

产品简介

蓝鲸容器服务（BCS，Blueking Container Service）是高度可扩展、灵活易用的容器管理服务。

蓝鲸容器服务支持两种不同的集群模式，分别为原生 **Kubernetes** 模式和基于 **Mesos** 自研的模式。

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。

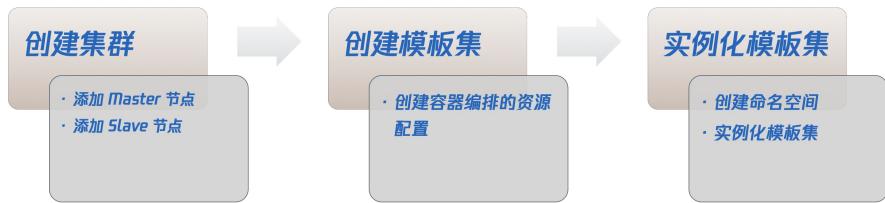
用户可以依据自身的需要选择容器编排的方式，以满足业务的特定要求。



容器服务怎么用

见下图，只需要三步即可运行服务：

1. 创建集群
2. 创建模板集
3. 实例化模板集



BCS 快速入门

目前，蓝鲸容器管理平台已正式对外开源，GitHub 地址：<https://github.com/Tencent/bk-bcs>，<https://github.com/Tencent/bk-bcs-saas>

术语解释

了解蓝鲸容器服务，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有 K8S、Mesos。
- **集群**：是指容器运行所需要的物理机或虚拟机资源的集合，可以选择集群的模式是 Kubernetes（简称 K8S）或者 Mesos。
- **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 中应用由 K8S 的工作负载（Workload）、服务（Service、Ingress）构成一个整体对外提供服务。
- **模板集**：配置文件模板的集合，简化服务管理的复杂度，可以实现部署、升级应用。
- **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。
- **资源**：资源中可以定义业务配置项，通过配置模板中关联这些业务配置项，可以实现对容器中业务进程使用配置的个性化管理。
- **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、配置管理、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>集群创建</p> <ul style="list-style-type: none"> 支持自定义设定Master和Slave节点，一键自动安装集群组件 用户独占集群，保证安全可靠 <p>集群管理</p> <ul style="list-style-type: none"> 支持动态伸缩，可以添加集群节点，也可以实时剔除集群节点 支持集群和节点级别的监控告警及主要数据的视图展示
配置管理	配置管理为用户提供了对容器编排方式的管理方案，支持用户设置容器编排的模板集合，支持设定命名空间以及对应的用户变量。通过配置管理，用户可以只编辑一套配置模板集，通过使用不同的命名空间进行模板集实例化得到不同的容器集合	<p>管理容器编排配置模板</p> <ul style="list-style-type: none"> 支持配置模板集的多版本管理 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，可以看到容器各个编排维度的信息和状态。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	<p>容器视图</p> <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 <p>容器操作</p> <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。此外还能够将感兴趣的镜像收藏到我的收藏中	<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。公共镜像所有人都能够看到 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。 我的收藏用于用户快速索引感兴趣的镜像
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	<p>服务管理</p> <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 <p>负载均衡器管理</p> <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

产品优势

双引擎驱动的容器编排方案

BCS 支持两种集群模式可选：**Kubernetes**、**Mesos**。

Kubernetes 和 Mesos 都是基于 Google borg 系统开源的项目，二者在产品定位上有所不同，Kubernetes 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；Mesos 则更侧重于**分布式资源调度**，对应用编排的能力更多依赖于第三方 Framework 系统或结合业务场景进行深度定制。

BCS 对这两种技术体系的集群模式都支持，用户可以根据需要进行选择。

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 Mesos

- 基于 Mesos 的分布式资源调度
- 结合游戏特点定制化开发的 Framework：BCS-Scheduler

基于 Docker 的服务生态

- 服务发现

不管是基于 Kubernetes 还是 Mesos 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 通过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

颁证日期: 2019 年 7 月 30 日
年检日期: 2020 年 7 月 30 日

有效期至: 2020 年 7 月 30 日



容器解决方案

证书编号: R01021

兹 证 明

深圳市腾讯计算机系统有限公司

(深圳市南山区高新区高新南一路飞亚达大厦 5-10 楼, 518000)

蓝鲸智云容器管理平台 符合:
云计算开源产业联盟文件《可信云容器解决方案评估方法》

本证书覆盖下述范围:

位于深圳市南山区高新区高新南一路飞亚达大厦 5-10 楼的深圳市腾讯计算机系统有限公司的容器解决方案在基本能力要求、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评估标准。应用场景满足以下四个:

- 开发测试场景 持续集成 (CI) 运维自动化 微服务
 持续交付 (CD)

(本证书有效性依据发证机构的定期监督获得保持。

可信云容器解决方案评估意味着该解决方案通过云计算开源产业联盟组织的可信云容器解决方案评估。可信云容器解决方案评估是对颁证日期前解决方案提供商所申请容器解决方案的信息披露, 评估过程以云计算开源产业联盟的评估标准文件为依据。由于可信云容器解决方案评估是针对颁证日期前解决方案情况进行披露, 用户依然应留意在具体使用过程中可能存在的风险。



中国信息通信研究院
地址: 北京市海淀区花园北路 52 号
网址: <http://www.caict.ac.cn/>



云计算开源产业联盟
地址: 北京市海淀区花园北路 52 号
网址: opensourcecloud.cn



产品开源

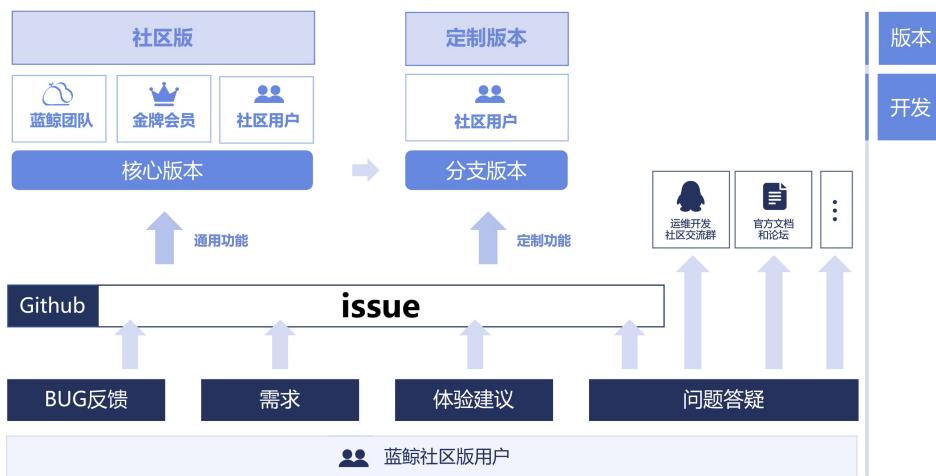
BCS 团队对容器管理平台进行开源, 希望将我们的技术和沉淀反馈给社区, 期望能帮助更多的人解决问题; 同时也邀请容器技术爱好者一起参与建设, 让产品变更更加强大和易用, 构建生态活跃的技术社区。



蓝鲸容器平台正式开源

为了社区，与君同行

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



官方微博公众号



社区版QQ交流群



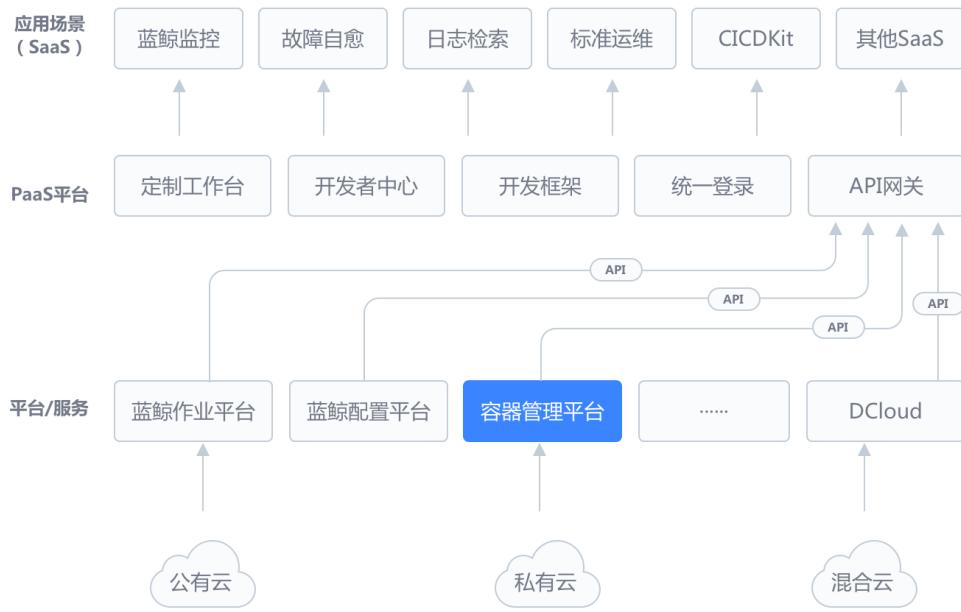
腾讯蓝鲸运维开发
交流群

蓝鲸官网: <http://bk.tencent.com>

GitHub: <https://github.com/Tencent/bk-bcs-saas>
<https://github.com/Tencent/bk-bcs>

产品架构图

BCS 是统一的容器部署管理解决方案，为了适应不同业务场景的需要，BCS 内部同时支持基于 Mesos 和基于 K8S 的两种不同的实现。



BCS 在蓝鲸中的位置

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）。



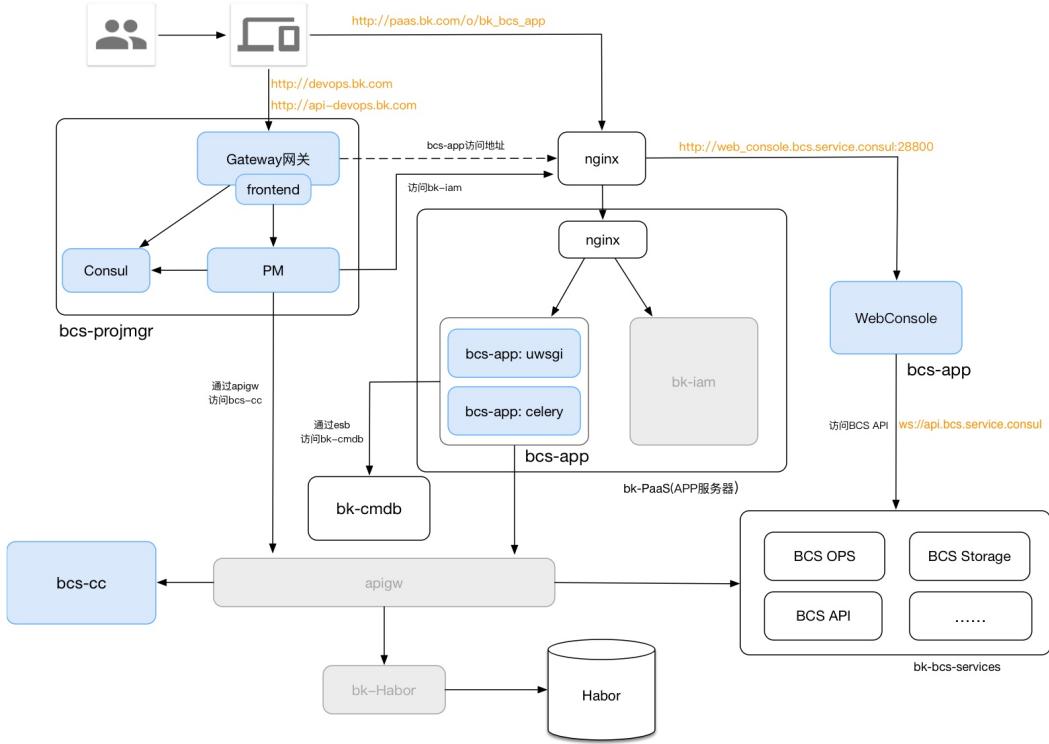
SaaS 依赖的服务介绍： - **bk-PaaS**: 为 BCS SaaS 提供了 4 大服务(统一登录、开发者中心、ESB 和应用引擎)，其中 bcs-app 由应用引擎托管

- **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

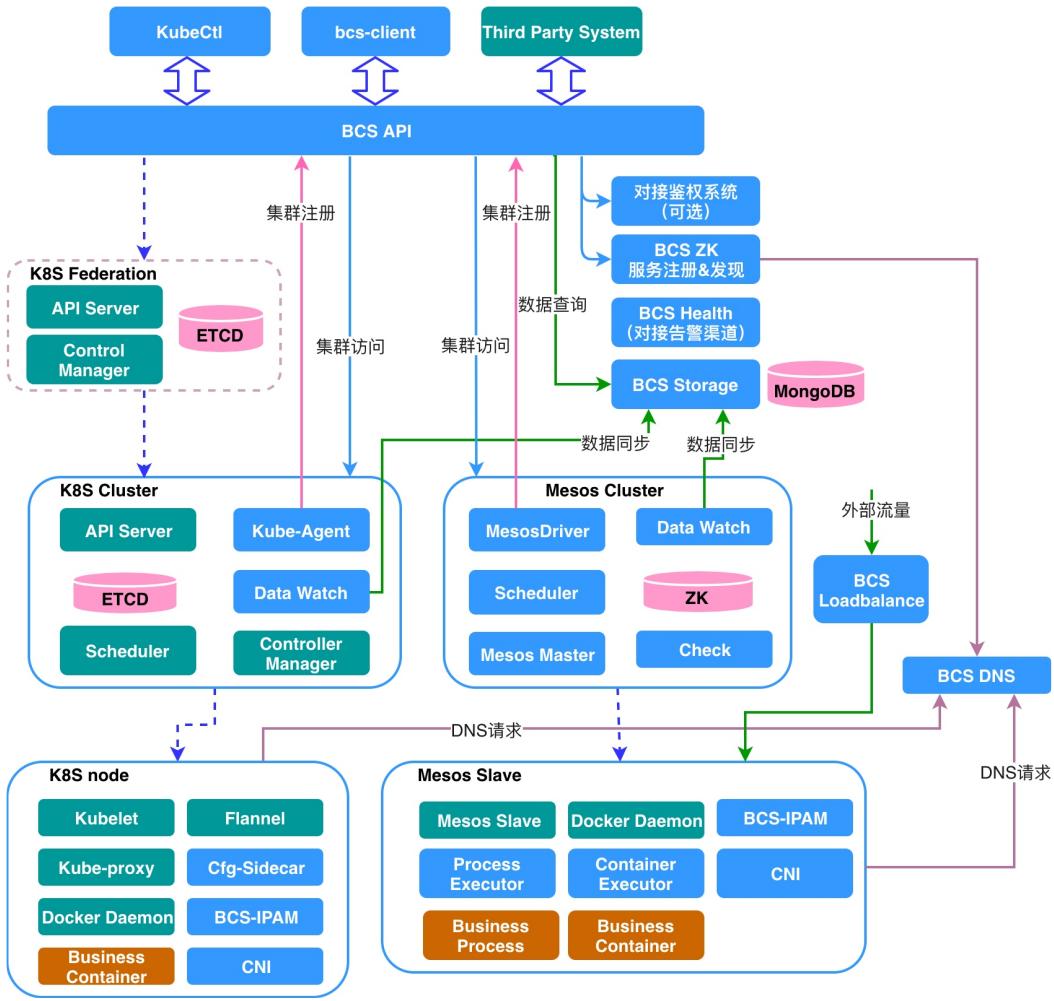
SaaS 包含 bcs-projmgr, bcs-app, bcs-cc 三个模块。

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 以及 Mesos 集群的整体架构图：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 Mesos 集群或者 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。

Mesos 编排的具体说明： - Mesos 自身包括 Mesos Master 和 Mesos Slave 两大部分，其中 Master 为管理中心节点，负责集群资源调度管理和任务管理；Slave 运行在业务主机上，负责宿主机资源和任务管理。 - Mesos 为二级调度机制，Mesos 本身只负责资源的调度，业务服务的调度需要通过实现调度器（图中 Scheduler）来支持，同时需实现执行器 Executor（Mesos 自身也自带有一个 Executor）来负责容器或者进程的起停和状态检测上报等工作。 - Mesos（Master 和 Slave）将集群当前可以的资源以 offer（包括可用 CPU、MEMORY、DISK、端口以及定义的属性键值对）的方式上报给 Scheduler，Scheduler 根据当前部署任务来决定是否接受 Offer，如果接受 Offer，则下发指令给 Mesos，Mesos 调用 Executor 来运行容器。 - Mesos 集群数据存储在 ZooKeeper，通过 Datawatch 负责将集群动态数据同步到 BCS 数据中心。 - Mesos Driver 负责集群接口转换。 - 所有中心服务多实例部署实现高可用：Mesos driver 为 Master-Master 运行模式，其他模块为 Master-Slave 运行模式。服务通过 ZooKeeper 实现状态同步和服务发现。

模板集使用介绍

在 BCS 中，模板集合是 Kubernetes/Mesos 配置文件模板的集合。您可以通过将多个应用的配置构建成一个模板集，来简化服务管理的复杂度；您还可以将模板里面需要频繁修改的数据设置成变量，以方便维护。



通过『模板集』+『变量』来管理您的业务将非常方便。

理解“服务”

如果您的服务由多个应用构成，通过 模板集 + 变量 生成“应用”来管理您的服务将会非常方便。

为了方便大家理解，本文将以一个简单的 Nginx 服务为例来说明。该服务下只有一个 Nginx 应用，通过修改模板中 **NGINX_PORT** 变量的值，将应用以不同的端口部署到不同的命名空间中。

创建模板集

推送镜像到仓库

在创建模板前，先通过以下几个步骤将 Nginx 应用的镜像推送到蓝鲸容器服务的镜像仓库中。

获取镜像的压缩包

在本地执行 `docker save` 获取镜像压缩包，您也可以直接点击下载 `nginx.tar`。

```
docker pull nginx
docker save nginx > nginx.tar
```

上传镜像

选中【仓库】菜单的【项目镜像】页面，参考 [Harbor 仓库使用指南](#)，通过命令行工具来推送镜像。

创建项目模板集

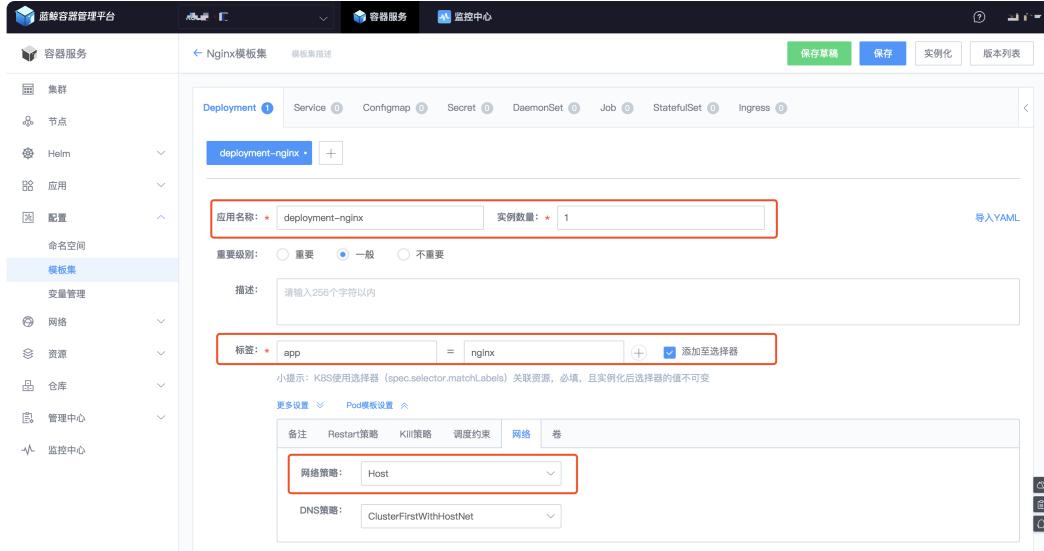
选中【模板集】菜单，点击【添加模板集】。

进入创建模板集页面后，可以在模板集定义 Deployment、Service 等资源，这些资源的详情介绍可以参考 [Deployment 说明文档](#)。本文的案例中我只需要创建一个 Deployment。

首先填入一些基本信息，包括名称、描述，实例数量指最终拉起的实例数。Kubernetes 使用选择器

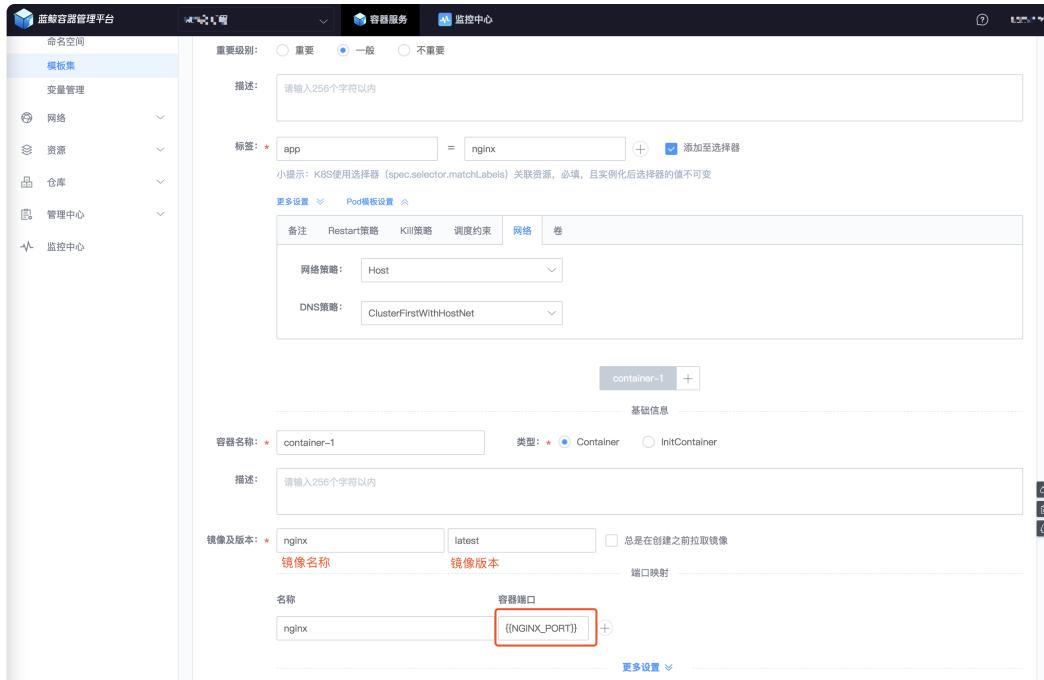
(spec.selector.matchLabels) 关联资源，实例化后选择器的值不可变，所以在创建模板的时候需要给应用打一个固定的标签，并且把这个标签添加到选择器中。

点击下方的 Pod 模板设置，将网络策略设置为 Host 模式，可以直接用主机网络，将 Nginx 容器端口对外暴露。



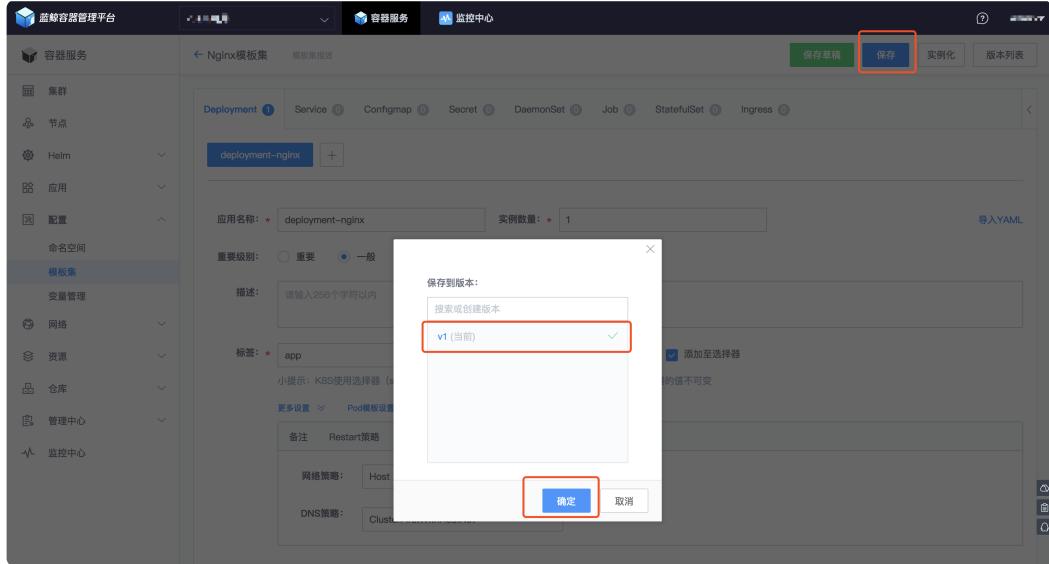
The screenshot shows the 'Container Services' section of the Bluewhale platform. A deployment named 'deployment-nginx' is selected. In the 'Pod模板设置' (Pod Template Settings) section, the 'Network Strategy' dropdown is set to 'Host', which is highlighted with a red box.

接下来定义 Nginx 容器，需要填写名称、描述，选择 Nginx 镜像。同时将容器端口以变量的形式填写，这样可以将应用以不同的端口部署到不同的命名空间中。



The screenshot shows the 'Container' configuration for the 'container-1' instance of the 'deployment-nginx' template. In the 'Container Port' field, the value '{{INGINX_PORT}}' is highlighted with a red box. Other fields shown include 'Container Name' (container-1), 'Image' (nginx latest), and 'Port' (80).

这样我们的模板集就创建好了，点击上方的保存按钮，将模板集的保存到一个版本中。

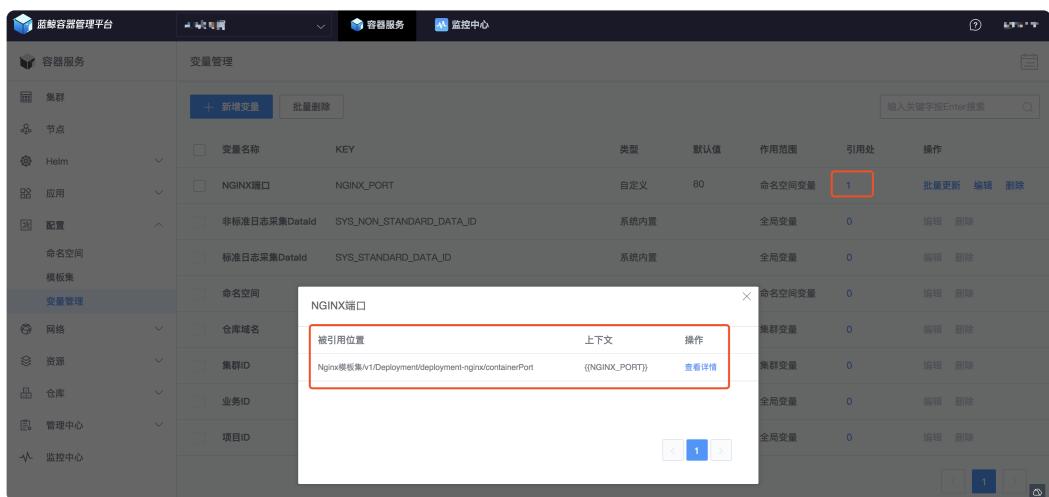


设置变量

选中【变量管理菜单】，统一管理所有的变量，变量分为三类：

- 全局变量：变量的生效范围为整个项目，如系统变量里面的：项目 ID、业务 ID
- 集群变量：变量的生效范围为集群，同一个变量可以针对不同的集群设置不同的值，如系统变量里面的：集群 ID、仓库域名
- 命名空间变量：变量的生效范围为命名空间，同一个变量可以针对不同的命名空间设置不同的值

点击引用处，可以查看所有引用了这个变量的模板的详情。



点击批量更新，可以同时对不同的集群、命名空间赋值。

The screenshot shows the 'Variable Management' section of a container service interface. On the left, there's a sidebar with categories like Cluster, Node, Helm, Application, Configuration, Variable Management, Network, Resource, Warehouse, and Management Center. The 'Variable Management' section is currently selected. The main area displays a table of variables:

变量名称	KEY
NGINX端口	NGINX_PORT
非标准日志采集DataId	SYS_NON_STANDARD_DATA_ID
标准日志采集DataId	SYS_STANDARD_DATA_ID
命名空间	SYS_NAMESPACE
仓库域名	SYS_JFROG_DOMAIN
集群ID	SYS_CLUSTER_ID
业务ID	SYS_CC_APP_ID
项目ID	SYS_PROJECT_ID

At the bottom right of the table are 'Save' and 'Cancel' buttons.

模板集中引用变量

模板集中可采用 {{变量名}} 的方式引入变量

This screenshot shows the 'Deployment' configuration page in a Kubernetes management platform. The deployment name is 'deployment-1-copy1'. In the 'spec.selector.matchLabels' field, the value is set to {{INSTANCE_NUM}}. A red box highlights this field. The deployment also includes labels '23' and '43' under the 'spec.labels' field. Other deployment details like container configuration and image source are visible.

This screenshot is similar to the previous one, showing the 'Deployment' configuration page for 'deployment-1-copy1'. However, a modal window titled 'Template Set Variable' is overlaid on the right side of the screen. This modal contains a table of variables and their corresponding keys:

变量名	KEY
实例数	INSTANCE_NUM
非标准日志采集DataId	SYS_NON_STANDARD_DATA_ID
标准日志采集DataId	SYS_STANDARD_DATA_ID
命名空间	SYS_NAMESPACE
仓库域名	SYS_JFROG_DOMAIN
集群ID	SYS_CLUSTER_ID
业务ID	SYS_CC_APP_ID
项目ID	SYS_PROJECT_ID

创建服务示例

选中【模板集】菜单，点击实例化按钮进入实例化页面，选择模板集、命名空间。变量默认值为在上一步设置的值，您也可以在实例化页面修改这个值。最后点击创建就完成了实例化操作。

The screenshot shows the 'Container Management Platform' interface. On the left, there's a sidebar with various navigation items like '集群', '节点', 'Helm', '应用', '配置', '命名空间', '模板集' (which is selected), '变量管理', '网络', '资源', '仓库', '管理中心', and '监控中心'. The main area has a title '模板实例化' (Template Instantiation). It shows a 'Nginx 模板集' (Nginx Template Set) created by 'admin'. There are dropdowns for '模板集版本' (Template Set Version) set to 'v1' and '模板' (Template) set to 'deployment-nginx'. A button '清空全选' (Clear All Selection) is also present. Below this, there's a section '简介: 模板集描述' (Introduction: Template Set Description) with a button 'namespace1'. A large text input field labeled '高可用集群1 / namespace1的详细配置' (Detailed configuration for highly available cluster 1 / namespace1) contains a variable 'NGINX端口(NGINX_PORT)' with a value of '80'. A red box highlights this input field. Below it is a '预览:' (Preview:) section showing a YAML configuration snippet for a deployment named 'deployment-nginx'. The snippet includes metadata, labels, annotations, and specific fields like 'io.tencent.paas.K8sDeployment_deployment-nginx' and 'io.tencent.bcs.clusterId: BCS-K8S-40003'. At the bottom of the preview section is a '部署' (Deploy) button.

确认完成

应用部署完成后，您可以在容器服务左侧导航中点击“应用”，查看 Nginx 服务应用实例，并可以通过 Host IP 和配置的容器端口访问服务。

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with various navigation options like Helm, Application, Configuration, Network, Resource, Library, Management Center, and Monitoring Center. The main area is focused on a deployment named 'deployment-nginx'. It shows basic information: Application name: deployment-nginx, Created time: 2019-03-15 10:24:24, Updated time: 2019-03-15 10:24:24, Namespace: namespace1, and Cluster: High Availability Cluster 1. Below this are two charts: 'CPU 使用率' (CPU Usage) and '内存使用率' (Memory Usage), both showing low utilization over time. Under the 'Pod 管理' tab, it lists a single pod named 'deployment-nginx-5cf8bd4854-k9q4q' in a 'Running' state. The IP address is highlighted with a red box. A terminal window at the bottom shows the output of a curl command to the host IP, displaying the standard Nginx welcome page.

滚动升级

您可以通过配置不同的模板集版本来实现滚动升级的功能。

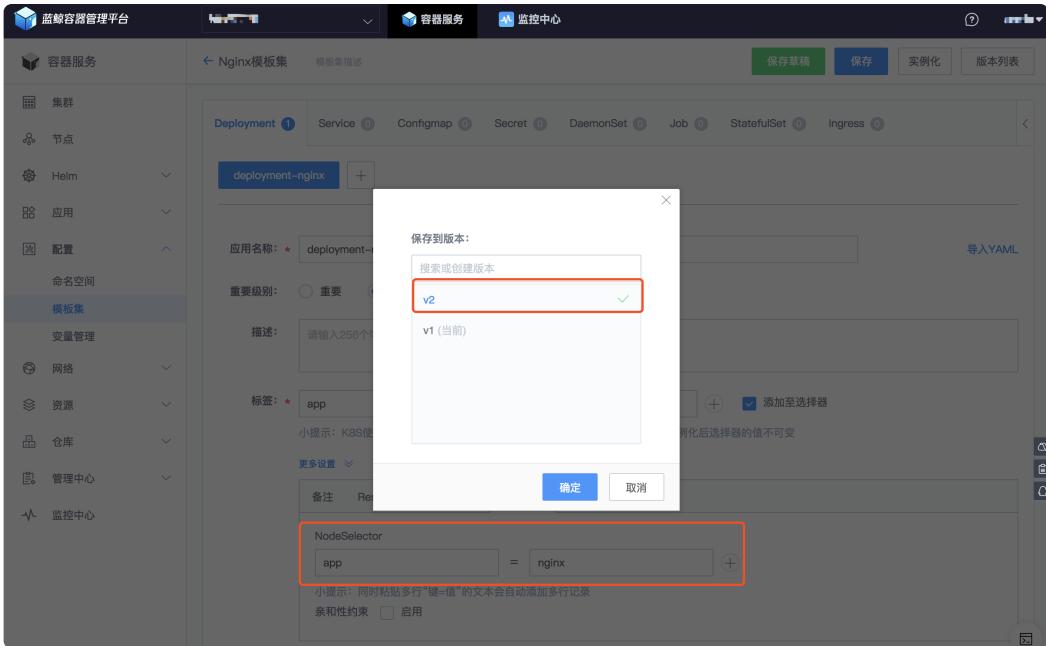
节点设置标签

通过节点设置标签，可以在调度的时候将应用调度到固定的主机上。本例中将集群节点中的一台主机添加 `app:nginx` 的标签。

This screenshot shows the 'Nodes' section of the BlueKing Container Management Platform. The sidebar on the left includes 'Container Service', 'Cluster', and 'Node' sections. In the main area, there's a 'Label Settings' button. Below it, a table lists nodes: '10.0.5.112' and '10.0.1.36'. Both nodes are in a 'Normal' status and belong to the 'High Availability Cluster 1'. The '10.0.1.36' node has the 'app:nginx' label applied, which is highlighted with a red box. There are 'Stop Allocation' and 'Set Label' buttons next to each row.

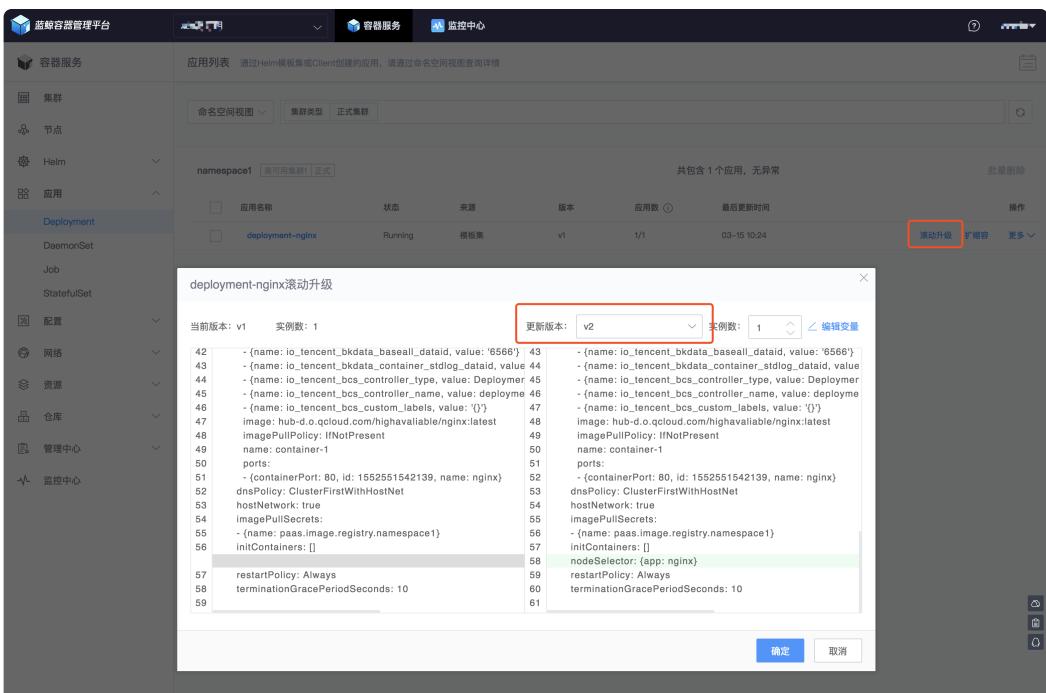
新增模板集版本

在模板集的调度约束中，将 `NodeSelector` 设置为 `app:nginx`，并保存到一个新的版本中。



滚动升级

在应用页面，点击滚动升级并选择最新的版本，可以页面上查看滚动升级前后的配置文件差异，点击确定将应用滚动升级到新版本。



进入应用的详情页面，可以发现 Nginx 应用已经被重新调度到了设置了 `app:nginx` 标签的主机上，并可以通过新的 Host IP 和配置的容器端口访问 Nginx 服务。

The screenshot shows the Bluewhale Container Management Platform's deployment details for a 'deployment-nginx' pod. Key information includes:

- 应用名称:** deployment-nginx
- 创建时间:** 2019-03-15 10:24:24
- 更新时间:** 2019-03-15 11:31:25
- 所在命名空间:** namespace1
- 所属集群:** 高可用集群1

Resource usage charts show CPU and memory usage over time. The pod management table highlights the pod status as 'Running' and its host IP as 10.0.1.36. A terminal window shows the curl command output of the Nginx welcome page.

