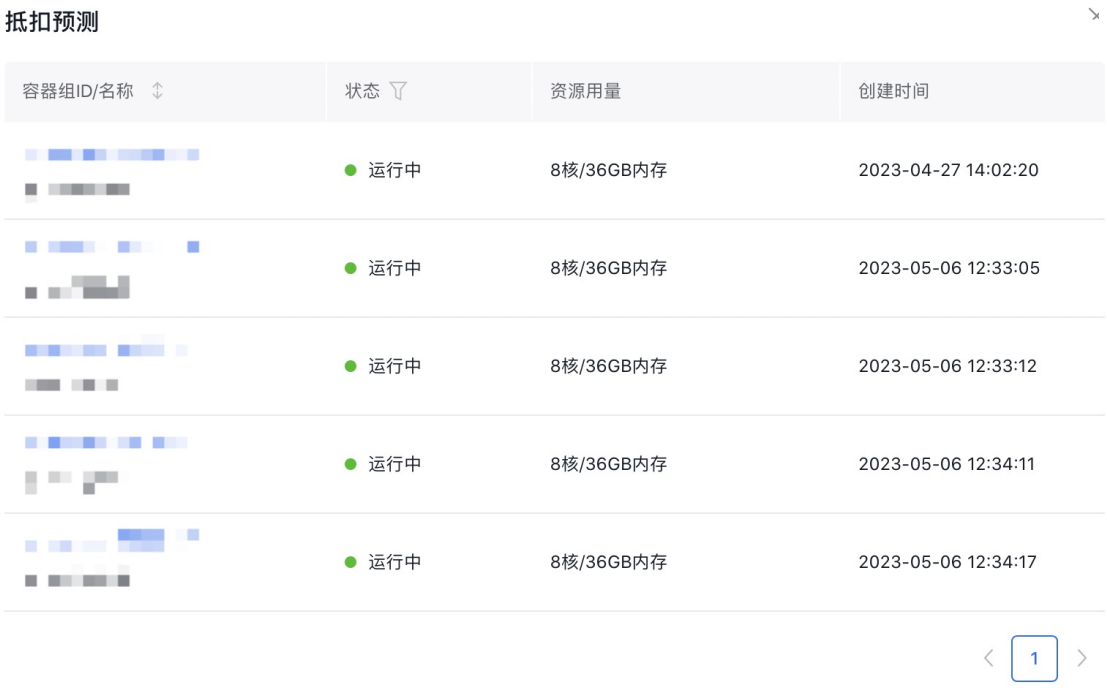
预留实例券支持抵扣预测功能，帮助您判断哪些已购买的按量付费实例符合匹配要求。通过抵扣预测功能查看到实例仅代表实例符合匹配要求，并不代表一定抵扣这些实例的账单，实际抵扣情况以账单为准。

操作步骤

1. 登录BCI管理控制台。
2. 在左侧导航栏，选择**预留实例券**。
3. 在**预留实例券**页面，在操作栏单击**抵扣预测**。



4. 在抵扣预测页面查看预留实例券可以匹配并抵扣账单的按量付费实例。



镜像

镜像缓存

镜像缓存概述

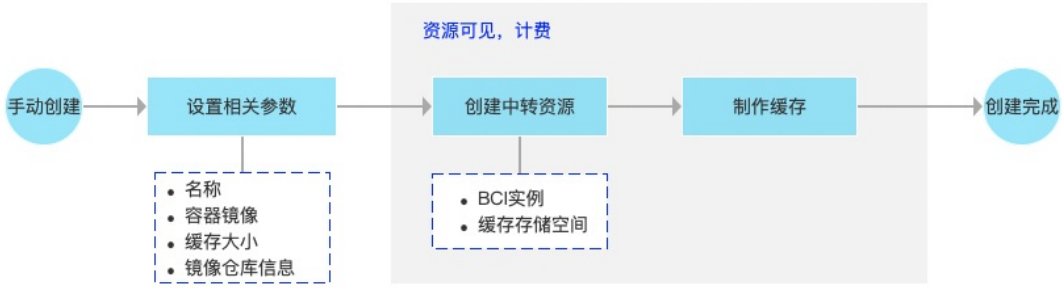
使用镜像缓存（ImageCache）创建BCI实例可以加速拉取镜像，减少BCI实例的启动耗时。本文介绍镜像缓存的基本功能、创建和使用方式、以及计费说明等。 **功能简介** 在运行容器前，BCI需要先拉取您指定的容器镜像，但因网络和容器镜像大小等因素，镜像拉取耗时往往成了BCI实例启动的主要耗时。为加快实例的创建速度，BCI提供镜像缓存功能。您可以预先将需要使用的镜像制作成镜像缓存，然后基于该镜像缓存来创建BCI实例，避免或者减少镜像层的下载，从而提升实例的创建速度。

> **说明：**具体提升速度由BCI实例中使用的镜像个数、镜像大小和镜像仓库网络等因素决定。

创建方式 支持手动创建和自动创建两种方式创建镜像缓存。

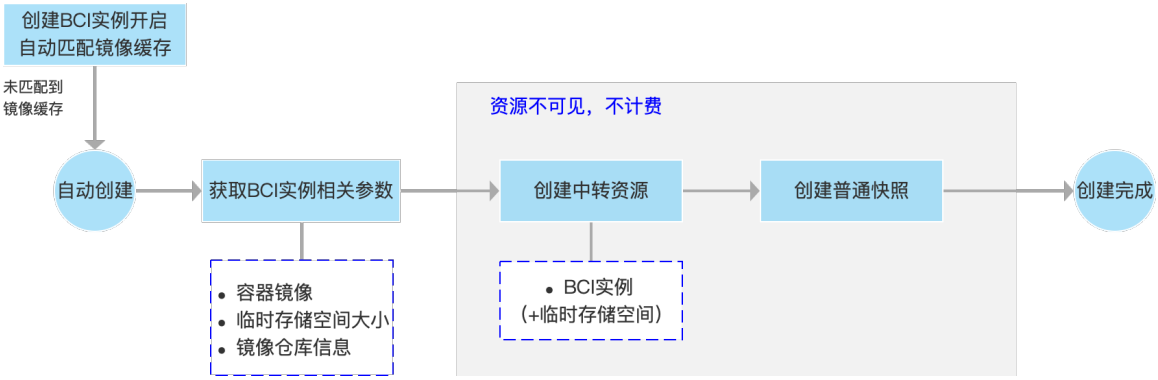
> **说明：**推荐使用自动创建的镜像缓存，可以节约使用成本。但对于首次创建BCI实例就需要加速的场景，建议提前手动创建镜像缓存。

手动创建 您可以根据需要，自行设置名称、大小等参数来创建镜像缓存。具体过程如下：



- 一个镜像缓存对应一份CCR快照，手动创建的镜像缓存快照由您自行管理。
- 创建过程中，系统将创建一个BCI实例（2vCPU、4GiB内存），并挂载一块CSD云盘用于中转创建镜像缓存对应的快照，默认大小20G，可自行根据镜像缓存大小调节。创建完成后，该实例和云盘将被自动释放。

自动创建 在创建BCI实例时，会为其自动匹配镜像缓存。在未匹配到镜像缓存的情况下，系统会在创建BCI实例的同时，自动创建一个镜像缓存。具体过程如下：



- 一个镜像缓存对应一份CCR快照，自动创建的镜像缓存快照由百度云BCI管理。注：该镜像缓存如果在30天内未使用，将被自动删除。
- 创建过程中，系统将创建一个BCI实例，并使用BCI实例自带的存储空间来中转创建镜像缓存对应的快照。创建完成后，该实例将被自动释放。 **使用方式** 创建BCI实例时，使用镜像缓存可以加快BCI实例的创建。目前只支持自动匹配镜像缓存方式：
- 自动匹配：自动匹配使用最优的镜像缓存。目前只支持完全匹配策略。
  - 完全匹配策略：即镜像名称及版本是否完全相同。
    - 注意1：镜像tag不能为latest，必须指定一个明确的tag。否则，镜像缓存不会生效。
    - 注意2：如果用户修改了镜像，必须发布一个新的tag。否则，使用的缓存会比较老。

注意事项

- 创建镜像缓存需要拉取容器镜像，因此创建时长由镜像个数、镜像大小、网络等多种因素决定。
- 自动创建镜像缓存时采用实例中所声明的容器镜像。
  - 如果镜像为私有镜像，则需要提供私有镜像仓库的访问凭证，包括地址、用户名和密码。
  - 如果镜像需要通过公网拉取（如Docker官方镜像），则需要配置EIP或者NAT来访问公网。更多信息，请参见 连接公网。
  - 如果镜像由于远程仓库超时等原因导致拉取失败，推荐您将镜像仓库和VPC打通，或者可以使用容器镜像服务CCR，将镜像上传至百度云仓库。

计费说明

- 创建镜像缓存
  - 手动创建镜像缓存，需支付临时资源（BCI实例和云盘（超出20GiB免费存储空间部分））费用
  - 自动创建镜像缓存，无需付费。

> 注：如果您的镜像需要通过公网拉取，则会产生相应的公网流量费用。

- 使用镜像缓存
  - 使用手动创建或自动创建的镜像缓存创建BCI实例时，如果镜像缓存大于20GiB，需增加临时存储空间并为其付费。否则只需支付BCI实例费用。

管理镜像缓存

镜像缓存可以加速拉取镜像，减少BCI实例启动耗时。本文介绍如何创建、查询、更新和删除镜像缓存。 **创建镜像缓存** 您可以通过控制台手动创建镜像缓存。 **通过控制台创建**

1. 在容器实例服务控制台的镜像缓存页面，单击创建镜像缓存。
2. 在弹出页面，配置相关参数。
   - 网络参数：设置地域、可用区、专有网络和安全组等参数，用于创建镜像中转实例。
   - 公网访问：如果镜像需要公网拉取，在专有网络没有配置NAT网关的情况下，请绑定弹性公网IP。
   - 基础信息：输入镜像缓存名称，选择镜像地址和填写版本号，按需设置临时存储大小。
   - 镜像仓库访问凭证：如果镜像是私有镜像，请填写镜像仓库的地址、访问用户名和密码。
   - 选中自动匹配镜像缓存：系统将自动匹配和使用最佳的镜像缓存。如果没有匹配到，系统将自动创建一个镜像缓存。
3. 单击“提交”，确认创建。
4. 在镜像缓存页面查看创建结果。

   状态列显示创建进度，当状态变为success(100%)时，表示镜像缓存创建成功。

查询镜像缓存 创建镜像缓存后，您可以在容器实例服务控制台的镜像缓存页面查看镜像缓存的名称、状态等信息。当镜像缓存状态为创建完成（success(100%)）时，可以使用该镜像缓存。

删除镜像缓存 目前每位用户最多可保留50个成功创建的镜像缓存，超过限制后将会无法创建新的镜像缓存。对于不再使用的镜像缓存，建议您及时手动删除。 删除方式如下：

- 在容器实例服务控制台的镜像缓存页面，选中目标镜像缓存，单击“删除”即可删除镜像缓存。

使用CCR镜像仓库

背景信息

CCR镜像仓库分为：个人版，企业版

个人版：容器镜像服务个人版主要面向个人开发者或企业客户临时测试使用，提供基础的容器镜像托管、分发等能力，同时在资源配额上有一定的限制。

企业版：容器镜像服务企业版主要面向企业用户业务生产使用，提供容器镜像、Helm Chart等云原生制品生命周期管理，支持多地域、多场景的制品高效分发，同时企业版会不断完善其功能特性并持续更新，帮助企业在业务生产过程中降本增效。

更多信息，请参考容器镜像服务CCR。

前提条件 需要创建好CCR镜像仓库，此处以企业版镜像仓库为例。

创建命名空间：可创建公有命名空间或私有命名空间，私有命名空间需要登录密钥才可访问。

若使用私有命名空间，需要设置访问凭据。

使用CCR镜像仓库

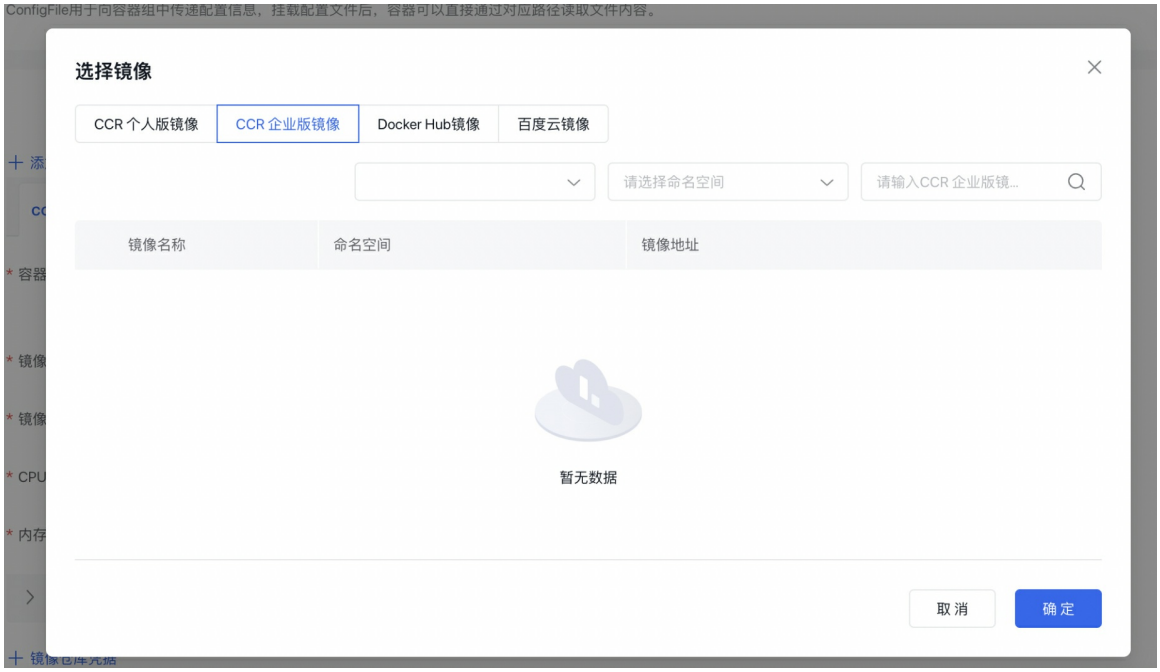
1.确定CCR企业版的网络访问方式：

- 通过公网访问，配置公网访问控制
- 通过VPC内访问，配置私有网络访问控制

在配置好访问控制后，使用相应的镜像仓库地址创建BCI实例即可

2.在容器配置中的“从仓库中选择”

选择CCR企业版镜像即可



使用第三方镜像仓库

使用第三方镜像仓库

BCI支持使用第三方镜像仓库，例如：自建镜像仓库，dockerhub及其他云厂商等第三方镜像仓库。

> **注意：**
>
> 访问第三方镜像仓库会产生额外的公网流量费用，请确定BCI实例的EIP功能已打开

使用

- BCI支持HTTP、HTTPS协议，并已经默认跳过TLS CA认证。
- 如私有镜像仓库需要登录的，方法如下：
  - 创建secret
    - kubectl create secret docker-registry --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
  - 使用secret

