

# KubeSphere 文档 Advanced v1.0

# Release Notes For 1.0.1

KubeSphere 高级版 (Advanced Edition 1.0.1) 已于 2019 年 1 月 28 日 正式发布。建议下载并安装最新的 1.0.1 版本，若您已经安装了 1.0.0 版本，请下载 1.0.1 版本的 Installer，支持一键升级 1.0.0 至 1.0.1。

## 下载高级版 1.0.1

请前往 KubeSphere 官网下载最新的 [KubeSphere Advanced Edition 1.0.1](#)。

## 安装指南

高级版 1.0.1 支持以下两种安装模式，安装前请参考 [安装说明](#)。

- [All-in-One](#): All-in-One 模式即单节点安装，支持一键安装，仅建议您用来测试或熟悉安装流程和了解 KubeSphere 高级版的功能特性。
- [Multi-Node](#): Multi-Node 即多节点集群安装，高级版支持 master 节点和 etcd 的高可用，支持在正式环境安装和使用。

## 升级指南

v1.0.1 对 v1.0.0 进行了改进和功能优化，并修复了已知的 Bug，支持一键升级 1.0.0 至 1.0.1，参考 [升级指南](#)。

## 高级版 1.0.1 更新详情

### Bug 修复

- 修复 workspaces-manager 角色的账户登录后无法看到企业空间列表的问题。
- 修复 git 类型代码仓库凭证传参问题。
- 修正不存在用户登录失败的提示信息。

- 修复已删除应用残留数据问题。
- 修复某些 Helm 应用的配置文件中的参数显示问题。
- 修复 Pod 横向自动弹性伸缩 (HPA) 针对不同权限用户的显示问题。
- 修复在某些场景下修改应用路由 Hosts 参数不生效的问题。
- 更正多处中英文页面文案。

## 功能优化

- 在更新有状态副本集的时候，可同步编辑其关联服务。
- 更新资源时，其详情页的子页面的内容会有保存提示。
- 规范创建以及编辑项目时的 CPU 和内存限额的单位。
- 允许用户为没有 CPU 和内存限额的项目添加限额。
- 校验 DevOps 工程下凭证信息的完整性。
- 支持用户级别设置界面语言。
- 允许将 NodePort 类型的服务修改为 None 类型。
- 允许登录用户在用户设置页面修改密码。
- 多处用户体验优化，包括容器编辑界面输入参数、更新策略、容器日志等。

## Kubernetes 相关

- Kubernetes v1.12.3 可升级至 [Kubernetes 1.12.5](#)，同时支持升级至 v1.13.2。
- etcd 支持升级至 v3.2.24。

## 安全

- 登录方式支持验证码，防止登录攻击。
- 增加资源删除前的确认提示。

# Release Notes For 1.0.0

KubeSphere 高级版 (Advanced Edition 1.0.0) 已于 2018 年 12 月 10 日 正式发布。

## 关于 KubeSphere 高级版

KubeSphere 共提供社区版、易捷版以及高级版三个版本，其中社区版和易捷版主要提供多租户管理、集群运维、应用管理等功能，而高级版主要针对企业用户，目的是满足企业内不同角色用户、提供多种以容器为资源载体的业务功能模块。

## 下载高级版 1.0.0

请前往 KubeSphere 官网下载最新的 [KubeSphere Advanced Edition 1.0.0](#)。

## 安装

请参考 [安装指南](#)。

## 功能介绍

- 基于 [Kubernetes 1.12.3](#)
- 提供企业空间、项目、DevOps 工程多层次租户管理，支持预置以及自定义角色
- 支持 LDAP 统一认证
- 支持部署 (Deployments)、有状态副本集 (StatefulSets)、守护进程集 (DaemonSets)、任务 (Jobs)、定时任务 (CronJob) 多种工作负载
- 支持 Pod 水平自动弹性伸缩 (HPA, Horizontal Pod Autoscaler)
- 支持配额管理
- 支持密钥 (Secrets)、配置 (ConfigMaps) 管理
- 支持镜像仓库管理

- 支持服务和应用路由
- 支持 Calico、Flannel 开源网络插件
- 支持 GlusterFS、Ceph RBD 和 NFS 开源存储插件
- 基于青云平台上的主机部署，可以使用 [青云块存储](#)、[SDN 直通网络](#) 以及 [负载均衡器](#)
- 支持 [QingStor NeonSAN](#) 存储
- 提供基于 Jenkins 的流水线可视化编辑，预置多种功能属性的任务，支持 Jenkinsfile in & out SCM 两种模式
- 代码仓库支持 github/git/svn
- DevOps 提供统一凭证管理功能
- 可选装 Harbor 镜像仓库和 GitLab 代码仓库，并与 Jenkins 集成
- 基于 Prometheus 提供多维度监控服务，涵盖集群、主机、企业空间、项目、工作负载、Pod、容器等多个层面，以实时、历史、排行等多种展现方式展现 CPU、内存、网络、存储、IO 等多项监控指标，并提供 API 接口以便和已有监控系统集成
- 基于 [OpenPitrix](#)，提供应用仓库管理功能
- 最小单节点安装，可部署于现有 Kubernetes 集群。支持物理机、虚拟机和云主机等多种基础设施，支持混合部署方式
- 支持控制节点、etcd、监控高可用

## 产品简介

[KubeSphere](#) 是在目前主流容器调度平台 [Kubernetes](#) 之上构建的企业级分布式多租户容器管理平台，提供简单易用的操作界面以及向导式操作方式，在降低用户使用容器调度平台学习成本的同时，极大减轻开发、测试、运维的日常工作的复杂度，旨在解决 Kubernetes 本身存在的存储、网络、安全和易用性等痛点。除此之外，平台已经整合并优化了多个适用于容器场景的功能模块，以帮助企业轻松应对多租户、工作负载和集群管理、服务与网络管理、应用管理、镜像仓库管理和存储管理等业务场景。

相比较易捷版，KubeSphere 高级版提供企业级容器应用管理服务，支持更强大的功能和灵活的配置，满足企业复杂的业务需求。比如支持 Master 和 etcd 节点高可用、可视化 CI/CD 流水线、多维度监控、多租户管理、LDAP 集成、新增支持 HPA（水平自动伸缩）、容器健康检查以及 Secrets、ConfigMaps 的配置管理等功能，后续还将支持微服务治理、大数据、人工智能等更为复杂的业务场景。

KubeSphere 从项目初始阶段就采用开源的方法来进行项目的良性发展，项目相关的源代码和文档都在 GitHub 可见，可访问 [KubeSphere](#)。

## 功能架构

KubeSphere 高级版提供了在生产环境集群部署的全栈化容器部署与管理平台，它的核心功能可以概括在以下的功能架构图中，了解高级版的具体功能说明，可以在 [产品功能](#) 进行查看，或访问 KubeSphere 的 [商业网站](#)。

**企业空间**

**用户管理**

- 企业成员
- 用户组
- 项目
- DevOps 工程

**平台管理**

- 基础设施
- 服务组件
- 监控中心
- 应用仓库

**项目资源**

- 工作负载
- 部署(Deployment)
- 有状态副本集(Statefulset)
- 守护进程集(Deamonset)
- 任务(Job)
- 定时任务(CronJob)

**持久化存储卷**

- 存储卷(PVC)
- 存储类型
- NeonSAN
- NFS
- GlusterFS
- 青云块存储
- Ceph RBD
- Local

**服务与网络**

- 服务(Service)
- 应用路由(Ingress)

**项目设置**

- 成员/角色管理
- 外网访问(Gateway)
- 配额管理(Quota)

**多维度监控 用户视角**

**集群**

内存使用量	CPU 利用率(%)	CPU 使用量	项目资源用量	CPU 使用量	CPU 使用量
磁盘使用量	内存利用率(%)	内存使用量	容器组数量变化	内存使用量	内存使用量
磁盘吞吐	磁盘利用率(%)	项目变化趋势		网卡入口流量	网卡入口流量
IOPS	容器组数量变化	网络速率		网卡出口流量	网卡出口流量
容器组运行状态	服务组件状态				
CPU 使用量	网卡流量				

**多维度监控 管理员视角**

**集群**

内存使用量	CPU 利用率(%)	内存使用量	CPU 利用率(%)	CPU 使用量	CPU 使用量
磁盘使用量	内存利用率(%)	内存利用率(%)	CPU 使用量	内存使用量	内存使用量
磁盘吞吐	磁盘利用率(%)	磁盘吞吐	网卡流量	网卡入口流量	
IOPS	容器组数量变化	IOPS	容器组使用情况	网卡出口流量	
容器组运行状态	服务组件状态	容器组数量变化	inode 使用率(%)		
CPU 使用量	网卡流量	CPU 平均负载			
CPU 平均负载	inode 使用率(%)				

**DevOps 工程资源**

- CI/CD 流水线
- 可视化流水线构建
- Jenkinsfile in SCM
- Jenkinsfile out of SCM
- SVN
- Git
- GitHub

**DevOps 工程管理**

- 工程信息设置
- 凭证管理(Credentials)
- 成员/角色管理

# 产品规划

Community Edition ( 社区版 ) => Express Edition ( 易捷版 ) => Advanced Edition ( 高级版 )



# 产品功能

KubeSphere 为用户提供了一个具备极致体验的 Web 控制台，让您能够像使用任何其他互联网产品一样，快速上手各项功能与服务。KubeSphere 目前集成了工作负载管理、DevOps 工程、多租户管理、多维度监控、服务与网络、应用管理、基础设施管理、镜像管理、应用配置密钥管理这九大功能模块，并支持对接多种开源的存储服务和高性能的商业存储。以下从专业的角度为您详解各个模块的功能服务：

功能	说明
工作负载管理	对 Kubernetes 中的多种 workload 提供向导式管理界面，包括 Deployments, Daemon Sets, Stateful Sets, Jobs, CronJobs 等，并提供弹性伸缩 (HPA) 和容器健康检查支持。
DevOps 工程	提供可视化 CI/CD 流水线，支持从仓库 (GitHub/SVN/Git)、代码编译、镜像制作、镜像安全、推送到仓库、应用版本、到定时构建的端到端流水线设置，支持使用者在开发、测试等环境下的端到端高效流水线能力，同时提供完整的日志功能，记录 CI/CD 流水线的每个过程。
多租户管理	提供基于多租户、细粒度安全架构设计，提供资源以及操作级别的权限管控，充分保障资源安全性，同时支持标准 AD/LDAP 协议，以实现集中化认证。平台服务间进行加密通信，提供操作审计日志，可对宿主机以及容器镜像进行安全扫描并发现漏洞。
多维度监控	内置的监控中心提供多维度监控，从集群、企业空间、项目、工作负载/Pod、容器等多个层级，提供节点状态、CPU 和内存使用情况，存储、磁盘吞吐、网络出入和网卡流量以及 IOPS 等实时监控和历史数据。也可对接企业自有监控告警系统，提供对主机、容器以及应用服务多维度监控，内置监控服务支持高可用，下一个版本将集成告警服务。
服务与网络管理	基于原生 API，对 k8s 中的服务 (Service)、应用路由 (ingress) 等功能提供向导式管理界面，快速将应用暴露以供用户访问。高级版将集成 istio 中的 微服务治理、熔断、灰度发布、限流、智能路由等功能提供向导式管理界面。 如果部署在青云平台之上，可以使用插件对接青云的负载均衡器。
应用管理	后端使用开源的 OpenPitrix 应用商店和仓库服务，为用户提供应用全生命周期管理功能，包括：应用仓库管理、应用拓扑图、APM、应用变更和发布、应用上下线审批、版本控制、鲁棒性测试等。
基础设施管理	提供存储类型、主机、集群以及配额管理。存储既支持主流开源存储解决方案，也可对接青云的块存储和 NeonSAN。可批量添加主机，且对主机平台及系统弱依赖。并且支持镜像仓库管理、镜像复制、权限管理、镜像安全扫描。

功能	说明
镜像管理	提供镜像仓库管理，支持添加 Docker 或私有的 Harbor 镜像仓库。
应用配置及秘钥	提供自定义应用配置组集中管理，支持配置文件历史查询功能；对不同应用版本可提供不同的应用配置，如：环境变量、端口管理、资源配置、健康检查、应用数据、服务配置等

# 产品优势

## 设计愿景

众所周知，开源项目 Kubernetes 已经成为事实上的编排平台的领导者，是下一代分布式架构的王者，其在自动化部署、扩展性、以及管理容器化的应用已经体现出独特的优势。然而，很多人学习 Kubernetes，就会发现有点不知所措，因为 Kubernetes 本身有许多组件并且还有一些组件需要自行安装和部署，比如存储和网络部分，目前 Kubernetes 仅提供的是开源的解决方案或项目，可能在某种程度上难以安装，维护和操作，对于用户而言，学习成本和门槛都很高，快速上手并不是一件易事。

如果无论如何都得将应用部署在云上运行，为什么不让 KubeSphere 为您运行 Kubernetes 且更好地管理运行的资源呢？这样您就可以继续运行应用程序和工作负载并专注于这些更重要的业务。因为通过 KubeSphere 来快速创建 Kubernetes 集群、部署应用、添加服务、CI/CD、集群扩容、微服务治理、日志记录和资源监控，以及利用 KubeSphere 的其他诸多强大功能是多么容易。换句话说，Kubernetes 是一个很棒的开源项目（或被认为是一个框架），但是 KubeSphere 是一款非常专业的企业级平台产品，专注于解决用户在复杂业务场景中的痛点，提供更好更专业的用户体验。

最重要的是，KubeSphere 在存储和网络方面提供了最优的解决方案，比如存储除了支持流行的开源共享存储如 Ceph RBD 和 GlusterFS 之外，还提供 [QingCloud 云平台块存储](#) 和青云自研的 [分布式存储 QingStor NeonSAN](#) 作为 Kubernetes 的持久化存储，通过集成的 QingCloud CSI 和 NeonSAN CSI 插件，即可使用青云提供的高性能块存储或 NeonSAN 作为存储卷挂载至工作负载，为企业应用和数据提供更稳定安全的存储。

## 为什么选择 KubeSphere？

KubeSphere 为企业用户提供高性能可伸缩的容器应用管理服务，旨在帮助企业完成新一代互联网技术驱动下的数字化转型，加速业务的快速迭代与交付，以满足企业日新月异的业务需求。

优势	说明
灵活便捷的存储配置方案	商业化存储服务，将青云云平台块存储和 NeonSAN 块存储与 Kubernetes 相对 接，提供自主可控的存储服务 对接开源存储系统，如 Ceph RBD, Glusterfs, NFS 为用户提供丰富的功能：存储卷管理，存储类型管理 将 Kubernetes 存储理念和丰富的存储功能融入 UI，易于使用和管理存储资源

优势	说明
弹性伸缩	支持部署后集群节点扩容以及 Pod 动态的横向伸缩，保证集群和资源的高可用和可靠性。
多维度监控	<p>与云原生监控领域事实上的标准 Prometheus 全面深度集成      为 Kubernetes 提供全方位、多维度、细粒度、灵活、易用的监控 API      Cluster/Node/Workspace/Namespace/Workload/Pod/Container 多维度监控全覆盖并支持逐级下钻      丰富的监控指标、简洁美观的 UI 展现，提供即时值、趋势、排行等多种展现方式      通过监控 API 将 Kubenetus 监控与现有监控系统集成</p>
多租户管理	<p>提供多层次的权限控制体系，可分别针对集群、租户、项目多个层级进行授权      细粒度的权限控制，基于 RBAC 权限管理策略，可以灵活的制定角色      组织机构管理，支持组织机构多层级用户、组管理，针对用户组的权限管理      支持 LDAP、AD 的账户同步，快速接入企业现有的账户系统      统一认证，基于 OAuth2.0 提供统一的认证入口</p>
安全第一位	<p>API 安全，API Gateway 和认证鉴权体系保证 API 的安全      网络安全，基于 Kubernetes NetworkPolicy 实现内部网络隔离      操作系统安全，基于 Kubernetes Namespace 实现计算资源隔离</p>
内置 DevOps 工程	<p>开箱即用的 DevOps 功能，无需对 Jenkins 进行复杂的插件配置      独立 DevOps 工程，提供访问可控、安全隔离的 CI/CD 操作空间      兼容 Jenkinsfile in &amp; out of SCM (Source Code Management) 两种模式      可视化流水线编辑工具，降低 CI/CD 学习成本      使用 KubeSphere 提供弹性、干净、可定制的构建环境</p>
极简体验，向导 UI	面向开发、测试、运维友好的 UI，向导式用户体验，降低 Kubernetes 学习成本的设计理念。
松耦合功能模块设计	除提供基于原生 k8s 的管理功能之外，用户可以使用 KubeSphere 集成的诸如镜像仓库、应用仓库、监控、日志模块，也可通过配置的方式集成自建的相关服务。
可选的商业网络解决方案	除了支持 Calico、Flannel 等开源解决方案外，如用户对网络有更高要求，可选用青云作为底层平台，可以使用性价比更高的网络解决方案如青云 SDN。

## 名词解释

了解和使用 KubeSphere 管理平台，会涉及到以下的基本概念：

KubeSphere	Kubernetes 对照释义
项目	Namespace，为 Kubernetes 集群提供虚拟的隔离作用，详见 <a href="#">Namespace</a> 。
容器组	Pod，是 Kubernetes 进行资源调度的最小单位，每个 Pod 中运行着一个或多个密切相关的业务容器
部署	Deployments，表示用户对 Kubernetes 集群的一次更新操作，详见 <a href="#">Deployment</a> 。
有状态副本集	StatefulSets，用来管理有状态应用，可以保证部署和 scale 的顺序，详见 <a href="#">StatefulSet</a> 。
守护进程集	DaemonSets，保证在每个 Node 上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用，详见 <a href="#">Daemonset</a> 。
任务	Jobs，在 Kubernetes 中用来控制批处理型任务的资源对象，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。任务管理的 Pod 根据用户的设置将任务成功完成就自动退出了。比如在创建工作负载前，执行任务，将镜像上传至镜像仓库。详见 <a href="#">Job</a> 。
定时任务	CronJob，是基于时间的 Job，就类似于 Linux 系统的 crontab，在指定的时间周期运行指定的 Job，在给定时间点只运行一次或周期性地运行。详见 <a href="#">CronJob</a>
服务	Service，一个 KubeSphere 服务是一个最小的对象，类似 Pod，和其它的终端对象一样，详见 <a href="#">Service</a> 。
应用路由	Ingress，是授权入站连接到达集群服务的规则集合。可通过 Ingress 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，详见 <a href="#">Ingress</a> 。
镜像仓库	Image Registries，镜像仓库用于存放 Docker 镜像，Docker 镜像用于部署容器服务，详见 <a href="#">Images</a> 。
存储卷	PersistentVolumeClaim (PVC)，满足用户对于持久化存储的需求，用户将 Pod 内需要持久化的数据挂载至存储卷，实现删除 Pod 后，数据仍保留在存储卷内。Kubesphere 推荐使用动态分配存储，当集群管理员配置存储类型后，集群用户可一键式分配和回收存储卷，无需关心存储底层细节。详见 <a href="#">Volume</a> 。
存储类型	StorageClass，为管理员提供了描述存储“Class (类) ”的方法，包含 Provisioner、ReclaimPolicy 和 Parameters 。详见 <a href="#">StorageClass</a> 。

KubeSphere	Kubernetes 对照释义
流水线	Pipeline，简单来说就是一套运行在 Jenkins 上的 CI/CD 工作流框架，将原来独立运行于单个或者多个节点的任务连接起来，实现单个任务难以完成的复杂流程编排和可视化的工作。
企业空间	Workspace，是 KubeSphere 实现多租户模式的基础，是您管理项目、DevOps 工程和企业成员的基本单位。
主机	Node，Kubernetes 集群中的计算能力由 Node 提供，Kubernetes 集群中的 Node 是所有 Pod 运行所在的工作主机，可以是物理机也可以是虚拟机。详见 <a href="#">Nodes</a> 。

# 安装说明

[KubeSphere](#) 是在目前主流容器调度平台 [Kubernetes](#) 之上构建的 [企业级分布式多租户容器管理平台](#)，为用户提供简单易用的操作界面以及向导式操作方式，KubeSphere 提供了在生产环境集群部署的全栈化容器部署与管理平台，以及细粒度的资源监控和 CI/CD 流水线等。

## 前提条件

目前高级版已发布了 v1.0.0 和 v1.0.1，建议下载最新的 [KubeSphere Advanced-v1.0.1](#) 至待安装机器中。

## 安装 KubeSphere

KubeSphere 安装支持 [all-in-one](#) 和 [multi-node](#) 两种模式，即支持单节点和多节点安装两种安装方式。KubeSphere Installer 采用 [Ansible](#) 对安装目标机器及安装流程进行集中化管理配置。采用预配置模板，可以在安装前通过对相关配置文件进行自定义实现对安装过程的预配置，以适应不同的 IT 环境，帮助您快速安装 KubeSphere。

另外，KubeSphere Installer 集成了 [Harbor](#) 的 Helm Chart，但默认情况下不会安装 Harbor 镜像仓库，因为内置的 [Harbor](#) 作为可选安装项，用户可以根据团队项目的需求来配置安装，仅需安装前在配置文件 `conf/vars.yml` 中简单配置即可，参考 [安装内置 Harbor](#)。

### 说明:

- 由于安装过程中需要更新操作系统和从镜像仓库拉取镜像，因此必须能够访问外网。
- KubeSphere 集群的架构中，由于各自服务的不同，分为管理节点和工作节点两个角色，即 Master 和 Node。
- Master 节点由三个紧密协作的组件组合而成，即负责 API 服务的 `kube-apiserver`、负责调度的 `kube-scheduler`、负责容器编排的 `kube-controller-manager`。
- 集群的持久化数据，由 `kube-apiserver` 处理后保存至 etcd 中。
- 当进行 all-in-one 模式进行单节点安装时，这个节点既是管理节点，也是工作节点。
- 当进行 multi-node 模式安装多节点集群时，可在配置文件中设置集群各节点的角色。
- 如果是新安装的系统，在 Software Selection 界面需要把 OpenSSH Server 选上。

## All-in-One 模式

All-in-One 模式即单节点安装，支持一键安装，仅建议您用来测试或熟悉安装流程和了解 KubeSphere 高级版的功能特性，详见 [All-in-One 模式](#)。在正式使用环境建议使用 Multi-Node 模式。

## Multi-Node 模式

Multi-Node 即多节点集群安装，高级版支持 master 节点和 etcd 的高可用，支持在正式环境安装和使用，详见 [Multi-Node 模式](#)。

## 离线安装

KubeSphere 支持离线安装，若机器无法访问外网，请下载离线安装包进行安装。

离线的安装步骤与在线安装一致，因此可参考以上两种安装模式的安装指南进行安装。目前离线安装支持的操作系统如下，系统盘需保证 100 G 以上，主机配置规格的其它参数可参考在线安装的主机配置。

- CentOS 7.4/7.5
- Ubuntu 16.04.4/16.04.5

## 存储配置说明

Multi-Node 模式安装 KubeSphere 可选择配置部署 NFS Server 来提供持久化存储服务，方便初次安装但没有准备存储服务端的场景下进行部署测试。若在正式环境使用需配置 KubeSphere 支持的持久化存储服务，并准备相应的存储服务端。本文档说明安装过程中如何在 Installer 中配置 [QingCloud 云平台块存储](#)、[企业级分布式存储 NeonSAN](#)、[NFS](#)、[GlusterFS](#)、[Ceph RBD](#) 这类持久化存储的安装参数，详见 [存储配置说明](#)。

## 安装内置 Harbor (可选)

KubeSphere Installer 集成了 Harbor 的 Helm Chart (版本为 harbor-18.11.1)，内置的 Harbor 作为可选安装项，用户可以根据团队项目的需求来配置安装，详见 [安装内置 Harbor](#)。

## Master 和 etcd 节点高可用配置

Multi–Node 模式安装 KubeSphere 可以帮助用户顺利地部署环境，由于在实际的生产环境我们还需要考虑 master 节点的高可用问题，本文档以配置负载均衡器 (Load Balancer) 为例，引导您在安装过程中如何配置高可用的 Master 和 etcd 节点，详见 [Master 和 etcd 节点高可用配置](#)。

## 升级

若您的机器已安装的环境为 v1.0.0 版本，我们强烈建议您升级至最新的版本 v1.0.1，最新的 Installer 支持将 KubeSphere 从 v1.0.0 环境一键升级至目前最新的 v1.0.1，详见 [升级](#)。

## 集群节点扩容

安装 KubeSphere 后，在正式环境使用时可能会遇到服务器容量不足的情况，这时就需要添加新的节点 (node)，然后将应用系统进行水平扩展来完成对系统的扩容，配置详见 [集群节点扩容](#)。

## 高危操作

KubeSphere 支持管理节点和 etcd 节点高可用，保证集群稳定性，同时基于 kubernetes 底层调度机制，可以保证容器服务的可持续性及稳定性，但并不推荐无理由关闭或者重启节点，因为这类后台操作均属于高危操作，可能会造成相关服务不可用，请谨慎操作。执行高危操作需将风险告知用户，并由用户以及现场运维人员同意之后，由运维人员进行后台操作。比如以下列表包含了高危操作和禁止操作，可能造成节点或集群不可用：

### 高危操作列表

序列	高危操作
1	重启集群节点或重装操作系统。
2	建议不要在集群节点安装其它软件，可能导致集群节点不可用。

### 禁止操作列表

序 列	禁止操作
1	删除 <code>/var/lib/etcd/</code> ，删除 <code>/var/lib/docker</code> ，删除 <code>/etc/kubernetes/</code> ，删除 <code>/etc/kubesphere/</code> 。
2	磁盘格式化、分区。

## 卸载

卸载将从机器中删除 KubeSphere，该操作不可逆，详见 [卸载说明](#)。

## 组件版本信息

KubeSphere Advanced-v1.0.1 中的相关组件将默认安装以下版本：

组件	版本
KubeSphere	Advanced Edition v1.0.1
Kubernetes	v1.12.5
etcd	3.2.18
OpenPitrix	v0.3.5
Prometheus	v2.3.1
Jenkins	v2.138

# All-in-One 模式

对于首次接触 KubeSphere 高级版的用户，想寻找一个最快安装和体验 KubeSphere 高级版核心功能的方式，all-in-one 模式支持一键安装 KubeSphere 至一台目标机器，请参考如下步骤开始安装。

## 前提条件

- 目前高级版已发布了 v1.0.0 和 v1.0.1，建议下载最新的 [KubeSphere Advanced Edition 1.0.1](#) 至待安装机器中。
- 建议使用 KubeSphere 支持的存储服务，并准备相应的存储服务端。若还未准备存储服务端，为方便测试部署，也可使用 [Local Volume](#) 作为默认存储。

## 第一步：准备主机

您可以参考以下节点规格准备一台符合要求的主机节点开始 all-in-one 模式的安装，若使用 ubuntu 16.04 建议使用其最新的版本 16.04.5。

**说明：**若 Debian 系统未安装 sudo 命令，则需要在安装前使用 root 用户执行 `apt update && apt install sudo` 命令安装 sudo 命令后再进行安装。

操作系统	最小配置	推荐配置
CentOS 7.5 (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 100 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G
Ubuntu 16.04/18.04 LTS (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 100 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G
Red Hat Enterprise Linux Server 7.4 (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 100 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G
Debian Stretch 9.5 (64 bit)	CPU: 4 核	CPU: 8 核

操作系统	最小配置	推荐配置
	内存：8 G 系统盘：100 G	内存：16 G 系统盘：不小于 100 G

## 第二步：准备安装包

- 建议下载最新的 [KubeSphere Advanced-v1.0.1](#)，获取下载链接后可使用 `curl -O url` or `wget url` 命令下载至待安装机器，并执行以下命令。

**注意：**若您的机器已安装了 Advanced-v1.0.0，请直接参考 [升级指南](#) 将您原有的环境一键升级至最新版本，无需参考以下步骤重复安装。

```
$ tar -zxf kubesphere-all-advanced-1.0.1.tar.gz
```

- 进入“[kubesphere-all-advanced-1.0.1](#)”目录。

```
$ cd kubesphere-all-advanced-1.0.1
```

## 第三步：安装 KubeSphere

KubeSphere 安装过程中将会自动化地进行环境和文件监测、平台依赖软件的安装、Kubernetes 和 etcd 的自动化安装，以及存储的自动化配置。最新的 Installer 默认安装的 Kubernetes 版本是 v1.12.5，安装成功后可通过 KubeSphere 控制台右上角点击关于查看安装的版本。KubeSphere 安装包将会自动安装一些依赖软件，如 Ansible (v2.4+)，Python-netaddr (v0.7.18+)，Jinja (v2.9+)。

**说明：**

- 通常情况您不需要修改任何配置，直接安装即可。
- 若您需要自定义配置文件的安装参数，如网络、存储等相关内容需在 `conf/vars.yml` 配置文件中指定或修改。

- 网络：默认插件 [calico](#)。
- All-in-One 默认会用 Local Volume 即本地存储设备作为存储类型，但 Local Volume 不支持动态分配，需手动创建 Persistent Volume (PV)，Installer 会预先创建 10 个可用的 10G PV 供使用。若存储空间不足时则需要手动创建，参见 [Local Volume 使用方法](#)。
- 支持存储类型：[QingCloud 云平台块存储](#)、[QingStor NeonSAN](#)、[GlusterFS](#)、[CephRBD](#)、[NFS](#)、[Local Volume](#)，存储配置相关的详细信息请参考 [存储配置说明](#)。
- 由于 Kubernetes 集群的 Cluster IP 子网网段默认是 [10.233.0.0/18](#)，Pod 的子网网段默认是 [10.233.64.0/18](#)，因此安装 KubeSphere 的节点 IP 地址范围不应与以上两个网段有重複，若遇到地址范围冲突可在配置文件 [conf/vars.yaml](#) 修改 [kube\\_service\\_addresses](#) 或 [kube\\_pods\\_subnet](#) 的参数。

## 注意事项

需要注意的是，如果在云平台上选择使用 KubeSphere 默认的 Calico 网络插件进行安装，并且主机是直接运行在基础网络中，则需要为源 IP (IP/端口集合) 添加防火墙的 ipip 协议，例如在 QingCloud 云平台添加防火墙的 ipip 协议：

添加防火墙[...]  
的规则

提示：编辑完所有规则后，请点击规则列表上方的“应用修改”按钮，使得规则生效。开放外网端口存在一定风险，如 Windows 3389、Redis 6379、Elasticsearch 9300、数据库 3306 等被攻击风险较高的端口。若开放 ssh 22 端口，请您确保禁用密码登录，使用密钥方式登录。

名称	<input type="text"/>	快捷方式
优先级	<input type="text" value="13"/>	<a href="#">ping</a>
数字越小优先级越高，最多可添加100条规则。		<a href="#">ping6</a>
方向	<input type="button" value="下行规则"/>	<a href="#">ssh</a>
行为	<input type="button" value="接受"/>	<a href="#">http</a>
协议	<input type="button" value="IPIP"/>	<a href="#">https</a>
源IP	<input type="text"/> ::	<a href="#">remote</a>
例如 192.168.9.1/24 或 fe80::5054:a8ff:fe81:a71e/64 等，不填表示所有IP地址		<a href="#">openvpn</a>
		<a href="#">pptp</a>
		<a href="#">I2tp</a>
		<a href="#">gre</a>
		<a href="#" style="background-color: #008000; color: white;">ipip</a>



参考以下步骤开始 all-in-one 安装：

**说明：安装时间跟网络情况和带宽、机器配置、安装节点个数等因素有关，已测试过的 all-in-one 模式，在网络良好的情况下以规格列表最小配置安装用时大约为 25 分钟。**

1. 进入 `scripts` 目录

```
$ cd scripts
```

2. 建议使用 `root` 用户安装，执行 `install.sh` 脚本：

```
$ ./install.sh
```

3. 输入数字 `1` 选择第一种即 all-in-one 模式开始安装：

```
#####
# KubeSphere Installer Menu
#####
* 1) All-in-one
* 2) Multi-node
* 3) Quit
#####
https://kubesphere.io/      2018-01-25
#####
Please input an option: 1
```

4. 测试 KubeSphere 单节点安装是否成功：

(1) 待安装脚本执行完后，当看到如下 `"Successful"` 界面，则说明 KubeSphere 安装成功。若需要在外网访问，可能需要绑定公网 EIP 并配置端口转发，若公网 EIP 有防火墙，请在防火墙添加规则放行对应的端口（比如 30880），保证外网流量可以通过该端口，外部才能够访问。

```
successssful!  
#####  
## Welcome to KubeSphere! ##  
#####  
  
Console: http://192.168.0.8:30880  
Account: admin  
Password: passw0rd  
  
NOTE: Please modify the default password after login.  
#####
```

提示：如需要再次查看以上的界面信息，可在安装包目录下执行 `cat kubesphere/kubesphere_running` 命令查看。

(2) 安装成功后，浏览器访问对应的 url，即可进入 KubeSphere 登录界面，可使用默认的用户名和密码登录 KubeSphere 控制台体验，参阅 [快速入门](#) 帮助您快速上手 KubeSphere。



# Multi–Node 模式

Multi–Node 即多节点集群部署，部署前建议您选择集群中任意一个节点作为一台任务执行机（taskbox），为准备部署的集群中其他节点执行部署的任务，且 Taskbox 应能够与待部署的其他节点进行 ssh 通信。

## 前提条件

- 目前高级版已发布了 v1.0.0 和 v1.0.1，建议下载最新的 [KubeSphere Advanced Edition 1.0.1](#) 至待安装机器中。
- 建议使用 KubeSphere 支持的存储服务，并准备相应的存储服务端，存储服务端的磁盘容量参考主机规格表中的推荐配置或选择更高的容量。为方便初次安装但没有准备存储服务端时进行部署测试，也可配置部署 NFS server in Kubernetes 到当前集群。

## 第一步：准备主机

您可以参考以下节点规格 准备 **至少 2 台** 符合要求的主机节点开始 multi–node 模式的部署，若使用 ubuntu 16.04 建议使用其最新的版本 16.04.5。

**说明：**若 Debian 系统未安装 sudo 命令，则需要在安装前使用 root 用户执行 `apt update && apt install sudo` 命令安装 sudo 命令后再进行安装。

操作系统	最小配置	推荐配置
CentOS 7.5 (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 40 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G
Ubuntu 16.04/18.04 LTS (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 40 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G
Red Hat Enterprise Linux Server 7.4 (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 40 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G

操作系统	最小配置	推荐配置
Debian Stretch 9.5 (64 bit)	CPU: 4 核 内存: 8 G 系统盘: 40 G	CPU: 8 核 内存: 16 G 系统盘: 不小于 100 G

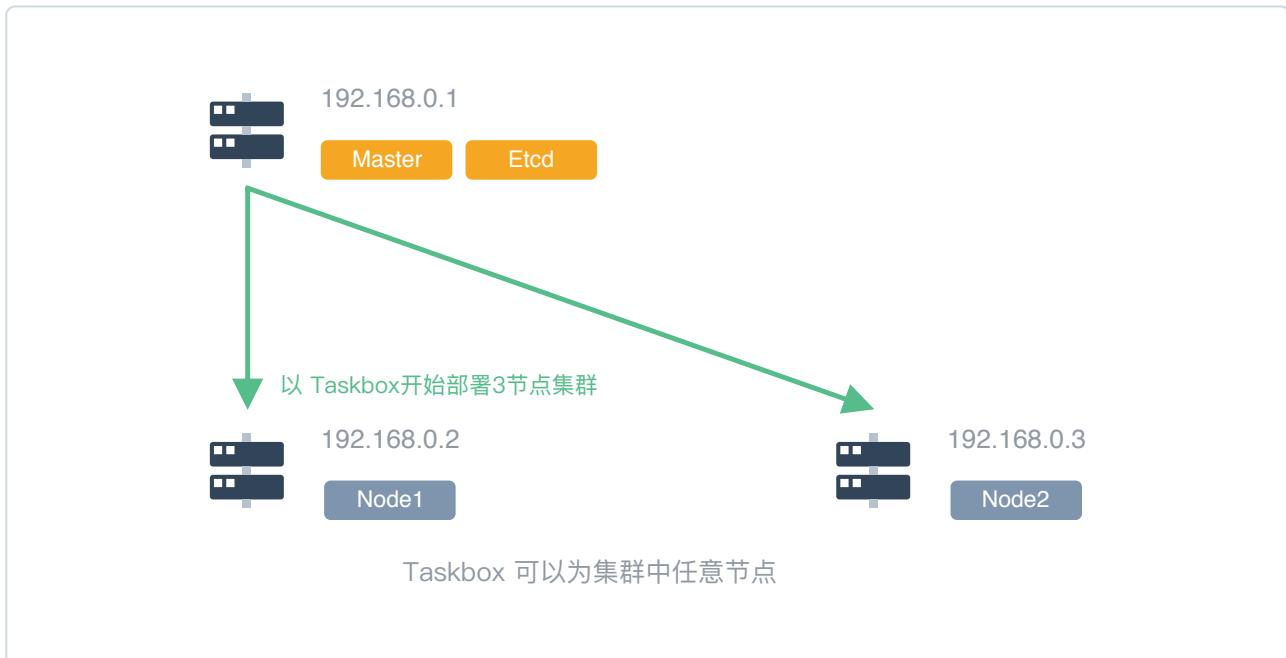
以下用一个示例介绍 multi-node 模式部署多节点环境，本示例准备了 3 台 CentOS 7.5 的主机并以 root 用户准备安装。登录主机名为 Master 的节点作为任务执行机 Taskbox 来执行安装步骤。在 [安装说明](#) 已经介绍了 KubeSphere 集群架构是由管理节点 (Master) 和工作节点 (Node) 构成的，这 3 台主机分别部署 1 个 Master 节点和 2 个 Node 节点，也称 Worker 节点，在底层的 Kubernetes 中 Worker 节点跟 Master 节点都运行着一个 kubelet 组件，但 Master 节点上还会运行 kube-apiserver、kube-scheduler、kube-controller-manager 这三个系统 Pod。

**说明：**高级版支持 Master 和 etcd 节点高可用配置，但本示例仅作为测试部署的演示，因此 3 台主机中仅部署单个 Master 和单个 etcd，正式环境建议配置 Master 和 etcd 节点的高可用，请参阅 [Master 和 etcd 节点高可用配置](#)。

假设主机信息如下所示：

主机 IP	主机名	集群角色
192.168.0.1	master	master, etcd
192.168.0.2	node1	node
192.168.0.3	node2	node

**集群架构：**单 master 单 etcd 双 node



## 第二步：准备安装配置文件

- 建议下载最新的 [KubeSphere Advanced Edition 1.0.1](#)，获取下载链接后可使用 `curl -O url` or `wget url` 命令下载至待安装机器，并执行以下命令。

**注意：**若您的机器已安装了 Advanced-v1.0.0，请直接参考 [升级指南](#) 将您原有的环境一键升级至最新版本，无需参考以下步骤重复安装。

```
$ tar -zxf kubesphere-all-advanced-1.0.1.tar.gz
```

- 进入“`kubesphere-all-advanced-1.0.1`”目录。

```
$ cd kubesphere-all-advanced-1.0.1
```

- 编辑主机配置文件 `conf/hosts.ini`，为了对待部署目标机器及部署流程进行集中化管理配置，集群中各个节点在主机配置文件 `hosts.ini` 中应参考如下配置，建议使用 `root` 用户进行安装。

## 说明：

- 若以非 root 用户（如 ubuntu 用户）进行安装，可参考配置文件 `conf/hosts.ini` 的注释中 `non-root` 用户示例部分编辑。
- 如果在 taskbox 使用 root 用户无法 ssh 连接到其他机器，也需要参考 `conf/hosts.ini` 的注释中 `non-root` 用户示例部分，但执行安装脚本 `install.sh` 时建议切换到 root 用户，如果对此有疑问可参考 [常见问题 - 问题 2](#)。
- master, node1, node2 作为集群各个节点的主机名，若需要自定义主机名则所有主机名都需要都使用小写形式。

以下示例在 CentOS 7.5 上使用 `root` 用户安装，每台机器信息占一行，不能分行。

## root 配置示例：

```
[all]
master ansible_connection=local ip=192.168.0.1
node1 ansible_host=192.168.0.2 ip=192.168.0.2 ansible_ssh_pass=PASSWORD
node2 ansible_host=192.168.0.3 ip=192.168.0.3 ansible_ssh_pass=PASSWORD

[kube-master]
master

[kube-node]
node1
node2

[etcd]
master

[k8s-cluster:children]
kube-node
kube-master
```

## 说明：

- [all] 中需要修改集群中各个节点的内网 IP 和主机 root 用户密码。主机名为 “master”的节点

作为已通过 SSH 连接的 Taskbox 所以无需填写密码，[all] 中其它节点的参数比如 node1 和 node2 的 `ansible_host` 和 `ip` 都替换为当前 node1 和 node2 的内网 IP，将 `ansible_ssh_pass` 替换为您准备的主机 `root` 用户登录密码。

- 应将主机名 “master” 填入 [kube-master] 和 [etcd] 部分，因为 “master” 节点作为 taskbox，用来执行整个集群的安装任务，同时 “master” 节点在 KubeSphere 集群架构中也将作为管理节点 Master 和负责保存持久化数据的 etcd。
- 将主机名 “node1”， “node2” 填入 [kube-node] 部分，作为 KubeSphere 集群的 node 节点。

#### 参数解释：

- `ansible_connection`: 与主机的连接类型，此处设置为 `local` 即本地连接。
- `ansible_host`: 集群中将要连接的主机地址或域名。
- `ip`: 集群中将要连接的主机 IP。
- `ansible_ssh_pass`: 待连接主机 `root` 用户的密码。

5. 为方便测试部署和演示，本文档以部署 NFS Server 至当前集群为例。注意，在正式环境使用需准备 KubeSphere 支持的存储服务端。以下存储配置示例需在 `vars.yml` 中修改，将安装 NFS server in Kubernetes 至当前集群作为默认存储类型，参数释义详见 [存储配置说明](#)。

#### 说明：

- 网络、存储等相关内容需在 `conf/vars.yml` 配置文件中指定或修改，根据配置文件按需修改相关配置项，未做修改将以默认参数执行。
- 网络：默认插件 `Calico`。
- 支持存储类型：[QingCloud 云平台块存储](#)、[QingStor NeonSAN](#)、[NFS](#)、[GlusterFS](#)、[Ceph RBD](#)、[Local Volume \(仅支持 all-in-one\)](#)，存储配置相关的详细信息请参考 [存储配置说明](#)。
- Multi-Node 安装时需要配置持久化存储，因为它不支持 Local Volume，因此把 Local Volume 的配置修改为 `false`，然后配置持久化存储如 QingCloud-CSI (QingCloud 块存储插件)、NeonSAN CSI (NeonSAN 存储插件)、NFS、GlusterFS、CephRBD 等。如下所示配置 NFS server in Kubernetes，将 `nfs_server_enable` 和 `nfs_server_is_default_class` 设置为 `true`。
- 由于 Kubernetes 集群的 Cluster IP 子网网段默认是 `10.233.0.0/18`，Pod 的子网网段默认是 `10.233.64.0/18`，因此部署 KubeSphere 的节点 IP 地址范围不应与以上两个网段有重复，若遇到地址范围冲突可在配置文件 `conf/vars.yaml` 修改 `kube_service_addresses` 或 `kube_pods_subnet` 的参数。

存储配置示例：

```
# Local volume provisioner deployment(Only all-in-one)
local_volume_provisioner_enabled: false
local_volume_provisioner_storage_class: local
local_volume_is_default_class: false

# NFS-Server provisioner deployment
nfs_server_enable: true
nfs_server_is_default_class: true
```

## 第三步：安装 KubeSphere

KubeSphere 多节点部署会自动化地进行环境和文件监测、平台依赖软件的安装、Kubernetes 和 etcd 集群的自动化部署，以及存储的自动化配置。Installer 默认安装的 Kubernetes 版本是 v1.12.5，安装成功后可通过 KubeSphere 控制台右上角点击关于查看安装的版本。KubeSphere 安装包将会自动安装一些依赖软件，如 Ansible (v2.4+)，Python-netaddr (v0.7.18+)，Jinja (v2.9+)。

参考以下步骤开始 multi-node 部署。

**说明：**由于 multi-node 的安装时间跟网络情况和带宽、机器配置、安装节点个数等因素都有关，  
此处暂不提供时间标准。

1. 进入 `scripts` 目录：

```
$ cd scripts
```

2. 建议使用 root 用户安装，执行 `install.sh` 脚本：

```
$ ./install.sh
```

3. 输入数字 2 选择第二种 Multi-node 模式开始部署，安装程序会提示您是否已经配置过存储，若未配置请输入 “no”，返回目录继续配置存储并参考 [存储配置说明](#)。

```
#####
# KubeSphere Installer Menu
#####
* 1) All-in-one
* 2) Multi-node
* 3) Quit
#####
https://kubesphere.io/      2018-01-25
#####
Please input an option: 2
```

#### 4. 测试 KubeSphere 集群部署是否成功：

(1) 待安装脚本执行完后，当看到如下“Successful”界面，则说明 KubeSphere 安装成功。若需要在外网访问，可能需要绑定公网 EIP 并配置端口转发，若公网 EIP 有防火墙，请在防火墙添加规则放行对应的端口（比如 30880），保证外网流量可以通过该端口，外部才能够访问。

```
successsful!
#####
###          Welcome to KubeSphere!          ###
#####

Console: http://192.168.0.1:30880
Account: admin
Password: passw0rd

NOTE: Please modify the default password after login.
```

**提示：**如需要再次查看以上的界面信息，可在安装包目录下执行 `cat kubesphere/kubesphere_running` 命令查看。

(2) 安装成功后，浏览器访问对应的 url，即可进入 KubeSphere 登录界面，可使用默认的用户名和密码登录 KubeSphere 控制台体验，参阅 [快速入门](#) 帮助您快速上手 KubeSphere。



# 升级

KubeSphere 目前最新的版本 1.0.1 已发布，该版本对 1.0.0 进行了改进和功能优化，并修复了已知的 Bug，更新详情可参考 [Release Note – v1.0.1](#)。

若您已安装的环境为 1.0.0 版本，我们强烈建议您升级至最新的版本 1.0.1，最新的 Installer 支持将 KubeSphere 从 1.0.0 一键升级至 1.0.1，无需卸载和重新安装；同时支持升级 Kubernetes 和 etcd 至指定版本，升级过程中所有节点将会逐个升级，可能会出现短暂的应用服务中断现象，请您安排合理的升级时间。

## 说明：

### 升级过程中以下组件的版本将会有更新：

- KubeSphere: 1.0.0 将升级至 1.0.1
- Kubernetes: v1.12.3 将升级至 v1.12.5，同时支持升级至 v1.13.2
- etcd: 默认 v3.2.18，同时支持升级至 v3.2.24

# 如何升级

## 第一步：下载最新安装包

1.1. 下载 [KubeSphere Advanced Edition 1.0.1 安装包](#) 至已安装了 1.0.0 的环境，获取下载链接后可使用 `curl -O url` or `wget url` 命令下载至待安装机器，并执行以下命令。

```
$ tar -zxf kubesphere-all-advanced-1.0.1.tar.gz
```

1.2. 进入 “[kubesphere-all-advanced-1.0.1](#)” 目录

```
$ cd kubesphere-all-advanced-1.0.1
```

## 查看版本参数 (可选)

待升级的组件版本号已在 `/conf/vars.yml` 文件中定义，默认情况下将升级至已指定的版本，若需要修改可编辑其参数。

```
## Change this to use another Kubernetes version
ks_version: 1.0.1
kube_version: v1.12.5
etcd_version: v3.2.18
```

## 第二步：修改配置文件

升级将默认读取 1.0.1 的 `conf` 目录下的配置文件，因此在升级前需要将 1.0.0 中 `conf` 下的配置文件中的参数都同步到 1.0.1 版本安装包的对应文件中，修改配置文件分以下两种情况：

### All-in-One

若 1.0.0 是以 **all-in-one** 模式安装的单节点集群，那么升级前在 1.0.1 中无需修改 `conf/hosts.ini` 文件，仅需要确认 1.0.0 的 `conf/vars.yml` 参数配置是否修改，若有修改则需要在 1.0.1 的对应文件中同步所有修改的参数。

例如，目前 1.0.1 中默认使用 **Local** 作为存储类型，如果您的 1.0.0 配置使用了其它存储类型，如 QingCloud 块存储、NFS、Ceph RBD 或 GlusterFS 等，那么在 1.0.1 安装包的 `conf/vars.yml` 中也需要进行相应的设置（即与 1.0.0 的配置保持一致），并将 Local 的存储配置设置为 `false`，参数释义详见 [存储配置参数](#)。

### Multi-Node

若 1.0.0 是以 **multi-node** 模式安装的多节点集群，那么升级前需将当前的 `conf/hosts.ini` 和 `conf/vars.yml` 中的配置都同步到 1.0.1 的对应文件中：

- 将 1.0.0 的 `conf/hosts.ini` 中的主机参数配置覆盖至 1.0.1 安装包的 `conf/hosts.ini`，参数释义详见 [Multi-Node 模式 – 准备安装配置文件](#)。

- 将 1.0.0 的 `conf/vars.yml` 中所有修改过的参数配置同步至 1.0.1 的 `conf/vars.yml`。例如，1.0.0 配置使用的是 QingCloud 块存储、NFS、Ceph RBD 或 GlusterFS 这一类存储，那么在 1.0.1 安装包的 `conf/vars.yml` 中也要进行相应的设置（即与 1.0.0 的配置保持一致），并将 Local 的存储配置设置为 `false`，参数释义详见 [存储配置参数](#)。

**注意：**`conf/vars.yml` 中的配置项都是 `key:value` 的形式，QingCloud CSI 和 NFS in Kubernetes 两项配置的 `key` 在 1.0.1 已更新，若配置了这两类存储则应使用 1.0.1 中的 `key`。

### 第三步：开始升级

参考如下步骤进行升级：

- 在 `kubesphere-all-advanced-1.0.1` 目录下进入 `/script` 目录，执行 `upgrade.sh` 脚本，建议使用 `root` 用户：

```
$ ./upgrade.sh
```

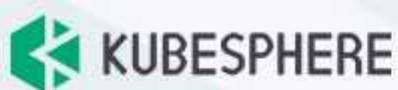
- 在以下返回中输入 `yes` 开始升级：

```
ks_version: 1.0.0 to 1.0.1  
k8s_version: v1.12.3 to v1.12.5
```

```
The relevant information is shown above, please confirm: (yes/no) yes
```

- 由于升级是对逐个节点进行升级，因此升级时间与集群节点规模与网络状况相关。升级完成后，可使用之前的 KubeSphere 访问地址和账户登陆 Console，点击右上角的「关于」查看版本是否更新成功。

Advanced Edition



KubeSphere Version: 1.0.1

Kubernetes Version: v1.12.5    OpenPitrix: v0.3.5

Copyright © 2018, Yunify Technology Co., Ltd. All Rights Reserved.

[官方网站](#)    [文档中心](#)

# Master 和 etcd 节点高可用配置

Multi–Node 模式安装 KubeSphere 可以帮助用户顺利地部署一个多节点集群用于开发和测试，在实际的生产环境我们还需要考虑 master 节点的高可用问题，因为如果 master 节点上的几个服务 kube–apiserver、kube–scheduler 和 kube–controller–manager 都是单点的而且都位于同一个节点上，一旦 master 节点宕机，可能不应答当前正在运行的应用，将导致 KubeSphere 集群无法变更，对线上业务存在很大的风险。

负载均衡器 (Load Balancer) 可以将来自多个公网地址的访问流量分发到多台主机上，并支持自动检测并隔离不可用的主机，从而提高业务的服务能力和可用性。除此之外，还可以通过 Keepalived 和 Haproxy 的方式实现多个 master 节点的高可用部署。

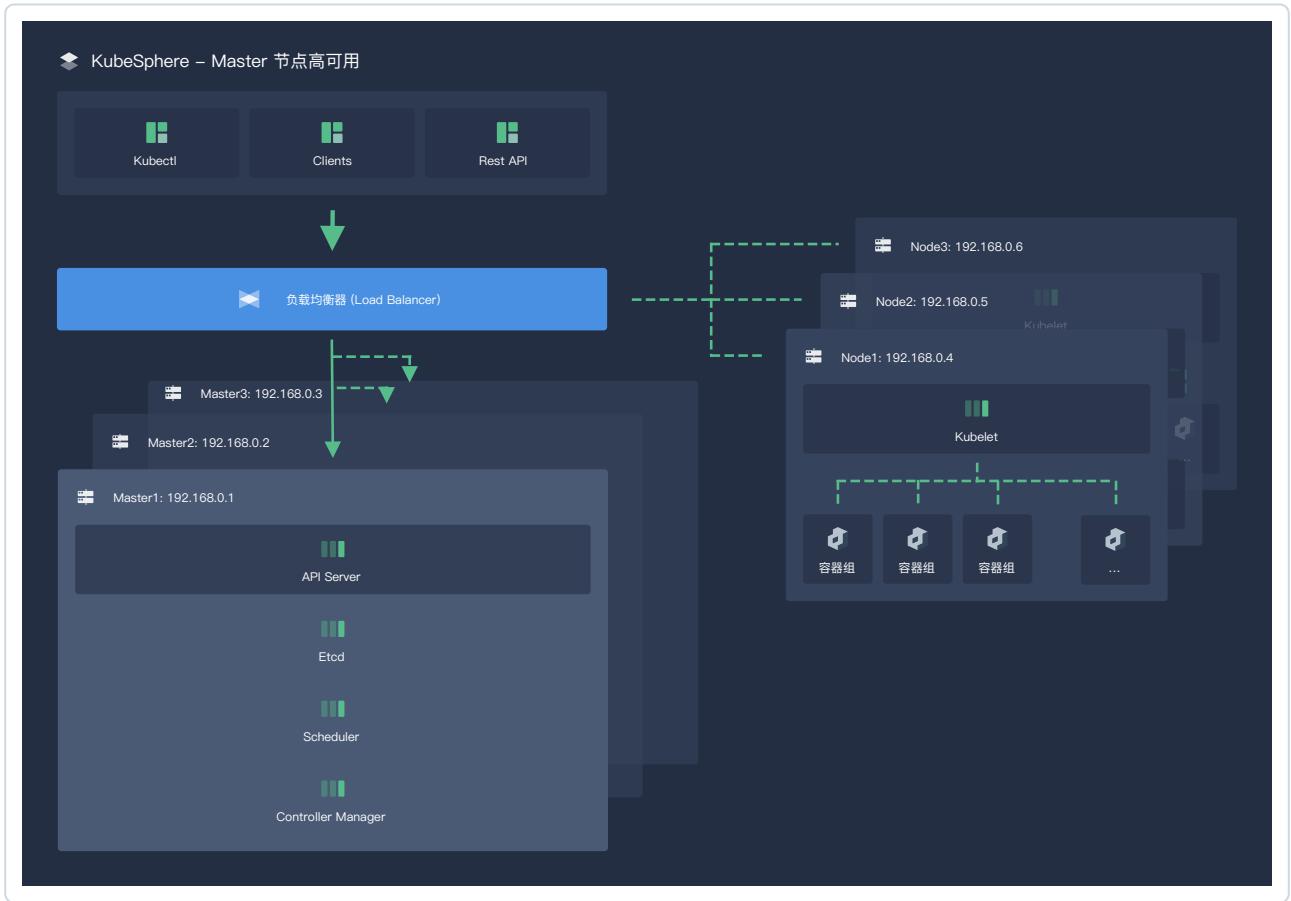
而 etcd 作为一个高可用键值存储系统，整个集群的持久化数据，则由 kube–apiserver 处理后保存到 etcd 中。etcd 节点至少需要 1 个，但部署多个 etcd (奇数个) 能够使集群更可靠。本文档以配置 [QingCloud 云平台](#) 的 [负载均衡器 \(Load Balancer\)](#) 为例，引导您如何配置高可用的 master 节点，并说明如何配置和部署高可用的 etcd 集群。

## 前提条件

- 请确保已参阅 [Multi–Node 模式](#)，本文档仅说明安装过程中如何修改配置文件来配置 master 节点高可用，该配置作为一个可选配置项，完整的安装流程和配置文件中的参数解释说明以 [Multi–Node 模式](#) 为准。
- 已有 [QingCloud 云平台](#) 账号，用于申请负载均衡器给多个 master 节点做负载均衡。

## Master 和 etcd 节点高可用架构

本示例准备了 6 台主机，主机规格参考 [Multi–Node 模式 – 节点规格](#)，将在其中 3 台部署 Master 和 etcd 集群，可编辑主机配置文件 `conf/hosts.ini` 来配置 Master 和 etcd 的高可用。



## 准备负载均衡器

以创建 QingCloud 云平台负载均衡器为例，简单说明其创建步骤，详细介绍可参考 [QingCloud 官方文档](#)。

### 第一步：创建负载均衡器

登录 [QingCloud 云平台](#)，在 **网络与 CDN** 选择 **负载均衡器**，填写基本信息。如下示例在北京 3 区 – C 创建一个负载均衡器，部署方式选择 **单可用区**，网络选择了集群主机所在的 VPC 网络的私有网络（例如本示例的六台主机都在 kubesphere 私有网络中），其它设置保持默认即可。填写基本信息后，点击 **提交** 完成创建。



## 第二步：创建监听器

进入上一步创建成功的负载均衡器，为其创建一个监听器，监听 TCP 协议的 6443 端口，监听的端口号也可以是其它任意端口，但在 vars.yml 中配置应与 port 一致，监听器的基本信息应参考如下填写。

- 监听协议：选择 TCP 协议
- 端口：填写 6443 端口
- 负载方式：选择轮询

**注意：创建监听器后请检查负载均衡器的防火墙规则，确保 6443 端口已添加至防火墙规则并且外网流量可以通过，否则外网无法访问该端口的服务，安装可能会失败。**



### 第三步：添加后端

在当前的负载均衡器中点击 **添加后端**，私有网络选择集群主机所在的私有网络，点击 **高级搜索**，可以一次勾选多台后端主机，例如要通过负载均衡器实现 Master 节点的高可用，此处则勾选 master1、master2、master3 这三台 Master 主机，注意这里端口需填写 **6443**，它是 api-server 的默认端口 (Secure Port)，添加完成后点击 **提交**。



添加后端后，需请点击 **应用修改** 使之生效，可以在列表查看目前负载均衡器的添加的三台 Master 节点。注意，刚添加完的后端主机状态在监听器中可能显示“不可用”，是因为 api-server 对应的 6443

服务端口还未运行开放，这属于正常现象。待安装成功后，后端主机上的 api-server 的服务端口 6443 将被暴露出来，后端状态变为“活跃”，说明负载均衡器已在正常工作。

The screenshot shows the KubeSphere UI for managing listeners. At the top, there are tabs for '监听器' (Listener), '图形化' (Graphic), and '配置备份' (Configuration Backup). A yellow message box says '修改尚未更新, 请点击"应用修改"使之生效' (Modification has not been updated, please click "Apply Modification" to make it effective). Below the tabs are three buttons: '+ 创建监听器' (Create Listener), '应用修改' (Apply Modification) which is highlighted with a red arrow, and '节点监控' (Node Monitoring). The main area displays a table for the 'monitoring' listener (ID: lbl-gqwq3c39). The table includes columns for Name, Status, Host / Router / IP, Port, Weight, Forwarding Strategy, Monitoring, and Operation. Three nodes are listed: master2 (不可用), master1 (不可用), and master3 (不可用), all assigned port 6443 with weight 1 and no forwarding strategy. There are buttons for '添加后端' (Add Backend) and '删除' (Delete) at the top of the table, and a page navigation bar with '合计: 6 每页: 100'.

## 修改主机配置文件

可在如下示例的 [kube-master] 和 [etcd] 部分填入主机名 master1、master2、master3 作为高可用的 Master 和 etcd 集群。注意，etcd 节点个数需要设置为 奇数个，由于 etcd 内存本身消耗比较大，部署到工作节点 (node) 上很容易出现资源不足，因此不建议在工作节点上部署 etcd 集群。为了对待部署目标机器及部署流程进行集中化管理配置，集群中各个节点在主机配置文件 hosts.ini 中应参考如下配置，建议使用 root 用户安装。

以下示例在 CentOS 7.5 上使用 root 用户安装，若以非 root 用户（如 ubuntu）进行安装，可参考主机配置文件的注释 non-root 示例部分编辑。

### 说明：

- 若以非 root 用户（如 ubuntu 用户）进行安装，可参考配置文件 conf/hosts.ini 的注释中 non-root 用户示例部分编辑。
- 如果在 taskbox 使用 root 用户无法 ssh 连接到其他机器，也需要参考 conf/hosts.ini 的注释中 non-root 用户示例部分，但执行安装脚本 install.sh 时建议切换到 root 用户，如果对此有疑问可参考 [常见问题 - 问题 2](#)。

## host.ini 配置示例

```
[all]
master1 ansible_connection=local ip=192.168.0.1
master2 ansible_host=192.168.0.2 ip=192.168.0.2 ansible_ssh_pass=PASSWORD
master3 ansible_host=192.168.0.3 ip=192.168.0.3 ansible_ssh_pass=PASSWORD
node1 ansible_host=192.168.0.4 ip=192.168.0.4 ansible_ssh_pass=PASSWORD
node2 ansible_host=192.168.0.5 ip=192.168.0.5 ansible_ssh_pass=PASSWORD
node3 ansible_host=192.168.0.6 ip=192.168.0.6 ansible_ssh_pass=PASSWORD

[kube-master]
master1
master2
master3

[kube-node]
node1
node2
node3

[etcd]
master1
master2
master3

[k8s-cluster:children]
kube-node
kube-master
```

## 配置负载均衡器参数

在 QingCloud 云平台准备好负载均衡器后，需在 `vars.yaml` 配置文件中修改相关参数。假设负载均衡器的内网 IP 地址是 `192.168.0.10`（这里需替换为您的负载均衡器实际 IP 地址），负载均衡器设置的 TCP 协议的监听端口（port）为 `6443`，那么在 `conf/vars.yml` 中参数配置参考如下示例（`loadbalancer_apiserver` 作为可选配置项，在配置文件中应取消注释）。

- 注意， address 和 port 在配置文件中应缩进两个空格。
- 负载均衡器的域名默认为 ”lb.kubesphere.local”， 供集群内部访问。如果需要修改域名则先取消注释再自行修改。

### vars.yml 配置示例

```
## External LB example config
## apiserver_loadbalancer_domain_name: "lb.kubesphere.local"
loadbalancer_apiserver:
  address: 192.168.0.10
  port: 6443
```

完成 master 和 etcd 高可用的参数配置后，请继续参阅 [Multi–Node 模式 – 存储配置示例](#) 在 vars.yml 中配置持久化存储相关参数，并继续多节点的安装。

## 存储配置说明

目前，Installer 支持以下类型的存储作为存储服务端，为 KubeSphere 提供持久化存储（更多的存储类型持续更新中）：

- QingCloud 云平台块存储
- QingStor NeonSAN
- Ceph RBD
- GlusterFS
- NFS
- NFS in Kubernetes (仅限 multi-node 部署测试使用)
- Local Volume (仅限 all-in-one 部署测试使用)

同时，Installer 集成了 [QingCloud 云平台块存储 CSI 插件](#) 和 [QingStor NeonSAN CSI 插件](#)，仅需在安装前简单配置即可对接 QingCloud 云平台块存储或 NeonSAN 作为存储服务，前提是需要有操作 [QingCloud 云平台](#) 资源的权限或已有 NeonSAN 服务端。Installer 也集成了 NFS、GlusterFS 或 Ceph RBD 这类存储的客户端，用户需提前准备相关的存储服务端，然后在 `vars.yml` 配置对应的参数即可对接相应的存储服务端。

Installer 对接的开源存储服务端和客户端，以及 CSI 插件，已测试过的版本如下：

名称	版本	参考
Ceph RBD Server	v0.94.10	若用于测试部署可参考 <a href="#">部署 Ceph 存储服务端</a> ，如果是正式环境搭建请参考 <a href="#">Ceph 官方文档</a>
Ceph RBD Client	v12.2.5	在安装 KubeSphere 前仅需在 <code>vars.yml</code> 配置相应参数即可对接其存储服务端，参考 <a href="#">Ceph RBD</a>
GlusterFS Server	v3.7.6	若用于测试部署可参考 <a href="#">部署 GlusterFS 存储服务端</a> ，如果是正式环境搭建请参考 <a href="#">Gluster 官方文档</a> 或 <a href="#">Gluster Docs</a> ，并且需要安装 <a href="#">Heketi 管理端 (v3.0.0)</a>
GlusterFS Client	v3.12.10	在安装 KubeSphere 前仅需在 <code>vars.yml</code> 配置相应参数即可对接其存储服务端，配置详见 <a href="#">GlusterFS</a>
NFS Server in	v1.0.9	配置详见 <a href="#">NFS Server 配置</a>

名称	版本	参考
Kubernetes		
NFS Client	v3.1.0	在安装 KubeSphere 前仅需在 <code>vars.yml</code> 配置相应参数即可对接其存储服务端，详见 <a href="#">NFS Client</a>
QingCloud-CSI	v0.2.0.1	在安装 KubeSphere 前仅需在 <code>vars.yml</code> 配置相应参数，详见 <a href="#">QingCloud CSI</a>
NeonSAN-CSI	v0.3.0	在安装 KubeSphere 前仅需在 <code>vars.yml</code> 配置相应参数，详见 <a href="#">Neonsan-CSI</a>

**说明：** 集群中不可同时存在两个默认存储类型，若要指定默认存储类型前请先确保当前集群中无默认存储类型。

## 配置文件释义

准备了满足要求的存储服务端后，只需要参考以下表中的参数说明，在 `conf/vars.yml` 中，根据您存储服务端所支持的存储类型，首先在 `conf/vars.yml` 的相应部分参考示例或注释修改对应参数并保存，然后再执行安装程序，即可完成集群存储类型的配置。

以下对 `vars.yml` 存储相关的参数配置做简要说明（参数详解请参考 [storage classes](#)）。

**特别注意：** `vars.yml` 中默认配置了 Local 类型的存储作为集群默认的存储类型，若配置其他类型的存储，则首先需要将 Local 相关的参数配置都修改为 false，然后再根据您的存储服务端类型，参考如下说明修改 `vars.yml` 中对应的存储相关部分的参数配置。

## QingCloud 云平台块存储

KubeSphere 支持使用 QingCloud 云平台块存储作为平台的存储服务，如果希望体验动态分配（Dynamic Provisioning）方式创建存储卷，推荐使用 [QingCloud 云平台块存储](#)，平台已集成 [QingCloud-CSI](#) 块存储插件支持对接块存储，仅需简单配置即可使用 QingCloud 云平台各种性能的块存储服务。

**QingCloud–CSI** 块存储插件实现了 CSI 接口，并且支持 KubeSphere 使用 QingCloud 云平台的存储资源。块存储插件部署后，用户可创建访问模式 (Access Mode) 为 **单节点读写 (ReadWriteOnce)** 的基于 QingCloud 的超高性能型 (超高性能型硬盘只能用在超高性能型主机)、性能型 (性能型硬盘只能用在性能型主机) 或容量型硬盘的存储卷并挂载至工作负载。在安装 KubeSphere 时配置 QingCloud–CSI 插件的参数说明如下。

**注意：在 KubeSphere 集群内使用到性能型、超高性能型、企业型或基础型硬盘时，集群的主机类型也应与硬盘的类型保持一致。**

<b>QingCloud–CSI</b>	<b>Description</b>
qingcloud_csi_enabled	是否使用 QingCloud–CSI 作为持久化存储，是：true；否：false
qingcloud_csi_is_default_class	是否设定为默认的存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认的存储类型
qingcloud_access_key_id , qingcloud_secret_access_key	通过 <a href="#">QingCloud 云平台控制台</a> 的右上角账户图标选择 <b>API 密钥</b> 创建密钥获得
qingcloud_zone	zone 应与 Kubernetes 集群所在区相同，CSI 插件将会操作此区的存储卷资源。例如：zone 可以设置为 sh1a（上海一区-A）、sh1b（上海1区-B）、pek2（北京2区）、pek3a（北京3区-A）、gd1（广东1区）、gd2a（广东2区-A）、ap1（亚太1区）、ap2a（亚太2区-A）
qingcloud_type	QingCloud 云平台块存储的类型，0 代表性能型硬盘，1 或 2（根据集群所在区不同而参数不同）代表容量型硬盘，3 代表超高性能型硬盘，详情见 <a href="#">QingCloud 官方文档</a>
qingcloud_maxSize, qingcloud_minSize	限制存储卷类型的存储卷容量范围，单位为 GiB
qingcloud_stepSize	设置用户所创建存储卷容量的增量，单位为 GiB
qingcloud_fsType	存储卷的文件系统，支持 ext3, ext4, xfs. 默认为 ext4

## QingStor NeonSAN

NeonSAN-CSI 插件支持对接青云自研的企业级分布式存储 [QingStor NeonSAN](#) 作为存储服务，若您准备好 NeonSAN 服务端后，即可在 `conf/vars.yml` 配置 NeonSAN-CSI 插件对接其存储服务端。详见 [NeonSAN-CSI 参数释义](#)。

NeonSAN	Description
neonsan_csi_enabled	是否使用 NeonSAN 作为持久化存储，是：true；否：false
neonsan_csi_is_default_class	是否设定为默认存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认存储类型
neonsan_csi_protocol	NeonSAN 服务端传输协议，如 TCP 或 RDMA
neonsan_server_address	NeonSAN 服务端地址
neonsan_cluster_name	NeonSAN 服务端集群名称
neonsan_server_pool	Kubernetes 插件从哪个 pool 内创建存储卷，默认值为 kube
neonsan_server_replicas	NeonSAN image 的副本个数，默认为 1
neonsan_server_stepSize	用户所创建存储卷容量的增量，单位为 GiB，默认为 1
neonsan_server_fsType	存储卷的文件系统格式，默认为 ext4

## Ceph RBD

[Ceph RBD](#) 是一个分布式存储系统，在 `conf/vars.yml` 配置的释义如下。

Ceph_RBD	Description
ceph_rbd_enabled	是否使用 Ceph RBD 作为持久化存储，是：true；否：false
ceph_rbd_storage_class	存储类型名称
ceph_rbd_is_default_class	是否设定为默认存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认存储类型
ceph_rbd_monitors	根据 Ceph RBD 服务端配置填写，可参考 <a href="#">Kubernetes 官方文档</a>

Ceph_RBD	Description
ceph_rbd_admin_id	能够在存储池中创建的客户端 ID , 默认: admin, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_admin_secret	Admin_id 的 secret, 安装程序将会自动在 kube-system 项目内创建此 secret, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_pool	可使用的 Ceph RBD 存储池, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_user_id	用于映射 RBD 的 ceph 客户端 ID 默认: admin, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_user_secret	User_id 的 secret, 需注意在所使用 rbd image 的项目内都需创建此 Secret, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_fsType	存储卷的文件系统, kubernetes 支持 fsType, 默认: ext4, 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_imageFormat	Ceph RBD 格式, 默认: "1", 可参考 <a href="#">Kubernetes 官方文档</a>
ceph_rbd_imageFeatures	当 <code>ceph_rbd_imageFormat</code> 字段不为 1 时需填写此字段, 可参考 <a href="#">Kubernetes 官方文档</a>

注：存储类型中创建 secret 所需 ceph secret 如 `ceph_rbd_admin_secret` 和 `ceph_rbd_user_secret` 可在 ceph 服务端通过以下命令获得：

```
$ ceph auth get-key client.admin
```

## GlusterFS

[GlusterFS](#) 是一个开源的分布式文件系统, 配置时需提供 heketi 所管理的 glusterfs 集群, 在 `conf/vars.yml` 配置的释义如下。

GlusterFS	Description
glusterfs_provisioner_enabled	是否使用 GlusterFS 作为持久化存储, 是: true; 否: false
glusterfs_provisioner_storage_class	存储类型的名称

GlusterFS	Description
glusterfs_is_default_class	是否设定为默认存储类型, 是: true; 否: false 注: 系统中存在多种存储类型时, 只能设定一种为默认存储类型
glusterfs_provisioner_restauthenabled	Gluster 启用对 REST 服务器的认证, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_resturl	Heketi 服务端 url, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_clusterid	Gluster 集群 id, 登录 heketi 服务端输入 heketi-cli cluster list 得到 Gluster 集群 id, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_restuser	能够在 Gluster pool 中创建 volume 的 Heketi 用户, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_secretName	secret 名称, 安装程序将会在 kube-system 项目内自动创建此 secret, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_gidMin	glusterfs_provisioner_storage_class 中 GID 的最小值, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_gidMax	glusterfs_provisioner_storage_class 中 GID 的最大值, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
glusterfs_provisioner_volumetype	Volume 类型, 参数配置请参考 <a href="#">Kubernetes 官方文档</a>
jwt_admin_key	heketi 服务器中 <code>/etc/heketi/heketi.json</code> 的 jwt.admin.key 字段

注: Glusterfs 存储类型中所需的 `glusterfs_provisioner_clusterid` 可在 glusterfs 服务端通过以下命令获得:

```
$ export HEKETI_CLI_SERVER=http://localhost:8080
$ heketi-cli cluster list
```

## NFS

**注意：NFS 与 NFS in Kubernetes 是两种不同类型的存储类型，在 vars.yml 中配置时仅需配置其中一种作为默认的存储类型即可。**

**NFS** 即网络文件系统，它允许网络中的计算机之间通过 TCP/IP 网络共享资源。需要预先准备 NFS 服务端，本方法可以使用 QingCloud 云平台 [vNAS](#) 作为 NFS 服务端。在 `conf/vars.yml` 配置的释义如下。

关于在安装前如何配置 QingCloud vNas，本文档在 [常见问题 – 安装前如何配置 QingCloud vNas](#) 给出了一个详细的示例供参考。

NFS	Description
<code>nfs_client_enable</code>	是否使用 NFS 作为持久化存储，是：true；否：false
<code>nfs_client_is_default_class</code>	是否设定为默认存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认存储类型
<code>nfs_server</code>	允许其访问的 NFS 服务端地址，可以是 IP 或 Hostname
<code>nfs_path</code>	NFS 共享目录，即服务器上共享出去的文件目录，可参考 <a href="#">Kubernetes 官方文档</a>

## NFS in Kubernetes（仅限 multi-node 部署测试使用）

NFS 即网络文件系统，它允许网络中的计算机之间通过 TCP/IP 网络共享资源。本安装方法将会在 Kubernetes 集群内安装 [容器化的 NFS 服务端](#)，要求 Kubernetes 节点有足够的硬盘空间。在 `conf/vars.yml` 配置的释义如下。

NFS	Description
<code>nfs_in_k8s_enable</code>	是否部署 NFS Server 到当前集群作为存储服务端，是：true；否：false
<code>nfs_in_k8s_is_default_class</code>	是否设定 NFS 为默认存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认存储类型

## Local Volume (仅限 all-in-one 部署测试使用)

**Local Volume** 表示挂载的本地存储设备，如磁盘、分区或目录，仅支持在 all-in-one 模式安装时使用 Local Volume，由于该类型暂不支持动态分配，因此仅建议用于测试部署，方便初次安装和体验。在 conf/vars.yml 配置的释义如下。

Local Volume	Description
local_volume_provisioner_enabled	是否使用 local volume 作为持久化存储，是：true；否：false
local_volume_provisioner_storage_class	存储类型的名称，默认：local
local_volume_is_default_class	是否设定为默认存储类型，是：true；否：false 注：系统中存在多种存储类型时，只能设定一种为默认的存储类型

说明：在您配置好 Local Volume (只有 all-in-one 支持这类存储) 并成功安装 KubeSphere 后，可参阅 [Local Volume 使用方法](#)。

# 安装内置 Harbor

KubeSphere Installer 集成了 Harbor 的 Helm Chart (版本为 harbor-18.11.1)，内置的 Harbor 作为可选安装项，用户可以根据团队项目的需求来配置安装，方便用户对项目的镜像管理，仅需安装前在配置文件 `conf/vars.yml` 中简单配置即可，参考以下步骤安装和访问 Harbor。

## 修改安装配置文件

1、安装 KubeSphere 前，在 Installer 中的 `conf/vars.yml` 文件中，参考如下配置修改。

```
# harbor deployment
harbor_enable: true
harbor_domain: harbor.devops.kubesphere.local
```

2、修改后保存，然后执行安装脚本，即可通过 Helm Chart 的方式来安装 Harbor。

## 访问 Harbor

在 KubeSphere 中添加 Harbor 作为镜像仓库，请参考如下步骤，先配置 Docker 访问，然后在 KubeSphere 添加 Harbor 的用户认证信息。

### 配置 Docker 访问

#### 第一步：修改 Docker 配置

修改集群中 **所有节点** 的 Docker 配置，需要在 `/etc/systemd/system/docker.service.d/docker-options.conf` 文件添加字段 `--insecure-registry=harbor.devops.kubesphere.local:30280`，如下所示：

#### 配置文件修改示例

```
[Service]
```

```
Environment="DOCKER_OPTS= --registry-mirror=https://registry.docker-cn.com --data-root=/var/lib/docker"
```

注意：Environment 配置应写在一行，不能手动换行。

## 第二步：修改 hosts 配置

在本地的 `/etc/hosts` 文件中，需要参考如下添加一条记录：

```
192.168.0.24 harbor.devops.kubesphere.local
```

说明：192.168.0.24 是当前主机的内网 IP，请根据实际情况填写。若需要将 Harbor 服务暴露给集群外部用户使用，则需要在外网配置 DNS 记录（DNS 服务器处或者用户的本地 hosts 文件内），把域名 `harbor.devops.kubesphere.local` 对应到相应的外网 IP，并将 Harbor 端口 30280 进行端口转发和开放防火墙，确保外部流量可以通过该端口。

## 第三步：重启 Docker 服务

修改后，需要重启 **所有节点** 的 Docker 服务使配置生效，比如在 Linux 系统中需执行以下命令：

```
$ sudo systemctl daemon-reload
```

```
$ sudo systemctl restart docker
```

## 第四步：登录 Harbor

执行以下命令登录 Harbor 镜像仓库。关于如何制作镜像、打包上传镜像以及 Dockerfile 的使用，请参考 [Docker 官方文档](#)。

```
$ docker login -u admin -p Harbor12345 http://harbor.devops.kubesphere.local:30280
```

WARNING! Using --password via the CLI is insecure. Use --password-stdin.

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded

## KubeSphere 添加 Harbor

当后台配置 Docker 访问 Harbor 成功后，即可登录 KubeSphere，在已创建的企业空间的项目下，左侧菜单栏选择 **配置中心 → 密钥**，点击 **创建**。

### 第一步：填写基本信息

填写密钥的基本信息，完成后点击 **下一步**。

- 名称：起一个简洁明了的名称，填写 **harbor**
- 别名：支持中文，帮助您更好的区分资源，比如 **内置 Harbor 镜像仓库**
- 描述信息：简单介绍该密钥的功能

The screenshot shows the 'Create Key' dialog in KubeSphere. At the top left is the title '创建密钥'. On the right is a 'Edit Mode' switch. Below the title are two tabs: '基本信息' (selected) and '密钥设置'. The '基本信息' tab contains four input fields:

- '名称 \*': A text input field containing 'harbor'.
- '项目': A dropdown menu currently set to 'demo-namespace'.
- '别名': A text input field containing '内置 Harbor 镜像仓库'.
- '描述信息': A text input field containing 'This is a demo'.

Below these fields is a note: '最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾'.

## 第二步：填写 Harbor 仓库信息

2.1. 在类型中选择镜像仓库密钥，参考如下提示填写 Harbor 仓库的登录信息。

- 仓库地址：填写内置的 Harbor 镜像仓库域名和节点端口

`http://harbor.devops.kubesphere.local:30280`

- 用户名：admin
- 密码：Harbor12345
- 邮箱：填写个人邮箱

创建密钥

基本信息 密钥设置

密钥设置

类型  
镜像仓库密钥

仓库地址 \*  
http://harbor.devops.kubesphere.local:30280

用户名 \*  
admin

邮箱  
邮箱

密码 \*  
\*\*\*\*\*

2.2. 点击 **创建**，如果 Harbor 安装配置都正确，并且验证信息都填写无误，即可添加成功。Harbor 镜像仓库添加完成后，可以上传镜像和拉取镜像。

项目 demo-namespace

密钥

名称	类型	Config Number	创建时间	操作
harbor(内置 Harbor 镜像仓库)	镜像仓库密钥	1	2018-12-27 18:34:46	⋮
default-token-45rr8	kubernetes.io/service-account-token	3	2018-12-27 17:00:05	⋮

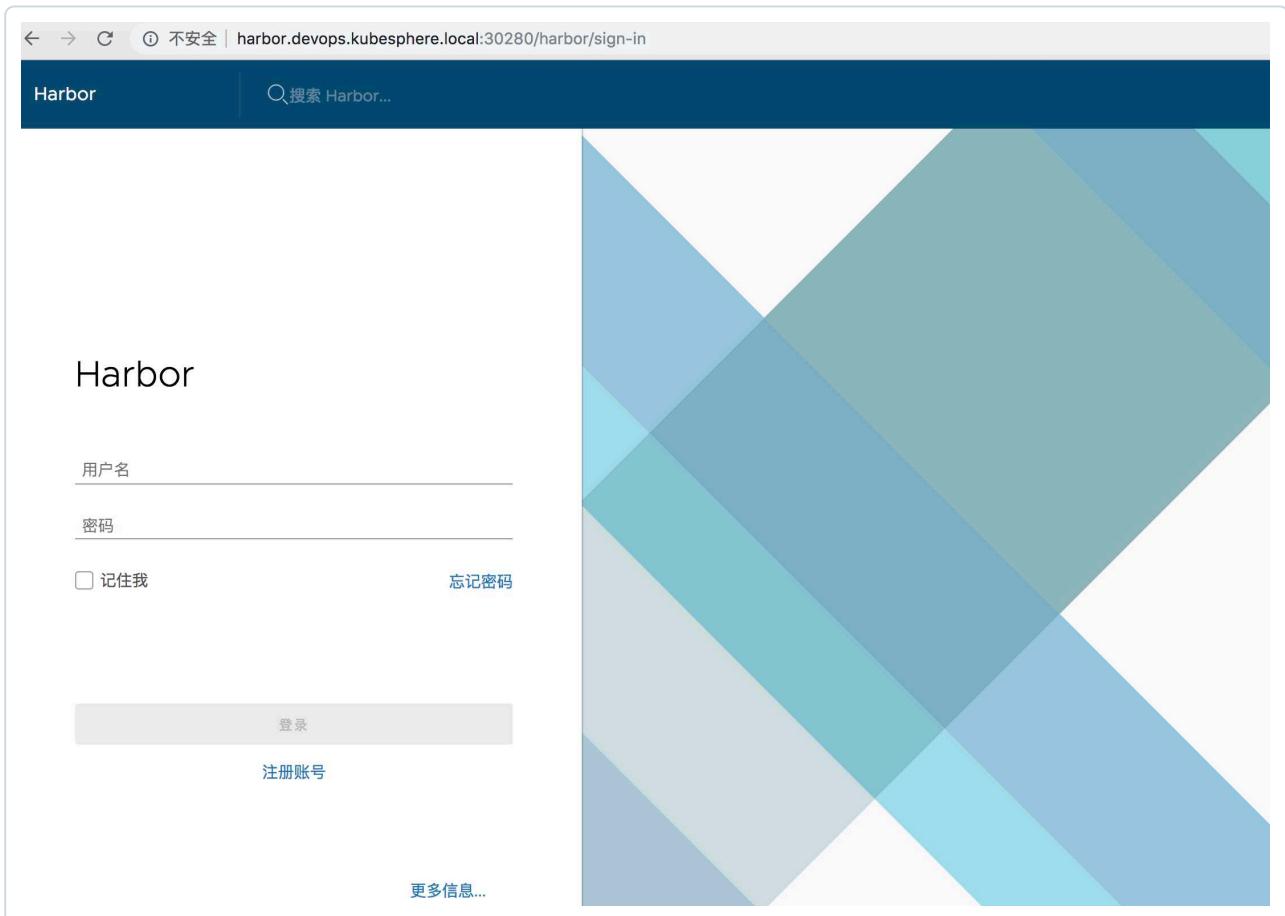
## 浏览器访问

KubeSphere 和 Harbor 都安装成功后，若需要在集群外部访问 Harbor，请在本地的 `/etc/hosts` 文件中参考如下示例添加一行记录，然后才可以在浏览器访问 Harbor 镜像仓库。

```
139.198.10.10 harbor.devops.kubesphere.local
```

**注意：**139.198.10.10 是 KubeSphere 集群的公网 IP，请根据实际情况填写。在外网访问 Harbor，需要绑定公网 IP 并配置端口转发，若公网 IP 有防火墙，请在防火墙添加规则放行对应的端口 30280，保证外网流量可以通过该端口，外部才能够访问。

Harbor 服务对外暴露的节点端口 (NodePort) 为 30280，内置的 Harbor 镜像仓库目前仅支持 http 协议，在浏览器中可以通过 `{$域名}:{$NodePort}` 如 `http://harbor.devops.kubesphere.local:30280` 访问 Harbor 登录页面。默认的管理员用户名和密码为 `admin / Harbor12345`，其它用户与 KubeSphere 的用户账户体系一致。关于 Harbor 的使用详见 [Harbor 官方文档](#)。



## 使用 Harbor 镜像仓库

若需要在 KubeSphere 中使用 Harbor 镜像仓库中的镜像，需要先将相关的镜像构建后上传至 Harbor。

关于 Harbor 的使用详见 [Harbor 官方文档](#)。

以创建 Deployment 为例展示如何使用镜像仓库来拉取仓库中的镜像。比如 Harbor 镜像仓库中已有

`mysql:5.6` 的 docker 镜像，在容器组模板中需要先选择镜像仓库，然后将镜像填写为

`http://harbor.devops.kubesphere.local:30280/mysql:5.6`，对应的格式为 `镜像仓库地址 / 镜像名`

`:tag`，填写后创建完成即可使用该 Harbor 镜像仓库中的镜像。

创建部署

编辑模式

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

◀ 添加容器

容器名称 \*

container-evf4j4

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

高级选项 ▾

镜像 \*

http://harbor.devops.kubesphere.local:30280/mysql:5.6

harbor  
http://harbor.devops.kubesphere.local:30280



# 集群节点扩容

安装 KubeSphere 后，在正式环境使用时可能会遇到服务器容量不足的情况，这时就需要添加新的服务器，然后将应用系统进行水平扩展来完成对系统的扩容。此时您可以根据实际业务需要对 Kubernetes 集群的 Node 节点进行扩容，KubeSphere 对于在 Kubernetes 集群中加入新 Node 是非常简单的，仅需简单两步即可完成集群节点扩容。节点扩容基于 Kubelet 的自动注册机制，新的 Node 将会自动加入现有的 Kubernetes 集群中。

需要说明的是，若以 all-in-one 模式安装的环境，集群默认存储为 local volume，不建议直接对其增加节点，增加节点适用于 multi-node 模式安装的环境。下面以 root 用户增加 node3 的配置为例。

## 第一步：修改主机配置文件

KubeSphere 支持混合扩容，即扩容新增的主机系统可以是 CentOS，也可以是 Ubuntu。当申请完新的主机后，在主机配置文件 `conf/hosts.ini` 根据需要增加的主机信息在 [all] 和 [kube-node] 部分对应添加一行参数，若扩容多台主机则依次添加多行参数。需要注意的是，扩容新的节点时不能修改原节点的主机名如 master、node1 和 node2，如下添加新节点 node3：

```
[all]
master ansible_connection=local ip=192.168.0.1
node1 ansible_host=192.168.0.2 ip=192.168.0.2 ansible_ssh_pass=PASSWORD
node2 ansible_host=192.168.0.3 ip=192.168.0.3 ansible_ssh_pass=PASSWORD
node3 ansible_host=192.168.0.4 ip=192.168.0.4 ansible_ssh_pass=PASSWORD
...
[kube-node]
master
node1
node2
node3
...
```

## 第二步：执行扩容脚本

在 “/script” 目录执行 `add-nodes.sh` 脚本。待扩容脚本执行成功后，即可看到包含新节点的集群节点信

息，可通过 KubeSphere 控制台的菜单选择 **基础设施** 然后进入 **主机管理** 页面查看，或者通过 Kubectl 工具执行 `kubectl get node` 命令，查看扩容后的集群节点详细信息。

```
$ ./add-nodes.sh
```

## 停用集群节点

同样，若需要停用或隔离集群中的节点，比如在硬件升级、硬件维护等情况下需要将某些 Node 进行隔离，集群管理员可以在 KubeSphere 控制台菜单选择 **基础设置** → **主机管理** 页面执行停用主机，可参考主机管理说明的 [停用或启用主机](#)。

## 卸载

注意：卸载将从机器中删除 KubeSphere，该操作不可逆，也不会备份任何数据，请谨慎操作。

如需卸载 KubeSphere，需执行以下步骤：

1、在 ”/scripts” 目录下，以 `root` 执行 `uninstall.sh` 脚本：

```
$ ./uninstall.sh
```

2、如果确认卸载，在以下问题返回时输入 `yes`：

```
Are you sure you want to reset cluster state? Type 'yes' to reset your cluster. [No]:
```

# 入门必读

相比较易捷版，KubeSphere 高级版提供了更强大更灵活的功能，满足企业更复杂的业务需求，比如新增 HPA（水平自动伸缩）、CI/CD、监控日志、LDAP 集成、多租户管理、Job/CronJob 等功能，以及 Secrets、ConfigMaps 配置管理。

本文旨在通过几个快速入门的示例帮助您了解 KubeSphere 容器平台的基本使用流程，带您快速上手 KubeSphere。快速入门包含六个示例，建议参考文档示例的步骤实践操作一遍。为了帮助用户更快地了解 KubeSphere 并进一步理解底层 Kubernetes 的基础概念和知识，我们还精心准备了 [Kubernetes 入门视频教程](#)。

## 管理员快速入门

若您是初次安装使用 KubeSphere 的集群管理员用户，请先参考 [管理员入门](#)，将引导新手用户创建企业空间、创建新的账户角色并邀请加入，它是创建工作负载和 DevOps 工程的前提条件。

## 工作负载快速入门

- [示例一 – 部署 MySQL](#)

本文以创建一个有状态副本集 (statefulset) 为例，使用 `Mysql:5.6` 镜像部署一个有状态的 MySQL 应用，作为 [Wordpress](#) 网站的后端。示例二依赖于示例一，请按顺序完成这两个示例。

- [示例二 – 部署 Wordpress](#)

本文以创建一个部署 (deployment) 为例，使用 `Wordpress:4.6–apache` 镜像部署一个无状态的 Wordpress 应用，最终可通过公网访问的 [Wordpress](#) 网站，其后端为示例一所演示的 MySQL 应用。

- [示例三 – 创建简单任务](#)

任务 (Job) 是 Kubernetes 中用来控制批处理型任务的资源对象，定时任务 (CronJob) 是基于时间的任务，可以定时地执行 Job，当您熟悉了任务的使用示例，那么上手定时任务也就不是一件难事了。本文以创建一个并行任务去执行简单的命令计算并输出圆周率到小数点后 2000 位作为示例，说明任务的基本功能。

- [示例四 – 一键部署应用](#)

一键部署应用基于 KubeSphere 应用模板，部署的应用一般包含相应的工作负载和服务，可通过 [应用](#) 列表查看，用户可以配置外网访问方式来访问该应用。

- [示例五 – 设置弹性伸缩 \(HPA\)](#)

弹性伸缩 (HPA) 是高级版独有的功能，支持 Pod 的水平自动伸缩，本示例以文档和视频的方式演示平台中如何设置 Pod 水平自动伸缩的功能。

## DevOps 工程快速入门

- [示例六 – Jenkinsfile in SCM](#)

本示例以文档和视频演示如何通过 GitHub 仓库中的 Jenkinsfile 来创建 CI/CD 流水线，包括拉取代码、单元测试、构建镜像、推送和发布版本，最终将一个文档网站部署到 KubeSphere 集群中的开发环境和产品环境，并且能够通过公网访问。

- [示例七 – Jenkinsfile out of SCM](#)

本示例以文档和视频演示如何基于 [示例六 – Jenkinsfile in SCM](#)，以可视化的方式构建 CI/CD 流水线（包含示例六的前六个阶段），最终将本文档网站部署到 KubeSphere 集群中的开发环境且能够通过公网访问。

# 管理员快速入门

## 目的

本文档面向初次使用 KubeSphere 的集群管理员用户，引导新手用户创建企业空间、创建新的角色和账户，然后邀请新用户进入企业空间后，创建项目和 DevOps 工程，并引导管理员使用 web Kubectl 工具，帮助用户熟悉多租户下的用户和角色管理，以及 web Kubectl 工具的使用，快速上手 KubeSphere。

## 前提条件

已安装 KubeSphere，并使用默认的 admin 用户名和密码登录了 KubeSphere。

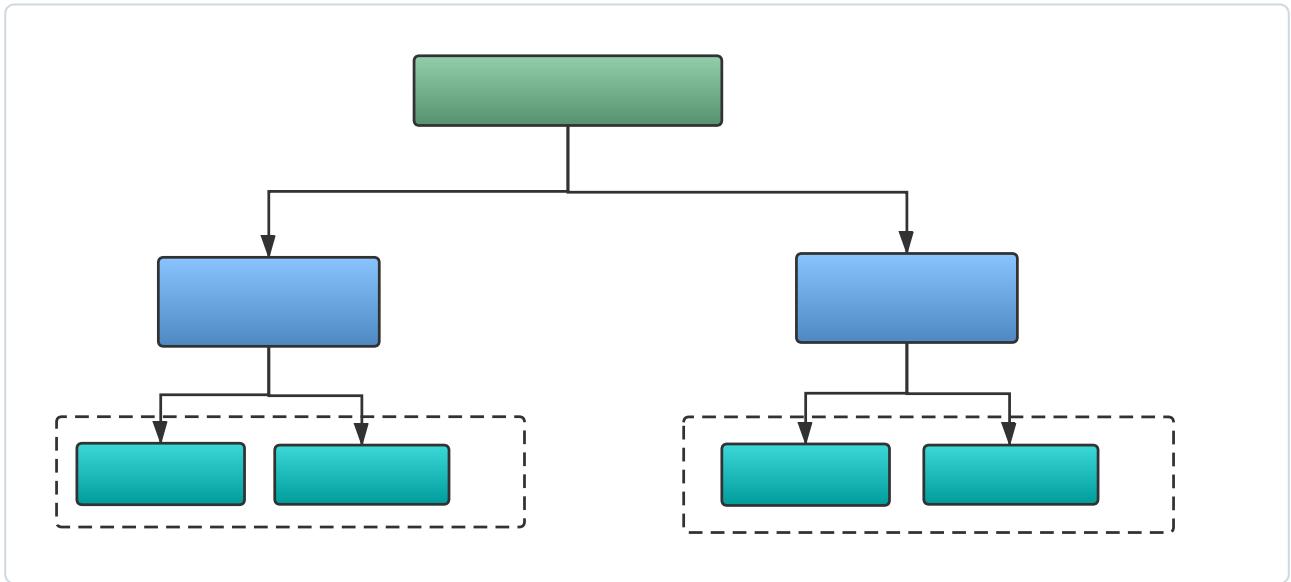
## 预估时间

约 15 分钟。

## 操作示例

### 示例视频

目前，平台的资源层级包括 Cluster, Workspace, Project 和 DevOps 工程，层级关系如下图所示，在每个层级中，每个组织中都有多个不同的内置角色。



## 集群管理员

### 第一步：创建角色和账号

平台中的 cluster-admin 角色可以为其他用户创建账号并分配平台角色，平台内置了集群层级的以下三个常用的角色，同时支持自定义新的角色。

- cluster-admin
- cluster-regular
- workspaces-manager

本示例首先新建一个角色 (user-manager)，为该角色授予账号管理和角色管理的权限，然后新建一个账号并给这个账号授予 user-manager 角色。

账号名	集群角色	职责
user-manager	user-manager	管理集群的账户和角色

1.1. 点击控制台左上角 **平台管理 → 平台角色**，可以看到当前的角色列表，点击 **创建**，创建一个角色用于管理所有账户和角色。



### 1.2. 填写角色的基本信息和权限设置。

- 名称：起一个简洁明了的名称，便于用户浏览和搜索，如 `user-manager`
- 描述信息：简单介绍该角色的职责，如 `管理账户和角色`

创建平台角色

基本信息

名称 \*  
user-manager  
最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

描述信息  
管理账户和角色

### 1.3. 权限设置中，勾选账户管理和角色管理的所有权限，点击 **创建**。

基本信息

权限设置

账户管理

查看  创建  编辑  删除

角色管理

查看  创建  编辑  删除

### 1.4. 点击控制台左上角 **平台管理 → 账号管理**，可以看到当前集群中所有用户的列表，点击 **创建** 按钮。



1.5. 填写新用户的基本信息，如用户名设置为 `user-manager`，角色选择 `user-manager`，其它信息可自定义，点击 确定。

说明：上述步骤仅简单地说明创建流程，关于账号管理与角色权限管理的详细说明，请参考 [角色权限概览](#) 和 [账号管理](#)。

 添加用户 X

用户名 \*

用户名只能包含小写字母及数字

邮箱 \*

邮箱地址可以方便项目管理员及时的添加您为项目成员

角色

▼

角色类型根据权限范围分为集群和项目两类，当前角色的授权范围为整个集群。

密码 \*

密码必须包含数字和字母，长度至少为 6 位

描述信息

取消 确定

1.6. 然后用 `user-manager` 来创建下表的一个 `workspace-manager` 和三个 `cluster-regular` 角色的账号，`workspace-manager` 将用于创建一个企业空间，并指定其中一个用户名为 `ws-admin` 作为企业空间管理员。切换成上一步创建的 `user-manager` 账号登录 KubeSphere，在 [账号管理](#) 下，新建四个账号，创建步骤同上，参考如下信息创建。

账号名	集群角色	企业空间角色	职责
ws–manager	workspaces–manager	workspace–admin (默认)	创建和管理企业空间
ws–admin	cluster–regular	workspace–admin	管理企业空间下所有的资源 (本示例用于邀请新成员加入企业空间)
project–admin	cluster–regular	workspace–regular	创建和管理项目、DevOps 工程，邀请新成员加入
project–regular	cluster–regular	workspace–regular	将被 project–admin 邀请加入项目和 DevOps 工程， 用于创建项目和工程下的工作负载、Pipeline 等资源

### 1.7. 查看新建的四个账号信息。

The screenshot shows the KubeSphere Platform Management interface with the 'Account Management' page selected. The page displays a list of accounts with the following details:

名称	状态	平台角色	最近登录
admin admin@kubesphere.io	活跃	cluster-admin	2018-12-19 22:39:19
user–manager user–manager@kubesphere.io	活跃	user–manager	2018-12-19 22:39:35
ws–manager ws–manager@kubesphere.io	活跃	workspaces–manager	2018-12-19 20:31:13
ws–admin ws–admin@kubesphere.io	活跃	cluster–regular	2018-12-19 20:48:26
project–admin project–admin@kubesphere.io	活跃	cluster–regular	尚未登录
project–regular project–regular@kubesphere.io	活跃	cluster–regular	尚未登录

## 企业空间管理员

### 第二步：创建企业空间

企业空间 (workspace) 是 KubeSphere 实现多租户模式的基础，是用户管理项目、DevOps 工程和企业成员的基本单位。

2.1. 切换为 `ws-manager` 登录 KubeSphere, `ws-manager` 有权限查看和管理平台的所有企业空间。

点击 `企业空间`, 可见新安装的环境只有一个系统默认的企业空间 `system-workspace`, 用于运行 KubeSphere 平台相关组件和服务, 禁止删除该企业空间。

在企业空间列表点击 `创建`:

The screenshot shows the KubeSphere workspace management interface. At the top, there's a header with the KubeSphere logo and a dropdown for 'ws-manager'. Below the header, the title '企业空间' (Workspaces) is displayed. A sub-header explains that workspaces are logical units for organizing projects and DevOps engineering. A search bar and a '创建' (Create) button are on the right. The main area shows a table with one row for 'system-workspace'. The table columns include: workspace icon, name ('system-workspace'), project count ('7'), DevOps engineering count ('0'), administrator ('admin'), and creation time ('2018-12-17 09:21:55'). Below the table, there's a '成员:' (Members) section with a user icon.