附录

节点标签说明文档

- K8S

节点标签主要是一组绑定到 K8S 资源对象上的 key/value 对。通过给指定的资源对象捆绑一个或多个不同的 label 来实现多维度的资源分组管理功能，以便于灵活，方便地资源分配，调度，配置，部署等管理工作。

- Mesos

节点标签主要是 Mesos 一组 key/value 对，允许以更灵活、多样的形式进行资源分配。

镜像仓库使用指南

蓝鲸容器服务采用 Harbor 做镜像存储，用户除了可以使用 Harbor 的管理页面对项目、用户和镜像做管理外，还提供与容器服务配套的 API，方便容器服务和仓库使用。

Harbor 镜像仓库简介

Harbor 是 Vmware 公司开源的企业级 Docker Registry 管理项目，开源项目地址：[Harbor](#)。

在 Harbor 仓库中，镜像在被推送到仓库之前都需要有一个自己所属的项目。用户和仓库都是基于项目进行组织的，而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](#)。

BCS 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】帮助，可以获取仓库账号。



包含以下信息：
- 镜像仓库地址：[`<Registry_URL>`](#)；Web 端也可以访问。注意：地址包含端口。
- 镜像仓库账号、密码（Harbor 中的一个项目账号，可以上传 镜像 和 Charts）
- 项目 ID：[`<Project_ID>`](#)，如上图中的 `joyfulgame`

登录镜像仓库

```
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:1.17.0` 推送到当前项目下，首先将镜像按下面的命令重新 tag。

```
docker tag nginx:1.17.0 <Registry_URL>/<Project_ID>/nginx:1.17.0
```

```
docker push <Registry_URL>/<Project_ID>/nginx:1.17.0
```

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。

```
docker pull <Registry_URL>/<Project_ID>/nginx:1.17.0
```

登录 Harbor Web 仓库

获取 Harbor 账号

- 获取项目仓库账号

在 [获取项目仓库账号](#) 中可以获取 Harbor 的项目账号，可以访问项目（BCS 中新建的项目）仓库。

- 获取 Harbor 管理员账号

在 [部署 BCS 的文档](#) 中有一个环境变量文件 `globals.env`，其中可以找到 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 Web 端

在 [获取项目仓库账号](#) 中可以获取 Harbor 的 Web 端访问 URL，进来可以看到项目列表，包含 公共项目 和 私有项目。

The screenshot shows the Harbor Web interface with the following details:

- Header:** Harbor logo, search bar, language switch (中文简体), user info (1564969588969835), and a sidebar menu with '项目' (Projects) selected.
- Left Sidebar:** Buttons for '+ 新建项目' (New Project) and '日志' (Logs).
- Right Sidebar:** Statistics: 1私有 (Private), 3公开 (Public), 2私有 (Private), 39公开 (Public), and a link '所有项目' (All Projects).
- Table:** A table listing projects with columns: 项目名称 (Project Name), 访问级别 (Access Level), 角色 (Role), 镜像仓库数 (Image Repository Count), Helm Chart 数目 (Helm Chart Count), and 创建时间 (Creation Time). The data is as follows:

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' (1 - 4 Total 4 records).

在上图中点击 项目仓库 `joyfulgame`，可以看到有 2 个镜像。

The screenshot shows the Harbor Helm Charts page for the project 'joyfulgame'. It lists two charts: 'joyfulgame/cmdb-standalone' and 'joyfulgame/nginx'. Both charts have a version of 1 and were downloaded once.

点击【Helm Charts】页面，可以看到仓库中上传的 Charts。

The screenshot shows the Harbor Helm Charts page for the project 'joyfulgame'. It lists eight charts: 'blueking-ingress', 'gitlab', 'gitlab-ce', 'jenkins', 'nfs-client-provisioner', 'rancher', 'rumpetroll', and 'wordpress'. All charts are in normal status, with versions ranging from 1 to 2.

- Harbor 管理员账号 Web 端

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。

The screenshot shows the Harbor User Management page. It displays a table of users with columns for '用户名' (Username), '管理员' (Administrator), '邮箱' (Email), and '注册时间' (Registration Time). The users listed are: 1565186309299214, 1565251993442350, 1565252001808958, and 1565252042487500.

BCS K8S / Mesos 方案功能对比

BCS 支持 K8S 和 Mesos 两种容器编排方案，以下是两种编排方案的功能对比：

功能点	K8S 方案	Mesos 方案	主要功能及差异说明
-----	--------	----------	-----------

POD	POD	POD(taskgroup)	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. Pod 是所有业务类型的基础，它是一个或多个容器的组合，例如：业务容器，sidecar容器 2. 这些容器共享存储、网络和命名空间，以及如何运行的规范 3. 在 Pod 中，所有容器都被统一编排和调度
无状态应用	ReplicaSet/Deployment	Application/Deployment	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. 支持设定编排策略（Restart、Kill等）； 2. 支持设定容器启动参数； 3. 支持设定镜像版本及加载方式； 4. 支持设定端口映射； 5. 支持设定环境变量； 6. 支持设定 label、备注； 7. 支持设定卷、configmap、secret 的关联关系； 8. 支持设定网络； 9. 支持设定资源配置设定； 10. 支持设定健康检查机制； <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. K8S 支持设定生命周期、就绪检查； 2. K8S 支持更多的存储 driver，并且从 1.9 版本开始支持CSI； 3. 调度策略配置不同：K8S：基于 label 的调度策略，通过 selector 进行筛选 Mesos：基于变量运算符的调度策略，支持获取 CC 的属性作为调度依据 4. 自定义调度支持方式不同：K8S：支持自定义 controller Mesos：支持自定义调度插件 5. Mesos 方案支持 POD 固定 IP 调度 6. Mesos 方案支持 bridge 模式下端口随机
部署方案	Deployment	Deployment	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. recreate 操作方式； <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. rollingUpdate：K8S 方案支持设置滚动方式，最大不可用数和最大更新数 Mesos 方案支持设置滚动方式，周期，频率，和手动模式

有状态应用	StatefulSets	Application	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. POD 拥有独立唯一的 ID; 2. POD 拥有独立唯一的域名; 3. 挂载独立的数据卷;
任务	Job/CronJob	Application	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. 短任务单次执行，例如数据库初始化任务； <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. Mesos 方案无专门 Workload，通过设置 Application 的参数实现 2. K8S 支持部署定时任务
配置文件	Configmap	Configmap	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. Configmap 内容落地为文件或者环境变量; <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. K8S 方案支持在 POD command 中以变量方式引用; 2. Mesos 方案支持远端配置，包括文本和二进制配置文件
DaemonSet	DaemonSet	Application	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. 按主机 1: 1 动态部署容器; 2. 能动态跟随物理机资源缩扩容; <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. Mesos 方案无专门 Workload，需要通过设置 Application 的调度算法实现相同功能;
服务	Service	Service	<p>K8S、Mesos 方案均支持:</p> <ul style="list-style-type: none"> 1. 通过自定义域名描述外部服务; 2. 对 RS 或者 Application 的服务做抽象; 3. 支持容器故障或者扩缩容时 Backends 的动态刷新; 4. 支持简单的负载均衡策略; 5. 支持 HTTP、TCP、UDP 等主流通讯协议; <p>差异点主要有:</p> <ul style="list-style-type: none"> 1. Mesos 方案支持通过 location 对不同域名进行分流 2. Mesos 方案支持对两个 Application 之间设置流量权重 3. K8S 支持更多对外暴露服务折射方式，除 ClusterIP、LB 外，还是支持 NodePort

加密配置	Secrets	Secrets	K8S、Mesos 方案均支持: ● 1. 对敏感信息加密;
挂载卷	Volumes	Volumes	K8S、Mesos 方案均支持: ● 1. 支持 Empty DIR; ● 2. 支持 Host Path; ● 3. 支持挂载 Ceph、nfs 等远端目录; ● 4. 支持 subPath;
Ingress	Ingress	Ingress	K8S、Mesos 方案均支持: ● 1. 实现集群外部对集群内服务的访问; ● 2. 支持 HTTP、TCP; ● 3. 支持轮询的负载均衡方式; 差异点主要有: ● 1. Mesos 方案支持流量权重设置; ● 2. Mesos 方案支持 Balance Source;

Kubernetes Deployment 说明

Deployment 是 kubernetes(简称 K8S)中用于管理 Pod 的对象，从 1.2 版本开始引入，与 Replication Controller 相比，它提供了更加完善的功能，集成了上线部署、滚动升级、创建副本、暂停/恢复上线任务，回滚等功能，使用起来也更加方便

1. 模板示例

```

apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: servergame
spec:
  selector:
    matchLabels:
      app: servergame
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 2
  template:
    metadata:
      labels:
        app: servergame
    spec:
      nodeSelector:
        network: private
      imagePullSecrets:

```

```
- name: paas.image.registry.default
containers:
- name: servergame
  image: servergame:2.0.0
  args: [ "-c /opt/live.env.conf" ]
  env:
    - name: FUEL_ROLE
      value: servergame
    - name: INSTANCE_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
  envFrom:
    - configMapRef:
        name: external-config
    - secretRef:
        name: external-secret-config
  livenessProbe:
    httpGet:
      path: /g/status
      port: http
    initialDelaySeconds: 900
    timeoutSeconds: 5
  readinessProbe:
    httpGet:
      path: /g/status
      port: http
    initialDelaySeconds: 15
    timeoutSeconds: 5
  resources:
    requests:
      cpu: 100m
      memory: 3Gi
  ports:
    - name: http
      containerPort: 8080
  volumeMounts:
    - name: config
      mountPath: /opt/servergame/config
    - name: applogs
      mountPath: /opt/servergame/log
    - name: shm
      mountPath: /dev/shm
    - name: g3
      image: g3:0.5
      volumeMounts:
        - name: shm
          mountPath: /dev/shm
  volumes:
    - name: config
      configMap:
        name: apps-servergame-config
    - name: applogs
      emptyDir: {}
    - name: shm
      emptyDir:
        medium: Memory
        sizeLimit: 1Gi
```

2. 配置项介绍

2.1 Selector

`.spec.selector` 是可选字段，用来指定 label selector，指定当前 Deployment 所能管理的 Pod 范围。如果被指定，`.spec.selector` 必须匹配 `.spec.template.metadata.labels`，否则它将被 API 拒绝。如果 `.spec.selector` 没有被指定，`.spec.selector.matchLabels` 默认是 `.spec.template.metadata.labels`

2.2 Replicas

`.spec.replicas` 是可以选字段，指定期望的 Pod 数量，默认是 1

2.3 Strategy

`.spec.strategy` 指定新 Pod 替换旧 Pod 的策略。`.spec.strategy.type` 可以是 "Recreate" 或者是 "RollingUpdate"。"RollingUpdate" 是默认值。- "Recreate" 相对比较“暴力”，Deployment 在创建出新的 Pod 之前会先杀掉所有已存在的 Pod - "RollingUpdate" 则是使用 rolling update 的方式更新 Pod。用户可以指定 `maxUnavailable` 和 `maxSurge` 来控制 rolling update 的进程。-

`.spec.strategy.rollingUpdate.maxUnavailable` 是可选配置项，用来指定在升级过程中不可用 Pod 的最大数量。该值可以是一个绝对值，也可以是期望 Pod 数量的百分比(例如 10%)，通过计算百分比的绝对值向下取整。如果 `.spec.strategy.rollingUpdate.maxSurge` 为 0 时，这个值不可以为 0。默认值是 1 - `.spec.strategy.rollingUpdate.maxSurge` 是可选配置项，用来指定可以超过期望的 Pod 数量的最大个数。该值可以是一个绝对值或者是期望的 Pod 数量的百分比。当 `MaxUnavailable` 为 0 时，该值不可以为 0。通过百分比计算的绝对值向上取整。默认值是 1 - 示例中 `maxUnavailable: 0, maxSurge: 2`，表示滚动升级的过程中，处于 Available(Ready) 的 Pod 数不低于 `.spec.replicas-0`，同时所有的 Pod 数之和不会超过 `.spec.replicas+2`

2.4 Pod Template

Pod 是 K8S 创建或部署的最小/最简单的基本单位，可由单个或多个容器共享组成的资源。Pod template 作为定义方式，可被嵌套到 Deployment、StatefulSet、DaemonSet 等对象中，即下面介绍的 `.spec.template`。`.spec.template` 是 `.spec` 中唯一必须的字段，和 Pod 具备同样的 schema，除了 `apiVersion` 和 `kind` 字段。为了划分 Pod 的范围，Deployment 中的 Pod template 必须指定适当的 label(如 `app: servergame`) 和适当的重启策略。`.spec.template.spec.restartPolicy` 在不指定的情况下，默认为 `Always`。

2.4.1 NODESELECTOR

`.spec.template.spec.nodeSelector` 是较常用的调度 Pod 到具体 node 节点上的方法，它通过 K8S 的 label-selector 机制进行节点选择。如示例中，Pod 会被调度到已经打上 `network=private` 这一 label 的 node 节点上。

2.4.2 CONTAINERS

一个 Pod 可以管理一个或多个容器，`.spec.template.spec.containers[]` 即是描述这些容器的数组(例如，示例模板中的 Pod 包含 `servergame` 和 `g3` 这两个容器)。每个数组对应一个容器的配置，包括容器 `name`, 所使用的 `image`, 环境变量, `resources` 的限制等。

- 环境变量 环境变量的设置方法有多种
 - env 指定
 - 直接设置 key, value
 - 通过 valueFrom 获取 pod 信息(如 `status.podIP`), 设置 value
 - envFrom 直接从 configmap 或 secret 中获取环境变量
- 健康度探测(livenessProbe 和 readinessProbe) 容器的健康度探测维度分为两类: 存活探针(livenessProbe)和就绪探针(readinessProbe)。kubelet 使用 livenessProbe 来确定何时重启容器, 使用 readinessProbe 来确定容器是否已经就绪可以接受流量。探测手段可以通过 http, tcp 或者定义 command。无论使用哪种方式, 都会用到下面这几个参数
 - `initialDelaySeconds` 容器启动后第一次执行探测是需要等待多少秒
 - `timeoutSeconds` 一次性探测的超时时间
 - `periodSeconds` 探测的时间周期
 - `successThreshold` 探测失败后, 最少连续探测成功多少次才被认定为成功。默认是 1。对于 liveness 必须是 1。最小值是 1
 - `failureThreshold` 探测成功后, 最少连续探测失败多少次才被认定为失败。默认是 3。最小值是 1
- 容器挂载卷的引用 `.spec.template.spec.containers[i].volumeMounts[]` 中描述了当前容器需要使用到的一些 volume, 通过 name 与 volumes 绑定, mountPath 表示挂载到容器中的位置

2.4.3 VOLUMES

K8S 支持多种类型的卷, 其核心是目录, 可以通过 Pod 中的容器来访问。目录是如何形成的、支持该目录的介质以及其内容取决于所使用的特定卷类型。卷的类型: - configMap - emptyDir - hostPath - persistentVolumeClaim - cephfs - ... 类型很多, 这里对常用的 `configMap` 和 `emptyDir` 做重点说明。`configMap` 类型提供了一种将配置文件注入到 Pods 中的方法, 如示例片段,

```

volumes:
- name: config
  configMap:
    name: apps-servergame-config
  
```

通过在 `.spec.template.spec.containers[i].volumeMounts[]` 中引用 `name: config`, 可以将 `apps-servergame-config` 中定义的文件内容挂载到容器指定的 `mountPath` 下 `emptyDir` 类型主要用于某些应用程序无需永久保存的临时目录, 多个容器的共享目录等。这个目录的初始内容为空, 当 Pod 从 Node 上移除时, `emptyDir` 中的数据会被永久删除。根据介质的不同, 分为硬盘

(emptyDir: {})和内存存储。内存存储用于多个容器需要共享内存的情况，如示例片段，

```
- name: shm
  emptyDir:
    medium: Memory
    sizeLimit: 1Gi
```

指定了介质类型为 Memory 和 1G 大小的内存空间。

3. Deployment 之“吃豆小游戏”实践

下面是 BCS 小游戏中，Deployment 用法的示例：openresty-v2 是接入层模块，redis-v2 是 redis 服务模块。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: openresty-v2
  namespace: rumpetroll-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: openresty-v2
  template:
    metadata:
      labels:
        app: openresty-v2
    spec:
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      hostAliases:
        - ip: 127.0.0.1
          hostnames:
            - "game2-got.o.qcloud.com"
      containers:
        - name: openresty
          image: rumpetroll-openresty:0.61
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
              hostPort: 80
              protocol: TCP
          env:
            - name: DOMAIN
              value: game2-got.o.qcloud.com
            - name: MAX_CLIENT
              value: "2"
            - name: MAX_ROOM
              value: "100"
            - name: REDIS_HOST
              value: "redis"
            - name: NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: REDIS_PORT
              value: "6379"
```

```
- name: REDIS_DB
  value: "0"
- name: REDIS_PASSWORD
  value: ""
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-v2
  namespace: rumpetroll-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-v2
  template:
    metadata:
      labels:
        app: redis-v2
    spec:
      containers:
        - name: redis-v2
          image: redis:1.0
          imagePullPolicy: IfNotPresent
```

大部分的配置项都在上一节中做了介绍，这里仅对 openresty-v2 中用到的一些其他配置做简单说明。

`DNS policies`：一般情况下，Pod 访问集群内 dns 是不需要额外配置的，`dnsPolicy` 的默认值为 `ClusterFirst`，但由于 Pod 绑定了主机网络 `hostNetwork: true`，因此如果需要访问集群内网络，必须设置成 `ClusterFirstWithHostNet`。

`hostAliases`：通过 `hostAliases` 向 hosts 文件添加额外的条目，即本例中的 `127.0.0.1 game2-got.o.qcloud.com`

4. BCS 模板集操作

关于 Deployment 的实战演练，请参照 [快速构建 Nginx 集群](#)。

Kubernetes StatefulSet 说明

StatefulSet 是为了解决有状态服务的问题而设计的，它是一个给 Pod 提供唯一标志的控制器，可以保证部署和扩展的顺序。StatefulSet 适用于以下某个或多个需求的应用：

- 稳定的、唯一的网络标识
- 稳定的、持久化的存储
- 有序的、优雅的部署和扩展
- 有序的、优雅的删除和停止

1. 模板示例

```

apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: relay
spec:
  selector:
    matchLabels:
      app: relay
  podManagementPolicy: "Parallel"
  serviceName: "relay"
  replicas: 50
  updateStrategy:
    type: OnDelete
  template:
    metadata:
      labels:
        app: "relay"

```

```
spec:
  schedulerName: tgw-domain-scheduler
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  terminationGracePeriodSeconds: 660
  containers:
    - name: relay-runner
      image: relay:2.0.0
      command: [ "/bin/bash", "-c", "source /tgwenv/tgwenv.sh && /opt/relay-runner/bin/run_server -c /opt/relay-runner/conf/application.conf" ]
      volumeMounts:
        - name: config
          mountPath: /opt/relay-runner/conf
        - name: tgwenv
          mountPath: /tgwenv
  initContainers:
    - name: tgw-init
      image: tgw-init:0.0.3
      command:
        - tgw-init
      args: [--kubeconf=]
      env:
        - name: ENV_PATH
          value: /tgwenv
      volumeMounts:
        - name: tgwenv
          mountPath: /tgwenv
  volumes:
    - name: config
      configMap:
        name: apps-relay-config
    - name: tgwenv
      emptyDir: {}
```

2. 配置项介绍

StatefulSet 和 Deployment 的大部分配置相同，这里就不在介绍相同点(具体参考 Deployment 说明中的配置项介绍)，重点说明 StatefulSet 常用的配置

2.1 Pod 管理策略

`.spec.podManagementPolicy` 有两种：“OrderedReady”和“Parallel”，其中“Parallel”策略告诉 StatefulSet 控制器并行的终止所有 Pod，在启动或终止另一个 Pod 前，不必等待这些 Pod 变成 Running 和 Ready 或者完全终止状态

2.2 initContainers

`initContainers` 并非 StatefulSet 特有的，它同样可以被 Deployment, Daemonset 所使用。这里单独介绍是因为 `initContainers` 在 StatefulSet 中较常用，它可以根据每个 Pod 的“状态”，完成复杂的初始化工作，为后续容器的启动运行提供环境基础。

3. StatefulSet 之“吃豆小游戏”实践

小游戏的后台房间服务用到了 StatefulSet，因为每个房间需要自己的网络标识，所以 Pod 是“有状

态”的。

```
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: rumpetrollv2
  namespace: rumpetroll-v2
spec:
  selector:
    matchLabels:
      app: web
  serviceName: "rumpetroll2"
  replicas: 5
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: pyrumpetroll:0.3
          ports:
            - containerPort: 20000
              name: web
          env:
            - name: DOMAIN
              value: game2-got.o.qcloud.com
            - name: MAX_CLIENT
              value: "2"
            - name: MAX_ROOM
              value: "100"
            - name: REDIS_HOST
              value: redis
            - name: REDIS_PORT
              value: "6379"
            - name: REDIS_DB
              value: "0"
            - name: REDIS_PASSWORD
              value:
            - name: NUMPROCS
              value: "1"
            - name: HOST
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
```

4. BCS 模板集操作

关于 StatefulSet 在 BCS 界面的用法, 请参照[在 K8S 中部署 WordPress](#)。

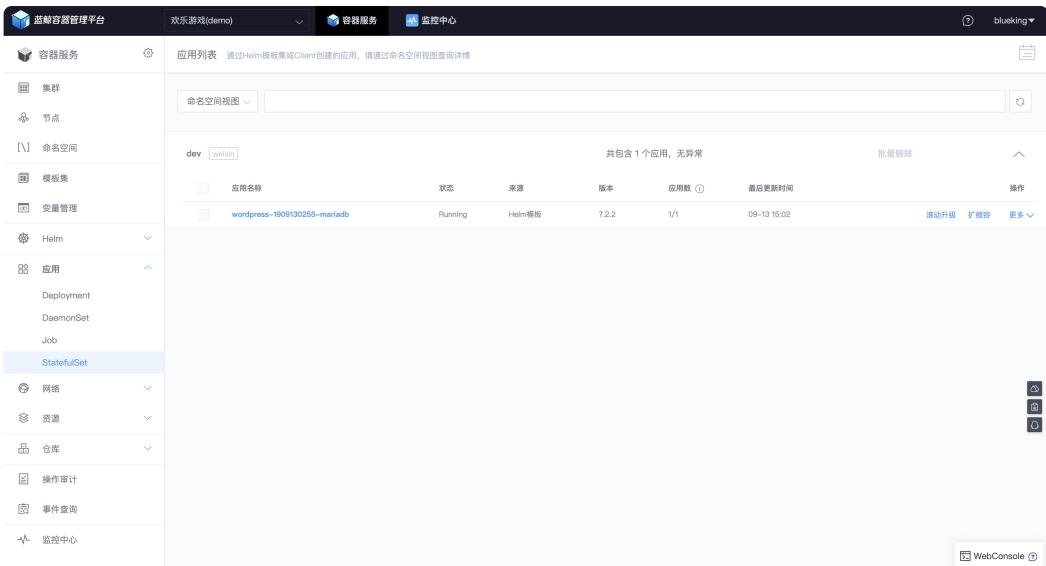
Helm 实例化 WordPress 资源描述文件如下:

```

apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  annotations:
    io.tencent.paus.creator: blueking
    io.tencent.paus.updator: blueking
  labels:
    app: mariadb
    chart: mariadb-6.8.7
    component: mariadb
    heritage: Tiller
    io.tencent.bcs.app.appid: '6'
    io.tencent.bcs.clusterid: BCS-K8S-40015
    io.tencent.bcs.controller.name: wordpress-1909121112-mariadb
    io.tencent.bcs.controller.type: StatefulSet
    io.tencent.bcs.namespace: Kubernetes
    io.tencent.bcs.monitor.level: general
    io.tencent.bcs.namespace: dev
    io.tencent.bkdata.baseall.dataid: '6566'
    io.tencent.bkdata.container.stdlog.dataid: '0'
    io.tencent.paus.projectid: 794b616a01940c2952b497e79a0b967
    io.tencent.paus.source_type: helm
    io.tencent.paus.state: 3
    io.tencent.paus.state_id: 1909121112
    release: wordpress-1909121112
    name: wordpress-1909121112-mariadb
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: mariadb
        component: master
        release: wordpress-1909121112
        service_name: wordpress-1909121112-mariadb
        template:
          metadata:
            labels:
              app: mariadb
              component: master
              io.tencent.bcs.app.appid: '6'

```

部署成功后的实例如下：



Kubernetes Job 说明

Job 负责批量处理短暂的一次性任务，即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束。kubernetes(简称 k8s)支持以下几种 Job：
- 非并行 Job：通常创建一个 Pod 直至其成功结束
- 固定结束次数的 Job：设置 `.spec.completions`，创建多个 Pod，直到 `.spec.completions` 个 Pod 成功结束
- 带有工作队列的并行 Job：设置 `.spec.Parallelism` 但不设置 `.spec.completions`，当所有 Pod 结束并且至少一个成功时，Job 就认为是成功

1. 模板示例

```

apiVersion: batch/v1
kind: Job

```

```
metadata:
  name: ah-buyer-init-auctionhouse
spec:
  template:
    metadata:
      name: ah-buyer-init-auctionhouse
    spec:
      nodeSelector:
        network: private
      containers:
        - name: ah-buyer-init-auctionhouse
          image: solr:5.5.5
          command:
            - sh
            - /opt/ahs/jobs/init-auctionhouse.sh
          volumeMounts:
            - name: initjobconfig
              mountPath: /opt/ahs/jobs
      restartPolicy: OnFailure
      volumes:
        - name: initjobconfig
          configMap:
            name: ah-buyer-init-jobs-config
  backoffLimit: 4
```

2. 配置项介绍

- `.spec.template`: 这里不再介绍(具体参考 Deployment 说明中的配置项介绍)
- `.spec.completions`: 标志 Job 结束需要成功运行的 Pod 个数, 默认为 1
- `.spec.parallelism`: 标志并行运行的 Pod 的个数, 默认为 1
- `.spec.template.spec.containers.restartPolicy`: 只能是"Never"或者"OnFailure"
- `.spec.backoffLimit`: 确定为失败前的重试次数
- `.spec.activeDeadlineSeconds`: 失败 Pod 的重试最大时间, 超过这个时间不会继续重试
(No more pods will be created, and existing pods will be deleted)

Kubernetes DaemonSet 说明

DaemonSet 能够让所有（或者一些特定）的 Node 节点运行同一个 Pod。当有节点加入集群时，也会为他们新增一个 Pod。

当有节点从集群移除时，这些 Pod 也会被回收。

删除 DaemonSet 将会删除它创建的所有 Pod。

使用 DaemonSet 的一些典型用法： - 运行集群存储 daemon，例如在每个节点上运行 glusterd、ceph - 在每个节点上运行日志收集 daemon，例如 fluentd、logstash - 在每个节点上运行监控 daemon，例如 Prometheus Node Exporter、collectd 等

1. 模板示例

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: node-exporter
  namespace: monitoring
  labels:
    name: node-exporter
spec:
  template:
    metadata:
      labels:
        name: node-exporter
    annotations:
      prometheus.io/scrape: "true"
      prometheus.io/port: "9100"
  spec:
    hostPID: true
    hostIPC: true
    hostNetwork: true
    containers:
      - ports:
          - containerPort: 9100
            protocol: TCP
        resources:
          requests:
            cpu: 0.15
        securityContext:
          privileged: true
        image: prom/node-exporter:v0.15.2
        args:
          - --path.procfs
          - /host/proc
          - --path.sysfs
          - /host/sys
          - --collector.filesystem.ignored-mount-points
          - '"^/(sys|proc|dev|host|etc)(\$|/)"'
        name: node-exporter
    volumeMounts:
      - name: dev
        mountPath: /host/dev
      - name: proc
        mountPath: /host/proc
      - name: sys
        mountPath: /host/sys
      - name: rootfs
        mountPath: /rootfs
    volumes:
      - name: proc
        hostPath:
          path: /proc
      - name: dev
        hostPath:
          path: /dev
      - name: sys
        hostPath:
          path: /sys
      - name: rootfs
        hostPath:
          path: /
```

2. 配置项介绍

DaemonSet 和 Deployment 的大部分配置相同，这里就不在介绍相同点(具体参考 Deployment 说明中的配置项介绍)。

由于 DaemonSet 本身的调度机制和 Deployment 有所不同，配置上会带来一些差异，例如 DaemonSet 并没有 `.spec.replicas` 字段，每个节点最多只运行一个 Pod，Pod 的总数取决于调度约束条件(节点个数、nodeSelector、taint 和 toleration 等)。

Kubernetes CronJob 说明

K8S 集群使用 Cron Job 管理基于时间的作业，可以在指定的时间点执行一次或在指定时间点执行多次任务。一个 Cron Job 就好像 Linux crontab 中的一行，可以按照 Cron 定时运行任务。

1. 模板示例

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: recordpopulation
spec:
  concurrencyPolicy: Allow
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - args:
                - /app/cronrunner
                - -host
                - psynet:8080
                - -job
                - /ServiceTask/Population/RecordPopulation
              env:
                - name: TZ
                  value: Asia/Shanghai
              image: cronrunner:tc1
              imagePullPolicy: IfNotPresent
              name: cronrunner
              resources: {}
              terminationMessagePath: /dev/termination-log
              terminationMessagePolicy: File
              dnsPolicy: ClusterFirstWithHostNet
              hostNetwork: true
              restartPolicy: Never
              schedulerName: default-scheduler
              securityContext: {}
              terminationGracePeriodSeconds: 30
  schedule: '*/1 * * * *'
  successfulJobsHistoryLimit: 3
  suspend: false
```

2. 配置项介绍

2.1 调度

.spec.schedule 是 .spec 中必需的字段，它的值是 Cron 格式字的字符串，例如： */1 * * * *
，根据指定的调度时间 Job 会被创建和执行

2.2 并发策略

.spec.concurrencyPolicy 字段也是可选的。它指定了如何处理被 Cron Job 创建的 Job 的并发执行。只允许指定下面策略中的一种：
- Allow（默认）：允许并发运行 Job
- Forbid：禁止并发运行，如果前一个还没有完成，则直接跳过下一个
- Replace：取消当前正在运行的 Job，用一个新的来替换
注意，当前策略只能应用于同一个 Cron Job 创建的 Job。如果存在多个 Cron Job，它们创建的 Job 之间总是允许并发运行。

2.3 挂起

.spec.suspend 字段是可选的。如果设置为 true，后续所有执行都将被挂起。它对已经开始执行的 Job 不起作用。默认值为 false

Kubernetes Service 说明

Service 是 kubernetes(简称 k8s)的一种抽象：一个 Pod 的逻辑分组，一种可以访问它们的策略——通常称为微服务。这一组 Pod 能够被 Service 访问到，通常是通过 Label Selector 实现。

1. 模板示例

```
kind: Service
apiVersion: v1
metadata:
  name: servergame
spec:
  type: NodePort
  selector:
    app: servergame
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: http
    nodePort: 30000
```

2. 配置项说明

2.1 ServiceTypes

ServiceTypes 允许指定一个需要的类型的 Service，默认是 ClusterIP 类型
- ClusterIP：通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部可以访问，这也是默认的 ServiceType。
-

NodePort: 通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求`:`，可以从集群的外部访问一个 NodePort 服务。 - LoadBalancer: 使用云提供商的负载均衡器，可以向外部暴露服务。外部的负载均衡器可以路由到 NodePort 服务和 ClusterIP 服务。 - ExternalName: 通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容（例如，`foo.bar.example.com`）。没有任何类型代理被创建，这只有 Kubernetes 1.7 或更高版本的 kube-dns 才支持。上面的示例中，Service 采用的是 NodePort 类型`.spec.type: NodePort`，同时也指定了服务端口号`nodePort: 30000`（如果不指定，会由系统在 service node range 中随机分配一个）

2.2 Selector

通过`.spec.selector`，将 Service 对象与指定的 Pods 进行关联。如示例中，`servergame` 这一 Service 对象，会将请求代理到使用了 TCP 端口 8080，并且具有 label`app=servergame` 的 Pods 上。

2.3 Ports

`.spec.ports[]` 用来描述 Service 的接收端口与后端 Pod 端口之间的映射关系。Service 能够将一个接收端口映射到任意的 targetPort。默认情况下，targetPort 将被设置为与 port 字段相同的值。targetPort 可以是 Pod 的端口号，也可以是一个字符串（引用了 backend Pod 的一个端口的名称）。

`.spec.ports[i].protocol` 能够支持 TCP 和 UDP 协议，默认 TCP 协议

3. Service 之“吃豆小游戏”实践

下面是 bcs 小游戏中，Service 用法的示例：rumpetroll2 用于暴露后端小游戏房间的服务，redis 用于暴露 redis 的服务。其中，rumpetroll2 会将请求代理到具有 label`app=web` 的 Pods 上，`clusterIP: None` 表示 rumpetroll2 是 Headless Service，适用于 statefulset 管理的 Pods。

```
apiVersion: v1
kind: Service
metadata:
  name: rumpetroll2
  namespace: rumpetroll-v2
  labels:
    app: rumpetroll-v2
spec:
  ports:
    - port: 80
      name: web
  clusterIP: None
  selector:
    app: web
```

```
apiVersion: v1
kind: Service
metadata:
  name: redis
  namespace: rumpetroll-v2
spec:
```

```

selector:
  app: redis-v2
ports:
- name: default
  protocol: TCP
  port: 6379
  targetPort: 6379

```

4. BCS 模板集操作

关于 Service 的实战演练, 请参照 [快速构建 Nginx 集群](#)。

kubernetes Ingress 说明

Ingress 是管理外部访问集群内服务(典型的如 HTTP)的 API 对象。Ingress 可配置提供外部可访问的 URL、负载均衡、SSL、基于名称的虚拟主机等。用户通过 POST Ingress 资源到 API server 的方式来请求 Ingress。Ingress controller 负责实现 Ingress, 通常使用负载均衡器(如常用的 nginx-controller)。

1. 模板示例

```

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: servergame
spec:
  tls:
  - hosts:
    - demo.bcs.com
    secretName: servergame-secret
  rules:
  - host: demo.bcs.com
    http:

```

```
paths:  
- backend:  
  serviceName: servergame  
  servicePort: 8080  
  path: /
```

2. 配置项介绍

2.1 基于名称的虚拟主机

通过 `.spec.rules[]` 来设置基于名称的虚拟主机，如示例中的 `host: fwx.ffmpeg.com`，所有 `https://demo.bcs.com:443/` 的请求都会被转发给名称是 `servergame` 的 Service 后端所关联的 Pods

2.2 TLS

通过指定包含 TLS 私钥和证书的 Secret 可以加密 Ingress。目前，Ingress 仅支持单个 TLS 端口 443，并假定 TLS termination。如示例中，利用 `.spec.tls` 给域名 `demo.bcs.com` 绑定了名为 `servergame-secret` 的 Secret 的 TLS 证书。`servergame-secret` 的示意配置如下

```
apiVersion: v1  
data:  
  tls.crt: base64 encoded cert  
  tls.key: base64 encoded key  
kind: Secret  
metadata:  
  name: servergame-secret  
type: Opaque
```

3. BCS 模板集操作

关于 Ingress 的实战演练，请参照 [应用的蓝绿发布](#)。

kubernetes ConfigMap 说明

ConfigMap 是用来存储配置文件的 kubernetes(简称 k8s)资源对象，它的作用是将配置文件从容器镜像中解耦，从而增强容器应用的可移植性。在一个 Pod 里面使用 ConfigMap 主要有两种方式： - 环境变量 - 数据卷文件

1. 模板示例

1.1 示例一：环境变量

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: external-config
data:
  AUX_CONNECTION_STRING: jdbc:mysql://demo.bcs.com:10000/db_ffm_aux
  PUBLIC_ASSETS_URL: https://demo.bcs.com/wxlive
```

全导入用法