相关参数说明

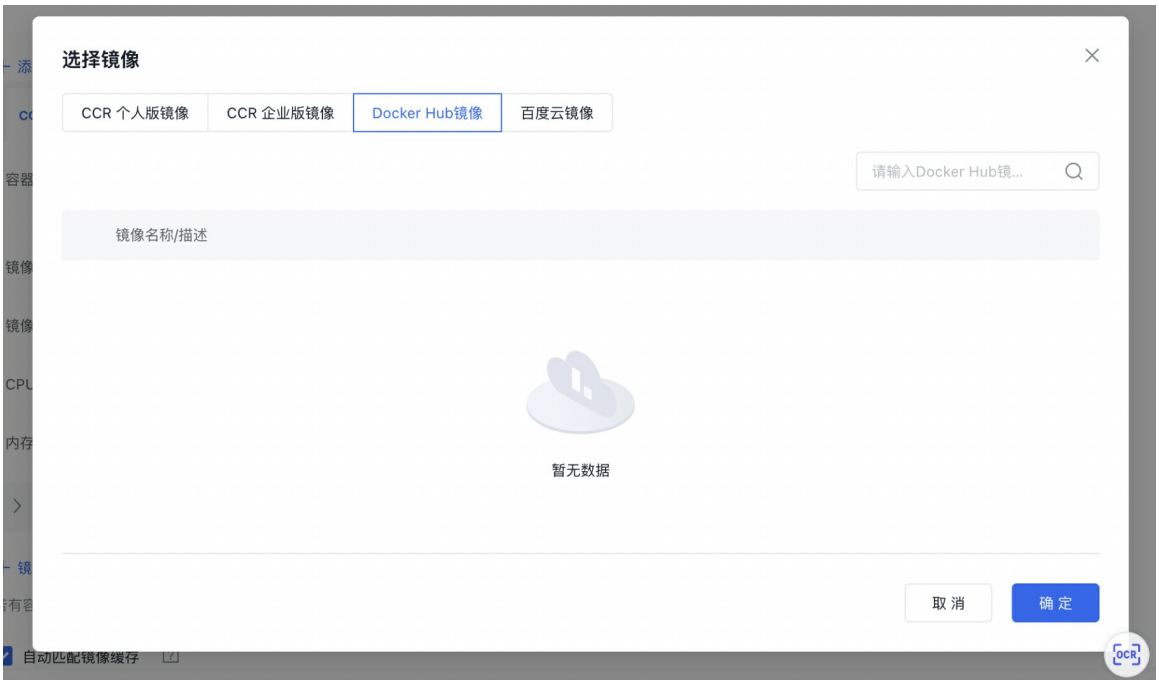
创建Secrect 参数	说明
name	secret的名称
DOCKER_REGISTRY_SERVER	镜像仓库地址
DOCKER_USER	镜像仓库用户名
DOCKER_PASSWORD	镜像仓库登录密码
DOCKER_EMAIL	邮箱

控制台使用 自建仓库

填写镜像地址、镜像版本、选择相应的CPU和内存并按要求填写镜像仓库凭证即可。

Docker Hub镜像仓库 下面从仓库中选择，可以使用Docker Hub镜像仓库。

从仓库中选择相关的镜像。



Docker Hub 镜像：Docker Hub是Docker的官方镜像库，它存储了大量的由社区用户和企业上传的Docker镜像。

网络

连接公网

连接公网

如果您的BCI实例（即BCI Pod）有连接公网的需求，则需要配置NAT网关或者弹性公网IP，并支付相应的网络费用。

本文介绍如何为您的BCI实例绑定EIP，或者为BCI实例所属的VPC绑定NAT网关，以实现BCI实例与公网互通。

背景信息

为BCI实例配置公网服务时，支持以下两种方式：

方式	说明	费用
BCI实例绑定EIP	EIP是独立购买的可单独持有的公网IP地址，可以绑定到BCI实例上提供公网服务。具体详情请参见：百度云EIP介绍。	EIP支持按固定带宽或者按使用流量计费。
BCI实例所属的VPC绑定NAT网关	NAT网关是可独立购买的网关产品，绑定EIP后，可以为关联VPC下的所有BCI实例提供公网服务。具体详情请参见：百度云NAT网关介绍。	NAT网关支持包年包月和按量付费。NAT网关需绑定EIP后才能具备公网能力，即除NAT网关费用外，您还需支付EIP费用。

您可以根据业务需要，选择合适的方式来配置公网服务：

- **示例场景一：单个BCI实例配置Nginx外网访问**

如果您有一个BCI实例用于部署Nginx服务，在创建实例时，您需要为该实例绑定EIP。当Nginx启动时，将自动暴露80端口到EIP。您可以通过EIP地址加端口的方式访问该BCI实例的Nginx服务。

- **示例场景二：多个BCI实例拉取Docker Hub镜像**

BCI默认不提供外部公网链路进行公网镜像的拉取。如果您有多个BCI实例需要从Docker Hub拉取镜像，您可以为BCI实例所属的VPC绑定NAT网关来实现公网访问，否则镜像将拉取失败。

> **注意：**
>
> 如果BCI实例已经绑定了EIP，则优先使用BCI实例绑定的EIP来访问公网，而不会使用NAT网关的SNAT功能访问公网。

操作指南

连接公网方式一：为BCI实例绑定EIP

创建BCI Pod时，您可以直接为Pod绑定EIP。

- 公网IP：选择购买弹性公网IP。
- 名称：填入公网IP名称。
- 公网带宽：
  1.按使用流量计费：后付费模式，按使用流量（单位为GB）计费，每小时扣费，请保证余额充足。
  2.按使用带宽计费：后付费模式，按分钟计费，每小时扣费，请保证余额充足。

连接公网方式二：为VPC绑定NAT网关

您可以在专有网络控制台为VPC绑定NAT网关，并为NAT网关绑定EIP，使其能够提供NAT代理（SNAT和DNAT）功能：

- SNAT功能：可以为VPC中没有公网IP的BCI实例提供访问公网的代理服务。
- DNAT功能：可以将NAT网关绑定的EIP映射给VPC中的BCI实例使用，使其能够面向公网提供服务。

操作步骤如下：

1. 登录私有网络VPC控制台。
2. 在顶部菜单栏，选择地域。
3. 在左侧菜单栏，选择“网络连接”->“NAT网关”，进入NAT网关页面，创建NAT网关。
    1. 单击创建NAT网关。
    2. 完成购买NAT网关相关的参数配置。配置时，“所在网络”选择BCI实例所属的VPC。更多信息，请参见NAT网关。
    3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。
4. 在弹性公网IP控制台，创建EIP。
    1. 在顶部菜单栏，选择地域后，单击“创建实例”。
    2. 完成购买EIP相关的参数配置。更多信息，请参见创建EIP实例。
    3. 单击“确认订单”后，确认配置信息和费用，再单击“提交订单”。
5. 绑定EIP与NAT网关。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“更多”->“绑定公网IP”。
    2. 在弹出的对话框中选择要绑定的EIP，然后单击确定。
6. 如果您的BCI实例需要访问公网，您需要为NAT网关创建SNAT条目，并在VPC中添加该NAT网关的路由。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置SNAT”。

2. 单击“添加SNAT条目”。
3. 配置SNAT相关条目的参数，然后单击确定。配置时，“源网段”选择BCI所属的子网。更多信息，请参见NAT网关的配置SNAT表。
4. 在私有网络页面，单击要操作的VPC名称进入VPC详情页，单击“路由表”，然后单击“添加路由”并配置路由参数。更多信息，请参见路由表的添加路由。

7. 如果您的BCI实例需要面向公网提供服务，您需要为NAT网关创建DNAT条目。
    1. 在NAT网关页面，找到要操作的NAT网关，单击右侧“设置DNAT”。
    2. 单击“添加DNAT条目”。
    3. 配置DNAT相关条目的参数，然后单击确定。更多信息，请参见NAT网关的配置DNAT表。

存储

挂载CFS数据卷

挂载CFS文件存储

文件存储CFS(Cloud File Storage)是百度智能云提供的安全、可扩展的文件存储服务。通过标准的文件访问协议，为云上的计算资源提供无限扩展、高可靠、全球共享的文件存储能力。本文为您介绍挂载CFS文件存储。更多CFS相关信息，请参见CFS说明。

前提条件

请确保您已经创建CFS并已获得CFS挂载地址（CFS Server）。

> **注意：**
>
> - CFS为共享存储，一个CFS可以挂载到多个BCI实例上。此时，如果多个BCI同时修改相同数据，请进行同步与冲突保护。
> - 在删除所有使用此挂载点的BCI实例前，请勿删除CFS挂载点，否则可能会造成操作系统无响应。

控制台操作

挂载EmptyDir数据卷

挂载EmptyDir数据卷

本文介绍如何挂载EmptyDir数据卷。EmptyDir数据卷是一个空的目录，用于临时存放数据，便于容器之间共享数据。

注意事项

EmptyDir为临时存储，当BCI实例删除或重启时，EmptyDir数据卷中保存的数据均会被清空。

> **提示：**
>
> 注意：当前暂不支持指定临时存储空间大小，不支持基于内存的临时存储。

声明数据卷

通过Volume相关参数声明数据卷时，需要先明确Volume的名称和类型。再根据Volume.N.Type的取值，进一步配置该类型数据卷的相关参数。

名称	类型	示例值	描述
Volume.N.Name	String	emptydir-demo	数据卷名称
Volume.N.Type	String	emptyDir	取值为EmptyDirVolume，表示创建一个EmptyDir类型的数据卷

挂载数据卷

声明数据卷后，可以通过VolumeMount相关参数将数据卷挂载到容器中。

名称	类型	示例值	描述
Container.N.VolumeMount.N.Name	String	test-volume	要挂载到容器的数据卷的名称，对应Volume.N.Name的值
Container.N.VolumeMount.N.MountPath	String	/usr/share	挂载目录。容器挂载目录下的内容会被数据卷的内容直接覆盖，请准确填写
Container.N.VolumeMount.N.ReadOnly	Boolean	false	挂载目录是否只读。默认为false。

控制台操作 在存储卷中选择Emptydir

输入Emptydir名称

按要求填写好数据卷名称即可。

挂载ConfigMap数据卷

挂载ConfigMap数据卷

本文介绍如何挂载ConfigMap数据卷。ConfigMap数据卷是一个配置文件，用于向BCI实例注入配置数据，具体详见create-a-configmap

> **注意：**

> BCI目前不支持configMap配置文件动态更新，以容器创建时配置为准。

控制台操作

按要求填入名称、Config File: * Path以及Config File即可

容器配置

设置容器启动命令和参数

设置容器启动命令和参数

BCI实例（即BCI Pod）通过容器镜像中的预设参数来启动容器。如果您在构建镜像时没有设置启动命令和参数，或者想要变更启动命令和参数，可以在创建BCI Pod时设置。本文介绍如何为容器设置启动时要执行的命令和参数。

功能说明

如果您想覆盖镜像中设置的启动默认值，包括工作目录、启动命令和参数，可以通过以下参数进行配置：

- 工作目录镜像构建时，通过WORKDIR可以指定容器的工作目录， 容器启动时执行的命令会在该目录下执行。更多信息，请参见WORKDIR。创建BCI实例时，通过配置BCI实例中容器的工作目录(WorkingDir)，可以覆盖WORKDIR。

> **注意：**
>
> 如果镜像里未指定WORKDIR，且创建BCI实例也未配置工作目录，则工作目录默认为根目录。 如果指定的工作目录不存在，系统将自动创建。

- 启动命令和参数镜像构建时，通过ENTRYPOINT和CMD可以指定启动容器后要执行的命令和参数。更多信息，请参见ENTRYPOINT和CMD。创建BCI实例时，通过配置BCI实例中容器的启动命令（Command）和参数（Arg），可以覆盖ENTRYPOINT和CMD。具体生效规则如下：

镜像ENTRYPOINT	镜像CMD	容器command	容器Arg	最终执行命令	说明
ls	/root/data	未设置	未设置	ls /root/data	容器的Command和Arg均未设置，则执行镜像ENTRYPOINT和CMD
ls	/root/data	cd	未设置	cd	容器设置了Command，未设置Arg，则只执行Command，忽略镜像ENTRYPOINT和CMD
ls	/root/data	未设置	/home/work	ls /home/work	容器设置了Arg，未设置Command，则执行镜像ENTRYPOINT和容器Arg
ls	/root/data	cd	/home/work	cd /home/work	同时设置了Command和Arg，则执行容器Command和Arg

> **注意：**
>
> 启动命令必须为容器镜像支持的命令，否则会导致容器启动失败。

控制台操作

可根据上面的描述填写相应的启动命令以及启动参数。

监控

查看实例监控指标

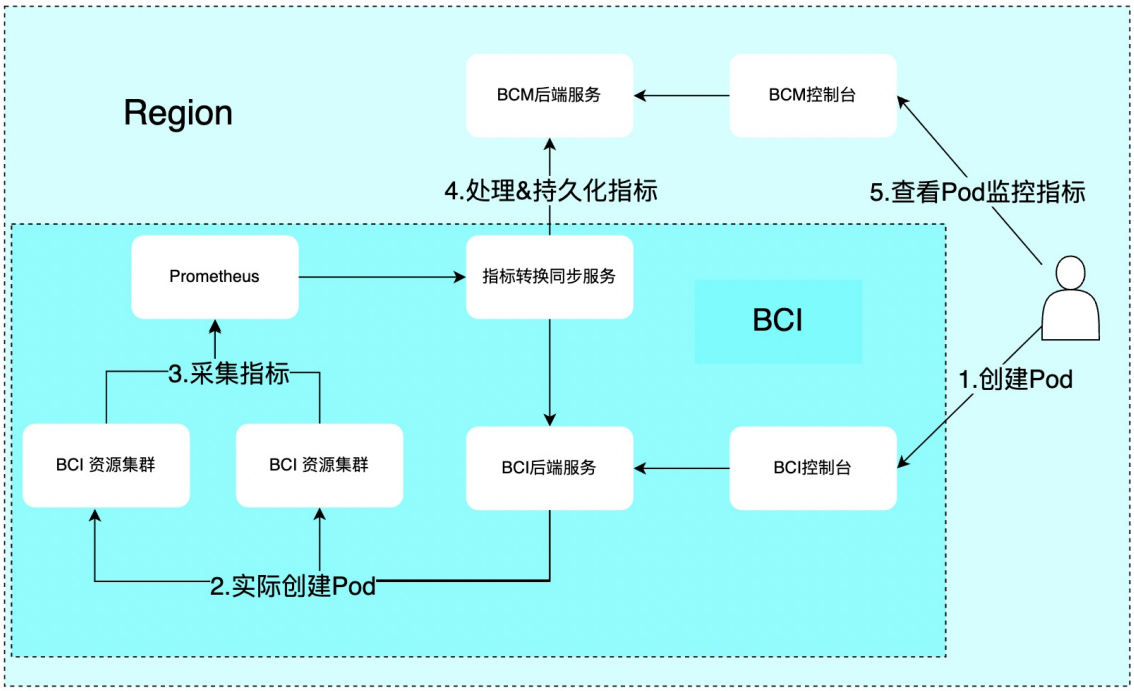
BCI支持自动采集实例的CPU、内存、网络和磁盘等相关监控指标，并可通过BCM（百度云监控）进行查看。