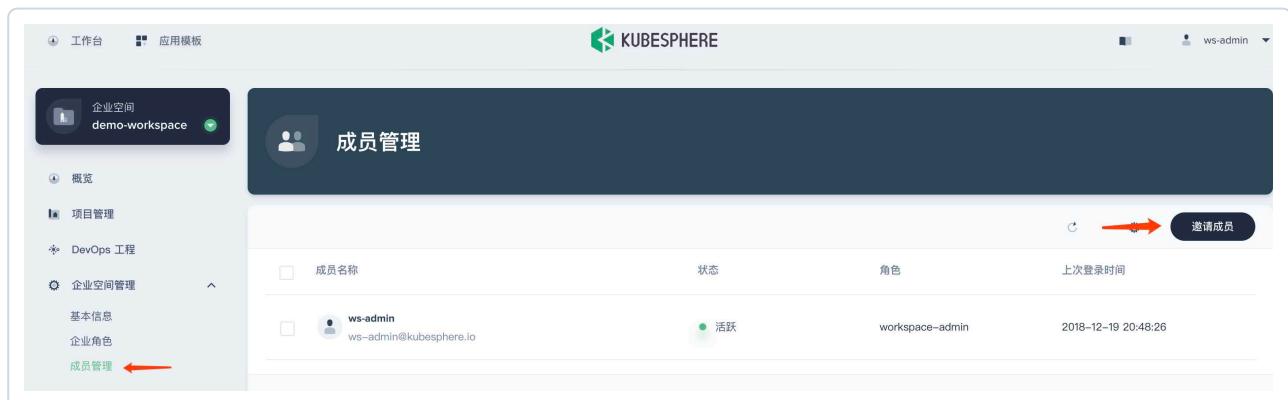
2.2. 参考如下提示填写企业空间的基本信息, 然后点击 `确定`。企业空间的创建者同时默认为该企业空间的管理员 (`workspace-admin`), 拥有企业空间的最高管理权限。

- 企业空间名称: 请尽量保持企业名称简短, 便于用户浏览和搜索, 本示例是 `demo-workspace`
- 企业空间管理员: 可从当前的集群成员中指定, 这里指定上一步创建的 `ws-admin` 用户为管理员, 相当于同时邀请了 `ws-admin` 用户进入该企业空间
- 描述信息: 简单介绍该企业空间

The screenshot shows the 'Create Workspace' form. The title is '创建工作空间' (Create Workspace). A sub-header provides a brief explanation of what a workspace is. The form has three main sections: '基本信息' (Basic Information), '企业空间名称 \*' (Workspace Name \*), and '企业空间管理员' (Workspace Administrator). In the '基本信息' section, there's also a '描述信息' (Description) field and a note about character limits. The '企业空间名称' field contains 'demo-workspace', and the '企业空间管理员' dropdown is set to 'ws-admin'.

说明：企业空间管理的详细说明请参考 [企业空间管理](#)。

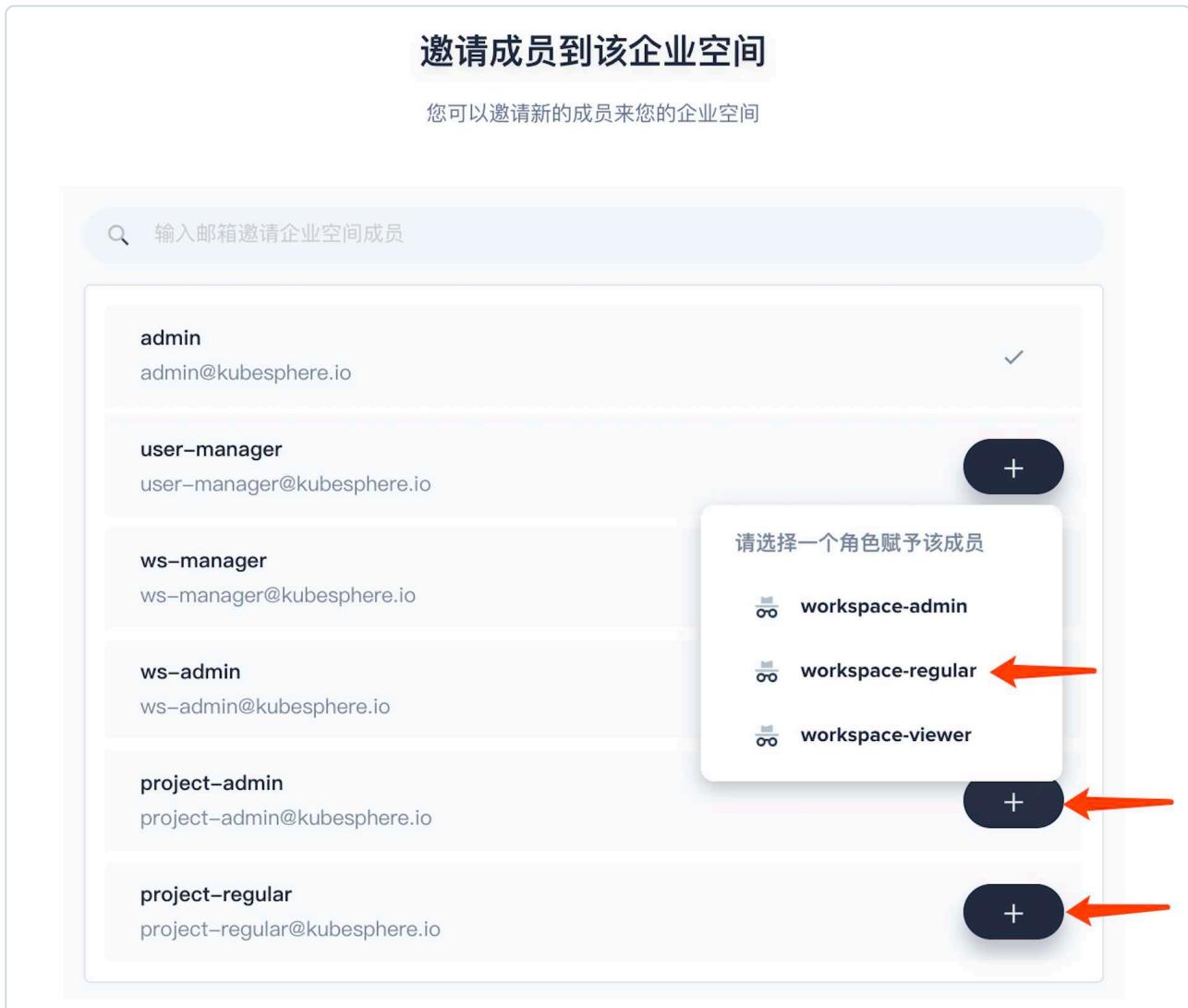
2.3. 企业空间 `demo-workspace` 创建完成后，切换为 `ws-admin` 登录 KubeSphere，如果用户只有一个企业空间，登录进去后会直接进入该企业空间，可看到该企业空间的概览页。`ws-admin` 可以从集群成员中邀请新成员加入当前企业空间，然后创建项目和 DevOps 工程。在左侧菜单栏选择 **企业空间管理** → **成员管理**，点击 **邀请成员**。



The screenshot shows the KubeSphere web interface with the following details:

- Left Sidebar (Enterprise Space Demo):** Includes links for Overview, Project Management, DevOps Engineering, Enterprise Space Management (selected), Basic Information, Corporate Roles, and Member Management (highlighted with a red arrow).
- Header:** KUBESPHERE logo and ws-admin user information.
- Current Space:** 企业空间 demo-workspace
- Main Content - Member Management:** Displays a table with columns: 成员名称 (Member Name), 状态 (Status), 角色 (Role), and 上次登录时间 (Last Login Time). One row is shown for ws-admin, which is active (green dot) and has the role workspace-admin, last logged in on 2018-12-19 20:48:26.
- Top Right:** Invite Member button (highlighted with a red arrow).

2.4. 这一步需要邀请在 **步骤 1.6.** 创建的两个用户 `project-admin` 和 `project-regular` 进入企业空间，并且都授予 `workspace-regular` 的角色。



## 项目和 DevOps 工程管理员

### 第三步：创建项目

创建工作负载、服务和 CI/CD 流水线等资源，需要先创建好项目和 DevOps 工程。

3.1. 上一步将用户 **project-admin** 邀请进入企业空间后，可切换为 **project-admin** 账号登录 KubeSphere，默认进入 **demo-workspace** 企业空间下，点击 **创建**，选择 **创建资源型项目**。



3.2. 填写项目的基本信息和高级设置，完成后点击 **下一步**。

## 基本信息

- 名称：为项目起一个简洁明了的名称，便于用户浏览和搜索，比如 `demo-namespace`
- 别名：帮助您更好的区分资源，并支持中文名称，比如 `示例项目`
- 描述信息：简单介绍该项目

This screenshot shows the 'Basic Information' configuration page. At the top, there are tabs for '基本信息' (Basic Information) and '高级设置' (Advanced Settings), with '基本信息' being active. Below the tabs, it says '项目基础信息设置' (Project basic information settings). The '名称' (Name) field is filled with 'demo-namespace'. The '别名' (Alias) field is filled with '示例项目'. A note below the alias field states: '别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。' (Aliases can consist of any character, helping you better distinguish resources, and support Chinese names.). The '描述信息' (Description) field contains the text 'This is a demo'. There is also a note above the description field: '最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾' (Maximum 63 characters, can only contain lowercase letters, numbers, and hyphens, and must start and end with lowercase letters or numbers).

## 高级设置

3.3. 此处可使用默认的请求值 (requests) 与限额值 (limits)，在项目使用过程中可根据实际情况再次编辑资源默认请求。高级设置是在当前项目中配置容器默认的 CPU 和内存的请求与限额，相当于是给项目创建了一个 Kubernetes 的 LimitRange 对象，在项目中创建工作负载后填写容器组模板时，若不填写容器 CPU 和内存的请求与限额，则容器会被分配在高级设置的默认 CPU 和内存请求与限额值。

完成高级设置后，点击 **创建**。



说明：项目管理和设置的详细说明，请参考用户指南下的项目设置系列文档。

3.4. 示例项目 demo-namespace 创建成功后，点击进入示例项目。在 **步骤 2.4. 已邀请用户 project-regular 加入了当前企业空间 demo-workspace**，下一步则需要邀请 project-regular 进入该企业空间下的项目 demo-namespace。点击项目列表中的 demo-namespace 进入该项目。

demo-workspace

1 参与项目 1 创建的项目 0 DevOps 工程

创建

列表 (1)

demo-namespace(示例项目)  
This is a demo

project-admin  
项目管理员  
2018-12-19 23:26:40  
创建时间

成员: 1人

3.5. 在项目的左侧菜单栏选择 **项目设置 → 项目成员**，点击 **邀请成员**。

The screenshot shows the KubeSphere project management interface. On the left, there's a sidebar with various project settings like Overview, Applications, Workload, Storage, Network & Services, Configuration Center, Project Settings, and External Network Access. The 'Project Members' option is highlighted with a red arrow. The main content area displays a table for 'demo-namespace (示例项目)'. The table has columns for Member Name, Status, Role, and Last Login Time. It shows one entry: 'project-admin' (status: active, role: admin, last login: never). A red arrow points to the 'Invite Member' button at the top right of the table.

3.6. 在弹窗中的 `project-regular` 点击 “+”，在项目的内置角色中选择 `operator` 角色。因此，后续在项目中创建和管理资源，都可以由 `project-regular` 用户登录后进行操作。

The screenshot shows the 'Invite Member' dialog. At the top, it says '邀请成员到该项目' and '您可以邀请新的成员来协助您的项目'. Below is a search bar labeled '输入邮箱邀请项目成员'. A list of users is shown: 'ws-admin' (ws-admin@kubesphere.io), 'ws-manager' (ws-manager@kubesphere.io), 'admin' (admin@kubesphere.io), 'project-admin' (project-admin@kubesphere.io), and 'project-regular' (project-regular@kubesphere.io). To the right of each user is a '+' button. A modal dialog is open for 'project-regular', titled '请选择一个角色赋予该成员'. It lists three roles: 'admin', 'viewer', and 'operator'. The 'operator' role is highlighted with a red arrow. Another red arrow points to the '+' button at the bottom right of the modal.

## 第四步：创建 DevOps 工程

4.1. 继续使用 `project-admin` 用户创建 DevOps 工程。点击 **工作台**，在当前企业空间下，点击 **创建**，在弹窗中选择 **创建一个 DevOps 工程**。DevOps 工程的创建者 `project-admin` 将默认为该工程的 Owner，拥有 DevOps 工程的最高权限。



4.2. 输入 DevOps 工程的名称和描述信息，比如名称为 `demo-devops`。点击 **创建**，注意创建一个 DevOps 有一个初始化环境的过程需要几秒钟。



创建 DevOps 工程

### 基本信息

请输入 DevOps 工程的基本信息

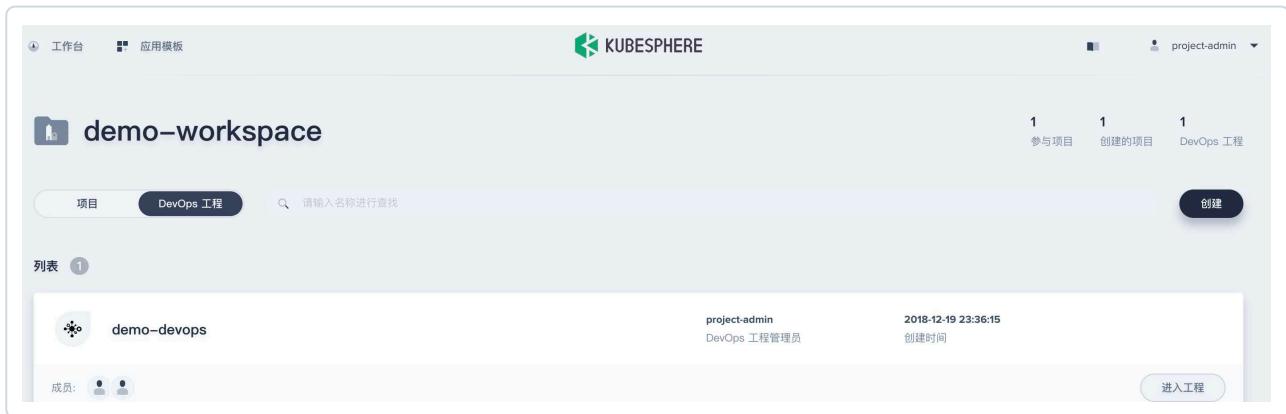
名称 \*

最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

描述信息

说明：DevOps 工程管理的详细说明请参考 [管理 DevOps 工程](#)。

4.3. 点击 DevOps 工程列表中的 `demo-devops` 进入该工程的详情页。



The screenshot shows the KubeSphere DevOps workspace interface. At the top, there are navigation tabs for '工作台' (Workstation) and '应用模板' (Application Templates). The title 'KUBESPHERE' is centered above the workspace list. Below the title, it says 'demo-workspace'. On the right side, there are statistics: 1 参与项目 (1 participating project), 1 创建的项目 (1 created project), and 1 DevOps 工程 (1 DevOps project). A '创建' (Create) button is also present. The main area displays a list of projects, with 'demo-devops' being the first item. It includes columns for the project name, creator ('project-admin'), and creation time ('2018-12-19 23:36:15'). Below the list, there is a '成员' (Members) section with two user icons and a '进入工程' (Enter Project) button.

4.4. 同上，这一步需要在 `demo-devops` 工程中邀请用户 `project-regular`，并设置角色为 `maintainer`，用于对工程内的 Pipeline、凭证等创建和配置等操作。菜单栏选择 **工程管理** → **工程成员**，然后点击 **邀请成员**，为用户 `project-regular` 设置角色为 `maintainer`。后续在 DevOps 工程中创建 Pipeline 和凭证等资源，都可以由 `project-regular` 用户登录后进行操作。



The screenshot shows the KubeSphere DevOps workspace interface, specifically the 'demo-devops' project details page. On the left, there is a sidebar with options: '流水线' (Pipeline), '工程管理' (Engineering Management), '基本信息' (Basic Information), '凭证' (Certificates), '成员角色' (Member Roles), and '工程成员' (Project Members). A red arrow points to the '工程成员' option. The main content area shows the project details: 'demo-devops', 'project-admin' (工程管理员), 'demo-workspace' (企业空间), and '2018-12-19 23:36:15' (创建时间). Below this, there is a table for managing members:

成员名称	状态	角色
admin	已启用	owner
project-admin	已启用	owner

A '邀请成员' (Invite Member) button is located at the top right of the member management section. The entire interface has a light blue header bar.



## 使用 web kubectl

kubectl 是用于操作 Kubernetes 集群的命令行接口。

KubeSphere 在界面提供了 web kubectl 方便用户使用，默认情况下，仅集群管理员 (cluster-admin) 拥有 web kubectl 的使用权限，可以直接使用 kubectl 命令行操作和管理集群资源。

集群管理员登录后，点击右下角的 web kubectl 按钮，即可打开 web kubectl 窗口

在 web kubectl 可输入 kubectl 命令查询或管理 Kubernetes 集群资源。另外，web kubectl 窗口支持查看当前集群的 Kubeconfig 文件。

例如，执行以下命令查询集群中所有 PVC 的挂载情况：

```
kubectl get pvc --all-namespaces
```

```
/ # kubectl get pvc --all-namespaces
NAME                                     STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
kubespHERE-devops-system   data-ks-harbor-harbor-redis-0   Bound    pvc-df9cca23091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   database-data-ks-harbor-harbor-database-0   Bound    pvc-df91e092091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-gitlab-minio   Bound    pvc-07f2e43e091411e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-gitlab-postgresql   Bound    pvc-07f3575e091411e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-gitlab-redis   Bound    pvc-07f46557091411e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-harbor-harbor-chartmuseum   Bound    pvc-df455486091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-harbor-harbor-jobservice   Bound    pvc-df463fa091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-harbor-harbor-registry   Bound    pvc-df46dfdc091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   ks-jenkins   Bound    pvc-e2e7ce80091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-devops-system   repo-data-ks-gitlab-gitaly-0   Bound    pvc-087f99fc091411e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-monitoring-system   prometheus-k8s-db-prometheus-k8s-0   Bound    pvc-fbcbb540b091311e9   20Gi     RWO          csi-qingcloud   12d
kubespHERE-system   openldap-pvc   Bound    pvc-50f394ce091311e9   10Gi      RWO          csi-qingcloud   12d
kubespHERE-system   redis-pvc   Bound    pvc-50f5a3f7091311e9   10Gi      RWO          csi-qingcloud   12d
openpitrix-system   openpitrix-db-pvc   Bound    pvc-453996c1091311e9   20Gi     RWO          csi-qingcloud   12d
openpitrix-system   openpitrix-etcd-pvc   Bound    pvc-4575c424091311e9   20Gi     RWO          csi-qingcloud   12d
openpitrix-system   openpitrix-minio-pvc   Bound    pvc-45b0d095e091311e9   20Gi     RWO          csi-qingcloud   12d
/ #
```

从 kubectl 终端窗口可使用以下语法运行 kubectl 命令：

```
kubectl [command] [TYPE] [NAME] [flags]
```

**说明： 其中 command, TYPE, NAME, 和 flags 分别是：**

- command: 指定要在一个或多个资源进行操作，例如 create, get, describe, delete。
- TYPE: 指定资源类型。资源类型区分大小写，您可以指定单数，复数或缩写形式。
- NAME: 指定资源的名称。名称区分大小写。如果省略名称，则会显示所有资源的详细信息，比如 `kubectl get pods`。
- flags: 指定可选标志。例如，您可以使用 -s 或 --serverflags 来指定 Kubernetes API 服务器的地址和端口。重要提示：从命令行指定的标志将覆盖默认值和任何相应的环境变量。

**如果需要查询常用命令帮助，可在窗口运行 `kubectl help`，或参阅 [Kubernetes 官方文档](#)。**

至此，本文档为您演示了如何在多租户的基础上，使用账户管理和角色管理的功能，以示例的方式介绍了常用内置角色的用法，以及创建项目和 DevOps 工程，并且说明了如何使用 web kubectl。请参考 [快速入门](#) 系列文档的 7 个示例，使用 project-regular 用户登录 KubeSphere 动手实践操作一遍，创建工作负载、服务和 CI / CD 流水线。

# 示例——部署 MySQL 有状态应用

## 目的

本文以创建一个有状态副本集 (Statefulset) 为例，使用 [Mysql:5.6](#) 镜像部署一个有状态的 MySQL 应用，作为 [Wordpress](#) 网站的后端，演示如何创建使用 Statefulset。本示例的 MySQL 初始密码将以 [密钥 \(Secret\)](#) 的方式进行创建和保存。为方便演示，本示例仅说明流程，关于参数和字段的详细释义参见 [密钥](#) 和 [有状态副本集](#)。

## 前提条件

- 已创建了企业空间和项目，若还未创建请参考 [管理员快速入门](#)
- 以上一篇文档创建的 [project-regular](#) 用户登录 KubeSphere，进入已创建的企业空间下的项目

## 预估时间

约 10 分钟。

## 操作示例

### 示例视频

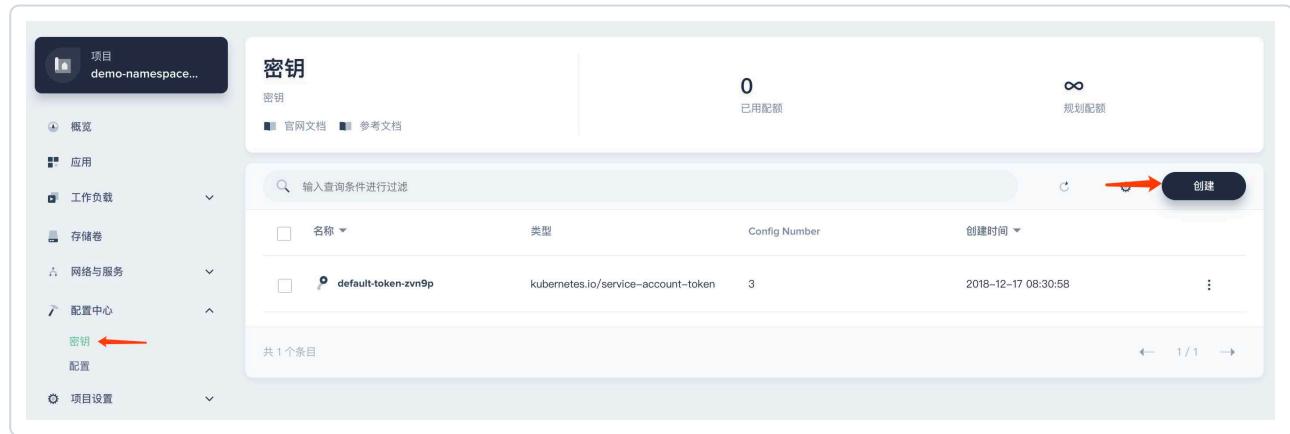
### 部署 MySQL

#### 第一步：创建密钥

MySQL 的环境变量 [MYSQL\\_ROOT\\_PASSWORD](#) 即 root 用户的密码属于敏感信息，不适合以明文的方式表现在步骤中，因此以创建密钥的方式来代替该环境变量。创建的密钥将在创建 MySQL 的容器组

设置时作为环境变量写入。

### 1.1. 在当前项目下左侧菜单栏的 配置中心 选择 密钥，点击 创建。



The screenshot shows the KubeSphere Configuration Center interface. On the left, there's a sidebar with various project and cluster management options. The 'Secrets' option under the 'Configuration Center' section is highlighted with a red arrow. The main area is titled 'Secrets' and displays a table of existing secrets. At the top right of this table, there's a large blue 'Create' button with a white outline, also highlighted by a red arrow. The table has columns for Name, Type, Config Number, and Create Time.

### 1.2. 填写密钥的基本信息，完成后点击 下一步。

- 名称：作为 MySQL 容器中环境变量的名称，可自定义，例如 `mysql-secret`
- 别名：别名可以由任意字符组成，帮助您更好的区分资源，例如 `MySQL 密钥`
- 描述信息：简单介绍该密钥，如 `MySQL 初始密码`



The screenshot shows the 'Create Secret' form. At the top, there are tabs for '基本信息' (Basic Information) and '密钥设置' (Key Settings), with '基本信息' being the active tab. Below the tabs, there are two main sections: '基本信息' and '项目'. In the '基本信息' section, the '名称' field is filled with 'mysql-secret'. There is a note below it stating: '最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾。' In the '项目' section, the 'demo-namespace' project is selected. Below these fields, there is a note: '将根据项目进行资源进行分组，可以按项目对资源进行查看管理。' In the '密钥设置' section, there is a '类型' dropdown set to '默认 (Opaque)' and a 'Data' section containing key-value pairs: 'MYSQL\_ROOT\_PASSWORD' and '123456'.

### 1.3. 密钥设置页，填写如下信息，完成后点击 创建。

- 类型：选择 `默认 (Opaque)`
- Data：Data 键值对填写 `MYSQL_ROOT_PASSWORD` 和 `123456`

创建密钥

编辑模式

基本信息 密钥设置

密钥设置

类型  
默认

可以选择也可以自定义一个密钥类型

Data \*

MYSQL\_ROOT\_PASSWORD 123456

Add data



## 第二步：创建有状态副本集

在左侧菜单栏选择 **工作负载 → 有状态副本集**，然后点击 **创建有状态副本集**。



## 第三步：填写基本信息

基本信息中，参考如下填写，完成后点击 **下一步**。

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，例如填写 `wordpress-mysql`
- 别名：可选，支持中文帮助更好的区分资源，例如填写 `MySQL 数据库`
- 描述信息：简单介绍该工作负载，方便用户进一步了解

创建有状态副本集

编辑模式

基本信息  容器组模板  存储卷模板  服务配置  标签设置  节点选择器

### 基本信息

您可以给有状态副本起一个名字，以便在使用的时候容易区分。

名称 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目

将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

## 第四步：容器组模板

4.1. 点击 **添加容器**，填写容器组设置，名称可由用户自定义，镜像填写 `mysql:5.6`（应指定镜像版本号），CPU 此处暂不作限定，将使用在创建项目时指定的默认请求值，内存的 **最大使用** 设置为 1024 Mi（即 1 Gi）。

展开 高级选项。

创建有状态副本集

编辑模式

基本信息  容器组模板  存储卷模板  服务配置  标签设置  节点选择器

### 添加容器

容器名称 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

镜像 \*

要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU

最小使用  m

最大使用  m

作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m

内存

最小使用  Mi

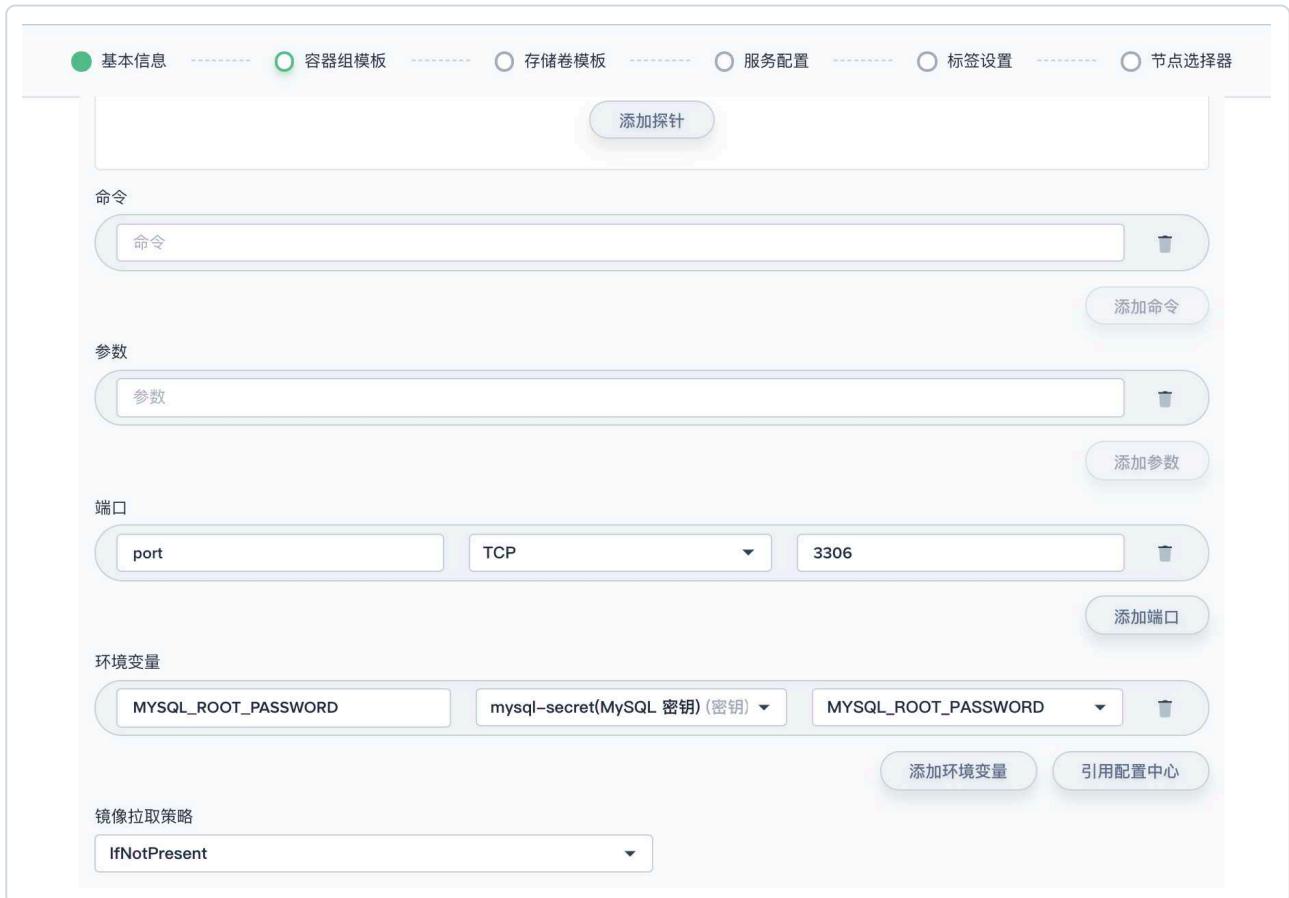
最大使用  Mi

作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项

4.2. 对 **端口** 和 **环境变量** 进行设置，其它项暂不作设置，完成后点击 **保存**。

- 端口：名称可自定义，选择 **TCP** 协议，填写 MySQL 在容器内的端口 **3306**。
- 环境变量：点击 **引用配置中心**，名称填写 **MYSQL\_ROOT\_PASSWORD**，选择在第一步创建的密钥 **mysql-secret (MySQL 密钥)** 和 **MYSQL\_ROOT\_PASSWORD**



4.3. 更新策略保持默认的 **滚动更新**，Partition 默认为 0，点击 **下一步**。

## 第五步：添加存储卷模板

容器组模板完成后点击 **下一步**，在存储卷模板中点击 **添加存储卷模板**。有状态应用的数据需要存储在持久化存储卷 (PVC) 中，因此需要添加存储卷来实现数据持久化。参考下图填写存储卷信息，完成后点击 **保存**。添加存储卷模板完成后，点击 **下一步**。

- 存储卷名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，此处填写 **mysql-pvc**
- 描述信息：简单介绍该存储卷，方便用户进一步了解，如 **MySQL 存储卷**
- 存储类型：选择集群已有的存储类型，如 **Local**
- 容量和访问模式：容量默认 **10 Gi**，访问模式选择 **ReadWriteOnce (单个节点读写)**

- 挂载路径：存储卷在容器内的挂载路径，选择 **读写**，路径填写 `/var/lib/mysql`

The screenshot shows the 'Volume' configuration page in KubeSphere. The 'Mount Path' field is highlighted in green, indicating it has been correctly set to `/var/lib/mysql`.

## 第六步：服务配置

一个 Kubernetes 的服务 (Service) 是一种抽象，它定义了一类 Pod 的逻辑集合和一个用于访问它们的策略 – 有的时候被称之为微服务。因此要将 MySQL 应用暴露给外部访问，需要创建服务，参考以下截图完成参数设置，完成后点击 **下一步**。

- 服务名称：此处填写 `mysql-service`，注意，这里定义的服务名称将关联 Wordpress，因此在创建 Wordpress 添加环境变量时应填此服务名
- 会话亲和性：默认 `None`
- 端口：名称可自定义，选择 `TCP` 协议，MySQL 服务的端口和目标端口都填写 `3306`，其中第一个端口是需要暴露出去的服务端口，第二个端口（目标端口）是容器端口

**说明：**若有实现基于客户端 IP 的会话亲和性的需求，可以在会话亲和性下拉框选择 "ClientIP" 或在代码模式将 `service.spec.sessionAffinity` 的值设置为 "ClientIP"（默认值为 "None"），该设置可将来自同一个 IP 地址的访问请求都转发到同一个后端 Pod。

创建有状态副本集

编辑模式

基本信息 容器组模板 存储卷模板 服务配置 标签设置 节点选择器

### 服务配置

集群不为服务生成IP，集群内部通过服务的后端Endpoint IP直接访问服务。此类型适合后端异构的服务，比如需要区分主从的服务。

服务名称 \* mysql-service 会话亲和性 None

最长253个字符，只能包含小写字母、数字及分隔符“-”，且必须以小写字母或数字开头及结尾

端口 \*

port	TCP	3306	3306	<input type="button" value="删除"/>
------	-----	------	------	-----------------------------------



## 第七步：标签设置

为方便识别此应用，我们标签设置为 `app: wordpress-mysql`。下一步的节点选择器可以指定容器组调度到期望运行的节点上，此处暂不作设置，直接点击 **创建**。

创建有状态副本集

编辑模式

基本信息 容器组模板 存储卷模板 服务配置 标签设置 节点选择器

### 标签设置

标签是一个或多个关联到资源如容器组上的键值对，我们通常通过标签来识别、组织或查找资源对象

app	wordpress-mysql	<input type="button" value="删除"/>
-----	-----------------	-----------------------------------



## 查看 MySQL 有状态应用

在列表页可以看到有状态副本集 “wordpress-mysql”的状态为“更新中”，该过程需要拉取镜像仓库中指定 tag 的 Docker 镜像、创建容器和初始化数据库等一系列操作，状态显示 `ContainerCreating`。正

常情况下在一分钟左右状态将变为“运行中”，点击该项即可进入有状态副本集的详情页，包括资源状态、版本控制、监控、环境变量、事件等信息。

The screenshot displays the KubeSphere UI for managing a MySQL StatefulSet named 'mysql'. The top navigation bar includes tabs for '资源状态' (Resource Status), '版本控制' (Version Control), '监控' (Monitoring), '环境变量' (Environment Variables), and '事件' (Events). The '资源状态' tab is active.

**服务** section:

- Service: mysql-service (Virtual IP)
- Port: 3306:3306/TCP

**副本** section:

- 副本运行状态: 副本数 1/1 (期望副本数 1, 实际运行副本数 1)
- 说明: 有状态副本集 (StatefulSet) 用来描述有状态应用, 比如副本之间有主从关系, 数据需要做持久化。与部署 (Deployment) 相同的是, 有状态副本集创建的副本也是完全相同的, 不同的是每个副本有个固定且唯一的标识。即使副本被重新调度, 标识也不会发生变化。您可以用有状态副本集来实现应用的有序部署, 有序的删除, 有序的滚动更新。

**端口** section:

名称	协议	端口	主机端口
port	TCP	3306	-

**容器组** section:

- 容器组过滤器: 请输入关键字过滤
- 容器组: wordpress-mysql-0 (ContainerCreating)
- IP: -
- 主机: i-rgem3qkr(192.168.0.74)
- 说明: 暂时没有监控数据

至此，有状态应用 MySQL 已经创建成功，将作为 Wordpress 网站的后端数据库。下一步需要创建 Wordpress 部署并通过外网访问该应用，参见 [快速入门 – 部署 Wordpress](#)。

## 示例二 – 部署 Wordpress

### 目的

本文以创建一个部署 (Deployment) 为例，部署一个无状态的 Wordpress 应用，基于示例一的 MySQL 应用最终部署一个外网可访问的 [Wordpress](#) 网站。Wordpress 连接 MySQL 数据库的密码将以 [配置 \(ConfigMap\)](#) 的方式进行创建和保存。

### 前提条件

- 已创建了有状态副本集 MySQL，若还未创建请参考 [示例一 – 部署 MySQL](#)。
- 以 `project-regular` 用户登录 KubeSphere，进入已创建的企业空间下的项目

### 预估时间

约 15 分钟。

### 操作示例

#### 示例视频

#### 部署 Wordpress

##### 第一步：创建配置

Wordpress 的环境变量 `WORDPRESS_DB_PASSWORD` 即 Wordpress 连接数据库的密码属于敏感信息，不适合以明文的方式表现在步骤中，因此以创建配置 (ConfigMap) 的方式来代替该环境变量。创建的配置将在创建 Wordpress 的容器组设置时作为环境变量写入。

1.1. 在当前项目下左侧菜单栏的 配置中心 选择 配置，点击 创建配置。



1.2. 填写部署的基本信息，完成后点击 下一步。

- 名称：作为 Wordpress 容器中环境变量的名称，填写 `wordpress-configmap`
- 别名：支持中文，帮助您更好的区分资源，比如 `连接 MySQL 密码`
- 描述信息：简单介绍该 ConfigMap，如 `MySQL password`

The screenshot shows the 'Create Configuration' form for a ConfigMap. At the top right is an 'Edit Mode' switch. Below it are tabs for 'Basic Information' (selected) and 'Configuration Settings'. The 'Basic Information' section contains fields for 'Name' (set to 'wordpress-configmap'), 'Project' (set to 'demo-namespace'), and 'Description' (set to 'MySQL password'). There are also sections for 'Alias' (set to '连接 MySQL 密码') and a note about aliases.

1.3. ConfigMap 是以键值对的形式存在，此处键值对设置为 `WORDPRESS_DB_PASSWORD` 和 `123456`，完成后点击 创建。



## 第二步：创建存储卷

2.1. 在当前项目下左侧菜单栏的 存储卷，点击创建，基本信息如下。

- 名称：wordpress-pvc
- 别名：Wordpress 持久化存储卷
- 描述信息：Wordpress PVC



2.2. 完成后点击 下一步，存储卷设置中，参考如下填写：

- 存储类型：选择集群中已创建的存储类型，例如 **Local**
- 访问模式：选择单节点读写 (RWO)
- 存储卷容量：默认 10 Gi

基本信息 存储卷设置 标签设置

### 存储卷设置

按需填写存储卷的容量大小，存储卷大小和访问模式必须与存储类型和存储服务端能力相适应，访问模式通常选择为 RWO。

存储类型  
local

由集群管理员配置存储服务端参数，并按类型提供存储给用户使用。

访问模式  
 **ReadWriteOnce(RWO)**  
单个节点读写

存储卷容量  
10 Gi

2.3. 点击 **下一步**，设置标签为 `app: wordpress-pvc`，点击创建。

创建存储卷 编辑模式

基本信息 存储卷设置 标签设置

### 标签设置

标签是一个或多个关联到资源如容器组上的键值对，我们通常通过标签来识别、组织或查找资源对象

app wordpress-pvc

2.4. 点击左侧菜单中的 **存储卷**，查看存储卷列表，可以看到存储卷 `wordpress-pvc` 已经创建成功，状态是“准备就绪”，可挂载至工作负载。

若存储类型为 Local，那么该存储卷在被挂载至工作负载之前都将显示创建中，这种情况是正常的，因为 Local 目前还不支持存储卷动态配置 ([Dynamic Volume Provisioning](#))，挂载后状态将显示“准备就绪”。

名称	状态	容量	访问模式	挂载状态	创建时间
wordpress-pvc (Wordpress 持久化存储卷)	创建中			未挂载	2018-12-17 12:28:57
mysql-pvc-wordpress-mysql-0	准备就绪	10Gi	ReadWriteOnce	已挂载	2018-12-17 12:04:41

### 第三步：创建部署

在左侧菜单栏选择 **工作负载 → 部署**，进入列表页，点击 **创建部署**。

### 第四步：填写基本信息

基本信息中，参考如下填写，完成后点击 **下一步**。

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，比如 **wordpress**
- 别名：可选，支持中文帮助更好的区分资源，如 **Wordpress 网站**
- 描述信息：简单介绍该工作负载，方便用户进一步了解

创建部署

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

### 基本信息

您可以给部署起一个名字，以便在使用的时候容易区分。

名称 *	wordpress	项目	demo-namespace
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾		将根据项目进行资源进行分组，可以按项目对资源进行查看管理	
别名	Wordpress 网站	描述信息	This is a demo
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。			

## 第五步：容器组模板

5.1. 点击 **添加容器**。容器组模板中，名称可自定义，镜像填写 `wordpress:4.8-apache`，CPU 和内存此处暂不作限定，将使用在创建项目时指定的默认值，点击 **高级选项**。

创建部署

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

### 添加容器

容器名称 *	wordpress	镜像 *	wordpress:4.8-apache
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾		要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。	
CPU	最小使用 10 m	最大使用 100 m	
作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量 ≥ 容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m			
内存	最小使用 10 Mi	最大使用 200 Mi	
作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量 ≥ 容器内存最小使用值时，才允许将容器调度到该节点。			
高级选项 ↴			

5.2. 对 **端口** 和 **环境变量** 进行设置，其它项暂不作设置。参考如下填写，完成后点击 **保存**。

- 端口：名称可自定义，选择 **TCP** 协议，填写 Wordpress 在容器内的端口 **80**。
- 环境变量：这里需要添加两个环境变量

- 点击 **引用配置中心**，名称填写 `WORDPRESS_DB_PASSWORD`，选择在第一步创建的配置 (ConfigMap) `wordpress-configmap` 和 `WORDPRESS_DB_PASSWORD`。
- 点击 **添加环境变量**，名称填写 `WORDPRESS_DB_HOST`，值填写 `mysql-service`，对应的是 **示例一 - 部署 MySQL** 创建 MySQL 服务的名称，否则无法连接 MySQL 数据库，可在服务列表中查看其服务名。

创建部署

命令

参数

端口

环境变量

WORDPRESS\_DB\_PASSWORD: wordpress-configmap(连接 MySQL)

WORDPRESS\_DB\_HOST: mysql-service

镜像拉取策略

IfNotPresent

5.3. 副本数量和弹性伸缩暂无需设置，更新策略选择推荐的 **滚动更新策略**，然后点击 **下一步**。

## 第六步：存储卷设置

6.1. 此处选择 **添加已有存储卷**，选择第二步创建的存储卷 `wordpress-pvc`。

创建部署

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

### 存储卷设置

可以将同一个临时存储卷或持久化存储卷挂载至部署的容器组的各个副本内。

您可以根据需要选择适合您的存储卷类型进行添加

→ 添加已有存储卷 添加临时存储卷 引用配置中心

6.2. 设置存储卷的挂载路径，其中挂载选项选择 **读写**，挂载路径为 `/var/www/html`，保存后点击 **下一步**。

创建部署

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

### 挂载存储卷 > 设置挂载路径

wordpress-pvc(Wordpress 持久化存储卷) 闲置  
容量:  
存储类型: local  
访问模式:  
wordpress 读写 /var/www/html

## 第七步：查看部署

7.1. 为方便识别此应用，我们标签设置如下。下一步的节点选择器可以指定容器组调度到期望运行的节点上，此处暂不作设置，点击 **创建**，部署创建完成。

app: wordpress

7.2. 创建完成后，部署的状态为“更新中”是由于创建后需要拉取 wordpress 镜像并创建容器（大概一分钟左右），可以看到容器组的状态是“ContainerCreating”，待部署创建完成后，状态会显示“运行中”。

This screenshot shows the deployment details for a Wordpress application. The top navigation bar includes 'demo-namespace / deployments / wordpress / resource-status'. The main interface has tabs for '资源状态' (Resource Status), '版本控制' (Version Control), '监控' (Monitoring), '环境变量' (Environment Variables), and '事件' (Events). The '资源状态' tab is active, showing a summary card with '0/1 副本运行状态' (0/1副本运行状态) and '期望副本: 1' (Expected replicas: 1), '实际运行副本: 0' (Actual running replicas: 0). A note explains that Deployments describe the target state of the application, and increasing replicas can handle higher loads. Below this are sections for '端口' (Ports) and '容器组' (Container Groups). The '容器组' section lists a single pod named 'wordpress-95cd769b9-75pdq', which is currently in the 'ContainerCreating' state, indicated by a red circle around the status text.

7.3. 查看创建的部署 Wordpress，可以看到其状态显示运行中，下一步则需要为 Wordpress 创建服务和应用路由，最终暴露给外网访问。

This screenshot shows the deployment list page. The left sidebar includes '项目 demo-namespace...' and '概览', '应用', '工作负载' (Workload) with '部署' (Deployment) selected, '有状态副本集' (StatefulSet), '守护进程集' (DaemonSet), and '任务' (Jobs). The main area is titled '部署' (Deployment) and describes it as providing fine-grained management for user applications. It shows deployment statistics: 1 已用配额 (1 used quota), ∞ 规划配额 (∞ planned quota), and ∞ 剩余 Pod 配额 (∞ remaining Pod quota). A search bar and filter options are available. The deployment list table shows one entry: 'wordpress (Wordpress 网站)' with a status of '运行中(1/1)' (Running (1/1)).

## 第八步：创建服务

8.1. 在当前项目中，左侧菜单栏选择 网路与服务 → 服务，点击 创建。

项目 demo-namespace...  
概览  
应用  
工作负载  
存储卷  
网络与服务  
服务  
应用路由  
配置中心

服务  
一个 Kubernetes 的服务 (Service) 是一种抽象，它定义了一类 Pod 的逻辑集合和一个用于访问它们的策略。  
官网文档 参考文档

名称	IP地址	端口	应用	创建时间
mysql-service	Virtual IP: 10.233.25.75	端口: 3306:3306/TCP 节点端口: -		2018-12-17 12:04:41

共 1 个条目 1 / 1

## 8.2. 基本信息中，信息填写如下，完成后点击 **下一步**：

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，比如 `wordpress-service`
- 别名和描述信息：如 `Wordpress 服务`

创建服务 编辑模式

基本信息 服务设置 标签设置 外网访问

**基本信息**  
创建服务需要提供服务的名称和描述，服务名称不能和同一项目下已有的服务名称相同。

名称 \*  项目   
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾  
将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名  描述信息   
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

## 8.3. 服务设置参考如下填写，完成后点击 **下一步**：

- 服务类型：选择第一项 **通过集群内部IP来访问服务 Virtual IP**
- 选择器：点击 **指定工作负载** 可以指定上一步创建的部署 Wordpress，指定后点击 **保存**
- 端口：端口名称可自定义，服务的端口和目标端口都填写 `TCP` 协议的 `80` 端口
- 会话亲和性：None，完成参数设置，选择下一步

**说明:** 若有实现基于客户端 IP 的会话亲和性的需求，可以在会话亲和性下拉框选择 "ClientIP" 或在代码模式将 `service.spec.sessionAffinity` 的值设置为 "ClientIP"（默认值为 "None"），该设置可将来自同一个 IP 地址的访问请求都转发到同一个后端 Pod。



基本信息 服务设置 标签设置 外网访问

◀ 虚拟 IP

选择器 \*

⚠ 当前设置的选择器(app=wordpress)共影响到 1 个工作负载 [查看](#)

app wordpress

添加选择器 指定工作负载

端口 \*

port TCP 80 80

添加端口

会话亲和性

None

8.4. 添加标签并保存，本示例标签设置如下，添加后选择 **下一步**。

app=wordpress-service

8.5. 服务暴露给外网访问的访问方式，支持 NodePort 和 LoadBalancer，由于示例将以应用路由 (Ingress) 的方式添加 Hostname 来暴露服务给外网访问，所以服务的访问方式选择 **None**。点击 **创建**，**wordpress-service** 服务可创建成功。

服务

一个 Kubernetes 的服务 (Service) 是一种抽象，它定义了一类 Pod 的逻辑集合和一个用于访问它们的策略。

1 已用配额 1 虚拟 IP 0 Headless

官网文档 参考文档

名称	IP地址	端口	应用	创建时间
wordpress-service(Wordpress 服务)	Virtual IP: 10.233.60.172	端口: 80:80/TCP 节点端口: -		2018-12-17 15:19:45
mysql-service	Virtual IP: 10.233.25.75	端口: 3306:3306/TCP 节点端口: -		2018-12-17 12:04:41

## 第九步：创建应用路由

通过创建应用路由的方式可以将 WordPress 暴露出去供外网访问，与将服务直接通过 NodePort 或 LoadBalancer 暴露出去不同之处是应用路由是通过配置 Hostname 和路由规则（即 ingress）来访问，请参考以下步骤配置应用路由。关于应用路由的管理详见 [应用路由](#)。

9.1. 在当前项目中，左侧菜单选择 网路与服务 → 应用路由，点击 创建。

应用路由

应用路由提供一种聚合服务的方式，您可以将集群的内部服务通过一个外部可访问的 IP 地址暴露给集群外部。

0 已用配额 外网访问 Internet IP

官网文档 参考文档

创建应用路由

9.2. 创建应用路由，填入基本信息，完成后点击 下一步。

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，如 `wordpress-ingress`
- 别名和描述信息：比如 wordpress 应用路由

创建应用路由

编辑模式

基本信息 路由规则 注解 标签设置

### 基本信息

应用路由提供一种聚合服务的方式，您可以将集群的内部服务通过一个外部可访问的 IP 地址暴露给集群外部。

名称 *	wordpress-ingress	项目	demo-namespace
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾		将根据项目进行资源进行分组，可以按项目对资源进行查看管理	
别名	wordpress 应用路由	描述信息	wordpress 应用路由
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。			

9.3. 下一步，点击 **添加路由规则**，参考如下配置路由规则。完成后点击 **保存**，然后点击 **下一步**。

- Hostname：可自定义，比如设置为 `wordpress.demo.io`，它是应用路由规则的访问域名，最终使用此域名来访问对应的服务（如果访问入口是以 NodePort 的方式启用，需要保证 Hostname 能够在客户端正确解析到集群工作节点上；如果是以 LoadBalancer 方式启用，需要保证 Hostname 正确解析到负载均衡器的 IP 上）。
- 协议：选择 `http`（若使用 `https` 需要预先在密钥中添加相关的证书）
- Paths：应用规则的路径和对应的后端服务，路径填写 “/”，服务选择之前创建的服务 `wordpress-service`，端口填写 `wordpress-service` 服务端口 `80`，完成后点击下一步：

创建应用路由

编辑模式

基本信息 路由规则 注解 标签设置

### 设置路由规则

HostName \*

`wordpress.demo.io`

协议

`http`

Paths \*

/ wordpress-service 80

添加 Path

9.4. 在本地的 hosts 文件中添加记录来使域名解析到对应的 IP。例如，集群的公网 IP 为 `139.198.16.160`，且上一步将 hostname 设置为 `wordpress.demo.io`，我们在 `/etc/hosts` 文件中添加如下记录。

139.198.16.160 wordpress.demo.io

9.5. **Annotation** 是用户定义的“附加”信息，本示例暂不设置注解，点击 **下一步** 进入标签设置。

9.6. 为方便识别此应用，我们标签设置如下。点击 **创建**，即可创建成功。

app=wordpress-ingress

## 第十步：配置外网访问

10.1. 在当前项目中，左侧菜单选择 **项目设置** → **外网访问**，点击 **设置网关**，即应用路由的网关入口，每个项目都有一个独立的网关入口。

10.2. 网关入口提供提供如下两种方式，本示例以 **NodePort** 访问方式为例配置网关入口，选择

NodePort 然后点击 保存。

### 说明：

- NodePort：使用 NodePort 方式可以通过访问工作节点对应的端口来访问服务
- LoadBalancer：如果用 LoadBalancer 的方式暴露服务，需要有云服务厂商的 LoadBalancer 插件支持，比如 [QingCloud KubeSphere 托管服务](#) 可以将公网 IP 地址的 ID 填入 Annotation 中，即可通过公网 IP 访问该服务。(如果外网访问方式设置的是 LoadBalancer，参见 [应用路由](#) 的 LoadBalancer 方式。)

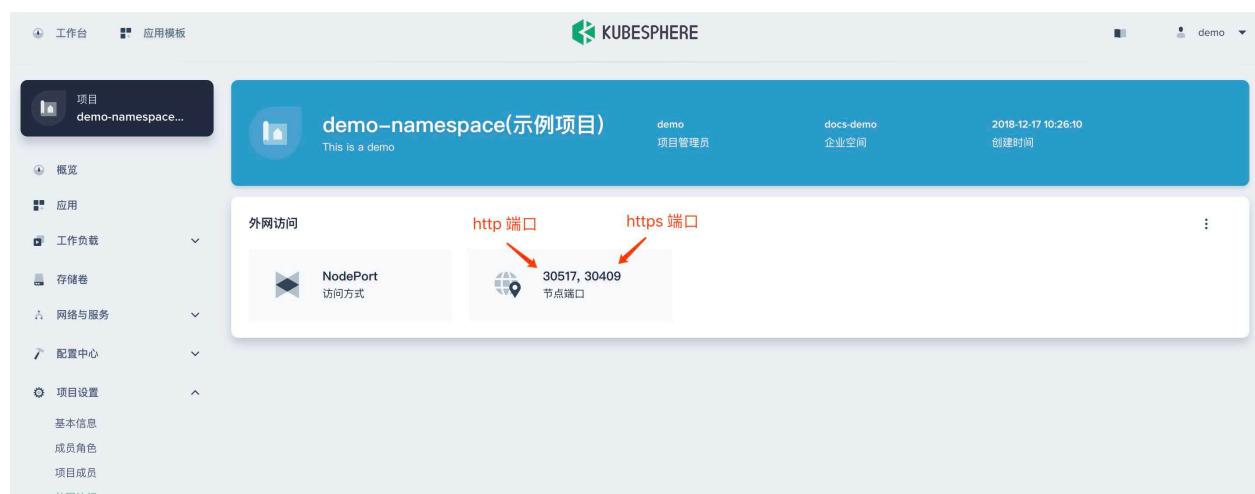
设置网关

### 设置网关

在创建应用路由之前，需要先启用外网访问入口，即网关。这一步是创建对应的应用路由控制器，来负责将请求转发到对应的后端服务。如不预先启用外网访问入口，创建的应用路由规则将无法访问。



10.3. 外网访问设置后，将产生两个节点端口（NodePort），分别是 HTTP 协议（80）和 HTTPS 协议（443）的节点端口，比如这里依次是 30517 和 30409。



注意：若需要在公网访问，可能需要进行端口转发和防火墙放行对应的端口，保证外网流量能够通过需要访问的端口如 30517，否则外网无法访问。

例如，在 QingCloud 云平台上，如果使用了 VPC 网络，则需要将 KubeSphere 集群中的任意一台主机上暴露的节点端口 (NodePort) `30517` 在 VPC 网络中添加端口转发规则，然后在防火墙放行该端口。

### 添加端口转发规则



The screenshot shows a table for adding port forwarding rules. The columns are: Name, Protocol, Source Port, Internal IP, Internal Port, Created At, and Operation. A row is selected with the name `wordpress-nodeport`, protocol `tcp`, source port `30517`, internal IP `192.168.0.4`, internal port `30517`, created at "just now", and status "disabled". The source port `30517` is highlighted with a red box.

<input type="checkbox"/>	名称	协议	源端口	内网 IP	内网端口	创建于	操作
<input type="checkbox"/>	wordpress-nodeport	tcp	30517	192.168.0.4	30517	几秒前	禁用

### 防火墙添加下行规则

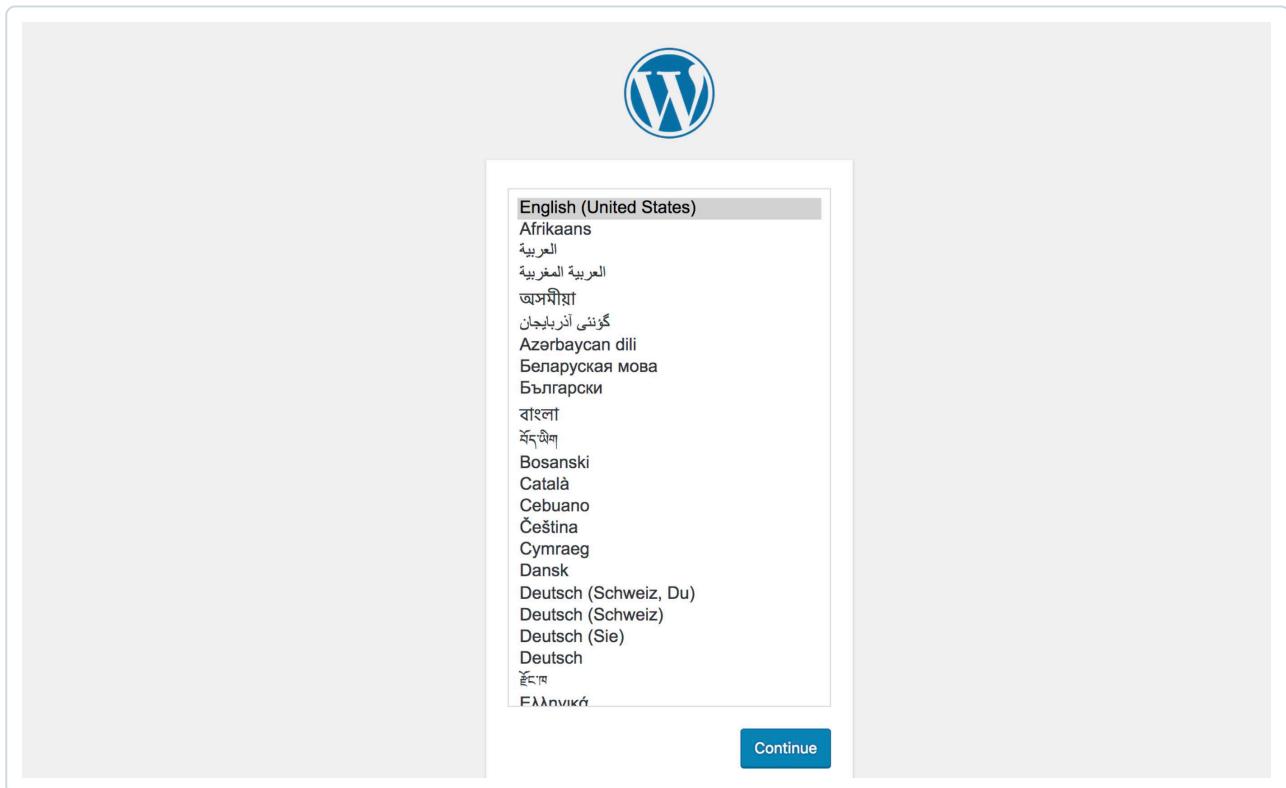


The screenshot shows a table for adding firewall downflow rules. The columns are: Name, Priority, Protocol, Behavior, Start Port (?), End Port (?), Source IP, and Operation. A row is selected with the name `wordpress-nodeport`, priority `10`, protocol `TCP`, behavior `Accept`, start port `30517`, and status "disabled". The start port `30517` is highlighted with a red box.

<input type="checkbox"/>	名称	优先级	协议	行为	起始端口 (?)	结束端口 (?)	源IP	操作
<input type="checkbox"/>	wordpress-nodeport	10	TCP	接受	30517			禁用

### 访问 Wordpress

设置完成后，WordPress 就以应用路由的方式通过网关入口暴露到外网，由于在路由规则中我们选择的是 http 协议，因此可以通过示例中在路由规则和外网访问配置的 `[$hostname]:{$NodePort}` 如 <http://wordpress.demo.io:30517> 访问 WordPress 博客网站。



至此，您已经熟悉了部署 (Deployments) 和有状态副本集 (Statefulsets) 的基本功能使用，关于部署和有状态副本集的各项参数释义，详见 [部署](#) 和 [有状态副本集](#)。

## 示例三 – 创建简单任务

实际工作中，我们经常需要进行批量数据处理和分析，以及按照时间进行调度执行。可以在 KubeSphere 中使用容器技术完成，也就是使用 Job (任务) 和 CronJob (定时任务) 来执行。这样方便维护较为干净的执行环境，减少不同任务工具的相互干扰。同时可以在集群上按照任务要求和资源状况进行动态伸缩执行。

Job 负责批处理任务，即仅执行一次的任务。任务具有并发的特性，可以抽象成一个任务中的多个 Pod 并行运行，保证批处理任务的一个或多个 Pod 成功结束。平时也存在很多需要并行处理的场景，比如批处理程序，每个副本（Pod）都会从任务池中读取任务并执行，副本越多，执行时间就越短，效率就越高，类似这样的场景都可以用任务来实现。

### 目的

本文档以创建一个并行任务去执行简单的命令计算并输出圆周率到小数点后 2000 位作为示例，说明任务的基本功能。

### 前提条件

已创建了企业空间和项目，若还未创建请参考 [管理员快速入门](#)。

### 预估时间

约 15 分钟。

### 操作示例

#### 示例视频

# 创建任务

登录 KubeSphere 控制台，在所属项目的左侧菜单栏，选择 **工作负载 → 任务**，进入任务列表页面。



## 第一步：填写基本信息

点击 **创建任务**，填写任务的基本信息，完成后点击 **下一步**。

基本信息页中，需要填写任务的名称和描述信息。

- **名称**：为创建的任务起一个简洁明了的名称，便于用户浏览和搜索。
- **别名**：别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。
- **描述**：简单介绍应用仓库的任务，让用户进一步了解该任务。



基本信息

名称 \*

job-demo

最长 253 个字符。只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目

demo-namespace

将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名

任务示例

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

This is a demo

## 第二步：任务设置

任务设置页中，通过设置 Job Spec 的四个配置参数来设置 Job 的任务类型，完成后点击 **下一步**。

- Back Off Limit：默认为 6，失败尝试次数，若失败次数超过该值，则 Job 不会继续尝试工作；如此处设置为 5 则表示最多重试 5 次。
- Completions：默认为 1，标志任务结束需要成功运行的 Pod 个数，如此处设置为 4 则表示任务结束需要运行 4 个 Pod。
- Parallelism：默认为 1，标志并行运行的 Pod 的个数；如此处设置为 2 则表示并行 2 个 Pod。
- Active Deadline Seconds：指定 Job 可运行的时间期限，超过时间还未结束，系统将会尝试进行终止，且 ActiveDeadlineSeconds 优先级高于 Back Off Limit；如此处设置 100 则表示超过 100s 后 Job 中的所有 Pod 运行将被终止。

创建任务

基本信息 任务设置 任务模板 存储卷设置 标签设置 节点选择器

编辑模式

### 任务设置

您可以在此配置任务(Job)的Job Spec格式，Job Controller负责根据Job Spec创建Pod，并持续监控Pod的状态，直至其成功结束。如果失败，则根据RestartPolicy（支持OnFailure和Never）决定是否创建新的Pod再次重试任务。

Back off Limit  
5  
失败尝试次数，若失败次数超过该值，则 job 不会继续尝试工作

Completions  
4  
标志 Job 结束需要成功运行的 Pod 个数

Parallelism  
2  
标志并行运行的 Pod 的个数

Active Deadline Seconds  
100  
job 运行的超时时间

## 第三步：配置任务模板

任务模板即设置 Pod 模板，其中 **RestartPolicy** 指通过同一节点上的 kubelet 重新启动容器，仅支持 Never 或 OnFailure，此处 RestartPolicy 选择 **Never**。

**说明：** **RestartPolicy** 表示当任务未完成的情况下：

- Never：任务会在容器组出现故障时创建新的容器组，且故障容器组不会消失。

- OnFailure: 任务会在容器组出现故障时其内部重启容器，而不是创建新的容器组。



下一步点击 **添加容器**，输入容器的名称 `pi` 和对应的镜像名 `perl`。CPU 和内存此处暂不作限定，将使用在创建项目时指定的默认值。

点击 **高级选项**，然后点击 **添加命令**，依次添加如下四行命令，即让每一个 Job 执行输出圆周率小数点后 2000 位。设置完成后点击 **保存**，然后选择 **下一步**。

```
# 命令
perl
-Mbignum=bpi
-wle
print bpi(2000)
```

创建任务

基本信息  任务设置  容器组模板  存储卷设置  标签设置  节点选择器

容器名称 \*

镜像 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU

最小使用	10	m	最大使用	100	m
------	----	---	------	-----	---

作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m

内存

最小使用	10	Mi	最大使用	200	Mi
------	----	----	------	-----	----

作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项 ▾

命令

- 
- 
- 
- 

## 第四步：标签设置

本示例暂不需要设置存储卷，可以跳过此步骤，点击 **下一步** 进入标签设置。标签（Label）用于指定资源对应的一组或者多组标签，便于标识和管理对象，这里的标签设置为 `app: job-demo`，无需设置节点选择器，点击 **创建**，任务创建成功，可在任务列表页查看。

项目 demo-namespace...

概览 应用 工作负载 部署 任务 定时任务

**任务**

任务 (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容器组成功结束。

1 已用配额  $\infty$  规划配额  $\infty$  剩余 Pod 配额

官网文档 参考文档

输入查询条件进行过滤

名称	状态	更新时间	Trigger
job-demo(任务示例)	已完成	2018-12-22 10:31:14	Trigger Config

创建

## 验证任务结果

1、点击该任务 `job-demo` 查看执行记录，可以看到任务执行的结果状态是“已完成”，并且一共运行了

4 个 Pod，这是因为在第二步 Completions 设置为 4。

The screenshot shows the KubeSphere UI for a job named 'job-demo'. On the left, there's a sidebar with 'demo-namespace / jobs / job-demo / records'. The main area has tabs: '执行记录' (Execution Record) (highlighted in green), '资源状态' (Resource Status), '环境变量' (Environment Variables), and '事件' (Events). Under '执行记录', it says '已完成(4/4)' (Completed 4/4). A table lists the execution record details:

序号	状态	消息	开始时间	结束时间
1	已完成(4/4)	-	2018-12-22 10:30:56	2018-12-22 10:31:14

On the left side, under '详情' (Details), the 'completions' field is highlighted with a red circle and shows the value '4'.

2、在任务详情页的 资源状态，可以查看任务执行过程中创建的容器组。由于 Parallelism 设置为 2，因此任务将预先并行地创建 2 个容器组，然后再继续并行创建 2 个容器组，任务结束时将创建 4 个容器组，

The screenshot shows the KubeSphere UI for the same job 'job-demo'. The left sidebar shows 'demo-namespace / jobs / job-demo / resource-status'. The main area has tabs: '执行记录' (Execution Record), '资源状态' (Resource Status) (highlighted in green), '环境变量' (Environment Variables), and '事件' (Events). Under '资源状态', there's a section for '容器组' (Container Groups). It shows two container groups: 'job-demo-5slt4' (IP: 10.233.87.155, Host: ks-allinone(192.168.0.19)) and 'job-demo-v79rw' (IP: 10.233.87.161, Host: ks-allinone(192.168.0.19)). Both groups have a note '暂时没有监控数据' (No monitoring data available). Below these, there's a summary for 'docker pi': Image: perl:latest, CPU: 100m request 1 limit, Memory: 50Mi request 1000Mi limit.

3、点击其中一个容器组，比如 `job-demo-5slt4`，查看该容器组中的容器。

demo-namespace / jobs / job-demo / job-demo-5slt4 / resource-status

容器组 job-demo-5slt4

容器

标签

- app: job-demo
- controller-uid: 9cc0d570-0591-11e9-ae3e-5254d5e7e238
- job-name: job-demo

详情

- 项目: 示例项目
- 应用: job-demo

容器

镜像: perl:latest

CPU: 100m request 1 limit  
内存: 50Mi request 1000Mi limit

存储设备

default-token-sskpr 容量: - 存储类型: Secret 密钥: default-token-sskpr 挂载: pi → /var/run/secrets/kubernetes.io/serviceaccount:unt read-only

4、在 资源状态 页，进一步点击该容器然后在 容器日志 的 Tab 下，查看 pi 容器中命令计算圆周率到小数点后 2000 位的输出结果。另外，可点击左侧的 终端 进入容器内部执行命令。

demo-namespace / jobs / job-demo / job-demo-5slt4 / pi / logs

容器 pi

终端

详情

- 项目: 示例项目
- 应用: -
- 状态: 更新中
- 镜像: perl:latest
- 端口: -
- 命令: perl -Mbignum=bpi -wle print bpi(200)

容器日志

```
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647095844609550
```

至此，您已经熟悉了任务 (Job) 的基本功能使用，关于任务的各项参数释义详见 [任务](#)。

## 示例四 – 一键部署应用

应用为用户提供完整的业务功能，由一个或多个特定功能的组件组成。KubeSphere 应用模板所纳管的应用基于 Helm 打包规范构建，并通过统一的公有或私有的应用仓库交付使用，应用可根据自身特性由一个或多个 Kubernetes 工作负载 (workload) 和服务组成。

一键部署应用基于 KubeSphere 应用模板，应用模板可以查看来自所有应用仓库的应用，通过可视化的方式在 KubeSphere 中展示并提供部署及管理功能，常用来提供开发和测试使用的场景所需的中间件服务。用户可以基于应用模板快速地一键部署常用的应用到 KubeSphere 中，通过编辑服务的外网访问方式，用户可以在外部访问该应用。KubeSphere 基于 [OpenPitrix](#) 构建了应用仓库服务，在使用应用模板前可以添加一个包含应用的应用仓库，详见 [添加应用仓库](#)，KubeSphere 会自动加载此仓库下的所有应用。本示例也准备了一个示例仓库仅供测试部署使用。

### 目的

本示例通过导入一个包含测试模板的应用仓库，演示如何在 KubeSphere 中一键部署应用，并通过外网访问该应用，演示应用仓库、应用模板和应用列表的基本功能。

### 前提条件

已创建了企业空间和项目，若还未创建请参考 [管理员快速入门](#)。

### 预估时间

约 15 分钟。

### 操作示例

#### 示例视频

## 第一步：添加应用仓库

应用仓库的后端可以是 QingStor 对象存储，也可以是 AWS 对象存储，里面存储的内容是开发者开发好的应用的配置包以及索引文件。因此在 KubeSphere 添加应用仓库之前，需提前在云平台创建对象存储并上传应用配置包，一般是基于 Helm Chart 规范开发的应用。本示例准备了一个基于 [QingStor 对象存储](#) 的应用仓库，里面包含了用于演示的 Nginx 应用配置包，若需要自行添加和上传应用，请参阅 [添加应用仓库](#)。

1、应用仓库一般仅集群管理员或拥有应用仓库权限的用户操作。以 **集群管理员** 或拥有应用仓库权限的账号登录 KubeSphere，选择 **平台管理 → 应用仓库**，点击添加应用仓库。



2、填写应用仓库的详细信息，如下添加一个 http 协议的示例仓库：<http://docs-repo.gd2.qingstor.com>，完成后点击 确定。

**添加应用仓库**

应用仓库名称 \*

类型

Helm

URL:

所输入的 URL 需要先验证才可进行添加或编辑操作

描述信息

This is a demo

3、添加完成后，可以在顶部的 **应用模板** 中查看新添加仓库的应用，点击应用名称为 **nginx** 的应用，准备部署该应用。

工作台 应用模板

应用一键部署

通过可视化的方式在 KubeSphere 中展示并提供部署及管理功能。用户可以基于应用模板快速地一键部署应用

全部仓库

应用模板

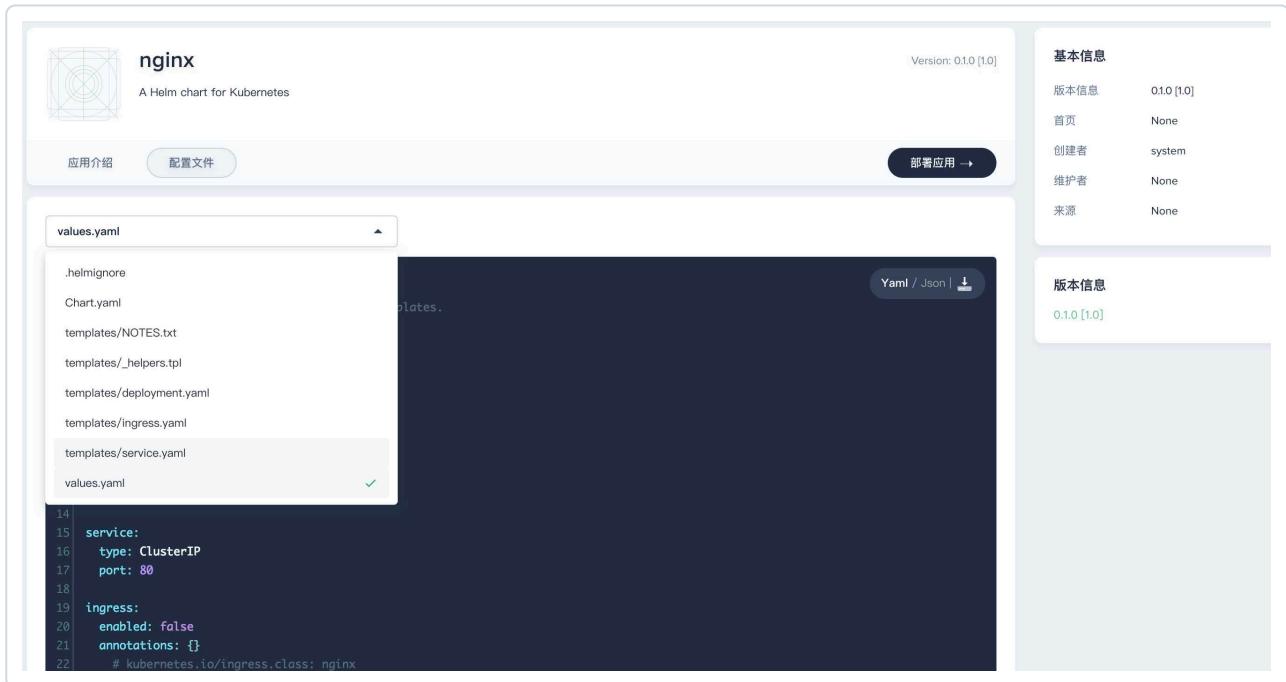
nginx Version: 0.1.0 [1.0]

A Helm chart for Kubernetes

## 第二步：部署应用

1、查看和部署应用等后续操作可使用管理员快速入门中创建的 project-regular 账号登录操作。点击进入 Nginx 应用模板的详情页面，可以查看其应用介绍、基本信息、版本信息和配置文件，该页面支持编

辑和下载应用的配置文件，支持 Yaml 和 Json 格式。

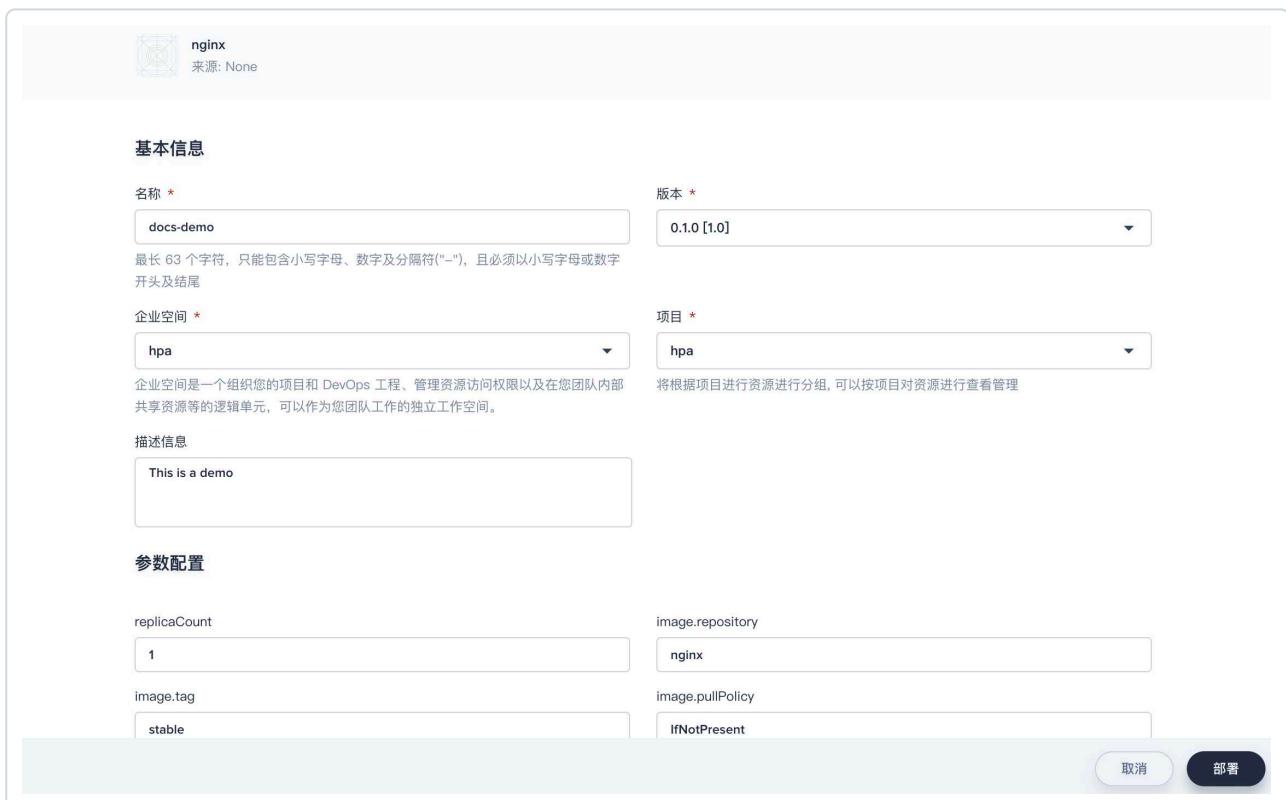


The screenshot shows the Helm chart interface for the 'nginx' chart. On the left, there's a sidebar with files like .helmignore, Chart.yaml, and various templates. The main area displays the 'values.yaml' file content:

```
14
15   service:
16     type: ClusterIP
17     port: 80
18
19   ingress:
20     enabled: false
21     annotations: {}
22     # kubernetes.io/ingress.class: nginx
```

On the right, there are sections for '基本信息' (Basic Information) and '版本信息' (Version Information), both showing '0.1.0 [1.0]'. A large '部署应用' (Deploy Application) button is located at the top right.

2、点击 **部署应用**，填写应用的基本信息，应用名称可自定义，其中参数配置是读取的 `values.yaml` 文件中的默认值。选择预先创建的企业空间和项目，暂无需修改其它配置，点击 **部署**。



The screenshot shows the deployment configuration dialog for the 'nginx' application. The '基本信息' (Basic Information) tab is selected, displaying fields for '名称' (Name) set to 'docs-demo', '版本' (Version) set to '0.1.0 [1.0]', '企业空间' (Enterprise Space) set to 'hpa', and '项目' (Project) set to 'hpa'. The '描述信息' (Description) field contains the text 'This is a demo'. The '参数配置' (Parameter Configuration) tab shows the following parameter settings:

参数	值
replicaCount	1
image.repository	nginx
image.tag	stable
image.pullPolicy	IfNotPresent

At the bottom right are '取消' (Cancel) and '部署' (Deploy) buttons.

