

产品简介

蓝鲸容器管理平台（BCS，Blueking Container Service）是高度可扩展、灵活易用的容器管理服务平台，支持社区原生 **Kubernetes** 编排引擎。

用户无需关注基础设施的安装、运维和管理，只需要调用简单的 API，或者在页面上进行简单的配置，便可对容器进行启动、停止等操作，查看集群、容器及服务的状态，以及使用各种组件服务。

容器服务怎么用

见下图，只需要三步即可运行服务：

1. 创建或导入集群
2. 上传helm chart包到chart仓库
3. 使用chart包创建应用helm release

术语解释

了解 BCS，会涉及到以下基本概念：

- **容器编排引擎**：用于容器化应用的自动化部署、扩展和管理的工具或平台，目前行业主流的引擎 **K8S**。
- **集群**：容器运行所需要的物理机或虚拟机资源的集合。
- **节点**：一台已经注册到集群中的服务器，为集群提供计算资源。
- **应用**：由一组容器及服务构成集合，这个集合可以代表一个业务，或者业务的某个大区，在 K8S 中应用通常由工作负载（Workload）、服务（Service、Ingress）、配置（ConfigMap、Secret）构成。
- **Helm Release**：Release 是运行在 Kubernetes 集群中的 chart 的实例。一个 chart 通常可以在同一个集群中安装多次。每一次安装都会创建一个新的 release。以 MySQL chart 为例，如果你想在你的集群中运行两个数据库，你可以安装该chart两次。每一个数据库都会拥有它自己的 release 和 release name
- **网络**：主要包含 **服务** 和 **负载均衡器** 的定义和管理，服务是由多个相同配置的容器和访问这些容器的规则组成的微服务，负载均衡器定义了访问规则的具体实现。
- **仓库**：用户存放 Docker 镜像、Helm Charts。

产品特性

BCS 的核心功能包含集群管理、资源模板管理、应用管理、镜像管理以及网络管理。

产品功能	概述	功能简介
集群管理	通过蓝鲸容器服务可以简单高效地管理容器集群	<p>K8S原生集群创建</p> <ul style="list-style-type: none">• 支持自定义设置Master和Node节点，一键自动安装集群组件• 用户独占集群，保证安全隔离性 <p>集群导入</p> <ul style="list-style-type: none">• 支持通过集群kubeconfig文件导入外部集群，使用BCS管理已存在K8S集群• 支持通过云凭证导入云上K8S集群，目前支持腾讯云TKE集群导入，支持Worker节点自动扩缩容 <p>集群管理</p> <ul style="list-style-type: none">• 支持节点添加、删除，集群删除• 支持节点标签、污点与资源调度等节点管理功能• 支持集群和节点级别的监控告警及主要数据的视图展示

模板管理	模板管理为用户提供了在集群中部署资源的管理方案。支持用户设置容器编排的Helm Chart。用户可以将同一套Helm Chart，实例化到不同的命名空间，通过不同的values，完成差异化的资源编排	管理容器资源模板 <ul style="list-style-type: none"> 支持模板集或Helm Chart的多版本管理 支持通过命名空间管理不同的环境
应用管理	应用管理为用户提供了对模板集实例化后得到的容器集合的管理方案，通过应用管理，可以看到容器各个维度的信息和状态，并且可以针对单个维度进行操作，重启容器，重新调度容器等等	容器视图 <ul style="list-style-type: none"> 通过应用视图或者命名空间视图管理容器 查看应用、POD、容器等的在线状态 容器操作 <ul style="list-style-type: none"> 启停容器，重新调度容器 对应用做更新，例如扩缩容、滚动升级等
镜像管理	镜像仓库包含公共镜像库、项目私有镜像库。项目私有镜像库只有项目中有指定权限的人才能访问	<ul style="list-style-type: none"> 公共镜像，包含了一些实用程度比较高，且开源共有的镜像资源。公共镜像对所有用户可见 项目镜像，是项目成员主动添加的镜像，或者是通过CI流程归档的私有镜像。项目镜像只有项目中指定的权限所有者才能访问。
网络管理	网络管理提供了用户管理服务和负载均衡器的方案。用户通过网络管理可以查看线上服务的状态和负载均衡器的状态	服务管理 <ul style="list-style-type: none"> 查看服务的列表，以及每个服务的详细信息 对服务进行操作，例如更新服务或者停止服务 负载均衡器管理 <ul style="list-style-type: none"> 查看线上负载均衡器列表，及每个负载均衡器的详细信息 启动、删除或者更新负载均衡器 使用bcs ingress controller可实现多云负载均衡器管理与功能增强
GameWorkload	GameDeployment 是针对游戏场景定制的面向无状态服务的增强版 Deployment，GameStatefulSet 是针对游戏场景定制的面向有状态服务的增强版 StatefulSet	GameDeployment <ul style="list-style-type: none"> 支持 Operator 高可用部署 支持滚动更新 / 原地更新 优雅删除和更新 支持设置 partition 灰度发布 支持分步骤自动化灰度发布 支持 HPA 指定 pod 删除 支持 pod 注入唯一序号 支持防误删功能 支持 readinessgate 可选功能 GameStatefulSet <ul style="list-style-type: none"> 支持 Operator 高可用部署 支持 Node 失联时，Pod 的自动漂移 支持优雅删除和更新 支持滚动更新/原地更新 支持分步骤自动化灰度发布 支持并行更新 支持 maxSurge / maxUnavailable 字段 支持 HPA 支持防误删功能
容器监控与日志	BCS打通蓝鲸容器监控与容器日志平台，为BCS集群提供监控告警与日志采集查询能力	蓝鲸容器监控 <ul style="list-style-type: none"> 分布式采集k8s集群的监控数据 支持控制平面，servicemonitor，podmonitor指标数据采集 完整兼容Prometheus协议的采集方式 支持集群事件采集 蓝鲸容器日志 <ul style="list-style-type: none"> 分布式采集k8s集群的日志 支持标准输出日志，容器内程序日志

API密钥 BCS打通蓝鲸容器监控与容器日志平台，为BCS集群提供监控告警与日志采集查询能力	蓝鲸容器监控 <ul style="list-style-type: none"> ● 分布式采集k8s集群的监控数据 ● 支持控制平面，servicemonitor，podmonitor指标数据采集 ● 完整兼容Prometheus协议的采集方式 ● 支持集群事件采集 蓝鲸容器日志 <ul style="list-style-type: none"> ● 分布式采集k8s集群的日志 ● 支持标准输出日志，容器内程序日志
---	--

产品优势

支持原生 Kubernetes 容器编排方案

Kubernetes 是基于 Google borg 系统开源的项目，集成了 [资源调度](#) 和 [应用编排](#) 的能力，面向分布式应用、微服务和大规模集群管理。

基于 Kubernetes

- 基于原生 Kubernetes 实现，秉承社区开源、开放的心态
- 支持社区容器、网络、存储实施方案

基于 Docker 的服务生态

- 服务发现

基于 Kubernetes 的集群，都自带了服务发现的能力。服务发现有两种模式，一是通过服务的域名访问服务，在域名上动态绑定当前服务的后端；另一种是通过服务代理容器，流量全部导向服务代理容器，由代理容器将流量转发到服务的后端。

- 负载均衡

负载均衡器是一组特殊的容器，用来帮一个服务或者多个服务实现后端流量或者处理能力的均衡。用户可以设定负载均衡的算法以达到不同的负载均衡效果。

- 分布式配置中心

业务程序在运行过程中往往需要使用不同的配置启动，在活动期间，也可能需要通过配置调整策略。蓝鲸容器服务提供了分布式配置中心，用户可以将配置存放在配置中心，业务容器可以通过指定的协议方式获取到对应的配置。

- CNI 格式的 Overlay 和 Underlay 网络支持

容器的网络方案不仅支持 Overlay，也支持 Underlay 的方案。在 Underlay 方案下，每个容器拥有一个真实的内网 IP，并且在容器销毁时自动回收该 IP，用户也可以设定容器重启、迁移时使用固定的一组 IP。

认证

蓝鲸智云容器管理平台于 2019 年 7 月 30 日通过了中国云计算开源产业联盟组织的可信云容器解决方案评估认证。

蓝鲸智云容器管理平台在基本能力要求、应用场景技术指标、安全性等解决方案质量方面，以及产品周期、运维服务、权益保障等服务指标的完备性和规范性方面均达到可信云容器解决方案的评估标准。应用场景满足以下四个：

- 开发测试场景
- 持续集成、持续交付
- 运维自动化
- 微服务

产品开源

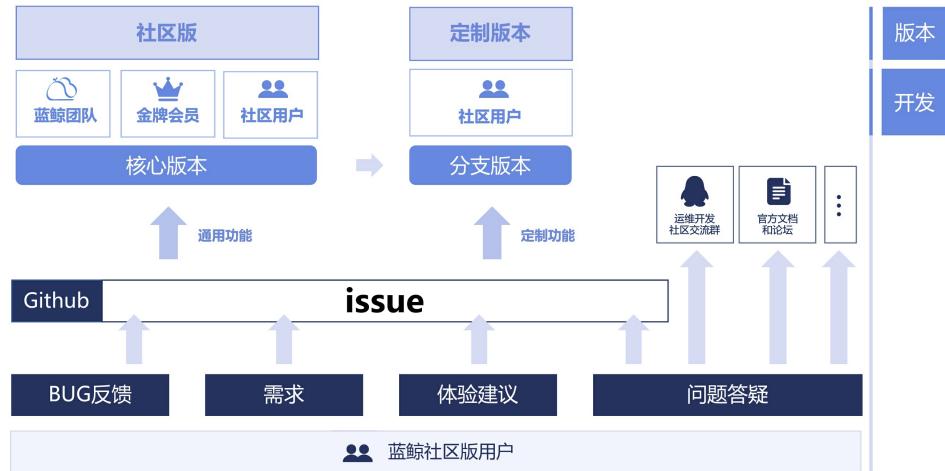
BCS 团队对容器管理平台进行开源，希望将我们的技术和沉淀反馈给社区，期望能帮助更多的人解决问题；同时也邀请容器技术爱好者一起参与建设，让产品变更更加强大和易用，构建生态活跃的技术社区。

开源地址：<https://github.com/Tencent/bk-bcs>、<https://github.com/Tencent/bk-bcs-saas>



开源协作方式





开源协议

蓝鲸容器管理平台采用的是 MIT 开源协议。MIT 是和 BSD 一样宽范的许可协议，BCS 团队只想保留版权，而无任何其他的限制。也就是说，你必须在你的发行版里包含原许可协议的声明，无论你是以二进制发布的还是以源代码发布的。

欢迎交流



产品架构图

BCS 是蓝鲸基于社区原生 K8S 的容器部署管理解决方案。



BCS（容器管理平台）架构图

BCS 由 **BCS SaaS** 和 **BCS 后台** 组成，以下为对应的架构图。

BCS SaaS 架构图

BCS SaaS 功能结构图

BCS SaaS 作为 BCS 的上层产品，包含已开源的项目管理系统（bcs-projmgr）、容器服务产品层主体功能模块（bcs-app）、底层的配置中心模块（bcs-cc）以及未开源的监控中心，同时它依赖蓝鲸体系下的其他产品服务（如 PaaS、CMDB 等）。



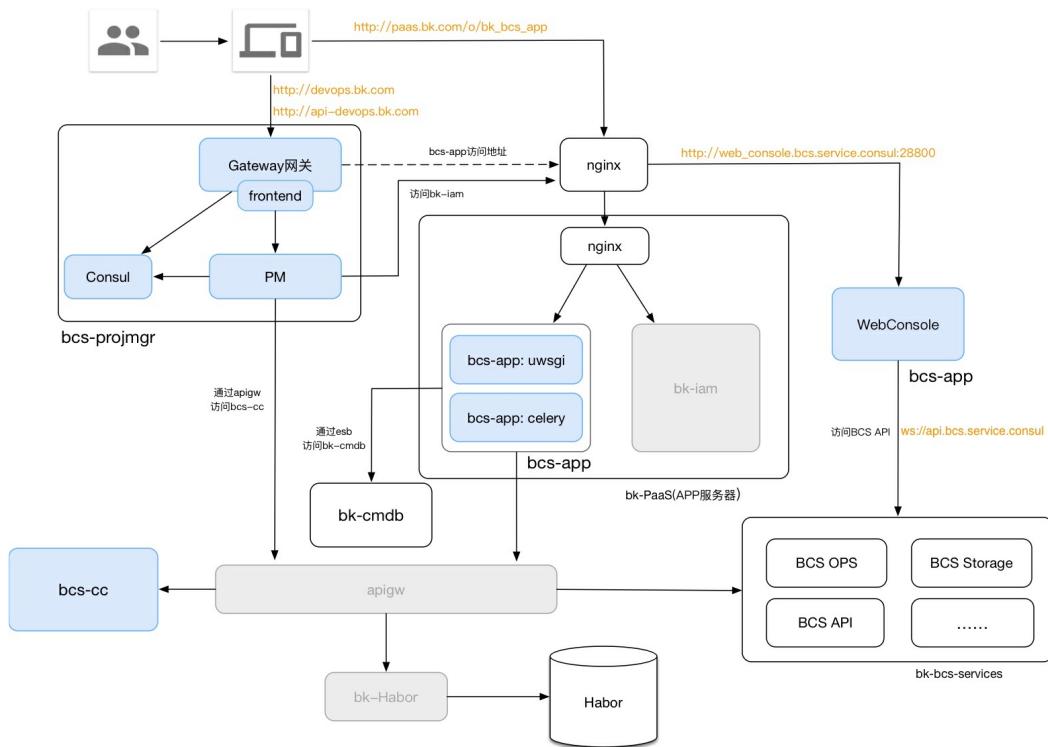
SaaS 依赖的服务介绍： - **bk-PaaS**：为 BCS SaaS 提供了 4 大服务(统一登录、开发者中心、ESB 和应用引擎)，其中 bcs-app 由应用引擎托管

- **bk-bcs-services**: BCS 底层服务。作为后台服务, bk-bcs-services 给 bcs-app 提供了集群搭建, 应用编排等丰富的底层接口, 更多详见下 [BCS 后台架构图](#)
- **bk-cmdb**: 蓝鲸配置平台。bcs-app 的集群管理功能涉及的业务和主机信息来源于配置平台
- **bk-iam**: 蓝鲸权限中心, BCS SaaS 基于 bk-iam, 实现了用户与平台资源之间的权限控制
- **bk-Habor**: 蓝鲸容器管理平台镜像仓库服务。bcs-app 使用 bk-Habor 提供的 API, 实现了业务镜像的查询与配置功能

BCS SaaS 部署拓扑图

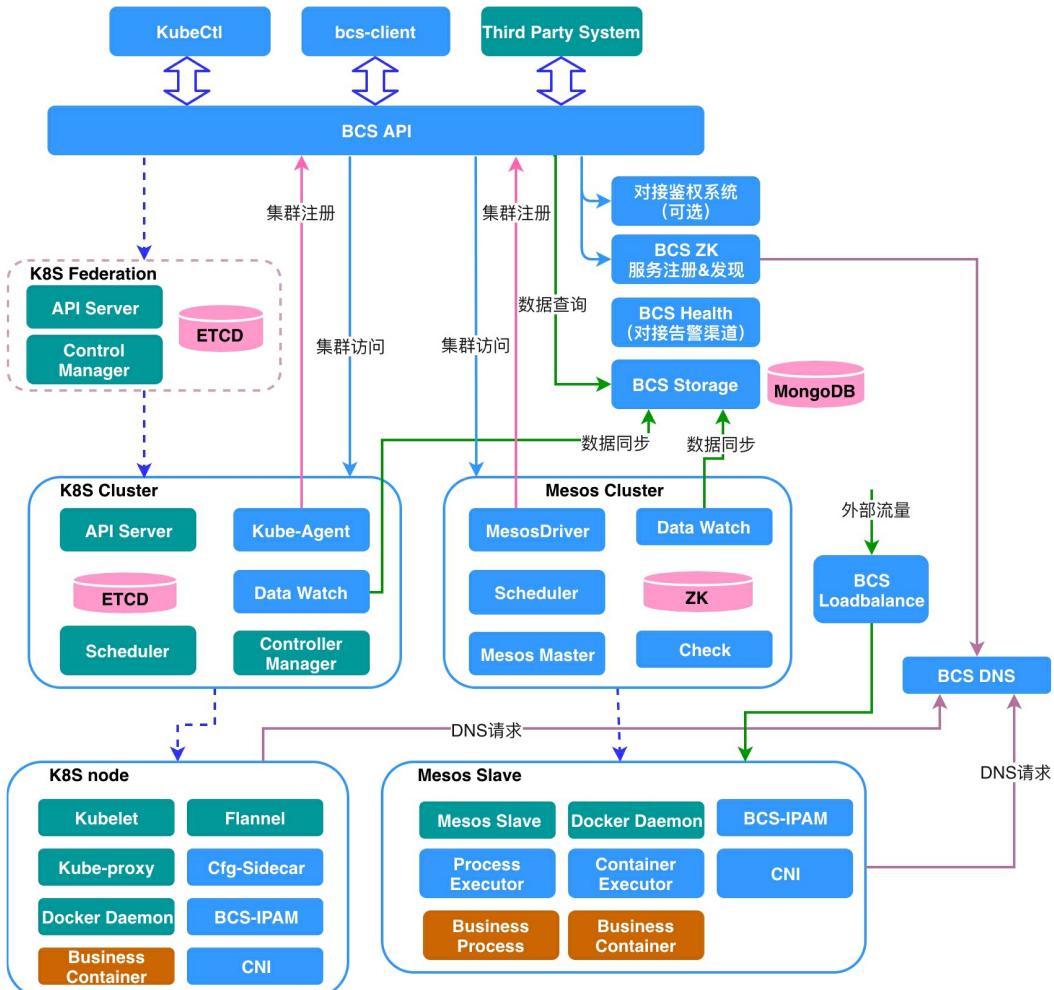
SaaS 包含 bcs-projmgr, bcs-app, bcs-cc 三个模块。

SaaS 依赖的后端服务 bk-bcs-services 也已开源, bk-iam 等灰色标注的系统暂未开源, 需要依托蓝鲸独立部署版本进行搭建。



BCS 后台架构图

下图为 BCS 的整体架构图: BCS Client 或者业务 SaaS 服务通过 API 接入, API 根据访问的集群将请求路由到 BCS 下的 K8S 集群。



Kubernetes 容器编排的说明： - BCS 支持原生 K8S 的使用方式 - K8S 集群运行的 Agent (bcs-k8s-agent) 向 BCS API 服务进行集群注册 - K8S 集群运行的 Data Watch 负责将该集群的数据同步到 BCS Storage

容器管理套餐快速入门



此外，场景案例中的 [如何构建 Nginx 集群](#) 也可以实现快速上手 BCS。

登录蓝鲸容器服务控制台

登录蓝鲸容器服务控制台。

创建项目（也可选择已有项目）

项目名称	项目英文名	项目说明	创建者	操作
test4proj 2020-12-21	test4proj	test4proj		编辑项目
ceshi1 2020-12-21	ceshi1	ceshi1	admin	编辑项目
test2312 2020-12-15	test2312		admin	编辑项目
lasttest 2020-12-08	lasttest		admin	编辑项目

- 创建新项目：进入项目管理页面，点击【新建项目】按钮，完成项目创建操作
- 获取已有项目权限：进入蓝鲸权限中心，【申请加入】已有项目来获取项目使用权限

关键项说明：通过项目进入容器服务后，还需要关联“蓝鲸配置平台（CMDB）”上的某个业务，目的是从该业务机器资源池中获取创建集群的主机列表。

创建集群

在容器服务左侧导航中点击【集群】进入集群管理页面，点击【创建集群】按钮。

关键项说明：- 集群 Master 节点为奇数个，最少 1 个，最多 7 个 - 创建集群，系统将对主机做以下初始化操作：- 机器前置检查 - 安装蓝鲸容器服务初始化包 - 安装蓝鲸容器服务基础组件 - Master 要求：- 机器配置：至少 CPU/内存为 4 核/8G - 系统版本：CentOS 7 及以上系统（内核版本 3.10.0-693 及以上），推荐 CentOS 7.4

添加集群节点

集群创建成功后，您可以进入集群节点列表，为集群增加节点。

关键项说明：- Node 要求：- 系统版本：CentOS 7 及以上系统（内核版本 3.10.0-693 及以上），推荐 CentOS 7.4 - NAT 模块：确认 NAT 模块已安装

创建命名空间

在容器服务左侧导航中点击【命名空间】，点击【新建】按钮，创建指定集群的命名空间信息（创建服务实例将以集群命名空间维度创建）。

注意：创建命名空间后，名称不允许修改。

创建服务实例

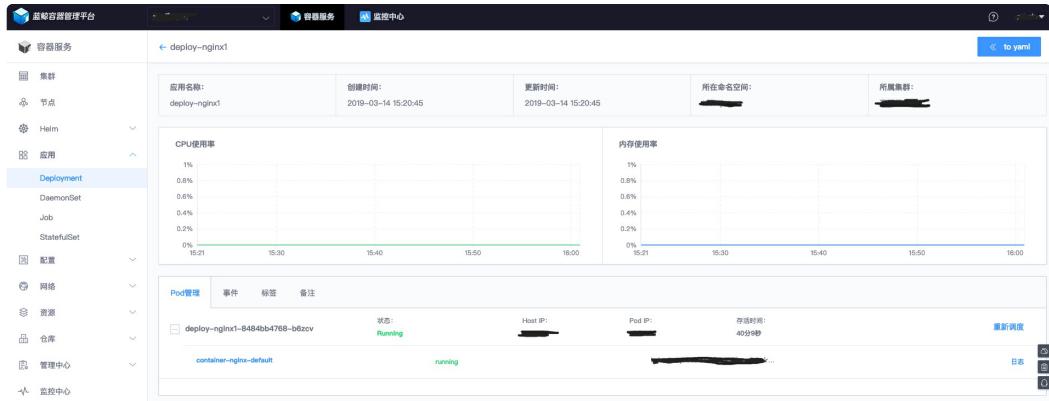
新建项目时，系统将初始“示例模板集”到当前项目的模板集库中，您可以直接使用该模板集体验操作。

示例模板集：- 是蓝鲸提供的《吃豆小游戏》模板配置，您可以查看该模板配置详情，对模板的使用有一个初步概念 - 模板集中使用了镜像仓库提供的公共镜像，您可以在容器服务左侧导航中点击“仓库”查看，并推送您的项目镜像到项目私有镜像仓库

创建服务应用实例：- 进入“示例模板集”，点击【实例化】按钮进入实例化页面 - 选择模板集，选择集群命名空间 - 点击【创建】按钮

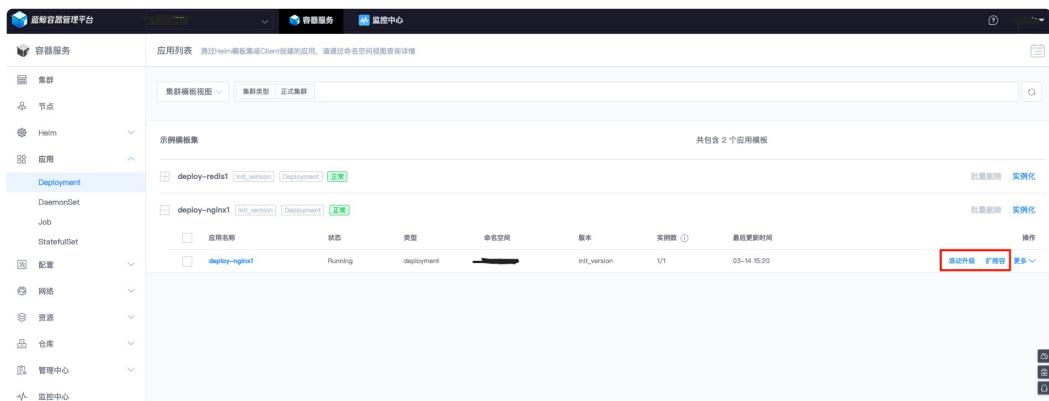
确认完成

《吃豆小游戏》已经部署完成，您可以在容器服务左侧导航中点击【应用】，查看小游戏服务应用实例。



接下来，您可以体验吃豆小游戏：

- 在 `deploy-nginx` 应用详情页面查看 `Host IP`，也就是接入层的 IP
- 将链接 `http://HOST_IP/rumpetroll/?openid=is__superuser&token=tPp5GwAmMPIrzXhyyA8X` 中的 `HOST_IP` 替换为接入层 IP(如下图红框部分)，访问即可体验



您可以尝试： - 在线滚动升级小游戏 - 在线扩缩容小游戏服务实例 - 在线删除或重建小游戏服务实例

小游戏使用说明

部署完成后，用户可以登入小游戏试玩使用。

注意：下面 token 默认为 `tPp5GwAmMPIrzXhyyA8X`。

PC 登入地址



`http://{{domain}}/rumpetroll/?openid=is__superuser&token={{token}}`



PC 登入可以显示倒计时页面，管理员或者投放到大屏使用。

玩家登入地址



`http://{{domain}}/rumpetroll/`



普通玩家使用上面地址登入游戏。

游戏开启和关闭

默认情况下，游戏是关闭的，可以调用 API 开启，关闭游戏。

```
```bash
```

## 开启游戏

```
curl -X POST -H 'Content-Type: application/x-www-form-urlencoded' -d 'func_code=is_start&enabled=1' 'http://{domain}/rumpetroll/api/func_controller/?token={token}'
```

## 关闭游戏

```
curl -X POST -H 'Content-Type: application/x-www-form-urlencoded' -d 'func_code=is_start&enabled=0' 'http://{domain}/rumpetroll/api/func_controller/?token={token}'
```

## 获取开关状态

```
curl -X GET 'http://{domain}/rumpetroll/api/func_controller/?token={token}&func_code=is_start' ```
```

### 发送豆子

可以调用 API 发送豆子，用户角色吃掉豆子后，体型有变大效果。

```
```bash
```

发送豆子，num 是发送豆子数量

```
curl -X GET 'http://{domain}/rumpetroll/api/gold/?token={token}&num={num}' ```
```

注意：发送完成后，PC 端会自动进入倒计时，默认 3 分钟。

游戏统计地址

- 在线统计：<http://{domain}/rumpetroll/api/stat/?token={token}&meter=online>
- 吃豆排名统计：<http://{domain}/rumpetroll/api/stat/?token={token}&meter=online>
- 豆子剩余数量：<http://{domain}/rumpetroll/api/stat/?token={token}&meter=golds>

重置数据

使用 `web-console` 登入到 redis 所在 pod 中，清空 redis 数据即可。

```
bash redis-cli flushall
```

快速构建 Nginx 集群

情景

传统的 Nginx 集群要先部署多个 Nginx 节点，然后通过 `upstream` 统一一个入口提供给用户访问。

该过程操作繁琐，接下来看 BCS（容器管理平台）如何通过 容器调度 快速构建 Nginx 集群。

前提条件

- K8S 基本概念，包含 Deployment、Services。
- 完成 BCS 部署
- 准备 2 台云主机：4 核 8 G，不低于 CentOS 7，K8s Master 和 Node 各 1 台

- 完成上述 2 台云主机的 Agent 安装，并分配至 CMDB 业务下

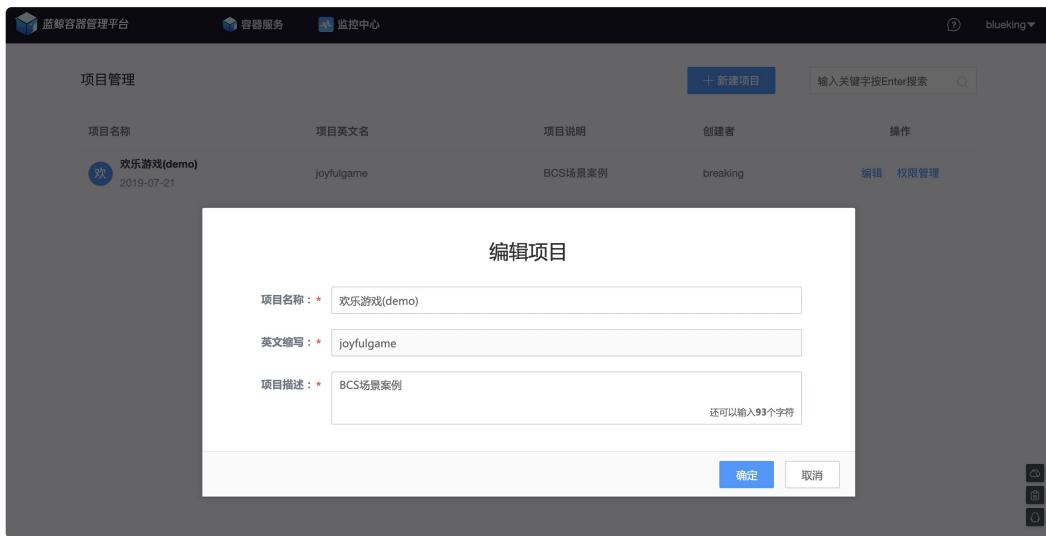
操作步骤

- 新建集群
- BCS 快速构建 Nginx 集群

新建集群

启用容器服务

在 BCS 首页，点击 新建项目，如 欢乐游戏(demo)。

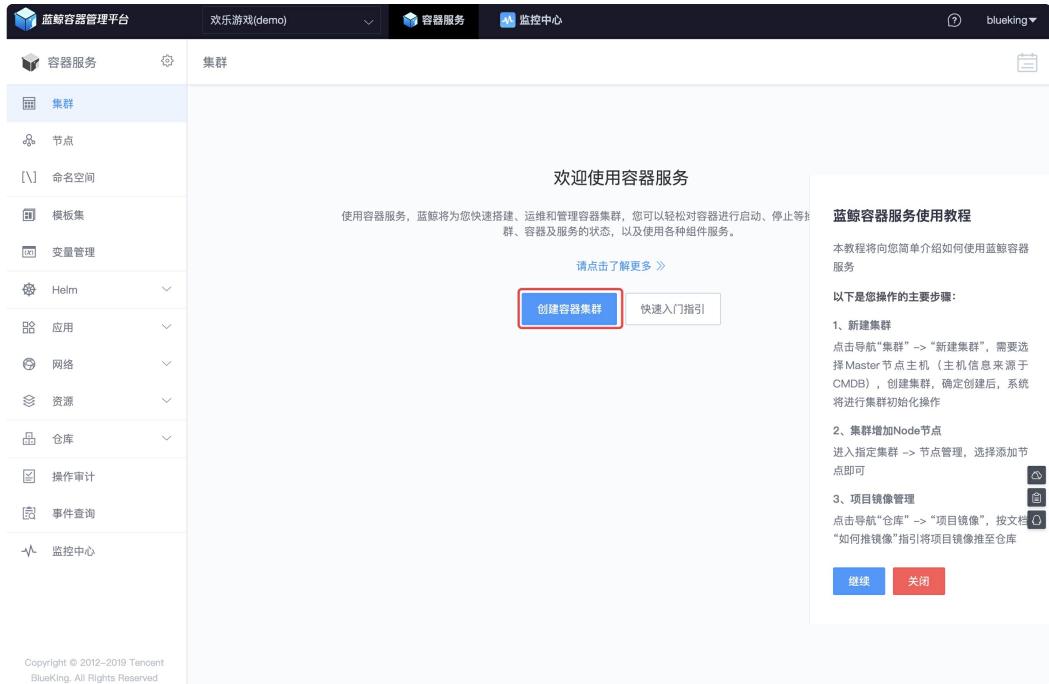


然后选择容器编排类型为 Kubernetes，关联 前提条件 中提到的 CMDB 业务，点击 启用容器服务。

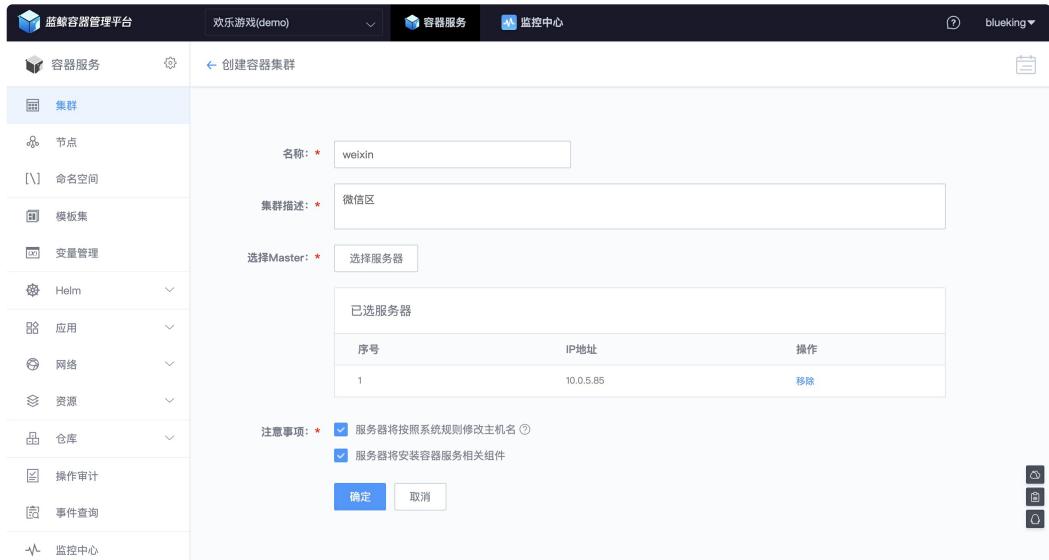


新建集群

启用容器服务后，进入容器服务欢迎页，点击 创建容器集群。



按提示填写集群的基本信息。



容器服务的集群划分和 [传统单体应用在 CMDB 中的集群划分](#) 很类似，可以按照 [地域（如华北区）](#) 或者 [完全独立的应用集合（微信区）](#) 来划分。

选择 1 台云主机作为 Master。

选择服务器 (关联业务: 欢乐游戏(demo)) 请选择奇数个服务器 已选择1个节点

主机名称	内网IP	Agent状态
--	10.0.5.85	正常
--	10.0.5.94	正常

确定 取消

点击 **确定** 后，集群开始初始化。

集群

weixin
BCS-K8S-40015

正在初始化中, 请稍等... 查看日志

点击新建集群

2019-08-03 20:53:50 blueking 初始化集群

- 前置检查 - SUCCESS
- 初始化 - SUCCESS
- 下发部署脚本 - SUCCESS
- 生成8s证书 - SUCCESS
- 下发安装包 - SUCCESS
- 生成etcd证书 - SUCCESS
- 分发etcd证书 - SUCCESS
- 分发8s证书 - SUCCESS
- 部署docker - SUCCESS
- 部署etcd - RUNNING

正在加载中...

点击 **节点管理**

容器服务 集群

weixin
BCS-K8S-40015

总览 节点管理 集群信息 删除

CPU使用率 内存使用率 磁盘使用率 0%

点击新建集群

点击 **添加节点**，按提示节点添加。

容器服务 集群 weixin (BCS-K8S-40015)

总览 节点管理 集群信息

+ 添加节点 批量操作

主机名/IP	状态	容器数量	CPU	内存	磁盘IO	操作
10.0.5.94	初始化中	1	29.13%	14.76%	8.53%	查看日志

共计1条 每页 5 条

至此，新建集群完毕。可以看到集群的基础信息。

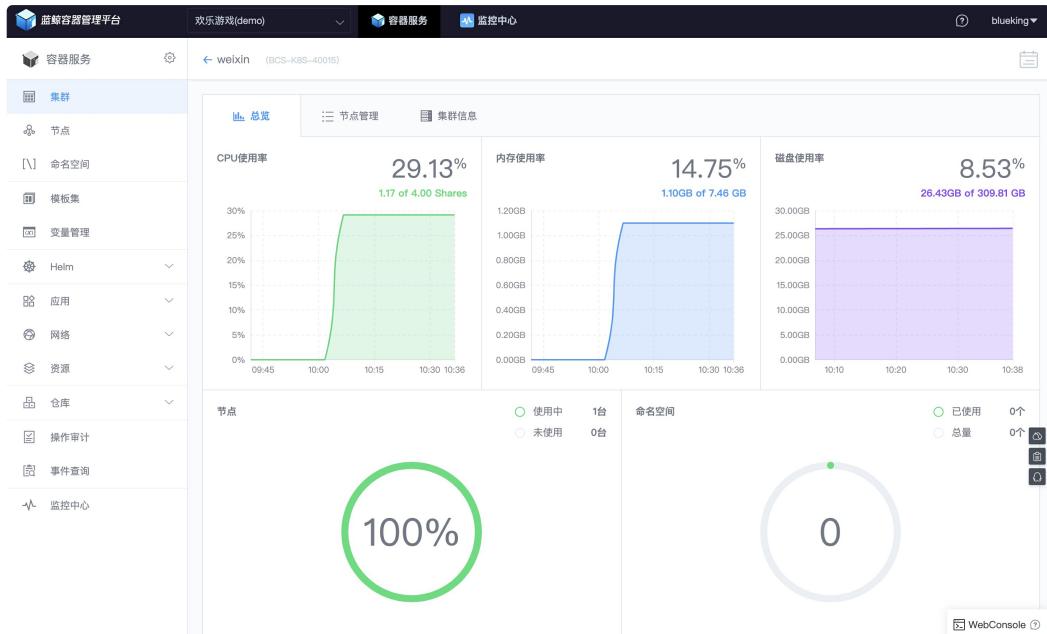
容器服务 集群

weixin
BCS-K8S-40015

CPU使用率 29.13%
内存使用率 14.76%
磁盘使用率 8.53%

点击新建集群

另外，在集群的设置(:)下拉菜单中，可以看到集群主要性能指标。



BCS 快速构建 Nginx 集群

新建命名空间

新建命名空间 `dev`。

The screenshot shows the 'Namespace' section of the BlueKing Container Management Platform. It lists existing namespaces: preprod, dev (which is highlighted with a red border), and prod. A 'New' button is visible at the top left of the list.

名称	所属集群	变量	操作
preprod	weixin	--	Set Variable Value
dev	weixin	--	Set Variable Value
prod	weixin	--	Set Variable Value

新建模板集

模板集，可以类比为 K8S 中 `Helm` 的 `Charts`，在 K8S 编排中，是 K8S 对象的集合：`Deployment`（无状态）、`StatefulSet`（有状态）、`DaemonSet`（守护进程集）、`Job`（定时任务）、`Configmap`（配置项）、`Secret`（保密字典），具体参见 [模板集使用介绍](#)。

打开菜单 [模板集]，新建模板集 `web-nginx`。

按提示，填写 Deployment

填写 Service

实例化

检查部署效果

在菜单 网络 -> Services 中, 找到刚实例化的 Service web-nginx。

1. 集群内可访问: 10.254.11.4:8088
web-nginx:8088

基础信息

选择器:	app==web-nginx	类型:	NodePort	Service IP:	10.254.11.4	Service Domain:	---
------	----------------	-----	----------	-------------	-------------	-----------------	-----

端口映射

端口名称	端口	协议	目标端口	NodePort
http	8088	TCP	container-port	30008

Endpoints

名称	Pod IP	Node IP
web-nginx-678bb9c4fb-nxwf4	172.32.1.17	10.0.5.94
web-nginx-678bb9c4fb-m8cf4	172.32.1.18	10.0.5.94

2. 对外提供访问: Node IP + Node Port
10.0.5.94

标签

io.tencent.bcs.clusterid	BCS-K8S-40015	io.tencent.bcs.kind	Kubernetes	io.tencent.bcs.namespace	dev
app	web-nginx	io.tencent.paas.version	v1	io.tencent.bcs.app.applid	6
io.tencent.paas.projectid	794bd16b4a194ac2952b497e79a0b967	io.tencent.bcs.cluster	BCS-K8S-40015	io.tencent.paas.templateid	68
io.tencent.paas.source_type	template	io.tencent.bkdata.basefield.dataid	65666	io.tencent.paas.instanceid	42
io.tencent.paas.containerid	42	io.tencent.bkdata.container.stdlog.dataid	0	io.tencent.paas.versionid	117

在菜单 [应用] -> [Deployment] 中可以找到 web-nginx。

应用列表 通过Helm模板集或Client创建的应用, 请通过命名空间视图查询详情

命名空间	应用名	状态	操作
game	weixin	共包含 2 个应用, 无异常	批量删除
dev	weixin	共包含 1 个应用, 无异常	批量删除
	web-nginx	Running	操作

以及其运行指标:

应用名称: web-nginx
创建时间: 2019-08-08 16:58:19
更新时间: 2019-08-08 16:58:19
所在命名空间: dev
所属集群: weixin

CPU使用率

内存

Pod	事件	标签	备注	状态	Host IP:	Pod IP:	存活时间:
web-nginx-678bb9c4fb-nxwf4				Running	10.0.5.94	172.32.1.17	11分40秒
web-nginx-678bb9c4fb-m8cf4				Running	10.0.5.94	172.32.1.18	11分40秒

通过访问 **Node+NodePort**, 可以查看刚刚部署 Nginx 集群的版本号。

```
bash [root@ip-10-0-5-94-n-bcs-k8s-40015 ~]# curl 10.0.5.94:30008 -I HTTP/1.1 200 OK Server: nginx/1.12.2 Date: Thu, 08 Aug 2019 09:11:42 GMT
```

通过访问 `Service IP + Port`，也可以查看刚部署 Nginx 的版本号。

```
bash [root@ip-10-0-5-94-n-bcs-k8s-40015 ~]# curl 10.254.11.4:8088 -I HTTP/1.1 200 OK Server: nginx/1.12.2 Date: Thu, 08 Aug 2019 09:12:33 GMT Content-Type: text/html Content-Length: 612 Last-Modified: Tue, 11 Jul 2017 13:29:18 GMT Connection: keep-alive ETag: "5964d2ae-264" Accept-Ranges: bytes
```

应用的滚动升级

情景

传统的应用更新方式是停服更新，用户在更新期间无法使用服务。

接下来，将以 Nginx 从 `1.12.2` 升级 `1.17.0` 为例，看 BCS 中的滚动更新能力是如何实现不停机更新，用户无感知。

前提条件

- K8S 基本概念，包含 Deployment、Services。
- [完成 BCS 部署]

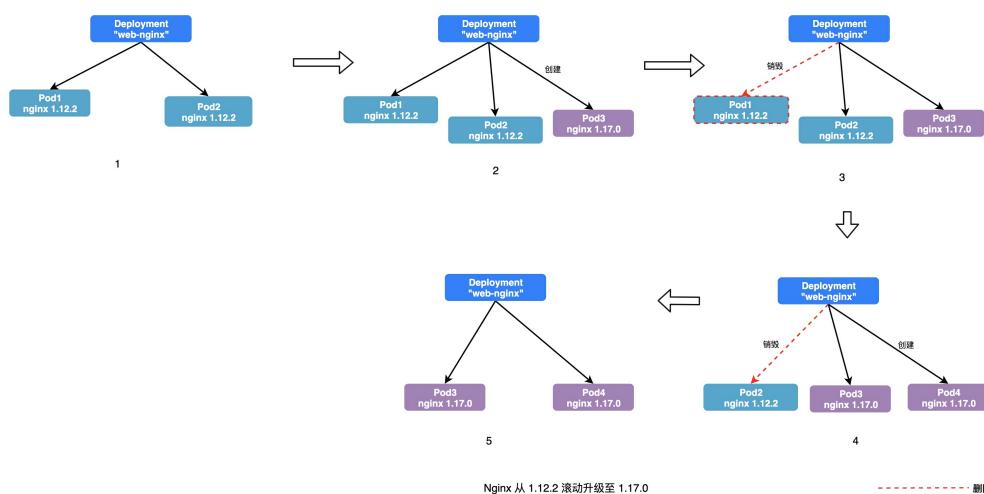
操作步骤

1. 滚动更新逻辑介绍
2. BCS 滚动更新操作指引

滚动更新逻辑介绍

滚动更新的逻辑如下图，创建一个新版本的实例（POD），销毁一个旧版本实例（POD），如此滚动，直至线上都是新版本，旧版本已全部销毁。

滚动更新对用户无感知。



BCS 滚动更新操作指引

推送 Nginx:1.17.0 至镜像仓库

参照 [Harbor 仓库使用指南](#)，将镜像 Nginx:1.17.0 推送至 BCS 公共镜像仓库。

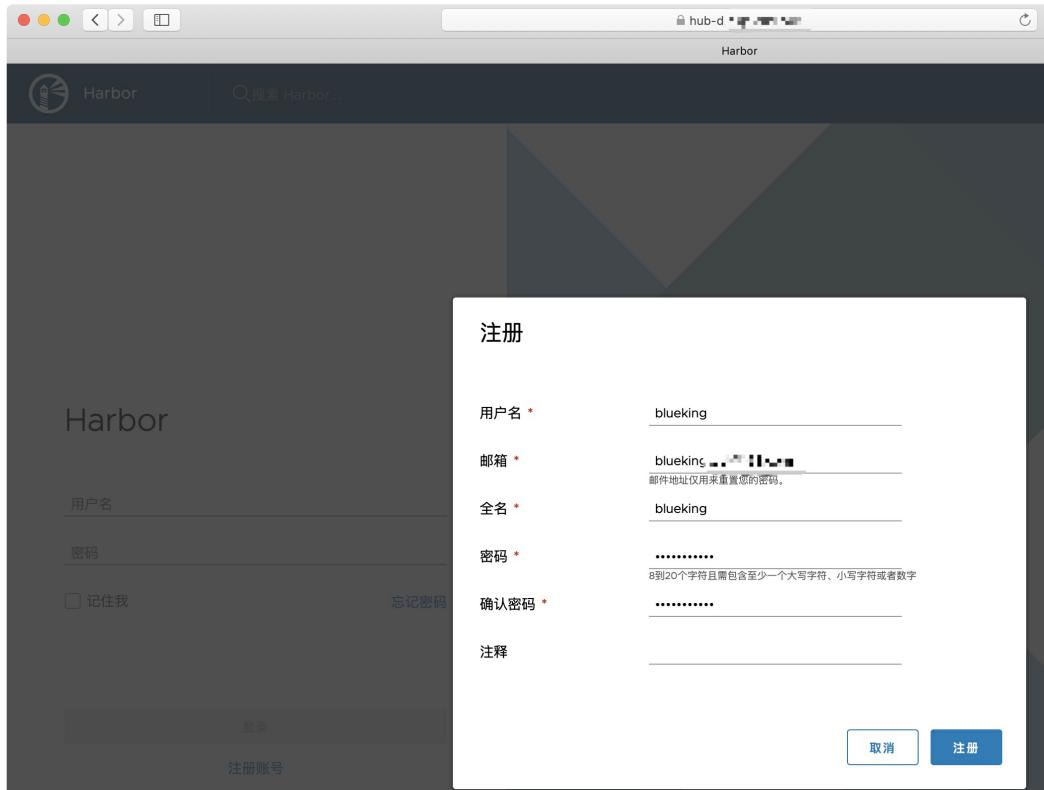
注册镜像仓库账号

在 [部署 BCS](#) 的中控机上获取镜像仓库的访问地址。

```bash

```
source /data/install/utils.fc && echo
${HARBOR_SERVER_FQDN}:${HARBOR_SERVER_HTTPS_PORT}
hub-d.o.**.com:443````
```

登录仓库地址，注册镜像仓库账号。



注册完，登录后可以访问公共仓库。

### 推送 Nginx:1.17.0 至镜像仓库

使用 `docker pull` 从 [hub.docker.com](#) 拉取镜像 `nginx:1.17.0`。

