阿里云
容器服务Kubernetes版

用户指南

文档版本：20181123

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 Kubernetes 集群

1.1 简介

1.1.1 概述

Kubernetes 是流行的开源容器编排技术。为了让用户可以方便地在阿里云上使用 Kubernetes 管理容器应用，阿里云容器服务提供了 Kubernetes 集群支持。

您可以通过容器服务管理控制台创建一个安全高可用的 Kubernetes 集群，整合阿里云虚拟化、存储、网络和安全能力，提供高性能可伸缩的容器应用管理能力，简化集群的搭建和扩容等工作，让您专注于容器化的应用的开发与管理。

Kubernetes 支持对容器化应用程序的部署、扩展和管理。它具有以下功能：

- 弹性扩展和自我修复
- 服务发现和负载均衡
- 服务发布与回滚
- 机密和配置管理

使用限制

- 目前 Kubernetes 集群只支持 Linux 容器，对 Kubernetes 的 Windows 容器的支持在计划中。
- 目前 Kubernetes 集群只支持 VPC 网络。您可以在部署 Kubernetes 集群时选择创建一个新的 VPC 或者使用已有的 VPC。

相关开源项目

- 阿里云 Kubernetes Cloud Provider 实现：*https://github.com/AliyunContainerService/kubernetes*
- Flannel 的阿里云 VPC 网络驱动：*https://github.com/coreos/flannel/blob/master/Documentation/alicloud-vpc-backend.md*

如果您对相关项目有问题或者建议，欢迎在社区提交 Issue 或者 Pull Request。

1.1.2 阿里云 Kubernetes vs. 自建 Kubernetes

阿里云 Kubernetes 的优势

便捷

- 通过 Web 界面一键创建 Kubernetes 集群。
- 通过 Web 界面一键完成 Kubernetes 集群的升级。

  您在使用自建 Kubernetes 集群的过程中，可能需要同时处理多个版本的集群（包括 1.8.6、1.9.4、以及未来的 1.10）。每次升级集群的过程都是一次大的调整和巨大的运维负担。容器服务的升级方案使用镜像滚动升级以及完整元数据的备份策略，允许您方便地回滚到先前版本。
- 通过 Web 界面轻松地实现 Kubernetes 集群的扩容和缩容。

  使用容器服务 Kubernetes 集群可以方便地一键垂直伸缩容来快速应对数据分析业务的峰值。

强大

功能	说明
网络	• 高性能 VPC 网络插件。 • 支持 network policy 和流控。 容器服务可以为您提供持续的网络集成和最佳的网络优化。
负载均衡	支持创建负载均衡实例（公网、内网）。 如果您在使用自建 Kubernetes 集群的过程使用自建的 Ingress 实现，业务发布频繁可能会造成 Ingress 的配置压力并增加出错概率。容器服务的 SLB 方案支持原生的阿里云高可用负载均衡，可以自动完成网络配置的修改和更新。该方案经历了大量用户长时间的使用，稳定性和可靠性大大超过用户自建的入口实现。
存储	集成阿里云云盘、文件存储NAS、块存储，提供标准的 FlexVolume 驱动。 自建 Kubernetes 集群无法使用云上的存储资源，阿里云容器服务提供了最佳的无缝集成。
运维	• 集成阿里云日志服务、云监控 • 支持弹性伸缩
镜像仓库	• 高可用，支持大并发。 • 支持镜像加速。 • 支持 p2p 分发。 您如果使用自建的镜像仓库，在百万级的客户端同时拉取镜像的时候，会存在镜像仓库崩溃的可

功能	说明
	能性。使用容器服务镜像仓库来提高镜像仓库的可靠性，减少运维负担和升级压力。
稳定	• 专门的团队保障容器的稳定性。 • 每个 Linux 版本，每个 Kubernetes 版本都会在经过严格测试之后之后才会提供给用户。 容器服务提供了 Docker CE 兜底和推动 Docker 修复的能力。当您遇到 Docker Engine hang、网络问题、内核兼容等问题时，容器服务可以为您提供最佳实践。
高可用	• 提供多可用区支持。 • 支持备份和容灾。
技术支持	• 提供 Kubernetes 升级能力，新版本一键升级。 • 阿里云容器团队负责解决在环境中遇到的各种容器问题。

自建 **Kubernetes** 的成本和风险

- 搭建集群繁琐。

  您需要手动配置 kubernetes 相关的各种组件、配置文件、证书、密钥、相关插件和工具，整个集群搭建工作需要专业人员数天到数周的时间。
- 在公共云上，需要投入大量的成本实现和云产品的集成。

  和阿里云上其他产品的集成，需要您自己投入成本来实现，如日志服务、监控服务和存储管理等。
- 容器是一个系统性工程，涉及网络、存储、操作系统、编排等各种技术，需要专门的人员投入。
- 容器技术一直在不断发展，版本迭代快，需要不断的踩坑、升级、测试。

1.1.3 Kubernetes集群稳定性最佳实践

在生产环境中使用Kubernetes集群时，为了保证稳定可靠的运行在Kubernetes里，下面将介绍构建Kubernetes集群的最佳实践。

Master节点规格

通过容器服务创建的Kubernetes集群，Master节点上运行着etcd、kube-apiserver、kube-controller等核心组件，对于Kubernetes集群的稳定性有着至关重要的影响，对于生产环境的集群，必须慎重选择Master规格。Master规格跟集群规模有关，集群规模越大，所需要的Master规格也越高。

说明：

您可从多个角度衡量"集群规模"：节点数量、Pod数量、部署频率、访问量。

这里简单的认为集群规模就是集群里的节点数量。对于常见的集群规模，可以参考这种如下的方式选择Master节点的规格（对于测试环境，规格可以小一些。下面的选择能尽量保证Master负载维持在一个较低的水平上）：

节点规模	Master规格
1-5个节点	4C8G(不建议2C4G)
6-20个节点	4C16G
21-100个节点	8C32G
100-200个节点	16C64G

选择合理的磁盘大小

Kubernetes节点需要的磁盘空间也不小，Docker镜像、系统日志、应用日志都保存在磁盘上。创建集群的时候，要考虑每个节点上要部署的Pod数量，每个Pod的日志大小、镜像大小、临时数据，再加上一些系统预留的值。

说明：

Kubernetes集群中，ECS操作系统占用3G的磁盘空间，剩余的磁盘空间由Kubernetes资源对象使用。

节点类型	磁盘大小
Master节点	经典Dedicated Kubernetes集群的系统盘默认大小为120GB，您可进行设置，不低于40GB。
Worker节点	经典Dedicated Kubernetes集群的系统盘默认大小为40GB；可选择挂载数据盘，默认大小为100GB，不低于40GB。

使用多可用区

阿里云支持多Region（地域），每个Region下又有不同的可用区。可用区是指在同一地域内，电力和网络互相独立的物理区域。多可用区能够实现跨地域的容灾能力。会带来额外的网络延时。创建Kubernetes集群时，您可选择创建多可用区Kubernetes集群。参见创建多可用区 *Kubernetes* 集群。

声明每个**Pod**的**resource**

您在使用Kubernetes集群时，经常遇到一个问题：在一个节点上调度了太多的Pod，导致节点负载太高，没法正常对外提供服务。怎么避免这种情况出现呢？

在Kubernetes中部署Pod时，您可以指定这个Pod需要的资源，Kubernetes在部署这个Pod的时候，就会根据Pod的需求找一个具有充足空闲资源的节点部署这个Pod。下面的例子中，声明tomcat这个Pod需要0.25核CPU，64M的内存，运行中实际使用不能超过0.5核CPU和128M内存。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    resources: # 资源声明
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

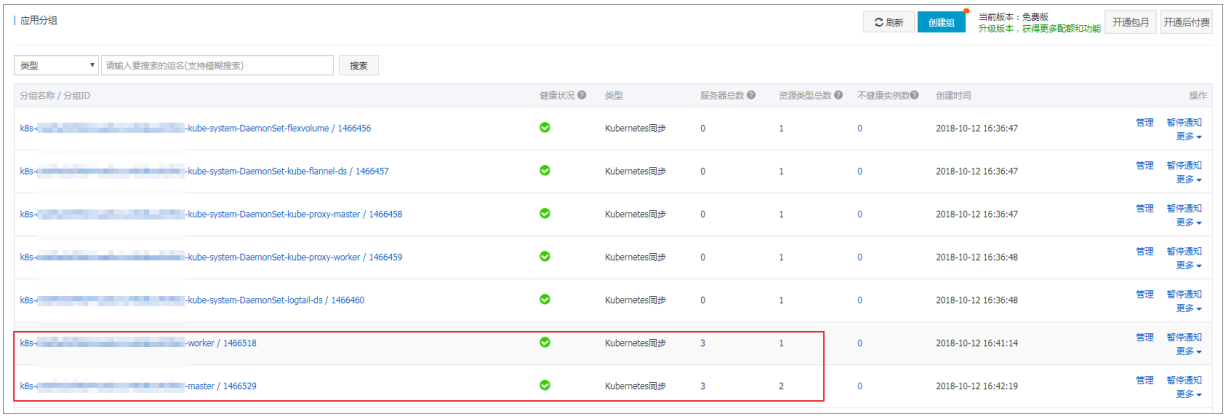
Kubernetes采用静态资源调度方式，对于每个节点上的剩余资源，它是这样计算的：节点剩余资源=节点总资源-已经分配出去的资源，并不是实际使用的资源。如果您自己手动运行一个很耗资源的程序，Kubernetes并不能感知到。

另外所有Pod上都要声明resources。对于没有声明resources的Pod，它被调度到某个节点后，Kubernetes也不会在对应节点上扣掉这个Pod使用的资源。可能会导致节点上调度过去太多的Pod

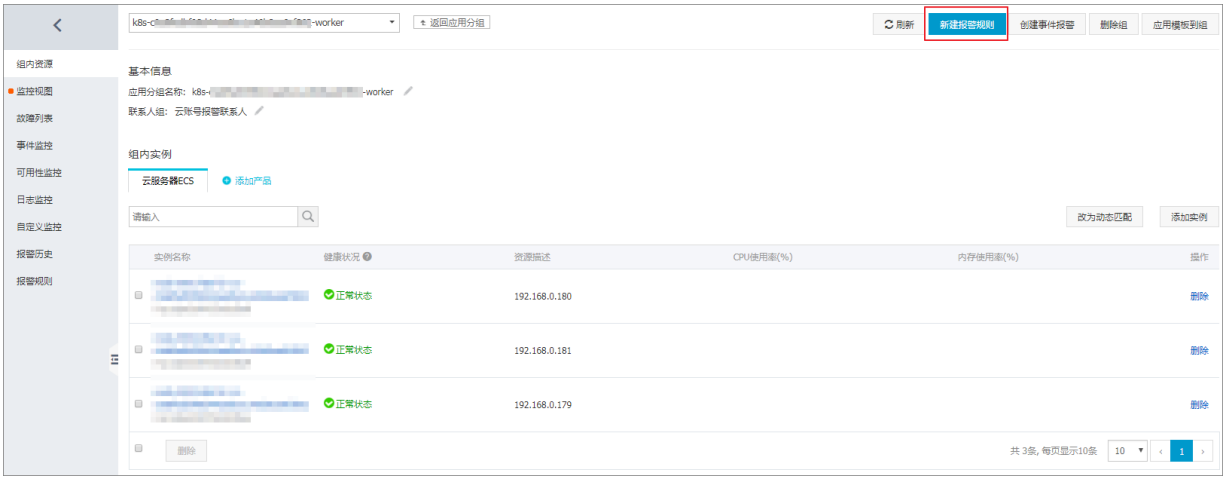
配置监控

阿里云容器服务与云监控进行集成，通过配置节点监控，提供实时的监控服务。通过添加监控告警规则，节点上的资源使用使用量很高的时候，可快速定位问题。

通过容器服务创建Kubernetes集群时，会自动在云监控创建两个应用分组：一个对应Master节点，一个对应Worker节点。我们可以在这两个组下面添加一些报警规则，对组里所有的机器生效。后续加入的节点，也会自动出现在组里，不用单独再去配置报警规则。



主要配置ECS资源的报警规则就可以了。



启动时等待下游服务，不要直接退出

有些应用可能会有一些外部依赖，比如需要从数据库（DB）读取数据或者依赖另外一个服务的接口。应用启动的时候，外部依赖未必都能满足。过去手工运维的时候，通常采用依赖不满足立即退出的方式，也就是所谓的**failfast**，但是在Kubernetes中，这种策略就未必合适了。原因在于Kubernetes中多数运维操作都是自动的，不需要人工介入，比如部署应用，您不用自己选择节

点，再到节点上启动应用，应用fail，也不用手动重启，Kubernetes会自动重启应用。负载增高，还可以通过HPA自动扩容。

针对启动时依赖不满足这个场景，假设有两个应用A和B，A依赖B，刚好运行在同一个节点上。这个节点因为某些原因重启了，重启之后，A首先启动，这个时候B还没启动，对A来说就是依赖不满足。如果A还是按照传统的方式直接退出A，当B启动之后，A也不会再启动，必须人工介入处理才行。

Kubernetes的最好的做法是启动时检查依赖，如果不满足，轮询等待，而不是直接退出。可以通过*Init Container*完成这个功能。

配置restart policy

Pod运行过程中进程退出是个很常见的问题，无论是代码里的一个bug，还是占用内存太多被OOM killer干掉，都会导致应用进程退出，Pod退出。您可在Pod上配置restartPolicy，就能实现Pod挂掉之后自动启动。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    restartPolicy: OnFailure #
```

restartPolicy有三个可选值

- Always： 总是自动重启
- OnFailure：异常退出才自动重启（进程退出状态非0）
- Never：永远不重启

配置Liveness Probe和Readiness Probe

Pod处于Running状态和Pod能正常提供服务是完全不同的概念，一个Running状态的Pod，里面的进程可能发生了死锁而无法提供服务。但是因为Pod还是Running的，Kubernetes也不会自动重启这个Pod。所以我们要在所有Pod上配置Liveness Probe，探测Pod是否真的存活，是否还能提供服务。如果Liveness Probe发现了问题，Kubernetes会重启Pod。

Readiness Probe用于探测Pod是不是可以对外提供服务了。应用启动过程中需要一些时间完成初始化，在这个过程中是没法对外提供服务的，通过Readiness Probe，我们可以告诉Ingress或者

Service能不能把流量转发给这个Pod上。当Pod出现问题的时候，Readiness Probe能避免新流量继续转发给这个Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    livenessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
```

每个进程一个容器

很多刚刚接触容器的人喜欢按照旧习惯把容器当作虚拟机（VM）使用，在一个容器里塞入多个进程，监控进程、日志进程、sshd进程、甚至整个Systemd。这种方式有什么问题呢？首先，判断Pod整体的资源占用会变复杂，不方便实施前面提到resource limit。其次，容器内只有一个进程的情况，进程挂了，外面的容器引擎可以清楚的感知到，然后重启容器，如果容器内有多个进程，某个进程挂了，容器未必受影响，外部的容器引擎感知不到容器内进程退出，也不会对容器做任何操作，但是容器实际上已经不能正常工作了。

如果确实有几个进程需要协同工作，在Kubernetes里也很容易实现，举个例子，nginx和php-fpm，通过unix domain socket通信，我们可以用一个包含两个容器的Pod，unix socket放在两个容器的共享volume中。

确保不存在SPOF

如果应用只有一个实例，当实例失败的时候，虽然Kubernetes能够重启实例，但是中间不可避免地存在一段时间的不可用。甚至更新应用，发布一个新版本的时候，也会出现这种情况。在Kubernetes里，尽量避免直接使用Pod，尽可能使用Deployment/StatefulSet，并且让应用的Pod在两个以上。

1.2 授权管理

1.2.1 角色授权

在用户开通容器服务时，需要授予名称为 AliyunCSDefaultRole 和 AliyunCSClusterRole 的系统默认角色给服务账号，当且仅当该角色被正确授予后，容器服务才能正常地调用相关服务（ECS，OSS、NAS、SLB 等），创建集群以及保存日志等。

使用说明

- 如果您是在2018年1月15日之前使用过容器服务的用户，系统将默认完成角色授权，详细的授权权限内容下面的默认角色包含的权限内容。如果您之前是通过子账号使用，您需要对子账号进行策略升级，参见创建自定义授权策略。
- 2018年1月15日全面接入跨服务授权后，新用户使用主账号只有进行了跨服务授权才能使用容器服务产品。如果新用户需要授权子账号使用容器服务，需要自行前往RAM控制台进行授权，详细操作参见使用子账号。

角色授权步骤

1. 当您进入容器服务控制台，如果之前没有正确地给服务账号授予默认角色，则会看到如下提示。单击 同意授权。

> 说明：
> 容器服务已经设置好默认的角色权限，如需修改角色权限，请前往 RAM 控制台角色管理中设置，需要注意的是，错误的配置可能导致容器服务无法获取到必要的权限。

2. 完成以上授权后，刷新容器服务控制台，然后就可以进行操作了。

   如果您想查看 AliyunCSDefaultRole 和 AliyunCSClusterRole 角色的详细策略信息，可以登录 *RAM* 的控制台 进行查看。

默认角色包含的权限内容

关于各个角色权限的详细信息，请参考各个产品的 API 文档。

AliyunCSDefaultRole 角色的权限内容

默认角色 AliyunCSDefaultRole 包含的主要权限信息如下：

- ECS 相关权限

权限名称（**Action**）	权限说明
ecs:RunInstances	查询实例信息
ecs:RenewInstance	ECS 实例续费
ecs:Create*	创建 ECS 相关资源，如实例、磁盘等
ecs:AllocatePublicIpAddress	分配公网 IP 地址
ecs:AllocateEipAddress	分配 EIP 地址
ecs:Delete*	删除机器实例
ecs:StartInstance	启动 ECS 相关资源
ecs:StopInstance	停止机器实例
ecs:RebootInstance	重启机器实例
ecs:Describe*	查询 ECS 相关资源
ecs:AuthorizeSecurityGroup	设置安全组入规则
ecs:RevokeSecurityGroup	撤销安全组规则
ecs:AuthorizeSecurityGroupEgress	设置安全组出规则
ecs:AttachDisk	添加磁盘
ecs:DetachDisk	清理磁盘
ecs:AddTags	添加标签
ecs:ReplaceSystemDisk	更换 ECS 实例的系统盘
ecs:ModifyInstanceAttribute	修改实例属性
ecs:JoinSecurityGroup	将实例加入到指定的安全组
ecs:LeaveSecurityGroup	将实例移出指定的安全组
ecs:UnassociateEipAddress	解绑弹性公网 IP
ecs:ReleaseEipAddress	释放弹性公网 IP

- VPC 相关权限

权限名称（**Action**）	权限说明
vpc:Describe*	查询 VPC 相关资源的信息
vpc:DescribeVpcs	查询 VPC 信息
vpc:AllocateEipAddress	分配 EIP 地址
vpc:AssociateEipAddress	关联 EIP 地址
vpc:UnassociateEipAddress	不关联 EIP 地址
vpc:ReleaseEipAddress	释放弹性公网 IP
vpc:CreateRouteEntry	创建路由接口
vpc:DeleteRouteEntry	删除路由接口

- SLB 相关权限

权限名称（**Action**）	权限说明
slb:Describe*	查询负载均衡相关信息
slb:CreateLoadBalancer	创建负载均衡实例
slb:DeleteLoadBalancer	删除负载均衡实例
slb:RemoveBackendServers	解绑负载均衡实例
slb:StartLoadBalancerListener	启动指定的监听服务
slb:StopLoadBalancerListener	停止指定的监听服务
slb:CreateLoadBalancerTCPListener	为负载均衡实例创建基于 TCP 协议的监听规则
slb:AddBackendServers	添加后端服务器

AliyunCSClusterRole 角色的权限内容

默认角色 AliyunCSClusterRole 包含的主要权限信息如下：

- OSS 相关权限

权限名称（**Action**）	权限说明
oss:PutObject	上传文件或文件夹对象
oss:GetObject	获取文件或文件夹对象

权限名称（**Action**）	权限说明
oss:ListObjects	查询文件列表信息

- NAS 相关权限

权限名称（**Action**）	权限说明
nas:Describe*	返回 NAS 相关信息
nas:CreateAccessRule	创建权限规则

- SLB 相关权限

权限名称（**Action**）	权限说明
slb:Describe*	查询负载均衡相关信息
slb:CreateLoadBalancer	创建负载均衡实例
slb:DeleteLoadBalancer	删除负载均衡实例
slb:RemoveBackendServers	解绑负载均衡实例
slb:StartLoadBalancerListener	启动指定的监听服务
slb:StopLoadBalancerListener	停止指定的监听服务
slb:CreateLoadBalancerTCPListener	为负载均衡实例创建基于 TCP 协议的监听规则
slb:AddBackendServers	添加后端服务器
slb:DeleteLoadBalancerListener	删除负载均衡实例监听规则
slb:CreateVServerGroup	创建虚拟服务器组，并添加后端服务器
slb:ModifyVServerGroupBackendServers	改变虚拟服务器组中的后端服务器
slb:CreateLoadBalancerHTTPListener	为负载均衡实例创建基于 HTTP 协议的 Listener
slb:SetBackendServers	配置后端服务器，为负载均衡实例后端的一组服务器（ECS 实例）配置权重值
slb:AddTags	为 SLB 实例添加标签

1.2.2 使用子账号

您可通过子账号使用容器服务。

使用子账号登录容器服务控制台并进行相关操作之前，您需要赋予子账号相应的权限。

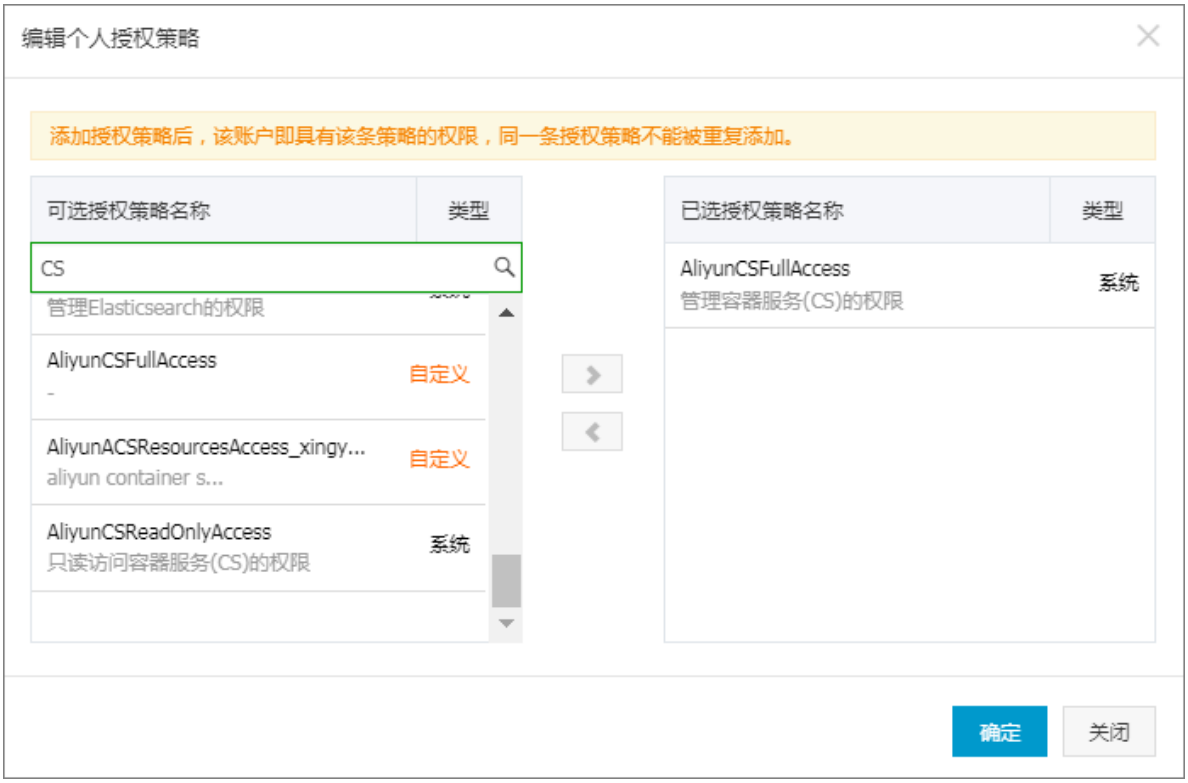
步骤 1 创建子账号并开启控制台登录

1. 登录 访问控制管理控制台。
2. 单击左侧导航栏中的用户管理并单击页面右上角的新建用户。
3. 填写子账号的名称并单击确定。
4. 在用户管理页面，选择创建的子账号，单击右侧的管理。
5. 在Web控制台登录管理中，单击启用控制台登录。
6. 输入登录密码并单击确定。

步骤 2 授予子账号访问容器服务的权限

1. 在用户管理页面，选择创建的子账号，单击右侧的授权。

2. 将所需的策略授权给子账号。



您可以使用系统默认的授权策略。

- AliyunCSFullAccess：容器服务的管理权限。

- AliyunCSReadOnlyAccess：容器服务的只读权限。

或者根据您自己的需要自定义授权策略并授权给子账号，参见创建自定义授权策略。

步骤 3 子账号登录容器服务控制台

- 如果您之前已经给主账号授予了 AliyunCSDefaultRole 和 AliyunCSClusterRole 角色，子账号可以直接登录到容器服务管理控制台，并进行相应的操作。

  使用子账号登录 容器服务管理控制台。
- 如果之前您没有给主账号授予 AliyunCSDefaultRole 和 AliyunCSClusterRole 角色，则需要使用主账号登录容器服务管理控制台。

  进入角色授权页面，单击**同意授权**，授予主账号如下权限。

完成以上授权后，然后使用子账号登录容器服务控制台，进行后续操作。

1.2.3 创建自定义授权策略

容器服务提供的系统授权策略的授权粒度比较粗，如果这种粗粒度授权策略不能满足您的需要，那么您可以创建自定义授权策略。比如，您想控制对某个具体的集群的操作权限，您必须使用自定义授权策略才能满足这种细粒度要求。

创建自定义授权策略

在创建自定义授权策略时，您需要了解授权策略语言的基本结构和语法，相关内容的详细描述请参考授权策略语言描述。

本文档以授予子账号查询、扩容和删除集群的权限为例进行说明。

操作步骤

1. 使用主账号登录*RAM*管理控制台。
2. 单击左侧导航栏中的策略管理并单击页面右上角的新建授权策略。
3. 选择一个模板，填写授权策略名称并编写您的授权策略内容。


```
{
 "Statement": [{
     "Action": [
         "cs:Get*",
         "cs:ScaleCluster",
         "cs:DeleteCluster"
     ],
     "Effect": "Allow",
     "Resource": [
         "acs:cs:*:*:cluster/集群ID"
     ]
 }],
 "Version": "1"
}
```

其中：

- `Action` 处填写您所要授予的权限。

> **说明：**

所有的 Action 均支持通配符。

- `Resource` 有如下配置方式。
  - 授予单集群权限

```
"Resource": [
     "acs:cs:*:*:cluster/集群ID"
 ]
```

  - 授予多个集群权限

```
"Resource": [
     "acs:cs:*:*:cluster/集群ID",
     "acs:cs:*:*:cluster/集群ID"
 ]
```

  - 授予您所有集群的权限

```
"Resource": [
     "*"
 ]
```

  其中，集群`ID` 需要替换为您要授权的真实的集群 ID。

4. 编写完毕后，单击新建授权策略。

表 1-1: 容器服务RAM Action

Action	说明
CreateCluster	创建集群
AttachInstances	向集群中添加已有ECS实例
ScaleCluster	扩容集群
GetClusters	查看集群列表
GetClusterById	查看集群详情
ModifyClusterName	修改集群名称
DeleteCluster	删除集群
UpgradeClusterAgent	升级集群Agent
GetClusterLogs	查看集群的操作日志
GetClusterEndpoint	查看集群接入点地址
GetClusterCerts	下载集群证书

Action	说明
RevokeClusterCerts	吊销集群证书
BindSLB	为集群绑定负载均衡实例
UnBindSLB	为集群解绑负载均衡实例
ReBindSecurityGroup	为集群重新绑定安全组
CheckSecurityGroup	检测集群现有的安全组规则
FixSecurityGroup	修复集群的安全组规则
ResetClusterNode	重置集群中的节点
DeleteClusterNode	移除集群中的节点
CreateAutoScale	创建节点弹性伸缩规则
UpdateAutoScale	更新节点弹性伸缩规则
DeleteAutoScale	删除节点弹性伸缩规则
GetClusterProjects	查看集群下的应用
CreateTriggerHook	为应用创建触发器
GetTriggerHook	查看应用的触发器列表
RevokeTriggerHook	删除应用的触发器
CreateClusterToken	创建 Token

1.2.4 子账号Kubernetes应用权限配置指导

本文旨在帮助您了解如何通过容器服务控制台配置子账号对应的Kubernetes RAM集群权限和在集群内相应的Kubernetes RBAC应用权限。

配置说明

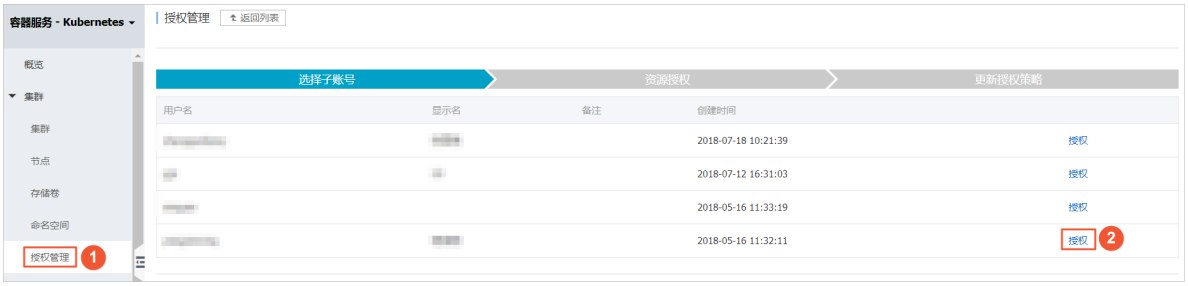
- 子账号授权页面仅主账号可见。您需要拥有一个阿里云主账号，并有一个或若干个子账号。
- 由于阿里云RAM的安全限制，当您通过容器服务控制台的授权配置涉及到子账号RAM授权的修改时，需要您按照页面上给出的参考策略内容和操作说明，在RAM控制台进行目标子账号的手动授权。

操作步骤

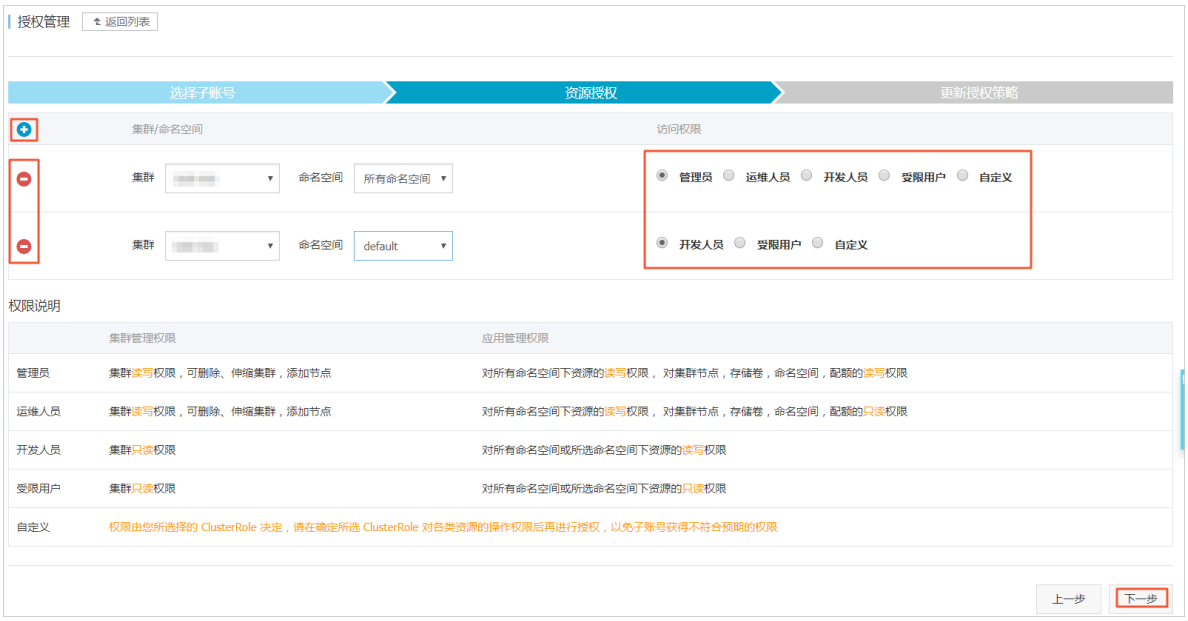
说明：

因为子账号授权页面仅主账号可见，您需要用主账号登录容器服务控制台。

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的集群 > 授权管理，进入授权管理页面。
3. 在子账号列表中选择需要授权的子账号，单击授权。



4. 进入资源授权页面，您可以通过单击表格左上角的加号添加集群或命名空间级别的权限配置，并选择相应的预置角色；也可以单击配置行首的减号删除目标角色。



集群和命名空间的预置角色定义可查看下面的角色权限说明：

表 1-2: 角色权限说明

	集群管理权限	应用管理权限
管理员	集群读写权限，可删除、伸缩集群，添加节点	对所有命名空间下资源的读写权限，对集群节点，存储卷，命名空间，配额的读写权限

	集群管理权限	应用管理权限
运维人员	集群读写权限，可删除、伸缩集群，添加节点	对所有命名空间下资源的读写权限，对集群节点，存储卷，命名空间，配额的只读权限
开发人员	集群只读权限	对所有命名空间或所选命名空间下资源的读写权限
受限用户	集群只读权限	对所有命名空间或所选命名空间下资源的只读权限
自定义	集群读写权限，可删除、伸缩集群，添加节点	权限由您所选择的 ClusterRole 决定，请在确定所选 ClusterRole 对各类资源的操作权限后再进行授权，以免子账号获得不符合预期的权限

5. 完成配置后，如果涉及目标子账号RAM集群权限的变更，在更新授权策略页面会展示配置项对应的Kubernetes集群RAM权限参考配置，您可以根据页面中的指导在RAM控制台完成子账号授权的更新。

补充说明

为了不影响子账号对当前存量可访问kubernetes集群的正常使用，容器服务控制台会暂时兼容旧的集群访问权限控制，在一段时间对原有子账号可访问kubernetes集群不进行RBAC的应用权限校验，如果您是子账号用户，请您根据集群类型和兼容方式，及时联系主账号进行授权操作。

对于由当前子账号自身创建的存量集群，可以通过在集群详情页面单击**升级当前集群授权信息**，完成集群应用权限的自动升级。

在公告期结束后，仍旧没有由主账号授权或进行权限管理升级的子账号，子账号将被禁止访问集群对应的应用控制台。

表 1-3: 兼容集群说明

兼容集群类型	兼容方式
存量子账号创建集群	提示权限管理升级公告，提供一键升级链接，子账号可通过点击升级链接完成应用授权
存量RAM授权访问集群	提示权限管理升级公告，请及时联系主账号完成应用授权

兼容集群类型	兼容方式
新建RAM授权访问集群	提示权限管理升级公告，请及时联系主账号完成应用授权

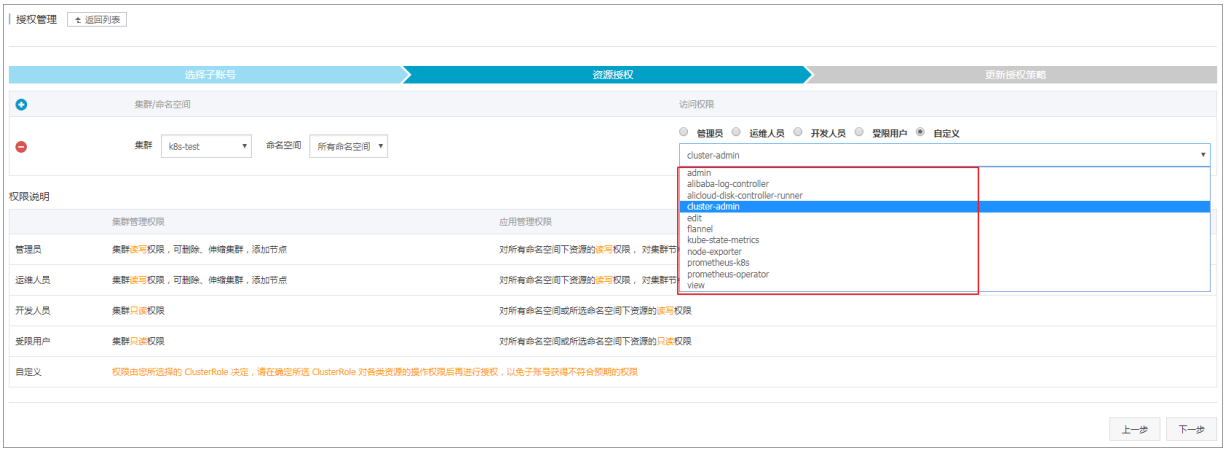
自定义权限说明

阿里云容器服务预置了管理员、运维人员、开发人员和受限人员4种标准的访问权限，可满足大部分用户在容器服务控制台上的使用需求。如果您想自由定义集群的访问权限，可使用自定义权限功能。

阿里云容器服务内置了一些自定义权限。

说明：

其中cluster-admin权限值得关注，属于集群超级管理员权限，对所有资源都默认拥有权限。



您可登录到集群Master节点，执行以下命令，查看自定义权限的详情。

说明：

只有部分clusterrole会在自定义下拉框显示。

```
# kubectl get clusterrole
NAME
  AGE
admin
  13d
alibaba-log-controller
  13d
alicloud-disk-controller-runner
  13d
cluster-admin
  13d
cs:admin
  13d
```

```
edit
  13d
flannel
  13d
kube-state-metrics
  22h
node-exporter
  22h
prometheus-k8s
  22h
prometheus-operator
  22h
system:aggregate-to-admin
  13d
....
system:volume-scheduler
  13d
view
  13d
```

以超级管理员cluster-admin为例，执行以下命令，查看其权限详情。

说明：

子账号被授予该集群角色后，在该集群内，可视为与主账号有相同权限的超级账号，拥有操作集群内所有资源的任意权限，建议谨慎授予。

```
# kubectl get clusterrole cluster-admin  -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: 2018-10-12T08:31:15Z
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: cluster-admin
  resourceVersion: "57"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterroles/cluster-
admin
  uid: 2f29f9c5-cdf9-11e8-84bf-00163e0b2f97
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

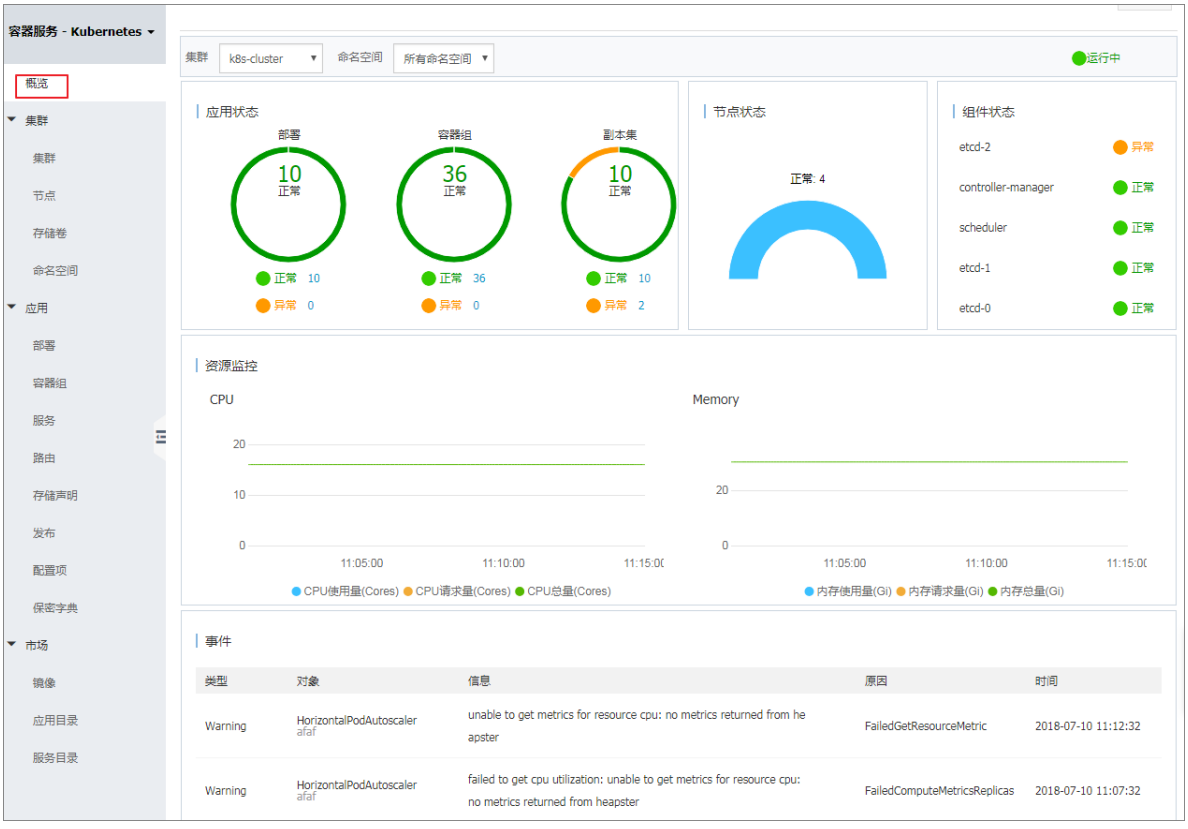
1.3 集群管理

1.3.1 查看集群概览

阿里云容器服务 Kubernetes 集群提供集群概览功能，提供应用状态、组件状态和资源监控等功能，方便您快速了解集群的健康状态信息。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的概览，进入 Kubernetes 集群概览页面。
3. 选择所需的集群和命名空间，您可查看应用状态、组件状态和资源监控图表。
   - 应用状态：显示当前运行的部署、容器组和副本集的状态示意图，绿色图标代表正常，黄色图标代表异常。
   - 节点状态：显示当前集群的节点状态。
   - 组件状态：Kubernetes 集群的组件通常部署在 kube-system 命名空间下，包括 scheduler、controller-manager和 etcd 等核心组件。
   - 资源监控：提供 CPU 和内存的监控图表。CPU 统计单位为 Cores（核），可显示小数点后 3 位，最小统计单位是 millcores，即一个核的 1/1000；内存的统计单位是 Gi，显示小数点后 3 位。更多相关信息，请参见 *Meaning of CPU*和 *Meaning of memory*。
   - 事件：显示集群的事件信息，例如警告和错误事件等。



1.3.2 创建Kubernetes集群

您可以通过容器服务管理控制台非常方便地快速创建 Kubernetes 集群。

使用须知

创建集群过程中，容器服务会进行如下操作：

- 创建 ECS，配置管理节点到其他节点的 SSH 的公钥登录，通过 CloudInit 安装配置 Kubernetes 集群。
- 创建安全组，该安全组允许 VPC 入方向全部 ICMP 端口的访问。
- 如果您不使用已有的 VPC 网络，会为您创建一个新的 VPC 及 VSwitch，同时为该 VSwitch 创建 SNAT。
- 创建 VPC 路由规则。
- 创建 NAT 网关和共享带宽包（或EIP）。
- 创建 RAM 子账号和 AK，该子账号拥有 ECS 的查询、实例创建和删除的权限，添加和删除云盘的权限，SLB 的全部权限，云监控的全部权限，VPC 的全部权限，日志服务的全部权限，NAS 的全部权限。Kubernetes 集群会根据用户部署的配置相应的动态创建 SLB，云盘，VPC路由规则。
- 创建内网 SLB，暴露 6443 端口。
- 创建公网 SLB，暴露 6443端口（如果您在创建集群的时候选择开放公网 SSH 登录，则会暴露 22 端口；如果您选择不开放公网 SSH 访问，则不会暴露 22 端口）。

前提条件

您需要开通容器服务、资源编排（ROS）服务和访问控制（RAM）服务。

登录 容器服务管理控制台、*ROS* 管理控制台 和 *RAM* 管理控制台 开通相应的服务。

说明：

容器服务 Kubernetes 集群部署依赖阿里云资源编排 ROS 的应用部署能力，所以创建 Kubernetes 集群前，您需要开通 ROS。

使用限制

- 随集群一同创建的负载均衡实例只支持按量付费的方式。
- Kubernetes 集群仅支持专有网络 VPC。

- 每个账号默认可以创建的云资源有一定的配额，如果超过配额创建集群会失败。请在创建集群前确认您的配额。如果您需要提高您的配额，请提交工单申请。
  - 每个账号默认最多可以创建 5 个集群（所有地域下），每个集群中最多可以添加 40 个节点。如果您需要创建更多的集群或者节点，请提交工单申请。

    说明：
    Kubernetes集群中，VPC默认路由条目不超过48条，意味着Kubernetes集群使用VPC时，默认节点上限是48个，如果需要更大的节点数，需要您先对目标VPC开工单，提高VPC路由条目，再对容器服务提交工单。
  - 每个账号默认最多可以创建 100 个安全组。
  - 每个账号默认最多可以创建 60 个按量付费的负载均衡实例。
  - 每个账号默认最多可以创建 20 个EIP。
- ECS 实例使用限制：
  - 仅支持 CentOS 操作系统。
  - 支持创建按量付费和包年包月的ECS实例。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群，进入集群列表页面。
3. 单击页面右上角的创建 **Kubernetes** 集群，进入创建 **Kubernetes** 集群页面。

默认进入**kubernetes**集群配置页面。

4. 填写集群的名称。

集群名称应包含1-63个字符，可包含数字、汉字、英文字符或连字符（-）。

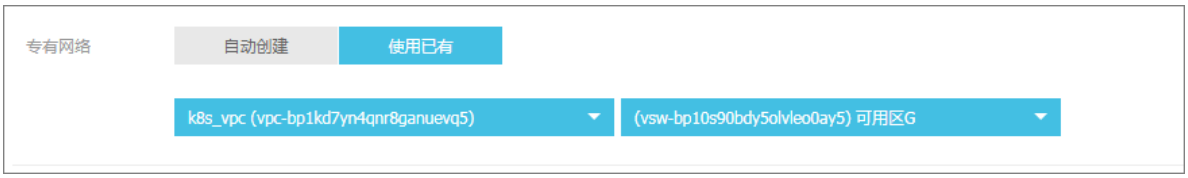
5. 选择集群所在的地域和可用区。

6. 设置集群的网络。Kubernetes 集群仅支持专有网络。

专有网络：您可以选择**自动创建**（创建 Kubernetes 集群时，同步创建一个 VPC）或者**使用已有**（使用一个已有的 VPC）。选择**使用已有**后，您可以在已有 VPC 列表中选择所需的 VPC 和交换机。

- 选择**自动创建**，创建集群时，系统会自动为您的 VPC 创建一个 NAT 网关。
- 选择**使用已有**，如果您使用的 VPC 中当前已有 NAT 网关，容器服务会使用已有的 NAT 网关；如果 VPC 中没有 NAT 网关，系统会默认自动为您创建一个 NAT 网关。如果您不希望系统自动创建 NAT 网关，可以取消勾选页面下方的**为专有网络配置 SNAT**。

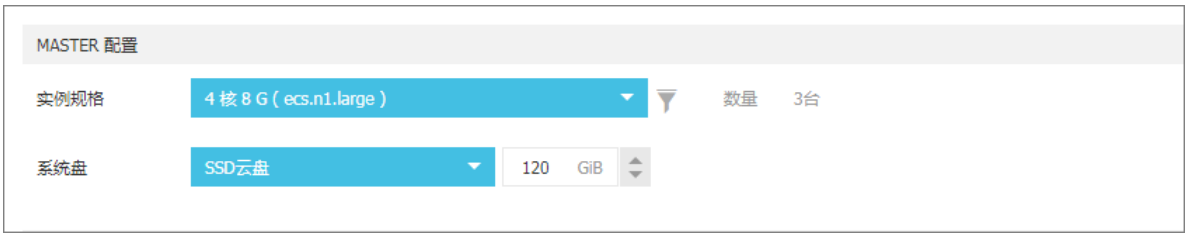
> **说明：**
> 若选择不自动创建 NAT 网关，您需要自行配置 NAT 网关实现 VPC 安全访问公网环境，或者手动配置 SNAT，否则 VPC 内实例将不能正常访问公网，会导致集群创建失败。



7. 设置节点类型，容器服务支持按量付费和包年包月两种节点类型。
8. 设置 Master 节点的配置信息。

您需要选择 Master 节点的实例规格。

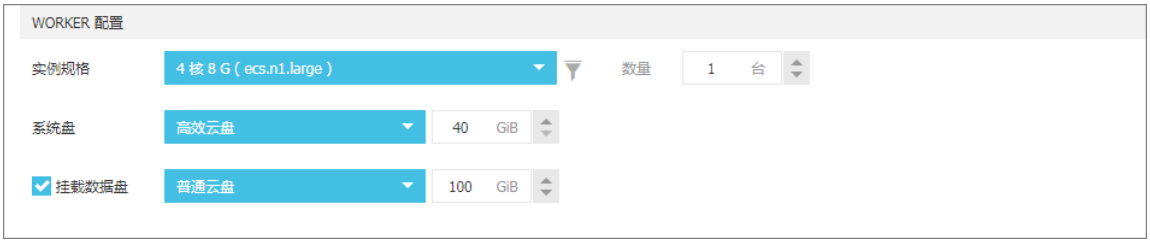
> **说明：**
> - 目前仅支持 CentOS 操作系统。
> - 目前仅支持创建 3 个 Master 节点。
> - 默认为Master节点挂载系统盘，支持SSD云盘和高效云盘。



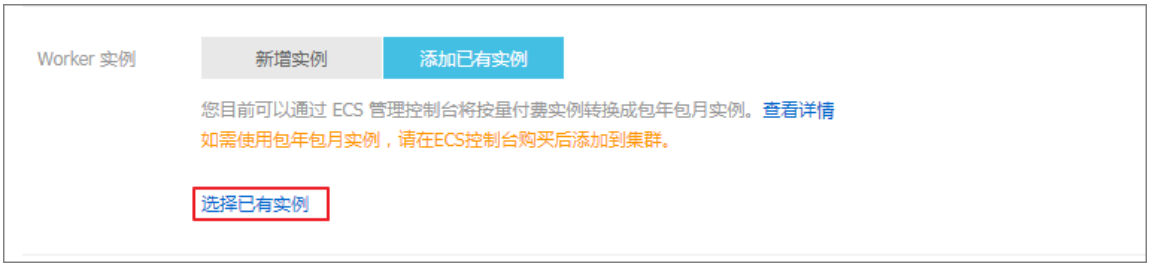
9. 设置 Worker 节点的配置信息。您可选择新增实例或添加已有实例。

> 说明：
>
> - 目前仅支持 CentOS 操作系统。
> - 每个集群最多可包含 37 个 Worker 节点。如果您需要创建更多的节点，请提交工单申请。
> - 默认为Worker节点挂载系统盘，支持SSD云盘和高效云盘。
> - 支持为Worker节点挂载一个数据盘，支持SSD云盘、高效云盘和普通云盘。

a. 若您选择新增实例，则需要选择 Worker 节点的实例规格，以及需要创建的 Worker 节点的数量（本示例创建 1 个 Worker 节点）。



b. 若您选择添加已有实例，则需要预先在此地域下创建 ECS 云服务器。



10. 配置登录方式。

- 设置密钥。

  您需要在创建集群的时候选择密钥对登录方式，单击新建密钥对，跳转到ECS云服务器控制台，创建密钥对，参见创建 SSH 密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。

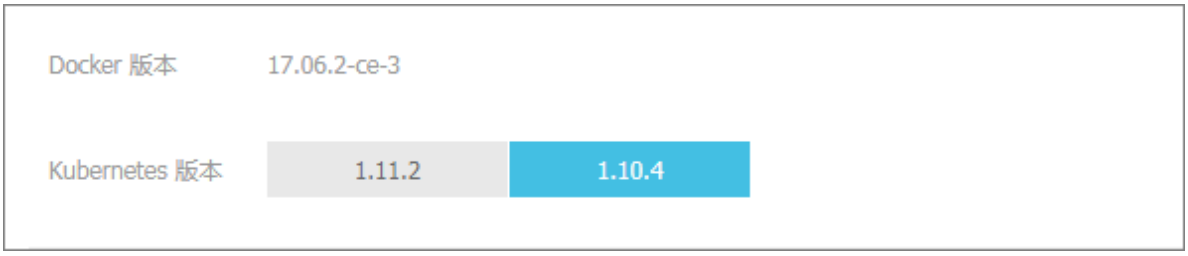
- 设置密码。
  - **登录密码**：设置节点的登录密码。
  - **确认密码**：确认设置的节点登录密码。

11. 设置**Pod网络 CIDR** 和**Service CIDR**。

> 说明：
> 该选项仅在选择使用已有VPC时出现。

您需要指定**Pod 网络 CIDR**和**Service CIDR**，两者都不能与 VPC 及 VPC 内已有 Kubernetes 集群使用的网段重复，创建成功后不能修改。而且 Service 地址段也不能和 Pod 地址段重复，有关 kubernetes 网络地址段规划的信息，请参考*VPC下 Kubernetes 的网络地址段规划*。

12. 显示当前支持的Docker版本和Kubernetes版本，您可查看对应版本，并根据需要进行选择。



13. 设置是否为专有网络配置 SNAT 网关。

> 说明：
> 若您选择自动创建 VPC 时必须配置 SNAT；若您选择使用已有VPC，可选择是否自动配置SNAT网关。若选择不自动配置 SNAT，您可自行配置NAT 网关实现 VPC 安全访问公网环境；或者手动配置 SNAT，否则 VPC 内实例将不能正常访问公网，会导致集群创建失败。

14.设置是否开放用公网**SLB**暴露**API SERVER**。

API Server提供了各类资源对象（Pod，Service等）的增删改查及watch等HTTP Rest接口。

1. 如果选择开放，会创建一个公网SLB，同时把Master节点的6443端口（对应API Server）暴露出来，用户可以在外网通过kubeconfig连接/操作集群。
2. 若选择不开放，不会创建公网SLB，用户只能在VPC内部用kubeconfig连接/操作集群。

15.设置是否开放公网 SSH 登录。

> 说明：
> 您需要开放公网**SLB**暴露**API SERVER**，才能设置公网SSH登录。

- 选择开放公网 SSH 登录，您可以 SSH 访问集群。
- 选择不开放公网 SSH 登录，将无法通过 SSH 访问集群，也无法通过 kubectl 连接 集群。如果您需要通过 SSH 访问集群实例，可以手动为 ECS 实例绑定 EIP，并配置安全组规则，开放 SSH（22）端口，具体操作参见*SSH*访问*Kubernetes*集群。

16.设置是否启用云监控插件。

您可以选择在 ECS 节点上安装云监控插件，从而在云监控控制台查看所创建 ECS 实例的监控信息。

17.设置是否启用日志服务，您可使用已有Project或新建一个Project。

勾选使用**SLS**，会在集群中自动配置日志服务插件。创建应用时，您可通过简单配置，快速使用日志服务，详情参见使用日志服务进行*Kubernetes*日志采集。

18. 是否启用高级选项。

a. 设置启用的网络插件，支持Flannel和Terway网络插件，具体可参考如何选择*Kubernetes*集群网络插件*#Terway*和*Flannel*。

- Flannel：简单稳定的社区的Flannel CNI插件。但功能偏简单，支持的特性少，例如：不支持基于Kubernetes标准的Network Policy。
- Terway：阿里云容器服务自研的网络插件，支持将阿里云的弹性网卡分配给容器，支持Kubernetes的**`Network Policy`**来定义容器间的访问策略，支持对单个容器做带宽的限流。

b. 设置节点 Pod 数量，是指单个节点可运行 Pod 数量的上限，建议保持默认值。

c. 设置是否使用自定义集群**CA**。如果勾选自定义集群 CA，可以将 CA 证书添加到 kubernetes 集群中，加强服务端和客户端之间信息交互的安全性。

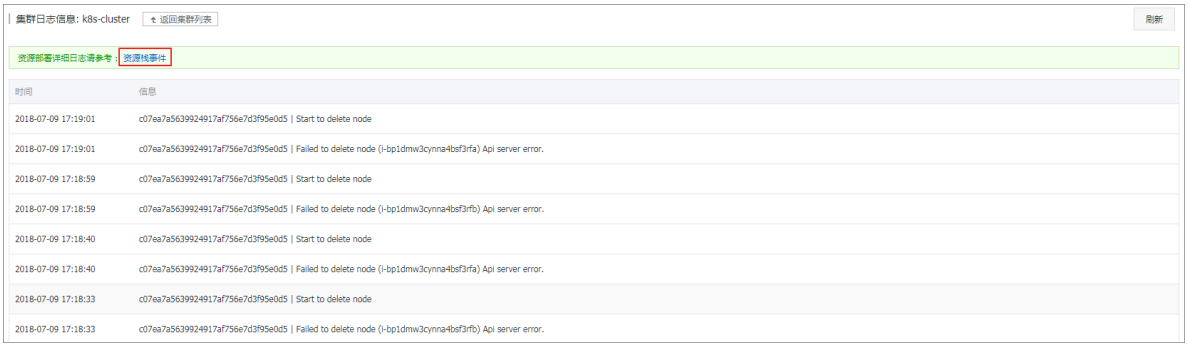
19. 单击**创建集群**，启动部署。

> **说明：**
> 一个包含多节点的 Kubernetes 集群的创建时间一般需要十几分钟。

查看集群部署结果

集群创建成功后，您可以在容器服务管理控制台的 Kubernetes 集群列表页面查看所创建的集群。

- 您可以单击操作列的**查看日志**，进入集群日志信息页面查看集群的日志信息。 您也可以在集群日志信息页面中，单击**资源栈事件**查看更详细的信息。



- 在集群列表中，找到刚创建的集群，单击操作列中的管理，查看集群的基本信息和连接信息。

其中：

- **API Server** 公网连接端点：Kubernetes 的 API server 对公网提供服务的地址和端口，可以通过此服务在用户终端使用 kubectl 等工具管理集群。

- **API Server 内网连接端点**：Kubernetes 的 API server 对集群内部提供服务的地址和端口。此 IP 为负载均衡的地址，后端有 3 台 Master 提供服务。
- **Master 节点 SSH 连接地址**：可以直接通过 SSH 登录到 Master 节点，以便对集群进行日常维护。
- **服务访问域名**：为集群中的服务提供测试用的访问域名。服务访问域名后缀是`<cluster_id>.<region_id>.alicontainer.com`。

例如，您可以通过 SSH 登录到 Master 节点，执行`kubectl get node`查看集群的节点信息。

```
login as: root
root@i[redacted]3's password:

Welcome to Alibaba Cloud Elastic Compute Service !

[root@iZbp1d7yvpa3j183u0urllZ ~]# kubectl get node
NAME                                STATUS    ROLES     AGE       VERSION
cn-hangzhou.i-[redacted]            Ready     <none>    17m       v1.8.4
cn-hangzhou.i-[redacted]            Ready     master    19m       v1.8.4
cn-hangzhou.i-[redacted]            Ready     master    24m       v1.8.4
cn-hangzhou.i-[redacted]            Ready     master    22m       v1.8.4
[root@iZbp1d7yvpa3j183u0urllZ ~]#
```

可以发现，一共有 4 个节点，包括 3 个 Master 节点和我们在参数设置步骤填写的 1 个 Worker 节点。

1.3.3 创建Kubernetes 托管版集群

您可以通过容器服务控制台非常方便的创建Kubernetes托管版集群。

前提条件

您需要开通容器服务、资源编排（ROS）服务和访问控制（RAM）服务。

登录 容器服务管理控制台、*ROS* 管理控制台 和 *RAM* 管理控制台 开通相应的服务。

> **说明：**
> 容器服务 Kubernetes 托管版集群部署依赖阿里云资源编排 ROS 的应用部署能力，所以创建 Kubernetes 托管版集群前，您需要开通 ROS。

背景信息

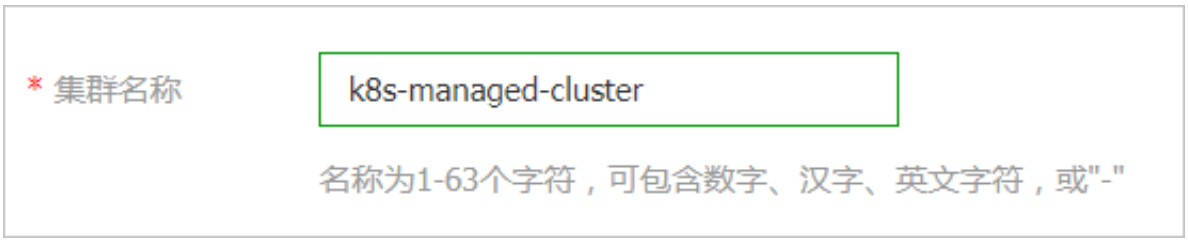
- 用户账户需有 100 元的余额并通过实名认证，否则无法创建按量付费的 ECS 实例和负载均衡。
- 随集群一同创建的负载均衡实例只支持按量付费的方式。
- Kubernetes 集群仅支持专有网络VPC。

- 每个账号默认可以创建的云资源有一定的配额，如果超过配额创建进群会失败。请在创建集群前确认您的配额。如果您需要提高您的配额，请提交工单申请。
  - 每个账号默认最多可以创建 100 个安全组。
  - 每个账号默认最多可以创建 60 个按量付费的负载均衡实例。
  - 每个账号默认最多可以创建 20 个EIP。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下， 单击左侧导航栏中的集群，进入集群列表页面。单击页面右上角的创建 **Kubernetes** 集群。
3. 在创建 **Kubernetes** 集群页面，选择**Kubernetes** 托管版（公测）。
4. 填写集群的名称。

   集群名称应包含 1~63 个字符，可包含数字、汉字、英文字符或连字符（-）。

5. 选择集群所在的地域和可用区。

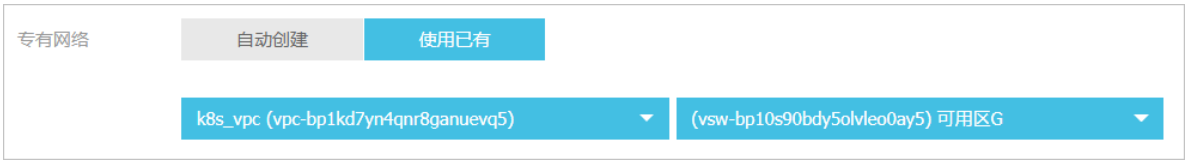
6. 设置集群的网络。

   说明：
   Kubernetes 集群仅支持专有网络。

   专有网络：您可以选择自动创建（创建 Kubernetes 集群时，同步创建一个 VPC）或者使用已有（使用一个已有的 VPC）。选择使用已有后，您可以在已有 VPC 列表中选择所需的 VPC 和交换机。

- 选择**自动创建**，创建集群时，系统会自动为您的 VPC 创建一个 NAT 网关。
- 选择**使用已有**，如果您使用的 VPC 中当前已有 NAT 网关，容器服务会使用已有的 NAT 网关；如果 VPC 中没有 NAT 网关，系统会默认自动为您创建一个 NAT 网关。如果您不希望系统自动创建 NAT 网关，可以取消勾选页面下方的**为专有网络配置 SNAT**。

> **说明：**
> 若选择不自动创建 NAT 网关，您需要自行配置 NAT 网关实现 VPC 安全访问公网环境，或者手动配置 SNAT，否则 VPC 内实例将不能正常访问公网，会导致集群创建失败。

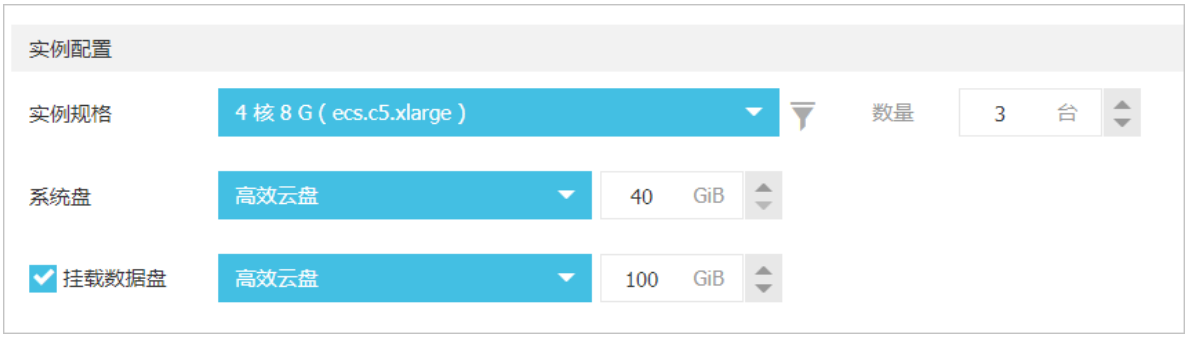


7. 设置节点类型。

> **说明：**
> 支持创建**按量付费**和**包年包月**两种节点类型。

8. 设置实例。

> **说明：**
> - 目前仅支持 CentOS 操作系统。
> - 每个集群最少包含 2 个节点。
> - 每个集群最多可包含 48 个节点。如果您需要创建更多的节点，请提交工单申请。
> - 默认为实例挂载系统盘，支持高效云盘和SSD云盘。
> - 支持为实例挂载 1 个数据盘，支持高效云盘和SSD云盘。



9. 设置登录方式。

- 设置密钥。

  您需要在创建集群的时候选择密钥对登录方式，单击新建密钥对，跳转到ECS云服务器控制台，创建密钥对，参见创建 SSH 密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。

- 设置密码。

  — **登录密码**：设置节点的登录密码。

  — **确认密码**：确认设置的节点登录密码。

10. 设置**Pod 网络 CIDR** 和**Service CIDR**

> 说明：
>
> - 该选项仅在设置专有网络选择使用已有VPC时出现。
> - **Pod 网络 CIDR**和**Service CIDR**两者都不能与 VPC 及 VPC 内已有 Kubernetes 集群使用的网段重复，创建成功后不能修改。且 Service 地址段也不能和 Pod 地址段重复，有关 Kubernetes 网络地址段规划的信息，请参考VPC下 Kubernetes 的网络地址段规划。

11. 设置是否为专有网络配置 SNAT 网关。

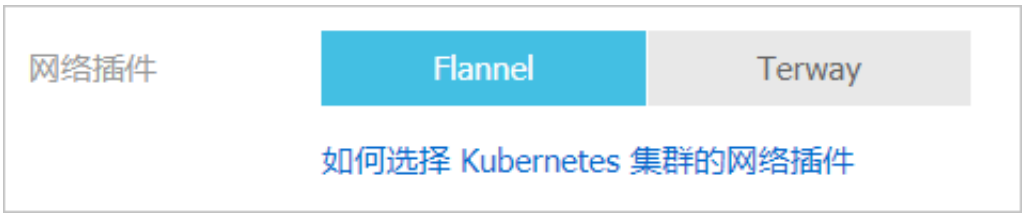
> **说明：**
>
> - 若您选择自动创建 VPC 时必须配置 SNAT；
> - 若您选择使用已有VPC，可选择是否自动配置SNAT网关。若选择不自动配置 SNAT，您可自行配置NAT 网关实现 VPC 安全访问公网环境；或者手动配置 SNAT，否则 VPC 内实例将不能正常访问公网，会导致集群创建失败。

12. 设置是否启用云监控插件。

    您可以选择在 ECS 节点上安装云监控插件，从而在云监控控制台查看所创建 ECS 实例的监控信息。

13. 设置启用的网络插件，支持Flannel和Terway网络插件，具体可参考如何选择*Kubernetes*集群网络插件*#Terway*和*Flannel*。

    - Flannel：简单稳定的社区的Flannel 插件。但功能偏简单，支持的特性少，例如：不支持基于Kubernetes标准的Network Policy。
    - Terway：阿里云容器服务自研的网络插件，支持将阿里云的弹性网卡分配给容器，支持Kubernetes的**`Network Policy`**来定义容器间的访问策略，支持对单个容器做带宽的限流。



14.设置RDS白名单。

将节点 IP 添加到 RDS 实例的白名单。

> **说明：**
> 该选项仅在设置专有网络选择使用已有VPC时出现。

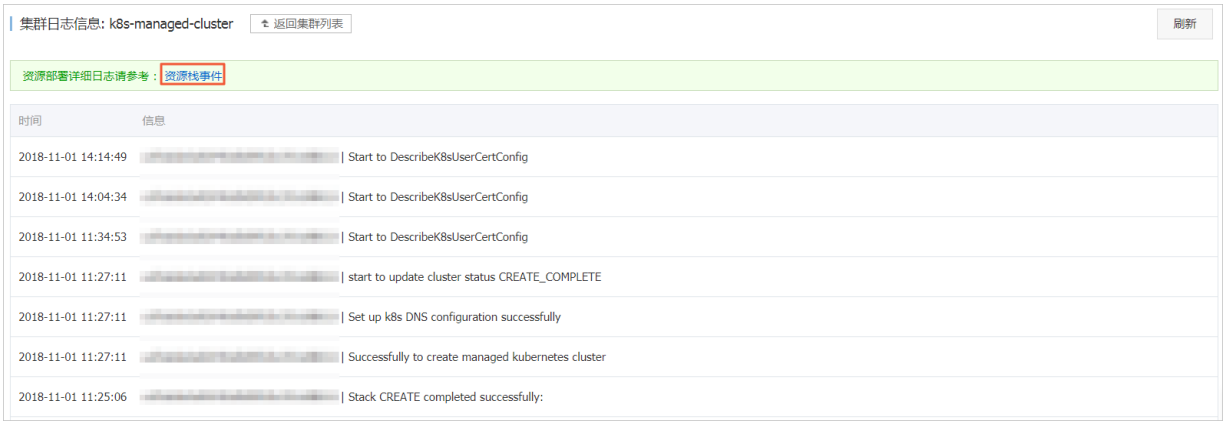
15.单击创建集群，启动部署。

> **说明：**
> 一个包含多节点的 Kubernetes 集群的创建时间一般约为 5 分钟。

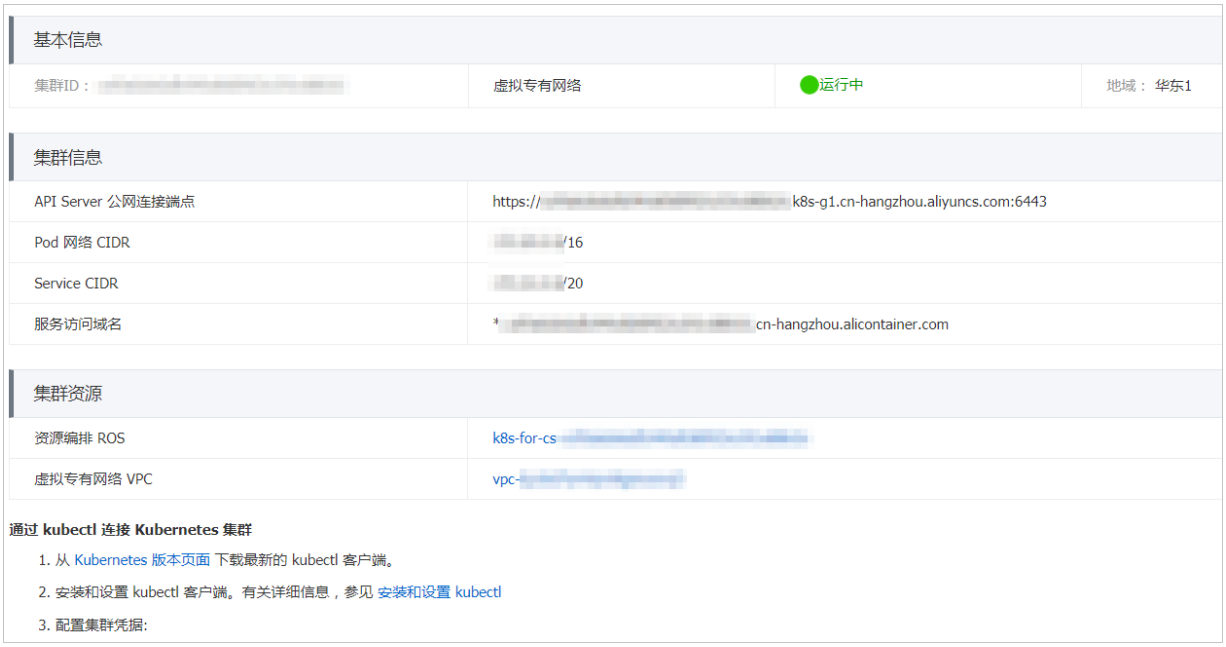
预期结果

集群创建成功后，您可以在容器服务管理控制台的 Kubernetes 集群列表页面查看所创建的集群。

您可以单击操作列的查看日志，进入集群日志信息页面查看集群的日志信息。 您也可以在集群日志信息页面中，单击资源栈事件查看更详细的信息。



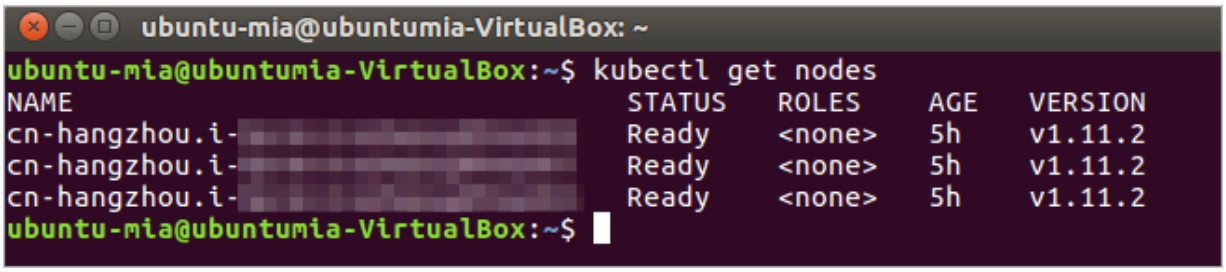
在集群列表页面中，找到刚创建的集群，单击操作列中的管理，查看集群的基本信息和连接信息。



其中：

- **API Server** 公网连接端点：Kubernetes 的 API Server 对公网提供服务的地址和端口，可以通过此服务在用户终端使用 kubectl 等工具管理集群。
- 服务访问域名：为集群中的服务提供测试用的访问域名。服务访问域名后缀是`<cluster_id>.<region_id>.alicontainer.com`。

您可以通过 *kubectl* 连接 *Kubernetes* 集群，执行`kubectl get node`查看集群的节点信息。



1.3.4 Kubernetes GPU集群支持GPU调度

自从1.8版本开始，Kubernetes已经明确表示要通过统一的设备插件方式支持像Nvidia PU，InfiniBand,FPGA等硬件加速设备，而社区的GPU方案将在1.10全面弃用，并在1.11版本彻底从主干代码移除。

若您需要通过阿里云Kubernetes集群+GPU运行机器学习，图像处理等高运算密度等任务，无需安装nvidia driver和CUDA，就能实现一键部署和弹性扩缩容等功能。

背景信息

创建集群过程中，容器服务会进行如下操作：

- 创建 ECS，配置管理节点到其他节点的 SSH 的公钥登录，通过 CloudInit 安装配置 Kubernetes 集群。
- 创建安全组，该安全组允许 VPC 入方向全部 ICMP 端口的访问。
- 如果您不使用已有的 VPC 网络，会为您创建一个新的 VPC 及 VSwitch，同时为该 VSwitch 创建 SNAT。
- 创建 VPC 路由规则。
- 创建 NAT 网关及 EIP。
- 创建 RAM 子账号和 AK，该子账号拥有 ECS 的查询、实例创建和删除的权限，添加和删除云盘的权限，SLB 的全部权限，云监控的全部权限，VPC 的全部权限，日志服务的全部权限，NAS 的全部权限。Kubernetes 集群会根据用户部署的配置相应的动态创建 SLB，云盘，VPC路由规则。
- 创建内网 SLB，暴露 6443 端口。
- 创建公网 SLB，暴露 6443、8443和 22 端口（如果您在创建集群的时候选择开放公网 SSH 登录，则会暴露 22 端口；如果您选择不开放公网 SSH 访问，则不会暴露 22 端口）。

前提条件

您需要开通容器服务、资源编排（ROS）服务和访问控制（RAM）服务。

登录 容器服务管理控制台、ROS 管理控制台 和 RAM 管理控制台 开通相应的服务。

> **说明：**
> 容器服务 Kubernetes 集群部署依赖阿里云资源编排 ROS 的应用部署能力，所以创建 Kubernetes 集群前，您需要开通 ROS。

使用限制

- 用户账户需有 100 元的余额并通过实名认证，否则无法创建按量付费的 ECS 实例和负载均衡。
- 随集群一同创建的负载均衡实例只支持按量付费的方式。
- Kubernetes 集群仅支持专有网络 VPC。
- 每个账号默认可以创建的云资源有一定的配额，如果超过配额创建集群会失败。请在创建集群前确认您的配额。如果您需要提高您的配额，请提交工单申请。
  - 每个账号默认最多可以创建 5 个集群（所有地域下），每个集群中最多可以添加 10 个节点。如果您需要创建更多的集群或者节点，请提交工单申请。
  - 每个账号默认最多可以创建 100 个安全组。
  - 每个账号默认最多可以创建 60 个按量付费的负载均衡实例。
  - 每个账号默认最多可以创建 20 个EIP。
- ECS 实例使用限制：
  - 仅支持 CentOS 操作系统。
  - 仅支持创建按量付费的 ECS 实例。

> **说明：**
> 实例创建后，您可以通过 ECS 管理控制台将按量付费转预付费。

创建GN5型Kubernetes集群

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群，进入集群列表页面。
3. 单击页面右上角的创建 **Kubernetes** 集群，进入创建 **Kubernetes** 集群页面。

默认进入**kubernetes**集群配置页面。

> **说明：**
> 为了创建GPU集群，通常情况下，Worker节点使用GPU类型的ECS。其他集群的参数配置，请参见创建*Kubernetes*集群。

4. 设置 Worker 节点的配置信息。本例中将Worker节点作为GPU工作节点，选择GPU计算型gn5。

   a. 若您选择新增实例，则需要选择 Worker 节点的系列和规格，以及需要创建的 Worker 节点的数量（本示例创建2个GPU节点）。

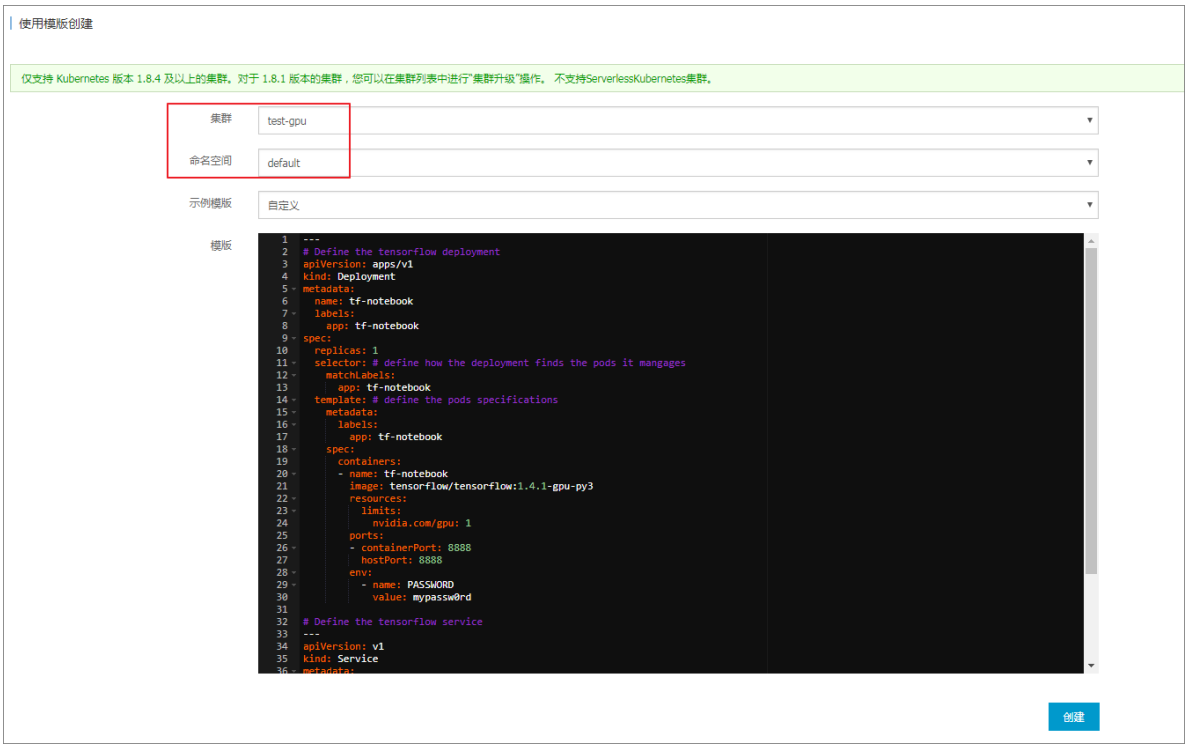
   b. 若您选择添加已有实例，则需要预先在此地域下创建GPU云服务器。

5. 完成其他配置后，单击**创建集群**，启动部署。
6. 集群创建成功后，单击左侧导航栏中的**集群 > 节点**，进入节点列表页面。
7. 选择所需的集群，选择创建集群时配置的Worker节点，单击右侧的**详情**，查看该节点挂载的GPU设备。

运行TensorFLow的GPU实验环境

数据科学家通常习惯使用Jupyter作为TensorFlow实验环境，我们这里可以用一个例子向您展示如何快速部署一个Jupyter应用。

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的**应用 > 部署**，进入部署列表页面。
3. 单击页面右上角的**创建**使用模板创建 。
4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击**创建**。



本例中，示例模板是一个Jupyter应用，包括一个deployment和service。

```
---
# Define the tensorflow deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tf-notebook
  labels:
    app: tf-notebook
spec:
  replicas: 1
  selector: # define how the deployment finds the pods it mangages
    matchLabels:
      app: tf-notebook
  template: # define the pods specifications
    metadata:
      labels:
        app: tf-notebook
    spec:
      containers:
      - name: tf-notebook
        image: tensorflow/tensorflow:1.4.1-gpu-py3
        resources:
          limits:
            nvidia.com/gpu: 1                          #指定调用nvidia
gpu的数量
        ports:
        - containerPort: 8888
          hostPort: 8888
        env:
          - name: PASSWORD                             # 指定访问Jupyter服
务的密码，您可以按照您的需要修改
            value: mypassw0rd
```

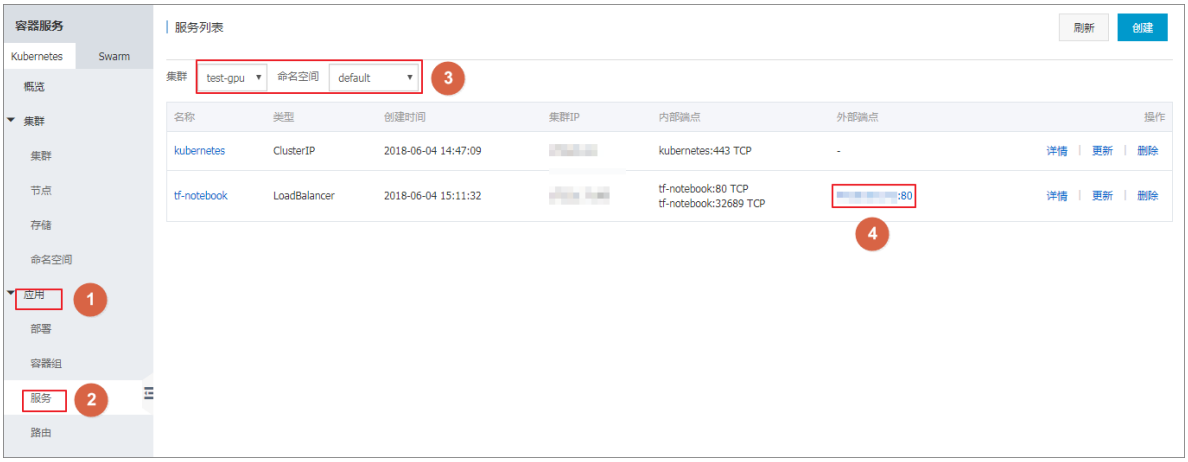
```
# Define the tensorflow service
---
apiVersion: v1
kind: Service
metadata:
  name: tf-notebook
spec:
  ports:
  - port: 80
    targetPort: 8888
    name: jupyter
  selector:
    app: tf-notebook
  type: LoadBalancer                                    #阿里云的负载均衡访问内部
服务和负载均衡
```

旧的GPU部署方案，您必须要定义如下的nvidia驱动所在的数据卷。

```
volumes:
    - hostPath:
        path: /usr/lib/nvidia-375/bin
        name: bin
    - hostPath:
        path: /usr/lib/nvidia-375
        name: lib
```

这需要您在编写部署文件时，强依赖于所在的集群，导致缺乏可移植性。但是在Kubernetes 1.9.3及之后的版本中，最终用户无需指定这些hostPath，nvidia的插件会自发现驱动所需的库链接和执行文件。

5. 单击左侧导航栏中的应用 > 服务，选择所需的集群和命名空间，选择tf-notebook服务，查看外部端点。



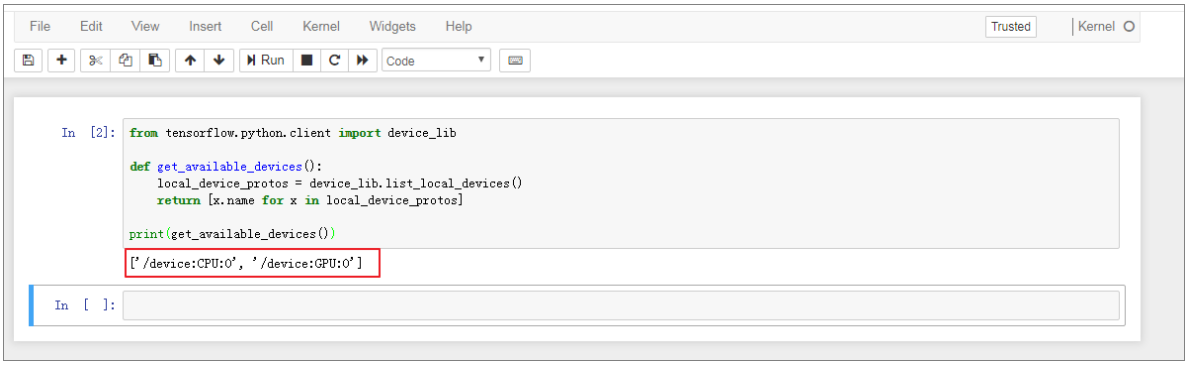
6. 在浏览器中访问Jupyter实例，访问地址是`http://EXTERNAL-IP`，输入模板中配置的密码。

7. 您可通过如下的程序，验证这个Jupyter实例可以使用GPU。它将列出Tensorflow可用的所有设备。

```
from tensorflow.python.client import device_lib

def get_available_devices():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos]

print(get_available_devices())
```



1.3.5 VPC下 Kubernetes 的网络地址段规划

在阿里云上创建 Kubernetes 集群时，通常情况下，可以选择自动创建专有网络，使用默认的网络地址即可。在某些复杂的场景下，需要您自主规划 ECS 地址、Kubernetes Pod 地址和 Service 地址。本文将介绍阿里云 VPC 环境下 Kubernetes 里各种地址的作用，以及地址段该如何规划。

Kubernetes 网段基本概念

首先来看几个和 IP 地址有关的概念。

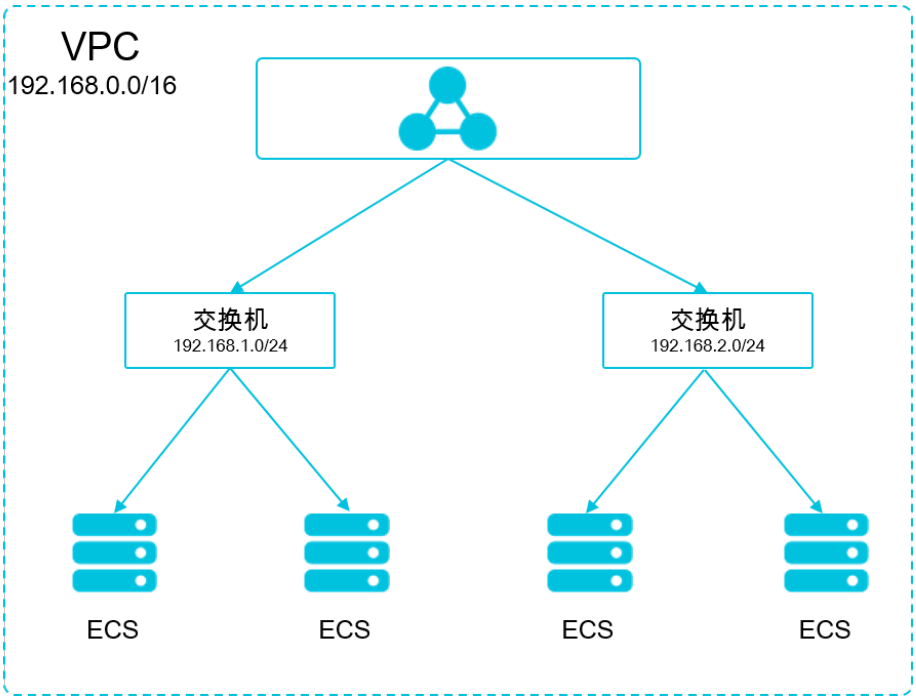
VPC 网段

您在创建 VPC 选择的地址段。只能从 10.0.0.0/8、172.16.0.0/12、192.168.0.0/16 三者当中选择一个。

交换机网段

在 VPC 里创建交换机时指定的网段，必须是当前 VPC 网段的子集（可以跟 VPC 网段地址一样，但不能超过）。交换机下面的 ECS 所分配到的地址，就是从这个交换机地址段内获取的。一个 VPC 下，可以创建多个交换机，但交换机网段不能重叠。

VPC 网段结构如下图。



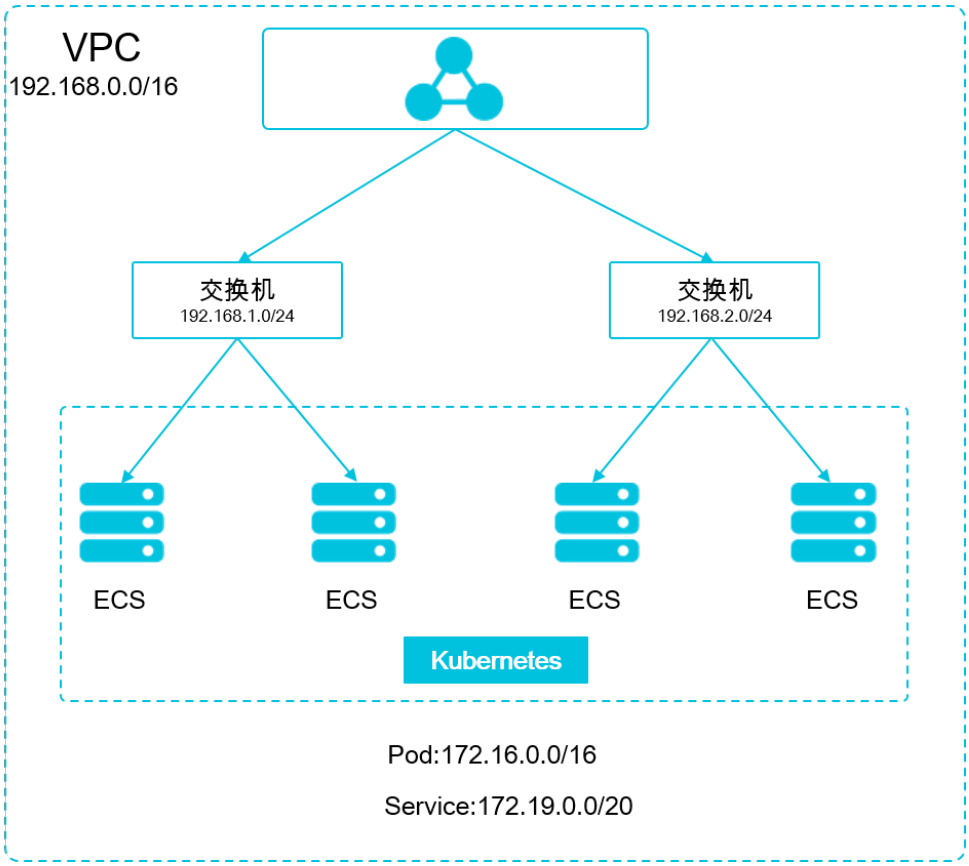
Pod 地址段

Pod 是 Kubernetes 内的概念，每个 Pod 具有一个 IP 地址。在阿里云容器服务上创建 Kubernetes 集群时，可以指定Pod 的地址段，不能和 VPC 网段重叠。比如 VPC 网段用的是 172.16.0.0/12，Kubernetes 的 Pod 网段就不能使用 172.16.0.0/16，不能使用 172.17.0.0/16…，这些地址都涵盖在 172.16.0.0/12 里了。

Service 地址段

Service 也是 Kubernetes 内的概念，每个 Service 有自己的地址。同样，Service 地址段也不能和 VPC 地址段重合，而且 Service 地址段也不能和 Pod 地址段重合。Service 地址只在 Kubernetes 集群内使用，不能在集群外使用。

Kubernetes 网段和 VPC 网段关系如下图。



如何选择地址段

单 VPC+单 Kubernetes 集群场景

这是最简单的情形。VPC 地址在创建 VPC 的时候就已经确定，创建 Kubernetes 集群时，选择和当前 VPC 不一样的地址段就可以了。

单 VPC+多 Kubernetes 集群场景

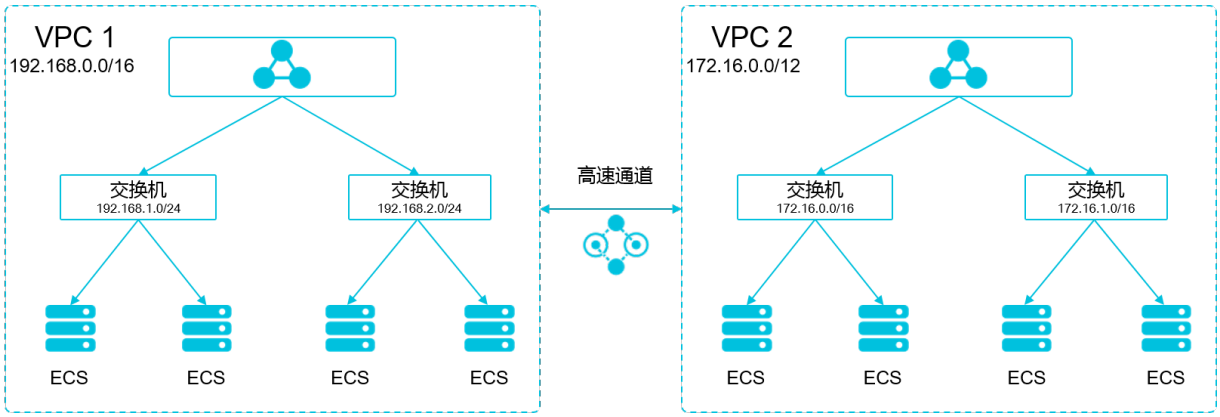
一个 VPC 下创建多个 Kubernetes 集群。在默认的网络模式下（Flannel），Pod 的报文需要通过 VPC 路由转发，容器服务会自动在 VPC 路由上配置到每个 Pod 地址段的路由表。所有 Kubernetes 集群的 Pod 地址段不能重叠，但 Service 地址段可以重叠。

VPC 地址还是创建 VPC 的时候确定的，创建 Kubernetes 的时候，为每个 Kubernetes 集群选择一个不重叠的地址段，不仅不能和 VPC 地址重叠，也不和其他 Kubernetes Pod 段重叠。

需要注意的是，这种情况下 Kubernetes 集群部分互通，一个集群的 Pod 可以直接访问另外一个集群的 Pod 和 ECS，但不能访问另外一个集群的 Service。

VPC 互联场景

两个 VPC 网络互联的情况下，可以通过路由表配置哪些报文要发送到对端 VPC 里。以下面的场景为例，VPC 1 使用地址段 192.168.0.0/16，VPC 2 使用地址段 172.16.0.0/12，我们可以通过路由表，指定在 VPC 1 里把 目的地址为 172.16.0.0/12 的报文都发送到 VPC 2。



在这种情况下，VPC 1 里创建的 Kubernetes 集群，首先不能和 VPC 1 的地址段重叠，同时其地址段也不能和 VPC 2 的地址段重叠。在 VPC 2 上创建 Kubernetes 集群也类似。这个例子中，Kubernetes 集群 Pod 地址段可以选择 10.0.0.0/8 下的某个子段。

> 说明：
> 这里要特别关注“路由到 VPC 2” 的地址段，可以把这部分地址理解成已经占用的地址，Kubernetes集群不能和已经占用的地址重叠。

如果 VPC 2 里要访问 VPC 1 的 Kubernetes Pod，则需要在 VPC 2 里配置到 VPC1 Kubernetes集群pod地址的路由。

VPC 网络到 IDC 的场景

和 VPC 互联场景类似，同样存在 VPC 里部分地址段路由到 IDC ，Kubernetes 集群的 Pod 地址就不能和这部分地址重叠。IDC 里如果需要访问 Kubernetes 里的 Pod 地址，同样需要在 IDC 端配置到专线 VBR 的路由表。

1.3.6 创建多可用区 Kubernetes 集群

您可以创建多可用区Kubernetes集群，保证集群高可用。

前提条件

- 您需要开通容器服务、资源编排（ROS）服务和访问控制（RAM）服务。

  登录 容器服务管理控制台、*ROS* 管理控制台 和 *RAM* 管理控制台 开通相应的服务。

> **说明：**
> 容器服务 Kubernetes 集群部署依赖阿里云资源编排 ROS 的应用部署能力，所以创建 Kubernetes 集群前，您需要开通 ROS。

- 您需要先创建一个 VPC 并在该 VPC 中至少创建三个 VSwitch。为了达到高可用的效果，建议您将 VSwitch 创建在不同的可用区。
- 你需要给VPC下的每个VSwtich都手动配置SNAT，否则VPC内实例不能正常访问公网。

背景信息

您可以通过容器服务管理控制台创建包含不同可用区 ECS 实例的 Kubernetes 集群从而实现高可用。

使用须知

创建集群过程中，容器服务会进行如下操作：

- 创建 ECS，配置管理节点到其他节点的 SSH 的公钥登录，通过 CloudInit 安装配置 Kubernetes 集群。
- 创建安全组，该安全组允许 VPC 入方向全部 ICMP 端口的访问。
- 创建 RAM 子账号和 AK，该子账号拥有 ECS 的查询、实例创建和删除的权限，添加和删除云盘的权限，SLB 的全部权限，云监控的全部权限，VPC 的全部权限，日志服务的全部权限，NAS 的全部权限。Kubernetes 集群会根据用户部署的配置相应的动态创建 SLB，云盘，VPC路由规则。
- 创建内网 SLB，暴露 6443 端口。
- 创建公网 SLB，暴露 6443端口（如果您在创建集群时选择开放公网 SSH 登录，会暴露 22 端口；如果选择不开放公网 SSH 登录，则不会暴露 22 端口）。

使用限制

- 随集群一同创建的负载均衡实例只支持按量付费的方式。
- Kubernetes 集群仅支持专有网络 VPC。
- 每个账号默认可以创建的云资源有一定的配额，如果超过配额创建集群会失败。请在创建集群前确认您的配额。如果您需要提高您的配额，请提交工单申请。
  - 每个账号默认最多可以创建 5 个集群（所有地域下），每个集群中最多可以添加 40 个节点。如果您需要创建更多的集群或者节点，请提交工单申请。

- 每个账号默认最多可以创建 100 个安全组。
- 每个账号默认最多可以创建 60 个按量付费的负载均衡实例。

- ECS 实例使用限制：
  - 仅支持 CentOS 操作系统。
  - 支持创建按量付费和包年包月的 ECS 实例。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。单击页面右上角的创建**Kubernetes** 集群。
3. 在创建Kubernetes集群页面，选择多可用区**Kubernetes**。
4. 填写集群的名称。

   集群名称应包含 1~63 个字符，可包含数字、汉字、英文字符或连字符（-）。
5. 选择集群所在的地域。
6. 选择专有网络。

   在已有 VPC 列表中选择一个 VPC 并在该 VPC 下选择三个交换机。为了达到高可用的效果，建议您选择位于不同可用区的交换机。

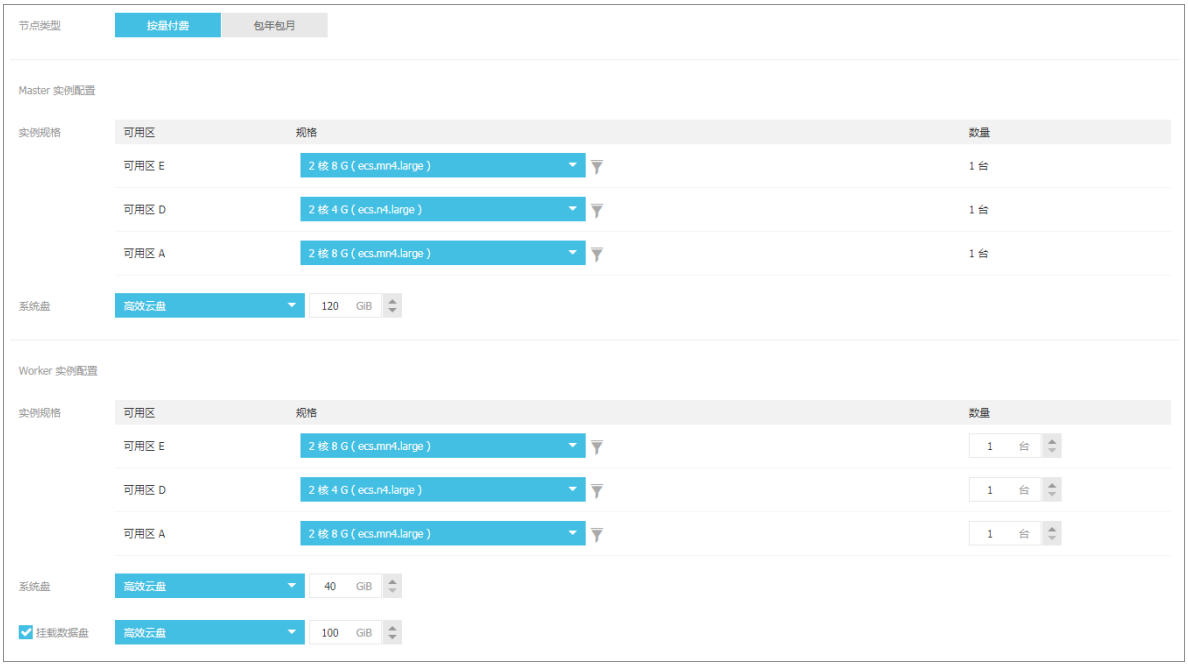
7. 设置 Master 节点和 Worker 节点的配置信息。

   a) 首先设置节点的付费类型，支持按量付费和包年包月两种类型。

   b) 您需要选择 Master 节点和 Worker 节点的实例规格，以及需要创建的 Worker 节点的数量。

> **说明：**
>
> - 目前仅支持 CentOS 操作系统。
> - 目前仅支持创建 3 个 Master 节点。

- 每个集群最多可包含 37 个 Worker 节点。如果您需要创建更多节点，请提交工单申请。
- Master和Worker节点会默认挂载系统盘，支持高效云盘和SSD云盘。
- 您可为Worker节点挂载数据盘，支持高效云盘和SSD云盘。



8. 配置登录方式。

   - 设置密钥。

     您需要在创建集群的时候选择密钥对登录方式，单击新建密钥对，跳转到ECS云服务器控制台，创建密钥对，参见创建 SSH 密钥对。密钥对创建完毕后，设置该密钥对作为登录集群的凭据。

   - 设置密码。

     — **登录密码**：设置节点的登录密码。

     — **确认密码**：确认设置的节点登录密码。

9. 指定**Pod 网络 CIDR** 和**Service CIDR**。

两者都不能与 VPC 及 VPC 内已有 Kubernetes 集群使用的网段重复，创建成功后不能修改。而且 Service 地址段也不能和 Pod 地址段重复，有关 kubernetes 网络地址段规划的信息，请参考*VPC下 Kubernetes 的网络地址段规划*。

10. 设置是否开放用公网**SLB暴露API SERVER**

API Server提供了各类资源对象（Pod，Service等）的增删改查及watch等HTTP Rest接口。

1. 如果选择开放，会创建一个公网SLB，同时把Master节点的6443端口（对应API Server）暴露出来，用户可以在外网通过kubeconfig连接/操作集群。
2. 若选择不开放，不会创建公网SLB，用户只能在VPC内部用kubeconfig连接/操作集群。

11. 设置是否开放公网 SSH 登录。

> **说明：**
> 您需要开放公网**SLB暴露API SERVER**，才能设置公网SSH登录。

- 选择开放公网 SSH 登录，您可以 SSH 访问集群。
- 选择不开放公网 SSH 登录，将无法通过 SSH 访问集群，也无法通过 kubectl 连接 集群。如果您需要通过 SSH 访问集群实例，可以手动为 ECS 实例绑定 EIP，并配置安全组规则，开放 SSH（22）端口，具体操作参见*SSH访问Kubernetes集群*。

12. 设置是否启用云监控插件。

您可以选择在 ECS 节点上安装云监控插件，从而在云监控控制台查看所创建 ECS 实例的监控信息。

13. 设置是否启用日志服务，您可使用已有Project或新建一个Project。

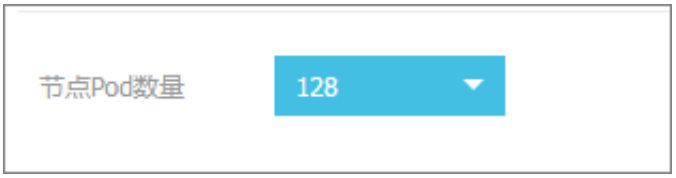
勾选使用**SLS**，会在集群中自动配置日志服务插件。创建应用时，您可通过简单配置，快速使用日志服务，详情参见使用日志服务进行*Kubernetes*日志采集。

14. 设置是否启用高级选项。

    1. 设置启用的网络插件，支持Flannel和Terway网络插件，具体可参考如何选择*Kubernetes*集群网络插件*#Terway*和*Flannel*。

       - Flannel：简单稳定的社区的Flannel CNI插件。但功能偏简单，支持的特性少，例如：不支持基于Kubernetes标准的Network Policy。
       - Terway：阿里云容器服务自研的网络插件，支持将阿里云的弹性网卡分配给容器，支持Kubernetes的**`NetworkPolicy`**来定义容器间的访问策略，支持对单个容器做带宽的限流。

    2. 设置节点 Pod 数量，是指单个节点可运行 Pod 数量的上限。

    3. 设置是否选择自定义镜像。或不选择自定义镜像，则 ECS 实例会安装默认的 CentOS 版本。

       目前您只能选择基于 CentOS 的自定义镜像来快速部署您需要的环境。

    4. 设置是否使用自定义集群**CA**。如果勾选自定义集群 CA，可以将 CA 证书添加到 kubernetes 集群中，加强服务端和客户端之间信息交互的安全性。

15. 单击创建集群，在弹出的配置确认对话框中，确认VPC公网访问，最后单击确定，启动部署。

说明：

一个包含多节点的 Kubernetes 集群的创建时间一般需要十几分钟。

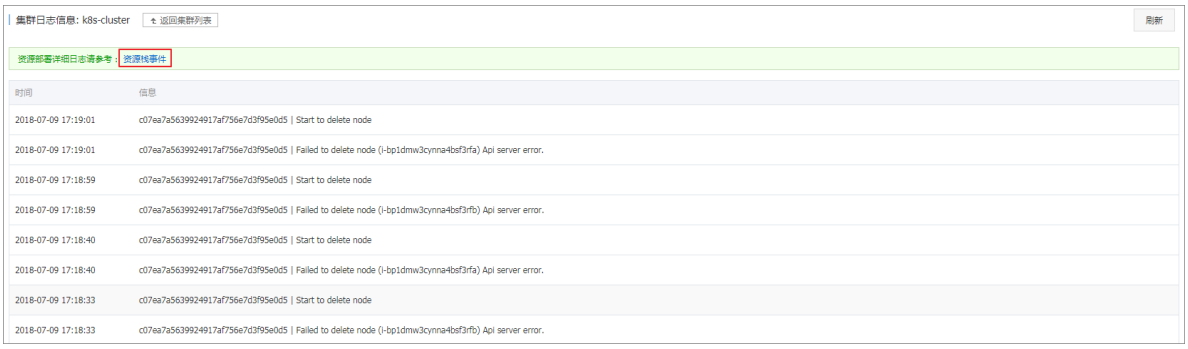
预期结果

查看集群部署结果。

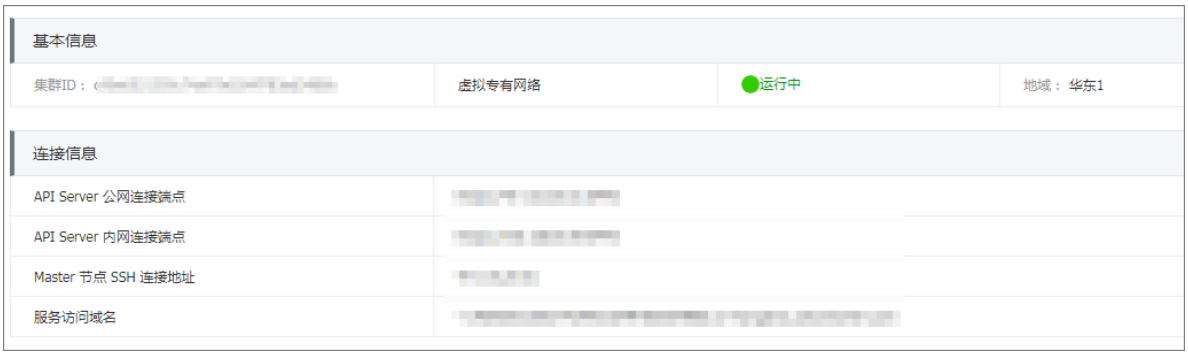
集群创建成功后，您可以在容器服务管理控制台的 Kubernetes 集群列表页面查看所创建的集群。

后续操作

- 在集群列表中，您可以单击操作列的**查看日志**，进入集群日志信息页面，查看集群的日志信息。您也可以在集群日志信息页面中，单击**资源栈事件**查看更详细的信息。



- 在集群列表中，找到刚创建的集群，单击操作列中的管理，查看集群的基本信息和连接信息。



其中：

- **API Server** 公网连接端点：Kubernetes 的 API server 对公网提供服务的地址和端口，可以通过此服务在用户终端使用 kubectl 等工具管理集群。
- **API Server** 内网连接端点：Kubernetes 的 API server 对集群内部提供服务的地址和端口。此 IP 为负载均衡的地址，后端有 3 台 Master 提供服务。
- **Master** 节点 **SSH** 连接地址：可以直接通过 SSH 登录到 Master 节点，以便对集群进行日常维护。
- 服务访问域名：为集群中的服务提供测试用的访问域名。服务访问域名后缀是 `<cluster_id>.<region_id>.alicontainer.com`。

1.3.7 通过 kubectl 连接 Kubernetes 集群

如果您需要从客户端计算机连接到 Kubernetes 集群，请使用 Kubernetes 命令行客户端 *kubectl*。

操作步骤

1. 从*Kubernetes* 版本页面 下载最新的 kubectl 客户端。
2. 安装和设置 kubectl 客户端。

   有关详细信息，参见 安装和设置 *kubectl*。
3. 配置集群凭据。

   您可以使用**scp**命令安全地将主节点的配置从 Kubernetes 集群主 VM 中的 `/etc/kubernetes/kube.conf` 复制到本地计算机的 `$HOME/.kube/config`（`kubectl` 预期凭据所在的位置）。

   - 如果您在创建集群时选择密码方式登录，请用以下方式拷贝 kubectl 配置文件。

     ```
     mkdir $HOME/.kube
     ```