```
envFrom:
- configMapRef:
    name: external-config
```

或者部分引用

```
env
- name: PUBLIC_ASSETS_URL
  valueFrom:
    configMapKeyRef:
      name: external-config
      key: PUBLIC_ASSETS_URL
```

1.2 示例二：数据卷文件

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: apps-servergame-tlog-config
data:
  logstash.conf: |->
    input {
      file {
        path => "/opt/servergame/log/tlog.log"
        type => "tlog"
        start_position => "beginning"
      }
    }

    filter { }
```

```
output {  
}
```

引用方法

```
volumes:  
- name: config  
  configMap:  
    name: apps-servergame-tlog-config
```

2. 配置项介绍

ConfigMap 的配置数据存储在 `data` 字段中，具体参考示例模板

3. BCS 模板集操作

关于 ConfigMap 的实战演练，请参照 [应用的蓝绿发布](#)。

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with navigation links like '容器服务', '集群', '节点', '命名空间', '模板集' (which is highlighted), '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 content area has tabs for 'Deployment', 'Service', 'Configmap' (which is selected and highlighted with a red box), 'Secret', 'DaemonSet', 'Job', 'StatefulSet', and 'Ingress'. Below these tabs, there's a section for 'Configmap' with a sub-section for 'nginx-conf-blue'. It shows fields for '名称' (name: nginx-conf-blue) and '键' (key: nginx.conf). The '值' (value) field contains the following YAML code:

```
user nginx;  
worker_processes 2;  
  
error_log /var/log/nginx/error.log warn;  
pid /var/run/nginx.pid;  
  
events {
```

kubernetes Secret 说明

kubernetes(简称 k8s)中的 Secret 资源可以用来存储密码、Token、秘钥等敏感数据。将这些敏感信息保存在 Secret 中，相对于暴露到 Pod、镜像中更加的安全和灵活。k8s 内置了三种类型的 Secret - Service Account Secret - Opaque Secret - kubernetes.io/dockerconfigjson

1. 模板示例

1.1 Service Account Secret

为了能从集群内部访问 k8s API，k8s 提供了 Service Account 资源。Service Account 会自动创建和挂载访问 k8s API 的 Secret，会挂载到 Pod

的 `/var/run/secrets/kubernetes.io/serviceaccount` 目录中。这种类型的 Secret 通常不需要用户显示定义，所以这里不展开

1.2 Opaque Secret

Opaque 类型的 Secret 是一个 map 结构(key-value)，其中 value 要求以 base64 格式编码

```
apiVersion: v1
kind: Secret
metadata:
  name: external-secret-config
type: Opaque
data:
  RDS_USERNAME: YWRtaW4=
  RDS_PASSWORD: MWYyZDF1MmU2N2Rm
```

引用方式

```
envFrom:
- secretRef:
  name: external-secret-config
```

1.3 kubernetes.io/dockerconfigjson

用于 docker registry 认证的 secret

```
apiVersion: v1
data:
  .dockercfg: eyJjY3IuY2NzLnRlbmNlbnR5dW4uY29tL3RlbmNlbnR5dW4iOnsidXNlc5hbWUiOiIzMzIxM
  zM3OTk0IiwicGFzc3dvcmQiOiIxMjM0NTYuY29tIiwizW1haWwiOiIzMzIxMzM3OTk0QHFxLmNvbSIsImF1dGgi
  OiJNek15TVRNek56azVORG94TWpNME5UWXVZMj10In19
kind: Secret
metadata:
  name: paas.image.registry.default
type: kubernetes.io/dockerconfigjson
```

引用方法

```
spec:
  imagePullSecrets:
  - name: paas.image.registry.default
```

2. 配置项介绍

`type` 字段用于指定 Secret 的类型，`data` 字段存储数据信息(value 需要 base64 编码)

3. BCS 模板集操作

关于 Secret 的使用，请参照 [在 K8S 中部署 WordPress](#)：

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。

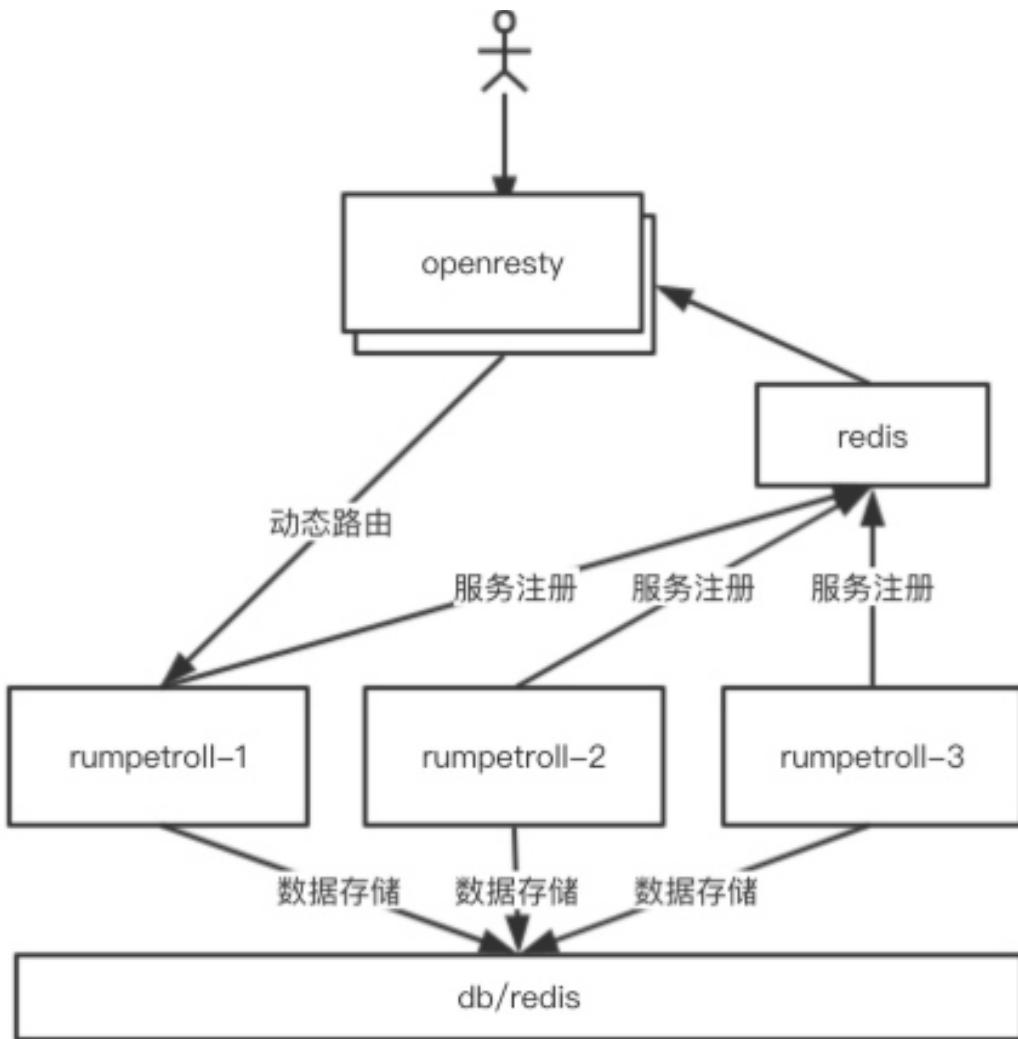
```
# kubectl get secret -n dev
NAME          TYPE
DATA    AGE
wordpress-1909130255   Opaque
1        4h37m

# 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
1X3NfoHrHV
```

吃豆小游戏案例

小游戏介绍



分为 3 个模块，openresty、rumpetroll 和 redis。其中 openresty 作为游戏的接入模块，rumpetroll 是游戏房间(游戏后台)，redis 是数据存储与服务发现模块。

模块示例

基于 Deployment 部署 openresty

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: openresty-v2
  namespace: rumpetroll-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: openresty-v2
  template:
    metadata:
      labels:
        app: openresty-v2
  
```

```
spec:
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  hostAliases:
  - ip: 127.0.0.1
    hostnames:
    - "game2-got.o.qcloud.com"
  containers:
  - name: openresty
    image: rumpetroll-openresty:0.61
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
      hostPort: 80
      protocol: TCP
    env:
    - name: DOMAIN
      value: game2-got.o.qcloud.com
    - name: MAX_CLIENT
      value: "2"
    - name: MAX_ROOM
      value: "100"
    - name: REDIS_HOST
      value: "redis"
    - name: NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: REDIS_PORT
      value: "6379"
    - name: REDIS_DB
      value: "0"
    - name: REDIS_PASSWORD
      value: ""
```

openresty 模块直接使用主机网络，对外提供访问服务

基于 Statefulset 部署 rumpetroll

```
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: rumpetrollv2
  namespace: rumpetroll-v2
spec:
  selector:
    matchLabels:
      app: web
  serviceName: "rumpetroll2"
  replicas: 5
  template:
    metadata:
      labels:
        app: web
  spec:
    containers:
    - name: web
      image: pyrumpetroll:0.3
      ports:
      - containerPort: 20000
        name: web
    env:
```

```
- name: DOMAIN
  value: game2-got.o.qcloud.com
- name: MAX_CLIENT
  value: "2"
- name: MAX_ROOM
  value: "100"
- name: REDIS_HOST
  value: redis
- name: REDIS_PORT
  value: "6379"
- name: REDIS_DB
  value: "0"
- name: REDIS_PASSWORD
  value:
- name: NUMPROCS
  value: "1"
- name: HOST
  valueFrom:
    fieldRef:
      fieldPath: status.podIP
```

```
apiVersion: v1
kind: Service
metadata:
  name: rumpetroll2
  namespace: rumpetroll-v2
  labels:
    app: rumpetroll-v2
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: web
```

由于每个房间有自己的标识，是有"状态的"，因此采用 StatefulSet 方式部署；同时配置 Headless Service，提供给 openresty 模块访问

基于 Deployment 部署 redis

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-v2
  namespace: rumpetroll-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-v2
  template:
    metadata:
      labels:
        app: redis-v2
    spec:
      containers:
      - name: redis-v2
        image: redis:1.0
        imagePullPolicy: IfNotPresent
```

```
apiVersion: v1
kind: Service
metadata:
  name: redis
  namespace: rumpetroll-v2
spec:
  selector:
    app: redis-v2
  ports:
  - name: default
    protocol: TCP
    port: 6379
    targetPort: 6379
```

redis 服务模块采用 Deployment 方式部署，并创建与其关联的 Service。Service 的 Cluster IP 会被注册到 dns 中，方便 openresty 和 rumpetroll 模块从集群内访问服务

应用列表说明文档

蓝鲸容器服务应用列表分两个维度展示：模板集维度和命名空间维度；其中，模板集维度主要展示由【模板集】实例化的应用；命名空间维度展示了集群下所有所有命名空间下的所有应用，来源包含模板集、Helm、Client 实例化的应用；以便用户查看和操作应用及查看应用下容器运行详情及资源使用详情。

- 应用类型 deployment/daemonset/job/statefuleset

查询展示

- 模板及应用展示

The screenshot shows the BlueKing Container Service application list interface. At the top, there are filters for '集群模板视图' (Cluster Template View), '集群类型' (Cluster Type) set to '测试集群' (Test Cluster), and a search bar. Below the filters, a section titled '小游戏部署模板' (Game Application Deployment Template) shows one application template. A table lists the application details:

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
deploy-redis1	v9 Running	Deployment	正常 yanzhen	v9	1/1	11-05 11:55	批量删除 实例化 滚动升级 扩缩容 更多

At the bottom, another section titled 'game-nemo部署模板' (Game-nemo Application Deployment Template) shows one application template.

- 应用详情

[deploy-redis1](#)[to yaml](#)

应用名称: deploy-redis1	创建时间: 2018-11-05 11:55:14	更新时间: 2018-11-05 11:55:14	所在命名空间: yanzhen	所属集群: bellke-test	
------------------------	------------------------------	------------------------------	--------------------	----------------------	--

CPU使用率



内存使用率



Pod管理事件标签备注Metric信息

[+/-] deploy-redis1-6656bbd895-jms...	状态: Running	Host IP: 10.223.49.220	Pod IP: 10.233.70.45	存活时间: 4天4小时18分6秒	重新调度
---	--	------------------------	----------------------	------------------	----------------------

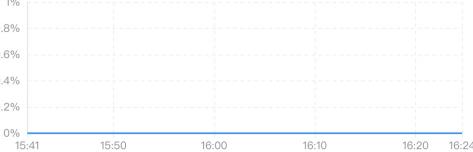
● 容器详情

主机名称: ip-10-223-49-...	主机IP: 10.223.49.220	容器IP: 10.233.70.45	容器ID: 3b4b1ac62b093...	镜像: test.artifactory....	网络模式: ClusterFirst
---------------------------	------------------------	-----------------------	---------------------------	-----------------------------	-----------------------

CPU使用率



内存使用率



网络使用率



存储使用率



端口映射命令挂载卷环境变量健康检查标签资源限制

Name	Host Port	Container Port	Protocol
------	-----------	----------------	----------

操作相关

● 滚动升级

注：允许应用类型包含 `deployment/daemonset/job/statefulset`

应用列表

部署模板视图 集群类型 测试集群

小游戏部署模板 共包含 1 个应用模板

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
deploy-redis1	Running	deployment	yanzhen	v9	1/1	11-05 11:55	滚动升级 扩缩容 更多

game-nemo部署模板 共包含 1 个应用模板

- 扩缩容

实例数量增加或减少

应用列表

部署模板视图 集群类型 测试集群

小游戏部署模板 共包含 1 个应用模板

deploy-redis1 扩缩容

实例数量
1

操作

确定 取消

- 重建应用

重建包含两个步骤: 删除当前应用, 然后以现有配置再创建应用(类似进程重启)

应用列表

部署模板视图 集群类型 测试集群

小游戏部署模板 共包含 1 个应用模板

deploy-redis1 v9 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
deploy-redis1	Running	deployment	yanzhen	v9	1/1	11-05 11:55	滚动升级 扩缩容 更多

更多
重建
删除

game-nemo部署模板 共包含 1 个应用模板

- 删除应用

应用列表

小游戏部署模板 共包含 1 个应用模板

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
deploy-redis1	Running	deployment	yanzhen	v9	1/1	11-05 11:55	滚动升级 扩缩容 更多

game-nemo部署模板 共包含 1 个应用模板

● 重新调度

← deploy-redis1 << to yaml

应用名称:	创建时间:	更新时间:	所在命名空间:	所属集群:
deploy-redis1	2018-11-05 11:55:14	2018-11-05 11:55:14	yanzhen	bellke-test

CPU使用率

内存使用率

重新调度

Pod管理	事件	标签	备注	Metric信息
+ deploy-redis1-6656bbd895-jms...	状态: Running	Host IP: 10.223.49.220	存活时间: 4天5小时14分51秒	重新调度

K8S 使用技巧

如何与集群中的资源进行比较

- 场景

- 已有 K8S 集群，并部署有业务容器，并希望尽量降低变更风险
- 怀疑手动改了集群的一些资源配置

在 K8S 1.14 中新增了 `kubectl diff` 命令，支持将线下 YAML 文件与线上环境做比对。

情景：将本地 `deployment.yaml` 中的 Nginx 的镜像从 1.17.0 修改为 1.17.1，与线上环境比对，结果如下：

```
# kubectl diff -f deployment.yaml
diff -u -N /tmp/LIVE-740052839/extensions.v1beta1.Deployment.default.bk-bcs-test /tmp/M
ERGED-190542234/extensions.v1beta1.Deployment.default.bk-bcs-test
```

```

--- /tmp/LIVE-740052839/extensions.v1beta1.Deployment.default.bk-bcs-test      2019-09
-09 16:11:26.933501898 +0800
+++ /tmp/MERGED-100542234/extensions.v1beta1.Deployment.default.bk-bcs-test      2019-09
-09 16:11:26.940501902 +0800
@@ -6,7 +6,7 @@
     kubectl.kubernetes.io/last-applied-configuration: |
       {"apiVersion": "extensions/v1beta1", "kind": "Deployment", "metadata": {"annotations": {}, "name": "bk-bcs-test", "namespace": "default"}, "spec": {"template": {"metadata": {"labels": {"app": "bk-bcs-test"}}, "spec": {"containers": [{"image": "nginx:1.17.0", "name": "nginx"}]} }}}
       creationTimestamp: "2019-09-09T07:40:57Z"
-  generation: 1
+  generation: 2
   labels:
     app: bk-bcs-test
     name: bk-bcs-test
@@ -33,7 +33,7 @@
     app: bk-bcs-test
   spec:
     containers:
-      - image: nginx:1.17.0
+      - image: nginx:1.17.1
        imagePullPolicy: IfNotPresent
        name: nginx
        resources: {}
exit status 1

```

bcs application 配置说明

bcs application 实现 Pod 的含义，并与 k8s 的 RC，Mesos 的 app 概念等价。

配置模板说明

```
{
  "apiVersion": "v4",
  "kind": "application",
  "restartPolicy": {
    "policy": "Never | Always | OnFailure",
    "interval": 5,
    "backoff": 10,
    "maxtimes": 10
  },
  "killPolicy": {
    "gracePeriod": 10
  },
  "constraint": {
    "intersectionItem": [
      {
        "unionData": [
          {
            "name": "ip-resources",
            "operate": "GREATER",
            "type": 1,
            "scalar": {
              "value": 0
            }
          }
        ]
      }
    ],
  }
}
```

```
{
    "unionData": [
        {
            "name": "label",
            "operate": "EXCLUDE",
            "type": 4,
            "set": {
                "item": [
                    "appname:gamesvr",
                    "appname:dbsvr"
                ]
            }
        }
    ],
    {
        "unionData": [
            {
                "name": "hostname",
                "operate": "CLUSTER",
                "type": 4,
                "set": {
                    "item": [
                        "slave3",
                        "slave4",
                        "slave5"
                    ]
                }
            }
        ]
    },
    {
        "unionData": [
            {
                "name": "district",
                "operate": "GROUPBY",
                "type": 4,
                "set": {
                    "item": [
                        "shanghai",
                        "shenzhen",
                        "tianjin"
                    ]
                }
            }
        ]
    },
    {
        "unionData": [
            {
                "name": "hostname",
                "operate": "LIKE",
                "type": 3,
                "text": {
                    "value": "slave[3-5]"
                }
            }
        ]
    },
    {
        "unionData": [
            {
                "name": "hostname",
                "operate": "UNLIKE",
                "type": 3,
                "text": {
                    "value": "slave[3-5]"
                }
            }
        ]
    }
}
```

```
        "text": {
            "value": "slave[1-2]"
        }
    }
},
{
    "unionData": [
        {
            "name": "hostname",
            "operate": "UNIQUE"
        }
    ],
    {
        "unionData": [
            {
                "name": "idc",
                "operate": "MAXPER",
                "type": 3,
                "text": {
                    "value": "5"
                }
            }
        ]
    }
],
"metadata": {
    "labels": {
        "io.tencent.bcs.netsvc.requestip.0": "10.168.1.1",
        "io.tencent.bcs.netsvc.requestip.1": "10.168.2.1",
        "io.tencent.bcs.netsvc.requestip.2": "10.168.1.3",
        "test_label": "test_label"
    },
    "name": "ri-test-rc-001",
    "namespace": "nfsol"
},
"spec": {
    "instance": 1,
    "template": {
        "metadata": {
            "labels": {
                "test_label": "test_label"
            }
        },
        "spec": {
            "containers": [
                {
                    "hostname": "container-hostname",
                    "command": "bash",
                    "args": [
                        "args1",
                        "args2"
                    ],
                    "parameters": [
                        {
                            "key": "rm",
                            "value": "false"
                        }
                    ],
                    "type": "MESOS",
                    "env": [
                        {
                            "name": "test_env",
                            "value": "true"
                        }
                    ]
                }
            ]
        }
    }
}
```

```
        "value": "test_env"
    }
],
"image": "hub_address.com/nfsol/log:92763",
"imagePullUser": "userName",
"imagePullPasswd": "passwd",
"imagePullPolicy": "Always|IfNotPresent",
"privileged": false,
"ports": [
{
    "containerPort": 8090,
    "hostPort": 8090,
    "name": "test-tcp",
    "protocol": "TCP"
},
{
    "containerPort": 8080,
    "hostPort": 8080,
    "name": "test-http",
    "protocol": "http"
}
],
"healthChecks": [
{
    "type": "HTTP|TCP|COMMAND|REMOTE_HTTP|REMOTE_TCP",
    "delaySeconds": 10,
    "intervalSeconds": 60,
    "timeoutSeconds": 20,
    "consecutiveFailures": 3,
    "gracePeriodSeconds": 300,
    "command" : {
        "value": ""
    },
    "http": {
        "port": 8080,
        "portName": "test-http",
        "scheme": "http|https",
        "path": "/check",
        "headers": {
            "key1": "value1",
            "key2": "value2"
        }
    },
    "tcp": {
        "port": 8090,
        "portName": "test-tcp"
    }
}
],
"resources": {
    "limits": {
        "cpu": "2",
        "memory": "8"
    },
    "requests": {
        "cpu": "2",
        "memory": "8"
    }
},
"volumes": [
{
    "volume": {
        "hostPath": "/data/host/path",
        "mountPath": "/container/path",
        "readOnly": false
    }
}
```

```

        },
        "name": "test-vol"
    }
],
"secrets": [
{
    "secretName": "mySecret",
    "items": [
        {
            "type": "env",
            "dataKey": "abc",
            "keyOrPath": "SRECT_ENV"
        },
        {
            "type": "file",
            "dataKey": "abc",
            "keyOrPath": "/data/container/path/filename.con
f",
            "subPath": "relativedir/",
            "readOnly": false,
            "user": "user00"
        }
    ]
},
"configmaps": [
{
    "name": "template-configmap",
    "items": [
        {
            "type": "env",
            "dataKey": "config-one",
            "keyOrPath": "SECRET_ENV"
        },
        {
            "type": "file",
            "dataKey": "config_two",
            "dataKeyAlias": "config-two",
            "keyOrPath": "/data/contianer/path/filename.txt
",
            "readOnly": false,
            "user": "root"
        }
    ]
}
],
"networkMode": "USER",
"networkType": "cni",
"netLimit": {
    "egressLimit": 100
}
}
}
}

```

KillPolicy 机制

gracePeriod: 宽限期描述在强制 kill container 之前等待多久，单位秒。默认为 1

RestartPolicy 机制

- policy: 支持Never Always OnFailure三种配置(默认为OnFailure),OnFailure表示在失败的情况下重新调度,Always表示在失败和Lost情况下重新调度, Never表示任何情况下不重新调度
- interval: 失败后到执行重启的间隔(秒),默认为0
- backoff: 多次失败时,每次重启间隔增加秒,默认为0.如果interval为5,backoff为10,则首次失败时5秒后重新调度,第二次失败时15秒后重新调度,第三次失败时25秒后重新调度
- maxtimes: 最多重新调度次数,默认为0表示不受次数限制.容器正常运行30分钟后重启次数清零
重新计算

constraint 调度约束

constraint 字段用于定义调度策略

- IntersectionItem

该字段为数组, 策略为: 所有元素同时满足时才进行调度。使用多个 IntersectionItem 可以同时满足多个条件限制。

每个 IntersectionItem 中支持多个 UnionData 为规则细节字段, 用于填写调度匹配规则, 也为数组, 该数组含义为: 所有元素只需要满足一个即可。使用 UnionData 可以实现多个条件满足其一即可。

UnionData 字段说明

```
{  
    "name": "hostname",  
    "operator": "CLUSTER",  
    "type": 3|4,  
    "set": {  
        "item": ["mesos-slave-1", "mesos-slave-2", "mesos-slave-3"]  
    },  
    "text":{  
        "value": "mesos-slave-1"  
    }  
}
```

- name: 调度字段 key (如主机名, 主机类型, 主机 IDC 等) 用来调度的字段需要 mesos slave 通过属性的方式进行上报, hostname 参数自动获取无需属性上报
- operator: 调度算法, 当前支持以下 5 种调度算法 (大写) :
- UNIQUE: 每个实例的 name 的取值唯一: 如果 name 为主机名则表示每台主机只能部署一个实例, 如果 name 为 IDC 则表示每个 IDC 只能部署一个实例。UNIQUE 算法无需参数。
- MAXPER: name 同一取值下最多可运行的实例数, 为 UNIQUE 的增强版 (数量可配置), MAXPER 算法需通过参数 text(type 为 3)指定最多运行的实例数。
- CLUSTER: 配合 set 字段 (type 为 4) , 要求 name 的取值必须是 set 中一个, 可以限定实例

部署在 name 的取值在指定 set 范围。

- LIKE: 配合 text 字段 (type 为 3) 或者 set 字段 (type 为 4), name 与 text (或者 set) 中的内容进行简单的正则匹配, 可以限定实例部署时 name 的取值。如果是参数是 set (type 为 4), 只要和 set 中某一个匹配即可。
- UNLIKE: LIKE 取反。如果是参数为 set (type 为 4), 必须和 set 中所有项都不匹配。
- GROUPBY: 根据 name 的目标个数, 实例被均匀调度在目标上, 与 set 一起使用, 如果实例个数不能被 set 的元素个数整除, 则会存在差 1 的情况, 例如: name 为 IDC, 实例数为 3, set 为 ["idc1", "idc2"], 则会在其中一个 idc 部署两个实例。
- EXCLUDE: 和具有指定标签的 application 不部署在相同的机器上, 即: 如果该主机上已经部署有这些标签 (符合一个即可) 的 application 的实例, 则不能部署该 application 的实例。目前 name 只支持 "label", label 的 k:v 在 set 数组中指定。
- GREATER: 配合 scalar 字段 (type 为 1), 要求 name 的取值必须大于 scalar 的值。
- type: 参数的数据类型, 决定 operator 所操作 key 为 name 的值的范围

1: scalar: float64

2: text: 字符串。

3: set: 字符串集合。

案例说明:

- 要求各实例运行在不同主机上

```
{  
    "name": "hostname",  
    "operate": "UNIQUE"  
}
```

- 要求实例限制运行在深圳和东莞地区 (要求 slave 上报 section 字段)

```
{  
    "name": "section",  
    "operate": "CLUSTER",  
    "type": 4,  
    "set": {  
        "item": ["shenzhen", "dongguan"]  
    }  
}
```

- 要求实例运行在 mesos-slave-1, mesos-slave-2 上

```
{  
    "name": "hostname",
```

```
        "operate": "LIKE",
        "type": 3,
        "text": {
            "value": "mesos-slave-[1-2]"
        }
    }
```

meta 元数据

- name: Application 名字，小写字母与数字构成，但不能完全由数字构成，不能数字开头
- namespace: App 命名空间，小写字母与数字构成，不能完全由数字构成，不能数字开头；不同业务必然不同，默认值为 defaultGroup
- label: app 的 label 信息，对应 k8s RC label
 - io.tencent.bcs.cluster: 用于标识集群 Id 信息
 - io.tencent.bcs.app.appid: 业务 ccID
 - io.tencent.bcs.app.setid: 业务 cc 大区 ID
 - io.tencent.bcs.app.moduleid: 业务 cc 模块 ID

当容器网络使用 bcs-cni 方案的时，如果想针对容器指定 IP，可以使用以下 label

- io.tencent.bcs.netsvc.requestip.[i]: 针对 Pod 申请 IP，i 代表 Pod 的实例，从 0 开始计算

当容器指定 ip 时，不同的 taskgroup 需要调度到特定的宿主机上面，宿主机的制定方式与 constraint 调度约束一致，如下是使用 innerIp:

- io.tencent.bcs.netsvc.requestip.[i]: "10.168.1.1|innerIp=10.158.49.13;10.158.49.12"

使用分隔符"|"分隔，"|"前面为容器 ip，后面为需要调度到的宿主机 ip，多个宿主机之间使用分隔符";"分隔，宿主机 ip 支持正则表示式，方式如下：

- io.tencent.bcs.netsvc.requestip.[i]: "10.168.1.1|innerIp=10.158.49.[12-25];10.158.48.[11-13]"

容器字段信息

- instance: 运行实例个数
- label: 运行时容器 label 信息，对应 k8s pod label
- name: pod 名字，mesos 中不启用
- type: DOCKER/MESOS
- hostname: 容器的 hostname 设置，如果网络模式为"HOST"，则该字段无效
- command: 字符串，容器启动命令，例如/bin/bash

- args: command 的参数, 例如 ["-c", "echo hello world"]
- env: key/value 格式, 环境变量。针对 BCS 默认注入的环境变量 (BCS_CONTAINER_IP, BCS_NODE_IP) 支持赋值操作
- parameters: docker 参数, 当前以下 docker 参数已支持
- oom-kill-disable: 有效值 true, 设置为 true 后, 如果容器资源超限会进行强杀
- ulimit: 可以设置 ulimit 参数, 例如 core=-1
- rm: 有效值为 true, 容器退出后, 是否直接删除容器
- image: 镜像链接
- imagePullSecrets: 存储仓库鉴权信息的 secret 名字
- imagePullPolicy: 拉取容器策略
 - Always: 每次都重新从仓库拉取
 - IfNotPresent: 如果本地没有, 则尝试拉取 (默认值)
 - privileged: 容器特权参数, 默认为 false
- resources: 容器使用资源
 - limits.cpu: 字符串, 可以填写小数, 1 为使用 1 核
 - limits.memory: 内存使用, 字符串, 单位默认为 M. 注意: 当 memory >= 4Mb, 使用 memory 的值限制内存; 否则, 不对 memory 做 limits
 - limits.storage: 磁盘使用大小, 默认单位 M
 - request 仅对 k8s 生效
- networkMode: 网络模式
 - HOST: docker 原生网络模式, 与宿主机共用一个 Network Namespace, 此模式下需要自行解决网络端口冲突问题
 - BRIDGE: docker 原生网络模式, 此模式会为每一个容器分配 Network Namespace、设置 IP 等, 并将一个主机上的 Docker 容器连接到一个虚拟网桥上, 通过端口映射的方式对外提供服务
 - NONE: 除了 lo 网络之外, 不配置任何网络
 - USER: 用户深度定制的网络模式, 支持 macvlan、calico 等网络方案
- networkType: 只有在 networkMode 为 USER 模式下, 该字段才有效
- cnf(小写): 使用 bcs 提供的 cnf 来构建网络, 具体 cnf 的类型是由配置决定
- 空或其它值: 使用 docker 原生或用户自定义的方式来构建网络

netLimit 说明

对容器的网络流量进行限制，包括 ingress(进流量)和 egress(出流量)，默认情况下不限制。暂时只支持对 egress 的限制。

- egressLimit: 容器出流量的限制，value 为大于 0 的整数，单位为 Mbps。

Volume 说明

支持主机磁盘挂载

- name: 挂载名，在 app 中需要保持唯一
- volume.hostPath: 主机目录
- 当目录没填写时，默认为创建一个随机目录，该目录 Pod 唯一
- 该目录路径可以支持变量\$BCS POD ID
- volume.mountPath: 需要挂载的容器目录，需要其父目录存在，否则报错
- readOnly: true/false, 是否只读，默认 false

ConfigMap 说明

主要功能是引用 configmap 数据，并作为环境变量/文件注入容器中。

- name: configmap 索引名字

环境变量注入

- items[x].type: env
- items[x].dataKey: configmap 子项索引名
- items[x].keyOrPath: 需要注入环境变量名

文件注入

- items[x].type: file
- items[x].dataKey: configmap 子项索引名，默认为文件名
- items[x].dataKeyAlias: 对于 k8s configmap，如果原始文件名带有"_"，则需要对文件进行重命名，使用 keyAlias 对子项进行索引。也可以增加前缀目录，keyAlias 拼接默认构成完成容器路径。
- items[x].keyOrPath: 文件在容器中路径，需要保证父目录存在
- items[x].readOnly: true/false, 默认 false, 文件是否只读
- items[x].user: 文件用户设置，默认 root，k8s 不生效

0 Secrets 机制

secret 在 k8s 和 mesos 中实现存在差异。在 k8s 中，即为默认支持的 secret 数据，并存储在 etcd 中；在 mesos 中，secret 为 bcs-scheduler 增加的数据结构，数据默认存储在 vault 中，读写控制需要通过 bcs-authserver。secrets 的数据默认可以注入环境变量/文件。

在 k8s 中，secret 只能存储一项数据，所以不存在子项数据结构。mesos 下，一个 secret 可以存储多项数据。

- secretName: 引用的 secret 名字

注入环境变量：

- items[x].type: env
- items[x].dataKey: secret 中子项索引，mesos 有效
- items[x].keyOrPath: 环境变量 KEY

注入文件

- items[x].type: file
- items[x].dataKey: secret 中子项索引，mesos 有效
- items[x].keyOrPath: 需要挂载的容器目录
- items[x].subPath: 需要挂载子目录，仅 k8s 有效
- items[x].readOnly: true/false，文件是否只读，默认为 false
- items[x].user: user00，文件属主，mesos 有效

1 容器 Ports 机制说明

ports 字段说明：

- protocol: 协议，必填，http, tcp, udp
- name: 标识 port 信息，唯一，必填
- containerPort: 容器中服务使用端口，必填
- hostPort: 物理主机使用的端口，0 代表 scheduler 进行随机选择

特别说明：

- host 模式下，containerPort 即代表 hostPort
- 填写固定端口，需要业务自行确认是否产生冲突
- 填写 0，意味着 scheduler 进行随机选择

- bridge 模式下, hostPort 代表物理主机上的端口
- hostPort 填写固定端口, 业务自行解决冲突的问题
- 填写 0, scheduler 默认进行端口随机

端口随机的状态下, scheduler 会根据 ports 字段序号, 生成 PORT0 ~ n 的环境变量, 以便业务读取该随机端口。不支持 PORT_NAME 的方式

容器 Health Check 机制说明

通过 mesos 协议下发检测机制到 executor, 通过 executor 执行检测

- 支持的类型为 HTTP,TCP 和 COMMAND 三种
- scheduler 根据 application 定义的检测机制, 启动进程时下发到 executor
- executor 根据检测配置实施检测 (并在多次检测失败的情况下 kill 进程, 可配置)
- executor 将检测结果通过 TaskStatus 中的 healthy (bool) 上报到 scheduler
- scheduler 根据 healthy 的值以及进程的其他数据(配置数据和动态数据)来确定后续行为:
- 状态修改, 数据记录, 触发告警等
- 重新调度

mesos scheduler 根据检测机制直接远程执行检测

- 支持的类型为 REMOTE_HTTP,REMOTE_TCP

health check Type 说明

- health check 可以同时支持多种类型的 check, 目前最多为三种
- HTTP,TCP 和 COMMAND 三种类型, 最多只能同时支持一种
- REMOTE_HTTP,REMOTE_TCP 两种类型可以同时支持

healthChecks 字段说明

- type: 检测方式, 目前支持 HTTP,TCP,COMMAND,REMOTE_TCP 和 REMOTE_HTTP 五种
- delaySeconds: 容器启动之后到开始进行健康检测的等待时长(mesos 协议中有,marathon 协议中不支持,因为有 gracePeriodSeconds,该参数好像意义不大,可能被废弃)
- intervalSeconds: 前后两次执行健康监测的时间间隔.
- timeoutSeconds: 健康监测可允许的等待超时时间。在该段时间之后, 不管收到什么样的响应, 都被认为健康监测是失败的, **timeoutSeconds** 需要小于 **intervalSeconds**

- consecutiveFailures: 在一个不健康的任务被杀掉之前, 连续的健康监测失败次数, 如果值设为 0, tasks 如果不通过健康监测, 则它不会被杀掉。进程被杀掉后 scheduler 根据 application 的 restartpolicy 来决定是否重新调度. marathon 协议中为 maxConsecutiveFailures
- gracePeriodSeconds: 启动之后在该时段内健康监测失败会被忽略。或直到任务首次变成健康状态.
- command: type 为 COMMAND 时有效
- value: 需要执行的命令,value 中支持环境变量.mesos 协议中区分是否 shell,这里不做区分,如果为 shell 命令,需要包括"/bin/bash - c",系统不会自动添加(参考 marathon)
- 后续可能需要补充其他参数如 USER
- http: type 为 HTTP 和 REMOTE_HTTP 时有效
- port: 检测的端口,如果配置为 0,则该字段无效
- portName: 检测端口名字(替换 marathon 协议中的 portIndex)
 - portName 在 port 配置大于 0 的情况下,该字段无效
 - portName 在 port 配置不大于 0 的情况下,检测的端口通过 portName 从 ports 配置中获取 (scheduler 处理)
 - 根据 portName 获取端口的时候,需要根据不同的网络模型获取不同的端口, 目前规则(和 exportservice 保持一致)如下:
 - BRIDGE 模式下如果 HostPort 大于零则为 HostPort,否则为 ContainerPort
 - 其他模式为 ContainerPort
- scheme: http 和 https(https 不会做认证的处理)
- path: 请求路径
- headers: http 消息头, 为了支持 health check 时, 需要认证的方式, 例如: Host: www.xxxx.com。NOTE:目前只支持 REMOTE_HTTP。
- 检测方式:
 - Sends a GET request to scheme://:port/path.
 - Note that host is not configurable and is resolved automatically, in most cases to 127.0.0.1.
 - Default executors treat return codes between 200 and 399 as success; custom executors may employ a different strategy, e.g. leveraging the `statuses` field.
 - bcs executor 需要根据网络模式等情况再具体确认规则
- tcp: type 为 TCP 和 REMOTE_TCP 的情况下有效:
- port: 检测的端口,如果配置为 0,则该字段无效