```
```bash
```

docker pull nginx:1.17.0

```
1.17.0: Pulling from library/nginx fc7181108d40: Pull complete c4277fc40ec2: Pull complete 780053e98559: Pull complete Digest:  
sha256:bd6b7f1f77fe7bd2a32e59235dff6ecf131e3b6b5b96061c652f30685f3a Status: Downloaded newer image for nginx:1.17.0 ````
```

规范镜像 `tag` 为仓库要求的格式。

```
```bash
```

## docker tag nginx:1.17.0 hub-d.\*\*.com/public/nginx:1.17.0

### docker images

```
REPOSITORY TAG IMAGE ID CREATED SIZE nginx 1.17.0 719cd2e3ed04 8 weeks ago 109MB hub-d.**.com/public/nginx 1.17.0
719cd2e3ed04 8 weeks ago 109MB ````
```

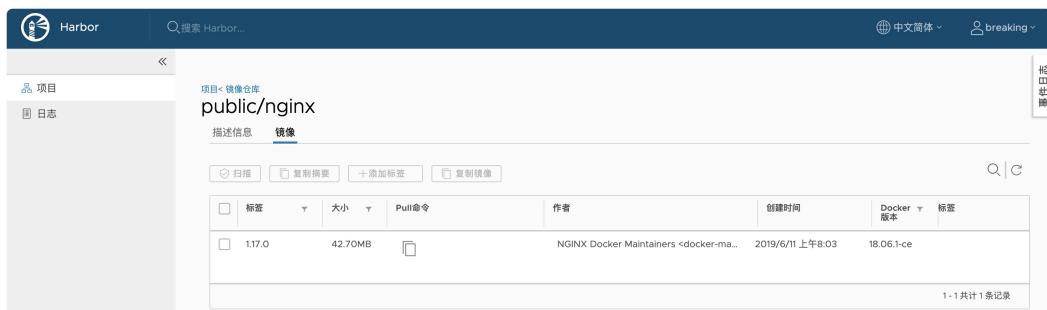
推动镜像至 BCS 镜像仓库。

```
```bash
```

docker push hub-d.**.com/public/nginx:1.17.0

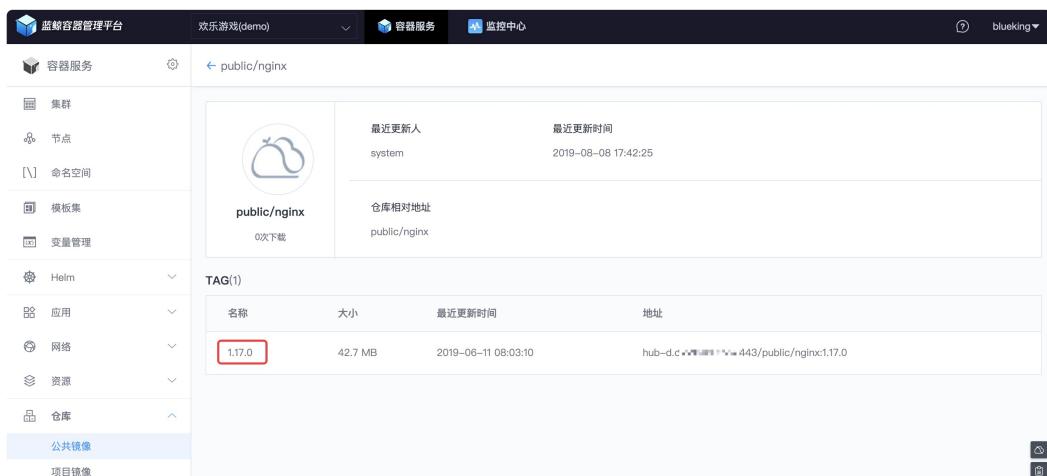
```
The push refers to repository [hub-d.**.com/public/nginx] d7acf794921f: Pushed d9569ca04881: Pushed cf5b3c6798f7: Pushed 1.17.0:  
digest: sha256:079aa93463d2566b7a81cbdf856afc6d4d2a6f9100ca3bcbeef24ade92c9a7fe size: 948 ````
```

在镜像仓库中，可以找到刚推送的 `Nginx:1.17.0` 镜像。



标签	大小	Pull命令	作者	创建时间	Docker版本	标签
1.17.0	42.70MB	□	NGINX Docker Maintainers <docker-ma...	2019/6/11 上午8:03	18.06.1-ce	

在 BCS 的 [\[仓库菜单\]](#) 中也可以找到。



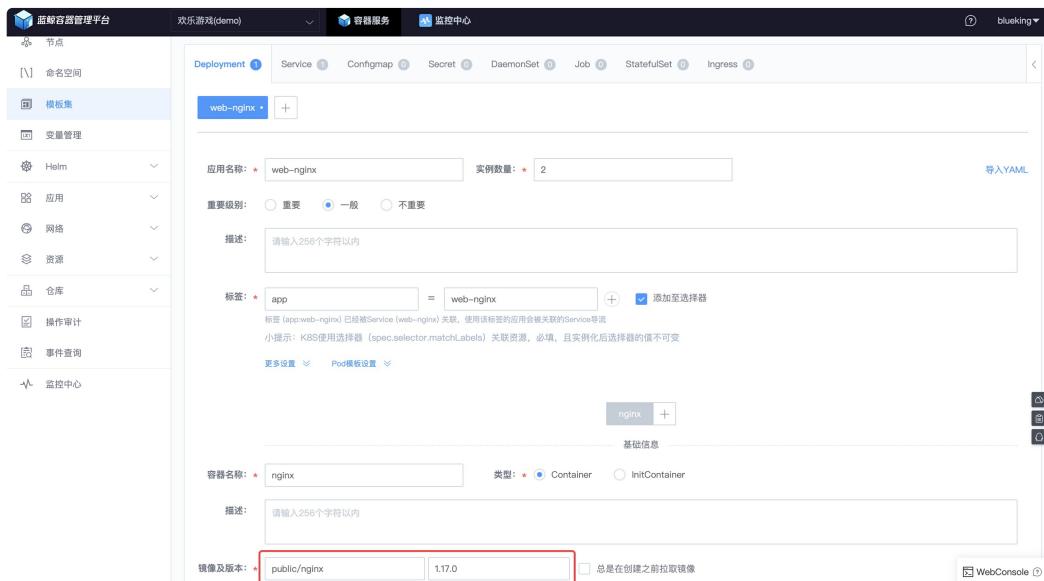
名称	大小	最近更新时间	地址
1.17.0	42.7 MB	2019-08-08 17:42:25	hub-d.**.com:443/public/nginx:1.17.0

滚动升级 Nginx：从 1.12.2 到 1.17.0

确认当前版本号为 nginx/1.12.2

```
bash [root@ip-10-0-5-94-n-bcs-k8s-40015 ~]# curl 10.0.5.94:30008 -I HTTP/1.1 200 OK Server: nginx/1.12.2 Date: Thu, 08 Aug 2019 09:11:42 GMT
```

在【模板集】的【Deployment】页面中，修改【镜像及版本】，将版本从 1.12.2 修改为在 推送 Nginx:1.17.0 至镜像仓库中上传的 Nginx 新镜像 1.17.0。



The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with various management sections like Helm, Application, Network, Resource, and Catalog. The main area is focused on a 'Deployment' configuration for a 'web-nginx' template set. Key fields include '应用名称' (Application Name) set to 'web-nginx', '实例数量' (Instance Count) set to '2', and '镜像及版本' (Image and Version) set to 'public/nginx 1.17.0'. A note at the bottom right says '总是在创建之前拉取镜像' (Always pull image before creation). There are tabs for '更多设置' (More Settings) and 'Pod模板设置' (Pod Template Settings).

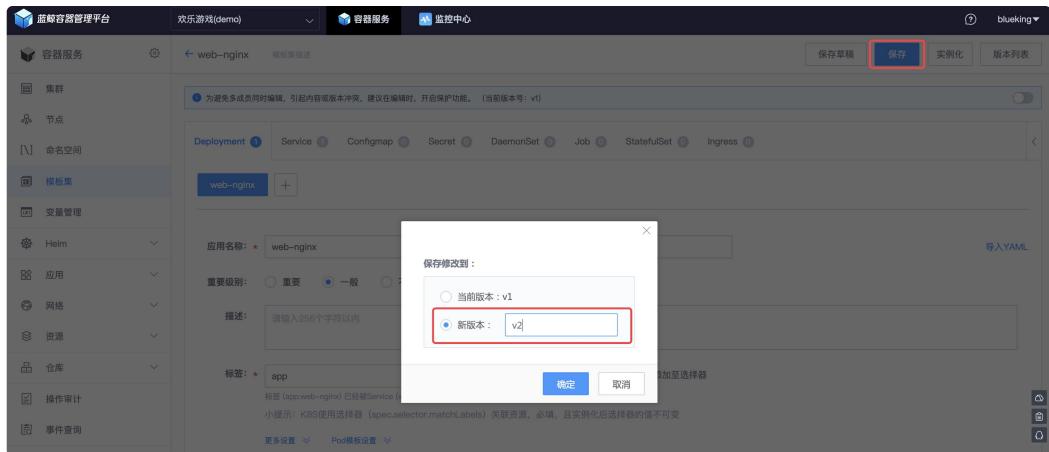
点击【更多设置】，了解默认滚动升级的更新策略。



The screenshot shows the 'More Settings' tab expanded. Under the '更新策略' (Update Strategy) section, '滚动升级' (Rolling Update) is selected. The configuration includes 'maxUnavailable' set to 1, 'maxSurge' set to 0, and 'minReadySeconds' set to 0 seconds. The 'Pod模板设置' (Pod Template Settings) tab is also visible.

- **maxUnavailable**：滚动升级期间，考虑应用容量，不可用 Pod 的数量上限
- **maxSurge**：滚动升级期间，考虑集群资源，超出期望 Pod 的数量上限
- **minReadySeconds**：滚动升级期间，考虑可用性，探测 Pod 正常后转为可用的时间

修改完镜像的版本后，接下来【保存】模板集，填写【新版本】的版本号。



接着，开始滚动升级。点击菜单【应用】->【Deployment】，找到 `web-nginx` 应用，点击【滚动升级】。

可以看到，差异点是 `image` 从 `nginx:1.12.2` 调整为 `nginx:1.17.0`

点击【确定】滚动升级后，正在更新。

通过右下角的【Web console】可以通过命令行的方式获取实例(POD)的基础信息。

以下是更新前后的对比

```
```bash root:~$ kubectl get pods -n dev -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE
web-nginx-678bb9c4fb-m8cf4 1/1 Running 0 134m 172.32.1.18 ip-10-0-5-94-n-bcs-k8s-40015
web-nginx-678bb9c4fb-nxwf4 1/1 Running 0 126m 172.32.1.17 ip-10-0-5-94-n-bcs-k8s-40015```

```

```
root:~$ kubectl get pods -n dev -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE
web-nginx-f95fc78d-cxhrq 1/1 Running 0 21s 172.32.1.20 ip-10-0-5-94-n-bcs-k8s-40015
web-nginx-f95fc78d-pqtcj 1/1 Running 0 14s 172.32.1.21 ip-10-0-5-94-n-bcs-k8s-40015```

```

可以看到 Nginx 版本已经从 **1.12.2** 更新为 **1.17.0**

```
bash [root@ip-10-0-5-94-n-bcs-k8s-40015 ~]# curl 10.0.5.94:30008 -I
HTTP/1.1 200 OK
Server: nginx/1.17.0
```

## 应用的蓝绿发布

### 情景

传统的应用更新方式是停服更新，用户在更新期间无法使用服务。

接下来，将以 Nginx 从 **1.12.2** 升级 **1.17.0** + 程序代码（index.html 的内容从 Nginx 默认页更新为 1.17.0）为例，看 BCS 中的蓝绿发布能力是如何实现不停机更新，用户无感知。

## 前提条件

- K8S 基本概念，包含 Deployment、Services；本节教程新增概念：ConfigMap、Ingress、Ingress Controllers。
- [完成 BCS 部署]

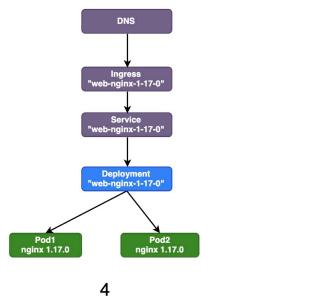
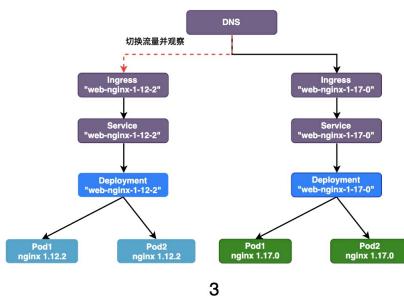
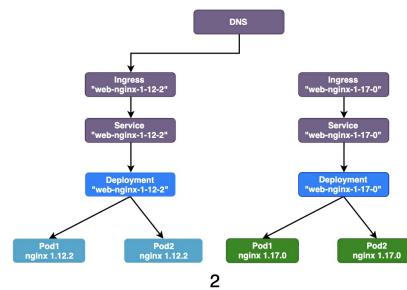
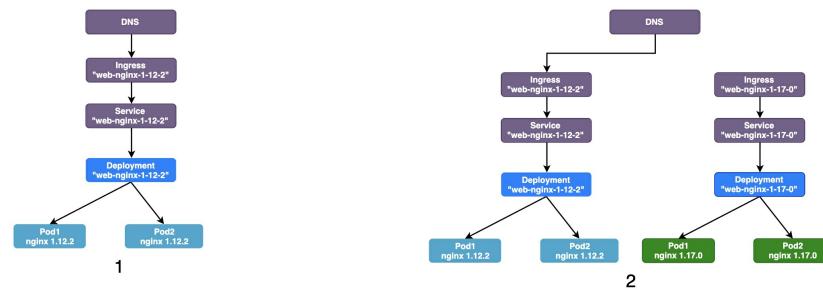
## 操作步骤

1. 应用的蓝绿发布逻辑介绍
2. 使用 K8S 资源准备版本
3. 使用 K8S 资源准备新版本
4. 切换流量并观察

## 蓝绿发布逻辑介绍

### 发布逻辑示意图

蓝绿发布，即准备 **当前运行版本** 和 **新版本** 两组实例，正式发布的时候，修改服务的域名的 DNS 记录将，将其指向新版本的 Ingress 指向的地址。



Nginx 从 1.12.2 蓝绿发布至 1.17.0

----- 删除

## 版本更新流程中引入的对象

以 Nginx 从 **1.12.2** 升级 **1.17.0** + 程序代码（index.html 的内容从 Nginx 默认页更新为 1.17.0）为例，使用以下几个新的对象：

- 程序代码或可执行文件（index.html）：Docker 镜像
- 程序或运行环境配置（nginx.conf）：ConfigMap
- 负载均衡器（LoadBalancer）+ Ingress：用户接入和负载均衡

其中 Deployment、Service 不再赘述。

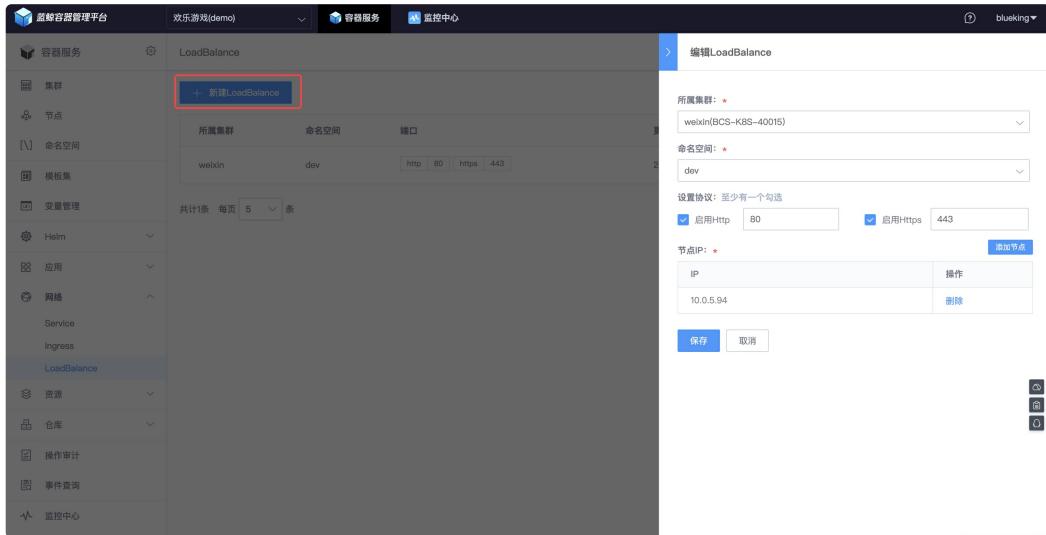
## 使用 K8S 资源准备版本

### 新增 LoadBalancer

Ingress 是 K8S 中描述用户接入的对象之一，需要配合 LB 应用才能对外提供访问。

在 BCS 中，LoadBalancer 背后的技术是 K8S 维护的 Nginx Ingress Controller。

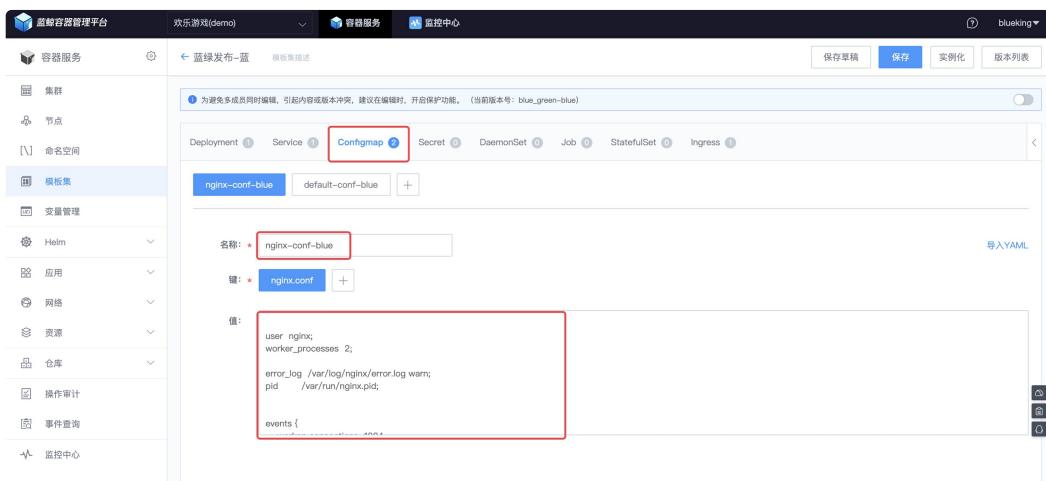
选择菜单【LoadBalancer】，点击【新建 LoadBalancer】，选择一个节点作为 LB 对外提供网络接入服务。



建议业务 Pod 不调度至 LB 所在的节点，可使用 节点亲和性（nodeAffinity）实现。

### Configmap：存放 nginx.conf

在【模板集】菜单中，选择【Configmap】，新建 nginx.conf 和 default.conf 两个 configmap，实现应用程序和配置的解耦。



### 创建 K8S 对象 Deployment、Service、Ingress

- 创建 Deployment

在【Deployment】中，填写 名称、标签（Label）、容器镜像、挂载卷（挂载 Configmap）。

The screenshot shows the BlueKing Container Management Platform interface. On the left, there is a sidebar with various navigation options: 容器服务, 集群, 节点, 命名空间, 模板集 (highlighted), Helm, 应用, 网络, 资源, 仓库, 操作审计, 事件查询, and 监控中心.

In the main area, the title is "蓝绿发布-蓝". The top bar includes "保存草稿", "保存", "实例化", and "版本列表".

**Deployment 配置:**

- 应用名称: web-nginx-blue
- 实例数量: 2
- 重要级别: 一般 (selected)
- 描述: 请输入256个字符以内
- 标签: app = web-nginx-blue (带有选择器)
- 卷设置 (Pod 中的卷设置):
  - 类型: configMap
  - 名称: nginx-conf-blue
  - 挂载路径: /etc/nginx/nginx.conf
  - 子目录: nginx.conf
  - 类型: configMap
  - 名称: default-conf-blue
  - 挂载路径: /etc/nginx/conf.d/default.conf
  - 子目录: default.conf

**Service 配置:**

- 容器名称: nginx
- 类型: Container (selected)
- 描述: 请输入256个字符以内
- 镜像及版本: public/bcs/k8s/nginx 1.12.2-autoreload
- 端口映射:
  - 名称: container-port
  - 容器端口: 80
- 更多设置 (Advanced):
  - 命令: nginx -g "daemon off;"
  - 挂载卷 (Mount Volumes):
    - 容器目录: /etc/nginx/nginx.conf
    - 子目录: nginx.conf
    - 权限: 只读 (+, -)
  - 容器目录: /etc/nginx/conf.d/default.conf
  - 子目录: default.conf
  - 权限: 只读 (+, -)

注意: 只挂载一个文件, 防止挂载整个目录

#### • 创建 Service

在【Service】中关联 Deployment 以及服务名称、暴露的端口。

The screenshot shows the BlueGreen Management Platform interface. On the left, there's a sidebar with navigation items like '容器服务', '集群', '节点', '命名空间', '模板集' (selected), '变量管理', 'Helm', '应用', '网络', '资源', '仓库', '操作审计', '事件查询', and '监控中心'. The main area has tabs for 'Deployment', 'Service' (selected), 'ConfigMap', 'Secret', 'DaemonSet', 'Job', 'StatefulSet', and 'Ingress'. Under 'Service', a 'web-nginx-blue' entry is selected. The form fields include:

- 名称: web-nginx-blue
- 关联应用: web-nginx-blue
- 关联标签: app:web-nginx-blue
- Service类型: ClusterIP
- ClusterIP: 请输入 ClusterIP  
不填或None
- 端口映射: 端口名称: http, 端口: 80, 协议: TCP, 目标端口: container~1
- 标签管理: app = web-nginx-blue

At the bottom right, there's a 'WebConsole' button.

#### • 新建 Ingress

在【Ingress】中填写【主机名】、【路径】以及绑定【Service】。

This screenshot shows the continuation of the Ingress configuration for the 'web-nginx-blue' service. The 'Ingress' tab is selected. The '基础信息' section contains:

- 主机名: blue.bk.tencent.com
- Service 名: web-nginx-blue
- Service 端口: 80
- 路径组: / → web-nginx-blue

#### 实例化模板集

保存模板集后，实例化模板集。

模板集名称	类型	容器 / 镜像	操作
蓝绿发布-蓝	自定义	nginx / nginx:1.12.2-autoreload	<button>实例化</button> 更多
web-nginx	自定义	nginx / nginx:1.17.0 apache / httpd:2.4.32	<button>实例化</button> 更多
示例模板集	自定义	container-redis-default / redis:1.0 container-nginx-default / rumpetroll-openresty:0.51 container-rumpetroll-v1 / pyrumpetroll:0.3	<button>实例化</button> 更多

可以看到对应资源已生成好。

#### • Deployment

应用名称	状态	来源	版本	应用数	最后更新时间	操作
web-nginx-blue	Running	模板集	blue_green-blue	2/2	08-16 16:20	<button>滚动升级</button> <button>扩缩容</button> 更多
web-nginx-blue	Running	模板集	blue_green-blue	2/2	08-16 16:20	<button>滚动升级</button> <button>扩缩容</button> 更多
blueking-nginx-ingress-controller	Running	Helm 模板	1.0.0	1/1	08-10 10:31	<button>滚动升级</button> <button>扩缩容</button> 更多
blueking-nginx-ingress-default-backend	Running	Helm 模板	1.0.0	1/1	08-10 10:31	<button>滚动升级</button> <button>扩缩容</button> 更多

#### • Service

Service名称	所属集群	命名空间	来源	更新时间	创建时间	更新人	操作
web-nginx-green	weixin	dev	模板集	2019-08-16 16:40:11	2019-08-16 16:40:11	blueking	<button>更新</button> <button>删除</button>
web-nginx-blue	weixin	dev	模板集	2019-08-16 16:20:52	2019-08-16 16:20:52	blueking	<button>更新</button> <button>删除</button>
blueking-nginx-ingress-default-backend	weixin	dev	Helm 模板	2019-08-10 10:30:56	2019-08-10 10:30:56	blueking	<button>更新</button> <button>删除</button>
prometheus-operated	weixin	thanos	Client	2019-08-07 23:36:54	2019-08-04 10:04:47	--	<button>更新</button> <button>删除</button>
alertmanager-operated	weixin	thanos	Client	2019-08-07 23:36:54	2019-08-04 10:04:40	--	<button>更新</button> <button>删除</button>
thanos-peers-advertise	weixin	thanos	Client	2019-08-07 23:36:54	2019-08-04 10:04:15	--	<button>更新</button> <button>删除</button>
po-prometheus-operator-thanos-sidecar	weixin	thanos	Client	2019-08-07 23:36:54	2019-08-04 10:04:15	--	<button>更新</button> <button>删除</button>

#### • Ingress

名称	所属集群	命名空间	来源	创建时间	更新时间	更新人	操作
web-nginx-green	weixin	dev	模板集	2019-08-16 16:40:11	2019-08-16 16:40:32	blueking	<a href="#">查看</a> <a href="#">删除</a>
web-nginx-blue	weixin	dev	模板集	2019-08-16 16:20:51	2019-08-16 16:21:32	blueking	<a href="#">查看</a> <a href="#">删除</a>

修改域名解析或修改 PC 上 hosts 文件（Mac 下路径为 /etc/hosts），将 Ingress 中配置的主机名解析到 LoadBalancer 中节点的外网 IP，然后打开浏览器访问。

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

以上作为线上环境运行的版本，接下来部署新版本。

## 使用 K8S 资源准备新版本

本次新版本参照微服务更新的最佳实践：将应用程序打入 Docker Image，更新 Deployment 中的镜像即更新版本。

### 制作新版本的 Docker Image

以 index.html 为应用程序，将其打入镜像，由于新版本的运行时是 Nginx 1.17.0，故以 Nginx 1.17.0 为基础镜像。

- 准备 dockerfile

```
```bash $ ll total 16 -rw-r--r-- 1 breaking staff 49B 9 10 10:55 dockerfile -rw-r--r-- 1 breaking staff 40B 9 10 10:54 index.html```
$ cat index.html
```

Welcome to BCS.

Nginx Version: 1.17.0

```
$ cat dockerfile FROM nginx:1.17.0 COPY index.html /usr/share/nginx/html ``
```

- 构建 Docker Image

```
```bash $ docker build -t bcs_nginx:1.17.0 . Sending build context to Docker daemon 3.072kB Step 1/2 : FROM nginx:1.17.0 --> 719cd2e3ed04 Step 2/2 : COPY index.html /usr/share/nginx/html --> 9e1342027c80 Successfully built 9e1342027c80 Successfully tagged bcs_nginx:1.17.0```
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
bcs_nginx 1.17.0 9e1342027c80 10 minutes ago 3.072 kB```
$ curl localhost:8080
Welcome to BCS.
```

```
$ docker images REPOSITORY TAG IMAGE ID CREATED SIZE bcs_nginx 1.17.0 9e1342027c80 18 seconds ago 109MB ``
```

- 推送镜像到仓库

```
```bash
```

```
$ docker tag bcs_nginx:1.17.0 /joyfulgame/bcs_nginx:1.17.0
```

```
$ docker push /joyfulgame/bcs_nginx:1.17.0 The push refers to repository [/joyfulgame/bcs_nginx] 8b2c2f2923c8: Pushed d7acf794921f:  
Mounted from joyfulgame/nginx d9569ca04881: Mounted from joyfulgame/nginx cf5b3c6798f7: Mounted from joyfulgame/nginx 1.17.0:  
digest: sha256:b608ac54ca92dd092529f9b86403d3a539eab030103e2e0c1a984787ece448c9 size: 1155 ``
```



更多 Docker Image 的构建方法可以参考 [docker-nginx](#)。



克隆模板集为新版本

由于新版本主要在 Deployment 的镜像版本、Ingress 绑定的主机名，以及这几个 K8S 对象的命名差别（同一个命名空间不允许重名），所以克隆模板集，然后修改即可。

点击【复制模板集】，会克隆一个模板集，命名为：蓝绿发布-绿

The screenshot shows the Blue-Green Release Management Platform interface. On the left, there's a sidebar with various navigation items like Helm, Application, Network, Resource, and Catalog. The main area is titled 'Template Set' and lists three entries:

- Blue-Green Release - Blue (最新版本: blue_green-blue) - Type: 自定义 (nginx / nginx:1.17.0)
- Blue-Green Release - 绿 (最新版本: blue_green-green) - Type: 自定义 (nginx / nginx:1.17.0)
- 示例模板集 (最新版本: init_version) - Type: 自定义 (container-redis-default / redis:1.0, container-nginx-default / rumpetroll-openresty:0.51, container-rumpetroll-v1 / pyrumpetroll:0.3)

A modal window titled 'Clone Template Set' is open over the second entry, with the input field containing 'Blue-Green Release - 绿' and a 'More' dropdown menu highlighted with a red box.

- 修改【Deployment】中的名称、标签以及镜像版本。

应用名称: * web-nginx-green 实例数量: * 2

重要级别: 一般

描述: 请输入256个字符以内

标签: app = web-nginx-green

容器名称: nginx 类型: Container

镜像及版本: bns_nginx:1.17.0 总是在创建之前拉取镜像

- 修改【Service】中的名称、关联标签。

名称: * web-nginx-green

关联应用: * web-nginx-green

关联标签: * app = web-nginx-green

- 修改 Ingress，主机名不能上一个版本一样。

The screenshot shows the BlueKing Container Service Management Platform interface. On the left, there's a sidebar with various navigation options like Container Services, Clusters, Nodes, Namespaces, Template Sets, Variables Management, Helm, Applications, Networks, Resources, Warehouses, Operation Audit, Event Monitoring, and Monitoring Center. The main area is titled 'Ingress' and shows a configuration for a service named 'web-nginx-green'. It includes fields for 'Name' (set to 'web-nginx-green'), 'TLS Settings', and 'Basic Information'. Under 'Basic Information', there's a 'Host Name' field containing 'green.bk.tencent.com' and a 'Path Group' section with a path '/'. The 'Host Name' and 'Path Group' fields are highlighted with red boxes. At the top right, there are buttons for 'Save Draft', 'Save', 'Instance', and 'List Versions'. A note at the top says 'To avoid multiple members editing simultaneously, causing content or version conflicts, it is recommended to enable protection when editing. (Current version number: green)'. A note below says 'K8S generates rules in "Network" => "LoadBalance" to create LoadBalance before it can be generated'. The bottom right corner has a 'WebConsole' button.

最后保存模板集，然后开始实例化模板集。

修改域名解析或修改 PC 上 hosts 文件（Mac 下路径为 /etc/hosts）后，访问 Ingress 中配置的主机名。

← → ⏪ ⓘ 不安全 | green.bk.tencent.com

Welcome to BCS.

Nginx Version: 1.17.0

切换流量并观察

如果是客户端业务，将请求的后端地址指向为新版本的主机名即可，如果客户端不方便更新配置，可以使用 CNAME 将域名指向到新的版本的主机名。

观察一段时间，如果没有异常，删除原版本的实例即可。

- 蓝绿发布的好处：新版本如果发现异常，可以快速的切换到老版本。
- 蓝绿发布的坏处：预备双倍的资源，直到下线老版本。不过如果有云平台，成本可控。

企业可以结合自身实际情况，选择合适的版本发布方式。

在 K8S 中部署 WordPress

情景

WordPress 是流行的开源博客程序，Helm 官方维护了 WordPress 的 Chart，接下来看在 BCS 中如何部署 WordPress，开始你的博客之旅。

前提条件

- 了解 Helm 的使用方法
- 集成 K8S 存储，例如 [将 NFS 作为 K8S PV Provisioner](#)
- Git Clone [Helm Charts](#)
- 新增 [LoadBalancer](#)

操作步骤

1. 上传 WordPress Chart 到仓库
2. 部署 WordPress
3. 访问测试

上传 WordPress Chart 到仓库

进入 [Charts](#) 本地仓库的 wordpress 目录。

```
```bash
```

```
cd charts/stable/wordpress/
```



```
总用量 84 -rw-r--r-- 1 root root 454 9月 12 11:01 Chart.yaml -rw-r--r-- 1 root root 186 9月 12 11:01 OWNERS -rw-r--r-- 1 root root 29059 9月 12 11:01 README.md -rw-r--r-- 1 root root 233 9月 12 11:01 requirements.lock -rw-r--r-- 1 root root 173 8月 27 17:00 requirements.yaml drwxr-xr-x 3 root root 4096 9月 12 11:01 templates -rw-r--r-- 1 root root 13278 9月 12 11:01 values-production.yaml -rw-r--r-- 1 root root 12925 9月 12 11:01 values.yaml ````
```

可以看到存在 requirements.yaml 文件，了解 WordPress 依赖 mariadb 的 Chart。

```
```bash
```

cat requirements.yaml

```
dependencies: - name: mariadb version: 6.x.x repository: https://kubernetes-charts.storage.googleapis.com/ condition: mariadb.enabled tags: - wordpress-database ````
```

在当前目录使用 Helm 命令下载依赖包。

```
```bash
```

## helm dep build

```
Hang tight while we grab the latest from your chart repositories... ...Successfully got an update from the "joyfulgame" chart repository
...Successfully got an update from the "stable" chart repository Update Complete. □ Happy Helming! □ Saving 1 charts Downloading
mariadb from repo https://kubernetes-charts.storage.googleapis.com/ Deleting outdated charts ````
```

推送 Chart 到仓库

```
```bash
```

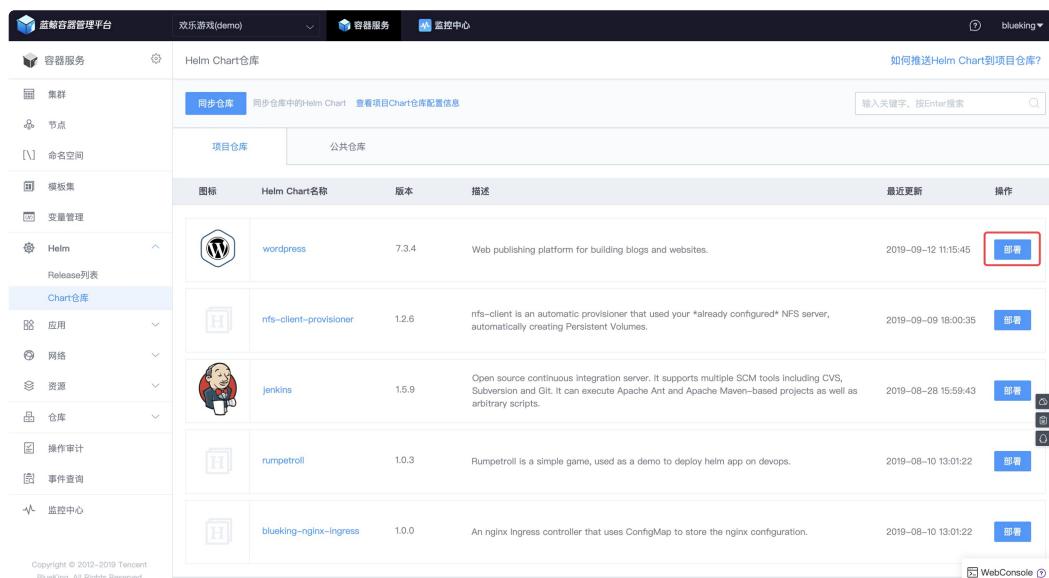
helm push . joyfulgame

```
Pushing wordpress-7.3.4.tgz to joyfulgame... Done. ````
```

在 BCS 【Chart 仓库】菜单中，点击【同步仓库】，将刚刚上传的 Chart 从仓库同步到 BCS 的界面中。

部署 WordPress

在 BCS 【Chart 仓库】菜单中，找到刚刚上传的 WordPress Chart，点击【部署】。



图标	Helm Chart名称	版本	描述	最近更新	操作
	wordpress	7.3.4	Web publishing platform for building blogs and websites.	2019-09-12 11:15:45	部署
	nfs-client-provisioner	1.2.6	nfs-client is an automatic provisioner that uses your *already configured* NFS server, automatically creating Persistent Volumes.	2019-09-09 18:00:35	部署
	jenkins	1.5.9	Open source continuous integration server. It supports multiple SCM tools including CVS, Subversion and Git. It can execute Apache Ant and Apache Maven-based projects as well as arbitrary scripts.	2019-08-28 15:59:43	部署
	rumpetroll	1.0.3	Rumpetroll is a simple game, used as a demo to deploy helm app on devops.	2019-08-10 13:01:22	部署
	blueking-ingress	1.0.0	An nginx Ingress controller that uses ConfigMap to store the nginx configuration.	2019-08-10 13:01:22	部署

可以修改 Helm 参数，例如 WordPress 管理员账号、密码等，这里重点提一下用户访问用到的 Service 和 Ingress：

- Service

使用 ClusterIP 即可，因为用户访问使用 Ingress。

```
yaml service: type: ClusterIP # HTTP Port port: 80
```

- Ingress

此处启用 Ingress，并填写绑定的主机名和路径。类似 Nginx 配置文件中 Server Section 部分的 server_name。

```

YAML模式 表单模式
Yaml
250 ##
251- ingress:
252   ## Set to true to enable ingress record generation
253   enabled: true
254
255   ## Set this to true in order to add the corresponding annotations for cert-manager
256   certManager: false
257
258   ## Ingress annotations done as key:value pairs
259   ## For a full list of possible ingress annotations, please see
260   ## ref: https://github.com/kubernetes/ingress-nginx/blob/master/docs/annotations.md
261
262   ## If this is set to true, annotation ingress.kubernetes.io/Secure-backends: "true" will automatically be set
263   ## If certManager is set to true, annotation kubernetes.io/tls-acme: "true" will automatically be set
264   annotations:
265     # kubernetes.io/ingress.class: nginx
266
267   ## The list of hostnames to be covered with this ingress record,
268   ## Most likely this will be just one host, but in the event more hosts are needed, this is an array
269   hosts:
270     - name: wordpress.bk.tencent.com
271       path: /
272
273   ## The tls configuration for the ingress
    
```

点击【预览】，可以看到 BCS 将 Charts 通过 `helm template` 命令渲染为 K8S 的对象描述文件。

```

YAML模式 表单模式
Yaml
1 apiVersion: apps/v1beta1
2 kind: StatefulSet
3 metadata:
4   annotations:
5     io.tencent.paa.creator: blueking
6     io.tencent.paa.updator: blueking
7   labels:
8     app: mariadb
9     chart: mariadb-6.8.7
10    component: master
11    heritage: Tiller
12   io.tencent.bcs.app.appid: '6'
13   io.tencent.bcs.clusterid: BCS-K8S-40015
14   io.tencent.bcs.controller.name: wordpress-1909121112-mariadb
15   io.tencent.bcs.controller.type: StatefulSet
16   io.tencent.bcs.kind: Kubernetes
17   io.tencent.bcs.monitor.level: general
18   io.tencent.bcs.namespace: dev
19   io.tencent.bcs.projectid: '5556'
20   io.tencent.bkdata.container.stdLog.droidid: '0'
21   io.tencent.paa.projectid: 79ab6164a194ac2952b497e790b967
22   io.tencent.paa.source_type: helm
23   io.tencent.paa.version: 7.3.4
24   io.tencent.bcs.cluster: BCS-K8S-40015
25   release: wordpress-1909121112
26   name: wordpress-1909121112-mariadb
27   spec:
28     replicas: 1
29     selector:
30       matchLabels:
31         app: mariadb
32         component: master
33         release: wordpress-1909121112
34         serviceName: wordpress-1909121112-mariadb
35       template:
36         metadata:
37           labels:
38             app: mariadb
39             chart: mariadb-6.8.7
40             component: master
41             io.tencent.bcs.app.appid: '6'
    
```

点击【部署】即可。

在【Release 列表】菜单中可以查看部署状态。

名称	类型	Pods
wordpress-1909130255	Deployment	1/1
wordpress-1909130255-mariadb	StatefulSet	1/1
wordpress-1909130255-mariadb	Service	-/-
wordpress-1909130255	Service	-/-
wordpress-1909130255	Ingress	-/-
wordpress-1909130255-mariadb	ConfigMap	-/-
wordpress-1909130255-mariadb-tests	ConfigMap	-/-
wordpress-1909130255-mariadb	Secret	-/-
wordpress-1909130255	Secret	-/-
wordpress-1909130255	PersistentVolumeClaim	-/-
wordpress-1909130255-mariadb-test-sgtpg	Pod	-/-
wordpress-1909130255-credentials-test	Pod	-/-

部署成功，接下来测试访问。

访问测试

修改域名解析或 PC 上 hosts 文件（Mac 下路径为 /etc/hosts），将 Ingress 中配置的主机名指向到 LoadBalancer 中节点的外网 IP，然后打开浏览器访问，可以看到 WordPress 首页。

User's Blog! — Just another WordPress site

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

user September 13, 2019 Uncategorized 1 Comment

Search ...

Search

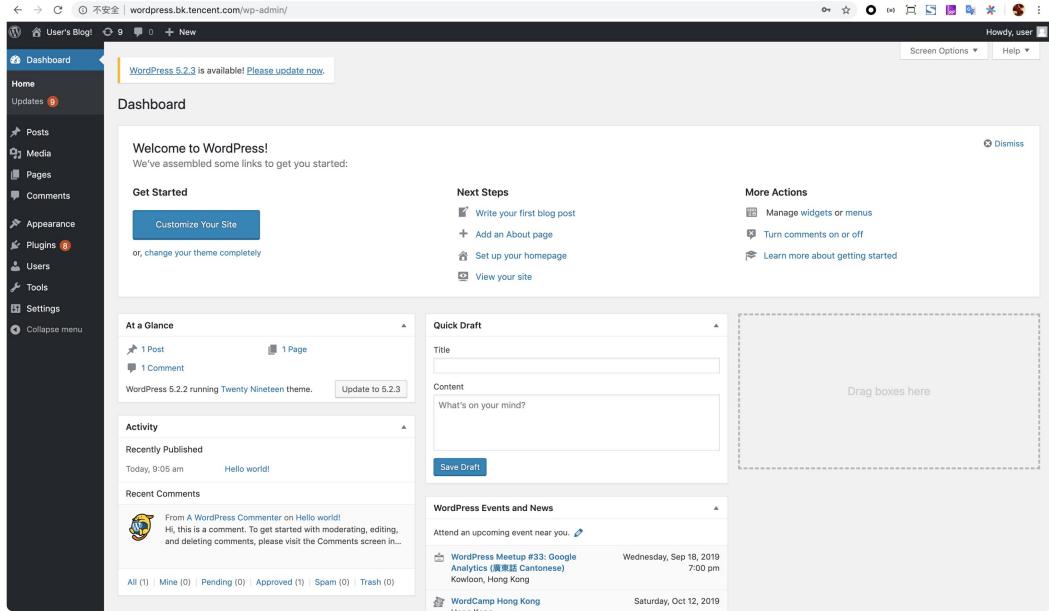
Recent Posts

Hello world!