3、点击左侧菜单栏的 **应用**，通过应用模板部署的应用可以在项目下的 **应用** 列表页面查看，此时将在当前项目的 **应用** 中创建一个新的 Nginx 应用，待镜像拉取和容器启动完毕，状态即可显示为 **已启用**。

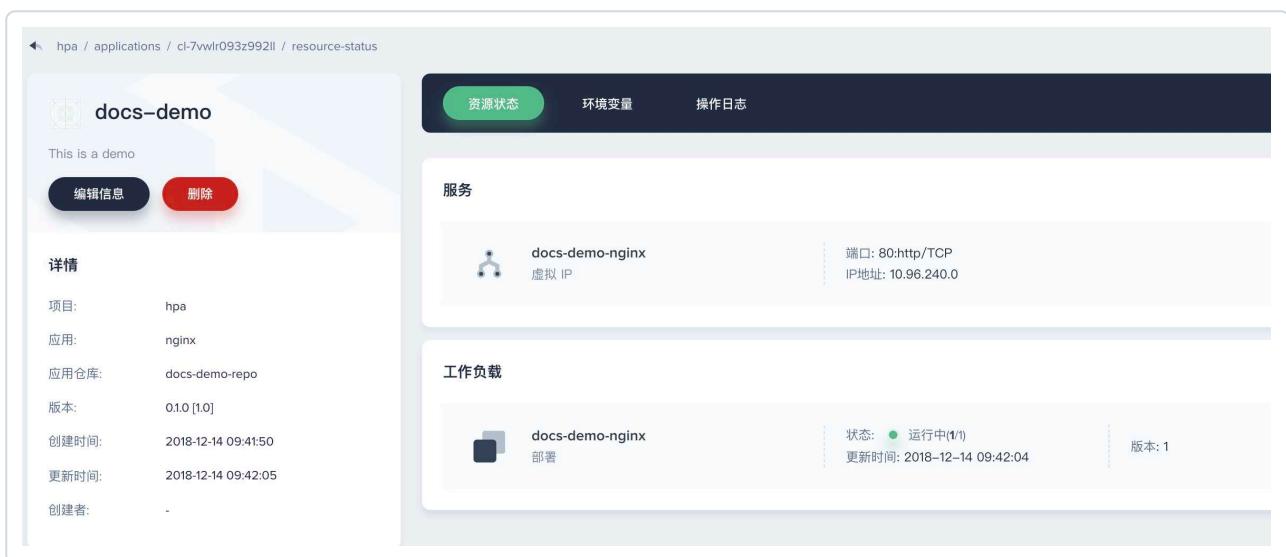


The screenshot shows the KubeSphere application list interface. On the left, there's a sidebar with project navigation (hpa), sections for Overview, Application, Workload, Storage, and Network & Services. The main area is titled '应用' (Application) with a brief description about Helm-based application templates. A search bar is at the top. Below it, a card for 'docs-demo' is displayed, showing its status as '已启用' (Enabled), version information, and last updated time (2018-12-14 09:42:05). A green button labeled '部署新应用' (Deploy New Application) is visible.

点击应用可查看该应用后端的工作负载详情，如部署 (Deployment), 有状态副本集 (Statefulset) 和服务 (Service)。在 **应用** 中也可以点击 **部署新应用** 进入 **应用模板** 完成一键部署。

### 第三步：查看应用详情

1、在应用列表点击 Nginx 应用 (docs-demo)，进入应用的资源状态页可以看到该应用是由服务和工作负载组成，并支持查看其环境变量和操作日志。



This screenshot shows the 'resource-status' page for the 'docs-demo' application. At the top, there are tabs for 资源状态 (Resource Status), 环境变量 (Environment Variables), and 操作日志 (Operational Log). The main content is divided into two sections: '服务' (Service) and '工作负载' (Workload).

- 服务:** Shows a service named 'docs-demo-nginx' with a virtual IP. It lists port 80/http/TCP and IP address 10.96.240.0.
- 工作负载:** Shows a deployment named 'docs-demo-nginx'. The status is '运行中(1/1)' (Running 1/1), updated at 2018-12-14 09:42:04. The version is 1.

On the left, there's a detailed '详情' (Details) panel with the following information:

- 项目: hpa
- 应用: nginx
- 应用仓库: docs-demo-repo
- 版本: 0.1.0 [1.0]
- 创建时间: 2018-12-14 09:41:50
- 更新时间: 2018-12-14 09:42:05
- 创建者: -

2、点击 **工作负载** 中的 **docs-demo-nginx**，可以查看该应用的部署 (Deployment) 详情页面，该页面可以更详细地查看 Pod 和容器的资源状态、监控，并支持编辑 Pod 的基本功能，如编辑配置模板、弹性伸缩、添加健康检查器等。

3、返回应用的详情页面，点击 **服务** 中的 `docs-demo-nginx` 可以查看该应用的服务详情。若需要将该应用暴露给外网访问，可点击 **更多操作** → **编辑外网访问**。

4、外网访问支持以下三种，本示例以 `NodePort` 访问方式为例，点击确定。

- 说明：
- None: 只在集群内部访问服务，集群外部无法访问。
- NodePort: 使用 NodePort 方式可以通过访问工作节点对应的端口来访问服务

- LoadBalancer: 如果用 LoadBalancer 的方式暴露服务, 需要有云服务厂商的 LoadBalancer 插件支持, 比如 [QingCloud KubeSphere 托管服务](#) 可以将公网 IP 地址的 ID 填入 Annotation 中, 即可通过公网 IP 访问该服务。

### 编辑外网访问

## 外网访问

将服务暴露给外网

访问方式

**NodePort**

**None**  
不提供外网访问

**NodePort**  
通过访问集群节点的对应端口来访问服务 (NodePort) 

**LoadBalancer**  
通过云服务商提供的负载均衡器来访问服务 (LoadBalancer)

5、将在当前服务的详情页面生成一个节点端口 (NodePort), 比如本示例是 **30055**, 可通过外网访问该 NodePort 对外暴露的服务。

docs-demo-nginx

资源状态 事件

端口

http	容器端口	→ TCP →	80	服务端口	→ TCP →	30055	节点端口
------	------	---------	----	------	---------	-------	------

工作负载

docs-demo-nginx	部署	状态: <span style="color: green;">运行中(1/1)</span>	更新时间: 2018-12-14 09:41:50
-----------------	----	---	---------------------------

## 第四步：访问应用

**注意：**若需要在外网访问，可能需要绑定公网 EIP 并配置端口转发，若公网 EIP 有防火墙，请在防火墙添加规则放行对应的端口（比如 30055），保证外网流量可以通过该端口，外部才能够访问。

例如，在 QingCloud 云平台上，如果使用了 VPC 网络，则需要将 KubeSphere 集群中的任意一台主机上暴露的节点端口（NodePort）**30055** 在 VPC 网络中添加端口转发规则，然后在防火墙放行该端口。

### 添加端口转发规则



<input type="checkbox"/>	名称	协议	源端口	内网 IP	内网端口	创建于	操作
<input type="checkbox"/>	nginx-nodeport	tcp	30055	192.168.0.4	30055	几秒前	禁用

### 防火墙添加下行规则



<input type="checkbox"/>	名称	优先级	协议	行为	起始端口 (?)	结束端口 (?)	源IP	操作
<input type="checkbox"/>	wordpress-nodeport	10	TCP	接受	30055			禁用

### 访问 Nginx

当以上步骤都顺利完成，此时在浏览器可以通过 公网 IP : 30055 ( **\${EIP}:\${NODEPORT}**) 在外网访问 Nginx 应用。



至此，您已经熟悉了如何通过添加应用仓库并使用应用模板一键部署应用的操作流程。

## 访问容器终端

在后台如果需要进入容器终端通常需要执行命令 `docker exec -it [容器编号] /bin/bash` 才可以进入容器内部，在 KubeSphere 中访问容器终端是非常方便的。

在 nginx 的部署详情页，点击容器组下的 nginx 容器组 → nginx 容器，即可进入容器的详情页。

部署状态

资源状态

版本控制

监控

环境变量

事件

部署 (Deployment) 用来描述期望应用达到的目标状态，主要用来描述无状态应用，副本的数量和状态由其背后的控制器来维护，确保状态与定义的期望状态一致。您可以增加副本数量来满足更高负载；回滚部署的版本来消除程序的错误修改；创建自动伸缩器来弹性应对不同场景下的负载。

标签

- app.kubernetes.io/instance: docs-demo
- app.kubernetes.io/managed-by: Tiller
- app.kubernetes.io/name: nginx
- helm.sh/chart: nginx-0.1.0

详情

项目: demo-namespace

应用:

创建时间: 2019-01-08 10:04:54

更新时间: 2019-01-08 10:05:42

创建者: 未知

端口

名称	协议	端口	主机端口
http	TCP	80	-

容器组

请输入关键字过滤

容器名	IP	CPU 使用量	内存 使用量
docs-demo-nginx-89954f5c8-qhlmw	10.233.87.132	CPU 0.09m	内存 6.89MB



点击左侧 **终端** 按钮，即可进入该容器的终端。



进入容器终端后，即可在容器内部执行命令进行操作。比如，执行 `cat /etc/hosts` 查看该容器的 Pod IP 和 Pod Name。



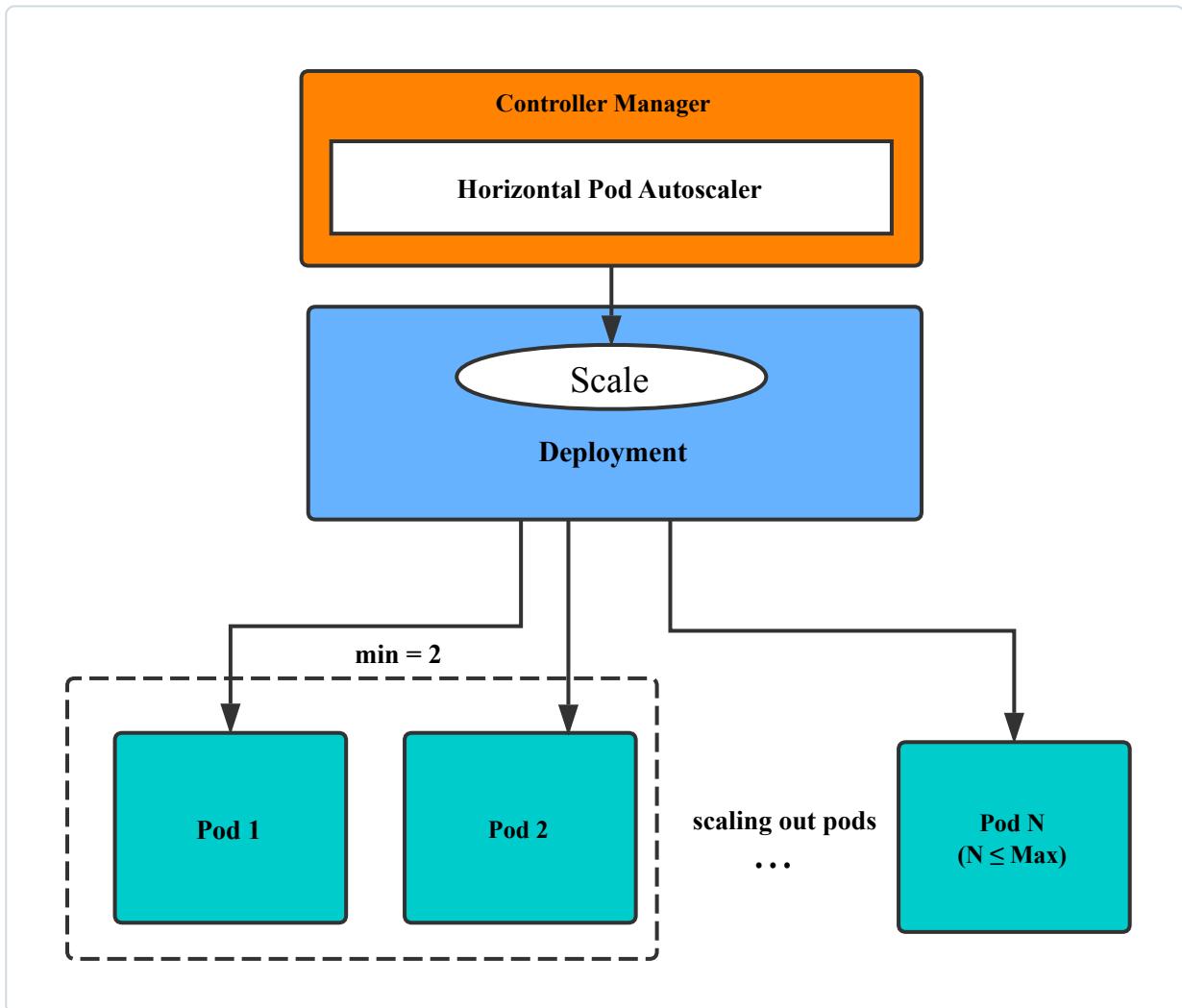
## 示例五 – 设置弹性伸缩

**Pod 弹性伸缩 (HPA)** 是高级版新增的功能，应用的资源使用率通常都有高峰和低谷的时候，如何动态地根据资源使用率来削峰填谷，提高集群的平台和集群资源利用率，让 Pod 副本数自动调整呢？这就有赖于 Horizontal Pod Autoscaling 了，顾名思义，能够使 Pod 水平自动伸缩，也是最能体现 KubeSphere 之于传统运维价值的地方，用户无需对 Pod 手动地水平扩容 (Scale out/in)。HPA 仅适用于创建部署 (Deployment) 时或创建部署后设置，支持根据集群的监控指标如 CPU 使用率和内存使用量来设置弹性伸缩，当业务需求增加时，KubeSphere 能够无缝地自动水平增加 Pod 数量，提高系统的稳定性。

### 弹性伸缩工作原理

HPA 在 Kubernetes 中被设计为一个 Controller，可以在 KubeSphere 中通过简单的设置或通过 UI 的 `kubectl autoscale` 命令来创建。HPA Controller 默认每 **30 秒** 轮询一次，检查工作负载中指定的部署 (Deployment) 的资源使用率，如 CPU 使用率或内存使用量，同时与创建部署时设定的值和指标做比较，从而实现 Pod 副本数自动伸缩的功能。

在部署中创建了 HPA 后，Controller Manager 将会访问用户自定义的 REST 客户端或 Heapster，获取用户定义的资源中每一个容器组的利用率或原始值（取决于指定的目标类型）的平均值，然后，与 HPA 中设置的指标进行对比，同时计算部署中的 Pod 需要弹性伸缩的具体值并实现操作。在底层 Kubernetes 中的 Pod 的 CPU 和内存资源，实际上还分为 limits 和 requests 两种情况，在调度的时候，kube-scheduler 将会根据 requests 的值进行计算。因此，当 Pod 没有设置资源请求值 (request) 时，弹性伸缩功能将不会工作。



## 目的

本示例演示创建一个设置了弹性伸缩的应用，通过另外创建的多个 Pod 循环向该应用发送无限的查询请求访问应用的服务，相当于手动增加 CPU 负载，即模拟多个用户同时访问该服务，演示其弹性伸缩的功能，详细说明 HPA 的工作原理以及如何在部署中设置 Pod 水平自动伸缩。

## 前提条件

已创建了企业空间和项目，若还未创建请参考 [管理员快速入门](#)。

# 预估时间

约 25 分钟。

## 操作示例

### 示例视频

## 创建 HPA

### 第一步：创建部署

1、以项目管理员 `project-admin` 登录 KubeSphere。在项目列表里点击之前创建好的项目 `demo-namespace`，进入项目；

在左侧菜单栏选择 **工作负载 → 部署**，点击 **创建部署**。



2、填写部署的基本信息如下：

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，如 `hpa-example`
- 别名：可选，更好的区分资源，并支持中文名称

- 描述信息：简单介绍该部署，方便用户进一步了解

点击 **下一步**；

创建部署

基本信息      容器组模板      存储卷设置      标签设置      节点选择器

**基本信息**

您可以给部署起一个名字，以便在使用的时候容易区分。

名称 \*

hpa-example

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目

demo-namespace

将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名

HPA 示例

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

This is a demo

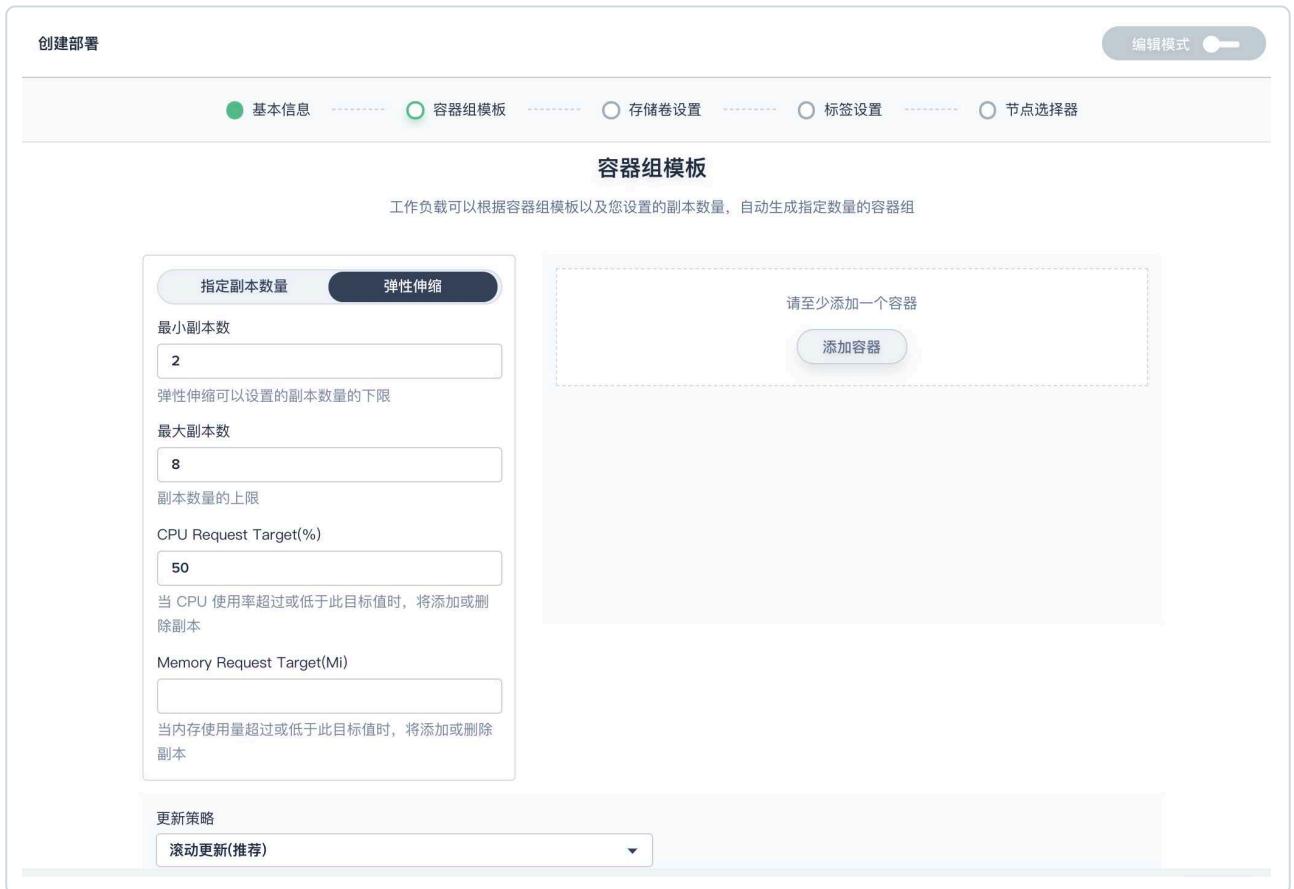
## 第二步：配置弹性伸缩参数

点击 **弹性伸缩**，填写信息如下：

- 弹性伸缩 (HPA)

- 最小副本数：弹性伸缩的容器组数量下限，此处设置 **2**
- 最大副本数：弹性伸缩的容器组数量上限，此处设置 **8**
- CPU Request Target(%) (CPU 目标值)：当 CPU 使用率超过或低于此目标值时，将相应地添加或删除副本，此处设置为 **50%**
- Memory Request Target(Mi) (内存目标值)：当内存使用量超过或低于此目标值时，将添加或删除副本，本示例以增加 CPU 负载作为测试，内存暂不作限定

**注：**容器组模板中可以设置 HPA 相关参数，以设置 CPU 目标值作为弹性伸缩的计算参考，实际上会为部署创建一个 **Horizontal Pod Autoscaler** 来调度其弹性伸缩。



### 第三步：添加容器

- 1、点击 **添加容器**，填写容器的基本信息，容器名称可自定义，镜像填写 **nginx**，将默认拉取 **latest** 的镜像。CPU 和内存此处暂不作限定，将使用在创建项目时指定的默认请求值；
- 2、高级设置暂不作设置，点击 **保存**；
- 3、更新策略选择推荐的 **滚动更新**，然后点击 **下一步**；
- 4、由于示例不会用到持久化存储，因此存储卷也不作设置，点击 **下一步**；

基本信息

容器组模板

存储卷设置

标签设置

节点选择器

◀ 添加容器

容器名称 \*

nginx

最长253个字符，只能包含小写字母、数字及分隔符“-”，且必须以小写字母或数字开头及结尾

镜像 \*

nginx

要从私有镜像仓库部署，需要先 创建镜像仓库，然后拉取镜像。

高级选项 ▾

## 第四步：标签设置

标签设置为 `app : nginx-hpa`；

节点选择器无需设置，点击 **创建**，即可查看 Nginx 的运行状态详情。

**注：**标签是一个或多个关联到资源如容器组上的键值对，通过标签来识别、组织或查找资源对象；  
kube-scheduler 将根据各主机 (Node) 的负载状态，将 Pod 随机调度到各台主机上。

## 创建服务

### 第一步：填写基本信息

在左侧菜单栏，点击 **网络与服务 → 服务** 进入服务列表页，点击 **创建**；

- 名称：此处填写 `hpa-example`，注意该服务名将用于被其它 Pod 访问
- 别名和描述信息：HPA 示例，帮助用户更好地了解该服务

点击 **下一步**；

创建服务

编辑模式

基本信息 服务设置 标签设置 外网访问

### 基本信息

创建服务需要提供服务的名称和描述，服务名称不能和同一项目下已有的服务名称相同。

名称 \* hpa-example

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

别名 HPA 服务

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

项目 demo-namespace

将根据项目进行资源进行分组，可以按项目对资源进行查看管理

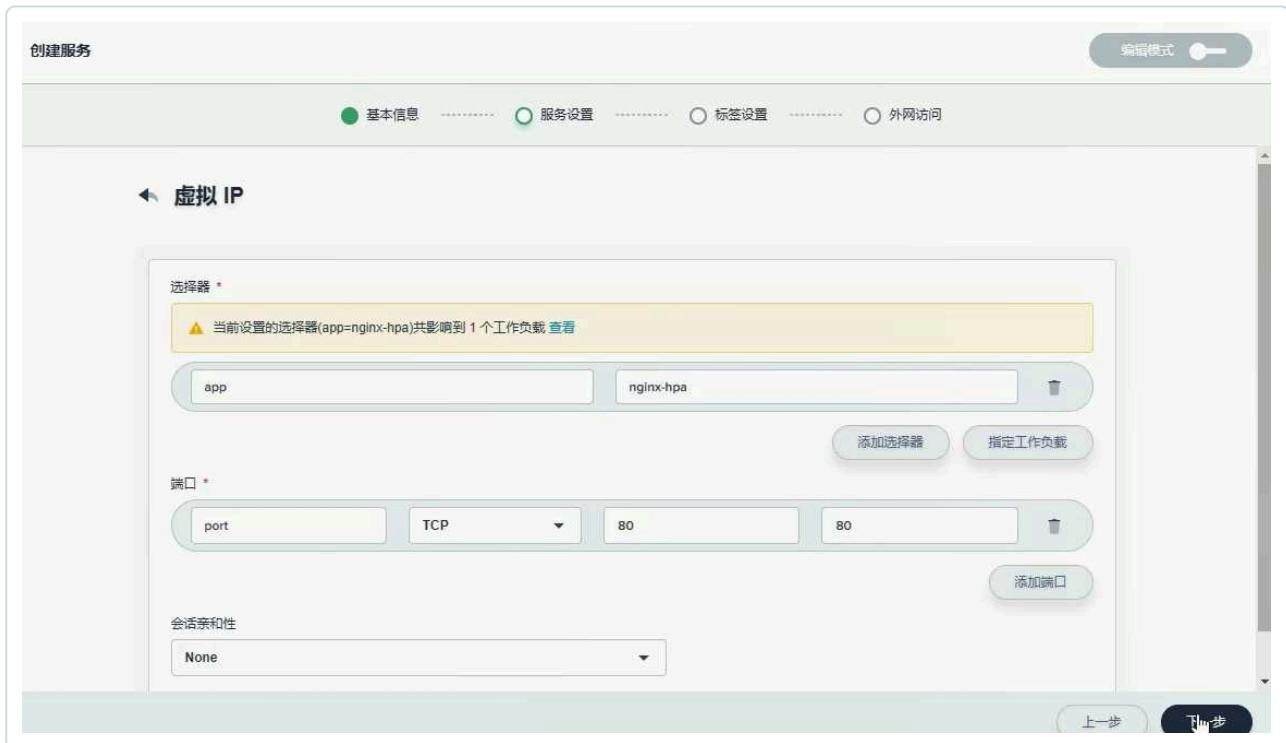
描述信息 This is a HPA service

## 第二步：服务设置

由于是在集群内部访问该服务，因此服务设置选择第一项 **通过集群内部 IP 来访问服务 Virtual IP**，服务信息参考以下填写：

- 选择器：点击 **指定工作负载**，在弹出的界面中指定上一步创建的部署 hpa-example；
- 端口：端口名称可自定义，服务的端口和目标端口都填写 **TCP** 协议的 **80** 端口；
- 会话亲和性：None

点击 **下一步**；



### 第三步：标签设置

为方便识别此服务，标签设置为 `app: hpa-example`，点击 **下一步**；

外网访问方式选择 `None`，即仅在集群内部访问该服务，点击 **创建**。

### 创建 Load-generator

另外创建一个部署 (Deployment) 用于向上一步创建的服务不断发送查询请求，模拟增加 CPU 负载。

### 第一步：基本信息

在左侧菜单栏选择 **工作负载 → 部署**，点击创建部署，填写部署的基本信息，其它项保持默认：

- 名称：必填，起一个简洁明了的名称，便于用户浏览和搜索，如 `load-generator`
- 别名：更好的区分资源，并支持中文名称，比如 `增加负载`
- 描述信息：简单介绍该部署

点击 **下一步**；

创建部署

基本信息 容器组模板 存储卷设置 标签设置 节点选择器 编辑模式

### 基本信息

您可以给部署起一个名字，以便在使用的时候容易区分。

名称 \*  最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目  将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名  别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

## 第二步：容器组模板

1、点击 **编辑模式**，将副本数的字段 `replicas` 改为 50，再次点击 **编辑模式** 切换回 UI；

创建部署 编辑模式

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: hpa
  annotations:
    kubesphere.io/isElasticReplicas: 'false'
  displayName: 增加负载
  desc: This is a demo
  labels: &ref_0
  app: load-generator
  name: load-generator
spec:
  replicas: 50| ←
  selector:
    matchLabels: *ref_0
  template:
    metadata:
      labels: *ref_0
    spec:
      containers:
        - name: busybox-container
          image: busybox
          imagePullPolicy: IfNotPresent
          serviceAccount: default
  strategy:
```

YAML / JSON

创建

2、回到 **容器组模板** 页，点击 **添加容器**，设置参数如下：

- 容器名称：必填，起一个简洁明了的名称，比如 `busybox-container`；
- 镜像名称：填写 `busybox`；

- CPU 和内存：此处暂不作限定，将使用在创建项目时指定的默认请求值。



3、展开 **高级选项**，然后分别点击 **添加命令** 和 **添加参数**，填写用于对 nginx 服务增加 CPU 负载的命令和参数，其它设置暂无需配置。设置命令和参数如下：

**注意：**参数中服务的 http 地址应替换为您实际的服务和项目名称。例如，我们在创建 HPA 的服务时，服务名称为 hpa-example，当前的项目名称为 demo-namespace，那么该服务在内部的 http 地址为 `http://hpa-example.demo-namespace.svc.cluster.local`。

```
# 命令
sh
-c

# 参数 (http 地址参考: http://{$服务名称}.{$项目名称}.svc.cluster.local)
while true; do wget -q -O- http://hpa-example.demo-namespace.svc.cluster.local; done
```



4、完成填写后，点击 **保存**；

5、更新策略选择推荐的 **滚动更新**，点击 **下一步**；

**注：本步骤用于创建 busybox 容器，并设置多个副本数使容器启动后同时循环地向上一步创建的服务发送请求。为了快速地看到弹性伸缩的效果，副本数可设置为 30 ~ 50 个。**

### 第三步：标签设置

本示例暂未用到存储，点击 **下一步** 跳过存储卷设置；

为方便识别此资源，标签设置为 `app:load-generator`；

点击 **创建**；

至此，以上一共创建了两个部署（分别是 `hpa-example` 和 `load-generator`）和一个服务 (`hpa-example`)。

## 验证弹性伸缩

### 第一步：查看部署状态

在部署列表中，点击之前创建的部署 **hpa-example**，进入资源详情页，请重点关注此时容器组的弹性伸缩状态和当前的 CPU 使用率以及它的监控情况。

The screenshot shows the KubeSphere UI for the deployment **hpa-example**. The top navigation bar includes tabs for 资源状态 (Resource Status), 版本控制 (Version Control), 监控 (Monitoring), 环境变量 (Environment Variables), and 事件 (Events). The main content area displays the deployment status (2/2 replicas), scaling configuration (min: 2, max: 8, target CPU: 50%, target memory: 0), and container group details for two pods.

### 第二步：查看弹性伸缩情况

待 **load-generator** 的所有副本的容器都创建成功并开始访问 **hpa-example** 服务时，如下图所示，刷新页面后可见 CPU 使用率明显升高，目前上升至 62%，并且期望副本和实际运行副本数都变成了 4，这是由于我们之前设置的 Horizontal Pod Autoscaler 开始工作了，**load-generator** 循环地请求该 **hpa-example** 服务使得 CPU 使用率迅速升高，HPA 开始工作后使得该服务的后端 Pod 副本数增加共同处理大量的请求，**hpa-example** 的副本数会随 CPU 的使用率升高而继续增加，正好也证明了弹性伸缩的工作原理。

This is a demo

编辑信息 更多操作 ▾

标签 app hpa-example

详情

项目: hpa  
应用: hpa-example  
创建时间: 2018-12-13 15:24:35  
更新时间: 2018-12-13 17:26:00  
创建者: admin

资源状态 版本控制 监控 环境变量 事件

副本运行状态  
期望副本: 4  
实际运行副本: 4

部署 (Deployment) 用来描述期望应用达到的目标状态，主要用来描述无状态应用，副本的数量和状态由其背后的控制器来维护，确保状态与定义的期望状态一致。您可以增加副本数量来满足更高负载；回滚部署的版本来消除程序的错误修改；创建自动伸缩器来弹性应对不同场景下的负载。

弹性伸缩

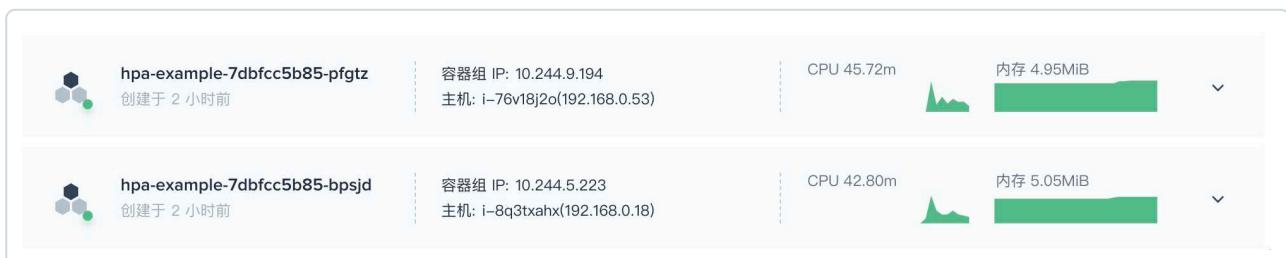
最小副本数 2 最大副本数 8 CPU 目标使用率 50% (当前 62%) 目标使用量 0 (当前 0)

容器组

请输入关键字过滤

容器组	IP	主机	CPU	内存
hpa-example-7dbfcc5b85-sd5b2	10.244.7.139	i-w03nj02(192.168.0.52)	47.85m	3.75MiB
hpa-example-7dbfcc5b85-4wh7k	10.244.6.201	i-1abu8ksg(192.168.0.67)	64.89m	3.48MiB

理论上，从容器组的 CPU 监控曲线中可以看到最初创建的 2 个容器组的 CPU 使用量有一个明显的升高趋势，待 HPA 开始工作时可以发现 CPU 使用量有明显降低的趋势，最终趋于平稳，而此时新增的 Pod 上可以看到 CPU 使用量在增加。



说明：HPA 工作后，Deployment 最终的副本数量可能需要几分钟才能稳定下来，删除负载后 Pod 数量回缩至正常状态也需要几分钟。由于环境的差异，不同环境中最终的副本数量可能与本示例中的数量不同。

## 停止负载

- 1、在左侧菜单栏选择 **工作负载 → 部署**，在部署列表中选择 **load-generator**，点击界面上方的 **删除**（或将之前设置的循环请求命令删除），停止负载；
- 2、再次查看 **hpa-example** 的运行状况，可以发现几分钟后它的 CPU 利用率已缓慢降到 18%，并且

HPA 将其副本数量最终减少至最小副本数 2，最终恢复了正常状态，从 CPU 使用量监控曲线反映的趋势也可以帮助我们进一步理解弹性伸缩的工作原理；

**注意：在完成本示例后，请将工作负载 load-generator 删除，防止其一直访问该应用而造成 CPU 资源的不必要的消耗或集群因资源不足而出现问题。**

The screenshot shows the KubeSphere interface for an HPA example. On the left, there's a sidebar with 'HPA 示例' (HPA Example) and deployment details like '项目: hpa', '应用: hpa-example', and '创建时间: 2018-12-13 15:24:35'. The main content area has tabs for '资源状态' (Resource Status), '版本控制' (Version Control), '监控' (Monitoring), '环境变量' (Environment Variables), and '事件' (Events).  
1. 副本运行状态: 显示 '期望副本: 2' 和 '实际运行副本: 2'。  
2. 弹性伸缩: 显示 '最小副本数: 2', '最大副本数: 8', '目标使用率: 50%' (当前 18%)，并有一个 '目标使用量' (0 当前 0) 的图标。  
3. 容器组: 列出了两个容器组：

- hpa-example-7dbfcc5b85-pfgtz: 容器组 IP: 10.244.9.194, 主机: i-76v18j2o(192.168.0.53), CPU: 3.52m, 内存: 4.95MiB
- hpa-example-7dbfcc5b85-bpsjd: 容器组 IP: 10.244.5.223, 主机: i-8q3txahx(192.168.0.18), CPU: 0.006m, 内存: 5.05MiB

每个容器组下方都有一个带有CPU和内存使用量的图表，以及一个表示当前CPU利用率的进度条。

3、在 Deployment 页面可以下钻到每个 Pod 的单个容器的监控详情，对比该容器的 CPU 使用量和网络流入、出速率监控曲线，与本示例的操作流程正好相符。



## 修改弹性伸缩

创建后若需要修改弹性伸缩的参数，可以在部署详情页，点击 **更多操作 → 弹性伸缩**，如下页面支持修改其参数：



## 取消弹性伸缩

若该部署无需设置 HPA，则可以在当前的部署详情页中，点击弹性伸缩右侧的 …，然后选择 取消。

The screenshot shows the deployment details for an 'nginx' deployment. At the top, there's a summary card with '1/1' instances, '副本运行状态' (Replica Running Status), '期望副本: 1' (Desired Replicas: 1), and '实际运行副本: 1' (Actual Running Replicas: 1). Below this is a detailed description of what a Deployment does. On the right, under the '弹性伸缩' (Horizontal Pod Autoscaler) section, there are three input fields: '最小副本数' (Min Replicas) set to 2, '最大副本数' (Max Replicas) set to 8, and '目标使用率' (Target Utilization) set to 50% (当前 0%). To the right of these fields is a dark blue button with a white 'X' icon and the text '取消' (Cancel), which is highlighted by a red arrow.

至此，您已经熟悉了如何在创建部署时设置弹性伸缩的基本操作。

## 示例六 – Jenkinsfile in SCM

Jenkinsfile in SCM 意为将 Jenkinsfile 文件本身作为源代码管理 (Source Control Management) 的一部分，因此可使用 `git clone` 或者其他类似的命令都能够获取此 Jenkinsfile，根据该文件内的流水线配置信息快速构建工程内的 CI/CD 功能模块，比如阶段 (Stage)，步骤 (Step) 和任务 (Job)。因此，在代码仓库中应包含 Jenkinsfile。

### 目的

本示例演示如何通过 GitHub 仓库中的 Jenkinsfile 来创建流水线，流水线共包括 8 个阶段，最终将一个文档网站部署到 KubeSphere 集群中的开发环境 (Dev) 和生产环境 (Production) 且能够通过公网访问，这两个环境在底层的 Kubernetes 是以项目 (Namespace) 为单位进行资源隔离的。

### 前提条件

- 本示例的代码仓库以 GitHub 和 DockerHub 为例，参考前确保已创建了 [GitHub](#) 和 [DockerHub](#) 账号。
- 已创建了企业空间和 DevOps 工程，若还未创建请参考 [管理员快速入门](#)。
- 熟悉 Git 分支管理和版本控制相关的基础知识，详见 [Git 官方文档](#)。

### 预估时间

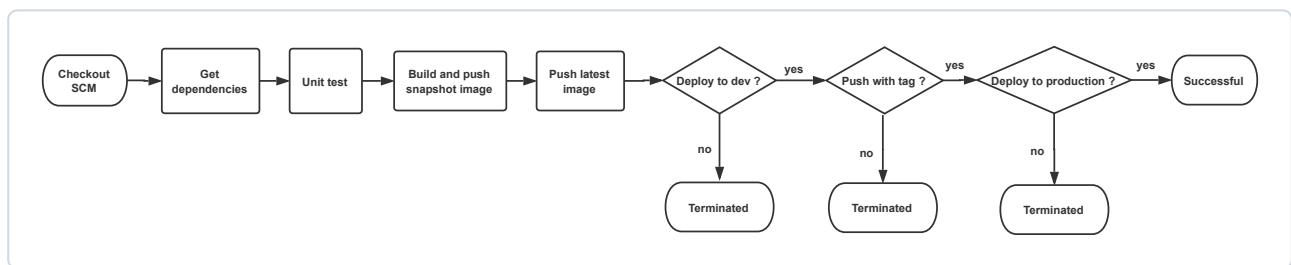
30–50 分钟 (时间由于环境的网速等因素而有所不同)。

### 操作示例

#### 演示视频

## 流水线概览

下面的流程图简单说明了整个 pipeline 的工作过程：



### 流程说明：

- 阶段一. **Checkout SCM**: 拉取 GitHub 仓库代码
- 阶段二. **Get dependencies**: 通过包管理器 [yarn](#) 安装项目的所有依赖
- 阶段三. **Unit test**: 单元测试，如果测试通过了才继续下面的任务
- 阶段四. **Build & push snapshot image**: 根据行为策略中所选择分支来构建镜像，并将 tag 为 `SNAPSHOT-$BRANCH_NAME-$BUILD_NUMBER` 推送至 DockerHub (其中 `$BUILD_NUMBER` 为 pipeline 活动列表的运行序号)。
- 阶段五. **Push latest image**: 将 master 分支打上 tag 为 latest，并推送至 DockerHub。
- 阶段六. **Deploy to dev**: 将 master 分支部署到 Dev 环境，此阶段需要审核。
- 阶段七. **Push with tag**: 生成 tag 并 release 到 GitHub，并推送到 DockerHub。
- 阶段八. **Deploy to production**: 将发布的 tag 部署到 Production 环境。

为演示方便，本示例就以添加本文档网站的 GitHub 仓库 [devops-docs-sample](#) 为例，可预先将其 Fork 至您的代码仓库中，并修改环境变量为实际参数。

## 创建凭证

在 [管理员快速入门](#) 中已给 project-regular 授予了 maintainer 的角色，因此使用 project-regular 登录 KubeSphere，进入已创建的 DevOps 工程，开始创建凭证。

本示例代码仓库中的 Jenkinsfile 需要用到 DockerHub、GitHub 和 Kubernetes (kubeconfig 用于访问接入正在运行的 Kubernetes 集群) 等一共 3 个凭证 (credentials)，这 3 个凭证 ID 需要与 Jenkinsfile

中前三个环境变量的值一致，先依次创建这三个凭证。

**注意：**若用户的凭证信息如账号或密码中包含了 @，\$ 这类特殊符号，可能在运行时无法识别而报错，这类情况需要用户在创建凭证时对密码进行 urlencode 编码，可通过一些第三方网站进行转换（比如 <http://tool.chinaz.com/tools/urlencode.aspx>），然后再将转换后的输出粘贴到对应的凭证信息中。

## 第一步：创建 DockerHub 凭证

1、进入之前创建的 DevOps 工程，在左侧的工程管理菜单下，点击 **凭证**，进入凭证管理界面。



2、点击 **创建**，创建一个用于 DockerHub 登录的凭证；

- 凭证 ID：必填，此 ID 将用于仓库中的 Jenkinsfile，此处命名为 **dockerhub-id**
- 类型：选择 **账户凭证**
- 用户名：填写您个人的 DockerHub 的用户名
- token / 密码：您个人的 DockerHub 的密码
- 描述信息：介绍凭证，比如此处可以备注为 DockerHub 登录凭证

完成后点击 **确定**。

创建凭证

凭证 ID \*

类型

账户凭证

用户名

token / 密码

描述信息

DockerHub 登录凭证

取消 确定

## 第二步：创建 GitHub 凭证

同上，创建一个用于 GitHub 的凭证，凭证 ID 命名为 `github-id`，类型选择 `账户凭证`，输入您个人的 GitHub 用户名和密码，备注描述信息，完成后点击 `确定`。

## 第三步：创建 kubeconfig 凭证

同上，在 `凭证` 下点击 `创建`，创建一个类型为 `kubeconfig` 的凭证，凭证 ID 命名为 `demo-kubeconfig`，完成后点击 `确定`。

**说明：**`kubeconfig` 类型的凭证用于访问接入正在运行的 Kubernetes 集群，在流水线部署步骤将用到该凭证。注意，此处的 `Content` 将自动获取当前 KubeSphere 中的 `kubeconfig` 文件内容，若部署至当前 KubeSphere 中则无需修改，若部署至其它 Kubernetes 集群，则需要将其

kubeconfig 文件的内容粘贴至 Content 中。

至此，3 个凭证已经创建完成，下一步需要在示例仓库中的 jenkinsfile 修改对应的三个凭证 ID 为用户自己创建的凭证 ID。



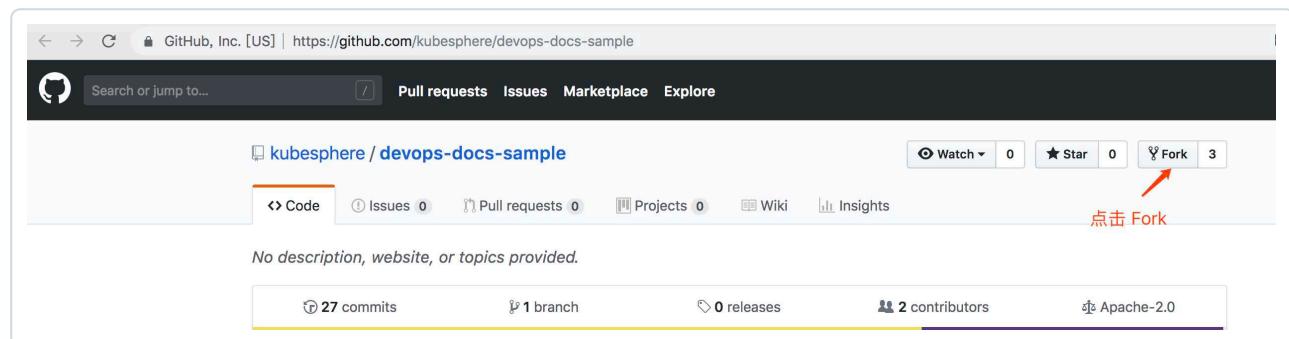
A screenshot of the KubeSphere UI showing a list of credentials. The table has columns: 名称 (Name), 类型 (Type), 描述信息 (Description), and 创建时间 (Created Time). There are three entries:

名称	类型	描述信息	创建时间
dockerhub-id	账户凭证	DockerHub登录凭证	2018-11-23 08:28:42
demo-kubeconfig	kubeconfig	Kubernetes	2018-11-23 08:44:48
github-id	账户凭证	GitHub登录凭证	2018-11-23 08:45:26

## 修改 Jenkinsfile

### 第一步：Fork 项目

登录 GitHub，将本示例用到的 GitHub 仓库 [devops-docs-sample](#) Fork 至您个人的 GitHub。

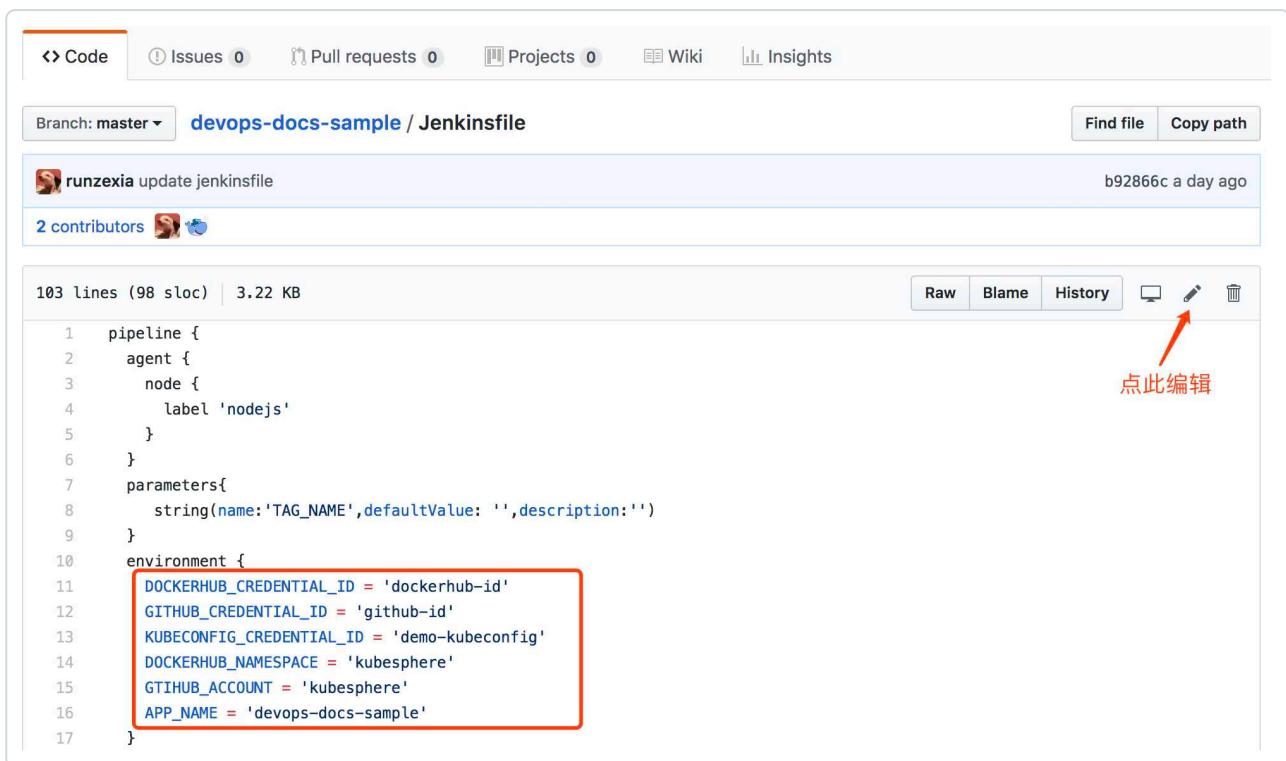


### 第二步：修改 Jenkinsfile

1、Fork 至您个人的 GitHub 后，在 **根目录** 进入 **Jenkinsfile**。

__mocks__	init	a month ago
content	rm static, update yarn lock	21 days ago
deploy	Update docs-sample-svc.yaml	19 days ago
plugins	init	a month ago
src	init	a month ago
.dockerignore	init	a month ago
.gitignore	init	a month ago
.prettierrc	init	a month ago
.travis.yml	init	a month ago
CNAME	init	a month ago
Dockerfile	init	a month ago
Jenkinsfile	rollback	19 days ago
LICENSE	init	a month ago
README.md	init	a month ago
docker_push	init	a month ago
gatsby-config.js	init	a month ago

2、在 GitHub UI 点击编辑图标，需要修改如下环境变量 (environment) 的值。



The screenshot shows the GitHub repository page for 'devops-docs-sample / Jenkinsfile'. The file has 103 lines (98 sloc) and 3.22 KB. A red arrow points to the edit icon (pencil) in the top right corner of the code editor. Another red box highlights a block of environment variable assignments:

```

1 pipeline {
2   agent {
3     node {
4       label 'nodejs'
5     }
6   }
7   parameters{
8     string(name:'TAG_NAME',defaultValue: '',description='')
9   }
10 environment {
11   DOCKERHUB_CREDENTIAL_ID = 'dockerhub-id'
12   GITHUB_CREDENTIAL_ID = 'github-id'
13   KUBECONFIG_CREDENTIAL_ID = 'demo-kubeconfig'
14   DOCKERHUB_NAMESPACE = 'kubesphere'
15   GTIHUB_ACCOUNT = 'kubesphere'
16   APP_NAME = 'devops-docs-sample'
17 }

```

修改项	值	含义
DOCKERHUB_CREDENTIAL_ID	dockerhub-id	填写创建凭证步骤中的 DockerHub 凭证 ID，用于登录您的 DockerHub

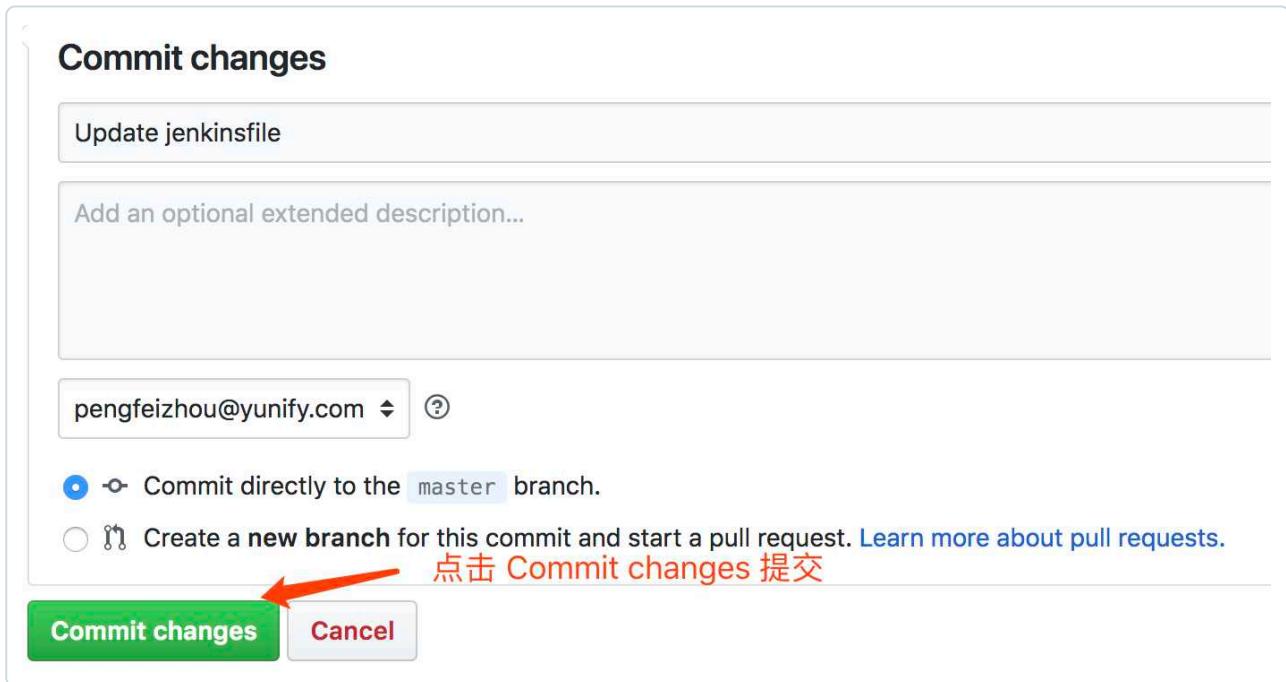
修改项	值	含义
GITHUB_CREDENTIAL_ID	github-id	填写创建凭证步骤中的 GitHub 凭证 ID，用于推送 tag 到 GitHub 仓库
KUBECONFIG_CREDENTIAL_ID	demo-kubeconfig	kubeconfig 凭证 ID，用于访问接入正在运行的 Kubernetes 集群
DOCKERHUB_NAMESPACE	your-dockerhub-account	替换为您的 DockerHub 账号名 (它也可以是账户下的 Organization 名称)
GITHUB_ACCOUNT	your-github-account	替换为您的 GitHub 账号名 (它也可以是账户下的 Organization 名称)
APP_NAME	devops-docs-sample	应用名称

```

...
environment {
    DOCKERHUB_CREDENTIAL_ID = 'dockerhub-id'
    GITHUB_CREDENTIAL_ID = 'github-id'
    KUBECONFIG_CREDENTIAL_ID = 'demo-kubeconfig'
    DOCKERHUB_NAMESPACE = 'your-dockerhub-account'
    GITHUB_ACCOUNT = 'your-github-account'
    APP_NAME = 'devops-docs-sample'
}
...

```

3、修改以上的环境变量后，点击 **Commit changes**，将更新提交到当前的 master 分支。



## 创建项目

CI/CD 流水线会根据文档网站项目的 [yaml 模板文件](#)，最终将文档网站分别部署到 Dev 和 Production 这两个项目 (Namespace) 环境中，即 `kubesphere-docs-dev` 和 `kubesphere-docs-prod`，这两个项目需要预先在控制台依次创建，参考如下步骤创建该项目。

### 第一步：填写项目信息

回到工作台，在之前创建的企业空间 (demo-workspace) 下，点击 **项目** → **创建**，创建一个 **资源型项目**，作为本示例的开发环境，填写该项目的基本信息，完成后点击 **下一步**。

- 名称：固定为 `kubesphere-docs-dev`，若需要修改项目名称则需在 [yaml 模板文件](#) 中修改 namespace
- 别名：可自定义，比如 **开发环境**
- 描述信息：可简单介绍该项目，方便用户进一步了解

创建项目

基本信息 高级设置

### 基本信息

项目基础信息设置

名称 \*  别名

最长 63 个字符，只能包含小写字母、数字及分隔符(“-”), 且必须以小写字母或数字开头及结尾。

描述信息

## 第二步：高级设置

本示例暂无资源请求和限制，因此高级设置中无需修改默认值，点击 **创建**，项目可创建成功。

项目 DevOps 工程

列表 (1)

名称	成员	操作
kubesphere-docs-dev(开发环境) jenkinsfile-in-scm	project-regular 项目管理员	2018-12-30 17:07:23 创建时间

## 创建第二个项目

同上，参考上一步创建一个名称为 `kubesphere-docs-prod` 的项目，作为生产环境。

**说明：**当 CI/CD 流水线后续执行成功后，在 `kubesphere-docs-dev` 和 `kubesphere-docs-prod` 项目中将看到流水线创建的部署 (Deployment) 和服务 (Service)。

项目 DevOps 工程 搜索框 列表 (3)

kubesphere-docs-prod(生产环境)  
kubesphere-docs-dev(开发环境)

成员: 成员: 成员:

## 创建流水线

参考以下步骤，创建并运行一个完整的流水线。

### 第一步：填写基本信息

1、进入已创建的 DevOps 工程，选择左侧菜单栏的 流水线，然后点击 创建。

工作台 应用模板 KUBESPHERE project-regular

DevOps 工程 devops-demo

流水线

Pipeline 是一系列的插件集合，可以通过组合它们来实现持续集成和持续交付的功能。Pipeline DSL为我们提供了一个可扩展的工具集，让我们可以将简单到复杂的逻辑通过代码实现。

官网文档 参考文档

创建

流水线 工程管理 基本信息 凭证 成员角色 工程成员

2、在弹出的窗口中，输入流水线的基本信息。

- 名称：为创建的流水线起一个简洁明了的名称，便于理解和搜索
- 描述信息：简单介绍流水线的主要特性，帮助进一步了解流水线的作用
- 代码仓库：点击选择代码仓库，代码仓库需已存在 Jenkinsfile

基本信息

请输入流水线的基本信息

名称 \* Jenkinsfile-in-SCM

项目 project-RyonypM4PX2q

Pipeline 的名称，同一个项目内 Pipeline 不能重名  
将根据项目进行资源进行分组，可以按项目对资源进行查看管理

描述信息 DevOps demo

代码仓库(可选)

点击添加代码仓库 → 请选择一个代码仓库作为 Pipeline 的代码源



## 第二步：添加仓库

1、点击代码仓库，以添加 Github 仓库为例。

2、点击弹窗中的 [获取 Token](#)。

创建流水线

基本信息 高级设置

◀ 选择代码仓库

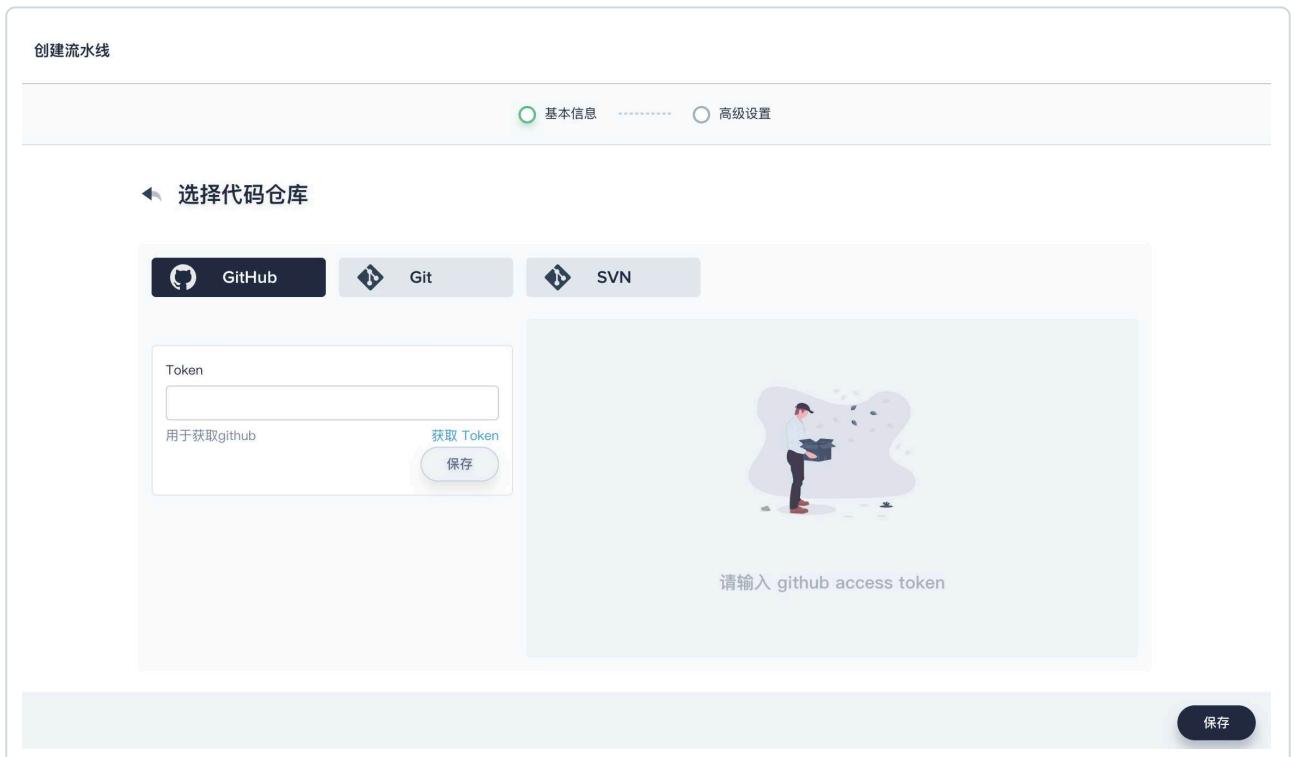
**GitHub** Git SVN

Token

用于获取github 获取 Token 保存

请输入 github access token

保存



3、在 GitHub 的 access token 页面填写 Token description，简单描述该 token，如 DevOps demo，

在 Select scopes 中无需任何修改，点击 [Generate token](#)，GitHub 将生成一串字母和数字组成的 token 用于访问当前账户下的 GitHub repo。

The screenshot shows the GitHub developer settings page under 'Personal access tokens'. A new token is being created with the description 'DevOps demo'. The 'Select scopes' section is visible, stating that scopes define access for personal tokens. A note at the bottom says 'Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)'.

4、复制生成的 token，在 KubeSphere Token 框中输入该 token 然后点击保存。

5、验证通过后，右侧会列出此 Token 关联用户的所有代码库，选择其中一个带有 Jenkinsfile 的仓库。比如此处选择准备好的示例仓库 [devops-docs-sample](#)，点击 [选择此仓库](#)，完成后点击 [下一步](#)。

The screenshot shows the 'Select Repository' step in the KubeSphere setup. It lists repositories under three categories: GitHub, Git, and SVN. Under GitHub, there are three repositories: 'FeynmanZhou' (selected), 'openpitrix', and 'kubesphere'. Under Git, there are three repositories: 'devops-docs-sample' (with a red arrow pointing to the 'Select This Repository' button), 'docs.kubesphere.io', and 'docs.openpitrix.io'. The 'Select This Repository' button is highlighted with a red arrow.

### 第三步：高级设置

完成代码仓库相关设置后，进入高级设置页面，高级设置支持对流水线的构建记录、行为策略、定期扫描等设置的定制化，以下对用到的相关配置作简单释义。

1、构建设置中，勾选 [丢弃旧的构建](#)，此处的 **保留构建的天数** 和 **保持构建的最大个数** 默认为 -1。



### 说明：

构建设置的保留构建的天数和保持构建的最大个数两个选项可以同时对构建进行作用，如果超出任一限制，则将丢弃超出该限制的任何构建。默认两个值为 -1，表示不自动删除构建。

丢弃旧的构建将确定何时应丢弃项目的构建记录。构建记录包括控制台输出，存档工件以及与特定构建相关的其他元数据。保持较少的构建可以节省 Jenkins 所使用的磁盘空间，我们提供了两个选项来确定应何时丢弃旧的构建：

- 保留构建的天数：如果构建达到一定的天数，则丢弃构建。
- 保留构建的个数：如果已经存在一定数量的构建，则丢弃最旧的构建。这两个选项可以同时对构建进行作用，如果超出任一限制，则将丢弃超出该限制的任何构建。

2、行为策略中，KubeSphere 默认添加了三种策略。由于本示例还未用到 **从 Fork 仓库中发现 PR** 这条策略，此处可以删除该策略，点击右侧删除按钮。

行为策略

添加操作

基本信息 高级设置

发现分支  
策略  
排除也作为 PR 提交的分支

从原仓库中发现 PR  
拉取策略  
PR 本身的源代码版本

从 Fork 仓库中发现 PR  
拉取策略  
PR 本身的源代码版本 可信用户  
管理员或有编辑权限的用户

**说明：**

支持添加三种类型的发现策略。需要说明的是，在 Jenkins 流水线被触发时，开发者提交的 PR (Pull Request) 也被视为一个单独的分支。

**发现分支：**

- 排除也作为 PR 提交的分支：选择此项表示 CI 将不会扫描源分支（比如 Origin 的 master branch），也就是需要被 merge 的分支
- 只有被提交为 PR 的分支：仅扫描 PR 分支
- 所有分支：拉取的仓库 (origin) 中所有的分支

**从原仓库中发现 PR：**

- PR 与目标分支合并后的源代码版本：一次发现操作，基于 PR 与目标分支合并后的源代码版本创建并运行流水线
- PR 本身的源代码版本：一次发现操作，基于 PR 本身的源代码版本创建并运行流水线
- 当 PR 被发现时会创建两个流水线，一个流水线使用 PR 本身的源代码版本，一个流水线使用 PR 与目标分支合并后的源代码版本：两次发现操作，将分别创建两条流水线，第一条流水线使用 PR 本身的源代码版本，第二条流水线使用 PR 与目标分支合并后的源代码版本

3、默认的 **脚本路径** 为 Jenkinsfile，此处无需修改。

**注：**路径是 Jenkinsfile 在代码仓库的路径，表示它在示例仓库的根目录，若文件位置变动则需修改其脚本路径。

4、在 **扫描 Repo Trigger** 勾选 **如果没有扫描触发，则定期扫描**，扫描时间间隔可根据团队习惯设定，本示例设置为 **5 minutes**。

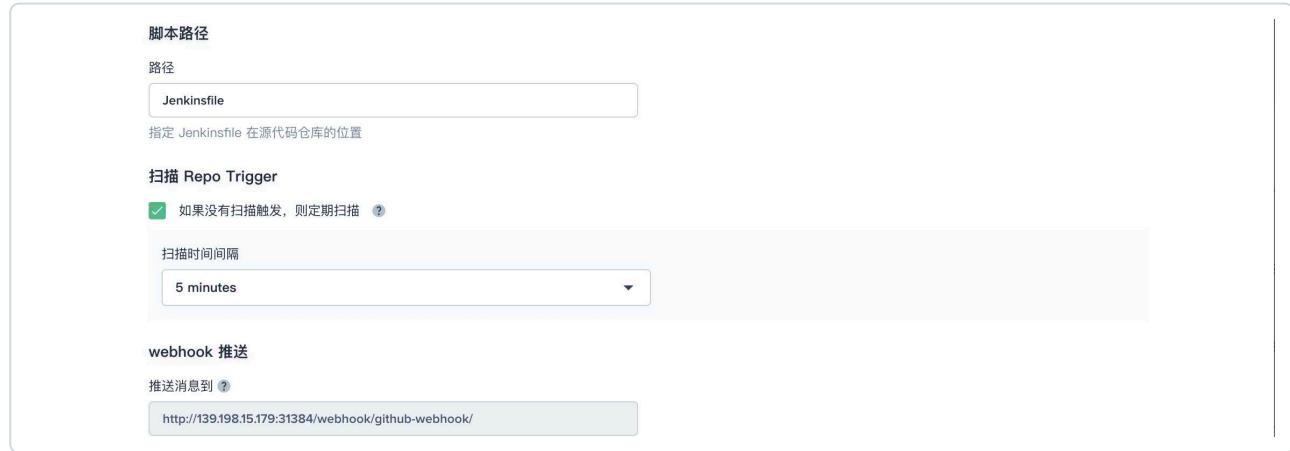
**说明：**定期扫描是设定一个周期让流水线周期性地扫描远程仓库，根据 行为策略 查看仓库有没有代码更新或新的 PR。

**Webhook 推送：**

Webhook 是一种高效的方式可以让流水线发现远程仓库的变化并自动触发新的运行，GitHub 和 Git (如 Gitlab) 触发 Jenkins 自动扫描应该以 Webhook 为主，以上一步在 KubeSphere 设置定期扫描为辅。在本示例中，可以通过手动运行流水线，如需设置自动扫描远端分支并触发运行，详

见 [设置自动触发扫描 – GitHub SCM](#)。

完成高级设置后点击 **创建**。



## 第四步：运行流水线

流水线创建后，点击浏览器的 **刷新** 按钮，可见一条自动触发远程分支后的运行记录。

- 1、点击右侧 **运行**，将根据上一步的 **行为策略** 自动扫描代码仓库中的分支，在弹窗选择需要构建流水线的 **master** 分支，系统将根据输入的分支加载 Jenkinsfile (默认是根目录下的 Jenkinsfile)。
- 2、由于仓库的 Jenkinsfile 中 **TAG\_NAME: defaultValue** 没有设置默认值，因此在这里的 **TAG\_NAME** 可以输入一个 tag 编号，比如输入 v0.0.1。
- 3、点击 **确定**，将新生成一条流水线活动开始运行。

**说明:** tag 用于在 GitHub 和 DockerHub 中分别生成带有 tag 的 release 和镜像。注意: 在主动运行流水线以发布 release 时，**TAG\_NAME** 不应与之前代码仓库中所存在的 tag 名称重复，如果重复会导致流水线的运行失败。



至此，Jenkinsfile in SCM 已完成创建并开始运行。

**注：点击 分支 切换到分支列表，查看流水线具体是基于哪些分支运行，这里的分支则取决于上一步 行为策略 的发现分支策略。**

## 第五步：审核流水线

为方便演示，此处默认用当前账户来审核，当流水线执行至 `input` 步骤时状态将暂停，需要手动点击 `继续`，流水线才能继续运行。注意，在 Jenkinsfile 中分别定义了三个阶段 (stage) 用来部署至 Dev 环境和 Production 环境以及推送 tag，因此在流水线中依次需要对 `deploy to dev, push with tag, deploy to production` 这三个阶段审核 3 次，若不审核或点击 `终止` 则流水线将不会继续运行。



**说明：**在实际的开发生产场景下，可能需要更高权限的管理员或运维人员来审核流水线和镜像，并决定是否允许将其推送至代码或镜像仓库，以及部署至开发或生产环境。Jenkinsfile 中的 `input` 步骤支持指定用户审核流水线，比如要指定用户名为 `project-admin` 的用户来审核，可以在 Jenkinsfile 的 `input` 函数中追加一个字段，如果是多个用户则通过逗号分隔，如下所示：

...

