

# KubeSphere 文档 Advanced v2.0

# Release Notes For 2.0.2

KubeSphere 高级版 (Advanced Edition 2.0.2) 已于 2019 年 7 月 9 日 正式发布，修复了已知的 Bug，增强了现有的功能。建议下载并安装最新的 2.0.2 版本，若您已经安装了 1.0.x、2.0.0 或 2.0.1 版本，请下载 2.0.2 版本的 Installer，支持一键升级至 2.0.2。

## 高级版 2.0.2 更新详情列表

### 功能增强

- 官网提供 [API docs](#)
- 防止暴力破解
- 规范资源名称的最大长度
- 升级项目网关 (Ingress Controller) 版本至 0.24.1，支持 Ingress 灰度发布

### Bug 修复详情列表

- 修复流量拓扑图显示项目之外资源的问题
- 修复流量拓扑图在特定条件下显示多余服务组件的问题
- 修复 Source to Image 重新构建镜像操作在特定条件下无法执行的问题
- 修复失败的 Source To Image 任务导致服务选择页面展示的问题
- 修复容器组 (Pod) 状态异常时无法查看日志的问题
- 修复磁盘监控无法检测到部分挂盘类型，例如 LVM 硬盘
- 修复已部署应用无法删除的问题
- 修复应用组件状态判断有误的问题
- 修复主机节点数计算错误的问题
- 修复添加环境变量时切换到引用配置按钮导致输入数据丢失的问题
- 修复项目 Operator 角色用户不能 rerun job 的问题

- 修复无 IPv4 环境 uuid 无法初始化的问题
- 修复日志详情页无法滚动下拉查看过往日志的问题
- 修复 KubeConfig 文件中 APIServer 地址错误的问题
- 修复无法修改 DevOps 工程名称的问题
- 修复容器日志无法指定时间查询的问题
- 修复在特定情况下关联的镜像仓库秘钥没有保存的问题
- 修复应用下服务组件创建页面没有镜像仓库秘钥参数的问题

## 安装 2.0.2

### 安装指南

高级版 2.0.2 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

高级版 2.0.2 修复了已知的 Bug，支持一键升级至 2.0.2，请参考 [升级指南](#)。

# Release Notes For 2.0.1

KubeSphere 高级版 (Advanced Edition 2.0.1) 修复了已知的 Bug, 2.0.1 已于 2019 年 6 月 9 日正式发布。建议下载并安装最新的 2.0.1 版本, 若您已经安装了 1.0.x 或 2.0.0 版本, 请下载 2.0.1 版本的 Installer, 支持一键升级至 2.0.1。

## Bug 修复详情列表

- 修复 CI/CD 流水线无法正确识别代码分支名中含有特殊字符的问题
- 修复 CI/CD 流水线在部分情况下无法查看日志的问题
- 修复日志查询时, 因索引文档分片异常造成的无日志数据输出的问题, 并增加对异常的提示
- 修复无日志情况下搜索日志, 提示异常的问题
- 修复流量治理拓扑图线条重叠的问题 修复镜像策略应用无效的问题

## 安装 2.0.1

### 安装指南

高级版 2.0.1 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

高级版 2.0.1 修复了已知的 Bug, 支持一键升级至 2.0.1, 请参考 [升级指南](#)。

# Release Notes For 2.0.0

KubeSphere 高级版 (Advanced Edition 2.0.0) 已于 2019 年 5 月 18 日 正式发布。建议下载并安装最新的 2.0.0 版本，若您已经安装了 1.0.1 或 1.0.0 版本，请下载 2.0.0 版本的 Installer，支持一键升级 1.0.0 (或 1.0.1) 至 2.0.0。

## 高级版 2.0.0 更新详情列表

### 组件升级

- 支持 Kubernetes 升级至 [Kubernetes 1.13.5](#)
- 集成 [QingCloud Cloud Controller](#)，安装后可通过 KubeSphere 控制台创建 QingCloud 负载均衡器并自动绑定后端工作负载
- 集成 [QingStor CSI v0.3.0](#) 存储插件，支持物理 NeonSAN 存储系统，可提供高可用、高性能分布式 SAN 存储服务
- 集成 [QingCloud CSI v0.2.1](#) 存储插件，支持 QingCloud 单副本硬盘，容量型、性能型、超高性能型、基础型、企业型、NeonSAN 硬盘
- 可选安装组件 Harbor 升级至 1.7.5
- 可选安装组件 GitLab 升级至 11.8.1
- Prometheus 升级至 2.5.0

### 微服务治理

- 集成 Istio 1.1.1，支持图形化创建包含多个微服务组件的应用
- 支持为项目（外网访问）和应用开启/关闭应用治理
- 内置微服务治理示例 [Bookinfo](#)
- 支持流量治理
- 支持流量镜像

- 基于 Istio 提供微服务级别的负载均衡功能
- 支持金丝雀发布
- 支持蓝绿部署
- 支持熔断
- 支持链路追踪

## DevOps

- CI/CD 流水线提供邮件通知功能，支持构建过程中发送邮件
- CI/CD 图形化编辑流水线增强，增加更多流水线常用插件和执行条件
- 提供基于 SonarQube 7.4 的代码漏洞扫描功能
- 提供 [Source to Image](#) 功能，基于代码无需 Dockerfile 直接构建应用负载并发布运行

## 监控

- 提供 Kubernetes 组件独立监控页，包括 etcd、kube-apiserver 和 kube-scheduler
- 优化若干监控指标算法
- 优化监控资源占用，减少 Prometheus 内存及磁盘使用量 80% 左右

## 日志

- 提供根据租户权限过滤的统一日志检索控制台
- 支持精确和模糊日志检索
- 支持实时和历史日志检索
- 支持组合条件日志检索，包括：项目、工作负载、容器组、容器、关键字和时间范围
- 支持单条日志直达日志详情页，并可切换不同容器组以及容器组下不同容器
- 支持日志详情页直达容器组或者容器详情页

- 基于自研 [FluentBit Operator](#) 提供灵活的日志收集配置：可添加 Elasticsearch、Kafka 和 Fluentd 作为日志接收者，并可按需激活/关闭；在发送给日志接收者前，可灵活输入过滤条件筛选出所需要的日志

## 告警和通知

- 支持针对集群节点和工作负载资源的邮件通知
- 支持组合通知规则：可组合多种监控资源，制定不同的告警级别、检测周期、推送次数和阈值
- 可配置通知时间段和通知人
- 支持针对不同通知级别设置独立的通知重复规则

## 安全加强

- 修复 [runc 容器逃逸漏洞](#)
- 修复 [Alpine 镜像 shadow 漏洞](#)
- 支持单点和多点登录配置项
- 多次无效登录后强制要求输入验证码
- 账户密码策略增强，阻止创建弱密码的账户
- 其他若干安全增强

## 界面优化

- Console 多处用户体验优化，如支持在 DevOps 工程与项目间切换
- 优化多处中英文页面文案

## 其他

- 支持 etcd 备份及恢复
- 支持 docker 镜像定期清理

## Bug 修复详情列表

- 修复删除页面资源页面未及时更新问题
- 修复删除 HPA 工作负载后残留脏数据问题
- 修复任务（Job）状态显示不正确的问题
- 更正资源配额、Pod 用量、存储指标算法
- 调整 CPU 用量结果精度
- 其他若干 Bug 修复

## 安装 2.0.0

### 安装指南

高级版 2.0.0 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

高级版 2.0.0 相较于 1.0.1 提供了更丰富的企业级功能，并对 1.0.1 已有功能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 (或 1.0.1) 至 2.0.0，请参考 [升级指南](#)。

## 产品简介

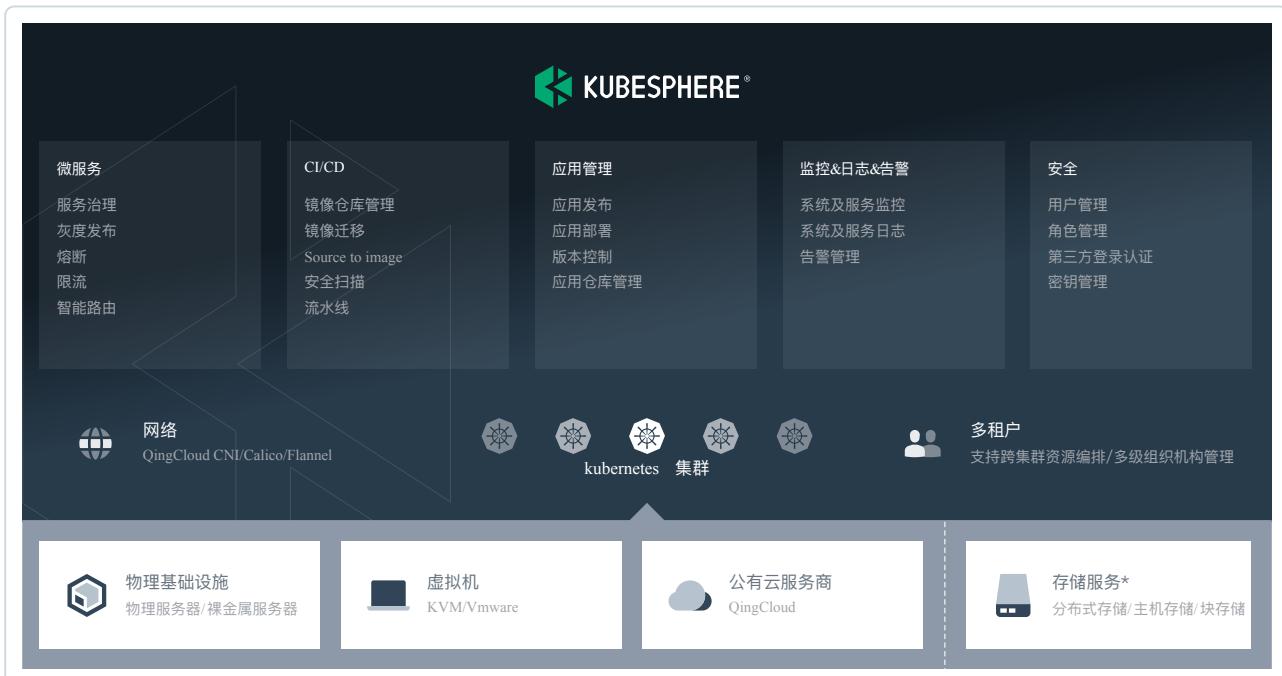
[KubeSphere®?](#) 是在目前主流容器调度平台 [Kubernetes](#) 之上构建的企业级分布式多租户容器管理平台，提供简单易用的操作界面以及向导式操作方式，在降低用户使用容器调度平台学习成本的同时，极大减轻开发、测试、运维的日常工作的复杂度，旨在解决 Kubernetes 本身存在的存储、网络、安全和易用性等痛点。除此之外，平台已经整合并优化了多个适用于容器场景的功能模块，以完整的解决方案帮助企业轻松应对敏捷开发与自动化运维、微服务治理、多租户管理、工作负载和集群管理、服务与网络管理、应用编排与管理、镜像仓库管理和存储管理等业务场景。

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、多维度监控告警日志、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，新增微服务治理、灰度发布、s2i、代码质量检查等，后续还将提供和支持多集群管理、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](#)。



## 产品规划

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



# 产品功能

KubeSphere®？作为企业级的全栈化容器管理平台，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 目前提供了工作负载管理、微服务治理、DevOps 工程、Source to Image、多租户管理、多维度监控、日志查询与收集、告警通知、服务与网络、应用管理、基础设施管理、镜像管理、应用配置密钥管理等功能模块，开发了适用于物理机部署 Kubernetes 的 [负载均衡器插件 Porter](#)，并支持对接多种开源的存储与网络方案，支持高性能的商业存储与网络服务。以下从专业的角度为您详解各个模块的功能服务：

## Kubernetes 资源管理

对底层 Kubernetes 中的多种类型的资源提供极简的图形化向导式 UI 实现工作负载管理、镜像管理、服务与应用路由管理（服务发现）、密钥配置管理等，并提供弹性伸缩（HPA）和容器健康检查支持，支持数万规模的容器资源调度，保证业务在高峰并发情况下的高可用性。

## 微服务治理

- 灵活的微服务框架：基于 Istio 微服务框架提供可视化的微服务治理功能，将 Kubernetes 的服务进行更细粒度的拆分
- 完善的治理功能：支持熔断、灰度发布、流量管控、限流、链路追踪、智能路由等完善的微服务治理功能，同时，支持代码无侵入的微服务治理

## 多租户管理

- 多租户：提供基于角色的细粒度多租户统一认证与三层级权限管理
- 统一认证：支持与企业基于 LDAP / AD 协议的集中认证系统对接，支持单点登录（SSO），以实现租户身份的统一认证
- 权限管理：权限等级由高至低分为集群、企业空间与项目三个管理层级，保障多层次不同角色间资源共享且互相隔离，充分保障资源安全性

## DevOps 工程

- 开箱即用的 DevOps：基于 Jenkins 的可视化 CI / CD 流水线编辑，无需对 Jenkins 进行配置，同时内置丰富的 CI/CD 流水线插件
- CI/CD 图形化流水线提供邮件通知功能，新增多个执行条件
- 端到端的流水线设置：支持从仓库 (GitHub / SVN / Git)、代码编译、镜像制作、镜像安全、推送仓库、版本发布、到定时构建的端到端流水线设置
- 安全管理：支持代码静态分析扫描以对 DevOps 工程中代码质量进行安全管理
- 日志：日志完整记录 CI / CD 流水线运行全过程

## Source to Image

提供 Source to Image (s2i) 的方式从已有的代码仓库中获取代码，并通过 Source to Image 构建镜像的方式来完成部署，并将镜像推送至目标仓库，每次构建镜像的过程将以任务 (Job) 的方式去完成。

## 多维度监控

- KubeSphere 全监控运维功能可通过可视化界面操作，同时，开放标准接口，易于对接企业运维系统，以统一运维入口实现集中化运维
- 立体化秒级监控：秒级频率、双重维度、十六项指标立体化监控
  - 在集群资源维度，提供 CPU 利用率、内存利用率、CPU 平均负载、磁盘使用量、inode 使用率、磁盘吞吐量、IOPS、网卡速率、容器组运行状态、ETCD 监控、API Server 监控等多项指标
  - 在应用资源维度，提供针对应用的 CPU 用量、内存用量、容器组数量、网络流出速率、网络流入速率等五项监控指标。并支持按用量排序和自定义时间范围查询，快速定位异常
- 提供按节点、企业空间、项目等资源用量排行
- 提供服务组件监控，快速定位组件故障

## 自研多租户告警系统

- 支持基于多租户、多维度的监控指标告警，目前告警策略支持集群管理员对节点级别和租户对工作负载级别等两个层级
- 灵活的告警策略：可自定义包含多个告警规则的告警策略，并且可以指定通知规则和重复告警的规则
- 丰富的监控告警指标：提供节点级别和工作负载级别的监控告警指标，包括容器组、CPU、内存、磁盘、网络等多个监控告警指标
- 灵活的告警规则：可自定义某监控指标的检测周期长度、持续周期次数、告警等级等
- 灵活的通知发送规则：可自定义发送通知时间段及通知列表，目前支持邮件通知
- 自定义重复告警规则：支持设置重复告警周期、最大重复次数并和告警级别挂钩

## 日志查询与收集

- 提供多租户日志管理，在 KubeSphere 的日志查询系统中，不同的租户只能看到属于自己的日志信息
- 多级别的日志查询（项目/工作负载/容器组/容器以及关键字）、灵活方便的日志收集配置选项等
- 支持多种日志收集平台，如 Elasticsearch、Kafka、Fluentd

## 多存储类型支持

- 支持 GlusterFS、CephRBD、NFS 等开源存储方案，支持有状态存储
- NeonSAN CSI 插件对接 QingStor NeonSAN，以更低时延、更加弹性、更高性能的存储，满足核心业务需求
- QingCloud CSI 插件对接 QingCloud 云平台各种性能的块存储服务

## 多网络方案支持

- 支持 Calico、Flannel 等开源网络方案
- 开发了适用于物理机部署 Kubernetes 的 [负载均衡器插件 Porter](#)

## 应用管理与编排

- 使用开源的 [OpenPitrix](#) 提供应用商店和应用仓库服务，为用户提供应用全生命周期管理功能
- 用户基于应用模板可以快速便捷地部署一个完整应用的所有服务

## 基础设施管理

提供存储类型管理、主机管理和监控、资源配额管理，并且支持镜像仓库管理、权限管理、镜像安全扫描。内置 Harbor 镜像仓库，支持添加 Docker 或私有的 Harbor 镜像仓库。

# 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 可以快速管理 Kubernetes 集群、部署应用、服务发现、CI/CD 流水线、集群扩容、微服务治理、日志查询和监控告警。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，提供更友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。

## 极简体验，向导式 UI

- 面向开发、测试、运维友好的用户界面，向导式用户体验，降低 Kubernetes 学习成本的设计理念
- 用户基于应用模板可以一键部署一个完整应用的所有服务，UI 提供全生命周期管理

## 业务高可靠与高可用

- 自动弹性伸缩：部署（Deployment）支持根据访问量进行动态横向伸缩和容器资源的弹性扩缩容，保证集群和容器资源的高可用
- 提供健康检查：支持为容器设置健康检查探针来检查容器的健康状态，确保业务的可靠性

## 容器化 DevOps 持续交付

- 简单易用的 DevOps：基于 Jenkins 的可视化 CI/CD 流水线编辑，无需对 Jenkins 进行配置，同时内置丰富的 CI/CD 流水线模版
- Source to Image (s2i)：从已有的代码仓库中获取代码，并通过 s2i 自动构建镜像完成应用部署并自动推送至镜像仓库，无需编写 Dockerfile
- 端到端的流水线设置：支持从仓库（GitHub / SVN / Git）、代码编译、镜像制作、镜像安全、推送仓库、版本发布、到定时构建的端到端流水线设置
- 安全管理：支持代码静态分析扫描以对 DevOps 工程中代码质量进行安全管理
- 日志：日志完整记录 CI / CD 流水线运行全过程

## 开箱即用的微服务治理

- 灵活的微服务框架：基于 Istio 微服务框架提供可视化的微服务治理功能，将 Kubernetes 的服务进行更细粒度的拆分，支持无侵入的微服务治理
- 完善的治理功能：支持灰度发布、熔断、流量监测、流量管控、限流、链路追踪、智能路由等完善的微服务治理功能

## 灵活的持久化存储方案

- 支持 GlusterFS、CephRBD、NFS 等开源存储方案，支持有状态存储
- NeonSAN CSI 插件对接 QingStor NeonSAN，以更低时延、更加弹性、更高性能的存储，满足核心业务需求

- QingCloud CSI 插件对接 QingCloud 云平台各种性能的块存储服务

## 灵活的网络方案支持

- 支持 Calico、Flannel 等开源网络方案
- 分别开发了 [QingCloud 云平台负载均衡器插件](#) 和适用于物理机部署 Kubernetes 的 [负载均衡器插件 Porter](#)
- 商业验证的 SDN 能力：可通过 QingCloud CNI 插件对接 QingCloud SDN，获得更安全、更高性能的网络支持

## 多维度监控日志告警

- KubeSphere 全监控运维功能可通过可视化界面操作，同时，开放标准接口对接企业运维系统，以统一运维入口实现集中化运维
- 可视化秒级监控：秒级频率、双重维度、十六项指标立体化监控；提供服务组件监控，快速定位组件故障
- 提供按节点、企业空间、项目等资源用量排行
- 支持基于多租户、多维度的监控指标告警，目前告警策略支持集群节点级别和工作负载级别等两个层级
- 提供多租户日志管理，在 KubeSphere 的日志查询系统中，不同的租户只能看到属于自己的日志信息

## 应用场景

KubeSphere®? 适用于企业在数字化转型时所面临的敏捷开发与自动化运维、微服务应用架构与流量治理、自动弹性伸缩和业务高可用、DevOps 持续集成与交付等应用场景。

### 一步升级容器架构，助力业务数字化转型

企业用户部署于物理机、传统虚拟化环境的业务系统，各业务模块会深度耦合，资源不能灵活的水平扩展。KubeSphere 帮助企业将 IT 环境容器化并提供完整的运维管理功能，同时依托青云QingCloud 为企业提供强大的网络、存储支持，并可高效对接企业原监控、运维系统，一站式高效完成企业 IT 容器化改造。

### 多维管控 Kubernetes，降低运维复杂度

无论将业务架构在 Kubernetes 平台上的用户，还是使用多套来自不同厂商提供的 Kubernetes 平台的用户，复杂的运维管理使企业压力倍增。KubeSphere 可提供统一平台纳管异构 Kubernetes 集群，支持应用自动化部署，减轻日常运维压力。同时，完善的监控告警与日志管理系统有效节省运维人工成本，使企业能够将更多精力投放到业务创新上。

### 敏捷开发与自动化运维，推动企业 DevOps 落地

DevOps 将开发团队与运营团队通过一套流程或方法建立更具协作性、更高效的关系，使得开发、测试、发布应用能够更加敏捷、高效、可靠。KubeSphere CI / CD 功能可为企业 DevOps 提供敏捷开发与自动化运维。同时，KubeSphere 的微服务治理功能，帮助企业以一种细粒度的方式开发、测试和发布服务，有效推动企业 DevOps 落地。

### 灵活的微服务解决方案，一步升级云原生架构

微服务架构可轻量级构建冗余，可扩展性强，非常适合构建云原生应用程序。KubeSphere 基于主流微服务解决方案 Istio，提供无代码侵入的微服务治理平台。后续将集成 SpringCloud，便于企业构建 Java 应用，助力企业一步实现微服务架构，实现应用云原生转型。

## 基于物理环境构建全栈容器架构，释放硬件最大效能

支持在全物理环境部署全栈容器架构，利用物理交换机，为 KubeSphere 提供负载均衡器服务，同时，通过 KubeSphere 与 QingCloud VPC 以及QingStor NeonSAN 的组合，可打通负载均衡、容器平台、网络、存储全栈功能，实现真正意义上的物理环境一体化多租户容器架构解决方案，并实现自主可控、统一管理。避免虚拟化带来的性能损耗，释放硬件最大效能。

## 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> 。
s2i	Source to Image，通过代码构建新的容器镜像，表示从已有的代码仓库中获取代码，并通过 Source to Image 的方式构建镜像的方式来完成部署，每次构建镜像的过程将以任务 (Job) 的方式去完成。
蓝绿部署	提供了一种零宕机的部署方式，在保留旧版本的同时部署新版本，将两个版本同时在线，如果有问题可以快速处理
金丝雀发布	将一部分真实流量引入一个新版本进行测试，测试新版本的性能和表现，在保证系统整体稳定运行的前提下，尽早发现新版本在实际环境上的问题。
流量镜像	流量镜像功能通常用于在生产环境进行测试，是将生产流量镜像拷贝到测试集群或者新的版本中，在引导用户的真实流量之前对新版本进行测试，旨在有效地降低新版本上线的风险。

# 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。

## 安装 KubeSphere

### 安装模式

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。

### 支持 GPU

KubeSphere 安装支持 GPU 节点，也支持 CPU 与 GPU 的混合部署模式，需在安装配置文件 `conf/vars.yml` 中进行设置，配置示例和说明参考 [集群组件配置释义](#)。

### 可选安装项

另外，KubeSphere Installer 集成了 [Harbor](#) 和 [GitLab](#)，但默认情况下不会安装 Harbor 和 GitLab，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](#) 和 [安装内置 GitLab](#)。

### 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。如果不能访问外网，则需要下载离线安装包。
- 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，因此可直接参考 [all-in-one](#) 和 [multi-node](#) 的安装指南下载安装。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](#)。

## 集群组件配置释义

注意，在 CI/CD 流水线中发送邮件通知需要安装前预先在 Installer 中配置邮件服务器，配置请参考 [集群组件配置释义](#) (下一版本将支持安装后在 UI 统一配置邮件服务器)。

如果需要查看或修改网络、组件版本、可选安装项（如 GitLab、Harbor）、外部负载均衡器、Jenkins、SonarQube、邮件服务器等配置参数时，可参考以下说明进行修改，集群组件配置释义文档对 installer 中的安装配置文件 `conf/vars.yml` 进行说明，简单介绍每一个字段的意义，参考 [集群组件配置释义](#)。

### 安装 QingCloud 负载均衡器插件（可选）

服务或应用路由如果通过 LoadBalancer 的方式暴露到外网访问，则需要安装对应的云平台负载均衡器插件来支持。如果在 QingCloud 云平台安装 KubeSphere，建议在 `conf/vars.yml` 中配置 QingCloud 负载均衡器插件相关参数，installer 将自动安装 [QingCloud 负载均衡器插件](#)，详见 [安装 QingCloud 负载均衡器插件](#)。

### 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，需 [安装前](#) 在配置文件 `conf/vars.yml` 中进行配置。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](#)。

### 安装内置 GitLab（可选）

KubeSphere Installer 集成了 Harbor 的 Helm Chart（版本为 harbor-18.11.1），内置的 Gitlab（版本为 v11.3.4）作为可选安装项，需 [安装前](#) 在配置文件 `conf/vars.yml` 中进行配置。用户可以根据团队项目的需求来配置安装，详见 [安装内置 GitLab](#)。

### 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高级版 1.0.x (或 2.0.0) 版本，我们强烈建议您升级至最新的高级版 2.0.1，最新的 Installer 支持将 KubeSphere 从 1.0.x (或 2.0.0) 环境一键升级至目前最新的 2.0.1，详见 [升级指南](#)。

## 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/</code>

序 列	禁止操作
	kubesphere/。
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 2.0.2 中的相关组件将默认安装以下版本：

组件	版本
KubeSphere	Advanced Edition 2.0.2
Kubernetes	v1.13.5
Istio	1.1.1
etcd	3.2.18
OpenPitrix	v0.3.5
Elasticsearch	v6.7.0
Prometheus	v2.3.1
Jenkins	v2.138
SonarQube	v7.4
GitLab	11.8.1
Harbor	1.7.5

# 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。

## 提示：

- 若需要安装 Harbor 和 GitLab 请在安装前参考 [安装内置 Harbor](#) 和 [安装内置 GitLab](#)。
- 若在云平台使用在线安装，可通过调高带宽的方式来加快安装速度。

## 前提条件

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。

## 说明：

- 若使用 ubuntu 16.04 建议使用其最新的版本 16.04.5；
- 若使用 ubuntu 18.04，则需要使用 root 用户；
- 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。
- 若需要选装 Harbor 和 GitLab，则主机的内存需要 16 G 以上。

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

## 第二步: 准备安装包

### 在线版 (2.0.2)

下载 [KubeSphere Advanced Edition 2.0.2](https://kubesphere.io/download/stable/advanced-2.0.2) 安装包至待安装机器，进入安装目录。

```
$ curl -L https://kubesphere.io/download/stable/advanced-2.0.2 > advanced-2.0.2.tar.gz &&
```

### 离线版 (2.0.2)

下载 [离线安装包 \(2.0.2\)](https://kubesphere.io/download/offline/advanced-2.0.2) 至待安装机器。

```
$ curl -L https://kubesphere.io/download/offline/advanced-2.0.2 > advanced-2.0.2.tar.gz &&
```

## 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 v1.13.5，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。

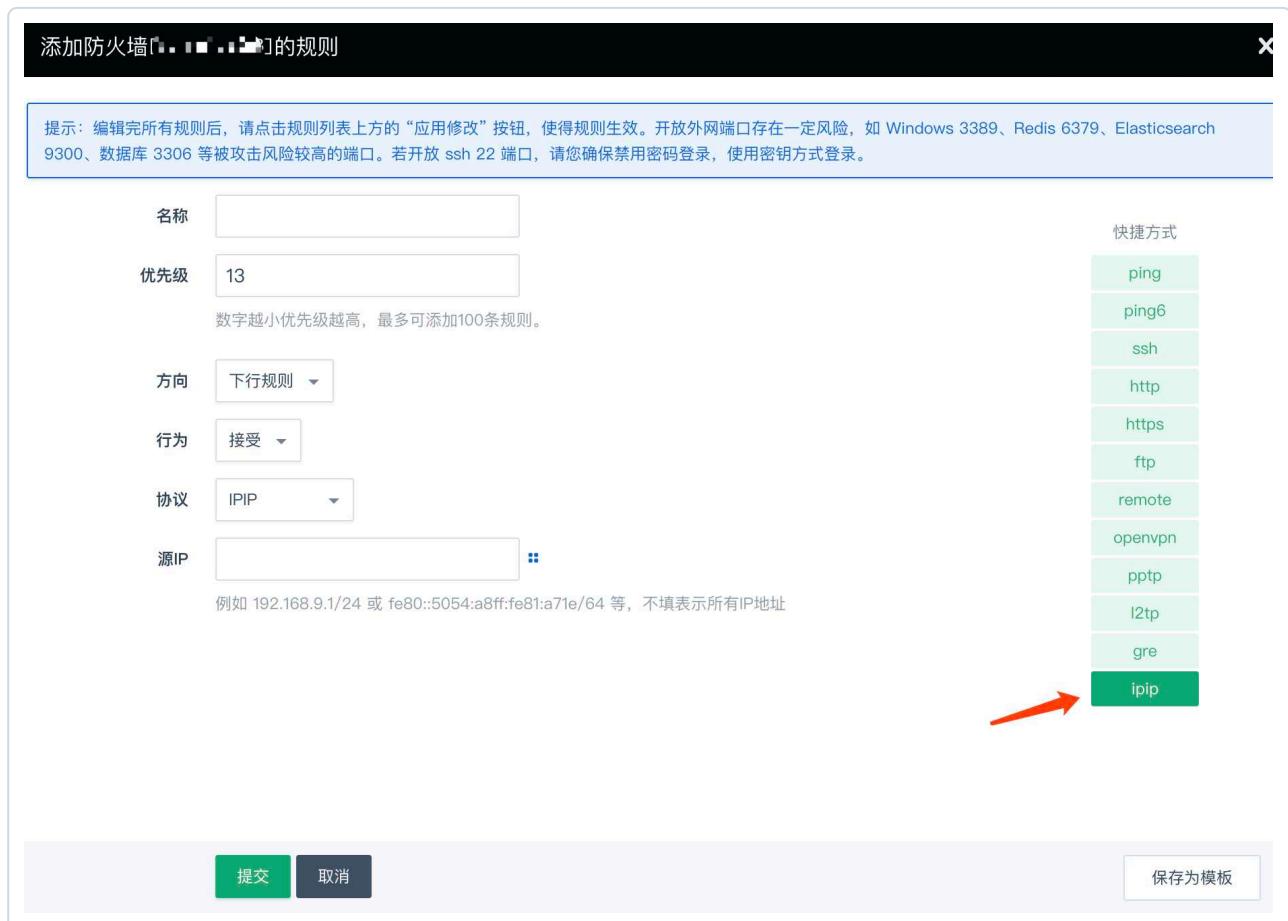
### 说明:

- 通常情况您不需要修改任何配置，直接安装即可。
- 网络默认插件是 [calico](#)，若您需要自定义安装参数，如网络、存储、GitLab、Harbor、负载均衡器插件等相关内容需在 [conf/vars.yml](#) 配置文件中指定或修改，参考 [集群组件配置说明](#)。
- 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 26 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](#) (QingCloud 公有云单节点挂盘限制为 10 块)、[QingStor NeonSAN](#)、[GlusterFS](#)、[Ceph RBD](#)、[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  
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。

## 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：



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. 建议使用 root 用户安装，执行 install.sh 脚本：

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

2. 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/      2018-07-08
#####
Please input an option: 1
```

3. 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。

```
successfull!
#####
###          Welcome to KubeSphere!          ###
#####

Console: http://192.168.0.8:30880
Account: admin
Password: P@88w0rd

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) 若需要在外网访问，在云平台需要在端口转发规则中将**内网端口** 30880 转发到**源端口** 30880，然后在防火墙开放这个**源端口**，确保外网流量可以通过该端口。

例如在 QingCloud 平台配置端口转发和防火墙规则，则可以参考 [云平台配置端口转发和防火墙](#)。

(3) 安装成功后，浏览器访问对应的 URL，如 [http://{\\$公网IP}:30880](http://{$公网IP}:30880)，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，**登录后请立即修改默认密码**。参阅 [快速入门](#) 帮助您快速上手 KubeSphere。



注意：登陆 Console 后请在 ”集群状态” 查看服务组件的监控状态，待所有组件启动完成后即可开始使用，通常所有服务组件都将在 15 分钟内启动完成。



# 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。

## 提示：

- 若需要安装 Harbor 和 GitLab 请在安装前参考 [安装内置 Harbor](#) 和 [安装内置 GitLab](#)。
- 若在云平台使用在线安装，可通过调高带宽的方式来加快安装速度。

## 前提条件

- 已准备 KubeSphere 支持的 [持久化存储服务端](#)，本篇文档以配置 QingCloud–CSI 插件对接 [QingCloud 云平台块存储](#) 为例，需要有 [QingCloud 云平台](#) 的账号。
- 注意，使用 QingCloud 云平台块存储作为存储服务，安装前需要确保用户账号在当前 Zone 资源配额满足最低要求。Multi–node 安装最少需要 14 块硬盘，本示例默认使用容量型硬盘，所需的容量型硬盘容量的最低配额为 1400 GB，若硬盘数量和容量配额不够请提工单申请配额。若使用其他类型的硬盘，参考 [QingCloud 各类型块存储配额表](#)。

The screenshot shows the QingCloud console interface at the URL <https://console.qingcloud.com/pek3/overview/>. The left sidebar has a green 'Overview' tab highlighted with a red arrow. The main area displays 'Resource Quota Usage Status' with two columns: 'Compute' and 'Storage'. In the 'Storage' column, there is a table with rows for various disk types. One row, 'Capacity Type Hard Disk (Blocks)', is highlighted with a red box and a red arrow pointing to it. It shows '0 / 20 (blocks)'. A green button labeled 'Apply Quota' is also visible.

# 第一步: 准备主机

您可以参考以下节点规格准备 至少 3 台 符合要求的主机开始 multi-node 模式的部署。

## 说明:

- 若选择离线安装，则系统盘建议在 100 G 以上；
- 若使用 ubuntu 16.04 建议使用其最新的版本 16.04.5；
- 若使用 ubuntu 18.04，则需使用 root 用户；
- 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。
- 若需要选装 Harbor 和 GitLab，则所有主机的总内存需要 24 GiB 以上。

操作系统	最小配置 (根据集群规模)
CentOS 7.5 (64 bit)	总 CPU 应不小于 8 核， 总内存不小于 16 G， 系统盘：40 G
Ubuntu 16.04/18.04 LTS (64 bit)	总 CPU 应不小于 8 核， 总内存不小于 16 G， 系统盘：40 G
Red Hat Enterprise Linux Server 7.4 (64 bit)	总 CPU 应不小于 8 核， 总内存不小于 16 G， 系统盘：40 G
Debian Stretch 9.5 (64 bit)	总 CPU 应不小于 8 核， 总内存不小于 16 G， 系统盘：40 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。

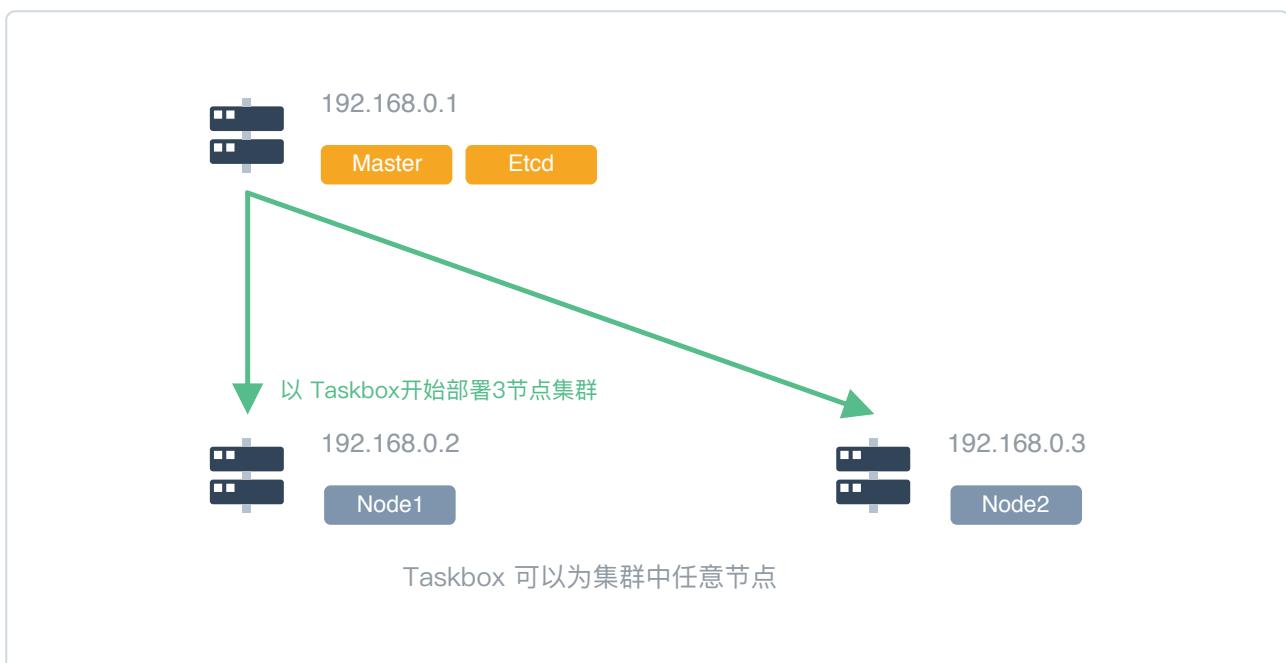
在 [安装说明](#) 已经介绍了 KubeSphere 集群架构是由管理节点 (Master) 和工作节点 (Node) 构成的，这 3 台主机分别部署 1 个 Master 节点和 2 个 Node 节点。

**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

### 在线版 (2.0.2)

1. 下载 [KubeSphere Advanced Edition 2.0.2](https://kubesphere.io/download/stable/advanced-2.0.2) 安装包至待安装机器，进入安装目录。

```
$ curl -L https://kubesphere.io/download/stable/advanced-2.0.2 > advanced-2.0.2.tar.gz &&
```

### 离线版 (2.0.2)

1. 下载 [离线安装包 \(2.0.2\)](#) 至待安装机器。

```
$ curl -L https://kubesphere.io/download/offline/advanced-2.0.2 > advanced-2.0.2.tar.gz &&
```

2. 编辑主机配置文件 `conf/hosts.ini`，为了对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，[all] 部分可参考配置文件 `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 [安装常见问题 – 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 hosts.ini 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。  
Node 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 相应替换为 node1 和 node2 的 root 用户密码。

#### 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\_user：默认的 SSH 用户名（非 root），例如 ubuntu
- ansible\_become\_pass：默认的 SSH 用户登录密码
- ansible\_ssh\_pass：待连接主机 root 用户的密码

- **[kube-master]** 和 **[etcd]**：应将主机名 “master” 填入 **[kube-master]** 和 **[etcd]** 部分，“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 Master 节点管理集群和 etcd 节点负责保存集群的数据。
- **[kube-node]**：将主机名 “node1”， “node2” 填入 **[kube-node]** 部分，作为 KubeSphere 集群的 node 节点。
- **[local-registry]**：离线安装包中该参数值表示设置哪个节点作为本地镜像仓库，默认值为 master 节点。建议给该节点的 **/mnt/registry** 单独挂盘（参考 fdisk 命令），使镜像可保存在持久化存储并节省机器空间。

3. 编辑 `conf/vars.yml` 配置文件，集群的存储以配置 QingCloud-CSI 插件对接 QingCloud 云平台块存储为例。

- 其中值带有 \* 号的值为必配项，参数释义详见 [存储配置说明 – QingCloud 云平台块存储](#)：
  - `qingcloud_access_key_id` 和 `qingcloud_secret_access_key`：通过 [QingCloud 云平台](#) 的右上角账户图标选择 **API 密钥** 创建密钥并下载获得（填写时仅粘贴单引号内的值）；
  - `qingcloud_zone`：根据您的机器所在的 Zone 填写，例如：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_csi_enabled`：是否使用 QingCloud-CSI 作为持久化存储，此处改为 true；
  - `qingcloud_csi_is_default_class`：是否设定为默认的存储类型，此处改为 true；
- 不带 \* 号的最后六行为可配项所以在示例中无需修改，当前默认配置了容量型硬盘，type 为 2(可挂载至任意类型主机)。

注意，安装前需要确认您 QingCloud 账号在当前 Zone 的容量型硬盘的配额是否大于 14。若需要使用其他类型的硬盘，也需要满足最低配额，修改配置可参考 [存储配置说明 – QingCloud 云平台块存储](#)。

`vars.yml` 配置存储示例：

```
# Access key pair can be created in QingCloud console
qingcloud_access_key_id: * Input your QingCloud key id *
qingcloud_secret_access_key: * Input your QingCloud access key *
# Zone should be the same as Kubernetes cluster
qingcloud_zone: * Input your Zone ID (e.g. pek3a, gd2) *
...
# QingCloud CSI
qingcloud_csi_enabled: * true *
qingcloud_csi_is_default_class: * true *
qingcloud_type: 2
qingcloud_maxSize: 5000
qingcloud_minSize: 100
qingcloud_stepSize: 50
qingcloud_fsType: ext4
qingcloud_disk_replica: 2
```

#### 说明：

- 网络、存储、GitLab、Harbor、负载均衡器插件等相关内容可在 `conf/vars.yml` 配置文件中修改或开启安装，其中网络的默认插件是 `Calico`。可按需修改相关配置项，未做修改将以默认参数执行；
- 支持存储类型：[QingCloud 云平台块存储](#)、[QingStor NeonSAN](#)、[NFS](#)、[GlusterFS](#)、[Ceph RBD](#)，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。

## 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 v1.13.5，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。

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. 进入安装目录，建议使用 root 用户执行 `install.sh` 安装脚本：

```
$ cd scripts  
$ ./install.sh
```

2. 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，若您已参考上述步骤配置了存储请输入 “yes” 开始安装。

```
#####  
KubeSphere Installer Menu  
#####  
* 1) All-in-one  
* 2) Multi-node  
* 3) Quit  
#####  
https://kubesphere.io/ 2018-07-08  
#####  
Please input an option: 2
```

3. 验证 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。

```
successssful!  
#####  
## Welcome to KubeSphere! ##  
#####  
  
Console: http://192.168.0.1:30880  
Account: admin  
Password: P@88w0rd  
  
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)** 若需要在外网访问，在云平台需要在端口转发规则中将**内网端口** 30880 转发到**源端口** 30880，然后在防火墙开放这个**源端口**，确保外网流量可以通过该端口。

例如在 QingCloud 平台配置端口转发和防火墙规则，则可以参考 [云平台配置端口转发和防火墙](#)。

**(3)** 安装成功后，浏览器访问对应的 URL，如 `http://[$公网IP]:30880`，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，**登录后请立即修改默认密码**。参阅 [快速入门](#) 帮助您快速上手 KubeSphere。



注意：登陆 Console 后请在 ”集群状态“ 查看服务组件的监控状态，待所有组件启动完成后即可开始使用，通常所有服务组件都将在 15 分钟内启动完成。

A screenshot of the KubeSphere cluster status page. At the top left, it shows "你好 admin" and "超级管理员". On the left sidebar, there are three items: "企业空间" (1), "项目" (11), "DevOps 工程" (0), and "账号管理" (1). In the center, there's a "集群状态" section with a blue circular icon showing "3/3" and text "节点在线状态 在线节点: 3 全部节点: 3". To the right, there's a table with four rows, each representing a service component. A red arrow points to the second row. The table data is as follows:

KUBESPHERE	7/7	kubernetes	6/6
OPENPITRIX	13/13	Istio	11/11
Monitoring	6/6	Logging	3/3

# 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。

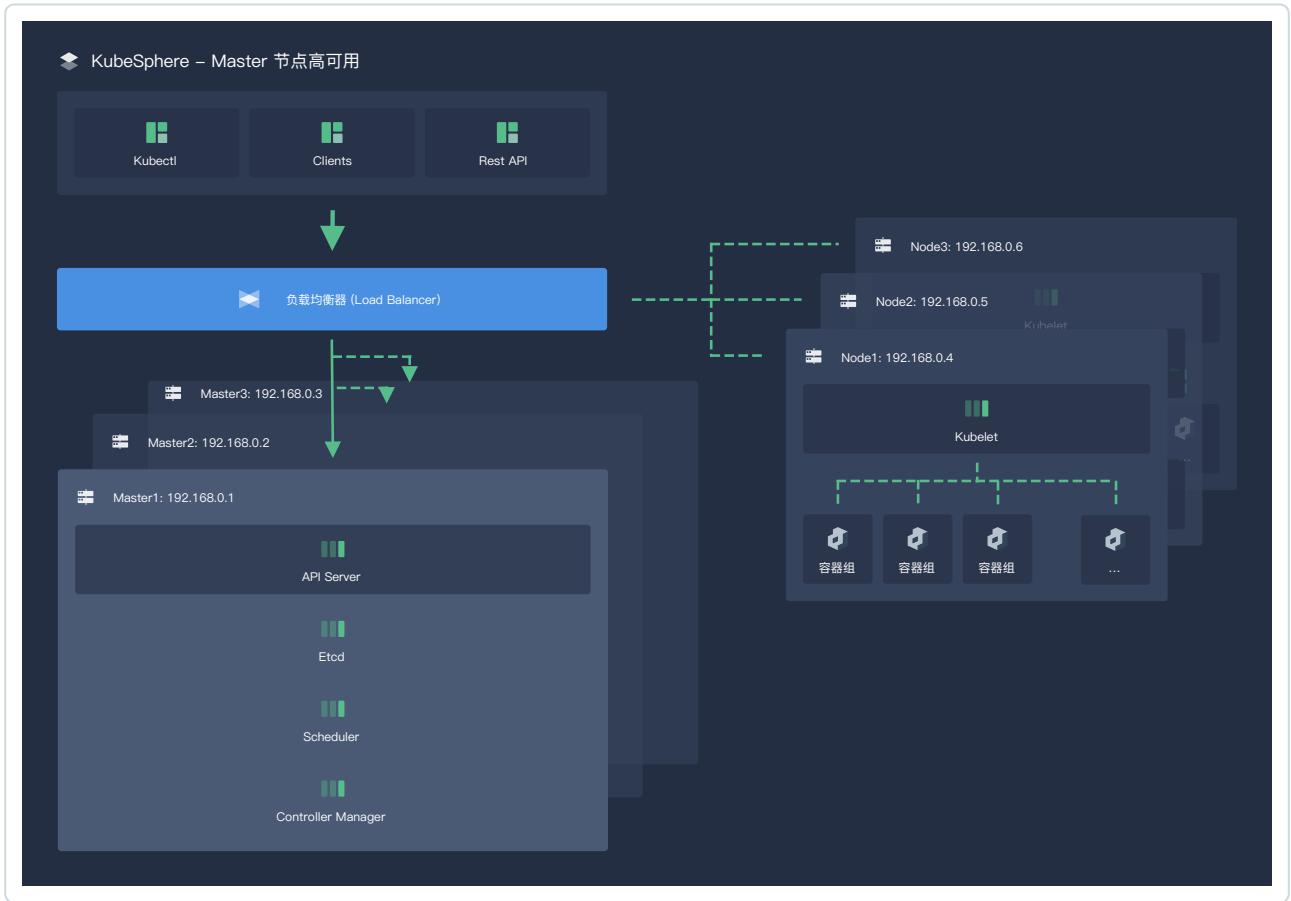
而 etcd 作为一个高可用键值存储系统，整个集群的持久化数据，则由 kube–apiserver 处理后保存到 etcd 中。etcd 节点至少需要 1 个，但部署多个 etcd (奇数个) 能够使集群更可靠。本文档以配置 [QingCloud 云平台](#) 的 [负载均衡器 \(Load Balancer\)](#) 为例，引导您如何配置高可用的 master 节点，并说明如何配置和部署高可用的 etcd 集群。

## 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。

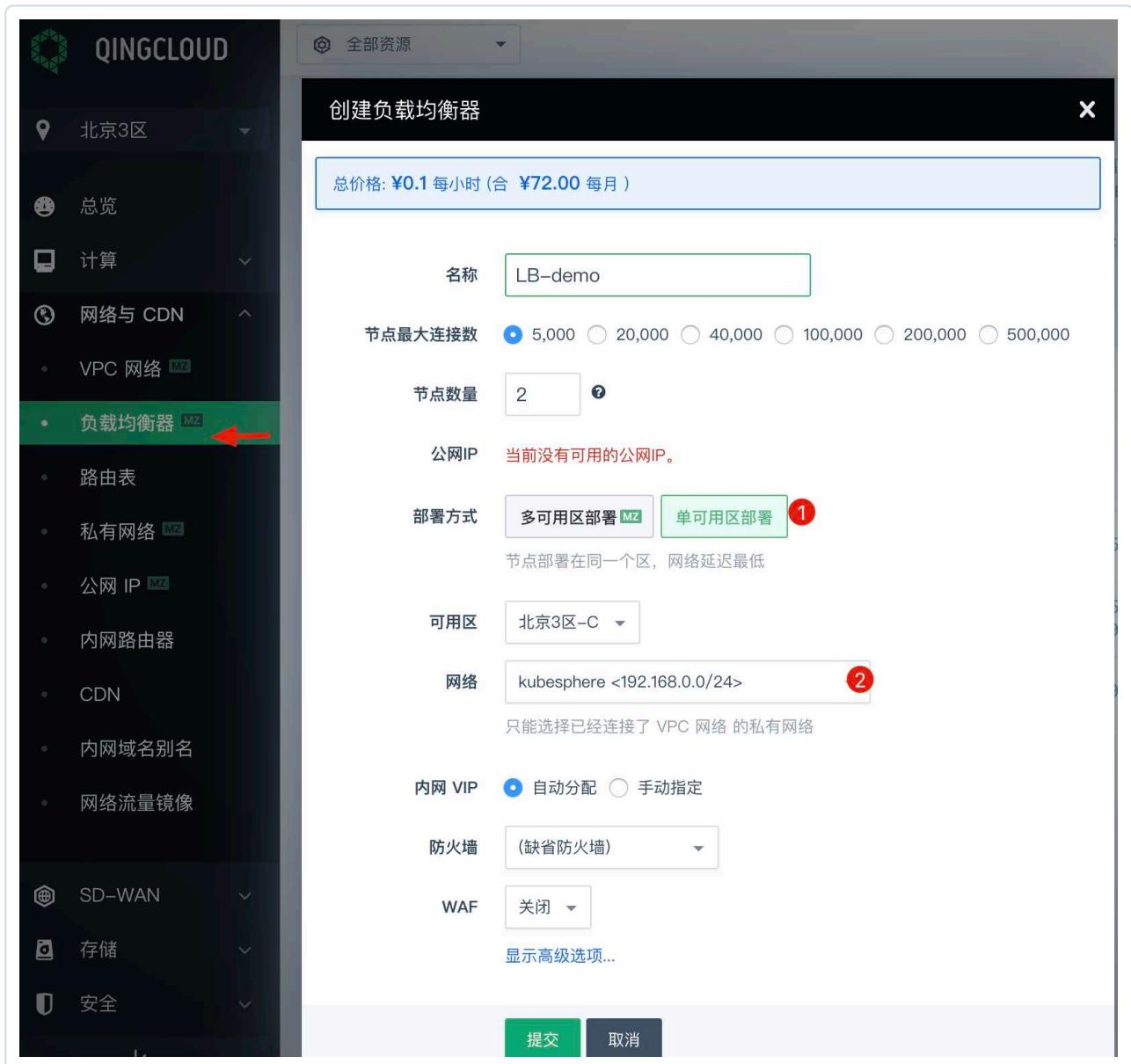


## 准备负载均衡器

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。

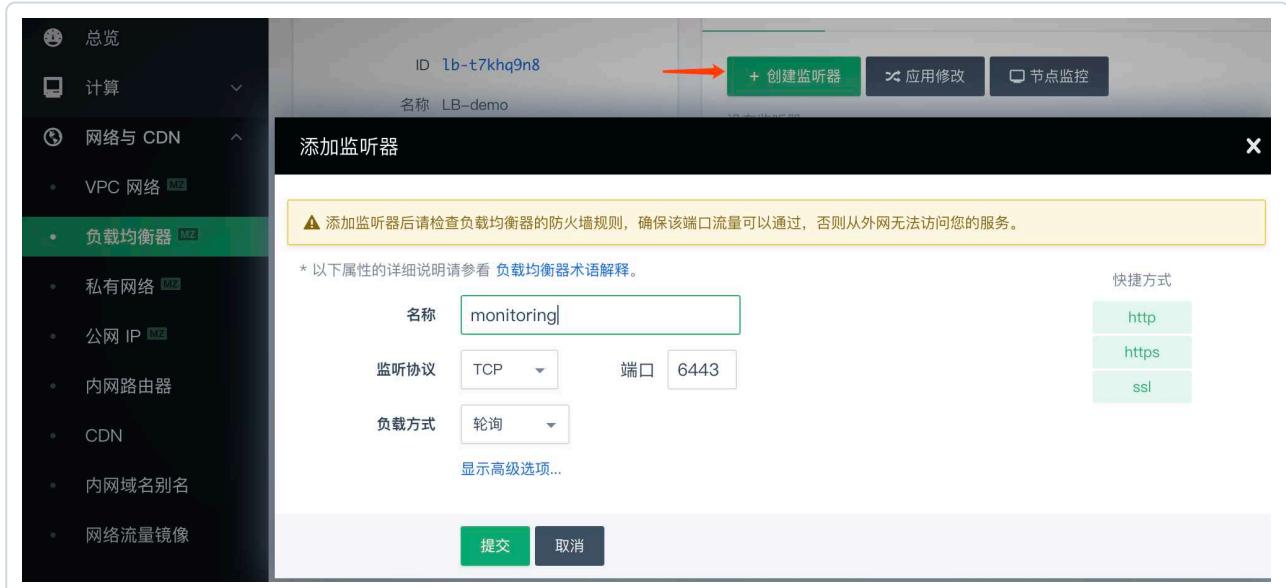


## 第二步：创建监听器

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 monitoring listener configuration page. At the top, there are tabs for '监听器' (Listener), '图形化' (Graphic), and '配置备份' (Configuration Backup). A yellow box highlights the message '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), a green '应用修改' (Apply Modification) button with a red arrow pointing to it, and '节点监控' (Node Monitoring). The main area displays the '监听器 : monitoring (ID: lbb-gqwq3c39)' section. It shows the listener protocol as TCP and port as 6443. A table lists three hosts:

名称	状态	主机 / 路由器 / IP	端口	权重	转发策略	监控	操作
(ID: lbb-flyo2rbr)	不可用	master2(i- <b>1</b> - <b>2</b> - <b>3</b> - <b>4</b> - <b>5</b> - <b>6</b> - <b>7</b> - <b>8</b> - <b>9</b> - <b>10</b> - <b>11</b> - <b>12</b> )	6443	1	不支持	查看	禁用
(ID: lbb-wzvt3bzd)	不可用	master1(i- <b>1</b> - <b>2</b> - <b>3</b> - <b>4</b> - <b>5</b> - <b>6</b> - <b>7</b> - <b>8</b> - <b>9</b> - <b>10</b> - <b>11</b> - <b>12</b> )	6443	1	不支持	查看	禁用
(ID: lbb-yavdw8yv)	不可用	master3(i- <b>1</b> - <b>2</b> - <b>3</b> - <b>4</b> - <b>5</b> - <b>6</b> - <b>7</b> - <b>8</b> - <b>9</b> - <b>10</b> - <b>11</b> - <b>12</b> )	6443	1	不支持	查看	禁用

## 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 [安装常见问题 - 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
- 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 和 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，可参考 [部署 Ceph RBD 存储服务端](#) 或 [部署 GlusterFS 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 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>

名称	版本	参考
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，在配置文件的相应部分参考示例或注释修改对应参数，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](#)）。

### 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，用户在 QingCloud 云平台可创建访问模式（Access Mode）为 **单节点读写（ReadWriteOnce）** 的存储卷并挂载至工作负载，包括以下几种类型：

- 容量型
- 基础型
- SSD 企业型
- 超高性能型
- 企业级分布式块存储 NeonSAN

**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。**

在安装 KubeSphere 时配置 QingCloud-CSI 插件的参数说明如下三个表所示，安装配置 QingCloud-CSI 插件和 QingCloud 负载均衡器插件都需要下载 API 密钥来对接 QingCloud API。

QingCloud-API	Description
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_host	QingCloud 云平台 api 地址，例如 <a href="#">api.qingcloud.com</a> (若对接私有云则以下值都需要根据实际情况填写)
qingcloud_port	API 请求的端口，默认 https 端口 (443)
qingcloud_protocol	网络协议，默认 https 协议
qingcloud_uri	URI 路径，默认值 iaas
qingcloud_connection_retries	API 连接重试时间 (默认 3 秒)
qingcloud_connection_timeout	API 连接超时时间 (默认 30 秒)

在 `vars.yml` 中完成上表中的 API 相关配置后，再修改 QingCloud-CSI 配置安装 QingCloud 块存储插件。

QingCloud-CSI	Description
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_type	QingCloud 云平台硬盘的类型

QingCloud-CSI	Description
	<ul style="list-style-type: none"> <li>* 性能型是 0</li> <li>* 容量型是 2</li> <li>* 超高性能型是 3</li> <li>* 企业级分布式块存储 NeonSAN 是 5</li> <li>* 基础型是 100</li> <li>* SSD 企业型是 200</li> </ul> <p>详情见 <a href="#">QingCloud 官方文档</a></p>
qingcloud_minSize, qingcloud_maxSize	即单块硬盘的最小容量和最大容量，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
disk_replica	硬盘的副本策略，支持单副本和多副本，1 表示单副本，2 表示多副本

## 硬盘类型与主机适配性

	性能型 硬盘	容量型 硬盘	超高性能型 硬盘	NeonSAN 硬盘	基础型 硬盘	SSD 企业型 硬盘
性能型主机	√	√	—	√	—	—
超高性能型 主机	—	√	√	√	—	—
基础型主机	—	√	—	√	√	—
企业型主机	—	√	—	√	—	√

## 各区应设置的 minSize, maxSize 和 stepSize 参数

下表中的值对应的格式为：qingcloud\_minSize – qingcloud\_maxSize, qingcloud\_stepSize。

	性能型硬盘	容量型硬盘	超高性能型硬盘	NeonSAN硬盘	基础型硬盘	SSD企业型硬盘
北京2区	10 – 1000, 10	100 – 5000, 50	10 – 1000,10	–	–	–
北京3区-A	10 – 2000, 10	100 – 5000, 50	10 – 2000,10	–	–	–
广东1区	10 – 1000, 10	100 – 5000, 50	10 – 1000,10	–	–	–
上海1区-A	–	100 – 5000, 50	–	100 – 50000, 100	10 – 2000, 10	10 – 2000, 10
亚太1区	10 – 1000, 10	100 – 5000, 50	–	–	–	–
亚太2区-A	–	100 – 5000, 50	–	–	10 – 2000, 10	10 – 2000, 10

## QingCloud 各类型块存储的最低配额

注意，使用 QingCloud 云平台块存储作为存储服务，安装前需要确保用户账号在当前 Zone 资源配额满足最低要求。在没有配置安装 GitLab 和 Harbor 的前提下，Multi-node 安装最少需要 14 块硬盘，请参考以下的最低配额表核对您账号的存储配额，若硬盘数量和容量配额不够请提工单申请配额。

注意，GitLab 和 Harbor 作为可选安装项，若安装前需要配置则需要考虑硬盘数量和容量的配额是否满足要求：Harbor 安装需要额外挂载 5 块硬盘，GitLab 安装需要额外挂载 4 块硬盘，若 KubeSphere 部署在云平台则需要考虑硬盘数量是否满足配额要求。

最低配额 \ 硬盘类型	性能型硬盘	容量型硬盘	超高性能型硬盘	NeonSAN硬盘	基础型硬盘	SSD企业型硬盘
块数 (块) / 容量 (GB)	14 / 230	14 / 1400	14 / 230	14 / 1400	14 / 230	14 / 230

## 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 物理服务端后，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

[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>

## 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
<code>local_volume_provisioner_enabled</code>	是否使用 local volume 作为持久化存储，是：true；否：false
<code>local_volume_provisioner_storage_class</code>	存储类型的名称，默认：local
<code>local_volume_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](#)。

# 集群组件配置释义

用户在获取 installer 并解压至目标安装机器后，如果需要查看或修改存储、网络、组件版本、可选安装项（如 GitLab、Harbor）、外部负载均衡器、Jenkins、邮件服务器等配置参数时，可参考以下说明进行修改，本文档对 installer 中的安装配置文件 `conf/vars.yml` 进行说明，简单介绍每一个字段的意义。

## 负载均衡器插件与 QingCloud CSI 配置

配置文件 `conf/vars.yml` 中的前 9 项 `qingcloud` 开头的配置项是与 QingCloud 云平台负载均衡器插件和 QingCloud CSI 块存储插件相关的公共配置参数，第 10、11 项是启用安装 QingCloud 负载均衡器插件和私有网络配置，释义分别在 [安装负载均衡器插件](#) 与 [存储配置说明 – QingCloud 云平台块存储](#)。

## 存储相关配置

Installer 默认使用 local 类型的存储方便 all-in-one 模式进行安装，若使用 Multi-node 进行安装，则需要配置持久化存储作为存储服务端再执行安装，在 `conf/vars.yml` 支持配置 QingCloud CSI、Ceph RBD、NFS Client、NeonSAN CSI、GlusterFS 等，存储配置相关参数释义请参考 [存储配置说明](#)。

## 集群组件相关配置

### 参数说明：

参数	含义
<code>ks_version</code>	KubeSphere 版本号
<code>kube_version</code>	Kubernetes 版本号
<code>etcd_version</code>	etcd 版本号
<code>openpitrix_version</code>	OpenPitrix 版本号
<code>ks_image_pull_policy</code>	默认 <code>IfNotPresent</code> ，表示优先使用本地镜像，还支持 <code>Always</code> （尝试重新下载镜像）和 <code>Never</code> （仅使用本地镜像）
<code>kube_network_plugin</code>	默认的网络插件（支持 Calico、Flannel）

参数	含义
kube_service_addresses	Service 网络 IP 地址段 (未被使用的地址段)
kube_pods_subnet	Pod 网络 IP 地址段 (未被使用的地址段)
kube_proxy_mode	kube-proxy 模式默认 ipvs (支持 ipvs, iptables)
kubelet_max_pods	单台机器默认 Pod 数量
dns_mode	DNS 模式, 建议 coredns
console_port	KubeSphere 控制台访问端口 (默认 30880)
disableMultiLogin	禁止同一用户多点登录, 默认 true 即禁用
loadbalancer_apiserver.address	外部负载均衡器地址
loadbalancer_apiserver.port	外部负载均衡器端口
apiserver_loadbalancer_domain_name	负载均衡器域名, 默认 lb.kubesphere.local
periodic_cleaning_time	weekly, Docker 自动清理镜像的周期
docker_registry_mirrors	默认 Docker 镜像仓库的 mirror 仓库, 可以加快镜像下载 (国外地区下载可将此参数注释)
etcd_backup_period	默认备份的周期为 30 分钟
keep_backup_number	默认保留最近 5 次备份的数据
etcd_backup_dir	默认备份的目录为 "/var/backups/kube_etcd"
prometheus_memory_size	Prometheus 内存请求大小
prometheus_volume_size	Prometheus 存储空间大小
keep_log_days	集群内置的 Elasticsearch 中日志保留时间, 默认是 7 天
kibana_enable	是否部署 Kibana (默认 false)
elasticsearch_volume_size	Elasticsearch 存储空间
EMAIL_SMTP_HOST	SMTP 邮件服务器地址
EMAIL_SMTP_PORT	SMTP 邮件服务器端口
EMAIL_FROM_ADDR	发件人邮箱地址

参数	含义
EMAIL_FROM_NAME	通知邮件名称
EMAIL_FROM_PASS	密码
EMAIL_USE_SSL	是否开启 SSL 配置
jenkins_memory_lim	Jenkins 内存限制
jenkins_memory_req	Jenkins 内存请求
Java_Opts	jvm 启动参数
JenkinsLocationUrl	jenkins 域名
harbor_enable	是否安装 Harbor
harbor_domain	Harbor 域名
gitlab_enable	是否部署 GitLab
gitlab_hosts_domain	GitLab 域名
sonarqube_enable	是否集成并开启 SonarQube, 默认安装内置的 SonarQube
sonar_server_url	对接已有的外部 SonarQube 地址 (如需集成安装则取消注释)
sonar_server_token	对接已有 SonarQube token (如需集成安装则取消注释)
nvidia_accelerator_enabled	是否开启 Nvidia GPU 加速
nvidia_gpu_nodes	hosts.ini 中要开启 GPU 加速的节点名称 (列表), 参考以下配置示例

## GPU 节点配置示例

注意, 在安装前可对 GPU 节点在 `vars.yml` 文件中进行设置, 例如在 `hosts.ini` 文件配置的两台工作节点 `node1` 是 CPU 节点, `node2` 是 GPU 节点, 那么在 `vars.yml` 仅需要在该处填写 `node2`, 注意 “-” 前面需缩进两格。

```
nvidia_accelerator_enabled: true
```

```
nvidia_gpu_nodes:
```

```
  - node2
```

# 安装负载均衡器插件

服务或应用路由如果通过 LoadBalancer 的方式暴露到外网访问，则需要安装对应的负载均衡器插件来支持，集群可以部署云平台的虚拟机或直接部署在物理机，因此分别需要不同类型的负载均衡器插件进行对接，KubeSphere 分别开发了 [QingCloud 云平台负载均衡器插件](#) 和适用于适用于物理机部署 Kubernetes 的 [负载均衡器插件 Porter](#)，并且已将它们在 GitHub 开源。

## 安装 QingCloud 云平台负载均衡器插件

如果在 QingCloud 云平台安装 KubeSphere，建议在 `conf/vars.yml` 中配置 QingCloud 负载均衡器插件相关参数，installer 将自动安装 [QingCloud 负载均衡器插件](#)。目前 QingCloud 负载均衡器插件只支持多节点集群的安装。

### 前提条件

已有 QingCloud 云平台账号，并下载 installer 至目标机器。

### 安装步骤

在安装 KubeSphere 前参考如下提示在 `conf/vars.yml` 中进行配置：

#### 1. 其中值带有 \* 号的值为必配项：

- `qingcloud_access_key_id` 和 `qingcloud_secret_access_key`：通过 [QingCloud 云平台](#) 的右上角账户图标选择 **API 密钥** 创建密钥获得；
- `qingcloud_zone`：根据您的机器所在的 Zone 填写，例如：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）；

最后六行为可配置项，适用于私有云场景下的配置，请根据实际情况配置。

```
# Access key pair can be created in QingCloud console
qingcloud_access_key_id: * Input your QingCloud key id *
qingcloud_secret_access_key: * Input your QingCloud access key *
# Zone should be the same as Kubernetes cluster
qingcloud_zone: * Input your Zone ID *
# QingCloud IaaS platform service url.
qingcloud_host: api.qingcloud.com
qingcloud_port: 443
qingcloud_protocol: https
qingcloud_uri: /iaas
qingcloud_connection_retries: 3
qingcloud_connection_timeout: 30
```

2. 设置 `qingcloud_lb_enable` 为 `true`, 启用并安装 QingCloud 负载均衡器插件。其中 `qingcloud_vxnet_id` 为可配项, 此参数配置适用于私有云场景, 若填写后将为集群的服务及应用路由生成内网 IP, 服务仅支持集群内部访问, 若需要配置则填写待安装机器所在的私有网络 ID。

```
## QingCloud LoadBlancer Plugin
qingcloud_lb_enable: true
qingcloud_vxnet_id: SHOULD_BE_REPLACED
```

完成以上步骤的配置后, 可继续参考安装指南完成其他配置并执行安装。

## 如何使用 QingCloud LB 插件

在「项目设置」→「外网访问」下, 点击「设置网关」, 选择 LoadBalancer。

LB 插件可通过添加 Annotation 来配置使用, 目前支持以下三种配置方式, 参考 [公网 IP 配置](#)。

- **手动配置 EIP:** 添加一条 Annotation 的 key 是 `service.beta.kubernetes.io/qingcloud-load-balancer-eip-ids`, 值是 `公网IP 的 ID`;
- **自动获取 EIP:** 需添加一条 Annotation 的 key `service.beta.kubernetes.io/qingcloud-load-balancer-eip-source`, 注意自动获取 EIP 支持以下三种方式 (对应不同的值):

- 自动获取当前账户下处于可用的 EIP，如果找不到返回错误，值设置为 `use-available`；
- 自动获取当前账户下处于可用的 EIP，如果找不到则申请一个新的，值设置为 `auto`；
- 不管账户下有没有可用的 EIP，申请一个新 EIP，值设置为 `allocate`；
- **配置多个服务 (Service) 共享一个公网 IP：**由于 EIP 是稀缺资源，QingCloud-LB 插件提供了多个 Service 共享一个 EIP 的模式，使用这个模式有一定限制，详见 [配置多个Service共享一个EIP](#)。

以下仅以其中一种配置方式“自动获取当前账户下处于可用的 EIP，如果找不到则申请一个新的，值设置为 `auto`”演示如何在 KubeSphere 配置使用。

**注意，示例中的这种方式可能会造成同一账号下其他用户空闲的公网 IP 被使用。**



1、如图在示例中添加一条注解，保存后即可允许 LB 插件自动为网关创建和绑定公网 IP，同时在 QingCloud 云平台创建负载均衡器。您在创建和使用应用路由、服务时即可为其自动生成公网 IP。

```
service.beta.kubernetes.io/qingcloud-load-balancer-eip-source : auto
```

2、等待 2 分钟左右负载均衡器将在云平台自动创建完毕并自动绑定公网 IP 至网关。

The screenshot shows the KubeSphere project management interface. On the left is a sidebar with navigation items: 概览 (Overview), 应用 (Application), 工作负载 (Workload), 存储卷 (Storage), 网络与服务 (Network & Services), 监控与告警 (Monitoring & Alerting), 配置中心 (Config Center), and 项目设置 (Project Settings). The main area displays the 'demo-namespace' project details, including the project name, owner (admin), space (demo), creation time (2019-06-05 22:25:00), and a 'Create Time' button. Below this, the 'External Access' section shows a LoadBalancer icon, an IP address (139.198.128.11), and a 'Not Enabled' application governance status. The 'Annotations' section lists two entries: 'service.beta.kubernetes.io/qingcloud-load-balancer-ip-source: auto' and 'service.beta.kubernetes.io/qingcloud-load-balancer-type: 0'.

3、在部署应用或创建应用路由、服务后，即可自动使用该公网 IP 或域名来暴露服务，方便用户快速访问。

The screenshot shows the KubeSphere application component interface for the 'bookinfo' application. It includes tabs for 应用组件 (Application Component), 流量治理 (Traffic Governance), 灰度发布 (Gradual Release), Tracing, and 事件 (Events). The '应用路由' (Application Route) section displays the 'bookinfo-ingress' route, which has a hostname of 'productpage.demo-namespace.139.198.128.11.nip.io' and a URL of 'http://productpage.demo-namespace.139.198.128.11.nip.io/'. A 'Click to Visit' button is also present.

## 安装负载均衡器插件 Porter

Porter 是一款适用于物理机部署 Kubernetes 的负载均衡器，该负载均衡器使用物理交换机实现，利用 BGP 和 ECMP 从而达到性能最优和高可用性，Porter 是一个提供用户在物理环境暴露服务和在云上暴露服务一致性体验的插件。

安装和使用 Porter 请参考 [Porter 项目](#)。

## 安装内置 Harbor

KubeSphere Installer 集成了 Harbor 版本为 1.7.5，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。

注意，Harbor 安装需要额外挂载 5 块硬盘，若 KubeSphere 部署在云平台则需要考虑硬盘数量是否满足配额要求，若硬盘数量或容量配额不够则需要提工单申请提高配额。

### 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

#### Docker 登录 Harbor

登录 Harbor 镜像仓库。

```
$ 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

## 使用 Harbor 镜像仓库

关于如何制作镜像、打包上传镜像以及 Dockerfile 的使用详情，请参考 [Docker 官方文档](#)。

KubeSphere 安装成功后，本地已经有了 nginx:1.14-alpine 镜像，因此演示如何在本地给示例镜像 nginx 打 tag 后推送到 Harbor 镜像仓库的 library 项目中：

1、给本地 nginx 镜像打上 1.14-alpine 的 tag。

```
docker tag nginx:1.14-alpine harbor.devops.kubesphere.local:30280/library/nginx:1.14-alpine
```

2、推送至 Harbor 镜像仓库的 library 项目中。

```
docker push harbor.devops.kubesphere.local:30280/library/nginx:1.14-alpine
```

镜像推送成功即可通过浏览器登录 Harbor，验证推送的结果。

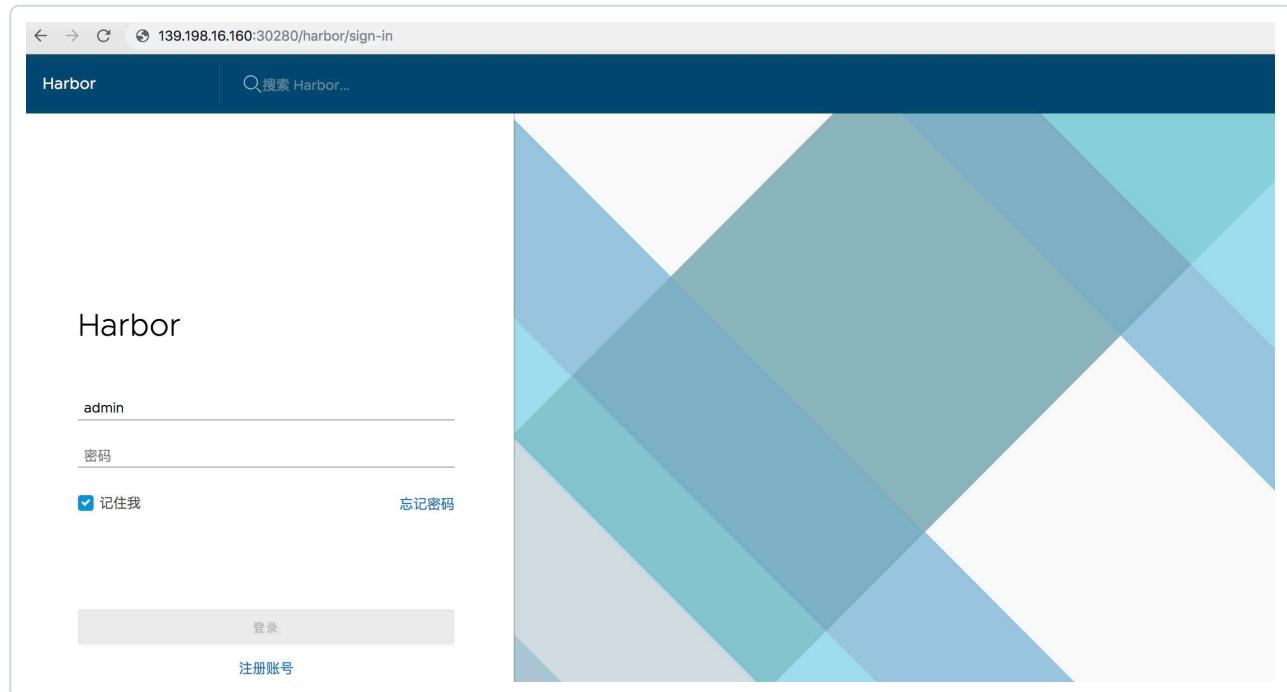
## 浏览器访问 Harbor

KubeSphere 安装成功后，即可在浏览器访问 Harbor 镜像仓库。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 `{$IP}:{$NodePort}` 如 <http://139.198.16.160:30280> 访问 Harbor 登录页面。

**注意：**若需要在外网访问，可能需要绑定公网 EIP 并配置端口转发和防火墙规则，端口转发需要将内网端口 30280 转发到源端口 30280，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，然后才可以通过 [http://\[\\$公网 IP\]:{\\$NodePort}](http://[$公网 IP]:{$NodePort}) 访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。提示：在浏览器中还可通过域名访问 Harbor，在本地 /etc/hosts 添加一行记录 139.198.16.160 harbor.devops.kubesphere.local，即可通过 <http://harbor.devops.kubesphere.local:30280> 访问。

1、输入默认的管理员用户名和密码 `admin / Harbor12345` 登录 Harbor。

**提示：**其它用户登录的账号密码与 KubeSphere 的 LDAP 用户账户体系一致。



2、登录成功后，点击进入 library 项目。

3、可以看到 library 项目下已有了 nginx 镜像。

The screenshot shows the Harbor interface. On the left, there's a sidebar with '项目' (Project) selected, showing 'library' as the active project. The main area displays a table of images. One row is highlighted with a red box, showing the image name 'library/nginx'. The search bar at the top right contains 'nginx'.

## KubeSphere 中使用 Harbor

### KubeSphere 添加 Harbor

**提示：**以下需要在企业空间下的项目中添加镜像仓库，若还未创建企业空间和项目，请参考[多租户管理快速入门](#)。

登录控制台，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

创建密钥

编辑模式

基本信息

名称 \* harbor  
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 内置 Harbor 镜像仓库  
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：填写 Harbor 仓库信息

2.1. 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

编辑模式

基本信息

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

用户名 \* admin

邮箱

密码 \*

2.2. 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。

The screenshot shows the KubeSphere interface for managing certificates. On the left, there's a sidebar with project navigation. The main area is titled 'Certificates' and displays a table of certificates. The first row, which is highlighted by a red arrow, represents a certificate for the 'Harbor Registry'.

名称	类型	Config Number	创建时间
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

## 使用 Harbor 中的镜像

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。

以创建 Deployment (部署) 为例展示如何使用镜像仓库来拉取仓库中的镜像。比如上述步骤已经将镜像 `nginx:1.14-alpine` 推送到了 library 项目，因此以下将演示如何创建工作负载并使用 Harbor 中的示例镜像 `nginx:1.14-alpine`。

1、上述步骤中已将镜像仓库添加到了项目中，因此进入当前项目，选择「工作负载」→「部署」，点击「创建」。

2、在弹窗中填写基本信息然后点击「下一步」，在容器组模板中需要先选择镜像仓库，镜像为 `harbor.devops.kubesphere.local:30280/library/nginx:1.14-alpine`，对应的格式为 `镜像仓库地址/项目名称/镜像名称:tag`，填写后点击「保存」，然后一直点击下一步直至创建，待创建成功后即可使用该镜像。

**提示：**关于如何创建工作负载的详细步骤说明，请参考快速入门系列文档。

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

选择已有镜像部署容器  
从公开或者私有镜像仓库中拉取镜像

镜像 \*

harbor.devops.kubesphere.local:30280/library/nginx:1.14-alpine

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。

提示：关于 Harbor 的使用详见 [Harbor 官方文档](#)。

# 安装内置 GitLab

KubeSphere Installer 内置的 Gitlab (版本为 v11.8.1) 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，方便用户对代码仓库的管理，仅需 [安装前](#) 在配置文件 `conf/vars.yml` 中简单配置即可。具体可参考以下步骤安装和访问 GitLab。

**注意：目前 GitLab 安装暂不支持块存储，安装前需预先配置 NAS 或 GlusterFS 作为集群的存储服务端。**

- 若您在青云云平台部署 KubeSphere 请参考 [QingCloud vNAS](#)。
- 若您自行部署的 NFS 或 GlusterFS 服务端，在 `conf/vars.yml` 的配置请参考 [存储配置说明 – GlusterFS 或 NFS](#)。

## 第一步：修改配置文件

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

```
# GitLab deployment
gitlab_enable: true
gitlab_hosts_domain: 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 GitLab。

## 第二步：配置 GitLab 访问

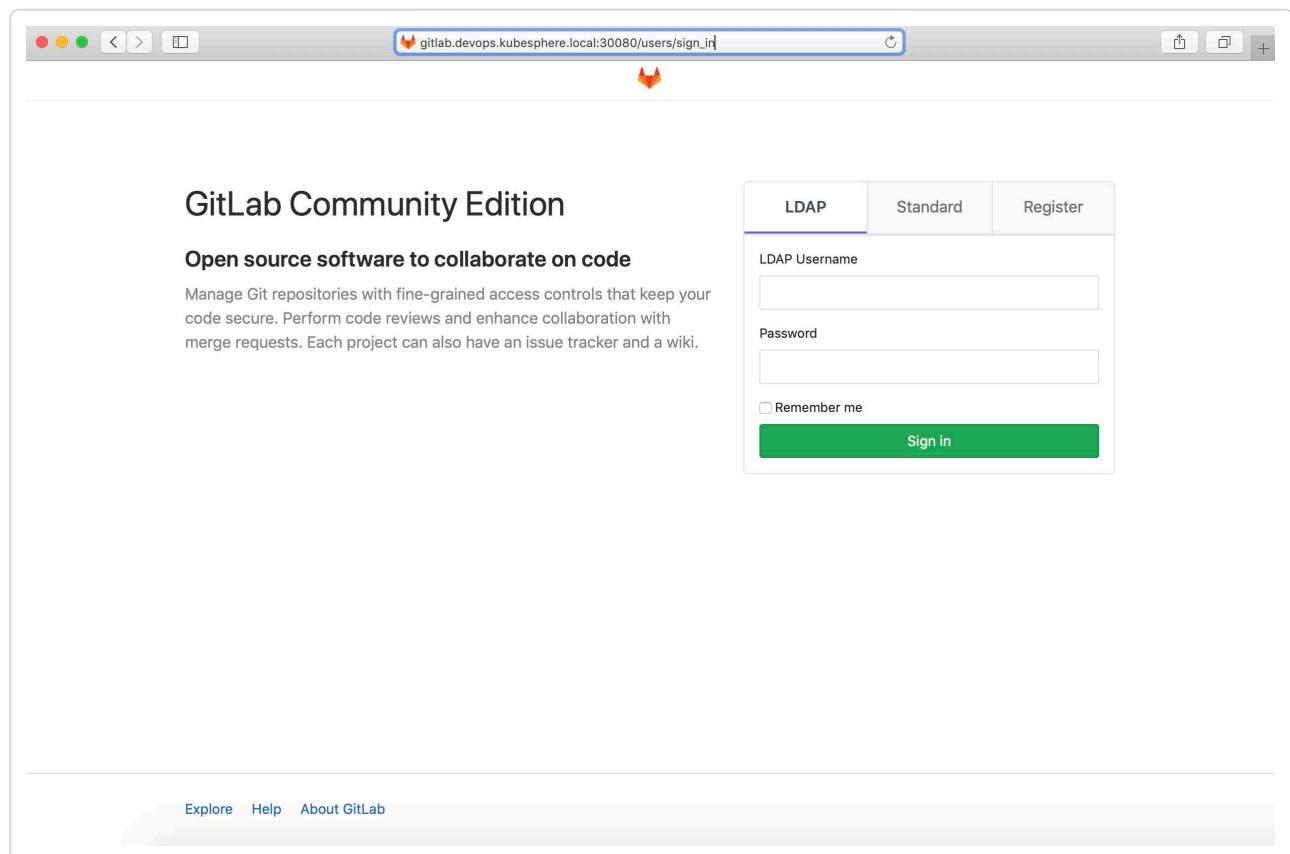
2、KubeSphere 和 GitLab 都安装完成后，若需要在集群外部访问 GitLab，请在本地的 `/etc/hosts` 文件中参考如下示例添加一行记录，然后即可在浏览器访问 GitLab。

```
# {$公网 IP} {$GitLab 域名}
139.198.10.10 gitlab.devops.kubesphere.local
```

**注意：**在外网访问 GitLab，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 GitLab 的端口 30080 (HTTP) 保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。

**提示：**若需要在外网使用 GitLab， HTTPS 端口 30443 和 SSH 端口 (输入 `kubectl get svc -n kubesphere-devops-system | grep 22` 查看) 也需要保证外网流量可以通过这些端口。

3、在浏览器中可以通过 `{$域名}:{$NodePort}` 即 `http://gitlab.devops.kubesphere.local:30080` 访问 GitLab 登录页面。默认的 GitLab 用户名和密码为 `admin / P@88w0rd`。



## 使用 GitLab 示例

本示例以 `devops-java-sample` 为例展示如何从 GitHub 导入项目至 GitLab。

1、请先将 GitHub 仓库 `devops-java-sample` Fork 至您个人的 GitHub 仓库。

kubesphere / devops-java-sample

Code Issues Pull requests Projects Wiki Insights

A sample for devops on kubesphere

29 commits 1 branch 0 releases 2 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find File Clone or download

souseen Merge pull request #1 from runzexia/runzexia-patch-1 ... Latest commit 8beeafa 23 hours ago

.s2i update struct 10 days ago  
artifacts/m2 update test report 8 days ago  
configuration update cmd sonar 8 days ago  
deploy modify ns & package off 2 days ago  
src Update HelloWorldController.java 23 hours ago

)

2、使用 Kubesphere 默认的用户名和密码 `admin / P@88w0rd` 登陆 GitLab 后，选择 `Create a project`。

Welcome to GitLab

Code, test, and deploy together

Create a project Projects are where you store your code, access issues, wiki and other features of GitLab.

Create a group Groups are the best way to manage projects and members.

Learn more about GitLab Take a look at the documentation to discover all of GitLab's capabilities.

3、选择「Import project from GitHub」。

GitLab Projects Activity Milestones Snippets

Search or jump to...

New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

Tip: You can also create a project from the command line. Show command

Import project

Import project from

- Blank project
- Create from template
- Import project

Import project from

- GitLab export
- GitHub**
- Bitbucket Cloud
- Bitbucket Server
- GitLab.com
- Google Code
- Fogbugz

Gitea

git Repo by URL

Manifest file

4、按照提示加入个人 Token，Personal access token 可在 GitHub [Setting](#) 页面生成。然后可选择 GitHub repositories。

GitLab Projects Groups Activity Milestones Snippets

Search or jump to...

Projects > GitHub import

Import repositories from GitHub

To import GitHub repositories, you can use a [Personal Access Token](#). When you create your Personal Access Token, you will need to select the `repo` scope, so we can display a list of your public and private repositories which are available to import.

\*\*\*\*\*

List your GitHub repositories

Note: Consider asking your GitLab administrator to configure [GitHub integration](#), which will allow login via GitHub and allow importing repositories without generating a Personal Access Token.

5、选择 [devops-java-sample](#) 项目 Import 至 GitLab。

6、等待 Status 显示为 Done，即导入成功。

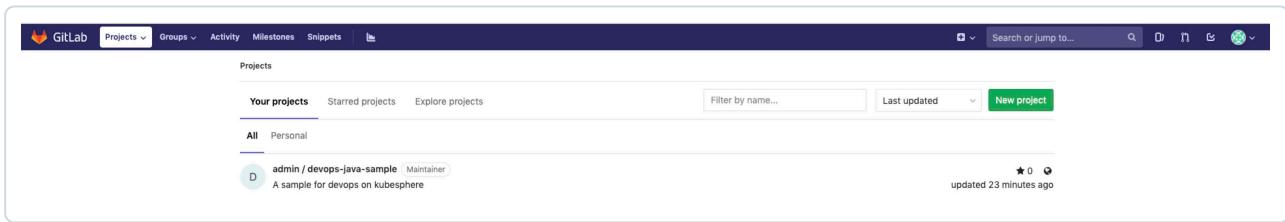
Import repositories from GitHub

Choose which repositories you want to import.

Import all repositories

From GitHub	To GitLab	Status
souleen/devops-java-sample	admin1/devops-java-sample	✓ Done

7、回到 Project 主页面，即可看到项目导入成功。



提示：关于 GitLab 的使用详见 [GitLab 文档](#)。

# 升级

KubeSphere 目前最新的版本 2.0.2 已发布，该版本修复了已知的 Bug，关于 2.0.2 版本的更新详情可参考 [Release Note – v2.0.2](#)。

若您已安装的环境为 1.0.x、2.0.0 或 2.0.1 版本，我们强烈建议您下载 2.0.2 版本的 Installer 并升级至最新的版本 2.0.2，最新的 Installer 支持将 KubeSphere 从 1.0.x 或 2.0.x 一键升级至 2.0.2，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。

## 如何升级

### 第一步：下载最新安装包

下载最新的 [KubeSphere Advanced v2.0.2](#) 安装包至任务执行机，进入安装目录。

```
$ curl -L https://kubesphere.io/download/stable/advanced-2.0.2 > advanced-2.0.2.tar.gz &&
```

### 第二步：修改配置文件

升级将默认读取 2.0.2 的 conf 目录下的配置文件，因此在升级前需要将原有安装包中 conf 目录下的配置文件中的参数都同步到 2.0.2 版本安装包的对应文件中，修改配置文件分以下两种情况（以下说明都以 2.0.1 作为示范）。

注意，在升级前请确保主机规格满足 2.0.2 的主机最低规格配置。请根据您的安装模式参考 [All-in-One 模式 – 准备主机](#) 或 [Multi-Node 模式](#) 主机规格表。

#### All-in-One

若 2.0.1 是以 **all-in-one** 模式安装的单节点集群，那么升级前在 2.0.2 中无需修改 [conf/hosts.ini](#) 文件，仅需要确认 2.0.1 的 [conf/vars.yml](#) 参数配置是否修改，若有修改则需要在 2.0.2 的对应文件中同步所有修改的参数。

例如，目前 2.0.2 中默认使用 [Local](#) 作为存储类型，如果您的 2.0.1 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 2.0.2 安装包的 `conf/vars.yml` 中也需要进行相应的设置（即与 2.0.1 的配置保持一致），参数释义详见 [存储配置参数](#)。

## Multi–Node

若 2.0.1 是以 **multi-node** 模式安装的多节点集群，那么升级前需将当前 Installer 中的 `conf/hosts.ini` 和 `conf/vars.yaml` 中的配置都同步到 2.0.2 的对应文件中：

- 将 2.0.1 的 `conf/hosts.ini` 中的主机参数配置覆盖至 2.0.2 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](#)。
- 选取 2.0.1 的 `conf/vars.yaml` 中所有修改过的参数配置项的值同步至 2.0.2 `conf/vars.yaml` 中的对应项。例如，2.0.1 配置使用的是 QingCloud 块存储、NFS、Ceph RBD 或 GlusterFS 这一类存储，那么在 2.0.2 安装包的 `conf/vars.yaml` 中也要进行相应的设置（即与 2.0.1 的配置保持一致），参数释义详见 [存储配置参数](#)。
- 注意，2.0.2 可选配置项如负载均衡器插件、邮件服务器、SonarQube 配置，在 Installer 的 `conf/vars.yaml` 中可按需进行安装。

## 第三步：开始升级

完成上述配置后，参考如下步骤进行升级：

3.1. 在 `kubesphere-all-advanced-2.0.2` 目录下进入 `/script` 目录，执行 `upgrade.sh` 脚本，建议使用 `root` 用户：

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

3.2. 在以下返回中输入 `yes` 开始升级：