Recent Comments

Archives

输入用户名（默认为 user）和密码登录 Wordpress 后台。



如果没有在 Chart 参数中没有设置密码，可以通过命令获取 WordPress 的 Secret。

```
```yaml
```

## kubectl get secret -n dev

```
NAME TYPE DATA AGE
wordpress-1909130255 Opaque 1 4h37m
```

## kubectl get secret/wordpress-1909130255 -n dev -o yaml

```
apiVersion: v1
data:
 wordpress-password: bFgzTmZvSHJIVg==
kind: Secret
creationTimestamp: 2019-09-13T07:01:33Z
name: wordpress-1909130255
namespace: dev
type: Opaque
```

```
echo "bFgzTmZvSHJIVg==" | base64 --decode
```

```
IX3NfoHrHV````
```

BCS 部署应用，如此简单。

## 在 K8S 中部署 GitLab

### 情景

GitLab，不仅仅一个独立部署的流行 Git 代码托管仓库，通过其 Pipeline 具备部分 CI 环节的能力，接下来看 BCS 如何快速部署 GitLab。

### 前提条件

- 了解 Helm 的使用方法
- 集成 K8S 存储，例如 将 NFS 作为 K8S PV Provisioner
- Git Clone Helm Charts
- 新增 LoadBalancer

### 操作步骤

1. 上传 GitLab Chart 到仓库
2. 部署 GitLab
3. 访问测试

### 上传 GitLab Chart 到仓库

进入 `Charts` 本地仓库的 `gitlab-ce` 目录。

```
```bash
```

cd charts/stable/gitlab-ce/

||

```
总用量 28 -rwxr-xr-x 1 root root 365 8月 27 17:00 Chart.yaml -rw-r--r-- 1 root root 2502 8月 27 17:00 README.md -rw-r--r-- 1 root root 336
8月 27 17:00 requirements.lock -rw-r--r-- 1 root root 209 8月 27 17:00 requirements.yaml drwxr-xr-x 2 root root 4096 9月 13 15:14
templates -rw-r--r-- 1 root root 3445 8月 27 17:00 values.yaml ````
```

可以看到存在 `requirements.yaml` 文件，了解 GitLab 依赖 redis、postgresql 的 Chart。

```
```bash
```

## cat requirements.yaml

```
dependencies: - name: redis version: 0.9.0 repository: https://kubernetes-charts.storage.googleapis.com/ - name: postgresql version:
0.8.1 repository: https://kubernetes-charts.storage.googleapis.com/ ````
```

在当前目录使用 Helm 命令下载依赖包。

```
```bash
```

helm dep build

```
Hang tight while we grab the latest from your chart repositories... ...Successfully got an update from the "joyfulgame" chart repository
...Successfully got an update from the "stable" chart repository Update Complete. □ Happy Helming! □ Saving 2 charts Downloading redis
from repo https://kubernetes-charts.storage.googleapis.com/ Downloading postgresql from repo https://kubernetes-
charts.storage.googleapis.com/ Deleting outdated charts ````
```

推送 Chart 到仓库

```
```bash
```

## helm push . joyfulgame

```
Pushing gitlab-ce-0.2.2.tgz to joyfulgame... Done. ````
```

在 BCS 【Chart 仓库】菜单中，点击【同步仓库】，将刚刚上传的 Chart 从仓库同步到 BCS 的界面中。

### 部署 GitLab

在 BCS 【Chart 仓库】菜单中，找到刚刚上传的 GitLab Chart，点击【部署】。

图标	Helm Chart名称	版本	描述	最近更新	操作
	gitlab-ce	0.2.2	GitLab Community Edition	2019-09-13 15:06:51	<button>部署</button>
	wordpress	7.3.4	Web publishing platform for building blogs and websites.	2019-09-12 11:15:45	<button>部署</button>
	nfs-client-provisioner	12.6	nfs-client is an automatic provisioner that uses your *already configured* NFS server, automatically creating Persistent Volumes.	2019-09-09 18:00:35	<button>部署</button>
	jenkins	1.5.9	Open source continuous integration server. It supports multiple SCM tools including CVS, Subversion and Git. It can execute Apache Ant and Apache Maven-based projects as well as arbitrary scripts.	2019-08-28 15:59:43	<button>部署</button>
	rumpetroll	1.0.3	Rumpetroll is a simple game, used as a demo to deploy helm app on devops.	2019-08-10 13:01:22	<button>部署</button>

选择 Chart 版本以及命名空间后，下方显示 Helm 参数。

```

YAML模式 表单模式
YAML初始值为创建时Chart中values.yaml内容，后续更新部署时以该YAML内容为准。YAML内容最终通过--values'选项传递给'helm template'命令

4 image: gitlab/gitlab-ce:9.4.1-ce.0
5
6 ## Specify a imagePullPolicy
7 ## 'Always' if imageTag is 'latest', else set to 'IfNotPresent'
8 ## ref: http://kubernetes.io/docs/user-guide/images/#pre-pulling-images
9
10 # imagePullPolicy:
11
12 - ## The URL (with protocol) that your users will use to reach the install.
13 ## ref: https://docs.gitlab.com/omnibus/settings/configuration.html#configuring-the-external-url-for-gitlab
14 #
15 externalUrl: http://gitlab.bk.tencent.com/
16
17 - ## Change the initial default admin password if set. If not set, you'll be
18 ## able to set it when you first visit your install.
19 #
20 # gitlabRootPassword: ""
21
22 - ## For minikube, set this to NodePort, elsewhere use LoadBalancer
23 ## ref: http://kubernetes.io/docs/user-guide/services/#publishing-services---service-types
24 #
25 serviceType: ClusterIP
26
27 - ## Ingress configuration options
28 #
29 ingress:
30 annotations:
31 # kubernetes.io/ingress.class: nginx
32 # kubernetes.io/tls-acme: "true"
33 enabled: true
34 else:
35 # - secretName: gitlab.cluster.local
36 # - hosts:
37 # - gitlab.cluster.local
38 url: gitlab.bk.tencent.com
39
40 ## Configure external secrets route

```

Copyright © 2012-2019 Tencent  
BlueKing. All Rights Reserved

部署 预览 取消 [WebConsole](#)

需调整 externalUrl、Service 和 Ingress：

- **externalUrl**

必填，安装过程需要 GitLab 访问地址。

- **Service**

使用 ClusterIP 即可，因为用户访问使用 Ingress。

```
yaml serviceType: ClusterIP
```

- **Ingress**

此处启用 Ingress，并填写 URL。

```
yaml ingress: annotations: # kubernetes.io/ingress.class: nginx # kubernetes.io/tls-acme: "true" enabled: true
```

```
tls: # - secretName: gitlab.cluster.local # hosts: # - gitlab.cluster.local url: gitlab.bk.tencent.com
```

点击【预览】，可以看到 BCS 将 Charts 通过 `helm template` 命令渲染为 K8S 的对象描述文件。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
 annotations:
 io.tencent.pdas.creator: blueking
 io.tencent.pdas.updater: blueking
 labels:
 app: gitlab-ce-1909130753-gitlab-ce
 chart: gitlab-ce-0.2.2
 heritage: Tiller
 io.tencent.bcs.opp.appid: '6'
 io.tencent.bcs.clusterid: BCS-K8S-40015
 io.tencent.bcs.controller.name: gitlab-ce-1909130753-gitlab-ce
 io.tencent.bcs.controller.type: Deployment
 io.tencent.bcs.kind: Kubernetes
 io.tencent.bcs.monitor.level: general
 io.tencent.bcs.namespace: game
 io.tencent.bkdata.basebill.dolid: '6566'
 io.tencent.bkdata.container.dolid: '0'
 io.tencent.bas.projectid: 794961640194ac2952b497e79a0b967
 io.tencent.pdas.source.type: helm
 io.tencent.pdas.version: 0.2.2
 io.tencent.bcs_cluster: BCS-K8S-40015
 release: gitlab-ce-1909130753
 name: gitlab-ce-1909130753-gitlab-ce
spec:
 replicas: 1
 template:
 metadata:
 labels:
 app: gitlab-ce-1909130753-gitlab-ce
 io.tencent.bcs.opp.appid: '6'
 io.tencent.bcs.clusterid: BCS-K8S-40015
 io.tencent.bcs.controller.name: gitlab-ce-1909130753-gitlab-ce
 io.tencent.bcs.controller.type: Deployment
 io.tencent.bcs.kind: Kubernetes
 io.tencent.bcs.monitor.level: general
 io.tencent.bcs.namespace: game
 io.tencent.bkdata.basebill.dolid: '6566'
 io.tencent.bkdata.container.dolid: '0'
 io.tencent.pdas.projectid: 794961640194ac2952b497e79a0b967
 io.tencent.pdas.source.type: helm
 io.tencent.bcs_cluster: BCS-K8S-40015
 spec:
```

点击【部署】即可。

在【Release 列表】菜单中可以查看部署状态。

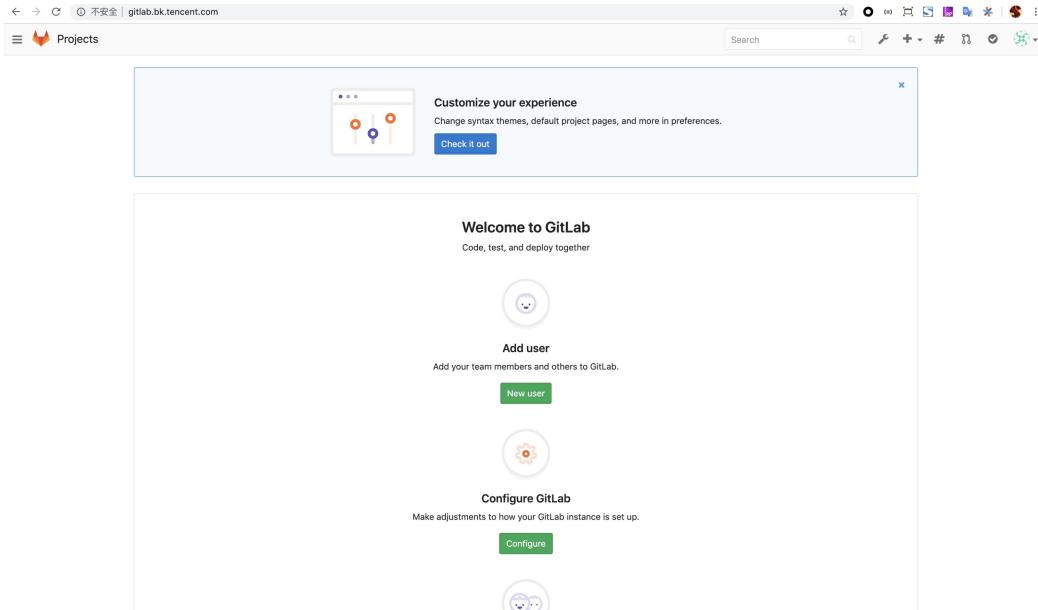
名称	类型	Pods ①
gitlab-ce-1909130302-postgresql	Deployment	1/1
gitlab-ce-1909130302-redis	Deployment	1/1
gitlab-ce-1909130302-gitlab-ce	Deployment	1/1
gitlab-ce-1909130302-postgresql	Service	-/-
gitlab-ce-1909130302-redis	Service	-/-
gitlab-ce-1909130302-gitlab-ce	Service	-/-
gitlab-ce-1909130302-gitlab-ce	Ingress	-/-
gitlab-ce-1909130302-gitlab-ce	ConfigMap	-/-
gitlab-ce-1909130302-postgresql	Secret	-/-
gitlab-ce-1909130302-redis	Secret	-/-
gitlab-ce-1909130302-gitlab-ce	Secret	-/-
gitlab-ce-1909130302-postgresql	PersistentVolumeClaim	-/-
gitlab-ce-1909130302-redis	PersistentVolumeClaim	-/-
gitlab-ce-1909130302-gitlab-ce-data	PersistentVolumeClaim	-/-
gitlab-ce-1909130302-gitlab-ce-etc	PersistentVolumeClaim	-/-

部署成功，接下来测试访问。

## 访问测试

修改域名解析或 PC 上 hosts 文件（Mac 下路径为 /etc/hosts），将 Ingress 中配置的主机名指向到 LoadBalancer 中节点的外网 IP，然后打开浏览器访问，可以看到 GitLab 首次登陆密码设置界面。

管理员用户名为 root，登陆后界面如下：



创建一个仓库，体验一下。

A screenshot of the GitLab interface showing a newly created project named 'blueking'. The left sidebar shows the project navigation menu with 'Project' selected. The main area displays a success message: 'Project "blueking" was successfully created.' Below this, there's a summary card for the project 'blueking' with a star icon, 0 issues, and a link to the project's URL. Further down, there's a section for 'Command line instructions' with 'Git global setup' and a code block for cloning the repository. At the bottom, there's a 'Create a new repository' section with a code block for initializing a new repository.

BCS 部署应用，如此简单。

## 项目管理介绍

项目是容器管理平台的一级入口，所有的资源都是围绕项目进行划分。

### 创建项目

进入项目管理页面



通过点击【新建项目】按钮，完成个人项目的创建

+ 创建项目

项目名称	项目英文名	项目说明	创建者	操作
blueking 2022-08-09 20:24:55	blueking	--	admin	编辑项目

共计1条 每页 20 条

新建项目

项目名称 \* 创建项目测试

项目英文名 \* createprojecttest

项目说明 \* 创建项目测试

6/100

确定 取消

如果是普通用户需要先申请创建项目的权限后方可有权限创建项目，等待审批完成，即可成功创建项目

系统	需要申请的权限	关联的资源实例
容器管理平台	项目创建	--

权限申请

必须申请的权限

操作	资源实例	生效条件	申请期限
项目创建		无生效条件	6个月

相关权限

操作	资源实例	生效条件	申请期限
项目查看	请选择项目实例	无生效条件	6个月
项目编辑	请选择项目实例	无生效条件	6个月
项目删除	请选择项目实例	无生效条件	6个月

理由 \*

业务使用蓝鲸容器平台需要创建独立项目

18/255

**提交** 取消

进入项目后，可以绑定项目关联的CMDB业务，主要是绑定创建集群、添加节点时提供服务器资源的业务，关联CMDB需要在CMDB业务中是“业务运维”的角色。

蓝鲸配置平台

首页 业务 资源 模型 运营分析

资源目录

业务集

业务

主机

云区域

云账户

云资源发现

业务【蓝鲸容器管理平台】

属性 关联 变更记录

基础信息

业务名	:	蓝鲸容器管理平台	生命周期	:	已上线
时区	:	Asia/Shanghai	语言	:	中文

角色

运维人员	:	[REDACTED]	产品人员	:	[REDACTED]
测试人员	:	--	开发人员	:	--
操作人员	:	--			

蓝鲸容器管理平台 集群管理

项目管理

+ 创建项目

项目名称	项目英文名	项目说明	创建者	操作
创建项目测试 2022-08-22 20:26:52	createprojecttest	创建项目测试	[REDACTED]	<a href="#">编辑项目</a>
蓝鲸 2022-08-09 20:24:55	blueking	--	admin	<a href="#">编辑项目</a>

共计 2 条 每页 20 条

蓝鲸容器管理平台 创建项目测试 集群管理

开通容器服务

业务编排类型

K8S  
k8s容器编排引擎

关联CMDB业务

蓝鲸容器管理平台

启用容器服务

蓝鲸容器管理平台 创建项目测试 集群管理

全部集群 集群 [业务: 蓝鲸容器管理平台 编排类型: K8S] 帮助

集群

节点

命名空间

模板集

变量管理

Helm

应用

欢迎使用容器服务

使用容器服务，蓝鲸将为您快速搭建、运维和管理容器集群。您可以轻松对容器进行启动、停止等操作，也可以查看集群、容器及服务的状态，以及使用各种组件服务。

请点击了解更多 >

创建容器集群 快速入门指引

### 申请加入已存在的项目

对于已存在的项目，如果没有权限，可以点击项目名称，到权限中心完成权限的申请

蓝鲸容器管理平台 集群管理

项目管理

+ 创建项目

输入关键字搜索

项目名称	项目英文名	项目说明	创建者	操作
创建项目测试 2022-08-22 21:08:10	createprojecttest	--	[redacted]	编辑项目
蓝鲸 2022-08-09 20:24:55	blueking	--	[redacted]	编辑项目

共计 2 条 每页 20 条 < 1 >

该操作需要以下权限

系统	需要申请的权限	关联的资源实例
容器管理平台	项目查看	蓝鲸

[去申请](#) 取消

蓝鲸权限中心 个人工作台 权限管理

发起需求

权限申请

临时权限申请

我的申请 我的审批

我的申请 我的审批

我的申请 我的审批

我的权限 我的分级管理员

申请自定义权限

以下是你必须申请的权限

操作	资源实例	生效条件	申请期限
项目查看	蓝鲸	无生效条件	6个月

以下相关权限，你可以按需申请

操作	资源实例	生效条件	申请期限
项目创建	无需关联实例	无生效条件	6个月
项目编辑	蓝鲸	无生效条件	6个月
项目删除	蓝鲸	无生效条件	6个月

理由 \*

工作需要查看该项目

9/255

[提交](#) 取消

## 集群管理

### 一. 启用容器服务

如果当前项目未启用过容器服务，会看到如下的启用页面。如果是管理员，这里会直接出现有权限的业务列表(业务列表来源于配置平台)；非管理员需要单独申请。

可以点击【查看帮助】，了解如何申请业务权限。

通过绑定有权限的业务，正式【启用容器服务】



## 二. 创建K8S原生集群

集群作为容器调度的基础，启用完容器服务后，首先需要创建集群。平台提供了两种创建集群的方式，一种是新建集群，另一种是导入已有集群。

欢迎使用容器服务

请点出了解更多 >

创建新集群 快速入门指引

### 创建集群 MASTER 节点

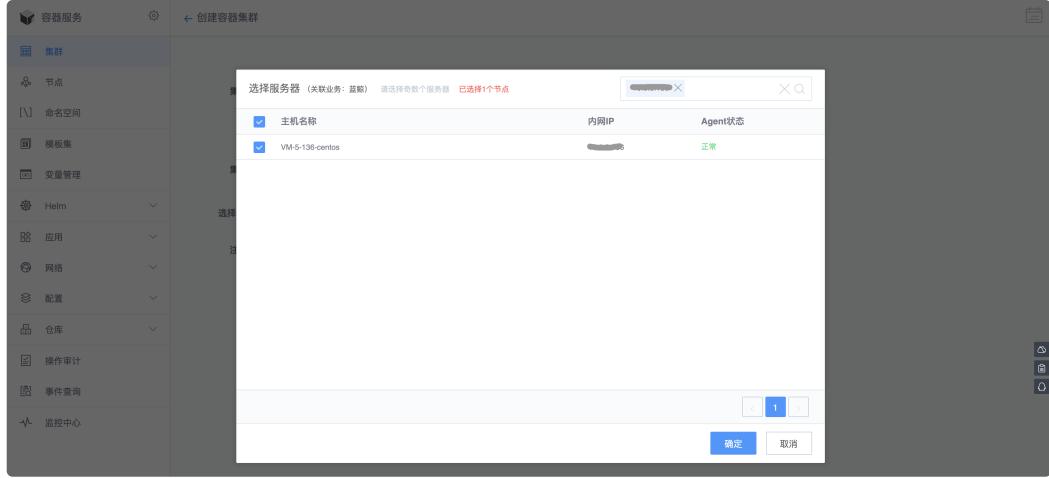
自建集群  
可自定义集群基本信息和集群版本

模板名称	描述	创建者	更新者	更新时间
自建云	--	admin	admin	2022-04-12T13:06:59+08:00

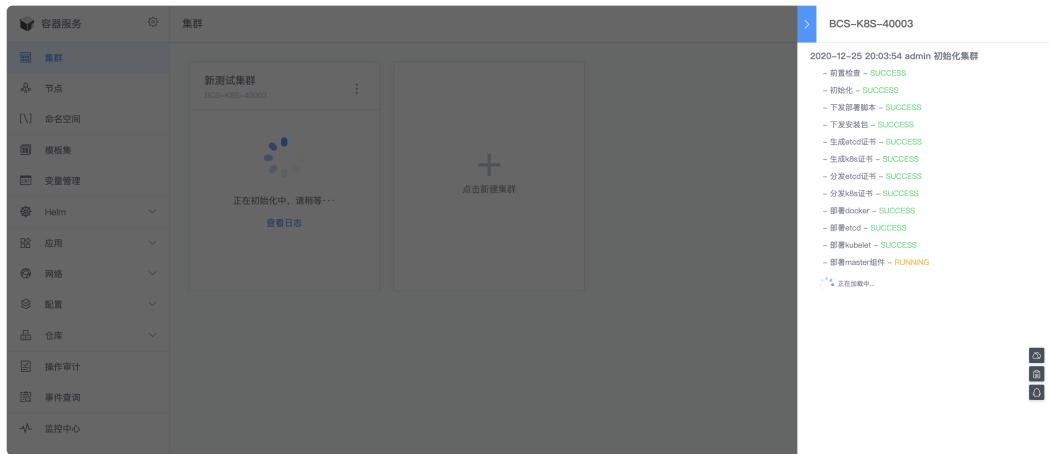
点击展开更多设置，附加参数设置，此步骤为非必选，有特殊场景时才会使用到该参数

```
| 变量名 | 描述 | 默认值 | ----- | ----- | ----- || DOCKER_LIB | Docker数据目录 |/data/bcs/lib/docker ||
DOCKER_VERSION | Docker版本 | 19.03.9 || KUBELET_LIB | kubelet数据目录 |/data/bcs/lib/kubelet || K8S_VER | 集群版本 | 1.20.11 ||
K8S_SVC_CIDR | 集群Service网段 | 10.96.0.0/12 || K8S_POD_CIDR | 集群Pod网段 | 10.244.0.0/16 |
```

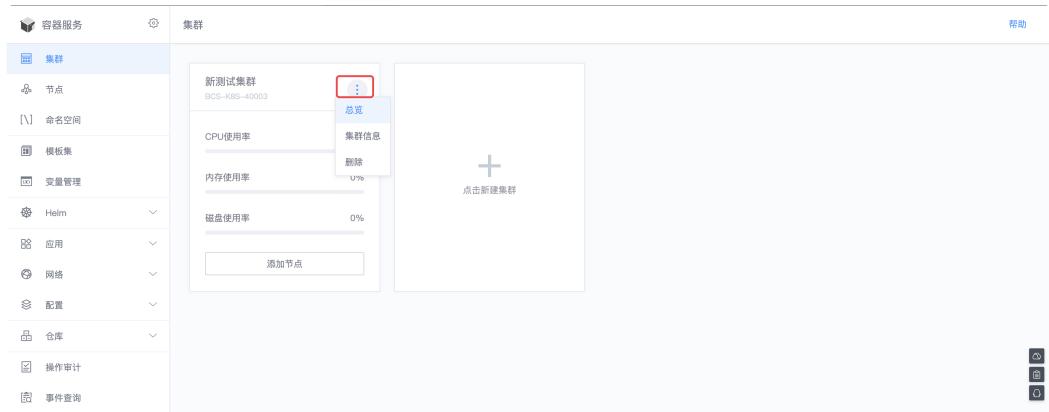
选择【新建集群】，通过【选择服务器】，完成 Master 节点的选择。



点击【确定】后，开始创建集群。可以通过【查看日志】观察集群的创建过程。



集群 Master 创建完成后，可以点击【总览】查看集群状态。



可以看到成功添加了一个 Master 节点。但其实当前集群并非可用，因为通常 Master 节点并不会设置成调度节点，而是需要添加 Node 节点(业务容器实际运行所在的主机)。

集群名称	新测试集群
集群ID	BCS-K8S-40003
状态	正常
Master数量	1个
节点数量	1个
配置	0版 0日
创建时间	2020-12-25 20:03:55
更新时间	2020-12-25 20:09:40
集群描述	新测试集群
集群变量	--

### 安装BCS-KUBE-AGENT组件

集群创建后还不能马上使用BCS SaaS管理集群，要做到管理集群的能力，必须安装好bcs-kube-agent，用于上报集群信息到bcs后台，具体操作如下：

- 注意：目前bcs-kube-agent只支持在BCS平台上创建的集群上安装，导入的集群无需安装，否则导入集群会出现异常
- 选择其中一台master角色服务器，如果master服务器上没有安装helm，可以执行下面命令安装，也可以自己去网上下载

```
 wget https://bkopen-1252002024.file.myqcloud.com/ce7/tools/helm && chmod +x helm && mv helm /usr/bin/
```

- 添加chart包镜像仓库

```
 ``shell helm repo add blueking https://hub.bktencent.com/chartrepo/blueking helm repo add blueking-dev https://hub.bktencent - 如果命名空间bcs-system不存在，则需要创建命名空间
```

```
 kubectl create ns bcs-system
```

- bcs-gateway绑定hosts，解决bcs网关访问问题，可以把域名bcs-api-gateway绑定到集群“蓝鲸蓝鲸7.0”任意一台node上，这个node最好是master角色

```
 kubectl edit cm coredns -n kube-system
```

添加以下内容：

```
 hosts { 1.1.1.1 bcs-api-gateway fallthrough } # 1.1.1.1 是集群“蓝鲸蓝鲸7.0”任意一台node，最好是master
```

```

Please edit the object below. Lines beginning with a '#' will be ignored,
and an empty file will abort the edit. If an error occurs while saving this file will be
reopened with the relevant failures.
#
apiVersion: v1
data:
 Corefile: |
 .:53 {
 errors
 health {
 lameduck 5s
 }
 ready
 kubernetes cluster.local in-addr.arpa ip6.arpa {
 pods insecure
 fallthrough in-addr.arpa ip6.arpa
 ttl 30
 }
 prometheus :9153
 forward . /etc/resolv.conf {
 max_concurrent 1000
 }
 hosts {
 [REDACTED] bcs-api-gateway
 fallthrough
 }
 cache 30
 loop
 reload
 loadbalance
 }
kind: ConfigMap
metadata:
 creationTimestamp: "2022-04-11T04:05:31Z"
 name: coredns
 namespace: kube-system
 resourceVersion: "207445"
 uid: a826175a-4080-4626-be17-a0e10daa4522

```

- 创建bcs-kube-agent所需证书

把以下内容保存到上一步集群master服务器上，文件名为：bcs-client-bcs-kube-agent.yaml

```

apiVersion: v1
data:
 ca.crt: tls.crt: tls.key:
 kind: Secret
 metadata:
 name: bcs-client-bcs-kube-agent
 namespace: bcs-system
 type: kubernetes.io/tls

```

通过webconsole或ssh到集群“蓝鲸蓝鲸7.0”

```
shell # 在集群中执行如下命令 kubectl get secret bcs-gateway-bcs-services-stack -n bcs-system -o yaml
```

把里面的ca.crt、tls.crt、tls.key里面的内容填充到bcs-client-bcs-kube-agent.yaml里

执行 `kubectl apply -f bcs-client-bcs-kube-agent.yaml` 创建好证书

- 通过helm chart安装bcs-kube-agent

把以下内容保存文件为bcs-kube-agent-values.yaml

```
global: serviceMonitor: enabled: false env: BK_BCS_clusterId: {集群ID} args: BK_BCS_API: https://bcs-api-gateway:31443 BK_BCS_APIToken: {BK_BCS_APIToken} BK_BCS_reportPath: /bcsapi/v4/clustermanager/v1/clustercredential/%s image: registry: hub.bktencent.com/dev repository: blueking/bcs-kube-agent tag: v1.25.0-alpha.6
```

把{集群ID}替换为目前你操作的集群ID，例如：BCS-K8S-40000

{BK\_BCS\_APIToken}替换为以下命名获取的字符串，获取方法如下

```
在集群 蓝鲸蓝鲸7.0 上执行以下命令 kubectl get secret bcs-password -n bcs-system -o yaml # 找到字段: gateway_token
base64解密: echo "gateway_token值" |base64 -d
```

执行以下命令安装bcs-kube-agent

```
``` helm repo update helm upgrade --install bcs-kube-agent blueking-dev/bcs-kube-agent -f ./bcs-kube-agent-values.yaml -n bcs-system --devel
```

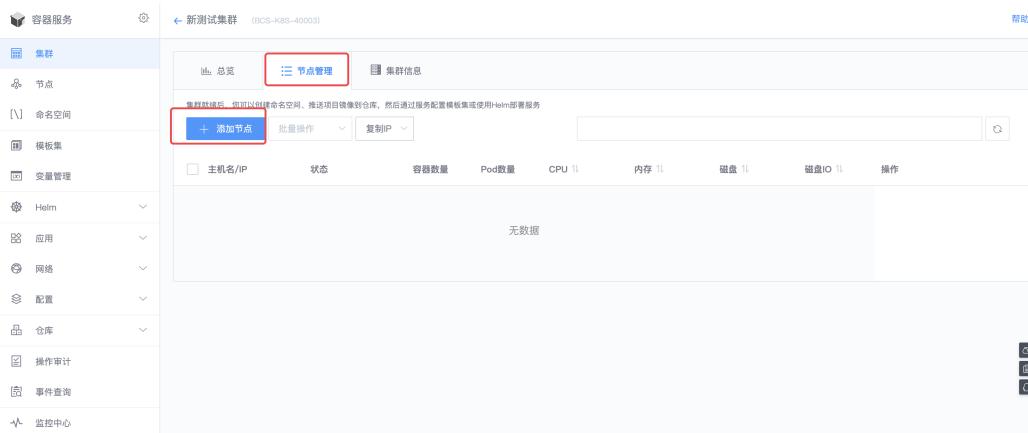
```
Release "bcs-kube-agent" does not exist. Installing it now.
NAME: bcs-kube-agent
LAST DEPLOYED: Wed Apr 13 20:14:42 2022
NAMESPACE: bcs-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

```
kubectl get pod -n bcs-system
NAME READY STATUS RESTARTS AGE
bcs-kube-agent-749d58b65-x7qkk 1/1 Running 0 9s
```

```
# pod日志没有错误就代表bcs-kube-agent安装成功
kubectl logs bcs-kube-agent-749d58b65-x7qkk -n bcs-system W0413 12:15:23.805420 1 client_config.go:552] Neither --kubeconfig nor --master was specified. Using the inClusterConfig. This might not work.
I0413 12:15:23.841994 1 report.go:74] apiserver addresses: https://1.1.1.1:6443,https://1.1.1.2:6443,https://1.1.1.3:6443 I0413 12:15:23.842031 1 report.go:77] bke-server url: https://bcs-api-gateway:31443/bcsapi/v4/clustermanager/v1/clustercredential/BCS-K8S-40000````
```

添加集群 NODE 节点

选中【节点管理】标签，点击【添加节点】。选择主机的方式和创建 Master 类似。【确定】后，节点的添加会进入初始化阶段，直到最终完成添加。



The screenshot shows the Container Service interface with the 'Nodes Management' tab selected. The main area displays a table of nodes with columns for Hostname/IP, Status, Container Count, CPU Usage, Memory Usage, Disk Usage, and Operations. A single node entry is highlighted with a red border.

【节点管理】标签除了可以查看集群的所有节点外，还提供了一些集群日常管理的操作，如【停止调度】等，具体可以参考[K8S 官方介绍](#)。

三. 导入已有集群

除了可以新建集群，容器服务也支持用户导入已有的集群。导入集群有两种方式：

- 通过集群的kubeconfig来达到BCS纳管的目的，这种导入方式优点是兼容各种各样的K8S集群，缺点是BCS只能控制K8S集群里的资源，而不能对云上其它资源进行管控，例如集群管理，节点添加，负载均衡等控制
- 通过各个云服务商的API Key方式导入集群，这种导入方式不同的云服务商有不同的API Key，目前只支持腾讯云公有云的TKE，后续会陆续支持Amazon EKS，Azure AKS，Google GKE等云服务商，这种放入方式优点是可以扩展对云资源的管控，例如集群管理，节点添加，负载均衡等控制

1. KUBECONFIG导入方式

The screenshot shows the Bluewhale Container Management Platform interface. The left sidebar has a 'Clusters' (集群) section selected. The main area features a 'Welcome to Container Service' (欢迎使用容器服务) message with a 'Learn More' (请点击了解更多) link. Below it are two buttons: 'Create Container Cluster' (创建容器集群) and 'Quick Start Guide' (快速入门指引). The 'Create Container Cluster' button is highlighted with a red box.

名称	描述	创建者	更新者	更新时间
原生K8S集群	原生K8S集群管理	admin	admin	2022-05-25T16:03:47+08:00
TKE	腾讯云容器服务 (Tencent Kubernetes Engine, TKE) 基...	evanxinli	evanxinli	2022-05-25T16:03:47+08:00

填写集群导入所需参数，导入方式选择“kubeconfig”，参数“集群kubeconfig”的获取方式一般有两种：

- 如果是云服务商提供的集群，一般在集群的基本信息可以找到，例如腾讯云公有云TKE的获取方式如下：

- 如果是自建原生K8S集群，可以在Master节点上的/root/.kube/config文件中获取

kubeconfig需同时满足以下两个条件，否则可能导致导入集群失败：

- 集群“蓝鲸社区版7.0（BCS-K8S-00000）”所有节点服务器到被导入集群APIServer网络端口连通正常，如果APIServer有防火墙的，请开通网络策略
- 腾讯云公网TKE要打开外网访问，托管集群需要添加“蓝鲸社区版7.0（BCS-K8S-00000）”集群所有Node节点的出口IP为

APIServer的访问白名单

The screenshot shows the Tencent Cloud Container Service (容器服务) interface. On the left sidebar, '集群' (Cluster) is selected. In the main area, under '集群', there's a '基本配置' (Basic Configuration) tab. The 'Ingress IP' section has a '已开启' (Enabled) switch. Below it, the 'Kubeconfig' section shows a snippet of a kubeconfig file:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: [REDACTED]
server: https://[REDACTED].tke.tencentcloud.com
token: [REDACTED]
```

- **独立集群** 需要在云产品“负载均衡”里，找到绑定TKE集群的CLB实例，实例名的命名规则一般是：\${TKE集群ID}_default_kubelb-internet，给CLB实例绑定允许访问TKE集群APIServer的IP地址或IP段安全组来保证集群APIServer的网络安全

The screenshot shows the Tencent Cloud Container Service (容器服务) interface. On the left sidebar, '集群' (Cluster) is selected. In the main area, under '负载均衡' (Load Balancing), there's a '云服务器' (Cloud Server) category. A red box highlights the '负载均衡' link under this category.

The screenshot shows the Tencent Cloud Load Balancer interface. On the left sidebar, '实例管理' (Instance Management) is selected under '负载均衡' (Load Balancer). The main content area shows a table of network interfaces. One interface, 'dns', is highlighted. The 'Security Groups' tab is active. A note at the top right says: '为了为您提供更优质的服务，腾讯云计划于2022年1月31日停止基础网络产品的续费，基础网络产品整体将于2022年12月31日正式下线。离开腾讯云基础网络 VPC 下对这些产品提供服务，查看公告'.

注意：如果“蓝鲸社区版7.0（BCS-K8S-00000）”集群的节点是云上服务器，请先确认好节点服务器是走服务器外网还是走nat出口后再配置APIServer白名单或安全组

- 提供的kubeconfig用户是cluster-admin角色，否则会因为权限不足导致管控失败

The screenshot shows the Tencent Cloud Kubernetes Engine (TKE) interface. On the left sidebar, '集群' (Cluster) is selected under '容器服务'. The main content area shows a table of ClusterRoles. One role, 'cert-manager-role', is highlighted. A note at the top right says: '为了保证托管集群的稳定性，自2022年4月30日起，腾讯云容器服务 TKE 会将所有集群角色，在集群的命名空间自动应用一组资源配额。详细请参考[资源配额说明](#)'.

获取到kubeconfig后，可以在导入集群页面进行有效性测试，如果测试通过，“导入”按钮会置为可用状态，否则请根据测试失败的错误提示处理错误，一般错误都是kubeconfig不同时满足上面讲的两个条件导致的

蓝鲸容器管理平台

集群管理 > 集群 > 导入集群

集群名称: kubeconfig导入集群测试
导入方式: kubeconfig 云服务商
集群描述: kubeconfig导入集群测试
集群kubeconfig > 文件导入
文件导入
kubeconfig可用性测试
导入 取消

Copyright © 2012-2022 Tencent BlueKing. All Rights Reserved

蓝鲸容器管理平台

集群管理 > 集群 > 导入集群

集群名称: kubeconfig导入集群测试
导入方式: kubeconfig 云服务商
集群描述: kubeconfig导入集群测试
集群kubeconfig > 文件导入
文件导入
kubeconfig可用性测试
导入 取消

Copyright © 2012-2022 Tencent BlueKing. All Rights Reserved

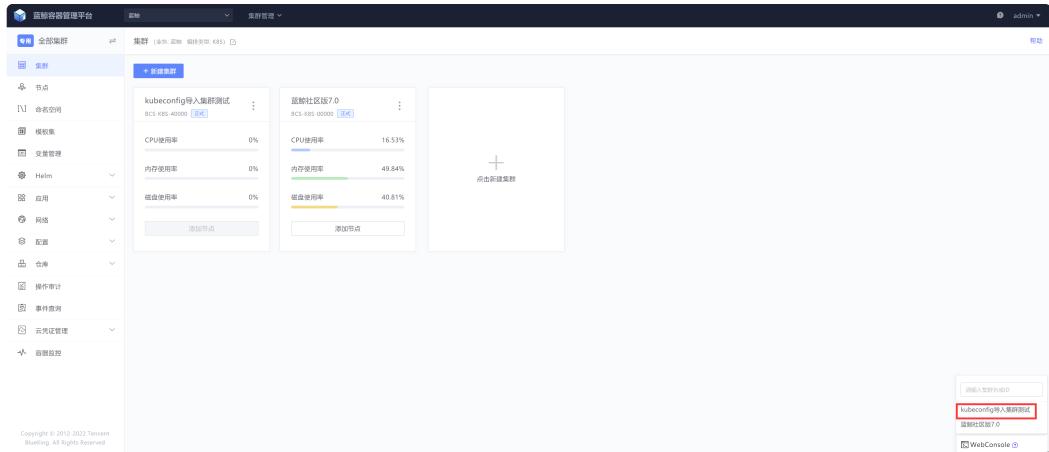
蓝鲸容器管理平台

集群管理 > 集群 > 新建集群

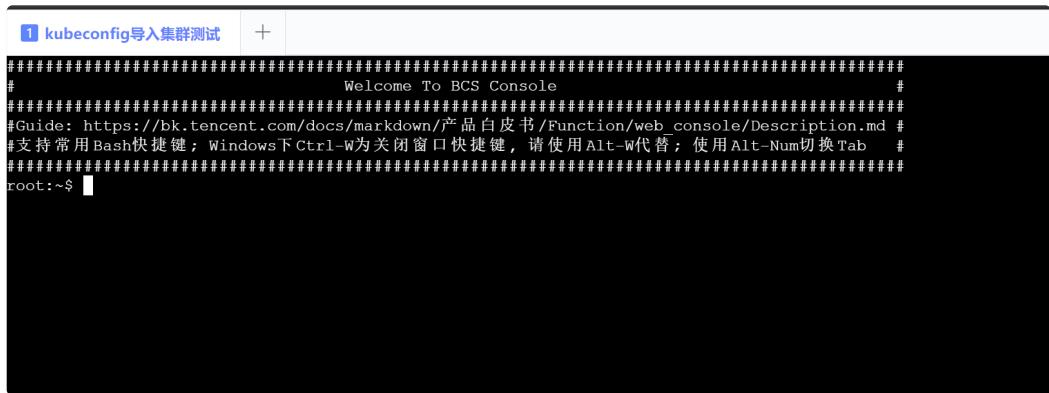
集群名称: BCS-k8s-40000
集群描述: BCS-k8s-40000
CPU使用率: 0%
内存使用率: 0%
磁盘使用率: 0%
添加节点
+ 新建集群
集群: (空) 蓝鲸社区版7.0
BCS-k8s-00000 [编辑]
CPU使用率: 20.83%
内存使用率: 50.07%
磁盘使用率: 40.81%
添加节点
+ 点击新建集群
WebConsole

Copyright © 2012-2022 Tencent BlueKing. All Rights Reserved

导入集群成功后，等待1分钟左右，使用webconsole验证是否导入成功



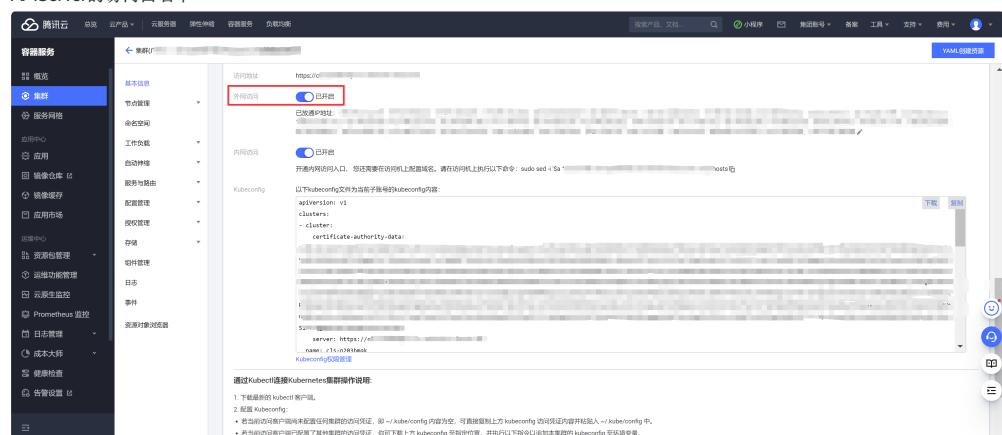
可以正常进入webconsole代表导入集群成功



2. 云服务商API密钥导入方式

通过云服务商API密钥导入集群与kubecfg导入方式一样，需同时满足以下两个条件，否则可能导致导入集群失败：

- 集群“蓝鲸社区版7.0（BCS-K8S-00000）”所有节点服务器到被导入集群APIServer网络端口连通正常，如果APIServer有防火墙的，请开通网络策略
- 腾讯云公网TKE要打开外网访问，托管集群需要添加“蓝鲸社区版7.0（BCS-K8S-00000）”集群所有Node节点的出口IP为APIServer的访问白名单



The screenshot shows the Bluewhale Cluster Management Platform interface. On the left, there's a sidebar with categories like '全部集群', '节点', '命名空间', '模板集', '变量管理', 'Helm', '应用', '网络', '配置', and '仓库'. The main area shows a cluster named '蓝鲸社区版7.0' (BCS-K8S-00000) with resource usage metrics (CPU, Memory, Disk) and a '添加节点' (Add Node) button. A red box highlights the cluster name and its status.

- 独立集群 需要在云产品“负载均衡”里，找到绑定TKE集群的CLB实例，实例名的命名规则一般是：\$(TKE集群ID)_default_kubelb-internet，给CLB实例绑定允许访问TKE集群APIServer的IP地址或IP段安全组来保证集群APIServer的网络安全

The screenshot shows the Tencent Cloud Container Service console. The left sidebar has sections like '概览', '集群' (selected), '服务网格', '应用中心', '应用', '镜像仓库', '镜像缓存', '应用市场', '运维中心', '资源包管理', '运维功能管理', '云原生监控', 'Prometheus 监控', '日志管理', '成本大师', '健康检查', and '告警设置'. The main area has tabs for '最近访问', '容器服务', '访问密钥', '访问管理', '账号中心', and '负载均衡'. Under '负载均衡', there are sections for '计算' (Cloud Server), '基础存储服务' (Object Storage), '网络' (Load Balancing), '人脸识别', '游戏服务', 'AI 泛娱乐', '金融服务', '教育服务', '人体识别', '文字识别', '汽车服务', '医疗服务', '图像分析与处理', and '图像分析'. A red box highlights the '负载均衡' tab.

The screenshot shows the Tencent Cloud Load Balancing console. The left sidebar has sections like '概述', '实例管理' (selected), '证书管理', '个性化配置', '日志中心', and '访问日志'. The main area shows an instance named 'clb-1234567890-abcd-internet' with status '正常' (Normal). It lists 'ID/Name', 'IP', '状态', 'VIP', '可用区', '健康类型', '健康阈值', '健康状态', '计费模式', and '操作'. A red box highlights the '实例管理' tab.

The screenshot shows the Tencent Cloud Load Balancing console. The left sidebar has sections like '概述', '实例管理' (selected), '证书管理', '个性化配置', '日志中心', and '访问日志'. The main area shows a security group configuration for a CLB instance. It includes tabs for '基本信息', '监听器管理', '重定向配置', '监控', and '安全组' (selected). The '安全组' tab shows '已绑定安全组' (1: bcs-kubeconfig 用途: 管理; 2: bcs-kubeconfig 用途: 管理) and '规则配置' (Inbound Rules: 1. bcs-kubeconfig 用途: 管理; 2. bcs-kubeconfig 用途: 管理). A red box highlights the '安全组' tab.

注意：如果“蓝鲸社区版7.0（BCS-K8S-00000）”集群的节点是云上服务器，请先确认好节点服务器是走服务器外网还是走nat出口后再配置APIServer白名单或安全组

- 提供的kubeconfig用户是cluster-admin角色，否则会因为权限不足导致管控失败

The screenshot shows the Tencent Cloud Container Service Cluster Management interface. On the left sidebar, '集群' (Cluster) is selected. In the main area, 'ClusterRole' is selected under '授权管理'. A table lists several ClusterRole entries, each with columns for '名称' (Name), 'Labels', '账号用户名' (Account Username), and '操作' (Operation). One row, 'cert-manager-controller-role', is highlighted with a blue background.

满足以上两个先决条件后就可以开始通过云服务商API密钥导入集群了

The screenshot shows the BlueWhale Container Management Platform Cluster Import interface. On the left sidebar, '集群' (Cluster) is selected. In the main area, the '导入集群' (Import Cluster) form is displayed. The '导入方式' (Import Method) section shows 'kubeconfig' and '云服务商' (Cloud Service Provider) as options, with '云服务商' selected and highlighted with a red box. Other fields include '集群名称' (Cluster Name), '集群描述' (Cluster Description), '云服务商' (Cloud Service Provider) set to 'TKE', '云凭证' (Cloud Certificate) dropdown, '所属区域' (Region) dropdown, and 'TKE集群ID' (TKE Cluster ID) dropdown. At the bottom are '导入' (Import) and '取消' (Cancel) buttons.

- 导入方式选择“云服务商”
- 云服务目前只支持TKE，后续会支持到Amazon EKS, Azure AKS, Google GKE等云服务商
- 如果没有云凭证需要先创建一个，云凭证创建流程如下：
- 首先在腾讯云公有云上使用主账户创建一个子账户（至少具备TKE集群管理、CVM管理、负载均衡网络管理权限），用于做集群管理，强烈不建议使用主账户
- 在子账户下创建API密钥，如果已经存在API密钥可以忽略此步骤

● 把创建的密钥录入到BCS中

The screenshot shows the 'Cluster Management' section of the Bluewhale Container Management Platform. A red box highlights a newly created certificate entry in the list:

名称	描述	SecretID	SecretKey	关联集群	创建时间	操作
腾讯云TKE集群管理凭证	腾讯云TKE集群管理凭证	... (redacted)	... (redacted)	-	2022-05-26 21:25:42 +08:00	查看

创建完云凭证后继续完成导入操作

The screenshot shows the 'Import Cluster' dialog. A red box highlights the 'Import' button at the bottom of the form.

Form fields include:

- 集群名称: 云服务商导入集群测试
- 导入方式: 云服务商 (radio button selected)
- 集群描述: 云服务商导入集群测试
- 云服务商: TKE
- 云凭证: 腾讯云TKE集群管理凭证
- 所属区域: 华东地区(上海)
- TKE集群ID: bcs-ingress-controller测试

- 所属区域: 通过云凭证拉取凭证有权限的区域列表, 选择要导入集群的所属区域
- TKE集群ID: 通过云凭证拉取所属区域下存在的TKE集群列表

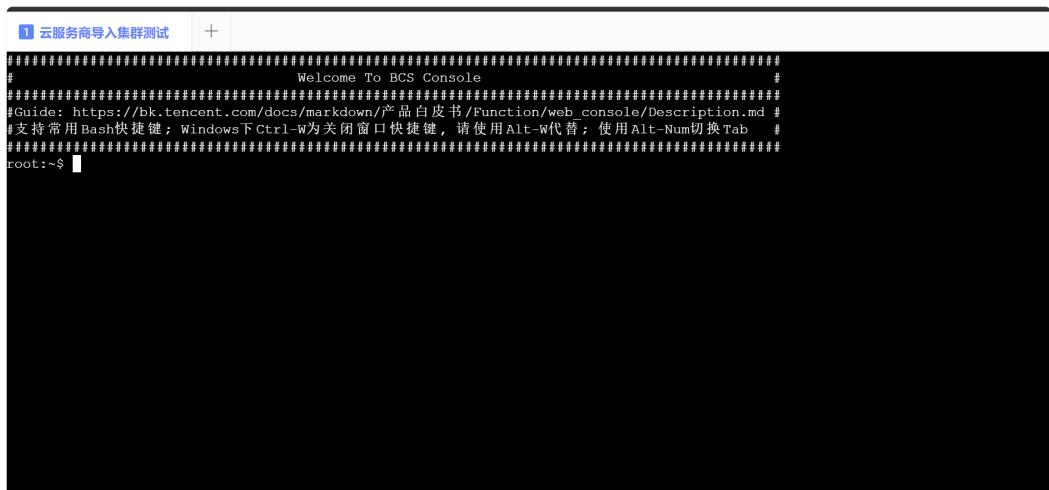
点击“导入”按钮, 完成集群导入

The screenshot shows the cluster list page. A red box highlights the imported cluster entry:

名称	状态	操作
云服务商导入集群测试 BCS-K8S-40001 [正式]	正常	... (three dots)
kubeconfig导入集群测试 BCS-K8S-40000 [正式]	正常	... (three dots)
蓝鲸社区版7.0 BCS-K8S-00000 [正式]	正常	... (three dots)

导入集群成功后, 等待1分钟左右, 使用webconsole验证是否导入成功

可以正常进入webconsole代表导入集群成功



节点管理

菜单【节点】是对集群管理的补充，该功能菜单主要是支持用户按照项目和集群的维度，查询节点信息，并提供如【设置标签】等操作。更多操作在【集群】->【概览】->【节点管理】中。

节点自动扩缩容

节点自动扩缩容功能介绍

节点自动扩缩容功能使用Cluster Autoscaler组件自动调整集群节点数量，当面对以下情况时：
- 集群中存在因资源不足无法运行的pod；
- 集群某些节点一段时间内资源利用率都处于较低状态，且这些节点上的pod可调度到其他现存节点上

节点自动扩缩容功能前置条件

- 使用云服务器商导入集群



详情请参考[集群管理的导入已有集群](#)

- 云凭证必须有以下权限