```
input(id: 'release-image-with-tag', message: 'release image with tag?', submitter: 'project-admin,project-admin')
```

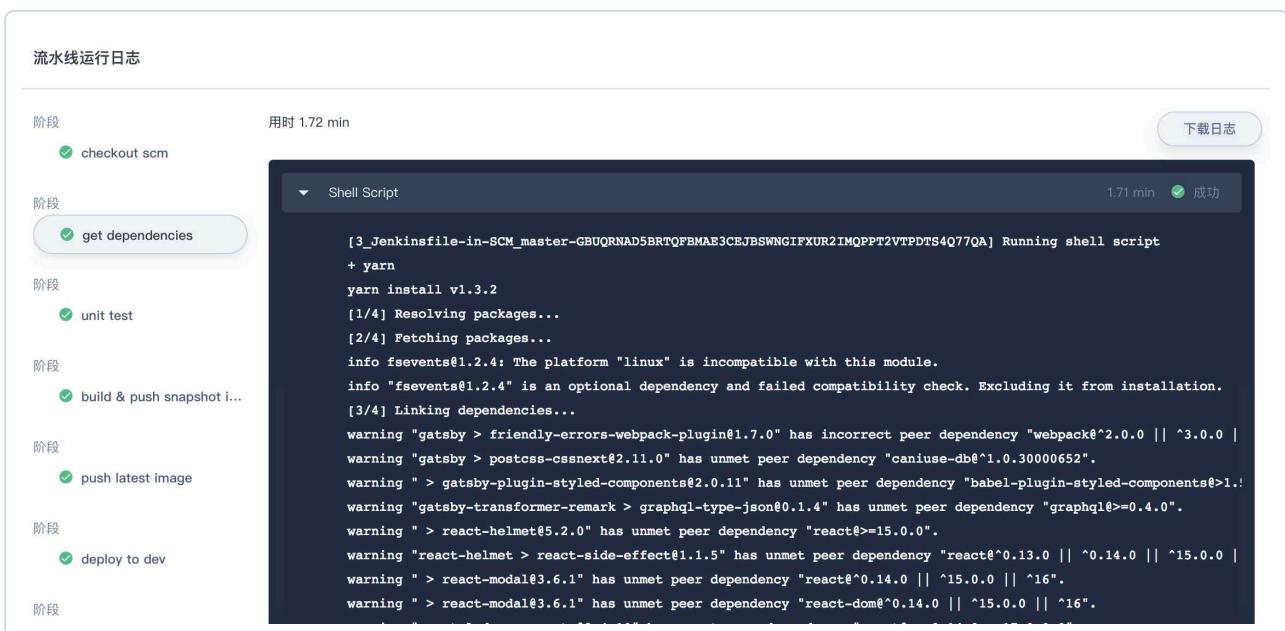
...

## 查看流水线

1、点击流水线中 `活动` 列表下当前正在运行的流水线序列号，页面展现了流水线中每一步骤的运行状态，注意，流水线刚创建时处于初始化阶段，可能仅显示日志窗口，待初始化（约一分钟）完成后即可看到流水线。黑色框标注了流水线的步骤名称，示例中流水线共 8 个 stage，分别在 [Jenkinsfile](#) 中被定义。



2、当前页面中点击右上方的 **查看日志**，查看流水线运行日志。页面展示了每一步的具体日志、运行状态及时间等信息，点击左侧某个具体的阶段可展开查看其具体日志。日志可下载至本地，如出现错误，下载至本地更便于分析定位问题。



## 验证运行结果

若流水线的每一步都能执行成功，那么流水线最终 build 的 Docker 镜像也将被成功地 push 到 DockerHub 中，我们在 Jenkinsfile 中已经配置过 DockerHub，登录 DockerHub 查看镜像的 push 结果，可以看到 tag 为 snapshot、TAG\_NAME(v0.0.1)、latest 的镜像已经被 push 到 DockerHub，并且在 GitHub 中也生成了一个新的 tag 和 release。文档网站最终将以 deployment 和 service 分别部署到 KubeSphere 的 `kubesphere-docs-dev` 和 `kubesphere-docs-prod` 项目环境中。

环境	访问地址	所在项目 (Namespace)	部署 (Deployment)	服务 (Service)
Dev	公网 IP : 30860 ( \${EIP}:\${NODEPORT})	kubesphere-docs-dev	ks-docs-sample-dev	ks-docs-sample-dev
Production	公网 IP : 30960 ( \${EIP}:\${NODEPORT})	kubesphere-docs-prod	ks-docs-sample	ks-docs-sample

1、可通过 KubeSphere 回到项目列表，依次查看之前创建的两个项目中的部署和服务的状态。例如，以下查看 `kubesphere-docs-prod` 项目下的部署。

进入该项目，在左侧的菜单栏点击 **工作负载 → 部署**，可以看到 `ks-docs-sample` 已创建成功。正常情况下，部署的状态应该显示 **运行中**。

## 查看部署

The screenshot shows the KubeSphere interface for the 'kubesphere-docs-prod' project. On the left, there's a sidebar with '概览' (Overview), '应用' (Application), and '工作负载' (Workload) sections. Under '工作负载', '部署' (Deployment) is selected. The main area is titled '部署' (Deployments) and contains the following information:

- Count: 1 已用配额 (1 used quota)
- Count: ∞ 规划配额 (∞ planned quota)
- Count: ∞ 剩余 Pod 配额 (∞ remaining Pod quota)
- Search bar: 输入查询条件进行过滤 (Input filtering conditions)
- Filtering options: 名称 (Name), 状态 (Status), 应用 (Application), 更新时间 (Last updated)
- Table rows:
 

名称	状态	更新时间	操作
ks-docs-sample	运行中 (2/2)	2019-01-04 16:43:11	⋮

2、在菜单栏中选择 **网络与服务 → 服务** 也可以查看对应创建的服务，可以看到该服务对外暴露的节点端口 (NodePort) 是 `30960`。

## 查看服务

The screenshot shows the KubeSphere interface for the 'kubesphere-docs-prod' project. On the left, there's a sidebar with '官网文档' (Official Document) and '参考文档' (Reference Document). The main area is titled '服务' (Services) and contains the following information:

- Count: 1 已用配额 (1 used quota)
- Count: 1 虚拟 IP (1 virtual IP)
- Count: 0 Headless (0 headless services)
- Search bar: 输入查询条件进行过滤 (Input filtering conditions)
- Filtering options: 名称 (Name), IP地址 (IP Address), 端口 (Port), 应用 (Application), 创建时间 (Creation Time)
- Table rows:
 

名称	IP地址	端口	应用	创建时间	操作
ks-docs-sample	Virtual IP: 10.233.11.210	端口: 80:80/TCP 节点端口: 30960 TCP		2019-01-29 10:26:20	⋮

3、查看推送到您个人的 DockerHub 中的镜像，可以看到 `devops-docs-sample` 就是 APP\_NAME 的

值，而 `v0.0.1`, `lastest`, `SNAPSHOT-master-2` 就是在 `jenkinsfile` 中定义的 tag。

The screenshot shows the 'Tags' tab of a GitHub repository. It displays 1-25 of 45 tags. The tags listed are `v0.0.1`, `latest`, and `SNAPSHOT-master-2`. Each tag entry includes its name, size (8 MB), the last updated time (2 hours ago), and a three-dot menu icon.

4、点击 `release`，查看 Fork 到您个人 GitHub repo 中的 `v0.0.1` tag 和 release，它是由 `jenkinsfile` 中的 `push with tag` stage 生成的。

The screenshot shows the main GitHub repository page for `FeynmanZhou / devops-docs-sample`. It highlights the '1 release' button under the repository statistics. A red arrow points to this button. The statistics also show 46 commits, 1 branch, 4 contributors, and Apache-2.0 license.

The screenshot shows the 'Releases' tab of the GitHub repository page. It displays a single release named `v0.0.1`, which was created 2 hours ago. The release contains a zip file (8f01176) and a tar.gz file.

5、若需要在外网访问，可能需要进行端口转发并开放防火墙，即可访问成功部署的文档网站示例的首页，以访问生产环境 `ks-docs-sample` 服务的 `30960` 端口为例。

例如，在QingCloud云平台上，如果使用了VPC网络，则需要将KubeSphere集群中的任意一台主机上暴露的节点端口（NodePort）**30960**在VPC网络中添加端口转发规则，然后在防火墙放行该端口。

## 添加端口转发规则



The screenshot shows a table of port forwarding rules. The columns are: Name, Protocol, Source Port, Internal IP, Internal Port, Created At, and Operation. One rule is listed:

名称	协议	源端口	内网 IP	内网端口	创建于	操作
demo6-docs-nodeport	tcp	30960	192.168.0.18	30960	1分钟前	禁用

## 防火墙添加下行规则



The screenshot shows a table of firewall rules. The columns are: Name, Priority, Protocol, Behavior, Start Port (?), End Port (?), Source IP, and Operation. One rule is listed:

名称	优先级	协议	行为	起始端口 (?)	结束端口 (?)	源IP	操作
demo6-docs-nodeport	10	TCP	接受	30960			禁用

## 访问示例服务

在浏览器访问部署到KubeSphere Dev和Production环境的服务：

### Dev环境

访问<http://127.0.0.1:30860/>或者<http://EIP:30860/>。



The screenshot shows a browser window displaying the KubeSphere Express v1.0.0 documentation. The URL in the address bar is <http://127.0.0.1:30860/express/zh-CH/basic/>. The page has a dark theme with a sidebar on the left containing links like '简介', '安装指南', '快速入门', '用户指南', and '附录'. The main content area features a title 'KubeSphere 简介' and a section '产品介绍' with a detailed description of what KubeSphere is and its features.

## Production 环境

访问 <http://127.0.0.1:30960/> 或者 <http://EIP:30960/>。



至此，创建一个 Jenkinsfile in SCM 类型的流水线已经完成了，若创建过程中遇到问题，可参考 [常见问题](#)。

## 示例七 – Jenkinsfile out of SCM

上一篇文档示例六是通过代码仓库中的 Jenkinsfile 构建流水线，需要对声明式的 Jenkinsfile 有一定的基础。而 Jenkinsfile out of SCM 不同于 [Jenkinsfile in SCM](#)，其代码仓库中可以无需 Jenkinsfile，支持用户在控制台通过可视化的方式构建流水线或编辑 Jenkinsfile 生成流水线，用户操作界面更友好。

### 目的

本示例演示基于 [示例六 – Jenkinsfile in SCM](#)，通过可视化构建流水线（包含示例六的前六个阶段），最终将一个文档网站部署到 KubeSphere 集群中的开发环境且能够通过公网访问，这里所谓的开发环境在底层的 Kubernetes 里是以项目（Namespace）为单位进行资源隔离的。若熟悉了示例六的流程后，对于示例七的手动构建步骤就很好理解了。为方便演示，本示例仍然以 GitHub 代码仓库 [devops-docs-sample](#) 为例。

### 前提条件

- 已有 [DockerHub](#) 的账号。
- 已创建了企业空间和 DevOps 工程，若还未创建请参考 [管理员快速入门](#)。
- 熟悉 Dockerfile 常用命令，可参考 [Docker 官方文档](#)。

### 预估时间

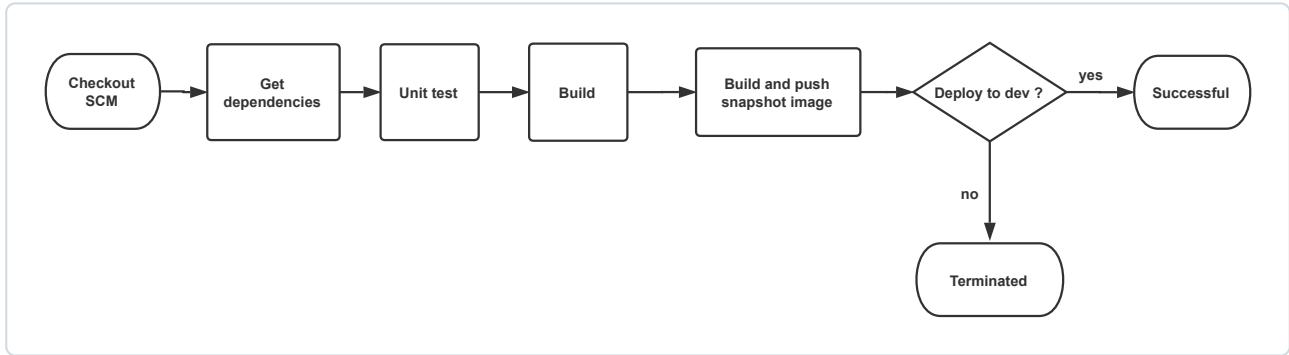
约 30 分钟。

### 操作示例

#### 演示视频

## 流水线概览

构建可视化流水线共包含以下 6 个阶段 (stage)，先通过一个流程图简单说明一下整个 pipeline 的工作流：



详细说明每个阶段所执行的任务：

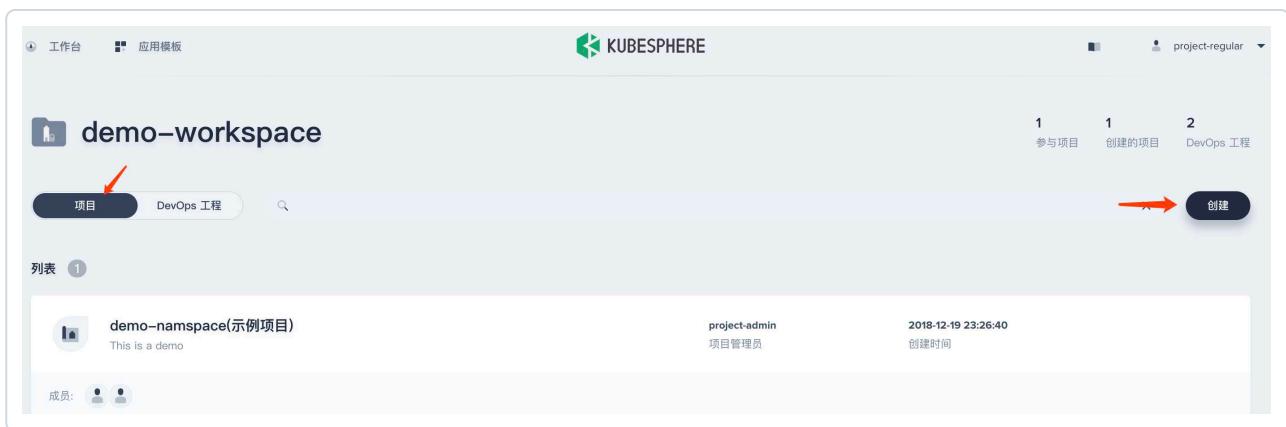
- 阶段一. **Checkout SCM**: 拉取 GitHub 仓库代码
- 阶段二. **Get dependencies**: 通过包管理器 `yarn` 安装项目的所有依赖
- 阶段三. **Unit test**: 单元测试，如果测试通过了才继续下面的任务
- 阶段四. **Build**: 执行项目中 `package.json` 的 scripts 里的 build 命令，本例中用于生成静态网站。
- 阶段五. **Build & push snapshot image**: 构建镜像，并将 tag 为 `SNAPSHOT-$BRANCH_NAME-$BUILD_NUMBER` 推送至 DockerHub (其中 `$BUILD_NUMBER` 为 pipeline 活动列表的运行序号)。
- 阶段六. **Deploy to dev**: 将 master 分支部署到 Dev 环境，此阶段需要审核。

## 创建项目

CI/CD 流水线会根据文档网站的 [yaml 模板文件](#)，最终将该网站部署到开发环境 `kubesphere-docs-dev`，它对应的是 KubeSphere 中的一个项目 (Namespace)，该项目需要预先在控制台创建。注意，若您已在 [示例六](#) 创建过该项目，则无需再次创建，可跳过创建项目步骤。

## 第一步：填写项目信息

1、以管理员快速入门中的 project-regular 登录 KubeSphere，在已创建的企业空间下，点击 项目 → 创建。



The screenshot shows the KubeSphere interface with the 'demo-workspace' project selected. The top navigation bar includes '工作台' and '应用模板' on the left, the 'KUBESPHERE' logo in the center, and 'project-regular' on the right. Below the navigation is a summary bar with counts: 1 参与项目, 1 创建的项目, and 2 DevOps 工程. The main area displays the 'demo-workspace' project details, including its name, description ('This is a demo'), owner ('project-admin 项目管理员'), creation time ('2018-12-19 23:26:40'), and members. A red arrow points to the '项目' tab in the navigation bar, and another red arrow points to the '创建' button in the top right corner.

2、参考如下提示，填写项目的基本信息，完成后点击 下一步。

- 名称：固定为 `kubesphere-docs-dev`，若需要修改项目名称则需先在 [yaml 模板文件](#) 中修改 namespace
- 别名：可自定义，比如 [开发环境](#)
- 描述信息：可简单介绍该项目，方便用户进一步了解



The screenshot shows the 'Create Project' form. At the top, there are two tabs: '基本信息' (selected) and '高级设置'. Below the tabs, the '基本信息' section is titled '基本信息' and includes a note '项目基础信息设置'. It has three input fields: '名称 \*' (filled with 'kubesphere-docs-dev'), '别名' (filled with '开发环境'), and '描述信息' (filled with 'Jenkinsfile in SCM demo'). There is also a note below the name field: '最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾'.

## 第二步：高级设置

本示例暂无资源请求和限制，因此高级设置中无需修改默认值，点击 **创建**，项目可创建成功。



**说明：**当 CI/CD 流水线后续执行成功后，在 **kubesphere-docs-dev** 项目中将看到流水线创建的部署（Deployment）和服务（Service）。

## 创建凭证

本示例创建流水线时需要访问 DockerHub 和 Kubernetes (创建 KubeConfig 用于接入正在运行的 Kubernetes 集群) 的 2 个凭证 (credentials)，先依次创建这 2 个凭证。

**注意：**

- 若示例七与示例六在同一 DevOps 工程，且示例六已创建了这两个凭证，则无需再次创建，可跳过创建凭证步骤，因为同一 DevOps 工程下的凭证对多个流水线是共用的。
- 若代码仓库属于私有仓库如 Gitlab 或 SVN，可能还需要再创建这类账户的凭证。

## 第一步：创建 DockerHub 凭证

1、在左侧的工程管理菜单下。点击 **凭证**，进入凭证管理界面，界面会展示当前工程所需的所有可用凭证。



2、点击创建按钮，创建一个用于 DockerHub 登录的凭证，完成后点击 确定。

- 凭证 ID：必填，此 ID 将用于仓库中的 Jenkinsfile，可自定义，此处命名为 `dockerhub-id`
- 类型：选择 **账户凭证**
- 用户名/密码：输入您个人的 DockerHub 用户名和密码
- 描述信息：介绍凭证，比如此处可以备注为 DockerHub 登录凭证

### 创建凭证

凭证 ID \*

类型

账户凭证

用户名

token / 密码

描述信息

DockerHub 登录凭证

取消确定

## 第二步：创建 kubeconfig 凭证

同上，在 **凭证** 下点击 **创建**，创建一个类型为 **kubeconfig** 的凭证。凭证 ID 命名为 **demo-kubeconfig**

**说明：** `kubeconfig` 用于访问接入正在运行的 Kubernetes 集群，在流水线部署步骤将用到该凭证。注意，此处的 Content 将自动获取当前 KubeSphere 中的 kubeconfig 文件内容，若部署至当前 KubeSphere 中则无需修改，若部署至其它 Kubernetes 集群，则需要将其 kubeconfig 文件（一般在 `$HOME/.kube/config`）的内容粘贴至 Content 中，完成后点击 确定。

至此，2 个凭证已经创建完成。

名称	类型	描述信息	创建时间
dockerhub-id	账户凭证	DockerHub登录凭证	2018-11-23 08:28:42
demo-kubeconfig	kubeconfig	Kubernetes	2018-11-23 08:44:48

## 创建流水线

参考以下步骤，创建并运行一个完整的流水线。

### 第一步：填写基本信息

1、进入已创建的 DevOps 工程，选择左侧 流水线 菜单项，然后点击 创建。

名称	状态	健康状态	分支	PullRequest
Jenkinsfile-in-SCM	0 分支成功	健康	1	0

2、在弹出的窗口中，输入流水线的基本信息，完成后点击 下一步。

- 名称：为流水线起一个简洁明了的名称，便于理解和搜索
- 描述信息：简单介绍流水线的主要特性，帮助进一步了解流水线的作用
- 代码仓库：此处不选择代码仓库

创建流水线

基本信息

请输入流水线的基本信息

名称 \* Jenkinsfile-out-of-SCM  
Pipeline的名称,同一个项目内pipeline 不能重名

项目 project-MGppvJ8D2Y93  
将根据项目进行资源进行分组, 可以按项目对资源进行查看管理

描述信息 This is a demo

代码仓库(可选)  
请选择一个代码仓库作为pipeline的代码源

下一步

## 第二步：高级设置

1、填写基本信息后，进入高级设置页面。此处需勾选 **丢弃旧的构建**，本示例中 **保留构建的天数** 设置为 **1**，**保持构建的最大个数** 设置为 **3**(这个数值大小可根据团队习惯来设置)。

**说明：**高级设置支持对流水线的构建记录、参数化构建、定期扫描等设置的定制化。例如 **丢弃旧的构建** 可以决定何时应丢弃项目的构建记录。构建记录包括控制台输出，存档工件以及与特定构建相关的其他元数据。保持较少的构建可以节省 Jenkins 所使用的磁盘空间。

构建设置

丢弃旧的构建 ?

保留构建的天数 1  
如果构建达到保留的天数将进行删除.. (默认值 -1: 不会丢弃记录)

保持构建的最大个数 3  
如果构建超过一定的数量将丢弃旧的构建. (默认值 -1: 不会丢弃记录)

2、点击 **添加参数**，如下添加 **两个** 字符串参数，将在流水线的 docker 命令中使用该参数。

参数类型	名称	默认值	描述信息
字符串参数 (string)	DOCKERHUB_NAMESPACE	填写您的 DockerHub 账号 (它也可以是账户下的 Organization 名称)	DockerHub Namespace
字符串参数 (string)	APP_NAME	填写 devops-docs-sample	Application Name

编辑流水线

参数化构建 ②

字符串参数 (string)  
名称: DOCKERHUB\_NAMESPACE  
默认值: pengfeizhou  
描述信息: DockerHub Namespace

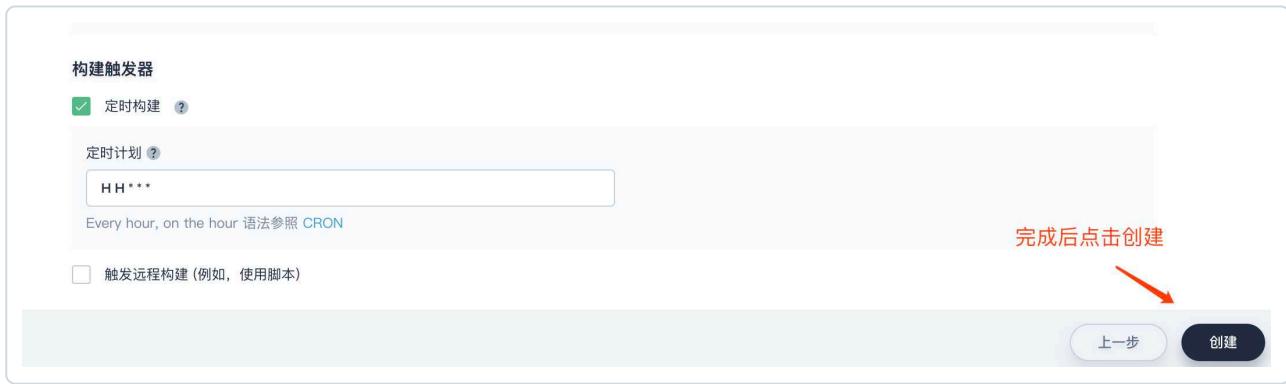
字符串参数 (string)  
名称: APP\_NAME  
默认值: devops-docs-sample  
描述信息: Application Name

添加参数 创建两个字符串参数

- 字符串参数 (string)
- 文本(Text)
- 布尔值(Boolean)
- 选项参数(Choice)
- 密码参数>Password)

3、设置 构建触发器，此处勾选 定时构建，日程表 (cron) 填写 `H H * * *`，表示每天构建一次 (不限具体时刻)。完成后点击 **创建**，创建完成后页面将自动跳转至流水线的可视化编辑页面。

**说明：**这里的定时构建是提供类似 Linux cron 的功能来定期执行此流水线，定时构建的语法以下作一个简单释义，语法详见 [Jenkins 官方文档](#)。



定时构建语法: `* * * * *`

- 第一个 \* : 表示分钟, 取值 0~59
- 第二个 \* : 表示小时, 取值 0~23
- 第三个 \* : 表示一个月的第几天, 取值 1~31
- 第四个 \* : 表示第几月, 取值 1~12
- 第五个 \* : 表示一周中的第几天, 取值 0~7, 其中 0 和 7 代表的都是周日

常用的定时构建比如:

- `H H/2 * * *`: 每两小时构建一次
- `0 18 * * *`: 每天 18:00 下班前定时构建一次
- `H H(9-18)/2 * * 1-5`: 在工作日的 9 AM ~ 6 PM 期间每两个小时构建一次 (或许在 10:38 AM, 12:38 PM, 2:38 PM, 4:38 PM)

## 可视化编辑流水线

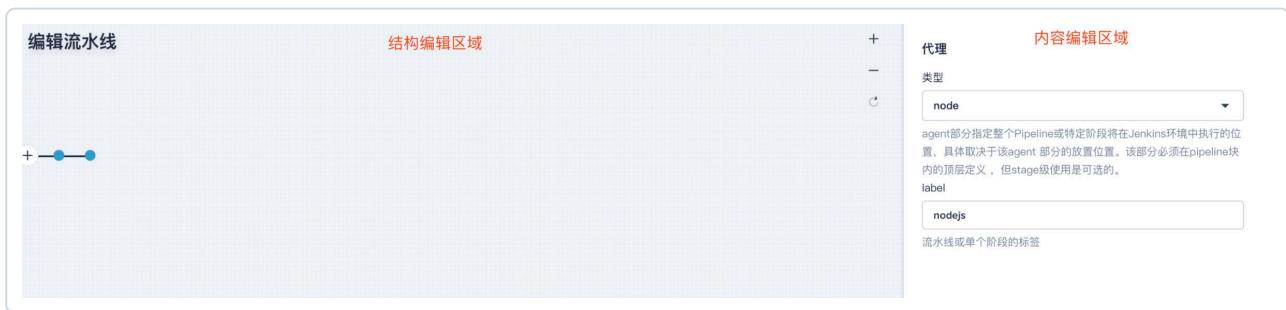
可视化流水线共包含 6 个阶段 (stage), 以下依次说明每个阶段中分别执行了哪些步骤和任务。

### 阶段一: Checkout SCM

1、可视化编辑页面, 分为结构编辑区域和内容编辑区域。通过构建流水线的每个阶段 (stage) 和步骤 (step) 即可自动生成 Jenkinsfile, 用户无需学习 Jenkinsfile 的语法, 非常方便。当然, 平台也支持手动编辑 Jenkinsfile 的方式, 流水线分为“声明式流水线”和“脚本化流水线”, 可视化编辑支持声明式流水线。Pipeline 语法参见 [Jenkins 官方文档](#)。

如下，此处代理的类型选择 `node`，label 填写 `nodejs`。

**说明：代理 (Agent) 部分指定整个 Pipeline 或特定阶段将在 Jenkins 环境中执行的位置，具体取决于该 agent 部分的放置位置，详见 [Jenkins Agent 说明](#)。**



2、点击左侧结构编辑区域的“+”号，增加一个阶段 (Stage)，点击界面中的 **No name**，在右侧将其命名为 **checkout SCM**。

3、然后在此阶段下点击 **添加步骤**。右侧选择 `git`，此阶段通过 Git 拉取仓库的代码，弹窗中填写的信息如下：

- Url: 填写 GitHub 示例仓库的 url <https://github.com/kubesphere/devops-docs-sample.git>
- 凭证 ID: 无需填写 (若是私有仓库，如 Gitlab 则需预先创建并填写其凭证 ID)
- 分支: 此处无需填写分支名，不填则默认为 master 分支

git

---

Url \*

凭证 ID

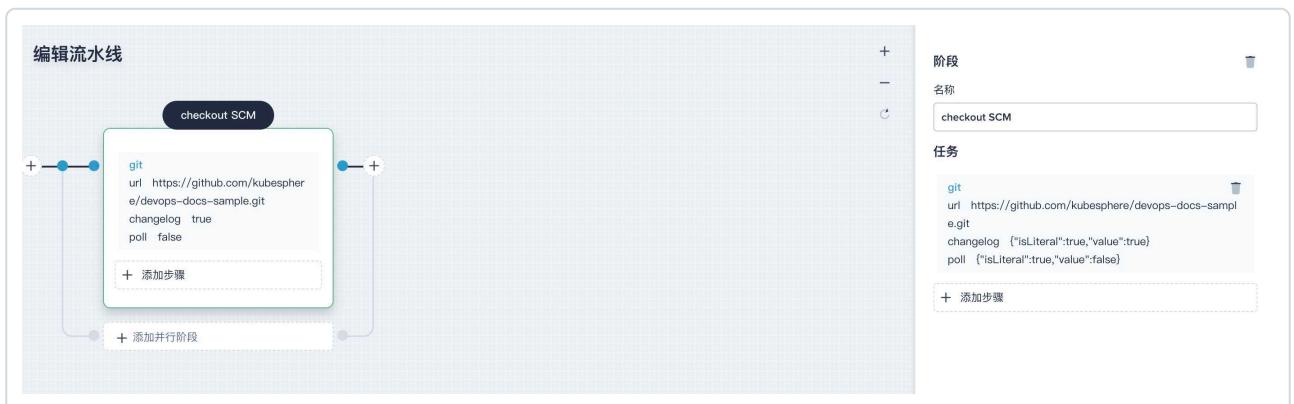
请选择

获取仓库代码可能需要凭证, 选择已有凭证或添加一个新的凭证 [新建凭证](#)

分支

取消
确定

填写后点击 **确定**, 可以看到构建的流水线的第一个阶段。



## 阶段二：Get dependencies

- 1、在第一个阶段 checkout SCM 右侧点击“+”继续增加一个阶段，此阶段用于获取依赖，命名为 **get dependencies**。
- 2、由于这个阶段需要在容器中执行 shell 脚本，因此点击 **添加步骤**，在右侧内容编辑区域首先选择 **container**，然后命名为 **nodejs**，完成后点击 **确定**。



3、然后在右侧内容编辑区域 nodejs 容器中点击 **添加嵌套步骤**，选择 **shell**，用于在创建的容器中执行 shell 命令，命令填写 **yarn**，完成后点击 **确定**。

**说明：**命令 **yarn** 等同于 **yarn install**。本文档网站使用包管理器 **yarn** 管理依赖，执行 **yarn** 命令可以安装项目需要的所有依赖。



### 阶段三：Unit test

1、同上，在 **get dependencies** 阶段右侧点击“+”继续增加一个阶段用于在容器中执行单元测试，名称为 **unit test**。

2、点击 **添加步骤** 选择 **container**，命名为 **nodejs**，完成后点击 **确定**；



3、然后在 nodejs 容器中点击 **添加嵌套步骤**，选择 **shell**，shell 命令填写 **yarn test**，用于单元测试，完成后点击 **确定**。

**说明：**如果单元测试通过了才允许继续下面的任务，由于本文档网站是静态网页，测试环节仅包含了单元测试。对于复杂的项目除了单元测试，通常还有端到端测试、集成测试、功能测试等步骤，这些都是项目持续部署的基础。



## 阶段四：Build

1、同上，在 **unit test** 阶段右侧点击“+”新添加一个阶段 **build** 用于在容器中执行生成静态网站的命令，名称为 **build**。

2、点击 **添加步骤** 选择 **container**，命名为 **nodejs**，完成后点击 **确定**；



3、然后点击 **添加嵌套步骤**，选择 **shell**，shell 命令填写 **yarn build**，完成后点击 **确定**（**yarn build** 用于执行项目中 **package.json** 的 scripts 里的 **build** 命令，如果 build 通过了流水线才会继续执行下面的任务）。



## 阶段五：Build and push snapshot image

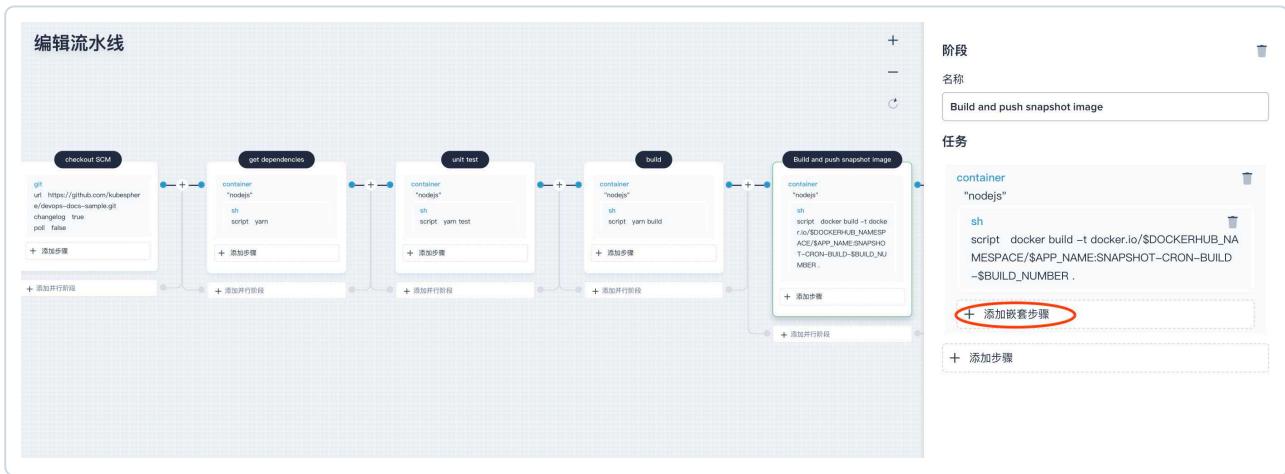
1、在 **build** 阶段右侧点击“+”新添加一个阶段，名称为 **Build and push snapshot image**。

**说明：**该阶段用于在容器中构建 snapshot 镜像，并将 tag 为 **SNAPSHOT-\$BRANCH\_NAME-\$BUILD\_NUMBER** 推送至 DockerHub，其中 **\$BRANCH\_NAME** 为分支名称，**\$BUILD\_NUMBER** 为 pipeline 活动列表的运行序号。

2、点击 **添加步骤** 选择 **container**，名称为 **nodejs**，完成后点击 **确定**；

3、然后点击 **添加嵌套步骤**，选择 **shell**，shell 命令填写如下一行 docker 命令，完成后点击 **确定**：

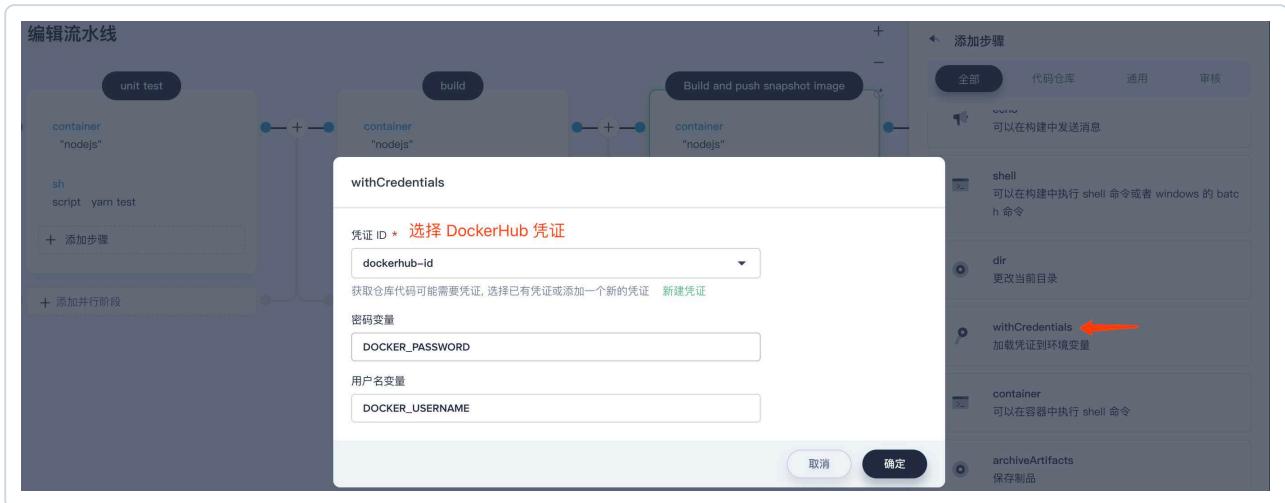
```
docker build -t docker.io/$DOCKERHUB_NAMESPACE/$APP_NAME:SNAPSHOT-CRON-BUILD-$BUILD_N
```



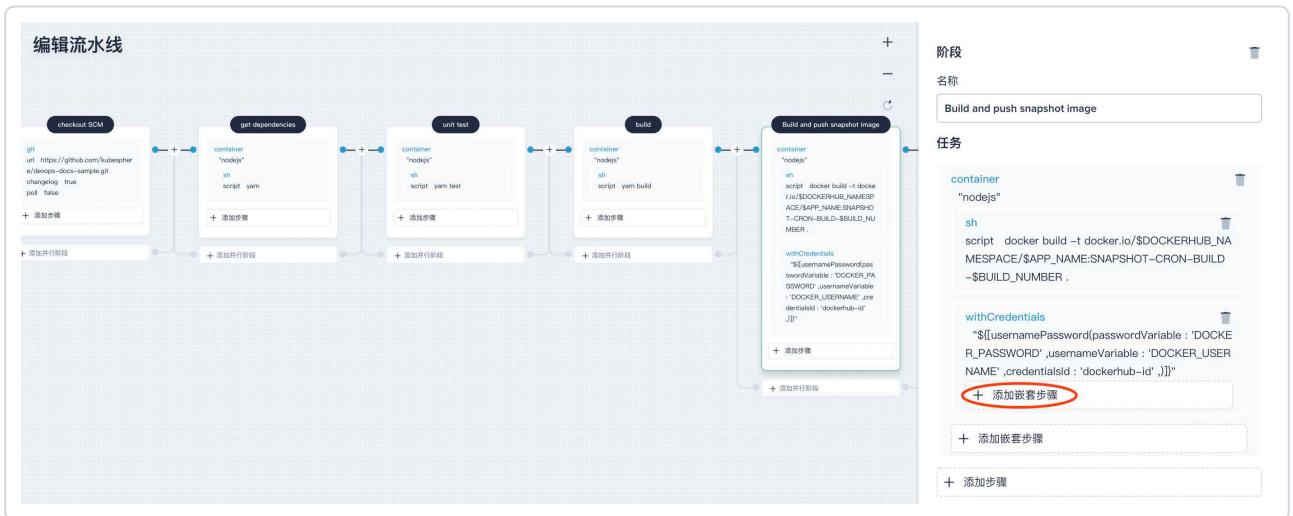
4、在当前阶段的 nodejs 容器中继续点击 **添加嵌套步骤** 在右侧选择 **withCredentials**，并填写如下信息用于登录 DockerHub，完成后点击 **确定**。

**说明：因为考虑到用户信息安全，账号类信息都不以明文出现在脚本中，而以变量的方式。**

- 凭证 ID：选择之前创建的 DockerHub 凭证，如 `dockerhub-id`
- 密码变量：`DOCKER_PASSWORD`
- 用户名变量：`DOCKER_USERNAME`

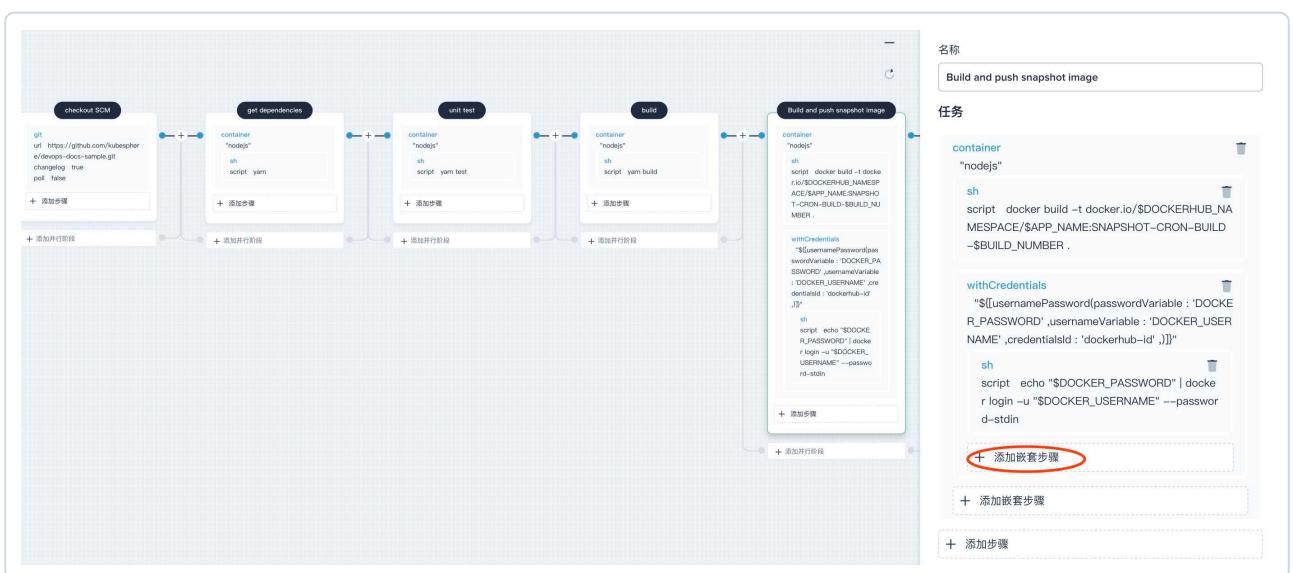


5、注意，下一步需要在 **withCredentials** 中依次添加两个 **嵌套步骤**，并且都选择 **shell**，用于登录 DockerHub 并推送镜像。在右侧点击 **添加嵌套步骤**，其中第一条 shell 命令填写如下，完成后点击 **确定**：



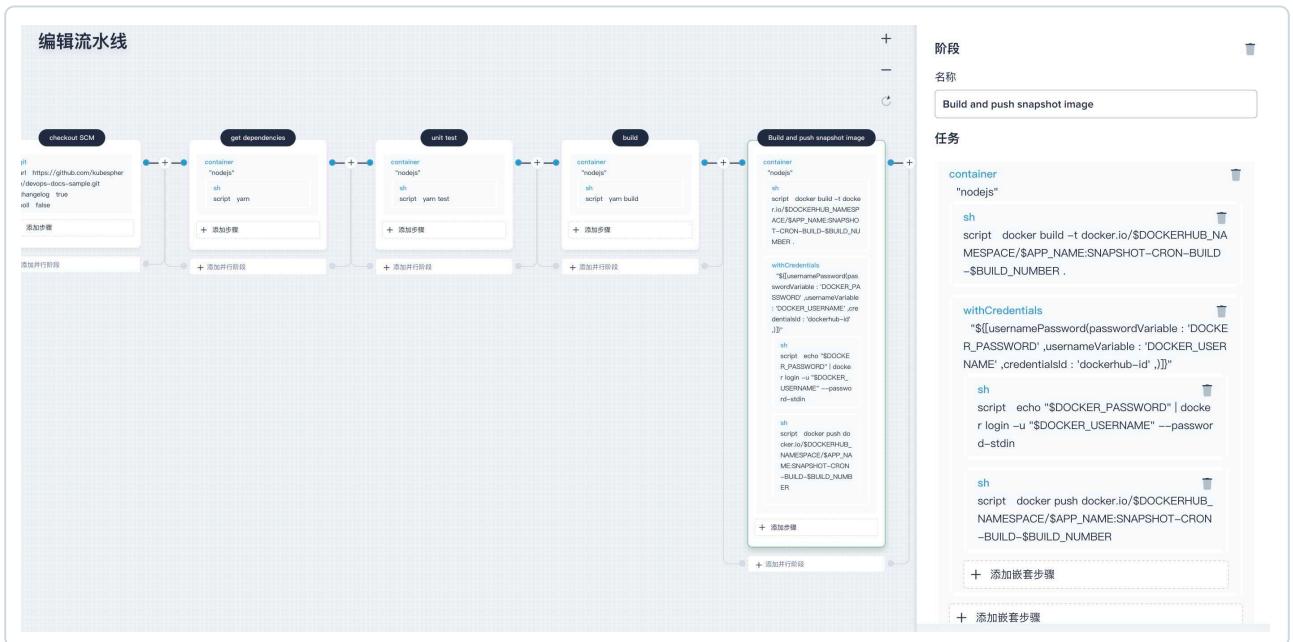
```
echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
```

6、同上，再次点击 **添加嵌套步骤**，第二条 shell 命令如下，完成后点击 **确定**：



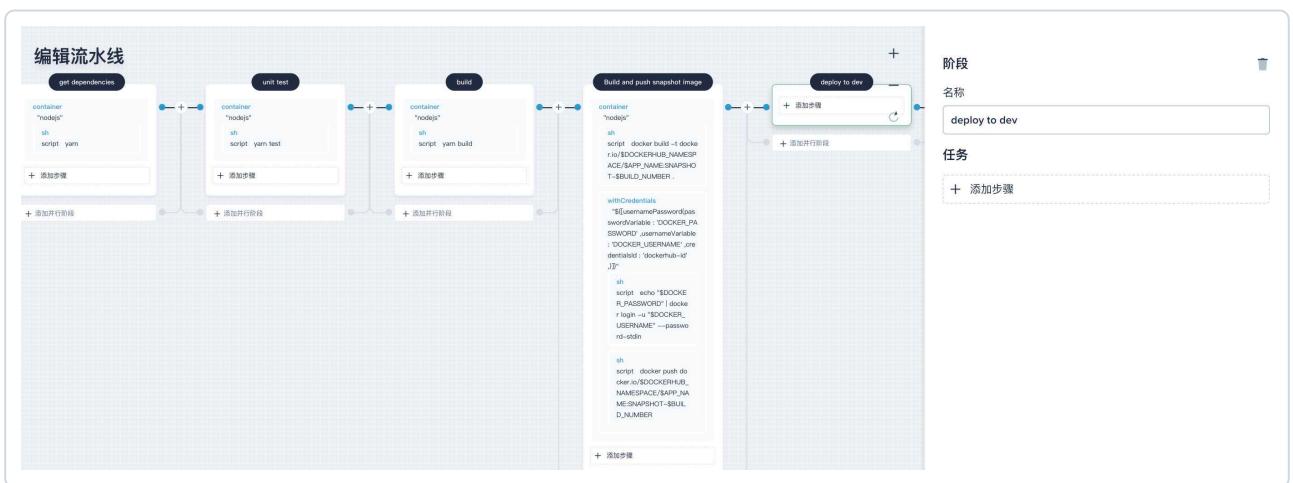
```
docker push docker.io/$DOCKERHUB_NAMESPACE/$APP_NAME:SNAPSHOT-CRON-BUILD-$BUILD_NUMBER
```

阶段五完成后的截图如下所示：



## 阶段六：Deploy to dev

1、最后一步，在阶段五右侧点击“+”增加最后一个阶段，此阶段用于人工审核，并部署到 Dev 环境，命名为 **deploy to dev**。

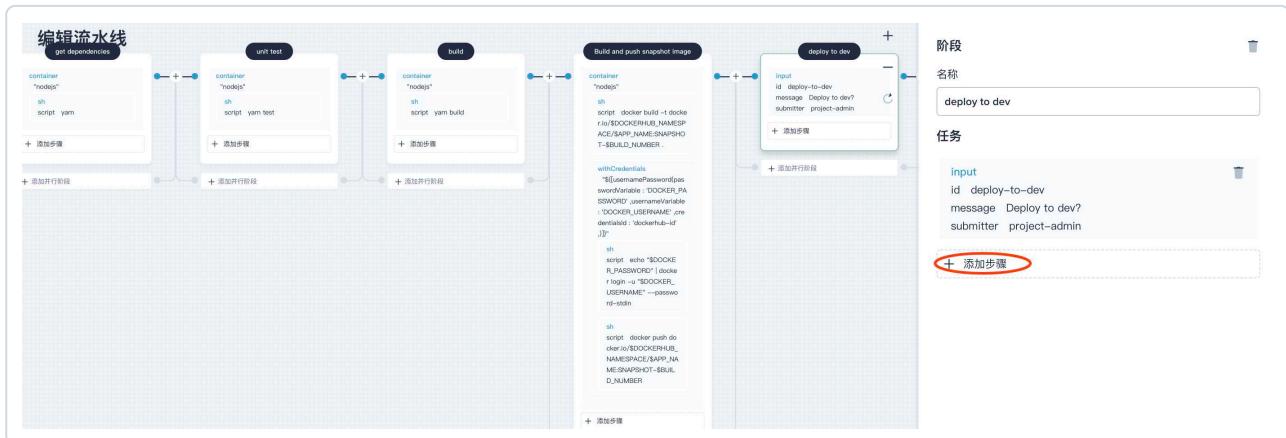


**说明：**因为在实际项目过程中，比如开发者提交了一次代码，测试也通过了，镜像也打包上传了，但是这个版本并不一定就是要立刻上线到生产环境的，可能需要将该版本先发布到开发环境、测试环境、或者预览环境之类，所以通常需要在 CD 的环节增加人工审核。

2、点击 **添加步骤**，右侧选择 **input**。该步骤用于人工审核，当流水线执行至审核步骤时将暂停，待审

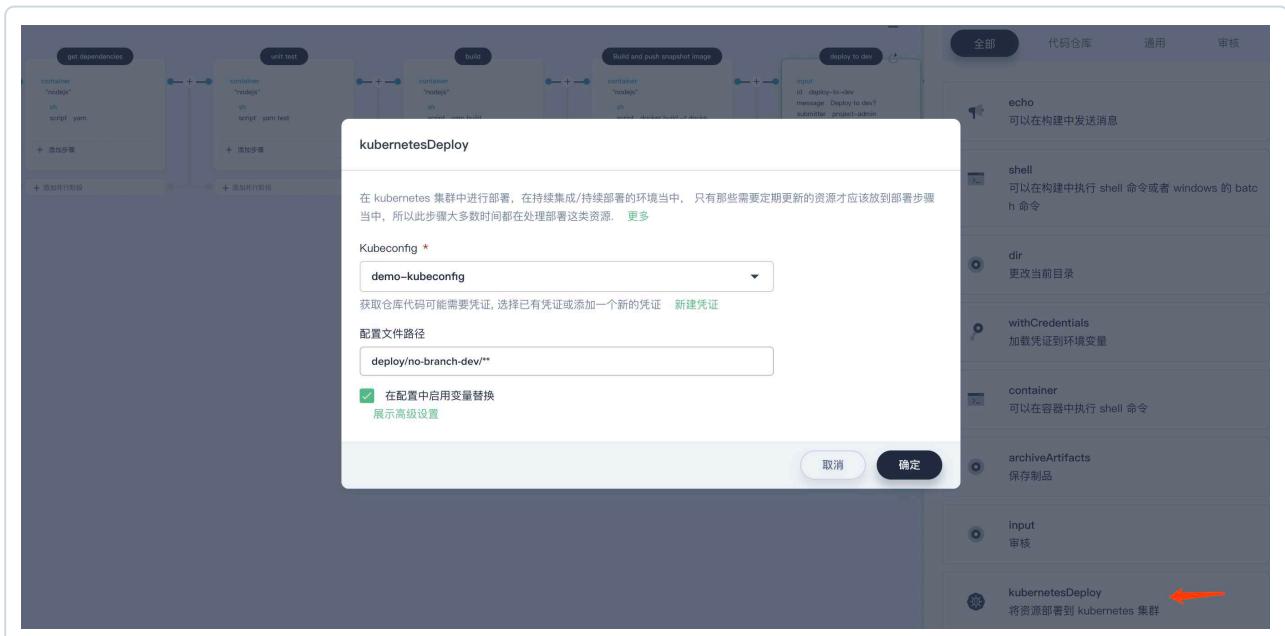
核通过后才可以继续进行。审核信息参考如下填写，完成后点击 确定：

- id：可选，填写步骤的 ID，如 `deploy-to-dev`
- 消息 (message)：必填，在用户人工审核时呈现给用户，此处可填 `Deploy to dev?`
- 审核者 (submitter)：本示例指定用户名为 `project-admin` 的用户，支持输入 **用户名** 指定用户人工审核。注意，此处审核者如果为空则默认此工程下包括当前用户在内的所有用户（不限角色）都具有审核权限。

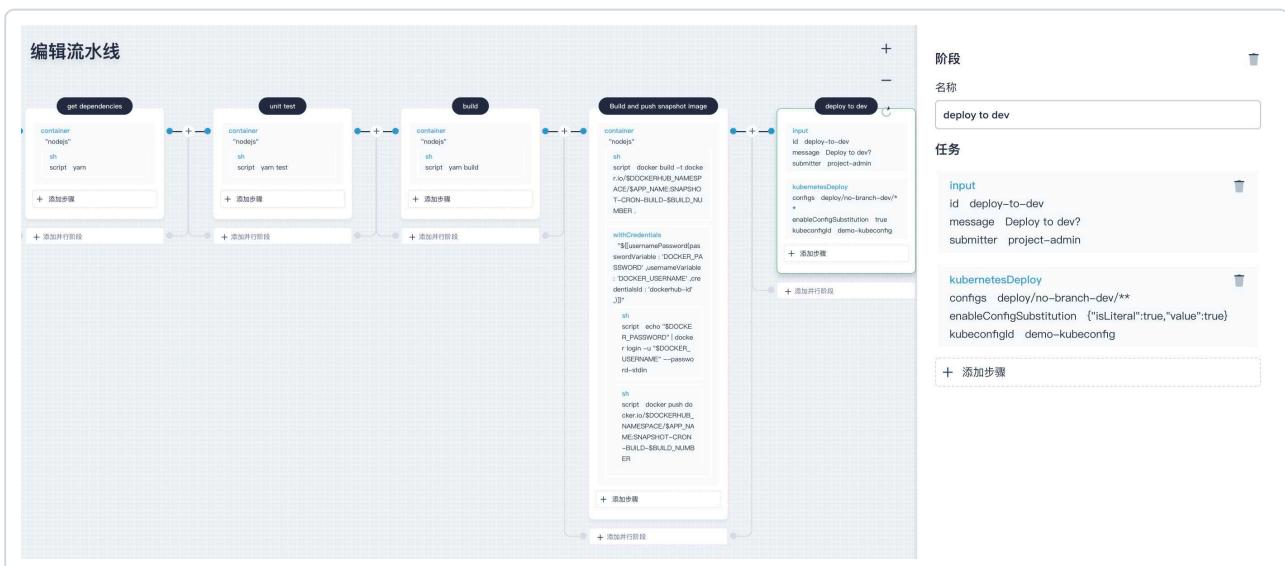


3、在当前阶段点击 **添加步骤**，右侧选择 `kubernetesDeploy`，这是 [Kubernetes Continuous Deploy Plugin](#) 中的函数，该步骤可将项目中指定路径的 yaml 模板部署到 Kubernetes 中，配置信息如下，填写后点击 **确定**：

- Kubeconfig: 选择之前创建的 kubeconfig 凭证
- 配置文件路径 (configs): yaml 模板在项目中的相对路径, 填写 `deploy/no-branch-dev/**`
- 在配置中启用变量替换 (enableConfigSubstitution), 保持默认, 允许在流水线中通过变量动态传值 (比如以上用到的 `$DOCKER_PASSWORD`、`$APP_NAME`)

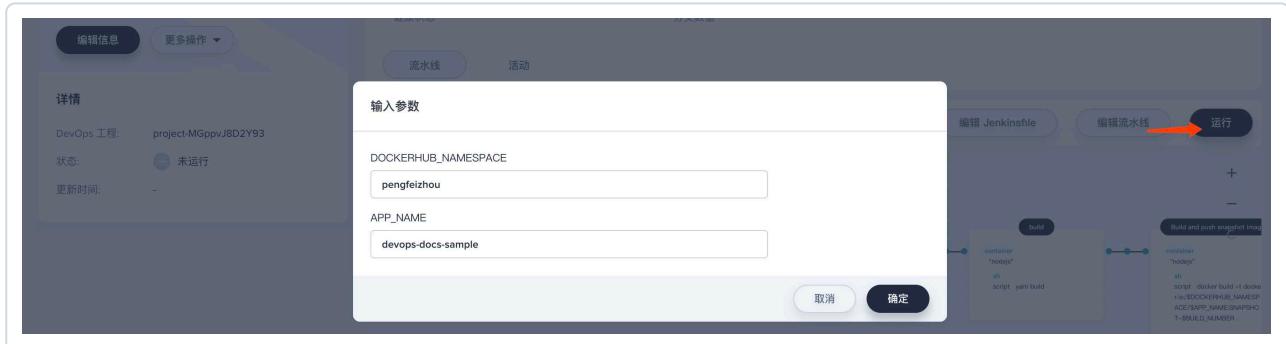


至此, 流水线的 6 个阶段都已构建完成, 点击 **确认 → 保存**, 可可视化创建的 pipeline 会默认生成 Jenkinsfile 文件。



## 运行流水线

1、手动构建的流水线在平台中需要手动运行，点击 **运行**，输入参数弹窗中可编辑之前定义的两个字符串参数，此处无需修改，点击 **确定**，流水线将开始运行。



2、在 **活动** 列表中可以看到流水线的运行状态，点击活动项可查看其运行活动的具体情况，例如以下查看 **运行序号** 为 1 和 2 的活动。

**说明：**流水线刚启动时可能仅显示其日志输出而无法看到其图形化运行的页面，这是因为它有个初始化的过程，流水线刚开始运行时，slave 启动并开始解析和执行流水线自动生成的 Jenkinsfile，待初始化完成即可看到图形化流水线运行的页面。

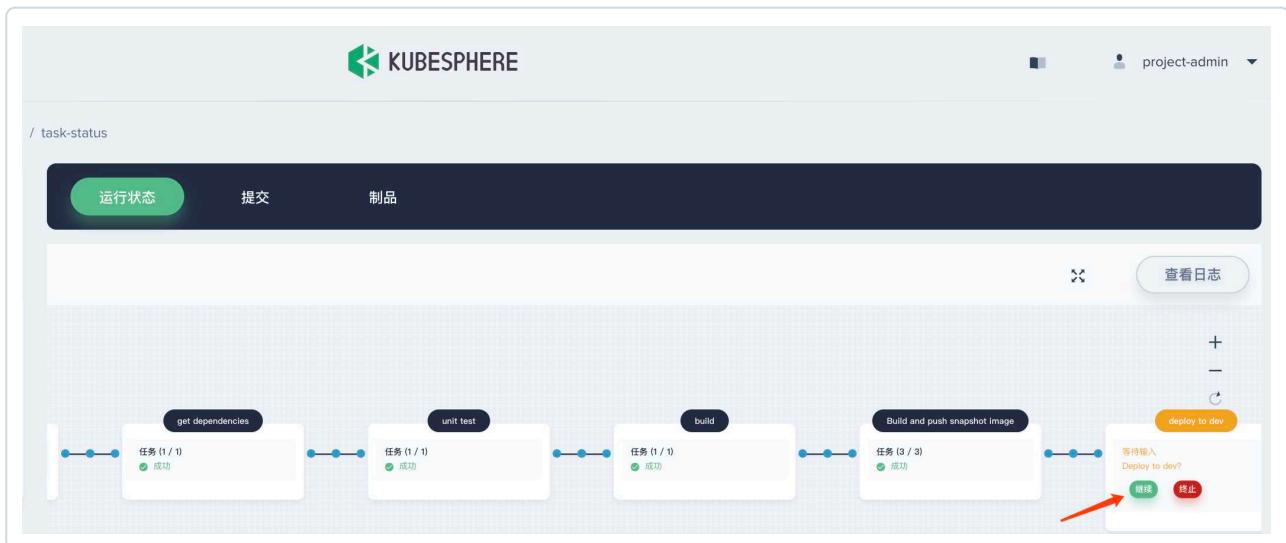


3、在活动列表点击运行序号 1，进入序号 1 的活动详情页查看流水线的具体运行情况。

4、登出平台并切换为 **project-admin** 登录后，进入示例 DevOps 工程下的流水线 jenkinsfile-out-of-scm。然后点击 **活动** 进入序号 1 的活动详情页，可以看到流水线已经运行至 **deploy to dev** 阶段，点击

继续，流水线的活动状态转变为 运行中。

说明：若前面的步骤配置无误则几分钟后可以看到流水线已经成功运行到了最后一个阶段。由于我们在最后一个阶段添加了 input 即人工审核步骤，并指定了用户名为 project-admin 的用户。因此流水线运行至此将暂停，等待审核者 project-admin 登录进入该流水线运行页面来手动触发。此时审核者可以测试构建的镜像并进一步审核整个流程，若审核通过则点击 继续，最终将部署到开发环境中。



## 查看流水线

1、几分钟后，流水线将运行成功。点击流水线中 活动 列表下查看当前正在运行的流水线序列号，页面展示了流水线中每一步骤的运行状态。黑色框标注了流水线的步骤名称，示例中流水线的 6 个 stage 就是以上创建的六个阶段。



2、当前页面中点击右上方的 查看日志 ，查看流水线运行日志。页面展示了每一步的具体日志、运行状

态及时间等信息，点击左侧某个具体的阶段可展开查看其具体日志，若出现错误可根据日志信息来分析定位问题，日志支持下载至本地查看。

流水线运行日志

阶段 用时 1.72 min

checkout scm

阶段

get dependencies

阶段

unit test

阶段

build & push snapshot i...

阶段

push latest image

阶段

deploy to dev

阶段

Shell Script 1.71 min 成功