监控指标概述 BCI支持查看的监控指标如下： **CPU**

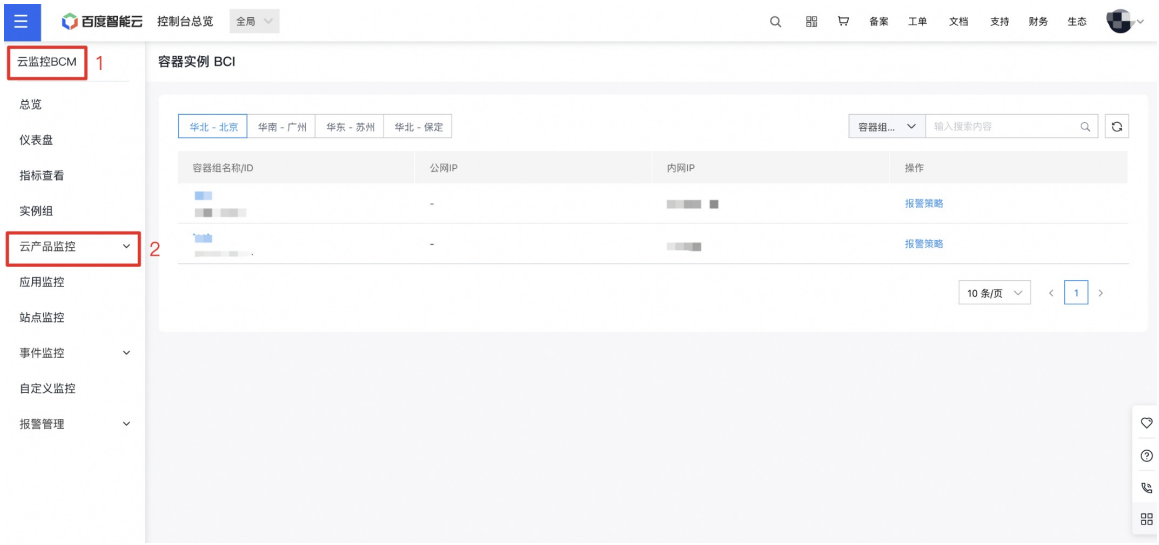
- CPU使用率：指标采集周期内，平均CPU使用率（单位：%）。CPU处于非IDLE状态即使用，CPU使用率上限为申请的CPU核数*100%。例如：如果申请了1.5 vCPU，且在指标采集窗口期内有50%的时间处于使用状态，则CPU使用率为75%。 **内存**
- 内存用量：指标采集时，正在使用的内存字节数（单位：字节）。使用的内存字节数不包含Page Cache占用的内存空间。 **网络**
- 网络接收量：指标采集周期内，每秒平均接收的网络数据比特数（单位：比特/秒）。
- 网络发送量：指标采集周期内，每秒平均发送的网络数据比特数（单位：比特/秒）。
- 网络接收包：指标采集周期内，每秒平均接收的网络IP包个数（单位：个/秒）。
- 网络发送包：指标采集周期内，每秒平均发送的网络IP包个数（单位：个/秒）。 **磁盘**
- 磁盘读取量：指标采集周期内，每秒平均从文件系统读取的字节数（单位：字节/秒）。指标名称中的磁盘，泛指以磁盘为代表的文件系统，从内存文件系统读取的数据量也会算入指标值中。下述磁盘相关指标的命名都有此含义。
- 磁盘写入量：指标采集周期内，每秒平均写到文件系统的字节数（单位：字节/秒）。
- 磁盘读取次数（暂未采集）：指标采集周期内，每秒平均发起文件系统读请求的次数（单位：次/秒）。

- 磁盘写入次数（暂未采集）：指标采集周期内，每秒平均发起文件系统写请求的次数（单位：次/秒）。

实例监控架构



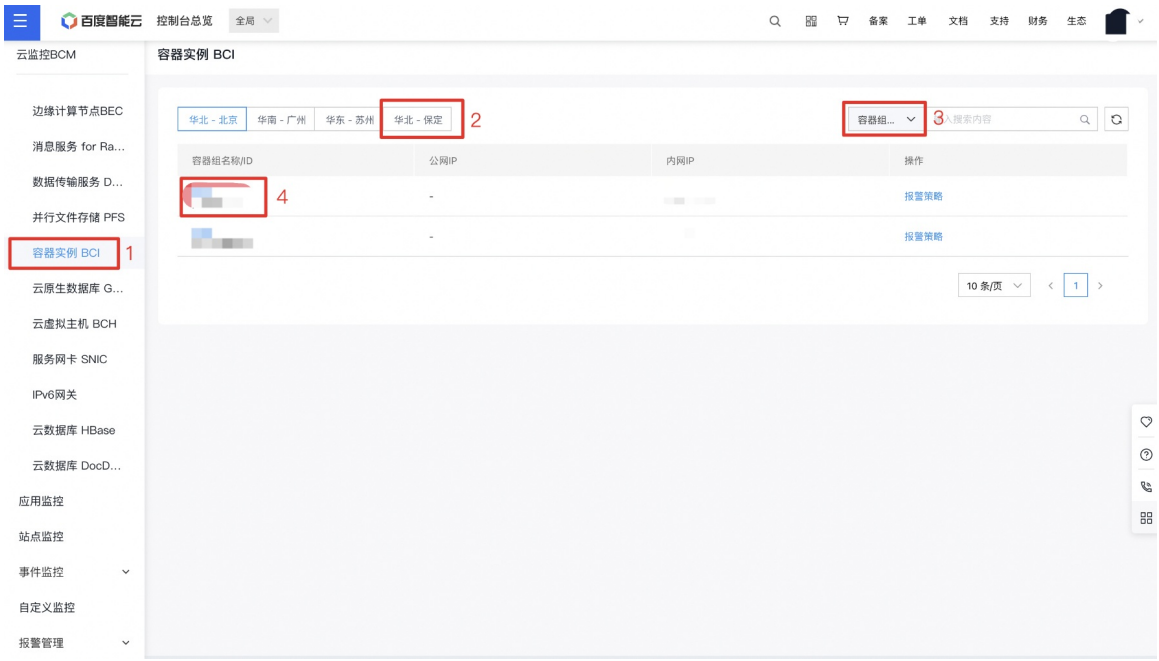
BCM查看监控指标 进入BCM产品页面



1.进入BCM产品页面

2.点击展开左侧『云产品监控』

选择容器实例 BCI



1.展开『云产品监控』后，选择『容器实例 BCI』

2.选择容器实例所在的 Region

3.选择查看容器组粒度的监控指标，或者容器粒度监控指标

4.点击要查看的容器组/容器

更多信息，请参考 云监控BCM操作指南。

多用户访问控制

概述

多用户访问控制(Identity and Access Management, IAM)，主要用于百度智能云的身份管理和访问控制，解决云账户的集中授权与管理、资源分享与多用户协同工作等问题。多用户访问控制适用于企业内的不同职能角色，你可以对不同员工赋予产品的不同权限，以共享你账户内的资源，完成他们的工作。当你的企业存在需要多用户协同工作、分享资源时，推荐你使用多用户访问控制。 以下是多用户访问控制适用的典型场景：

- 中大型企业客户：对公司内多个部门的不同员工进行集中资源和权限管理；
- 独立软件服务商(ISV)或SaaS平台商：对代理客户进行集中的资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源和权限管理。

创建用户

1. 主账号用户登录后在控制台右上角，选择“多用户访问控制”，进入“多用户访问控制”页面。
2. 在左侧导航栏点击“用户管理”，在“子用户”页，点击“创建子用户”。
3. 在弹出的“创建子用户”对话框中，完成填写“用户名”等信息，“确定”后返回“子用户管理”列表区可以查看到刚刚创建的子用户。

配置策略

BCI容器实例当前支持系统策略，实现产品级权限控制。 系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。

系统策略

本节说明接入IAM的产品系统策略，当前BCI系统策略包含只读权限、运维权限和管理权限。

策略名称	权限说明	权限范围
BCIFullControlAccessPolicy	完全控制管理容器实例(BCI)的权限	创建Pod、删除Pod、删除泄露Pod、与Pod建立WebShell链接 更新： * 为Pod开启日志推送 * 为Pod绑定EIP * 为Pod解绑EIP 查询： * 所有Pod * 指定时间后更新过的Pod * 指定Pod * 指定Pod的VPC信息 * 指定Pod的ConfigFile * 指定Pod的ID、资源以及状态信息 * 指定Pod事件 * 指定子网下Pod数量 * 指定UUID的Pod列表 * 指定容器日志 * 当前账户所有CCR镜像 * 当前账户的所有用户镜像 * 所有官方镜像 * 虚拟机相关配额 * 指定Pod详情（粗略版） * 当前账户容器列表（粗略版）
BCIOperateAccessPolicy	运维操作容器实例(BCI)的权限	与Pod建立WebShell链接 更新： * 为Pod开启日志推送 * 为Pod绑定EIP * 为Pod解绑EIP 查询： * 所有Pod * 指定时间后更新过的Pod * 指定Pod * 指定Pod的VPC信息 * 指定Pod的ConfigFile * 指定Pod的ID、资源以及状态信息 * 指定Pod事件 * 指定子网下Pod数量 * 指定UUID的Pod列表 * 指定容器日志 * 当前账户所有CCR镜像 * 当前账户的所有用户镜像 * 所有官方镜像 * 虚拟机相关配额 * 指定Pod详情（粗略版） * 当前账户容器列表（粗略版）
		查询： * 所有Pod * 指定时间后更新过的Pod

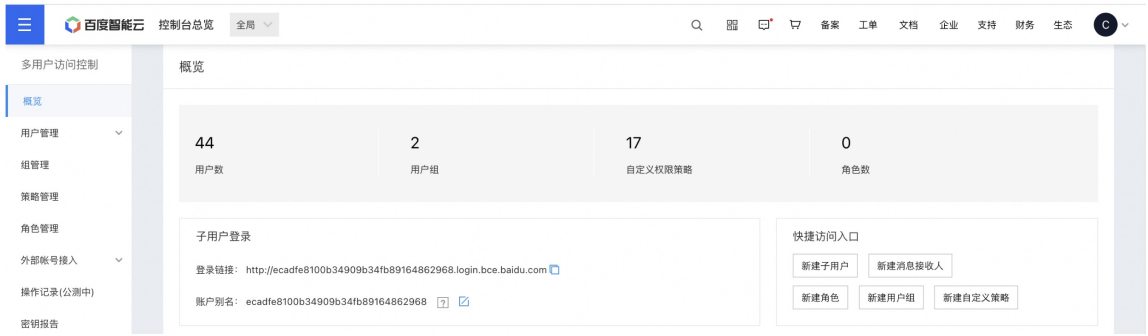
BCIReadAccessPolicy	只读访问容器实例(BCI)的权限	* 指定Pod * 指定Pod的VPC信息 * 指定Pod的ConfigFile * 指定Pod的ID、资源以及状态信息 * 指定Pod事件 * 指定子网下Pod数量 * 指定UUID的Pod列表 * 指定容器日志 * 当前账户所有CCR镜像 * 当前账户的所有用户镜像 * 所有官方镜像 * 虚拟机相关配额 * 指定Pod详情（粗略版） * 当前账户容器列表（粗略版）

用户授权

在“用户管理”下的“子用户”页面中，对应子用户的“操作”列选择“添加权限”，并为用户选择系统权限或自定义策略进行授权。
说明：如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。



其他详细操作参考：多用户访问控制。

运维

使用coredump分析实例程序异常

功能概述

coredump是指在程序运行过程中发生异常终止或崩溃时，操作系统将程序的内存内容转储到一个特殊的文件（即coredump文件）中，以便于后续的调试和分析。BCI容器实例支持例开启coredump运维任务，以便在容器异常终止时可以查看分析coredump生成的文件，从而定位问题原因，修复程序异常。 BCI支持将coredump文件存入对象存储BOS中。

使用流程

BCI默认关闭coredump，避免磁盘占用过多而导致业务不可用。您可以根据需要选择以下方式开启coredump运维任务。

方式一：支持手动开启coredump运维任务 手动通过OpenAPI开启coredump后，BCI将生成一个coredump运维任务,在容器运行异常终止或者退出时，触发coredump生成的core文件将自动保存到对象存储BOS中。

手动开启coredump运维任务 当前BCI仅支持通过OpenAPI开启运维任务。步骤如下

1. 确保BCI实例成功创建并处于Running状态，获取到podID
2. 为BCI实例开启coredump运维任务。调用OpenAPI创建coredump运维任务，接口需指定目标BCI实例，然后将OpsType设为coredump，OpsValue设为enable，设置保存coredump文件的BOS bucket地址，即可开启coredump运维任务。一个注意事项是指定BCI实例时，需确保目标BCI实例在创建时没有设置CorePattern Annotation。

```
 POST http://{{endpoint_bci}}/v2/opstask

示例request body:
{
   "podId":"pod-xxxx",
   "opsType":"coredump",
   "opsValue":"enable",
   "bucket":"xxxbucket"
}
```

3. 触发coredump。连接BCI实例，在容器内执行sleep 100命令后按Ctrl+\键，可触发coredump，生成的core文件将自动保存到BOS中。
4. 下载Core文件。调用OpenAPI查询运维任务接口，指定BCI实例podId和运维任务类型OpsType为coredump查询参数，查看运维任务的结果，从返回信息的result中，可以获取core文件是否已保存到BOS中，可到BOS相关bucket下载core文件。

```
 GET http://{{endpoint_bci}}/v2/opsrecord?podId=xxx&opsType=coredump

示例reponse
{
   "result": [
      {
         "opsType": "coredump",
         "opsStatus": "success",
         "storageType": "bos",
         "~~~~storageContent": "please goto bos page to download file core.p-ysappvy3.sh.8.1713786425 .",
         "createTime": 1713786437000
      }
   ]
}
```

API参考

概述

欢迎使用百度智能云产品——百度智能云容器实例BCI（Baidu Container Instance）。

百度智能云容器实例（Baidu Container Instance，即BCI）提供无服务器化的容器资源。您只需提供容器镜像及启动容器所需的配置参数，即可运行容器，而无需关心这些容器如何被调度部署到底层的物理服务器资源中。

BCI服务将为您完成IaaS层资源的调度和运维工作，从而简化您对容器的使用流程，降低部署和维护成本。同时BCI只会对您创建容器时申请的资源计费，因此实现真正的按需付费。结合容器本身秒级启动的特性，BCI可以帮助您在百度智能云上构建灵活弹性且易于维护的大规模容器集群。

BCI API 提供下列接口类型：

实例相关接口

接口类型	请求方式	描述
/v{version}/instance	POST	创建BCI实例
/v{version}/instance	GET	查询BCI实例列表
/v{version}/instance/{instanceId}	GET	查询BCI实例详情
/v{version}/instance/{instanceId}	DELETE	删除BCI实例
/v{version}/instance	DELETE	批量删除BCI实例

通用说明

API调用遵循HTTP协议，各Region采用不同的域名，具体域名为bci.{region}.baidubce.com。 数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

实名认证

使用BCI API的用户需要实名认证，没有通过实名认证的可以前往百度开放云官网控制台中的安全认证下的实名认证中进行认证。没有通过实名认证的用户请求将会得到以下错误提示码：

错误码	错误描述	HTTP状态码	中文解释
QualifyNotPass	The User has not pass qualify.	403	账号没有通过实名认证

API认证机制

所有API的安全认证一律采用Access Key与请求签名机制。 Access Key由Access Key ID和Secret Access Key组成，均为字符串。 对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version是正整数。
- timestamp是生成签名时的UTC时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。（本API签名仅使用host和x-bce-date两个header）
- signature是256位签名的十六进制表示，由64个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。

鉴权认证机制的详细内容请参见鉴权认证。

通信协议

BCI API支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体,如:POST /v{version}/instance/{instanceId}
Query参数	URL中携带的请求参数
HEADER	通过HTTP头域传入,如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

响应结构说明

响应值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL参数	API版本号，当前值为2。	是

幂等性

当调用创建接口时如果遇到了请求超时或服务器内部错误，用户可能会尝试重发请求，这时用户通过clientToken参数避免创建出比预期要多的资源，即保证请求的幂等性。

幂等性基于clientToken，clientToken是一个长度不超过64位的ASCII字符串，通常放在query string里，如`http://bci.bj.baidubce.com/v2/instance?clientToken=be31b98c-5e41-4838-9830-9be700de5a20`。

如果用户使用同一个clientToken值调用创建接口，则服务端会返回相同的请求结果。因此用户在遇到错误进行重试的时候，可以通过提供相同的clientToken值，来确保只创建一个资源；如果用户提供了一个已经使用过的clientToken，但其他请求参数（包括queryString和requestBody）不同甚至url Path不同，则会返回IdempotentParameterMismatch的错误代码。

clientToken的有效期为24小时，以服务端最后一次收到该clientToken为准。也就是说，如果客户端不断发送同一个clientToken，那么该clientToken将长期有效。

密码加密传输规范

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用AES 128位加密算法进行加密，用SK的前16位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

```
  byte[] bCiphertext= AES(明文,SK)
String strHex = HexStr(bCiphertext)
```