```
json { "version": "2.0", "statement": [ { "effect": "allow", "action": [ "tke:*" ], "resource": [ "*" ] }, { "effect": "allow", "action": [ "cvm:DescribeSecurityGroups", "cvm:DescribeImages", "cvm:DescribeInstances" ], "resource": [ "*" ] }, { "effect": "allow", "action": [ "vpc:Describe*" ], "resource": [ "*" ] }, { "effect": "allow", "action": [ "as: *" ], "resource": [ "*" ] } ] }
```

节点池与Autoscaler组件配置

新建节点池

节点池信息

节点池名称: 在该名字的节点池内自动调节, 不会超出设定范围。

节点数量范围: 0 ~ 10

禁用应用节点池

Labels Taints

实例创建策略: 首选可用区 (子网) 优先 多可用区 (子网) 打散

重试策略: 快速重试 间隔递增重试 不重试

扩容模式: 释放模式 关机模式

节点配置

镜像提供方: 公共镜像 市场镜像

操作系统:

机型	规格	CPU	内存	配置费用	状态
<input type="radio"/> 标准型S4	S4.LARGE8	4核	8G	¥1.28元/小时起	售卖
<input type="radio"/> 标准型S3	S3.LARGE8	4核	8G	¥1.26元/小时起	售卖
<input checked="" type="radio"/> 标准型SA2	SA2.LARGE8	4核	8G	¥0.5元/小时起	售卖
<input type="radio"/> 标准型S5	S5.LARGE8	4核	8G	¥0.6元/小时起	售卖
<input type="radio"/> 标准型SR1	SR1.LARGE8	4核	8G	¥0.5元/小时起	售卖
<input type="radio"/> 标准型SK1	SK1.LARGE8	4核	8G	¥0.54元/小时起	售卖
<input type="radio"/> 标准均衡化...	SN3ne.LARGE8	4核	8G	¥1.28元/小时起	售卖
<input type="radio"/> 标准型SA1	SA1.LARGE8	4核	8G	¥0.5元/小时起	售卖
<input type="radio"/> 标准型S2	S2.LARGE8	4核	8G	¥1.25元/小时起	售卖
<input type="radio"/> 标准型S1	S1.LARGE8	4核	8G	¥1.22元/小时起	售卖

每页: 10 条

购买数据盘 分配免费公网IP

设置root密码: 确认root密码:

安全组:

支持子网:

子网ID	子网名称	可用区	剩余IP数	不可用原因
<input type="checkbox"/> subnet-ra709qba	压测临时	ap-guang...	4085	--
<input type="checkbox"/> subnet-qtvx52jbk	gz-7	ap-guang...	4075	--
<input type="checkbox"/> subnet-ejxo2xbo	gz-6	ap-guang...	219	--
<input type="checkbox"/> subnet-tbxao2u50	agent子网...	ap-guang...	218	--
<input type="checkbox"/> subnet-bkplwfu	版本测试	ap-guang...	140	--
<input type="checkbox"/> subnet-9tq07bg6	agent子网	ap-guang...	4083	该可用区...
<input type="checkbox"/> subnet-83u5pqyy	其他	ap-guang...	250	该可用区...
<input type="checkbox"/> subnet-hpnemgrm	社区预留...	ap-guang...	252	该可用区...

容器目录:

运行时组件: docker containerd

运行时版本:

自定义脚本:

技术支持 | 社区论坛 | 产品官网
Copyright © 2012-2022 Tencent
Blueking. All Rights Reserved. V1.27.0

WebConsole

开启 Cluster Autoscaler 组件

开启 Cluster Autoscaler 组件必须至少有一个状态为“正常”的节点池，否则无法开启

快速验证节点自动扩缩容功能

一般节点自动扩缩容是结合K8S集群的HPA功能一起完成，HPA具体功能详情请见 [Pod 水平自动扩缩](#)

这里为了快速验证节点自动扩缩容功能，我们手工触发节点的扩容与缩容

资源计算方式

集群资源使用不是计算真实的节点资源使用率，而是使用workload实例的Request值来计算，具体情况请参考为 [Pod 和容器管理资源](#)

触发扩容

触发扩容条件

- 参数“触发扩容资源阈值”为100%（默认），存在因资源不足不可调度的 Pod，即 Pod 的 Condition 中 PodScheduled 为 False，且通过原生调度算法判断确实不能将该 Pod 调度到现有节点上
- 参数“触发扩容资源阈值”为非100%，当集群资源使用率大于等于设定阈值触发扩容

模拟触发扩容

根据集群节点资源情况来决定创建测试 workload 实例数，例如：集群中目前有 1 个节点，这个节点的配置为 4 核 8G，那我创建一个 workload，实例数为 5，每个实例的 Request 为 1 核，就必须会触发一次节点扩容

```

yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-test
  namespace: default
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: 'nginx:latest'
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "1"

```

`yaml apiVersion: apps/v1 kind: Deployment metadata: name: deployment-test namespace: default labels: app: nginx spec: replicas: 5 selector: matchLabels: app: nginx template: metadata: labels: app: nginx spec: containers: - name: nginx image: 'nginx:latest' ports: - containerPort: 80 resources: requests: cpu: "1"`

Pod	事件	标签	注解
deployment-test-5c5d8998b0-... nginx:latest		Status: Running	Ready: 1/1
deployment-test-5c5d8998b0-... nginx:latest		Status: Running	Ready: 1/1
deployment-test-5c5d8998b0-... nginx:latest		Status: Pending	Ready: 0/0
deployment-test-5c5d8998b0-... nginx:latest		Status: Pending	Ready: 0/0
deployment-test-5c5d8998b0-... nginx:latest		Status: Pending	Ready: 0/0

Cluster Autoscaler配置

扩缩容记录

事件类型	事件信息	开始时间	结束时间
扩容节点池	集群BC5-rg-MGlynpA扩容节点数1	2022-10-08 17:04:54	
扩容节点池	集群BC5-rg-MGL...扩容节点数1	2022-10-08 17:05:18	
创建节点池	集群BC5-rg-MGL...创建节点池BC5-rg-MGL...	2022-10-08 16:23:52	2022-10-08 16:24:22

节点池管理

节点池 ID (名称)	节点数量范围	节点数	机型	操作系统	节点状态	操作
BC5-rg-MGlynpA (ce-test)	0 - 10	2	SA2.LARGE8	TencentOS Server 3.1 (TK4)	正常	扩缩容

Deployment

deployment-test Ready 5/5 Up-to-date 5 Available 5

CPU 使用率

内存 使用率

Pod

名称	镜像	Status	Ready	Restarts	Host IP	Pod IP	Node	Age	操作
deployment-test-5c5c889cb0-...	nginx:latest	Running	1/1	0	10.0.5.153	172.17.0.252	10.0.5.153	11m16s	日志 查看调度
deployment-test-5c5c889cb0-...	nginx:latest	Running	1/1	0	10.0.5.153	172.17.0.251	10.0.5.153	11m16s	日志 查看调度
deployment-test-5c5c889cb0-...	nginx:latest	Running	1/1	0	10.0.5.184	172.17.1.197	10.0.5.184	11m16s	日志 查看调度
deployment-test-5c5c889cb0-...	nginx:latest	Running	1/1	0	10.0.5.184	172.17.1.196	10.0.5.184	11m16s	日志 查看调度
deployment-test-5c5c889cb0-...	nginx:latest	Running	1/1	0	10.0.5.184	172.17.1.195	10.0.5.184	11m16s	日志 查看调度

触发缩容

触发缩容条件

- 节点处于空闲状态
- 节点装箱率低于一定阈值（cpu、内存 默认50%），且其上的Pod可以被调度到其余节点
- 以上两种情况之一持续一段时间后将被缩容
- 缩容时也会考虑剩余资源比例，如果缩容一个节点会导致不符合剩余资源比例，则不会缩容该节点
- 扩容后20分钟是缩容冷却期，此时间段内不会执行缩容操作

模拟触发缩容

删除 模拟触发扩容的workload “deployment-test”，等待20分钟之后再看

Deployments

名称	命名空间	镜像	Ready	Up-to-date	Available	Age	操作
deployment-test	default	nginx:latest	5 / 5	5	5	15m44s	删除

The screenshot shows the 'Cluster Autoscaler' configuration page. At the top, there's a search bar and tabs for '基本配置' (Basic Configuration) and '节点池' (Node Pool). Below the tabs, there's a table for managing node pools. A modal window titled '扩缩容记录' (Scaling Log) is open, showing a list of events from October 8, 2022, including node addition and removal logs.

编辑节点池

This screenshot shows the 'Edit Node Pool' page. It includes sections for '基本配置' (Basic Configuration) and '节点池管理' (Node Pool Management). The '节点池管理' section contains a table with columns for node pool ID, range, count, machine type, operating system, status, and operations. A red box highlights the '操作' (Operation) column in the table.

目前只支持编辑“节点池信息”，由于腾讯云接口限制暂不支持“节点配置”编辑，如需更换节点配置，请新建一个节点池，然后关闭老的节点池即可（如果节点池中有节点不要删除节点池，关闭节点池即可，后续扩容节点不会使用老的节点池；如果节点池中没有节点，也可以删除老的节点池）

This screenshot shows the 'Edit Node Pool Information' page. It includes sections for '节点池信息' (Node Pool Information) and '节点配置' (Node Configuration). The '节点池信息' section allows setting the node pool name, range, and configuration like labels and taints. The '节点配置' section includes fields for instance creation strategy, rescheduling strategy, and scaling mode.

关闭与删除节点池

如果在Cluster Autoscaler组件开启的情况下，然后又只有1个节点，为了保证Cluster Autoscaler组件的正常运行，是不可以关闭或删除节点池的，需要先关闭Cluster Autoscaler组件

在停用Cluster Autoscaler组件后，如果节点池已存在节点，也是不能删除节点池的，需要先删除

蓝鲸容器管理平台

集群管理 资源地图

概览

节点

命名空间

Helm 应用 网络 配置 组件库 事件查询 云凭证管理 容器监控 日志采集

技术支撑 | 社区论坛 | 产品官网 Copyright © 2012-2022 Tencent

节点总览

节点数: 3 / 3

集群信息

弹性扩缩容

Cluster Autoscaler配置 Cluster Autoscaler

基本配置

状态: 已启用
允许Unready节点: 3个
扩缩容检测时间间隔: 10秒
unready节点超过集群节点: 45% 时停止自动扩缩容

自动扩缩容配置

扩缩容算法: random
等待节点提供最长时间: 900秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
触发扩容资源阈值 (CPU内存): 50%
等待 Pod 退出最长时间: 600秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
执行扩容等待时间: 600秒
扩容后判断缩容时间间隔: 1200秒
宽容失败后重试时间间隔: 180秒
单次扩容最大节点数: 10个

自动缩容配置

允许缩容节点:

节点池管理

节点池 ID (名称): BCS-rg-MGlypcA (ca-test)
节点数量范围: 0 ~ 10
节点数量: 1
机型: SA2.LARGE8
操作系统: CentOS Server 3.1 (TK4)
节点池状态: 正常

每页: 10 条

操作: 新建节点池 编辑节点池

操作: 调整节点池后再删除节点池

蓝鲸容器管理平台

集群管理 资源地图

概览

节点

命名空间

Helm 应用 网络 配置 组件库 事件查询 云凭证管理 容器监控 日志采集

技术支撑 | 社区论坛 | 产品官网 Copyright © 2012-2022 Tencent

节点总览

节点数: 3 / 3

集群信息

弹性扩缩容

Cluster Autoscaler配置 Cluster Autoscaler

基本配置

状态: 已启用
允许Unready节点: 3个
扩缩容检测时间间隔: 10秒
unready节点超过集群节点: 45% 时停止自动扩缩容

自动扩缩容配置

扩缩容算法: random
等待节点提供最长时间: 900秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
触发扩容资源阈值 (CPU内存): 50%
等待 Pod 退出最长时间: 600秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
执行扩容等待时间: 600秒
扩容后判断缩容时间间隔: 1200秒
宽容失败后重试时间间隔: 180秒
单次扩容最大节点数: 10个

自动缩容配置

允许缩容节点:

节点池管理

节点池 ID (名称): BCS-rg-MGlypcA (ca-test)
节点数量范围: 0 ~ 10
节点数量: 1
机型: SA2.LARGE8
操作系统: CentOS Server 3.1 (TK4)
节点池状态: 正常

每页: 10 条

操作: 新建节点池

操作: 点击节点数进入节点池管理

蓝鲸容器管理平台

集群管理 资源地图

概览

节点

命名空间

Helm 应用 网络 配置 组件库 事件查询 云凭证管理 容器监控 日志采集

技术支撑 | 社区论坛 | 产品官网 Copyright © 2012-2022 Tencent

节点总览

节点数: 3 / 3

集群信息

弹性扩缩容

Cluster Autoscaler配置 Cluster Autoscaler

基本配置

状态: 已启用
允许Unready节点: 3个
扩缩容检测时间间隔: 10秒
unready节点超过集群节点: 45% 时停止自动扩缩容

自动扩缩容配置

扩缩容算法: random
等待节点提供最长时间: 900秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
触发扩容资源阈值 (CPU内存): 50%
等待 Pod 退出最长时间: 600秒
连试两次检测时间间隔: 0秒
unready节点扩容等待时间: 1200秒
执行扩容等待时间: 600秒
扩容后判断缩容时间间隔: 1200秒
宽容失败后重试时间间隔: 180秒
单次扩容最大节点数: 10个

自动缩容配置

允许缩容节点:

管理节点数量

注意: 若节点已开启自动伸缩, 则数据将不会被集群自动调整

节点池名称: ca-test
节点数范围: 0 ~ 10
节点数量: 1

节点名称: 状态 操作
ca-test: 正常 停止缩容

每页: 10 条

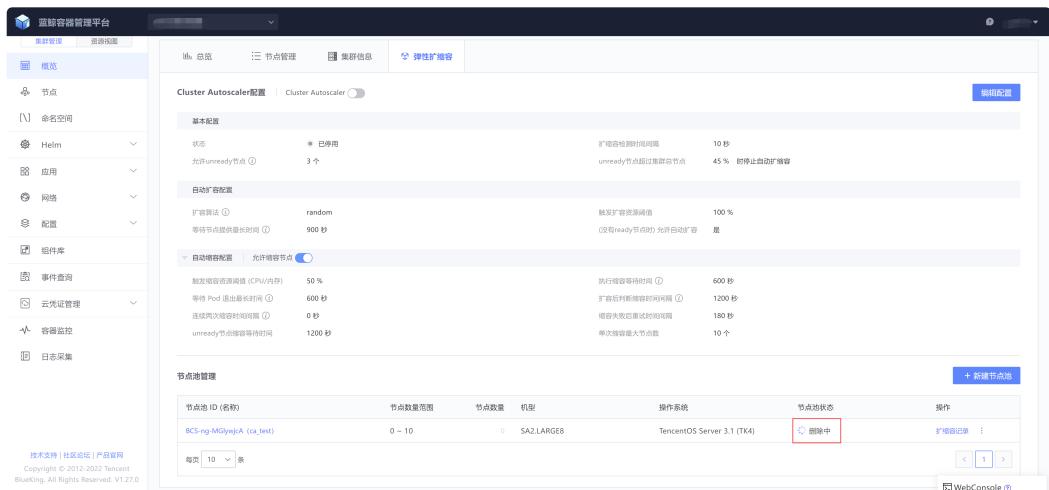
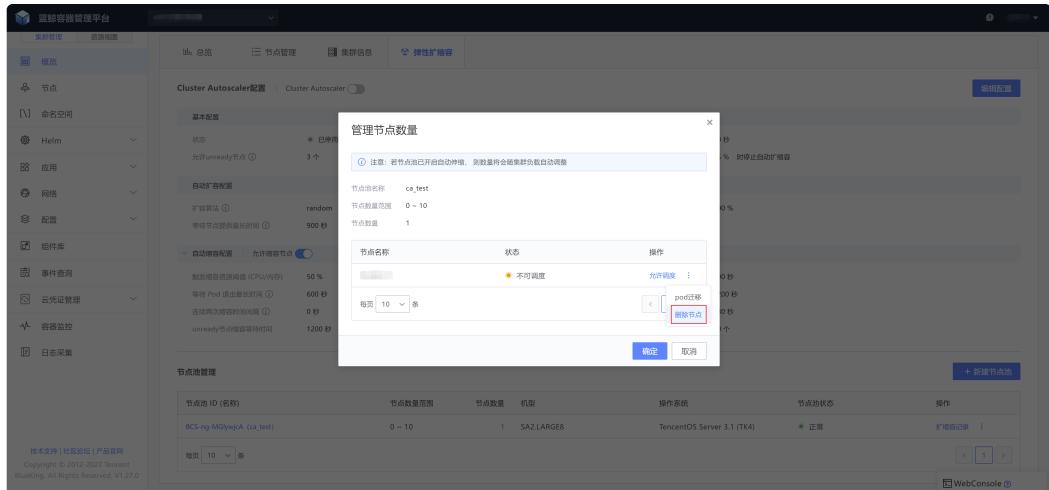
操作: 确定 取消

节点池管理

节点池 ID (名称): BCS-rg-MGlypcA (ca-test)
节点数量范围: 0 ~ 10
节点数量: 1
机型: SA2.LARGE8
操作系统: CentOS Server 3.1 (TK4)
节点池状态: 正常

每页: 10 条

操作: 新建节点池



Cluster Autoscaler组件参数说明

| 参数类型 | 参数名称 | 对应组件参数 | 默认值 | 参数说明 | ----- | ----- | ----- | ----- | ----- | ----- |
----- | 基本配置 | 状态 | 无 | 已停用 | Cluster Autoscaler组件状态，有“正常”与“已停用”两种状态 | 基本配置 | 扩缩容检测时间间隔 | scan-interval | 10秒 | Cluster Autoscaler多久执行一次循环 | 基本配置 | 允许unready节点 | ok-total-unready-count | 3个 | 自动扩缩容保护机制，集群中unready节点大于允许unready节点数量，且unready节点的比例大于设置的比例，会停止Cluster Autoscaler功能，否则Cluster Autoscaler功能正常运行 | 基本配置 | unready节点超过集群总节点百分比 | max-total-unready-percentage | 45% | 自动扩缩容保护机制，集群中unready节点大于允许unready节点数量，且unready节点的比例大于设置的比例，会停止Cluster Autoscaler功能，否则Cluster Autoscaler功能正常运行 | 自动扩容配置 | 扩容算法 | expander | random | random: 在有多个节点池时，随机选择节点池

least-waste: 在有多个节点池时，以最小浪费原则选择，选择有最少可用资源的节点池

most-pods: 在有多个节点池时，选择容量最大（可以创建最多Pod）的节点池 | 自动扩容配置 | 触发扩容资源阈值 | buffer-resource-ratio | 100% | 集群资源CPU或内存使用率达到多少时触发扩容（Request），100%为workload实例触发Pending事件后触发扩容 | 自动扩容配置 | 等待节点提供最长时间 | max-node-provision-time | 900秒 | 如果节点池在设置的时间范围内没有提供可用资源，会导致此次自动扩容失败 | 自动扩容配置 | 没有ready节点时允许自动扩容 | scale-up-from-zero | true | 当没有 ready 节点时允许 CA 扩容 | 自动缩容配置 | 允许缩容节点 | scale-down-enabled | true | 是否允许缩容节点 | 自动缩容配置 | 触发缩容资源阈值 | scale-down-utilization-threshold | 50% | 集群资源CPU或内存使用率（Request）低于阈值触发缩容节点流程 | 自动缩容配置 | 执行缩容等待时间 | scale-down-unneeded-time | 600秒 | Cluster Autocaler组件评估集群可以缩容多久后开始执行缩容，防止集群容量在短时间内或高或低设置的缩容阈值造成频繁扩缩容操作 | 自动缩容配置 | 等待Pod退出最长时间 | max-graceful-termination-sec | 600秒 | 缩容节点时，等待 pod 停止的最长时间（不会遵守 terminationGracefulPeriodSecond，超时强杀） | 自动缩容配置 | 扩容后判断缩容时间间隔 | scale-down-delay-after-add | 1200秒 | 扩容节点后多久才继续缩容判断，如果业务自定义初始化任务所需时间比较长，需要适当上调此值 | 自动缩容配置 | 连续两次缩容时间间隔 | scale-down-delay-after-delete | 0秒 | 缩容节点后多久再继续缩容节点，默认设置为0，代表与扩缩容检测时间间隔设置的值相同 | 自动缩容配置 | 缩容失败后重试时间间隔 | scale-down-delay-after-failure | 180秒 | 缩容节点失败后多久，再继续缩容节点 | 自动缩容配置 | unready节点缩容等待时间 | scale-down-unready-time | 1200秒 | 节点 unready 状态后多久才被缩容（如果业务自定义初始化

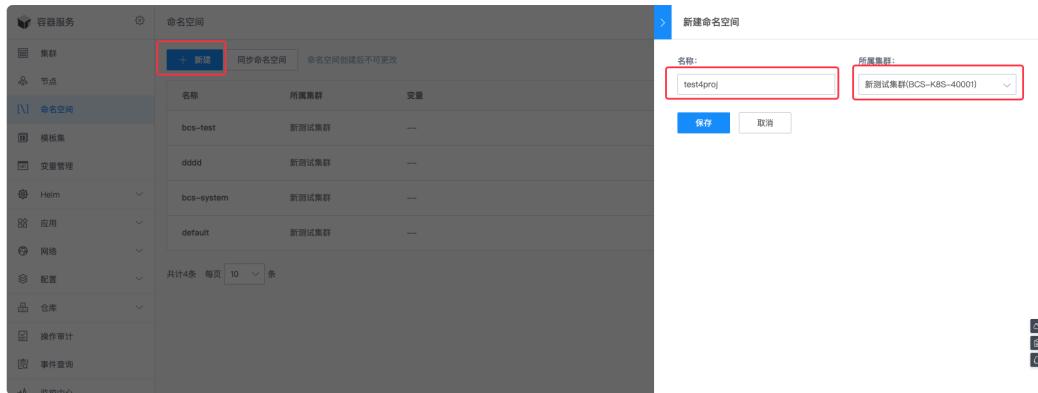
任务所需时间比较长，需要修改此参数) || 自动缩容配置 | 单次缩容最大节点数 | max-empty-bulk-delete | 10个 | CA一次缩容空节点的最大数量 |

命名空间

在一个K8S集群中可以使用命名空间(namespace)创建多个“虚拟集群”，它是多个用户或服务之间划分集群资源的一种方法，具备隔离性。

创建命名空间

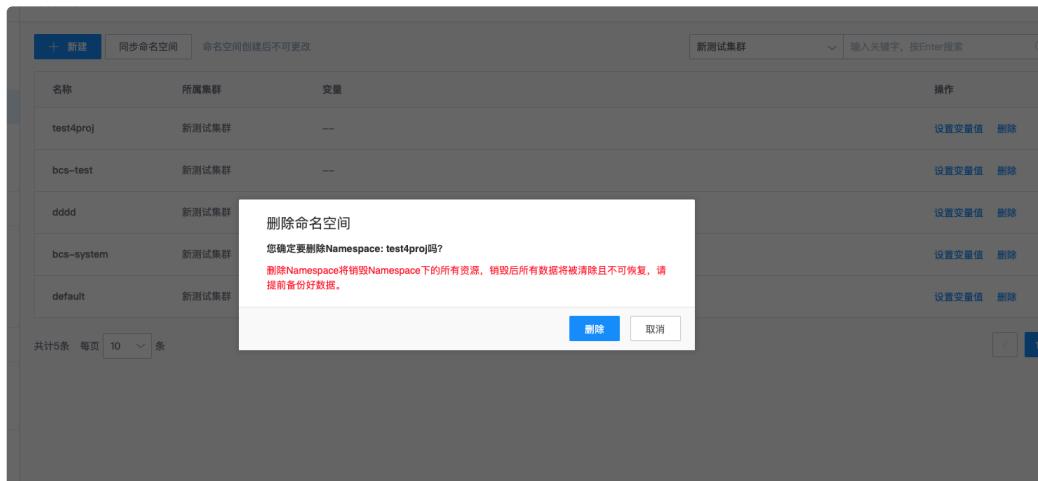
点击【新建】按钮，输入命名空间名称，同时选择实际要下发到的集群，完成命名空间的创建。



The screenshot shows the 'Namespaces' management interface. On the left sidebar, '命名空间' (Namespaces) is selected. In the main content area, there is a modal window titled '新建命名空间' (Create Namespace). Inside the modal, there are two input fields: 'Name' (set to 'test4proj') and '所属集群' (Selected Cluster) (set to '新测试集群(BCS-K8S-40001)'). Below these fields are 'Save' and 'Cancel' buttons. The 'Name' field is highlighted with a red box.

删除命名空间

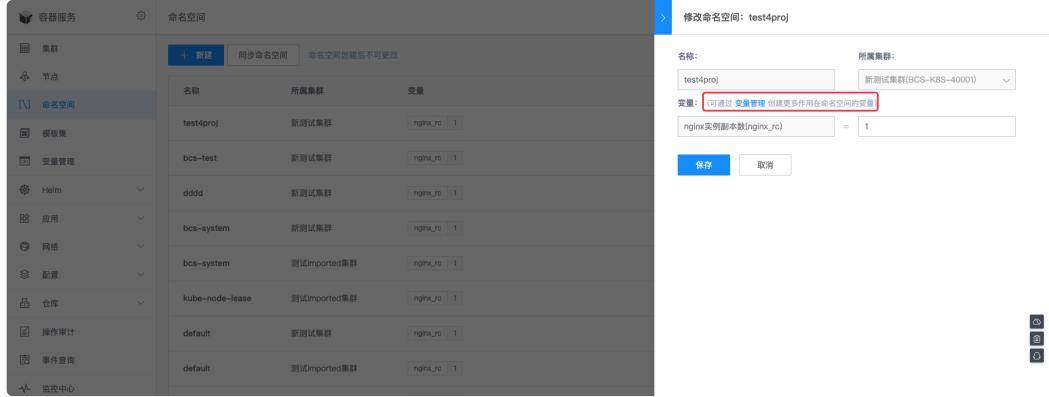
命名空间并不支持直接更新名字，只能通过删除再新建的方式来完成“重命名”。然而，命名空间的删除是一个比较高危的操作，它将删除该命名空间下所有资源，操作时需要谨慎。



The screenshot shows the 'Namespaces' list interface. A modal window titled 'Delete Namespace' is open over the list. It contains a question 'Do you really want to delete Namespace: test4proj?', a warning message 'Deleting Namespace will delete all resources in it. All data will be cleared and cannot be recovered. Please back up data before proceeding.', and two buttons at the bottom: 'Delete' (highlighted with a red box) and 'Cancel'. The 'Delete' button is highlighted with a red box.

设置变量值

命名空间变量值是针对模板集/Helm 来使用的。在左侧菜单的【变量管理】中，用户可以创建命名空间变量，对于这些变量值的调整，可以通过【设置变量值】完成。



业务接入 Helm

本文先简单介绍什么是 Helm，然后以蓝鲸小游戏为例，介绍如何使用蓝鲸容器服务部署 Helm Release。

什么是 Helm

Helm 是 Kubernetes 的一个包管理工具，用来简化应用的部署和管理。可以把 Helm 比作 CentOS 的 yum 工具。

使用 Helm 可以完成以下事情：

- 管理 Kubernetes manifest files
- 管理 Helm 安装包 Charts
- 基于 Chart 的 Kubernetes 应用分发

更多可以参考 [Helm 官方文档](#)。



蓝鲸容器服务支持 Helm3 和 Helm2，其中 Helm2 使用了 `helm template` 生成 Kubernetes YAML，但是部署没有使用 Helm Tiller，而是直接使用的 `kubectl apply`。



Helm Chart 仓库

平台会给项目分配 Helm Chart 仓库，用于存放项目的 Chart，仓库的读写操作均需要密码；另外，所有项目只读共享一个公共仓库，用于共享公共资源，比如社区中常用开源组件的部署方案，所有项目对公共仓库享有只读权限。

如何推送Helm Chart到项目仓库?

输入关键字, 按Enter搜索

最近更新 操作

图标 Helm Chart名称 版本 描述

gitlab-ce gitlab 2.10.1 项目Chart仓库配置信息

gitlab 1.0.0

rancher 1.0.0

jenkins 1.5.9 Open source continuous Integration server. It supports multiple SCM tools including CVS, Subversion and Git. It can execute Apache Ant and Apache Maven-based projects as well as arbitrary scripts.

wordpress 7.2.2 Web publishing platform for building blogs and websites.

如果用户的 Chart 需要推送到公共仓库，可以在 Harbor 页面的 `public` 项目下的 `Helm Charts` 上传。

中文简体

项目

public 系统管理员

镜像仓库 Helm Charts 成员 复制 标签 日志 配置管理

上傳 刪除 下載

名称	状态	版本	创建时间
blueking-nginx-ingress	正常	1	2019年3月19日
rumptroll	正常	1	2019年3月19日

1 - 2 共计 2 条记录

推送业务 Helm Chart 到仓库

点击 [【如何推送 Helm Chart 到项目仓库】](#) 指引，平台会为每个项目生成对应的 Chart 推送命令，可直接复制使用。

图标	Helm Chart名称	版本	描述	最近更新	操作
	gitlab-ce	0.2.2	GitLab Community Edition	2019-09-02 16:18:40	<button>部署</button>
	gitlab	2.2.1	Web-based Git-repository manager with wiki and issue-tracking features.	2019-08-29 14:55:18	<button>部署</button>
	rancher	2.2.8	Install Rancher Server to manage Kubernetes clusters across providers.	2019-08-29 14:54:21	<button>部署</button>
	jenkins	1.5.9	Open source continuous integration server. It supports multiple SCM tools including CVS, Subversion and Git. It can execute Apache Ant and Apache Maven-based projects as well as arbitrary scripts.	2019-08-28 15:59:43	<button>部署</button>
	wordpress	7.2.2	Web publishing platform for building blogs and websites.	2019-08-28 11:48:37	<button>部署</button>

建议参照指引完成 Helm Chart 的推送。

如何推送Helm Chart到项目仓库

开始之前, 本文假设您已将业务部署方案改成 Helm Chart 格式。为了方便您快速上手, 本文将以蓝鲸小游戏 Chart (rumpetroll)为例, 说明如何推送 Chart 到仓库。
注意事项: 文档内容根据项目生成, 其中的账号信息均为项目的真实账号, 请妥善保管。

1 安装 Helm

安装 Helm
方式一: 包管理工具

```
# package manager for Mac
brew install kubernetes-helm

# package manager for Windows
choco install kubernetes-helm

# cross-platform system package manager
gofish install helm
```

方式二: 手动下载二进制
[下载地址](#)
初始化 Helm

```
helm init --client-only --skip-refresh
```

安装 Helm Chart 推送工具
方式一: 命令安装

```
helm plugin install https://github.com/chartmuseum/helm-push
```

方式二: 手动下载二进制
[下载地址](#)

2 添加 Helm Chart 仓库

注意: 仓库的账号密码为项目私有, 请妥善保管

```
helm repo add joyfulgame https://hub-d.m.k8s.io.com:443/chartrepo/joyfulgame/ --username=156496958
```

使用 Helm 发布应用

使用容器服务 Helm 部署 Chart

参考上一步推送完 Chart 之后, 用户可以在【项目仓库】栏目中看到新推送的 Chart, 如果没有可以点击【同步仓库】进行同步。

- 创建入口
 - 【Release 列表】菜单下, 点击【部署 Helm Chart】
 - 跳转到【Chart 仓库】, 选择目标 Chart 点击【部署】

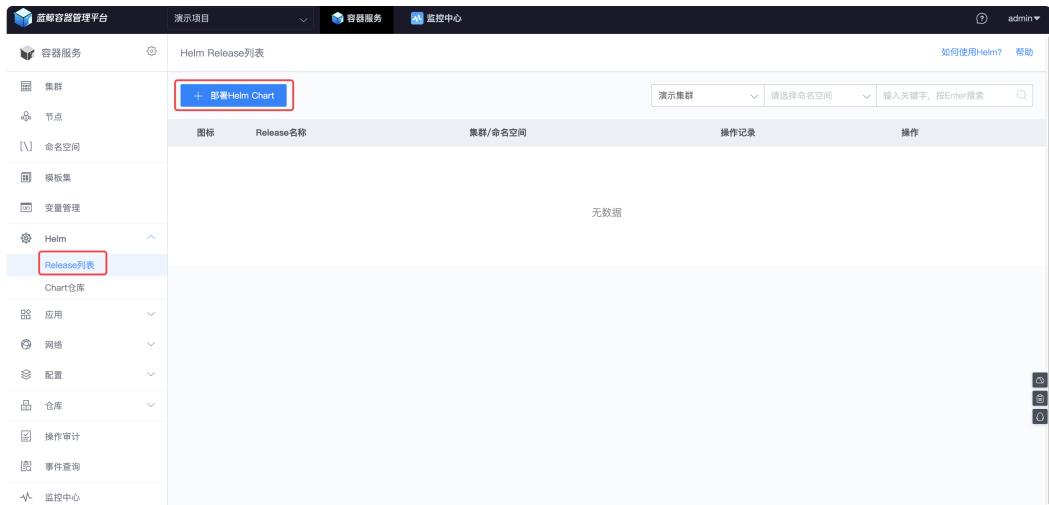


图 1. Helm Release 创建入口

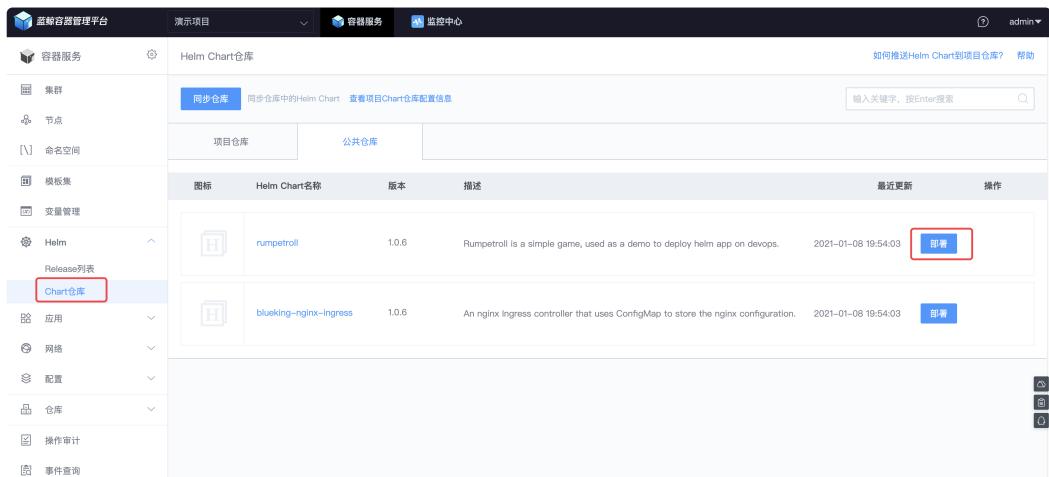


图 2. Helm Chart 部署按钮

- 参数填写

- 名称: 只能输入字母, 数字或者 -
- Chart 版本: Chart 版本
- 命名空间: 需要实例化的目标集群+命名空间
- 用于实例化 Helm Release 的参数, 包括直接编辑 yaml 和表单两种方式:
 - yaml 文本编辑, 相当于给 `helm template` 命令传递 `-f, --values` 参数
 - 表单, 主要为了提升输入体验和参数校验, 只有在 Chart 中定义了 `questions.yaml` 文件时, 创建 Helm Release 页面才会生成根据 `questions.yaml` 内容生成表单页面。

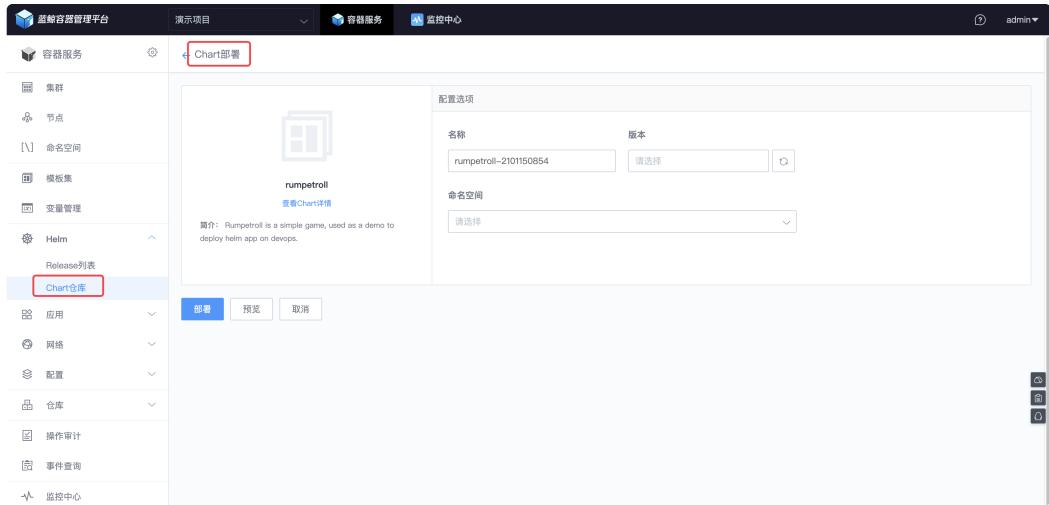


图 3. Helm Release 创建参数页面

- Helm Release 预览

◦ 在真正创建 Helm Release 之前，可以通过【预览】功能查看，将要生成的 K8S 资源配置是否符合预期。

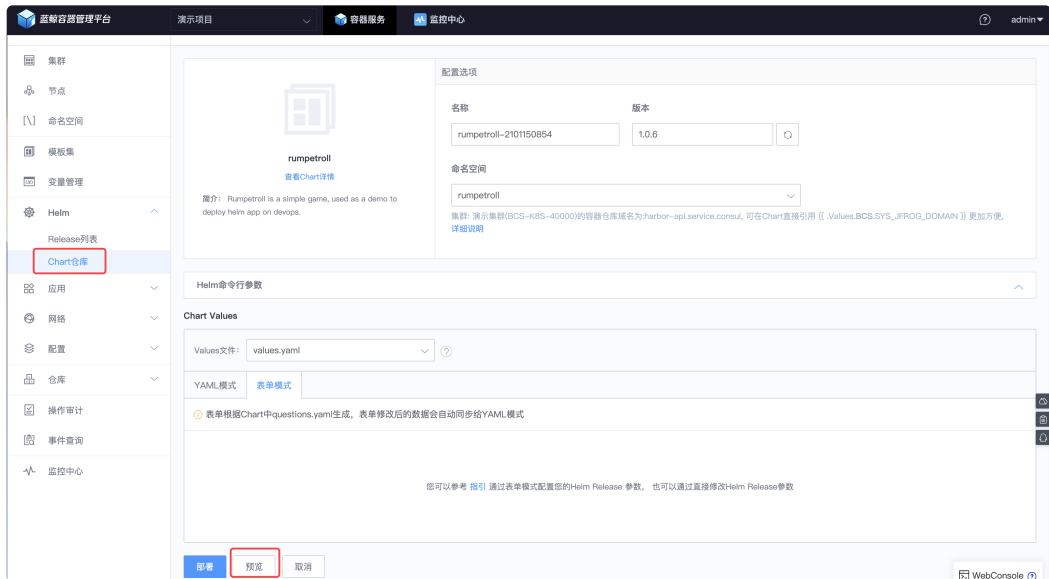


图 4. Helm Release 创建预览入口

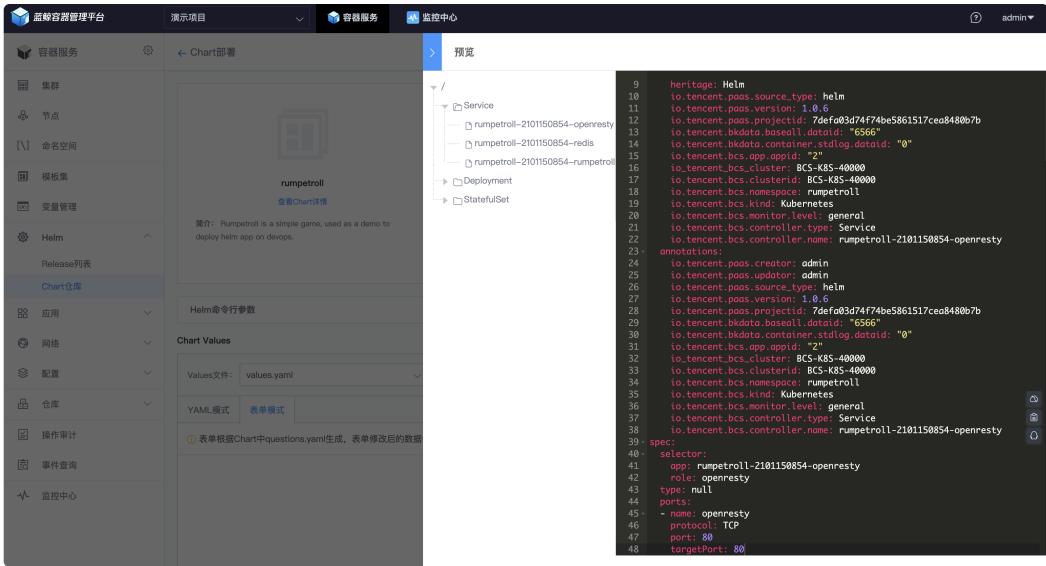


图 5. Helm Release 创建预览页面示例

- Helm Release 升级-对比查看功能

- 相比于创建，针对更新的场景，蓝鲸容器服务提供了对比的功能，可以确认变更内容之后，再执行更新

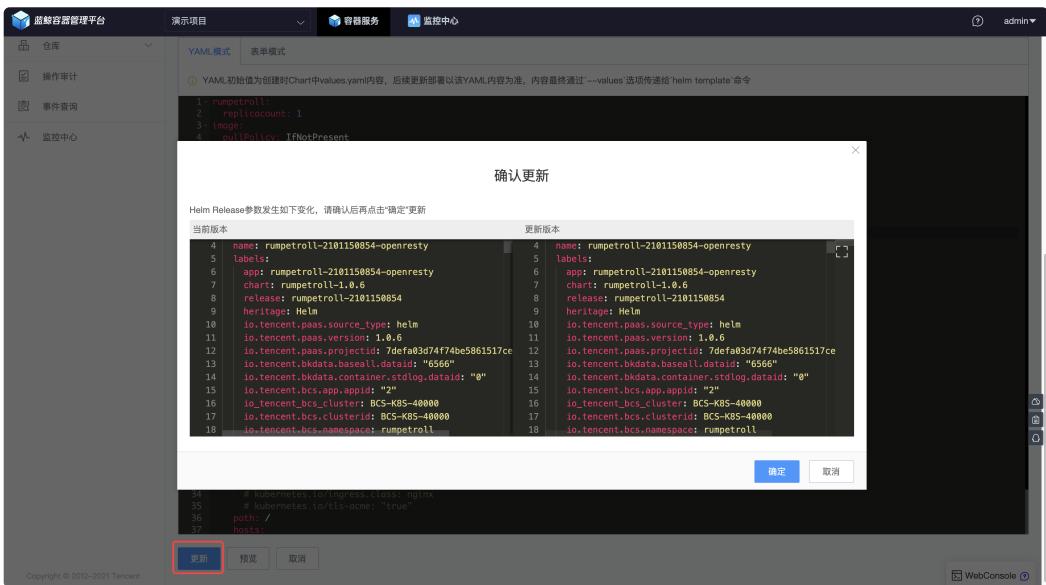


图 6. Helm Release 升级-对比查看功能

- 回滚

- 针对 Helm Release 的变更，容器服务还提供了回滚功能，在 Helm Release 列表中，选中需要回滚 Helm Release 的 **回滚** 按钮。即可以看到回滚页面，选中会要回滚的版本，确认回滚操作的变更内容即可执行回滚。

The screenshot shows the Helm Release list page in the BlueKing Container Management Platform. The left sidebar is collapsed. The main area displays a table with the following columns: 图标 (Icon), Release名称 (Release Name), 集群/命名空间 (Cluster/Namespace), 操作记录 (Operation Log), and 操作 (Operations). A single row is shown for 'rumpetroll-2101150854', which is associated with the '演示集群' (Demo Cluster) and 'rumpetroll' namespace. The release was created by 'admin' on '2021-01-15 21:05:29'. The '操作' (Operations) column contains buttons for '操作' (Operation), '查看状态' (View Status), '更新' (Update), and '删除' (Delete). The '查看状态' button is highlighted with a red rectangle.

图 7. Helm Release 回滚入口

检查 Helm Release 状态

- 通过 Helm Release 列表页面查看
 - 选中【查看状态】即可查看资源的状态信息。

This screenshot is identical to Figure 7, showing the Helm Release list page. The '查看状态' button in the operations column of the first release row is highlighted with a red rectangle, indicating the step to check the status.

图 8. Helm Release 状态查看入口

图 9. Helm Release 状态效果图

- 使用蓝鲸容器服务 WebConsole 功能
 - 使用蓝鲸容器服务的 WebConsole 功能，用户可以不用登录主机，直接使用 kubectl 命令查看业务容器的状态。