- portName: 检测端口名字(替换 marathon 协议中的 portIndex)
 - portName 在 port 配置大于 0 的情况下,该字段无效
 - portName 在 port 配置不大于 0 的情况下,检测的端口通过 portName 从 ports 配置中获取 (scheduler 处理)
 - 根据 portName 获取端口的时候,需要根据不同的网络模型获取不同的端口, 目前规则(和 exportService 保持一致)如下:
 - BRIDGE 模式下如果 HostPort 大于零则为 HostPort,否则为 ContainerPort
 - 其他模式为 ContainerPort
- 检测方式: tcp 连接成功即表示健康, 需根据不同网络模型获取不同的地址

ConfigMap 数据定义

配置模板说明

```
{
  "apiVersion": "v4",
  "kind": "configmap",
  "metadata": {
    "name": "template-configmap",
    "namespace": "defaultGroup",
    "labels": {
      "io.tencent.bcs.app.appid": "756",
      "io.tencent.bcs.cluster": "SET-SH-16111614092707",
      "io.tencent.bcs.app.moduleid": "5088",
      "io.tencent.bcs.app.setid": "1767"
    }
  },
  "datas": {
    "bcs.conf": {
      "type": "file",
      "content": "Y29uZmlnIGNvbnRleHQ="
    },
    "config-one": {
      "type": "http",
      "content": "http://adfasdasdasdfasadfa.txt"
    },
    "myftp": {
      "type": "ftp",
      "content": "ftp://path/to/a.txt",
      "RemoteUser": "myuser",
      "remotePasswd": "nIGNvbnRleHQ="
    }
  }
}
```

- RemoteUser //存储该文件第三方存储的用户名, 如果请求需要认证
- remotePasswd //密码

bcs application数据结构

```
"configmaps": [
    {
        "name": "template-configmap",
        "items": [
            {
                "type": "env",
                "dataKey": "config-one",
                "KeyOrPath": "SECRET_ENV"
            },
            {
                "type": "file",
                "dataKey": "config_two",
                "dataKeyAlias": "config-two",
                "KeyOrPath": "/data/contianer/path/myfile.conf",
                "readOnly": false,
                "user": "root"
            }
        ]
    }
]
```

字段含义: (field comment)

- name: 索引 configmap 名字 (index configmap name)
- type:
- env: 将 configmap 子项注入环境变量
- file: 将 configmap 注入文件
- dataKey: 索引 configmap 中子项 (subitem index in index configmap)
- KeyOrPath: 在容器中绝对路径 (absolute directory in container)
- readOnly: 权限 (permission, true or false)
- user: 文件用户, 默认 root, 如果用户在系统中找不到, 默认 root (file user, default value is root. default is root if user is not exist in OS)

BCS Deployment 说明

bcs-deployment 简介

bcs-deployment 是基于 bcs-application 抽象出的顶层概念、主要满足应用的滚动升级, 回滚, 暂停, 扩、缩容等需求。

注: 不支持deployment关联已经存在的application

配置模板说明

```
{  
    "apiVersion": "v4",  
    "kind": "deployment",  
    "metadata": {  
        "labels": {  
            "label_deployment": "label_deployment"  
        },  
        "name": "deployment-test-001",  
        "namespace": "defaultGroup"  
    },  
    "restartPolicy": {  
        "policy": "Always",  
        "interval": 5,  
        "backoff": 10  
    },  
    "killPolicy":{  
        "gracePeriod": 10  
    },  
    "constraint": {  
        "IntersectionItem": [  
            {  
                "UnionData": [  
                    {  
                        "name": "hostname",  
                        "operate": "CLUSTER",  
                        "type": 4,  
                        "set": {  
                            "item": [  
                                "mesos-slave-1",  
                                "mesos-slave-2"  
                            ]  
                        }  
                    }  
                ]  
            }  
        ]  
    },  
    "spec": {  
        "instance": 2,  
        "selector": {  
            "podname": "app-test-001"  
        },  
        "strategy": {  
            "type": "RollingUpdate",  
            "rollingupdate": {  
                "maxUnavailable": 1,  
                "maxSurge": 1,  
                "upgradeDuration": 60,  
                "rollingOrder": "CreateFirst",  
                "rollingManually": false  
            }  
        },  
        "template": {  
            "metadata": {  
                "labels": {  
                    "label_deployment": "label_deployment"  
                },  
                "name": "deployment-test-001",  
                "namespace": "defaultGroup"  
            },  
            "spec": {  
                "containers": [  

```

```
{
    "command": "python",
    "args": [
        "-m",
        "SimpleHTTPServer",
        "8888"
    ],
    "parameters": [],
    "type": "MESOS",
    "env": [
        {
            "name": "DNS_HOSTS",
            "value": "test_env"
        }
    ],
    "image": "hub_address.com/bcs/<IMAGE_NAME>",
    "imagePullUser": "xxx",
    "imagePullPasswd": "xxx",
    "imagePullPolicy": "Always",
    "privileged": false,
    "ports": [
        {
            "containerPort": 8899,
            "name": "test-port",
            "protocol": "HTTP"
        }
    ],
    "healthChecks": [
        {
            "protocol": "tcp",
            "path": "/http/path/only",
            "delaySeconds": 10,
            "gracePeriodSeconds": 12,
            "intervalSeconds": 10,
            "timeoutSeconds": 10,
            "consecutiveFailures": 10
        }
    ],
    "resources": {
        "limits": {
            "cpu": "0.5",
            "memory": "8"
        }
    },
    "volumes": [],
    "secrets": [],
    "configmaps": []
},
],
"networkMode": "BRIDGE",
"networktype": "cnm"
}
}
}
```

- 基础信息简介 由于 bcs-deployment 是基于 bcs-application 构建，其中的大部分信息与 bcs-application 一致，包含以下内容：

 - restartPolicy
 - killPolicy

- constraint
- spec.template 中所有字段信息

关于这部分字段与结构的详细信息请见。

下面介绍一下关于bcs-deployment本身特性的相关策略。

```

"spec": {
  "instances": 2,
  "selector": {
    "podname": "app-test-001"
  },
  "strategy": {
    "type": "RollingUpdate",
    "rollingupdate": {
      "maxUnavailable": 1,
      "maxSurge": 1,
      "upgradeDuration": 60,
      "rollingOrder": "CreateFirst",
      "rollingManually":false
    }
  }
}

```

- 实例数 相关参数为 spec.instances(第 2 行), 用于配置要创建的 taskgroup 的数量。该 taskgroup 是由 deployment 创建的一个 application 来管理。
- application 选择器 相关参数为 spec.selector (第 3 行) , 用于配置 deployment 所需要管理的 bcs-application,默认这些 bcs-application 是由 bcs-deployment 自动创建的。

deployment 升级策略

相关配置项为 spec.strategy (第 6-15 行) , 用于配置 deployment 执行 rolling 操作时所需要的策略: - type: 定义 deployment 进行 rolling 时要选择的策略, 目前只支持 RollingUpdate: -

RollingUpdate : RollingUpdate 即为滚动升级, 该策略允许我们对滚动操作的过程中每次新创建的容器数量, 删除的容器数量, 创建间隔等策略进行控制。当原有的 taskgroup 全部删除, 新的 taskgroup (个数通过 instances 参数定义) 全部创建, 则 update 结束。

- RollingUpdate 可以对 Rolling 的操作进行详细的配置, 包含以下参数:
- **maxUnavailable** : 决定了每个 rolling 周期内可以 **删除** 的 taskgroup 数量。如果原有的 taskgroup 已经全部删除, 则后续每一次 rolling 中不会再删除 taskgroup。
- **maxSurge** : 决定了每个 rolling 周期内可以 **创建** 的 taskgroup 数量。如果新的 taskgroup 已经全部创建, 则后续每一次 rolling 中不会再创建 taskgroup。
- **upgradeDuration** : 配置每次 rolling 操作之间的 **最小** 间隔时间。
- **rollingOrder** : 配置在进行每次 rolling 操作期间的每个周期内, 创建和删除应用的先后顺序。该配置支持两种模式 **CreateFirst** , **DeleteFirst** 。 **CreateFirst**策略会先创建新的应

用，然后删除老的应用。而**DeleteFirst**策略会先删除老的应用，再创建新的应用。

- **rollingManually**：配置每次滚动是否需要手动触发，默认为 false，即一次滚动完成之后在时间间隔结束之后自动进行下一次滚动，如果配置为 true，则在每次滚动后自动 pause，需输入 resume 命令才会在时间间隔结束之后进行下一次滚动

rolling update 的策略示例

我们假设 rolling 前 deployment 的 instances 为 oldInstances, rolling 的 deployment instances 为 newInstances。可以预见在 rolling 过程中会有以下三种场景：
- oldInstances < newInstances 对 application 进行一次 rolling update，并且达到扩容的效果
- oldInstances = newInstances 对 application 进行一次 rolling update，容器数量保持不变
- oldInstances > newInstances 对 application 进行一次 rolling update，并且达到缩容的效果

支持的 deployment 操作

- **create** 创建一个 deployment：如果已有绑定的 application，则 delete 当前 application 并创建新的 application；如果不存在绑定的 application，则创建 application。也可以创建一个空的 deployment 绑定到当前已有的 application。
- **update** 对 deployment 进行 rolling update：通过 instances 的变化，可以同时到达扩容和缩容的效果。
- **rollback** 停止当前的 update 操作，将新创建的 application 和 taskgroup 删除，将原有的 application 的 taskgroup 恢复到原有的 instances 个数
- **pause** 暂停 rolling update：rolling update 过程中可以通过该命令暂停 update。
- **resume** 继续 rolling update：可以将暂停的 update 继续。
- **delete** 删除 deployment，以及相应的 application

bcs Ingress 配置说明

bcs ingress 基于 bcs-service 抽象出将流量导入 bcs 集群的基本规则，满足应用多场景下的负载均衡需求。

配置模板说明

```
{  
    "apiVersion": "v1",  
    "kind": "ingress",  
    "metadata": {  
        "name": "ingress-demo",  
        "namespace": "default",  
        "labels": {  
            "app": "nginx"  
        }  
    },  
    "spec": {  
        "rules": [  
            {"host": "www.ingress-test.com", "http": {  
                "paths": [  
                    {"path": "/index.html", "backend": {  
                        "serviceName": "nginx",  
                        "port": 80  
                    }  
                ]  
            }  
        }]  
    }  
}
```

```
        "ingressname": "ingress-demo"
    },
    "annotation": {
        "annotation-demo": "annotation-value"
    }
},
"spec": {
    "lbGroup": "external",
    "clusterid": "sz-loldemo-0000123",
    "rules": [
        {
            "kind": "HTTP",
            "balance": "roundrobin",
            "httpIngress": {
                "host": "demo.bcs.com",
                "paths": [
                    {
                        "path": "/read/demo",
                        "backend": [
                            {
                                "serviceName": "service-1",
                                "servicePort": "8801",
                                "weight": 70
                            },
                            {
                                "serviceName": "service-2",
                                "servicePort": "8802",
                                "weight": 30
                            }
                        ]
                    }
                ]
            }
        },
        {
            "kind": "TCP",
            "balance": "roundrobin",
            "tcpIngress": {
                "listenPort": 8083,
                "backend": [
                    {
                        "serviceName": "service-3",
                        "servicePort": "8803",
                        "weight": 70
                    },
                    {
                        "serviceName": "service-4",
                        "servicePort": "8804",
                        "weight": 30
                    }
                ]
            }
        }
    ]
}
```

- 基本信息
- apiVersion: 版本
- kind: ingress

- metadata:
- name: ingress name
- namespace: ingress 所属的 namespace
- labels: ingress 所需要的 label 信息。
- annotation: ingress 所需要的相关标注信息。

Ingress Spec 信息

lbGroup

标识该 ingress 所属的 lb 管理节点。该名称需要和 lb 的名称相对应。否则无法正常工作。

clusterid

标识该 ingress 所属的 cluster。

rules

rules 明确的指出该 ingress 所需要负载均衡的一组相关信息，具体如下:
- kind: 支持 TCP 和 HTTP 两种类型, 使用 HTTP 则需要配置 httpIngress, 使用 TCP 则需要配置 tcpIngress。
- balance: 配置负载均衡的策略, 支持"roundrobin"与"source"。
- httpIngress: - host: 需要负载均衡的域名。
- paths: 该域名下具体的 uri 分配规则
- path:
指定具体的 path
- backend: 配置后端具体负载的实例, 以 service 为逻辑单位 ("serviceName") , 同时可以指定多个 service 之间的流量比例 ("weight")

- tcplngress:
- listenPort: lb 监听的服务端口。
- backend: 配置后端具体负载的实例, 以 service 为逻辑单位 ("serviceName") , 同时可以指定多个 service 之间的流量比例 ("weight")

注意 1. 目前 k8s 只支持 HTTP 的路由策略, TCP 的暂不支持。2. 对

于 httpIngress 中的 paths, 现预留为数组, 目前仅支持一个, 如果配置多个, 数组中的第 0 个元素生效, 其它的不会生效。

Secret 数据定义

数据结构说明

```
{
  "apiVersion": "v4",
  "kind": "secret",
  "metadata": {
    "name": "template-secret",
    "namespace": "defaultGroup",
    "labels": {
      "io.tencent.bcs.app.appid": "756",
      "io.tencent.bcs.cluster": "SET-SH-16111614092707",
      "io.tencent.bcs.app.moduleid": "5088",
      "io.tencent.bcs.app.setid": "1767"
    }
  },
  "type": "",
  "datas": {
    "first-secret": {
      "path": "/path/to/store/in/vault",
      "content": "Y29uZmlnIGNvbRleHQ="
    },
    "second-secret": {
      "path": "/path/to/store/in/vault",
      "content": "Y29uZmlnIGNvbRleHQ="
    }
  }
}
```

字段含义：

- name: 具体 secret 名字
- content: secret 内容

bcs application数据结构

```
"secrets": [
  {
    "secretName": "mySecret",
    "items": [
      {
        "type": "env",
        "dataKey": "abc",
        "keyOrPath": "SRECT_ENV"
      },
      {
        "type": "file",
        "dataKey": "abc",
        "keyOrPath": "/data/container/path/myfile.conf",
        "readOnly": false,
        "user": "user00"
      }
    ]
  }
]
```

工作机制和流程

数据流转流程

- bcs-client create --type secret --name template-secret --clusterid saldfkaslkdfj
- --path vault 存储 path, 必填
- --file key:/path/to/file.conf 指定文件, 文件大小不能超过 100k
- --from-files dir 指定文件夹, 根据文件名做 key, 总大小不能超过 2M
- --content key:content 直接指定具体内容
- bcs-apiserver (正常存入 vault?)
- DELETE: 根据规则删除 secret
- POST: 存储 vault 中
- GET: 查询 secret 信息
- bcs-route 流程
- 查询 secret 信息, 并入 taskgroup 请求, 转发给 scheduler

相关调整工作

- apiserver 增加 secret 接口
- post
- get
- delete
- bcs-route 数据解析
- 捕获 secret 字段, 提取 secret 具体内容
- 转发给 bcs-scheduler
- bcs-client 增加 secret 子命令, 参数如下
- --path vault 存储 path, 必填
- --file key:/path/to/file.conf 指定文件, 文件大小不能超过 100k
- --from-files dir 指定文件夹, 根据文件名做 key, 总大小不能超过 2M
- --content key:content 直接指定具体内容

Mesos Service 定义

Service 数据结构说明

service 主要用服务发现, DNS 基础数据, loadbalance 服务导出。

```
{
  "apiVersion": "v4",
  "kind": "service",
  "metadata": {
    "name": "template-service",
    "namespace": "defaultGroup",
    "labels": {
      "BCSGROUP": "external",
      "BCSBALANCE": "source|roundrobin|leastconn",
      "BCS-WEIGHT-lol-summoner": "3",
      "BCS-WEIGHT-new-lol-summoner": "7"
    }
  },
  "spec": {
    "selector": {
      "label-one": "lol-summoner",
      "label-two": "new-lol-summoner"
    },
    "type": "ClusterIP|NodePort|None|Integration",
    "clusterIP": ["1.1.1.1", "1.1.1.2"],
    "ports": [
      {
        "name": "http_8080",
        "domainName": "demo.bcs.com",
        "path": "/local/path",
        "protocol": "http",
        "servicePort": 80,
        "targetPort": 8080,
        "nodePort": 31000
      },
      {
        "name": "tcp-28800",
        "domainName": "demo.bcs.com",
        "path": "/local/path",
        "protocol": "tcp",
        "servicePort": 28800,
        "targetPort": 8080,
        "nodePort": 31001
      }
    ]
  }
}
```

endpoints

如果存在 endpoints， DNS 直接 watch 并直接关联 DNS 解析记录。如果没有，自行关联 service 和 taskgroup 信息，解析 status 信息提取 IP。

```
{
  "apiVersion": "v1",
  "kind": "endpoint",
  "metadata": {
    "name": "template-endpoint",
    "namespace": "defaultGroup",
    "label": {
      "io.tencent.bcs.app.appid": "756",
      "io.tencent.bcs.cluster": "SET-SH-16111614092707",
      "io.tencent.bcs.app.moduleid": "5088",
      "io.tencent.bcs.app.setid": "1767"
    }
  }
}
```

```
        },
        "eps": [
            {
                "nodeIP": "10.1.1.1",
                "containerIP": "192.168.1.2"
            },
            {
                "nodeIP": "10.1.1.2",
                "containerIP": "192.168.1.1"
            }
        ]
    }
```

特别字段说明

clusterIP: 考虑未来需要做外部域名访问和服务发现，clusterIP 暂留用于指向 proxyIP 信息

ports[x]说明：

- name: 该 name 需要和 taskgroup 中 ports 字段中的 name 一致
- domainName: http 协议的状态下，需要填写域名信息，做转发用途
- servicePort: 主要用于负载均衡和服务导出端口
- targetPort: 用于指向 taskgroup 中 ports 字段中目标端口，默认目标端口为 containerPort，如果有 hostPort，则指向 hostPort

BCSGROUP: 用于 service 导出标识

BCSBALANCE: 用于服务导出负载均衡算法，默认值为 roundrobin

BCS-WEIGHT-: 当使用 selector 匹配多个 application 时，用于表达多个 application 之间的权重，该值为大于等于 0 的整数，类型为 string。如果等于 0，则该 application 没有流量导入。

bcs-loadbalance 功能使用

如果要启动 loadbalance 的功能，需要：

- label 中增加特殊字段 "**BCSGROUP":"external**" : external 代表 bcs-loadbalance 模块的集群 ID，默认值 external
- label 中增加特殊字段 "**BCSBALANCE":"roundrobin**" : 负载均衡算法，默认值为 roundrobin，其他值为 source (ip_hash) , leastconn
- application 有定义 ports 信息，并和 service 对应

ports 信息在 loadbalance 中的含义说明：

- protocol: http 或者 tcp，当前不支持 udp

- name: 名字为 application 中定义 port 的名字，必须要对应
- domainName: 协议为 http 时有效
- servicePort: 如果协议是 http, 该值被忽略, loadbalance 默认使用 80 端口; 如果是 tcp, loadbalance 则监听该指定端口, 各 service 之间该端口不能冲突

bcs-loadbalance 可以工作两种环境下:

- overlay 方式, 默认使用 servicePort 作为服务端口, 流量转发至 containerPort
- underlay 方式, 使用 servicePort 作为服务端口, 流量转发至 hostPort (限于 host/bridge 模式)

taskgroup 服务端口机制

服务端口数据来源于镜像字段 ports, 主要包含:

- containerPort: 容器中使用端口
- servicePort: 多实例状态使用的服务端口, 用于服务导出和负载均衡使用
- hostPort: 从容器映射到主机的端口
- hostIP: 预留字段, 当物理主机包含多个网卡时, 默认是所有网卡都映射。如果 0.0.0.0 监听存在风险, 可以监听内网
- protocol: 协议, tcp/udp, http (默认转换为 tcp 处理, 仅对 datawatch/loadbalance 生效)
- name: 域名信息, http 时必须为域名, 必须唯一
- path: 路由转发, 用于转发到后端对应的 endpoint

填写端口信息, 至少需要填写 name、protocol、containerPort、servicePort 信息。如果不实现端口映射, hostPort 无需填写。

docker 存在网络模式: None, Host, Bridge, User (CNM/CNI), 端口映射对 None (当前 Executor 暂未校验)

hostPort 使用定义: -1 为不使用, 0 为随机端口, 正数为指定端口

针对 docker 容器端口映射实现, 端口绑定有两个情况:

- 指定端口绑定

下发参数中没有 parameter 参数-P, 并直接指定 hostPort, executor 默认直接将该指定参数提交给 docker 进行绑定。端口冲突是否需要用户执行决断。如果没有特别指定调度策略将实例分开, 有极大几率造成端口冲突。

- Host 模式下 hostPort 保持和 containerPort 一致，方便 datawatch 处理
- 随机端口绑定

下发参数中存在-P，并且有 ports 字段，hostPort 字段默认为 0，scheduler 使用 offer 上报的端口信息确认端口，更新 taskgroup 字段，并下发给 Executor。随机端口绑定可以消除端口冲突的问题。有随机端口的情况下，为了方便应用获取到端口信息，需要将 port 端口根据需要导入环境变量 PORT0 ~ PORTn

更新（2017-05-22，根据 LOL 需求更新）：

- Host 模式下：
- containerPort 设置为 0， 默认使用 offer 提供端口资源，hostPort 和 containerPort 保持一致。
port 端口根据序号导入环境变量 PORT0 ~ PORTn
- containerPort 指定端口， 默认传递该端口，port 端口根据序号导入环境变量 PORT0 ~ PORTn
- Bridge 模式
- hostPort 不填写默认补齐为-1，仅有 containerPort
- hostPort 为 0， 默认随机端口，使用 offer 端口资源，port 端口根据序号导入环境变量 PORT0 ~ PORTn

服务端口代码调整说明

- executor

executor 上报容器状态时，增加 ports 字段，结构包括 name, hostPort, containerPort, protocol

- api/scheduler

使用随机端口时， 默认 hostPort 直接填写 0， 使用 offer 里面端口资源更新 hostPort 信息，不使用 HostPort，建议默认值为负数

- datawatch-mesos/datawatch-kube/loadbalance 支持

上报 ExportService 时，结构调整，当前为

```
//ExportService info to hold export service
type ExportService struct {
    Cluster      string      `json:"cluster"`      //cluster info
    Namespace    string      `json:"namespace"`    //namespace info, for business
    ServiceName  string      `json:"serviceName"` //service name
    ServicePort  []ExportPort `json:"ports"`       //export ports info
    BCSGroup     []string    `json:"BCSGroup"`    //service export group
    SSLCert      bool        `json:"sslcert"`     //SSL certificate for ser
    Balance      string      `json:"balance"`     //loadbalance algorithm, default sour
ce
    MaxConn     int         `json:"maxconn"`    //max connection setting
}
```

```
}
```

字段信息：

- SSLCert: 是否使用 https, 当前默认不开启, 忽略
- Balance: 负载均衡的算法, 默认是 source, 忽略
- MaxConn: 最大连接数, 默认是 20000

调整 ServicePort 和 Backends 字段, 对其进行合并

```
type ExportPort struct {
    BCSVHost    string      `json:"BCSVHost"`
    Protocol    string      `json:"protocol"`
    Path        string      `json:"path"`
    ServicePort int         `json:"servicePort"`
    Backends    []Backend   `json:"backends"`
}

type Backend struct {
    TargetIP string `json:"targetIP"`
    TargetPort int   `json:"targetPort"`
}
```

datawatch 构建 ExportService 时需要按照新结构进行。构建 ExportService 信息需要结合 Service 和 TaskGroup 信息。

- service 中, 多个 ports 对应多个 ExportPort
- Service.ports[i].name == TaskGroup.container[i].ports[i].name
- 提取 TaskGroup 多个实例的 containerIP, NodeIP, ContainerPort, HostPort
- backend 组装说明, 多个实例有多个 backend
- 如果 TaskGroup 网络模式 host, 取 NodeIP 和 ContainerPort, 组成 backend
- 网络模式为 bridge, 如果有 HostPort: NodeIP + HostPort; 如果没有, 则 ContainerIP + ContainerPort
- 其他模式: ContainerIP + ContainerPort

应用列表说明文档

蓝鲸容器服务应用列表分两个维度展示：模板集维度和命名空间维度；其中，模板集维度主要展示由【模板集】实例化的应用；命名空间维度展示了集群下所有所有命名空间下的所有应用，来源包含模板集、Client 实例化的应用；以便用户查看和操作应用及查看应用下容器运行详情及资源使用详情。

- 应用类型 application/deployment

查询展示

● 模板及应用展示

The screenshot shows a dashboard for managing application templates and deployments. At the top, there are dropdown menus for '集群模板视图' (Cluster Template View) and '集群类型' (Cluster Type), with '测试集群' (Test Cluster) selected. A search bar and a refresh icon are also present.

Below the header, there are two sections:

- 小游戏部署模板**: Shows a single deployment named 'deploy-redis1' (version v9, status Running, deployment type, namespace yanzhen). It includes buttons for '批量删除' (Batch Delete), '实例化' (Instantiate), and '操作' (Operations).
- game-nemo部署模板**: Shows a single deployment named 'deploy-redis1' (status Running, deployment type, namespace yanzhen). It includes buttons for '滚动升级' (Rolling Upgrade), '扩缩容' (Scale), and '更多' (More).

Both sections indicate they contain 1 application template.

● 应用详情

The screenshot displays detailed information for the deployment 'deploy-redis1'. At the top, there's a back arrow, a 'to yaml' button, and a chart icon.

应用名称:	deploy-redis1	创建时间:	2018-11-05 11:55:14	更新时间:	2018-11-05 11:55:14	所在命名空间:	yanzhen	所属集群:	beilike-test
-------	---------------	-------	---------------------	-------	---------------------	---------	---------	-------	--------------

Below the table are two line charts showing CPU and memory usage over time (15:30 to 16:10). The CPU usage chart shows a single data point at 0.00% for the container 'container-redis-default' on 2018-11-09 15:56:22. The memory usage chart shows a flat line at 0.0%.

Pod管理	事件	标签	备注	Metric信息
deploy-redis1-6656bbdb895-jms...	Running			状态: Running Host IP: 10.223.49.220 Pod IP: 10.233.70.45 存活时间: 4天4小时18分6秒 重新调度

● 容器详情



操作相关

- 滚动升级

注：允许应用类型包含 deployment/daemonset/job/statefulset

应用列表

集群模板视图
集群类型
测试集群
过滤

小游戏部署模板

共包含 1 个应用模板

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
deploy-redis1	v9	Deployment	正常		1/1	11-05 11:55	批量删除 实例化 滚动升级 扩缩容 更多

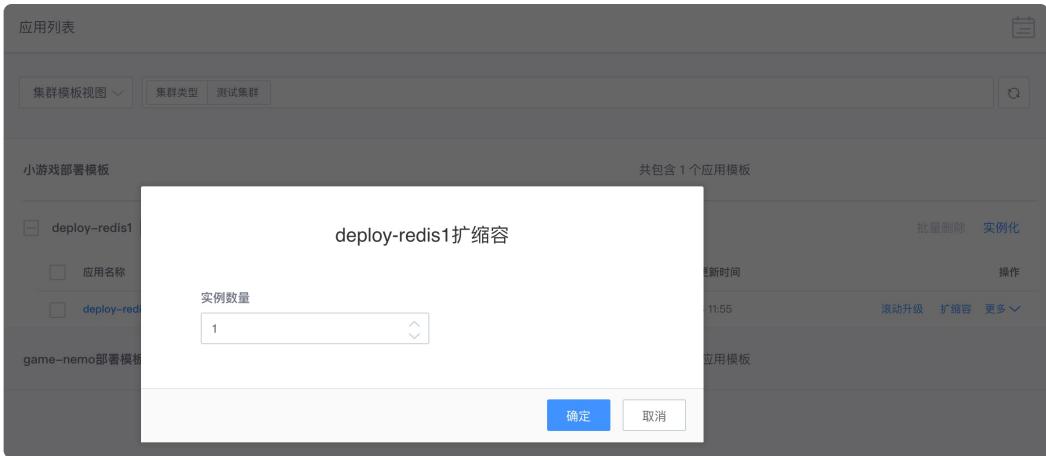
game-nemo部署模板

共包含 1 个应用模板

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
							滚动升级 扩缩容 更多

- 扩缩容

实例数量增加或减少



- 重建应用

重建包含两个步骤: 删除当前应用, 然后以现有配置再创建应用(类似进程重启)



- 删除应用



- 重新调度



BCS 容器网络方案集成

面对复杂业务应用需求，BCS 针对业务的 网络支持是按照插件化 方式进行设计，方便业务在不同环境下能对接不同的网络实现，完成具体的定制。

在网络标准选择上，BCS 建议采用 CNI 网络，以方便容器网络能获得最佳的扩展能力。如果启用 BCS 相关的 **SaaS 服务**，集群的相关安装与配置可以通过 SaaS 来完成，如果只是启用 BCS 后台服务，相关配置请参照以下指引。

- **BCS Kubernetes 网络实践**
- **BCS Mesos 网络实践**

Kubernetes 网络方案

如果针对容器网络，没有额外设计需求，BCS 建议采用 Kubernetes 社区建议的网络方案。

Kubernetes 与 BCS 集成可以参考 **BCS 高可用 Kubernetes 集群部署**

CNI 扩展

Kubernetes node 节点默认安装 CNI 工具的目录

- 二进制目录: /opt/cni/bin/
- 配置目录: /etc/cni/net.d/

如果已经针对网络进行 CNI 扩展，则需要进行扩展 CNI 工具进行部署。

手动部署

- CNI 部署，将编译好的 CNI 工具部署到所有 Node 节点 /opt/cni/bin
- 配置部署，编写对应的 CNI 配置，放置到 /etc/cni/net.d/，注意配置文件字母序要为第一。

镜像部署

社区采用方案，将扩展好的 CNI 工具与 CNI 社区提供的所有工具整合成 CNI 工具镜像， 默认在每个节点启动 Daemonset，实现 CNI 工具部署。

细节可以参考 [Flannel-CNI 项目](#)

kubelet 在 /etc/cni/net.d 目录下读取 CNI 的配置文件时，是按文件名的字母顺序来的。在这个目录下可以放多个配置文件，kubelet 会读取字母顺序排第一的配置文件。

多网络集成

针对部分复杂的场景，容器可能需要多网络设置与集成，由于 Kubernetes 默认不支持 CNI 链式调用，BCS 建议在以下场景，可以采用链式 CNI 插件实现多 CNI 插件集成。

- 多网络栈配置
- 网络带宽限制
- 额外路由配置
- IP 端口映射
- 网络内核参数调整

社区当前有多重链式 CNI 插件可以选择：

- [multus-CNI](#)
- [CNI-genie](#)

插件安装方式请参照上述链接。

SaaS 插件使用

完成 multus-CNI 安装之后，multus-CNI 在 bcs-saas 相关页面设置：

“

模板集--Deployment 设置--更多设置--备注，完成 annotation 的 KV 填写

如下图：

”

The screenshot shows the BCS Container Service interface. On the left, there's a sidebar with various options like Cluster, Node, Namespace, Template Set, Variable Management, Helm, Application, Network, Resource, Repository, Audit, Event Query, and Monitoring Center. The main area is titled 'cni-test' and 'Template Set Overview'. It shows a deployment named 'cni-test' with two instances. The 'Annotations' section is highlighted with a red box, showing the key 'v1.multus-cni.io/default-network' with the value 'default/macvlan'. There are tabs for 'Annotations', 'Restart Policy', 'Kill Policy', 'Scheduling Constraints', 'Network', 'Volume', and 'Log Collection'.

LoadBalancer

Ingress 一节中提到 Ingress 描述资源对外的访问方式，背后需要 Ingress Controller 支撑，本节介绍 BCS 中使用的 Ingress Controller。

Ingress Controller 有很多种实现，比如 Traefik、Nginx Ingress Controller、HAProxy Ingress Controller 等，在 BCS 中定义 LoadBalancer，采用 K8S 官方维护的 Nginx Ingress Controller。

如何创建 LoadBalancer

在【LoadBalancer】菜单中，点击【新建 LoadBalancer】，添加一台或多台节点作为 LoadBalancer，点击【保存】即可。

The screenshot shows the BCS Container Service interface. On the left, there's a sidebar with various options like Cluster, Node, Namespace, Template Set, Variable Management, Helm, Application, Network, Service, Ingress, LoadBalance, Resource, Repository, Audit, Event Query, and Monitoring Center. The main area is titled 'LoadBalance' and 'Edit LoadBalance'. It shows a LoadBalance named 'LoadBalance' with the following configuration:所属集群: weixin(BCS-K8S-40015), 命名空间: dev, 端口: http 80, https 443. The 'Annotations' section is highlighted with a red box, showing the key 'v1.multus-cni.io/default-network' with the value 'default/macvlan'. There are tabs for 'Annotations', 'Port Mapping', 'Protocol', 'Node Selection', and 'Save'.