```
ks_version: 2.0.1 to 2.0.2  
k8s_version: v1.12.5 to v1.13.5
```

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

3.3. 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。

# 访问内置 SonarQube 和 Jenkins 服务端

## 访问 SonarQube

[SonarQube](#) 是一个开源的代码分析软件，用来持续分析和检测代码的质量，支持检测 Java、C#、C、C++、JavaScript 等二十多种编程语言。通过 SonarQube 可以检测出项目中潜在的 Bug、漏洞、代码规范、重复代码、缺乏单元测试等问题，SonarQube 提供了 UI 界面进行查看和管理。KubeSphere 安装时默认内置了 SonarQube 服务，可参考如下步骤，访问内置 SonarQube。

### 第一步：查看 SonarQube 服务端口

执行如下命令，查看 SonarQube 服务端口，可以看到其端口为 [31359](#)。

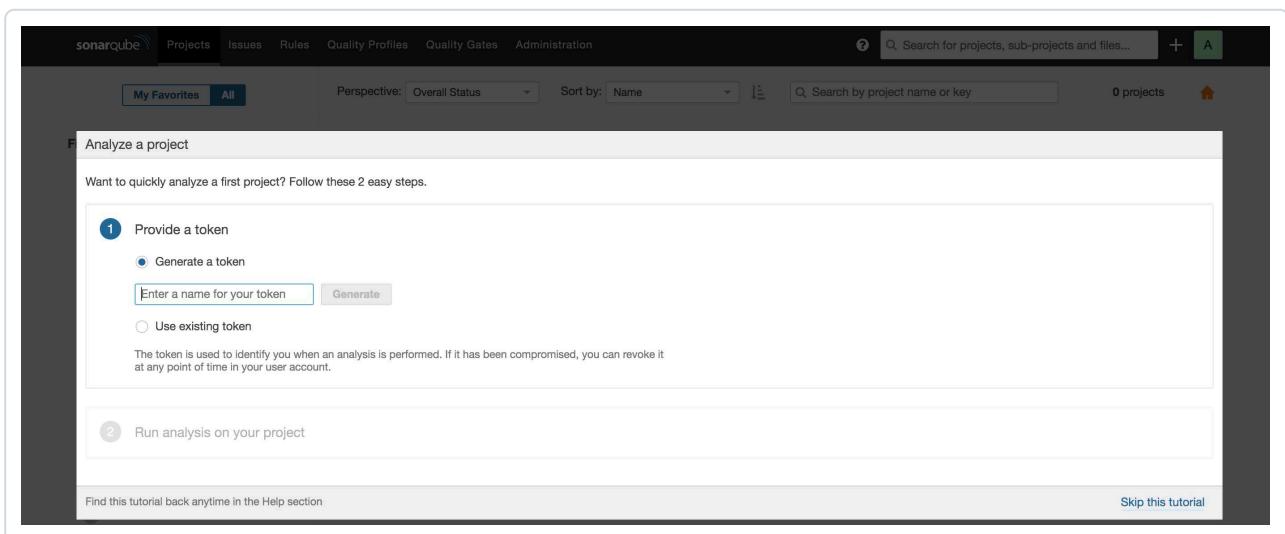
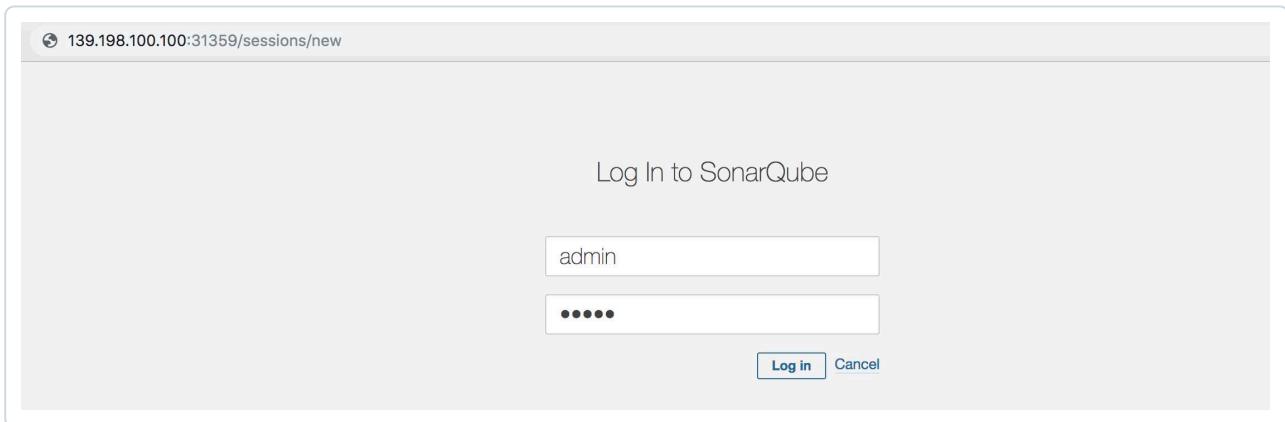
```
$ kubectl get svc -n kubesphere-devops-system | grep ks-sonarqube-sonarqube  
ks-sonarqube-sonarqube           NodePort    10.233.20.169 <none>      9000:31359/TCP
```

### 第二步：访问 SonarQube

若 KubeSphere 部署在云平台，需要在外网访问 SonarQube，在端口转发规则中将**内网端口** 31359 转发到**源端口** 31359，然后在防火墙开放这个**源端口**，确保流量能够通过该端口，然后通过 [http://{\\$公网 IP}:{\\$NodePort}](#) 进行访问。例如在 QingCloud 平台配置端口转发和防火墙规则，可参考 [云平台配置端口转发和防火墙](#)。

**说明：**若部署在私有环境，则可以在集群的任意节点通过 [http://{\\$节点 IP}:{\\$NodePort}](#) 进行访问。

如下，在浏览器中访问 SonarQube，初次登录的默认账号密码为 [admin / admin](#)。



## 创建 SonarQube Token

可参考 [创建 SonarQube Token](#)。

关于 SonarQube 的使用说明请参考 [SonarQube 官方文档](#)。

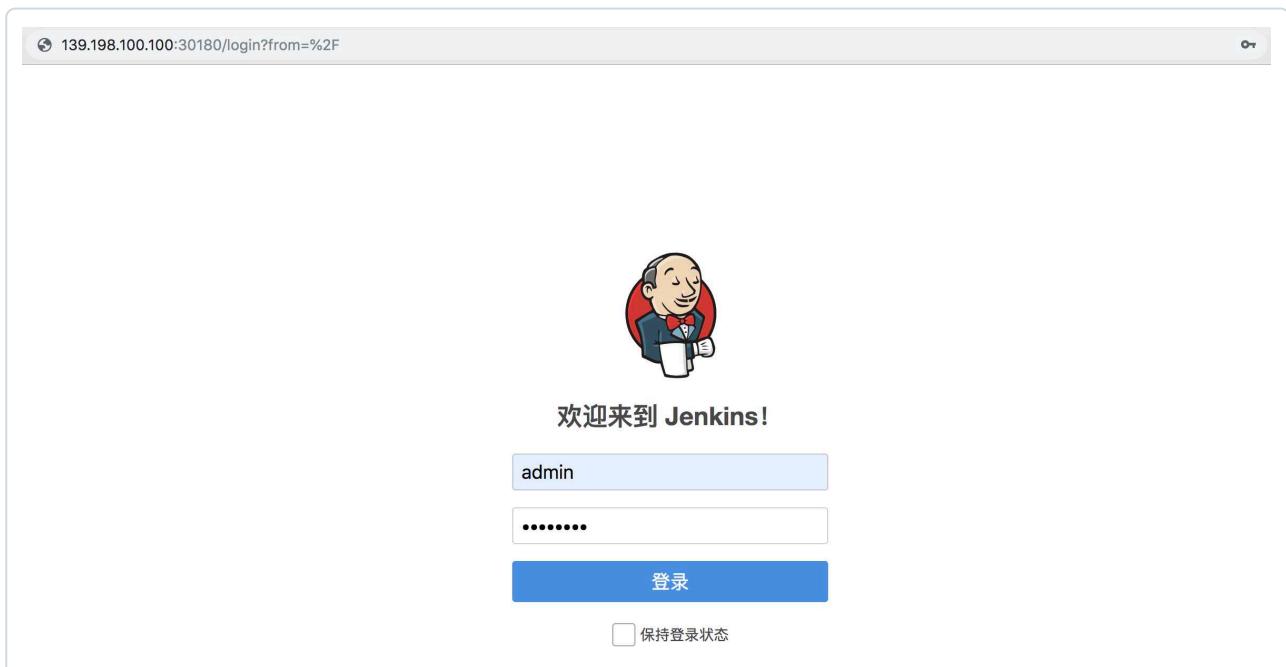
## 访问 Jenkins 服务端

Jenkins 是一款由 Java 开发的开源的持续集成工具，KubeSphere 内置的 Jenkins 服务端可参考如下步骤登录访问。

1、Jenkins Dashboard 服务暴露的端口 (NodePort) 默认为 [30180](#)，若在云平台部署 KubeSphere，则需要进行端口转发和添加防火墙规则，确保外网流量能够正常通过该端口。

2、然后访问公网 IP 和端口号即 `http://${EIP}:${NODEPORT}`，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（初始密码为 `P@88w0rd`）登录 Jenkins Dashboard。

**说明：**若部署在私有环境，则可以在集群的任意节点通过 `http://[$节点 IP]:30180` 进行访问。



关于 Jenkins 的使用说明请参考 [Jenkins 官方文档](#)。

# 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

KubeSphere Advanced 2.0.0 相比较 1.0.1 提供了更强大的功能，比如新增微服务治理、Source-to-Image、日志收集与查询、监控告警、更多监控指标、代码质量检查工具 SonarQube 等，并提供适用于物理机部署 Kubernetes 的负载均衡器 [Porter](#)，满足企业更复杂的业务场景和需求。

本文旨在通过多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。建议参考示例文档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](#)。

## 多租户管理快速入门

若您是初次安装使用 KubeSphere 的集群管理员用户，请先参考 [多租户管理快速入门](#)，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。

## 应用路由与服务示例

KubeSphere 在项目中为用户项目内置了一个全局的负载均衡器，即应用路由控制器 (Ingress Controller)，为了代理不同后端服务 (Service) 而设置的负载均衡服务，用户访问 URL 时，应用路由可以把请求转发给不同的后端服务，[服务与应用路由示例](#) 在 KubeSphere 创建相关资源来说明这个应用路由的示例。

## 工作负载快速入门

- [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。

- [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。

- [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。

- [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。

- [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

- [Source to Image](#)

Source to Image(S2I) 是一个创建 Docker 镜像的工具。它可以通过将源代码放入一个单独定义的负责编译源代码的 Builder image 中，来将编译后的代码打包成 Docker 镜像。

- [基于Spring Boot项目构建流水线](#)

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、代码质量检测、构建镜像、推送和发布版本，最终示例 Web 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。

- [Harbor + GitLab 流水线示例\(离线版\)](#)

本示例演示如何在离线环境下使用内置的 Harbor 和 GitLab，仓库中的 Jenkinsfile 来创建 CI/CD 流水线，包括拉取代码、单元测试、代码质量检测、构建镜像、推送和发布版本，最终示例网站部署到 KubeSphere 集群中的开发环境和产品环境，并且能够通过公网访问。

- [Jenkinsfile out of SCM](#)

本示例以文档和视频演示如何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），最终将示例 Web 部署到 KubeSphere 集群中的开发环境且能够通过公网访问。

# 多租户管理快速入门

## 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，帮助用户熟悉多租户下的用户和角色管理，快速上手 KubeSphere。

## 前提条件

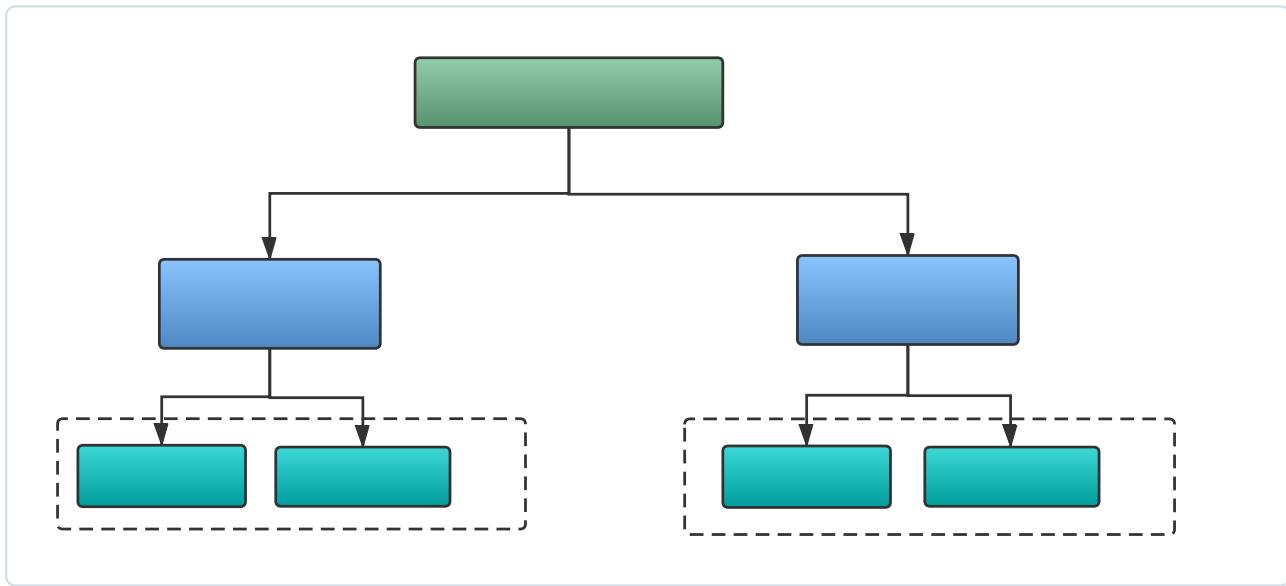
已安装 KubeSphere，并使用默认的 admin 用户名和密码登录了 KubeSphere。

## 预估时间

约 15 分钟。

## 操作示例

目前，平台的资源一共有三个层级，包括集群（Cluster）、企业空间（Workspace）、项目（Project）和 DevOps Project（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。

内置角色	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。

本示例首先新建一个角色 (users–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 users–manager 角色。

账号名	集群角色	职责
user–manager	users–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 `users-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 `users-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](#)。

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

1.6. 然后用 `user-manager` 来创建下表中的四个账号，`ws-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	职责
ws-manager	workspaces-manager	创建和管理企业空间
ws-admin	cluster-regular	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	创建和管理项目、DevOps 工程，邀请新成员加入
project-regular	cluster-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 'Accounts Management' section. The table lists the following accounts:

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。

点击左上角的 **平台管理** → **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**；

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。

列表 (1)

名称	项目数量	DevOps 工程数量	管理员	创建时间
system-workspace system workspace	7	0	admin	2018-12-17 09:21:55

成员:

创建

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

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 \*

demo-workspace

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

ws-admin

描述信息

企业空间示例

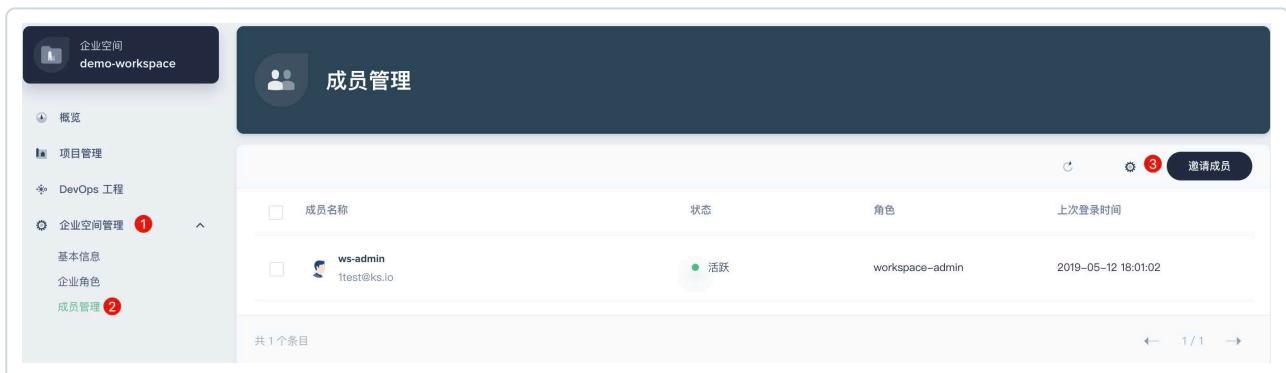
描述信息不超过 1000 个字符

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，可看到该企业空间下的项目和 DevOps 工程列表，点击左侧「进入企业空间」进入企业空间详情页。

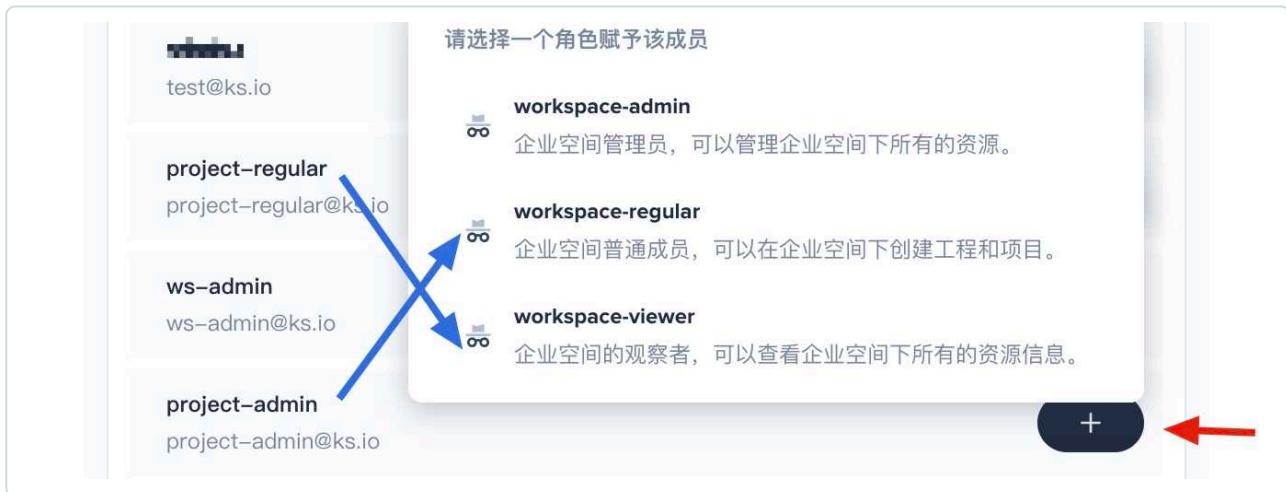


ws-admin 可以从集群成员中邀请新成员加入当前企业空间，然后创建项目和 DevOps 工程。在左侧菜单栏选择 企业空间管理 进入企业空间概览页，然后选择 企业空间管理 → 成员管理，点击 邀请成员。



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，且分别授予 workspace-regular 和 workspace-viewer 的角色，此时该企业空间一共有如下三个用户：

账号名	企业空间角色	职责
ws-admin	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	workspace-regular	创建和管理项目、DevOps 工程，邀请新成员加入
project-regular	workspace-viewer	将被 project-admin 邀请加入项目和 DevOps 工程， 用于创建工作负载、流水线等业务资源



## 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. 上一步将用户项目管理员 **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

**基本信息**

项目基础信息设置

名称 \*

demo-namespace

别名

示例项目

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

This is a demo

## 高级设置

3.3. 此处将默认的最大 CPU 和内存分别设置 **2 Core** 和 **2 Gi**，后续的示例都需要在这个示例项目中完成，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。

完成高级设置后，点击 **创建**。

**说明：**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。

**高级设置**

设置项目资源默认请求

① 默认最大 CPU 为 2 Core  
② 默认最大内存为 2 Gi

默认最大使用资源

CPU	2	Core	1	内存	2	Gi	2
-----	---	------	---	----	---	----	---

默认最小使用资源

CPU	10	m	10	内存	10	Mi	10
-----	----	---	----	----	----	----	----

值为空表示无限制, CPU 1核 = 1000m

**提示：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。**

## 邀请成员

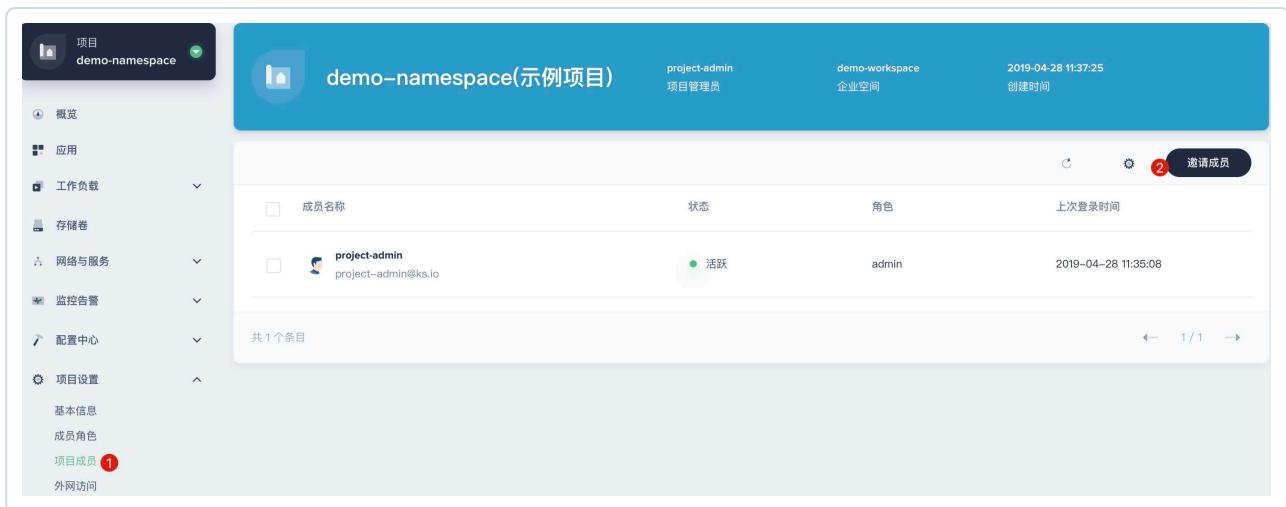
3.4. 示例项目 demo-namespace 创建成功后，点击进入示例项目。在 **步骤 2.4. 已邀请用户 project-regular** 加入了当前企业空间 **demo-workspace**，下一步则需要邀请 project-regular 进入该企业空间下的项目 demo-namespace。点击项目列表中的 demo-namespace 进入该项目。



The screenshot shows the KubeSphere dashboard with the following details:

- Top navigation bar: 工作台, 应用模板, KUBE SPHERE, project-admin.
- User profile: 你好 project-admin, 普通用户, 最近登录: 2019-05-13 09:57:27.
- Project list section:
  - Search bar: 请输入名称进行查找.
  - Filter buttons: 项目 (selected), DevOps 工程.
  - Table header: 列表 ①.
  - Data row for 'demo-namespace':
    - Icon: 基础设施
    - Name: demo-namespace
    - Owner: project-admin (项目管理员)
    - Created: 2019-05-12 20:41:59 (创建时间)
    - Resource usage: 容器组: 1, CPU: 4.00 m, 内存: 70.87 MiB

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 'demo-namespace' project settings page with the following details:

- Left sidebar: 项目 demo-namespace, 概览, 应用, 工作负载, 存储卷, 网络与服务, 监控告警, 配置中心, 项目设置 (highlighted), 基本信息, 成员角色, 项目成员 (highlighted with red 1), 外网访问.
- Main content area:
  - Project summary: demo-namespace(示例项目), owner: project-admin (项目管理员), workspace: demo-workspace (企业空间), created: 2019-04-28 11:37:25.
  - Members table:
    - Header: 成员名称, 状态, 角色, 上次登录时间.
    - Data row: project-admin (project-admin@ks.io), 活跃 (active), admin, 2019-04-28 11:35:08.
    - Total: 共 1 个条目.
  - Buttons: 重新加载, 编辑, 邀请成员 (highlighted with red 2).

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。



## 设置外网访问

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，负责接收项目外部进入的流量，并将请求转发到对应的后端服务。

3.7. 请使用项目管理员 **project-admin** 设置外网访问，选择「项目设置」 → 「外网访问」，点击「设置网关」。



3.8. 在弹窗中，选择默认的 NodePort，然后点击「保存」。



### 3.9. 当前可以看到网关地址、http/https 端口号都已经开启。

## 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。

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。

DevOps 工程  
demo-devops

demo-devops

project-admin 工程管理员

demo-workspace 企业空间

2018-12-19 23:36:15 创建时间

成员名称 状态 角色

<input type="checkbox"/> admin	已启用	owner
<input type="checkbox"/> project-admin	已启用	owner

流水线 工程管理 基本信息 凭证 成员角色 工程成员 邀请成员

### 邀请成员到该工程

您可以邀请新的成员来协助您的工程

输入邮箱邀请项目成员

请选择一个角色赋予该成员

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

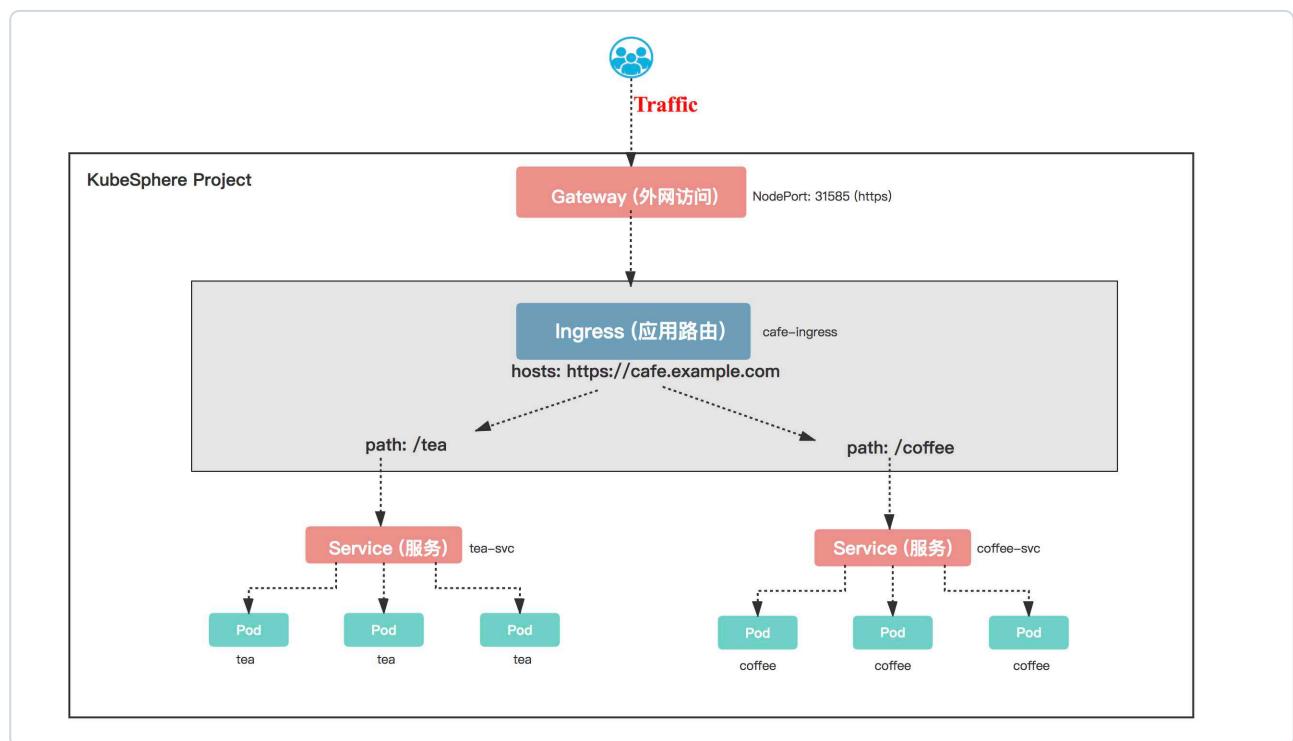
至此，本文档为您演示了如何在多租户的基础上，使用账户管理和角色管理的功能，以示例的方式介绍了常用内置角色的用法，以及创建项目和 DevOps 工程。请继续参考 [快速入门](#) 系列文档的其他示例比如 [应用路由与服务示例](#)，使用项目普通用户 [project-regular](#) 登录 KubeSphere 动手实践操作一遍，创建具体的工作负载、服务和 CI/CD 流水线。

# 应用路由与服务示例

KubeSphere 在项目中为用户项目内置了一个全局的负载均衡器，即应用路由控制器（Ingress Controller），为了代理不同后端服务（Service）而设置的负载均衡服务，用户访问 URL 时，应用路由控制器可以把请求转发给不同的后端服务。因此，应用路由的功能其实可以理解为 Service 的“Service”。

[Kubernetes Ingress](#) 官方提供了这样一个例子：对于 <https://cafe.example.com>，如果访问 <https://cafe.example.com/coffee> 则返回“咖啡点餐系统”，如果访问 <https://cafe.example.com/tea>，则返回“茶水点餐系统”。这两个系统分别由后端的 coffee 和 tea 这两个部署（Deployment）来提供服务。

以下将在 KubeSphere 创建相关资源来说明这个应用路由的示例。



## 前提条件

- 已创建了企业空间、项目和普通用户 `project-regular` 账号，并开启了外网访问，请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 - 邀请成员](#)。

# 预估时间

约 20 分钟。

## 操作示例

### 创建部署

#### 创建 tea

1. 以用户 project-regular 登录，进入 demo-namespace 项目后，选择「工作负载」→「部署」，点击「创建部署」。



2. 参考如下提示填写基本信息，完成后点击「下一步」。

- 名称：必填，填写 `tea`，一个简洁明了的名称可便于用户浏览和搜索；
- 别名：可选，支持中文帮助更好的区分资源，例如填写 `茶水点餐系统`；
- 描述信息：简单介绍该工作负载，方便用户进一步了解。

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 *	tea	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	茶水点餐系统	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。			

3. 副本数增加至 2，点击「添加容器」，在容器组模板的表单中，参考如下提示填写镜像和服务设置，其它信息默认即可：

- 镜像：nginxdemos/hello:plain-text；
- 容器名称：tea
- 服务设置：名称输入 port，协议默认 TCP，端口号输入 80（容器端口）；

完成后点击「保存」，选择「下一步」。

镜像 \*

nginxdemos/hello:plain-text ①

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。

通过代码构建新的容器镜像

从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

容器规格设置

对容器的名称及容器的计算资源进行设置

容器名称 \*

tea 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

CPU

最小使用 10 m 最大使用 2 Core

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 2 Gi

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。

服务设置

设置容器的访问策略

port ③ TCP ④ 80

① 输入 nginxdemos/hello:plain-text  
② 输入 tea  
③ 输入 port  
④ 填写 80

4. 无需设置存储卷，点击「下一步」。标签设置保留默认即可，点击「创建」，即可看到 tea 已创建成功。

## 创建 coffee

1. 同上，点击「创建」，参考以上步骤创建一个名为 coffee 的咖啡点餐系统的部署 (deployment)。
2. 副本数增加至 2，点击「添加容器」，在容器组模板的表单中，参考如下提示填写 coffee 的镜像和服务设置，其它信息默认即可：
  - 镜像：nginxdemos/hello:plain-text；
  - 容器名称：coffee
  - 服务设置：名称输入 port，协议默认 TCP，端口号输入 80 (容器端口)；
3. 完成后点击「保存」，选择「下一步」。无需设置存储卷，点击「下一步」，然后点击「创建」，即可看到 coffee 已创建成功，至此一共创建了如下两个部署。

The screenshot shows the KubeSphere interface under the 'demo-namespace' project. On the left, there's a sidebar with '概览', '应用', '工作负载' (selected), '存储卷', and '网络与服务'. The main area is titled '部署' (Deployments). It shows 0 used quota, infinity planned quota, and infinity remaining pod quota. There's a search bar and filter options for name, status, application, and update time. Two deployments are listed: 'coffee(咖啡点餐系统)' is running with 2/2 replicas, updated on 2019-04-24 10:07:40; 'tea(茶水点餐系统)' is also running with 2/2 replicas, updated on 2019-04-24 09:59:24.

## 创建服务

为 tea 和 coffee 分别创建两个服务，选择「网络与服务」→「服务」，点击「创建服务」。

The screenshot shows the KubeSphere interface under the 'demo-namespace' project. On the left, there's a sidebar with '概览', '应用', '工作负载', '存储卷', '网络与服务' (selected), '灰度发布', and '应用路由'. The main area is titled '服务' (Services). It shows 0 used quota, 0 virtual IP, and 0 Headless. There's a brief description of what a service is. A large '创建服务' (Create Service) button is prominent. A callout box on the right indicates step 1: '选择『服务』' (Select 'Service') and step 2: '点击『创建服务』' (Click 'Create Service').

## 创建 tea-svc

1. 名称填写 `tea-svc`，点击「下一步」。
2. 选择第一项 `通过集群内 IP 来访问服务 Virtual IP`，参考如下提示填写服务设置：
  - 点击「指定工作负载」，选择 `tea`，点击保存；
  - 端口：名称为 `port`，默认 `TCP` 协议，端口和目标端口都填写 `80`（前者表示暴露在 Cluster IP 上的端口，仅提供给集群内部访问服务的入口，目标端口是当前的服务后端 Pod 上的端口）。

The screenshot shows the 'Service Settings' tab of the service creation wizard. At the top, there are tabs for '基本信息' (Basic Information), '服务设置' (Service Settings) (which is selected), '标签设置' (Label Settings), and '外网访问' (External Network Access). Below the tabs, the title is '虚拟 IP'. The configuration section includes:

- 选择器 \***: A dropdown menu showing 'app' and 'tea'. A warning message above it says: '当前设置的选择器(app=tea)共影响到 1 个工作负载 查看' (The current selector (app=tea) affects 1 workload). Buttons for '添加选择器' (Add Selector) and '指定工作负载' (Specify Workload) are available.
- 端口 \***: A configuration group for port settings. It includes a dropdown for 'port' (set to 'port'), a dropdown for '协议' (Protocol) set to 'TCP', an input field for '端口号' (Port Number) set to '80', and a button for '添加端口' (Add Port).
- 会话亲和性**: A dropdown menu currently set to 'None'.

3. 点击「下一步」，标签设置保留默认，点击「创建」，`tea-svc` 即可创建成功。

## 创建 coffee-svc

同上，点击「创建」，参考以上步骤创建一个名为 `coffee-svc` 的服务，工作负载指定为 `coffee`，其它信息与 `tea-svc` 服务相同，完成后点击「创建」。至此已成功创建了两个服务。

项目  
demo-namespace

概览 应用 工作负载 存储卷 网络与服务 监控告警

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

输入查询条件进行过滤

名称	IP地址	端口	应用	创建时间
coffee-svc	Virtual IP: 10.233.37.95	端口: 80:80/TCP 节点端口: -	-	2019-04-24 10:33:38
tea-svc	Virtual IP: 10.233.25.188	端口: 80:80/TCP 节点端口: -	-	2019-04-24 10:28:35

0 已用配额 0 虚拟 IP 0 Headless

创建

## 创建 TLS 证书密钥

由于在应用路由中绑定的域名为 https 协议，因此需要预先在密钥中创建 TLS 证书。

1. 点击「配置中心」→「密钥」，点击「创建」。
2. 密钥名称填写 `cafe-secret`，点击「下一步」。
3. 类型选择 `TLS`，凭证和私钥填写如下，完成后点击「创建」。

#凭证

-----BEGIN CERTIFICATE-----

MIIDLjCCAhYCCQDAOF9tLsaXWjANBgkqhkiG9w0BAQsFADBaMQswCQYDVQQGEwJV  
UzELMAkGA1UECAwCQ0ExTAfBgNVBAoMGEIudGVybmv0IFdpZGdpdHMgUHR5IE0  
ZDEbMBkGA1UEAwSY2FmZS5leGFtcGxLmNvbSAgMB4XDTE4MDkxMjE2MTUzNV0X  
DTIzMDkxMTE2MTUzNVowWDELMAkGA1UEBhMCVVMxszAJBgNVBAgMAkNBMSewHwYD  
VQQKDBhJbnRlcm5ldCBXaWRnaXRzIB0eSBMdGQxGTAXBgNVBAMMEGNhZmUuZXhh  
bXBsZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCP6Kn7sy81  
p0juJ/cyk+vCAmlsfjtFM2muZNK0KtecqG2fjWQb55xQ1YFA2XOSwHAYvSdwI2jZ  
ruW8qXXCL2rb4CZCFxwpVECrcxdjm3teViRXVsYImmJHPPSyQgpiobs9x7DILc6I  
BA0ZjUOyl0PqG9SJexMV73Wll5rDVSF2r4kSkbAj4Dcj7LXeFIvxH2I5XwXCptC  
n67JCg42f+k8wgzcRVp8XZkZWZVjwq9RUKDXmFB2YyN1XEWdZ0ewRuKYUJlsm692  
skOrKQj0vkoPn41EE/+TaVEpqLTRoUY3rzg7DkdzfdBizFO2dsPNFx2CW0jXkNLv  
Ko25CZrOhXAHAgnBAAEwDQYJKoZIhvcNAQELBQADggEBAKHFCcyOjZvoHswUBMdL  
RdHlb383pWFynZq/LuUovsVA58B0Cg7BEfy5vWVVrq5Rlkv4IZ81N29x21d1JH6r  
jSnQx+DXCO/TJEV5ISCUPIGzEUYaUPgRyjsM/NUdCJ8uHVhZJ+S6FA+CnOD9rn2i  
ZBePCI5rHwEXwnnl8ywij3vvQ5zHluyBglWr/Qyui9fjPpwWUvUm4nv5SMG9zCV7  
PpuvuuatqjO1208BjfE/cZHIg8Hw9mvW9x9C+IQMIMDE7b/g6OcK7LGTlwIFxvA8  
7WjEequanylphMhKRXVf1N349eN98Ez38fOTHTPbdJjFA/PcC+Gyme+iGt5OQdFh  
yRE=

-----END CERTIFICATE-----

#私钥

-----BEGIN RSA PRIVATE KEY-----

MIIEowlBAAKCAQEaQeip+7MvNadl7if3MpPrwgJpbH47RTNprmTStCrXnKhtn41k  
G+ecUNWBQNIzksBwGL0ncCNo2a7lvKI1wi9q2+AmQhccKVRAq3MXY5t7XIYkV1bG  
CJpiRzz0sklKYqG7Pcew5S3OiaQNGY1DspdD6hvUiXsTFe91iCGuaw1Uhdq+JEpG  
wl+A3I+y13hZVVx9iOV8FwqbQp+uyQoONn/pPMIM3EVafF2ZGVmVY8KvUVcg15hQ  
dmMjdVxFnWdHsEbimFCZbJuvdrJDqykI9L5KD5+NRBP/k2IRKai00aFGN684Ow5H  
c33QYsxTtnbDzRcdglti15DS7yqNuQmazoVwBwlDAQABAoIBAQCPsdSYnQtSPyql  
FfVFpTOsoOYRhfs8sl+ibFxIOuRauWehhJxdm5RORpAzmCLyL5VhjtJme223gLrw2  
N99EjUKb/VOmZuDsBc6oCF6QNR58dz8cnORTewcotsJR1pn1hhlnR5HqJJBJask1  
ZEnUQfcXZrL94lo9JH3E+Uqjo1FFs8xxE8woPBqjZsV7pRUZgC3LhxnwLSExyFo4  
cbx9SOG5OmAJozStFoQ2GJOes8rJ5qfdvytg9xbLaQL/x0kpQ62BoFMBDdqOePW  
KfP5zZ6/07/vpj48yA1Q32PzobubsBLd3Kcn32jfm1E7prtWI+JeOFiOznBQFJbN  
4qPVRz5hAoGBANTWyxhNCSLu4P+XgKyckljJ6F5668fNj5CzgFRqJ09zn0TlsNro  
FTLZcxDqnR3HPYM42JERh2J/qDFZynRQo3cg3oeivUdBVGY8+FI1W0qdub/L9+yu  
edOZTQ5XmGGp6r6jexymcJim/OsB3ZnYOpOrID7SPmBvzNLk4MF6gbxAoGBAMZO  
0p6HbBmcP0tjFXfcKE77lmLm0sAG4uHoUx0ePj/2qrnTnOBBN4MvgDuTJzy+caU  
k8RqmdHCbHzTe6fzYq/9it8sZ77KVN1qkbIcuc+RTxA9nNh1TjsRne74Z0j1FCLk

名称	类型	配置数量	创建时间
cafe-secret	TLS	2	2019-05-22 11:45:12
default-token-5w4xz	kubernetes.io/service-account-token	3	2019-05-19 01:35:04
istio.default	istio.io/key-and-cert	3	2019-05-19 01:35:04

## 创建应用路由 cafe-ingress

1. 选择「网络与服务」→「应用路由」，点击「创建应用路由」。
2. 输入名称 `cafe-ingress`，点击「下一步」，点击「添加路由规则」。
3. 选择「指定域名」，按照如下提示填写路由规则，应用路由的路由规则即它的核心所在。

- 域名：cafe.example.com
- 协议：选择 https
- 密钥：选择 `cafe-secret`
- 路径：
  - 输入 `/coffee`，服务选择 coffee-svc，选择 80 端口作为服务端口，点击「添加 Path」
  - 输入 `/tea`，服务选择 tea-svc，选择 80 端口作为服务端口

模式

自动生成 指定域名

**1** 请确保您设置的域名可以解析到访问入口的 IP 地址;  
如果在私有云环境中, 请修改本地的host文件, 并通过 域名+节点端口 的方式来访问

域名 \*

cafe.example.com

协议

https

secretName

cafe-secret

路径 \*

/coffee	coffee-svc	80	删除
/tea	tea-svc	80	删除

添加 Path

- 完成路由规则设置后点击「保存」，无需设置注解，选择「下一步」，点击「创建」，cafe-ingress 创建成功。

项目 demo-namespace

概览 应用 工作负载 存储卷 网络与服务 服务 灰度发布

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

输入查询条件进行过滤

名称	网关地址	应用	创建时间
cafe-ingress	192.168.0.25	-	2019-04-24 10:58:45

创建

## 访问应用路由

在云平台需要把应用路由在外网访问的 https 端口（比如本示例是 31198），在端口转发规则中将内网端口 31198 转发到源端口 31198，然后在防火墙开放这个源端口，确保外网流量可以通过该端口，即可通过 curl 命令进行访问测试。例如在 QingCloud 平台配置端口转发和防火墙规则，可参考 [云平台配置端口转发和防火墙](#)。

至此，即可通过外网分别访问“咖啡点餐系统”和“茶水点餐系统”，即访问应用路由下不同的服务。

The screenshot shows the KubeSphere UI for the 'demo-namespace' project. The top navigation bar includes '项目 demo-namespace', 'admin' (项目管理员), 'demo-workspace' (企业空间), and '2019-05-07 20:34:55' (创建时间). On the left, a sidebar lists project components: 概览 (Overview), 应用 (Application), 工作负载 (Workload), 存储卷 (Storage), 网络与服务 (Network & Services), 监控告警 (Monitoring & Alarming), 配置中心 (Configuration Center), and 项目设置 (Project Settings). Under '项目设置', there are links for 基本信息 (Basic Information), 成员角色 (Member Roles), 项目成员 (Project Members), and 外网访问 (External Access). The main content area displays '外网访问' (External Access) with four options: NodePort (访问方式), 192.168.0.20 (网关地址), http:32067 (节点端口), and https:31198 (节点端口). The 'https:31198' link is highlighted with a red circle.

**提示：如果在内网环境可登录集群中的任意节点或通过 web kubectl，通过以下 curl 命令进行访问测试，需要将公网 IP 替换为网关地址。**

比如，我们访问 `https://cafe.example.com:{$HTTPS_PORT}/coffee` 时，应该是 coffee 的部署负责响应请求，访问这个 URL 得到的返回信息是：Server name: coffee-6cbd8b965c-9659v，即 coffee 这个 Deployment 的名字。

```
# curl --resolve {$HOSTNAME}:{$HTTPS_PORT}:{$IP} https://{$hostname}:{$HTTPS_PORT}/{$PATH} --in
$ curl --resolve cafe.example.com:31198:139.198.100.100 https://cafe.example.com:31198/c
Server address: 10.233.122.100:80
Server name: coffee-6cbd8b965c-9659v
Date: 23/Apr/2019:03:17:43 +0000
URI: /coffee
Request ID: 1dcb8794548dd6013439b85bbaef0dd6
```

而访问 `https://cafe.example.com:{$HTTPS_PORT}/tea` 的时候，则应该是 tea 的部署负责响应我的请求（Server name: tea-588dbb89d5-bgxqn），说明应用路由已经成功将不同的请求转发给了对应的后端服务。

```
# curl --resolve {$HOSTNAME}:{$HTTPS_PORT}:{$IP} https://{$hostname}:{$HTTPS_PORT}/{$PATH} --ins
curl --resolve cafe.example.com:31198:139.198.100.100 https://cafe.example.com:31198/tea
Server address: 10.233.122.97:80
Server name: tea-588dbb89d5-bgxqn
Date: 23/Apr/2019:03:16:23 +0000
URI: /tea
Request ID: 98db6b03aab0b6ab29c26ba37ab2ba2
```

# 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、项目和普通用户 `project-regular` 账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

## 预估时间

约 10 分钟。

## 操作示例

### 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. 以项目普通用户 `project-regular` 登录 KubeSphere，在当前项目下左侧菜单栏的 [配置中心](#) 选择 [密](#)

钥，点击 **创建**。

项目 demo-namespace

概览 应用 工作负载 存储卷 网络与服务 监控告警 配置中心 密钥 1 配置 项目设置

密钥

输入查询条件进行过滤

名称	类型	配置数量	创建时间
istio.default	istio.io/key-and-cert	3	2019-04-28 11:37:26
default-token-v5jpd	kubernetes.io/service-account-token	3	2019-04-28 11:37:25

共 2 个条目

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**

创建密钥

编辑模式

基本信息 密钥设置

基本信息

名称 \*

mysql-secret

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

MySQL 密钥

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

MySQL 初始密码

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**。



项目 demo-namespace...

概览 应用 工作负载 部署 有状态副本集 守护进程集 任务 定时任务 存储卷

有状态副本集

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

官网文档 参考文档

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

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 *	wordpress-mysql	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	MySQL 数据库	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 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。

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

从公开或者私有镜像仓库中拉取镜像

镜像 \*

mysql:5.6

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。

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

**容器规格设置**  
对容器的名称及容器的计算资源进行设置

容器名称 \*

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

CPU

最小使用	10	m	最大使用	2	Core
------	----	---	------	---	------

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

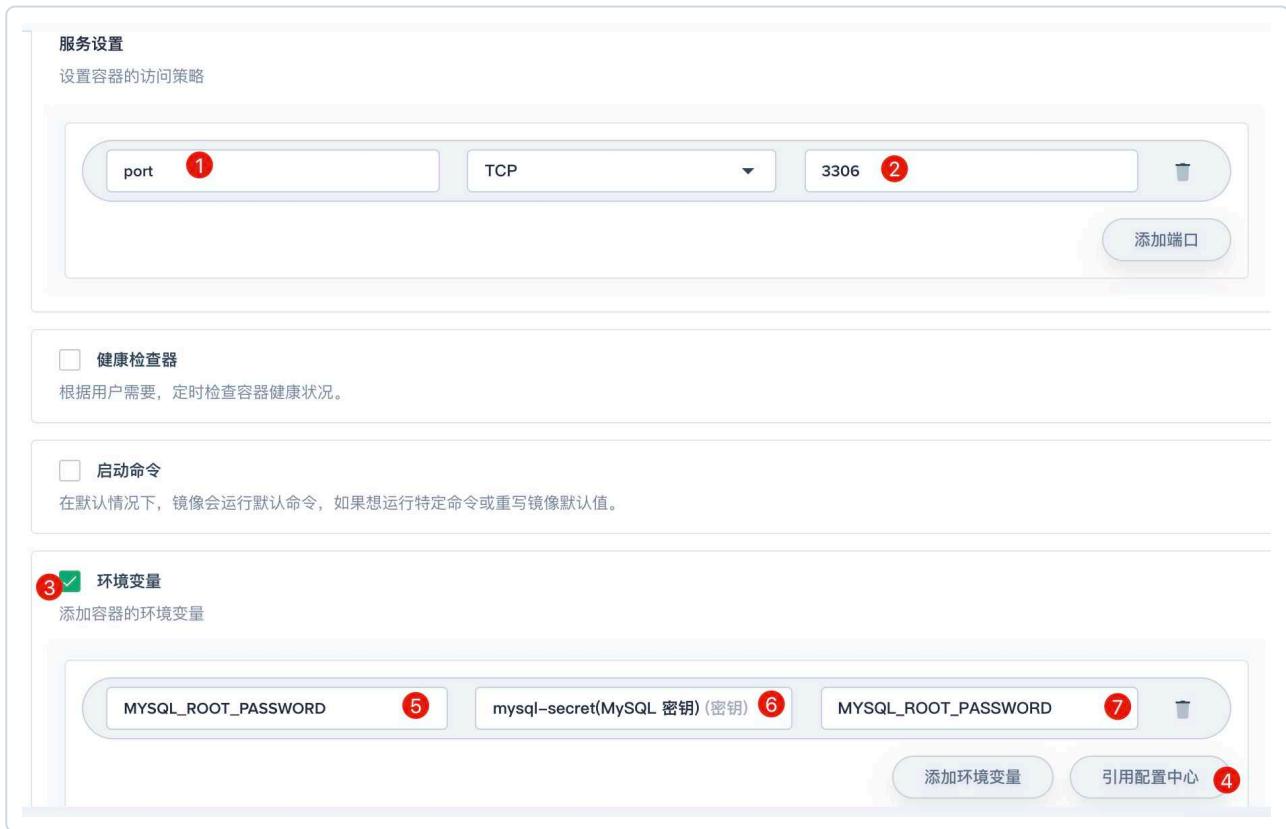
内存

最小使用	10	Mi	最大使用	2	Gi
------	----	----	------	---	----

4.2. 对 **服务设置** 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：名称可自定义如 port，选择 `TCP` 协议，填写 MySQL 在容器内的端口 `3306`。

- 环境变量：勾选环境变量，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. 点击 **保存**，然后点击 **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。

- 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`
- 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`，访问模式默认 `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`

完成后点击 **保存**，然后点击 **下一步**。

基本信息 ··· 命名空间模板 ··· 存储卷模板 ··· 服务配置 ··· 标签设置 ··· 节点选择器

◀ 添加存储卷模板

存储卷名称 \*

mysql-pvc

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

存储类型

local

容量

0 512Gi 1024Gi 1.5Ti 2Ti 10 Gi

访问模式

ReadWriteOnce(RWO)  
单个节点读写

挂载路径

doctor mysql 读写 /var/lib/mysql

## 第六步：服务配置

要将 MySQL 应用暴露给其他应用或服务访问，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。



## 第七步：标签设置

标签保留默认设置 `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**。

## 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 shows the KubeSphere UI for managing a MySQL service. 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 is divided into several sections:

- 服务 (Service):** Shows a summary for the mysql-service Virtual IP, with a port of 3306/TCP.
- 副本运行状态 (Replica Status):** Displays a circular icon with "1/1" indicating one replica is running. It also shows the expected replicas (1) and actual replicas (1).
- 端口 (Port):** Lists a single port entry: port (TCP) 3306.
- 容器组 (Container Group):** Shows a list item for "wordpress-mysql-0" with a status of "ContainerCreating". A red circle highlights this status. Below it, it lists the container group IP as - and the host as i-rgem3qkr(192.168.0.74). A note states "暂时没有监控数据" (No monitoring data available).

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](#)。

# 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 [示例二](#) 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，若还未创建请参考上一篇 [部署 MySQL](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

## 预估时间

约 15 分钟。

## 操作示例

### 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，为演示方便，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. 以项目普通用户 `project-regular` 登录 KubeSphere，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. 填写配置的基本信息，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`

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

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

## 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



创建存储卷

基本信息

存储卷可将数据持久化，生命周期独立于应用负载。创建存储卷前请确保已创建存储类型。

名称 *	wordpress-pvc	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 持久化存储卷	描述信息	Wordpress PVC
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。			

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. 标签默认为 `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**。

状态	已用配额	规划配额	剩余 Pod 配额
0	∞	∞	∞

### 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 *	项目
wordpress	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

别名	描述信息
Wordpress 网站	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。

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

选择已有镜像部署容器  
从公开或者私有镜像仓库中拉取镜像

镜像 *
wordpress:4.8-apache

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。

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

**容器规格设置**  
对容器的名称及容器的计算资源进行设置

容器名称 *
wordpress

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

CPU	
最小使用 10 m	最大使用 2 Core

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

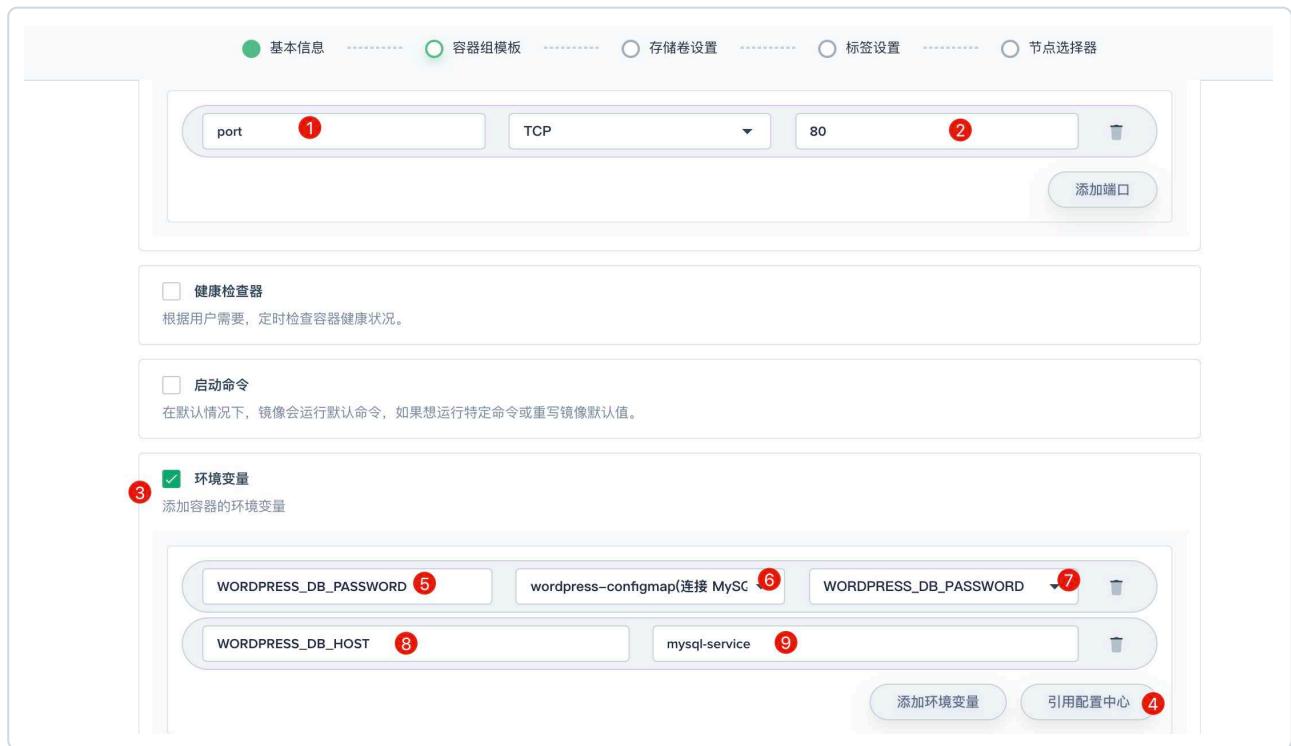
内存	
最小使用 10 Mi	最大使用 2 Gi

5.2. 下滑至服务设置，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。

- 端口：名称可自定义如 port，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。



5.3. 完成后点击 **保存**，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`。



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**。



## 第七步：查看部署

7.1. 标签保留默认值，节点选择器此处暂不作设置，点击 **创建**，部署创建完成。

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 KubeSphere UI for managing a Wordpress deployment. On the left, there's a sidebar with project navigation and a deployment summary for 'Wordpress 网站'. The main 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, displaying a summary card with '0/1 副本运行状态' (0/1副本运行状态) and a note about Deployments. Below this are sections for '端口' (Ports) and '容器组' (Container Groups). A specific container named 'wordpress-95cd769b9-75pdq' is highlighted with a red circle around its status 'ContainerCreating'.

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，最终暴露给外网访问。

This screenshot shows the deployment details for the Wordpress application. The left sidebar shows the '部署' (Deployment) section is selected. The main panel displays deployment statistics: 1 pod created, infinity pods planned, and infinity pods remaining. It also shows the deployment name 'wordpress' and its status as '运行中' (Running). A '创建' (Create) button is visible at the top right.

## 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。

This screenshot shows the '服务' (Service) creation interface. The left sidebar has a '服务' (Service) section with a red '1' badge. The main panel has a '创建' (Create) button with a red '2' badge. It displays service statistics: 1 pod used, 0 virtual IP, and 0 Headless services. A table lists an existing service named 'mysql-service' with a Virtual IP of '10.233.10.133' and port '3306:3306/TCP'. The bottom of the screen shows a footer with '共 1 个条目' (1 item) and page navigation.

## 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

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 \*

wordpress-service

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

Wordpress 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。

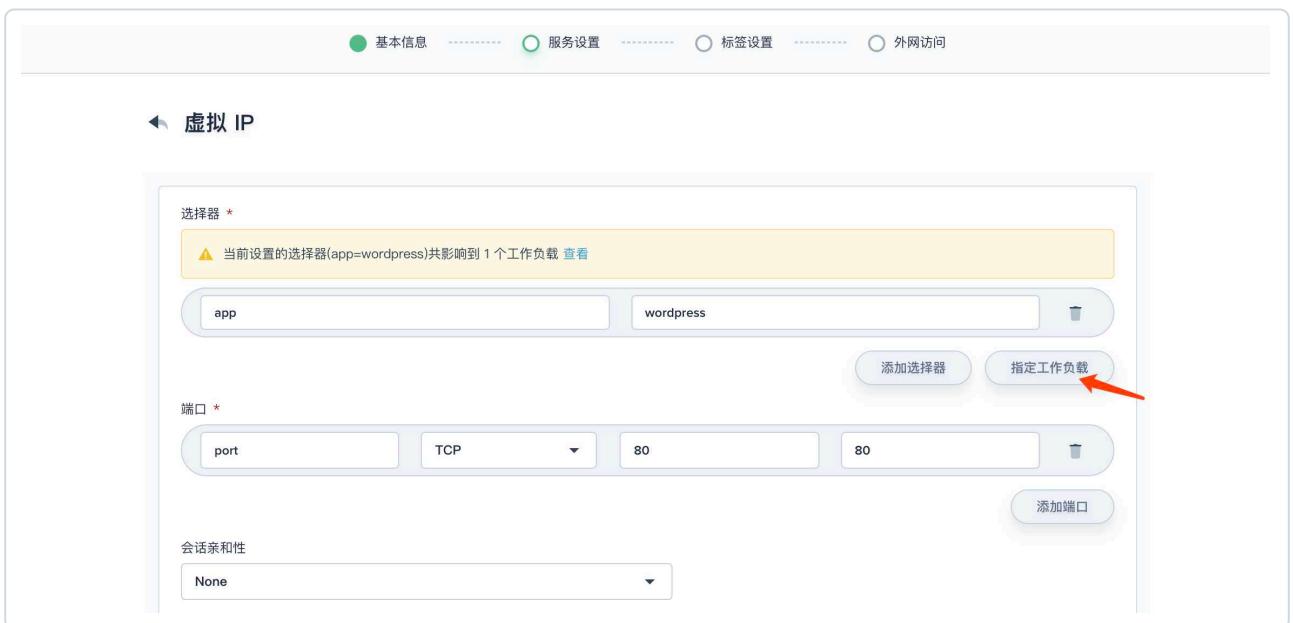
描述信息

Wordpress 服务

## 8.3. 服务设置参考如下填写，完成后点击 **下一步**：

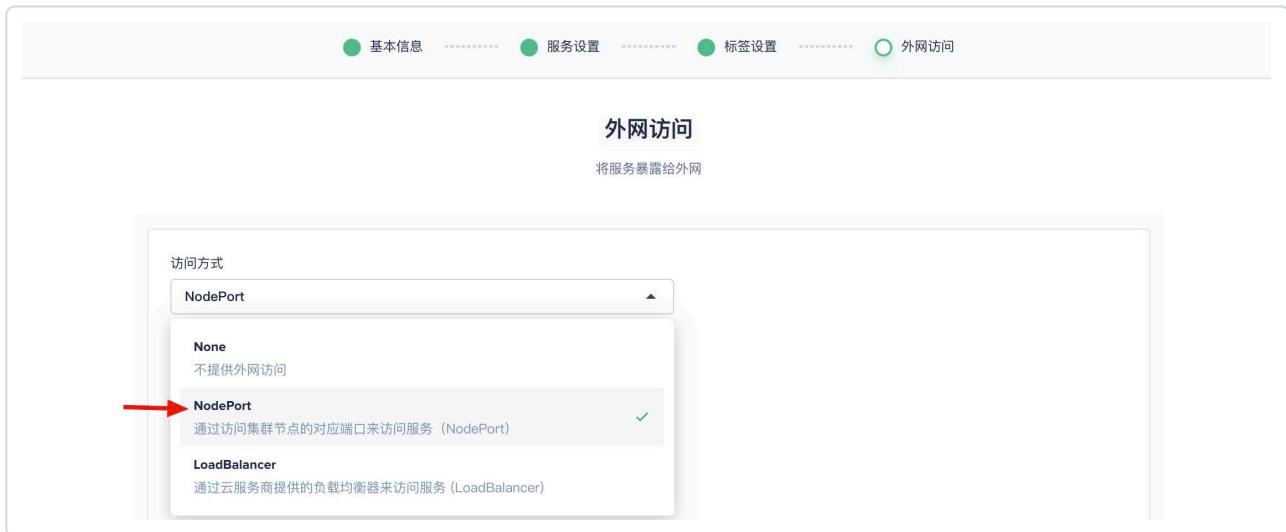
- 服务类型：选择第一项 **通过集群内部IP来访问服务 Virtual IP**
- 选择器：点击 **指定工作负载** 可以指定上一步创建的部署 Wordpress，指定后点击 **保存**
- 端口：端口名称可自定义如 port，服务的端口和目标端口都填写 `TCP` 协议的 `80` 端口
- 会话亲和性：None，完成参数设置，选择下一步

**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。



8.4. 本示例标签保留默认值，选择 **下一步**。

8.5. 服务暴露给外网访问支持 NodePort 和 LoadBalancer，这里服务的访问方式选择 **NodePort**。



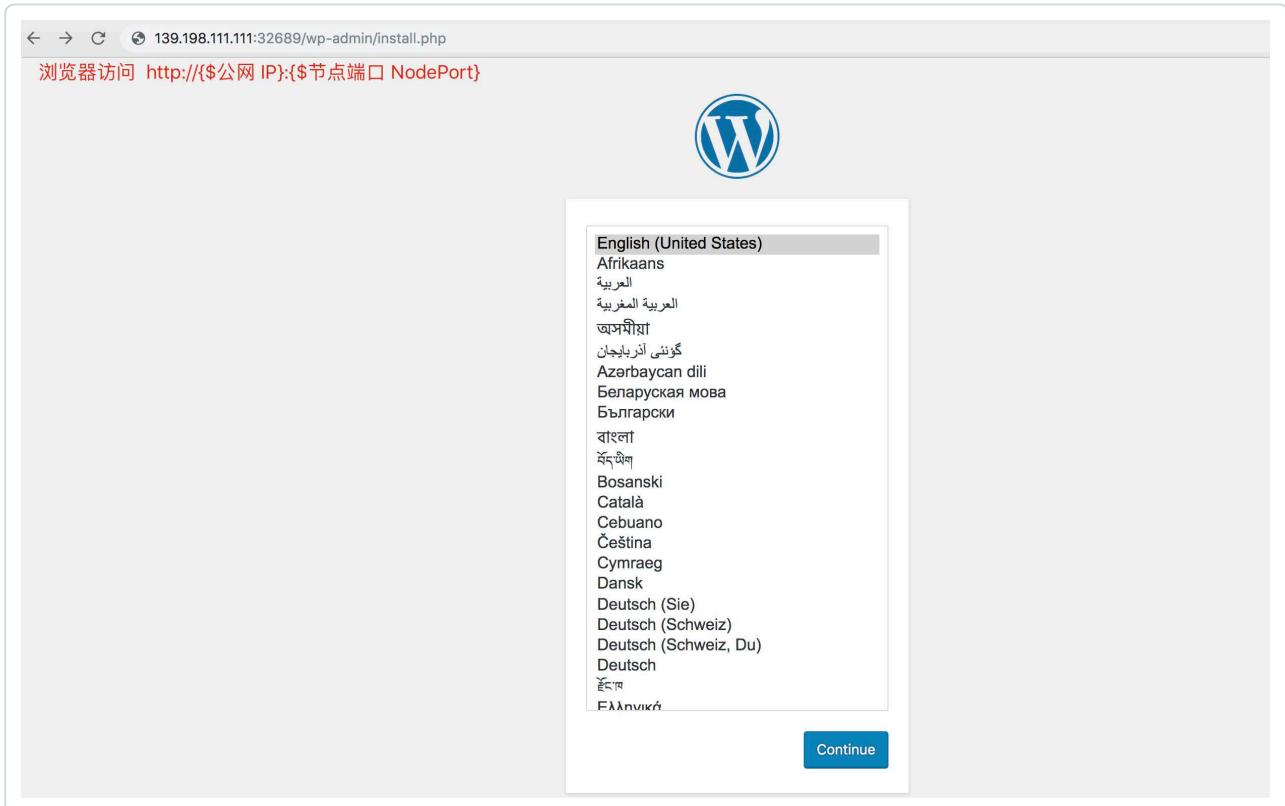
8.6. 点击 **创建**, wordpress-service 服务可创建成功。注意, wordpress-service 服务生成了一个节点端口 **32689**。

名称	IP地址	端口	应用	创建时间
wordpress-service(Wordpress Service)	Virtual IP: 10.233.31.86	端口: 80:80/TCP 节点端口: 32689/TCP	-	2019-05-08 15:33:02

**注意：若需要在外网访问，可能需要绑定公网 EIP 并配置端口转发和防火墙规则。在端口转发规则中将内网端口 32689 转发到源端口 32689，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。**

## 访问 Wordpress

设置完成后, WordPress 就以服务的方式通过 NodePort 暴露到集群外部, 可以通过 [http://{\\$公网IP}:{\\$节点端口 NodePort}](http://{$公网IP}:{$节点端口 NodePort}) 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）、密钥（Secret）、配置（ConfigMap）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](#)。

# 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，以及按照时间执行任务。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、项目和普通用户 `project-regular` 账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

## 预估时间

约 15 分钟。

# 操作示例

## 创建任务

以项目普通用户 `project-regular` 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。

The screenshot shows the KubeSphere control panel interface. At the top, there are navigation links for '工作台' (Dashboard) and '应用模板' (Application Templates). The central header is 'KUBESPHERE'. On the right, it shows the user 'project-regular' and a dropdown menu. The left sidebar is for the project 'demo-namespace', with sections for '概览' (Overview), '应用' (Applications), '工作负载' (Workloads), '部署' (Deployments), '有状态副本集' (StatefulSets), '守护进程集' (DaemonSets), '任务' (Jobs) - which is highlighted with a red circle containing the number 1, '定时任务' (Scheduled Jobs), and '存储卷' (Storage Volumes). The main content area is titled '任务' (Jobs) and contains a brief description: '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。' Below this, there are statistics: '0 已用配额' (0 used quota), '∞ 规划配额' (∞ planned quota), and '∞ 剩余 Pod 配额' (∞ remaining Pod quota). There are also links for '官网文档' (Official Documentation) and '参考文档' (Reference Documentation). A search bar at the top right says '输入查询条件进行过滤' (Enter filtering conditions). The job list table has columns for '任务' (Job), 'S2I 任务' (S2I Job), '状态' (Status), and '更新时间' (Last Updated). The table shows '共 0 个条目' (0 items). At the bottom right of the table, there are navigation arrows for pages 1/1. A large blue '创建' (Create) button is located at the top right of the main content area.

## 第一步：填写基本信息

点击 **创建**，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。

创建任务

编辑模式

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

### 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  
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。

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：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: 输入 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；如此处设置为 5 则表示最多重试 5 次。
- Completions: 输入 4，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: 输入 2，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: 输入 300，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 300 则表示如果超过 300s 后 Job 中的所有 Pod 运行将被终止。

### 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，则任务不会继续尝试工作

Completions 4  
标志任务结束需要成功运行的容器组个数

Parallelism 2  
标志并行运行的容器组的个数

Active Deadline Seconds 300  
任务运行的超时时间

### 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。

The screenshot shows the 'Container Template' configuration page. At the top, there are tabs for '基本信息' (Basic Information), '任务设置' (Task Settings), '容器组模板' (Container Template) (which is selected and highlighted in green), '存储卷设置' (Storage Volume Settings), '标签设置' (Label Settings), and '节点选择器' (Node Selector). Below the tabs, the title '容器组模板' (Container Template) is displayed, followed by the subtitle '指定任务中运行的容器组模板' (Specify the container group template running in the task). Under the '重启策略' (Restart Policy) section, a dropdown menu is set to 'Never (容器组出现故障时创建新的容器组)' (Never (Create a new container group when the container group fails)). A note below the dropdown says '重启策略(Never/OnFailure)' (Restart Policy (Never/OnFailure)). Below the dropdown is a large input field with a dashed border, containing the placeholder text '请至少添加一个容器' (Please add at least one container). A button labeled '添加容器' (Add Container) is located at the bottom right of this field.

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。

The screenshot shows the 'Add Container' configuration dialog. At the top left, there is a back arrow icon and the text '添加容器'. The dialog is divided into several sections:

- 容器规格设置** (Container Specification Settings):
  - 容器名称 \***: Input field containing 'pi'.
  - 镜像 \***: Input field containing 'perl'.
  - A note below the fields says: '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 contain lowercase letters, digits, and hyphens, and must start and end with lowercase letters or digits).
  - CPU settings:
    - 最小使用: 10
    - m
    - 最大使用: 2
    - CoreA note below the CPU section 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' (Dependency for resource allocation during container scheduling. Only if the total allocatable CPU on the node is ≥ the container's minimum usage value, it can be scheduled to the node. Unit conversion rule: 1 core = 1000m).
  - 内存 settings:
    - 最小使用: 10
    - Mi
    - 最大使用: 2
    - GiA note below the memory section 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' (Dependency for resource allocation during container scheduling. Only if the total allocatable memory on the node is ≥ the container's minimum usage value, it can be scheduled to the node).

勾选 **启动命令**，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)
```

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

名称: TCP 端口:

健康检查器  
根据用户需要，定时检查容器健康状况。

**①**  启动命令  
在默认情况下，镜像会运行默认命令。如果想运行特定命令或重写镜像默认值。

运行命令:

- perl ②
- Mbignum=bpi ④
- wle ⑤
- print bpi(2000) ⑥

## 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。标签默认为 `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。

任务

1 已用配额

$\infty$  规划配额

$\infty$  剩余 Pod 配额

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

## 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。

任务示例

执行记录

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

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 'Resource Status' tab for a Job named 'job-demo'. On the left, there's a sidebar with '任务示例' (Task Examples) and a 'Parallelism' field set to 2, which is circled in red. The main area displays two container groups: 'job-demo-5slt4' and 'job-demo-v79rw', both created 1 minute ago. A third container group, 'pi', is listed below them.

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。

This screenshot shows the detailed view of the 'job-demo-5slt4' container group. It includes sections for '容器' (Containers) and '存储设备' (Storage Devices). The container 'pi' is shown with its resource limits: CPU 100m request 1 limit and memory 50Mi request 1000Mi limit. The storage device 'default-token-sskpr' is listed with its type as Secret and key as default-token-sskpr.

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。

This screenshot shows the 'Logs' tab for the 'pi' container. The terminal section displays the output of the command 'perl -Mbignum=bpi -wle print bpi(2000)', which calculates pi to 2000 decimal places.

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](#)。

## 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、项目和普通用户 `project-regular` 账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

### 预估时间

约 15 分钟。

# 操作示例

## 第一步：添加应用仓库

应用仓库的后端可以是 QingStor 对象存储或 AWS 对象存储，还可以是 GitHub，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，需提前上传应用配置包至对象存储或 GitHub，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，请参考 [添加示例应用仓库](#) 进行操作。

## 第二步：部署应用

1、添加示例应用仓库后，查看和部署应用等后续操作使用项目普通用户 [project-regular](#) 账号登录 KubeSphere 进行操作，在顶部的 **应用模板** 中查看新添加仓库的应用。

A screenshot of the KubeSphere web interface. At the top, there's a navigation bar with tabs for '工作台' (Workstation) and '应用模板' (Application Template). The main header says 'KUBESPHERE'. On the right, it shows the user 'project-regular'. Below the header, a blue banner reads '应用一键部署' (One-click deployment of applications) with a sub-instruction: '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' (Through visualization, KubeSphere displays and provides deployment and management functions. Users can quickly one-click deploy applications based on application templates). Underneath the banner, there's a dropdown menu set to 'docs-demo-repo' and a search bar with placeholder text '请输入名称进行查找'. A large card titled '应用模板' (Application Template) is displayed, featuring a thumbnail of an Nginx icon, the name 'nginx', the version 'Version: 0.1.0 [1.0]', and the note 'A Helm chart for Kubernetes'. The overall interface has a clean, modern design with a light gray background.

2、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。

基本信息

版本信息	0.1.0 [1.0]
首页	None
创建者	system
维护者	None
来源	None

版本信息

0.1.0 [1.0]
-------------

3、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**。

基本信息

名称 *	docs-demo
企业空间 *	hpa
项目 *	hpa

参数配置

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

4、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**。

应用

docs-demo

已启用

状态

版本

应用仓库

2018-12-14 09:42:05

上次更新时间

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。

资源状态

环境变量

操作日志

服务

docs-demo-nginx

虚拟 IP

端口: 80:HTTP/TCP

IP地址: 10.96.240.0

工作负载

docs-demo-nginx

部署

状态: 运行中(1/1)

更新时间: 2018-12-14 09:42:04

版本: 1

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

编辑信息 更多操作 ▾

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: hpa  
类型: Virtual IP

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: 运行中(1/1)  
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 并配置端口转发和防火墙规则。在端口转发规则中将内网端口 30055 转发到源端口 30055，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。

### 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 ([http://\\${EIP}:\\${NODEPORT}](http://${EIP}:${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 (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。

容器组 IP: 10.233.87.132  
主机: ks-allinone(192.168.0.18)

CPU 0.09m 内存 6.89MiB

点击左侧 **终端** 按钮，即可进入该容器的终端。

存储设备

default-token-45rr8 容量: - 存储类型: Secret 密钥: default-token-45rr8 挂载: nginx → /var/run/secrets/kubernetes.io/serviceacount read-only

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。



A screenshot of a terminal window titled "Terminal". The status bar at the top right shows a green dot icon and the text "已连接(nginx)". In the center of the window, the command "# cat /etc/hosts" is run, followed by its output:

```
# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1      localhost ip6-localhost ip6-loopback
::1             localhost ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.233.87.132  docs-demo-nginx-89954f5c8-qhlmw
#
```

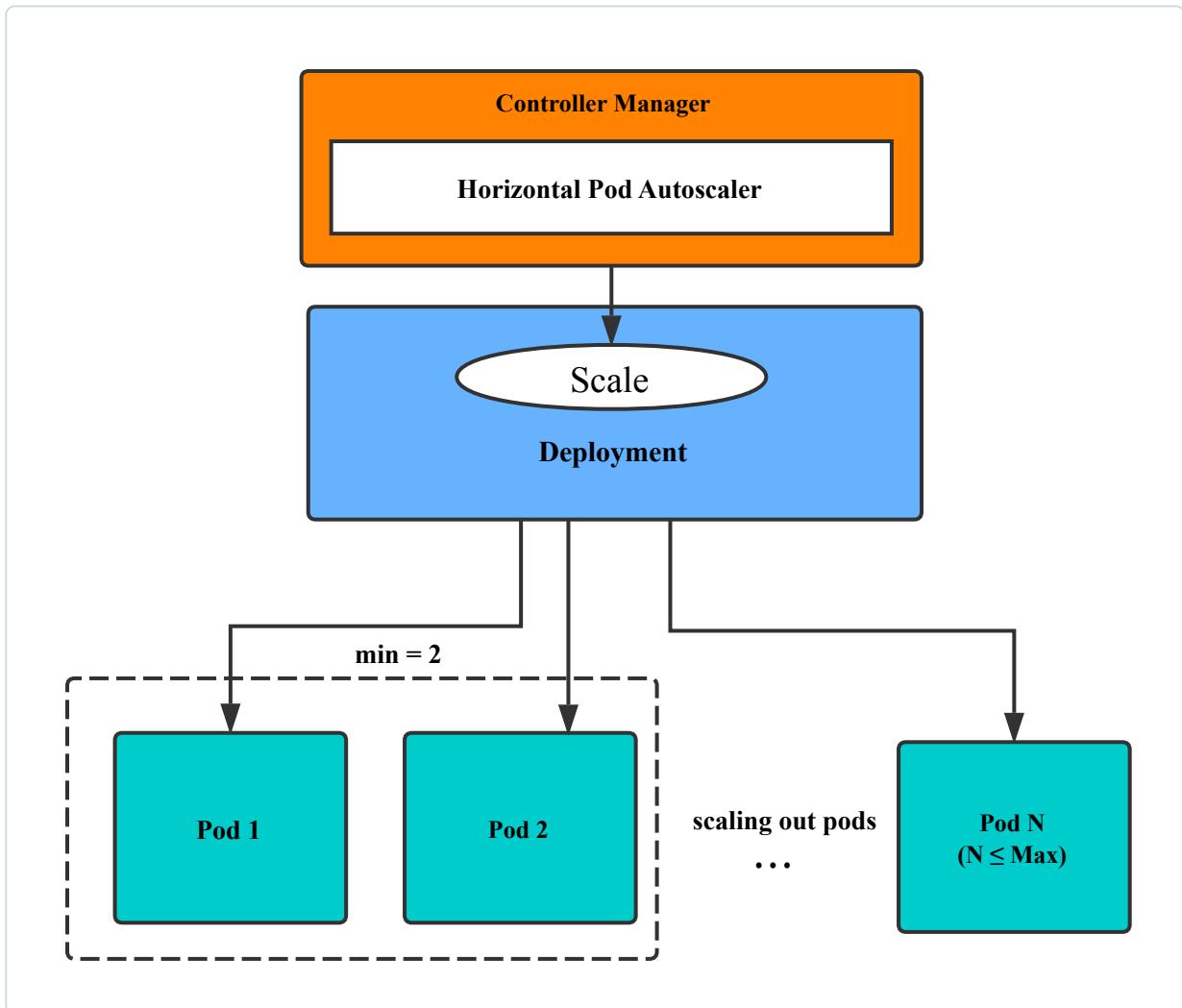
## 设置弹性伸缩

**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 Metrics-server，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。

## 预估时间

约 25 分钟。

# 前提条件

- 已创建了企业空间、项目和普通用户 `project-regular` 账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

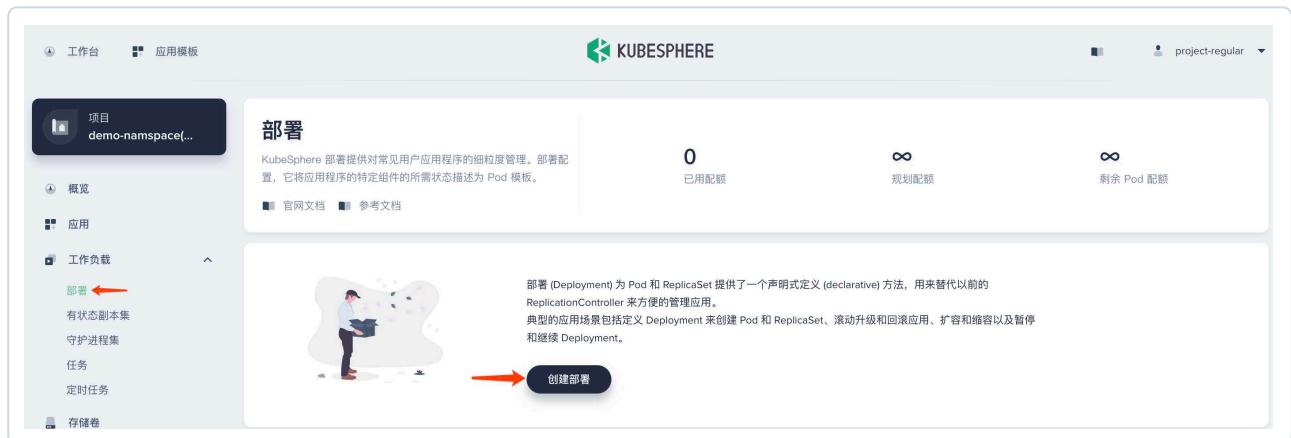
# 操作示例

## 创建 HPA

### 第一步：创建部署

1、以项目普通用户 `project-regular` 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

- 更新策略：选择 RollingUpdate

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 **1**
- 最大副本数：弹性伸缩的容器组数量上限，此处设置 **10**
- 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。

The screenshot shows the 'Container Group Template' configuration page. At the top, there are tabs: 基本信息 (Basic Information) (selected), 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). Below the tabs, the title '容器组模板' is displayed. A note says: '工作负载可以根据容器组模板以及您设置的副本数量，自动生成指定数量的容器组' (Workload can generate specified numbers of pods based on the container group template and the number of replicas you set). On the left, there is a section for '弹性伸缩' (Autoscaling) with fields for '最小副本数' (Min Replicas) set to 1, '最大副本数' (Max Replicas) set to 10, 'CPU Request Target(%)' set to 50, and 'Memory Request Target(Mi)' (Memory Request Target) which is empty. On the right, there is a large dashed box labeled '请至少添加一个容器' (Please add at least one container) with a '添加容器' (Add Container) button.

### 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 `mirrorgooglecontainers/hpa-example`，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；

The screenshot shows the 'Add Container' configuration page. At the top, there are tabs: 基本信息 (Basic Information) (selected), 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). Below the tabs, the title '添加容器' is displayed with a back arrow. There are two options: 选择已有镜像部署容器 (Select existing image for deployment) (selected) and 通过代码构建新的容器镜像 (Build new container image via code). The selected option has a note: '从公开或者私有镜像仓库中拉取镜像' (Pull image from public or private image repository). The input field for the image name is 'mirrorgooglecontainers/hpa-example'. A note below the input field says: '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。' (To deploy from a private image repository, you need to first create an image repository and then pull the image.). The second option has a note: '从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。' (Get code from an existing code repository and build the image using the Source to Image method to complete the deployment, each build process will be completed by a task).

2、下拉至**服务设置**，自定义端口名称，如 port，默认 TCP 协议，端口号为 80。



3、点击**保存**，然后点击**下一步**；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**；标签和节点选择器无需设置，点击**创建**，即可查看 HPA 示例的运行状态详情。

## 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**；

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

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

别名

HPA 服务

描述信息

This is a HPA service

## 第二步：服务设置

1、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**；

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=hpa-example)共影响到 1 个工作负载 [查看](#)

app: hpa-example

添加选择器 指定工作负载

端口 \*

port: 80 TCP 80

添加端口

会话亲和性

None

2、标签设置保留默认值 **app: hpa-example**，点击 **下一步**；

3、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** 即可创建成功。

项目 demo-namespace

概览 应用 工作负载 存储卷 网络与服务 服务 灰度发布 应用路由

服务

1 已用配额 ∞ 规划配额

官网文档 参考文档

输入查询条件进行过滤

名称	IP地址	端口	应用	创建时间
hpa-example(HPA示例)	Virtual IP: 10.233.11.237	端口: 80:80/TCP 节点端口: -	-	2019-05-14 01:21:48

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 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

### 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 **添加容器**，设置参数如下：

- 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、然后下滑至 **启动命令**，勾选 **启动命令**，在展开的运行命令和参数中填写用于对 hpa-example 服务增加 CPU 负载的命令和参数，其它设置暂无需配置。设置 **运行命令** 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`。

```
# 运行命令
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**；点击 **下一步**；本示例暂未用到存储，点击 **下一步** 跳过存储卷设置；标签设置保留默认值即可，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 **hpe-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 interface for the deployment 'hpa-example'. The 'Resource Status' tab is selected. In the 'Deployment Status' section, it shows 2/2 replicas running. The 'Horizontal Pod Autoscaler' (HPA) configuration is displayed with a target CPU usage of 50% (current 10%). The 'Ports' section shows a single port on TCP port 80.

## 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 `hpe-example` 服务时，如下图所示，刷新页面后可见 CPU 使用率明显升高，先快速上升至 5210%，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 副本数迅速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。

一分钟左右 CPU 使用率降低至 629%，副本数增加至所设置 HPA 的最大值 10，正好也证明了弹性伸缩的工作原理。

The screenshot shows the KubeSphere interface for the deployment 'hpa-example'. The 'Resource Status' tab is selected. In the 'Deployment Status' section, it shows 10/10 replicas running. The 'Horizontal Pod Autoscaler' (HPA) configuration is displayed with a target CPU usage of 50% (current 626%). The 'Ports' section shows a single port on TCP port 80.

理论上，从容器组的 CPU 监控曲线中可以看到最初创建的 1 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 **hpe-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 10 %，并且 HPA 将其副本数量最终减少至最小副本数 1，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。

hpa-example(HPA 示例)

资源状态 版本控制 监控 环境变量 事件

标签 app: hpa-example

详情

项目: demo-namespace  
应用: -  
创建时间: 2019-05-14 01:17:26  
更新时间: 2019-05-14 01:47:02  
创建者: 未知

副本运行状态 副本数: 1/1  
期望副本: 1  
实际运行副本: 1

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: 1 最大副本数: 10 目标使用率: 50% (当前 10%) 目标使用量: 0 (当前 0)

端口

名称	协议	端口
port	TCP	80

3、在部署的详情页面，可以下钻到每个 Pod 的单个容器的监控详情，点击最初创建的容器组进入容器组详情页，选择「监控」，查看该容器组的 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和 HPA 原理正好相符。

hpa-example-59544f76f8...

资源状态 环境变量 监控 事件

容器组 查看配置文件

标签 app: hpa-example pod-template-hash: 59544f76f8

详情

项目: demo-namespace  
应用: -  
状态: 运行中  
容器组 IP: 10.233.87.149  
主机名称: ks-allinone  
主机 IP: 192.168.0.20  
重启次数(总计): 0  
创建时间: 2019-05-14 01:17:26

监控

CPU 使用量 (core)

内存使用量 (MIB)

网络流出速率 (Kbps)

## 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。



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。

# Source-to-image

Source to Image (S2I) 是一个允许程序员直接输入源代码然后打包成可运行程序到 Docker 镜像的工具，在程序员不需要了解 Dockerfile 的情况下方便构建镜像。它是通过将源代码放入一个负责编译源代码的 Builder image 中，自动将编译后的代码打包成 Docker 镜像。

## 目的

本示例通过官方给出的 Hello World 的 Java 示例，演示如何在 KubeSphere 上使用 Source to Image 来实现构建镜像，并且实现自动推送到镜像仓库，最后部署到集群中。

## 前提条件

- 本示例以 GitHub 代码仓库和 DockerHub 镜像仓库为例，参考前确保已创建了 [GitHub](#) 和 [DockerHub](#) 账号；
- 已创建了企业空间、项目和普通用户 `project-regular` 账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，参考 [多租户管理快速入门 – 邀请成员](#)。

## 预估时间

20–30 分钟（时间由于网速等因素而有所不同）。

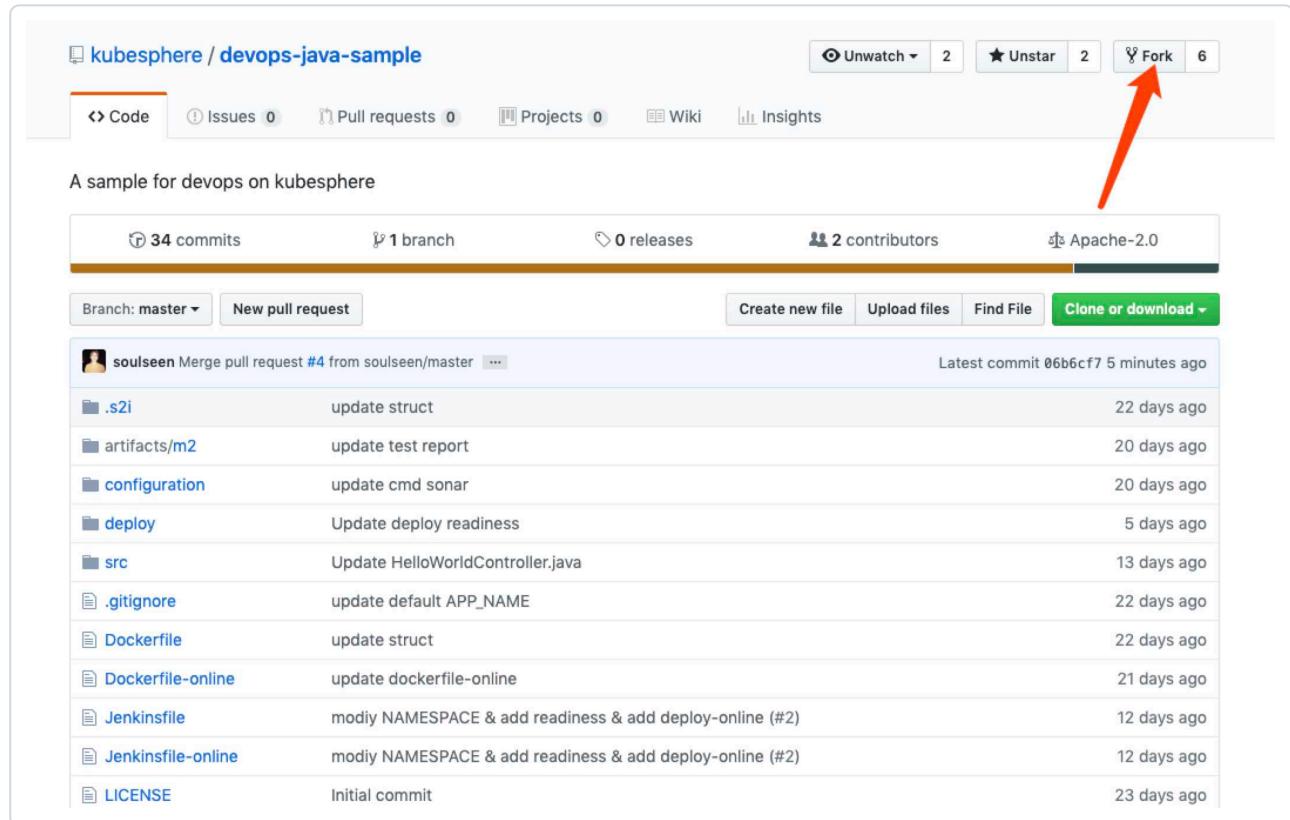
## 操作示例

### 创建密钥

需要预先创建 DockerHub 镜像仓库和 GitHub 代码仓库的密钥，分别为 `dockerhub-id` 和 `github-id`，参考 [创建常用的几类密钥](#)。

## Fork 项目

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



The screenshot shows the GitHub repository page for 'kubesphere / devops-java-sample'. At the top, there are buttons for 'Unwatch' (2), 'Unstar' (2), and a prominent green 'Fork' button with the number '6' next to it. Below the header, there are tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', and 'Insights'. A summary bar indicates '34 commits', '1 branch', '0 releases', '2 contributors', and 'Apache-2.0' license. Below this, there's a 'New pull request' button and a 'Clone or download' button. The main content area lists recent commits from user 'souleen':

Commit	Message	Date
.s2i	update struct	22 days ago
artifacts/m2	update test report	20 days ago
configuration	update cmd sonar	20 days ago
deploy	Update deploy readiness	5 days ago
src	Update HelloWorldController.java	13 days ago
.gitignore	update default APP_NAME	22 days ago
Dockerfile	update struct	22 days ago
Dockerfile-online	update dockerfile-online	21 days ago
Jenkinsfile	modiy NAMESPACE & add readiness & add deploy-online (#2)	12 days ago
Jenkinsfile-online	modiy NAMESPACE & add readiness & add deploy-online (#2)	12 days ago
LICENSE	Initial commit	23 days ago

## 创建部署

### 第一步：填写基本信息

1、在左侧的工作负载菜单下，点击部署，进入部署管理界面。

部署

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的所需状态描述为 Pod 模板。

2 已用配额

∞ 规划配额

∞ 剩余 Pod 配额

输入查询条件进行过滤

名称 状态 应用 更新时间

名称 状态 应用 更新时间

共 2 个条目

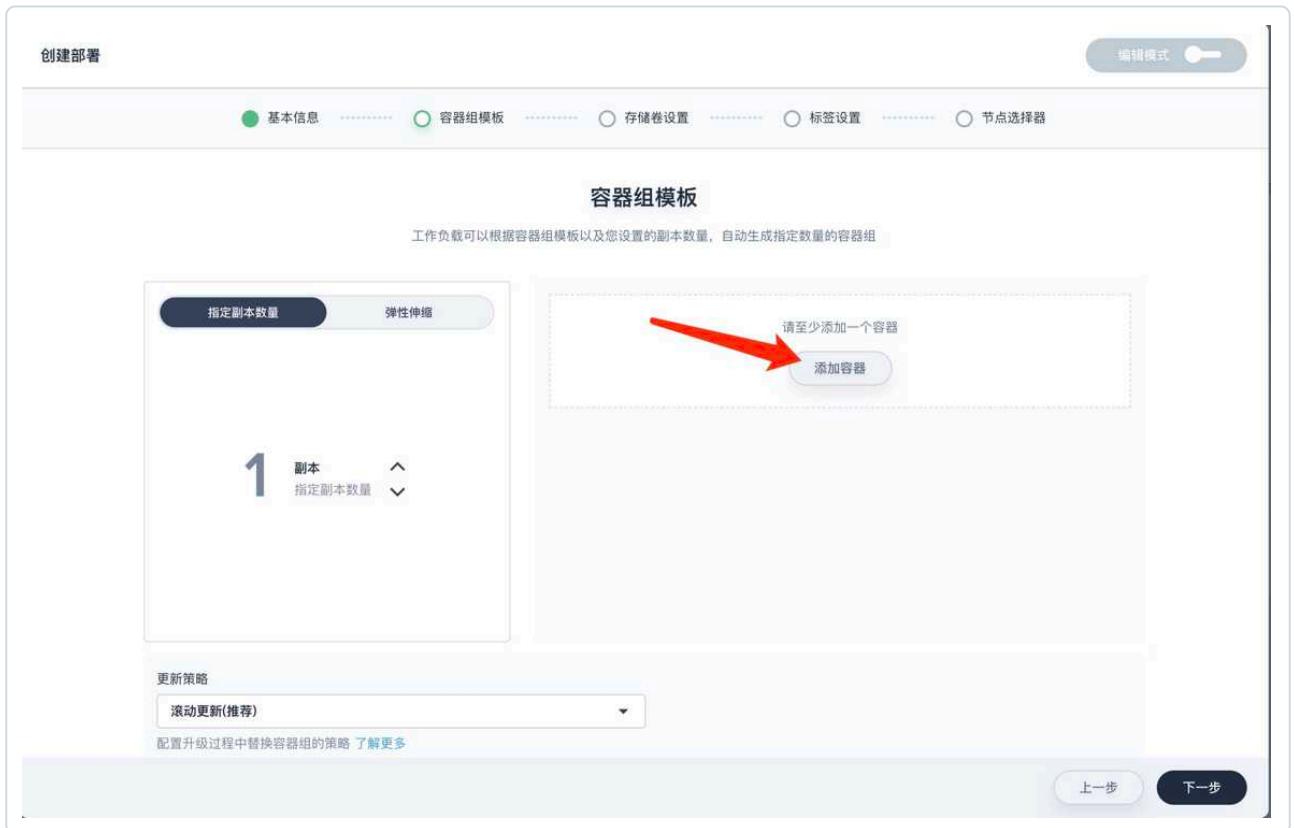
1 / 1

2、点击创建，创建一个部署。

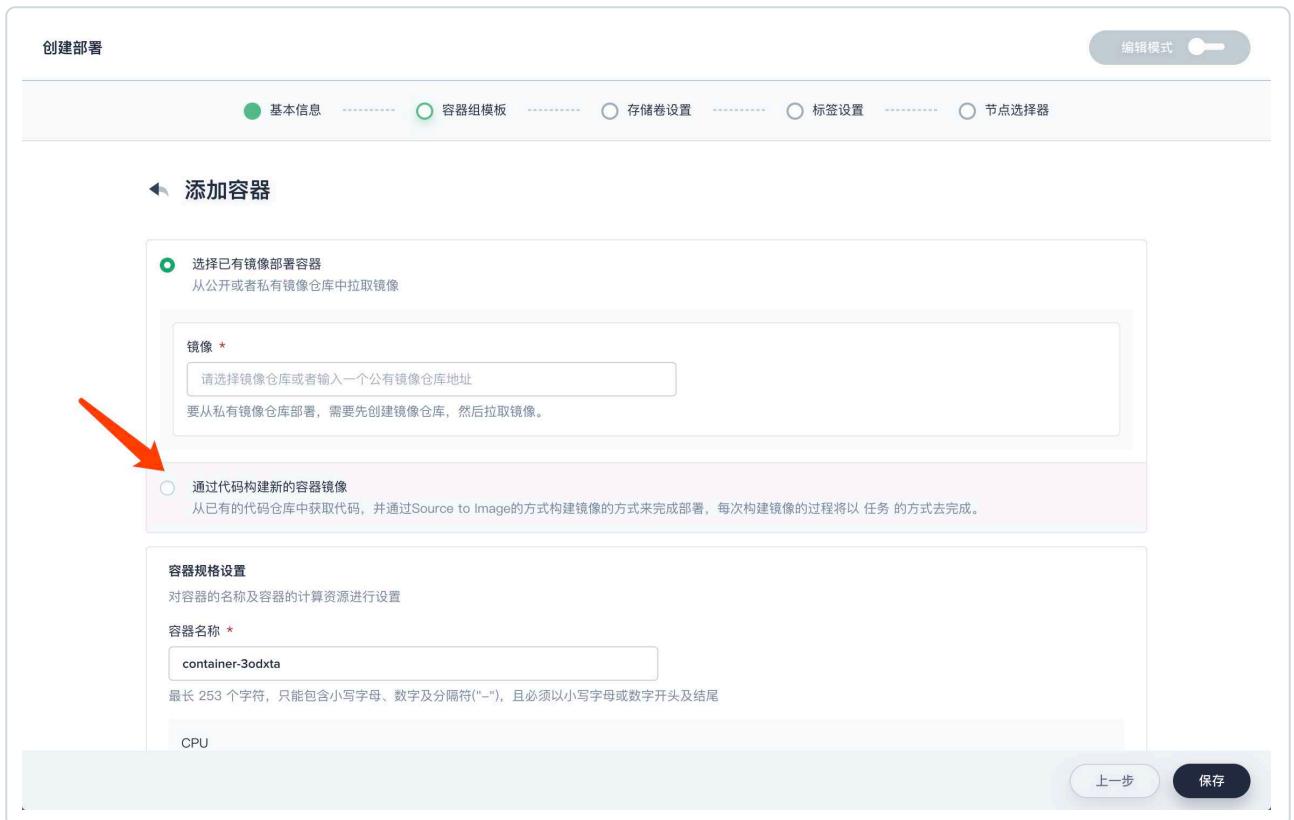
- 名称：必填，给部署起一个名字，以便在使用的时候容易区分，此处使用 s2i-test；
- 别名：为了方便理解可自定义设置；
- 描述信息：简单描述该部署的相关信息，可自定义；

## 第二步：容器组模版设置

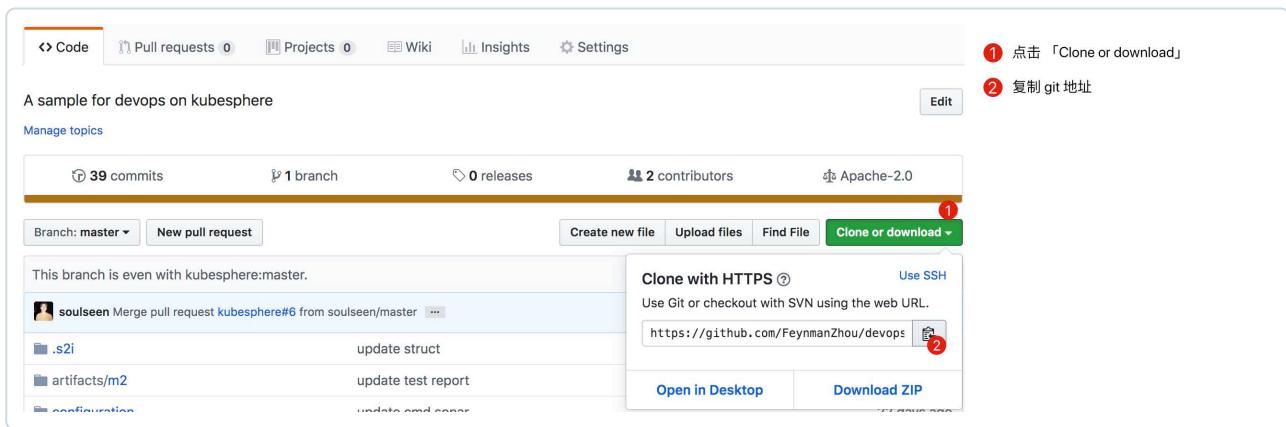
1、点击「下一步」，进入容器组模版设置界面，选择「添加容器」。



2、然后选择 **通过代码构建新的容器镜像**。



### 3、示例仓库 Fork 至您个人的 GitHub 后，复制您个人仓库的 git 地址。



### 4、参考如下提示填写信息。

**说明：**KubeSphere 内置了常用的 Java、Node.js、Python 等 s2i 的模板，若需要自定义其它语言或依赖环境的 s2i 模板，请参考 [自定义 s2i 模板](#)。

- 代码地址：粘贴上一步复制的 git 地址（目前支持 Git，支持 HTTP、HTTPS，并且可以指定代码分支以及在源代码终端的相对路径）；
- 密钥：选择之前创建的 `github-id`；
- 映像模板：选择 `kubesphere/dev/java-8-centos7` 作为此示例的 Builder image；
- 代码相对路径：使用默认的 `/` 即可；
- 映像名称：可根据自己情况定义，此示例使用 `<dockerhub_username>/hello`，`dockerhub_username` 为自己的账户名称，确保具有推拉权限；
- tag：镜像标签使用默认 `latest` 即可；
- 目标镜像仓库：选择之前创建的 `dockerhub-id`。

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

### 添加容器

选择已有镜像部署容器  
从公开或者私有镜像仓库中拉取镜像

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以任务的方式去完成。

代码地址 *	秘钥
<input type="text" value="https://github.com/soulseen/devops-sample-s2i.git"/>	<input type="text" value="github-id"/>
源代码仓库的址（目前支持git）并且可以指定代码分支及在源代码终端相对路径地	
映像模板 *	代码相对路径(可选):
<input type="text" value="kubesphere/kubesphere-dev/java-8-centos7"/>	<input type="text" value="/"/>
选择编辑环境，您也可以查看对应的编译模板	
映像名称 *	tag *
<input type="text" value="zhuxiaoyang/hello"/>	<input type="text" value="latest"/>
镜像名称及Tag，默认为代码仓库的项目名称	
目标镜像仓库:	
<input type="text" value="dockerhub"/>	
需要选择一个有推送权限的镜像仓库存放镜像，如果没有可以新建镜像仓库密钥	
环境变量参数	

上一步 保存

4、往下滑动至 **容器规格设置**，建议最大 CPU 和最大内存设置为 500m 和 1000Mi

容器规格设置  
对容器的名称及容器的计算资源进行设置

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

CPU

最小使用	<input type="text" value="10"/>	m	最大使用	<input type="text" value="500"/>	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	<input type="text" value="10"/>	Mi	最大使用	<input type="text" value="1000"/>	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。

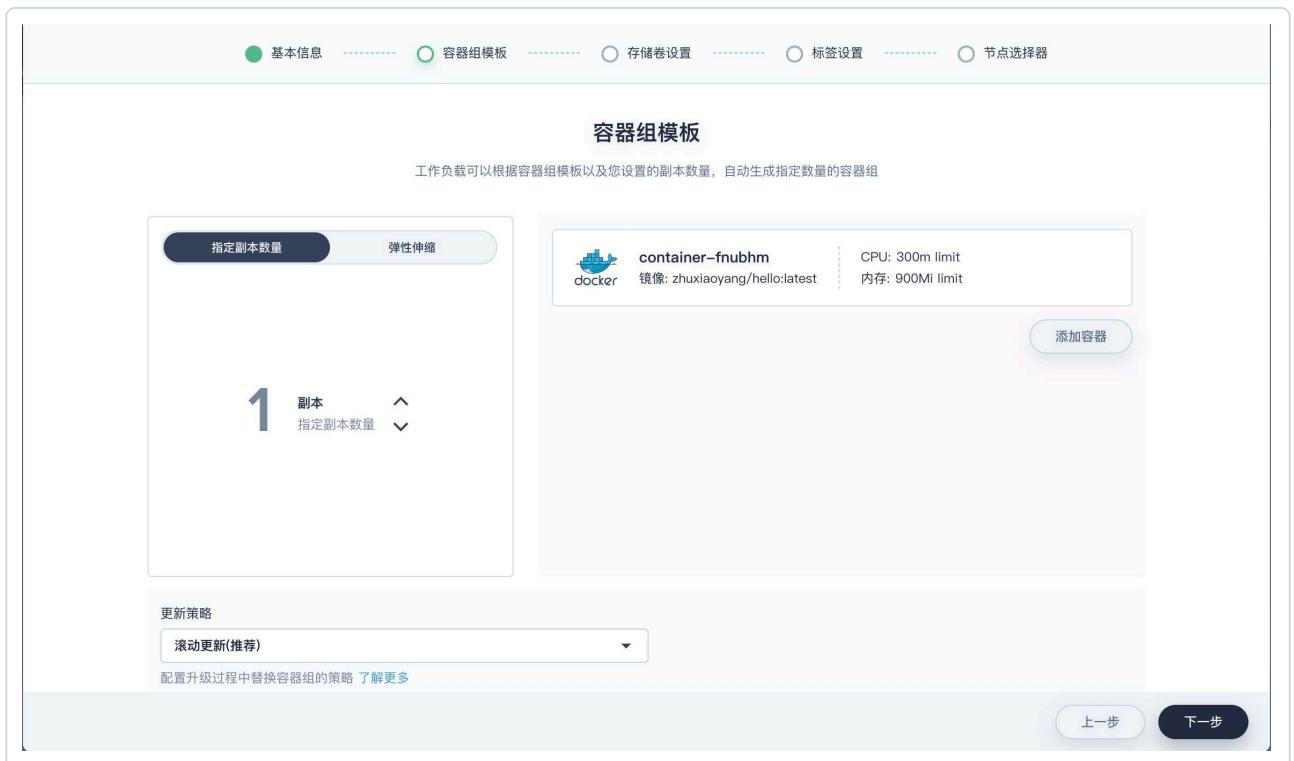
服务设置  
设置容器的访问策略

5、往下滑至 **服务设置**，配置端口为 8080，如：



## 6、然后点击「保存」

副本数量可选择为 1，然后点击「下一步」。

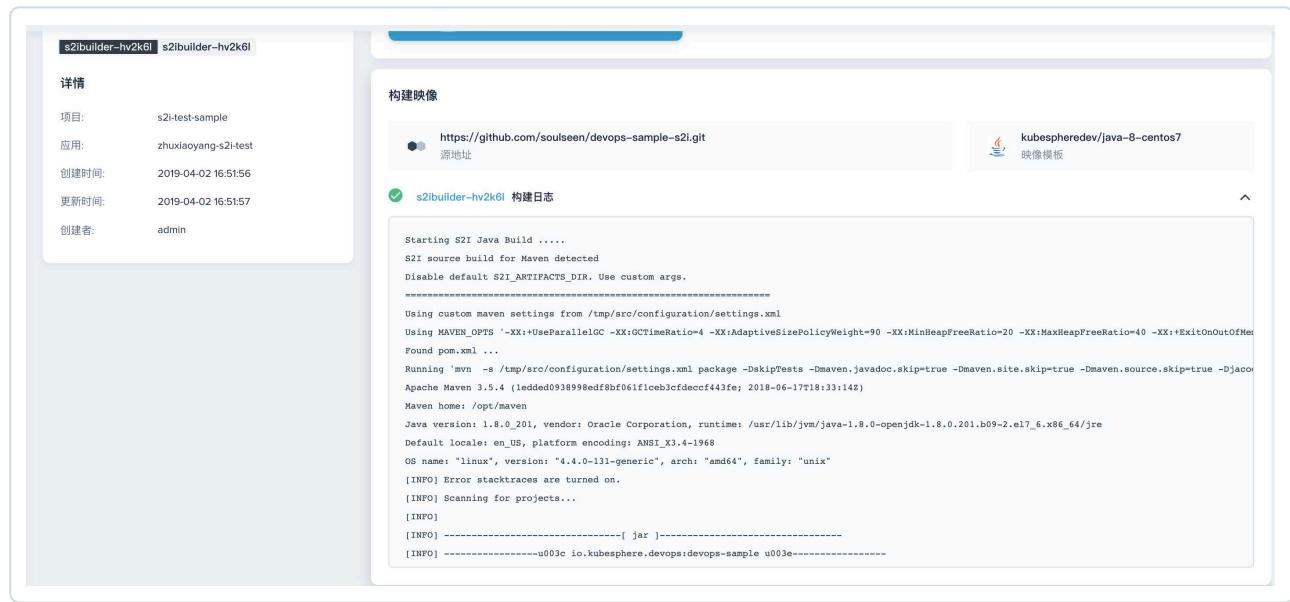


## 第三步：创建 s2i 部署

本示例无需配置存储卷，点击「下一步」，标签保留默认值即可，选择「创建」，s2i 示例部署创建完成。

# 构建完成

出现如下图绿勾即表示镜像通过 s2i 构建成功。



查看容器组，正常运行。

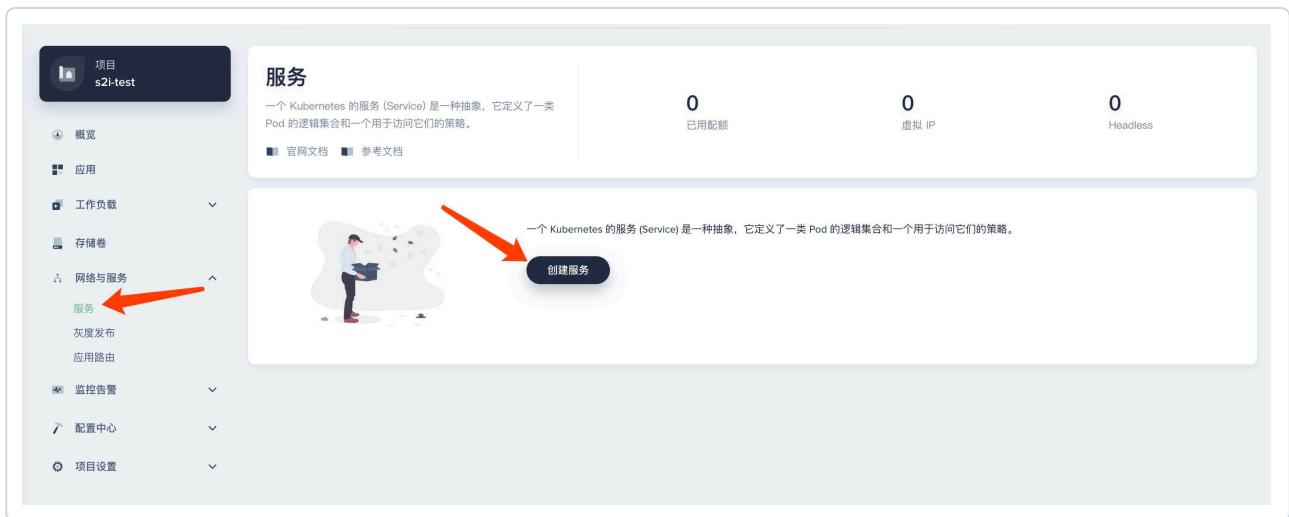


## 验证运行结果

若通过 s2i 部署顺利，则将会在设置的 Dockerhub 中查看到设置的镜像，名称和标签为 **容器组模版设置** 中设置的值。若想在浏览器中查看到部署结果，可进行如下配置。

## 第一步：创建服务

选择左侧网络与服务下的服务，点击「创建服务」。



## 第二步：基本信息填写

基本信息与创建部署类似，这里名称填写示例名称 `s2i-test-service`，其余可根据自己情况填写，点击「下一步」，如下图。

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 \*

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

**下一步**

## 第三步：服务设置

1、服务类型选择第一项 **通过集群内部IP来访问服务 Virtual IP**。

2、然后点击「指定工作负载」，选择刚刚创建的名称为 **s2i-test** 的部署，如下图。



3、点击保存，参考如下提示配置端口信息。

- 端口名称：s2i-port；
- 协议默认 TCP，端口号：**8080**，目标端口为 **8080**；
- 设置完成后点击「下一步」。

选择器 \*

⚠️ 当前设置的选择器(app=s2i-demo, s2ibuilder-9q9n1n=s2ibuilder-9q9n1n)共影响到 1 个工作负载 查看

app	s2i-demo
s2ibuilder-9q9n1n	s2ibuilder-9q9n1n

端口 \*

s2i-port	TCP	8080	8080
----------	-----	------	------

会话亲和性

None
------

添加选择器 指定工作负载

添加端口

## 第五步：标签设置

默认即可，点击「下一步」。

## 第六步：配置外网访问

选择访问方式为 **NodePort**。



至此，查看服务创建完成，如下图所示，Virtual IP 为 10.233.40.25，服务端口设置的是 8080，节点端口 (NodePort) 为 30454。

The screenshot shows the 'Services' page in the KubeSphere UI. On the left, there's a sidebar with '项目 test-s2i' and various navigation options like '概览', '应用', '工作负载', '存储卷', '网络与服务', '服务', '灰度发布', '应用路由', '监控告警', and '配置中心'. The main area is titled '服务' and contains 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 shows 1 Pod 已经分配，0 虚拟 IP，0 Headless。下方有一个搜索栏 '输入查询条件进行过滤' and a table with columns '名称', 'IP地址', '端口', '应用', and '创建时间'. There is one entry: 's2i-test-service' with 'Virtual IP: 10.233.40.25' and '节点端口: 30454 TCP'. The '节点端口' value is highlighted with a red box.

## 第七步：验证访问

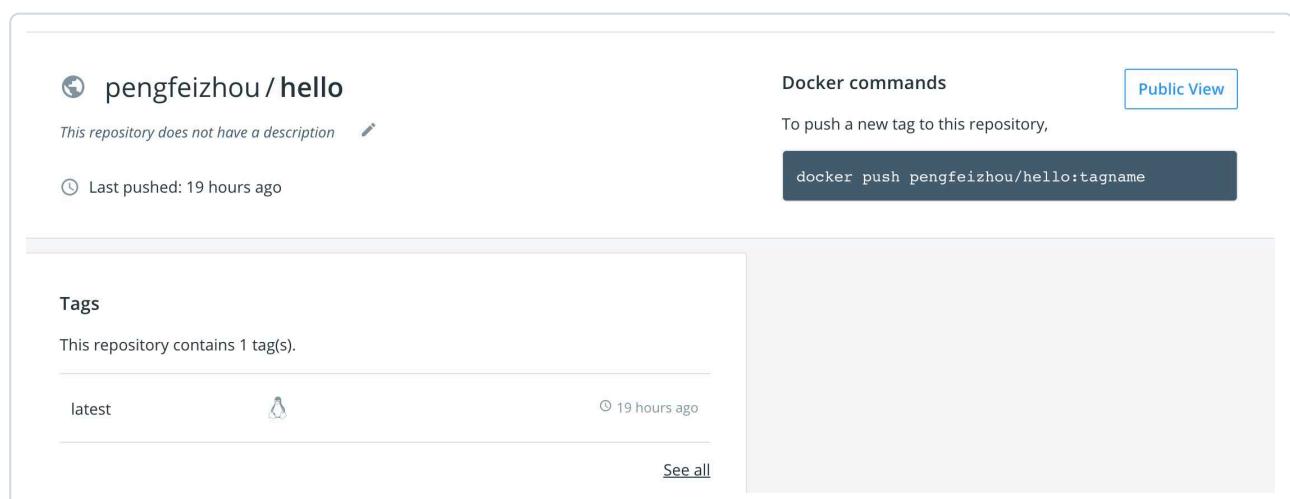
若在内网环境访问部署的 HelloWorld 示例服务，可通过 SSH 登陆集群节点，或使用集群管理员登陆 KubeSphere 在 web kubectl 中输入以下命令验证访问：