```

#####
# Welcome To BCS Console
#####
#Guide: https://bk.tencent.com/docs/markdown/产品白皮书/Function/web_console/Description.md
#支持常用Bash快捷键：Windows下Ctrl-w为关闭窗口快捷键，请使用Alt-w代替；使用Alt-Num切换Tab
#####
root:$ kubectl get deploy -n rumpetroll
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
rumpetroll-2101150854-openresty   1         1         1           1          15m
rumpetroll-2101150854-redis     1         1         1           1          15m
rumpetroll-2101150854-rumpetroll   1         1         1           1          15m
root:$ kubectl get svc -n rumpetroll
NAME          TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
rumpetroll-2101150854-openresty   ClusterIP  10.254.100.113 <none>        80/TCP     15m
rumpetroll-2101150854-redis     ClusterIP  10.254.240.231 <none>        6379/TCP   15m
rumpetroll-2101150854-rumpetroll   ClusterIP  None          <none>        80/TCP     15m
root:$

```

图 10. WebConsole 使用效果图

Helm Release 升级的版本

通过容器服务对 Helm Release 做升级时，有默认选中的版本(当前版本)。如下图所示，蓝鲸小游戏 rumpetroll 当前版本是 1.0.6，标记的是 `unchanged`，表示当前版本自上一次被当前 Helm Release 使用之后未发生过变化。

```

values.yaml
1: rumpetroll:
2:   replicas: 1
3:   image:
4:     pullPolicy: IfNotPresent
5:     repository: /public/bcs/k8s/pyrumpetroll
6:     tag: '0.3'

```

- 开源 Helm Chart

Helm 使用技巧

在 Chart 中使用 BCS 变量

BCS 提供的 Helm 变量

- 这些变量可以在 helm chart 中通过 `{{{ .Values.__BCS__.Key }}}` 的方式引用。

| Key | Value | 说明 | ----- | ----- | ----- | SYS_JFROG_DOMAIN | xxx | 仓库域名 | SYS_CLUSTER_ID | BCS-K8S-XXX | 集群 ID | SYS_PROJECT_ID | xxx | 项目 ID | SYS_CC_APP_ID | 1 | 业务 ID | SYS_NAMESPACE | default | 命名空间 |

如下两种使用方式支持包含子 Chart 的场景

方式 1：直接在 CHART 中使用

```
bash {{ default "127.0.0.1" $.Values.global.__BCS__.SYS_JFROG_DOMAIN }}
```

方式 2：通过模板的方式使用

```
```bash {{/ domain template /}} {{- define "bcsDomain" -}}{{- $default := "127.0.0.1" -}}{{- $values := .Values }} {{- if $values -}}
```

```

{{- $global := $values.global }}
{{- if $global -}}

{{- $bcs := $global.__BCS__ -}}
{{- if $bcs -}}

{{- if $bcs.SYS_JFROG_DOMAIN -}}
{{- $bcs.SYS_JFROG_DOMAIN -}}
{{- else -}}
{{- $default -}}
{{- end -}}

{{- else -}}
{{- $default -}}
{{- end -}}

{{- else -}}
{{- $default -}}
{{- end -}}

{{- else -}}
{{- $default -}}
{{- end -}}

{{- end -}} ```


```

{{- end -}}

在 Chart 中使用方式如下：

```
bash {{ template "bcsDomain" $ }}
```

### Helm Release 创建时表单与 `values.yaml` 参数说明

- 在创建 Helm Release 时，您可以通过填写表单或者直接编辑 `values.yaml` 来给 chart 传递参数。
- 表单是为了提升输入体验（规避错误输入）而引入的一种技术，它的值最终通过 `--set` 方式传递给 `helm template` 命令（string 类型的值通过 `--set-string` 传递给 `helm template`）。
- 页面编辑的 `values.yaml` 默认值是 Chart 中 `values.yaml` 文件内容，用于生成 Helm Release 时并不会替换 Chart 中的 `values.yaml` 文件，而是通过 `--values` 参数传递给 `helm template` 命令。
- 优先级问题
  - 为了消除您对参数优先级的困惑，我们通过一种双向同步的技术，将最新修改的输入源同步到另外一个输入源。比如，您在 form 表单中修改了镜像的 tag，当 form 表单失去焦点时，这个 tag 值会同步到 `values.yaml` 中，对应的，如果您接着编辑 `values.yaml` 的值，文本内容的变化也会自动反应到 form 表单中。

## \*如何编写 Helm `questions.yaml`

### Helm `questions.yaml` 是什么

Helm 是一个软件包管理器，提供了一种简单的方法来查找、共享和使用为 Kubernetes 而构建的软件。它提供 key-value 或者 `values.yaml` 用于设置 Helm 应用的实例化参数。

`questions.yaml` 是为了提高蓝鲸容器服务中 Helm 功能的易用性，参考开源产品 `Rancher` 提供的一种动态表单生成技术。

用户可在 Chart 中提供 `questions.yaml` 文件，在蓝鲸容器服务产品中实例化 Helm 应用的时候，蓝鲸容器服务会根据 `questions.yaml` 生成表单供用户输入实例化参数。

如下图所示，即为一个包含了 `questions.yaml` 的 Chart：

```
|- Chart.yaml
|- OWNERS
|- README.md
|- app-readme.md
|- charts
| |- mariadb-3.0.3.tgz
| +-- questions.yaml
|- requirements.lock
|- requirements.yaml
|- templates
| |- NOTES.txt
| |- _helpers.tpl
| |- deployment.yaml
| |- externaldb-secrets.yaml
| |- ingress.yaml
| |- pvc.yaml
| |- secrets.yaml
| |- svc.yaml
| |- tls-secrets.yaml
+-- values.yaml
```

## 可以干什么

为了说明 `questions.yaml` 可以提供什么样的动态表单，下面先展示一个 `questions.yaml` 内容段用于说明。

```
yaml - variable: persistence.enabled default: "false" description: "Enable persistent volume for WordPress"
type: boolean required: true label: WordPress Persistent Volume Enabled show_subquestion_if: true group:
```

```
"WordPress Settings" subquestions: - variable: persistence.size default: "10Gi" description: "WordPress Persistent Volume Size" type: string label: WordPress Volume Size - variable: persistence.storageClass default: "" description: "If undefined or null, uses the default StorageClass. Default to null" type: storageclass label: Default StorageClass for WordPress
```

上面这段配置在用户创建或者更新 helm 应用的时候一个如下表单，

### WordPress Persistent Volume Enabled \*

True  False

Enable persistent volume for WordPress

### WordPress Volume Size

10Gi

WordPress Persistent Volume Size

### Default StorageClass for WordPress

Use the default class...

If undefined or null, uses the default StorageClass. Default to null

该表单包含一个 radio 用于设置是否使用可持久化存储。如果用户选中使用可持久化存储, `storage class` 和 `volume size` 就可以用来供用户填写对应的存储类和卷大小。

## 使用场景

- 业务相对稳定之后, 把常规发布经常修改的参数项设置成 Form 表单
- 产品自助, 如果希望让产品自行发布, 表单是个不错的选择
- 防止错误输入, 为了尽量减少拼写错误, 可以为输入项设置校验规则

## 如何编写

```
| Variable | Type | Required | Description || ----- | ----- | --- |----- | variable | string | true | 申明该表单值对应于
values.yaml 中的字断, 及联字段使用 . 进行隔开, 比如 persistence.enabled .|| label | string | true | 表单项的标签 label .||
description | string | false | 关于变量的特别说明.| type | string | false | 表单值的字断类型, 默认是 string 类型 (当前支持的字断的类型为:
string, boolean, int, enum, password, storageclass and hostname).|| required | bool | false | 申明该字断是否是必填项 (真/假)||
default | string | false | 设置默认值.|| group | string | false | 输入项所属的组.|| min_length | int | false | 最小字符串长度.|| max_length | int |
false | 最大字符串长度.|| min | int | false | 最小整数长度.|| max | int | false | 最大整数长度.|| options | [string | false | 如果变量类型是
enum 类型, 该字断用于设置可选项, 比如选项:
- "ClusterIP"
- "NodePort"
- "LoadBalancer"|| valid_chars | string | false | 有效输入字符串校验.|| invalid_chars | string | false | 无效输入字符串的校验.|| subquestions
| []subquestion | false| 数组类型, 用户包含子问题.| show_if | string | false | 控制是否显示当前输入项, 比如 show_if:
"serviceType=Nodeport" || show_subquestion_if | string | false | 如果当前值为 true 或者可选项的值, 则该子问题会被显示出来. 比如
show_subquestion_if: "true"
```

**subquestions:** `subquestions[]` 除了不能包含 `subquestions` 或者 `show_subquestions_if` 字段, 上表中的其它字断均支持.

## 应用列表

容器服务应用列表分为两个视图展示：“集群模板”视图和“命名空间”视图。其中，集群模板视图展示由“表单模板集”页面实例化的应用，命名空间视图展示集群中所有命名空间下的应用，包含模板集、Helm、Client(kubectl、K8S API 等)方式部署的应用。通过应用页面可以方便查看应用详情、调度 Pod 以及对应用做滚动升级、扩缩容等操作。

### 应用类型

- Deployment
- DaemonSet
- Job
- StatefulSet

### 查询展示

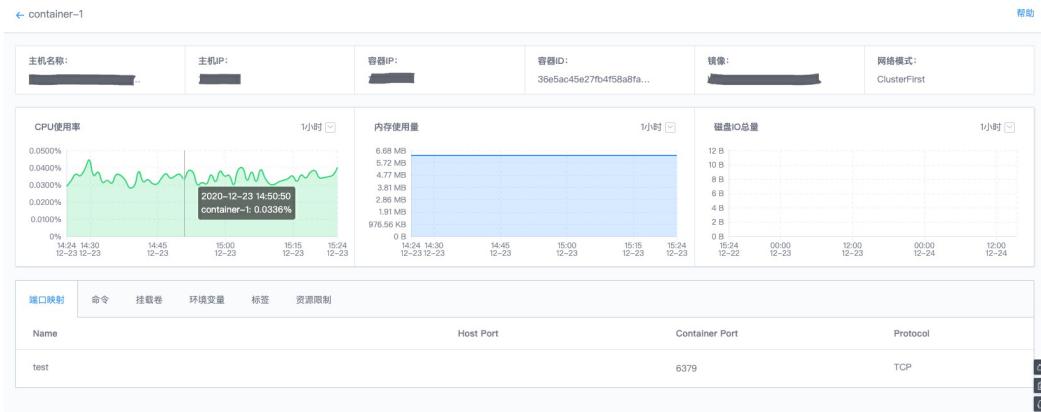
- 应用列表

The screenshot shows the 'Application List' interface. At the top, there's a search bar and a dropdown menu set to 'Cluster Template View'. Below the header, a message says '显示通过模板集表单模式实例化的应用详情'. The main area displays a single application entry under the heading 'test-template'. The entry includes the name 'test-deployment', status 'Running', type 'Deployment', namespace 'bcs-test', version 'v1', and 1 instance. There are buttons for 'Batch Upgrade', 'Horizontal Scaling', and 'More'. A help icon is in the top right corner.

- 应用详情

The screenshot shows the 'Application Detail' interface for 'test-deployment'. At the top, it shows basic info: application name 'test-deployment', creation time '2020-12-23 10:52:28', update time '2020-12-23 10:52:28', namespace 'bcs-test', and cluster '创建集群测试-01'. Below this are two line charts: 'CPU 使用率' and '内存使用量', both spanning from 10:53 to 11:39 on December 23. The CPU chart ranges from 0% to 0.0500%, and the memory chart ranges from 0 B to 6.68 MB. At the bottom, there's a 'Pod Management' section with tabs for '事件' (Events), '标签' (Labels), and '注解' (Annotations). It shows one pod entry: 'test-deployment-6fb9f56d8-r5t7m' with status 'Running', host IP '...', pod IP '...', and creation time '47分27秒'. A '重新调度' (Reschedule) button is also present.

- 容器详情



## 应用管理

- 滚动升级

选择【更新版本】，查看版本差异，点击【确定】滚动升级

应用列表 (通过Helm Chart, Client, 模板集YAML模式创建的应用, 请通过命名空间视图查询详情)

集群模板视图

显示通过概要表模式实例化的应用详情

test-template

test-deployment v2 Deployment [正在]

应用名称	状态	类型	命名空间	版本	实例数	最后更新时间	操作
test-deployment	Running	deployment	bcs-test	v1	1/1	12-23 10:52	<b>滚动升级</b> 扩容容 更多

应用列表 (通过Helm Chart, Client, 模板集YAML模式创建的应用, 请通过命名空间视图查询详情)

集群模板视图

test-deployment 滚动升级

当前版本: v1 实例数: 1

更新版本: v2 实例数: 2

```

1 kind: Deployment
2 metadata:
3 annotations: {}
4 labels: {}
5 name: test-deployment
6 namespace: bcs-test
7 spec:
8 minReadySeconds: 0
9 replicas: 1
10 selector:
11 matchLabels:
12 k8s-app: test-deployment
13 strategy:
14 rollingUpdate:
15 maxSurge: 0
16 maxUnavailable: 1
17 type: RollingUpdate
18 template:
19 metadata:

```

**确定** 取消

- 扩缩容

增加或缩实例数量

应用列表 通过Helm Chart, Client, 模板集YAML模式创建的应用, 请通过命名空间视图查询详情

集群模板视图 ▾

显示通过模板集表单模式实例化的应用详情

test-template 共包含 1 个应用模板

test-deployment (v2) Deployment 正常

操作	应用名称	状态	类型	命名空间	版本	实例数 (1)	最后更新时间
滚动升级 扩容 更多~	test-deployment	Running	deployment	bcs-test	v1	1/1	12-23 10:52

应用列表 通过Helm Chart, Client, 模板集YAML模式创建的应用, 请通过命名空间视图查询详情

集群模板视图 ▾

显示通过模板集表单模式实例化的应用详情

test-template 共包含 1 个应用模板

test-deployment (v2) Deployment 正常

test-deployment 扩容

操作	最后更新时间
滚动升级 扩容 更多~	12-23 10:52

实例数量

1

确定 取消

- 回滚上一版本

回滚到上一个版本

应用列表 通过Helm Chart, Client, 模板集YAML模式创建的应用, 请通过命名空间视图查询详情

集群模板视图 ▾

显示通过模板集表单模式实例化的应用详情

test-template 共包含 1 个应用模板

test-deployment (v2) Deployment 正常

操作	应用名称	状态	类型	命名空间	版本	实例数 (1)	最后更新时间
滚动升级 扩容 更多~ 回滚上一版本 重建 删除	test-deployment	Running	deployment	bcs-test	v2	2/2	12-23 10:50

应用列表 通过 Helm Chart, Client, 模板集 YAML 模式创建的应用, 请通过命名空间图标查看详情

集群模板视图 ▾

显示通过模板集单模式

test-template

test-deployment

应用名称

test-deployment

当前版本配置 上一版本配置

```

1 kind: Deployment
2 metadata:
3 annotations:
4 io.tencent.bcs.app.appid: '2'
5 io.tencent.bcs.cluster: BCS-KBS-40001
6 io.tencent.bcs.clusterid: BCS-KBS-40001
7 io.tencent.bcs.kind: Kubernetes
8 io.tencent.bcs.namespace: bcs-test
9 io.tencent.bkdata.baseall.dataid: '6566'
10 io.tencent.paaS.createTime: '2020-12-23 10:52:28'
11 io.tencent.paaS.creator: admin
12 io.tencent.paaS.instanceid: '7'
13 io.tencent.paaS.projectid: 42f1ad6d6ebf4e40a51bc
14 io.tencent.paaS.some_type: template
15 io.tencent.paaS.templateid: '7'
16 io.tencent.paaS.templatename: '7'
17 io.tencent.paaS.updateTime: '2020-12-23 10:52:28'
18 io.tencent.paaS.updator: admin
19 io.tencent.paaS.version: v1

```

```

1 kind: Deployment
2 metadata:
3 annotations:
4 io.tencent.bcs.app.appid: '2'
5 io.tencent.bcs.cluster: BCS-KBS-40001
6 io.tencent.bcs.clusterid: BCS-KBS-40001
7 io.tencent.bcs.kind: Kubernetes
8 io.tencent.bcs.namespace: bcs-test
9 io.tencent.bkdata.baseall.dataid: '6566'
10 io.tencent.paaS.createTime: '2020-12-23 10:52:28'
11 io.tencent.paaS.creator: admin
12 io.tencent.paaS.instanceid: '7'
13 io.tencent.paaS.projectid: 42f1ad6d6ebf4e40a51bc
14 io.tencent.paaS.some_type: template
15 io.tencent.paaS.templateid: '7'
16 io.tencent.paaS.templatename: '7'
17 io.tencent.paaS.updateTime: '2020-12-23 10:52:28'
18 io.tencent.paaS.updator: admin
19 io.tencent.paaS.version: v1

```

批量重建 批量删除 实例化

操作

滚动升级 扩容 容 更多 ▾

回滚上一版本

确定 取消

- 重建应用

重新调度应用下的 Pod

应用列表 通过 Helm Chart, Client, 模板集 YAML 模式创建的应用, 请通过命名空间图标查看详情

集群模板视图 ▾

显示通过模板集单模式实例化的应用详情

test-template

共包含 1 个应用模板

test-deployment v2 Deployment 正常

应用名称	状态	类型	命名空间	版本	实例数	最后更新时间	操作
test-deployment	Running	deployment	bcs-test	v2	2/2	12-23 15:50	滚动升级 扩容 容 更多 ▾

回滚上一版本

**重建**

删除

应用列表 通过 Helm Chart, Client, 模板集 YAML 模式创建的应用, 请通过命名空间图标查看详情

集群模板视图 ▾

显示通过模板集单模式实例化的应用详情

test-template

test-deployment v2 Deployment

应用名称

test-deployment

确定重建【test-deployment】?

重建是指应用关联pod的重新调度

```

1 kind: Deployment
2 metadata:
3 annotations: {}
4 labels: {}
5 name: test-deployment
6 namespace: bcs-test
7 spec:
8 minReadySeconds: 0
9 replicas: 2
10 selector:
11 matchLabels:
12 k8s-app: test-deployment
13 strategy:
14 rollingUpdate:
15 maxSurge: 0
16 maxUnavailable: 1
17 type: RollingUpdate
18 template:
19 metadata:
20 labels:
21 k8s-app: test-deployment
22 spec:
23 containers:
24 - env:

```

批量重建 批量删除 实例化

操作

滚动升级 扩容 容 更多 ▾

确定 取消

- 删除应用

应用列表 通过 Helm Chart、Client、模板集 YAML 样式创建的应用，请通过命名空间视图查询详情。 [帮助](#)

集群模板视图 [显示通过模板集表单模式实例化的应用详情](#)

**test-template** 共包含 1 个应用模板 [^](#)

**test-deployment** [v2] Deployment 正常 [操作](#) [批量重建](#) [批量删除](#) [实例化](#)

应用名称	状态	类型	命名空间	版本	实例数	最后更新时间	操作
test-deployment	Running	deployment	bcs-test	v2	2/2	12-23 15:50	<a href="#">滚动升级</a> <a href="#">扩缩容</a> <a href="#">更多</a>

[回滚上一版本](#) [重建](#) [删除](#) [回滚](#) [扩容](#) [缩容](#)

• Pod 重新调度

[test-deployment](#) [to YAML](#)

应用名称: test-deployment	创建时间: 2020-12-23 10:52:28	更新时间: 2020-12-23 16:11:16	所在命名空间: bcs-test	所属集群: 新测试集群
-----------------------	---------------------------	---------------------------	------------------	-------------

POD视图 容器视图

CPU使用率 内存使用量 网络

Pod管理 事件 标签 注解

状态: Running	Host IP: <a href="#">[REDACTED]</a>	Pod IP: <a href="#">[REDACTED]</a>	存活时间: 20分23秒	<a href="#">重新调度</a>
-------------	-------------------------------------	------------------------------------	--------------	----------------------

## Game workload 使用指南

### 功能介绍

GameDeployment 是针对游戏场景定制的面向无状态服务的增强版 Deployment

GameStatefulSet 是针对游戏场景定制的面向有状态服务的增强版 StatefulSet

### GameDeployment

- 支持 Operator 高可用部署
- 支持滚动更新 / 原地更新
- 优雅删除和更新
- 支持设置 partition 灰度发布
- 支持分步骤自动化灰度发布
- 支持 HPA
- 指定 pod 删除
- 支持 pod 注入唯一序号
- 支持防误删功能
- 支持 readinessgate 可选功能

### GameStatefulSet

- 支持 Operator 高可用部署
- 支持 Node 失联时，Pod 的自动漂移
- 支持优雅删除和更新
- 支持滚动更新/原地更新
- 支持分步骤自动化灰度发布
- 支持并行更新
- 支持 maxSurge / maxUnavailable 字段
- 支持 HPA
- 支持防误删功能

## 组件安装

图标	组件名称	版本	状态	描述	操作
	GameDeployment-Operator	...	未启用	GameDeployment-Operator 支持优雅删除/更新，原地更新及智能化分步骤灰度发布功能	<span style="border: 1px solid red; padding: 2px;">启用</span>
	GameStatefulSet-Operator	...	未启用	GameStatefulSet-Operator 支持优雅删除/更新，原地更新及智能化的分步骤灰度发布功能	<span style="border: 1px solid red; padding: 2px;">启用</span>
	Hook-Operator	...	未启用	Hook-Operator 支持优雅删除/更新的 hook 检查	<span style="border: 1px solid red; padding: 2px;">启用</span>
	General-Pod-Autoscaler	...	未启用	General-Pod-Autoscaler 支持更多模式自动扩缩副本数	<span style="border: 1px solid red; padding: 2px;">启用</span>
	Prometheus-Adapter	...	未启用	Prometheus-Adapter 自定义指标转换所需	<span style="border: 1px solid red; padding: 2px;">启用</span>
	Cluster-Autoscaler	...	未启用	Cluster-Autoscaler 支持集群自动扩缩	<span style="border: 1px solid red; padding: 2px;">启用</span>
	GameIngress-Controller	1.27.0-alpha.22	已禁用	蓝鲸容器服务扩展 Ingress，多云环境下为 GameServer 提供灵活的流量接入方式	<span style="border: 1px solid red; padding: 2px;">操作</span>

图标	组件名称	版本	状态	描述	操作
	GameDeployment-Operator	1.26.0-alpha.6	已启用	GameDeployment-Operator 支持优雅删除/更新，原地更新及智能化分步骤灰度发布功能	<span style="border: 1px solid red; padding: 2px;">操作</span>
	GameStatefulSet-Operator	1.26.0-alpha.6	已启用	GameStatefulSet-Operator 支持优雅删除/更新，原地更新及智能化的分步骤灰度发布功能	<span style="border: 1px solid red; padding: 2px;">操作</span>
	Hook-Operator	1.26.0-alpha.6	已启用	Hook-Operator 支持优雅删除/更新的 hook 检查	<span style="border: 1px solid red; padding: 2px;">操作</span>

## 快速体验

### GameDeployment

#### 创建实例

```
将以下保存为 gamedeployment-inplaceupdate.yaml，使用 kubectl apply -f gamedeployment-inplaceupdate.yaml ```yaml
apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test-hooktemplate labels: io.tencent.bcs.dev/deletion-allow: Always spec: metrics: - name: webtest count: 10 interval: 5s successfulLimit: 1 successCondition: "asInt(result) == 1" provider: web: url: http://{{ args.PodIP }}:8080/hook jsonPath: "{$.result}" timeoutSeconds: 60
- name: webtest2 count: 10 interval: 2s successfulLimit: 1 successCondition: "asInt(result) == 1" provider: web: url: http://{{ args.PodIP }}:8080/hook jsonPath: "{$.result}" timeoutSeconds: 60
```
```

```
apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test-post labels: io.tencent.bcs.dev/deletion-allow: Always spec: metrics: - name: patch-test provider: kubernetes: function: patch fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-
```

```
deleting value: "false" successfulLimit: 1 interval: 10s - name: get-test provider: kubernetes: function: get fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-deleting value: "false" successfulLimit: 1 interval: 10s
```

```
apiVersion: tkex.tencent.com/v1alpha1 kind: GameDeployment metadata: name: test-gamedeploy labels: app: test-gamedeploy io.tencent.bcs.dev/deletion-allow: Always annotations: tkex.bkbc.tencent.com/gamedeployment-index-on: "true" tkex.bkbc.tencent.com/gamedeployment-index-range: '{"podStartIndex": 100, "podEndIndex": 110}' spec: updateStrategy: type: InplaceUpdate maxSurge: 0 maxUnavailable: 25% partition: 0 replicas: 2 selector: matchLabels: app: test-gamedeploy preDeleteUpdateStrategy: hook: templateName: test-hooktemplate preInplaceUpdateStrategy: hook: templateName: test-hooktemplate postInplaceUpdateStrategy: hook: templateName: test-post template: metadata: labels: app: test-gamedeploy annotations: io.tencent.bcs.dev/pod-cost-metrics-name: cpu;memory io.tencent.bcs.dev/pod-cost-metrics-weight: 2;0.001 io.tencent.bcs.dev/pod-deletion-cost-sort-method: ascend spec: containers: - name: web image: hub.bktencent.com/blueking/game-demo:0.0.1 imagePullPolicy: IfNotPresent 创建完后预期结果如下: bash
```

kubectl get gamedeployments.tkex.tencent.com

```
NAME DESIRED UPDATED UPDATED_READY READY TOTAL AGE test-gamedeploy 2 2 2 2 2 3m27s
```

kubectl get hooktemplates.tkex.tencent.com

```
NAME AGE test-hooktemplate 12m test-post 12m ``
```

原地更新

将 gamedeployment-inplaceupdate.yaml 中， gamedeplyoment 的 image 改为： image: hub.bktencent.com/blueking/game-demo:0.0.2，再执行 `kubectl apply -f gamedeployment-inplaceupdate.yaml`

预期结果如下： ``bash

kubectl get hookrun

```
NAME PHASE AGE preinplace-test-gamedeploy-75c467595-wt9g5-test-hooktemplate Running 7s `` 等一段时间后， preinplace hookrun 运行完成，开始运行 post inplace hookrun (hookrun 跑完会自动被删除，所以有可能来不及看到 hookrun 的存在，重点看 Pod 有没有被更新)
```

``bash

kubectl get hookrun

```
NAME PHASE AGE postinplace-test-gamedeploy-5ccfb5c6c4-b5w28-test-post Running 3s ``
```

更新完成后，对应 pod 的 RESTAR 数加一 ``bash

kubectl get pod

```
NAME READY STATUS RESTARTS AGE test-gamedeploy-b5w28 1/1 Running 1 102s test-gamedeploy-wt9g5 1/1 Running 1 102s ``
```

滚动更新

把以下内容保存在文件 gamedeployment-rollingupdate.yaml 中：

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameDeployment metadata: name: test-gamedeploy labels: app: test-gamedeploy io.tencent.bcs.dev/deletion-allow: Always annotations: tkex.bkbc.tencent.com/gamedeployment-index-on: "true" tkex.bkbc.tencent.com/gamedeployment-index-range: '{"podStartIndex": 100, "podEndIndex": 110}' spec: updateStrategy: type: RollingUpdate replicas: 2 selector: matchLabels: app: test-gamedeploy preDeleteUpdateStrategy: hook: templateName: test-hooktemplate preInplaceUpdateStrategy: hook: templateName: test-post template: metadata: labels: app: test-gamedeploy annotations: io.tencent.bcs.dev/pod-cost-metrics-name: cpu;memory io.tencent.bcs.dev/pod-cost-metrics-weight: 2;0.001 io.tencent.bcs.dev/pod-deletion-cost-sort-method: ascend spec: containers: - name: web image: hub.bktencent.com/blueking/game-demo:0.0.1 imagePullPolicy: IfNotPresent 执行 kubectl apply -f gamedeployment-rollingupdate.yaml , 预期结果如下： pre delete hookrun 正在运行
```

```
```bash
```

## kubectl get hookrun

```
NAME PHASE AGE predelete-test-gamedeploy-5ccfb5c6c4-b5w28-test-hooktemplate Running 6s 等一段时间后, pod 更新为新的版本
bash
```

## kubectl get po

```
NAME READY STATUS RESTARTS AGE test-gamedeploy-ckfkr 1/1 Running 0 38s test-gamedeploy-rw74v 1/1 Running 0 56s ...
```

### 删除实例

```
执行 kubectl delete -f gamedeployment-inplaceupdate.yaml 删除实例 ```bash
```

## kubectl delete -f gamedeployment-inplaceupdate.yaml

```
hooktemplate.tkex.tencent.com "test-hooktemplate" deleted hooktemplate.tkex.tencent.com "test-post" deleted
gamedeployment.tkex.tencent.com "test-gamedeploy" deleted ...
```

### GameStatefulSet

#### 创建实例

```
将以下保存为 gamestatefulset-inplaceupdate.yaml, 使用 kubectl apply -f gamestatefulset-inplaceupdate.yaml 创建
```

```
```yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test-hooktemplate labels: io.tencent.bcs.dev/deletion-allow: Always spec: metrics: - name: webtest count: 10 interval: 5s successfulLimit: 1 successCondition: "asInt(result) == 1" provider: web: url: http://{{ args.PodIP }}:8080/hook jsonPath: "{$.result}" timeoutSeconds: 60  
- name: webtest2 count: 10 interval: 2s successfulLimit: 1 successCondition: "asInt(result) == 1" provider: web: url: http://{{ args.PodIP }}:8080/hook jsonPath: "{$.result}" timeoutSeconds: 60
```

```
apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test-post labels: io.tencent.bcs.dev/deletion-allow: Always spec: metrics: - name: patch-test provider: kubernetes: function: patch fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-deleting value: "false" successfulLimit: 1 interval: 10s - name: get-test provider: kubernetes: function: get fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-deleting value: "false" successfulLimit: 1 interval: 10s
```

```
apiVersion: tkex.tencent.com/v1alpha1 kind: GameStatefulSet metadata: name: test-gameset labels: app: test-gameset io.tencent.bcs.dev/deletion-allow: Always spec: serviceName: test-svc updateStrategy: type: InplaceUpdate replicas: 2 selector: matchLabels: app: test-gameset preDeleteUpdateStrategy: hook: templateName: test-hooktemplate preInplaceUpdateStrategy: hook: templateName: test-hooktemplate postInplaceUpdateStrategy: hook: templateName: test-post template: metadata: labels: app: test-gameset spec: containers: - name: web image: hub.bktencent.com/blueking/game-demo:0.0.1 imagePullPolicy: IfNotPresent ... 预期结果如下:
```

```
```bash
```

## kubectl get gamestatefulsets.tkex.tencent.com

```
NAME REPLICAS READY_REPLICAS CURRENT_REPLICAS UPDATED_REPLICAS UPDATED_READY_REPLICAS AGE test-gameset 2 2 2 2 2 61s
```

## kubectl get hooktemplates.tkex.tencent.com

```
NAME AGE test-hooktemplate 109s test-post 109s ...
```

### 原地更新

```
将 gamestatefulset-inplaceupdate.yaml 中, gamestatefulset 的 image 改为: image: hub.bktencent.com/blueking/game-demo:0.0.2, 再执行 kubectl apply -f gamestatefulset-inplaceupdate.yaml
```

预期结果如下： preinplace hookrun 正在运行

```
kubectl get hookrun NAME PHASE AGE preinplace-test-gameset-777f889cc-1-test-hooktemplate Running 3s
```

更新完成后，对应 pod 的 RESTARS 数加一

...

## kubectl get pod

```
NAME READY STATUS RESTARTS AGE test-gameset-0 1/1 Running 1 3m58s test-gameset-1 1/1 Running 1 3m55s ...
```

### 滚动更新

将以下内容保存到文件 gamestatefulset-rollingupdate.yaml 中：

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameStatefulSet metadata: name: test-gameset labels: app: test-gameset io.tencent.bcs.dev/deletion-allow: Always spec: serviceName: test-svc updateStrategy: type: RollingUpdate replicas: 2 selector: matchLabels: app: test-gameset preDeleteUpdateStrategy: hook: templateName: test-hooktemplate preInplaceUpdateStrategy: hook: templateName: test-hooktemplate postInplaceUpdateStrategy: hook: templateName: test-post template: metadata: labels: app: test-gameset spec: containers: - name: web image: hub.bktencent.com/blueking/game-demo:0.0.1 执行 kubectl apply -f gamestatefulset-rollingupdate.yaml
```

预期结果：

```
predelete hookrun 正在运行
```

```bash

kubectl get hookrun

```
NAME PHASE AGE predelete-test-gameset-c594b9cb8-1-test-hooktemplate Running 3s ... 一段时间后，更新完成，对应 pod 被重建为新版本 pod
```

```bash

## \$ kubectl get pod

```
NAME READY STATUS RESTARTS AGE test-gameset-0 0/1 ContainerCreating 0 2s test-gameset-1 1/1 Running 0 27s ...
```

### 删除实例

```
```bash kubectl delete -f gamestatefulset-inplaceupdate.yaml
```

```
hooktemplate.tkex.tencent.com "test-hooktemplate" deleted hooktemplate.tkex.tencent.com "test-post" deleted gamestatefulset.tkex.tencent.com "test-gameset" deleted ...
```

Game workload Hook

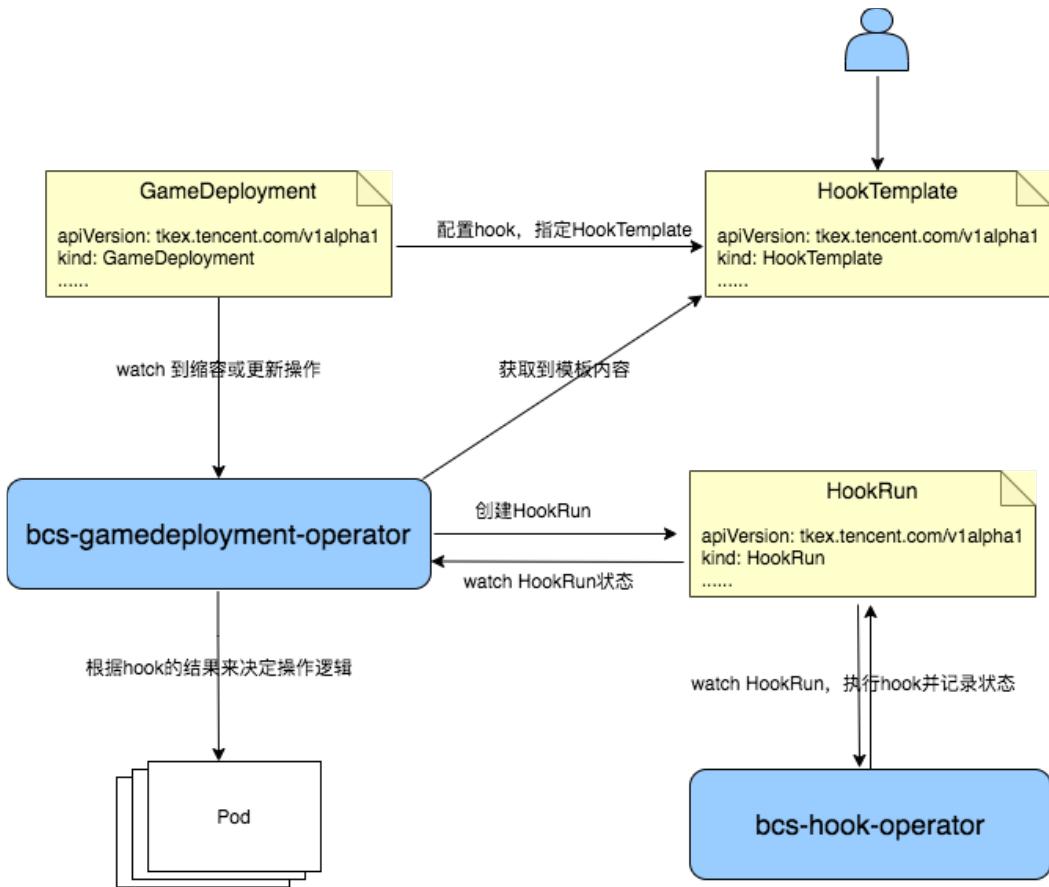
概述

多数复杂类应用在发布更新过程中有许多外部依赖或应用本身的数据指标依赖，如实例扩缩容或更新前需要进行数据搬迁；缩容一个实例前需要先完成路由变更；实例缩容或更新前需要等待游戏对局结束此外，在灰度发布时，有时我们需要从 prometheus 监控数据中查看指标是否符合预期，以决定是否继续灰度更多的实例

这其实可以看作为应用发布过程中的各种 hook 勾子，通过 hook 的结果来判断是否可以继续下一步的发布流程无论是面向无状态应用的 GameDeployment 还是面向有状态应用的 GameStatefulSet，都有这种发布需求

bk-bcs 在 kubernetes 层面抽象出了一个通用的 operator: bcs-hook-operator bcs-hook-operator 主要职责是根据 hook 模板执行 hook 操作并记录 hook 的状态，GameDeployment 或 GameStatefulSet watch hook 的最终状态，根据 hook 结果来决定下一步执行何种操作

GameDeployment/GameStatefulSet 与 bcs-hook-operator 在应用发布过程中使用 hook 时的交互架构图：



hooktemplate

HookTemplate 用来定义一个 hook 的模板。在一个 HookTemplate 中可以定义多个 metric，每个 metric 都是需要执行的一个 hook。在 metric 中可以定义 hook 的次数、两次之间的间隔、成功的条件、provider 等等。provider 定义的是 hook 的类型，目前支持三种类型的 hook： webhook、prometheus 和 K8S 类型。

WEB 类型

```
yaml
apiVersion: tkex.tencent.com/v1alpha1
kind: HookTemplate
metadata:
  name: test-hooktemplate
  namespace: test
spec:
  args:
    - name: abc
      value: "123"
  metrics:
    - name: webtest
      count: 10
      interval: 3s
      successfulLimit: 1
      successCondition: "asInt(result) == 1"
      provider:
        web:
          url: http://{{ args.PodIP }}:8080/hook?podnamespace={{ args.PodNamespace }}&podname={{ args.PodName }}&podcontainer={{ args.PodContainer[0] }}&podip={{ args.PodIP }}&abc={{ args.abc }}
      jsonPath: "${.result}"
      timeoutSeconds: 60

```

PROMETHEUS 类型

成功/失败条件的设置需要注意查询语句返回的类型。asInt 函数只适用于类型为 string 的变量，以下为返回类型是 string 的示例。

```
yaml
apiVersion: tkex.tencent.com/v1alpha1
kind: HookTemplate
metadata:
  namespace: test
  name: test-hooktemplate
spec:
  metrics:
    - name: promtest
      count: 10
      interval: 1s
      consecutiveErrorLimit: 10
      failureCondition: "asInt(result) == 0"
      provider:
        prometheus:
          query:
            prometheus_operator_watch_operations_failed_total{controller="alertmanager"}
          address: http://po-prometheus-operator-prometheus.thanos.svc.cluster.local:9090
      如果查询语句返回类型是 vector，则不能使用 "==" 等判断，而要用下标进行取值，并且如果 vector 中的数值类型不是 string，则不能使用 asInt 函数，否则会报错；以下为返回类型是 vector，且 vector 内数值类型为 float64 的示例

```

```
successCondition: "result[0] == 0"
```

如果查询语句返回类型是 vector，且 vector 中元素为数值，若只需判断某个数值是否在数组内，则可以如下设置成功/失败条件

```
successCondition: "0 in result"
```

如果返回类型是 scalar，则不能使用 asInt 等类型转换，可以直接进行比较

```
successCondition: "result == 0"
```

查询语句的返回值类型一方面可以从暴露指标的代码中确定，另一方面也可以通过查询语句的函数进行转换，如在查询语句中加上 `vector()` 或 `scalar()` 函数可以将返回值转换成 vector 或 scalar 类型，具体见：
<https://prometheus.io/docs/prometheus/latest/querying/functions/>

K8S 类型

1. 功能：支持对特定资源对象特定字段的 get 和 patch 操作；
2. 资源对象：
 - 可在 hooktemplate 的 args 字段中增加 Group, Version, Kind 变量，指定特定资源对象，默认为 Pod 对象；
 - 资源对象指定 GVK 只支持 GameDeployment 和 GameStatefulSet 两种，如需支持更多对象种类，需要给 hook-operator 的 RBAC 增加权限，请联系相关同学操作；
 - 资源对象名称和命名空间强制为 hookrun 的 args 字段中的默认 PodName 和 PodNamespace（防止操作其他人资源）；
3. 特定字段 fields：
 - 支持多个字段，每个字段含 path 和 value 两个值
 - path 只支持 "metadata.annotations.xxx" 形式，即只能对 annotations 字段进行操作；
 - value 在 patch 操作时指的是写到资源对象对应字段的值，在 get 操作时指的是想要得到的预期值；
 - 将要操作的对象需已存在 annotations 字段，因为 patch 操作不支持递归创建；
4. 操作 function：
 - 只支持 get 和 patch；
 - get 操作对比实际拿到的值和预期值 value，不一致则失败，存在多个字段时有一个不一致就会失败；
 - patch 操作将 value 写到对象对应的 path 处，操作过程中出错则失败，多个字段时有一个失败则整体失败；
 - k8s metric 无需指定 successfulCondition，需要指定 successfulLimit

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test-hooktemplate namespace: test spec: metrics: - name: patch-test provider: kubernetes: function: patch fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-deleting value: "false" successfulLimit: 1 interval: 10s - name: get-test provider: kubernetes: function: get fields: - path: metadata.annotations.io.tencent.bcs.dev/game-pod-deleting value: "false" successfulLimit: 1 interval: 10s
```

k8s 类型的 hook 也可用于原地更新前后切断和恢复 CLB 流量如在更新前执行如下 preinplace hook，将 clb-weight 置为 0，clb-weight-ready 置为 false，然后等 bcs-ingress-controller 进行相关操作，等切断 CLB 流量后，bcs-ingress-controller 将 clb-weight-ready 置为 true，此时 hook 成功，可以开始进行原地更新操作（preinplace hook 在 GameWorkload 的 spec.preInplaceUpdateStrategy 字段设置）

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test namespace: default spec: metrics: - name: patch-test provider: kubernetes: function: patch fields: - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight value: "0" - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight-ready value: "false" successfulLimit: 1 interval: 10s - name: get-test provider: kubernetes: function: get fields: - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight value: "0" - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight-ready value: "true" successfulLimit: 1 interval: 10s
```

在更新成功之后，执行如下 postinplace hook，将 clb-weight 置为 1，clb-weight-ready 置为 false，然后等 bcs-ingress-controller 进行相关操作，等恢复 CLB 流量后，bcs-ingress-controller 将 clb-weight-ready 置为 true，此时 hook 成功，pod 开始接受流量（postinplace hook 在 GameWorkload 的 spec.postInplaceUpdateStrategy 字段设置）

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: name: test namespace: default spec: metrics: - name: patch-test provider: kubernetes: function: patch fields: - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight value: "1" - path: metadata.annotations.networkextension.bkbc.tencent.com/clb-weight-ready value: "false" successfulLimit: 1
```

```
interval: 10s - name: get-test provider: kubernetes: function: get fields: - path:  
metadata.annotations.networkextension.bkbc.tencent.com/clb-weight value: "1" - path:  
metadata.annotations.networkextension.bkbc.tencent.com/clb-weight-ready value: "true" successfulLimit: 1  
interval: 10s
```

参数解读

1. args

配置 hook 需要的参数 bcs 默认提供了以上4种参数，可只配置 key, value 则由创建 HookRun 时自动传入， preDeleteUpdateStrategy/preInplaceUpdateStrategy 情况下，默认支持以上四种参数 PodContainer 参数按 spec 中定义的容器顺序，以 PodContainer[0], PodContainer[1], ... 形式存储每个容器的名字信息 如果是 preInplace hookrun, 还会有默认参数 ModifiedContainer, 以 ModifiedContainer[0], ModifiedContainer[1], ... 的形式存储触发原地更新的容器名字（也即镜像发生改变的容器名字） args 中预填的 PodIP、PodName、PodNamespace、PodContainer 因为和 Pod 状态相关，所以只适用于 preDeleteUpdateStrategy/preInplaceUpdateStrategy/postInplaceUpdateStrategy 的 Hook 而 canary 中的 hook 面对的是整个应用，所以 args 字段 不适用于 canary.hook

2. metrics

在模板的 metrics 中可以定义多个 metric, 每个 metric 可以配置一个 hook 调用, metric 中的 provider 定义了该 hook 的类型

1. count

count 表示该 metric 需要完成的 hook 调用的次数如果 count 和 interval 都没配置, 将只进行一次 hook 调用; 如果只配置了 interval, 将会无限制地一直进行 hook 调用, 直到成功或者失败条件满足即退出; 如果配置 count 且大于 1, 必须配置 interval

1. interval

interval 表示两次 hook 调用间的时间间隔, 如果没有定义, 将只会进行一次 hook 调用

1. successCondition

successCondition 表示 hook 的成功条件

1. failureCondition

failureCondition 表示 hook 的失败条件

1. failureLimit

failureLimit 定义了标识 metric 失败的次数, 只要执行 metrics 失败次数 > n, 那这个 metric 就是失败的 如果配置了 count, 且 failureLimit 的值 小于 count, 执行该 metrics 错误次数为 failureLimit 时, 此 metrics 标记为失败; 如果配置了 count, 执行该 metric count 次后仍失败, 则约定检测该 metrics 达到阈值 仍失败, 标记此 metric 成功, 即可强制删除