LoadBalancer 的背后技术

Nginx Ingress Controller

实际是创建了 2 个 Deployment

```
# kubectl get deploy -n dev
NAME                      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   A
GE
blueking-ingress-controller   1         1         1           1          3
2h
blueking-ingress-default-backend 1         1         1           1          3
2h
```

查看 nginx-ingress-controller 的配置文件，可以看到我们在场景案例 [应用的蓝绿发布](#) 中定义的 Ingress。

```
# kubectl exec blueking-ingress-controller-79c687dc95-c4k5b -n dev -- cat /etc/nginx/nginx.conf

## start server blue.bk.tencent.com
server {
    server_name blue.bk.tencent.com ;

    listen 80;

    listen [::]:80;

    set $proxy_upstream_name "-";

    location / {

        port_in_redirect off;

        set $proxy_upstream_name "dev-web-nginx-blue-80";

        set $namespace      "dev";
        set $ingress_name   "web-nginx-blue";
        set $service_name   "web-nginx-blue";

        client_max_body_size          "500m";

        proxy_set_header Host          $best_http_host;

        # Pass the extracted client certificate to the backend

        proxy_set_header ssl-client-cert      "";
        proxy_set_header ssl-client-verify     "";
        proxy_set_header ssl-client-dn        "";

        # Allow websocket connections
        proxy_set_header Upgrade          $http_upgrade;
        proxy_set_header Connection        $connection_upgrade;

        proxy_set_header X-Real-IP        $the_real_ip;

        proxy_set_header X-Forwarded-For   $the_real_ip;
```

```

proxy_set_header X-Forwarded-Host $best_http_host;
proxy_set_header X-Forwarded-Port $pass_port;
proxy_set_header X-Forwarded-Proto $pass_access_scheme;
proxy_set_header X-Original-URI $request_uri;
proxy_set_header X-Scheme $pass_access_scheme;

# Pass the original X-Forwarded-For
proxy_set_header X-Original-Forwarded-For $http_x_forwarded_for;

# mitigate HTTPoxy Vulnerability
# https://www.nginx.com/blog/mitigating-the-httpoxy-vulnerability-with-nginx
x/
proxy_set_header Proxy "";

# Custom headers to proxied server

proxy_connect_timeout 5s;
proxy_send_timeout 60s;
proxy_read_timeout 60s;

proxy_buffering "off";
proxy_buffer_size "4k";
proxy_buffers 4 "4k";
proxy_request_buffering "on";

proxy_http_version 1.1;

proxy_cookie_domain off;
proxy_cookie_path off;

# In case of errors try the next upstream server before returning an error
proxy_next_upstream error timeout invalid_header http_502 http_503 http_504;

proxy_pass http://dev-web-nginx-blue-80;

proxy_redirect off;

}

}

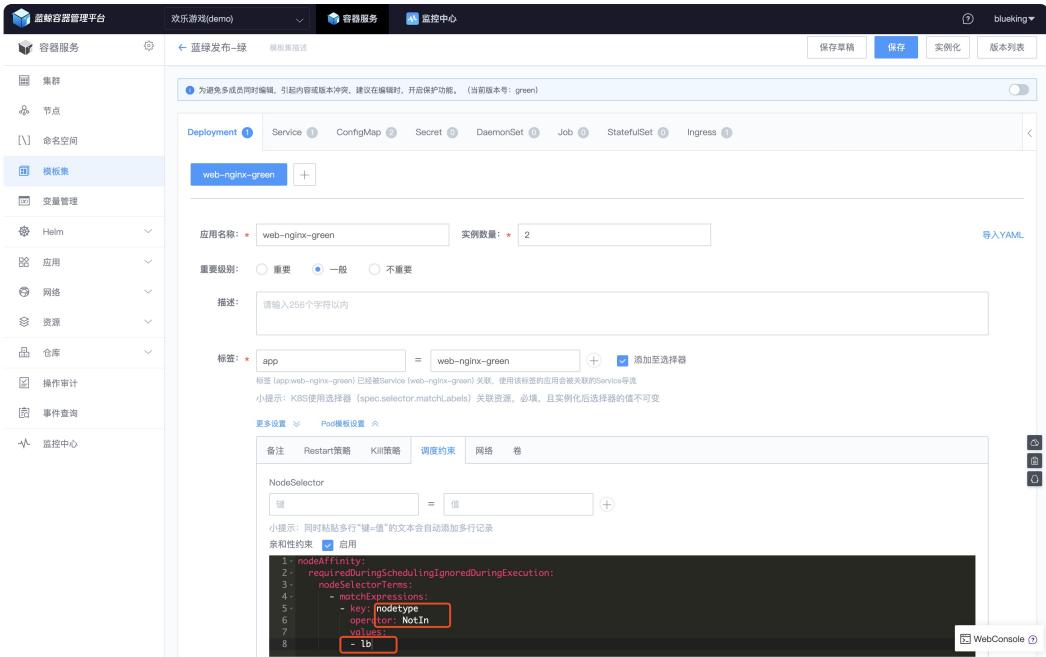
## end server blue.bk.tencent.com

```

使用建议

一般我们建议选择多台专属节点作为 LoadBalancer，不要分配其他 Pod 到该节点。

可以通过 **节点亲和性约束** 实现。



BCS 自研 Mesos 容器方案网络集成

BCS 在设计 Mesos 方案之初已经考虑多种网络的集成问题。在容器 json 文件中有相关参数进行网络参数详细设定。原始 json 文档可以参考

- Application

关键参数说明

- **networkType:** 容器网络标准，可选 cni 与 cnm
- **cnm:** 默认值，与 docker 网络集成时使用
- **cni:** CNI 标准支持，支持基于 CNI 标准的所有插件
- **networkMode:** 容器网络模式
- **HOST:** host 网络模式，需要和 cnm 组合，容器共享主机网络栈
- **BRIDGE:** 网桥模式，cnm/cni 标准皆可以支持
- **自定义方式:** docker 支持的其他网络模式、CNI 支持的插件皆可

与 docker 集成

与 docker 网络集成时，application 的 json 配置 networkType 需要填写 cnm，默认即为该模式。

networkMode 的值则为 docker 已生效的网络模式，docker 网络模式查看：

```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
437f2816be0a    bridge    bridge      local
1093cd0625ed    host      host       local
a1a0f9fb969     none      null       local
```

其他自定义 docker 网络模式需要自行手动创建或对支持 CNM 模式的网络方案进行集成，例如 macvlan：

```
#创建macvlan，网络名字为test-macvlan
$ docker network create -d macvlan \
--subnet=172.16.86.0/24 \
--gateway=172.16.86.1 \
-o parent=eth1 \
test-macvlan
```

当使用 docker host、bridge 时，BCS 方案会针对 json 中填写的端口信息进行映射。例如 application 定义：

```
"ports": [
    "containerPort": 8090,
    "hostPort": 8090,
    "name": "test-tcp",
    "protocol": "TCP"
]
```

映射方式说明：
* host 模式下，containerPort 即代表 hostPort * 填写固定端口，需要业务自行确认是否产生冲突 * 填写 0，意味着 BCS 针对端口进行随机选择，端口实际值会写入环境变量中
bridge 模式下，hostPort 代表物理主机上的端口 * hostPort 填写固定端口，业务自行解决冲突的问题 * 填写 0，BCS 默认进行端口随机 * 小于 0，不进行端口映射

CNI 集成

BCS Mesos 方案默认支持 CNI 插件，使用 CNI 方案需要配置：

- CNI 目录，安装 BCS scheduler 时设置，默认为 /data/bcs/bcs-cni
- 二进制插件目录： /data/bcs/bcs-cni/bin
- 配置目录： /data/bcs/bcs-cni/conf
- 部署 CNI 插件与配置到对应目录
- application 或 deployment json 中指定使用的网络插件

例如采用 macvlan

```
"networkType": "cni",
```

```
"networkMode": "mymacvlan",
```

macvlan 插件配置:

```
{
  "cniVersion": "0.3.0",
  "name": "mymacvlan",
  "type": "macvlan",
  "master": "eth1",
  "ipam": {
    "type": "bcs-ipam",
    "routes": [
      {"dst": "0.0.0.0/0"}
    ]
  }
}
```

多网络集成

BCS Mesos 方案下的容器实现默认支持 CNI 链式调用方案。如要启动 CNI 链式调用，需要:

- CNI conf 目录下需要放置有链式调用配置
- CNI bin 目录下放置有所依赖的二进制文件

例如 CNI 配置:

```
{
  "cniVersion": "0.3.1",
  "name": "dbnet",
  "plugins": [
    {
      "type": "bridge",
      "bridge": "cni0",
      "ipam": {
        "type": "host-local",
        "subnet": "10.1.0.0/16",
        "gateway": "10.1.0.1"
      }
    },
    {
      "type": "tuning",
      "sysctl": {
        "net.core.somaxconn": "500"
      }
    }
  ]
}
```

- 确保 CNI bin 目录下部署有 bridge 与 tuning 二进制文件
- 运行时依次调用 bridge 与 tuning 插件

调用该 CNI 配置时，Application 或 Deployment 需要明确填写:

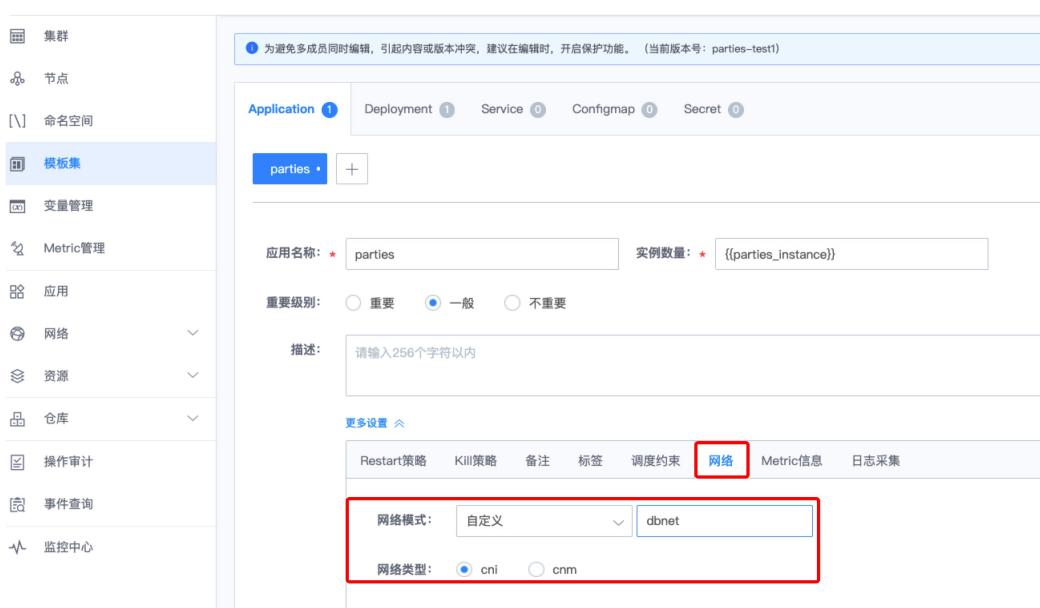
```
"networkType": "cni",
"networkMode": "dbnet",
```

SaaS 页面插件使用

完成插件配置后之后，在 bcs-saas 相关页面设置：

模板集--Application 设置--更多设置--网络，完成相关参数填写

如下图：



K8S 存储

BCS 支持 K8S 原生的多种存储能力。在存储类型上，支持本地存储 hostpath、emptyDir、local 等；在存储方式上，支持 Volume、静态 PV、动态 PV；在存储插件方案上，支持 K8S 内置的 In-tree 存储驱动、FlexVolume 存储驱动，以及 CSI 存储驱动的方案。

基础概念

- Volume、PV 和动态 PV K8S 使用 Volume、PV 来管理 Pod 容器的持久化数据。
- Volume Volume 生命周期与 Pod 绑定，容器挂掉重启后，Volume 的数据依然存在。Pod 被删除时，Volume 被清理，数据是否丢失取决于 Volume Driver 类型。
- PV 为了更好管理应用的持久化数据存储，K8S 推出了 PV (**Persistent Volume**) 的概念。PV

独立于 Pod 的生命周期。应用在使用 PV 时，先创建 PVC ([PersistentVolumeClaim](#))，然后在 Pod 中声明绑定 PVC。

PV 有 Static PV 和 [Dynamic PV](#) 两种使用方式。

- PV、PVC 的关系：
- PV、PVC 类似 Nodes、Pods 的关系，Pod 是最小调度单元，资源是 Node 提供。
- 工作负载 (Workload) 调度 PVC 申请网络资源，PVC 按照规范匹配合适的 PV，或者动态创建，然后 PV 和 PVC 进行绑定

本地存储

hostpath

hostpath 映射宿主机 host 上的目录或文件至 K8S Pod 容器中，典型场景如运行的容器需要访问 Docker 内部结构，此时挂载宿主机 host 上的 /var/lib/docker 目录至容器中，又如将 cAdvisor 部署在 Pod 容器中，需要挂载宿主机 host 上的 /sys 目录至容器中。

使用示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: nginx:latest
      name: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: /data
          name: testvolume
  volumes:
    - name: testvolume
      hostPath:
        path: /tmp/testvolume
```

emptyDir

emptyDir 用于 Pod 需要挂载临时目录的情况。当 Pod 被调度到 host 节点上时，在 host 上自动创建一个空目录并挂载进 Pod 容器中，当 Pod 从 host 上删除时，这个目录也被删除。因此目录中存储的数据在 Pod 删除后并不会持久化到宿主机 host 上。

使用示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
```

```

containers:
- image: nginx:latest
  name: nginx
  ports:
  - containerPort: 80
  volumeMounts:
  - mountPath: /cache
    name: cache-volume
volumes:
- name: cache-volume
  emptyDir: {}

```

本地存储使用方式

使用 hostpath 和 emptyDir 等本地存储时，建议用 volume 方式来做存储管理。在 "模板集" » Deployment » Pod 模板设置 » 卷" 中，可以配置使用 hostpath 或 emptyDir 本地存储。

The screenshot shows the Baidu Cloud Kubernetes console interface. At the top, there's a navigation bar with a back arrow and the text '模板集_1562209195469'. Below the navigation is a tabs bar with 'Deployment' (selected), Service, Configmap, Secret, DaemonSet, Job, StatefulSet, and Ingress. The main area shows a deployment named 'deployment-1'. Underneath the deployment name are fields for '应用名称:' (application name) set to 'deployment-1', '实例数量:' (instance count) set to '1', and '重要级别:' (importance level) with '一般' (General) selected. There's also a '描述:' (Description) text area and a '标签:' (Labels) section. At the bottom, there's a '更多设置' (More Settings) dropdown and a '卷' (Volume) tab which is currently active. A red box highlights the '挂载名' (Mount Name) dropdown set to 'hostPath'.

存储插件方案

K8S 支持多种后端存储，包括公有云存储产品 AWS EBS, AzureFile, GCE PD 等；商业存储产品 ScaleIO 等；开源存储产品 Ceph, GlusterFS 等。后端存储是以插件化的方式提供支持的，K8S 前期都是以内置 in-tree 的方式接入第三方存储，存储驱动的代码是嵌入 K8S 核心代码当中的。从 1.13 版本开始，CSI(Container Storage Interface)插件方案正式进入到稳定版本，使用 CSI 插件方案可以解耦 K8S 与存储 driver，以容器化的方式把 CSI 存储驱动部署在 K8S 集群中。

BCS 原生支持 K8S 的存储方案，考虑到安全性、可扩展性以及可运维性，BCS 推荐以 CSI 的方式使用第三方后端存储。下面以 Ceph RBD 为例，展示 BCS K8S 第三方分布式存储的接入。

Ceph RBD 接入和使用

RBD CSI 部署

RBD CSI 推荐以 Helm 的方式部署到 K8S 集群当中。BCS 会在 Helm 仓库中提供 RBD CSI 的 Helm Chart 包，用户如果需要在 K8S 集群的应用中使用 Ceph RBD 分布式块存储，可直接通过 Helm 把 RBD CSI 部署至集群当中。

RBD CSI 使用

创建 STORAGECLASS

BCS 管理员后台创建 RBD 的 StorageClass

- secret:

```
# This is a template secret that helps define a Ceph cluster configuration
# as required by the CSI driver. This is used when a StorageClass has the
# "clusterID" defined as one of the parameters, to provide the CSI instance
# Ceph cluster configuration information.
apiVersion: v1
kind: Secret
metadata:
  # The <cluster-id> is used by the CSI plugin to uniquely identify and use a
  # Ceph cluster, the value MUST match the value provided as `clusterID` in the
  # StorageClass
  name: ceph-cluster-{ceph_cluster_id}
  namespace: default
data:
  # Base64 encoded and comma separated Ceph cluster monitor list
  #   - Typically output of: `echo -n "mon1:port,mon2:port,..." | base64`
  monitors:
    # Base64 encoded and comma separated list of pool names from which volumes
    # can be provisioned
  pools:
    # Base64 encoded admin ID to use for provisioning
    #   - Typically output of: `echo -n "<admin-id>" | base64`
    # Substitute the entire string including angle braces, with the base64 value
    adminid: =
    # Base64 encoded key of the provisioner admin ID
    #   - Output of: `ceph auth get-key client.<admin-id> | base64`
    # Substitute the entire string including angle braces, with the base64 value
    adminkey:
      # Base64 encoded user ID to use for publishing
      #   - Typically output of: `echo -n "<admin-id>" | base64`
      # Substitute the entire string including angle braces, with the base64 value
      userid:
        # Base64 encoded key of the publisher user ID
        #   - Output of: `ceph auth get-key client.<admin-id> | base64`
        # Substitute the entire string including angle braces, with the base64 value
        userkey:
```

- StorageClass:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
    name: csi-rbd
    provisioner: rbd.csi.ceph.com
  parameters:
    clusterID: {ceph_cluster_id}
    pool:
      # RBD image format. Defaults to "2".
      imageFormat: "2"

      # RBD image features. Available for imageFormat: "2"
      # CSI RBD currently supports only `layering` feature.
      imageFeatures: layering
    reclaimPolicy: Delete
```

创建 PVC

用户在模板集中创建 PVC, 或者使用 kubectl 通过 bcs-api 操作 K8S 集群创建 PVC

- PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rbd-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 0.1Gi
  storageClassName: csi-rbd
```

引用 PVC

在模板集中创建的 Deployment 应用时, 在 Pod 中指定挂载 PVC

- Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: csirbd-demo-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: csirbd-demo-Pod
  template:
    metadata:
      labels:
        app: csirbd-demo-Pod
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
      volumeMounts:
```

```
- name: mypvc
  mountPath: /data1
volumes:
- name: mypvc
  persistentVolumeClaim:
    claimName: rbd-pvc
    readOnly: true
```

将 NFS 作为 K8S PV Provisioner

情景

互联网应用常见的三层架构：接入层、逻辑层、存储层，在操作系统中 **文件系统** 提供存储层的存储介质，在 K8S 中是 **Persistent Volumes**（持久卷，简称 PV），而 PV 背后需要对接存储介质，比如 NFS、CephFS 以及公有云的云硬盘[1]。

接下来以 NFS 作为 K8S PV 的存储介质（Provisioner）为例，介绍在 K8S 中如何申请以及使用存储空间。

前提条件

- 了解 K8S 中 **存储** 的基础的概念。
- 了解 **K8S 的包管理工具 Helm**

操作步骤

1. 部署 NFS Server
2. 部署 NFS-Client-Provisioner
3. 创建 PVC 测试

部署 NFS Server

以下为测试环境在 CentOS 7 下搭建 NFS 的示例，生产环境请咨询公司系统管理员。

安装 NFS Server 端（IP：10.0.5.85），并启动以及设置开机自启动。

```
# yum -y install nfs-utils
systemctl enable rpcbind
systemctl enable nfs
systemctl start rpcbind
```

```
systemctl start nfs
```

设置 `/nfs` 目录为挂载目录，对 `10.0.0.0/16` 网段开放。

```
# mkdir /nfs

# vim /etc/exports
/nfs    10.0.0.0/16(rw,sync,no_root_squash,no_all_squash)

# systemctl restart nfs

# showmount -e localhost
Export list for localhost:
/nfs 10.0.0.0/16
```

本地挂载测试，验证 NFS 部署是否成功。

```
# mount -t nfs <NFS_SERVER_IP>:/nfs /mnt

# nfsstat -m
/mnt from <NFS_SERVER_IP>:/nfs
Flags: rw,relatime,vers=4.1,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,time
o=600,retrans=2,sec=sys,clientaddr=<NFS_SERVER_IP>,local_lock=none,addr=<NFS_SERVER_IP>
```

部署 NFS-Client-Provisioner

K8S 使用 NFS 资源，需要能挂载 NFS 以及配套的 K8S 资源（StorageClass、ServerAccout、PersistentVolume、PersistentVolumeClaim 等）。

为了简化部署，以及为了使用 K8S 中的包管理器 Helm（类比 yum）来部署 NFS-Client-Provisioner 的 Chart（类比 rpm）。

将 Chart 推到仓库

“

由于 Helm V2 需要在集群中部署 tiller，存在安全风险，无法直接使用 Helm install 部署应用，BCS 会将 Chart 通过 Helm template 解析为 K8S 的资源配置来部署。

”

下载的 Charts。

```
git clone https://github.com/helm/charts/
```

```
# pwd
charts/stable/nfs-client-provisioner
# ll
总用量 28
-rw-r--r-- 1 root root 479 8月 27 17:00 Chart.yaml
drwxr-xr-x 2 root root 4096 8月 27 17:00 ci
```

```
-rw-r--r-- 1 root root 74 8月 27 17:00 OWNERS
-rw-r--r-- 1 root root 5193 8月 27 17:00 README.md
drwxr-xr-x 2 root root 4096 8月 27 17:00 templates
-rw-r--r-- 1 root root 1677 8月 27 17:00 values.yaml
```

```
# helm push . joyfulgame
Pushing nfs-client-provisioner-1.2.6.tgz to joyfulgame...
Done.
```

选择菜单【Chart仓库】，点击【同步仓库】可以看到刚刚推送上的Chart。

图标	Helm Chart名称	版本	描述	最近更新	操作
	nfs-client-provisioner	1.2.6	nfs-client is an automatic provisioner that used your *already configured* NFS server, automatically creating Persistent Volumes.	2019-09-09 18:00:35	<button>部署</button>
	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>

部署 Chart

点击【部署】，准备部署 Chart

图标	Helm Chart名称	版本	描述	最近更新	操作
	nfs-client-provisioner	1.2.6	nfs-client is an automatic provisioner that used your *already configured* NFS server, automatically creating Persistent Volumes.	2019-09-09 18:00:35	<button>部署</button>
	gitlab-ce	0.2.2	GitLab Community Edition	2019-09-02 16:18:40	<button>部署</button>

填写 Helm 参数：NFS Server 的 IP (nfs.server)、开放挂载的目录 (nfs.path)。

如果集群中没有 StorageClass，建议将其设置为默认（StorageClass.defaultClass）。

然后点击【部署】。

The screenshot shows the Helm Chart deployment interface for the 'nfs-client-provisioner' chart. The 'values.yaml' file is displayed, specifically the 'nfs' section. The 'server' field is set to '10.0.5.85' and the 'path' field is set to '/nfs'. A red box highlights the 'mountOptions' field, which contains 'nfs'. The 'WebConsole' button is visible at the bottom right.

完成部署后，接下来创建 PVC 测试。

创建 PVC 测试

- 检查 StorageClass 是否设置成功

```
# kubectl get storageclass
NAME          PROVISIONER   AGE
nfs-client (default)  nfs        12d
```

- 创建 PVC

```
# cat auto-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: auto-claim
  annotations:
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

```
# kubectl apply -f auto-claim.yaml
persistentvolumeclaim/auto-claim created
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODE
S	STORAGECLASS	AGE		

```

auto-claim   Bound      pvc-9803c5c5-d2f8-11e9-86f7-525400673e62   1Gi        RWX
nfs-client   13s

```

可以看到 PVC 创建成功。

- 在 NFS 的挂载目录中，可以看到自动创建对应的目录。

```

# ll
总用量 32
drwxrwxrwx 2 root root 4096 9月  9 19:54 default-auto-claim-pvc-9803c5c5-d2f8-11e9-86f7-525400673e62

```

附录 1：默认 StorageClass 不生效

指定了默认的 StorageClass，申请 PVC 不指定 StorageClass 时无法自动创建 PVC。

问题原因

在 `PersistentVolumeClaims` 中提到如果想实现上述功能，除了指定了默认的 StorageClass 外，还需要开启 `DefaultStorageClass`。

下图是解决问题之后的截图，和出问题是配置的差异是 kube-system 命名空间下的 `kube-apiserver` Pod 的启动参数中多了 `DefaultStorageClass`。

The screenshot shows two terminal sessions comparing the startup parameters of the `kube-apiserver` pod in the `kube-system` namespace.

Before Fix:

```

[root@ip-10-0-5-85-m-bcs-k8s-40015 deploy]# cat auto-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: auto-claim
  annotations: {}
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi

```

After Fix:

```

[root@ip-10-0-5-85-m-bcs-k8s-40015 deploy]# kubectl apply -f auto-claim.yaml
persistentvolumeclaim/auto-claim created

```

A red arrow points from the text "排查原因：kube-apiserver.manifest 中未设置 DefaultStorageClass" to the parameter `--enable-admission-plugins` in the second terminal session.

`--admission-control` was deprecated in [1.10](#) and replaced with `--enable-admission-plugins`.

如何解决

`kube-apiserver` 是 **Static Pod**，修改对应配置文件即可。

登录 K8S Master，在 `/etc/kubernetes/manifests/kube-apiserver.manifest` 文件的 `admission-control` 参数后追加 `DefaultStorageClass`。

将该文件移出、移入当前目录来创建 `kube-apiserver` POD。

重新创建 PVC，即可生效。

Mesos 存储

目前，BCS Mesos 解决方案中，在存储支持上使用 docker 的 volume 存储方案，使用 docker volume 的 bind mount 方式，把宿主机 host 上的一个目录挂载进容器当中。如果要使用 ceph rbd 等分布式块存储，可由 BCS 管理员将 rbd 块存储挂载至集群 node 的某个目录上，然后在创建 Mesos 应用时，再把该目录挂载进容器当中。

以 Ceph rbd 为使用示例来说明：

准备存储卷

管理员创建 ceph rbd 块，把 rbd 块设备 map 到集群 node 节点上，格式化并挂载至某个目录中，如 `/mnt/testvolume`

挂载进容器

用户在模板集中创建 Mesos 应用，在容器配置中指定挂载源为 `/mnt/testvolume`

The screenshot shows a container configuration interface with the following details:

- 基础信息** tab is selected.
- 容器名称:** container-1
- 描述:** 请输入256个字符以内
- 镜像及版本:** /public/bcs/istio/citadel 1.1.7 总是在创建之前拉取镜像
- 端口映射** section is present but empty.
- 更多设置** section is present but empty.
- 挂载卷** tab is selected in the bottom navigation bar.
- 挂载卷** configuration:
 - 类型:** 自定义
 - 挂载名:** testvolume
 - 挂载源:** /mnt/testvolume
 - 容器目录:** /data
 - 子路径:** 请输入
 - 只读

业务接入 Helm

本文先简单介绍什么是 Helm，以及蓝鲸容器服务（BCS）提供的 Helm 与标准 Helm 的差异，然后以蓝鲸小游戏为例，介绍如何使用蓝鲸容器服务部署您的 Helm Release。

什么是 Helm

Helm 是 Kubernetes 的一个包管理工具，用来简化 Kubernetes 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

如果您希望进一步了解 Helm，可以阅读 [Helm, Kubernetes 的包管理工具](#) 这篇文章。

“

蓝鲸容器服务 Helm 中使用了 `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

系统会给项目分配 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

项目仓库 公共仓库

图标 Helm Chart名称 版本 描述

gitlab-ce gitlab 2019-09-02 16:18:40 部署

gitlab 2019-06-29 14:55:18 部署

rancher 2019-08-29 14:54:21 部署

jenkins 2019-08-28 15:59:43 部署

wordpress 2019-08-29 11:48:37 部署

WebConsole

如果您有 Chart 需要推送到公共仓库，可以在 Harbor 的页面的 `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

全上传 删除 下载

	名称	状态	版本	创建时间
<input type="checkbox"/>	blueking-nginx-ingress	正常	1	2019年3月19日
<input type="checkbox"/>	rumpetroll	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

推送 Chart 到项目仓库请移步到蓝鲸容器服务的，`容器服务 / Helm / Chart 仓库` 页面，点击右上角 `如何推送 Helm Chart 到项目仓库` 指引，系统为您的项目生成了对应的 Chart 推送命令，可直接复制使用。

The screenshot shows the Helm Chart repository interface. On the left, there's a sidebar with various management options like Cluster, Node, Namespace, Template Set, Variable Management, Helm (with Release List), and a Chart Catalog. The main area displays a table of Helm Charts with columns for Icon, Helm Chart Name, Version, Description, Last Updated, and Action (Deploy). The charts listed are:

Icon	Helm Chart Name	Version	Description	Last Updated	Action
	githab-ce	0.2.2	GitLab Community Edition	2019-09-02 16:18:40	<button>部署</button>
	githab	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。

The screenshot shows the same Helm Chart repository interface as above, but with a prominent red box highlighting the '如何推送Helm Chart到项目仓库' (How to Push Helm Chart to Project Catalog) section. This section contains step-by-step instructions and command-line examples for installing Helm and pushing charts to the repository.

步骤一：安装 Helm

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 systems package manager
goget install helm
```