```
scp root@<master-public-ip>:/etc/kubernetes/kube.conf $HOME/.kube/
config
```

- 如果您在创建集群时选择密钥对方式登录，请用以下方式拷贝 kubectl 配置文件。

```
mkdir $HOME/.kube
scp -i [.pem 私钥文件在本地机器上的存储路径] root@:/etc/kubernetes/kube
.conf $HOME/.kube/config
```

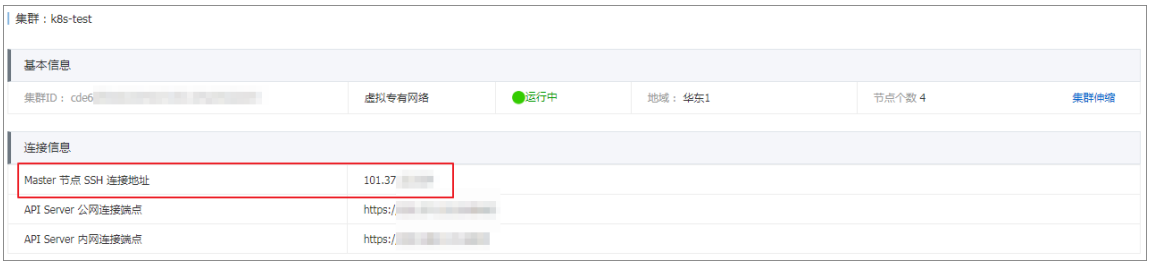
您可以在集群信息页面查看集群的 `master-public-ip`。

a) 登录容器服务管理控制台。

b) 单击**Kubernetes**进入 Kubernetes 集群列表页面。

c) 选择所需的集群并单击右侧的管理

您可以在连接信息处查看集群的连接地址。

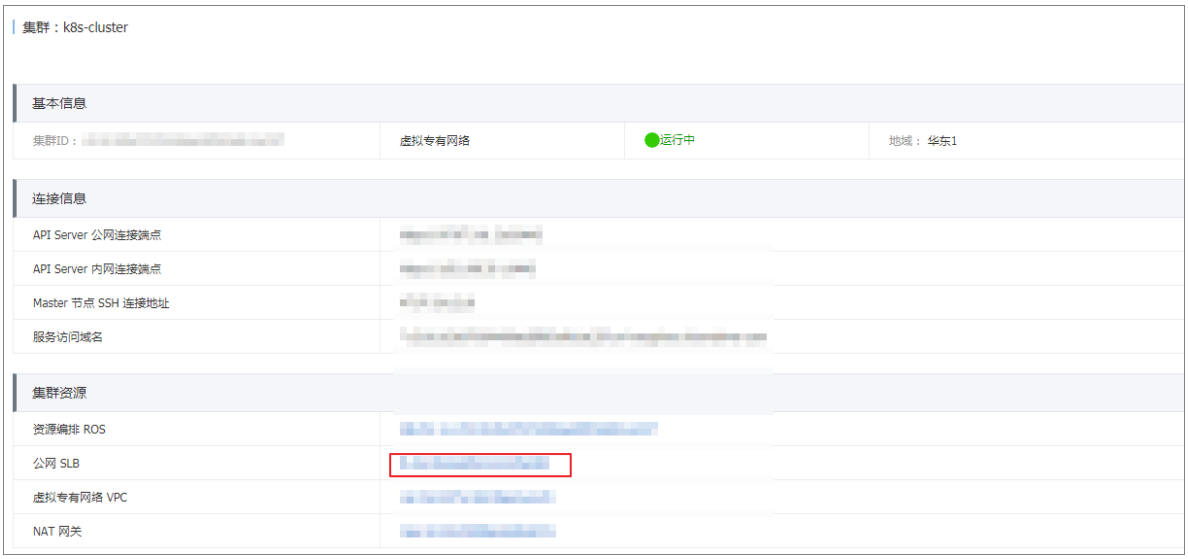


1.3.8 SSH访问Kubernetes集群

如果您在创建集群时，选择不开放公网 SSH 访问，您将无法 SSH 访问 Kubernetes 集群，而且也无法通过 kubectl 连接 Kubernetes 集群。如果创建集群后，您需要 SSH 访问您的集群，可以手动为 ECS 实例绑定 EIP，并配置安全组规则，开放 SSH（22）端口。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的管理。
4. 在集群资源中，找到您的公网负载均衡实例，并单击实例 ID，页面跳转至您的公网负载均衡实例的详细信息页面。



5. 单击左侧导航栏中的实例 > 实例管理，单击添加监听。
6. 添加 SSH 监听规则。
   a. 前端协议（端口）选择 TCP：22。
   b. 后端协议（端口）选择 TCP：22。
   c. 选择 使用服务器组 并选择虚拟服务器组。
   d. 服务器组ID 选择**sshVirtualGroup**。
   e. 单击下一步 并单击确认，创建监听。

7. 监听创建完成后，您就可以使用该负载均衡的服务地址 SSH 访问您的集群了。

1.3.9 SSH密钥对访问Kubernetes集群

阿里云容器服务支持通过 SSH 密钥对的方式登录集群，保障远程SSH访问的安全性。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 单击右上角的创建 **Kubernetes** 集群。

4. 对集群登录方式进行配置，设置通过密钥的登录方式。对集群其他的参数进行配置，参见创建*Kubernetes*集群，最后单击创建。
   1. 若您已在ECS云服务器控制台创建了密钥，只需在密钥对下拉框中进行选择。
   2. 若您没有密钥，单击新建密钥对，前往ECS云服务器控制台创建密钥对，参见创建 *SSH* 密钥对。

5. 集群创建成功后，在集群列表中找到该集群，单击集群右侧的管理，在连接信息中查看**Master**节点**SSH**连接地址。

6. 下载 *.pem*私钥文件，按照本地操作系统环境，如 Windows 和 linux 操作系统，进行相关配置，参见使用*SSH*密钥对连接*Linux*实例。以 linux 环境为例。

   a) 找到您所下载的.pem私钥文件在本地机上的存储路径，如 `/root/xxx.pem`

   b) 运行命令修改私钥文件的属性：`chmod 400 [.pem私钥文件在本地机上的存储路径]`，如 `chmod 400 /root/xxx.pem`。

   c) 运行命令连接至集群：`` `ssh -i [.pem私钥文件在本地机上的存储路径] root@[master-public-ip]`，其中 master-public-ip 即是 Master 节点 SSH 连接地址。如 `ssh -i /root/xxx.pem root@10.10.10.100`。

1.3.10 使用ServiceAccount Token访问托管版Kubernetes集群

本文介绍如何使用ServiceAccount Token访问托管版Kubernetes集群。

背景信息

- 您已经成功创建一个 Kubernetes 托管版集群，参见创建*Kubernetes* 托管版集群。
- 您可以通过Kubectl连接到Kubernetes 托管版集群，参见通过 *kubectl* 连接 *Kubernetes* 集群。

操作步骤

1. 执行以下命令获取API Server 内网连接端点。

```
$ kubectl get endpoints kubernetes
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ kubectl get endpoints kubernetes
NAME         ENDPOINTS           AGE
kubernetes   [redacted]:6443     13d
```

2. 创建*kubernetes-public-service.yaml*文件，**ip**替换为步骤1查询到的API Server 内网连接端点。

```
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-public
spec:
  type: LoadBalancer
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
---
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes-public
  namespace: default
subsets:
- addresses:
  - ip: <API Service address>  #替换为步骤1查询到的API Server内网连接端点
  ports:
  - name: https
    port: 6443
    protocol: TCP
```

3. 执行以下命令，部署公网Endpoints。

```
$ kubectl apply -f kubernetes-public-service.yaml
```

4. 执行以下命令，获取公网SLB地址，即**EXTERNAL-IP**。

```
$ kubectl get service name
```

说明：

此处的*name*与步骤2*kubernetes-public-service.yaml*文件中的*name*保持一致，本文为kubernetes-public。

```
ubuntu-mia@ubuntumia-VirtualBox:~$ kubectl get service kubernetes-public
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
kubernetes-public   LoadBalancer                                   443:     /TCP    7d
```

5. 执行以下命令，查看ServiceAccount对应的Secret，本文的*namespace*以default为例。

```
$ kubectl get secret --namespace=namespace
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ kubectl get secret --namespace=default
NAME                       TYPE                                  DATA   AGE
aliyun-acr-credential-a    kubernetes.io/dockerconfigjson        1      13d
aliyun-acr-credential-b    kubernetes.io/dockerconfigjson        1      13d
                           kubernetes.io/service-account-token   3      13d
```

6. 执行以下命令，获取token值。

```
$ kubectl get secret -n --namespace=namespace -o
 jsonpath={.data.token} | base64 -d
```

> 说明：
>
> 此处的*namespace*与步骤5中的*namespace*保持一致。

7. 执行以下命令，访问托管版Kubernetes集群。

```
$ curl -k -H 'Authorization: Bearer token' https://service-ip
```

> 说明：
>
> - **token**为步骤6获取的token值。
> - **service-ip**为步骤4获取的公网SLB地址，即**EXTERNAL-IP**。

预期结果

执行命令后，出现以下信息，表明连接成功。

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -k -H 'Authorization: Bearer
                                                                    https://
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:serviceaccount:default:default\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}ubuntu-mia@ubuntumia-VirtualBox:~$ ^C
```

1.3.11 升级集群

您可以通过容器服务管理控制台升级您集群的 Kubernetes 版本。

您可以在 Kubernetes 集群列表页面查看您的集群的 Kubernetes 版本。

注意事项

- 集群升级需要机器可以公网访问，以便下载升级所需的软件包。
- 集群升级 Kubernetes 过程中，可能会有升级失败的情况，为了您的数据安全，强烈建议您先打快照然后再升级。有关 ECS 打快照的操作参见创建快照。
- 集群升级 Kubernetes 过程中，集群上部署中的服务会中断，同时无法进行集群和应用的操作，请您在升级之前安排好相关事宜。升级时间大约 5-30 分钟， 升级完成后集群会变成运行中状态。

准备工作

请在集群升级前检查集群的健康状况，并且确保集群健康。

登录 Master 节点，参见SSH访问Kubernetes集群和通过 kubectl 连接 Kubernetes 集群。

1. 执行`kubectl get cs`命令，确保所有模块都处于健康状态。

```
NAME                 STATUS    MESSAGE              ERROR
 scheduler           Healthy   ok
 controller-manager  Healthy   ok
 etcd-0              Healthy   {"health": "true"}
 etcd-1              Healthy   {"health": "true"}
 etcd-2              Healthy   {"health": "true"}
```

2. 执行 `kubectl get nodes` 命令，确保所有节点都处于 Ready 状态。

```
kubectl get nodes
 NAME                                 STATUS    ROLES     AGE
VERSION
 cn-shanghai.i-xxxxxx   Ready     master    38d       v1.9.3
 cn-shanghai.i-xxxxxx   Ready     <none>    38d       v1.9.3
 cn-shanghai.i-xxxxxx   Ready     <none>    38d       v1.9.3
 cn-shanghai.i-xxxxxx   Ready     <none>    38d       v1.9.3
 cn-shanghai.i-xxxxxx   Ready     master    38d       v1.9.3
 cn-shanghai.i-xxxxxx   Ready     master    38d       v1.9.3
```

如果节点不正常可以自行修复，也可以通过提交工单，请阿里云工程师协助修复。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群，并单击更多 > 集群升级 。

4. 在弹出的对话框中，单击升级。

   系统开始升级 Kubernetes 的版本。

升级完成后，您可以在 Kubernetes 集群列表页面查看集群 Kubernetes 的版本，确认升级成功。

1.3.12 扩容和缩容集群

通过容器服务管理控制台，您可以根据实际业务需要对 Kubernetes 集群的 Worker 节点进行扩容和缩容。

背景信息

- 目前不支持集群中 Master 节点的扩容和缩容。
- 集群缩容只能缩减集群创建和扩容时增加的 Worker 节点，不能通过**`kubectl delete`**或在控制台手工删除，通过添加已有节点功能添加到集群中的 Worker 节点不能被缩减。
- 缩减规则是按创建时间进行的，最新扩容出来的节点会被先回收。
- 必须有大于1个非手动添加的节点，才能进行缩容。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的集群伸缩。

4. 选择扩容 或者缩容，设置 Worker 节点的数量。

   本示例进行集群扩容，将集群的 Worker 节点数由 1 个扩容到 4 个。

5. 填写节点的登录密码。

   > **说明：**
   > 由于升级过程依赖登录到 ECS 来拷贝配置信息，所以集群伸缩时填写的密码必须和创建集群时填写的密码一致。

6. 单击提交。

后续操作

伸缩完成后，单击左侧导航栏中的集群 > 节点，查看节点列表页面，您可以看到 Worker 节点的数量从 1 变成了 4。

1.3.13 自动伸缩集群

根据集群负载的工作情况，自动伸缩集群。

前提条件

您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。

背景信息

自动伸缩集群与传统基于资源阈值的扩容和缩容集群不同，根据配置好的伸缩条件，当集群工作负载的工作情况达到配置条件时，集群可进行自动伸缩。

操作步骤

执行自动伸缩

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入Kubernetes集群列表页面。
3. 选择所需的集群并单击操作列的更多 > 自动伸缩。

授权

- 开通**ESS**服务

  1. 单击弹出对话框中的第一个链接，进入弹性伸缩服务 **ESS**页面。

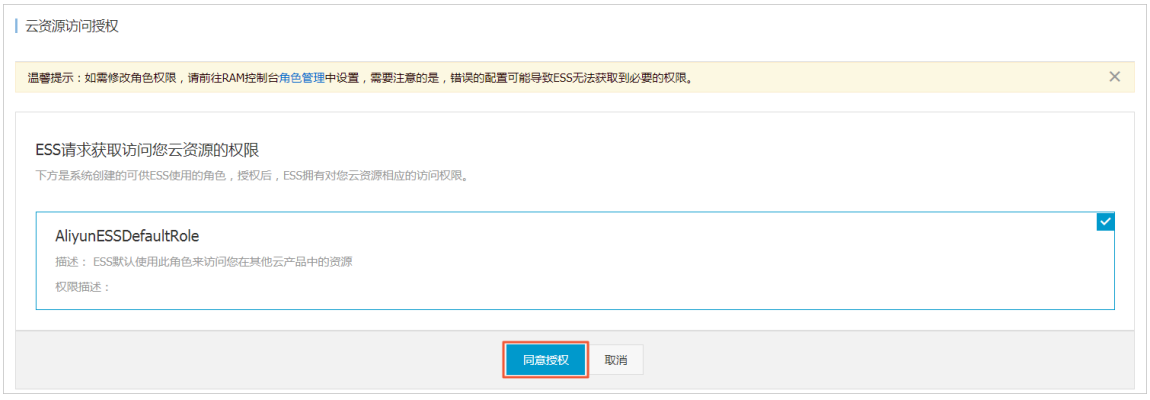
2. 单击开通**ESS**服务，进入云产品开通页。

3. 选中我已阅读并同意复选框，单击立即开通。

4. 开通成功后，在开通完成页签，单击管理控制台，进入弹性伸缩服务 **ESS**页面。

5. 单击**前往授权**，进入**云资源访问授权**页面，配置对云资源的访问权限。

6. 单击**同意授权**。



预期结果

页面自动跳转至弹性伸缩控制台，说明授权成功。关闭页面，继续配置授权角色。

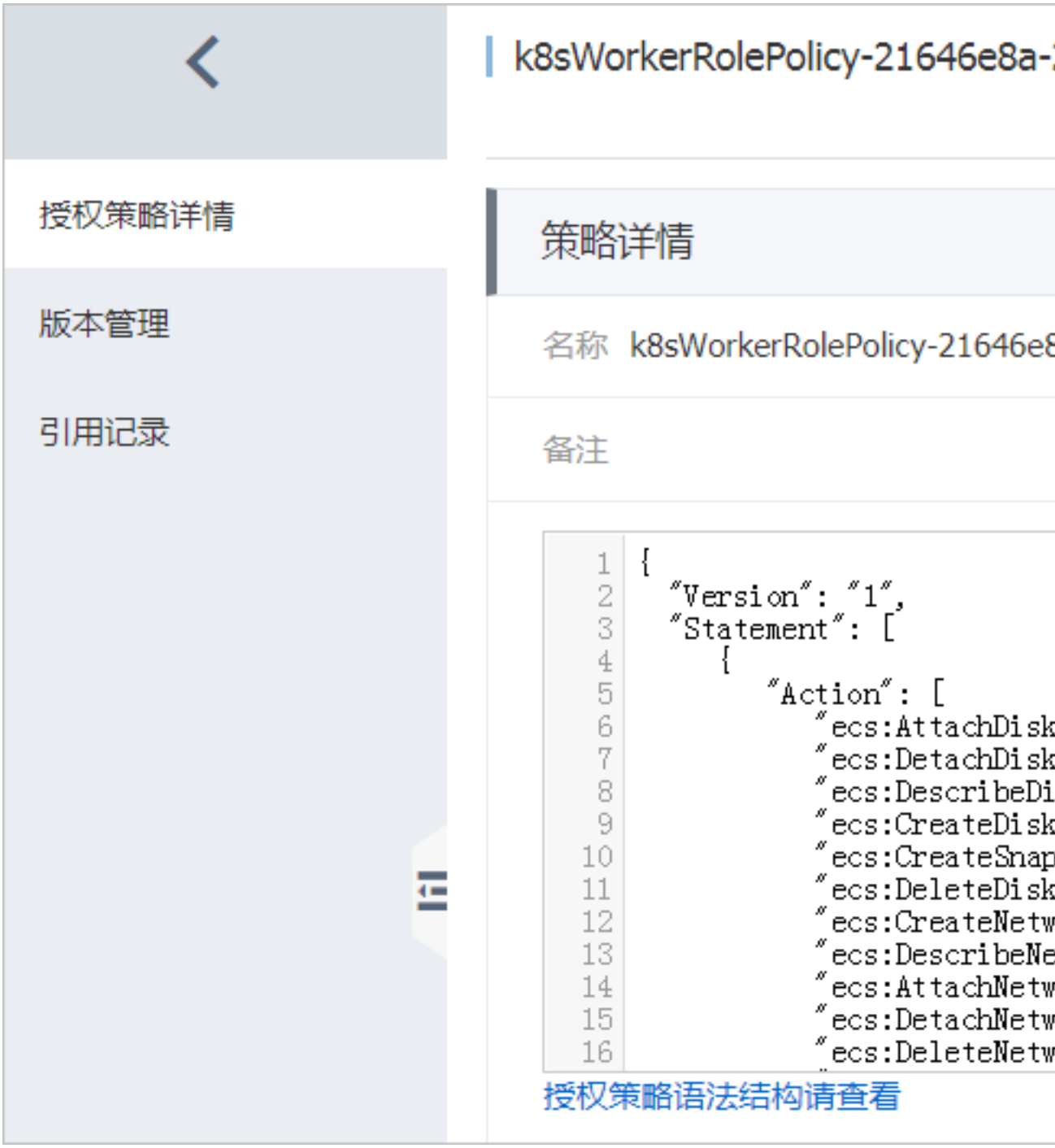
- 授权角色

1. 单击弹出对话框中的第二个链接，进入角色授权策略页面。

> 说明：
> 此处需要以主账号登录控制台。

2. 选择目标授权策略名称并单击操作列查看权限，进入授权策略详情页面。

3. 单击页面右上角修改授权策略。



4. 在策略内容的Action字段，补充以下策略：

```
"ess:Describe*",
"ess:CreateScalingRule",
"ess:ModifyScalingGroup",
"ess:RemoveInstances",
"ess:ExecuteScalingRule",
"ess:ModifyScalingRule",
"ess:DeleteScalingRule",
"ecs:DescribeInstanceTypes",
```

```
"ess:DetachInstances"
```

说明：

需要在Action字段的最后一行补充“，”，再添加以上内容。

5. 单击修改策略。

配置自动伸缩

1. 在弹性伸缩配置页面，填写以下信息，配置自动缩容：

配置	说明
集群	目标集群名称。
缩容阈值	集群中负载申请的资源量与集群资源量的比值。当集群中负载申请的资源值小于等于配置的缩容阈值时，集群会自动缩容。默认情况下，缩容阈值为50%。
缩容触发时延	集群满足配置的缩容阈值时，在配置的缩容触发时延到达后，集群开始缩容。单位：分钟。默认情况下是10分钟。
静默时间	在集群添加或删除节点后，在静默时间内，集群不会再次触发扩容或缩容，单位 ：分钟。默认情况下是10分钟。

2. 根据所需要弹性伸缩的资源类型（CPU/GPU），单击操作列创建。

在伸缩配置页面填写以下信息，创建对应的伸缩组：

配置	说明
地域	所创建伸缩组将要部署到的地域。与伸缩组所在集群的地域一致，不可变更。
可用区	所创建伸缩组的可用区。
专有网络	所创建伸缩组的网络，与伸缩组所在集群的网络一致。

配置 Worker 节点的信息。

配置	说明
实例规格	伸缩组内实例的规格。
系统盘	伸缩组的系统盘。
挂载数据盘	是否在创建伸缩组时挂载数据盘，默认情况下不挂载。
实例数	伸缩组所包含的实例数量。 **说明：** • 实例不包含客户已有的实例。 • 默认情况，实例的最小值是0台，超过0台的时候，集群会默认向伸缩组中添加实例，并将实例加入到伸缩组对应的Kubernetes集群中。
密钥对	登录伸缩后的节点时所使用的密钥对。可以在ECS控制台新建密钥对。 **说明：** 目前只支持密钥对方式登录。
RDS白名单	弹性伸缩后的节点可以访问的RDS实例。

3. 单击**确定**，创建伸缩组。

预期结果

- 在**自动伸缩**页面，可以看到CPU下已创建好一个伸缩组。

- 1. 单击左侧导航栏应用 > 部署，进入部署列表页面。
  2. 选择目标集群和kube-system命名空间，可以看到名称为cluster-autoscaler的组件已创建成功。

1.3.14 删除集群

您可以通过容器服务管理控制台删除不再使用的集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的更多 > 删除。

后续操作

删除集群失败

如果您在 ROS 创建的资源下手动添加了一些资源，ROS 是没有权限删除这些资源的。比如在 ROS 创建的 VPC 下手动添加了一个 VSwitch，这样在就会导致 ROS 删除时无法处理该 VPC，从而最终删除集群失败。

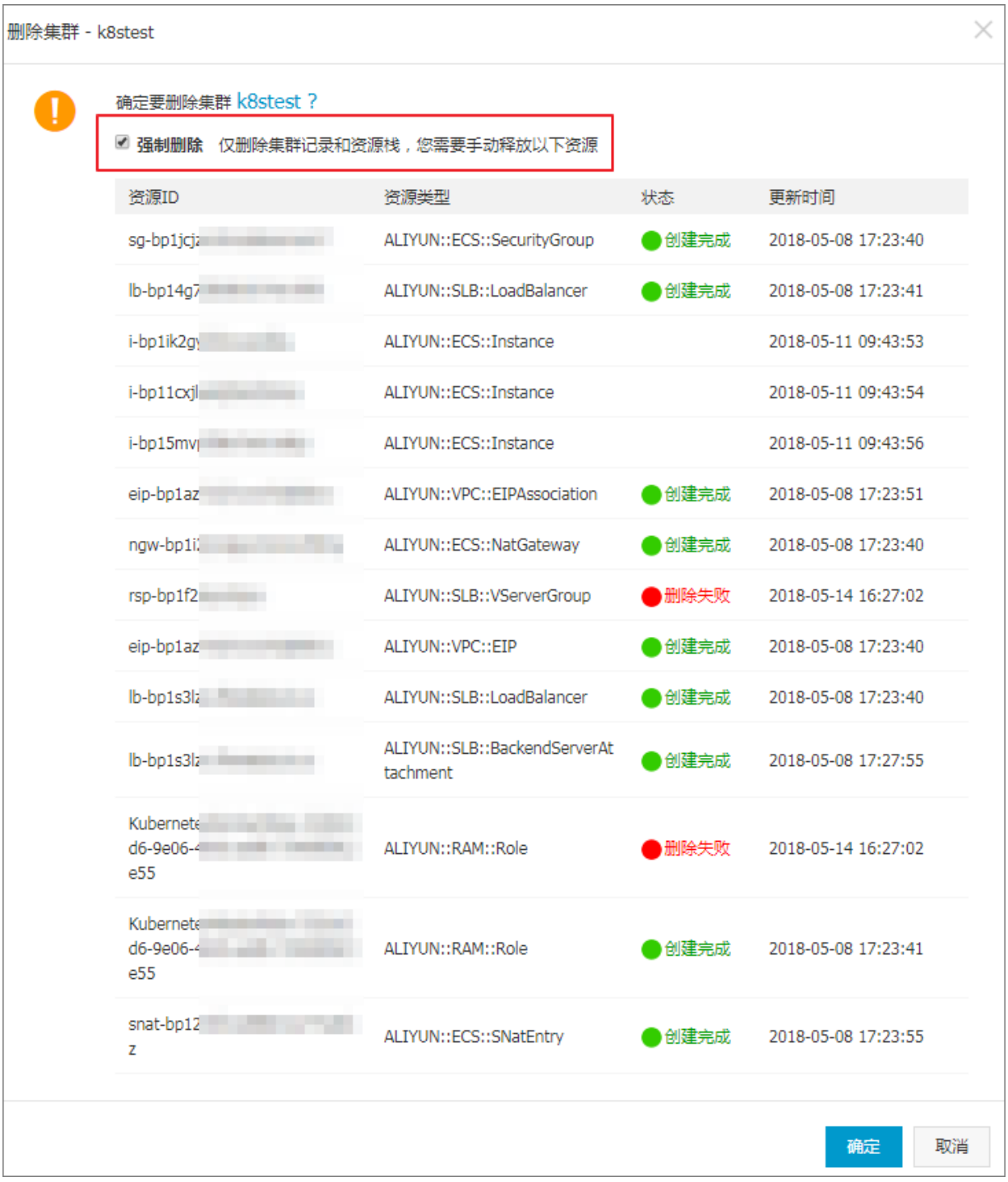
容器服务提供了强制删除集群的功能。通过强制删除功能，您可以在集群删除失败后，强制删除集群记录和 ROS 资源栈。但是，强制删除操作不会自动释放您手动创建的这些资源，您需要手动进行释放。

集群删除失败时，会显示如下信息。

单击删除失败的集群对应的更多 > 删除，在弹出的对话框里，您可以看到删除失败的资源，勾选强制删除 并单击 确定，即可删除集群和 ROS 资源栈。

说明：

该操作不会释放这些资源，您需要手动释放资源。有关如何如果排查不能释放的资源，可参见删除 Kubernetes 集群失败#ROS stack 无法删除。



1.4 节点管理

1.4.1 添加已有节点

您可以向已经创建的 Kubernetes 集群中添加已有的 ECS 实例。目前，仅支持添加 Worker 节点。

前提条件

- 如果之前没有创建过集群，您需要先创建Kubernetes集群。
- 需要先把待添加的 ECS 实例添加到 Kubernetes 集群的安全组里。

背景信息

- 默认情况下，每个集群中最多可包含 40 个节点。如果您需要添加更多节点，请提交工单申请。
- 添加的云服务器必须与集群在同一地域同一 VPC 下。
- 添加已有云服务器时，请确保您的云服务器有EIP（专有网络），或者相应 VPC 已经配置了NAT网关。总之，需要确保相应节点能正常访问公网，否则，添加云服务器会失败。
- 容器服务不支持添加不同账号下的云服务器。
- 仅支持添加操作系统为 CentOS 的节点。

操作步骤

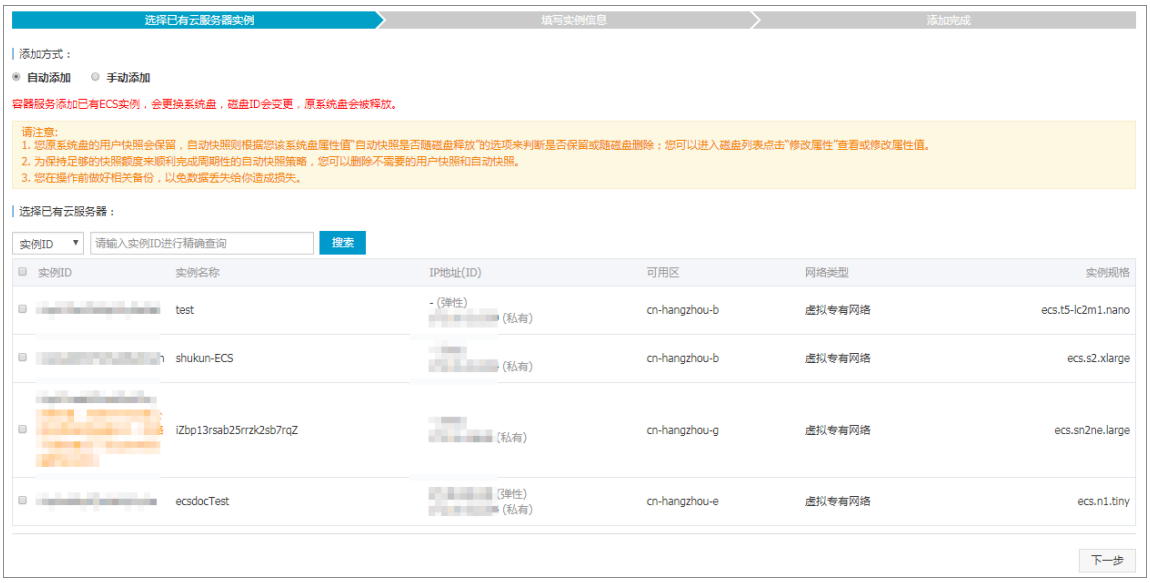
1. 登录 容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的更多 > 添加已有节点。

   进入添加节点页面，您可以选择自动添加或手动添加的方式，添加现有云服务器实例。

   自动添加方式会列出当前账号下可用的 ECS 云服务器，在 Web 界面进行选择，安装部署，并自动添加到集群中；手动添加方式要求您获取安装命令，登录到对应 ECS 云服务器上进行安装，每次只能添加一个 ECS 云服务器。

4. 您可选择自动添加的方式，您可以一次性添加多个ECS云服务器。

   a) 在已有云服务器的列表中，选择所需的ECS云服务器，然后单击下一步。

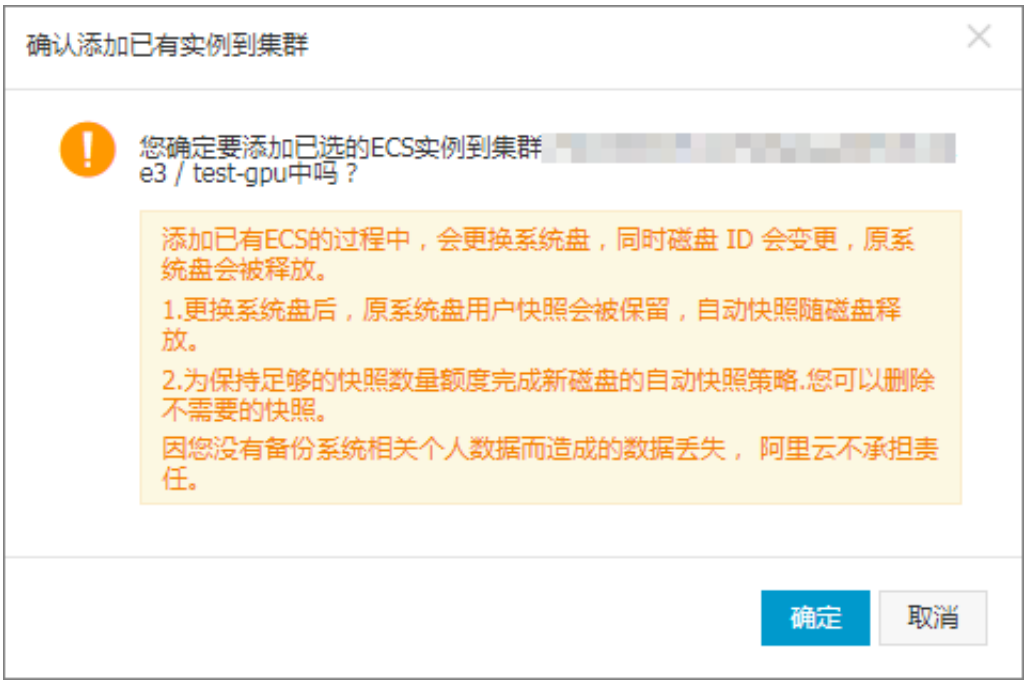


b) 填写实例信息，设置登录密码，然后单击下一步。

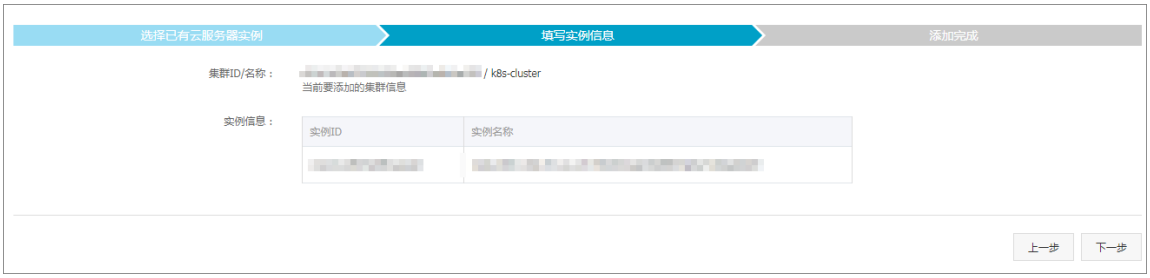
c) 在弹出的对话框中，单击确定，选择的 ECS 云服务器会自动添加到该集群中。

5. （可选）选择手动添加的方式。

a) 选择所需的 ECS 云服务器，单击下一步。您一次只能添加一个 ECS 云服务器。



b) 进入实例信息页面，确认无误后，单击下一步。

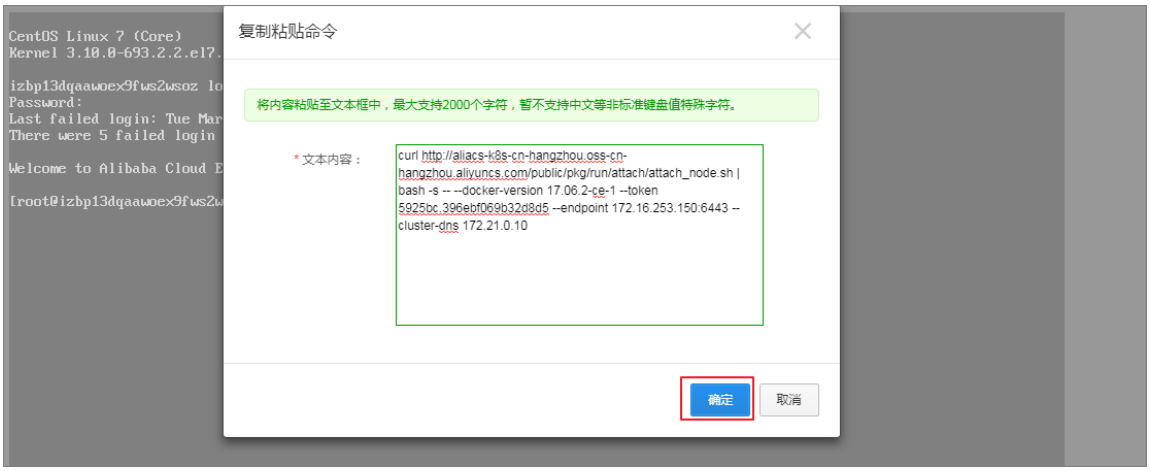


c) 进入添加节点页面，您需要复制其中的执行命令。

d) 最后单击**完成**。

e) 登录 *ECS* 管理控制台，单击左侧导航栏中的**实例**，选择集群所在的地域，选择需要添加的 ECS 实例。

f) 单击 ECS 实例右侧的**远程连接**。进入 ECS 实例远程连接界面，根据页面指导，输入远程连接密码并单击**确定**，成功后，输入上面保存的命令，单击**确定** ，开始执行脚本。



g）等待脚本执行成功，该云服务器即添加成功。您可以在集群列表页面单击集群的 ID 查看该集群下的节点列表。查看节点是否成功添加到集群中。

1.4.2 查看节点列表

您可以通过命令、容器服务管理控制台的节点列表页面或者 Kubernetes Dashboard 的节点列表页面查看 Kubernetes 集群的节点列表。

通过命令查看

> 说明：
> 使用命令查看 Kubernetes 集群的节点列表页面之前，您需要先设置通过 *kubectl* 连接 *Kubernetes* 集群。

通过 kubectl 连接到 Kubernetes 集群后，运行以下命令查看集群中的节点。

```
kubectl get nodes
```

输出示例：

```
$ kubectl get nodes
NAME                        STATUS    AGE       VERSION
iz2ze2n6ep53tch701yh9zz     Ready     19m       v1.6.1-2+ed9e3d33a07093
iz2zeafr762wibijx39e5az     Ready     7m        v1.6.1-2+ed9e3d33a07093
iz2zeafr762wibijx39e5bz     Ready     7m        v1.6.1-2+ed9e3d33a07093
iz2zef4dnn9nos8elyr32kz     Ready     14m       v1.6.1-2+ed9e3d33a07093
iz2zeitvvo8enoreufstkmz     Ready     11m       v1.6.1-2+ed9e3d33a07093
```

通过容器服务管理控制台查看

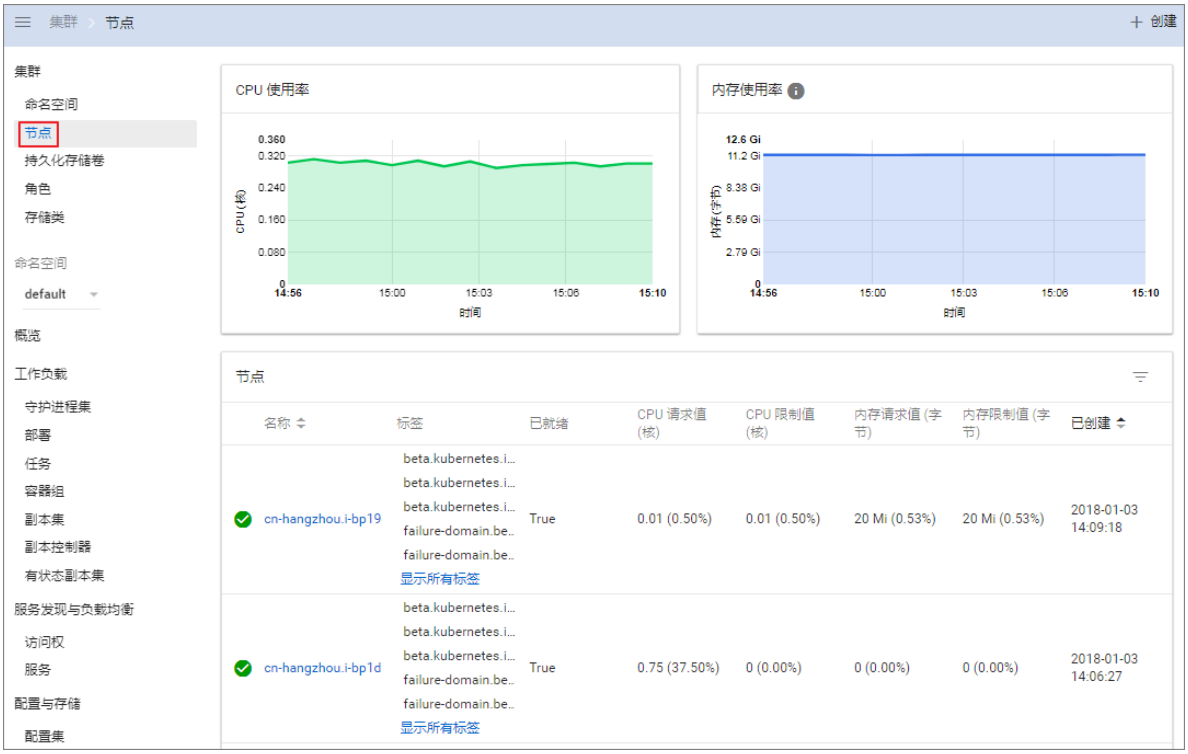
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。

3. 选择所需的集群，您可以查看该集群下的节点列表。

通过 **Kubernetes Dashboard** 查看

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

4. 在 Kubernetes Dashboard 中，单击左侧导航栏中的节点。即可查看集群中的节点列表。



1.4.3 节点监控

kubernetes 集群与阿里云监控服务无缝集成，您可以查看 kubernetes 节点的监控信息，了解Kubernetes集群下 ECS 实例的节点监控指标。

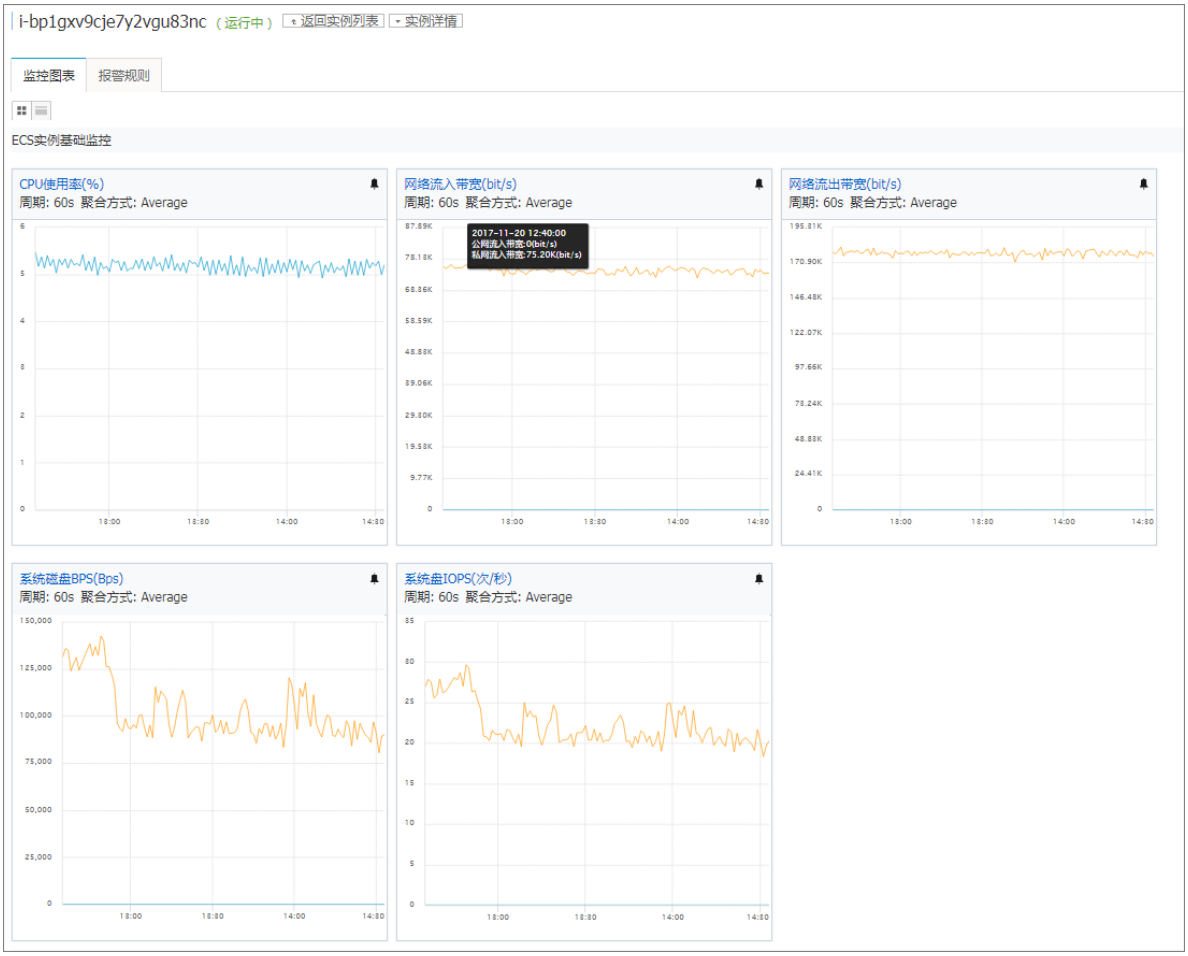
操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。

3. 选择所需的集群，在该集群下选择所需的节点。

4. 单击监控，查看对应节点的监控信息。

5. 进入云监控管理控制台，查看对应 ECS 实例的基本监控信息，包括 CPU使用率、网络流入带宽、网络流出带宽、系统磁盘 BPS、系统盘 IOPS 等指标。



后续操作

要想查看关于操作系统级别的监控指标，需要安装云监控组件。参见主机监控概览。

Kubernetes 集群新增了关于应用分组的监控功能，您可以参见通过资源分组进行监控与告警进行升级。

1.4.4 节点标签管理

您可以通过容器服务 Web 界面对节点进行标签管理，包括批量添加节点标签、通过标签筛选节点和快速删除节点标签。

关于如何使用节点标签实现节点调度，请参见节点调度设置。

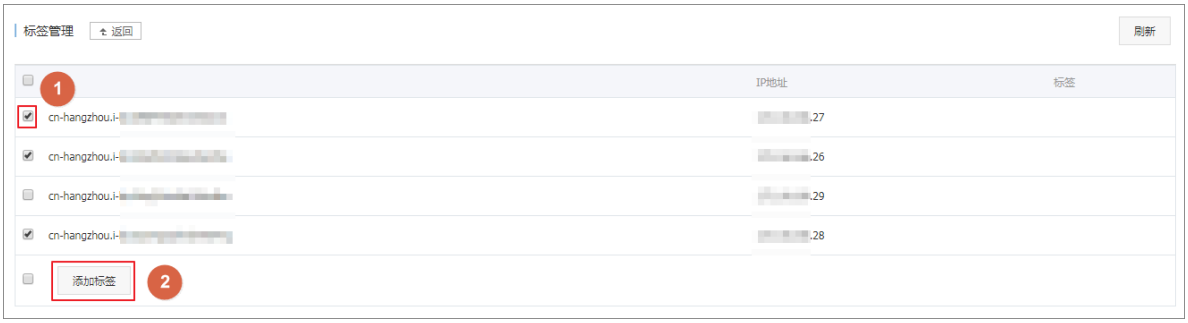
前提条件

您已经成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。

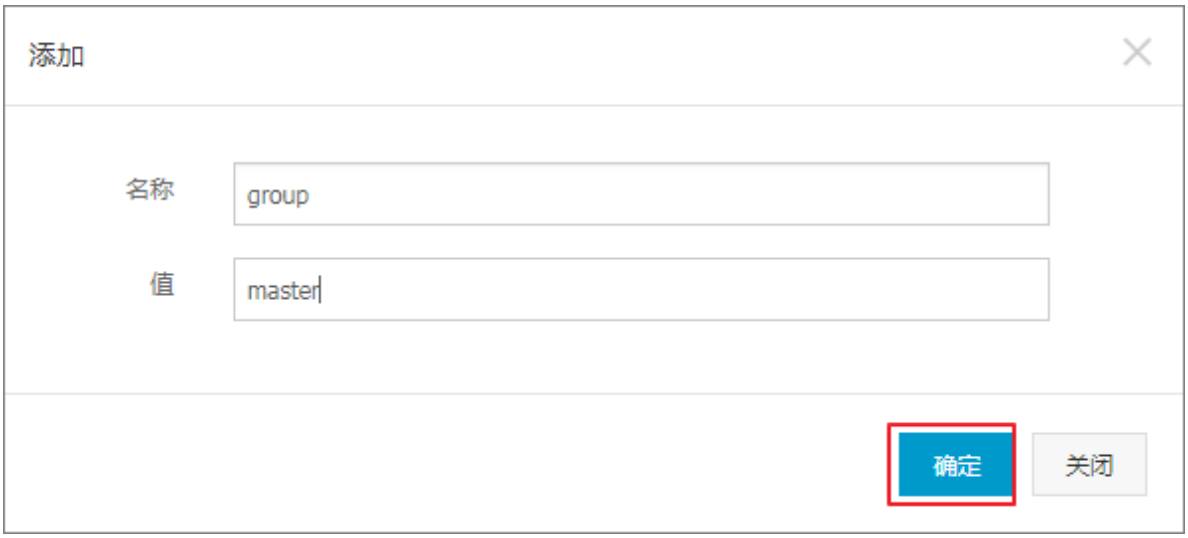
批量添加节点标签

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。
3. 选择所需的集群，在页面右上角单击标签管理。

4. 在节点列表中，批量选择节点，然后单击添加标签。



5. 在弹出的添加标签对话框中，输入标签的名称和值，然后单击确定。



您可以在标签管理页面，看到批量节点具有相同的标签。

通过标签筛选节点

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。
3. 选择所需的集群，在页面右上角单击标签管理。

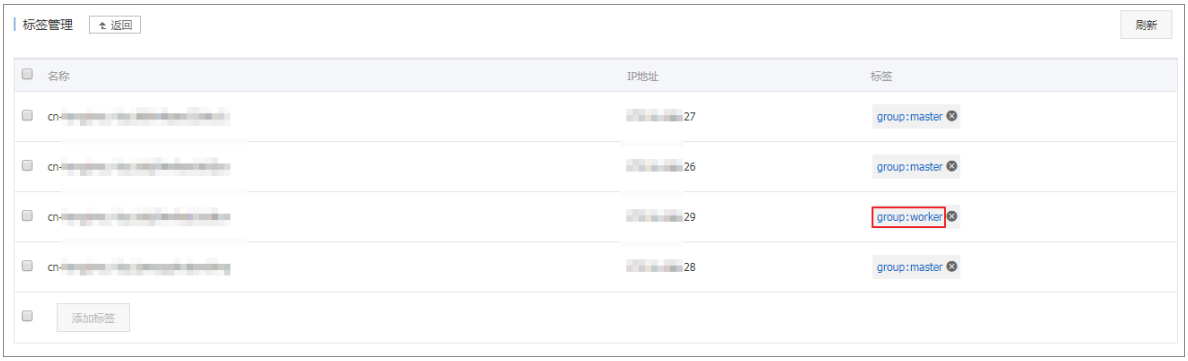
4. 选择某个节点，单击右侧的标签，如 `group:worker`，可通过标签来筛选节点。

   您可看到通过 `group:worker` 标签成功筛选出所需的节点。

删除节点标签

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。
3. 选择所需的集群，在页面右上角单击标签管理。

4. 选择某个节点，单击标签的删除图标，如 `group:worker`。



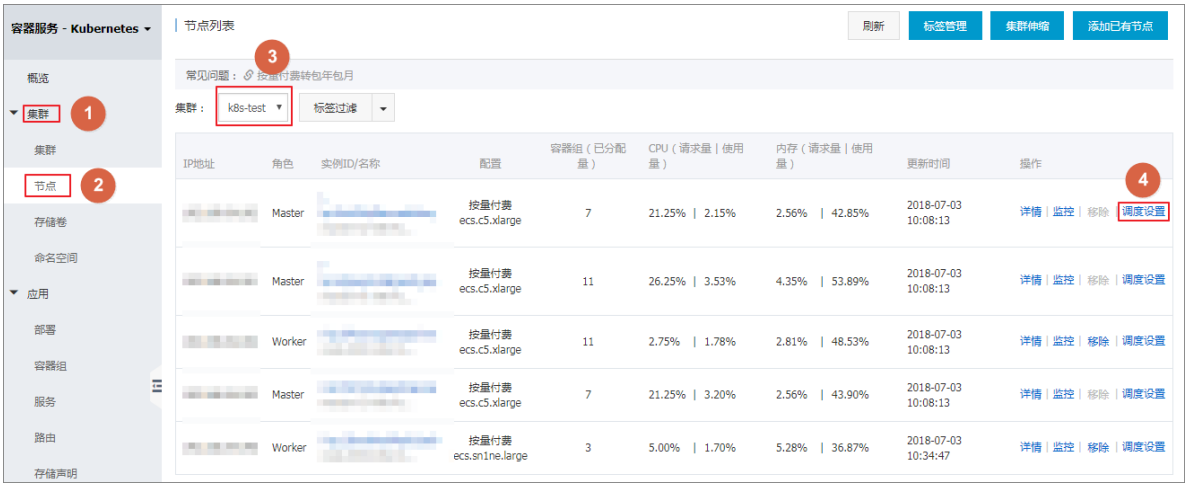
您可以看到该节点右侧的标签消失，节点标签被删除。

1.4.5 节点调度设置

您可以通过Web界面设置节点调度，从而合理分配各节点的负载。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。
3. 选择所需的集群，在该集群下选择所需的节点，单击右侧的调度设置



4. 在弹出的对话框中，进行调度设置。在本例中，单击设置为不可调度，将节点设为不可调度的节点。

> **说明：**
> 调度设置对话框中会显示当前节点的调度状态，默认情况下为可调度，您可进行修改。

设置完毕后，对话框中，节点的调度状态发生变化。

后续操作

您在后续进行应用部署时，会发现Pod不会再调度到该节点。

1.4.6 利用阿里云Kubernetes的GPU节点标签进行调度

在使用Kubernetes集群实现GPU计算时，为了有效利用GPU设备，可根据需要将应用调度到具有GPU设备的节点上，为此，您可利用GPU节点标签进行灵活调度。

前提条件

- 您已成功创建一个拥有GPU节点的Kubernetes集群，参见*Kubernetes GPU集群支持GPU调度*。
- 您已连接到Master节点，方便快速查看节点标签等信息，参见*通过 kubectl 连接 Kubernetes 集群*。

背景信息

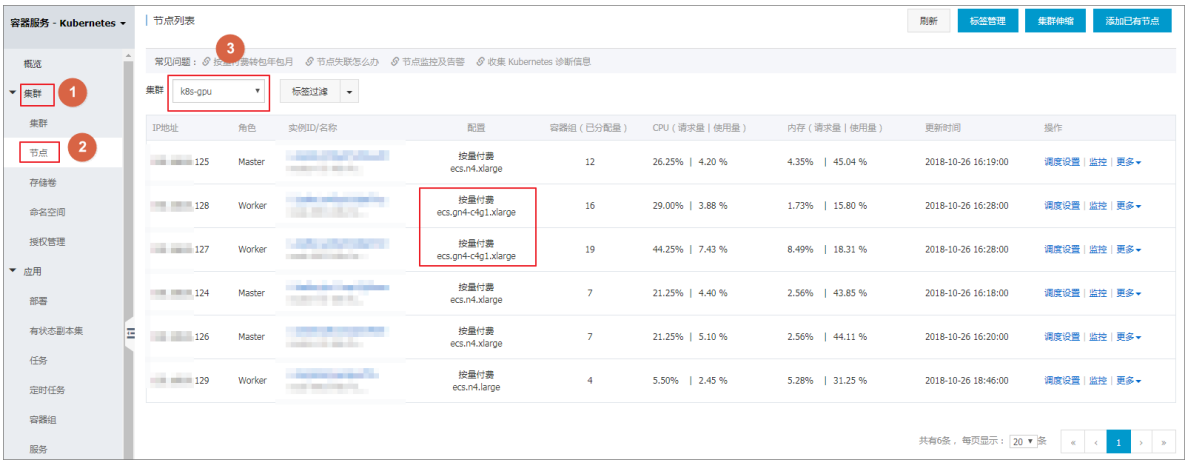
阿里云Kubernetes在部署Nvidia GPU节点的时候会发现GPU的属性，并且作为NodeLabel信息暴露给用户，拥有如下优势：

1. 可以快速筛选GPU节点
2. 部署时刻可以作为调度条件使用

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的集群 > 节点，查看该集群的节点。

> 说明：
> 本例中，该集群中拥有3个Worker节点，其中有两个Worker节点挂载了GPU设备，请查看节点IP，方便进行验证。



3. 选择GPU节点，单击操作列的更多 > 详情，进入Kubernetes Dashboard页面，查看GPU节点提供的节点标签。

您也可登录到Master节点，执行以下命令，查看GPU节点的标签。

```
# kubectl get nodes
NAME                                 STATUS     ROLES     AGE
VERSION
cn-beijing.i-2ze2dy2h9w97v65uuaft    Ready      master    2d        v1
.11.2
cn-beijing.i-2ze8o1a45qdv5q8a7luz    Ready      <none>    2d        v1
.11.2              #可与控制台进行比对，确定GPU节点
cn-beijing.i-2ze8o1a45qdv5q8a7lv0    Ready      <none>    2d        v1
.11.2
cn-beijing.i-2ze9xylyn11vop7g5bwe    Ready      master    2d        v1
.11.2
cn-beijing.i-2zed5sw8snjniq6mf5e5    Ready      master    2d        v1
.11.2
cn-beijing.i-2zej9s0zijykp9pwf7lu    Ready      <none>    2d        v1
.11.2
```

选择一个GPU节点，执行以下命令，查看该GPU节点的标签。

```
# kubectl describe node cn-beijing.i-2ze8o1a45qdv5q8a7luz
```

```
Name:               cn-beijing.i-2ze8o1a45qdv5q8a7luz
Roles:              <none>
Labels:             aliyun.accelerator/nvidia_count=1
          #注意
                    aliyun.accelerator/nvidia_mem=12209MiB
                    aliyun.accelerator/nvidia_name=Tesla-M40
                    beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/instance-type=ecs.gn4-c4g1.
xlarge
                    beta.kubernetes.io/os=linux
                    failure-domain.beta.kubernetes.io/region=cn-
beijing
                    failure-domain.beta.kubernetes.io/zone=cn-
beijing-a
                    kubernetes.io/hostname=cn-beijing.i-2ze8o1a45q
dv5q8a7luz
 ......
```

本例中，该GPU节点包含如下3个节点标签（NodeLabel）。

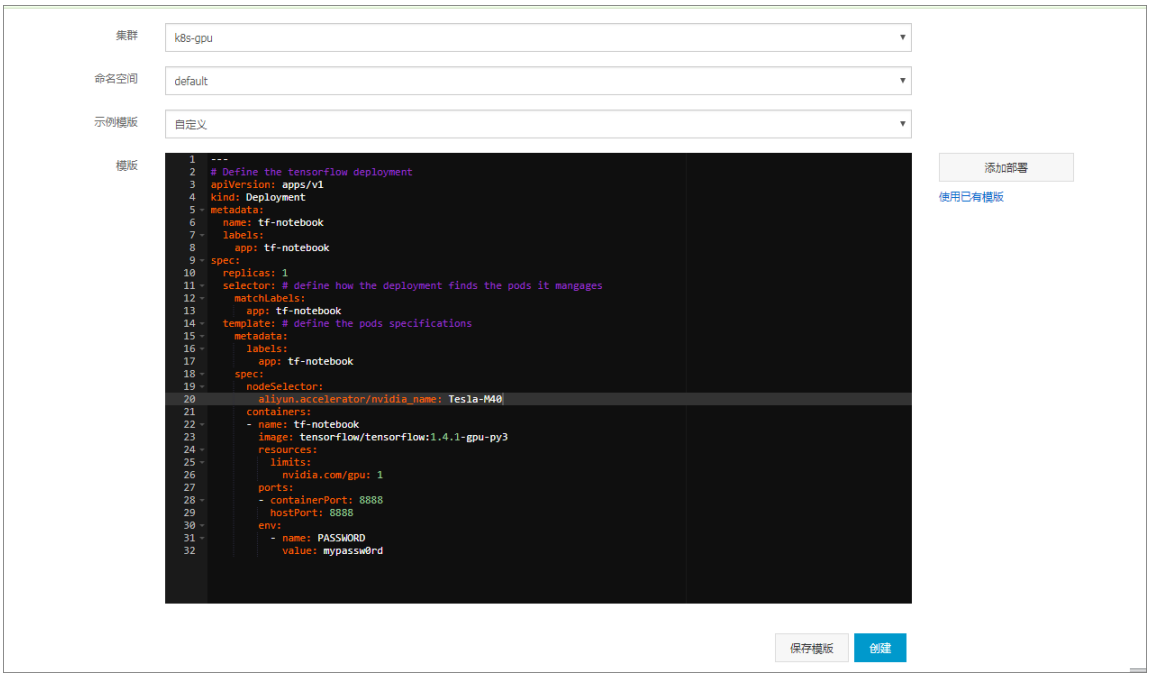
key	value
aliyun.accelerator/nvidia_count	GPU核心数量
aliyun.accelerator/nvidia_mem	GPU显存，单位为MiB
aliyun.accelerator/nvidia_name	nvida设备的GPU计算卡名称

同一类型的GPU云服务器的GPU计算卡名称相同，因此，您可通过该标签筛选节点。

```
# kubectl get no -l aliyun.accelerator/nvidia_name=Tesla-M40
NAME                                 STATUS    ROLES     AGE
VERSION
cn-beijing.i-2ze8o1a45qdv5q8a7luz   Ready     <none>    2d        v1
.11.2
cn-beijing.i-2ze8o1a45qdv5q8a7lv0   Ready     <none>    2d        v1
.11.2
```

4. 返回容器服务控制台主页，单击左侧导航栏应用 > 部署，单击右上角使用模板创建。

   a) 创建一个tensorflow的Deployment，将该应用调度到GPU节点上。



本例的yaml编排如下所示。

```
---
# Define the tensorflow deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tf-notebook
  labels:
    app: tf-notebook
spec:
  replicas: 1
  selector: # define how the deployment finds the pods it mangages
    matchLabels:
      app: tf-notebook
  template: # define the pods specifications
    metadata:
      labels:
        app: tf-notebook
    spec:
      nodeSelector:
   #注意
        aliyun.accelerator/nvidia_name: Tesla-M40
      containers:
      - name: tf-notebook
        image: tensorflow/tensorflow:1.4.1-gpu-py3
        resources:
          limits:
            nvidia.com/gpu: 1
   #注意
        ports:
        - containerPort: 8888
          hostPort: 8888
        env:
          - name: PASSWORD
            value: mypassw0rdv:
```

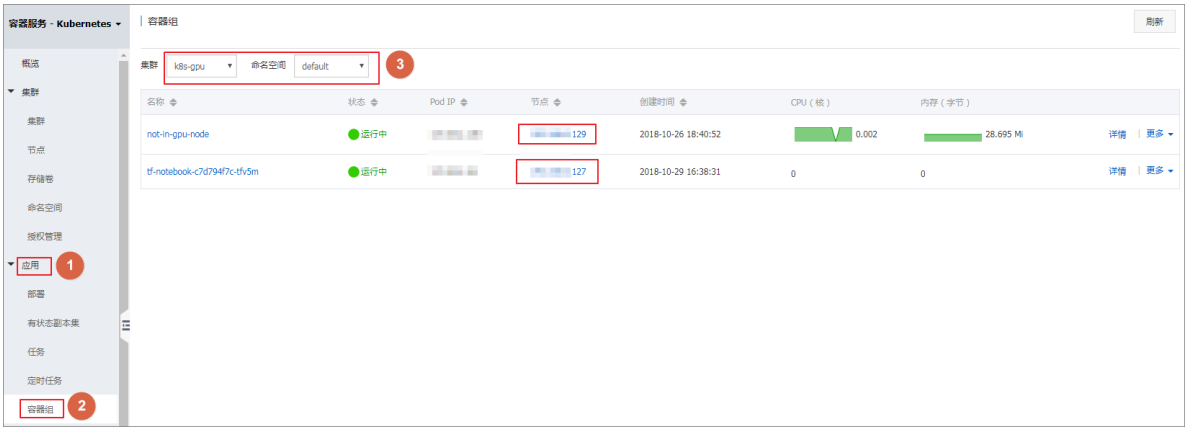
```
            - name: PASSWORD
```

b) 您也可避免将某些应用部署到GPU节点。下面部署一个nginx的Pod，利用节点亲和性的特性进行调度，具体参见使用镜像创建无状态Deployment应用中关于节点亲和性的说明。

该示例的yaml编排如下所示：

```
apiVersion: v1
kind: Pod
metadata:
  name: not-in-gpu-node
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: aliyun.accelerator/nvidia_name
            operator: DoesNotExist
  containers:
  - name: not-in-gpu-node
    image: nginx
```

5. 单击左侧导航栏应用 > 容器组，选择所需的集群和命名空间，进入容器组列表。



预期结果

在容器组列表中，您可看到两个示例的Pod（容器组）成功调度到对应的节点上，从而实现基于GPU节点标签的灵活调度。

1.4.7 查看节点资源请求量/使用量

通过容器服务控制台，您可查看Kubernetes集群各节点资源占用情况。

前提条件

您已成功创建一个Kubernetes集群，参见创建Kubernetes集群。

操作步骤

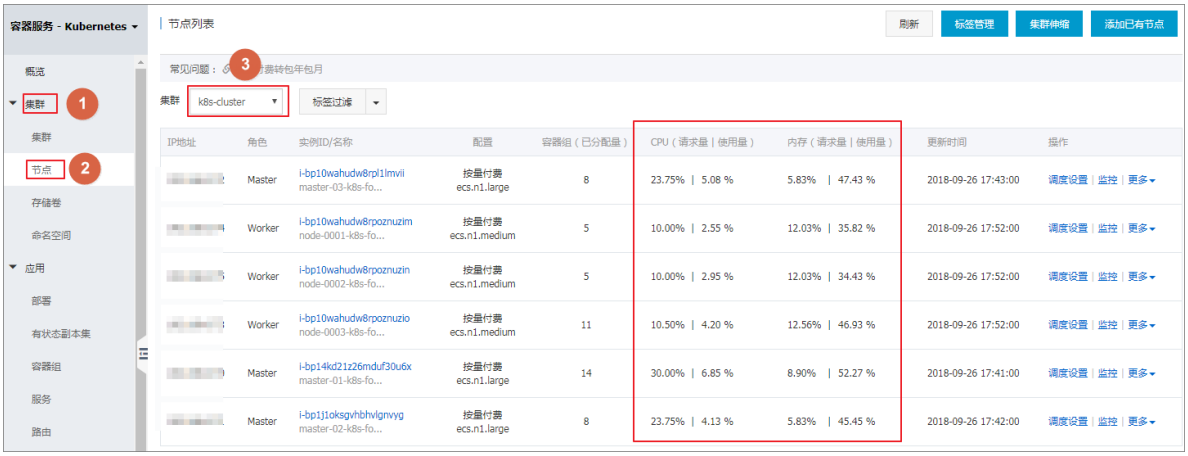
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。

   您可查看各个节点CPU和内存的资源使用情况，即请求量和使用量，其计算方式如下：

   - CPU请求量 = sum(当前节点所有Pod的CPU request值) / 当前节点CPU总量。
   - CPU使用量 = sum(当前节点所有Pod的CPU实际使用量) / 当前节点CPU总量。
   - 内存请求量 = sum(当前节点所有Pod的内存request值) / 当前节点内存总量。
   - 内存使用量 = sum(当前节点所有Pod的内存实际使用量) / 当前节点内存总量。

   说明：

   - 您可根据节点的资源占用情况，规划节点的工作负载，参见节点调度设置。
   - 请求量和使用量为100%的节点时，不会调度新的Pod到该节点上。



1.5 命名空间管理

1.5.1 创建命名空间

前提条件

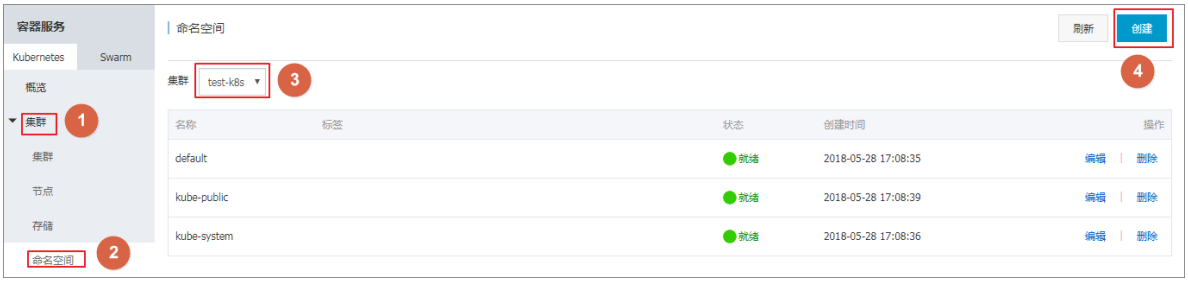
您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。

背景信息

在 Kubernetes 集群中，您可使用 Namespaces（命名空间）功能创建多个虚拟的空间，在集群用户数量较多时，多个命名空间可以有效划分工作区间，将集群资源划分为多个用途，并通过 *resource-quotas* 对命名空间的资源进行分配。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 命名空间，进入命名空间列表页面。
3. 选择所需的集群，单击页面右上角的创建。

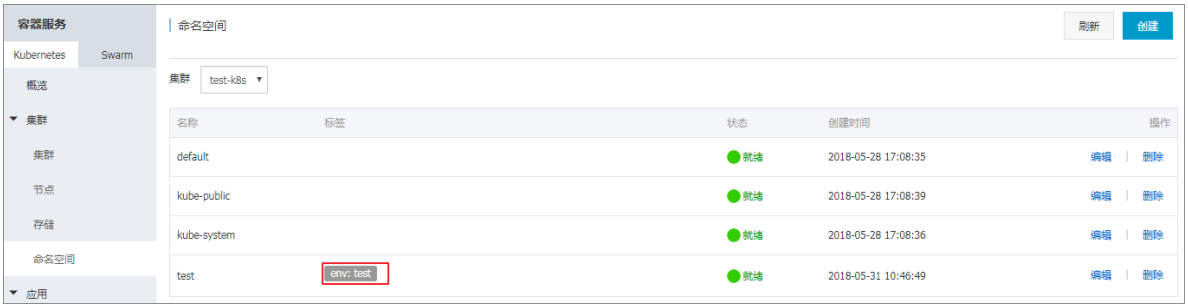


4. 在弹出的对话框中，配置命名空间。

- 名称：您需要设置命名空间的名称，本例中为 test 。长度为 1-63 个字符，只能包含数字、字母、和“-”，且首尾只能是字母或数字
- 标签：您可为命名空间添加多个标签。标签用于标识该命名空间的特点，如标识该命名空间用于测试环境。

  您可输入变量名称和变量值，单击右侧的添加，为命名空间新增一个标签。

5. 完成命名空间设置后，单击确定。
6. 返回命名空间列表，您可看到 test 命名空间出现在列表中，命名空间成功创建。



1.5.2 设置资源配额和限制

您可通过容器服务控制台，设置命名空间的资源配额和限制。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已成功创建一个示例的命名空间**test**，参见创建命名空间。
- 您已成功连接到集群的Master节点地址，参见通过 *kubectl* 连接 *Kubernetes* 集群。

背景信息

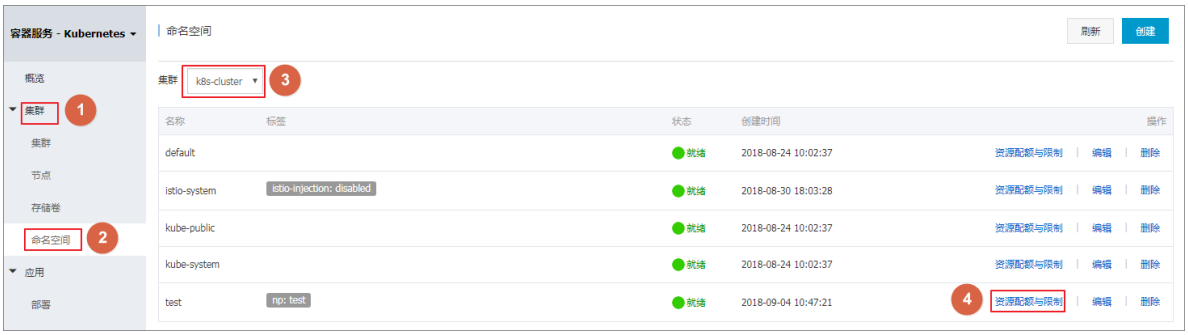
在默认的情况下，运行中的 Pod 可以无限制的使用 Node 上的 CPU 和内存，这意味着任意一个 Pod 都可以无节制地使用集群的计算资源，某个命名空间的 Pod 可能会耗尽集群的资源。

命名空间的一个重要的作用是充当一个虚拟的集群，用于多种工作用途，满足多用户的使用需求，因此，为命名空间配置资源额度是一种最佳实践。

您可为命名空间配置包括 CPU、内存、Pod 数量等资源的额度，更多信息请参见 *Resource Quotas* 。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 命名空间，选择所需的集群，然后右侧的资源配额与限制。

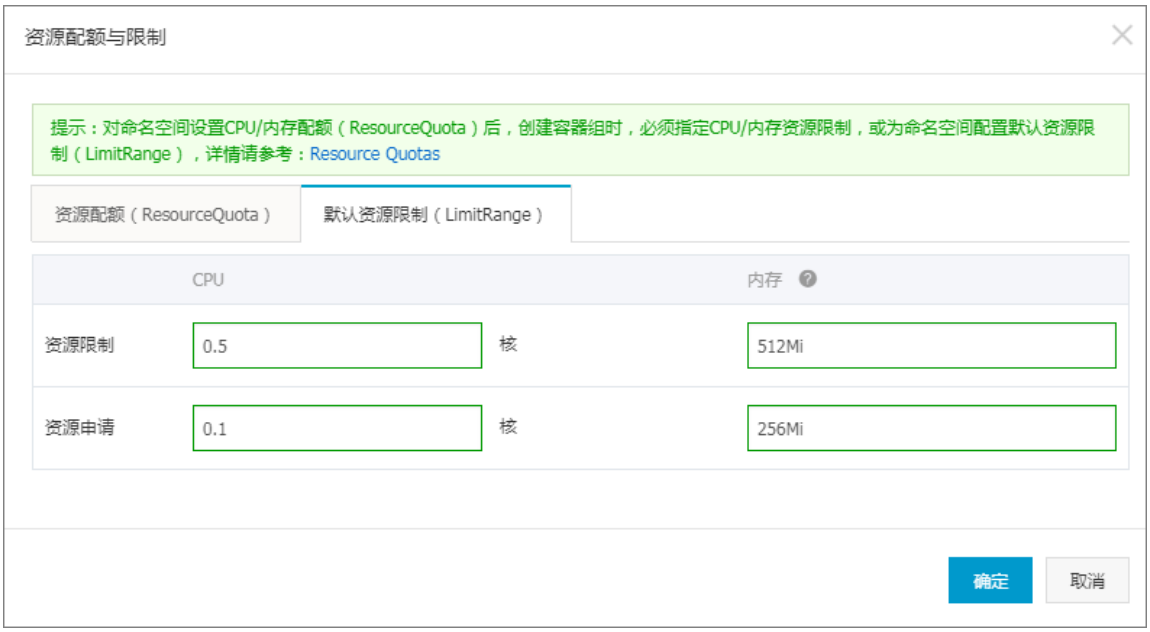


3. 在资源配额与限制对话框中，您可快速设置资源配额和默认资源限制。

> **说明：**
>
> 对命名空间设置CPU/内存配额（ResourceQuota）后，创建容器组时，必须指定CPU/内存资源限制，或为命名空间配置默认资源限制（LimitRange），详情请参考：*Resource Quotas*。

a) 您可为命名空间配置资源限额（ResourceQuota）。

b) 您可为该命名空间下的容器设置资源限制和资源申请（defaultRequest），从而控制容器的开销。详情参见*https://kubernetes.io//memory-default-namespace/*。



4. 您已为该命名空间创建资源配额和限制，连接到 Master 节点连接地址，执行以下命令，查看该命名空间的资源情况。

```
#kubectl get limitrange,ResourceQuota -n test
NAME AGE
limitrange/limits 8m

NAME AGE
resourcequota/quota 8m


# kubectl describe limitrange/limits resourcequota/quota -n test
Name: limits
Namespace: test
Type Resource Min Max Default Request Default Limit Max Limit/
Request Ratio
---- -------- --- --- --------------- -------------
 -----------------------
Container cpu - - 100m 500m -
Container memory - - 256Mi 512Mi -

Name: quota
Namespace: test
Resource Used Hard
-------- ---- ----
configmaps 0 100
limits.cpu 0 2
limits.memory 0 4Gi
persistentvolumeclaims 0 50
pods 0 50
requests.storage 0 1Ti
secrets 1 10
services 0 20
```

```
services.loadbalancers 0 5
```

1.5.3 编辑命名空间

前提条件

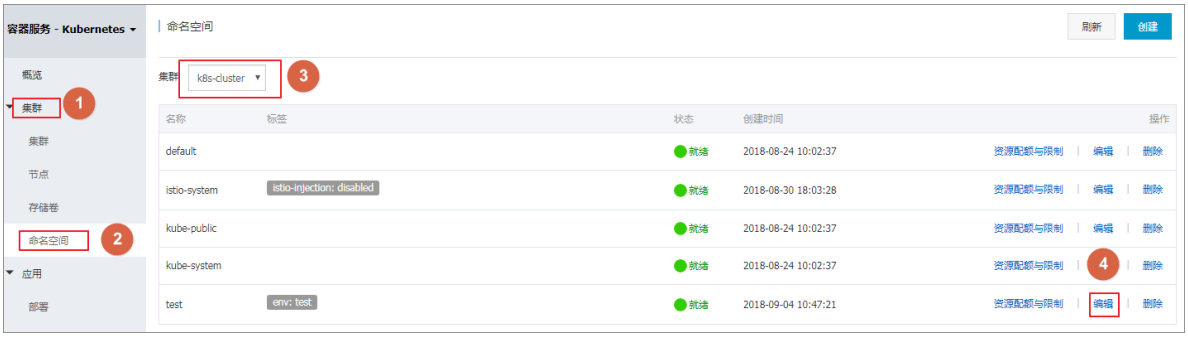
- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已成功创建一个示例的命名空间**test**，参见创建命名空间。

背景信息

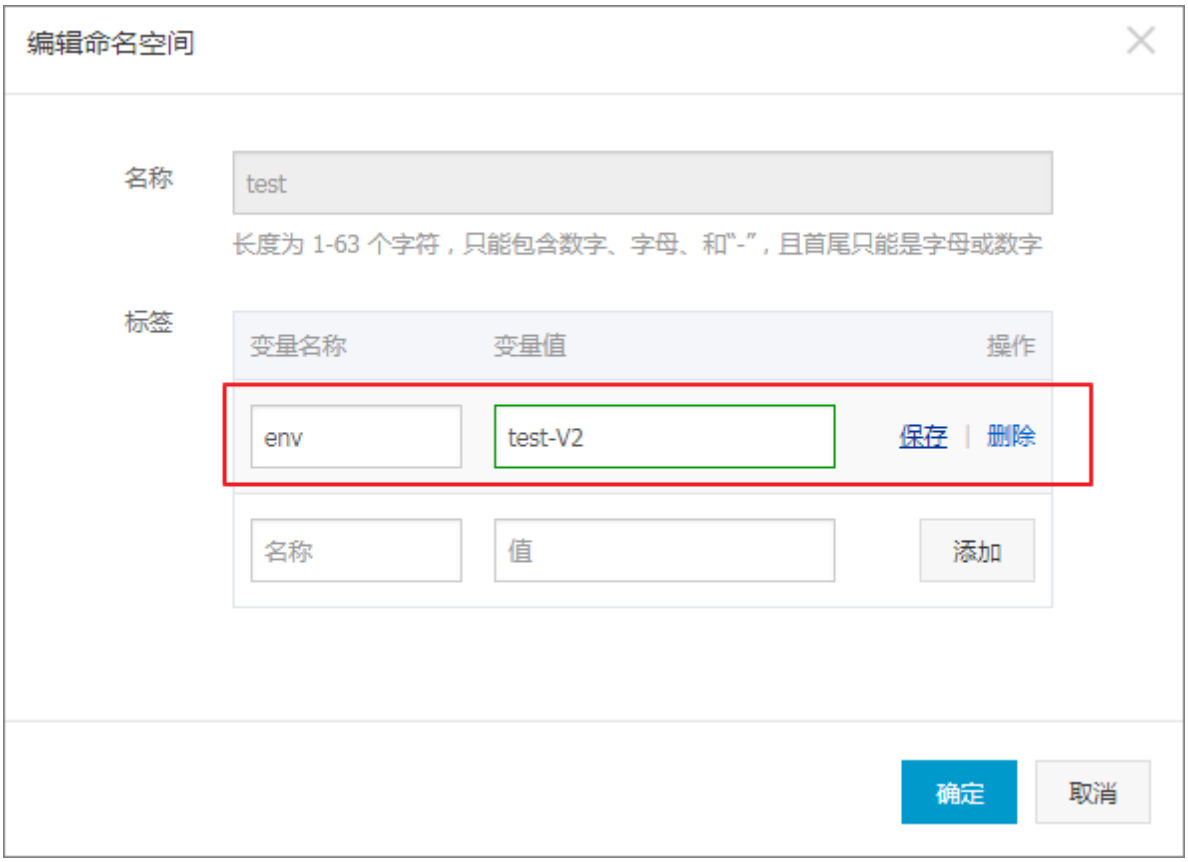
您可对命名空间进行编辑，对命名空间的标签进行增、删、改等操作。

操作步骤

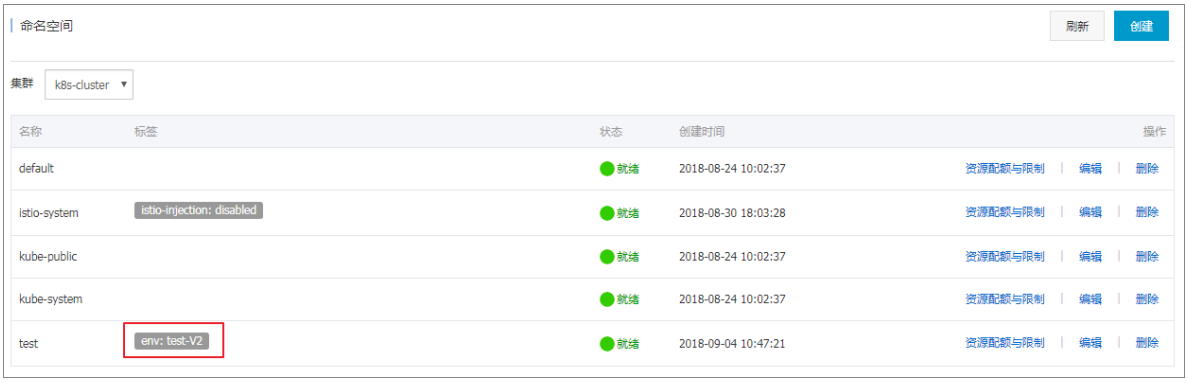
1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的**集群 > 命名空间**，进入命名空间列表页面。
3. 选择所需的集群，选择所需的命名空间，单击右侧的**编辑**。



4. 在弹出的对话框中，单击**编辑**，对命名空间的标签进行修改，如将标签修改为`env:test-V2`，然后单击**保存**。



5. 单击确定，返回命名空间列表，该命名空间的标签发生变化。



1.5.4 删除命名空间

您可删除不再需要的命名空间。

前提条件

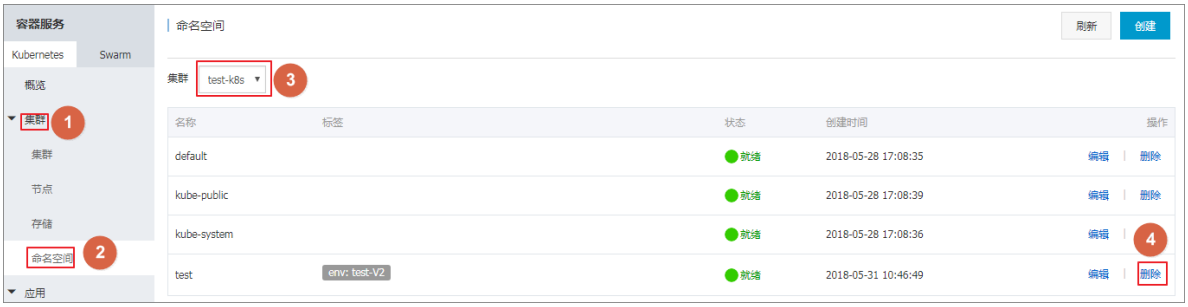
- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已成功创建一个示例的命名空间**test**，参见创建命名空间。

背景信息

> **说明：**
> 删除命名空间会导致其下所有的资源对象一起被删除，请慎重操作。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 命名空间，进入命名空间列表页面。
3. 选择所需的集群，选择所需的命名空间，单击右侧的删除。



4. 在弹出的对话框中，单击确定。

5. 返回命名空间列表，您可看到该命名空间已被删除，其下的资源对象也会被删除。

1.6 应用管理

1.6.1 使用镜像创建无状态Deployment应用

您可以使用镜像创建一个可公网访问的nginx应用。

前提条件

创建一个 Kubernetes 集群。详情请参见创建*Kubernetes*集群。

操作步骤

1. 登录容器服务管理控制台。

2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，然后单击页面右上角的使用镜像创建。
3. 设置应用名称、部署集群 和 命名空间、副本数量和类型，副本数量即应用包含的Pod数量。然后单击下一步 进入容器配置页面。

   说明：

   本例中选择无状态类型，即Deployment类型。

   如果您不设置命名空间，系统会默认使用 default 命名空间。

4. 设置容器配置。

   说明：

   您可为应用的Pod设置多个容器。

   a) 设置容器的基本配置。

   - 镜像名称：您可以单击选择镜像，在弹出的对话框中选择所需的镜像并单击确定，本例中为 nginx。

     您还可以填写私有 registry。填写的格式为`domainname/namespace/imagename:tag`
   - 镜像版本：您可以单击选择镜像版本 选择镜像的版本。若不指定，默认为 latest。
   - 总是拉取镜像：为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像 Tag 与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改 Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

- **资源限制**：可指定该应用所能使用的资源上限，包括 CPU 和 内存两种资源，防止占用过多资源。其中，CPU 资源的单位为 millicores，即一个核的千分之一；内存的单位为 Bytes，可以为 Gi、Mi 或 Ki。
- **所需资源**：即为该应用预留资源额度，包括 CPU 和 内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争抢资源，导致应用不可用。
- **Init Container**：勾选该项，表示创建一个Init Container，Init Container包含一些实用的工具，具体参见*https://kubernetes.io/docs/concepts/workloads/pods/init-containers/*。

b) （可选）配置数据卷信息。

支持配置本地存储和云存储。

- **本地存储**：支持主机目录（hostpath）、配置项（configmap）、保密字典（secret）和临时目录，将对应的挂载源挂载到容器路径中。更多信息参见 *volumes*。
- **云存储**：支持云盘/NAS/OSS三种云存储类型。

本例中配置了一个云盘类型的数据卷，将该云盘挂载到容器中*/tmp* 路径下，在该路径下生成的容器数据会存储到云盘中。

c) （可选）配置日志服务，您可进行采集配置和自定义Tag设置。

说明：

请确保已部署Kubernetes集群，并且在此集群上已安装日志插件。

您可对日志进行采集配置：

- 日志库：即在日志服务中生成一个对应的logstore，用于存储采集到的日志。
- 容器内日志路径：支持stdout和文本日志。
  - **stdout**： stdout 表示采集容器的标准输出日志。
  - 文本日志：表示收集容器内指定路径的日志，本例中表示收集/var/log/nginx下所有的文本日志，也支持通配符的方式。

您还可设置自定义 tag，设置tag后，会将该tag一起采集到容器的日志输出中。自定义 tag 可帮助您给容器日志打上tag，方便进行日志统计和过滤等分析操作。

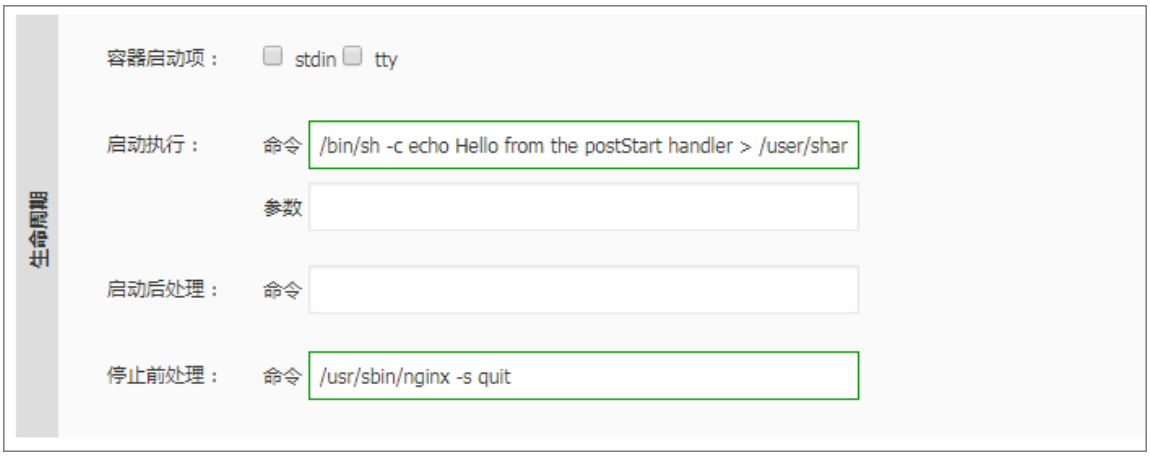
d) （可选）配置环境变量。

支持通过键值对的形式为 Pod 配置环境变量。用于给 Pod 添加环境标志或传递配置等，具体请参见 *Pod variable*。

e) 配置生命周期。

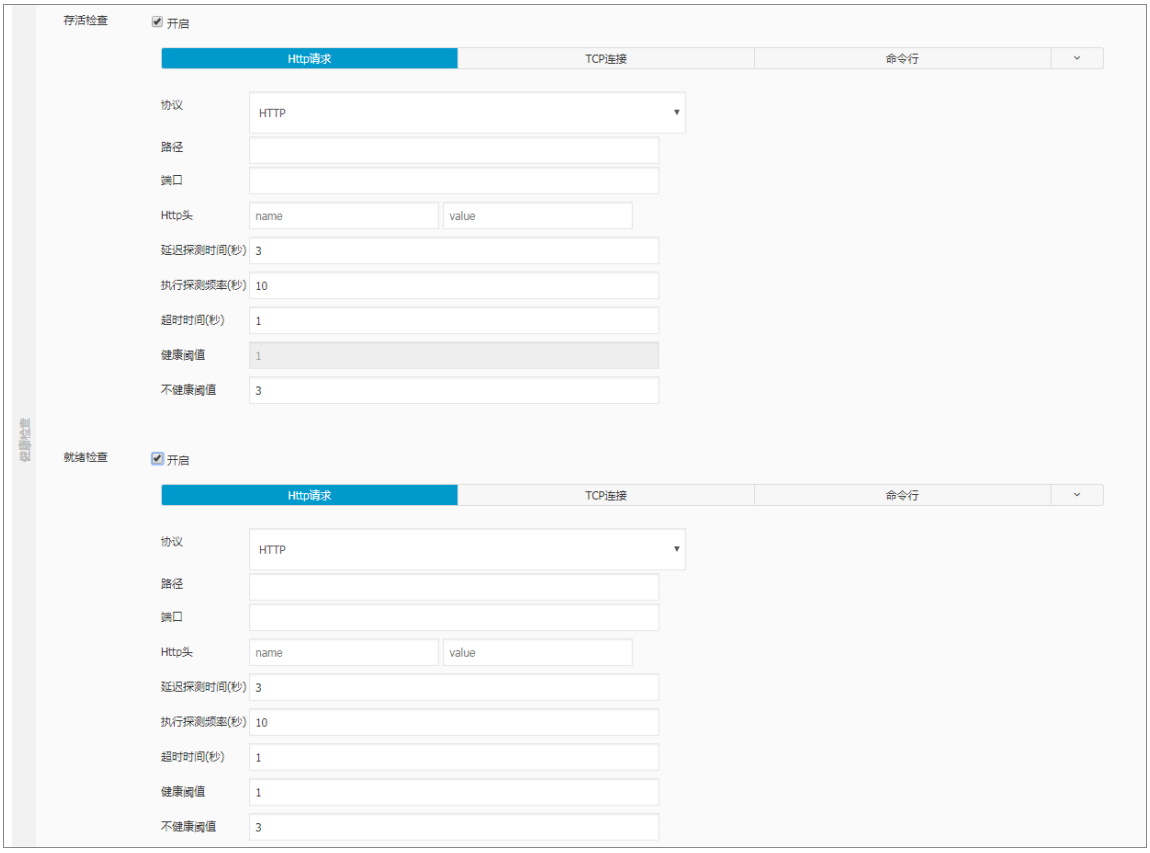
您可以为容器的生命周期配置容器启动项、启动执行、启动后处理和停止前处理。具体参见 *https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/*。

- 容器启动项：勾选 stdin 表示为该容器开启标准输入；勾选 tty 表示为该容器分配一个虚拟终端，以便于向容器发送信号。通常这两个选项是一起使用的，表示将终端 (tty) 绑定到容器的标准输入（stdin）上，比如一个交互式的程序从用户获取标准输入，并显示到终端中。
- 启动执行：为容器设置预启动命令和参数。
- 启动后处理：为容器设置启动后的命令。
- 停止前处理：为容器设置预结束命令。



f) （可选）设置健康检查

支持存活检查（liveness）和就绪检查（Readiness）。存活检查用于检测何时重启容器；就绪检查确定容器是否已经就绪，且可以接受流量。关于健康检查的更多信息，请参见*https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*。



请求类型	配置说明
HTTP请求	即向容器发送一个HTTPget 请求，支持的参数包括：

请求类型	配置说明
	• 协议：HTTP/HTTPS • 路径：访问HTTP server 的路径 • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • HTTP头：即HTTPHeaders，HTTP请求中自定义的请求头，HTTP允许重复的header。支持键值对的配置方式。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。
TCP连接	即向容器发送一个TCP Socket，kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。