日期与时间规范

日期与时间的表示有多种方式。为统一起见，除非是约定俗成或者有相应规范的，凡需要日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

1. 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日
2. 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
3. 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

规范化字符串

通常一个字符串中可以包含任何Unicode字符。在编程中这种灵活性会带来不少困扰。因此引入“规范字符串”的概念。一个规范字符串只包含百分号编码字符以及URI（Uniform Resource Identifier）非保留字符（Unreserved Characters）。

RFC 3986规定URI非保留字符包括以下字符：字母（A-Z，a-z）、数字（0-9）、连字号（-）、点号（.）、下划线（_）、波浪线（~）。

将任意一个字符串转换为规范字符串的方式是:

将字符串转换成UTF-8编码的字节流。 保留所有URI非保留字符原样不变。 对其余字节做一次RFC 3986中规定的百分号编码（Percent-Encoding），即一个%后面跟着两个表示该字节值的十六进制字母。字母一律采用大写形式。

示例：

原字符串：`this is an example for` 测试, 对应的规范字符串：`this%20is%20an%20example%20for%20%E6%B5%8B%E8%AF%95`。

服务域名

API 的服务域名为：

Region	Endpoint	Protocol
北京	bci.bj.baidubce.com	HTTP and HTTPS
广州	bci.gz.baidubce.com	HTTP and HTTPS
苏州	bci.su.baidubce.com	HTTP and HTTPS
保定	bci.bd.baidubce.com	HTTP and HTTPS
武汉	bci.fwh.baidubce.com	HTTP and HTTPS

说明： API 支持 HTTP 和 HTTPS 两种调用方式。为了提升数据的安全性，建议通过 HTTPS 调用。

公共请求头与响应头

公共请求头

公共头部	描述
Authorization	包含Access Key与请求签名。具体请参考鉴权认证
Content-Type	application/json; charset=utf-8
x-bce-date	表示日期的字符串，符合API规范。具体请参考日期与时间规范

HTTP协议的标准头域不在这里列出。公共头域将在每个BCI API中出现，是必需的头域。POST、PUT、DELETE等请求数据放在request body中。

公共响应头

公共头部	描述
Content-Type	application/json; charset=utf-8。
x-bce-request-id	BCI后端生成，并自动设置到响应头域中。

其中，x-bce-request-id 使用UUID version4由BCI服务生成。

实例相关接口

创建实例

接口描述

创建一个BCI容器实例。

请求结构

```
POST /v{version}/instance HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	String	是	URL参数	API版本号，当前取值2。
clientToken	String	否	Query参数	保证请求幂等性。 从您的客户端生成一个参数值，确保不同请求间该参数值唯一。只支持ASCII字符，且不能超过64个字符。
name	String	是	RequestBody参数	BCI实例名称，即容器组名称。 长度必须介于2和252之间，有效的名称必须由字母数字字符、“-”或“.”组成，并且必须以字母数字字符开始和结束。 如果填写大写字母，后台会自动转为小写。
zoneName	String	否	RequestBody参数	可用区名称
securityGroupIds	List<String>	是	RequestBody参数	实例所属于的安全组ID。 所有安全组、子网应属于同一个VPC。 数量上限：10
subnetIds	List<String>	是	RequestBody参数	实例所属的子网ID。 所有子网应该属于同一个VPC，不可重复。 数量上限：10
restartPolicy	String	否	RequestBody参数	实例重启策略。 取值范围： Always：总是重启。 Never：从不重启。 OnFailure：失败时重启。 默认值：Always
eipIp	String	否	RequestBody参数	弹性公网IP
autoCreateEip	Boolean	否	RequestBody参数	是否自动创建一个EIP，并绑定到BCI实例上。 只有当eipIp为空的情况下，此字段才生效。 默认值：false
eipName	String	否	RequestBody参数	弹性公网名称。 当autoCreateEip为true时，此字段才生效。 默认值：eip
			Reques	EIP线路类型，包含标准BGP（BGP）和增强BGP（BGP_S）

eipRouteType	String	否	RequestBody参数	EIP线路类型，包含标准BGP（BGP）和增强BGP（BGP_S）。 当autoCreateEip为true时，此字段才生效。 默认值：BGP
eipBandwidthInMbps	Integer	否	RequestBody参数	公网带宽，单位：Mbps。 对于预付费以及按使用带宽计费的后付费EIP，标准型BGP限制为1~500之间的整数，增强型BGP限制为100~5000之间的整数（代表带宽上限）；对于按使用流量计费的后付费EIP， 标准型BGP限制为1~200之间的整数（代表允许的带宽流量峰值）。 如果填写浮点数会向下取整。 当autoCreateEip为true时，此字段才生效。 默认值：100
eipBillingMethod	String	否	RequestBody参数	计费方式，按流量（ByTraffic）、按带宽（ByBandwidth）、按增强95（ByPeak95）（只有共享带宽后付费支持）。 当autoCreateEip为true时，此字段才生效。 增强型BGP_S不支持按流量计费（ByTraffic），需要按带宽计费（ByBandwidth）。 默认值：ByTraffic
gpuType	String	否	RequestBody参数	实例所需的 GPU 资源型号。 目前仅支持：Nvidia A10 PCIE。
terminationGracePeriodSeconds	Long	否	RequestBody参数	程序的缓冲时间。 用于处理关闭之前的操作。
hostName	String	否	RequestBody参数	主机名称
tags	List<Tag>	否	RequestBody参数	用户标签列表
imageRegistryCredentials	List<ImageRegistryCredential>	否	RequestBody参数	镜像仓库凭证信息
containers	List<Container>	是	RequestBody参数	业务容器组
initContainers	List<Container>	否	RequestBody参数	Init 容器
volume	Volume	是	RequestBody参数	数据卷信息。 名称不可重复，有效名称必须由小写字母数字字符或“-”组成，并且必须以字母数字字符开头和结尾。

响应头域

除公共头域外，无其它特殊头域。

返回参数

参数	类型	描述
instanceId	String	BCI容器实例ID

请求示例

```
 POST /v2/instance HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "name": "mybci",
    "zoneName": "zoneC",
    "securityGroupIds": ["g-59gf44p4jmwe"],
    "subnetIds": ["sbn-g463qx0aqu7q"],
    "restartPolicy":"Always",
    "tags":[
        {
            "tagKey":"pod",
            "tagValue":"tag"
        }
    ],
    "imageRegistryCredentials": [
        {
            "server":"docker.io/wywcoder",
            "userName":"wywcoder",
            "password":"Qaz123456"
        }
    ],
    "containers":[
        {
            "name":"container01",
            "image":"registry.baidubce.com/bci-zjm-public/ubuntu:18.04",
            "memory":0.25,
            "cpu":0.25,
            "workingDir":"",
            "imagePullPolicy":"Always",
            "commands":[
                "/bin/sh",
                "-c",
                "sleep 30000 && exit 0"
            ],
            "args":[

            ],
            "volumeMounts":[
                {
                    "mountPath":"/usr/local/share",
                    "readOnly":false,
                    "name":"emptydir",
                    "type":"EmptyDir"
                },
                {
                    "mountPath":"/config",
                    "readOnly":false,
                    "name":"configfile",
                    "type":"ConfigFile"
                }
            ],
            "ports":[
```

```
            {
                "port":8099,
                "protocol":"TCP"
            }
        ],
        "environmentVars":[
            {
                "key":"java",
                "value":"/usr/local/jre"
            }
        ]
    }
  ],
  "volume":{
    "nfs":[],
    "emptyDir":[
      {
        "name":"emptydir"
      }
    ],
    "configFile":[
      {
        "name":"configfile",
        "configFiles":[
          {
            "path":"podconfig",
            "file":"filenxx"
          }
        ]
      }
    ]
  }
}
```

响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Tue, 06 Sep 2016 10:08:44 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
  "instanceId": "p-khyaaskt"
}
```

查询实例列表

接口描述

批量查询BCI实例列表。

请求结构

```
 GET /v{version}/instance HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	String	是	URL参数	API版本号，当前取值2。
keywordType	String	否	Query参数	查询关键字名称，取值范围：name、podId。
keyword	String	否	Query参数	查询关键字值。
marker	String	否	Query参数	表示下一个查询开始的marker。 说明：首次查询时无需设置该参数，后续查询的marker从返回结果中获取，返回结果中的marker为空表示没有下一个。
maxKeys	Integer	否	Query参数	每页包含的最大数量。 取值范围：[1, 1000]之间的正整数。 默认值：10

响应头域

除公共头域外，无其它特殊头域。

返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置
isTruncated	Boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	获取下一页所需要传递的marker值。当isTruncated为false时，该域不出现
maxKeys	Integer	每页包含的最大数量
result	List<InstanceModel>	BCI实例信息列表

请求示例

```
 GET /v{version}/instance?maxKeys=5 HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Tue, 06 Sep 2016 10:08:44 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "marker": "p-rpjlzbuu",
    "isTruncated": true,
    "nextMarker": "p-qsyvnuzg",
    "maxKeys": 5,
```

```
  "result": [
    {
      "instanceId": "p-rpjlzbuu",
      "instanceName": "default-eni-test-558cfc6dfc-st8sg-1",
      "status": "Running",
      "zoneName": "",
      "cpuType": "intel",
      "gpuType": "",
      "cpu": 0.25,
      "memory": 0.5,
      "bandwidthInMbps": 0,
      "publicIp": "",
      "internalIp": "192.168.64.157",
      "createTime": "2023-03-02T05:55:43Z",
      "updateTime": "2023-03-02T14:19:45Z",
      "deleteTime": null,
      "restartPolicy": "Always",
      "tags": null
    },
    {
      "instanceId": "p-iwafloba",
      "instanceName": "default-eni-test-558cfc6dfc-qwcjq-1",
      "status": "Running",
      "zoneName": "",
      "cpuType": "intel",
      "gpuType": "",
      "cpu": 0.25,
      "memory": 0.5,
      "bandwidthInMbps": 0,
      "publicIp": "",
      "internalIp": "192.168.64.88",
      "createTime": "2023-03-02T05:55:43Z",
      "updateTime": "2023-03-02T14:19:45Z",
      "deleteTime": null,
      "restartPolicy": "Always",
      "tags": null
    },
    {
      "instanceId": "p-7314w80l",
      "instanceName": "default-eni-test-558cfc6dfc-lcztq-1",
      "status": "Running",
      "zoneName": "",
      "cpuType": "intel",
      "gpuType": "",
      "cpu": 0.25,
      "memory": 0.5,
      "bandwidthInMbps": 0,
      "publicIp": "",
      "internalIp": "192.168.64.132",
      "createTime": "2023-03-02T05:55:57Z",
      "updateTime": "2023-03-02T14:19:45Z",
      "deleteTime": null,
      "restartPolicy": "Always",
      "tags": null
    },
    {
      "instanceId": "p-bk399zaw",
      "instanceName": "default-eni-test-558cfc6dfc-hktl9-1",
      "status": "Running",
      "zoneName": "",
      "cpuType": "intel",
      "gpuType": "",
      "cpu": 0.25,
```

```
            "cpu": 0.25,
            "memory": 0.5,
            "bandwidthInMbps": 0,
            "publicIp": "",
            "internalIp": "192.168.64.191",
            "createTime": "2023-03-02T05:55:57Z",
            "updateTime": "2023-03-02T14:19:45Z",
            "deleteTime": null,
            "restartPolicy": "Always",
            "tags": null
        },
        {
            "instanceId": "p-fhunfwdn",
            "instanceName": "default-eni-test-558cfc6dfc-5fds8-1",
            "status": "Running",
            "zoneName": "",
            "cpuType": "intel",
            "gpuType": "",
            "cpu": 0.25,
            "memory": 0.5,
            "bandwidthInMbps": 0,
            "publicIp": "",
            "internalIp": "192.168.64.85",
            "createTime": "2023-03-02T05:55:57Z",
            "updateTime": "2023-03-02T14:19:45Z",
            "deleteTime": null,
            "restartPolicy": "Always",
            "tags": null
        }
    ]
}
```


查询实例详情

接口描述

查询BCI实例详情

请求结构

```
 GET /v{version}/instance/{instanceId} HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	String	是	URL参数	API版本号，当前取值2
instanceId	String	是	URL参数	BCI实例ID

响应头域

除公共头域外，无其它特殊头域。

返回参数

参数名称	类型	描述
instance	InstanceDetailModel	BCI实例详细信息

请求示例

```
 GET /v{version}/instance/p-rpjlzbuu HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Tue, 06 Sep 2016 10:08:44 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "instance": {
        "instanceId": "p-rpjlzbuu",
        "instanceName": "default-eni-test-558cfc6dfc-st8sg-1",
        "status": "Running",
        "zoneName": "",
        "cpuType": "intel",
        "gpuType": "",
        "cpu": 0.25,
        "memory": 0.5,
        "bandwidthInMbps": 0,
        "publicIp": "",
        "internalIp": "192.168.64.157",
        "createTime": "2023-03-02T05:55:43Z",
        "updateTime": "2023-03-02T14:19:45Z",
        "deleteTime": null,
        "restartPolicy": "Always",
        "tags": null,
        "volume": {
            "nfs": [],
            "emptyDir": [],
            "configFile": [],
            "podVolumes": null,
            "flexVolume": []
        },
        "containers": [
            {
                "name": "security-group",
                "image": "registry.baidubce.com/cce-plugin-dev/netperf:v1.0.0",
                "cpu": 0.25,
                "memory": 0.5,
                "gpu": 0.0,
                "workingDir": "",
                "imagePullPolicy": "Always",
                "commands": [
                    "/bin/sh",
                    "-c",
                    "sleep 3600"
                ],
                "args": [],
```

```
            "ports": [],
            "volumeMounts": [],
            "envs": [
              {
                "key": "SERVICE_ZHTEST_PORT",
                "value": "tcp://172.16.52.63:80",
                "valueFrom": null
              }
            ],
            "createTime": "2023-03-02T05:55:43Z",
            "updateTime": "2023-03-02T14:19:45Z",
            "deleteTime": null,
            "currentState": null,
            "previousState": {
              "state": "Succeeded",
              "reason": null,
              "message": null,
              "startTime": "2023-03-02T12:56:16Z",
              "finishTime": "2023-03-02T13:56:16Z",
              "detailStatus": "Completed",
              "exitCode": 0
            },
            "ready": true,
            "restartCount": 8
          }
        ],
        "initContainers": [],
        "securityGroups": [
          {
            "securityGroupId": "g-59gf44p4jmwe",
            "name": "默认安全组",
            "description": "default",
            "vpcId": "7cfef2de-e17b-4362-885e-4c291e3a9163"
          }
        ],
        "vpc": {
          "vpcId": "vpc-rfjkdgxphqcm",
          "name": "zhanghong-vpc",
          "cidr": "192.168.0.0/16",
          "createTime": "Wed Mar 16 08:40:43 UTC 2022",
          "description": "",
          "isDefault": false
        },
        "subnet": {
          "subnetId": "sbn-kbmujmezgzyd",
          "name": "zrq-eni",
          "cidr": "192.168.64.0/24",
          "vpcId": "vpc-rfjkdgxphqcm",
          "subnetType": "BCC",
          "description": "",
          "createTime": "Wed Sep 14 09:51:08 UTC 2022"
        },
        "terminationGracePeriodSeconds": null
      }
    }
```


删除实例

接口描述

删除一个BCI容器实例，删除后其使用的物理资源可被收回，比如EIP。

请求结构

```
DELETE /v{version}/instance/{instanceId} HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	String	是	URL参数	API版本号，当前取值2
instanceId	String	是	URL参数	待删除的BCI实例ID
relatedReleaseFlag	Boolean	否	Query参数	释放关联资源，目前只有EIP资源，默认值为false

响应头域

除公共头域外，无其它特殊头域。

返回参数

无特殊返回参数

请求示例