```
# curl {${Virtual IP}:${$Port} 或者 curl {${内网 IP}}:${$NodePort}
curl 10.233.40.25:8080
Hello,World!
```

提示：若需要在外网访问该服务，可能需要绑定公网 EIP 并配置端口转发和防火墙规则。在端口转发规则中将内网端口 30454 转发到源端口 30454，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。

## 查看推送的镜像

由于我们在容器组模板设置中设置的目标镜像仓库为 DockerHub，此时可以登录您个人的 DockerHub 查看 Source to Image 示例推送的镜像，以下验证发现 `hello:latest` 镜像已成功推送至 DockerHub。



# Bookinfo 微服务的灰度发布示例

灰度发布，是指在黑与白之间，能够平滑过渡的一种发布方式。通俗来说，即让产品的迭代能够按照不同的灰度策略对新版本进行线上环境的测试，灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以对新版本进行测试、发现和调整问题，以保证其影响度。KubeSphere 基于 [Istio](#) 提供了蓝绿部署、金丝雀发布、流量镜像等三种灰度策略，无需修改应用的服务代码，即可实现灰度、流量治理、Tracing、流量监控、调用链等服务治理功能，关于每一种策略的描述参见 [灰度发布](#)。

## 目的

本示例在 KubeSphere 中使用 Istio 官方提供的 [Bookinfo](#) 示例，创建一个微服务应用并对其中的服务组件进行灰度发布，演示 KubeSphere 服务治理的能力。

## Bookinfo 微服务应用架构

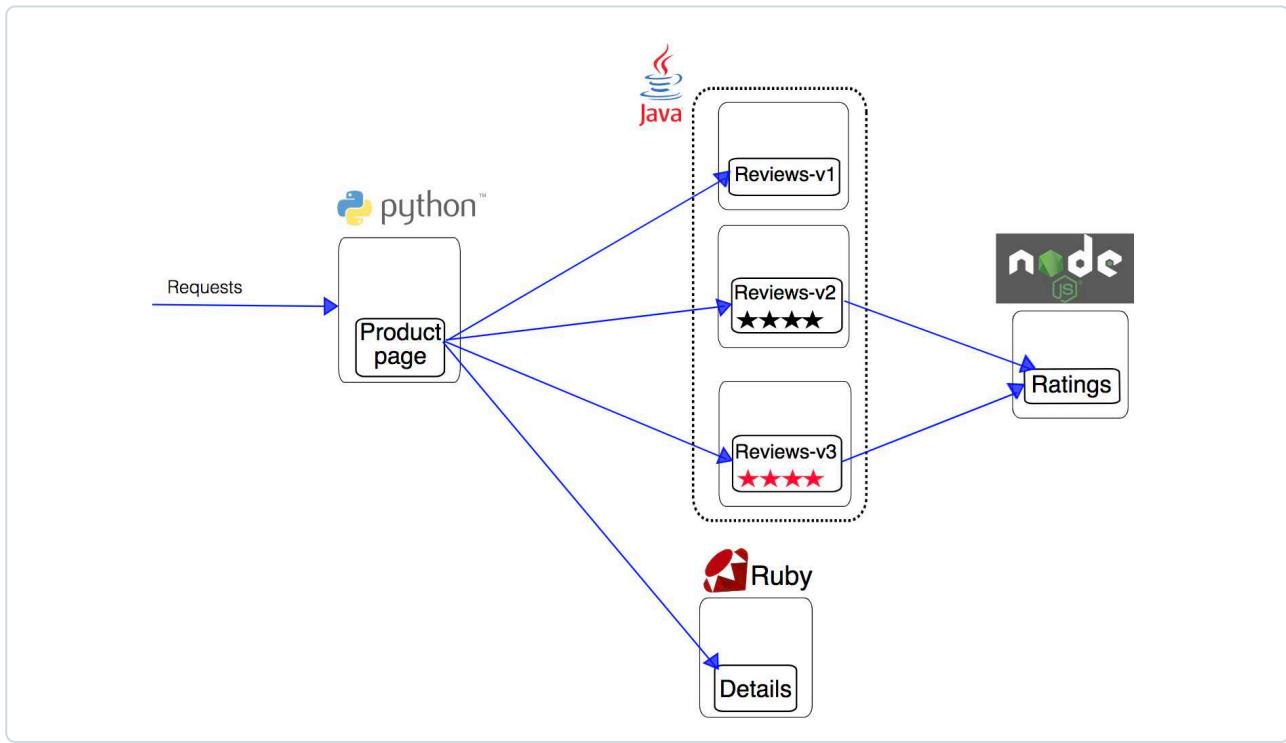
Bookinfo 应用分为四个单独的微服务：

- productpage：productpage 微服务会调用 details 和 reviews 两个微服务，用来生成页面。
- details：这个微服务包含了书籍的信息。
- reviews：这个微服务包含了书籍相关的评论，它还会调用 ratings 微服务。
- ratings：ratings 微服务中包含了由书籍评价组成的评级信息。

reviews 微服务有 3 个版本：

- v1 版本不会调用 ratings 服务，因此在界面不会显示星形图标。
- v2 版本会调用 ratings 服务，并使用 1 到 5 个黑色星形图标来显示评分信息。
- v3 版本会调用 ratings 服务，并使用 1 到 5 个红色星形图标来显示评分信息。

下图展示了这个应用的端到端架构。



(Bookinfo 架构图与示例说明参考自 <https://istio.io/docs/examples/bookinfo/>)

## 预估时间

约 20 ~ 30 分钟。

## 前提条件

- 已创建了企业空间、项目和普通用户 `project-regular` 账号, 请参考 [多租户管理快速入门](#);
- 查看 Tracing 需在当前项目下选择 「项目设置」 → 「外网访问」 → 「设置网关」, 点击「应用治理」的开启按钮 (注意访问方式为 `NodePort`);
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色, 若还未邀请请参考 [多租户管理快速入门 – 邀请成员](#)。

# 操作示例

## 创建自制应用

1. 使用项目普通用户 `project-regular` 账号进入项目 `demo-namespace` 后，点击「应用」，选择「自制应用」，点击「部署示例应用」。



2. 在弹窗中，点击「确定」部署 bookinfo 示例应用。
3. 确认应用状态显示 `Ready`，则说明 bookinfo 微服务创建成功。



## 访问 Bookinfo 应用

1. 点击 bookinfo 进入应用详情页，可以看到应用路由下自动生成的 hostname。

The screenshot shows the KubeSphere interface. At the top, there are tabs: 应用组件 (highlighted in green), 流量治理, 灰度发布, Tracing, and 事件. Below the tabs, under the heading '应用路由', there is a card for 'bookinfo-demo-ingress-da3616'. It displays the hostname 'productpage.demo-namespace.192.168.0.8.nip.io' with a red arrow pointing to it. Below the hostname is the URL 'http://productpage.demo-namespace.192.168.0.8.nip.io:31680/' and a '点击访问' (Click to Visit) button. Under the heading '服务', there are two cards: one for 'details' (灰度版本: --, 容器组数量: 1) and one for 'productpage' (灰度版本: --, 容器组数量: 1). Both cards show internal access information: IP address and port.

- 由于外网访问启用的是 NodePort 的方式，NodePort 会在主机上开放 http 端口，要访问 bookinfo 应用需要将该端口进行转发并在防火墙添加下行规则，确保流量能够通过该端口。

例如在 QingCloud 云平台进行上述操作，假设外网访问开启的主机端口 NodePort 为 31680 (http)，则可以参考 [云平台配置端口转发和防火墙](#)。

- 在本地打开 `/etc/hosts` 文件，为 hostname 添加一条记录，例如：

```
#{公网 IP} {hostname}  
139.198.111.111 productpage.demo-namespace.192.168.0.8.nip.io
```

- 完成上述步骤后，在应用路由下选择「点击访问」，可以看到 bookinfo 的 details 页面。

← → ⌂ ⓘ 不安全 | productpage.demo-namespace.192.168.0.8.nip.io:31680

Hello! This is a simple bookstore application consisting of three services as shown below

endpoint	details															
<b>name</b>	http://details:9080															
<b>children</b>	<table border="1"> <thead> <tr> <th>endpoint</th> <th>name</th> <th>children</th> </tr> </thead> <tbody> <tr> <td>details</td> <td>http://details:9080</td> <td></td> </tr> <tr> <td>reviews</td> <td>http://reviews:9080</td> <td> <table border="1"> <thead> <tr> <th>endpoint</th> <th>name</th> <th>chil</th> </tr> </thead> <tbody> <tr> <td>ratings</td> <td>http://ratings:9080</td> <td></td> </tr> </tbody> </table> </td> </tr> </tbody> </table>	endpoint	name	children	details	http://details:9080		reviews	http://reviews:9080	<table border="1"> <thead> <tr> <th>endpoint</th> <th>name</th> <th>chil</th> </tr> </thead> <tbody> <tr> <td>ratings</td> <td>http://ratings:9080</td> <td></td> </tr> </tbody> </table>	endpoint	name	chil	ratings	http://ratings:9080	
endpoint	name	children														
details	http://details:9080															
reviews	http://reviews:9080	<table border="1"> <thead> <tr> <th>endpoint</th> <th>name</th> <th>chil</th> </tr> </thead> <tbody> <tr> <td>ratings</td> <td>http://ratings:9080</td> <td></td> </tr> </tbody> </table>	endpoint	name	chil	ratings	http://ratings:9080									
endpoint	name	chil														
ratings	http://ratings:9080															

Click on one of the links below to auto generate a request to the backend as a real user or a tester

Normal user

Test user

5. 点击 **Normal user** 访问 productpage。注意此时 Book Reviews 部分只显示了 Reviewer1 和 Reviewer2。

BookInfo Sample

Sign in

### The Comedy of Errors

Summary: [Wikipedia Summary](#): The Comedy of Errors is one of William Shakespeare's early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

**Book Details**

Type: paperback  
Pages: 200  
Publisher: PublisherA  
Language: English  
ISBN-10: 1234567890  
ISBN-13: 123-1234567890

**Book Reviews**

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!  
— Reviewer1

Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.  
— Reviewer2

## 添加灰度发布

1. 回到 KubeSphere，选择「灰度发布」，点击「发布灰度任务」。

bookinfo

应用组件 流量治理 灰度发布 ① Tracing 事件

① 选择「灰度发布」  
② 点击「发布灰度任务」

暂无进行中的灰度任务  
您可以绑定灰度策略进行灰度任务发布

共计 0 个灰度任务

2. 在跳转的灰度发布页面，选择「金丝雀发布」作为灰度策略，点击「发布任务」。



3. 在弹窗中，填写发布任务名称为 `bookinfo-canary`，点击「下一步」。
4. 点击 `reviews` 一栏的「选择」，即选择对应用组件 `reviews` 进行灰度发布，点击「下一步」。
5. 参考如下填写灰度版本信息，完成后点击「下一步」。
  - 灰度版本号：v2；
  - 镜像：`istio/examples-bookinfo-reviews-v2:1.10.1` (将 v1 改为 v2)。
6. 金丝雀发布允许按流量比例下发与按请求内容下发等两种发布策略，来控制用户对新老版本的请求规则。本示例选择 **按流量比例下发**，流量比例选择 v1 与 v2 各占 50%，点击「创建」。



## 验证金丝雀发布

再次访问 Bookinfo 网站，重复刷新浏览器后，可以看到 bookinfo 的 reviews 模块在 v1 和 v2 模块按

50% 概率随机切换。

The screenshot shows a web application interface for 'BookInfo Sample'. At the top, there's a dark header bar with the text 'BookInfo Sample' on the left and a 'Sign in' button on the right. Below the header, the title 'The Comedy of Errors' is displayed in bold. A summary text follows, mentioning it's one of William Shakespeare's early plays, known for slapstick humour and puns. To the left, under 'Book Details', there's a table of book information:

Type:	paperback
Pages:	200
Publisher:	PublisherA
Language:	English
ISBN-10:	1234567890
ISBN-13:	123-1234567890

To the right, under 'Book Reviews', there are two reviews:

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!  
— Reviewer1  
★★★★★

Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.  
— Reviewer2  
★★★★☆

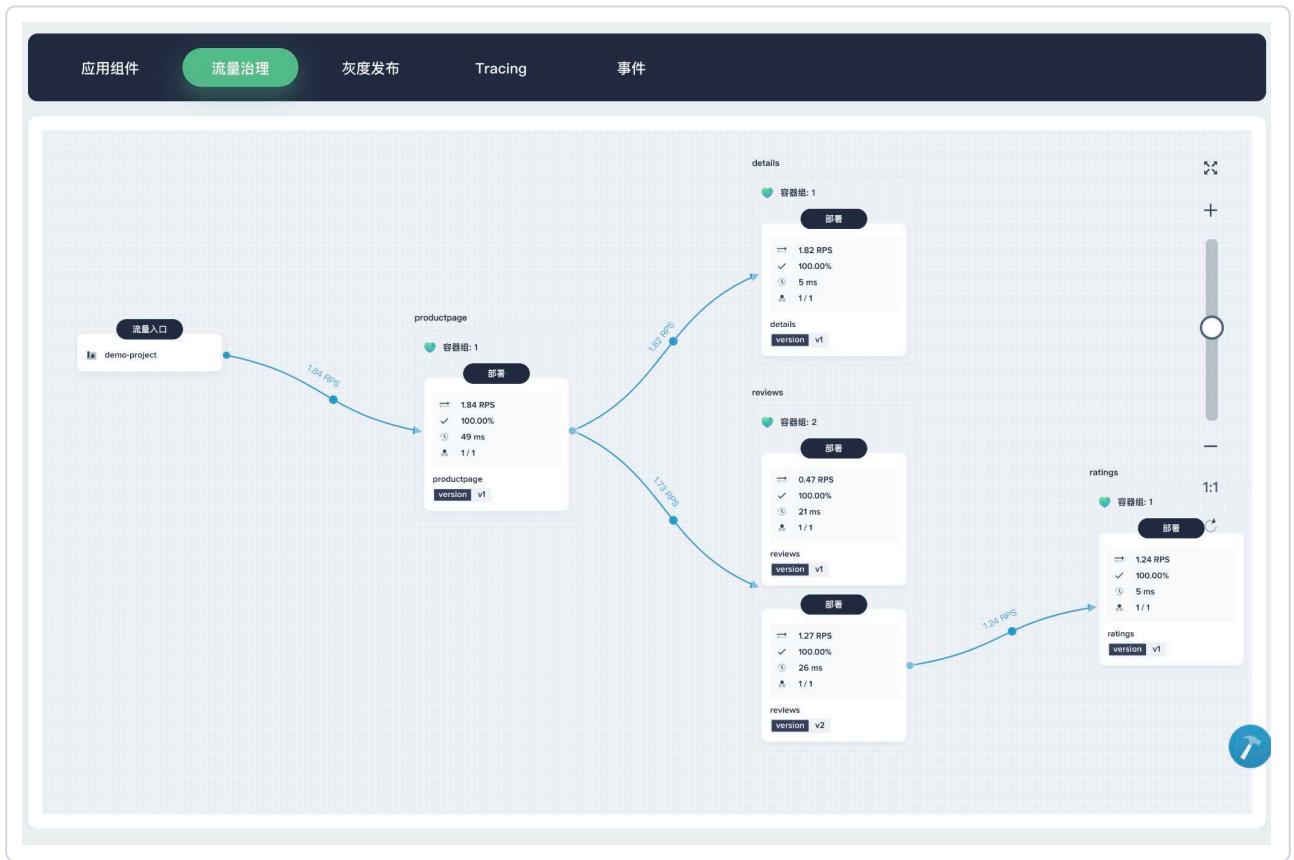
## 查看流量拓扑图

打开命令行窗口输入以下命令，引入真实的访问流量，模拟对 bookinfo 应用每 0.5 秒访问一次。注意以下命令是模拟 `Normal user` 访问，需要输入完整的命令访问到具体的服务在链路图中才有流量数据。

```
$ watch -n 0.5 "curl http://productpage.demo-namespace.139.198.111.111.nip.io:31680/productpage?u=
```

从流量治理的链路图中，可以看到各个微服务之间的服务调用和依赖、健康状况、性能等情况。

**提示：**点击其中一个应用组件，还可以为该服务组件设置流量治理策略，如连接池管理、熔断器等，详见 [流量治理](#)。



## 查看流量监测

点击 reviews 服务，查看该服务的实时流量监测，包括每秒请求的流量 (RPS)、成功率、持续时间等指标，这类指标都可以分析该服务的健康状况和请求成功率。



## 查看 Tracing

如果在链路图中发现了服务的流量监测异常，还可以在 Tracing 中追踪一个端到端调用经过了哪些服务，以及各个服务花费的时间等详细信息，支持进一步查看相关的 Header 信息，每个调用链由多个 Span 组成。

界面上可以清晰的看到某个请求的所有阶段和内部调用，以及每个阶段所耗费的时间。

The screenshot shows the KubeSphere UI for the bookinfo application. On the left, there's a sidebar for the 'bookinfo' service with options to edit or delete it. The main area is titled 'reviews' and shows the 'Tracing' tab. It lists several tracing sessions with their execution times (e.g., 20.699 ms, 23.536 ms, 27.398 ms, 24.427 ms, 84.681 ms) and the number of spans (e.g., 5 Spans, 7 Spans). Each session includes a color-coded legend for components like details.servicemesh, productpage.servicemesh, and reviews.servicemesh.

展开某个阶段，还能下钻看到这个阶段是在哪台机器（或容器）上执行的。

This screenshot shows a detailed tracing view for a specific span. At the top, there's a 'Service & Operation' section with a timeline chart. The chart shows various components contributing to the total latency: reviews.servicemesh (4.84ms), details.servicemesh (3.73ms), productpage.servicemesh (68.73ms), ratings.servicemesh (67.97ms), and reviews:9080/\* (1.56ms, 0.93ms). Below the chart is a table of tags and process details, including information like component (proxy), node\_id, guid, http.url, http.method, and downstream\_cluster.

## 接管全部流量

当新版本 v2 灰度发布后，发布者可以对线上的新版本进行测试和收集用户反馈。如果测试后确定新版本没有问题，希望将流量完全切换到新版本，则进入灰度发布页面进行流量接管。

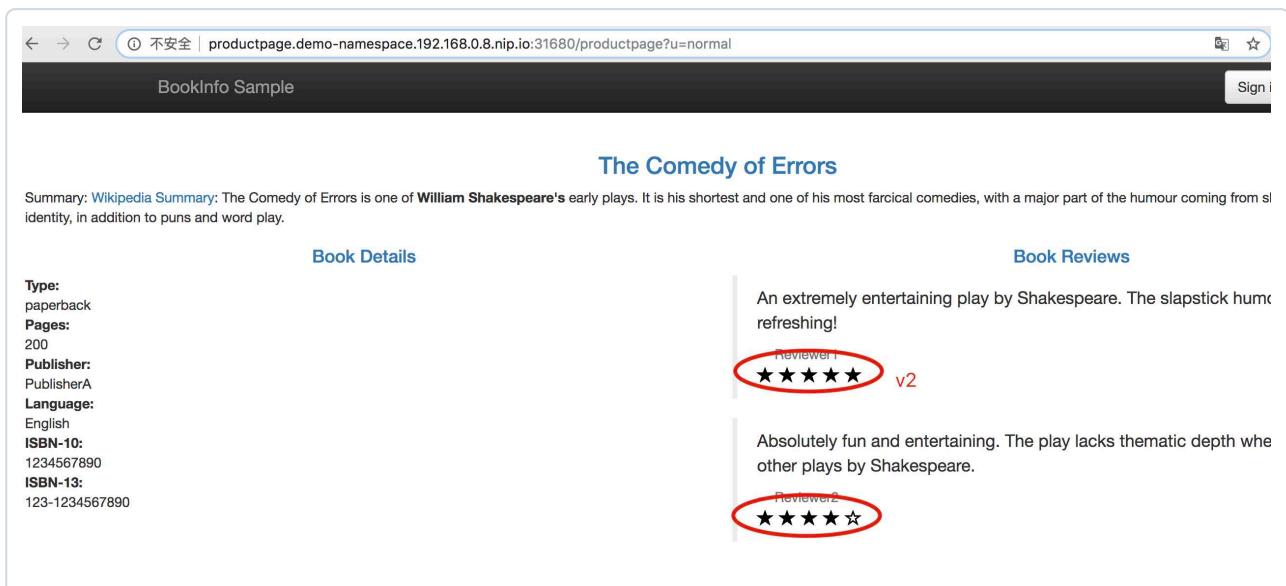
1. 点击「灰度发布」，进入 bookinfo 的灰度发布详情页，点击 … 选择「接管所有流量」，正常情

况下所有流量将会指向 v2。

**提示：此时 v1 也将保持在线，若 v2 上线后发现问题，允许发布者随时将新版本 v2 的流量切回 v1。**



2. 再次打开 bookinfo 页面，多次刷新后 reviews 模块也仅仅只显示 v2 版本。



## 下线旧版本

当新版本 v2 上线接管所有流量后，并且测试和线上用户反馈都确认无误，即可下线旧版本，释放资源 v1 的资源。下线应用组件的特定版本，会同时将关联的工作负载和 istio 相关配置资源等全部删除。

任务状态

现在可下线任务，版本 v1 将被移除。

**test-canary**  
灰度策略: Canary

点击「任务下线」释放 v1 资源

灰度组件  
将一部分实际流量引入一个新版本进行测试，测试新版本的性能和表现，在保证系统整体稳定运行的前提下，尽早发现新版本在实际环境上的问题。

reviews v2 副本: 1/1 reviews-v2-6f6b5b594c-zpj2p 7.08 m 144.50 MiB

reviews v1 副本: 1/1 reviews-v1-86dd4f8bb7-d8pwd 5.58 m 143.62 MiB

实时流量分布  
将所有的流量按比例分配给灰度组件

v2 traffic 100%

任务下线

至此，Bookinfo 微服务以金丝雀发布作为发布策略，演示了灰度发布的 basic 功能。

# 基于Spring Boot项目构建流水线

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，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 Hello World 页面部署到 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。

## 前提条件

- 本示例以 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 工程并且创建了项目普通用户 project-regular 的账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入 DevOps 工程并授予 `maintainer` 角色，若还未邀请请参考 [多租户管理快速入门 - 邀请成员](#)。

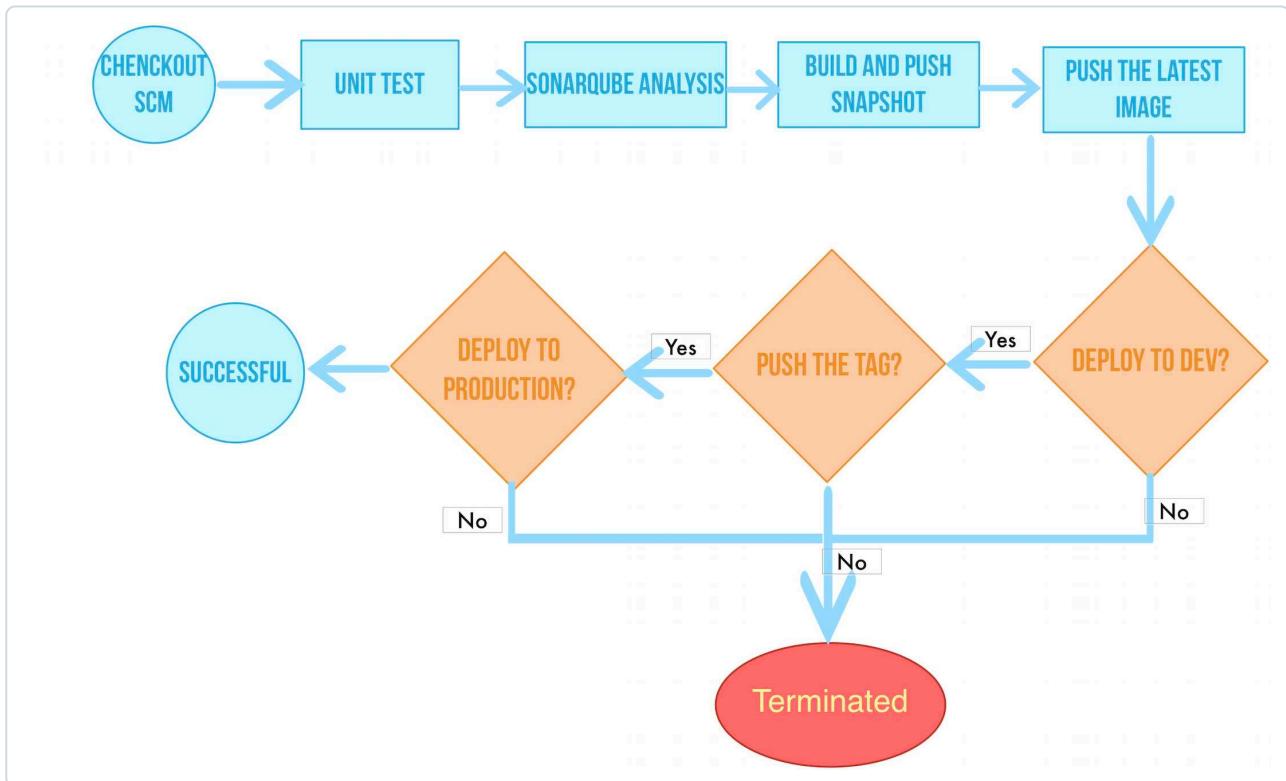
## 预估时间

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

### 流水线概览

下面的流程图简单说明了流水线完整的工作过程：



#### 流程说明：

- 阶段一. Checkout SCM: 拉取 GitHub 仓库代码
- 阶段二. Unit test: 单元测试, 如果测试通过了才继续下面的任务
- 阶段三. sonarQube analysis: sonarQube 代码质量检测
- 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 Harbor (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。

## 创建凭证

在 [多租户管理快速入门](#) 中已给项目普通用户 `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-demo` 工程, 开始创建凭证。

- 1、本示例代码仓库中的 Jenkinsfile 需要用到 DockerHub、GitHub 和 kubeconfig (kubeconfig 用于访问接入正在运行的 Kubernetes 集群) 等一共 3 个凭证 (credentials)，参考 [创建凭证](#) 依次创建这三个凭证。
- 2、然后参考 [访问 SonarQube 并创建 Token](#)，创建一个 Java 的 Token 并复制。
- 3、最后在 KubeSphere 中进入 `devops-demo` 的 DevOps 工程中，与上面步骤类似，在 **凭证** 下点击 **创建**，创建一个类型为 **秘密文本** 的凭证，凭证 ID 命名为 **sonar-token**，密钥为上一步复制的 token 信息，完成后点击 **确定**。

### 创建凭证

凭证 ID \*

类型

秘钥

描述信息

取消 确定

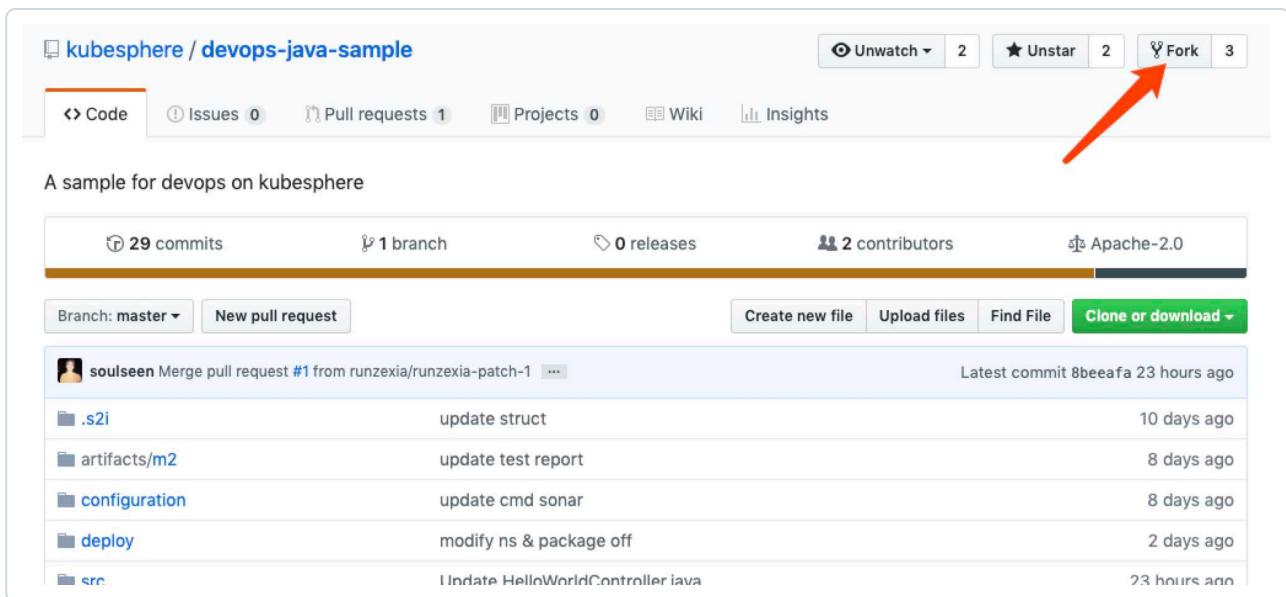
至此，4 个凭证已经创建完成，下一步需要在示例仓库中的 jenkinsfile 修改对应的四个凭证 ID 为用户自己创建的凭证 ID。

名称	类型	描述信息	创建时间
github-id	账户凭证		2019-04-09 13:35:47
demo-kubeconfig	kubeconfig		2019-04-09 13:36:13
dockerhub-id	账户凭证		2019-04-11 18:38:07
sonar-token	秘密文本		2019-04-11 18:38:45

## 修改 Jenkinsfile

### 第一步：Fork项目

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



The screenshot shows the GitHub repository page for 'devops-java-sample'. The header includes the repository name, a watch count of 2, an unstar count of 2, and a fork count of 3. Below the header, there are tabs for Code, Issues (0), Pull requests (1), Projects (0), Wiki, and Insights. The main content area displays a sample Java code snippet. At the bottom, there are statistics: 29 commits, 1 branch, 0 releases, 2 contributors, and Apache-2.0 licensing. A 'Clone or download' button is also present. A red arrow points to the 'Fork' button in the top right corner of the header.

### 第二步：修改 Jenkinsfile

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

Branch: master ▾ New pull request

Create new file Upload files Find File Clone or download ▾

This branch is 2 commits ahead of kubesphere:master.

Pull request Compare

 soulseen	Update Jenkinsfile	Latest commit f41fd02 5 days ago
 .s2i	update struct	7 days ago
 artifacts/m2	update test report	5 days ago
 configuration	update cmd sonar	5 days ago
 deploy	init	7 days ago
 src	update struct	7 days ago
 .gitignore	update default APP_NAME	7 days ago
 Dockerfile	update struct	7 days ago
 Dockerfile-online	update dockerfile-online	6 days ago
 Jenkinsfile	Update Jenkinsfile	5 days ago
 Jenkinsfile-online	Update Jenkinsfile-online	5 days ago
 LICENSE	Initial commit	8 days ago
 README.md	Initial commit	8 days ago
 pom.xml	update test report	5 days ago

A red arrow points to the 'Jenkinsfile-online' entry in the list.

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

Branch: master [devops-java-sample](#) / Jenkinsfile-online

[Find file](#) [Copy path](#)

souleensen modify APP\_NAME 7eb3aa7 37 minutes ago

2 contributors

120 lines (111 sloc) | 4.39 KB

Raw Blame History

```

1 pipeline {
2   agent {
3     node {
4       label 'maven'
5     }
6   }
7
8   parameters {
9     string(name:'TAG_NAME',defaultValue: '',description='')
10  }
11
12  environment {
13    DOCKER_CREDENTIAL_ID = 'dockerhub-id'
14    GITHUB_CREDENTIAL_ID = 'github-id'
15    KUBECONFIG_CREDENTIAL_ID = 'demo-kubeconfig'
16    REGISTRY = 'docker.io'
17    DOCKERHUB_NAMESPACE = 'docker_username'
18    GITHUB_ACCOUNT = 'kubesphere'
19    APP_NAME = 'devops-java-sample'
20    SONAR_CREDENTIAL_ID= 'sonar-token'
21  }
22
23  stages {
24    stage ('checkout scm') {
25      steps {
26        checkout(scm)
27      }
28    }
29  }

```

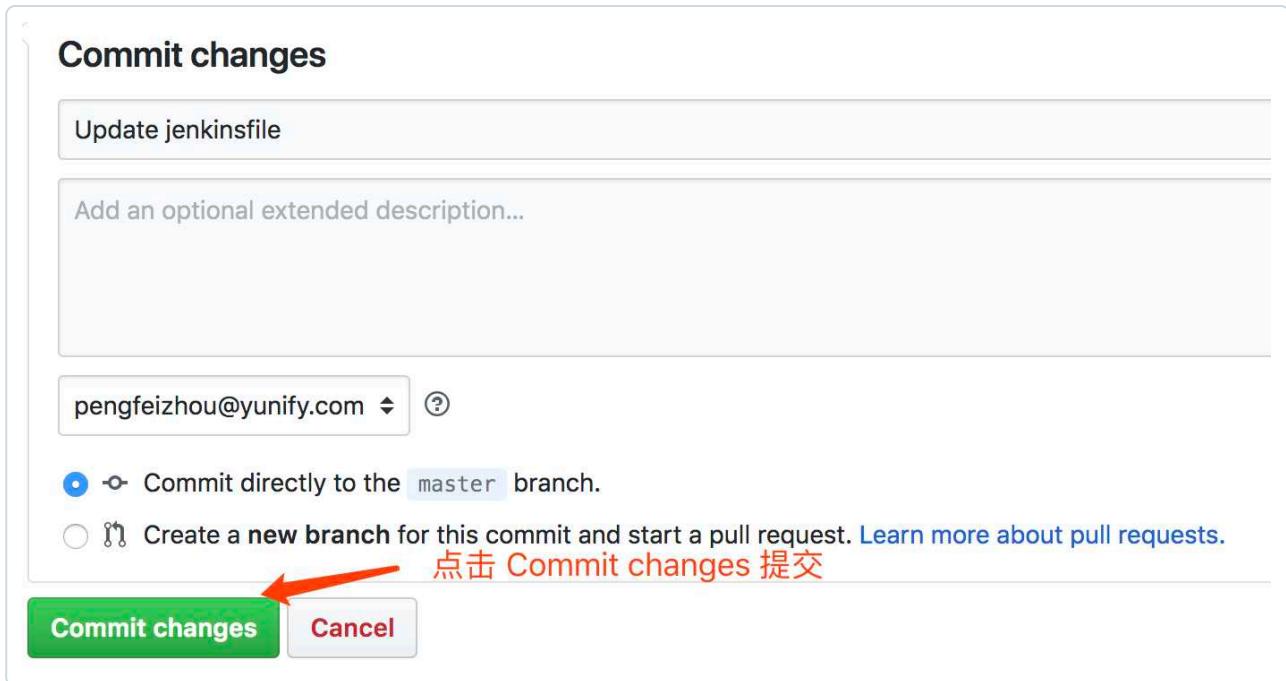


修改项	值	含义
DOCKER_CREDENTIAL_ID	dockerhub-id	填写创建凭证步骤中的 DockerHub 凭证 ID, 用于登录您的 DockerHub
GITHUB_CREDENTIAL_ID	github-id	填写创建凭证步骤中的 GitHub 凭证 ID, 用于推送 tag 到 GitHub 仓库
KUBECONFIG_CREDENTIAL_ID	demo-kubeconfig	kubeconfig 凭证 ID, 用于访问接入正在运行的 Kubernetes 集群
REGISTRY	docker.io	默认为 docker.io 域名, 用于镜像的推送
DOCKERHUB_NAMESPACE	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-	替换为您的 GitHub 账号名, 例如 <a href="https://github.com/kubesphere/">https://github.com/kubesphere/</a> 则填写

修改项	值	含义
	account	kubesphere (它也可以是账户下的 Organization 名称)
APP_NAME	devops- java- sample	应用名称
SONAR_CREDENTIAL_ID	sonar- token	填写创建凭证步骤中的 SonarQube token 凭证 ID，用于代码质量检测

注： Jenkinsfile 中 mvn 命令的参数 `-o`，表示开启离线模式。本示例为适应某些环境下网络的干扰，以及避免在下载依赖时耗时太长，已事先完成相关依赖的下载， 默认开启离线模式。

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 流水线会根据示例项目的 [yaml 模板文件](#)，最终将示例分别部署到 Dev 和 Production 这两个项目 (Namespace) 环境中，即 `kubesphere-sample-dev` 和 `kubesphere-sample-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：创建第一个项目

提示：项目管理员 `project-admin` 账号已在 [多租户管理快速入门](#) 中创建。

1、使用项目管理员 `project-admin` 账号登录 KubeSphere，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-sample-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

2、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。

## 第二步：邀请成员

第一个项目创建完后，还需要项目管理员 `project-admin` 邀请当前的项目普通用户 `project-regular` 进入 `kubesphere-sample-dev` 项目，进入「项目设置」→「项目成员」，点击「邀请成员」选择邀请 `project-regular` 并授予 `operator` 角色，若对此有疑问可参考 [多租户管理快速入门 – 邀请成员](#)。

## 第三步：创建第二个项目

同上，参考以上两步创建一个名称为 `kubesphere-sample-prod` 的项目作为生产环境，并邀请普通用户 `project-regular` 进入 `kubesphere-sample-prod` 项目并授予 `operator` 角色。

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

The screenshot shows the KubeSphere project management interface. At the top, there is a search bar with placeholder text "请输入名称进行查找" and a "创建" (Create) button. Below the search bar is a table with columns: 名称 (Name), 容器组数量 (Number of Container Groups), CPU 使用量 (CPU Usage), 内存使用量 (Memory Usage), 管理员 (Administrator), and 创建时间 (Creation Time). Two projects are listed:

名称	容器组数量	CPU 使用量	内存使用量	管理员	创建时间
kubesphere-sample-prod(生产环境)	0	-	-	admin	2019-04-12 15:49:27
kubesphere-sample-dev(开发环境)	0	-	-	admin	2019-04-12 15:43:38

## 创建流水线

### 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。

The screenshot shows the KubeSphere DevOps engineering pipeline creation interface. On the left, there is a sidebar with a "DevOps 工程 devops-demo" section containing a "流水线" (Pipeline) button, and a "工程管理" (Engineering Management) section with links for "基本信息" (Basic Information), "凭证" (Certificate), "成员角色" (Member Roles), and "工程成员" (Engineering Members). The main content area has a title "流水线" 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 "官网文档" (Official Document) and a "参考文档" (Reference Document) link. At the bottom, there is a "创建" (Create) button.

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

Pipeline 的名称，同一个项目内 Pipeline 不能重名

项目

project-RyonypM4PX2q

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](#)

请输入 github access token

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 for creating a new personal access token. The left sidebar has three options: OAuth Apps, GitHub Apps, and Personal access tokens, with Personal access tokens selected. The main area is titled "New personal access token". It includes a "Token description" field containing "DevOps demo", a "What's this token for?" section, and a "Select scopes" section with a note about scopes defining access for personal tokens. A red arrow points to the "Select scopes" link.

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-java-sample](#)，点击 [选择此仓库](#)，完成后点击 [下一步](#)。

The screenshot shows the KubeSphere pipeline creation interface. The top bar says "创建流水线". Below it are tabs for "基本信息" and "高级设置". A sub-section titled "选择代码仓库" is shown. It lists several GitHub repositories: "mos", "dpu", "kubesphere", and "souiseen". On the right, a list of repositories is shown: "devops-java-sample" (selected), "docs.kubesphere.io", "flask", and "industrial". A red arrow points to the "选择此仓库" button next to "devops-java-sample". At the bottom right is a "保存" button.

## 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 **保留分支的天数** 和 **保留分支的最大个数** 默认为 -1。



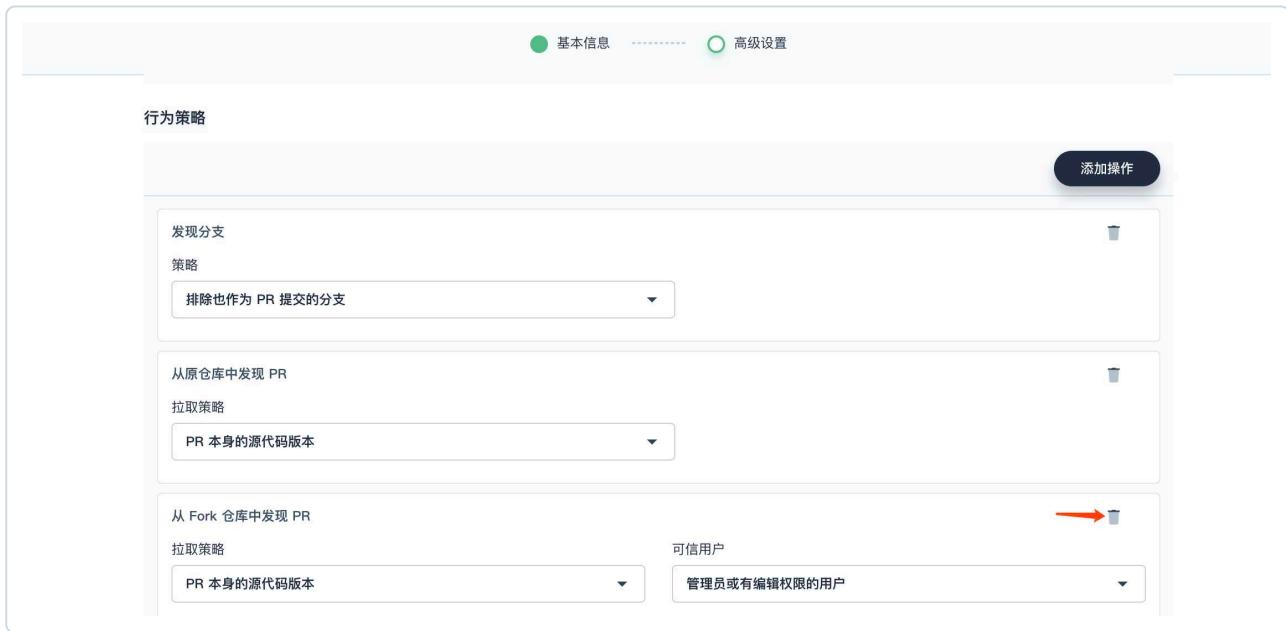
### 说明：

分支设置的**保留分支的天数**和**保持分支的最大个数**两个选项可以同时对分支进行作用，只要某个分支的保留天数和个数不满足任何一个设置的条件，则将丢弃该分支。假设设置的保留天数和个数为2和3，则分支的保留天数一旦超过2或者保留个数超过3，则将丢弃该分支。默认两个值为-1，表示不自动删除分支。

丢弃旧的分支将确定何时应丢弃项目的分支记录。分支记录包括控制台输出，存档工件以及与特定分支相关的其他元数据。保持较少的分支可以节省 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分支：

- 保留分支的天数：如果分支达到一定的天数，则丢弃分支。
- 保留分支的个数：如果已经存在一定数量的分支，则丢弃最旧的分支。

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。



## 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`，请将其修改为 `Jenkinsfile-online`。

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。

#### 脚本路径

路径

Jenkinsfile-online

指定 Jenkinsfile 在源代码仓库的位置

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**。

#### 扫描 Repo Trigger

启用正则表达式，将忽略与提供的正则表达式不匹配的名称（包括分支与PR等）

如果没有扫描触发，则定期扫描 ?

扫描时间间隔

5 minutes

#### Git 克隆参数

克隆深度

1

流水线 clone 超时时间 (单位: 分钟)

20

是否开启浅克隆

#### webhook 推送

推送消息到 ?

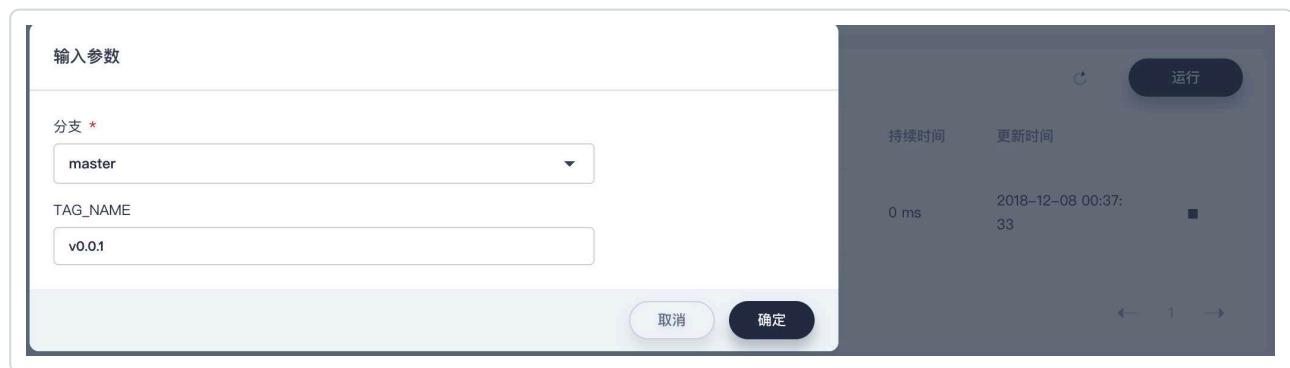
http://139.198.13.21:30880/webhook/github-webhook/

## 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-online (默认是根目录下的 Jenkinsfile)。
- 2、由于仓库的 Jenkinsfile-online 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，流水线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。

The screenshot shows the KubeSphere interface for managing Jenkins pipelines. On the left, there's a sidebar with '平台管理', '工作台', and '应用模板'. The main area is titled 'Jenkinsfile-in-SCM' under 'project-MGppvJ8D2Y93 / pipelines / Jenkinsfile-in-SCM / activity'. It displays a summary card with a green '健康' (Healthy) status, 1 branch, and a table of recent activities. One activity is shown in detail: '运行中' (Running), ID 2, branch 'master', triggered by user 'admin', duration 0 ms, last updated 2018-11-23 12:06:58.

## 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-online` 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。

This screenshot shows the pipeline execution interface. At the top, there are tabs for '运行状态' (Running Status), '提交' (Submit), and '制品' (Products). Below, the pipeline is visualized as a sequence of stages connected by arrows. Stage 1: 'build & push snapshot image' (任务 4 / 4, 成功). Stage 2: 'push latest image' (任务 2 / 2, 成功). Stage 3: 'deploy to dev' (显示 '等待输入 deploy to dev?'，有 '继续' 和 '终止' 按钮)。The 'deploy to dev' stage is currently paused, requiring manual intervention to proceed.

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：

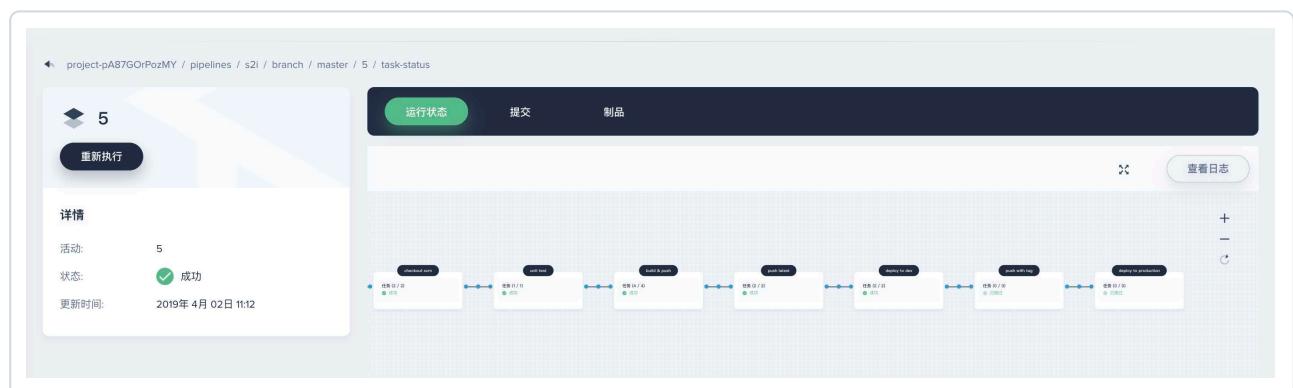
...

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

...

## 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-online](#) 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。

流水线运行日志

阶段 用时 43.82 s

- checkout scm
- unit test
- sonarqube analysis
- build & push**
- push latest
- deploy to dev
- push with tag

Shell Script

```
++ pwd
+ mvn -o -Dmaven.test.skip=true -gs /home/jenkins/workspace/QQ6RWYpADYJ_devops-online_master/configuration/settings.xml clean pac
[INFO] Scanning for projects...
[INFO] -----
[INFO] < io.kubesphere.devops:devops-sample >-----
[INFO] Building devops-sample :: HelloWorld Demo 0.0.1-SNAPSHOT
[INFO] -----
[INFO]   [ jar ] -----
[INFO]
[INFO] --- maven-clean-plugin:2.6.1:clean (default-clean) @ devops-sample ---
[INFO] Deleting /home/jenkins/workspace/QQ6RWYpADYJ_devops-online_master/target
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.2:prepare-agent (agent-for-ut) @ devops-sample ---
[INFO] argLine set to -javaagent:/home/jenkins/workspace/QQ6RWYpADYJ_devops-online_master/artifacts/m2/org/jacoco/org.jacoco.agent
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ devops-sample ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/jenkins/workspace/QQ6RWYpADYJ_devops-online_master/src/main/resources
[INFO] skip non existing resourceDirectory /home/jenkins/workspace/QQ6RWYpADYJ_devops-online_master/src/main/resources
```

刷新 下载日志

关闭

## 验证运行结果

1、若流水线执行成功，点击该流水线下的 **代码质量**，即可看到通过 sonarQube 的代码质量检测结果，如下图(仅供参考)。

流水线列表 / 代码质量

**devops-online**

编辑信息 更多操作

健康 状态: 成功 更新时间: 2019年4月19日 10:57

**代码质量** 活动 分支

检测结果

行数	Bug	代码漏洞	容易出错	覆盖率
118	0个	0个	0行	40.0%

sonarqube

2、流水线最终 build 的 Docker 镜像也将被成功地 push 到 DockerHub 中，我们在 Jenkinsfile-online 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(master-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。Hello World 示例页面最终将以 deployment 和 service 分别部署到 KubeSphere 的 `kubesphere-sample-dev` 和 `kubesphere-sample-prod` 项目环境中。

环境	访问地址	所在项目 (Namespace)	部署 (Deployment)	服务 (Service)
Dev	<code>http://{\$Virtual IP}:{\$8080}</code> 或者 <code>http://{\$内网/公网 IP}:{\$30861}</code>	<code>kubesphere-sample-dev</code>	<code>ks-sample-dev</code>	<code>ks-sample-dev</code>
Production	<code>http://{\$Virtual IP}:{\$8080}</code> 或者 <code>http://{\$内网/公网 IP}:{\$30961}</code>	<code>kubesphere-sample-prod</code>	<code>ks-sample</code>	<code>ks-sample</code>

3、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-sample-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-sample` 已创建成功。正常情况下，部署的状态应该显示 **运行中**。

The screenshot shows the KubeSphere interface for the `kubesphere-sample-prod` project. On the left, there's a sidebar with options like Overview, Applications, Workload (selected), Deployments, StatefulSets, DaemonSets, Jobs, and CronJobs. In the main area, under the 'Deployments' tab, there's a search bar and filters for Name, Status, Application, and Last Updated. A deployment named `ks-sample` is listed with a status of `Running (2/2)`. The status is highlighted with a red circle.

4、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，可以看到该服务的 Virtual IP 为 `10.233.42.3`，对外暴露的节点端口 (NodePort) 是 `30961`。

## 查看服务

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。

名称	IP地址	端口	应用	创建时间
ks-sample	Virtual IP: 10.233.42.3	端口: 8080:8080/TCP 节点端口: 8080:30961/TCP		2019-03-28 10:25:38

5、查看推送到您个人的 DockerHub 中的镜像，可以看到 `devops-java-sample` 就是 APP\_NAME 的值，而 tag 也是在 jenkinsfile-online 中定义的 tag。

Showing 1-25 of 45 Tags

Tag	Size	Last Updated
v0.0.1	8 MB	2 hours ago
latest	8 MB	2 hours ago
SNAPSHOT-master-2	8 MB	2 hours ago

6、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` 生成的。

## 访问示例服务

若在内网环境访问部署的 HelloWorld 示例服务，可通过 SSH 登录集群节点，或使用集群管理员登陆 KubeSphere 在 web kubectl 中输入以下命令验证访问，其中 Virtual IP 和节点端口 (NodePort) 可通过对应项目下的服务中查看：

### 验证 Dev 环境的示例服务

```
# curl {$Virtual IP}:{$Port} 或者 curl {$内网 IP}:{$NodePort}
curl 10.233.40.5:8080
Hello,World!
```

Virtual IP 在

### 验证 Production 环境的示例服务

```
# curl {$Virtual IP}:{$Port} 或者 curl {$内网 IP}:{$NodePort}
curl 10.233.42.3:8080
Hello,World!
```

若两个服务都能访问成功，则说明流水线运行结果也是符合预期的。

**提示：**若需要在外网访问该服务，可能需要绑定公网 EIP 并配置端口转发和防火墙规则。在端口转发规则中将内网端口比如 30861 转发到源端口 30861，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考[云平台配置端口转发和防火墙](#)。

至此，基于 GitHub 和 DockerHub 的一个 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](#)。

# 图形化构建流水线 (Jenkinsfile out of SCM)

上一篇文档示例十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 [示例十 – Jenkinsfile in SCM](#)，通过可视化构建流水线最终将一个 HelloWorld 示例服务部署到 KubeSphere 集群中的开发环境且能够允许用户访问，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。若熟悉了示例十的流程后，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-java-sample](#) 为例。

## 前提条件

- 已有 [DockerHub](#) 的账号；
- 已创建了企业空间和 DevOps 工程并且创建了普通用户 `project-regular` 的账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请普通用户 `project-regular` 加入 DevOps 工程并授予 `maintainer` 角色，若还未邀请请参考 [多租户管理快速入门 – 邀请成员](#)。
- 邮件发送需安装前在 Installer 中配置，请参考 [集群组件配置释义](#) (下一版本将支持安装后在 UI 统一配置邮件服务器)。

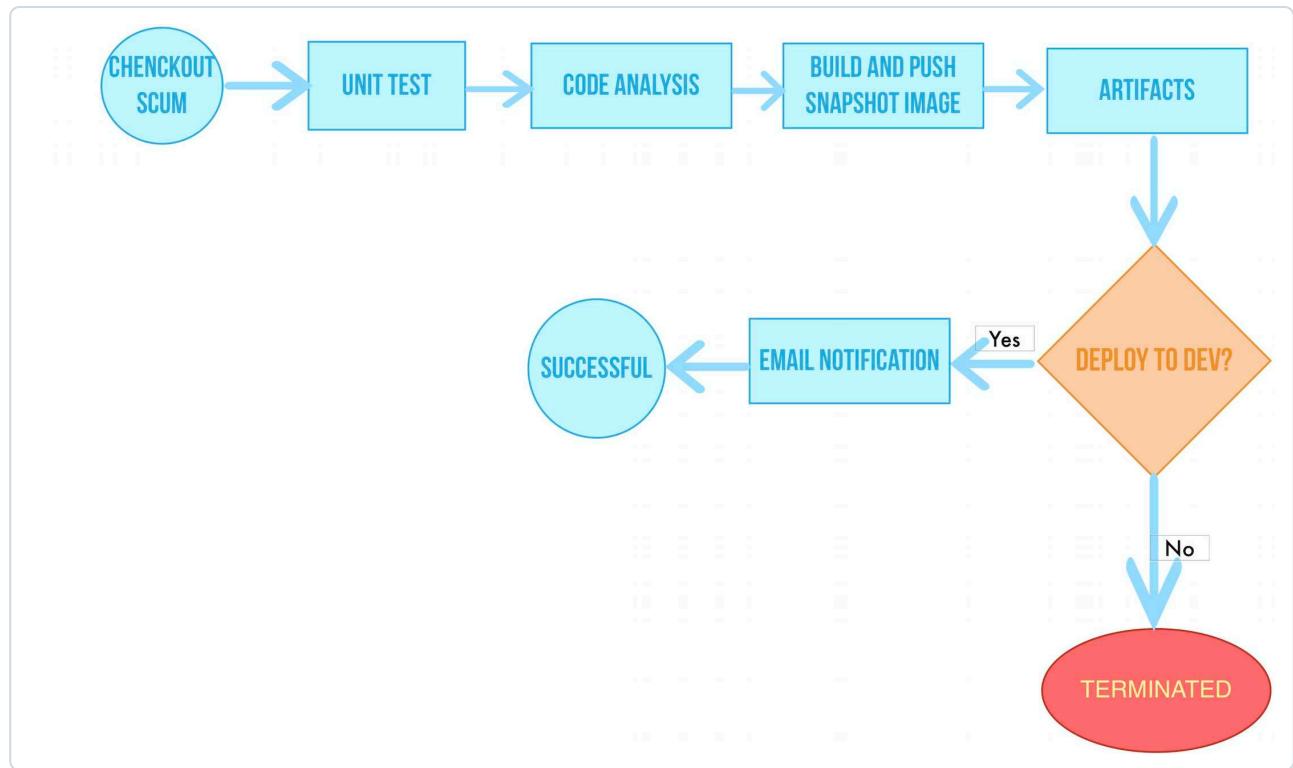
## 预估时间

约 30 分钟。

# 操作示例

## 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流水线的工作流：



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. Unit test: 单元测试，如果测试通过了才继续下面的任务；
- 阶段三. Code Analysis: 配置 SonarQube 进行静态代码质量检查与分析；
- 阶段四. Build and Push: 构建镜像，并将 tag 为 `SNAPSHOT-$BUILD_NUMBER` 推送至 DockerHub (其中 `$BUILD_NUMBER` 为 pipeline 活动列表的运行序号)；
- 阶段五. Artifacts: 制作制品 (jar 包) 并保存；
- 阶段六. Deploy to DEV: 将项目部署到 Dev 环境，此阶段需要预先审核，若部署成功后则发送邮件。

## 创建项目

CI/CD 流水线会根据文档网站的 [yaml 模板文件](#)，最终将该示例 Web 部署到开发环境 [kubesphere-sample-dev](#)，它对应的是 KubeSphere 中的一个项目，该项目需要预先创建，若还未创建请参考 [示例十一 - 创建第一个项目](#) 使用 项目管理员 project-admin 账号创建 [kubesphere-sample-dev](#) 项目，并邀请项目普通用户 [project-regular](#) 进入该项目授予 [operator](#) 角色。

The screenshot shows the KubeSphere interface. On the left, there's a sidebar with various project management options like Workstation, Application Dashboard, Project Overview, Applications, Workload, Storage, Network & Services, Monitoring, Configuration Center, and Project Settings. The main area is titled 'KUBESPHERE' and shows the 'kubesphere-docs-dev' project details: managed by 'project-admin' in the 'demo-workspace' space, created on '2019-05-14 11:19:47'. Below this, the 'Members' section lists:

成员名称	状态	角色	上次登录时间
project-regular project-regular@ks.io	活跃	operator	2019-05-14 10:59:28
project-admin project-admin@ks.io	活跃	admin	2019-05-14 11:18:59

A red circle highlights the 'operator' role for the 'project-regular' member. To the right of the table, a numbered list provides steps for this process:

- 1 使用项目管理员登录
- 2 创建 kubesphere-docs-dev 项目
- 3 选择「项目成员」
- 4 点击「邀请成员」
- 5 邀请项目普通用户 project-regular
- 6 授予 operator 角色

## 创建凭证

本示例创建流水线时需要访问 DockerHub、Kubernetes (创建 KubeConfig 用于接入正在运行的 Kubernetes 集群) 和 SonarQube 共 3 个凭证 (Credentials)。

- 1、使用项目普通用户登录 KubeSphere，参考 [创建凭证](#) 创建 DockerHub 和 Kubernetes 的凭证，凭证 ID 分别为 [dockerhub-id](#) 和 [demo-kubeconfig](#)。
- 2、然后参考 [访问 SonarQube](#) 创建 Token，创建一个 Java 的 Token 并复制。
- 3、最后在 KubeSphere 中进入 devops-demo 的 DevOps 工程中，与上面步骤类似，在 [凭证](#) 下点击创建，创建一个类型为秘密文本的凭证，凭证 ID 命名为 [sonar-token](#)，密钥为上一步复制的 token 信息，完成后点击「确定」。

至此，3 个凭证已经创建完毕，将在流水线中使用它们。

名称	类型	描述信息	创建时间
sonar-token	秘密文本		2019-05-12 22:49:51
dockerhub-id	账户凭证	DockerHub	2019-05-12 22:50:15
demo-kubeconfig	kubeconfig		2019-05-12 22:51:04

## 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、在 DevOps 工程中，选择左侧 流水线，然后点击 创建。

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，例如 graphical-pipeline
- 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

基本信息

请输入流水线的基本信息

名称 *	graphical-pipeline	项目	project-23m45l7ONmEr
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	This is a demo.		

## 第二步：高级设置

1、点击 **添加参数**, 如下添加 3 个字符串参数, 将在流水线的 docker 命令中使用该参数, 完成后点击确定。

参数类型	名称	默认值	描述信息
字符串参数 (string)	REGISTRY	仓库地址, 本示例使用 docker.io	Image Registry
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	应用名称, 填写 devops-sample	Application Name

字符串参数 (String)	<span style="float: right;">删除</span>
名称	默认值
<input type="text" value="REGISTRY"/>	<input type="text" value="docker.io"/>
指定字段的默认值，允许用户保存键入实际值。	
描述信息	
<input type="text" value="仓库地址"/>	
注释信息	
字符串参数 (String)	<span style="float: right;">删除</span>
名称	默认值
<input type="text" value="DOCKERHUB_NAMESPACE"/>	<input type="text" value="pengfeizhou"/>
指定字段的默认值，允许用户保存键入实际值。	
描述信息	
<input type="text" value="DockerHub 账号"/>	
注释信息	
字符串参数 (String)	<span style="float: right;">删除</span>
名称	默认值
<input type="text" value="APP_NAME"/>	<input type="text" value="devops-sample"/>

## 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：拉取源代码 (Checkout SCM)

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](#)。

1、如下，此处代理的类型选择 `node`，label 填写 `maven`。

**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](#)。

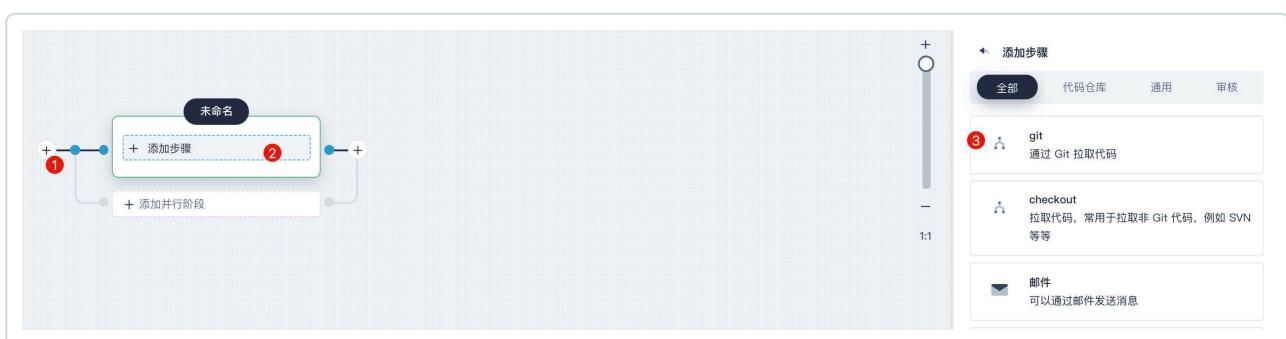
2、在图形化构建流水线的界面，点击左侧结构编辑区域的“+”号，增加一个阶段（Stage），点击界面中的添加步骤，在右侧输入框将其命名为 **Checkout SCM**。



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：

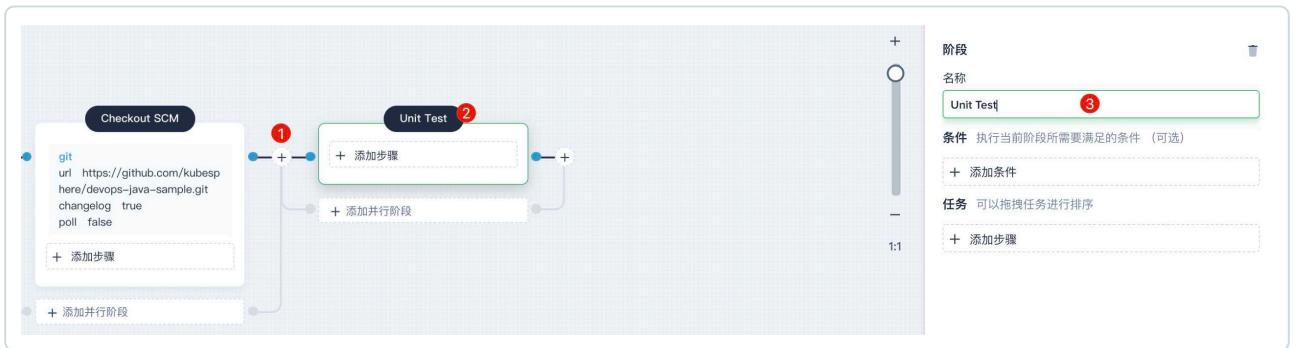
- Url: 填写 GitHub 示例仓库的 URL <https://github.com/kubesphere/devops-java-sample.git>
- 凭证 ID: 无需填写 (若是私有仓库, 如 Gitlab 则需预先创建并填写其凭证 ID)
- 分支: 此处无需填写分支名, 不填则默认为 master 分支

完成后点击「确定」保存，可以看到构建的流水线的第一个阶段。

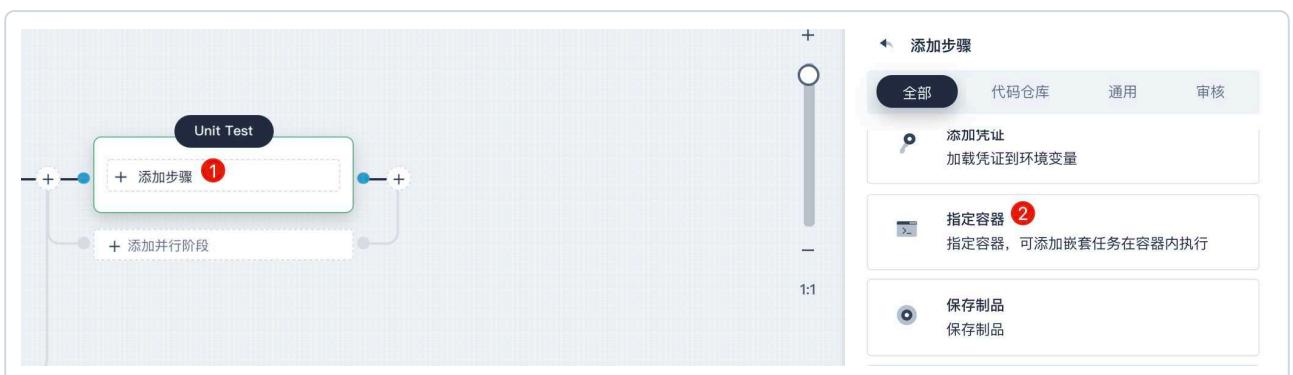


## 阶段二：单元测试 (Unit Test)

1、在 **Checkout SCM** 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**。

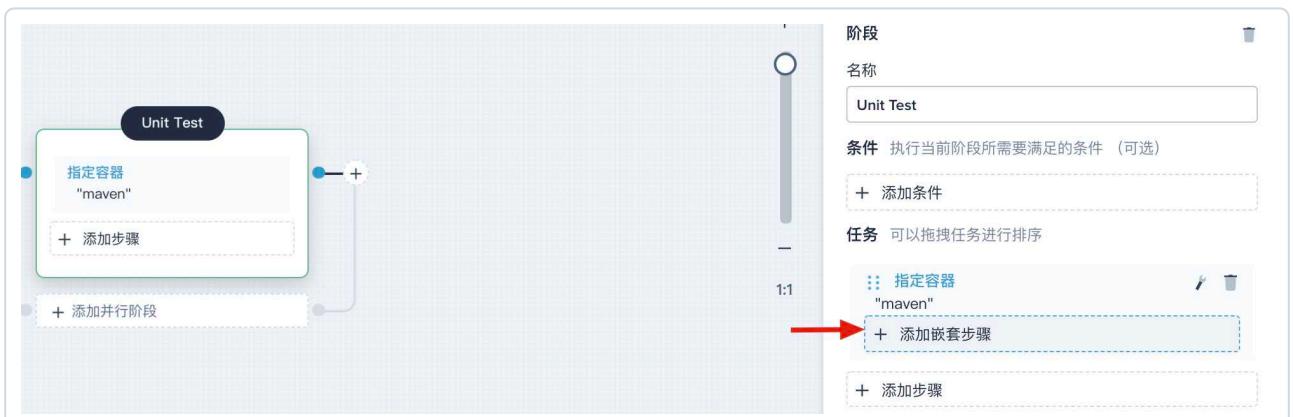


2、点击 **添加步骤** 选择 **指定容器**，命名为 **maven**，完成后点击 **确定**；



3、在右侧 **maven** 容器中点击 **添加嵌套步骤**。然后选择 **shell**，在弹窗中输入以下命令，然后点击保存：

```
mvn clean -o -gs `pwd`/configuration/settings.xml test
```



### 阶段三：代码质量分析 (Code Analysis)

- 1、同上，在 **Unit Test** 阶段右侧点击“+”继续增加一个阶段用于配置 SonarQube 在容器中执行静态代码质量分析，名称为 **Code Analysis**。
- 2、点击 **添加步骤** 选择 **指定容器**，命名为 **maven**，完成后点击 **确定**。
- 3、右侧点击 **添加嵌套步骤**，选择 **添加凭证**，在弹窗中选择之前创建的凭证 ID **sonar-token**，文本变量填写 **SONAR\_TOKEN**，点击「确定」。



- 4、在右侧的添加凭证中，点击 **添加嵌套步骤**，然后选择 **SonarQube 配置**，默认名称为 sonar，点击确定。



5、在右侧的 SonarQube 配置 中点击 添加嵌套步骤，右侧选择 Shell，在弹窗中如下输入 SonarQube 分支和认证配置的命令，点击确定保存信息。

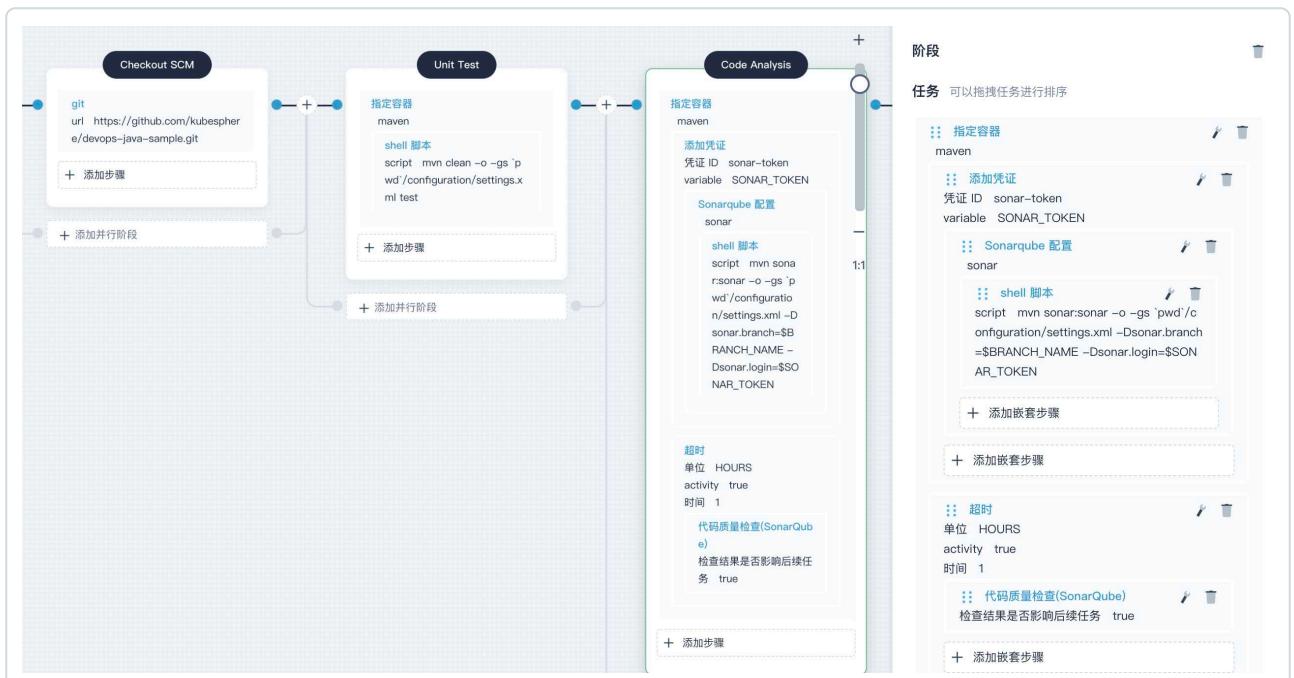
```
mvn sonar:sonar -o -gs `pwd`/configuration/settings.xml -Dsonar.branch=$BRANCH_NAME -Dsonar.login=$SONAR_TOKEN
```



6、右侧点击第三个 添加嵌套步骤，选择 超时，时间输入 1，单位选择 小时，点击「确定」。



7、在超时的步骤中，点击 **添加嵌套步骤**，选择 **代码质量检查 (SonarQube)**，弹窗中保留默认的 **检查通过后开始后续任务**，点击「确定」。



## 阶段四：构建并推送镜像 (Build and Push)

1、在 **Code Analysis** 阶段右侧点击 “+” 继续增加一个阶段用于构建并推送镜像至 DockerHub，名称为 **Build and Push**。

2、点击 **添加步骤** 选择 **指定容器**，命名为 **maven**，完成后点击 **确定**。

3、在右侧点击 **添加嵌套步骤**，右侧选择 **Shell**，在弹窗中如下输入以下命令：

```
mvn -o -Dmaven.test.skip=true -gs `pwd`/configuration/settings.xml clean package
```

4、右侧继续点击 **添加嵌套步骤**，选择 **Shell**，在弹窗中如下输入以下命令基于仓库中的 **Dockerfile** 构建 Docker 镜像，完成后点击确认保存：

```
docker build -f Dockerfile-online -t $REGISTRY/$DOCKERHUB_NAMESPACE/$APP_NAME:$SNAPSHOT
```

5、点击 **添加嵌套步骤**，右侧选择 **添加凭证**，在弹窗中填写如下信息，完成后点击「确定」保存信息：

**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。**

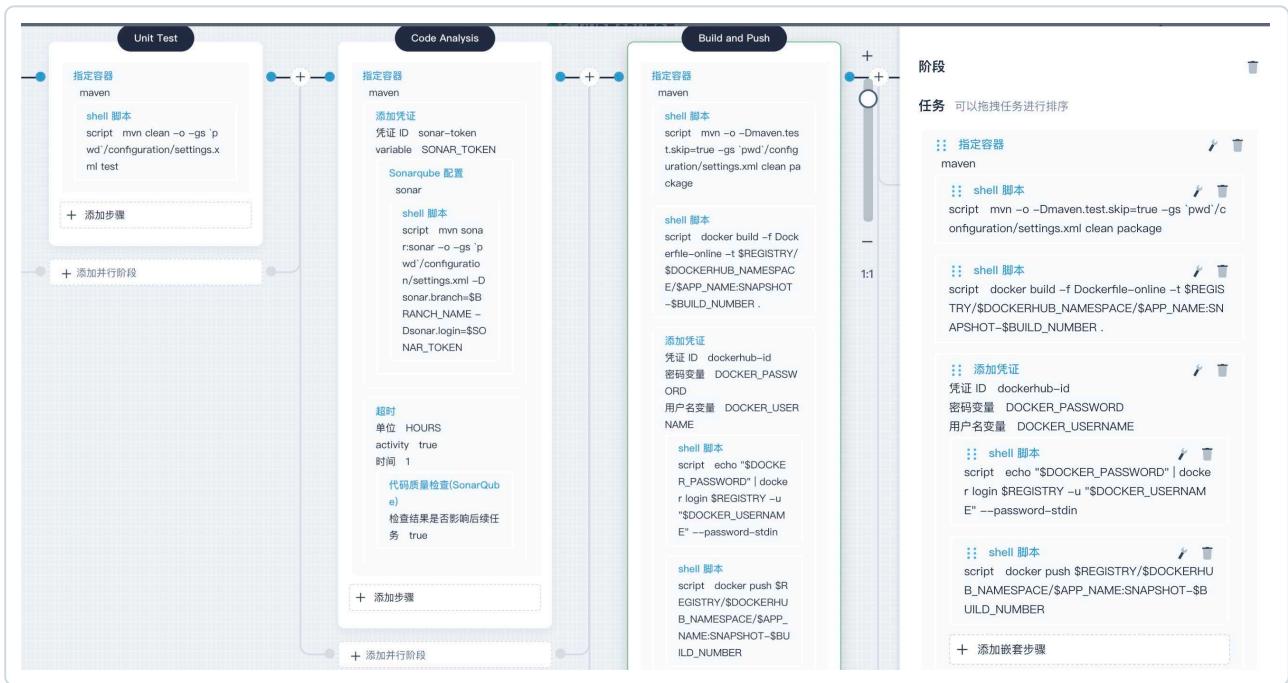
- 凭证 ID：选择之前创建的 DockerHub 凭证，如 **dockerhub-id**
- 密码变量：**DOCKER\_PASSWORD**
- 用户名变量：**DOCKER\_USERNAME**

6、在 **添加凭证** 步骤中点击 **添加嵌套步骤**，右侧选择 **Shell**，在弹窗中如下输入以下命令登录 Docker Hub：

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

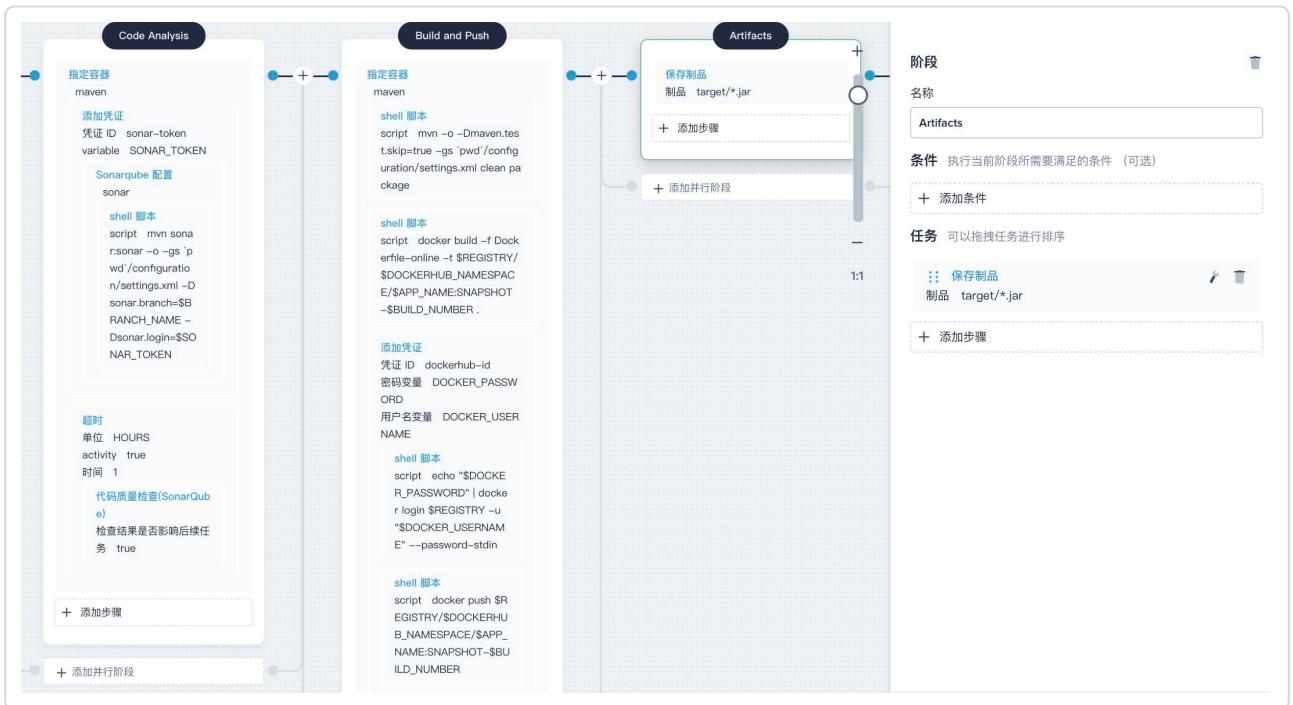
7、同上，继续点击 **添加嵌套步骤** 添加 **Shell** 输入一条命令推送 SNAPSHOT 镜像至 Docker Hub：

```
docker push $REGISTRY/$DOCKERHUB_NAMESPACE/$APP_NAME:SNAPSHOT-$BUILD_NUMBER
```



## 阶段五：保存制品 (Artifacts)

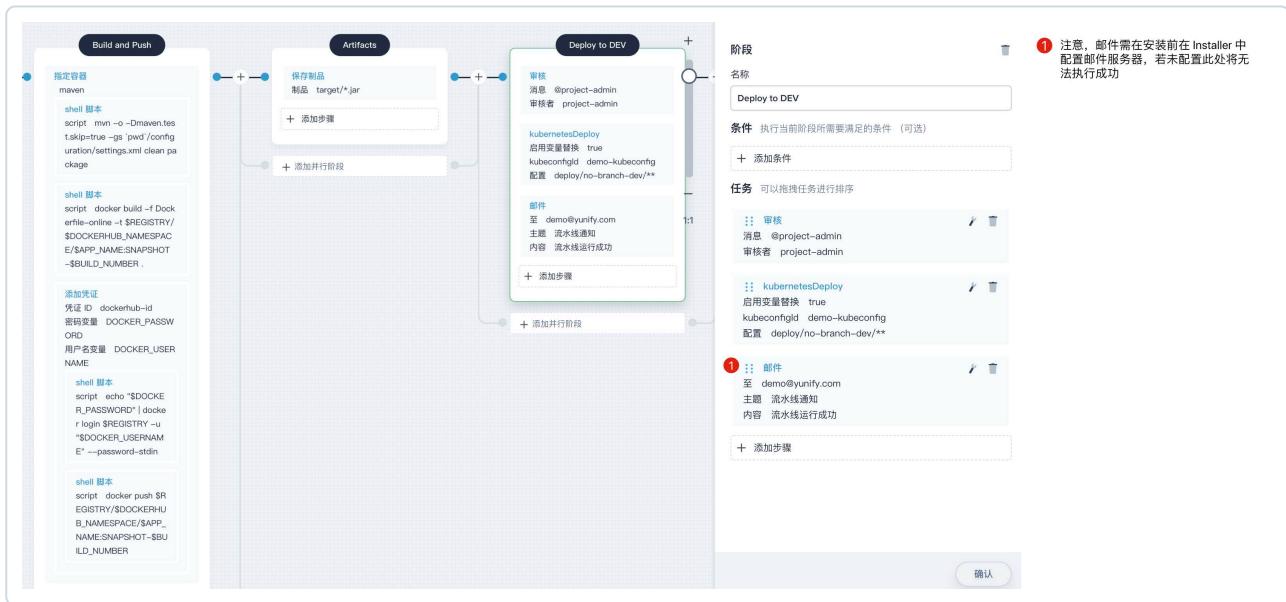
- 1、在 **Build and Push** 阶段右侧点击“+”继续增加一个阶段用于保存制品，本示例用于保存项目的 jar 包，命名为 **Artifacts**。
- 2、点击 **添加步骤**，选择 **保存制品**，在弹窗中输入 **target/\*.jar**，用于捕获构建包含模式匹配的文件 (**target/\*.jar**) 并保存到 Jenkins，点击「确定」。



## 阶段六：部署至 Dev 环境 (Deploy to DEV)

- 1、在 **Artifacts** 阶段右侧点击“+”增加最后一个阶段，命名为 **Deploy to DEV**，用于将容器镜像部署到开发环境即 **kubesphere-sample-dev** 项目中。
- 2、点击 **添加步骤**，选择 **审核**，在弹窗中输入 **@project-admin** 指定项目管理员 project-admin 用户来进行流水线审核，点击「确定」。
- 3、点击 **添加步骤**，选择 **KubernetesDeploy**，在弹窗中参考以下填写，完成后点击「确定」保存信息：
  - Kubeconfig：选择 **demo-kubeconfig**
  - 配置文件路径：输入 **deploy/no-branch-dev/\*\***，这里是本示例的 Kubernetes 资源部署 **yaml 文件** 在代码仓库中的相对路径
- 4、同上再添加一个步骤，用于在这一步部署和流水线执行成功后给用户发送通知邮件。点击 **添加步骤**，选择 **邮件**，自定义收件人、抄送、主题和内容。

**注意，在流水线中发送邮件需要在安装前预先在 Installer 中配置邮件服务器，配置请参考 [集群组件配置释义](#)，若还未配置请跳过第 4 步（下一版本将支持安装后在 UI 统一配置邮件服务器）。**



至此，图形化构建流水线的六个阶段都已经添加成功，点击 **确认 → 保存**，可视化构建的流水线创建完成，同时也会生成 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**，输入参数弹窗中可看到之前定义的三个字符串参数，此处暂无需修改，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** 可查看其运行活动的具体情况。

**说明：**流水线刚启动时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

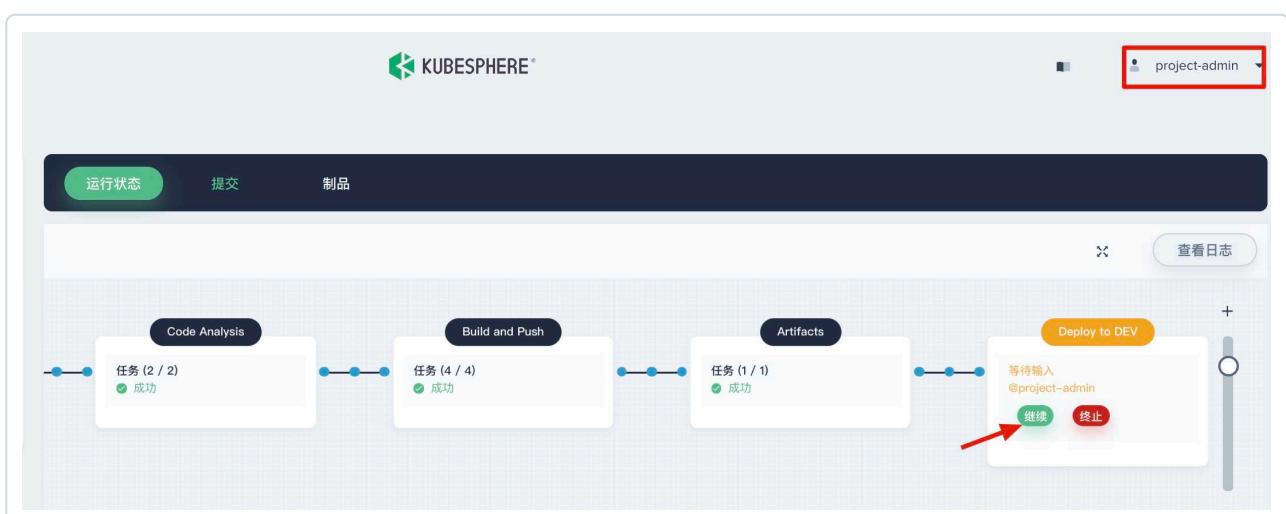
始运行时，Jenkins slave 启动并开始解析和执行流水线自动生成的 Jenkinsfile，待初始化完成即可看到图形化流水线运行的页面。

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 graphical-pipeline。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。

The screenshot shows the KubeSphere pipeline log view for the 'Build and Push' stage. The log details the execution of four shell scripts. The first three scripts completed successfully in 7.20s, 22.60s, and 4.85s respectively. The fourth script completed successfully in 21.91s. The log output includes Docker push commands and layer preparation logs. A red arrow points to the '查看日志' (View Log) button in the top right corner of the log view. The left sidebar lists other stages: Checkout SCM, Unit Test, Code Analysis, Artifacts, and Deploy to DEV.

## 查看代码质量

在流水线中点击 [代码质量](#)，查看代码质量的检测检测结果，该数据由集群内置的 SonarQube 提供，示例代码较为简单因此未显示 Bug 或代码漏洞的情况。点击右侧的 SonarQube 图标即可在浏览器访问 SonarQube，访问 SonarQube 可参考 [访问内置 SonarQube](#)。

**提示：**若需要在外网访问 SonarQube，可能需要绑定公网 EIP 并配置端口转发和防火墙规则。在端口转发规则中将 内网端口 例如 31359 转发到源端口 31359，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。



## 在 SonarQube 查看测试质量报告

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

devops-sample :: HelloWorld De... master May 14, 2019, 4:18 PM Version 0.0.1-SNAPSHOT

Overview Issues Security Reports Measures Code Activity Administration

**Quality Gate** Passed

Bugs 0 Vulnerabilities 0 New code: since previous version started 30 minutes ago

Bugs 0 Vulnerabilities 0 New Bugs 0 New Vulnerabilities

Code Smells 0 Debt 0 Code Smells 0 New Debt 0 New Code Smells

Coverage 40.0% Unit Tests 1 Coverage on New Code

Duplications 0.0% Duplicated Blocks 0 Duplications on New Code

About This Project  
Parent pom providing dependency and plugin management for applications built with Maven

No tags

118 Lines of Code XML 97 Java 21

Project Activity

May 14, 2019 0.0.1-SNAPSHOT

May 14, 2019 Project Analyzed

May 14, 2019 Project Analyzed

Show More

Quality Gate (Default) Sonar way

Quality Profiles (Java) Sonar way (XML) Sonar way

## 查看制品

点击 **活动** 然后选择 **制品**，可以查看流水线在运行过程中保存的制品 Jar 包，点击即可下载至本地。

运行状态 提交 制品

名称	大小	下载
target/devops-sample-0.0.1-SNAPSHOT.jar	13.86 MB	

## 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-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-sample-dev` 项目，在左侧的菜单栏点击 **工作负载 → 部署**，可以看到 `ks-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	<code>http://{\$Virtual IP}:{\$8080}</code> 或者 <code>http://{\$内网/公网IP}:{\$30861}</code>	kubesphere-sample-dev	ks-sample-dev	ks-sample-dev

## 查看部署

The screenshot shows the 'Deployments' page in the KubeSphere interface. On the left, there's a sidebar with project navigation and a 'Workload' section. Under 'Workload', 'Deployments' is selected. The main area displays a table with one row for the deployment 'ks-sample-dev'. The table columns include 'Name', 'Status', 'Last Update', and 'Created Time'. The deployment is shown as 'Running'.

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 `30861`。

## 查看服务

The screenshot shows the 'Services' page in the KubeSphere interface. On the left, there's a sidebar with project navigation and a 'Network & Services' section. Under 'Network & Services', 'Services' is selected. The main area displays a table with one row for the service 'ks-sample-dev'. The table columns include 'Name', 'IP Address', 'Port', 'Created Time', and 'Last Update'. The 'IP Address' column shows 'Virtual IP: 10.233.4.154' and the 'Port' column shows '端口: 8080:8080/TCP' and '节点端口: 8080:30861/TCP'. These two entries are highlighted with red boxes.

3、查看推送到 DockerHub 的镜像，可以看到 `devops-sample` 就是 `APP_NAME` 的值，而 Tag 则是 `SNAPSHOT-$BUILD_NUMBER` 的值 (`$BUILD_NUMBER` 对应活动的运行序号)，其中 tag 为 `SNAPSHOT-1` 的镜像，正是 `ks-sample-dev` 这个部署所用到的镜像。

The screenshot shows a container named 'pengfeizhou/devops-sample'. It has 7 tags:

- SNAPSHOT-3 55 MB (Last update: 23 minutes ago)
- SNAPSHOT-2 55 MB (Last update: 33 minutes ago)
- SNAPSHOT-1 55 MB (Last update: 41 minutes ago) - This tag is highlighted with a red box.

4、由于我们在流水线最后一个阶段设置了邮件通知，因此可在邮箱中验证收到的构建通知邮件。

The email subject is '构建成功 ☆'. The message content is:

发件人: KubeSphere <admin@app-center.com.cn>   
时间: 2019年5月14日(星期二) 下午4:16  
收件人:

流水线构建成功!

## 访问示例服务

若在内网环境访问部署的 HelloWorld 示例服务，可通过 SSH 登陆集群节点，或使用集群管理员登陆 KubeSphere 在 web kubectl 中输入以下命令验证访问，其中 Virtual IP 和节点端口 (NodePort) 可通过对应项目下的服务中查看：

```
# curl {$Virtual IP}:{$Port} 或者 curl {$内网 IP}:{$NodePort}
curl 10.233.4.154:8080
Hello,World!
```

**提示：**若需要在外网访问该服务，可能需要绑定公网 EIP 并配置端口转发和防火墙规则。在端口转发规则中将内网端口 30861 转发到源端口 30861，然后在防火墙开放这个源端口，保证外网流量可以通过该端口，外部才能够访问。例如在 QingCloud 云平台进行上述操作，则可以参考 [云平台配置端口转发和防火墙](#)。

至此，图形化构建流水线的示例已经完成了，若创建过程中遇到问题，可参考 [常见问题](#)。

# CI/CD 流水线示例（离线版）

KubeSphere Installer 集成了 Harbor 和 GitLab，内置的 Harbor 和 GitLab 作为可选安装项，需在安装前进行配置开启安装。用户可以根据团队项目的需求来安装，方便对项目的镜像和代码进行管理，本文档适用于离线环境的流水线构建。

## 目的

本示例演示通过内置 GitLab 仓库中的 Jenkinsfile 来创建流水线，流水线共包含 7 个阶段，首先会将 GitLab 中的源码构建成镜像，然后推送到 Harbor 私有仓库，最终将一个输出“Hello,World!”的 Web 例子部署到 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

- 本示例以 GitLab 和 Harbor 为例，请确保已安装 [内置 Harbor](#) 和 [内置 GitLab](#)，已准备了基础镜像 `java:openjdk-8-jre-alpine`；
- 已创建了企业空间和 DevOps 工程并且创建了项目普通用户 `project-regular` 的账号，若还未创建请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入 DevOps 工程并授予 `maintainer` 角色，若还未邀请请参考 [多租户管理快速入门 – 邀请成员](#)。

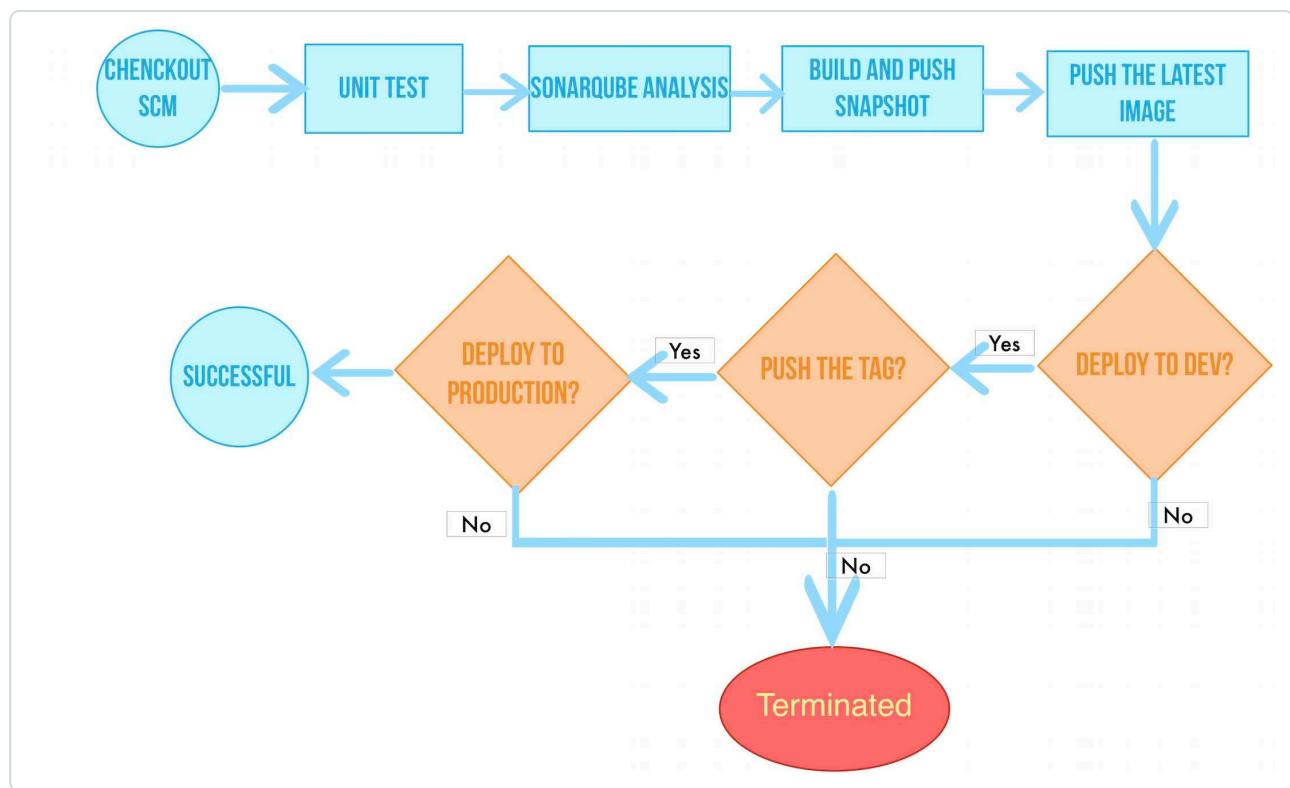
## 预估时间

30–50 分钟

# 操作示例

## 流水线概览

下面的流程图简单说明了流水线完整的工作过程：



### 流程说明：

- 阶段一. Checkout SCM: 拉取 GitLab 仓库代码
- 阶段二. Unit test: 单元测试，如果测试通过了才继续下面的任务
- 阶段三. sonarQube analysis: sonarQube 代码质量检测
- 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 Harbor (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 Harbor。
- 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 GitLab，并推送到 Harbor。
- 阶段八. Deploy to production: 将发布的 tag 部署到 Production 环境。

## 基础设置

### 第一步：修改 CoreDNS 配置

通过 CoreDNS 的 hosts 插件配置 KubeSphere 集群的 DNS 服务，使集群内部可通过 hostname 域名访问外部服务，参考 [修改系统配置 – 如何修改 CoreDNS 配置](#)。

### 第二步：上传基础镜像到 Harbor

参考 [如何上传基础镜像到 Harbor](#) 导入预先准备好的基础镜像 `java:openjdk-8-jre-alpine`。

## 创建凭证

使用项目普通用户 `project-regular` 登录 KubeSphere，进入已创建的 DevOps 工程，开始创建凭证。

- 1、本示例代码仓库中的 Jenkinsfile 需要用到 Harbor、GitLab 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)，参考 [创建凭证](#) 依次创建这三个凭证。
- 2、然后参考 [访问 SonarQube 并创建 Token](#)，创建一个 Java 的 Token 并复制。
- 3、最后在 KubeSphere 中进入 `devops-demo` 的 DevOps 工程中，与上面步骤类似，在 **凭证** 下点击 **创建**，创建一个类型为 `秘密文本` 的凭证，凭证 ID 命名为 `sonar-token`，密钥为上一步复制的 token 信息，完成后点击 **确定**。

### 创建凭证

凭证 ID \*

类型

秘密文本

秘钥

描述信息

取消 确定

至此，4个凭证已经创建完成，下一步需要在示例仓库中的 jenkinsfile 修改对应的四个凭证 ID 为用户自己创建的凭证 ID。

名称	类型	描述信息	创建时间
demo-kubeconfig	kubeconfig		2019-04-09 13:36:13
sonar-token	秘密文本		2019-04-11 18:38:45
harbor-id	账户凭证		2019-04-11 18:41:53
gitlab-id	账户凭证		2019-04-11 18:42:10

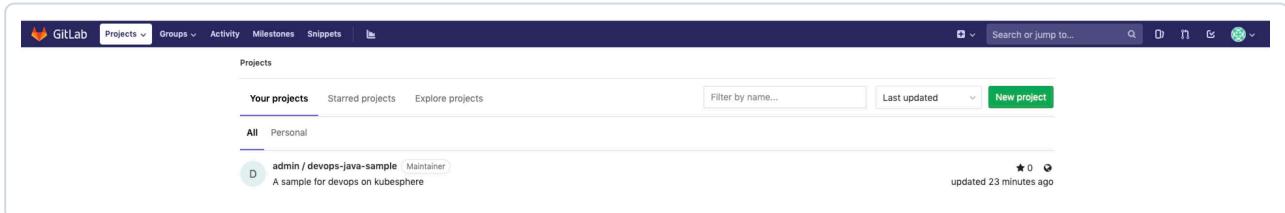
## 修改 Jenkinsfile

### 第一步：进入项目

- 根据前提条件中的要求，现应已按照 [安装 GitLab](#) 要求正确将 GitHub 中的 `devops-java-sample`

导入到GitLab中。

注：若因网络限制，无法从 GitHub 导入，请自行 clone 至其他服务器，然后上传至 GitLab 仓库，仓库名称请保持一致。

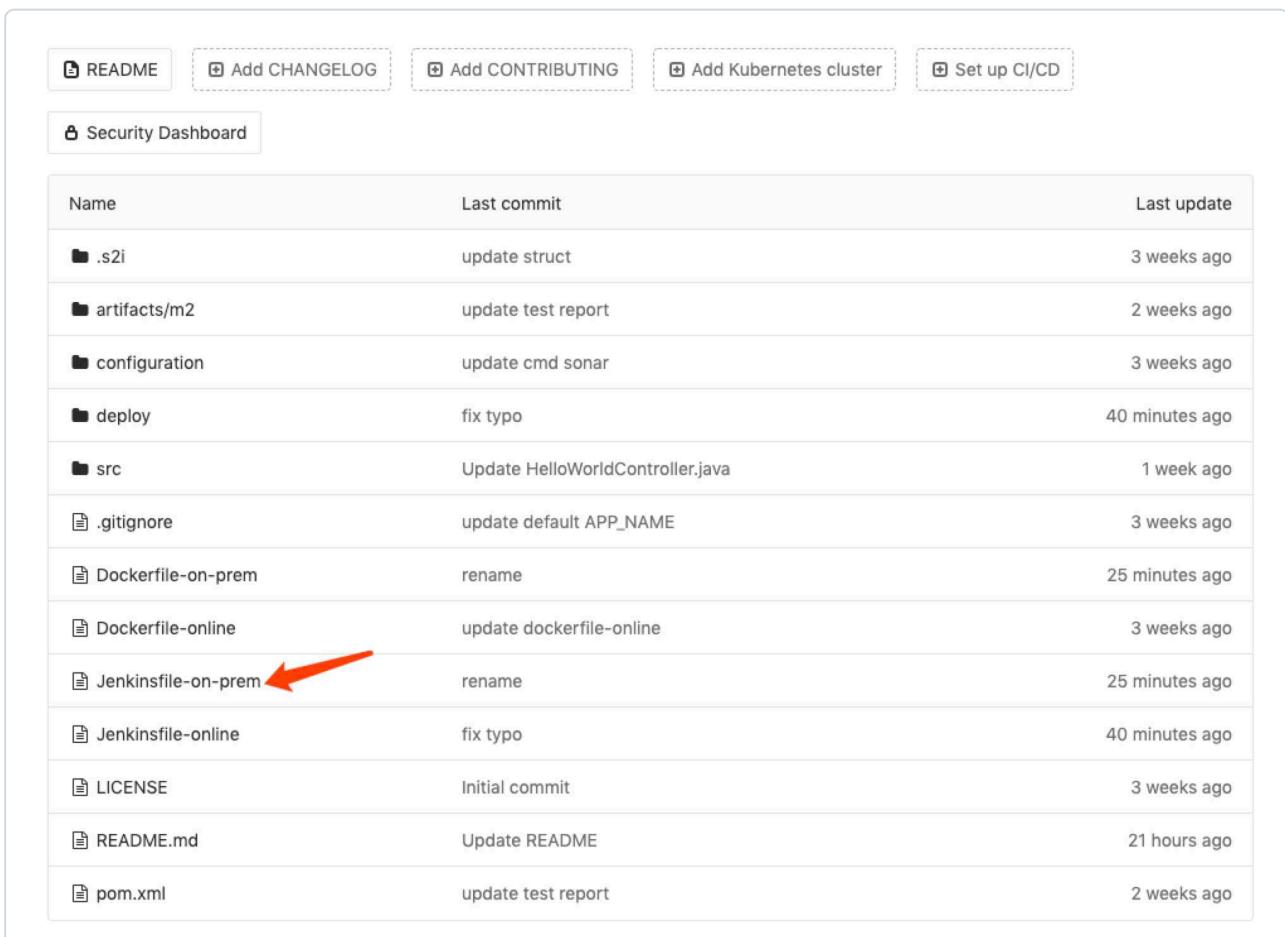


The screenshot shows the GitLab homepage with a single project named 'admin / devops-java-sample' listed under 'Your projects'. The project has a description 'A sample for devops on kubesphere'. There are filters for 'Last updated' and a 'New project' button.

2. 点击项目进入。

## 第二步：修改 Jenkinsfile

1、在根目录进入 Jenkinsfile-on-prem。



The screenshot shows the commit history for the 'Jenkinsfile-on-prem' repository. A red arrow points to the commit 'Jenkinsfile-on-prem'.

Name	Last commit	Last update
.s2i	update struct	3 weeks ago
artifacts/m2	update test report	2 weeks ago
configuration	update cmd sonar	3 weeks ago
deploy	fix typo	40 minutes ago
src	Update HelloWorldController.java	1 week ago
.gitignore	update default APP_NAME	3 weeks ago
Dockerfile-on-prem	rename	25 minutes ago
Dockerfile-online	update dockerfile-online	3 weeks ago
Jenkinsfile-on-prem	rename	25 minutes ago
Jenkinsfile-online	fix typo	40 minutes ago
LICENSE	Initial commit	3 weeks ago
README.md	Update README	21 hours ago
pom.xml	update test report	2 weeks ago

2、在 GitLab UI 点击编辑 [Edit](#)，需要修改如下环境变量 (environment) 的值。



```
1 pipeline {
2   agent {
3     node {
4       label 'maven'
5     }
6   }
7
8   parameters {
9     string(name:'TAG_NAME',defaultValue: '',description='')
10  }
11
12  environment {
13    HARBOR_CREDENTIAL_ID = 'harbor-id'
14    GITLAB_CREDENTIAL_ID = 'gitlab-id'
15    KUBECONFIG_CREDENTIAL_ID = 'demo-kubeconfig'
16    REGISTRY = 'harbor.devops.kubesphere.local:30280'
17    HARBOR_NAMESPACE = 'library'
18    GITLAB_ACCOUNT = 'admin1'
19    APP_NAME = 'devops-java-sample'
20    SONAR_CREDENTIAL_ID= 'sonar-token'
21  }
22
23  stages {
24    stage ('checkout scm') {
25      steps {
26        checkout(scm)
27      }
28    }
29
30    stage ('unit test') {
31      steps {
32        container ('maven') {
33          sh 'mvn clean -o -gs `pwd`/configuration/settings.xml test'
34        }
35      }
36    }
37
38    stage('sonarqube analysis') {
39      steps {
40        container ('maven') {
41          withCredentials([string(credentialsId: "$SONAR_CREDENTIAL_ID", variable: 'SONAR_TOKEN')]) {
42            sh "mvn sonar:sonar"
43          }
44        }
45      }
46    }
47  }
48}
```

修改项	值	含义
HARBOR_CREDENTIAL_ID	harbor-id	填写创建凭证步骤中的 Harbor 凭证 ID，用于登录您的 Harbor 仓库
GITLAB_CREDENTIAL_ID	gitlab-id	填写创建凭证步骤中的 GitLab 凭证 ID，用于推送 tag 到 GitLab 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

修改项	值	含义
REGISTRY	harbor.devops.kubesphere.local:30280	默认为 Harbor 域名，用于镜像的推送
HARBOR_NAMESPACE	library	默认为 Harbor 下的 library 项目，可根据实际情况更改项目名称
GITLAB_ACCOUNT	admin1	GitLab 用户， 默认为 admin1
APP_NAME	devops-docs-sample	应用名称
SONAR_CREDENTIAL_ID	sonar-token	填写创建凭证步骤中的 sonarQube token 凭证 ID，用于代码质量检测

## 创建两个项目

CI/CD 流水线会根据示例项目的 [yaml 模板文件](#)，最终将示例分别部署到 Dev 和 Production 这两个项目 (Namespace) 环境中，项目名为 `kubesphere-sample-dev` 和 `kubesphere-sample-prod`，这两个项目需要预先在控制台依次创建，可参考 [基于 Spring Boot 项目构建流水线 - 创建项目](#) 进行创建。

The screenshot shows the KubeSphere project management interface. At the top, there is a dark header bar with the title '项目'. Below it, a sub-header explains that 'KubeSphere 中的项目对应的是 Kubernetes 的 namespace，是对一组资源和对象的抽象集合，常用来将系统内部的对象划分为不同的项目组或用户组。' A search bar with placeholder text '请输入名称进行查找' is located above the main table. The main table lists two projects:

名称	容器组数量	CPU 使用量	内存使用量	管理员	创建时间
<code>kubesphere-sample-prod(生产环境)</code>	0	-	-	admin	2019-04-12 15:49:27
<code>kubesphere-sample-dev(开发环境)</code>	0	-	-	admin	2019-04-12 15:43:38

# 创建流水线

## 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。



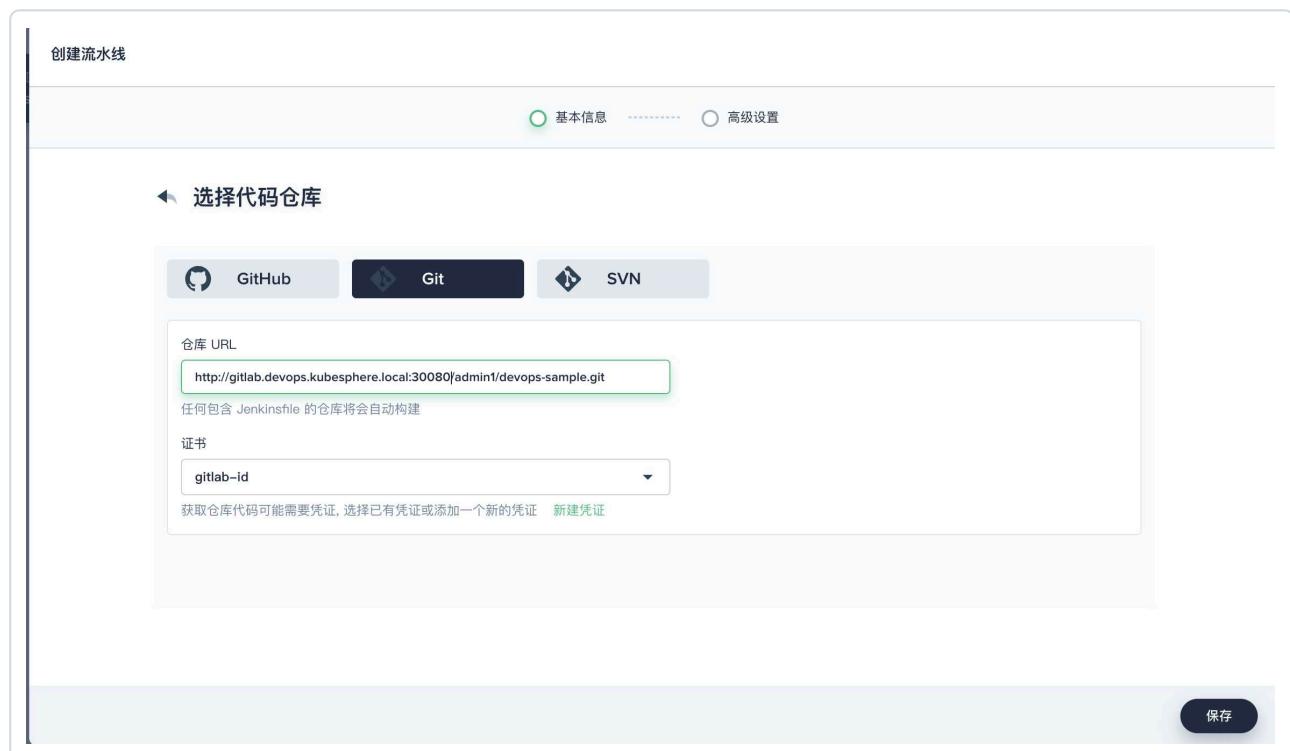
2、在弹出的窗口中，输入流水线的基本信息。

- 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

The screenshot shows the 'Basic Information' section of the Pipeline creation form. It includes fields for 'Name' (必填) containing 'jenkinsfile-in-SCM', 'Project' (必填) containing 'project-RyonypM4PX2q', and a 'Description' field with the text 'DevOps demo'. A note below the project field says: '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' (Resources will be grouped by project, allowing resources to be viewed by project). At the bottom, there's a red arrow pointing to a button labeled '点击添加代码仓库' (Click to add code repository) with the placeholder text '请选择一个代码仓库作为 Pipeline 的代码源' (Select a code repository as the source for the Pipeline).

## 第二步：添加 Git 仓库

1、点击代码仓库，以添加 GitLab 仓库为例。



2、输入仓库URI，默認為 `http://gitlab.devops.kubesphere.local:30080/admin1/devops-java-sample.git`，

注意：GitLab 中提供的 HTTP 和 SSH URI 有误。HTTP URI 需要手动加上端口号30080，SSH URI 需要手动加上协议 `ssh://` 和端口号：30090。

证书选择之前创建的 `gitlab-id`。

点击「保存」后进行下一步。

## 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 **保留分支的天数** 和 **保留分支的最大个数** 默认为 -1。



### 说明：

分支设置的保留分支的天数和保持分支的最大个数两个选项可以同时对分支进行作用，只要某个分支的保留天数和个数不满足任何一个设置的条件，则将丢弃该分支。假设设置的保留天数和个数为 2 和 3，则分支的保留天数一旦超过 2 或者保留个数超过 3，则将丢弃该分支。默认两个值为 -1，表示不自动删除分支。

丢弃旧的分支将确定何时应丢弃项目的分支记录。分支记录包括控制台输出，存档工件以及与特定分支相关的其他元数据。保持较少的分支可以节省 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分支：

- 保留分支的天数：如果分支达到一定的天数，则丢弃分支。
- 保留分支的个数：如果已经存在一定数量的分支，则丢弃最旧的分支。

2、默认的 脚本路径 为 Jenkinsfile，需要修改为 `Jenkinsfile-on-prem`。

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。

3、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`。

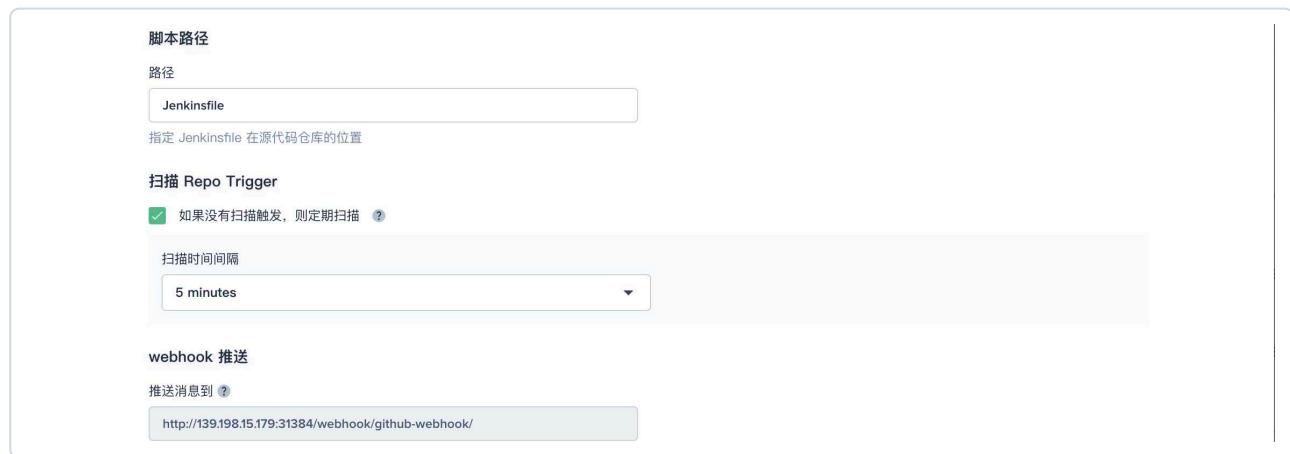
说明：定期扫描是设定一个周期让流水线周期性地扫描远程仓库，根据 行为策略 查看仓库有没有代码更新或新的 PR。

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**。

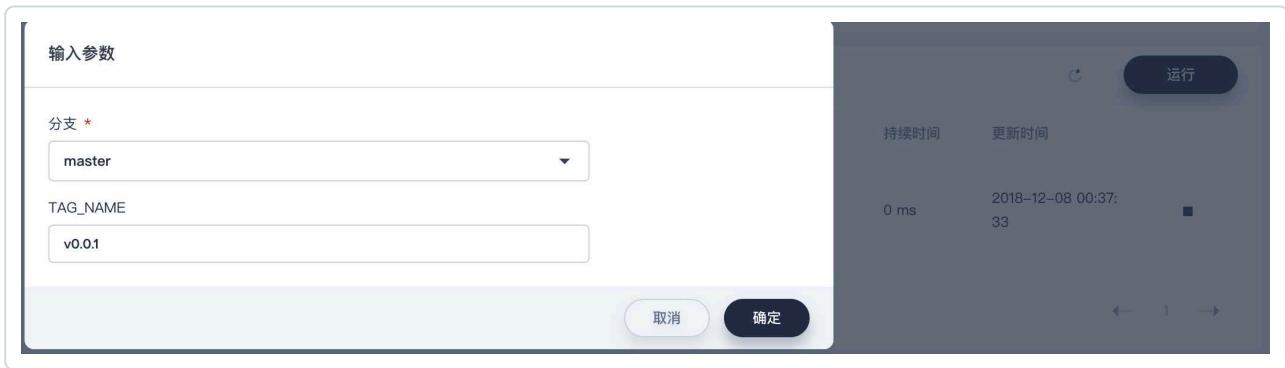


## 第四步：运行流水线

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 (此示例为根目录下的 Jenkinsfile-on-prem)。
- 2、由于仓库的 Jenkinsfile-on-prem 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 GitLab 和 Harbor 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，流水线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-on-prem 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')
```

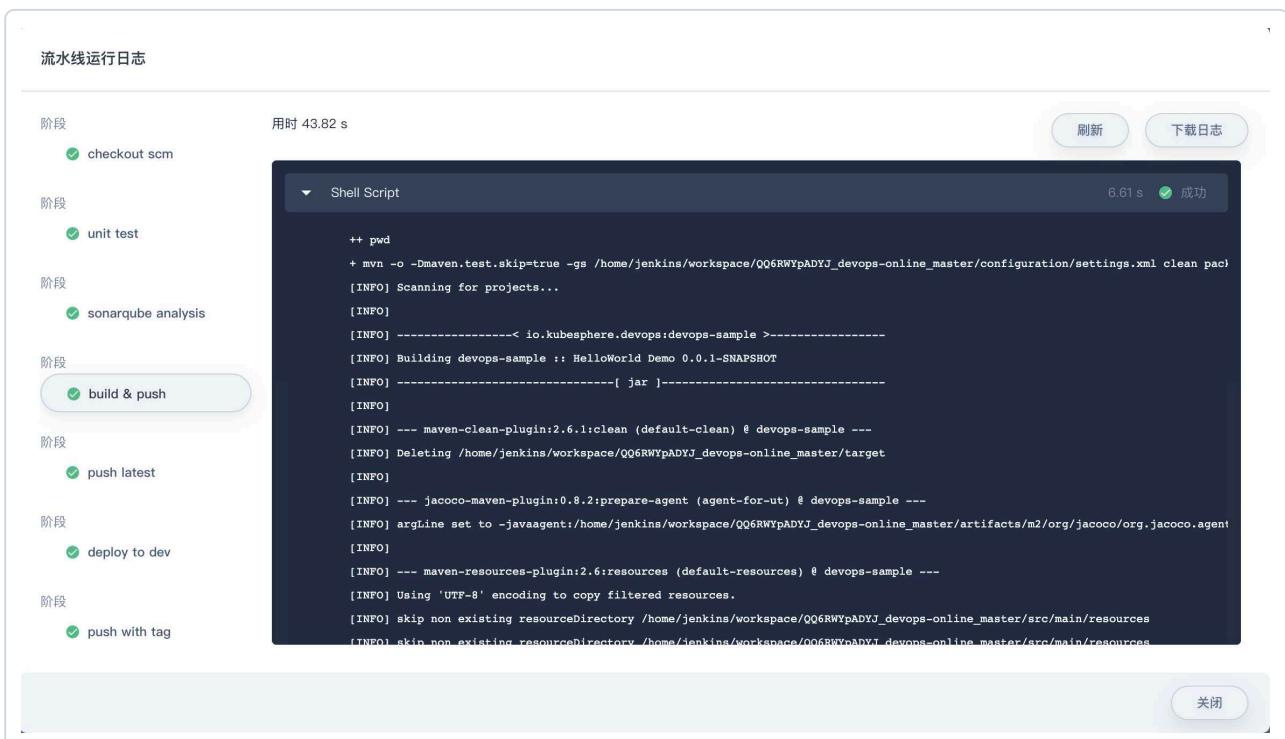
...