请求类型	配置说明
	• 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。
命令行	通过在容器中执行探针检测命令，来检测容器的健康情况。支持的参数包括： • 命令行：用于检测容器健康情况的探测命令。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为5秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。

5. 完成容器配置后，单击 下一步。
6. 进行高级设置。
   a) 设置访问设置。

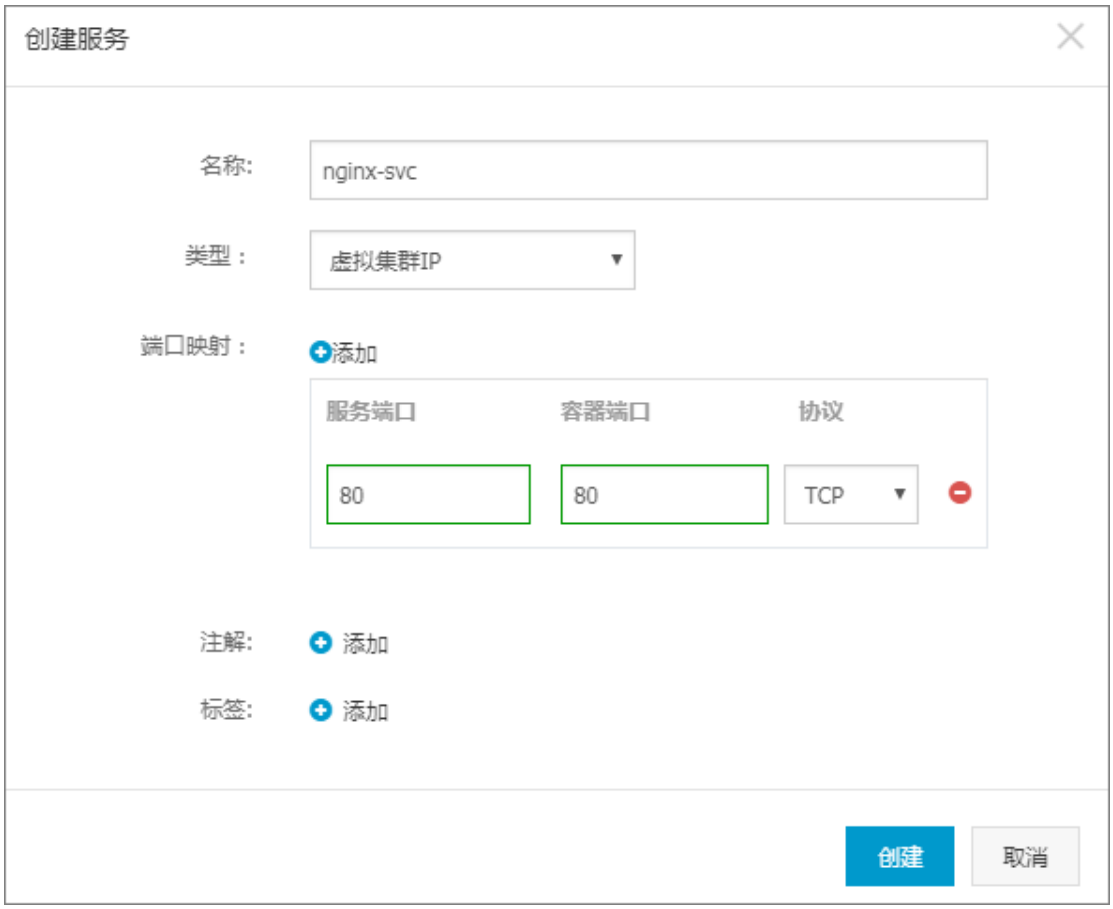
   您可以设置暴露后端Pod的方式，最后单击创建。本例中选择ClusterIP服务和路由（Ingress），构建一个可公网访问的nginx应用。

   说明：

   针对应用的通信需求，您可灵活进行访问设置：

- 内部应用：对于只在集群内部工作的应用，您可根据需要创建ClusterIP或NodePort类型的服务，来进行内部通信。
- 外部应用：对于需要暴露到公网的应用，您可以采用两种方式进行访问设置：
  - 创建LoadBalancer类型的服务：使用阿里云提供的负载均衡服务（Server Load Balancer，SLB），该服务提供公网访问能力。
  - 创建ClusterIP、NodePort类型的服务，以及路由（Ingress）：通过路由提供公网访问能力，详情参见*https://kubernetes.io/docs/concepts/services-networking/ingress/*。

1. 在服务栏单击创建，在弹出的对话框中进行配置，最后单击创建。

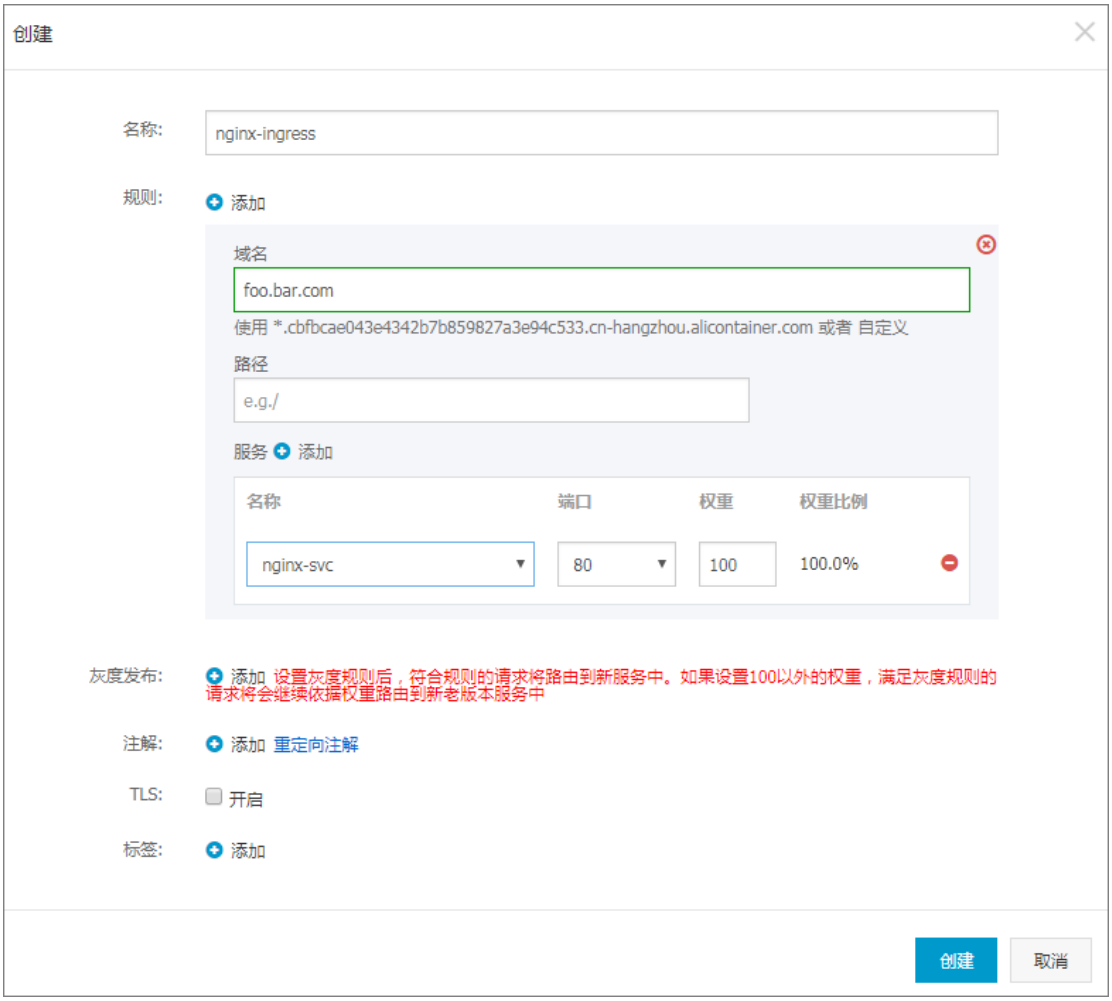


- 名称：您可自主设置，默认为`applicationname-svc`。
- 类型：您可以从下面 3 种服务类型中进行选择。
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该项，服务只能够在集群内部可以访问。
  - 节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
  - 负载均衡：即 LoadBalancer，是阿里云提供的负载均衡服务，可选择公网访问或内网访问。负载均衡可以路由到 NodePort 服务和 ClusterIP 服务。
- 端口映射：您需要添加服务端口和容器端口，若类型选择为节点端口，还需要自己设置节点端口，防止端口出现冲突。支持 TCP/UDP 协议。
- 注解：为该服务添加一个注解（annotation），支持负载均衡配置参数，参见通过负载均衡#Server Load Balancer#访问服务。
- 标签：您可为该服务添加一个标签，标识该服务。

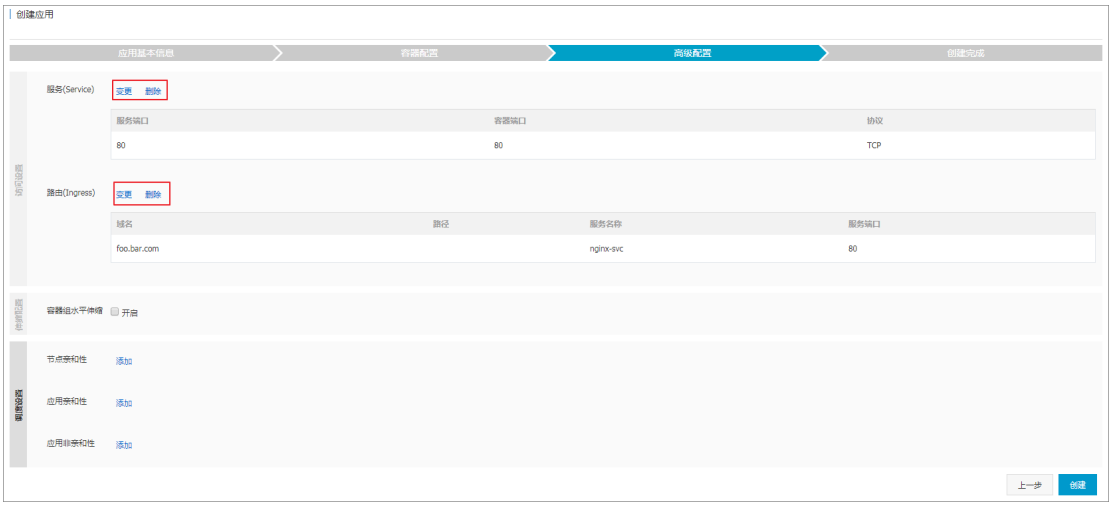
2. 在路由栏单击创建，在弹出的对话框中，为后端Pod配置路由规则，最后单击创建。更多详细的路由配置信息，请参见路由配置说明。

> **说明：**
> 通过镜像创建应用时，您仅能为一个服务创建路由（Ingress）。本例中使用一个虚拟主机名称作为测试域名，您需要在hosts中添加一条记录。在实际工作场景中，请使用备案域名。

```
101.37.224.146   foo.bar.com    #即ingress的IP
```



3. 在访问设置栏中，您可看到创建完毕的服务和路由，您可单击变更和删除进行二次配置。



b) （可选）容器组水平伸缩。

您可勾选是否开启**容器组水平伸缩**，为了满足应用在不同负载下的需求，容器服务支持服容器组（Pod）的弹性伸缩，即根据容器 CPU 和内存资源占用情况自动调整容器组数量。

说明：

若要启用自动伸缩，您必须为容器设置所需资源，否则容器自动伸缩无法生效。参见容器基本配置环节。

- **指标**：支持CPU和内存，需要和设置的所需资源类型相同。
- **触发条件**：资源使用率的百分比，超过该使用量，容器开始扩容。
- **最大容器数量**：该Deployment可扩容的容器数量上限。
- **最小容器数量**：该Deployment可缩容的容器数量下限。

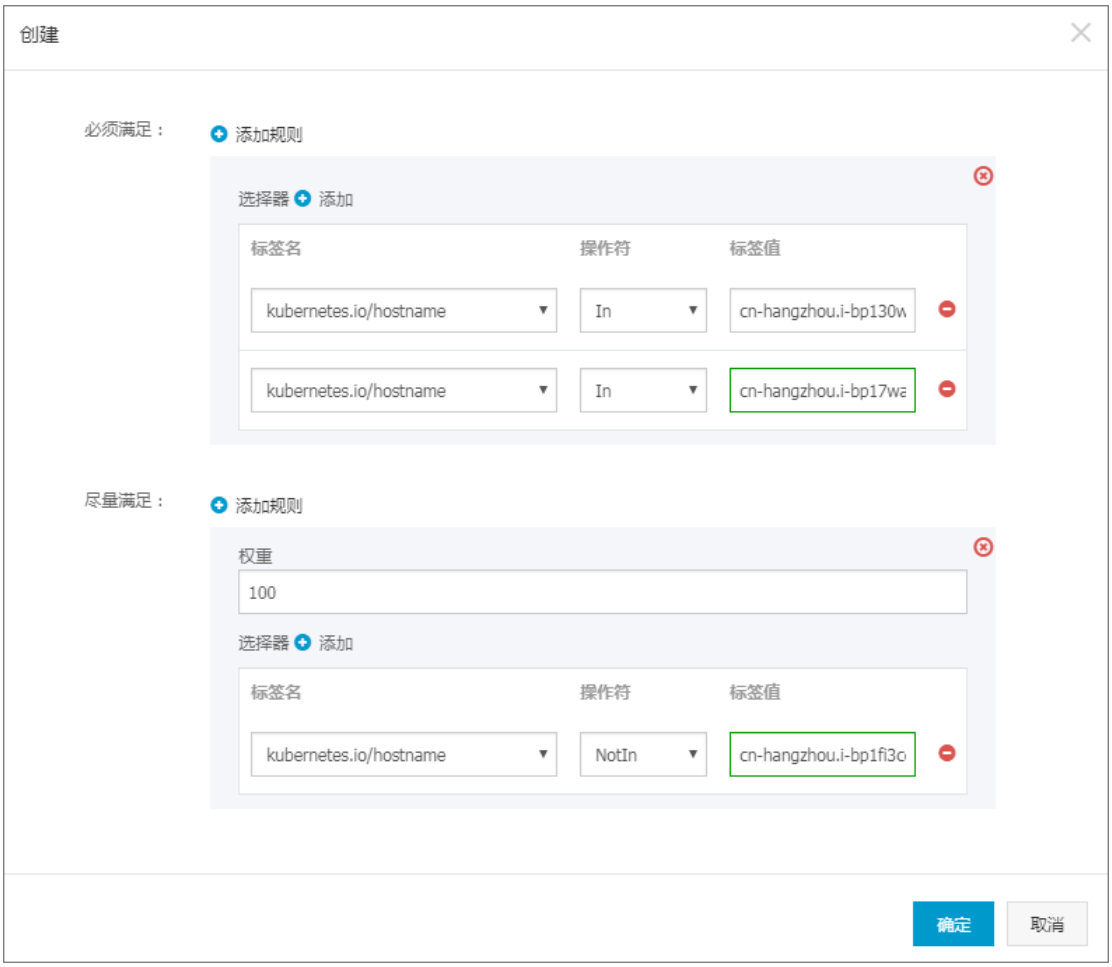
c) （可选）设置调度亲和性。

您可设置节点亲和性、应用亲和性和应用非亲和性，详情参见*https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity*。

说明：

亲和性调度依赖节点标签和Pod标签，您可使用内置的标签进行调度；也可预先为节点、Pod配置相关的标签。

1. 设置节点亲和性，通过Node节点的Label标签进行设置。

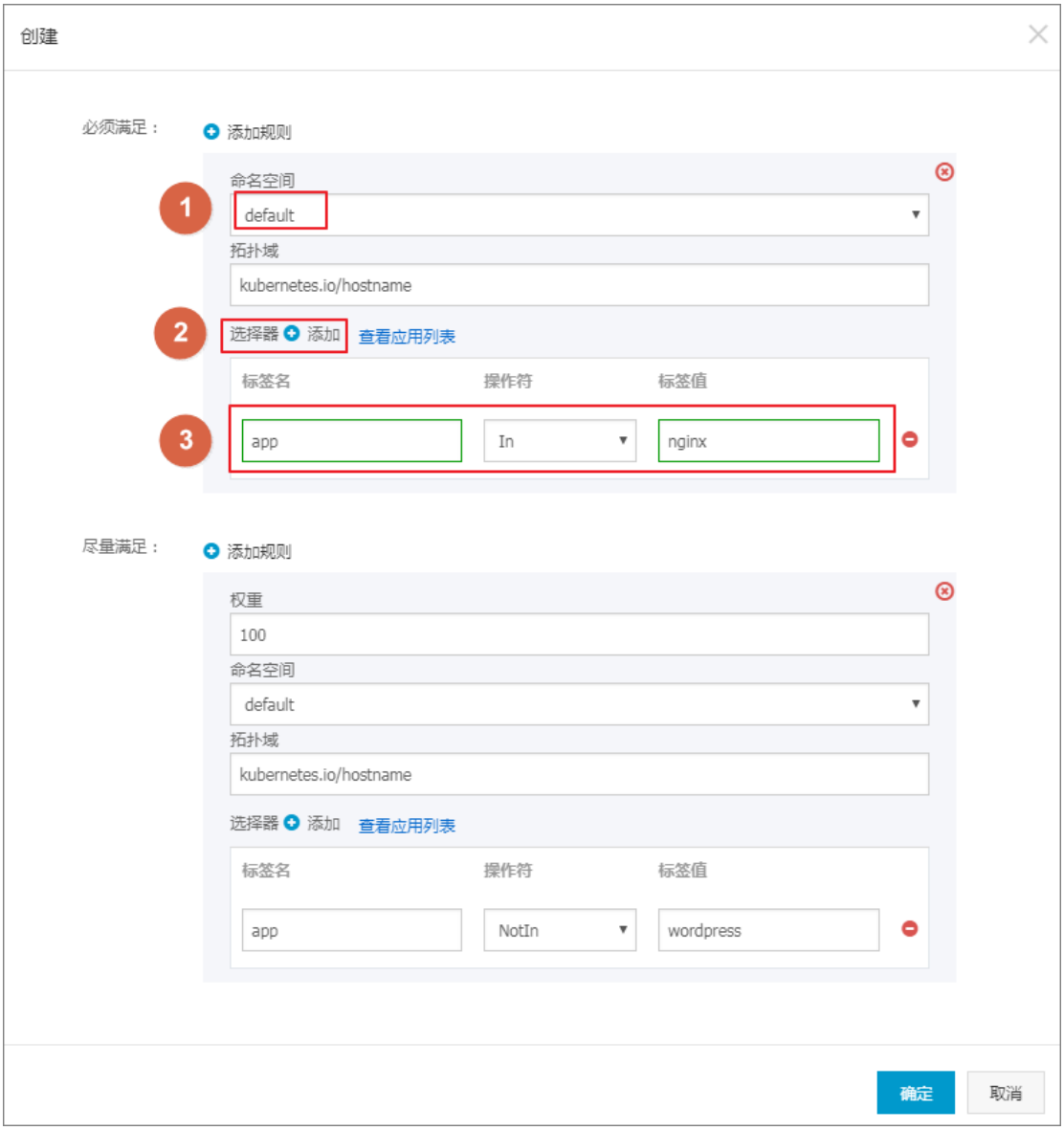


节点调度支持硬约束和软约束（Required/Preferred），以及丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- 必须满足，即硬约束，一定要满足，对应**requiredDuringSchedulingIgnoredDuringExecution**，效果与`NodeSelector`相同。本例中Pod只能调度到具有对应标签的Node节点。您可以定义多条硬约束规则，但只需满足其中一条。
- 尽量满足，即软约束，不一定满足，对应**preferredDuringSchedulingIgnoredDuringExecution**。本例中，调度会尽量不调度Pod到具有对应标签的Node节点。您还可为软约束规则设定权重，具体调度时，若存在多个符合条件的节点，权重最

高的节点会被优先调度。您可定义多条软约束规则，但必须满足全部约束，才会进行调度。

2. 设置应用亲和性调度。决定应用的Pod可以和哪些Pod部署在同一拓扑域。例如，对于相互通信的服务，可通过应用亲和性调度，将其部署到同一拓扑域（如同一个主机）中，减少它们之间的网络延迟。



根据节点上运行的Pod的标签（Label）来进行调度，支持硬约束和软约束，匹配的表达式有：`In, NotIn, Exists, DoesNotExist`。

- 必须满足，即硬约束，一定要满足，对应**`requiredDuringSchedulingIgnoredDuringExecution`**，Pod的亲和性调度必须要满足后续定义的约束条件。

  — 命名空间：该策略是依据Pod的Label进行调度，所以会受到命名空间的约束。

- **拓扑域**：即topologyKey，指定调度时作用域，这是通过Node节点的标签来实现的，例如指定为`kubernetes.io/hostname`，那就是以Node节点为区分范围；如果指定为`beta.kubernetes.io/os`，则以Node节点的操作系统类型来区分。
- **选择器**：单击选择器右侧的加号按钮，您可添加多条硬约束规则。
- **查看应用列表**：单击**应用列表**，弹出对话框，您可在此查看各命名空间下的应用，并可将应用的标签导入到亲和性配置页面。
- 硬约束条件：设置已有应用的标签、操作符和标签值。本例中，表示将待创建的应用调度到该主机上，该主机运行的已有应用具有`app:nginx`标签。

- **尽量满足**，即软约束，不一定满足，对应**`preferredDuringSchedulingIgnoredDuringExecution`**。Pod的亲和性调度会尽量满足后续定义的约束条件。对于软约束规则，您可配置每条规则的权重，其他配置规则与硬约束规则相同。

  说明：
  权重：设置一条软约束规则的权重，介于1-100，通过算法计算满足软约束规则的节点的权重，将Pod调度到权重最高的节点上。

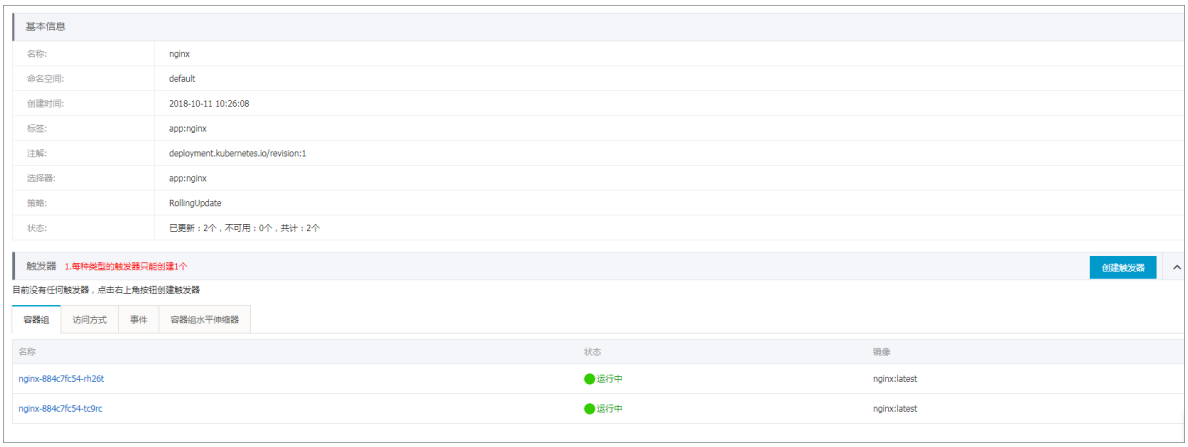
3. 设置**应用非亲和性调度**，决定应用的Pod不与哪些Pod部署在同一拓扑域。应用非亲和性调度的场景包括：

   - 将一个服务的Pod分散部署到不同的拓扑域（如不同主机）中，提高服务本身的稳定性。
   - 给予Pod一个节点的独占访问权限来保证资源隔离，保证不会有其它Pod来分享节点资源。
   - 把可能会相互影响的服务的Pod分散在不同的主机上。

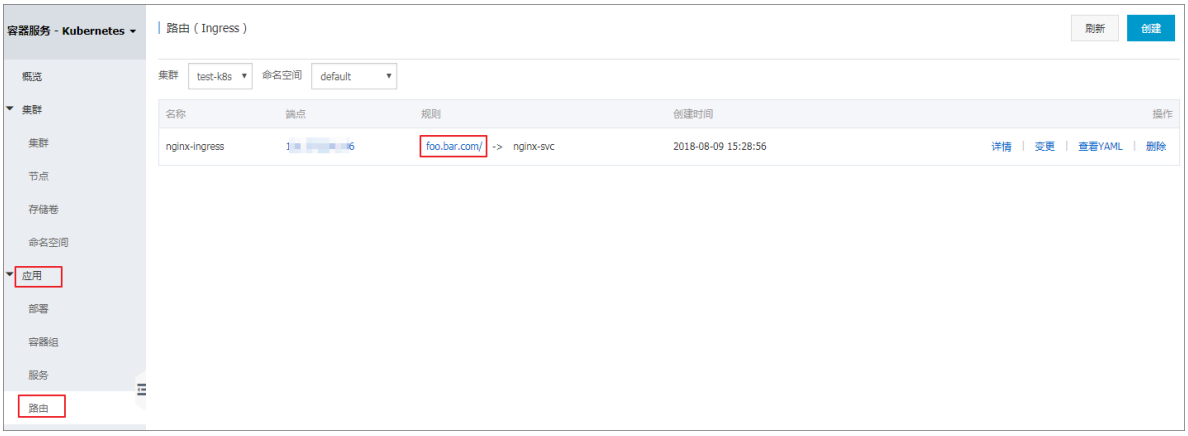
   说明：
   应用非亲和性调度的设置方式与亲和性调度相同，但是相同的调度规则代表的意思不同，请根据使用场景进行选择。

7. 最后单击**创建**。
8. 创建成功后，默认进入创建完成页面，会列出应用包含的对象，您可以单击**查看应用详情**进行查看。

默认进入新建的nginx-deployment的详情页面。



9. 单击左侧导航栏的应用 > 路由，可以看到路由列表下出现一条规则。



10. 在浏览器中访问路由测试域名，您可访问 nginx 欢迎页。

1.6.2 使用镜像创建有状态StatefulSet应用

阿里云容器服务Kubernetes集群支持通过界面创建StatefultSet类型的应用，满足您快速创建有状态应用的需求。本例中将创建一个nginx的有状态应用，并演示StatefulSet应用的特性。

前提条件

- 您已成功创建一个 Kubernetes 集群。参见创建*Kubernetes*集群。
- 您已成功创建一个云盘存储卷声明，参见创建持久化存储卷声明。
- 您已连接到Kubernetes集群的Master节点，参见通过 *kubectl* 连接 *Kubernetes* 集群。

背景信息

StatefulSet包括如下特性：

场景	说明
Pod一致性	包含次序（启动、停止次序）、网络一致性。此一致性与Pod相关，与被调度到哪个node节点无关。
稳定的持久化存储	通过VolumeClaimTemplate为每个Pod创建一个PV。删除、减少副本，不会删除相关的卷。
稳定的网络标志	Pod的hostname模式为：`(statefulset名称)-(序号)`。
稳定的次序	对于N个副本的StatefulSet，每个Pod都在[0，N)的范围内分配一个数字序号，且是唯一的。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，然后单击页面右上角的使用镜像创建。

3. 在应用基本信息页面进行设置，然后单击下一步 进入应用配置页面。

   - 应用名称：设置应用的名称。
   - 部署集群：设置应用部署的集群。
   - 命名空间：设置应用部署所处的命名空间，默认使用default命名空间。
   - 副本数量：即应用包含的Pod数量。
   - 类型：可选择无状态（Deployment）和有状态（StatefulSet）两种类型。

   > 说明：
   > 本例中选择有状态类型，创建StatefulSet类型的应用。

4. 设置容器配置。

   > 说明：
   > 您可为应用的Pod设置多个容器。

   a) 设置容器的基本配置。

   - 镜像名称：您可以单击选择镜像，在弹出的对话框中选择所需的镜像并单击确定，本例中为 nginx。

     您还可以填写私有 registry。填写的格式为`domainname/namespace/imagename:tag`
   - 镜像版本：您可以单击选择镜像版本 选择镜像的版本。若不指定，默认为 latest。
   - 总是拉取镜像：为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像 Tag 与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改 Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。

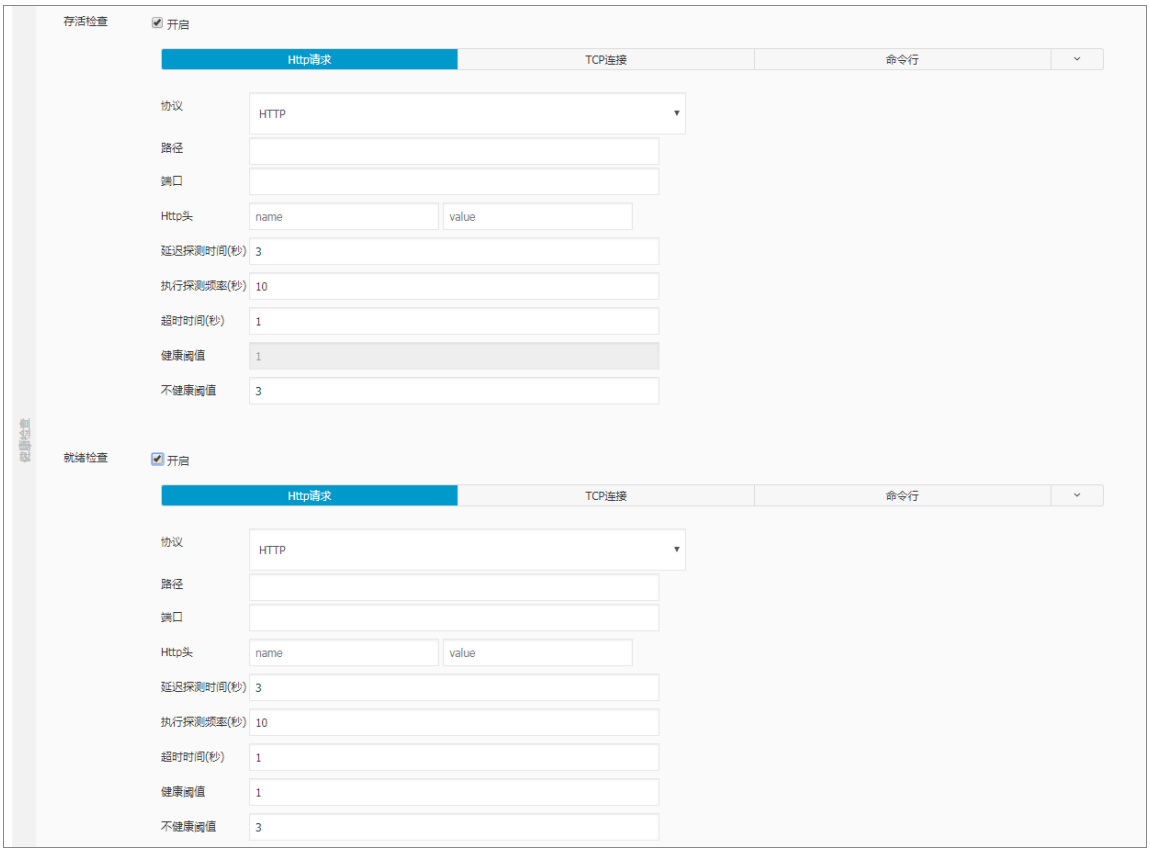
- **资源限制**：可指定该应用所能使用的资源上限，包括 CPU 和 内存两种资源，防止占用过多资源。其中，CPU 资源的单位为 millicores，即一个核的千分之一；内存的单位为 Bytes，可以为 Gi、Mi 或 Ki。
- **所需资源**：即为该应用预留资源额度，包括 CPU 和 内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争抢资源，导致应用不可用。
- **Init Container**：勾选该项，表示创建一个Init Container，Init Container包含一些实用的工具，具体参见*https://kubernetes.io/docs/concepts/workloads/pods/init-containers/*。

b) （可选）配置环境变量。

支持通过键值对的形式为 Pod 配置环境变量。用于给 Pod 添加环境标志或传递配置等，具体请参见 *Pod variable*。

c) （可选）配置健康检查。

支持存活检查（liveness）和就绪检查（Readiness）。存活检查用于检测何时重启容器；就绪检查确定容器是否已经就绪，且可以接受流量。关于健康检查的更多信息，请参见*https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*。



请求类型	配置说明
HTTP请求	即向容器发送一个HTTPget 请求，支持的参数包括： • 协议：HTTP/HTTPS • 路径：访问HTTP server 的路径 • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • HTTP头：即HTTPHeaders，HTTP请求中自定义的请求头，HTTP允许重复的header。支持键值对的配置方式。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。

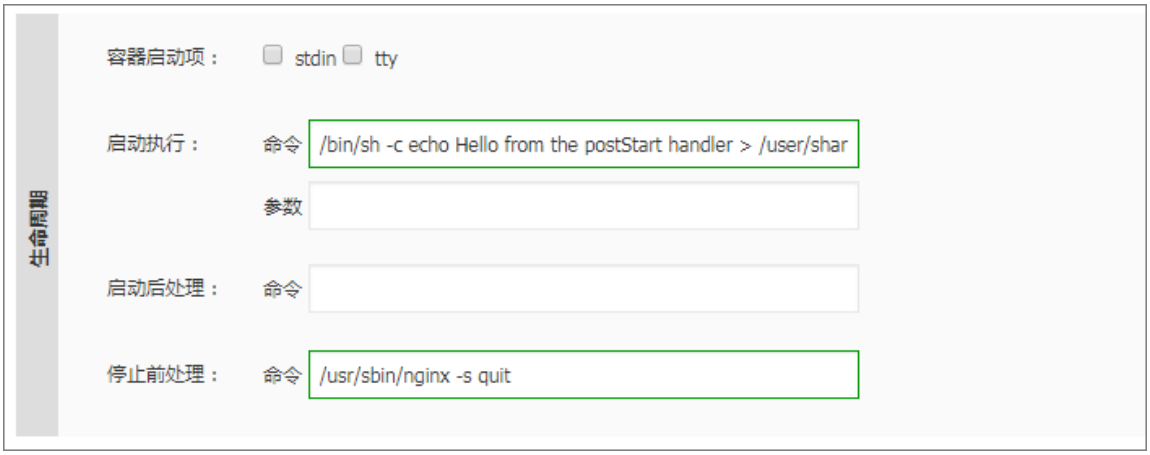
请求类型	配置说明
	• 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。
TCP连接	即向容器发送一个TCP Socket，kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。
命令行	通过在容器中执行探针检测命令，来检测容器的健康情况。支持的参数包括： • 命令行：用于检测容器健康情况的探测命令。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为5秒。

请求类型	配置说明
	• 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。

d) （可选）配置生命周期。

您可以为容器的生命周期配置容器启动项、启动执行、启动后处理和停止前处理。具体参见 *https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/*。

- **容器启动项**：勾选 stdin 表示为该容器开启标准输入；勾选 tty 表示为该容器分配一个虚拟终端，以便于向容器发送信号。通常这两个选项是一起使用的，表示将终端 (tty) 绑定到容器的标准输入（stdin）上，比如一个交互式的程序从用户获取标准输入，并显示到终端中。
- **启动执行**：为容器设置预启动命令和参数。
- **启动后处理**：为容器设置启动后的命令。
- **停止前处理**：为容器设置预结束命令。



e) 配置数据卷信息。

支持配置本地存储和云存储。

- **本地存储**：支持主机目录（hostpath）、配置项（configmap）、保密字典（secret）和临时目录，将对应的挂载源挂载到容器路径中。更多信息参见 *volumes*。
- **云存储**：支持云盘/NAS/OSS三种云存储类型。

本例中配置了一个云盘类型的数据卷声明disk-ssd，将其挂载到容器的`/data`路径下。

f) （可选）配置**日志服务**，您可进行采集配置和自定义Tag设置。

> **说明：**
> 请确保已部署Kubernetes集群，并且在此集群上已安装日志插件。

您可对日志进行采集配置：

- **日志库**：即在日志服务中生成一个对应的logstore，用于存储采集到的日志。
- **容器内日志路径**：支持stdout和文本日志。
  - **stdout**： stdout 表示采集容器的标准输出日志。
  - **文本日志**：表示收集容器内指定路径的日志，本例中表示收集/var/log/nginx下所有的文本日志，也支持通配符的方式。

您还可设置自定义 tag，设置tag后，会将该tag一起采集到容器的日志输出中。自定义 tag 可帮助您给容器日志打上tag，方便进行日志统计和过滤等分析操作。

5. 完成容器配置后，单击 下一步。

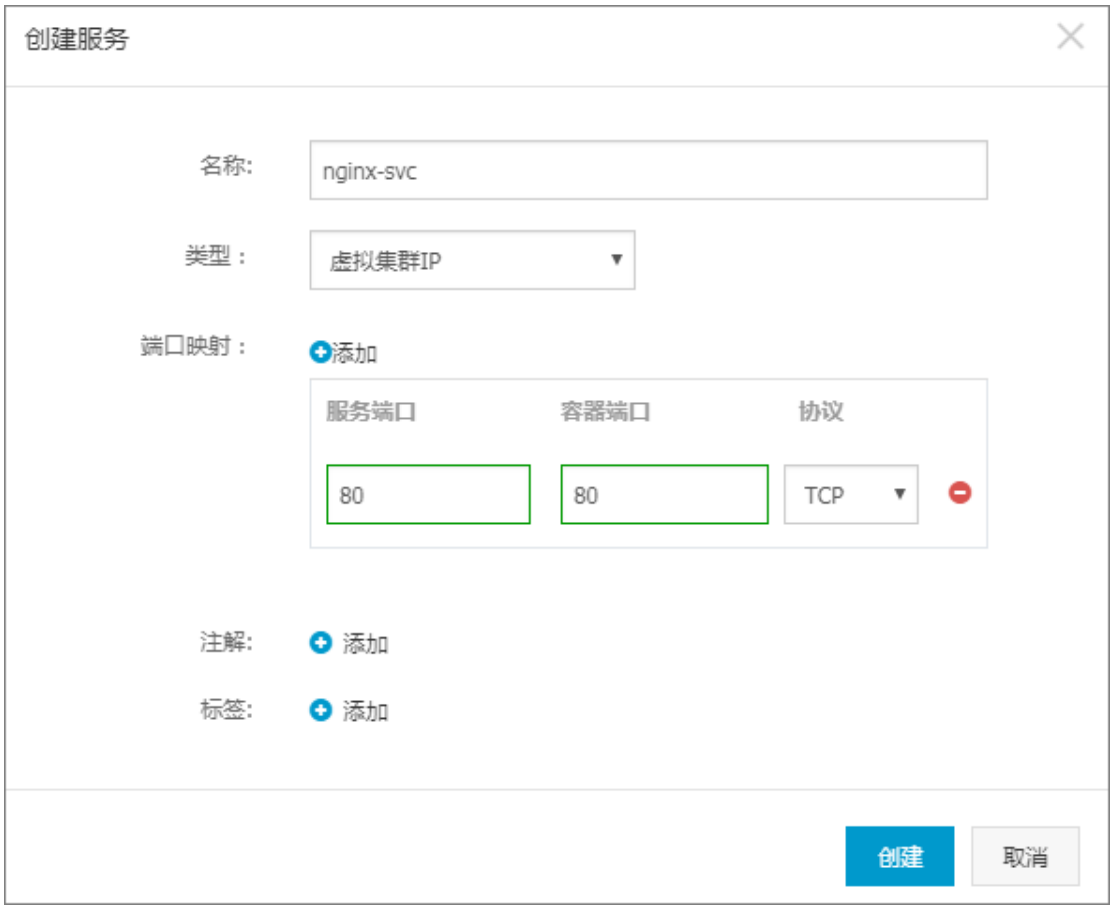
6. 进行高级设置。本例中仅进行访问设置。

a) 设置访问设置。

您可以设置暴露后端Pod的方式，最后单击创建。本例中选择ClusterIP服务和路由（Ingress），构建一个可公网访问的nginx应用。

> 说明：
>
> 针对应用的通信需求，您可灵活进行访问设置：
>
> - 内部应用：对于只在集群内部工作的应用，您可根据需要创建ClusterIP或NodePort类型的服务，来进行内部通信。
> - 外部应用：对于需要暴露到公网的应用，您可以采用两种方式进行访问设置：
>   - 创建LoadBalancer类型的服务：使用阿里云提供的负载均衡服务（Server Load Balancer，SLB），该服务提供公网访问能力。
>   - 创建ClusterIP、NodePort类型的服务，以及路由（Ingress）：通过路由提供公网访问能力，详情参见*https://kubernetes.io/docs/concepts/services-networking/ingress/*。

1. 在服务栏单击创建，在弹出的对话框中进行配置，最后单击创建。

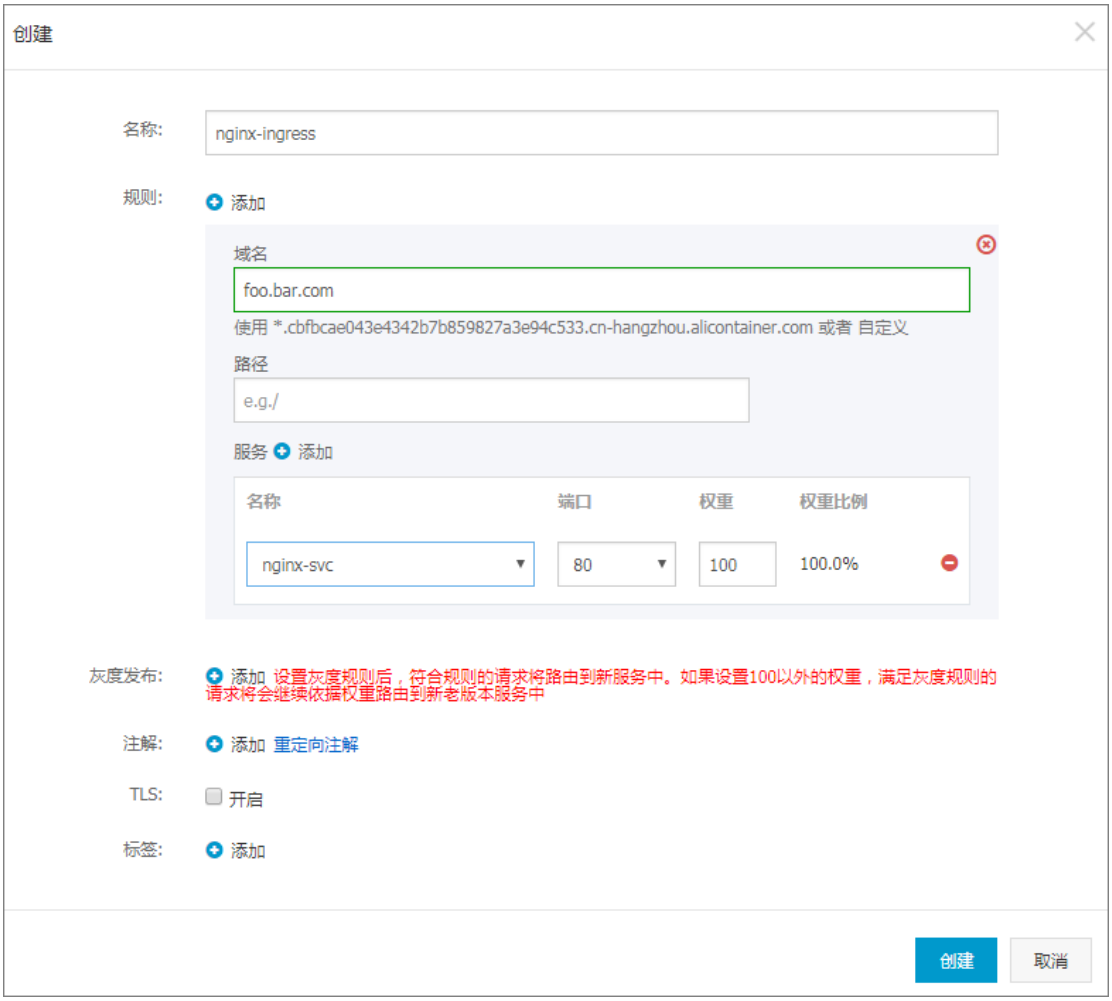


- 名称：您可自主设置，默认为`applicationname-svc`。
- 类型：您可以从下面 3 种服务类型中进行选择。
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该项，服务只能够在集群内部可以访问。
  - 节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
  - 负载均衡：即 LoadBalancer，是阿里云提供的负载均衡服务，可选择公网访问或内网访问。负载均衡可以路由到 NodePort 服务和 ClusterIP 服务。
- 端口映射：您需要添加服务端口和容器端口，若类型选择为节点端口，还需要自己设置节点端口，防止端口出现冲突。支持 TCP/UDP 协议。
- 注解：为该服务添加一个注解（annotation），支持负载均衡配置参数，参见通过负载均衡#Server Load Balancer#访问服务。
- 标签：您可为该服务添加一个标签，标识该服务。

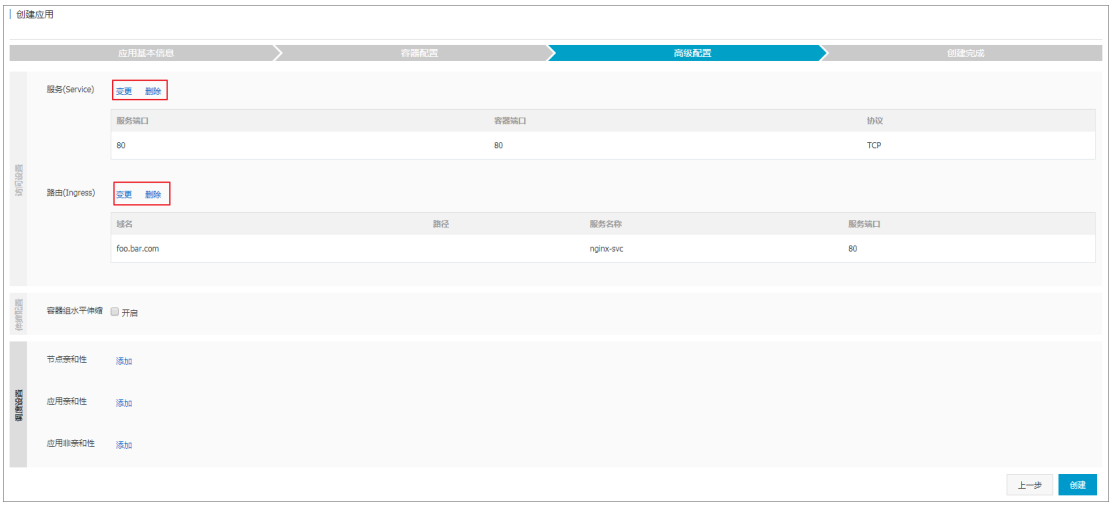
2. 在路由栏单击创建，在弹出的对话框中，为后端Pod配置路由规则，最后单击创建。更多详细的路由配置信息，请参见路由配置说明。

> **说明：**
> 通过镜像创建应用时，您仅能为一个服务创建路由（Ingress）。本例中使用一个虚拟主机名称作为测试域名，您需要在hosts中添加一条记录。在实际工作场景中，请使用备案域名。

```
101.37.224.146   foo.bar.com    #即ingress的IP
```



3. 在访问设置栏中，您可看到创建完毕的服务和路由，您可单击变更和删除进行二次配置。



b) （可选）容器组水平伸缩。

您可勾选是否开启**容器组水平伸缩**，为了满足应用在不同负载下的需求，容器服务支持服容器组（Pod）的弹性伸缩，即根据容器 CPU 和内存资源占用情况自动调整容器组数量。

说明：

若要启用自动伸缩，您必须为容器设置所需资源，否则容器自动伸缩无法生效。参见容器基本配置环节。

- **指标**：支持CPU和内存，需要和设置的所需资源类型相同。
- **触发条件**：资源使用率的百分比，超过该使用量，容器开始扩容。
- **最大容器数量**：该Deployment可扩容的容器数量上限。
- **最小容器数量**：该Deployment可缩容的容器数量下限。

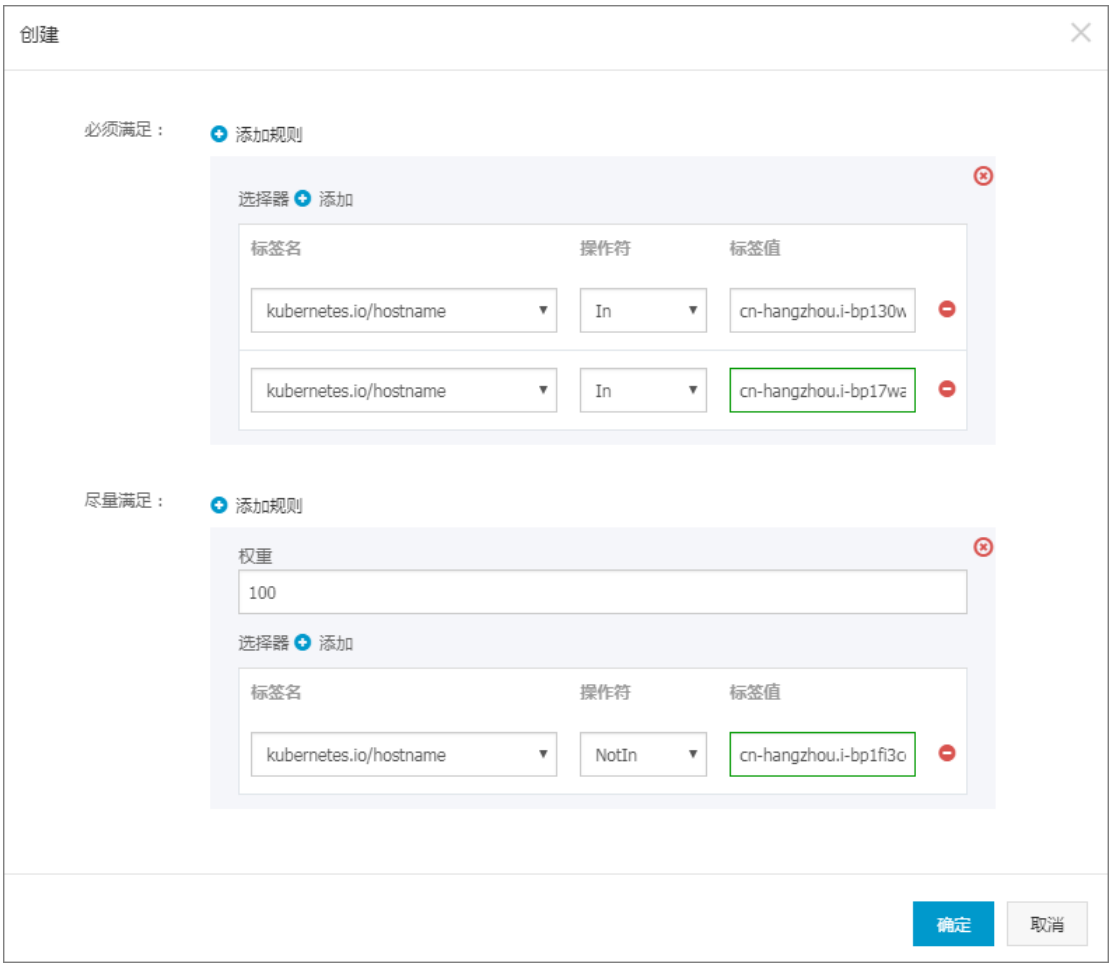
c) （可选）设置调度亲和性。

您可设置节点亲和性、应用亲和性和应用非亲和性，详情参见*https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity*。

说明：

亲和性调度依赖节点标签和Pod标签，您可使用内置的标签进行调度；也可预先为节点、Pod配置相关的标签。

1. 设置节点亲和性，通过Node节点的Label标签进行设置。

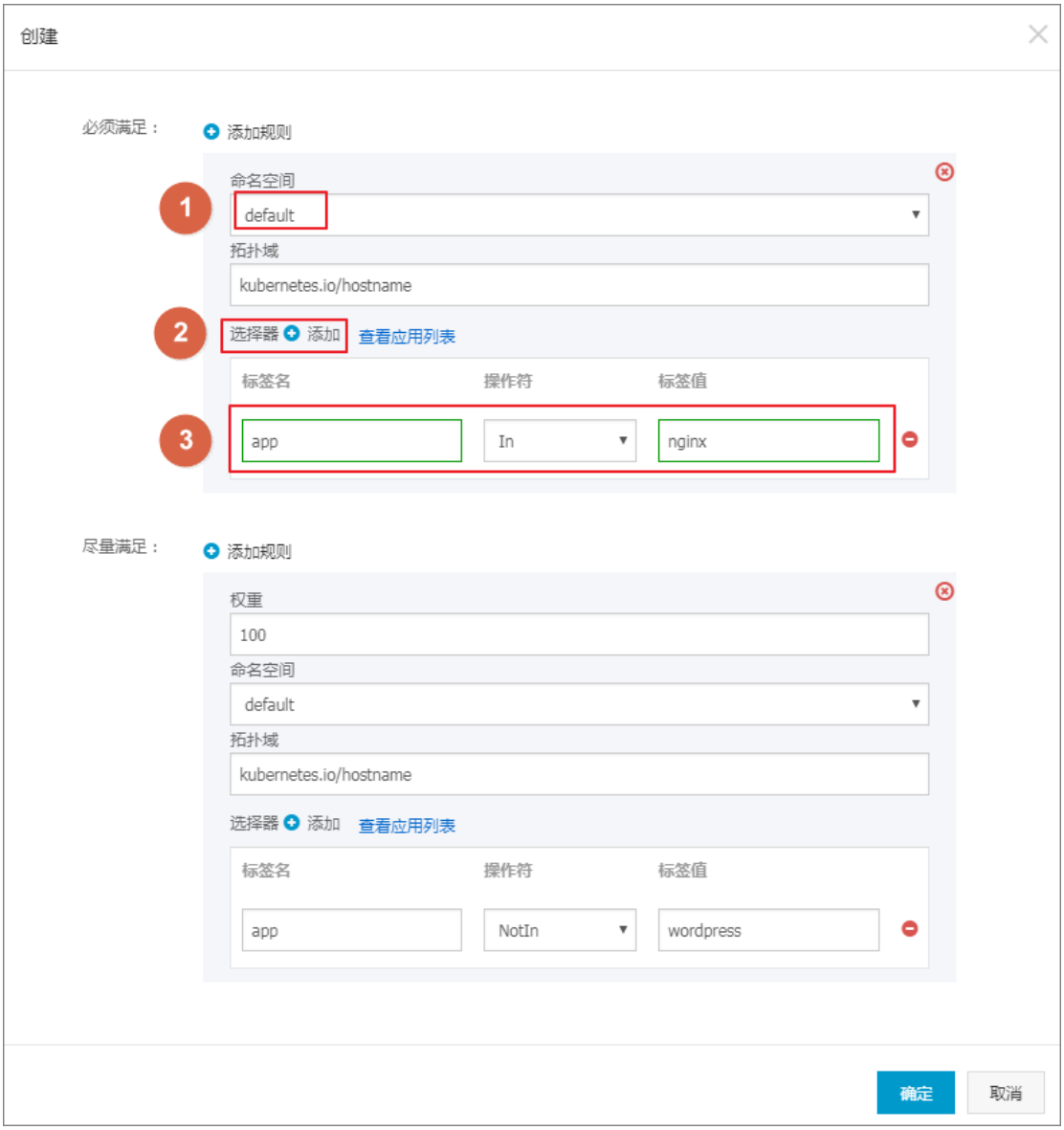


节点调度支持硬约束和软约束（Required/Preferred），以及丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- 必须满足，即硬约束，一定要满足，对应**requiredDuringSchedulingIgnoredDuringExecution**，效果与`NodeSelector`相同。本例中Pod只能调度到具有对应标签的Node节点。您可以定义多条硬约束规则，但只需满足其中一条。
- 尽量满足，即软约束，不一定满足，对应**preferredDuringSchedulingIgnoredDuringExecution**。本例中，调度会尽量不调度Pod到具有对应标签的Node节点。您还可为软约束规则设定权重，具体调度时，若存在多个符合条件的节点，权重最

高的节点会被优先调度。您可定义多条软约束规则，但必须满足全部约束，才会进行调度。

2. 设置应用亲和性调度。决定应用的Pod可以和哪些Pod部署在同一拓扑域。例如，对于相互通信的服务，可通过应用亲和性调度，将其部署到同一拓扑域（如同一个主机）中，减少它们之间的网络延迟。



根据节点上运行的Pod的标签（Label）来进行调度，支持硬约束和软约束，匹配的表达式有：`In, NotIn, Exists, DoesNotExist`。

- 必须满足，即硬约束，一定要满足，对应**`requiredDuringSchedulingIgnoredDuringExecution`**，Pod的亲和性调度必须要满足后续定义的约束条件。

  — 命名空间：该策略是依据Pod的Label进行调度，所以会受到命名空间的约束。

- **拓扑域**：即topologyKey，指定调度时作用域，这是通过Node节点的标签来实现的，例如指定为`kubernetes.io/hostname`，那就是以Node节点为区分范围；如果指定为`beta.kubernetes.io/os`，则以Node节点的操作系统类型来区分。
- **选择器**：单击选择器右侧的加号按钮，您可添加多条硬约束规则。
- **查看应用列表**：单击**应用列表**，弹出对话框，您可在此查看各命名空间下的应用，并可将应用的标签导入到亲和性配置页面。
- 硬约束条件：设置已有应用的标签、操作符和标签值。本例中，表示将待创建的应用调度到该主机上，该主机运行的已有应用具有`app:nginx`标签。

- **尽量满足**，即软约束，不一定满足，对应**`preferredDuringSchedulingIgnoredDuringExecution`**。Pod的亲和性调度会尽量满足后续定义的约束条件。对于软约束规则，您可配置每条规则的权重，其他配置规则与硬约束规则相同。

  说明：
  权重：设置一条软约束规则的权重，介于1-100，通过算法计算满足软约束规则的节点的权重，将Pod调度到权重最高的节点上。

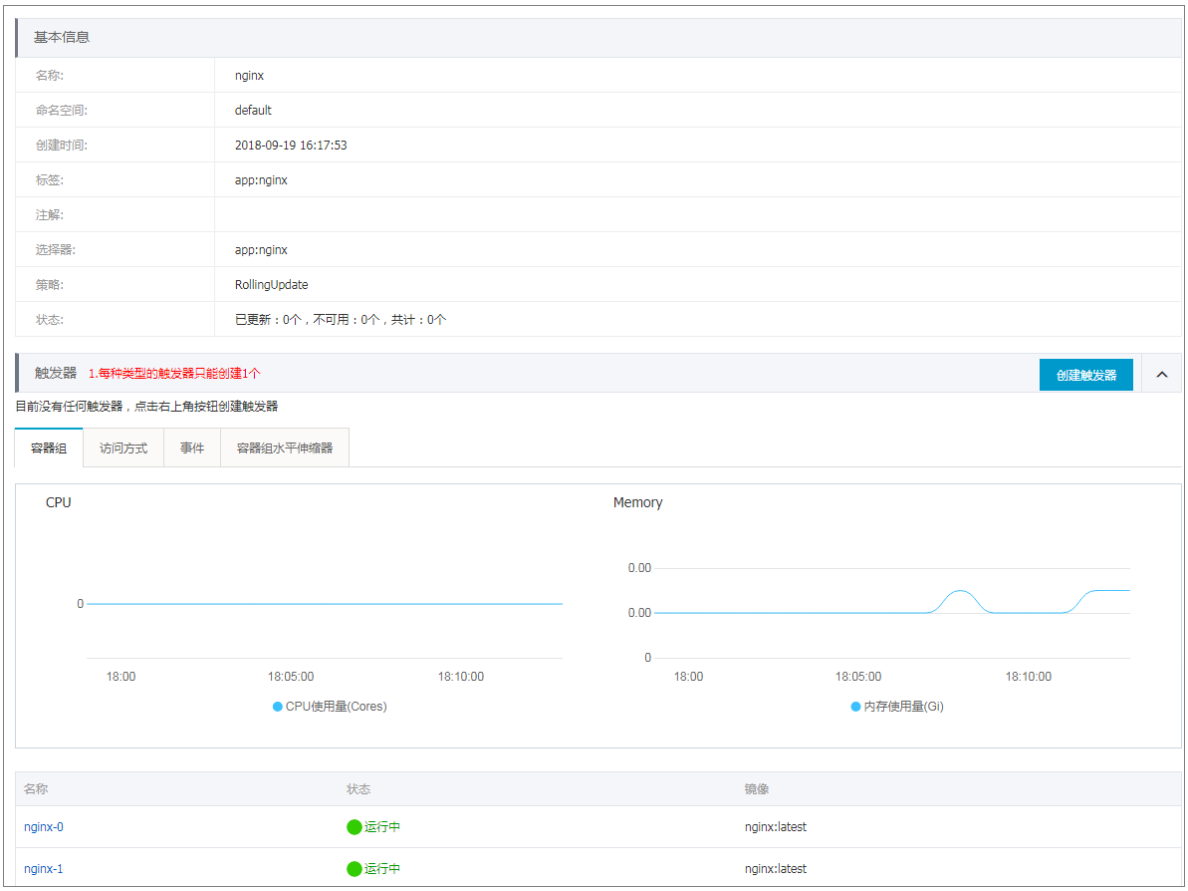
3. 设置**应用非亲和性调度**，决定应用的Pod不与哪些Pod部署在同一拓扑域。应用非亲和性调度的场景包括：

   - 将一个服务的Pod分散部署到不同的拓扑域（如不同主机）中，提高服务本身的稳定性。
   - 给予Pod一个节点的独占访问权限来保证资源隔离，保证不会有其它Pod来分享节点资源。
   - 把可能会相互影响的服务的Pod分散在不同的主机上。

   说明：
   应用非亲和性调度的设置方式与亲和性调度相同，但是相同的调度规则代表的意思不同，请根据使用场景进行选择。

7. 最后单击**创建**。
8. 创建成功后，默认进入创建完成页面，会列出应用包含的对象，您可以单击**查看应用详情**进行查看。

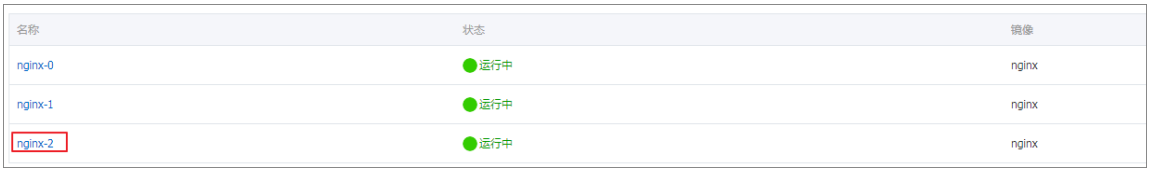
默认进入有状态副本集详情页面。



9. 然后单击左上角返回列表，进入有状态副本集列表页面，查看创建的StatefulSet应用。

10. （可选）选择所需的nginx应用，单击右侧**伸缩**，验证服务伸缩性。

a) 在弹出的对话框中，将容器组数量设置为3，您可发现扩容时，扩容容器组的排序依次递增；反之，进行缩容时，先按Pod次序从高到低进行缩容。这体现StatefulSet中Pod的次序稳定性。



b) 单击左侧导航栏中的**应用 > 存储声明**，您可发现，随着应用扩容，会随着Pod创建新的云盘卷；缩容后，已创建的PV/PVC不会删除。

后续操作

连接到Master节点，执行以下命令，验证持久化存储特性。

在云盘中创建临时文件：

```
# kubectl exec nginx-1 ls /tmp                    #列出该目录下的文件
lost+found

# kubectl exec nginx-1 touch /tmp/statefulset          #增加一个临时文件
statefulset

# kubectl exec nginx-1 ls /tmp
lost+found
statefulset
```

删除Pod，验证数据持久性：

```
# kubectl delete pod nginx-1
pod"nginx-1" deleted

# kubectl exec nginx-1 ls /tmp                                #数据持久化存储
```

```
lost+found
statefulset
```

此外，您还可发现，删除容器组后，过一段时间，容器组（Pod）会自动重启，证明StatefulSet应用的高可用性。

1.6.3 使用镜像创建Job类型应用

阿里云容器服务Kubernetes集群支持通过界面创建Job类型的应用。本例中将创建一个Job类型的busybox应用，并演示任务（Job）应用的特性。

前提条件

您已成功创建一个 Kubernetes 集群。参见创建*Kubernetes*集群。

背景信息

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。

Kubernetes支持以下几种Job：

- 非并行Job：通常创建一个Pod直至其成功结束
- 固定结束次数的Job：设置`.spec.completions`，创建多个Pod，直到`.spec.completions`个Pod成功结束
- 带有工作队列的并行Job：设置`.spec.Parallelism`但不设置`.spec.completions`，当所有Pod结束并且至少一个成功时，Job就认为是成功。
- 固定结束次数的并行Job：同时设置`.spec.completions`和`.spec.Parallelism`，多个Pod同时处理工作队列。

根据`.spec.completions`和`.spec.Parallelism`的设置，可以将Job划分为以下几种模式：

说明：

本例中创建的任务属于固定结束次数的并行Job。

Job类型	使用示例	行为	completions	Parallelism
一次性Job	数据库迁移	创建一个Pod直至其成功结束	1	1
固定结束次数的Job	处理工作队列的Pod	依次创建一个Pod运行直至	2+	1

Job类型	使用示例	行为	completions	Parallelism
		completions个成功结束		
固定结束次数的并行Job	多个Pod同时处理工作队列	依次创建多个Pod运行直至completions个成功结束	2+	2+
并行Job	多个Pod同时处理工作队列	创建一个或多个Pod直至有一个成功结束	1	2+

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 任务，然后单击页面右上角的使用镜像创建。
3. 在应用基本信息页面进行设置，然后单击下一步 进入应用配置页面。
   - 应用名称：设置应用的名称。
   - 部署集群：设置应用部署的集群。
   - 命名空间：设置应用部署所处的命名空间，默认使用default命名空间。
   - 类型：设置类型为任务。

   说明：
   本例中选择任务类型，即Job。

4. 设置容器配置。

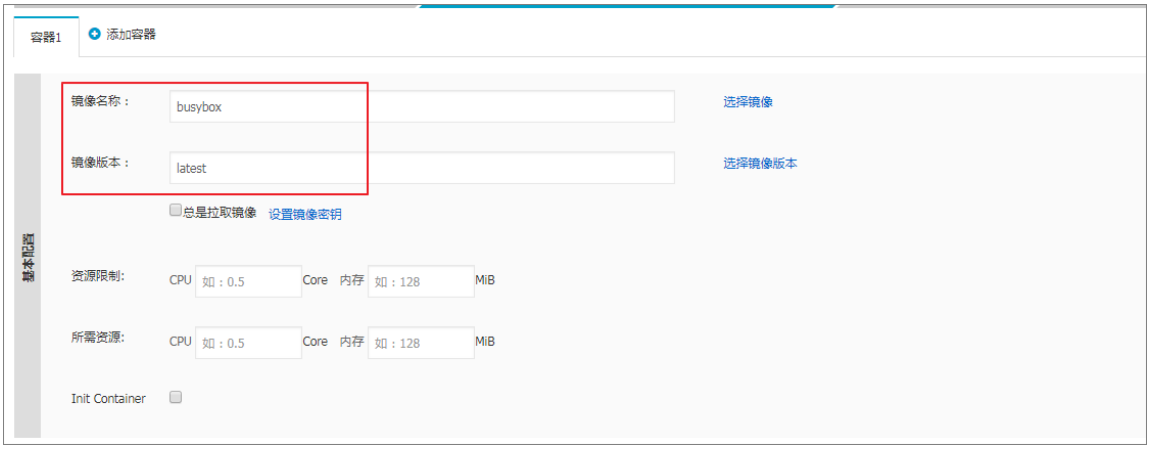
   说明：

您可为应用的Pod设置多个容器。

a) 设置容器的基本配置。

- **镜像名称**：您可以单击**选择镜像**，在弹出的对话框中选择所需的镜像并单击**确定**，本例中为 busybox。

  您还可以填写私有 registry。填写的格式为`domainname/namespace/imagename:tag`
- **镜像版本**：您可以单击**选择镜像版本** 选择镜像的版本。若不指定，默认为 latest。
- **总是拉取镜像**：为了提高效率，容器服务会对镜像进行缓存。部署时，如果发现镜像 Tag 与本地缓存的一致，则会直接复用而不重新拉取。所以，如果您基于上层业务便利性等因素考虑，在做代码和镜像变更时没有同步修改 Tag ，就会导致部署时还是使用本地缓存内旧版本镜像。而勾选该选项后，会忽略缓存，每次部署时重新拉取镜像，确保使用的始终是最新的镜像和代码。
- **设置镜像密钥**：若您在使用私有镜像时，您可使用镜像密钥，保障镜像安全。具体配置请参见使用镜像密钥。
- **资源限制**：可指定该应用所能使用的资源上限，包括 CPU 和 内存两种资源，防止占用过多资源。其中，CPU 资源的单位为 millicores，即一个核的千分之一；内存的单位为 Bytes，可以为 Gi、Mi 或 Ki。
- **所需资源**：即为该应用预留资源额度，包括 CPU 和 内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争抢资源，导致应用不可用。
- **Init Container**：勾选该项，表示创建一个Init Container，Init Container包含一些实用的工具，具体参见*https://kubernetes.io/docs/concepts/workloads/pods/init-containers/*。

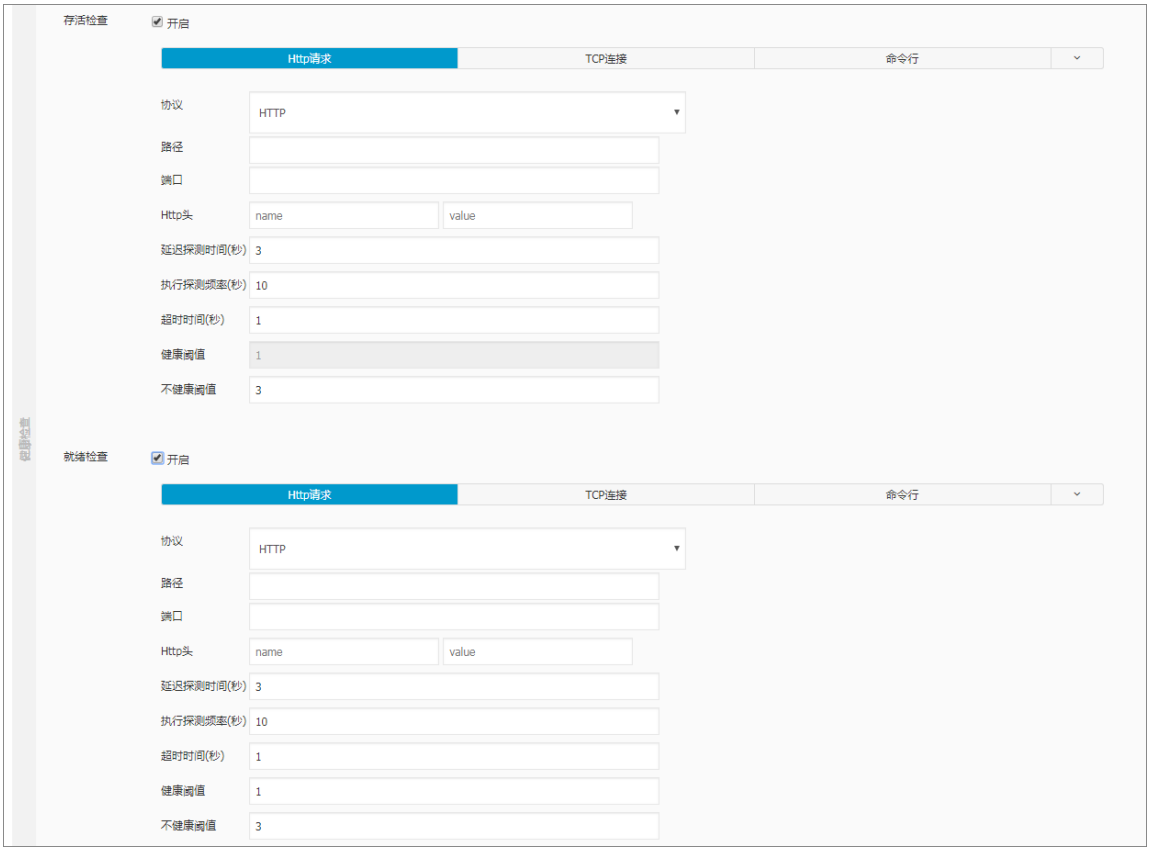


b) （可选）配置环境变量。

支持通过键值对的形式为 Pod 配置环境变量。用于给 Pod 添加环境标志或传递配置等，具体请参见 *Pod variable*。

c) （可选）配置健康检查。

支持存活检查（liveness）和就绪检查（Readiness）。存活检查用于检测何时重启容器；就绪检查确定容器是否已经就绪，且可以接受流量。关于健康检查的更多信息，请参见*https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*。



请求类型	配置说明
HTTP请求	即向容器发送一个HTTPget 请求，支持的参数包括： • 协议：HTTP/HTTPS • 路径：访问HTTP server 的路径 • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • HTTP头：即HTTPHeaders，HTTP请求中自定义的请求头，HTTP允许重复的header。支持键值对的配置方式。

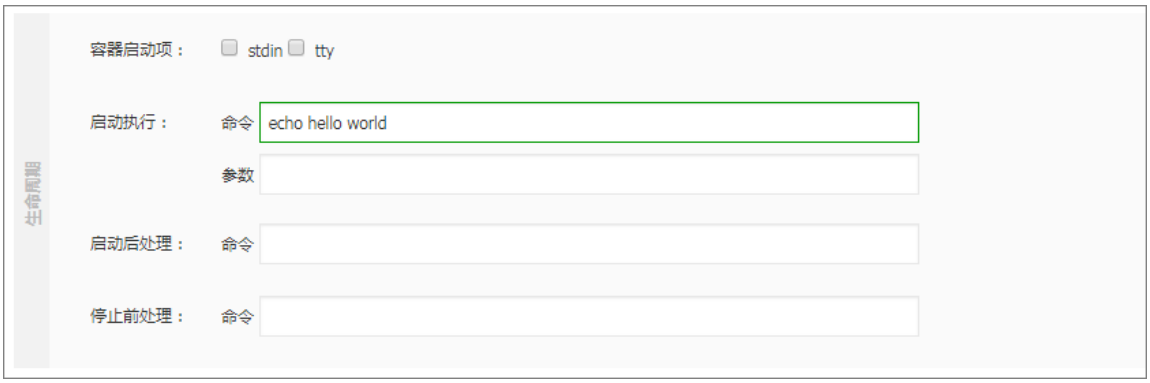
请求类型	配置说明
	• 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为3秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。
TCP连接	即向容器发送一个TCP Socket，kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： • 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。

请求类型	配置说明
命令行	通过在容器中执行探针检测命令，来检测容器的健康情况。支持的参数包括： • 命令行：用于检测容器健康情况的探测命令。 • 延迟探测时间（秒）：即**initialDelaySeconds**，容器启动后第一次执行探测时需要等待多少秒，默认为5秒。 • 执行探测频率（秒）：即**periodSeconds**，指执行探测的时间间隔，默认为10s，最低为1s。 • 超时时间（秒）：即**timeoutSeconds**，探测超时时间。默认1秒，最小1秒。 • 健康阈值：探测失败后，最少连续探测成功多少次才被认定为成功。默认是1，最小值是1。对于存活检查（liveness）必须是1。 • 不健康阈值：探测成功后，最少连续探测失败多少次才被认定为失败。默认是3。最小值是1。

d) （可选）配置生命周期。

您可以为容器的生命周期配置容器启动项、启动执行、启动后处理和停止前处理。具体参见*https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/*。

- **容器启动项**：勾选 stdin 表示为该容器开启标准输入；勾选 tty 表示为该容器分配一个虚拟终端，以便于向容器发送信号。通常这两个选项是一起使用的，表示将终端 (tty) 绑定到容器的标准输入（stdin）上，比如一个交互式的程序从用户获取标准输入，并显示到终端中。
- **启动执行**：为容器设置预启动命令和参数。
- **启动后处理**：为容器设置启动后的命令。
- **停止前处理**：为容器设置预结束命令。



e) （可选）配置数据卷信息。

支持配置本地存储和云存储。

- **本地存储**：支持主机目录（hostpath）、配置项（configmap）、保密字典（secret）和临时目录，将对应的挂载源挂载到容器路径中。更多信息参见 *volumes*。
- **云存储**：支持云盘/NAS/OSS三种云存储类型。

f) （可选）配置**日志服务**，您可进行采集配置和自定义Tag设置。

> **说明：**
> 请确保已部署Kubernetes集群，并且在此集群上已安装日志插件。

您可对日志进行采集配置：

- **日志库**：即在日志服务中生成一个对应的logstore，用于存储采集到的日志。
- **容器内日志路径**：支持stdout和文本日志。
  - **stdout**： stdout 表示采集容器的标准输出日志。
  - 文本日志：您可收集容器内指定路径的文本日志，同时支持通配符的方式。

您还可设置自定义 tag，设置tag后，会将该tag一起采集到容器的日志输出中。自定义 tag 可帮助您给容器日志打上tag，方便进行日志统计和过滤等分析操作。

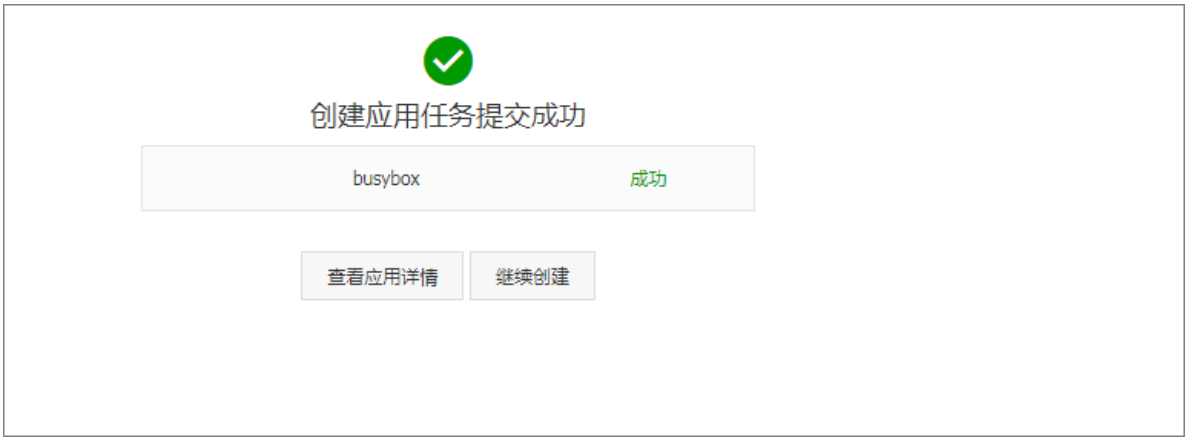
5. 完成容器配置后，单击 下一步。
6. 进行高级设置。

您可进行**任务配置**。

参数	说明
成功运行的Pod数	即completions，指定job需要成功运行Pods的次数。默认值为1

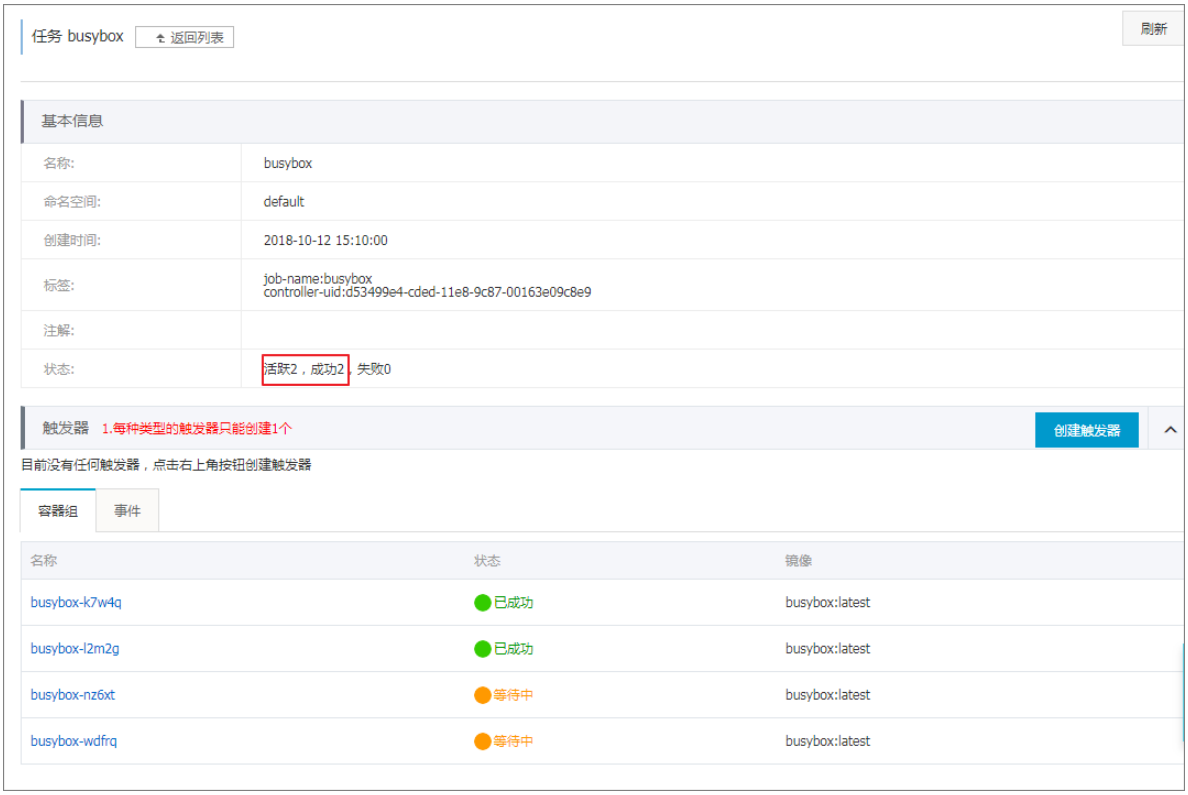
参数	说明
并行运行的Pod数	即parallelism，指定job在任一时刻应该并发运行Pod的数量。默认值为1
超时时间	即activeDeadlineSeconds，指定job可运行的时间期限，超过时间还未结束，系统将会尝试进行终止。
重试次数	即backoffLimit，指定job失败后进行重试的次数。默认是6次，每次失败后重试会有延迟时间，该时间是指数级增长，最长时间是6min。
重启策略	仅支持不重启（Never）和失败时（OnFailure）

7. 最后单击创建。
8. 创建成功后，默认进入创建完成页面，会列出应用包含的对象。

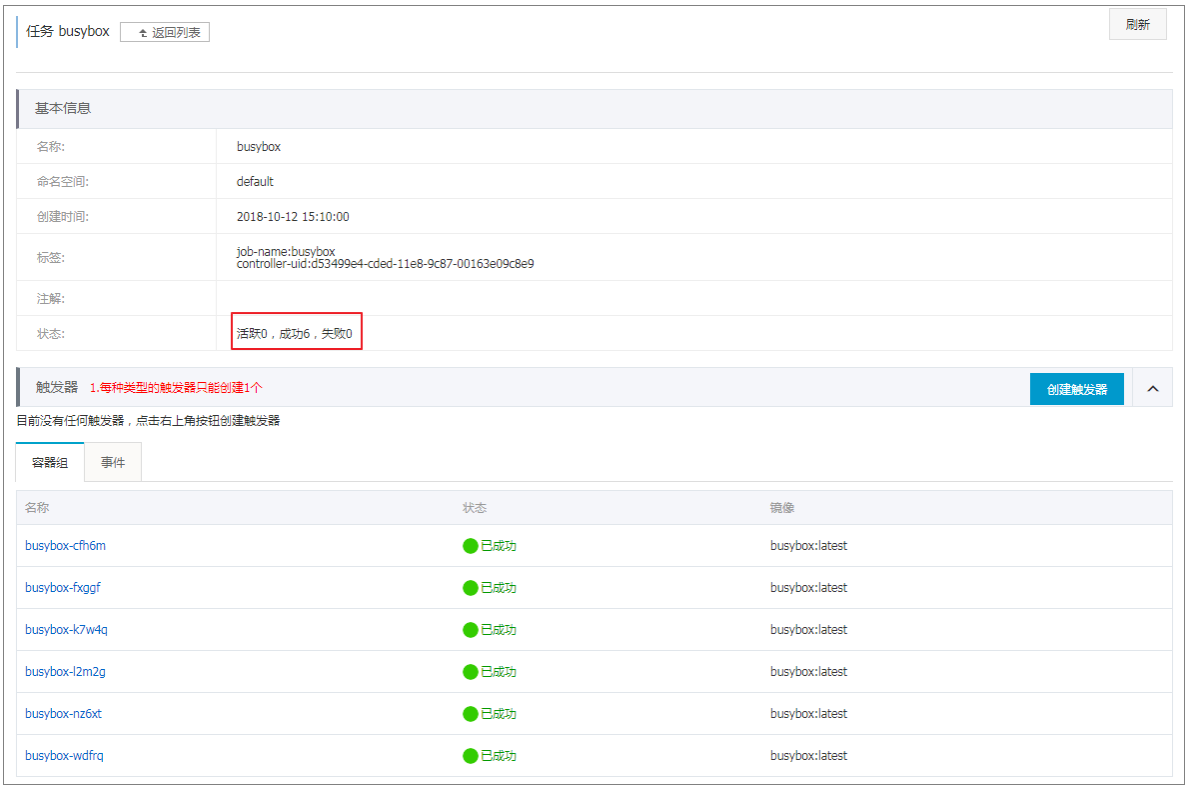


您可以单击查看应用详情，进入任务详情页面。

创建过程中，您可在状态栏中查看容器组的创建情况。本例中按照任务定义，一次性并行创建2个Pod。



等待一段时间，所有容器组创建完毕。



9. 单击左上角返回列表，进入任务列表页面中，您可看到，该任务已显示完成时间。

> **说明：**
> 若任务未创建完毕所有容器组，任务不会显示完成时间。

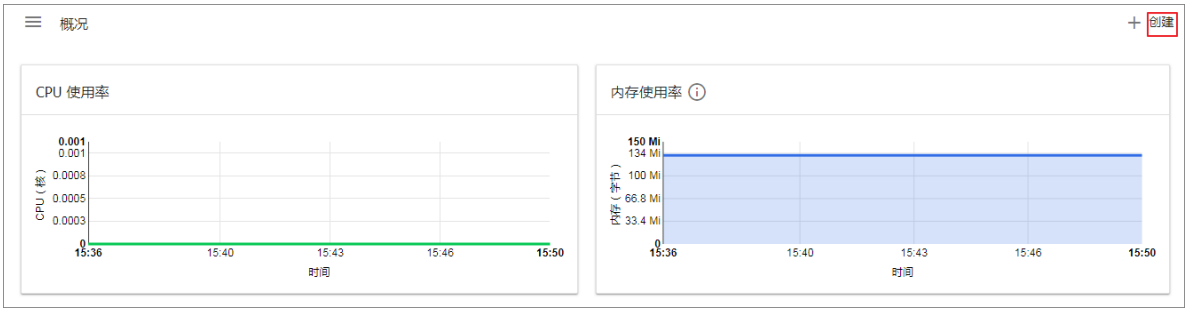
1.6.4 通过 Kubernetes Dashboard 创建应用

您可以通过 Kubernetes Dashboard 创建应用。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

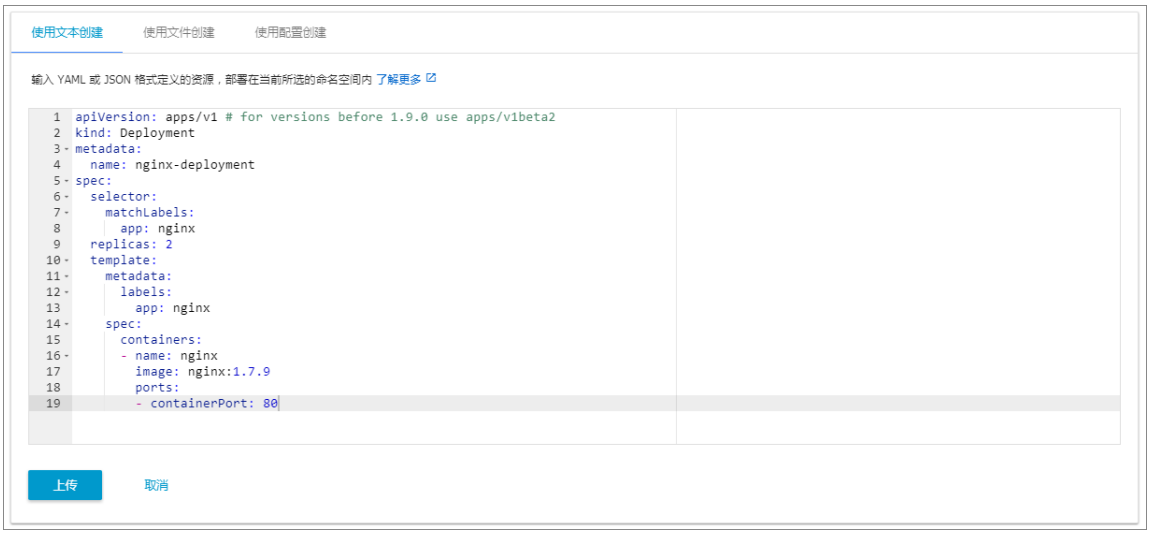
4. 在 Kubernetes Dashboard 中，单击页面右上角的创建。



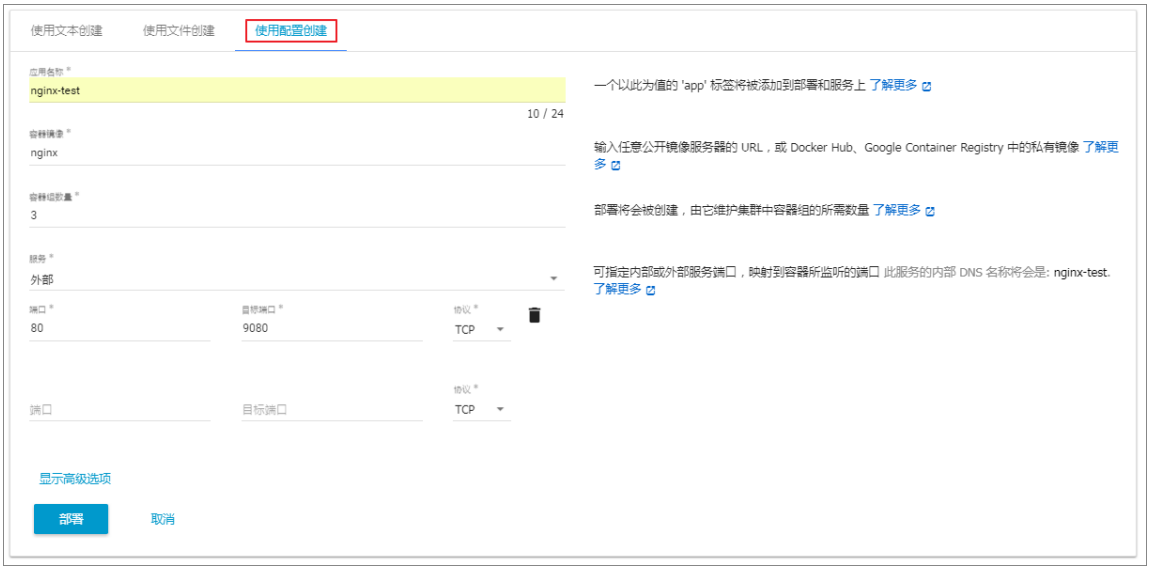
5. 在弹出的对话框中，设置应用的信息。

   您可以通过以下 3 种方法之一创建应用：

   - 使用文本创建：直接输入 YAML 或 JSON 格式的编排代码创建应用。您需要了解对应的编排格式，如下所示是一个 YAML 格式的编排模板。



- 使用文件创建：通过导入已有的 YAML 或 JSON 配置文件创建应用。
- 使用配置创建：本例中通过指定下边的设置创建应用。
  - 应用名称：所创建应用的名称。本示例中为 `nginx`。
  - 容器镜像：所要使用的镜像的 URL。本示例使用的是 Docker *Nginx*。
  - 容器组数量：创建的应用的 pod 个数。
  - 服务：可设置为外部 或内部。外部表示创建一个可以从集群外部访问的服务；内部表示创建一个集群内部可以访问的服务。
  - 高级选项：您可以选择显示高级选项，对标签、环境变量等选项进行配置。 此设置将流量负载均衡到三个 Pod。



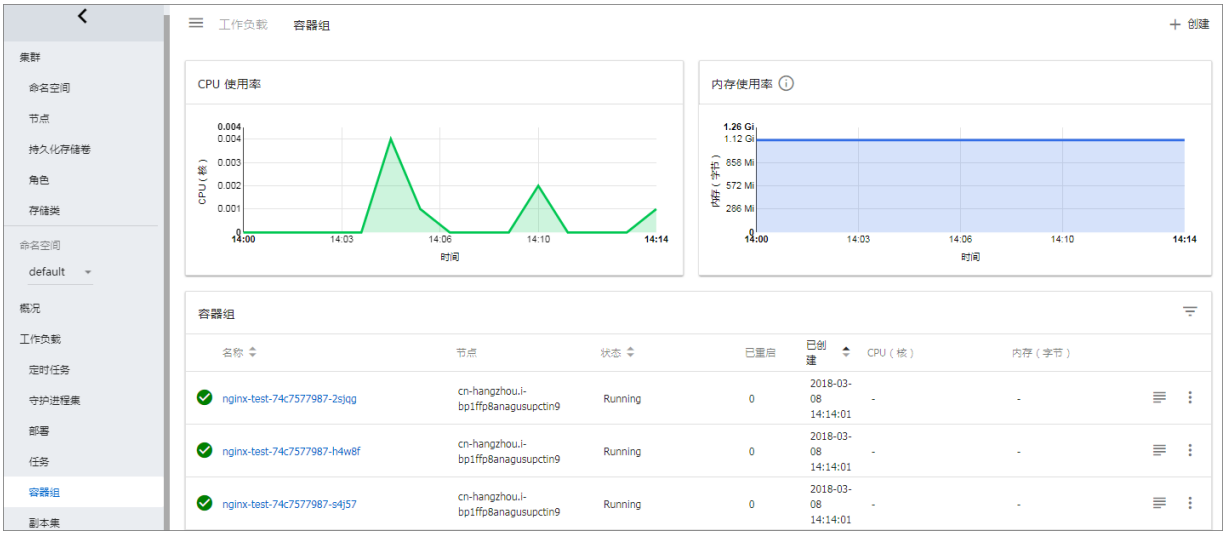
6. 单击部署 部署这些容器和服务。

   您也可以单击显示高级选项 展开高级选项进一步配置相关参数。

后续操作

单击部署后，您可以查看应用的服务或查看应用的容器。

单击左侧导航栏中的容器，您可以通过左侧的图标查看每个 Kubernetes 对象的状态。 表示对象仍然处于部署状态。 表示对象已经完成部署。



1.6.5 通过编排模板创建应用

在容器服务 kubernetes 模板编排中，您需要自己定义一个应用运行所需的资源对象，通过标签选择器等机制，将资源对象组合成一个完整的应用。

前提条件

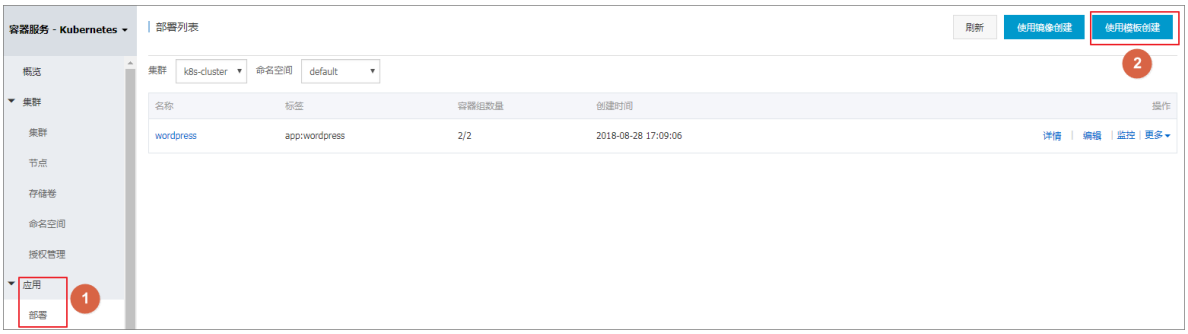
创建一个 kubernetes 集群，参见创建*Kubernetes*集群。

背景信息

本例演示如何通过一个编排模板创建 nginx 应用，包含一个 Deployment 和 Service，后端 Deployment会创建Pod 资源对象， Service 会绑定到后端 Pod 上，形成一个完整的 nginx 应用。

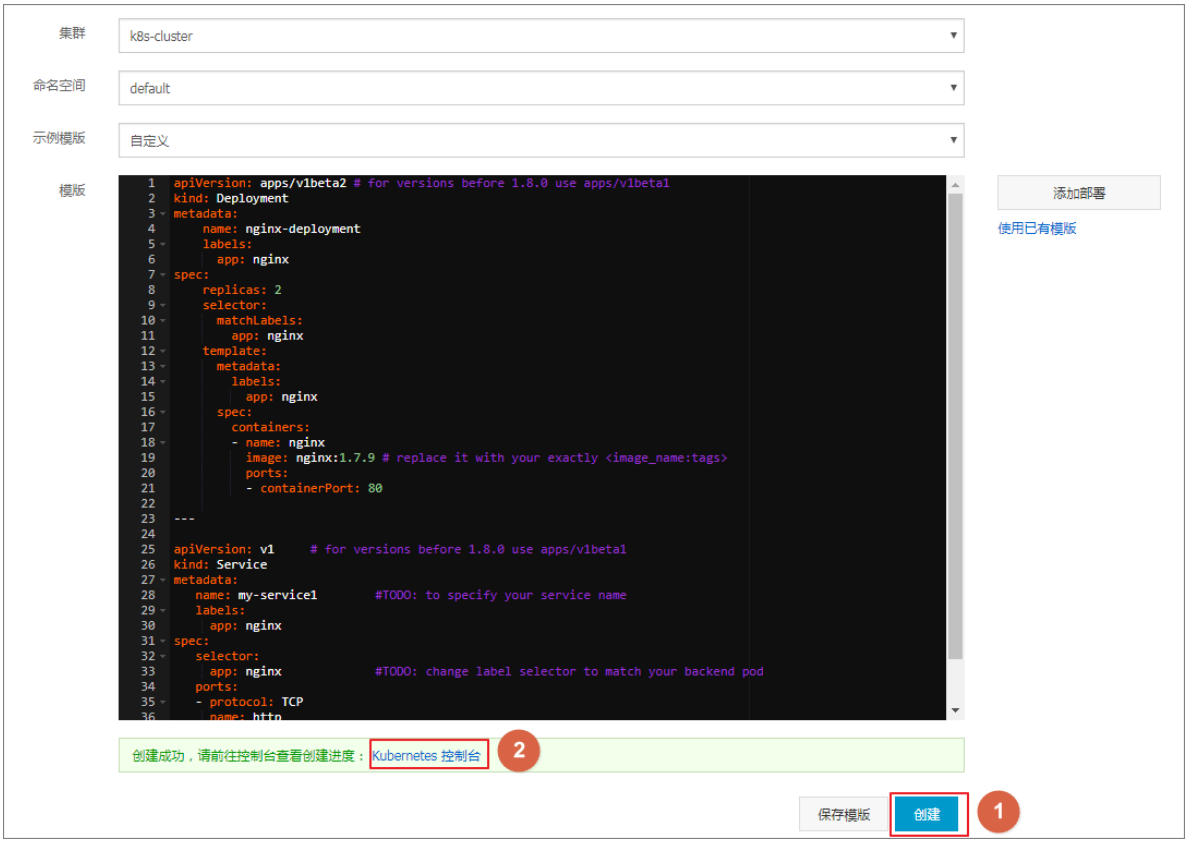
操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。



4. 对模板进行相关配置，完成配置后单击创建。

- 集群：选择目标集群。资源对象将部署在该集群内。
- 命名空间：选择资源对象所属的命名空间，默认是 default。除了节点、持久化存储卷等底层计算资源以外，大多数资源对象需要作用于命名空间。
- 示例模板：阿里云容器服务提供了多种资源类型的 Kubernetes yaml 示例模板，让您快速部署资源对象。您可以根据 Kubernetes Yaml 编排的格式要求自主编写，来描述您想定义的资源类型。
- 添加部署：您可通过此功能快速定义一个Yaml模板。
- 使用已有模板：您可将已有编排模板导入到模板配置页面。



下面是一个 nginx 应用的示例编排，基于容器服务内置的编排模板。通过该编排模板，即可快速创建一个属于 nginx 应用的 deployment 。

说明：

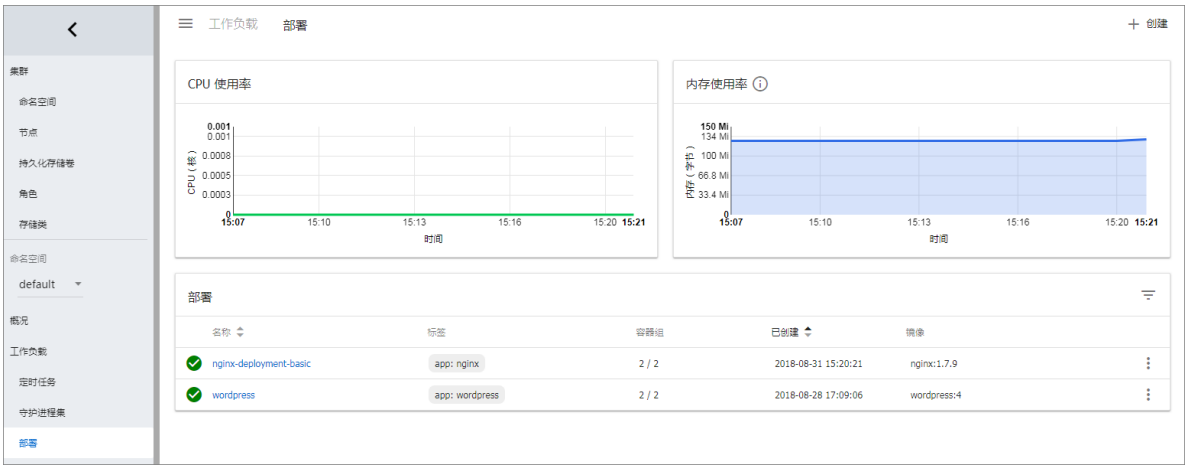
容器服务支持Kubernetes Yaml编排，支持通过---符号将资源对象分隔，从而通过一个模板创建多个资源对象。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
    name: nginx-deployment
    labels:
      app: nginx
spec:
    replicas: 2
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - name: nginx
          image: nginx:1.7.9 # replace it with your exactly <
image_name:tags>
          ports:
          - containerPort: 80