1. consecutiveErrorLimit

consecutiveErrorLimit 定义了允许的 hook 连续产生 error 的次数, 如设置为 4, 则只要有连续4次检测失败, 那这个 metric 就是失败的

1. successfulLimit

successfulLimit 定义了标识 metric 成功的次数, 即只要有n次返回是成功的, 那这个 metric 就是成功的

1. consecutiveSuccessfulLimit

consecutiveSuccessfulLimit 定义了达到metrics连续执行成功的次数时, 标识此metric状态是成功的

2. provider

hook 的类型, 目前仅支持 webhook, prometheus 和 k8s

以 webhook 类型为例, url 定义了 webhook 调用的地址, jsonPath 表示提取返回 json 中的某个字段, timeoutSeconds 指定超时时间 (默认为 10s)

url 中可以通过模板的形式配置, 比如 `http://{{ args.PodIP }}:9091`, hookrun-controller 在进行 hook 调用时会通过 args 渲染出真实值

注意 `args.PodIP` 等参数与双括号之间有空格 当前 webhook 类型只支持 GET 方式 注: 如果使用集群内 prometheus 作为

provider, address填写为 <http://po-prometheus-operator-prometheus.thanos.svc.cluster.local:9090> 即可

其它参数

1. 防误删功能 在 K8S 机制中, 如果删除 CRD, 会导致所有 CR 一起被级联删除, 因此如果手误删除了 HookTemplate 的 CRD (比如误卸载组件), 会导致集群中所有 HookTemplate 实例一起被删除, 造成严重影响, 为了防止误操作造成的影响, 我们加了一个 Admission Webhook, 对 HookTemplate 的 CRD 和 CR 的删除操作做验证, 根据 CRD 和 CR 的 label 中对应标签值的不同, 进行相应操作 在 HookTemplate 的 CRD 中标签效果如下 (默认为空, 也即是不允许删除)

key value 效果	----- ----- -----
io.tencent.bcs.dev/deletion-allow Always 总是允许删除	io.tencent.bcs.dev/deletion-allow Cascading 如果有级联资源则不允许删除, 如删除 HookTemplate 的 CRD 时有 HookTemplate 实例存在 不添加labels 不添加labels 不允许删除

2. 执行策略

同一 HookTemplate 内不同 metric 的执行顺序可以通过 Policy 字段指定, 当前支持两种策略, 默认为 Parallel 策略

- Parallel: 在执行时会并行执行所有 metrics
- Ordered: 执行时顺序执行所有 metrics, 只有上一 metric 执行结束后才会继续执行下一 metric

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookTemplate metadata: ... spec: policy: Parallel
```

HookRun

HookRun 是根据模板 HookTemplate 创建的一个实际运行的 hook CRD, bcs-hook-operator 监测并控制 HookRun 的运行状态和生命周期, 根据其 metrics 中的定义来执行 hook 操作, 并实时记录 hook 调用的结果

HOOKRUN示例

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookRun metadata: name: test-gamedeployment-67864c6f65-4-test namespace: default spec: metrics: - name: webtest provider: web: jsonPath: '${.age}' url: http://1.1.1.1:9091 successCondition: asInt(result) < 30 terminate: true status: metricResults: - count: 1 failed: 1 measurements: - finishedAt: "2020-11-09T10:08:49Z" phase: Failed startedAt: "2020-11-09T10:08:49Z" value: "32" name: webtest phase: Failed phase: Failed startedAt: "2020-11-09T10:08:49Z"
```

HOOKRUN 参数解读

一般用户无需设置 HookRun 相关参数, 由控制器根据 HookTemplate 自动生成对应的 HookRun 但有一种情况是需要手动设置 hookrun 参数的: 当 hookrun 正在执行, 但此时想要中断 hookrun 执行, 不能直接手动删除 hookrun, 因为控制器会自动重建, 此时应该手动修改 hookrun 的 spec.terminate 字段, 将其设置为 true, 则 hookrun 会中断执行

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: HookRun metadata: ... spec: terminate: true
```

GameDeployment特性说明

CRD定义样例

GameDeployment 是 kubernetes 原生 Deployment 的增强实现, 一个典型的 GameDeployment 定义如下:

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameDeployment metadata: name: test-gamedeployment namespace: test labels: app: test-gamedeployment spec: replicas: 5 selector: matchLabels: app: test-gamedeployment scaleStrategy: podsToDelete: - test-gamedeployment-ghrwg minReadySeconds: 5 updateStrategy: type: InplaceUpdate partition: 0 maxUnavailable: 1 maxSurge: 0 paused: false inPlaceUpdateStrategy: gracePeriodSeconds: 30 preDeleteUpdateStrategy: hook: templateName: test preInplaceUpdateStrategy: hook: templateName: test-inplace template: metadata: labels: app: test-gamedeployment spec: containers: - name: nginx image: nginx:1.7.9 imagePullPolicy: IfNotPresent resources: requests: cpu: 20m limits: cpu: 20m
```

CR 字段说明

GameDeployment 详细字段定义及其意义, 请查阅 API 文档: http://htmlpreview.github.io/?https://github.com/Tencent/bk-bcs/blob/master/docs/features/bcs-gamedeployment-operator/API_Reference.html

滚动更新 / 原地更新

涉及字段: updateStrategy updateStrategy.type 支持 RollingUpdate, InplaceUpdate 2种更新方式, 相比原生 Deployment, 新增 InplaceUpdate 更新模式 滚动更新, 为 k8s 原生支持更新策略, 更新时删除旧版本 Pod, 建立新版本 Pod 由于建立 Pod 需要进行调度流

程，因此开销较大，速度较慢，另外共享内存等信息会丢失 原地更新，指更新时只重启 Pod 中的容器，其他保持不变由于无需进行调度过程，速度较快，并且共享内存等信息会被保持下来

注意：InplaceUpdate，只支持以原地重启容器的方式更新 image 字段，以及以不重启不重建的方式更新 labels/annotations 字段更改除此之外的其它字段，则此次原地更新不会生效

每个模式后，有相应的策略供配置，比如 inPlaceUpdateStrategy.gracePeriodSeconds 决定原地更新时优雅更新时间、rollingUpdate.partition 来决定滚动更新时发布比例

spec.updateStrategy.inPlaceUpdateStrategy.gracePeriodSeconds 原理及用法： gracePeriodSeconds 用来实现原地升级当中的流量服务的平滑切换原地升级的更新策略下，可以配置 spec/updateStrategy/inPlaceUpdateStrategy/gracePeriodSeconds 参数，假设配置为 30 秒，那么 GameDeployment 在原地更新一个 pod 前，会通过 ReadinessGate 先把这个 pod 设置为 unready 状态，30 秒过后才会真正去原地重启 pod 中的容器这样，在这 30 秒的时间内因为 pod 变为 unready 状态，k8s 会把该 pod 实例从 service 的 endpoints 中剔除等原地升级成功后，GameDeployment 再把该 pod 设为 ready 状态，之后 k8s 才会重新把该 pod 实例加入到 service 的 endpoints 当中通过这样的逻辑，在整个原地升级过程中，能保证服务流量的无损 gracePeriodSeconds 的默认值为 0，当为 0 时，GameStatefulSet/GameDeployment 会立刻原地升级 pod 中的容器，可能会导致服务流量的丢失

GameDeployment参考配置

```
yaml updateStrategy: type: InplaceUpdate inPlaceUpdateStrategy: gracePeriodSeconds: 30 paused: false partition: 0 maxUnavailable: 1 maxSurge: 0
```

原地更新-策略指定

涉及字段：updateStrategy.inplaceUpdateStrategy.policy 可选值：DisOrdered（默认），OrderedReady, OrderedUpdated 在之前的原地更新中，容器更新顺序是不确定的，并且只有当一个容器更新完后，才会去更新下一个容器，这种方式对有容器依赖关系的 Pod 影响较大现支持指定策略去原地更新容器，分别有以下策略可指定：

- DisOrdered：默认策略，不保证容器原地更新顺序；
- OrderedReady：按容器定义顺序进行更新，当上一个容器更新并 Ready 之后，才去更新下一个容器；
- OrderedUpdated：按容器定义顺序进行更新，当上一个容器更新后，就去更新下一个容器

```
yaml updateStrategy: type: InplaceUpdate inPlaceUpdateStrategy: policy: DisOrdered
```

优雅删除/更新

涉及字段：preDeleteUpdateStrategy preInplaceUpdateStrategy PostInplaceUpdateStrategy

```
yaml spec: preDeleteUpdateStrategy: hook: templateName: test-delete preInplaceUpdateStrategy: hook: templateName: test-inplace postInplaceUpdateStrategy: hook: templateName: test-post
```

此字段表示在删除/更新应用实例前 调用定义的 Hook 以确认能否进行删除/更新操作，以及在原地更新之后，调用定义的 Hook 进行一些操作该功能与HookTemplate及HookRun资源共同使用，在使用前，请确保bcs-hook-operator已部署、对应HookTemplate已定义；

在删除 Pod 时，会首先触发 preDeleteUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric 验证逻辑，以决定能否进行 Pod 的删除；在更新 Pod 时，会首先触发 preInplaceUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric 验证逻辑，以决定能否进行 Pod 的原地更新；（若只定义了 preDeleteUpdateStrategy，在更新策略是原地更新(InplaceUpdate)时会触发 preDeleteUpdateStrategy.hook.templateName 对应的规则，来确定能否进行 Pod 原地更新，建议2种场景定义各自的策略）在原地更新 Pod 之后，触发 postInplaceUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric，进行原地更新后的一些操作

注意：只支持控制器控制下的Pod删除/更新操作（缩容、滚动更新、原地更新），若用户手工删除Pod，则不会触发这些策略；

设置 partition 灰度发布

涉及字段：GameDeployment中的updateStrategy.partition，用于实现灰度发布 Partition 的语义是 保留旧版本 Pod 的数量或百分比，默认为 0，用来控制灰度发布的个数这里的 partition 不表示任何 order 序号为了兼容，InplaceUpdate 的灰度发布个数也由这个参数配置如果在发布过程中设置了 partition：

- 如果是数字，控制器会将 (replicas - partition) 数量的 Pod 更新到最新版本
- 如果是百分比，控制器会将 (replicas * (100% - partition)) 数量的 Pod 更新到最新版本

滚动更新时最大不可用 pod 数

涉及字段: updateStrategy.maxUnavailable 指在更新过程中每批执行更新的实例数量, 在更新过程中这批实例是不可用的比如一共有 8 个实例, maxUnavailable 设置为 2, 那么每批滚动或原地重启 2 个实例, 等这 2 个实例更新完成后, 再进行下一批更新可设置为整数值或百分比, 默认值为 25%

滚动更新前最多新建旧版本 pod 数量

涉及字段: updateStrategy.maxSurge 在滚动更新过程中, 如果每批都是先删除 maxUnavailable 数量的旧版本 pod 数, 再新建新版本的 pod 数, 那么在整个更新过程中, 总共只有 replicas - maxUnavailable 数量的实例数能够提供服务在总实例数 replicas 数量比较小的情况下, 会影响应用的服务能力设置 maxSurge 后, 会在滚动更新前先多创建 maxSurge 数量的 pod, 然后再逐批进行更新, 更新完成后, 最后再删掉 maxSurge 数量的 pod, 这样就能保证整个更新过程中可服务的总实例数 maxSurge 默认值为 25%

交互式灰度发布

涉及字段: updateStrategy.canary

```
yaml
canary:
  steps:
    - partition: 3
    - pause: {}
    - partition: 2
```

在这个配置中, 假设期望实例数为 4, 使用原地重启的更新策略, 配置了 5 个灰度发布的步骤:

- 步骤 0: partition 为 3, 一共灰度 1 个实例;
- 步骤 1: 暂停灰度发布, 等待用户介入, 用户需要手动将 updateStrategy.paused 的值改为 false, 才能继续发布;
- 步骤 2: partition 为 2, 一共灰度 2 个实例;
- 步骤 3: 暂停灰度发布 60 秒, 60 秒过后继续后续步骤;
- 步骤 4: 使用名为 test 的 HookTemplate 进行 hook 调用, 如果返回的结果满足预期, 则继续执行后续步骤, 如果返回结果不满足预期, 则暂停灰度, 等待用户手动介入来决定是继续灰度发布还是进行回滚操作 如果不需要分步骤灰度发布, 那么无需配置 spec.updateStrategy.canary

发布暂停

涉及字段: updateStrategy.paused 用户可以通过设置 paused 为 true 暂停发布不过此字段只会影响更新过程, 不会影响扩缩过程, 也即如果 replicas 发生变动, pod 数量还是会相应变化

支持 HPA

可以通过 HPA 对 GameDeployment 进行自动伸缩

指定 Pod 缩容

涉及字段: scaleStrategy.podsToDelete 在 GameDeployment 中使用, 选填如果 podsToDelete 列表里写了一些 Pod 名字, 控制器会优先删除这些 Pod 对于已经被删除的 Pod, 控制器会自动从 podsToDelete 列表中清理掉 注意:

- 若只把 Pod 名字加到 podsToDelete, 但没有修改 replicas 数量, 那么控制器会先把指定的 Pod 删掉, 然后再扩一个新的 Pod;
- 若不指定 podsToDelete, 控制器按照默认顺序来选择 Pod 删除: not-ready < ready, unscheduled < scheduled, and pending < running若存在 PodDeletionCost, 则还会再按 cost 值排序, 然后选择 Pod 删除

Pod 注入唯一序号

该特性需要在 gameDeployment 的 yaml 文件注入以下 annotation, 添加 annotation 开关以及序号 range 配置

```
yaml
apiVersion: tkex.tencent.com/v1alpha1
kind: GameDeployment
metadata:
  annotations:
    tkex.bkbc.tencent.com/gamedeployment-index-on: "true"
    tkex.bkbc.tencent.com/gamedeployment-index-range:
      {"podStartIndex": 100, "podEndIndex": 110}
```

注入的唯一 ID 通过环境变量暴露给用户, 通过读取 pod 环境变量 'POD_INDEX' 可获取该 pod 注入的唯一序号

防误删功能

在 K8S 机制中, 如果删除 CRD, 会导致所有 CR 一起被级联删除, 因此如果手误删除了 GameStatefulset 的 CRD (比如误卸载组件), 会导致集群中所有 GameDeployment 实例一起被删除, 造成严重影响同样的, 如果手误删除一个 GameDeployment 实例, 也会导致其所有对应的 Pod 一起被删除为了防止误操作造成的影响, 我们加了一个 Admission Webhook, 对 GameDeployment 的 CRD 和

CR 的删除操作做验证，根据 CRD 和 CR 的 label 中对应标签值的不同，进行相应操作 在 GameDeployment 的 CR 和 CRD 中，标签效果如下（默认为空，也即是不允许删除）

key	value	效果
io.tencent.bcs.dev/deletion-allow	Always	总是允许删除
io.tencent.bcs.dev/deletion-allow	Cascading	如果有级联资源则不允许删除，如删除 HookTemplate 的 CRD 时有 HookTemplate 实例存在
io.tencent.bcs.dev/deletion-allow	不添加labels	不添加labels

推荐删除 GameDeployment 时，设置 io.tencent.bcs.dev/deletion-allow=Cascading，然后缩容副本数为0，再删除对应 GameDeployment。如果确认要删除，则设置 io.tencent.bcs.dev/deletion-allow=Always，然后方可删除，如下所示

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameDeployment metadata: name: test-gamedeployment namespace: test labels: app: test-gamedeployment io.tencent.bcs.dev/deletion-allow: Always
```

readinessgate 可选功能

在之前的版本中，gamedeployment 所管理的 pod 一定会强制加上一个 readinessgate 如下

```
yaml readinessGates: - conditionType: InPlaceUpdateReady
```

在 1.27.0 及以上版本中，此 readinessgate 改为可选项，可通过设置 spec.disableReadinessGate 值来控制是否加上此 readinessgate 若值为 false，则表示要给 pod 加上 readinessgate，若值为 true，则表示不要给 pod 加上 readinessgate

```
yaml spec: disableReadinessGate: false
```

注意：此字段值不能在原地更新策略下进行更改，因为涉及对 Pod 的 ReadinessGate 字段的修改，因此原地更新会失败，需要以重建或者滚动更新的方式，重建 Pod 之后，再改为原地更新策略，然后才会生效

GameStatefulset特性说明

CRD定义样例

GameStatefulSet 是 kubernetes 原生 StatefulSet 的增强实现，一个典型的 GameStatefulSet 定义如下

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameStatefulSet metadata: name: test-gamestatefulset namespace: test labels: app: test-gamestatefulset spec: serviceName: "test" replicas: 5 selector: matchLabels: app: test-gamestatefulset podManagementPolicy: OrderedReady updateStrategy: type: InplaceUpdate rollingUpdate: partition: 0 paused: false inPlaceUpdateStrategy: gracePeriodSeconds: 30 canary: steps: - hook: templateName: test - partition: 3 - pause: {} - partition: 2 - pause: duration: 60 preDeleteUpdateStrategy: hook: templateName: test preInplaceUpdateStrategy: hook: templateName: test-inplace template: metadata: labels: app: test-gamestatefulset spec: containers: - name: nginx image: nginx:1.7.9 imagePullPolicy: IfNotPresent resources: requests: cpu: 20m limits: cpu: 20m
```

CR 字段说明

GameStatefulSet 详细字段定义及其意义，请查阅 API 文档：http://htmlpreview.github.io/?https://github.com/Tencent/bk-bcs/blob/master/docs/features/bcs-gamestatefulset-operator/API_Reference.html

优雅删除/更新

涉及字段：preDeleteUpdateStrategy/preInplaceUpdateStrategy

```
yaml spec: preDeleteUpdateStrategy: hook: templateName: test-delete preInplaceUpdateStrategy: hook: templateName: test-inplace postInplaceUpdateStrategy: hook: templateName: test-post
```

此字段表示在删除/更新应用实例前 调用定义的 Hook 以确认能否进行删除/更新操作，以及在原地更新之后，调用定义的 Hook 进行一些操作该功能与 HookTemplate 及 HookRun 资源共同使用，在使用前，请确保 bcs-hook-operator 已部署、对应 HookTemplate 已定义

在删除 Pod 时，会首先触发 preDeleteUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric 验证逻辑，以决定能否进行 Pod 的删除；在更新 Pod 时，会首先触发 preInplaceUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric 验证逻辑，以决定能否进行 Pod 的原地更新；（若只定义了 preDeleteUpdateStrategy，在更新策略是原地更新(InplaceUpdate)时会触发 preDeleteUpdateStrategy.hook.templateName 对应的规则，来确定能否进行 Pod 原地更新，建议2种场景定义各自的策略）在原地更新 Pod 之后，触发 postInplaceUpdateStrategy.hook.templateName 对应的 HookRun 对象创建，Hook-Operator 执行 metric，进行原地更新后的一些操作 注意：只支持控制器控制下的Pod删除/更新操作（缩容、滚动更新、原地更新），若

用户手工删除Pod，则不会触发这些策略；

更新策略

涉及字段：updateStrategy updateStrategy.type 支持 RollingUpdate, OnDelete, InplaceUpdate 三种更新方式，相比原生StatefulSet, 新增 InplaceUpdate 更新模式

每个模式后，有相应的策略供配置，比如 inPlaceUpdateStrategy.gracePeriodSeconds 决定原地更新时优雅更新时间、rollingUpdate.partition 来决定滚动更新时发布比例

spec/updateStrategy/inPlaceUpdateStrategy/gracePeriodSeconds 原理及用法：gracePeriodSeconds 用来实现原地升级当中的流量服务的平滑切换原地升级的更新策略下，可以配置 spec/updateStrategy/inPlaceUpdateStrategy/gracePeriodSeconds 参数，假设配置为 30 秒，那么 GameStatefulSet/GameDeployment 在原地更新一个 pod 前，会通过 ReadinessGate 先把这个 pod 设置为 unready 状态，30 秒过后才会真正去原地重启 pod 中的容器这样，在这 30 秒的时间内因为 pod 变为 unready 状态，k8s 会把该 pod 实例从 service 的 endpoints 中剔除等原地升级成功后，GameStatefulSet/GameDeployment 再把该 pod 设为 ready 状态，之后 k8s 才会重新把该 pod 实例加入到 service 的 endpoints 当中通过这样的逻辑，在整个原地升级过程中，能保证服务流量的无损 gracePeriodSeconds 的默认值为 0，当为 0 时，GameStatefulSet/GameDeployment 会立刻原地升级 pod 中的容器，可能会导致服务流量的丢失

注意：InplaceUpdate，只支持以原地重启容器的方式更新 image 字段，以及以不重启不重建的方式更新 labels/annotations 字段更改除此之外的其它字段，则此次原地更新会卡住！

下面是 GameStatefulSet 的 workload 类型参考配置

```
yaml updateStrategy: type: InplaceUpdate inPlaceUpdateStrategy: gracePeriodSeconds: 30 paused: false
rollingUpdate: partition: 0
```

原地更新-策略指定

涉及字段：updateStrategy.inplaceUpdateStrategy.policy 可选值：DisOrdered（默认），OrderedReady, OrderedUpdated 在之前的原地更新中，容器更新顺序是不确定的，并且只有当一个容器更新完后，才会去更新下一个容器，这种方式对有容器依赖关系的 Pod 影响较大现支持指定策略去原地更新容器，分别有以下策略可指定：

- DisOrdered: 默认策略，不保证容器原地更新顺序；
- OrderedReady: 按容器定义顺序进行更新，当上一个容器更新并 Ready 之后，才去更新下一个容器；
- OrderedUpdated: 按容器定义顺序进行更新，当上一个容器更新后，就去更新下一个容器

```
yaml updateStrategy: type: InplaceUpdate inPlaceUpdateStrategy: policy: DisOrdered
```

交互式灰度发布

涉及字段：updateStrategy.canary

```
yaml canary: steps: - partition: 3 - pause: {} - partition: 2 - pause: {duration: 60}
```

在这个配置中，假设期望实例数为 4，使用原地重启的更新策略，配置了 5 个灰度发布的步骤：

- 步骤 0: partition 为 3，灰度 1 个实例；
- 步骤 1: 暂停灰度发布，等待用户介入，用户需要手动将 updateStrategy.paused 的值改为 false，才能继续发布；
- 步骤 2: partition 为 2，灰度 2 个实例；
- 步骤 3: 暂停灰度发布 60 秒，60 秒过后继续后续步骤；
- 步骤 4: 使用名为 test 的 HookTemplate 进行 hook 调用，如果返回的结果满足预期，则继续执行后续步骤，如果返回结果不满足预期，则暂停灰度，等待 用户手动介入来决定是继续灰度发布还是进行回滚操作 如果不需要分步骤灰度发布，那么无需配置 spec.updateStrategy.canary

发布暂停

涉及字段：updateStrategy.paused 用户可以通过设置 paused 为 true 暂停发布不过此字段只会影响更新过程，不影响扩缩容，也即如果 replicas 发生变动，pod 数量还是会相应变化

滚动/原地更新时灰度个数

涉及字段：GamestatefulSet中的updateStrategy.rollingUpdate.partition Partition 的语义是 保留旧版本 Pod 的数量或百分比，默认为 0，用来控制灰度发布的个数这里的 partition 不表示任何 order 序号为了兼容，InplaceUpdate 的灰度发布个数也由这个参数配置

并行更新/删除

涉及字段：podManagementPolicy

```
yaml podManagementPolicy: Parallel
```

在GameStatefulSet中使用 支持 "OrderedReady" 和 "Parallel" 两种方式，定义和 StatefulSet 一致，默认为 OrderedReady与 StatefulSet 不同的是，如果配置为 Parallel，那么不仅删除和创建 pod 实例是并行的，实例更新也是并行的，即自动并行更新并行更新时会综合考虑 maxUnavailable 和 maxSurge 字段

支持 maxUnavailable/maxSurge

- 滚动/原地更新时支持最大不可用pod数（maxUnavailable） 涉及字段：GamestatefulSet中的 updateStrategy.rollingUpdate.maxUnavailable 指在更新过程中每批执行更新的实例数量，在更新过程中这批实例是不可用的比如一共有 8 个实例，maxUnavailable 设置为 2，那么每批滚动或原地重启 2 个实例，等这 2 个实例更新完成后，再进行下一批更新可设置为整数值或百分比，默认值为 25% 只有在 **podManagementPolicy** 为 Parallel 时才会生效，并且只有更新时才会生效
- 滚动/原地更新前支持最多新建新版本pod数量（maxSurge） 涉及字段：GamestatefulSet中的 updateStrategy.rollingUpdate.maxSurge 设置 maxSurge 后，会先多创建 maxSurge 数量的新版本 pod，然后再逐批进行更新，更新完成后，最后再删掉 maxSurge 数量的 pod，这样就能保证整个更新过程中可服务的总实例数量比如一共有 8 个实例，maxSurge 设置为 2，那么在更新前会新建两个新版本的实例，然后再去处理旧版本实例，更新完成后再删除多出来的两个实例 maxSurge 默认值为 0 只有在 **podManagementPolicy** 为 Parallel 时才会生效，并且只有更新时才会生效

支持 HPA

可以通过 HPA 对 GameStatefulset进行自动伸缩

服务名称

涉及字段：serviceName 在GameStatefulSet中使用，必填这个参数指的是控制此GamestatefulSet的服务 name此service必须在 GameStatefulset 创建前创建，负责对应pod的网络ID命名如果你的 pod 之间是完全独立的（不需要服务发现），你可以随意指定一个 service（不需要提前创建，实际上就是为 pod 增加了一个 subdomain）但是大多数应用都需要服务发现，这时创建 headless service 就显得非常必要，之后你可以通过 my-headless-service.my-namespace.svc.cluster.local 解析到所有 pod 的 ip，也可以通过 my-pod.my-headless-service.my-namespace.svc.cluster.local 解析到某个 pod 的 ip

防误删功能

在 K8S 机制中，如果删除 CRD，会导致所有 CR 一起被级联删除，因此如果手误删除了 GameStatefulset 的 CRD（比如误卸载组件），会导致集群中所有 GameStatefulset 实例一起被删除，造成严重影响同样的，如果手误删除一个 GameStatefulset 实例，也会导致其所有对应的 Pod 一起被删除 为了防止误操作造成的影响，我们加了一个 Admission Webhook，对 GameStatefulset 的 CRD 和 CR 的删除操作做验证，根据 CRD 和 CR 的 label 中对应标签值的不同，进行相应操作 在 GameStatefulset 的 CR 和 CRD 中标签效果如下（默认为空，也即是不允许删除）

key	value	效果
io.tencent.bcs.dev/deletion-allow	Always	总是允许删除
io.tencent.bcs.dev/deletion-allow	Cascading	如果有级联资源则不允许删除，如删除 HookTemplate 的 CRD 时有 HookTemplate 实例存在
io.tencent.bcs.dev/deletion-allow	不添加labels	不添加labels
io.tencent.bcs.dev/deletion-allow	不允许删除	不允许删除

推荐删除 GameStatefulset 时，设置 io.tencent.bcs.dev/deletion-allow=Cascading，然后缩容副本数为0，再删除对应 GameStatefulset。如果确认要删除，则设置 io.tencent.bcs.dev/deletion-allow=Always，然后方可删除，如下所示

```
yaml apiVersion: tkex.tencent.com/v1alpha1 kind: GameStatefulSet metadata: name: test-gamestatefulset namespace: test labels: app: test-gamestatefulset io.tencent.bcs.dev/deletion-allow: Always
```

readinessgate 可选功能

在之前的版本中，gamedeployment 所管理的 pod 一定会强制加上一个 readinessgate 如下

```
yaml readinessGates: - conditionType: InPlaceUpdateReady
```

在 1.27.0 及以上版本中，此 readinessgate 改为可选项，可通过设置 spec.disableReadinessGate 值来控制是否加上此 readinessgate 若

值为 false，则表示要给 pod 加上 readinessgate，若值为 true，则表示不要给 pod 加上 readinessgate

```
yaml spec: disableReadinessGate: false
```

注意：此字段值不能在原地更新策略下进行更改，因为涉及对 Pod 的 ReadinessGate 字段的修改，因此原地更新会失败，需要以重建或者滚动更新的方式，重建 Pod 之后，再改为原地更新策略，然后才会生效

Hook 发布控制

可以在应用发布过程中配置 hook 勾子，通过 hook 结果判断是否可以继续下一步发布流程

自动扩缩容

支持与原生HPA相结合进行服务的自动扩缩容并且，可以配置相应的 Hook 发布控制，来保证优雅缩容避免在线业务被强杀 以下是针对 GameDeployment的HPA配置示例：

```
yaml apiVersion: autoscaling/v2beta2 kind: HorizontalPodAutoscaler metadata: name: goserver-hpa spec: scaleTargetRef: apiVersion: tkex.tencent.com/v1alpha1 kind: GameDeployment name: test-gamedeployment minReplicas: 1 maxReplicas: 10 metrics: - type: Resource resource: name: cpu target: type: Utilization averageUtilization: 80
```

Service 列表

Service 列表用于展示集群中所有 Service 信息，包含名称、Service 类型、集群内访问或集群外访问方式等，并且允许通过页面更新和删除 Service 资源。

列表展示

Service						帮助
新测试集群						输入关键字，按Enter搜索
批量删除		操作	所属集群/命名空间	类型	集群内访问	集群外访问
<input type="checkbox"/>	rumpetroll-2012240854-openresty	更新 删除	所属集群: 新测试集群 命名空间: bcs-test	ClusterIP	TCP	--
<input type="checkbox"/>	rumpetroll-2012240854-redis	更新 删除	所属集群: 新测试集群 命名空间: bcs-test	ClusterIP	TCP	--
<input type="checkbox"/>	rumpetroll-2012240854-rumpetroll	更新 删除	所属集群: 新测试集群 命名空间: bcs-test	ClusterIP	--80 TCP	--
<input type="checkbox"/>	test-svc	更新 删除	所属集群: 新测试集群 命名空间: bcs-test	ClusterIP	UDP	--

详情查看

Service						帮助
po-prometheus-operator-kubelet						操作
基础信息						操作
选择器:	--	类型:	ClusterIP	Service IP:	None	---
端口映射	端口名称	端口	协议	目标端口	NodePort	
https-metrics	10250	TCP	10250	--		
Endpoints						操作
名称	Pod IP	Node IP				
ip-10-0-5-150-m-bcs-k8s-40003	10.0.5.150	--				
ip-10-0-5-75-n-bcs-k8s-40003	10.0.5.75	--				
其他信息						操作
创建时间:	2020-12-25 20:32:19	更新时间:	2020-12-25 20:32:19	更新人:	---	来源:
						Client
标签						操作
k8s-app	kubelet					

更新操作

注意：仅允许操作通过表单模板集部署的 service 资源

The screenshot shows two parts of the Service management interface. The top part is a list view with a '批量删除' (Batch Delete) button. It lists several services: rumpetroll-2012240854-openresty, rumpetroll-2012240854-redis, rumpetroll-2012240854-rumpetroll, and test-svc. The bottom part is a detailed edit form titled '更新Service' (Update Service). It shows the service name 'test-svc' and its association with 'test-deployment'. It includes sections for '关联应用' (Associated Application), '关联标签' (Associated Labels), 'Service类型' (Service Type), '端口映射' (Port Mapping), '标签管理' (Label Management), and '备注管理' (Comment Management). A note at the bottom says '小提示：同时粘贴多行“键=值”的文本会自动添加多行记录' (Tip: Pasting multi-line "key=value" text will automatically add multiple entries).

删除操作

This screenshot shows a list view of services for deletion. The '批量删除' (Batch Delete) button is visible. One service, 'test-svc', has its '删除' (Delete) button highlighted with a red border.

Ingress 列表

Ingress 列表展示集群中所有 Ingress 信息，通过列表页面方便查看、更新及删除 Ingress 资源。

列表展示

The screenshot shows a list view of Ingress resources. There is one entry: 'test-ingress'. The table columns include '名称' (Name), '所属集群' (Cluster), '命名空间' (Namespace), '来源' (Source), '创建时间' (Created Time), '更新时间' (Updated Time), and '操作' (Operations). The '操作' column for 'test-ingress' contains '更新' (Update) and '删除' (Delete) buttons.

更新操作

注意：仅允许操作通过模板集部署的 ingress 资源

The screenshot shows two pages of the Ingress management interface. The top page is a list view with a single entry: 'test-ingress' under '新测试集群' in the '命名空间' column. The bottom page is a detailed view of the 'test-ingress' entry, showing its configuration. The '操作' column in both tables has a '更新' button, which is highlighted with a red box in the list view.

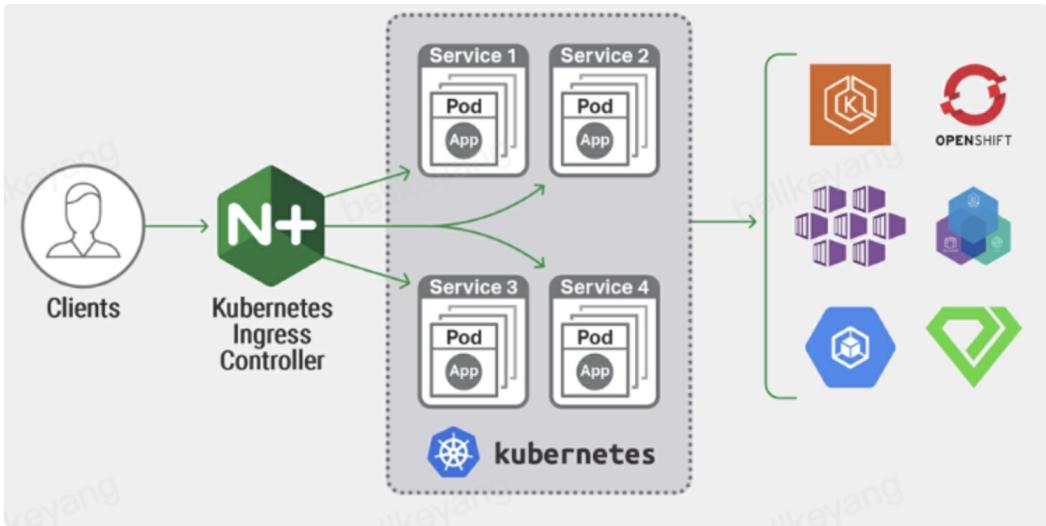
删除操作

The screenshot shows the same two pages of the Ingress management interface. The '操作' column in both tables has a '删除' button, which is highlighted with a red box in the list view.

LoadBalancer

容器服务 LoadBalancer 采用 K8S 官方维护的 nginx ingress controller 方案，通过 Helm Chart 的方式进行部署管理。LoadBalancer 主要包含两部分：nginx 和 nginx controller，其中，nginx controller 主要负责监听 Ingress 规则，最终转换成 nginx 配置。

如下图所示，用户通过 LoadBalancer 管理的 ingress 配置，访问对应的服务。



部署 LoadBalancer

选择集群，【添加节点】，调整 Values 中 `replicaCount` 的值和节点数量相同，部署 LoadBalancer。

```

controller:
  name: controller
  image:
    repository: /public/bcs/k8s/nginx-ingress-controller
    tag: "0.12.0"
    pullPolicy: IfNotPresent
  
```

LoadBalancer 列表

展示集群、命名空间(默认为 bcs-system)及对应的 chart 和版本

所属集群	命名空间	Chart名称及版本	更新时间	更新人	操作
新测试集群	bcs-system	名称: blueking-ingress-ingress 版本: 1.0.6	2020-12-28 10:32:48	admin	<button>更新</button> <button>删除</button>

更新 LoadBalancer

调整部署节点及 Values 配置，更新集群 LoadBalancer。

The screenshot shows two pages of the BCS console:

- LoadBalancer** page: Displays a list of LoadBalancers. One entry is shown: "新测试集群" in namespace "bcs-system" with chart "blueking-inginx-ingress" version "1.0.6".
- 编辑LoadBalancer** page: Shows the configuration for the selected LoadBalancer. It includes fields for "所属集群" (selected: "新测试集群 (BCS-K8S-40001)"), "节点IP" (IP address), and "调整节点" (button). The "Values内容" section displays the YAML configuration for the controller pod:

```

1- controller:
2   name: controller
3   image:
4     repository: /public/bcs/k8s/nginx-ingress-controller
5     tag: "0.12.0"
6     pullPolicy: IfNotPresent
7
8- # Will add custom header to Nginx https://github.com/kubernetes
# /ingress-nginx/tree/master/docs/examples/customization
# /custom-headers
9 # Recommended use with CN1 based kubernetes installations
# (can't do it now set up by kubeadm).
10 # since CN1 and hostport don't mix yet. Can be deprecated once
# https://github.com/kubernetes/kubernetes/issues/23920
11 # is merged
12   hostNetwork: true
13
14- # Optionally change this to ClusterFirstWithHostNet in case you
# have 'hostNetwork: true'.
15 # By default, while using host network, name resolution uses
# the host's DNS. If you wish nginx-controller
16 # to keep resolving names inside the k8s network, use
# ClusterFirstWithHostNet.
17   dnsPolicy: ClusterFirstWithHostNet
  
```

bcs-ingress-controller TKE使用指南

bcs-ingress-controller介绍

bcs-ingress-controller是蓝鲸容器服务扩展 Ingress，多云环境下为业务服务提供灵活的流量接入方式，目前支持的云服务商有Tencent TKE、Amazon EKS、Google GKE

特性支持

- 支持HTTPS, HTTP, TCP, UDP协议
- 支持腾讯云Clb健康检查等参数配置
- 支持单个ingress同时控制多个clb实例
- 支持转发到NodePort模式和转发到直通Pod模式
- 支持单端口多Service流量转发，以及WRR负载均衡方法下权重配比
- 直通Pod模式下，支持Service内部通过Label选择Pod，以及WRR负载均衡方法下权重配比
- 支持StatefulSet和GameStatefulSet端口段映射
- 云接口的客户端限流与重试
- 为任意Pod动态分配端口

bcs-ingress-controller组件部署

1. 腾讯云账户与权限

部署bcs-ingress-controller组件的SecretID、SecretKey至少需要有如下权限： ["clb:Describe*", "clb:DescribeTargetHealth", "clb:DescribeForwardLBListeners", "clb:DescribeLoadBalancerListeners", "clb:DescribeForwardLBBackends"]

```

"clb:DescribeExclusiveClusters", "clb:DescribeBlockIPList", "clb:DescribeBlockIPTask", "clb:DescribeClassicalLBByInstanceId",
"clb:DescribeClusterResources", "clb:DescribeLoadBalancerListByCertId", "clb:DescribeQuota", "clb:DescribeSetInnerName",
"clb:DescribeCustomizedConfigList", "clb:DescribeCustomizedConfigContent", "clb:AutoRewrite", "clb:ModifyTargetWeight",
"clb:BatchModifyTargetWeight", "clb>CreateListener", "clb>DeleteListener", "clb>CreateRule", "clb>DeleteRule", "clb:ModifyRule",
"clb:ManualRewrite", "clb:ModifyListener", "clb:RegisterTargets", "clb:DeregisterTargets", "clb:BatchDeregisterTargets",
"clb:ReplaceCertForLoadBalancers", "clb:DeleteLoadBalancerListeners", "clb:BatchRegisterTargets", "clb:BatchDeregisterTargets",
"clb:AssociateTargetGroups", "clb:DescribeTargetGroupInstances", "clb:DescribeTargetGroupList", "clb:DescribeTargetGroups",
"clb:DisassociateTargetGroups", "cvm:DescribeInstances"]

```

账户授权

设置单独的权限需要添加自定义策略，参考 <https://cloud.tencent.com/document/product/598/37739>，然后给予账号授权该策略。

为了方便，可以给直接子账号授权预设策略中 CLB 全部权限和 CVM 只读权限，这样可以不用创建自定义策略。如：

关联策略

选择策略（共 13 条）

clb	
策略名	策略类型
QcloudCLBFinanceAccess 负载均衡 (CLB) 财务权限	预设策略
<input checked="" type="checkbox"/> QcloudCLBFullAccess 负载均衡 (CLB) 全读写访问权限	预设策略
QcloudCLBReadOnlyAccess 负载均衡 (CLB) 只读访问权限	预设策略
QcloudCVMFullAccess 云服务器 (CVM) 全读写访问权限，包括...	预设策略
QcloudTKEFullAccess	

支持按住 shift 键进行多选

已选择 2 条

策略名	策略类型
QcloudCVMInnerReadOnlyAccess 云服务器 (CVM) 只读访问权限	预设策略
QcloudCLBFullAccess 负载均衡 (CLB) 全读写访问权限	预设策略

确定

取消

生成 SECRET ID 和 SECRET KEY

参考 <https://cloud.tencent.com/document/product/598/37140> 通过子账号生成 Secret ID 和 Secret Key

2. 组件部署

在组件库中找到“BcsIngressController”，点击“启用”按钮

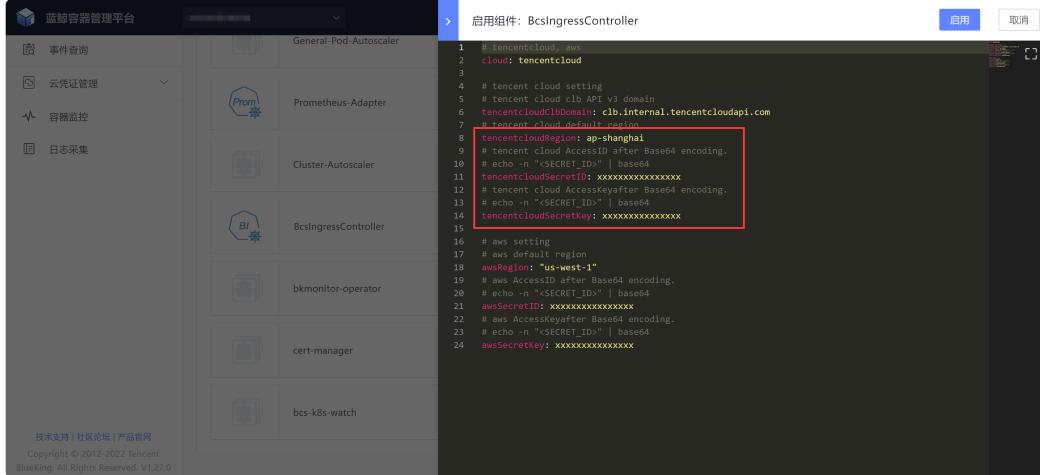
The screenshot shows the Bluewhale Component Management Platform interface. On the left sidebar, there are categories: 事件查询, 云凭证管理, 容器监控, and 日志采集. The main area lists components with their status and descriptions:

- General-Pod-Autoscaler: 未启用 (Not Enabled)
- Prometheus-Adapter: 未启用 (Not Enabled)
- Cluster-Autoscaler: 未启用 (Not Enabled)
- BcsIngressController**: 未启用 (Not Enabled). Description: 蓝鲸容器服务扩展 Ingress，多云环境下为 GameServer 提供灵活的流量接入方式. An 'Enable' button is highlighted with a red box.
- bkmonitor-operator: 未启用 (Not Enabled)
- cert-manager: 未启用 (Not Enabled)
- bcs-k8s-watch: 未启用 (Not Enabled)

At the bottom left, there is a footer with links: 技术支持 | 社区论坛 | 产品官网. Copyright © 2012-2022 Tencent BlueKing. All Rights Reserved. V1.27.0. At the bottom right, there is a 'WebConsole' link.