## 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-on-prem](#) 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的日志。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

1、若流水线执行成功，点击该流水线下的 **代码质量**，即可看到通过 sonarQube 的代码质量检测结果，如下图(仅供参考)。

2、流水线最终 build 的 Docker 镜像也将被成功地 push 到 Harbor 中，我们在 Jenkinsfile-on-prem 中已经配置过 Harbor，登录 Harbor 查看镜像的 push 结果，可以看到 tag 为 snapshot、TAG\_NAME(master-1)、latest 的镜像已经被 push 到 Harbor，并且在 GitLab 中也生成了一个新的 tag 和 release。示例网站最终将以 deployment 和 service 分别部署到 KubeSphere 的 `kubesphere-sample-dev` 和 `kubesphere-sample-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 : 30861 ( <code> \${EIP}:\${NODEPORT} </code> )	<code>kubesphere-sample-dev</code>	<code>ks-sample-dev</code>	<code>ks-sample-dev</code>
Production	公网 IP : 30961 ( <code> \${EIP}:\${NODEPORT} </code> )	<code>kubesphere-sample-prod</code>	<code>ks-sample</code>	<code>ks-sample</code>

3、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-sample-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-sample` 已创建成功。正常情况下，部署的状态应该显示 **运行中**。

The screenshot shows the KubeSphere interface. On the left, there's a sidebar with project navigation (Platform Management, Workstation, Application Catalog) and a user dropdown. The main area is titled '部署' (Deployment). It shows a summary: 1 used quota, infinity planned quota, and infinity remaining pod quota. Below is a search bar and a table with columns: Name, Status, Application, and Last Updated. One entry is listed: 'ks-sample' is running (2/2), last updated on 2019-03-28 13:56:05. A note at the bottom says '共 1 个条目' (1 item).

4、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 **30961**。

## 查看服务

The screenshot shows the KubeSphere interface. On the left, there's a sidebar with project navigation (Platform Management, Workstation, Application Catalog) and a user dropdown. The main area is titled '服务' (Service). It shows a summary: 1 used quota, 1 virtual IP, and 0 Headless services. Below is a search bar and a table with columns: Name, IP地址 (IP Address), 端口 (Port), 应用 (Application), and 创建时间 (Creation Time). One entry is listed: 'ks-sample' has a Virtual IP of 10.233.42.3, port 8080:8080/TCP, application 'ks-sample', and was created on 2019-03-28 10:25:38. A note at the bottom says '共 1 个条目' (1 item).

5、查看推送到您个人的 Harbor 中的镜像，可以看到 **devops-java-sample** 就是 APP\_NAME 的值，而 tag 也是在 Jenkinsfile-on-prem 中定义的 tag。

## 访问示例服务

在浏览器或通过后台命令访问部署到 KubeSphere Dev 和 Production 环境的服务：

### Dev 环境

例如，浏览器访问 **http://192.168.0.20:30861/** (即 **http://IP:NodePort/**) 可访问到 **Hello,World!** 页面，或通过后台命令验证：

```
curl http://192.168.0.20:30861  
Hello,World!
```

## Production 环境

同上可访问 <http://192.168.0.20:30961/> (即 <http://IP:NodePort/> )。

至此，结合 GitLab 和 Harbor，在离线环境下创建一个 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](#)。

## 示例十三 – 使用 Ingress–Nginx 进行灰度发布

在 [Bookinfo 微服务的灰度发布示例](#) 中，KubeSphere 基于 Istio 对 Bookinfo 微服务示例应用实现了灰度发布。有用户表示自己的项目还没有上 Istio，要如何实现灰度发布？

在 [Ingress–Nginx \(0.21.0 版本\)](#) 中，引入了一个新的 Canary 功能，可用于为网关入口配置多个后端服务，还可以使用指定的 annotation 来控制多个后端服务之间的流量分配。KubeSphere 在 [2.0.2 的版本](#) 中，升级了项目网关 (Ingress Controller) 版本至 0.24.1，支持基于 [Ingress–Nginx](#) 的灰度发布。

上一篇文章已经对灰度发布的几个应用场景进行了详细介绍，本文将直接介绍和演示基于 KubeSphere 使用应用路由 (Ingress) 和项目网关 (Ingress Controller) 实现灰度发布。

说明：本文用到的示例 yaml 源文件及代码已上传至 [GitHub](#)，可 clone 至本地方便参考。

### Ingress–Nginx Annotation 简介

KubeSphere 基于 [Nginx Ingress Controller](#) 实现了项目的网关，作为项目对外的流量入口和项目中各个服务的反向代理。而 Ingress–Nginx 支持配置 Ingress Annotations 来实现不同场景下的灰度发布和测试，可以满足金丝雀发布、蓝绿部署与 A/B 测试等业务场景。

[Nginx Annotations](#) 支持以下 4 种 Canary 规则：

- `nginx.ingress.kubernetes.io/canary-by-header`：基于 Request Header 的流量切分，适用于灰度发布以及 A/B 测试。当 Request Header 设置为 `always` 时，请求将一直被发送到 Canary 版本；当 Request Header 设置为 `never` 时，请求不会被发送到 Canary 入口；对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他金丝雀规则进行优先级的比较。
- `nginx.ingress.kubernetes.io/canary-by-header-value`：要匹配的 Request Header 的值，用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。当 Request Header 设置为此值时，它将被路由到 Canary 入口。该规则允许用户自定义 Request Header 的值，必须与上一个 annotation (即：canary-by-header) 一起使用。
- `nginx.ingress.kubernetes.io/canary-weight`：基于服务权重的流量切分，适用于蓝绿部署，权重范围 0 – 100 按百分比将请求路由到 Canary Ingress 中指定的服务。权重为 0 意味着该金丝雀规则不会向 Canary 入口的服务发送任何请求。权重为 100 意味着所有请求都将被发送到 Canary 入口。

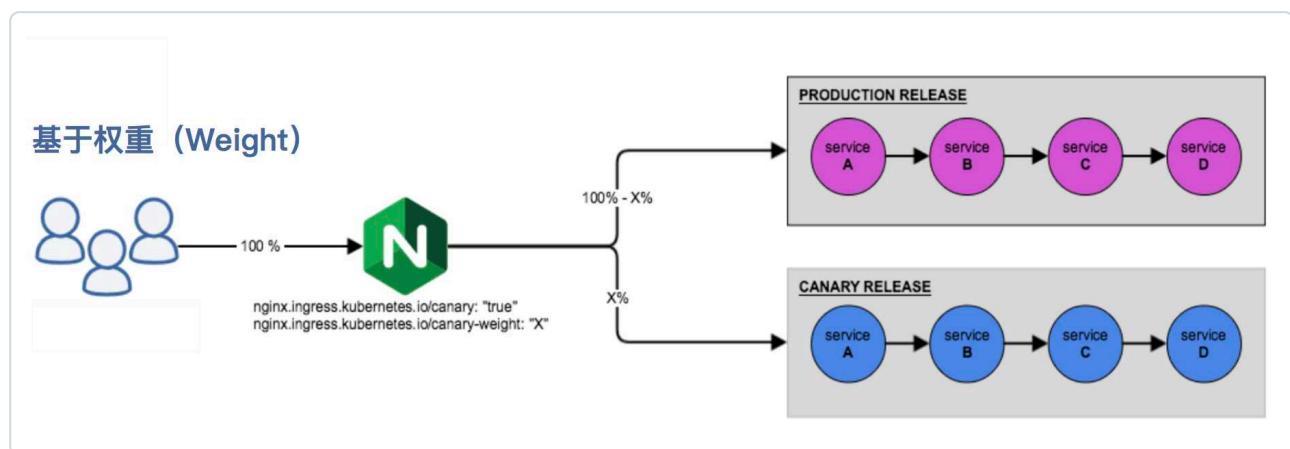
- [nginx.ingress.kubernetes.io/canary-by-cookie](#): 基于 Cookie 的流量切分，适用于灰度发布与 A/B 测试。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务的 cookie。当 cookie 值设置为 `always` 时，它将被路由到 Canary 入口；当 cookie 值设置为 `never` 时，请求不会被发送到 Canary 入口；对于任何其他值，将忽略 cookie 并将请求与其他金丝雀规则进行优先级的比较。

**注意：金丝雀规则按优先顺序进行如下排序：**

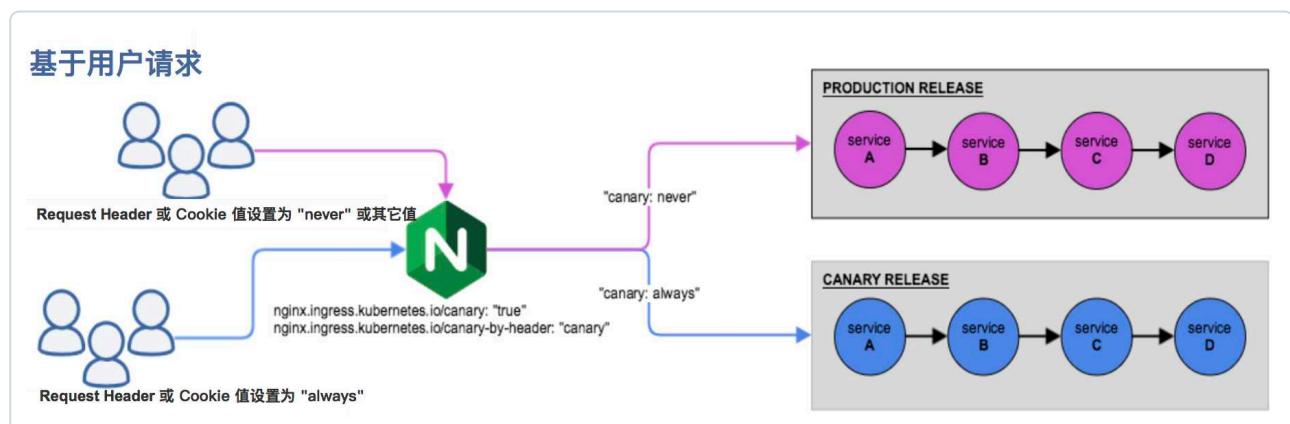
`canary-by-header -> canary-by-cookie -> canary-weight`

把以上的四个 annotation 规则可以总体划分为以下两类：

- 基于权重的 Canary 规则



- 基于用户请求的 Canary 规则



# 第一步：创建项目和 Production 版本的应用

1.1. 在 KubeSphere 中创建一个企业空间 (workspace) 和项目 (namespace)，可参考 [多租户管理快速入门](#)。如下已创建了一个示例项目。

The screenshot shows the KubeSphere project management interface for the 'demo-project'. The top navigation bar includes '平台管理', '工作台', '应用模板', 'KUBESPHERE' logo, and 'admin'. The main content area has a blue header with the project name 'demo-project', owner 'project-admin', workspace 'demo-workspace', and creation time '2019-07-16 13:25:47'. On the left is a sidebar with sections: 概览, 应用, 工作负载, 存储卷, 网络与服务, 监控告警, 配置中心, and 项目设置. The central dashboard displays resource status (部署: 2, 有状态副本集: 0, 守护进程集: 0, 任务: 0, 定时任务: 0, 存储卷: 4, 服务: 1, 应用路由: 0), container group quantity changes over time (08:11 to 10:47:36), and a deployed application 'grafana-i47bmc' (grafana). A bottom right '工具箱' (Toolbox) panel contains links for 'kubectl' (with a red arrow pointing to it), '日志查询' (Log Query), and 'kubeconfig'. A note at the bottom right says 'Shift + 鼠标左键 可以在新窗口中打开' (Shift + Left Click to open in a new window).

1.2. 为了快速创建应用，在项目中创建工作负载和服务时可通过 [编辑 yaml](#) 的方式，或使用 KubeSphere 右下角的工具箱打开 [web kubectl](#) 并使用以下命令和 yaml 文件创建一个 Production 版本的应用并暴露给集群外访问。如下创建 Production 版本的 [deployment](#) 和 [service](#)。

kubectl

```
/ # kubectl apply -f production.yaml -n ingress-demo
```

**kubectl 常用命令**  
查阅更多命令请参照 [官方文档](#)

**kubectl 输出格式**

- 显示Pod的更多信息  
`kubectl get pod <pod-name> -o wide`
- 以yaml格式显示Pod的详细信息  
`kubectl get pod <pod-name> -o yaml`

**kubectl 操作**

1. 创建资源对象
  - 根据yaml配置文件一次性创建service和rc  
`kubectl create -f my-service.yaml -f my-rc.yaml`
  - 对目录下所有.yaml、.yml、.json文件进行创建操作  
`kubectl create -f <directory>`

```
$ kubectl apply -f production.yaml -n ingress-demo
deployment.extensions/production created
service/production created
```

其中用到的 yaml 文件如下：

### production.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: production
spec:
  replicas: 1
  selector:
    matchLabels:
      app: production
  template:
    metadata:
      labels:
        app: production
    spec:
      containers:
        - name: production
          image: mirror.googlecontainers/echoserver:1.10
          ports:
            - containerPort: 8080
      env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
```

---

```
apiVersion: v1
```

KubeSphere

kind: Service

metadata:

256 / 670

1.3. 创建 Production 版本的应用路由 (Ingress)。

```
$ kubectl apply -f production.ingress -n ingress-demo  
ingress.extensions/production created
```

其中用到的 yaml 文件如下：

#### production.ingress

```
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: production  
  annotations:  
    kubernetes.io/ingress.class: nginx  
spec:  
  rules:  
  - host: kubesphere.io  
    http:  
      paths:  
      - backend:  
          serviceName: production  
          servicePort: 80
```

## 第二步：访问 Production 版本的应用

2.1. 此时，在 KubeSphere UI 的企业空间 demo-workspace 下，可以看到 ingress-demo 项目下的所有资源。

#### Deployment

部署

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的所需状态描述为 Pod 模板。

2 已用配额  
∞ 规划配额  
∞ 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	应用	更新时间
production	运行中(1/1)	-	2019-08-26 11:34:32

## Service

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。

2 已用配额  
∞ 规划配额

官网文档 参考文档

输入查询条件进行过滤

名称	IP地址	端口	应用	创建时间
production	Virtual IP: 10.233.18.127	端口: 80:8080/TCP 节点端口: -	-	2019-08-26 11:34:32

## Ingress

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。

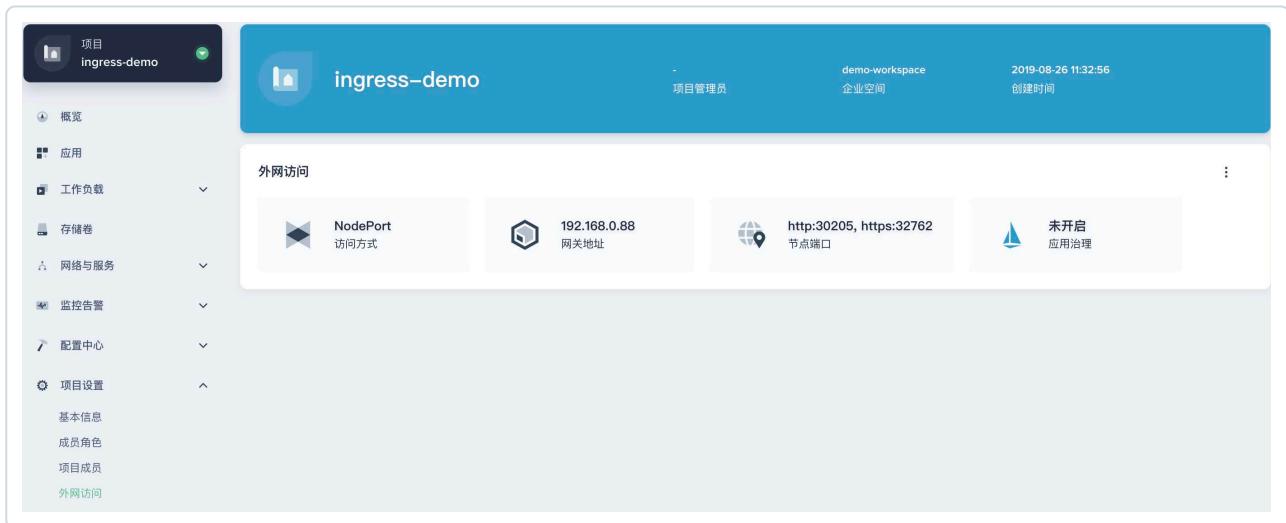
2 已用配额  
- 外网访问  
- Internet IP

官网文档 参考文档

输入查询条件进行过滤

名称	网关地址	应用	创建时间
production	192.168.0.88	-	2019-08-26 11:48:26

2.2. 访问 Production 版本的应用需确保当前项目已开启了网关，在外网访问下打开网关，类型为 NodePort。



2.3. 如下访问 Production 版本的应用。

```
$ curl kubesphere.io:30205
```

Hostname: production-6b4bb8d58d-7r889

Pod Information:

```
node name: ks-allinone
pod name: production-6b4bb8d58d-7r889
pod namespace: ingress-demo
pod IP: 10.233.87.165
```

Server values:

```
server_version=nginx: 1.12.2 – lua: 10010
```

Request Information:

```
client_address=10.233.87.225
method=GET
real path=/
query=
request_version=1.1
request_scheme=http
request_uri=http://kubesphere.io:8080/
```

Request Headers:

```
accept=*/
host=kubesphere.io:30205
user-agent=curl/7.29.0
apiVersion: extensions/v1beta1
x-forwarded-for=192.168.0.88
x-forwarded-host=kubesphere.io:30205
x-forwarded-port=80
x-forwarded-proto=http
x-original-uri=/
x-real-ip=192.168.0.88
x-request-id=9596df96e994ea05bece2ebbe689a2cc
x-scheme=http
```

Request Body:

```
-no body in request-
```

## 第三步：创建 Canary 版本

参考将上述 Production 版本的 `production.yaml` 文件，再创建一个 Canary 版本的应用，包括一个 Canary 版本的 `deployment` 和 `service`（为方便快速演示，仅需将 `production.yaml` 的 `deployment` 和 `service` 中的关键字 `production` 直接替换为 `canary`，实际场景中可能涉及业务代码变更）。

## 第四步：Ingress–Nginx Annotation 规则

### 基于权重 (Weight)

基于权重的流量切分的典型应用场景就是 `蓝绿部署`，可通过将权重设置为 0 或 100 来实现。例如，可将 Green 版本设置为主要部分，并将 Blue 版本的入口配置为 Canary。最初，将权重设置为 0，因此不会将流量代理到 Blue 版本。一旦新版本测试和验证都成功后，即可将 Blue 版本的权重设置为 100，即所有流量从 Green 版本转向 Blue。

- 4.1. 使用以下 `canary.ingress` 的 yaml 文件再创建一个基于权重的 Canary 版本的应用路由 (Ingress)。

**注意：**要开启灰度发布机制，首先需设置 `nginx.ingress.kubernetes.io/canary: "true"` 启用 Canary，以下 Ingress 示例的 Canary 版本使用了基于权重进行流量切分的 annotation 规则，将分配 30% 的流量请求发送至 Canary 版本。

```

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: canary
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "30"
spec:
  rules:
  - host: kubesphere.io
    http:
      paths:
      - backend:
          serviceName: canary
          servicePort: 80

```

The screenshot shows the KubeSphere application routing interface. On the left, there's a sidebar with project navigation (ingress-demo), a summary section, and links for Overview, Application, Workload, Storage, Network & Services, and Application Routing. The Application Routing section is currently selected. The main area is titled '应用路由' (Application Routing) and contains a brief description: '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。'. It shows 2 routes listed:

名称	网关地址	应用	创建时间
production	192.168.0.88	-	2019-08-26 11:48:26
canary	192.168.0.88	-	2019-08-26 14:24:24

#### 4.2. 访问应用的域名。

**说明：应用的 Canary 版本基于权重 (30%) 进行流量切分后，访问到 Canary 版本的概率接近 30%，流量比例可能会有小范围的浮动。**

```
for i in $(seq 1 10); do curl -s http://kubesphere.io:30205 | grep "Hostname"; done
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: canary-57554dbcdd-cddmd
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: canary-57554dbcdd-cddmd
Hostname: canary-57554dbcdd-cddmd
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
```

## 基于 Request Header

4.3. 基于 Request Header 进行流量切分的典型应用场景即 灰度发布或 A/B 测试场景。参考以下截图，在 KubeSphere 给 Canary 版本的 Ingress 新增一条 annotation `nginx.ingress.kubernetes.io/canary-by-header: canary` (这里的 annotation 的 value 可以是任意值)，使当前的 Ingress 实现基于 Request Header 进行流量切分。

**说明：**金丝雀规则按优先顺序 `canary-by-header -> canary-by-cookie -> canary-weight` 进行如下排序，因此以下情况将忽略原有 `canary-weight` 的规则。

<code>kubernetes.io/ingress.class</code>	<code>nginx</code>
<code>nginx.ingress.kubernetes.io/canary</code>	<code>true</code>
<code>nginx.ingress.kubernetes.io/canary-weight</code>	<code>30</code>
<code>nginx.ingress.kubernetes.io/canary-by-header</code>	<code>canary</code>

4.4. 在请求中加入不同的 Header 值，再次访问应用的域名。

说明：

举两个例子，如开篇提到的当 Request Header 设置为 never 或 always 时，请求将 不会 或 一直 被发送到 Canary 版本；

对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他 Canary 规则进行优先级的比较（如下第二次请求已将 基于 30% 权重 作为第一优先级）。

```
$ for i in $(seq 1 10); do curl -s -H "canary: never" http://kubesphere.io:30205 | grep "Hostname"; done
Hostname: production-6b4bb8d58d-7r889
$ for i in $(seq 1 10); do curl -s -H "canary: other-value" http://kubesphere.io:30205 | grep "Hostname"; done
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: canary-57554dbcdd-cddmd
Hostname: production-6b4bb8d58d-7r889
Hostname: production-6b4bb8d58d-7r889
Hostname: canary-57554dbcdd-cddmd
Hostname: canary-57554dbcdd-cddmd
```

4.5. 此时可以在上一个 annotation (即 canary-by-header) 的基础上添加一条

`nginx.ingress.kubernetes.io/canary-by-header-value: user-value`。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。

## 注解

The screenshot shows a list of annotations for a service. The annotations are:

- kubernetes.io/ingress.class: nginx
- nginx.ingress.kubernetes.io/canary: true
- nginx.ingress.kubernetes.io/canary-by-header: canary
- nginx.ingress.kubernetes.io/canary-weight: 30
- nginx.ingress.kubernetes.io/canary-by-header-value: user-value

A red box highlights the last two annotations: "nginx.ingress.kubernetes.io/canary-by-header-value" and its value "user-value". There is also a button labeled "添加注解" (Add Annotation) at the bottom right.

4.6. 如下访问应用的域名，当 Request Header 满足此值时，所有请求被路由到 Canary 版本（该规则允许用户自定义 Request Header 的值）。

```
$ for i in $(seq 1 10); do curl -s -H "canary: user-value" http://kubesphere.io:30205 | grep "Hostname"; done
Hostname: canary-57554dbcdd-cddmd
```

## 基于 Cookie

4.7. 与基于 Request Header 的 annotation 用法规则类似。例如在 [A/B 测试场景](#) 下，需要让地域为北京的用户访问 Canary 版本。那么当 cookie 的 annotation 设置为 `nginx.ingress.kubernetes.io/canary-by-cookie: "users_from_Beijing"`，此时后台可对登录的用户请求进行检查，如果该用户访问源来自北京则设置 cookie `users_from_Beijing` 的值为 `always`，这样就可以确保北京的用户仅访问 Canary 版本。

## 总结

灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以对新版本进行测试、发现和调整问题，以

保证其影响度。本文通过多个示例演示和说明了基于 KubeSphere 使用应用路由 (Ingress) 和项目网关 (Ingress Controller) 实现灰度发布，并详细介绍了 Ingress–Nginx 的四种 Annotation，还未使用 Istio 的用户也能借助 Ingress–Nginx 轻松实现灰度发布与金丝雀发布。

## 参考

- [NGINX Ingress Controller – Annotations](#)
- [canary deployment with ingress-nginx](#)
- [Canary Deployments on Kubernetes without Service Mesh](#)

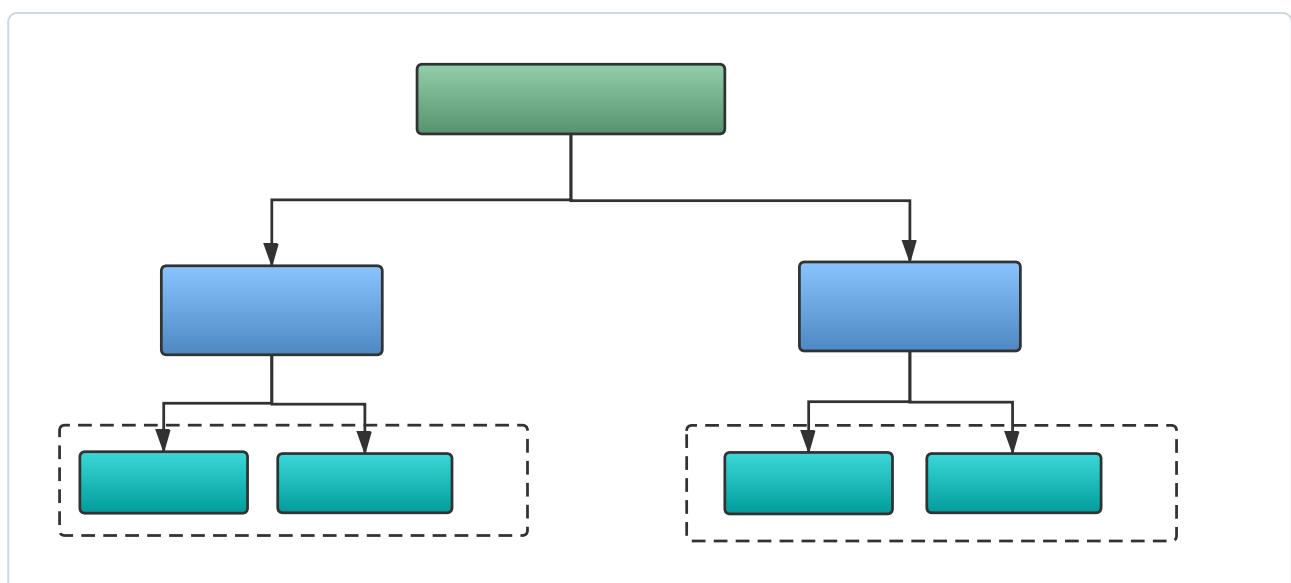
# 多租户管理概述

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

针对不同层级的资源都可以灵活地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。

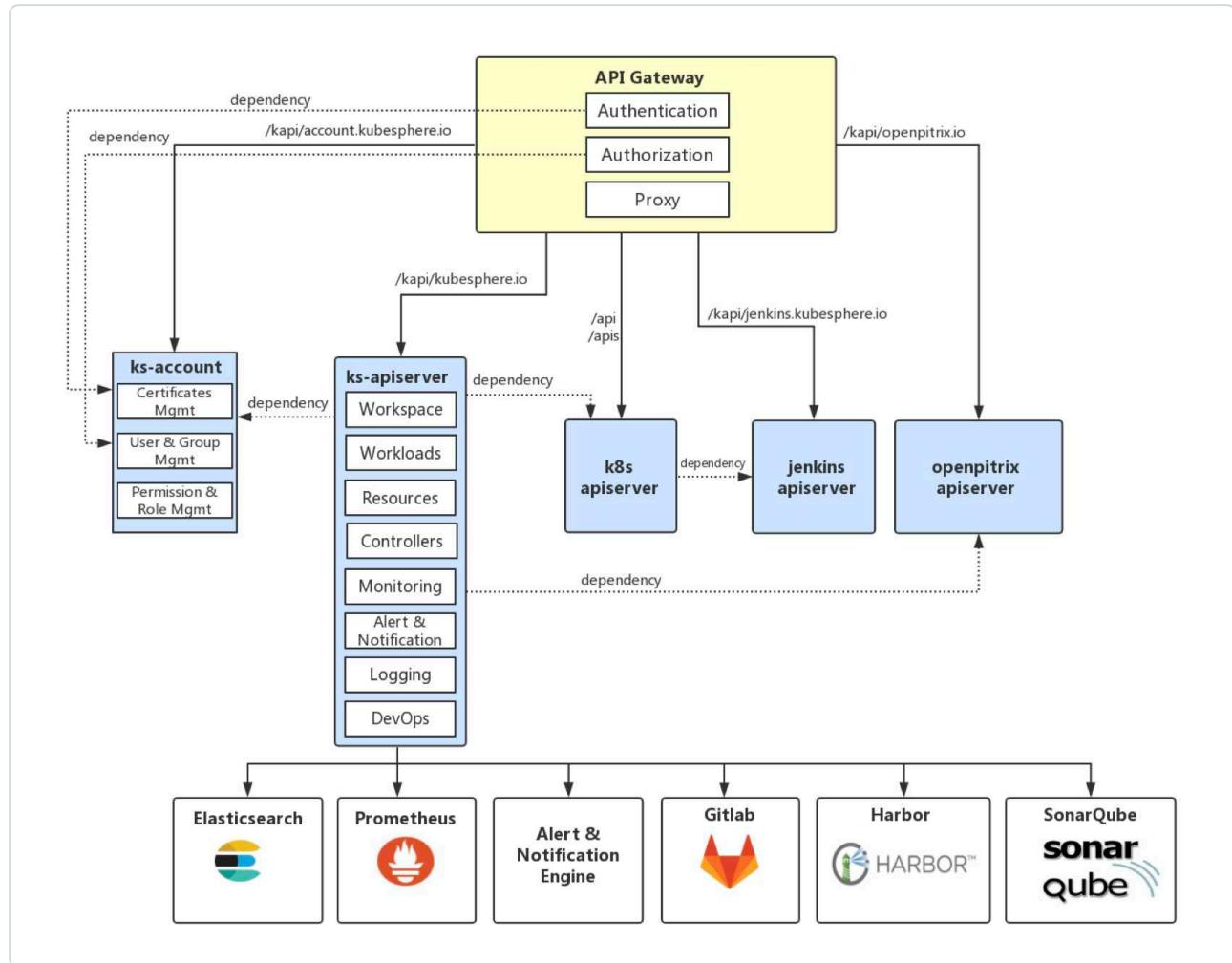
## 企业空间权限控制

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，如果各层级内置的角色不能满足用户需求时，平台支持管理员创建新的角色并自定义角色权限的规则列表，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

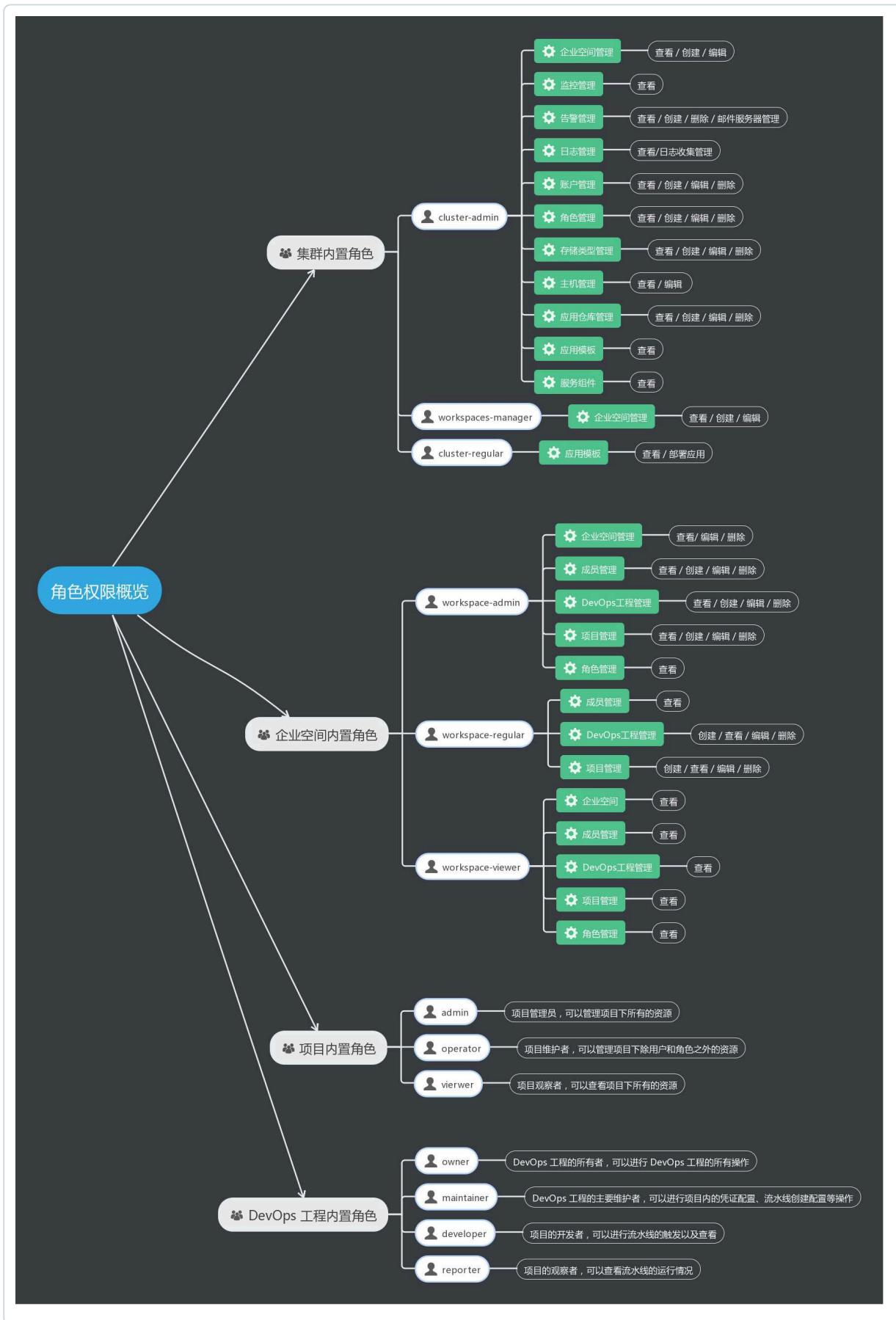
角色分为集群、企业空间、项目 / DevOps 工程共三个层级，将用户和不同层级的资源关联起来。



## 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、角色管理、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、存储类型、集群监控、应用仓库、邮件服务器、日志收集、服务组件、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，向项目内邀请成员并授予预置的项目角

色，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 / 编辑 / 删除	查看
监控管理	查看	查看	查看
告警管理	查看 / 创建 / 删除	查看 / 创建 / 删除	查看
成员管理	查看 / 创建 / 编辑 / 删除	查看	查看
角色管理	查看 / 创建 / 编辑 / 删除	查看	查看
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 / 查看	远程连接 / 查看	远程连接 / 查看
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

资源/权限/ 角色	admin	operator	viewer
应用管理	查看 / 部署 / 编辑 / 删除	查看 / 部署 / 编辑 / 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 理	查看 / 创建 / 编辑 / 删除	查看 / 创建 / 编辑 / 删除	查看
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

为了方便细粒度的对 DevOps 工程进行管控，DevOps 工程的管理员可以从企业空间下的用户池中搜索并添加用户，为新邀请的用户授予内置的角色。

### 预置的 DevOps 工程角色：

预置角色	描述
owner	DevOps 工程的所有者，可以进行项目的所有操作。
maintainer	DevOps 工程的主要维护者，可以进行项目内的凭证配置、流水线配置等操作。
developer	DevOps 工程的开发者，可以进行流水线的触发和查看。
reporter	DevOps 工程的观察者，可以查看流水线的运行情况。

# 企业空间管理

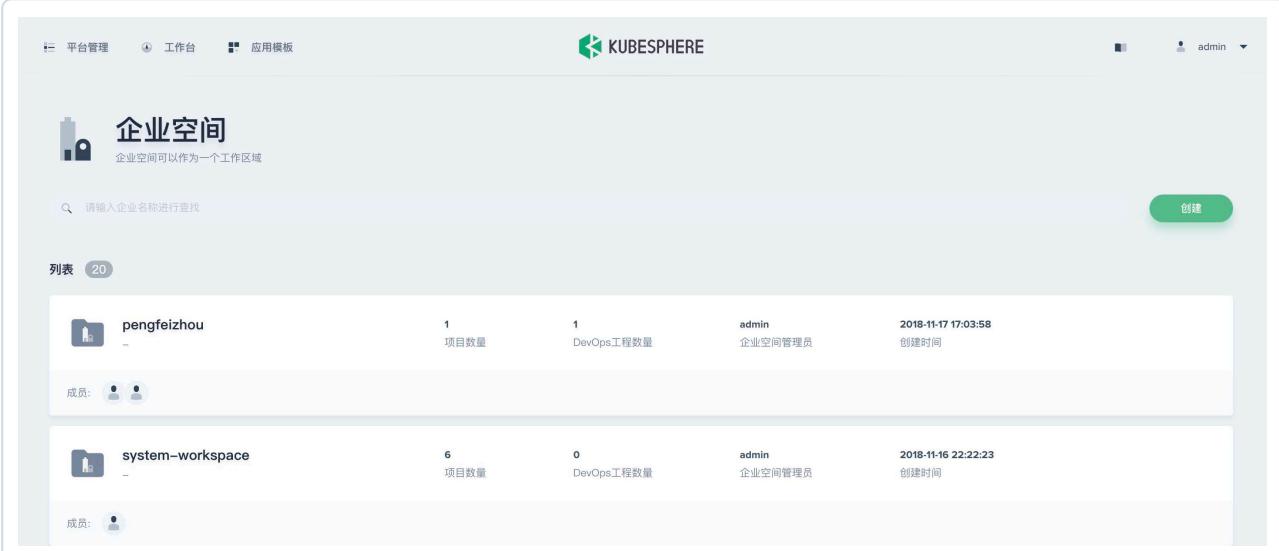
企业空间 (workspace) 是 KubeSphere 实现多租户模式的基础，是您管理项目、DevOps 工程和企业成员的基本单位。当用户创建企业空间后，该用户即是此企业空间的管理员 (admin)，有权查看和管理这个企业空间内的所有对象 (项目、DevOps 工程)，同时管理员将继承该企业空间内项目和工程的管理权限。集群中的 cluster-admin 和 workspaces-manager 角色拥有企业空间管理的权限，支持查看、创建、编辑或删除企业空间。本文档说明管理企业空间的常用功能。

## 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. The main area displays a list of existing workspaces:

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

A green 'Create' button is located in the top right corner of the workspace list area.

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 icon for admin. The main header features the KubeSphere logo and the text 'KUBESPHERE'. On the left sidebar, there are sections for 企业空间 (Enterprise Space) named '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。

This screenshot shows a detailed monitoring view for the 'CPU 使用量 (核)' (CPU Usage (Core)) over time. The chart displays usage from 09:00 to 08:20, with a notable peak around 17:24 where the usage reaches approximately 2.4 cores. A specific data point is highlighted at 2018-11-18 21:50 with a value of 1.87 核 (1.87 cores).

时间	使用量
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 menu includes: 企业空间 (Enterprise Space) with sample selected, 概览 (Overview), 项目管理 (Project Management), DevOps工程 (DevOps Project), 企业空间管理 (Enterprise Space Management) with 基本信息 (Basic Information) selected, 企业角色 (Enterprise Role), and 成员管理 (Member Management). 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

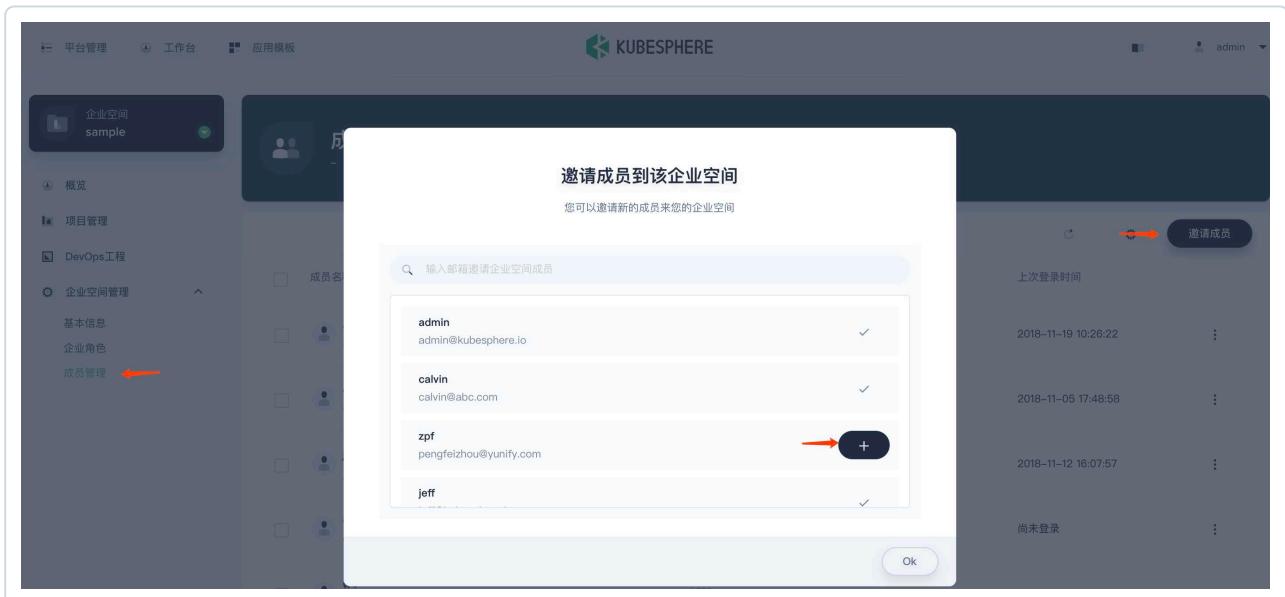
平台预置了三个常用的企业角色 workspace-admin、workspace-regular 和 workspace-viewer，预置角色拥有的权限见下表。

### 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。

目前在服务组件可以查看以下服务的相关组件的监控状态：

- KubeSphere
- Kubernetes
- OpenPitrix
- Istio
- Monitoring
- Logging
- DevOps

## 查看服务组件

使用集群管理员账号登录 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：

The screenshot shows the KubeSphere Service Component page. At the top, there are navigation tabs: 平台管理 (Platform Management), 工作台 (Workstation), 应用模板 (Application Template), and the KubeSphere logo. Below the navigation is a dark header bar with the title '服务组件' (Service Components) and a subtitle explaining its function: '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。' (Service components provide monitoring of the health status of various service components in the KubeSphere, Kubernetes, and OpenPitrix clusters. It can check the current cluster's health status and running time, helping users monitor the cluster's status and quickly locate problems.)

Below the header, there is a navigation bar with tabs: KubeSphere (7), Kubernetes (5), OpenPitrix (13), Istio (11), Monitoring (6), Logging (2), and DevOps (30). The main content area displays a table of service components:

组件	状态	副本数量	运行时间
ks-account	健康	1 / 1	3小时
ks-apigateway	健康	1 / 1	3小时
ks-apiserver	健康	1 / 1	3小时
ks-console	健康	2 / 2	3小时
ks-docs	健康	2 / 2	3小时
openldap	健康	1 / 1	3小时

## 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，其 Tab 将显示为异常组件的数目。

## KubeSphere

Components	Description
ks-account	提供用户、权限管理相关的API
ks-apigateway	负责处理服务请求和处理 API 调用过程中的所有任务

Components	Description
ks-apiserver	是整个集群管理的 API 接口和集群内部各个模块之间通信的枢纽，以及集群安全控制
ks-console	提供 KubeSphere 的控制台服务
ks-docs	<a href="https://docs.kubesphere.io">docs.kubesphere.io</a> 文档服务
OpenLDAP	负责集中存储和管理用户账号信息
redis	将结构化的数据存储在内存中的存储系统

## Kubernetes

Components	Description
coredns	为 Kubernetes 集群提供服务发现的功能
metrics-server	Kubernetes 的监控组件，从每个节点的 Kubelet 采集指标信息
tiller-deploy	Helm 的服务端，负责管理发布 release
kube-controller-manager	kube-controller-manager 由一系列的控制器组成，处理集群中常规任务的后台线程
kube-scheduler	kubernetes 的调度器，将 Pod 调度到合适的 Node 节点上去

## OpenPitrix

Components	Description
openpitrix-api-gateway	负责处理平台的服务请求和处理 API 调用过程中的所有任务
openpitrix-app-manager	提供 OpenPitrix 的应用生命周期管理
openpitrix-category-manager	提供 OpenPitrix 中的应用分类管理
openpitrix-cluster-manager	提供 OpenPitrix 中的应用实例生命周期管理

Components	Description
openpitrix-db	OpenPitrix 数据库
openpitrix-etcd	高可用键值存储系统，用于共享配置、服务发现和全局锁
openpitrix-iam-service	控制哪些用户可使用您的资源（身份验证）以及可使用的资源和采用的方式（授权）
openpitrix-job-manager	具体执行 OpenPitrix 应用实例生命周期 Action
openpitrix-minio	对象存储服务，用于存储非结构化数据
openpitrix-repo-indexer	提供 OpenPitrix 的应用仓库索引服务
openpitrix-repo-manager	提供 OpenPitrix 的应用仓库管理
openpitrix-runtime-manager	提供平台中的云运行时环境管理
openpitrix-task-manager	具体执行 OpenPitrix 应用实例生命周期 Action 子任务

## Istio

Components	Description
istio-citadel	通过内置身份和凭证管理赋能强大的服务间和最终用户身份验证
istio-galley	代表其他的 Istio 控制平面组件，用来验证用户编写的 Istio API 配置
istio-ingressgateway	提供外网访问的网关
istio-pilot	为 Envoy Sidecar 提供服务发现功能
istio-policy	用于向 Envoy 提供准入策略控制，黑白名单控制，速率限制等相关策略
istio-sidecar-injector	为配置注入的 pod 自动注入 Sidecar
istio-telemetry	为 Envoy 提供了数据上报和日志搜集服务
jaeger-collector	收集 sidecar 的数据，Istio 里面 sidecar 就是 jaeger-agent
jaeger-collector-	收集 sidecar 的数据，Istio 里面 Sidecar 就是 jaeger-agent

Components	Description
headless	
jaeger-query	接收查询请求，然后从后端存储系统中检索 trace 并通过 UI 进行展示
jaeger-operator	负责创建 Jaeger 服务，并在配置更新时自动应用到 jaeger 服务

## Monitoring

Components	Description
kube-state-metrics	监听 Kubernetes API server 以获取集群中各种 API 对象的状态包括节点，工作负载和 Pod 等，并生成相关监控数据供 Prometheus 抓取
node-exporter	收集集群各个节点的监控数据，供 Prometheus 抓取
prometheus-k8s	提供节点、工作负载、API 对象相关监控数据
prometheus-k8s-system	提供 etcd, coredns, kube-apiserver, kube-scheduler, kube-controller-manager 等 Kubernetes 组件的监控数据
prometheus-operator	管理 Prometheus 实例的 Operator
prometheus-operated	所有 Prometheus 实例对应的服务，供 Prometheus Operator 内部使用

## Logging

Components	Description
elasticsearch-logging-data	提供 Elasticsearch 数据存储、备份、搜索等数据服务
elasticsearch-logging-discovery	提供 Elasticsearch 集群管理服务

## DevOps

Components	Description
controller-manager-metrics-service	提供 s2i 控制器的监控数据
ks-jenkins	jenkins master 服务, 提供 DevOps 基础功能
ks-jenkins-agent	jenkins agent 连接 jenkins master 所使用的服务
ks-sonarqube-postgresql	代码质量分析组件 SonarQube 的后端数据库
ks-sonarqube-sonarqube	SonarQube 的主服务
s2ioperator	s2i 控制器, s2i 的全声明周期管理
uc-jenkins-update-center	jenkins 更新中心, 提供 Jenkins 插件的安装包
webhook-server-service	为 s2i 提供默认值和验证 webhook

# 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 listing four hosts:

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-wo83nj02 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：

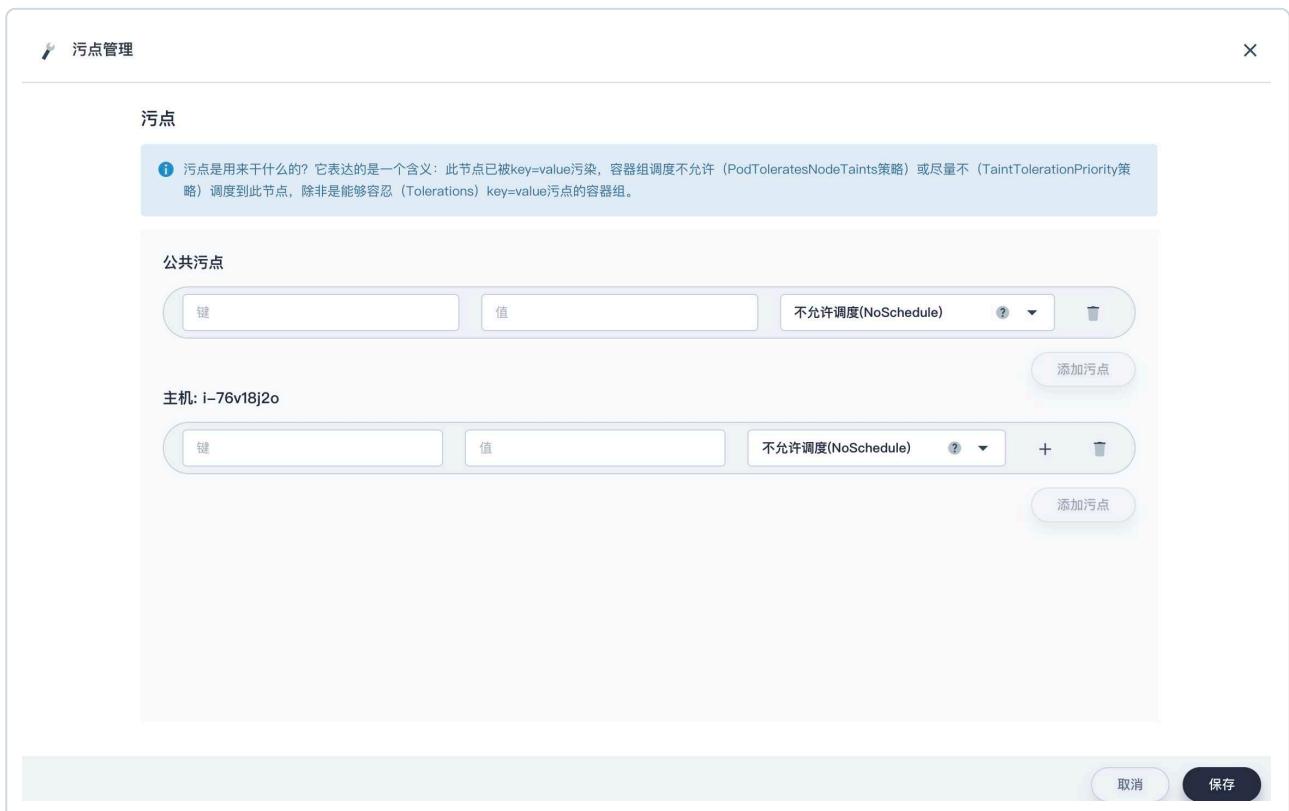
- NoSchedule: 表示不允许调度，已调度的资源不受影响。
- PreferNoSchedule: 表示尽量不调度。
- NoExecute: 表示不允许调度

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。



## 如何将日志和监控的 Pod 调度到专用节点

目前，在 KubeSphere 中已对日志和监控的 Pod 添加了如下的 toleration，若希望将监控和日志调度到专用节点可以给需要调度到的监控节点和日志节点分别打上与 tolerations 匹配的 taint。

```
# 监控
tolerations:
- effect: NoSchedule
  key: dedicated
  operator: Equal
  value: monitoring

# 日志
tolerations:
- key: CriticalAddonsOnly
  operator: Exists
- effect: NoSchedule
  key: dedicated
  value: log
```

## 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

**资源状态**

- CPU使用情况: 2% (已使用: 0.09 Core, 总计: 4.00 Core)
- 内存使用情况: 27% (已使用: 2.14 GiB, 总计: 7.80 GiB)
- 容器组使用情况: 38% (已使用: 23, 容量: 60)

**节点状态**

- OutOfDisk**: 原因: KubeletHasSufficientDisk, 消息: kubelet has sufficient disk space available
- MemoryPressure**: 原因: KubeletHasSufficientMemory, 消息: kubelet has sufficient memory available
- DiskPressure**: 原因: KubeletHasNoDiskPressure, 消息: kubelet has no disk pressure
- PIDPressure**: 原因: KubeletHasSufficientPID, 消息: kubelet has sufficient PID available

## 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。

**监控**

选择时间范围: 最近 5 小时

最近 5 分钟	最近 3 小时	最近 3 天
最近 15 分钟	最近 5 小时	最近 7 天
最近 30 分钟	最近 12 小时	最近 10 天
最近 1 小时	最近 1 天	
最近 2 小时	最近 2 天	

自定义时间范围

开始时间: 2018-11-05 14:47:44

结束时间: 2018-11-05 14:47:44

时间段隔: 30 分钟

**CPU利用率**

**内存利用率**

**IOPS** (读: 蓝色, 写: 橙色)

## 停用或启用主机

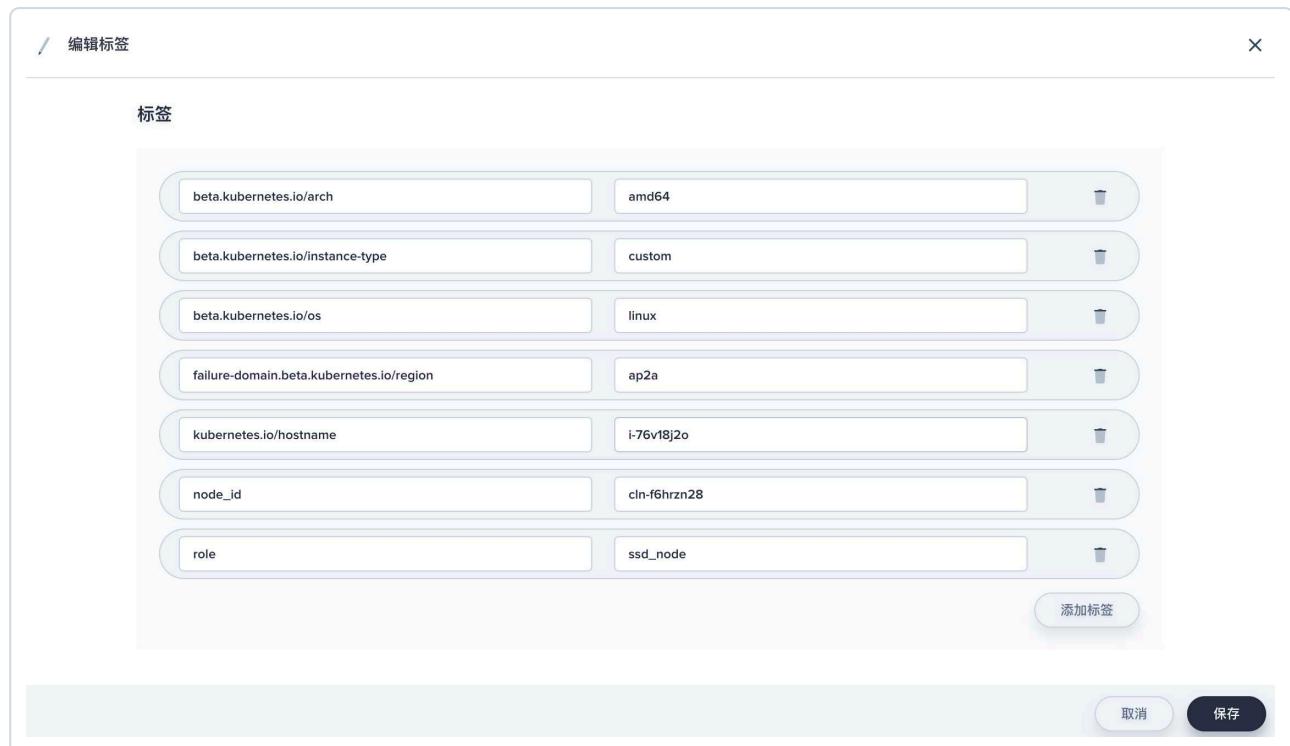
在主机详情页面，点击左侧 **停用** (cordon) 按钮，主机状态变为 **无法调度**，当前按钮变为 **启用** (uncordon)，当有新的工作负载被创建时将不会被调度到此节点，如想回复为可调度状态，点击 **启用** 按钮。

## 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。



# 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

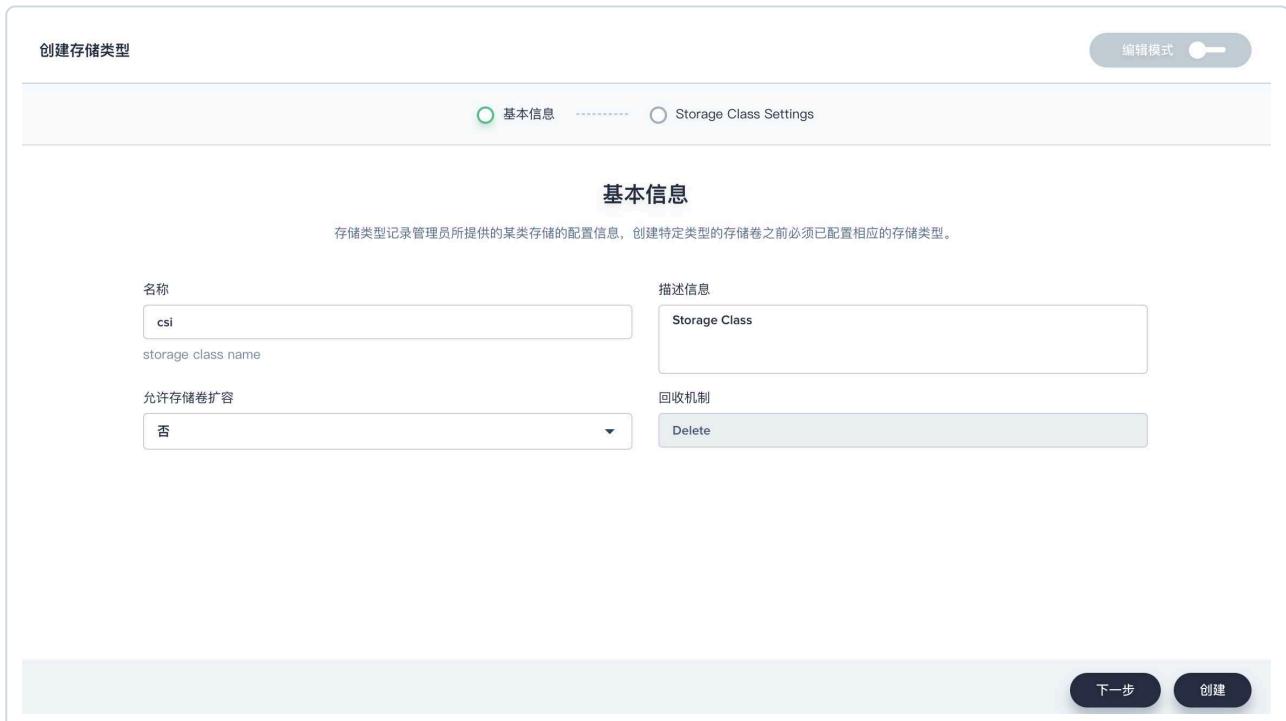
编辑模式

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: 支持的访问模式: ReadWriteOnce, ReadOnlyMany, ReadWriteMany

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

maxSize: minSize: 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。

## 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。



The screenshot shows the KubeSphere interface for managing storage classes. On the left, there's a sidebar with '平台管理', '工作台', and '应用商店'. The main header says 'KUBESPHERE'. Below it, the path is 'infrastructure / storageclasses / csi / volumes'. The left panel is titled 'StorageClasses' and contains a 'Storage Class' section with fields like '查看配置文件', '更多操作', '详情', '供应商: csi-qingcloud', '默认存储类型: False', '可扩展性: 否', and '回收机制: Delete'. A red arrow points to the '更多操作' dropdown menu, specifically to the '设为默认存储类型' option. The right panel is titled '存储卷' and displays the message '暂时没有可用的存储卷'.

## 监控概述

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**的服务组件状态、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，以及应用资源用量、资源用量趋势，监控数据支持选择节点、企业空间或项目按具体指标进行排行。KubeSphere 监控还提供逐级钻取能力，用户可以很方便地查看某个服务组件下工作负载中的 Pod 和容器监控状况，帮助快速定位故障。

## 监控维度

KubeSphere 的监控中心包括 [集群状态监控](#) 和 [应用资源监控](#) 两大监控维度。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`

The screenshot shows the KubeSphere Monitoring Center dashboard. At the top, there are navigation links for Platform Management, Workstation, Application Templates, and a user dropdown for 'admin'. The main header is 'KUBESPHERE' with a logo. Below the header, the title '集群状态监控' (Cluster Status Monitoring) is displayed, along with a sub-header '监控集群的运行状态' (Monitor the running status of the cluster). On the left sidebar, there are sections for Monitoring (Cluster Status, Application Resources), Alerts (Alert Messages, Alert Strategies), and a general 'Monitoring Center' section. The main content area is divided into two main sections: '集群节点状态' (Cluster Node Status) and '组件状态' (Component Status). The '集群节点状态' section shows a summary card with '8/8' nodes online and a chart showing node status over time. The '组件状态' section displays various component健康状况，如KUBESPHERE (7/7), kubernetes (3/3), OPENPITRIX (13/13), Istio (12/12), Monitoring (13/13), Logging (5/5), etcd (green), 管理控制中心 (green), K8S 调度器 (green), and 主机 (green). The bottom section, '集群资源使用情况' (Cluster Resource Usage), includes a summary card for CPU core usage (7.51/56.00) and memory usage (53.71/82.13), followed by a detailed line chart of CPU utilization percentage over time.

This screenshot shows the KubeSphere Monitoring Center dashboard focusing on application resources. The layout is similar to the first one, with navigation links at the top. The main title is '监控中心' (Monitoring Center) and the sub-header is '应用资源' (Application Resources). The left sidebar includes sections for Monitoring, Application Resources, and Alerts. The main content area has three main sections: '集群资源使用情况' (Cluster Resource Usage), '应用资源用量' (Application Resource Usage), and a summary card for '最近1天' (Last 1 Day). The '集群资源使用情况' section shows a line chart for CPU usage (7.51 core CPU) and memory usage (53.71 GiB). The '应用资源用量' section provides a summary of deployment counts (181), stateful set副本集 (16), service accounts (8), tasks (113), scheduled tasks (8), and storage volumes (33). Each summary card includes a small icon and a numerical value.

从上图中不难发现，KubeSphere 平台的监控指标和 IaaS 层相似，有我们常见的 CPU、内存、磁盘和网络等四个方面的使用量和使用率，还包括 Kubernetes 集群的 ETCD、API Server 和 kube-scheduler 的监控。

另外 KubeSphere 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。

# 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 `hpa-example` 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，将演示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，判断该监控图表的趋势是否符合预期。

## 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		
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) at 47/110, and a line graph showing '容器组数量变化' (Container Group Number Change) over time. 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	51.54 m

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。

# 集群状态监控

KubeSphere 监控中心在 **集群状态监控** 提供了集群的 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。在控制台的 **平台管理 → 监控中心** 页面可查看集群状态的监控指标。

## 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 **集群状态**，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。

The screenshot shows the KubeSphere Monitoring Center interface. At the top, there are tabs for Overview, Physical Resource Monitoring, Node Usage Ranking, ETCD Monitoring, and Scheduler Monitoring. The left sidebar has sections for Platform Management, Workstation, Application Templates, and Monitoring Center. Under Monitoring Center, it lists Cluster Status, Application Resources, and Alarms. The main area is divided into three sections: Cluster Node Status, Component Status, and Cluster Resource Usage.

- 集群节点状态:** Shows 8/8 nodes online, with 8 online and 8 total nodes.
- 组件状态:** Displays status for various components:
  - KUBESPHERE: 7/7
  - kubernetes: 3/3
  - OPENPITRIX: 13/13
  - Istio: 11/12
  - Monitoring: 13/13
  - Logging: 7/7
  - etcd: Green checkmark
  - 管理控制中心: Green checkmark
  - KBS 调度器: Green checkmark
  - 主机: Green checkmark
- 集群资源使用情况:** Shows resource usage statistics and a line chart of CPU utilization over time.
  - CPU core: 7.42/56.00
  - 内存 GB: 55.25/82.13
  - 本地存储 GB: 202.45/305.55
  - 容器组: 326/880

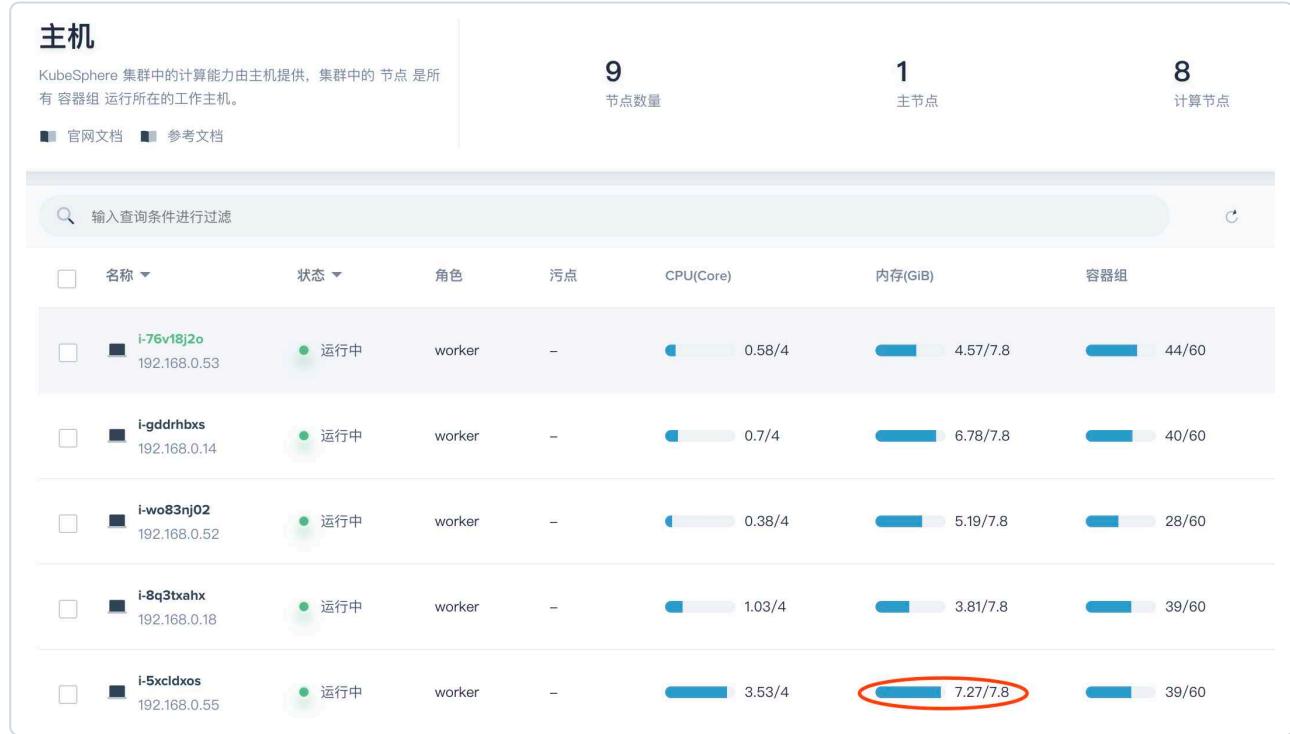
The line chart shows CPU Utilization (%) from 05:32 to 13:27:44, fluctuating between 12% and 18%.

## 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。



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。

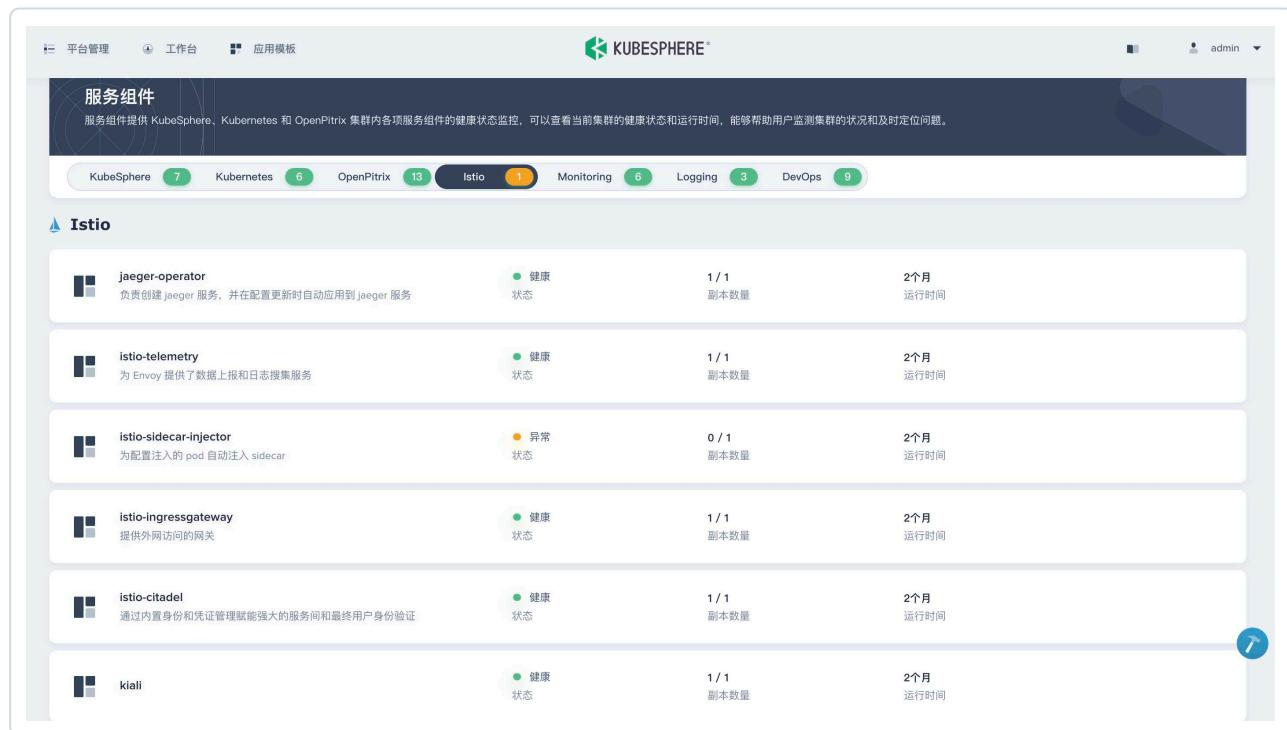
在主机详情页点击 监控 tab，可以查看该主机的 CPU 使用率、CPU 平均负载、内存使用率、磁盘使用率、inode 使用率、IOPS、磁盘吞吐、网卡速率。对于主机而言，除了 CPU、内存、磁盘等使用率、磁盘吞吐这类常用监控指标，inode 使用率和 CPU 平均负载的监控信息对于主机资源管理也是非常有帮助的。

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

KubeSphere 提供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 Istio 有一个组件状态异常，其 Tab 显示黄色的异常状态。



The screenshot shows the KubeSphere Service Components page. At the top, there are tabs for Platform Management, Workstation, Application Templates, and Admin. Below the tabs, a banner states: "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。" A navigation bar below the banner includes links for KubeSphere (7), Kubernetes (6), OpenPitrix (13), Istio (1), Monitoring (6), Logging (3), and DevOps (9). The main content area is titled "Istio" with a warning icon. It lists six service components:

组件	描述	状态	副本数量	运行时间
jaeger-operator	负责创建 jaeger 服务，并在配置更新时自动应用到 jaeger 服务	健康	1 / 1	2个月
istio-telemetry	为 Envoy 提供了数据上报和日志收集服务	健康	1 / 1	2个月
istio-sidecar-injector	为配置注入的 pod 自动注入 sidecar	异常	0 / 1	2个月
istio-ingressgateway	提供外网访问的网关	健康	1 / 1	2个月
istio-citadel	通过内置身份和凭证管理赋能强大的服务间和最终用户身份验证	健康	1 / 1	2个月
kiali		健康	1 / 1	2个月

## 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。



## 物理资源监控

物理资源监控的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 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。



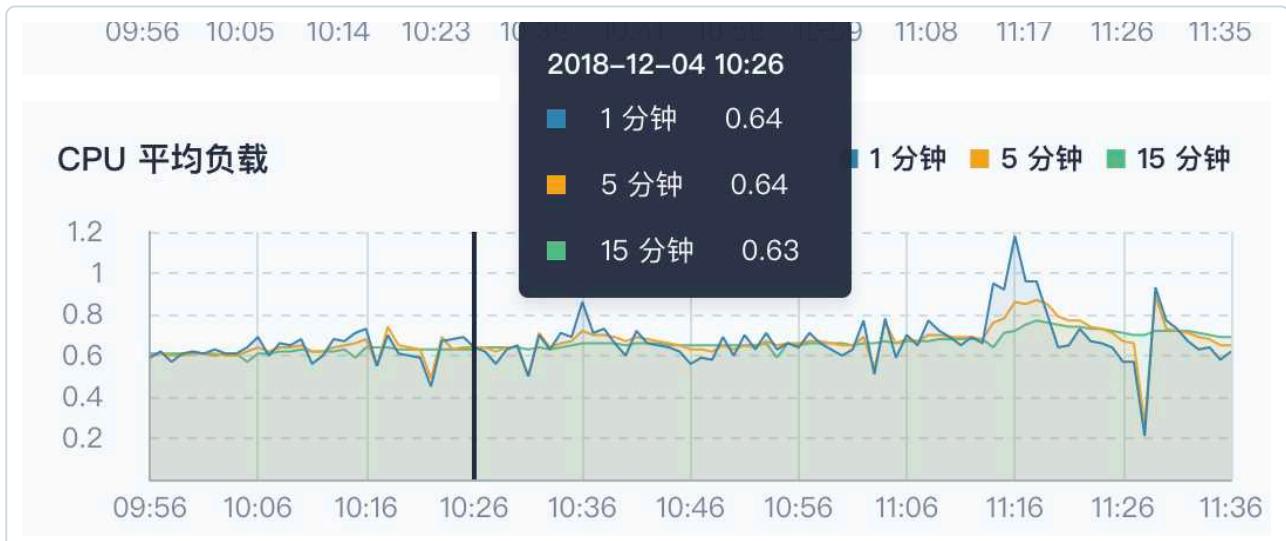
## CPU 平均负载

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都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。

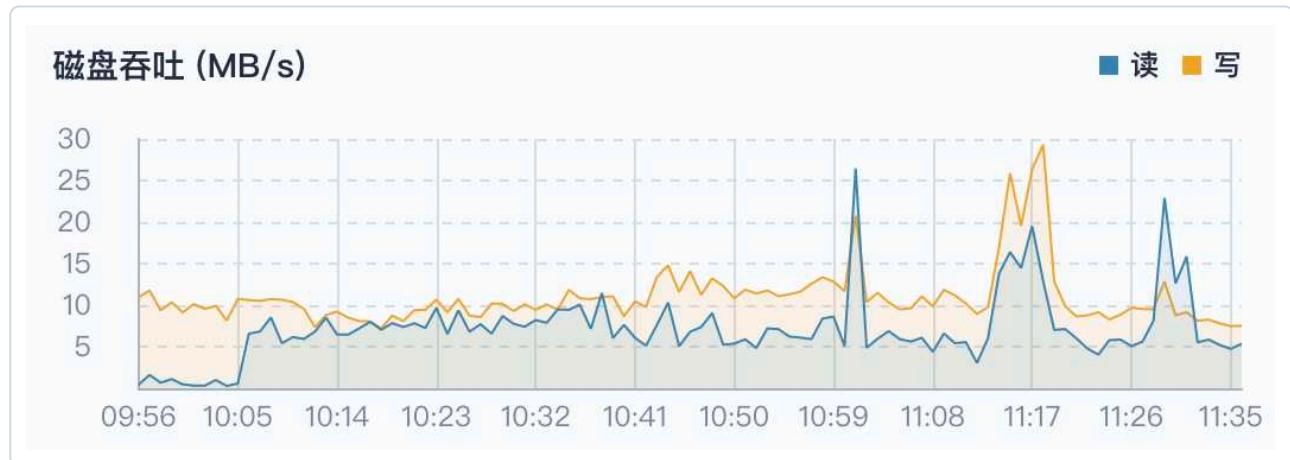
而 inode 使用率的监控恰好就可以预先发现上述提到的这类情况，帮助用户知道集群 inode 的使用情况，防止因 inode 耗尽使得集群无法正常工作，提示用户及时清理临时文件。

inode 使用率 (%)



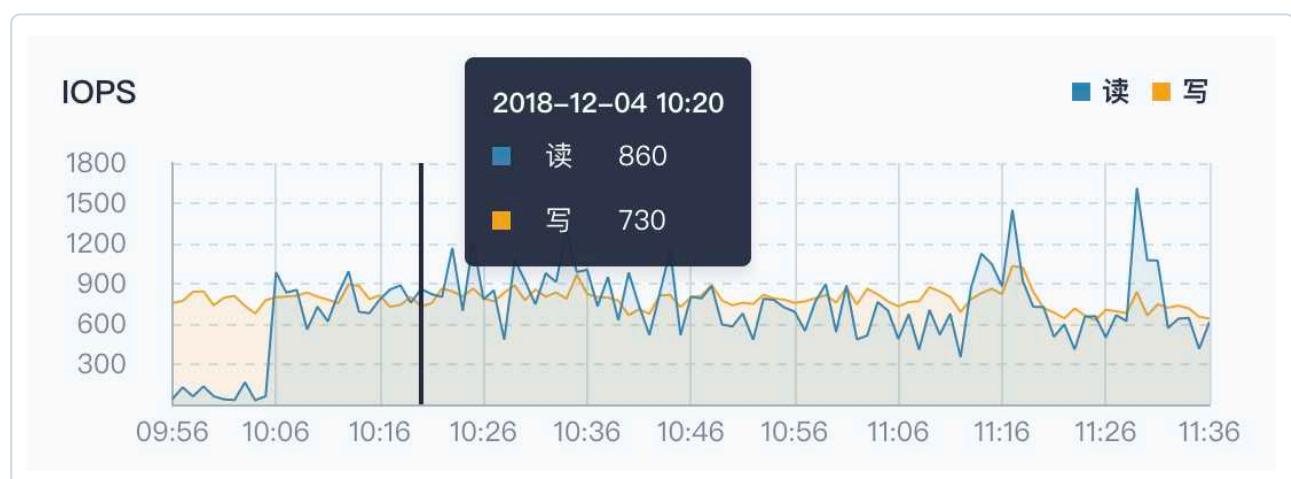
## 磁盘吞吐

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的和。当传输大块不连续数据的数据，该指标有重要参考作用。



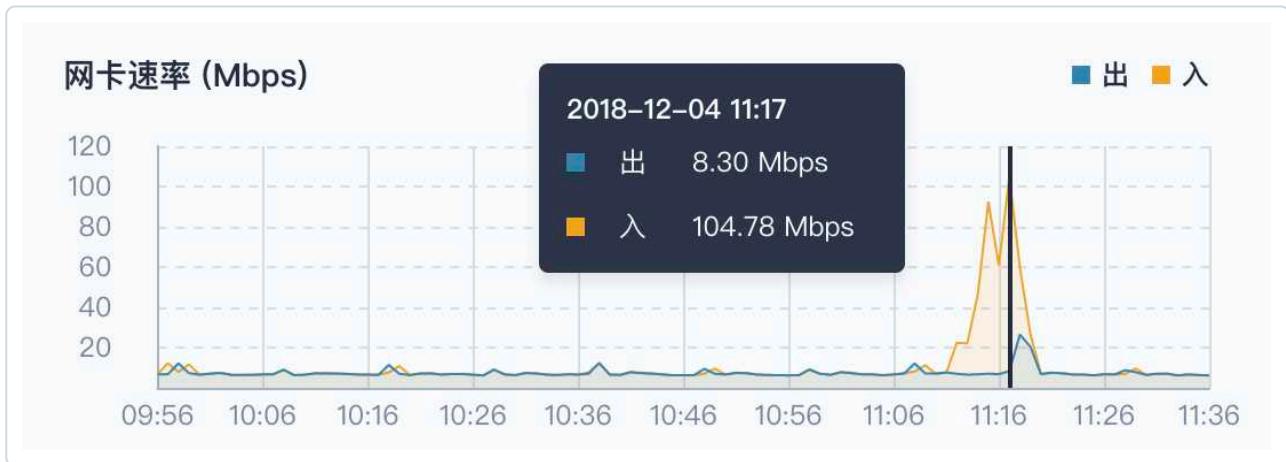
## IOPS

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。已完成状态通常是指任务 (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-5xclidxos 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-gddrhbxz 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	

## ETCD 监控

要想用好 ETCD，特别是定位性能问题，ETCD 的监控必不可少，ETCD 服务原生提供了度量 (metrics) 接口，KubeSphere 监控系统对其原生的数据提供了很好的监控展示效果。

监控指标	指标说明
ETCD 节点	<p><b>是否有 Leader</b>：表示该成员是否有 Leader，如果成员没有 Leader，则完全不可用。如果集群中的所有成员都没有任何 Leader，则整个集群完全不可用。</p> <p><b>Leader 变更次数</b>：用于计算集群中的成员自开始以来所看到的 Leader 变更的数量。频繁的 Leader 变更会显著影响 ETCD 的表现。它还表明 Leader 不稳定，可能是由于网络连接问题或过度负载击中了 ETCD 集群。</p>
库大小	ETCD 底层数据库的大小（以 MiB 为单位），目前展示的是 ETCD 各成员数据库大小平均值。
客户端流量	包括发送给 grpc 客户端的总流量和收到 grpc 客户端的总流量，指标释义详见 <a href="#">etcd Network</a>
gRPC	Server 端的 gRPC 流式消息接收速率和发送速率，该速率可以反映集群中是否有大规模

监控指标	指标说明
流式消息	数据的读和写操作，指标释义详见 <a href="#">go-grpc-prometheus</a>
WAL 日志同步时间	WAL 调用 fsync 的延迟，当 ETCD 在应用它们之前将其日志条目保留到磁盘时，将调用 wal_fsync，指标释义详见 <a href="#">etcd Disk</a>
库同步时间	后端调用的提交延迟分布，当 ETCD 提交其最近更改到磁盘的增量快照时，将调用 backend_commit。注意，磁盘操作高延迟（WAL 日志同步时间或库同步时间较长）通常表示磁盘问题，它可能会导致高请求延迟或使集群不稳定，指标释义详见 <a href="#">etcd Disk</a>
Raft 提议	<ul style="list-style-type: none"> <li>- <b>提议提交速率</b>：记录已提交的共识提案的速率。如果集群健康，该指标应随时间增加。ETCD 集群的几个健康成员可能同时拥有不同的总提议。单个成员与其 Leader 之间的持续大滞后表明成员缓慢或不健康。</li> <li>- <b>提议应用速率</b>：记录所应用的共识提案总速率。ETCD 服务器异步应用每个已提交的提议。提议提交速率和提议应用速率之间的差异通常应该很小（即使在高负载下也只有几千）。</li> </ul> <p>如果它们之间的差异继续上升，则表明 ETCD 服务器过载。在使用大规模查询（如重范围查询或大型 txn 操作）时可能会发生这种情况。</p> <ul style="list-style-type: none"> <li>- <b>提议失败速率</b>：失败提案总速率，通常与两个问题相关，与 Leader 选举相关的临时故障或由于集群中的仲裁丢失导致的更长停机时间。</li> <li>- <b>排队提议数</b>：当前待处理提案的数量，上升的待定提案表明 Client 端负载较高或成员无法提交提案。</li> </ul> <p>目前，界面上展示的数据是 ETCD 各成员指标大小的平均值。提议相关参数释义详见 <a href="#">etcd Server</a>。</p>



## API Server 监控

[API Server](#) 作为 Kubernetes 集群中所有组件交互的枢纽，下表列出了 API Server 监控的主要指标。

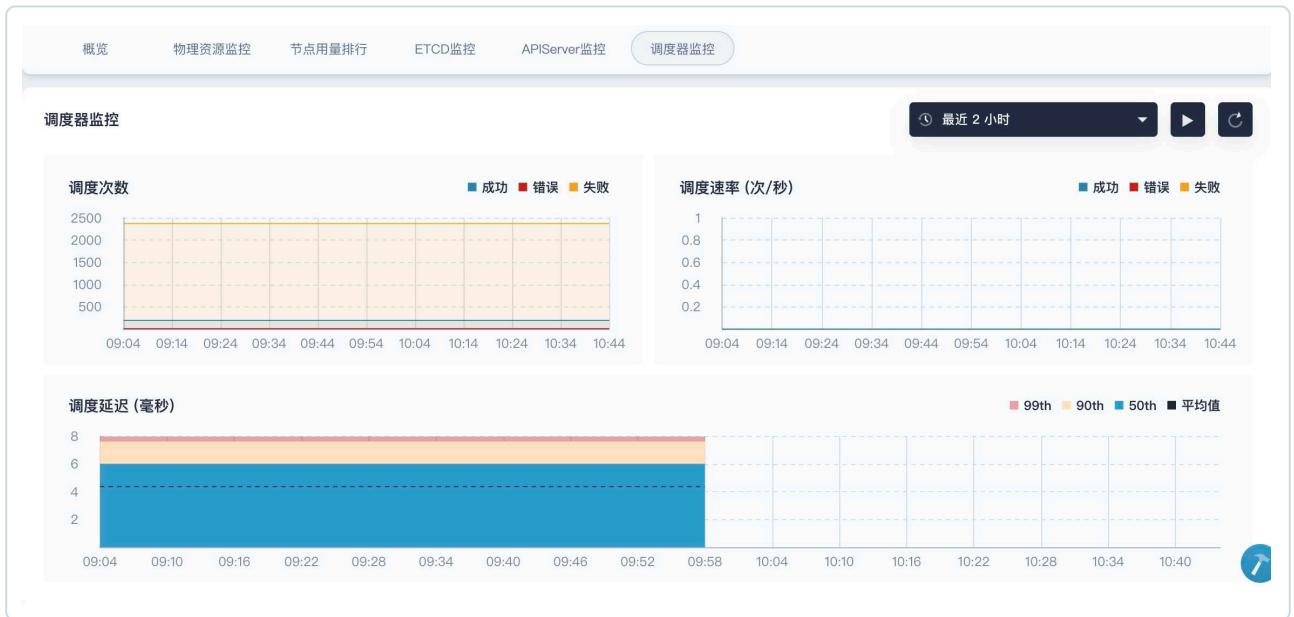
监控指标	指标说明
请求延迟	按 HTTP 请求方法分类统计，资源请求响应的延迟（以毫秒为单位）
每秒请求数	kube-apiserver 每秒接受请求数量



## 调度器监控

[Scheduler](#) 监控新创建的 Pod 的 Kubernetes API，并确定新建的 Pod 应该运行在哪些节点。它根据可用数据做出此决策，包括收集的资源可用性以及 Pod 的资源需求。监控调度延迟的数据可确保您可以查看调度程序所面临的任何延迟。

监控指标	指标说明
调度次数	包括调度成功、错误、失败的次数
调度速率(次/秒)	包括调度成功、错误、失败的调度速率
调度延迟(毫秒)	端到端调度延迟，它是调度算法延迟和绑定延迟的总和



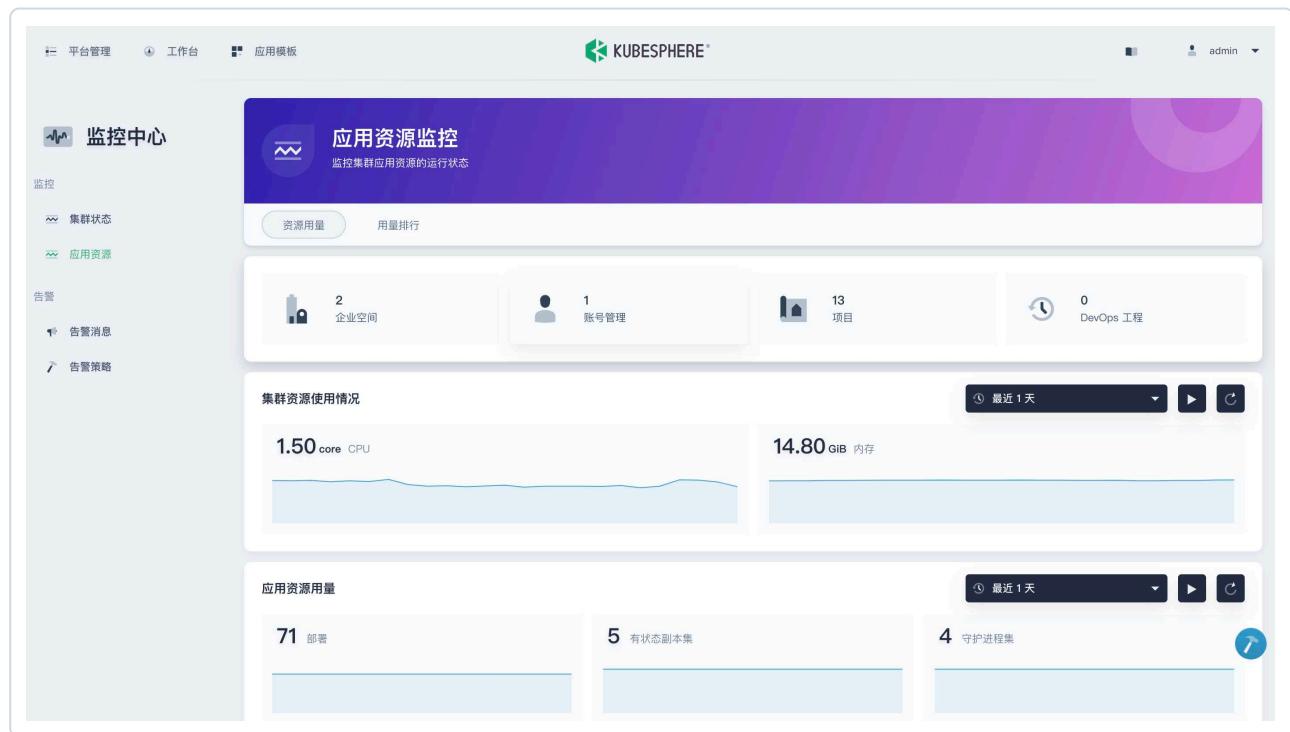
# 应用资源监控

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

选择 **监控中心 → 应用资源**，进入应用资源监控的总览页，包括集群所有资源的使用量汇总情况，如下图所示。

### 集群资源使用情况



### 应用资源用量监控与项目变化趋势

The screenshot shows the KubeSphere Monitoring Center dashboard. On the left, there's a sidebar with navigation links: 平台管理, 工作台, 应用模板, 监控中心 (selected), 集群状态, 应用资源, 告警, 告警消息, and 告警策略. The main area has a title '应用资源用量' and a time range selector '最近 1 天'. Below it is a grid of nine cards showing resource counts:

类别	数量
部署	71
有状态副本集	5
守护进程集	4
任务	30
定时任务	2
存储卷	11
服务	78
应用路由	3
运行中的容器组	91

Below this is a section titled '项目变化趋势' with a line chart showing values over time from 09:12 to 13:12.

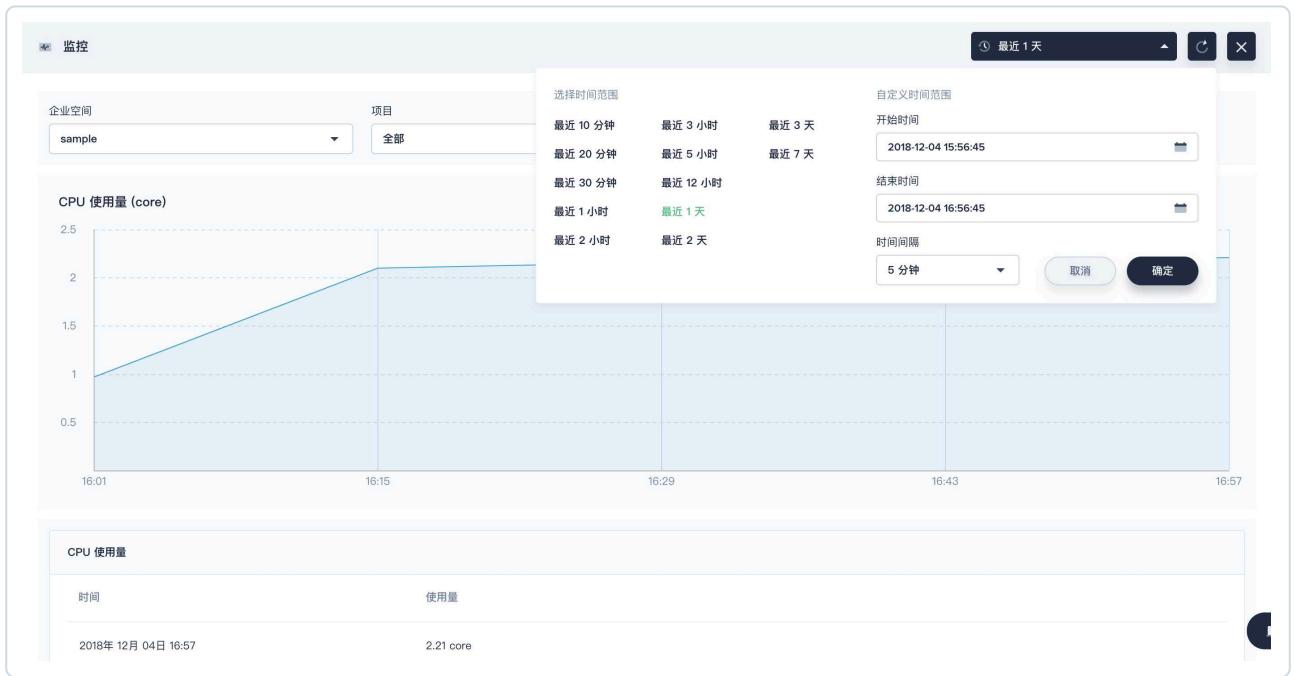
其中，集群资源使用情况和应用资源使用情况保留最近 7 天的监控数据，支持自定义时间范围查询。

This screenshot shows the '集群资源使用情况' section of the monitoring dashboard. It displays the total CPU usage: **7.71 core CPU**. Below this is a bar chart for '应用资源用量' showing 147 部署. To the right is a '选择时间范围' (Select Time Range) dialog with the following options:

最近 10 分钟	最近 3 小时	最近 3 天
最近 20 分钟	最近 5 小时	最近 7 天
最近 30 分钟	最近 12 小时	
最近 1 小时	最近 1 天	
最近 2 小时	最近 2 天	

On the right side of the dialog, there are fields for '自定义时间范围' (Custom Time Range): '开始时间' (Start Time) set to 2018-12-04 15:52:16, '结束时间' (End Time) set to 2018-12-04 16:52:16, and a '时间间隔' (Time Interval) dropdown set to '请选择' (Please Select). At the bottom are '取消' (Cancel) and '确定' (Confirm) buttons.

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
	3.27 core	16.00 GiB	165	94.61 Mbps	63.85 Mbps
	2.24 core	4.76 GiB	51	90.85 Kbps	92.96 Kbps
	1.85 core	5.57 GiB	19	29.39 Kbps	18.23 Kbps
	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				
	项目	CPU 使用量	内存使用量	容器组数量	网络流入速率
	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

# 告警策略 —— 节点级别

告警是 KubeSphere Advanced 2.0.0 的新功能，自研的多租户告警系统支持灵活的告警策略和告警规则，目前 KubeSphere 告警系统具备以下特性：

- 支持基于多租户、多维度的监控指标告警，目前告警策略支持集群管理员对节点级别和租户对工作负载级别等两个层级；
- 灵活的告警策略：可自定义包含多个告警规则的告警策略，并且可以指定通知规则和重复告警的规则；
- 丰富的监控告警指标：提供节点级别和工作负载级别的监控告警指标，包括容器组、CPU、内存、磁盘、网络等多个监控告警指标；
- 灵活的告警规则：可自定义某监控指标的检测周期长度、周期次数、告警等级等；
- 灵活的通知发送规则：可自定义发送通知时间段及通知列表，目前支持邮件通知；
- 灵活的重复告警规则：可自定义重复告警周期、最大重复次数并和告警级别挂钩。

## 目的

本篇文档以创建一个节点级别的告警策略并发送邮件通知作为示例，引导集群管理员如何设置节点级别的告警策略。

## 操作示例

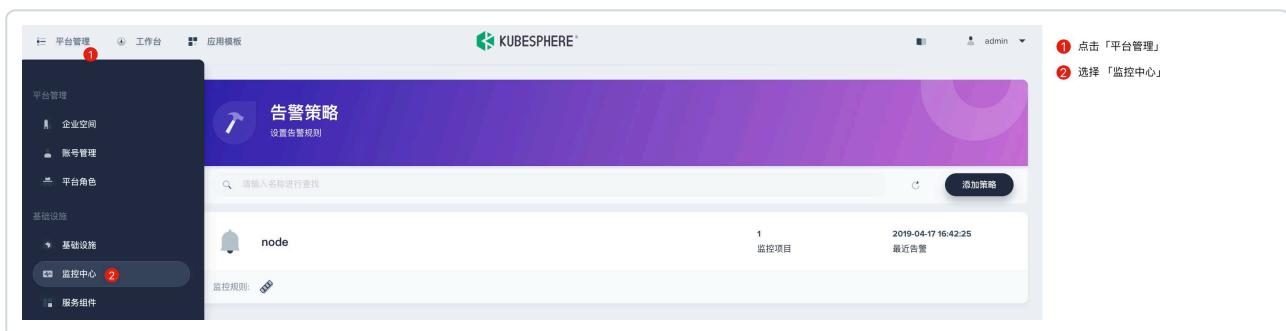
### 前提条件

管理员需预先配置邮件服务器，若还未配置可参考 [邮件服务器](#)。

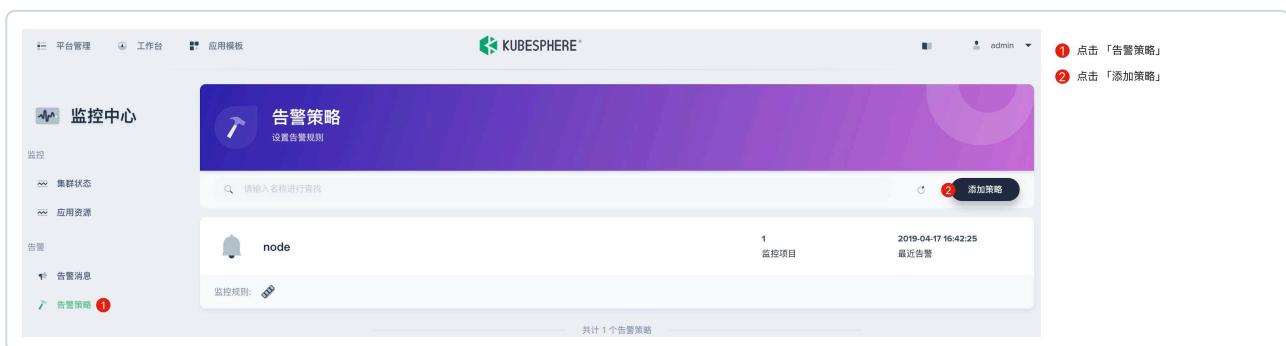
### 演示视频

## 第一步：添加告警策略

1. 以 `cluster-admin` 登录 KubeSphere，点击「平台管理」，选择「监控中心」。



2. 选择「告警策略」，点击「添加策略」。



## 第二步：填写基本信息

在弹窗中，参考如下提示填写基本信息，完成后点击「下一步」。

- 名称：为告警策略起一个简洁明了的名称，便于用户浏览和搜索，比如 `alert-demo`；
- 别名：帮助您更好的区分资源，并支持中文名称，比如 `示例告警`；
- 描述信息：简单介绍该告警策略。

## 第三步：选择监控目标

监控目标支持按三种指标进行排行，这里选择 **按内存使用率排行**，选择排行前三的三台主机，注意这三台主机的内存利用率都超过了 **50%**，为了演示方便，下一步告警规则中可以设置内存使用的阈值为 **> 50%**。

50%。完成后点击「下一步」。

### 监控目标支持按如下三种指标排行：

- 按内存使用率排行
- 按 CPU 使用率排行
- 按容器组用量排行

The screenshot shows the 'Monitoring Targets' section of the KubeSphere interface. At the top, there are tabs for '基本信息' (Basic Information), '监控目标' (Monitoring Targets), '告警规则' (Alert Rules), and '通知规则' (Notification Rules). The '监控目标' tab is selected. Below the tabs, it says '选择告警策略的监控目标' (Select monitoring targets for alerting strategies). There are two tabs at the top of the main area: '集群节点' (Cluster Nodes) and '节点选择器' (Node Selector). A dropdown menu next to the selector is set to '按内存使用率排行' (Sort by memory usage rate). The main list contains three items:

主机 IP	CPU	内存	容器组
i-dzg6r2pr 主机 IP: 192.168.0.10	0.43/8.00 core	9.45/11.73 GiB	28.00/110.00
i-caoijter 主机 IP: 192.168.0.11	0.83/8.00 core	7.42/11.73 GiB	35.00/110.00
i-ydnp0qjj 主机 IP: 192.168.0.12	0.53/8.00 core	6.85/11.73 GiB	Nan/Nan

## 第四步：添加告警规则

告警规则支持的指标、扫描周期、连续次数、告警等级非常丰富，本示例以设置内存利用率作为告警指标，内存使用率的阈值为 **> 50%**，级别为重要告警，设置的规则如截图所示：

The screenshot shows the 'Add Rule' configuration page. At the top, there are tabs for '基本信息' (Basic Information), '监控目标' (Monitoring Targets), '告警规则' (Alert Rules), and '通知规则' (Notification Rules). The '告警规则' tab is selected. The main area has a heading '添加规则' (Add Rule). It includes fields for '规则名称' (Rule Name) with a placeholder '内存利用率' (Memory Utilization), a note about aliasing, and a '规则' (Rule) section with dropdown menus for '指标' (Metric) set to '内存利用率' (Memory Utilization), '周期' (Period) set to '1分钟/周期' (1 minute per cycle), '次数' (Occurrences) set to '连续2次' (2 consecutive times), '操作符' (Operator) set to '>', '阈值' (Threshold) set to '50 %', and '级别' (Level) set to '重要告警' (Important Alert). Below the rule section, there is a note about alert repetition and a link to notification rule settings.

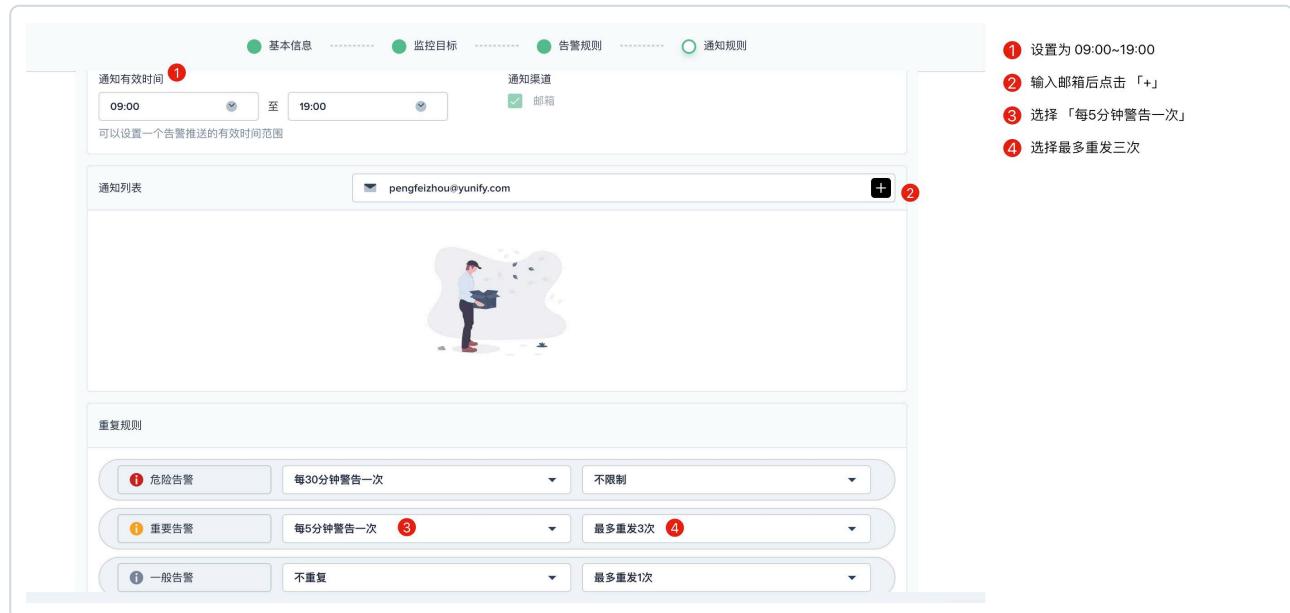
**说明： 节点支持的告警指标如下：**

- CPU：节点CPU利用率、节点CPU1分钟平均负载、节点CPU5分钟平均负载、节点CPU15分钟平均负载；
- 内存：节点可用内存、节点内存利用率；
- 磁盘：节点本地磁盘可用空间、节点本地磁盘空间利用率、节点本地磁盘读取IOPS、节点本地磁盘写入IOPS、本地磁盘读取吞吐量、本地磁盘写入吞吐量、inode利用率；
- 网络：网络发送数据速率、网络接收数据速率；
- 容器组：容器组异常率、容器组利用率。

完成后点击「保存」，然后选择「下一步」。

## 第五步：设置通知规则

1. 通知有效时间可以设置发送通知邮件的时间范围，例如 **09:00 ~ 19:00**，通知渠道目前仅支持邮箱，在通知列表中输入需要通知的成员邮箱。
2. 重复规则设置的是告警通知的发送周期和重发频度，如果告警一直未解决，相隔一定的时间将会重复发送告警。针对不同级别的告警也可以设置不同的重复规则，由于上一步设置的告警级别是重要告警，因此选择重要告警的规则为 **每5分钟警告一次，最多重发3次**。参考如下截图设置通知规则：



3. 点击「创建」，可以看到示例告警策略创建成功。

**说明：告警的等待时间 = 检测周期 × 连续次数。例如检测周期为 1 分钟/周期，连续次数为 2 次，那么需要等待 2 分钟。**

The screenshot shows the KubeSphere monitoring center interface. On the left sidebar, under the '监控' (Monitoring) section, the '告警策略' (Alert Strategy) item is selected. The main content area is titled '告警策略' (Alert Strategy) with a sub-section '设置告警规则' (Set Alert Rules). A search bar at the top says '请输入名称进行查找' (Enter name to search). Below it, there is a table with one row: 'alert-demo(示例告警)' (alert-demo(Demo Alert)), which has a status of '3' (3), a monitor project of '最近告警' (Recent Alert), and a timestamp of '2019-04-17 19:08:11'. At the bottom of the table, it says '监控规则: [编辑]' (Monitor Rule: [Edit]).

## 第六步：查看告警策略

告警策略创建成功后，点击进入 `alert-demo` 告警策略的详情页，查看告警规则当前的状态和详细信息，包括监控目标、通知规则和告警历史等。

The screenshot shows the 'alert-demo' alert strategy detail page. At the top, there is a breadcrumb navigation: '监控中心 / 告警策略 / 告警规则'. Below it, there is a header with tabs: '告警规则' (Alert Rule), '监控目标' (Monitor Target), '通知规则' (Notification Rule), and '告警历史' (Alert History). The '告警规则' tab is active. The main content area is titled '告警规则' (Alert Rule) and contains a table with one row: '内存利用率' (Memory Utilization Rate). This row includes a status indicator '正在告警' (Alerting) with a red heart icon, a notification rule '每 5 分钟通知一次' (Notify once every 5 minutes), and a note '连续2次(分钟/周期) > 50%' (Continuous 2 times (minute/cycle) > 50%).

左侧点击「更多操作」 → 「更改状态」，支持启用或停用告警策略。

# 告警消息 —— 节点级别

告警消息记录了在节点级别的告警策略中，所有已发出的满足告警规则的告警信息。在 [告警策略 —— 节点级别](#) 这篇文档中，演示了创建一个节点级别的告警策略并发送邮件通知，同时所有平台发出的告警消息都已经记录在了告警消息列表中，管理员可以进一步查看告警详情、监控指标、告警策略、最近通知和处理意见等详细信息。

## 前提条件

已创建了告警策略并收到了告警消息，若还未创建请参考 [告警策略 —— 节点级别](#)。

## 查看告警消息

1. 以集群管理员 cluster-admin 账号登录 KubeSphere，点击「平台管理」，选择「监控中心」。
2. 点击「告警消息」，在告警消息列表中查看全部的告警消息。由于我们在告警策略的示例中设置的监控对象为 3 台主机，并且这三台主机在示例中的内存使用率都大于告警的阈值 50%，因此在告警消息列表中看到了 3 条与监控目标对应的告警消息。

告警消息	等级	类型	描述信息	时间
内存利用率	● 危险	主机告警	i-caojnter 内存利用率 > 60%	2019-04-17 17:26:52
内存利用率	● 重要	主机告警	i-ydnp0qij 内存利用率 > 50%	2019-04-17 18:58:09
内存利用率	● 重要	主机告警	i-dzg6r2pr 内存利用率 > 50%	2019-04-17 18:58:10
内存利用率	● 重要	主机告警	i-caojnter 内存利用率 > 50%	2019-04-17 18:58:10

3. 点击其中一条告警消息进入详情页，在告警详情中查看监控目标主机的内存利用率，可以看到在最近一段时间内内存利用率持续高于设定的阈值 50%，因此触发了告警。



## 查看告警策略

切换到「告警策略」 查看本条告警消息对应的告警策略，可以看到主机告警策略的触发规则正是在上一篇告警策略示例中设定的。



## 查看最近通知

点击「最近通知」 即可看到当前的通知人已收到了 3 条告警通知，因为当前监控主机的告警指标内存利用率连续 2 次超过了阈值 50%，通知规则设置为 **每 5 分钟警告一次，最多重发 3 次**。

The screenshot shows the KubeSphere alerting interface. At the top, there are tabs: 'Alert Details', 'Alert Policies', 'Recent Notifications' (which is highlighted in green), and 'Handle Opinions'. Below the tabs, the title 'Recent Notifications' is displayed. A red circle highlights the text 'Current notification 3 times'. A table follows, with columns: Time, Notifier, and Status. One row is shown: '2019-04-17 19:08:09', 'pengfeizhou@yunify.com', and 'Success'.

## 验证邮件通知

登录通知邮箱即可看到 KubeSphere 的邮件服务器给通知人发送的告警消息，如下所示。示例邮箱先后一共收到了 9 封邮件，这是因为告警目标 3 台主机的内存利用率都 **连续 2 次** 超过了阈值 **50%**，并且告警的通知规则设置为 **每 5 分钟警告一次，最多重发 3 次**。

The email subject is '告警消息'. The body contains the following information:  
资源 i-ydnp0qjj 内存利用率 告警 5次  
首次发生时间: 2019-04-17 19:04:08  
最后一次发生时间: 2019-04-17 19:08:08  
最后一次实时值: 58.77%  
\* 此为系统邮件请勿回复

## 添加处理意见

点击「处理意见」可以对当前告警进行处理，添加意见信息。例如，由于当前告警主机的内存利用率高

于阈值，所以我们可以处理意见的窗口中添加一条信息：需要对该主机加污点，不允许新的 Pod 调度。

The screenshot shows the KubeSphere monitoring center interface. On the left, there's a summary card for a memory utilization alert: '内存利用率' (Memory Utilization) with the message 'i-ydhp0qij 内存利用率 > 50%' and a '处理意见' (Handle Suggestion) button. A red arrow points to this button. On the right, there's a detailed view of the alert with tabs for '告警详情' (Alert Details), '告警策略' (Alert Strategy), '最近通知' (Recent Notifications), and '处理意见' (Handle Suggestion). The '处理意见' tab is active. It shows two entries:

- admin 邮箱: admin@kubesphere.io 时间: 2019-04-18 10:04:57  
处理意见: 需要对该主机加污点, 不允许新的 Pod 调度。
- admin 邮箱: admin@kubesphere.io 时间: 2019-04-18 10:02:00  
处理意见: 已处理

# 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。

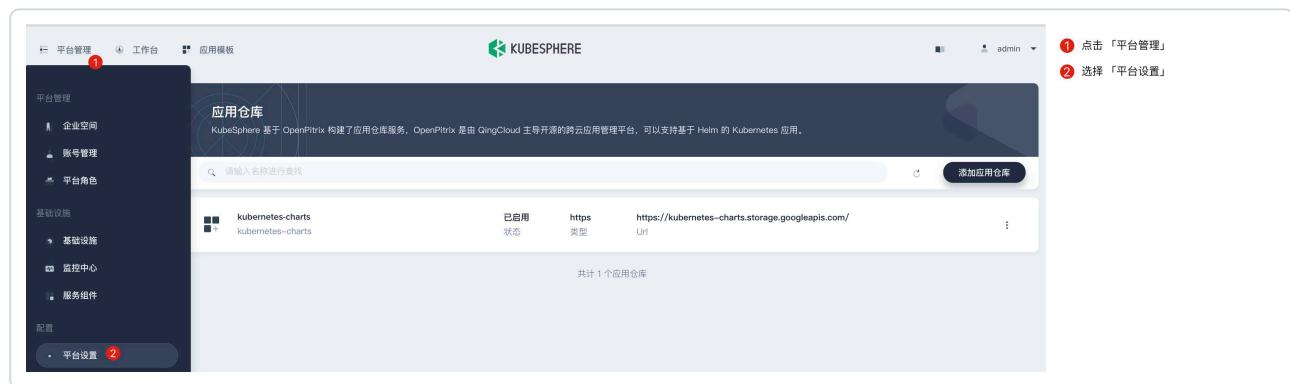
## 准备应用仓库

[Helm 官方文档](#) 已经提供了多种方式创建应用仓库，这里我们仅补充以下两种基于 GitHub 准备应用仓库方便用户使用：

- [KubeSphere 官方应用仓库](#) (极简操作，**推荐！**)
- [基于 GitHub 搭建自己的应用仓库](#)

## 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** → **平台设置**，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://docs-repo.gd2.qingstor.com>，该示例包含一个 Nginx 示例应用，创建后将自动导入到平台中，可在应用模板中进行部署。
  - 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|

取消 确定

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。

## 添加示例应用仓库

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 `cluster-admin` 登录 KubeSphere，选择 [平台管理](#) → [平台设置](#)，进入应用仓库页面，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

平台管理 工作台 应用模板 KUBESPHERE® admin

平台设置

应用仓库

请输入名称进行查找

添加应用仓库

操作	名称	状态	类型	Url	更多
	docs-demo-repo This is a demo	已启用	http	http://docs-repo.gd2.qingstor.com	

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。



# 上传应用到 KubeSphere 官方仓库

KubeSphere 提供了公共应用仓库供开发测试使用，用户可以把做好的应用上传到此仓库，审核后便可以使用。

## 如何上传应用

根据 [Helm 官方指南](#) 编写应用，同时可浏览 Kubesphere 应用仓库已有的应用作为参考：官方应用位于 [src/qingcloud](#) 目录下，测试应用位于 [src/experimental](#) 目录下。

### 第一步：开发应用

1、Fork 官方项目 <https://github.com/kubesphere/helm-charts>。

2、根据 [官方文档](#) 安装 Helm 客户端。

3、打开命令行工具，初始化 Helm 客户端：