---

apiVersion: v1     # for versions before 1.8.0 use apps/v1beta1
kind: Service
metadata:
   name: my-service1        #TODO: to specify your service name
   labels:
     app: nginx
spec:
   selector:
     app: nginx             #TODO: change label selector to match
your backend pod
   ports:
   - protocol: TCP
     name: http
     port: 30080            #TODO: choose an unique port on each
node to avoid port conflict
     targetPort: 80
   type: LoadBalancer        ##本例中将type从Nodeport修改为LoadBalancer
```

5. 单击创建后。会提示部署状态信息。成功后，单击**Kubernetes** 控制台前往Kubernetes Dashboard 查看部署进度。



6. 在 Kubernetes Dashboard 里，您可以看到 my-service1 服务已成功部署，并暴露了外部入口。单击外部入口的访问地址。

7. 您可以在浏览器中访问 nginx 服务欢迎页面。

后续操作

您也可返回容器服务首页，单击左侧导航栏中的应用 > 服务，查看该nginx的服务。

1.6.6 通过命令管理应用

您可以通过命令创建应用或者查看应用的容器。

前提条件

在本地使用命令前，您需要先设置通过 *kubectl* 连接 *Kubernetes* 集群。

通过命令创建应用

可以通过运行以下语句来运行简单的容器（本示例中为 Nginx Web 服务器）：

```
 # kubectl run -it nginx --image=registry.aliyuncs.com/spacexnice/
netdia:latest
```

此命令将为该容器创建一个服务入口，指定 `--type=LoadBalancer` 将会为您创建一个阿里云负载均衡路由到该 Nginx 容器。

```
# kubectl expose deployment nginx --port=80 --target-port=80 --type=
LoadBalancer
```

通过命令查看容器

运行如下命令列出所有 default 命名空间里正在运行的容器。

```
root@master # kubectl get pods
NAME                                READY     STATUS    RESTARTS
AGE
nginx-2721357637-dvwq3              1/1       Running   1
9h
```

1.6.7 利用 Helm 简化应用部署

在 Kubernetes 中，应用管理是需求最多、挑战最大的领域。Helm 项目提供了一个统一软件打包方式，支持版本控制，可以大大简化 Kubernetes 应用分发与部署中的复杂性。

阿里云容器服务在应用目录管理功能中集成了 Helm 工具，并进行了功能扩展，支持官方 Repository，让您快速部署应用。您可以通过命令行或容器服务控制台界面两种方式进行部署。

本文档介绍 Helm 的基本概念和使用方式，演示在阿里云的 Kubernetes 集群上利用 Helm 来部署示例应用 WordPress 和 Spark。

Helm 基本概念

Helm 是由 Deis 发起的一个开源工具，有助于简化部署和管理 Kubernetes 应用。

Helm 可以理解为 Kubernetes 的包管理工具，可以方便地发现、共享和使用为 Kubernetes 构建的应用，它包含几个基本概念

- **Chart**：一个 Helm 包，其中包含了运行一个应用所需要的镜像、依赖和资源定义等，还可能包含 Kubernetes 集群中的服务定义，类似 Homebrew 中的 formula、APT 的 dpkg 或者 Yum 的 rpm 文件。

- **Release**：在 Kubernetes 集群上运行的 Chart 的一个实例。在同一个集群上，一个 Chart 可以安装很多次。每次安装都会创建一个新的 release。例如一个 MySQL Chart，如果想在服务器上运行两个数据库，就可以把这个 Chart 安装两次。每次安装都会生成自己的 Release，会有自己的 Release 名称。
- **Repository**：用于发布和存储 Chart 的存储库。

Helm 组件

Helm 采用客户端/服务器架构，由如下组件组成：

- Helm CLI 是 Helm 客户端，可以在 Kubernetes 集群的 master 节点或者本地执行。
- Tiller 是服务器端组件，在 Kubernetes 集群上运行，并管理 Kubernetes 应用程序的生命周期。
- Repository 是 Chart 存储库，Helm 客户端通过 HTTP 协议来访问存储库中 Chart 的索引文件和压缩包。

使用 Helm 部署应用

前提条件

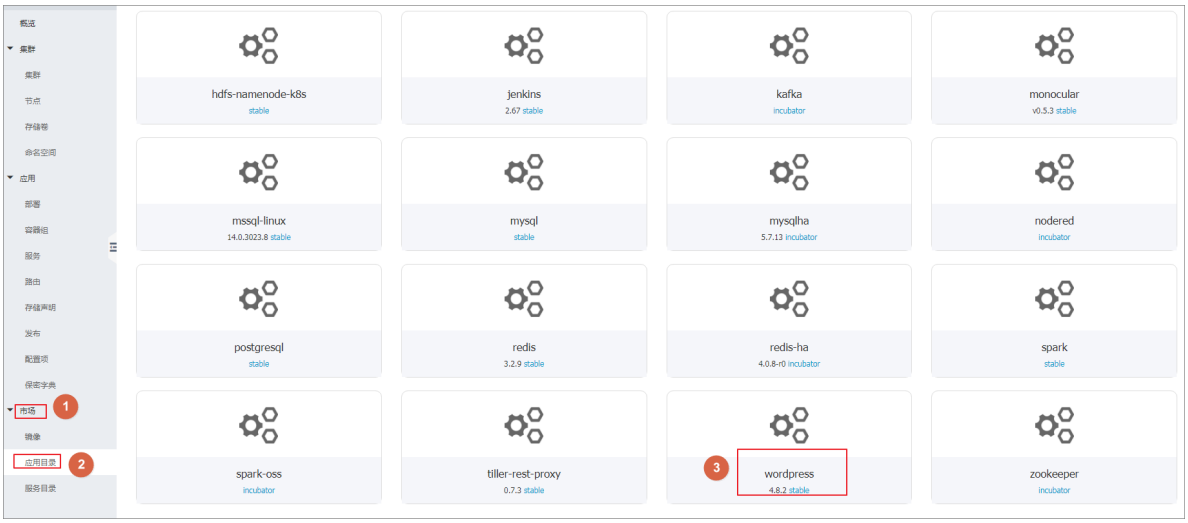
- 通过 Helm 部署应用之前，利用阿里云容器服务来创建 Kubernetes 集群。参见创建*Kubernetes*集群。

  在 Kubernetes 集群创建的同时，Tiller 将会被自动部署到集群之中，并且在所有的 master 节点上自动安装 Helm CLI 以及配置指向阿里云的 Chart 存储库。
- 查看您集群中 Kubernetes 的版本。

  仅支持 Kubernetes 版本 1.8.4 及以上的集群。对于 1.8.1 版本的集群，您可以在集群列表中进行集群升级操作。

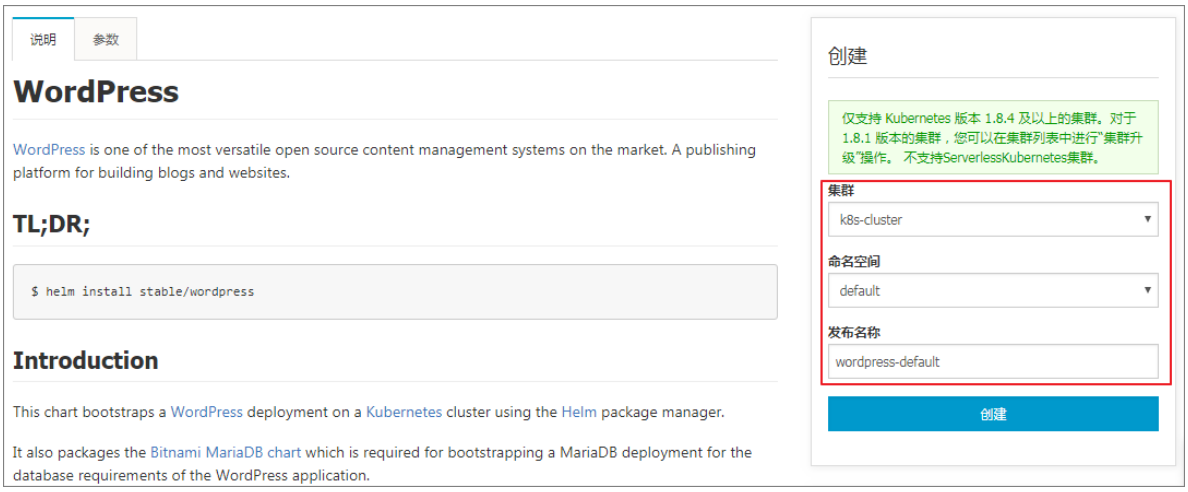
通过控制台界面部署应用

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**市场 > 应用目录**，进入应用目录列表页面。
3. 选择一个 chart（本示例选择 WordPress），单击该 chart，进入 chart 详情页面。



4. 在页面右侧，填写部署的基本信息。

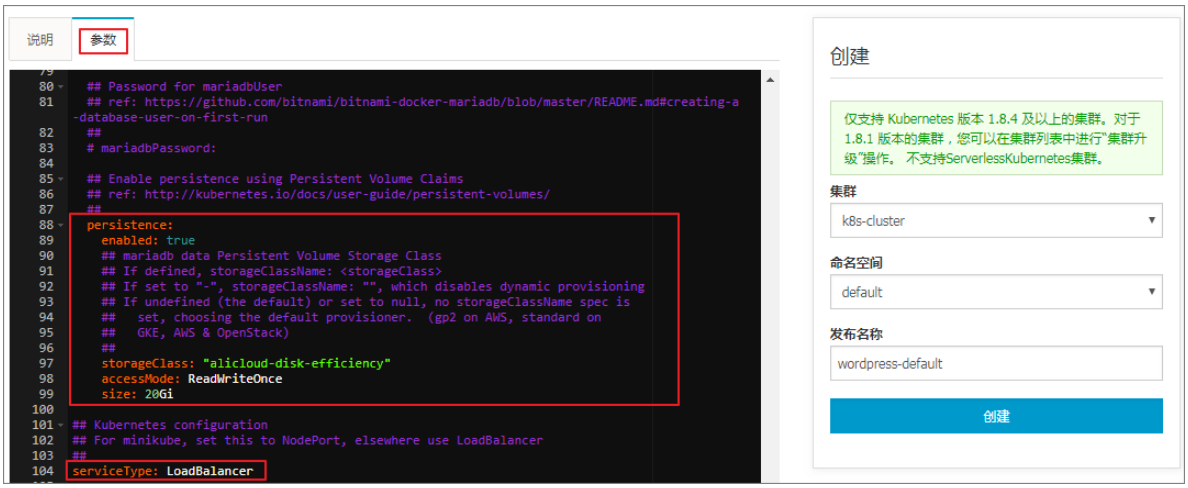
   - 集群：应用要部署到的集群。
   - 命名空间：选择命名空间。默认为 default。
   - 发布名称：填写应用发布的名称。本例中为 test。



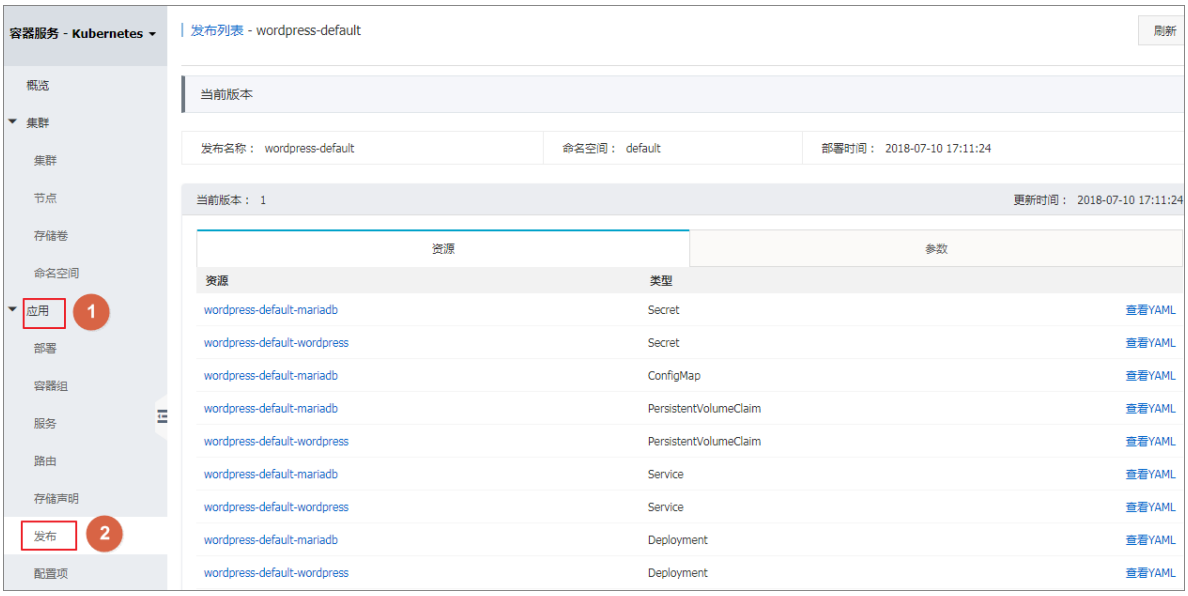
5. 单击参数，对配置进行修改。

   本示例中使用云盘的动态数据卷绑定一个PVC，参见使用阿里云云盘。

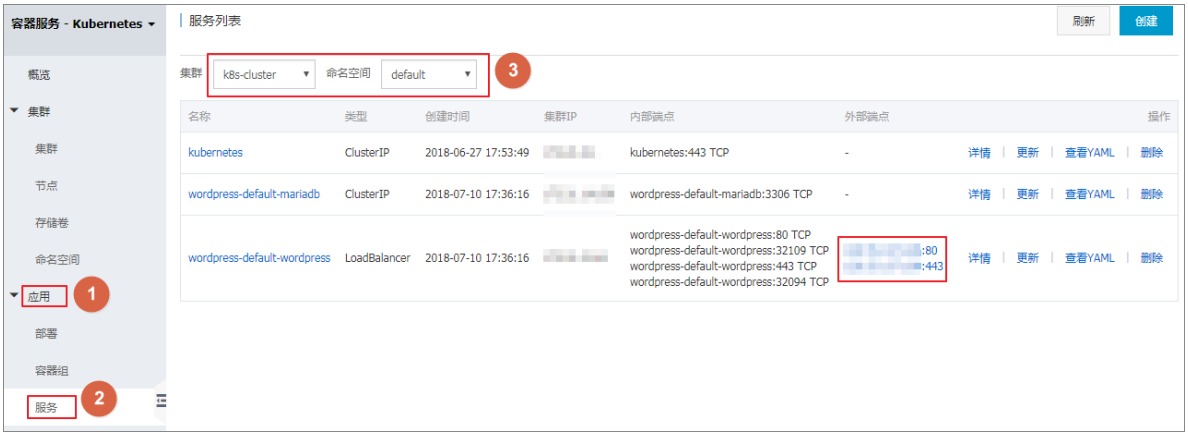
   > **说明：**
   > 您需要预先创建一个云盘存储卷（PV），并且存储卷的容量不能小于PVC定义的数值。



6. 配置完成后，单击创建，部署成功后，默认进入该应用的发布页面。



7. 单击左侧导航栏中的应用 > 服务，选择所需的集群和命名空间，找到对应的服务，您可获取 http/https 外部端点的地址。



8. 单击上面的访问地址，进入 WordPress 博客发布页面。

通过命令行部署应用

通过命令行部署应用时，您可以 SSH 登录 Kubernetes 集群的 master 节点（Helm CLI 已自动安装并已配置Repository）进行操作，参见*SSH访问Kubernetes集群*。您也可以在本地安装配置 kubectl 和 Helm CLI。

本示例以在本地安装配置 kubectl 和 Helm CLI 并部署 WordPress 和 Spark 应用为例进行说明。

安装配置 kubectl 和 Helm CLI

1. 在本地计算机上安装和配置 kubectl。

   参见*通过 kubectl 连接 Kubernetes 集群*。

   若要查看 Kubernetes 目标集群的信息，键入命令 `kubectl cluster-info`。

2. 在本地计算机上安装 Helm。

   安装方法，参见 *Install Helm*。

3. 配置 Helm 的 Repository。这里我们使用了阿里云容器服务提供的 Charts 存储库。

   ```
   helm init --client-only --stable-repo-url https://aliacs-app-catalog
   .oss-cn-hangzhou.aliyuncs.com/charts/
   helm repo add incubator https://aliacs-app-catalog.oss-cn-hangzhou.
   aliyuncs.com/charts-incubator/
   helm repo update
   ```

Helm 基础操作

- 若要查看在集群上安装的 Charts 列表，请键入：

  `helm list`

  或者缩写

  `helm ls`

- 若要查看存储库配置，请键入：

  `helm repo list`

- 若要查看或搜索存储库中的 Helm charts，请键入以下任一命令：

  ```
  helm search
  helm search 存储库名称 #如 stable 或 incubator
  helm search chart名称 #如 wordpress 或 spark
  ```

- 若要更新 charts 列表以获取最新版本，请键入：

  `helm repo update`

有关 Helm 使用的详细信息，请参阅 *Helm项目*。

部署 WordPress

下面我们将利用 Helm，来部署一个 WordPress 博客网站。

输入如下命令。

```
helm install --name wordpress-test stable/wordpress
```

说明：

阿里云 Kubernetes 服务提供块存储（云盘）的动态存储卷支持，您需要预先创建一个云盘存储卷。

得到如下的结果。

```
NAME:   wordpress-test
LAST DEPLOYED: Mon Nov  20 19:01:55 2017
NAMESPACE: default
STATUS: DEPLOYED
...
```

利用如下命令查看 WordPress 的 release 和 service。

```
helm list
kubectl get svc
```

利用如下命令查看 WordPress 相关的 Pod，并等待其状态变为 Running。

```
kubectl get pod
```

利用如下命令获得 WordPress 的访问地址。

```
echo http://$(kubectl get svc wordpress-test-wordpress -o jsonpath='{.
status.loadBalancer.ingress[0].ip}')
```

通过上面的 URL，可以在浏览器上看到熟悉的 WordPress 站点。

也可以根据 Charts 的说明，利用如下命令获得 WordPress 站点的管理员用户和密码。

```
echo Username: user
```

```
echo Password: $(kubectl get secret --namespace default wordpress-test
-wordpress -o jsonpath="{.data.wordpress-password}" | base64 --decode)
```

如需彻底删除 WordPress 应用，可输入如下命令。

```
helm delete --purge wordpress-test
```

通过 Helm 部署 Spark

下面我们将利用 Helm，来部署 Spark 以用于大数据处理。

输入如下命令。

```
helm install --name myspark stable/spark
```

得到如下的结果。

```
NAME:   myspark
LAST DEPLOYED: Mon Nov 20 19:24:22 2017
NAMESPACE: default
STATUS: DEPLOYED
...
```

利用如下命令查看 Spark 的 release 和 service。

```
helm list
kubectl get svc
```

利用如下命令查看 Spark 相关的 Pod，并等待其状态变为 Running。因为 Spark 的相关镜像体积较大，所以拉取镜像需要一定的时间。

```
kubectl get pod
```

利用如下命令获得 Spark Web UI 的访问地址。

```
echo http://$(kubectl get svc myspark-webui -o jsonpath='{.status.
loadBalancer.ingress[0].ip}'):8080
```

通过上面的 URL，可以在浏览器上看到 Spark 的 Web UI，上面显示 worker 实例当前为 3 个。

接下来，我们将利用如下命令，使用 Helm 对 Spark 应用做升级，将 worker 实例数量从 3 个变更为 4 个。请注意参数名称是大小写敏感的。

```
helm upgrade myspark --set "Worker.Replicas=4" stable/spark
```

得到如下结果。

```
Release "myspark" has been upgraded. Happy Helming!
LAST DEPLOYED: Mon Nov 20 19:27:29 2017
NAMESPACE: default
```

```
STATUS: DEPLOYED
...
```

利用如下命令查看 Spark 新增的 Pod，并等待其状态变为 Running。

```
kubectl get pod
```

在浏览器上刷新 Spark 的 Web UI，可以看到此时 worker 数量已经变为 4 个。

如需彻底删除 Spark 应用，可输入如下命令。

```
helm delete --purge myspark
```

使用第三方的 Chart 存储库

您除了可以使用预置的阿里云的 Chart 存储库，也可以使用第三方的 Chart 存储库（前提是网络是可达的）。使用如下命令格式添加第三方 Chart 存储库。

```
helm repo add 存储库名 存储库URL
helm repo update
```

关于 Helm 相关命令的说明，您可以参阅 *Helm* 文档

参考信息

Helm 催生了社区的发展壮大，越来越多的软件提供商，如 Bitnami 等公司，开始提供高质量的 Charts。您可以在 `https://kubeapps.com/` 中寻找和发现已有的 Charts。

1.6.8 创建服务

Kubernetes Service 定义了这样一种抽象：一个 Pod 的逻辑分组，一种可以访问它们的策略，通常称为微服务。这一组 Pod 能够被 Service 访问到，通常是通过 Label Selector 来实现的。

在 Kubernetes 中，pod 虽然拥有独立的 IP， 但 pod 会快速地创建和删除，因此，通过 pod 直接对外界提供服务不符合高可用的设计准则。通过 service 这个抽象，Service 能够解耦 frontend（前端）和 backend（后端） 的关联，frontend 不用关心 backend 的具体实现，从而实现松耦合的微服务设计。

更多详细的原理，请参见 *Kubernetes service*。

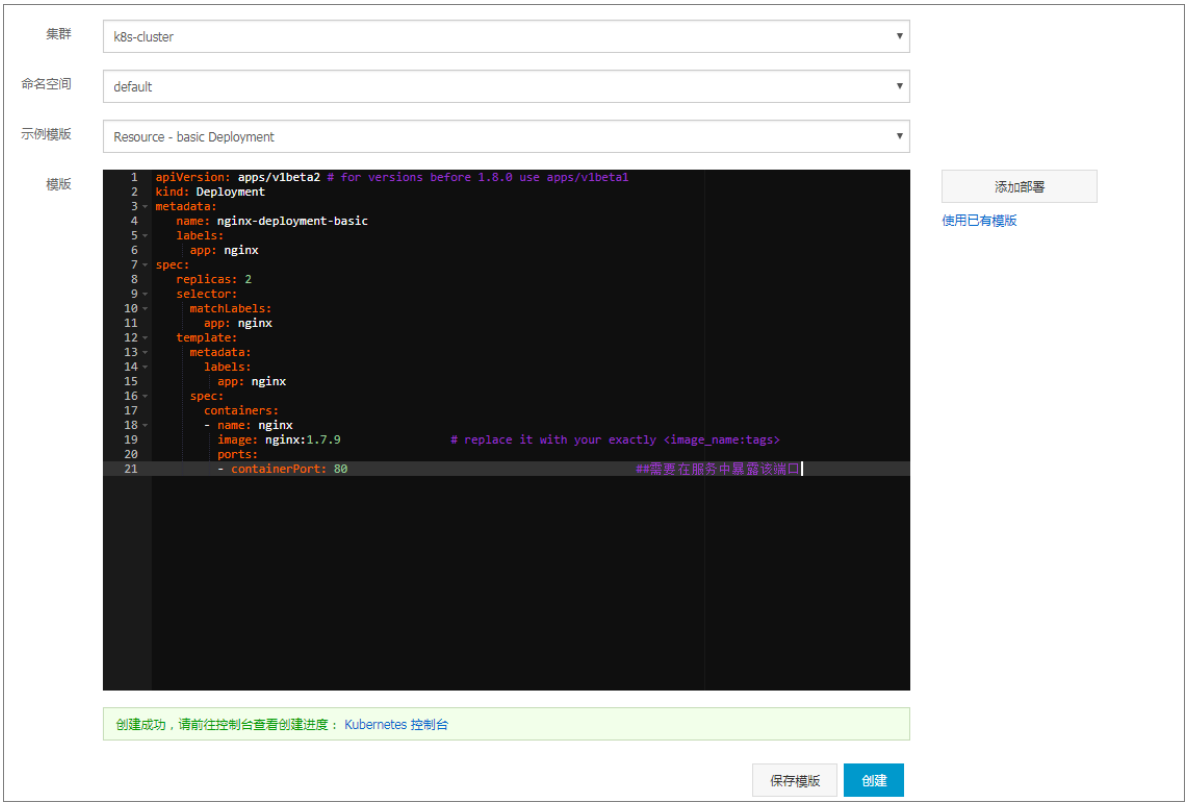
前提条件

您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。

步骤1 创建 deployment

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，单击页面右上角的使用模板创建。

3. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。

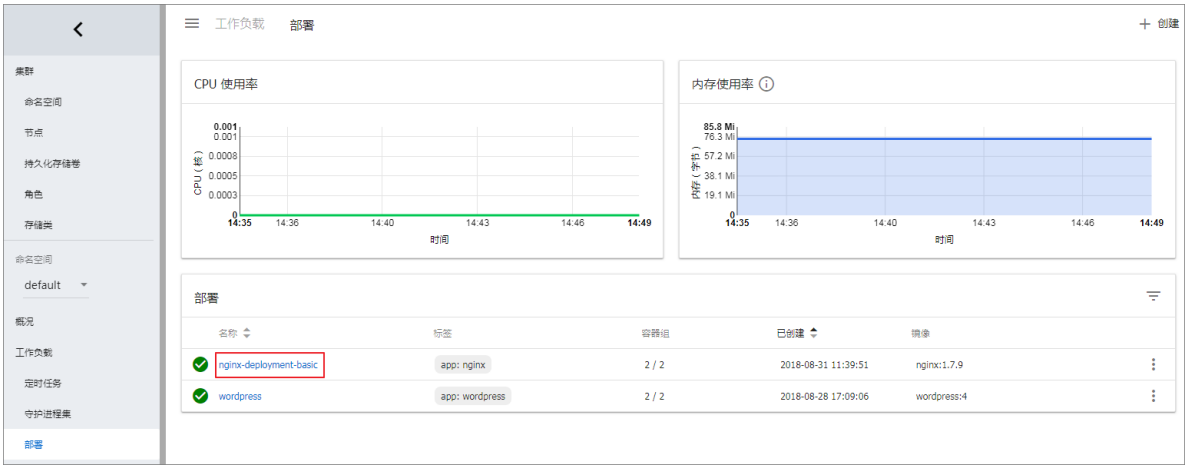


本例中，示例模板是一个 nginx 的 deployment。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
   name: nginx-deployment-basic
   labels:
     app: nginx
spec:
   replicas: 2
   selector:
     matchLabels:
       app: nginx
```

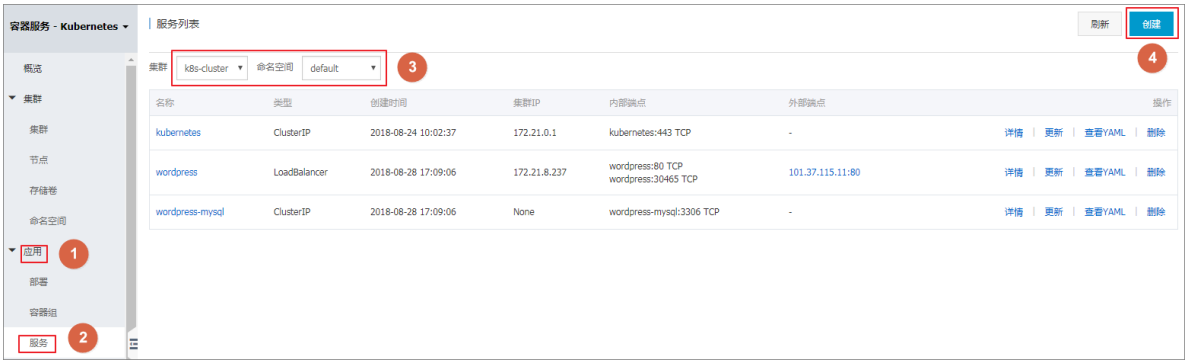
```
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - name: nginx
          image: nginx:1.7.9                # replace it with your
exactly <image_name:tags>
          ports:
          - containerPort: 80
  ##需要在服务中暴露该端口
```

4. 单击**Kubernetes** 控制台，进入 Kubernetes Dashboard，查看该 Deployment 的运行状态。



步骤2 创建服务

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 服务，进入服务列表页面。
3. 选择所需的集群和命名空间，单击页面右上角的创建。



4. 在弹出的创建服务对话框中，进行配置。

- **名称**：输入服务的名称，本例中为 nginx-vc。
- **类型**：选择服务类型，即服务访问的方式，包括：
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部可以访问，这也是默认的 ServiceType。

  - 节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
  - 负载均衡：即 LoadBalancer，指阿里云提供的负载均衡服务（SLB），可选择公网访问或内网访问。阿里云负载均衡服务可以路由到 NodePort 服务和 ClusterIP 服务。
- 关联部署：选择服务要绑定的后端对象，本例中是前面创建的 nginx-deployment-basic 。若不进行关联部署，则不会创建相关的 Endpoints 对象，您可自己进行绑定，参见 *services-without-selectors*。
- 端口映射：添加服务端口和容器端口，容器端口需要与后端的 pod 中暴露的容器端口一致。
- 注解：为该服务添加一个注解（annotation），配置负载均衡的参数，例如设置`service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:20`表示将该服务的带宽峰值设置为20Mbit/s，从而控制服务的流量。更多参数请参见通过负载均衡#Server Load Balancer#访问服务。
- 标签：您可为该服务添加一个标签，标识该服务。

5. 单击创建，nginx-svc 服务出现在服务列表中。

6. 您可查看服务的基本信息，在浏览器中访问 nginx-svc 的外部端点。

至此，您完成如何创建一个关联到后端的 deployment 的服务，最后成功访问 Nginx 的欢迎页面。

1.6.9 服务伸缩

应用创建后，您可以根据自己的需求来进行服务扩容或缩容。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。

3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

4. 在 Kubernetes Dashboard 中，单击左侧导航栏中的部署，查看部署的 Deployment。

5. 选择所需的 Deployment，单击右侧的操作图标并单击伸缩。

6. 在弹出的对话框中，将所需容器组数修改为 2，并单击确定。

   此操作会扩容一个 Pod，将副本数升到 2。

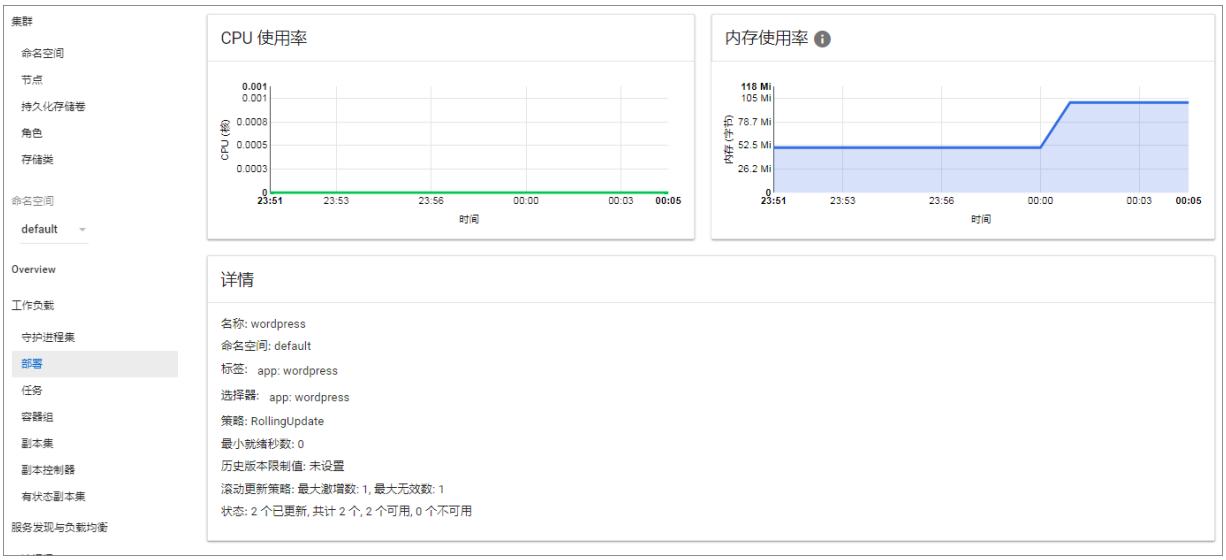
后续操作

您可以通过左侧的图标查看每个 Kubernetes 对象的状态。 表示对象仍然处于部署状态。 表示对象已经完成部署。

应用部署完成后，您可以单击某个部署的名称查看正在运行的 Web 服务的详细信息。您可以查看部署中的副本集以及这些副本集所使用的 CPU 和 Memory 资源的相关信息。

说明：

如果看不到资源，请耐心等待几分钟。



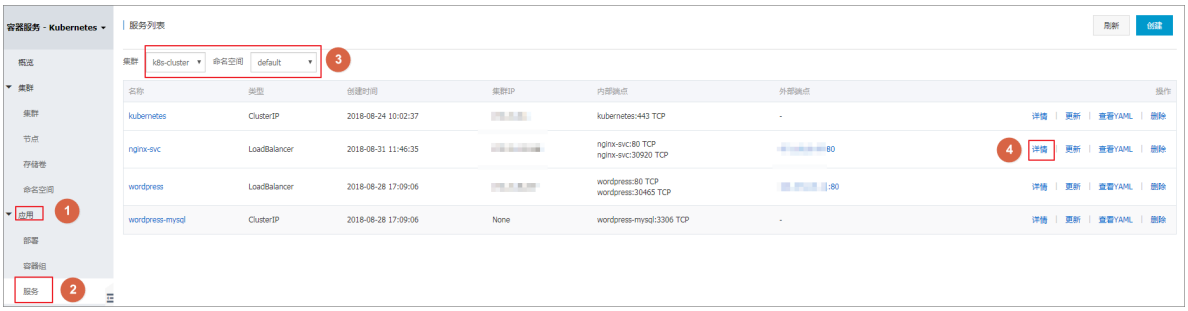
1.6.10 查看服务

背景信息

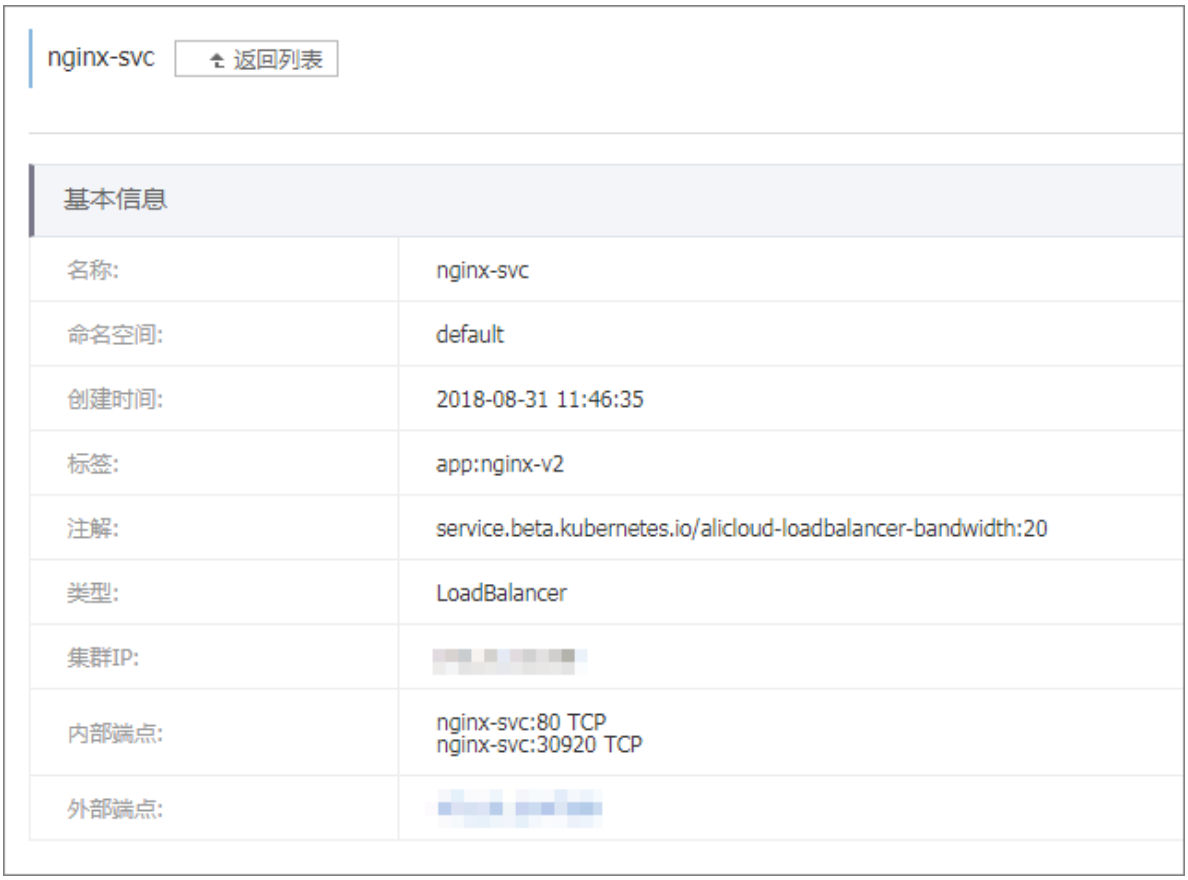
您在创建应用时，如果配置了外部服务，除了运行容器，Kubernetes Dashboard 还会创建外部 Service，用于预配负载均衡器，以便将流量引入到集群中的容器。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 服务，进入服务列表页面。
3. 您可以选择所需的集群和命名空间，选择所需的服务，单击右侧的详情。



您可查看服务的名称、类型、创建时间、集群 IP 以及外部端点等信息。本例中您可看到分配给服务的外部端点（IP 地址）。您可以单击该 IP 地址访问nginx应用。



4. （可选）您也可进入集群的 Kubernetes Dashboard。在左侧导航栏中单击**服务**，查看所有服务。

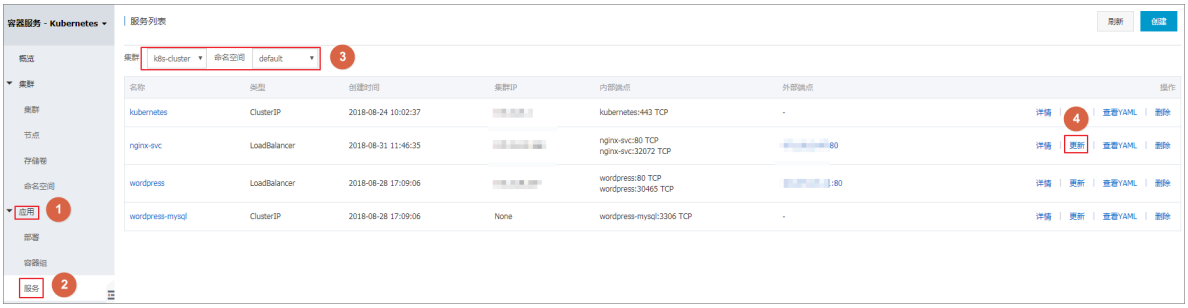
1.6.11 更新服务

您可以通过容器服务管理控制台的服务列表页面或者 Kubernetes Dashboard 变更服务的配置。

通过容器服务控制台服务列表页面

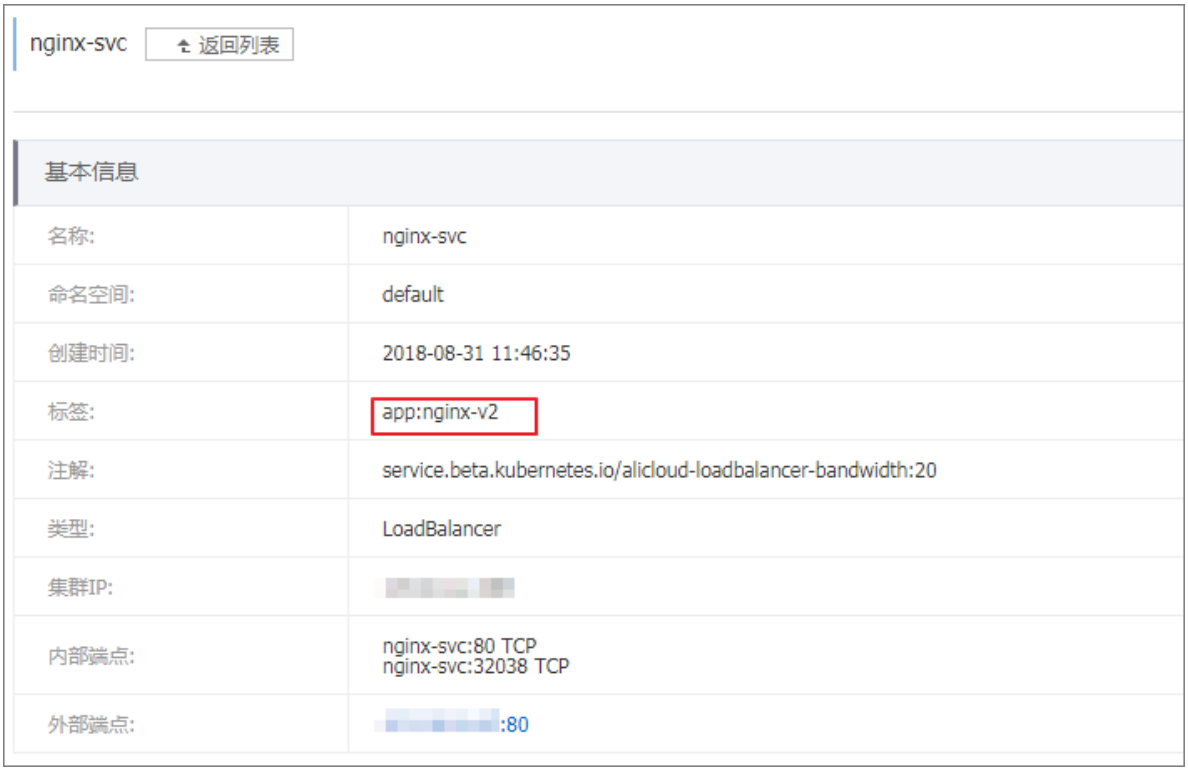
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**应用 > 服务**，进入服务列表页面。

3. 选择集群和命名空间，选择所需的服务（本示例中选择 nginx-svc），单击右侧的更新。



4. 在弹出的更新对话框中，进行配置修改，然后单击确定。

5. 在服务列表中，找到所需的服务，单击右侧的详情，查看服务变化的情况。本例中，对服务的标签进行修改。

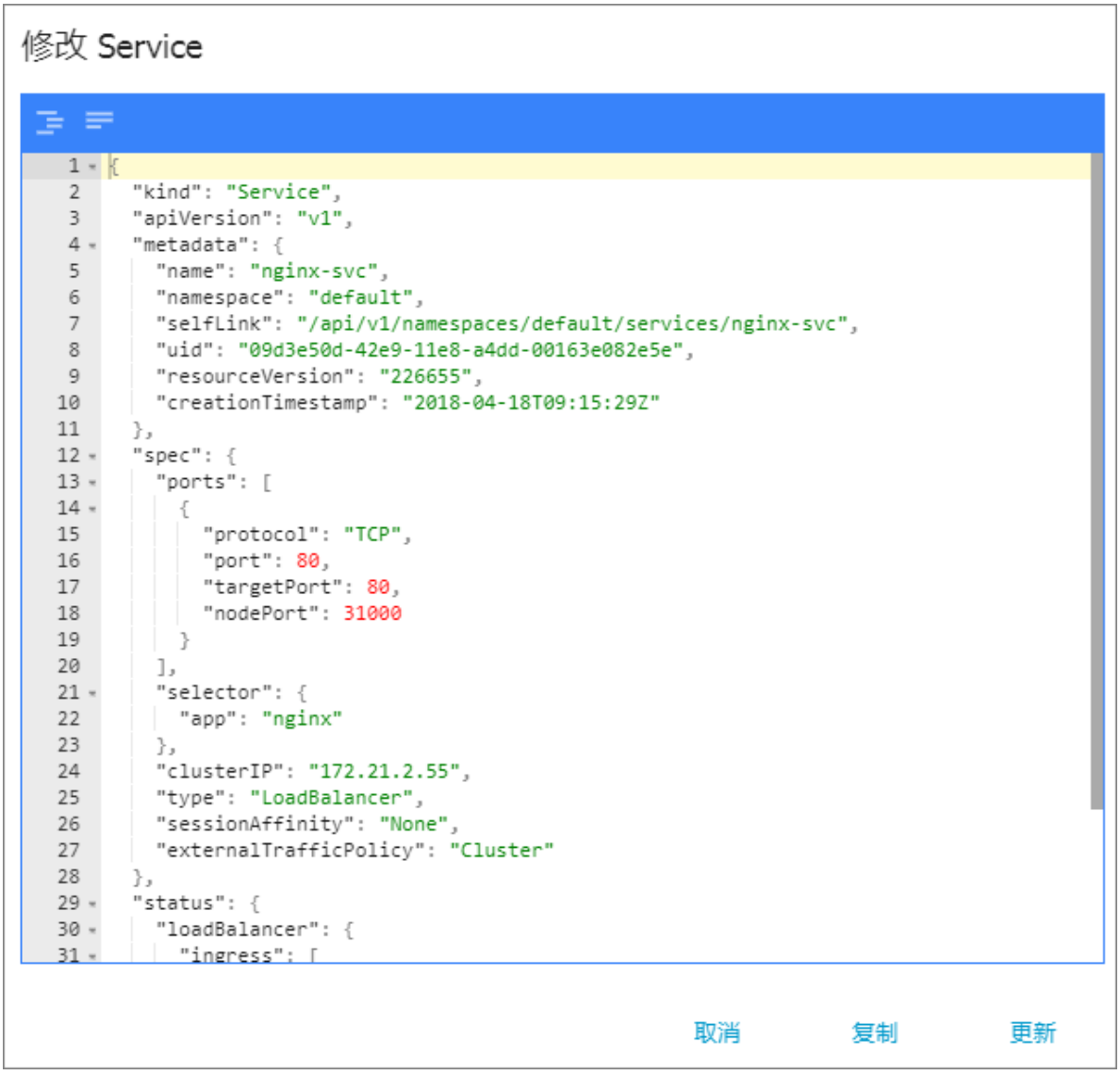


通过 **Kubernetes Dashboard**

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

4. 在 Kubernetes Dashboard 中，选择所需的命名空间，单击左侧导航栏中的服务。
5. 选择所需的服务，单击右侧的操作图标并单击查看/编辑YAML。

6. 在弹出的更新对话框中，修改配置信息，如将 nodePort 修改为 **31000**，然后单击更新。



1.6.12 删除服务

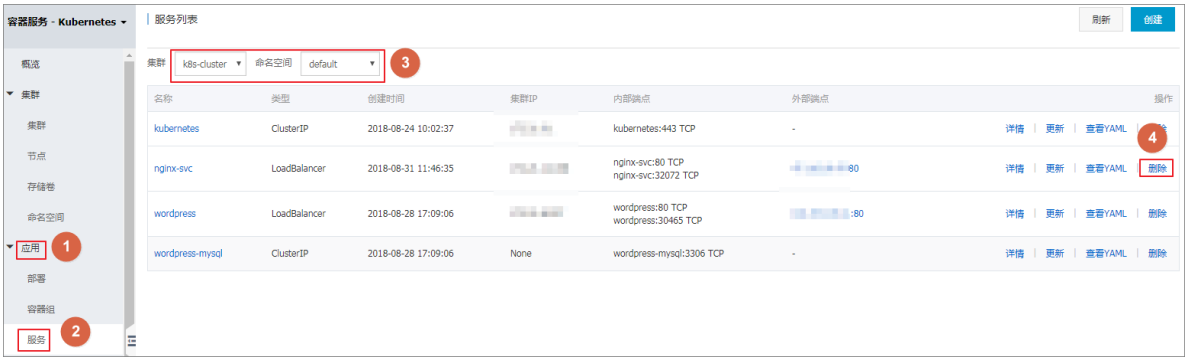
阿里云 Kuberntes 服务提供 Web 界面，让您快速对服务进行删除。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已经成功创建一个服务，参见创建服务。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 服务，进入服务列表页面。
3. 选择集群和命名空间，选择所需的服务（本示例中选择 nginx-svc），单击右侧的删除。



4. 在弹出的窗口中，单击确定，确认删除，该服务在服务列表中消失。

1.6.13 使用应用触发器

阿里云容器服务Kubernetes支持应用触发器的功能，您可通过多种方式使用应用触发器。

前提条件

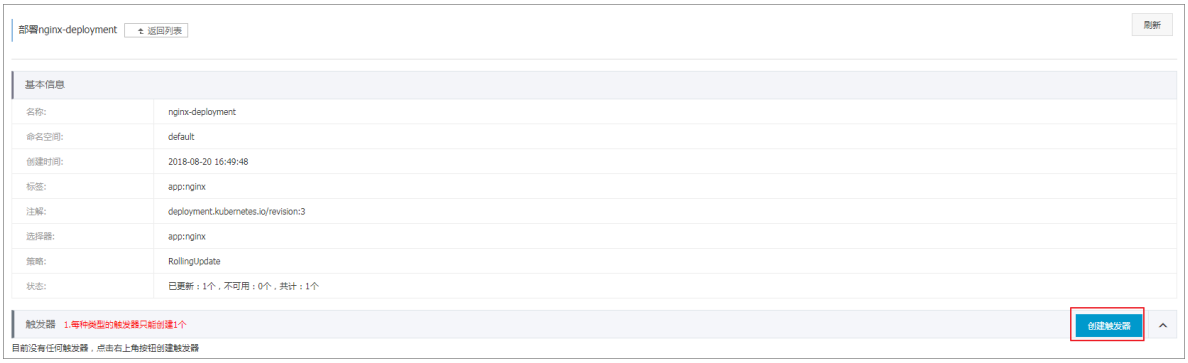
- 您已成功创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已成功创建一个应用，用于创建应用触发器，并测试触发器的作用。本例中创建一个nginx应用。

操作步骤

1. 登录容器服务管理控制台。

2. 单击左侧导航栏的应用 > 部署，选择所需的集群和命名空间，进入部署列表，选择所需的nginx应用，单击右侧的详情。

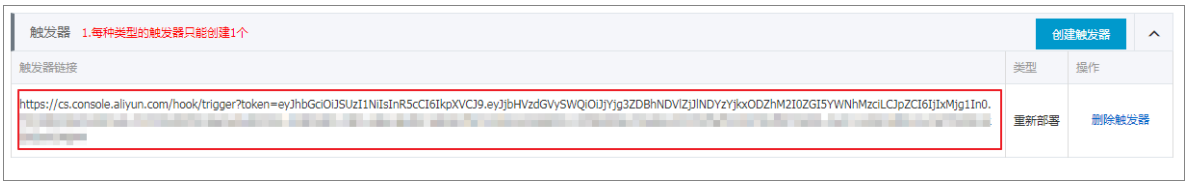
3. 在nginx应用详情页面中，单击触发器栏中右侧的创建触发器。



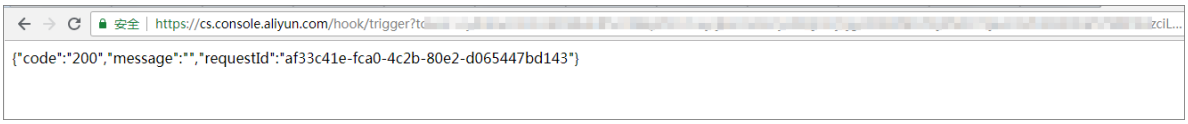
4. 在弹出的对话框中，选择重新部署触发器，然后单击确定。

> 说明：
> 目前仅支持重新部署触发器。

触发器创建完毕后，您可在nginx应用详情页面看到触发器栏中出现一条触发器链接。



5. 复制该触发器，在浏览器中访问。在该页面，会返回一条消息，其中包含请求ID等信息。



6. 返回nginx应用详情页面，您可看到出现一个新的Pod。

等待一段时间，重新部署完毕后，nginx应用将会删除旧Pod，只保留新的Pod。

后续操作

您也可通过三方集成系统进行触发，使用 GET 或者 POST 都可以进行触发，例如使用 **curl** 命令触发。

调用重新部署触发器，如下所示：

```
curl https://cs.console.aliyun.com/hook/trigger?token=xxxxxxx
```

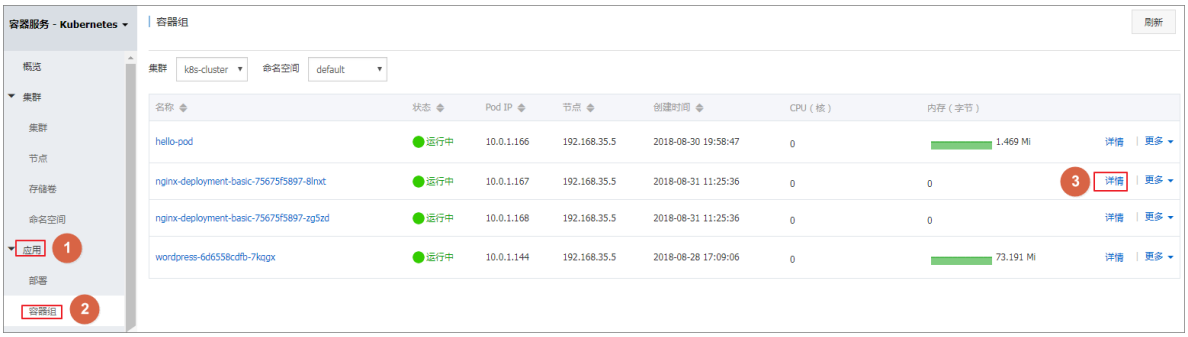
1.6.14 查看容器

您可以通过容器服务管理控制台的容器组页面或者 Kubernetes Dashboard 查看 Kubernetes 集群的容器组页面。

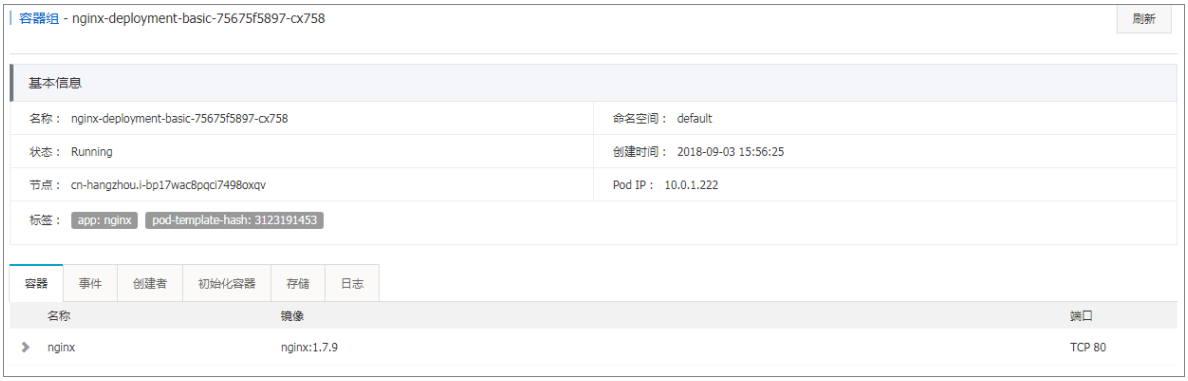
通过容器组页面进行查看

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 容器组，进入容器组页面。
3. 选择所需的集群和命名空间，选择所需的容器组，单击右侧的详情。

> **说明：**
> 您可对容器组进行更新和删除操作，对于通过部署（deployment）创建的容器组，建议您通过 deployment 进行管理。



4. 进入容器组的详情页，您可查看该容器组的详情信息。

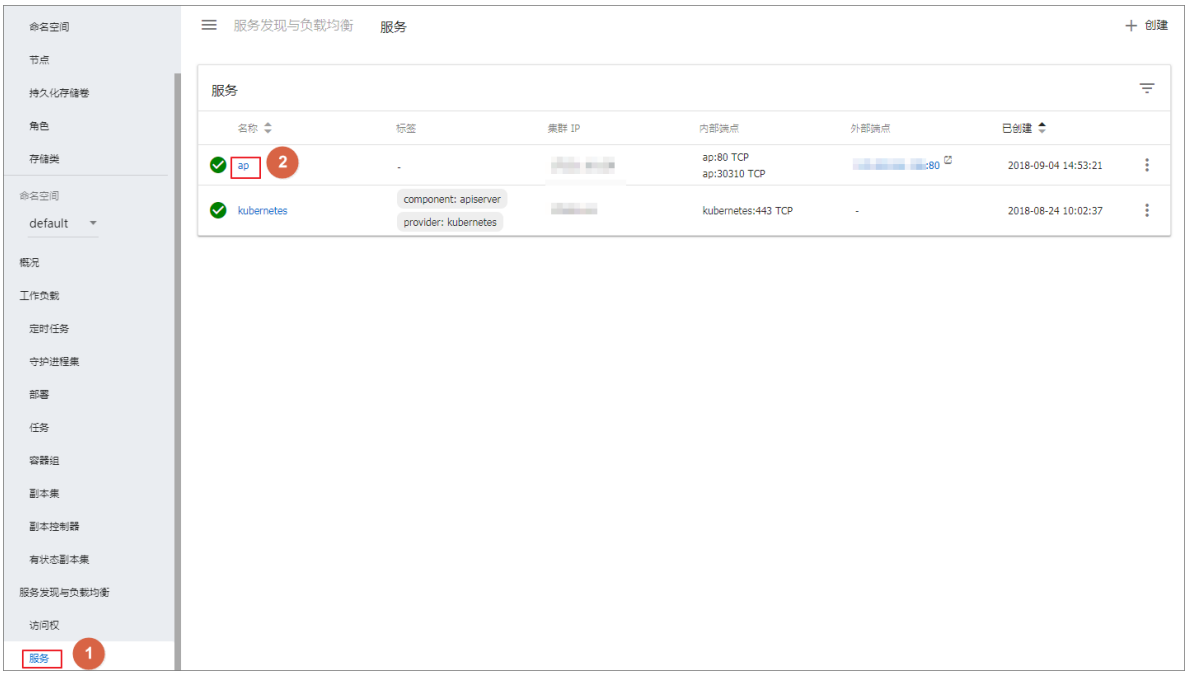


通过 Kubernetes Dashboard 查看容器组

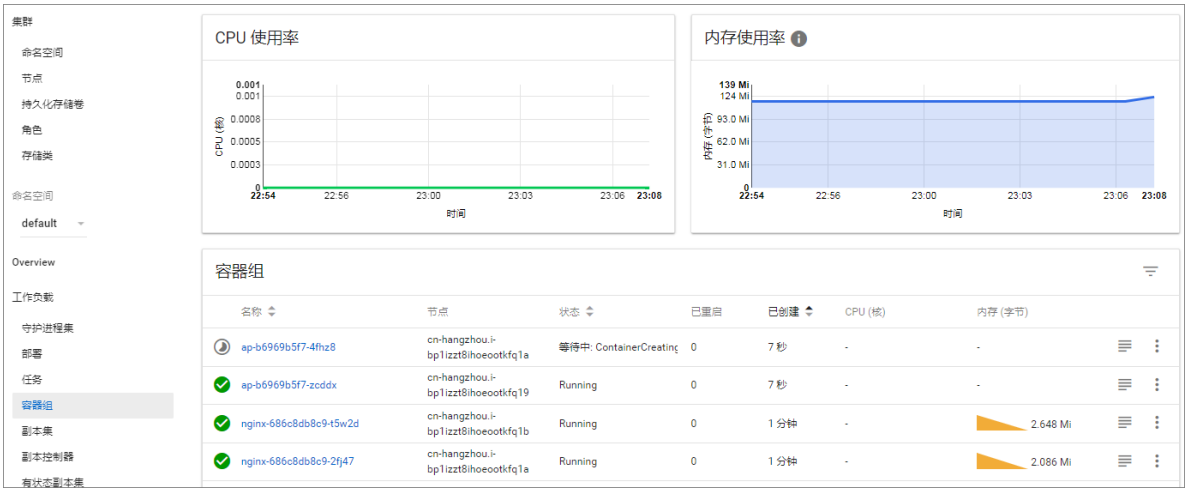
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

4. 单击左侧导航栏中的容器组，查看集群中的容器组。

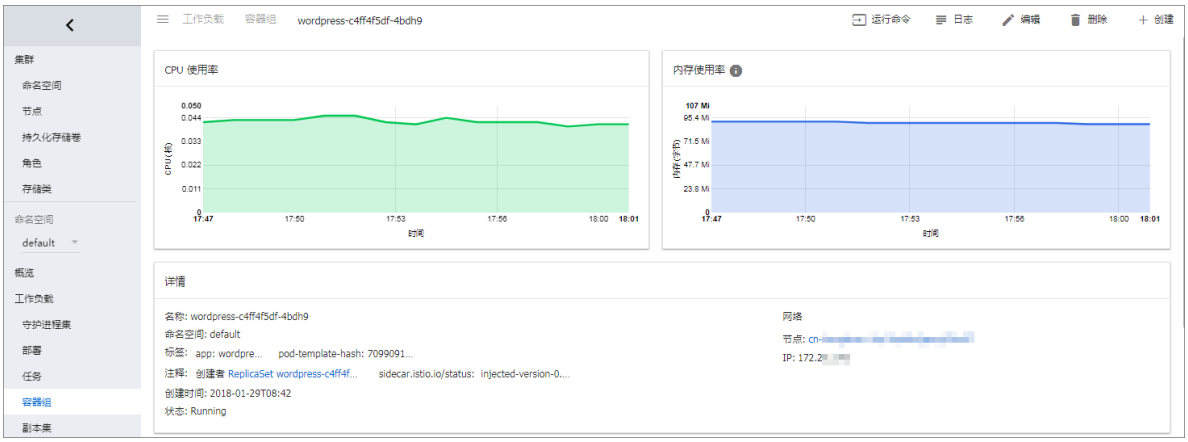
   或者，您可以单击左侧导航栏中的服务，选择所需的服务并单击服务的名称，查看该服务下的容器组。



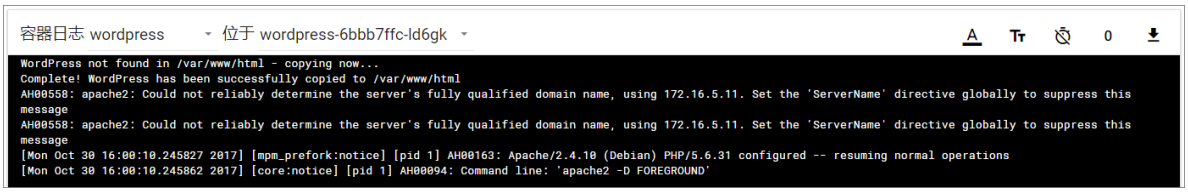
5. 您可以通过左侧的图标查看每个 Kubernetes 对象的状态。 表示对象仍然处于部署状态。 表示对象已经完成部署。



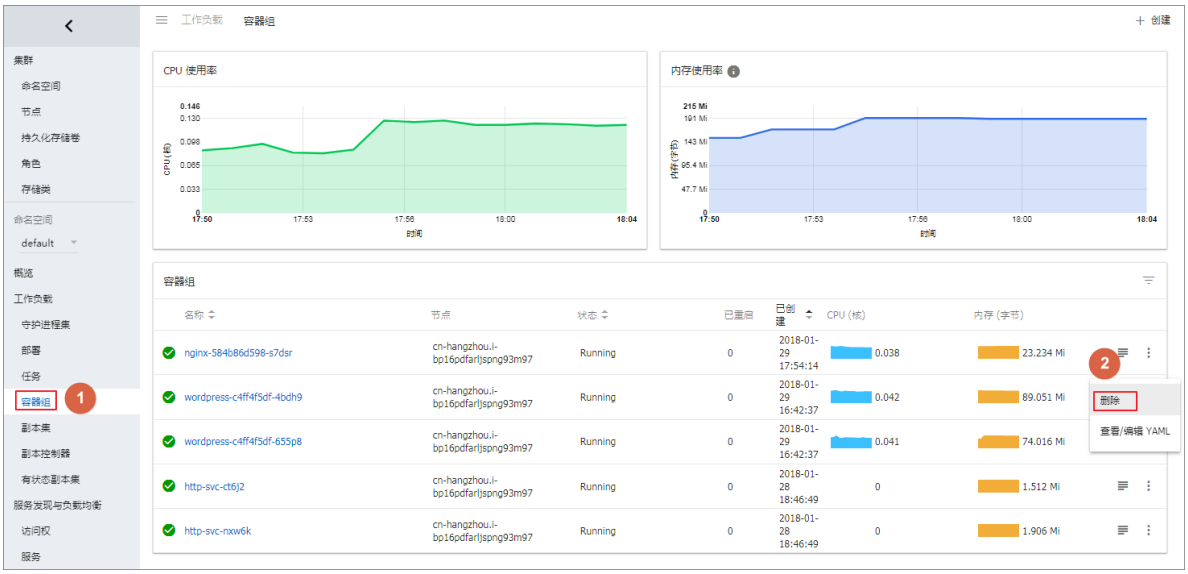
6. 选择所需的容器组，单击容器组的名称，您可以查看容器组的详细信息以及容器组使用的 CPU 和 Memory。



7. 选择所需的容器组，单击右上角的日志查看容器的日志信息。



8. 您可以单击右侧的删除 删除该容器组。

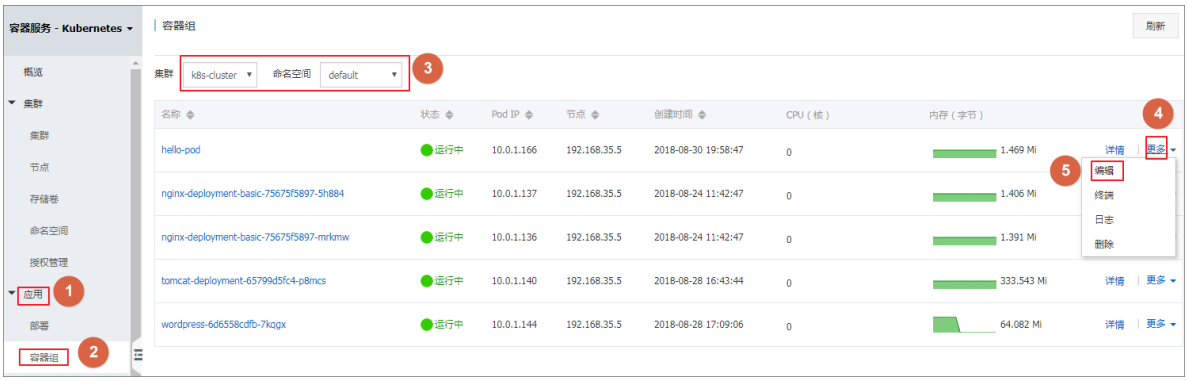


1.6.15 变更容器配置

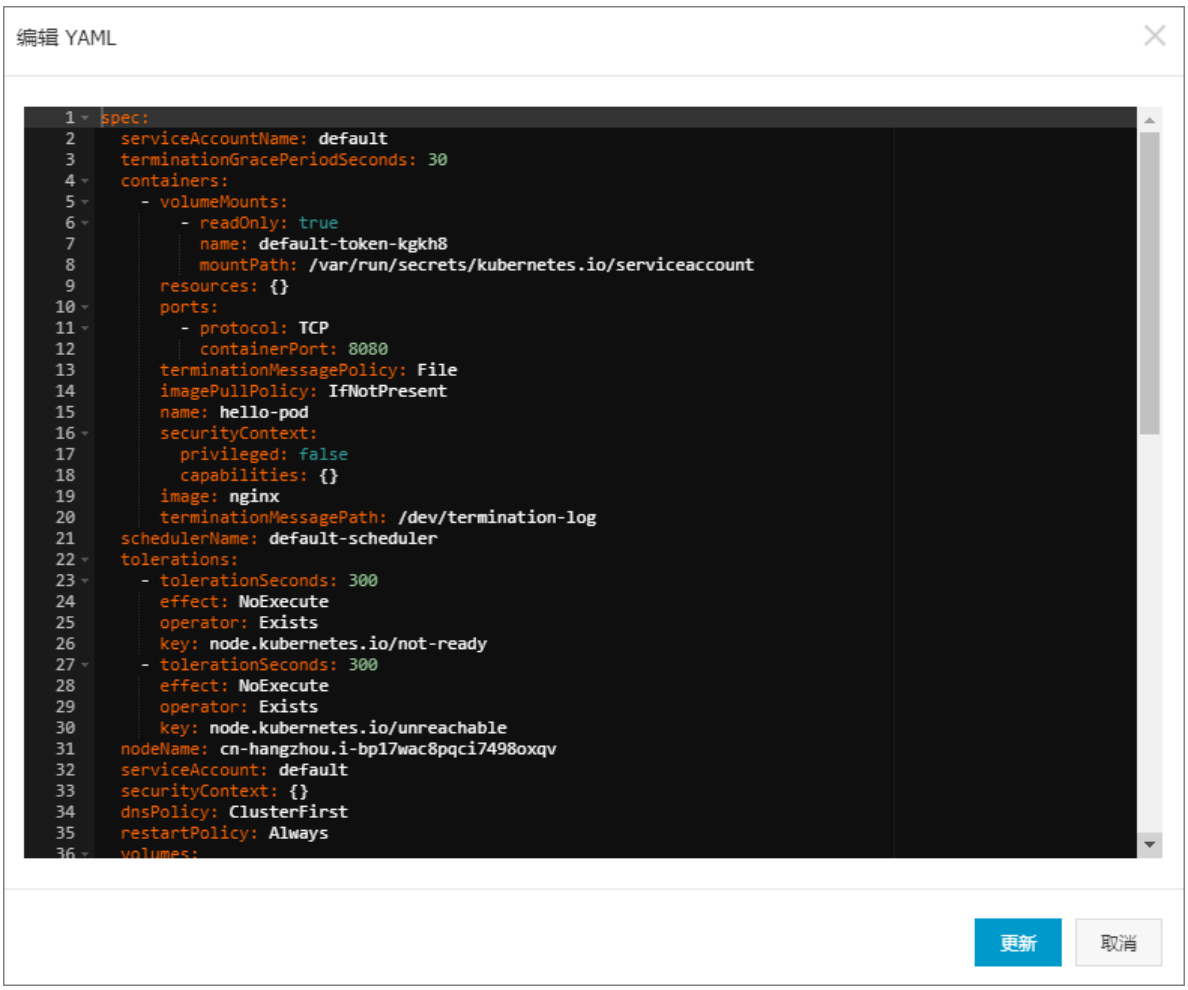
您可以通过容器服务管理控制台变更容器组（Pod）的配置。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 容器组，进入容器组列表。
3. 选择所需的集群和命名空间，选择所需的容器组，单击右侧的更多 > 编辑。



4. 更新容器组的配置并单击更新。



1.6.16 指定节点调度

您可通过为节点设置节点标签，然后通过配置 `nodeSelector` 对 pod 调度进行强制约束，将 pod 调度到指定的 Node 节点上。关于 nodeSelector 的详细实现原理，请参见 *nodeselector*。

出于业务场景需要，如您需要将管控服务部署到 Master 节点；或者将某些服务部署到具有 SSD 盘的机器上。您可以采用这种方式实现指定节点调度。

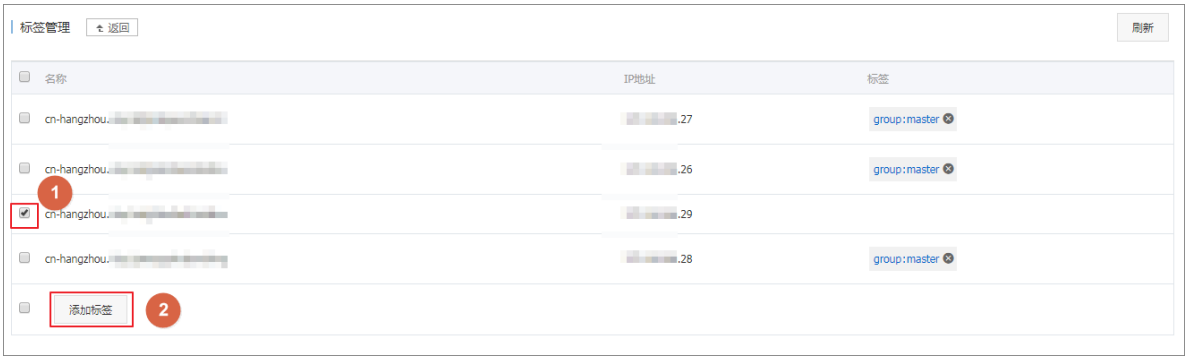
前提条件

您已经成功部署一个 Kubernetes 集群，参见创建*Kubernetes*集群。

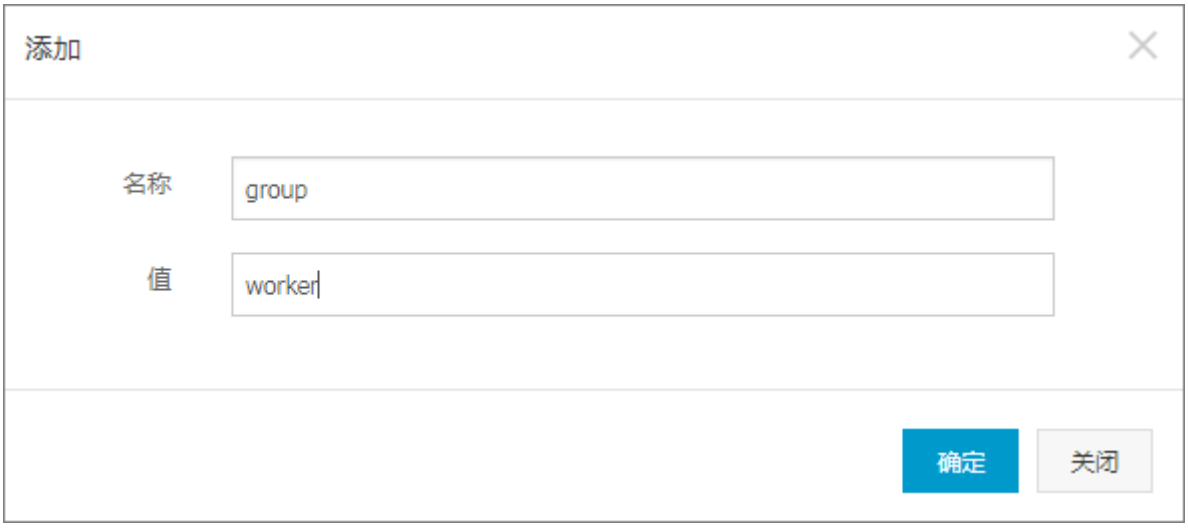
步骤1 为节点添加标签

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**集群 > 节点** ，进入节点列表页面。
3. 选择所需的集群，在页面右上角单击**标签管理**。

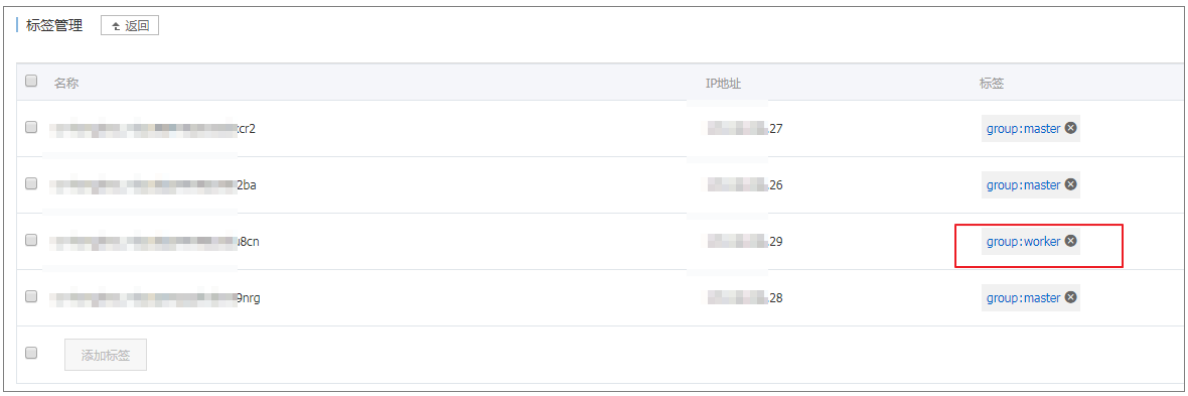
4. 在节点列表中，选择所需节点，然后单击**添加标签**。本例中选择一个 worker 节点。



5. 在弹出的添加标签对话框中，输入标签的名称和值，然后单击**确定**。



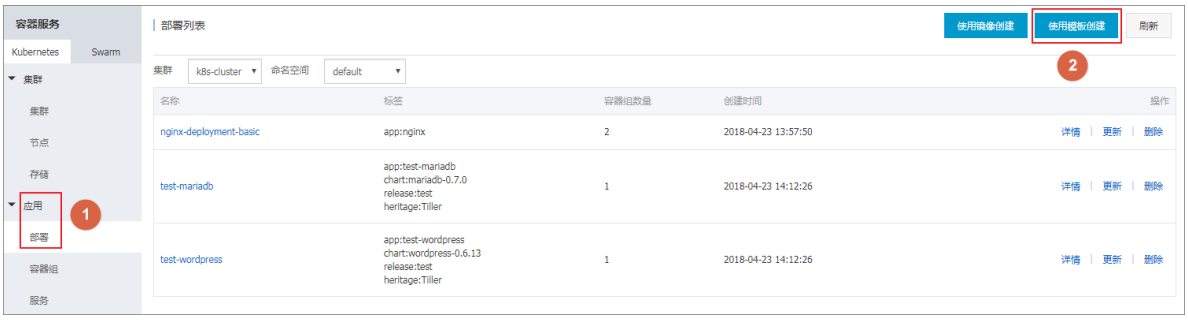
您可以在标签管理页面，看到该节点具有 `group:worker`标签。



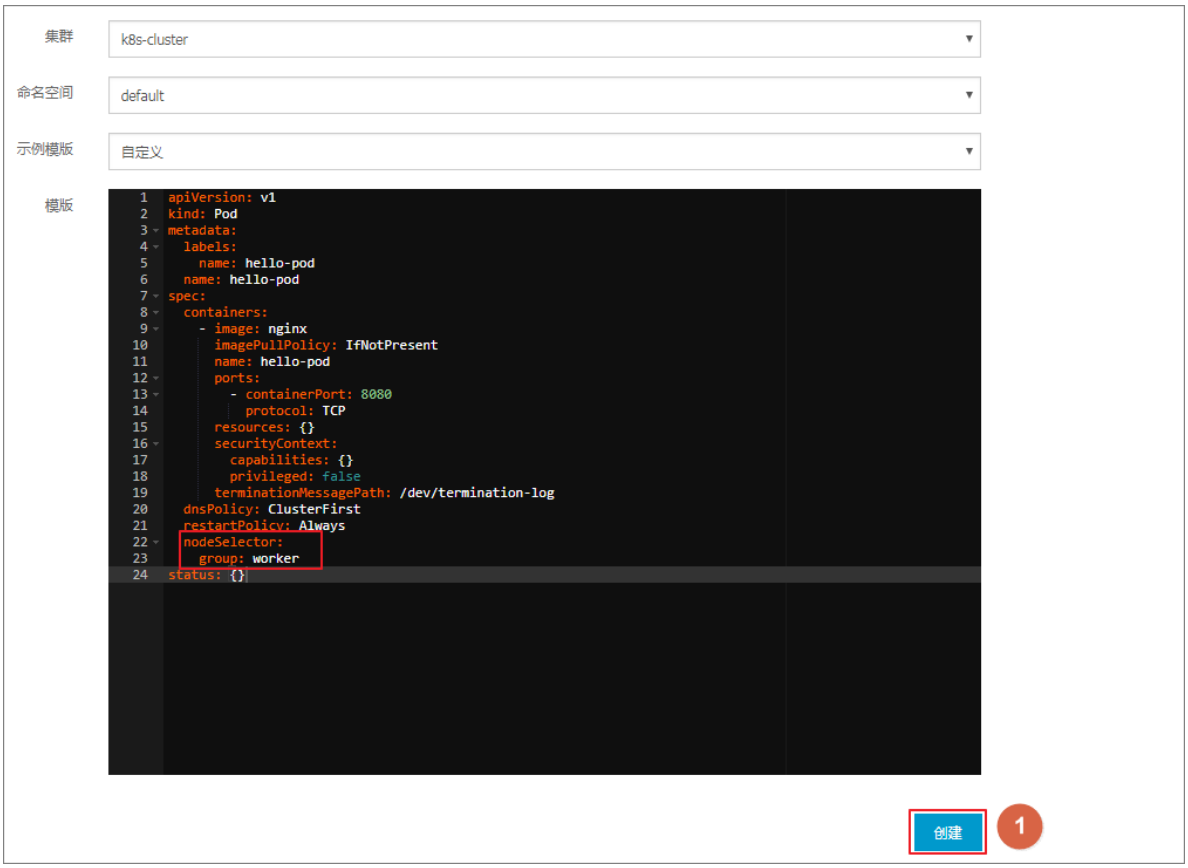
您也可以通过命令 `kubectl label nodes <node-name> <label-key>=<label-value>` 为节点添加标签。

步骤2 将 pod 部署到指定节点

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署 ，进入部署列表页面。
3. 单击页面右上角的使用模板创建。



4. 对模板进行相关配置，部署一个 pod，完成配置后，单击创建。
   - 集群：选择所需的集群。
   - 命名空间：选择资源对象所属的命名空间，本例中是 default。
   - 示例模板：本示例选择自定义模板。

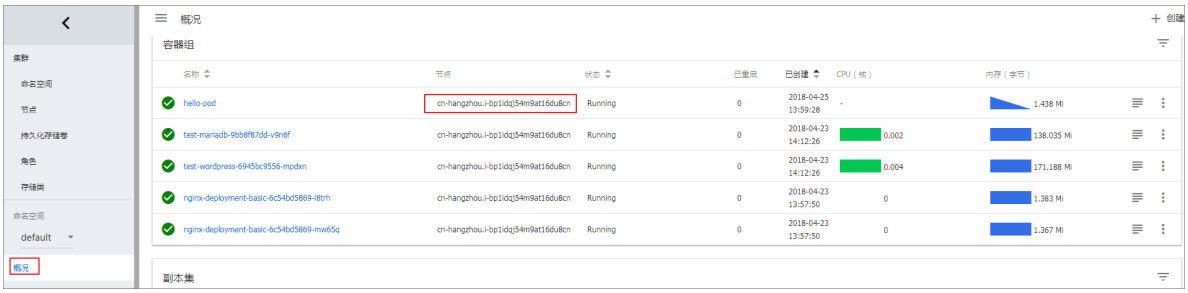


本示例的编排模板如下。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: hello-pod
  name: hello-pod
spec:
  containers:
    - image: nginx
      imagePullPolicy: IfNotPresent
      name: hello-pod
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      securityContext:
        capabilities: {}
        privileged: false
      terminationMessagePath: /dev/termination-log
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  nodeSelector:
    group: worker                                    ##注意与前面配置的节点标签一致
```

```
status: {}
```

5. 单击创建 后，会提示部署状态信息。 成功后，单击**Kubernetes 控制台**前往控制台查看部署状态。



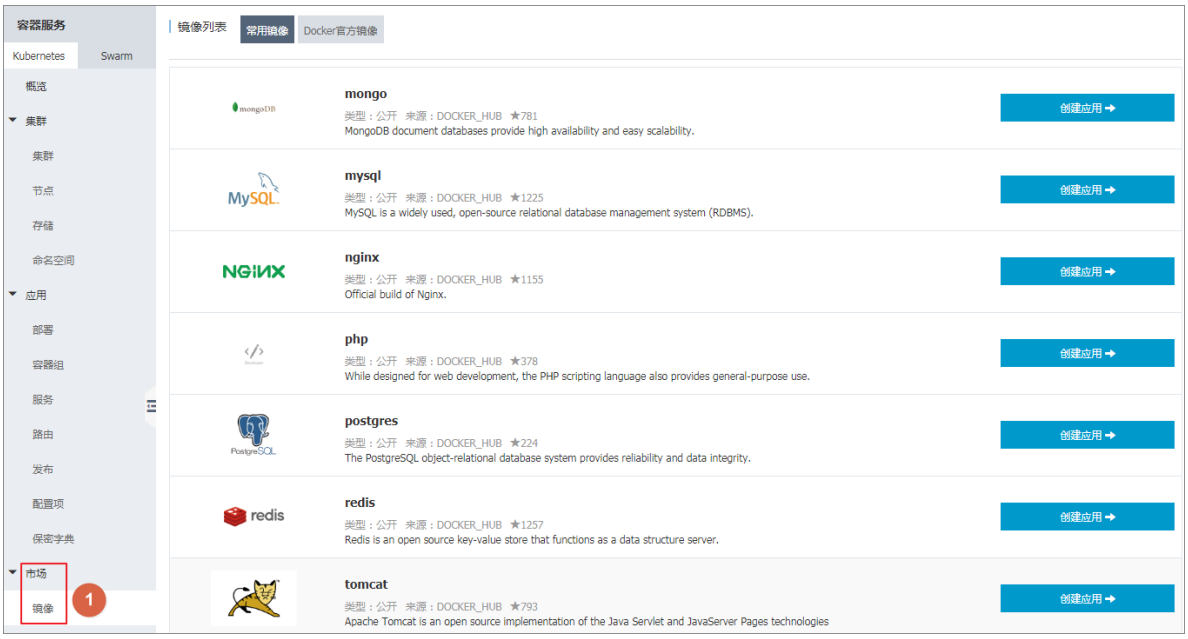
6. 您可单击 pod 名称，进入详情页，了解 pod 详情。

   您可看到 pod 的标签、所处的节点 ID 等信息，表明该 pod 已经成功部署到具有 `group: worker` 标签的指定节点上。

1.6.17 查看镜像列表

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中**市场 > 镜像**，进入镜像列表页面。



您可以查看镜像的种类过。

- 常用镜像： 容器服务推荐的一些常用镜像。
- **Docker** 官方镜像： Docker Hub 提供的官方镜像。

1.7 网络管理

1.7.1 概述

本文介绍阿里云容器服务Kubernetes支持的网络类型。

容器网络

容器服务通过将Kubernetes网络和阿里云VPC的深度集成，提供了稳定高性能的容器网络。在容器服务中，支持以下类型的互联互通：

- 同一个容器集群中，Pod之间相互访问。
- 同一个容器集群中，Pod访问Service。
- 同一个容器集群中，ECS访问Service。
- Pod直接访问同一个VPC下的ECS(*)。
- 同一个VPC下的ECS直接访问Pod(*)。

说明：

* 需要正确设置安全组规则。

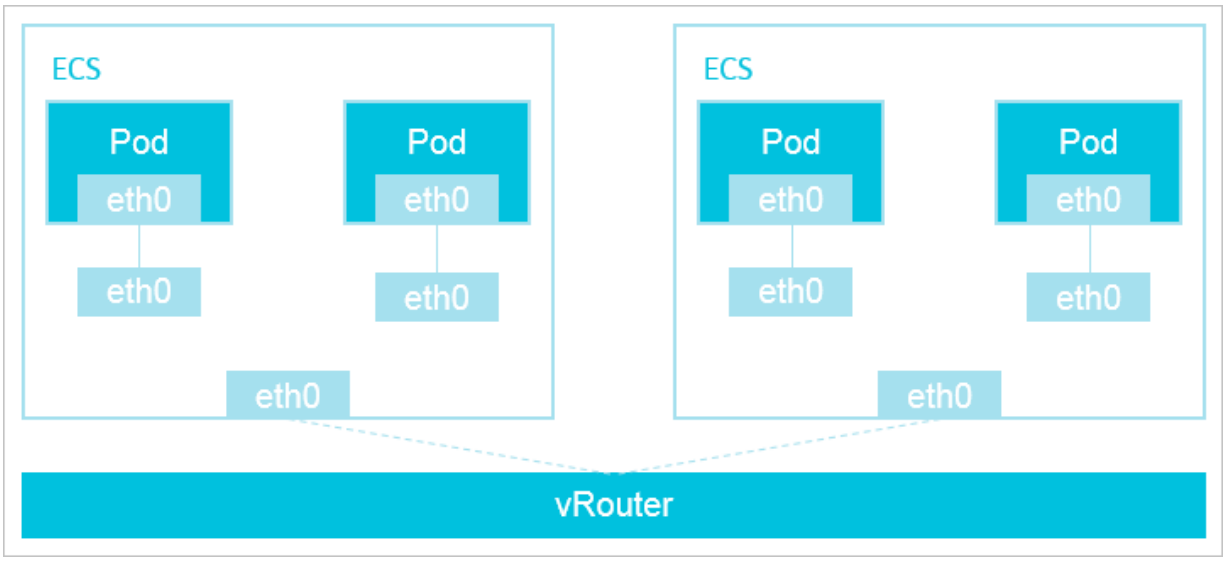
1.7.2 Terway网络插件

本文介绍如何使用阿里云容器服务Kubernetes集群的Terway网络插件。

Terway网络插件

Terway网络插件是阿里云容器服务自研的网络插件，功能上完全兼容Flannel：

- 支持将阿里云的弹性网卡分配给容器。
- 支持基于Kubernetes标准的NetworkPolicy来定义容器间的访问策略，兼容Calico的Network Policy。



在Terway网络插件中，每个Pod拥有自己网络栈和IP地址。同一台ECS内的Pod之间通信，直接通过机器内部的转发，跨ECS的Pod通信，报文通过VPC的vRouter转发。由于不需要使用VxLAN等的隧道技术封装报文，因此具有较高的通信性能。

使用Terway网络插件

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群，进入集群列表页面。
3. 单击页面右上角的创建 **Kubernetes** 集群，进入创建 **Kubernetes** 集群页面。

默认进入**kubernetes**集群配置页面。

> 说明：
> 本文以创建经典Dedicated Kubernetes模式为例，具体可参考创建*Kubernetes*集群。

4. 设置启动用的网络插件，选择**Terway**。

Flannel与Terway

在创建Kubernetes集群时，阿里云容器服务提供两种网络插件：Terway和Flannel：

> 说明：
> 如何选择，可参考如何选择*Kubernetes*集群网络插件*#Terway*和*Flannel*

- Flannel：使用的是简单稳定的社区的*Flannel* CNI插件，配合阿里云的VPC的高速网络，能给集群高性能和稳定的容器网络体验，但功能偏简单，支持的特性少，例如：不支持基于Kubernetes标准的Network Policy。
- Terway：是阿里云容器服务自研的网络插件，功能上完全兼容Flannel，支持将阿里云的弹性网卡分配给容器，支持基于Kubernetes标准的NetworkPolicy来定义容器间的访问策略，支持对单个容器做带宽的限流。对于不需要使用Network Policy的用户，可以选择Flannel，其他情况建议选择Terway。

> 说明：
> - Terway的Network Policy是通过集成Calico的Felix组件实现的，因此Network Policy的能力和Calico完全一致。对于最初为了使用Calico而自建集群的客户，目前可以通过Terway转换到容器服务Kubernetes上来。
> - Terway目前集成的Felix版本为2.6.6。

1.7.3 为Pod分配ENI网卡

本文介绍如何将ENI网卡分配给Pod。

背景信息

- 创建Kubernetes集群时，网络插件请选择**Terway**，可参考创建*Kubernetes*集群。
- 对于已经选择的Terway网络插件创建的集群，请升级Terway版本至v1.0.0.1及以上版本。

> 说明：
>
> 1. 登录容器服务管理控制台，在 Kubernetes 菜单下，选择集群，进入集群列表页面。
> 2. 选择目标集群，单击操作列更多 > 系统组件升级。
> 3. 在系统组件升级页面，查看Terway当前版本。
> 4. 根据当前版本及可升级版本，判断是否需要升级。如需升级，请单击操作列的升级。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

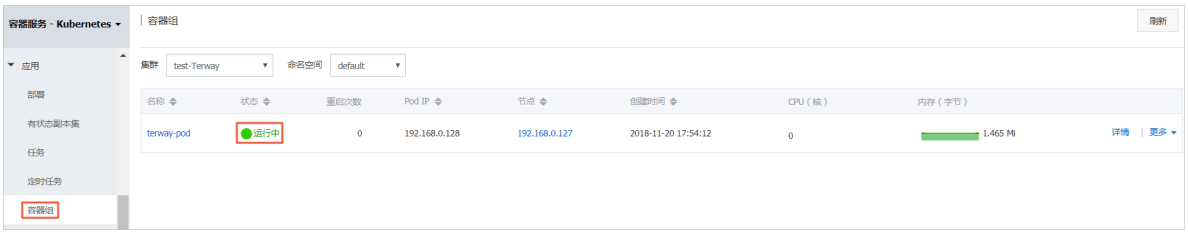
   您可以使用如下 yaml 示例模板创建Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: terway-pod
  labels:
    app: nginx
spec:
  containers:
```

```
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    resources:
      limits:
        aliyun/eni: 1
```

预期结果

1. 在 Kubernetes 菜单下，单击左侧导航栏中的 应用 > 容器组，可以看到名称为**terway-pod**的pod已部署成功。



2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入进群列表页面。
3. 选择目标集群，单击集群名称，查看集群详细信息。
4. 在集群资源区域，单击虚拟专有网络 **VPC**查看集群的VPC网段。
5. 执行以下命令，获取已部署Pod的IP地址，且此IP地址在集群VPC网段内。

```
$ kubectl get pod -o wide
```

1.7.4 使用Network Policy

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您在创建集群时，网络插件选择**Terway**，参见创建*Kubernetes*集群。
- 您可以通过Kubectl连接到Kubernetes 集群，参见通过 *kubectl* 连接 *Kubernetes* 集群。

测试nginx服务可以被其他pod正常访问

1. 执行以下命令，创建一个nginx的应用，并通过名称为ngnix的Service将其暴露。

```
$ kubectl run nginx --image=nginx
deployment.apps/nginx created
$ kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-64f497f8fd-znbxb   1/1     Running   0          45s

$ kubectl expose deployment nginx --port=80
service/nginx exposed
$ kubectl get service
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
```

```
kubernetes   ClusterIP   172.19.0.1    <none>        443/TCP   3h
nginx        ClusterIP   172.19.8.48   <none>        80/TCP    10s
```

2. 执行以下命令，创建名称为busybox的Pod，访问步骤1创建的nginx服务。

```
$ kubectl run busybox --rm -ti --image=busybox /bin/sh
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and
will be removed in a future version. Use kubectl create instead.
If you don't see a command prompt, try pressing enter.
/ # wget nginx
Connecting to nginx (172.19.8.48:80)
index.html           100% |
**********************************************************************
   612  0:00:00 ETA
/ #
```

通过Network Policy限制服务只能被带有特定label的应用访问

1. 执行以下命令，创建*policy.yaml*文件。

```
$ vim policy.yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
```

2. 执行以下命令，根据步骤1的*policy.yaml*文件创建Network Policy。

```
$ kubectl apply -f policy.yaml
networkpolicy.networking.k8s.io/access-nginx created
```

3. 执行以下命令，当没有定义访问标签时，测试访问nginx服务，请求会超时，无法访问。

```
$ kubectl run busybox --rm -ti --image=busybox /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget nginx
Connecting to nginx (172.19.8.48:80)
wget: can't connect to remote host (172.19.8.48): Connection timed
out
/ #
```

4. 执行以下命令，定义访问标签，测试访问nginx服务，请求成功，正常访问。

```
$ kubectl run busybox --rm -ti --labels="access=true" --image=
busybox /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget nginx
Connecting to nginx (172.19.8.48:80)
```

```
index.html           100% |
*********************************************************************************
   612  0:00:00 ETA
/ #
```

通过Network Policy限制能够访问暴露了公网SLB服务的来源IP段

1. 执行以下命令，为上述nginx应用创建阿里云负载均衡服务，指定 `type=LoadBalancer` 来向外网用户暴露 nginx 服务。

```
$ vim nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nginx
  name: nginx-slb
spec:
  externalTrafficPolicy: Local
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer

$ kubectl apply -f nginx-service.yaml
service/nginx-slb created

$ kubectl get service nginx-slb
NAME        TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)
      AGE
nginx-slb   LoadBalancer   172.19.12.254   47.110.200.119   80:32240
/TCP   8m
```

2. 执行以下命令，访问刚刚创建的SLB的IP地址47.110.200.119，访问失败。

```
$ wget 47.110.200.119
--2018-11-21 11:46:05--  http://47.110.200.119/
Connecting to 47.110.200.119:80... failed: Connection refused.
```

说明：

访问失败的原因是：

- 我们刚刚配置了nginx服务只能被带有特定label即`access=true`的应用访问。
- 访问SLB的IP地址，是从外部访问Kubernetes，与通过*Network Policy*限制服务只能被带有特定*label*的应用访问不同。

解决方法：修改Network Policy，增加允许访问的来源IP地址段。

3. 执行以下命令，查看本地的IP地址。

```
$ curl myip.ipip.net

当前 IP：10.0.0.1 来自于：中国 北京 北京          #此处仅为示例，具体请以实际设备为准
```

4. 执行以下命令，修改已经创建的*policy.yaml*文件。

```
$ vim policy.yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
    - ipBlock:
        cidr: 100.64.0.0/10
    - ipBlock:
        cidr: 10.0.0.1/24       #本地IP地址，此处仅为示例，具体请以实际设备为准

$ kubectl apply -f policy.yaml
networkpolicy.networking.k8s.io/access-nginx unchanged
```

说明：

- 有些网络的出口有多个IP地址，这里请使用/24的地址范围。
- SLB健康检查地址在100.64.0.0/10地址段内，因此请务必配置100.64.0.0/10。

5. 执行以下命令，测试访问nginx服务，访问成功。

```
$  kubectl run busybox --rm -ti --labels="access=true" --image=
busybox /bin/sh

If you don't see a command prompt, try pressing enter.
/ # wget 47.110.200.119
Connecting to 47.110.200.119 (47.110.200.119:80)
index.html           100% |
**************************************************|   612
0:00:00 ETA
```

```
/ #
```

通过Network Policy限制一个Pod只能访问www.aliyun.com

1. 执行以下命令，获取*www.aliyun.com*域名解析到的IP地址列表。

```
$ dig +short www.aliyun.com
www-jp-de-intl-adns.aliyun.com.
www-jp-de-intl-adns.aliyun.com.gds.alibabadns.com.
v6wagbridge.aliyun.com.
v6wagbridge.aliyun.com.gds.alibabadns.com.
106.11.93.21
140.205.32.4
140.205.230.13
140.205.34.3
```

2. 执行以下命令，创建*busybox-policy*文件。

```
$ vim busybox-policy.yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: busybox-policy
spec:
  podSelector:
    matchLabels:
      run: busybox
  egress:
  - to:
    - ipBlock:
        cidr: 106.11.93.21/32
    - ipBlock:
        cidr: 140.205.32.4/32
    - ipBlock:
        cidr: 140.205.230.13/32
    - ipBlock:
        cidr: 140.205.34.3/32
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
    ports:
    - protocol: UDP
      port: 53
```

说明：

在*busybox-policy*文件中，配置了Egress，即出网规则，限制应用的对外访问。在这里需配置允许UDP请求，否则无法做DNS解析。

3. 执行以下命令，根据*busybox-policy*文件创建Network Policy。

```
$ kubectl apply -f busybox-policy.yaml
```

```
networkpolicy.networking.k8s.io/busybox-policy created
```

4. 执行以下命令，访问*www.aliyun.com*之外的网站，例如：*www.google.com*，访问失败。

```
$ kubectl run busybox --rm -ti --image=busybox /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget www.google.com
Connecting to www.google.com (64.13.192.74:80)
wget: can't connect to remote host (64.13.192.74): Connection timed
out
```

5. 执行以下命令，访问*www.aliyun.com*，访问成功。

```
/ # wget www.aliyun.com
Connecting to www.aliyun.com (140.205.34.3:80)
Connecting to www.aliyun.com (140.205.34.3:443)
wget: note: TLS certificate validation not implemented
index.html           100% |
**********************************************************|  462k
0:00:00 ETA
/ #
```

1.8 负载均衡及路由管理

1.8.1 概述

Kubernetes 集群提供了多种访问容器应用的方式，支持通过阿里云的负载均衡服务（Server Load Balancer）或者 Ingress 方式来访问内部服务的和实现负载均衡。

1.8.2 通过负载均衡（Server Load Balancer）访问服务

您可以使用阿里云负载均衡来访问服务。

背景信息

如果您的集群的cloud-controller-manager版本大于等于v1.9.3，对于指定已有SLB的时候，系统默认不再为该SLB处理监听，用户需要手动配置该SLB的监听规则。

执行以下命令，可查看cloud-controller-manager的版本。

```
root@master # kubectl get po -n kube-system -o yaml|grep image:|grep
cloud-con|uniq

  image: registry-vpc.cn-hangzhou.aliyuncs.com/acs/cloud-controller-
manager-amd64:v1.9.3
```

通过命令行操作

方法一：

1. 通过命令行工具创建一个 Nginx 应用。

```
root@master # kubectl run nginx --image=registry.aliyuncs.com/acs/
netdia:latest
root@master # kubectl get po
NAME                                READY     STATUS    RESTARTS
  AGE
nginx-2721357637-dvwq3              1/1       Running   1
  6s
```

2. 为 Nginx 应用创建阿里云负载均衡服务，指定 `type=LoadBalancer` 来向外网用户暴露 Nginx 服务。

```
root@master # kubectl expose deployment nginx --port=80 --target-
port=80 --type=LoadBalancer
root@master # kubectl get svc
NAME                CLUSTER-IP      EXTERNAL-IP      PORT(S)
                 AGE
nginx               172.19.10.209   101.37.192.20   80:31891/TCP
                4s
```

3. 在浏览器中访问 `http://101.37.192.20`，来访问您的 Nginx 服务。

方法二

1. 将下面的 yml code 保存到 `nginx-svc.yml`文件中。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nignx
  name: nginx-01
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

2. 执行`kubectl apply -f nginx-svc.yml`命令。

```
root@master # kubectl apply -f nginx-svc.yml
root@master # kubectl get service
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(
S)        AGE9d
ngi-01nx       LoadBalancer   172.19.9.243    101.37.192.129   80:
32325/TCP   3h
```