方式二：手动下载二进制
[下载地址](#)，下载与您操作系统对应的版本
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***.com:443/chartrepo/joyfulgame/ --username=156496958
```

使用 Helm 发布应用

使用蓝鲸容器服务 Helm 部署 Chart

参考上一步推送完 Chart 之后，您就可以在 **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

- 容器服务 / Helm / Release列表 菜单，点击 **部署Helm Chart**

- 容器服务 / Helm / Chart仓库 菜单，选择目标 Chart 点击 部署

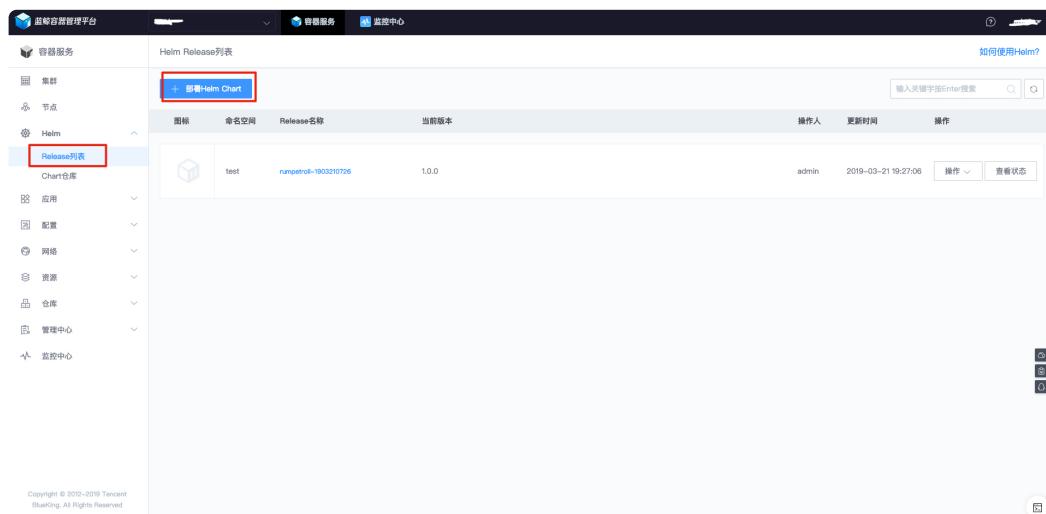


图 1.Helm Release 创建入口 - 从 Helm / Release列表 菜单进入

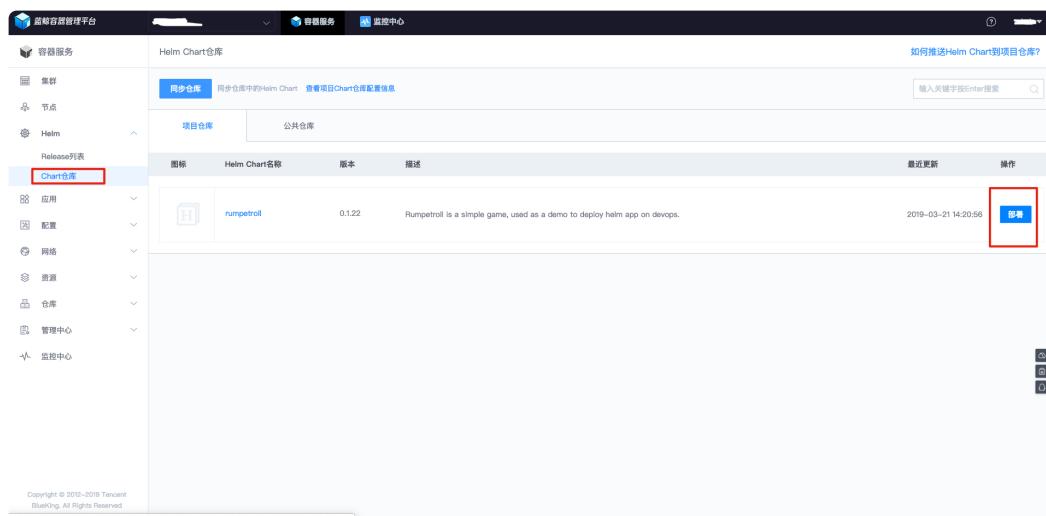


图 2.Helm Release创建入口 - 从 Helm / Chart 仓库 菜单进入

● 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。

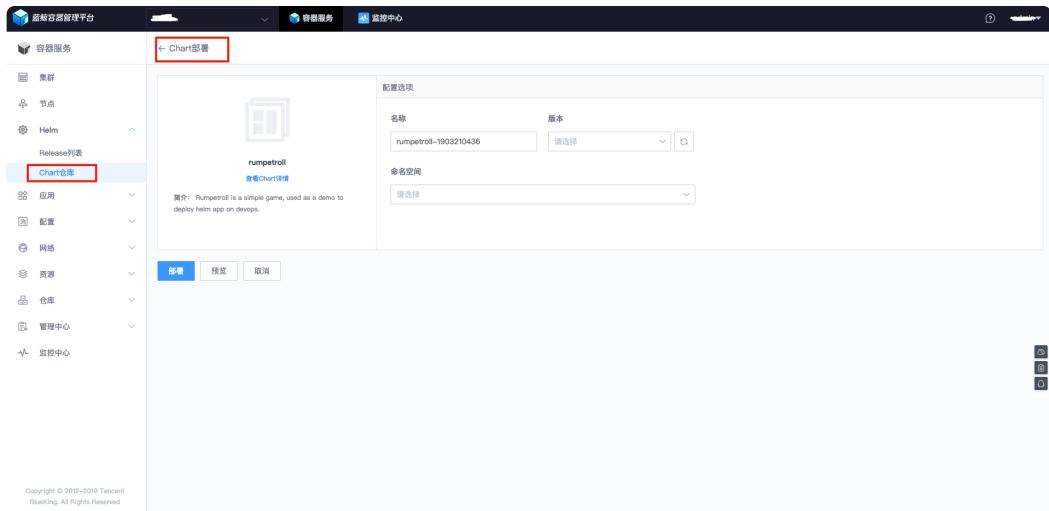


图 3.Helm Release创建参数页面

- 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。

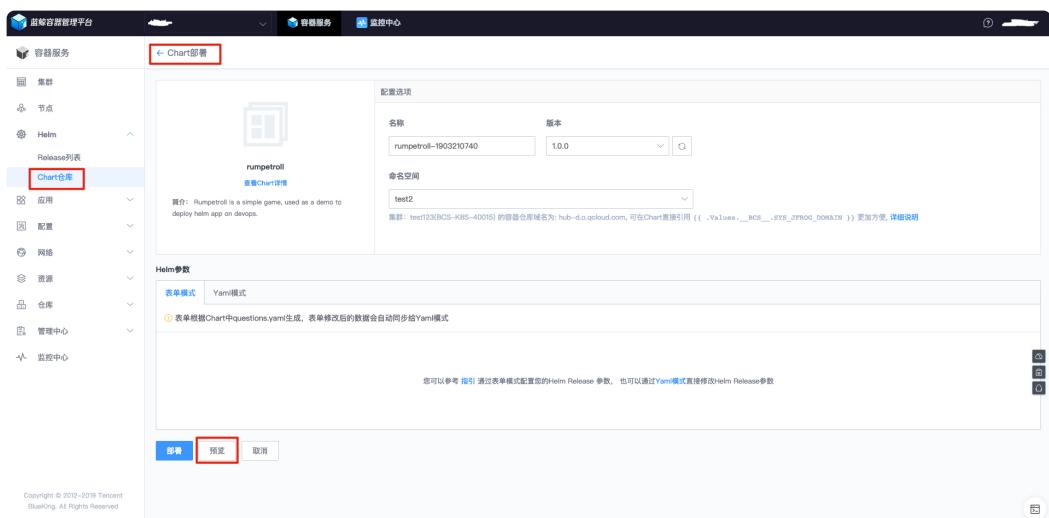


图 4.Helm Release创建 预览 入口

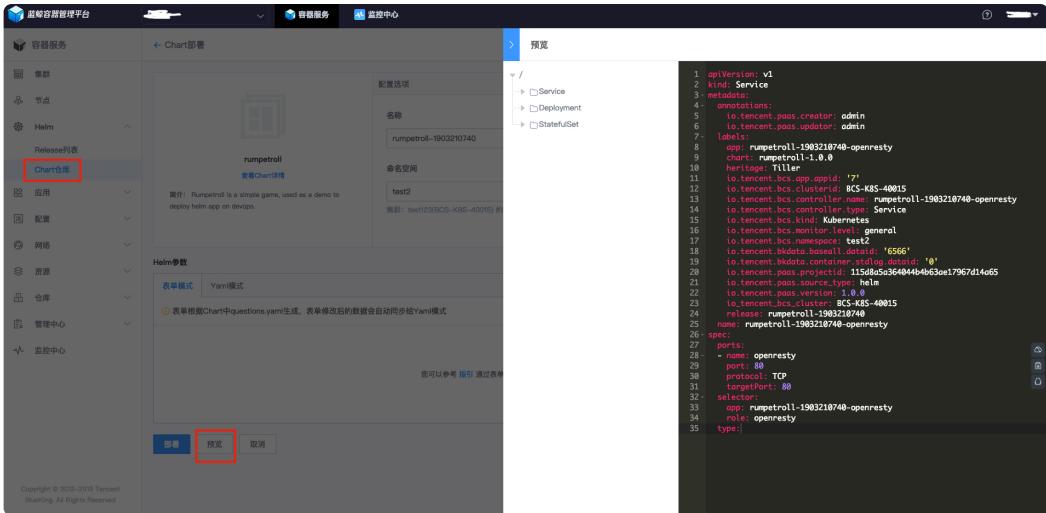


图 5.Helm Release 创建预览页面示例

- 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

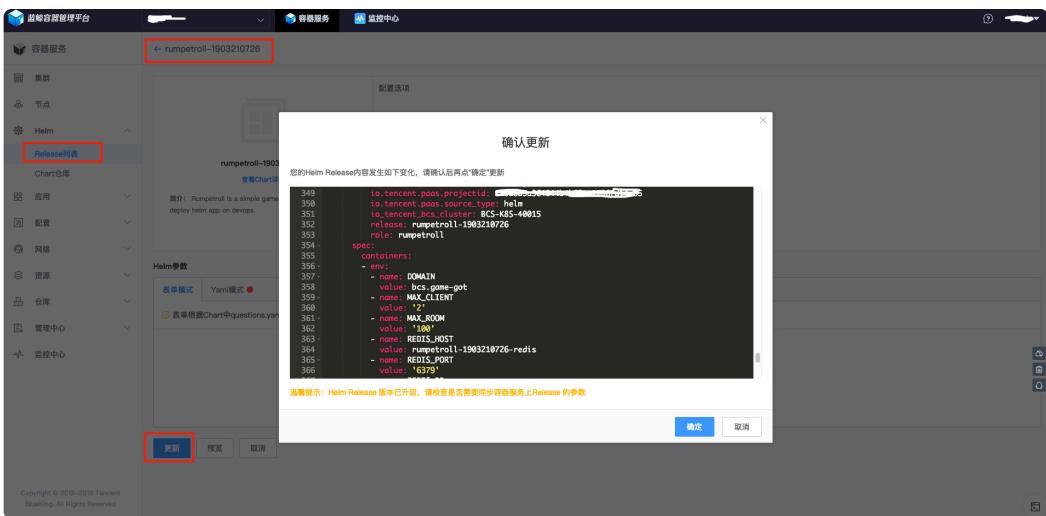


图 6.Helm Release 升级-对比查看功能

- 回滚

- 针对您对 Helm Release 的变更，蓝鲸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** 操作按钮。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。

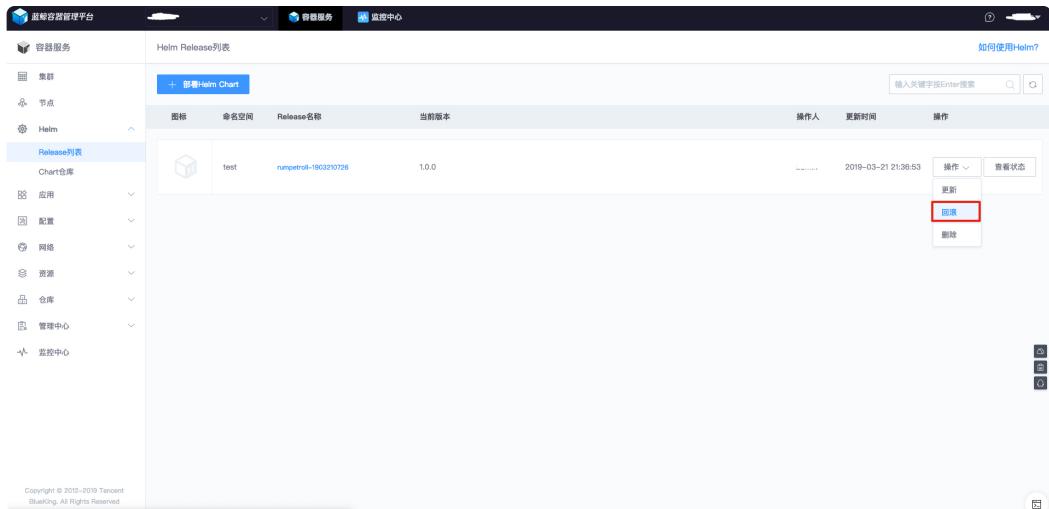


图 7.Helm Release回滚入口

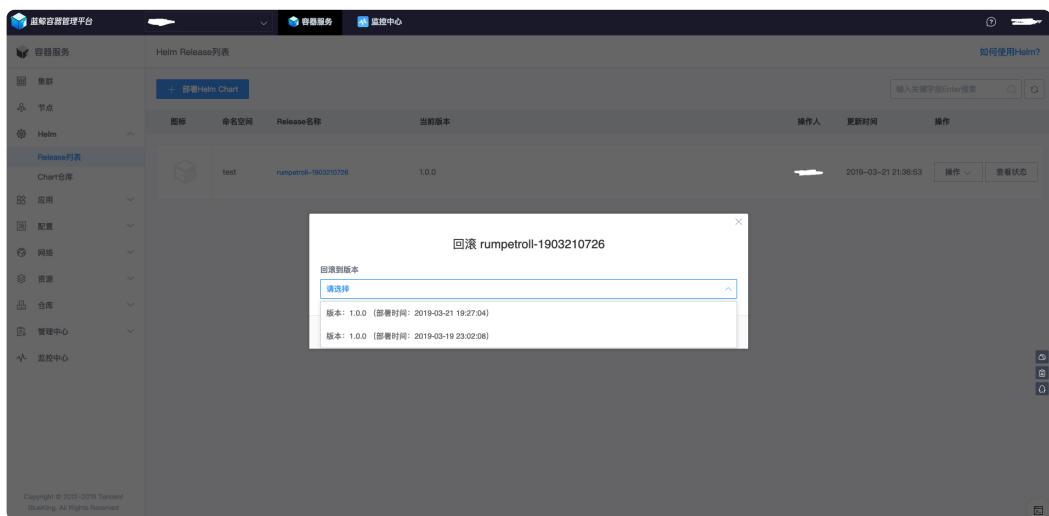


图 8.Helm Release回滚版本选择

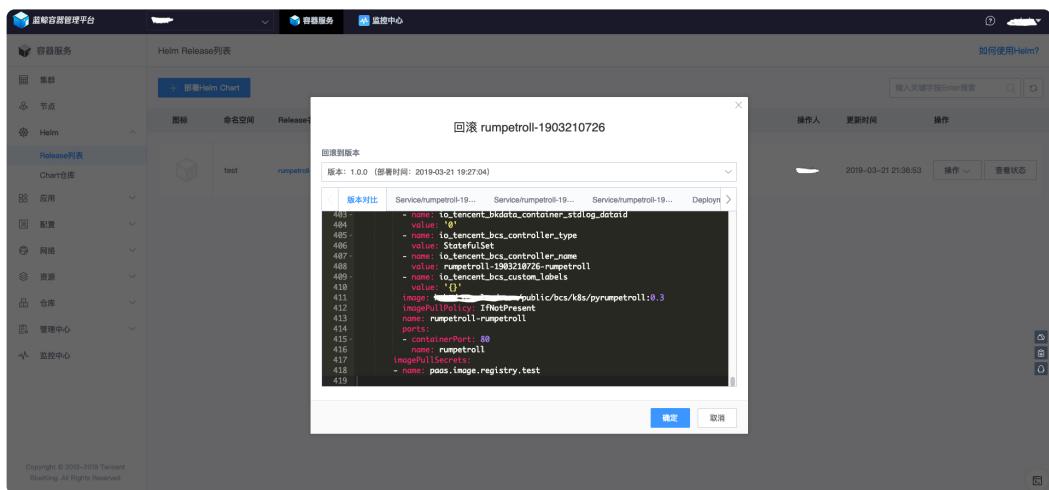
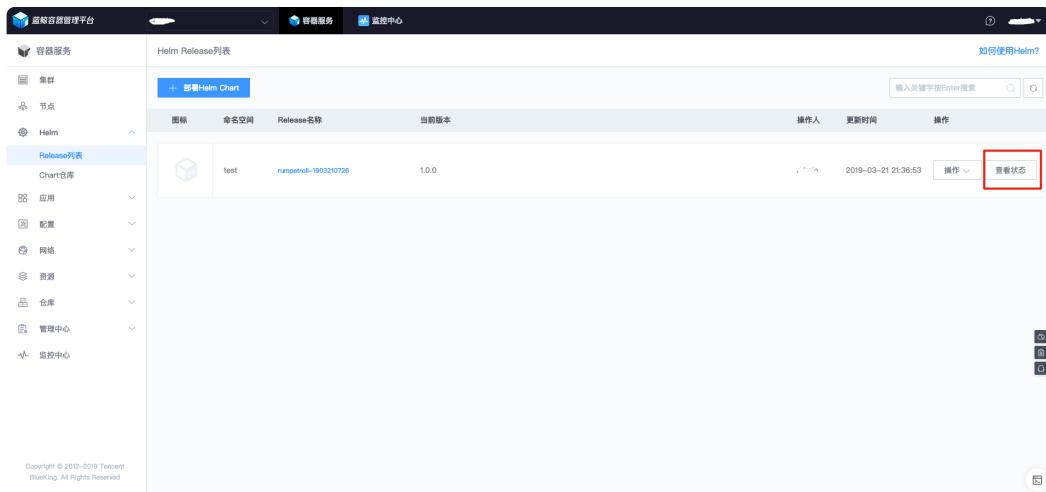


图 9.Helm Release回滚对比预览页面

检查 Helm Release 状态

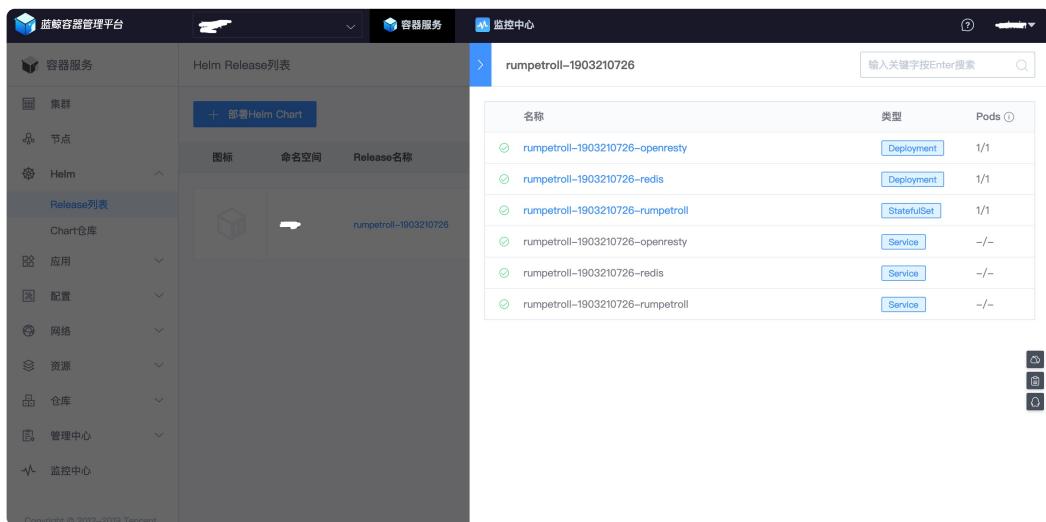
- 通过 Helm Release 列表页面查看

- 在 容器服务 / Helm / Release列表 页面中，选中 查看状态 即可查看资源的状态信息。



The screenshot shows the Helm Release list interface. On the left is a sidebar with '容器服务' (Container Service) selected, followed by '集群', '节点', 'Helm', 'Release列表' (which is highlighted in blue), 'Chart仓库', '应用', '配置', '网络', '资源', '仓库', '管理中心', and '监控中心'. The main area has a title 'Helm Release列表' and a sub-section '部署Helm Chart'. It includes a search bar with placeholder '输入关键字按Enter搜索' and a date range selector '2019-03-21 21:36:53'. A table lists releases with columns: 图标 (Icon), 命名空间 (Namespace), Release名称 (Release Name), 当前版本 (Current Version), 操作人 (Operator), 更新时间 (Update Time), and 操作 (Operations). One row is shown: 'test' namespace, 'rumpetroll-1903210726' release name, '1.0.0' version, '操作' operator, '2019-03-21 21:36:53' update time, and '操作' operations. A red box highlights the '查看状态' (View Status) button in the operations column. The bottom right corner of the interface has three small icons.

图 10.Helm Release状态查看入口



The screenshot shows the detailed status of the 'rumpetroll-1903210726' release. The left sidebar is identical to Figure 10. The main area has a title 'rumpetroll-1903210726'. A search bar at the top right contains 'rumpetroll-1903210726'. Below it is a table with columns: 名称 (Name), 类型 (Type), and Pods (Pods). The table lists six entries, each with a green circular icon and a checkmark:

名称	类型	Pods
rumpetroll-1903210726-openresty	Deployment	1/1
rumpetroll-1903210726-redis	Deployment	1/1
rumpetroll-1903210726-rumpetroll	StatefulSet	1/1
rumpetroll-1903210726-openresty	Service	-/-
rumpetroll-1903210726-redis	Service	-/-
rumpetroll-1903210726-rumpetroll	Service	-/-

The bottom right corner of the interface has three small icons.

图 11.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，您可以不用登录跳板机，直接使用 kubectl 命令查看业务容器的状态。

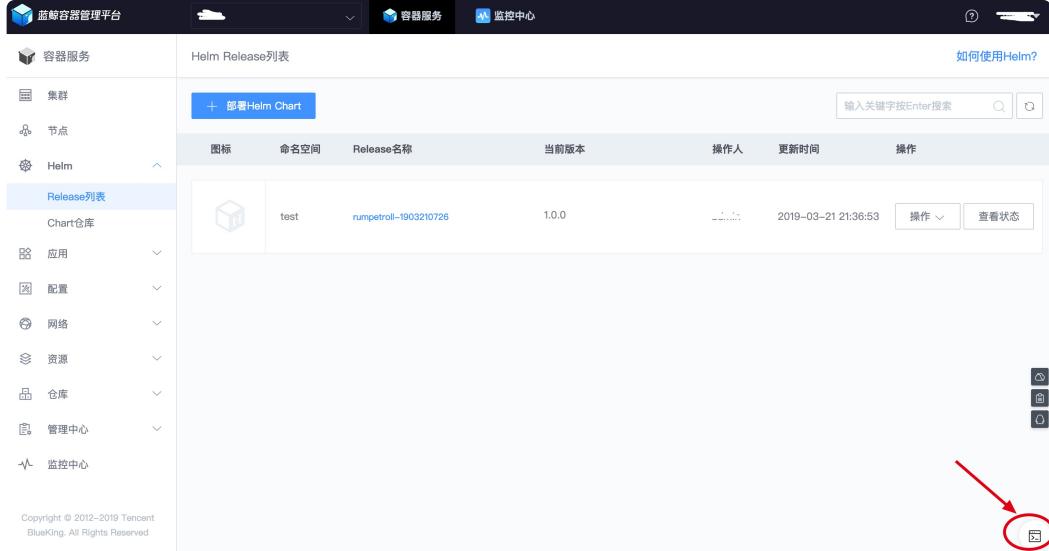


图 12.WebConsole 入口

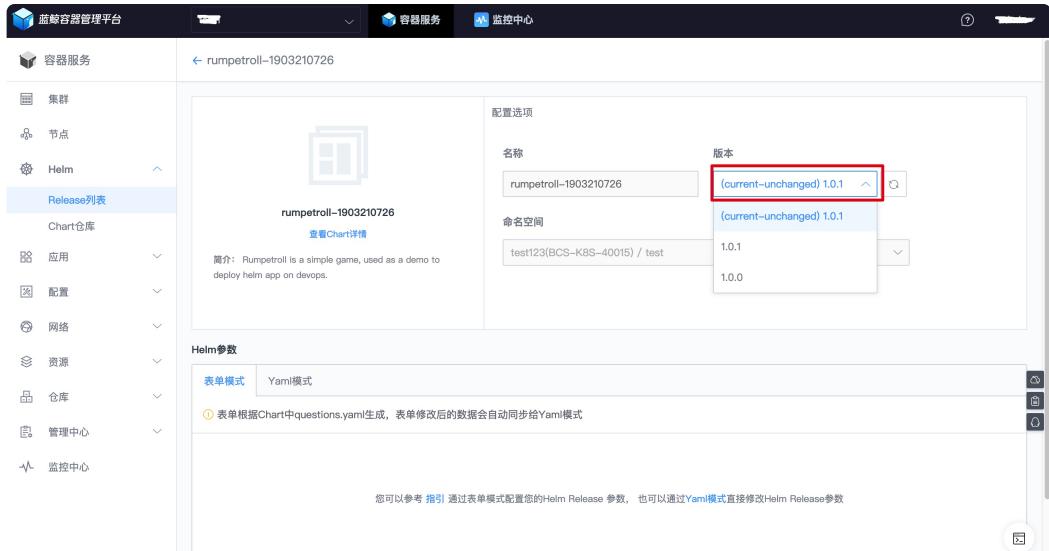
The screenshot shows a terminal window titled 'BCS Console /'. The URL in the address bar is 'https://bk.tencent.com/p/web_console/'. The terminal output shows:

```
# ##### Welcome To BCS Console #####
# Guide: https://bk.tencent.com/docs/
# 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
#####
root:~$ kubectl get pod -n test
NAME                      READY   STATUS    RESTARTS   AGE
rumpetroll-1903210726-openresty-5d849b8c9-przlf   1/1     Running   0          3h32m
rumpetroll-1903210726-redis-68b6fdc467-8v59s      1/1     Running   0          3h32m
rumpetroll-1903210726-rumpetroll-0                 1/1     Running   0          3h32m
root:~$
```

图 13.WebConsole 使用效果图

Helm Release 的升级的版本

通过蓝鲸容器服务对 Helm Release 做升级时，有个默认选中的版本，也就是当前版本，在当前版本的版本号左边有个备注，备注的作用用于说明，Release 所使用的版本，是否发生过变化，或者已经被删除。如下图所示，蓝鲸小游戏 rumpetroll 当前版本是 1.0.0，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。



- 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 示例	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

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

方式 2：通过模板的方式使用

```
 {{/*
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 -}}
{{- end -}}

```

在 Chart 中使用方式如下：

```
{{ 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。

```
- 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，您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 <code>values.yaml</code> 中的字断，及联字段使用 <code>.</code> 进行隔开，比如 <code>persistence.enabled</code> .
label	string	true	表单项的标签 <code>label</code> .
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 <code>string</code> 类型 (当前支持的字断的类型为： <code>string, boolean, int, enum, password, storageclass and hostname</code>).
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 <code>enum</code> 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 <code>show_if: "serviceType=Nodeport"</code>
show_subquestion_if	string	false	如果当前值为 <code>true</code> 或者可选项的值，则该子问题会被显示出来. 比如 <code>show_subquestion_if: "true"</code>

`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.

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

注意： WebConsole 目前只支持K8S集群

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 Service interface. On the left is a sidebar with navigation items like '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' (Networking), '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 cluster statistics for 'weixin' (BCS-K8S-40015), including CPU usage (18.75%), memory usage (8.48%), and disk usage (25.01%). A large '+' button is available to create a new cluster. In the bottom right corner, there is a 'weixin' button and a 'WebConsole' button with a dropdown arrow.

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'. It displays the output of several kubectl commands:

```
# Welcome To BCS Console
# Guide: https://docs.bk.tencent.com/bcs
# 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 替代; 使用 Alt-Num 切换 Tab #
root::$ 
root::$ 
root::$ 
root::$ kubectl get all -n web-console
NAME                               READY   STATUS    RESTARTS   AGE
pod/kubectld-bcs-k8s-40015-ublueking-71ece8db46f7   1/1     Running   0          7m33s
root::$ 
root::$ 
root::$ 
root::$ 
root::$ 
root::$ kubectl get deploy -n dev
NAME            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bk-bcs-test      1         1         1           1          9h
blueking-nginx-ingress-controller   2         2         2           2          25d
blueking-nginx-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-nginx-blue      2         2         2           0          19d
web-nginx-green     2         2         2           0          8d
root::$ 
```

kubectl 常用命令介绍

查询集群信息

```
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

```
kubectl get nodes
```

- 获取 NameSpace 信息

```
kubectl get namespace
```

- 查询 Pods 列表

```
kubectl get pods
```

- 以 JSON 格式输出 Pod 的详细信息

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

- 查询打印 DEBUG 信息

```
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。

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

Help 查看帮助

```
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](#)。

快速入门

1

2

3

4

5

6

7

登录控制台

创建项目

创建集群

添加节点

创建命名空间

创建服务实例

确认完成

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）

- 创建新项目：进入项目管理页面，点击“创建新项目”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，申请加入已有项目用户组来获取项目使用权限

关键项说明： - 当您选择使用蓝鲸容器服务部署业务，创建项目后，进入容器服务，还需要关联“蓝鲸配置平台（CMDB）”上的某个业务，该项目集群节点将从该业务机器资源池中拉取 - 您可以根据业务情况，选择使用容器编排类型为 **Kubernetes** 或 **Mesos**（注意：一旦创建集群后，容器编排类型将不可更改）

项目管理： 您可以在项目管理页面管理您的项目基础信息。

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“配置”→“命名空间”，点击“新建命名空间”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，系统将初始“示例模板集”到您的项目模板集库中，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

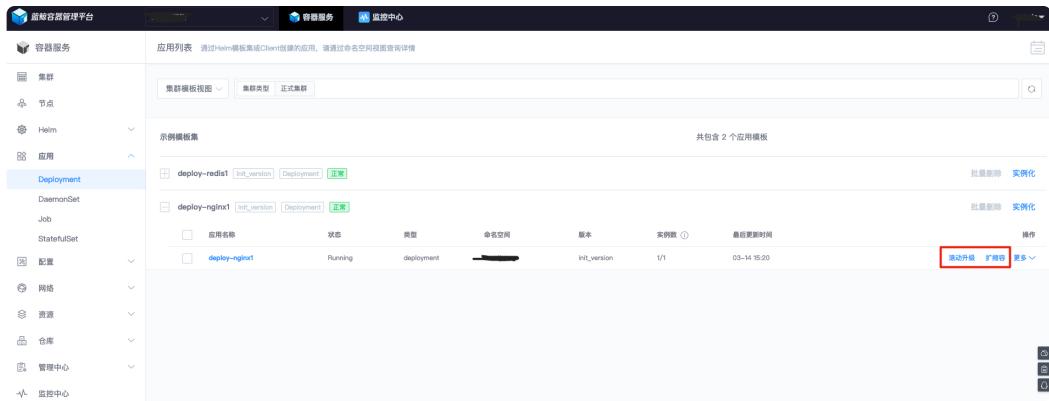
确认完成

《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。

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with navigation links like '容器服务', '集群', '节点', 'Helm', '应用' (Deployment, DaemonSet, Job, StatefulSet), '配置', '网络', '资源', '仓库', '管理中心', and '监控中心'. The main content area is titled 'deploy-nginx1'. It displays basic info: 应用名称: deploy-nginx1, 创建时间: 2019-03-14 15:20:45, 更新时间: 2019-03-14 15:20:45, 所在命名空间: [REDACTED], 所属集群: [REDACTED]. Below this are two line charts: 'CPU 使用率' and '内存使用率', both showing usage over time from 15:21 to 16:00. The CPU chart has values from 0% to 1%, and the memory chart has values from 0% to 1%. At the bottom is a 'Pod管理' table with columns: 事件, 标签, 备注. It lists one pod: deploy-nginx1-8494b4768-b6zcv, 状态: Running, Host IP: [REDACTED], Pod IP: [REDACTED], 存活时间: 40分钟. There are also buttons for '重新调度', '日志', and '容器'.

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=tPp5GwAmMPIrzXhyA8X` 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

```
# 开启游戏
curl -X POST -H 'Content-Type: application/x-www-form-urlencoded' -d 'func_code=is_star
t&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_star&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

```
# 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

```
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（以 K8S 编排为例，BCS 同时还支持 Mesos）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

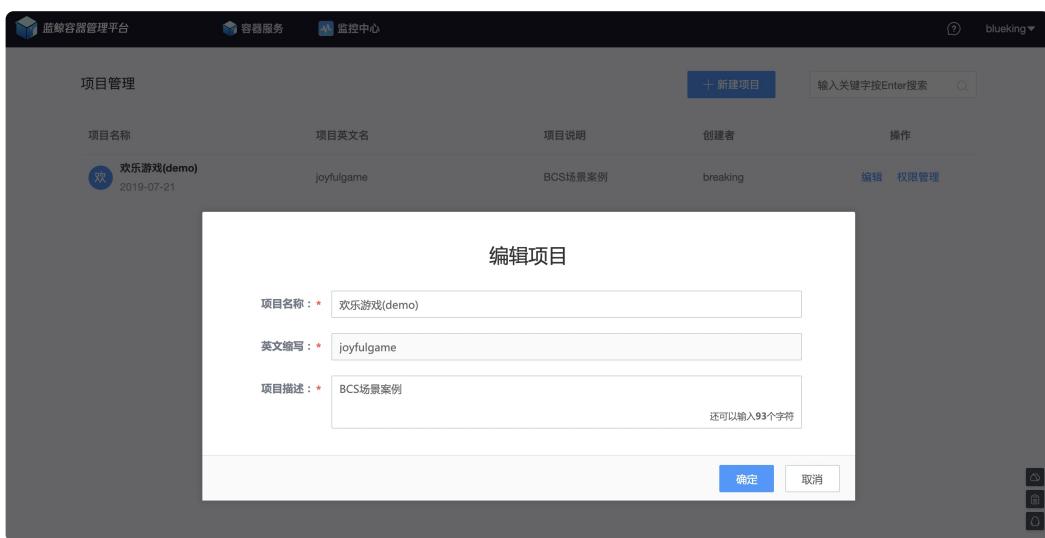
操作步骤

- 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)。

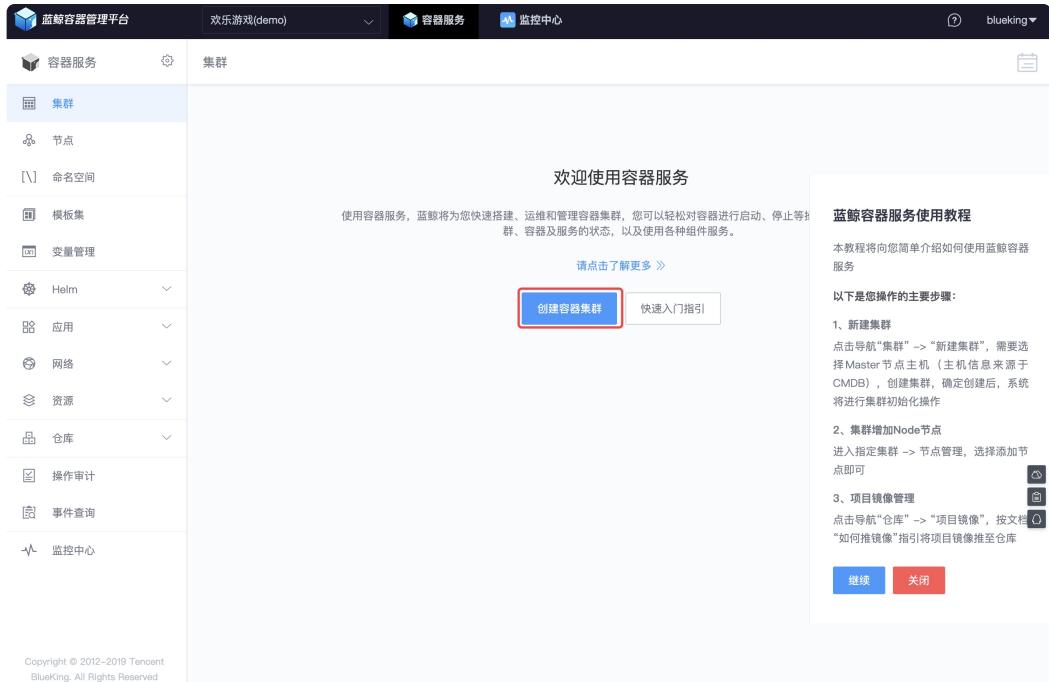


然后选择容器编排类型为 Kubernetes，关联 前提条件 中提到的 CMDB 业务，点击 启用容器服务。

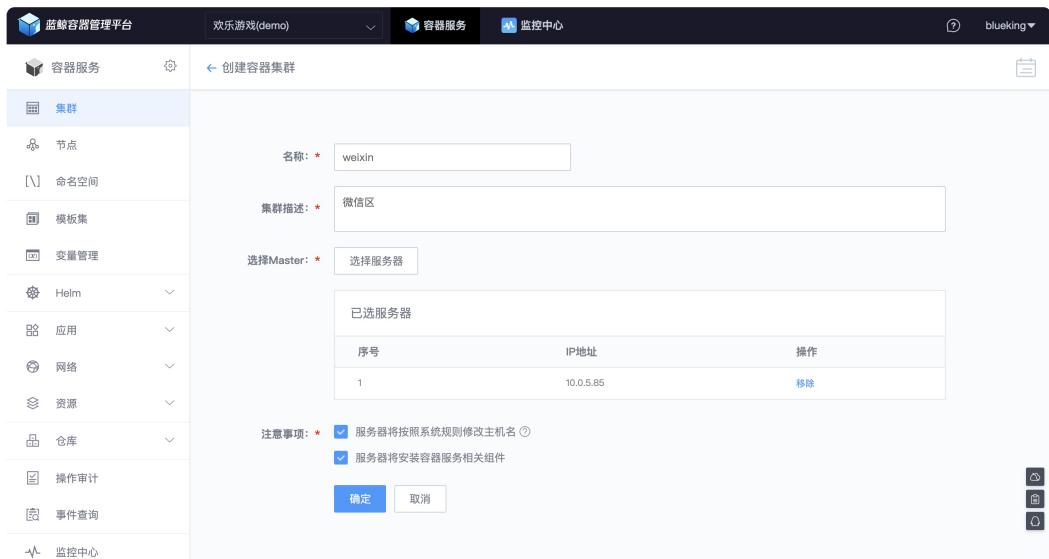


新建集群

启用容器服务后，进入容器服务欢迎页，点击 创建容器集群。

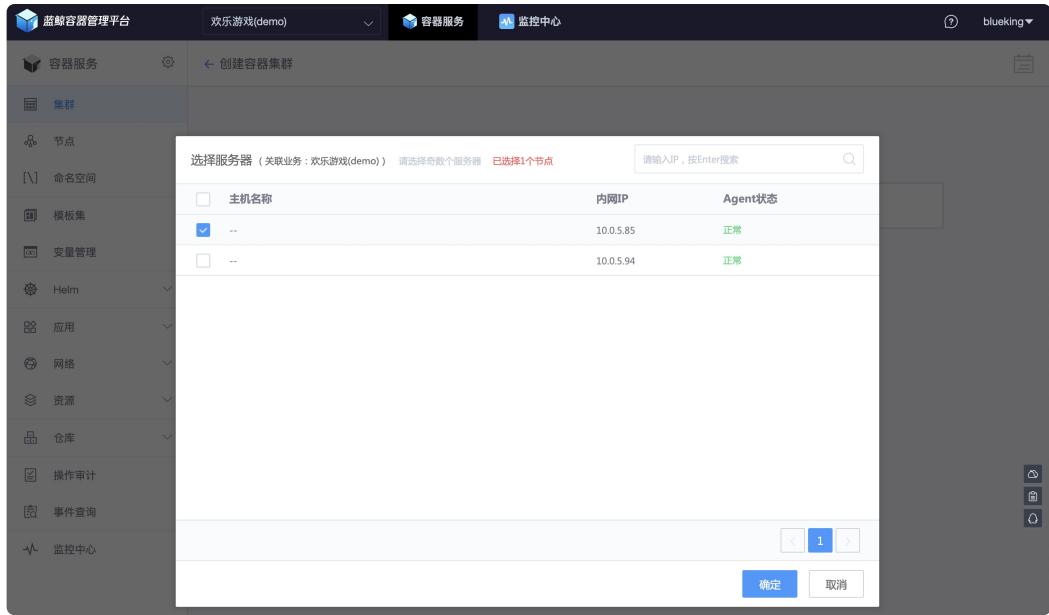


按提示填写集群的基本信息。

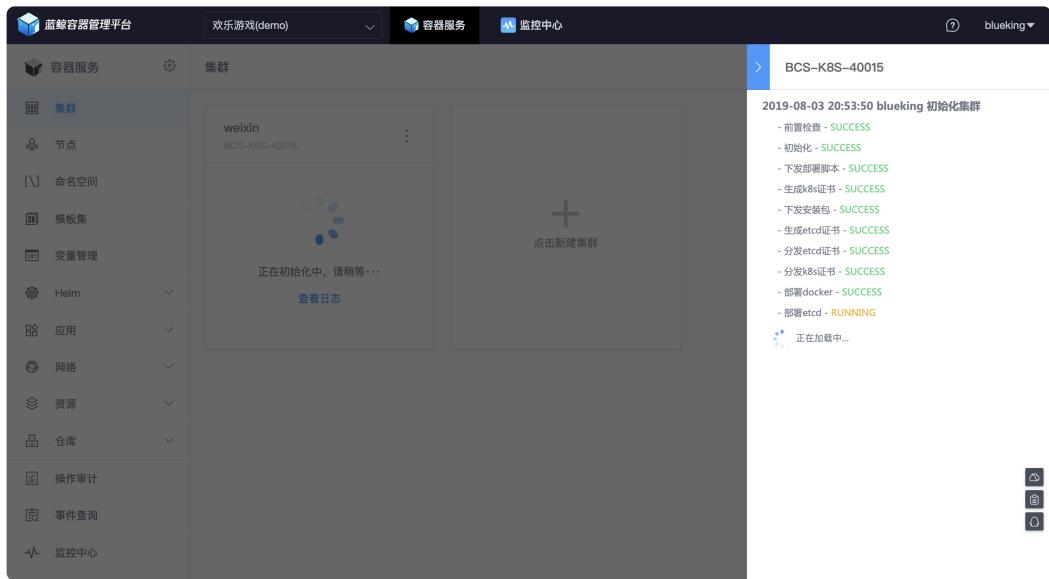


容器服务的集群划分和传统单体应用在 CMDB 中的集群划分很类似，可以按照 地域（如 华北区）或者 完全独立的应用集合（微信区）来划分。

选择 1 台云主机作为 Master。



点击 **确定** 后，集群开始初始化。



点击 **节点管理**

The screenshot shows the BlueKing Container Management Platform interface. The top navigation bar includes links for '容器服务' (Container Service), '欢乐游戏(demo)', '容器服务' (Container Service), and '监控中心' (Monitoring Center). The user is currently in the '集群' (Cluster) section under the '容器服务' menu. On the left sidebar, there are various management options like '节点' (Nodes), '命名空间' (Namespaces), '模板集' (Template Sets), etc. The main content area displays a cluster named 'weixin' (BCS-K8S-40015). A context menu is open over this cluster, with the '节点管理' (Node Management) option highlighted and surrounded by a red box. Other options in the menu include '总览' (Overview), '集群信息' (Cluster Information), and '删除' (Delete). To the right of the cluster details, there's a large empty area with a plus sign and the text '点击新建集群' (Click to create a new cluster). On the far right of the interface, there are three small circular icons.

点击 **添加节点**，按提示节点添加。

This screenshot shows the '节点管理' (Node Management) page for the 'weixin' cluster. The top navigation bar and sidebar are identical to the previous screenshot. The main content area is titled 'weixin (BCS-K8S-40015)'. It features tabs for '总览' (Overview), '节点管理' (Node Management), and '集群信息' (Cluster Information). The '节点管理' tab is active and has a red box around its title. Below the tabs, there's a search bar with a red box around the '添加节点' (Add Node) button. A table lists one node: '10.0.5.94' which is '初始化中' (Initializing). There are columns for '主机名/IP' (Host Name/IP), '状态' (Status), '容器数量' (Number of Containers), 'CPU' (CPU), '内存' (Memory), '磁盘IO' (Disk I/O), and '操作' (Operations). At the bottom of the table, it says '共计1条 每页 5 条' (1 total item, 5 items per page) with navigation arrows. On the far right, there are three small circular icons.

至此，新建集群完毕。可以看到集群的基础信息。

The screenshot shows the BlueKing Container Management Platform interface. On the left is a sidebar with various navigation options: 容器服务 (Container Service), 集群 (Cluster), 节点 (Nodes), 命名空间 (Namespace), 模板集 (Template Set), 变量管理 (Variable Management), Helm, 应用 (Application), 网络 (Network), 资源 (Resources), 仓库 (Repository), 操作审计 (Audit), 事件查询 (Event Query), and 监控中心 (Monitoring Center). The main area displays a cluster named 'weixin' (BCS-K8S-40015) with resource usage metrics: CPU 使用率 (CPU Usage Rate) at 29.13%, 内存使用率 (Memory Usage Rate) at 14.76%, and 磁盘使用率 (Disk Usage Rate) at 8.53%. A large button labeled '点击新建集群' (Click to Create New Cluster) is visible.

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。

This screenshot shows the detailed performance metrics for the 'weixin' cluster. It includes three line charts: CPU 使用率 (CPU Usage Rate) at 29.13% (1.17 of 4.00 Shares), 内存使用率 (Memory Usage Rate) at 14.75% (1.10GB of 7.46 GB), and 磁盘使用率 (Disk Usage Rate) at 8.53% (26.43GB of 309.81 GB). Below the charts, there are two circular progress indicators: '节点' (Nodes) at 100% and '命名空间' (Namespaces) at 0. A 'WebConsole' button is also present.

BCS 快速构建 Nginx 集群

新建命名空间

新建命名空间 **dev**

容器服务 - 命名空间

名称	所属集群	变量	操作
preprod	weixin	--	设置变量值
dev	weixin	--	设置变量值
prod	weixin	--	设置变量值

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**。

容器服务 - 模板集

模板集名称	类型	容器 / 镜像	操作
示例模板集 最新版本: init_version	自定义	container-redis-default / redis:1.0 container-nginx-default / rumpetroll-openresty:0.51 container-rumpetroll-v1 / pyrumpetroll:0.3	实例化 更多

按提示，填写 **Deployment**

容器服务 - web-nginx

应用名称: * web-nginx 实例数量: * 2 2个副本, Pods 导入YAML

重要级别: 一般 重要 不重要

描述: 请输入256个字符以内

标签: * app = web-nginx (+) 添加至选择器

更多设置 Pod模板设置



填写 Service

名称: **web-nginx**

关联应用: **web-nginx** 1. 关联Deployment

关联标签: **app:web-nginx**

Service类型: **NodePort** 2. 访问Service的方式, NodePort会占用Node的端口资源以提供外部访问。
ClusterIP仅用于集群内部访问。
如果想通过LoadBalancer将服务提供外部访问, 需要二次开发做集成

端口映射: 端口名称 **http** 端口 **8088** 协议 **TCP** 目标端口 **30008** 3. Service暴露的端口
4. Deployment中定义容器监听的端口 5. Node上监听的端口

标签管理: **app = web-nginx**

小提示: 同时粘贴多行“键=值”的文本会自动添加多行记录
Label用于标识Service, 在查询的时候会用到, 比如 kubectl get services -l app=web-nginx -n dev

实例化

为避免多成员同时编辑, 引起内容或版本冲突。建议在编辑时, 开启保护功能。 (当前版本号: v1)

Deployment Service Configmap Secret DaemonSet Job StatefulSet Ingress



检查部署效果

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

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

以及其运行指标

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with various navigation options like Cluster, Node, Namespace, Template Set, Variable Management, Helm, Application, Deployment, DaemonSet, Job, StatefulSet, Network, Resource, and Warehouse. The 'Deployment' section is currently selected. In the main area, it displays information for a deployment named 'web-nginx'. It includes a summary table with fields: 应用名称 (Application Name: web-nginx), 创建时间 (Creation Time: 2019-08-08 16:58:19), 更新时间 (Update Time: 2019-08-08 16:58:19), 所在命名空间 (Namespace: dev), and 所属集群 (Cluster: weixin). Below this is a CPU Usage Rate chart from 17:00 to 17:10, showing usage between 0% and 1%. To the right is a Memory Usage chart for the same period, showing usage between 0MB and 5MB. Under the 'Pod管理' tab, there are two entries: 'web-nginx-678bb9c4fb-nxwrf4' and 'web-nginx-678bb9c4fb-m8cf4', both listed as 'Running'. Each entry shows Host IP (10.0.5.94), Pod IP (172.32.1.17 and 172.32.1.18 respectively), and a存活时间 (Liveness) of 11分48秒 (11 minutes and 48 seconds).

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