```
helm init --client-only
```

4、根据 [Helm 官方文档](#) 创建应用，下面我们创建一个名为 mychart 的示例应用。

5、在命令行工具进入到源代码目录 [src/experimental](#)，并初始化一个应用：

```
cd src/experimental  
helm create mychart  
cd mychart
```

6、此时会看到 Helm 工具帮我们生成了一个基于 Nginx 的应用（以模板文件组织）：

```
src/
├── experimental/
│   └── mychart/
│       ├── Chart.yaml
│       ├── values.yaml
│       ├── templates/
│           ├── Notes.txt      # 用户执行 helm install 安装应用时显示的文字
│           ├── deployment.yaml # 对应 k8s deployment (部署) 的模板文件
│           └── service.yaml   # 对应 k8s service (服务) 的模板文件
│
└── ...

```

7、然后根据实际需要进行修改和开发。

## 第二步：提交应用

开发完成后，先提交到自己 fork 的项目，然后提交 Pull Request 到 [KubeSphere 官方代码仓库](#) 待审核。

## 第三步：等待审核

我们会随时关注所有 PR 并尽快完成审核。如果需要紧急处理，建议通过 [青云 QingCloud 控制台](#) 提交工单在第一时间得到处理。

## 第四步：部署应用

PR 合进主分支后，应用就生效了。如果是第一次部署，参照 [添加应用仓库](#) 把 <https://charts.kubesphere.io/experimental> 添加到 KubeSphere 就可以使用了。

# 基于 GitHub 搭建自有应用仓库

GitHub 是流行的代码仓库，可以直接托管静态文件，利用这个特性可以搭建应用仓库。

## 基于 GitHub 搭建自有应用仓库

根据 [Helm 官方指南](#) 编写应用，同时可浏览 Kubesphere 应用仓库已有的应用作为参考：官方应用位于 [src/qingcloud](#) 目录下，测试应用位于 [src/experimental](#) 目录下。

## 前提条件

创建一个 GitHub 仓库，比如创建一个示例仓库 <https://github.com/FeynmanZhou/helm-repo-example>，所有的应用包都可以存放在此处。

## 第一步：开发应用

1、参考 [官方文档](#) 安装 Helm 客户端。

2、打开命令行工具，初始化 Helm 客户端：

```
helm init --client-only
```

3、根据 [Helm 官方文档](#) 创建应用。下面我们创建一个名为 mychart 的示例应用。

在命令行窗口初始化一个应用：

```
cd helm-repo-example  
helm create mychart
```

此时会看到 Helm 工具帮我们生成了一个基于 Nginx 的应用（以模板文件组织）：

```
| helm-repo-example/
|   └── mychart/
|     ├── Chart.yaml
|     ├── values.yaml
|     ├── templates/
|       ├── Notes.txt      # 用户执行 helm install 安装应用时显示的文字
|       ├── deployment.yaml # 对应 k8s deployment (部署) 的模板文件
|       └── service.yaml   # 对应 k8s service (服务) 的模板文件
|   └── ...

```

4、然后根据实际需要进行修改和开发。

## 第二步：打包应用

1、开发完成后，在目录 helm-repo-example 下把应用打包（准备上传）：

```
helm package mychart
```

2、打包完成后的目录结构如下：

```
| helm-repo-example/
|   └── mychart-0.1.0.tgz      # 可部署的应用包
|   └── mychart/
|     ├── Chart.yaml
|     ├── templates/
|       └── service.yaml    # 对应 k8s service (服务) 的模板文件
|   └── ...

```

## 第三步：打包应用

1、在目录 helm-repo-example 下利用 helm 命令生成索引文件 index.yaml：

```
helm repo index .
```

2、最后的目录结构如下：

```
| helm-repo-example/
|   ├── index.yaml      # 可部署的应用包
|   ├── mychart-0.1.0.tgz    # 可部署的应用包
|   └── mychart/
|       ├── Chart.yaml
|       └── templates/
|           └── service.yaml  # 对应 k8s service (服务) 的模板文件
|
|   └── ...

```

## 第四步：上传应用

把上面产生的所有文件都提交到 GitHub 仓库里，此处默认上传到 master 分支。

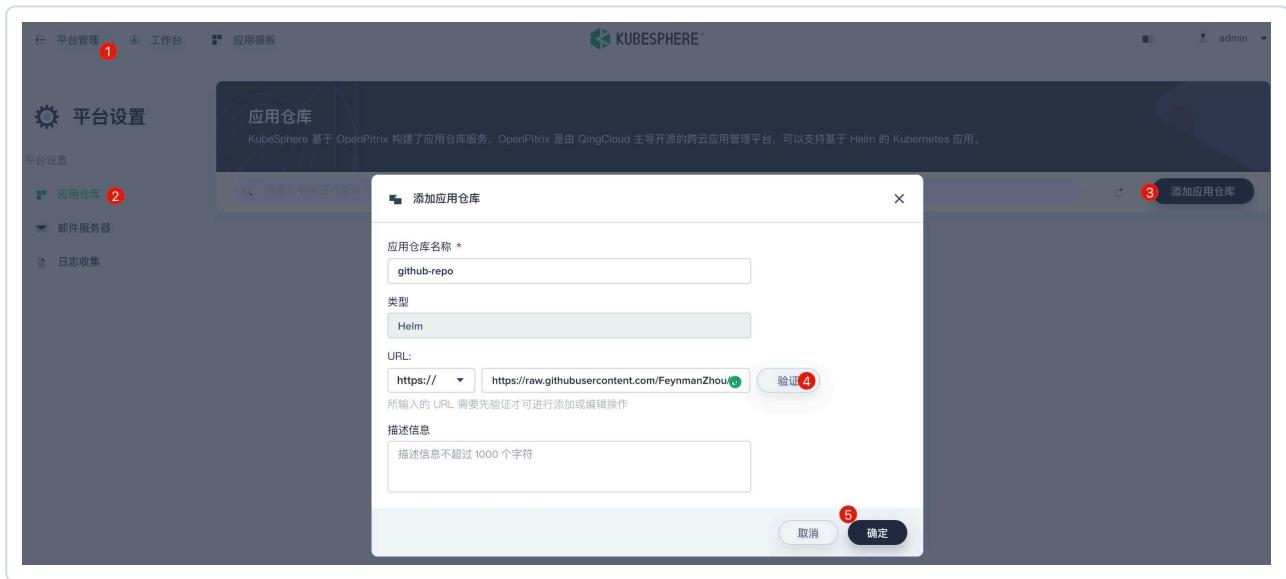
## 第五步：添加应用仓库

如果是第一次部署，把 GitHub 示例仓库 https 地址添加到 KubeSphere 就可以使用了。

以下简单说明如何将该 GitHub 仓库作为应用仓库添加至 KubeSphere，详见 [添加应用仓库](#)。

1、在「平台管理」→「平台设置」的应用仓库下，点击添加应用仓库，输入示例仓库

<https://raw.githubusercontent.com/FeynmanZhou/helm-repo-example/master> 点击验证通过后再点击确定。



2、点击「应用模板」，即可看到该仓库中的应用已成功导入至 KubeSphere，点击进入应用即可一键部署至指定项目中。



3、以下已成功将 mychart 应用部署至项目中，关于如何部署应用可参考 [一键部署应用](#)。



## 自动化上传

上述第二~四步可以自动化，解放手工操作并减少人工错误。可以利用现有的 CICD 工具（如 Jenkins）或开源平台（如 Travis CI）实现。大体流程如下：

```
# 下载代码
git clone https://github.com/FeynmanZhou/helm-repo-example

# 拉取分支
git fetch
git checkout app/mychart

# 初始化
helm init --client-only

# 打包
helm package mychart

# 更新索引
helm repo index .

# 提交
git commit -a -m "[CI Build] Upload App"
git push origin HEAD:app/mychart

# Merge PR (人工或自动)
```

# 邮件服务器

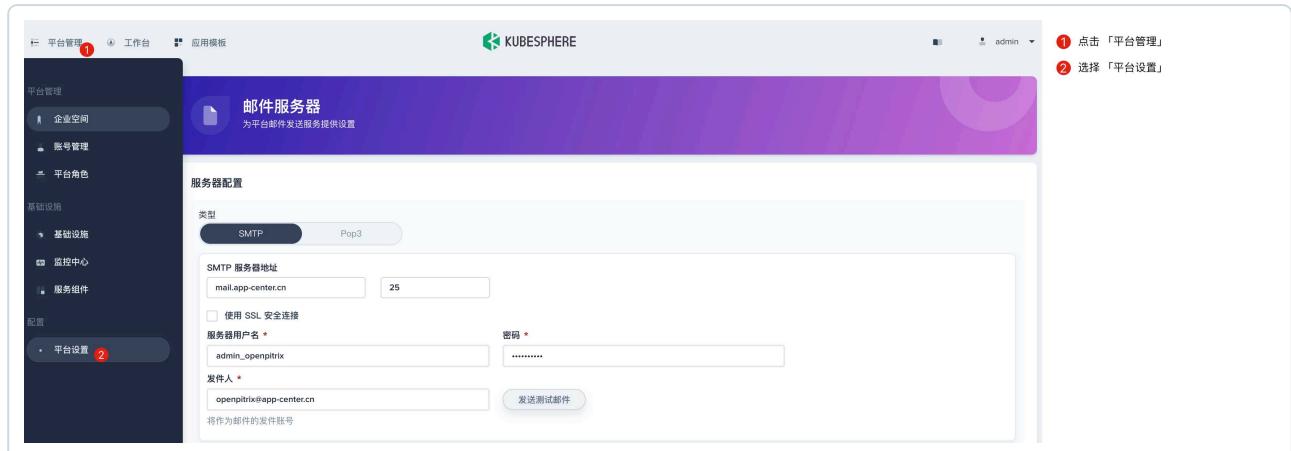
添加告警策略中，目前支持以邮件的方式给用户发送告警消息。大多企业考虑到安全审计的问题，通常会在集群内部部署邮件服务器，目前 KubeSphere 支持添加 SMTP 类型的邮件服务器，SMTP 协议是一个简单的邮件传输协议，利用它您可以将邮件发送给别人。此协议使用命令和应答在客户端与服务器之间传输报文。

以 `cluster-admin` 角色用户登录 KubeSphere，在 **平台管理 → 平台设置** 下选择 **邮件服务器**，

在添加邮件服务器之前，需要用户配置服务器的地址与 SMTP 认证信息。所谓 SMTP 认证，就是要求必须在提供了账户名和密码之后才可以登录 SMTP 服务器。

- SMTP 服务器地址：填写可提供邮件服务的 SMTP 服务器地址，端口通常为 25
- 使用 SSL 安全连接：在邮件服务器中，可使用 SSL 来实现对邮件的加密，从而提高通过用邮件传递信息的安全性。通常需要为邮件服务器配置证书
- 服务器认证信息：填写您邮件服务器的用户名、发件人、密码等信息

以上设置完成后，点击保存即可，可通过发送测试邮件来检验服务器配置是否成功。



# 日志收集

KubeSphere 提供了非常灵活的日志收集的管理配置功能，用户可以添加日志接收者包括 Elasticsearch、Kafka 和 Fluentd，甚至可以暂停（关闭）向某个日志接收者输出日志，也支持稍后重新启用（激活）该日志接收者。

在平台管理中的平台设置页面提供了日志收集配置的选项，KubeSphere 日志查询的日志是默认存在集群内部的 Elasticsearch 中。注意，生产环境强烈推荐配置外部独立的 Elasticsearch 或者 Kafka 集群来存储日志数据，添加了新的外部日志接收者后，日志查询就从这个新的日志接收者查询日志了。

## 添加日志收集者

以 `cluster-admin` 角色用户登录 KubeSphere，点击 平台管理 → 平台设置，选择 日志收集。



点击 **添加日志接收者**，在弹窗中可选择 Elasticsearch、Kafka 或 Fluentd 作为日志收集器。

**提示：在真实环境中，如果用户想输出日志到其他地方，那么可选择将日志输出到 Fluentd，它可以通过众多的 Output 插件转发日志到非常多的地方，比如 S3, Mongodb, Cassandra, Mysql, syslog, Splunk 等等。**



集群将内置的 Elasticsearch 作为默认的日志收集器。注意，生产环境需要配置外部独立的 Elasticsearch 或者 Kafka 集群来存储日志数据。以下分别演示添加 Fluentd 和 Kafka 作为日志收集器：

## 添加 Fluentd 作为日志收集

本文档演示如何在 KubeSphere 部署容器化的 Fluentd 并将其添加为日志收集者，展示 Fluentd 接收到日志数据后，输出到容器的 stdout 标准输出，参考 [添加 Fluentd 作为日志收集](#)。

## 添加 Kafka 作为日志收集

本文档演示如何添加 Kafka 为日志收集者，可以通过 Kafka Consumer 相关命令，验证添加日志收集者的流程是否成功，参考 [添加 Kafka 作为日志收集](#)。

## 日志收集者设置

点击进入 Kafka 的配置详情页，可以更改状态、删除日志收集者或编辑配置文件。



# 添加 Fluentd 作为日志接收者

KubeSphere 目前支持添加的日志接收者包括 Elasticsearch、Kafka 和 Fluentd，本文档演示如何在 KubeSphere 部署容器化的 Fluentd 并将其添加为日志接收者，展示 Fluentd 接收到日志数据后，输出到容器的 stdout 标准输出，最后可以通过查看 Fluentd 容器的日志，验证添加日志接收者的流程是否成功。

## 前提条件

已创建了企业空间、项目，若还未创建请参考 [多租户管理快速入门](#)。这里我们为本演示创建示例项目 `test-fluentd`；

## 第一步：创建配置 (ConfigMap)

1、使用集群管理员账号登录 KubeSphere，进入企业空间下的项目 `test-fluentd` 中，选择「配置中心」→ 「配置」，点击 「创建配置」。



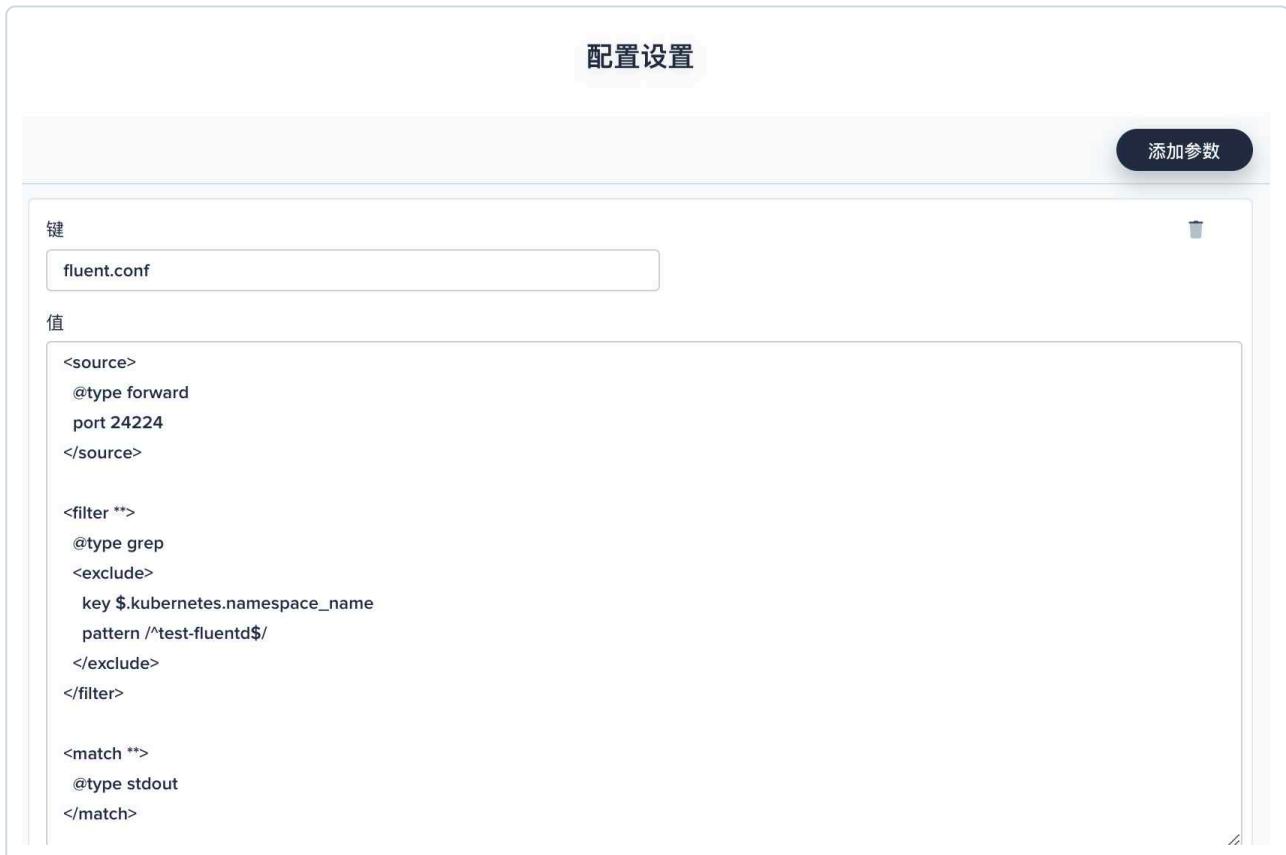
2、在基本信息中名称填写为 `fluentd-config`，点击下一步进行配置设置，键值对填写以下配置，完成后点击 「创建」。

```
# 键  
fluent.conf  
  
# 值  
<source>  
  @type forward  
  port 24224  
</source>  
  
<filter **>  
  @type grep  
  <exclude>  
    key $.kubernetes.namespace_name  
    pattern /^test-fluentd$/  
  </exclude>  
</filter>  
  
<match **>  
  @type stdout  
</match>
```

## 参数释义

- source: Fluentd 在 24224 端口接受数据
- filter: 因为我们要把 Fluentd 接受到的日志输出至 stdout, 为避免 Fluent Bit 与 FluentD 循环采集日志, 这里过滤 Fluentd 所在项目 test-fluentd 下的日志
- match: 输出到标准输出

注意：本示例仅演示输出到标准输出。Fluentd 支持多种第三方接收器转发，比如 S3, Mongodb, Cassandra, MySQL, syslog, Splunk 等。如需配置其他 Fluentd 支持的外部存储收集日志，请参考 [Fluentd 官方文档](#)。



## 第二步：创建部署

- 1、选择「工作负载」→「部署」，点击「创建」。基本信息中名称可自定义，如 fluentd-logging。
- 2、点击「下一步」进入容器组模板，在镜像中输入 fluent/fluentd:v1.4.2-2.0，点击「保存」然后点击「下一步」。
- 3、存储卷设置中，点击「引用配置中心」，然后在配置中选择之前创建的 fluentd-config，挂载路径选择 只读，路径为 /fluentd/etc，然后点击保存。



4、点击「下一步」，标签保留默认值，点击「创建」。



## 第三步：创建服务

1、选择「网络与服务」→「服务」，点击「创建」。在基本信息中名称可自定义，例如 `fluentd-svc`，点击「下一步」。

2、选择第一项 `通过集群内部IP来访问服务 Virtual IP`，点击指定工作负载，选择部署 `fluentd-logging`，端口名称 `fluentd`，默认 TCP 协议的端口和目标端口都设置为 `24224`，点击「下一步」然后点击「创建」。

**◀ 虚拟 IP**

选择器 \*

⚠️ 当前设置的选择器(app=fluentd-logging)共影响到 1 个工作负载 查看

app	fluentd-logging	<input type="button" value="删除"/>
-----	-----------------	-----------------------------------

添加选择器 指定工作负载

端口 \*

fluentd	TCP	24224	24224	<input type="button" value="删除"/>
---------	-----	-------	-------	-----------------------------------

添加端口

会话亲和性

None
------

3、可以看到 **fluentd-svc** 服务创建成功。

项目 test-fluentd

概览 应用 工作负载 存储卷 网络与服务 服务 灰度发布

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

输入查询条件进行过滤

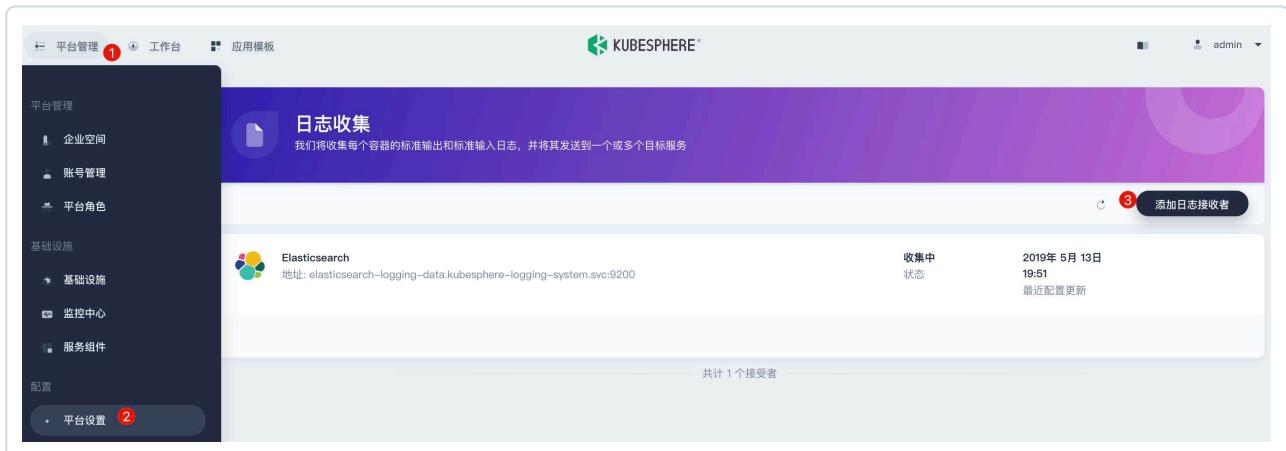
名称	IP地址	端口	应用	创建时间
fluentd-svc	Virtual IP: 10.233.40.154	端口: 24224:24224/TCP 节点端口: -	-	2019-05-14 21:36:51

2 已用配额 ∞ 规划配额

创建

## 第四步：添加 Fluentd 作为日志接收者

1、点击「平台管理」 → 「平台设置」，选择「日志收集」，点击「添加日志接收者」。



2、在弹窗中选择 Fluentd，参考如下填写信息，点击确定保存信息。

- 访问地址：格式参考 {\$Fluentd 服务名}.{\$Fluentd 所在的项目名}.svc,
- 端口号：24224

3、可以看到日志状态显示收集中，说明添加成功。



说明：若需要对 Fluent Bit 转发 Fluentd 做个性化配置，可在日志接收者页面下进入「Fluentd」 → 「更多操作」，点击「编辑配置文件」然后修改 parameters，可参看 [Fluent Bit 官方文档 forward 插件支持的参数项](#)。

/ 编辑配置文件

```

1 type: fluentbit_output
2 name: fluentbit-output-forward
3 parameters:
4   - name: Match
5     value: kube.* ①
6   - name: Port
7     value: '24224' ②
8   - name: Host
9     value: fluentd-svcs.test-fluentd.svc ③
10  - name: Name
11    value: forward ④
12

```

① Fluent Bit 会在收集容器日志时，为数据流打上 kube.\* 标签。这里转发时 Match kube.\* 数据。默认无需修改  
② Fluentd 数据监听端口。默认24224。如需修改，可在 Fluentd 「编辑」页修改或在此修改【value】值  
③ Fluentd 地址。如需修改，可在 Fluentd 「编辑」页修改，或在此修改【value】值  
④ Fluentd 是 Fluent Bit 支持的一种 Output 插件。在 Fluent Bit 中取名为 forward。不可修改

## 第五步：验证日志输出

1、回到项目下的「工作负载」→「部署」，进入 fluentd-logging，然后展开容器组点击进入容器日志。

The screenshot shows the KubeSphere interface for the fluentd-logging deployment. At the top, there's a navigation bar with tabs: 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). Below the navigation bar, there's a summary card for the deployment: 副本运行状态 (Replica Running Status) showing 1/1, 期望副本数 (Desired Replicas) as 1, and 实际运行副本 (Actual Running Replicas) as 1. A note explains what a Deployment is. On the left, there's a sidebar with a 'fluentd-logging' icon, '编辑信息' (Edit Information), and '更多操作' (More Operations). Under '标签' (Labels), it shows 'app: fluentd-logging'. Under '详情' (Details), it lists: 项目 (Project): test-fluentd, 应用 (Application): -, 创建时间 (Created Time): 2019-05-14 21:35:54, 更新时间 (Updated Time): 2019-05-14 21:36:08, and 创建者 (Creator): admin. On the right, under '容器组' (Container Group), there's a search bar and a table for the container named 'fluentd-logging-7f98fc894b-qpsr9'. The table includes columns for 容器组 IP (Container Group IP), 主机 (Host), CPU (CPU usage), and 内存 (Memory usage). A red arrow points to the '容器日志' (Container Log) button for the container, with the text '点击查看容器日志' (View Container Log) above it.

2、在容器日志中，可以看到日志信息的实时数据在动态的输出，即说明 Fluentd 日志收集添加成功。

The screenshot shows a detailed view of a container's status. At the top, there are tabs for '资源状态' (Resource Status), '监控' (Monitoring), '环境变量' (Environment Variables), and '容器日志' (Container Logs), with '容器日志' being the active tab. Below this, a search bar contains the placeholder '请输入关键字进行查找'. The main content area displays '实时数据 (刷新频率 5s)' (Real-time data (refresh rate 5s)) with a table of log entries. Each entry includes timestamp, log content, and labels. The logs show various Kubernetes events like pod creation, deployment updates, and resource limits. A sidebar on the left lists '详情' (Details) and '容器' (Container). A bottom navigation bar includes '终端' (Terminal), '日志' (Logs), '配置' (Configuration), and '历史' (History).

## 添加 Kafka 作为日志接收者

KubeSphere 目前支持添加的日志接收者包括 Elasticsearch、Kafka 和 Fluentd，本文档通过以下两种方式说明如何创建 Kafka 和 Zookeeper 集群，并通过 KubeSphere 添加日志接收者将日志输出到 Kafka 的 topic，最终通过 [Kafkacat 客户端](#) 验证接收实时的日志消息：

- 在 Kubernetes 部署一个单节点的 Kafka 和 Zookeeper (适用于测试或演示)
- 在 AppCenter 创建 Kafka 和 Zookeeper 集群 (适用于正式环境)

### Kubernetes 部署单节点 Kafka 和 Zookeeper

#### 第一步：通过 yaml 创建 Zookeeper、Kafka

**注意：单节点 Kafka 和 Zookeeper 仅用于测试日志输出到 Kafka，正式环境建议搭建多节点的 Kafka 集群。**

1、在工具箱打开 web kubectl 或后台 SSH 到 KubeSphere 后台，新建一个 kafka 的 Namespace，然后在该 Namespace 中创建 Zookeeper 的 Service 和 Deployment。

`zookeeper-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: zookeeper-service
    name: zookeeper-service
spec:
  type: NodePort
  ports:
  - name: zookeeper-port
    port: 2181
    nodePort: 30181
    targetPort: 2181
  selector:
    app: zookeeper
```

**zookeeper.yaml**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: zookeeper
  name: zookeeper
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: zookeeper
    spec:
      containers:
        - image: wurstmeister/zookeeper
          imagePullPolicy: IfNotPresent
          name: zookeeper
          ports:
            - containerPort: 2181
```

## 创建命令

```
$ kubectl create ns kafka
$ kubectl create -f zookeeper-service.yaml,zookeeper.yaml -n kafka
```

2、创建 Kafka 的 Deployment 和 Service，其中 192.168.0.16 请替换为您实际的 IP 地址：

**kafka.yaml**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: kafka
    name: kafka
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: kafka
    spec:
      containers:
        - env:
            - name: KAFKA_ADVERTISED_HOST_NAME
              value: "192.168.0.16"
            - name: KAFKA_ADVERTISED_PORT
              value: "30092"
            - name: KAFKA_BROKER_ID
              value: "1"
            - name: KAFKA_ZOOKEEPER_CONNECT
              value: 192.168.0.16:30181
            - name: KAFKA_CREATE_TOPICS
              value: "test:1:1"
        image: wurstmeister/kafka
        imagePullPolicy: IfNotPresent
        name: kafka
        ports:
          - containerPort: 9092
```

## kafka-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: kafka-service
    name: kafka-service
spec:
  type: NodePort
  ports:
  - name: kafka-port
    port: 9092
    nodePort: 30092
    targetPort: 9092
  selector:
    app: kafka
```

## 创建命令

```
$ kubectl create -f kafka.yaml,kafka-service.yaml -n kafka
```

3、查看 kafka namespace 中，以上步骤创建的资源。

```
$ kubectl get all -n kafka
...
NAME          TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)      AGE
service/kafka-service  NodePort  10.233.0.194 <none>        9092:30092/TCP  6h41m
service/zookeeper-service  NodePort  10.233.0.234 <none>        2181:30181/TCP  6h44m

NAME          READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/kafka     1/1    1           1          6h41m
deployment.apps/zookeeper 1/1    1           1          6h44m
...
```

4、安装 [Kafkacat 客户端](#)，Kafkacat 是一款开源的 Kafka 调试工具。以下仅演示在 ubuntu 安装 Kafkacat (其它 OS 安装请参考 [Kafkacat](#))：

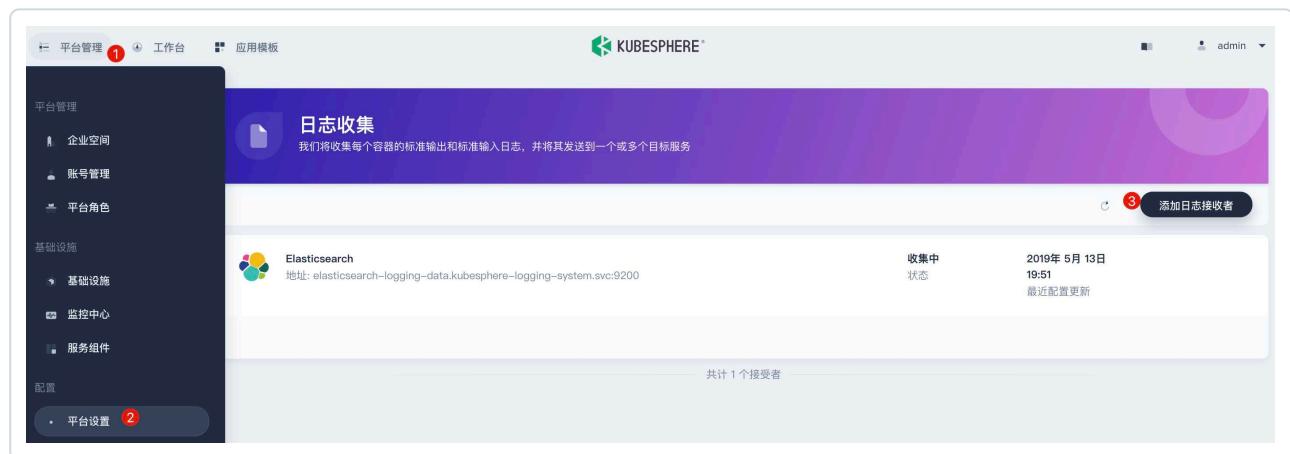
```
$ apt-get install kafka
```

5、使用 Kafkacat 验证单节点 kafka 消息的发送和接收，如下表明发送和接收成功：

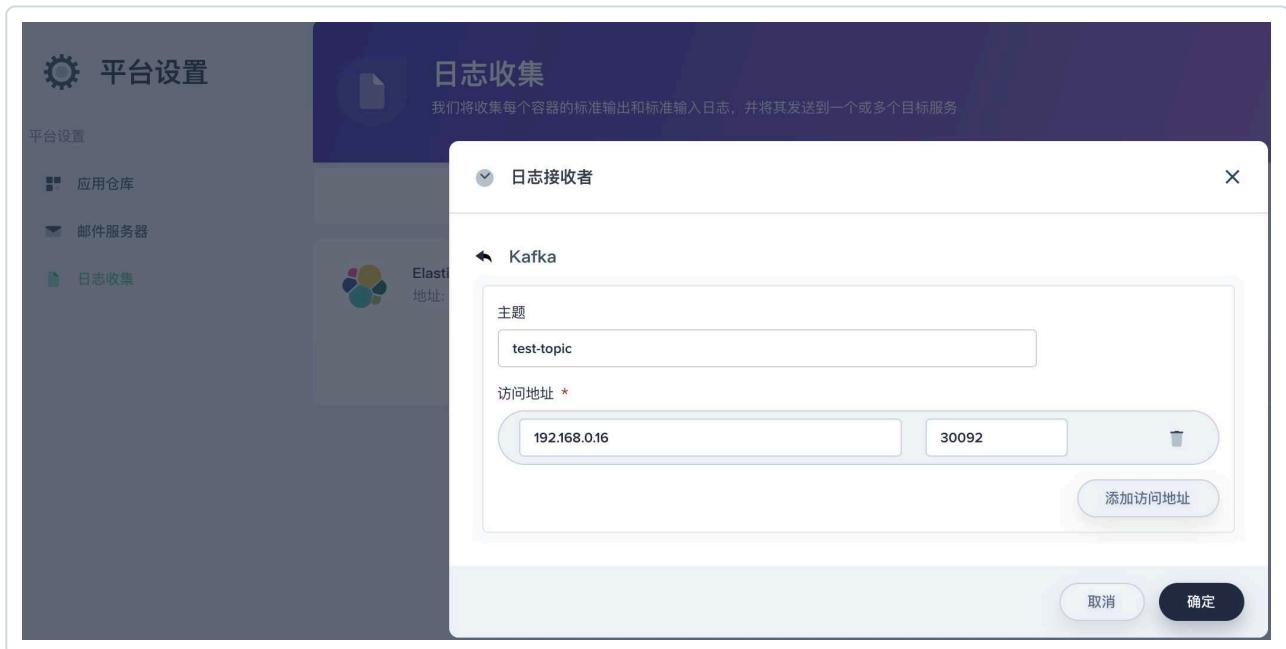
```
$ cho "Hello KubeSphere!" | kafkacat -P -b 192.168.0.16:30092 -t test-topic  
$ kafkacat -C -b 192.168.0.16:30092 -t test-topic  
Hello KubeSphere!
```

## 第二步：添加 Kafka 作为日志接收者

1、点击「平台管理」 → 「平台设置」，选择「日志收集」，点击「添加日志接收者」。



2、在弹窗中选择 Kafka，参考如下填写信息，点击确定保存信息。



说明：若需要对 Fluent Bit 转发 Kafka 做个性化配置，可在日志接收者页面下进入「Kafka」→「更多操作」，点击「编辑配置文件」然后修改 `parameters`，可参看 [Fluent Bit 官方文档 kafka 插件支持的参数项](#)。

/ 编辑配置文件

```
1 type: fluentbit_output
2 name: fluentbit-output-kafka
3 parameters:
4   - name: Match
5     value: kube.* ①
6   - name: Brokers
7     value: '192.168.2.8:19092,192.168.2.9:29092,192.168.2.14:39092' ②
8   - name: Topics
9     value: kafka ③
10  - name: Name
11    value: kafka ④
```

① Fluent Bit 会在收集容器日志时，为数据流打上 `kube.*` 标签。这里转发时 Match `kube.*` 数据。默认无需修改  
② Brokers 是 Kafka 节点 IP/Port 信息，用逗号隔开  
③ Topics 是 Kafka 的消息主题，可自定义  
④ Kafka 是 Fluent Bit 支持的一种 Output 插件，在 Fluent Bit 中取名为 `kafka`，不可修改

### 第三步：验证日志输出

通过 Kafkacat 的消费命令观察日志的动态输出流信息：

```
$ kafkacat -C -b 192.168.0.16:30092 -t test-topic
Hello KubeSphere!
{"@timestamp":1563122754.655668, "log":"2019/07/14 16:45:54 config map updated\n", "time":"2019-07-14T16:45:54Z", "type":"info"}, {"@timestamp":1563122754.659812, "log":"2019/07/14 16:45:54 successfully triggered reload\n", "time":"2019-07-14T16:45:54Z", "type":"info"}, {"@timestamp":1563123117.109699, "log":"2019-07-14 16:51:57.109 [INFO][96] ipsets.go 295: Finished", "time":"2019-07-14T16:51:57.109Z", "type":"info"}, {"@timestamp":1563123117.109884, "log":"2019-07-14 16:51:57.109 [INFO][96] int_dataplane.go 747: F", "time":"2019-07-14T16:51:57.109Z", "type":"info"}, ...
```

## AppCenter 部署 Kafka 和 Zookeeper 集群

### 第一步：创建 Zookeeper + Kafka 集群

除了上述方式，正式环境建议自行搭建一个 Zookeeper 和 Kafka 集群，可通过 [QingCloud AppCenter](#) 完成 Zookeeper 和 Kafka 集群的一键部署，参考文档 [Zookeeper 服务](#) 和 [Kafka 服务](#)。

**说明：**建议将 Kafka、Zookeeper 集群与 KubeSphere 集群置于同一个私有网络环境。若它们处于不同的集群，需要在 broker 所在的路由器上配置端口转发，并且需要修改 broker 的 advertised host 与 advertised port 为路由器转发的源地址和源端口。因为 Kafka 各节点（broker, producer, consumer）之间是靠 advertised host 与 advertised port 通讯的，配置详见 [跨网访问](#)。

1、创建 Zookeeper 集群，Zookeeper 作为 Kafka 的依赖，需要在 Kafka 之前创建。

**基本属性**

- ID: cl-6xxi2ukb
- 状态: 活跃
- 名称: ZooKeeper
- 标签
- 描述
- 应用: ZooKeeper
- 版本: QingCloud 1.3.1 - ZooKeeper 3.4.13
- 节点数量: 1
- 私有网络: (ks-installer)
- 授权提供商: 否
- 创建时间: [redacted]
- 创建于: 8小时前

**节点**

节点	名称	角色	主机类型	节点状态	服务状态	配置	IP	防火墙	告警状态	监控
cln-38f8mwfk	无	无	高性能型	活跃	正常	8核16G 40G	192.168.2.13	无	无	无

\* 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。

**最近6小时** 最近一天 最近两周 最近一个月 最近6个月

**服务模式 (L:LEADER,F:FOLLOWER,S:STANDALONE)**

单位: 间隔: 5m  
监控项: 服务模式 (L:Leader,F:Follower,S:Standalone)

## 2、创建 Kafka 集群。

**基本属性**

- ID: cl-q8y35rpz
- 状态: 活跃
- 名称: Kafka
- 标签
- 描述
- 应用: Kafka
- 版本: Kafka 1.0.0-QingCloud1.1.8
- 节点数量: 4
- 私有网络: (ks-installer)
- 授权提供商: 否
- 创建时间: [redacted]
- 创建于: 8小时前

**节点**

节点	名称	角色	主机类型	节点状态	服务状态	配置	IP	防火墙	告警状态	监控
cln-o85wpc2s	无	客户端节点	性能型	活跃	正常	1核1G 10G	192.168.2.16	无	无	无
cln-v18am5xv	无	Kafka 节点	高性能型	活跃	正常	8核16G 40G	192.168.2.14	无	无	无
cln-oftpoq6n	无	Kafka 节点	高性能型	活跃	正常	8核16G 40G	192.168.2.9	无	无	无
cln-xwy7auro	无	Kafka 节点	高性能型	活跃	正常	8核16G 40G	192.168.2.8	无	无	无

\* 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。

**最近6小时** 最近一天 最近两周 最近一个月 最近6个月

## 第二步：添加日志接收者并验证

同上, 在 KubeSphere 中添加 Kafka 集群作为日志接收者。登录 Kafka 集群客户端节点的 VNC, 可安装使用 Kafkacat 验证 Kafka 集群的消息发送和接收, 或通过 `kafka-console-consumer.sh` 脚本验证, 参考 [Kafka 文档](#)。



```
5.31.10.99981100002 , kubernetes : { pod_name : coredns-77b8449dc9-dg8x7 , namespace_name : kube-system , "host": "i-nsoydb08" , "container_name": "coredns" , "docker_id": "e49d0f9fb8476926131db02e2c1f2200736a20759176239283edf8a63cea2b8e" } }  
{"@timestamp":1558194701.001536, "log": "2019-05-18T15:51:41.000Z [INFO] 10.233.100.198:60604 - 6317 \\"A IN etcd-0.etcd.m3db.svc.cluster.local. udp 52 false 512\" NXDOMAIN qr,rd 145 0.000144378s\n", "time": "2019-05-18T15:51:41.001535599Z", "kubernetes": {"pod_name": "coredns-77b8449dc9-dg8x7", "namespace_name": "kube-system", "host": "i-nsoydb08", "container_name": "coredns", "docker_id": "e49d0f9fb8476926131db02e2c1f2200736a20759176239283edf8a63cea2b8e"} }  
{"@timestamp":1558194701.001993, "log": "2019-05-18T15:51:41.000Z [INFO] 10.233.83.252:45978 - 21522 \\"AAAA IN logs.timber.io.logging.svc.cluster.local. udp 58 false 512\" NXDOMAIN qr,rd 151 0.000106753s\n", "time": "2019-05-18T15:51:41.001992896Z", "kubernetes": {"pod_name": "coredns-77b8449dc9-dg8x7", "namespace_name": "kube-system", "host": "i-nsoydb08", "container_name": "coredns", "docker_id": "e49d0f9fb8476926131db02e2c1f2200736a20759176239283edf8a63cea2b8e"} }  
{"@timestamp":1558194701.003906, "log": "2019-05-18T15:51:41.000Z [INFO] 10.233.68.196:40067 - 29858 \\"A IN logs.timber.io.cluster.local. udp 46 false 512\" NXDOMAIN qr,rd 139 0.00014358s\n", "time": "2019-05-18T15:51:41.003905562Z", "kubernetes": {"pod_name": "coredns-77b8449dc9-dg8x7", "namespace_name": "kube-system", "host": "i-nsoydb08", "container_name": "coredns", "docker_id": "e49d0f9fb8476926131db02e2c1f2200736a20759176239283edf8a63cea2b8e"} }  
[2019-05-18 23:51:49,448] ERROR [Consumer clientId=consumer-1, groupId=console-consumer-16226] Offset commit failed on partition ben-1 at offset 298589: The request timed out. (org.apache.kafka.clients.consumer.internals.ConsumerCoordinator)  
Processed a total of 367 messages  
root@i-gcffkg.jz:/opt/kafka# bin/kafka-console-consumer.sh --bootstrap-server 192.168.0.5:9092,192.168.0.15:9092,192.168.0.9:9092 --topic ben_
```

# Web Kubectl

## 如何使用 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， 目前仅集群管理员 (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。

使用集群管理员登录 KubeSphere， 将鼠标移到右下角的锤子图标展开工具箱列表，选择 `kubectl`，即  
可打开 web kubectl 窗口

The screenshot shows the KubeSphere dashboard for a user named 'admin'. The top navigation bar includes '平台管理', '工作台', '应用模板', and a dropdown for 'admin'. On the left, there's a sidebar with '你好 admin', '超级管理员', '最近登录: 2019-05-13 21:41:47', and sections for '企业空间' (1), '项目' (11), 'DevOps 工程' (0), and '账号管理' (1). The main area displays '集群状态' with a summary: '节点在线状态' (3/3), '在线节点: 3', and '全部节点: 3'. To the right, there's a grid of cluster components: KUBESPHERE (7/7), OPENPITRIX (13/13), Monitoring (4/6), kubernetes (6/6), Istio (10/11), Logging (2/2). A prominent blue '工具箱' (Toolbox) is open at the bottom right, containing three items: 'kubectl' (with a red notification badge '2'), '日志查询' (Log Query), and 'kubeconfig'. A tooltip for 'kubectl' says: '工具箱提供了日志查询以及CLI的操作工具' (The toolbox provides log query and CLI operation tools). A note at the bottom of the toolbox says: 'Shift + 鼠标左键 可以在新窗口中打开' (Shift + Left Click to open in a new window).

在 web kubectl 可输入 kubectl 命令查询或管理 Kubernetes 集群资源。



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						
NAMESPACE	STORAGECLASS	AGE	NAME	STATUS	VOLUME	CAPACITY
kubesphere-devops-system	csi-qingcloud	9h	data-ks-harbor-harbor-redis-0	Bound	pvc-3ee8db56753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	database-data-ks-harbor-harbor-database-0	Bound	pvc-3ee5f3b0753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-gitlab-minio	Bound	pvc-4e1d4874753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-gitlab-postgresql	Bound	pvc-4e1dd46a753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-gitlab-redis	Bound	pvc-4e1e76b0753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-harbor-harbor-chartmuseum	Bound	pvc-3ebdee06753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-harbor-harbor-jobservice	Bound	pvc-3ebe5676753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-harbor-harbor-registry	Bound	pvc-3ebbee533753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-jenkins	Bound	pvc-3ce16bcb753211e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	ks-sonarqube-postgresql	Bound	pvc-ccca339a753111e9	100Gi
kubesphere-devops-system	csi-qingcloud	9h	repo-data-ks-gitlab-gitaly-0	Bound	pvc-4e4f1403753211e9	100Gi
kubesphere-logging-system	csi-qingcloud	9h	data-elasticsearch-logging-data-0	Bound	pvc-740e8b89753211e9	100Gi
kubesphere-logging-system	csi-ainacloud	9h	data-elasticsearch-logging-data-1	Bound	pvc-f7d88763753211e9	100Gi

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。

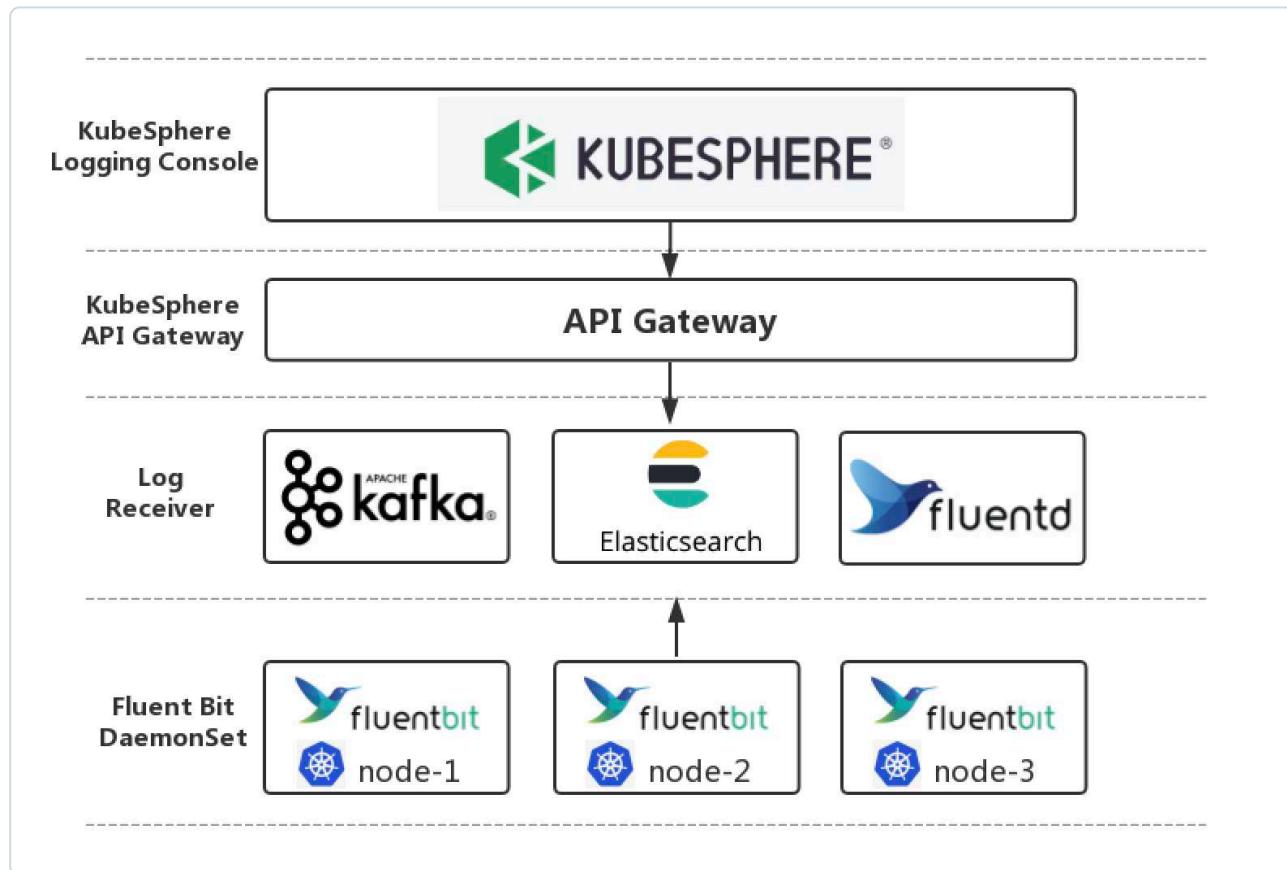
如果需要查询常用命令帮助，可在窗口运行 `kubectl help`，或参阅 [Kubernetes 官方文档](#)。

# 日志查询

应用日志的收集、分析和监控是日常运维工作中非常关键的部分，妥善地处理应用日志收集也是应用容器化的一个重要主题。KubeSphere 提供了非常强大且易用的日志管理功能比如多租户日志管理、多级别日志查询（项目/工作负载/容器组/容器以及关键字）、灵活方便的日志收集配置选项等。在 KubeSphere 的日志查询系统中，不同的租户只能看到属于自己的日志信息，而 Kibana 虽然强大，但是它无法区分不同租户的日志，用户只能在整个集群范围内查看和搜索日志，并且 Kibana 中展现的 Kubernetes 日志包含了很多无关的信息。

## 日志查询演示视频

## 日志系统架构



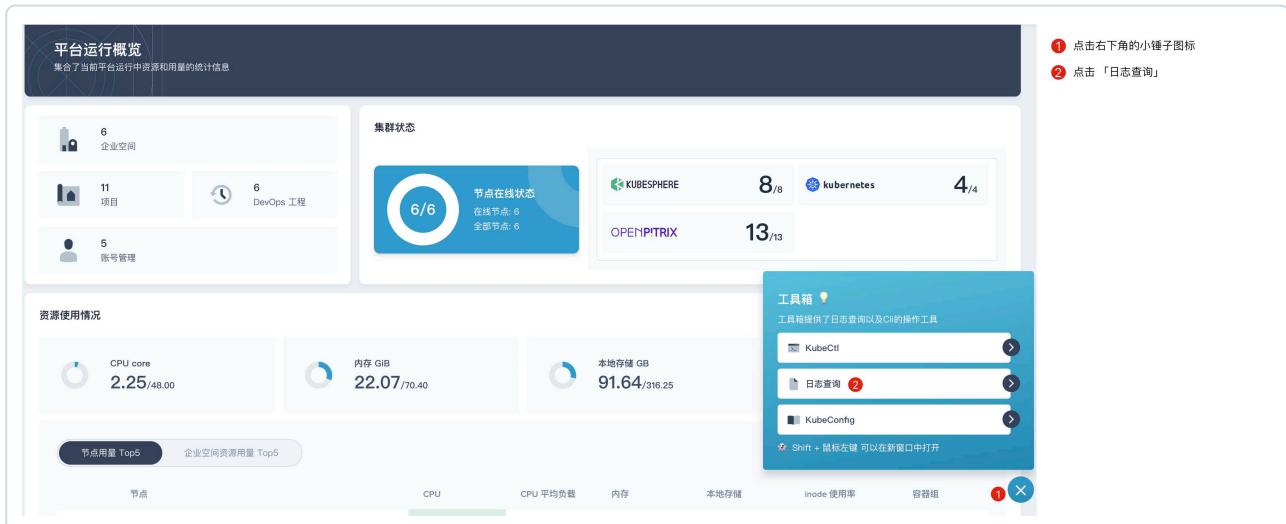
KubeSphere 的日志系统目前是通过 FluentBit Operator 在集群的所有节点上统一部署和配置 Fluent

Bit, 由 Fluent Bit 进行收集所有容器的日志信息, 然后直接传到 Elasticsearch, 集群管理员也可以指定 Kafka 或 Fluentd 等存储、消息队列中。

- FluentBit-operator 以 DaemonSet 方式在每个节点上运行一个实例, 节点物理机的 `/var/log/containers` 目录映射到 FluentBit-operator 容器中。FluentBit-operator 的 Input 插件 tail 主机映射到容器的 log 日志文件, Output 插件依据配置, 将采集到的日志信息发送到 Elasticsearch、Kafka 等存储、消息队列中。
- Elasticsearch 以 StatefulSet 的方式部署在集群中, 输出插件依据日志信息。在 Elasticsearch 创建对应的 Index (默认是一天的日志一个 Index), 为 Kubernetes 日志创建指定格式的 mapping , 并向其中写入日志信息。
- Elasticsearch Curator 以 CronJob 的形式, 定时运行并删除过期的日志信息, 删除方式为删除过期的 Index (一个 Index 存放一天的日志信息), 默认存放日志的时间长度为 1 周, 支持修改其默认存放时间。
- 最终的日志信息可通过 KubeSphere 日志查询的可视化界面, 提供用户对 Elasticsearch 数据的查询、分析、统计等操作。

## 日志查询

KubeSphere 支持所有用户进行日志查询, 登录 KubeSphere 后, 点击右下角的小锤子图标, 选择 日志查询。



在弹窗的日志查询页面，即可看到搜索框和日志总数的变化趋势。目前支持以下几种查询规则，其中关键字查询可通过“Error”“Fail”“Fatal”“Exception”“Warning”等关键字查找错误日志，查询条件支持组合规则查询，并支持精确匹配和模糊匹配。

- 关键字
- 项目名称
- 工作负载名称
- 容器组名称
- 容器名称
- 时间范围

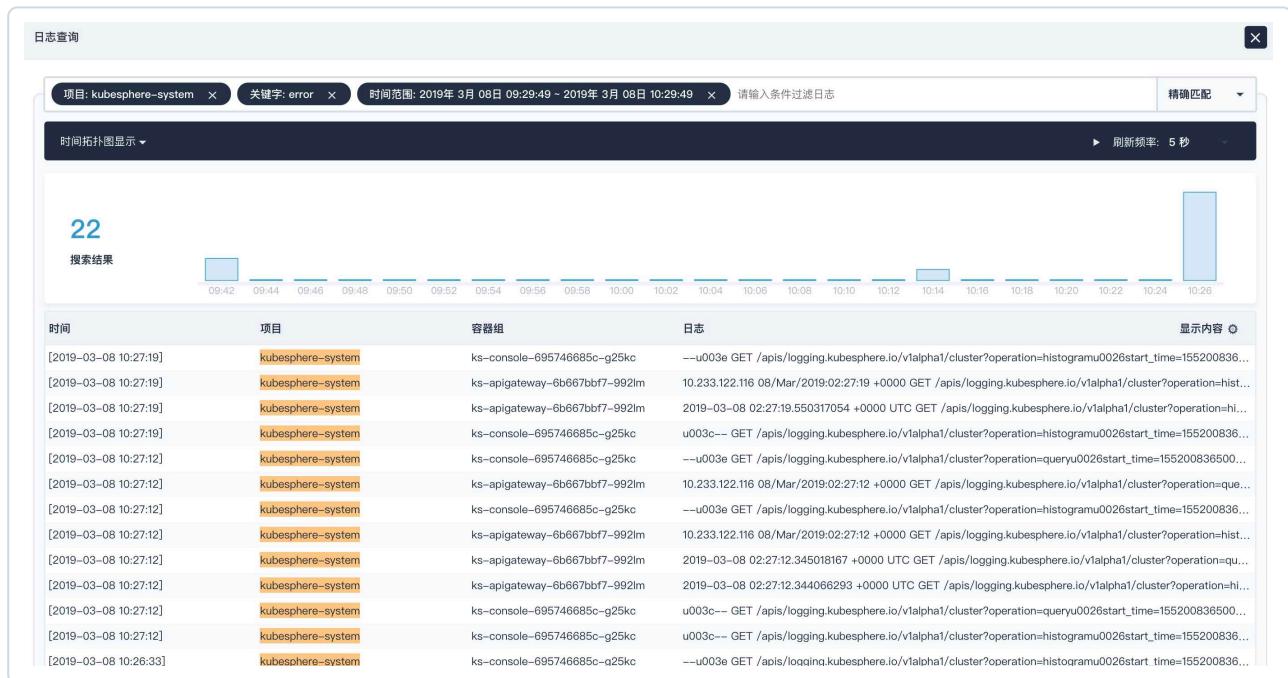


查询支持自定义时间范围，默认可查询近 7 天的日志信息，也可以修改日志保留的时间。



## 日志查询示例

例如，查询在最近 1 小时内，kubesphere-system 项目中包含 error 关键字的日志信息，返回了 22 条搜索结果。



用户可以看到相应日志的收集时间、所属项目、容器组和容器等信息。

点击其中一条日志条目，可查看该项目具体的容器日志信息，包含这条日志相关的上下文信息，用户可以在这个页面查看这条错误发生时所在容器的上下文日志，这里的日志具体信息可以帮助用户快速的定位和分析问题。

**提示：日志查询支持动态加载，刷新频率为 5 秒、10 秒或 15 秒。**

The screenshot shows the KubeSphere log search interface. On the left, there's a sidebar with sections for '项目' (Project), '容器组' (Container Group), and '容器' (Container). The '容器' section is currently selected, showing 'ks-apigateway' as the chosen container. The main area displays log entries from March 8, 2019, at 10:27:19. The logs show several GET requests to the /apis/logging.kubesphere.io/v1alpha1/cluster endpoint, with various error messages related to log queries and intervals.

可以看到在日志查询详情页面左侧有元数据的选项。用户可以在容器一栏选择其他容器，或者同一个容器组中的全部容器；还可以选择同个项目下的其他容器组或所有容器组，这样就可以看到是否其他容器或容器组会对当前错误产生影响了。日志的错误有可能会和 CPU，内存等的用量有关系，比如内存不够。

This screenshot shows the same log search interface as the previous one, but with different parameters. The '项目' dropdown is set to 'kube-system', and the '容器组' dropdown is set to 'csi-qingcloud-controller-0'. The '容器' dropdown is also set to 'csi-qingcloud'. The right panel displays log entries from April 9, 2019, at 13:15:43. Annotations explain that the first red circle indicates the current container group and the second red circle indicates the specific container being viewed.

## 支持一键定位

若通过日志输出信息发现问题后，用户可以一键跳转容器、容器组、项目的详情页，进一步查看监控信息和资源状态。同时，用户可以在容器的详情页进入容器终端定位问题。

- 单击项目也可以转到项目页面查看相应监控信息；
- 点击元信息相应容器/容器组右侧的定位按钮，就可以转到相应的容器/容器组页面。

KubeSphere

平台管理 工作台 应用模板

KUBE SPHERE

admin

监控

ks-console

容器

终端

资源状态 监控 环境变量 容器日志

最近 5 小时

CPU 使用量 (m)

内存使用量 (MIB)

详细

项目: kubesphere-system

应用:

状态: 运行中

镜像: dockerhub.qingcloud.com/kubesphere/ks-console:advanced-dev

镜像 ID: docker-pullable://dockerhub.qingcloud.com/kubesphere/ks-console@sha256:13fe4ddbd47978ec3df4d1a db0e05d7ea31eb2246108788b07f 04e8d509f9254

端口:

命令:

资源请求:

资源限制:

镜像拉取策略: Always

重启次数(总计): 0

## 系统配置修改

### 如何修改 CoreDNS 配置

一、通过 CoreDNS 的 hosts 插件配置 KubeSphere 集群的 DNS 服务，使集群内部可通过 hostname 域名访问外部服务。

1、登陆集群控制节点，执行命令 `kubectl edit configmap coredns -n kube-system -o yaml`，该命令可编辑 coredns 的配置文件，编辑 data 字段，如下， hosts 的内容为新增内容。

```
data:  
Corefile: |  
.53 {  
    errors  
    health  
    hosts {  
        192.168.0.2 harbor.devops.kubesphere.local  
        192.168.0.2 gitlab.devops.kubesphere.local  
        fallthrough  
    }  
    kubernetes cluster.local in-addr.arpa ip6.arpa {  
        pods insecure  
        upstream /etc/resolv.conf  
        fallthrough in-addr.arpa ip6.arpa  
    }  
    prometheus :9153  
    proxy . /etc/resolv.conf  
    cache 30  
    loop  
    reload  
    loadbalance  
}
```

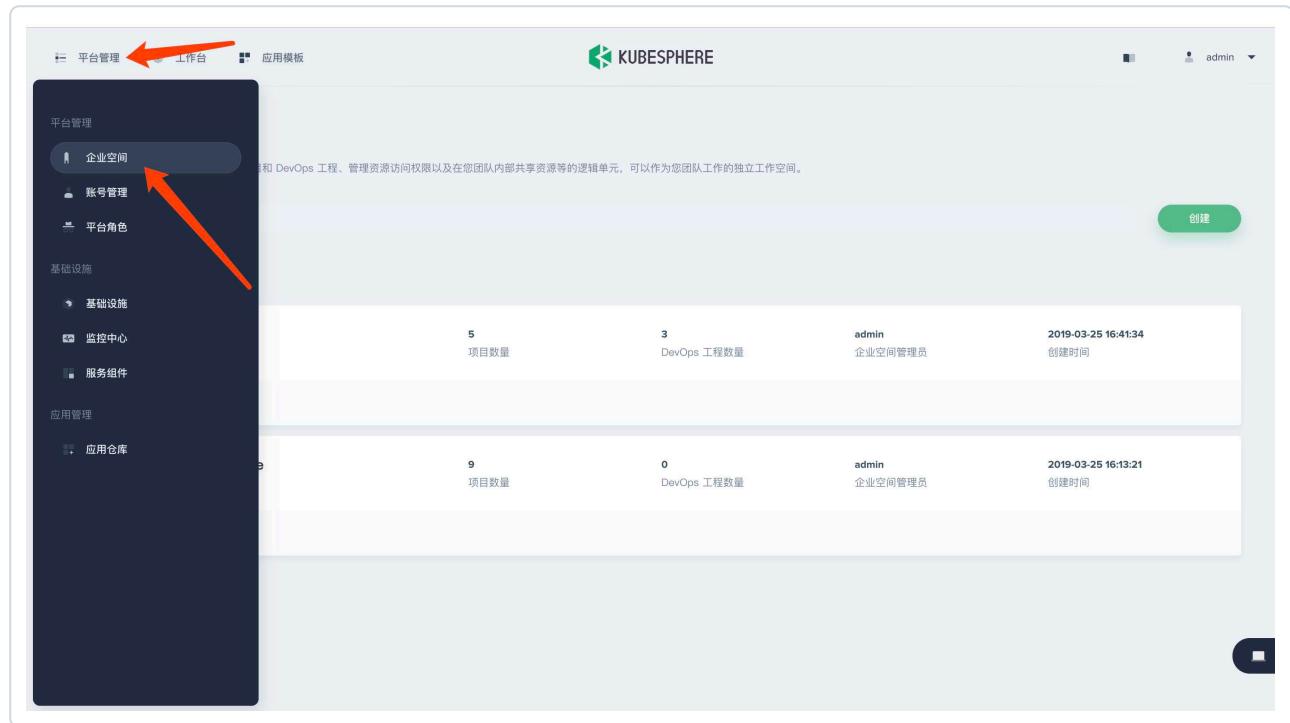
说明：192.168.0.2 是 KubeSphere 集群的内任意节点IP，请根据实际情况填写。

harbor.devops.kubesphere.local 和 gitlab.devops.kubesphere.local 分别为 Harbor 和 GitLab 的域名。

# 如何修改 Jenkins 初始化配置

通过修改 Jenkins 初始化配置，使在执行构建任务的容器中配置--insecure-registry，使 Harbor 能正常推拉镜像。

1、以集群管理员账号 admin 登录 KubeSphere，先点击平台管理，然后进入企业空间。



2、选择 system-workspace，点击进入该企业空间



3、选择 **项目管理**，进入项目 **kubesphere-devops-system**。

The screenshot shows the KubeSphere Project Management interface. On the left sidebar, there is a navigation menu with items like '概览', '项目管理' (highlighted with a red arrow), and '企业空间管理'. The main content area displays a table of projects. One row for 'kubesphere-devops-system' is highlighted with a red arrow, indicating it is the target project. The table columns include '名称', '容器组数量', 'CPU 使用量', '内存使用量', '管理员', and '创建时间'.

名称	容器组数量	CPU 使用量	内存使用量	管理员	创建时间
kubesphere-monitoring-system	7	48.43 m	924.09 MiB	admin	2019-03-25 16:15:16
<b>kubesphere-devops-system</b>	28	0.11 core	6.76 GiB	admin	2019-03-25 14:53:12
kubesphere-logging-system	7	0.20 core	4.93 GiB	-	2019-03-25 14:53:12
kubesphere-system	6	11.85 m	981.41 MiB	admin	2019-03-25 14:53:11
openpitrix-system	13	17.55 m	1.00 GiB	admin	2019-03-25 14:53:11
kubesphere-controls-system	2	0.09 m	10.80 MiB	admin	2019-03-25 14:53:10
default	0	-	-	admin	2019-03-25 14:18:09

4、点击配置中心下的 **配置**，然后找到名称为 **jenkins-casc-config** 的配置文件，点击进入。

The screenshot shows the Configuration Center interface for the 'kubesphere-devops-system' project. On the left sidebar, under the '配置中心' section, the '配置' item is highlighted with a red arrow. The main content area lists several configuration files. One file, 'jenkins-casc-config', is highlighted with a red arrow and is the target for selection. The table columns include '名称', '内容', and '最后更新时间'.

名称	内容	最后更新时间
ks-harbor-harbor-adminserver	TSECRET,EMAIL_HOST,GODEBUG,POSTGRESQL_DATA BASE,REGISTRY_STORAGE_PROVIDER_NAME,CLAIR_U RL,POSTGRESQL_SSLMODE,RESET,ADMIRAL_URL,CLAI R_DB_HOST,CLAIR_DB_PORT,CLAIR_DB_SSLMODE,CFG _EXPIRATION,CHART_REPOSITORY_URL,IMAGE_STORE _PATH,NOTARY_URL,SELF_REGISTRATION,WITH_NOTA RY,CLAIR_DB_USERNAME,EXT_ENDPOINT,POSTGRESQ L_HOST,PROJECT_CREATION_RESTRICTION,TOKEN_SE RVICE_URL,UA_CLIENTID,AUTH_MODE,CORE_URL,E AIL_FROM,POSTGRESQL_PORT	2019-03-25 16:13:41
ks-harbor-harbor-chartmuseum	ALLOW_OVERWRITE,CACHE_DEPTH,INDEX_LIMIT,MAX_ STORAGE_OBJECTS,PROV_POST_FORM_FIELD_NAME, CACHE_REDIS_ADDR,CHART_POST_FORM_FIELD_NAM E,CHART_URL_DISABLE_APILOG_JSON,CONTEXT_PAT H_PORT,STORAGE_STORAGE_LOCAL_ROOTDIR,AUTH_A NONYMOUS_GET,BASIC_AUTH_USER,CACHE_REDIS_D B,DEBUG,DISABLE_METRICS,DISABLE_STATEFILES,MA X_UPLOAD_SIZE,TLS_CERT,TLS_KEY	2019-03-25 16:13:41
ks-jenkins	apply_config.sh,config.xml,initK8sCredentials.groovy, Mailer.groovy,initRBAC.groovy,jenkins.CLI.xml,jenkins.mo del,JenkinsLocationConfiguration.xml,plugins.xml	2019-03-25 16:13:22
ks-jenkins-tests	run.sh	2019-03-25 16:13:22
<b>jenkins-casc-config</b>	jenkins.yaml	2019-03-25 16:13:21

5、进入配置后，点击 **更多操作** 下面的 **编辑配置文件**。

The screenshot shows the KubeSphere UI for managing Jenkins configurations. On the left, there's a sidebar with a 'jenkins-casc-config' icon, '编辑信息' (Edit Information) button, and a dropdown menu. Below that is a '详情' (Details) section with fields for '项目' (Project: kubesphere-devops-system), '创建时间' (Created Time: 2019-03-25 16:13:21), and '创建者' (Creator: Unknown). On the right, a large panel titled '配置项' (Configuration Items) displays a file named 'jenkins.yaml'. The yaml content includes sections for 'jenkins' and 'clouds' (specifically 'kubernetes'). A red arrow points to the '编辑配置文件' (Edit Configuration File) button at the top of the configuration panel.

```
jenkins:
  mode: EXCLUSIVE
  numExecutors: 5
  scmCheckoutRetryCount: 2

clouds:
  - kubernetes:
      name: "kubernetes"
      serverUrl: "https://kubernetes.default"
      skipTlsVerify: true
      namespace: "kubesphere-devops-system"
      credentialsId: "k8s-service-account"
      jenkinsUrl: "http://ks-jenkins.kubesphere-devops-system:80"
      jenkinsTunnel: "ks-jenkins-agent.kubesphere-devops-system:50000"
      containerCapStr: "100"
      connectTimeout: "60"
      readTimeout: "60"
```

6、该配置文件共有配置4个服务，在 data/jenkins.yaml/jenkins.clouds.kubernetes.templates 下，name 分别为 base、nodejs、maven、go，分别修改其各个 name 为 docker-server 下的 args 参数，将原来的 args: "--insecure-registry harbor.devops.kubesphere:30280" 改为 args: "--insecure-registry harbor.devops.kubesphere.local:30280"。以 go 为示例，修改完后的配置如下(其中标注\*的为修改行):

```
- name: "go"
  namespace: "kubesphere-devops-system"
  label: "go"
  nodeUsageMode: "EXCLUSIVE"
  idleMinutes: 0 # Do not reuse pod.
  containers:
    - name: "go"
      image: "kubesphere/builder-go:advanced-1.0.0"
      command: "cat"
      ttyEnabled: true
      envVars:
        - containerEnvVar:
            key: "DOCKER_HOST"
            value: "tcp://localhost:2375"
    - name: "jnlp"
      image: "jenkins/jnlp-slave:3.27-1"
      args: "${computer.jnlpmac} ${computer.name}"
      resourceRequestCpu: "100m"
      resourceRequestMemory: "32Mi"
    - name: "docker-server"
      image: "docker:18.06.1-ce-dind"
      ttyEnabled: true
      privileged: true
      * args: "--insecure-registry harbor.devops.kubesphere.local:30280"
      envVars:
        - containerEnvVar:
            key: "DOCKER_HOST"
            value: "tcp://localhost:2375"
  workspaceVolume:
    emptyDirWorkspaceVolume:
      memory: false
```

修改更新后，需要登陆 Jenkins 重新加载，具体步骤请参考 [登陆 Jenkins 重新加载](#)。

# 上传镜像至 Harbor

## 前提条件

请确保正确安装了 [内置 Harbor](#)，并准备好了基础镜像 `java:openjdk-8-jre-alpine`。

## 如何上传基础镜像到 Harbor

1、执行以下命令，导入预先准备好的基础镜像 `java:openjdk-8-jre-alpine`。

```
$ docker load < XXX.tar
```

```
$ docker load < java.tar
Loaded image ID: sha256:fdc893b19a147681ee764b2edab6c494d60fe99d83b14b8794bbc040ec7aa7
```

2、对镜像打 tag，在终端执行命令 `docker tag fdc893b19a14`

`harbor.devops.kubesphere.local:30280/library/java:openjdk-8-jre-alpine`，注意：其中 `fdc893b19a14` 为 image ID，根据实际情况进行修改，镜像名称格式为<仓库地址>/<项目名称>/<镜像名称>:<标签>。

3、本地加入 insecure

若改主机为 KubeSphere 集群的节点，则无需配置，请跳过此步骤。若为集群外部主机，请修改 Docker 配置文件 `daemon.json`，在 Linux 上的默认路径为 `/etc/docker/daemon.json`，Windows 上的默认路径为 `%programdata%\docker\config\daemon.json`，在该文件中加入配置

```
"insecure-registries": [
    "harbor.devops.kubesphere.local:30280"
]
```

具体文件参数详情可参考 [Docker Daemon 配置文件](#)。

然后重启 Docker。

4、登陆 Harbor，执行命令 `docker login -u admin -p Harbor12345`

<http://harbor.devops.kubesphere.local:30280>。

```
$ 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

5、推送镜像，执行命令 `docker push harbor.devops.kubesphere.local:30280/library/`

`java:openjdk-8-jre-alpine`

```
$ docker push harbor.devops.kubesphere.local:30280/library/java:openjdk-8-jre-alpine
```

The push refers to repository [harbor.devops.kubesphere.local:30280/library/java-openjdk-8]

20dd87a4c2ab: Pushed

78075328e0da: Pushed

9f8566ee5135: Pushed

v1: digest: sha256:955dbe76c31f802d537d0c5e4160b3a010091e7e8323f46ecbb2a0f2174a5e

6、登陆 Harbor 查看到推送的镜像，即完成镜像推送。

The screenshot shows the Harbor UI for managing Docker images. The top navigation bar has '项目< 镜像仓库' and the repository name 'library/java'. Below the navigation, there are tabs for '描述信息' and '镜像', with '镜像' being active. There are several buttons at the top: '扫描' (Scan), '复制摘要' (Copy Summary), '添加标签' (Add Label), '复制镜像' (Copy Image), and '删除' (Delete). To the right are search and refresh icons. A toolbar below the buttons includes filters for '标签' (Label), '大小' (Size), 'Pull命令' (Pull Command), '漏洞' (Vulnerability), '已签名' (Signed), '作者' (Author), '创建时间' (Created Time), and '标签' (Label). The main content area displays a single image entry: 'openjdk-8-jre...' with a size of '40.46MB'. It shows a '扫描' (Scan) button, a status indicator '未扫描' (Not Scanned), and a red 'X' icon. To the right, it says '2017/3/4 上午6:01'. At the bottom right of the table, it says '1-1 共计 1 条记录' (1-1 Total 1 records).

# 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**。

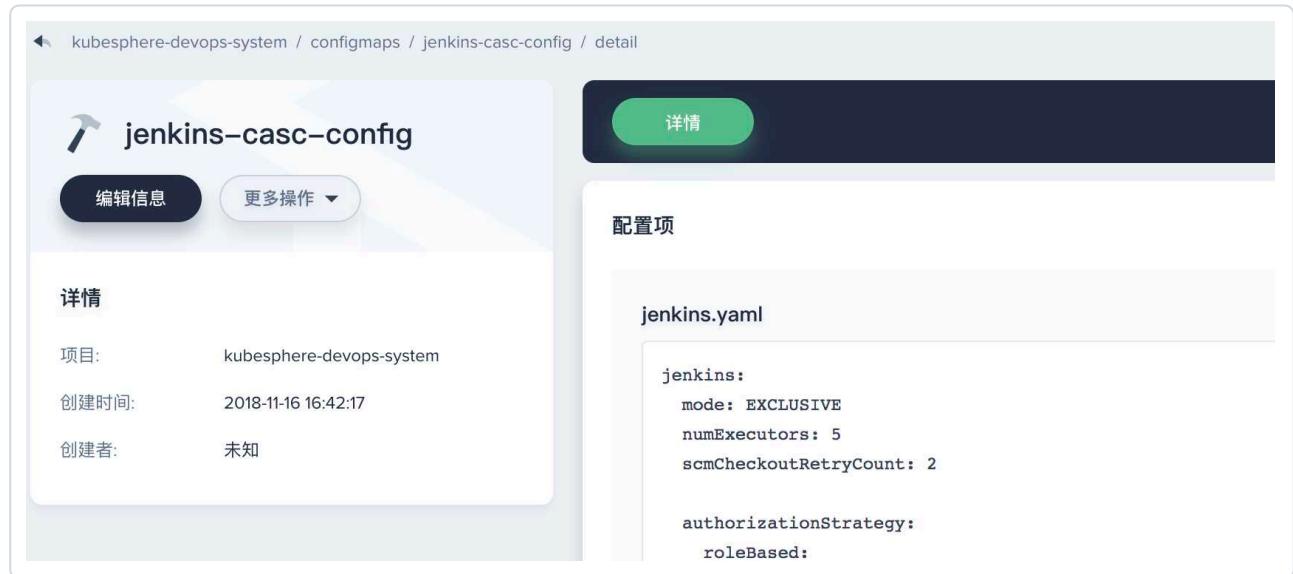
The screenshot shows the KubeSphere Configuration interface. On the left sidebar, under '配置中心' (Configuration Center), the '配置' (Config) option is selected, indicated by a red arrow. The main area displays a table of ConfigMaps:

名称	Config Field	创建时间
jenkins	plugins.txt,apply_config.sh,config.xml,initK8sCredentials.ls,groovy,initMailer,groovy,jenkins,CLIXML,jenkins.model,JenkinsLocationConfiguration.xml	2018-11-16 16:46:44
jenkins-tests	run.sh	2018-11-16 16:46:44
jenkins-casc-config	jenkins.yaml	2018-11-16 16:42:17

At the bottom right of the table, there is a context menu with three options: '/ 编辑' (Edit), '/ 编辑配置文件' (Edit Configuration File), and '/ 编辑 ConfigMap' (Edit ConfigMap). A red arrow points to the '/ 编辑 ConfigMap' option.