配置TKE集群所在Region、SecretID、SecretKey参数后即可点击“启用”按钮部署bcs-ingress-controller组件，填入SecretID、SecretKey内容记得用base64加密

```
bash echo -n "<SecretID>" | base64 echo -n "<SecretKey>" | base64
```



bcs-ingress-controller快速入门

1. 创建 CLB

参考 <https://cloud.tencent.com/document/product/214/6149> 创建腾讯云 CLB，地域和 VPC 需要和当前集群一致。实例类型选推荐类型，即【负载均衡（原“应用型负载均衡”）】。网络类型公网和内网都可以，IP 版本选 IPv4，其他默认即可。创建完成后，进入负载均衡控制台获取 CLB ID。如：

2. 准备 workload

为了验证 ingress 绑定 pod ip，需要提前准备好对应的 workload，保存下面内容为 ingress-test.yaml

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx namespace: ingress-test labels: app: nginx spec: replicas: 1 selector: matchLabels: app: nginx template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.14.2 ports: - containerPort: 80
```

```
apiVersion: apps/v1 kind: Service metadata: name: nginx namespace: ingress-test spec: selector: app: nginx ports: - protocol: TCP port: 8080 targetPort: 80 type: NodePort 把以下内容保存为文件nginx-sts.yaml
```

```
yaml apiVersion: apps/v1 kind: StatefulSet metadata: name: nginx-sts namespace: ingress-test labels: app: nginx-sts spec: replicas: 3 serviceName: nginx-sts selector: matchLabels: app: nginx-sts
```

```
template: metadata: labels: app: nginx-sts spec: containers: - name: nginx image: nginx:1.14.2 ports: - containerPort: 80
```

```
apiVersion: v1 kind: Service metadata: name: nginx-sts namespace: ingress-test spec: selector: app: nginx-sts ports: - protocol: TCP port: 8080 targetPort: 80 type: ClusterIP ...
```

执行如下命令，创建workload `bash kubectl apply -f ingress-test.yaml kubectl apply -f nginx-sts.yaml`

3. 安装组说明

- CVM 安全组：控制 CLB 到 CVM 的流量，如果开启默认放通，则不用特地设置 CVM 安全组，否则需要放通 CLB 到 CVM 后端端口。如果是 Pod 直通，则已 Pod 端口为后端端口设置安全组。
- CLB 安全组：控制客户端到 CLB 的流量，放通指定客户端即可



4. 快速体验

NODEPORT转发

`networkextension.bkbcn.tencent.com/lbids` 替换为自己的 lb，创建完成后，查看腾讯云 CLB 监听器，看 38080 端口是否被创建，后端 ip port 是否正常，ip 应该为 pod 所在节点的 ip，port 为 service 的 nodeport `yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test1 namespace: ingress-test annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxx spec: rules: - port: 38080 protocol: TCP services: - serviceName: nginx serviceNamespace: ingress-test servicePort: 8080`

CLB直通POD

`networkextension.bkbcn.tencent.com/lbids` 替换为自己的 lb 创建完成后，查看腾讯云 CLB 监听器，看 39090 端口是否被创建，后端 ip port 是否正常，ip 应该为 pod 的 ip，port 为 80 `yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test2 namespace: ingress-test annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxx spec: rules: - port: 39090 protocol: TCP services: - serviceName: nginx serviceNamespace: ingress-test servicePort: 8080 isDirectConnect: true`

端口段

`networkextension.bkbcn.tencent.com/lbids` 替换为自己的 lb 创建完成后，查看腾讯云 CLB 监听器，观察 30000-30001, 30002-30003, 30004-30005 三个端口号是否被创建，因为我们的验证Pod只监听一个80端口，所以只有vport映射的80端口是通的，30001映射的81, 30002映射的82, 30003映射的83, 30004映射的84, 30005映射的85都不可访问是正常现象，主要看CLB的规则是否创建正确即可

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test3 namespace: ingress-test annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxx spec: portMappings: - startPort: 30000 protocol: TCP startIndex: 0 endIndex: 3 segmentLength: 2 workloadKind: StatefulSet workloadName: nginx-sts workloadNamespace: ingress-test rsStartPort: 80
```

端口池

步骤一：创建端口池 `loadBalancerIDs` 替换为自己的 lb

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: pool1 namespace: ingress-test
```

```
test spec: poolItems: - itemName: item1 loadBalancerIDs: ["lb-xxx"] startPort: 31000 endPort: 31010 查看 Portpool  
是否创建成功, status 为 ready 说明创建成功 kubectl describe portpool pool1 -n ingress-test 步骤二: 声明命名空间使用端  
口池 给 ingress-test 命名空间打上 label bash kubectl label ns ingress-test bcs-ingress-controller-inject=true 步骤三:  
声明 pod 使用端口池 给之前创建的 nginx deployment 打上 annotation, 声明使用端口池。 annotations 内容:  
portpools.networkextension.bkbcns.tencent.com: "true" ports.portpools.networkextension.bkbcns.tencent.com: "{端口  
池名称} {端口协议名称} {端口号}" 例如: yaml apiVersion: apps/v1 kind: Deployment metadata: name: nginx-portpool  
namespace: ingress-test labels: app: nginx-portpool spec: replicas: 1 selector: matchLabels: app: nginx-  
portpool template: metadata: labels: app: nginx-portpool annotations:  
portpools.networkextension.bkbcns.tencent.com: "true" ports.portpools.networkextension.bkbcns.tencent.com: "pool1  
TCP 80" spec: containers: - name: nginx-portpool image: nginx:1.14.2 ports: - containerPort: 80 创建完成后, 查看  
腾讯云 CLB 监听器, 观察 31000 - 31009 端口是否被监听, 创建完后31000映射的80端口是正常的, 其它端口因为没有在容器中监听,  
所以没有绑定后端服务
```

常用命令

```
```bash
```

## 查看 Ingress

```
kubectl get ingress.networkextension.bkbcns.tencent.com -n namespace
```

## 查看监听器

```
kubectl get listener -n namespace
```

## 查看端口池

```
kubectl get portpool -n namespace
```

## 查看端口池绑定 pod

```
kubectl get portbinding -n namespace ``
```

### bcs-ingress-controller全部使用场景

#### 1. CLB转到Service NodePort

##### 场景描述

- 命名空间 **default** 下有名为 **test-svc** 的NodePort类型的Service
- service **test-svc** 中有端口号为8080的TCP端口, 对应的NodePort为31003
- service **test-svc** 8080端口对应后端pod端口同样为8080
- clb实例的id是**lb-xxxxxx**
- Overlay集群, 从集群外无法访问集群内Pod的IP地址

##### 想要达到的效果

- clb流量无法直接转发到Pod上, 需要转到对应Service的NodePort上
- 需要对外暴露TCP 38080端口

```
text clb(38080 port) -----> service对应pod所在的node节点(31003 port) -----> pod(8080)
```

##### 配置示例

```
yaml apiVersion: networkextension.bkbcns.tencent.com/v1 kind: Ingress metadata: name: test1 annotations:
networkextension.bkbcns.tencent.com/lbids: lb-xxxxxx spec: rules: - port: 38080 protocol: TCP services: -
serviceName: test-svc serviceNamespace: default servicePort: 8080
```

## 2. CLB直通 Pod

### 场景描述

- 命名空间 **default** 下有名为 **test-svc** 的NodePort类型的Service
- service **test-svc** 中有端口号为28080的TCP端口
- service **test-svc** 28080端口对应后端deployment中的pod端口为8080
- Pod拥有underlay IP地址，从集群外可以访问集群内Pod的IP地址（或者Clb实例开通了SnatPro）
- clb实例的id是 **lb-xxxxxx**

### 想要达到的效果

- 假设集群是Underlay的，流量需要直通Pod
- 需要对外暴露TCP 38080端口

```
text clb(38080 port) -----> pod(8080)
```

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test2 annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx spec: rules: - port: 38080 protocol: TCP services: - serviceName: test-svc serviceNamespace: default servicePort: 28080 isDirectConnect: true 注意: service 对应的工作负载需要在 pod template 中给容器暴露的端口指定 containerPort，如: yaml apiVersion: apps/v1 kind: Deployment ... spec: ... template: ... spec: containers: - name: nginx image: nginx:1.14.2 ports: - containerPort: 80 # 对应 service 中的 targetPort protocol: TCP
```

## 3. 使用 hostPort

### 场景描述

- 命名空间 **default** 下有名为 **test-svc** 的Service
- service **test-svc** 中有端口号为28080的TCP端口
- service **test-svc** 28080端口对应后端deployment中的pod端口为8080，pod 8080 对应的 hostPort 为 18080
- clb实例的id是 **lb-xxxxxx**

### 想要达到的效果

- 需要对外暴露TCP 38080端口

```
text clb(38080 port) -----> pod hostPort(18080)
```

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test2 annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx spec: rules: - port: 38080 protocol: TCP services: - serviceName: test-svc serviceNamespace: default servicePort: 28080 isDirectConnect: true hostPort: true
```

## 4. CLB对外暴露 HTTPS (单向验证) 协议443端口

### 场景描述

- 命名空间 **default** 下有名为 **test-svc** 的NodePort类型的Service
- service **test-svc** 中有端口号为8080的TCP端口，对应的NodePort为31003
- service **test-svc** 8080端口对应后端pod端口同样为8080
- Overlay集群，从集群外无法访问集群内Pod的IP地址
- 腾讯云上HTTPS证书ID为**xxx**，单向验证
- clb实例的id是 **lb-xxxxxx**

### 想要达到的效果

- clb流量无法直接转发到Pod上，需要转到对应Service的NodePort上
- 需要对外暴露TCP 443端口

```
text clb(443 port)(www.qq.com)(/path1) -----> service对应pod所在的node节点(31003 port) -----> pod(8080)
```

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test3 annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx spec: rules: - port: 443 protocol: HTTPS certificate: mode: UNIDIRECTIONAL certID: xxx layer7Routes: - domain: www.qq.com path: /path1 services: - serviceName: test-tcp serviceNamespace: default servicePort: 8080
```

## 5. StatefulSet/GameStatefulSet端口段映射

### 场景描述

- 命名空间 **game** 下有名为 **gameserver** StatefulSet，有6个Pod
- gameserver** 的每个Pod都需要单独直接对外提供UDP服务
- gameserver** 每个Pod的端口号与Pod需要有一定对应关系
- gameserver** clb直通到每个Pod，underlay网络和host网络不需要额外设置
- VPC路由方案（overlay）参考[在TKE上使用负载均衡直通Pod](#)
- clb实例的id是lb-xxxxxx

### 想要达到的效果

- 端口映射
- gameserver-0对应30000~30004端口
- gameserver-1对应30005~30009端口
- .....
- gameserver-5对应30020~30024端口

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test4 annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx spec: portMappings: - startPort: 30000 protocol: UDP # [startIndex, endIndex), 左闭右开区间 startIndex: 0 endIndex: 6 segmentLength: 5 # 目前支持StatefulSet和 GameStatefulSet workloadKind: StatefulSet workloadName: gameserver workloadNamespace: game # rsStartPort: 3000 # 设置后端pod的起始端口 # isRsPortFixed: false # 设置所有后端Pod端口是否一样 # hostPort: false # 是否使用 hostPort
```

注意：端口段映射依赖公网clb端口段映射接口，该接口只支持公网clb的TCP和UDP协议。当ignoreSegment为false时，默认使用端口段接口，即一个端口段对应一个规则；而当ignoreSegment为true时，使用普通接口，一个端口对应一个规则；使用端口段接口可以使得一个clb实例注册更多端口使用hostPort需要开启 **isRsPortFixed** 和 **ignoreSegment**，同时需要定义端口段中所有端口的 **containerPort** 和 **hostPort**，如果使用随机HostPort方案，则可不手动指定hostPort。

## 6. 随机 HostPort方案

### 场景描述

Kubernetes技术已经成为潮流，但是Kubernetes的集群流量导入方案并不能够完美适应各种场景：

一般情况下，从集群外部进入集群内的流量，必定经过一层service或者Ingress的转发，存在一层性能损失；而且k8s service对tcp长连接的负载均衡不太理想（连接数较少的情况下，比如30个连接），会导致业务中长连接服务的负载不均衡，进而导致资源浪费以及故障时影响面可能较大。很多云提供商借助云网络，能够实现负载均衡器直连Pod，但是Pod需要消耗VPC内underlay的IP资源。如果在裸金属机器上，则就不能实现Pod的直接访问。在无云基础设施的情况下，Pod之间无法进行跨集群的服务发现。有状态服务之间需要互访、同时供外部访问。如果能够利用Kubernetes hostport特性，实现hostport的随机分配，在无云网络特别支持的情况下，也能够提供Pod之

间的跨集群访问，同时避免service层转发带来的性能开销。将pod的监听端口映射到其所在node的某个随机可用的端口，从而集群内外都可以通过node上的这个端口来访问这个pod

具体流程如下：

pod启动的时候，把pod想要监听的端口随机映射到该pod所在node上的某个随机可用的端口 通过环境变量，在pod内进程启动之前，把node的ip和port告诉pod里的业务进程 pod内的业务进程启动时，可以把该node的ip和port通过服务发现等机制告诉k8s集群内部或外部的其他业务进程 k8s集群内部或外部的其他业务进程访问该node的ip和port，node会把流量转发给pod，从而达到了与pod通信的目的

## 使用方法

- 针对带有特定annotation的Pod进行Hook

```
shell randhostport.webhook.bkbc.tencent.com: "true" ports.randhostport.webhook.bkbc.tencent.com:
"8080,metric"
```

- 被注入Pod内容变化过程

注入前的Deployment

```
yaml apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment labels: app: nginx spec: replicas: 3
selector: matchLabels: app: nginx template: metadata: labels: app: nginx annotations:
randhostport.webhook.bkbc.tencent.com: "true" # 可以是端口名字或者端口号
ports.randhostport.webhook.bkbc.tencent.com: "8080,metric" spec: containers: - name: nginx image: nginx:1.14.2
ports: - containerPort: 8080 - name: metric containerPort: 8081
```

注入后的Pod

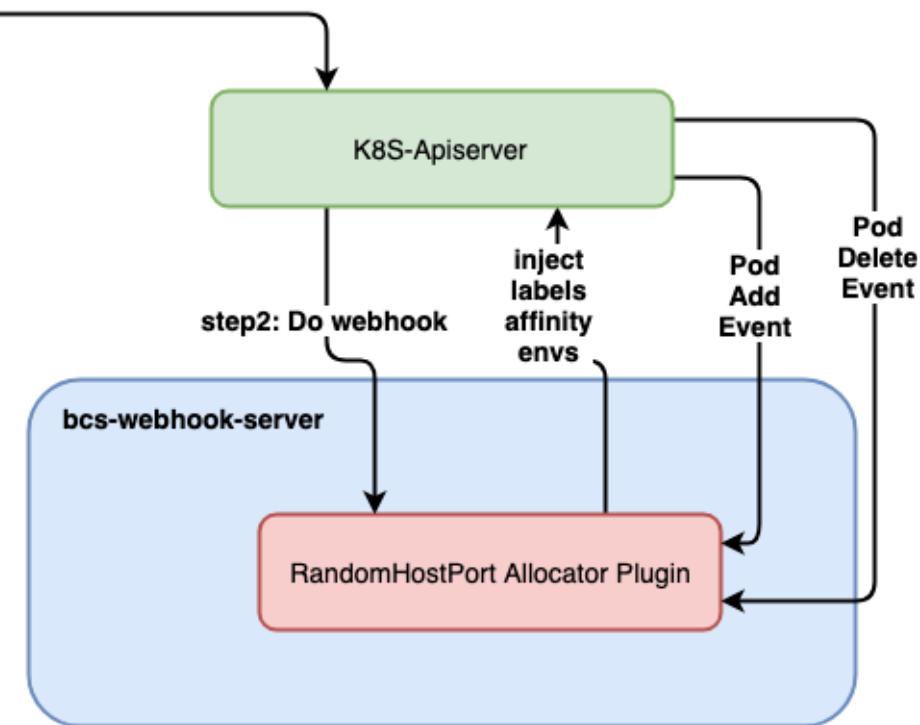
用户最终可以通过以下环境变量获取到注入信息 BCS\_RANDHOSTPORT\_FOR\_CONTAINER\_PORT\_  
BCS\_RANDHOSTPORT\_HOSTIP BCS\_RANDHOSTPORT\_POD\_NAME BCS\_RANDHOSTPORT\_POD\_NAMESPACE

```
yaml apiVersion: v1 kind: Pod metadata: name: nginx-deployment-xxxx-xxxxx metadata: labels: app: nginx # 注入随机hostport相关的label，用于pod亲和性调度，防止调度失败 29001.randhostport.webhook.bkbc.tencent.com: "29001"
29323.randhostport.webhook.bkbc.tencent.com: "29323" annotations: randhostport.webhook.bkbc.tencent.com:
"true" ports.randhostport.webhook.bkbc.tencent.com: "8080,metric" spec: containers: - name: nginx image:
nginx:1.14.2 ports: - containerPort: 8080 hostPort: 29001 - name: metric containerPort: 8081 hostPort: 29323
env: - name: BCS_RANDHOSTPORT_FOR_CONTAINER_PORT_8080 value: "29001" - name:
BCS_RANDHOSTPORT_FOR_CONTAINER_PORT_8081 value: "29323" - name: BCS_RANDHOSTPORT_HOSTIP valueFrom: fieldRef:
apiVersion: v1 fieldPath: status.hostIP - name: BCS_RANDHOSTPORT_POD_NAME valueFrom: fieldRef: apiVersion: v1
fieldPath: metadata.name - name: BCS_RANDHOSTPORT_POD_NAMESPACE valueFrom: fieldRef: apiVersion: v1 fieldPath:
metadata.namespace # 注入pod亲和性 affinity: podAntiAffinity: requiredDuringSchedulingIgnoredDuringExecution: -
labelSelector: matchLabels: 29001.randhostport.webhook.bkbc.tencent.com: "29001" topologyKey:
kubernetes.io/hostname - labelSelector: matchLabels: 29323.randhostport.webhook.bkbc.tencent.com: "29323"
topologyKey: kubernetes.io/hostname
```

## 详细设计

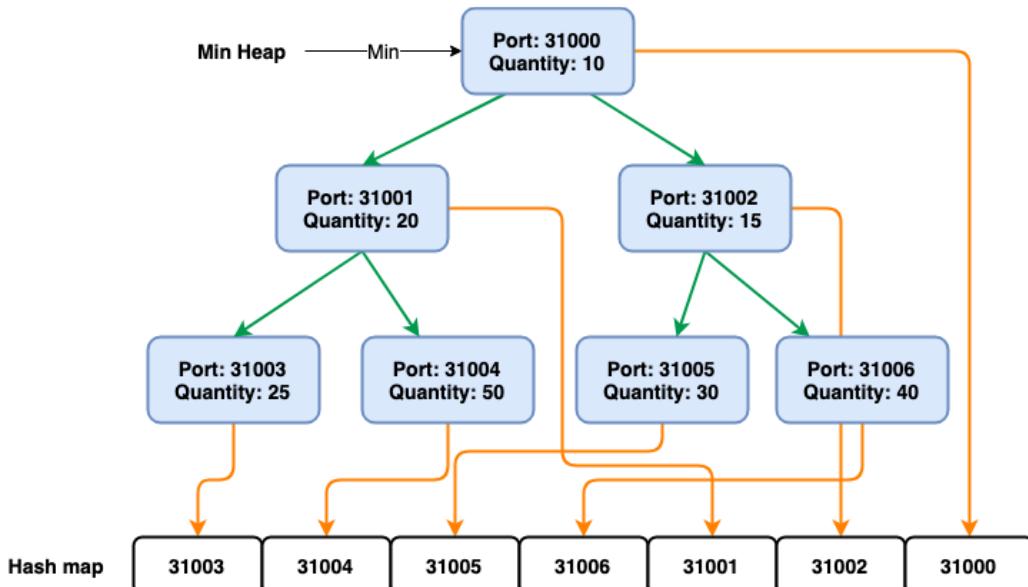
整体架构

### step1: Create Pod With Certain Annotation



- randhostport作为bcs-webhook-server的插件存在，减少了部署的复杂性和对kube-apiserver压力
- randhostport拦截特定Pod的创建，监听Pod的变化
- randhostport维护一个端口资源池

### 随机端口分配策略



- 维护一个针对（端口号，已分配的Pod的数量）映射关系的优先级队列，以已分配的Pod数量为优先级，总是挑选已分配Pod最少的端口号作为本次随机的hostport端口号。
- 每次都Pod增加或者删除时，更新优先级队列中对应端口号的Quantity值。

### 7. NodePort模式下，不同Service之间进行权重配比

### 场景描述

- 命名空间 **test** 下有名为 **svc1** 和 **svc2** 的两个 Service
- service **svc1** 和 **svc2** 中都有端口号为 8080 的 TCP 端口, **svc1** 对应的 NodePort 为 31003, **svc2** 对应的 NodePort 为 31004
- 80% 的流量转给 **svc1**, 20% 的流量转给 **svc2**
- clb 实例的 id 是 **lb-xxxxxx**

### 想要达到的效果

```
text clb(38080 port) ----80%--> service svc1 对应pod所在的node节点(31003 port) -----> svc1 pods(8080) |--20%--> service svc2 对应pod所在的node节点(31004 port) -----> svc2 pod(8080)
```

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test1 annotations: networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx spec: rules: - port: 38080 protocol: TCP services: - serviceName: svc1 serviceNamespace: default servicePort: 8080 # 注意: 这里的权重只是 clb 控制台上 RS 的权重, 最终的流量比例, 需要考虑后端实例的数量 weight: value: 80 - serviceName: svc2 serviceNamespace: default servicePort: 8080 weight: value: 20
```

## 8. 直通模式下, 同 Service 不同 Deployment Pod 之间权重配比

### 场景描述

- 存在名为 **nginx-svc2** 的 service, 同时关联 **nginx-v2-1** 和 **nginx-v2-2** 不同的两个 deployment
- nginx-v2-1** 和 **nginx-v2-2** 对应的 pod template labels 不同

### 想要达到的效果

- nginx-v2-1** 的 pods 在 clb 上对应的 rs ip 权重为 10
- nginx-v2-2** 的 pods 在 clb 上对应的 rs ip 权重为 20

### 配置示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test-tcp namespace: ingress-test annotations: networkextension.bkbcn.tencent.com/lbids: lb-7hxxxx spec: rules: - port: 8080 protocol: TCP services: - serviceName: nginx-v2 serviceNamespace: ingress-test servicePort: 8080 isDirectConnect: true subsets: - labelSelector: version: v2-1 weight: value: 10 - labelSelector: version: v2-2 weight: value: 20
```

## 9. StatefulSet 端口 HTTP 协议一一映射

### 场景描述

- 命名空间 **game** 下有名为 **gameserver** StatefulSet, 有 6 个 Pod
- gameserver** 的每个 Pod 都需要单独直接对外提供 HTTPS 服务
- gameserver** 每个 Pod 的端口号与 Pod 需要有一定对应关系
- gameserver** 拥有 underlay IP 地址或者是 Host 模式, 或者 clb 开通了 Global Router 直通模式
- clb 实例的 id 是 **lb-xxxxxx**

### 想要达到的效果

- 端口映射 **gameserver-0** 对应 30000 端口 **gameserver-1** 对应 30001 端口 ..... **gameserver-5** 对应 30005 端口
- 配置示例 ``yaml
  - startPort: 30000 protocol: HTTPS startIndex: 0 endIndex: 6 workloadKind: StatefulSet workloadName: gameserver workloadNamespace: game certificate: mode: UNIDIRECTIONAL certID: xxxxxxx routes:
    - domain: www.qq.com path: /path2 ``

## 10. 为Pod动态分配云负载均衡器端口资源

### 场景描述

在某些业务场景下，某些服务为了减少延迟，通常直接对客户端对外提供公网IP和端口，然而，服务器直接配置公网IP地址会存在一定安全风险，同时也面临公网IP消耗量大，无法保证高可用等问题。所以各大云厂商都提供了高可用的云负载均衡器，通过介入云负载均衡器能实现防护网络攻击，收敛IP地址，消除单点故障的效果。

在云原生架构下，各大云厂商的都提供了Kubernetes原生LoadBalancer和Ingress对接云负载均衡器的实现，但是并没有提供为单个服务实例暴露公网端口的能力。基于BCS提供的bcs-ingress-controller，用户可以实现为单个Pod分配一个或者一段端口，但是需要提前设计映射规则，在多环境多版本场景下，管理不够便捷直观。期望有一种更加直观的方案，在不需要提前设定映射规则的情况下，使得单个pod能够绑定云负载均衡器端口资源。该方案需要具有以下功能：

- 维护云负载均衡器端口资源池，能够动态地向资源池中增加和删除云负载均衡器
- Pod通过简单的annotation进行端口资源绑定
- Pod中程序需要读取到绑定的负载均衡器信息和端口信息，需要某种机制通知Pod绑定过程是否已完成
- 能够为Pod的同一端口分配多个云负载均衡器，实现多运营商接入
- 为端口分配增加时间窗口，使得同名Pod在重新调度之后能够复用原来的VIP和端口
- 为集群中可用端口资源提供可观测手段

### 操作指引

#### 创建端口池

假设我们已经拥有了一个id为lb-00001的clb实例，我们就可以创建一个端口池资源

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: test-pool1 namespace: test spec: poolItems: # 定义了一个包含31000-31099总共100个端口的端口池Item - itemName: item1 # 端口池item名称，每个item不能重名 loadBalancerIDs: ["lb-00001"] protocol: "TCP" # 端口池的协议，不填则默认为 "TCP,UDP" startPort: 31000 endPort: 31100
```

当我们创建完PortPool之后，PortPool会进行初始化，并完成负载均衡监听器的提前创建，我们需要等待PortPool中item的状态变成Ready之后，才能绑定端口资源。

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: annotations: finalizers: - ingresscontroller.bkbcn.tencent.com name: test-pool1 namespace: test spec: poolItems: - endPort: 31100 itemName: item1 loadBalancerIDs: - lb-00001 startPort: 31000 status: poolItems: - itemName: item1 loadBalancerIDs: - lb-00001 message: Ready poolItemLoadBalancers: - ips: - 1.1.1.1 loadbalancerID: lb-00001 loadbalancerName: test-nj-loadtest-public1 region: ap-nanjing type: OPEN segmentLength: 0 startPort: 31000 endPort: 31100 # 此时表示端口池Item已初始化成功 status: Ready
```

给需要注入端口的POD所在NAMESPACE打上端口注入标签

```
shell kubectl label ns test bcs-ingress-controller-inject=true
```

给POD TEMPLATE增加端口池注解

用户只需要为Pod Template增加两条annotations即可给Pod注入端口池中的端口：

- portpools.networkextension.bkbcn.tencent.com: "true"
- ports.portpools.networkextension.bkbcn.tencent.com: "{端口池名称} {端口协议名称} {端口号}"

```yaml

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx namespace: test labels: app: nginx spec: replicas: 1 selector: matchLabels: app: nginx template: metadata: labels: app: nginx annotations: # 此注解表示Pod需要注入端口池 portpools.networkextension.bkbcn.tencent.com: "true" # 此注解表示Pod需要关联命名空间下的端口池test-pool1，暴露端口 containerPort 80，且为TCP协议 ports.portpools.networkextension.bkbcn.tencent.com: "test-pool1 TCP 80" spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 # 挂载annotation到文件目录/etc/podinfo，此路径用户可以自行定义 volumeMounts: - mountPath: /etc/podinfo name: podinfo volumes: - name: podinfo downwardAPI: defaultMode: 420 items: - fieldRef: apiVersion: v1 fieldPath: metadata.annotations path: annotations ``
```

读取端口信息和端口绑定状态

当Pod成功创建之后，bcs-ingress-controller会自动给Pod注入端口信息，并根据实际端口绑定过程，动态修改Pod的annotation来通知Pod

- poolbindings.portpool.networkextension.bkbcn.tencent.com: 此注解为Pod动态注入的详细端口信息
- status.portpools.networkextension.bkbcn.tencent.com: 此注解为Pod绑定端口的状态，当绑定成功时，会从NotReady变成Ready

```
yaml apiVersion: v1 kind: Pod metadata: annotations: # 注入的端口端口
poolbindings.portpool.networkextension.bkbcn.tencent.com: '[ { "poolName": "test-pool1", "poolNamespace": "test",
"loadBalancerIDs": [ "lb-00001" ], "poolItemLoadBalancers": [ { "loadbalancerName": "test-nj-loadtest-public1",
"loadbalancerID": "lb-00001", "region": "ap-nanjing", "type": "OPEN", "ips": [ "1.1.1.1" ] } ],
"poolItemName": "item1", "protocol": "TCP", "startPort": 31000, "endPort": 0, "rsStartPort": 80 } ]'
portpools.networkextension.bkbcn.tencent.com: "true" ports.portpools.networkextension.bkbcn.tencent.com: test-
pool1 TCP 80 status.portpools.networkextension.bkbcn.tencent.com: Ready labels: app: nginx name: nginx-
5c56b6f598-f6417 namespace: test spec: containers: - env: - name: BCS_PORTPOOL_PORT_VIPLIST_TCP_80 value:
1.1.1.1:31000 image: nginx:1.15 imagePullPolicy: IfNotPresent name: nginx ports: - containerPort: 80 protocol:
TCP volumeMounts: - mountPath: /etc/podinfo name: podinfo volumes: - downwardAPI: defaultMode: 420 items: -
fieldRef: apiVersion: v1 fieldPath: metadata.annotations path: annotations name: podinfo
```

由于我们通过downwardAPI将Pod的annotation挂载成了一个文件，用户应用程序可以通过监听文件的内容来监听pod端口绑定状态的变化

进入容器之后，我们可以看到文件内容如下

```
```shell kubectl exec -it nginx-d47f57c99-6wrzn -n test -- /bin/sh
```

## cat /etc/podinfo/annotations

```
kubernetes.io/config.seen="2021-07-27T10:15:00.995308727+08:00" kubernetes.io/config.source="api"
poolbindings.portpool.networkextension.bkbcn.tencent.com="[{\"poolName\": \"test-pool1\", \"poolNamespace\": \"test\", \"loadBalancerIDs\": [
[\"lb-00001\"]], \"poolItemLoadBalancers\": [{\"loadbalancerName\": \"test-nj-loadtest-public1\", \"loadbalancerID\": \"lb-00001\", \"region\": \"ap-nanjing\", \"type\": \"OPEN\", \"ips\": [
\"1.1.1.1\"]}], \"poolItemName\": \"item1\", \"protocol\": \"TCP\", \"startPort\": 31000, \"endPort\": 0, \"rsStartPort\": 80}]"
portpools.networkextension.bkbcn.tencent.com="true" ports.portpools.networkextension.bkbcn.tencent.com="test-pool1 TCP 80"
status.portpools.networkextension.bkbcn.tencent.com="Ready" ````
```

### 删除端口池

对于已分配端口的端口池，不能直接删除 PortPool。需要先清理掉已使用该端口池的 Workload，等清理完成 Workload 后，再删除 PortPool。

查看已使用端口池的 Pod 命令：

```
kubectl get portbinding --all-namespaces
```

portbinding 名称和命名空间就是使用端口池的 Pod 名称和命名空间。

### 端口池详细配置

注意事项：

- PortPool中可以动态增加和删除item
- item中的loadBalancerIDs一旦定义好不能修改，只能通过删除和新建item来修改
- item中的端口如果已经被使用，则不能被删除，当端口被释放掉之后可以删除
- item中的startPort不能修改；endPort可以变大，但是不能变小
- aws 中不支持一个端口同时监听多种协议，因此需要在 item 中 protocol 字段指定协议，且只能是 TCP 或 UDP。腾讯云可以同时监听两种协议，如果需要创建两种协议的端口池，则不需要填此参数

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: portpool-example1
```

```
namespace: default spec: poolItems: - itemName: item1 loadBalancerIDs: ["lb-test1", "lb-test2"] protocol: TCP
startPort: 30000 endPort: 31000 - itemName: item2 loadBalancerID: ["lb-test3", "lb-test4"] startPort: 30000
endPort: 31000
```

- 端口段形式的端口池声明

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: portpool-example1
namespace: default spec: poolItems: - itemName: item1 loadBalancerIDs: ["lb-test1", "lb-test2"] # 此处声明了1000
个端口，每5个端口一个段，总共可以供200个Pod使用 startPort: 30000 endPort: 31000 segmentLength: 5
```

更多POD注解形式

```
```yaml
```

如果只指定端口池的名字，则默认查找pod所在命名空间的端口池

可用协议[TCP, UDP, TCP_UDP]

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool-sample TCP 8000
```

同时暴露TCP和UDP端口

实际会为Pod分配端口号相同的TCP和UDP两个端口

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool-sample TCP_UDP 8000
```

可以指定端口池的命名空间

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool1.ns1 TCP 8000
```

可以绑定 Pod hostPort，如 8000 为 containerPort，绑定 lb 时则使用 containerPort 对应的 hostPort

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool1.ns1 TCP 8000/hostport
```

可以使用端口名称来索引

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool1.ns1 TCP httpport
```

不指定协议，则默认以container的协议分配端口

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool1.ns1 httpport
```

多个端口

```
ports.portpools.networkextension.bkbcn.tencent.com: |- portpool1.ns1 TCP 8000 portpool1.ns1 UDP 8000 portpool2.ns1 TCP 8080
```

多个端口

```
ports.portpools.networkextension.bkbcn.tencent.com: "portpool1.ns1 TCP 8000;portpool1.ns1 UDP 8000;portpool2.ns1 TCP 8080" ``
```

为POD端口设置保留时间

在某些情况下，Pod名字是固定的，用户希望固定名字的Pod在重建之后依然使用原来的VIP和端口；此时可以通过下面的Annotation来设置VIP和端口保留时间

- "keepduration.portbinding.bkbcn.tencent.com": "5m"

时间字符串是一个十进制数序列，例如“5m”、或“2h45m”。有效时间单位为“ns”、“us”（或“μs”）、“ms”、“s”、“m”、“h”。

为POD增加READINESSGATE

如果在clb端口绑定好之前，需要Pod处于NotReady状态，则需要为Pod增加readinessGate相关的annotations

- "readinessgate.portpools.networkextension.bkbcn.tencent.com": "true"
- 存在此annotations，则会为Pod增加readinessGate

为每个POD增加端口段

如果需要为一个pod暴露一整个端口段，则需要在端口池crd中定义段的长度。注：部署在 AWS 中不支持端口段。

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: test-pool1 namespace: test spec: poolItems: # 定义了一个包含31000-31099总共100个端口的端口池Item - itemName: item1 # 端口池item名称，每个item不能重名 loadBalancerIDs: ["lb-00001"] startPort: 31000 endPort: 31100 # 此处定义为每个Pod segmentLength: 5
```

容器中注入的端口是

```
yaml apiVersion: v1 kind: Pod metadata: annotations: # 注入的端口端口 poolbindings.portpool.networkextension.bkbcn.tencent.com: '[ { "poolName": "test-pool1", "poolNamespace": "test", "loadBalancerIDs": [ "lb-00001" ], "poolItemLoadBalancers": [ { "loadbalancerName": "test-nj-loadtest-public1", "loadbalancerID": "lb-00001", "region": "ap-nanjing", "type": "OPEN", "ips": [ "1.1.1.1" ] } ], "poolItemName": "item1", "protocol": "TCP", "startPort": 31000, "endPort": 31004, "rsStartPort": 80 } ]' portpools.networkextension.bkbcn.tencent.com: "true" ports.portpools.networkextension.bkbcn.tencent.com: test-pool1 TCP 80 status.portpools.networkextension.bkbcn.tencent.com: Ready .....
```

为POD注入自定义字段

在创建端口池时，可以为每个poolItem指定自定义字段（external）
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: test-pool1 namespace: test spec: poolItems: # 定义了一个包含31000-31099总共100个端口的端口池Item - itemName: item1 # 端口池item名称，每个item不能重名 loadBalancerIDs: ["lb-00001"] protocol: "TCP" # 端口池的协议，不填则默认为 "TCP,UDP" startPort: 31000 endPort: 31100 external: "test" #自定义字段 自定义字段会被透传到绑定的pod上，用户可以通过监听文件来使用这个字段（见2.5）。 yaml apiVersion: v1 kind: Pod metadata: annotations: # 注入的端口 poolbindings.portpool.networkextension.bkbcn.tencent.com: '[{ "poolName": "test-pool1", "poolNamespace": "test", "loadBalancerIDs": ["lb-00001"], "poolItemLoadBalancers": [{ "loadbalancerName": "test-nj-loadtest-public1", "loadbalancerID": "lb-00001", "region": "ap-nanjing", "type": "OPEN", "ips": ["1.1.1.1"] }], "poolItemName": "item1", "protocol": "TCP", "startPort": 31000, "endPort": 0, "rsStartPort": 80, "external": "test" # 透传字段 }]' portpools.networkextension.bkbcn.tencent.com: "true"
ports.portpools.networkextension.bkbcn.tencent.com: test-pool1 TCP 80
status.portpools.networkextension.bkbcn.tencent.com: Ready labels: app: nginx name: nginx-5c56b6f598-f6417 namespace: test spec: containers: - env: - name: BCS_PORTPOOL_PORT_VIPLIST_TCP_80 value: 1.1.1.1:31000 image: nginx:1.15 imagePullPolicy: IfNotPresent name: nginx ports: - containerPort: 80 protocol: TCP volumeMounts: - mountPath: /etc/podinfo name: podinfo volumes: - downwardAPI: defaultMode: 420 items: - fieldRef: apiVersion: v1 fieldPath: metadata.annotations path: annotations name: podinfo 自定义字段需要更新时，可以直接修改PortPool.Spec 中对应poolItem的external， ingress会将更新应用到所有相关的pod上。

配置GRAFANA监控

详细metric指标

| 指标类型 | 指标 | 自定义 Label | 单位 | 具体含义 ||--|---|---|---|---|| counter | bcs_network_ingress_controller_cloud_request_total |
errcode(错误码), rpc (函数调用哦) | 次 | controller请求云API的接口次数 || summary |
bcs_network_ingress_controller_cloud_response_time_sum,
bcs_network_ingress_controller_cloud_response_time_count | rpc (函数调用) | sum (秒),count (次) | controller请求云API的接口返回时延概况 || histogram | bkbcn_ingressctrl_lib_request_latency_seconds_bucket | handler (接口handler名字), method (接口方法), status (状态码), system (外部系统名称) | 秒 | controller请求外部接口时延 || counter | bkbcn_ingressctrl_lib_request_total |
handler (接口handler名字), method (接口方法), status (状态码), system (外部系统名称) | 次 | controller请求外部接口次数 || histogram | bkbcn_ingressctrl_api_request_latency_seconds_bucket | handler (接口handler名字), method (接口方法), status (状态码) | 秒 | controller自身接口响应时延 || counter | bkbcn_ingressctrl_api_request_total | handler (接口handler名字), method (接口方法), status (状态码) | 次 | controller自身接口被请求次数 || gauge | cloudloadbalance_backendhealth_backend_status | host (http域名),lbid (clb ID),listenerid (监听器ID),path (http路径),port (端口号),protocol (协议类型),rsip (绑定的后段IP),rsport (后端端口) | 0代表不健康1代表健康2代表未知 | 云负载均衡监听器后端健康状态 || counter | bkbcn_ingressctrl_cache_poolitem_portitem_total |

```
itemname (端口池item名称),poolkey (端口池key),protocol (端口池协议) | 个 | 端口池端口数量|| counter |
bkbcn_ingressctrl_cache_poolitem_portitem_used | itemname (端口池item名称),poolkey (端口池key),protocol (端口池协议) | 个 |
端口池已使用端口数量 |
```

11. 跨域绑定

场景描述

满足 P2P 等游戏业务中，多地同服的场景。客户后端服务集群在广州，客户希望在上海、北京等多地创建 CLB，绑定相同的广州后端服务集群。起到游戏加速、流量收敛的作用，有效保证数据传输质量，降低时延。

前置条件

- 为自己申请的clb实例开启snat pro2.0（即跨域绑定2.0）功能，参考 [开启 snat pro2.0](#)
- 配置Snat IP（选择一个当前CLB所在VPC的子网中的IP），参考 [开启 snat pro2.0](#)
- （非跨域直连没有此需求）如果在跨域情况下需要使用直连模式，确保用户的Pod使用的underlay网络（vpc-cni）

INGRESS 示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: name: test-cross namespace:
ingress annotations: # 跨域绑定需要给跨域的 lb 加上地域，如: ap-chongqing:lb-xxxxxxx
networkextension.bkbcn.tencent.com/lbids: "ap-chongqing:lb-xxxxxxx,ap-shanghai:lb-xxxxxxx" spec: rules: -
port: 38080 protocol: TCP services: - serviceName: nginx-host serviceNamespace: ingress servicePort: 8080
```

端口池 PORTPOOL 示例

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: PortPool metadata: name: test-pool1 namespace:
ingress spec: poolItems: - itemName: item1 loadBalancerIDs: ["ap-chongqing:lb-xxxxxxx", "ap-shanghai:lb-
xxxxxxx"] startPort: 31000 endPort: 31100
```

使用限制

- **SNAT IP的数量限制： 10**
 - 配置多个SNAT IP的场景原理
 - 数量单个CLB的单个规则（四层监听器、七层location），单个 SNAT IP 后单个后端之间的连接数最大是5.5万个，若增加 SNAT IP 或增加后端服务时，连接数等比增加
 - 若某 CLB 只有1个 SNAT IP，CLB 只绑定了1个 IP的一个端口，此时该后端服务最多支持5.5w个连接
 - 若某 CLB 有2个 SNAT IP，CLB后端绑定了10个端口，此时该CLB总的连接数是： $2 \times 10 \times 5.5w = 110w$ 个
- **SNAT IP在删除时，该 IP 上的所有连接会全部断开，请谨慎操作。**

更多参数解释

```
yaml apiVersion: networkextension.bkbcn.tencent.com/v1 kind: Ingress metadata: # ingress的名字 name: bcs-
ingress1 # ingress的命名空间 namespace: test annotations: # 可以通过id和name来关联clb实例，优先匹配id，以下annotation二选一 # 通过此annotation来关联多个clb实例，value的形式为"lb1-xxxx,lb2-xxxx,ap-xxxx:lb3-xxxx"
networkextension.bkbcn.tencent.com/lbids: lb-xxxxxx # 通过此annotation来关联多个clb实例，value的形式为"name1,name2,ap-xxx:xxxx" networkextension.bkbcn.tencent.com/lbnames: name1 spec: # rules是一个数组，每个数组元素定义单个端口到一个或者多个Service的映射 rules: # rule定义的端口号，rule之间的端口号不能冲突 - port: 1111 # rule定义的端口协议，可选有[HTTP, HTTPS, TCP, UDP] protocol: TCP # TCP或者UDP监听器转发属性 # 详情参考链接
https://cloud.tencent.com/document/product/214/30693 listenerAttribute: # 会话保持时间，单位：秒。可选值：30~3600，默认 0，表示不开启。 sessionTime: 100 # 负载均衡策略：可选值：WRR, LEAST_CONN lbPolicy: WRR # 健康检查策略
healthCheck: # 是否开启健康检查，如果不设置该选项，默认开启，采用腾讯云默认值 enabled: true # 健康检查的响应超时时间（仅适用于四层监听器），可选值：2~60，默认值：2，单位：秒。响应超时时间要小于检查间隔时间。 timeout: 2 # 健康检查探测间隔时间，默认值：5，可选值：5~300，单位：秒。 intervalTime: 15 # 健康阀值，默认值：3，表示当连续探测三次健康则表示该转发正常，可选值：2~10，单位：次。 healthNum: 3 # 不健康阀值，默认值：3，表示当连续探测三次不健康则表示该转发异常，可选值：2~10，单位：次。
unHealthNum: 3 # TCP端口或者UDP端口映射的Service的数组 services: # service的名字 - serviceName: test-tcp # service的命名空间的名字 serviceNamespace: default # service定义的服务 port的名字 servicePort: 1111 # 是否直通Pod，默认为 false isDirectConnect: true # 是否使用 hostPort，默认为false hostPort: true # 属于该service的监听器后端的权重（只有负载均衡为WRR时才生效） weight: value: 10 # HTTP端口的映射规则 - port: 2222 protocol: HTTP # 只对HTTP或者HTTPS端口生
```

```
效, 7层协议端口映射规则数组 (port+domain+path确定唯一的规则) layer7Routes: # 7层规则的域名 - domain: www.qq.com # 7层
规则的url path: /test1 # 7层规则的转发详细属性 # 详情参考链接 https://cloud.tencent.com/document/product/214/30691
listenerAttribute: # 会话保持时间, 单位: 秒。可选值: 30~3600, 默认 0, 表示不开启。 sessionTime: 100 # 负载均衡策略: 可选
值: WRR, LEAST_CONN, IP_HASH lbPolicy: LEAST_CONN # 健康检查策略, 如果不设置该选项, 默认开启, 采用腾讯云默认值
healthCheck: # 是否开启健康检查 enabled: true # 健康检查的响应超时时间 (仅适用于四层监听器), 可选值: 2~60, 默认值: 2, 单
位: 秒。响应超时时间要小于检查间隔时间。 timeout: 2 intervalTime: 15 healthNum: 3 unHealthNum: 3 # 健康检查状态码 (仅适
用于HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式)。可选值: 1~31, 默认 31。1 表示探测后返回值 1xx 代表健康, 2 表示返回
2xx 代表健康, 4 表示返回 3xx 代表健康, 8 表示返回 4xx 代表健康, 16 表示返回 5xx 代表健康。若希望多种返回码都可代表健康, 则将
相应的值相加。注意: TCP监听器的HTTP健康检查方式, 只支持指定一种健康检查状态码。 httpCode: 31 # 健康检查路径 (仅适用于
HTTP/HTTPS转发规则、TCP监听器的HTTP健康检查方式)。 httpCheckPath: / # 健康检查方法 (仅适用于HTTP/HTTPS转发规则、TCP监听
器的HTTP健康检查方式), 默认值: HEAD, 可选值HEAD或GET。 httpCheckMethod: HEAD # HTTP或者HTTPS规则映射的后端数组
services: - serviceName: test2 serviceNamespace: default servicePort: 2222 # 属于该service的监听器后端的权重 (只有
负载均衡为WRR时才生效) weight: value: 10
```

常见问题

如果遇到组件无法正常工作的情况下, 通过如下方法来排查问题

1. 查看gameingress-controller组件的状态 `bash # 查看组件运行状态是否异常 kubectl get pod -n bcs-system|grep 'gameingress-controller' kubectl describe pod gameingress-controller-xxx -n bcs-system`

1. 查看gameingress-controller组件日志, 常见问题如下:

```
```bash
```

## 确认是否存在错误日志

```
kubectl logs gameingress-controller-xxx -n bcs-system ``
```

2. 日志显示 is not add to global route 通常为 CLB 和集群不在一个 vpc, 使用和集群相同 vpc 的 CLB 即可。
3. 日志显示 has no permission 表示部署 bcs-ingress-controller 使用的 secret id/ secret key 账号没有相关权限, 可以参考 Ingress-controller 替换secret id & secret key 流程 查看需要的权限。通常给 secret id 分配权限后, 不需要重启 bcs-ingress-controller 和 ingress 实例, 等同步完就可以了。
4. bcs-ingress-controller 启动失败 describe controller的pod显示错误如下: Message: OCI runtime create failed: container\_linux.go:380: starting container process caused: process\_linux.go:545: container init caused: setenv: invalid argument: unknown 原因: secretid和secretkey没有进行base64, 参考 Ingress-controller 替换secret id & secret key 流程 替换
5. 腾讯云公有云端口号创建失败问题 CreateListener failed, err [TencentCloudSDKError] Code=FailedOperation, Message=uin not allow to create port range listener.需要给腾讯云账户添加端口号白名单, 请联系腾讯云助手添加
6. 腾讯云公有云直通 Pod 绑定失败 BatchRegisterTargets failed, err [TencentCloudSDKError] Code=InvalidParameterValue, Message=Account {0} is not network shift up, do not support to register other ip 需要账户为带宽上移用户, 请联系腾讯云助手确认
7. 后端 ip 已经绑定到 clb 上, 但是用 clb vip 不能访问
8. 检查 pod 是否正常访问, 使用 pod ip 能否正常访问
9. 检查 cvm 安全组策略, 是否放通 clb → cvm, 参考 <https://cloud.tencent.com/document/product/214/14733> 设置
10. 健康检查正常, 但 clb vip 不能访问后端服务
11. 检查 clb 安全组, 查看是否放通客户端 ip
12. 检查 cvm 安全组策略, 是否放通 clb → cvm, 参考 <https://cloud.tencent.com/document/product/214/14733> 设置
13. 镜像拉取失败 查看组件库 values.yaml 镜像相关参数是否填写正确, 镜像版本参考 bcs-ingress-controller Changelog chart 在 v1.26.0-alpha.1 以上需要在 values.yaml 中填写 registry。
14. 日志显示 invalid header field value "TC3-HMAC-SHA256 Credential....." 原因: 腾讯云 secret id& secret key 没配置正确, 按照 Ingress-controller 替换secret id & secret key 流程 更改即可。
15. 重新安装后, 有些 crd 没创建 原因: 可能是卸载和安装间隔太快, 还没卸载干净 解决方法: 可以再尝试重装一下
16. 日志显示 loadbalancer not found 原因: 部署 bcs-ingress-controller 时 values 填的 lb 地域和当前 lb 地域不一致 解决方法: 方法一: 修改 values 中 tencentcloudRegion 字段为 lb 的地域, 地域列表, 修改完更新组件库即可 方法二: 修改 ingress 或 portpool 中声明的 lb id, 格式改为 ap-xxx:lb-xxx