```
DELETE /v2/instance/p-u1snow2n?relatedReleaseFlag=true HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Tue, 06 Sep 2016 10:08:44 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{}
```

批量删除实例

接口描述

批量删除BCI容器实例，删除后其使用的物理资源可被收回，比如EIP等资源。

请求结构

```
POST /v2/instance/batchDel HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	String	是	URL参数	API版本号，当前取值2。
instanceIds	List<String>	是	RequestBody参数	待删除的BCI实例ID列表。
relatedReleaseFlag	Boolean	否	RequestBody参数	释放关联资源，目前只有EIP资源。 默认值：false

响应头域

除公共头域外，无其它特殊头域。

返回参数

无特殊返回参数

请求示例

```
 POST /v2/instance/batchDel HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de

{
    "instanceIds": [
        "p-u1snow2n",
        "p-3f26zqfs"
    ],
    "relatedReleaseFlag": true
}
```

响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Tue, 06 Sep 2016 10:08:44 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{}
```

镜像缓存相关接口

创建镜像缓存

接口描述

此接口用于创建一个BCI镜像缓存，配置包括临时存储大小、自动匹配镜像缓存、原始镜像信息、子网 ID、安全组 ID、弹性公网 IP、镜像缓存名称、区域名称、是否需要弹性公网 IP 以及镜像仓库凭据。

请求结构

```
POST /v{version}/imageCache HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

请求体（Body）包含以下参数：

参数名	类型	是否必须	描述
temporaryStorageSize	Integer	是	临时存储大小（单位：GB）
autoMatchImageCache	Boolean	否	是否自动匹配镜像缓存
originImages	List<OriginImage>	是	原始镜像数组，每个对象包含镜像地址和版本信息
subnetId	String	是	子网 ID
securityGroupId	String	是	安全组 ID
eipIp	String	否	弹性公网 IP 地址
imageCacheName	String	是	镜像缓存名称
zoneName	String	是	区域名称
needEip	Boolean	是	是否需要弹性公网 IP
imageRegistrySecrets	List<ImageRegistryCredential>	否	镜像仓库凭据，每个对象包含服务器地址、用户名和密码

请求示例

```
POST /v2/imageCache
Content-Type: application/json
Authorization: Bearer <YOUR_ACCESS_TOKEN>

{
  "temporaryStorageSize": 20,
  "autoMatchImageCache": true,
  "originImages": [
    {
      "originImageAddress": "dasda",
      "originImageVersion": "dada"
    }
  ],
  "subnetId": "sbn-xxx",
  "securityGroupId": "g-yyy",
  "eipIp": "10.10.10.10",
  "imageCacheName": "dasda",
  "zoneName": "zoneB",
  "needEip": false,
  "imageRegistrySecrets": [
    {
      "server": "http://dsada",
      "userName": "dasdd",
      "password": "dasdad"
    }
  ]
}
```

响应示例

成功响应

- **HTTP 状态码**：200 OK
- **Body**：

```
{
  "imageCacheId": "xxxxx"
}
```

错误响应

- **HTTP 状态码**：400 Bad Request
- **Body**：

```
{
 "code": "Invalid parameters",
 "message": "One or more parameters are missing or invalid."
}
```

或

- **HTTP 状态码**：401 Unauthorized
- **Body**：

```
{
 "code": "Unauthorized",
 "message": "You are not authorized to create resources."
}
```

注意事项

- 确保请求体中的所有必须字段都有值，且数据类型正确。
- 如果 needEip 设置为 true，则 eipIp 字段必须提供。
- 确保 Authorization 头部中的访问令牌是有效的。

查询镜像缓存列表

接口描述

此接口用于查询BCI镜像缓存列表。

请求结构

```
GET /v{version}/imageCache HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
version	Integer	是	URL参数	API版本号，当前取值2
pageSize	Integer	否	Query参数	每页包含的最大数量。取值范围：[1，1000]之间的正整数。默认值：10。
pageNo	Integer	否	Query参数	当前要查询的页。取值范围：大于0的正整数。默认值：1。

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	描述
totalCount	Integer	当前用户空间下镜像缓存数目。
pageSize	Integer	当前查询页。
pageNo	Integer	当前查询页的镜像缓存条目数量。
result	List<ImageCacheModel>	镜像缓存信息列表。

请求示例

```
 GET /v2/imageCache?pageNo=4&pageSize=10 HTTP/1.1
Host: bci.bj.baidubce.com
ContentType: application/json
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-
11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
 HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Wed, 08 Jul 2015 03:28:11 GMT
x-bce-request-id: d8752367-38e8-45e4-b4c7-e53be3137ce5
Server: BWS
{
    "totalCount": 32,
    "result": [
        {
            "imageCacheId": "00dc1b52d8354d9193536e4dd2c41ae6faf9d193a03ed5d7ce24d43a0f55e218-failed-
n1dm5cos",
            "originImages": [
                "registry.baidubce.com/ubuntu:18.04"
            ],
            "status": "创建失败",
            "progress": 0,
            "expiredTime": "2023-07-12T18:05:29Z",
            "createdTime": "2023-06-12T10:04:22Z",
            "lastestMatchedTime": "2023-06-12T18:05:29Z"
        },
        {
            "imageCacheId": "00dc1b52d8354d9193536e4dd2c41ae643249601b601ac6f6c6dec81a528cb92-failed-
rly8lror",
            "originImages": [
                "registry.baidubce.com/gohttp:v0.0.1"
            ],
            "status": "创建失败",
            "progress": 60,
            "expiredTime": "2023-07-12T11:24:21Z",
            "createdTime": "2023-06-12T03:21:54Z",
            "lastestMatchedTime": "2023-06-12T11:24:21Z"
        }
    ],
    "pageSize": 4,
    "pageNo": 10
}
```

批量删除镜像缓存

接口描述

此接口用于单个或批量删除BCI镜像缓存。

请求结构

```
 POST /v{version}/imageCache/batchDel HTTP/1.1
Host: bci.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	描述
imageCacheIds	List< String >	是	需要被删除的镜像缓存ID列表

请求示例

```
 POST /v2/imageCache/batchDel
Content-Type: application/json
Authorization: Bearer <YOUR_ACCESS_TOKEN>

{
  "imageCacheIds": [
    xxx
  ]
}
```

响应示例

成功响应

- **HTTP 状态码**：200
- **Body**：

```
 {
}
```

错误响应

- **HTTP 状态码**：400 Bad Request
- **Body**：

```
 {
   "requestId": "c9b4e202-dfd9-gray-bci0-bcecanarytag",
   "code": "BadRequest",
   "message": "Bad request parameters or illegal request"
}
```

错误码

请求发生错误时通过respone body返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	错误码
message	String	错误描述
requestId	String	本次请求的requestId

示例：

```
 {
   "requestId" : "ae2225f7-1c2e-427a-a1ad-5413b762957d",
   "code" : "NoSuchKey",
   "message" : "The resource you requested does not exist"
}
```

BCE公共错误码

下表列出了百度智能云API的公共错误码。

错误码	错误描述	HTTP状	说明

错误码	错误描述	态码	语义
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或者值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error. Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided does not exist in our records.	403 Forbidden	Access Key ID不存在
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	400 Bad Request	Authorization头域格式错误
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配等问题，应定义更加具体的错误码，例如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 Bad Request	JSON格式不合法
InvalidVersion	The API version specified was invalid.	404 Not Found	URI的版本号不合法
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务
PreconditionFailed	The specified If-Match header doesn't match the ETag header.	412 Precondition Failed	ETag不匹配
RequestExpired	Request has expired. Timestamp date is XXX.	400 Bad Request	请求超时
IdempotentPara	The request uses the same client token as a	403	

meterMismatch	previous, but non-identical request.	Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致

BCI错误码

错误码	错误描述	HTTP状态码	语义
Pod.NameInvalid	%s	400	Pod名称不合法，详细信息见错误描述。
Pod.LogicalZoneInvalid	%s	400	可用区信息不合法，详细信息见错误描述。
Pod.SecurityGroupInvalid	%s	400	安全组信息不合法，详细信息见错误描述。
Pod.SubnetInvalid	%s	400	子网信息不合法，详细信息见错误描述。
Pod.RestartPolicyInvalid	RestartPolicy is invalid. Only Always/OnFailure/Never are supported.	400	Pod重启策略不合法。
Pod.CreateEipFailed	%s	400	创建EIP失败，详细信息见错误描述。
Pod.EipAreadyInUse	Eip already be in use by %s.	400	EIP已经被其他实例使用。
Pod.EipNotAvailable	Eip status should be available.	400	EIP不可用。
Pod.EipOperationDenied	Account has no permission to bind EIP.	403	账户权限不足。
Pod.EipRouteTypeInvalid	Eip routetype is invalid.	400	EIP线路类型不合法。
Pod.EipBandwidthInMbpsInvalid	%s	400	EIP带宽信息不合法，详细信息见错误描述。
Pod.EipCountInvalid	%s	400	EIP数量不合法，详细信息见错误描述。
Pod.CpuMemSpecInvalid	The pod cpu and memory should in valid specification.	400	Pod的CPU或memory规格不合法。
Pod.GPUTypeInvalid	%s	400	GPU类型不合法，详细信息见错误描述。
Pod.GPUCountInvalid	The container's gpuCount should be greater than 0.	400	容器的GPU数量应大于0。
Pod.TagInvalid	%s	400	Tag不合法，详细信息见错误描述。
Pod.ImageRegistryInvalid	Pod ImageRegistry is invalid.	400	Pod的镜像仓库凭证信息不合法。
Pod.ContainerMountTypeInvalid	%s	400	数据卷挂载类型不合法，详细信息见错误描述。
Pod.ContainerInvalid	%s	400	容器信息不合法，详细信息见错误描述。

Pod.ContainerNameInvalid	%s	400	容器名称不合法，详细信息见错误描述。
Pod.ContainerImagePullPolicyInvalid	The container imagePullPolicy is invalid. Only Always/IfNotPresent/Never are supported.	400	容器的镜像拉取策略不合法。
Pod.ContainerWorkingDirInvalid	The container workingDir is invalid, a valid workingDir length must be between 1 and 1024.	400	容器工作目录不合法，合法的工作目录长度必须介于1和1024之间。
Pod.ContainerCommandsInvalid	The container commands is invalid, a valid commands length must be between 1 and 4096.	400	容器启动命令不合法，合法的启动命令长度必须介于1和4096之间。
Pod.ContainerProbeInvalid	The container probe is invalid. Only exec/tcpSocket/httpGet are supported.	400	容器探针不合法。
Pod.ContainerEnvironmentInvalid	The container environment is invalid. Only status.podIP or status.podIPs are supported.	400	容器环境变量不合法。
Pod.ContainerSecurityContextCapInvalid	The container securityContext is invalid, capability [%s] is forbidden.	400	容器的安全信息上下文不合法。
Pod.VolumeNameInvalid	%s	400	数据卷名称不合法，详细信息见错误描述。
Pod.VolumeMountsInvalid	The volume mount [%s] does not match any volume.	400	数据卷挂载信息不合法。

附录

基础类型

Tag

参数	类型	是否必选	描述
tagKey	String	否	标签名 不可重复，有效的tag key必须由字母数字字符、'-'、'.'组成，并且必须以字母数字字符开始和结束。
tagValue	String	否	标签值

Label

参数	类型	是否必选	描述
labelKey	String	否	标签名
labelValue	String	否	标签值

Port

参数	类型	是否必选	描述
name	String	否	端口名
port	Integer	否	端口号
protocol	String	否	协议类型。 取值范围： TCP UDP

HTTPHeader

参数	类型	是否必选	描述
name	String	否	http header name
value	String	否	http header value

Environment

参数	类型	是否必选	描述
key	String	否	环境变量名
value	String	否	环境变量值

ExecAction

参数	类型	是否必选	描述
command	List<String>	否	使用命令行方式进行健康检查时，在容器内执行的命令。

TCPSocketAction

参数	类型	是否必选	描述
port	Integer	否	使用TCP Socket方式进行健康检查时，TCP Socket检测的端口。
host	String	否	使用TCP Socket方式进行健康检查时，TCP Socket检测host。

HTTPGetAction

参数	类型	是否必选	描述
path	String	否	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的路径。
port	Integer	否	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的端口号。
scheme	String	否	使用HTTP请求方式进行健康检查时，HTTP Get请求对应的协议类型，取值范围： HTTP HTTPS
host	String	否	使用HTTP请求方式进行健康检查时，host值。
httpHeaders	List<HTTPHeader>	否	使用HTTP请求方式进行健康检查时，http header。

GRPCAction

参数	类型	是否必选	描述
port	Integer	否	使用GRPC方式进行健康检查时，GRPC检测的端口。
service	String	否	使用GRPC方式进行健康检查时，GRPC检测服务。

实例相关

Probe

参数	类型	是否必选	描述
initialDelaySeconds	Integer	否	检查开始执行的时间，以容器启动完成为起点计算。
timeoutSeconds	Integer	否	检查超时的时间，默认为1秒，最小为1秒。
periodSeconds	Integer	否	检查执行的周期，默认为10秒，最小为1秒。
successThreshold	Integer	否	从上次检查失败后重新认定检查成功的检查次数阈值（必须是连续成功），默认为1。当前必须为1。
failureThreshold	Integer	否	从上次检查成功后认定检查失败的检查次数阈值（必须是连续失败），默认为3。
terminationGracePeriodSeconds	Long	否	程序的缓冲时间，用于处理关闭之前的操作。
httpGet	HTTPGetAction	否	HttpGet检测参数。
tcpSocket	TCPSocketAction	否	TcpSocket检测的端口参数。
exec	ExecAction	否	容器内检测命令参数。
grpc	GRPCAction	否	grpc检测的端口参数。

Container

参数	类型	是否必选	描述
name	String	是	容器名 长度必须介于1和40之间，不可重复，有效的名称必须由字母数字字符、“-”或“.”组成，并且必须以字母数字字符开始和结束。
image	String	是	镜像
memory	Float	否	内存，单位：GiB。 限制该容器最多可使用的内存值，该值不可超过容器实例的总内存值。
cpu	Float	否	CPU，单位：核。 限制容器最多可使用的核数，该值不可超过容器实例的总核数。
gpu	Float	否	容器使用的GPU个数
workingDir	String	否	容器工作目录
imagePullPolicy	String	否	镜像拉取策略。 取值范围： Always：总是拉取。每次都拉取镜像。 IfNotPresent：按需拉取。优先使用本地镜像，本地没有镜像时则拉取镜像。 Never：从不拉取。使用本地镜像，不拉取镜像。
commands	List<String>	是	容器启动命令
args	List<String>	否	容器启动参数
volumeMounts	List<VolumeMount>	是	数据卷挂载信息
ports	List<Port>	否	容器内端口信息
environmentVars	List<Environment>	否	容器内操作系统的环境变量
livenessProbe	Probe	否	存活探针
readinessProbe	Probe	否	就绪探针
startupProbe	Probe	否	启动探针
stdin	Boolean	否	此容器是否应在容器运行时为标准输入分配缓冲区。 如果未设置，则容器中标准输入的读取值将导致EOF。 默认值：false
stdinOnce	Boolean	否	当标准输入为true时，标准输入流将在多个附加会话中是否保持开启状态。默认值：false
tty	Boolean	否	是否开启交互。 当Command为/bin/bash命令时，需要设置为true。 默认值：false
securityContext	ContainerSecurityContext	否	容器安全上下文信息。