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 an 'EXCLUSIVE' mode, 5 executors, and a scmCheckoutRetryCount of 2. It also specifies a role-based authorization strategy.

```
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。

The screenshot shows the Jenkins dashboard with several management options:

- Manage and Assign Roles**: Handle permissions by creating roles and assigning them to users/groups.
- 关于 Jenkins**: View version and certificate information.
- 管理旧数据**: Clean up configuration files from old and early versions of plugins.
- Configuration as Code**: Reload your configuration or update configuration source. This option is highlighted with a red arrow.

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。

The screenshot shows the Jenkins Configuration as Code page with the following details:

- Jenkins 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** (Reload) button, highlighted with a red arrow.

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。

# DevOps 运维常见问题

## 性能及缓存命中问题

在 Jenkins 的内存空间不足的情况下可能会下列问题：

1. 流水线页面浏览较慢，部分列表页响应超过10秒
2. 流水线运行列表页、分支列表页等页面排序顺序不对 .....

## 问题排查

当您发生这类问题时可以进入到 Jenkins 的 Pod 当中排查 GC 日志(ks-jenkins后面的xxx视环境输入)：

```
$ kubectl exec -n kubesphere-devops-system ks-jenkins-xxxxxx-xxxx -it bash
```

属于下列命令进入 `jenkins_home` 目录您可以看到有 GC Log 相关文件，在下面的文件中 `gc-2019-05-11_13-05-37.log.0.current` 为当前的 GC LOG 文件：

```
$ cd /var/jenkins_home
```

```
$ ls
caches                               io.kubesphere.jenkins.devops.auth.KubesphereTokenAuthGlobalConfig
casc_configs                         jenkins.CLI.xml                      queue.xml.bak
config.xml                            jenkins.install.InstallUtil.lastExecVersion      scriptApproval
copy_reference_file.log               jenkins.model.JenkinsLocationConfiguration.xml
credentials.xml                      jenkins.telemetry.Correlator.xml          secret.key.not-
gc-2019-05-11_13-05-37.log.0.current   jobs
hudson.model.UpdateCenter.xml       logs
hudson.plugins.git.GitTool.xml       lost+found
hudson.plugins.sonar.SonarGlobalConfiguration.xml nodeMonitors.xml
hudson.plugins.sonar.SonarRunnerInstallation.xml nodes
hudson.tasks.Mailer.xml              org.jenkinsci.plugins.workflow.flow.FlowExecutionList.xml
identity.key.enc                    plugins
init.groovy.d                        plugins.txt
                                         workspace
```

此时我们可以使用 `kubectl cp` 命令取出相关的 Jenkins 的 GC 日志并查看。

## 对 Jenkins 的内存进行扩容(需要重启 Jenkins 服务)

当您确定 Jenkins 的内存已经不足时，可以通过调整 Jenkins 的 JVM 参数等配置进行扩容。

使用 `kubectl edit` 编辑 `Jenkins` 部署

输入下面的命令编辑 Jenkins 配置

```
$ kubectl edit -n kubesphere-devops-system deployments.apps ks-jenkins
```

修改文件中的环境变量 `JAVA_TOOL_OPTIONS` 中的 `Xms`（最小堆大小）与 `Xmx`（最大堆大小），默认情况下多节点部署的最小堆大小为3g、最大堆大小为6g。

```
spec:  
  containers:  
    - args:  
        - --argumentsRealm.passwd.$(ADMIN_USER)=$(ADMIN_PASSWORD)  
        - --argumentsRealm.roles.$(ADMIN_USER)=admin  
    env:  
      - name: JAVA_TOOL_OPTIONS  
        value: -Xms3g -Xmx6g -XX:MaxPermSize=512m -XX:MaxRAM=8g -verbose:gc -Xloggc:  
              -XX:NumberOfGCLogFiles=2 -XX:+UseGCLogFileRotation -XX:GCLogFileSize=100m  
              -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:+PrintHeapAtGC  
              -XX:+PrintGCCause -XX:+PrintTenuringDistribution -XX:+PrintReferenceGC  
              -XX:+PrintAdaptiveSizePolicy -XX:+UseG1GC -XX:+UseStringDeduplication  
              -XX:+ParallelRefProcEnabled -XX:+DisableExplicitGC -XX:+UnlockDiagnosticVMOptions  
              -XX:+UnlockExperimentalVMOptions
```

修改文件中的 `limit` 避免发生 kubernetes 中的 OOM, requests 与 limits 中的内存大小分别比 Xms 大小和 Xmx 大小略大。

```
resources:  
  limits:  
    cpu: "1"  
    memory: 8Gi  
  requests:  
    cpu: 500m  
    memory: 4Gi
```

#### 注意 修改部署配置将使 Jenkins 重启

修改完成后保存即可完成 Jenkins 的内存扩容。

## 减少不必要的构建数据

为了尽量减少 Jenkins 的内存消耗，可以使用以下方法来清理不需要的构建、分支等。

## 为不关联代码仓库的流水线清理数据

在不关联代码仓库的流水线的编辑页面当中，我们可以为流水线配置丢弃旧的构建。

在默认情况下我们推荐设置为保留7天，并最多保留十个具体如下图所示。



## 为关联代码仓库的流水线清理数据

类似于不关联代码仓库的流水线，我们可以为关联代码仓库的流水线配置丢弃旧的分支。

丢弃旧的分支是指丢弃流水线下的某个分支，及这个分支下的所有构建。

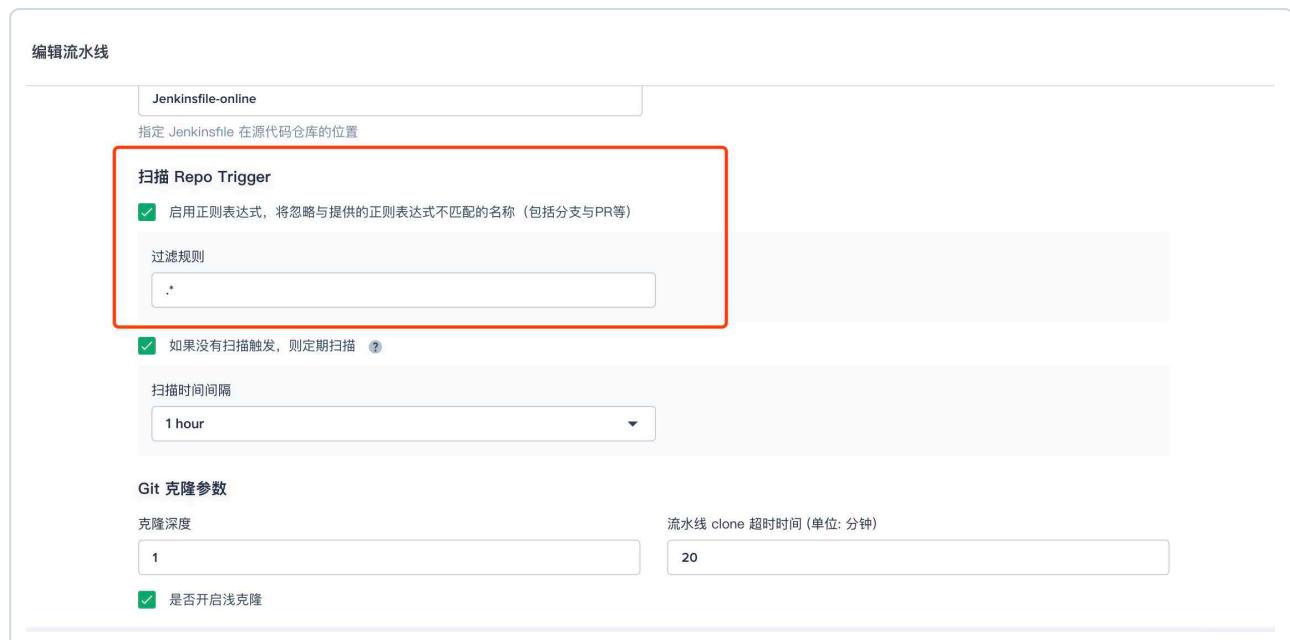
我们不推荐在远程代码仓库仍存在分支的情况下丢弃旧的分支，因此我们这里将如下图所示进行设置，这表示当代码仓库中的分支被删除时，Jenkins 中对应的分支也会删除。



## 避免构建不需要DevOps功能的分支

用户的代码仓库中往往都有各种不同的分支，其中一些分支可能是临时的开发分支，这些分支并不需要运行 DevOps 流水线。

我们可以通过 DevOps 流水线的设置来过滤一些不需要的分支，如下图所示。



## 升级 Jenkins Agent 的包版本

在默认的安装中，我们为用户内置了一部分 agent，具体可以查看[内置 agent 信息](#)的相关文档。

用户在自己的使用场景当中，可能会使用不同的语言版本或不同的工具版本。这篇文档主要介绍如何替换内置的 agent。

Kubesphere Jenkins 的每一个 agent 都是一个 Pod，如果要替换内置的 agent，就需要替换 agent 的相应镜像。

## 构建最新 nodejs 版本的 agent 镜像

我们 agent 的源代码都在 Github KubeSphere 组织下，其中 nodejs agent 的镜像的源代码仓库地址为

# builder-nodejs。

可以看到我们现在镜像的 Dockerfile 为：

```
FROM kubespheredev/builder-base:latest

RUN curl -f --silent --location https://rpm.nodesource.com/setup_9.x | bash - && \
    yum install -y nodejs gcc-c++ make bzip2 GConf2 gtk2 chromedriver chromium xorg-x11-se

RUN npm i -g watch-cli vsce typescript

# Yarn
ENV YARN_VERSION 1.3.2
RUN curl -f -L -o /tmp/yarn.tgz https://github.com/yarnpkg/yarn/releases/download/v${YARN_VERSI
    tar xf /tmp/yarn.tgz && \
    mv yarn-v${YARN_VERSION} /opt/yarn && \
    ln -s /opt/yarn/bin/yarn /usr/local/bin/yarn
```

我们将修改 Dockerfile，将 nodejs 版本调整为 10.x，并将 yarn 的版本调整到 1.16.0，则我们的 Dockerfile 如下所示：

当我们完成了 Dockerfile 的编写就可以构建 Docker 镜像并验证镜像是否满足要求：

使用下面的命令构建 Docker 镜像，其中 -t 后参数为镜像名称，可以根据自己的需要进行调整。

```
$ docker build -t runzexia/builder-nodejs:advanced-2.1.0-dev .
Step 1/5 : FROM kubesphere/builder-base:latest
--> 19d0e8cccd4bb
Step 2/5 : RUN curl -f --silent --location https://rpm.nodesource.com/setup_10.x | bash - &&
.....
Step 5/5 : RUN curl -f -L -o /tmp/yarn.tgz https://github.com/yarnpkg/yarn/releases/download/v1.16.0/yarn-v1.16.0.tar.gz
--> Running in 5ce7b5e09e7e
% Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
          Dload  Upload Total Spent   Left Speed
100  609    0  609    0    0  566      0 --:--:-- 0:00:01 --:--:--  566
100 1145k  100 1145k   0    0 88823      0 0:00:13 0:00:13 --:--:-- 101k
Removing intermediate container 5ce7b5e09e7e
--> a4ba9a74fd03
Successfully built a4ba9a74fd03
Successfully tagged runzexia/builder-nodejs:advanced-2.1.0-dev
```

可以看到我们成功完成了镜像的构建，现在让我们在本地运行容器来查看内置环境的版本是否满足需求：

```
$ docker run -it runzexia/builder-nodejs:advanced-2.1.0-dev bash

$ node -v
v10.16.0
$ npm -v
6.9.0
$ yarn -v
1.16.0
```

可以看到 nodejs 版本已经更新为较新的版本了，现在我们将镜像推送到镜像仓库当中。

```
$ docker push runzexia/builder-nodejs:advanced-2.1.0-dev
```

## 修改 Jenkins 配置，并重新加载 Jenkins 系统设置

当我们完成了镜像的推送，就可以通过修改配置来修改 Jenkins 的系统设置。

使用集群管理员账号登陆 KubeSphere 页面，进入 企业空间 -> system-workspace (企业空间) -> kubesphere-devops-system (项目) -> 配置中心 -> 配置 -> jenkins-casc-config -> 更多操作 -> 编辑 ConfigMap 。

我们现在可以修改 Agent 的镜像版本了，修改

`{}.jenkins.clouds.templates.nodejs.containers.nodejs.image}` 为 `runzexia/builder-nodejs:advanced-2.1.0-dev` 。

```
jenkins:
  mode: EXCLUSIVE
  numExecutors: 5
  scmCheckoutRetryCount: 2

clouds:
  - kubernetes:
      name: "kubernetes"
      serverUrl: "https://kubernetes.default"
      skipTlsVerify: true
      namespace: "kubesphere-devops-system"
      credentialsId: "k8s-service-account"
      jenkinsUrl: "http://ks-jenkins.kubesphere-devops-system:80"
      jenkinsTunnel: "ks-jenkins-agent.kubesphere-devops-system:50000"
      containerCapStr: "100"
      connectTimeout: "60"
      readTimeout: "60"
      maxRequestsPerHostStr: "32"
    templates:
      - name: "nodejs"
        namespace: "kubesphere-devops-system"
        label: "nodejs"
        nodeUsageMode: "EXCLUSIVE"
        idleMinutes: 0 # Do not reuse pod.
      containers:
        - name: "nodejs"
          image: "runzexia/builder-nodejs:advanced-2.1.0-dev" // 修改此处镜像版本
          command: "cat"
          ttyEnabled: true
          envVars:
            - containerEnvVar:
                key: "DOCKER_HOST"
                value: "tcp://localhost:2375"
        - name: "jnlp"
          image: "jenkins/jnlp-slave:3.27-1"
          args: "${computer.jnlpmac} ${computer.name}"
          resourceRequestCpu: "100m"
          resourceRequestMemory: "32Mi"
        - name: "docker-server"
          image: "docker:18.06.1-ce-dind"
          ttyEnabled: true
          privileged: true
```

修改完毕后，我们就可以参考文档 [Jenkins 系统设置](#)来重新加载我们的配置了。当重新加载配置完成后，我们的 Agent 版本也就更新成功了。

可以在 Jenkins 系统管理的系统设置中已经更新为我们设置的镜像版本。

The screenshot shows the Jenkins 'Container Template' configuration page. The title is 'Container Template'. The configuration fields are as follows:

- 名称 (Name): nodejs
- Docker 镜像 (Docker Image): runzexia/builder-nodejs:advanced-2.1.0-dev
- 总是拉取镜像 (Always pull image): Unchecked
- 工作目录 (Working Directory): /home/jenkins
- 运行的命令 (Command): cat
- 命令参数 (Command Parameters): cat
- 分配伪终端 (Allocate pseudo-terminal): Checked

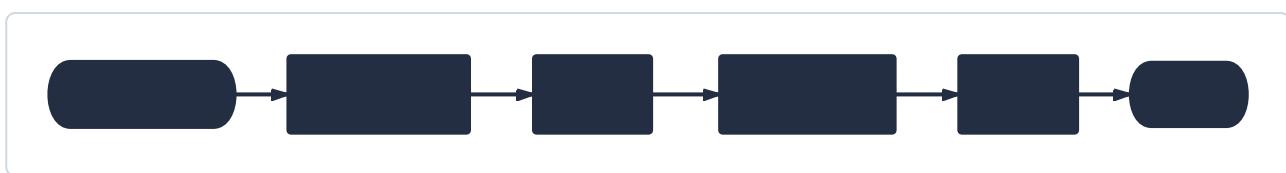
At the bottom, there is a 'EnvVars' section which is currently empty.

## 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，在使用应用模板前需要先将符合 Helm 规范的应用配置包上传至应用仓库后端的对象存储中，然后在应用仓库下基于该对象存储添加一个应用仓库，KubeSphere 会自动加载此仓库下的所有应用，详见 [添加应用仓库](#)。



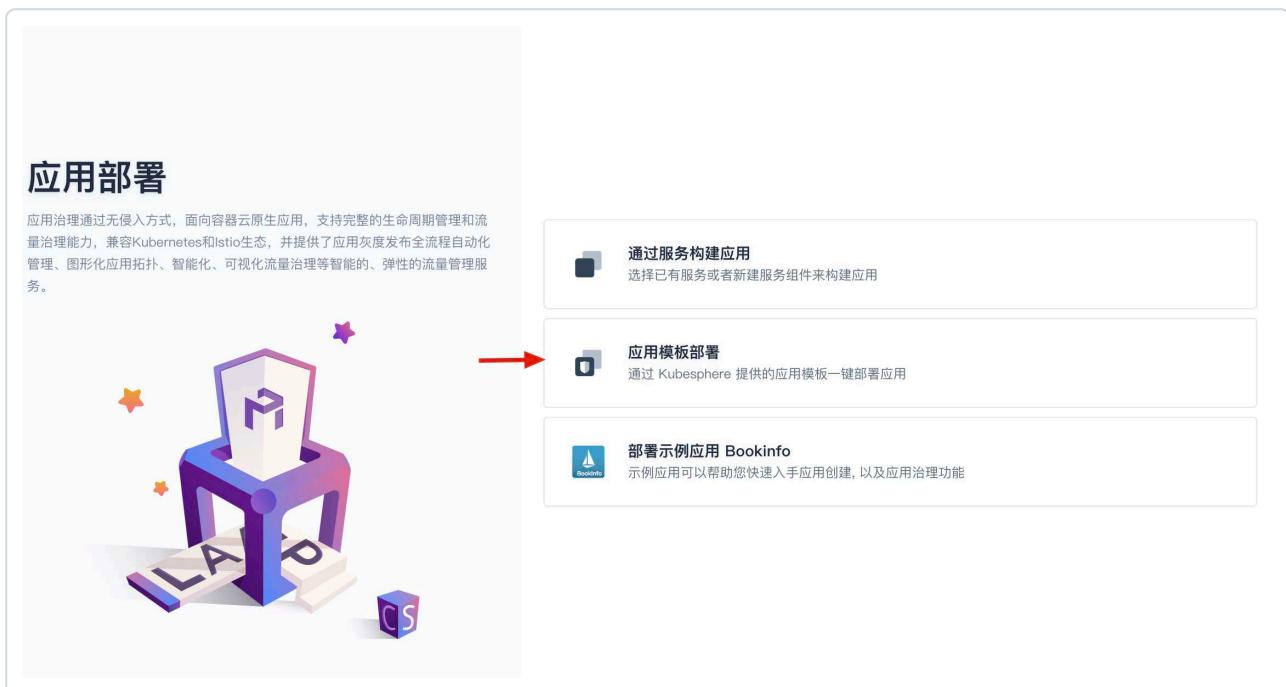
除此之外，应用模板还可以结合 OpenPitrix 的应用全生命周期管理的功能，支持对接应用服务商、开发者和普通用户，通过应用上传、应用审核、部署测试、应用发布、应用版本管理等功能，构建公有或私有的应用商店，为 KubeSphere 提供应用模板服务，企业也可以基于此来建立行业公有或专有应用商店，实现标准化的应用一键交付部署，详见 [OpenPitrix 官方文档](#)。

## 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** → **应用模板部署**，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包然

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** → **平台设置**，选择 **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。

### 添加应用仓库

应用仓库名称 \*

类型

Helm

URL:

验证

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 repo

取消 确定

## 访问应用模板

切换为项目的普通用户 project-regular 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 the title is a search bar with the placeholder '请输入名称进行查找' (Enter name to search) and a dropdown menu labeled '全部仓库' (All Repositories). A large blue banner at the bottom of the page contains the text '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](#)'.

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]  Sematext 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]

# 自制应用

应用通常是一个独立完整的业务功能，比如一个 bookinfo 的书城网站就是一个应用，一个应用由多个服务组件组成，对于微服务而言每个组件都可以独立于其他组件创建、启动、运行和治理的。一个应用组件中又可以有一个或多个组件版本，例如 bookinfo 示例应用中 Reviews 组件就有三个版本，而不同版本后端对应了不同的工作负载。

自制应用允许用户选择已有服务或者新建服务组件来构建应用。

## 创建自制应用

### 前提条件

- 已创建了企业空间、项目和普通用户 `project-regular` 账号，并且项目已开启了外网访问（访问方式为 `NodePort`），请参考 [多租户管理快速入门](#)；
- 使用项目管理员 `project-admin` 邀请项目普通用户 `project-regular` 加入项目并授予 `operator` 角色，若还未邀请请参考 [多租户管理快速入门 - 邀请成员](#)；
- 已准备微服务的各个组件及镜像。

### 操作说明

- 使用 `project-regular` 账号进入项目 `demo-namespace` 后，点击「应用」，选择「自制应用」，点击「部署新应用」，然后在弹窗中选择「通过服务构建应用」。



## 2. 填写应用的基本信息。

- 应用名称：为应用起一个简洁明了的名称，便于用户浏览和搜索。
- 应用版本：应用的版本号，如 v1
- 应用治理：默认开启，开启应用治理后会在每个组件中以 SideCar 的方式注入 Istio-proxy 容器
- 描述信息：简单介绍该应用，让用户进一步了解应用的功能。

## 3. 应用组件组合了工作负载和服务作为应用中的组件，点击「添加组件」。

The screenshot shows the '通过服务构建应用' (Build Application via Service) configuration page. It includes sections for '基本信息' (Basic Information), '应用组件' (Application Components), and '应用路由' (Application Routing). The '基本信息' section contains fields for '应用名称' (bookinfo), '应用版本(可选)' (v1), '应用治理' (Enabled), and '描述信息'. The '应用组件' section has a placeholder for components and a '添加组件' (Add Component) button. The '应用路由' section has a placeholder for routes and a '添加路由规则' (Add Route Rule) button.

## 4. 参考如下提示填写新组件的基本信息。

- 名称：为组件起一个简洁明了的名称，例如 productpage
- 组件版本：应用组件的版本号，例如 v1、v2
- 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。
- 负载类型：支持无状态服务(部署)及有状态服务(有状态副本集)
- 副本：指定副本策略，设置多个副本可以保证应用组件的高可用
- 容器组模板：点击「添加容器」，可以添加容器工作负载和服务，容器组模板的说明可参考 [部署 – 容器组模板](#), [有状态副本集 – 容器组模板](#) 和 [服务](#)。

◀ 应用组件 > 添加新组件

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

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。

副本 1  
指定副本数量 ^ v

组件版本 \*

用于应用治理时区分组件版本，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

负载类型

无状态服务(部署) ▾

支持无状态服务(部署)及有状态服务(有状态副本集)

容器组模板 \*

istio-proxy  
镜像: istio/proxyv2:1.0.0

添加容器

5. 添加存储卷需要预先创建存储卷或使用临时存储卷，可参考 [存储卷](#)。

存储卷

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  
添加临时存储卷

6. 点击「保存」，应用需要暴露给其他用户访问则需要添加应用路由，点击「添加应用路由规则」，可参考 [添加应用路由](#)。

### ◀ 设置路由规则

模式

自动生成 指定域名

通过通配DNS访问，将域名修改为 hostname + 网关地址 +nip.io，然后可以通过 hostname.网关地址.nip.io:节点端口 的形式来访问服务；  
请确保所在网络环境可以正常访问网关地址

路径 \*

/ productpage 端口 添加 Path

取消 保存

This screenshot shows the 'Route Configuration' page in the KubeSphere UI. It includes a header with a back arrow and the title '设置路由规则'. Below the header are two tabs: '自动生成' (Automatically Generated) and '指定域名' (Specify Domain). A note below the tabs explains how to use a domain name with a gateway address and nip.io suffix to access services via a specific port. The main area contains a 'Path' input field with a placeholder of '/'. To its right are dropdown menus for 'productpage' and '端口' (Port), and a '添加 Path' (Add Path) button. At the bottom are '取消' (Cancel) and '保存' (Save) buttons.

# 流量治理

KubeSphere 基于 istio 通过服务网格中部署的 Envoy sidecar 代理，为微服务应用的服务组件提供了流量治理的能力，允许用户修改负载均衡的方式，对连接池管理和熔断器进行细粒度的配置，本文档将对流量治理中的策略配置的所有参数进行说明，建议参考 [熔断](#) 示例文档进行实际操作。

## 流量治理策略配置

点击流量治理拓扑图中的任意一个服务组件，例如点击 reviews，右侧可以看到流量治理策略配置的弹窗。

### 负载均衡算法



### 会话保持



其中，流量治理的负载均衡算法表格中的两种策略：

参数	参数说明
负载均衡算法	<ul style="list-style-type: none"> <li>– ROUND_ROBIN (轮询): 默认负载均衡算法</li> <li>– LEAST_CONN (最小连接数): 随机选取两个健康的主机，再从所选取的两个主机中选择一个链接数较少的主机</li> <li>– RANDOM (随机): 从所有健康的主机中，随机选取一个主机，在负载平衡池的端点上均匀分配负载。在没有健康检查策略的情况下，随机通常会比轮询调度策略更加高效，但不会有任何顺序。</li> </ul>
会话保持	<p><b>根据 HTTP header 中的内容获取哈希:</b> 流量治理根据 HTTP header 中的内容获取哈希。</p> <p><b>根据 Cookie 中的内容获取哈希:</b> 支持用户输入 Cookie 键的名称，转发方式则由设定的 Cookie 键对应的值来计算哈希，哈希相同的请求则会转发至同一个容器组中。例如我们设定 Cookie 中的 User 为键，则通过计算 User 对应的值的哈希来确认转发规则。</p> <p><b>根据源 IP 获取哈希:</b> 根据源 IP 中的内容获得哈希。</p>

在微服务的一个服务组件中，流量治理中可选择开启连接池管理和熔断器等配置，以下将对相关参数进行释义。

## 连接池管理

在连接池管理一栏，点击「开启」即可配置连接池管理的各项参数。

参数	参数说明
最大连接数	是指 Envoy 将为上游群集中的所有主机建立的最大连接数，适用于 HTTP/1.1。
每连接最大请求数	对某一后端的请求中，一个连接内能够发出的最大请求数量。对后端连接中最大的请求数量若设为 1 则会禁止 keep alive 特性。
最大请求重试次数	在指定时间内对目标主机最大重试次数。
连接超时时间	TCP 连接超时时间，最小值必须大于 1ms。 最大连接数和连接超时时间是对 TCP 和 HTTP 都有效的通用连接设置
最大等待请求数	等待列队的长度，默认为 1024。

The screenshot shows the KubeSphere UI with the '流量治理' tab selected. On the left, three services are listed: 'productpage', 'reviews', and 'ratings'. The 'reviews' service is currently selected. On the right, the '流量治理' (Traffic Governance) section is expanded, showing configuration for connection pools. A red arrow points to the '开启' (Enable) button for connection pool management.

## 熔断器

在熔断器一栏，点击「开启」即可配置熔断器的各项参数。

参数	参数说明
连续错误响应个数	在一个检查周期内，连续出现 5xx 状态码的错误的个数，超过该值后，实例将会被移出连接池。
检查周期(单位: s)	将会对检查周期内的响应码进行筛选，检测实例上一次被移除和这一次被移除之间的时间间隔。默认值为 10s，最小值为 1ms。
容器组隔离比例(单位: %)	上游服务的负载均衡池中允许被移除的实例的最大百分比，采用向上取整。
最短隔离时间 (单位: s)	实例最短的移除时间。实例每次被移除后的隔离时间为被移除的次数与最小移除时间的乘积。该策略设置让系统能够自动增加不健康上游服务实例的隔离时间。

The screenshot shows the KubeSphere interface for managing traffic governance. It displays two service configurations: `productpage` and `reviews`.

- productpage Service:** Has a green heart icon indicating it's healthy. It shows metrics: 0.84 RPS, 100.00% success rate, 95 ms latency, and 1/1 instances.
- reviews Service:** Also has a green heart icon. It shows three instances of the `v1` version. Metrics for these instances are: 0.41 RPS, 100.00% success rate, 5 ms latency, and 1/1 instances; 0.44 RPS, 100.00% success rate, 148 ms latency, and 1/1 instances; and another `v2` instance with 0.00 RPS, 100.00% success rate, 0 ms latency, and 1/1 instances.
- Configuration Panel (Right Side):**
  - 流量监测 (Traffic Monitoring):** Shows the `reviews` service selected. A red arrow points to the "开启" (Enable) button for the **熔断器 (Circuit Breaker)** feature.
  - 熔断器 (Circuit Breaker) Settings:** Set to 5 errors in a 10-second window.
  - 检查周期 (Check Interval) Settings:** Set to 10 seconds.
  - 容器组隔离比例 (Container Group Isolation Ratio) Settings:** Set to 10%.

## 熔断

在微服务中，系统的各个服务之间在网络上存在大量的调用，在调用过程中，如果某个服务繁忙或者无法响应请求，可能会引发集群的大规模级联故障，从而造成整个系统不可用，引发服务雪崩效应。当下游服务因访问压力过大而响应变慢或失败，上游服务为了保护系统整体的可用性，可以暂时切断对下游服务的调用，达到服务降级的效果，通过牺牲局部保全整体的措施就叫做**熔断（Circuit Breaking）**。

本文基于示例应用 Bookinfo，演示如何对其中的一个服务设置熔断规则，并通过一个负载测试客户端 ([fortio](#)) 来触发熔断机制，在 KubeSphere 中演示熔断现象。

## 预估时间

约 15 分钟。

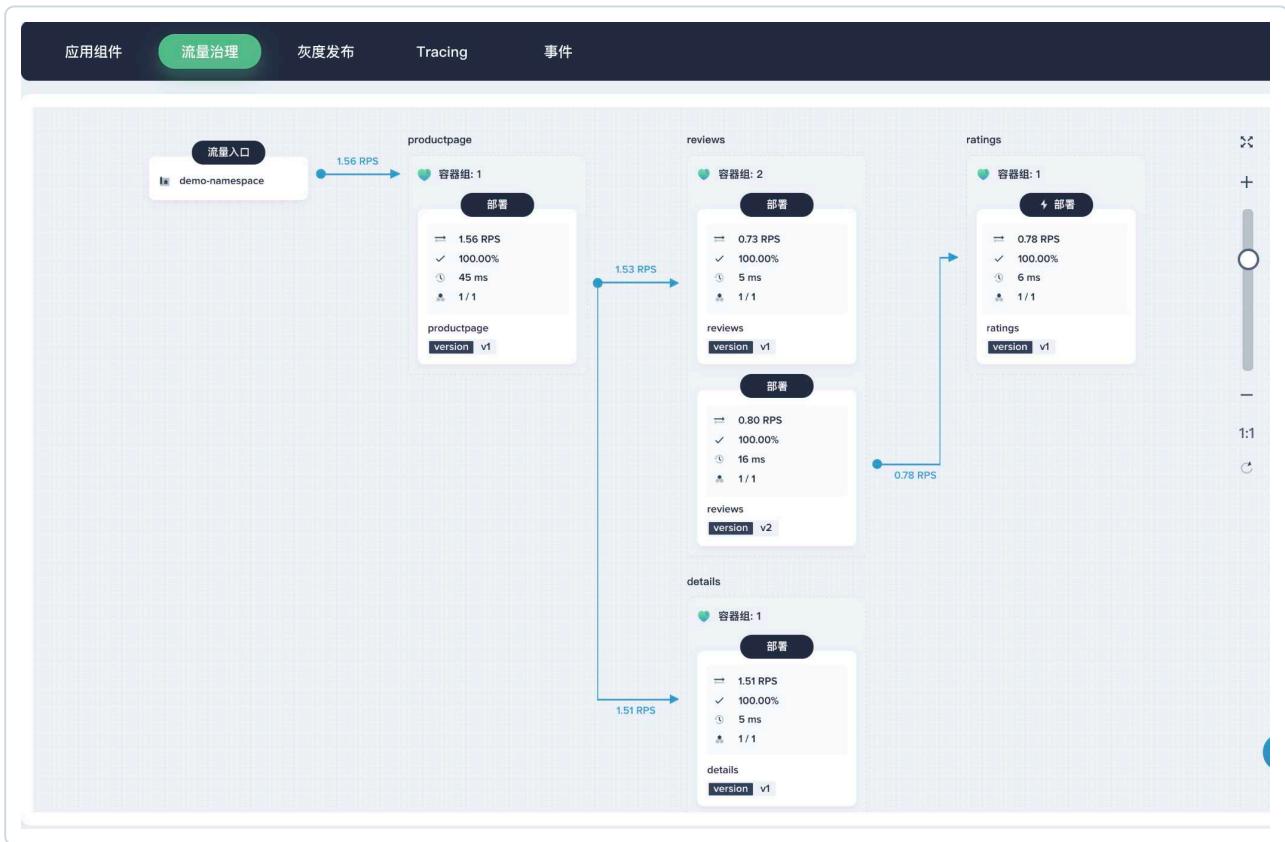
## 前提条件

已完成了 [Bookinfo 微服务的灰度发布](#) 示例中 [查看流量监测](#) 之前的所有步骤。

## 操作示例

### 第一步：设置熔断规则

使用项目普通用户 `project-regular` 登录 KubeSphere，进入示例应用 Bookinfo 的详情页。由于在 [查看流量拓扑图](#) 步骤中，通过 `watch` 命令引入真实的访问流量后，此时在流量治理中可以看到流量治理拓扑图。



1、点击 ratings，在右侧展开「流量治理」，打开「连接池管理」和「熔断器」，参考如下设置。

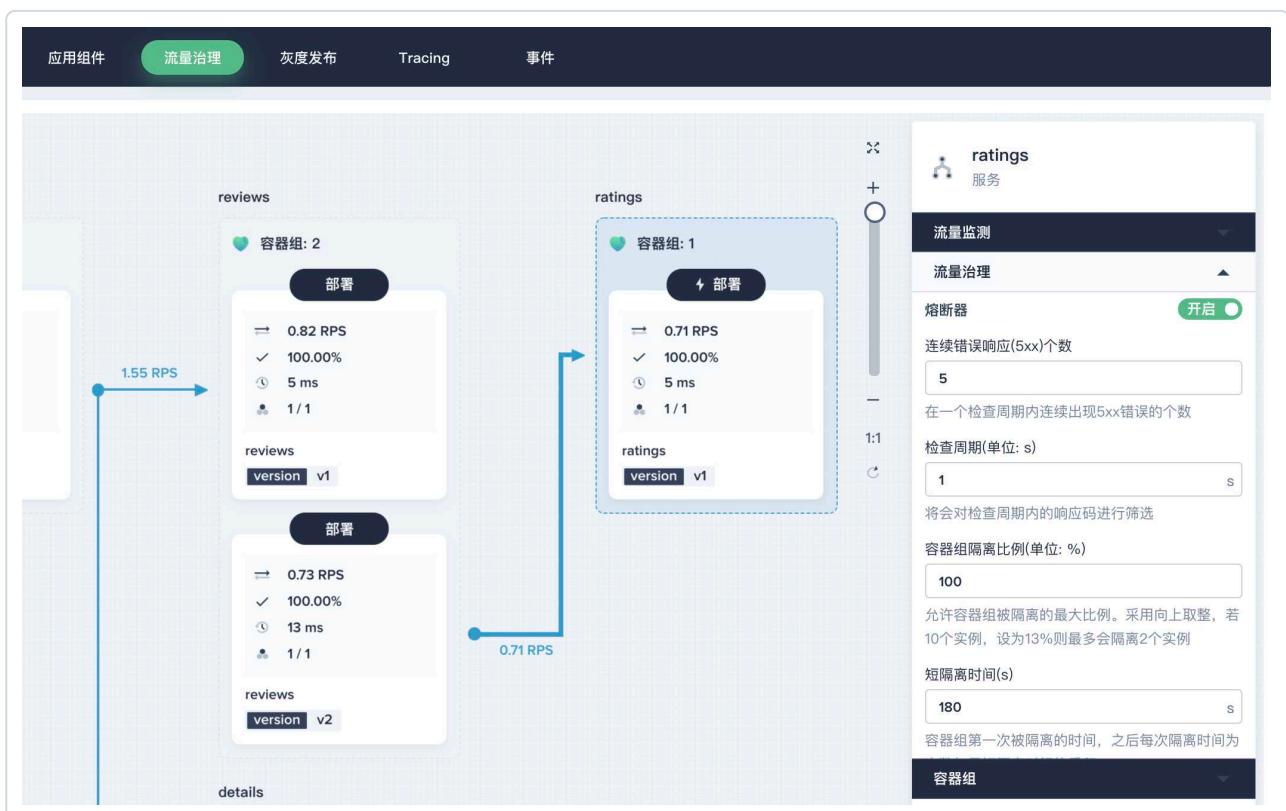
- 连接池管理：将 **最大连接数** 和 **最大等待请求数(等待列队的长度)** 都设置为 **1**，表示如果超过了一个连接同时发起请求，Istio 就会熔断，阻止后续的请求或连接；
- 熔断器：参考如下设置，表示每 **1** 秒钟扫描一次上游主机，连续失败 **5** 次返回 **5xx** 错误码的 **100%** 数量的主机 (Pod) 会被移出连接池 **180** 秒，熔断器参数释义详见 [流量治理 – 熔断器](#)
  - 连续错误响应(**5xx**)个数：5；
  - 检查周期(单位: s)：1；
  - 容器组隔离比例(单位: %)：100；
  - 短隔离时间(s)：180。

### 连接池参数说明：

- 最大连接数：表示在任何给定时间内，Envoy 与上游集群（比如这里是 ratings 服务）建立的最大连接数，适用于 HTTP/1.1；
- 每连接最大请求数：表示在任何给定时间内，上游集群中所有主机（比如这里是 ratings 服

务) 可以处理的最大请求数。对后端连接中最大的请求数量若设为 1 则会禁止 keep alive 特性;

- 最大请求重试次数: 在指定时间内对目标主机最大重试次数;
- 连接超时时间: TCP 连接超时时间, 最小值必须大于 1ms。最大连接数和连接超时时间是对 TCP 和 HTTP 都有效的通用连接设置;
- 最大等待请求数 (等待列队的长度): 表示待处理请求队列的长度, 默认为 1024。如果该断路器溢出, 集群的 `upstream_rq_pending_overflow` 计数器就会递增。



2、完成设置后, 下拉至底部点击「确定」, 保存规则。

## 第二步：设置客户端

由于我们已经在 reviews-v2 中 reviews 容器中注入了负载测试客户端 ([fortio](#)), 它可以控制连接数量、并发数以及发送 HTTP 请求的延迟, 能够触发在上一步中设置的熔断策略。因此可以通过 reviews 容器来向后端服务发送请求, 观察是否会触发熔断策略。

1、使用集群管理员账号 `admin` 登入 KubeSphere, 在右下角找到小锤子图标, 然后打开 `kubectl` (或直接 SSH 登录集群任意节点)。

2、执行以下命令登录客户端 Pod (reviews-v2) 并使用 Fortio 工具来调用 ratings 服务，`--curl` 参数表明只调用一次，返回 200 OK 表示调用成功。

```
$ FORTIO_POD=$(kubectl get pod -n demo-namespace | grep reviews-v2 | awk '{print $1}')
$ kubectl exec -n demo-namespace -it $FORTIO_POD -- curl -s -X GET "http://ratings:9080/ratings/0"
HTTP/1.1 200 OK
...
```

### 第三步：触发熔断机制

1、在 ratings 中设置了连接池管理的熔断规则，`最大连接数` 和 `最大等待请求数(等待列队的长度)` 都设置为 1，接下来设置两个并发连接 (`-c 2`)，发送 20 请求 (`-n 20`)：

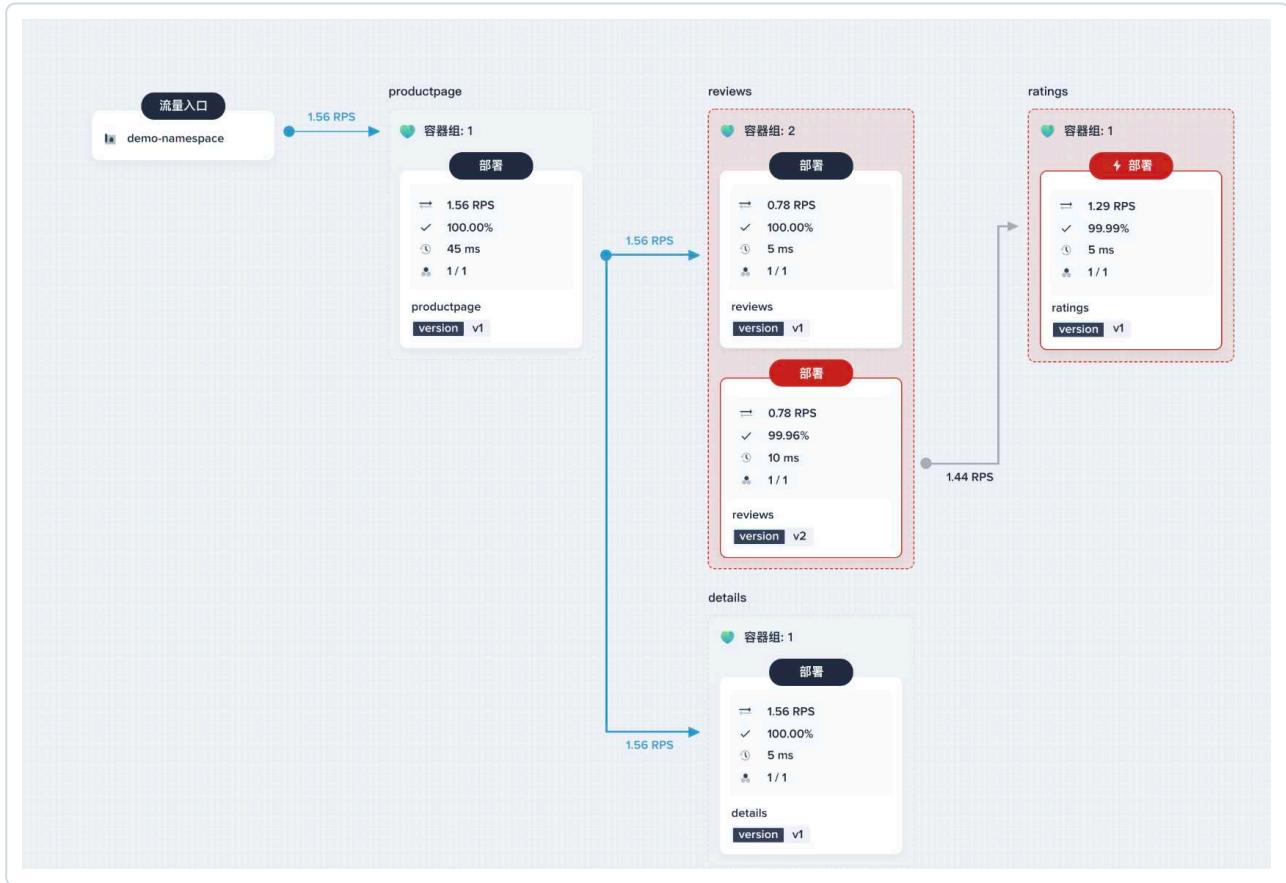
```
$ kubectl exec -n demo-namespace -it $FORTIO_POD -- curl -s -X GET "http://ratings:9080/ratings/0" -c 2
...
Code 200 : 18 (90.0 %)
Code 503 : 2 (10.0 %)
...
```

2、以上可以看到，几乎所有请求都通过了，Istio-proxy 允许存在一些误差。接下来把并发连接数量提高到 3：

```
$ kubectl exec -n demo-namespace -it $FORTIO_POD -- curl -s -X GET "http://ratings:9080/ratings/0" -c 3
...
Code 200 : 22 (73.3 %)
Code 503 : 8 (26.7 %)
...
```

3、查看结果发现熔断行为按照之前的设置规则生效了，此时仅 73.3 % 的请求允许通过，剩余请求被断

路器拦截了。由于此时 503 的返回次数为 8，超过了预先设置的连续错误响应(5xx)个数 5，此时 ratings 的 Pod 将被 100% 地隔离 180 s，ratings 与 reviews-v2 之间也出现了灰色箭头，表示服务间的调用已断开，ratings 被完全隔离。



4、可以给 reviews-v2 的部署 (Deployment) 添加如下一条 annotation，即可查询 ratings 的 istio-proxy 的状态。在「工作负载」→「部署」列表中找到 reviews-v2，点击右侧 ⋮ 选择「编辑配置文件」，添加一条 `sidecar.istio.io/statsInclusionPrefixes` 的 annotation。

...

annotations:

`sidecar.istio.io/inject: 'true'`

`sidecar.istio.io/statsInclusionPrefixes: 'cluster.outbound,cluster_manager,listener_manager,http_mixer,fil`

...

完成后点击「更新」。

5、查询 istio-proxy 的状态，获取更多相关信息。如下所示 `upstream_rq_pending_overflow` 的值是 10，说明有 10 次调用被熔断。

```
$ kubectl exec -n demo-namespace -it $FORTIO POD -- sh -c 'curl localhost:15000/statistics'|jq .  
...  
cluster.outbound|9080|v1|ratings.demo-namespace.svc.cluster.local.upstream_rq_pending_overflows:  
cluster.outbound|9080|v1|ratings.demo-namespace.svc.cluster.local.upstream_rq_pending_total:  
...  
...
```

# 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

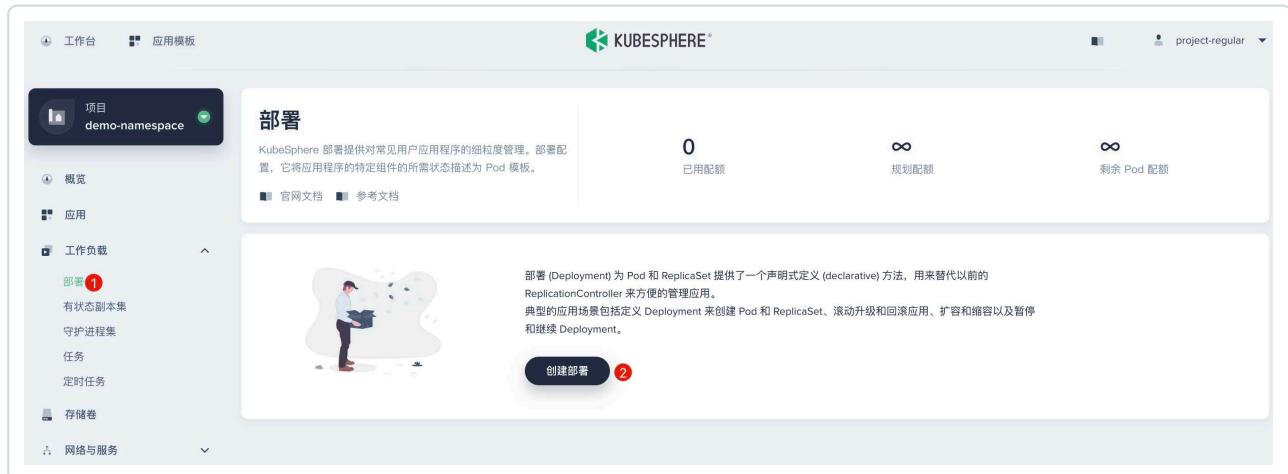
部署 (Deployment) 为 Pod 和 ReplicaSet 提供了一个声明式定义 (declarative) 方法来管理应用。典型的应用场景包括定义 Deployment 来创建 Pod 和 ReplicaSet、滚动升级和回滚应用、扩容和缩容以及暂停和继续 Deployment。

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

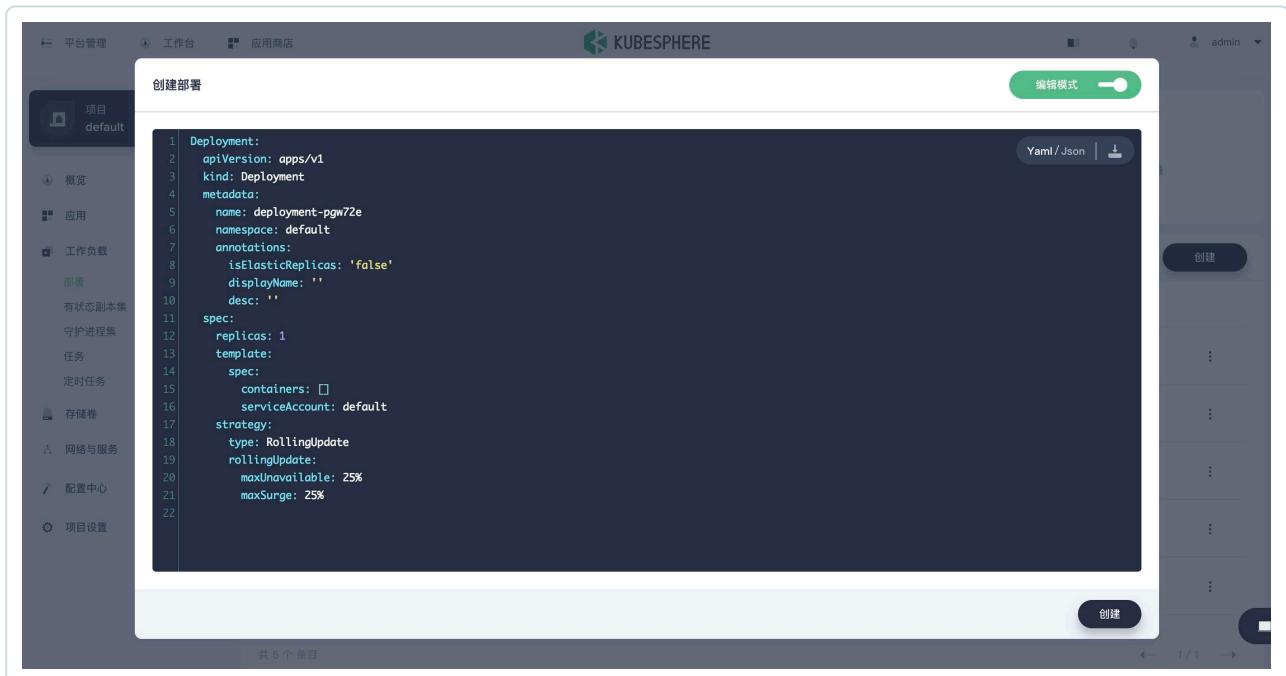
登录 KubeSphere 控制台，在已创建的项目下选择 [工作负载 → 部署](#)，进入部署列表页面。

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 \*

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。

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

Nginx 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。

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，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，目前支持以下两种方式：

### 通过代码构建新的容器镜像

从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以任务 (S2i Job) 的方式去完成。

- 代码地址：源代码仓库地址(目前支持 git)并且可以指定代码分支及在源代码终端的相对路径，如 <https://github.com/kubesphere/devops-java-sample.git>；
- 密钥：如果是私有代码仓库，请选择代码仓库密钥，参考 [创建 GitHub 密钥](#)；

- 映像模板：选择编译环境和对应的编译模板作为 Builder image；
- 代码相对路径：可以指定代码编译的相对路径，默认为 /；
- 映像名称：根据您的 Docker Hub 账号填写，例如 <dockerhub\_username>/<image\_name>，  
dockerhub\_username 为自己的账户名称，确保具有推拉权限；
- tag：镜像标签；
- 目标镜像仓库：选择已创建的镜像仓库，若还未创建请参考 [创建 DockerHub 密钥](#)。注意，基于代码地址中的源代码构建的镜像在部署和 S2i 任务创建完成后，该镜像直接 Push 至目标镜像仓库；
- 环境变量参数（可选）：键值对，应用程序开发人员可以使用环境变量来配置此镜像的运行时行为。

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

代码地址 *	密钥			
<input type="text" value="https://github.com/kubesphere/devops-java-sample.git"/>	<input type="button" value="请选择"/>			
如果是私有代码仓库，请选择代码仓库密钥				
镜像模板 *	代码相对路径(可选):			
<input type="button" value="kubespheredev/java-8-centos7"/>	<input type="text" value="/"/>			
选择编辑环境，您也可以查看对应的 编译模板				
镜像名称 *	tag *			
<input type="text" value="pengfeizhou/sample"/>	<input type="text" value="latest"/>			
镜像名称及Tag，默认为代码仓库的项目名称				
目标镜像仓库				
<input type="button" value="请选择"/>				
需要选择一个有推送权限的镜像仓库存放镜像，如果没有可以新建镜像仓库密钥				
环境变量参数 ^				
<p>应用程序开发人员可以使用以下环境变量来配置此镜像的运行时行为；详细的配置说明请查看 <a href="#">编译模板</a></p> <table border="1"> <tr> <td>键</td> <td>值</td> <td><input type="button" value="删除"/></td> </tr> </table>		键	值	<input type="button" value="删除"/>
键	值	<input type="button" value="删除"/>		

## 选择已有镜像部署容器

从公开或者私有镜像仓库中拉取镜像，若不填写镜像仓库地址则镜像默认从 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 nginx:1.16。

**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

选择已有镜像部署容器  
从公开或者私有镜像仓库中拉取镜像

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。

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

容器规格设置  
对容器的名称及容器的计算资源进行设置

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

CPU  
     
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

2.2. 如果用户有更进一步的需求，可下滑至服务设置和高级设置部分。

- **服务设置：**即设置容器的访问策略，指定容器需要暴露的端口并自定义端口名称，端口协议可以选择 TCP 和 UDP。
- **健康检查：**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。
- **启动命令：**
  - **运行命令：**可自定义容器的启动的运行命令，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](#)。
- **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。
  - **添加环境变量：**以添加键值对的形式来设置环境变量。
  - **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：**默认的镜像拉取策略是 IfNotPresent，在镜像已经在本地存在的情况下，kubelet 将不再去拉取镜像将使用本地已有的镜像。如果需要每次拉取仓库中的镜像，则设置拉取策略为 Always。如果设置为 IfNotPresent 或者 Never，则会优先使用本地镜像。

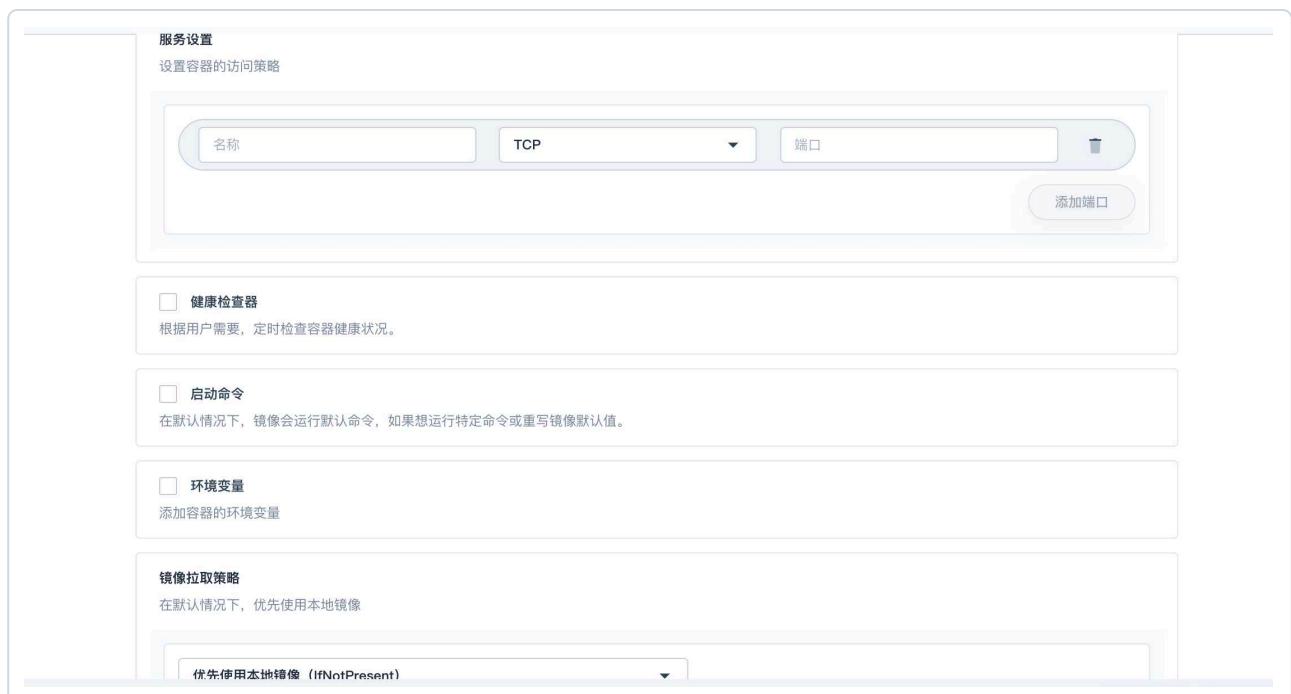
**注意，运行命令和参数部分需要参考如下规则进行使用：**

如果在容器启动时执行一段 shell 命令，则需要在运行命令分别添加两行命令，然后在参数中填写需要执行的 shell 命令，如果是执行 bash 命令则需要把 sh 换成 bash。

```
# 运行命令  
sh # 若执行 bash 命令这里需要替换为 bash  
-c  
# 参数 (填写需要执行的 shell 命令, 如下给出一个示例)  
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```



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](#)。



## 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`。



## 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。



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。

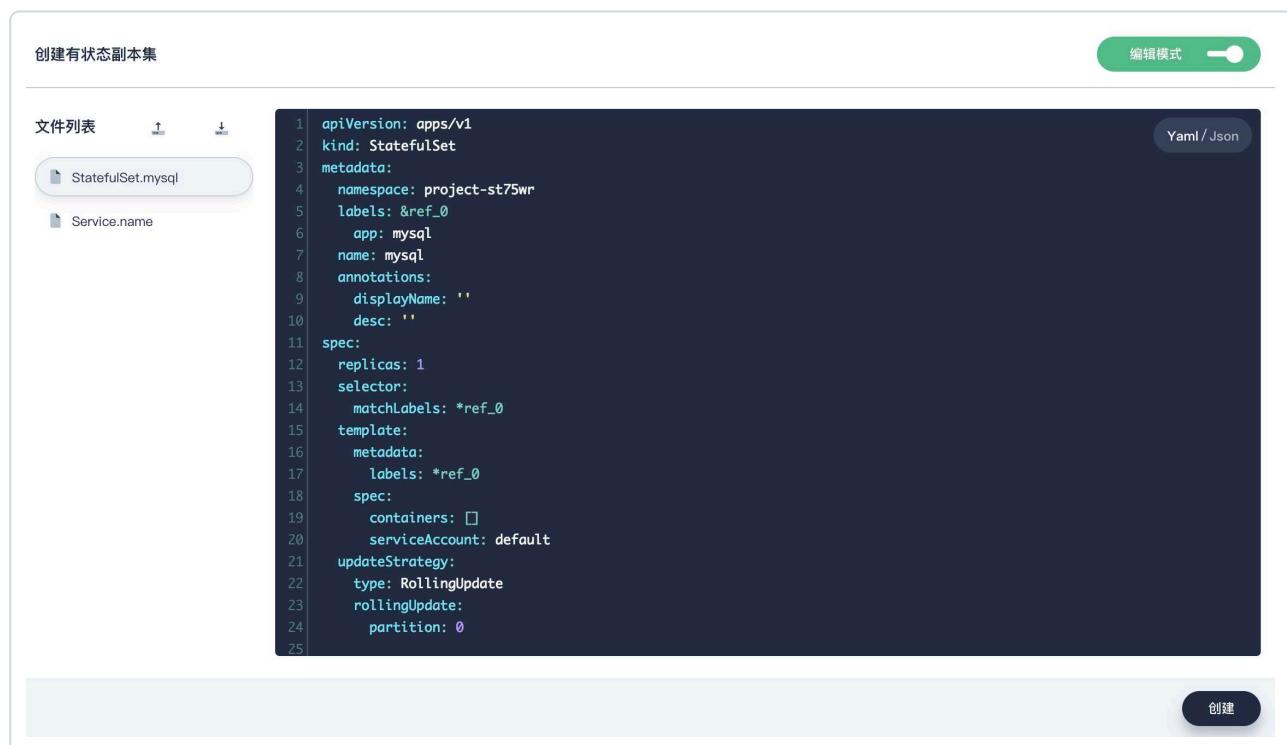
左上角为当前所在项目，如果是管理员登录，可以看到集群所有项目的有状态副本集情况，如果是普通用户，则只能查看授权项目下的所有有状态副本集。列表顶部显示了当前项目的有状态副本集 Pod 配额和数量信息。

The screenshot shows the KubeSphere Control Panel with the following details:

- Project:** demo-namespace
- Workload Type:** StatefulSet (highlighted with a red circle)
- Count:** 0 (Used Capacity), ∞ (Planned Capacity), ∞ (Remaining Pod Capacity)
- Documentation:** Official Documentation, Reference Documentation
- Description:** StatefulSet (StatefulSet) is used to solve the problem of stateful services, used to manage stateful applications, can ensure deployment and scaling order.
- Create StatefulSet:** A button labeled "创建有状态副本集" with a red circle containing the number 2.

## 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。



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**。



## 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**, 然后根据需求添加容器镜像, 目前支持以下两种方式:

### 通过代码构建新的容器镜像

从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以任务(S2i Job)的方式去完成。

- 代码地址: 源代码仓库地址(目前支持 git)并且可以指定代码分支及在源代码终端的相对路径, 如  
<https://github.com/kubesphere/devops-java-sample.git>;
- 密钥: 如果是私有代码仓库, 请选择代码仓库密钥, 参考 [创建 GitHub 密钥](#);
- 映像模板: 选择编译环境和对应的编译模板作为 Builder image;
- 代码相对路径: 可以指定代码编译的相对路径, 默认为 /;
- 映像名称: 根据您的 Docker Hub 账号填写, 例如 <dockerhub\_username>/<image\_name>,   
`dockerhub_username` 为自己的账户名称, 确保具有推拉权限;
- tag: 镜像标签;
- 目标镜像仓库: 选择已创建的镜像仓库, 若还未创建请参考 [创建 DockerHub 密钥](#)。注意, 基于代码地址中的源代码构建的镜像在部署和 S2i 任务创建完成后, 该镜像直接 Push 至目标镜像仓库;
- 环境变量参数 (可选): 键值对, 应用程序开发人员可以使用环境变量来配置此镜像的运行时行为。

通过代码构建新的容器镜像  
从已有的代码仓库中获取代码，并通过Source to Image的方式构建镜像的方式来完成部署，每次构建镜像的过程将以 任务 的方式去完成。

**代码地址 \***

**密钥**

如果是私有代码仓库，请选择代码仓库密钥

**代码相对路径(可选):**

可以指定代码编译的相对路径，默认为 /

**镜像模板 \***

选择编辑环境，您也可以查看对应的 编译模板

**镜像名称 \***

**tag \***

**目标镜像仓库**

需要选择一个有推送权限的镜像仓库存放镜像，如果没有可以新建镜像仓库密钥

**环境变量参数 ^**

应用程序开发人员可以使用以下环境变量来配置此镜像的运行时行为；详细的配置说明请查看 [编译模板](#)

键	值	删除
---	---	----

## 选择已有镜像部署容器

从公开或者私有镜像仓库中拉取镜像，若不填写镜像仓库地址则镜像默认从 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 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
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。

The screenshot shows the 'Container Resource Configuration' section for a MySQL deployment. It includes fields for the image (mysql:5.6), a note about pulling from a private registry, and options for building the image from code. The 'Container Specification Settings' section contains fields for the container name (mysql) and resource limits. For CPU, the minimum request is 10m and the maximum limit is 500m. For memory, the minimum request is 10Mi and the maximum limit is 500Mi.

2.2. 如果用户有更进一步的需求，可下滑至服务设置和高级设置部分。

- **服务设置：**即设置容器的访问策略，指定容器需要暴露的端口并自定义端口名称，端口协议可以选择 TCP 和 UDP。
- **健康检查：**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。
- **启动命令：**

- **运行命令**: 可自定义容器的启动的运行命令, 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](#)。
- **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。
    - **添加环境变量**: 以添加键值对的形式来设置环境变量。
    - **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**: 默认的镜像拉取策略是 IfNotPresent, 在镜像已经在本地存在的情况下, kubelet 将不再去拉取镜像将使用本地已有的镜像。如果需要每次拉取仓库中的镜像, 则设置拉取策略为 Always。如果设置为 IfNotPresent 或者 Never, 则会优先使用本地镜像。

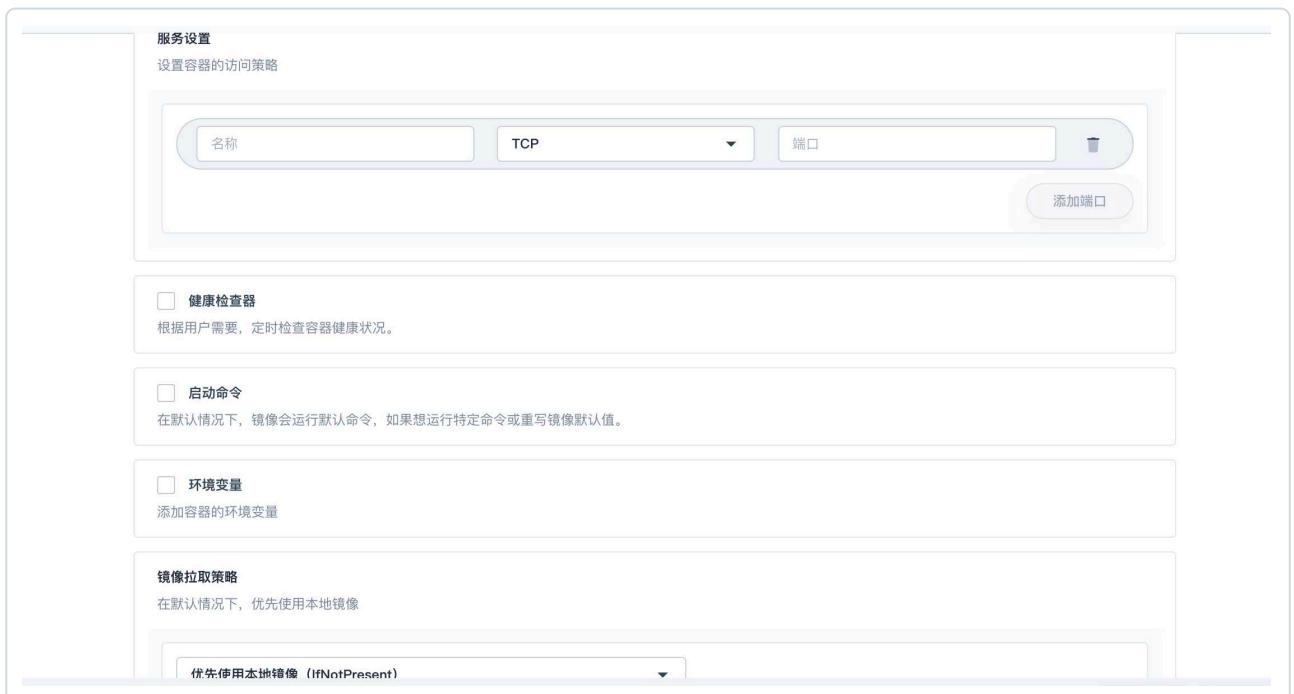
注意, 运行命令和参数部分需要参考如下规则进行使用:

如果在容器启动时执行一段 shell 命令, 则需要在运行命令分别添加两行命令, 然后在参数中填写需要执行的 shell 命令, 如果是执行 bash 命令则需要把 sh 换成 bash。

```
# 运行命令
sh # 若执行 bash 命令这里需要替换为 bash
-c
# 参数 (填写需要执行的 shell 命令, 如下给出一个示例)
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```



设置完成后点击 **保存**。



## 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

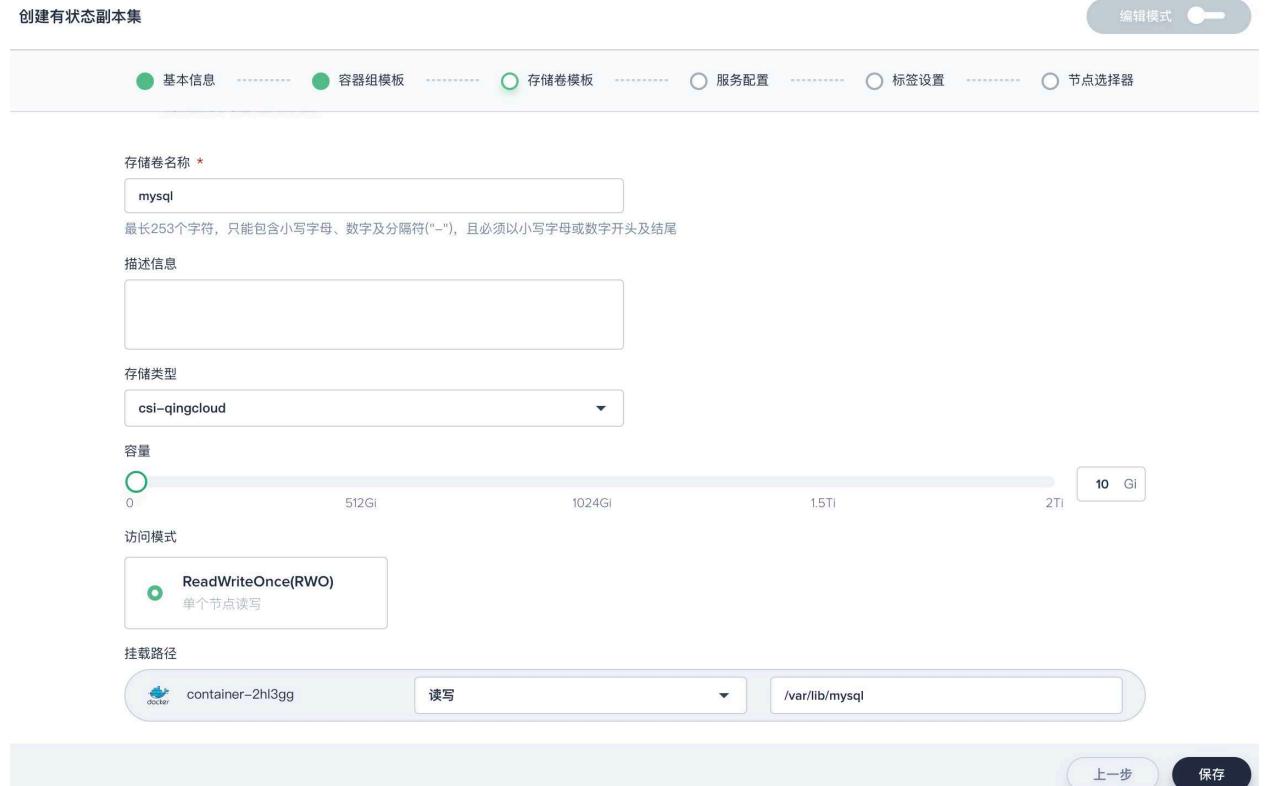
存储类型 csi-qingcloud

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。



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 \*

会话亲和性

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 垃圾桶

添加端口

## 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`。



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 mysql 垃圾桶

添加

## 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。



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". At the top right is a green button labeled "编辑模式" with a switch icon. Below it is a "Yaml / Json" toggle. 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 "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。

The screenshot shows the "基本信息" (Basic Information) step of the creation wizard. At the top left is a "创建守护进程集" button. To its right is an "编辑模式" (Edit Mode) switch. Below these 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). The "基本信息" tab is selected and highlighted in green.

**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

**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**  
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。



## 2.2. 如果用户有更进一步的需求，可下滑至服务设置和高级设置部分。

- **服务设置：**即设置容器的访问策略，指定容器需要暴露的端口并自定义端口名称，端口协议可以选择 TCP 和 UDP。
- **健康检查：**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。
- **启动命令：**
  - **运行命令：**可自定义容器的启动的运行命令，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](#)。
- **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。
  - **添加环境变量：**以添加键值对的形式来设置环境变量。
  - **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：**默认的镜像拉取策略是 IfNotPresent，在镜像已经在本地存在的情况下，kubelet 将不再去拉取镜像将使用本地已有的镜像。如果需要每次拉取仓库中的镜像，则设置拉取策略为 Always。如果设置为 IfNotPresent 或者 Never，则会优先使用本地镜像。

注意，运行命令和参数部分需要参考如下规则进行使用：

如果在容器启动时执行一段 shell 命令，则需要在运行命令分别添加两行命令，然后在参数中填写需要执行的 shell 命令，如果是执行 bash 命令则需要把 sh 换成 bash。

```
# 运行命令  
sh # 若执行 bash 命令这里需要替换为 bash  
-c  
# 参数 (填写需要执行的 shell 命令，如下给出一个示例)  
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```



设置完成后点击 **保存**。

The screenshot shows two tabs of a configuration interface:

- 服务设置 (Service Settings):** This tab is for setting up how containers are accessed. It includes fields for port selection (TCP), health checks, startup commands, environment variables, and image pull strategies.
- 容器配置 (Container Configuration):** This tab is for detailed container configuration. It includes sections for command, parameters, ports, environment variables, and image pull strategies.

## 更新策略

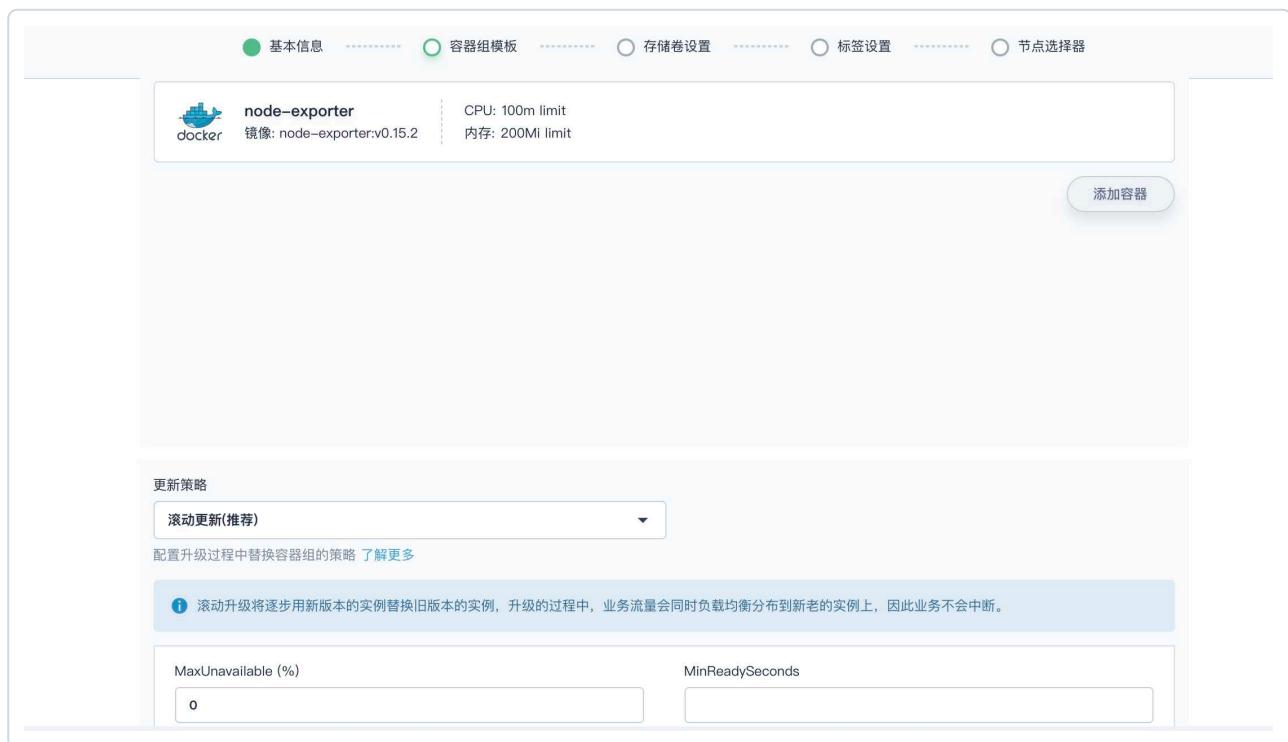
在 Kubernetes 中策略是指定新的 Pod 替换旧的 Pod 的策略，守护进程集的更新策略分为 **滚动更新 (RollingUpdate)** 和 **删除容器组时更新 (OnDelete)** 两种类型：

- 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。



## 第四步：添加标签

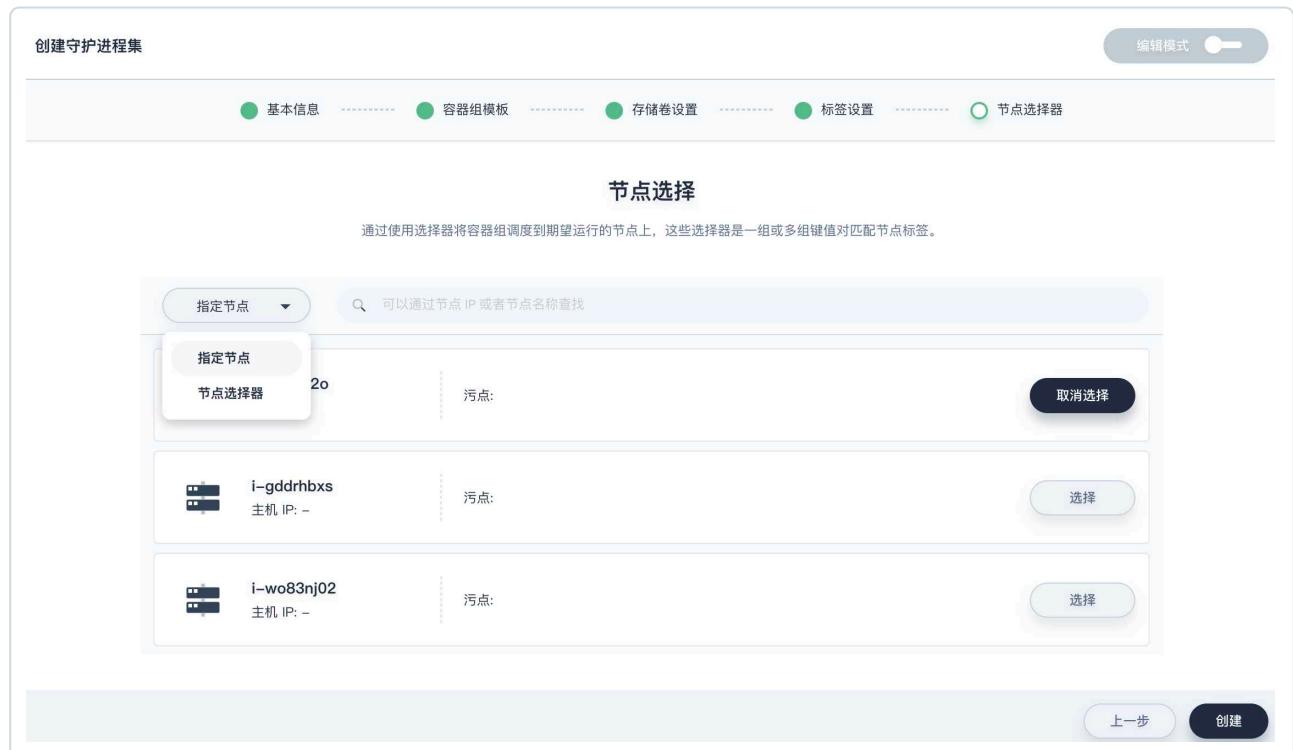
标签设置页用于指定资源对应的一组或者多组标签 (Label)。Label 以键值对的形式附加到任何对象上，如 Pod, Service, Node 等，定义好标签后，其他对象就可以通过标签来对对象进行引用，最常见的用法便是通过节点选择器来引用对象。一般来说，我们可以为一个 Pod (或其他对象) 定义多个标签，以便于配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；或者监控和分析应用 (日志记录、监控、报警等)。通过多个标签的设置，我们就可以多维度地对对象进行精细化管理，如 `relase: stable ; tier: frontend`。



## 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。



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”。

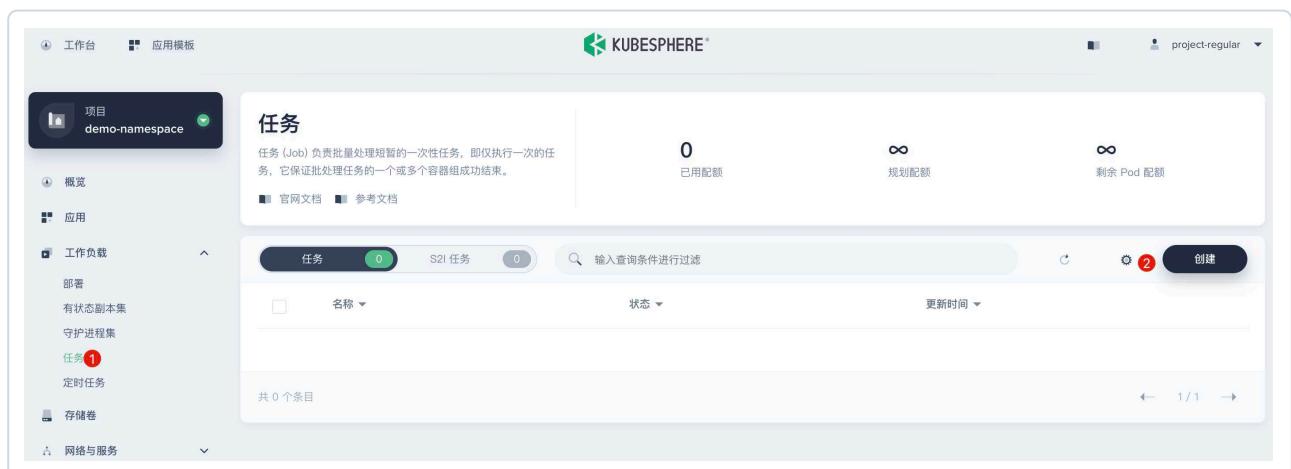
# 任务

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。

A screenshot of the KubeSphere Control Panel. The top navigation bar shows '工作台' (Workstation) and '应用模板' (Application Template). The title bar says 'KUBESPHERE'. On the left, there's a sidebar with project navigation (demo-namespace), sections like '概览' (Overview), '应用' (Application), '工作负载' (Workload) which is expanded to show '部署' (Deployment), '有状态副本集' (StatefulSet), '守护进程集' (DaemonSet), '任务' (Job) with a red notification badge '1', and '定时任务' (CronJob). Below that are '存储卷' (Storage) and '网络与服务' (Network & Services). The main content area is titled '任务' (Job) with a sub-description: '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。' It shows metrics: '0 已用配额' (0 used quota), '∞ 规划配额' (∞ planned quota), and '∞ 剩余 Pod 配额' (∞ remaining Pod quota). Below this is a search bar with filters for '任务' (0), 'S2I 任务' (0), and a search input '输入查询条件进行过滤' (Filter by query condition). A table below shows 0 items. At the bottom right of the main area is a '创建' (Create) button with a red badge '2'. The footer of the page shows '← 1 / 1 →'.

## 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。

创建任务

编辑模式

```
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
```

Yaml / Json

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。

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

### 基本信息

名称 *	项目
job-demo	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	
别名	描述信息
任务示例	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：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。

<p>Back off Limit</p> <input type="text" value="5"/> <p>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</p>	<p>Completions</p> <input type="text" value="10"/> <p>标志 Job 结束需要成功运行的 Pod 个数</p>
<p>Parallelism</p> <input type="text" value="5"/> <p>标志并行运行的 Pod 的个数</p>	<p>Active Deadline Seconds</p> <input type="text" value="100"/> <p>job 运行的超时时间</p>

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 $>$ 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，可下滑至服务设置和高级设置部分。

- **服务设置：**即设置容器的访问策略，指定容器需要暴露的端口并自定义端口名称，端口协议可以选择 TCP 和 UDP。
- **启动命令：**
  - **运行命令：**可自定义容器的启动的运行命令，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](#)。
- **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。
  - **添加环境变量：**以添加键值对的形式来设置环境变量。
  - **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：**默认的镜像拉取策略是 IfNotPresent，在镜像已经在本地存在的情况下，kubelet 将不再去拉取镜像将使用本地已有的镜像。如果需要每次拉取仓库中的镜像，则设置拉取策略为 Always。如果设置为 IfNotPresent 或者 Never，则会优先使用本地镜像。

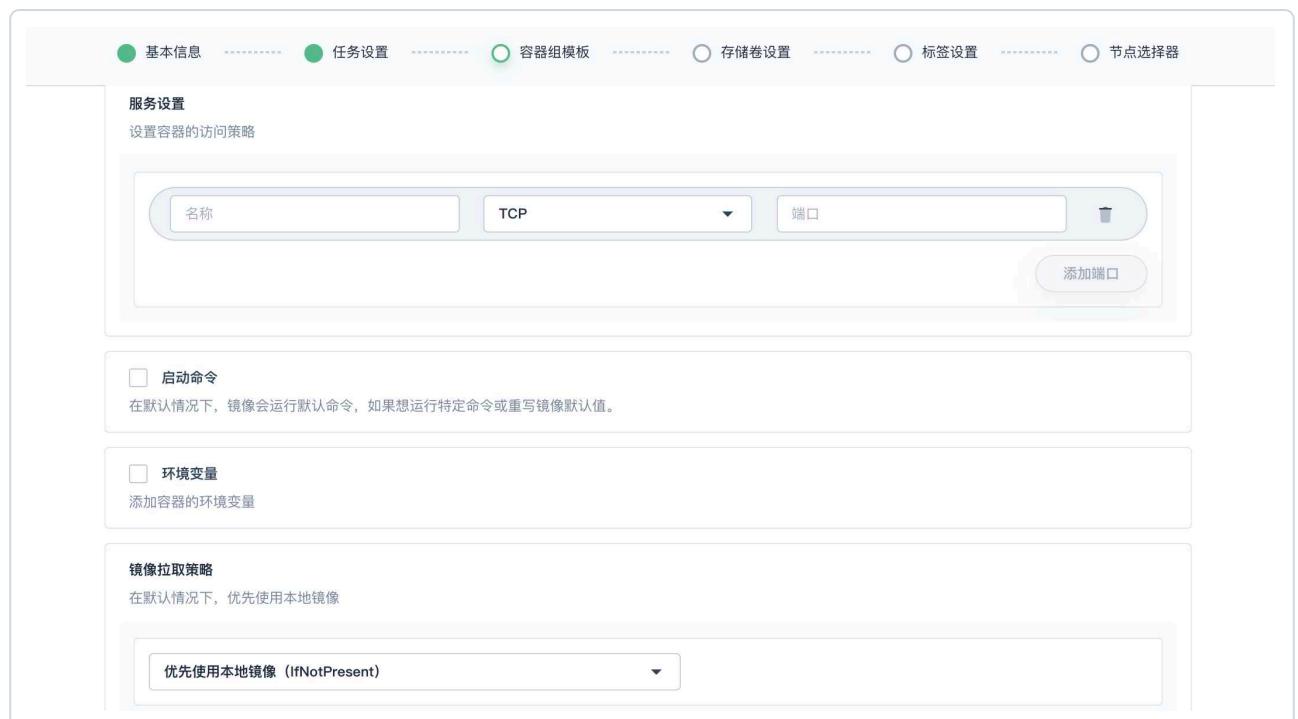
注意，运行命令和参数部分需要参考如下规则进行使用：

如果在容器启动时执行一段 shell 命令，则需要在运行命令分别添加两行命令，然后在参数中填写需要执行的 shell 命令，如果是执行 bash 命令则需要把 sh 换成 bash。

```
# 运行命令  
sh # 若执行 bash 命令这里需要替换为 bash  
-c  
# 参数 (填写需要执行的 shell 命令，如下给出一个示例)  
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```



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。

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。

The screenshot shows a horizontal navigation bar with six items: 基本信息 (Basic Information), 任务设置 (Task Settings), 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). Below the navigation bar, the title "节点选择" (Node Selection) is displayed. A subtitle explains: "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 a selector to schedule a container group to run on the expected node, these selectors are a group or multiple key-value pairs to match node labels.) On the left, there is a dropdown menu with two options: "指定节点" (Specify Node) and "节点选择器" (Node Selector). The "节点选择器" option is currently selected. To the right of the dropdown is a search bar with placeholder text "可以通过节点 IP 或者节点名称查找" (Search by node IP or node name). Below the search bar is a section labeled "污点:" (Taints:) with a list of three entries: "node.kubernetes.io/not-ready:NoSchedule", "node.kubernetes.io/unreachable:NoSchedule", and "node.kubernetes.io/resource-pressure:NoSchedule". On the far right, there is a blue button labeled "选择" (Select) with a red arrow pointing to it.

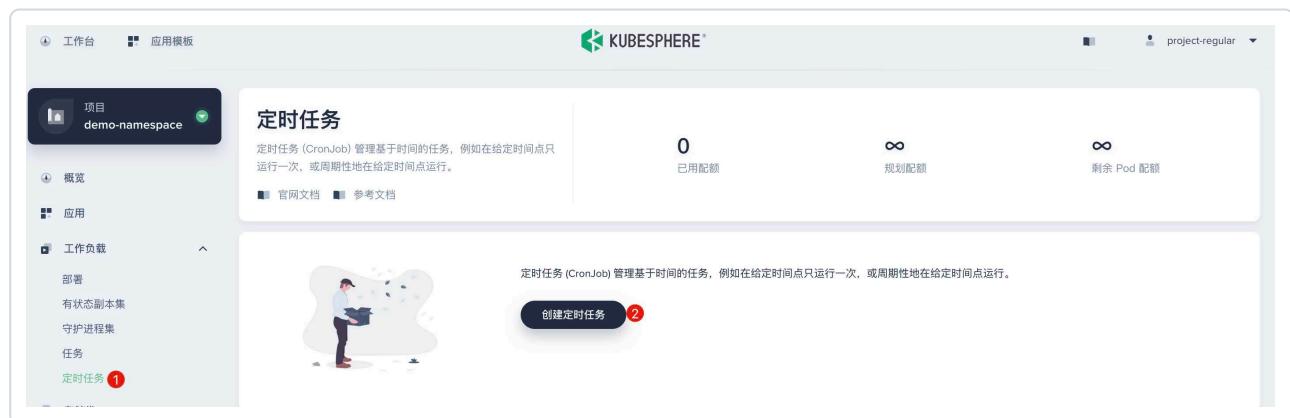
点击创建，即可完成定时任务的创建，状态显示“更新中”是由于拉取镜像需要一定时间，待镜像 pull 成功后状态将显示“运行中”。

# 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。



## 第一步：填写基本信息

1.1. 点击 **创建定时任务**，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。

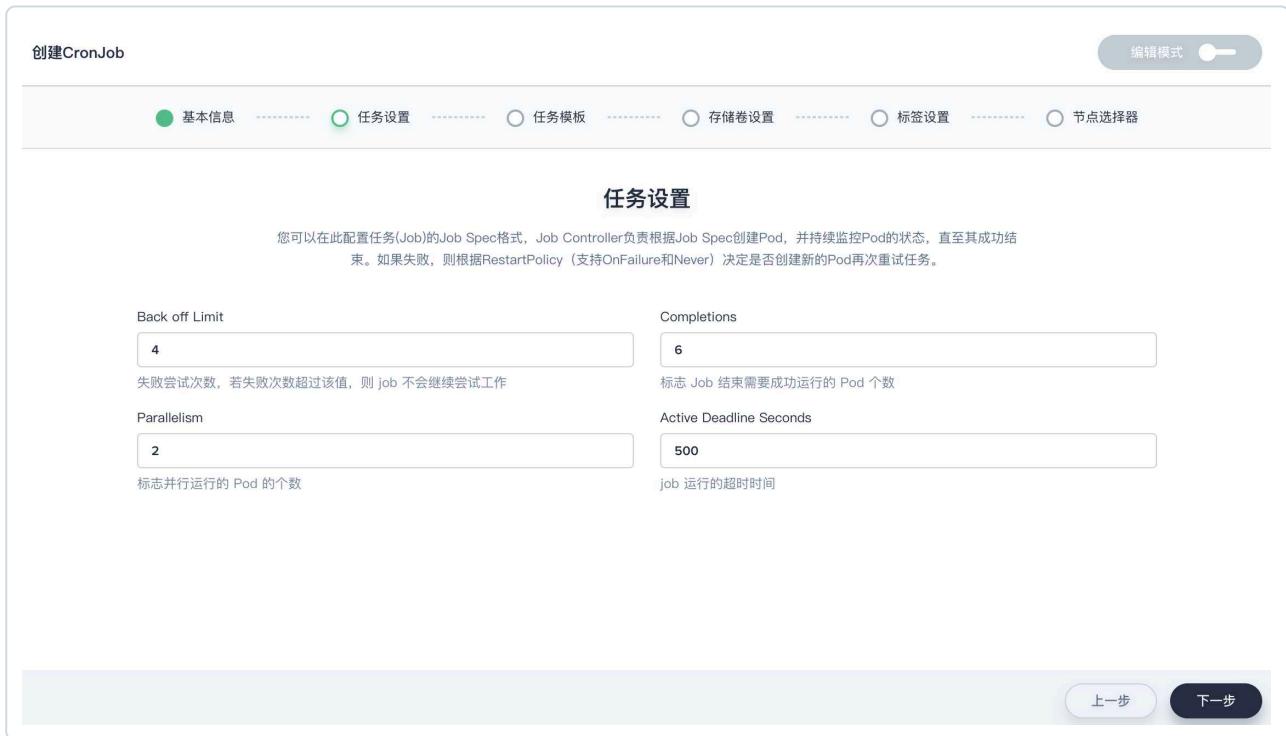
Back off Limit: 4  
失败尝试次数, 若失败次数超过该值, 则 job 不会继续尝试工作

Completions: 6  
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: 500  
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。

**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 $>$ 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。

◀ 添加容器

容器规格设置  
对容器的名称及容器的计算资源进行设置

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	500	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	500	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。

3.3. 如果用户有更进一步的需求，可下滑至服务设置和高级设置部分。

- **服务设置：**即设置容器的访问策略，指定容器需要暴露的端口并自定义端口名称，端口协议可以选择 TCP 和 UDP。
- **启动命令：**

- **运行命令**: 可自定义容器的启动的运行命令, 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](#)。
- **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。
    - **添加环境变量**: 以添加键值对的形式来设置环境变量。
    - **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**: 默认的镜像拉取策略是 IfNotPresent, 在镜像已经在本地存在的情况下, kubelet 将不再去拉取镜像将使用本地已有的镜像。如果需要每次拉取仓库中的镜像, 则设置拉取策略为 Always。如果设置为 IfNotPresent 或者 Never, 则会优先使用本地镜像。

注意, 运行命令和参数部分需要参考如下规则进行使用:

如果在容器启动时执行一段 shell 命令, 则需要在运行命令分别添加两行命令, 然后在参数中填写需要执行的 shell 命令, 如果是执行 bash 命令则需要把 sh 换成 bash。

```
# 运行命令
sh # 若执行 bash 命令这里需要替换为 bash
-c
# 参数 (填写需要执行的 shell 命令, 如下给出一个示例)
while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
```



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

The screenshot shows the KubeSphere interface for managing storage volumes. At the top, there is a navigation bar with tabs: 基本信息 (Basic Information), 定时任务设置 (定时 Task Settings), 容器组模板 (Container Group Template), 存储卷设置 (Storage Volume Settings) (which is currently selected and highlighted in blue), 标签设置 (Label Settings), and 节点选择器 (Node Selector). Below the tabs, the page title is "存储卷设置" (Storage Volume Settings). A sub-instruction below the title reads: "可以将临时存储卷、持久化存储卷挂载至定时任务的容器组内。" (You can mount temporary storage volumes, persistent storage volumes, and so on, to the container groups of scheduled tasks.) There is a large input area with a dashed border, intended for adding storage volumes. Below this area are three buttons: "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：

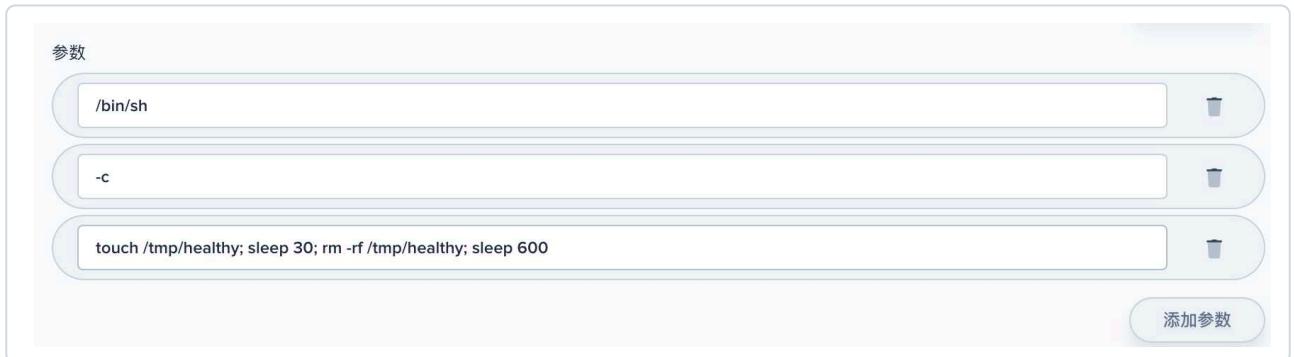
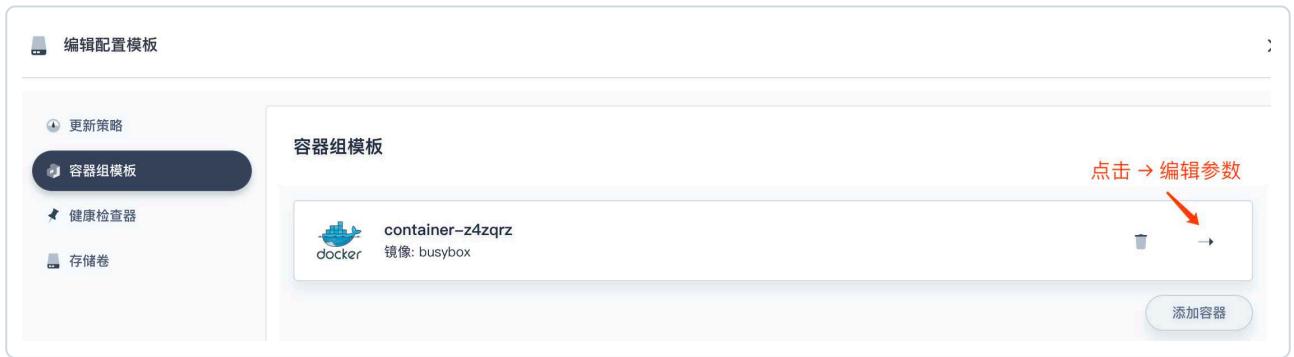
- 检查频率 (periodSeconds): 执行探测的频率，默认是 10 秒。
- successThreshold: 检查失败后，最少连续检查成功多少次才被认定为成功，默认为 1，而对于存活探针其必须是 1
- failureThreshold: 检查成功后，最少连续检查失败多少次才被认定为失败， 默认为 3

## 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

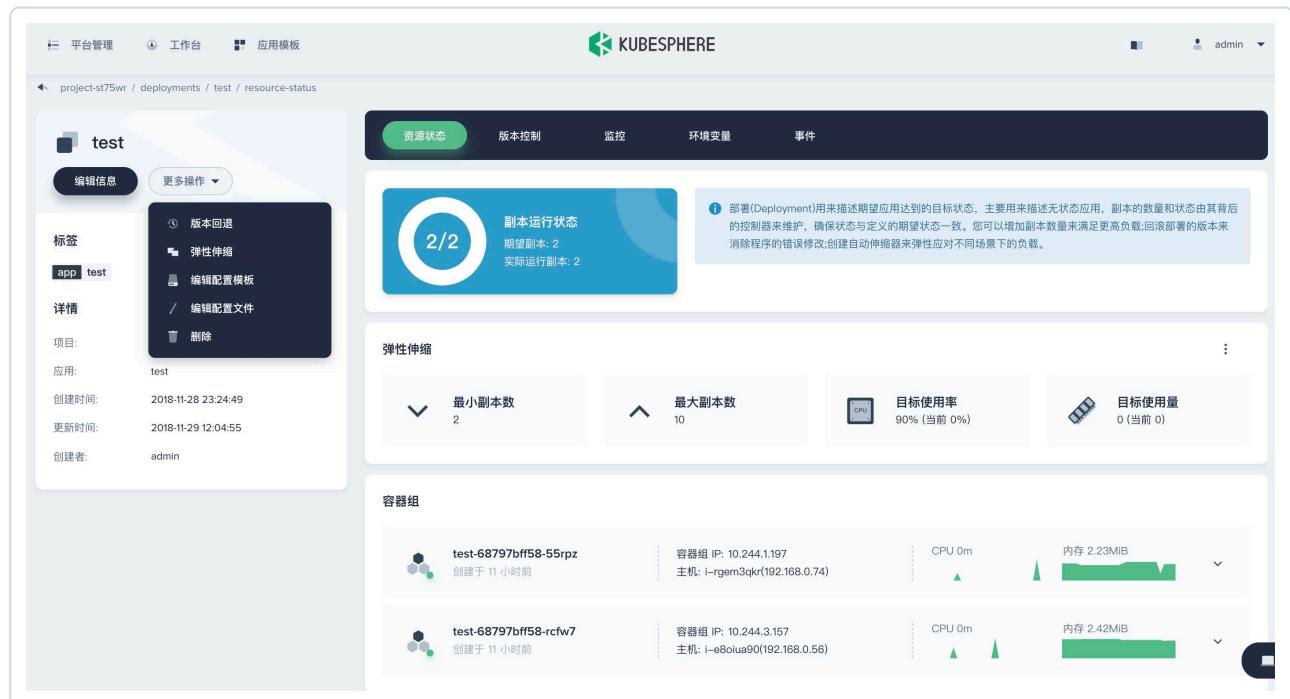
至此，您已经初步地熟悉了如何对容器设置健康检查器，若希望了解更高级的配置方法，可参考 [Kubernetes 官方文档](#)。

# 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,期望副本数: 2, 实际运行副本: 2. Below this, there's 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 targets. 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 metrics (CPU 0m, 内存 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。

project-st75wr / deployments / test / revision-control

test

编辑信息 更多操作

标签

app test

详情

项目: docs-demo  
应用: test  
创建时间: 2018-11-28 23:24:49  
更新时间: 2018-11-29 12:15:31  
创建者: admin

版本 #2 正在运行  
刚刚更新

#1 创建于 2018年 11月 28日 23:24:49

#2 创建于 2018年 11月 29日 00:15:04

## 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。

project-st75wr / deployments / test / monitors

test

编辑信息 更多操作

标签

app test

详情

项目: docs-demo  
应用: test  
创建时间: 2018-11-28 23:24:49  
更新时间: 2018-11-29 12:15:31  
创建者: admin

监控

CPU 使用量 (m)

选择时间范围

- 最近 10 分钟
- 最近 20 分钟
- 最近 30 分钟
- 最近 1 小时
- 最近 2 小时
- 最近 3 小时
- 最近 5 小时
- 最近 12 小时
- 最近 1 天
- 最近 2 天
- 最近 3 天
- 最近 7 天

自定义时间范围

开始时间: 2018-11-29 11:21:35

结束时间: 2018-11-29 12:21:35

时间间隔: 10 分钟

取消 确定

内存使用量 (MiB)

网络出口流量 (Bps)

test-68797bff58-55rpz test-68797bff58-rcfw7

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 deployment named 'Wordpress 网站'. On the left, there's a sidebar with deployment details like 'version 2', '编辑信息' (Edit Information), and '更多操作' (More Operations). The main 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 '事件' (Events). Under '环境变量', two containers are listed: 'wordpress-container-2' and 'wordpress'. Each container has environment variables: 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 deployment named 'busybox'. The left sidebar shows deployment details like 'busybox', '编辑信息' (Edit Information), and '更多操作' (More Operations). The main 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) (selected). The '事件' section lists two events under 'ScalingReplicaSet': one from 2018-11-29 11:54:21 scaling up to 1, and another from 2018-11-29 11:54:26 scaling down 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

ReplicaSet 可确保指定数量的pod“replicas”在任何设定的时间运行。然而，Deployments 是一个更高层次的概念，它管理ReplicaSets，并提供对pod的声明性更新以及许多其他的功能。因此，我们建议您使用Deployments而不是直接使用ReplicaSets，除非您需要自定义更新编排或根本不需更新。  
这实际上意味着您可能永远不需要操作ReplicaSet对象：直接使用Deployments并在规范部分应用此功能。

标签 编辑信息 更多操作

版本回退 弹性伸缩 编辑配置模板 编辑配置文件 删除

组件 标签 应用 项目 详情

app: harbor component: notary release: my-release

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

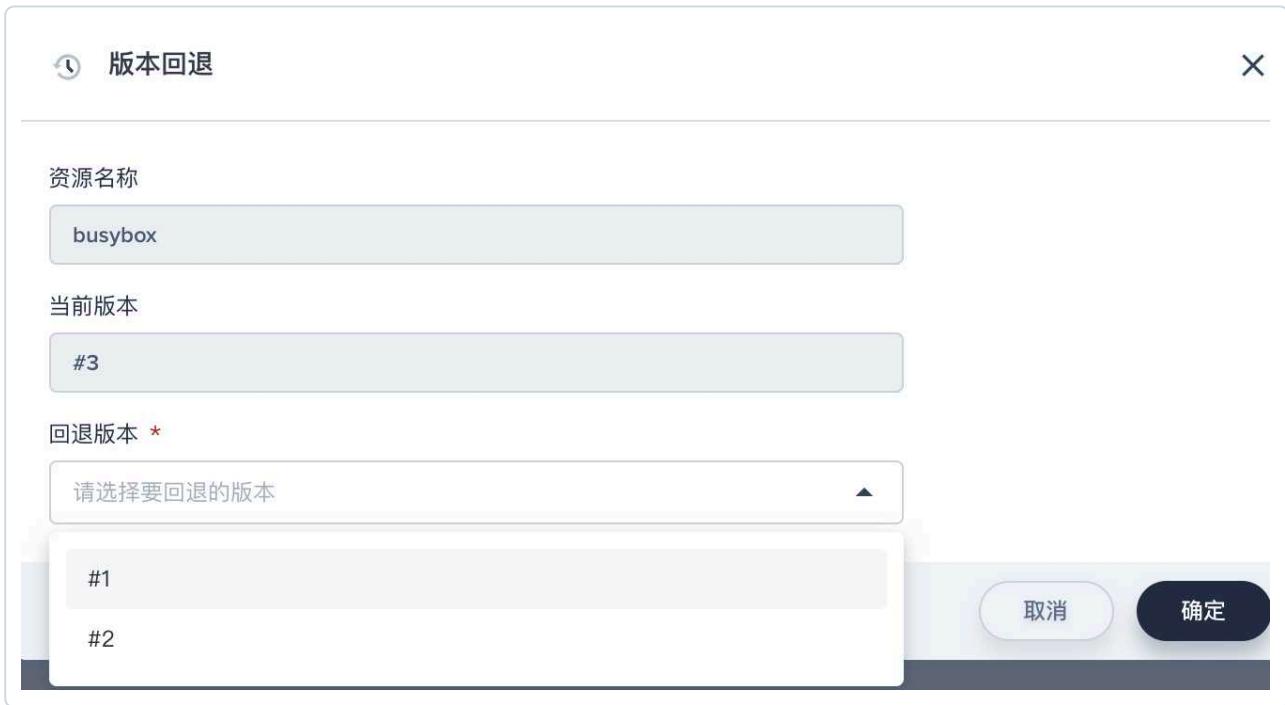
名称	协议	端口	主机端口
port	TCP	3306	-

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。

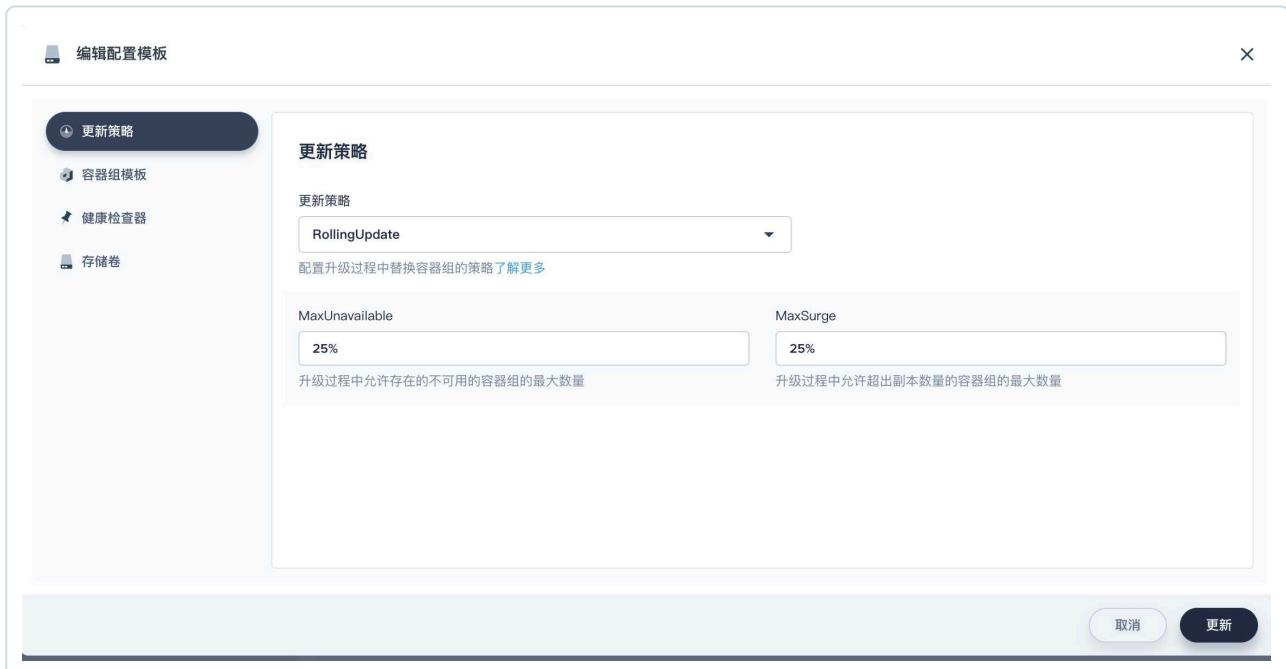


## 配置弹性伸缩

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 UI for managing a deployment named 'test'. On the left, there's a sidebar with deployment details like 'app: test', '项目: test', and creation time. The '更多操作' (More Operations) dropdown is open, and the '删除' (Delete) option is selected, indicated by a red arrow. On the right, the main panel displays the deployment status as '2/2' replicas running, with a note explaining what a deployment is. Below that, there's a '弹性伸缩' (Elastic Scaling) section with sliders for '最小副本数' (Min Replicas) set to 2, '最大副本数' (Max Replicas) set to 10, and '目标使用率' (Target Utilization) set to 90% (current 0%).

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。



# 自定义 S2i 模板

S2i (Source-to-image) 使用源代码和构建器镜像生成新的 Docker 镜像，在我们的项目当中提供了部分常用的构建器镜像，例如[Python](#)、[Java](#)，您也可以定义自己的构建器镜像（即 S2i 模板）扩展 S2i。

在详细介绍构建器影响之前，我们会先说明一下自定义 S2i 模板的步骤，以及工作原理和构建器镜像的作用。

1. 下载 S2i 构建器镜像（镜像提供构建脚本）。
2. 下载源代码。
3. 将源代码传入构建器镜像当中。
4. 运行构建器镜像所提供的 `assemble` 脚本进行应用构建。
5. 保存制品镜像。

构建器镜像需要有一些必要的内容才能完成所有工作。

首先由于构建器镜像负责构建应用程序，因此它必须包含构建和运行应用程序所有需要的库和工具。例如 Java 构建器镜像将安装 JDK、Maven 等，而 Python 构建器镜像则可能需要 pip 等工具。

其次构建器镜像需要提供脚本来执行构建和运行操作。这部分脚本在 S2I 当中为：

- `assemble` – 负责构建应用程序
- `run` – 负责运行应用程序

在以下的步骤中，我们将向您展示如何创建一个[Nginx](#) 服务的构建器镜像。

## 第一步：S2i CLI 构建项目目录

[S2i 命令行工具](#) 带有一个方便的命令，可以引导构建器所需的目录结构。如下安装 S2i CLI：

```
$ wget https://github.com/openshift/source-to-image/releases/download/v1.1.14/source-to-image-v1.1.14-874754de-linux-386.tar.gz
```

```
$ ls  
s2i source-to-image-v1.1.14-874754de-linux-386.tar.gz sti  
  
$ cp s2i /usr/local/bin
```

```
$ s2i create nginx-centos7 s2i-builder-docs
```

本文使用 `nginx-centos7` 作为构建器镜像的名字创建了初始目录，目录结构如下所示。

```
s2i-builder-docs/  
  Dockerfile – 一个标准的Dockerfile, 定义了构建器镜像。  
  Makefile – 用于测试和构建构建器镜像的帮助脚本。  
  test/  
    run – 测试脚本, 测试构建器镜像是否正常工作。  
    test-app/ – 用于测试的应用程序的目录  
  s2i/bin  
    assemble – 负责构建应用程序的脚本  
    run – 负责运行应用程序的脚本  
    usage – 负责打印构建器镜像用法的脚本
```

## 第二步：修改 Dockerfile

如下修改 `Dockerfile` 来定义构建器镜像。

`Dockerfile`

```
# nginx-centos7
FROM kubespheredev/s2i-base-centos7:1

LABEL maintainer="Runze Xia <runzexia@yunify.com>"

# 声明当前应用的版本
ENV NGINX_VERSION=1.6.3

LABEL io.k8s.description="Nginx Webserver" \
      io.k8s.display-name="Nginx 1.6.3" \
      io.kubesphere.expose-services="8080:http" \
      io.kubesphere.tags="builder,nginx,html"

# 安装nginx并且清理yum cache
RUN yum install -y epel-release && \
    yum install -y --setopt=tsflags=nodocs nginx && \
    yum clean all

# 修改nginx的默认开放端口
RUN sed -i 's/80/8080/' /etc/nginx/nginx.conf
RUN sed -i 's/user nginx;//' /etc/nginx/nginx.conf

# 将s2i的脚本复制到构建器镜像当中
COPY ./s2i/bin/ /usr/libexec/s2i

RUN chown -R 1001:1001 /usr/share/nginx
RUN chown -R 1001:1001 /var/log/nginx
RUN chown -R 1001:1001 /var/lib/nginx
RUN touch /run/nginx.pid
RUN chown -R 1001:1001 /run/nginx.pid
RUN chown -R 1001:1001 /etc/nginx

USER 1001

# 声明默认使用的端口
EXPOSE 8080

# 修改构建器的默认启动命令，以展示构建器镜像的用法
CMD ["/usr/libexec/s2i/usage"]
```

## 第三步 处理s2i构建器脚本

当我们完成了 `Dockerfile` 的定义，我们现在可以完成构建器镜像的其他部分。我们现在添加S2I脚本，我们将从 `assemble`（负责构建应用程序）开始。如下编辑 `assemble` 文件，在我们的例子中，它只是把 `nginx` 的配置文件以及静态内容复制到目标容器中：

### assemble

```
#!/bin/bash -e

if [[ "$1" == "-h" ]]; then
    exec /usr/libexec/s2i/usage
fi

echo "---> Building and installing application from source..."
if [ -f /tmp/src/nginx.conf ]; then
    mv /tmp/src/nginx.conf /etc/nginx/nginx.conf
fi

if [ "$(ls -A /tmp/src)" ]; then
    mv /tmp/src/* /usr/share/nginx/html/
fi
```

默认情况下，`s2i build` 将应用程序源代码放在 `/tmp/src` 目录中，在上面的命令当中，我们将应用源代码复制到了 `kubespheredev/s2i-base-centos7:1` 定义的工作目录 `/opt/app-root/src` 当中。

现在我们可以来处理第二个脚本 `run`（用于启动应用程序），在我们的例子当中，它只是启动 `nginx` 服务器：

### run

```
#!/bin/bash -e

exec /usr/sbin/nginx -g "daemon off;"
```

我们使用 `exec` 命令将执行 `run` 脚本替换为执行 `nginx` 服务器的主进程。我们这样做是为了让所有 `docker` 发出的信号都可以被 `nginx` 收到，并且可以让 `nginx` 使用容器的标准输入和标准输出流。

我们在例子当中被没有实现增量构建，因此我们可以直接删除 `save-artifacts` 脚本。

最后我们在 `usage` 脚本当中添加一些使用信息：

### usage

```
#!/bin/bash -e
cat <<EOF
This is the nginx-centos7 S2I image:
To use it, install S2I: https://github.com/kubesphere/s2i-operator
Sample invocation:
s2i build test/test-app kubespheredev/nginx-centos7 nginx-centos7-app
You can then run the resulting image via:
docker run -d -p 8080:8080 nginx-centos7-app
and see the test via http://localhost:8080
EOF
```

## 第四步 构建与运行

当我们完成了 `Dockerfile` 和 S2i 的脚本，我们现在修改一下 `Makefile` 当中的镜像名称：

### Makefile

```
IMAGE_NAME = kubespheredev/nginx-centos7-s2ibuilder-sample

.PHONY: build
build:
    docker build -t $(IMAGE_NAME) .