17. Not support to register other ip ['xxx.xxx.xxx.xxx'] before enable SnatPro

原因：clb 和集群不在一个地域

解决方法：1. 需要使用与集群相同的 VPC 和相同的地域的 lb，或者开启 SnatPro 开启跨域绑定 2.0，参考 BCS Ingress Controller 跨地域绑定指引

- pod 创建等待时间很长， webhook 超时

原因：在 webhook 请求前，有端口池正在创建中，持有了整个端口池的锁，需要等待端口池创建完成，释放了锁， webhook 才能继续执行。

- workload 声明了端口池，但 pod 没有分配 clb 端口 原因可能有两个：

- namespace 没有打 label，需要打上 bcs-ingress-controller label

```
kubectl label ns {namespace} bcs-ingress-controller-inject=true
```

- portpool 端口声明 annotation 位置有问题，需要放在 workload template 里面，如：

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx namespace: mars labels: app: nginx spec:
replicas: 1 selector: matchLabels: app: nginx template: metadata: annotations: # 此注解表示Pod需要注入端口池
portpools.networkextension.bkbcst.tencent.com: "true" # 此注解表示Pod需要关联命名空间下的端口池test-pool1，暴露
端口containerPort 80，且为TCP协议 ports.portpools.networkextension.bkbcst.tencent.com: "test-pool1 TCP 80"
```

## ConfigMap 列表

ConfigMap 列表用于展示集群中存储的非敏感数据，并且允许通过页面进行数据的更新和删除操作。

### 列表展示

ConfigMaps								帮助	
批量删除		所属集群		命名空间	来源	创建时间	更新时间	更新人	操作
<input type="checkbox"/>	名称								
<input type="checkbox"/>	test-cfg	新测试集群	bcs-test	模板集	2020-12-23 10:52:29	2020-12-23 10:52:28	admin	<a href="#">更新</a> <a href="#">删除</a>	
<input type="checkbox"/>	ingress-controller-leader-nginx	新测试集群	bcs-system	Client	2020-12-09 17:48:44	2020-12-09 17:48:44	--	<a href="#">更新</a> <a href="#">删除</a>	

### 详情展示

ConfigMaps								帮助
批量删除		所属集群		命名空间	来源	操作	明文显示	帮助
<input type="checkbox"/>	名称							
<input checked="" type="checkbox"/>	test-cfg	新测试集群	bcs-test	模板集	2020-12-23 10:52:29	2020-12-23 10:52:28	admin	<a href="#">更新</a> <a href="#">删除</a>

键	值	明文显示
data	*****	

[显示摘要](#)

### 更新操作

注意：仅允许操作通过表单模板集部署的 ConfigMap 资源

ConfigMaps								帮助	
批量删除		所属集群		命名空间	来源	创建时间	更新时间	更新人	操作
<input type="checkbox"/>	名称								
<input checked="" type="checkbox"/>	test-cfg	新测试集群	bcs-test	模板集	2020-12-23 10:52:29	2020-12-23 10:52:28	admin	<a href="#">更新</a> <a href="#">删除</a>	
<input type="checkbox"/>	ingress-controller-leader-nginx	新测试集群	bcs-system	Client	2020-12-09 17:48:44	2020-12-09 17:48:44	--	<a href="#">更新</a> <a href="#">删除</a>	

ConfigMaps 请通过模板集或Helm创建ConfigMap

批量删除

名称	所属集群	命名空间
test-cfg	新测试集群	bcs-test
ingress-controller-leader-nginx	新测试集群	bcs-system
prometheus-po-prometheus-operator-prometheus-nodes-0	新测试集群	thanos
kubedns-autoscaler	新测试集群	kube-system
kube-dns	新测试集群	kube-system

更新test-cfg

名称: test-cfg

键: data

值: update\_test

保存 取消

## 删除操作

ConfigMaps 请通过模板集或Helm创建ConfigMap

批量删除

名称	所属集群	命名空间	来源	创建时间	更新时间	更新人	操作
test-cfg	新测试集群	bcs-test	模板集	2020-12-23 10:52:29	2020-12-23 10:52:28	admin	更新 <b>删除</b>

## Secret 列表

Secret 列表用于展示集群中存储的敏感数据，并且允许通过页面进行数据的更新和删除操作。

### 列表展示

Secrets 请通过模板集或Helm创建Secret

批量删除

名称	所属集群	命名空间	来源	创建时间	更新时间	更新人	操作
sh.helm.release.v1.rumpetrol-2012240854.v1	新测试集群	bcs-test	Client	2020-12-24 20:54:50	2020-12-24 20:54:50	---	更新 <b>删除</b>
paas.image.registry.test4proj	新测试集群	test4proj	Client	2020-12-23 16:11:18	2020-12-23 16:11:18	---	更新 <b>删除</b>
default-token-pbjg2	新测试集群	test4proj	Client	2020-12-23 16:11:17	2020-12-23 16:11:17	---	更新 <b>删除</b>
test-secret	新测试集群	bcs-test	模板集	2020-12-23 10:52:29	2020-12-23 10:52:28	admin	更新 <b>删除</b>

### 详情展示

Secrets 请通过模板集或Helm创建Secret

批量删除

名称	所属集群	命名空间	来源	创建时间
sh.helm.release.v1.rumpetrol-2012240854.v1	新测试集群	bcs-test	Client	2020-12-24 20:54:50
paas.image.registry.test4proj	新测试集群	test4proj	Client	2020-12-23 16:11:18
default-token-pbjg2	新测试集群	test4proj	Client	2020-12-23 16:11:17
<b>test-secret</b>	新测试集群	bcs-test	模板集	2020-12-23 10:52:29

test-secret

键	值	明文显示
data	*****	<b>明文显示</b>

显示摘要

io.tencent.bcs.app.appid	2	io.tencent.bcs.cluster	BCS-K8S-40001
io.tencent.bcs.clusterid	BCS-K8S-40001	io.tencent.bcs.kind	Kubernetes
io.tencent.bcs.namespace	bcs-test	io.tencent.bkdata.baseall.dataid	6566
io.tencent.bkdata.container.stderr.log.dataid	0	io.tencent.paa.instanceid	7
io.tencent.paa.projectid	42261ad6e6b74a40a5fcba519141f933	io.tencent.paa.templateid	7
io.tencent.paa.source_type	template	io.tencent.paa.templatedid	7
io.tencent.paa.version	v1		

### 更新操作

注意：仅允许操作通过表单模板集部署的 secret 资源

The screenshot shows two views of the Harbor Secrets management interface. The top view is a list of secrets in the 'new test cluster' namespace, including 'sh.helm.release.v1.rumpetroll-2012240854.v1', 'paas.image.registry.test4proj', 'default-token-pbjg2', and 'test-secret'. The bottom view is a detailed edit screen for 'test-secret', showing a 'data' key with the value 'ceshi'. A note at the bottom says '实例化时会将值的内容做base64编码'.

## 删除操作

This screenshot shows the same list of secrets from the previous view, but with the 'test-secret' row selected. The '操作' (Operation) column for 'test-secret' now contains '更新' (Update) and '删除' (Delete), with '删除' highlighted by a red box.

## 仓库

容器服务采用 Harbor 做镜像仓库。仓库分为公共镜像和项目镜像，公共镜像可以被所有项目共享，项目镜像只有当前项目有权限访问。

This screenshot shows the Harbor Registry interface under the '仓库' (Repository) section. The left sidebar has '公共镜像' (Public Registry) selected. The main area displays a list of public images, each with a preview icon, name, type, source, and a '详情' (Details) link. The images listed are 'public/bcs/k8s/bcs-kube-agent', 'public/bcs/k8s/bcs/datwatch', 'public/bcs/k8s/cluster-proportional-autoscaler-amd64', 'public/bcs/k8s/defaultbackend', 'public/bcs/k8s/etcd-gateway', and 'public/bcs/k8s/etcd-sidecar'.

## Harbor 镜像仓库简介

Harbor 是 VMware 公司开源的企业级 Docker Registry 管理项目。在 Harbor 仓库中，用户和仓库都是基于项目进行组织，用户基于项目可以拥有不同的权限。

Harbor 项目类型分为公共仓库和私有仓库两种类型，其中

- 公共仓库：任何使用者都可以获取这个仓库中的镜像。
- 私有仓库：只有被授予权限的用户可以获取这个仓库中的镜像。

更多关于 Harbor 项目和用户的使用指引，请参考 [Harbor 的用户手册](#)。

## 容器服务中的 Harbor 仓库

### 获取项目仓库账号

选中【Helm】菜单中的【Charts 仓库】，点击右上角【如何推送 Helm Chart 到项目仓库】指引，可以获取仓库信息。

The screenshot shows the Harbor Helm Chart Repository interface. On the left, there's a sidebar with options like '容器服务', '集群', '节点', '命名空间', '模板集', '变量管理', 'Helm', and 'Chart仓库'. Under 'Helm', 'Release列表' and 'Chart仓库' are visible. The main area has tabs for '同步仓库' (Sync Repository) and '公共仓库' (Public Repository). A modal window titled '如何推送 Helm Chart 到项目仓库' (How to Push Helm Chart to Project Repository) is open. It contains three steps: 1. 下载地址 (Download Address), which shows the URL: http://[harbor-api.service.consul]:chartrepo/demoproj1 --username=[REDACTED] --password=[REDACTED]; 2. 添加 Helm Chart 仓库 (Add Helm Chart Repository), with a note about private repositories; 3. 推送 Helm Chart (Push Helm Chart), with a command example: helm push rumpetroll/ demoproj1. Below the modal, there are sections for '准备 Chart' (Prepare Chart) and '推送 Chart' (Push Chart).

具体包含以下内容：  
 - 镜像仓库地址，下文用 `<registry_url>` 表示(注意：地址包含端口)。`<registry_url>` 实际是服务上(如集群中的 Node 节点)配置的内网 consul 域名，如果需要浏览器或者本地终端访问，可以本地配置 hosts，如 `10.0.0.1 <registry_url>`，其中 `10.0.0.1` 是 `<registry_url>` 后端服务器上的 IP(如外网 IP，保证本地可访问)  
 - 镜像仓库用户名和密码 (Harbor 中的一个项目账号，可以上传 镜像 和 Charts)  
 - 项目 code，下文用 `<project_code>` 表示，如上图中的 `demoproj1`

### 登录镜像仓库

```
bash docker login --username=<username> <registry_url> Password: Login Succeeded 注意: 镜像仓库地址包含端口。
```

### 推送镜像

推送镜像之前，要先将镜像 tag 为满足 Harbor 项目前缀要求的格式。

例如需要将本地镜像 `nginx:1.18.0` 推送到当前项目下，首先将镜像按下面的命令重新 tag。

```
bash docker tag nginx:1.18.0 <registry_url>/<project_code>/nginx:1.18.0 docker push <registry_url>/<project_code>/nginx:1.18.0
```

### 拉取镜像

拉取项目镜像前需先完成登录，拉取公共镜像可以不登录。

```
bash docker pull <registry_url>/<project_code>/nginx:1.18.0
```

## 登录 Harbor Web 仓库

### 获取 Harbor 账号

- 获取项目仓库账号

在【Chart 仓库】菜单下，点击【查看项目 Chart 仓库配置信息】后，可以获取 Harbor 的项目账号，访问项目（BCS 中新建的项目）仓库。

- 获取 Harbor 管理员账号

在[常用环境变量](#)中可以找到 Harbor 的用户名 `HARBOR_SERVER_ADMIN_USER` 以及密码 `HARBOR_SERVER_ADMIN_PASS`。

## 访问 Harbor Web 端

- 项目仓库账号视角

通过浏览器可以访问地址 `<registry_url>`，进入 Harbor 管理页面，从中可以看到项目的访问级别有“公开”和“私有”。

The screenshot shows the Harbor project management interface. At the top right, there are language and user information settings. Below the header, a sidebar on the left has '项目' and '日志' options. The main content area is titled '项目' (Projects) and displays a table of four projects:

项目名称	访问级别	角色	镜像仓库数	Helm Chart 数目	创建时间
library	公开		0	0	2019/5/23 上午11:50
public	公开		38	2	2019/5/23 上午11:57
joyfulgame	私有	开发人员	2	8	2019/8/3 下午8:49
goharbor	公开		1	0	2019/8/10 下午2:03

At the bottom right of the table, it says '1-4 共计 4 条记录'.

在上图中点击项目仓库 `joyfulgame`，可以看到有 2 个镜像，其中 `joyfulgame` 是 `<project_code>`。

This screenshot shows the 'joyfulgame' project details page. The top navigation bar includes '中文简体' and '1564969588969835'. The sidebar on the left shows '项目' and '日志'. The main content area is titled 'joyfulgame' and '镜像仓库' (Image Repository). It lists two images:

名称	标签数	下载数
joyfulgame/cmdb-standalone	1	1
joyfulgame/nginx	1	1

At the bottom right, it says '1-2 共计 2 条记录'.

点击【Helm Charts】页面，可以看到仓库中上传的 Charts。

This screenshot shows the 'joyfulgame' project's Helm Charts page. The top navigation bar includes '中文简体' and '1564969588969835'. The sidebar on the left shows '项目' and '日志'. The main content area is titled 'joyfulgame' and 'Helm Charts'. It lists eight charts:

名称	状态	版本	创建时间
blueking-ingress	正常	1	2019年8月10日
gitlab	正常	1	2019年8月29日
gitlab-ce	废弃	1	2019年9月2日
jenkins	正常	1	2019年8月28日
nfs-client-provisioner	正常	1	2019年9月2日
rancher	正常	1	2019年8月29日
rumpetroll	正常	1	2019年8月10日
wordpress	正常	1	2019年8月28日

At the bottom right, it says '1-8 共计 8 条记录'.

- Harbor 管理员账号视角

使用管理员账号，可以实现用户管理、仓库管理、复制管理、配置管理。

## 操作审计

操作审计功能主要是记录用户在使用容器服务时的一些关键操作和结果，方便回溯问题。

## 事件查询

容器服务提供了收集 kubelet 上报事件的能力。通过【事件查询】，用户可以查询项目下集群的 kubelet 上报的部分事件，例如拉取镜像成功或失败等。

## WebConsole 说明

### WebConsole 简介

WebConsole 是容器服务提供快捷查看集群状态的命令行服务。

kubectl 是 K8S 官方的命令行工具，用于管理 K8S 集群，用户添加完节点，部署完 Deployments, Helm 等，都可以通过 WebConsole 内的 kubectl 命令工具查看节点，Deployment 等信息。

## WebConsole 使用

进入容器服务，在任意页面右下角，选择对应集群进入 WebConsole。

The screenshot shows the BlueKing Container Management Platform interface. On the left is a sidebar with various navigation options: 集群 (Cluster), 节点 (Nodes), 命名空间 (Namespaces), 模板集 (Template Sets), 变量管理 (Variable Management), Helm, 应用 (Applications), 网络 (Networks), 资源 (Resources), 仓库 (Repositories), 操作审计 (Audit), 事件查询 (Event Query), and 监控中心 (Monitoring Center). The main area displays resource usage statistics for the 'weixin' cluster, specifically CPU usage at 18.75%, memory usage at 8.48%, and disk usage at 25.01%. A large plus sign icon indicates the option to '新建集群' (Create New Cluster). At the bottom right, there is a 'weixin' button and a 'WebConsole' button.

大概 1 秒钟打开 WebConsole，背后实际是启动了一个 WebConsole 的 Pod。

在 WebConsole 中可以使用 kubectl 命令操作集群。

The screenshot shows a terminal window titled 'weixin'. The user has run the command 'kubectl get all -n web-console'. The output shows a table of pods:

NAME	READY	STATUS	RESTARTS	AGE
pod/kubectld-bcs-k8s-40015-ublueking-71ece8db46f7	1/1	Running	0	7m33s

Below this, the user runs 'kubectl get deploy -n dev'. The output shows a table of deployments:

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
bk-bcs-test	1	1	1	1	9h
blueking-inginx-ingress-controller	2	2	2	2	25d
blueking-inginx-ingress-default-backend	1	1	1	1	25d
nfs-client-provisioner-1908281037	1	1	1	1	7d9h
rancher-1908290432	3	3	3	3	6d3h
web-inginx-blue	2	2	2	0	19d
web-inginx-green	2	2	2	0	8d

## kubectl 常用命令介绍

### 查询集群信息

```
bash kubectl cluster-info
```

### Kubectl Get

获取 K8S 集群资源列表，包含 Nodes、Namespaces、Pods、Deployment、Services 等。

- 查询 Node 节点信息

```
bash kubectl get nodes
```

- 获取 NameSpace 信息

```
bash kubectl get namespace
```

- 查询 Pods 列表

```
bash kubectl get pods
```

- 以 JSON 格式输出 Pod 的详细信息

```
bash kubectl get pod <podname> -o json
```

- 查询打印 DEBUG 信息

```
bash kubectl get pods -v=11
```

## Kubectl Describe

获取集群资源的详情，在排查故障的非常有用，例如某一个 Pod 无法启动。

```
bash kubectl describe pod <podname>
```

## Help 查看帮助

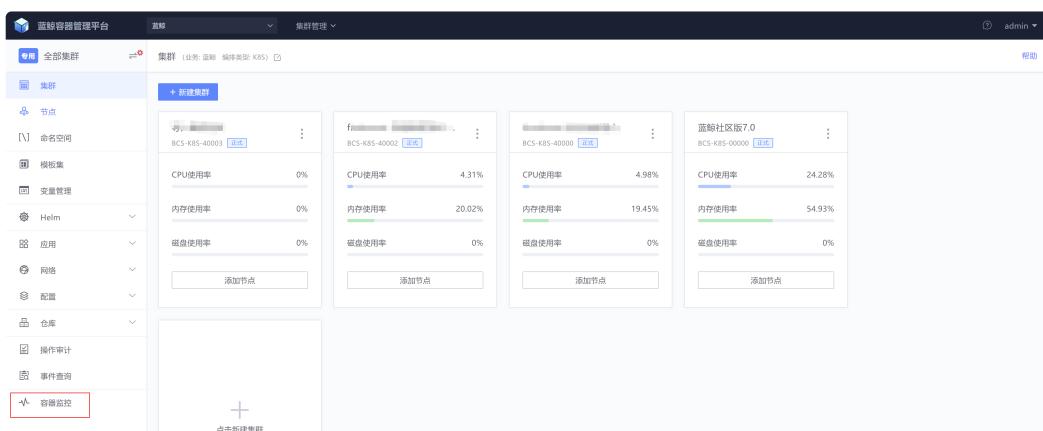
`bash kubectl --help` kubectl 的帮助信息、示例相当详细，而且简单易懂。

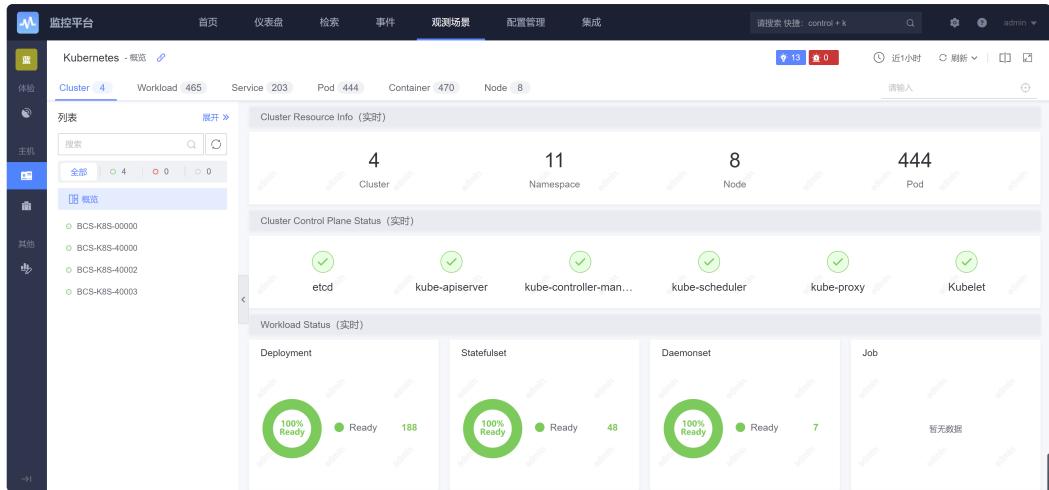
更多 kubectl 使用说明请参照 [Kubectl 操作手册](#)。

# 容器监控

## 概述

蓝鲸7.0的容器监控已从prometheus切换到了蓝鲸容器监控





蓝鲸容器监控使用依赖两个组件bcs-k8s-watch与bkmonitor-operator，否则会没有数据

## bcs-k8s-watch组件安装

- 选择其中一台master角色服务器，如果master服务器上没有安装helm，可以执行下面命令安装，也可以自己去网上下载

```
 wget https://bkopen-1252002024.file.myqcloud.com/ce7/tools/helm && chmod +x helm && mv helm /usr/bin/
```

- 添加chart包镜像仓库 `shell helm repo add blueking https://hub.bktencent.com/chartrepo/blueking`
- 如果命名空间bcs-system不存在，则需要创建命名空间

```
kubectl create ns bcs-system
```

- bcs-gateway绑定hosts，解决bcs网关访问问题，可以把域名bcs-api-gateway绑定到集群“蓝鲸7.0”任意一台node上，这个node最好是master角色

```
kubectl edit cm coredns -n kube-system
```

添加以下内容：

```
hosts { 1.1.1.1 bcs-api-gateway fallthrough } # 1.1.1.1 是集群“蓝鲸7.0”任意一台node，最好是master
```

- 
- 创建bcs-k8s-watch所需证书

把以下内容保存到集群master服务器上，文件名为：bcs-client-bcs-k8s-watch.yaml

```
apiVersion: v1
data:
 ca.crt: tls.crt
 tls.key: tls.key
kind: Secret
metadata:
 name: bcs-client-bcs-k8s-watch
namespace: bcs-system
type: kubernetes.io/tls
```

通过webconsole或ssh到集群“蓝鲸7.0”

The screenshot shows the BlueKing Cluster Management Platform. On the left, there's a sidebar with various management options like Nodes, Namespaces, Templates, Helm, Applications, Networks, Configurations, Warehouses, Audit, Events, and Monitoring. The main area displays three existing clusters: BCS-K8S-40002, BCS-K8S-40001, and BCS-K8S-40000. Each cluster card shows resource usage (CPU, Memory, Disk) and a 'Add Node' button. To the right, there's a section for creating a new cluster, with a text input field for 'Cluster ID' containing 'rodomel-独立集群测试-01' and a red-bordered button labeled 'BlueCommunity v7.0'. A 'WebConsole' link is also present.

```
shell # 在集群中执行如下命令 kubectl get secret bcs-gateway-bcs-services-stack -n bcs-system -o yaml
```

把里面的ca.crt、tls.crt、tls.key里面的内容填充到bcs-client-bcs-k8s-watch.yaml里

执行 `kubectl apply -f bcs-client-bcs-k8s-watch.yaml` 创建好证书

- 通过helm chart安装bcs-k8s-watch

把以下内容保存文件为bcs-k8s-watch-values.yaml，把{集群ID}替换为目前你操作的集群ID，例如：BCS-K8S-40000

```
global: serviceMonitor: enabled: false env: BK_BCS_clusterId: {集群ID} telnet: registry: "hub.bktencent.com"
repository: blueking/bcs-telnet tag: "v1.21.1" storage: zookeeper: endpoints: ["bcs-api-gateway:31746"] image:
registry: hub.bktencent.com repository: blueking/bcs-k8s-watch tag: v1.26.0-alpha.1 env: BK_BCS_customStorage:
"https://bcs-api-gateway:31024"
```

执行以下命令安装bcs-k8s-watch

```
``` helm repo update helm upgrade --install bcs-k8s-watch blueking/bcs-k8s-watch -f ./bcs-k8s-watch-values.yaml -n bcs-system --devel
Release "bcs-k8s-watch" does not exist. Installing it now. NAME: bcs-k8s-watch LAST DEPLOYED: Thu Apr 14 18:58:24 2022
NAMESPACE: bcs-system STATUS: deployed REVISION: 1 TEST SUITE: None
```

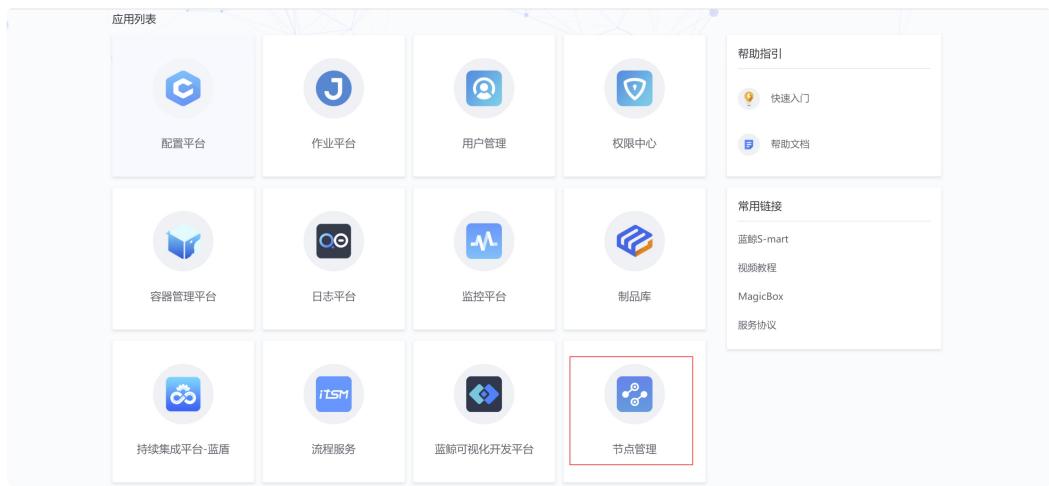
```
kubectl get pod -n bcs-system|grep bcs-k8s-watch bcs-k8s-watch-58ccdb4894-9fwbv 1/1 Running 0 25s
```

```
kubectl logs bcs-k8s-watch-58ccdb4894-9fwbv -n bcs-system ```
```

`## gse_agent`安装

容器监控依赖`gse_agent`做数据管道，在部署容器监控后要安装好`gse_agent`，如果集群master与node都已经安装好`gse_agent`，可以忽略此步骤

判断节点上是否安装`gse_agent`命令：`ps -ef|grep gse_agent`



BlueKing Node Management

- Agent Management**: Shows a list of agents installed on various hosts, including their status, version, and deployment method.
- 普通安装**: Shows a configuration form for installing an Agent via Remote Installation or Manual Installation. It includes fields for IP address, operating system, port, account, password, and advanced options like password expiration and file transfer speed.

BlueKing Node Management

- Agent Management**: Shows a list of agents installed on various hosts, including their status, version, and deployment method.
- 普通安装**: Shows a configuration form for installing an Agent via Remote Installation or Manual Installation. It includes fields for IP address, operating system, port, account, password, and advanced options like password expiration and file transfer speed.

bkmonitor-operator 安装

bkmonitor-operator 使用 helm chart 安装，首先把以下内容保存为文件 bkmonitor-operator-values.yaml

```
bkmonitor-operator-charts: bkmonitor-operator: image: registry: hub.bktencent.com repository: blueking/bkmonitor-operator targetNamespaces: - default - kube-system - blueking - bcs-system bkmonitorbeat: bkmonitorbeatReloader: image: registry: hub.bktencent.com repository: blueking/bkmonitorbeat-reloader image: registry: hub.bktencent.com repository: blueking/bkmonitorbeat bkmonitorbeat-event: bkmonitorbeatReloader: image: registry: hub.bktencent.com repository: blueking/bkmonitorbeat-reloader image: registry: hub.bktencent.com repository: blueking/bkmonitorbeat enabled: true
```

执行以下安装命令：

...

如果不存在bkmonitor-operator命名空间则创建bkmonitor-operator命名空间

```
kubectl create ns bkmonitor-operator
```

```
helm repo update
helm upgrade --install bkmonitor-operator-stack blueking/bkmonitor-operator-stack --version=3.5.98 -f ./bkmonitor-operator-values.yaml -n bkmonitor-operator
Release "bkmonitor-operator-stack" does not exist. Installing it now.
NAME: bkmonitor-operator-stack
LAST DEPLOYED: Thu Apr 14 20:18:00 2022
NAMESPACE: bkmonitor-operator
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES: This chart包安装内容及检查方式如下:
```

1.bkmonitor-operator 检查deployment下是否存在bkmonitor-operator实例 kubectl get deployments -n bkmonitor-operator -o wide -l app.kubernetes.io/name=bkmonitor-operator kubectl get pods -n bkmonitor-operator -o wide -l app.kubernetes.io/name=bkmonitor-operator

2.bkmonitorbeat 检查daemonset下是否存在bkmonitorbeat实例 kubectl get daemonsets -n bkmonitor-operator -o wide -l app.kubernetes.io/name=bkmonitorbeat kubectl get pods -n bkmonitor-operator -o wide -l app.kubernetes.io/name=bkmonitorbeat

3.检查采集配置是否正确下发 登录bkmonitorbeat其中一个pod，检查是否正确下发了子任务配置文件 kubectl exec {bkmonitorbeat的pod名} -n bkmonitor-operator -- ls /data/bkmonitorbeat/config/child_configs

4.检查各pod是否在持续报错 kubectl logs {pod名} {container名} -n bkmonitor-operator ``

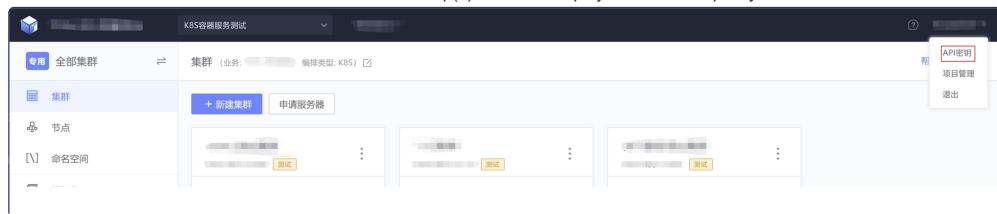
因为目前bkmonitor-operator是使用轮询的方式生效，预计可以看到数据上报时间为10-20分钟，后续会在这方面有所优化

BCS API密钥使用场景

BCS API密钥是调用BCS API的身份凭证，主要有以下使用场景：
1. kubectl的kubeconfig文件 K8S原生API调用 * BCS API调用（开发中，待上线）

BCS API密钥使用流程

1. BCS平台创建API密钥 通过菜单按钮或直接通过URL: <http://bcs.xxxxxx/project-en-name/api-key> 进入到API密钥管理页面



点击“新建API密钥”按钮

The screenshot shows the 'Create New API Key' form. At the top left is a back arrow and the text 'API密钥'. At the top right is a '使用说明' (Usage Instructions) link. The instructions list: 1. API密钥适用于BCS API调用与 kubeconfig。2. API密钥与个人账户绑定，使用蓝鲸权限中心做权限控制，点击 权限中心 可以查看与设置您的API密钥权限。3. 为了您的应用安全，请妥善保存API密钥，请勿通过任何方式上传或者分享您的API密钥信息。 Below the instructions is a table with columns: '用户名' (Username), 'API密钥' (API Key), '过期时间' (Expiration Time), '状态' (Status), and '操作' (Operations). A note at the bottom says '您暂无当前操作权限，请新建API密钥后继续操作'. A red box highlights the '+ 新建API密钥' (Create New API Key) button at the bottom center.

选择申请期限，如果长期使用选择“永久”，如果是临时使用选择您要使用的时间期限，到期后则密钥无法使用

新建API密钥

申请期限

永久	30天	90天	180天	365天	自定义
----	-----	-----	------	------	-----

确定

取消

密钥创建成功

用户名	API密钥	过期时间	状态	操作
*****	-----	永久	正常	编辑 删除

2. 蓝鲸权限中心申请权限 点击“权限中心”超链接

API密钥

1. API密钥适用于 BCS API 调用与 kubeconfig
2. API密钥与个人账户绑定，使用蓝鲸权限中心做权限控制，点击[权限中心](#)可以查看与设置您的API密钥权限
3. 为了您的应用安全，请妥善保存API密钥，请勿通过任何方式上传或者分享您的API密钥信息

点击“申请权限”按钮，点击“申请自定义权限”

我的权限

申请权限 批量续期 权限交接

用户名	描述	所属分权限员	加入用户组的时间	加入方式	到期时间	操作
*****	*****	超级管理员	2021-10-28 10:36:26	直接加入	永久	退出
*****	*****	超级管理员	2021-11-01 17:46:58	直接加入	已过期	退出

共计 2 条 每页 10 条

申请加入用户组

选择用户组 * 输入用户名、描述、所属系统、分权限员等进行搜索
如果以下用户组不满足您的实际需求，可以[申请自定义权限](#)

用户名	描述	所属分权限员
*****	*****	超级管理员
*****	*****	超级管理员
*****	*****	超级管理员

选择“容器管理平台”，如果只需要查看集群（只读），推荐权限选择“集群资源查看”，如果要操作集群，推荐权限选择“集群资源管理”（读写），如果要申请更细粒度的权限，请展开项目标签进行选择

申请自定义权限

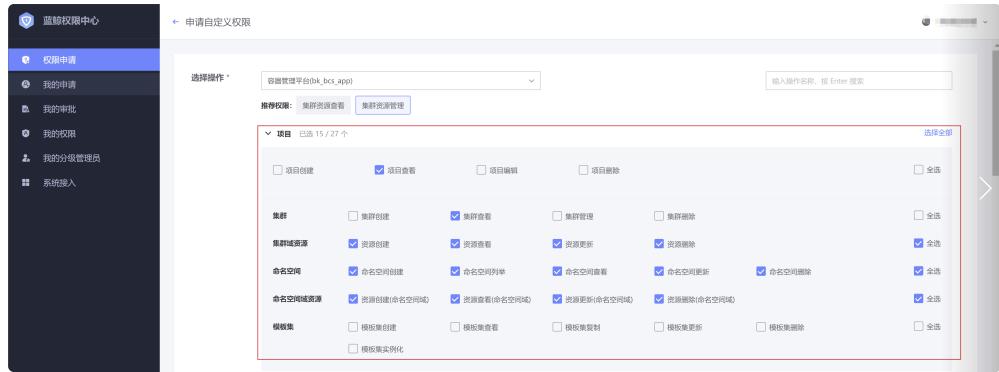
选择操作 * 容器管理平台(bk_bcs_app)

推荐权限： 集群资源查看 集群资源管理

项目 已选 0 / 27 个

未启用 已选 0 / 2 个

选择全部 取消全部



3. 选择“关联资源实例”，如果要精细化申请实例权限，可以编辑每一个资源操作项，如果实例太多，可以开启“批量编辑”，申请整个项目的权限，提交权限申请

关联资源实例		操作	资源实例	生效条件	申请期限
		资源删除(命名空间域)	<input type="text" value="请选择命名空间实例"/>	无生效条件	6个月
		资源更新(命名空间域)	<input type="text" value="请选择命名空间实例"/>	无生效条件	6个月
		资源查看(命名空间域)	<input type="text" value="请选择命名空间实例"/>	无生效条件	6个月
		资源创建(命名空间域)	<input type="text" value="请选择在哪些【命名空间】下【资源创建(命名空间域)】"/>	无生效条件	6个月
		命名空间进阶	<input type="text" value="请选择命名空间实例"/>	无生效条件	6个月
		命名空间更新	<input type="text" value="请选择命名空间实例"/>	无生效条件	6个月

选择操作 * 容器管理平台(bk_bcs_app) 搜索操作名称, 按 Enter 搜索

操作权限: 集群资源查看 集群资源管理

项目 已选 18 / 27 个

未启用 已选 0 / 2 个

关联资源实例

操作	资源实例	生效条件	申请期限
项目查看, 集群创建, 集群管理, 集群删除, 命名空间创建, 命名空间查看, 命名空间更新, 命名空间删除, 资源创建, 资源更新, 资源删除, 资源查看, 资源更新(命名空间), 资源删除(命名空间), 资源查看(命名空间), 命名空间更新(命名空间), 命名空间删除, 模板集创建, 模板集查看, 模板集复制, 模板集更新, 模板集删除, 模板集实例化	K8S容器服务测试	无生效条件	12个月

理由 *

项目集群管理

提交 取消

最后走完审批流程即可

我的申请

申请列表 3天

容器管理平台权限申请 03-07
理由: 项目集群管理

基本信息

申请单号:
申请类型: 自定义权限申请
申请人:
申请时间:
所在组织:
理由: 项目集群管理

申请内容 (容器管理平台)

4. kubeconfig使用流程 在权限中心申请完权限可以把API密钥作为kubeconfig token使用，在一台中心的服务器（例如：运维机）上使用kubectl命令管控相关集群，kubeconfig使用文档请见：<https://kubernetes.io/zh/docs/concepts/configuration/organize-cluster-access-kubeconfig/> 具体使用步骤如下：

5. 如果中心服务器不存在kubectl二进制文件，可以从现有服务器复制kubectl二进制文件，也可以从<https://kubernetes.io/docs/tasks/tools/>下载对应环境的kubectl二进制到/usr/bin/ PATH环境变量路径下，并添加可执行权限：chmod +x /usr/bin/kubectl

6. 在中心服务器上创建kubeconfig文件，kubectl默认使用的kubeconfig文件为：在/root/.kube/config，也可以把kubeconfig文件内容存在某个文件，然后在使用kubectl命令时指定kubeconfig文件，例如，把kubeconfig内容存在文件/root/.kube/demo_config中，使用如下命令使用kubeconfig文件。注意，使用示例中kubeconfig内容时需要把里面的 \${cluster_id} 变量替换为实际的集群ID，例如：BCS-K8S-40001，把 \${API密钥} 替换为你刚刚创建的API密钥

```
kubectl --kubeconfig=/root/.kube/demo_config get node
```

Kubeconfig使用示例：

```
kubectl --kubeconfig=/root/.kube/demo_config get node
```

/root/.kube/demo_config内容示例如下：

```
apiVersion: v1
kind: Config
clusters:
  - cluster:
      server: 'https://[bcs-api-gateway地址]/clusters/${cluster_id}'  
      bcs-api-gateway地址
    contexts:
      - context:
          cluster: '${cluster_id}'
          user: '...'
        name: BCS
    current-context: BCS
  users:
    - name: '...'
      user:
        token: '${API密钥}'
```

7. K8S原生API使用流程 如果在一些应用中无法使用kubectl命令的时候，可以使用API密钥调用Kubernetes API，来完成对集群的管控，以下介绍下常用的使用场景：

8. 获取kubelet版本号 kubectl命令 [kubectl version](#)

Kubernetes API

```
...
curl -X GET -H "Authorization: Bearer ${API密钥}" -H "accept: application/json" "https://[bcs-api-gateway]/clusters/${cluster_id}/version"
...
```

- 获取命名空间列表 kubectl命令 [bash kubectl get ns](#)

Kubernetes API

```
...
curl -X GET -H "Authorization: Bearer ${API密钥}" -H "accept: application/json" "https://[bcs-api-gateway]/clusters/${cluster_id}/api/v1/namespaces"
...
```

- 获取节点列表 kubectl命令 [bash kubectl get node](#)

Kubernetes API

```
...
curl -X GET -H "Authorization: Bearer ${API密钥}" -H "accept: application/json" "https://[bcs-api-gateway]/clusters/${cluster_id}/apis/metrics.k8s.io/v1beta1/nodes"
...
```

- 更多Kubernetes API使用文档请参考：<https://kubernetes.io/docs/reference/kubernetes-api/>

FAQ

1. 如何申请到平台使用的API密钥？ 目前蓝鲸权限中心只提供个人实名权限申请功能，暂不支持平台权限申请，如果平台要申请API密钥，先申请个人实名权限，待蓝鲸权限中心支持平台权限申请后再做权限切换

FAQ

产品使用

BCS 与 K8S 的关系和区别是什么

K8S 为容器编排引擎，BCS 是容器管理平台，支持 K8S 容器编排引擎，提供便捷的容器管理服务，更多介绍详见 [产品架构](#)。

一个集群需要至少需要几台机器

集群由 Master 和 Node 组成，其中 Master 主要用来部署集群的基础组件，Slave 主要用来承载业务容器。

“

注：master 不允许调度，因此，至少需要一台 slave 运行业务容器。

”

- 平台建议 Master 配置 建议至少 4 核 / 8 G / 3 台 / CentOS 7.4 +
- 平台建议 Node 配置 配置和数量视业务需求而定(至少需要 1 台)；操作系统：CentOS 7.4 +

调度约束

用户通过调度约束可以自动的为 POD 选择指定节点，而且可以通过 **亲和性** 和 **反亲和性** 实现个性化的调度能力。

“

注意：使用不当可能导致其他的 POD 调度不成功。

”

- 使用场景
 - 想要自己的服务运行到指定的节点
 - 限制其他服务运行到指定节点
- 出现节点匹配提示问题

```bash

## 错误信息

No nodes are available that match all of the predicates: MatchInterPodAffinity (2), MatchNodeSelector (3), NodeNotReady (3), PodToleratesNodeTaints (3).``

- 确认是否必须调度约束

“

如果自己的场景不必使用调度约束，可以直接去掉；如果必须使用可以确认下面条件是否满足

”

- 是否设置相应的节点标签，比如 nodeSelect 为 app=test，这样节点必须要打 app=test 的标签
- 当前节点是否有设置亲和性，比如设置了亲和性，那么必须满足亲和性条件
- 其他服务是否有设置反亲和性，比如别的节点设置了反亲和性，那么设置的节点就不允许其他服务调度

## 问题排查

### 拉取镜像失败

出现镜像拉取失败的问题，可能有如下两种可能导致：

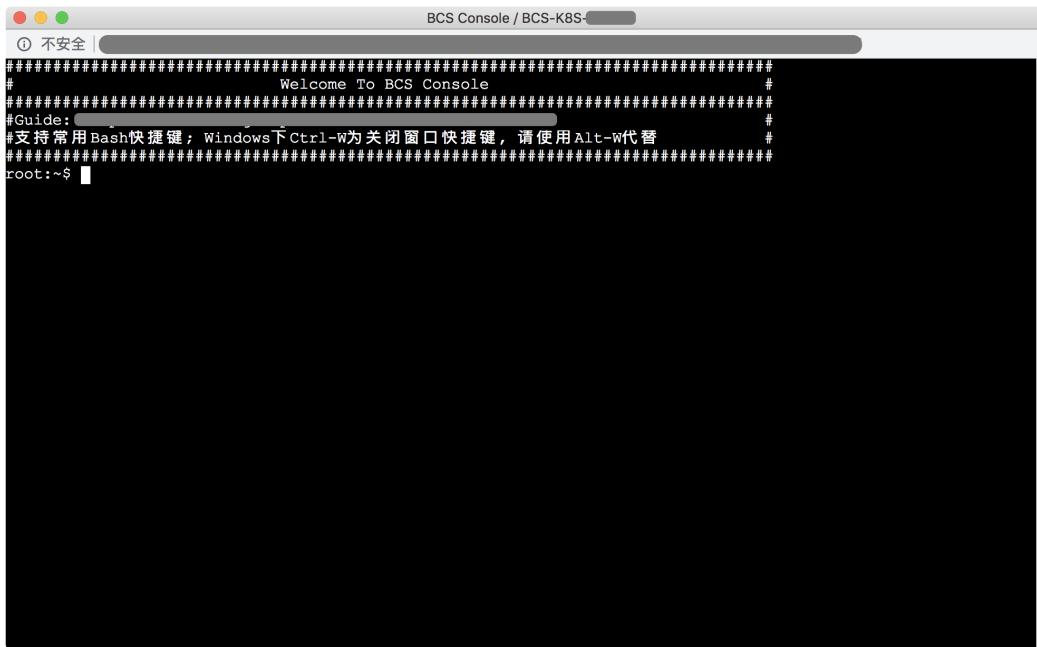
- 镜像不存在 针对这种情况，需要用户上传镜像即可。
- 节点没有权限拉取镜像 这种情况，一般是由平台侧创建 `imagePullSecrets` 失败导致。

## 启动容器失败

这种情况一般是用户镜像有问题，用户可以通过如下两种方式查看日志：  
- 使用 Webconsole，`kubectl logs pod_name -n namespace_name`  
- 登录节点机器，通过 `docker logs container_id` 查看相应日志信息

### 使用 Webconsole 查看日志

- 登录 Webconsole 点击容器服务的右下角“WebConsole”，点击相应的集群，弹出 Webconsole 页面



然后，输入 `kubectl logs pod_name -n namespace_name`，查看指定命名空间下的应用日志

### 登录节点查看日志

- 通过点击应用详情，跳转到应用详情页

- 查找 Pod / Taskgroup 管理项，查看到部署的节点

- 登录节点查看相应的容器日志

### 关于 POD 创建后一直处于 Waiting 或 ContainerCreating 状态

#### 检查应用配置的资源设置

首先，通过查看事件日志，如果此时出现下面这种错误，可以认为启动容器的资源不能满足需求。

```
bash to start sandbox container for pod ... Error response from daemon: OCI runtime create failed:
container_linux.go:348: starting container process caused "process_linux.go:301: running exec setsns process for
init caused \"signal: killed\"": unknown
```

其次，检查下应用下面容器的配置，类似下图；然后根据需求设置 request 和 limit 数量

更多设置 

|        |     |      |                                                                                      |      |                                                                                      |         |      |  |
|--------|-----|------|--------------------------------------------------------------------------------------|------|--------------------------------------------------------------------------------------|---------|------|--|
| 命令     | 挂载卷 | 环境变量 | 资源限制                                                                                 | 健康检查 | 就绪检查                                                                                 | 非标准日志采集 | 生命周期 |  |
| CPU限制: |     | 请输入  | m   | 请输入  | m   |         |      |  |
| 内存限制:  |     | 请输入  | Mi  | 请输入  | Mi  |         |      |  |

### 镜像问题

请参考 查看本文上面的 [拉取镜像失败](#) 处理流程