```
[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。

```
[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

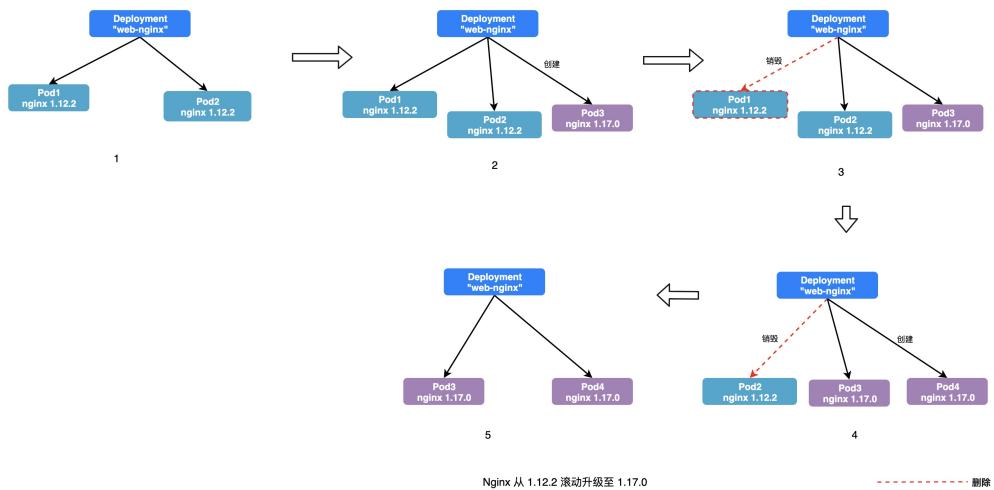
操作步骤

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

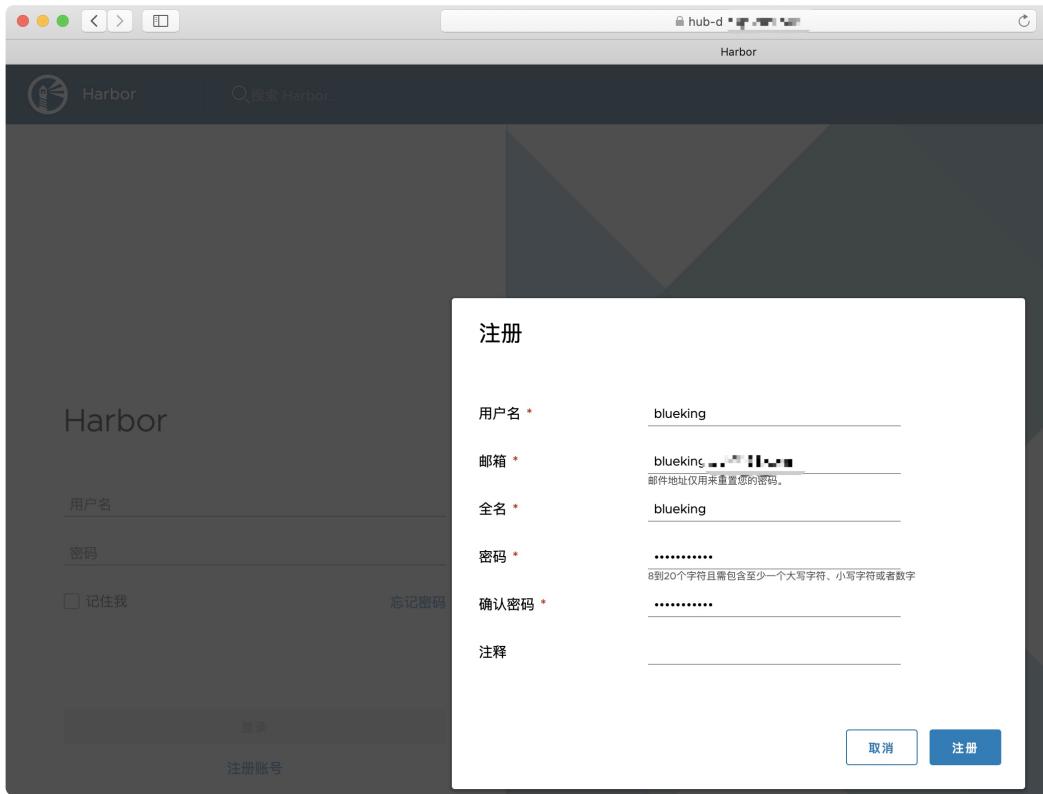
参照 [Harbor 仓库使用指南](#)，将镜像 Nginx:1.17.0 推送至 BCS 公共镜像仓库。

注册镜像仓库账号

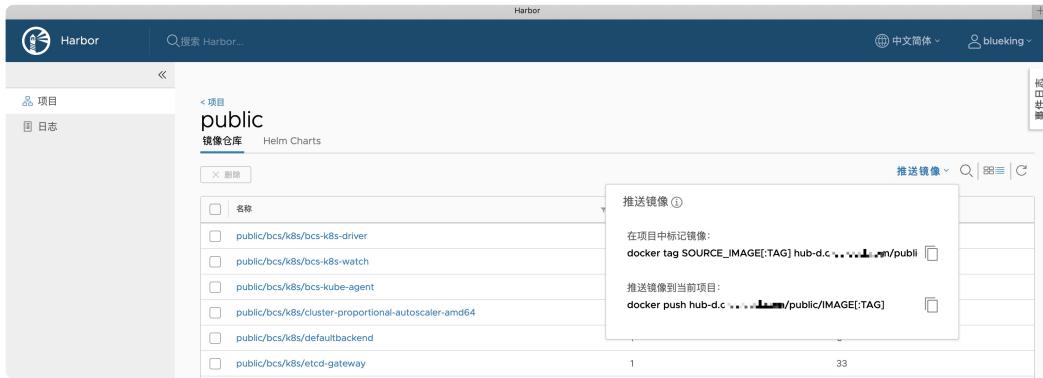
在[部署 BCS](#)的中控机上获取镜像仓库的访问地址。

```
# 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`。

```
# docker pull nginx:1.17.0
1.17.0: Pulling from library/nginx
fc7181108d40: Pull complete
c4277fc40ec2: Pull complete
780053e98559: Pull complete
Digest: sha256:bdbf36b7f1f77ffe7bd2a32e59235dff6ecf131e3b6b5b96061c652f30685f3a
Status: Downloaded newer image for nginx:1.17.0
```

规范镜像 tag 为仓库要求的格式。

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

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。

```
# 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: sha256:079aa93463d2566b7a81cbdf856afc6d4d2a6f9100ca3bcbece24ade92c9a7fe
size: 948
```

在镜像仓库中，可以找到刚推送的 Nginx:1.17.0 镜像。

The screenshot shows the Harbor UI for the public/nginx repository. The 'Image' tab is selected. A single row is listed:

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。

The screenshot shows the BCS UI for the public/nginx repository. The 'Image' tab is selected. A single row is listed:

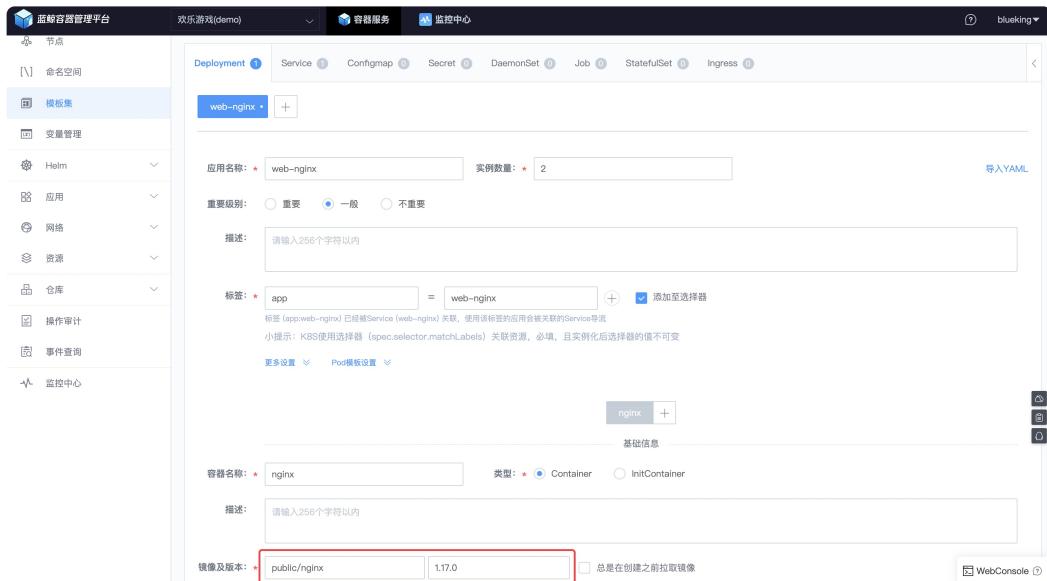
名称	大小	最近更新时间	地址
1.17.0	42.7 MB	2019-06-11 08:03:10	hub-d.c.*****.com:443/public/nginx:1.17.0

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`

```
[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 Service interface. On the left sidebar, '模板集' (Template Set) is selected. In the main area, under 'Deployment' tab, a deployment named 'web-nginx' is listed. The configuration details include:

- 应用名称: web-nginx
- 实例数量: 2
- 重要级别: 一般 (selected)
- 描述: (empty)
- 标签: app = web-nginx
- 容器名称: nginx
- 类型: Container (selected)
- 镜像及版本: public/nginx 1.17.0 (highlighted with a red box)

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 '滚动升级' (Rolling Update) strategy, the following parameters are configured:

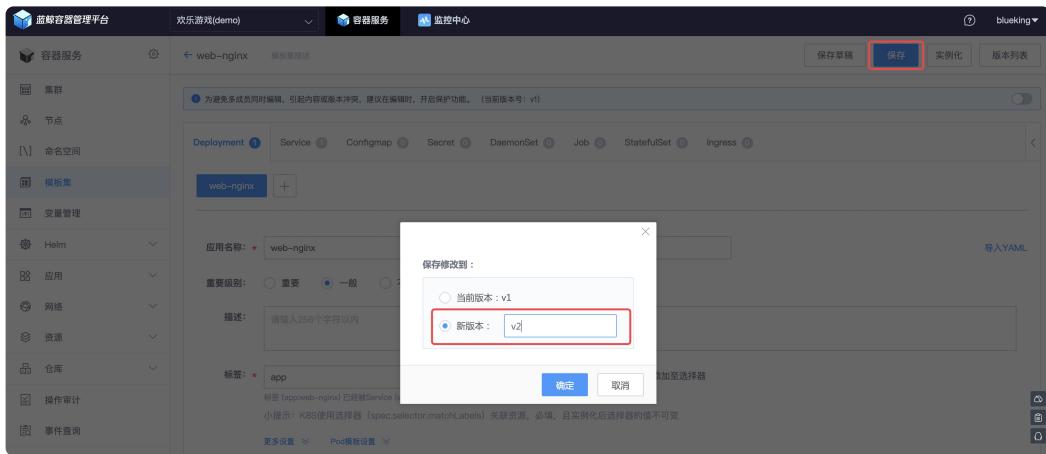
- 类型: 滚动升级 (selected)
- maxUnavailable: 1 (highlighted with a red box)
- maxSurge: 0
- minReadySeconds: 0 秒

“

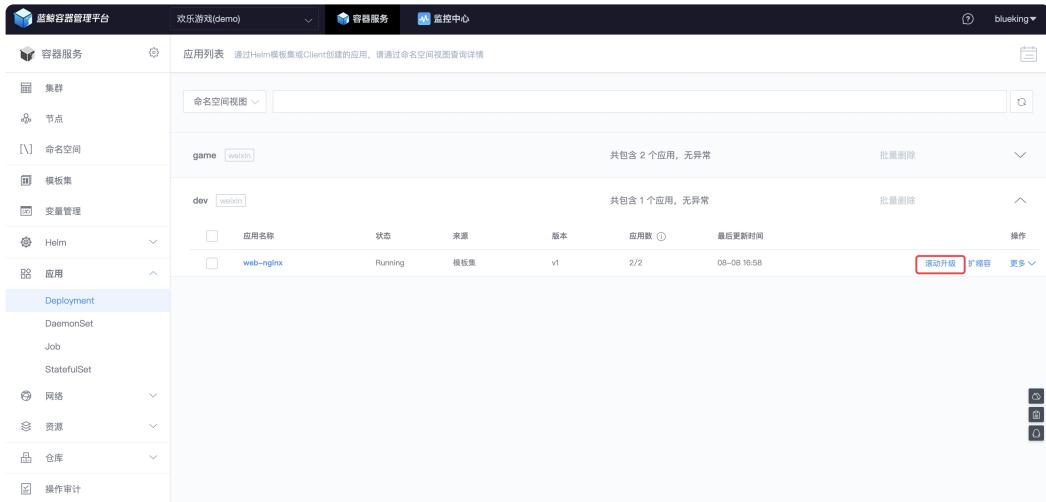
- `maxUnavailable`：滚动升级期间，考虑应用容量，不可用 Pod 的数量上限
- `maxSurge`：滚动升级期间，考虑集群资源，超出期望 Pod 的数量上限
- `minReadySeconds`：滚动升级期间，考虑可用性，探测 Pod 正常后转为可用的时间

”

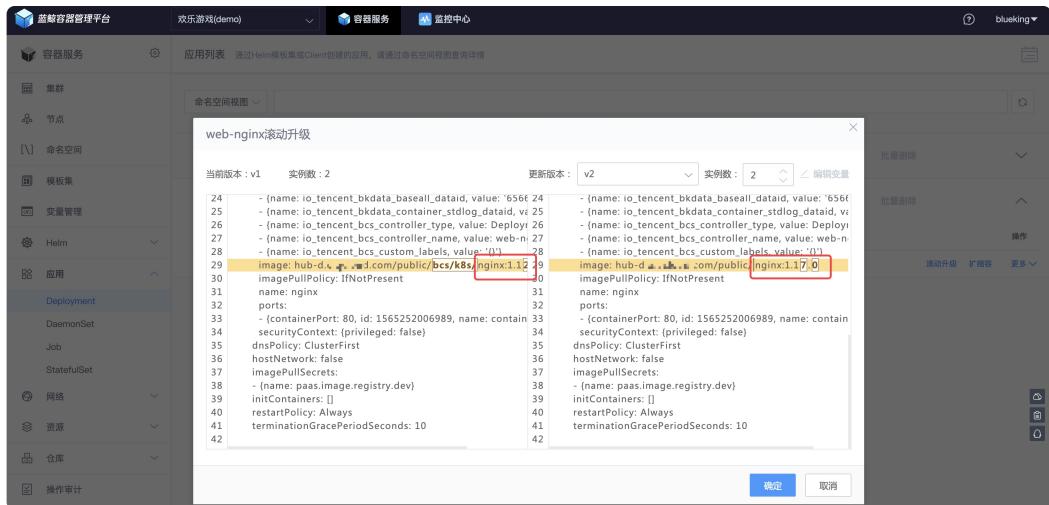
修改完镜像的版本后，接下来【保存】模板集，填写【新版本】的版本号。



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 `images` 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

```
root:~$ kubectl get pods -n dev -o wideNAME                                READY   STATUS    R
ESTARTS   AGE      IP           NODE
web-nginx-678bb9c4fb-m8cf4   1/1     Running   0          134m   172.32.1.18   ip-10-0-
5-94-n-bcs-k8s-40015   <none>
web-nginx-678bb9c4fb-nxwf4   1/1     Running   0          134m   172.32.1.17   ip-10-0-
5-94-n-bcs-k8s-40015   <none>

root:~$ kubectl get pods -n dev -o wide
NAME        READY   STATUS    RESTARTS   AGE      IP           NODE
NOMINATED NODE
web-nginx-f95ffc78d-cxhrq   1/1     Running   0          21s    172.32.1.20   ip-10-0-5-
94-n-bcs-k8s-40015   <none>
web-nginx-f95ffc78d-pqtcj   1/1     Running   0          14s    172.32.1.21   ip-10-0-5-
94-n-bcs-k8s-40015   <none>
```

可以看到 Nginx 版本已经从 1.12.2 更新为 1.17.0

```
[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

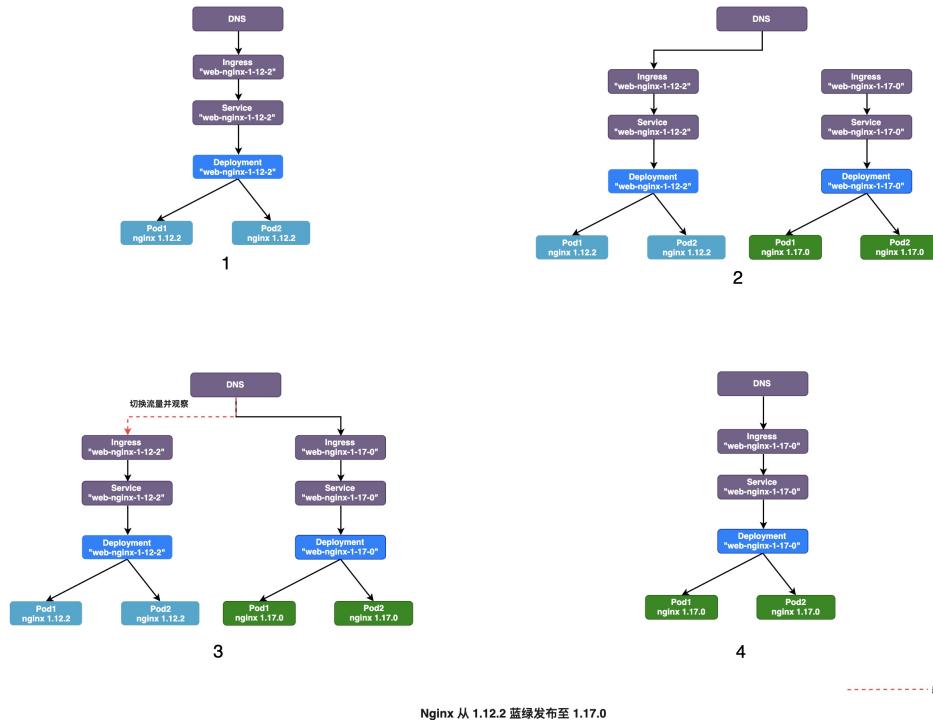
1. 应用的蓝绿发布逻辑介绍
2. 使用 K8S 资源准备版本
3. 使用 K8S 资源准备新版本

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。



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。

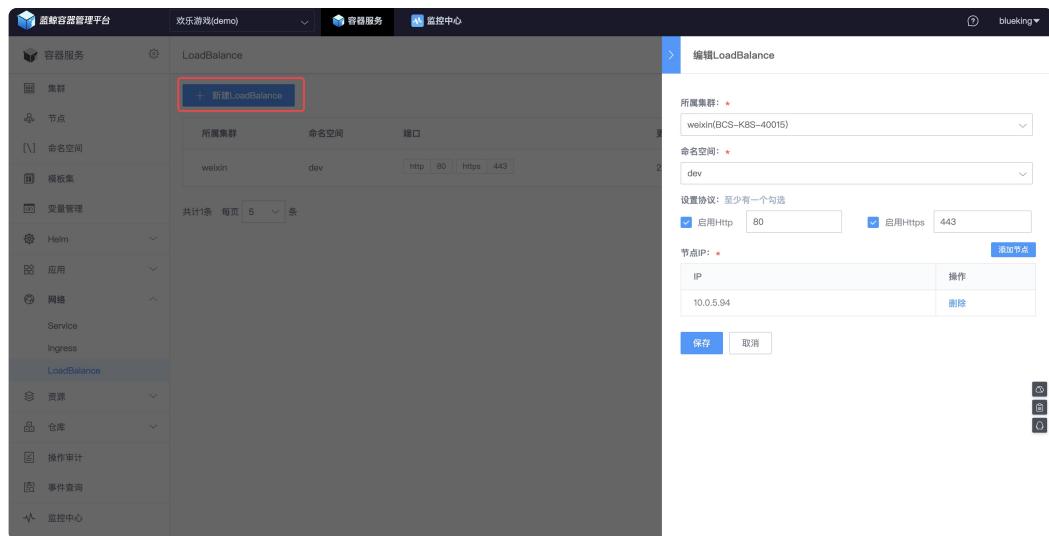
使用 K8S 资源准备版本

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**。

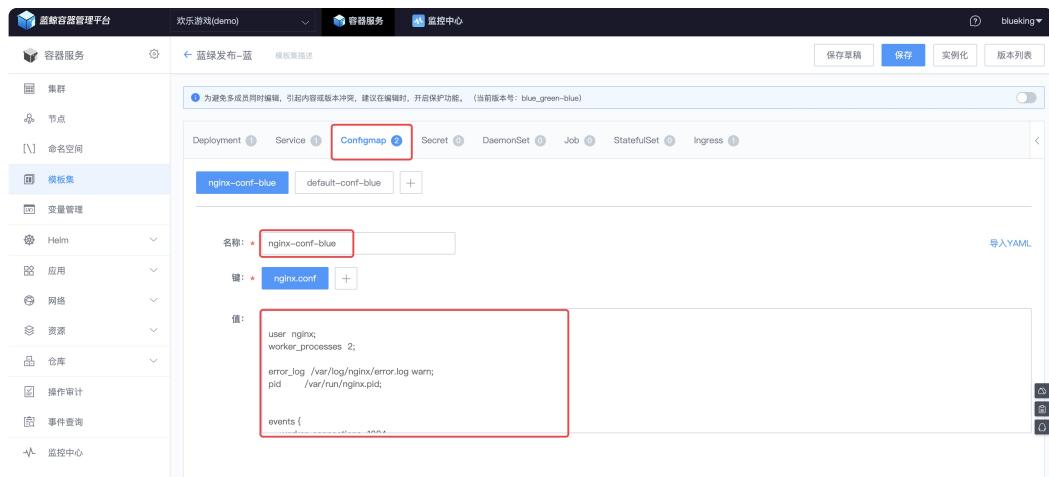
选择菜单【LoadBalancer】， 点击【新建 LoadBalancer】， 选择一个节点作为 LB 对外提供网络接入服务。



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. In the top navigation bar, the path is '容器服务' > '蓝绿发布-蓝' > '模板集描述'. The main panel is titled 'Deployment' with the name 'web-nginx-blue'. The '卷' (Volume) tab is selected, showing two 'configMap' entries under '类型' (Type). The first entry maps 'nginx.conf' to '/etc/nginx/nginx.conf' and 'default.conf' to '/etc/nginx/conf.d/default.conf'. A note at the bottom of this section says '只挂载一个文件, 防止挂载整个目录' (Mount only one file, prevent mounting the entire directory). Below this, the '挂载卷' (Mount Volume) tab for the 'nginx' service is shown, mapping 'container-port 80' to 'host-port 80' and selecting 'nginx-conf-blue' for the '/etc/nginx' directory.

- 创建 Service

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

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with various navigation options like '容器服务', '集群', '节点', '命名空间', '模板集' (which is 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 content area is titled '蓝绿发布-蓝' and '模板集描述'. It shows a 'Deployment' tab with a 'Service' sub-tab selected. A 'Service' dropdown menu is open, showing 'web-nginx-blue' as the selected item. Below it, there are fields for '名称' (Name) set to 'web-nginx-blue', '关联应用' (Associated Application) set to 'web-nginx-blue', and '关联标签' (Associated Labels) with 'app: web-nginx-blue' checked. Under 'Service类型' (Service Type), 'ClusterIP' is selected. There's also a 'ClusterIP' input field with placeholder '请输入 ClusterIP' and a note '不填或None'. The '端口映射' (Port Mapping) section shows 'http' mapped to port 80 with protocol 'TCP' and target port 'container~1'. The '标签管理' (Label Management) section shows 'app' mapped to 'web-nginx-blue'. A note at the bottom says 'ClusterIP为None时，端口映射可以不填；否则请先关联应用后，再填写端口映射' (If ClusterIP is None, port mapping can be omitted; otherwise, please associate the application first before filling in the port mapping). At the top right, there are buttons for '保存草稿', '保存' (Save), '实例化' (Instantiate), and '版本列表' (Version History).

● 新建 Ingress

在【Ingress】中填写【主机名】、【路径】以及绑定【Service】。

This screenshot shows the same interface as the previous one, but with the 'Ingress' tab now selected in the top navigation bar. The 'Service' dropdown still has 'web-nginx-blue' selected. The '基础信息' (Basic Information) section shows '主机名' (Host Header) set to 'blue.bk.tencent.com', 'Service 名' (Service Name) set to 'web-nginx-blue', and 'Service 端口' (Service Port) set to '80'. The '路径组' (Path Group) section shows a path '/> web-nginx-blue' with port '80'. The rest of the interface is identical to the previous screenshot.

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-green	2/2	08-16 16:40	滚动升级 扩缩容 更多
web-nginx-blue	Running	模板集	blue_blue-green-blue	2/2	08-16 16:20	滚动升级 扩缩容 更多
blueking-ingress-ingress-controller	Running	Heim模板	1.0.0	1/1	08-10 10:31	滚动升级 扩缩容 更多
blueking-ingress-ingress-default-backend	Running	Heim模板	1.0.0	1/1	08-10 10:31	滚动升级 扩缩容 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-x-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-x-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-ingress-ingress-default-backend	weixin	dev	Heim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	查看 删除
web-nginx-blue	weixin	dev	模板集	2019-08-16 16:20:51	2019-08-16 16:21:32	blueking	查看 删除

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。

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

```
$ 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

```
$ 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
bcs_nginx            1.17.0   9e1342027c80
              SIZE
```

- 推送镜像到仓库

```
$ docker tag bcs_nginx:1.17.0 <Registry_URL>/joyfulgame/bcs_nginx:1.17.0

$ docker push <Registry_URL>/joyfulgame/bcs_nginx:1.17.0
The push refers to repository [<Registry_URL>/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

容器服务 欢乐游戏(demo) 容器服务 监控中心 blueking

模板集

+ 添加模板集

输入关键字, 按Enter搜索

模板集名称	类型	容器 / 镜像	操作
蓝绿发布-绿	自定义	nginx / nginx:1.17.0	实例化 更多
蓝绿发布-蓝	自定义	最新版本: blue_green-blue	实例化 更多
web-nginx	自定义	最新版本: v12-apache-nginx	实例化 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	实例化 更多

全部数据加载完成

- 修改【Deployment】中的名称、标签以及镜像版本。

容器服务 欢乐游戏(demo) 容器服务 监控中心 blueking

模板集

BlueGreen Release - Green 编辑描述 保存草稿 保存 实例化 版本列表

Deployment 1 Service ConfigMap Secret DaemonSet Job StatefulSet Ingress

应用名称: * web-nginx-green 实例数量: * 2 导入YAML

重要级别: 一般 重要 不重要

描述: 请输入256个字符以内

标签: * app = web-nginx-green + 添加至选择器

容器名称: * nginx 类型: * Container InitContainer

描述: 请输入256个字符以内

镜像及版本: * bcs_nginx 1.17.0 总是在创建之前拉取镜像 WebConsole

- 修改【Service】中的名称、关联标签。

- 修改 Ingress，主机名不能上一个版本一样。

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。

模板集

+ 添加模板集

输入关键字, 按Enter搜索

模板集名称	类型	操作
蓝绿发布-绿	自定义	实例化 更多
蓝绿发布-蓝	自定义	实例化 更多 复制模板集 删除模板集
web-nginx	自定义	实例化 更多
示例模板集	自定义	实例化 更多

全部数据加载完成

- 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。

```
# cd charts/stable/wordpress/  
  
# ll  
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。

```
# 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。

```
# 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

```
# 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 查看 编辑 删除
	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 查看
	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 查看 WebConsole

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。

```
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初始值为创建时Chart中values.yaml内容，后续更新部署以该Yaml内容为准，Yaml内容最终通过--values选项传递给helm template命令  
250 ##  
251 ingress:  
252   ## Set to true to enable ingress record generation  
253   enabled: true  
254   ## Set this to true in order to add the corresponding annotations for cert-manager  
255   certManager: false  
256   ## Ingress annotations done as key:value pairs  
257   ## For a full list of possible ingress annotations, please see  
258   ## ref: https://github.com/kubernetes/ingress-nginx/blob/master/docs/annotations.md  
259   ## If this is set to true, annotation ingress.kubernetes.io/secure-backends: "true" will automatically be set  
260   ## If certManager is set to true, annotation kubernetes.io/tls-acme: "true" will automatically be set  
261   ## annotations:  
262   # kubernetes.io/ingress.class: nginx  
263   ## The list of hostnames to be covered with this ingress record.  
264   ## hosts:  
265   - name: wordpress.bk.tencent.com  
266   path: /  
267   ## The tls configuration for the ingress  
268   ##  
269   ## The tls configuration for the ingress  
270   ##  
271   ##  
272   ## The tls configuration for the ingress  
273   ## The tls configuration for the ingress  
274   ## The tls configuration for the ingress  
275   ## The tls configuration for the ingress  
276   ## The tls configuration for the ingress  
277   ## The tls configuration for the ingress  
278   ## The tls configuration for the ingress  
279   ## The tls configuration for the ingress  
280   ## The tls configuration for the ingress  
281   ## The tls configuration for the ingress  
282   ## The tls configuration for the ingress  
283   ## The tls configuration for the ingress  
284   ## The tls configuration for the ingress  
285   ## The tls configuration for the ingress  
286   ## The tls configuration for the ingress  
287   ## The tls configuration for the ingress  
288   ## The tls configuration for the ingress  
289   ## The tls configuration for the ingress  
290   ## The tls configuration for the ingress  
291   ## The tls configuration for the ingress  
292   ## The tls configuration for the ingress  
293   ## The tls configuration for the ingress  
294   ## The tls configuration for the ingress  
295   ## The tls configuration for the ingress  
296   ## The tls configuration for the ingress  
297   ## The tls configuration for the ingress  
298   ## The tls configuration for the ingress  
299   ## The tls configuration for the ingress  
300   ## The tls configuration for the ingress  
301   ## The tls configuration for the ingress  
302   ## The tls configuration for the ingress  
303   ## The tls configuration for the ingress  
304   ## The tls configuration for the ingress  
305   ## The tls configuration for the ingress  
306   ## The tls configuration for the ingress  
307   ## The tls configuration for the ingress  
308   ## The tls configuration for the ingress  
309   ## The tls configuration for the ingress  
310   ## The tls configuration for the ingress  
311   ## Enable client source IP preservation  
312   ## ref https://kubernetes.io/docs/tutorials/stateful-set/expose/#enable-client-source-ip-preservation  
313   ##  
314 externalTrafficPolicy: Cluster
```

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初始值为创建时Chart中values.yaml内容，后续更新部署以该Yaml内容为准，Yaml内容最终通过--values选项传递给helm template命令  
1 apiVersion: apps/v1beta1  
2 kind: StatefulSet  
3 metadata:  
4   annotations:  
5     io.tencent.paus.creator: blueking  
6     io.tencent.paus.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.controllerType: StatefulSet  
15    io.tencent.bcs.kind: Kubernetes  
16    io.tencent.bcs.monitor.level: general  
17    io.tencent.bcs.namespace: dev  
18    io.tencent.bkdata.baseall.dataid: '6566'  
19    io.tencent.bkdata.container.stderr.dataid: '0'  
20    io.tencent.bkdata.pod.procid: '79ab616a0a194ac2952b497e79d0b967'  
21    io.tencent.bkdata.source.type: helm  
22    io.tencent.paus.version: 7.3.0  
23    io.tencent.paus.version: 7.3.0  
24    io.tencent.bcs.cluster: BCS-K8S-40015  
25    release: wordpress-1909121112-mariadb  
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。

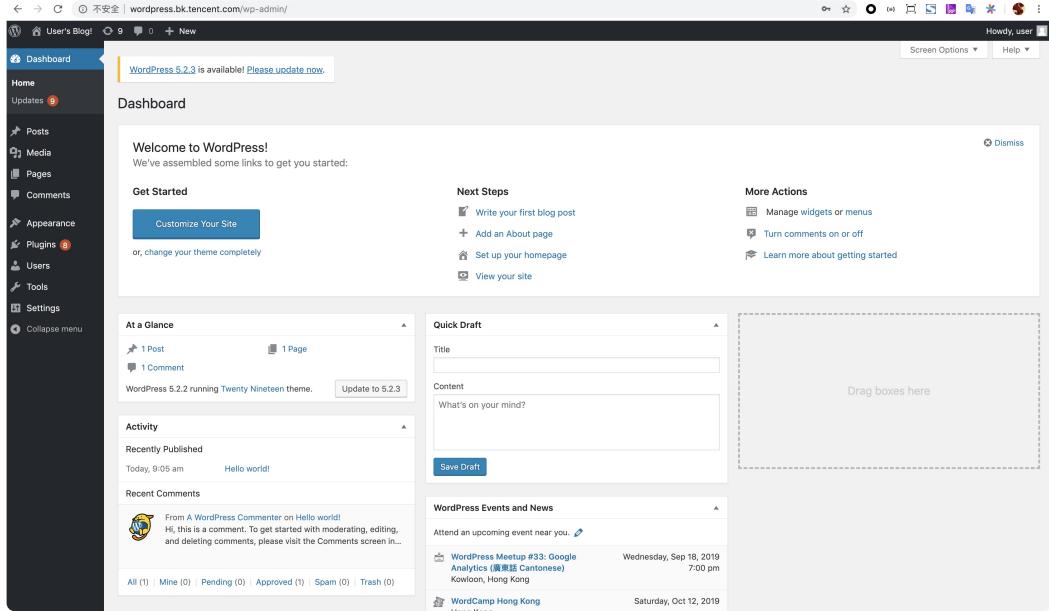
在【Release 列表】菜单中可以查看部署状态。

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。

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。

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

# echo "bFgzTmZvSHJIVg==" | base64 --decode
1X3NfoHrHV
```

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。

```
# cd charts/stable/gitlab-ce/  
  
# ll  
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。

```
# 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。

```
# 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

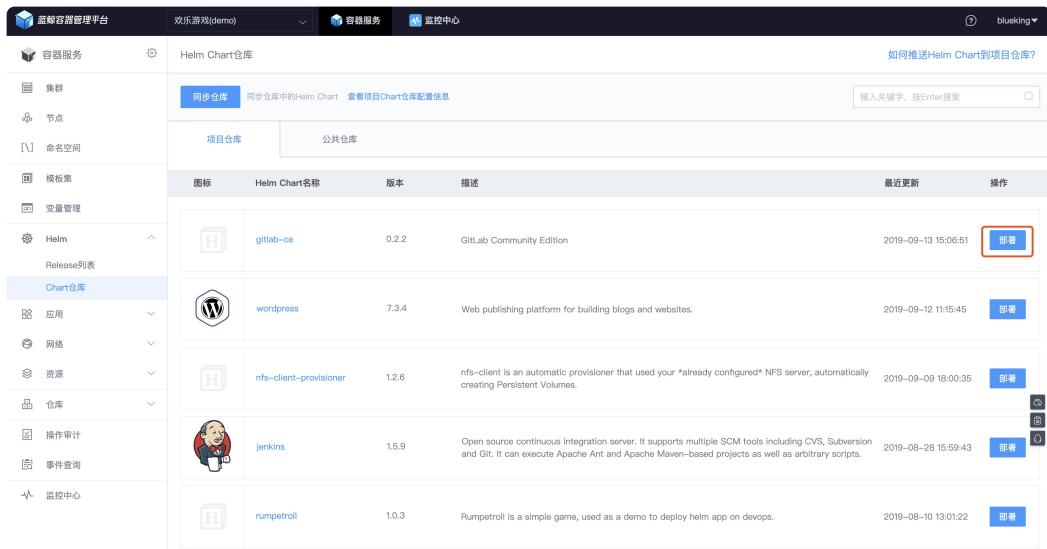
推送 Chart 到仓库

```
# 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】。



The screenshot shows the BCS Helm Chart Repository interface. On the left, there's a sidebar with various navigation options like Container Service, Cluster, Node, Namespace, Template Set, and Helm. Under Helm, there are sub-options for Release List, Chart Repository, Application, Network, Resource, and Library. The Chart Repository section is currently selected. In the main content area, there are two tabs: 'Project Library' (selected) and 'Public Library'. Below these tabs, there's a search bar with placeholder text '输入关键字, 按Enter搜索' and a 'Search' button. A help link '如何推送Helm Chart到项目仓库?' is also present. The main table lists several charts, each with columns for Icon, Helm Chart Name, Version, Description, Last Updated, and Action (Deployment). The 'gitlab-ce' chart is listed with version 0.2.2, description 'GitLab Community Edition', last updated on 2019-09-13 15:06:51, and a 'Deploy' button highlighted with a red border. Other charts listed include 'wordpress' (version 7.3.4), 'nfs-client-provisioner' (version 1.2.6), 'jenkins' (version 1.5.9), and 'rumpetroll' (version 1.0.3).

选择 Chart 版本以及命名空间后，下方显示 Helm 参数。

The screenshot shows the BlueKing Platform Helm Chart Management interface. On the left, there's a sidebar with categories like Application, Network, Resource, Library, Audit, and Monitoring. The main area is titled 'Helm参数' (Helm Parameters) and shows a code editor with a YAML file. The code is a Helm template for a GitLab chart, defining parameters such as image versions, external URLs, and service types. At the bottom of the code editor are buttons for '部署' (Deploy), '预览' (Preview), and '取消' (Cancel). To the right of the code editor is a 'WebConsole' button.