InstanceModel

参数	类型	描述
instanceId	String	实例ID
instanceName	String	实例名称，即容器组名称。
status	String	实例状态。可能值：状态信息不标准。 • Pending：启动中 • Running：运行中 • Succeeded：运行成功 • Failed：运行失败
zoneName	String	实例所属的可用区
cpuType	String	cpu类型
gpuType	String	gpu类型
cpu	Float	cpu核数,单位核
memory	Float	memory大小，单位GiB
bandwidthInMbps	Integer	弹性公网IP带宽大小，单位为Mb。
internalIp	String	内网Ip
publicIp	String	外网Ip
createTime	String	接到请求后的系统创建时间。UTC时间，RFC3339标准。
updateTime	String	实例更新时间
deleteTime	String	实例删除时间
restartPolicy	String	容器组的重启策略。取值范围： • Never：不重启 • Always：始终重启 • OnFailure：失败时重启
tags	List<Tag>	实例的标签键值对

InstanceDetailModel

继承InstanceModel，并新增如下字段：

参数	类型	描述
volume	Volume	容器实例存储卷
containers	List<ContainerDetailModel>	业务容器
initContainers	List<ContainerDetailModel>	init容器
securityGroups	List<SecurityGroupModel>	安全组信息
vpc	VpcModel	vpc信息
subnet	SubnetModel	子网信息

ContainerDetailModel

参数	类型	描述
name	String	容器名称
image	String	容器镜像
cpu	Float	cpu数量
gpu	Float	gpu数量
memory	Float	内存大小
workingDir	String	容器工作目录
imagePullPolicy	String	镜像拉取策略
commands	List<String>	容器启动命令
args	List<String>	容器启动参数
ports	List<Port>	容器内端口信息
volumeMounts	List<VolumeMount>	容器存储卷信息
envs	List<Environment>	容器环境变量列表
createTime	String	容器创建时间
updateTime	String	容器更新时间
deleteTime	String	容器删除时间
previousState	ContainerStatus	容器上一次状态
currentState	ContainerStatus	容器当前状态
ready	Boolean	容器是否已通过其就绪探针
restartCount	Integer	重启次数

ContainerStatus

参数	类型	描述
state	String	容器状态。可能值： • Waiting：启动中 • Running：运行中 • Terminated：运行终止
reason	String	容器状态Reason
message	String	容器状态信息
startTime	String	容器运行开始时间
finishTime	String	容器运行结束时间
detailStatus	String	状态详情
exitCode	Integer	容器运行退出码

SecurityGroupModel

参数	类型	描述
securityGroupId	String	安全组 Id
name	String	安全组名称
description	String	安全组描述
vpcId	String	vpc Id

SubnetModel

参数	类型	描述
subnetId	String	子网ID
name	String	子网名称
cidr	String	网段及子网掩码
vpcId	String	vpcID
subnetType	String	子网类型 • BCC • BBC • BCC_NAT • BBC_NAT
description	String	subnet描述
createTime	String	创建时间

VpcModel

参数	类型	描述
vpcId	String	vpc Id
name	String	vpc名称
cidr	String	网段及子网掩码
createTime	String	创建时间
description	String	vpc描述
isDefault	Boolean	是否是默认vpc

存储卷相关

Volume

参数	类型	是否必选	描述
nfs	List<NfsVolume>	否	NFS类型数据卷，表示网络文件系统
emptyDir	List<EmptyDirVolume>	否	EmptyDir类型数据卷，表示空目录
configFile	List<ConfigFileVolume>	否	ConfigFile类型数据卷，表示配置文件

NfsVolume

参数	类型	是否必选	描述
name	String	是	NFS服务器名称。
server	String	是	NFS服务器地址。
path	String	是	NFS数据卷路径。
readOnly	Boolean	否	NFS数据卷是否只读，默认为false。

EmptyDirVolume

参数	类型	是否必选	描述
name	String	是	EmptyDirVolume名称。
medium	String	否	EmptyDirVolume的存储媒介，默认为空，使用node文件系统；支持 memory，表示使用内存。
sizeLimit	Float	否	EmptyDirVolume的大小。单位为GiB。

ConfigFileVolume

参数	类型	是否必选	描述
name	String	是	ConfigFileVolume名称
defaultMode	Integer	是	ConfigFileVolume默认的权限。
configFiles	List<ConfigFileDetail>	是	ConfigFile类型的数据卷信息。

ConfigFileDetail

参数	类型	是否必选	描述
path	String	是	ConfigFile文件路径
file	String	是	ConfigFile文件名

VolumeMount

参数	类型	是否必选	描述
name	String	是	数据卷名称
type	String	是	数据卷类型
mountPath	String	是	容器挂载数据卷的目录
readOnly	Boolean	否	数据卷是否只读 默认值：false

镜像相关

ImageRegistryCredential

参数	类型	是否必选	描述
server	String	否	镜像仓库注册地址
userName	String	否	镜像仓库用户名
password	String	否	镜像仓库密码

OriginImage

字段名	类型	是否必选	描述
originImageAddress	String	否	原始镜像地址
originImageVersion	String	否	原始镜像版本

ImageCacheModel

参数	类型	描述
imageCacheId	String	镜像缓存id，全局唯一
originImages	List<String>	用户原始镜像地址
status	String	镜像缓存制作状态。可能状态为：创建成功，创建失败，创建中
progress	Integer	镜像缓存制作进度，范围[0,100]的正整数
expiredTime	TimeStamp	镜像缓存超时回收时间
createdTime	TimeStamp	镜像缓存创建时间
lastestMatchedTime	TimeStamp	镜像缓存最近一次使用时间

安全特性相关

ContainerSecurityContext

参数	类型	是否必选	描述
capabilities	Capabilities	否	安全特性能力
runAsUser	Long	否	运行容器的用户ID
runAsGroup	Long	否	运行容器进程入口点的GID
runAsNonRoot	Boolean	否	容器是否必须以非root用户运行
readOnlyRootFilesystem	Boolean	否	容器运行的根文件系统是否为只读，默认为false

Capabilities

参数	类型	是否必选	描述
add	List<String>	否	启用安全能力项列表
drop	List<String>	否	禁用安全能力项列表

SDK

Go-SDK

概述

本文档主要介绍BCI GO SDK的使用。在使用本文档前，您需要先了解BCI的一些基本知识，并已开通了BCI服务。若您还不了解BCI，可以参考产品描述和操作指南。

安装SDK工具包

运行环境

GO SDK可以在go1.3及以上环境下运行。

安装SDK

直接从github下载

使用go get工具从github进行下载：

```
go get github.com/baidubce/bce-sdk-go
```

SDK目录结构

```
bce-sdk-go
  |--auth                 //BCE签名和权限认证
  |--bce                  //BCE公用基础组件
  |--http                 //BCE的http通信模块
  |--services               //BCE相关服务目录
  |  |--bci               //BCI
  |--util                 //BCE公用的工具实现
```

卸载SDK

预期卸载SDK时，删除下载的源码即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BCI服务域名的部分，理解Endpoint相关的概念。百度云目前开放了多区域支持，请参考区域选择说明。

目前支持“华北-北京”、“华南-广州”、“华东-苏州”和“华北-保定”四个区域。对应信息为：

访问区域	对应Endpoint	协议
华北-北京	bci.bj.baidubce.com	HTTP and HTTPS
华南-广州	bci.gz.baidubce.com	HTTP and HTTPS
华东-苏州	bci.su.baidubce.com	HTTP and HTTPS
华北-保定	bci.bd.baidubce.com	HTTP and HTTPS
华中-武汉	bci.fwh.baidubce.com	HTTP and HTTPS

获取密钥

要使用百度云BCI，您需要拥有一个有效的AK(Access Key ID)和SK(Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问BCI做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

注册百度云账号

创建AK/SK

新建BCI Client

BCI Client是BCI服务的客户端，为开发者与BCI服务进行交互提供了一系列的方法。

使用AK/SK新建BCI Client

通过AK/SK方式访问BCI，用户可以参考如下代码新建一个BCI Client：

```
 import (
"github.com/baidubce/bce-sdk-go/services/bci"
)

func main() {
 // 用户的Access Key ID和Secret Access Key
 ACCESS_KEY_ID, SECRET_ACCESS_KEY := <your-access-key-id>, <your-secret-access-key>

 // 用户指定的Endpoint
 ENDPOINT := <domain-name>

 // 初始化一个BCIClient
 bciClient, err := bci.NewClient(AK, SK, ENDPOINT)
}
```

在上面代码中，`ACCESS_KEY_ID`对应控制台中的“Access Key ID”，`SECRET_ACCESS_KEY`对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 如何获取AKSK》。第三个参数`ENDPOINT`支持用户自己指定域名，如果设置为空字符串，会使用默认域名作为BCI的服务地址。

> **注意：**`ENDPOINT`参数需要用指定区域的域名来进行定义，如服务所在区域为北京，则为`bci.bj.baidubce.com`。

使用STS创建BCI Client

申请STS token

BCI可以通过STS机制实现第三方的临时授权访问。STS（Security Token Service）是百度云提供的临时授权服务。通过STS，您可以为第三方用户颁发一个自定义时效和权限的访问凭证。第三方用户可以使用该访问凭证直接调用百度云的API或SDK访问百度云资源。

通过STS方式访问BCI，用户需要先通过STS的client申请一个认证字符串。

用STS token新建BCI Client

申请好STS后，可将STS Token配置到BCI Client中，从而实现通过STS Token创建BCI Client。

代码示例

GO SDK实现了STS服务的接口，用户可以参考如下完整代码，实现申请STS Token和创建BCI Client对象：

```
 import (
 "fmt"

 "github.com/baidubce/bce-sdk-go/auth"        //导入认证模块
 "github.com/baidubce/bce-sdk-go/services/bci" //导入BCI服务模块
 "github.com/baidubce/bce-sdk-go/services/sts" //导入STS服务模块
)

func main() {
 // 创建STS服务的Client对象，Endpoint使用默认值
 AK, SK := <your-access-key-id>, <your-secret-access-key>
 stsClient, err := sts.NewClient(AK, SK)
 if err != nil {
  fmt.Println("create sts client object :", err)
  return
 }

 // 获取临时认证token，有效期为60秒，ACL为空
 stsObj, err := stsClient.GetSessionToken(60, "")
 if err != nil {
  fmt.Println("get session token failed:", err)
  return
    }
 fmt.Println("GetSessionToken result:")
 fmt.Println("  accessKeyId:", stsObj.AccessKeyId)
 fmt.Println("  secretAccessKey:", stsObj.SecretAccessKey)
 fmt.Println("  sessionToken:", stsObj.SessionToken)
 fmt.Println("  createTime:", stsObj.CreateTime)
 fmt.Println("  expiration:", stsObj.Expiration)
 fmt.Println("  userId:", stsObj.UserId)

 // 使用申请的临时STS创建BCI服务的Client对象，Endpoint使用默认值
 bciClient, err := bci.NewClient(stsObj.AccessKeyId, stsObj.SecretAccessKey, "bci.bj.baidubce.com")
 if err != nil {
  fmt.Println("create bci client failed:", err)
  return
 }
 stsCredential, err := auth.NewSessionBceCredentials(
  stsObj.AccessKeyId,
  stsObj.SecretAccessKey,
  stsObj.SessionToken)
 if err != nil {
  fmt.Println("create sts credential object failed:", err)
  return
 }
 bciClient.Config.Credentials = stsCredential
}
```

> 注意： 目前使用STS配置BCI Client时，无论对应BCI服务的Endpoint在哪里，STS的Endpoint都需配置为http://sts.bj.baidubce.com。 上述代码中创建STS对象时使用此默认值。

配置HTTPS协议访问BCI

BCI支持HTTPS传输协议，您可以通过在创建BCI Client对象时指定的Endpoint中指明HTTPS的方式，在BCI GO SDK中使用HTTPS访问BCI服务：

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

ENDPOINT := "https://bci.bj.baidubce.com" //指明使用HTTPS协议
AK, SK := <your-access-key-id>, <your-secret-access-key>
bciClient, _ := bci.NewClient(AK, SK, ENDPOINT)
```

配置BCI Client

如果用户需要配置BCI Client的一些细节的参数，可以在创建BCI Client对象之后，使用该对象的导出字段Config进行自定义配置，可以为客户端配置代理，最大连接数等参数。

使用代理

下面一段代码可以让客户端使用代理访问BCI服务：

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

//创建BCI Client对象
AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bci.bj.baidubce.com"
client, _ := bci.NewClient(AK, SK, ENDPOINT)

//代理使用本地的8080端口
client.Config.ProxyUrl = "127.0.0.1:8080"
```

设置网络参数

用户可以通过如下的示例代码进行网络参数的设置：

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bci.bj.baidubce.com"
client, _ := bci.NewClient(AK, SK, ENDPOINT)

// 配置不进行重试，默认为Back Off重试
client.Config.Retry = bce.NewNoRetryPolicy()

// 配置连接超时时间为30秒
client.Config.ConnectionTimeoutInMillis = 30 * 1000
```

配置生成签名字符串选项

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bci.bj.baidubce.com"
client, _ := bci.NewClient(AK, SK, ENDPOINT)

// 配置签名使用的HTTP请求头为`Host`
headersToSign := map[string]struct{}{"Host": struct{}{}}
client.Config.SignOption.HeadersToSign = HeadersToSign

// 配置签名的有效期为30秒
client.Config.SignOption.ExpireSeconds = 30
```

参数说明

用户使用GO SDK访问BCI时，创建的BCI Client对象的Config字段支持的所有参数如下表所示：

配置项名称	类型	含义
Endpoint	string	请求服务的域名
ProxyUrl	string	客户端请求的代理地址
Region	string	请求资源的区域
UserAgent	string	用户名称，HTTP请求的User-Agent头
Credentials	*auth.BceCredentials	请求的鉴权对象，分为普通AK/SK与STS两种
SignOption	*auth.SignOptions	认证字符串签名选项
Retry	RetryPolicy	连接重试策略
ConnectionTimeoutInMillis	int	连接超时时间，单位毫秒，默认20分钟

说明：

1. Credentials字段使用auth.NewBceCredentials与auth.NewSessionBceCredentials函数创建，默认使用前者，后者为使用STS鉴权时使用，详见“**使用STS创建BCI Client**”小节。
2. SignOption字段为生成签名字符串时的选项，详见下表说明：

名称	类型	含义
HeadersToSign	map[string]struct{}	生成签名字符串时使用的HTTP头
Timestamp	int64	生成的签名字符串中使用的时间戳，默认使用请求发送时的值
ExpireSeconds	int	签名字符串的有效期

其中，HeadersToSign默认为`Host`，`Content-Type`，`Content-Length`，`Content-MD5`；TimeStamp一般为零值，表示使用调用生成认证字符串时的时间戳，用户一般不应该明确指定该字段的值；ExpireSeconds默认为1800秒即30分钟。

3. Retry字段指定重试策略，目前支持两种：NoRetryPolicy和BackOffRetryPolicy。默认使用后者，该重试策略是指定最大重试次数、最长重试时间和重试基数，按照重试基数乘以2的指数级增长的方式进行重试，直到达到最大重试测试或者最长重试时间为止。

异常处理

错误处理

GO语言以error类型标识错误，BCI支持两种错误见下表：

错误类型	说明
BceClientError	用户操作产生的错误
BceServiceError	BCI服务返回的错误

用户使用SDK调用BCI相关接口，除了返回所需的结果之外还会返回错误，用户可以获取相关错误进行处理。实例如下：

```
// client 为已创建的BCI Client对象
args := &bci.ListInstanceArgs{}
result, err := client.ListInstances(args)
if err != nil {
    switch realErr := err.(type) {
    case *bce.BceClientError:
        fmt.Println("client occurs error:", realErr.Error())
    case *bce.BceServiceError:
        fmt.Println("service occurs error:", realErr.Error())
    default:
        fmt.Println("unknown error:", err)
    }
}
```

客户端异常

客户端异常表示客户端尝试向BCI发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用时，则会返回BceClientError。