```
[3_Jenkinsfile-in-SCM_master-GBUQRNAD5BRTQFBMAE3CEJBSWNGIFXUR2IMQPPT2VTPDTS4Q77QA] Running shell script
+ yarn
yarn install v1.3.2
[1/4] Resolving packages...
[2/4] Fetching packages...
info fsevents@1.2.4: The platform "linux" is incompatible with this module.
info "fsevents@1.2.4" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
warning "gatsby > friendly-errors-webpack-plugin@1.7.0" has incorrect peer dependency "webpack@^2.0.0 || ^3.0.0".
warning "gatsby > postcss-cssnext@2.11.0" has unmet peer dependency "caniuse-db@1.0.30000652".
warning " > gatsby-plugin-styled-components@2.0.11" has unmet peer dependency "babel-plugin-styled-components@>1.0.0".
warning "gatsby-transformer-remark > graphql-type-json@0.1.4" has unmet peer dependency "graphql@>0.4.0".
warning " > react-helmet@5.2.0" has unmet peer dependency "react@>=15.0.0".
warning "react-helmet > react-side-effect@1.1.5" has unmet peer dependency "react@^0.13.0 || ^0.14.0 || ^15.0.0".
warning " > react-modal@3.6.1" has unmet peer dependency "react@^0.14.0 || ^15.0.0 || ^16".
warning " > react-modal@3.6.1" has unmet peer dependency "react-dom@^0.14.0 || ^15.0.0 || ^16".
```

## 验证运行结果

若流水线的每一步都能执行成功，那么流水线最终 build 的 Docker 镜像也将被成功地 push 到 DockerHub 中，我们在 Jenkinsfile 中已经配置过 Docker 镜像仓库，登录 DockerHub 查看镜像的 push 结果，可以看到 tag 为 SNAPSHOT-CRON-BUILD-xxx 的镜像已经被 push 到 DockerHub。在 KubeSphere 中最终以 deployment 和 service 的形式部署到了开发环境中。

1、切换为 `project-regular` 登录 KubeSphere，进入 `kubesphere-docs-dev` 项目，在左侧的菜单栏点击 **工作负载 → 部署**，可以看到 `ks-docs-sample-dev` 已创建成功。

环境	访问地址	所在项目 (Namespace)	部署 (Deployment)	服务 (Service)
Dev	公网IP : 30860 ( <code> \${EIP}:\${NODEPORT} </code> )	kubesphere-docs-dev	ks-docs-sample-dev	ks-docs-sample-dev

## 查看部署

The screenshot shows the KubeSphere deployment interface. On the left, there's a sidebar with '项目 kubesphere-docs...' selected. The main area has a title '部署' (Deployment) with a brief description: 'KubeSphere 部署提供对常见用户应用程序的细粒度管理。部署配置，它将应用程序的特定组件所需状态描述为 Pod 模板。'. It displays deployment statistics: '1 已用配额' (1 used quota), '∞ 规划配额' (∞ planned quota), and '∞ 剩余 Pod 配额' (∞ remaining Pod quota). Below this is a search bar and a table with one row: '名称: ks-docs-sample-dev 状态: 运行中(1/1)' (Name: ks-docs-sample-dev Status: Running (1/1)).

2、在菜单栏中选择 **网络与服务 → 服务** 也可以查看对应创建的服务，可以看到该服务对外暴露的节点端口 (NodePort) 是 **30860**。

## 查看服务

The screenshot shows the KubeSphere service interface. On the left, there's a sidebar with '服务' selected. The main area has a title '服务' with a brief description: '一个 Kubernetes 的服务 (Service) 是一种抽象，它定义了一类 Pod 的逻辑集合和一个用于访问它们的策略。'. It displays service statistics: '1 已用配额' (1 used quota), '1 虚拟 IP' (1 virtual IP), and '0 Headless'. Below this is a search bar and a table with one row: '名称: ks-docs-sample-dev IP地址: Virtual IP: 10.233.33.78 端口: 80:80/TCP 节点端口: 80:30860/TCP' (Name: ks-docs-sample-dev IP Address: Virtual IP: 10.233.33.78 Port: 80:80/TCP Node Port: 80:30860/TCP). A red circle highlights the '30860' value.

3、查看推送到 DockerHub 的镜像，可以看到 **devops-docs-sample** 就是 **APP\_NAME** 的值，而 **TagName** 则是 **SNAPSHOT-CRON-BUILD-\$BUILD\_NUMBER** 的值 (**\$BUILD\_NUMBER** 对应活动的运行序号)，而 tag 为 **SNAPSHOT-CRON-BUILD-1** 的镜像，正是 **kubesphere-docs-dev** 这个部署所用到的镜像。

Showing 1-25 of 45 Tags

Tag	Size	Last updated	操作
v0.0.1	8 MB	2 hours ago	⋮
latest	8 MB	2 hours ago	⋮
SNAPSHOT-master-2	8 MB	2 hours ago	⋮

4、若需要在外网访问，可能需要进行端口转发并开放防火墙，才能访问到成功部署的文档网站示例。

例如，在 QingCloud 云平台上，如果使用了 VPC 网络，则需要将 KubeSphere 集群中的任意一台主机上暴露的节点端口 (NodePort) **30860** 在 VPC 网络中添加端口转发规则，然后在防火墙放行该端口。

### 添加端口转发规则

名称	协议	源端口	内网 IP	内网端口	创建于	操作
demo7-docs-nodeport	tcp	30860	192.168.0.18	30860	几秒前	禁用

\* 提示：可通过在各个资源上点击「右键」来进行常用操作，以及「双击」来修改基本属性。

### 防火墙添加下行规则

名称	优先级	协议	行为	起始端口 (?)	结束端口 (?)	源IP	操作
demo6-docs-nodeport	10	TCP	接受	30860			禁用

### 访问示例服务

如下在浏览器访问部署到 KubeSphere 的开发环境的服务：<http://127.0.0.1:30860/>。

### Dev 环境

The screenshot shows the KubeSphere Express v1.0.0 documentation website. The left sidebar has a dark background with a green header bar containing the text "简介". Below this are links for "安装指南", "快速入门", and two collapsed sections starting with "用户指南" and "附录". The main content area has a light gray background. At the top, it displays the KubeSphere logo and the text "KUBESPHERE | 文档中心". On the right, there is a search icon and the text "快速查找". The main title "KubeSphere 简介" is in bold black font. Below it, a section titled "产品介绍" is visible. A detailed description follows: "KubeSphere 是在目前主流容器调度平台 [Kubernetes](#) 之上构建的企业级分布式多租户容器管理平台，提供简单易用的操作界面以及向导式操作方式，在降低用户使用容器调度平台学习成本的同时，极大减轻开发、测试、运维的日常工作的复杂度。除此之外，平台已经整合并优化了多个适用于容器场景的功能模块，以帮助企业轻松应对多租户、工作负载和集群管理、服务与网络管理、应用管理、镜像仓库管理和存储管理等业务场景，下一个版本将支持服务治理、CI/CD、监控日志、大数据、人工智能以及 LDAP 集成等复杂业务场景。KubeSphere 提供了在生产环境集群部署的全栈化容器部署与管理平台。"

至此，创建一个 Jenkinsfile Out of SCM 类型的流水线已经完成了，若创建过程中遇到问题，可参考 [常见问题](#)。

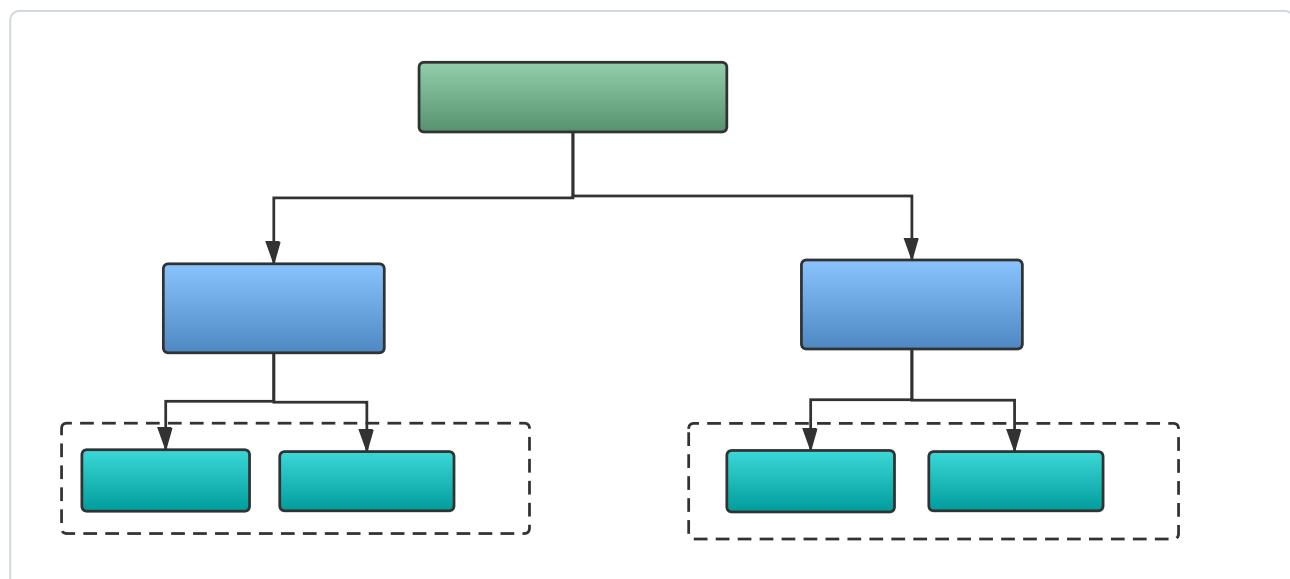
## 多租户管理概述

多租户管理的核心是分配好人员(组织)和资源之间的权限关系。对于容器管理平台来说，主要关注的资源有计算资源、存储资源和网络资源，这也是 KubeSphere 多租户管理的核心。

KubeSphere 多租户体系中，将资源划分为以下三个层级：

- 集群
- 企业空间
- 项目和 DevOps 工程

针对不同的层级的资源都可以灵活的定制角色用以划分用户的权限范围，实现不同用户之间的资源隔离。



## 权限管理模型

常见的权限管理模型有 ACL、DAC、MAC、RBAC、ABAC 这几类。在 KubeSphere 中我们借助 RBAC 权限管理模型来做用户的权限控制，用户并不直接和资源进行关联，而是经由角色定义来进行权限控制。

# 资源层级划分

## 集群

集群指的是当前的 Kubernetes 集群，为租户提供计算、存储和网络资源。集群下可以创建企业空间。

## 企业空间

在集群下可以创建企业空间，用以对不同的项目进行分组管理。企业空间下可以创建项目和 DevOps 工程。

## 项目和工程

项目、DevOps 工程是目前版本权限管理的最小层级，消耗集群的资源来部署应用、构建应用。

# 多层级权限控制

## 集群权限控制

通过集群角色来定义用户对集群资源的控制权限，例如：节点、监控、账户等。

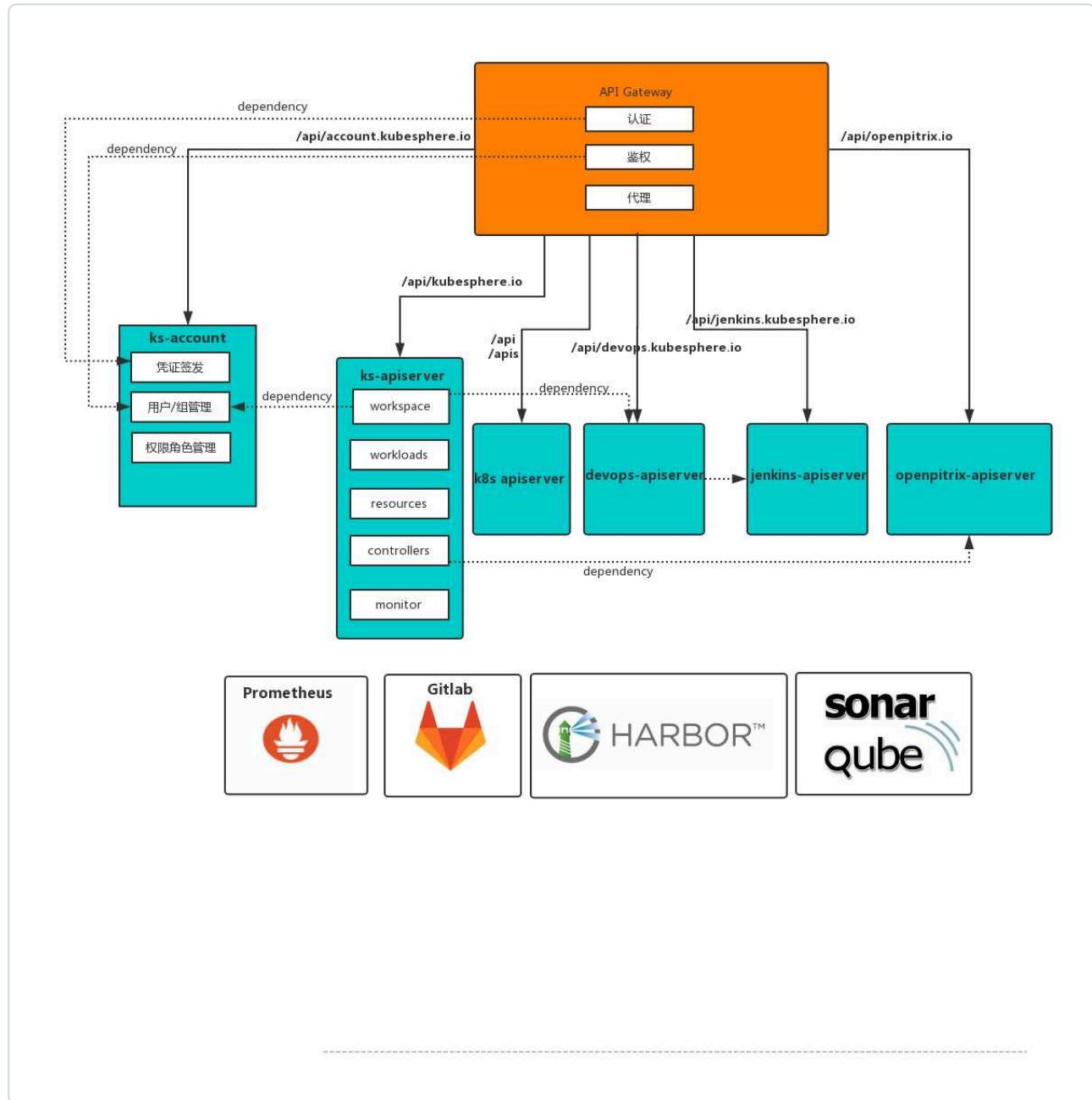
## 企业空间权限控制

企业空间角色则定义用户在企业空间下对项目和工程的控制权限以及企业空间成员的管理权限等。

## 项目和工程权限控制

项目和工程的创建者，可以通过邀请成员的方式将自己的项目共享给其他用户，赋予不同的成员不同的角色，在权限上加以区分。

# IAM架构



## 详细说明

在熟悉并了解了资源层级划分、权限管理方式之后，对于平台中各个层级的管理员和普通用户而言，理解每个层级的具体成员和角色的含义，如何更好地管理平台中成员和角色，就是实际使用中的关键环节。

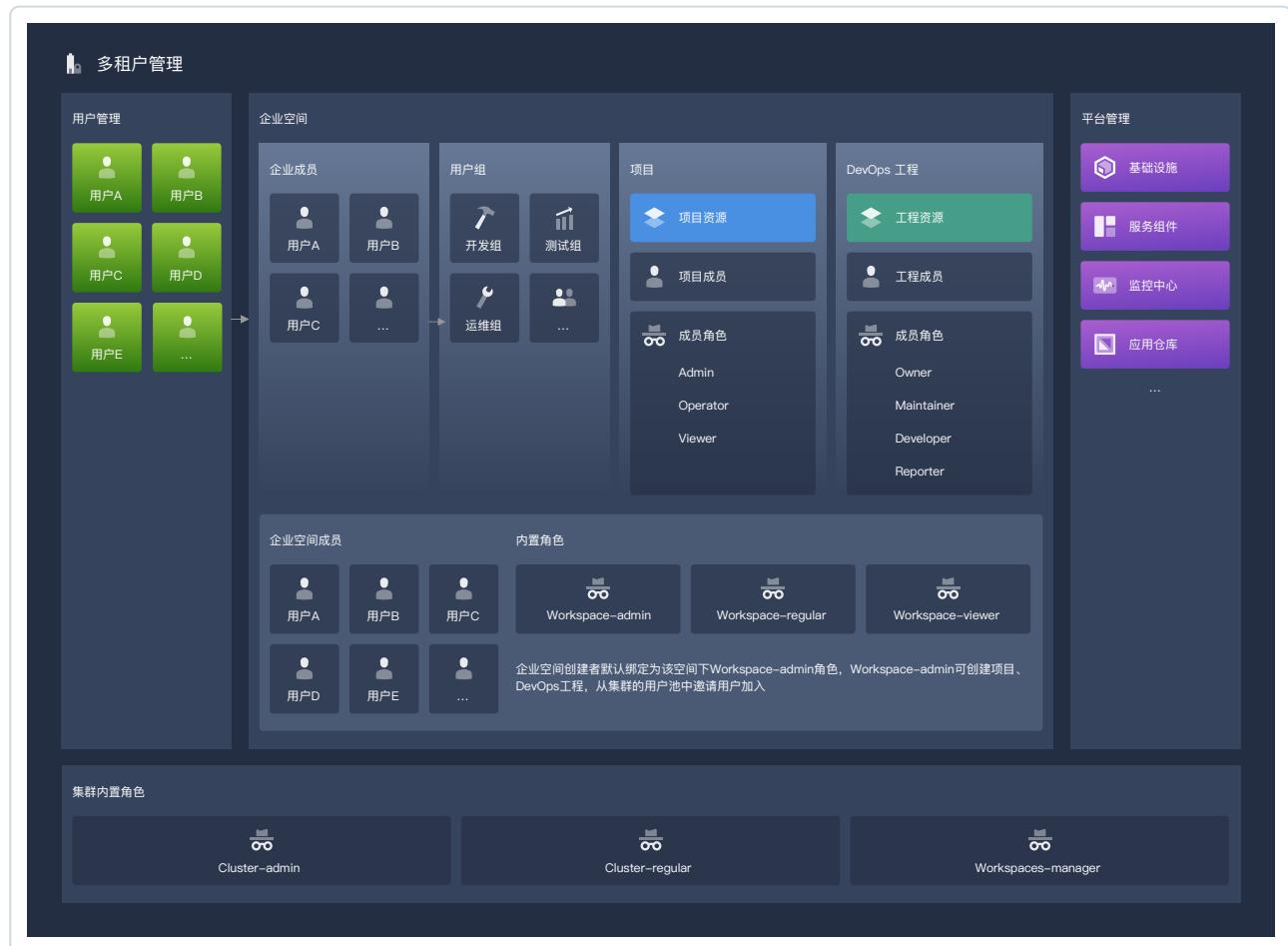
了，请继续参阅 [角色权限概览](#)。

# 角色权限概览

在实际的应用场景中，KubeSphere 的资源层级分为 **集群、企业空间、项目/ DevOps 工程** 等三个层级，多层级的划分也是实现多租户和资源隔离的基础。在 **账号管理** 中为用户创建了账号后，先由集群或企业空间的管理员通过 **邀请** 的方式邀请新成员加入该企业空间，然后才可以被同一企业空间下的项目或 DevOps 工程的管理员，邀请加入到项目和 DevOps 工程中，邀请时管理员可以对新成员赋予对应的角色，而不同的角色拥有不同的操作权限，平台内置了几个常用的角色供使用。同时，平台支持管理员自定义角色和权限列表，以下分别说明不同层级下的权限管理和预置角色。

## 角色权限管理

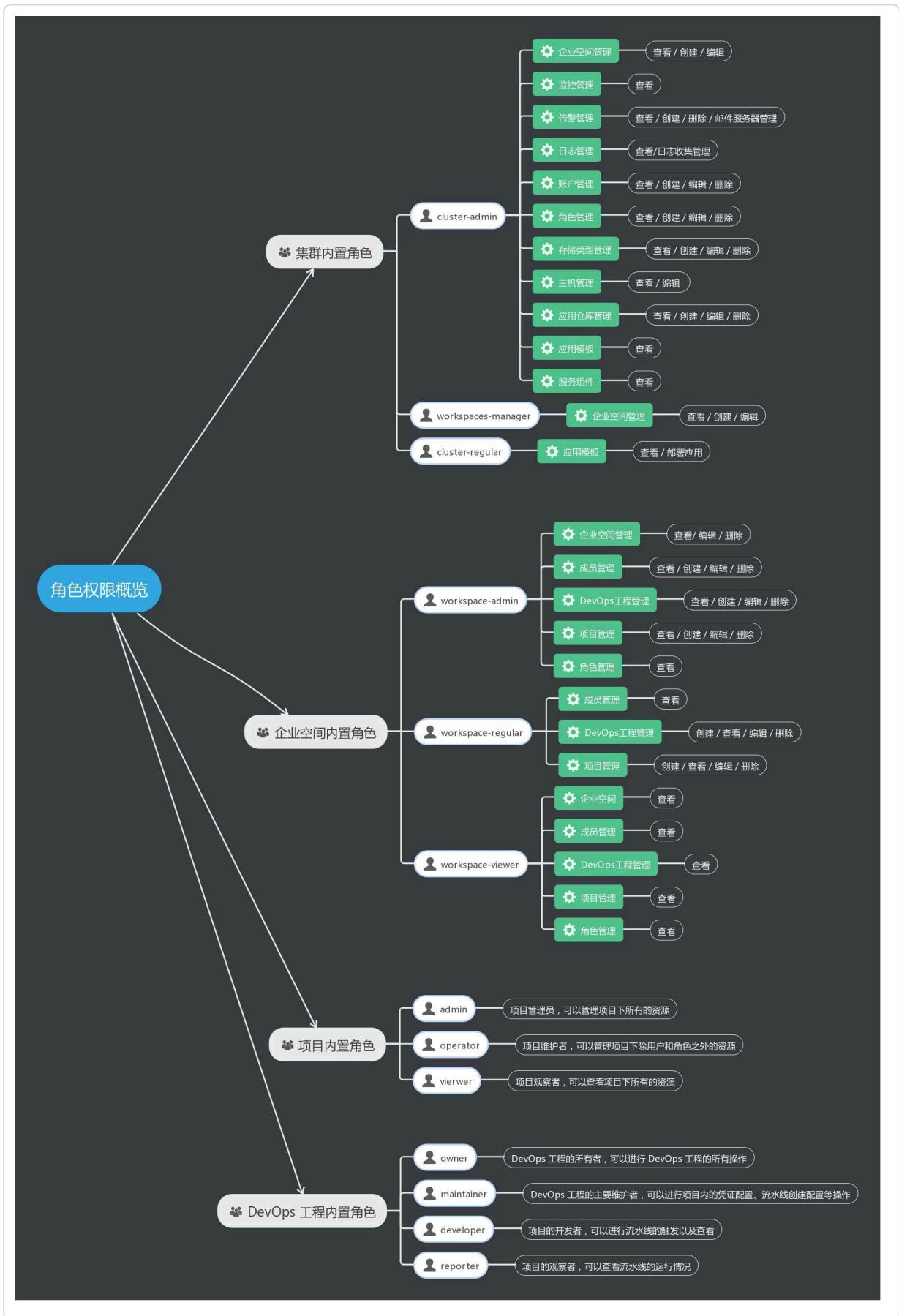
角色分为集群、企业空间、项目/ DevOps 工程共三个层级，将用户和不同层级的资源关联起来。



## 预置角色权限

在集群和集群空间下之所以如此划分多个内置角色，旨在针对不同角色的权限进行细粒度的管理，让拥有不同角色的成员更准确地查看和管理在集群或企业空间下的资源。只有对不同角色的权限进行细分后，多租户模式下对不同资源的管控和隔离才会更加安全。下图汇总了在集群和企业空间下不同角色分别具有哪些权限。

### 预置角色权限概览



## 集群角色

集群下包括用户管理、企业空间管理、各类集群层面的资源管理。集群层面的权限，主要针对节点、集群监控、存储类型、企业空间等集群层面的资源控制。

### 预置的集群角色：

预置角色	描述
cluster-admin	集群管理员，可以管理集群中所有的资源。
workspaces-manager	集群中企业空间管理员，可以管理集群中所有的企业空间及其下面的项目和工程资源。
cluster-regular	集群中的普通用户，在被邀请加入企业空间之前没有任何资源操作权限。

## 企业空间角色

独立的租户、项目和 Devops 工程都在企业空间下，企业空间下也有成员，可以从集群的用户池中添加。当企业空间被创建之后，创建者默认绑定为该空间下的 admin 角色。企业空间的管理员，可以在企业空间中创建项目、DevOps 工程资源，从集群的用户池中邀请用户加入到企业空间下。

### 预置的企业空间角色：

预置角色	描述
workspace-admin	企业空间管理员，可以管理企业空间下所有的资源。
workspace-regular	企业空间普通成员，可以在企业空间下创建工程和项目。
workspace-viewer	企业空间的观察者，可以查看企业空间下所有的资源信息。

## 项目角色

集群计算资源的虚拟划分，项目有资源配额限制。当项目、DevOps 工程被创建之后，创建者默认绑定至该项目或 DevOps 工程下的 admin 角色。由项目的管理员，向项目内邀请成员，可以从企业空间下的用户池中搜索并添加用户。

## 预置的项目角色：

预置角色	描述
admin	项目管理员，可以管理项目下所有的资源。
operator	项目维护者，可以管理项目下除用户和角色之外的资源。
viewer	项目观察者，可以查看项目下所有的资源。

## 预置角色权限概览

资源/权限/ 角色	admin	operator	viewer
项目管理	查看 / 编辑 / 删除	查看	查看
成员管理	查看 / 创建 / 编辑 / 删除	/	/
角色管理	查看 / 创建 / 编辑 / 删除	/	/
部署	查看 / 创建 / 编辑 / 删除 / 横向伸缩	查看 / 创建 / 编辑 / 删除 / 横向伸缩	查看
有状态副本集	查看 / 创建 / 编辑 / 删除 / 横向伸缩	查看 / 创建 / 编辑 / 删除 / 横向伸缩	查看
守护进程集	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
容器组管理	远程连接	远程连接	远程连接
服务管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
外网访问管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
应用路由管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
存储卷	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
应用管理	查看 / 部署 / 删除	查看 / 部署 / 删除	查看
任务管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
定时任务管	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看

资源/权限/ 角色	admin	operator	viewer
理			
密钥管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
配置管理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看

## DevOps 工程角色

方便细粒度的对 DevOps 工程进行管控。由 DevOps 工程的管理员从企业空间下的用户池中搜索并添加用户。

预置的 DevOps 工程角色：

预置角色	描述
owner	项目的所有者，可以进行项目的所有操作。
maintainer	项目的主要维护者，可以进行项目内的凭证配置、pipeline 配置等操作。
developer	项目的开发者，可以进行 pipeline 的触发以及查看。
reporter	项目的观察者，可以查看 pipeline 的运行情况。

# 企业空间管理

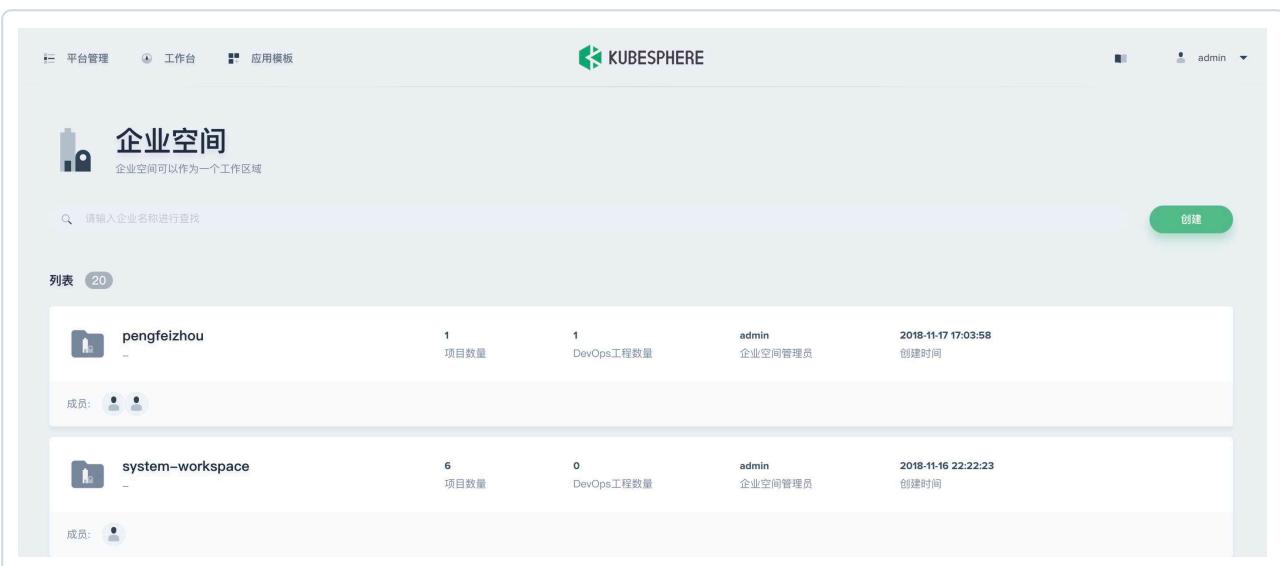
企业空间 (workspace) 是 KubeSphere 实现多租户模式的基础，是您管理项目、DevOps 工程和企业成员的基本单位。当用户创建企业空间后，该用户即是此企业空间的管理员 (admin)，有权查看和管理这个企业空间内的所有对象 (项目、DevOps 工程)，同时管理员将继承该企业空间内项目和工程的管理权限。集群中的 cluster-admin 和 workspaces-manager 角色拥有企业空间管理的权限，支持查看、创建、编辑或删除企业空间。本文档说明管理企业空间的常用功能。

## 前提条件

已有 cluster-admin 或 workspaces-manager 角色的账号并已登录了控制台，

## 创建企业空间

1、点击控制台左上角 平台管理 → 企业空间，可以看到当前集群中所有企业空间的列表，点击 创建 按钮。



The screenshot shows the KubeSphere Platform Management interface with the 'Workspace' tab selected. At the top, there are navigation links: 平台管理, 工作台, 应用模板, and the KubeSphere logo. On the right, there is a user dropdown for 'admin'. Below the header, the title '企业空间' is displayed with a sub-instruction: '企业空间可以作为一个工作区域'. A search bar with placeholder text '请输入企业名称进行查找' is followed by a green '创建' button. The main area is titled '列表 (20)' and contains two workspace entries:

名称	项目数量	DevOps工程数量	管理员	创建时间
pengfeizhou	1	1	admin 企业空间管理员	2018-11-17 17:03:58
system-workspace	6	0	admin 企业空间管理员	2018-11-16 22:22:23

2、参考如下提示填写基本信息。

- 企业空间名称：请尽量保持企业名称简短，便于用户浏览和搜索。
- 企业空间管理员：可从当前的集群成员中指定。

- **描述信息：**详细地介绍企业空间，当用户想进一步了解该企业空间时，描述内容将变得尤为重要。

### 创建企业空间

企业空间是一个组织您的项目和 DevOps 工程、管理资源访问权限以及在您团队内部共享资源等的逻辑单元，可以作为您团队工作的独立工作空间。

**基本信息**

企业空间名称 \*

请尽量保持名称简短，比如用企业名称的缩写或者大家经常的称呼，无需使用企业的完整名称或者营业执照上的注册名称。

企业空间管理员

描述信息

KubeSphere的企业空间

描述信息不超过 1000 个字符

**取消** **确定**

## 管理企业空间

当企业空间被创建之后，创建者默认绑定为该空间下的 admin 角色。企业空间的管理员，可以在企业空间中创建项目、DevOps 工程资源，从集群的用户池中邀请用户加入到企业空间下。

## 资源概览

进入所属的企业空间后，管理员可以在 **概览** 页查看当前集群的资源用量如物理资源和应用资源的监控详情，支持对监控情况的自定义时间范围查询。管理员还可以看到所有项目的用量排行，方便管理员对企业空间资源管理有更准确的把控。

The screenshot shows the KubeSphere platform management interface. At the top, there are navigation tabs: 平台管理 (Platform Management), 工作台 (Workstation), 应用模板 (Application Template), and a user account admin. The main header features the KubeSphere logo and the text 'KUBESPHERE'. On the left sidebar, there are links for 企业空间 (Enterprise Space) 'system-workspace', 概览 (Overview), 项目管理 (Project Management), DevOps工程 (DevOps Project), and 企业空间管理 (Enterprise Space Management). The central area displays resource usage statistics:

- 物理资源用量 (Physical Resource Usage): 1.84 核 CPU
- 应用资源用量 (Application Resource Usage): 74 部署 (Deployments), 13 任务 (Tasks), 52 服务 (Services), 119 运行中的容器组 (Running Container Groups)
- DevOps工程 (DevOps Project): 0
- 企业角色 (Enterprise Role): 3
- 企业成员 (Enterprise Member): 1

Time range selection tools are available, including a dropdown for '最近 1 天' (Last 1 Day) and a '自定义时间范围' (Custom Time Range) section with start and end times set to 2018-11-19 08:13:14.

点击概览页的监控折线图，可以查看该资源更细粒度的监控详情，例如查看 CPU 使用量。

The screenshot shows a detailed monitoring view for the 'CPU 使用量 (核)' (CPU Usage (Core)) metric. The top part is a line chart from 09:00 to 08:20 on November 19, 2018. A vertical line marks the time 2018-11-18 21:50. A tooltip at this point shows a usage of 1.87 核 (1.87 Cores). Below the chart is a table of specific usage data points:

时间 (Time)	使用量 (Usage)
2018年 11月 19日 08:20	1.92 核
2018年 11月 19日 08:06	1.73 核
2018年 11月 19日 07:52	1.75 核

# 项目管理

项目的创建者即该项目的 admin，可以邀请同一企业空间下的成员加入项目并授权。进入所属的企业空间后，企业空间管理员可以查看、创建、编辑或删除当前企业空间下的所有项目。

## 创建项目

在指定的企业空间下，点击 **项目管理 → 创建**，填写项目的基本信息和高级设置。

基本信息

项目基础信息设置

名称 \*

demo-namespace

最长 63 个字符，只能包含小写字母、数字及分隔符(" - ")，且必须以小写字母或数字开头及结尾

别名

示例项目

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

This is a demo

### 基本信息

- 名称：为项目起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：详细地介绍该项目，当用户想进一步了解该企业空间时，描述内容将变得尤为重要。

### 高级设置

此处是在当前项目中配置容器默认的 CPU 和内存的请求与限额，相当于是给项目创建了一个 Kubernetes 的 LimitRange 对象，建议在此设置默认的请求与限额值。在项目中创建工作负载后填写容器组模板时，若不填写容器 CPU 和内存的请求与限额，则容器会被分配在这里配置的默认的 CPU 和内存请求与限额值。



创建项目后，关于项目内的各类资源和成员角色的管理，参见用户指南下各部分的文档。

## DevOps 工程

企业空间下的 admin 和赋予适当权限的用户可以创建和管理 DevOps 工程，详见 [管理 DevOps 工程](#)。

## 企业空间管理

企业空间的 admin 有权限管理基本信息、企业角色和邀请用户加入。

### 基本信息

基本信息显示了当前企业空间的企业成员和角色数量，点击右侧“...”按钮可以编辑或删除企业空间。

The screenshot shows the KubeSphere platform management interface. At the top, there are navigation links: 平台管理 (Platform Management), 工作台 (Workstation), 应用模板 (Application Template), and a user account icon for admin. The main header is KUBESPHERE. On the left, a sidebar for the 'sample' enterprise space lists: 概览 (Overview), 项目管理 (Project Management), DevOps工程 (DevOps Project), 企业空间管理 (Enterprise Space Management) (which is expanded to show 基本信息 (Basic Information), 企业角色 (Enterprise Roles), and 成员管理 (Member Management)), and a collapsed section for 云原生应用 (Cloud Native Application). The central content area is titled '基本信息' (Basic Information) for the 'sample' enterprise space. It displays the space's logo, name, and member count: 7 企业成员 (7 Enterprise Members) and 3 企业角色 (3 Enterprise Roles). There are buttons for 编辑信息 (Edit Information) and 删除 (Delete). A three-dot menu icon is also present.

## 企业角色

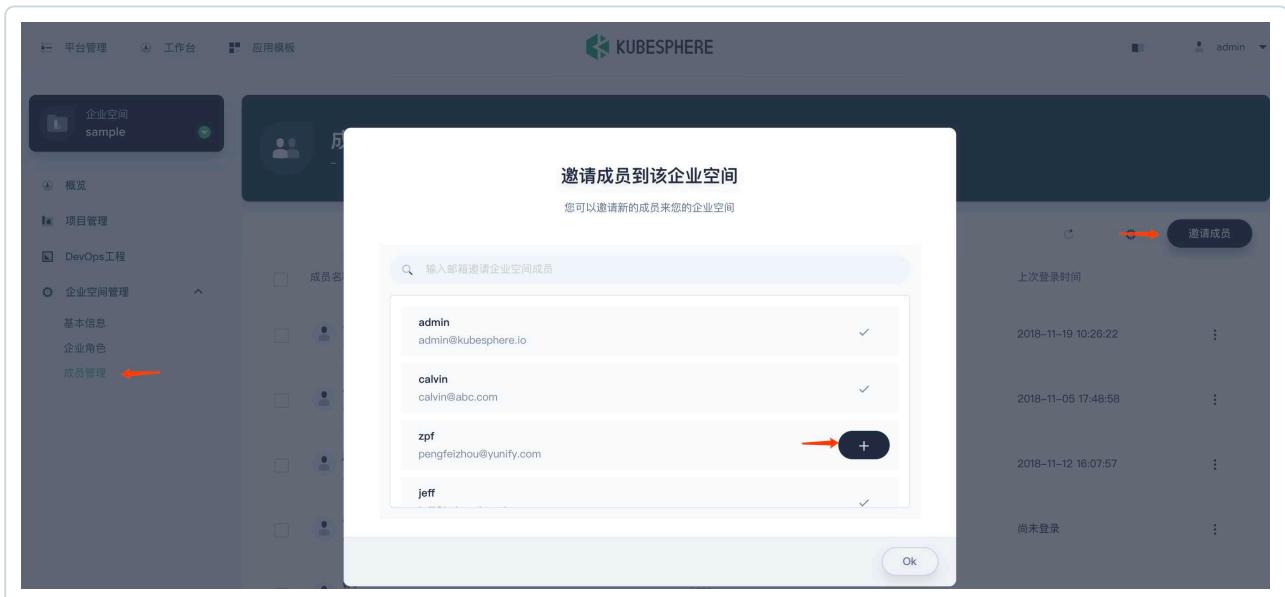
平台预置了三个常用的企业角色 workspace-admin、workspace-regular 和 workspace-viewer，预置角色拥有的权限见下表。

### 预置的企业空间角色：

预置角色	描述
workspace-admin	企业空间管理员，可以管理企业空间下所有的资源。
workspace-regular	企业空间普通成员，可以在企业空间下创建工程和项目。
workspace-viewer	企业空间的观察者，可以查看企业空间下所有的资源信息。

## 成员管理

admin 可以邀请集群中的成员加入到企业空间并分配角色，注意，集群中的 cluster-regular 只有被邀请到企业空间内，才能够被邀请加入同一企业空间下的项目和 DevOps 工程。



# 账号管理

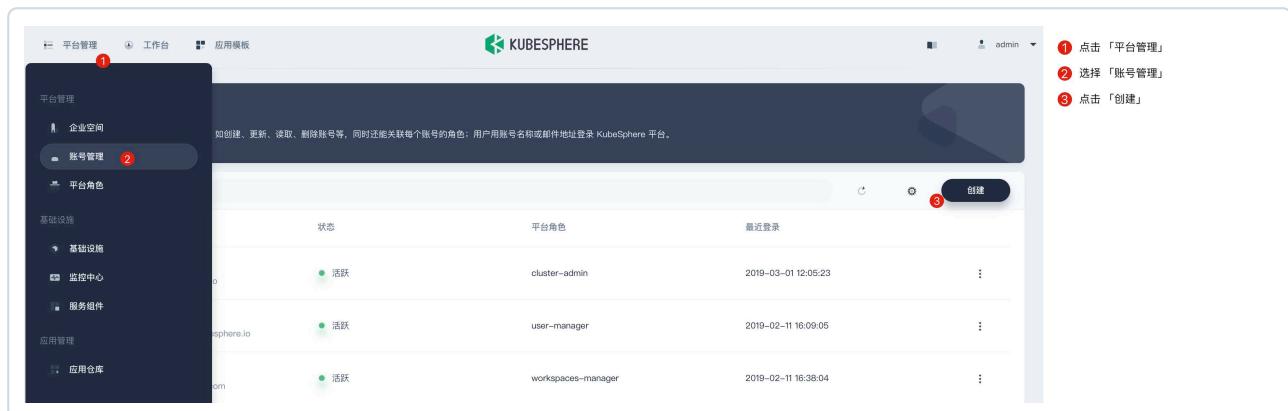
用户管理通常包含账号、角色、权限等三大部分，是任何一款产品必备的模块，而账号管理是管理员最常用到的功能。平台中的 cluster-admin 角色可以为其他用户创建账号和分配平台角色，平台为管理员内置了三个常用的角色。本文档说明如何为新用户创建账号。

## 前提条件

已有 cluster-admin 角色的账号且已登录到控制台。

## 创建账号

1、点击控制台左上角 平台管理 → 账号管理，可以看到当前集群中所有用户的列表，支持通过用户名进行搜索，点击 创建 按钮。



2、填写新用户的基本信息，其中内置角色的权限见下表。

预置的集群角色：

预置角色	描述
cluster-admin	集群管理员，可以管理集群中所有的资源。
cluster-regular	集群中的普通用户，在被邀请加入企业空间之前没有任何资源操作权限。
workspaces-manager	集群中企业空间管理员，可以管理集群中所有的企业空间及其下面的项目和工程资源。

 **添加用户** X

---

用户名 \*

用户名只能包含小写字母及数字

邮箱 \*

邮箱地址可以方便项目管理员及时的添加您为项目成员

角色

请选择

**cluster-admin**

**cluster-regular**

**workspaces-manager**

**user-manager**  
manage all accounts and roles

取消确定

## 编辑或删除账号

在列表中点击右侧“...”可编辑或删除该用户账号。

## 修改密码

若需修改账户密码，点击已创建的账号进入账号的详情页面，点击 **更多操作 → 修改密码**。详情页还支持查看账号的登录历史。

# 平台角色

正如 [账号管理](#) 所述，角色管理也是用户管理非常重要的一部分。角色管理是用来管理平台用户的角色信息。角色是对具有共同特征的某一类人群的身份归纳，在角色管理模块，我们需要描述角色信息并设置权限规则，让管理员能够轻松识别角色的特质，为不同的用户赋予对应的角色身份来更细粒度地管理资源。平台预置了 cluster-admin、wordspace-manager 和 cluster-regular 三类常用的角色，同时支持管理员自定义角色。本文档介绍内置角色的权限和如何自定义新的角色。

## 前提条件

已有 cluster-admin 角色的账号且已登录到控制台。

## 创建角色

在创建新的角色之前，先了解预置的集群角色有哪些权限。

**预置的集群角色：**

预置角色	描述
cluster-admin	集群管理员，可以管理集群中所有的资源。
cluster-regular	集群中的普通用户，在被邀请加入企业空间之前没有任何资源操作权限。
workspaces–manager	集群中企业空间管理员，可以管理集群中所有的企业空间及其下面的项目和工程资源。

管理员可以查看平台中所有的角色，若上述角色不满足实际需求，管理员可自定义平台角色和权限规则。在平台管理下选择 [平台角色](#)，点击创建。



## 权限设置

1、填写基本信息。

- 名称：为角色起一个简洁明了的名称，便于用户浏览和搜索。
- 描述信息：详细地介绍该角色，当用户想进一步了解该角色时，描述内容将变得尤为重要。

2、权限设置，支持对平台中的所有资源（如企业空间、监控、账户、角色、存储类型等）操作的细粒度控制，若勾选某项操作则表示用户将拥有相应的权限，一般来说，创建和编辑操作是相关联的，用户若需要则应同时勾选。对平台中的资源而言，删除操作是不可逆的，请谨慎选择。



## 编辑或删除角色

若需修改角色的基本信息或权限规则，点击角色列表右侧的“...”按钮，即可编辑或删除该角色。

# 服务组件

服务组件提供 KubeSphere、Kubernetes 和 OpenPitrix 集群内各项服务组件的健康状态监控，可以查看当前集群的健康状态和运行时间，能够帮助用户监测集群的状况和及时定位问题。

## 查看服务组件

登录 KubeSphere 管理控制台，访问 [平台管理 → 服务组件](#) 进入列表页。作为集群管理员，可以查看当前集群下所有的服务组件：

服务组件	健康状态	副本数量	运行时间
ks-account	健康	1: TODO	3个月
ks-apiserver	健康	1: TODO	3个月
ks-apiserver-whim	健康	1: TODO	1个月
ks-apiserver-xz	异常	1: TODO	3个月

## 服务组件的作用

服务组件可以查看当前集群健康状态，当集群出现异常时，管理员可以查看是否存在某个服务组件出现异常。比如使用应用模板部署应用时，应用没有部署成功，那么就可以查看是不是 Openpitrix 的组件出现异常，管理员就可以快速定位问题，然后根据出现异常的组件进行修复。当某个服务组件出现异常时，KubeSphere、OpenPitrix 和 Kubernetes 的 Tab 显示为异常组件的数目。

# 主机管理

Kubernetes 集群中的计算能力由主机 (Node) 提供, Kubernetes 集群中的 Node 是所有 Pod 运行所在的工作主机, 可以是物理机也可以是虚拟机。而 KubeSphere 主机管理的功能完全满足企业对集群运维监控的需求, 支持实时查看资源状态和节点状态, 查看任意主机上容器组的运行状态以及 CPU 和 内存的消耗, 并且主机的监控页面还能够实时地提供全方位细粒度的资源监控如 IOPS、磁盘吞吐、网卡流量, 让用户一目了然所有主机资源状态。

节点 (Node) 中有一个重要的功能即污点 (Taints) 管理。我们知道节点亲和性 (NodeAffinity) 是 Pod 上定义的一种属性, 使 Pod 能够按我们的要求调度到某个节点上, 而污点则恰恰相反, 它可以让节点拒绝运行 Pod, 甚至驱逐 Pod。污点是节点的一个属性, 如果主机设置了污点后, 底层的 Kubernetes 是不会将 Pod 调度到这个节点上的。

## 主机管理列表

首先登录 KubeSphere 管理控制台, 访问左侧菜单栏, 在 **资源** 菜单下, 点击 **主机管理** 按钮进入列表页。作为集群管理员, 可以查看当前集群下所有主机信息。列表即可一目了然每个节点的状态、污点情况和最常用的 CPU、内存、容器组数量等监控指标。

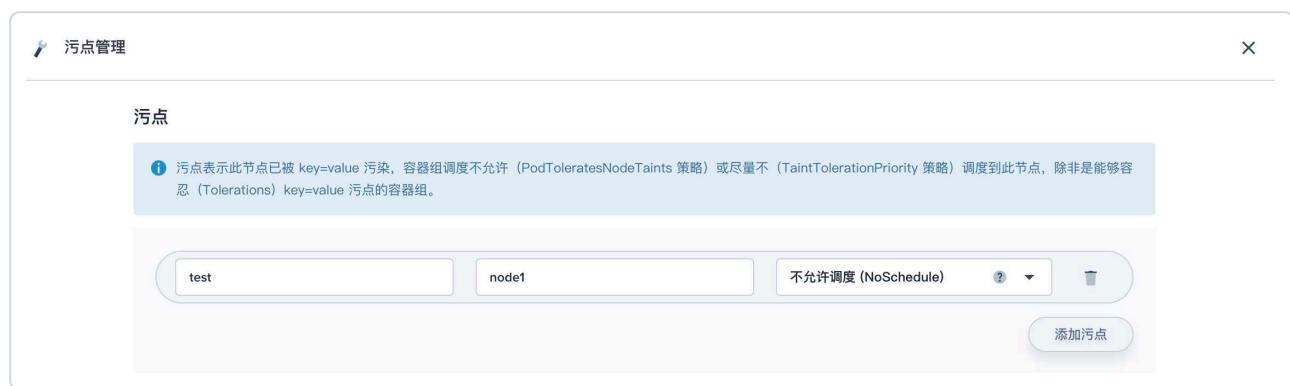
The screenshot shows the KubeSphere Management Console interface. On the left, there's a sidebar with '基础设施' (Infrastructure) selected, showing '主机' (Hosts) and '存储类型' (Storage Types). The main area has a header '主机' (Hosts) with stats: 10 节点数量 (Nodes), 1 主节点 (Master Node), and 9 计算节点 (Compute Nodes). Below is a search bar and a table with columns: 名称 (Name), 状态 (Status), 角色 (Role), 污点 (Taints), CPU(Core), 内存(GiB), and 容器组 (Container Groups). Four hosts are listed:

名称	状态	角色	污点	CPU(Core)	内存(GiB)	容器组
i-1abu8ksg 192.168.0.67	运行中	-	-	1.23/4	6.36/7.8	40/60
i-76v18j2o 192.168.0.53	运行中	worker	-	1.85/4	7.04/7.8	52/60
i-gddrhbxs 192.168.0.14	运行中	worker	-	1.4/4	4.73/7.8	43/60
i-w083nj02 192.168.0.52	运行中	worker	-	0.49/4	7.07/7.8	33/60

# 污点管理

点击列表中的某台主机，进入详情页。例如以下节点，我们发现它的内存使用情况已经高达 91%，因此不建议再继续往这台节点上调度新的 Pod，可以为其设置一个污点。设置了污点后，KubeSphere 是不会将 Pod 调度到这个 Node 上的。

1、点击左上角 **污点管理** 按钮，进入主机污点 (Taints) 管理页面。



2、污点的属性一般是 `key=value [effect]`，其中 `key=value` 一般用来匹配 Pod 的容忍 (Toleration)。而 effect 有 3 个选项，详见参数解释。

3、比如此处设置为 `test=node1`，effect 设置为 NoSchedule，那么只有设置了 Toleration 与该 Node 污点属性一致的 Pod，才允许被调度到该节点。

## 参数解释:

Node 的 effect 存在以下 3 个值，对于还未被调度的 Pod 来说：

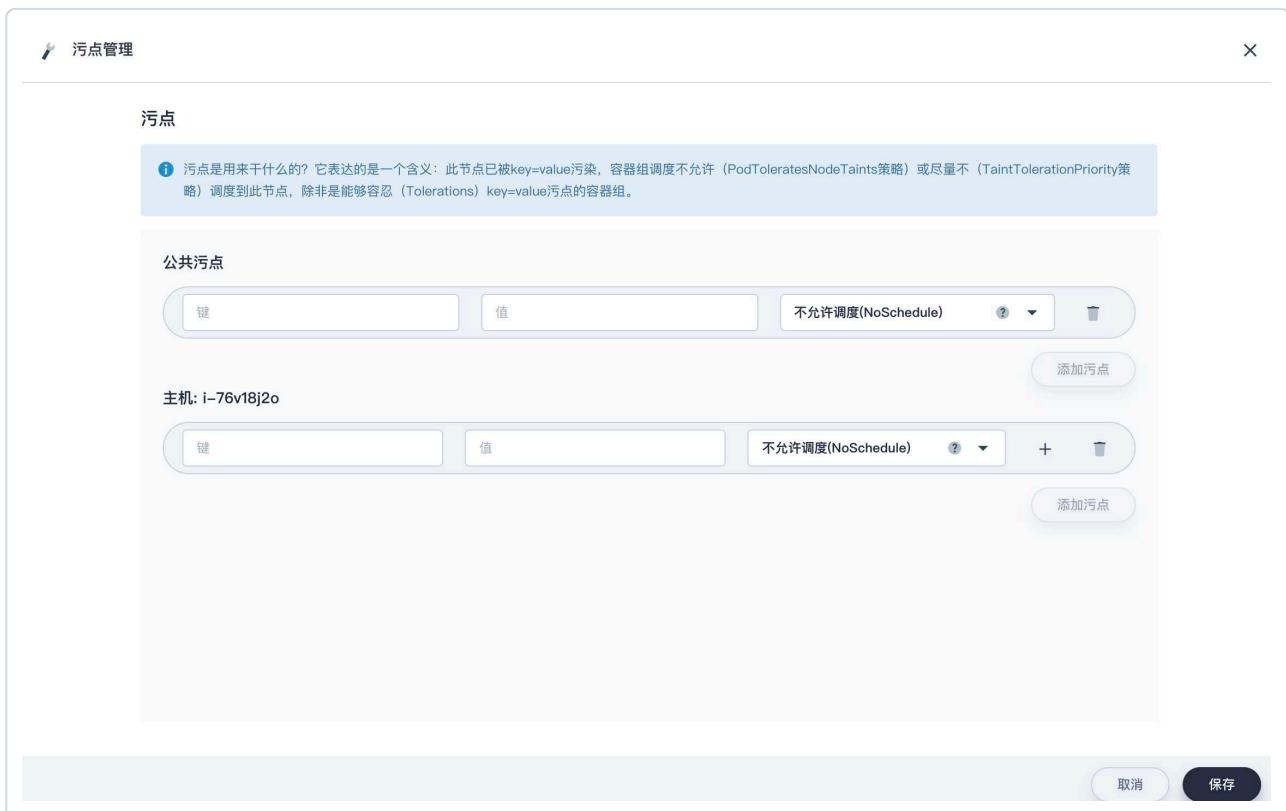
- NoSchedule: 表示不允许调度，已调度的资源不受影响。
- PreferNoSchedule: 表示尽量不调度。
- NoExecute: 表示不允许调度

如果在设置 node 的 Taints (污点) 之前，就已经运行了一些 Pod，则分为以下几种情况：

- 若 effect 的值是 NoSchedule 或 PreferNoSchedule，对于已运行的 Pod 仍然可以运行，只是新 Pod (如果没有设置容忍) 不会再往上调度。
- 若 effect 的值是 NoExecute，那么此 Node 上正在运行的 Pod，只要没有容忍的，立刻被驱

逐。effect 为 NoExecute 的污点，在容忍 (Toleration) 属性中有一个可选配置：tolerationSeconds 字段，用来设置这些 Pod 还可以在这个 Node 之上运行多久，给它们一点宽限的时间，到时间才驱逐。

- 如果是部署 (Deployment)，那么被该 Node 驱逐的 Pod 会漂移其它节点运行。
- 如果是守护进程集 (DaemonSet) 被驱逐后也不会再被运行到其它 Node，直到 Node 上的 NoExecute 污点被删除或者为该 Pod 设置了容忍。



## 查看主机详情

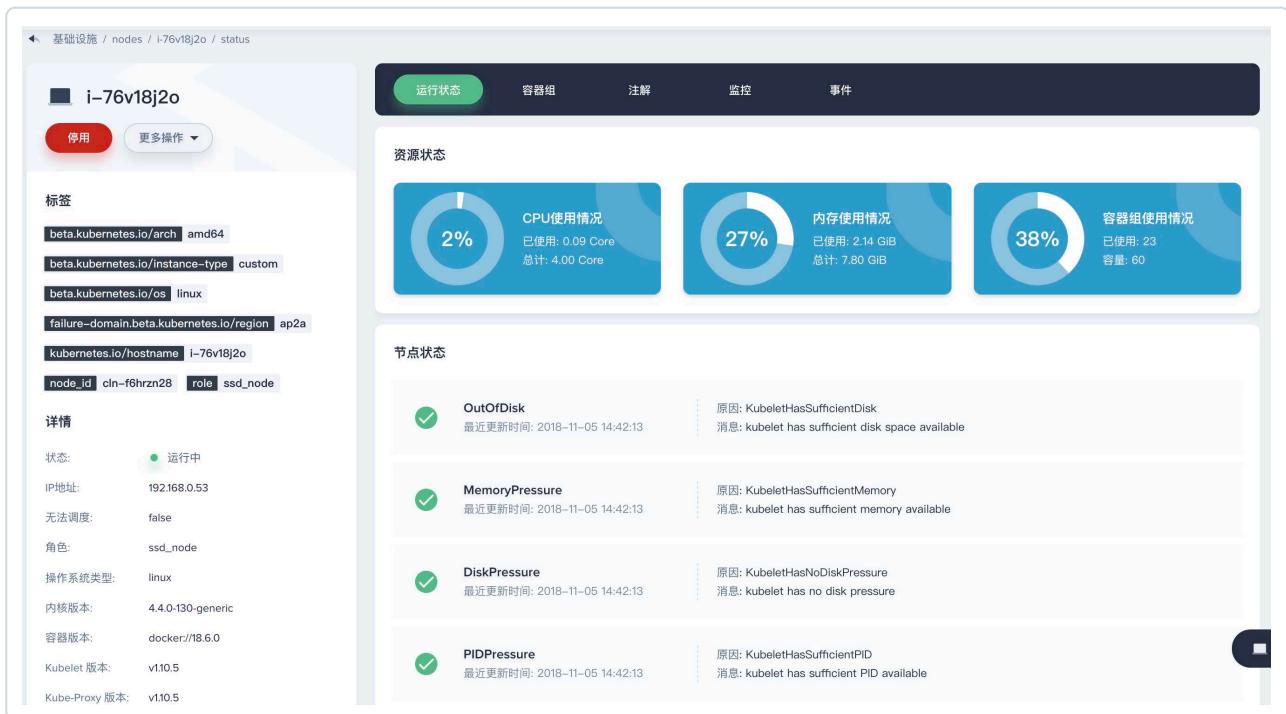
在主机列表页，点击某个主机节点打开其详情页，可以看到当前主机下所有资源的概况，包括主机的 CPU、内存和容器组资源运行和使用状态，并且支持查看主机上所有容器组的资源使用情况和数量变化，以及注解 (Annotation) 和事件 (Events) 信息。

## 节点状态

节点状态 (Conditions) 描述了所有运行中节点的状态，KubeSphere 对主机的管理有以下五种状态，集

群管理员通过以下五种状态可以判断当前节点的负载和能力，更合理地管理主机资源。

- OutOfDisk：表示当前节点是否有足够的空间添加和运行新的 Pods
- MemoryPressure：表示当前节点的内存压力的高低
- DiskPressure：表示当前节点的磁盘压力高低
- PIDPressure：表示当前节点的进程是否存在压力
- Ready：表示当前节点的状态是否健康和能够准备接收新的 Pods 运行，Node Controller 如果 40 秒 内没有收到节点的状态报告则为 Unknown



## 查看监控

值得一提的是，主机管理支持细粒度的资源监控，可筛选指定时间范围内的监控数据以查看变化情况。



## 停用或启用主机

在主机详情页面，点击左侧 **停用** (cordon) 按钮，主机状态变为 **无法调度**，当前按钮变为 **启用** (uncordon)，当有新的工作负载被创建时将不会被调度到此节点，如想回复为可调度状态，点击 **启用** 按钮。

## 更新主机标签

如果需要限制 Pod 到指定的 Node 上运行，则可以给 Node 打标签 (Label) 并给 Pod 配置节点选择器 (NodeSelector)。

例如，给其中一个 Node 打上标签 `role:ssd_node` 后，如果给 Pod 也设置了 NodeSelector 为 `role: ssd_node`，那么该 Pod 将只会在一个节点上运行。

如果需要更新更新主机标签，可在项目详情页面，点击左侧项目操作菜单，点击 **编辑标签** 按钮编辑当前主机上的标签 (Labels)，最后点击 **确认** 按钮完成修改。

/ 编辑标签 X

标签

beta.kubernetes.io/arch	amd64	<span style="font-size: 2em;">删除</span>
beta.kubernetes.io/instance-type	custom	<span style="font-size: 2em;">删除</span>
beta.kubernetes.io/os	linux	<span style="font-size: 2em;">删除</span>
failure-domain.beta.kubernetes.io/region	ap2a	<span style="font-size: 2em;">删除</span>
kubernetes.io/hostname	i-76v18j2o	<span style="font-size: 2em;">删除</span>
node_id	cIn-f6hrzn28	<span style="font-size: 2em;">删除</span>
role	ssd_node	<span style="font-size: 2em;">删除</span>

添加标签 取消 保存

# 存储类型

存储类型 (StorageClass) 是由 [集群管理员](#) 配置存储服务端的参数，并按类型提供存储给集群用户使用。通常情况下创建存储卷之前需要先创建存储类型，目前支持的存储类型如 [QingCloud 云平台块存储](#)、[QingStor NeonSAN](#)、[GlusterFS](#)、[Ceph RBD](#)、[NFS](#)、[Local Volume \(仅 all-in-one 支持\)](#) 等。需要注意的是，当系统中存在多种存储类型时，只能设定一种为默认的存储类型。

## 创建存储类型

首先登录 KubeSphere 管理控制台，选择 **平台管理 → 基础设施 → 存储类型**，进入存储类型列表页面。作为集群管理员，可以查看当前集群下所有的存储类型和详细信息。

### 第一步：填写基本信息

在存储类型列表页，点击 **创建** 按钮，填写基本信息：

- 名称：为存储类型起一个简洁明了的名称，便于用户浏览和搜索。
- 描述信息：详细介绍存储类型的特性，当用户想进一步了解该存储类型时，描述内容的完整将变得尤为重要。
- 允许存储卷扩容：存储卷允许扩容与否开关。
- 回收机制：默认 Delete，相关的存储资产比如 QingCloud 块存储也会一并删除。

创建存储类型

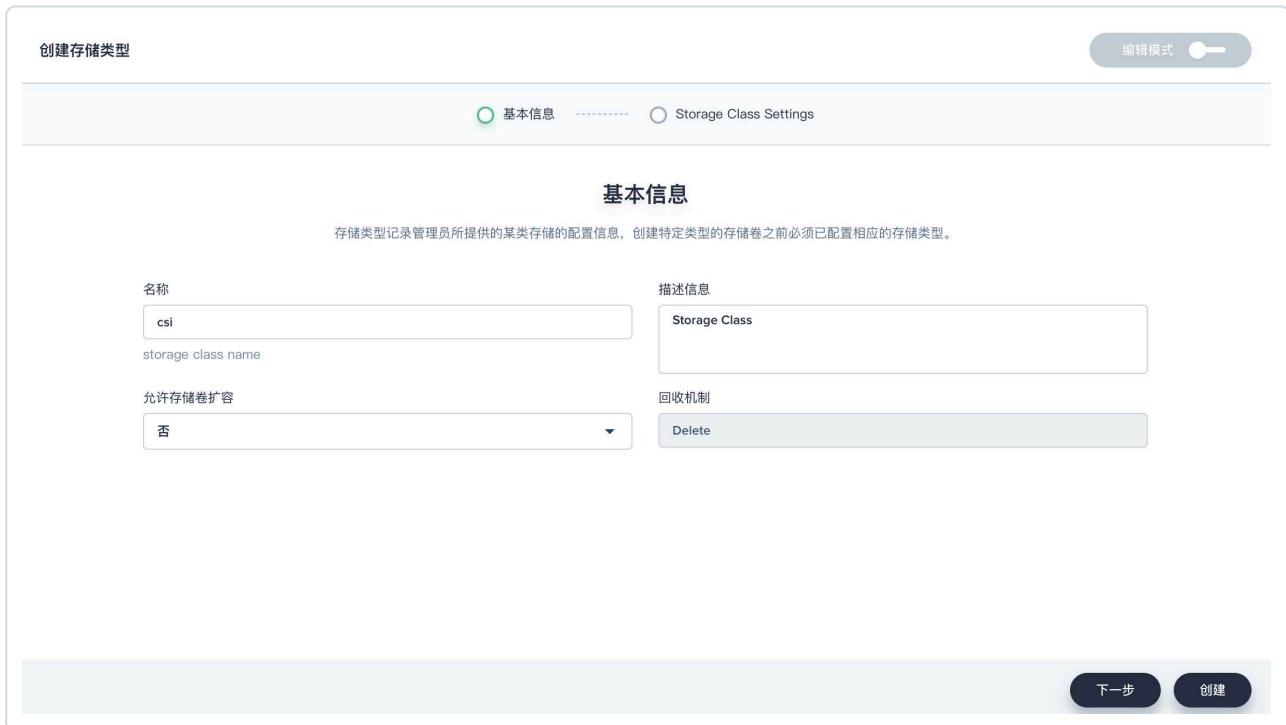
编辑模式

基本信息 ······  Storage Class Settings

### 基本信息

存储类型记录管理员所提供的某类存储的配置信息，创建特定类型的存储卷之前必须已配置相应的存储类型。

名称 storage class name <input type="text" value="csi"/>	描述信息 Storage Class <input type="text" value=""/>
允许存储卷扩容 <input type="button" value="否"/>	回收机制 <input type="button" value="Delete"/>



## 第二步：存储类型设置

设置存储类型的详细参数，这一步的参数会根据 供应者 (Provisioner) 不同而变化，以设置青云块存储插件 CSI-QingCloud 为例，其他存储类型的参数释义参见 [存储配置说明](#)。

创建存储类型

基本信息 存储类型设置

编辑模式

### 存储类型设置

存储类型记录管理员所提供的某类存储的配置信息，创建特定类型的存储卷之前必须已配置相应的存储类型。

供应商: csi-qingcloud 提供后端存储: type: stepSize: fsType: maxSize: minSize:

In QingCloud public cloud platform, 0 represents high performance volume, 3 represents super high performance volume.

Set the increment of volumes size in GiB

Limit the range of volume size in GiB

Limit the range of volume size in GiB

ext4 ext3, ext4, xfs

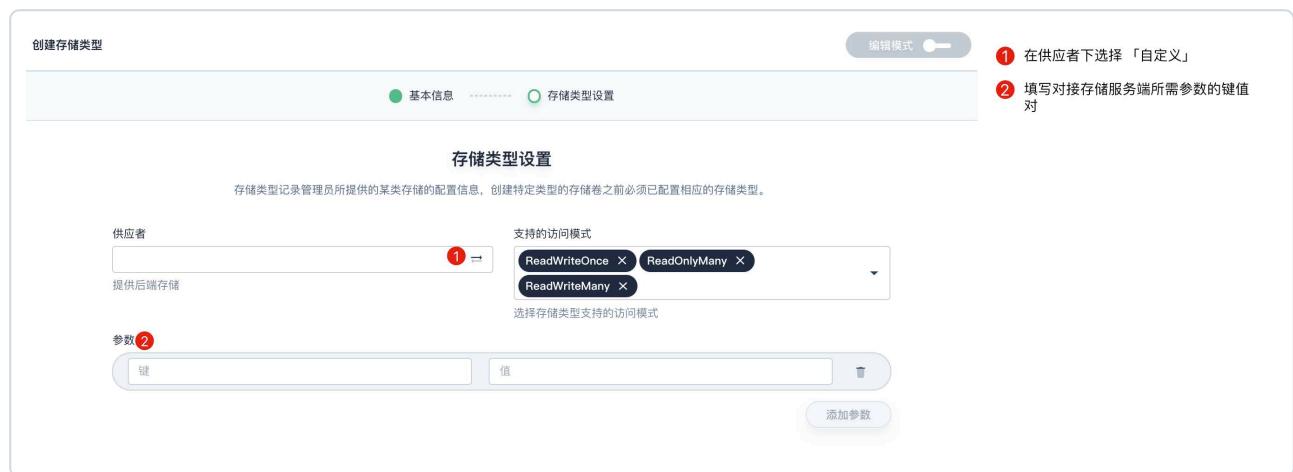
上一步 创建

- 供应商 (Provisioner): 实际上是个存储分配器，用来决定使用哪个卷插件分配 PV，例如选择 csi-qingcloud, Ceph RBD 或 GlusterFS。
- 访问模式 (Access Modes): 指定 PV 的访问模式，每个 PV 都有一套自己的用来描述特定功能的访问模式。注意，一个卷一次只能使用一种访问模式挂载，即使它支持多种访问模式。
  - ReadWriteOnce——该卷可以被单个节点以读/写模式挂载。
  - ReadOnlyMany——该卷可以被多个节点以只读模式挂载。
  - ReadWriteMany——该卷可以被多个节点以读/写模式挂载。
- type: 云平台存储卷类型。比如在青云的公有云上，0 代表性能型硬盘；3 代表超高性能型硬盘；1 或 2（根据集群所在区不同而参数不同）代表容量型硬盘。详见 [QingCloud 文档](#)。
- maxSize/minSize: 限制存储卷类型的存储卷容量范围，单位为 GiB。
- stepSize: 设置用户所创建存储卷容量的增量，单位为 GiB。
- fsType: 卷的文件系统，支持 ext3, ext4, xfs. 默认为 ext4。

# 创建自定义存储类型

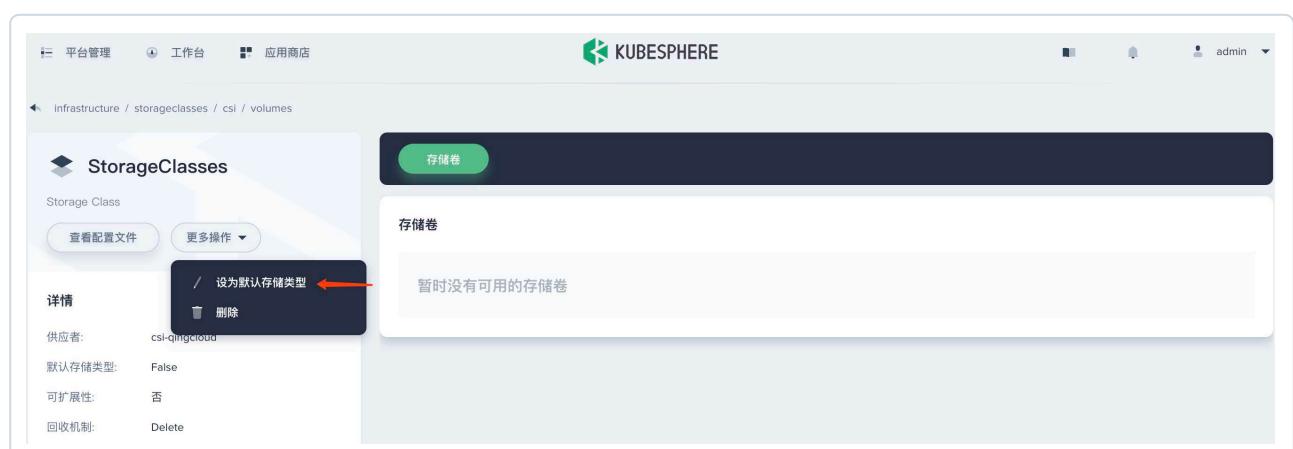
在存储类型设置中，除了可以选择 KubeSphere 支持的几种存储类型以外，还支持创建自定义的存储类型，对接用户自有的存储服务端。

创建自定义的存储类型，需要预先在 KubeSphere 后端环境预先安装某个存储服务端的客户端，然后在创建存储类型的设置中自定义其所需参数的 key-value，去对接它的存储服务端。



# 设置默认存储类型

一个 Kubernetes 集群中仅允许设置一个默认存储类型，在存储类型的详情页点击 **更多操作** 可设置默认的存储类型。若需要删除存储类型同样在当前页面操作。



## 监控概述

### 容器平台监控面临的挑战

根据 CNCF 最新调查显示，有 38% 的用户认为监控是其应用 Kubernetes 遇到的最大挑战之一，随着企业规模的增长，这个比例甚至达到了 46%。这其中存在的重要原因是与传统监控相比，容器与微服务的监控面临着更多难点和考验，因为容器监控与传统监控的区别在于：

- 监控频率不同：容器启停都是秒级，而且扩缩容也非常快，传统系统监控可能是分钟级的，容器系统监控是秒级的；
- 数量级不同：传统的监控针对基础设备，数量较少，容器是运行在基础设施之上，每个节点可以支撑多个容器实例，容器数量通常是以前监控设备的数十倍以上；
- 更偏向应用监控：容器随时可以启停，具备可以动态扩缩容等特点，应用的健康状况对用户来说更为重要。

KubeSphere 监控系统支持大规模系统监控、多指标监控、多维度监控，为每一个层级资源的运行状态都提供实时的多种指标监控，而且收集资源实时监控数据和历史监控数据，旨在帮助用户观察和建立资源和集群性能的正常标准。通过不同时间、不同负载条件下监测集群各项基础指标，并以图表或列表的形式展现。例如，用户可以监控集群和节点的 CPU 利用率、内存使用率和磁盘 I/O、网卡流量等物理层级的基础指标，还可以监控平台的企业空间、容器组 (Pod) 和容器的 CPU 使用量、内存使用量等指标，以及项目资源用量、资源用量趋势和服务组件的状态。监控数据支持按节点、企业空间或项目排行。KubeSphere 监控还提供逐级钻取能力，用户可以很方便地掌握资源和业务的运行情况并快速定位故障。

### 监控指标

KubeSphere 的监控中心包括 [物理资源监控](#) 和 [应用资源监控](#) 两大类。通常，只有平台管理员 (cluster-admin) 或在该平台角色的权限列表中勾选了 [查看监控管理](#) 的用户才有权限在控制台查看监控中心，详见 [物理资源监控](#) 和 [应用资源监控](#)。

值得一提的是，监控中心支持用户按用量排序和自定义时间范围查询，帮助快速定位故障。

KubeSphere 对资源的监控从两条线提供多维度的监控指标，即

- 管理员视角: Cluster -> Node -> Pod -> Container
- 用户视角: Cluster -> Workspace -> Namespace -> Workload/Pod -> Container

⌚ 多维度监控 管理员视角 (集群 -> 节点 -> 容器组 -> 容器)

⌚ 集群

- ⌚ CPU 利用率(%)
- ⌚ CPU 使用量(核)
- ⌚ CPU 平均负载
- ⌚ 内存利用率(%)
- ⌚ 内存利用量(GiB)
- ⌚ 磁盘利用率(%)
- ⌚ 磁盘使用量
- ⌚ inode 使用率 (%)
- ⌚ 容器组数量变化
- ⌚ 磁盘吞吐(KB/s)
- ⌚ 服务组件状态
- ⌚ IOPS
- ⌚ 网卡流量(Kbps)
- ⌚ 网卡流量(Kbps)
- ⌚ 容器组运行状态

⌚ 节点

- ⌚ CPU 利用率(%)
- ⌚ 内存利用率(%)
- ⌚ 容器组使用情况
- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)
- ⌚ 容器组运行状态
- ⌚ CPU 平均负载
- ⌚ 磁盘吞吐(KB/s)
- ⌚ 网卡流量(Kbps)
- ⌚ IOPS

⌚ 容器组

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)
- ⌚ 网卡流量(Kbps)

⌚ 容器

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)

支持按用量排序和自定义时间范围查询, 帮助快速定位问题

⌚ 多维度监控 用户视角 (集群 -> 企业空间 -> 项目 -> 工作负载/容器组 -> 容器)

⌚ 集群

- ⌚ CPU 利用率(%)
- ⌚ CPU 使用量(核)
- ⌚ CPU 平均负载
- ⌚ 内存利用率(%)
- ⌚ 内存利用量(GiB)
- ⌚ 磁盘利用率(%)
- ⌚ 磁盘使用量
- ⌚ inode 使用率 (%)
- ⌚ 容器组数量变化
- ⌚ 磁盘吞吐(KB/s)
- ⌚ 服务组件状态
- ⌚ IOPS
- ⌚ 网卡流量(Kbps)
- ⌚ 网卡流量(Kbps)
- ⌚ 容器组运行状态

⌚ 企业空间

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)
- ⌚ 项目变化趋势
- ⌚ 资源用量趋势

⌚ 项目

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)
- ⌚ 网卡流量(Kbps)

⌚ 工作负载/容器组

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)
- ⌚ 网卡流量(Kbps)

⌚ 容器

- ⌚ CPU 使用量(核)
- ⌚ 内存利用量(GiB)

支持按用量排序和自定义时间范围查询, 帮助快速定位问题

从上图中不难发现，平台的监控指标和 IaaS 层相似，也就是我们常见的 CPU、内存、磁盘和网络等四个方面的使用量和使用率。另外我们也提供主机的 inode 监控，Kubernetes 对镜像和日志都有回收机制，但没有对 inode 的回收或清理机制，有可能发生 inode 已经用光，但是硬盘还未存满的情况，此时已无法在硬盘上创建新文件，可能会造成整个集群中某个节点无法创建工作负载。

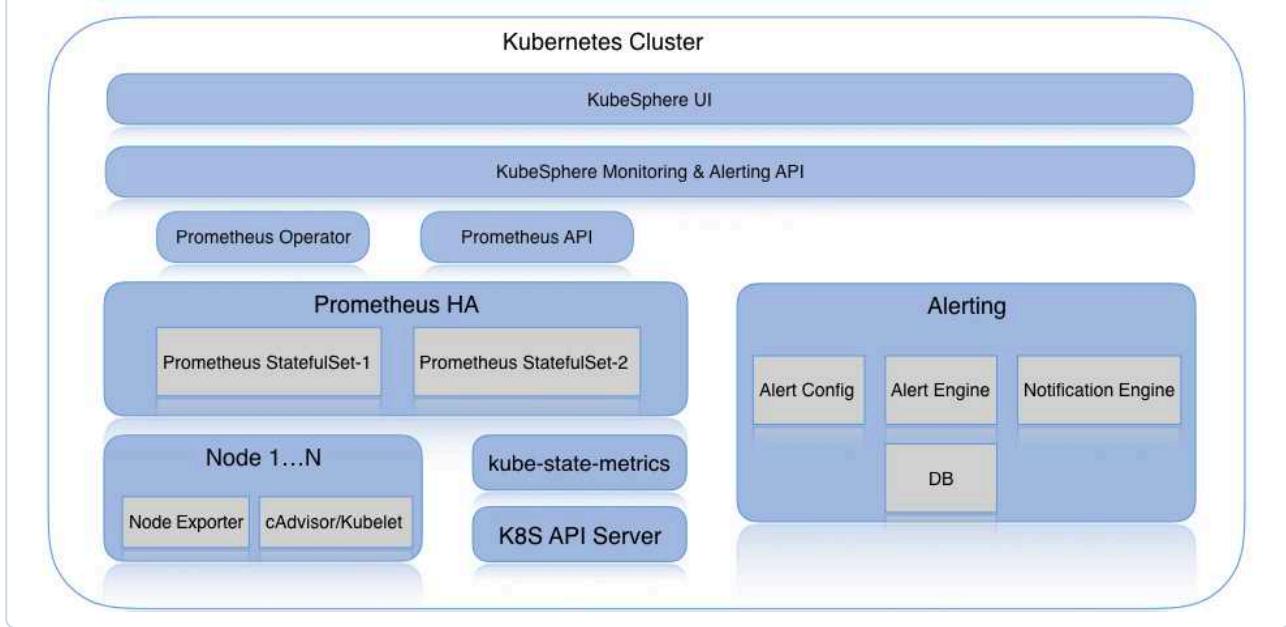
## 监控技术选型

KubeSphere 监控模块的设计和架构主要是采用了开源的解决方案，经过长时间的技术选型，最后选择了 [Prometheus](#)，主要原因有：

- Prometheus 和 Kubernetes 天然集成，为 CloudNative 而设计，多种工具和服务纷纷推出适配的 Exporter，供 Prometheus 抓取，Prometheus + Grafana 逐渐成为监控告警的首选；
- 采用主动拉数据的 Pull 模式，相当于有了数据获取 Agent 的能力；
- 2.0 之后重写了底层时序数据存储层，针对时序数据特点优化，能利用更少的资源存储和查询更多的数据；
- 灵活好用的查询语言 PromQL，能把 PromQL 嵌入 http 请求的强大的 API；
- OpenMetrics 项目推动 Prometheus exposure format 成为监控数据的标准格式；
- Prometheus Operator 打包了 Kubernetes 监控所需的所有必要组件，提供更丰富的监控数据；
- 单节点处理海量数据的强大能力。

KubeSphere 监控中心的所有组件都是容器化且可以部署在 Kubernetes 之上。下图是目前 KubeSphere 监控系统以及即将集成的告警模块的设计架构。

## KubeSphere Monitoring & Alerting



## 展望未来

目前 KubeSphere 监控中心已经具备了高可用和多维度的特性。在提供了监控之后，我们将在下一个版本支持创建告警、消息通知和日志收集等功能，支持根据给定阈值每隔一段时间发送告警，比如设置告警策略类型、告警规则和接收方式。

# 如何利用监控定位问题

KubeSphere 监控提供逐级钻取能力，对资源的监控从以下两条线提供多维度的监控指标，用户可以很方便地掌握资源和业务的运行情况并快速定位故障。

- 管理员视角：`Cluster -> Node -> Pod -> Container`
- 用户视角：`Cluster -> Workspace -> Namespace -> Workload/Pod -> Container`

我们在[示例五 – 设置弹性伸缩 \(HPA\)](#)中设置过一个 Load-generator 循环地向 Nginx 应用发送无限的查询请求访问应用的服务，模拟多个用户同时访问该服务，造成了 CPU 负载迅速上升。那么在本示例中，将 Nginx 应用的 Pod 所有副本设置一个节点选择器，使其 2 个副本仅运行在其中某一个节点，然后通过多维度监控逐级地去排查是哪一个节点的哪些容器造成的资源消耗上升。

## 查看负载前监控数据

在 Load-generator 创建之前，我们先记录一下此时集群的监控数据。目前集群一共三个节点，集群资源的监控数据记录如下：

初试时间	CPU 使用率	内存 (GiB)	本地存储 (GB)	容器组
08:48	12.39 %	13.56	46.68	68



# 查看负载后监控数据

## 第一步：查看集群资源监控

此时，参考示例五创建 HPA 和 Load-generator，并将设置了 HPA 的所有副本固定在某一个节点，待 Load-generator 开始工作后，理论上集群的 CPU 使用会有一个突增，在 **监控中心 → 物理资源 → 集群状态** 的 **集群资源使用状况** 监控数据中，我们发现 Load-generator 开始工作后集群 CPU 使用率的曲线在 **08:48 ~ 09:00** 有一个非常明显的上升，CPU 使用率在 **09:00** 已经上升至 **66.1 %**，**09:10** 高达 **67.90 %**。这种情况就需要引起集群管理员的特别注意了，具体是什么工作负载或服务造成 CPU 利用率突增，要去判断造成资源消耗突然增大的根源具体是在哪一个节点或企业空间下哪个项目的工作负载，并根据这一时段的监控数据状况去判断该情况是否属于正常，并观察该资源消耗的趋势是继续保持上升还是趋于平稳。

监控时间	CPU 使用率	内存 (GiB)	本地存储 (GB)	容器组
09:10	67.90 %	14.48	56.10	122



此时，最好的办法就是先判断这些资源负载的上升主要来自于哪个节点，该节点的 CPU 和内存使用量以及磁盘使用空间是否已经接近饱和或达到极限，是否存在因资源不足造成对宿主机的威胁。

## 第二步：查看节点用量排行

点击 **节点用量排行**，按 CPU 使用率、CPU 平均负载排行，可以很方便地发现 node2 节点的这两项指

标都排在第一位。那么就需要定位到 node2 查看该节点具体运行了哪些工作负载以及它们运行状况的监控数据。

节点用量排行						
按 CPU 使用率排行		导出				
节点	CPU	CPU 平均负载	内存	本地存储	inode 使用率	容器组
node2 172.20.1.4	76% 3.04 / 4.00 core	2.31	48% 3.69 / 7.80 GiB	25% 15.25 / 63.28 GB	7% 274809 / 3932160	43% 47 / 110
node1 172.20.1.3	72% 2.87 / 4.00 core	1.61	76% 5.91 / 7.80 GiB	25% 15.68 / 63.28 GB	9% 338251 / 3932160	40% 44 / 110
master 172.20.1.2	57% 2.27 / 4.00 core	1.98	64% 4.93 / 7.80 GiB	45% 28.35 / 63.28 GB	8% 276556 / 3932160	29% 31 / 110

### 第三步：查看节点监控

#### 查看资源状态

点击 node2 或从 **基础设施 → 主机** 中查看主机列表，即可看到所有节点在当前时刻的资源使用量。例如在主机列表中，node2 的 CPU 使用量和容器组数量是排在第一位的。

基础设施		主机						
		KubeSphere 集群中的计算能力由主机提供，集群中的节点是所有容器组运行所在的工作主机。			3	1	3	
		官网文档 参考文档			节点数量	主节点	计算节点	
<input type="text"/> 输入查询条件进行过滤								
名称	状态	角色	污点	CPU(Core)	内存(GiB)	容器组		
node2 172.20.1.4	运行中	node	-	3.06/4	3.71/7.8	47/110		
node1 172.20.1.3	运行中	node	-	2.7/4	5.93/7.8	44/110		
master 172.20.1.2	运行中	master, node	-	2.29/4	4.96/7.8	31/110		

进入 node2 详情页，首先可以查看主机的资源状态和节点状态，其中资源状态四项指标的饼图显示均为正常（若资源不足饼图则显示黄色或红色），并且通过节点状态的五个属性也可以判断出当前节点的 CPU、内存或存储的压力并未超过该节点的最大负荷，关于节点的五个属性释义，详见 [主机管理](#)。



**注意：如果此时发现节点的资源状态或节点状态的监控数据显示当前节点的 CPU 或内存使用量已经接近总量，没有充足的资源可供新的 Pod 调度到该节点运行，那么便需要为该节点添加污点并设置 effect 规则，不允许新的 Pod 调度到该节点，详见 [污点管理](#)。**

## 查看监控指标

在上面的 Tab 中点击 **监控**，查看 node2 的监控详情，右侧支持按时间范围和间隔查看历史监控数据，我们查看最近三小时的数据可以发现从初始时间 08:48 到 09:00 时段 CPU 使用率和 CPU 平均负载同时有一个非常明显的上升，这与我们在同一时间段看到的 **集群资源使用状况** 监控数据的趋势是基本一致的，因此可以判断造成资源消耗突然上升的工作负载或服务大概率就落在 node2 中，那么我们可以进一步去查看具体是企业空间下的哪个项目中的工作负载引起的。

**说明：在 09:00 到现在时刻，可以发现 CPU 利用率和平均负载有一个趋于平缓的曲线，该情况即可反馈两类信息：**

- 当前节点在 09:00 以后的状态已恢复正常工作状态，但由于工作负载的数量增加，其使用率要比之前高出很多，可以继续保持观察
- 该曲线在 09:00 以后趋于平稳是因为弹性伸缩 (HPA) 开始工作，使 Nginx 服务后端的 Pod 数量增加来共同处理 Load-generator 的循环请求，这也是 HPA 的工作原理，详见 [示例二 - 弹性伸缩工作原理](#)。



下拉查看 node2 节点的 **IOPS**, **磁盘吞吐** 读写和 **网络带宽** 出入的监控曲线，在初始时间 08:48 至 09:00 这个时间段也有较为明显的上升趋势，之后渐渐趋于平稳。



### 第三步：查看容器组监控

在上面的 Tab 点击 **容器组**，查看 node2 上运行的所有容器组 (Pod) 的监控数据。通过 **容器组数量变化** 的监控曲线可以发现在 08:48 至 09:00 这个时间新增了 20 个容器组调度到了 node2，因此可以初步判断 CPU 使用率的突然上升是由于容器组数量的增加而不是由于物理资源异常造成的。在容器组列表中，可以查看所有 30 分钟内容器组的 CPU 和内存的使用量，理论上在 08:48 到 09:00 这个时间段，最初创建的 2 个 hpa-example 容器组的 CPU 使用量也会有一个明显的上升趋势，因此点击查看其中一个容器组的监控数据。

The screenshot shows the KubeSphere interface for a node named 'node2'. On the left, there's a sidebar with labels like 'beta.kubernetes.io/arch: amd64', 'beta.kubernetes.io/os: linux', 'kubernetes.io/hostname: node2', and 'node-role.kubernetes.io/node: role node'. The main area has tabs for '运行状态' (Running Status), '容器组' (Container Group), '注解' (Annotations), '监控' (Monitoring), and '事件' (Events). The '容器组' tab is selected. It displays a pie chart showing '容器组使用情况' (Container Group Usage) with 47/110, and a line graph showing '容器组数量变化' (Container Group Number Change) over time from 07:03 to 10:03, with a peak at 09:03. Below this, a table lists three container groups:

容器组	IP 地址	状态	CPU 使用量	内存 使用量
hpa-example-7d7c9b9554-jtvnn	10.233.75.30	正在运行	42.29m	4.39MiB
hpa-example-7d7c9b9554-j4lhv	10.233.75.40	正在运行	49.80m	4.18MiB
hpa-example-7d7c9b9554-zbmkd	10.233.75.31	正在运行	(2018-12-15 10:18)	已使用 4.11MiB

查看其中一个 CPU 使用量曲线有明显上升趋势的容器组 `hpa-example-7d7c9b9554-zbmkd`，右上角选择自定义时间范围为最近 2 小时，可以看到该容器组的 **CPU 使用量** 和 **网络流出速率** 在相同时间段内有较为明显的上升，这与第三步中阐述的现象是相符的，因此可以进一步查看该 Pod 中的容器监控状态，并判断其是否运行正常。



## 第四步：查看容器监控

在上面的 Tab 点击 **资源状态**，点击容器进入容器详情页，查看容器的监控数据，在 **08:48 ~ 09:10** 也有一段明显的上升，但往后也基本趋于平稳。这个趋势与 HPA 的过程也是相符合的。



本示例说明了如何通过多维度监控，逐级地去排查问题和定位原因。

# 物理资源监控

KubeSphere 监控中心在 **物理资源监控** 提供了物理资源层级的 CPU、内存、网络和磁盘等相关指标的监控，并支持回看历史监控和节点用量排行。物理资源状态在控制台的 **平台管理 → 监控中心** 页面可供查看。

## 前提条件

已有集群管理员 (Cluster-admin) 账号，或当前用户的角色在权限列表中勾选了 **查看监控管理**。

## 查看集群状态

在左侧选择 **物理资源监控**，可以看到集群状态的监控页面，包括 **集群节点状态**、**组件状态**、**集群资源使用情况** 等三个部分。



## 集群节点状态

集群节点状态显示当前所有节点在线状态，支持下钻到主机管理页面查看所有主机实时的资源使用情况，点击 **节点在线状态** 进入主机管理列表。

例如以下列表的 [192.168.0.55](#) 节点，显然它的内存空间不足，点击进去查看该主机的详情页面。

节点数量: 9  
主节点: 1  
计算节点: 8

名称	状态	角色	污点	CPU(Core)	内存(GiB)	容器组
i-76v18j2o 192.168.0.53	运行中	worker	-	0.58/4	4.57/7.8	44/60
i-gddrhbxs 192.168.0.14	运行中	worker	-	0.7/4	6.78/7.8	40/60
i-wo83nj02 192.168.0.52	运行中	worker	-	0.38/4	5.19/7.8	28/60
i-8q3txahx 192.168.0.18	运行中	worker	-	1.03/4	3.81/7.8	39/60
i-5xclidxos 192.168.0.55	运行中	worker	-	3.53/4	7.27/7.8	39/60

从节点详情页，当前主机的 CPU、内存、本地存储、容器组使用情况从监控反馈的饼图中即可一目了然。明细发现内存使用率已高达 93%，这种情况可能需要对该主机采取措施，比如对其进行扩容或通过添加污点的方式禁止调度新的工作负载到当前主机。

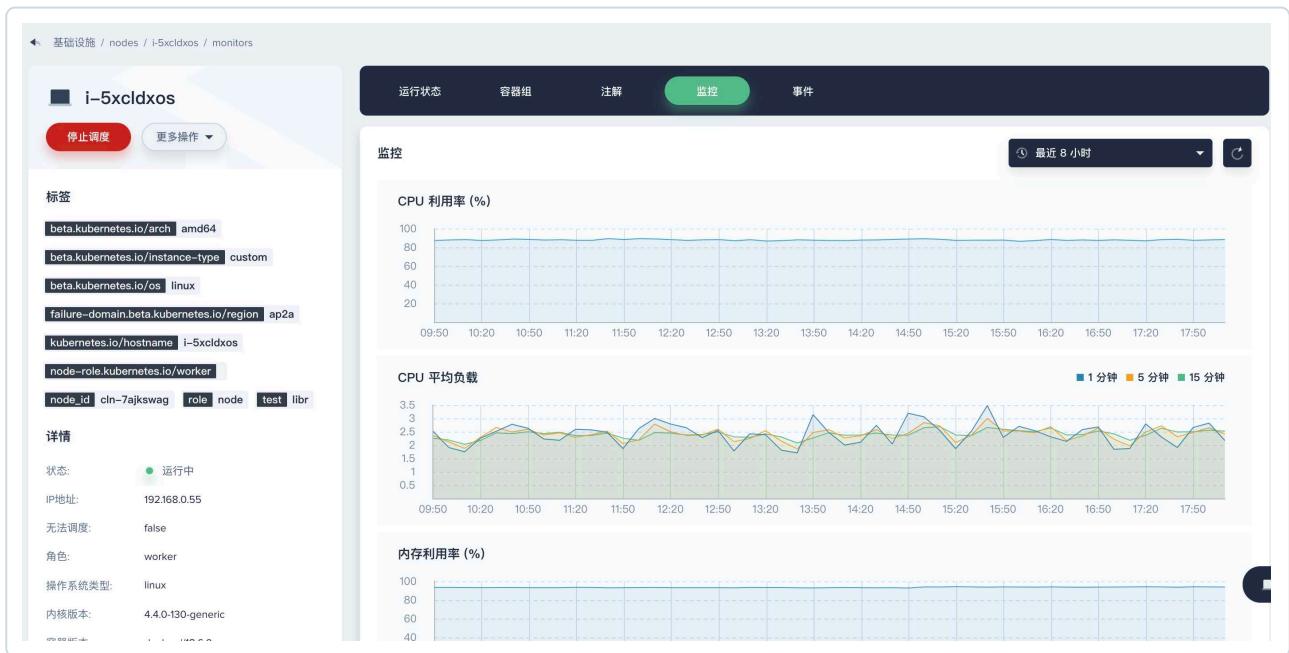
资源状态

CPU 使用情况 已使用: 3.54 core 总计: 4.00 core	内存使用情况 已使用: 7.28 GiB 总计: 7.80 GiB	本地存储 已使用: 34.77 GB 容量: 52.71 GB	容器组使用情况 已使用: 40 容量: 60
---	---	---------------------------------------	------------------------------

节点状态

KernelDeadlock 最近更新时间: 2018-11-29 21:08:30 原因: KernelHasNoDeadlock 消息: kernel has no deadlock	OutOfDisk 最近更新时间: 2018-12-03 17:59:33 原因: KubeletHasSufficientDisk 消息: kubelet has sufficient disk space available
MemoryPressure 最近更新时间: 2018-12-03 17:59:33 原因: KubeletHasSufficientMemory 消息: kubelet has sufficient memory available	DiskPressure 原因: KubeletHasNoDiskPressure

在主机详情页点击 **监控** tab，可以查看该主机的 CPU 使用率、CPU 平均负载、内存使用率、磁盘使用率、inode 使用率、IOPS、磁盘吞吐、网卡速率。对于主机而言，除了 CPU、内存、磁盘等使用率、磁盘吞吐这类常用监控指标，inode 使用率和 CPU 平均负载的监控信息对于主机资源管理也是非常有帮助的。



## 查看组件状态

服务组件提供 KubeSphere、Kubernetes 和 OpenPitrix 集群内各项服务组件的健康状态监控，若某些核心的服务组件出现异常，可能会导致系统不可用。查看当前集群服务组件的健康状态和运行时间，能够帮助用户监测集群的状况和及时定位问题。

点击 **组件状态** 即可跳转到服务组件的详情页面，如下可以看到 OpenPitrix 有一个组件状态异常，其 Tab 显示黄色的异常状态。

## 服务组件

服务组件提供 KubeSphere、Kubernetes 和 OpenPitrix 集群内各项服务组件的健康状态监控，可以查看当前集群的健康状态和运行时间，能够帮助用户监测集群的状况和及时定位问题。

全部 33 KubeSphere 9 Kubernetes 8 Openpitrix 1

### OPENPITRIX

 openpitrix-api-gateway	● 健康状态	1 / 1 副本数量	14天 运行时间
 openpitrix-app-manager	● 健康状态	1 / 1 副本数量	14天 运行时间
 openpitrix-category-manager	● 健康状态	1 / 1 副本数量	14天 运行时间
 openpitrix-cluster-manager	● 健康状态	1 / 1 副本数量	14天 运行时间
 openpitrix-dashboard	● 健康状态	1 / 1 副本数量	14天 运行时间

## 查看集群资源使用情况

集群资源的监控指标包括当前集群所有节点的 **CPU、内存、磁盘等利用率和容器组使用数量变化**，点击左边的饼状图即可切换指标。监控的时间间隔为 40 分钟，四项监控指标的纵坐标都会随时间动态变化。该监控图可以一览整个集群的资源消化状况，对于集群管理员而言，是需要密切关注这部分的监控指标的。当资源接近饱和状态时，可以通过增加节点或扩容磁盘、内存等操作，调节集群的负载能力。



# 查看历史监控

历史监控数据能够帮助用户观察和建立资源和集群性能的正常标准，KubeSphere 支持查看集群物理资源的 7 天以内的监控数据，包括 CPU 利用率、内存利用率、CPU 平均负载（1 分钟 / 5 分钟 / 15 分钟）、inode 使用率、磁盘吞吐（读/写）、IOPS（读/写）、网卡速率（出/入）、容器组运行状态。

KubeSphere 支持自定义时间范围和时间间隔来查看历史监控状况。以下分别简单介绍每一项监控指标的意义。



## 监控指标

### CPU 利用率

CPU 利用率（%），是对一个时间段内 CPU 使用状况的统计，通过这个指标可以看出在某一个时间段内 CPU 被占用的情况。在监控中若发现某个时间段内系统的 CPU 使用率飙高时，首先要定位到是哪个进程占用的 CPU 较高。比如对于 Java 应用来说，可能存在内存泄漏问题或代码存在死循环这类情况。

## 运行状态

CPU 利用率 (%)



## 内存利用率

内存是计算机中重要的部件之一，它是与 CPU 进行沟通的桥梁，因此内存的性能对计算机的影响非常大。程序运行时的数据加载、线程并发、I/O 缓冲等都依赖于内存，可用内存的大小决定了程序是否能正常运行以及运行的性能，而内存利用率 (%) 可反映集群的内存利用状况和性能。

内存利用率 (%)



## CPU 平均负载

在说明其监控意义之前，先了解一下 CPU 平均负载的含义，它是单位时间内，系统处于可运行状态和不可中断状态的平均进程数，即平均活跃进程数，注意区别其与 CPU 使用率没有直接关系。那么平均负载为多少时是合理的？实际上，平均负载在理想状况下应该等于 CPU 个数，所以在判断平均负载大小时，

先确定系统有几个 CPU。只有平均负载比 CPU 个数多时，就说明系统出现了过载。

问题是，CPU 平均负载在下图中分为 1 分钟 / 5 分钟 / 15 分钟 三个数值，应该怎么看？

通常情况下，三个时间都要看，通过分析系统负载的趋势，能更全面地了解目前的负载状况：

- 如果 1 分钟 / 5 分钟 / 15 分钟 这三个时间的曲线在某个时间段内基本相似，说明集群的 CPU 负载比较稳定
- 如果在某个时间段（或时间点）1 分钟的值远大于 15 分钟则说明最近 1 分钟的负载呈增加趋势，需要保持观察，一旦 1 分钟的值超过了 CPU 个数可能意味着系统出现过载，就需要进一步分析问题来源了
- 反之，如果某时段或时刻 1 分钟的值远小于 15 分钟则说明最近 1 分钟内系统的负载在降低，而之前 15 分钟内已产生了很高的负载。



## 磁盘使用量

KubeSphere 的工作负载比如有状态副本集、守护进程集都依赖于持久化存储服务，并且其自身的一些组件和服务也需要持久化存储提供支持，而这类后端存储就依赖于磁盘，比如块存储或网络共享存储。为磁盘使用量提供实时的监控环境是保持数据高可靠性的重要部分，因为在 Linux 系统的日常管理中，平台管理员可能会遇到因磁盘空间不足导致数据丢失，甚至系统崩溃等情况。所以，关注系统的磁盘使用情况，并确保文件系统不被占满或滥用是集群管理的重要任务。通过监控磁盘使用量的历史数据，即可预先了解磁盘的使用情况，如果发现磁盘使用量过高，可以通过清理不必要的镜像或容器，来节省磁盘空间。



## inode 使用率

每个文件都必须有一个 inode，用于储存文件的元信息，比如文件的创建者、创建日期，inode 也会消耗硬盘空间，大量的 cache 小文件也容易导致 inode 资源被使用耗尽。并且，有可能发生 inode 已经用光，但是硬盘还未存满的情况，此时就无法在硬盘上创建新文件。

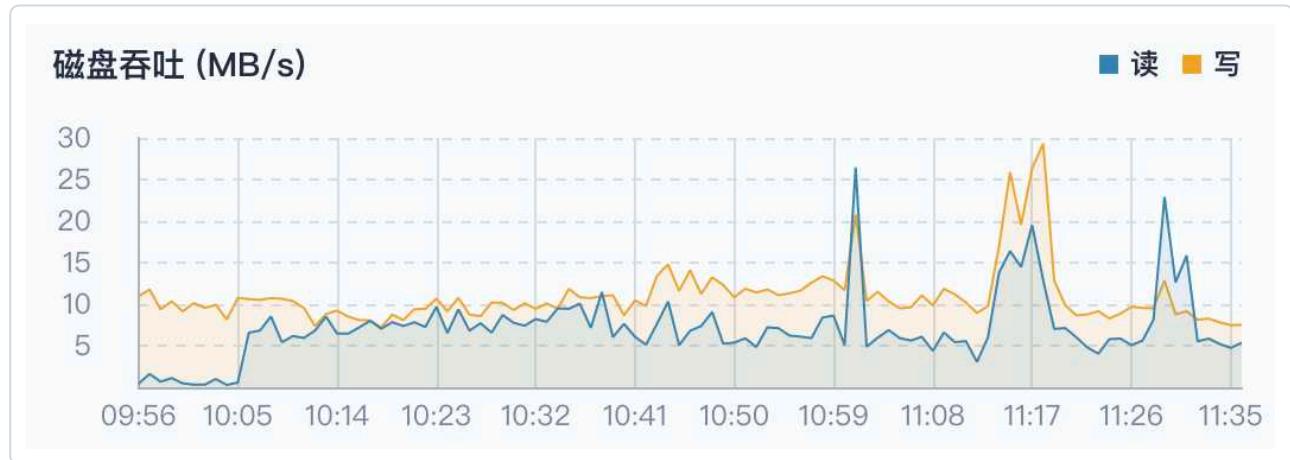
而 inode 使用率的监控恰好就可以预先发现上述提到的这类情况，帮助用户知道集群 inode 的使用情况，防止因 inode 耗尽使得集群无法正常工作，提示用户及时清理临时文件。



## 磁盘吞吐

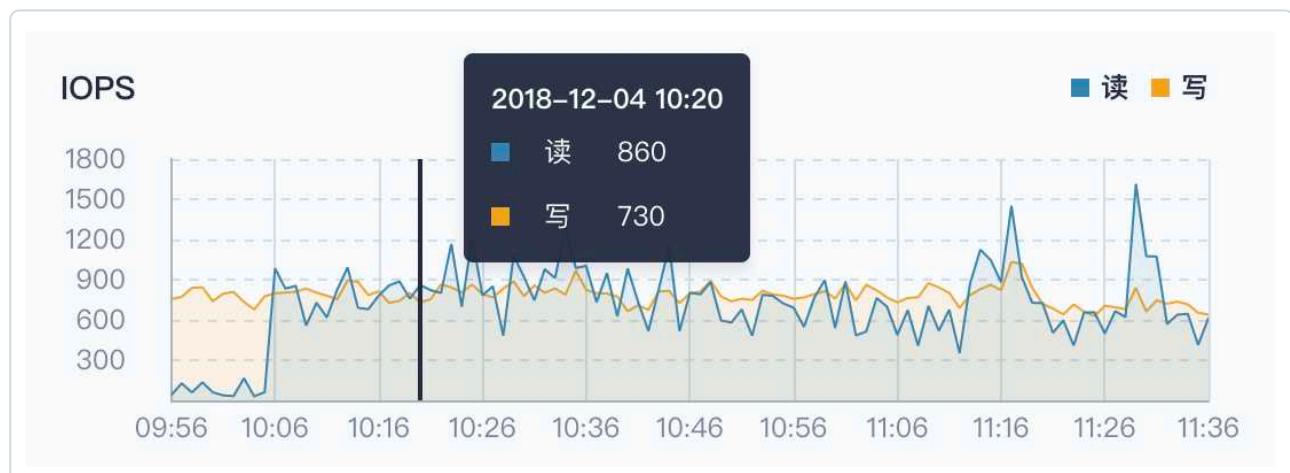
磁盘监控是 Linux 系统管理中一个非常重要的组成部分，以上提过了磁盘利用率的监控，那么磁盘吞吐和 IOPS 的监控也是不可或缺的，便于集群管理员进行调整数据布局等管理活动以达到优化集群总体性能的目的。磁盘吞吐 (Throughput) 指磁盘传输数据流的速度，单位是 MB/s，传输数据为读出数据和写入数据。

入数据的和。当传输大块不连续数据时，该指标有重要参考作用。



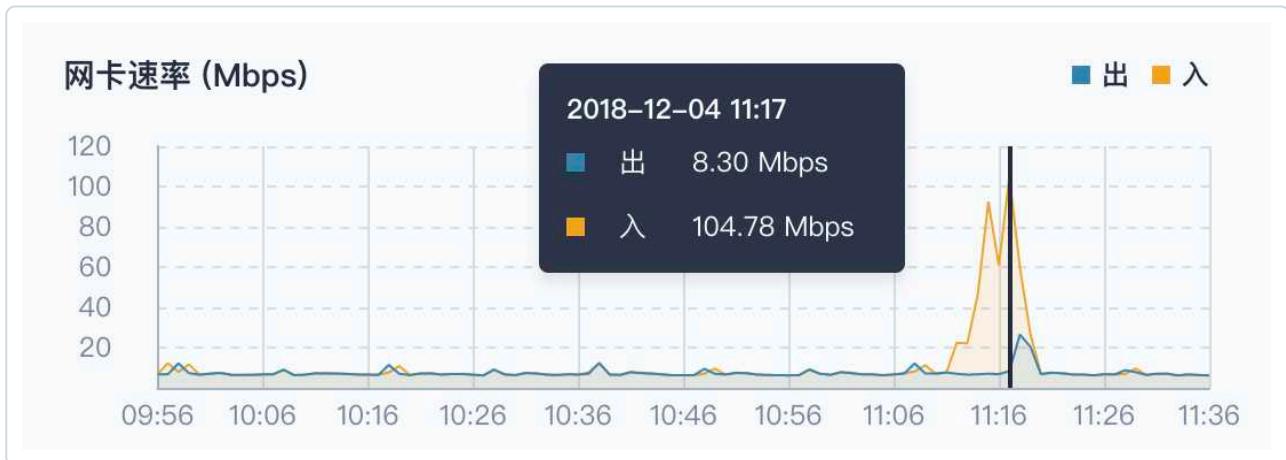
## IOPS

IOPS 对于磁盘来说，一次磁盘的连续读或者连续写称为一次磁盘 I/O，磁盘的 IOPS 就是每秒磁盘连续读次数和连续写次数之和。当传输小块不连续数据时，该指标有重要参考意义。



## 网卡速率

网卡速率是指网卡每秒钟接收或发送数据的能力，单位是 Mbps (兆位 / 秒)。



## 容器组运行状态

容器组 (Pod) 运行状态支持筛选 **运行中**、**异常中**、**已完成** 三种状态的容器组总数量。已完成状态通常是任务 (Job) 或定时任务 (CronJob) 这类容器组，异常状态的容器组数量需要引起特别关注。



## 节点用量排行

节点用量排行功能对于主机监控是非常实用的，支持按 **CPU 使用率**、**CPU 平均负载**、**内存使用率**、**本地存储用量 (磁盘使用量)**、**inode 使用率**、**容器组用量** 这一类指标进行排行，支持升序和降序排列。管理员通过按指标进行排序即可快速发现潜在问题或定位某台节点资源不足的情况。比如，按内存使用率降序排行，可以发现排行前两位的主机内存已经非常高了，管理员可以通过扩容内存、打上污点或其它手段来防止这两台主机因内存不足而不工作。

## 节点用量排行

按内存使用率排行

导出

节点	CPU	CPU 平均负载	内存	本地存储	inode 使用率	容器组
i-5xcldxos 192.168.0.55	89% 3.53 / 4.00 core	2.44	95% 7.33 / 7.80 GiB	67% 35.07 / 52.71 GB	30% 968931 / 3276800	62% 37 / 60
i-ai2p6j3y 192.168.0.73	31% 1.23 / 4.00 core	0.33	94% 3.61 / 3.86 GiB	33% 34.01 / 105.56 GB	13% 813671 / 6553600	35% 21 / 60
i-rgem3qkr 192.168.0.74	14% 0.55 / 4.00 core	0.39	88% 3.36 / 3.86 GiB	31% 32.18 / 105.56 GB	12% 755731 / 6553600	45% 27 / 60
i-wo83nj02 192.168.0.52	10% 0.36 / 4.00 core	0.08	75% 5.84 / 7.80 GiB	81% 42.42 / 52.71 GB	36% 1170292 / 3276800	57% 34 / 60
i-gddrhbxs 192.168.0.14	37% 1.47 / 4.00 core	0.48	75% 5.80 / 7.80 GiB	53% 27.44 / 52.71 GB	24% 760418 / 3276800	79% 47 / 60
i-6soe9z11 192.168.0.57	44% 3.45 / 8.00 core	0.52	67% 5.19 / 7.80 GiB	15% 5.91 / 42.14 GB	4% 94887 / 2621440	24% 14 / 60
i-e8oiua90 192.168.0.56	20% 0.79 / 4.00 core	0.21	60% 4.66 / 7.80 GiB	54% 28.10 / 52.71 GB	25% 790103 / 3276800	37% 22 / 60

# 应用资源监控

集群管理员除了需要关注物理资源层面的监控数据，还需要了解在整个平台中，用户实际上使用了多少应用资源，如项目数量、DevOps 工程数量，有多少个具体类型的工作负载和服务。应用资源监控是查看平台的应用级别的资源用量和变化趋势的汇总情况。

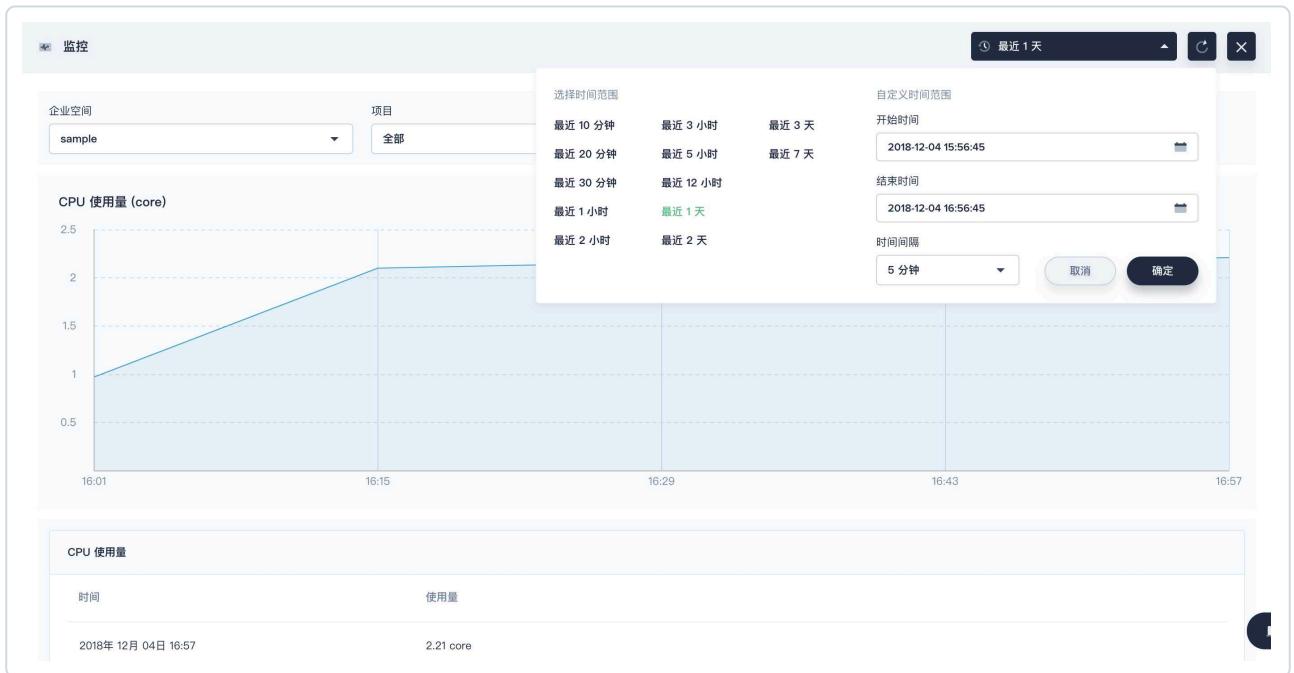
## 资源用量

The screenshot shows the 'Application Resource Monitoring' section of the KubeSphere dashboard. It includes a navigation bar with tabs for 'Resource Usage' and 'Usage Ranking'. Below the navigation are four summary cards: 'Enterprise Space' (21), 'Accounts' (45), 'Projects' (115), and 'DevOps Projects' (26). The main area displays 'Cluster Resource Usage' with metrics: 7.86 core CPU and 32.23 GiB Memory. It also shows 'Application Resource Usage' with metrics: 147 Deployments, 13 StatefulSets, 7 DaemonSets, 40 Tasks, 5 CronJobs, and 59 PersistentVolumeClaims. A time range selector at the top right indicates 'Recent 1 Day'.

其中，集群资源使用情况和应用资源使用情况保留最近 7 天的监控数据，支持自定义时间范围查询。

This screenshot shows the 'Cluster Resource Usage' detail page. It features a summary card for 'CPU' usage (7.71 core) and a 'Deployment' count (147). To the right is a detailed time range selector. It includes a 'Recent 1 Day' button, a date range from '2018-12-04 15:52:16' to '2018-12-04 16:52:16', and a 'Custom Time Range' section with dropdowns for 'Start Time' and 'End Time' and a 'Time Interval' dropdown set to 'Please Select'. Buttons for 'Cancel' and 'Confirm' are at the bottom right.

点击具体的资源还可以查看集群在某个时间内的具体用量和变化趋势，比如点击 CPU 使用量进入其详情页。详情页支持按企业空间和项目查看具体的监控数据，用户可自定义时间范围。



## 用量排行

### 企业空间资源用量排行

用量排行支持对企业空间资源用量排行和项目资源用量排行，方便平台管理员了解当前集群中每一个企业空间的资源使用情况，包括 **CPU 使用量、内存使用量、容器组数量、网络流出速率、网络流入速率** 等 5 项指标，支持按其中任意一项指标升序或降序排行。

企业空间资源用量排行
项目资源用量排行

按 CPU 使用量排行
按内存使用量排行
按容器组用量排行
按网络流出速率排行
按网络流入速率排行

		CPU 使用量	内存使用量	容器组数量	网络流出速率	网络流入速率
	leo-2	3.27 core	16.00 GiB	165	94.61 Mbps	63.85 Mbps
	leo	2.24 core	4.76 GiB	51	90.85 Kbps	92.96 Kbps
	zhangli	1.85 core	5.57 GiB	19	29.39 Kbps	18.23 Kbps
	calvinwk01	10.87 m	132.31 MiB	3	2.11 Kbps	1.07 Kbps
		8.25 m	144.17 MiB	6	2.15 Mbps	15.19 Kbps
		1.38 m	123.69 MiB	1	0 bps	0 bps

## 项目资源用量排行

一个企业空间下可以有多个项目(namespace)，而不同项目之间的配额、资源用量和网络速率通过这部分监控都可以一目了然，同样支持以上提到的5项监控指标。集群管理员通过当前资源用量数据与配额进行比较后，可以根据监控情况来调整配额。

资源用量
用量排行

企业空间资源用量排行
项目资源用量排行

企业空间: sample
按 CPU 使用量排行
按↓

项目	CPU 使用量	内存使用量	容器组数量	网络流入速率	网络流出速率
default	2.16 core 配额: 18.00 core	4.54 GiB 配额: 20.00 GiB	30 配额: 200	119.24 Kbps	119.07 Kbps
project-a93lz1	5.88 m 配额: -	187.57 MiB 配额: -	3 配额: -	0 bps	0 bps
project-st75wr	0.01 m 配额: -	24.51 MiB 配额: -	17 配额: -	0 bps	0 bps
project-5puzt2	0.001 m 配额: -	3.55 MiB 配额: -	1 配额: -	0 bps	0 bps

# 应用仓库

KubeSphere 基于 [OpenPitrix](#) 构建了应用仓库服务，OpenPitrix 是由 [QingCloud](#) 主导开源的跨云应用管理平台，支持基于 Helm Chart 类型的 Kubernetes 应用。在应用仓库中，每个应用程序都是基础软件包存储库，如果要将 OpenPitrix 用于应用程序管理，则需要先创建存储库。应用程序管理器可以将包存储到 http / https 服务器或 S3 对象存储。OpenPitrix 应用仓库是独立于 OpenPitrix 的外部存储，可以是青云 QingCloud 的 QingStor 对象存储，也可以是 AWS 对象存储，里面存储的内容是开发者开发好的应用的配置包以及索引文件。注册好仓库后，存储的应用配置包会被自动索引成为可部署的应用。

## 添加应用仓库

使用集群管理员账户登录 KubeSphere 管理控制台，点击左上角 **平台管理** → **应用仓库**，进入列表页。



1. 点击右上角 **添加应用仓库** 按钮。
2. 在弹出窗口填入应用仓库的基本信息后，点击 **验证** 按钮。

创建 QingStor 对象存储，详见 [QingStor 官方文档](#)，获取密钥详见 [Access Key](#)。创建 AWS S3 对象存储获取 Access Key ID 和 Secret Access Key 详见 [官方文档](#)。

- 应用仓库名称：为应用仓库起一个简洁明了的名称，便于用户浏览和搜索。
- 类型：支持 Helm Chart 类型的应用
- URL：支持以下三种协议

- S3：支持将仓库内的应用部署到运行环境。QingStor 的 Bucket URL 是 http 开头，但是可以兼容 S3 协议，URL 按照 S3 风格 `s3.<zone-id>.qingstor.com/<bucket-name>/` 就可以使用 S3 接口访问 QingStor 服务。
  - HTTP：可读，不可写，仅支持获取该应用仓库（对象存储）中的应用，支持部署到运行环境，比如：<http://openpitrix.pek3a.qingstor.com/package/>，该示例仓库包含三个应用，创建后将自动导入到平台中。
  - HTTPS：可读，不可写，仅支持获取该应用仓库（对象存储）中的应用，支持部署到运行环境。
- 描述信息：简单介绍应用仓库的主要特性，让用户进一步了解该应用仓库；
  - 验证通过后，点击 确认 按钮完成应用仓库的添加。当添加应用仓库后，KubeSphere 会自动加载此仓库下的所有应用模板。

### 添加应用仓库

应用仓库名称 \*

类型

Helm

URL:

所输入的 URL 需要先验证才可进行添加或编辑操作

描述信息

QingCloud App repo

Google 有两个应用仓库可以试用，QingStor 对其中稳定的仓库做了一个 mirror (后续我们会开发商业版的应用仓库供企业使用)，用户可根据需要添加所需应用仓库：

- QingStor Helm Repo: <https://helm-chart-repo.pek3a.qingstor.com/kubernetes-charts/>
- Google Stable Helm Repo: <https://kubernetes-charts.storage.googleapis.com/>
- Google Incubator Helm Repo: <https://kubernetes-charts-incubator.storage.googleapis.com/>

在企业内私有云场景下，用户可以基于 [Helm](#) 规范去构建自己的应用仓库，并且可以开发和上传满足企业业务需求的应用到自己的应用仓库中，然后基于 KubeSphere 完成应用的分发部署。

# Jenkins 系统设置

Jenkins 功能强大的同时其本身也非常灵活，如今已成为 CI / CD 的事实标准，拥有一个活跃的社区来维护几乎任何工具和用例组合的插件。但灵活性需要付出代价：因为除 Jenkins 核心外，许多插件还需要设置一些系统级的配置才能完成工作。

KubeSphere 的 DevOps 工程底层基于 Jenkins 实现了容器化的 CI / CD 功能。为了给用户提供一个可调度的 Jenkins 环境，KubeSphere 使用了 **Configuration-as-Code** 进行 Jenkins 的系统设置，该设置需要用户在 KubeSphere 修改配置文件后再登录到 Jenkins Dashboard 的系统管理中执行重新加载。在当前的版本当中，在控制台中还未提供 Jenkins 的系统设置选项，将在后续版本中支持。

## 修改 ConfigMap

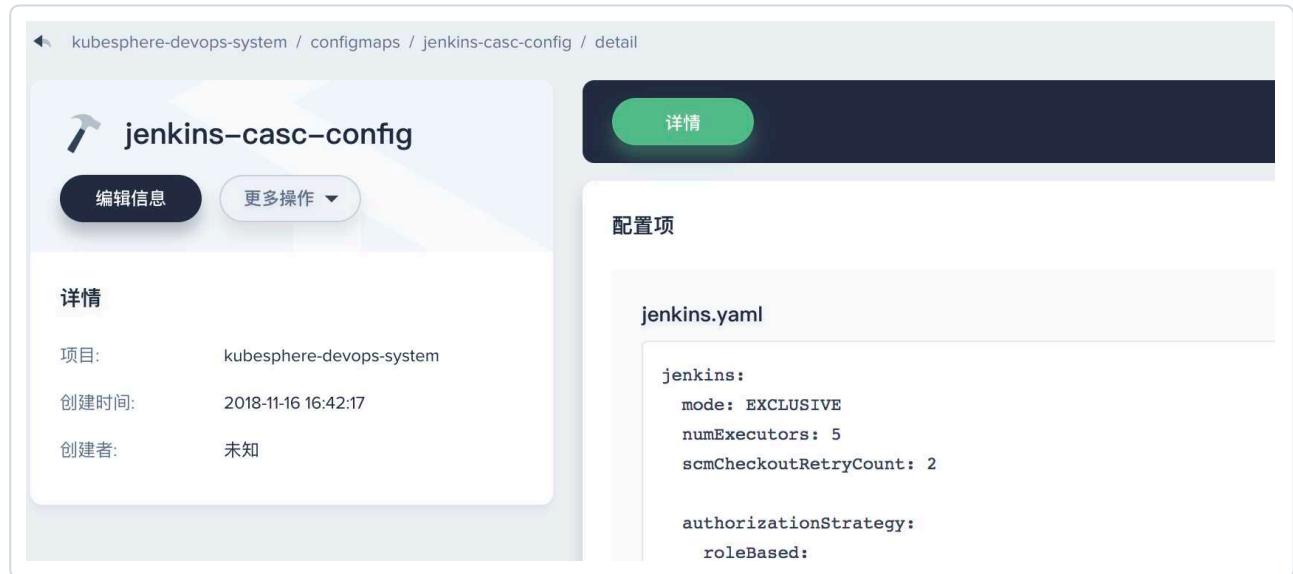
如果您是 KubeSphere 的系统管理员，若需要修改 Jenkins 的系统配置，建议您在 KubeSphere 使用 Configuration-as-Code (CasC) 进行系统设置，需要先在 KubeSphere 的配置 (ConfigMap) 中修改 `jenkins-casc-config`，然后再登录 Jenkins Dashboard 执行 **重新加载**。(因为通过 Jenkins Dashborad 直接写入的系统设置在 Jenkins 重新调度以后可能会被 CasC 配置所覆盖)。

系统内置的 Jenkins CasC 文件以 **ConfigMap** 的形式存储在 `/system-workspace/kubesphere-devops-system/configmaps/jenkins-casc-config/` 中，如下所示，若需修改可点击 **编辑 ConfigMap**。

The screenshot shows the KubeSphere configuration interface. On the left, there's a sidebar with project navigation (kubesphere-dev...), application management, and a 'Configuration' section highlighted with a red arrow. The main area is titled '配置' (Configuration) with a sub-section for 'Config' (配置). It displays a list of ConfigMaps, including 'jenkins', 'jenkins-tests', and 'jenkins-casc-config'. The 'jenkins-casc-config' entry is selected, indicated by a red arrow pointing to its row. A modal window is open over the list, showing options: '/ 编辑' (Edit), '/ 编辑配置文件' (Edit Configuration File), and '/ 编辑 ConfigMap' (Edit ConfigMap), with the last option also highlighted by a red arrow.

如下所示是 `jenkins-casc-config` 的配置模板，是一个 yaml 类型的文件。比如，可以在 ConfigMap 修

改代理 (Kubernetes Jenkins agent) 中的容器镜像、label 等这类信息或新增 podTemplate 中的容器。



The screenshot shows the KubeSphere UI for managing configmaps. On the left, there's a sidebar with a back arrow, the project name 'kubesphere-devops-system', and the configmap name 'jenkins-casc-config'. Below that is a '编辑信息' (Edit Information) button and a '更多操作' (More Operations) dropdown. On the right, there's a large green '详情' (Details) button. The main area is titled '配置项' (Config Items) and contains a code editor with the file 'jenkins.yaml' open. The code defines a Jenkins configuration with parameters like mode: EXCLUSIVE, numExecutors: 5, and scmCheckoutRetryCount: 2, along with an authorization strategy set to roleBased.