3. 在浏览器中访问 `http://101.37.192.129`，来访问您的 Nginx 服务。

通过 Kubernetes Dashboard 操作

1. 将下面的 yml code 保存到 `nginx-svc.yml`文件中。

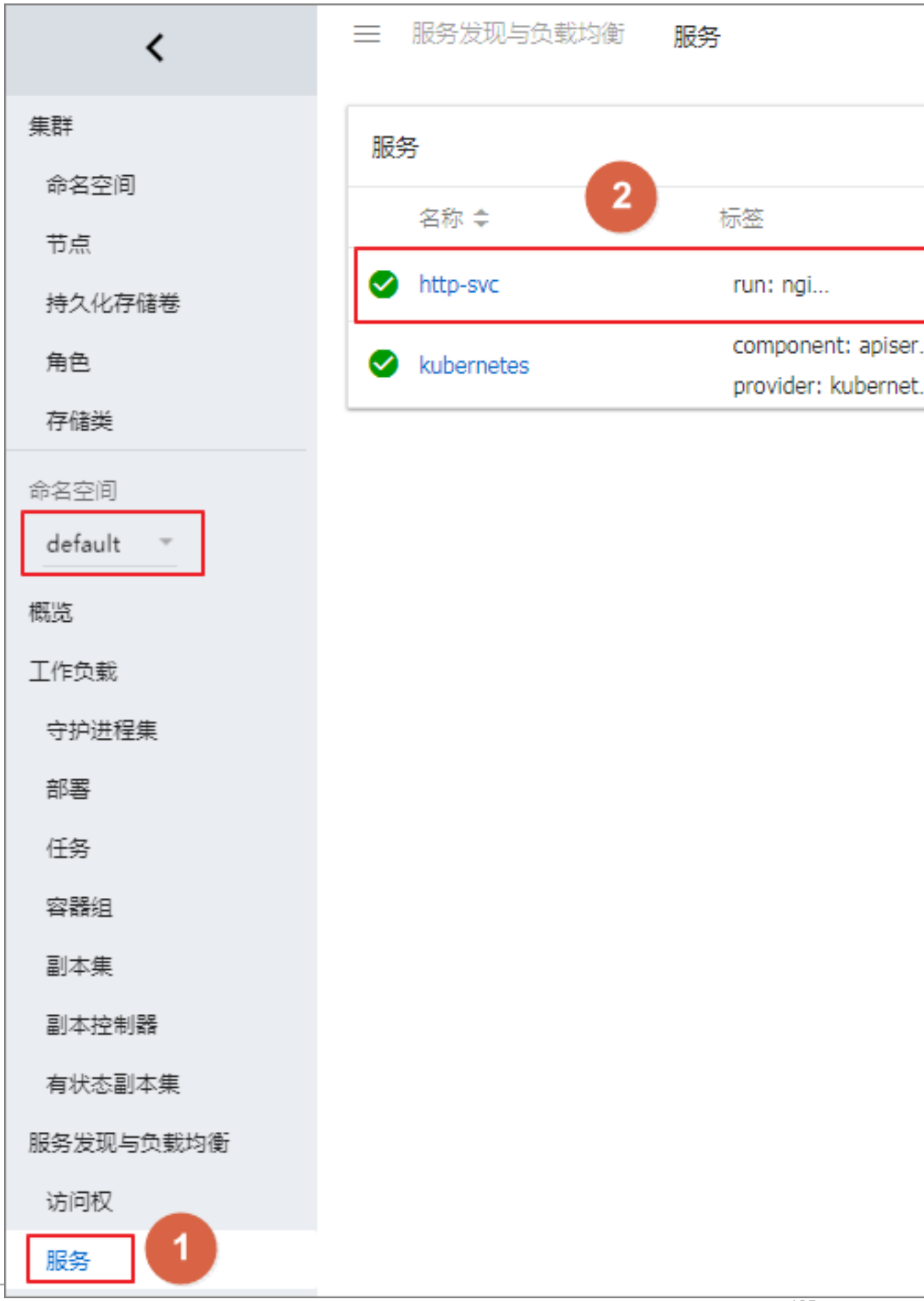
```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nginx
  name: http-svc
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

2. 登录容器服务管理控制台，单击目标集群右侧的控制台，进入 Kubernetes Dashboard 页面。
3. 单击创建，开始创建应用。

4. 单击使用文件创建。选择刚才保存的*nginx-svc.yml* 文件
5. 单击上传。

   这样会创建一个阿里云负载均衡实例指向创建的 Nginx 应用，服务的名称为 `http-svc`。
6. 在 Kubernetes Dashboard 上定位到 default 命名空间，选择服务。

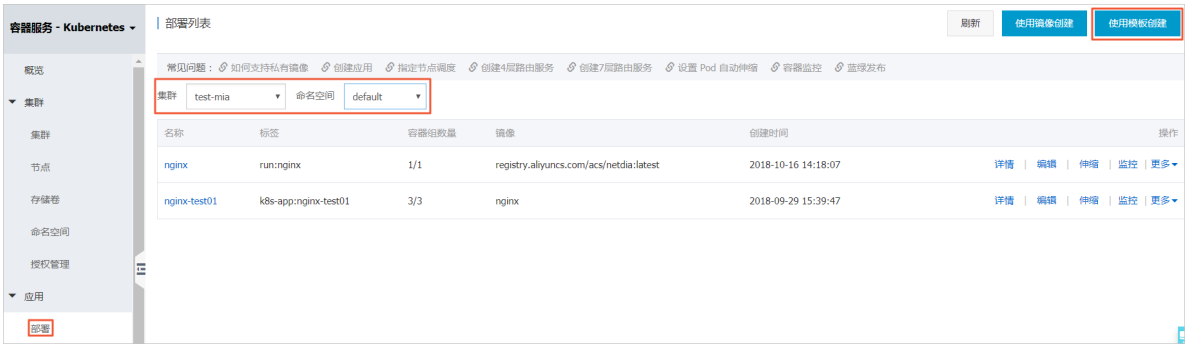
   可以看到刚刚创建的 `http-svc` 的 Nginx 服务和机器的负载均衡地址 `http://114.55.79.24:80`。


集群
命名空间
节点
持久化存储卷
角色
存储类
命名空间
default
概览
工作负载
守护进程集
部署
任务
容器组
副本集
副本控制器
有状态副本集
服务发现与负载均衡
访问权
服务
1
服务发现与负载均衡
服务
服务
2
名称
标签
http-svc
run: ngi...
kubernetes
component: apiser.
provider: kubernet.

7. 将该地址拷贝到浏览器中即可访问该服务。

通过控制台操作

1. 登录容器服务管理控制台，单击目标集群右侧的控制台。
2. 在Kubernetes菜单下，单击左侧导航栏应用 > 部署，进入部署列表页面。
3. 选择目标集群和命名空间，单击右上角使用模板创建。

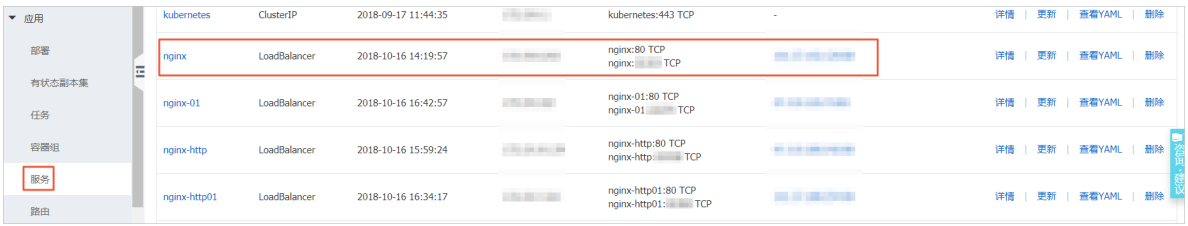


4. 示例模板选为自定义，将以下内容复制到模板中。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nginx
  name: ngnix
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

5. 单击创建。
6. 创建成功，单击**Kubernetes** 控制台前往控制台查看创建进度。

或单击左侧导航栏应用 > 服务，选择目标集群和命名空间，查看已部署的服务。



更多信息

阿里云负载均衡还支持丰富的配置参数，包含健康检查、收费类型、负载均衡类型等参数。详细信息参见负载均衡配置参数表。

注释

阿里云可以通过注释annotations的形式支持丰富的负载均衡功能。

使用已有的内网 SLB

需要指定两个annotation。注意修改成您自己的 Loadbalancer-id。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: "
intranet"
    service.beta.kubernetes.io/alicloud-loadbalancer-id: "your-
loadbalancer-id"
  labels:
    run: nginx
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  sessionAffinity: None
  type: LoadBalancer
```

创建HTTP类型的负载均衡

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-protocol-port: "
http:80"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

创建 HTTPS 类型的负载均衡

需要先在阿里云控制台上创建一个证书并记录 cert-id，然后使用如下 annotation 创建一个 HTTPS 类型的 SLB。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-cert-id: "your-
cert-id"
    service.beta.kubernetes.io/alicloud-loadbalancer-protocol-port: "
https:443"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  sessionAffinity: None
  type: LoadBalancer
```

限制负载均衡的带宽

只限制负载均衡实例下的总带宽，所有监听共享实例的总带宽，参见共享实例带宽。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-charge-type: "
paybybandwidth"
    service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth: "100"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

指定负载均衡规格

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-spec: "slb.s1.
small"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
```

```
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

使用已有的负载均衡

默认情况下，使用已有的负载均衡实例，不会覆盖监听，如要强制覆盖已有监听，请配置**service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners**为true。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-id: "your_loadb
alancer_id"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer: LoadBalancer
```

使用已有的负载均衡，并强制覆盖已有监听

强制覆盖已有监听，会删除已有负载均衡实例上的已有监听。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-id: "your_loadb
alancer_id"
    service.beta.kubernetes.io/alicloud-loadbalancer-force-override-
listeners: "true"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancere: LoadBalancer
```

使用指定**label**的**worker**节点作为后端服务器

多个label以逗号分隔。例如："k1:v1,k2:v2"

多个label之间是and的关系。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-backend-label: "
failure-domain.beta.kubernetes.io/zone:ap-southeast-5a"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

为TCP类型的负载均衡配置会话保持保持时间

该参数**service.beta.kubernetes.io/alicloud-loadbalancer-persistence-tim**仅对TCP协议的监听生效。

如果负载均衡实例配置了多个TCP协议的监听端口，则默认将该配置应用到所有TCP协议的监听端口。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-persistence-
timeout: "1800"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

为HTTP&HTTPS协议的负载均衡配置会话保持（insert cookie）

仅支持HTTP及HTTPS协议的负载均衡实例。

如果配置了多个HTTP或者HTTPS的监听端口，该会话保持默认应用到所有HTTP和HTTPS监听端口。

```
apiVersion: v1
kind: Service
metadata:
```

```
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session: "
on"
    service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session-
type: "insert"
    service.beta.kubernetes.io/alicloud-loadbalancer-cookie-timeout: "
1800"
    service.beta.kubernetes.io/alicloud-loadbalancer-protocol-port: "
http:80"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

为HTTP&HTTPS协议的负载均衡配置会话保持（server cookie）

仅支持HTTP及HTTPS协议的负载均衡实例。

如果配置了多个HTTP或者HTTPS的监听端口，该会话保持默认应用到所有HTTP和HTTPS监听端口。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session: "
on"
    service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session-
type: "server"
    service.beta.kubernetes.io/alicloud-loadbalancer-cooyour_cookie: "
your_cookie"
    service.beta.kubernetes.io/alicloud-loadbalancer-protocol-port: "
http:80"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

创建负载均衡时，指定主备可用区

某些region的负载均衡不支持主备可用区，如ap-southeast-5。

一旦创建，主备可用区不支持修改。

```
apiVersion: v1
```

```
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-master-zoneid: "
ap-southeast-5a"
    service.beta.kubernetes.io/alicloud-loadbalancer-slave-zoneid: "ap
-southeast-5a"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

使用**Pod**所在的节点作为后端服务器

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
spec:
  externalTrafficPolicy: Local
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

 说明：

注释的内容是区分大小写的。

注释	描述	默认值
service.beta.kubernetes.io/alicloud-loadbalancer-protocol-port	多个值之间由逗号分隔，比如：https:443,http:80	无
service.beta.kubernetes.io/alicloud-loadbalancer-address-type	取值可以是internet或者intranet	internet
service.beta.kubernetes.io/alicloud-loadbalancer-slb-network-type	负载均衡的网络类型，取值可以是classic或者vpc	classic

注释	描述	默认值
service.beta.kubernetes.io/alicloud-loadbalancer-charge-type	取值可以是paybytraffic或者paybybandwidth	paybytraffic
service.beta.kubernetes.io/alicloud-loadbalancer-id	负载均衡实例的 ID。通过**service.beta.kubernetes.io/alicloud-loadbalancer-id**指定您已有的SLB，已有监听会被覆盖， 删除 service 时该 SLB 不会被删除。	无
service.beta.kubernetes.io/alicloud-loadbalancer-backend-label	通过 label 指定 SLB 后端挂载哪些worker节点。	无
service.beta.kubernetes.io/alicloud-loadbalancer-spec	负载均衡实例的规格。可参考：*#unique_98*	无
service.beta.kubernetes.io/alicloud-loadbalancer-persistence-timeout	会话保持时间。 仅针对TCP协议的监听，取值：0-3600（秒） 默认情况下，取值为0，会话保持关闭。 可参考：*CreateLoadBalancerTCPListener*	0
service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session	是否开启会话保持。取值：on | off 说明： 仅对HTTP和HTTPS协议的监听生效。 可参考：*CreateLoadBalancerHTTPListener*和*CreateLoadBalancerHTTPSListene*	off
service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session-type	cookie的处理方式。取值： • insert：植入Cookie。 • server：重写Cookie。 说明：	无

注释	描述	默认值
	• 仅对HTTP和HTTPS协议的监听生效。 • 当**service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session**取值为on时，该参数必选。 可参考：*CreateLoadBalancerHTTPListener*和*CreateLoadBalancerHTTPSListene*	
service.beta.kubernetes.io/alicloud-loadbalancer-cookie-timeout	Cookie超时时间。取值：1-86400（秒） 说明：当**service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session**为on且**service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session-type**为insert时，该参数必选。 可参考：*CreateLoadBalancerHTTPListener*和*CreateLoadBalancerHTTPSListene*	无
service.beta.kubernetes.io/alicloud-loadbalancer-cookie	服务器上配置的Cookie。长度为1-200个字符，只能包含ASCII英文字母和数字字符，不能包含逗号、分号或空格，也不能以$开头。 说明：当**service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session**	无

注释	描述	默认值
	为on且**service.beta.kubernetes.io/alicloud-loadbalancer-sticky-session-type**为server时，该参数必选。 可参考：*CreateLoadBalancerHTTPListener*和*CreateLoadBalancerHTTPSListene*	
service.beta.kubernetes.io/alicloud-loadbalancer-master-zoneid	主后端服务器的可用区ID。	无
service.beta.kubernetes.io/alicloud-loadbalancer-slave-zoneid	备后端服务器的可用区ID。	无
externalTrafficPolicy	哪些节点可以作为后端服务器，取值： • `Cluster`：使用所有后端节点作为后端服务器。 • `Local`：使用Pod所在节点作为后端服务器。	`Cluster`
service.beta.kubernetes.io/alicloud-loadbalancer-force-override-listeners	绑定已有负载均衡时，是否强制覆盖该SLB的监听。	`false`：不覆盖
service.beta.kubernetes.io/alicloud-loadbalancer-region	负载均衡所在的地域	无
service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth	负载均衡的带宽	`50`
service.beta.kubernetes.io/alicloud-loadbalancer-cert-id	阿里云上的认证 ID。您需要先上传证书	无

注释	描述	默认值
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-flag	取值是on | off	默认为off。TCP 不需要改参数。因为 TCP 默认打开健康检查，用户不可设置。
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-type	健康检查类型，取值：tcp | http。 可参考：*CreateLoadBalancerTCPListener*	tcp
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-uri	用于健康检查的URI。 说明： 当健康检查类型为TCP模式时，无需配置该参数。 可参考：*CreateLoadBalancerTCPListener*	无
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-connect-port	健康检查使用的端口。取值： • -520：默认使用监听配置的后端端口。 • 1-65535：健康检查的后端服务器的端口。 可参考：*CreateLoadBalancerTCPListener*	无
service.beta.kubernetes.io/alicloud-loadbalancer-healthy-threshold	可参考：*CreateLoadBalancerTCPListener*	无
service.beta.kubernetes.io/alicloud-loadbalancer-unhealthy-threshold	健康检查连续成功多少次后，将后端服务器的健康检查状态由fail判定为success。取值： 2-10 可参考：*CreateLoadBalancerTCPListener*	无
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-interval	健康检查的时间间隔。 取值：1-50（秒） 可参考：*CreateLoadBalancerTCPListener*	无

注释	描述	默认值
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-connect-timeout	接收来自运行状况检查的响应需要等待的时间。如果后端ECS在指定的时间内没有正确响应，则判定为健康检查失败。 取值：1-300（秒） 说明： 如果**service.beta.kubernetes.io/alicloud-loadbalancer-health-check-connect-timeout**的值小于**service.beta.kubernetes.io/alicloud-loadbalancer-health-check-interval**的值，则**service.beta.kubernetes.io/alicloud-loadbalancer-health-check-connect-timeout**无效，超时时间为**service.beta.kubernetes.io/alicloud-loadbalancer-health-check-interval**的值。 可参考：*CreateLoadBalancerTCPListener*	无
service.beta.kubernetes.io/alicloud-loadbalancer-health-check-timeout	接收来自运行状况检查的响应需要等待的时间。如果后端ECS在指定的时间内没有正确响应，则判定为健康检查失败。 取值：1-300（秒） 说明： 如果 **service.beta.kubernetes.io/alicloud-loadbalanc**	无

注释	描述	默认值
	er-health-check-timeout的值小于**service.beta.kubernetes.io/alicloud-loadbalancer-health-check-interval**的值，则 **service.beta.kubernetes.io/alicloud-loadbalancer-health-check-timeout** 无效，超时时间为 **service.beta.kubernetes.io/alicloud-loadbalancer-health-check-interval**的值。 可参考：*CreateLoadBalancerHTTPListener*	

1.8.3 Ingress 支持

在 Kubernetes 集群中，Ingress是授权入站连接到达集群服务的规则集合，为您提供七层负载均衡能力。您可以给 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。

前置条件

为了测试复杂路由服务，本例中创建一个 nginx 的示例应用，您需要事先创建 nginx 的 deployment，然后创建多个 Service，用来观察路由的效果。实际测试请替换成自己的服务。

```
root@master # kubectl run nginx --image=registry.cn-hangzhou.aliyuncs.com/acs/netdia:latest

root@master # kubectl expose deploy nginx --name=http-svc --port=80 --target-port=80
root@master # kubectl expose deploy nginx --name=http-svc1 --port=80 --target-port=80
root@master # kubectl expose deploy nginx --name=http-svc2 --port=80 --target-port=80
```

```
root@master # kubectl expose deploy nginx --name=http-svc3 --port=80
 --target-port=80
```

简单的路由服务

通过以下命令创建一个简单的 Ingress，所有对 */svc* 路径的访问都会被路由到名为 http-svc 的服务。`nginx.ingress.kubernetes.io/rewrite-target: /`会将*/svc*路径重定向到后端服务能够识别的/路径上面。

```
root@master # cat <<EOF | kubectl create -f -
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /svc
        backend:
          serviceName: http-svc
          servicePort: 80
EOF
root@master # kubectl get ing
NAME            HOSTS         ADDRESS          PORTS     AGE
simple          *             101.37.192.211   80        11s
```

现在访问 `http://101.37.192.211/svc` 即可访问到 Nginx 服务。

基于域名的简单扇出路由

如果您有多个域名对外提供不同的服务，您可以生成如下的配置达到一个简单的基于域名的扇出效果。

```
root@master # cat <<EOF | kubectl create -f -
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: http-svc1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: http-svc2
          servicePort: 80
  - host: foo.example.com
    http:
```

```
      paths:
      - path: /film
        backend:
          serviceName: http-svc3
          servicePort: 80
EOF
root@master # kubectl get ing
NAME            HOSTS         ADDRESS          PORTS     AGE
simple-fanout   *             101.37.192.211   80        11s
```

这时您可以通过 `http://foo.bar.com/foo` 访问到 `http-svc1` 服务；通过 `http://foo.bar.com/bar` 访问到 `http-svc2` 服务；通过 `http://foo.example.com/film` 访问到 `http-svc3` 服务。

说明：

- 如果是生产环境，您需要将您的这个域名指向上面返回的 ADDRESS `101.37.192.211`。
- 如果是测试环境测试，您可以修改 `hosts` 文件添加一条域名映射规则。

```
101.37.192.211 foo.bar.com
101.37.192.211 foo.example.com
```

简单路由默认域名

如果您没有域名地址也没有关系，容器服务为 Ingress 服务绑定了一个默认域名，您可以通过这个域名来访问服务。域名的格式如下：`*.[cluster-id].[region-id].alicontainer.com`。您可以直接在控制台集群基本信息页获取到该地址。

您可以通过下面的配置借助该默认域名暴露两个服务。

```
root@master # cat <<EOF | kubectl create -f -
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: shared-dns
spec:
  rules:
  - host: foo.[cluster-id].[region-id].alicontainer.com  ##替换为您集群默认的服务访问域名
    http:
      paths:
      - path: /
        backend:
          serviceName: http-svc1
          servicePort: 80
  - host: bar.[cluster-id].[region-id].alicontainer.com  ##替换为您集群默认的服务访问域名
    http:
      paths:
      - path: /
        backend:
          serviceName: http-svc2
```

```
          servicePort: 80
EOF
root@master # kubectl get ing
NAME             HOSTS          ADDRESS           PORTS     AGE
shared-dns   foo.[cluster-id].[region-id].alicontainer.com,bar.[
cluster-id].[region-id].alicontainer.com             47.95.160.171
80        40m
```

这时您可以通过 `http://foo.[cluster-id].[region-id].alicontainer.com/` 访问到 `http-svc1` 服务；通过 `http://bar.[cluster-id].[region-id].alicontainer.com` 访问到 `http-svc2` 服务。

配置安全的路由服务

支持多证书管理，为您的服务提供安全防护。

1. 准备您的服务证书。

   如果没有证书，可以通过下面的方法生成测试证书。

   > **说明：**
   > 域名与您的 Ingress 配置要一致。

   ```
   root@master # openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
   keyout tls.key -out tls.crt -subj "/CN=foo.bar.com/O=foo.bar.com"
   ```

   上面命令会生成一个证书文件 *tls.crt*、一个私钥文件*tls.key*。

   然后用该证书和私钥创建一个名为*foo.bar* 的 Kubernetes Secret。创建 Ingress 时需要引用这个 Secret。

   ```
   root@master # kubectl create secret tls foo.bar --key tls.key --cert
    tls.crt
   ```

2. 创建一个安全的 Ingress 服务。

   ```
   root@master # cat <<EOF | kubectl create -f -
   apiVersion: extensions/v1beta1
   kind: Ingress
   metadata:
     name: tls-fanout
   spec:
     tls:
     - hosts:
       - foo.bar.com
       secretName: foo.bar
     rules:
     - host: foo.bar.com
       http:
         paths:
         - path: /foo
           backend:
   ```

```
          serviceName: http-svc1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: http-svc2
          servicePort: 80
EOF
root@master # kubectl get ing
NAME            HOSTS         ADDRESS          PORTS     AGE
tls-fanout      *             101.37.192.211   80        11s
```

3. 按照 基于域名的简单扇出路由 中的注意事项，配置 `hosts` 文件或者设置域名来访问该 tls 服务。

   您可以通过 `http://foo.bar.com/foo` 访问到 `http-svc1` 服务；通过 `http://foo.bar.com/bar` 访问到 `http-svc2` 服务。

   您也可以通过 HTTP 的方式访问该 HTTPS 的服务。Ingress 默认对配置了 HTTPS 的 HTTP 访问重定向到 HTTPS 上面。所以访问 `http://foo.bar.com/foo` 会被自动重定向到 `https://foo.bar.com/foo`。

通过 Kubernetes Dashboard 部署 Ingress

1. 将下面的 yml code 保存到 `nginx-ingress.yml` 文件中。

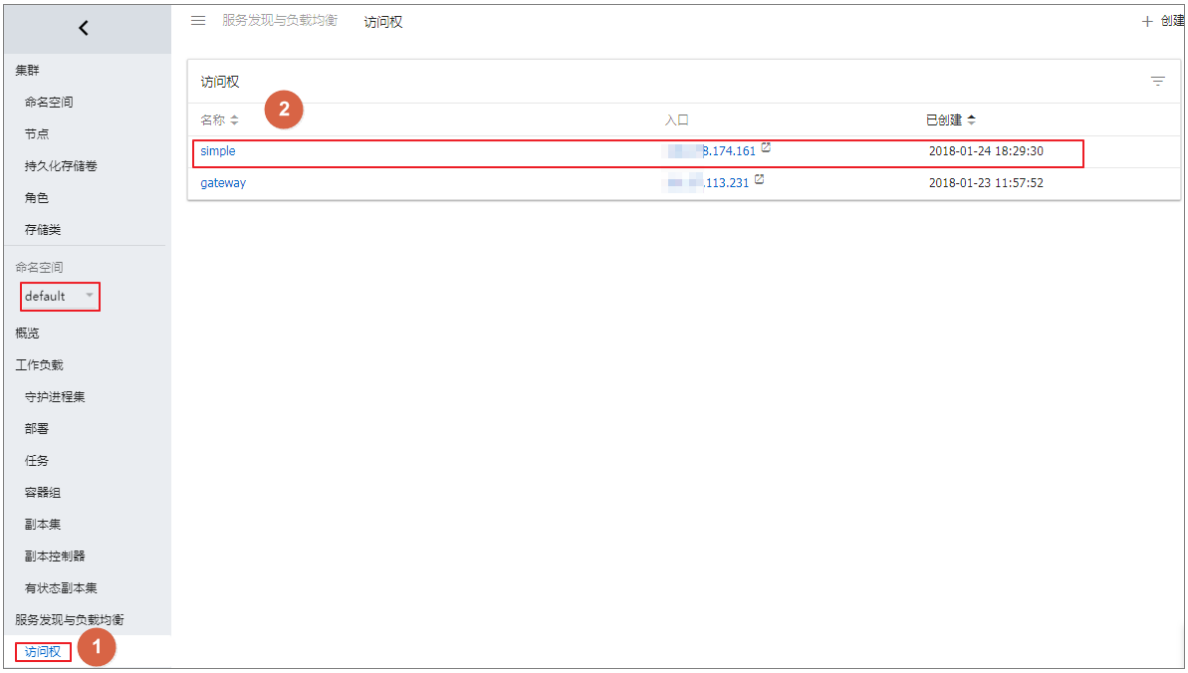
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple
spec:
  rules:
  - http:
      paths:
      - path: /svc
        backend:
          serviceName: http-svc
          servicePort: 80
```

2. 登录容器服务管理控制台，在 Kubernetes 菜单下，在集群列表页面中，单击目标集群右侧的控制台，进入 Kubernetes Dashboard 页面。
3. 单击创建，开始创建应用。

4. 单击使用文件创建。选择刚才保存的 `nginx-ingress.yml` 文件。
5. 单击上传。

   这样就创建了一个 Ingress 的七层代理路由到 `http-svc` 服务上。
6. 在 Kubernetes Dashboard 上定位到 default 命名空间，选择访问权。

   可以看到您刚刚创建的 Ingress 资源及其访问地址 `http://118.178.174.161/svc`。



7. 打开浏览器输入该地址即可访问前面创建的 `http-svc` 服务。

1.8.4 Ingress 监控配置

您可以通过开启 Ingress 默认 VTS 模块来查看 Ingress 监控数据。

通过命令行操作

1. 修改 Ingress ConfigMap 配置，增加配置项 `enable-vts-status: "true"`。

```
root@master # kubectl edit configmap nginx-configuration -n kube-
system
configmap "nginx-configuration" edited
```

修改后 Ingress ConfigMap 内容如下。

```
apiVersion: v1
data:
  enable-vts-status: "true" # 开启VTS模块
  proxy-body-size: 20m
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"proxy-body-size":"20m"},"kind":"
ConfigMap","metadata":{"annotations":{},"labels":{"app":"ingress-
nginx"},"name":"nginx-configuration","namespace":"kube-system"}}
  creationTimestamp: 2018-03-20T07:10:18Z
  labels:
    app: ingress-nginx
  name: nginx-configuration
  namespace: kube-system
  selfLink: /api/v1/namespaces/kube-system/configmaps/nginx-
configuration
```

2. 验证 Ingress Nginx 正常开启 VTS 模块。

```
root@master # kubectl get pods --selector=app=ingress-nginx -n kube-
system
NAME                                        READY     STATUS
RESTARTS   AGE
nginx-ingress-controller-79877595c8-78gq8   1/1       Running   0
       1h
root@master # kubectl exec -it nginx-ingress-controller-79877595c8-
78gq8 -n kube-system -- cat /etc/nginx/nginx.conf | grep vhost_traf
fic_status_display
         vhost_traffic_status_display;
         vhost_traffic_status_display_format html;
```

3. 本地访问 Ingress Nginx 监控控制台。

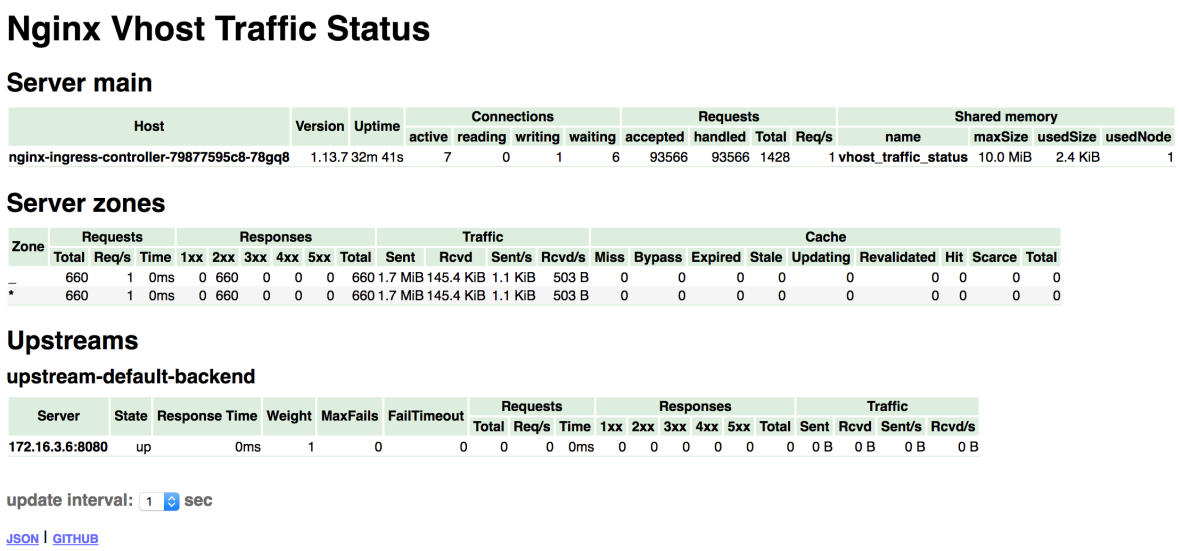
说明：

鉴于安全考虑，默认 VTS Port 并不对外开放，这里通过 port-forward 方式来访问。

```
root@master # kubectl port-forward nginx-ingress-controller-
79877595c8-78gq8 -n kube-system 18080
Forwarding from 127.0.0.1:18080 -> 18080
```

```
Handling connection for 18080
```

4. 通过 `http://localhost:18080/nginx_status` 来访问 VTS 监控控制台。



Nginx Vhost Traffic Status

Server main

Host	Version	Uptime	Connections				Requests				Shared memory			
			active	reading	writing	waiting	accepted	handled	Total	Req/s	name	maxSize	usedSize	usedNode
nginx-ingress-controller-79877595c8-78gq8	1.13.7	32m 41s	7	0	1	6	93566	93566	1428	1	vhost_traffic_status	10.0 MiB	2.4 KiB	1

Server zones

Zone	Requests			Responses					Traffic				Cache							
	Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s	Miss	Bypass	Expired	Stale	Updating	Revalidated	Hit	Scarce	Total
_	660	1	0ms	0	660	0	0	0	660	1.7 MiB	145.4 KiB	1.1 KiB	503 B	0	0	0	0	0	0	0	0	0
*	660	1	0ms	0	660	0	0	0	660	1.7 MiB	145.4 KiB	1.1 KiB	503 B	0	0	0	0	0	0	0	0	0

Upstreams

upstream-default-backend

Server	State	Response Time	Weight	MaxFails	FailTimeout	Requests			Responses						Traffic			
						Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s
172.16.3.6:8080	up	0ms	1	0	0	0	0	0ms	0	0	0	0	0	0	0 B	0 B	0 B	0 B

update interval: 1 sec

JSON | GITHUB

通过 **Kubernetes Dashboard** 操作

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择所需的集群并单击集群右侧的控制台，进入 Kubernetes Dashboard 页面。
3. 编辑 kube-system 命名空间下的配置字典 nginx-configuration，增加配置项 `enable-vts-status: "true"`。

   保存后 Ingress ConfigMap 内容如下。

```
{
  "kind": "ConfigMap",
  "apiVersion": "v1",
  "metadata": {
    "name": "nginx-configuration",
    "namespace": "kube-system",
    "selfLink": "/api/v1/namespaces/kube-system/configmaps/nginx-
configuration",
    "creationTimestamp": "2018-03-20T07:10:18Z",
    "labels": {
      "app": "ingress-nginx"
    },
    "annotations": {
      "kubectl.kubernetes.io/last-applied-configuration": "{\"
apiVersion\":\"v1\",\"data\":{\"proxy-body-size\":\"20m\"},\"kind\":
\"ConfigMap\",\"metadata\":{\"annotations\":{},\"labels\":{\"app\":
\"ingress-nginx\"},\"name\":\"nginx-configuration\",\"namespace\":\"
kube-system\"}}\n"
    }
  },
  "data": {
    "proxy-body-size": "20m",
```

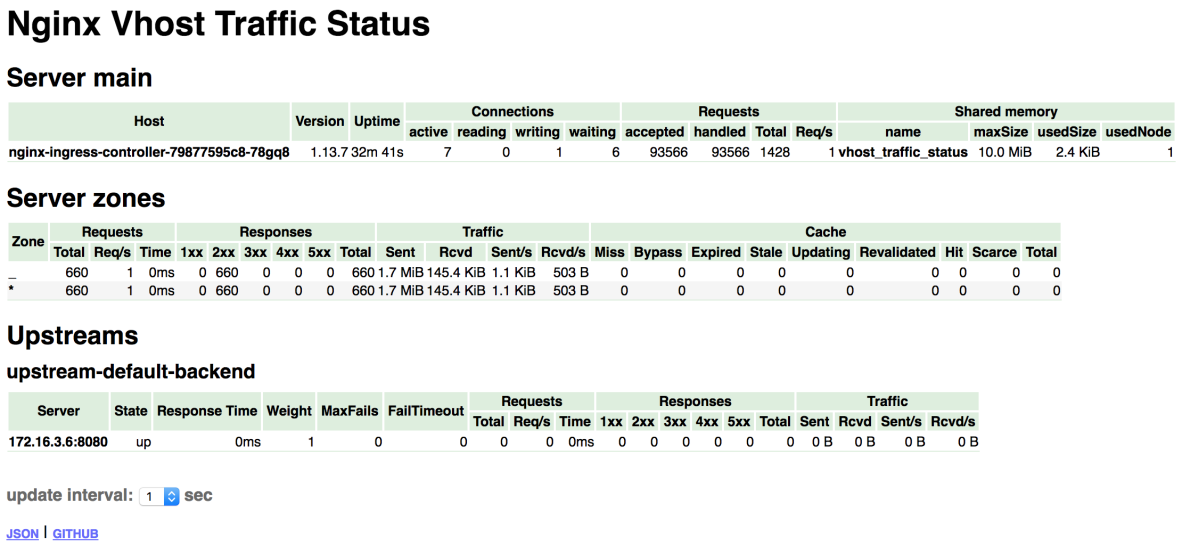
```
    "enable-vts-status": "true"
  }
}
```

4. 本地访问 Ingress Nginx 监控控制台。

> 说明：
> 鉴于安全考虑，默认 VTS Port 并不对外开放，这里通过 port-forward 方式来访问。

```
root@master # kubectl port-forward nginx-ingress-controller-
79877595c8-78gq8 -n kube-system 18080
Forwarding from 127.0.0.1:18080 -> 18080
Handling connection for 18080
```

5. 通过 `http://localhost:18080/nginx_status` 来访问 VTS 监控控制台。



Nginx Vhost Traffic Status

Server main

Host	Version	Uptime	Connections				Requests				Shared memory			
			active	reading	writing	waiting	accepted	handled	Total	Req/s	name	maxSize	usedSize	usedNode
nginx-ingress-controller-79877595c8-78gq8	1.13.7	32m 41s	7	0	1	6	93566	93566	1428	1	vhost_traffic_status	10.0 MiB	2.4 KiB	1

Server zones

Zone	Requests			Responses					Traffic					Cache							
	Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s	Miss	Bypass	Expired	Stale	Updating	Revalidated	Hit	Scarce	Total
_	660	1	0ms	0	660	0	0	0	660	1.7 MiB	145.4 KiB	1.1 KiB	503 B	0	0	0	0	0	0	0	0	0
*	660	1	0ms	0	660	0	0	0	660	1.7 MiB	145.4 KiB	1.1 KiB	503 B	0	0	0	0	0	0	0	0	0

Upstreams

upstream-default-backend

Server	State	Response Time	Weight	MaxFails	FailTimeout	Requests			Responses						Traffic			
						Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s
172.16.3.6:8080	up	0ms	1	0	0	0	0	0ms	0	0	0	0	0	0	0 B	0 B	0 B	0 B

update interval: 1 sec

JSON | GITHUB

1.8.5 路由配置说明

阿里云容器服务提供高可靠的 ingress controller 组件，集成了阿里云 SLB 服务，为您的 Kubernetes 集群提供灵活可靠的路由服务（Ingress）。

下面是一个 Ingress 编排示例。通过 Web 界面进行配置时，您需要对注释的参数进行配置，部分配置需要创建依赖项，具体请参见通过 *Web* 界面创建路由。 也可以参考*Ingress* 支持和 *Kubernetes Ingress*。此外，Ingress也支持configmap的配置方式，请参见*https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/*。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/service-match: 'new-nginx: header("foo
", /^bar$/)'            #灰度发布规则，本例为Header请求头
```

```
    nginx.ingress.kubernetes.io/service-weight: 'new-nginx: 50,old-
nginx: 50'                #流量权重注解
  creationTimestamp: null
  generation: 1
  name: nginx-ingress
  selfLink: /apis/extensions/v1beta1/namespaces/default/ingresses/
nginx-ingress
spec:
  rules:                                                          ##路由
规则
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: new-nginx
          servicePort: 80
        path: /
      - backend:
          serviceName: old-nginx
          servicePort: 80
        path: /
tls:                                                            ## 开启
TLS，配置安全路由
  - hosts:
    - *.xxxxxx.cn-hangzhou.alicontainer.com
    - foo.bar.com
    secretName: nginx-ingress-secret                             ##使用的
secret 名称
status:
  loadBalancer: {}
```

注解

您可以指定 ingress 的 annotation ，指定使用的 ingress controller，以及路由的规则，如路由权重规则、灰度发布规则和重写规则等。Ingress的注解请参见*https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/*。

例如，一个典型的重定向注解：`nginx.ingress.kubernetes.io/rewrite-target: /` 会将*/path*路径重定向到后端服务能够识别的*/* 路径上面。

规则

规则指的是授权入站连接到达集群服务的路由规则，通常指 http/https 规则，包括域名（虚拟主机名称）、URL 访问路径、服务及端口等。

每条规则需要配置以下信息：

- host 配置项：比如阿里云 Kubernetes 集群服务测试域名；或虚拟主机名称，如 `foo.bar.com` 。

- URL 路径：指定访问的 URL 路径，即 path。每个 path 都关联一个 backend（服务），在阿里云 SLB 将流量转发到 backend 之前，所有的入站请求都要先匹配 host 和 path。
- backend 配置：即服务配置，是一个 `service:port`和流量权重的组合。Ingress 的流量根据设置的权重被转发到它所匹配的 backend。
  - 服务名称：Ingress转发的backend服务名称。
  - 服务端口：服务暴露的端口。
  - 服务权重：一个服务组中各服务的权重比例。

> 说明：
>
> 1. 服务权重采用相对值计算方式。例如两个服务权重都设置为50，则两个服务的权重比例都是50%。
> 2. 一个服务组（同一个ingress yaml中具有相同Host和Path的服务）中未明确设置权重的服务默认权重值为100。

灰度发布

容器服务支持多种流量切分方式，适用于灰度发布以及AB测试场景。

> 说明：
> 目前阿里云容器服务K8S Ingress Controller需要**`0.12.0-5`**及其以上版本才支持流量切分特性。

1. 基于Request Header的流量切分
2. 基于Cookie的流量切分
3. 基于Query Param的流量切分

设置灰度规则后，请求头中满足灰度发布匹配规则的请求才能被路由到设置的服务中。如果该服务设置了100%以下的权重比例，满足灰度规则的请求会继续依据权重比例路由到服务组下的各个服务。

TLS

您可以通过指定包含 TLS 私钥和证书的 secret 来加密 Ingress，实现安全的路由访问。TLS secret 中必须包含名为 tls.crt 和 tls.key 的证书和私钥。更多 TLS 的原理，请参见 *TLS*；关于如何生成 secret，请参见配置安全的路由服务。

标签

您可为 ingress 添加标签，标示该 Ingress 的特点。

1.8.6 通过 Web 界面创建路由

阿里云容器服务 Web 界面集成了路由（Ingress）服务，您可通过 Web 界面快速创建路由服务，构建灵活可靠的流量接入层。

前提条件

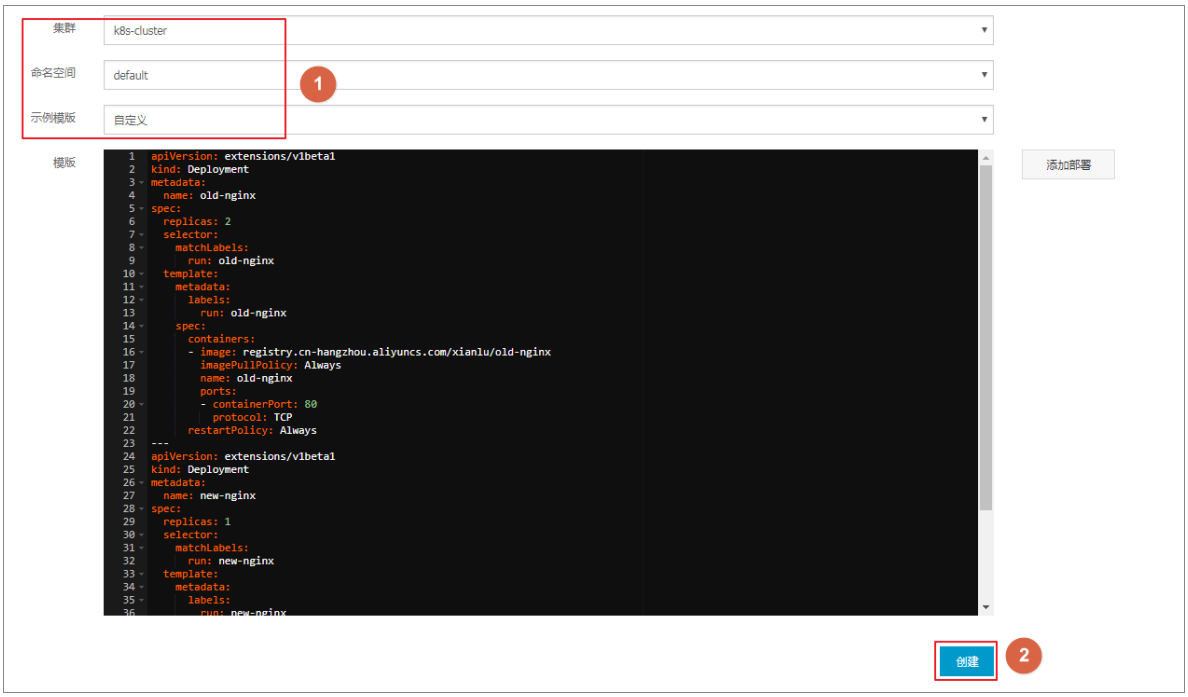
- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群，并且集群中 Ingress controller 正常运行。
- SSH 登录到 Master 节点，参见*SSH*访问*Kubernetes*集群。

步骤1 创建 deployment 和服务

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角使用模板创建。

4. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   本例中，示例中创建3个nginx应用，一个代表旧的应用old-nginx，一个代表新的应用 new-nginx，此外创建一个domain-nginx应用，用于测试集群访问域名。



old-nginx的编排模板如下所示：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: old-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
        imagePullPolicy: Always
        name: old-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffinity: None
```

```
  type: NodePort
```

new-nginx的编排模板如下所示：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: new-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: new-nginx
  template:
    metadata:
      labels:
        run: new-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
        imagePullPolicy: Always
        name: new-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

domain-nginx应用的编排模板如下所示：

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: domain-nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

```
      containers:
      - name: nginx
        image: nginx:1.7.9 # replace it with your exactly <
image_name:tags>
        ports:
        - containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: domain-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: NodePort
```

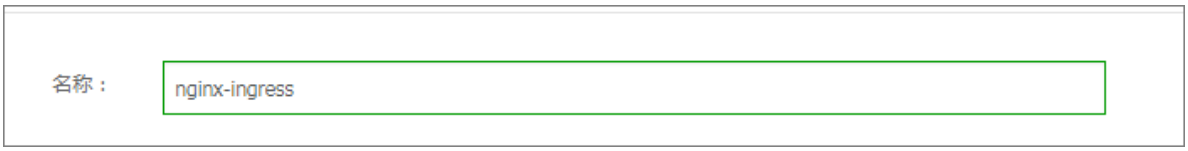
5. 单击左侧导航栏中的应用 > 服务 ，进入服务列表页面。

   等待服务创建完成后，在服务列表，您可看到本示例创建的服务。

步骤2 创建路由

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 路由，进入路由页面。
3. 选择所需的集群和命名空间，单击页面右上角的创建。

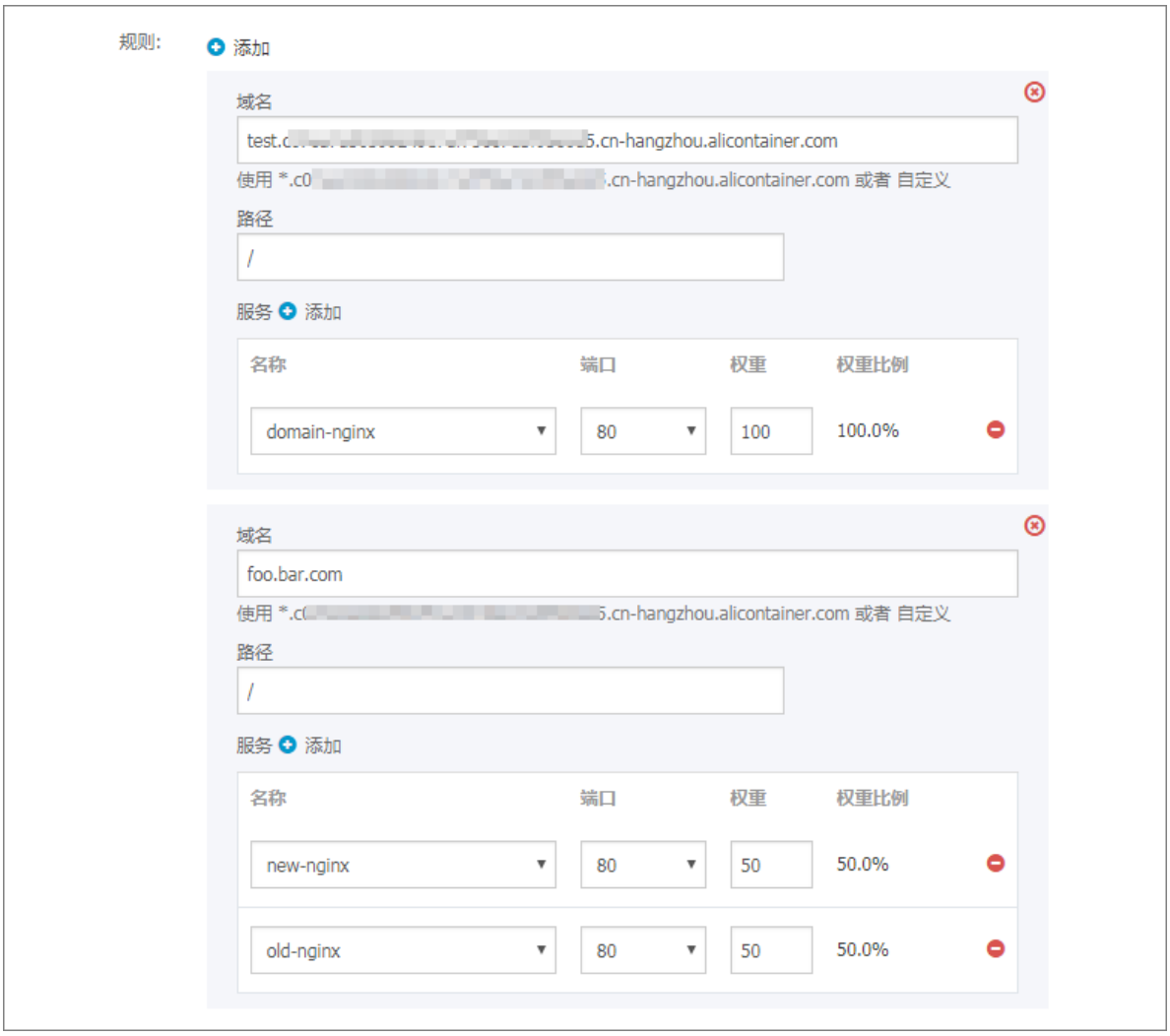
4. 在弹出的路由创建对话框中，首先配置路由名称，本例为 nginx-ingress。



5. 对路由规则进行配置。

   路由规则是指授权入站到达集群服务的规则，支持 http/https 规则，配置项包括域名(虚拟主机名称)、URL 路径、服务名称、端口配置和路由权重等。详细的信息请参见路由配置说明。

   本例中配置添加一条复杂的路由规则，配置集群默认的测试域名和虚拟主机名称，展示基于域名的路由服务。



- 基于默认域名的简单路由，即使用集群的默认域名对外提供访问服务。
  - **域名配置**：使用集群的默认域名，本例中是 `test.[cluster-id].[region-id].alicontainer.com`。

    在创建路由对话框中，会显示该集群的默认域名，域名格式是 `*.[cluster-id].[region-id].alicontainer.com`；您也可在集群的基本信息页面中获取。
  - **服务配置**：配置服务的访问路径、名称以及端口。
    - 访问路径配置：您可指定服务访问的 URL 路径，默认为根路径/，本例中不做配置。每个路径（path）都关联一个 backend（服务），在阿里云 SLB 将流量转发到 backend 之前，所有的入站请求都要先匹配域名和路径。
    - 服务配置：支持服务名称、端口、服务权重等配置，即 backend 配置。同一个访问路径下，支持多个服务的配置，Ingress的流量会被切分，并被转发到它所匹配的 backend。

- 基于域名的简单扇出路由。本例中使用一个虚拟的主机名称作为测试域名对外提供访问服务，为两个服务配置路由权重，并为其中一个服务设置灰度发布规则。若您在生产环境中，可使用成功备案的域名提供访问服务。
  - **域名配置**：本例中使用测试域名`foo.bar.com`。

    您需要修改 hosts 文件添加一条域名映射规则。

    ```
    118.178.108.143 foo.bar.com       #IP即是Ingress的Address
    ```

  - **服务配置**：配置服务的访问路径、服务名称、服务端口和服务权重。
    - 访问路径配置：指定服务访问的 URL 路径。本例中不做配置，保留根路径/。
    - 服务名称：本例中设置新旧两个服务 nginx-new 和 nginx-old 。
    - 服务端口：暴露80端口。
    - 权重设置：设置该路径下多个服务的权重。服务权重采用相对值计算方式，默认值为100，如本例中所示，新旧两个版本的服务权重值都是50，则表示两个服务的权重比例都是50%。

6. 配置灰度发布。

   说明：

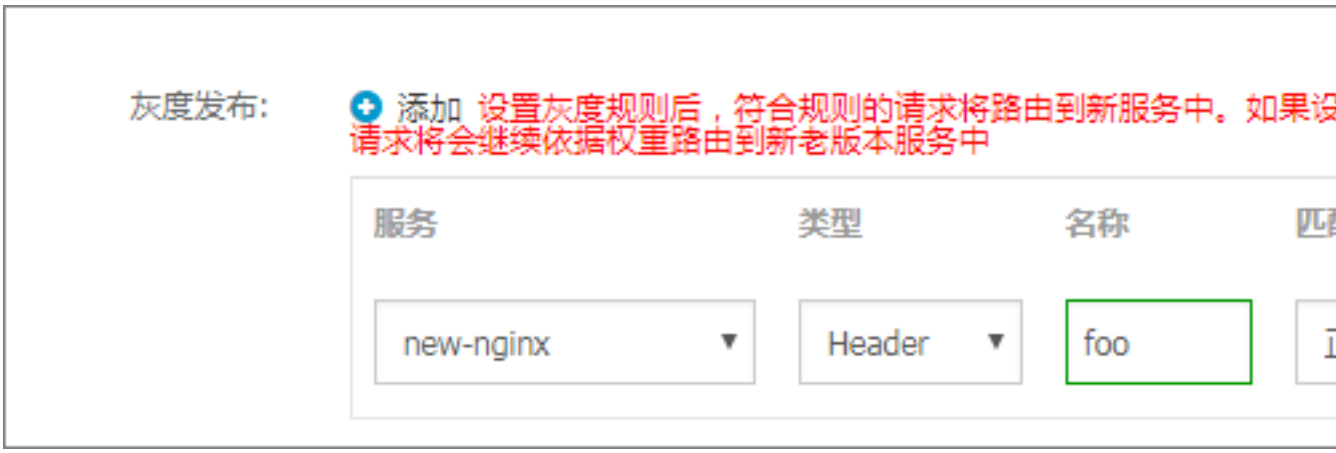
   目前阿里云容器服务Kubernetes Ingress Controller需要**0.12.0-5**及其以上版本才支持流量切分特性。

   容器服务支持多种流量切分方式，适用于灰度发布以及AB测试场景。

   1. 基于Request Header的流量切分
   2. 基于Cookie的流量切分
   3. 基于Query Param的流量切分

   设置灰度规则后，请求头中满足灰度发布匹配规则的请求才能被路由到新版本服务new-nginx中。如果该服务设置了100%以下的权重比例，满足灰度规则的请求会继续依据权重比例路由到对应服务。

   在本例中，设置Header请求头带有`foo=^bar$`的灰度发布规则，仅带有该请求头的客户端请求才能访问到new-nginx 服务。



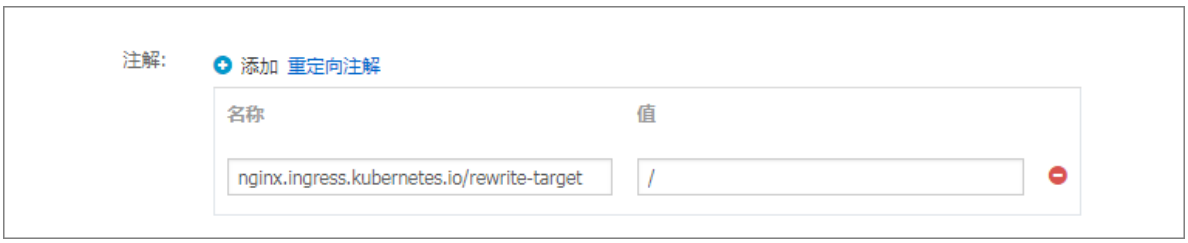
- **服务**：路由规则配置的服务。
- **类型**：支持Header（请求头）、Cookie和Query（请求参数）的匹配规则。
- **名称和匹配值**：用户自定义的请求字段，名称和匹配值为键值对。
- **匹配规则**：支持正则匹配和完全匹配。

7. 配置注解。

   单击**重定向注解**，可为路由添加一条典型的重定向注解。即`nginx.ingress.kubernetes.io/rewrite-target`：`/`，表示将*/path* 路径重定向到后端服务能够识别的根路径/上面。

   > **说明**：
   > 本例中未对服务配置访问路径，因此不需要配置重定向注解。重定向注解的作用是使Ingress以根路径转发到后端，避免访问路径错误配置而导致的404错误。

   您也可单击**添加**按钮，输入注解名称和值，即Ingress的annotation键值对，Ingress的注解参见*https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/*。



8. 配置 TLS。勾选**开启 TLS**，配置安全的路由服务。具体可参见配置安全的路由服务。

   - 您可选择使用已有密钥。

1. 登录 master 节点，创建 *tls.key* 和 *tls.crt*。

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls
.key -out tls.crt -subj "/CN=foo.bar.com/O=foo.bar.com"
```

2. 创建一个 secret。

```
kubectl create secret tls foo.bar --key tls.key --cert tls.crt
```

3. 执行命令 `kubectl get secret`，您可看到该 secret 已经成功创建。在 Web 界面可选择创建的*foo.bar*这个 secret。

- 您可选择在 TLS 界面上利用已创建的 TLS 私钥和证书，一键创建 secret。

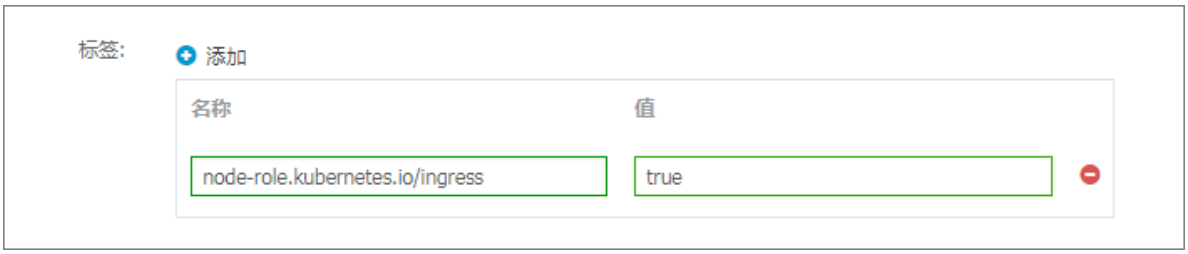
1. 登录 master 节点，创建 *tls.key* 和 *tls.crt*。

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls
.key -out tls.crt -subj "/CN=foo.bar.com/O=foo.bar.com"
```

2. 执行 `vim tls.key` 和 `vim tls.crt` 获取生成的私钥和证书。
3. 将证书和私钥的内容复制到 TLS 新建密钥的面板。

9. 添加标签。

   标签的作用是为Ingress添加对应的标签，标示该Ingress的特点。



10. 最后单击创建，返回路由列表。

    等待一段时间，可以看到一条路由。

11. 单击路由中的访问域名 `test.[cluster-id].[region-id].alicontainer.com`，以及 `foo.bar.com` ，可访问 nginx 的欢迎页面。

单击指向new-nginx服务的路由地址，发现指向了old-nginx应用的页面。

> **说明：**
> 在浏览器中访问路由地址，默认情况下，请求头（Header）中没有前面步骤中定义的`foo=^bar$`，因此流量会导向old-nginx应用。

12. SSH登录到Master节点，执行以下命令，模拟带有特定请求头的访问结果。

```
curl -H "Host: foo.bar.com" http://47.107.20.35
old
 curl -H "Host: foo.bar.com" http://47.107.20.35
old
 curl -H "Host: foo.bar.com" http://47.107.20.35
 #类似于浏览器的访问请求
old
 curl -H "Host: foo.bar.com" -H "foo: bar" http://47.107.20.35
     #模拟带有特有header的访问请求，会根据路由权重返回结果
new
 curl -H "Host: foo.bar.com" -H "foo: bar" http://47.107.20.35
old
 curl -H "Host: foo.bar.com" -H "foo: bar" http://47.107.20.35
old
 curl -H "Host: foo.bar.com" -H "foo: bar" http://47.107.20.35
```

```
new
```

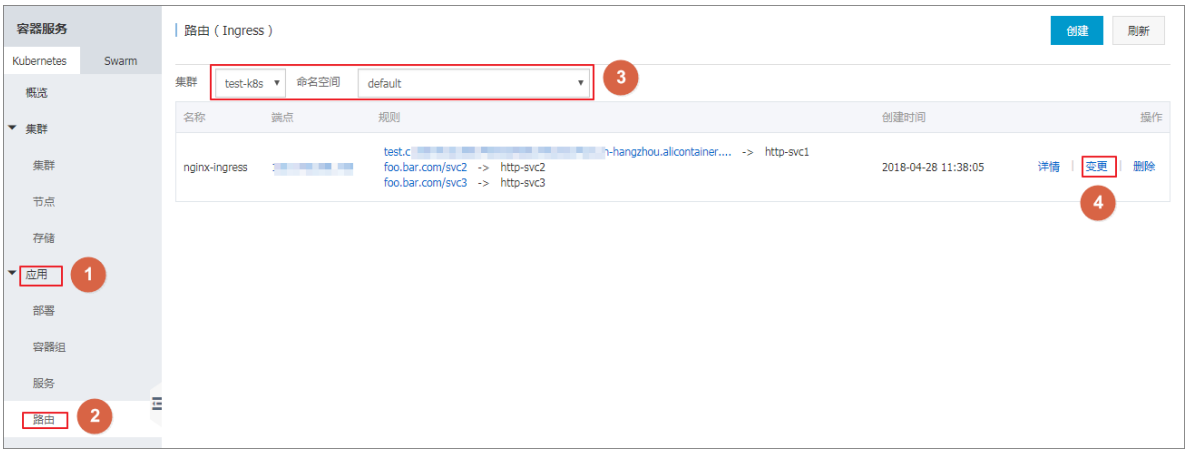
1.8.7 变更路由

前提条件

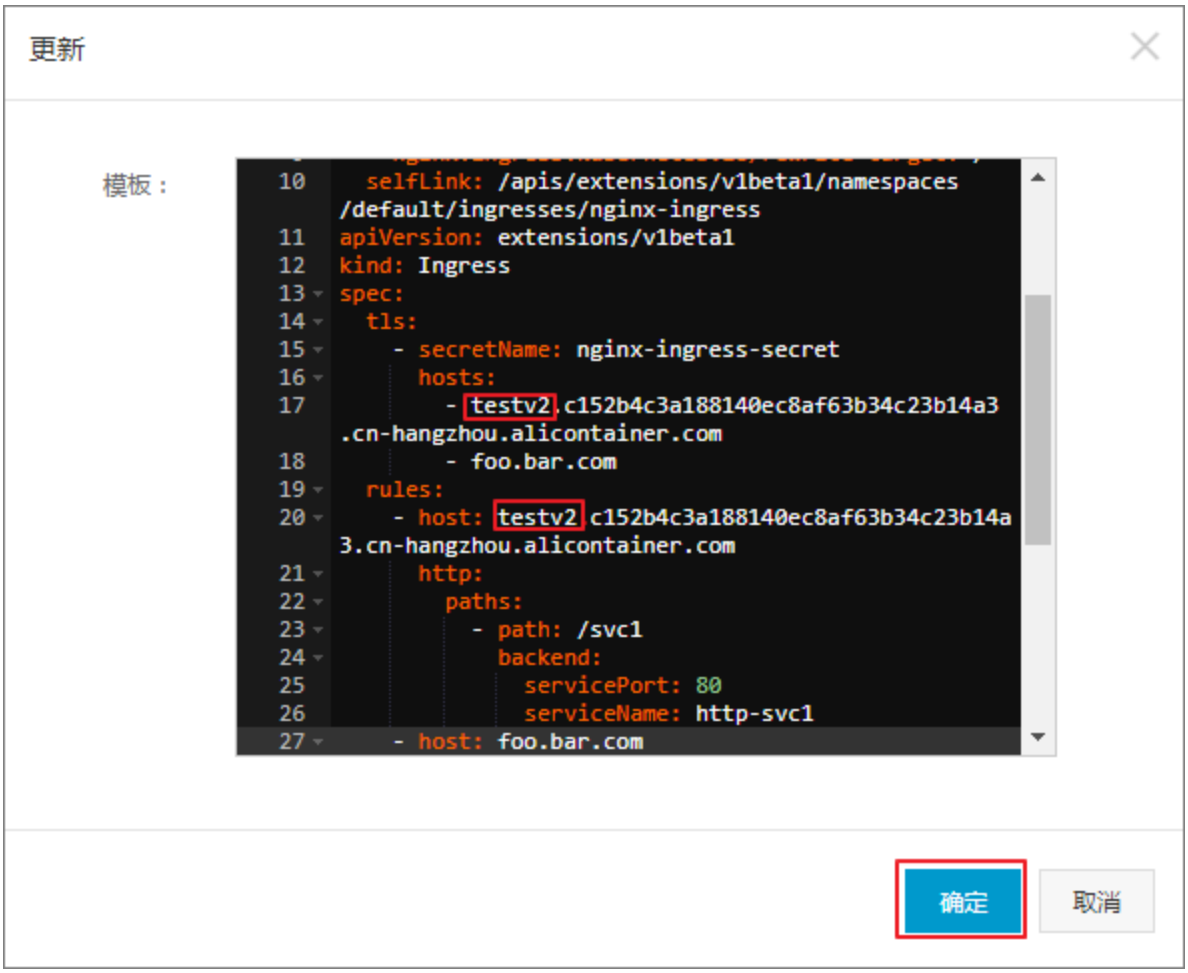
- 您已经成功创建一个 Kubernetes 集群，参见创建 Kubernetes 集群，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见通过 Web 界面创建路由。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 路由，进入路由页面。
3. 选择所需的集群和命名空间，选择所需的路由，然后单击路由右侧的变更。

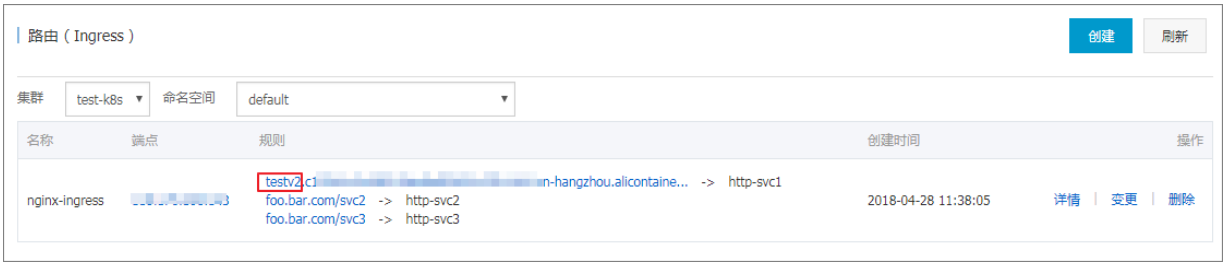


4. 在弹出的对话框中，对路由的相关参数进行变更，然后单击确定 。本例中将 `test.[cluster-id].[region-id].alicontainer.com` 修改为 `testv2.[cluster-id].[region-id].alicontainer.com`。



后续操作

返回路由列表，您可看到该路由的其中一条路由规则发生变化。



1.8.8 查看路由

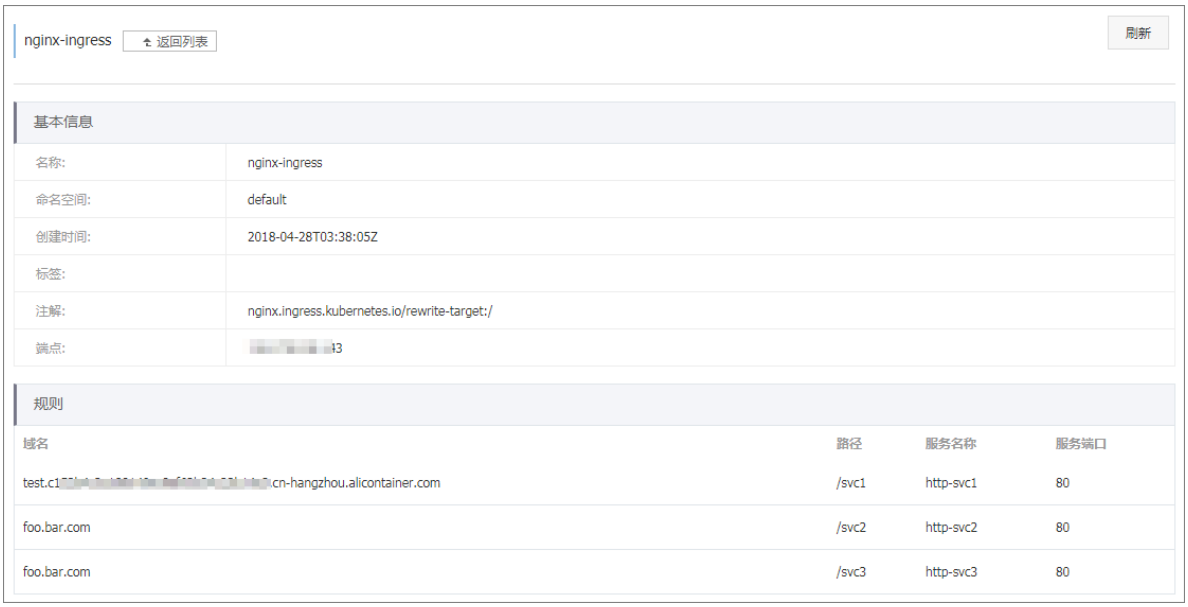
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建 Kubernetes 集群，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见通过 Web 界面创建路由。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的 应用 > 路由，进入路由页面。
3. 选择所需的集群和命名空间，选择所需的路由，然后单击路由右侧的详情。

进入路由详情页面，您可查看该路由的基本信息以及路由规则。



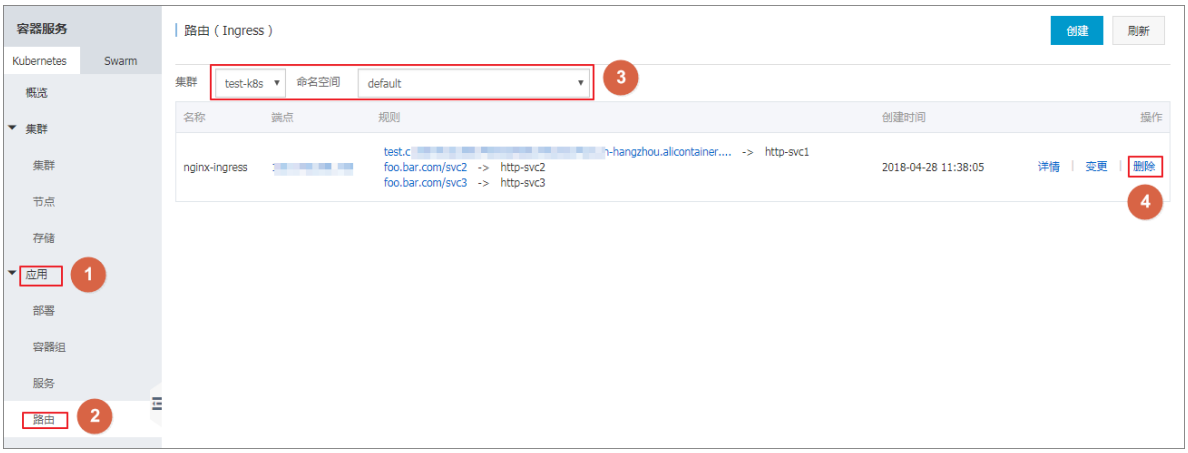
1.8.9 删除路由

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建 *Kubernetes* 集群，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见通过 *Web* 界面创建路由。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 路由，进入路由页面。
3. 选择所需的集群和命名空间，选择所需的路由，然后单击路由右侧的删除。



4. 在弹出的对话框中，单击确定，完成删除。

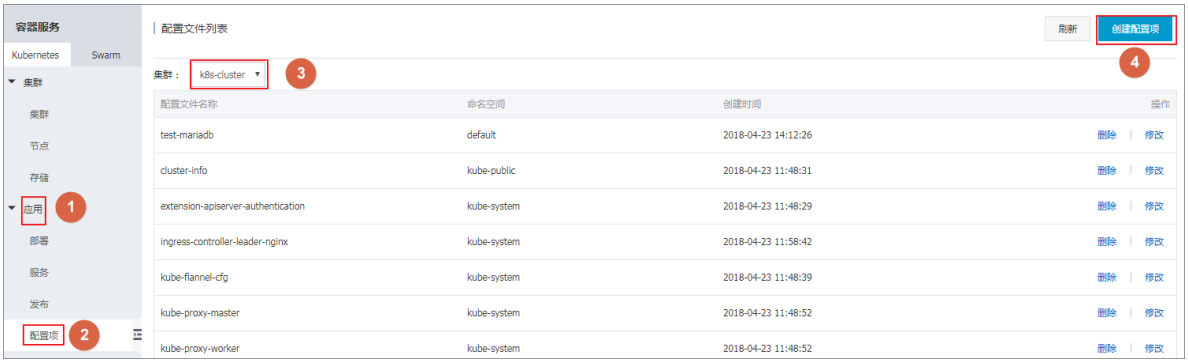
1.9 配置项及密钥管理

1.9.1 创建配置项

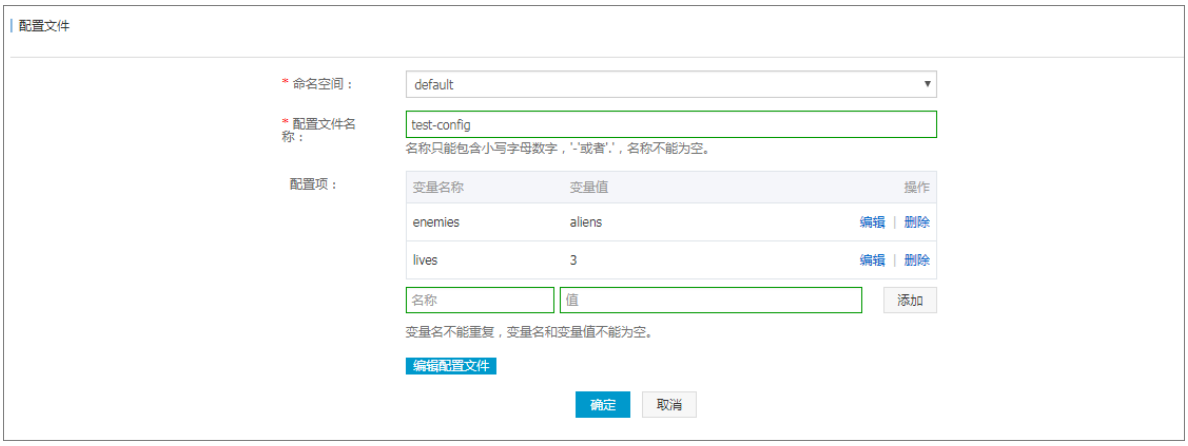
在容器服务管理控制台上，您可以通过配置项菜单或使用模板来创建配置项。

通过配置项创建

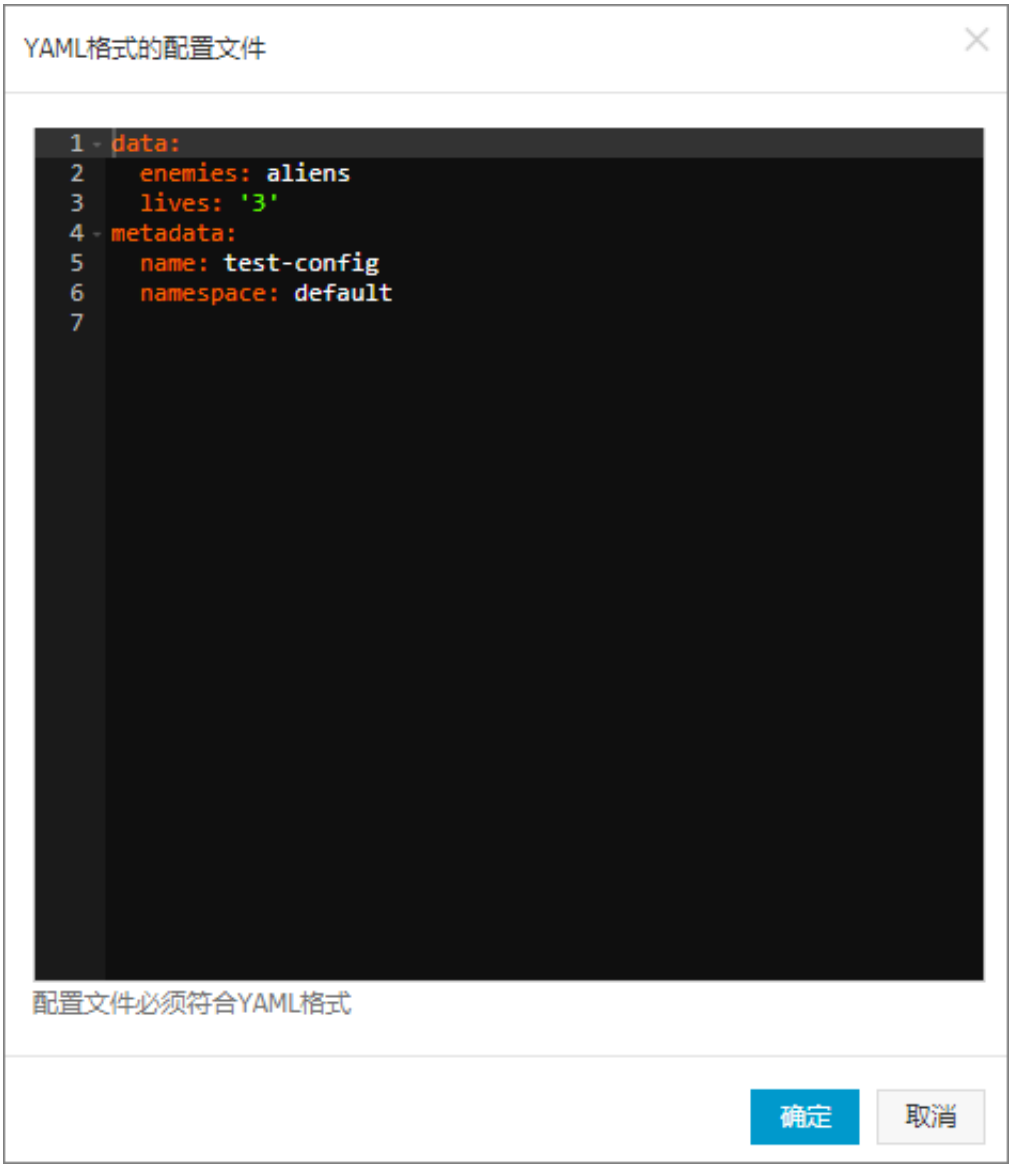
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 配置项，进入配置文件列表。
3. 在配置文件列表页面，选择需要创建配置项的集群，然后单击创建配置项。



4. 填写配置文件的信息并单击确定。
   - 命名空间：选择该配置项所属的命名空间。配置项（ConfigMap）是 kubernetes 资源对象，需要作用于命名空间。
   - 配置文件名：指定配置项的文件名，名称可以包含小写字母、数字、连字符（-）或者点号（.），名称不能为空。其他资源对象需要引用配置文件名来获取配置信息。
   - 配置项:填写变量名称和变量值后，需要单击右侧的添加。您也可以单击编辑配置文件 在弹出的对话框里编写配置项并单击确定。



本示例中设置了 enemies 和 lives 变量，分别用于传递 aliens 和 3 这两个参数。

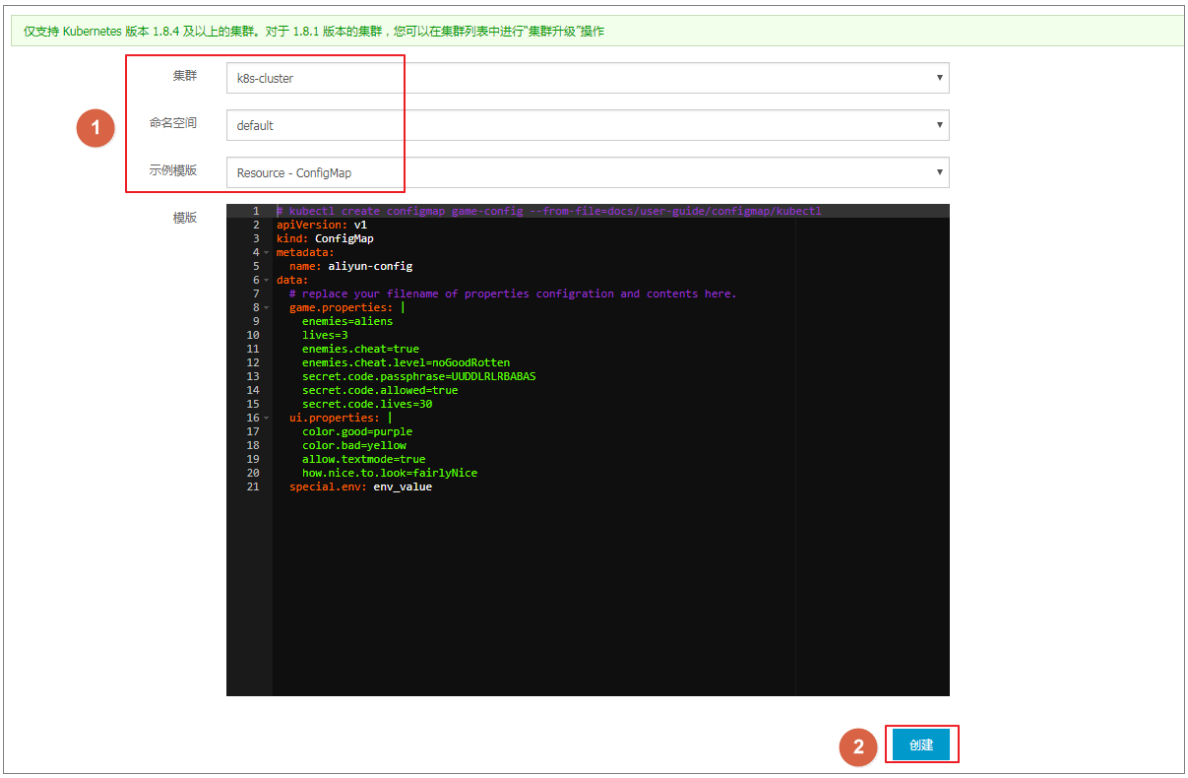


5. 单击确定后，您可以在配置文件列表中看到 test-config 配置文件。

使用模板创建

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表。
3. 单击页面右上角的使用模板创建。

4. 在使用模版部署的页面，设置配置文件的信息并单击部署。
   - 集群：选择需要创建配置项的集群。
   - 命名空间：选择该配置项所属的命名空间。配置项（ConfigMap）是 kubernetes 资源对象，需要作用于命名空间。
   - 示例模板：您可以选择自定义，根据 Kubernetes yaml 语法规则编写 ConfigMap 文件，或者选择示例模板 **resource-ConfigMap**。该示例模板的 ConfigMap 名称为 aliyun-config，包含两个变量文件 `game.properties` 和 `ui.properties`，您可以在此基础上进行修改，然后单击部署。



5. 部署成功后，您可以在配置文件列表下看到 aliyun-config 配置文件。

1.9.2 在 pod 中使用配置项

您可以在 Pod 中使用配置项，有多种使用场景，主要包括：

- 使用配置项定义 pod 环境变量
- 通过配置项设置命令行参数
- 在数据卷中使用配置项

更多关于配置项的信息，可以参见 *Configure a Pod to Use a ConfigMap*。

使用限制

您在 pod 里使用配置项时，需要两者处于同一集群和命名空间中。

创建配置项

本示例创建配置项 special_config，包含 `SPECIAL_LEVEL: very` 和 `SPECIAL_TYPE: charm` 两个键值对。

使用编排模板创建配置项

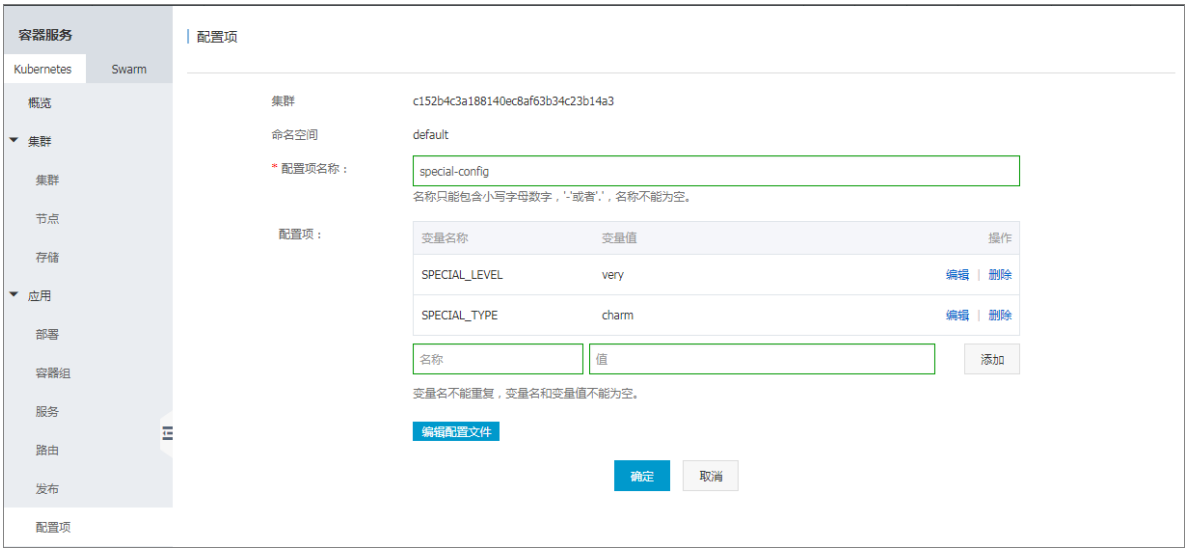
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您可以使用如下 yaml 示例模板创建配置项。

```
apiVersion: v1
kind: ConfigMap
metadata:
   name: special-config
   namespace: default
data:
   SPECIAL_LEVEL: very
   SPECIAL_TYPE: charm
```

通过 Web 界面创建配置项

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 配置项，进入配置项列表页面。
3. 选择所需的集群和命名空间，然后单击右上角的使用模板创建。
4. 输入配置项名称，然后单击添加，输入配置项，最后单击确定。



使用配置项定义 pod 环境变量

使用配置项的数据定义 pod 环境变量

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您可以在 pod 中定义环境变量，使用 `valueFrom` 引用 SPECIAL_LEVEL 的 value 值，从而定义 pod 的环境变量。

   下面是一个编排示例。

```
apiVersion: v1
kind: Pod
metadata:
   name: config-pod-1
spec:
   containers:
     - name: test-container
       image: busybox
       command: [ "/bin/sh", "-c", "env" ]
       env:
         - name: SPECIAL_LEVEL_KEY
           valueFrom:                                  ##使用valueFrom来指
定env引用配置项的value值
             configMapKeyRef:
               name: special-config                    ##引用的配置文件名称
               key: SPECIAL_LEVEL                      ##引用的配置项key
   restartPolicy: Never
```

同理，如果您需要将多个配置项的 value 值定义为 pod 的环境变量值，您只需要在 pod 中添加多个 env 参数即可。

将配置项的所有 key/values 配置为 pod 的环境变量

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   如果您想在一个 pod 中将配置项的所有 key/values 键值对配置为 pod 的环境变量，可以使用 envFrom 参数，配置项中的 key 会成为 Pod 中的环境变量名称。

   下面是一个编排示例。

```
apiVersion: v1
kind: Pod
metadata:
```

```
   name: config-pod-2
spec:
   containers:
     - name: test-container
       image: busybox
       command: [ "/bin/sh", "-c", "env" ]
       envFrom:                ##引用 sepcial-config 配置文件的所有 key
/values 键值对
       - configMapRef:
           name: special-config
   restartPolicy: Never
```

通过配置项设置命令行参数

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您可以使用配置项设置容器中的命令或者参数值，使用环境变量替换语法 `$(VAR_NAME)`来进行。

   下面是一个编排示例。

```
apiVersion: v1
kind: Pod
metadata:
   name: config-pod-3
spec:
   containers:
     - name: test-container
       image: busybox
       command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(
SPECIAL_TYPE_KEY)" ]
       env:
         - name: SPECIAL_LEVEL_KEY
           valueFrom:
             configMapKeyRef:
               name: special-config
               key: SPECIAL_LEVEL
         - name: SPECIAL_TYPE_KEY
           valueFrom:
             configMapKeyRef:
               name: special-config
               key: SPECIAL_TYPE
   restartPolicy: Never
```

   运行这个 pod 后，会输出如下结果。

```
very charm
```

在数据卷中使用配置项

1. 登录容器服务管理控制台。

2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 部署，然后单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您也可以在数据卷里面使用配置项，在 volumes 下指定配置项名称，会将 key/values 的数据存储到 mountPath 路径下（本例中是 */etc/config*）。最终生成以 key 为文件名，values 为文件内容的配置文件。

   下面是一个编排示例。

```
apiVersion: v1
kind: Pod
metadata:
   name: config-pod-4
spec:
   containers:
     - name: test-container
       image: busybox
       command: [ "/bin/sh", "-c", "ls /etc/config/" ]    ##列出该目录
下的文件名
       volumeMounts:
       - name: config-volume
         mountPath: /etc/config
   volumes:
     - name: config-volume
       configMap:
         name: special-config
   restartPolicy: Never
```

   运行 pod 后，会输出配置项的 key。

```
SPECIAL_TYPE
SPECIAL_LEVEL
```

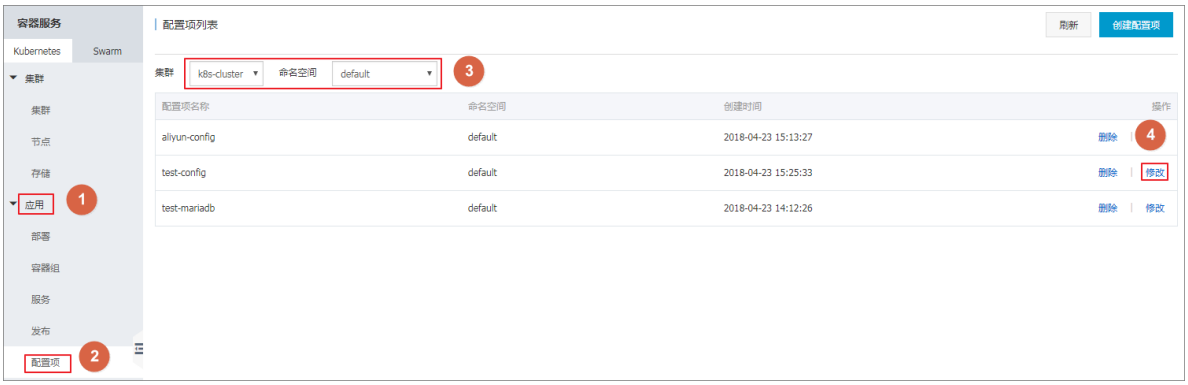
1.9.3 修改配置项

您可以修改配置项的配置。

> **说明：**
> 修改配置文件会影响使用该配置文件的应用。

通过配置项进行修改

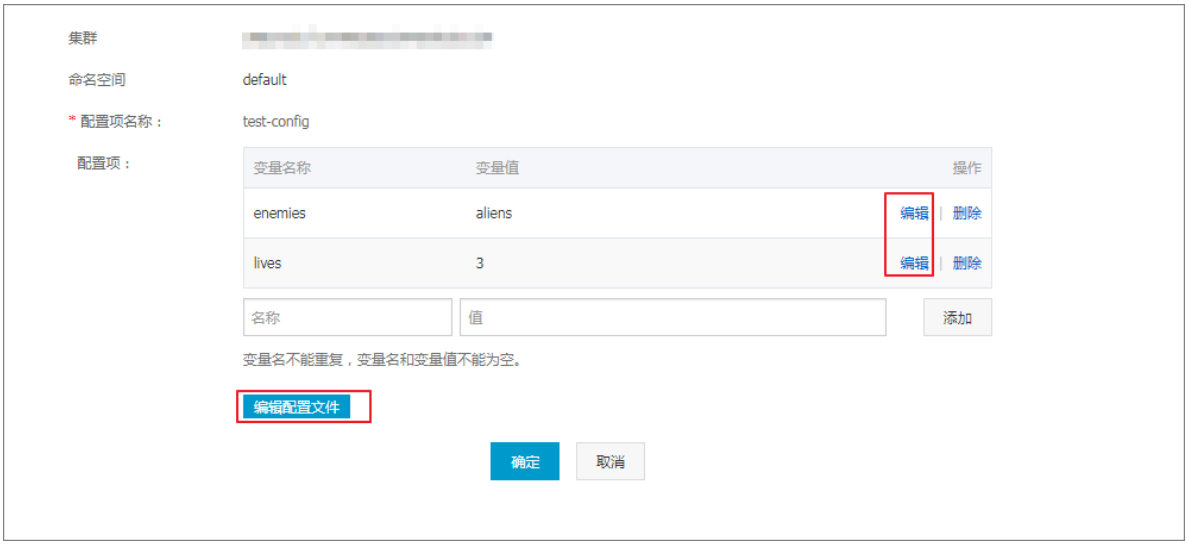
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 配置项，进入配置文件列表。
3. 选择所需的集群和命名空间，选择需要修改的配置项并单击右侧的修改。



4. 在弹出的确认对话框中，单击确定。

5. 修改配置项。

   - 选择需要修改的配置项并单击右侧的编辑，修改配置后单击保存。
   - 或者单击编辑配置文件，完成编辑后单击确定。



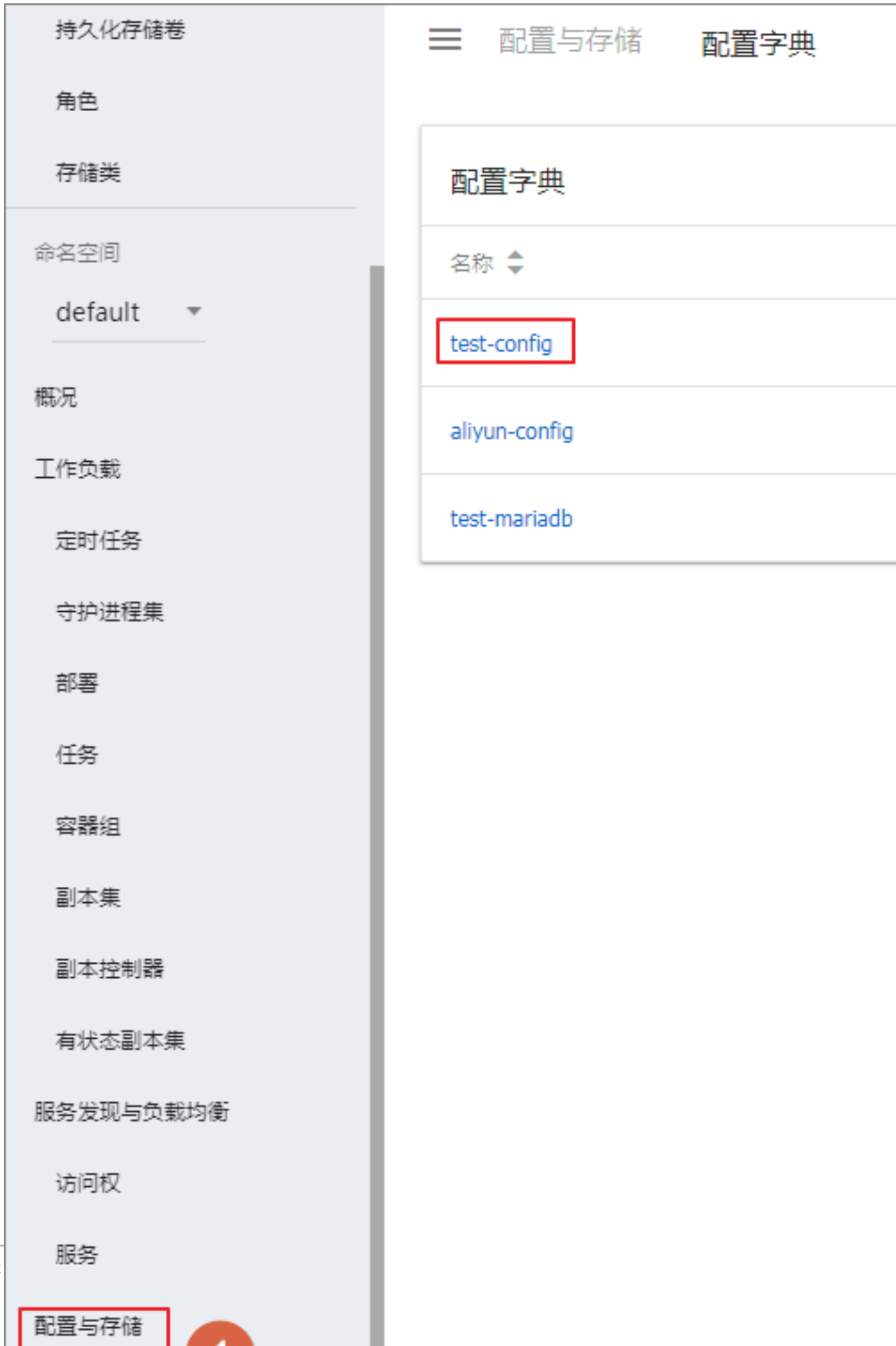
6. 完成配置项修改后，单击确定。

通过 Kubernetes Dashboard 进行修改

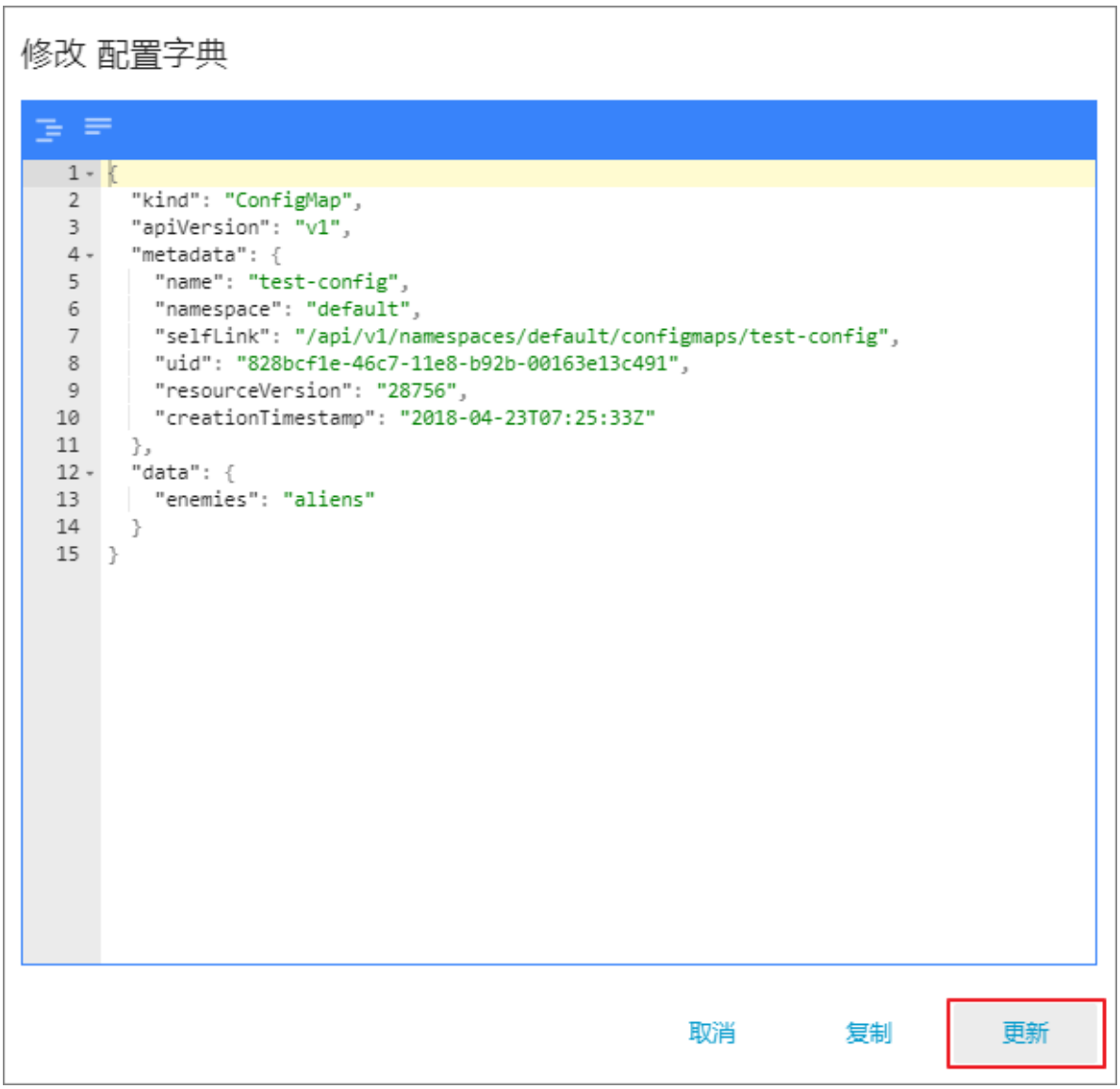
1. 登录容器服务管理控制台。

2. 在 Kubernetes 菜单下，单击左侧导航栏中的**集群**，进入集群列表。
3. 选择所需的集群，并单击右侧的**控制台**。

4. 在 Kubernetes Dashboard 页面，单击左侧导航栏中的**配置与存储 > 配置字典**，选择所需的配置字典，单击右侧的**操作 > 查看/编辑 YAML** 。


持久化存储卷
角色
存储类
命名空间
default
概况
工作负载
定时任务
守护进程集
部署
任务
容器组
副本集
副本控制器
有状态副本集
服务发现与负载均衡
访问权
服务
配置与存储
配置与存储
配置字典
配置字典
名称
test-config
aliyun-config
test-mariadb