```

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 ## Specify a imagePullPolicy
6 ## Always if imageTag is 'latest', else set to 'IfNotPresent'
7 ## ref: http://kubernetes.io/docs/user-guide/images/#pre-pulling-images
8 ##
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-   tls:
35-     # - secretName: gitlab.cluster.local
36-     #   hosts:
37-       #     - gitlab.cluster.local
38-     url: gitlab.bk.tencent.com
39-   ## Configure external service metrics

```

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。

```
serviceType: ClusterIP
```

- Ingress

此处启用 Ingress，并填写 URL。

```

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。

```

1  apiVersion: extensions/v1beta1
2  kind: Deployment
3  metadata:
4    annotations:
5      io.tencent.paaas.creator: blueking
6      io.tencent.paaas.updater: blueking
7    labels:
8      app: gitlab-ce-1909130753-gitlab-ce
9      chart: gitlab-ce-0.2.2
10     io.tencent.bcs.app.oppid: '6'
11     io.tencent.bcs.clusterid: BCS-K8S-40015
12     io.tencent.bcs.controller.name: gitlab-ce-1909130753-gitlab-ce
13     io.tencent.bcs.controller.type: Deployment
14     io.tencent.bcs.kind: Kubernetes
15     io.tencent.bcs.monitor.level: general
16     io.tencent.bcs.namespace: game
17     io.tencent.bcs.projectid: '6566'
18     io.tencent.bkdata.bosself.datoid: '6566'
19     io.tencent.bkdata.container.stdlog.datoid: '0'
20     io.tencent.paaas.projectid: 794b6164a194ac2952b497e79a0b967
21     io.tencent.paaas.source_type: helm
22     io.tencent.paaas.upstream_id: '2'
23     io.tencent.paaas.cluster: BCS-K8S-40015
24     release: gitlab-ce-1909130753
25     name: gitlab-ce-1909130753-gitlab-ce
26   spec:
27     replicas: 1
28     template:
29       metadata:
30         labels:
31           app: gitlab-ce-1909130753-gitlab-ce
32           io.tencent.bcs.app.oppid: '6'
33           io.tencent.bcs.clusterid: BCS-K8S-40015
34           io.tencent.bcs.controller.name: gitlab-ce-1909130753-gitlab-ce
35           io.tencent.bcs.controller.type: Deployment
36           io.tencent.bcs.kind: Kubernetes
37           io.tencent.bcs.monitor.level: general
38           io.tencent.bcs.namespace: game
39           io.tencent.bkdata.bosself.datoid: '6566'
40           io.tencent.bkdata.container.stdlog.datoid: '0'
41           io.tencent.paaas.projectid: 794b6164a194ac2952b497e79a0b967
42           io.tencent.paaas.source_type: helm
43           io.tencent.paaas.upstream_id: '2'
44     httpPort: 80
45     httpsPort: 443
46     ## Configure external service ports
47     ## ref: http://kubernetes.io/docs/user-guide/services/
48     readinessPort: http
49     livenessPort: http
50     ## Configure resource requests and limits
51     ## ref: http://kubernetes.io/docs/user-guide/compute-resources/

```

点击【部署】即可。

在【Release 列表】菜单中可以查看部署状态。

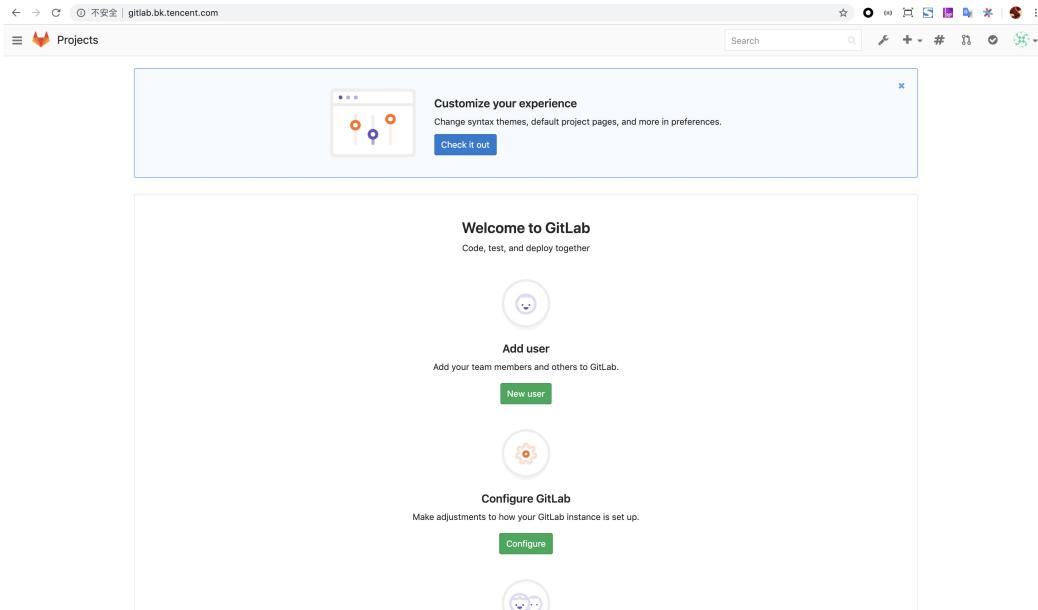
名称	类型	Pods
gitlab-ce-1909130302-postgresql	Deployment	1/1
gitlab-ce-1909130302-redis	Deployment	1/1
gitlab-ce-1909130302-gitlab-ce	Deployment	1/1
gitlab-ce-1909130302-postgresql	Service	-/-
gitlab-ce-1909130302-redis	Service	-/-
gitlab-ce-1909130302-gitlab-ce	Ingress	-/-
gitlab-ce-1909130302-gitlab-ce	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。

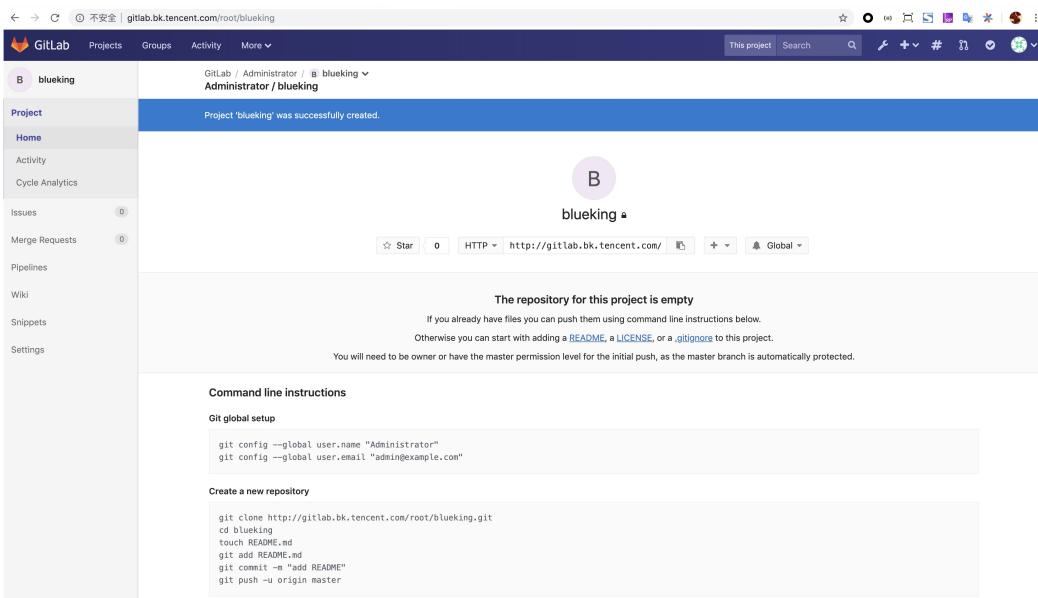
访问测试

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。



```
git config --global user.name "Administrator"  
git config --global user.email "admin@example.com"
```

Create a new repository

```
git clone http://gitlab.bk.tencent.com/root/blueking.git  
cd blueking  
touch README.md  
git add README.md  
git commit -m "add README"  
git push -u origin master
```

BCS 部署应用，如此简单。

Dashboard - Metric 数据查看和配置

容器的基础性能数据、容器的日志（标准输出日志采集、非标准输出日志采集）、容器内运行的应用程序的自定义 Metric 在采集，清洗完成，且在数据平台生成相应结果表后，即可在 Dashboard 中查询。

入口地址：【监控中心】->【Dashboard】

Dashboard 含默认 和 自定义 2 种。



注意：请勿修改默认 Dashboard， 默认视图下次升级会覆盖您修改的配置

点击 **New Dashboard** 可以添加自定义数据仪表盘，点击 **Import Dashboard** 可以导入自定义数据仪表盘。

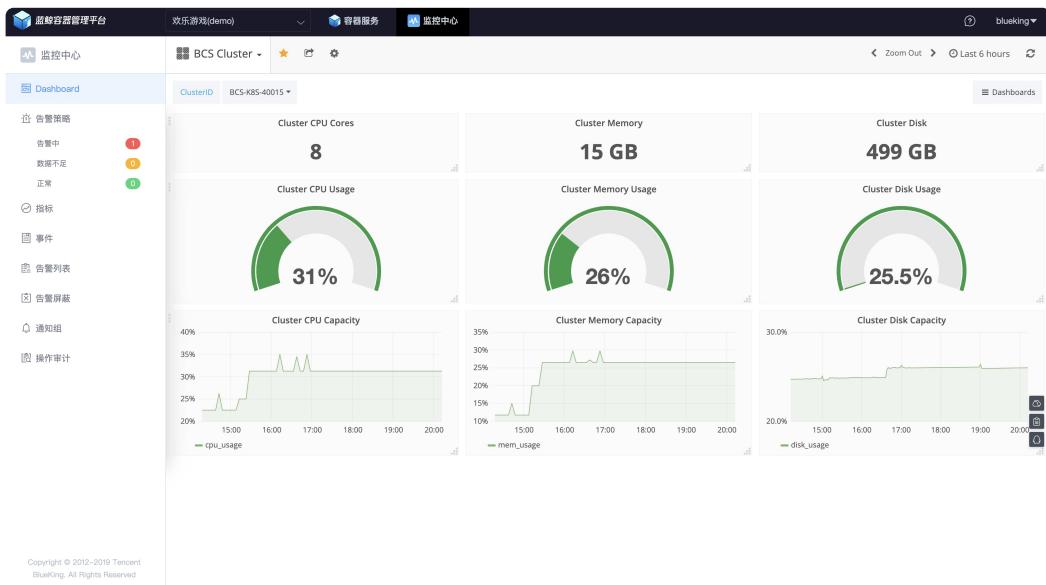
视图

默认 Dashboard 包含

- **BCS Cluster**, 集群视图
- **BCS Node**, 集群节点视图
- **BCS Pods**, 容器 Pod 视图

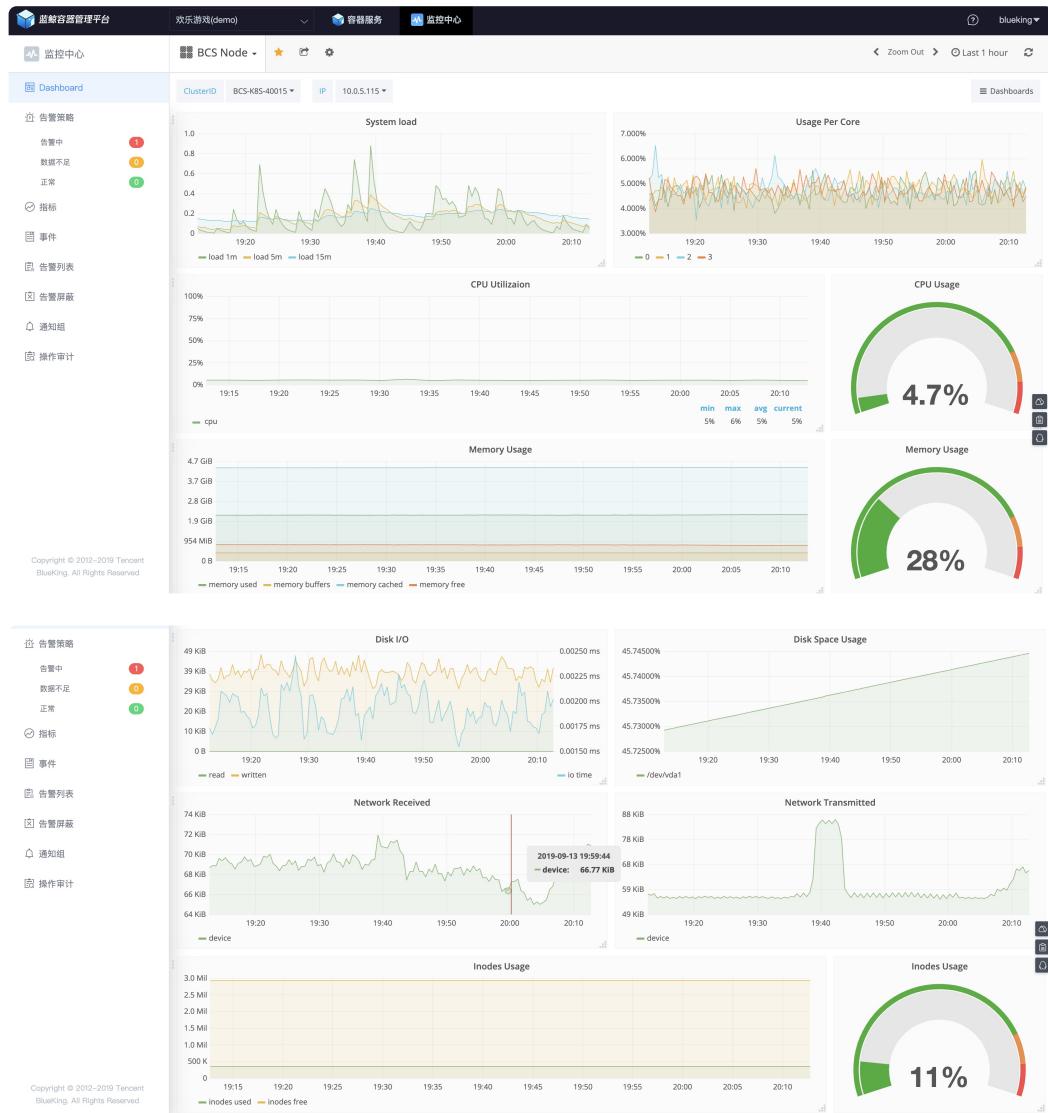
BCS Cluster

呈现集群的 CPU、内存资源的容量以及使用率。



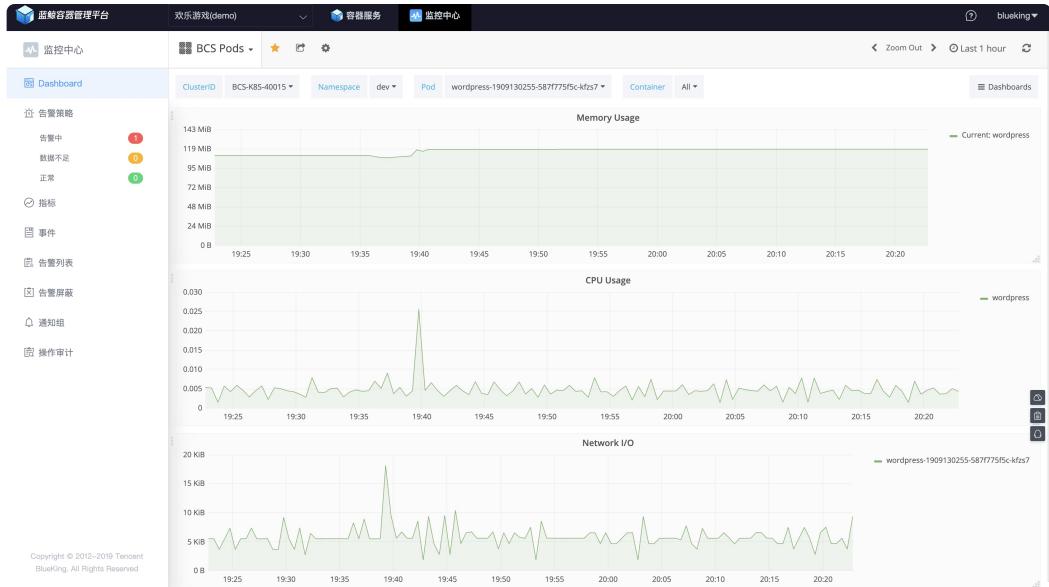
BCS Node

呈现节点的平均负载以及 CPU、内存使用率。



BCS Pods

呈现 Pod 的 CPU、内存、网络资源的使用情况。



告警策略配置

选择【告警策略】菜单，可以容器、主机等维度的告警策略，如下图：



如选择容器维度后，还可以继续选择集群、命名空间、Deployment、StatefulSet 等维度。

The screenshot shows the BlueKing Container Management Platform interface. On the left, there's a sidebar with navigation links like 监控中心, Dashboard, 告警策略 (selected), 事件, 告警列表, 告警屏蔽, 通知组, and 操作审计. The main content area is titled '选择指标 定义告警策略' (Select Metrics to Define Alert Strategy). It lists several alerting rule templates: 按集群 (7 metrics), 按命名空间 (7 metrics), 按Deployment (7 metrics), 按StatefulSet (7 metrics), 跨所有集群 (7 metrics), and 跨所有Pod (7 metrics). Each template has a brief description and a '指标数' (Metrics Count) field.

例如选择 集群 维度，接下来选择 CPU 使用率

This screenshot continues from the previous one, showing the '集群' (Cluster) dimension selected. The left sidebar remains the same. The main area is titled '选择指标 定义告警策略' and shows a list of metrics under '集群' for the 'weixin[BCS-K8S-40015]' service. The metrics listed are: CPU使用率 (CPU Usage Rate), 内存使用量 (Memory Usage), 磁盘读速率 (Disk Read Rate), 磁盘写速率 (Disk Write Rate), and 磁盘I/O时间 (Disk I/O Time). Below the metric list is a chart titled '[weixin[BCS-K8S-40015]] CPU使用率' (weixin[BCS-K8S-40015] CPU Usage Rate) with a note '提醒: 只展示了前5条和后5条数据' (Reminder: Only the first 5 and last 5 data points are displayed). The chart shows CPU usage fluctuating between 0% and 0.7% over a period from 09:22 to 10:22. At the bottom right, there are '取消' (Cancel) and '下一步' (Next Step) buttons.

设置对应的告警检测策略。

监控中心 欢乐游戏(demo) 容器服务 监控中心 blueking

告警策略

基本信息

名称为: 集群 CPU 使用率
描述为: 请输入

触发条件

每当: CPU使用率 算法是: 静态阈值 同比策略 环比策略
指标当前值 > 60 % 时触发报警

收敛设置

收敛规则: 收敛方式一 收敛方式二
自第 1 次告警后 5分钟 内不再发出报警

操作

通知: 每当此告警: 状态为“告警中” 通知方式: 微信
发送通知到: checked breaking (1) 新增通知组
附加通知人员:

Copyright © 2012-2019 Tencent

上图是生产环境设置的策略，为了测试告警也可以降低阈值，可以看到阈值在下图的右侧以红线呈现，以直观的视角辅助阈值设置。

监控中心 欢乐游戏(demo) 容器服务 监控中心 blueking

告警策略

基本信息

名称为: cpu
描述为: 请输入

触发条件

每当: CPU使用率 算法是: 静态阈值 同比策略 环比策略
指标当前值 > 0.4 % 时触发报警

收敛设置

收敛规则: 收敛方式一 收敛方式二
自第 1 次告警后 5分钟 内不再发出报警

操作

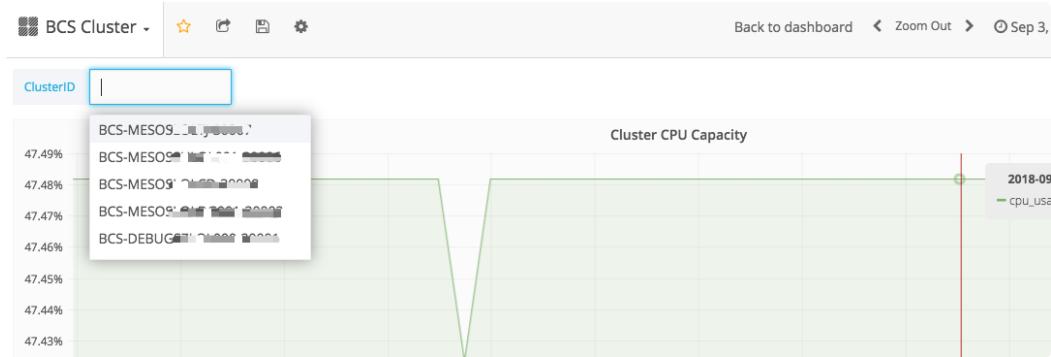
通知: 每当此告警: 状态为“告警中” 通知方式: 邮件
发送通知到: checked breaking (1) 新增通知组
附加通知人员:

【保存】后告警策略设置成功。

Dashboard - 模板(Templating)

模板允许更多交互式, 动态的仪表板。您可以在度量标准查询中使用变量代替硬编码

cluster_id, namespace 和 container_id 名称等内容。变量显示为仪表板顶部的下拉选择框。通过这些下拉菜单, 您可以轻松更改仪表板中显示的数据。



什么是变量

变量是值的占位符。您可以在度量标准查询和面板标题中使用变量。因此, 当您更改值时, 使用仪表板顶部的下拉列表, 面板的度量标准查询将更改以反映新值。

Interpolation

面板标题和度量标准查询可以使用两种不同的语法引用变量:

- `$<varname>` Example: `apps.frontend.$server.requests.count`
- `[[varname]]` Example: `apps.frontend.[$server].requests.count`

为什么两种方式? 第一种语法更易于读写, 但不允许在单词中间使用变量。在表达式中使用第二种

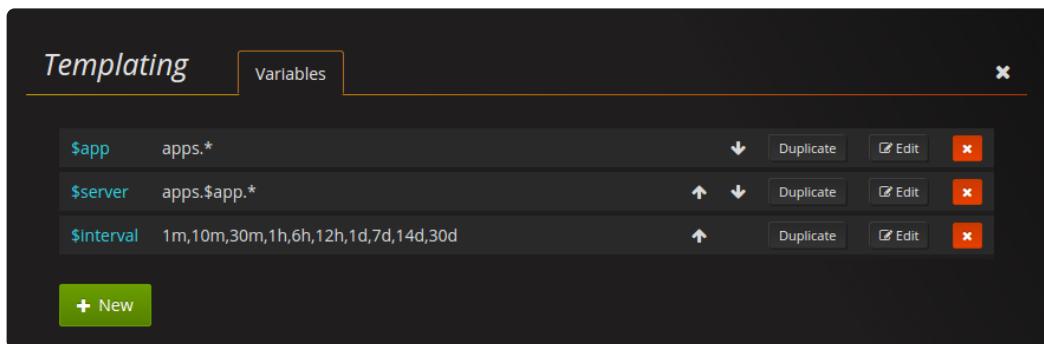
语法 `my.server[[serverNumber]].count` .

在将查询发送到数据源之前，将对查询进行插值，这意味着将变量替换为其当前值。在插值期间，可以对变量值进行转义，以符合查询语言的语法和使用它的位置。例如，InfluxDB 或 Prometheus 查询中的正则表达式中使用的变量将进行正则表达式转义。有关插值期间值转义的详细信息，请阅读数据源特定文档文章。

变量选项

变量显示为仪表板顶部的下拉选择框。它具有当前值和一组选项。该选项是一组可供选择的值。

添加变量



您可以通过 Dashboard > Templating 添加变量。这将打开一个变量列表和一个 New 用于创建新变量的按钮。

基本变量选项

选项	描述
<i>Name</i>	变量的名称，这是在度量标准查询中引用变量时使用的名称。必须是唯一的，不包含空格。
<i>Label</i>	此变量的下拉列表的名称。
<i>Hide</i>	隐藏下拉选择框的选项。
<i>Type</i>	定义变量类型。

变量类型

类型	描述
<i>Query</i>	此变量类型允许您编写数据源查询，该查询通常返回度量标准名称，标记值或键的列表。例如，返回服务器名称，传感器 ID 或数据中心列表的查询。
<i>Interval</i>	此变量可表示时间跨度。不是按时间或日期直方图间隔对组进行硬编码，而是使用此类型的变量。
<i>Datasource</i>	此类型允许您更改整个仪表板的数据源。如果您在不同的环境中具有多个数据源实例，则非常有用。
<i>Custom</i>	使用逗号分隔列表手动定义变量选项。
<i>Constant</i>	定义隐藏常量。对于要共享的仪表板的度量标准路径前缀很有用。在仪表板导出期间，常量变量将变为导入选项。

类型	描述
<i>Ad hoc filters</i>	非常特殊的变量，目前仅适用于某些数据源，InfluxDB 和 Elasticsearch。它允许您添加键/值过滤器，这些过滤器将自动添加到使用指定数据源的所有度量标准查询中。

查询选项

此变量类型是最强大和最复杂的，因为它可以使用数据源查询动态获取其选项。

选项	描述
<i>Data source</i>	查询的数据源目标。
<i>Refresh</i>	控制何时更新变量选项列表（下拉列表中的值）。在仪表板上加载将减慢仪表板负载，因为在初始化仪表板之前需要完成变量查询。如果变量选项查询包含时间范围过滤器或取决于仪表板时间范围，则仅将此设置为“开启时间范围更改”。
<i>Query</i>	数据源特定的查询表达式。
<i>Regex</i>	正则表达式用于过滤或捕获数据源查询返回的名称的特定部分。可选的。
<i>Sort</i>	在下拉列表中定义选项的排序顺序。已禁用表示将使用数据源查询返回的选项顺序。

查询表达式

每个数据源的查询表达式都不同。

- [Elasticsearch templating queries](#)
- [InfluxDB templating queries](#)

需要注意的一点是，查询表达式可以包含对其他变量的引用，实际上可以创建链接变量。Grafana 将检测到这一点并在其中一个变量包含变量时自动刷新变量。

选择选项

选项	描述
<i>Multi-value</i>	如果启用，该变量将支持同时选择多个选项。
<i>Include All option</i>	添加一个特殊 All 选项，其值包括所有选项。
<i>Custom all value</i>	默认情况下，该 All 值将包括组合表达式中的所有选项。这可能会变得很长并且可能会出现性能问题。很多时候，最好指定一个自定义的所有值，比如通配符正则表达式。为了能够在 Custom all value 选项中使用自定义正则表达式，glob 或 lucene 语法，它永远不会被转义，因此您必须考虑哪些是数据源的有效值。

形成多个值

使用所选择的多个值对变量进行插值是很棘手的，因为如何将多个值格式化为在使用该变量的给定上下文中有效的字符串。Grafana 试图通过允许每个数据源插件通知模板插值引擎用于多个值的格式来解决这个问题。

例如，Graphite 使用 glob 表达式。在这种情况下，具有多个值的变量将被内插，`{host1,host2,host3}` 就像当前变量值是 host1, host2 和 host3 一样。

InfluxDB 和 **Prometheus** 使用正则表达式，因此相同的变量将被插值为 (`host1|host2|host3`)。

如果没有，每个值也将是正则表达式转义，具有正则表达式控制字符的值将破坏正则表达式。

Elasticsearch 使用 lucene 查询语法，因此在这种情况下，相同的变量将被格式化为 (`"host1" OR "host2" OR "host3"`)。在这种情况下，每个值都需要进行转义，以便该值可以包含 lucene 控制字和引号。

值组/标签

如果您在多值变量的下拉列表中有很多选项。您可以使用此功能将值分组为可选标记。

选项	描述
<code>Tags query</code>	应返回标记列表的数据源查询
<code>Tag values query</code>	应返回指定标记键值的列表的数据源查询。 <code>\$tag</code> 在查询中使用以引用当前选定的标记。

The screenshot shows a user interface for managing tag values. At the top left is a search bar containing the text 'server'. Below it is a list titled 'Selected (4)' which contains five items: 'All', 'backend_01', 'backend_02', 'backend_03', and 'backend_04'. Each item has a checkbox next to it; 'All' and the four 'backend...' items have their checkboxes checked. To the right of this list is a 'Tags' section. Inside the 'Tags' section, there are two entries: 'backend' and 'fakesite'. Each entry consists of a small icon, the tag name, and a trash can icon.

区间变量

使用 `Interval` 类型创建表示时间跨度的变量（例如 `1m`，`1h`，`1d`）。还有一个特殊 `auto` 选项会根据当前时间范围而改变。您可以指定当前时间范围应分多少次以计算当前 `auto` 时间跨度。

此变量类型可用作按时间分组的参数（对于 InfluxDB），日期直方图间隔（对于 Elasticsearch）或作为汇总函数参数（对于 Graphite）。

在 graphite 函数中使用 `myinterval` 类型的模板变量的示例 `Interval`：

```
summarize($myinterval, sum, false)
```

全局内置变量

Grafana 具有全局内置变量，可以在查询编辑器中的表达式中使用。

`$__interval` 变量

这个`$__interval` 变量类似于上面描述的 `auto` interval 变量。它可以用作按时间分组的参数（对于 InfluxDB），日期直方图间隔（对于 Elasticsearch）或作为汇总函数参数（对于 Graphite）

Grafana 自动计算可用于在查询中按时间分组的间隔。当数据点多于图表上显示的数据点时，可以通过更大的间隔分组来提高查询效率。在查看 3 个月的数据时，分组比 1 天比 10 分更有效，图表看起来相同，查询会更快。的`$__interval` 使用时间范围和图形（像素数）的宽度来计算。

近似计算： `(from - to) / resolution`

例如，当时间范围为 1 小时且图形为全屏时，则可以计算间隔 `2m` - 以 2 分钟为间隔对点进行分组。如果时间范围是 6 个月并且图表是全屏，则间隔可能是 `1d`（1 天） - 点按天分组。

在 InfluxDB 数据源中，遗留变量 `$interval` 是同一个变量。`$__interval` 应该用来代替。

InfluxDB 和 Elasticsearch 数据源具有 Group by time interval 用于对间隔进行硬编码或设置 `$__interval` 变量的最小限制的字段（通过使用>语法 - > `>10m`）。

`$__interval_ms` 变量

此变量是以 `$__interval` 毫秒为单位的变量（而不是格式化字符串的时间间隔）。例如，如果 `$__interval` 是，`20m` 那么 `$__interval_ms` 是 `1200000`。

`$timeFilter` or `$__timeFilter` 变量

`$timeFilter` 变量返回当前选定的时间范围作为表达式。例如，时间范围间隔 `Last 7 days` 表达式为 `time > now() - 7d`。

这在 InfluxDB 数据源的 WHERE 子句中使用。在查询编辑器模式下，Grafana 会自动将其添加到 InfluxDB 查询中。必须在文本编辑器模式下手动添加：`WHERE $timeFilter`。

`$__name` 变量

此变量仅在 Singlestat 面板中可用，可以在“选项”选项卡上的前缀或后缀字段中使用。变量将替换为

系列名称或别名。

重复面板

模板变量对于在整个仪表板中动态更改查询非常有用。如果您希望 Grafana 根据您选择的值动态创建新面板或行，则可以使用“重复”功能。

如果您启用了变量 `Multi-value` 或 `Include all value` 选项，则可以选择一个面板或一行，并让 Grafana 为每个选定值重复该行。您可以在面板编辑模式的“常规”选项卡下找到此选项。选择要重复的变量，然后选择 `min span`。该 `min span` 控制小 Grafana 将如何使面板（如果您有很多值中选择）。Grafana 将自动调整每个重复面板的宽度，以便填满整行。目前，您不能将一行中的其他面板与重复面板混合使用。

仅对第一个面板（原始模板）进行更改。要使更改在所有面板上生效，您需要触发动态仪表板重建。您可以通过更改变量值（这是重复的基础）或重新加载仪表板来完成此操作。

重复行

此选项要求您打开行选项视图。将鼠标悬停在行左侧以触发行菜单，在此菜单中单击 `Row Options`。这将打开行选项视图。在这里，您可以找到重复下拉列表，您可以在其中选择要重复的变量。

URL 状态

变量值始终使用语法同步到 URL `var-<varname>=value`。

示例

- [Elasticsearch Templated Dashboard](#)
- [InfluxDB Templated Dashboard](#)

FAQ

产品使用

BCS 与 K8S、Mesos 的关系和区别是什么

K8S、Mesos 为容器编排引擎，BCS 是容器管理平台，兼容 K8S、Mesos 两种容器编排引擎，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

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：

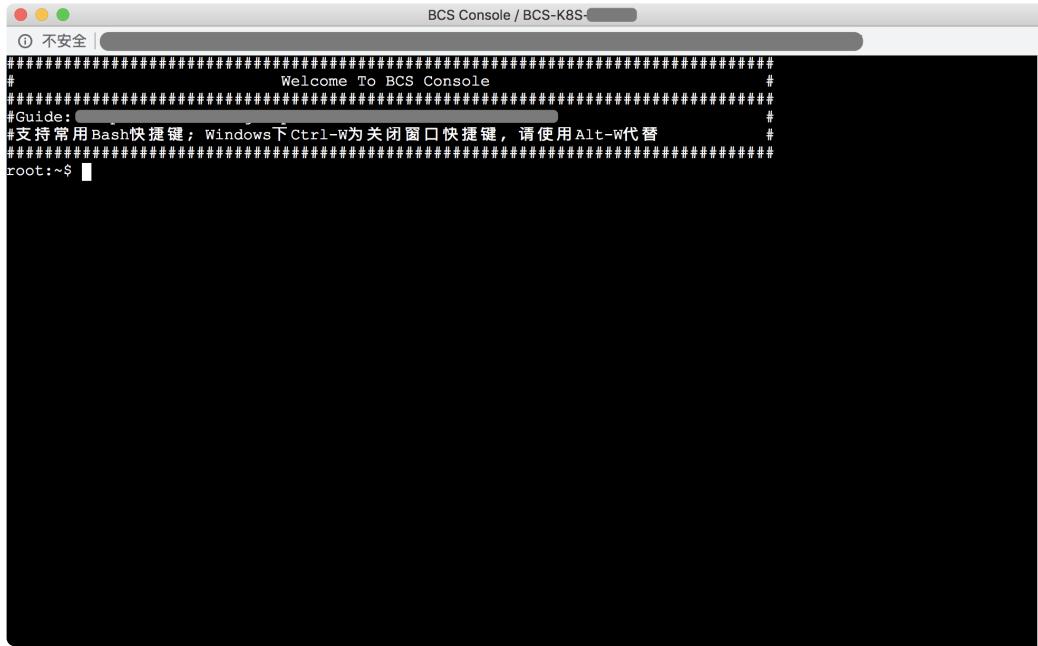
- 镜像不存在 针对这种情况，需要用户上传镜像即可。
- 节点没有权限拉取镜像 这种情况，一般是由于平台侧创建 `imagePullSecrets` 失败导致。

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 Webcosole 点击容器服务的右下角“WebConsole”，点击相应的集群，弹出 Webcosole 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

应用列表							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
	deploy-nginx1	Running	deployment	black	init_version	1/1	03-15 15:09	滚动升级 扩容 更多

- 查找 Pod / Taskgroup 管理项，查看到部署的节点

Pod管理		事件	标签	备注	状态:	Host IP:	Pod IP:	存活时间:	重新调度
	deploy-nginx1-744d596b4c-xi6ws				Running	black	black	4分20秒	

- 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

```
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
CPU限制:	请输入	m	⑦	请输入	m	⑦
内存限制:	请输入	Mi	⑦	请输入	Mi	⑦

镜像问题

请参考 查看本文上面的 [拉取镜像失败](#) 处理流程