```
jenkins:
  mode: EXCLUSIVE
  numExecutors: 5
  scmCheckoutRetryCount: 2

  authorizationStrategy:
    roleBased:
```

在 KubeSphere 修改 **jenkins-casc-config** 以后，您需要在 Jenkins Dashboard 系统管理下的 **configuration-as-code** 页面重新加载您更新过的系统配置。

## 登陆 Jenkins 重新加载

1、Installer 安装将会同时部署 Jenkins Dashboard，Jenkins 已对接了 KubeSphere 的 LDAP，因此可使用用户名 **admin** 和 KubeSphere 集群管理员的密码登录 Jenkins Dashboard，访问公网 IP (EIP) + Nodeport (30180) 并登陆 Jenkins Dashboard。登陆后，在左侧导航栏点击 **系统管理**。

**说明：**访问 Jenkins Dashboard 可能需要将端口转发和防火墙放行该端口才可以在公网访问。



2、在控制台底部找到 Configuration as Code，点击进入。

**Manage and Assign Roles**  
Handle permissions by creating roles and assigning them to users/groups

**关于 Jenkins**  
查看版本以及证书信息。

**管理旧数据**  
从旧的、早期版本的插件中清理配置文件。

**Configuration as Code**   
Reload your configuration or update configuration source

3、在 Configuration as Code 部分点击 **重新加载**，即可将在 KubeSphere 的 ConfigMap 修改的系统配置重新加载并更新到 Jenkins Dashboard。

## Configuration as Code

Configuration loaded from :

- /var/jenkins\_home/casc\_configs/..data/jenkins.yaml

Last time applied :2018-11-24 上午09时33分43秒

**重新加载**

有关如何通过 CasC 进行系统设置，详见 [官方文档](#)。

**注:** 在现在版本当中，并不是所有插件都支持 CasC 的设置。CasC 只会覆盖使用 CasC 进行设置

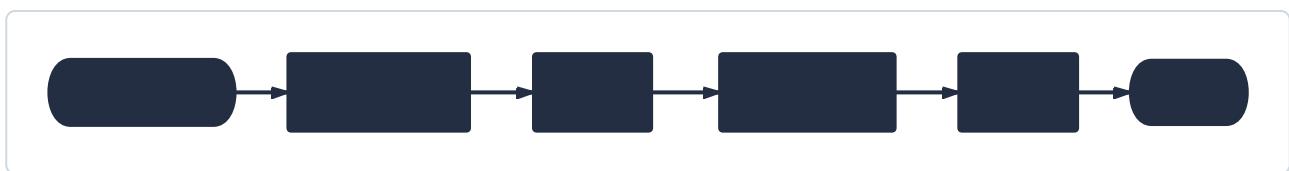
的插件配置。

# 应用模板

应用模板是 KubeSphere 中应用的存储、交付、管理途径，应用模板所纳管的应用基于 [Helm](#) 打包规范构建，并通过统一的公有或私有的应用仓库交付使用，应用可根据自身特性由一个或多个 Kubernetes 工作负载 (workload) 和服务 (Service) 组成。

应用模板通过可视化的方式在 KubeSphere 中展示并提供部署和管理的功能，用户能够基于应用模板快速地一键部署应用至所选的项目中。应用模板对内可作为团队间共享企业创造的中间件、业务系统等，对外可作为根据行业特性构建行业交付标准、交付流程和交付路径的基础，用户根据不同场景需求和可见级别服务于不同的业务场景。

在使用应用模板前，需要预先添加应用仓库。KubeSphere 基于 [OpenPitrix](#) 构建了应用仓库服务，在使用应用模板前可以添加一个包含应用的应用仓库，并将符合 Helm 规范的应用配置包上传至应用仓库后端的对象存储中，KubeSphere 会自动加载此仓库下的所有应用，详见 [添加应用仓库](#)。



除此之外，应用模板还可以结合 OpenPitrix 的应用全生命周期管理的功能，支持对接应用服务商、开发者和普通用户，通过应用上传、应用审核、部署测试、应用发布、应用版本管理等功能，构建公有或私有的应用商店，为 KubeSphere 提供应用模板服务，企业也可以基于此来建立行业公有或专有应用商店，实现标准化的应用一键交付部署，详见 [OpenPitrix 官方文档](#)。

## 应用列表

在所有的项目中，都提供了一个 **应用** 入口，这里作为应用模板的入口，应用部署后其也可以作为一个应用列表来管理当前项目下的所有应用。



点击 **部署新应用** 即可进入 **应用模板** 页。

## 应用模板

### 添加示例仓库

在前面提到过，使用应用模板前，需要集群管理员预先添加可用的应用仓库，用户才可以在应用模板中访问和部署应用。

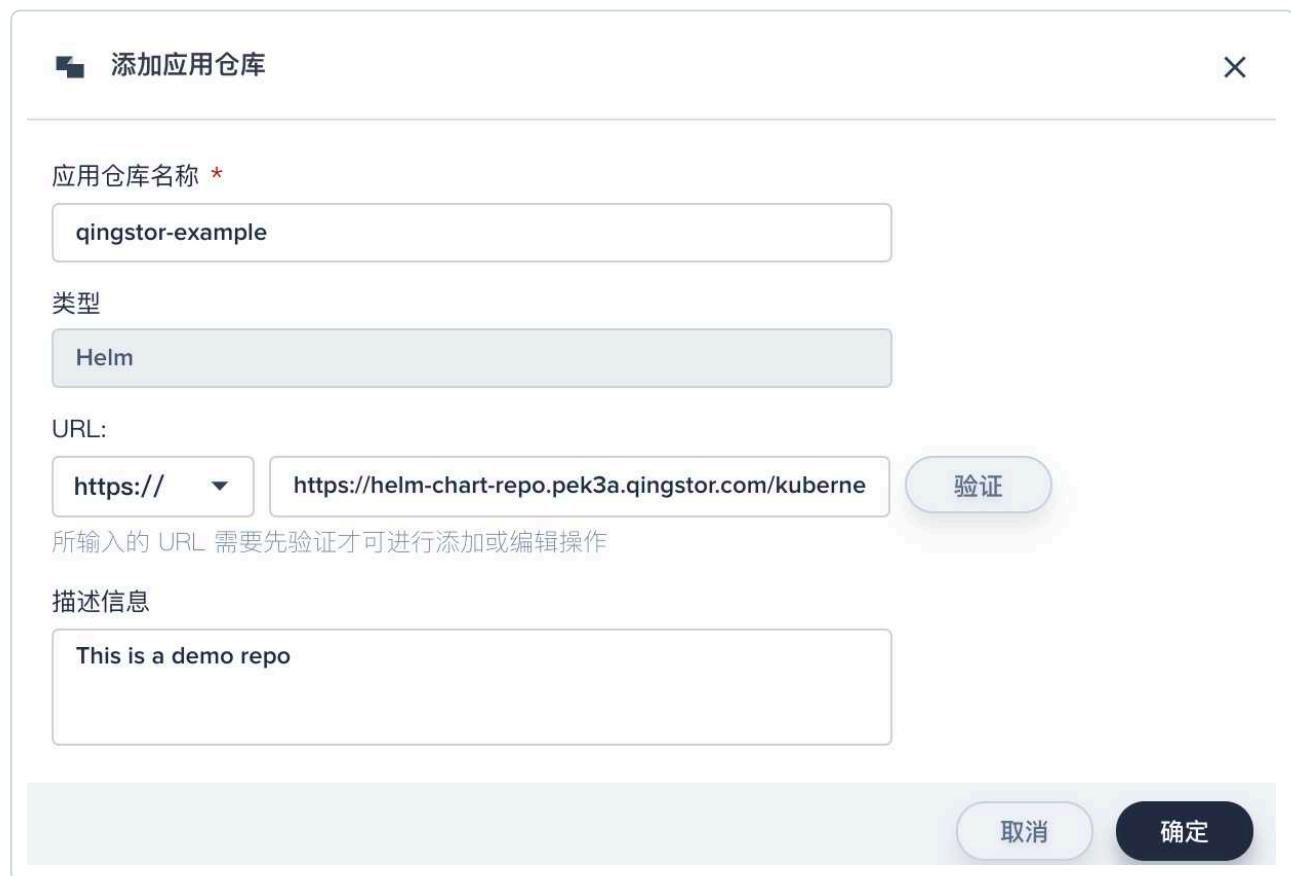
本文档提供了一个示例应用仓库仅用于功能演示，用户可根据需求自行在对象存储中上传应用配置然后添加应用仓库。

1、使用集群管理员账户登录 KubeSphere 管理控制台，点击左上角 **平台管理** → **应用仓库**，进入列表页。



2、点击右上角 **添加应用仓库** 按钮。

3、在弹出窗口填写示例应用仓库的基本信息，URL 选择 https，地址填写 <https://helm-chart-repo.pek3a.qingstor.com/kubernetes-charts/>，然后点击 验证 按钮，待验证通过后点击 确定，完成应用仓库的添加。



## 访问应用模板

切换为 project-regular 即项目的普通用户，登录 KubeSphere，点击控制台顶部的 **应用模板**，即可看到示例应用仓库中的所有应用已被导入到了应用模板中，此时用户即可浏览或搜索所需应用进行一键部署至所需的项目中。

The screenshot shows the KubeSphere application catalog interface. At the top, there are navigation tabs: '工作台' (Workstation) and '应用模板' (Application Template), with '应用模板' highlighted by a red arrow. The main title is '应用一键部署' (One-click Deployment Application). Below it, there's a search bar with the placeholder '请输入名称进行查找' (Enter name to search) and a dropdown menu '全部仓库' (All Repositories). A large blue banner at the bottom of the page reads '关于一键部署应用的步骤示例，参考 [快速入门 - 一键部署应用](#)' (Example steps for one-click deployment applications, refer to [Quick Start - One-click Deployment Application](#)).

应用模板	描述
<b>envoy</b> Version: 1.1.2 [1.6]  Envoy is an open source edge and service proxy, designed for cloud-native applications.	<b>sematext-docker-agent</b> Version: 0.2.0 [1.31.53]  Semantext Docker Agent
<b>zeppelin</b> Version: 1.0.1 [0.7.2]  Web-based notebook that enables data-driven, interactive data analytics and	<b>metallb</b> Version: 0.8.0 [0.7.3]  MetalLB is a load-balancer implementation for bare metal Kubernetes clusters
<b>mssql-linux</b> Version: 0.6.2 [14.0.3023.8]  SQL Server 2017 Linux Helm Chart	<b>cockroachdb</b> Version: 2.0.6 [2.1.1]  CockroachDB is a scalable, survivable, strongly-consistent SQL database.
<b>testlink</b> Version: 4.0.0 [1.9.18]	<b>stackdriver-exporter</b> Version: 0.0.4 [0.5.1]
<b>gce-ingress</b> Version: 1.0.0 [1.1.1]  A GCE Ingress Controller	<b>falco</b> Version: 0.5.3 [0.13.0]  Sysdig Falco
<b>mailhog</b> Version: 2.3.0 [1.0.0]  An e-mail testing tool for developers	<b>buildkite</b> Version: 0.2.4 [3]  DEPRECATED Agent for Buildkite
<b>elastic-stack</b> Version: 1.1.0 [6]  A Helm chart for ELK	<b>prometheus-to-sd</b> Version: 0.1.1 [0.2.2]  Scrape metrics stored in prometheus format and push them to the Stackdriver
<b>kapacitor</b> Version: 1.1.0 [1.5.1]	<b>phpmyadmin</b> Version: 1.3.0 [4.8.3]

# 工作负载概述

Kubernetes 中对一组 Pod 的抽象模型即工作负载，用于描述业务的运行载体，包括 部署 (Deployment)、有状态副本集 (Statefulset)、守护进程集 (Daemonset)、任务 (Job)、定时任务 (CronJob) 等。KubeSphere 控制台提供向导式的用户界面引导用户快速创建工作负载。

- [创建部署](#)

部署 (Deployment) 为 Pod 和 ReplicaSet 提供声明式定义方法，实现无状态应用伸缩、滚动升级、回滚的功能，常用来部署无状态应用实现快速的伸缩，相较于有状态服务，实例数量可以灵活伸缩。

- [创建有状态副本集](#)

有状态副本集 (Statefulset) 是为了解决有状态应用的问题，为应用提供数据的持久化存储、稳定的网络标志，有序的部署、升级、收缩功能，常用来部署数据库、缓存等有状态服务，通常情况只会用到一个实例。

- [创建守护进程集](#)

守护进程集 (Daemonset) 保证在每个 Node 上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。

- [创建任务](#)

任务 (Job) 负责批量处理短暂的一次性任务 (short lived one-off tasks)，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。

- [创建定时任务](#)

定时任务 (CronJob)，就类似于 Linux 系统的 Crontab，在指定的时间周期运行指定的任务。

## 工作负载基本操作

工作负载创建后，您可以对其进行查看、扩缩容、启停、删除、升级、资源监控等操作，详见 [工作负载管理](#)。

# 部署

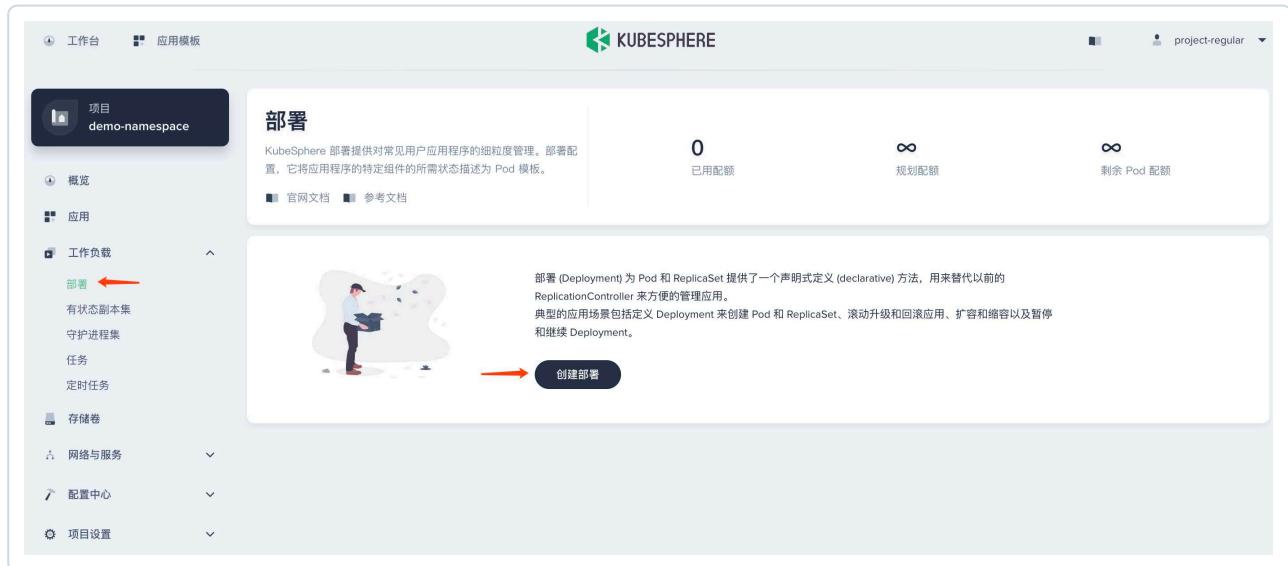
部署 (Deployment) 为 Pod 和 ReplicaSet 提供了一个声明式定义 (declarative) 方法来管理应用。典型的应用场景包括定义 Deployment 来创建 Pod 和 ReplicaSet、滚动升级和回滚应用、扩容和缩容以及暂停和继续 Deployment。

本文档仅说明创建部署中的可能用到的参数或字段意义，创建工作负载后应如何管理，请参考 [工作负载管理](#)。同时，[部署 Wordpress 示例](#) 也可帮助您快速理解 Deployment。

## 创建部署

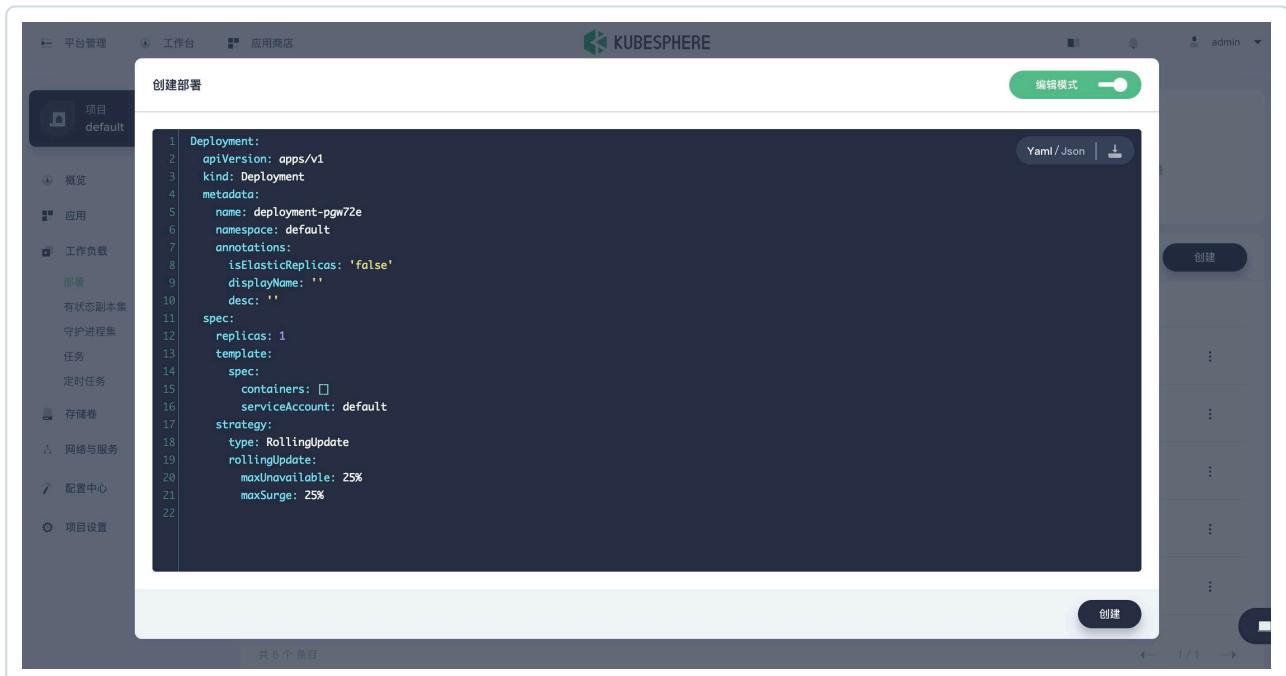
登录 KubeSphere 控制台，在已创建的项目下选择 [工作负载 → 部署](#)，进入部署列表页面。

左上角为当前所在项目，如果是管理员登录，可以看到集群所有项目的部署情况，如果是普通用户，则只能查看授权项目下的所有部署。列表顶部显示了当前项目的部署 Pod 配额和数量信息。



## 第一步：填写基本信息

1.1. 点击 **创建** 按钮，将弹出创建部署的详情页。创建部署支持两种方式，[页面创建](#) 和 [编辑模式](#) 创建。以下主要介绍页面创建的方式，若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式。左上角显示配置文件列表和导入导出按钮。编辑模式可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件并创建部署。



1.2. 在基本信息页，输入部署的名称，用户可以根据需求填写部署的描述信息。

- 名称：为创建的部署起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：简单介绍部署，让用户进一步了解部署的作用。

点击 **下一步**。

基本信息

您可以给部署起一个名字，以便在使用的时候容易区分。

名称 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾。

别名

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

项目

将根据项目进行资源进行分组，可以按项目对资源进行查看管理。

描述信息

## 第二步：配置容器组模板

2.1. 在 **容器组模板** 页面，用户可以设置 Pod 副本数量和弹性伸缩 [HPA](#)，HPA 能够使 Pod 水平自动缩放，提高集群的整体资源利用率。文档提供了一个弹性伸缩的示例并说明了弹性伸缩工作原理，详见 [设置弹性伸缩](#)。



点击 **添加容器**，然后根据需求添加容器镜像，容器中定义的镜像默认从 Docker Hub 中拉取。输入容器的名称和对应的镜像名，镜像名一般需要指定 tag，比如 wordpress:4.8–apache。

**说明：**若需要使用私有镜像仓库如 Harbor，参见 [镜像仓库 – 添加镜像仓库](#)。

为了实现集群的资源被有效调度和分配同时提高资源的利用率，平台采用了 request 和 limit 两种限制类型对资源进行分配。request 通常是容器使用的最小资源需求，而 limit 通常是容器能使用资源的最大值，设置为 0 表示对使用的资源不做限制，可无限的使用。request 能保证 pod 有足够的资源来运行，而 limit 则是防止某个 Pod 无限制的使用资源，导致其他 Pod 崩溃。

表1：CPU 配额说明

参数	说明
最小使用 (requests)	容器使用的 CPU 最小值，作为容器调度时资源分配的判断依赖。只有当节点上可分配 CPU 总量 $\geq$ 容器 CPU 最小值时，才允许将容器调度到该节点。
最大使用 (limits)	容器能使用的 CPU 最大值。

表2：内存配额说明

参数	说明
最小使用 (requests)	容器使用的最小内存需求，作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量 $\geq$ 容器内存申请数时，才允许将容器调度到该节点。
最大使用 (limits)	容器能使用的内存最大值，如果内存使用量超过这个限定值，容器可能会被 kill。

创建部署

编辑模式

基本信息  容器组模板  存储卷设置  标签设置  节点选择器

容器名称 \*  镜像 \*   
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾。  
要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU  
 最小使用  最大使用   
作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m

内存  
 Mi 最小使用  Mi 最大使用   
作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项 ^

就绪探针

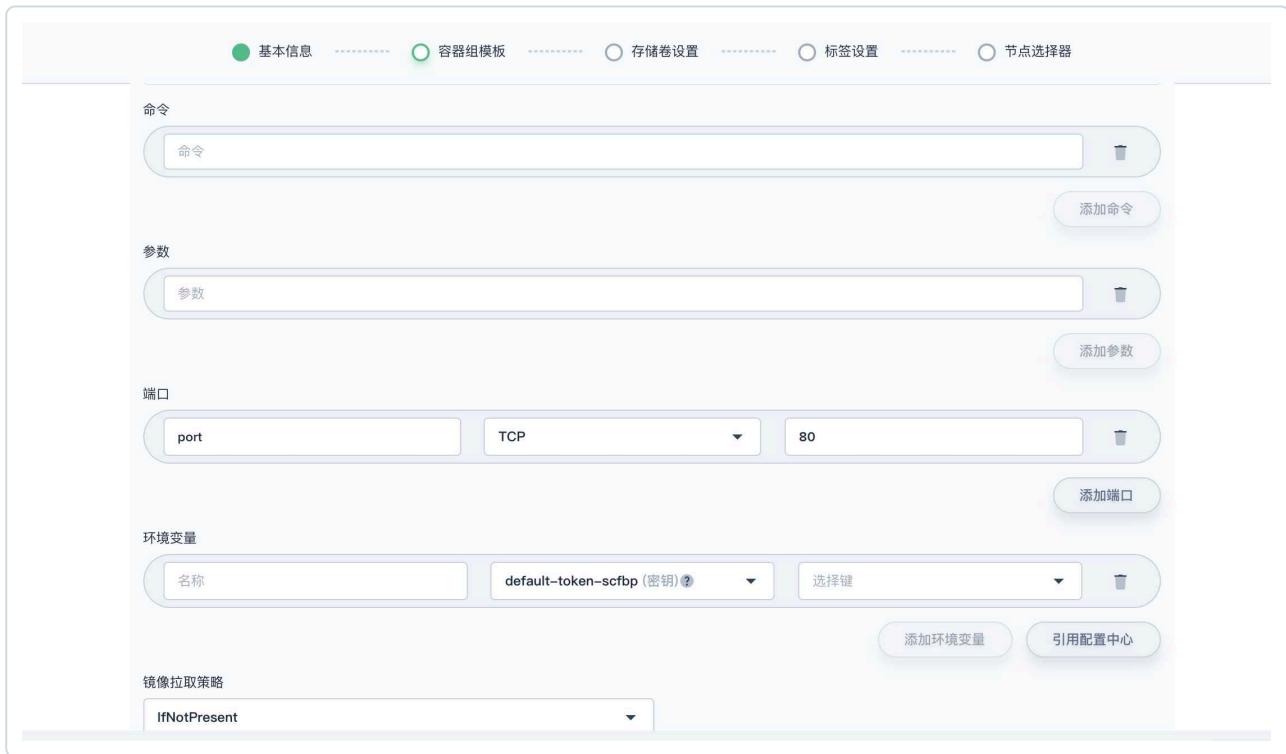
存活探针



2.2. 如果用户有更进一步的需求，可以点击 **高级选项**。

- **就绪探针/存活探针**: 在业务级的监控检查方面, Kubernetes 定义了两种类型的健康检查探针, 详见 [设置健康检查器](#)。
  - 存活探针: 监测到容器实例不健康时, 重启应用。
  - 就绪探针: 监测到容器实例不健康时, 将工作负载设置为未就绪状态, 业务流量不会导入到该容器中。
- **命令**: 可自定义容器的启动命令, Kubernetes 的容器启动命令可参见 [Kubernetes 官方文档](#)。
- **参数**: 可自定义容器的启动参数, Kubernetes 的容器启动的参数可参见 [Kubernetes 官方文档](#)。
- **端口**: 即容器端口, 用于指定容器需要暴露的端口, 端口协议可以选择 TCP 和 UDP。
- **环境变量**: 环境变量是指容器运行环境中设定的一个变量, 与 Dockerfile 中的“ENV”效果相同, 为创建的工作负载提供极大的灵活性。
- **引入配置中心**: 支持添加 Secret 和 ConfigMap 作为环境变量, 用来保存键值对形式的配置数据, 详见 [配置](#) 和 [密钥](#)。
- **镜像拉取策略**: imagePullPolicy, 默认的镜像拉取策略是 IfNotPresent, 在镜像已经存在的情况下, kubelet 将不再去拉取镜像。如果需要频繁拉取镜像, 则设置拉取策略为 Always。如果容器属性 imagePullPolicy 设置为 IfNotPresent 或者 Never, 则会优先使用本地镜像。

设置完成后点击 **保存**。



## 更新策略

更新策略包括滚动更新 (RollingUpdate) 和替换升级 (Recreate):

- 滚动更新: 推荐使用滚动更新 ([Rolling-update](#)) 的方式更新 Deployment, 滚动升级将逐步用新版的容器组替换旧版本的容器组, 升级的过程中, 业务流量会同时负载均衡分布到新老的容器组上, 所以业务不会中断。您可以指定 **容器组最小可用数量** 和 **更新时容器组最大数量** 来控制滚动更新的进程。
  - 容器组最小可用数量: 可选配置项, 每次滚动升级要求存活的最小容器组数量, 建议配置为正整数, 最小为 1, 该值可以是一个绝对值 (例如 5)。
  - 更新时容器组最大数量: 可选配置项, 升级过程中, Deployment 中允许超出副本数量的容器组的最大数量。
- 替换升级: 在创建出新的 Pod 之前会先杀掉所有已存在的 Pod, 意味着替换升级会先删除旧的容器组, 再创建新容器组, 升级过程中业务会中断。

上述配置信息填写完成以后, 点击 **下一步**。

## 第三步：添加存储卷

在存储卷页面可以添加 **持久化存储卷**、**临时存储卷** 和 **引用配置中心**。

### 持久化存储卷

持久化存储卷可用于持久化存储用户数据，需要预先创建存储卷，参考 [存储卷 – 创建存储卷](#)。

### 临时存储卷

临时存储卷是 `emptyDir` 类型，随 Pod 被分配在主机上。当 Pod 从主机上被删除时，临时存储卷也同时会删除，存储卷的数据也将永久删除，容器崩溃不会从节点中移除 Pod，因此 `emptyDir` 类型的卷中数据在容器崩溃时是安全的。

### 引入配置中心

支持配置 ConfigMap 或 Secret 中的值添加为卷，支持选择要使用的密钥以及将公开每个密钥的文件路径，最后设置目录在容器中的挂载路径。

其中 Secret 卷用于将敏感信息（如密码）传递到 Pod。Secret 卷由 tmpfs（一个 RAM 支持的文件系统）支持，所以它们永远不会写入非易失性存储器。

ConfigMap 用来保存键值对形式的配置数据，这个数据可以在 Pod 里使用，或者被用来为像 Controller 一样的系统组件存储配置数据。虽然 ConfigMap 跟 Secret 类似，但是 ConfigMap 更方便的处理不含敏感信息的字符串。它很像 Linux 中的 /etc 目录，专门用来存储配置文件的目录。ConfigMaps 常用于以下场景：

- 设置环境变量的值
- 在容器里设置命令行参数
- 在数据卷里面创建 config 文件

重要提示：您必须先在配置中心创建 Secret 或 ConfigMap，然后才能使用它，详见 [创建 Secret](#) 和 [创建 ConfigMap](#)。

The screenshot shows the 'Create Deployment' wizard with the 'Storage Volume Settings' step selected. At the top, there are tabs for '基本信息' (Basic Information), '容器组模板' (Container Group Template), '存储卷设置' (Storage Volume Settings) (which is active and highlighted in green), '标签设置' (Label Settings), and '节点选择器' (Node Selector). A 'Edit Mode' switch is also present. Below the tabs, the title '存储卷设置' is displayed with a note: '您可以根据需要选择适合您的存储卷类型进行添加' (You can choose the appropriate storage volume type according to your needs). Three buttons are available: '添加已有存储卷' (Add Existing Storage Volume), '添加临时存储卷' (Add Temporary Storage Volume), and '引用配置中心' (Reference Configuration Center).

## 第四步：添加标签

标签设置页用于指定资源对应的一组或者多组标签 (Label)。Label 以键值对的形式附加到任何对象上，如 Pod, Service, Node 等，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod (或其他对象) 定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用 (日志记录，监控，报警等)。通过多个标签的设置，我们就可以多维度地对对象进行精细化管理，如 `release: stable ; tier: frontend`。

The screenshot shows the 'Create Deployment' wizard with the 'Label Settings' step selected. The interface is similar to the previous one, with tabs for '基本信息', '容器组模板', '存储卷设置', '标签设置' (active), and '节点选择器'. The title '标签设置' is shown with a note: '标签是一个或多个关联到资源如容器组上的键值对，我们通常通过标签来识别、组织或查找资源对象' (A label is a key-value pair associated with a resource such as a container group, we usually identify, organize or find resources through labels). Below this, there is a table-like structure with two rows: 'tier' and 'frontend'. Each row has a delete icon and a '添加' (Add) button.

## 第五步：添加节点选择器

用户可以通过按节点选择或通过 Selector 设置一组或者多组键值对来指定期望运行容器组的主机。当不指定时，容器组将有可能调度到集群内满足调度条件的任意节点。最后点击创建，集群就会按照用户的配置创建部署。

创建部署

编辑模式

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

### 节点选择

通过使用选择器将容器组调度到期望运行的节点上，这些选择器是一组或多组键值对匹配节点标签。

指定节点 ▾ 可以通过节点 IP 或者节点名称查找

指定节点 节点选择器 20 取消选择

i-gddrbxs 污点: 选择

主机 IP: -

点击创建，即可完成部署资源的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

## 有状态副本集

有状态副本集 (StatefulSet)，是为了解决有状态服务的问题，在运行过程中会保存数据或状态，例如 Mysql，它需要存储产生的新数据。而 Deployments 是为无状态服务而设计。应用场景包括：

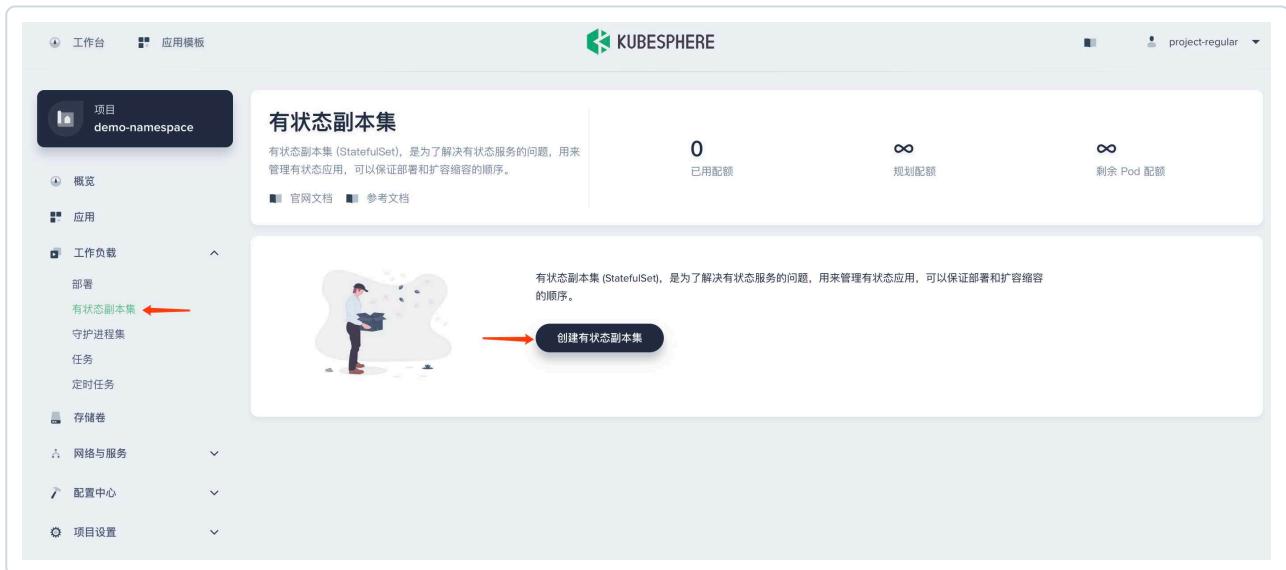
- 稳定的持久化存储，即 Pod 重新调度后还是能访问到相同的持久化数据，基于 PVC 来实现。
- 稳定的网络标志，即 Pod 重新调度后其 PodName 和 HostName 不变，基于 Headless Service (即没有 Cluster IP 的 Service) 来实现。
- 有序部署、有序扩展，即 Pod 是有序的，在部署或者扩展的时候要依据定义的顺序依次进行 (即从 0 到 N-1，在下一个 Pod 运行之前所有之前的 Pod 必须都是 Running 和 Ready 状态)，基于 init containers 来实现。
- 有序收缩、有序删除 (即从 N-1 到 0)。

本文档仅说明创建有状态副本集中可能用到的参数或字段意义，创建工作负载后应如何管理，请参考 [工作负载管理](#)。同时，[部署 MySQL 有状态应用示例](#) 也可帮助您快速理解 StatefulSet。

## 创建有状态副本集

登录 KubeSphere 控制台，在已创建的项目下选择 [工作负载 → 有状态副本集](#)，进入列表页。

左上角为当前所在项目，如果是管理员登录，可以看到集群所有项目的有状态副本集情况，如果是普通用户，则只能查看授权项目下的所有有状态副本集。列表顶部显示了当前项目的有状态副本集 Pod 配额和数量信息。



## 第一步：填写基本信息

1.1. 点击 **创建** 按钮，将弹出创建部署的详情页。创建有状态副本集支持三种方式，**页面创建**，**导入 yaml 文件 创建**，**编辑模式 创建**。以下主要介绍页面创建的方式。若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式。左上角显示配置文件列表和导入导出按钮。其中导入 yaml 文件方式会自动将 yaml 文件内容填充到页面上，用户根据需要可以在页面上调整后再行创建。编辑模式可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件并创建有状态副本集。

创建有状态副本集

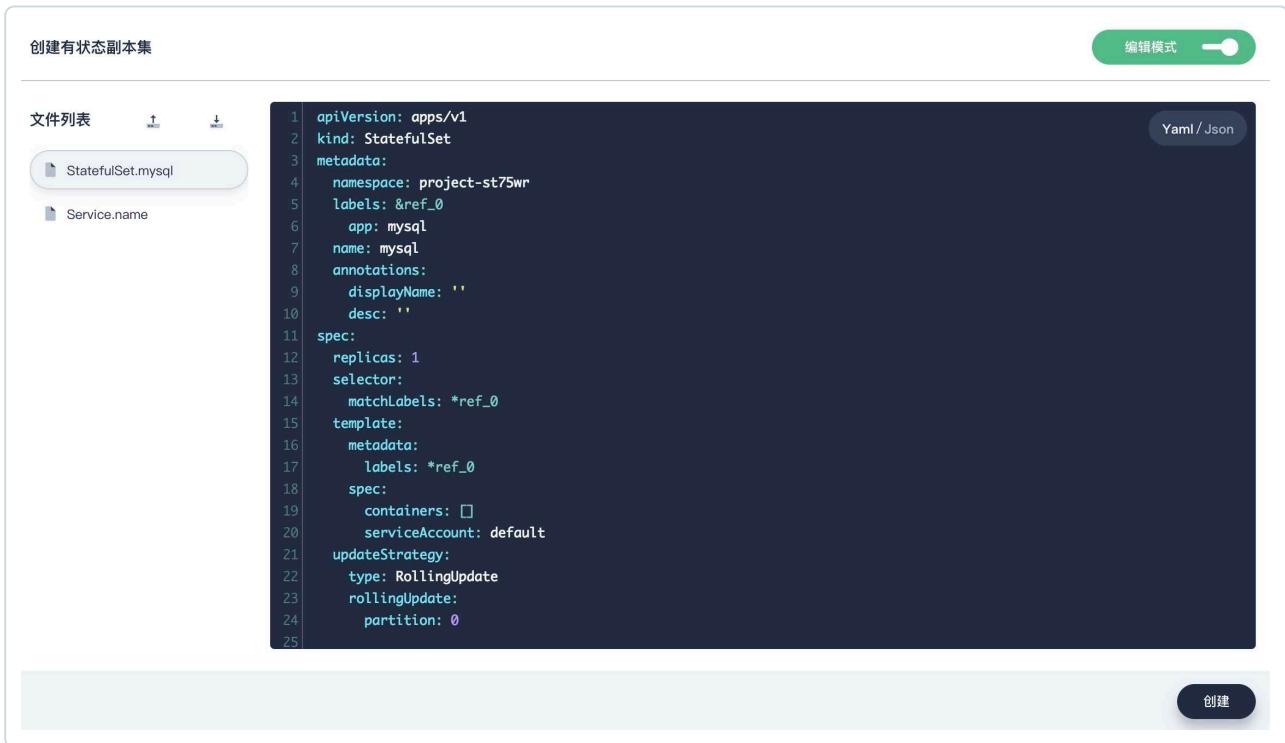
编辑模式

文件列表

Yaml / Json

```
1 apiVersion: apps/v1
2 kind: StatefulSet
3 metadata:
4   namespace: project-st75wr
5   labels: &ref_0
6     app: mysql
7   name: mysql
8   annotations:
9     displayName: ''
10    desc: ''
11 spec:
12   replicas: 1
13   selector:
14     matchLabels: *ref_0
15   template:
16     metadata:
17       labels: *ref_0
18     spec:
19       containers: []
20         serviceAccount: default
21   updateStrategy:
22     type: RollingUpdate
23     rollingUpdate:
24       partition: 0
25
```

创建



1.2. 在基本信息页，需要输入部署的名称并选择创建部署的项目，用户可以根据需求填写部署的描述信息。

- 名称：为创建的有状态副本集起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：简单介绍该有状态副本集，让用户进一步了解其作用。

点击 **下一步**。

创建有状态副本集

编辑模式

基本信息  容器组模板  存储卷模板  服务配置  标签设置  节点选择器

### 基本信息

您可以给有状态副本集起一个名字，以便在使用的时候容易区分。

名称 **\***  
  
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目  
  
将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名  
  
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息



## 第二步：配置容器组模板

2.1. 在 **容器组模板** 页面, 点击 **添加容器**, 根据需求添加容器镜像, 输入容器的名称和对应的镜像名, 镜像名一般需要指定 tag, 比如 mysql:5.6。容器中定义的镜像默认从 Docker Hub 中拉取。

**说明：**若需要使用私有镜像仓库如 Harbor, 参见 [镜像仓库 – 添加镜像仓库](#)。

为了实现集群的资源被有效调度和分配同时提高资源的利用率, 平台采用了 request 和 limit 两种限制类型对资源进行分配。request 通常是容器使用的最小资源需求, 而 limit 通常是容器能使用资源的最大值, 设置为 0 表示对使用的资源不做限制, 可无限的使用。request 能保证 pod 有足够的资源来运行, 而 limit 则是防止某个 Pod 无限制的使用资源, 导致其他 Pod 崩溃。

表1：CPU 配额说明

参数	说明
<b>最小使用 (requests)</b>	容器使用的 CPU 最小值, 作为容器调度时资源分配的判断依赖。 只有当节点上可分配 CPU 总量 $\geq$ 容器 CPU 最小值时, 才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的 CPU 最大值。

表2：内存配额说明

参数	说明
<b>最小使用 (requests)</b>	容器使用的最小内存需求, 作为容器调度时资源分配的判断依赖。 只有当节点上可分配内存总量 $\geq$ 容器内存申请数时, 才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的内存最大值, 如果内存使用量超过这个限定值, 容器可能会被 kill。

创建有状态副本集

基本信息  容器组模板  存储卷模板  服务配置  标签设置  节点选择器

容器名称 \* mysql 镜像 \* mysql:5.6

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾。

要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU

最小使用	10	m	最大使用	100	m
------	----	---	------	-----	---

作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则：1核 = 1000m

内存

最小使用	10	Mi	最大使用	200	Mi
------	----	----	------	-----	----

作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项 ^

就绪探针

存活探针

添加探针

2.2. 如果用户有更进一步的需求，可以点击 **高级选项**。

- **就绪探针/存活探针**：在业务级的监控检查方面，Kubernetes 定义了两种类型的健康检查探针，详见 [设置健康检查器](#)。
  - 存活探针：监测到容器实例不健康时，重启应用。
  - 就绪探针：监测到容器实例不健康时，将工作负载设置为未就绪状态，业务流量不会导入到该容器中。
- **命令**：可自定义容器的启动命令，Kubernetes 的容器启动命令可参见 [Kubernetes 官方文档](#)。
- **参数**：可自定义容器的启动参数，Kubernetes 的容器启动的参数可参见 [Kubernetes 官方文档](#)。
- **端口**：即容器端口，用于指定容器需要暴露的端口，端口协议可以选择 TCP 和 UDP。
- **环境变量**：环境变量是指容器运行环境中设定的一个变量，与 Dockerfile 中的“ENV”效果相同，为创建工作负载提供极大的灵活性。
- **引入配置中心**：支持添加 Secret 和 ConfigMap 作为环境变量，用来保存键值对形式的配置数据，详见 [配置](#) 和 [密钥](#)。
- **镜像拉取策略**：imagePullPolicy，默认的镜像拉取策略是 IfNotPresent，在镜像已经存在的情况下

下, kubelet 将不再去拉取镜像。如果需要频繁拉取镜像, 则设置拉取策略为 Always。如果容器属性 imagePullPolicy 设置为 IfNotPresent 或者 Never, 则会优先使用本地镜像。

容器组的镜像设置完成后, 点击 **保存**。

The screenshot shows the 'Create StatefulSet' configuration page. At the top, there are tabs for '基本信息' (Basic Information), '容器组模板' (Container Group Template), '存储卷模板' (Storage Volume Template), '服务配置' (Service Configuration), '标签设置' (Label Settings), and '节点选择器' (Node Selector). The '基本信息' tab is selected. Below the tabs, there are sections for '命令' (Commands), '参数' (Parameters), '端口' (Ports), '环境变量' (Environment Variables), and '镜像拉取策略' (Image Pull Policy). The '命令' section has a '命令' input field and a '添加命令' button. The '参数' section has a '参数' input field and a '添加参数' button. The '端口' section has 'port' (port), 'TCP' (protocol), and '3306' (port number) inputs, along with a '添加端口' button. The '环境变量' section has '名称' (Name), '选择资源' (Select Resource), and '选择键' (Select Key) dropdowns, with '添加环境变量' (Add Environment Variable) and '引用配置中心' (Reference Configuration Center) buttons. The '镜像拉取策略' section has a dropdown set to 'IfNotPresent'. A '编辑模式' switch is at the top right.

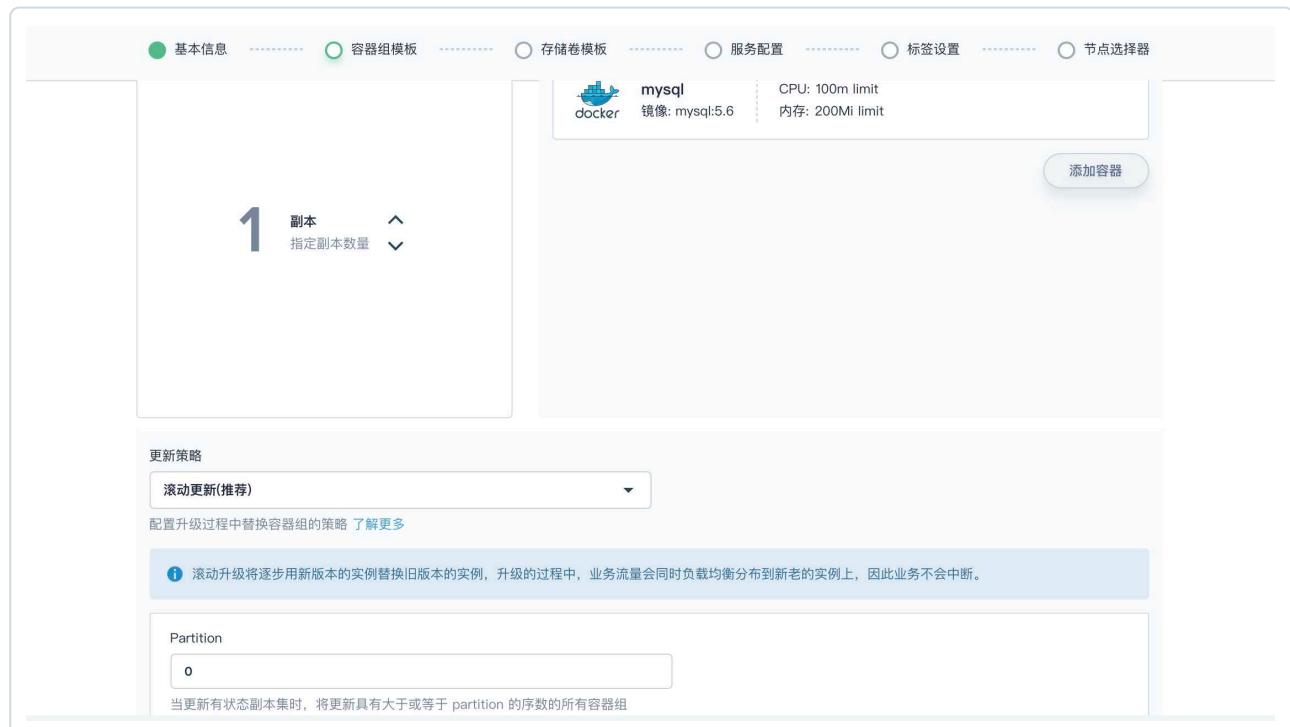
## 更新策略

更新策略是指定新的 Pod 替换旧的 Pod 的策略, 有状态副本集的更新策略分为 **滚动更新 (RollingUpdate)** 和 **删除容器组时更新 (OnDelete)** 两种类型:

- 滚动更新 (RollingUpdate): 推荐使用该策略, 有状态副本集中实现 Pod 的自动滚动更新。当更新策略设置为滚动更新时, 有状态副本集控制器将在有状态副本集中删除并重新创建每个 Pod。它将以与 Pod 终止相同的顺序进行 (从最大的序数到最小的序数), 每次更新一个 Pod。在更新其前身之前, 它将等待正在更新的 Pod 状态变成正在运行并就绪。
  - Partition: 通过指定 Partition 来对滚动更新策略进行分区。如果指定了分区, 则当 StatefulSet 的 template 更新时, 具有大于或等于分区序数的所有 Pod 将被更新。具有小于分区的序数的所有 Pod 将不会被更新, 即使删除它们也将被重新创建。如果 Partition 大于其副本数, 则其 template 的更新将不会传播到 Pod。在大多数情况下, 您不需要使用分区, 只有需要进行分阶段更新时才会使用到。

- **删除容器组时更新 (OnDelete):** 设置为 OnDelete 时, StatefulSet 控制器将不会自动更新 StatefulSet 中的 Pod。用户必须手动删除旧版本 Pod 以触发控制器创建新的 Pod。

上述配置信息填写完成以后, 点击 **下一步**。



### 第三步：添加存储卷

点击 **添加存储卷模板**, 填写存储卷的名称, 选择存储类型, 存储类型需要预先创建, 参考 [存储类型 - 创建存储类型](#)。然后指定卷的容量和访问模式, 确定存储卷在容器内的挂载路径, 详见 [存储卷](#)。

创建有状态副本集

编辑模式

基本信息 容器组模板 存储卷模板 服务配置 标签设置 节点选择器

存储卷名称 \*  
mysql  
最长253个字符，只能包含小写字母、数字及分隔符“-”，且必须以小写字母或数字开头及结尾

描述信息

存储类型 csi-qingcloud

容量 10 Gi

访问模式 ReadWriteOnce(RWO)  
单个节点读写

挂载路径

container-2hl3gg 读写 /var/lib/mysql

上一步 保存

## 第四步：服务设置

由于有状态副本集必须包含一个 Headless 服务，因此需创建服务。填写服务名称，服务端口和目标端口，目标端口对应容器对外暴露的端口。基于客户端 IP 地址进行会话保持的模式，即第一次客户端访问后端某个 Pod，之后的请求都转发到这个 Pod 上。若要实现基于客户端 IP 的会话亲和性，可以将会话亲和性的值设置为 "ClientIP" (默认值为 "None")，该设置可将来自同一个 IP 地址的访问请求都转发到同一个后端 Pod。

The screenshot shows the 'Service Configuration' step of a 'Create StatefulSet' wizard. At the top, there are tabs for '基本信息' (Basic Information), '容器组模板' (Container Group Template), '存储卷模板' (Storage Volume Template), '服务配置' (Service Configuration) (which is selected and highlighted in green), '标签设置' (Label Settings), and '节点选择器' (Node Selector). A '编辑模式' (Edit Mode) switch is at the top right. The main area is titled '服务配置' (Service Configuration) and contains a note: '集群不为服务生成 IP, 集群内部通过服务的后端 Endpoint IP 直接访问服务。此类型适合后端异构的服务, 比如需要区分主从的服务。'. It includes fields for '服务名称' (Service Name) set to 'service', '会话亲和性' (Session Affinity) set to 'None', and a '端口' (Port) section where 'port' is '3306' and '协议' (Protocol) is 'TCP'. A '添加端口' (Add Port) button is also present.

## 第五步：添加标签

标签设置页用于指定资源对应的一组或者多组标签 (Label)。Label 以键值对的形式附加到任何对象上，如 Pod, Service, Node 等，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod (或其他对象) 定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用 (日志记录、监控、报警等)。通过多个标签的设置，我们就可以多维度地对对象进行精细化管理，如 `relase: stable ; tier: frontend`。

The screenshot shows the 'Label Settings' step of the 'Create StatefulSet' wizard. The tabs at the top are the same as the previous screen. The main area is titled '标签设置' (Label Settings) and contains a note: '标签是一个或多个关联到资源如容器组上的键值对，我们通常通过标签来识别、组织或查找资源对象' (Labels are key-value pairs associated with resources like container groups, we usually identify, organize or find resources through labels). It features two input fields for labels: 'app' and 'mysql', each with a trash icon. A '添加' (Add) button is located below the inputs.

## 第六步：添加节点选择器

带有标签的对象创建好之后，我们就可以通过节点选择器 (Selector) 来引用这些对象。节点选择器页

面，用户可以通过按节点选择或通过设置一组或者多组键值对来指定期望运行容器组的主机。当不指定时，将会在集群内的所有节点上启动容器组。点击右下角的创建后，集群就会按照用户的配置创建对应的守护进程集。

The screenshot shows the 'Create StatefulSet' wizard in KubeSphere. The current step is 'Node Selection'. At the top, there are tabs for '基本信息' (Basic Information), '容器组模板' (Container Group Template), '存储卷模板' (Storage Volume Template), '服务配置' (Service Configuration), '标签设置' (Label Settings), and '节点选择器' (Node Selector). A 'Edit Mode' switch is also present. Below the tabs, the title '节点选择' (Node Selection) is displayed, followed by a subtitle: '通过使用选择器将容器组调度到期望运行的节点上，这些选择器是一组或多组键值对匹配节点标签。' (By using selectors to schedule the container group to the nodes where it is expected to run, these selectors are a group or multiple key-value pairs to match node labels.) There are two sections for selecting nodes: '指定节点' (Specify Node) and '节点选择器' (Node Selector). The '节点选择器' section is active, showing a search bar and a table with one row: 'i-gddrhbx' (IP: -) with a '污点' (Taint) column and a '选择' (Select) button.

点击创建，即可完成有状态副本集的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

# 守护进程集

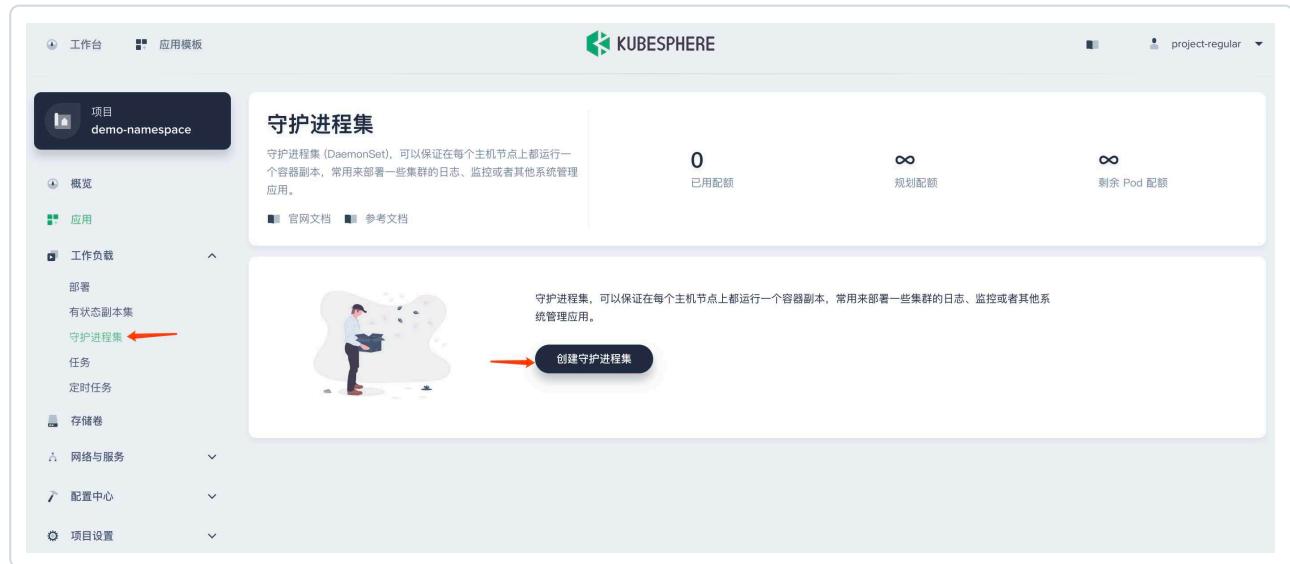
守护进程集 (DaemonSet)，保证在每个 Node 上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。典型的应用场景包括：

- 日志收集，比如 Fluentd, Logstash 等。
- 系统监控，比如 Prometheus Node Exporter, collectd, New Relic agent, Ganglia gmond 等。
- 系统程序，比如 kube-proxy, kube-dns, Gluster, Ceph 等。

## 创建守护进程集

登录 KubeSphere 控制台，在已创建的项目中选择 **工作负载 → 守护进程集**，进入守护进程集列表页面。

左上角为当前所在项目，点击下拉框可以切换到其他的项目。如果是管理员登录，可以看到集群所有项目的守护进程集情况，如果是普通用户，则只能查看授权项目下的所有守护进程集。列表顶部显示了当前项目的守护进程集 Pod 配额和数量信息。



## 第一步：填写基本信息

1.1. 点击 **创建守护进程集** 按钮，将弹出创建守护进程集的详情页。创建守护进程集支持三种方式，**页面创建**，**导入 yaml 文件** 创建，**编辑模式** 创建。以下主要介绍页面创建的方式，若选择以编辑模式，可点

点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式。左上角显示配置文件列表和导入导出按钮。其中导入 yaml 文件方式会自动将 yaml 文件内容填充到页面上，用户根据需要可以在页面上调整后再行创建。编辑模式可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件并创建守护进程集。

The screenshot shows a modal window titled "创建守护进程集" (Create DaemonSet). At the top right is a green button labeled "编辑模式" (Edit Mode) with a switch icon. Below it is a "Yaml / Json" button. The main area contains a code editor with the following YAML configuration:

```
1 apiVersion: apps/v1
2 kind: DaemonSet
3 metadata:
4   namespace: project-st75wr
5   labels: &ref_0
6     app: node-exporter
7   name: node-exporter
8   annotations:
9     displayName: ''
10    desc: This is a demo
11 spec:
12   replicas: 1
13   selector:
14     matchLabels: *ref_0
15   template:
16     metadata:
17       labels: *ref_0
18     spec:
19       containers: []
20       serviceAccount: default
21   updateStrategy:
22     type: RollingUpdate
23     rollingUpdate:
24       maxUnavailable: 1
25       minReadySeconds: 0
```

At the bottom right of the modal is a dark blue "创建" (Create) button.

1.2. 在基本信息页，需要输入守护进程集的名称，用户可以根据需求填写守护进程集的描述信息，完成后点击 **下一步**。

- 名称：为创建的守护进程集起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：简单介绍守护进程集，让用户进一步了解其作用。

创建守护进程集

基本信息      容器组模板      存储卷设置      标签设置      节点选择器

### 基本信息

守护进程集保证在每个主机上都运行一个容器副本，常用来部署一些集群的日志、监控或者其他系统管理应用。

名称 *	node-exporter
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾	
别名	
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。	
项目	demo-namespace
将根据项目进行资源进行分组，可以按项目对资源进行查看管理	
描述信息	This is a demo

## 第二步：配置容器组模板

2.1. 在 **容器组模板** 页面，点击 **添加容器**，根据需求添加容器镜像，输入容器的名称和对应的镜像名，镜像名一般需要指定 tag，比如 node-exporter:v0.15.2，容器中定义的镜像默认从 Docker Hub 中拉取。

**说明：**若需要使用私有镜像仓库如 Harbor，参见 [镜像仓库 – 添加镜像仓库](#)。

为了实现集群的资源被有效调度和分配同时提高资源的利用率，平台采用了 request 和 limit 两种限制类型对资源进行分配。request 通常是容器使用的最小资源需求，而 limit 通常是容器能使用资源的最大值，设置为 0 表示对使用的资源不做限制，可无限的使用。request 能保证 pod 有足够的资源来运行，而 limit 则是防止某个 Pod 无限制的使用资源，导致其他 Pod 崩溃。

表1：CPU 配额说明

参数	说明
<b>最小使用 (requests)</b>	容器使用的 CPU 最小值，作为容器调度时资源分配的判断依赖。 只有当节点上可分配 CPU 总量 $\geq$ 容器 CPU 最小值时，才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的 CPU 最大值。

表2：内存配额说明

参数	说明
<b>最小使用 (requests)</b>	容器使用的最小内存需求，作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量 $\geq$ 容器内存申请数时，才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的内存最大值，如果内存使用量超过这个限定值，容器可能会被 kill。

创建守护进程集

编辑模式

基本信息  容器组模板  存储卷设置  标签设置  节点选择器

容器名称 \* node-exporter 镜像 \* node-exporter:v0.15.2  
最长 253 个字符，只能包含小写字母、数字及分隔符“-”，且必须以小写字母或数字开头及结尾。  
要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU  
最小使用 10 m 最大使用 100 m  
作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m

内存  
最小使用 10 Mi 最大使用 200 Mi  
作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项 ▾

就绪探针  
添加探针

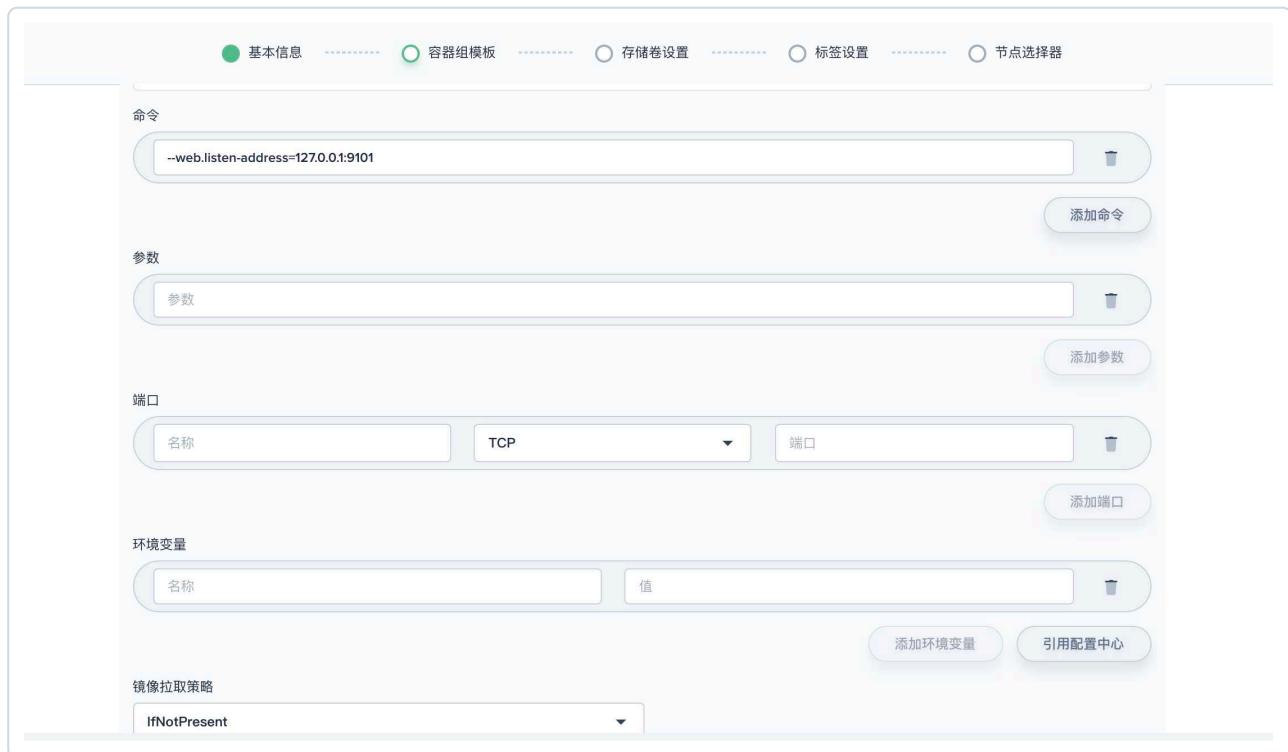
存活探针  
添加探针

2.2. 如果用户有更进一步的需求，可以点击 **高级选项**。

- 就绪探针/存活探针：**在业务级的监控检查方面，Kubernetes 定义了两种类型的健康检查探针，详见 [设置健康检查器](#)。
  - 存活探针：监测到容器实例不健康时，重启应用。
  - 就绪探针：监测到容器实例不健康时，将工作负载设置为未就绪状态，业务流量不会导入到该容器中。
- 命令：**可自定义容器的启动命令，Kubernetes 的容器启动命令可参见 [Kubernetes 官方文档](#)。
- 参数：**可自定义容器的启动参数，Kubernetes 的容器启动的参数可参见 [Kubernetes 官方文档](#)。

- **端口**: 即容器端口，用于指定容器需要暴露的端口，端口协议可以选择 TCP 和 UDP。
- **环境变量**: 环境变量是指容器运行环境中设定的一个变量，与 Dockerfile 中的“ENV”效果相同，为创建的工作负载提供极大的灵活性。
- **引入配置中心**: 支持添加 Secret 和 ConfigMap 作为环境变量，用来保存键值对形式的配置数据，详见 [配置](#) 和 [密钥](#)。
- **镜像拉取策略**: imagePullPolicy，默认的镜像拉取策略是 IfNotPresent，在镜像已经存在的情况下，kubelet 将不再去拉取镜像。如果需要频繁拉取镜像，则设置拉取策略为 Always。如果容器属性 imagePullPolicy 设置为 IfNotPresent 或者 Never，则会优先使用本地镜像。

容器组的镜像设置完成后，点击 **保存**。



## 更新策略

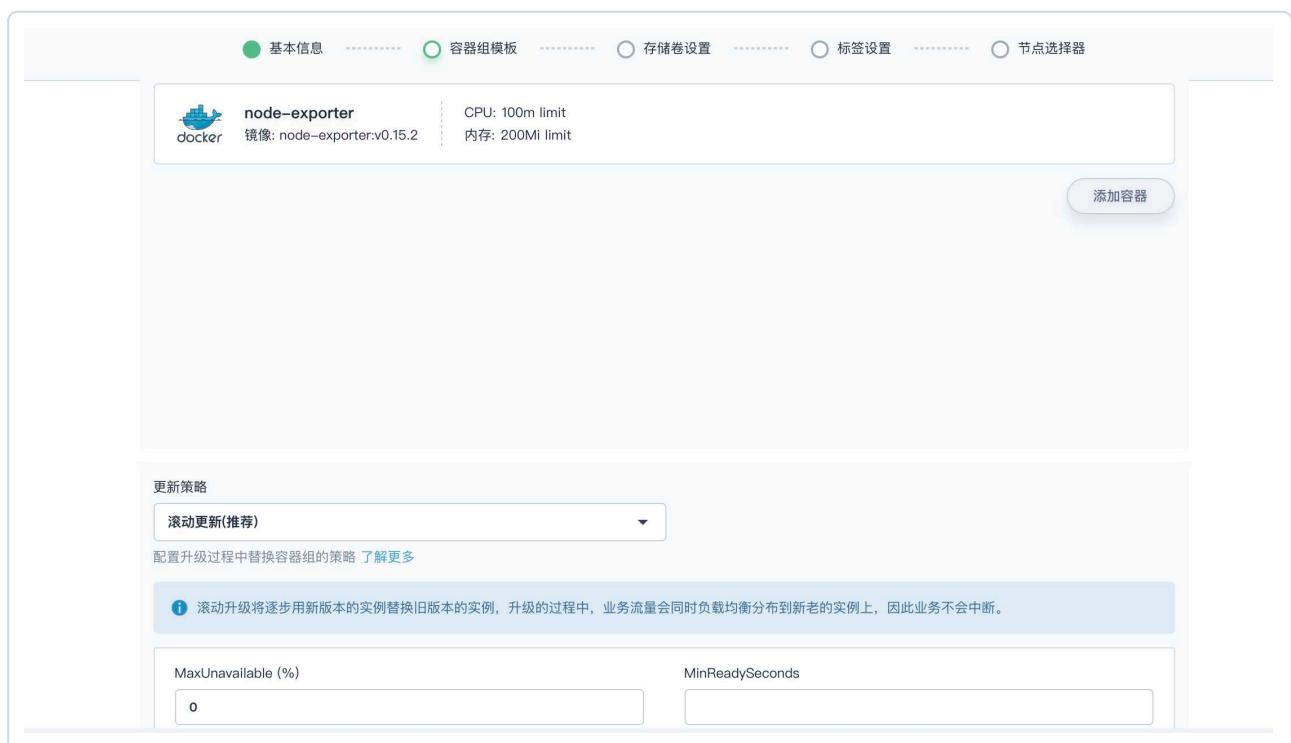
在 Kubernetes 中策略是指定新的 Pod 替换旧的 Pod 的策略，守护进程集的更新策略分为 [滚动更新 \(RollingUpdate\)](#) 和 [删除容器组时更新 \(OnDelete\)](#) 两种类型：

- 滚动更新：推荐使用滚动更新 ([Rolling-update](#)) 的方式更新 Pod。您可以指定 maxUnavailable 和 MinReadySeconds 来控制滚动更新的进程。
  - MaxUnavailable 是可选配置项，当前默认值是 1，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。

- MinReadySeconds 是一个可选配置项，默认是 0（Pod 在 ready 后就会被认为是可用状态），用来指定没有任何容器 crash 的 Pod 并被认为是可用状态的最小秒数。进一步了解什么情况下 Pod 会被认为是 ready 状态，请参阅 [Kubernetes 官方文档 – Container Probes](#)。
- 删除容器组时更新：设置为删除容器组时更新（OnDelete）时，StatefulSet 控制器将不会自动更新 StatefulSet 中的 Pod，用户必须手动删除旧版本 Pod 以触发控制器创建新的 Pod。

上述配置信息填写完成以后，点击 **下一步**。



### 第三步：添加存储卷

在存储卷页面可以添加 **已有持久化存储卷**、**添加临时存储卷**、**HostPath**、**引用配置中心**。

#### 持久化存储卷

持久化存储卷可用于持久化存储用户数据，需要预先创建存储卷，参考 [存储卷 – 创建存储卷](#)。

## 临时存储卷

临时存储卷是 [emptyDir](#) 类型，随 Pod 被分配在主机上。当 Pod 从主机上被删除时，临时存储卷也同时会删除，存储卷的数据也将永久删除，容器崩溃不会从节点中移除 Pod，因此 emptyDir 类型的卷中数据在容器崩溃时是安全的。

## 引用配置中心

支持配置 ConfigMap 或 Secret 中的值添加为卷，支持选择要使用的密钥以及将公开每个密钥的文件路径，最后设置目录在容器中的挂载路径。

其中，Secret 卷用于将敏感信息（如密码）传递到 pod。Secret 卷由 tmpfs（一个 RAM 支持的文件系统）支持，所以它们永远不会写入非易失性存储器。

ConfigMap 用来保存键值对形式的配置数据，这个数据可以在 Pod 里使用，或者被用来为像 Controller 一样的系统组件存储配置数据。虽然 ConfigMap 跟 Secret 类似，但是 ConfigMap 更方便的处理不含敏感信息的字符串。它很像 Linux 中的 /etc 目录，专门用来存储配置文件的目录。ConfigMaps 常用于以下场景：

- 设置环境变量的值
- 在容器里设置命令行参数
- 在数据卷里面创建 config 文件

重要提示：您必须先在配置中心创建 Secret 或 ConfigMap，然后才能使用它，详见 [创建 Secret](#) 和 [创建 ConfigMap](#)。

## HostPath

HostPath 允许将宿主机上的指定卷加载到容器之中。这种卷一般和 DaemonSets 搭配使用，用来操作主机文件，例如进行日志采集中 EFK 中的 [FluentD](#) 就采用这种方式，加载主机的容器日志目录，达到收集本主机所有日志的目的。

创建守护进程集

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

存储卷设置

您可以根据需要选择适合您的存储卷类型进行添加

添加已有存储卷 添加临时存储卷 添加 Host Path 引用配置中心

## 第四步：添加标签

标签设置页用于指定资源对应的一组或者多组标签 (Label)。Label 以键值对的形式附加到任何对象上，如 Pod, Service, Node 等，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod (或其他对象) 定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用（日志记录、监控、报警等）。通过多个标签的设置，我们就可以多维度地对对象进行精细化管理，如 `release: stable ; tier: frontend`。

创建守护进程集

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

标签设置

标签是一个或多个关联到资源如容器组上的键值对，我们通常通过标签来识别、组织或查找资源对象

app	node-exporter	删除
-----	---------------	----

添加

## 第五步：添加节点选择器

带有标签的对象创建好之后，我们就可以通过节点选择器 (Selector) 来引用这些对象。节点选择器页面，用户可以通过按节点选择或通过 Selector 设置一组或者多组键值对来指定期望运行容器组的主机。当不指定时，将会在集群内的所有节点上运行容器组。点击右下角的创建后，集群就会按照用户的配置

创建对应的守护进程集。

The screenshot shows the 'Create Pod Disruption Budget' interface. At the top, there are tabs for '基本信息' (Basic Information), '容器组模板' (Container Group Template), '存储卷设置' (Storage Volume Settings), '标签设置' (Label Settings), and '节点选择器' (Node Selector). A '编辑模式' (Edit Mode) switch is also present. The current step is '节点选择' (Node Selection), which is described as selecting nodes where the container group will run. It includes a search bar for node IP or name and a dropdown for '指定节点' (Specify Node). Below this, two nodes are listed: 'i-gddrhhbx' and 'i-wo83nj02'. Each node entry shows its name, host IP (both listed as '-'), and a '污点' (Taint) field. Buttons for '取消选择' (Cancel Selection), '选择' (Select), and '创建' (Create) are available. Navigation buttons '上一步' (Previous Step) and '下一步' (Next Step) are at the bottom.

点击创建，即可完成守护进程集的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

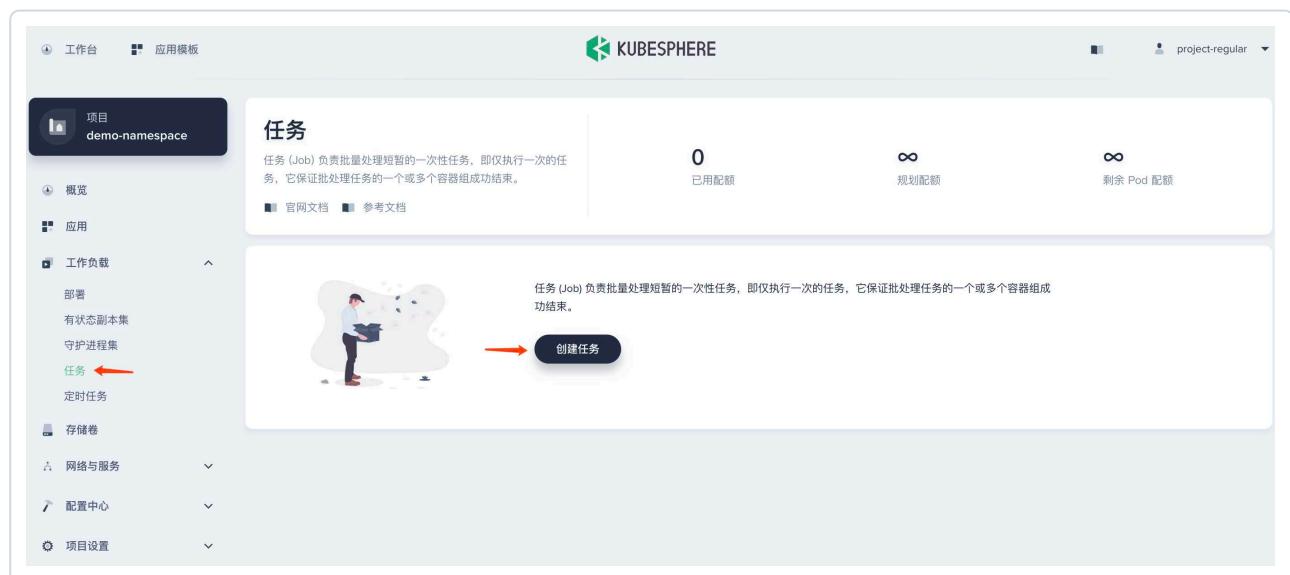
# 任务

任务 (Job)，在 Kubernetes 中用来控制批处理型任务的资源对象，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。任务管理的 Pod 根据用户的设置在任务成功完成就自动退出，比如在创建工作负载前，执行任务；将镜像上传至镜像仓库等一次性任务。

本文档主要解释说明创建任务中所有参数或字段的释义，配合 [快速入门 – 创建简单任务](#) 帮助您快速创建一个任务来执行简单的命令计算并输出圆周率到小数点后 2000 位作为示例，说明任务的基本功能。

## 创建任务

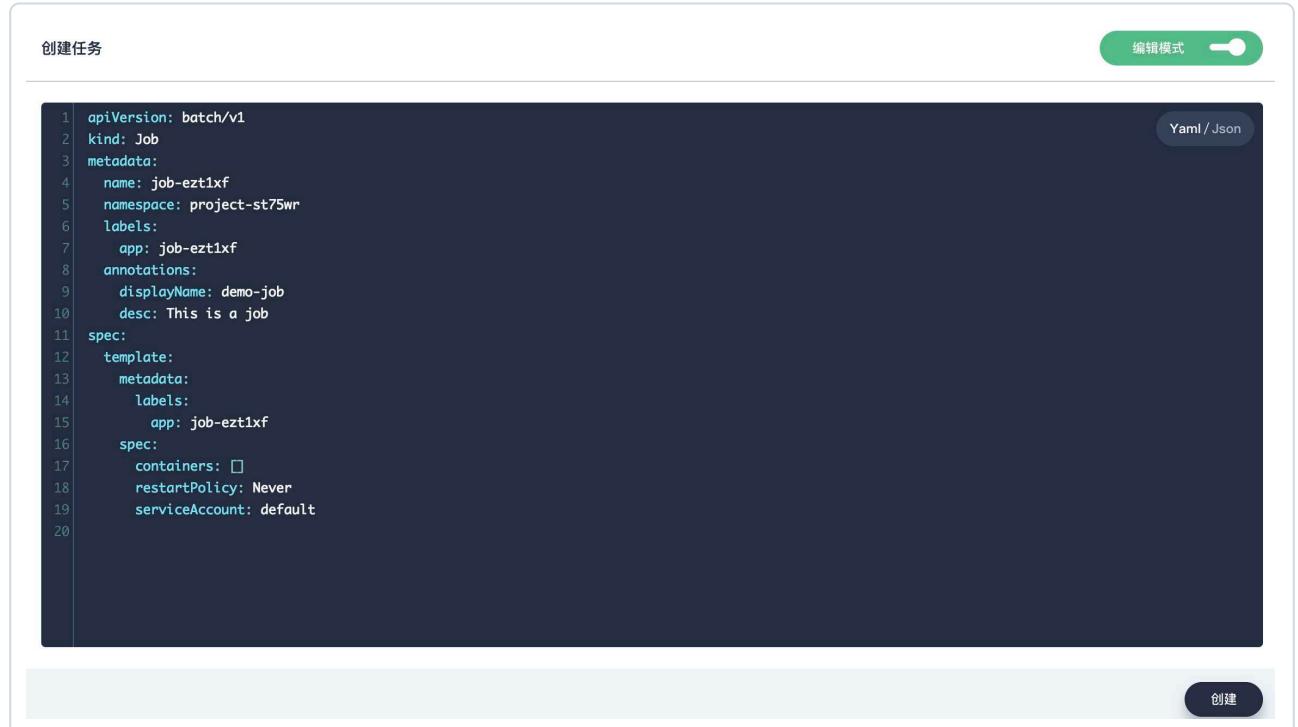
登录 KubeSphere 控制台，在已创建的项目下，进入 [工作负载 → 任务](#)，进入任务列表页面。左上角为当前所在项目。如果是管理员登录，可以看到集群所有项目的任务情况，如果是普通用户，则只能查看授权项目下的所有任务。列表顶部显示了当前项目的任务 Pod 配额和数量信息。



## 第一步：填写基本信息

1.1. 点击 **创建** 按钮，将弹出创建任务的详情页。创建任务支持三种方式，[页面创建](#)，[导入 yaml 文件](#)，[编辑模式](#)。以下主要介绍页面创建的方式，若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式。左上角显示配置文件列表和导入导出按钮。其中导入 yaml 文件方式会自动将 yaml 文件内容填充到页面上，用户根据需要可以在页面上调整后再行创建。编辑模式可以方便习惯命令

行操作的用户直接在页面上编辑 yaml 文件并创建任务。



The screenshot shows a 'Create Task' interface. At the top right is a green 'Edit mode' button with a key icon. Below it is a dark-themed code editor window containing a YAML configuration for a 'Job'. The code includes fields for metadata (name: job-ezt1xf, namespace: project-st75wr), labels (app: job-ezt1xf), annotations (displayName: demo-job, desc: This is a job), and a template specification with a container. A 'Yaml / Json' switch is located in the top right corner of the editor. At the bottom right of the editor is a 'Create' button.

```
1 apiVersion: batch/v1
2 kind: Job
3 metadata:
4   name: job-ezt1xf
5   namespace: project-st75wr
6   labels:
7     app: job-ezt1xf
8   annotations:
9     displayName: demo-job
10    desc: This is a job
11 spec:
12   template:
13     metadata:
14       labels:
15         app: job-ezt1xf
16     spec:
17       containers: []
18       restartPolicy: Never
19       serviceAccount: default
20
```

基本信息页中，需要填写任务的名称和描述信息。

- 名称：为创建的任务起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：简单介绍该任务有什么作用，让用户进一步了解该任务。



The screenshot shows the 'Create Task' interface with the 'Basic Information' tab selected. At the top, there is a navigation bar with tabs: 基本信息 (selected), 任务设置, 容器组模板, 存储卷设置, 标签设置, and 节点选择器. Below the tabs is a section titled '基本信息' (Basic Information). It contains two main input fields: '名称' (Name) with value 'job-demo' and '项目' (Project) with value 'demo-namespace'. There is also a note explaining that the project will be used for resource grouping and management. Below these, there are two more sections: '别名' (Alias) with value '任务示例' and '描述信息' (Description) with value 'This is a demo'. A note below the alias field states that aliases can be composed of any character and help distinguish resources.

## 第二步：任务设置

任务设置页中，通过设置 Job Spec 的四个配置参数来设置 Job 的任务类型。

- Back Off Limit: 失败尝试次数，若失败次数超过该值，则 Job 不会继续尝试工作；如设置为 5 则表示最多重试 5 次。
- Completions: 标志任务结束需要成功运行的 Pod 个数，如设置为 10 则表示任务结束需要运行 10 个 Pod。
- Parallelism: 标志并行运行的 Pod 的个数；如设置为 5 则表示并行 5 个 Pod。
- Active Deadline Seconds: Active Deadline Seconds: 指定 Job 可运行的时间期限，超过时间还未结束，系统将会尝试进行终止，且 ActiveDeadlineSeconds 优先级高于 Back Off Limit；如设置 20 则表示超过 20s 后 Job 运行将被终止。



创建任务

编辑模式

基本信息 任务设置 任务模板 存储卷设置 标签设置 节点选择器

任务设置

您可以在此配置任务(Job)的Job Spec格式，Job Controller负责根据Job Spec创建Pod，并持续监控Pod的状态，直至其成功结束。如果失败，则根据RestartPolicy（支持OnFailure和Never）决定是否创建新的Pod再次重试任务。

Back off Limit  
5  
失败尝试次数，若失败次数超过该值，则 job 不会继续尝试工作

Completions  
10  
标志 Job 结束需要成功运行的 Pod 个数

Parallelism  
5  
标志并行运行的 Pod 的个数

Active Deadline Seconds  
100  
job 运行的超时时间

上一步 下一步

## 第三步：配置任务模板

3.1. 任务模板即设置 Pod 模板，其中 RestartPolicy 指通过同一节点上的 kubelet 重新启动容器，仅支持 Never 或 OnFailure，当任务未完成的情况下：

- Never：任务会在容器组出现故障时创建新的容器组，且故障容器组不会消失，返回的字段“.status.failed”加1。
- OnFailure：任务会在容器组出现故障时其内部重启容器，而不是创建新的容器组，返回的字段“.status.failed”不变。

3.2. 点击 **添加容器**，然后根据需求添加容器镜像，容器中定义的镜像默认从 Docker Hub 中拉取。输入容器的名称和对应的镜像名，镜像名一般需要指定 tag，比如 perl:5.28.0。

**说明：**若需要使用私有镜像仓库如 Harbor，参见 [镜像仓库 – 添加镜像仓库](#)。

为了实现集群的资源被有效调度和分配同时提高资源的利用率，平台采用了 request 和 limit 两种限制类型对资源进行分配。request 通常是容器使用的最小资源需求，而 limit 通常是容器能使用资源的最大值，设置为 0 表示对使用的资源不做限制，可无限的使用。request 能保证 pod 有足够的资源来运行，而 limit 则是防止某个 Pod 无限制的使用资源，导致其他 Pod 崩溃。

**表1：CPU 配额说明**

参数	说明
<b>最小使用 (requests)</b>	容器使用的 CPU 最小值，作为容器调度时资源分配的判断依赖。 只有当节点上可分配 CPU 总量 $\geq$ 容器 CPU 最小值时，才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的 CPU 最大值。

**表2：内存配额说明**

参数	说明
<b>最小使用 (requests)</b>	容器使用的最小内存需求，作为容器调度时资源分配的判断依赖。 只有当节点上可分配内存总量 $\geq$ 容器内存申请数时，才允许将容器调度到该节点。
<b>最大使用 (limits)</b>	容器能使用的内存最大值，如果内存使用量超过这个限定值，容器可能会被 kill。



### 3.3. 如果用户有更进一步的需求，可以点击 **高级选项**。

- **命令**: 可自定义容器的启动命令, Kubernetes 的容器启动命令可参见 [Kubernetes 官方文档](#)。
- **参数**: 可自定义容器的启动参数, Kubernetes 的容器启动的参数可参见 [Kubernetes 官方文档](#)。
- **端口**: 即容器端口, 用于指定容器需要暴露的端口, 端口协议可以选择 TCP 和 UDP。
- **环境变量**: 环境变量是指容器运行环境中设定的一个变量, 与 Dockerfile 中的 “ENV” 效果相同, 为创建工作负载提供极大的灵活性。
- **引入配置中心**: 支持添加 Secret 和 ConfigMap 作为环境变量, 用来保存键值对形式的配置数据, 详见 [配置](#) 和 [密钥](#)。
- **镜像拉取策略**: imagePullPolicy, 默认的镜像拉取策略是 IfNotPresent, 在镜像已经存在的情况下, kubelet 将不再去拉取镜像。如果需要频繁拉取镜像, 则设置拉取策略为 Always。如果容器属性 imagePullPolicy 设置为 IfNotPresent 或者 Never, 则会优先使用本地镜像。

The screenshot shows the 'Task Configuration' (任务设置) step in the KubeSphere UI. The configuration includes:

- 命令 (Command):** -wle  
print bpi(2000)
- 参数 (Parameters):** 可以添加参数。
- 端口 (Ports):** 名称: , 类型: TCP, 端口: , 可以添加端口。
- 环境变量 (Environment Variables):** 可以添加环境变量，引用配置中心。
- 镜像拉取策略 (Image Pull Policy):** IfNotPresent

上述配置信息填写完成以后，点击 **保存**，然后点击 **下一步**。

## 第四步：存储卷设置

在存储卷页面可以添加以下三类存储存储卷：

### 持久化存储卷

持久化存储卷可用于持久化存储用户数据，需要预先创建存储卷，参考 [存储卷 – 创建存储卷](#)。

### 临时存储卷

临时存储卷是 `emptyDir` 类型，随 Pod 被分配在主机上。当 Pod 从主机上被删除时，临时存储卷也同时会删除，存储卷的数据也将永久删除，容器崩溃不会从节点中移除 Pod，因此 `emptyDir` 类型的卷中数据在容器崩溃时是安全的。

## 引用配置中心

引入配置中心支持配置 ConfigMap 或 Secret 中的值添加为卷，支持选择要使用的密钥以及将公开每个密钥的文件路径，最后设置目录在容器中的挂载路径。

其中，Secret 卷用于将敏感信息（如密码）传递到 Pod。您可以将 Secret 存储在 Kubernetes API 中，并将它们挂载为文件，以供 Pod 使用，而无需直接连接到 Kubernetes。Secret 卷由 tmpfs（一个 RAM 支持的文件系统）支持，所以它们永远不会写入非易失性存储器。

ConfigMap 用来保存键值对形式的配置数据，这个数据可以在 Pod 里使用，或者被用来为像 Controller 一样的系统组件存储配置数据。虽然 ConfigMap 跟 Secret 类似，但是 ConfigMap 更方便的处理不含敏感信息的字符串。它很像 Linux 中的 /etc 目录，专门用来存储配置文件的目录。ConfigMaps 常用于以下场景：

- 设置环境变量的值
- 在容器里设置命令行参数
- 在数据卷里面创建 config 文件



## 第五步：标签设置

标签（Label）用于指定资源对应的一组或者多组标签。Label 以键值对的形式附加到任何对象上，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod（或其他对象）定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用（日志记录、监控、报警等）。通过多个标签的

设置，我们就可以多维度地对对象进行精细化管理。



## 第六步：添加节点选择器

带有标签的对象创建好之后，我们就可以通过选择器 (Selector) 来引用这些对象。节点选择器页面，用户可以通过按节点选择或通过设置一组或者多组键值对来指定期望运行容器组的主机。比如给 Node 打上标签 “disktype=ssd”，然后给 Pod 添加选择器 “disktype=ssd”，那么该 Pod 将调度到标签为 “disktype=ssd” 的 Node 上。



点击创建，即可完成定时任务的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

# 定时任务

定时任务 (CronJob)，是基于时间的 Job，就类似于 Linux 系统的 crontab，在指定的时间周期运行指定的 Job，在给定时间点只运行一次或周期性地运行，执行完成后 Pod 便会停止。定时计划设置可设置任务的执行周期，比如每隔 1 min 执行一次任务：`*/1 * * * *`，定时计划的格式参考 [CRON](#)。

## 创建定时任务

登录 KubeSphere 控制台，在已创建的项目下，进入 **工作负载 → 定时任务**，进入定时任务列表页面，左上角为当前所在项目。如果是管理员登录，可以看到集群所有项目的定时任务情况，如果是普通用户，则只能查看授权项目下的定时任务。列表顶部显示了当前项目的定时任务 Pod 配额和数量信息。

The screenshot shows the KubeSphere web interface. On the left, there's a sidebar with project navigation (demo-namespace), general information, applications, workload management (with '定时任务' highlighted by a red arrow), storage volumes, network and services, configuration center, and project settings. The main content area is titled '定时任务' (CronJob) and contains a brief description: '定时任务 (CronJob) 管理基于时间的任务，例如在给定时间点只运行一次，或周期性地在给定时间点运行。'. It shows current resource usage: 0 used, infinity planned, and infinity remaining. Below this is a large button labeled '创建定时任务' (Create CronJob). A red arrow points to this button.

## 第一步：填写基本信息

1.1. 点击 **创建** 按钮，将弹出创建定时任务的详情页。创建定时任务支持三种方式，**页面创建**，**导入 yaml 文件**，**编辑模式**。以下主要介绍页面创建的方式，若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式，且支持下载配置文件。其中导入 yaml 文件方式会自动将 yaml 文件内容填充到页面上，用户根据需要可以在页面上调整后再行创建。编辑模式可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件并创建任务。

创建CronJob

编辑模式

```
1 apiVersion: batch/v1beta1
2 kind: CronJob
3 metadata:
4   name: cronjob-qno1zu
5   namespace: project-st75wr
6   labels:
7     app: cronjob-qno1zu
8   annotations:
9     displayName: demo-cronjob
10    desc: ''
11 spec:
12   concurrencyPolicy: Forbid
13   jobTemplate:
14     metadata:
15       labels:
16         app: cronjob-qno1zu
17     spec:
18       template:
19         spec:
20           containers: []
21             restartPolicy: Never
22             serviceAccount: default
23           backoffLimit: 4
24           completions: 6
25           parallelism: 2
```

Yaml / Json

创建

基本信息页中，需要填写任务的名称和描述信息。

- 名称：为创建的定时任务起一个简洁明了的名称，便于用户浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。
- 描述信息：简单介绍定时任务的主要特性，让用户进一步了解该定时任务。
- 定时计划：必需字段，指定定时任务的运行周期，格式同 [CRON](#)。

点击 **高级选项**，还可以对一些可选配置如并发策略、启动 Job 的期限、保留历史等进行设置。

- 启动 Job 的期限 (StartingDeadlineSeconds)：启动 Job 的期限（秒级别），如果因为任何原因而错过了被调度的时间，那么错过执行时间的 Job 将被认为是失败的。如果没有指定，则没有期限。
- 保留完成 Job 数 (SuccessfulJobsHistoryLimit)：允许保留的成功的任务个数。
- 保留失败 Job 数 (FailedJobsHistoryLimit)：允许保留的失败的任务个数。
- 并发策略 (ConcurrencyPolicy)：并发策略，它指定了如何处理被 CronJob 创建的 Job 的并发执行。只允许指定下面策略中的一种：
  - Allow：允许并发运行 Job。
  - Forbid（默认）：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。

完成后点击 **下一步**。

创建定时任务

基本信息  定时任务设置  容器组模板  存储卷设置  标签设置  节点选择器

名称 \*  
demo-cronjob  
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目  
demo-namespace  
将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名  
定时任务  
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

定时计划 \*  
0 \* \* \* \*  
Every hour, on the hour 语法参照 CRON

描述信息  
This is a demo

高级选项 ▾

启动 Job 的期限 (秒)	保留完成 Job 数
4	
即在指定 启动时间 + 启动 Job 的期限 这个周期之内都可以启动任务	
保留失败 Job 数	并发策略
	Forbid
允许保留的失败的任务个数	

## 第二步：任务设置

定时任务 (CronJob) 同时也具备任务 (Job) 的如下 Job Spec 格式，通过设置四个配置参数来设置 Job 的任务类型。

Back Off Limit：失败尝试次数，若失败次数超过该值，则 Job 不会继续尝试工作；如设置为 4 则表示最多重试 4 次。

- Completions：标志任务结束需要成功运行的 Pod 个数，如设置为 6 则表示任务结束需要运行 6 个 Pod。
- Parallelism：标志并行运行的 Pod 的个数；如设置为 2 则表示并行 2 个 Pod。
- Active Deadline Seconds：标志失败 Pod 的重试最大时间，超过这个时间不会继续重试，且 ActiveDeadlineSeconds 优先级高于 Back Off Limit；如设置 500 则表示达到 500s 时 Job 即其所有的 Pod 都会停止。

创建CronJob

编辑模式

基本信息  任务设置  任务模板  存储卷设置  标签设置  节点选择器

### 任务设置

您可以在此配置任务(Job)的Job Spec格式, Job Controller负责根据Job Spec创建Pod, 并持续监控Pod的状态, 直至其成功结束。如果失败, 则根据RestartPolicy (支持OnFailure和Never) 决定是否创建新的Pod再次重试任务。

Back off Limit 4	Completions 6
失败尝试次数, 若失败次数超过该值, 则 job 不会继续尝试工作	标志 Job 结束需要成功运行的 Pod 个数
Parallelism 2	Active Deadline Seconds 500
标志并行运行的 Pod 的个数	job 运行的超时时间

上一步  下一步

### 第三步：配置任务模板

3.1. 任务模板即设置 Pod 模板, 其中 **重启策略 (RestartPolicy)** 指通过同一节点上的 kubelet 重新启动容器, 仅支持 Never 或 OnFailure, 当任务未完成的情况下:

- Never: 任务会在容器组出现故障时创建新的容器组, 且故障容器组不会消失, 返回的字段 “.status.failed” 加 1。
- OnFailure: 任务会在容器组出现故障时其内部重启容器, 而不是创建新的容器组, 返回的字段 “.status.failed” 不变。

3.2. 点击 **添加容器**, 然后根据需求添加容器镜像, 容器中定义的镜像默认从 Docker Hub 中拉取。输入容器的名称和对应的镜像名, 镜像名一般需要指定 tag, 比如 perl:5.28.0。

**说明:** 若需要使用私有镜像仓库如 Harbor, 参见 [镜像仓库 – 添加镜像仓库](#)。

为了实现集群的资源被有效调度和分配同时提高资源的利用率, 平台采用了 request 和 limit 两种限制类型对资源进行分配。request 通常是容器使用的最小资源需求, 而 limit 通常是容器能使用资源的最大值, 设置为 0 表示对使用的资源不做限制, 可无限的使用。request 能保证 pod 有足够的资源来运行, 而

limit 则是防止某个 Pod 无限制的使用资源，导致其他 Pod 崩溃。

表1：CPU 配额说明

参数	说明
最小使用 (requests)	容器使用的 CPU 最小值，作为容器调度时资源分配的判断依赖。 只有当节点上可分配 CPU 总量 $\geq$ 容器 CPU 最小值时，才允许将容器调度到该节点。
最大使用 (limits)	容器能使用的 CPU 最大值。

表2：内存配额说明

参数	说明
最小使用 (requests)	容器使用的最小内存需求，作为容器调度时资源分配的判断依赖。 只有当节点上可分配内存总量 $\geq$ 容器内存申请数时，才允许将容器调度到该节点。
最大使用 (limits)	容器能使用的内存最大值，如果内存使用量超过这个限定值，容器可能会被 kill。

创建定时任务

编辑模式

基本信息  定时任务设置  容器组模板  存储卷设置  标签设置  节点选择器

◀ 添加容器

容器名称 \*  镜像 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾  
要从私有镜像仓库部署，需要先创建镜像仓库，然后拉取镜像。

CPU

最小使用	<input type="text" value="10"/>	m	最大使用	<input type="text" value="100"/>	m
------	---------------------------------	---	------	----------------------------------	---

作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量  $\geq$  容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m

内存

最小使用	<input type="text" value="10"/>	Mi	最大使用	<input type="text" value="200"/>	Mi
------	---------------------------------	----	------	----------------------------------	----

作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量  $\geq$  容器内存最小使用值时，才允许将容器调度到该节点。

高级选项 ▾

命令

<input type="text" value="命令"/>	<input type="button" value="删除"/>
---------------------------------	-----------------------------------

3.3. 如果用户有更进一步的需求，可以点击 **高级选项**。

- **命令**: 可自定义容器的启动命令，Kubernetes 的容器启动命令可参见 [Kubernetes 官方文档](#)。
- **参数**: 可自定义容器的启动参数，Kubernetes 的容器启动的参数可参见 [Kubernetes 官方文档](#)。
- **端口**: 即容器端口，用于指定容器需要暴露的端口，端口协议可以选择 TCP 和 UDP。
- **环境变量**: 环境变量是指容器运行环境中设定的一个变量，与 Dockerfile 中的“ENV”效果相同，为创建的工作负载提供极大的灵活性。
- **引入配置中心**: 支持添加 Secret 和 ConfigMap 作为环境变量，用来保存键值对形式的配置数据，详见 [配置和密钥](#)。
- **镜像拉取策略**: imagePullPolicy，默认的镜像拉取策略是 IfNotPresent，在镜像已经存在的情况下，kubelet 将不再去拉取镜像。如果需要频繁拉取镜像，则设置拉取策略为 Always。如果容器属性 imagePullPolicy 设置为 IfNotPresent 或者 Never，则会优先使用本地镜像。

上述配置信息填写完成以后，点击 **保存**，然后点击 **下一步**。

## 第四步：存储卷设置

在存储卷页面可以添加以下三类存储卷：

### 持久化存储卷

持久化存储卷可用于持久化存储用户数据，需要预先创建存储卷，参考 [存储卷 – 创建存储卷](#)。

### 临时存储卷

临时存储卷是 `emptyDir` 类型，随 Pod 被分配在主机上。当 Pod 从主机上被删除时，临时存储卷也同时会删除，存储卷的数据也将永久删除，容器崩溃不会从节点中移除 Pod，因此 `emptyDir` 类型的卷中数据在容器崩溃时是安全的。

### 引用配置中心

引入配置中心支持配置 ConfigMap 或 Secret 中的值添加为卷，支持选择要使用的密钥以及将公开每个

密钥的文件路径，最后设置目录在容器中的挂载路径。

**Secret** 用于将敏感信息（如密码）传递到 pod。您可以将 Secret 存储在 Kubernetes API 中，并将它们挂载为文件，以供 Pod 使用，而无需直接连接到 Kubernetes。Secret 卷由 tmpfs（一个 RAM 支持的文件系统）支持，所以它们永远不会写入非易失性存储器。

**ConfigMap** 用来保存键值对形式的配置数据，这个数据可以在 Pod 里使用，或者被用来像 Controller 一样的系统组件存储配置数据。虽然 ConfigMap 跟 Secret 类似，但是 ConfigMap 更方便的处理不含敏感信息的字符串。它很像 Linux 中的 /etc 目录，专门用来存储配置文件的目录。

ConfigMaps 常用于以下场景：

- 设置环境变量的值
- 在容器里设置命令行参数
- 在数据卷里面创建 config 文件



## 第五步：标签设置

标签（Label）用于指定资源对应的一组或者多组标签。Label 以键值对的形式附加到任何对象上，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod（或其他对象）定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用（日志记录、监控、报警等）。通过多个标签的设置，我们就可以多维度地对对象进行精细化管理。



## 第六步：添加节点选择器

带有标签的对象创建好之后，我们就可以通过选择器 (Selector) 来引用这些对象。节点选择器页面，用户可以通过按节点选择或通过设置一组或者多组键值对来指定期望运行容器组的主机。比如，给 Node 打上标签 “disktype=ssd”，然后给 Pod 添加选择器 “disktype=ssd”，那么，该 Pod 将调度到标签为 “disktype=ssd” 的 Node 上。



点击创建，即可完成定时任务的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

# 健康检查器

## 简介

在实际的生产环境中，如果要使开发者提供的应用程序没有任何 Bug，并且一直保持运行正常，这几乎是不可能完成的任务。那么，一套管理系统对运行的应用程序进行周期性的健康检查和修复就是不可或缺的了，而在底层的 Kubernetes 中，系统和应用程序的健康检查任务是由 kubelet 来完成的。在某些特殊的场景下，例如一个典型的程序发生“死锁”的例子，虽然 Docker 会认为其容器进程一直在运行，但从应用程序角度而言该状态下的容器将不会正常响应用户的业务请求，因此在业务级的监控检查方面，Kubernetes 定义了两种类型的健康检查探针。

在 KubeSphere 中，用户可以为容器设置健康检查探针 (Probe) 来检查容器的健康状态。因为 kubelet 会根据用户定义的这个健康检查探针的返回值，来决定容器的状态，而不是直接以容器是否运行 (来自 Docker 返回的信息) 作为判断依据。这种健康检查的机制，在实际的生产环境中是保证应用程序正常运行的重要手段。至于什么状态才算正常，则由用户自己定义。

KubeSphere 支持添加以下两种健康检查探针：

- **存活探针 (liveness Probe)**

存活探针用于检测容器是否存活，类似于我们在 Linux 中执行 ps 命令来检查系统中的进程是否存在。kubelet 根据用户定义的 periodSeconds (默认为 10 秒) 周期性地对容器的健康状态进行检查，如果检查失败，集群会根据容器组的重启机制 (RestartPolicy，默认为 Always) 对容器执行重启操作，若检查成功则不执行任何操作。

- **就绪探针 (Readiness Probe)**

另一种健康检查方案叫就绪探针，虽然它的用方法与存活探针相似，但作用却大不相同，就绪探针用于检测结果的成功与否，检测容器是否准备好开始处理用户请求。如果检查结果是 fail 的，kubelet 不会杀死容器进程而是将其所属的容器组 (Pod) 从 endpoint 列表删除，然后访问该容器组的请求会被路由到其他容器组。只有当 Pod 中的容器都处于就绪状态时 kubelet 才会认定其处于就绪状态，它决定 Pod 是否能被通过服务 (Service) 的方式访问到。

存活探针和就绪探针可以并行用于同一容器，使用这两类探针能够确保流量将不会到达未准备好的容器，且容器能在失败时重新启动。

The screenshot shows the KubeSphere interface for editing a configuration template. On the left, there's a sidebar with options like '更新策略' (Update Strategy), '容器组模板' (Container Group Template), '健康检查器' (Health Check), and '存储卷' (Storage Volumes). The '健康检查器' option is selected and highlighted with a dark blue background. The main content area is titled '设置健康检查器' (Set Health Check) and shows a container named 'container-z4zqrz' using the 'busybox' image. It provides detailed information about Readiness Probes and Liveness Probes, including their descriptions and '添加探针' (Add Probe) buttons.

## 健康检查方式

在容器中支持以下三种健康检查方式，Pod 可以暴露一个健康检查 URL (比如 /health)，或者直接让健康检查探针去检测应用的监听端口，这两种方式在 Web 服务类的应用非常广泛，除此之外还可以在容器中定期执行命令来检查。

### HTTP GET (HTTP 请求检查)

HTTP 请求检查主要针对提供 HTTP/HTTPS 服务的容器，集群将周期性对这类容器发起 HTTP/HTTPS GET 请求，查看 HTTP/HTTPS response 返回码，如果其属于 200~399 范围，则说明检查结果成功，否则检查失败。若使用 HTTP 请求检查，需指定容器监听的端口和 HTTP/HTTPS 的请求路径 (path)。

- 端口：访问容器的端口号，介于 1 ~ 65535 之间
- 路径：访问的 HTTP server 的 路径 (path)

类型

HTTP GET Container Command TCP Socket

https

路径 / 端口 \*

初始延迟(秒) 0 超时时间(秒) 0

在检查其运行状况之前，容器启动后需要等待多长时间。

等待探头完成多长时间。如果超过时间，则认为探测失败。

## TCP Socket (TCP 端口检查)

TCP 端口检查主要针对 TCP 通信服务的容器，系统将周期性地与该容器建立 TCP 连接，如果连接成功直接说明健康检查成功，否则检查失败。如果选择该方式，需指定容器监听的端口。

比如，在 [快速入门](#) 中创建了一个 MySQL 容器，服务端口为 3306，如果对该容器配置了 TCP 端口探测，指定探测端口 3306，系统将周期性地对该容器的 3306 端口发起 TCP 连接，若连接成功则说明检查成功，否则失败。

类型

HTTP GET Container Command TCP Socket

端口 \* 80

初始延迟(秒) 0 超时时间(秒) 0

在检查其运行状况之前，容器启动后需要等待多长时间。

等待探头完成多长时间。如果超过时间，则认为探测失败。

## Container Command (容器命令检查)

容器命令检查是一种强大的检查方式，需要指定一个在 **容器内** 的可执行命令，系统将周期性地在容器内执行该命令，如果命令返回结果为 0 则说明检查成功，否则检查失败。

对于上面提到的 TCP 端口检查和 HTTP 请求检查，都可以通过执行命令检查的方式来替代。比如 TCP 端口探测，可以写一个程序对容器的端口进行连接访问，如果连接成功，脚本返回 0，否则连接失败。对于 HTTP 请求检查，同样写一个脚本对容器进行 wget 操作，比如 wget <http://127.0.0.1:80/path>



#### 说明：以下配置属于上述三种方式的公共参数

- 初始延迟 (initialDelaySeconds): 容器启动后第一次执行探测是需要等待多少秒
- 超时时间 (timeoutSeconds): 探测超时时间，单位为秒

#### 以下几个配置支持通过在页面编辑 yaml 模板文件进行配置：

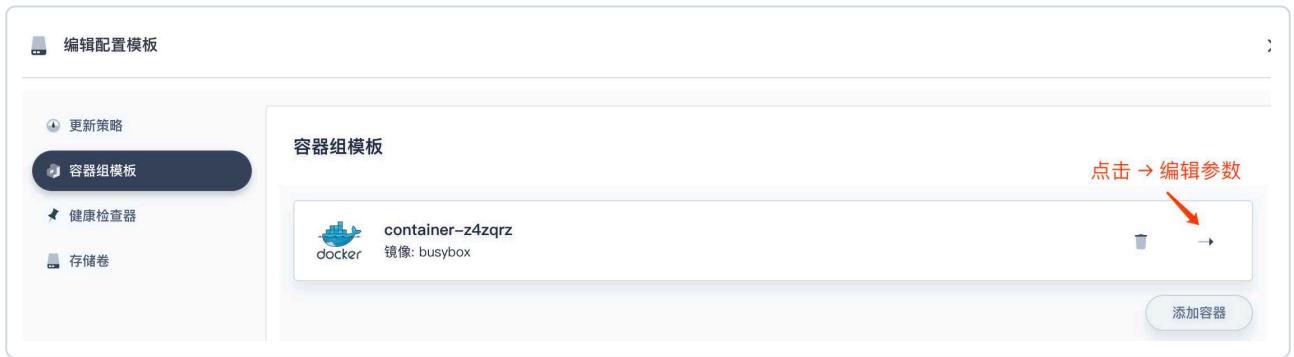
- 检查频率 (periodSeconds): 执行探测的频率，默认是 10 秒。
- successThreshold: 检查失败后，最少连续检查成功多少次才被认定为成功，默认为 1，而对于存活探针其必须是 1
- failureThreshold: 检查成功后，最少连续检查失败多少次才被认定为失败， 默认为 3

## 设置健康检查器

健康检查器支持在 **工作负载** 的部署、有状态副本集和守护进程集中设置检查探针，以创建一个 **busybox** 容器的部署 (deployment) 为例，演示如何设置一个存活探针 (就绪探针配置类似)，请确保已创建了该部署资源，若还未创建请参考 [创建部署](#)。

### 第一步：设置容器参数

另外，创建部署时在 busybox 容器内需设置以下三个参数，它在容器启动后在 `/tmp` 目录下创建了一个 `healthy` 文件，以此作为容器已正常运行的标志，而 30 s 后该文件将被删除。在创建部署的 **容器组模板** 中，或在 **部署详情页** → **更多操作** 选择 **编辑配置模板** 都可以设置容器组的参数。