5. 弹出修改配置字典对话框，对配置变量进行修改后，单击更新。

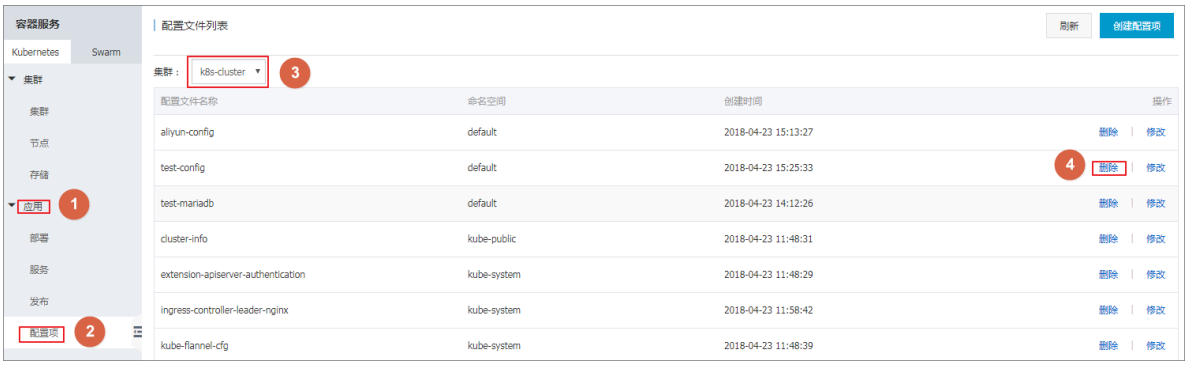


1.9.4 删除配置项

您可以删除不再使用的配置项。

通过配置项菜单进行删除

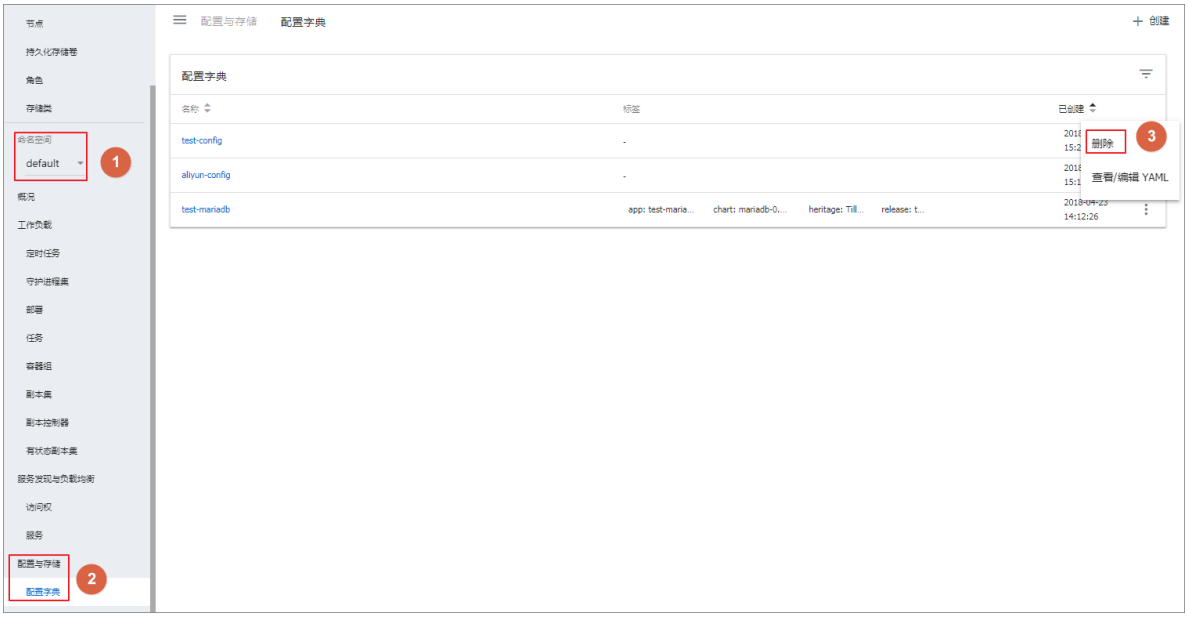
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 配置项，进入配置文件列表。
3. 选择目标集群，选择需要修改的配置项并单击右侧的删除。



通过 **Kubernetes Dashboard** 进行删除

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 单击左侧导航栏中的集群，选择目标集群，单击右侧的控制台。

4. 在 Kubernetes Dashboard 页面，选择所需的命名空间，单击左侧导航栏中的**配置与存储 > 配置字典**，单击右侧的的操作按钮，在下拉框中单击**删除** 。



5. 在弹出的对话框中，单击**删除**。

1.9.5 创建密钥

前提条件

您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。

背景信息

若您需要在 Kubernetes 集群中使用一些敏感的配置，比如密码、证书等信息时，建议使用密钥（secret），即保密字典。

密钥有多种类型，例如：

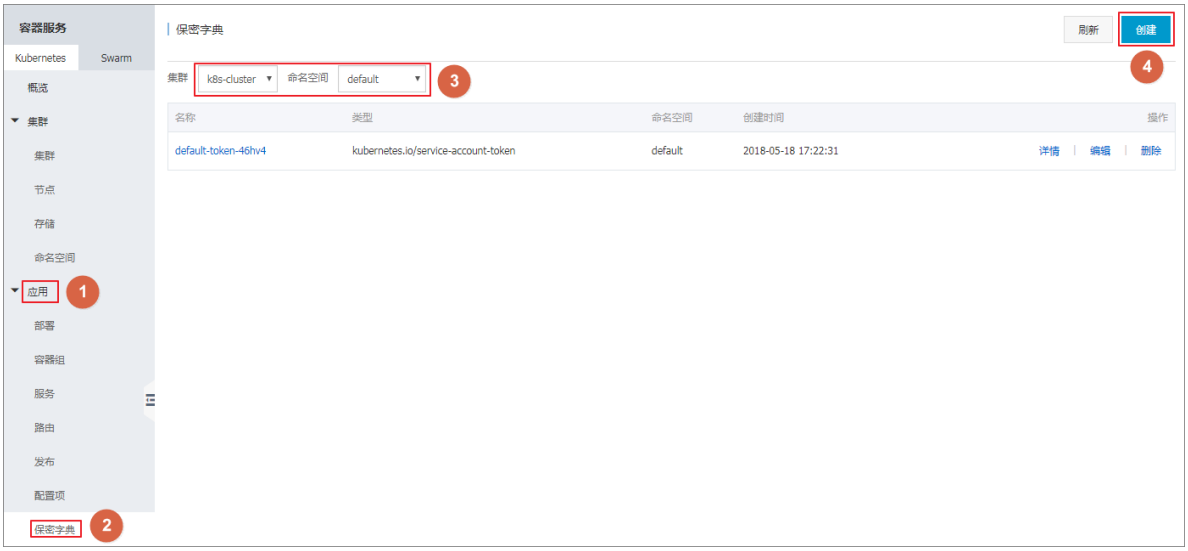
- Service Account：用来访问 Kubernetes API，由 Kubernetes 自动创建，并且会自动挂载到 Pod 的`/run/secrets/kubernetes.io/serviceaccount`目录中。
- Opaque：base64 编码格式的 Secret，用来存储密码、证书等敏感信息。

默认情况下，通过 Web 界面只能创建 Opaque 类型的密钥。Opaque 类型的数据是一个 map 类型，要求value 是 base64 编码格式。阿里云容器服务提供一键创建密钥的功能，自动将明文数据编码为 base64 格式。

您也可通过命令行手动创建密钥，请参见 *kubernetes secret* 了解更多信息。

操作步骤

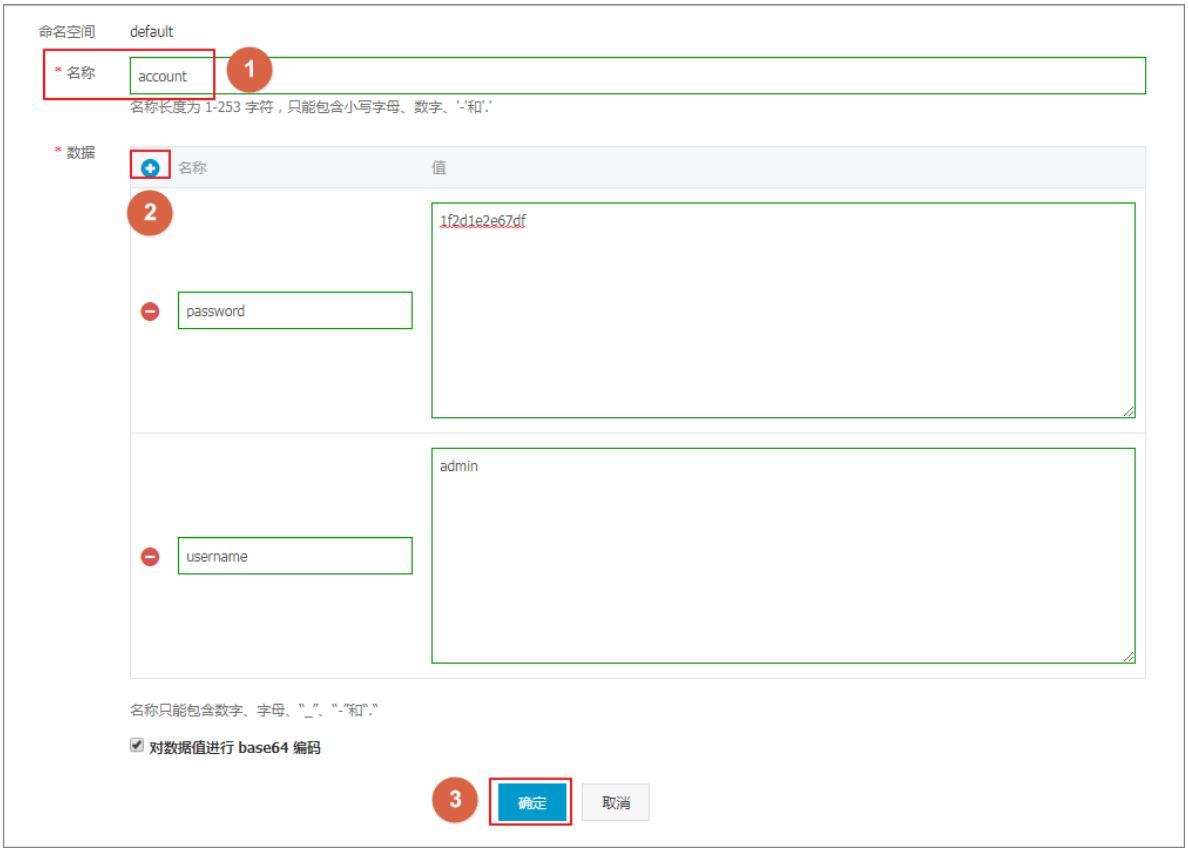
1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 保密字典，进入保密字典页面。
3. 选择所需的集群和命名空间，单击右上角的创建。



4. 配置新的保密字典。

说明：

若您输入密钥的明文数据，请注意勾选对数据值进行**base64编码**



1. 名称：输入密钥的名称。名称长度为 1-253 字符，只能包含小写字母、数字、’-‘和’.’
2. 配置密钥的数据。单击添加，在弹出的对话框中，输入密钥名称和值，即键值对。本例中创建的密钥包含两个 value，`username:admin`和 `passwrod:1f2d1e2e67df`。
3. 单击确定。

5. 默认返回保密字典页面，您可看到新建的密钥出现在列表中。

1.9.6 查看密钥

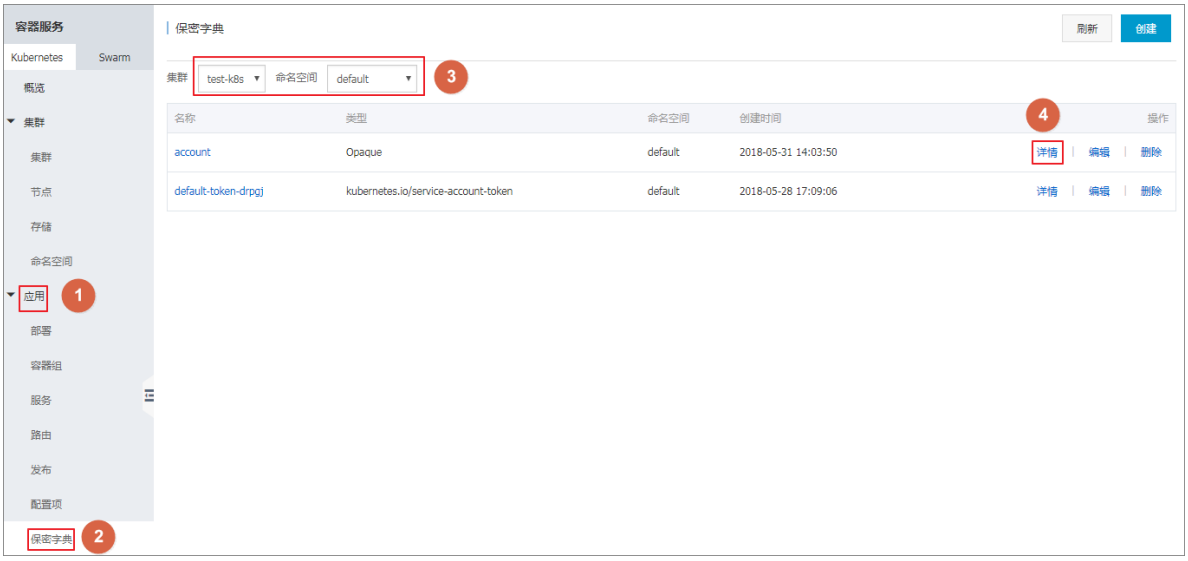
您可通过阿里云容器服务 Web 界面查看已创建的密钥。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- 您已创建一个密钥，参见创建密钥。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 保密字典，进入保密字典页面。
3. 选择所需的集群和命名空间，选择所需的密钥，并单击右侧的详情。



4. 进入密钥的详情页面，您可查看该密钥的基本信息，以及密钥包含的数据信息。

   单击详细信息中数据名称右侧的图标，可查看数据的明文。

1.9.7 编辑密钥

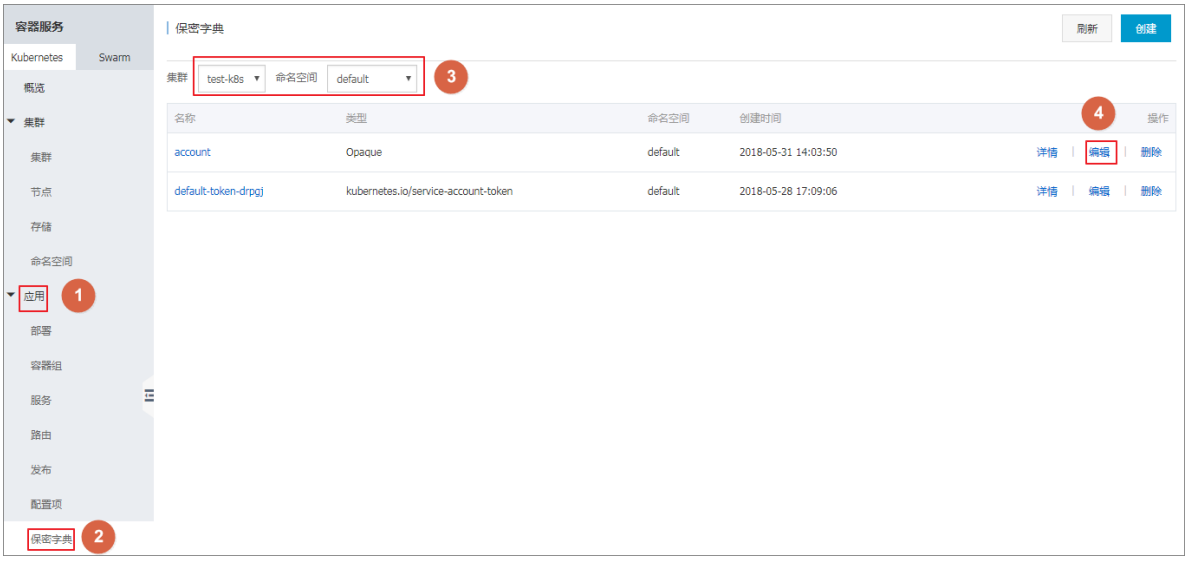
您可通过阿里云容器服务 Web 界面直接编辑已有密钥。

前提条件

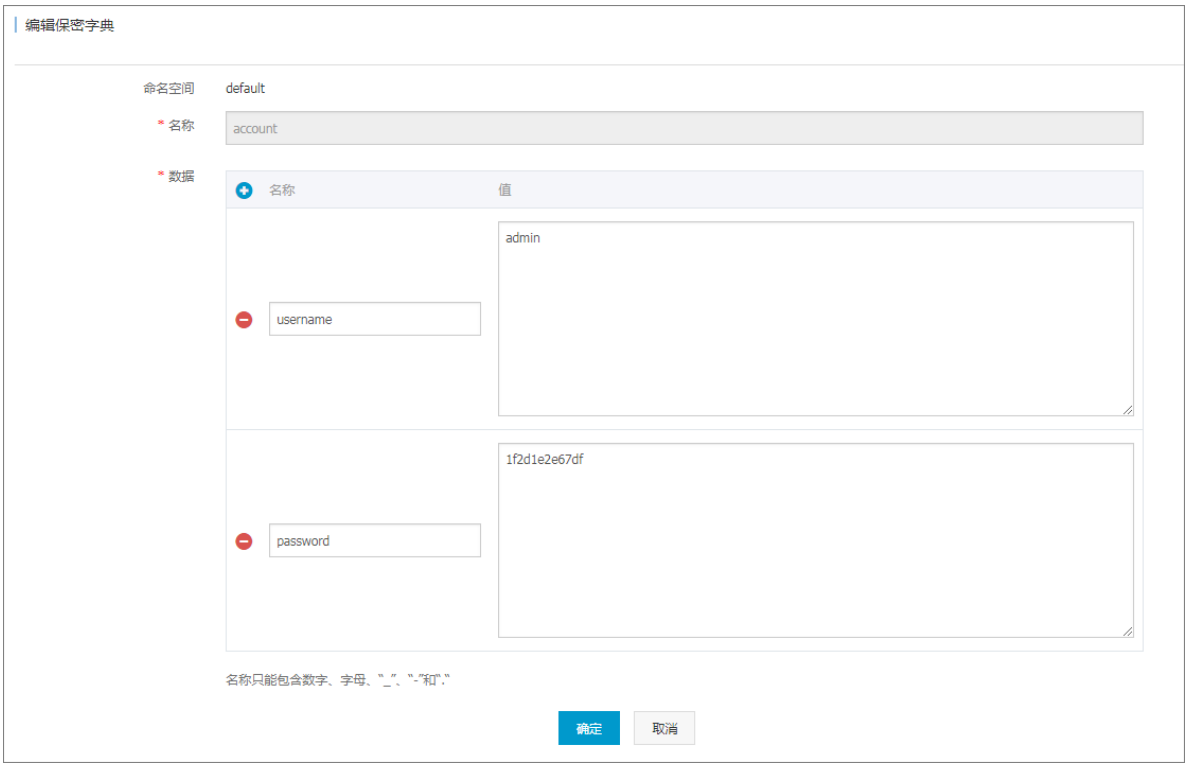
- 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- 您已创建一个密钥，参见创建密钥。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 保密字典，进入保密字典页面。
3. 选择所需的集群和命名空间，选择所需的密钥，并单击右侧的编辑。



4. 在编辑密钥的页面，可对密钥的数据进行编辑。



5. 最后单击确定。

1.9.8 删除密钥

您可通过阿里云容器服务 Web 界面删除已有密钥。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已创建一个密钥，参见创建密钥。

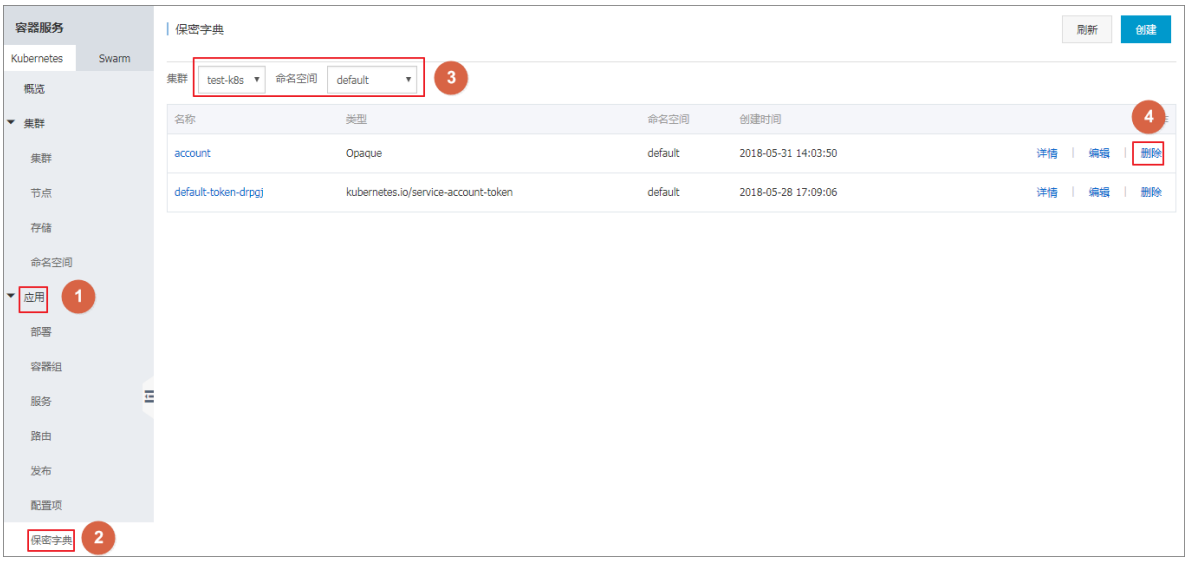
背景信息

说明：

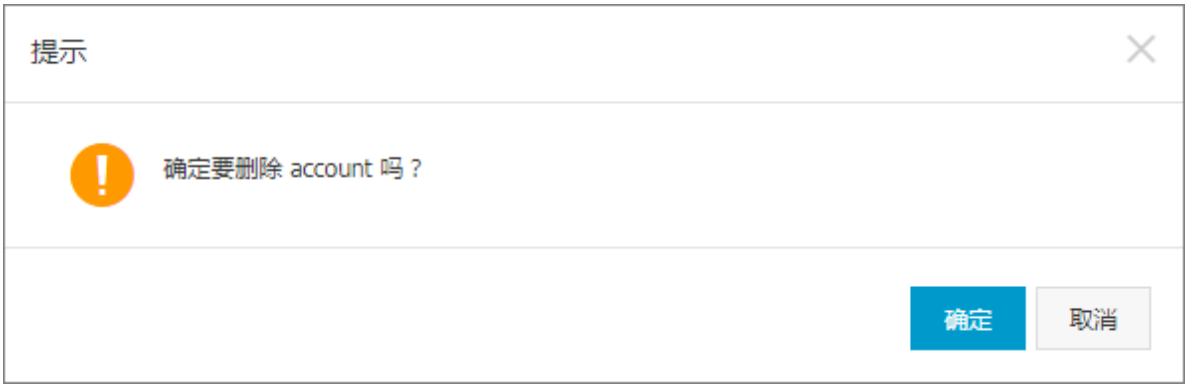
集群创建过程中生成的密钥请勿删除。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 保密字典，进入保密字典页面。
3. 选择所需的集群和命名空间，选择所需的密钥，并单击右侧的删除。



4. 在弹出的对话框中，单击确定，完成删除。



1.10 存储管理

1.10.1 概述

容器服务支持 kubernetes Pod 自动绑定阿里云云盘、NAS、 OSS 存储服务。

目前，支持静态存储卷和动态存储卷。每种数据卷的支持情况如下：

阿里云存储	静态数据卷	动态数据卷
阿里云云盘	您可以通过以下两种方式使用云盘静态存储卷： • 直接通过 volume 使用 • 通过 PV/PVC 使用	支持
阿里云 NAS	NAS 静态存储卷又支持下面两种使用方式：	支持

阿里云存储	静态数据卷	动态数据卷
	• 通过 flexvolume 插件使用 — 通过 volume 方式使用 — 使用 PV/PVC • 通过 Kubernetes 的 NFS 驱动使用	
阿里云 OSS	您可以通过以下两种方式使用 OSS 静态存储卷： • 直接使用 volume 方式 • 使用 PV/PVC	不支持

1.10.2 安装插件

通过下面的 yaml 配置部署阿里云 Kubernetes 存储插件。

> 说明：
> 如果您的 Kubernetes 集群是在 2018 年 2 月 6 日之前创建的，那么您使用数据卷之间需要先安装阿里云 Kubernetes 存储插件；如果您的 Kubernetes 集群是在 2018 年 2 月 6 日之后创建的，那么您可以直接使用数据卷，不需要再安装阿里云 Kubernetes 存储插件。

使用限制

目前支持 CentOS 7 操作系统。

注意事项

- 使用 flexvolume 需要 kubelet 关闭 `--enable-controller-attach-detach` 选项。默认阿里云 Kubernetes 集群已经关闭此选项。
- 在 kube-system 用户空间部署 flexvolume。

验证安装完成：

在 master 节点上：

- 执行命令：`kubectl get pod -n kube-system | grep flexvolume` 。输出若干（节点个数）Running 状态的 Pod 列表。
- 执行命令：`kubectl get pod -n kube-system | grep alicloud-disk-controller`。输出一个 Running 状态的 Pod 列表。

安装步骤

安装 Flexvolume：

```
apiVersion: apps/v1 # for versions before 1.8.0 use extensions/v1beta1
kind: DaemonSet
metadata:
  name: flexvolume
  namespace: kube-system
  labels:
    k8s-volume: flexvolume
spec:
  selector:
    matchLabels:
      name: acs-flexvolume
  template:
    metadata:
      labels:
        name: acs-flexvolume
    spec:
      hostPID: true
      hostNetwork: true
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      containers:
      - name: acs-flexvolume
        image: registry.cn-hangzhou.aliyuncs.com/acs/flexvolume:v1.9.7
-42e8198
        imagePullPolicy: Always
        securityContext:
          privileged: true
        env:
        - name: ACS_DISK
          value: "true"
        - name: ACS_NAS
          value: "true"
        - name: ACS_OSS
          value: "true"
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: usrdir
          mountPath: /host/usr/
        - name: etcdir
          mountPath: /host/etc/
        - name: logdir
          mountPath: /var/log/alicloud/
      volumes:
      - name: usrdir
        hostPath:
          path: /usr/
      - name: etcdir
        hostPath:
          path: /etc/
      - name: logdir
```

```
        hostPath:
           path: /var/log/alicloud/
```

安装 Disk Provisioner：

```
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-common
provisioner: alicloud/disk
parameters:
  type: cloud
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-efficiency
provisioner: alicloud/disk
parameters:
  type: cloud_efficiency
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-ssd
provisioner: alicloud/disk
parameters:
  type: cloud_ssd
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-available
provisioner: alicloud/disk
parameters:
  type: available
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: alicloud-disk-controller-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: alicloud-disk-controller
  namespace: kube-system
```

```
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: run-alicloud-disk-controller
subjects:
  - kind: ServiceAccount
    name: alicloud-disk-controller
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: alicloud-disk-controller-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: alicloud-disk-controller
  namespace: kube-system
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: alicloud-disk-controller
    spec:
      tolerations:
      - effect: NoSchedule
        operator: Exists
        key: node-role.kubernetes.io/master
      - effect: NoSchedule
        operator: Exists
        key: node.cloudprovider.kubernetes.io/uninitialized
      nodeSelector:
         node-role.kubernetes.io/master: ""
      serviceAccount: alicloud-disk-controller
      containers:
        - name: alicloud-disk-controller
          image: registry.cn-hangzhou.aliyuncs.com/acs/alicloud-disk-
controller:v1.9.3-ed710ce
          volumeMounts:
            - name: cloud-config
              mountPath: /etc/kubernetes/
            - name: logdir
              mountPath: /var/log/alicloud/
      volumes:
        - name: cloud-config
          hostPath:
            path: /etc/kubernetes/
        - name: logdir
          hostPath:
            path: /var/log/alicloud/
```

1.10.3 使用阿里云云盘

您可以在阿里云容器服务 Kubernetes 集群中使用阿里云云盘存储卷。

目前，阿里云云盘提供两种 Kubernetes 挂载方式：

- 静态存储卷

  您可以通过以下两种方式使用云盘静态存储卷：

  - 直接通过*volume*使用
  - 通过 *PV/PVC* 使用

- 动态存储卷

说明：

对创建的云盘容量有如下要求：

- 普通云盘：最小5Gi
- 高效云盘：最小20Gi
- SSD云盘：最小20Gi

静态存储卷

您可以直接通过volume使用阿里云云盘存储卷或者通过 PV/PVC 使用阿里云云盘存储卷。

前提条件

使用云盘数据卷之前，您需要先在 ECS 管理控制台上创建云盘。有关如何创建云盘，参见创建云盘。

使用说明

- 云盘为非共享存储，只能同时被一个 pod 挂载。
- 使用云盘存储卷前需要先申请一个云盘，并获得磁盘 ID。参见 创建云盘。
- volumeId： 表示所挂载云盘的磁盘ID；volumeName、PV Name要与之相同。
- 集群中只有与云盘在同一个可用区（Zone）的节点才可以挂载云盘。

直接通过 volume 使用

使用*disk-deploy.yaml*文件创建 Pod。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-disk-deploy
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
```

```
    spec:
      containers:
      - name: nginx-flexvolume-disk
        image: nginx
        volumeMounts:
          - name: "d-bp1j17ifxfasvts3tf40"
            mountPath: "/data"
      volumes:
        - name: "d-bp1j17ifxfasvts3tf40"
          flexVolume:
            driver: "alicloud/disk"
            fsType: "ext4"
            options:
              volumeId: "d-bp1j17ifxfasvts3tf40"
```

通过 PV/PVC 使用

步骤 1 创建云盘类型的 PV

您可以使用 yaml 文件或者控制台界面创建云盘类型的 PV。

通过 yaml 文件创建 PV

使用 `disk-pv.yaml` 文件创建 PV。

说明：

pv name 要与阿里云盘 ID 相同。

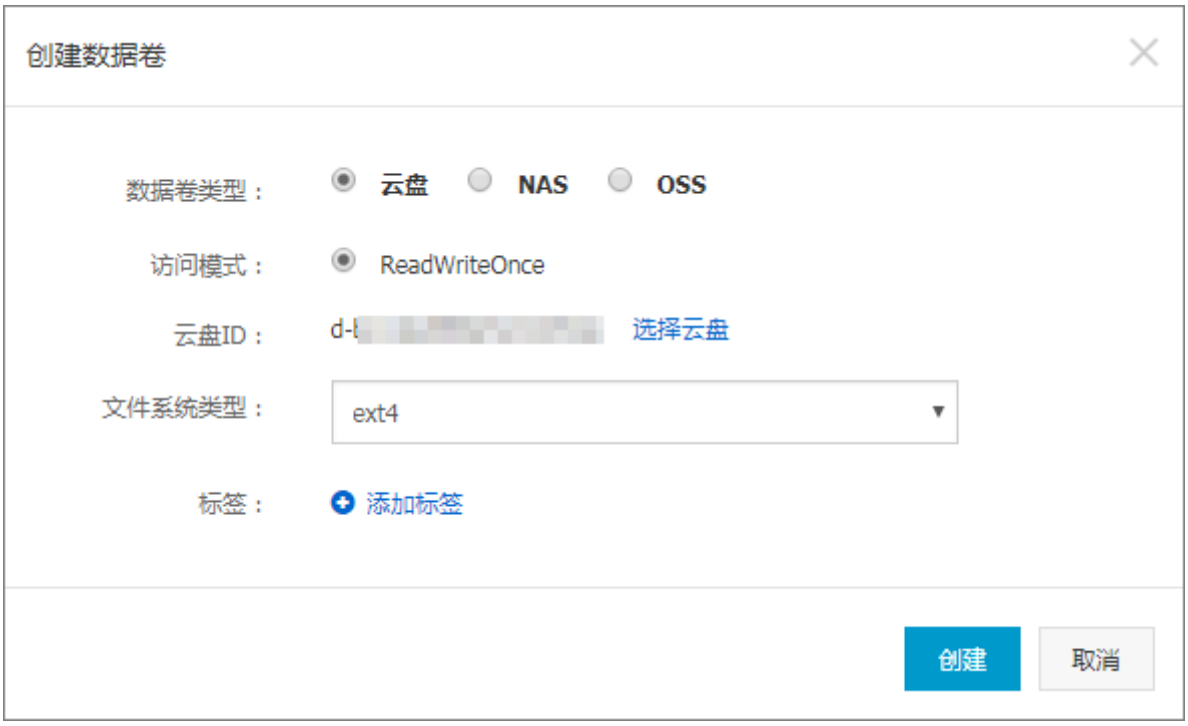
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: d-bp1j17ifxfasvts3tf40
  labels:
    failure-domain.beta.kubernetes.io/zone: cn-hangzhou-b
    failure-domain.beta.kubernetes.io/region: cn-hangzhou
spec:
  capacity:
    storage: 20Gi
  storageClassName: disk
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "alicloud/disk"
    fsType: "ext4"
    options:
      volumeId: "d-bp1j17ifxfasvts3tf40"
```

通过控制台界面创建云盘数据卷

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 存储卷，进入数据卷列表页面。
3. 选择所需的集群，单击页面右上角的创建。

4. 在创建数据卷对话框中，配置数据卷的相关参数。

   - 数据卷类型：本示例中为云盘。
   - 访问模式：默认为 ReadWriteOnce。
   - 云盘 ID：您可以选择与集群属于相同地域和可用区下处于待挂载状态的云盘。
   - 文件系统类型：您可以选择以什么数据类型将数据存储到云盘上，支持的类型包括 ext4、ext3、xfs、vfat。默认为 ext4。
   - 标签：为该数据卷添加标签。



5. 完成配置后，单击创建。

步骤 2 创建 PVC

使用 `disk-pvc.yaml` 文件创建 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-disk
spec:
  accessModes:
```

```
    - ReadWriteOnce
  storageClassName: disk
  resources:
    requests:
      storage: 20Gi
```

步骤 **3** 创建 **Pod**

使用*disk-pod.yaml*文件创建 pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: "flexvolume-alicloud-example"
spec:
  containers:
    - name: "nginx"
      image: "nginx"
      volumeMounts:
        - name: pvc-disk
          mountPath: "/data"
  volumes:
  - name: pvc-disk
    persistentVolumeClaim:
      claimName: pvc-disk
```

动态存储卷

动态存储卷需要您手动创建 StorageClass，并在PVC中通过 storageClassName 来指定期望的云盘类型。

创建 **StorageClass**

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-common-hangzhou-b
provisioner: alicloud/disk
parameters:
  type: cloud_ssd
  regionid: cn-hangzhou
  zoneid: cn-hangzhou-b
```

参数说明：

- provisioner：配置为 alicloud/disk，标识使用阿里云云盘 Provsioner 插件创建。
- type：标识云盘类型，支持 cloud、cloud_efficiency、cloud_ssd、available 四种类型；其中 available 会对高效、SSD、普通云盘依次尝试创建，直到创建成功。
- regionid：期望创建云盘的区域。
- reclaimPolicy: 云盘的回收策略，默认为Delete，支持Retain。
- zoneid：期望创建云盘的可用区。

创建服务

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-common
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: alicloud-disk-common-hangzhou-b
  resources:
    requests:
      storage: 20Gi
---
kind: Pod
apiVersion: v1
metadata:
  name: disk-pod-common
spec:
  containers:
  - name: disk-pod
    image: nginx
    volumeMounts:
      - name: disk-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: disk-pvc
      persistentVolumeClaim:
        claimName: disk-common
```

默认选项

集群默认提供了下面几种 StorageClass，可以在单 AZ 类型的集群中使用。

- alicloud-disk-common：普通云盘。
- alicloud-disk-efficiency：高效云盘。
- alicloud-disk-ssd：SSD云盘。
- alicloud-disk-available：提供高可用选项，先试图创建高效云盘；如果相应AZ的高效云盘资源售尽，再试图创建SSD盘；如果SSD售尽，则试图创建普通云盘。

使用云盘创建多实例**StatefulSet**

使用 volumeClaimTemplates 的方式来创建，这样会动态创建多个 PVC 和 PV 并绑定。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
```

```
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-common
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-common
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "alicloud-disk-common"
      resources:
        requests:
          storage: 10Gi
```

1.10.4 使用阿里云 NAS

您可以在容器服务 Kubernetes 集群中使用阿里云 NAS 数据卷。

目前阿里云 NAS 提供两种 Kubernetes 挂载方式：

- 静态存储卷

  其中，静态存储卷又支持下面两种使用方式：

  - 通过 flexvolume 插件使用
    - 通过 volume 方式使用
    - 使用 PV/PV
  - 通过 Kubernetes 的 NFS 驱动使用

- 动态存储卷

前提条件

使用 NAS 数据卷之前，您需要先在 NAS 管理控制台上创建文件系统，并在文件系统中添加 Kubernetes 集群的挂载点。创建的 NAS 文件系统需要和您的集群位于同一 VPC。

静态存储卷

您可以通过阿里云提供的 flexvolume 插件使用阿里云 NAS 文件存储服务或者通过 Kubernetes 的 NFS 驱动使用阿里云 NAS 文件存储服务。

通过 flexvolume 插件使用

使用 flexvolume 插件，您可以通过 volume 方式使用阿里云 NAS 数据卷或者通过 PV/PVC 方式使用阿里云 NAS 数据卷。

说明：

- NAS 为共享存储，可以同时为多个 Pod 提供共享存储服务。
- server：为 NAS 数据盘的挂载点。
- path：为连接 NAS 数据卷的挂载目录，支持挂载 NAS 子目录，且当子目录不存在时，会自动创建子目录并挂载。
- vers：定义 nfs 挂载协议的版本号，支持3.0 和 4.0。
- mode：定义挂载目录的访问权限，注意挂载 NAS 盘根目录时不能配置挂载权限。当 NAS 盘中数据量很大时，配置 mode 会导致执行挂载非常慢，甚至挂载失败。

通过 volume 方式使用

使用 `nas-deploy.yaml` 文件创建 Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: "flexvolume-nas-example"
spec:
  containers:
    - name: "nginx"
      image: "nginx"
      volumeMounts:
        - name: "nas1"
          mountPath: "/data"
  volumes:
    - name: "nas1"
      flexVolume:
        driver: "alicloud/nas"
        options:
          server: "0cd8b4a576-grs79.cn-hangzhou.nas.aliyuncs.com"
          path: "/k8s"
```

```
          vers: "4.0"
```

使用 **PV/PVC**

步骤 **1** 创建 **PV**

您可以使用 yaml 文件或者通过阿里云容器服务控制台界面创建 NAS 数据卷。

- 使用 yaml 文件创建 PV

  使用 `nas-pv.yaml` 文件创建 PV。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nas
spec:
  capacity:
    storage: 5Gi
  storageClassName: nas
  accessModes:
    - ReadWriteMany
  flexVolume:
    driver: "alicloud/nas"
    options:
      server: "0cd8b4a576-uih75.cn-hangzhou.nas.aliyuncs.com"
      path: "/k8s"
      vers: "4.0"
```

- 通过控制台界面创建 NAS 数据卷

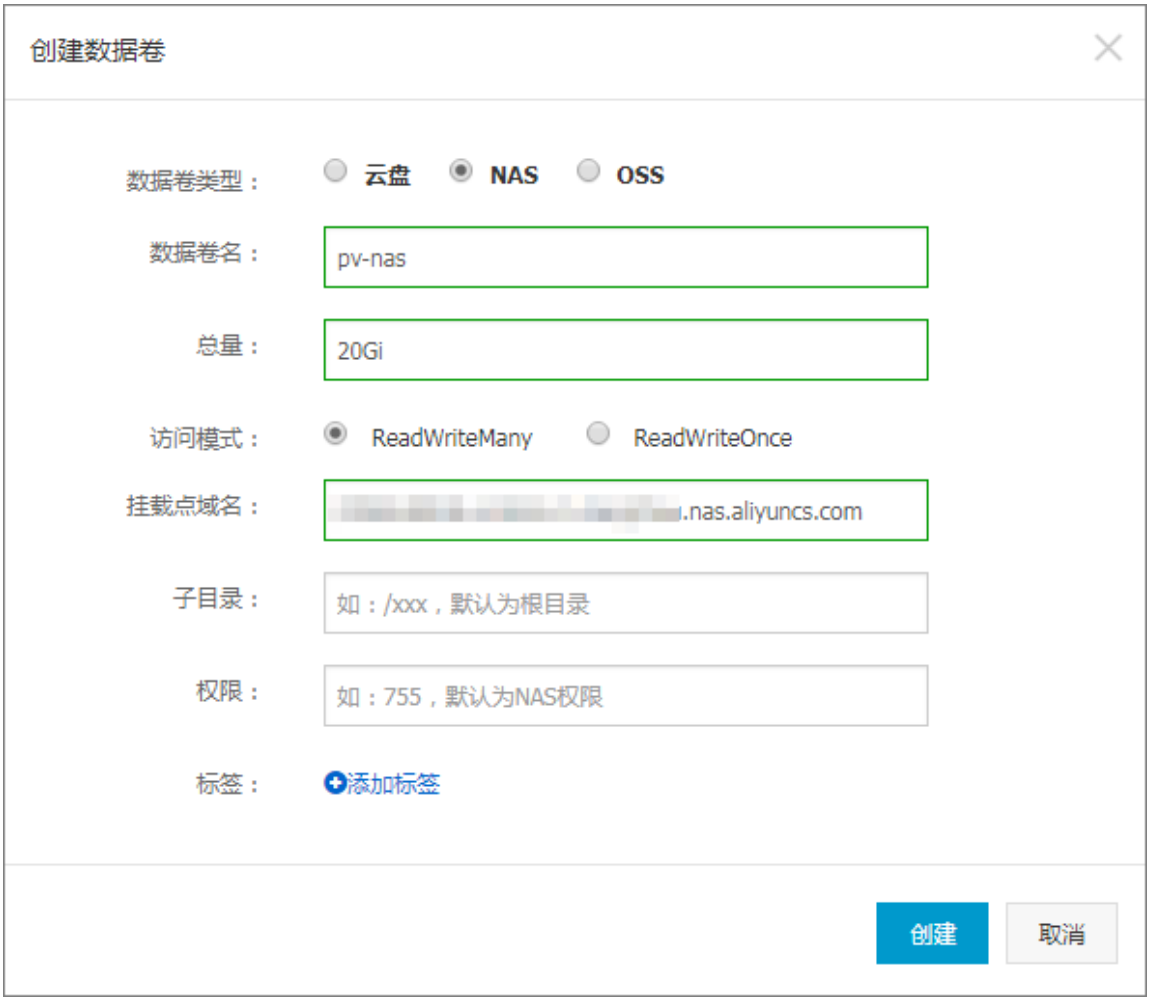
  1. 登录容器服务管理控制台。
  2. 在 Kubernetes 菜单下，单击左侧导航栏中的**集群 > 存储**，进入数据卷列表页面。
  3. 选择所需的集群，单击页面右上角的**创建**。

  4. 在创建数据卷对话框中，配置数据卷的相关参数。

     - **数据卷类型**：本示例中为NAS。
     - **数据卷名**：创建的数据卷的名称。数据卷名在集群内必须唯一。本例为 pv-nas。
     - **数据卷总量**：所创建数据卷的容量。注意不能超过磁盘容量。
     - **访问模式**：默认为 ReadWriteMany。
     - **挂载点域名**：集群在 NAS 文件系统中挂载点的挂载地址。
     - **子目录**：NAS 路径下的子目录，以 / 开头，设定后数据卷将挂载到指定的子目录。

- 如果 NAS 根目录下没有此子目录，会默认创建后再挂载。
- 您可以不填此项，默认挂载到 NAS 根目录。

- 权限：设置挂载目录的访问权限，例如：755、644、777 等。
  - 只有挂载到 NAS 子目录时才能设置权限，挂载到根目录时不能设置。
  - 您可以不填此项，默认权限为 NAS 文件原来的权限。
- 标签：为该数据卷添加标签。



5. 完成配置后，单击**创建**。

步骤 2 创建 PVC

使用 *nas-pvc.yaml* 文件创建 PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nas
spec:
  accessModes:
```

```
    - ReadWriteMany
  storageClassName: nas
  resources:
    requests:
      storage: 5Gi
```

步骤 3 创建 Pod

使用*nas-pod.yaml*文件创建 pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: "flexvolume-nas-example"
spec:
  containers:
    - name: "nginx"
      image: "nginx"
      volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
  volumes:
  - name: pvc-nas
    persistentVolumeClaim:
        claimName: pvc-nas
```

通过 Kubernetes 的 NFS 驱动使用

步骤 1 创建 NAS 文件系统

登录 文件存储管理控制台，创建一个 NAS 文件系统。

说明：

创建的 NAS 文件系统需要和您的集群位于同一地域。

假设您的挂载点为 `055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com`。

步骤 2 创建 PV

您可以使用编排模板或者通过阿里云容器服务控制台界面创建 NAS 数据卷。

- 使用编排模板

  使用*nas-pv.yaml*文件创建 PV。

  执行以下命令创建一个类型为 NAS 的 PesistentVolume。

```
root@master # cat << EOF |kubectl apply -f -
apiVersion: v1
kind: PersistentVolume
metadata:
   name: nas
spec:
```

```
    capacity:
      storage: 8Gi
    accessModes:
      - ReadWriteMany
    persistentVolumeReclaimPolicy: Retain
    nfs:
      path: /
      server: 055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com
EOF
```

- 通过控制台界面创建 **NAS** 数据卷

  具体操作参见 使用*PV/PVC* 中关于通过控制台界面创建 NAS 数据卷的内容。

步骤 **2** 创建 **PVC**

创建一个 PersistentVolumeClaim 来请求绑定该 PersistentVolume。

```
root@master # cat << EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
     name: nasclaim
spec:
     accessModes:
       - ReadWriteMany
     resources:
       requests:
         storage: 8Gi
EOF
```

步骤 **3** 创建 **Pod**

创建一个应用来申明挂载使用该数据卷。

```
root@master # cat << EOF |kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
     name: mypod
spec:
     containers:
       - name: myfrontend
         image: registry.aliyuncs.com/spacexnice/netdia:latest
         volumeMounts:
         - mountPath: "/var/www/html"
           name: mypd
     volumes:
       - name: mypd
         persistentVolumeClaim:
           claimName: nasclaim
EOF
```

至此，您就将 NAS 远程文件系统挂载到了您的 Pod 应用当中了。

动态存储卷

使用动态 NAS 存储卷需要您手动安装驱动插件，并配置 NAS 挂载点。

说明：

动态生成NAS存储卷的本质是在一个已有的文件系统上，自动生成一个目录，这个目录定义为目标存储卷。

安装插件

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-nas
mountOptions:
- vers=4.0
provisioner: alicloud/nas
reclaimPolicy: Retain

---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: alicloud-nas-controller
  namespace: kube-system
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: alicloud-nas-controller
    spec:
      tolerations:
      - effect: NoSchedule
        operator: Exists
        key: node-role.kubernetes.io/master
      - effect: NoSchedule
        operator: Exists
        key: node.cloudprovider.kubernetes.io/uninitialized
      nodeSelector:
         node-role.kubernetes.io/master: ""
      serviceAccount: admin
      containers:
        - name: alicloud-nas-controller
          image: registry.cn-hangzhou.aliyuncs.com/acs/alicloud-nas-
controller:v3.1.0-k8s1.11
          volumeMounts:
          - mountPath: /persistentvolumes
            name: nfs-client-root
          env:
            - name: PROVISIONER_NAME
              value: alicloud/nas
            - name: NFS_SERVER
              value: 0cd8b4a576-mmi32.cn-hangzhou.nas.aliyuncs.com
            - name: NFS_PATH
```

```
              value: /
      volumes:
        - name: nfs-client-root
          nfs:
            server: 0cd8b4a576-mmi32.cn-hangzhou.nas.aliyuncs.com
            path: /
```

使用动态存储卷

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  volumeClaimTemplates:
  - metadata:
      name: html
    spec:
      accessModes:
        - ReadWriteOnce
      storageClassName: alicloud-nas
      resources:
        requests:
          storage: 2Gi
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:alpine
        volumeMounts:
        - mountPath: "/usr/share/nginx/html/"
          name: html
```

1.10.5 使用阿里云 OSS

您可以在阿里云容器服务 Kubernetes 集群中使用阿里云 OSS 数据卷。

目前，仅支持 OSS 静态存储卷，不支持 OSS 动态存储卷。您可以通过以下方式使用 OSS 静态存储卷：

- 直接使用 volume 方式
- 使用 PV/PVC

前提条件

使用 OSS 静态存储卷之前，您需要先在 OSS 管理控制台上创建 Bucket。

使用说明

- OSS 为共享存储，可以同时为多个 Pod 提供共享存储服务。
- bucket：目前只支持挂载 Bucket，不支持挂载 Bucket 下面的子目录或文件。
- url: OSS endpoint，挂载 OSS 的接入域名。
- akId: 用户的 access id 值。
- akSecret：用户的 access secret 值。
- otherOpts: 挂载 OSS 时支持定制化参数输入，格式为: `-o *** -o ***`。

注意事项

如果您的 Kubernetes 集群是在 2018 年 2 月 6 日之前创建的，那么您使用数据卷之间需要先安装安装插件。使用 OSS 数据卷必须在部署 flexvolume 服务的时候创建 Secret，并输入 AK 信息。

使用 **OSS** 静态卷

直接使用 **volume** 方式

使用*oss-deploy.yaml* 文件创建 Pod。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-oss-deploy
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-flexvolume-oss
        image: nginx
        volumeMounts:
          - name: "oss1"
            mountPath: "/data"
      volumes:
        - name: "oss1"
          flexVolume:
            driver: "alicloud/oss"
            options:
              bucket: "docker"
              url: "oss-cn-hangzhou.aliyuncs.com"
              akId: ***
              akSecret: ***
              otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

使用 **PV/PVC**（目前不支持动态 **pv**）

步骤 **1** 创建 **PV**

您可以使用 yaml 文件或者通过容器服务控制台界面创建 PV。

使用 **yaml** 文件创建 **PV**

使用*oss-pv.yaml*文件创建 PV。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-oss
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  flexVolume:
    driver: "alicloud/oss"
    options:
      bucket: "docker"
      url: "oss-cn-hangzhou.aliyuncs.com"
      akId: ***
      akSecret: ***
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

通过控制台界面创建 **OSS** 数据卷

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**集群** > **存储**，进入数据卷列表页面。
3. 选择所需的集群，单击页面右上角的**创建**。

4. 在创建数据卷对话框中，配置数据卷的相关参数。
   - **数据卷类型**：本示例中为 OSS。
   - **数据卷名**：创建的数据卷的名称。数据卷名在集群内必须唯一。本例为 pv-oss。
   - **数据卷总量**：所创建数据卷的容量。
   - **访问模式**：默认为 ReadWriteMany。
   - **AccessKey ID**、**AccessKey Secret**：访问 OSS 所需的 AccessKey。
   - **Bucket ID**：您要使用的 OSS bucket 的名称。单击选择**Bucket**，在弹出的对话框中选择所需的 bucket 并单击选择。

- **访问域名**：如果 Bucket 和 ECS 实例位于不同地域（Region），请选择**外网域名**；如果位于相同地域，需要根据集群网络类型进行选择，若是 VPC 网络，请选择**VPC域名**，若是经典网络，请选择**内网域名**。
- **标签**：为该数据卷添加标签。

5. 完成配置后，单击**创建**。

步骤 **2** 创建 **PVC**

使用*oss-pvc.yaml*文件创建 PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-oss
```

```
spec:
  storageClassName: oss
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

步骤 3 创建 Pod

使用*oss-pod.yaml* 创建 Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: "flexvolume-oss-example"
spec:
  containers:
    - name: "nginx"
      image: "nginx"
      volumeMounts:
          - name: pvc-oss
            mountPath: "/data"
  volumes:
  - name: pvc-oss
    persistentVolumeClaim:
        claimName: pvc-oss
```

使用 OSS 动态卷

目前暂不支持。

1.10.6 创建持久化存储卷声明

您可通过容器服务控制台创建持久化存储卷声明（PVC）。

前提条件

- 您已创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已创建一个存储卷，本例中使用云盘创建一个云盘存储卷，参见使用阿里云云盘。

  默认根据标签**alicloud-pvname**将存储声明和存储卷绑定，通过容器服务控制台创建数据卷时，会默认给存储卷打上该标签。如果存储卷上没有该标签，您需要添加标签后才可以选择关联这个存储卷。

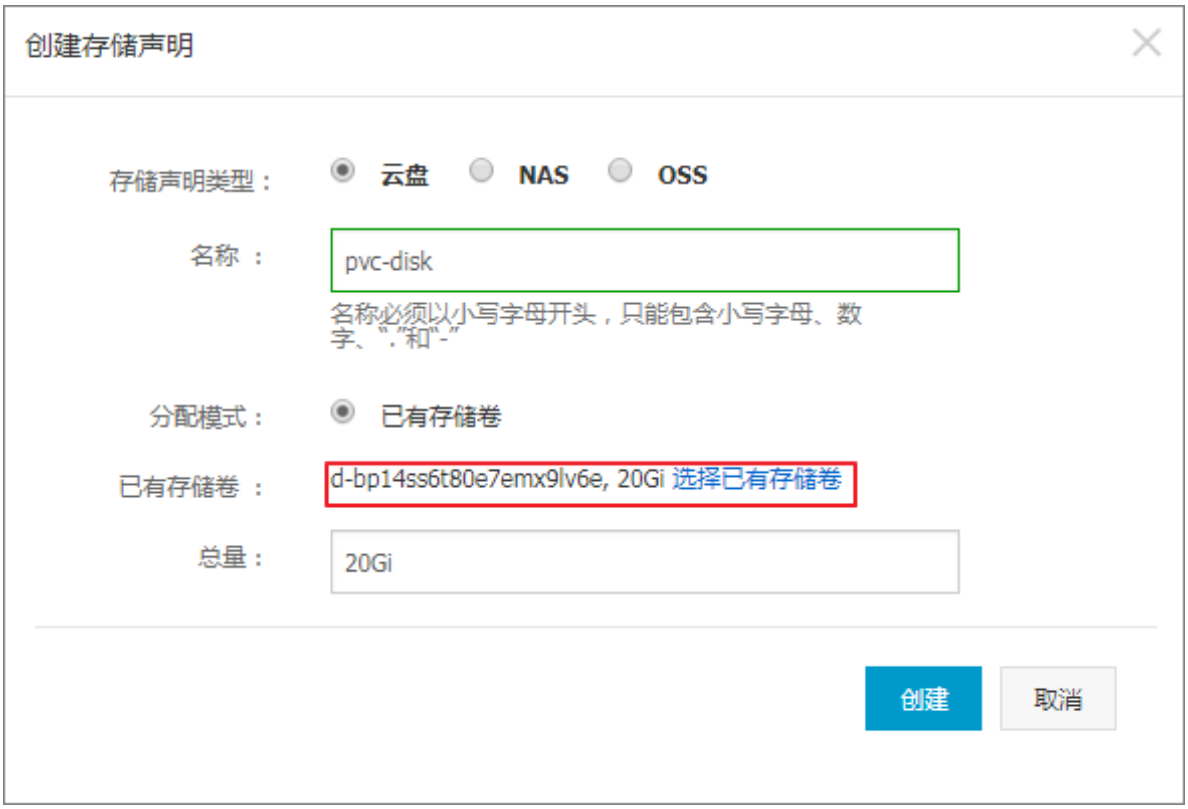
背景信息

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 存储声明，进入存储声明列表页面。

3. 选择所需的集群和命名空间，单击右上角的创建。

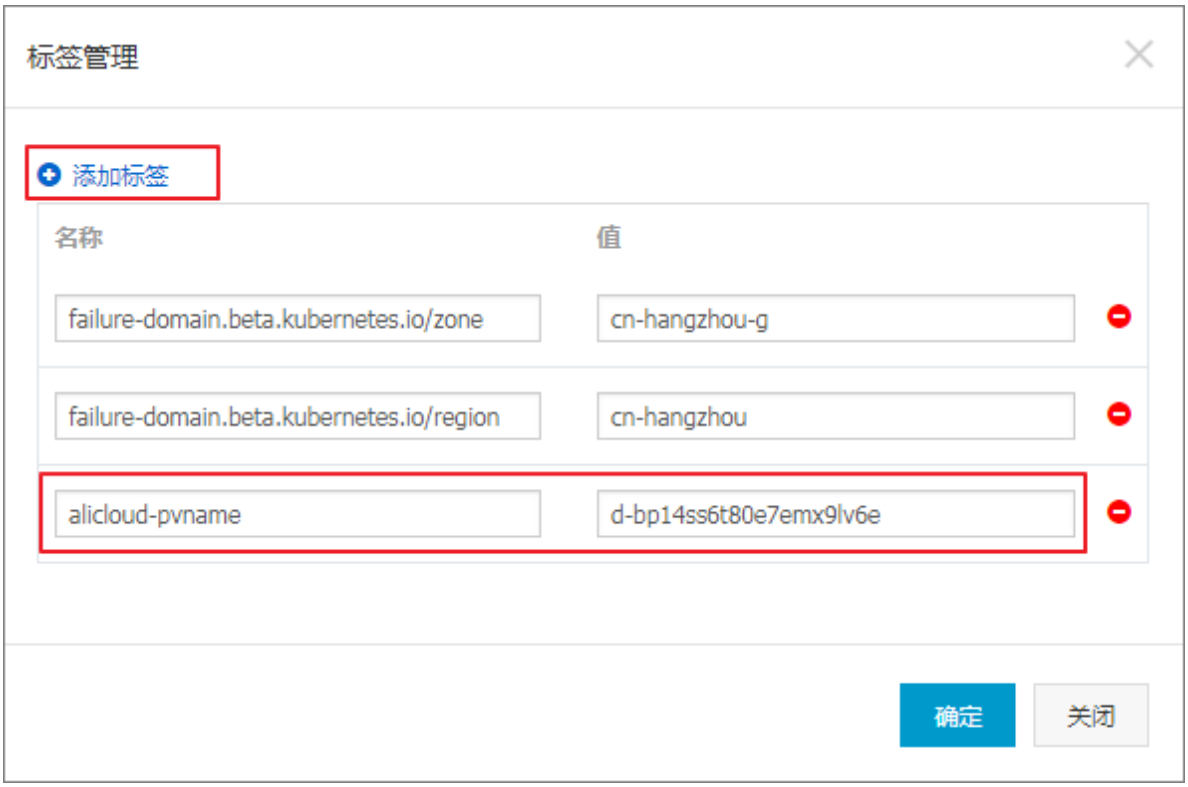
4. 在创建存储声明对话框中进行配置，最后单击创建。



- 存储声明类型： 和存储卷一致， 包括云盘/NAS/OSS
- 名称： 输入存储卷声明名称
- 分配模式： 目前只支持已有存储卷
- 已有存储卷： 选择该类型下的存储卷绑定
- 总量： 声明使用量，不能大于存储卷的总量

> **说明：**
> 若您的集群中已有存储卷，且未被使用，但在选择已有存储卷无法找到，则可能是未定义**alicloud-pvname**标签。

若无法找到可用的存储卷，您可在左侧导航栏中单击集群 > 存储卷，找到所需存储卷，单击右侧的标签管理，添加对应标签，其中名称为**alicloud-pvname**，值为存储卷的名称，云盘存储卷默认以云盘ID作为存储卷的名称。



5. 返回存储声明列表，您可看到新建的存储声明出现在列表中。

1.10.7 使用持久化存储卷声明

您可通过容器服务控制台，通过镜像或模板部署应用，从而使用持久化存储声明。本例中使用镜像来创建应用，若您想通过模板使用持久化存储卷声明，请参见使用阿里云云盘。

前提条件

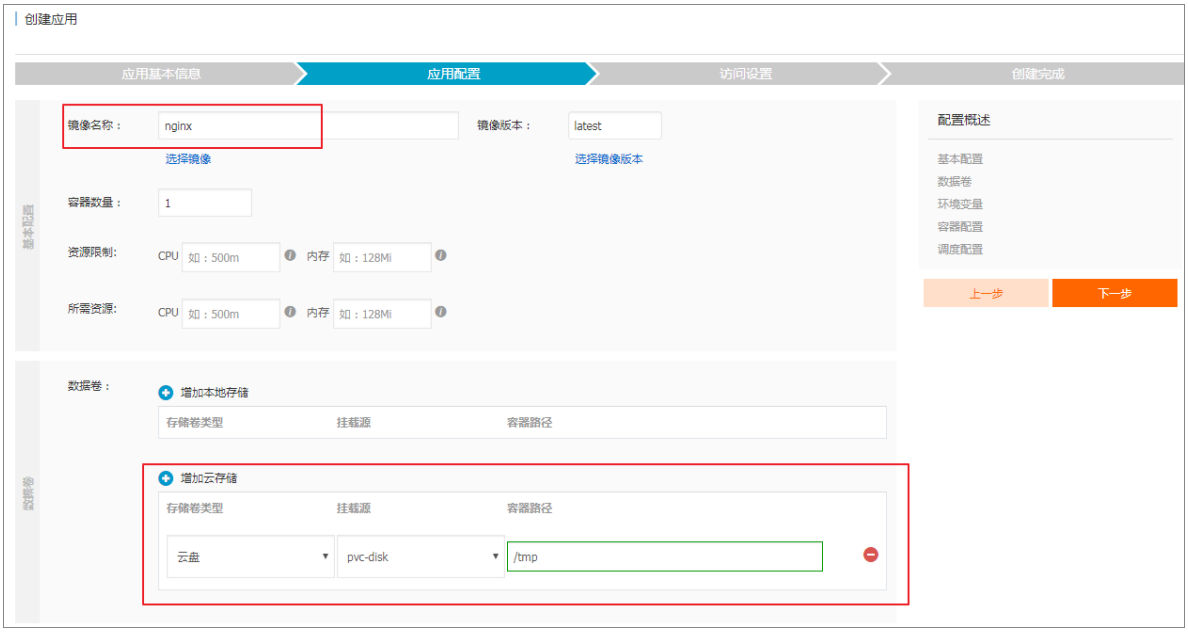
- 您已创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已创建一个存储卷声明，本例中使用云盘创建一个云盘存储卷声明pvc-disk，参见创建持久化存储卷声明。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面，单击右上角的使用镜像创建。

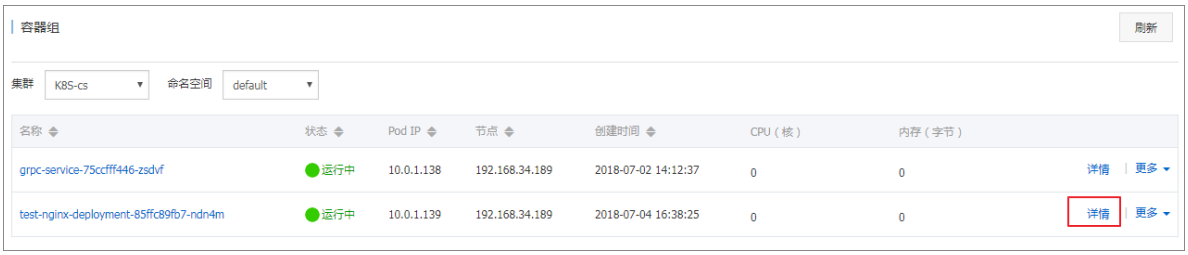
3. 在应用基本信息配置页面，设置应用名称、部署集群和命名空间，然后单击下一步。

4. 在应用配置页面，选择镜像，然后配置云存储类型的数据卷，支持云盘/NAS/OSS 三种类型。本例中使用准备好的云盘存储卷声明，最后单击下一步。

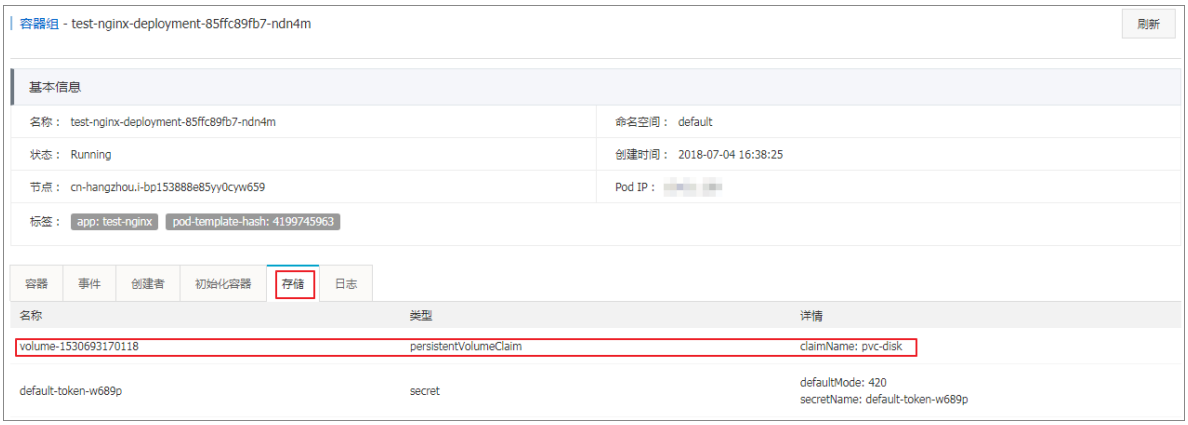


5. 配置test-nginx应用的服务，最后单击创建。

6. 应用创建完毕后，单击左侧导航栏中的应用 > 容器组，找到该应用所属容器组，单击详情。



7. 在容器组详情页面，单击存储，您可看到该容器组正确绑定了pvc-disk。



1.11 日志管理

1.11.1 概述

阿里云容器服务 Kubernetes 集群提供给您多种方式进行应用的日志管理。

- 通过使用日志服务进行*Kubernetes*日志采集，您可以方便地使用日志服务采集应用日志，从而更好地利用阿里云日志服务提供给您的各种日志统计分析等功能。
- 通过阿里云容器服务提供的开源 *Log-pilot* 项目，利用 *log-pilot + elasticsearch + kibana* 搭建 *kubernetes* 日志解决方案，您可以方便地搭建自己的应用日志集群。

1.11.2 查看集群日志

背景信息

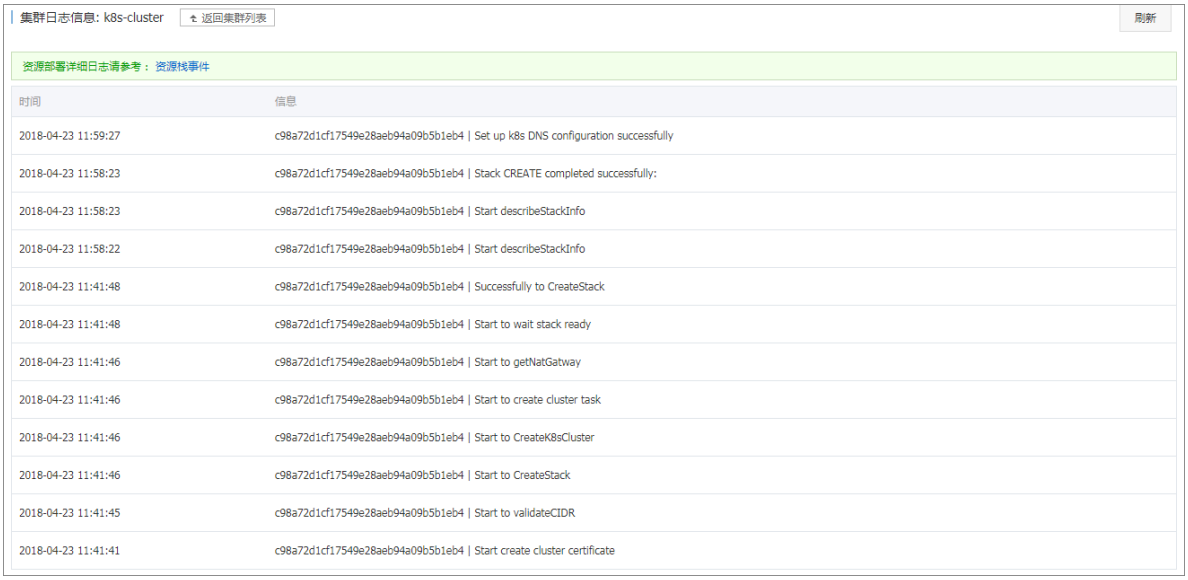
您可以通过容器服务的简单日志服务查看集群的操作日志。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。

3. 选择所需的集群并单击右侧的查看日志。

您可以查看该集群的操作信息。



1.11.3 使用日志服务进行Kubernetes日志采集

阿里云容器服务Kubernetes集群集成了日志服务（SLS），您可在创建集群时启用日志服务，快速采集Kubernetes 集群的容器日志，包括容器的标准输出以及容器内的文本文件。

新建 Kubernetes 集群

如果您尚未创建任何的 Kubernetes 集群，可以按照本节的步骤来进行操作：

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中集群，单击右上角创建Kubernetes集群。
3. 进入创建页面后，参见创建*Kubernetes*集群进行配置。
4. 拖动到页面底部，勾选日志服务配置项，表示使用在新建的 Kubernetes 集群中安装日志插件。
5. 当勾选了使用日志服务后，会出现创建 Project（日志服务管理日志的组织结构，具体可见项目）的提示，目前有两种方式可选：
   - 选择一个现有的 Project 来管理采集的日志。

- 自动创建一个新的 Project 来管理采集的日志，Project 会自动命名为 `k8s-log-{ClusterID}`，ClusterID 表示您新建的 Kubernetes 集群的唯一标识。

6. 配置完成后，单击右上角**创建集群**，在弹出的窗口中单击**确定**，完成创建。

   完成创建后，您可在集群列表页面看到创建的Kubernetes集群。

已创建 Kubernetes 集群，手动安装日志服务组件

如果您先前已创建了 Kubernetes 集群，可以根据本节的内容来进行所需的操作以使用日志服务：

- 未安装：手动安装日志服务组件。
- 已安装但版本较老：升级日志服务组件，若不升级则只能使用日志服务控制台或 CRD 进行采集配置。

检查日志服务组件版本

1. 配置本地kubeconfig以便于通过 kubectl 连接 Kubernetes 集群。

   如何配置参考通过 *kubectl* 连接 *Kubernetes* 集群 。

2. 快速判定是否需要进行升级或迁移操作，命令如下：

```
$ kubectl describe daemonsets -n kube-system logtail-ds | grep ALICLOUD_LOG_DOCKER_ENV_CONFIG
```

   - 如果输出结果为 `ALICLOUD_LOG_DOCKER_ENV_CONFIG: true`，表示您可正常使用，不需要进行升级或迁移。

- 其他情况则需要进行后续检查。

3. 执行以下检查命令确定是否是采用 helm 安装。

```
$ helm get alibaba-log-controller | grep CHART
CHART: alibaba-cloud-log-0.1.1
```

- 输出内容中的 0.1.1 表示日志服务组件的版本，请使用 0.1.1 及以上版本的组件，如果版本过低，请参照升级日志服务组件进行升级。如果您已采用 helm 安装且版本正确，可以跳过后续步骤。
- 如果无任何内容输出，表示未采用 helm 安装日志服务组件，但可能采用了 DaemonSet 的方式安装，请根据后续步骤进行检查。

4. DaemonSet 分为新旧两种方式：

```
$ kubectl get daemonsets -n kube-system logtail
```

- 如果无输出结果或内容是 `No resources found.`，则表明未安装日志服务组件，请参照操作手动安装日志服务组件。
- 如果有正确的输出结果表示使用旧 DaemonSet 方式进行了安装，需要进行升级，请参照升级日志服务组件操作。

手动安装日志服务组件

1. 配置本地kubeconfig以便于通过 kubectl 连接 Kubernetes 集群。

   如何配置参考通过 *kubectl* 连接 *Kubernetes* 集群 。

2. 替换参数后执行以下安装命令。

   将下述命令中的 ${your_k8s_cluster_id} 替换为您的 Kubernetes 集群 ID，并执行此命令。

```
wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/alicloud-k8s
-log-installer.sh -O alicloud-k8s-log-installer.sh; chmod 744 ./
alicloud-k8s-log-installer.sh; ./alicloud-k8s-log-installer.sh --
cluster-id ${your_k8s_cluster_id} --ali-uid ${your_ali_uid} --region
-id ${your_k8s_cluster_region_id}
```

   参数说明：

   - your_k8s_cluster_id：您的Kubernetes集群ID。
   - your_ali_uid：您的阿里云账号ID，可在个人资料查到。。
   - your_k8s_cluster_region_id：您的Kubernetes集群所在的Region，可在地域和可用区中查看到，如在杭州，则为cn-hangzhou

安装示例

```
[root@iZbp******biaZ ~]# wget https://acs-logging.oss-cn-hangzhou
.aliyuncs.com/alicloud-k8s-log-installer.sh -O alicloud-k8s-log-
installer.sh; chmod 744 ./alicloud-k8s-log-installer.sh; ./alicloud-
k8s-log-installer.sh --cluster-id c77a*****************0106 --ali-uid
19*********19 --region-id cn-hangzhou
--2018-09-28 15:25:33--  https://acs-logging.oss-cn-hangzhou.aliyuncs.
com/alicloud-k8s-log-installer.sh
Resolving acs-logging.oss-cn-hangzhou.aliyuncs.com... 118.31.219.217,
118.31.219.206
Connecting to acs-logging.oss-cn-hangzhou.aliyuncs.com|118.31.219.217
|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2273 (2.2K) [text/x-sh]
Saving to: ‘alicloud-k8s-log-installer.sh’

alicloud-k8s-log-installer.sh                       100
%[=================================================================================
   2.22K  --.-KB/s    in 0s

2018-09-28 15:25:33 (13.5 MB/s) - ‘alicloud-k8s-log-installer.sh’ saved
 [2273/2273]

--2018-09-28 15:25:33--  http://logtail-release-cn-hangzhou.oss-cn-
hangzhou.aliyuncs.com/kubernetes/alibaba-cloud-log.tgz
Resolving logtail-release-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com...
118.31.219.49
Connecting to logtail-release-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com
|118.31.219.49|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2754 (2.7K) [application/x-gzip]
Saving to: ‘alibaba-cloud-log.tgz’

alibaba-cloud-log.tgz                               100
%[=================================================================================
   2.69K  --.-KB/s    in 0s

2018-09-28 15:25:34 (79.6 MB/s) - ‘alibaba-cloud-log.tgz’ saved [2754/
2754]

[INFO] your k8s is using project : k8s-log-c77a92ec5a3ce4e64a1b
f13bde1820106
NAME:   alibaba-log-controller
LAST DEPLOYED: Fri Sep 28 15:25:34 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/CustomResourceDefinition
NAME                                AGE
aliyunlogconfigs.log.alibabacloud.com  0s

==> v1beta1/ClusterRole
alibaba-log-controller  0s

==> v1beta1/ClusterRoleBinding
NAME                   AGE
alibaba-log-controller  0s

==> v1beta1/DaemonSet
```

```
NAME        DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR  AGE
logtail-ds  2        2        0      2           0          <none>
     0s

==> v1beta1/Deployment
NAME                    DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
alibaba-log-controller  1        1        1           0          0s

==> v1/Pod(related)
NAME                                    READY  STATUS
RESTARTS  AGE
logtail-ds-6v979                        0/1    ContainerCreating  0
       0s
logtail-ds-7ccqv                        0/1    ContainerCreating  0
       0s
alibaba-log-controller-84d8b6b8cf-nkrkx  0/1    ContainerCreating  0
       0s

==> v1/ServiceAccount
NAME                    SECRETS  AGE
alibaba-log-controller  1        0s


[SUCCESS] install helm package : alibaba-log-controller success.
```

升级日志服务组件

如果您已安装旧版本的日志服务组件（通过 helm 或 DaemonSet），可以按照以下操作进行升级或迁移。

说明：

配置本地kubeconfig以便于通过 kubectl 连接 Kubernetes 集群，如何配置参考通过 *kubectl* 连接 *Kubernetes* 集群。

Helm 升级（推荐）

1. 下载最新的日志服务组件 helm 包，命令如下：

```
wget http://logtail-release-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com
/kubernetes/alibaba-cloud-log.tgz -O alibaba-cloud-log.tgz
```

2. 使用 helm upgrade 进行升级，命令如下：

```
helm get values alibaba-log-controller --all > values.yaml && helm
upgrade alibaba-log-controller alibaba-cloud-log.tgz --recreate-pods
 -f values.yaml
```

DaemonSet 迁移

此升级步骤适用于检查日志服务组件版本时结果为使用旧方式 DaemonSet 的用户，该方式不支持在容器服务中进行日志服务配置，您可按如下操作进行升级迁移：

1. 按照新版本的方式安装，安装命令最后新增一个参数为您之前 Kubernetes 集群使用的日志服务 Project 名。

   例如 Project 名为 k8s-log-demo，集群 ID 为 c12ba2028cxxxxxxxxx6939f0b，则安装命令为：

   ```
   wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/alicloud-k8s-log-installer.sh -O alicloud-k8s-log-installer.sh; chmod 744 ./alicloud-k8s-log-installer.sh; ./alicloud-k8s-log-installer.sh --cluster-id c12ba2028cxxxxxxxxx6939f0b --ali-uid 19*********19 --region-id cn-hangzhou --log-project k8s-log-demo
   ```

2. 安装成功后，进入日志服务控制台。
3. 在日志服务控制台，将相应 Project 以及 Logstore 下的历史采集配置应用到新的机器组 `k8s-group-${your_k8s_cluster_id}`。
4. 一分钟后，将历史采集配置从历史的机器组中解绑。
5. 观察日志采集正常后，可以选择删除之前安装的 logtail daemonset。

> 说明：
> 升级期间会有部分日志重复；CRD 配置管理方式只对使用 CRD 创建的配置生效（由于历史配置使用非 CRD 方式创建，因此历史配置不支持 CRD 管理方式）。

创建应用时配置日志服务

在容器服务中，您可以在创建应用的同时配置日志服务对容器的日志进行采集，目前支持以控制台向导和 YAML 模板两种方式进行创建。

控制台向导创建

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击页面右上角的使用镜像创建。
3. 设置应用名称、部署集群、命名空间、副本数量和类型，单击下一步，进入容器配置页面。

4. 进入容器配置页面中，本例中选择nginx镜像，对容器采集进行配置。

   以下仅介绍日志服务相关的配置，其他的应用配置可参见使用镜像创建无状态Deployment应用。

5. 进行日志配置。单击+号创建新的采集配置，每个采集配置由 **Logstore** 名称和日志采集路径两项构成。

   - Logstore 名称：您可以使用它来指定所采集日志存储于哪个 Logstore，如果该 Logstore 不存在的话，我们会自动为您在集群关联的日志服务 Project 下创建相应的 Logstore。

     > **说明：**
     > 名称中不能包含下划线（_），可以使用 - 来代替。

   - 日志采集路径：您可以用它来指定希望采集的日志所在的路径，如使用`/usr/local/tomcat/logs/catalina.*.log` 来采集tomcat的文本日志。

     > **说明：**
     > 如果指定为 stdout 则表示采集容器的标准输出和标准错误输出。

     每一项采集配置都会被自动创建为对应 Logstore 的一个采集配置，默认采用极简模式（按行）进行采集，如果您需要更丰富的采集方式，可以前往日志服务控制台，进入相应的 Project（默认是 k8s-log 前缀）和 Logstore 对配置进行修改。

6. 自定义 Tag。单击+号创建新的自定义 Tag，每一个自定义 Tag 都是一个键值对，会拼接到所采集到的日志中，您可以使用它来为容器的日志数据进行标记，比如版本号。

7. 当完成所有配置后，可单击右上角的下一步进入后续流程，后续操作可见使用镜像创建无状态*Deployment*应用。

使用 **YAML** 模板创建

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，然后单击页面右上角的使用模板创建。
3. YAML 模板的语法同 Kubernetes 语法，但是为了给容器指定采集配置，需要使用 env 来为 container 增加采集配置和自定义 **Tag**，并根据采集配置，创建对应的 volumeMounts 和 volumns。以下是一个简单的 Pod 示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-demo
spec:
  containers:
  - name: my-demo-app
    image: 'registry.cn-hangzhou.aliyuncs.com/log-service/docker-log
-test:latest'
    env:
    ######### 配置 环境变量 ###########
    - name: aliyun_logs_log-stdout
      value: stdout
    - name: aliyun_logs_log-varlog
      value: /var/log/*.log
    - name: aliyun_logs_mytag1_tags
      value: tag1=v1
    ################################
    ######### 配置vulume mount ###########
    volumeMounts:
    - name: volumn-sls-mydemo
      mountPath: /var/log
  volumes:
  - name: volumn-sls-mydemo
    emptyDir: {}
  ################################
```

- 其中有三部分需要根据您的需求进行配置，一般按照顺序进行配置。
- 第一部分通过环境变量来创建您的采集配置和自定义 **Tag**，所有与配置相关的环境变量都采用 `aliyun_logs_` 作为前缀。
- 创建采集配置的规则如下：

```
- name: aliyun_logs_{Logstore 名称}
```

```
  value: {日志采集路径}
```

示例中创建了两个采集配置，其中 `aliyun_logs_log-stdout` 这个 env 表示创建一个 Logstore 名字为 log-stdout，日志采集路径为 stdout 的配置，从而将容器的标准输出采集到 log-stdout 这个 Logstore 中。

> **说明：**
> Logstore 名称中不能包含下划线（_），可以使用 - 来代替。

- 创建自定义 **Tag** 的规则如下：

```
- name: aliyun_logs_{任意不包含'_'的名称}_tags
  value: {Tag 名}={Tag 值}
```

配置 Tag 后，当采集到该容器的日志时，会自动附加对应的字段到日志服务。

- 如果您的采集配置中指定了非 stdout 的采集路径，需要在此部分创建相应的 volumnMounts。

  示例中采集配置添加了对*/var/log/*.log* 的采集，因此相应地添加了*/var/log*的 volumeMounts。

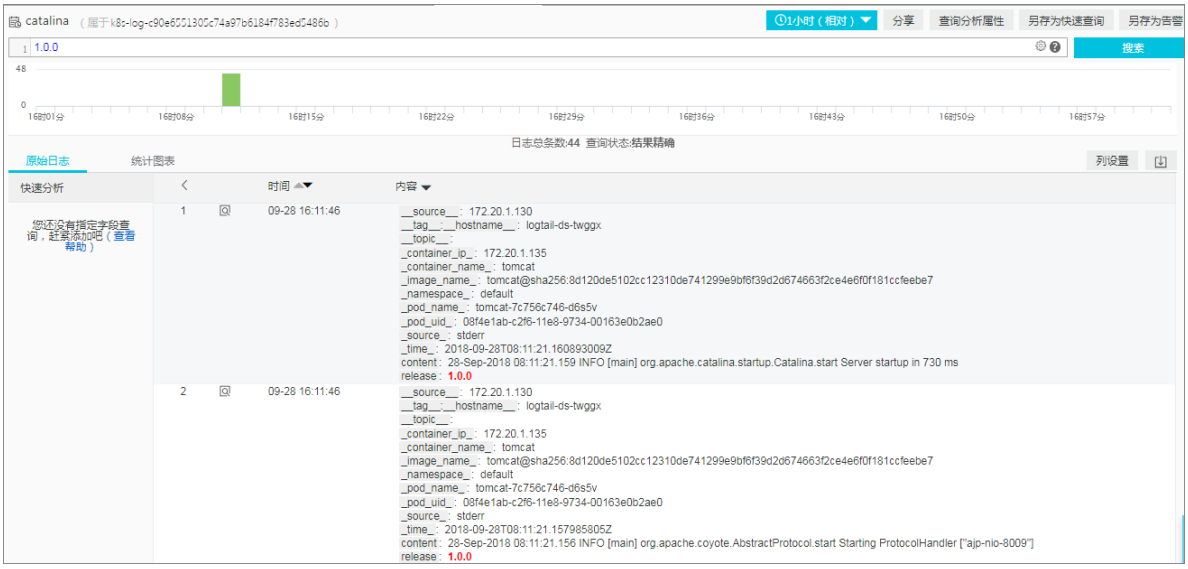
4. 当 YAML 编写完成后，单击**创建**，即可将相应的配置交由 Kubernetes 集群执行。

查看日志

本例中查看通过控制台向导创建的tomcat应用的日志。完成配置后，tomcat应用的日志已被采集并存储到日志服务中，您可以如下步骤来查看您的日志：

1. 安装成功后，进入日志服务控制台。
2. 在进入控制台后，选择 Kubernetes 集群对应的 Project（默认为 k8s-log-{Kubernetes 集群 ID}），进入 Logstore 列表页面。
3. 在列表中找到相应的 Logstore（采集配置中指定），单击**查询**。

4. 本例中，在日志查询页面，您可查看tomcat应用的标准输出日志和容器内文本日志，并可发现自定义tag附加到日志字段中。



更多信息

1. 默认情况下，我们会使用极简模式来采集您的数据（按行采集、不解析），如果您需要更复杂的配置，可以参考以下日志服务文档并前往日志服务控制台进行配置修改：
   - 容器文本日志
   - 容器标准输出
2. 目前日志服务使用了插件系统来采集容器的标准输出日志，您可以在此基础上配置一系列的处理插件对采集得到的日志进行进一步地处理（比如过滤、提取字段等），更多可参考处理采集数据。
3. 除了通过控制台配置采集以外，您还可以直接通过 CRD 配置来对 Kubernetes 集群进行日志采集，具体可参考*Kubernetes-CRD*配置日志采集。
4. 对于异常的排查，可以参考排查日志采集异常。

1.11.4 利用 log-pilot + elasticsearch + kibana 搭建 kubernetes 日志解决方案

开发者在面对 kubernetes 分布式集群下的日志需求时，常常会感到头疼，既有容器自身特性的原因，也有现有日志采集工具的桎梏，主要包括：

- 容器本身特性：

  - 采集目标多：容器本身的特性导致采集目标多，需要采集容器内日志、容器 stdout。对于容器内部的文件日志采集，现在并没有一个很好的工具能够去动态发现采集。针对每种数据源都有对应的采集软件，但缺乏一站式的工具。
  - 弹性伸缩难：kubernetes 是分布式的集群，服务、环境的弹性伸缩对于日志采集带来了很大的困难，无法像传统虚拟机环境下那样，事先配置好日志的采集路径等信息，采集的动态性以及数据完整性是非常大的挑战。
- 现有日志工具的一些缺陷：
  - 缺乏动态配置的能力。目前的采集工具都需要事先手动配置好日志采集方式和路径等信息，因为它无法能够自动感知到容器的生命周期变化或者动态漂移，所以它无法动态地去配置。
  - 日志采集重复或丢失的问题。因为现在的一些采集工具基本上是通过 tail 的方式来进行日志采集的，那么这里就可能存在两个方面的问题：一个是可能导致日志丢失，比如采集工具在重启的过程中，而应用依然在写日志，那么就有可能导致这个窗口期的日志丢失；而对于这种情况一般保守的做法就是，默认往前多采集 1M 日志或 2M 的日志，那么这就又会可能引起日志采集重复的问题。
  - 未明确标记日志源。因为一个应用可能有很多个容器，输出的应用日志也是一样的，那么当我们将所有应用日志收集到统一日志存储后端时，在搜索日志的时候，我们就无法明确这条日志具体是哪一个节点上的哪一个应用容器产生的。

本文档将介绍一种 Docker 日志收集工具 log-pilot，结合 Elasticsearch 和 kibana 等工具，形成一套适用于 kubernetes 环境下的一站式日志解决方案。

log-pilot 介绍

log-Pilot 是一个智能容器日志采集工具，它不仅能够高效便捷地将容器日志采集输出到多种存储日志后端，同时还能够动态地发现和采集容器内部的日志文件。

针对前面提出的日志采集难题，log-pilot 通过声明式配置实现强大的容器事件管理，可同时获取容器标准输出和内部文件日志，解决了动态伸缩问题，此外，log-pilot 具有自动发现机制，CheckPoint 及句柄保持的机制，自动日志数据打标，有效应对动态配置、日志重复和丢失以及日志源标记等问题。

目前 log-pilot 在 Github 完全开源，项目地址是 *https://github.com/AliyunContainerService/log-pilot* 。您可以深入了解更多实现原理。

针对容器日志的声明式配置

Log-Pilot 支持容器事件管理，它能够动态地监听容器的事件变化，然后依据容器的标签来进行解析，生成日志采集配置文件，然后交由采集插件来进行日志采集。

在 kubernetes 下，Log-Pilot 可以依据环境变量 `aliyun_logs_$name = $path` 动态地生成日志采集配置文件，其中包含两个变量：

- $name 是我们自定义的一个字符串，它在不同的场景下指代不同的含义，在本场景中，将日志采集到 ElasticSearch 的时候，这个 $name 表示的是 Index。
- 另一个是 $path，支持两种输入形式，stdout 和容器内部日志文件的路径，对应日志标准输出和容器内的日志文件。
  - 第一种约定关键字 stdout 表示的是采集容器的标准输出日志，如本例中我们要采集 tomcat 容器日志，那么我们通过配置标签 `aliyun.logs.catalina=stdout` 来采集 tomcat 标准输出日志。
  - 第二种是容器内部日志文件的路径，也支持通配符的方式，通过配置环境变量 `aliyun_logs_access=/usr/local/tomcat/logs/*.log`来采集 tomcat 容器内部的日志。当然如果你不想使用 aliyun 这个关键字，Log-Pilot 也提供了环境变量 PILOT_LOG_PREFIX 可以指定自己的声明式日志配置前缀，比如 `PILOT_LOG_PREFIX: "aliyun,custom"`。

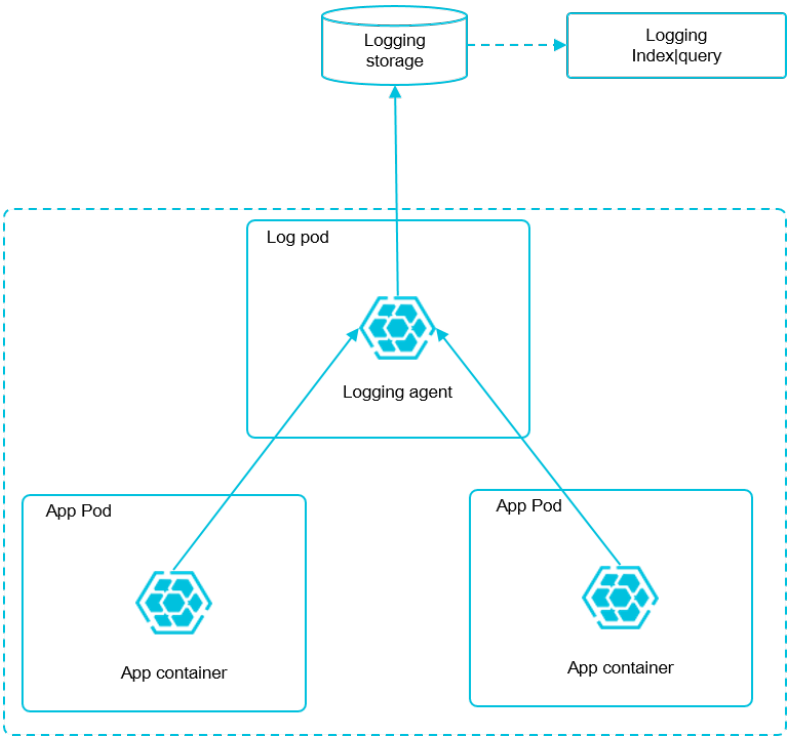
此外，Log-Pilot 还支持多种日志解析格式，通过 `aliyun_logs_$name_format=<format>` 标签就可以告诉 Log-Pilot 在采集日志的时候，同时以什么样的格式来解析日志记录，支持的格式包括：none、json、csv、nginx、apache2 和 regxp。

Log-Pilot 同时支持自定义 tag，我们可以在环境变量里配置 `aliyun_logs_$name_tags="K1=V1,K2=V2"`，那么在采集日志的时候也会将 K1=V1 和 K2=V2 采集到容器的日志输出中。自定义 tag 可帮助您给日志产生的环境打上 tag，方便进行日志统计、日志路由和日志过滤。

日志采集模式

本文档采用 node 方式进行部署，通过在每台机器上部署一个 log-pilot 实例，收集机器上所有 Docker 应用日志。

该方案跟在每个 Pod 中都部署一个 logging 容器的模式相比，最明显的优势就是占用资源较少，在集群规模比较大的情况下表现出的优势越明显，这也是社区推荐的一种模式。



前提条件

您已经开通容器服务，并创建了一个 kubernetes 集群。本示例中，创建的 Kubernetes 集群位于华东 1 地域。

步骤1 部署 elasticsearch

1. 连接到您的 Kubernetes 集群。具体操作参见通过*SSH*访问*Kubernetes*集群 或 通过 *kubectl* 连接 *Kubernetes* 集群。
2. 首先部署 elasticsearch 相关服务，该编排模板包含一个 elasticsearch-api 的服务、elasticsearch-discovery 的服务和 elasticsearch 的状态集，这些对象都会部署在 kube-system 命名空间下。

```
kubectl apply -f https://acs-logging.oss-cn-hangzhou.aliyuncs.com/elasticsearch.yml
```

3. 部署成功后，kube-system 命名空间下会出现相关对象，执行以下命令查看运行情况。

```
$ kubectl get svc,StatefulSet -n=kube-system
 NAME                          TYPE           CLUSTER-IP
EXTERNAL-IP     PORT(S)                       AGE
 svc/elasticsearch-api         ClusterIP      172.21.5.134    <none>
         9200/TCP                      22h
 svc/elasticsearch-discovery   ClusterIP      172.21.13.91    <none>
         9300/TCP                      22h
 ...
 NAME                          DESIRED   CURRENT   AGE
```

```
 statefulsets/elasticsearch   3      3      22h
```

步骤2 部署 log-pilot 和 kibana 服务

1. 部署 log-pilot 日志采集工具，如下所示：

```
kubectl apply -f https://acs-logging.oss-cn-hangzhou.aliyuncs.com/
log-pilot.yml
```

2. 部署 kibana 服务，该编排示例包含一个 service 和一个 deployment。

```
kubectl apply -f https://acs-logging.oss-cn-hangzhou.aliyuncs.com/
kibana.yml
```

步骤3 部署测试应用 tomcat

在 elasticsearch + log-pilot + Kibana 这套日志工具部署完毕后，现在开始部署一个日志测试应用 tomcat，来测试日志是否能正常采集、索引和显示。

编排模板如下。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
  namespace: default
  labels:
    name: tomcat
spec:
  containers:
  - image: tomcat
    name: tomcat-test
    volumeMounts:
    - mountPath: /usr/local/tomcat/logs
      name: accesslogs
    env:
     - name: aliyun_logs_catalina
       value: "stdout"
 ##采集标准输出日志
     - name: aliyun_logs_access
       value: "/usr/local/tomcat/logs/catalina.*.log"
 ## 采集容器内日志文件
  volumes:
    - name: accesslogs
      emptyDir: {}
```

tomcat 镜像属于少数同时使用了 stdout 和文件日志的 Docker 镜像，适合本文档的演示。在上面的编排中，通过在 pod 中定义环境变量的方式，动态地生成日志采集配置文件，环境变量的具体说明如下：

- `aliyun_logs_catalina=stdout`表示要收集容器的 stdout 日志。

- `aliyun_logs_access=/usr/local/tomcat/logs/catalina.*.log` 表示要收集容器内 */usr/local/tomcat/logs/* 目录下所有名字匹配 *catalina.*.log* 的文件日志。

在本方案的 elasticsearch 场景下，环境变量中的 `$name` 表示 Index，本例中 `$name`即是 catalina 和 access 。

步骤 4 将 kibana 服务暴露到公网

上面部署的 kibana 服务的类型采用 NodePort，默认情况下，不能从公网进行访问，因此本文档配置一个 ingress 实现公网访问 kibana，来测试日志数据是否正常索引和显示。

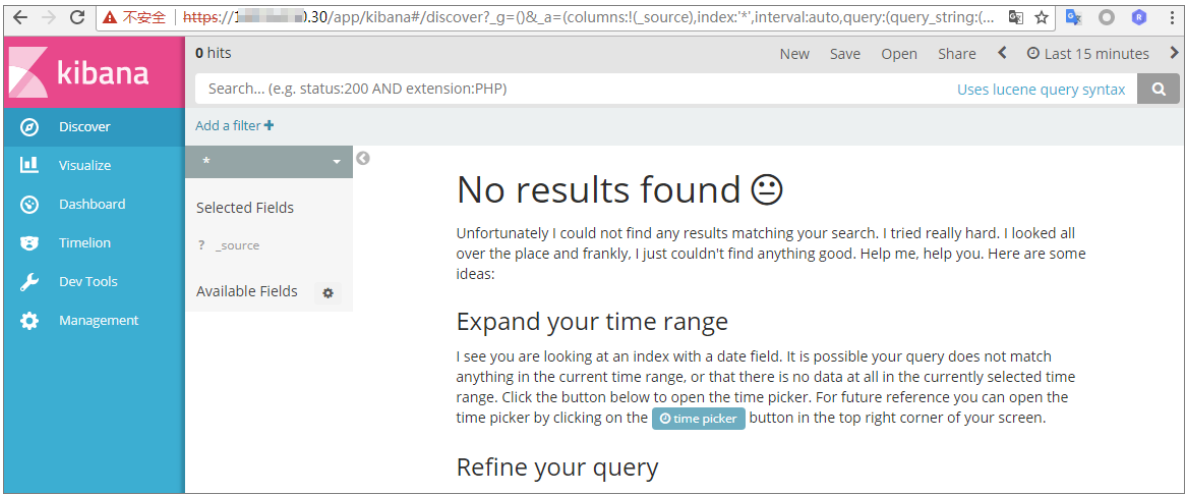
1. 通过配置 ingress 来实现公网下访问 kibana 。本示例选择简单的路由服务来实现，您可参考*Ingress* 支持 获取更多方法。该 ingress 的编排模板如下所示。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
   name: kibana-ingress
   namespace: kube-system                          #要与 kibana 服务处
于同一个 namespace
spec:
   rules:
   - http:
       paths:
       - path: /
         backend:
           serviceName: kibana                     #输入 kibana 服务的
名称
           servicePort: 80                         #输入 kibana 服务暴
露的端口
```

2. 创建成功后，执行以下命令，获取该 ingress 的访问地址。

```
$ kubectl get ingress -n=kube-system
 NAME         HOSTS     ADDRESS          PORTS     AGE
 shared-dns   *         120.55.150.30    80        5m
```

3. 在浏览器中访问该地址，如下所示。

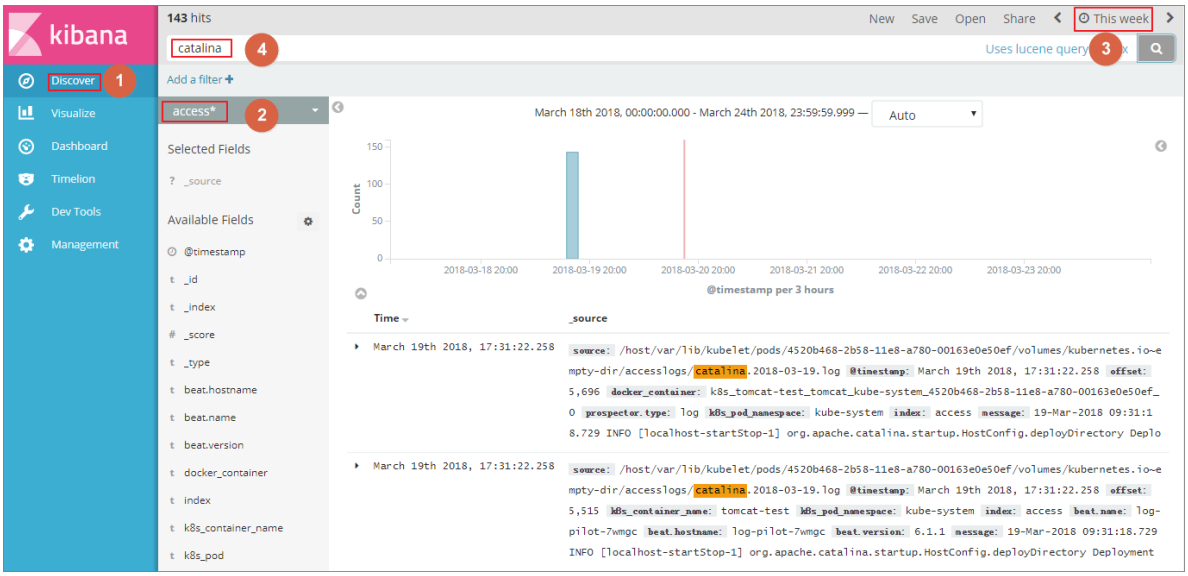


4. 单击左侧导航栏中的**management** ，然后单击**Index Patterns** > **Create Index Pattern**。具体的索引名称会在 `$name` 变量后缀一个时间字符串，您可以配合通配符 `*` 进行创建。本例中使用 `$name*` 来创建 Index Pattern。

   您也可以执行以下命令，进入 elasticsearch 对应的 pod，在 index 下列出 elasticsearch 的所有索引。