.PHONY: test
test:
    docker build -t $(IMAGE_NAME)-candidate .
    IMAGE_NAME=$(IMAGE_NAME)-candidate test/run
```

可以执行 `make build` 命令来构建我们的构建器镜像了。

```
$ make build
docker build -t kubespheredev/nginx-centos7-s2ibuilder-sample .
Sending build context to Docker daemon 164.9kB
Step 1/17 : FROM kubespheredev/s2i-base-centos7:1
--> 48f8574c05df
Step 2/17 : LABEL maintainer="Runze Xia <runzexia@unify.com>"
--> Using cache
--> d60ebf231518
Step 3/17 : ENV NGINX_VERSION=1.6.3
--> Using cache
--> 5bd34674d1eb
Step 4/17 : LABEL io.k8s.description="Nginx Webserver"      io.k8s.display-name="Nginx 1.6.3"
--> Using cache
--> c837ad649086
Step 5/17 : RUN yum install -y epel-release &&      yum install -y --setopt=tsflags=nodocs nginx &&
--> Running in d2c8fe644415

.....
.....
.....
Step 17/17 : CMD ["/usr/libexec/s2i/usage"]
--> Running in c24819f6be27
Removing intermediate container c24819f6be27
--> c147c86f2cb8
Successfully built c147c86f2cb8
Successfully tagged kubespheredev/nginx-centos7-s2ibuilder-sample:latest
```

可以看到我们的镜像已经构建成功了，现在我们执行 `s2i build ./test/test-app kubespheredev/nginx-centos7-s2ibuilder-sample:latest sample app` 命令来构建我们的应用镜像。

```
$ s2i build ./test/test-app kubespheredev/nginx-centos7-s2ibuilder-sample:latest sample-app
--> Building and installing application from source...
Build completed successfully
```

当我们完成了应用镜像的构建，我们可以在本地运行这个应用镜像看构建出的应用是否符合我们的要求：

```
$ docker run -p 8080:8080 sample-app
```

在浏览器中访问，可以看到我们的网页已经可以正常访问了：



## 第五步 推送镜像并在 KubeSphere 添加 S2i 模版

当我们在本地完成 S2I 构建器镜像的测试之后，就可以推送镜像到自定义的镜像仓库当中，并创建构建器模版 `yaml` 文件：

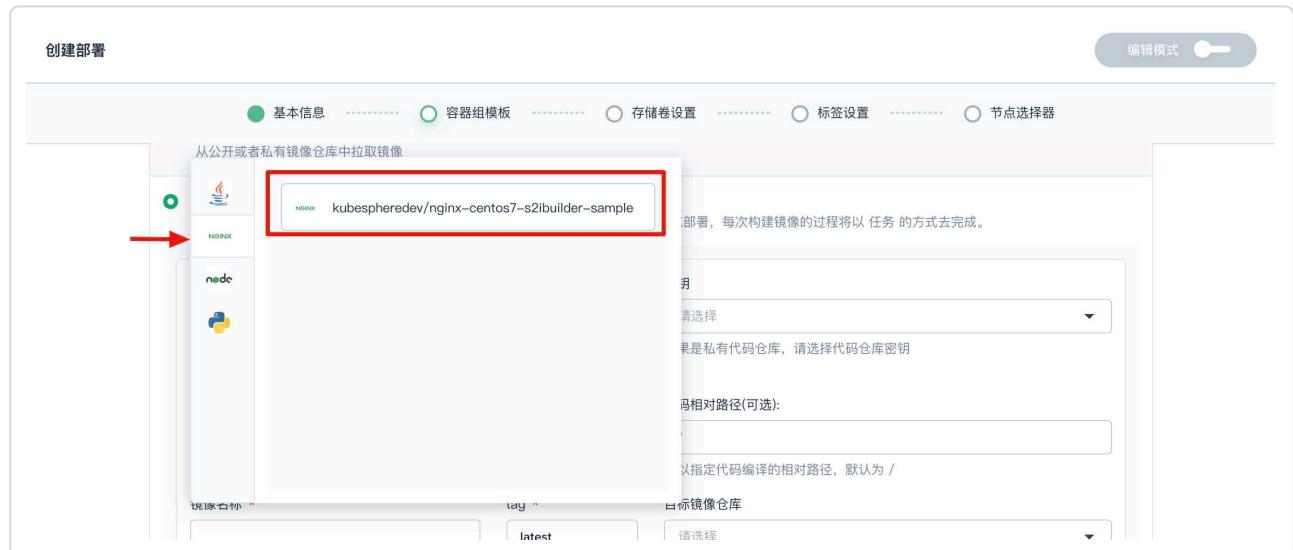
`s2ibuildertemplate.yaml`

```
apiVersion: devops.kubesphere.io/v1alpha1
kind: S2iBuilderTemplate
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: nginx-demo
spec:
  baseImages: # 构建器镜像名称，同一代码框架的多个不同版本。
  - kubespheredev/nginx-centos7-s2ibuilder-sample
  codeFramework: nginx # 代码框架类型
  defaultBaseImage: kubespheredev/nginx-centos7-s2ibuilder-sample # 默认使用的构建器镜像 (可替换为自
  version: 0.0.1 # 构建器模版的版本
  description: "This is a S2I builder template for Nginx builds whose result can be run directly without any f
```

在创建好构建器模版后我们可以使用 `kubectl` 将构建器模版提交到 KubeSphere 环境当中：

```
$ kubectl apply -f s2ibuildertemplate.yaml
s2ibuildertemplate.devops.kubesphere.io/nginx created
```

现在我们来到 KubeSphere 的控制台界面，我们已经可以选择添加的构建器模版了：



至此我们就完成了 S2i 构建器镜像与构建器模版的创建。类似地，可以参考上述步骤，您可以基于 S2i CLI 自定义任何所需的 S2i 模板，然后在 KubeSphere 构建所需的镜像并一键部署至 Kubernetes 环境

中，方便快速与多次构建环境。

# 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](#)。

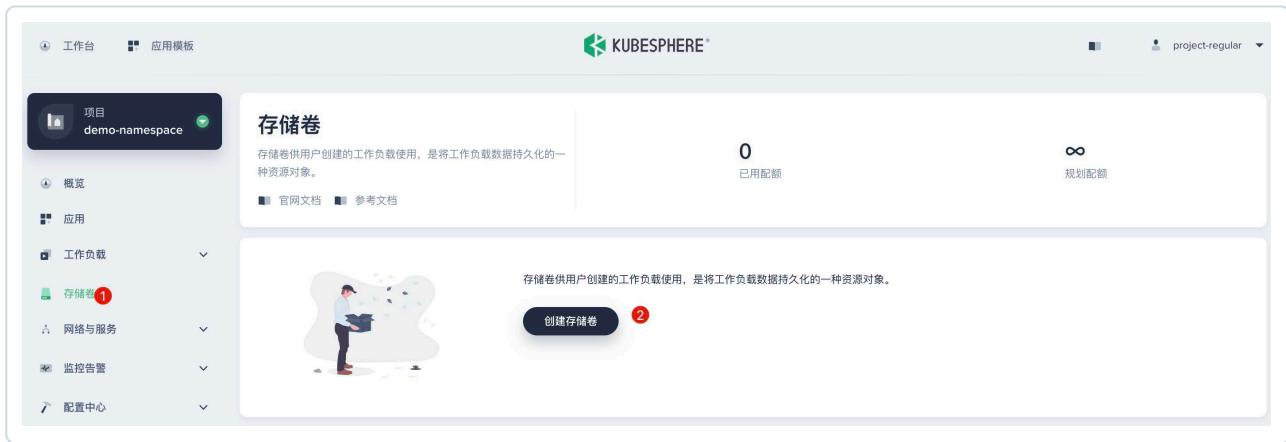
存储卷生命周期包括存储卷创建、挂载、卸载、删除等操作，如下演示一个存储卷完整生命周期的操作。

## 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。



## 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：



## 第二步：存储卷设置

存储设置中，选择存储卷的存储类型，存储类型需要预先创建，详见 [创建存储类型](#)。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](#)。

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。



### 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 remaining quota, 0 allocated storage, and 0 remaining storage. Below this is a search bar and a toolbar with a 'Create' button. The main table lists two storage 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示例。

### 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

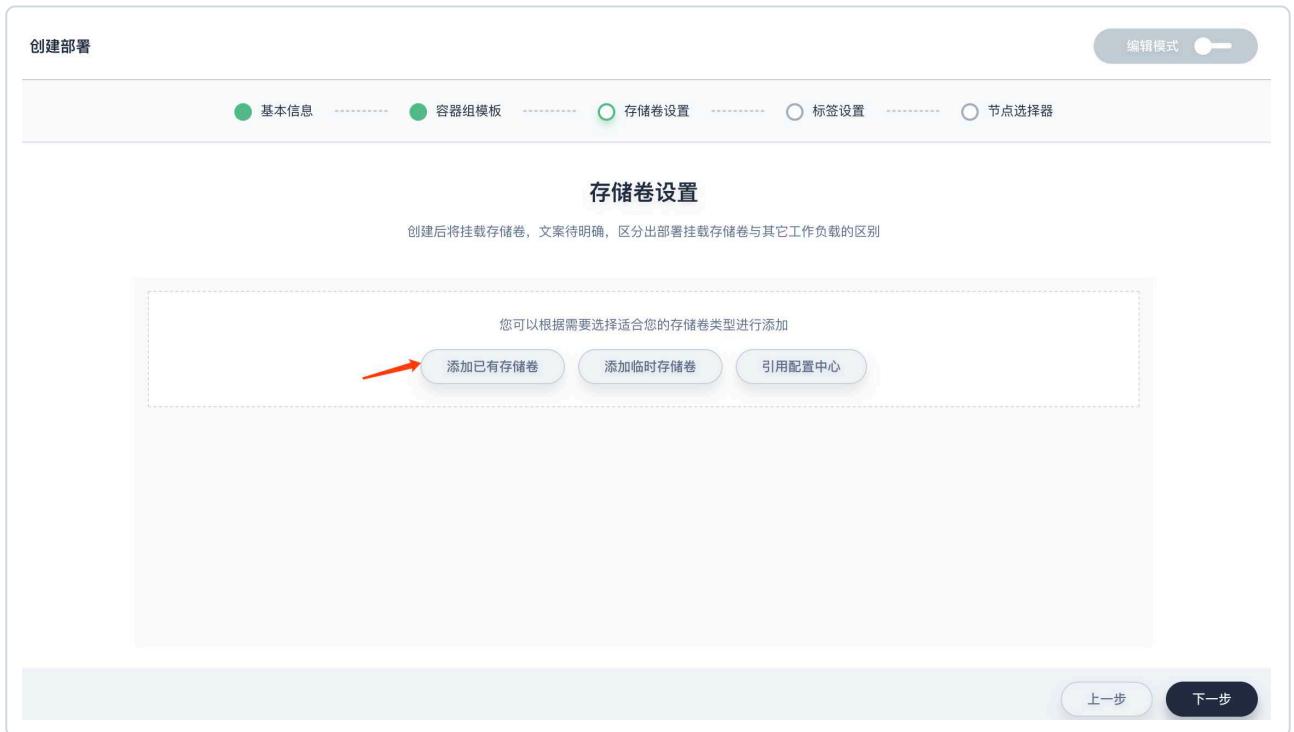
**添加容器**

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

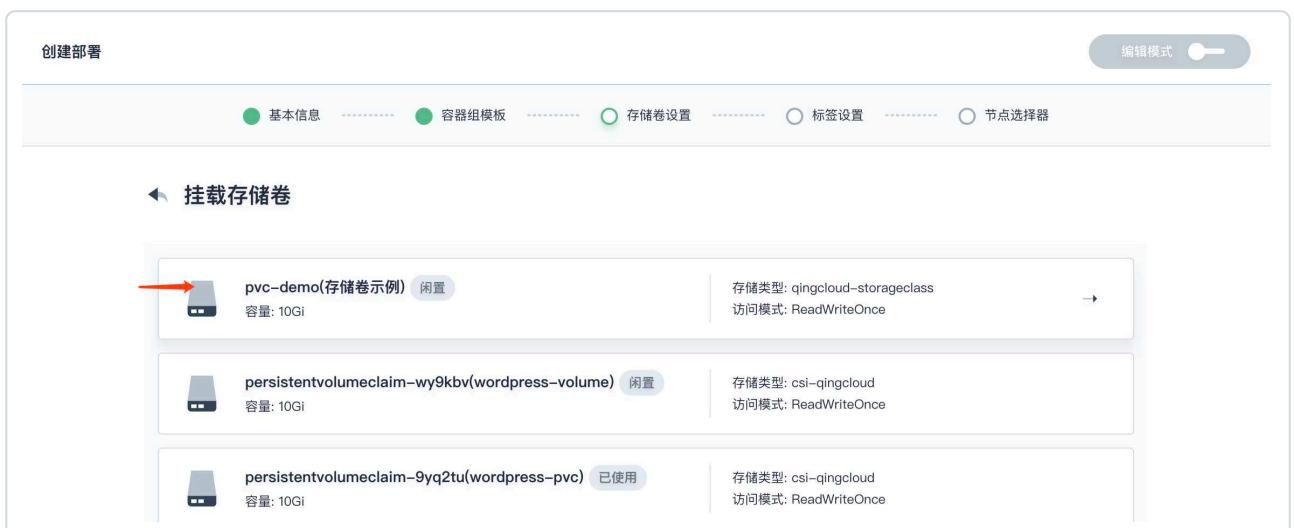
镜像 \*  要从私有镜像仓库部署，需要先 创建镜像仓库，然后拉取镜像。

## 第三步：添加存储卷

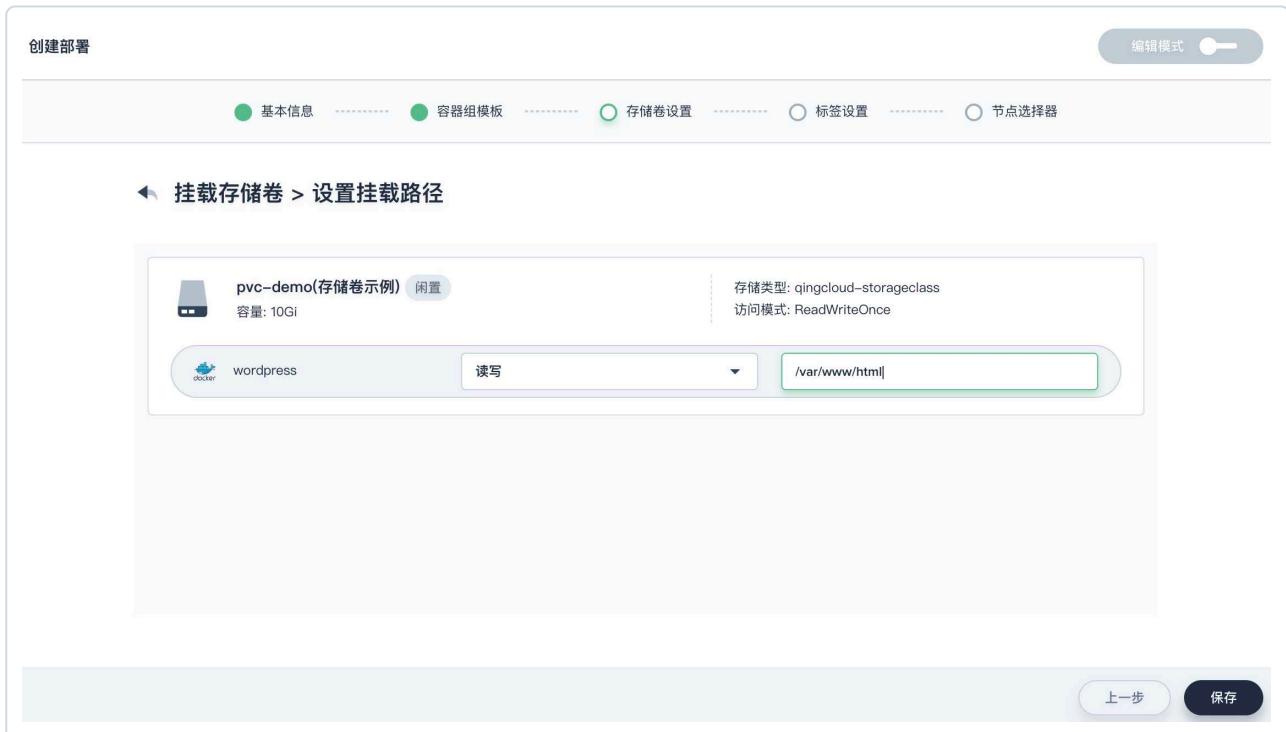
1、点击 **添加已有存储卷**。



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。



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`。



## 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**。

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

## 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。

## 前提条件

已创建了工作负载，若还未创建工作负载可参考 [工作负载](#)。

## 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。

创建服务支持三种方式，**页面创建**，**导入 yaml/json 文件**，**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。

The screenshot shows the KubeSphere service creation interface. On the left, there's a sidebar with navigation items: '项目 demo-namespace' (highlighted), '概览', '应用', '工作负载' (with a red '1' badge), '存储卷', '网络与服务' (with a red '1' badge), '服务' (highlighted), '灰度发布', and '应用路由'. The main content area has a title '服务' and a brief description of what a service is. It shows '0 已用配额' and '∞ 规划配额'. Below this is a large button labeled '创建服务' with a red '2' badge. There's also a small illustration of a person carrying a box.

## 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. At the top, there is a header '创建服务' and a 'Edit Mode' switch. Below the header, there are four tabs: '基本信息' (selected), '服务设置', '标签设置', and '外网访问'. The '基本信息' tab contains fields for 'Name' (必填), 'Namespace' (必填), 'Alias' (可选), and 'Description' (可选). The 'Name' field is set to 'service-demo', 'Namespace' is set to 'demo-namespace', and 'Description' is set to 'This is a demo'.

## 第二步：服务设置

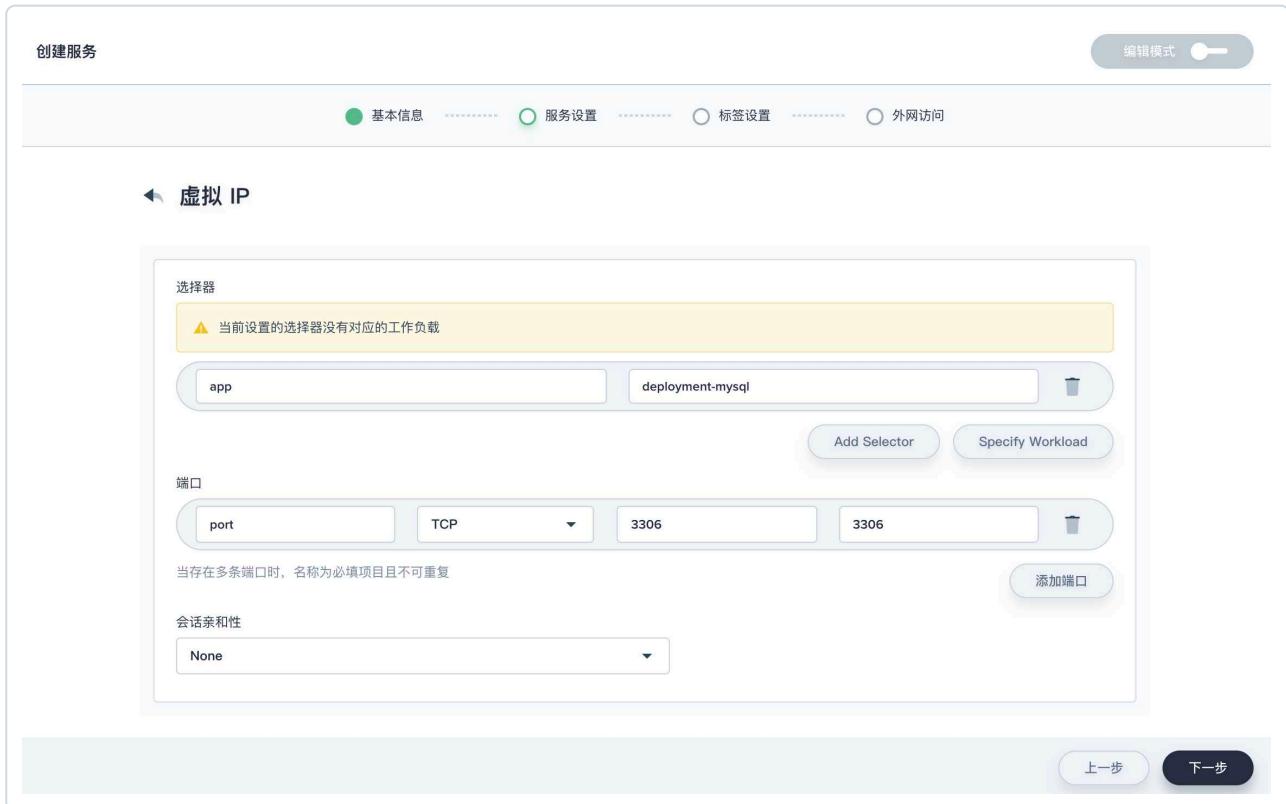
2.1. 选择需要创建服务的类型，每种服务类型适合不同的场景：

- 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)：将集群或者项目外部服务映射到集群或项目内。

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。

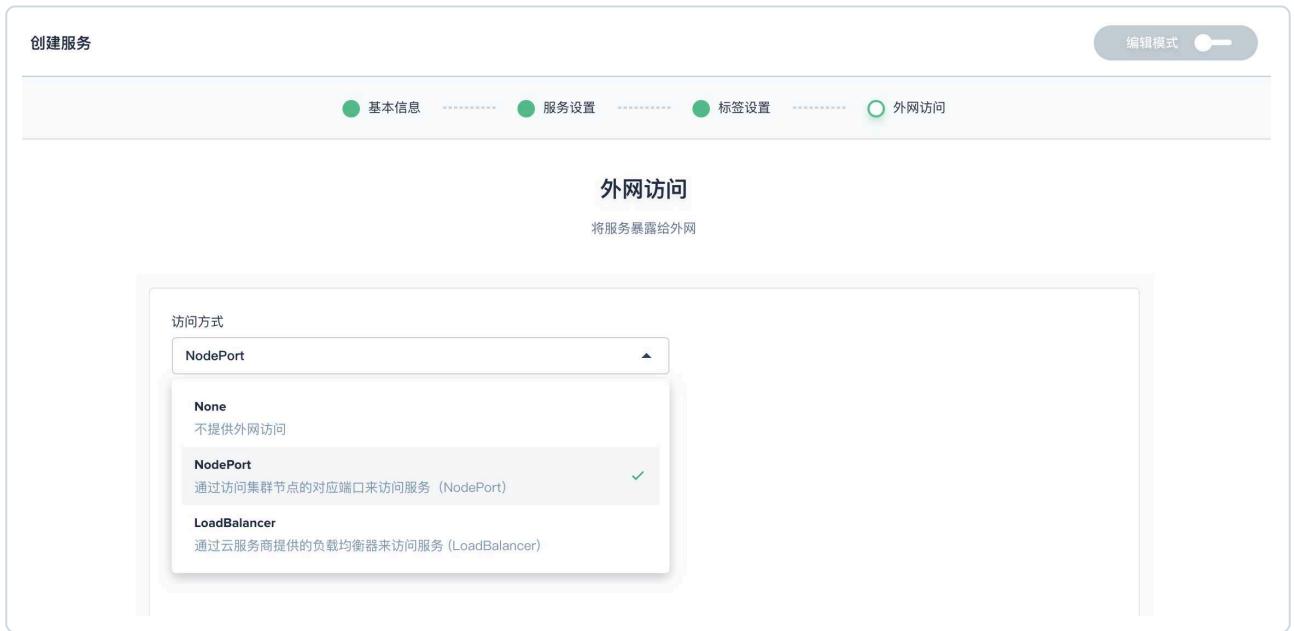


## 第四步：外网访问设置

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 需要在安装前配置与安装与云服务商对接的 cloud-controller-manage 插件，参考 [安装负载均衡器插件](#) 来安装和使用负载均衡器插件。



创建完成后即可在服务列表中查看成功创建的服务详情。

## 访问服务

服务创建后对外提供了不同的访问方式，可满足不同网络环境下提供不同的访问通道。

### 集群内部访问 (服务内部域名或 Virtual IP)

后端工作负载的服务暴露给集群内其他工作负载访问的方式，可通过服务在集群中的域名或 Virtual IP 访问。

- 其中集群内部域名格式为：`<服务名称>.<服务所在的项目名称>.svc.cluster.local`，例如 `nginx.demo-namespace.svc.cluster.local`，可通过 curl 命令进行验证。
- Virtual IP 的形式为：`Virtual IP:Port`，例如访问下图的 nginx：`curl 10.233.18.78:80`

名称	IP地址	端口	应用	创建时间
nginx	Virtual IP: 10.233.18.78	端口: 80:80/TCP 节点端口: 80 31639/TCP	-	2019-05-18 01:11:24

## 节点访问 (NodePort)

节点访问是在每个节点的 IP 上开放一个节点端口 (NodePort)，通过节点端口对外暴露服务。通过请求 `<节点 IP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。

## 负载均衡 (LoadBalancer)

通过云服务商提供的负载均衡器从公网来访问服务，由于使用 LoadBalancer 需要在安装前配置与安装与云服务商对接的 cloud-controller-manage 插件，参考 [安装负载均衡器插件](#) 来安装和使用负载均衡器插件。

安装完成后，在创服务的外网访问中，需要在 LoadBalancer 中添加两条 Annotation 并填入需要绑定的公网 IP 的 ID，然后就可以通过公网 IP 访问该服务。

```
service.beta.kubernetes.io/qingcloud-load-balancer-type      0  
service.beta.kubernetes.io/qingcloud-load-balancer-eip-ids    填写公网 IP 的 ID
```

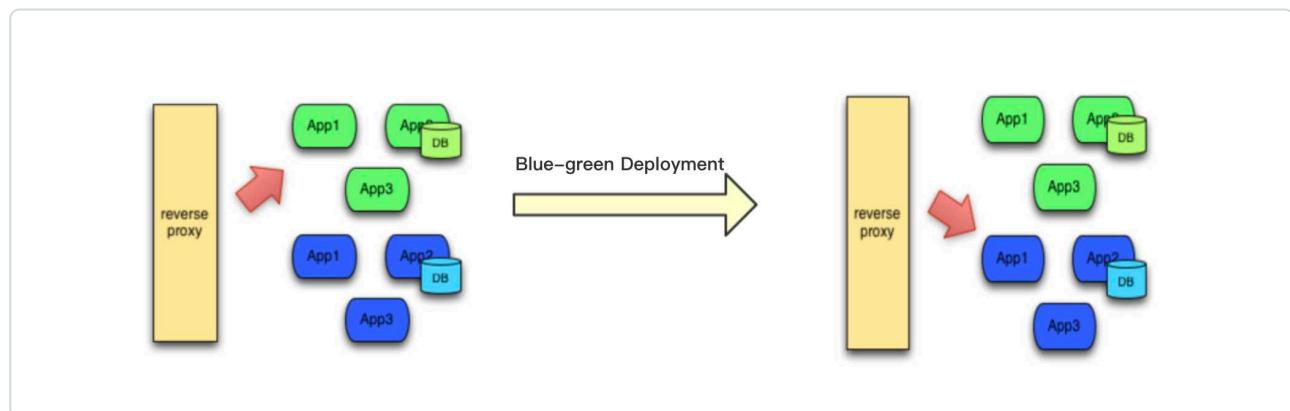


# 灰度发布

灰度发布是指在黑与白之间，能够平滑过渡的一种发布方式。灰度发布是互联网产品在快速迭代的过程中，安全地发布到生产环境的一种方法，包含多种发布策略。目前灰度发布提供了蓝绿部署、金丝雀发布、流量镜像三类灰度发布策略。

## 蓝绿部署

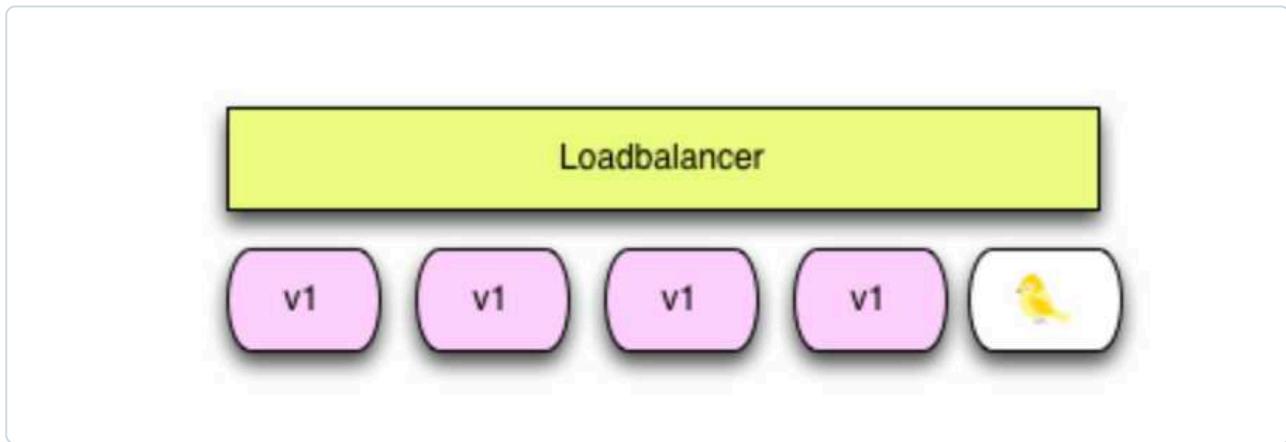
蓝绿发布提供了一种零宕机的部署方式，在保留旧版本的同时部署新版本，将两个版本同时在线，新版本和旧版本是相互热备的，通过切换路由权重 (weight) 的方式（非 0 即 100）实现应用的不同版本上线或者下线，如果有问题可以快速地回滚到老版本。



(图参考自 <https://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases>)

## 金丝雀发布

在生产环境运行的服务中引一部分实际流量对一个新版本进行测试，测试新版本的性能和表现，然后从这部分的新版本中快速获取用户反馈。



(图参考自 <https://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases>)

## 流量镜像

流量镜像功能通常用于在生产环境进行测试，是将生产流量镜像拷贝到测试集群或者新的版本中，在引导用户的真实流量之前对新版本进行测试，旨在有效地降低新版本上线的风险。流量镜像可用于以下场景：

- 验证新版本：可以实时对比镜像流量与生产流量的输出结果。
- 测试：生产实例的真实流量可用于集群测试。
- 隔离测试数据库：与数据处理相关的业务，可使用空的数据存储并加载测试数据，针对该数据进行镜像流量操作，实现测试数据的隔离。



# 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。本文档说明创建应用路由时需要设置的规则和参数，建议同时参考 [应用路由与服务示例](#) 进行实际操作。

## 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。

The screenshot shows the KubeSphere application routing interface. On the left, there is a sidebar with navigation items: Workstation (selected), Application Catalog, Project (demo-namespace), Overview, Application, Workload, Storage, Network & Services, Services, Gray Release, and Application Routing (marked with a red circle). The main content area has a title '应用路由' (Application Routing). It includes a brief description: '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。' Below this is a button labeled '创建应用路由' (Create Application Routing) with a red circle containing the number '2'. At the top right, there are filters for '外网访问' (External Network Access) and 'Internet IP'.

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 are tabs for '基本信息' (selected), '路由规则', '注解', and '标签设置'. A 'Edit Mode' switch is on the right. The '基本信息' tab contains fields for 'Name' (ingress-demo), 'Namespace' (demo-namespace), 'Alias' (应用路由示例), and 'Description' (Ingress demo). A note says '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' for the name field, and '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' for the namespace field.

## 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**。

创建应用路由

基本信息 路由规则 注解 标签设置

◀ 设置路由规则

HostName \*

demo.nip.io

协议

http

Paths \*

/ service-demo 80

添加 Path

### 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。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

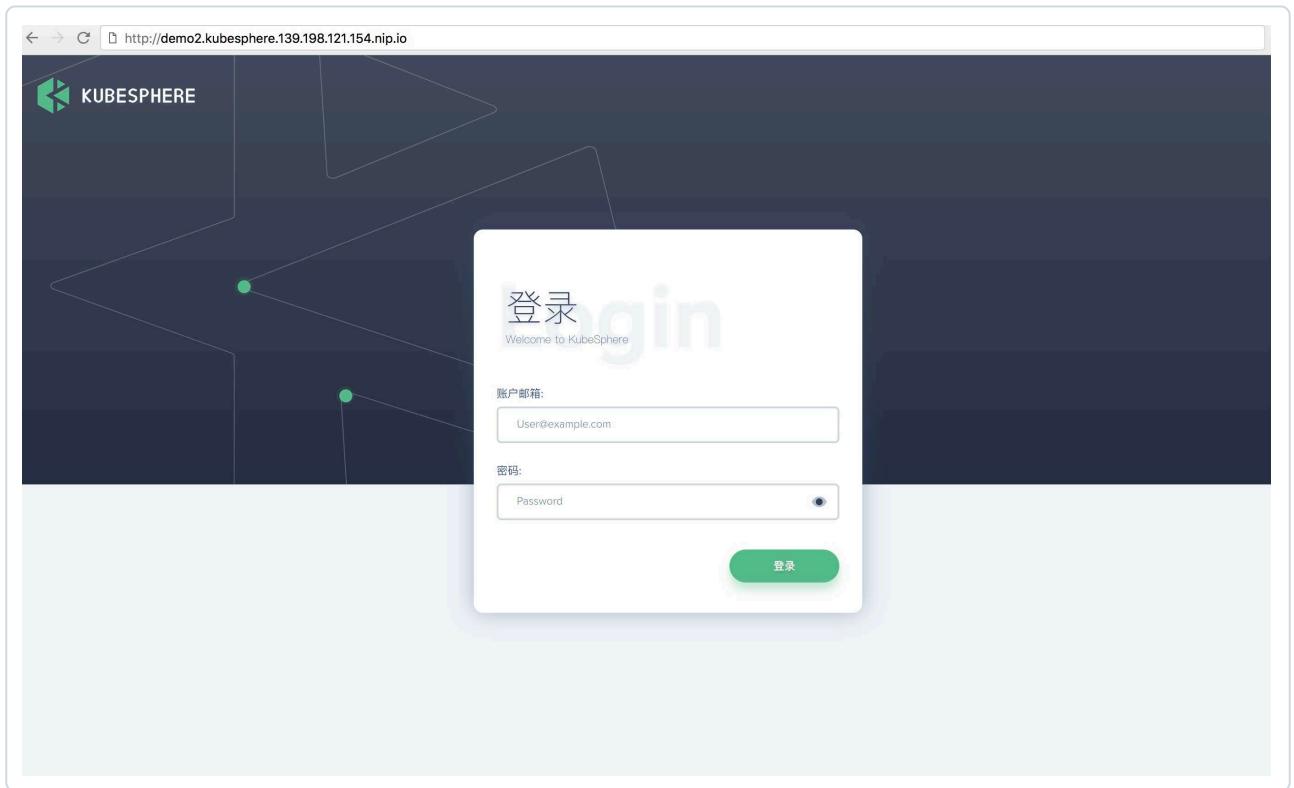
HostName: demo2.kubesphere.139.198.121.154.nip.io

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。



# 告警策略 —— 工作负载级别

## 目的

KubeSphere 提供节点和工作负载级别的告警策略，普通用户可以在项目中设置工作负载级别的告警策略。本篇文档以创建一个工作负载级别的告警策略并发送邮件通知作为示例，引导用户在项目中如何设置工作负载级别的告警策略。

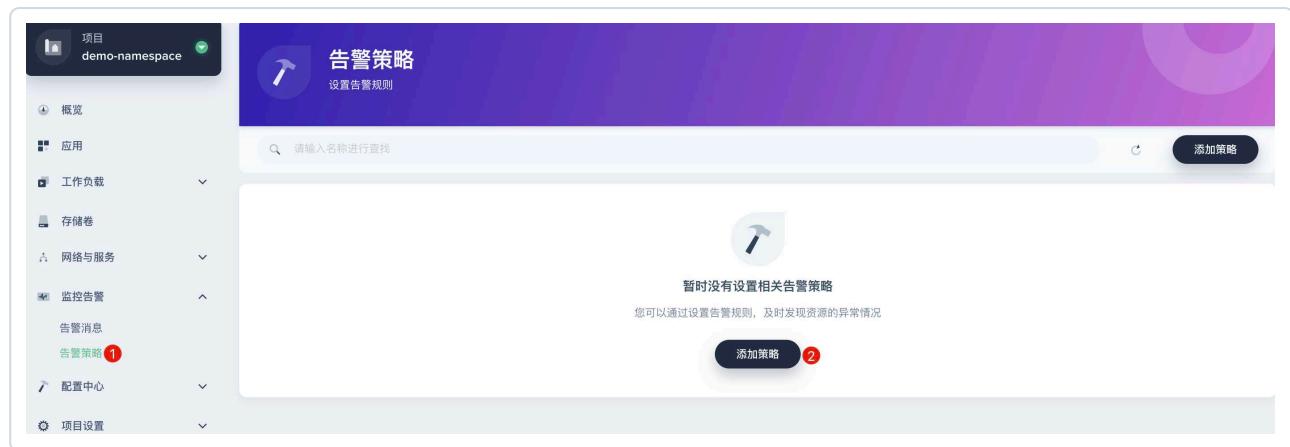
## 操作示例

### 前提条件

- 需由集群管理员预先配置邮件服务器，若还未配置可参考[邮件服务器](#)；
- 已创建了企业空间和项目并且创建了项目普通用户 project-regular 的账号，若还未创建请参考[多租户管理快速入门](#)；
- 项目中已有工作负载，若没有可在「应用」→「部署新应用」，选择「部署示例应用 Bookinfo」快速部署一个应用。

### 第一步：添加告警策略

以项目普通用户 `project-regular` 登录 KubeSphere，进入示例项目 `demo-namespace`，选择「监控告警」→「告警策略」，点击「添加策略」。



## 第二步：填写基本信息

在弹窗中，参考如下提示填写基本信息，完成后点击「下一步」。

- 名称：为告警策略起一个简洁明了的名称，便于用户浏览和搜索，比如 `alert-demo`；
- 别名：帮助您更好的区分资源，并支持中文名称，比如 `告警策略示例`；
- 描述信息：简单介绍该告警策略。

The screenshot shows the 'Basic Information' step of a KubeSphere alert configuration wizard. At the top, there are tabs for '基本信息' (selected), '监控目标', '告警规则', and '通知规则'. The main area is titled '基本信息' with the subtitle '设置告警策略的基础信息'. It contains four input fields: '名称' (Name) with value 'alert-demo', '别名' (Alias) with value '告警策略示例', '项目' (Project) with value 'demo-namespace', and '描述信息' (Description). Below each field is a help text or note.

名称 *	alert-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。	别名
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

## 第三步：选择监控目标

监控目标支持部署、有状态副本集、守护进程集三种工作负载，这里选择 `部署`，选择 `reviews-v1` 和 `details-v1` 作为监控目标，然后点击「下一步」。

The screenshot shows the 'Monitoring Targets' step of the alert configuration wizard. The title is '监控目标' with the subtitle '选择告警策略的监控目标'. There are three tabs at the top: '工作负载' (selected), '选择器', and a dropdown menu showing '部署'. Below is a table listing four workloads:

工作负载	运行时间	应用
reviews-v1 状态: 运行中(1/1)	12 小时	bookinfo 应用
details-v1 状态: 运行中(1/1)	12 小时	bookinfo 应用
ratings-v1 状态: 运行中(1/1)	12 小时	bookinfo 应用
productpage-v1 状态: 运行中(1/1)	12 小时	bookinfo 应用

## 第四步：添加告警规则

点击「添加规则」，本示例以设置 **内存用量** 作为告警指标，监控周期为 **1分钟/周期**，选择 **连续2次**，内存用量的阈值 > **20 MiB**，级别为重要告警，设置的规则如截图所示：

说明：工作负载支持的告警规则如下：

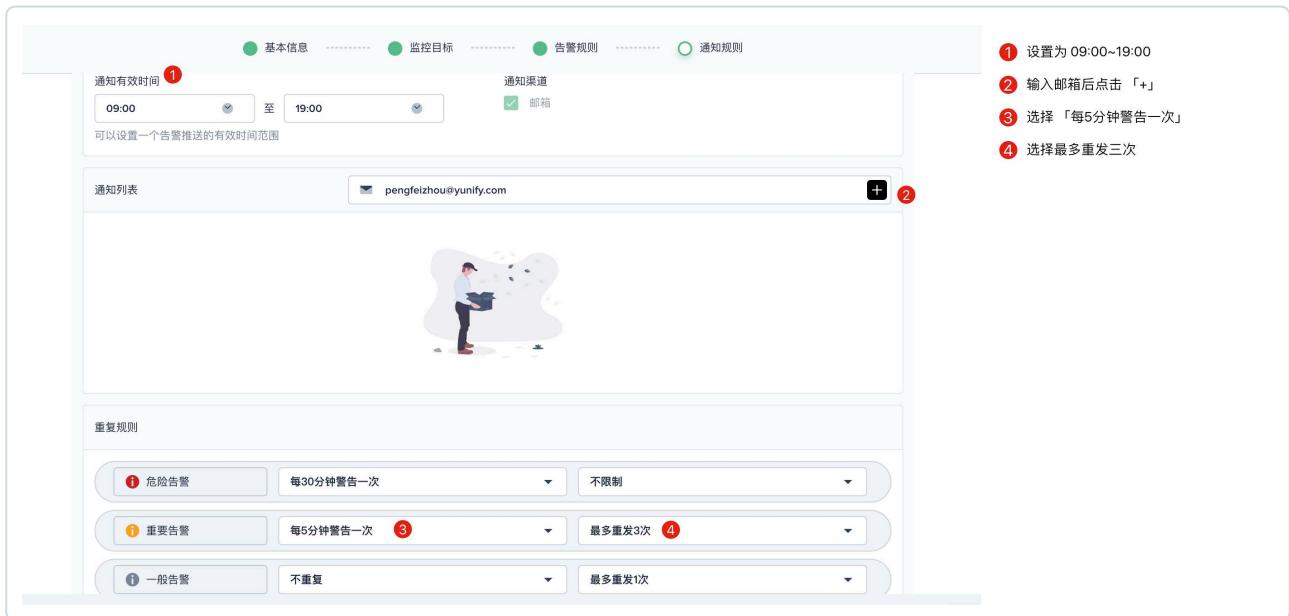
- CPU 用量；
- 内存用量 (包含缓存)；
- 内存用量；
- 网络：网络发送数据速率、网络接收数据速率；
- 工作负载指标：部署副本不可用率、有状态副本集副本不可用率、守护进程集不可用率 (工作负载的副本不可用率：比如对 Nginx 的部署设置 5 个副本后正常运行的副本状态是 5/5，如果部署不可用率设置了大于等于 20%，那么只要当副本运行状态为 4/5 的时刻就会发送告警)

完成后点击「保存」，然后点击「下一步」。

## 第五步：设置通知规则

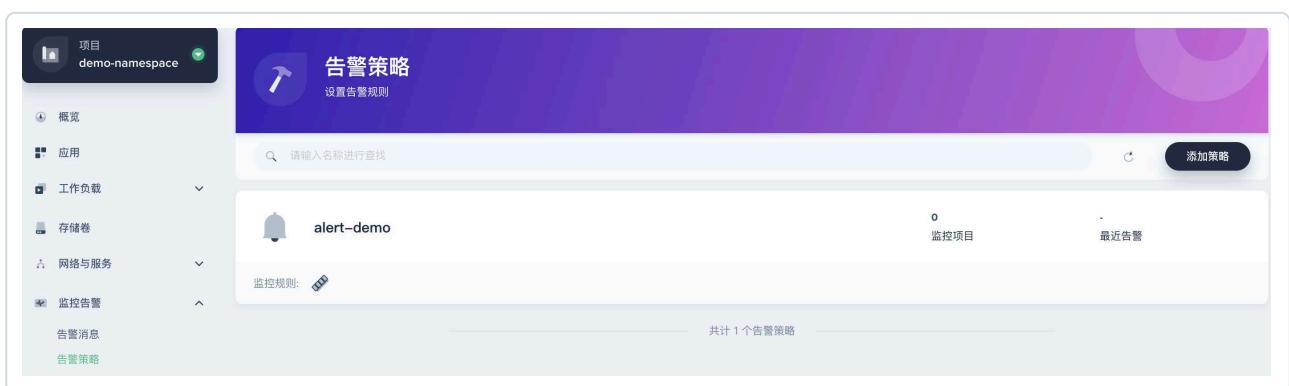
1. 通知有效时间可以设置发送通知邮件的时间范围，例如 **09:00 ~ 19:00**，通知渠道目前仅支持邮箱，在通知列表中输入需要通知的成员邮箱。
2. 重复规则设置的是告警通知的发送周期和重发频度，如果告警一直未解决，相隔一定的时间将会重

复发送告警。针对不同级别的告警也可以设置不同的重复规则，由于上一步设置的告警级别是重要告警，因此选择重要告警的规则为 **每 5 分钟警告一次，最多重发 3 次**。参考如下截图设置通知规则：



3. 点击「创建」，可以看到示例告警策略创建成功。

**说明：告警的等待时间 = 检测周期 × 连续次数。例如检测周期为 1 分钟/周期，连续次数为 2 次，那么需要等待 2 分钟。**



## 第六步：查看告警策略

告警策略创建成功后，点击进入 `alert-demo` 告警策略的详情页，查看告警规则当前的状态和详细信

息，包括监控目标、通知规则和告警历史等。

The screenshot shows the KubeSphere alerting rules interface. At the top, there's a navigation bar with '项目 / 告警策略 / 告警规则'. Below it, a header bar has tabs for '告警规则' (selected), '监控目标', '通知规则', and '告警历史'. On the left, there's a sidebar for 'alert-demo' with '编辑信息' and '更多操作' buttons. The main area is titled '告警规则' and contains a single rule card for '内存用量' (Memory Usage). The rule details are: '正在告警' (Alerting) with a red heart icon, '每 5 分钟通知一次' (Notify once every 5 minutes), '连续2次 (1分钟/周期) > 20MiB' (Consecutive 2 times (1 minute per cycle) > 20MiB), and an '重要告警' (Important Alert) icon. The rule was created on '2019-04-30 11:32:25'.

左侧点击「更多操作」→「更改状态」，支持启用或停用告警策略。

# 告警消息 —— 工作负载级别

告警消息记录了在工作负载级别的告警策略中，所有已发出的满足告警规则的告警信息。在[告警策略——工作负载级别](#)这篇文档中，演示了创建工作负载级别的告警策略并发送邮件通知，同时所有平台发出的告警消息都已经记录在了告警消息列表中，管理员可以进一步查看告警详情、监控指标、告警策略、最近通知和处理意见等详细信息。

## 前提条件

已创建了告警策略并收到了告警消息，若还未创建请参考[告警策略——工作负载级别](#)。

## 查看告警消息

1. 以项目普通用户 project-regular 账号登录 KubeSphere，进入示例项目 demo-namespace，选择「监控告警」→「告警消息」，
2. 点击「告警消息」，在告警消息列表中查看**全部**的告警消息。由于我们在告警策略的示例中设置的监控对象为 reviews-v1 和 details-v1，并且这两个示例工作负载的内存用量都大于告警的阈值 20 MiB，因此在告警消息列表中看到了 2 条与监控目标对应的告警消息。



告警消息	等级	类型	时间
details-v1 内存用量 > 20MiB	重要	工作负载告警	2019-04-30 11:33:35

3. 点击其中一条告警消息进入详情页，在告警详情中查看被监控工作负载的内存用量，可以看到在最近一段时间内监控对象的内存用量持续高于设定的阈值 20 MiB，因此触发了告警。

The screenshot shows the 'Alert Details' page for a memory usage alert. At the top, there are tabs for '告警详情' (Alert Details), '告警策略' (Alert Strategy), '最近通知' (Recent Notifications), and '处理意见' (Treatment Opinions). The '告警详情' tab is active. On the left, there's a sidebar with a title '内存用量' (Memory Usage) and a message 'details-v1 内存用量 > 20MiB'. Below this are sections for '详情' (Details), '告警等级' (Alert Level - Important), '告警状态' (Alert Status -待解决), '告警类型' (Alert Type - Workload Alert), and '告警时间' (Alert Time - 2019-04-30 11:33:35). On the right, the main area shows a chart titled '内存用量 (MiB)' (Memory Usage (MiB)) from 11:23 to 11:43. A red line at 20 MiB indicates the threshold, and a blue line shows the usage reaching 48.63 MiB at 11:26. A callout box highlights the event: '发生告警 内存用量 > 20 MiB' (Alert triggered: Memory usage > 20 MiB) at '2019-04-30 11:26'. Below the chart, a note says '重要 工作负载-部署: details-v1 内存用量 > 20MiB 告警时间: 2019-04-30 11:33:35'.

## 查看告警策略

切换到「告警策略」 查看本条告警消息对应的告警策略，可以看到该工作负载告警策略的触发规则正是在上一篇工作负载告警策略示例中设定的。

The screenshot shows the 'Alert Strategy' page for the same memory usage alert. The top navigation bar has tabs for '告警详情' (Alert Details), '告警策略' (Alert Strategy), '最近通知' (Recent Notifications), and '处理意见' (Treatment Opinions). The '告警策略' tab is active. On the left, it shows the alert details again. On the right, the '告警策略' section is expanded, showing a configuration for a 'Deployment告警策略' (Deployment Alert Strategy). It specifies a trigger rule: '触发规则- 满足以下任意条件' (Trigger Rule - Satisfy any of the following conditions). The configuration table includes columns for '监控项' (Monitoring Item), '周期' (Period), '连续次数' (Continuous Occurrences), '条件' (Condition), and '阈值' (Threshold). For this entry, the values are: '内存用量' (Memory Usage), '1分钟/周期' (1 minute per period), '连续2次' (2 consecutive times), '>' (greater than), and '20MiB'.

## 查看最近通知

点击「最近通知」 即可看到当前的通知人已收到了 3 条告警通知，因为当前监控工作负载的告警指标（内存用量）连续 2 次超过了阈值，通知规则设置的是 **每 5 分钟警告一次，最多重发 3 次**。

The screenshot shows the KubeSphere monitoring interface. On the left, there's a summary card for a memory usage alert: "details-v1 内存用量 > 20MiB". Below it, under "详情", are the following details:

- 告警等级: 重要
- 告警状态: 待解决
- 告警类型: 工作负载告警
- 告警时间: 2019-04-30 11:33:35

On the right, a navigation bar has tabs: 告警详情, 告警策略, 最近通知 (highlighted in green), and 处理意见. The "最近通知" section shows a table with one row, circled in red. The table columns are Time and Notifier.

时间	通知人
2019-04-30 11:33:35	pengfeizhou@yunify.com

## 验证邮件通知

登录通知邮箱即可看到 KubeSphere 的邮件服务器给通知人发送的告警消息。示例邮箱先后一共收到了 6 封邮件，这是因为告警目标设置的 2 个工作负载（部署）内存利用率都 **连续 2 次** 超过了阈值 20 MiB，并且告警的通知规则设置的是 **每 5 分钟警告一次，最多重发 3 次**。

## 添加处理意见

点击「处理意见」可以对当前告警进行处理，添加意见信息。例如，由于当前告警工作负载的内存用量高于阈值，所以我们可以添加一条信息：**需要对该部署提高默认的最大内存使用的配额。**

The screenshot shows the KubeSphere monitoring interface. On the left, there's a summary card for a memory utilization alert: "i-ydnp0qij 内存利用率 > 50%". Below it, under "详情", are the following details:

- 告警等级: 重要
- 告警状态: 待解决
- 告警类型: 主机告警
- 告警时间: 2019-04-17 18:58:09

On the right, a navigation bar has tabs: 告警详情, 告警策略, 最近通知, and 处理意见 (highlighted in green). The "处理意见" section shows a table with two rows. A red arrow points to the "处理意见" button in the alert summary card.

操作者	邮箱	时间	处理意见
admin	admin@kubesphere.io	2019-04-18 10:04:57	处理意见: 需要对该主机加污点，不允许新的 Pod 调度。
admin	admin@kubesphere.io	2019-04-18 10:02:00	处理意见: 已处理

# 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。

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	配置数量	创建时间
istio.default	istio.io/key-and-cert	3	2019-05-14 01:38:12
default-token-kjh6p	kubernetes.io/service-account-token	3	2019-05-14 01:38:11

## 创建 Secret

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。

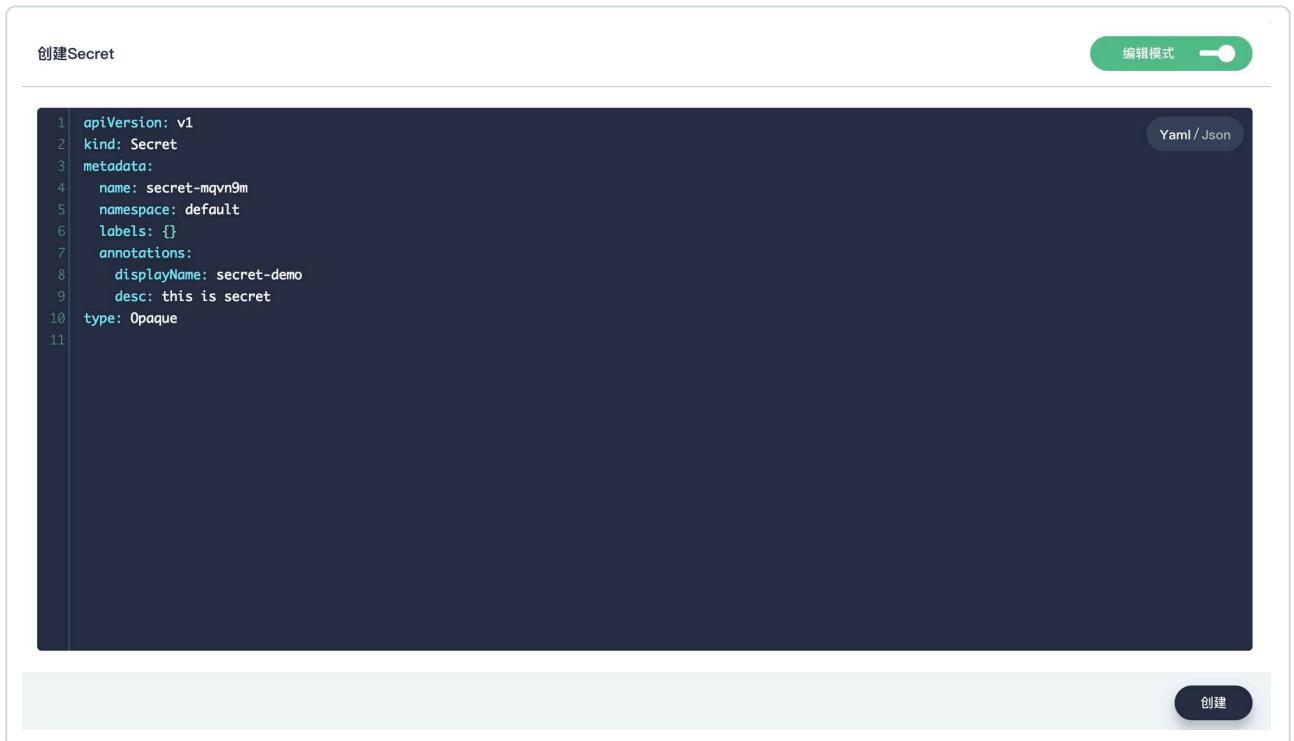
创建Secret

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

密钥支持以下五种类型：

- 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：

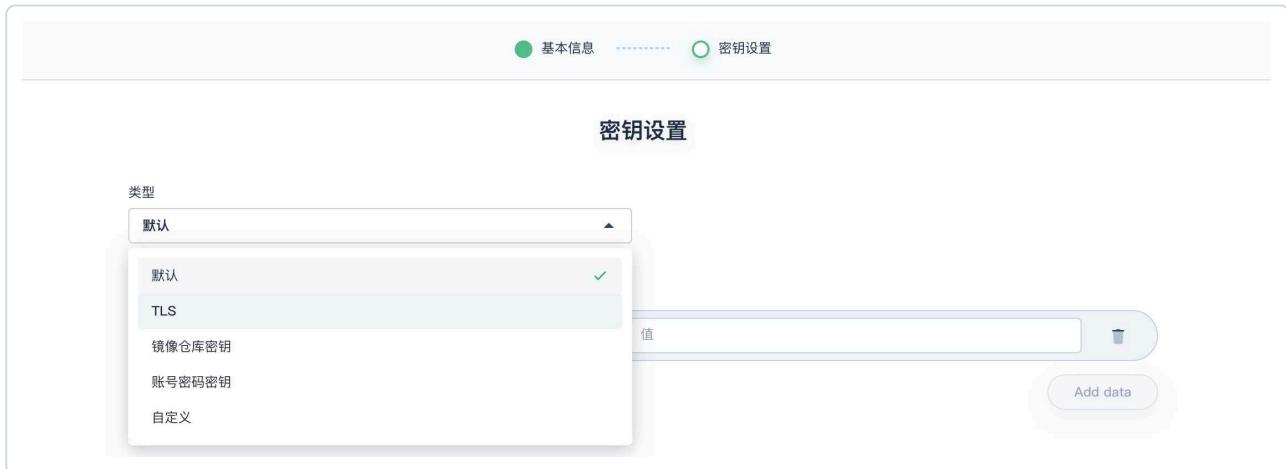
```
data:  
  password: hello123  
  username: guest
```

- 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
- 账号密码密钥：用来存储系统的账号密码，例如 DockerHub 或 GitHub。

- 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](#)。

## 创建常用的几类密钥

### 创建 DockerHub 密钥

1、以项目普通用户 `project-regular` 进入之前创建的项目 `demo-namespace`，在左侧的配置中心菜单下，点击 **密钥**，进入密钥管理界面。

The screenshot shows the KubeSphere interface for managing secrets. On the left, there's a sidebar with project navigation. The main area is titled 'Secrets' and displays a table of existing secrets. The table columns include 'Name', 'Type', 'Count', and 'Created Time'. Two secrets are listed: 'default-token-p4qml' (kubernetes.io/service-account-token) and 'istio.default' (istio.io/key-and-cert), both created on April 2, 2019, at 10:36:42. A red arrow points to the 'Create' button in the top right corner of the table header.

2、点击创建，创建一个用于 DockerHub 登录的密钥；

- 名称：必填，此名称作为改密钥的名称使用，此处命名为 **dockerhub-id**
- 别名：为了方便理解可自定义设置
- 描述信息：简单描述该密钥的用途等相关信息，可自定义

然后点击下一步

The screenshot shows the 'Basic Information' step of a secret creation wizard. It has two tabs: '基本信息' (selected) and '密钥设置'. The '基本信息' tab contains fields for '名称' (name), '项目' (project), '别名' (alias), and '描述信息' (description). The '名称' field is filled with 'dockerhub-id'. The '项目' dropdown is set to 's2i-test-sample'. The '别名' and '描述信息' fields are empty. A red arrow points to the '下一步' (Next Step) button in the bottom right corner.

- 类型：选择 **镜像仓库密钥**

- 仓库地址：填写 DockerHub 的仓库地址，如docker.io
- 用户名：填写您个人的 DockerHub 的用户名
- 密码：填写您个人的 DockerHub 的密码

3、完成后点击「创建」。

## 创建 GitHub 密钥

同上，创建一个用于 GitHub 的密钥，凭证 ID 命名为 **github-id**，类型选择 **账号密码密钥**，输入您个人的 GitHub 用户名和密码，完成后点击 **确定**。

# 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。

The screenshot shows the KubeSphere web interface. On the left, there is a sidebar with various project management options like Overview, Applications, Workloads, Storage, Networks & Services, Monitoring, Configuration Center, Secrets, and Project Settings. The 'Configuration' section is highlighted with a red circle containing the number '1'. In the main content area, the 'Configuration' page is displayed. It features a large central panel with a heading '配置' (Configuration), a brief description about managing environment variables, and a 'Create Configuration' button. Above this panel, there are two status indicators: '0 已用配额' (0 used quota) and '∞ 规划配额' (Infinite planned quota). The top navigation bar includes tabs for Workstation, Application Catalog, and the KubeSphere logo, along with a dropdown for project selection.

## 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

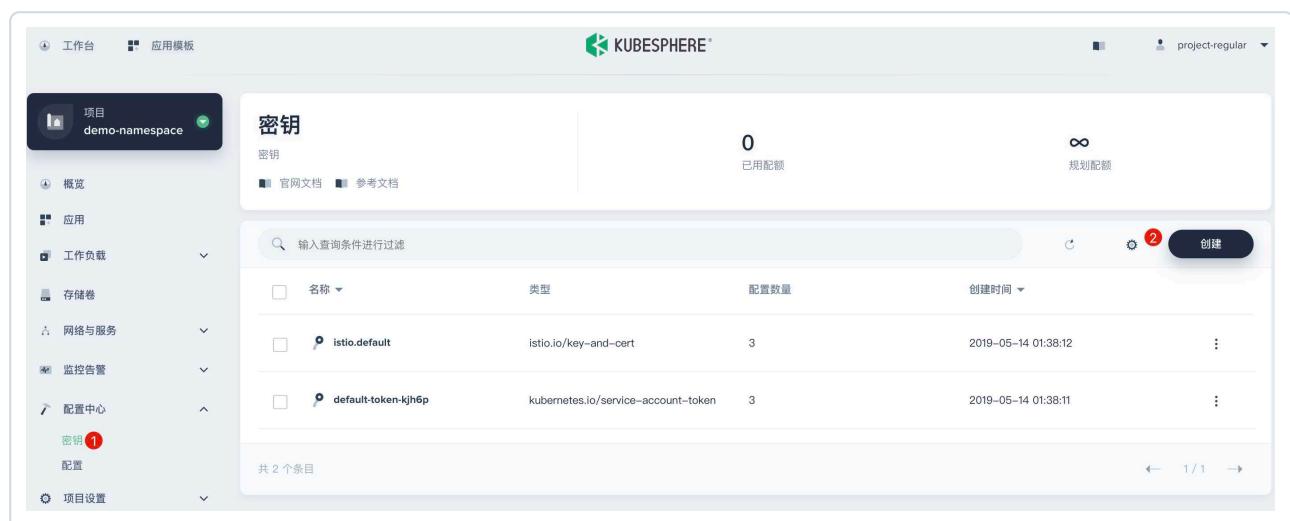
Docker 镜像是一个只读的模板，可用于部署容器服务，每个镜像有特定的唯一标识（镜像的 Registry 地址 + 镜像名称+镜像 Tag）。例如：一个镜像可以包含一个完整的 Ubuntu 操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。而镜像仓库是集中存放镜像文件的场所，镜像仓库用于存放 Docker 镜像。

## 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**。



名称	类型	配置数量	创建时间
istio.default	istio.io/key-and-cert	3	2019-05-14 01:38:12
default-token-kjh6p	kubernetes.io/service-account-token	3	2019-05-14 01:38:11

## 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. At the top right is a 'Edit Mode' switch. Below it, two tabs are visible: 'Basic Information' (selected) and 'Key Settings'. The 'Basic Information' section contains fields for 'Name' (dockerhub-qingcloud), 'Project' (project-st75wr), 'Alias' (QingCloud 镜像仓库), and 'Description' (QingCloud DockerHub Demo). A note below the alias field states: 'Alias can be composed of any character, helping you better distinguish resources and support Chinese names.'

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. At the top right is a 'Edit Mode' switch. Below it, two tabs are visible: 'Basic Information' (selected) and 'Key Settings'. The 'Key Settings' section contains a 'Type' dropdown set to 'Image Repository Key', a note stating 'You can also define a custom key type', and fields for 'Repository Address' (dockerhub.qingcloud.com), 'Username' (guest), 'Email' (demo@kubesphere.io), and 'Password' (a masked password).

3、点击 **创建**，即可查看创建结果。

The screenshot shows the 'Certificates' page within the KubeSphere interface. On the left, there's a sidebar with navigation items like '概览', '应用', '工作负载', '存储卷', '网络与服务', and '配置中心'. The '密钥' item is currently selected. The main area has a title '密钥' with counts '0 已用配额' and '∞ 规划配额'. Below is a search bar and a table with columns: '名称' (Name), '类型' (Type), 'Config Number', and '创建时间' (Created Time). Two entries are listed:

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

## 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。

This screenshot shows the 'Create Certificate' form. At the top, it says '创建密钥' and has an 'Edit Mode' switch. Below is a tab navigation with '基本信息' (Basic Information) and '密钥设置' (Key Settings), where '密钥设置' is selected. The '密钥设置' section contains fields for '类型' (Type, set to '镜像仓库密钥'), '仓库地址\*' (Repository Address, set to 'docker.io'), '用户名\*' (Username, set to 'kubesphere'), '邮箱' (Email, set to 'demo@kubesphere.io'), and '密码\*' (Password, masked).

## 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=10.233.130.1
--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'. On the left, there's a sidebar with a back arrow and the text '添加容器'. The main form has fields for '容器名称 \*' (Container Name) containing 'mysql', '镜像 \*' (Image) containing 'dockerhub.qingcloud.com/mysql:5.6', and a dropdown for '镜像仓库' (Image Registry) with options 'dockerhub' (selected), 'docker.io', and 'dockerhub-qingcloud'. A note below the container name says: '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'.

## 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. On the left, there's 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, 监控告警, 配置中心, 项目设置, 基本信息 (highlighted with a red arrow), 成员角色, 项目成员, and 外网访问. The main content area has a header with the project name 'demo-namespace', a user 'admin' (项目管理员), a workspace 'demo-workspace' (企业空间), and a creation time '2019-05-14 01:38:11'. Below the header, there are two sections: '基本信息' and '配额信息'. The '基本信息' section includes icons for workspace (demo-workspace, 企业空间), members (2 成员), and roles (3 成员角色). The '配额信息' section lists two items: '部署' (Deployment) with '规划配额: 无限制', '已用配额: 0', and '剩余配额: 无限制'; and '有状态副本集' (StatefulSet) with '规划配额: 无限制', '已用配额: 0', and '剩余配额: 无限制'.

## 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

使用项目管理员账号 `project-admin` 登录 KubeSphere 可进行配额信息管理。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		200Mi
CPU	内存		

默认最小使用资源

	10m		10Mi
CPU	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 '添加容器' (Add Container) section, 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 note about mirrors: '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) and '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' (To deploy from a private image repository, you need to first create an image repository and then pull the image).

Resource settings for the container group template are shown in a red-bordered box:

- CPU**:
  - 最小使用 (Minimum Usage): 10 m
  - 最大使用 (Maximum Usage): 100 m
- 内存 (Memory)**:
  - 最小使用 (Minimum Usage): 10 Mi
  - 最大使用 (Maximum Usage): 200 Mi

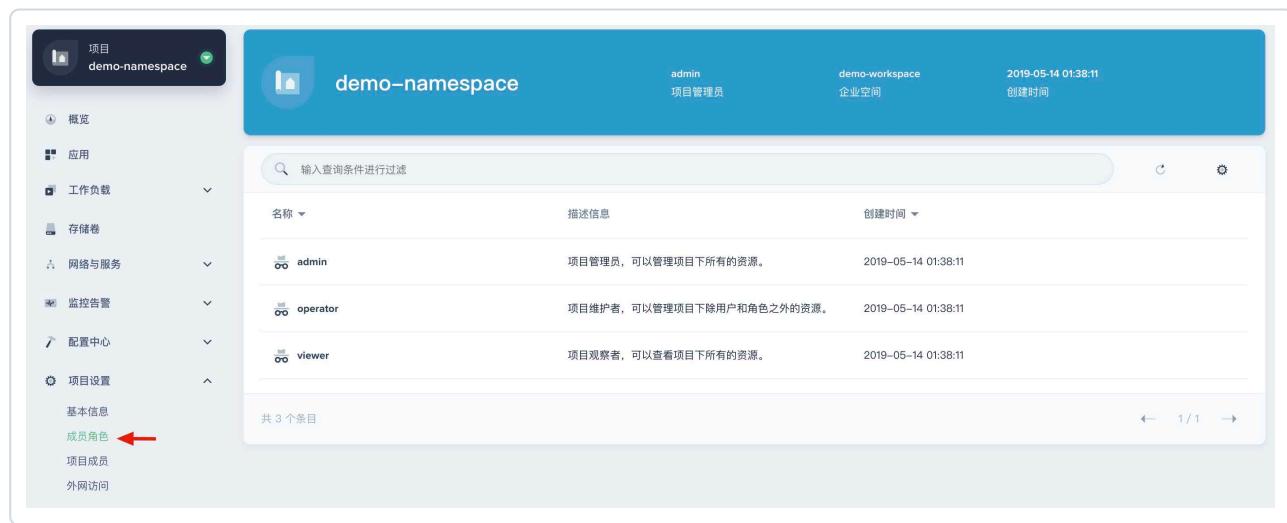
Below these fields, there are notes: '作为容器调度时资源分配的判断依赖。只有当节点上可分配CPU总量 ≥ 容器CPU最小使用值时，才允许将容器调度到该节点。单位换算规则: 1核 = 1000m' (As a dependency for resource allocation during pod scheduling. Only if the total allocatable CPU on the node is ≥ the container's minimum usage, will the pod be scheduled to the node. Unit conversion rule: 1 core = 1000m) and '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 pod scheduling. Only if the total allocatable memory on the node is ≥ the container's minimum usage, will the pod 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

使用项目管理员账号 `project-admin` 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。



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, Networks & Services, Monitoring, Configuration Center, Project Settings, Basic Information, and Member Roles. A red arrow points to the 'Member Roles' option under Project Settings. The main content area is titled 'demo-namespace' and displays a table of existing roles:

名称	描述信息	创建时间
admin	项目管理员，可以管理项目下所有的资源。	2019-05-14 01:38:11
operator	项目维护者，可以管理项目下除用户和角色之外的资源。	2019-05-14 01:38:11
viewer	项目观察者，可以查看项目下所有的资源。	2019-05-14 01:38:11

At the bottom of the table, it says '共 3 个条目' (3 items) and has navigation arrows for '1 / 1'.

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。

创建成员角色

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。

创建成员角色

权限设置

项目管理

查看  编辑  删除

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

有状态副本集

查看  创建  编辑  删除  
 横向伸缩

## 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。



## 修改角色权限

进入角色详情页面，点击 **编辑信息** 按钮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](#)。

使用项目管理员账号 `project-admin` 登录 KubeSphere，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 web interface. On the left, there's a sidebar with various project management options like Overview, Applications, Workload, Storage, Network & Services, Configuration Center, Project Settings, Basic Information, Member Roles, and Project Members. The 'Project Members' option is highlighted with a red arrow. The main content area is titled 'demo-namespace' and shows two members: 'admin' and 'project-admin'. Each member entry includes their name, email, status (Active), role (admin), and last login time (Never). At the top right of the member list, there's a 'Invite Member' button with a red arrow pointing to it.

## 邀请成员

点击 **邀请成员**，在弹窗中选择用户点击 **+**，并为该用户授予角色，即可邀请该成员加入项目。项目中内置了常用的三类角色，若这三类角色不满足需求，可自行创建角色并自定义它的权限规则，参考 [成员角色](#)。

## 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**

test@test.com

请选择一个角色赋予该成员



workload-operator

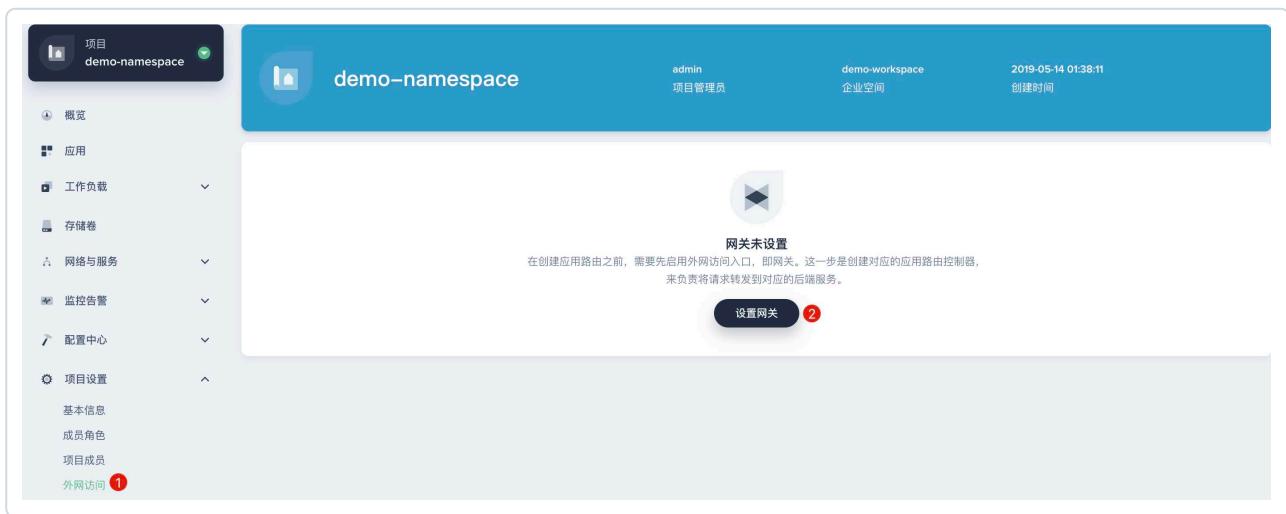


**calvin**

Ok

# 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。
- 应用治理: 通过 Istio 提供完整的非入侵式的微服务治理解决方案，可以有效的解决云原生应用的网络治理问题，提供熔断、限流及灰度发布等功能。

**注意：由于使用 Load Balancer 需要在安装前配置与安装与云服务商对接的 cloud-controller-manage 插件，参考 [安装负载均衡器插件](#) 来安装和使用负载均衡器插件。**

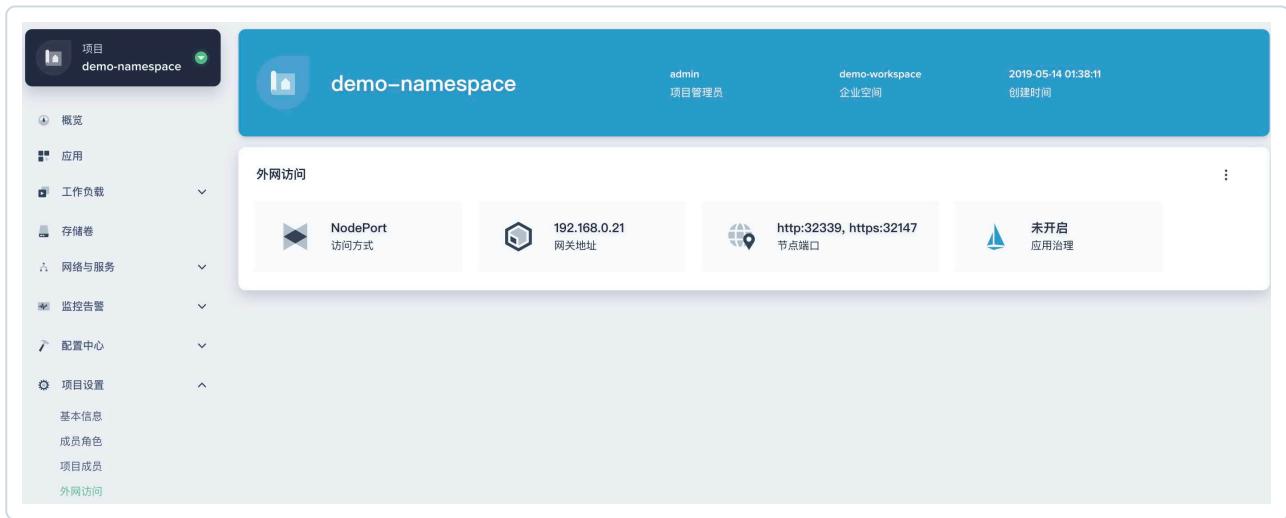


3. 点击 **保存** 来创建网关。

## 通过 NodePort 暴露网关

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>



## 通过 LoadBalancer 暴露网关

**前提条件：**若 KubeSphere 部署在 QingCloud 云平台，确保已安装了 [QingCloud LB 插件](#)。

若选择的是 LoadBalancer，负载均衡器插件将自动在云平台创建负载均衡器并绑定公网 IP 至网关，提供内部服务和应用路由的快速访问，最终可通过公网 IP 或域名的方式访问应用路由和服务。

在「项目设置」→「外网访问」下，点击「设置网关」，选择 LoadBalancer。

LB 插件可通过添加 Annotation 来配置使用，目前支持以下三种配置方式，参考 [公网 IP 配置](#)。

- 手动配置 EIP：**添加一条 Annotation 的 key 是 `service.beta.kubernetes.io/qingcloud-load-balancer-eip-ids`，值是 `公网IP 的 ID`；
- 自动获取 EIP：**需添加一条 Annotation 的 key `service.beta.kubernetes.io/qingcloud-load-balancer-eip-source`，注意自动获取 EIP 支持以下三种方式 (对应不同的值)：
  - 自动获取当前账户下处于可用的 EIP，如果找不到返回错误，值设置为 `use-available`；
  - 自动获取当前账户下处于可用的 EIP，如果找不到则申请一个新的，值设置为 `auto`；
  - 不管账户下有没有可用的 EIP，申请一个新 EIP，值设置为 `allocate`；
- 配置多个服务 (Service) 共享一个公网 IP：**由于 EIP 是稀缺资源，QingCloud-LB 插件提供了多个 Service 共享一个 EIP 的模式，使用这个模式有一定限制，详见 [配置多个Service共享一个EIP](#)。

以下仅以其中一种配置方式“自动获取当前账户下处于可用的 EIP，如果找不到则申请一个新的，值设置为 `auto`”演示如何在 KubeSphere 配置使用。

**注意，示例中的这种方式可能会造成同一账号下其他用户空闲的公网 IP 被使用。**



1、如图在示例中添加一条注解，保存后即可允许 LB 插件自动为网关创建和绑定公网 IP，同时在 QingCloud 云平台创建负载均衡器。您在创建和使用应用路由、服务时即可为其自动生成公网 IP。

```
service.beta.kubernetes.io/qingcloud-load-balancer-eip-source : auto
```

2、等待 2 分钟左右负载均衡器将在云平台自动创建完毕并自动绑定公网 IP 至网关。

The screenshot shows the KubeSphere project management interface. At the top, there's a header bar with the project name "demo-namespace". Below the header, there's a sidebar on the left containing various project management options like Overview, Applications, Workload, Storage, Network & Services, Monitoring & Alerts, Configuration Center, and Project Settings. The main content area displays the "demo-namespace" project details, including the administrator "admin", the namespace "demo", the creation time "2019-06-05 22:25:00", and the status "创建时间". It also shows external access settings for a LoadBalancer service, with an IP address of 139.198.1.1 and an external IP of 139.198.1.1. A note indicates that application governance is not enabled. There are also sections for annotations and labels.

3、在部署应用或创建应用路由、服务后，即可自动使用该公网 IP 或域名来暴露服务，方便用户快速访问。

The screenshot shows the KubeSphere application component interface for the "bookinfo" application. On the left, there's a sidebar with "bookinfo" application details, including its name and version. The main content area has tabs for Application Components, Traffic Management, Gray Release, Tracing, and Events. Under the Application Components tab, there's a section for Application Routes. It shows a route named "bookinfo-ingress" with the host "productpage.demo-namespace.139.198.1.1.nip.io". Below it, a URL is provided: "http://productpage.demo-namespace.139.198.1.1.nip.io/". A "Click to Visit" button is available to directly access the application.

# 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 v2.0.0 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；
- 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；
- 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。

## 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

我们提供了几篇具有代表性的示例和文档，帮助您快速上手 CI/CD。

- [Source to Image](#)
- [基于Spring Boot项目构建流水线](#)
- [CI/CD 流水线 \(离线版\)](#)
- [Jenkinsfile out of SCM](#)

## 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 is a sidebar with navigation items: '企业空间 sample' (Enterprise Space sample), '概览' (Overview), '项目管理' (Project Management), 'DevOps工程' (DevOps Engineering) which is currently 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 sub-note: 'DevOps 工程对应的是 Jenkins 的文件夹，用户可以按照自己的分类方法通过 DevOps 工程来组织 Pipeline。'. Below this is a search bar with placeholder text '输入查询条件进行过滤' (Enter query conditions for filtering) and a '创建' (Create) button. A table lists existing DevOps projects, with one entry visible: '名称' (Name): 'test-cicd', '项目管理员' (Project Manager): 'zpf', and '创建时间' (Creation Time): '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
- 创建人:** 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 items:

- 企业空间:** sample (with a green icon)
- 成员:** 1 (with a user icon)
- 成员角色:** 4 (with a role 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 DevOps 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', '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) are listed. 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。

This screenshot shows the 'Invite Member to Project' dialog box. It has a search bar at the top labeled '输入邮箱邀请项目成员'. Below it is a list of users:

- admin (checked)
- carlosfeng (unchecked)
- zpf (unchecked)

Each user entry has a small '+' button to its right. A red arrow points to the '+' button next to 'carlosfeng'. At the bottom right of the dialog box is a large '邀请成员' (Invite Member) button.

## 编辑或删除工程

在 DevOps 工程列表页，点击 “...” 按钮，可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。

# 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。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

## 前提条件

- 本示例以 GitLab 和 Harbor 为例，参考前请确保正确安装了 [内置 Harbor](#) 和 [内置 GitLab](#)。
- 已创建了企业空间和 DevOps 工程并创建了项目普通用户 project-regular，若还未创建请参考 [多租户管理快速入门](#)。

## 创建凭证

以项目 project-regular 登录 KubeSphere，在 devops-demo 的 DevOps 工程点击 [凭证](#)，进入凭证管理界面，以下说明几个在 DevOps 工程中常用的凭证。



## 创建 DockerHub 凭证

1、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，此示例中可命名为 **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

**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。

2、完成后点击 **确定**。

凭证

凭证是包含了一些敏感数据的对象，如用户名密码，SSH 密钥和 Token 等，用于在 Pipeline 运行时，为拉取代码、push/pull 镜像、SSH 执行脚本等过程提供认证

输入查询条件进行过滤

名称      类型      描述信息      创建时间

名称	类型	描述信息	创建时间
dockerhub-id	账户凭证		2019-04-27 13:35:48

创建

## 创建 GitHub 凭证

同上，创建一个用于 GitHub 的凭证，凭证 ID 可命名为 `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`。

## 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 可命名为 `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。

## 创建 Harbor 凭证

同上，创建一个用于 内置 Harbor 登录的凭证，此处命名为 `harbor-id`，类型选择 `账户凭证`。用户名和密码填写 `admin` 和 `Harbor12345`，完成后点击 `确定`。

## 创建 GitLab 凭证

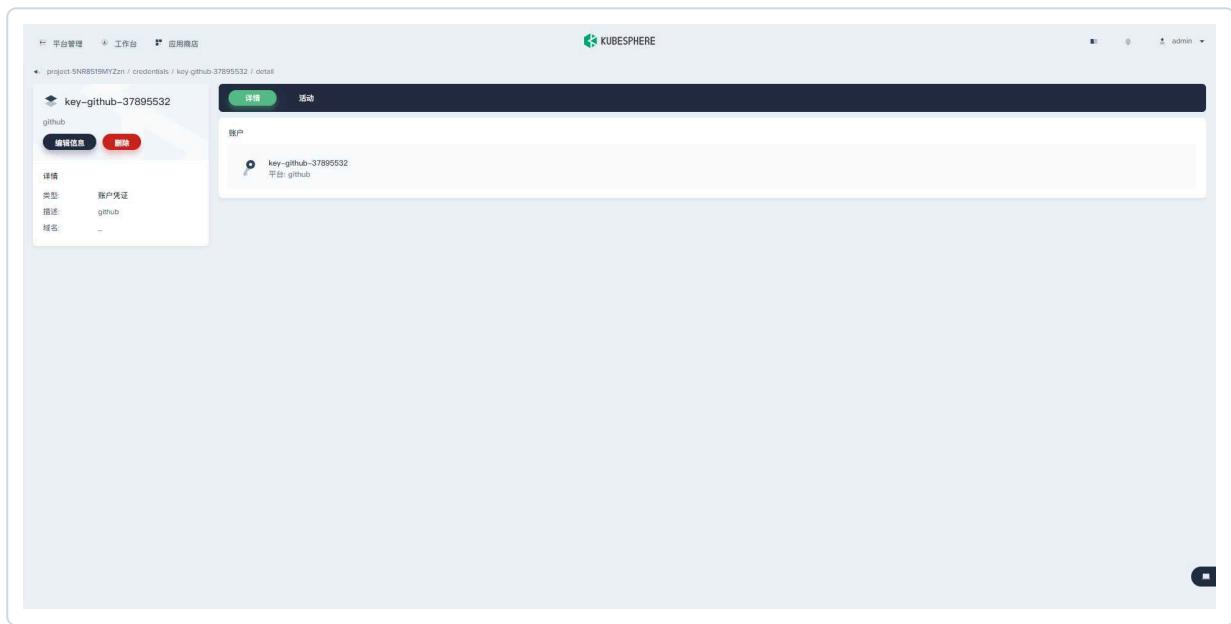
同上，创建一个用于 GitLab 的凭证，凭证 ID 命名为 `gitlab-id`，类型选择 `账户凭证`，GitLab 默认的管理员账号密码与 KubeSphere 一致，输入 GitLab 用户名 (`admin`) 和密码 (`passw0rd`)，备注描述信

息，完成后点击 确定。

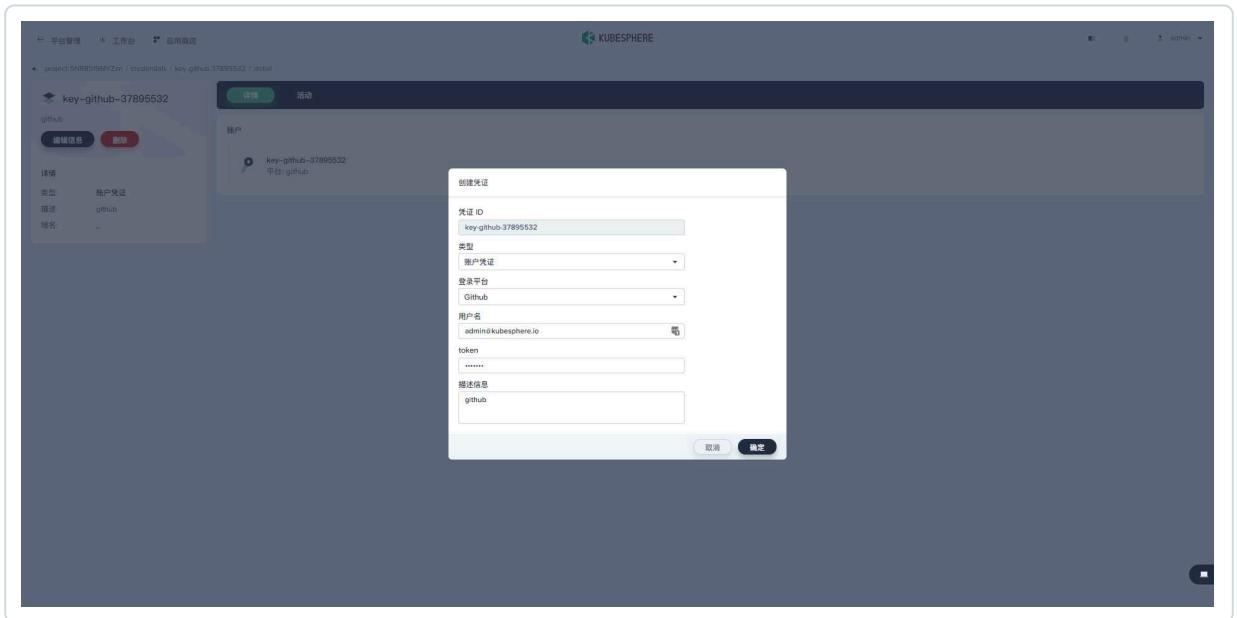
## 管理凭证

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。

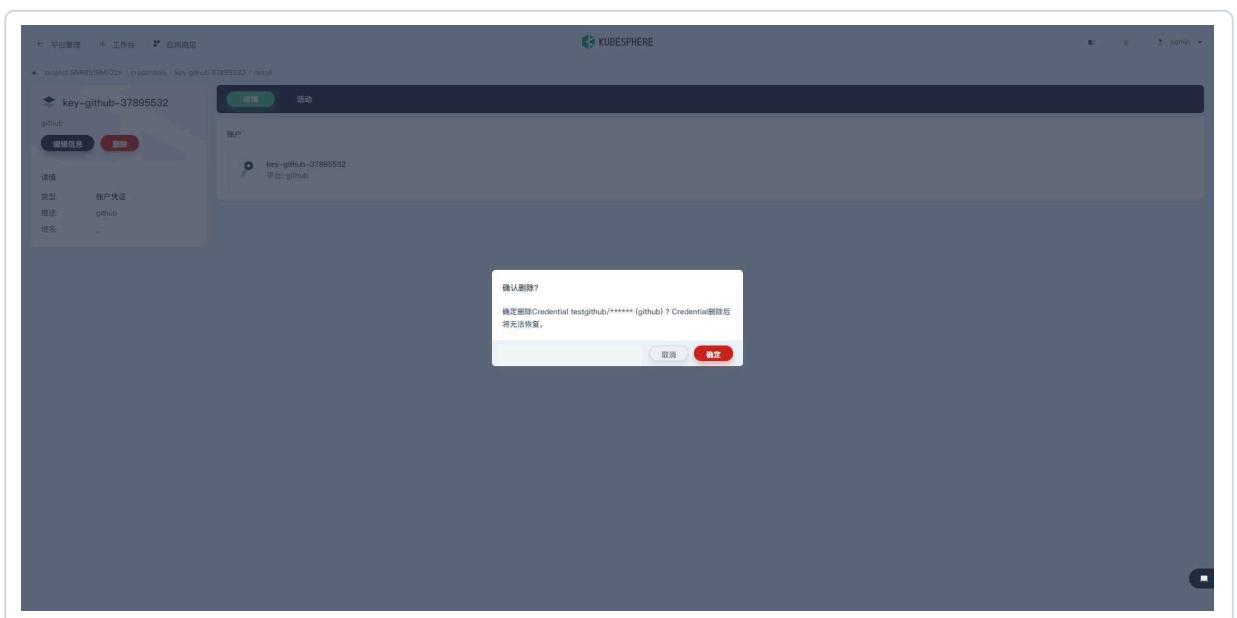
1. 在左侧的工程管理菜单下，点击 **凭证**，进入凭证管理界面，展示当前工程下的所有可用凭证。
2. 点击任意某一凭证，进入其详情页面。在此页面中可进行凭证的编辑，删除以及查看凭证的使用情况等操作。



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。



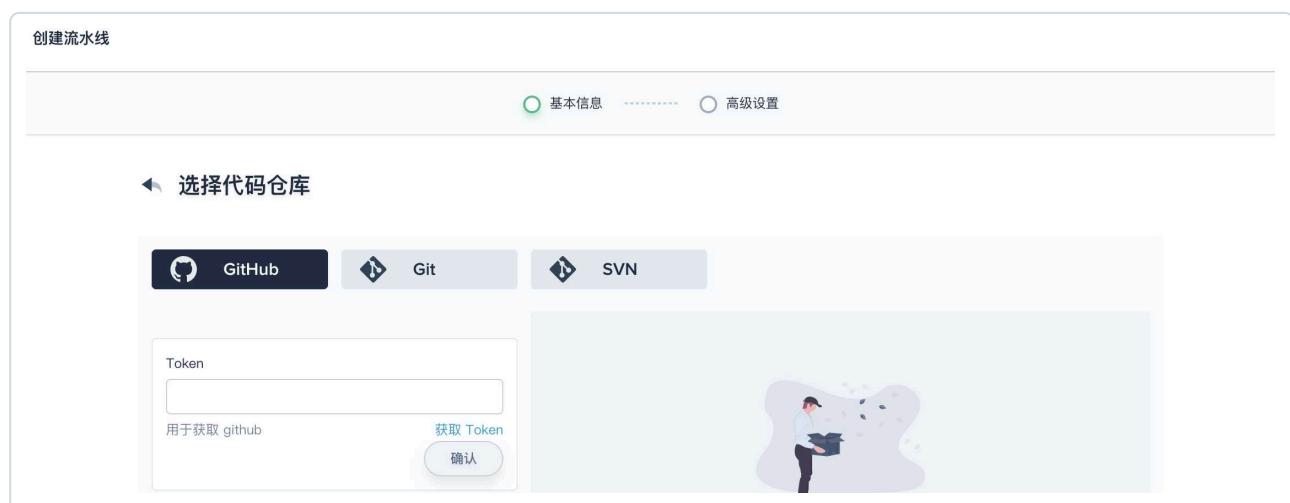
#### 4. 点击删除按钮，可删除此凭证。



## 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：

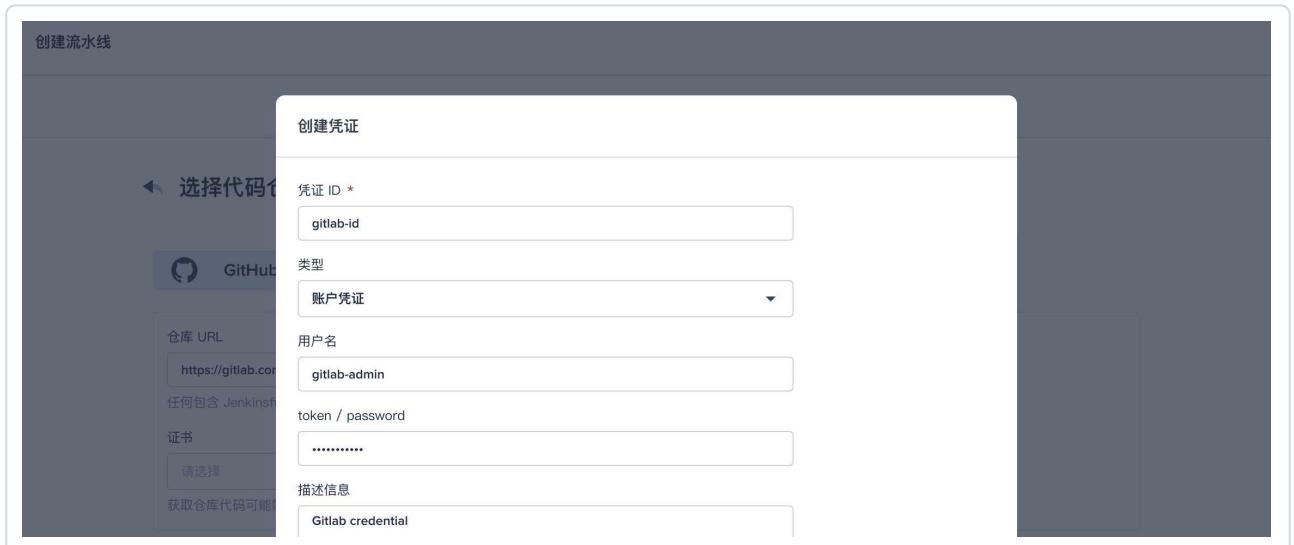
### 添加 GitHub

添加 GitHub 仓库已在 基于 Spring Boot 项目构建流水线示例文档中以添加示例的方式给出，详见 [基于 Spring Boot 项目构建流水线 – 添加 GitHub](#)。

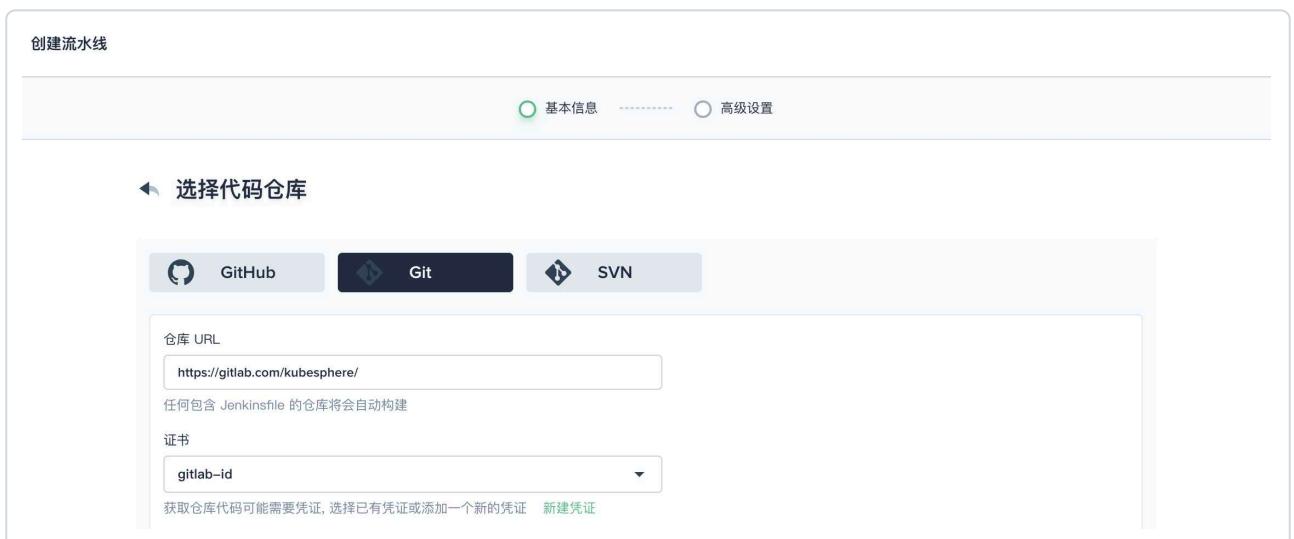
### 添加 Git

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

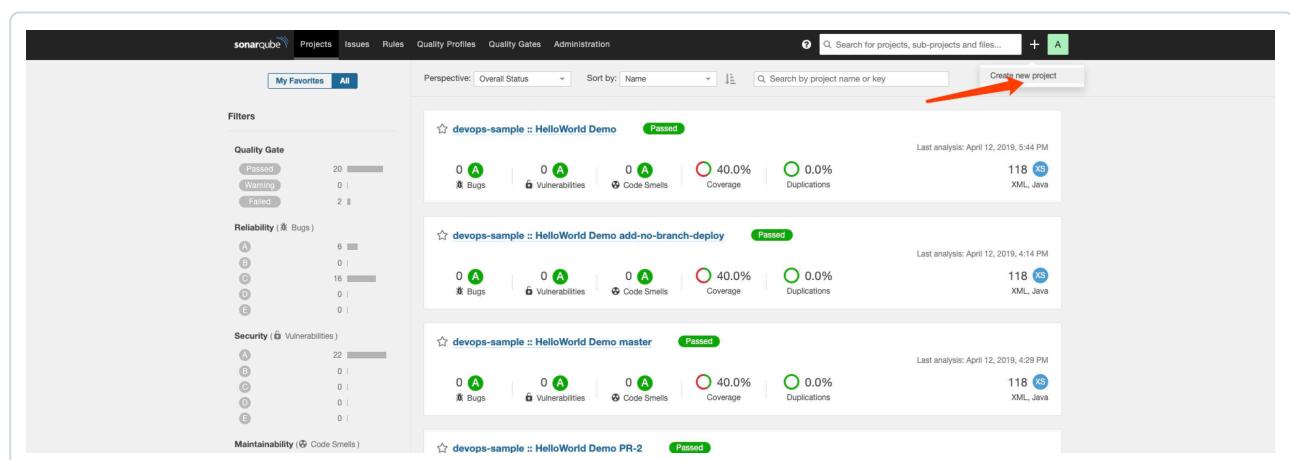
# 创建 SonarQube Token

## 访问 SonarQube

参考 [访问内置 SonarQube](#)。

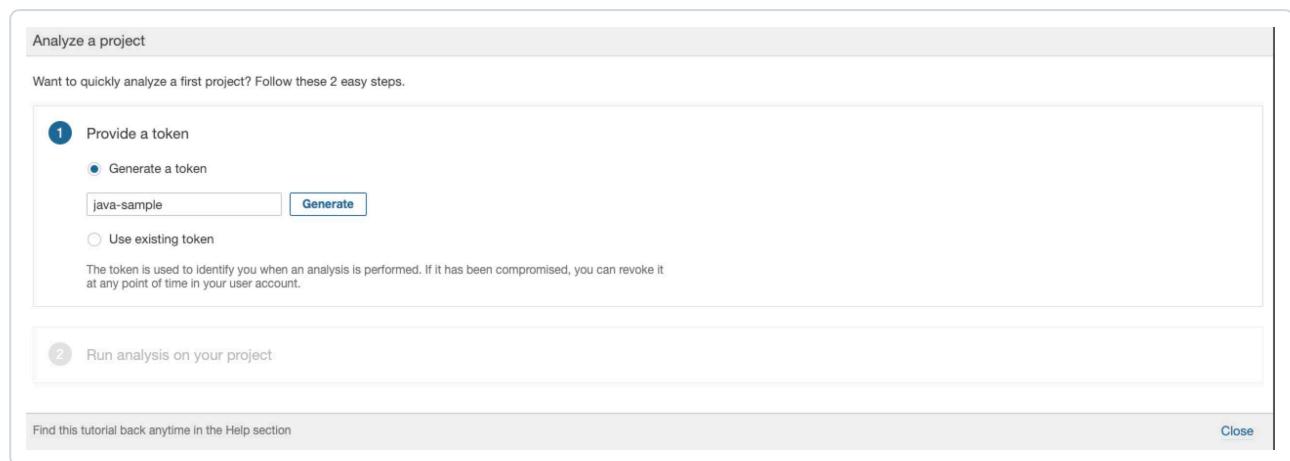
## 创建 SonarQube Token

1、使用默认账号 `admin/admin` 登入 sonar，然后点击右上角加号下的 `Create new project`。



The screenshot shows the SonarQube dashboard. At the top right, there is a green button labeled "Create new project". A red arrow points to this button, indicating it should be clicked.

2、然后输入 `name`，然后点击 `Generate`。



The screenshot shows the "Analyze a project" dialog. It has two main sections:

- 1 Provide a token**
  - Generate a token
  - 
  - 
  - Use existing token
- 2 Run analysis on your project**

At the bottom of the dialog, there is a note: "Find this tutorial back anytime in the Help section" and a "Close" button.

3、即可获取 token，然后点击 `Continue`。

Analyze a project

Want to quickly analyze a first project? Follow these 2 easy steps.

1 Provide a token

java-sample: **14f73b99a898e72ecdac632f6d148403b7755489**

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

**Continue**

2 Run analysis on your project

Find this tutorial back anytime in the Help section **Close**

4、然后选择 Language **Java**，选择 build technology 为 **Maven**，复制 token。点击 **Finish this tutorial** 即可。

Analyze a project

Want to quickly analyze a first project? Follow these 2 easy steps.

1 Provide a token ads:c1c0c5133288718af415035e9a7ce53a5802c9e1

2 Run analysis on your project

What is your project's main language? **Java** C# or VB.NET Other (JS, TS, Go, Python, PHP, ...)

You are developing primarily in Java: what is your build technology? **Maven** Gradle

Execute the Scanner for Maven from your computer

Running a SonarQube analysis with Maven is straightforward. You just need to run the following command in your project's folder.