```
/bin/sh
-c
touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
```

## 第二步：设置存活探针

同时，配置一个存活探针并定义容器内执行的命令 `cat /tmp/healthy`，如果该文件存在，命令执行后的返回值就是 0，认为该容器是健康的，检查将在容器启动 5 秒后执行，默认每 10 秒执行一次，超时时间为 5 秒。

现在，开始具体实践一下这个过程：

在项目下，进入 **工作负载 → 部署** 选择 busybox 容器的部署，进入部署详情页。点击 **更多操作 → 编辑配置模板**。

选择 **监控检查器**, 以设置容器命令 **Container Command** 为检查方式, 参考如下设置填写:

### 第三步：验证健康检查

创建完成后, 将生成一个新的版本 #2, 可以在 **版本控制中查看**。通过页面的 Kubectl 工具可以查看这个 Pod 的状态是 Running, 而 30 秒之后 Pod 在 Events 中报告了一个异常:

```
$ kubectl describe pod busybox-86c54ccdfc-jgz9x -n project-st75wr
Type      Reason     Age         From           Message
----      ----     --         --           --
Warning   Unhealthy  27s (x6 over 2m)  kubelet, i-w083nj02  Liveness probe failed: cat: can't open '
```

是因为 30 秒后 **/tmp/healthy** 目录被命令删了, 存活探针检测到目录不存在了, 所以报告容器是不健康的,

此时查看该 Pod 的状态，发现其状态没有 Failed，反而还是 Running，但 RESTARTS 从 0 变成了 1，这是因为存活探针检查容器状态异常后，容器已被系统重启了。Pod 的重启机制 (RestartPolicy)，默认值就是 Always，说明任何时候容器发生异常就会被自动重启。

```
$ kubectl get pod busybox-86c54ccdfc-jgz9x -n project-st75wr
NAME           READY   STATUS    RESTARTS   AGE
busybox-86c54ccdfc-jgz9x  0/1     Running   1          3h
```

重启机制除了 Always 还有 OnFailure 和 Never 两种情况：

- Always：任何情况下，只要容器出现异常系统将自动重启容器
- OnFailure：只在容器异常时，才会自动重启
- Never：无论某种情况都不重启容器

至此，您已经初步地熟悉了如何对容器设置健康检查器，若希望了解更高级的配置方法，可参考 [Kubernetes 官方文档](#)。

# 工作负载管理

当创建了工作负载比如部署、有状态副本集、任务这类资源对象后，KubeSphere 支持对其执行查看、扩缩容、启停、删除、升级、资源监控等基本操作，以部署为例，说明如何查看、编辑和删除部署，其它类型的工作负载操作基本相似。

## 查看部署详情

在部署列表页，点击某个部署的名称，就可以进入到部署详情页，可以查看部署的基本信息、资源状态、版本控制、监控、环境变量、事件等。点击 **更多操作** 支持版本回退、弹性伸缩、编辑配置模板、编辑 yaml 配置文件或删除当前部署。

The screenshot shows the KubeSphere Deployment Resource Status page for a deployment named 'test'. On the left, there's a sidebar with deployment details: app: test, project: test, created at 2018-11-28 23:24:49, updated at 2018-11-29 12:04:55, and created by admin. The main content area has tabs for 资源状态 (Resource Status), 版本控制 (Version Control), 监控 (Monitoring), 环境变量 (Environment Variables), and 事件 (Events). The 资源状态 tab is active, showing a summary card with 2/2 replicas,期望副本: 2, 实际运行副本: 2. Below this is a 弹性伸缩 (Scaling) section with sliders for 最小副本数 (Min Replicas) set to 2 and 最大副本数 (Max Replicas) set to 10, along with CPU and memory usage metrics. At the bottom, there are two container group cards: 'test-68797bff58-55rpz' and 'test-68797bff58-rcfw7', each with its IP address, host, and resource usage (CPU 0m, Memory 2.23MiB and 2.42MiB).

## 版本控制页

比如部署、有状态副本集、守护进程集这类工作负载都有版本的概念，若修改了其中的配置模板或 yaml 文件并点击更新后，就会新生成一个版本，在版本控制页即可查看到所有的历史版本，点击其中任意一个版本右侧的图标，即可查看其配置模板（支持 yaml 和 json）。若更新后的版本存在问题，点击 **更多操作** 支持将其版本回退到历史中的任何一个版本。

The screenshot shows the deployment details for a deployment named 'test'. On the left, there's a sidebar with deployment information and a '更多操作' (More Operations) dropdown. The main area has tabs for '资源状态' (Resource Status), '版本控制' (Version Control, highlighted in green), '监控' (Monitoring), '环境变量' (Environment Variables), and '事件' (Events). The '版本控制' tab displays two versions: '#1' (created on 2018-11-28 23:24:49) and '#2' (created on 2018-11-29 00:15:04, marked as running).

## 监控

监控详情页支持对部署资源的所有容器组 (Pod) 的 CPU 和内存的使用量、网络进、出口流量等四项指标，按自定义的时间范围进行搜索查看。

The screenshot shows the deployment details for a deployment named 'test'. The left sidebar includes deployment information and a '更多操作' (More Operations) dropdown. The right side features a '监控' (Monitoring) tab with three charts: 'CPU 使用量 (m)' (CPU Usage), '内存使用量 (MiB)' (Memory Usage), and '网络出口流量 (Bps)' (Network Outbound Traffic). A time range selector allows for choosing from '最近 10 分钟' to '最近 2 天', or a '自定义时间范围' (Custom Time Range) with fields for '开始时间' (Start Time) and '结束时间' (End Time). The 'CPU 使用量' chart shows usage spikes between 04:01 and 05:01. The '内存使用量' chart shows usage fluctuating between 1.5 and 3 MiB over the same period. The '网络出口流量' chart shows traffic between 04:01 and 12:01.

当容器组 (Pod) 数量超过五个副本时，可以单击 [查看全部副本](#) 查看更多的副本监控。例如，查看一个副本数为 203 的守护进程集 `node-exporter` (部署在集群的所有主机上用于监控集群物理主机的 CPU、内存、网络、磁盘等情况)，点击 [查看全部副本](#)：



弹窗将显示 203 个副本的监控数据，左侧可以选择多个容器组，每个容器组监控的数据将以单个曲线图显示。这对于集群规模较大的情景而言，是非常适用的。



## 环境变量页

可查看当前部署中所有容器的环境变量。

The screenshot shows the KubeSphere interface for managing a Wordpress deployment. On the left, there's a sidebar with navigation links like 'project-st75wr / deployments / wordpress / env'. The main content area has tabs for '资源状态' (Resource Status), '版本控制' (Version Control), '监控' (Monitoring), '环境变量' (Environment Variables) (which is selected and highlighted in green), and '事件' (Events). Under '环境变量', two containers are listed: 'wordpress-container-2' and 'wordpress'. Both containers have environment variables defined: WORDPRESS\_DB\_PASSWORD (123456) and WORDPRESS\_DB\_HOST (mysql-service).

## 事件页

相当于 Kubernetes 中的 Events，它是 kubelet 负责用来记录多个容器运行过程中的事件，包含容器事件（创建、启动、失败等）、镜像事件（镜像拉取失败等）、kubelet 事件（节点失效、节点不可调度等）、Pod Worker 事件（同步失败）等，在实际运行环境中方便用户排错和快速定位问题。

The screenshot shows the KubeSphere interface for managing a busybox deployment. The left sidebar shows 'busybox' under 'busybox' with a 'events' tab selected. The main content area displays a table of events:

原因	状态	消息
ScalingReplicaSet 2018-11-29 11:54:21	通用	Scaled up replica set busybox-74fb8b7c4b to 1
ScalingReplicaSet 2018-11-29 11:54:26	通用	Scaled down replica set busybox-86c54ccdfc to 0

## 编辑或删除部署

可通过 **更多操作** 进行版本回退、弹性伸缩、编辑配置模板和编辑配置文件。其中，编辑配置模板支持修改更新策略、容器组模板、健康检查器和存储卷等配置，编辑配置文件在代码模式下是编辑对应的 yaml 文件，完成后点击右下方的更新按钮，就会按照配置文件进行更新。如果只是修改部署的描述信息，可以使用部署详情页左上角的 **编辑信息** 进行修改。

KUBE SPHERE

default / deployments / my-release-harbor-notary-server / resource-status

my-release-harbor-notary-server

资源状态 版本控制 监控 环境变量 事件

副本运行状态 期望副本: 3 实际运行副本: 3

3/3

标签

app: harbor component: notary release: my-release

版本回退 弹性伸缩 编辑配置模板 编辑配置文件 删除

编辑信息 更多操作

详情

项目: default  
唯一标识符: my-release-harbor-notary-server  
应用: my-release/harbor-0.2.0  
创建时间: 2018-09-21 13:24:59  
更新时间: 2018-10-15 21:09:37  
创建者: 未知

容器组

容器组	IP	主机	CPU	内存
my-release-harbor-notary-server-5757f58959-4v47t	10.244.4.218	i-5xcldxos(192.168.0.55)	0.29m	9.32MiB
my-release-harbor-notary-server-5757f58959-gc4nc	10.244.3.72	i-e8oiua90(192.168.0.56)	0.33m	8.89MiB

## 修改副本数

点击蓝色部分的上、下图标即可增加或减少容器组的期望副本数 (Desired Replicas)，只有部署和有状态副本集可以修改其副本数。如下所示，点击 **增加副本数** 图标，期望副本数由 1 增加至 2，可以看到容器组列表显示新增了一个 Pod，状态显示 **ContainerCreating**。

副本运行状态  
期望副本: 2  
实际运行副本: 2

增加副本数  
减少副本数

端口

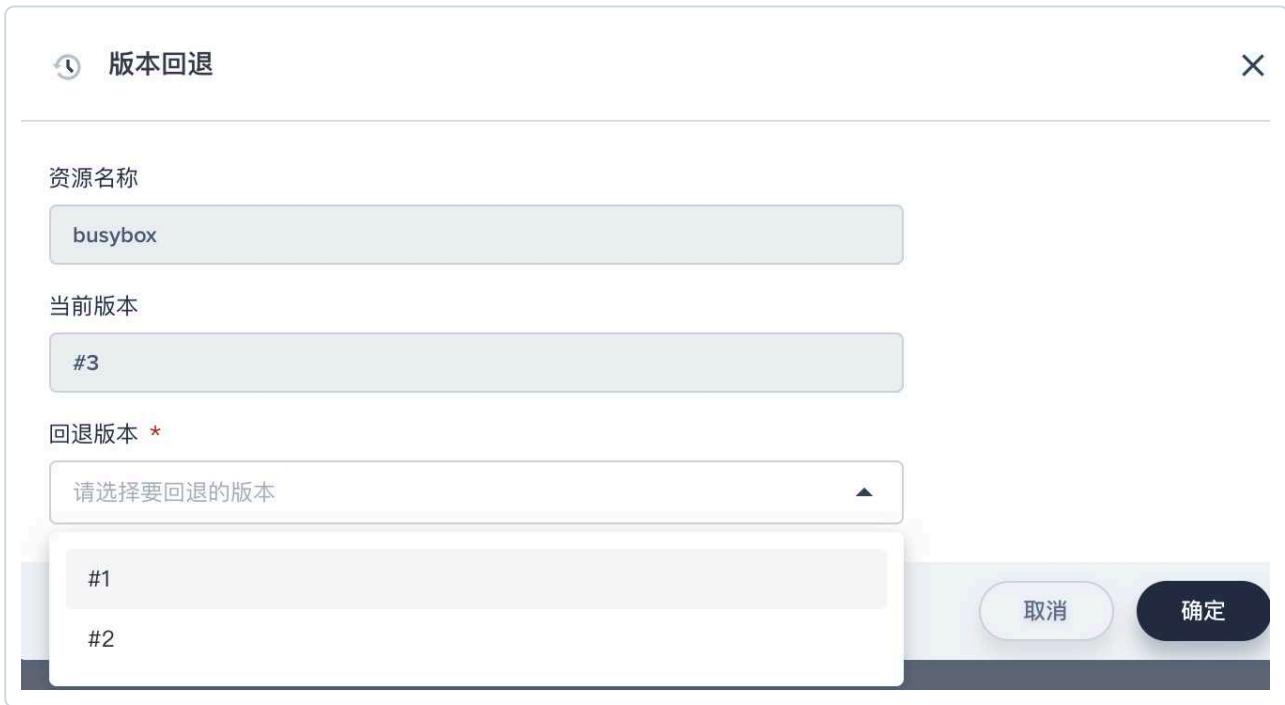
名称	协议	端口	主机端口
port	TCP	3306	-

容器组

wordpress-mysql-0 创建于 1 天前	容器组 IP: 10.244.5.34 主机: i-8q3txahx(192.168.0.18)	CPU 0.78m 	内存 185.92MIB 
wordpress-mysql-1 ContainerCreating	容器组 IP: - 主机: i-76v18j2o(192.168.0.53)	暂时没有监控数据	

## 版本回退

若修改了其中的配置模板或 yaml 文件并点击更新后，就会新生成一个版本，如果该新版本存在问题，可以点击 **版本回退**，在弹窗中选择需要回退的历史版本。

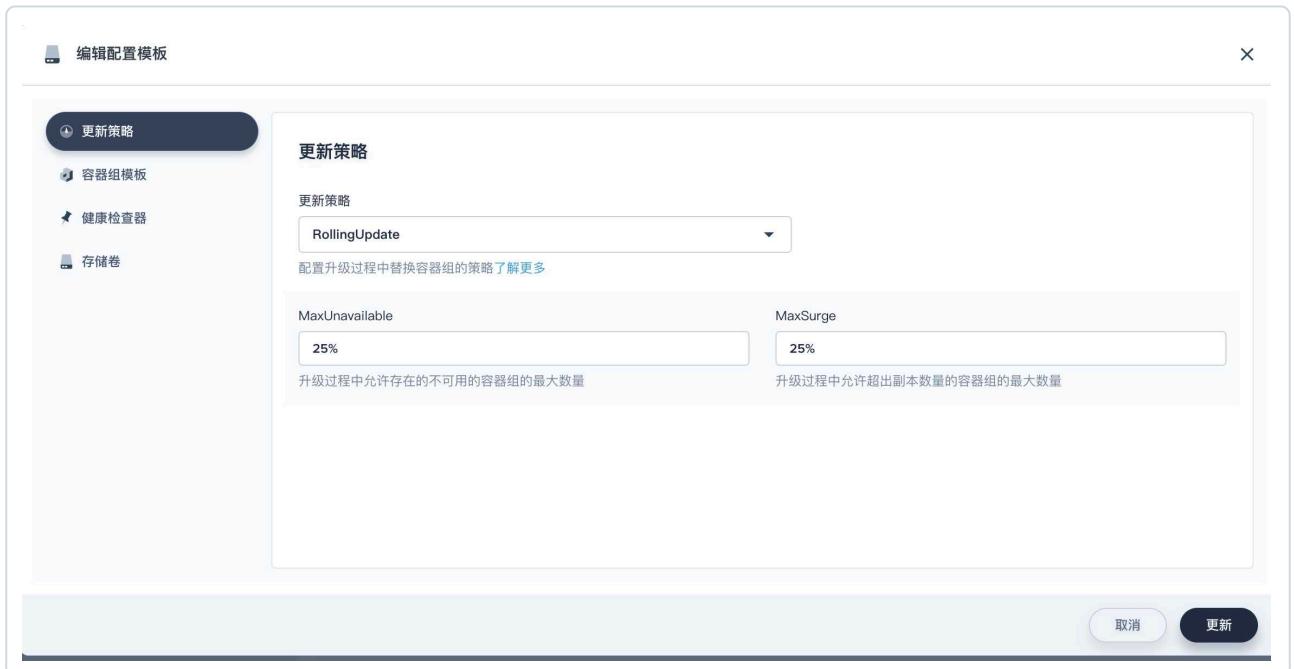


## 配置弹性伸缩

弹性伸缩 (Horizontal Pod Autoscaling) 顾名思义就是使 Pod 能够水平自动伸缩，利用弹性伸缩，KubeSphere 能够根据监测到的 CPU 利用率或内存使用率自动地扩容或缩容部署 (Deployment)，也只有部署才可以配置弹性伸缩。关于如何配置弹性伸缩，详见 [弹性伸缩](#)。

## 编辑配置模板

编辑配置模板是在创建了工作负载之后，需要对其修改 [更新策略](#)、[容器组模板](#)、[添加健康检查器](#)和[存储卷](#) 等配置，其中健康检查器的配置和使用示例可参考这篇文档 – [设置健康检查器](#) (修改其它三类配置模板已在对应用户指南和快速入门中给出了释义和创建示例)。



## 编辑配置文件

如果对配置文件（支持 yaml/json）的语法比较熟悉后，可以点击 [编辑配置文件](#) 来修改当前的工作负载，注意，工作负载的名字不支持修改。另外，更新配置文件后将生成一个新的版本，可以在版本详情页进行查看。

## 编辑配置文件

```
1 kind: Deployment
2 apiVersion: apps/v1
3 metadata:
4   name: test
5   namespace: project-st75wr
6   selfLink: /apis/apps/v1/namespaces/project-st75wr/deployments/test
7   uid: be97b342-f321-11e8-9a45-525445c0b555
8   resourceVersion: '59696517'
9   generation: 10
10  creationTimestamp: '2018-11-28T15:24:49Z'
11  labels:
12    app: test
13  annotations:
14    creator: admin
15    deployment.kubernetes.io/revision: '2'
16    desc: ''
17    displayName: Nginx
18    kubesphere.io/isElasticReplicas: 'true'
19    kubesphere.io/relatedHPA: test
20 spec:
21   replicas: 2
22   selector:
23     matchLabels:
24       app: test
25   template:
26     metadata:
27       creationTimestamp: null
28     labels:
29       app: test
30   spec:
```

## 删除部署

在列表页最右侧的“...”和部署详情页的左上方的更多选项中都提供了部署的删除功能。



The screenshot shows the KubeSphere interface for managing a deployment named 'test'. On the left, there's a sidebar with deployment details like 'app: test'. In the center, there's a main panel with tabs for '资源状态', '版本控制', '监控', '环境变量', and '事件'. The '资源状态' tab is active, showing a summary card with '2/2' replicas running. Below it, there's a section for '弹性伸缩' (Auto Scaling) with sliders for '最小副本数' (Min Replicas) set to 2 and '最大副本数' (Max Replicas) set to 10. A red arrow points to the '删除' (Delete) button in the sidebar, which is part of a dropdown menu labeled '更多操作' (More Operations). A tooltip for the 'Deployment' resource type is visible on the right.

以上以部署为例，介绍了工作负载的常用操作，其它工作负载的操作类似，可同样参考上述步骤。



# 存储概述

存储是为 KubeSphere 平台的容器运行的工作负载 (Pod) 提供存储的组件，支持多种类型的存储，并且同一个工作负载中可以挂载任意数量的存储卷。

## 存储分类

作为一个容器管理平台，除了支持如 Local Volume(仅用于 all-in-one 测试安装), EmptyDir, HostPath 本地存储之外，还需要支持对接开源的分布式文件系统或网络文件系统。KubeSphere 提供多种网络存储方案作为持久化存储，目前支持的存储分类有本地存储和持久化存储，创建持久化存储卷之前需要预先创建存储类型，详见 [创建存储类型](#)。

### 本地存储

#### Local Volume

[Local Volume](#) 表示挂载的本地存储设备，如磁盘、分区或目录，只能用作静态创建的 PersistentVolume。All-in-One 模式安装默认会用 Local Volume 作为存储类型，由于 Local Volume 不支持动态分配，Installer 会预先创建 10 个可用的 10G PV 供使用，若存储空间不足则需要手动创建 Persistent Volume (PV)，参见 [Local Volume 使用方法](#)。

#### EmptyDir

[EmptyDir](#) 的生命周期和所属的 Pod 是完全一致的，可以在同一工作负载内的不同容器之间共享工作过程中产生的文件，在部署、任务和定时任务中支持添加临时存储卷，临时存储卷是使用主机磁盘进行存储的，

#### HostPath

[HostPath](#) 这种会把宿主机上的指定卷加载到容器之中，这种卷一般和有状态副本集 (DaemonSet) 搭配使用，用来操作主机文件，例如进行日志采集的 FLK 中的 FluentD 就采用这种方式，加载主机的容器日志目录，达到收集本主机所有日志的目的。

## 持久化存储

KubeSphere 支持的存储类型有 QingCloud 云平台块存储插件、QingStor NeonSAN 分布式存储、NFS、GlusterFS、Ceph RBD，这类存储类型都支持创建和使用持久化存储卷，并且支持动态存储卷分配 (Dynamic Volume Provisioning)[<https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/>]，必须预先准备好相应的服务端。安装前可在 Installer 的 `conf/vars.yml` 中配置这类存储服务，详见 [存储配置说明](#)。

### QingCloud 云平台块存储

支持使用 QingCloud 云平台块存储作为平台的存储服务，如果希望体验 KubeSphere 推荐的动态分配 (Dynamic Provisioning) 方式创建存储卷，推荐使用 [QingCloud 云平台块存储](#)，平台已集成 [QingCloud-CSI](#) 块存储插件，仅需简单配置即可使用 QingCloud 云平台的各种性能的块存储服务，免去手动配置存储服务端的繁琐，详见 [QingCloud-CSI 参数配置](#)。

### QingStor NeonSAN

支持对接青云自研的企业级分布式存储 [QingStor NeonSAN](#) 作为存储服务，若您准备好 NeonSAN 服务端后，即可在 `conf/vars.yml` 配置 NeonSAN-CSI 插件对接其存储服务端，详见 [存储配置说明](#)

### Ceph RBD

[Ceph RBD](#) 是一个分布式存储系统，KubeSphere 测试过的存储服务端 Ceph RBD 服务端版本为 v0.94.10，[Ceph](#) 服务端集群部署可参考 [部署 Ceph 存储集群](#)，正式环境搭建 Ceph 存储服务集群请参考 [Install Ceph](#)。

### GlusterFS

[GlusterFS](#) 是一个开源的分布式文件系统，Heketi 用来管理 GlusterFS 存储服务端，KubeSphere 测试过的 GlusterFS 服务端版本为 v3.7.6，GlusterFS 部署可参考 [部署 GlusterFS 存储集群](#)，正式环境搭建 GlusterFS 集群请参考 [Install Gluster](#) 或 [Gluster Docs](#) 并且需要安装 [Heketi 管理端](#)，Heketi 版本为 v3.0.0。

## NFS

[NFS](#) 即网络文件系统，它允许网络中的计算机之间通过 TCP/IP 网络共享资源。本地 NFS 的客户端应用可以透明地读写位于远端 NFS 服务器上的文件，就像访问本地文件一样。使用 NFS 需提前准备存储服务端，比如 QingCloud 云平台 [vNAS](#)，然后可以在 Installer 配置参数对接 NFS 存储服务端，详见 [NFS 配置](#)。

## 存储卷

存储卷，具有单个磁盘的功能，供用户创建工作负载使用。在创建存储类型后，即可创建存储卷，并挂载至工作负载，详见 [创建存储卷](#)。

# 存储卷

存储卷，在 KubeSphere 中一般是指基于 PVC 的持久化存储卷，具有单个磁盘的功能，供用户创建的工作负载使用，是将工作负载数据持久化的一种资源对象。

在 all-in-one 部署方式中，可以使用 Local 存储卷将数据持久化，无需存储服务端支持，但此类型存储卷不支持动态分配方式。如果希望体验 KubeSphere 推荐的动态分配 (Dynamic Provisioning) 方式创建存储卷，平台已集成 [QingCloud-CSI](#) 块存储和 [QingStor NeonSAN](#) 插件，支持使用 [QingCloud 云平台块存储](#) 或 [QingStor NeonSAN](#) 作为平台的存储服务，免去手动配置存储服务端的繁琐。

另外，Installer 也已集成了 [NFS](#)、[GlusterFS](#)、[CephRBD](#) 等存储的客户端，需在 `conf/vars.yml` 中配置，但需要自行准备和安装相应的存储服务端，若用于 KubeSphere 测试存储服务端部署可参考附录中的 [部署 Ceph RBD 存储服务端](#) 或 [部署 GlusterFS 存储服务端](#)。

存储卷生命周期包括存储卷创建、挂载、卸载、删除等操作，如下演示一个存储卷完整生命周期的操作。

## 前提条件

创建存储卷之前必须先创建相应的存储类型，参考 [创建存储类型](#)。

## 创建存储卷

首先登录 KubeSphere 控制台，选择已有 [项目](#) 或新建项目，访问左侧菜单栏，点击 [存储卷](#) 进入列表页。作为集群管理员，可以查看当前集群下所有项目的存储卷以及挂载情况，而普通用户只能看到其所属项目的存储卷。

The screenshot shows the KubeSphere platform management interface. On the left, there's a sidebar with project navigation (Platform Management, Workstation, Application Template), user information (admin), and a search bar. The main area is titled '存储卷' (Storage Volumes) and displays a list of three volumes. Each volume entry includes a checkbox, name, status (准备就绪 - Ready), capacity (10Gi), access mode (ReadWriteOnce), and creation time. The volumes listed are: persistentvolumeclaim-wy9kbv(wordpress-volume), persistentvolumeclaim-9yq2tu(wordpress-pvc), and mysql-pvc-statefulset-m69tg3-0.

## 第一步：填写基本信息

点击存储卷列表页的 **创建存储卷** 按钮进入创建存储卷界面，填写存储卷基本信息，完成后点下一步：

This screenshot shows the 'Create Storage Volume' form. At the top, there are tabs for '基本信息' (Basic Information), '存储卷设置' (Storage Volume Settings), and '标签设置' (Label Settings). A 'Edit Mode' switch is also present. The '基本信息' tab is selected. It contains fields for '名称' (Name) with value 'pvc-demo', '项目' (Project) with value 'project-st75wr', and '别名' (Alias) with value '存储卷示例'. Below these fields is a note: '别名可以由任意字符组成。帮助您更好的区分资源，并支持中文名称。' (The alias can consist of any character. It helps you better distinguish resources and supports Chinese names.) On the right, there's a '描述信息' (Description) field with the value 'Demo'. At the bottom right of the form is a '下一步' (Next Step) button.

## 第二步：存储卷设置

存储设置中，选择存储卷的存储类型，按需填写存储卷的容量大小，存储卷大小和访问模式必须与存储类型和存储服务端能力相适应。各类型存储支持的访问模式参见 [Kubernetes 官方文档](#)。

**提示：**若需要选择集成多种 存储类型需要预先创建，详见 [创建存储类型](#)。

访问模式包括以下三种，注意，块存储仅支持单节点读写，若选择 QingCloud CSI 则访问只能选择 RWO。

- ReadWriteOnce — 可以被单个节点以读/写模式挂载。
- ReadOnlyMany — 可以被多个节点以只读模式挂载。
- ReadWriteMany — 可以被多个节点以读/写模式挂载。

创建存储卷

基本信息 存储卷设置 标签设置

### 存储卷设置

按需填写存储卷的容量大小，存储卷大小和访问模式必须与存储类型和存储服务端能力相适应，访问模式通常选择为 RWO。

存储类型  
csi-qingcloud

由 集群管理员 配置存储服务端参数，并按类型提供存储给用户使用。

访问模式  
 **ReadWriteOnce(RWO)**  
单个节点读写

存储卷容量  
0 512Gi 1024Gi 1.5Ti 2Ti **10 Gi**

**上一步** **下一步**

## 第三步：标签设置

为存储卷设置标签，可通过标签来识别、组织和查找资源对象，selector 可以根据标签的键值对来调度

资源。

## 第四步：查看存储卷

设置完成后点击创建，即可创建成功，刚创建的存储卷 `pvc-demo` 状态显示“创建中”，待其状态变为“准备就绪”就可以将其挂载至工作负载。

**注意：**若以 all-in-one 模式安装（存储类型为 Local Volume），创建存储卷后，在存储卷被挂载至工作负载前，存储卷状态将一直显示“创建中”，直至其被挂载至工作负载之后状态才显示“准备就绪”，这种情况是正常的，因为 Local Volume 的 [延迟绑定（delay volume binding）](#) 且 Local Volume 暂不支持动态配置。

The screenshot shows the KubeSphere storage volume management interface. At the top, there are summary statistics: 3 (已用) available, 0 allocated, and 0 remaining. Below this is a search bar and a toolbar with a 'Create' button. The main table lists two volumes:

名称	状态	容量	访问模式	挂载状态	创建时间
pvc-demo(存储卷示例) qingcloud-storageclass	● 创建中			未挂载	2018-11-22 11:17:09
persistentvolumeclaim-wy9kbv(wordpre... ss-volume) csi-qingcloud	● 准备就绪	10Gi	ReadWriteOnce	未挂载	2018-11-06 13:27:08

## 挂载存储卷

在创建工作负载时可以添加已创建的存储卷。参考如下，以创建一个 Wordpress 部署并挂载存储卷为例。

## 第一步：填写部署基本信息

选择已有项目，点击 **工作负载 → 部署 → 创建部署**，填写基本信息。

创建部署

基本信息      容器组模板      存储卷设置      标签设置      节点选择器

**基本信息**

您可以给部署起一个名字，以便在使用的时候容易区分。

名称 \*  最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目  将根据项目进行资源进行分组，可以按项目对资源进行查看管理

别名  别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

描述信息

## 第二步：配置容器组模板

配置容器组模板，以 Wordpress 为例，配置完后点击保存。

创建部署

基本信息      容器组模板      存储卷设置      标签设置      节点选择器

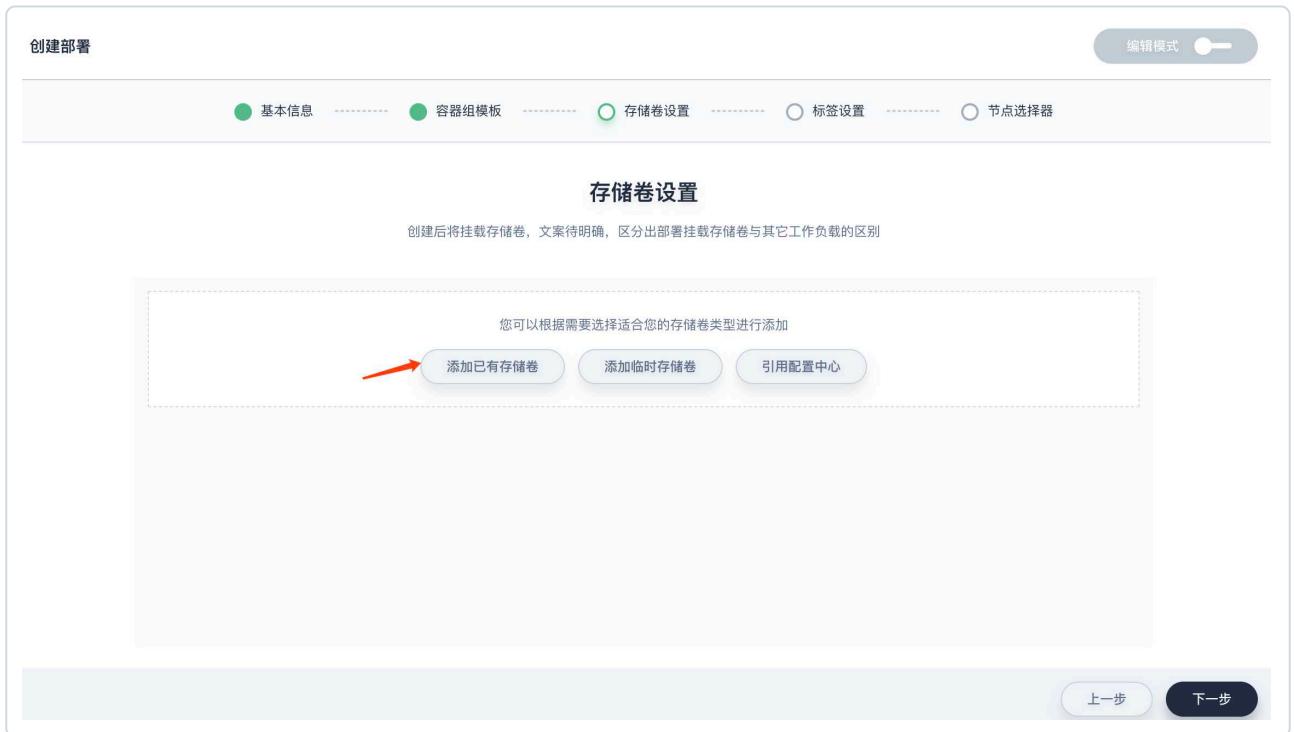
**添加容器**

容器名称 \*  最长253个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

镜像 \*  要从私有镜像仓库部署，需要先 创建镜像仓库，然后拉取镜像。

## 第三步：添加存储卷

1、点击 **添加已有存储卷**。



2、如下所示选择已有存储卷，此处选择我们在前面创建的 pvc-demo。

创建部署

编辑模式

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

◀ 挂载存储卷

名称	状态	容量	存储类型	访问模式
pvc-demo(存储卷示例)	闲置	容量: 10Gi	存储类型: qingcloud-storageclass	访问模式: ReadWriteOnce
persistentvolumeclaim-wy9kbv(wordpress-volume)	闲置	容量: 10Gi	存储类型: csi-qingcloud	访问模式: ReadWriteOnce
persistentvolumeclaim-9yq2tu(wordpress-pvc)	已使用	容量: 10Gi	存储类型: csi-qingcloud	访问模式: ReadWriteOnce

3、填写读写方式和挂载路径即可使用存储卷，选择 读写，路径填写 `/var/www/html`。

创建部署

编辑模式

基本信息 容器组模板 存储卷设置 标签设置 节点选择器

◀ 挂载存储卷 > 设置挂载路径

pvc-demo(存储卷示例) 闲置  
容量: 10Gi  
存储类型: qingcloud-storageclass  
访问模式: ReadWriteOnce

wordpress 读写 /var/www/html

上一步 保存



## 第四步：标签设置

标签设置为 `app: wordpress`，节点选择器暂不作配置，点击创建。

## 查看挂载状态

待部署创建完成后，在存储卷列表中可以看到 `pvc-demo` 卷显示挂载状态为 **已挂载**。

项目 project-st75wr

概览 应用 工作负载 存储卷 网络与服务 配置中心 项目设置

存储卷

存储卷供用户创建工作负载使用，是将工作负载数据持久化的一种资源对象。

4 已用配额 ∞ 规划配额

输入查询条件进行过滤

名称	状态	容量	访问模式	挂载状态	创建时间
pvc-demo(存储卷示例)	准备就绪	10Gi	ReadWriteOnce	已挂载	2018-11-22 11:17:09

创建



## 卸载存储卷

注意，若需要删除存储卷，请确保存储卷挂载状态处于 **未挂载**。如果存储卷已挂载至工作负载，在删除前需要先在工作负载中卸载（删除）存储卷或删除工作负载，完成卸载操作。如下，将挂载在工作负载 Wordpress 的存储卷进行删除（卸载）操作。



## 删除存储卷

在项目下，左侧菜单栏点击 **存储卷**，如下所示 pvc-demo 存储卷的状态为 **未挂载**，说明可以被删除。勾选需在上一步卸载的存储卷 pvc-demo，点击 **删除** 即可。



# Local Volume 使用方法

[Local Volume](#) 表示挂载的本地存储设备，如磁盘、分区或目录，而 Local Volume 只能用作静态创建的 PersistentVolume，与 HostPath 卷相比，Local Volume 可以以持久的方式使用，而无需手动将 pod 调度到节点上，因为系统会通过查看 PersistentVolume 上的节点关联性来了解卷的节点约束。

## 创建 Local Volume

Local Volume 仅用于 [all-in-one](#) 单节点部署，也是单节点部署的默认存储类型，Installer 会预先创建 10 个可用的 10G PV 供使用，若存储空间不足时则需要参考如下步骤手动创建。

1、若 Local Volume 还不是默认的存储类型，可参考创建 Local Volume 的存储类型详细步骤如下：

- 1.1. 通过 `sc.yaml` 文件定义 Local Volume 的存储类型：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

- 1.2. 执行创建命令：

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

2、创建 Local Volume 文件夹：

- 登录宿主机，创建文件夹，以文件夹 `vol-test` 为例，执行以下命令：

```
sudo mkdir -p /mnt/disks/vol-test
```

3、创建 Local PV：

- 3.1. 通过 `pv.yaml` 文件定义 Local PV：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-local
spec:
  capacity:
    storage: 10Gi
  # volumeMode field requires BlockVolume Alpha feature gate to be enabled.
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local
  local:
    path: /mnt/disks/vol-test
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
          - key: node-role.kubernetes.io/master
            operator: Exists
```

- 3.2. 执行创建命令：

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

4、执行以下命令验证创建结果：

```
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM   STOR
pv-local   10Gi       RWO           Delete         Available   local   4s
```

上述工作完成后可在 KubeSphere 控制台创建存储卷，KubeSphere 控制台创建的存储卷容量不可大于预分配 PV 容量。

注：Local Volume 存储卷创建成功后为 Pending 属于正常状态，当创建工作负载调度 Pod 后存储卷状态即可变化为 Bound。

## 删除 Local Volume PV 和文件夹

若需要删除 Local Volume，则手动创建的 PersistentVolume 也需要手动清理和删除。

1. 删除 Local Volume PV：

```
$ kubectl delete pv pv-local
```

2. 删除 Local Volume 文件夹，此操作也会删除 vol-test 文件夹里内容：

```
$ sudo cd /mnt/disks  
$ sudo rm -rf vol-test
```

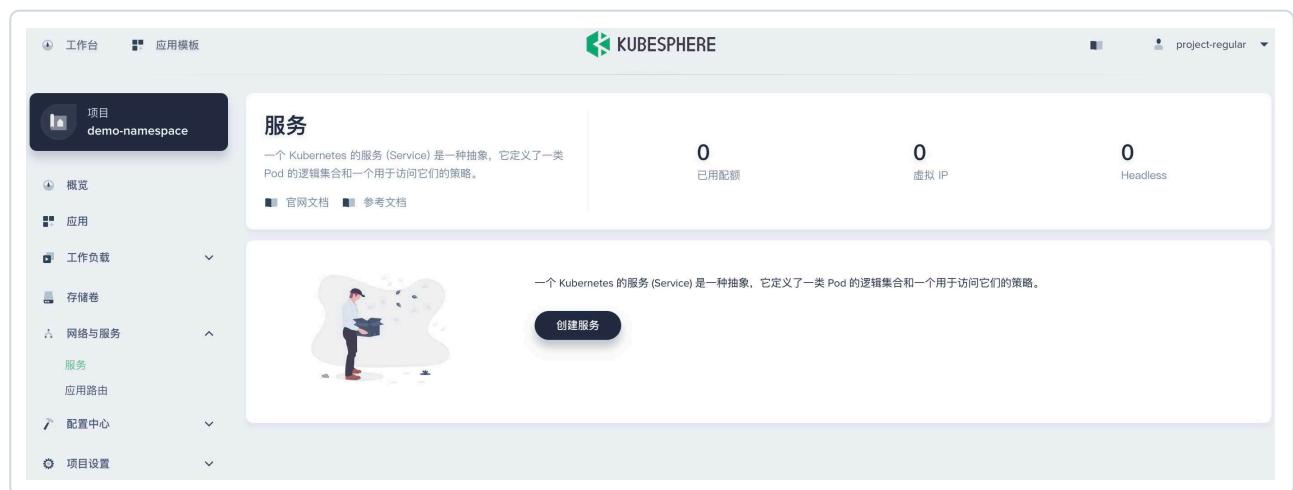
# 服务管理

一个 Kubernetes 的服务 (Service) 是一种抽象，它定义了一类 Pod 的逻辑集合和一个用于访问它们的策略 – 有的时候被称之为微服务，而在这个集合中的 Pod 的 IP 地址以及数量等都会发生动态变化，这个服务的客户端并不需要知道这些变化，也不需要自己来记录这个集合的 Pod 信息，这一切都是由抽象层 Service 来完成。

## 创建服务

登录 KubeSphere 控制台，在所属的企业空间中选择已有 **项目** 或新建项目，访问左侧菜单栏，点击 **网络与服务 → 服务** 进入服务列表页。

创建服务支持两种方式，**页面创建** 和 **编辑模式** 创建。若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式。左上角显示配置文件列表和导入导出按钮。其中导入 yaml 文件方式会自动将 yaml 文件内容填充到页面上，用户根据需要可以在页面上调整后再行创建，编辑模式可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件创建服务。以下主要介绍页面创建的方式。



## 第一步：填写基本信息

在服务列表页，点击 **创建** 按钮，填写基本信息：

- **名称**：为服务起一个简洁明了的名称，便于用户浏览和搜索。
- **别名**：帮助您更好的区分资源，并支持中文名称。

- **描述信息**: 详细介绍服务的特性，当用户想进一步了解该服务时，描述内容将变得尤为重要。

The screenshot shows the 'Create Service' page with the 'Basic Information' tab selected. At the top, there are tabs for '基本信息' (selected), 'Service Settings', 'Label Settings', and 'External Network Access'. A 'Edit Mode' switch is also present. The 'Basic Information' section contains fields for 'Name' (service-demo), 'Namespace' (demo-namespace), 'Alias' (服务), and 'Description' (This is a demo). There is also a note about the name length and format.

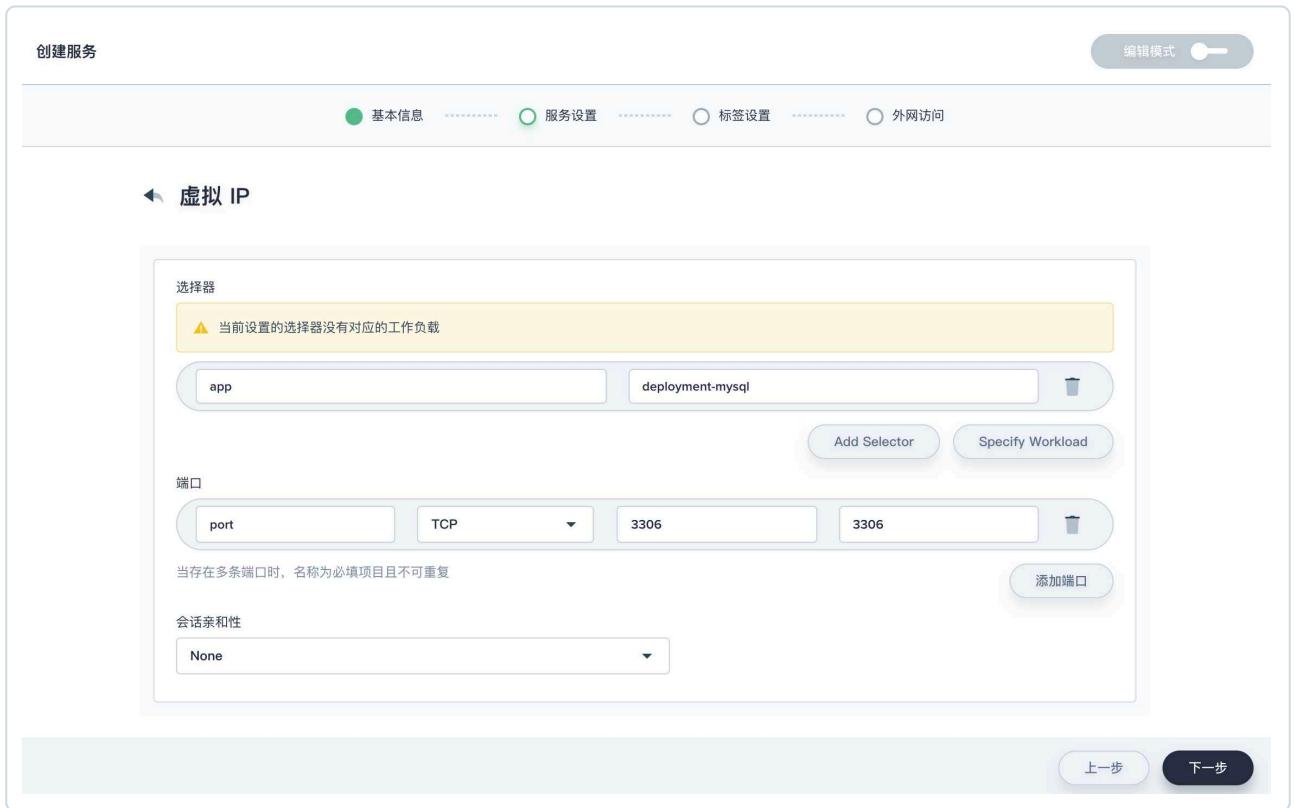
## 第二步：服务设置

### 2.1. 选择需要创建服务的类型，每种服务类型适合不同的场景：

- **VirtualIP**: 以集群为服务生成的集群内唯一的 IP 为基础，集群内部可以通过此 IP 来访问服务，集群外部可以通过 NodePort 和 LoadBalancer 方式来访问服务。此类型适合绝大多数服务。
- **Headless (selector)**: 集群不为服务生成 IP，集群内部通过服务的后端 Pod IP 直接访问服务。此类型适合后端异构的服务，比如需要区分主从的服务。
- **Headless (externalname)**: 给服务集群内设置一个 CNAME 记录，作为域名。

### 2.2. 若选择 VIP 或 Headless (selector)，需填写服务设置：

- **选择器**: 选择器来选择不同的后端，使用键值对 (Label Selector) 或 **指定工作负载** 可以选择多个部署。
- **端口**: 服务的端口号和目标端口，目标端口是对应的后端工作负载的端口号，如 MySQL 的 3306 端口。
- **会话亲和性**
  - **None**: 以 Round robin 的方式轮询后端的 Pods。
  - **ClientIP**: 来自同一个 IP 地址的访问请求都转发到同一个后端 Pod。



若选择 Headless (externalname)，通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容。

### 第三步：添加标签

标签设置页用于指定资源对应的一组或者多组标签 (Label)。Label 以键值对的形式附加到任何对象上，如 Pod, Service, Node 等，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。

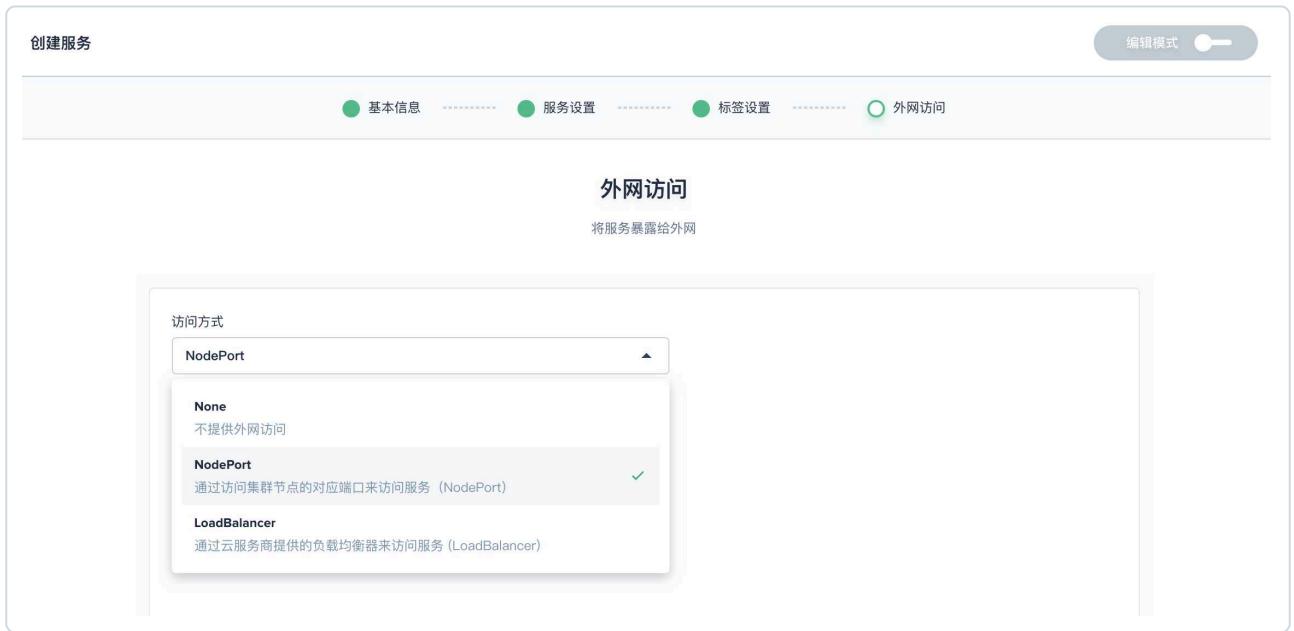


## 第四步：外网访问设置

为服务选择外网访问方式，LoadBalancer 的方式需要对应的负载均衡器插件来启用，如果未安装插件则无法使用。服务创建后支持修改外网访问方式。

- None: 只在集群内部访问服务，集群外部无法访问。
- NodePort: 集群外部可以通过访问集群节点的对应端口来访问服务，端口将由集群自动创建。
- LoadBalancer: 通过云服务商提供的负载均衡器来访问服务。

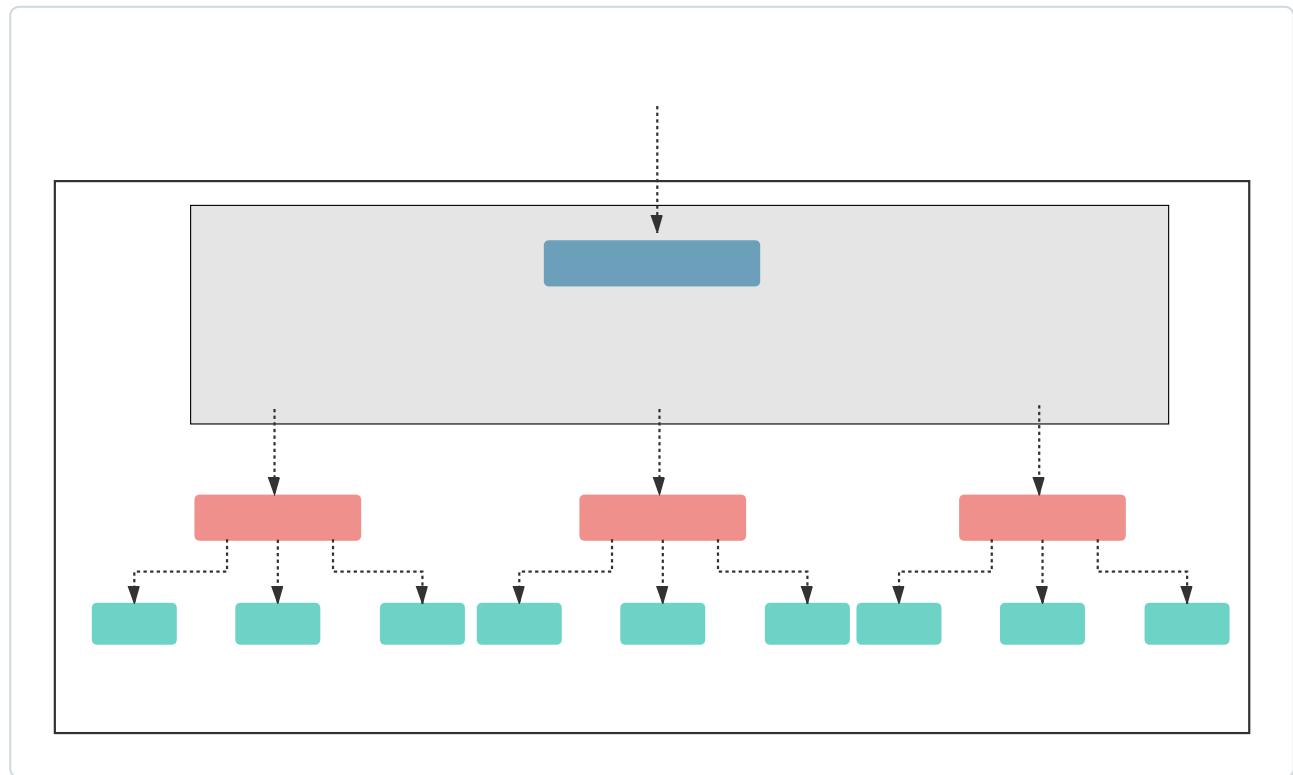
**注意：**由于使用 Load Balancer 需要安装与云服务商对接的 cloud-controller-manage 插件，目前 qingcloud-cloud-controller-manager 插件正在开发阶段，且即将上线，待上线后即可使用 Load Balancer 将内部的服务暴露给外网访问。



创建完成后即可在服务列表中查看成功创建的服务详情。

## 应用路由

应用路由 (Ingress) 是用来聚合集群内服务的方式，对应的是 Kubernetes 的 Ingress 资源，后端使用了 Nginx Controller 来处理具体规则。Ingress 可以给 service 提供集群外部访问的 URL、负载均衡、SSL termination、HTTP 路由等。



## 前提条件

- 请确保已预先创建了 [服务](#)，定义应用路由规则时需要选择后端的服务，Ingress 的流量被转发到它所匹配后端的服务。
- 若创建 https 协议的应用路由，请确保已预先创建了 https 证书相关的 [secret](#)。

## 创建应用路由

登录 KubeSphere 控制台，在所属的企业空间中选择已有 [项目](#) 或新建项目，访问左侧菜单栏，点击 [网络与服务](#) → [应用路由](#) 进入应用路由列表页。

创建应用路由支持两种方式，[页面创建](#) 和 [编辑模式创建](#)。若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式，可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件创建服务，以下主要介绍页面创建的方式。

## 第一步：填写基本信息

在应用路由列表页，点击 **创建** 按钮，填写基本信息：

- **名称**：为应用路由起一个简洁明了的名称，便于用户浏览和搜索。
- **别名**：帮助您更好的区分资源，并支持中文名称。
- **描述信息**：详细介绍应用路由的特性，当用户想进一步了解该应用路由时，描述内容将变得尤为重要。

The screenshot shows the 'Create Application Ingress' interface. At the top, there is a header 'Create Application Ingress' and a 'Edit Mode' switch. Below the header, there are four tabs: 'Basic Information' (selected), 'Route Rule', 'Annotation', and 'Label Setting'. The 'Basic Information' tab contains fields for 'Name' (ingress-demo), 'Project' (demo-namespace), 'Alias' (Application Routing Example), and 'Description' (Ingress demo). A note below the alias field states: 'Alias can be composed of any character, helping you better distinguish resources and support Chinese names.' A note below the project field states: 'Will manage resources based on projects, allowing you to view and manage resources by project.'

## 第二步：填写应用路由规则

点击 **添加路由规则**，如下图所示。

- **Hostname**：应用规则的访问域名，最终使用此域名来访问对应的服务。如果访问入口是以 NodePort 的方式启用，需要保证 Host 能够在客户端正确解析到集群工作节点上；如果是以 LoadBalancer 方式启用，需要保证 Host 正确解析到负载均衡器的 IP 上。
- **协议**：支持 http 和 https 协议；

- http：一个 Host 配置项（Hostname）和一个 Path 列表，每个 Path 都关联一个后端服务，若使用 loadbalancer 将流量转发到 backend 之前，所有的入站请求都要先匹配 Host 和 Path。
- https：除了 http 所包含的配置，还需要 secret，在 secret 中可指定包含 TLS 私钥和证书来加密 Ingress，并且 TLS secret 中必须包含名为 tls.crt 和 tls.key 的密钥。
- Paths：应用规则的路径和对应的后端服务，端口需要填写成服务的端口。

设置完成后点击 **保存**，然后点击 **下一步**。

The screenshot shows the 'Create Application Rule' interface. At the top, there are tabs for '基本信息' (Basic Information) [selected], '路由规则' (Route Rules), '注解' (Annotations), and '标签设置' (Label Settings). Below the tabs, the 'Route Rules' section is active, titled '设置路由规则' (Set Route Rule). It includes fields for 'HostName' (demo.nip.io), '协议' (Protocol) set to 'http', and a 'Paths' section where a path entry '/service-demo:80' is listed. A '添加 Path' (Add Path) button is located at the bottom right of the paths section.

### 第三步：添加注解

为应用规则添加注解，annotation 和 label 一样都是 key/value 键值对映射结构，例如添加以下一条注解，表示将 /path 路径重定向到后端服务能够识别的根路径 / 上面。重定向注解的作用是使应用路由以根路径转发到后端，避免因访问路径错误配置而导致页面返回 404 错误。

```
nginx.ingress.kubernetes.io/rewrite-target: /
```

然后点击 **下一步**。

### 第四步：添加标签

标签设置页用于指定资源对应的一组或者多组标签（Label）。Label 以键值对的形式附加到任何对象上，

定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。

设置完成后点击 **创建**。

## 访问应用路由

应用路由创建完成后，确保设置的域名可以解析到外网访问入口的 IP 地址，即可使用域名访问。如在私有环境中，可以使用修改本地 hosts 文件的方式来使用应用路由。例如，

设置的域名	路径	外网访问入口方式	端口/IP	集群工作节点IP
demo.kubesphere.io	/	NodePort	32586,31920	192.168.0.4,192.168.0.3,192.168.0.2
demo2.kubesphere.io	/	LoadBalancer	139.198.1.1	192.168.0.4,192.168.0.3,192.168.0.2

如上表格，创建了两条应用路由规则，分别使用 NodePort 和 LoadBalancer 方式访问入口。

### NodePort

对于外网访问方式设置的 NodePort，如果是在私有环境下，我们可以直接在主机的 hosts 文件中添加记录来使域名解析到对应的 IP。例如，对于 `demo.kubesphere.io`，我们添加如下记录：

```
192.168.0.4 demo.kubesphere.io
```

需要保证客户端与集群工作节点 192.168.0.4 网络可通，可以使用其它工作节点的 IP，只要客户端和工作节点网络是通的，设置完之后，在浏览器中使用域名和网关的端口号

`http://demo.kubesphere.io:32586` 即可访问。(此示例中用的是第一个端口 32586，它对应的 HTTP 协议的 80 端口，目前仅支持 HTTP 协议，将在下个版本中支持 HTTPS 协议。)

### LoadBalancer

如果外网访问方式设置的是 LoadBalancer，对于 `demo2.kubesphere.io`，除了参考以上方式在 hosts 文件中添加记录之外，还可以使用 [nip.io](#) 作为应用路由的域名解析服务。[nip.io](#) 是一个免费的域名解析服务，可以将符合下列格式的域名解析对应的 ip，可用来作为应用路由的解析服务，省去配置本地 hosts 文件的步骤。

## 格式

```
10.0.0.1.nip.io maps to 10.0.0.1  
app.10.0.0.1.nip.io maps to 10.0.0.1  
customer1.app.10.0.0.1.nip.io maps to 10.0.0.1  
customer2.app.10.0.0.1.nip.io maps to 10.0.0.1  
otherapp.10.0.0.1.nip.io maps to 10.0.0.1
```

例如，应用路由的网关公网 IP 地址为 139.198.121.154，在创建应用路由时，Hostname 一栏填写为 `demo2.kubesphere.139.198.121.154.nip.io`，其它保持原来的设置。

创建应用路由

基本信息 路由规则 注解 标签设置

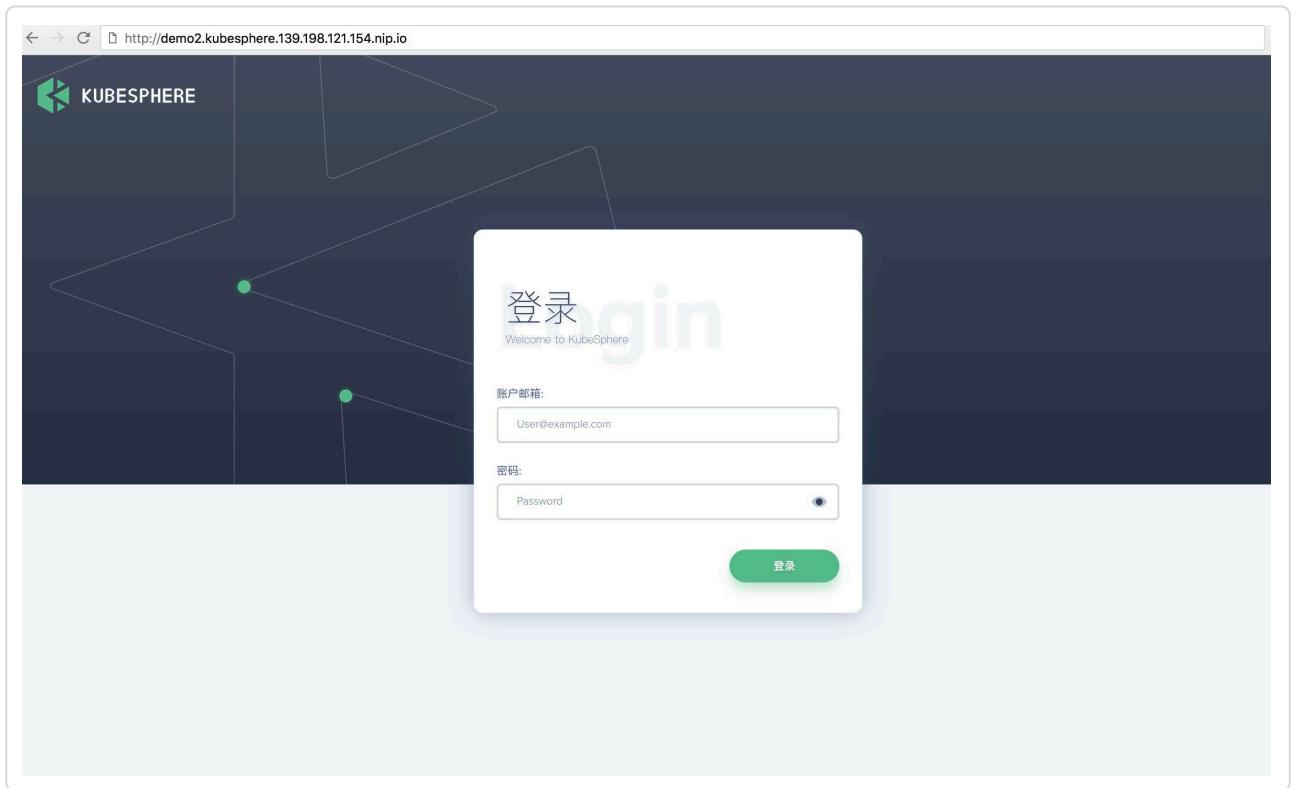
◀ Set Route Rule

HostName: demo2.kubesphere.139.198.121.154.nip.io

协议: http

Paths: / Please select a service 80 Add Path

创建完成后，直接使用 `http://demo2.kubesphere.139.198.121.154.nip.io`，即可访问对应的服务。



# 密钥

密钥 (Secret) 解决了密码、token、密钥等敏感数据的配置问题，配置密钥后不需要把这些敏感数据暴露到镜像或者工作负载 (Pod) 的 Spec 中。密钥可以在创建工作负载时以存储卷或者环境变量的方式使用。

## 创建 Secret

登录 KubeSphere 控制台，在所属的企业空间中选择已有 **项目** 或新建项目，访问左侧菜单栏，点击 **配置中心** → **密钥**，进入密钥列表页。

名称	类型	Config Number	创建时间
default-token-nhfnb	kubernetes.io/service-account-token	3	2018-12-10 13:35:48

创建密钥支持两种方式，**页面创建** 和 **编辑模式** 创建，以下主要介绍页面创建的方式。若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式，可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件创建密钥。

创建Secret

编辑模式

```
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: secret-mqvn9m
5   namespace: default
6   labels: {}
7   annotations:
8     displayName: secret-demo
9     desc: this is secret
10    type: Opaque
11
```

Yaml / Json

创建



## 第一步：填写基本信息

在服务列表页，点击 **创建** 按钮，填写基本信息：

- 名称：为密钥起一个简洁明了的名称，便于快速了解、浏览和搜索。
- 描述信息：详细介绍密钥的特性，当用户想进一步了解该密钥时，描述内容将变得尤为重要。

创建密钥

编辑模式

基本信息      密钥设置

### 基本信息

名称 \*

最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

别名

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

项目

将根据项目进行资源进行分组，可以按项目对资源进行查看管理

描述信息

## 第二步：Secret 设置

密钥设置中，支持添加以下四种类型：

- 默认 (Opaque): base64 编码格式的 Secret，用来存储密码、密钥等，这种类型应用得比较多。

例如：

```
data:  
  password: hello123  
  username: guest
```

- TLS (kubernetes.io/tls): 常用于保存 TLS 证书和私钥等信息，可用来加密 Ingress，TLS secret 中必须包含名为 tls.crt 和 tls.key 的密钥，以 Credential 和 Private Key 保存。例如：

```
apiVersion: v1  
data:  
  tls.crt: base64 encoded cert  
  tls.key: base64 encoded key  
kind: Secret  
metadata:  
  name: testsecret  
  namespace: default  
type: kubernetes.io/tls
```

- 镜像仓库密钥 (kubernetes.io/dockerconfigjson): 用来存储镜像仓库的认证信息，比如下面这类信息，详见 [镜像仓库](#)：
  - 仓库地址: dockerhub.qingcloud.com
  - 用户名: guest
  - 密码: 'guest'
  - 邮箱: 123@test.com
- 自定义：支持用户自己创建一种密钥类型 (type)，格式与默认 (Opaque) 类型相似，都是键值对的形式。



## 使用密钥

在当前的项目中创建好密钥之后，创建工作负载时可以通过两种方式使用该密钥：

- 以 Volume 方式，在添加存储卷时点击 **引入配置中心** 选择创建的密钥。
- 以环境变量方式，在添加容器镜像的高级设置中，点击 **引入配置中心** 选择创建的密钥。

关于如何使用密钥，建议参考 [示例一 - 部署 MySQL](#)。

## 配置

配置 (ConfigMap) 常用于存储工作负载所需的配置信息，许多应用程序会从配置文件、命令行参数或环境变量中读取配置信息。这些配置信息需要与 docker 镜像解耦，避免每修改一个配置需要重做一个镜像。配置给我们提供了向容器中注入配置信息的机制，可以被用来保存单个属性，也可以用来保存整个配置文件或者 JSON 二进制对象，在工作负载中作为文件或者环境变量使用。

登录 KubeSphere 控制台，在所属的企业空间中选择已有 **项目** 或新建项目，访问左侧菜单栏，点击 **配置中心** ➡ **配置**，进入配置列表页。



## 创建配置

创建配置支持两种方式，**页面创建** 和 **编辑模式** 创建，以下主要介绍页面创建的方式。若选择以编辑模式，可点击右上角编辑模式进入代码界面，支持 yaml 和 json 格式，可以方便习惯命令行操作的用户直接在页面上编辑 yaml 文件创建配置。

创建ConfigMap

编辑模式

```
1 ConfigMap:  
2   apiVersion: v1  
3   kind: ConfigMap  
4   metadata:  
5     name: configmap-i032em  
6     namespace: default  
7     labels: {}  
8     annotations:  
9       displayName: game-config  
10      desc: demo  
11   data:  
12     game.properties: 158 bytes  
13     ui.properties: 66 bytes  
14
```

Yaml / Json |

创建

## 第一步：填写基本信息

在服务列表页，点击 **创建** 按钮，填写基本信息：

- **名称**：为配置起一个简洁明了的名称，便于用户浏览和搜索。
- **描述信息**：详细介绍配置的特性，当用户想进一步了解该服务时，描述内容将变得尤为重要。

创建配置

编辑模式

基本信息 -----  配置设置

**基本信息**

名称 *	项目
configmap-demo	demo1
最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾	
别名	描述信息
示例配置	This is a demo
别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。	

## 第二步：配置设置

配置资源用来保存键值对配置数据，通常用来在工作负载（Pod）中设置环境变量的值（高级选项中）、在容器数据卷里创建 config 文件，这些数据可以在工作负载里使用。

例如：

```
data:  
  game.properties: 158 bytes  
  ui.properties: 86 bytes
```

创建配置

基本信息 配置设置

配置设置

添加参数

键	值
game.properties	158 bytes
ui.properties	86 bytes

点击 **创建**，查看配置创建结果：

The screenshot shows the KubeSphere configuration management interface. At the top, there are navigation links: 平台管理 (Platform Management), 工作台 (Workstation), and 应用模板 (Application Template). The central header displays the KubeSphere logo and the word "KUBESPHERE". A green success message "创建成功!" (Created successfully!) is visible. On the left, a sidebar menu includes: 项目 (Project) (selected), 概览 (Overview), 应用 (Application), 工作负载 (Workload), 存储卷 (Storage Volume), 网络与服务 (Network & Services), and 配置中心 (Configuration Center). The main content area is titled "配置" (Configuration) and describes it as a way to store通用的配置变量 (general configuration variables) for distributed systems. It shows two metrics: 0 已用配额 (Used Quota) and ∞ 规划配额 (Planned Quota). Below this is a search bar with placeholder text "输入查询条件进行过滤" (Filter by input query conditions). A table lists a single configuration entry: 名称 (Name): demo-configmap(示例配置), Config Field: game.properties,ui.properties, 创建时间 (Created Time): 2018-12-10 14:15:43. There are also icons for edit, delete, and more options.

## 使用配置

创建好配置之后，创建工作负载时可以通过两种方式使用：

- 以 Volume 方式，在添加存储卷时点击 引入配置中心 选择创建的配置。
- 以环境变量方式，在添加容器镜像的高级设置中，点击 引入配置中心 选择创建的配置。

关于如何使用配置，建议参考 [示例二 – 部署 Wordpress](#)。

# 镜像仓库

Docker 镜像是一个只读的模板，可用于部署容器服务，每个镜像有特定的唯一标识（即镜像名称：镜像 Tag）。例如：一个镜像可以包含一个完整的 Ubuntu 操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。而镜像仓库是集中存放镜像文件的场所，镜像仓库用于存放 Docker 镜像。

## 前提条件

添加镜像仓库需要预先创建企业空间和项目，若还未创建请参考 [管理员快速入门](#)。

## 添加镜像仓库

登录 KubeSphere 管理控制台，在已创建的项目中，左侧菜单栏中选择 **配置中心 → 密钥**，点击 **创建**。

## 添加 QingCloud 镜像仓库

QingCloud Docker Hub 基于 Docker 官方开源的 Docker Distribution 为用户提供 Docker 镜像集中存储和分发服务，请参考 [QingCloud 容器镜像仓库](#) 预先创建。若还未创建 QingCloud 镜像仓库，可以先参考文档添加一个示例仓库。

### 1、填写镜像仓库的基本信息

- 名称：为镜像仓库起一个简洁明了的名称，便于浏览和搜索。
- 别名：帮助您更好的区分资源，并支持中文名称。

- 描述信息：简单介绍镜像仓库的主要特性，让用户进一步了解该镜像仓库。

The screenshot shows the 'Create Key' interface with the 'Basic Information' tab selected. It includes fields for Name (dockerhub-qingcloud), Project (project-st75wr), Nickname (QingCloud 镜像仓库), and Description (QingCloud DockerHub Demo). A note below the nickname field specifies that it can be any character string to distinguish resources.

2、密钥设置中，类型选择 **镜像仓库密钥**，填写镜像仓库的登录信息。

- 仓库地址：用 QingCloud 镜像仓库地址 `dockerhub.qingcloud.com` 作为示例
- 用户名/密码：填写 `guest / guest`
- 邮箱：填写个人邮箱

The screenshot shows the 'Create Key' interface with the 'Key Settings' tab selected. The 'Type' dropdown is set to 'Image Repository Key'. It includes fields for Repository Address (dockerhub.qingcloud.com), Username (guest), and Password (\*\*\*\*\*).

3、点击 **创建**，即可查看创建结果。

项目  
project-st75wr(do...)

密钥

0 已用配额

∞ 规划配额

输入查询条件进行过滤

名称	类型	Config Number	创建时间
dockerhub-qingcloud(QingCloud 镜像仓库)	镜像仓库密钥	1	2018-12-10 12:04:31
default-token-9wq8w	kubernetes.io/service-account-token	3	2018-11-05 01:40:00

创建

## 添加 Docker Hub 镜像仓库

如果需要添加 [Docker Hub](#) 中的镜像仓库，请先确保已在 Docker Hub 注册过账号。添加步骤同上，仓库地址填写 [docker.io](#)，输入个人的 DockerHub 用户名和密码即可。

创建密钥

基本信息 密钥设置

密钥设置

类型  
镜像仓库密钥

仓库地址 \*  
docker.io

用户名 \*  
kubesphere

邮箱  
demo@kubesphere.io

密码 \*

## 添加 Harbor 镜像仓库

### Harbor 简介

[Harbor](#) 是一个用于存储和分发 Docker 镜像的企业级 Registry 服务器，通过添加一些企业必需的功能特性，例如安全、标识和管理等，扩展了开源 Docker Distribution，作为一个企业级私有 Registry 服务器，Harbor 提供了更好的性能和安全。注意，添加之前请确保已创建了 Harbor 镜像仓库服务端，以下详细介绍如何在 KubeSphere 中添加 Harbor 镜像仓库。

## 添加内置 Harbor 镜像仓库

KubeSphere Installer 集成了 **Harbor** 的 Helm Chart，内置的 **Harbor** 作为可选安装项，用户可以根据团队项目的需求来配置安装，仅需安装前在配置文件 `conf/vars.yml` 中简单配置即可，关于如何安装和使用内置的 Harbor 镜像仓库详见 [安装内置 Harbor](#)。

## 对接外部 Harbor 镜像仓库

根据 Harbor 镜像仓库的地址类型，需要分 http 和 https 两种认证方法：

### http

- 首先，需要修改集群中所有节点的 docker 配置。以 `http://139.198.16.232` 为例（用户操作时镜像仓库的地址应替换为您实际创建的仓库地址），在 `/etc/systemd/system/docker.service.d/docker-options.conf` 文件添加字段 `--insecure-registry=139.198.16.232`：

示例：

```
[Service]
Environment="DOCKER_OPTS=--registry-mirror=https://registry.docker-cn.com --insecure-registry=10.233.131.1
--iptables=false \
--insecure-registry=139.198.16.232"
```

- 添加完成以后，需要重载修改过的配置文件并重启 docker：

```
$ sudo systemctl daemon-reload
```

```
$ sudo systemctl restart docker
```

- 然后通过 KubeSphere 控制台，填写镜像仓库所需要的信息如仓库地址和用户认证，创建 Harbor 镜像仓库。

创建密钥

基本信息 密钥设置

密钥设置

类型  
镜像仓库密钥

可以选择也可以自定义一个密钥类型

仓库地址 \*  
http://139.198.16.232

用户名 \*  
admin

邮箱  
demo@kubesphere.io

密码 \*  
.....

https

- 对于 https 协议的镜像仓库，首先需要获取镜像仓库的证书，记为 `ca.crt`，以 <https://harbor.openpitrix.io> 这个镜像仓库的地址为例，对集群中的所有节点都需要执行以下操作：

```
$ sudo cp ca.crt /etc/docker/certs.d/harbor.openpitrix.io/ca.crt
```

- 如果还是报权限错误，针对不同的操作系统，需要执行以下操作：

#### UBUNTU

```
$ sudo cp ca.crt /usr/local/share/ca-certificates/harbor.openpitrix.io.ca.crt
```

```
$ sudo update-ca-certificates
```

#### RED HAT ENTERPRISE LINUX

```
$ sudo cp ca.crt /etc/pki/ca-trust/source/anchors/harbor.openpitrix.io.ca.crt
```

```
$ sudo update-ca-trust
```

2. 添加完成以后，需要重载修改过的配置文件并重启 docker (详情可参照 [docker官网](#)):

```
$ sudo systemctl systemctl daemon-reload
```

```
$ sudo systemctl restart docker
```

3. 然后通过 KubeSphere 控制台，填写镜像仓库所需要的信息如仓库地址和用户认证，参考添加 Docker Hub 的步骤，创建 Harbor 镜像仓库。

## 使用镜像仓库

以创建 Deployment 为例展示如何使用镜像仓库来拉取仓库中的镜像。比如 QingCloud 镜像仓库中有 mysql:5.6 的 docker 镜像。创建 Deployment 时，在容器组模板中需要选择镜像仓库，镜像地址填写为 dockerhub.qingcloud.com/mysql:5.6，镜像地址的格式为 镜像仓库地址 / 镜像名称:tag，填写后创建完成即可使用该镜像仓库中的镜像。

The screenshot shows the 'Create Deployment' interface in KubeSphere. The top navigation bar has tabs for '基本信息' (selected), '容器组模板', '存储卷设置', '标签设置', and '节点选择器'. A 'Edit Mode' switch is on the right. Below the tabs, there's a 'Back' button and the title '添加容器'. Under 'Container' settings, 'Container Name' is set to 'mysql' and 'Image' is set to 'dockerhub.qingcloud.com/mysql:5.6'. A dropdown menu shows two options: 'dockerhub' (selected) and 'dockerhub-qingcloud'. At the bottom left is a 'Advanced Options' button.

## 基本信息

基本信息支持管理当前项目的信息、配额和资源默认请求。其中配额信息是针对项目级别的，用于限制项目内的资源使用上限。而资源默认请求是相对容器级别的，在创建工作负载添加容器组时，默认情况下容器的 CPU 和内存的 requests 和 limits 值将使用其设置的值。

The screenshot shows the KubeSphere project management interface for the project 'demo-namespace'. The top navigation bar displays the project name 'demo-namespace' and the creation time '2019-01-26 13:30:36'. On the left, a sidebar menu lists various project settings: Overview, Application, Workload, Storage, Network & Services, Configuration Center, and Project Settings. The 'Project Settings' section is expanded, showing '基本信息' (Basic Information) highlighted with a red arrow. The main content area is divided into two sections: '基本信息' (Basic Information) and '配额信息' (Quota Information). The '基本信息' section shows a summary: 'demo-workspace' (企业空间), '1 成员' (1 member), and '3 成员角色' (3 member roles). The '配额信息' section displays resource quotas for three resource types: Deployments, StatefulSets, and DaemonSets, all set to '无限制' (Unlimited). A '编辑配额' (Edit Quota) button is located at the top right of the quota table.

## 基本信息管理

点击“...”选择 编辑信息，即可修改该项目的别名和描述信息。

 编辑信息 X

---

项目名称

最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

别名

别名可以由任意字符组成，帮助您更好的区分资源，并支持中文名称。

管理员

可以指定项目内一个成员为管理员

描述信息

取消确定

## 配额信息管理

默认情况下，项目中没有设置资源的使用量上限。配额管理支持对多种类型的资源如工作负载、CPU、内存等资源设置上限，点击 **编辑配额**，即可设置各类资源如的使用上限。

/ 编辑项目配额 X

---

项目名称

部署

配额	100
----	-----

有状态副本集

配额	100
----	-----

守护进程集

配额	100
----	-----

任务

配额	100
----	-----

定时任务

配额	100
----	-----

## 资源默认请求

在创建项目时，管理员已为该项目设置了资源默认请求，它是相对容器级别的，若需要修改其资源的默认请求，可点击“...”选择编辑。

资源默认请求 (82865257-212b-11e9-ac6d-525444e73450) – Container

! CPU无单位时为核数, 1核 = 1000m

/ 编辑

默认最大使用资源

	100m	CPU
	200Mi	内存

默认最小使用资源

	10m	CPU
	10Mi	内存

( )

比如，在创建部署时，容器组模板的高级选项中，若不填 CPU 和内存的 requests 和 limits 值，那么这两项的资源请求和限制值默认是上图中设置的值。

The screenshot shows the 'Create Deployment' page in KubeSphere. At the top, there are tabs for '基本信息' (Basic Information), which is selected, and other tabs for '容器组模板' (Container Group Template), '存储卷设置' (Storage Volume Settings), '标签设置' (Label Settings), and '节点选择器' (Node Selector). A 'Edit Mode' switch is also present.

In the main area, under the heading '添加容器' (Add Container), there is a '容器名称' (Container Name) input field containing 'container-svt9gr'. Below it is a note: '最长 253 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾' (Up to 253 characters, must start and end with lowercase letters or digits, and can contain lowercase letters, digits, and hyphens). There is also a '镜像' (Image) input field with placeholder text '请选择镜像仓库或者输入一个公有镜像仓库地址' (Select image repository or enter a public image repository address).

Below these fields is a section for 'CPU' resources. It contains two sets of input fields: '最小使用' (Minimum Usage) and '最大使用' (Maximum Usage). The first set has a value of '10' and a unit dropdown 'm' (milli). The second set has a value of '100' and a unit dropdown 'm'. A note below says: '作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量 ≥ 容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m' (As a dependency for resource allocation during container scheduling. Only if the sum of allocatable CPU on the node is ≥ the container's minimum usage, will the container be scheduled to the node. Unit conversion rule: 1 core = 1000m).

Below the CPU section is a section for '内存' (Memory) resources. It also contains two sets of input fields: '最小使用' (Minimum Usage) and '最大使用' (Maximum Usage). The first set has a value of '10' and a unit dropdown 'Mi'. The second set has a value of '200' and a unit dropdown 'Mi'. A note below says: '作为容器调度时资源分配的判断依赖。只有当节点上可分配内存总量 ≥ 容器内存最小使用值时，才允许将容器调度到该节点' (As a dependency for resource allocation during container scheduling. Only if the sum of allocatable memory on the node is ≥ the container's minimum usage, will the container be scheduled to the node).

# 成员角色

用户的权限管理依赖角色定义，角色标识了用户的身份，定义了用户和可访问/操作的资源之间的关系。当 KubeSphere 预置角色不满足使用要求的时候，可以根据实际情况，为用户创建自定义角色，自定义角色最大的优势即对平台资源的细粒度管理，指定该角色拥有某些指定资源的何种权限。

## 创建角色

登录 KubeSphere 管理控制台，进入已创建的项目下，访问左侧菜单栏，选择 **项目设置 → 成员角色**。作为项目管理员，可以查看当前项目下所有角色信息。

点击 **创建** 按钮创建角色，填写基本信息和设置权限。

### 第一步：填写基本信息

- 名称：为角色起一个简洁明了的名称，便于用户快速了解该角色的意义。
- 描述信息：详细介绍角色的特性，当用户想进一步了解该角色时，描述内容将变得尤为重要。



创建工作角色

基本信息 权限设置

**基本信息**

名称 \*  
workload-operator  
最长 63 个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母或数字开头及结尾

项目  
demo-namespace  
将根据项目进行资源进行分组，可以按项目对资源进行查看管理

描述信息  
This is a demo

### 第二步：权限设置

权限设置中支持管理员自定义一个角色拥有 KubeSphere 平台资源的何种操作权限，勾选角色所需权限规则，比如对部署的查看、创建、编辑、横向伸缩等这类操作。

**注意：对于资源的删除权限，应谨慎授予。**

The screenshot shows the 'Create Member Role' interface. At the top, there are two tabs: 'Basic Information' (基本信息) and 'Permission Settings' (权限设置), with 'Permission Settings' being active. Below the tabs, there are several sections for configuring permissions:

- Project Management**: Includes 'View', 'Edit', and 'Delete' checkboxes.
- Member Management**: Includes 'View', 'Create', 'Edit', and 'Delete' checkboxes.
- Role Management**: Includes 'View', 'Create', 'Edit', and 'Delete' checkboxes.
- Deployment**: Includes 'View', 'Create', 'Edit', and 'Delete' checkboxes. Additionally, there are checkboxes for 'Horizontal Scaling' (横向伸缩) which are checked for Create, Edit, and Delete.
- Stateful Set**: Includes 'View', 'Create', 'Edit', and 'Delete' checkboxes. Additionally, there is a checkbox for 'Horizontal Scaling' (横向伸缩) which is checked for View.

## 查看角色详情

在角色列表页，点击某个角色，打开角色详情页，可以看到当前角色权限列表和授权用户。

The screenshot shows the 'Role Details' page for the 'workload-operator' role. The top navigation bar includes back, forward, and search functions, along with the path: default / roles / workload-operator / authorities. The main area is divided into two tabs: 'Permissions List' (权限列表) and 'Granted Users' (授权用户). The 'Permissions List' tab is active, displaying the following permissions:

Resource Type	Permissions
Deployment	View / Create / Edit / Horizontal Scaling
Stateful Set	View / Create / Edit / Delete / Horizontal Scaling
DaemonSet	View / Create / Edit / Delete

## 修改角色权限

进入角色详情页面，点击 **编辑信息** 按钮修改角色名称和描述信息。

## 删除角色

进入角色详情页面，点击 **删除** 按钮删除角色。注意，删除角色之前需要解绑和该角色相关的用户，使用中的角色无法删除。

# 项目成员

您可以邀请新的成员来协助您的项目，在邀请新成员之前，请确保已预先创建了用户和成员角色，并且该成员已被邀请进入了该项目所在的企业空间，才可以被邀请进入项目，详见 [账户管理](#) 和 [成员角色](#)。

## 邀请成员

登录 KubeSphere 管理控制台，选择已有的项目或新建项目，左侧菜单栏点击 **项目设置 → 项目成员**。

The screenshot shows the KubeSphere management console interface. On the left, there is a sidebar with various project management options like Overview, Applications, Workloads, Storage Volumes, Network & Services, Configuration Center, Project Settings, Basic Information, Member Roles, and External Network Access. The 'Member Roles' option is highlighted with a red arrow. The main area displays the 'demo-namespace' project details, including the owner 'admin', workspace 'demo-workspace', and creation time '2019-01-26 13:30:36'. Below this, a table lists the current members: 'admin' (active, last login 未登录) and 'project-admin' (active, last login 未登录). A prominent red arrow points to the 'Invite Member' button at the top right of the member list table.