```
$ kubectl get pods -n=kube-system                    #找到 elasticsearch
 对应的 pod
   ...
$ kubectl exec -it  elasticsearch-1 bash              #进入
elasticsearch 的一个 pod
 ...
$ curl 'localhost:9200/_cat/indices?v'                   ## 列出所有索
引
 health status index                uuid                   pri rep
docs.count docs.deleted store.size pri.store.size
 green  open   .kibana             x06jj19PS4Cim6Ajo51PWg   1   1
       4            0     53.6kb         26.8kb
 green  open   access-2018.03.19   txd3tG-NR6-guqmMEKKzEw   5   1
     143            0    823.5kb        411.7kb
 green  open   catalina-2018.03.19 ZgtWd16FQ7qqJNNWXxFPcQ   5   1
     143            0    915.5kb        457.5kb
```

5. 索引创建完毕后，单击左侧导航栏中的**Discover**，然后选择前面创建的 Index，选择合适的时间段，在搜索栏输入相关字段，就可以查询相关的日志。



至此，在阿里云 Kubernetes 集群上，我们已经成功测试基于 log-pilot、elasticsearch 和 kibana 的日志解决方案，通过这个方案，我们能有效应对分布式 kubernetes 集群日志需求，可以帮助提升运维和运营效率，保障系统持续稳定运行。

1.11.5 为 Kubernetes 和日志服务配置 Log4JAppender

Log4j 是 Apache 的一个开放源代码项目。Log4j 由三个重要组件构成：日志信息的优先级、日志信息的输出目的地、日志信息的输出格式。通过配置 Log4jAppender，您可以控制日志信息输送的目的地是控制台、文件、GUI 组件、甚至是套接口服务器、NT 的事件记录器、UNIX Syslog 守护进程等。

本文介绍在不需要修改应用代码的前提下，通过配置一个 yaml 文件，将阿里云容器服务 Kubernetes 集群中产生的日志输出到阿里云日志服务。此外，通过在 Kubernetes 集群上部署一个示例 API 程序，来进行演示。

前提条件

- 您已经开通容器服务，并且创建了 Kubernetes 集群。

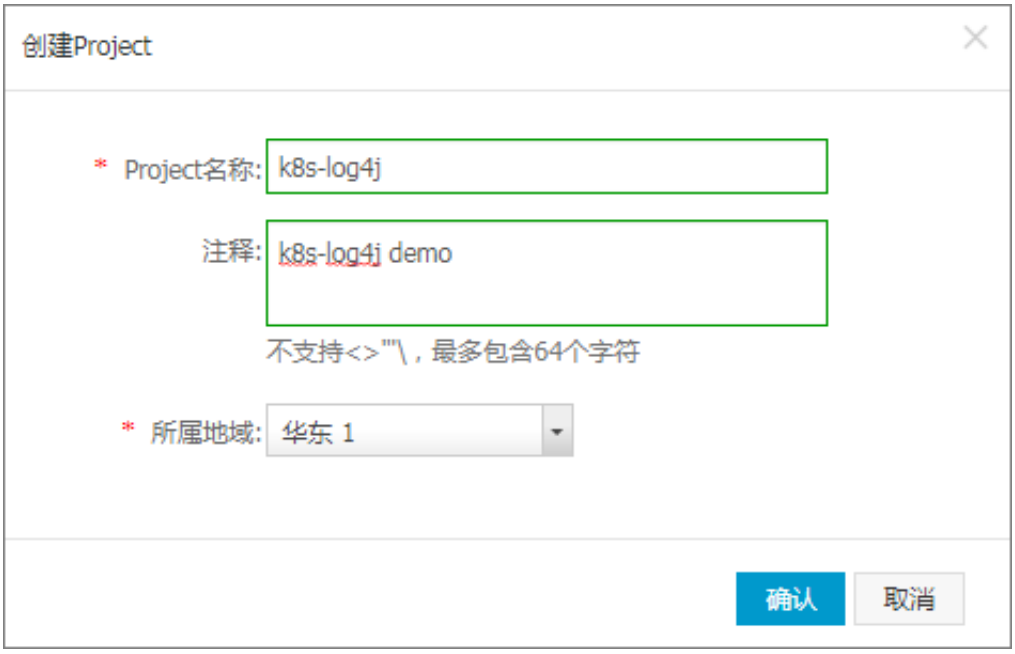
  本示例中，创建的 Kubernetes 集群位于华东 1 地域。
- 启用 AccessKey 或 RAM，确保有足够的访问权限。本例中使用 AccessKey。

步骤 1 在阿里云日志服务上配置 Log4jAppender

1. 登录 日志服务管理控制台。
2. 单击页面右上角的创建 **Project**，填写 Project 的基本信息并单击确认进行创建。

本示例创建一个名为 k8s-log4j，与 Kubernetes 集群位于同一地域（华东 1）的 Project。

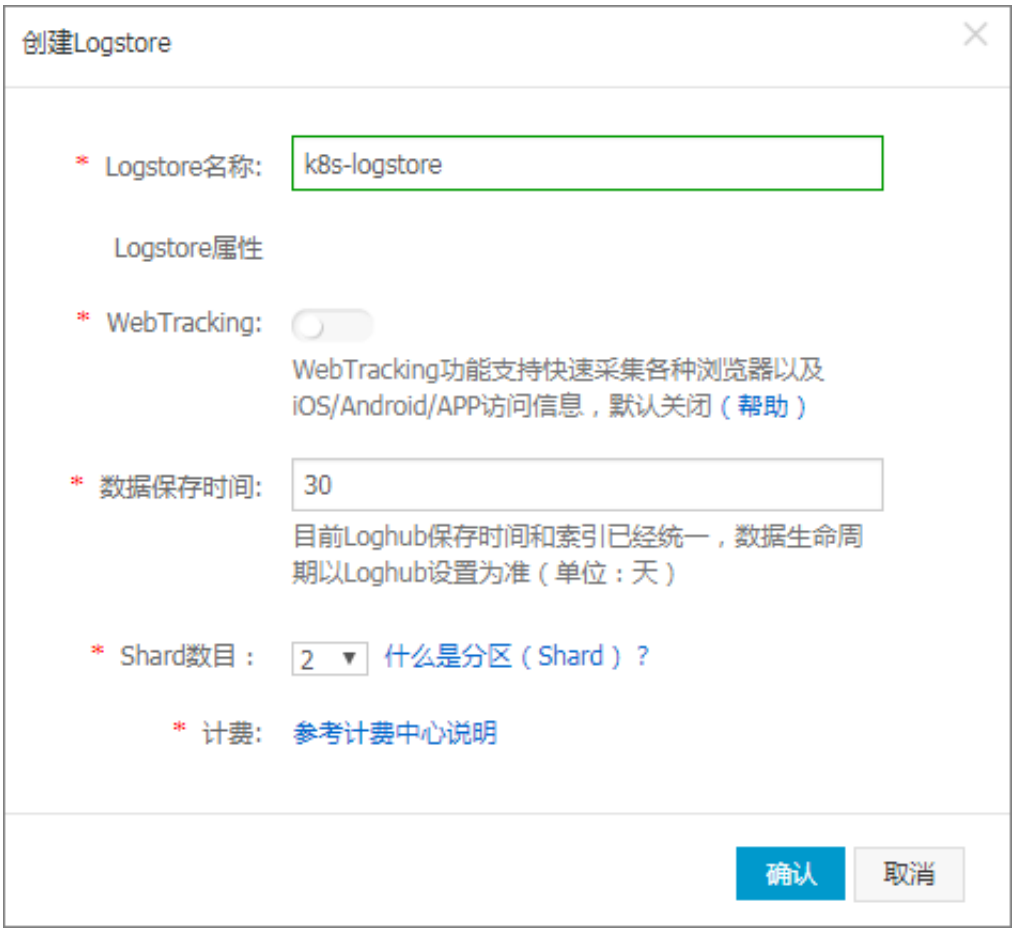
> **说明：**
> 在配置时，一般会使用与 Kubernetes 集群位于同一地域的日志服务 Project。因为当 Kubernetes 集群和日志服务 Project 位于同一地域时，日志数据会通过内网进行传输，从而避免了因地域不一致而导致的数据传输外网带宽费用和耗时，从而实现实时采集、快速检索的最佳实践。



3. 创建完成后，k8s-log4j 出现在 project 列表下，单击该 project 名称，进入 project 详情页面。
4. 默认进入日志库页面，单击右上角的创建。

5. 填写日志库配置信息并单击确认。

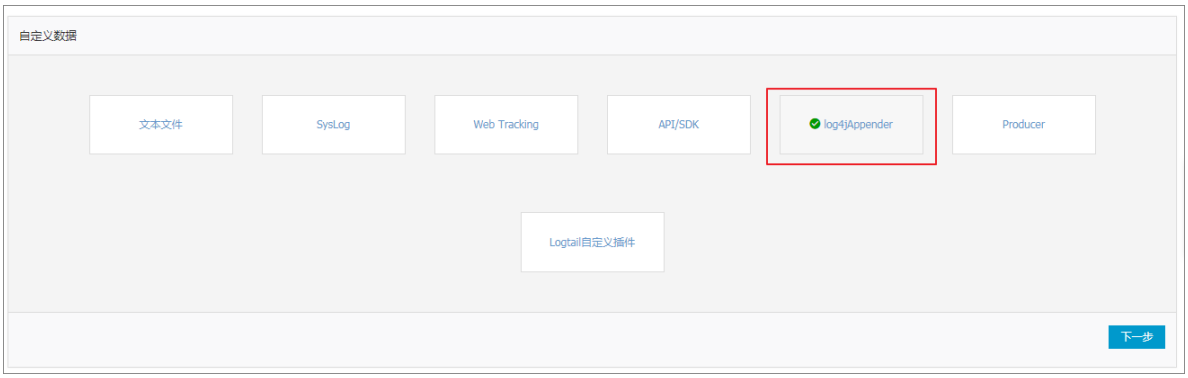
   本示例创建名为 k8s-logstore 的日志库。



6. 创建完毕后，页面会提示您创建数据接入向导。

7. 选择自定义数据下的log4jAppender，根据页面引导进行配置。

   本示例使用了默认配置，您可根据日志数据的具体使用场景，进行相应的配置。



步骤 2 在 Kubernetes 集群中配置 log4j

本示例使用 *demo-deployment* 和 *demo-Service* 示例 yaml 文件进行演示。

1. 连接到您的 Kubernetes 集群。

   具体操作参见*SSH访问Kubernetes集群* 或 *通过 kubectl 连接 Kubernetes 集群*。

2. 获取*demo-deployment.yaml*文件并配置环境变量 `JAVA_OPTS` 设置 Kubernetes 集群日志的采集。

   demo-deployment.yaml 文件的示例编排如下。

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: log4j-appender-demo-spring-boot
  labels:
    app: log4j-appender
spec:
  replicas: 1
  selector:
    matchLabels:
      app: log4j-appender
  template:
    metadata:
    labels:
      app: log4j-appender
  spec:
    containers:
    - name: log4j-appender-demo-spring-boot
      image: registry.cn-hangzhou.aliyuncs.com/jaegertracing/log4j-
appender-demo-spring-boot:0.0.2
      env:
      - name: JAVA_OPTS                    ##注意
        value: "-Dproject={your_project} -Dlogstore={your_logstore
} -Dendpoint={your_endpoint} -Daccess_key_id={your_access_key_id} -
Daccess_key={your_access_key_secret}"
      ports:
      - containerPort: 8080
```

   其中：

- `-Dproject`：您所使用的阿里云日志服务 Project 的名称。本示例中为 k8s-log4j。
- `-Dlogstore`：您所使用的阿里云日志服务 Logstore 的名称。本示例中为 k8s-logstore。
- `-Dendpoint`：日志服务的服务入口，用户需要根据日志 Project 所属的地域，配置自己的服务入口，参见服务入口进行查询。本示例中为 cn-hangzhou.log.aliyuncs.com。
- `-Daccess_key_id`：您的 AccessKey ID。
- `-Daccess_key`：您的 AccessKey Secret。

3. 在命令行中执行以下命令，创建 deployment。

```
kubectl create -f demo-deployment.yaml
```

4. 获取*demo-Service.yaml*文件，并运行以下命令创建 service。

   您不需要修改*demo-Service.yaml*中的配置。

```
kubectl create -f demo-service.yaml
```

步骤 3 测试生成 Kubernetes 集群日志

您可以使用 `kubectl get` 命令查看资源对象部署状况，等待 deployment 和 service 部署成功后，执行 `kubectl get svc` 查看 service 的外部访问 IP，即 EXTERNAL-IP。

```
$ kubectl get svc
NAME                          TYPE           CLUSTER-IP     EXTERNAL-IP
  PORT(S)          AGE
log4j-appender-demo-spring-boot-svc   LoadBalancer   172.21.XX.XX
120.55.XXX.XXX   8080:30398/TCP   1h
```

在本示例中，通过运行login命令来测试生成 Kubernetes 集群日志。其中 `K8S_SERVICE_IP` 即为 `EXTERNAL-IP`。

说明：

您可以在 *GitHub log4j-appender-demo* 中查看完整的 API 集合。

```
curl http://${K8S_SERVICE_IP}:8080/login?name=bruce
```

步骤 4 在阿里云日志服务中查看日志

登录 日志服务管理控制台。

进入对应的 Project 的详情页面，选择对应的日志库 k8s-logstore，并单击右侧的**查询分析 - 查询**，查看 Kubernetes 集群输出的日志，如下所示。

日志的输出内容对应上面的命令。本示例演示了将示例应用产生的日志输出到阿里云日志服务的操作。通过这些操作，您可以在阿里云上配置 Log4JAppender，并通过阿里云日志服务，实现日志实时搜集、数据过滤、检索等高级功能。

1.12 监控管理

1.12.1 部署Prometheus监控方案

Prometheus是一款面向云原生应用程序的开源监控工具，本文介绍如何基于阿里云容器Kubernetes版本部署Prometheus监控方案。

背景信息

对于监控系统而言，监控对象通常分为以下两类：

- 资源监控：节点、应用的资源使用情况，在容器Kubernetes中可理解为节点的资源利用率、集群的资源利用率、Pod的资源利用率等。
- 应用监控：应用内部指标的监控，例如实时统计应用的在线人数，并通过端口暴露来实现应用业务级别的监控与告警等。

在Kubernetes系统中，监控对象具体为：

- 系统组件：Kubernetes集群中内置的组件，包括apiserver、controller-manager、etcd等。
- 静态资源实体：节点的资源状态、内核事件等。
- 动态资源实体：Kubernetes中抽象工作负载的实体，例如Deployment、DaemonSet、Pod等。
- 自定义应用：应用内部需要定制化的监控数据以及监控指标。

对于系统组件和静态资源实体的监控方式，在配置文件中指明即可。

对于动态资源实体的监控，可以使用Prometheus监控部署方案。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已连接到Master节点，方便快速查看节点标签等信息，参见通过 *kubectl* 连接 *Kubernetes* 集群。

部署Prometheus监控方案

1. 执行以下命令，下载prometheus-operator代码。

```
git clone https://github.com/AliyunContainerService/prometheus-operator
```

2. 执行以下命令，部署Prometheus监控方案。

```
cd prometheus-operator/contrib/kube-prometheus
kubectl apply -f manifests
```

3. 执行以下命令，设置Prometheus访问。

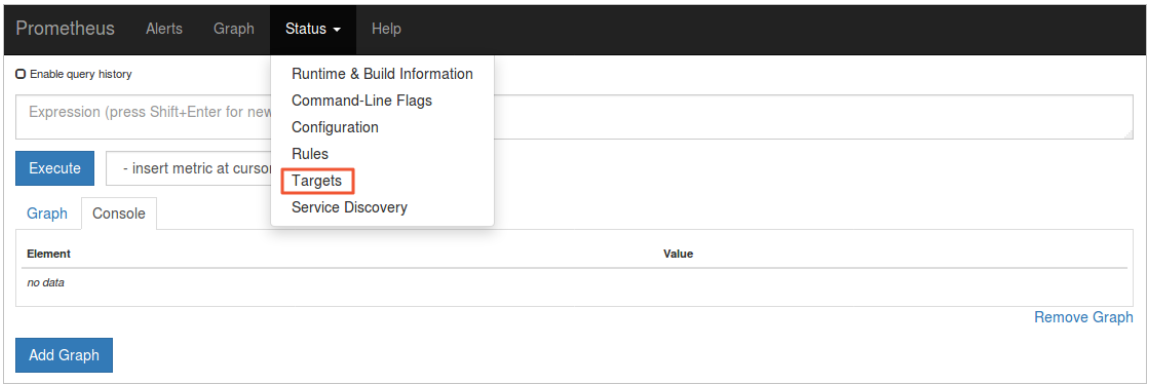
```
kubectl --namespace monitoring port-forward svc/prometheus-k8s 9090
```

4. 查看部署结果：

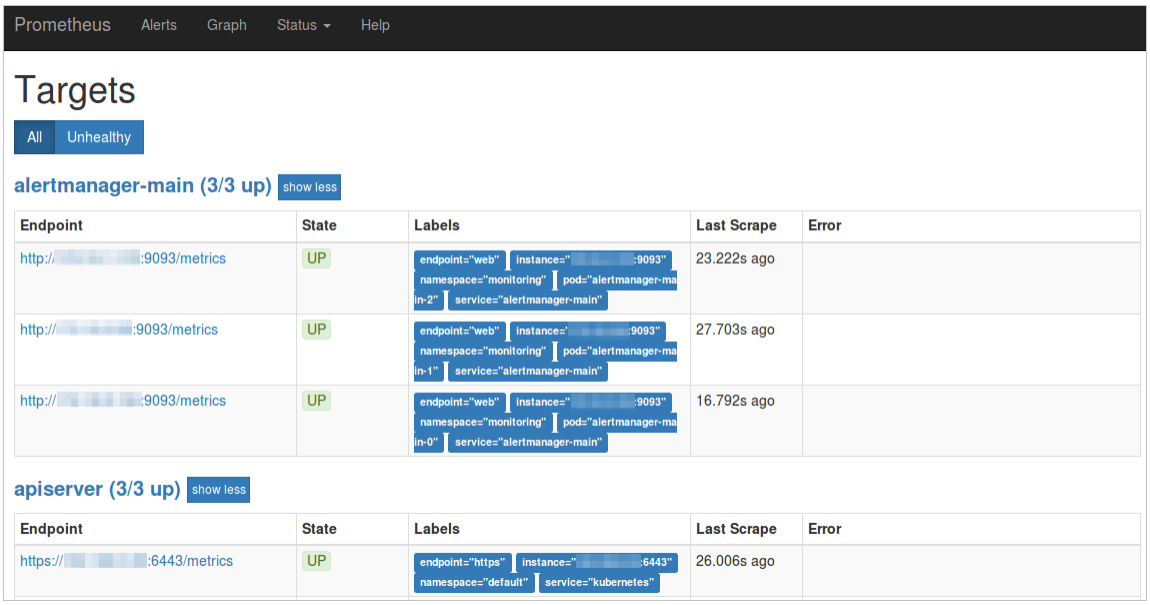
   a. 在浏览器中访问`localhost:9090`，即可查看Prometheus。

   > 说明：
   > 默认情况下，无法通过公网访问Prometheus，需要通过本地Proxy的方式查看。

   b. 选择菜单栏**Status**下的**Targets**，查看所有采集任务。



   如果所有任务的状态为**UP**，表明所有采集任务均已正常运行。



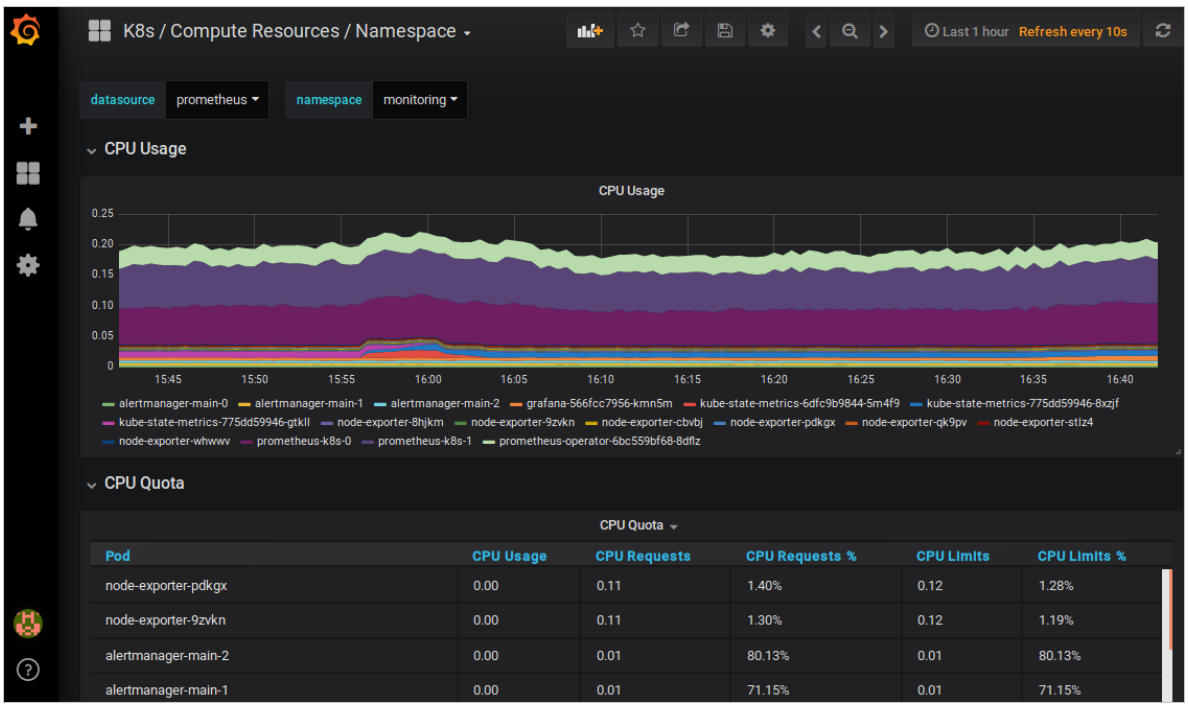
查看与展示数据聚合

1. 执行以下命令，访问Grafana：

```
kubectl --namespace monitoring port-forward svc/grafana 3000
```

2. 在浏览器中访问`localhost:3000`，选择相应的Dashboard，即可查看相应的聚合内容。

> 说明：
>
> 默认的用户名/密码是：admin/admin。

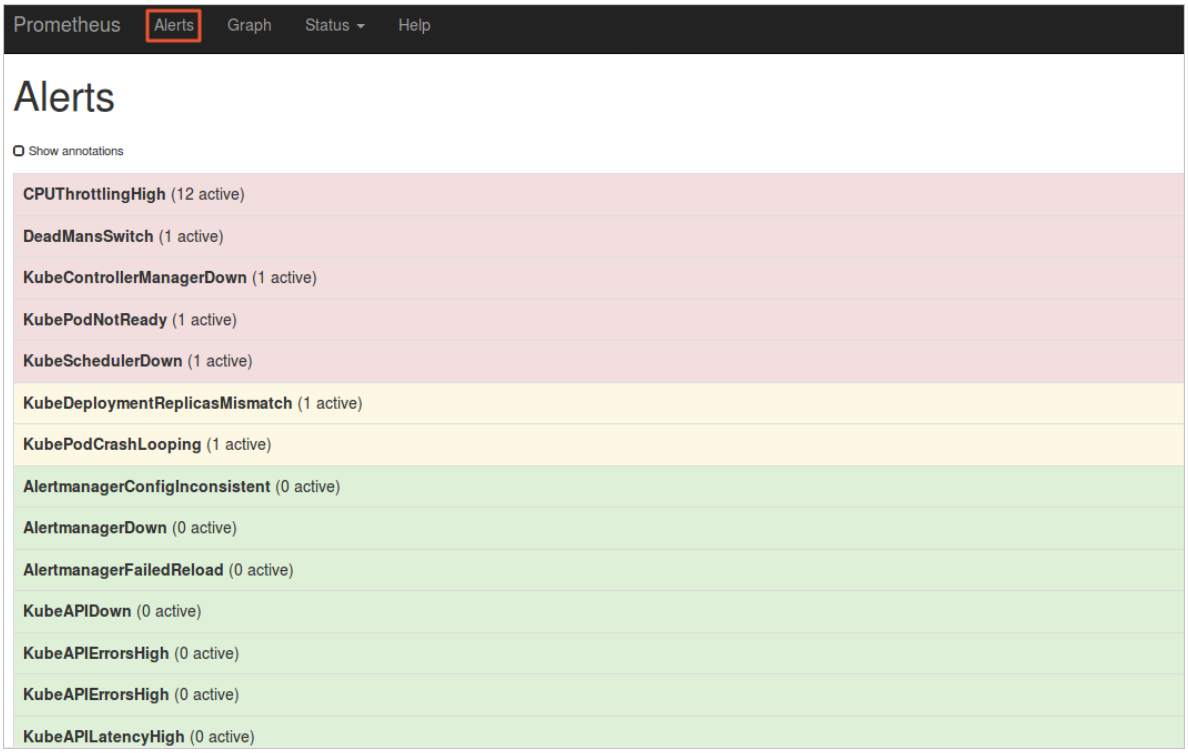


查看告警规则与设置告警压制

- 查看告警规则

  在浏览器中访问`localhost:9090`，选择菜单栏**Alerts**，即可查看当前的告警规则。

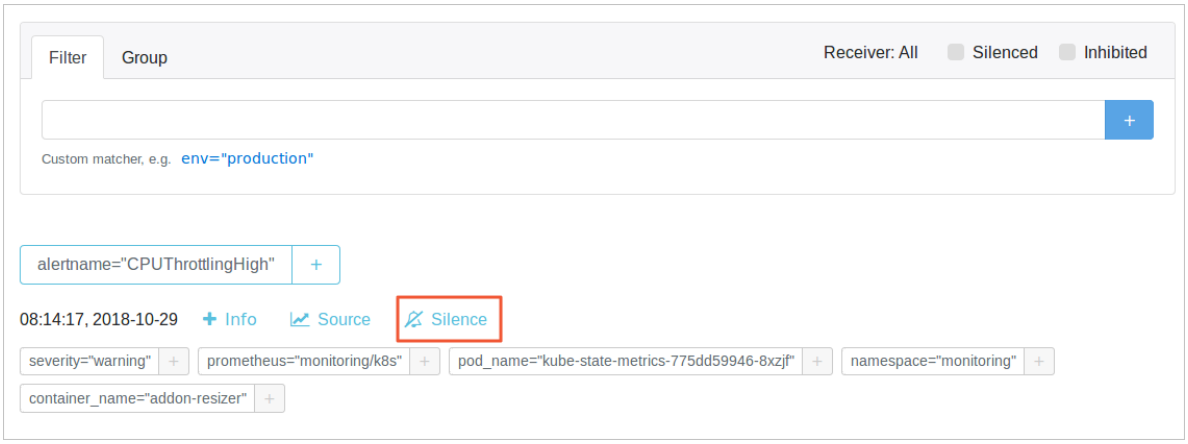
  - 红色：正在触发告警。
  - 绿色：正常状态。



- 设置告警压制

  执行以下命令，并在浏览器中访问`localhost:9093`，选择**Silenced**，设置告警压制。

```
kubectl --namespace monitoring port-forward svc/alertmanager-main 9093
```



1.12.2 通过资源分组进行监控与告警

阿里云容器服务与云监控产品进行集成，为您提供基于资源分组的监控服务。

前提条件

- 如果之前没有创建过集群，您需要创建Kubernetes集群。
- Kubernetes 版本需要在 1.8.4 及以上，若集群版本过低，您可以先对集群进行升级，然后通过升级监控服务的方式快速建立资源报警分组。

背景信息

在 IT 系统基础设施运维中，监控告警一直是保证可靠性和安全性的基础，有助于日常运维、系统监测、故障排除和调试。

在 Kubernetes 场景下，传统的容器监控方案通过静态配置化的监控 agent 或中心化的 server 进行资源监控和告警，会遇到很大的难题。例如，由于容器更多的是在资源池中调度，宿主机部署监控 agent 会造成缺乏必要信息来识别监控对象；容器的生命周期与传统应用相比而言会更加短暂，而由容器抽象的上层概念如 kubernetes 中的 ReplicaSet、Deployment 等则没有太好的办法从采集的数据中进行反向的抽象，造成单纯的容器监控数据无法有效地进行监控数据的聚合和告警，一旦应用的发布可能会导致原有的监控与报警规则无法生效。

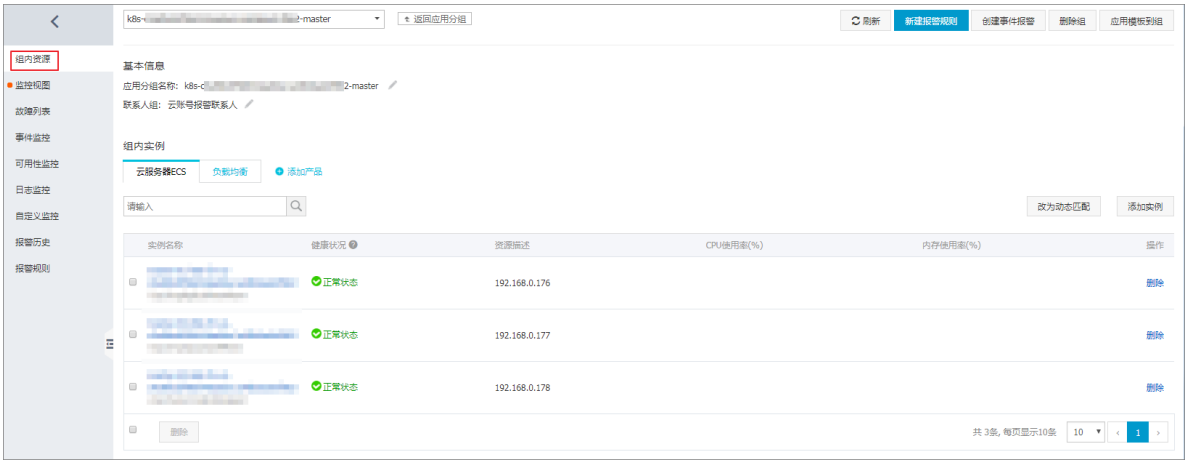
阿里云容器服务 Kubernetes 与云监控进行了深度集成，用应用分组来抽象逻辑概念，实现逻辑的概念和物理概念在监控数据、生命周期上面的统一。此外，阿里云云监控服务提供丰富的功能特性和自定义工具，帮助您快速实现 Kuberbetes 资源监控和告警的最佳实践。

操作步骤

1. 登录 云监控服务控制台。
2. 在左侧导航栏中单击应用分组，在应用分组中，可以看到包含集群 ID 信息的 kubernetes 资源分组。

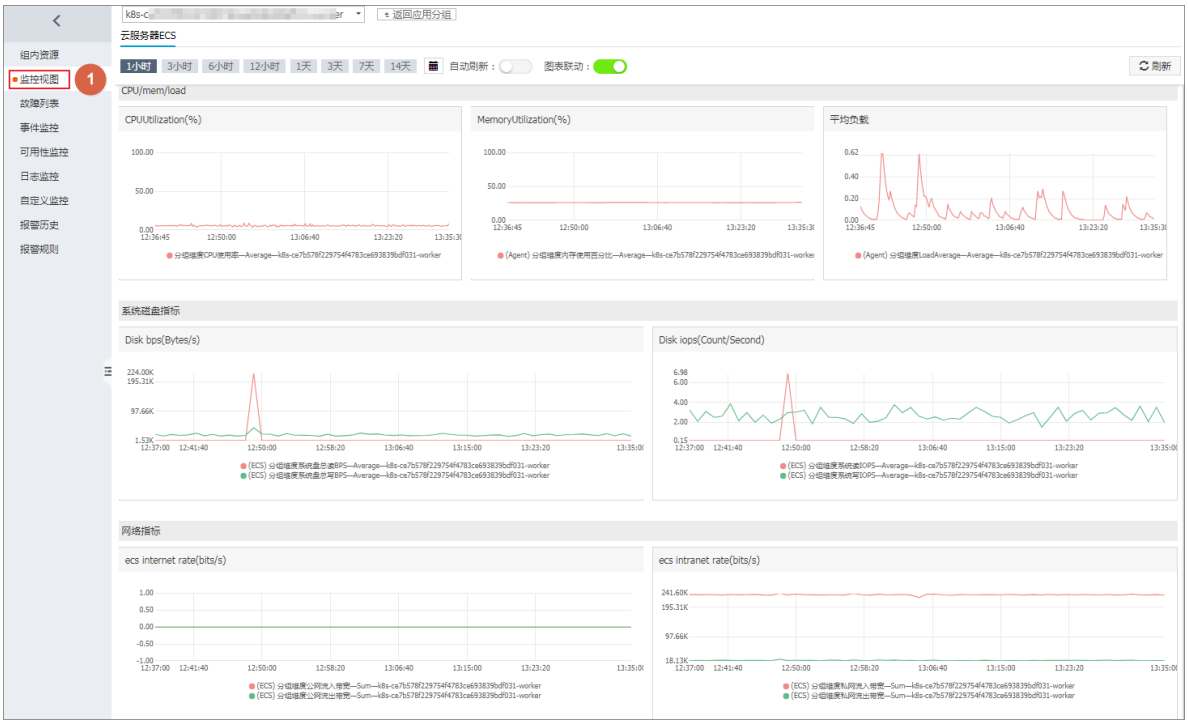
3. 单击分组名称，进入具体的分组页面，您可以查看组内各项资源的情况。以 kubernetes 的 master 分组为例，显示组内的资源，包括云服务器ECS和负载均衡。

Kubernetes 节点从职能上分为 Worker 和 Master 两种不同的节点。Master 节点上面通常会部署管控类型的应用，整体的资源要求以强鲁棒性为主；而 Worker 节点更多的承担实际的 Pod 调度，整体的资源以调度能力为主。当你创建资源报警分组时，容器服务会为你自动创建两个资源分组，一个是 Master 组，一个是 Worker 组。Master 组中包含了 Master 节点以及与其相关的负载均衡器；Worker 组包含了所有的工作节点。



4. 您可以查看分组内的其他云产品的详细信息，如负载均衡。

5. 在左侧导航栏中单击监控视图，您可以查看分组内各云产品的详细监控指标。



6. 在左侧导航栏中单击报警规则，该页面列出当前分组中已有的报警规则列表，默认会在 Master 分组中设置所有节点的核心组件的健康检查。

   1. 您可以单击新建报警规则，根据业务需求，来创建属于该分组的报警规则。

   2. 进入报警规则页面，您需要设置报警规则。

      - 您可选择报警关联对象。
      - 您可选择是否使用模板创建报警规则。若选择使用模板创建报警规则，您可以在选择模板下拉框中选择已有的报警模板；或者可以单击创建报警模板，创建新的自定义报警模板，参见 ，然后再进行选择。
      - 您可设置通知方式，如通过钉钉、邮件、短信的方式在第一时间获取到 Kubernetes 的集群状态。

3. 最后单击**确认**，本例中新建的报警规则会出现在规则列表下。

后续操作

在左侧导航栏中，您可以探索更多符合自己资源监控需求的功能特性，如故障列表、事件监控、可用性监控、日志监控等。

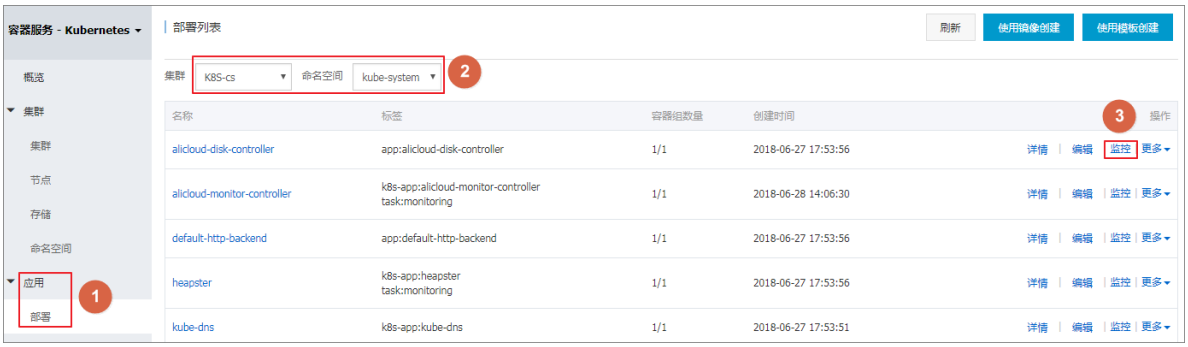
1.12.3 与云监控集成与使用

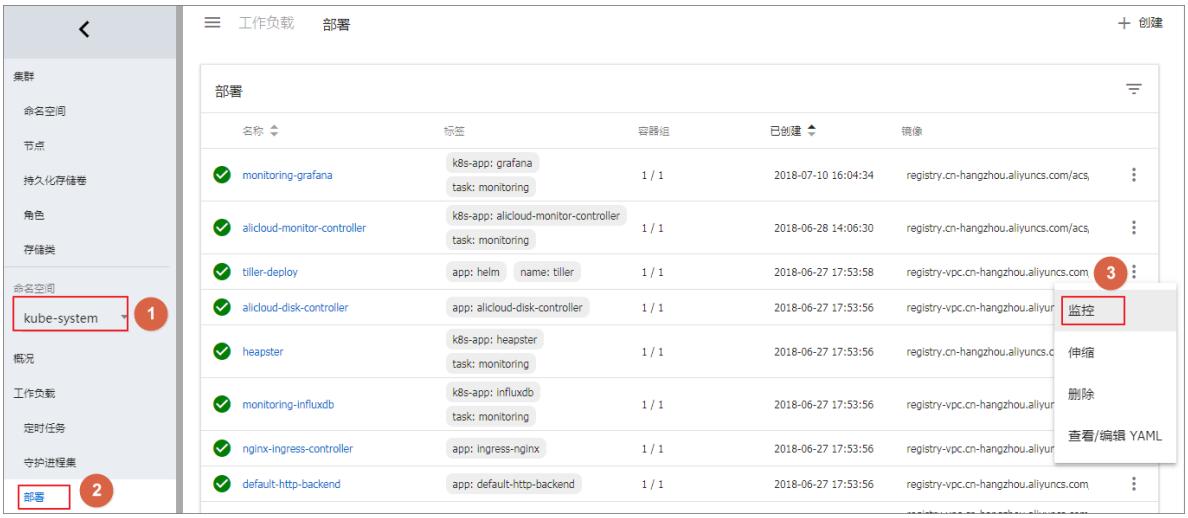
前提条件

请先检查`kube-system`命名空间下是否已经部署了`alicloud-monitor-controller`，若未部署，请进行旧版本集群升级。

使用方式

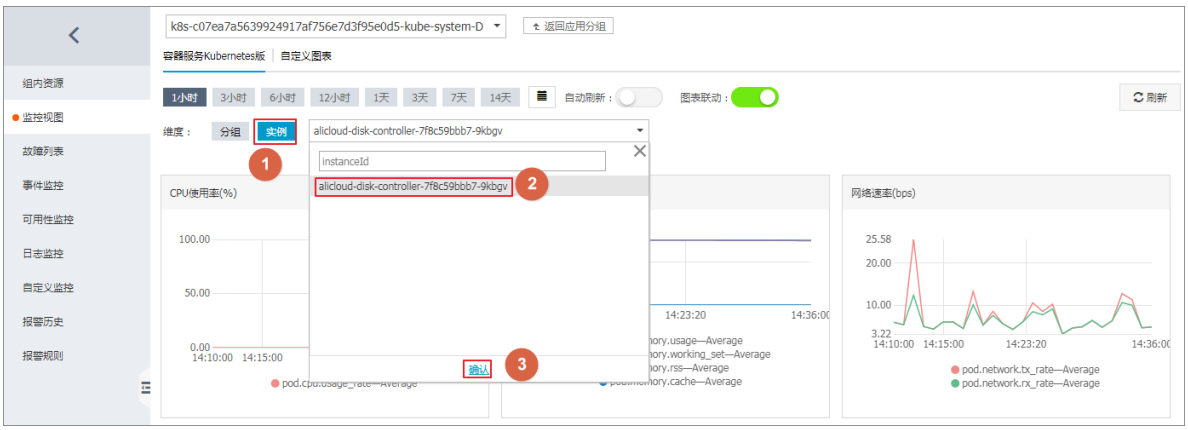
1. 登录 容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的部署，进入部署列表页面。
3. 选择所需的deployment，单击右侧的监控，或者在内置的Kubernetes Dashboard的部署页面中单击监控。



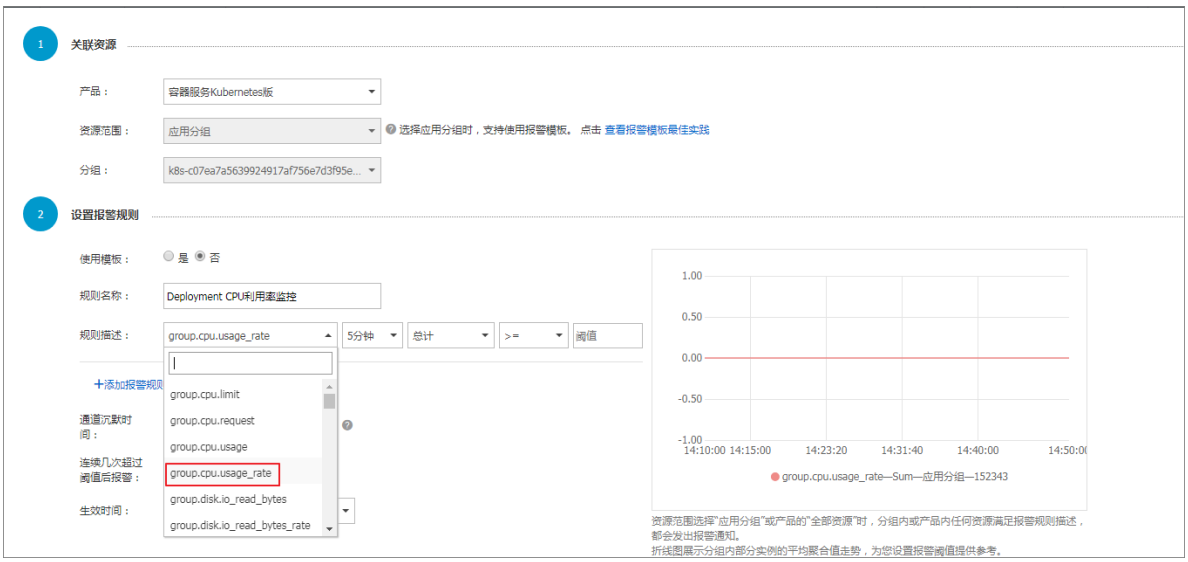


此时会跳转到云监控的相应的监控视图页面。

4. 应用分组支持分组和实例两个维度的监控。



5. 如需告警设置，分组级别的指标以**group**开头，实例级别的指标以**pod**开头。



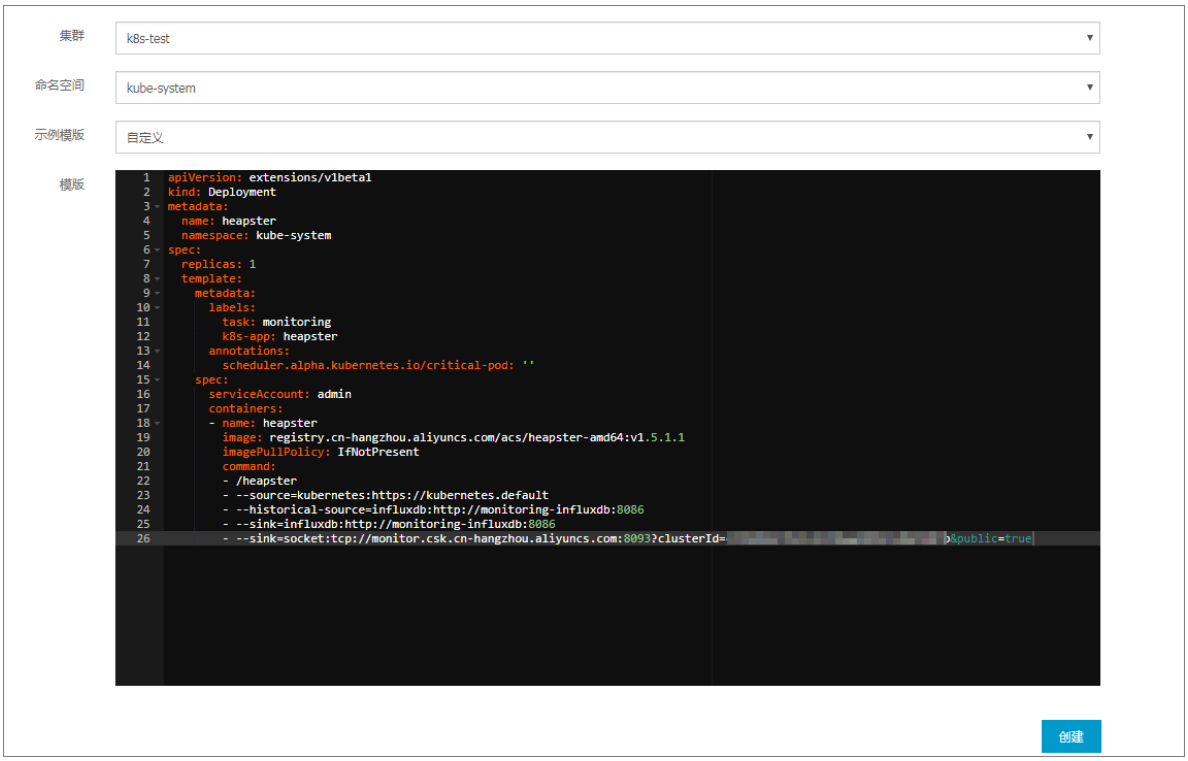
旧版本集群升级

1. 登录 容器服务管理控制台。

2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面，单击右上角的使用模板创建。

3. 选择所需的集群，Kube-system命名空间，使用以下的示例模板，然后单击创建。

> 说明：
> 根据自己的集群替换**REGION**与**CLUSTER_ID**，并重新部署Heapster的yaml编排。



heapster示例编排模板如下。若集群中已有旧版本的heapster，您也可登录到Kubernetes集群，执行`kubectl apply -f xxx.yaml`命令进行更新。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
```

```
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: heapster
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      serviceAccount: admin
      containers:
      - name: heapster
        image: registry.##REGION##.aliyuncs.com/acs/heapster-amd64:
v1.5.1.1
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes:https://kubernetes.default
        - --historical-source=influxdb:http://monitoring-influxdb:
8086
        - --sink=influxdb:http://monitoring-influxdb:8086
        - --sink=socket:tcp://monitor.csk.##REGION##.aliyuncs.com:
8093?clusterId=##CLUSTER_ID##&public=true
```

alicloud-monitor-controller 的示例编排如下，执行`kubectl create -f xxx.yaml`命令部署alicloud-monitor-controller。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: alicloud-monitor-controller
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: alicloud-monitor-controller
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      hostNetwork: true
      tolerations:
      - effect: NoSchedule
        operator: Exists
        key: node-role.kubernetes.io/master
      - effect: NoSchedule
        operator: Exists
        key: node.cloudprovider.kubernetes.io/uninitialized
      serviceAccount: admin
      containers:
      - name: alicloud-monitor-controller
        image: registry.##REGION##.aliyuncs.com/acs/alicloud-monitor
-controller:v1.0.0
        imagePullPolicy: IfNotPresent
        command:
        - /alicloud-monitor-controller
        - agent
        - --regionId=##REGION##
```

```
        - --clusterId=##CLUSTER_ID##
        - --logtostderr
        - --v=4
```

4. 更新完毕后，进入Kubernetes 控制台，在kube-system命名空间中，可看到这两个Deployment处于运行中，即升级完毕。

对于不清楚自己REGION信息的开发者，可以通过如下的方式快速查询，打开ECS控制台，选择自己集群所在的地域，页面地址URL中最后一段即是REGION。

1.12.4 使用 Grafana 展示监控数据

前提条件

- 您已经成功部署一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 本示例使用的 Grafana 的镜像地址是 `registry.cn-hangzhou.aliyuncs.com/acs/grafana:5.0.4`，内置了相关监控模板。

背景信息

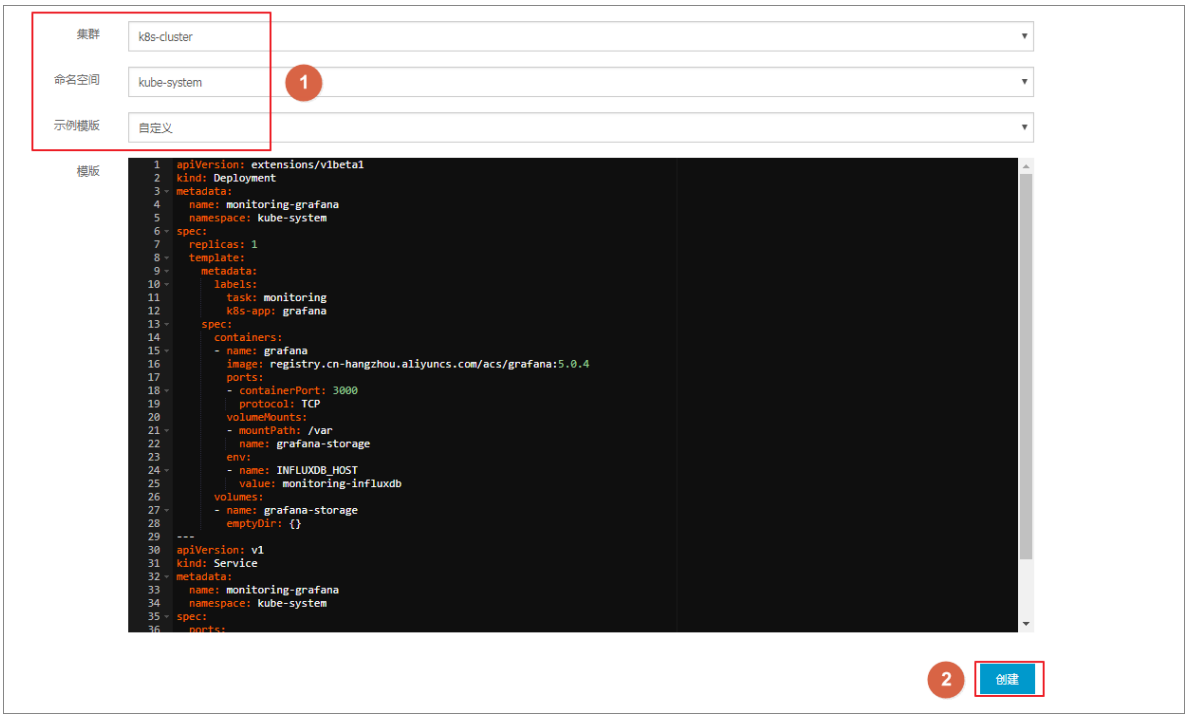
在 kubernetes 的监控方案中，Heapster+Influxdb+Grafana 的组合相比 prometheus 等开源方案而言更为简单直接。而且 Heapster 在 kubernetes 中承担的责任远不止监控数据的采集，控制台的监控接口、HPA的 POD 弹性伸缩等都依赖于 Heapster 的功能。因此 Heapster 成为 kubernetes 中一个必不可少的组件，在阿里云的 Kubernetes 集群中已经内置了 Heapster+Influxdb 的组合，如果需要将监控的数据进行展示，只需要配置一个可用的 Grafana 与相应的 Dashboard 即可。

操作步骤

1. 登录容器服务管理控制台。

2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

4. 对模板进行相关配置，部署 Grafana 的 deployment 和 service，完成配置后，单击创建。
   - 集群：选择所需的集群。
   - 命名空间：选择资源对象所属的命名空间，必须是**kube-system**。
   - 示例模板：本示例选择自定义模板，其中包含一个 deployment 和 service。

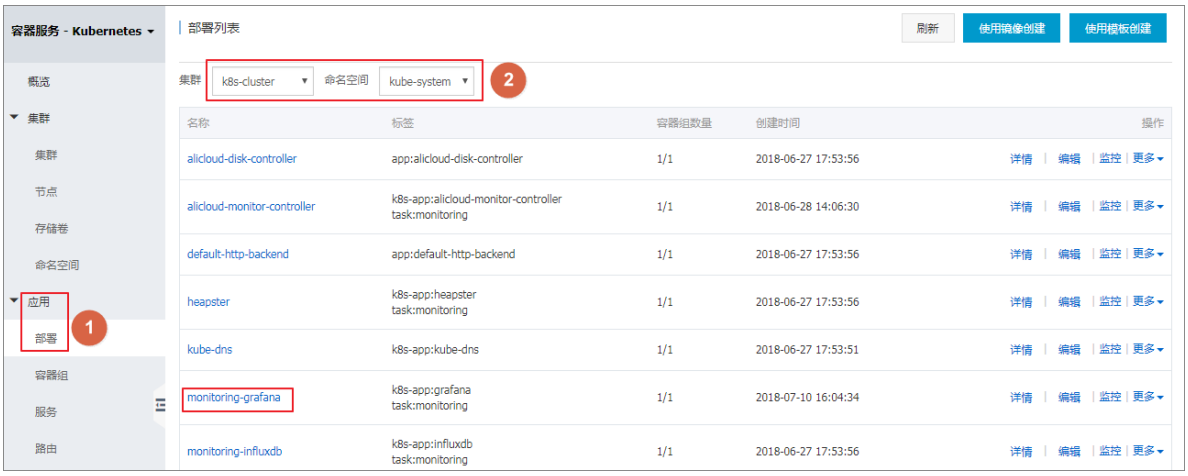


本示例的编排模板如下。

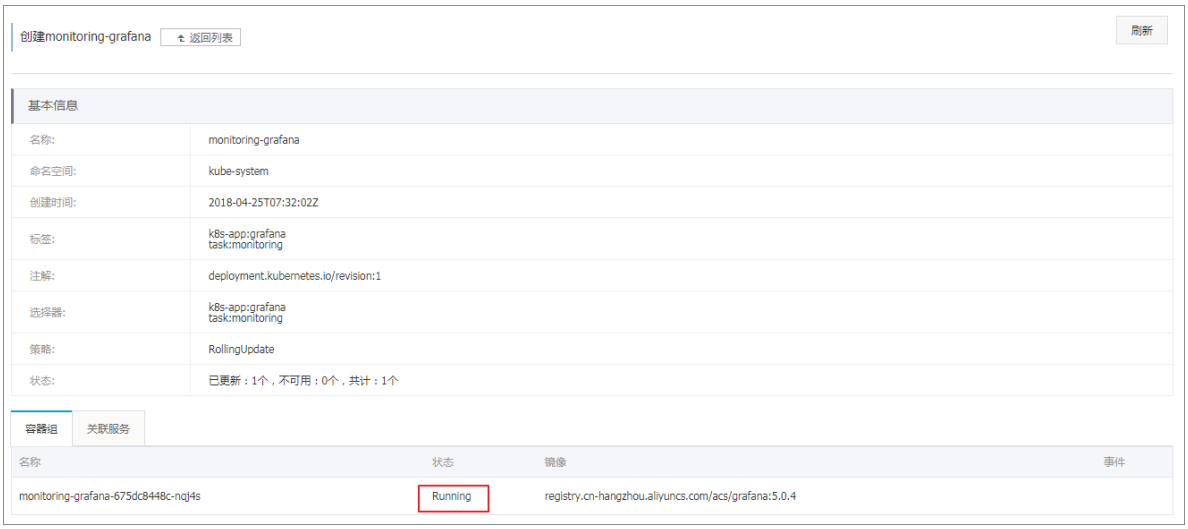
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
   name: monitoring-grafana
   namespace: kube-system
spec:
   replicas: 1
```

```
    template:
      metadata:
        labels:
          task: monitoring
          k8s-app: grafana
      spec:
        containers:
        - name: grafana
          image: registry.cn-hangzhou.aliyuncs.com/acs/grafana:5.0.4
          ports:
          - containerPort: 3000
            protocol: TCP
          volumeMounts:
          - mountPath: /var
            name: grafana-storage
          env:
          - name: INFLUXDB_HOST
            value: monitoring-influxdb
        volumes:
        - name: grafana-storage
          emptyDir: {}
  ---
apiVersion: v1
kind: Service
metadata:
    name: monitoring-grafana
    namespace: kube-system
spec:
    ports:
    - port: 80
      targetPort: 3000
    type: LoadBalancer
    selector:
      k8s-app: grafana
```

5. 完成部署后，返回部署页面，选择所需集群，然后选择 kube-system，查看其下部署的应用。

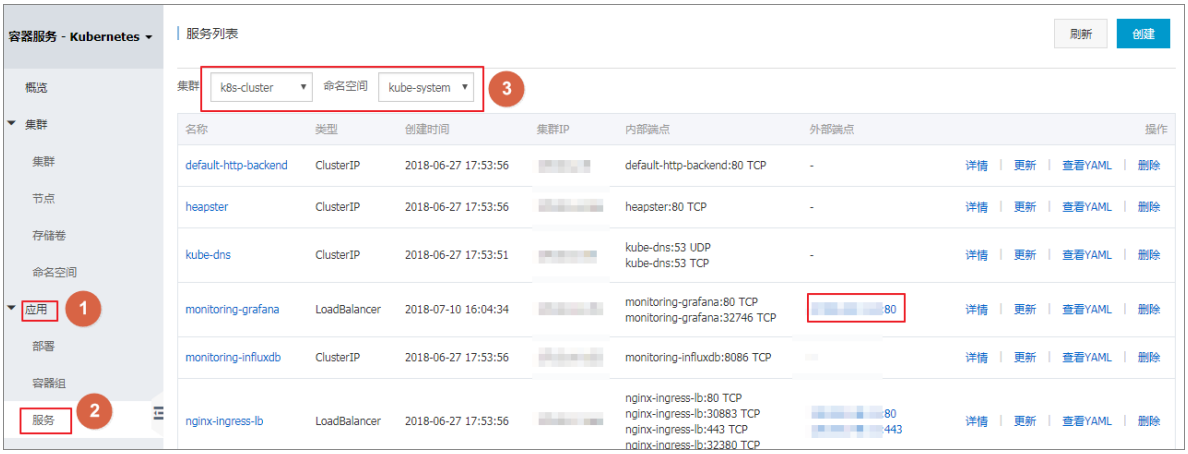


6. 单击 monitoring-grafana 的名称，查看部署状态，等待运行状态变为 running。



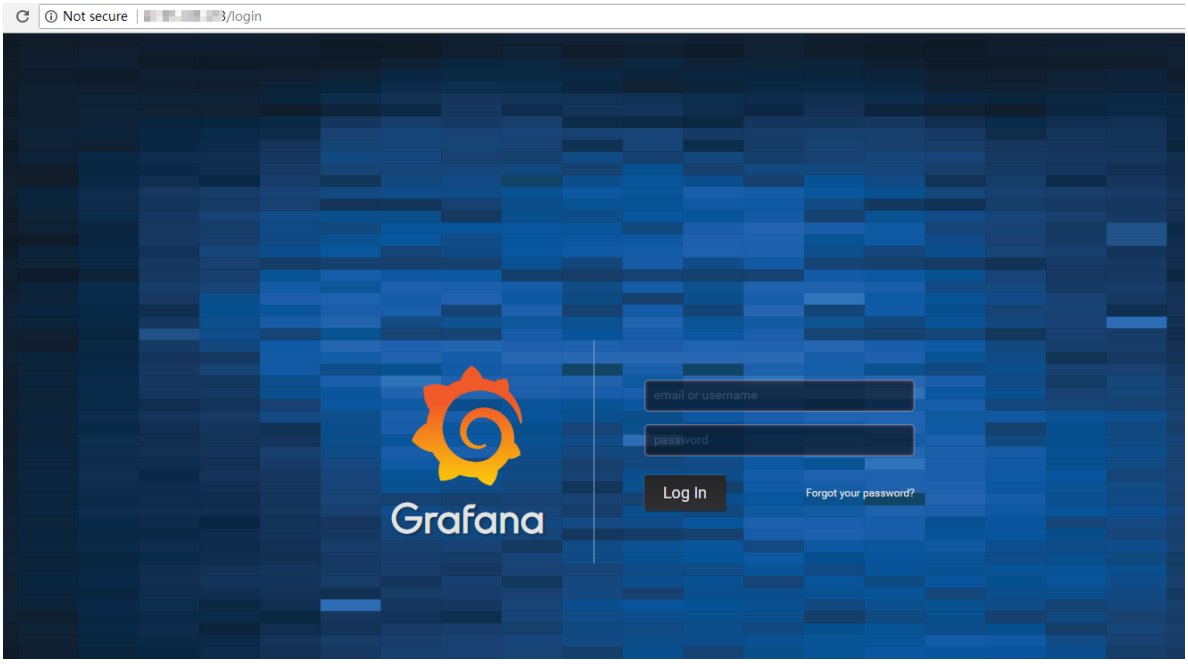
7. 单击左侧导航栏中的应用 > 服务，进入服务列表，选择所需的集群和命名空间（kube-system），查看外部端点。

   这个地址是通过 LoadBalancer 类型的 service 自动创建的，对于要求更安全访问策略的开发者而言，建议考虑添加 IP 白名单或者使用配置证书等方式增强安全性。



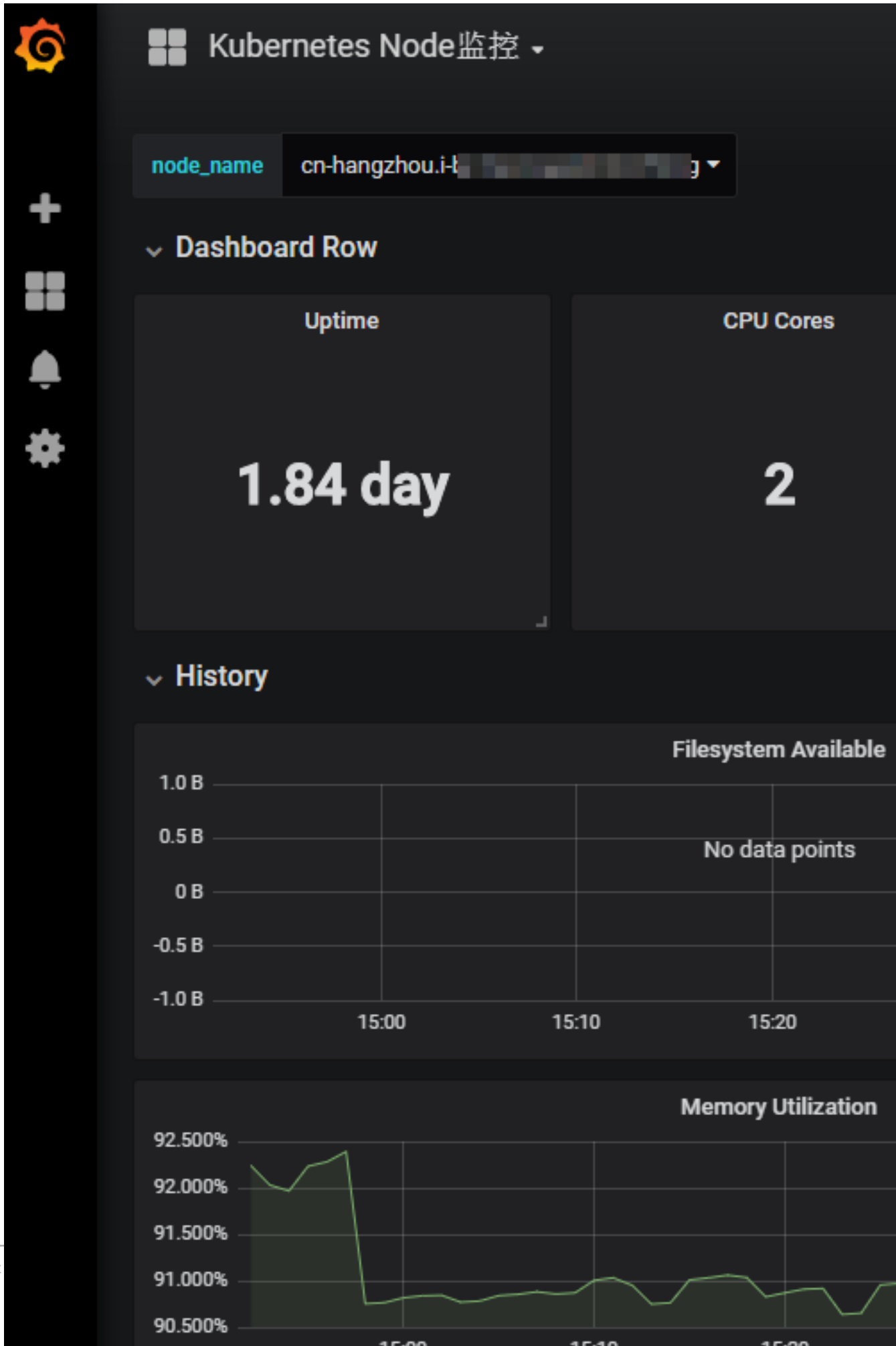
8. 选择 monitoring-grafana 服务，单击右侧的外部端点，登录 Grafana 监控界面。

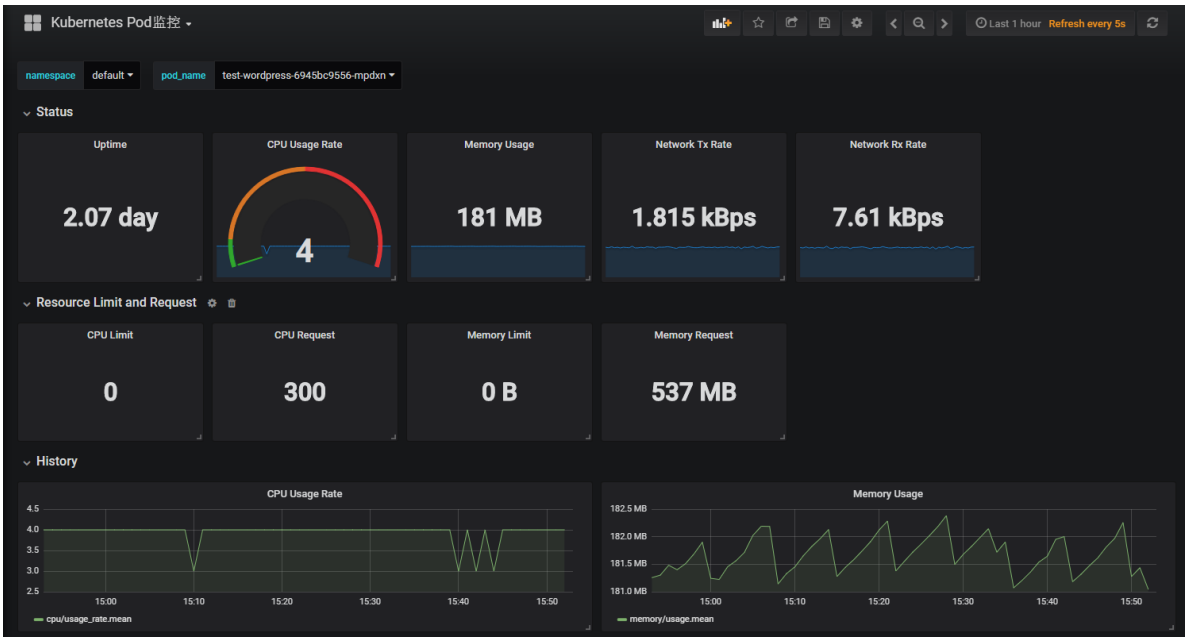
   默认的 Grafana 的用户名和密码都是 admin，建议在登录后先修改为更复杂的密码。



9. 您可选择内置的监控模板 ，查看 Pod 和 Node 的监控 Dahsboard。

   本示例使用的 Grafana 版本内置了两个模板，一个负责展示节点级别的物理资源，一个负责展示 Pod 相关的资源。开发者也可以通过添加自定义的 Dashboard 的方式进行更复杂的展现，也可以基于 Grafana 进行资源的告警等。


Kubernetes Node监控
node_name
cn-hangzhou.i-b
Dashboard Row
Uptime
1.84 day
CPU Cores
2
History
Filesystem Available
1.0 B
0.5 B
0 B
-0.5 B
-1.0 B
No data points
15:00
15:10
15:20
Memory Utilization
92.500%
92.000%
91.500%
91.000%
90.500%



1.12.5 使用HPA弹性伸缩容器

阿里云容器服务支持在控制台界面上快速创建支持HPA的应用，实现容器资源的弹性伸缩。您也可通过定义HPA(Horizontal Pod Autoscaling)的yaml配置来进行配置。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已成功连接到Kubernetes集群的Master节点。

方法1 通过容器服务控制台创建HPA应用

在阿里云容器服务中，已经集成了HPA，开发者可以非常简单地通过容器服务控制台进行创建。

1. 登录 容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，单击右上角的使用镜像创建。

3. 填写应用的名称，设置应用部署集群和命名空间，单击下一步。
4. 首先进行应用设置，设置副本数量，然后勾选开启自动伸缩，设置伸缩的条件和配置。

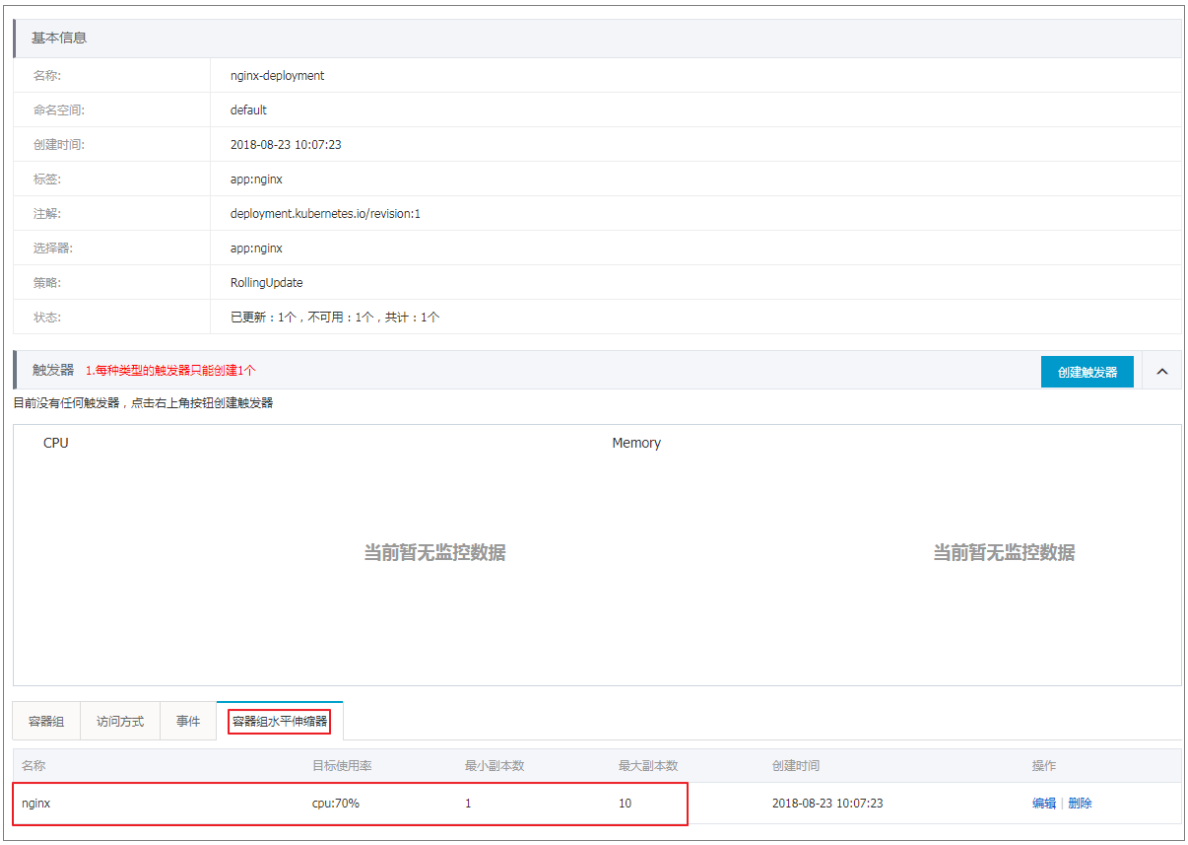
- **指标**：支持CPU和内存，需要和设置的所需资源类型相同。
- **触发条件**：资源使用率的百分比，超过该使用量，容器开始扩容。
- **最大容器数量**：该Deployment可扩容的容器数量上限。
- **最小容器数量**：该Deployment可缩容的容器数量下限。

5. 进行容器设置，选择镜像，并设置所需的资源。然后单击下一步

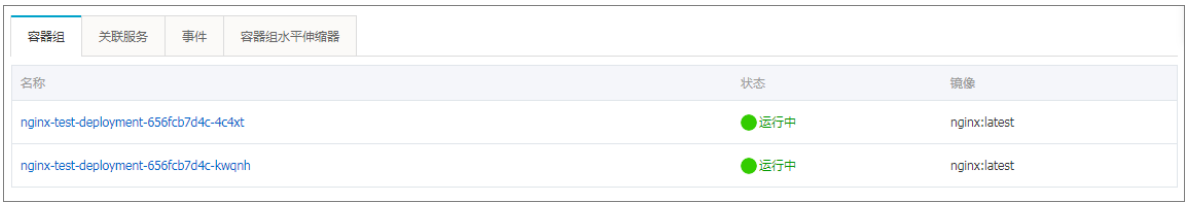
> **说明：**
> 您必须为Deployment设置所需资源，否则无法进行容器自动伸缩。

6. 进入访问设置页面，本例中不进行访问设置，单击创建。

此时一个支持HPA的Deployment就已经创建完毕，您可在部署的详情中查看伸缩组信息。



7. 在实际使用环境中，应用会根据CPU负载进行伸缩。您也可在测试环境中验证弹性伸缩，通过给Pod进行CPU压测，可以发现Pod在半分钟内即可完成水平的扩展。



方法2 通过kubectl命令进行使用

您也可通过编排模板来手动创建HPA，并将其绑定到要伸缩的Deployment对象上，通过**`kubectl`**命令实现容器自动伸缩配置。

下面针对一个Nginx应用进行举例，Deployment的编排模板如下，执行**`kubectl create -f xxx.yml`**命令进行创建。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
```

```
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9 # replace it with your exactly <image_name:
tags>
        ports:
        - containerPort: 80
        resources:
          requests:                          ##必须设置，不然HPA无法运行
            cpu: 500m
```

然后创建HPA，通过**scaleTargetRef**设置当前HPA绑定的对象，在本例中绑定是名叫nginx的Deployment。

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
  namespace: default
spec:
  scaleTargetRef:                              ##绑定名为nginx的
Deployment
    apiVersion: apps/v1beta2
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

说明：

HPA需要给Pod设置request资源，如果没有request资源，HPA不会运行。

执行**kubectl describe hpa [name]**会发现有类似如下的warnning。

```
Warning  FailedGetResourceMetric       2m (x6 over 4m)  horizontal-pod
-autoscaler  missing request for cpu on container nginx in pod default
/nginx-deployment-basic-75675f5897-mqzs7
```

```
Warning  FailedComputeMetricsReplicas  2m (x6 over 4m)  horizontal-pod
-autoscaler  failed to get cpu utilization: missing request for cpu on
 container nginx in pod default/nginx-deployment-basic-75675f5
```

创建好HPA后，再次执行**kubectl describe hpa [name]**命令，可以看到如下信息，则表示HPA已经正常运行。

```
Normal SuccessfulRescale 39s horizontal-pod-autoscaler New size: 1;
reason: All metrics below target
```

此时当Nginx的Pod的利用率超过本例中设置的50%利用率时，则会进行水平扩容，低于50%的时候会进行缩容。

1.12.6 使用钉钉实现Kubernetes集群监控告警

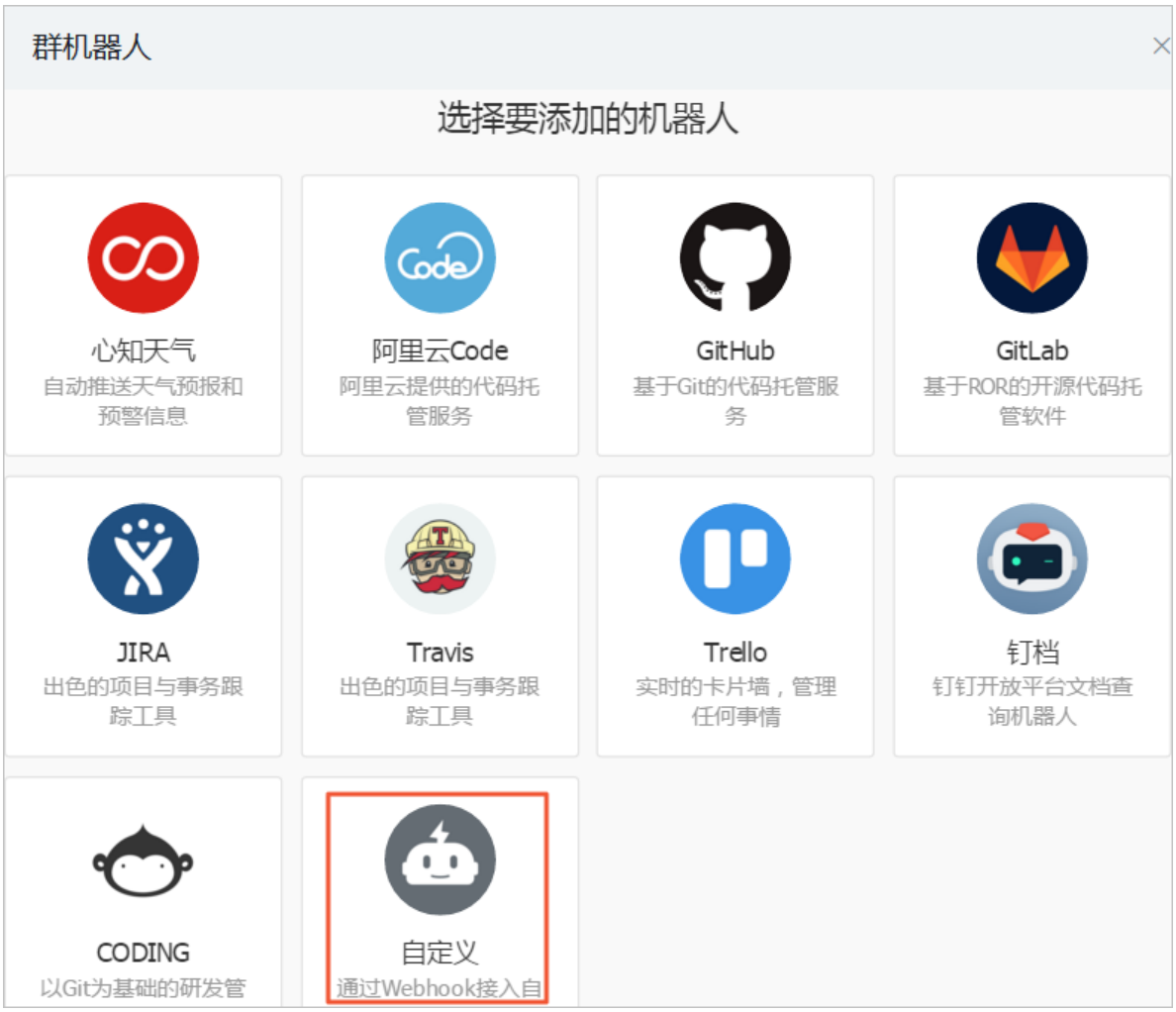
在钉钉群部署群机器人后，当Kubernetes集群出现异常事件时，可将异常事件通过群机器人发送到指定钉钉群，从而实现集群异常事件的实时监控告警。

背景信息

- 您已成功创建一个钉钉群。
- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。

操作步骤

1. 单击钉钉群右上角  图标，进入**群设置**页面。
2. 单击**群机器人**，进入**群机器人**页面，选择需要添加的机器人。此处选择**自定义**机器人。



3. 在机器人详情页面，单击添加，进入添加机器人页面。

4. 根据如下信息配置群机器人后，单击完成添加：

配置	说明
编辑头像	（可选）为群机器人设置头像。
机器人名字	添加的机器人名称。
添加到群组	添加机器人的群组。
是否开启Outgoing机制	（可选）通过@群机器人，将消息发送到指定外部服务，还可以将外部服务的响应结果返回到群组。 **说明：** 建议不开启。
POST 地址	接收消息的HTTP服务地址。 **说明：** 当选择开启Outgoing机制时，此项可配置。

配置	说明
Token	用于验证请求来自钉钉的密钥。 **说明：** 当选择开启Outgoing机制时，此项可配置。

5. 单击复制，复制webhook地址。

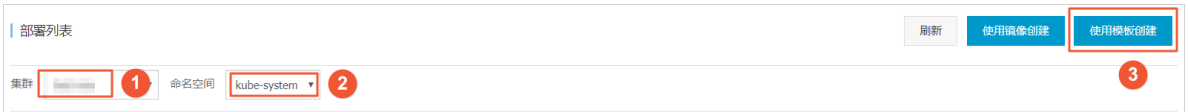
说明：

在群机器人页面，选择目标群机器人，单击右侧 图标可以：

- 修改群机器人的头像及机器人名字。
- 开启或关闭消息推送。
- 重置webhook地址。

- 删除群机器人。

6. 登录 容器服务管理控制台。
7. 在Kubernetes菜单下，单击左侧导航栏中的应用 > 部署，进入 部署列表页面。
8. 选择目标集群，命名空间选为**kube-system**，单击右上角使用模板创建。



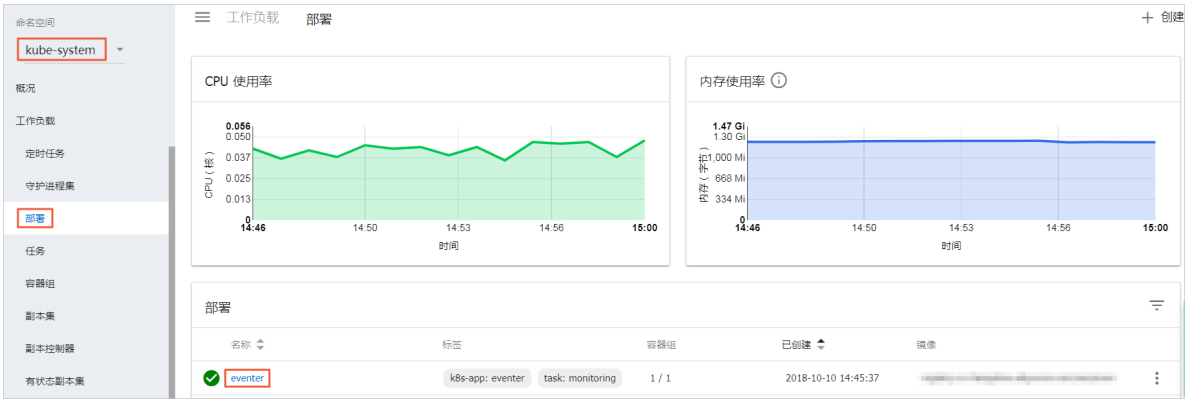
9. 根据以下信息配置模板，完成后单击创建。

配置	说明
集群	选择目标集群。
命名空间	选择资源对象所属的命名空间，默认是 **default**。此处选择**kube-system**。
示例模板	阿里云容器服务提供了多种资源类型的 Kubernetes yaml 示例模板，让您快速部署资源对象。您可以根据 Kubernetes Yaml 编排的格式要求自主编写，来描述您想定义的资源类型。此处选择自定义。

模板	填写以下自定义内容：
	`apiVersion: extensions/v1beta1`
	`kind: Deployment`
	`metadata:`
	`  name: eventer`
	`  namespace: kube-system`
	`spec:`
	`  replicas: 1`
	`  template:`
	`  metadata:`

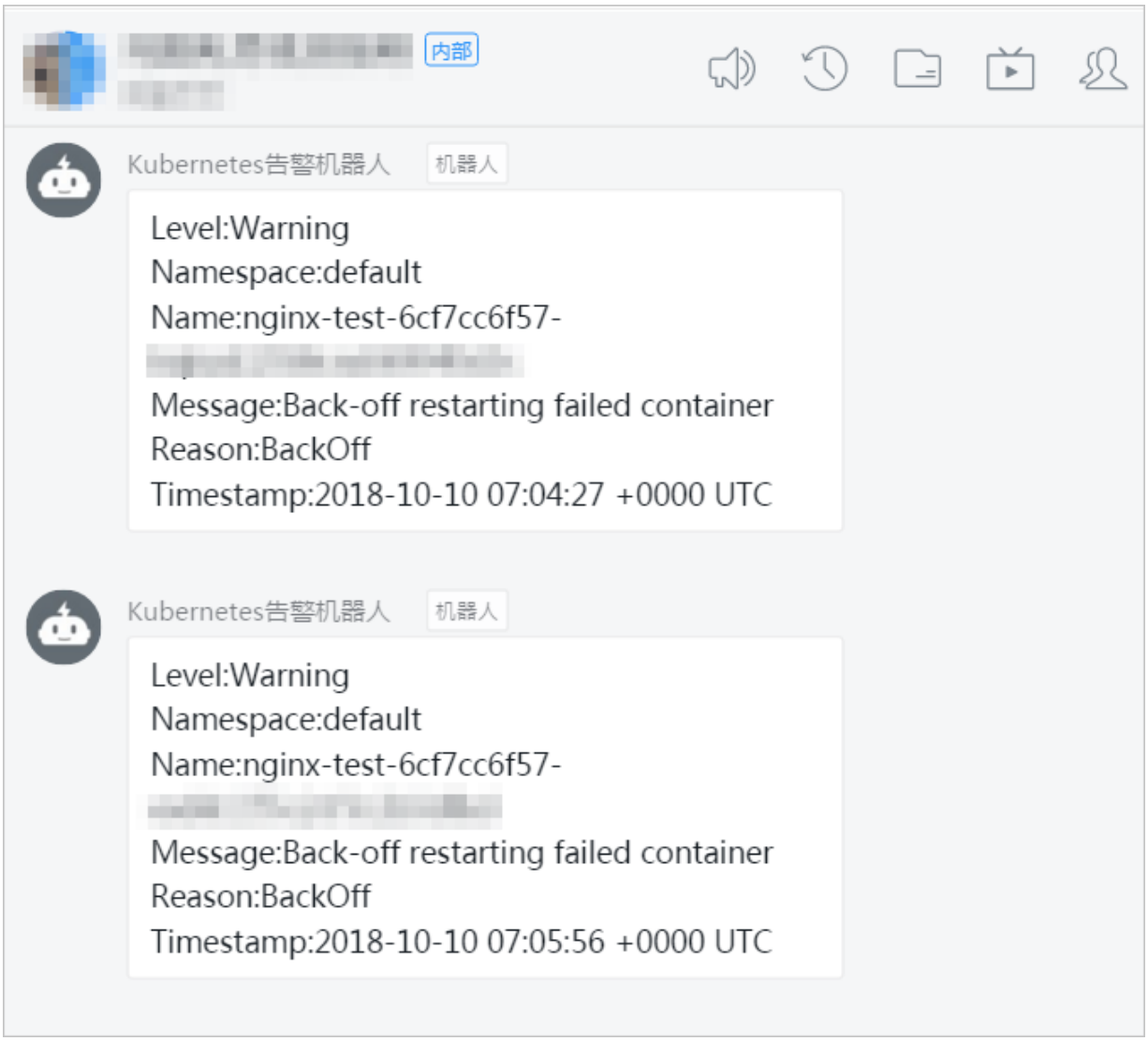
配置	说明
	labels: task: monitoring k8s-app: eventer annotations: scheduler.alpha.kubernetes.io/critical-pod: '' spec: serviceAccount: admin containers: - name: eventer image: registry.cn-hangzhou.aliyuncs.com/acs/eventer:v1.6.0 imagePullPolicy: IfNotPresent command: - /eventer - --source=kubernetes:https://kubernetes.default - --sink=dingtalk:[your_webhook_url]&label=[your_cluster_id]&level #level可配置为：Normal或Warning，默认值为：Warning。当配置Normal时，会在钉钉群收到Normal和Warning级别的告警；不配置或配置为Warning时，钉钉群仅收到Warning级别的告警。

在集群列表页面选择目标集群，单击操作列控制台，进入**Kubernetes** 控制台，选择命名空间为**kube-system**，单击左侧导航栏部署，可查看到eventer已部署成功。



预期结果

部署成功后30s，eventer生效，当事件等级超过阈值等级时，即可在钉钉群收到如下告警：



1.13 安全管理

1.13.1 概述

授权管理

Kubernetes 集群支持集群级别的操作的子账号授权。

详细操作参见使用子账号。

全链路 TLS 证书

在容器服务提供的 Kubernetes 集群中，存在的以下通信链路，均会进行 TLS 证书校验，以保证通信不被窃听或篡改。

- 位于 Worker 节点上的 `kubelet` 主动连接位于 Master 节点上的 `apiserver` 时

- 位于 Master 节点上的 `apiserver` 主动连接位于 Worker 节点上的 `kubelet` 时

在初始化过程中，发起初始化的 Master 节点会通过 SSH 隧道的方式连接到其他节点的 SSH 服务（22 端口）进行初始化。

原生 Secret&RBAC 支持

Kubernetes Secret 用于存储密码、OAuth Token、SSH Key 等敏感信息。明文地将这些敏感信息写在 Pod YAML 文件中或者写在 Dockerfile 固化到镜像中，会有信息泄露的可能，使用 Secret 能有效地避免这些安全隐患。

详细信息参见*Secret*。

Role-Based Access Control (RBAC) 使用 Kubernetes 内置的 API 组来驱动授权鉴权管理，您可以通过 API 来管理不同的 Pod 对应到不同的角色，以及各自的角色拥有的访问权限。

详细信息参见*Using RBAC Authorization*。

网络隔离

Kubernetes 集群中不同节点间 Pod 默认是可以互相访问的。在部分场景中，不同业务之间不应该网络互通，为了减少风险，您需要引入网络隔离（Network Policy）。在 Kubernetes 集群中，您可以使用 Canal 网络驱动实现网络隔离支持。

镜像安全扫描

Kubernetes 集群可使用容器镜像服务进行镜像管理，镜像服务支持镜像安全扫描。

镜像安全扫描可以快速识别镜像中存在的安全风险，减少 Kubernetes 集群上运行的应用被攻击的可能性。

详细描述参见 镜像安全扫描。

安全组与公网访问

每个新建的集群会被默认分配一个新的、安全风险最小化的安全组。该安全组对于公网入方向仅允许 ICMP。

创建集群默认不允许公网 SSH 连入，您可以参考*SSH访问Kubernetes集群* 配置通过公网 SSH 连入到集群节点中。

集群节点通过 NAT 网关访问公网，可进一步减少安全风险。

1.13.2 Kube-apiserver审计日志

在Kubernetes集群中，apiserver的审计日志可以帮助集群管理人员记录或追溯不同用户的日常操作，是集群安全运维中重要的环节。本文旨在帮助您了解阿里云Kubernetes集群apiserver审计日志的相关配置，以及如何通过SLS日志服务收集和搜索指定的日志内容。

配置介绍

当前创建Kubernetes集群会默认开启apiserver审计功能，相关的参数配置功能如下：

说明：

登录到Master节点，apiserver配置文件的目录是*/etc/kubernetes/manifests/kube-apiserver.yaml*。

配置	说明
audit-log-maxbackup	审计日志最大分片存储10个日志文件
audit-log-maxsize	单个审计日志最大size为100MB
audit-log-path	审计日志输出路径为*/var/log/kubernetes/kubernetes.audit*
audit-log-maxage	审计日志最多保存期为7天
audit-policy-file	审计日志配置策略文件，文件路径为：*/etc/kubernetes/audit-policy.yml*

登录Master节点机器，审计配置策略文件的目录是*/etc/kubernetes/audit-policy.yml*，内容如下：

```
apiVersion: audit.k8s.io/v1beta1 # This is required.
kind: Policy
# Don't generate audit events for all requests in RequestReceived
stage.
omitStages:
  - "RequestReceived"
rules:
  # The following requests were manually identified as high-volume and
 low-risk,
  # so drop them.
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
      - group: "" # core
        resources: ["endpoints", "services"]
  - level: None
    users: ["system:unsecured"]
    namespaces: ["kube-system"]
    verbs: ["get"]
```

```
    resources:
      - group: "" # core
        resources: ["configmaps"]
  - level: None
    users: ["kubelet"] # legacy kubelet identity
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes"]
  - level: None
    userGroups: ["system:nodes"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes"]
  - level: None
    users:
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
    resources:
      - group: "" # core
        resources: ["endpoints"]
  - level: None
    users: ["system:apiserver"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["namespaces"]
  # Don't log these read-only URLs.
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
  # Don't log events requests.
  - level: None
    resources:
      - group: "" # core
        resources: ["events"]
  # Secrets, ConfigMaps, and TokenReviews can contain sensitive &
binary data,
  # so only log at the Metadata level.
  - level: Metadata
    resources:
      - group: "" # core
        resources: ["secrets", "configmaps"]
      - group: authentication.k8s.io
        resources: ["tokenreviews"]
  # Get repsonses can be large; skip them.
  - level: Request
    verbs: ["get", "list", "watch"]
    resources:
      - group: "" # core
      - group: "admissionregistration.k8s.io"
      - group: "apps"
      - group: "authentication.k8s.io"
      - group: "authorization.k8s.io"
      - group: "autoscaling"
      - group: "batch"
```

```
      - group: "certificates.k8s.io"
      - group: "extensions"
      - group: "networking.k8s.io"
      - group: "policy"
      - group: "rbac.authorization.k8s.io"
      - group: "settings.k8s.io"
      - group: "storage.k8s.io"
  # Default level for known APIs
  - level: RequestResponse
    resources:
      - group: "" # core
      - group: "admissionregistration.k8s.io"
      - group: "apps"
      - group: "authentication.k8s.io"
      - group: "authorization.k8s.io"
      - group: "autoscaling"
      - group: "batch"
      - group: "certificates.k8s.io"
      - group: "extensions"
      - group: "networking.k8s.io"
      - group: "policy"
      - group: "rbac.authorization.k8s.io"
      - group: "settings.k8s.io"
      - group: "storage.k8s.io"
  # Default level for all other requests.
  - level: Metadatak8s.io"
      - group: "storage.k8s.io"
  # Default level for all other requests.
  - level: Metadata
```

说明：

- 在收到请求后不立即记录日志，当返回体header发送后才开始记录。
- 对于大量冗余的kube-proxy watch请求，kubelet和system:nodes对于node的get请求，kube组件在kube-system下对于endpoint的操作，以及apiserver对于namespaces的get请求等不作审计。
- 对于*/healthz**，*/version**，*/swagger**等只读url不作审计。
- 对于可能包含敏感信息或二进制文件的secrets，configmaps，tokenreviews接口的日志等级设为metadata，该level只记录请求事件的用户、时间戳、请求资源和动作，而不包含请求体和返回体。
- 对于一些如authenticatioin、rbac、certificates、autoscaling、storage等敏感接口，根据读写记录相应的请求体和返回体。

采集和检索

在使用Kube-apiserver审计日志之前，请确保在创建集群的配置中开启了SLS日志服务，并成功创建对应的日志Project和Logstore。

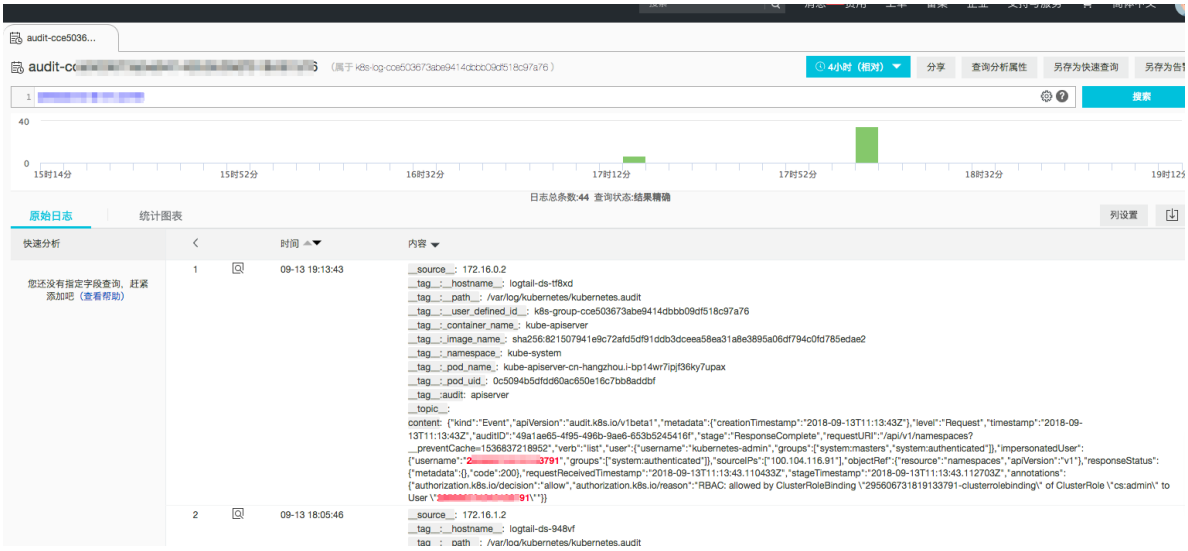
1. 登录 日志服务管理控制台。

2. 单击左侧导航栏中**Project管理**，选择创建集群时设置的日志Project，单击名称进入日志Project页面。

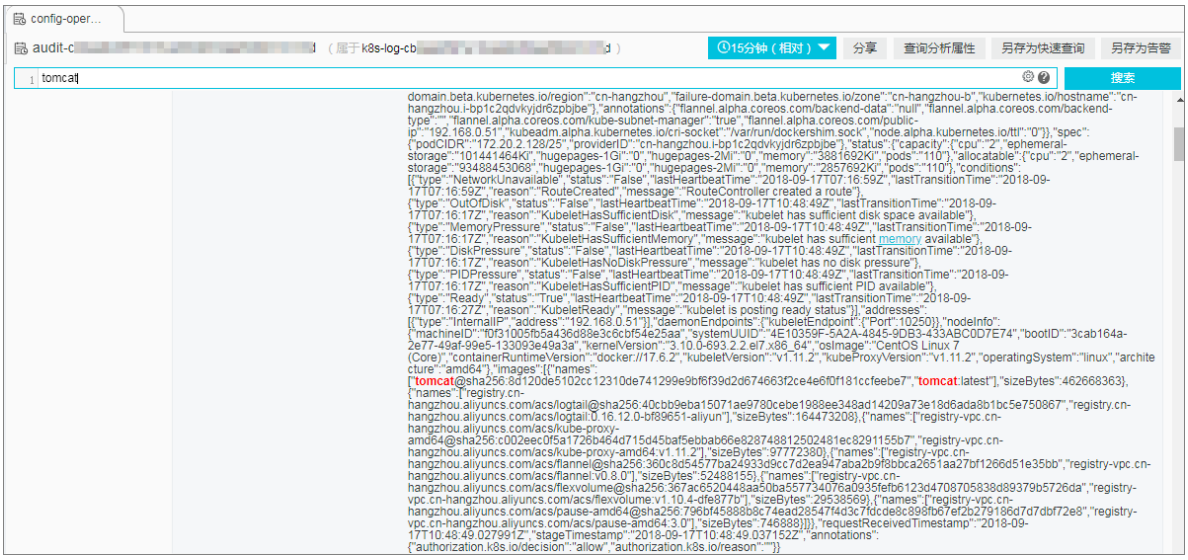
3. 在Project详情页中，默认进入日志库页面，查看名为audit-${clustered}的日志库（logstore），单击右侧的**查询**，集群对应的审计日志会收集在该日志库中。

> **说明：**
> 在您创建过程中，指定的日志Project中会自动添加一个名称为audit-${clustereid}的日志库。

4. 当集群管理员需要关注某一子账号的行为时，可以输入相应子账号id，追溯其操作。



5. 当集群管理员关注某一具体资源对象时，可以输入相应的资源名称，检索时间段内的指定操作。



支持第三方日志解决

您可以在集群Master各节点，在*/var/log/kubernetes/kubernetes.audit* 路径下找到审计日志的源文件。该文件是标准的json格式，您可以在部署集群时选择不使用阿里云的日志服务，根据需要对接其他的日志解决方案，完成相关审计日志的采集和检索。

1.13.3 在Kubernetes中实现HTTPS安全访问

当前容器服务Kubernetes集群支持多种应用访问的形式，最常见形式如`SLB:Port`，`NodeIP:NodePort`和域名访问等。但是Kubernetes集群默认不支持HTTPS访问，如果用户希望能够通过HTTPS进行应用的访问，容器服务和阿里云负载均衡服务为您提供安全的HTTPS访问。本文旨在通过实际案例演示的HTTPS访问配置，帮助用户在容器服务Kubernetes中配置自己的证书。

根据访问的方式不同，当前可以分为两种配置证书的方式：

- 在前端SLB上配置证书
- 在Ingress中配置证书

前提条件

- 您已创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已经通过SSH连接到Master节点，参见*SSH*访问*Kubernetes*集群。
- 连接到Master节点后，创建集群的服务器证书，包括公钥证书和私钥。您可通过以下命令快速创建。

```
$  openssl genrsa -out tls.key 2048

Generating RSA private key, 2048 bit long modulus
.....................................................................+++
```

```
.....................................................................................
+++
e is 65537 (0x10001)

$  openssl req -sha256 -new -x509 -days 365 -key tls.key -out tls.
crt

You are about to be asked to enter information that will be
incorporated
...
-----
Country Name (2 letter code) [XX]:CN
State or Province Name (full name) []:zhejiang
Locality Name (eg, city) [Default City]:hangzhou
Organization Name (eg, company) [Default Company Ltd]:alibaba
Organizational Unit Name (eg, section) []:test
Common Name (eg, your name or your server's hostname) []:foo.bar.com
           #注意，您需要正确配置域名
Email Address []:a@alibaba.com
```

方法1 在SLB上配置HTTPS证书

该方式有如下特点：

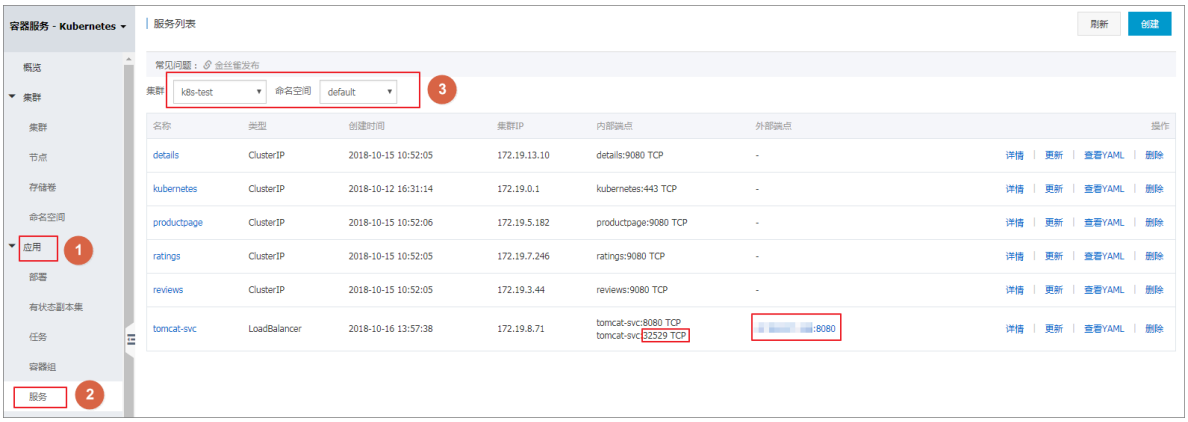
- **优点**： 证书配置在SLB上，为应用外部访问的入口，在集群内部进行应用的访问依然用的是http访问方式。
- **缺点**：需要维护较多的域名与IP地址的对应关系。
- **适用场景**：应用不使用Ingress暴露访问方式，通过LoadBalancer类型的service进行应用访问的暴露。

准备工作

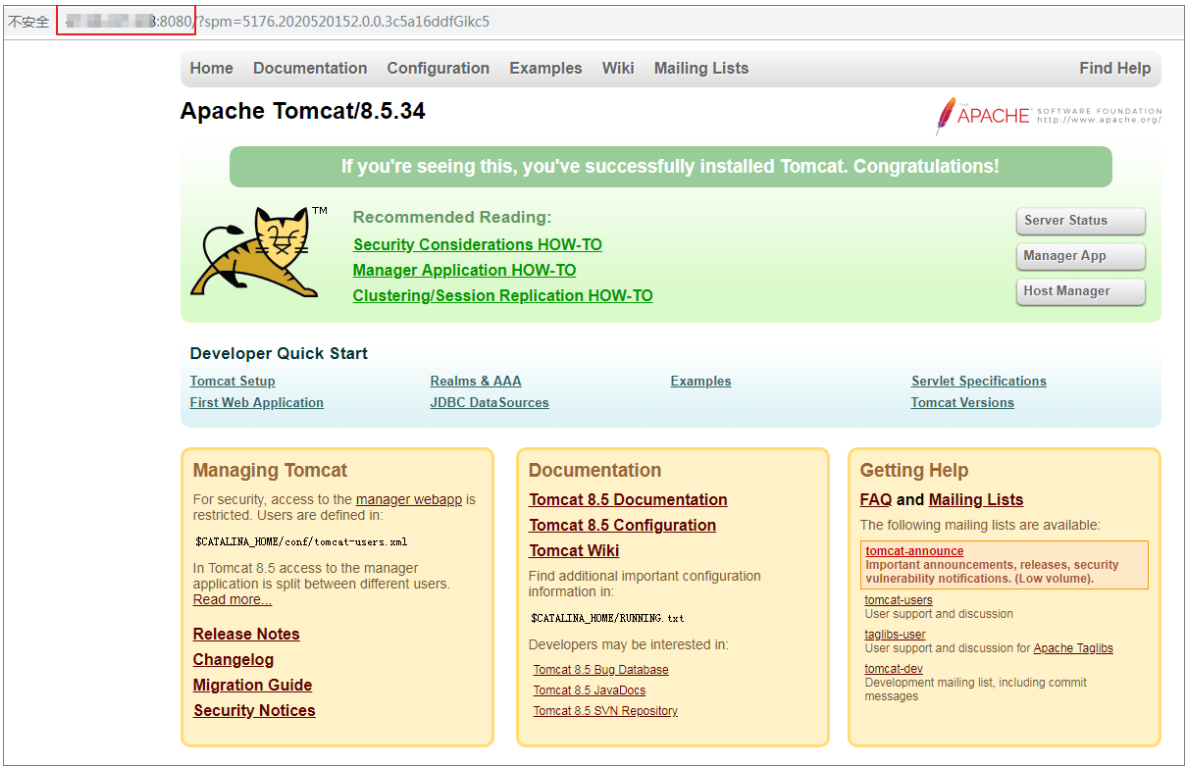
您已在该Kubernetes集群中创建一个Tomcat应用，该应用采用LoadBalancer类型的服务（service）对外提供访问。参见创建服务。