```
mvn sonar:sonar \
-Dsonar.host.url=http://139.198.121.109:31147 \
-Dsonar.login=c1c0c5133288718af415035e9a7ce53a5802c9e1
```

**Copy**

Please visit the [official documentation of the Scanner for Maven](#) for more details.

Once the analysis is completed, you will be able to browse your project at the URL displayed at the end of the logs.

Find this tutorial back anytime in the Help section **Finish this tutorial**

## 设置自动触发扫描

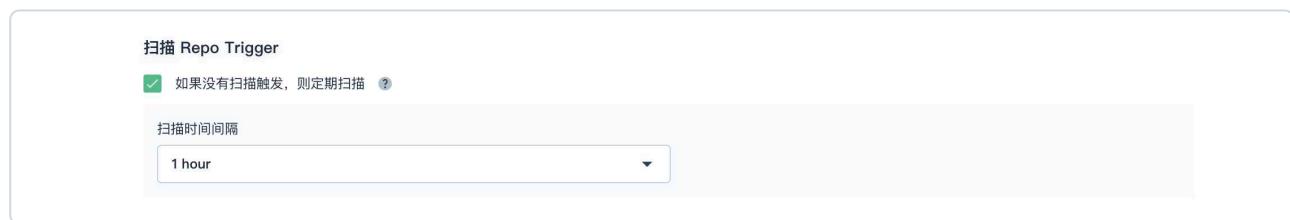
在含有 SCM (Source Code Management) 的 Pipeline 中，用户如果需要为流水线设置自动发现远程分支的变化，以生成新的 Pipeline 并使其自动地重新运行，可参考以下方式设置自动触发扫描。在 KubeSphere 中根据 SCM 类型的不同，也相应地提供了不同的方式进行触发。

### 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。

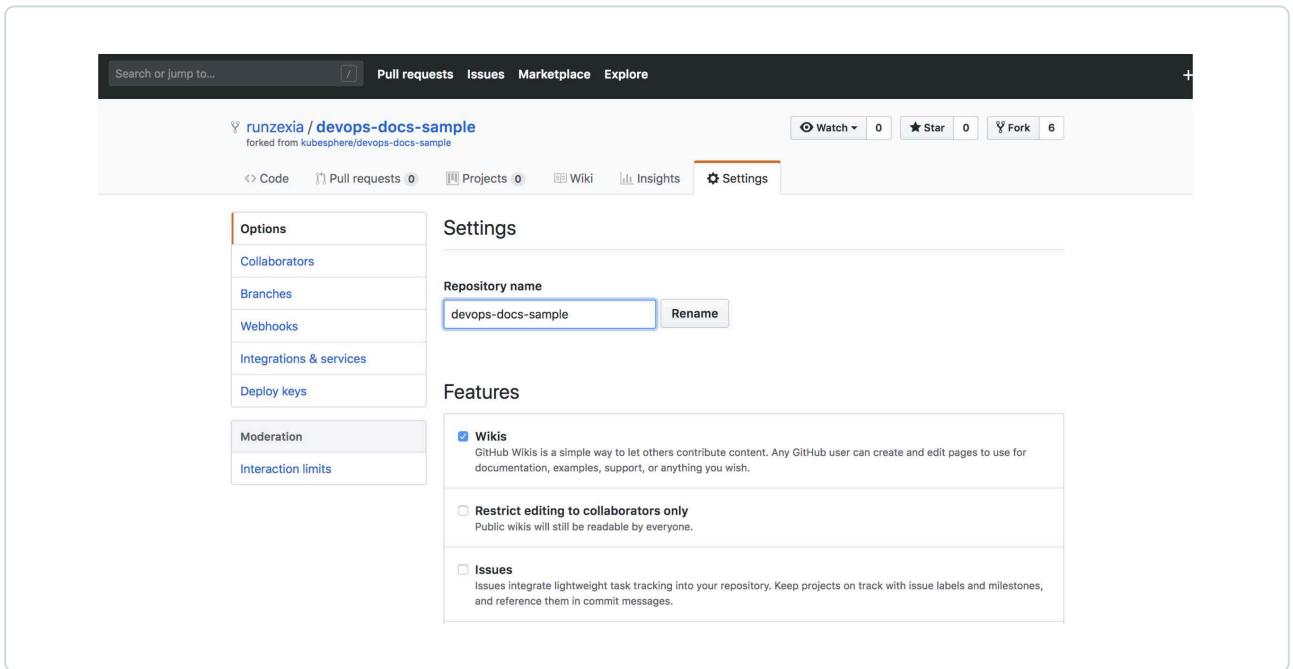
#### 第一步：设置定期扫描

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）。

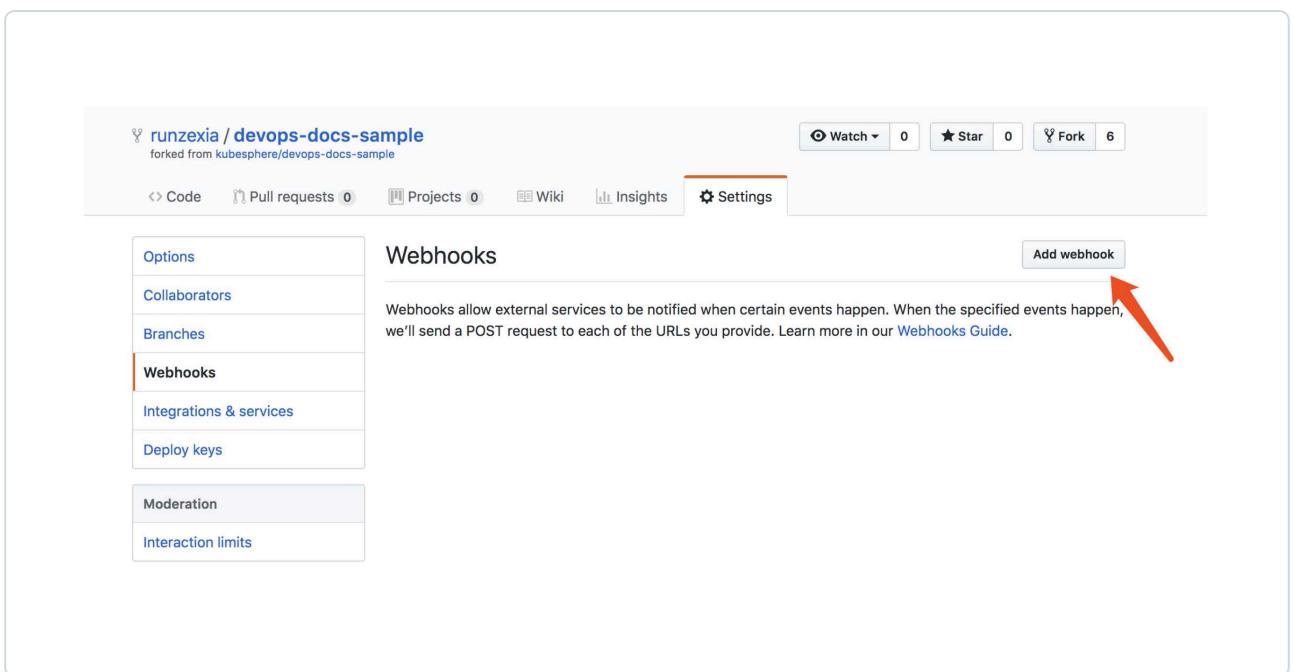


#### 第二步：设置 GitHub Webhook

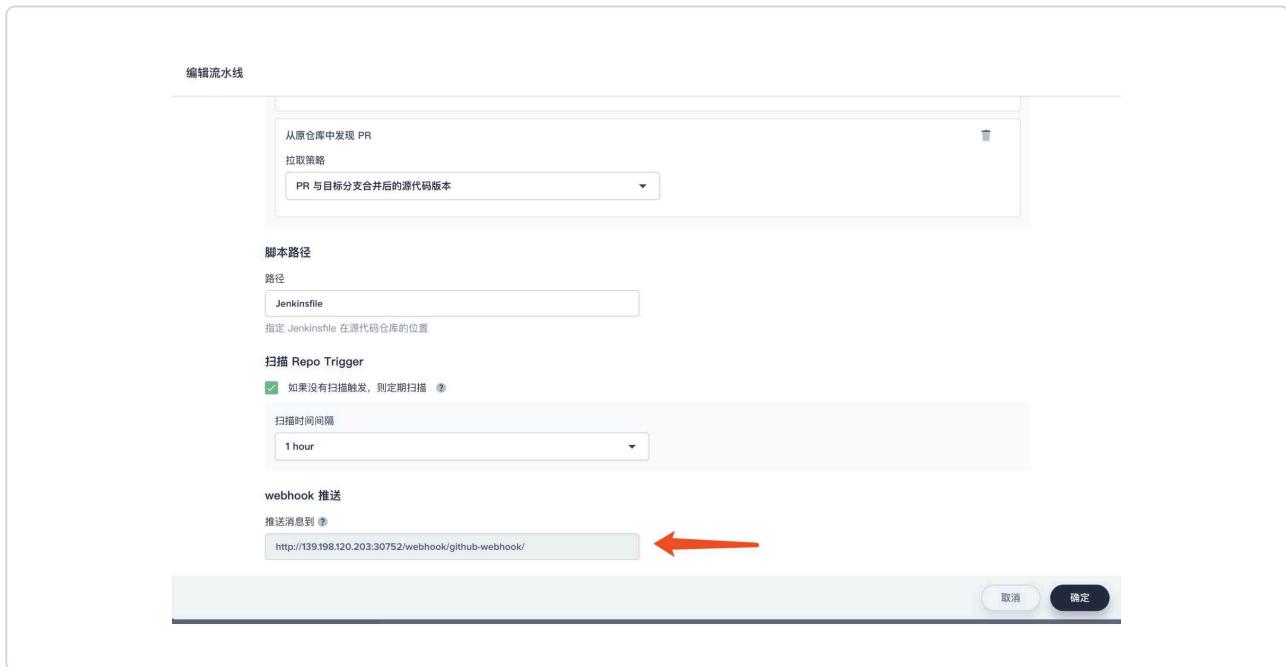
1、Webhook 需要用户自行到 GitHub 的 **Settings → Webhooks** 自行进行配置，并且需要 GitHub 能够访问到您安装的 KubeSphere 控制台地址。进入 GitHub，访问需要配置 Webhook 的仓库，比如当前的示例仓库 `devops-docs-sample`，选择 **Settings → Webhooks** 进行设置。



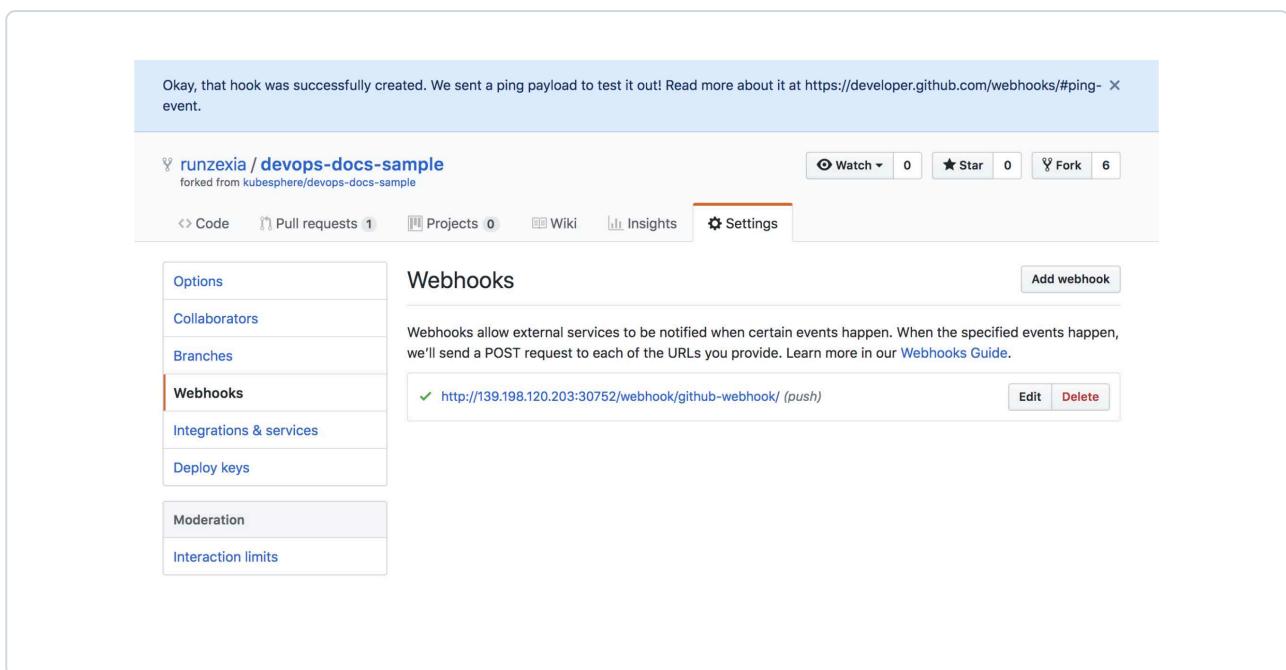
2、点击左侧 Webhooks，进入 Webhook 配置页面。点击 Add webhook 即可添加新的 Webhook。



注意，Payload 地址填写为关联的流水线 Webhook 推送 的默认地址，



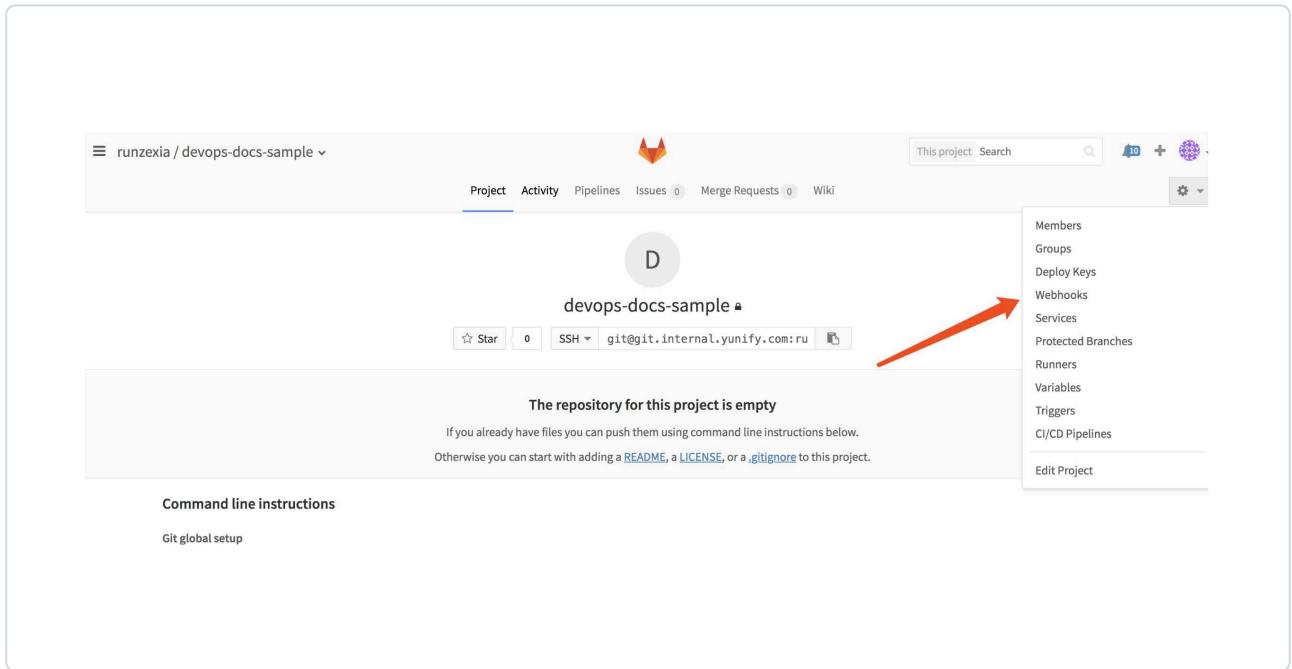
3、点击 Add Webhook 完成 Webhook 的添加，可以看到 Webhook 已经创建成功。



## 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

A screenshot of the 'Webhooks' configuration page in KubeSphere. The 'URL' field contains the value 'https://console-ae.kubesphere.io/webhook/git/notifyCommit/?url=https://gitlab.com/runzexia/devops-docs-sample'. Below the URL field is a 'Secret Token' input field. Under the 'Trigger' section, several checkboxes are listed: 'Push events' (checked), 'Tag push events' (checked), 'Comments' (unchecked), 'Issues events' (unchecked), 'Confidential Issues events' (unchecked), 'Merge Request events' (unchecked), 'Build events' (unchecked), and 'Pipeline events' (unchecked). A note at the bottom states: 'This URL will be triggered when the pipeline status changes.'

## 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

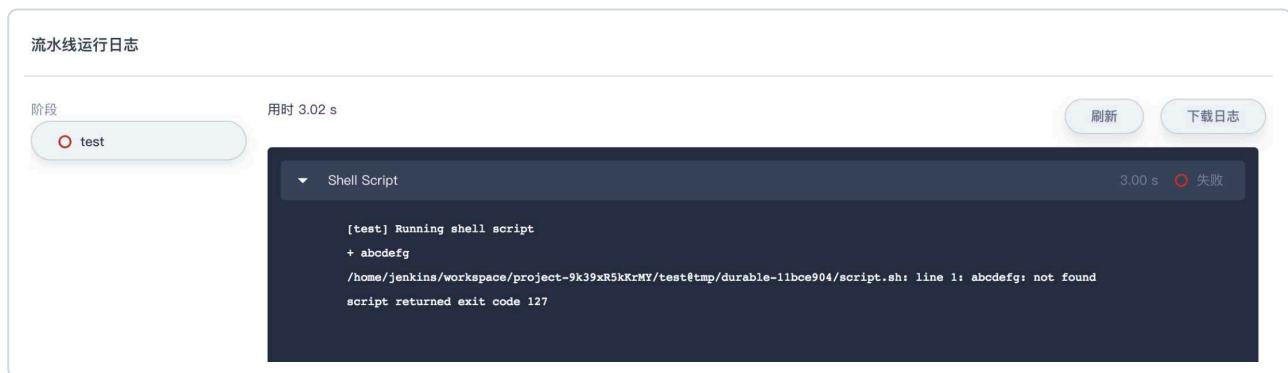
名称	类型 / 版本
Jenkins Agent Label	go
Container Name	go
操作系统	centos-7
Go	1.11
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
Docker	18.06.0
Helm	2.11.0
Kubectl	stable release
内置工具	unzip、which、make、wget、zip、bzip2、git

# 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？

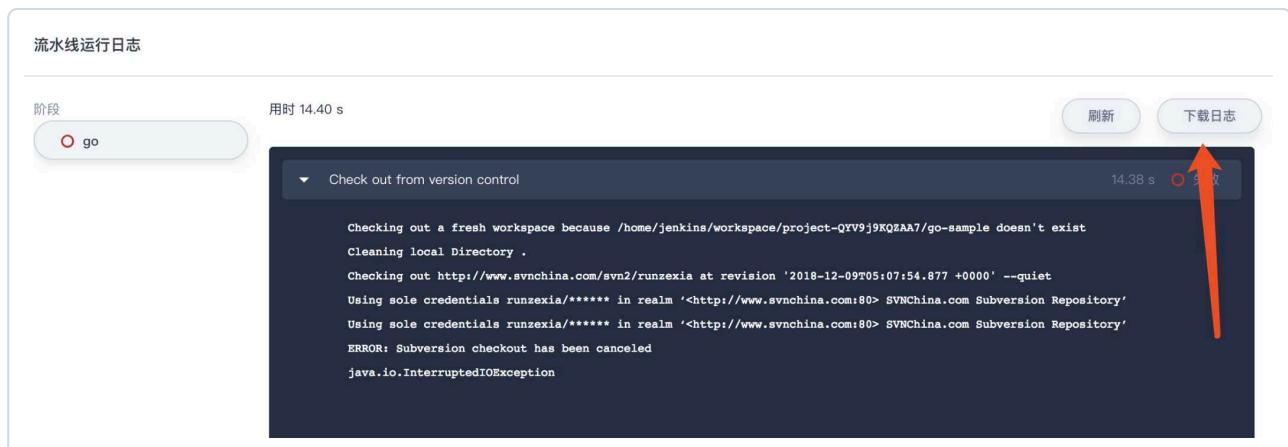
## 如何查看日志

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。

kubesphere / devops-docs-sample

Code Issues Pull requests Projects Wiki Insights

No description, website, or topics provided.

42 commits 1 branch 0 releases 2 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

rayzhou2017 Update docs-sample-svc.yaml

__mocks_	init	23 days ago
content	rm static, update yarn lock	23 days ago
deploy	Update docs-sample-svc.yaml	23 days ago
plugins	init	23 days ago
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。

## Jenkins 流水线质量门

Jenkins 流水线质量门需要在流水线中先执行代码质量分析，在执行代码质量分析时应该使用加载 SonarQube 配置中执行质量分析。

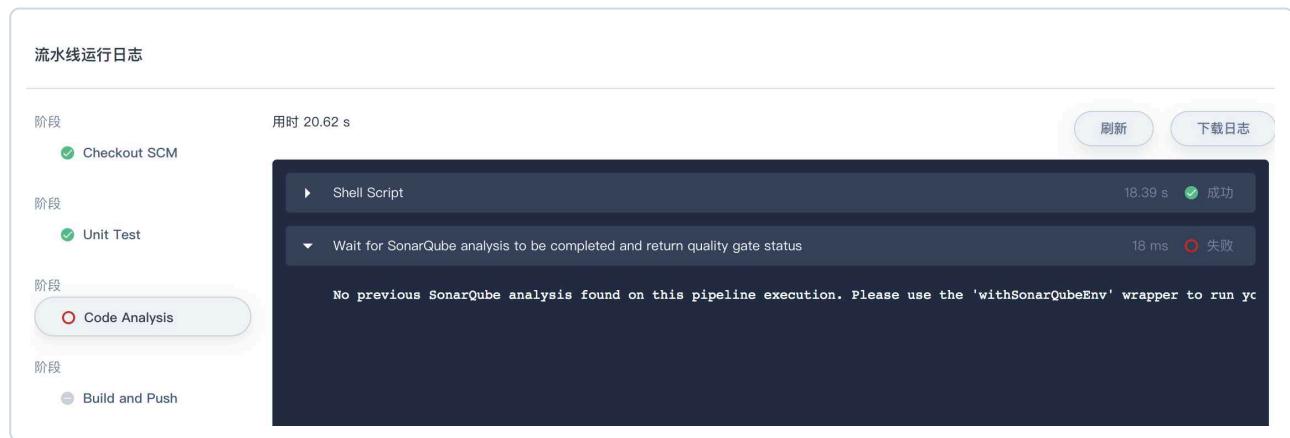
对应的 `Jenkinsfile` 为：

```
withSonarQubeEnv('sonar') {  
    sh "${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=ks-devops -Dsonar.sources=. -Dsonar.login=$sonarToken"  
}
```

当执行完代码分析之后，用户就可以使用代码质量检查步骤来检查代码质量是否符合标准。对应的 `Jenkinsfile` 为：

```
waitForQualityGate abortPipeline: true
```

部分用户在使用 SonarQube 质量门时遇到下图所示问题：



这是因为在 `Jenkinsfile` 当中 `waitForQualityGate` 的代码位置有误，`waitForQualityGate` 应该在 `withSonarQubeEnv` 代码块之外，正确的 `Jenkinsfile` 为：

```
withSonarQubeEnv('sonar') {  
    sh "${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=ks-devops -Dsonar.sources=. -Dsonar.login=$sonarToken"  
}  
waitForQualityGate abortPipeline: true
```

## Jenkins Kubernetes Deploy 所支持的资源类型与资源版本

受限于目前的 Kubernetes Deploy 的插件实现，目前仅能支持特定 API 版本的特定资源。

支持的资源列表如下表格所示：

资源类型	API 版本
ConfigMap	v1
Daemon Set	extensions/v1beta1
Deployment	extensions/v1beta1
Ingress	extensions/v1beta1
Job	batch/v1
Namespace	v1
Pod	v1
Secret	v1
Service	v1

## 为各种不同语言的项目执行代码分析

在流水线示例当中，我们使用 `mvn sonar:sonar` 为 Maven 管理的 Java 项目执行代码分析。

现在我们介绍一下如何为其他语言的项目执行代码分析，在为其他语言的项目做分析时候，我们大都会使用 [SonarQube Scanner](#) 作为执行分析命令的工具。

### 流水线中一键安装并使用 SonarQube Scanner

下面我就来介绍一下如何在流水线中获取并使用 [SonarQube Scanner](#)：

在 KubeSphere 所提供的 Jenkins 当中可以使用 Jenkins `tool` 命令在不同的 Pipeline agent 上安装 SonarQube Scanner，如下面的 Jenkinsfile 所示：

```
stage('sonarqube analysis'){
    steps{
        script {
            scannerHome = tool 'sonar'; // 安装sonar工具，并且获取sonar工具路径
        }
        withCredentials([string(credentialsId: 'sonar-token', variable: 'SONAR_TOKEN')]) {
            withSonarQubeEnv('sonar') {
                sh "${scannerHome}/bin/sonar-scanner -Dsonar.branch=$BRANCH_NAME -Dsonar.projectKey=... -Dsonar.host.url=https://sonarcloud.io"
            }
        }
        timeout(time: 1, unit: 'HOURS') {
            waitForQualityGate abortPipeline: true
        }
    }
}
```

## 各种语言的分析方式

对于各种不同的语言，SonarQube 执行分析的配置不太相同，具体可以参考 [SonarQube 官方文档](#)。

# API 文档

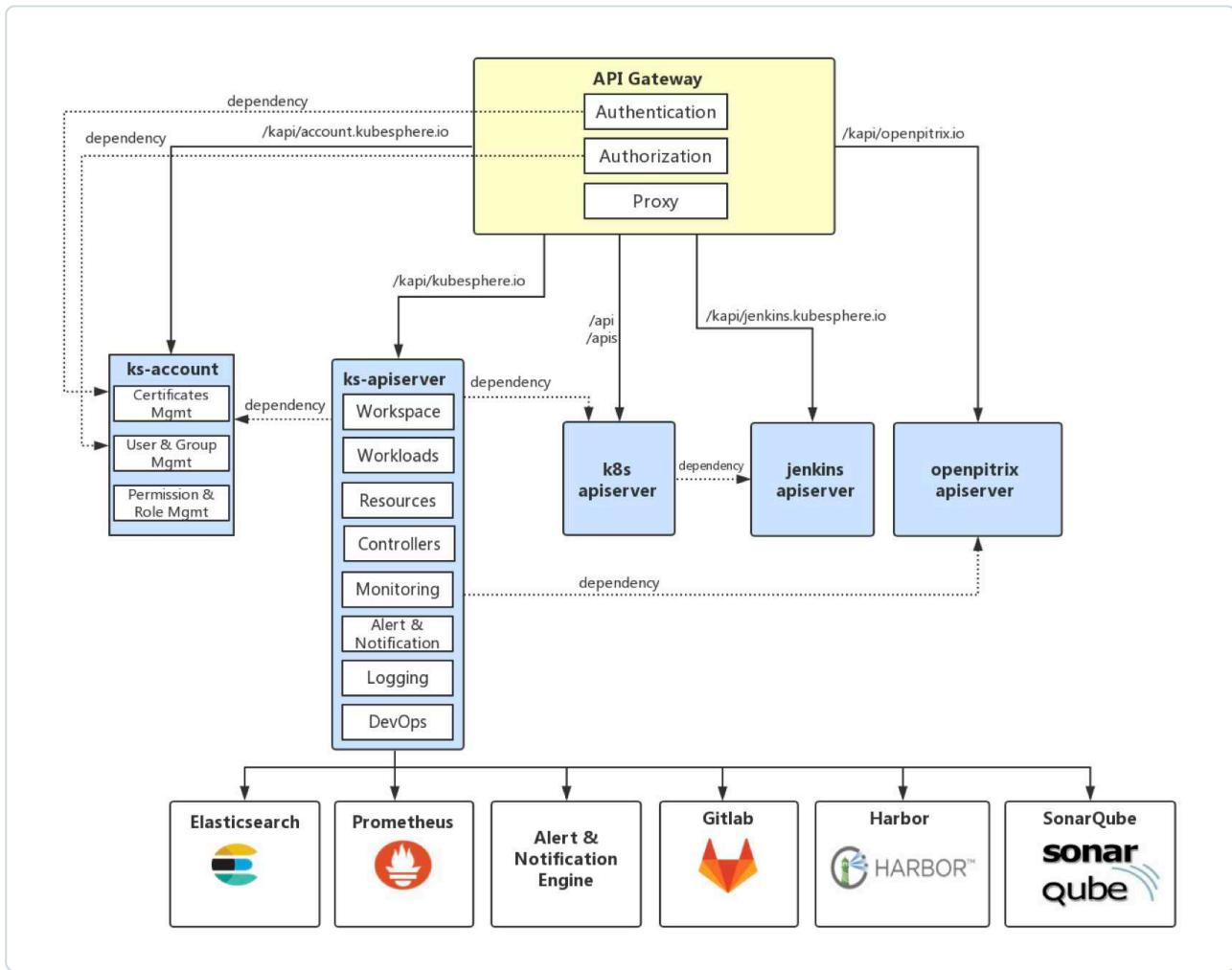
KubeSphere 提供了丰富的 RESTful 规范的 API 供您使用，在开始调用之前需要登录 KubeSphere 后台或访问 Swagger UI 登录后获得 JWT Bearer token 后，才可以使用 KubeSphere API。

- [KubeSphere API 文档](#)
- [Notification API 文档](#)

## API 架构

下图是 KubeSphere API 的架构，所有请求都将通过 API Gateway 进行认证授权代理后发送到各个服务组件。

其中 `/kapi/xxx.kubesphere.io` 是 KubeSphere 拓展聚合的 API, `/api` 和 `/apis` 开头的都属于 Kubernetes 原生的 API, KubeSphere 把用户对原生 Kubernetes 资源的请求通过 API Gateway 转发到 Kubernetes API Server 对原生资源进行操作和管理。



# 如何调用 API

ks-apigateway 是 KubeSphere 的 API 网关，当您部署 KubeSphere 之后，可参考如下的 API 调用流程。

## 第一步：暴露 ks-apigateway 服务

可通过将 ks-apigateway 服务的端口设置为 NodePort 形式暴露 ks-apigateway 服务，可以使用 UI 或命令行这两种方式：

### 使用 UI

### 使用 UI

- 1、登录 KubeSphere 控制台，进入企业空间 `system-workspace` -> 项目 `kubesphere-system`，在这个项目下服务列表，点击进入 `ks-apigateway` 这个服务的详情页。
- 2、点击「更多操作」 -> 「编辑外网访问」，设置访问方式为 `NodePort`，点击确定。
- 3、在服务详情页即可看到生成的 NodePort 为 31078。

名称	IP地址	端口	应用	创建时间
ks-apigateway	Virtual IP: 10.233.17.73	端口: 80:2018/TCP 节点端口: 80:31078/TCP	-	2019-06-28 20:01:04

### 使用命令行

### 使用命令行

- 1、使用 admin 账户登录 KubeSphere，在右下角的「工具箱」打开 Web Kubectl，执行下述命令。

```
$ kubectl -n kubesphere-system patch svc ks-apigateway -p '{"spec":{"type":"NodePort"}}'
service/ks-apigateway patched
```

2、通过以下命令查看生成的端口号，如下查看的端口号返回是 31078。

```
$ kubectl -n kubesphere-system get svc ks-apigateway -o jsonpath='{.spec.ports[0].nodePort}'
31078
```

## 第二步：获取 Token

KubeSphere 所有 API 都需要通过 JWT Bearer token 进行认证，在开始 API 调用之前，需要先通过 [/kapis/iam.kubesphere.io/v1alpha2/login](#) 接口获取 `access_token`，并在之后的请求中带上认证请求头 `Authorization: Bearer <access_token>`。

在 KubeSphere 右下角的「工具箱」打开 Web Kubectl，执行下述命令，其中 `192.168.0.20` 是本示例的节点 IP，`31078` 是上一步暴露的 `ks-apigateway` 服务。

```
$ curl -X POST "http://192.168.0.20:31078/kapis/iam.kubesphere.io/v1alpha2/login" -H "acc
{
  "access_token": "eyJhbGxxxxxxxxS44"
}
```

## 第三步：调用 API

通过上述步骤拿到 Access Token 后，即可在用户自定义的请求函数中调用 KubeSphere API，可进一步参考 [API 文档](#)。

## 如何访问 Swagger UI

KubeSphere 的 API 可以在 [Swagger UI](#) 中预览，在浏览器中通过 URL <http://IP:NodePort/swagger-ui> 访问 Swagger UI，比如 <http://192.168.0.20:31078/swagger-ui/>。

swagger /swagger-ui/api.json Explore

## KubeSphere 2.0.0

/swagger-ui/api.json

KubeSphere OpenAPI

[kubesphere – Website](#)

[Send email to kubesphere](#)

[Apache](#)

Authorize

### DevOps

GET /apis/devops.kubesphere.io/v1alpha2/crumbIssuer Get crumb issuer. A CrumbIssuer represents an algorithm to generate a nonce value, known as a crumb, to counter cross site request forgery exploits. Crumbs are typically hashes incorporating information that uniquely identifies an agent that sends a request, along with a guarded secret so that the crumb value cannot be forged by a third party.

GET /apis/devops.kubesphere.io/v1alpha2/devops/{devops} Get the specified DevOps Project

PATCH /apis/devops.kubesphere.io/v1alpha2/devops/{devops} Update the specified DevOps Project

## API 常用术语对照

### DevOps

英文	中文
DevOps	DevOps 工程
Workspace	企业空间
Pipeline	流水线
Credential	凭证
Artifact	制品
Node	流水线执行过程中的阶段
Step	阶段中的步骤
Branch	分支
SCM	源代码管理工具，例如github、gitlab等
sonar	代码质量分析工具 sonarqube

### Monitoring

英文	中文
Metric	指标
Usage	用量
Utilisation	利用率
Throughput	吞吐量
Capacity	容量
Proposal	Etcd 提案

## Logging

英文	中文
Fuzzy Matching	模糊匹配

## Router

英文	中文
Router	网关
Routes	应用路由

## Service Mesh

英文	中文
ServiceMesh	服务网格
Tracing	追踪(分布式追踪)
Canary Release	金丝雀发布
Mirror	流量镜像
BlueGreen Release	蓝绿发布

## Notification

英文	中文
addresslist	通知地址列表

## 监控指标说明

KubeSphere 资源监控共分为八个层级：Cluster（集群），Node（节点），Workspace（企业空间），Namespace（项目），Workload（工作负载），Pod（容器组），Container（容器），Component（KubeSphere 核心组件）。

### Cluster

指标名	说明	单位
cluster_cpu_utilisation	集群 CPU 使用率	
cluster_cpu_usage	集群 CPU 用量	Core
cluster_cpu_total	集群 CPU 总量	Core
cluster_load1	集群 1 分钟 CPU 平均负载 <sup>1</sup>	
cluster_load5	集群 5 分钟 CPU 平均负载	
cluster_load15	集群 15 分钟 CPU 平均负载	
cluster_memory_utilisation	集群内存使用率	
cluster_memory_available	集群可用内存	Byte
cluster_memory_total	集群内存总量	Byte
cluster_memory_usage_wo_cache	集群内存使用量 <sup>2</sup>	Byte
cluster_net_utilisation	集群网络数据传输速率	Byte/s
cluster_net_bytes_transmitted	集群网络数据发送速率	Byte/s
cluster_net_bytes_received	集群网络数据接受速率	Byte/s
cluster_disk_read_iops	集群磁盘每秒读次数	次/s
cluster_disk_write_iops	集群磁盘每秒写次数	次/s
cluster_disk_read_throughput	集群磁盘每秒读取数据量	Byte/s
cluster_disk_write_throughput	集群磁盘每秒写入数据量	Byte/s
cluster_disk_size_usage	集群磁盘使用量	Byte

指标名	说明	单位
cluster_disk_size_utilisation	集群磁盘使用率	
cluster_disk_size_capacity	集群磁盘总容量	Byte
cluster_disk_size_available	集群磁盘可用大小	Byte
cluster_disk_inode_total	集群 inode 总数	
cluster_disk_inode_usage	集群 inode 已使用数	
cluster_disk_inode_utilisation	集群 inode 使用率	
cluster_node_online	集群节点在线数	
cluster_node_offline	集群节点下线数	
cluster_node_offline_ratio	集群节点下线比例	
cluster_node_total	集群节点总数	
cluster_pod_count	集群中调度完成 <sup>3</sup> Pod 数量	
cluster_pod_quota	集群各节点 Pod 最大容纳量 <sup>4</sup> 总和	
cluster_pod_utilisation	集群 Pod 最大容纳量使用率	
cluster_pod_running_count	集群中处于 Running 阶段 <sup>5</sup> 的 Pod 数量	
cluster_pod_succeeded_count	集群中处于 Succeeded 阶段的 Pod 数量	
cluster_pod_abnormal_count	集群中异常 Pod <sup>6</sup> 数量	
cluster_pod_abnormal_ratio	集群中异常 Pod 比例 <sup>7</sup>	
cluster_ingresses_extensions_count	集群 Ingress 数	
cluster_cronjob_count	集群 CronJob 数	
cluster_pvc_count	集群 PersistentVolumeClaim 数	
cluster_daemonset_count	集群 DaemonSet 数	
cluster_deployment_count	集群 Deployment 数	
cluster_endpoint_count	集群 Endpoint 数	
cluster_hpa_count	集群 Horizontal Pod Autoscaler 数	

指标名	说明	单位
cluster_job_count	集群 Job 数	
cluster_statefulset_count	集群 StatefulSet 数	
cluster_replicaset_count	集群 ReplicaSet 数	
cluster_service_count	集群 Service 数	
cluster_secret_count	集群 Secret 数	
cluster_namespace_count	集群 Namespace 数	

### 【说明】

**1** 指单位时间内，单位 CPU 运行队列中处于可运行或不可中断状态的平均进程数。如果数值大于 1，表示 CPU 不足以服务进程，有进程在等待。

**2** 不包含 buffer、cache。

**3** Pod 已经被调度到节点上，即 status.conditions.PodScheduled = true 。参考：[Pod Lifecycle](#)

**4** 节点 Pod 最大容纳量一般默认 110 个 Pod。参考：[kubelet Options](#)

**5** Running 阶段表示该 Pod 已经绑定到了一个节点上，Pod 中所有的容器都已被创建。至少有一个容器正在运行，或者正处于启动或重启状态。参考：[Pod Lifecycle](#)

**6** 异常 Pod：如果一个 Pod 的 status.conditions.ContainersReady 字段值为 false，说明该 Pod 不可用。我们在判定 Pod 是否异常时，还需要考虑到 Pod 可能正处于 ContainerCreating 状态或者 Succeeded 已完成阶段。综合以上情况，异常 Pod 总数的算法可表示为：Abnormal Pods = Total Pods – ContainersReady Pods – ContainerCreating Pods – Succeeded Pods 。

**7** 异常 Pod 比例：异常 Pod 数 / 非 Succeeded Pod 数。

## Node

指标名	说明	单位
node_cpu_utilisation	节点 CPU 使用率	
node_cpu_total	节点 CPU 总量	Core

指标名	说明	单位
node_cpu_usage	节点 CPU 用量	Core
node_load1	节点 1 分钟 CPU 平均负载	
node_load5	节点 5 分钟 CPU 平均负载	
node_load15	节点 15 分钟 CPU 平均负载	
node_memory_utilisation	节点内存使用率	
node_memory_usage_wo_cache	节点内存使用量 <sup>1</sup>	Byte
node_memory_available	节点可用内存	Byte
node_memory_total	节点内存总量	Byte
node_net_utilisation	节点网络数据传输速率	Byte/s
node_net_bytes_transmitted	节点网络数据发送速率	Byte/s
node_net_bytes_received	节点网络数据接受速率	Byte/s
node_disk_read_iops	节点磁盘每秒读次数	次/s
node_disk_write_iops	节点磁盘每秒写次数	次/s
node_disk_read_throughput	节点磁盘每秒读取数据量	Byte/s
node_disk_write_throughput	节点磁盘每秒写入数据量	Byte/s
node_disk_size_capacity	节点磁盘总容量	Byte
node_disk_size_available	节点磁盘可用大小	Byte
node_disk_size_usage	节点磁盘使用量	Byte
node_disk_size_utilisation	节点磁盘使用率	
node_disk_inode_total	节点 inode 总数	
node_disk_inode_usage	节点 inode 已使用数	
node_disk_inode_utilisation	节点 inode 使用率	
node_pod_count	节点调度完成 Pod 数量	
node_pod_quota	节点 Pod 最大容纳量	

指标名	说明	单位
node_pod_utilisation	节点 Pod 最大容纳量使用率	
node_pod_running_count	节点中处于 Running 阶段的 Pod 数量	
node_pod_succeeded_count	节点中处于 Succeeded 阶段的 Pod 数量	
node_pod_abnormal_count	节点异常 Pod 数量	
node_pod_abnormal_ratio	节点异常 Pod 比例	

### 【说明】

<sup>1</sup> 不包含 buffer、 cache。

## Workspace

指标名	说明	单位
workspace_cpu_usage	企业空间 CPU 用量	Core
workspace_memory_usage	企业空间内存使用量（包含缓存）	Byte
workspace_memory_usage_wo_cache	企业空间内存使用量	Byte
workspace_net_bytes_transmitted	企业空间网络数据发送速率	Byte/ s
workspace_net_bytes_received	企业空间网络数据接受速率	Byte/ s
workspace_pod_count	企业空间内非终止阶段 Pod 数量 <sup>1</sup>	
workspace_pod_running_count	企业空间内处于 Running 阶段的 Pod 数量	
workspace_pod_succeeded_count	企业空间内处于 Succeeded 阶段的 Pod 数量	
workspace_pod_abnormal_count	企业空间异常 Pod 数量	
workspace_pod_abnormal_ratio	企业空间异常 Pod 比例	
workspace_ingresses_extensions_count	企业空间 Ingress 数	

指标名	说明	单位
workspace_cronjob_count	企业空间 CronJob 数	
workspace_pvc_count	企业空间 PersistentVolumeClaim 数	
workspace_daemonset_count	企业空间 DaemonSet 数	
workspace_deployment_count	企业空间 Deployment 数	
workspace_endpoint_count	企业空间 Endpoint 数	
workspace_hpa_count	企业空间 Horizontal Pod Autoscaler 数	
workspace_job_count	企业空间 Job 数	
workspace_statefulset_count	企业空间 StatefulSet 数	
workspace_replicaset_count	企业空间 ReplicaSet 数	
workspace_service_count	企业空间 Service 数	
workspace_secret_count	企业空间 Secret 数	
workspace_all_project_count	企业空间下项目总数	

### 【说明】

<sup>1</sup> 非终止阶段的 Pod 指处于 Pending、Running、Unknown 阶段的 Pod，不包含被成功终止，或者因非 0 状态退出被系统终止的 Pod。参考：[Pod Lifecycle](#)

若 Workspace Monitoring API 设置了查询参数 type 为 statistics，则返回企业空间统计信息：

指标名	说明	单位
workspace_all_organization_count	集群企业空间总数	
workspace_all_account_count	集群账号总数	
workspace_all_project_count	集群项目总数	
workspace_all_devops_project_count <sup>1</sup>	集群 DevOps 工程总数	
workspace_namespace_count	企业空间项目总数	
workspace_devops_project_count	企业空间 DevOps 工程总数	
workspace_member_count	企业空间成员数	

指标名	说明	单位
workspace_role_count <sup>2</sup>	企业空间角色数	

### 【说明】

<sup>1</sup> 前四个指标适用于 /kapis/devops.kubesphere.io/v1alpha2/workspaces

<sup>2</sup> 后四个指标适用于 /kapis/devops.kubesphere.io/v1alpha2/workspaces/{workspace}

## Namespace

指标名	说明	单位
namespace_cpu_usage	项目 CPU 用量	Core
namespace_memory_usage	项目内存使用量（包含缓存）	Byte
namespace_memory_usage_wo_cache	项目内存使用量	Byte
namespace_net_bytes_transmitted	项目网络数据发送速率	Byte/ s
namespace_net_bytes_received	项目网络数据接受速率	Byte/ s
namespace_pod_count	项目内非终止阶段 Pod 数量	
namespace_pod_running_count	项目内处于 Running 阶段的 Pod 数量	
namespace_pod_succeeded_count	项目内处于 Succeeded 阶段的 Pod 数量	
namespace_pod_abnormal_count	项目异常 Pod 数量	
namespace_pod_abnormal_ratio	项目异常 Pod 比例	
namespace_cronjob_count	项目 CronJob 数	
namespace_pvc_count	项目 PersistentVolumeClaim 数	
namespace_daemonset_count	项目 DaemonSet 数	
namespace_deployment_count	项目 Deployment 数	

指标名	说明	单位
namespace_endpoint_count	项目 Endpoint 数	
namespace_hpa_count	项目 Horizontal Pod Autoscaler 数	
namespace_job_count	项目 Job 数	
namespace_statefulset_count	项目 StatefulSet 数	
namespace_replicaset_count	项目 ReplicaSet 数	
namespace_service_count	项目 Service 数	
namespace_secret_count	项目 Secret 数	
namespace_ingresses_extensions_count	项目 Ingress 数	

## Workload

指标名	说明	单位
workload_pod_cpu_usage	工作负载 <sup>1</sup> CPU 用量	Core
workload_pod_memory_usage	工作负载内存使用量 (包含缓存)	Byte
workload_pod_memory_usage_wo_cache	工作负载内存使用量	Byte
workload_pod_net_bytes_transmitted	工作负载网络数据发送速率	Byte/ s
workload_pod_net_bytes_received	工作负载网络数据接受速率	Byte/ s
workload_deployment_replica	Deployment 期望副本数	
workload_deployment_replica_available	Deployment 可用副本数 <sup>2</sup>	
workload_deployment_unavailable_replicas_ratio	Deployment 不可用副本数比例 <sup>3</sup>	
workload_statefulset_replica	StatefulSet 期望副本数	
workload_statefulset_replica_available	StatefulSet 可用副本数	
workload_statefulset_unavailable_replicas_ratio	StatefulSet 不可用副本数比例	

指标名	说明	单位
workload_daemonset_replica	DaemonSet 期望副本数	
workload_daemonset_replica_available	DaemonSet 可用副本数	
workload_daemonset_unavailable_replicas_ratio	DaemonSet 不可用副本数比例	

### 【说明】

- 1 目前支持的工作负载类型包括：Deployment, StatefulSet 和 DaemonSet。
- 2 可用副本指工作负载创建出的 Pod 处于可用状态，即该 Pod 的 status.conditions.ContainersReady 字段值为 true。
- 3 不可用副本数比例：不可用副本数 / 期望副本数。

## Pod

指标名	说明	单位
pod_cpu_usage	容器组 CPU 用量	Core
pod_memory_usage	容器组内存使用量（包含缓存）	Byte
pod_memory_usage_wo_cache	容器组内存使用量	Byte
pod_net_bytes_transmitted	容器组网络数据发送速率	Byte/s
pod_net_bytes_received	容器组网络数据接受速率	Byte/s

## Container

指标名	说明	单位
container_cpu_usage	容器 CPU 用量	Core
container_memory_usage	容器内存使用量（包含缓存）	Byte
container_memory_usage_wo_cache	容器内存使用量	Byte

# Component

指标名	说明	单位
etcd_server_list	etcd 集群节点列表 <sup>1</sup>	
etcd_server_total	etcd 集群节点总数	
etcd_server_up_total	etcd 集群在线节点数	
etcd_server_has_leader	etcd 集群各节点是否有 leader <sup>2</sup>	
etcd_server_leader_changes	etcd 集群各节点观察到 leader 变化数 (1h 内)	
etcd_server_proposals_failed_rate	etcd 集群各节点提案失败 <sup>3</sup> 频率平均数	次/s
etcd_server_proposals_applied_rate	etcd 集群各节点提案应用频率平均数	次/s
etcd_server_proposals_committed_rate	etcd 集群各节提案提交频率平均数	次/s
etcd_server_proposals_pending_count	etcd 集群各节点排队提案数平均值	
etcd_mvcc_db_size	etcd 集群各节点数据库大小平均值	Byte
etcd_network_client_grpc_received_bytes	etcd 集群向 gRPC 客户端发送数据速率	Byte/s
etcd_network_client_grpc_sent_bytes	etcd 集群接受 gRPC 客户端数据速率	Byte/s
etcd_grpc_call_rate	etcd 集群 gRPC 请求速率	次/s
etcd_grpc_call_failed_rate	etcd 集群 gRPC 请求失败速率	次/s
etcd_grpc_server_msg_received_rate	etcd 集群 gRPC 流式消息接收速率	次/s
etcd_grpc_server_msg_sent_rate	etcd 集群 gRPC 流式消息发送速率	次/s

指标名	说明	单位
	率	
etcd_disk_wal_fsync_duration	etcd 集群各节点 WAL 日志同步时间平均值	秒
etcd_disk_wal_fsync_duration_quantile	etcd 集群 WAL 日志同步时间平均值 (按分位数统计) <a href="#">4</a>	秒
etcd_disk_backend_commit_duration	etcd 集群各节点库同步时间 <a href="#">5</a> 平均值	秒
etcd_disk_backend_commit_duration_quantile	etcd 集群各节点库同步时间平均值 (按分位数统计)	秒
apiserver_up_sum	APIServer <a href="#">6</a> 在线实例数	
apiserver_request_rate	APIServer 每秒接受请求数	
apiserver_request_by_verb_rate	APIServer 每秒接受请求数 (按 HTTP 请求方法分类统计)	
apiserver_request_latencies	APIServer 请求平均迟延	秒
apiserver_request_by_verb_latencies	APIServer 请求平均迟延 (按 HTTP 请求方法分类统计)	秒
scheduler_up_sum	调度器 <a href="#">7</a> 在线实例数	
scheduler_schedule_attempts	调度器累计调度次数 <a href="#">8</a>	
scheduler_schedule_attempt_rate	调度器调度频率	次/s
scheduler_e2e_scheduling_latency	调度器调度延迟	秒
scheduler_e2e_scheduling_latency_quantile	调度器调度延迟 (按分位数统计)	秒
controller_manager_up_sum	Controller Manager <a href="#">9</a> 在线实例数	
coredns_up_sum	CoreDNS 在线实例数	
coredns_cache_hits	CoreDNS 缓存命中频率	次/s
coredns_cache_misses	CoreDNS 缓存未命中频率	次/s
coredns_dns_request_rate	CoreDNS 每秒请求数	

指标名	说明	单位
coredns_dns_request_duration	CoreDNS 请求耗时	秒
coredns_dns_request_duration_quantile	CoreDNS 请求耗时 (按分位数统计)	秒
coredns_dns_request_by_type_rate	CoreDNS 每秒请求数 (按请求类型分类统计)	
coredns_dns_request_by_rcode_rate	CoreDNS 每秒请求数 (按 rcode 分类统计)	
coredns_panic_rate	CoreDNS 异常发生频率	次/s
coredns_proxy_request_rate	CoreDNS 代理每秒请求数	
coredns_proxy_request_duration	CoreDNS 代理请求耗时	秒
coredns_proxy_request_duration_quantile	CoreDNS 代理请求耗时 (按分位数统计)	秒
prometheus_up_sum	Prometheus 在线实例数量	
prometheus_tsdb_head_samples_appended_rate	Prometheus 每秒存储监控指标数	

## 【说明】

**1** 如果某一节点返回值为 1 说明该 etcd 节点在线，0 说明节点下线。

**2** 如果某一节点返回值为 0 说明该节点没有leader，即该节点不可使用；如果集群中，所有节点都没有任何 leader，则整个集群不可用。

**3** 中英文对照说明：提案 (consensus proposals) ,失败提案 (failed proposals) ，已提交提案 (committed proposals) ，应用提案 (applied proposals) ，排队提案 (pending proposals) 。

**4** 支持三种分位数统计：99th 百分位数、90th 百分位数、中位数。

**5** 反映磁盘 I/O 延迟。如果数值过高，通常表示磁盘问题。

**6** 指 kube-apiserver。

**7** 指 kube-scheduler。

**8** 按调度结果分类统计：error (因调度器异常而无法调度的 Pod 数量) , scheduled (成功被调度的

Pod 数量) , unschedulable (无法被调度的 Pod 数量)。

<sup>9</sup> 指 kube-controller-manager。

## 安装常见问题

### KubeSphere 安装支持的存储类型

1、KubeSphere 安装支持的存储类型有哪些？如何配置？

答：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（企业级分布式存储）
- Ceph RBD
- GlusterFS
- NFS
- Local Volume（仅限 all-in-one 部署测试使用）

在安装前需要在 Installer 中配置已准备的存储服务端，配置方法详见 [存储安装配置说明](#)。

### 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@123456
```

```
node1 ansible_host=192.168.0.2 ip=192.168.0.2 ansible_user=ubuntu ansible_become_pass=Qcloud@123456
```

```
node2 ansible_host=192.168.0.3 ip=192.168.0.3 ansible_user=ubuntu ansible_become_pass=Qcloud@123456
```

```
[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

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	属性	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_001-28mpmazqne-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 +0800 (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
```

4.1、遇到ansible指令不能用，Command "python setup.py egg\_info" failed with error code 1 in/  
tmp/pip.build.344f90/ansible

答： 方法1：pip install setuptools==33.1.1 方法2：下载新版setuptools，[wget](https://files.pythonhosted.org/packages/e5/53/92a8ac9d252ec170d9197dcf988f07e02305a06078d7e83a41ba4e3ed65b/setup-tools-33.1.1-py2.py3-none-any.whl)  
<https://files.pythonhosted.org/packages/e5/53/92a8ac9d252ec170d9197dcf988f07e02305a06078d7e83a41ba4e3ed65b/setup-tools-33.1.1-py2.py3-none-any.whl> pip install setuptools-33.1.1-py2.py3-none-any.whl

4.2、dial tcp 20.233.0.1: 443: connect: no route to host"], "stdout": "", "stdout\_lines": []}

答：检查下是不是机器防火墙还开着 建议关掉 systemctl stop firewalld

4.3、FAELED – RETRYING: KubeSphere Waiting for ks-console (30 retries left)一直卡在这里。

答： 检查cpu和内存， 目前单机版部署至少8核16G 当 `free -m` 时buff/cache占用内存资源多， 执行 `echo 3 > /proc/sys/vm/drop_caches` 指令释放下内存

4.4、kubectl get pod –n kubesphere-system出现大量的Pending状态时

答： 可以kubectl describe看下这几个没起来的pod， 应该是内存不足

4.5、kubesphere v2.0.2 离线安装失败， 提示Failed to create 'IPPool' resource: resource already exists: IPPool(default-pool)"

答： 机器目前需要干净的机器， 如果已安装了k8s，则可以参考如下链接安装kubesphere：

<https://github.com/kubesphere/ks-installer>

4.6、centos7.4 2.0.2离线安装后， reboot系统重启服务不恢复

答： kube-system下的 coredns 没有起来， kubectl describe看下coredns的错误日志； coredns 无法连接 apiserver， 可以执行 `ipvsadm -Ln` 看下 6443 端口是否对应主节点地址

4.7、kubesphere能否独立使用

答：<https://github.com/kubesphere/ks-installer>

4.8、centos7.6 file:///kubeinstaller/yumrepo/iso/repodata/repo.xml: [Errno 14] curl#37 – "Couldn't open file /kubeinstaller/yumrepo/iso/repodata/repo.xml" Trying other mirror.

答： centos7.6系统还不支持离线下载部署。可以用centos7.4或7.5或在线部署。如果环境可以联网，建议使用在线安装，支持centos7.6 链接：<https://pan.baidu.com/s/1HBIBaD69mx6z050sSgj0Kg> 提取码：eqe6 把该iso放到离线环境的kubesphere-all-offline-advanced-2.0.2/Repos/下， `umount /kubeinstaller/yum_repo/iso` 重新跑

4.9、离线安装失败转为在线安装

答： 刚才执行离线安装修改的yum源，先还原一下，/etc/yum.repo.d 有备份

4.10、Could not find a version that satisfies the requirement ansible==2.7.6 (from -r /home/sunhaizhou/all-in-one/kubesphere-all-offline-advanced-2.0.2/scripts/os/requirements.txt (line 1)) (from versions: ) Error: Install the pip packages failed!

答：方法1：`pip install ansible==2.7.6` 方法2：可以手动umount一下/kubeinstaller/pip\_repo/pip27/iso 或者重启机器 应该可以解决该问题

4.11、fatal: [ks-allinone]: FAILED! => {"ansible\_facts": {"pkgmgr": "yum"}, "changed": false, "msg": "The Python 2 bindings for rpm are needed for this module. If you require Python 3 support use the dnf Ansible module instead.. The Python 2 yum module is needed for this module. If you require Python 3 support use the dnf Ansible module instead."}

答：将机器的python3改成python2 4.12、KubeSphere 2.0.1 安装失败：no matches for kind "S2iBuilderTemplate\"

答：`kubectl get pvc --all-namespaces` 看下pvc是否处于pending状态 如果pvc pending，地址和路径也都配置正确的话，可以试下本地机器上是否可以挂载；如果是自建的nfs，可以看下nfs的配置参数。先执行uninstall，然后再install。

4.13、fatal: [hippo04kf]: FAILED! => {"changed": true, "msg": "non-zero return code", "rc": 5, "stderr": "Warning: Permanently added '99.13.XX.XX' (ECDSA) to the list of known hosts.\r\nnPermission denied, please try again(publickey,gssapi-keyex,gssapi-with-mic).\r\n", "stderr\_lines": ["Warning: Permanently added '99.13.XX.XX' (ECDSA) to the list of known hosts."]}

答：只贴了all的格式，`ansibleshpass`改成`ansiblebecomepass`，最后执行脚本时，需要转成root用户执行。

[all]

```
master ansible_connection=local ip=192.168.0.5 ansible_user=tester ansible_become_pass=@@  
node1 ansible_host=192.168.0.6 ip=192.168.0.6 ansible_user=tester ansible_become_pass=@@
```

4.14、ks-devops/jenkins unable to parse quantity's suffix,error found in #10 byte of

答：有特殊字符无法解析 如果编辑过jenkins的相关配置的话可以检查下是不是写入了特殊字符

4.15、kubesphere-all-offline-advanced-2.0.0 离线安装提示 Failed connect to  
192.168.10.137:5080; Connection refused

答：`docker ps -al|grep nginx` nginx是否正常启动，且5080端口已监听。`df -hT|grep -v docker|grep -v kubelet` 看pip和iso是否都已挂载。以上都有问题的话，机器先停止运行，然后执行uninstall脚本，再运行install脚本。

#### 4.16、failed ansibelundefinedvariable dict object has no attribute address

答： 第一行这样配试下， ip换成自己的，如果有网的话建议联网安装

```
ks-allinone ansible_connection=local ip=192.168.0.2
[local-registry]
ks-allinone
[kube-master]
ks-allinone
[etcd]
ks-allinone
[kube-node]
ks-allinone
[k8s-cluster:children]
kube-node
kube-master
```

#### 4.17、FAILED! => {"reason": "'delegate\_to' is not a valid attribute for a TaskInclude

答： 不要手动安装ansible， 如果安装过程中出现了ERROR: Command "python setup.py egg\_info" failed with error code 1 in /tmp/pip-install-voTIRk/ansible/. ./multi-node.sh: line 41: ansible-playbook: command not found **pip install --upgrade setuptools==30.1.0先执行这个再安装**

#### 4.18、centos7.6 FAILED – RETRYING: KubeSphere| Installing JQ (YUM) (5 retries left),没有jq这个package， 我下了jq的二进制包放在/usr/bin目录下， jq可以使用。

答： 可以在kubesphere/roles/prepare/nodes/tasks/main.yaml中注释掉安装jq的相关tasks

#### 4.19、FAILED – RETRYING: ks-alerting | Waiting for alerting-db-init (2 retries left).fatal: [ks-allinone]: FAILED! => {"attempts": 3, "changed": true, "cmd": "/usr/local/bin/kubectl -n kubesphere-alerting-system get pod | grep alerting-db-init | awk '{print \$3}'", "delta": "stdout": "Init:0/1"

答： 检查配置是否满足要求8核cpu、16G。检查存储相关配置。检查本地/etc/resolve.conf下的域名是否都可以ping通。

#### 4.20、2.0.2 版本安装后配置docker 私有库问题， 配置了一个 docker 私有库， 修改完 /etc/docker/daemon.json 文件， 然后重新启动 docker报错。

答：kubesphere 在安装时指定了 DOCKER\_OPTS= --insecure-registry，所以不能再加载 /etc/docker/daemon.json 的 insecure-registries 设置，如果要添加的，/etc/systemd/system/docker.service.d文件中添加，重启。

4.21、用的自己搭建的nfs服务器，发现创建的pvc全是pending导致了相关的pod启动不起来。

答：可以参考 \*(rw,insecure,sync,no\_subtreecheck,no\_root\_squash)这个配置下nfs，然后先试下主机上是否可以挂载nfs

4.22、ubuntu系统，非root用户安装时，在kubernetes-apps/network\_plugin/calico: start calico resources get <http://localhost:8080/api?timeout=32s>: dial tcp 127.0.0.1: 8080: connect: connection refused

答：采用root用户安装脚本。4.23、FAILED – RETRYING: container\_download | Download containers if pull is required or told to always pull (all nodes) (4 retries left)

答：由于到dockerhub网络不是特别好造成的。

## QingCloud 云平台块存储卷问题

5、为什么存储卷创建失败？

答：用户可以查看存储卷的事件，查看 kube-system 项目的 csi-qingcloud-controller-0 容器组的 csi-qingcloud 容器日志，寻找错误关键字。详细问题原因列表：

问题字样	说明	解决办法
pki access frequency exceed permission denied	此问题通常由其他存储或者集群问题引起，需要继续检查集群状态，有无长时间正在删除的 pvc。	PKI 密钥访问次数过多，调大 PKI 密钥访问频率配额。
resource quota exceed permission denied	云平台配额限制	提工单调大相关资源配额，如硬盘容量和个数
IO timeout	容器组与云平台 API Server 通信问题，	使用 <a href="#">QingCloud CLI</a> 检查

问题字样	说明	解决办法
	通常私有云出现	config.yaml 配置文件正确性。可能是没有安装云平台 API Server 或 Kubernetes 集群内部通信问题
Cannot resolve host	私有云环境用户往往仅配置了 hosts 解析云平台 API Server 域名，在容器组内无此记录，造成无法通过 CSI 插件调用云平台 API Server。	需要用户配置 DNS 服务器。

## 6、为什么存储卷挂载到容器组失败？

答：用户可以查看容器组事件，查看 kube-system 项目的 csi-qingcloud-controller-0 容器组的 csi-qingcloud 容器日志，寻找错误关键字。详细问题原因列表：

问题字样	说明	解决办法
pki access frequency exceed permission denied	此问题通常由其他存储或者集群问题引起，需要继续检查集群状态，有无长时间正在删除的存储卷。	PKI 密钥访问次数过多，调大 PKI 密钥访问频率配额。
cannot find device path	云平台挂盘没主机设备路径。可在 QingCloud Console 查看硬盘的确无设备路径	将存储卷挂载的容器组所属的工作负载副本数设置为 0。容器组删除成功后，再将工作负载副本数设置为 1 即可
PermissionDenied, ... volume can only be attached to ...	存储卷类型无法挂载至容器组调度的节点的类型	改变存储卷类型或设置容器组调度规则
PermissionDenied, you can only attach [10] volumes to one instance at most	云平台单个主机挂盘个数超过限制	尝试让容器组重新调度，如：将存储卷挂载的容器组所属的工作负载副本数设置为 0。容器组删除成功后，再将工作负载副本数设置为 1
IO timeout	容器组与云平台 API Server 通信问题，通常私有云出现	使用 <a href="#">QingCloud CLI</a> 检查 config.yaml 配置文件正确性。可能是没有安装云平台 API Server 或 Kubernetes 集群内部通信问题
Cannot resolve	私有云环境用户往往仅配置了 hosts	需要用户配置 DNS 服务器。

问题字样	说明	解决办法
host	解析云平台 API Server 域名，在容器组内无此记录，造成无法通过 CSI 插件调用云平台 API Server。	

# 存储常见问题

## 物理环境灾难恢复相关问题

若物理环境的集群中某台节点意外关机或故障，如何将该节点上运行的有状态应用快速调度到其它节点上同时保证数据安全？

答：以 Ceph RBD 和物理 NeonSAN 作为集群存储为例，分别说明其灾难恢复的方法。

### Ceph RBD

若集群的持久化存储服务端配置的是 Ceph RBD，当其中某台节点意外关机或故障时，可参考如下步骤：

1、查看集群节点的状态，定位故障或关闭的主机，通常该节点的状态（STATUS）将显示 NotReady。

```
$ kubectl get node -o wide
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE
i-njy2abnw NotReady <none> 2d7h v1.13.5 172.17.0.8 <none> Ubuntu 16.04.3 LTS 4.4.0-116-
...
```

2、查看集群中 Pod 的状态，集群节点的状态为 NotReady 后，通常该节点上的 Pod 状态是 Terminating，若通过 kubectl 命令确定该 Pod 正是运行在故障节点上，则需要强行删除 Terminating 的 Pod：

```
$ kubectl delete pod pod_name --grace-period=0 --force
```

3、等待一段时间（约几分钟），当旧 Pod 被删除后，RBD 的 PVC 将会自动挂载至 Kubernetes 新创建的 Pod 中，新创建的 Pod 也将被自动调度到新节点上运行，此时可通过 kubectl get 或 describe 验证 Pod 的运行恢复和挂盘情况。

### NeonSAN

若集群的持久化存储服务端配置的是 NeonSAN，当其中某台节点意外关机或故障时，可参考如下步骤：

1、通过命令 `kubectl get node -o wide` 查看集群节点的状态，定位故障或关闭的主机，通常该节点的状态 (STATUS) 将显示 NotReady。

2、查看集群中 Pod 的状态，集群节点的状态为 NotReady 后，通常该节点上的 Pod 状态是 Terminating，若通过 kubectl 命令确定该 Pod 正是运行在故障节点上，则需要强行删除 Terminating 的 Pod：

```
$ kubectl delete pod pod_name --grace-period=0 --force
```

3、找到 Terminating 的 Pod 挂载的 PVC，假如我们发现 redis 的 Pod 状态为 Terminating，先找到该 PVC 的 VOLUME 名称 (以下返回的是 `pvc-1023666b779211e9`)：

```
$ kubectl get pvc --all-namespaces | grep redis
kubesphere-system redis-pvc Bound pvc-1023666b779211e9 100Gi RWO csi-qing
```

4、通过 VOLUME 名称找到对应的 volumeattachment，可通过 `-o yaml` 命令重定向到文件 (以下返回的是 `csi-90ef9b5f64f8156db2a2230f2c634e3afbd2abd5fd9c8238a0363a9b374e21d9`)：

```
$ kubectl get volumeattachment -o yaml | grep pvc-1023666b779211e9 -B 14
- apiVersion: storage.k8s.io/v1
  kind: VolumeAttachment
  ...
  - external-attacher/csi-qingcloud
  name: csi-90ef9b5f64f8156db2a2230f2c634e3afbd2abd5fd9c8238a0363a9b374e21d9
  ...
  source:
    persistentVolumeName: pvc-1023666b779211e9
```

5、删除对应的 volumeattachment：

```
$ kubectl delete volumeattachment csi-90ef9b5f64f8156db2a2230f2c634e3afbd2abd5fd9c82
```

6、等待一段时间，当旧 Pod 被删除后，NeonSAN 的 PVC 将会自动挂载至 Kubernetes 新创建的 Pod 中，新创建的 Pod 也将被自动调度到新节点上运行，此时可通过 kubectl get 或 describe 验证 Pod

的运行恢复和挂盘情况。

## 控制台使用常见问题

### 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。

### 如何查看 kubeconfig 文件

2、如何查看当前集群的 Kubeconfig 文件？

答：用户可以点击“小锤子”工具箱的图标，选择「kubeconfig」即可查看，仅管理员用户有权限查看。



### CPU 用量异常问题

3、为何安装后 CPU 用量数值异常大？



答：Kubesphere 计算 CPU 用量的方法是求 user 、 nice 、 system 、 iowait 、 irq 、 softirq 以及 steal 七种 CPU 模式下的用量合，数据通过 node\_exporter 采集。由于 Linux 内核 4.x 存在 steal 数值异常大的 bug，导致了如上图的异常值。建议尝试重启节点主机，或升级内核版本。

更多信息请参考 [node exporter issue #742](#)

4、每个节点默认的最大 Pod 上限数目为 110，若要调整每个节点的 Pod 上限数目要怎么配置？

答：

如果还没安装，安装前在 installer 可以修改 vars.yml 配置文件中的 kubeletmaxpods。

KubeSphere 安装后，可以在后台手工修改 master 节点 `/etc/kubernetes/kubelet.env` 中的 `max-pods` 参数，然后执行 `systemctl restart kubelet` 重启 kubelet 服务。

5、我在后台通过 kubectl 创建的 namespace 以及该 namespace 下的资源，要如何添加到企业空间 (workspace) 下呢？

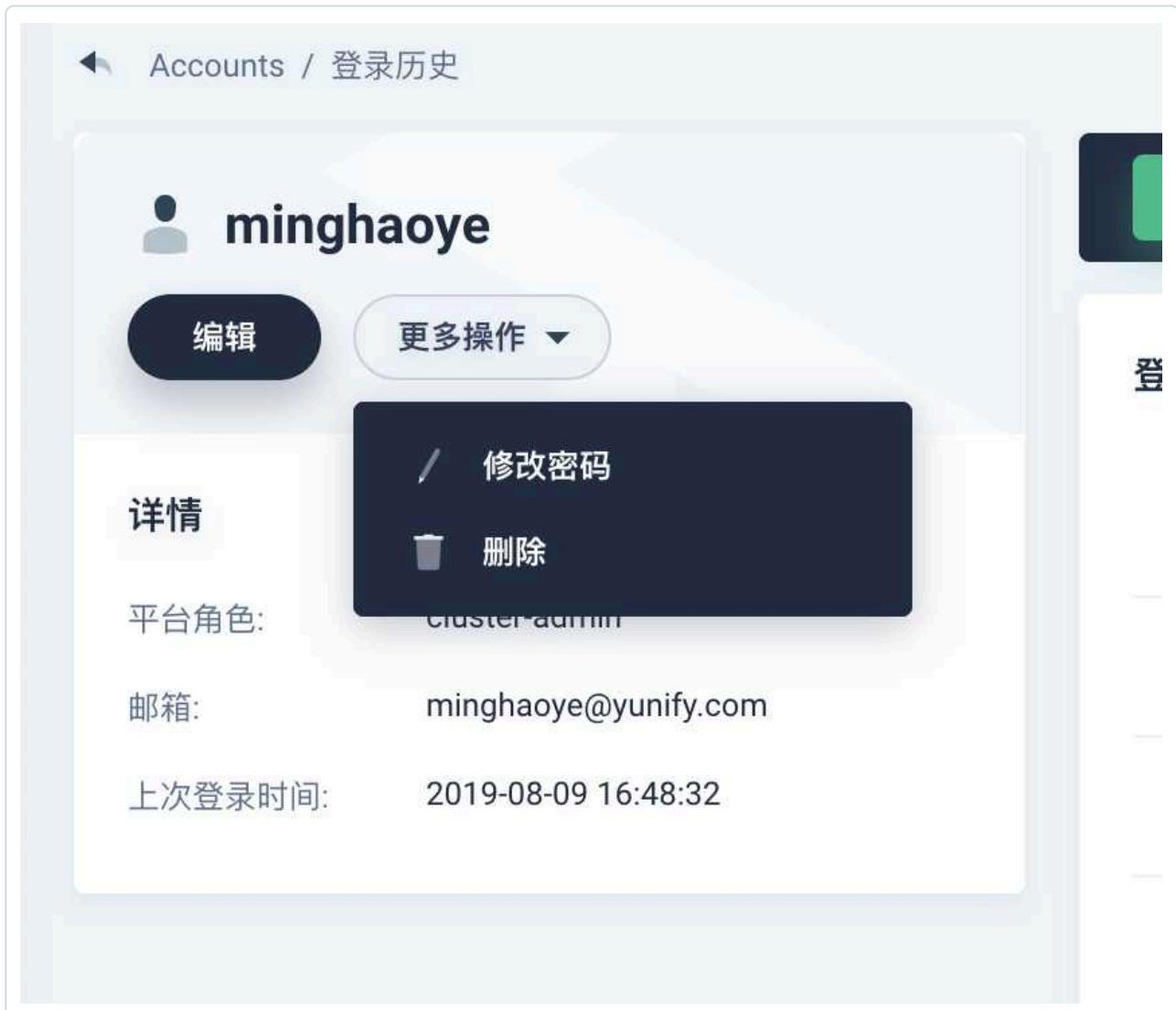
答：可以对新建的 namespace 打 label，指定其加入至哪一个企业空间 (workspace) 下。如下，将 echo-production 加入企业空间 demo-workspace 中。

```
kubectl label namespace echo-production kubesphere.io/workspace=demo-workspace
namespace/echo-production labeled
```

The screenshot shows the KubeSphere platform management interface. On the left, there is a sidebar with navigation items: 平台管理 (Platform Management), 工作台 (Workstation), 应用模板 (Application Template), 企业空间 (Enterprise Space) (selected, showing demo-workspace), 概览 (Overview), 项目管理 (Project Management) (selected), DevOps 工程 (DevOps Project), and 企业空间管理 (Enterprise Space Management). The main content area is titled '项目' (Project) with a sub-instruction: 'KubeSphere 中的项目对应的是 Kubernetes 的 namespace，是对一组资源和对象的抽象集合，常用来将系统内部的对象划分为不同的项目组或用户组。' (In KubeSphere, a project corresponds to a Kubernetes namespace, which is an abstract collection of resources and objects, often used to divide objects within the system into different project groups or user groups.). Below this is a search bar with placeholder text '请输入关键字进行查找' (Enter key to search) and a '刷新' (Refresh) button. A table lists projects with columns: 名称 (Name), 容器组数量 (Container Group Count), CPU 使用量 (CPU Usage), 内存使用量 (Memory Usage), 管理员 (Administrator), and 创建时间 (Creation Time). Two projects are listed: 'demo-project' and 'echo-production'. The row for 'echo-production' is highlighted with a red border.

6、admin 账号的密码忘记了，如何重置？

答：如果还有可以登录的管理员账号的话，可以到 **平台管理->账号管理->用户列表->单个用户详情页->更多操作->修改密码**，这样去重置 admin 的密码。



如果没有其它管理员账号，则需要进入到 `ks-account` 的 pod 里去执行：

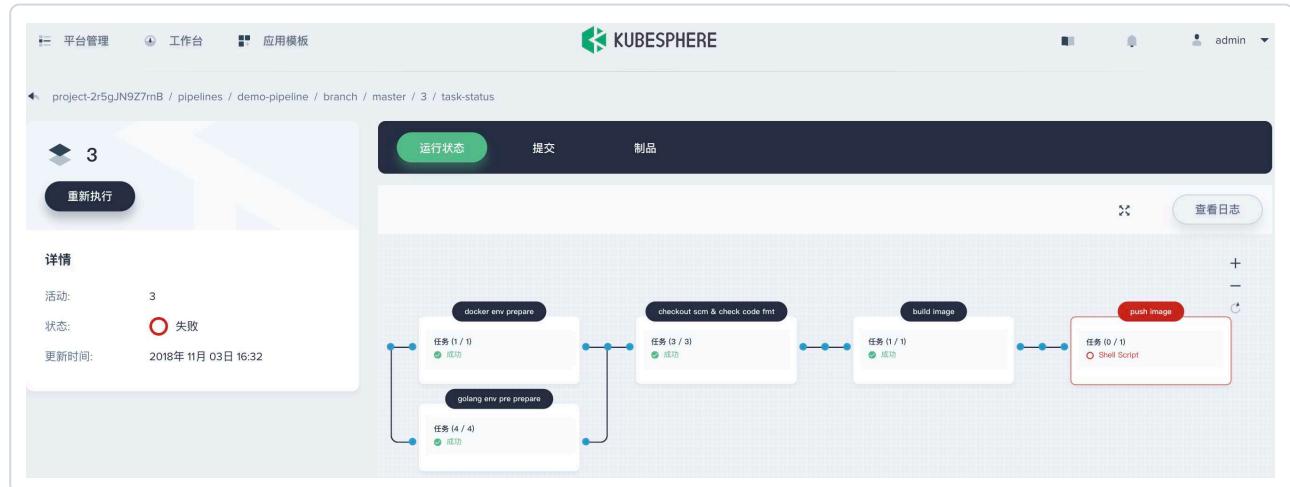
```
apk add curl && curl -X PUT http://localhost:9090/kapis/iam.kubesphere.io/v1alpha2/users/a
```

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。

# DevOps 常见问题

## 流水线运行报错相关问题

### 1、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 detailed log for the failed 'push image' stage. The left sidebar lists stages: docker env prepare, golang env pre prepare, checkout scm & check, build image, and push image (highlighted with a blue border). The main area shows the log output for the 'Shell Script' step under the 'push image' stage. The log includes:

```
[9Z7rnB_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
```

At the bottom right of the log area, there is a '关闭' (Close) button.

2、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？

The screenshot shows a Jenkins pipeline log titled "流水线运行日志". On the left, a list of pipeline stages is shown: "checkout scm" (green checkmark), "get dependencies" (green checkmark), "unit test" (green checkmark), "build & push snapshot" (red circle), and "push latest" (grey circle). The "build & push snapshot" stage is expanded, showing two "Shell Script" steps. The first step, "Running shell script", took 54.06 seconds and was successful (green checkmark). The second step, "16.31 s" (red circle), failed. The log output for the failing step shows a Docker build command and a failed GET request to the Docker registry, indicating a timeout or connection issue.

答：可能由于网络问题造成，建议尝试再次运行该流水线。

3、流水线运行时遇到如下报错应如何处理？

```
+ git push http://****:****@github.com/****/devops-java-sample.git --tags
fatal: unable to access 'http://****:****@github.com/****/devops-java-sample.git/': Could not
script returned exit code 128
```

答：可能是 GitHub 账号或密码带有 "@" 这类特殊字符，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。

# 部署 Ceph RBD 存储服务端

## 简介

[Ceph](#) 是一个分布式存储系统，本指南将介绍如何在 Ubuntu 系统部署一个节点数为 2 的 Ceph (v10.2.10) 存储服务端集群。本指南仅供测试 KubeSphere 存储服务端的搭建，正式环境搭建 Ceph 集群请参考 [Ceph 官方文档](#)。

## Ceph 基本组件

Ceph 主要有三个基本进程：

- OSD 用于集群中所有数据与对象的存储，处理集群数据的复制、恢复、回填、再均衡，并向其他 osd 守护进程发送心跳，然后向 Monitor 提供一些监控信息。
- Monitor 监控整个集群的状态，维护集群的 cluster MAP 二进制表，保证集群数据的一致性。
- MDS (可选) 为 Ceph 文件系统提供元数据计算、缓存与同步。MDS 进程并不是必须的进程，只有需要使用 CephFS 时，才需要配置 MDS 节点。

## 准备节点

### 主机规格

Hostname	IP	OS	CPU	RAM	Device
ceph1	172.20.1.7	Ubuntu 16.04.4	4 Core	4 GB	/dev/vda 100 GiB
ceph2	172.20.1.8	Ubuntu 16.04.4	4 Core	4 GB	/dev/vda 100 GiB

### 集群架构



注：

- `ceph1` 作为集群的管理主机，用来执行安装任务。
- 如需创建更大容量的 Ceph 存储服务端，可创建更大容量主机磁盘或挂载大容量磁盘至主机并挂载至 `ceph1` 的 `/osd1` 和 `ceph2` 的 `/osd2` 文件夹。
- 两个节点的 Hostname 需要与主机规格的列表中一致，因为后续步骤的命令行中有匹配到 Hostname，若与以上列表不一致请注意在后续的命令中对应修改成实际的 Hostname。

## 预备工作

### 配置 root 登录

1、参考如下步骤分别为 `ceph1` 和 `ceph2` 配置 root 用户登录：

- 1.1. ubuntu 账户登录主机后切换 root 用户：

```

ubuntu@ceph1:~$ sudo -i
[sudo] password for ubuntu:
root@ceph1:~#

```

- 1.2. 设置 root 用户登录密钥：

```
root@ceph1:~# passwd
```

- 同上，在 `ceph2` 修改 root 密码。

## 修改 hosts 文件

2、参考如下步骤修改 ceph1 和 ceph2 的 hosts 文件：

```
root@ceph1:~# vim /etc/hosts
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# hostname loopback address
172.20.1.7 ceph1
172.20.1.8 ceph2
```

- 同上，在 ceph2 执行以上命令并修改 hosts。

## 配置 SSH 免密登录

3、以下为 ceph1 的 root 用户配置无密码登录至 ceph1 与 ceph2。

- 3.1. 创建密钥，提示“Enter passphrase”时，直接回车，口令即为空：

```
root@ceph1:~# ssh-keygen
```

- 3.2. 拷贝密钥到两个 Ceph 节点，按照提示输入密钥：

```
root@ceph1:~# ssh-copy-id root@ceph1
...
root@ceph1:~# ssh-copy-id root@ceph2
...
```

- 3.3. 验证免密登录，即 ceph1 的 root 用户无需输入密码可以登录 ceph1 和 ceph2：

```
root@ceph1:~# ssh root@ceph1
root@ceph1:~# ssh root@ceph2
```

## 开始安装

### 安装 Ceph 和 ceph-deploy

4、Ceph 官方推出了一个用 python 写的工具 cpeh-deploy，可以很大程度地简化 Ceph 集群的配置过程，参考如下步骤为 ceph1 和 ceph2 安装 Ceph 和 ceph-deploy：

```
root@ceph1:~# apt-get update
root@ceph1:~# apt-get install -y ceph ceph-deploy
```

- 同上，在 ceph2 执行以上命令。

### 创建文件夹

5、参考如下步骤为 ceph1 和 ceph2 创建文件夹。

- 5.1. 在 ceph1 创建文件夹存放初始化配置：

```
root@ceph1:~# mkdir -p /root/cluster
root@ceph1:~# rm -f /root/cluster/*
```

- 5.2. 分别在 ceph1 和 ceph2 存放 ceph 数据：

```
root@ceph1:~# mkdir -p /osd1 & rm -rf /osd1/*
root@ceph1:~# chown ceph:ceph /osd1
```

```
root@ceph2:~# mkdir -p /osd2 & rm -rf /osd2/*
root@ceph2:~# chown ceph:ceph /osd2
```

- 5.3. 在 ceph1 创建 ceph 文件夹:

```
root@ceph1:~# mkdir -p /var/run/ceph/
root@ceph1:~# chown ceph:ceph /var/run/ceph/
```

- 同上，在 ceph2 执行以上命令创建 ceph 文件夹。

## 初始化 ceph

6、执行以下命令在 ceph1 节点初始化 ceph:

```
root@ceph1:~# cd /root/cluster
root@ceph1:~/cluster# ceph-deploy new ceph1
```

- 查看各文件夹内容:

```
root@ceph1:~/cluster# ls
ceph-deploy-ceph.log  ceph.conf  ceph.mon.keyring
root@ceph1:~/cluster# ls /etc/ceph/
rbdmap
root@ceph1:~/cluster# ls /var/run/ceph/
```

## 修改 Ceph 配置文件

7、在 ceph1 配置 `ceph.conf`，添加以下参数:

```
root@ceph1:~/cluster# vim ceph.conf
[global]
...
...
osd pool default size = 2
osd crush chooseleaf type = 0
osd max object name len = 256
osd journal size = 128
```

## 激活 Mon 节点

8、Mon 节点监控着整个 Ceph 集群的状态信息，监听于 TCP 的 6789 端口。每一个 Ceph 集群中至少要有一个 Mon 节点，如下在 ceph1 激活 Monitor：

```
root@ceph1:~/cluster# ceph-deploy mon create-initial
...
[ceph_deploy.gatherkeys][DEBUG ] Got ceph.bootstrap-rgw.keyring key from ceph1.
```

## 创建 OSD 节点

9、OSD 是强一致性的分布式存储，用于集群中所有数据与对象的存储，如下在 ceph1 为集群中两个节点创建 OSD：

```
root@ceph1:~/cluster# ceph-deploy osd prepare ceph1:/osd1 ceph2:/osd2
...
[ceph_deploy.osd][DEBUG ] Host ceph1 is now ready for osd use.
...
[ceph_deploy.osd][DEBUG ] Host ceph2 is now ready for osd use.
```

- 启动 OSD 节点：

```
root@ceph1:~/cluster# ceph-deploy osd activate ceph1:/osd1 ceph2:/osd2
```

## 拷贝配置

10、拷贝配置到 ceph1 和 ceph2：

```
root@ceph1:~/cluster# ceph-deploy admin ceph1 ceph2
```

```
root@ceph1:~/cluster# chmod +r /etc/ceph/ceph.client.admin.keyring
root@ceph2:~/cluster# chmod +r /etc/ceph/ceph.client.admin.keyring
```

## 验证安装结果

11、至此，一个简单的 Ceph 存储服务集群搭建就完成了，接下来查看安装的 Ceph 版本和状态信息。

- 11.1. 查看 Ceph 版本：

```
root@ceph1:~/cluster# ceph -v
ceph version 10.2.10 (5dc1e4c05cb68dbf62ae6fce3f0700e4654fdbbe)
```

- 11.2. 在两个节点检查 ceph 状态，可以使用 `ceph -s` 查看，如果是 health 显示 `HEALTH_OK` 状态说明配置成功。

## 使用 ceph 服务

12、首先，需要创建 rbd image，image 是 Ceph 块设备中的磁盘映像文件，可使用 `rbd create ...` 命令创建指定大小的映像。

- 12.1. 此处在 ceph1 以创建 foo 为例：

```
root@ceph1:~/cluster# rbd create foo --size 4096 --pool rbd --image-format=1
rbd: image format 1 is deprecated
```

- 12.2. 检查 rbd image:

```
root@ceph1:~/cluster# rbd ls
foo
```

- 12.3. 在 ceph1 把 foo image 映射到内核:

```
root@ceph1:~/cluster# rbd map foo
/dev/rbd0
```

- 12.4. 检查挂载状态:

```
root@ceph1:~/cluster# fdisk -l
...
Disk /dev/rbd0: 4 GiB, 4294967296 bytes, 8388608 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 4194304 bytes / 4194304 bytes
```

## 分区格式化

13、将 foo image 格式化为 ext4 格式的文件系统:

```
root@ceph1:~/cluster# mkfs.ext4 /dev/rbd0
```

- 13.1. 创建文件夹，然后挂载至目标文件夹:

```
root@ceph1:~/cluster# mkdir -p /mnt/rbd  
root@ceph1:~/cluster# mount /dev/rbd0 /mnt/rbd
```

- 13.2 在 ceph1 检查挂载结果:

```
root@ceph1:~/cluster# df -ah  
...  
/dev/rbd0      3.9G  8.0M  3.6G  1% /mnt/rbd
```

```
root@ceph1:~/cluster# ls /mnt/rbd/  
lost+found
```

## 释放资源

14、注意，在使用完毕之后，可参考如下步骤卸载和删除不需要的资源。

- 14.1. 参考如下将文件系统从文件中卸载:

```
root@ceph1:~/cluster# umount /mnt/rbd
```

- 14.2. 检查卸载结果:

```
root@ceph1:~/cluster# df -ah  
...
```

- 14.3. 卸载 rbd image:

```
root@ceph1:~/cluster# rbd unmap foo
```

- 14.4. 检查卸载结果:

```
root@ceph1:~/cluster# fdisk -l
Disk /dev/vda: 100 GiB, 107374182400 bytes, 209715200 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5735896b
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/vda1	*	2048	209713247	209711200	100G	83	Linux

```
Disk /dev/vdb: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

- 14.5. 删除 rbd image:

```
root@ceph1:~/cluster# rbd remove foo
Removing image: 100% complete...done.
```

# 部署 GlusterFS 存储服务端

GlusterFS 是一个开源的分布式文件系统，本指南将介绍如何在 Ubuntu 系统部署一个节点数为 2 的 GlusterFS (v3.12.12) 存储服务端集群和 Heketi，Heketi 用来管理 GlusterFS，并提供 RESTful API 接口供 Kubernetes 调用。本指南仅供测试 KubeSphere 存储服务端的搭建，正式环境搭建 GlusterFS 集群请参考 [GlusterFS 官方网站](#)，搭建 Heketi 请参考 [官方文档](#)。

## 准备节点

### 主机规格

Hostname	IP	OS	CPU	RAM	Device
glusterfs-server1	172.20.1.5	Ubuntu 16.04.4	4 Core	4 GB	/dev/vda 100GiB, /dev/vdc 300GiB
glusterfs-server2	172.20.1.6	Ubuntu 16.04.4	4 Core	4 GB	/dev/vda 100GiB, /dev/vdc 300GiB

### 注：

- glusterfs-server1 作为集群的管理主机，用来执行安装任务。
- 如需创建更大容量 GlusterFS 存储服务端，可挂载更大容量块存储磁盘至主机。
- 两个节点的 Hostname 需要与主机规格的列表中一致，因为后续步骤的命令行中有匹配到 Hostname，若与以上列表不一致请注意在后续的命令中对应修改成实际的 Hostname。
- GlusterFS 服务端将数据存储至 /dev/vdc 块设备中，/dev/vdc 必须是未经分区格式化的原始块设备。

### 集群架构



# 预备工作

## 配置 root 登录

1、参考如下步骤分别为 glusterfs-server1 和 glusterfs-server2 配置 root 用户登录：

- 1.1. ubuntu 用户登录主机后切换 root 用户：

```
ubuntu@glusterfs-server1:~$ sudo -i  
[sudo] password for ubuntu:  
root@glusterfs-server1:~#
```

- 1.2. 设置 root 用户登录密钥：

```
root@glusterfs-server1:~# passwd  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

- 同上，在 glusterfs-server2 修改 root 密码。

## 修改 hosts 文件

2、参考如下步骤修改 glusterfs-server1 和 glusterfs-server2 的 `hosts` 文件：

```
root@glusterfs-server1:~# vi /etc/hosts  
...  
  
# hostname loopback address  
172.20.1.5 glusterfs-server1  
172.20.1.6 glusterfs-server2
```

- 同上，在 glusterfs-server2 执行以上命令并修改 hosts。

## 配置 SSH 免密登录

3、以下为 glusterfs-server1 的 root 用户配置无密码登录至 glusterfs-server1 与 glusterfs-server2。

- 3.1. 创建密钥，提示“Enter passphrase”时，直接回车键，口令即为空：

```
root@glusterfs-server1:~# ssh-keygen
```

```
...
```

- 3.2. 拷贝密钥到各个 GlusterFS 节点，按照提示输入密钥：

```
root@glusterfs-server1:~# ssh-copy-id root@glusterfs-server1
```

```
...
```

```
root@glusterfs-server1:~# ssh-copy-id root@glusterfs-server2
```

```
...
```

- 3.3. 验证免密登录，即 glusterfs-server1 无需输入密码可以登录 glusterfs-server1 和 glusterfs-server2：

```
root@glusterfs-server1:~# ssh root@glusterfs-server1
```

```
root@glusterfs-server1:~# ssh root@glusterfs-server2
```

## 开始安装

### 安装 Glusterfs

4、以下用 `apt-get` 软件包管理工具在 glusterfs-server1 安装 GlusterFS：

```
root@glusterfs-server1:~# apt-get install software-properties-common  
root@glusterfs-server1:~# add-apt-repository ppa:gluster/glusterfs-3.12  
root@glusterfs-server1:~# apt-get update  
root@glusterfs-server1:~# apt-get install glusterfs-server -y
```

- 同上，在 glusterfs-server2 执行以上命令安装 GlusterFS。待安装完成后分别在两个节点检查安装的 GlusterFS 版本：

```
$ glusterfs -V  
glusterfs 3.12.12  
...
```

## 加载内核模块

5、执行以下命令为 glusterfs-server1 加载必需的三个内核模块：

```
root@glusterfs-server1:~# echo dm_thin_pool | sudo tee -a /etc/modules  
dm_thin_pool  
root@glusterfs-server1:~# echo dm_snapshot | sudo tee -a /etc/modules  
dm_snapshot  
root@glusterfs-server1:~# echo dm_mirror | sudo tee -a /etc/modules  
dm_mirror
```

```
root@glusterfs-server1:~# apt-get -y install thin-provisioning-tools
```

- 同上，在 glusterfs-server2 执行以上命令。

## 创建 Glusterfs 集群

6、参考如下步骤创建 GlusterFS 集群：

```
root@glusterfs-server1:~# gluster peer probe glusterfs-server2
peer probe: success.
```

```
root@glusterfs-server2:~# gluster peer probe glusterfs-server1
peer probe: success. Host glusterfs-server1 port 24007 already in peer list
```

## 验证安装结果

- 分别在 glusterfs-server1 和 glusterfs-server2 检查 GlusterFS 集群节点间的连接状态，若 State 显示 `Peer in Cluster (Connected)` 则说明 GlusterFS 集群已成功搭建：

```
$ gluster peer status
```

## 安装 Heketi

7、Heketi 是用来管理 GlusterFS 卷的生命周期的，并提供了一个 RESTful API 接口供 Kubernetes 调用，因为 GlusterFS 没有提供 API 调用的方式，所以我们借助 heketi，通过 Heketi，Kubernetes 可以动态配置 GlusterFS 卷。以下将介绍如何在 glusterfs-server1 安装 Heketi（v7.0.0）。

- 7.1. 下载 Heketi Installer:

```
root@glusterfs-server1:~# wget https://github.com/heketi/heketi/releases/download/v7.0.0/he
```

- 7.2. 解压并安装 Heketi:

```
root@glusterfs-server1:~# tar -xf heketi-v7.0.0.linux.amd64.tar.gz
root@glusterfs-server1:~# cd heketi/
root@glusterfs-server1:~/heketi# cp heketi /usr/bin/
root@glusterfs-server1:~/heketi# cp heketi-cli /usr/bin
```

## 配置 Heketi

8、参考如下步骤配置 Heketi：

- 8.1. 将 Heketi 纳入 `systemd` 管理：

```
root@glusterfs-server1:~# vi /lib/systemd/system/heketi.service
[Unit]
Description=Heketi Server
[Service]
Type=simple
WorkingDirectory=/var/lib/heketi
ExecStart=/usr/bin/heketi --config=/etc/heketi/heketi.json
Restart=on-failure
StandardOutput=syslog
StandardError=syslog
[Install]
WantedBy=multi-user.target
```

- 8.2. 创建文件夹：

```
root@glusterfs-server1:~# mkdir -p /var/lib/heketi
root@glusterfs-server1:~# mkdir -p /etc/heketi
```

- 8.3. 创建 Heketi 配置文件：

```
root@glusterfs-server1:~# vim /etc/heketi/heketi.json
{
  "_port_comment": "Heketi Server Port Number",
  "port": "8080",

  "_use_auth": "Enable JWT authorization. Please enable for deployment",
  "use_auth": false,

  "_jwt": "Private keys for access",
  "jwt": {
    "_admin": "Admin has access to all APIs",
    "admin": {
      "key": "123456"
    },
    "_user": "User only has access to /volumes endpoint",
    "user": {
      "key": "123456"
    }
  },
}

"_glusterfs_comment": "GlusterFS Configuration",
"glusterfs": {
  "_executor_comment": [
    "Execute plugin. Possible choices: mock, ssh",
    "mock: This setting is used for testing and development.",
    "It will not send commands to any node.",
    "ssh: This setting will notify Heketi to ssh to the nodes.",
    "It will need the values in sshexec to be configured.",
    "kubernetes: Communicate with GlusterFS containers over",
    "Kubernetes exec api."
  ],
  "executor": "ssh",

  "_sshexec_comment": "SSH username and private key file information",
  "sshexec": {
    "keyfile": "/root/.ssh/id_rsa",
    "user": "root"
  },
}

"_kubeeexec_comment": "Kubernetes configuration",
"kubeeexec": {
  "host": "https://kubernetes.host:8443",
```

## 启动Heketi

9、在 glusterfs-server1 安装 Heketi 后，需要启动 Heketi，Heketi 的状态 ”Active” 显示 `active (running)` ... 则说明成功启动：

```
root@glusterfs-server1:~# systemctl start heketi
root@glusterfs-server1:~# systemctl status heketi
● heketi.service – Heketi Server
   Loaded: loaded (/lib/systemd/system/heketi.service; disabled; vendor preset: enabled)
   Active: active (running) since Tue 2018-08-14 13:50:18 CST; 9ms ago
     Main PID: 6854 (heketi)
        Tasks: 4
       Memory: 1.3M
          CPU: 3ms
        CGroup: /system.slice/heketi.service
                └─6854 /usr/bin/heketi --config=/etc/heketi/heketi.json

Aug 14 13:50:18 glusterfs-server1 systemd[1]: Started Heketi Server.
```

```
root@glusterfs-server1:~# systemctl enable heketi
Created symlink from /etc/systemd/system/multi-user.target.wants/heketi.service to /lib/systemd/system/heketi.service.
```

## 编辑拓扑文件

10、参考如下步骤编辑 Heketi 的拓扑文件，以下所有 IP 地址应替换为您安装环境的实际主机 IP 地址。GlusterFS 服务端将数据存储至 `/dev/vdc` 块设备中，以下 `"/dev/vdc"` 可按实际情况修改：

```
root@glusterfs-server1:~# vim /etc/heketi/topology.json
```

```
{  
  "clusters": [  
    {  
      "nodes": [  
        {  
          "node": {  
            "hostnames": {  
              "manage": [  
                "172.20.1.5"  
              ],  
              "storage": [  
                "172.20.1.5"  
              ]  
            },  
            "zone": 1  
          },  
          "devices": [  
            "/dev/vdc"  
          ]  
        },  
        {  
          "node": {  
            "hostnames": {  
              "manage": [  
                "172.20.1.6"  
              ],  
              "storage": [  
                "172.20.1.6"  
              ]  
            },  
            "zone": 1  
          },  
          "devices": [  
            "/dev/vdc"  
          ]  
        }  
      ]  
    }  
  ]  
}
```

## 载入拓扑文件

11、执行以下命令为 Heketi 载入拓扑文件：

```
root@glusterfs-server1:~# export HEKETI_CLI_SERVER=http://localhost:8080  
root@glusterfs-server1:~# heketi-cli topology load --json=/etc/heketi/topology.json
```

## 验证 Heketi 安装

12、执行以下命令查看 Heketi 的安装信息：

```
heketi-cli volume list --secret 123456 --user admin
```

# 云平台配置端口转发和防火墙

在 KubeSphere 中，若服务、应用路由或外网访问启用的是 NodePort 的方式，NodePort 会在主机上开放一个节点端口。一般来说，如果集群部署在 VPC 中，那么在云平台需要将该端口进行转发并在防火墙添加下行规则，确保流量能够通过该端口。

例如，在 QingCloud 云平台进行上述操作，假设在集群开启的节点端口 NodePort 为 31680，则需要参考如下步骤：

## a.端口转发

其中内网 IP 可选集群中任意一个节点，例如 192.168.0.8。

- 1、点击「VPC 网络」进入集群所属的 VPC。
- 2、点击「管理配置」。
- 3、点击「添加规则」，在弹窗中填写端口转发的规则，将示例端口 31680 转发出来，详见 [QingCloud SDN 官方文档](#)。

The screenshot shows the 'VPC 网络' section of the QingCloud management console. The left sidebar lists network components like '私有网络', '隧道服务', '路由推送', and '网关过滤控制(ACL)'. The '端口转发' tab is selected. A modal window titled 'bookinfo' is open, showing a table with one row: 'bookinfo-test' (协议: tcp, 源端口: 31680, 内网 IP: 192.168.0.8, 内网端口: 31680). Three numbered callouts point to the steps: ① points to the 'VPC 网络' button; ② points to the '管理配置' button; ③ points to the '+ 添加规则' button.

## b.添加防火墙规则

进入当前 VPC 的防火墙，为主机端口 31680 添加一条下行规则。

+ 添加规则

应用修改

下行规则 (从外部访问云资源)

提示：未配置的下行规则和端口默认拒绝访问。TCP 端口 445 / 5554 / 9996 是病毒“震荡波”所使用的端口，可能会被 IDC 屏蔽，为保证资源正常访问，建议使用其他端口。

<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"/>	bookinfo-test	10	TCP	接受	31680			禁用

合计 : 1 每页: 50