服务端异常

当BCI服务端出现异常时，BCI服务端会返回给用户相应的错误信息，以便定位问题。常见服务端异常可参见BCI错误码

SDK日志

BCI GO SDK支持六个级别、三种输出（标准输出、标准错误、文件）、基本格式设置的日志模块，导入路径为`github.com/baidubce/bce-sdk-go/util/log`。输出为文件时支持设置五种日志滚动方式（不滚动、按天、按小时、按分钟、按大小），此时还需设置输出日志文件的目录。

默认日志

BCI GO SDK自身使用包级别的全局日志对象，该对象默认情况下不记录日志，如果需要输出SDK相关日志需要用户自定指定输出方式和级别，详见如下示例：

```
// import "github.com/baidubce/bce-sdk-go/util/log"

// 指定输出到标准错误，输出INFO及以上级别
log.SetLogHandler(log.STDERR)
log.SetLogLevel(log.INFO)

// 指定输出到标准错误和文件，DEBUG及以上级别，以1GB文件大小进行滚动
log.SetLogHandler(log.STDERR | log.FILE)
log.SetLogDir("/tmp/gosdk-log")
log.SetRotateType(log.ROTATE_SIZE)
log.SetRotateSize(1 << 30)

// 输出到标准输出，仅输出级别和日志消息
log.SetLogHandler(log.STDOUT)
log.SetLogFormat([]string{log.FMT_LEVEL, log.FMT_MSG})
```

说明：

1. 日志默认输出级别为`DEBUG`
2. 如果设置为输出到文件，默认日志输出目录为`/tmp`，默认按小时滚动
3. 如果设置为输出到文件且按大小滚动，默认滚动大小为1GB
4. 默认的日志输出格式为：`FMT_LEVEL, FMT_LTIME, FMT_LOCATION, FMT_MSG`

BCI实例

创建实例

使用以下代码可以创建一个BCI实例。

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

args := &CreateInstanceArgs{
    // 保证请求幂等性
    ClientToken: "random-uuid",
    // BCI实例名称
    Name: "instanceName",
    // 可用区名称
    ZoneName: "zoneC",
    // 实例所属于的安全组Id
    SecurityGroupIds: []string{"g-59gf44p4jmwe"},
    // 实例所属的子网Id
    SubnetIds: []string{"sbn-g463qx0aqu7q"},
    // 实例重启策略
    RestartPolicy: "Always",
    // 弹性公网IP
    EipIp: "106.13.234.xx",
    // 自动创建EIP
    AutoCreateEip: false,
    // 弹性公网名称
    EipName: "zwj-test-eip",
    // EIP线路类型
    EipRouteType: "BGP",
    // 公网带宽，单位为Mbps
    EipBandwidthInMbps: 1,
    // 计费方式
    EipBillingMethod: "ByTraffic",
    // 实例所需的 GPU 资源型号
    GPUType: "Nvidia A10 PCIE",
    // 程序的缓冲时间，用于处理关闭之前的操作
    TerminationGracePeriodSeconds: 0,
    // 主机名称
    HostName: "zwj-go-sdktest",
    // 用户标签列表
    Tags: []Tag{
        {
            TagKey:   "appName",
            TagValue: "zwj-test",
        },
    },
    // 镜像仓库凭证信息
    ImageRegistryCredentials: []ImageRegistryCredential{
        {
            Server:   "docker.io/wywcoder",
            UserName: "wywcoder",
            Password: "Qaz123456",
        },
    },
    // 业务容器组
    Containers: []Container{
        {
            Name:          "container01",
            Image:         "registry.baidubce.com/bci-zjm-public/ubuntu:18.04",
            Memory:        0.5,
            CPU:           0.25,
            GPU:           0
```

```
            WorkingDir:      "",
            ImagePullPolicy: "IfNotPresent",
            Commands:        []string{"/bin/sh"},
            Args:            []string{"-c", "sleep 30000 && exit 0"},
            VolumeMounts: []VolumeMount{
                {
                    MountPath: "/usr/local/nfs",
                    ReadOnly:  false,
                    Name:      "nfs",
                    Type:      "NFS",
                },
                {
                    MountPath: "/usr/local/share",
                    ReadOnly:  false,
                    Name:      "emptydir",
                    Type:      "EmptyDir",
                },
                {
                    MountPath: "/config",
                    ReadOnly:  false,
                    Name:      "configfile",
                    Type:      "ConfigFile",
                },
            },
            Ports: []Port{
                {
                    Port:     8099,
                    Protocol: "TCP",
                },
            },
            EnvironmentVars: []Environment{
                {
                    Key:   "java",
                    Value: "/usr/local/jre",
                },
            },
            LivenessProbe: &Probe{
                InitialDelaySeconds:           0,
                TimeoutSeconds:                0,
                PeriodSeconds:                 0,
                SuccessThreshold:              0,
                FailureThreshold:              0,
                TerminationGracePeriodSeconds: 0,
                Exec: &ExecAction{
                    Command: []string{"echo 0"},
                },
            },
            Stdin:          false,
            StdinOnce:      false,
            Tty:            false,
            SecurityContext: &ContainerSecurityContext{},
        },
    },
    // Init 容器
    InitContainers: []Container{},
    // 数据卷信息
    Volume: &Volume{
        Nfs: []NfsVolume{
            {
                Name:   "nfs",
                Server: "xxx.cfs.gz.baidubce.com",
                Path:   "/",
```

```
            },
        },
        EmptyDir: []EmptyDirVolume{
            {
                Name: "emptydir",
            },
        },
        ConfigFile: []ConfigFileVolume{
            {
                Name: "configfile",
                ConfigFiles: []ConfigFileDetail{
                    {
                        Path: "podconfig",
                        File: "filenxx",
                    },
                },
            },
        },
    },
}
result, err := client.CreateInstance(args)
if err != nil {
    fmt.Printf("CreateInstance error: %+v \n", err)
    return
}
fmt.Printf("CreateInstance success, bci instance id: %+v \n", result.InstanceId)
```

注意:

- 保证请求幂等性。从您的客户端生成一个参数值，确保不同请求间该参数值唯一。只支持ASCII字符，且不能超过64个字符。
- BCI实例名称，即容器组名称；支持长度为2~253个英文小写字母、数字或者连字符（-），不能以连接字符开始或结尾。如果填写大写字母，后台会自动转为小写。
- 实例重启策略。取值范围：Always：总是重启，Never：从不重启，OnFailure：失败时重启。默认值：Always。
- 自动创建EIP，并绑定到BCI实例上。只有当eipIp为空的情况下，此字段才生效。默认值为：false。
- EIP线路类型，包含标准BGP（BGP）和增强BGP（BGP_S），默认标准BGP。当autoCreateEip为true时，此字段才生效。默认值为：BGP。
- 公网带宽，单位为Mbps。对于预付费以及按使用带宽计费的后付费EIP，标准型BGP限制为1~500之间的整数，增强型BGP限制为100~5000之间的整数（代表带宽上限）；对于按使用流量计费的后付费EIP，标准型BGP限制为1~200之间的整数（代表允许的带宽流量峰值）。如果填写浮点数会向下取整。当autoCreateEip为true时，此字段才生效。默认值为100。
- 计费方式，按流量（ByTraffic）、按带宽（ByBandwidth）、按增强95（ByPeak95）（只有共享带宽后付费支持）。当autoCreateEip为true时，此字段才生效。增强型BGP_S不支持按流量计费（ByTraffic），需要按带宽计费（ByBandwidth）。默认值为ByTraffic。
- 实例所需的 GPU 资源型号。目前仅支持：Nvidia A10 PCIE。

查询实例列表

使用以下代码可以查询BCI实例列表。

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

args := &ListInstanceArgs{
    // 查询关键字名称
    KeywordType:    "podId",
    // 查询关键字值
    keyword:        "p-xxx",
    // 表示下一个查询开始的marker，marker为空表示没有下一个
    Marker:         "",
    // 每页包含的最大数量
    MaxKeys:        5,
}
result, err := client.ListInstances(args)
fmt.Printf("ListInstances result: %+v, err: %+v \n", result, err)
```

注意:

- 查询关键字名称，取值范围：name、podId。
- 表示下一个查询开始的marker，marker为空表示没有下一个。说明：首次查询时无需设置该参数，后续查询的marker从返回结果中获取。
- 每页包含的最大数量，最大数量通常不超过1000，缺省值为10。maxKeys的取值范围：[1, 1000]之间的正整数。

查询实例详情

使用以下代码可以查询BCI实例详情。

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

args := &GetInstanceArgs{
    // BCI实例ID
    InstanceId: "p-xxx",
}
result, err := client.GetInstance(args)
fmt.Printf("ListInstances result: %+v, err: %+v \n", result, err)
```

删除实例

使用以下代码可以删除BCI实例。

```
// import "github.com/baidubce/bce-sdk-go/services/bci"

args := &DeleteInstanceArgs{
    // 待删除的BCI实例ID
    InstanceId:         "p-xxxx",
    // 释放关联资源
    RelatedReleaseFlag: true,
}
err := client.DeleteInstance(args)
fmt.Printf("DeleteInstance err: %+v\n", err)
```

注意:

- 释放关联资源，目前只有EIP资源，默认值为false。

批量删除实例

使用以下代码可以批量删除BCI实例。

```
 // import "github.com/baidubce/bce-sdk-go/services/bci"

args := &BatchDeleteInstanceArgs{
    // 待删除的BCI实例ID列表
    InstanceIds:        []string{"p-axxx", "p-bxxx"},
    // 释放关联资源
    RelatedReleaseFlag: true,
}
err := client.BatchDeleteInstance(args)
fmt.Printf("BatchDeleteInstance err: %+v\n", err)
```

> 注意:
>
> - 释放关联资源，目前只有EIP资源，默认值为false。

Java-SDK

概述

本文档主要介绍BCI Java SDK的安装和使用。在使用本文档前，您需要先了解BCI的一些基本知识，若您还不了解BCI，可以参考产品描述。

安装SDK工具包

运行环境

Java SDK工具包可在jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以SDK下载页面找到。

方式二：直接使用JAR包安装

1. 下载最新版Java SDK压缩工具包。
2. 将下载的bce-java-sdk-version.zip解压后，复制到工程文件夹中。
3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。
4. 添加SDK工具包lib/bce-java-sdk-{version}.jar和第三方依赖工具包third-party/*.jar。 其中，{version}为版本号，可以在SDK下载页面找到。

SDK目录结构

```
com.baidubce
├─── auth                          //BCE签名相关类
├─── http                          //BCE的Http通信相关类
├─── internal                      //SDK内部类
├─── model                         //BCE公用model类
├─── services
│    └── bci                      //BCI服务相关类
│        ├── model                 //BCI内部model，如Request或Response
│        ├── BciClient.class       //BCI客户端入口类
│        └── BciClientConfiguration.class    //针对BCI特有的HttpClient的配置
├─── util                          //BCE公用工具类
├─── BceClientConfiguration.class  //对BCE的HttpClient的配置
├─── BceClientException.class      //BCE客户端的异常类
├─── BceServiceException.class     //与BCE服务端交互后的异常类
├─── ErrorCode.class               //BCE通用的错误码
└─── Region.class                  //BCE提供服务的区域
```

卸载SDK

预期卸载 SDK 时，删除 pom 依赖或源码即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先参考阅读API参考中关于 API服务域名 的部分，理解Endpoint相关的概念。 百度智能云目前开放了多区域支持，请参考区域选择说明。 对应信息为：

访问区域	对应Endpoint
北京	bci.bj.baidubce.com
广州	bci.gz.baidubce.com
苏州	bci.su.baidubce.com
保定	bci.bd.baidubce.com
武汉	bci.fwh.baidubce.com

获取密钥

要使用百度智能云BCI，您需要拥有一个有效的 AK（Access Key ID）和SK(Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问BCI做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

新建BciClient

BciClient是BCI服务的客户端，为开发者与BCI服务进行交互提供了一系列的方法。

使用AK/SK新建BciClient

通过AK/SK方式访问BCI，用户可以参考如下代码新建一个BciClient：

```
 public class Sample {
public static void main(String[] args) {
    String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的    Access Key ID
    String SECRET_ACCESS_KEY = <your-secret-access-key>;          // 用户的Secret Access Key
    String ENDPOINT = <domain-name>;                            // 用户自己指定的域名

    BciClientConfiguration config = new BciClientConfiguration();
    config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
    config.setEndpoint(ENDPOINT);
    BciClient client = new BciClient(config);
}
```

使用STS创建BciClient

申请STS token

BCI可以通过STS机制实现第三方的临时授权访问。STS（Security Token Service）是百度智能云提供的临时授权服务。通过STS，您可以为第三方用户颁发一个自定义时效和权限的访问凭证。第三方用户可以使用该访问凭证直接调用百度智能云的API或SDK访问百度智能云资源。

通过STS方式访问BCI，用户需要先通过STS的client申请一个认证字符串，申请方式可参见百度智能云STS使用介绍。

用STS token新建BciClient

申请好STS后，可将STStoken配置到BciClient中，用户可以参考如下代码新建一个BciClient：

```
public class StsExample {
    private static final String STS_ENDPOINT = "http://sts.bj.baidubce.com";
    private static final String ENDPOINT = "bci.bj.baidubce.com";
    private static final String ACCESS_KEY_ID = "your accesskey id";
    private static final String SECRET_ACCESS_KEY = "your secret accesskey";

    public static void main(String[] args) {
        BceCredentials credentials = new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY);
        StsClient client = new StsClient(
                new BceClientConfiguration().withEndpoint(STS_ENDPOINT).withCredentials(credentials)
        );
        GetSessionTokenResponse response = client.getSessionToken(new GetSessionTokenRequest());
        // or simply call:
        // GetSessionTokenResponse response = client.getSessionToken();
        // or you can specify limited permissions with ACL:
        // GetSessionTokenResponse response = client.getSessionToken(new
GetSessionTokenRequest().withAcl("blabla"));
        // build DefaultBceSessionCredentials object from response:
        BceCredentials bcistsCredentials = new DefaultBceSessionCredentials(
                response.getAccessKeyId(),
                response.getSecretAccessKey(),
                response.getSessionToken());
        System.out.println("===================================");
        System.out.println("GetSessionToken result:");
        System.out.println("    accessKeyId:  " + response.getAccessKeyId());
        System.out.println("    secretAccessKey:  " + response.getSecretAccessKey());
        System.out.println("    securityToken:  " + response.getSessionToken());
        System.out.println("    expiresAt:  " + response.getExpiration().toString());
        System.out.println("===================================");

        // build bci client
        BciClientConfiguration config = new BciClientConfiguration();
        config.setCredentials(bcistsCredentials);
        config.setEndpoint(ENDPOINT);
        BciClient bciClient = new BciClient(config);
    }
}
```

> 注意： 目前使用STS配置client时，无论对应bucket的区域在哪里，endpoint都需配置为`http://sts.bj.baidubce.com`， 但创建BciClient时，仍需使用BCI的endpoint，如`bci.bj.baidubce.com`、`bci.su.baidubce.com`等.

配置自定义域名访问BCI

使用自定义域名 如果希望使用自定义域名作为访问BCI的endpoint，在控制台将自定义域名和BCI某个bucket绑定之后，配置endpoint为自定义域名并打开CnameEnabled开关，例如cdn-test.cdn.bcebci.com，配置代码如下:

```
String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;        // 用户的Secret Access Key
String ENDPOINT = "https://cdn-test.cdn.bcebci.com";        // 用户自己指定的域名

BciClientConfiguration config = new BciClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
config.setCnameEnabled(true);
BciClient client = new BciClient(config);
```

配置HTTPS协议访问BCI

BCI支持HTTPS传输协议，您可以通过如下两种方式在BCI Java SDK中使用HTTPS访问BCI服务：

- 在endpoint中指明https:

```
String endpoint = "https://bci.bj.baidubce.com";
String ak = "ak";
String sk = "sk";
BciClientConfiguration config = new BciClientConfiguration();
config.setEndpoint(ENDPOINT);
config.setCredentials(new DefaultBceCredentials(ak, sk));
BciClient client = new BciClient(config);
```

- 通过调用setProtocol方法设置https协议:

```
String endpoint = "bci.bj.baidubce.com"; // endpoint中不包含protocol
String ak = "ak";
String sk = "sk";
BciClientConfiguration config = new BciClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(ENDPOINT);
config.setProtocol(Protocol.HTTPS); // 如果不指明，则使用http
BciClient client = new BciClient(config);
```

> **注意**：如果在endpoint中指明了protocol，则endpoint中的生效，另外单独再调用setProtocol()不起作用。

```
String endpoint = "http://bci.bj.baidubce.com";
String ak = "ak";
String sk = "sk";
BciClientConfiguration config = new BciClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(ENDPOINT);
config.setProtocol(Protocol.HTTPS); // endpoint中已经指明，此为无效操作，对http也是如此
BciClient client = new BciClient(config);
```

配置BciClient

如果用户需要配置BciClient的一些细节的参数，可以在构造BciClient的时候传入BciClientConfiguration对象。
BciClientConfiguration是BCI服务的配置类，可以为客户端配置代理，最大连接数等参数。

使用代理

下面一段代码可以让客户端使用代理访问BCI服务：

```
String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;        // 用户的Secret Access Key
String ENDPOINT = <domain-name>;                            // 用户自己指定的域名

// 创建BciClientConfiguration实例
BciClientConfiguration config = new BciClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

// 创建BCI客户端
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
BciClient client = new BciClient(config);
```

使用上面的代码段，客户端的所有操作都会通过127.0.0.1地址的8080端口做代理执行。

对于有用户验证的代理，可以通过下面的代码段配置用户名和密码：

```
// 创建BciClientConfiguration实例
BciClientConfiguration config = new BciClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

//设置用户名和密码
config.setProxyUsername(<username>);                    //用户名
config.setProxyPassword(<password>);                    //密码
```

设置网络参数

用户可以用BciClientConfiguration对基本网络参数进行设置：

```
BciClientConfiguration config = new BciClientConfiguration();

// 设置HTTP最大连接数为10
config.setMaxConnections(10);

// 设置TCP连接超时为5000毫秒
config.setConnectionTimeoutInMillis(5000);

// 设置Socket传输数据超时的时间为2000毫秒
config.setSocketTimeoutInMillis(2000);
```

设置同步PUT

对PUT操作，默认使用CloseableHttpAsyncClient，可能会出现用户进程执行完未退出的现象。用户可以通过BciClientConfiguration设置所有BCI请求都通过BciClient同步的方式：

```
BciClientConfiguration config = new BciClientConfiguration();

// 设置PUT操作为同步方式，默认异步
config.setEnableHttpAsyncPut(false);
```

参数说明

通过BciClientConfiguration能指定的所有参数如下表所示：

参数	说明
CnameEnabled	使用cname访问BCI资源
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
Credentials	客户端用于签署HTTP请求的BCE凭据
EnableHttpAsyncPut	异步put
Endpoint	访问域名
LocalAddress	本地地址
MaxConnections	允许打开的最大HTTP连接数
Protocol	连接协议类型
ProxyDomain	访问NTLM验证的代理服务器的Windows域名
ProxyHost	代理服务器主机地址
ProxyPassword	代理服务器验证的密码
ProxyPort	代理服务器端口
ProxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
ProxyUsername	代理服务器验证的用户名
ProxyWorkstation	NTLM代理服务器的Windows工作站名称
Region	地域
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
StreamBufferSize	流文件缓冲区大小
UserAgent	用户代理，指HTTP的User-Agent头
RedirectsEnabled	是否开启HTTP重定向。默认开启

实例相关

创建实例

描述 创建一个BCI容器实例。

请求参数 详见API创建实例

示例代码

如下代码可以创建一个BCI实例

```
public CreateInstanceResponse createInstance(BciClient bciClient) {
    CreateInstanceRequest request = new CreateInstanceRequest();
    // 设置实例名称
    request.setName("test-instance-01");
    // 设置可用区
    request.setZoneName("zoneB");
    // 设置安全组
    request.setSecurityGroupIds(new ArrayList<String>()).getSecurityGroupIds().add("g-xxxx");
    // 设置可用区
    request.setSubnetIds(new ArrayList<String>()).getSubnetIds().add("sbn-xxxx");
    // 设置实例重启策略
    request.setRestartPolicy("Always");
    // 设置弹性公网IP
    request.setEipIp("xx.xx.xx.xx");
    // 设置实例标签
    request.setTags(new ArrayList<Tag>()).getTags().add(new Tag("tagkey", "tagvalue"));
    // 设置镜像仓库凭证信息
    request.setImageRegistryCredentials(new ArrayList<ImageRegistryCredential>()).getImageRegistryCredentials().add(new
ImageRegistryCredential(
            "docker.io/wywcoder", "username", "password"
        ));

    // 设置数据卷信息
    Volume volume = new Volume();
    volume.setNfs(xxx);
    volume.setEmptyDir(xxx);
    volume.setConfigFile(xxx);
    request.setVolume(volume)

    Container container = new Container();
    // 设置容器名称
    container.setName("container01");
    // 设置容器镜像
    container.setImage("registry.baidubce.com/bci-zjm-public/ubuntu:18.04");
    // 设置内存大小
    container.setMemory((float) 0.25);
    // 设置cpu大小
    container.setCpu((float) 0.25);
    // 设置容器工作目录
    container.setWorkingDir("/hoeme/work");
    // 设置镜像拉取策略
    container.setImagePullPolicy("Always");
    // 设置容器启动命令
    Collections.addAll(container.setCommands(new ArrayList<String>()).getCommands(), "/bin/sh", "-c", "sleep 36000 &&
exit 0");
    // 设置容器内端口信息
    container.setPorts(new ArrayList<Port>()).getPorts().add(new Port(80, "TCP", "myport"));
    // 设置容器环境变量
    container.setEnvironmentVars(new ArrayList<Environment>()).getEnvironmentVars().add(new Environment("envkey",
"envvalue"));
    // 设置业务容器组
    request.setContainers(new ArrayList<Container>()).getContainers().add(container);

    // 创建BCI实例
    return bciClient.createInstance(request);
}
```

查询实例列表

描述 批量查询BCI实例列表。

请求参数 详见API删除实例

示例代码

如下代码可以查询BCI实例列表

```
public void listInstances(BciClient bciClient) {
  ListInstancesRequest request = new ListInstancesRequest();
  // 设置分页标志
  listInstancesRequest.setMarker(instanceId);
  // 设置分页返回数据大小
  listInstancesRequest.setMaxKeys(maxKey);
  // 执行实例列表操作
  bciClient.listInstances(request);
}
```

查询实例详情

描述 查询BCI实例详情

请求参数 详见API查新实例详情

示例代码

```
public GetInstanceResponse getInstance(BciClient bciClient, String instanceId) {
  // 指定要查询的instanceId
  return bciClient.getInstance(instanceId);
}
```

删除实例

描述 删除一个BCI容器实例，删除后其使用的物理资源可被收回，比如EIP。

请求参数 详见API删除实例

示例代码

如下代码可以删除一个BCI实例

```
public void deleteInstance(BciClient bciClient, String instanceId, Boolean relatedReleaseFlag) {
  // 删除Bci实例
  bciClient.deleteInstance(instanceId, relatedReleaseFlag);  //指定instanceId
}
```

批量删除实例

描述 批量删除BCI容器实例，删除后其使用的物理资源可被收回，比如EIP等资源。

请求参数 详见API批量删除实例

示例代码

如下代码可以批量删除BCI实例

```
public void batchDeleteInstance(BciClient bciClient) {
    DeleteInstanceRequest request = new DeleteInstanceRequest();
    List<String> instanceIds = new ArrayList<String>();
    // 指定要删除的instanceId
    instanceIds.add("p-instance0");
    instanceIds.add("p-instance1");
    request.setInstanceIds(instanceIds);
    // 指定是否释放关联资源
    request.setRelatedReleaseFlag(true);
    bciClient.batchDeleteInstance(request);
}
```

异常处理

BCI异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServerException	服务器异常

用户可以使用try获取某个事件所产生的异常，例如：

```
try {
    bciClient.createInstance(request);
} catch (BceServiceException bce){
    System.out.println(bce.getMessage());
} catch ( BceClientException bce){
    System.out.println(bce.getMessage());
}
```

客户端异常

客户端异常表示客户端尝试向BCI发送请求以及数据传输时遇到的异常。 例如，当发送请求时网络连接不可用时，则会抛出ClientException；当上传文件时发生IO异常时，也会抛出ClientException。

服务端异常

当BCI服务端出现异常时，BCI服务端会返回给用户相应的错误信息，以便定位问题。常见服务端异常可参见BCI错误码

SDK日志

Java SDK发布版本中增加了logback作为slf4j的实现，如果用户没有自己的实现可以直接用，如果工程中有其他的实现，如log4j，则可以替换。

默认日志

如果用户使用默认的logback，则需要配置logback.xml到classpath中。如果没有这个配置文件，日志级别默认为DEBUG。

```
<configuration>
  <property name="LOG_HOME" value="./log/"/>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>${LOG_HOME}/BciUnitTest.%d{yyyy-MM-dd}.log</FileNamePattern>
      <MaxHistory>30</MaxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
    </encoder>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>10MB</MaxFileSize>
    </triggeringPolicy>
  </appender>

  <root level="info">
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="FILE"/>
  </root>
</configuration>
```

自有日志模块

若用户使用自己的日志实现模块，例如项目依赖于Maven，则可以类似下面的配置到pom.xml中来去除logback。

```
<?xml version="1.0" encoding="utf-8"?>

<dependency>
 <groupId>com.baidubce</groupId>
 <artifactId>bce-java-sdk</artifactId>
 <version>${bce.sdk.version}</version>
 <exclusions>
   <exclusion>
     <groupId>ch.qos.logback</groupId>
     <artifactId>logback-classic</artifactId>
   </exclusion>
   <exclusion>
     <groupId>ch.qos.logback</groupId>
     <artifactId>logback-core</artifactId>
   </exclusion>
   <exclusion>
     <groupId>org.slf4j</groupId>
     <artifactId>jcl-over-slf4j</artifactId>
   </exclusion>
 </exclusions>
</dependency>
```

版本变更记录

v0.10.337

- Java SDK开发包[2024-08-27] 版本号 0.10.337

- BCI openApi 查询实例详情和实例列表增加规格信息返回

v0.10.271

- Java SDK开发包[2023-07-12] 版本号 0.10.271
  - 更新创建容器实例参数

v0.10.254

- Java SDK开发包[2023-04-17] 版本号 0.10.254
  - 首次发布
    - 支持BCI创建实例、查询实例列表、查询实例详情、删除实例、批量删除实例接口

BCI服务等级协议SLA

BCI服务等级协议SLA

本服务等级协议（Service Level Agreement，简称“SLA”）规定了百度智能云向客户提供的百度智能云容器实例（简称“BCI服务”）的服务可用性等级指标及赔偿方案。

1.术语和定义

1.1 服务周期：一个服务周期为一个自然月。如客户使用 BCI服务不满一个月则以当月该 BCI服务累计使用时间作为一个服务周期。

1.2 单实例服务周期总分钟数：按照单实例服务月度内的总天数 × 24（小时）× 60（分钟）计算。

1.3 实例不可用：某一分钟内，如果一台设置了出入允许规则的BCI实例以TCP或者UDP协议与任一IP地址的双向（出/入）都无法联通，则视为该分钟内服务无法使用。如果连续超过 5 分钟无法使用，则计入服务不可用分钟数；低于 5 分钟的服务无法使用时间不计入服务不可用分钟数。

1.4 单实例服务不可用分钟数：在一个服务周期内单实例不可用分钟数之和。

1.5 月度服务费用：为客户在一个服务周期（即自然月）中就单实例所支付的服务费用总额，以代金券结算不计入月服务费用。

2. 服务可用性计算规则

2.1 服务可用性计算方式：

服务可用性以单个实例为维度，按照如下方式计算： 服务可用性 =（单实例服务周期总分钟数 - 单实例服务不可用分钟数）/ 单实例服务周期总分钟数 × 100%

2.2 服务可用性标准：

本服务的服务可用性不低于99.95%，如未达到上述可用性标准（属于免责条款情形的除外），您可以根据本协议第3条（赔偿方案）约定获得赔偿。

3. 赔偿方案

对于本服务，如服务可用性低于标准，您有权按照如下条款约定获得赔偿：

3.1 赔偿标准

（1）赔偿以百度智能云发放代金券的形式实现，您应当遵守代金券的使用规则（包括使用期限等，具体以百度智能云官网发布的代金券相关规则为准）。发放的代金券不能折现、不开具发票，仅限您通过您的百度智能云账户购买本服务，不能购买其他的百度智能云服务，您也不可以将代金券进行转让、赠予等。

（2）如果某服务月度没有达到服务可用性标准，赔偿额按照相应未达标服务月度单独计算，赔偿总额不超过相应未达标服务月度内您就本服务支付的相应月度服务费（此处的月度服务费不含用代金券、优惠券、服务费减免等抵扣的费用）。

服务可用性	赔付代金券金额
低于99.95%但是等于或高于99%	月度服务费的10%
低于99%但等于或高于95%	月度服务费的25%
低于95%	月度服务费的100%

3.2 赔偿申请时限

（1）如某服务月度没有达到服务可用性标准，您可以在没有达标的相应服务月度结束后的次月的第五（5）个工作日后，仅通过您相应账户的工单系统提出赔偿申请。您提出赔偿申请后百度智能云会进行相应核实，对于服务月度的服务可用性的计算，若双方出现争议的，双方均同意最终以百度智能云的后台记录为准。

（2）您最晚提出赔偿申请的时间不应超过未达标的相应服务月度结束后六十（60）个自然日。如果您在未达标的相应服务月度结束后的六十（60）日内未提出赔偿申请，或者在未达标的相应服务月度结束后的六十（60）日之后才提出赔偿申请，或者您通过非本协议约定的方式提出申请的，均视为您自动放弃要求赔偿的权利及向百度智能云主张其他权利的权利，百度智能云有权不受理您的赔偿申请，不对您进行任何赔偿或补偿。

4. 免责条款

由以下原因导致的服务不可用时间，则认为该时间不属于服务不可用的计算范畴和百度智能云的赔偿范畴内，百度智能云无须向您承担责任：

(1) 百度智能云预先通知的进行系统维护，如割接、升级、模拟故障演练等计划内停机时间。

(2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的。

(3) 百度智能云以外第三方引起的，例如遭受黑客攻击、或您第三方供应商疏忽导致的不可用。

(4) 您维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的。

(5) 您的疏忽导致的误操作或由您授权的操作所引起的。例如，用户主动重建等操作。

(6) 您未遵循百度智能云产品使用文档或使用建议引起的。

(7) 非百度智能云原因造成的服务不可用或服务不达标的情况。

(8) BCI容器服务产品关联的其他云产品或服务（如：CCE、VPC、BCC、BBC、EIP、BLB、CDS等）故障导致的不可用。

(9) 其他属于相关法律法规、相关协议、相关规则或百度智能云单独发布的相关规则、说明等中所述的百度智能云可以免责、免除赔偿责任等的情况。

(10) 其他不可抗力引起的。

5.其他说明

(1) 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。

(2) 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将以网站公示或发送邮件的方式通知您。如您不同意百度智能云对SLA所做的修改，您有权停止使用Prometheus监控服务，如您继续使用Prometheus监控服务，则视为您接受修改后的SLA。

(3) 本协议项下百度智能云对于用户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该等通知于发送之日视为已送达收件人。因用户未及时获知百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担任何责任。

(4) 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。

(5) 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。

(6) 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。

(7) 关于用户约束条款，详见《用户服务协议》中的“用户的权利与义务”相关条款内容。

(8) 关于服务商免责条款，详见《用户服务协议》中的”免责声明”相关条款内容。