点击 **邀请成员**，在弹窗中选择用户点击 **+**，并选择一个用户，即可邀请该成员加入项目。例如，以下截图中邀请 workspace 进入当前项目中，并为其授予 [成员角色](#) 中创建的 [workload-operator](#)

## 邀请成员到该项目

您可以邀请新的成员来协助您的项目

输入邮箱邀请项目成员

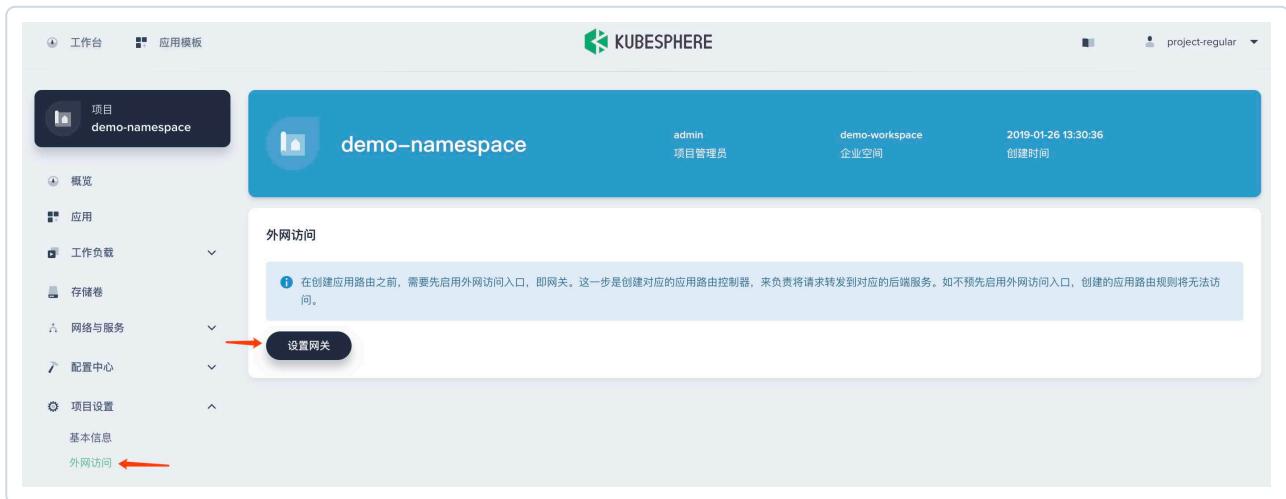
admin	✓
admin@kubesphere.io	
workspace	+
workspace@workspace.com	
test	请选择一个角色赋予该成员
test@test.com	workload-operator
calvin	+

Ok

至此，该用户就已经被邀请进了当前的项目，并拥有了 **workload-operator** 角色所定义的权限。

## 外网访问

- 在当前项目中，左侧菜单选择 **项目设置** → **外网访问**，点击 **设置网关**，即应用路由的网关入口，每个项目都有一个独立的网关入口。



- 在弹出的窗口选择网关的类型，支持以下两种访问方式：

- NodePort: 此方式网关可以通过工作节点对应的端口来访问服务。
- LoadBalancer: 此方式网关可以通过统一的一个外网 IP 来访问。

**注意：由于使用 Load Balancer 需要安装与云服务商对接的 cloud-controller-manage 插件，目前 qingcloud-cloud-controller-manager 插件正在开发阶段，且即将上线，待上线后即可使用 Load Balancer 将内部的服务暴露给外网访问。**

- 点击 **保存** 来创建网关，如下图选择的是 NodePort 的方式，左边节点端口生成的两个端口，分别是 HTTP 协议的端口和 HTTPS 协议的端口，外网可通过 EIP:NodePort 或 Hostname:NodePort 来访问服务，如：

- 通过 EIP 访问：

- <http://139.198.0.20:30798>
- <https://139.198.0.20:31279>

- 通过在应用路由规则中设置的 Hostname 访问：

- <http://demo.kubesphere.io:30798>

- <https://demo.kubesphere.io:31279>

The screenshot shows the KubeSphere web interface. On the left is a sidebar with navigation items: 工作台, 应用模板, 项目 (demo-namespace), 概览, 应用, 工作负载, 存储卷, 网络与服务, 配置中心, and 项目设置. Under 项目设置, there are 基本信息 and 外网访问. The main area displays the 'demo-namespace' project details. At the top right are user info (admin, demo-workspace) and creation time (2019-01-26 13:30:36). Below this is a section titled '外网访问' (External Network Access) which shows 'NodePort' access method and ports 'http:30798, https:31279' on '节点端口' (Node Port).

# DevOps 工程概述

由于软件开发复杂度的增高和更多的协同工作，团队开发成员间如何更好地在协同工作中确保软件开发和交付质量，逐渐成为研发过程中不可回避的问题。众所周知，敏捷开发 (Agile) 在业内日趋流行，团队如何在不断变化的需求中快速适应和保证软件质量就变得极其重要了。而 CI/CD 就是专门为解决上述需求的软件开发实践。CI/CD 要求每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件，减轻了软件发布时的压力。

## KubeSphere DevOps 特点

相较于易捷版，DevOps 工程是高级版独有的功能，针对企业实际的快速迭代和快速交付业务需求和场景，可以发现很多企业和 IT 团队都有持续集成和持续交付的需求。DevOps 工程提供 Jenkinsfile in & out of SCM 两种模式，从仓库 (SVN/Git/GitHub)、代码编译、镜像制作、镜像安全、推送到仓库、应用版本、到定时构建的端到端流水线设置，支持用户在开发、测试等环境下的端到端高效流水线能力，支持用户成员管理，同时提供完整的日志功能，记录 CI/CD 流水线的每个过程。

KubeSphere 高级版 v1.0.0 提供的 DevOps 具有以下功能：

- 开箱即用的 DevOps 功能，无需对 Jenkins 进行复杂的插件配置；
- 独立 DevOps 工程，提供访问可控、安全隔离的 CI/CD 操作空间；
- 兼容 Jenkinsfile in & out of SCM (Source Code Management) 两种模式；
- 可视化流水线编辑工具，降低 CI/CD 学习成本；
- 使用 KubeSphere 基于 Kubernetes 提供弹性、干净、可定制的构建环境。

KubeSphere 高级版下一个版本 v2.0.0 将增加如下新的功能：

- 支持 Source to Image (S2I)，快速交付容器镜像；
- 多语言代码静态检查，持续提升代码质量。

## 理解 KubeSphere DevOps

KubeSphere 的 DevOps 工程目前支持 GitHub、Git 和 SVN 这一类源代码管理工具，提供可视化的 CI/CD 流水线构建，或基于代码仓库已有的 Jenkinsfile 构建流水线。

软件开发的生命周期中，持续构建和发布是 IT 团队在日常工作中必不可少的步骤。但是，相比较传统的 Jenkins 集群一主多从的方式必然存在一些痛点：

- Master 一旦发生单点故障，那么整个 CI/CD 流水线就崩溃了；
- 资源分配不均衡，有的 Slave 要运行的 job 出现排队等待，而有的 Slave 处于空闲状态；
- 不同的 Slave 的配置环境可能不一样，需要完成不同语言的编译打包，这类差异化的配置进一步导致管理不便，维护起来也不是一件易事。

KubeSphere 的 CI/CD 是基于底层 Kubernetes 的动态 Jenkins Slave，也就是说 Jenkins Slave 具有动态伸缩的能力，能够根据任务的执行状态进行动态创建或自动注销释放资源。实际上，Jenkins Master 和 Jenkins Slave 以 Pod 形式运行在 KubeSphere 集群的 Node 上，Master 运行在其中一个节点，并且将其配置数据存储到一个 Volume 中，Slave 运行在各个节点上，并且它不是一直处于运行状态，它会按照需求动态的创建并自动删除。

上述的工作流程可以理解为：当 Jenkins Master 接受到 Build 请求时，会根据配置的 Label 动态创建运行在 Pod 中的 Jenkins Slave 并注册到 Master 上，当这些 Slave 运行完任务后，就会被注销，并且相关的 Pod 也会自动删除，恢复到最初状态。

所以，这种动态的 Jenkins Slave 优势就显而易见了：

- 动态伸缩，合理使用资源，每次运行任务时，会自动创建一个 Jenkins Slave，任务完成后，Slave 自动注销并删除容器，资源自动释放，而且 KubeSphere 会根据每个资源的使用情况，动态分配 Slave 到空闲的节点上创建，降低出现因某节点资源利用率高，还排队等待在该节点的情况；
- 扩展性好，当 KubeSphere 集群的资源严重不足而导致任务排队等待时，可以很容易的添加一个 KubeSphere Node 到集群中，从而实现扩展；
- 高可用，当 Jenkins Master 出现故障时，KubeSphere 会自动创建一个新的 Jenkins Master 容器，并且将 Volume 分配给新创建的容器，保证数据不丢失，从而达到集群服务高可用。

## 快速上手 CI/CD

我们提供了两个具有代表性的示例和文档，帮助您快速上手 CI/CD。

- [示例六 – Jenkinsfile in SCM](#)

本示例以文档和视频演示如何通过 GitHub 仓库中的 Jenkinsfile 来创建 CI/CD 流水线，最终将一个文档网站部署到 KubeSphere 集群中的开发环境和生产环境，并且能够通过公网访问。

- [示例七 – Jenkinsfile out of SCM](#)

本示例以文档和视频演示如何基于 [示例一 – Jenkinsfile in SCM](#), 以可视化的方式构建 CI/CD 流水线(包含示例一的前六个阶段), 最终将本文档网站部署到 KubeSphere 集群中的开发环境且能够通过公网访问。

## Jenkins Agent 说明

在 DevOps 工程中, KubeSphere 使用 Kubernetes Jenkins Agent 来执行具体的构建。Agent 部分指定整个 Pipeline 或特定阶段将在 Jenkins 环境中执行的位置, 具体取决于该 Agent 部分的放置位置。该部分必须在 Pipeline 块内的顶层或 Stage 内部定义, 详见 [Jenkins Agent 说明](#)。

## 添加代码仓库

在创建 Jenkinsfile in SCM 这类流水线时, 可参考 [添加代码仓库](#) 选择添加 Git 或 SVN 这类代码仓库。

## 设置自动触发扫描

在构建已有 SCM (Source Code Management) 的流水线中, 用户如果需要为流水线设置自动发现远程分支的变化, 以生成新的流水线并使其自动地重新运行, 可参考 [设置自动触发扫描](#)。

## 高级设置

KubeSphere 使用了 Configuration-as-Code 进行 Jenkins 的系统设置, 详见 [Jenkins 系统设置](#)。

## 流水线常见问题

本篇文档总结了流水线运行可能遇到的问题以及如何排错, 详见 [流水线常见问题](#)。

# 管理 DevOps 工程

## 创建 DevOps 工程

1、登录控制台，进入指定企业空间，点击左侧的 **DevOps 工程**，进入 DevOps 工程管理页面，页面中显示当前用户可查看的 DevOps 工程列表。



The screenshot shows the DevOps Engineering management interface. On the left, there's a sidebar with navigation items: 概览 (Overview), 项目管理 (Project Management), DevOps工程 (DevOps Engineering) (which is selected and highlighted in green), 镜像仓库 (Image Repository), and 企业空间管理 (Enterprise Space Management). The main content area has a title 'DevOps工程' with a subtitle explaining it corresponds to Jenkins' file夹. It includes a search bar, filter buttons, and a 'Create' button. Below is a table listing projects:

名称	项目管理员	创建时间
test-cicd	zpf	2018-11-02 18:03:10

2、点击右侧的创建按钮，创建一个新的工程，输入 DevOps 工程的基本信息。

- 名称：为创建的工程起一个简洁明了的名称，便于浏览和搜索。
- 描述信息：简单介绍 DevOps 工程的主要特性，帮助进一步了解该工程。

创建项目

## 基本信息

请输入 DevOps 工程的基本信息

名称 \*

显示名称

描述信息

**创建**

3、创建完成之后，进入了 DevOps 工程管理页面，可进行创建、编辑和查看当前工程下的 [流水线 \(Jenkins Pipeline\)](#) 和创建 [凭证 \(Credential\)](#)，凭证是包含了敏感数据的对象，例如用户名密码，SSH 密钥和一些 Token 等。点击左侧的 [基本信息](#)，可查看 DevOps 工程的信息和状态。

The screenshot shows the DevOps project management interface. On the left, there is a sidebar with navigation links: 'DevOps工程' (selected), '流水线', '工程管理', '基本信息' (highlighted in green), '凭证管理', '成员角色', and '工程成员'. The main content area displays the project details for 'demo-cicd':

- 项目名称:** demo-cicd
- 描述:** This is a demo
- 创建者:** admin
- 创建时间:** 2018-11-03 13:42:14

Below this, there is a section titled '基本信息' (Basic Information) with three cards:

- 企业空间:** sample (green icon)
- 成员:** 1 (blue icon)
- 成员角色:** 4 (orange icon)

4、另外，还支持管理工程成员和成员角色等操作，点击左侧的 [成员角色](#)，查看当前工程下已有哪些角色，平台为工程预置了几个常用的角色。

The screenshot shows the KubeSphere platform management interface. On the left, there's a sidebar with '平台管理' (Platform Management), '工作台' (Workstation), and '应用模板' (Application Template). Under '工程管理', there are sub-options: '基本信息' (Basic Information), '凭证管理' (Certificate Management), '成员角色' (Member Roles), and '工程成员' (Project Members). The main content area is titled 'hello-devops' and shows the following details:

- 名称: hello-devops
- 描述信息: (empty)
- admin 工程管理员 创建时间: 2018-10-20 20:04:02
- 成员角色 (Role):
  - owner: 项目的所有者, 可以进行项目的所有操作
  - maintainer: 项目的主要维护者, 可以进行项目内的凭证配置、pipeline配置等操作
  - developer: 项目的开发者, 可以进行pipeline的触发以及查看
  - reporter: 项目的观察者, 可以查看pipeline的运行情况

5、点击 **工程成员**，查看当前工程已有哪些用户。点击 **邀请成员** 按钮，邀请相应组织下的开发、测试或者运维人员进入此 DevOps 工程。在弹出的页面中搜索成员名，点击右侧的“+”号，可从企业空间的用户池中邀请成员加入当前的 DevOps 工程进行协同工作。注意，管理员将成员用户邀请到当前的 DevOps 工程后，一般来说，组内成员创建的资源 (pipeline、凭证等) 在组内是互相可见的。

The screenshot shows a modal dialog titled '邀请成员到该项目' (Invite Member to Project). It contains a search bar '输入邮箱邀请项目成员' (Input email to invite project member) and a list of users:

- admin: admin@kubesphere.io
- carlosfeng: carlosfeng@yunify.com
- zpf: pengfeizhou@yunify.com

Each user entry has a small '+' button to its right. An orange arrow points to the '+' button next to 'carlosfeng'. At the bottom right of the dialog is a large '邀请成员' (Invite Member) button.

## 编辑或删除工程

在 **工程管理 → 基本信息** 下，点击 “...” 按钮，即可看到编辑信息和删除等选项。支持 DevOps 工程的管理员编辑工程的基本信息如名称、指定管理员和描述信息，或删除工程。

The screenshot shows the KubeSphere interface for managing a DevOps project named "demo-devops".

**Left Sidebar:**

- 工作台
- 应用模板
- 流水线
- 工程管理

  - 基本信息 **1** (highlighted)
  - 凭证
  - 成员角色
  - 工程成员

**Top Header:**

KUBESPHERE project-admin

**Project Overview Card:**

- DevOps 工程 demo-devops
- demo-devops This is a demo
- project-admin 工程管理员
- demo-workspace 企业空间
- 2019-02-12 10:12:34 创建时间

**Right Panel - 基本信息 (Basic Information):**

- demo-workspace 企业空间
- 3 成员
- 4 成员角色
- 编辑信息
- 删除 **2** (highlighted)

**Annotations:**

- 选择「基本信息」
- 点击该按钮

# 流水线

[Jenkins Pipeline](#) (流水线) 表示应用从代码编译、测试、打包和部署的过程，KubeSphere 的流水线管理使用了业界常用的 [Jenkinsfile](#) 来表述一组 CI/CD 流程。Jenkinsfile 是一个文本文件，使用了 Jenkins 提供的 DSL (Domain-Specific Language) 语法。为降低学习 Jenkinsfile 语法的门槛，KubeSphere 提供了可视化编辑器，用户只需在页面上输入少量配置信息，接口自动组装完成 Jenkinsfile。也可直接编辑 Jenkinsfile，结合 KubeSphere 平台提供的一些功能插件，为更复杂的场景定制复杂流水线。

Pipeline 的几个常用概念：

- Stage: 阶段，一个 Pipeline 可以划分为若干个 Stage，每个 Stage 代表一组操作。注意，Stage 是一个逻辑分组的概念，可以跨多个 Node。
- Node: 节点，一个 Node 就是一个 Jenkins 节点，或者是 Master，或者是 Agent，是执行 Step 的具体运行时环境。
- Step: 步骤，Step 是最基本的操作单元，小到创建一个目录，大到构建一个 Docker 镜像，由各类 Jenkins Plugin 提供。

## 创建流水线 (Pipeline)

创建流水线支持 [Jenkinsfile in SCM \(Source Code Management\)](#) 和 [Jenkinsfile out of SCM \(Source Code Management\)](#)，请确保在创建流水线之前已创建了 DevOps 工程。

The screenshot shows the KubeSphere web interface. On the left, there's a sidebar with a 'DevOps 工程 demo-devops' section containing '流水线' (Pipeline), '工程管理' (Project Management), '基本信息' (Basic Information), '凭证' (Credentials), '成员角色' (Member Roles), and '工程成员' (Project Members). The main content area has a header '流水线' with a sub-header 'Pipeline' and a brief description: 'Pipeline 是一系列的插件集合，可以通过组合它们来实现持续集成和持续交付的功能。Pipeline DSL为我们提供了一个可扩展的工具集，让我们可以将简单到复杂的逻辑通过代码实现。'. Below this is a large button labeled '创建' (Create).

本节准备了两个示例，通过以下两种方式，分别说明如何将本文档网站构建一个 CI/CD 的 Jenkins 流水

线，并最终发布并部署到 KubeSphere 中。

- Jenkinsfile in SCM：创建此类型流水线时需要添加代码仓库且仓库中存在 Jenkinsfile，平台将扫描仓库中的 Jenkinsfile 自动构建流水线。本文档给出了一个示例和视频，参阅 [Jenkinsfile in SCM](#)。
- Jenkinsfile out of SCM：创建此类型的流水线无需添加代码仓库，支持可视化构建流水线，本文档给出了一个示例和视频，请参阅 [Jenkinsfile out of SCM](#)。

## 凭证管理

# 凭证

凭证 (Credential) 是包含了敏感数据的对象，例如用户名密码、SSH 密钥和一些 Token 等。流水线运行中，会与很多外部环境交互，如拉取代码，push/pull 镜像，SSH 连接至相关环境中执行脚本等，此过程中需提供一系列凭证，而这些凭证不应明文出现在流水线中，尤其是代码仓库管理 Jenkinsfile 文件的情况，用户需统一管理这些凭证，在流水线中只需要提供凭证的 ID。目前支持以下四种凭证类型：

- 账户凭证：常用于用户名和密码验证登录的代码仓库
- SSH：通过用户名、密码和私钥的方式
- 秘密文本：通过秘钥的方式连接
- kubeconfig：常用于配置跨集群认证，页面将自动获取当前 Kubernetes 集群的 kubeconfig 文件内容

## 创建凭证

1. 在左侧的工程管理菜单下。点击 **凭证管理**，进入凭证管理界面，界面会展示当前工程所需的所有可用凭证。



2. 点击创建按钮，创建一个用于拉取私人代码的 Git 代码仓库的用户名和密码。页面中会显示此凭证 ID，在流水线中需要用此 ID 获取凭证。

创建凭证

凭证 ID  
key-git-02641952

类型  
账户凭证

登录平台  
git

用户名  
root

密码  
.....

描述信息

取消 确定

This screenshot shows the 'Create Credential' dialog in Jenkins. The 'Type' is set to 'Account Credential'. The 'Platform' dropdown is set to 'git'. The 'Username' is 'root' and the 'Password' is redacted. There is a note field which is empty. At the bottom, there are 'Cancel' and 'Confirm' buttons.

3. 创建一个使用此凭证的流水线，其使用的模式是 Jenkinsfile Out of SCM，及不关联代码仓库（代码仓库中不含 Jenkinsfile）。

#### credentialcreatepipeline

4. Jenkinsfile 中 git 的定义如下，这里说明一下，此 Git 为一个 GitLab 上的私有仓库，需使用对应的用户名和密码才可拉取代码，在凭证 ID 一栏填入刚才的 ID，如下图：

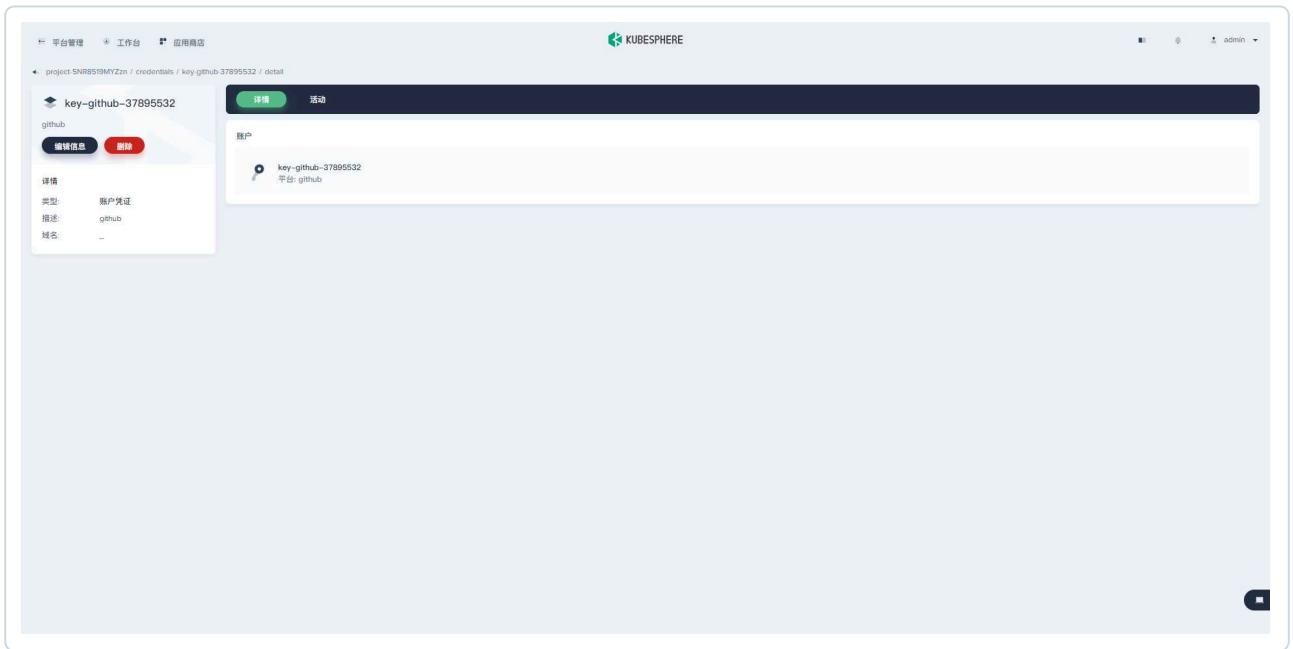


5. 运行此流水线，查看是否如期地正常运行。

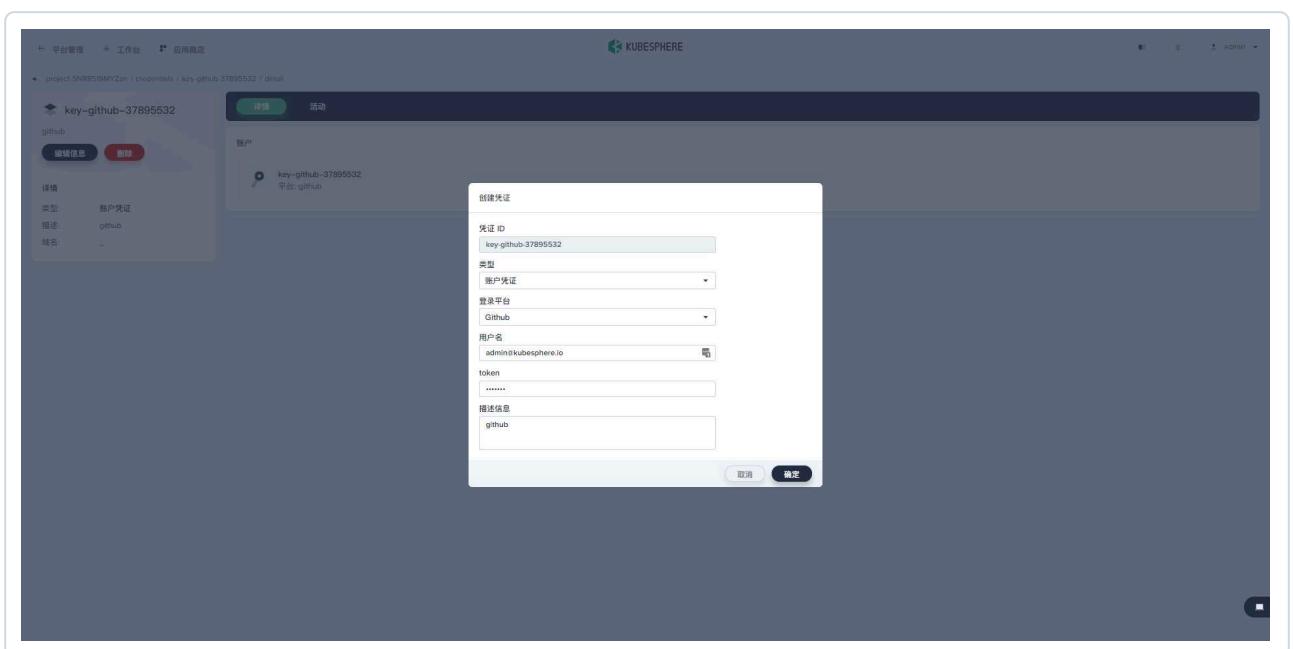
## 管理凭证

企业大型的工程往往需要与很多环境交互，会用到较多的凭证，KubeSphere 提供了凭证管理的页面，帮助用户统一集中管理这些凭证。

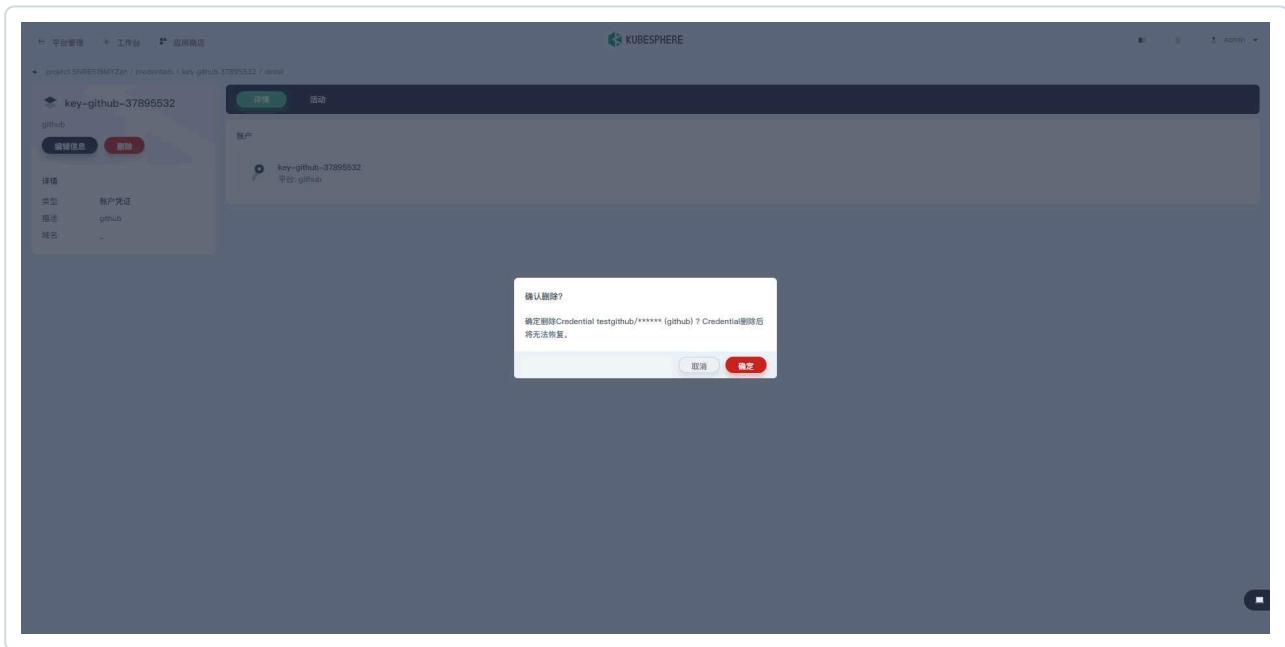
1. 在左侧的工程管理菜单下，点击 [凭证管理](#)，进入凭证管理界面，展示当前工程下的所有可用凭证。
2. 点击任意某一凭证，进入其详情页面。在此页面中可进行凭证的编辑，删除以及查看凭证的使用情况等操作。



3. 点击左侧编辑信息，弹出窗口，可对凭证进行修改。除凭证 ID，其他都可进行修改。



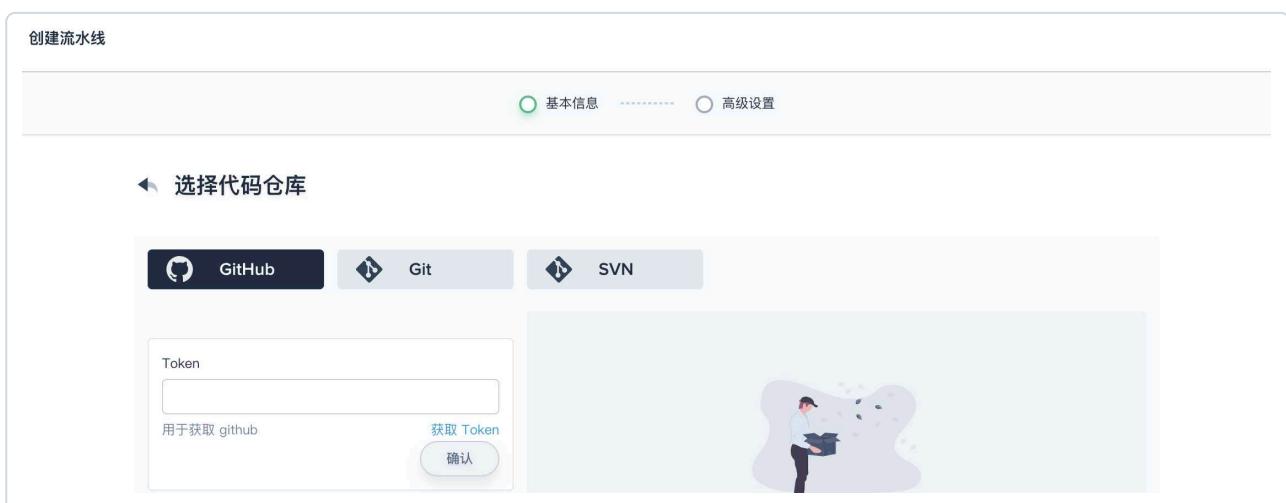
4. 点击删除按钮，可删除此凭证。



## 添加代码仓库

KubeSphere 的 DevOps 工程中，目前已支持了以下几种主流的源代码管理工具 (Source Code Management)，可以在创建 Jenkinsfile-in-SCM 这类流水线的高级设置添加这类源代码仓库，添加代码仓库之前需要预先创建一个账户凭证 (Credentials)。

- GitHub
- SVN
- Git



参考如下步骤添加代码仓库：

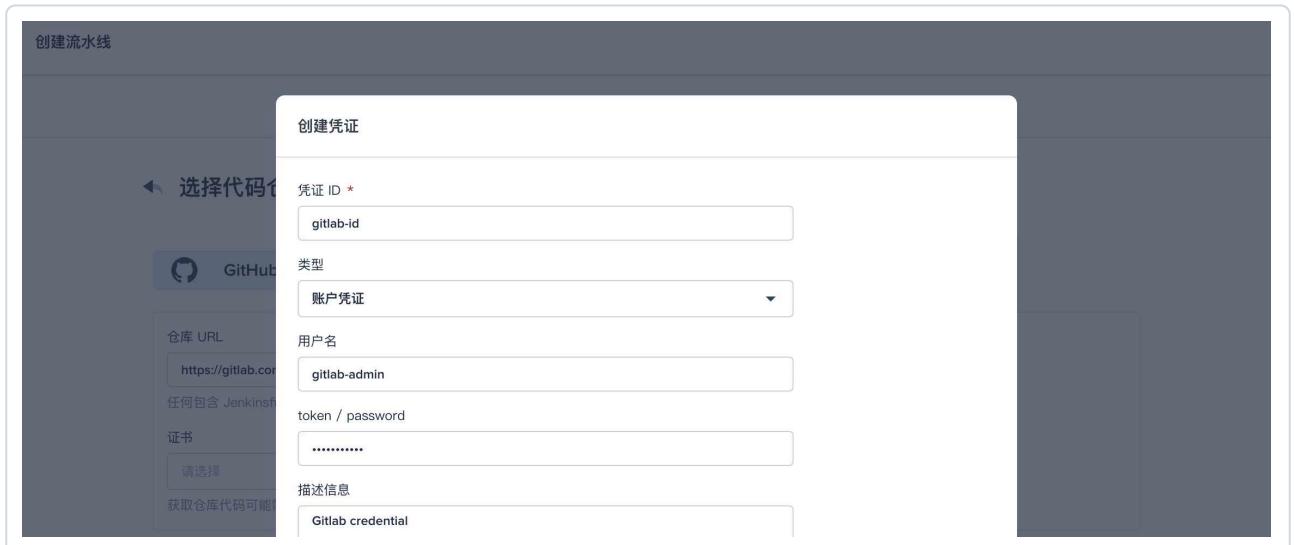
### 添加 GitHub

添加 GitHub 仓库已在示例六中以示例的方式给出，详见 [添加 GitHub](#)。

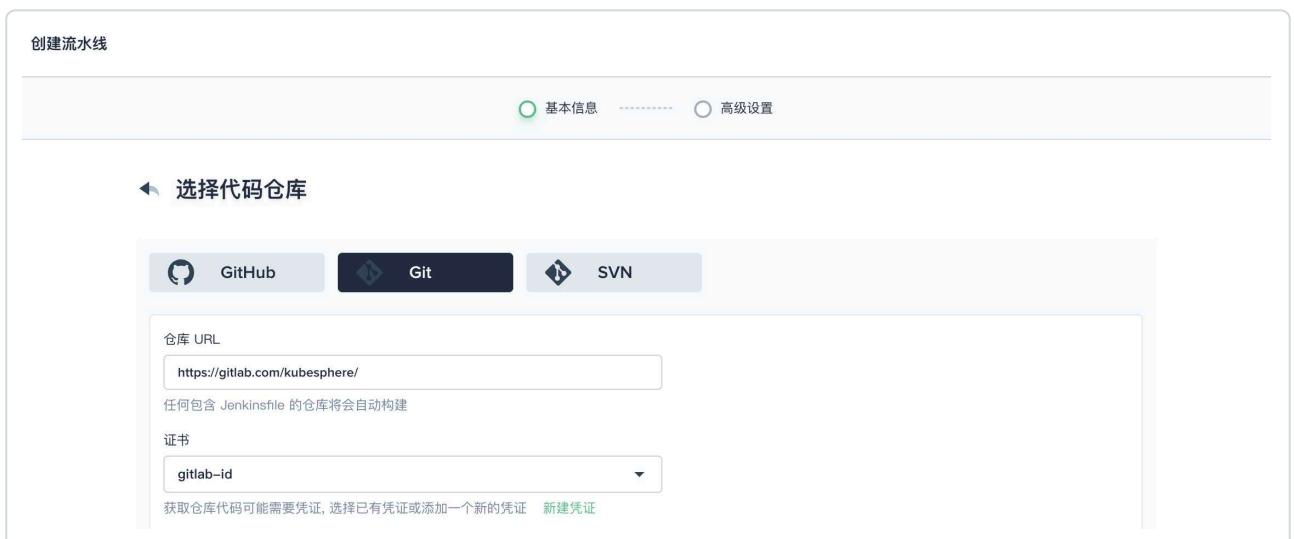
### 添加 Git

添加 Git 类型的代码仓库，原则是只要满足 Git 协议的仓库都支持添加，比如 GitLab、Gitee，添加这类代码仓库与添加 GitHub 步骤类似，需要预先为其创建凭证。在创建流水线的基本信息中，填写 Git 的仓库 URL 和证书 (credentials)，其中的凭证一般选择 **账户凭证** 并填写账户信息，若还未创建凭证可以点击 **新建凭证** 创建。

如下添加 Gitlab 账户凭证。



完成代码仓库的基本信息，证书选择上一步创建的 gitlab-id，点击保存。



## 添加 SVN

Subversion (SVN) 是一个开源的版本控制系统，它的版本控制与 Git 协议类型的代码仓库有很大区别，如下所示：

## SVN 仓库目录结构

- branch (分支): 分支开发和主线开发是可以同时并行开发, 分支常用于修复 bug 时使用。
- truck (主线 | 主分支): 可以理解为开发分支, 新功能的开发应放在主线中, 当各部分功能开发完后, 如需修改代码就用 branch。
- tag (标记): 类似 GitHub 中的 tag, 用于标记某个可用的版本, 可以标记已经上线发布的版本, 也可以标记正在测试的版本, 一般是只读的。

runzexia – Revision 6: /

- Jenkinsfile
- branches/
- tags/
- trunk/

---

Powered by [Apache Subversion](#) version 1.8.8 (r1568071).

添加 SVN 作为代码管理工具, 需预先填写 SVN 的远程仓库地址 (URL) 和证书 (credentials), 其中的凭证一般选择 **账户凭证** 并填写账户信息。流水线将扫描 SVN 上存在 Jenkinsfile 的分支然后触发该分支来运行流水线, 添加 SVN 详见以下信息:

- 类型
  - 单分支 SVN: 如果 Jenkinsfile 在根目录并且流水线在根目录运行, 就用单分支 SVN
  - SVN: 如果 Jenkinsfile 在根目录的文件夹中, 则选择该类型
- 远程仓库地址: 必填, 并且是需要公网或者内网能访问到的 SVN 仓库地址
- 证书: 同 Git, 需要添加账户凭证
- 包括分支: 和 GitHub 设置分支的发现策略类似, 即选择流水线将要扫描哪些分支 (目录)。如下将扫描这四个分支目录下所有文件
- 排除分支: 不扫描哪些分支 (目录)

GitHub    Git    SVN

类型: **svn**

远程仓库地址 \*

`http://www.svnchina.com/svn2/runzexia/`

证书 \*

**svn**

获取仓库代码可能需要凭证, 选择已有凭证或添加一个新的凭证 [新建凭证](#)

包括分支

`trunk,branches/*,tags/*,sandbox/*`

排除分支

## 设置自动触发扫描

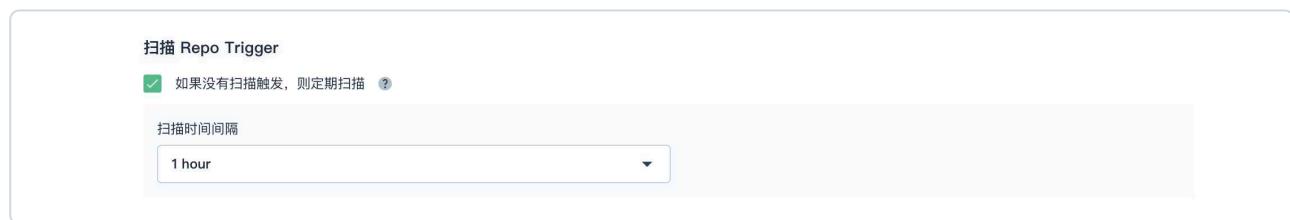
在含有 SCM (Source Code Management) 的 Pipeline 中，用户如果需要为流水线设置自动发现远程分支的变化，以生成新的 Pipeline 并使其自动地重新运行，可参考以下方式设置自动触发扫描。在 KubeSphere 中根据 SCM 类型的不同，也相应地提供了不同的方式进行触发。

### Github SCM

在 GitHub SCM 中，我们提供了两种方式可以让用户配置以实现自动扫描，我们推荐用户同时配置两个设置以达到最佳的效果：触发 Jenkins 自动扫描应该以 Webhook 为主，以在 KubeSphere 设置定期扫描为辅。

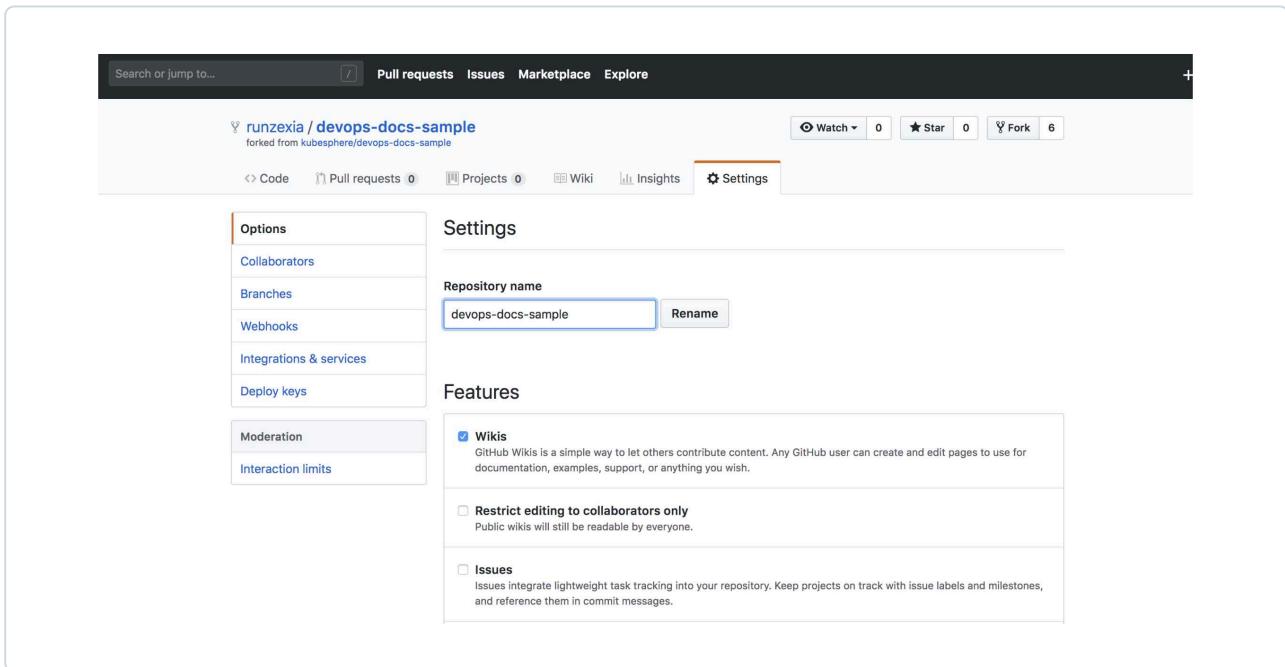
#### 第一步：设置定期扫描

Webhook 是一种高效的方式可以让我们发现远程仓库的变化，但是因为网络等问题，Webhook 消息可能不是总能被收到，因此推荐用户在 KubeSphere 创建 DevOps 工程的高级设置中，勾选 **如果没有扫描触发，则定期扫描**，并将时间间隔设置为可以容忍的最大时长（推荐 1 小时到 1 天之间）。

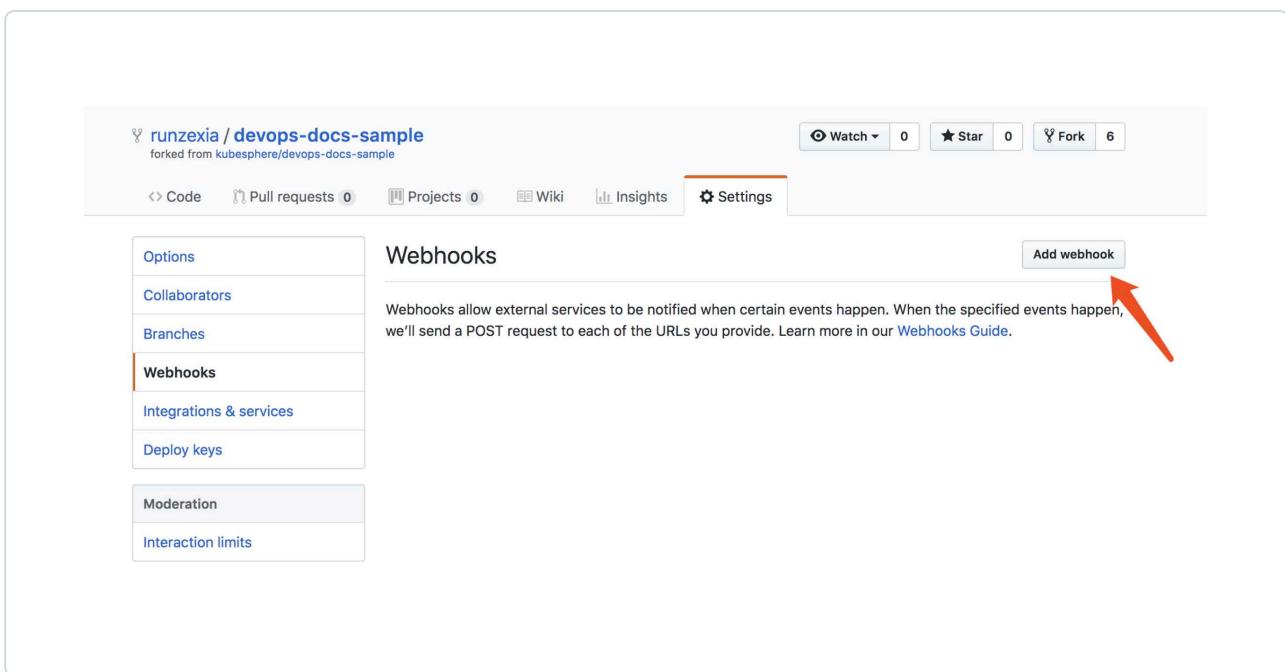


#### 第二步：设置 GitHub Webhook

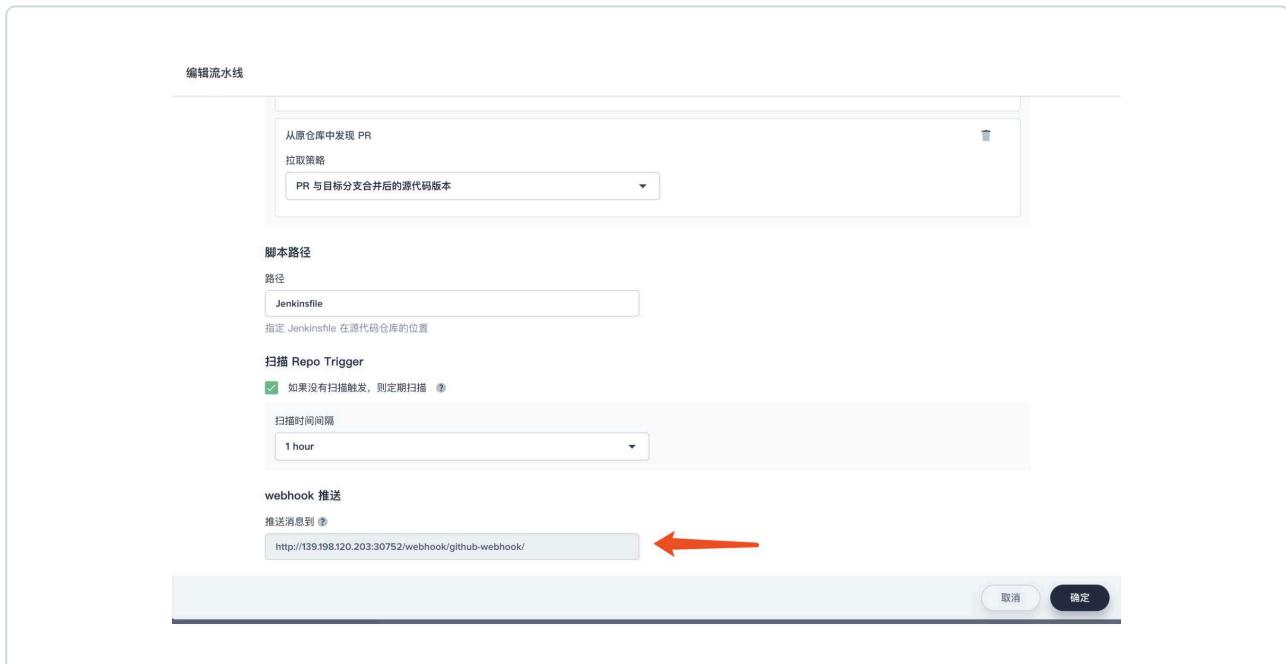
1、Webhook 需要用户自行到 GitHub 的 **Settings → Webhooks** 自行进行配置，并且需要 GitHub 能够访问到您安装的 KubeSphere 控制台地址。进入 GitHub，访问需要配置 Webhook 的仓库，比如当前的示例仓库 `devops-docs-sample`，选择 **Settings → Webhooks** 进行设置。



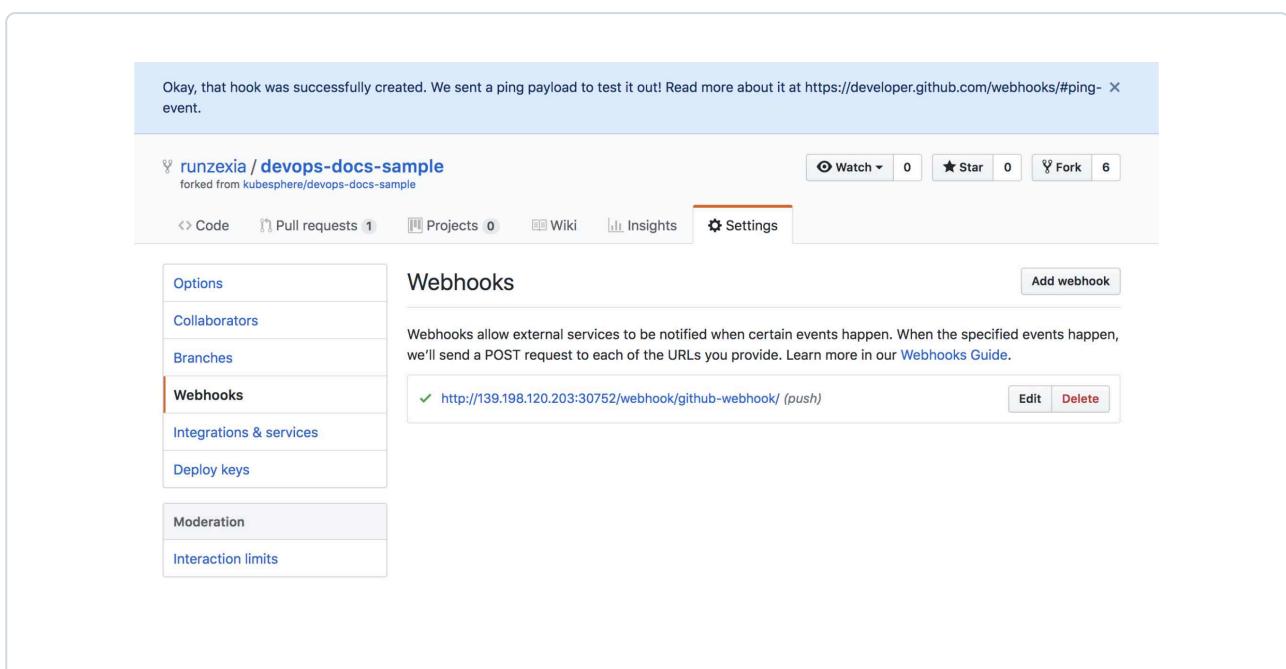
2、点击左侧 Webhooks，进入 Webhook 配置页面。点击 Add webhook 即可添加新的 Webhook。



注意，Payload 地址填写为关联的流水线 Webhook 推送 的默认地址，



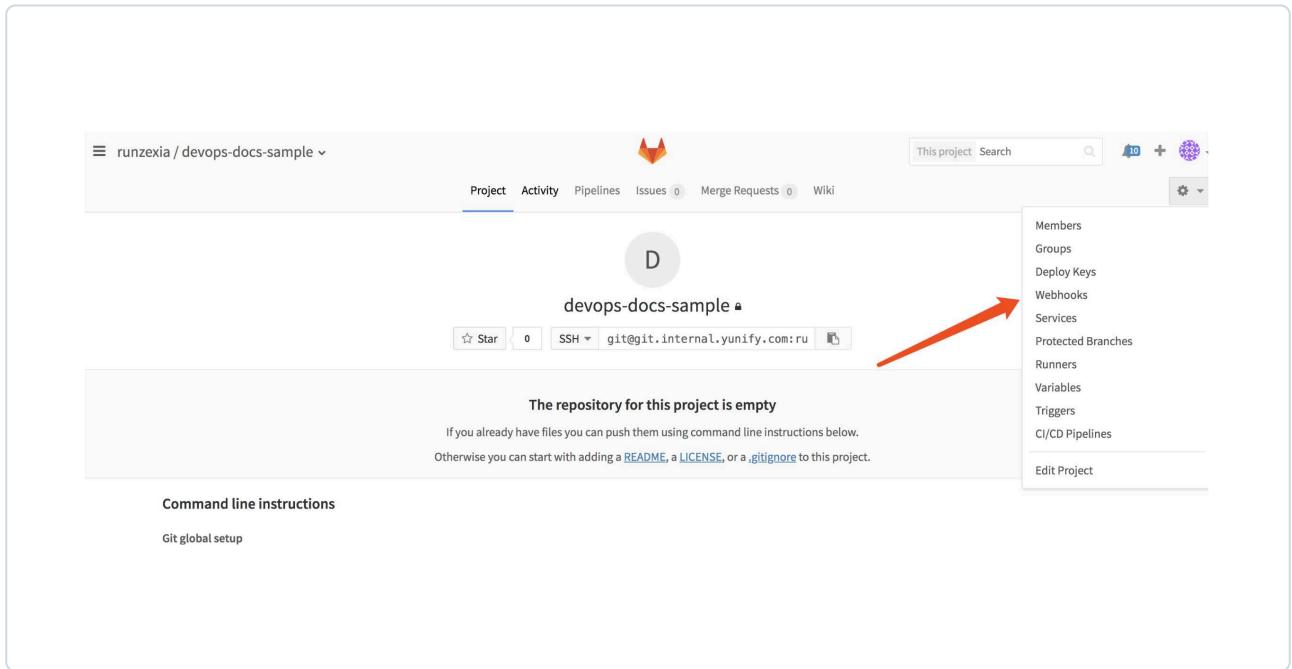
3、点击 **Add Webhook** 完成 Webhook 的添加，可以看到 Webhook 已经创建成功。



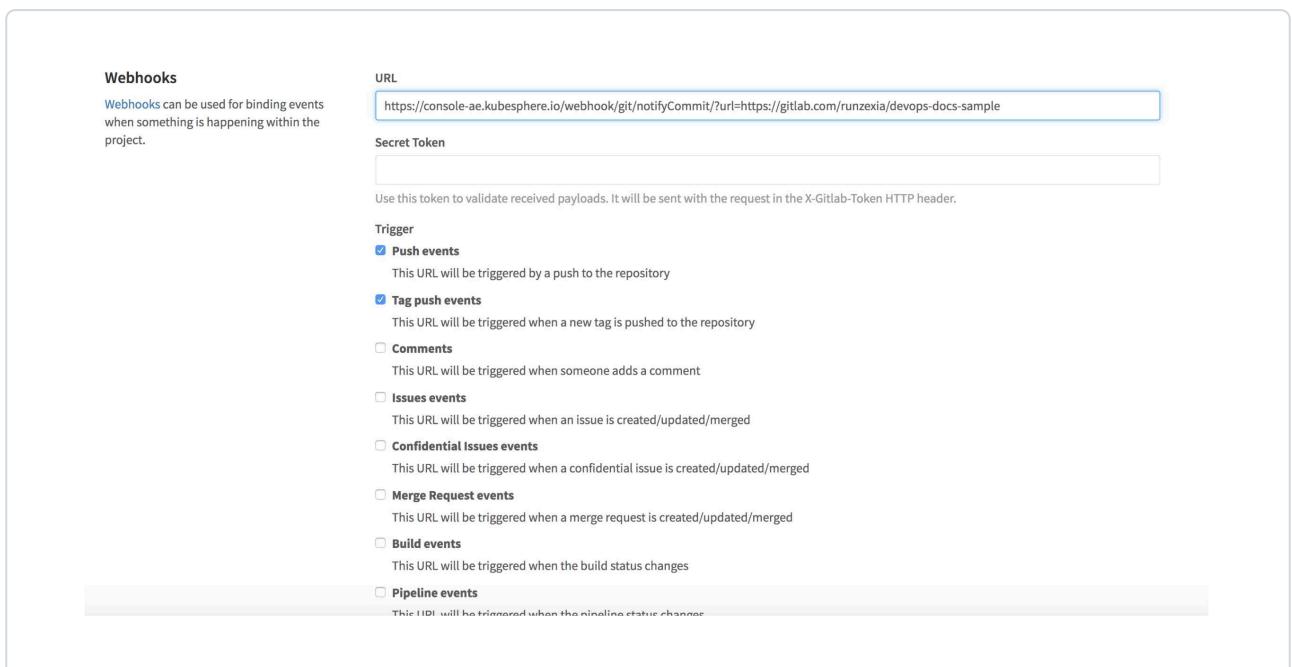
## Git SCM (以 GitLab 为例)

与 GitHub SCM 类似，Git SCM 也是以 Webhook 为主，定期扫描为辅的方式进行配置。下面我们将以 GitLab 为例配置 Webhook。

1、在项目设置按钮下点击 Webhooks 进入 Webhook 设置页面



2、同上，输入 Pipeline 配置的地址，最后点击页面最下方完成创建



## SVN SCM

在传统的 SVN 中不含 Webhook 的概念，因此推荐在 KubeSphere 设置时间间隔较短的定期扫描

来进行远程构建的触发，通常我们会将时间间隔设置为 15 分钟到 1 小时，团队可以根据自己的实际情况来设置定时扫描。

# Jenkins Agent 说明

## 简介

Agent 部分指定了整个流水线或特定的部分，将会在 Jenkins 环境中执行的位置，这取决于 Agent 区域的位置，该部分必须在 Pipeline 的顶层 或 Stage 中被定义。

## 内置的 podTemplate

在使用过程当中，每个 Pod 至少包含 jnlp 容器用于 Jenkins Master 与 Jenkins Agent 的通信。除此之外用户可以自行添加 podTemplate 当中的容器，以满足自己的需求。用户可以选择使用自己传入 Pod yaml 的形式来灵活的控制构建的运行环境，通过 `container` 指令可以进行容器的切换。

```
pipeline {
    agent {
        kubernetes {
            //cloud 'kubernetes'
            label 'mypod'
            yaml """
apiVersion: v1
kind: Pod
spec:
    containers:
        - name: maven
          image: maven:3.3.9-jdk-8-alpine
          command: ['cat']
          tty: true
        """
    }
}
stages {
    stage('Run maven') {
        steps {
            container('maven') {
                sh 'mvn --version'
            }
        }
    }
}
```

同时为了减少降低用户的使用成本，我们内置了一些 podTemplate，使用户可以避免 yaml 文件的编写。

在目前版本当中我们内置了 4 种类型的 podTemplate，`base`、`nodejs`、`maven`、`go`，并且在 Pod 中提供了隔离的 Docker 环境。

可以通过指定 Agent 的 label 使用内置的 podTempalte，例如要使用 nodejs 的 podTemplate，可以在创建 Pipeline 时指定 label 为 `nodejs`，如下给出示例。

## 代理

### 类型

node

agent部分指定整个Pipeline或特定阶段将在Jenkins环境中执行的位置，具体取决于该agent 部分的放置位置。该部分必须在pipeline块内的顶层定义，但stage级使用是可选的。

### label

nodejs

流水线或单个阶段的标签

```

pipeline {
    agent {
        node {
            label 'nodejs'
        }
    }

    stages {
        stage('nodejs hello') {
            steps {
                container('nodejs') {
                    sh 'yarn -v'
                    sh 'node -v'
                    sh 'docker version'
                    sh 'docker images'
                }
            }
        }
    }
}

```

## podTemplate base

名称	类型 / 版本
Jenkins Agent Label	base
Container Name	base
操作系统	centos-7
Docker	18.06.0
Helm	2.11.0
Kubectl	Stable release
内置工具	unzip、which、make、wget、zip、bzip2、git

## podTemplate nodejs

名称	类型 / 版本
Jenkins Agent Label	nodejs
Container Name	nodejs
操作系统	centos-7
Node	9.11.2
Yarn	1.3.2
Docker	18.06.0
Helm	2.11.0
Kubectl	stable release
内置工具	unzip、which、make、wget、zip、bzip2、git

## podTemplate maven

名称	类型 / 版本
Jenkins Agent Label	maven
Container Name	maven
操作系统	centos-7
Jdk	openjdk-1.8.0
Maven	3.5.3
Docker	18.06.0
Helm	2.11.0
Kubectl	stable release
内置工具	unzip、which、make、wget、zip、bzip2、git

## podTemplate go

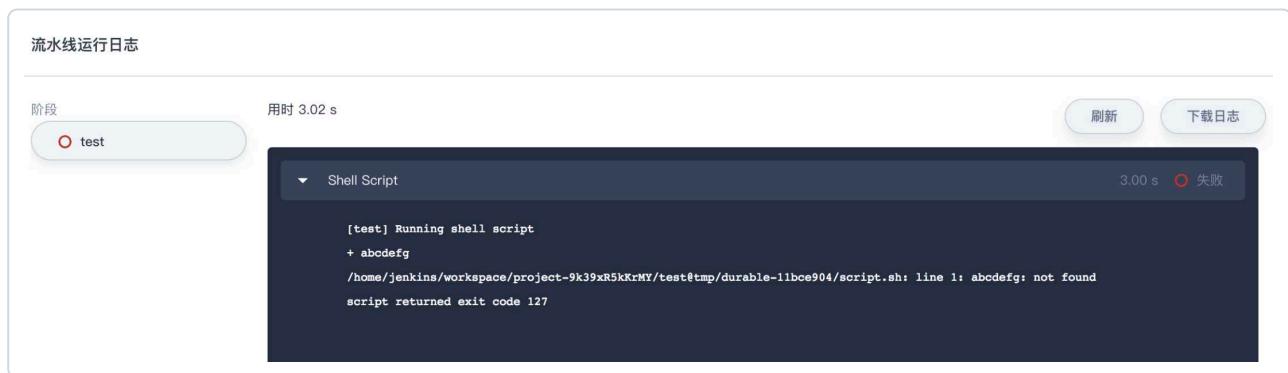
名称	类型 / 版本
Jenkins Agent Label	go
Container Name	go
操作系统	centos-7
Go	1.11
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
Docker	18.06.0
Helm	2.11.0
Kubectl	stable release
内置工具	unzip、which、make、wget、zip、bzip2、git

# 流水线常见问题

在触发 CI / CD 流水线时可能会因为一些其它因素造成流水线运行失败，比如凭证信息填写错误或网络问题等这类不确定的原因。在流水线运行失败时，用户应该如何排错呢？

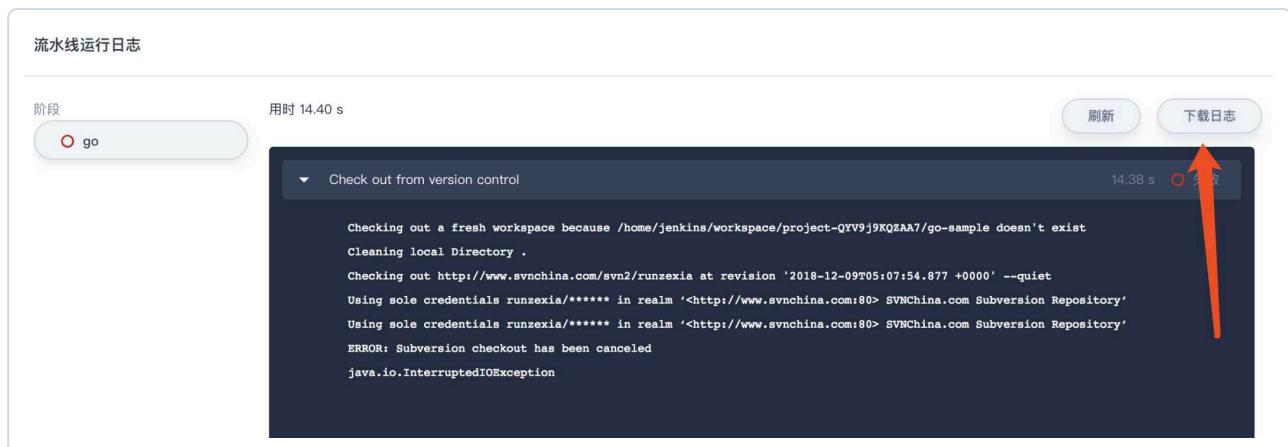
## 如何查看日志

在大多时间用户可以直接通过 Pipeline 的图形化页面查看日志完成排错，通常您执行命令的错误信息会在 Pipeline 的每个 Stage 的日志中。如下执行了一个不存在的 Shell 命令，错误信息位于每个 Stage 的日志中。



但是在部分情况下，错误的参数可能导致 Jenkins Pipeline 的异常中断，这时错误的日志在 Stage 日志中可能不会展示，需要用户使用流水线的下载日志下载整个 Pipeline 的完整日志。

例如下面这种情况，我们在使用 Pipeline Step 时传入了非法的参数，这种情况下部分 Step 可能会发生异常，但这种异常可能不会在 Stage 日志中展示。用户可以点击 [下载日志](#) 获取 Pipeline 的完整日志进行排错。



## 重新执行与运行之间的区别

重新执行将尽力还原 Pipeline 运行时所在的运行环境。尽管每次 Pipeline 运行完毕后整个 Agent 环境将被销毁，但是 Jenkins 将 Pipeline 运行时所在的 SCM 环境与环境变量、参数变量以及 Jenkinsfile 信息都进行存储，在重新运行时将加载上次运行的环境，因此提供了几乎一致的运行环境。运行将重新触发一次运行，将刷新所有环境，根据用户的配置生成全新的环境。**重新执行** 按钮位于运行图形化展示页的左侧。



## Jenkins 语言支持问题

KubeSphere 拥有很多母语为中文的用户，但是 Jenkins 目前对中文以及一些需要 UTF-8 字符集的语言支持度不是很好，因此强烈建议您在使用 KubeSphere Devops 时（如命名或添加参数）尽量使用英文。

## 其他错误

在运行 Pipeline 时，我们需要用户配置 credential 以获得推送到 DockerHub、Git 仓库的权限。需要注意的是 Git 仓库的密码是在 http 链接中直接进行了使用，若用户的凭证信息如密码中包含了 @，\$ 这类符号，可能在运行时无法识别而报错，当遇到这类情况需要用户在创建 credential 时对密码进行 urlencode 编码，可通过一些第三方网站进行转换（比如 <http://tool.chinaz.com/tools/urlencode.aspx>）。

## 在 Agent 中切换工作空间

目前 `dir` 或 `ws` 命令都不能在 Kubernetes Agent 中切换到 `/home/jenkins/` 外的目录，如果需要切换到 `/home/jenkins/` 外的目录并执行命令，您可以使用 `cd` 命令进行目录的切换。例如：`cd /usr/ && ls`。

这是 Jenkins Kubernetes Plugin 一个已知的问题，我们会跟进社区后续修复。

## 示例运行失败如何排错

快速入门中的示例六和示例七用到的 `devops-docs-sample` 代码仓库位于 GitHub 之中，并且 Pipeline 运行过程需要拉取项目所需的 npm 依赖，同时 sample 还将构建完成的镜像推往 DockerHub，这些仓库的服务端都在国外，因此该示例对执行网络环境有比较严格的要求。如果 `devops-docs-sample` 运行失败，同样可以通过查看 Pipeline 日志来获取完整的错误信息，以定位问题。

## 如何解决网络问题

对于网络错误，可以通过重新运行来尝试解决。如果多次重新运行后不能解决这个问题，可以通过将源代码仓库、Docker 镜像仓库转移到国内的方法进行解决。

国内许多 SCM 服务的提供商都提供一键导入 GitHub Repo 的功能，用户只需下载示例源代码的仓库，通过国内的 SCM 服务商来完成仓库的转移。

No description, website, or topics provided.

Branch: master ▾ New pull request

42 commits 1 branch 0 releases 2 contributors Apache-2.0

rayzhou2017 Update docs-sample-svc.yaml

__mocks_	init	
content	rm static, update yarn lock	
deploy	Update docs-sample-svc.yaml	
plugins	init	
src	init	23 days ago
.dockerignore	init	23 days ago
.gitignore	init	23 days ago
.prettierrc	init	23 days ago
.travis.yml	init	23 days ago
CNAME	init	23 days ago
Dockerfile	init	23 days ago
Jenkinsfile	change deploy production ns	25 minutes ago

Clone with HTTPS ⓘ Use SSH  
https://github.com/kubesphere/devops- ↗

Open in Desktop Download ZIP

对于自建的 SCM 服务，可以通过下面几个步骤来进行迁移。

- 1、在工作目录初始化一个 Git 仓库，并在自建 SCM 中使两者关联。
- 2、下载源代码仓库的 zip 压缩包，并将压缩包的内容解压到 Git 仓库目录中。
- 3、将例子代码提交并推送到自建 SCM 中。
- 4、创建 Git 类型的项目，并填入自建 SCM 仓库中的例子信息。
- 5、镜像仓库替换需要修改 Jenkinsfile 的中 docker build、docker tag、docker push 命令的镜像名称，以及示例仓库中 `/deploy` 的 yaml 文件中的镜像地址，用户可以按照自己的情况进行镜像仓库地址的修改。

## 内置 Maven agent 构建缓慢

在默认情况下，我们将 Maven agent 的 JVM 设置为 32 位以降低构建所造成的内存消耗。

在 32 位的 JVM 当中，最大堆大小不能超过 2G，对于一些大型项目来说会造成构建无法完成、构建缓

慢等情况。

可以在 Jenkinsfile 当中主动运行以下命令，将 agent 的 JVM 切换至 64 位：

```
$ alternatives --set java /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64/jre/bin/java  
$ alternatives --set javac /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64/bin/javac
```

## 常见问题

### 如何快速了解 KubeSphere

1、作为新手，如何快速了解 KubeSphere 的使用？

答：我们提供了多个快速入门的示例包括工作负载和 DevOps 工程，建议从 [快速入门](#) 入手，参考 [快速入门](#) 并实践和操作每一个示例。

### Multi–Node 安装配置相关问题

2、[Multi–Node 模式](#) 安装时，如果某些服务器的 Ubuntu 系统默认管理员用户为 `ubuntu`，若切换为 `root` 用户进行安装，应该如何配置和操作？

可通过命令 `sudo su` 切换为 `root` 用户后，在该节点查看是否能 ssh 连接到其他机器，如果 ssh 无法连接，则需要参考 `conf/hosts.ini` 的注释中 `non-root` 用户示例部分，如下面第二步 `hosts.ini` 配置示例所示，而最终执行安装脚本 `install.sh` 时建议以 `root` 用户执行安装。

第一步，查看是否能 ssh 连接到其他机器，若无法连接，则参考第二步配置示例。相反，如果 `root` 用户能够 ssh 成功连接到其它机器，则可以参考 Installer 中默认的 `root` 用户配置方式。

```
root@192.168.0.3 # ssh 192.168.0.2
Warning: Permanently added 'node1,192.168.0.2' (ECDSA) to the list of known hosts.
root@192.168.0.2's password:
Permission denied, please try again.
```

第二步，如下示例使用 3 台机器，参考以下示例修改主机配置文件 `hosts.ini`。

#### hosts.ini 配置示例

```
[all]
```

```
master ansible_connection=local ip=192.168.0.1 ansible_user=ubuntu ansible_become_pass=Qcloud@123
node1 ansible_host=192.168.0.2 ip=192.168.0.2 ansible_user=ubuntu ansible_become_pass=Qcloud@123
node2 ansible_host=192.168.0.3 ip=192.168.0.3 ansible_user=ubuntu ansible_become_pass=Qcloud@123
```

```
[kube-master]
```

```
master
```

```
[kube-node]
```

```
node1
```

```
node2
```

```
[etcd]
```

```
master
```

```
[k8s-cluster:children]
```

```
kube-node
```

```
kube-master
```

## 安装前如何配置 QingCloud vNas

3、KubeSphere 支持对接 [QingCloud vNas](#) 作为集群的存储服务端，以下说明如何在 [QingCloud 控制台](#) 创建文件存储 vNas：

3.1. 选择 **文件存储 vNAS**，点击 **创建**。

QINGCLOUD

北京3区-A

总览

计算

网络与 CDN

SD-WAN

存储

- 硬盘
- 对象存储
- 共享存储
- 文件存储 vNAS ①
- 备份

全部资源 北京3区-A / NAS / NAS

NAS 是支持基于 NFS 和 Samba(CIFS) 协议的网络共享存储服务。你可以将硬盘

文件存储 vNAS 权限组 账户

+ 创建 ② 更多操作

ID 名称 状态

结果为空

\* 提示：可通过在各个资源上点击「右键」来进行常用操作，以及「双击」来修改

3.2. 自定义名称，并选择与待安装机器相同的私有网络。

创建 NAS

总价格: ¥0.132 每小时 (合 ¥95.04 每月)

名称 kubesphere-vnas ①

类型 性能型 超高性能型

配置 小型 中型 大型

支持1-2个客户端同时读写

私有网络 kubesphereoffline ②

IP 自动分配 手动指定

③ 提交 取消

① 自定义名称，其它项保持默认  
② vNAS 应与待安装机器位于同一个私有网络中  
③ 点击「提交」

3.3. 点击创建的 vNAS 进入详情页，在共享存储目标下点击 创建。

北京3区-A / NAS / NAS / S2-XRUPHZM9

**基本属性**

ID: s2-xrphzm9  
名称: kubesphere-test-vnmas  
标签:  
描述: 无  
状态: 活跃 (Active)  
类型: 性能型 (Performance)  
节点数量: 1  
配置: 小型 (Small)  
私有网络: (kubesphereoffline)  
内网 IP: 192.168.0.22

**共享存储目标**

提示: 在执行共享存储目标的所有修改、禁用、删除等操作之前请在客户端停止对共享存储目标的访问，并执行 umount 操作，否则可能会引起客户端无法响应。

+ 创建 (highlighted with a red arrow)

ID	共享目录	类型	硬盘	权限组	状态	创建时间

合计: 0 每页: 10

3.4. 目标类型保持 NFS，参考如下截图填写信息，完成后点击 提交。

**创建共享存储目标**

目标类型: NFS (highlighted with a red circle 1)

共享目录: /mnt/shared\_dir (highlighted with a red circle 1)

目录名称, 如: /mnt/shared\_dir

硬盘: 没有可用的硬盘。 + 创建硬盘 (highlighted with a red circle 2)

建议使用新申请的硬盘（容量型或性能型），仅支持有文件系统且类型为 Ext4, XFS, NTFS 的硬盘，不支持存在多个分区的硬盘。

权限组: 缺省权限组 (highlighted with a red circle 3) + 创建权限组

提交 (highlighted with a red circle 3) 取消

**共享存储目标**

提示: 在执行共享存储目标的所有修改、禁用、删除等操作之前请在客户端停止对共享存储目标的访问，并执行 umount 操作，否则可能会引起客户端无法响应。

+ 创建 (highlighted with a red circle)

ID	共享目录	类型	硬盘	权限组	状态	创建时间
s2st-y9pqsytg	/mnt/shared_dir	NFS	(ks-test)	(缺省权限组)	已启用	2019-03-07 17:10:46

\* 提示: 可通过在各个资源上点击「右键」来进行常用操作，以及「双击」来修改基本属性。

3.5. 选择 账户，点击 创建。

NAS 是支持基于 NFS 和 Samba(CIFS) 协议的网络共享存储服务。你可以将硬盘挂载到 NAS 服务器上，让多个客户端通过网络连接进行共享，同时支持可访问服务的账号的控制管理。

文件存储 vNAS 权限组 账户 ①

通过创建 NFS 和 Samba 类型的账户，并绑定到需要访问的共享目标上，可以有效的控制和管理客户端主机对 NAS 的访问。账户可以和多个共享目标或服务器关联。

②

+ 创建 :: 更多操作 合计 : 0 每页: 10

ID	名称	类型	权限组	所属项目	创建时间

结果为空

\* 提示：可通过在各个资源上点击「右键」来进行常用操作，以及「双击」来修改基本属性。

3.6. 自定义名称，IP 地址填写集群机器所在的网段，如 `192.168.0.0/24`。

### 创建账户

① 自定义名称  
② 注意 IP 地址为集群机器所在的网段  
③ 点击「提交」

名称 kubesphere-nas-account ①

类型 NFS ②

IP 地址 192.168.0.0/24 ②

权限组 缺省权限组 + 创建权限组 读写权限 读写 ③

显示高级选项...

③ 提交 取消

文件存储 vNAS 权限组 账户

通过创建 NFS 和 Samba 类型的账户，并绑定到需要访问的共享目标上，可以有效的控制和管理客户端主机对 NAS 的访问。账户可以和多个共享目标或服务器关联。

+ 创建 :: 更多操作

ID	名称	类型	权限组	所属项目
s2a-xftlagpp	kubesphere-nas-account	NFS	IP 地址: 192.168.0.0/24	1

3.7. 查看 vNAS 的详情页，可以看到内网 IP 与 共享目录，这两处信息则需要在 Installer 中进行指定。

ID s2-xruphzm9  
名称 kubesphere-test-vnas  
标签  
描述 无  
状态 ● 活跃  
类型 性能型  
节点数量 1  
配置 小型  
私有网络 (kubesphereoffline)  
内网 IP 192.168.0.22

共享存储目标 监控

提示：在执行共享存储目标的所有修改、禁用、删除等操作之前请在客户端停止对共享存储目标的访问，并执行 umount 指令。无法响应。

ID	共享目录	类型	硬盘	权限组	状态
s2st-y9pqsytg	/mnt/shared_dir	NFS	(ks-test)	(缺省权限组)	已启用

\* 提示：可通过在各个资源上点击「右键」来进行常用操作，以及「双击」来修改基本属性。

如下，在 `/conf/vars.yml` 中先将 Local Volume Provisioner 设置为 false，然后在 NFS-Client provisioner 进行如下设置：

```
# NFS-Client provisioner deployment
nfs_client_enable: true
nfs_client_is_default_class: true
# Hostname of the NFS server(ip or hostname)
nfs_server: 192.168.0.22
# Basepath of the mount point to be used
nfs_path: /mnt/shared_dir
```

完成以上设置后，可参考安装指南继续进行配置和安装。

## 安装失败相关问题

4、安装过程中，如果遇到安装失败并且发现错误日志中有这类信息：`The following packages have pending transactions`，这种情况应该如何处理？

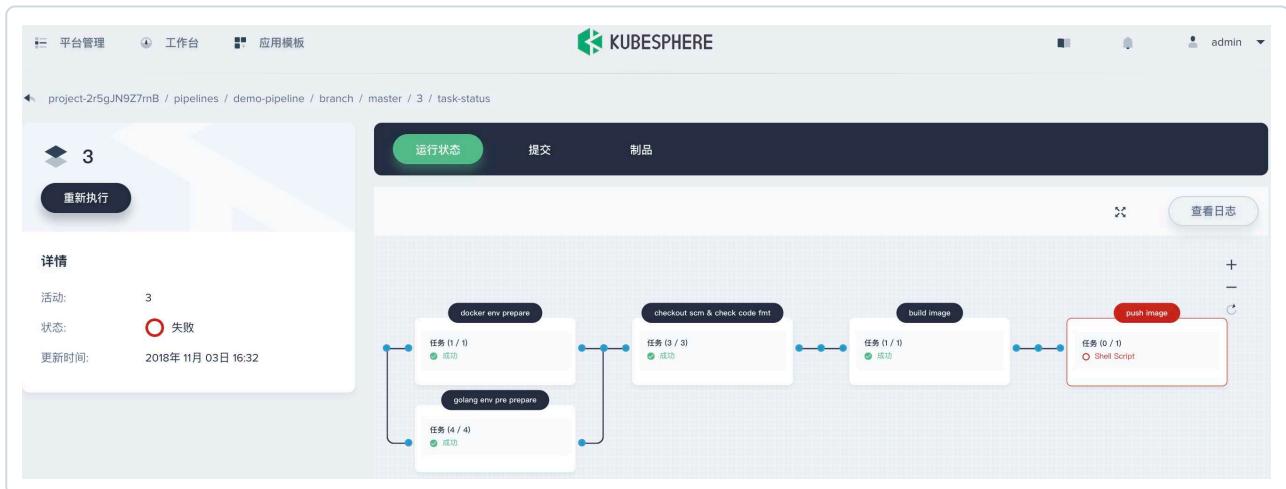
```
https://console.qingcloud.com/terminal/instance_001-28mpmzq0e-3mg  
按键操作 * 注意: 缺省的登录用户和密码可查看 映像描述, 若自定义了密码, 请使用您设定的密码进行登录。  
ending transactions: audit-libs-x86_64", "rc": 125, "results": []}  
fatal: [node1]: FAILED! => {"attempts": 4, "changed": false, "msg": "The following packages have pending transactions: docker-ce-x86_64, audit-x86_64, selinux-policy-targeted-noarch, selinux-policy-noarch, policycoreutils-x86_64, libsemanage-x86_64, audit-libs-x86_64", "rc": 125, "results": []}  
NO MORE HOSTS LEFT *****  
PLAY RECAP *****  
localhost : ok=1    changed=0    unreachable=0    failed=0  
master    : ok=61   changed=1    unreachable=0    failed=1  
node1     : ok=47   changed=0    unreachable=0    failed=1  
node2     : ok=48   changed=0    unreachable=0    failed=0  
  
Saturday 08 December 2018 15:23:16 +0000 (0:00:56.245)      0:01:31.361 *****  
=====  
kubernetes/preinstall : Update package management cache (YUM) ----- 56.25s  
kubernetes/preinstall : Create kubernetes directories ----- 2.47s  
gather facts from all instances ----- 2.17s  
kubernetes/preinstall : Create cni directories ----- 1.54s  
bootstrap-os : Assign inventory name to unconfigured hostnames (non-CoreOS and Tumbleweed) --- 1.03s  
bootstrap-os : Install pip for bootstrap ----- 0.99s  
bootstrap-os : Install packages requirements for bootstrap ----- 0.97s  
bootstrap-os : Gather nodes hostnames ----- 0.97s  
download : Sync container ----- 0.89s  
download : Download items ----- 0.87s  
Gathering Facts ----- 0.86s  
adduser : User ! Create User ----- 0.83s  
Gathering Facts ----- 0.68s  
kubernetes/preinstall : check resolvconf ----- 0.66s  
kubernetes/preinstall : check if atomic host ----- 0.66s  
bootstrap-os : Create remote_tmp for it is used by another module ----- 0.65s  
adduser : User ! Create User Group ----- 0.65s  
kubernetes/preinstall : Remove swapfile from /etc/fstab ----- 0.64s  
kubernetes/preinstall : check if /etc/dhcp/dhclient.conf exists ----- 0.62s  
kubernetes/preinstall : check if kubelet is configured ----- 0.62s  
Failed!  
[root@master scripts]#
```

答：这是因为有些 transactions 操作没有完成，可以连接到安装失败的节点上，依次执行下列命令，并重新执行 `install.sh` 脚本：

```
$ yum install yum-utils -y  
$ yum-complete-transaction  
$ yum-complete-transaction --cleanup-only
```

## 流水线运行报错相关问题

5、创建 Jenkins 流水线后，运行时报错怎么处理？



答：最快定位问题的方法即查看日志，点击 **查看日志**，具体查看出错的阶段 (stage) 输出的日志。比如，在 **push image** 这个阶段报错了，如下图中查看日志提示可能是 DockerHub 的用户名或密码错误。

This screenshot shows the '流水线运行日志' (Pipeline Run Log) section. It lists the stages and their statuses: docker env prepare (成功), golang env pre prepare (成功), checkout scm & check (成功), build image (成功), and push image (失败). The 'push image' stage is expanded to show its log output. The log content is as follows:

```

[927rnB_demo-pipeline_master-HQPU44W4MT4JGOXADR54FOFXAKLLCL4Z7NTCOOR3M7AOCLR72QA] Running shell script
+ docker login -u **** -p *****
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Error response from daemon: Get https://registry-1.docker.io/v2/: unauthorized: incorrect username or password
script returned exit code 1

```

The log shows an unsuccessful attempt to log in to DockerHub using placeholder credentials ('\*\*\*\*'). The error message indicates an 'unauthorized' response due to incorrect credentials. The log entry ends with 'script returned exit code 1'.

6、运行流水线失败时，查看日志发现是 Docker 镜像 push 到 DockerHub 超时问题 (Timeout)，比如以下情况，要怎么处理？

流水线运行日志

阶段	用时 1.18 min	
checkout scm		<a href="#">刷新</a> <a href="#">下载日志</a>
get dependencies	54.06 s <span style="color: green;">成功</span>	
unit test	16.31 s <span style="color: red;">失败</span>	
build & push snapshot	<pre>[I _jenkinsfile-in-SCM_master-0ZKRDPHU43YDJB14UVXMFWBWLCHNTZSXZ2Z5FZTBXXCJGZOFMSA] Running shell script + docker build -t docker.io/pengfeizhou/devops-docs-sample:SNAPSHOT-master-3 . Sending build context to Docker daemon 12.62MB Step 1/2 : FROM gatsbyjs/gatsby:latest Get https://registry-1.docker.io/v2/gatsbyjs/gatsby/manifests/latest: Get https://auth.docker.io/token?scope=repository%2findex: dial tcp: lookup auth.docker.io on 127.0.0.5:53: no such host script returned exit code 1</pre>	
push latest		

答：可能由于网络问题造成，建议尝试再次运行该流水线。

## 如何查看 kubeconfig 文件

### 7、如何查看当前集群的 Kubeconfig 文件？

用户可以通过打开 web kubectl 查看 Kubeconfig 文件，仅管理员或拥有 web kubectl 权限的用户有权限。

Kubeconfig 文件

把它加入到 ~/.kube/config:

```

1 apiVersion: v1
2 clusters:
3 - cluster:
4   certificate-authority-data:
LS0tLS1CR0dJTiBDRVJUSUZjQ0FURS0tLS0tCk1JSUNSRENDQWJDZ0F3SUJBZ01CQURBTkJna3Foa2lHOxwQkFR
c0ZBREfWTvJN0d0VRURWUVFERXdwcmRXSmwKZ01bGrIVnpNQjRYRFRFNE1USxdPREUwTlRnME0xb1hEVEk0TVRJ
d0SURTB0VGcwTTFvd0ZURVRNQkVHQTfVRQpBeE1LYTNWaVpYSnVzWFjSy3pDQ0FTSxjEUlKS29aSWh2Y05BUJVC
Q1fBRGdnRVBBRENQVqFvQ2dnRUJB52NGc19DZzs1ZzRqbVhpVdGCrjU1UN0SUno2NEhKEFzZmk1L0352RDTvNP
QzJGU0vNzRhMF0STRQbJ3MTFNv88KTzhkUXpxzb2fUTk4Y0zLz2gbExJUvhkVRJS21pb0fjU3p2cnVNWZE
enNTK0FNdWHydm1uWhmMcWdKRVNJTQpxbF1rcUk0R1dWWNgvVZsdWh10UcrbUo1EVPRGERMvdJWFZXS09JUERN
Z1VkmStacT2Bnj10R2U0c3BcmdiCLVtb2krV3Q1YNCazV1WkV4RTdEVnhkd31vZ1lcfkN0Y4Shcws5jBMcOpj
Ykh6RHAvWX1qNUJ0d1p0ZVZDU28KMUNFNhIArcnViehPM3d5ckVmuJrNaU85aG9iV1N4YXMyMm5FL2hzMU44amNG
VHNW$0ImVG1mcHdkOE1NTNtegorSFVPlEvbnndRX1OsRNVC9rQ0F3RUFByU1qTUNF0dRmURWUjBQVFl0J8
UURBZ0tTUE4R0ExVWRFd0VCC193UUZNQ01CQYYd0RRWUp1b1pJdHzjtTkfRRUxCUfEZ2dF0kFINGRHEZ1UTJn
TUF5d3k0azzczUzVMRS9CeEKRTRaZXgxehFybE9yelpQTEU0c01MK1hmM3NFd3dwU0NLNVc30VGZ3RMl3dc000
ZDQxKz2CVdFCbm1TTV0ngpZUWxGymdnNzNCUKsrQ2pSNXZ1SKBPY01YL1ptuMzvY292TE1DUi91NXB45FvRnMm0w
SGs5KzJ1V0drra3NyNWNNTCmdybjzqY3ZPUnovR1JFYkxPbnipsZEs2a3pjcllwSHZaWE1PRHIp6dG1zcmY1cVBLRnR6
WFFD5nBERURUTzZtV2YKrJYyQUEMERyUmNsYThJY0tEdlNr-UNZ1zEp2UJHV0xKTXtUkkc2T2YxOgxZs2v1Fq
chLRVFUvYtJNHZh0QpG0Xo1YkhF0C9hM0p0TUM2S0N1bFZVMWFJUJswL2ptbVFLRHxWHN3oDF6VzQ1SGkzQXJG
```

[取消](#) [复制](#)

## 如何访问 Jenkins 服务端

### 8、如何访问和登录 Jenkins 服务端？

Installer 安装将会同时部署 Jenkins Dashboard，该服务暴露的端口（NodePort）为 **30180**，确保外网流量能够正常通过该端口，然后访问公网 IP 和端口号（\${EIP}:\${NODEPORT}）即可。Jenkins 已对接了 KubeSphere 的 LDAP，因此可使用用户名 **admin** 和 KubeSphere 集群管理员的密码登录 Jenkins Dashboard。

## 关于对 CephRBD、GlusterFS 开源存储的支持方式

### 9、关于对 CephRBD、GlusterFS 开源存储的支持方式

Installer 集成了这两类开源存储的存储插件，并在安装过程基于配置文件帮助用户完成部署配置工作，但其存储服务端的部署和运维并不包含在 KubeSphere 平台支持范围内。

## NeonSAN 存储插件是否支持非 KubeSphere 的 Kubernetes 环境

### 10、NeonSAN 存储插件是否支持非 KubeSphere 的 Kubernetes 环境

支持，[NeonSAN 存储插件](#) 基于 CSI 0.3.0 开发，理论上可以支持 Kubernetes **1.11** 及以上版本，经过 KubeSphere 已验证可支持的版本包括 Kubernetes **1.12** 及以上版本，在非 KubeSphere 环境中部署 NeonSAN 存储插件可参考[此链接文档](#)，关于部署、使用的各种问题可直接在 GitHub 上提 issue。

**说明：**若您在使用中遇到任何产品相关的问题，欢迎在 [GitHub Issue](#) 提问。