示例

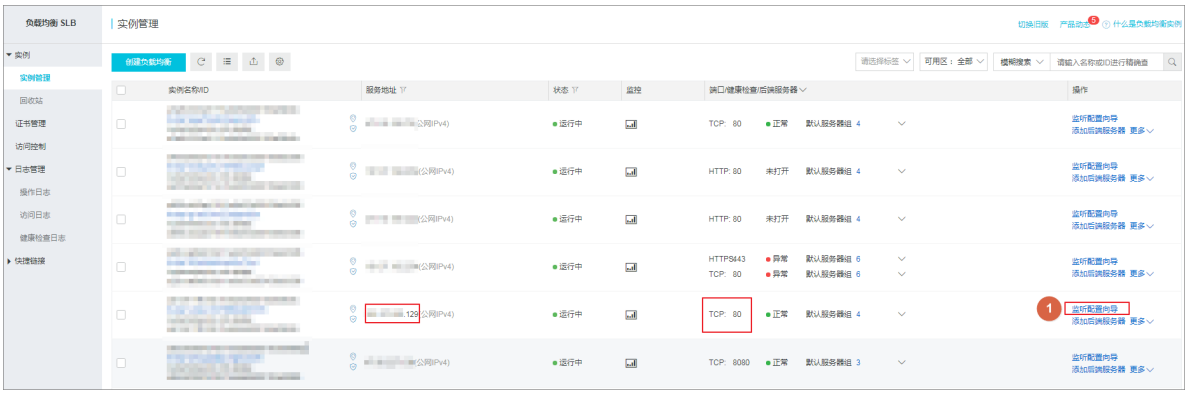
1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中**应用 > 服务**，选择所需集群和命名空间，查看预先创建的tomcat示例应用。如下图所示创建的tomcat应用名称为tomcat，服务名称为tomcat-svc。其中，服务类型为LoadBalancer，暴露的服务端口为8080。



3. 单击外部端点，您可通过`IP:Port`的方式访问tomcat应用。



4. 登录负载均衡管理控制台。
5. 默认进入实例管理页面，在服务地址栏中，找到与tomcat-svc服务外部端点对应的负载均衡实例，单击操作列中的监听配置向导。



6. 开始进行负载均衡配置，首先进行配置监听协议。选择**HTTPS协议**，监听端口设置为**443**，然后单击下一步。
7. 配置**SSL证书**。
   a. 首先单击新建服务器证书。

   b. 在弹出的创建证书页面中，选择证书来源。本例中选择**上传第三方签发证书**，然后单击下一步。
   c. 在上传第三方签发证书页面中，配置证书名称，选择证书部署区域，然后在**公钥证书**和**私钥**栏中输入前提条件中创建的服务器公钥证书和私钥，最后单击**确定**。

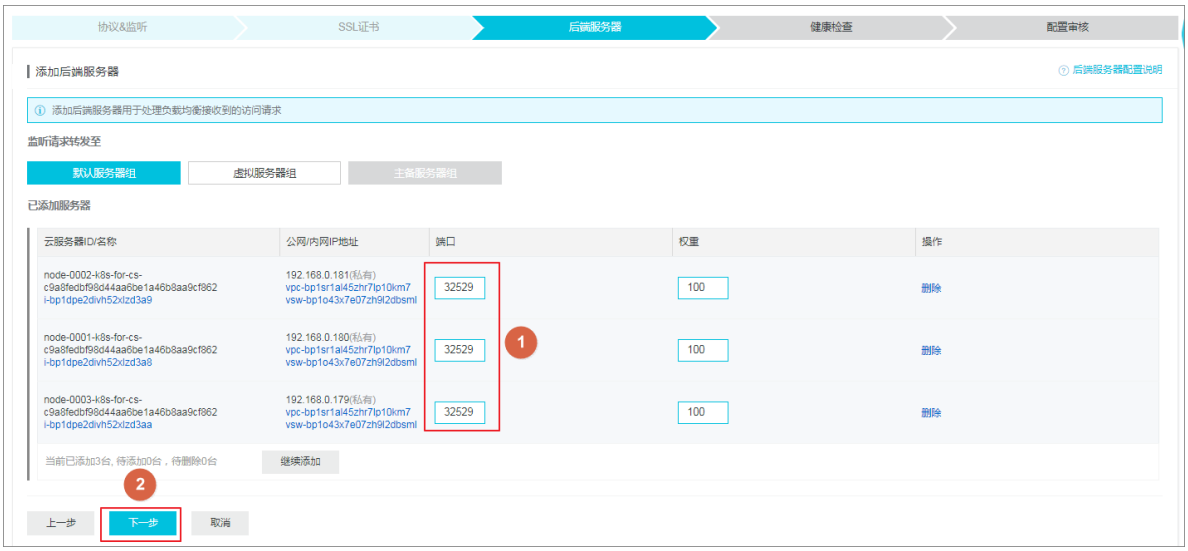

上传第三方签发证书
上传证书
证书名称
cert-tomcat
证书部署地域
华东 1
公钥证书：
(兼容NGINX格式)
上传
查看样例
私钥：
-----BEGIN RSA PRIVATE KEY-----
(兼容NGINX格式)
上传
查看样例
上一步
确定
取消

d. 然后在选择服务器证书栏选择刚创建的服务器证书。

e. 最后单击下一步。

8. 配置后端服务器，默认情况下已添加服务器，您需要配置后端服务器端口，用于监听tomcat-svc服务，最后单击下一步。

> 说明：
> 您需要在容器服务Web界面找到该服务对应的NodePort，并在后端服务器端口中配置该端口。



9. 配置健康检查，然后单击下一步。本例中采用默认配置。

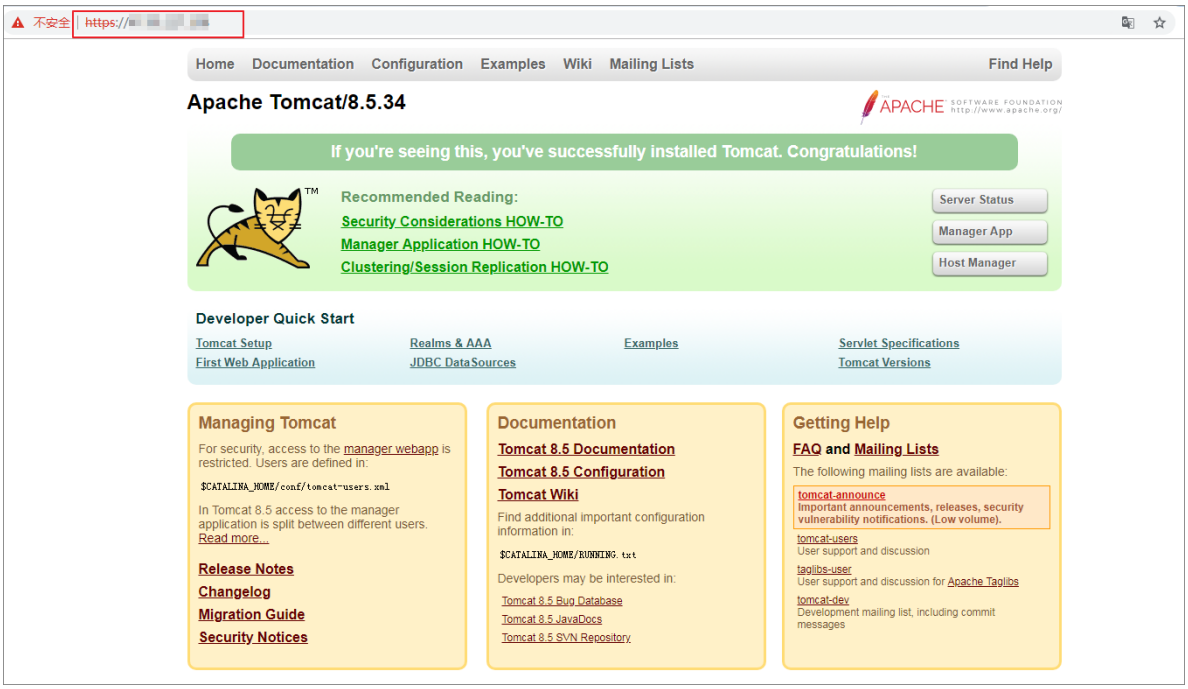
10. 进行配置审核，确认配置正确后，单击提交。

11. 配置成功后，单击确定。

12. 返回实例管理页面，您查看该实例，HTTPS:443监听规则已经生成。

13. 访问HTTPS的tomcat应用，在浏览器中输入`https://slb_ip`并进行访问。

> 说明：
> 如果在证书中加入了域名验证，可以使用域名进行访问。同时我们没有删除`tcp:8080`，所以通过`slb_ip:8080`也可以访问。



方法2 在Ingress上配置证书

该方法有如下特点：

- 优点：无需改动SLB的配置；每一个应用都可以通过Ingress管理自己的证书，互不干扰
- 适用场景：每个应用都需要单独的证书进行访问；或者集群中存在需要证书才能访问的应用。

准备工作

您已在该Kubernetes集群中创建一个Tomcat应用，该应用的服务（Service）采用ClusterIP的方式提供访问。本例中准备使用Ingress对外提供HTTPS访问服务。

示例

1. 登录到Kubernetes集群的Master节点，根据准备好的证书创建secret。

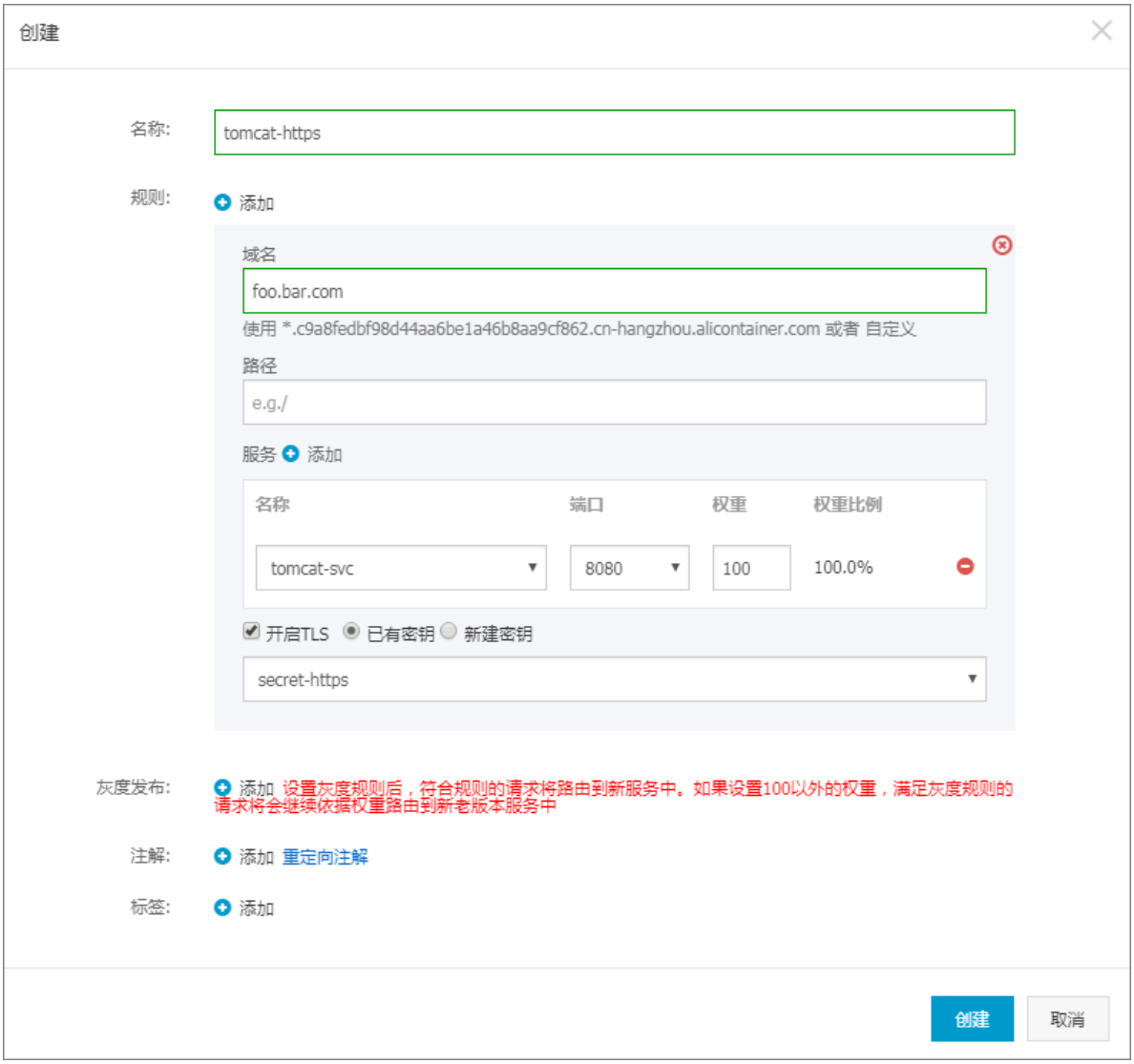
   > **说明：**
   > 在这里需要正确配置域名，否则后续通过HTTPS访问会有问题。

   ```
   kubectl create secret tls secret-https --key tls.key --cert tls.crt
   ```

2. 登录 容器服务管理控制台。
3. 单击左侧导航栏的应用 > 路由，选择所需的集群和命名空间，单击右上角创建。
4. 在创建路由对话框中，配置可HTTPS访问的路由，完成后单击确定。

更多详细的路由配置信息，请参见通过 Web 界面创建路由。本例中进行如下配置。

- 名称：输入该路由的名称
- 域名：即是前面配置的正确域名，与ssl证书中配置的保持一致。
- 服务：选择tomcat应用对应的service，端口为8080。
- 开启**TLS**：开启TLS后，选择已创建的secret。



您也可采用yaml文件的方式创建路由（Ingress），本例对应的yaml示例文件如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
   name: tomcat-https
spec:
```

```
tls:
- hosts:
  - foo.bar.com
  secretName: secret-https
rules:
- host: foo.bar.com
  http:
    paths:
    - path: /
      backend:
        serviceName: tomcat-svc
        servicePort: 8080
```

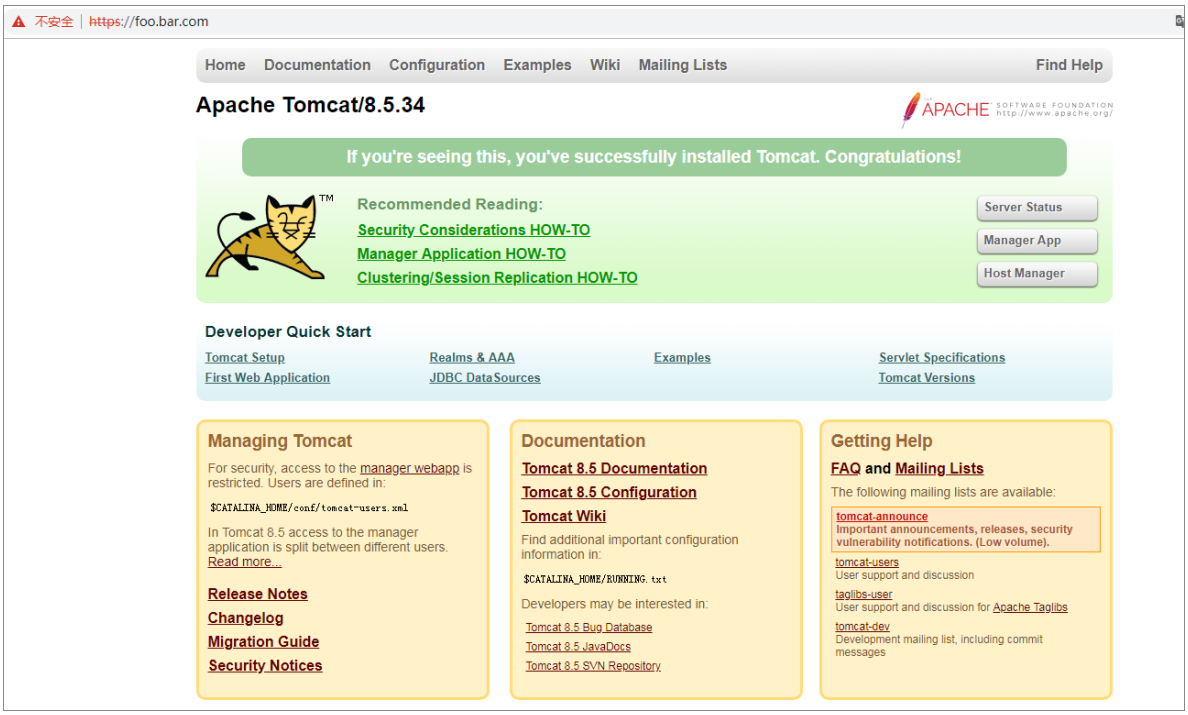
5. 返回路由列表，查看创建的路由（Ingress），本例中域名为`foo.bar.com`，并查看端点和域名，您也可进入路由详情页进行查看。

> 说明：
> 本例中以`foo.bar.com`作为测试域名，您需要在hosts文件中创建一条记录。

```
47.110.119.203  foo.bar.com                    #其中IP地址即是路由的端点。
```

6. 在浏览器中访问`https://foo.bar.com`。

> 说明：
> 由于创建了TLS证书访问，所以要用HTTPS来进行域名访问，针对该应用，本例以`foo.bar.com`为示例，在本地进行解析。在具体使用场景中，请使用备案过的域名。



1.14 发布管理

1.14.1 基于Helm的发布管理

阿里云 Kubernetes 服务集成了 Helm 包管理工具，帮助您快速构建云上应用，因为 chart 可以多次发布（release），这就带来一个发布版本管理的问题。因此，阿里云 Kubernetes 服务提供了发布功能，通过 Web 界面的方式对通过 Helm 发布的应用进行管理。

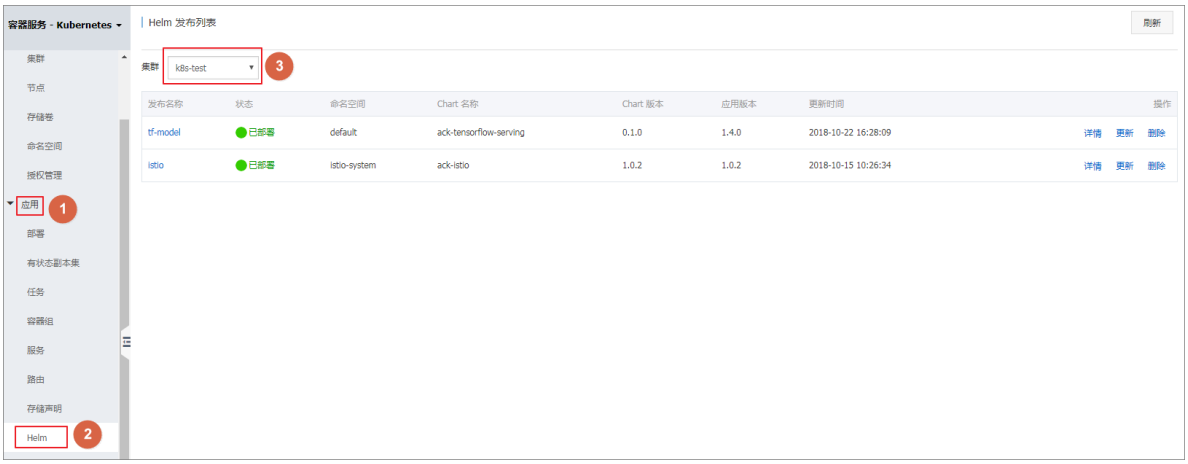
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已经使用应用目录或服务目录功能，安装了 Helm 应用，参见利用 *Helm* 简化应用部署。本例中是一个 tf-model 应用。

查看发布的详情

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > **Helm**，选择所需的集群，进入发布列表页面。

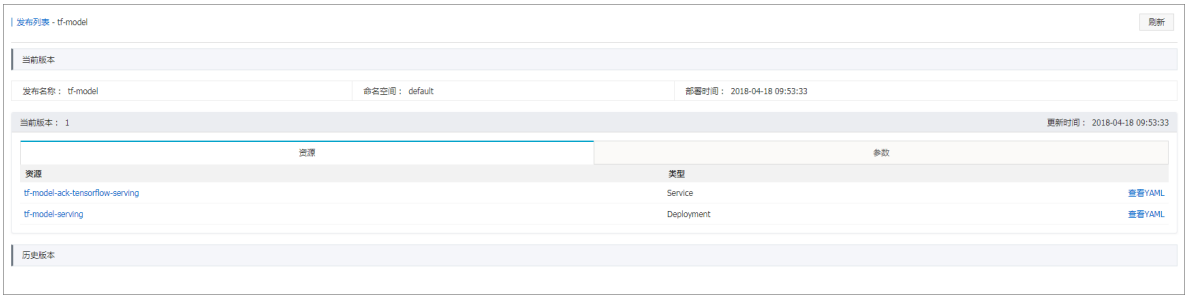
   您可看到该集群下通过 Helm 包管理工具发布的应用及服务等。



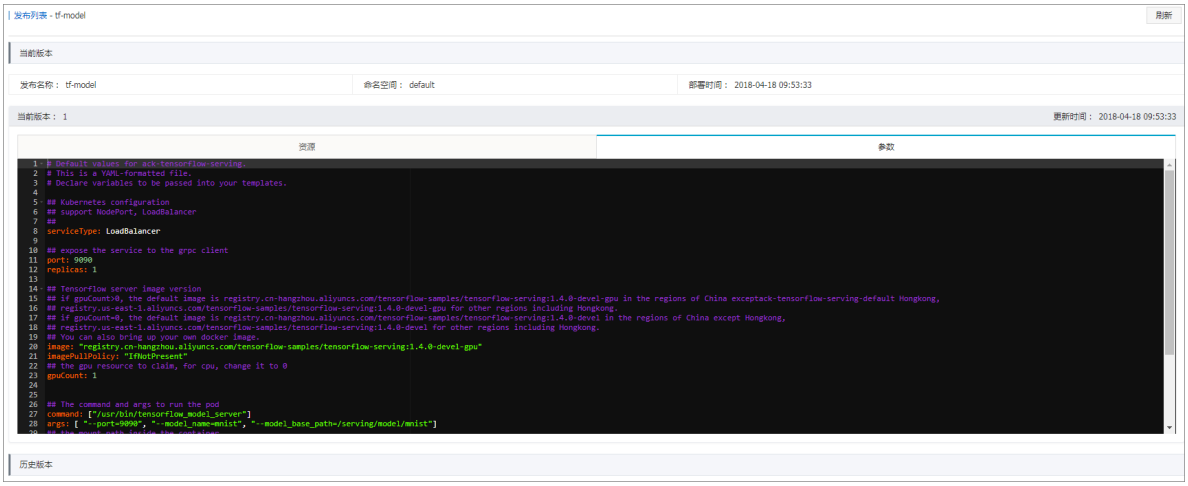
3. 以 tf-model 为例，您可查看发布的详情信息，单击右侧的详情，进入该发布的详情页面。

   你可以查看该发布的当前版本和历史版本等信息，当前版本为1，无历史版本。您还可查看 tf-model 的资源信息，如资源名称，资源类型，以及查看 YMAL 信息。

> **说明：**
>
> 单击资源名称，可进入 Kubernetes Dashboard 页面，查看对应资源的详细运行状态。



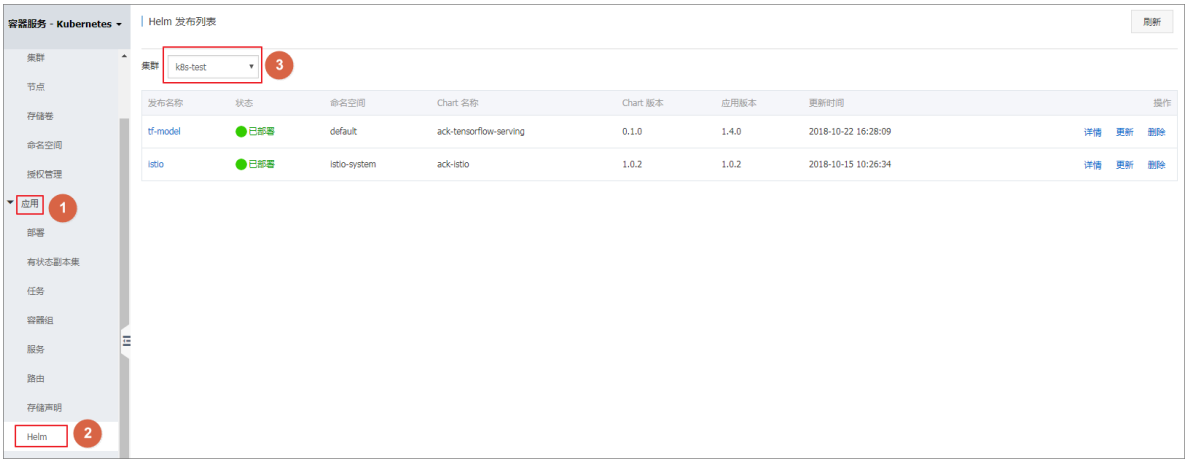
4. 单击参数，你可查看该 Helm 包安装的参数配置。



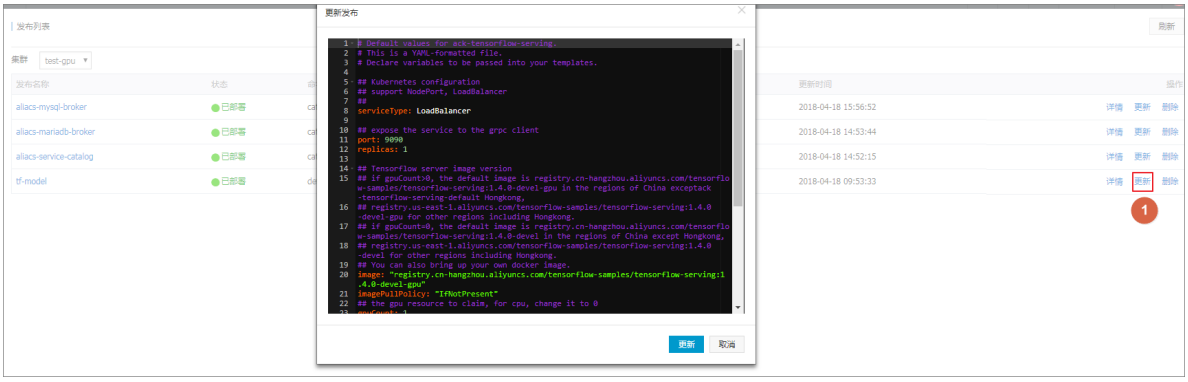
更新发布的版本

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > **Helm**，选择所需的集群，进入发布列表页面。

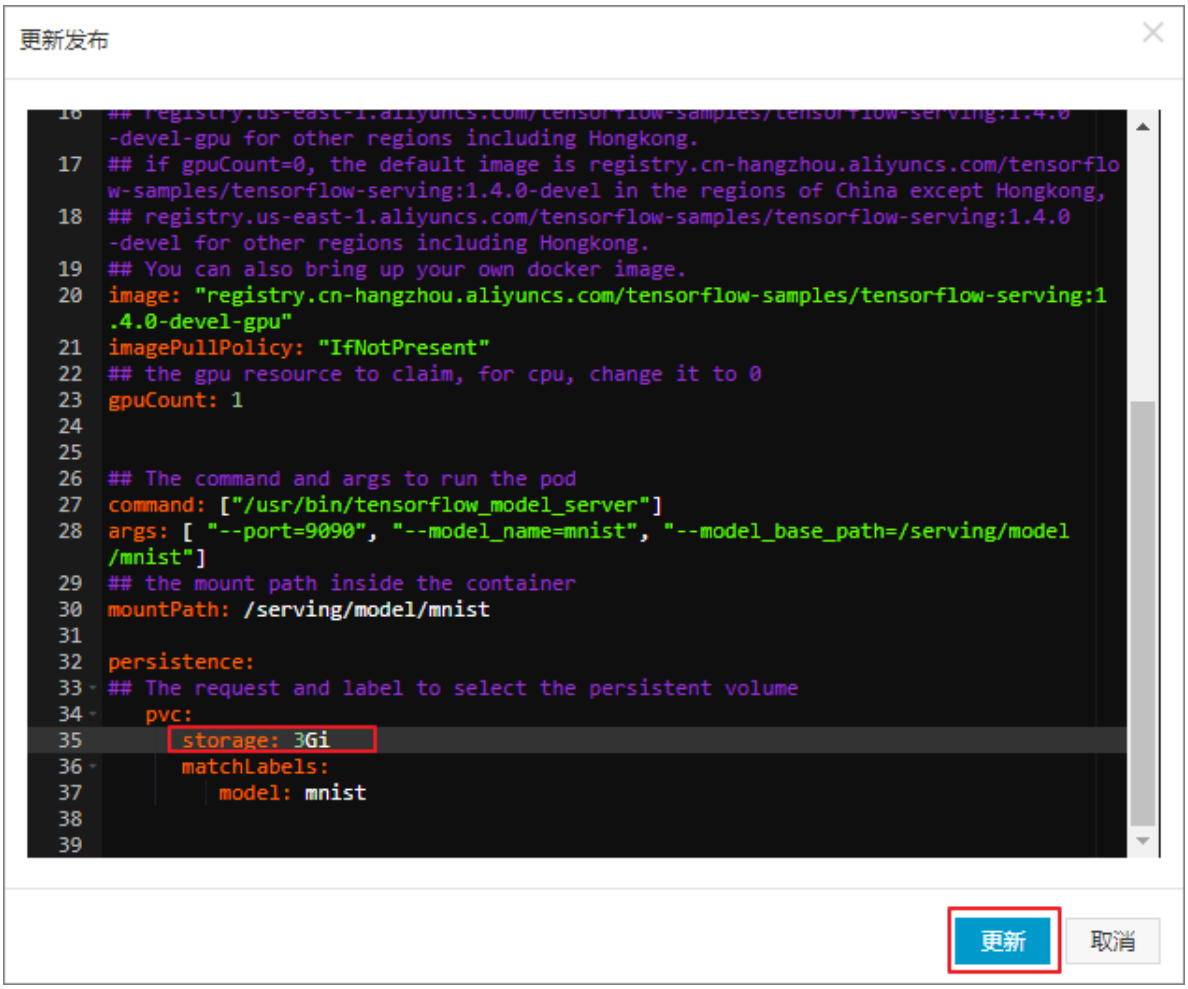
   您可看到该集群下通过 Helm 包管理工具发布的应用及服务等。



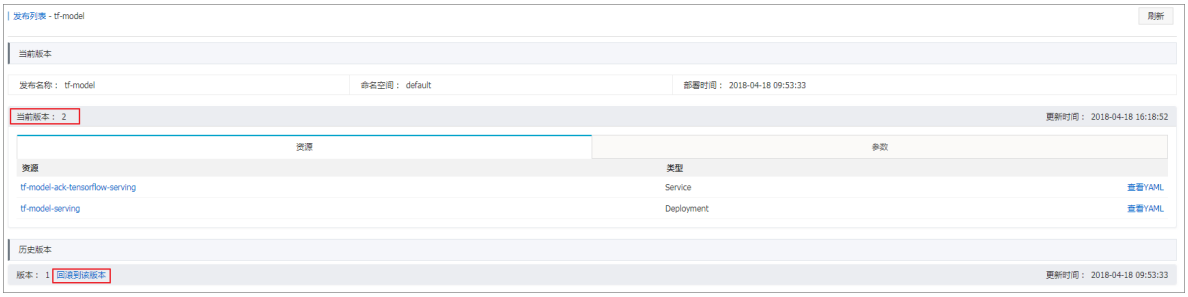
3. 以 tf-model 为例，您可更新该发布，单击右侧的更新，弹出更新发布对话框。



4. 在对话框中修改相关参数，随后单击更新，可对该发布进行更新。



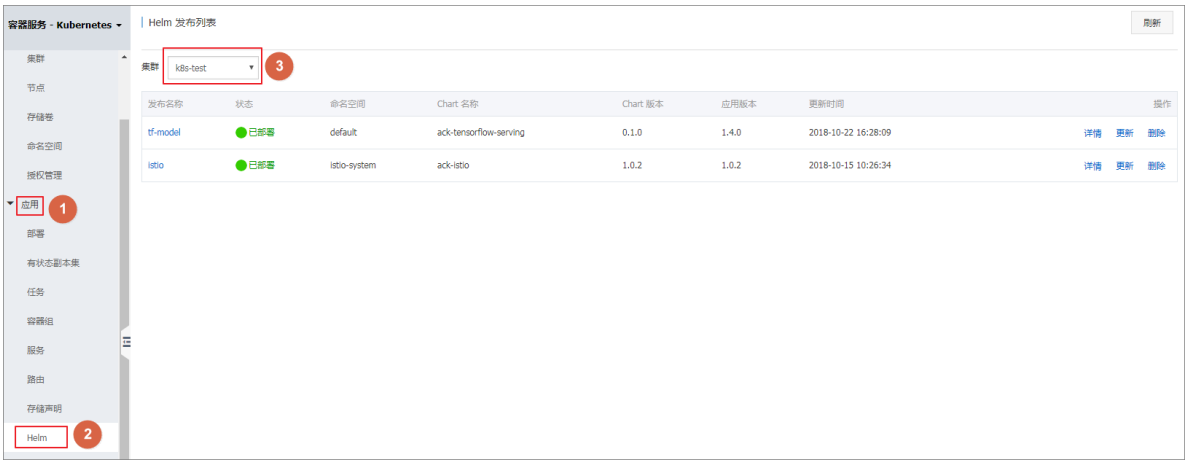
在发布列表页面，您可以看到当前版本变为 2，您可以在历史版本菜单下找到版本1，单击 回滚到该版本，可进行回滚。



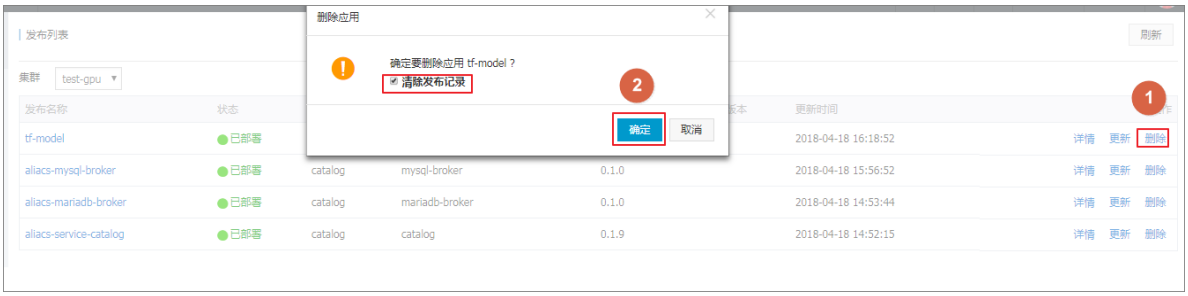
删除发布

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > **Helm**，选择所需的集群，进入发布列表页面。

   您可看到该集群下通过 Helm 包管理工具发布的应用及服务等。



3. 以 tf-model 为例，您可删除该发布，单击右侧的**删除**，弹出删除对话框。



4. 勾选是否**清除发布记录**，然后单击**确定**，您可以删除 tf-model 应用，其包含的 service、deployment 等资源都会一并删除。

1.14.2 在阿里云容器服务Kubernetes上使用分批发布

您可使用阿里云容器服务Kubernetes实现应用版本的分批发布，快速实现版本验证，支持应用快速迭代。

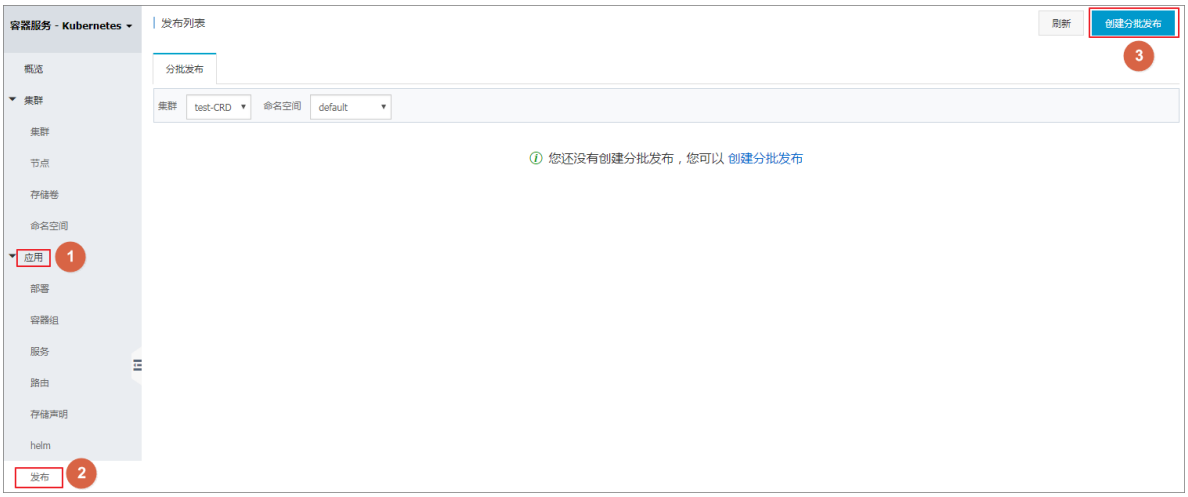
背景信息

> **说明：**
> Kubernetes最新集群已经默认安装alicloud-application-controller；对于旧版本的集群，目前仅支持1.9.3及以上的版本，您可通过控制台上的提示链接进行升级。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的**应用 > 发布**，单击右上角**创建分批发布**。

> **说明：**
> 如果按钮是灰色的，可以参考升级连接进行升级。

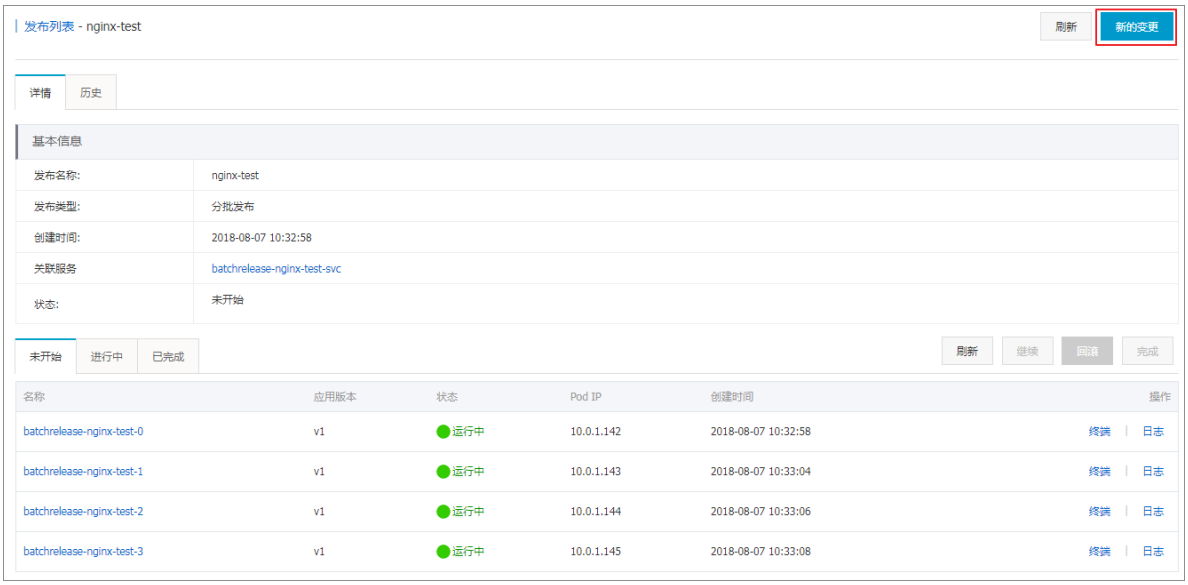


3. 首先配置分批发布信息，包括应用名称、部署集群、命名空间和发布选项。最后单击下一步。

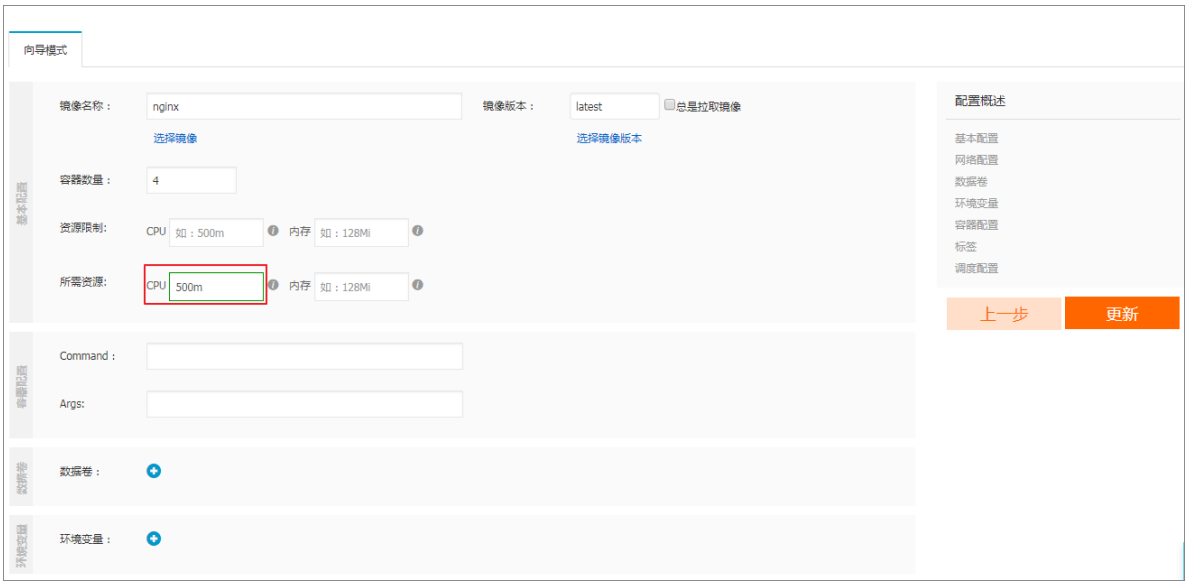
4. 在分批发布配置页面，配置后端的Pod和Service，最后单击更新，创建应用。

5. 返回发布列表，您可看到下面出现一个应用，状态显示为未开始，单击右侧的详情。

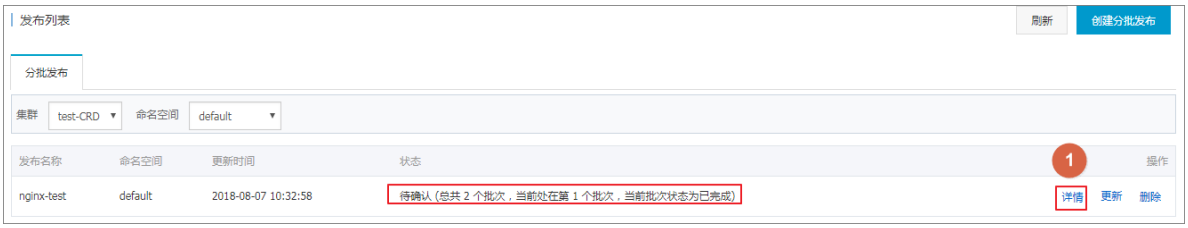
6. 在应用详情页面中，您可查看更多信息，单击页面右上角的新的变更，进行一次分批发布的变更。



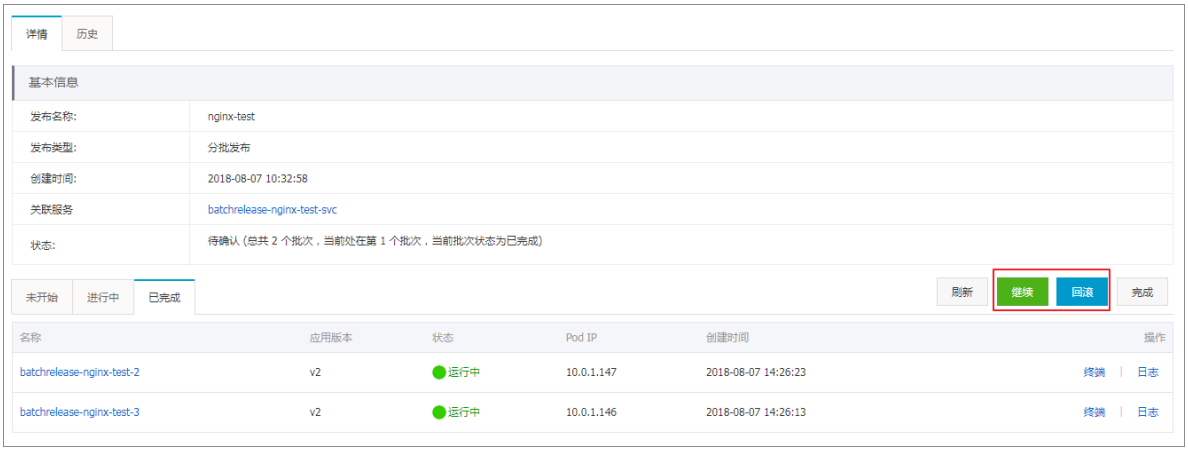
7. 在变更配置页面中，进行新版本应用的变更配置，完成后单击更新。



8. 默认返回发布列表页面，您可查看该应用的分批发布状态，完成第一批次部署后，单击详情。



9. 进入应用详情页面。您可看到未开始列表下的Pod为2个，已完成列表下的Pod为2个，表示分批发布第一个批次已完成。单击继续，可发布第二批Pod，单击回滚，此时会回滚到之前的版本。



10. 当发布完成后。单击历史，可以进行历史版本的回滚。

后续操作

您可使用分批发布功能实现应用版本的快速验证，没有流量损失，与蓝绿发布相比更节省资源，目前暂时只支持页面操作，后续会开放yaml文件编辑，支持更复杂的操作。

1.15 Istio管理

1.15.1 概述

Istio是一个提供连接、保护、控制以及观测微服务功能的开放平台。

微服务目前被越来越多的IT企业重视。微服务是将复杂的应用切分为若干服务，每个服务均可以独立开发、部署和伸缩；微服务和容器组合使用，可进一步简化微服务的交付，提升应用的可靠性和可伸缩性。

随着微服务的大量应用，其构成的分布式应用架构在运维、调试、和安全管理等维度变得更加复杂，开发者需要面临更大的挑战，如：服务发现、负载均衡、故障恢复、指标收集和监控，以及A/B测试、灰度发布、蓝绿发布、限流、访问控制、端到端认证等。

Istio应运而生。Istio是一个提供连接、保护、控制以及观测微服务功能的开放平台，提供了一种简单的创建微服务网络的方式，并提供负载均衡、服务间认证以及监控等能力，同时Istio不需要修改服务即可实现以上功能。

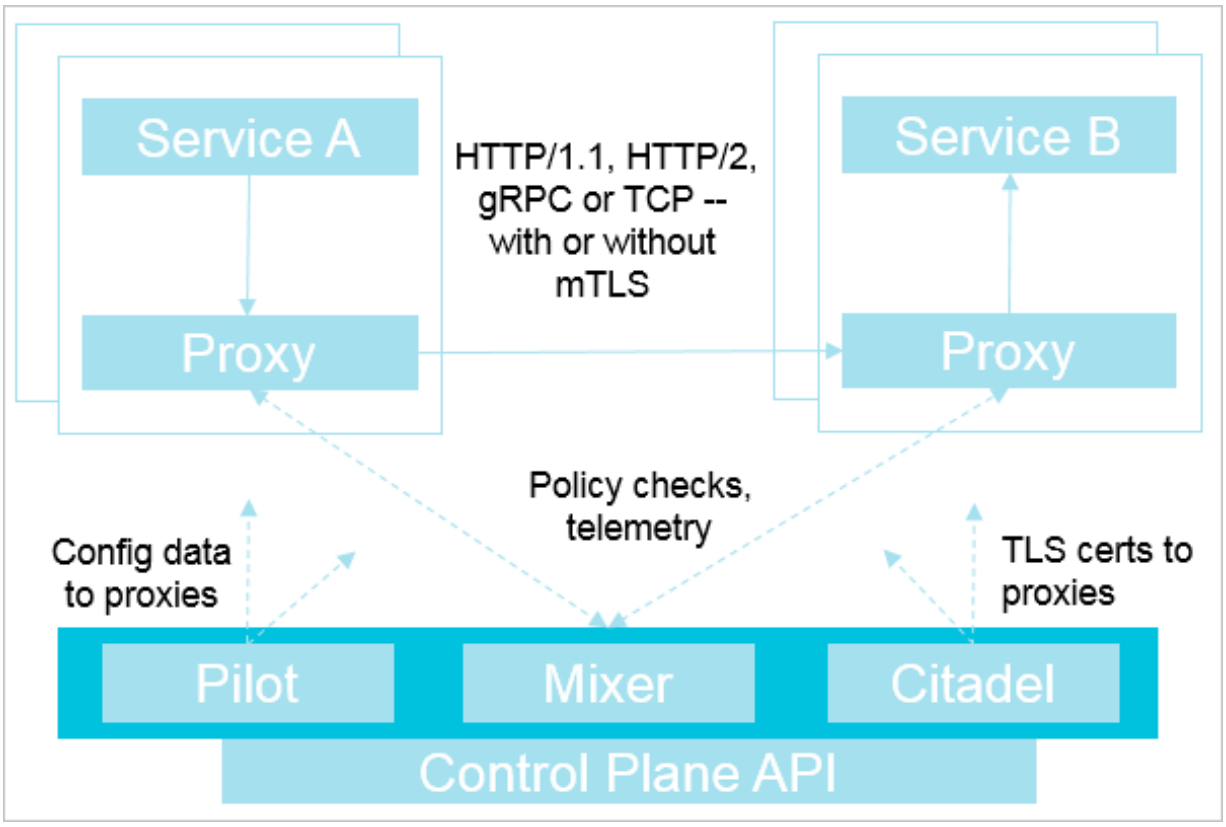
Istio提供如下功能：

- 流量管理：控制服务之间的流量及API调用。增强系统的可靠性。
- 鉴权及安全保护：为网格中的服务提供身份验证，并保护服务的流量。增强系统的安全性。
- 策略执行：控制服务之间的访问策略，且不需要改动服务。
- 可观察性： 获取服务之间的流量分布及调用关系。 快速定位问题。

Istio架构

Istio在逻辑上分为控制层面和数据层面：

- 控制层面：管理代理（默认为Envoy），用于管理流量路由、运行时策略执行等。
- 数据层面：由一系列代理（默认为Envoy）组成，用于管理和控制服务之间的网络通信。



Istio主要由以下组件构成：

- Istio管理器（Pilot）：负责收集和验证配置，并将其传播到各种Istio组件。它从策略执行模块（Mixer）和智能代理（Envoy）中抽取环境特定的实现细节，为他们提供用户服务的抽象表示，独立于底层平台。此外，流量管理规则（即通用4层规则和7层HTTP/gRPC路由规则）可以在运行时通过Pilot进行编程。
- 策略执行模块（Mixer）：负责在服务网格上执行访问控制和使用策略，并从智能代理（Envoy）和其他服务收集遥测数据。依据智能代理（Envoy）提供的属性执行策略。
- Istio安全模块：提供服务间以及用户间的认证，确保在不需要修改服务代码的前提下，增强服务之间的安全性。包括3个组件：
  - 身份识别：当Istio运行在Kubernetes时，根据容器Kubernetes提供的服务账号，识别运行服务的主体。
  - Key管理：提供CA自动化生成和管理key和证书。
  - 通信安全：通过智能代理（Envoy）在客户端和服务端提供通道（tunnel）保证服务的安全性。
- 智能代理（Envoy）：作为一个独立的组件与相关微服务部署在同一个Kubernetes的pod上，并提供一系列的属性给策略执行模块（Mixer）。策略执行模块（Mixer）以此作为执行策略的依据，并发送到监控系统。

1.15.2 部署Istio

为解决微服务的分布式应用架构在运维、调试、和安全管理等维度存在的问题，可通过部署Istio创建微服务网络，并提供负载均衡、服务间认证以及监控等能力，同时Istio不需要修改服务即可实现以上功能。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 请以主账号登录，或赋予子账号足够的权限，如自定义角色中的`cluster-admin`，参考子账号*Kubernetes*应用权限配置指导。

背景信息

- 阿里云容器服务Kubernetes 1.10.4及之后版本支持部署Istio，如果是1.10.4之前的版本，请先升级到1.10.4或之后版本。
- 一个集群中，worker节点数量需要大于等于3个，保证资源充足可用。

操作步骤

通过集群界面部署Istio

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中的集群，进入集群列表页面。
3. 选择所需的集群并单击操作列更多 > 部署Istio。

4. 根据如下信息，部署Istio：

配置	说明
集群	部署Istio的目标集群。
命名空间	部署Istio的命名空间。
发布名称	发布的Istio名称。
启用 Prometheus 度量日志收集	是否启用Prometheus 收集度量日志。默认情况下启用。
启用 Grafana 度量展示	是否启用Grafana 展示指标的度量数据。默认情况下启用。
启用 Sidecar 自动注入	是否启用 Sidecar 进行容器的自动注入。默认情况下启用。
启用 Kiali 可视化服务网格	是否启用 Kiali 可视化服务网格。默认情况下不启用。 • 用户名：指定用户名称。默认情况下是admin。 • 密码：指定密码。默认情况下是admin。
启用阿里云日志服务 SLS 及 Jaeger	是否启用阿里云的日志服务SLS 及 Jaeger。默认情况下不启用。 服务入口地址（Endpoint）：根据配置的日志服务所在region选择对应的地址。具体请参见服务入口 项目名称（Project）：采集日志所在的项目名称。

配置	说明
	日志库名称（Logstore）：采集日志所在的日志库名称。 AccessKeyID：访问日志服务时，所使用的访问秘钥ID。 **说明：** 需要选择有权限访问日志服务的AccessKeyID。 AccessKeySecret：访问日志服务时，所使用的访问秘钥Secret。
Pilot设置	跟踪采样百分比（0-100）：默认取值为1。
控制Egress流量	• 直接放行对外访问的地址范围：Istio 服务网格内的服务可以直接对外访问的地址范围。默认情况下为空，使用英文半角逗号分隔。 • 拦截对外访问的地址范围：拦截直接对外访问的地址范围。默认情况下，已包含集群 Pod CIDR 与 Service CIDR，使用英文半角逗号分隔。 **说明：** 直接放行对外访问的地址范围的优先级高于拦截对外访问的地址范围。 例如您将同一个IP地址同时配置在直接放行对外访问的地址与拦截对外访问的地址时，您仍可以直接访问此地址，即直接放行对外访问的地址范围生效。

5. 单击部署 **Istio**，启动部署。

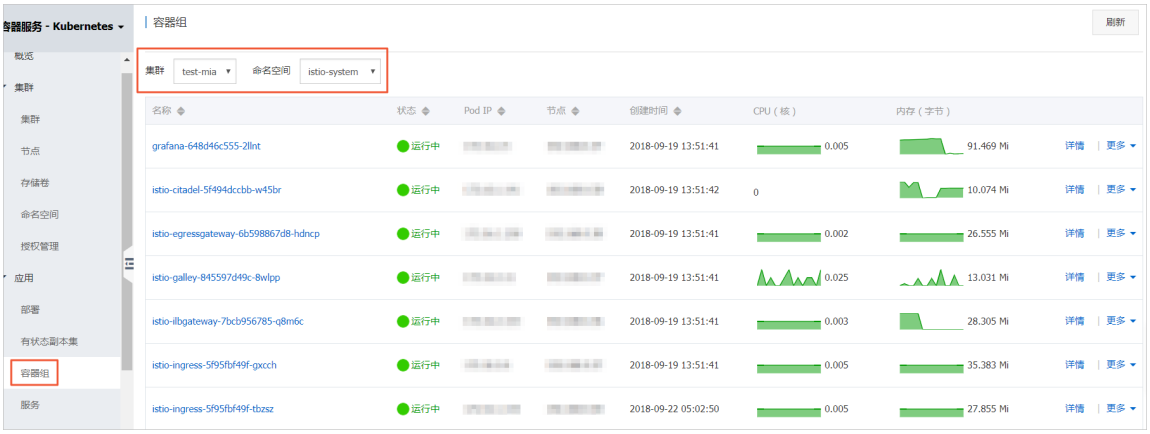
在部署页面下方，可实时查看部署进展及状态。

预期结果

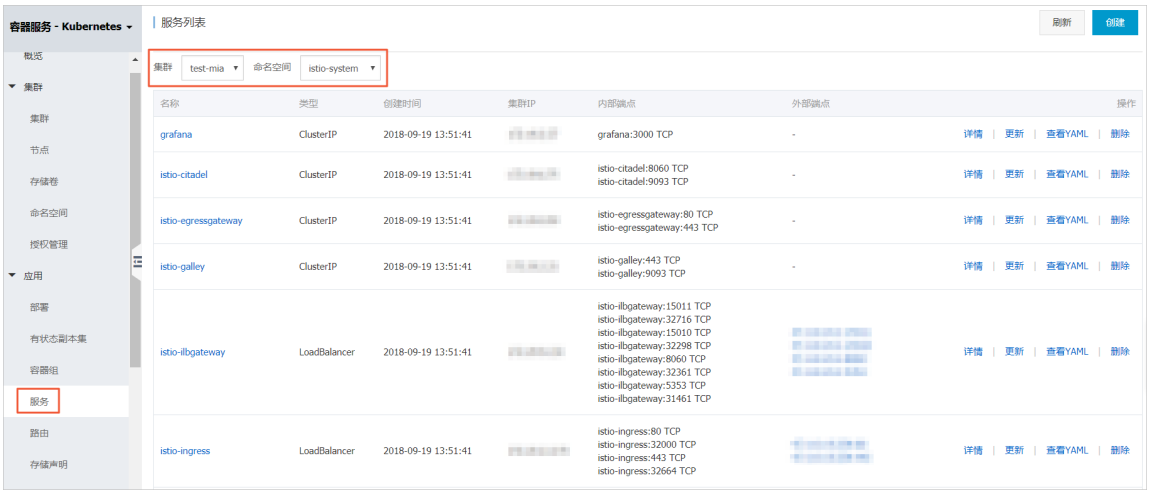
可通过以下方法查看部署是否成功：

- 在部署 **Istio**页面下方，部署 **Istio**变为已部署。

- • 单击左侧导航栏应用 > 容器组，进入容器组页面。
  - 选择部署Istio的集群及命名空间，可查看到已经部署Istio的相关容器组。

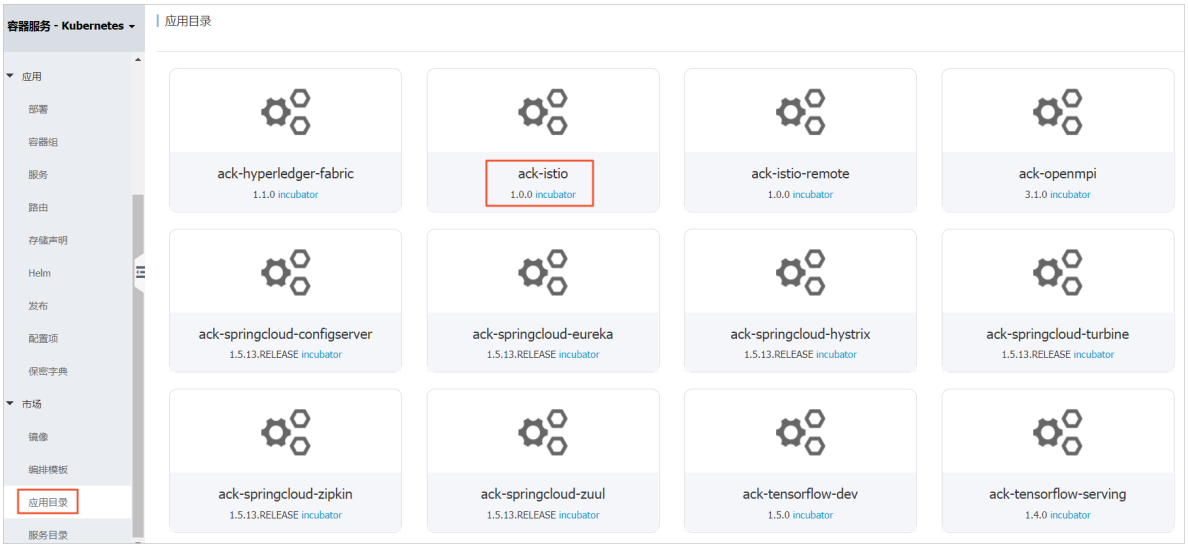


- • 单击左侧导航栏应用 > 服务，进入服务列表页面。
  - 选择部署Istio的集群及命名空间，可查看到已经部署Istio相关服务所提供的访问地址。

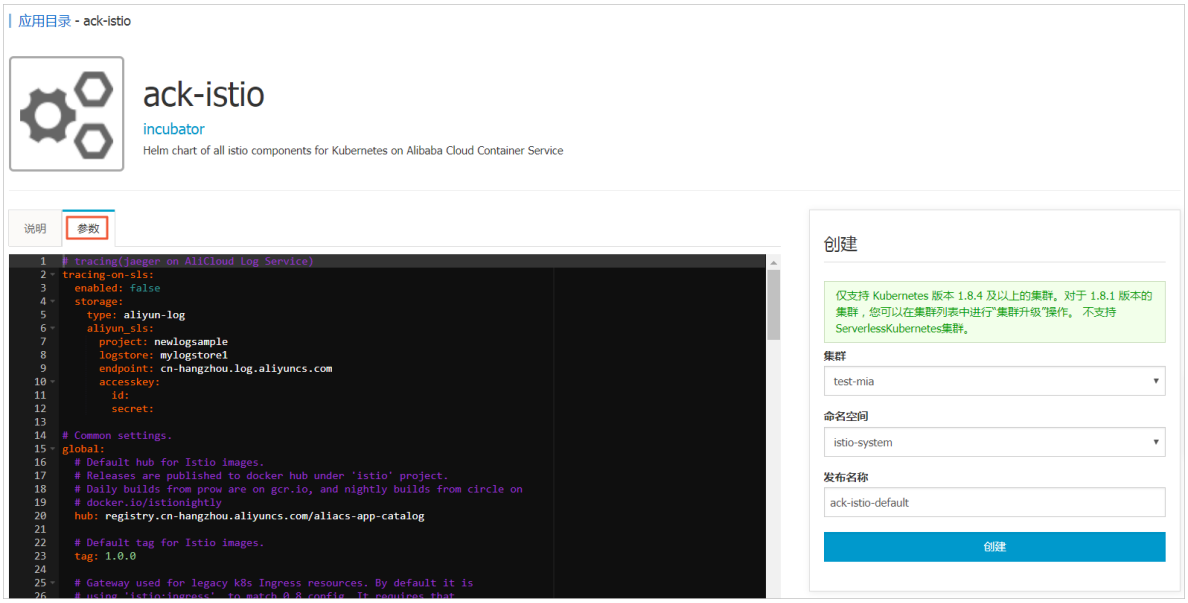


通过应用目录部署Istio

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中的**市场 > 应用目录**，进入应用目录页面。



3. 单击**ack-istio**，进入应用目录 **- ack-istio**页面。
4. 单击参数页签，进行参数配置。



说明：

- 通用参数的含义、取值及默认情况，可参考说明页签**Configuration**字段。
- 也可以针对特定参数进行定制化配置，例如：是否启用**grafana**、**prometheus**、**tracing**、**weave-scope**以及**kiali**等，可参考：

```
#
# addons configuration
#
```

```
grafana:
enabled: true
replicaCount: 1
image: istio-grafana
service:
name: http
type: ClusterIP
externalPort: 3000
internalPort: 3000
....
prometheus:
enabled: true
replicaCount: 1
image:
repository: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/
istio-prometheus
tag: latest
....
tracing:
enabled: true
jaeger:
enabled: true
....
weave-scope:
enabled: true
global:
# global.image: the image that will be used for this release
image:
repository: weaveworks/scope
tag: "1.9.0"
# global.image.pullPolicy: must be Always, IfNotPresent, or Never
pullPolicy: "IfNotPresent"
....
kiali:
enabled: true
replicaCount: 1
image:
repository: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/
istio-kiali
tag: dev
```

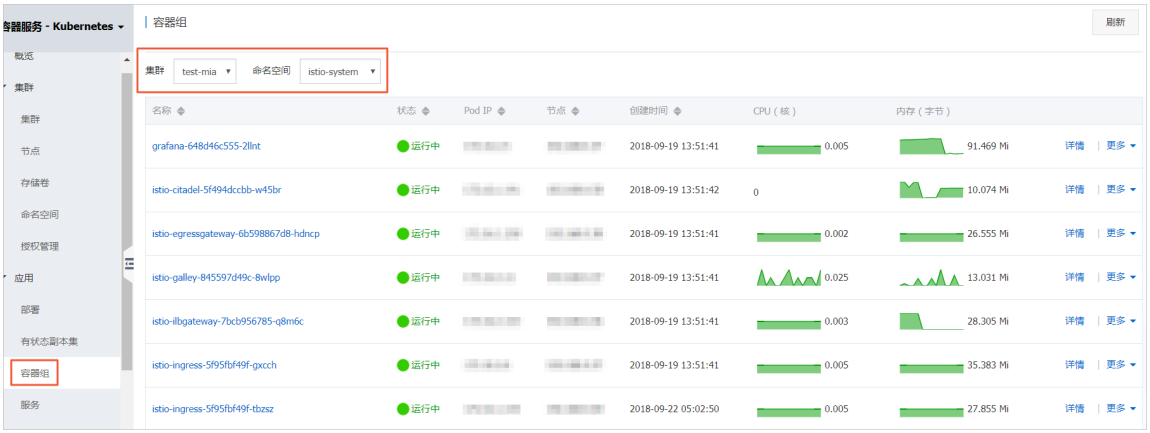
5. 在右侧创建区域，填写以下基本信息：

配置	说明
集群	部署Istio的目标集群。
命名空间	部署Istio的命名空间。默认情况下为 default。
发布名称	发布的Istio名称。

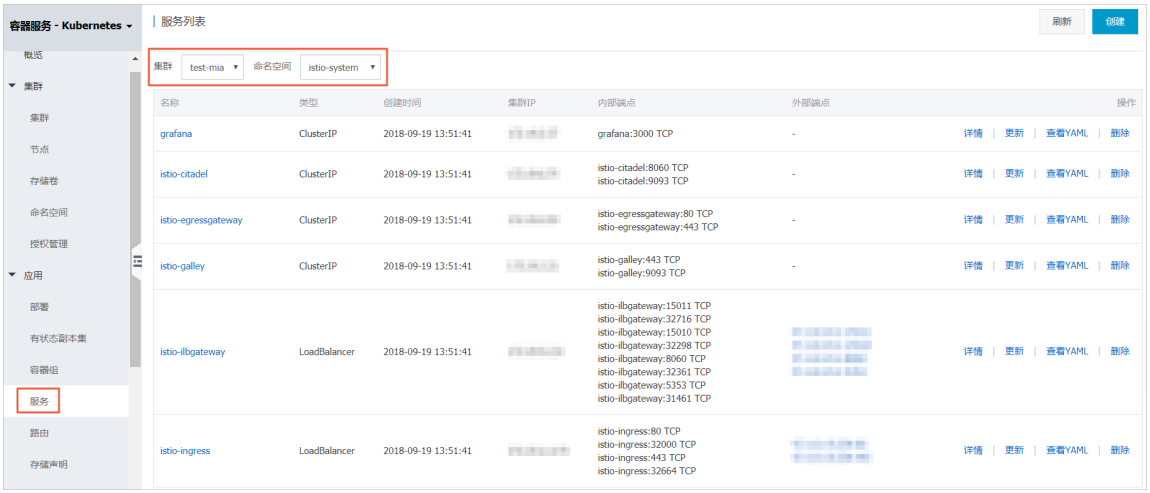
6. 单击创建，启动部署。

预期结果

- • 单击左侧导航栏应用 > 容器组，进入容器组页面。
  - 选择部署Istio的集群及命名空间，可查看到已经部署Istio的相关容器组。



- • 单击左侧导航栏应用 > 服务，进入服务列表页面。
  - 选择部署Istio的集群及命名空间，可查看到已经部署Istio相关服务所提供的访问地址。



1.15.3 更新Istio

您可以通过更新操作，编辑已部署的Istio。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已成功创建一个**Istio**，参见部署*Istio*。

操作步骤

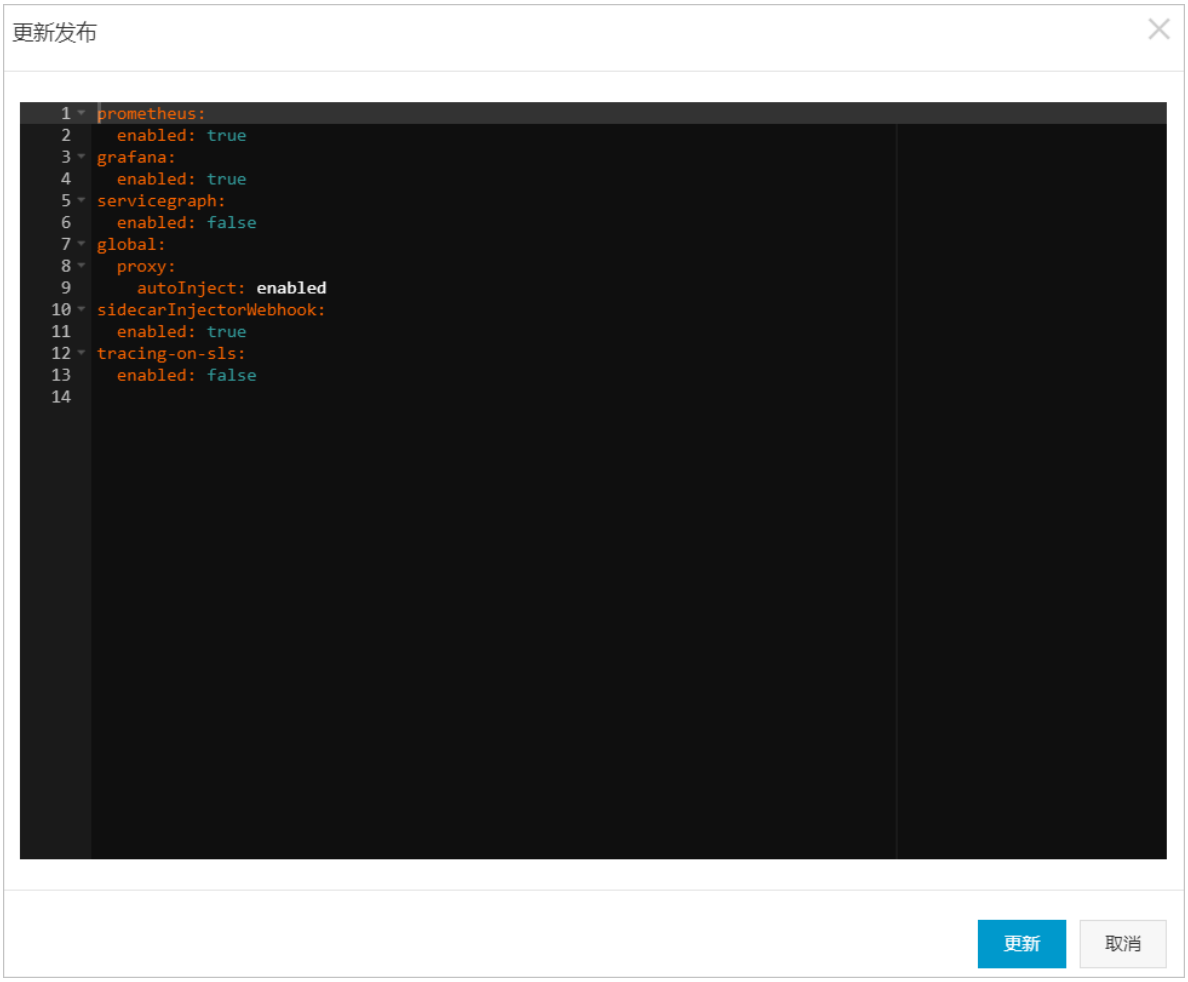
1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的应用 > **Helm**，进入发布列表页面。
3. 选择所需的集群，选择待更新的Istio，单击操作列的更新。

说明：

- 通过集群界面部署的Istio，发布名称为istio，更新的内容与部署时的选项一致。
- 通过应用目录部署的Istio，发布名称为用户创建时指定的名称，更新的内容与部署时的参数一致。

4. 在弹出的对话框中，对Istio的参数进行编辑后，单击更新。

   此处以更新通过集群页面部署的Istio为例：



预期结果

可通过以下两种方式，查看更新后的内容：

- 更新完成后，页面自动跳转到**发布列表**页面，在**资源**页签，可以查看更新的内容。

- 在 Kubernetes 菜单下，单击左侧导航栏的应用 > 容器组，进入容器组页面，选择目标集群和目标空间进行查看。

1.15.4 删除Istio

您可以通过删除操作，删除已部署的Istio。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建*Kubernetes*集群。
- 您已成功创建一个**Istio**，参见部署*Istio*。

操作步骤

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的应用 > **Helm**，进入发布列表页面。
3. 选择所需的集群及待删除的Istio，单击操作列的删除。

4. 在弹出的对话框中，单击确定。

说明：

- 不勾选清除发布记录：
  - 发布记录不会被删除：

— 此Istio的名称不可被再次使用。

通过集群界面重新部署Istio时，显示已完成。

通过应用目录部署Istio时，提示：已存在同名的部署或资源，请修改名称。

- 勾选清除发布记录，会同时删除所有发布记录，且此Istio的名称可被再次使用。

建议保持默认状态，勾选清除发布记录。

预期结果

返回发布列表页面，可以看到该Istio已被删除。

1.16 模板管理

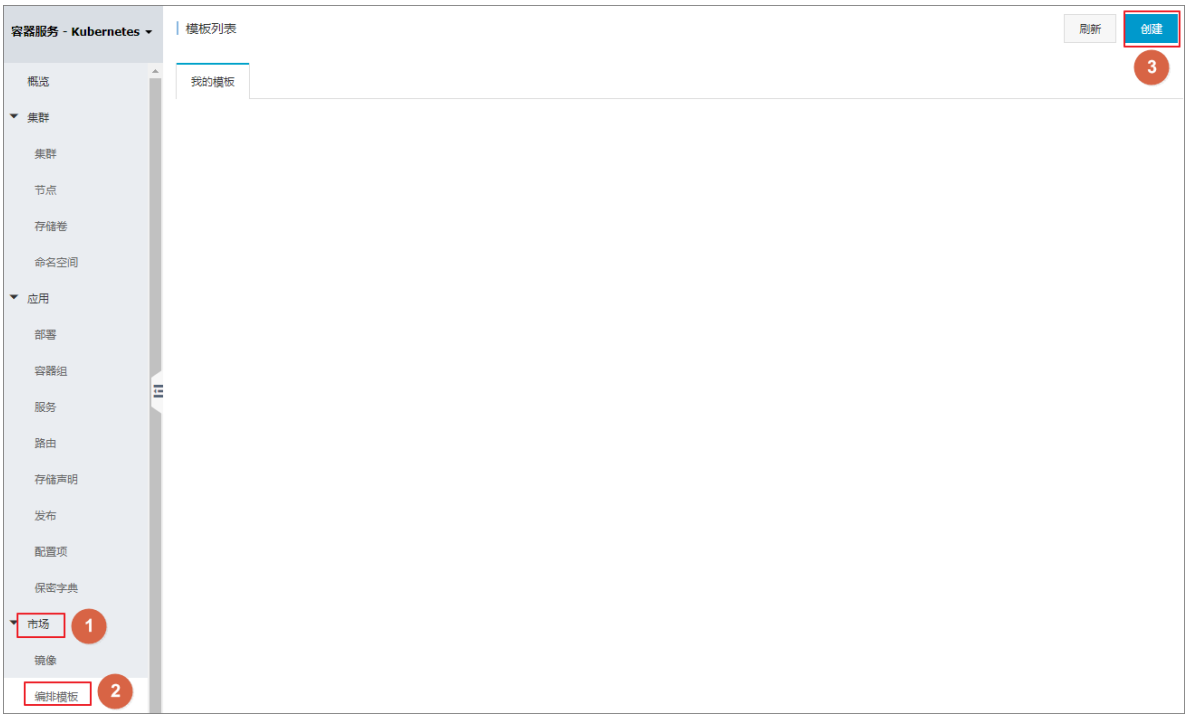
1.16.1 创建编排模板

通过容器服务管理控制台，您可通过多种方式创建编排模板。

操作步骤

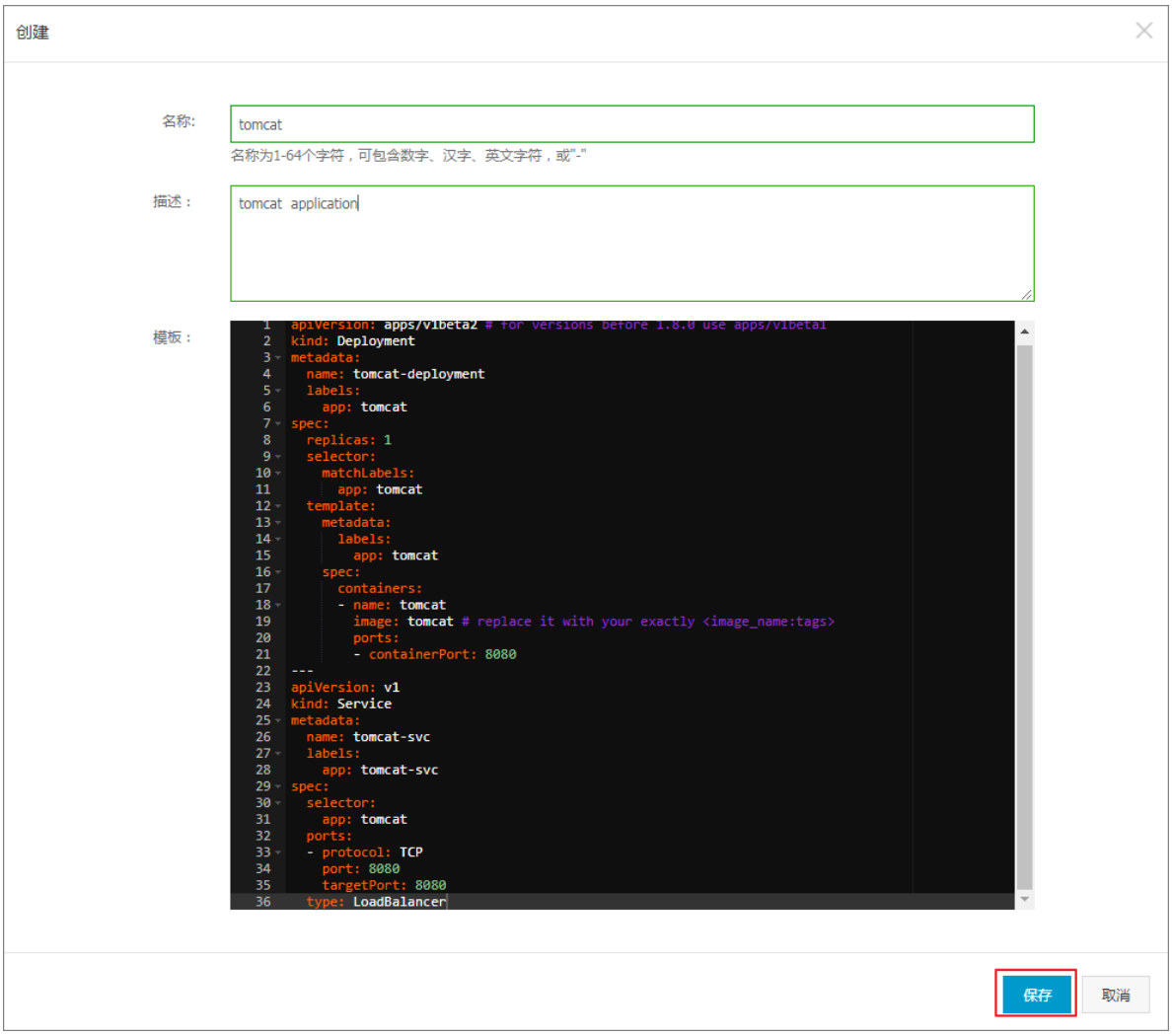
1. 登录容器服务管理控制台。

2. 在Kubernetes菜单下，单击左侧导航栏中的**市场 > 编排模板**，单击右上角**创建**。



3. 在弹出的对话框中，配置编排模板，最后单击**保存**。本例构建一个tomcat应用的模板，包含一个deployment和service。

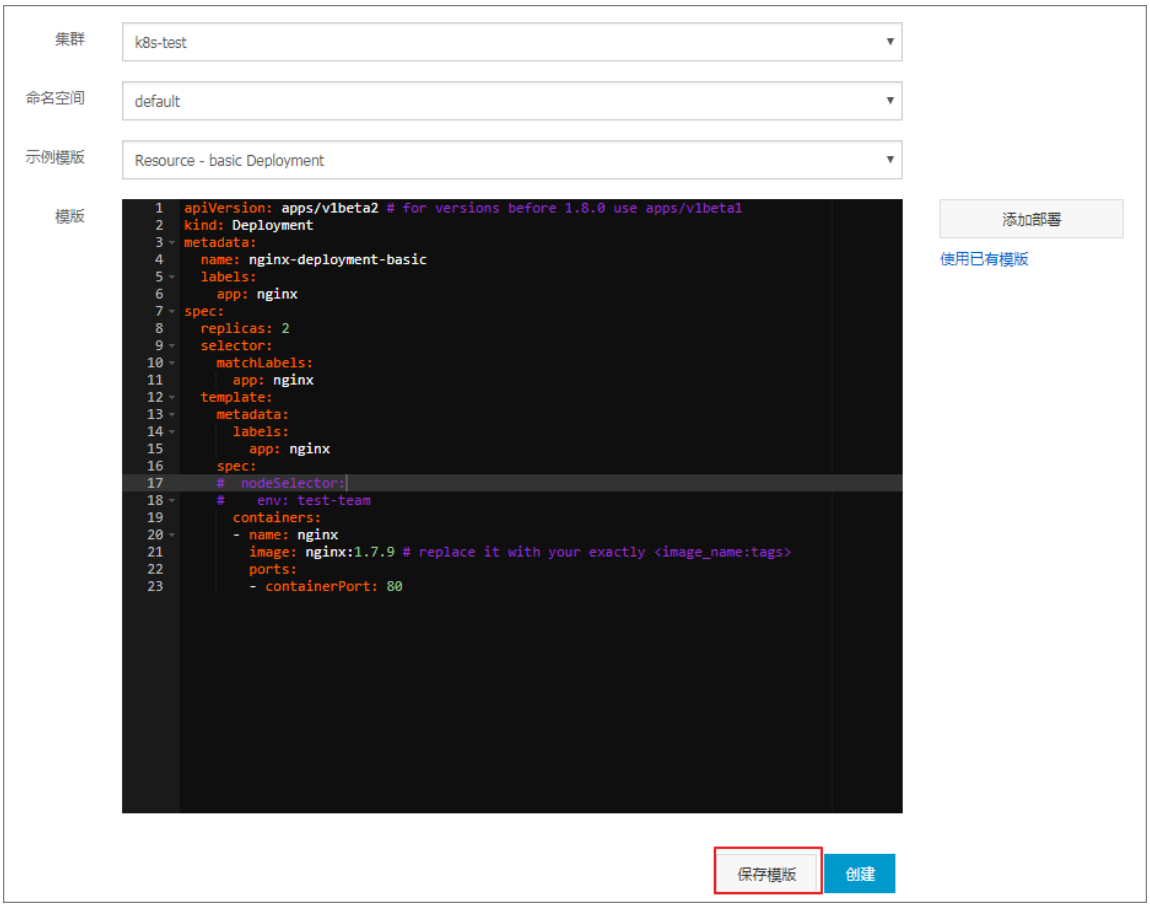
   - **名称**：设置该模板的名称。
   - **描述**：输入对该模板的描述，可不配置。
   - **模板**：配置符合Kubernetes Yaml语法规则的编排模板，可包含多个资源对象，以`---`进行分割。



4. 创建完毕后，默认进入模板列表页面，您可在我的模板下看到该模板。

5. （可选）您也可单击左侧导航栏中的应用 > 部署，单击使用模板部署，进入使用模版创建页面，以容器服务内置的编排模板为基础，保存为自定义模板。

a) 选择一个内置模板，然后单击保存模板。



b) 在弹出的对话框中，配置模板信息，包括名称、描述和模板。完成后，单击保存。

> **说明：**
> 您可在内置模板的基础上进行修改。

c) 单击左侧导航栏**市场 > 编排模板**，您可看到该模板出现在**我的模板**下。

后续操作

您可使用**我的模板**下的编排模板，快速创建应用。

1.16.2 更新编排模板

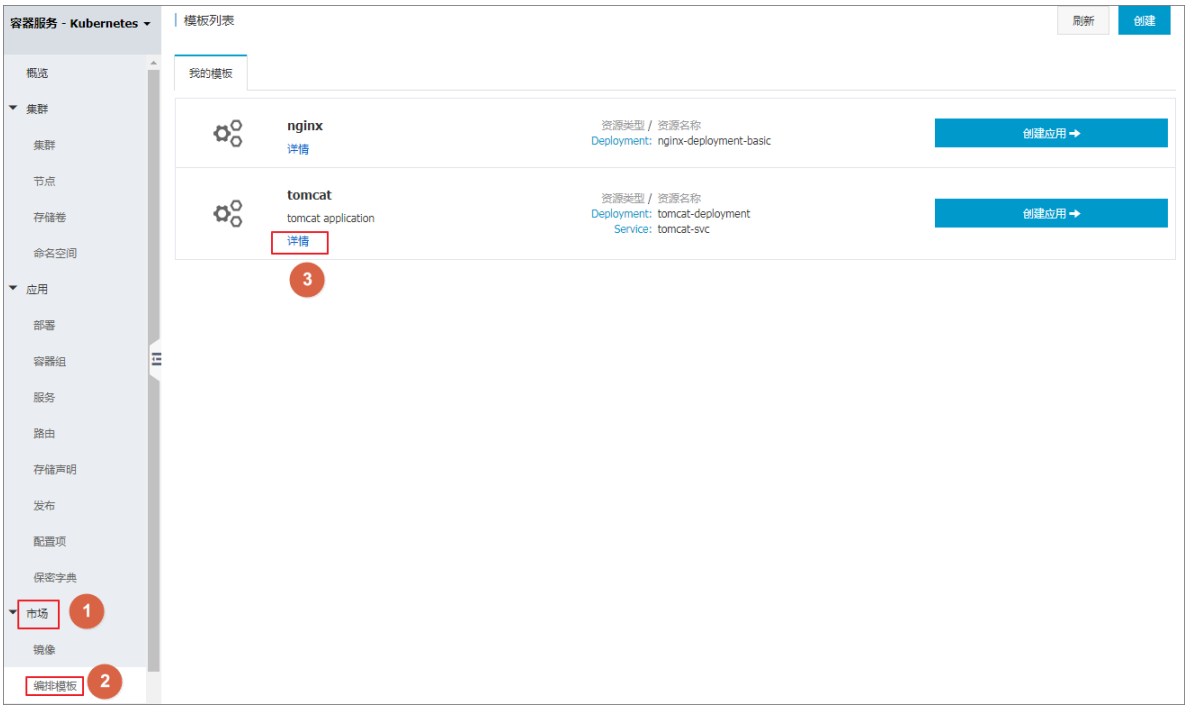
您可对已有的编排模板进行编辑，从而更新编排模板。

前提条件

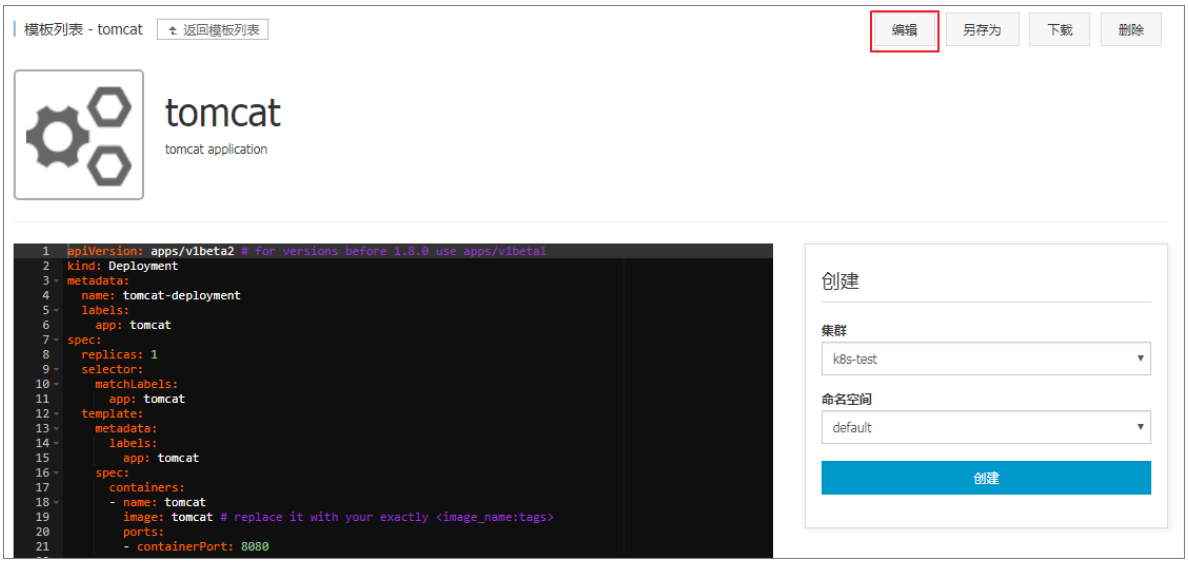
您已经创建一个编排模板，参见创建编排模板。

操作步骤

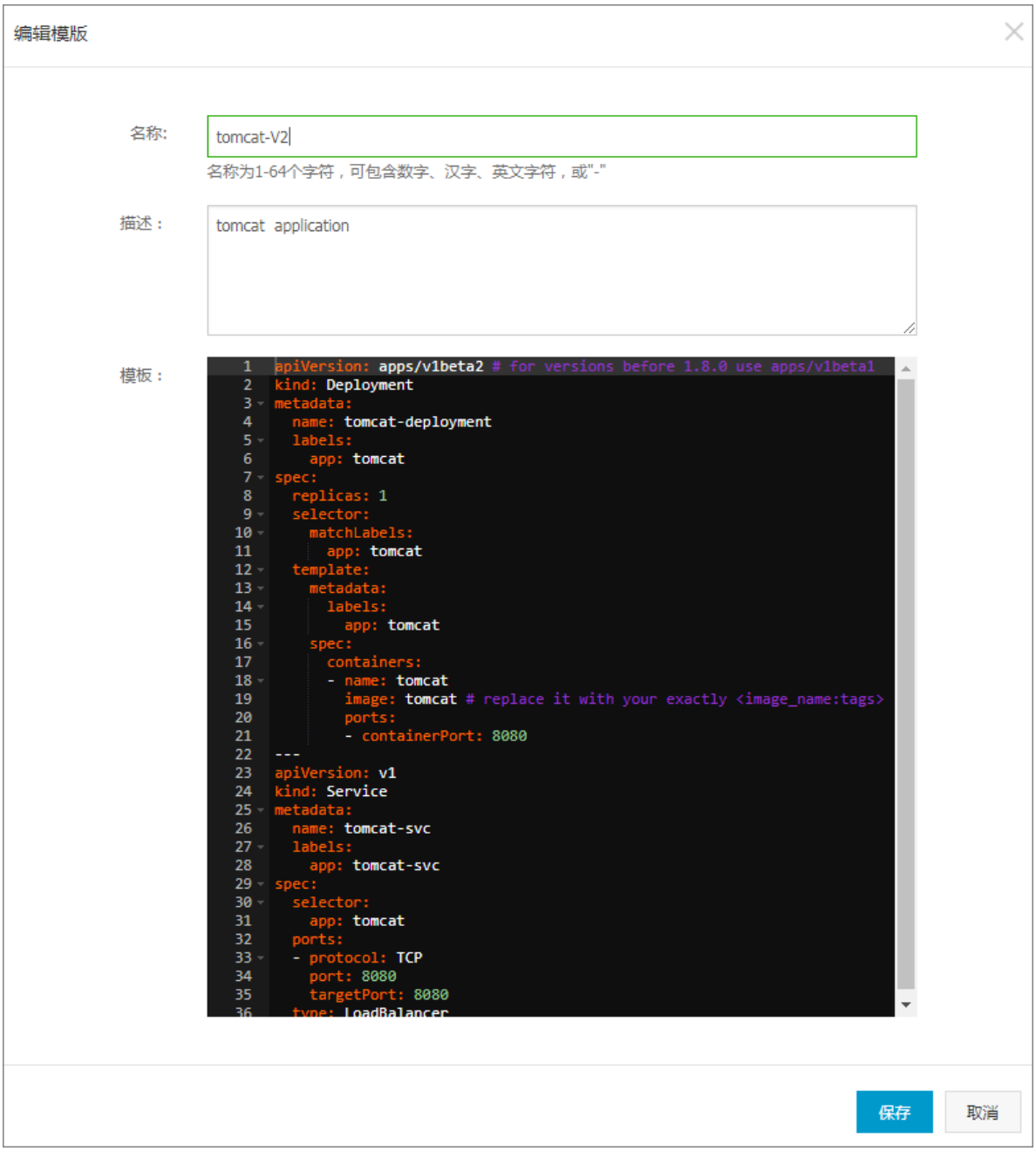
1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的**市场 > 编排模板**，进入**模板列表**页面，在**我的模板**下，您可看到已有的编排模板。
3. 选择所需的模板，单击**详情**，进入模板详情页面。



4. 在该模板的详情页中，单击右上角**编辑**。



5. 弹出编辑模板对话框，在该对话框中编辑名称、描述和模板，完成配置后，单击保存。



6. 返回模板列表页面，在我的模板下，您可看到该模板已发生变化。

1.16.3 另存编排模板

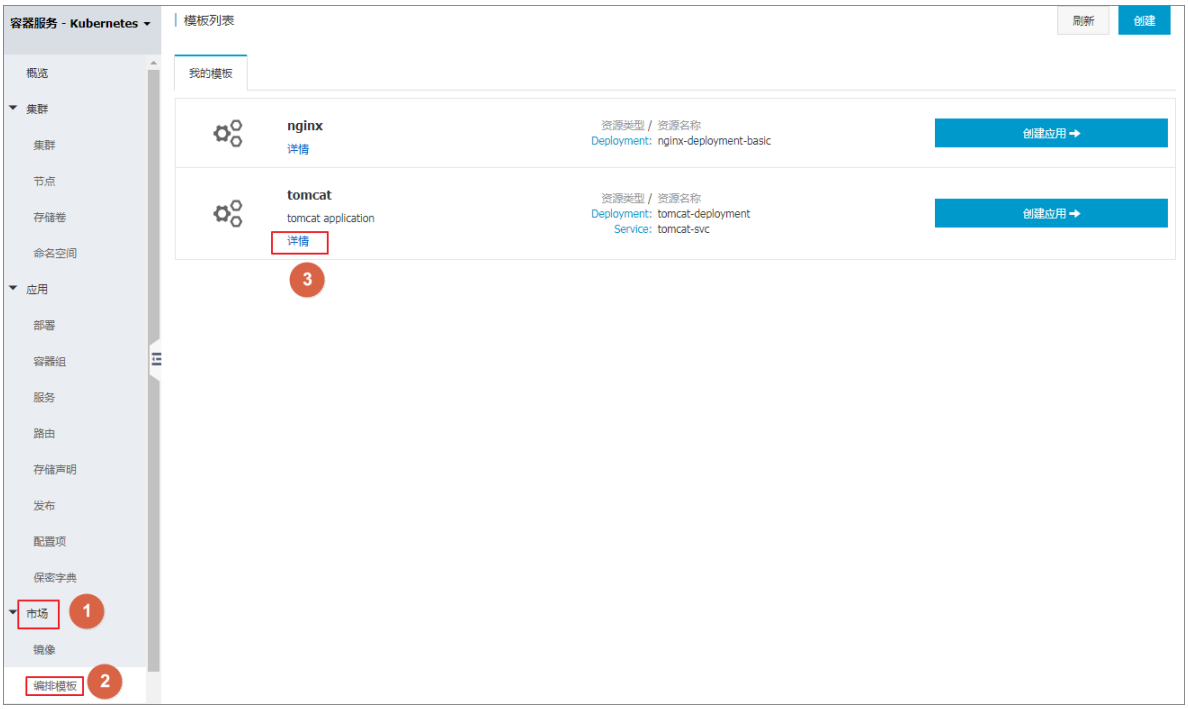
您可对已有模板进行另存，保存为一个新的编排模板。

前提条件

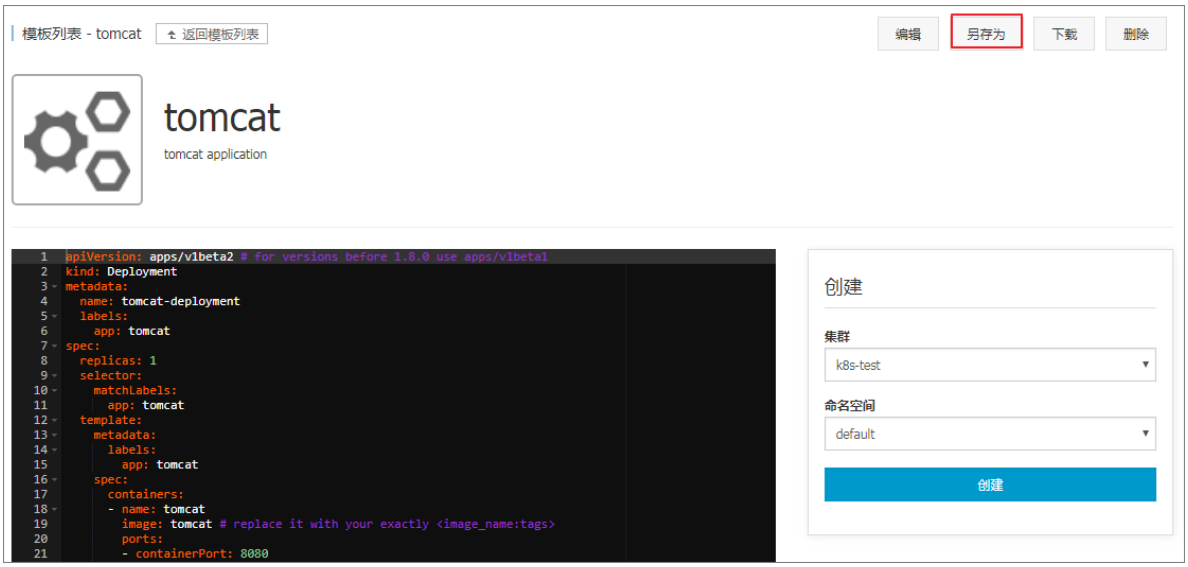
您已经创建一个编排模板，参见创建编排模板。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的**市场 > 编排模板**，进入**模板列表**页面，在**我的模板**下，您可看到已有的编排模板。
3. 选择所需的模板，单击**详情**，进入模板详情页面。



4. 在该模板的详情页中，您可修改该模板，然后单击右上角**另存为**。



5. 弹出对话框，在该对话框中配置模板名称，然后单击确定。

6. 返回模板列表页面，在我的模板下，您可看到另存的模板出现在该列表下。

1.16.4 下载编排模板

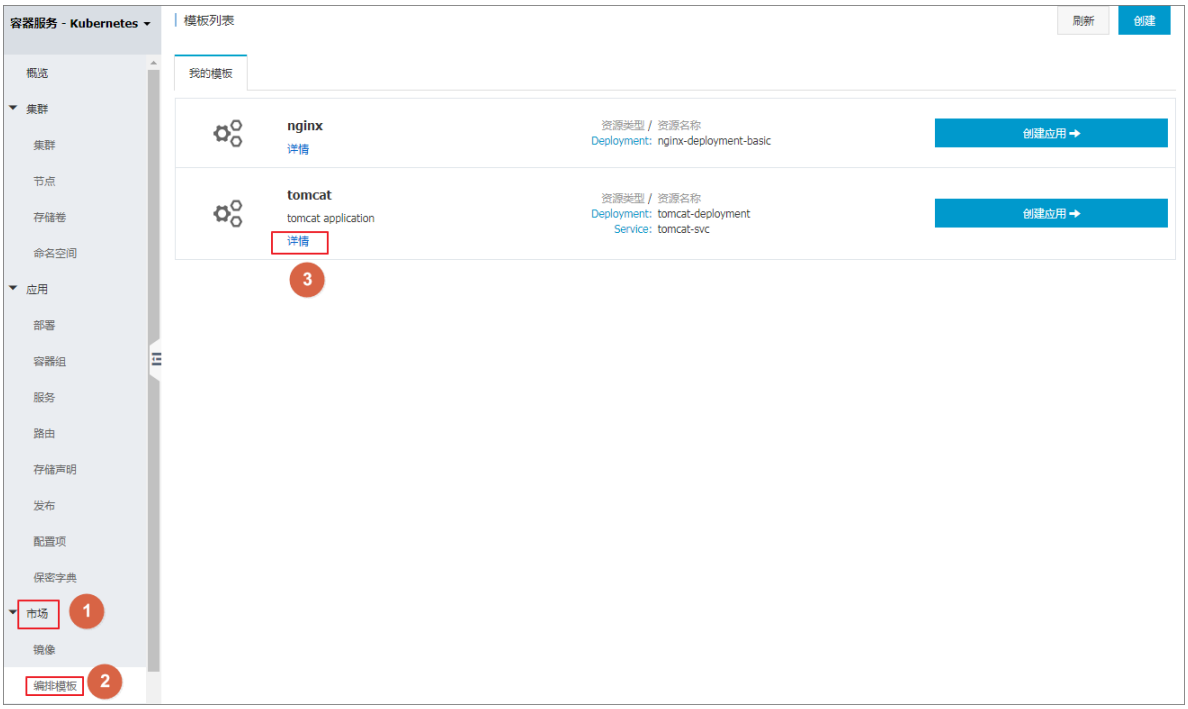
您可下载已有的编排模板。

前提条件

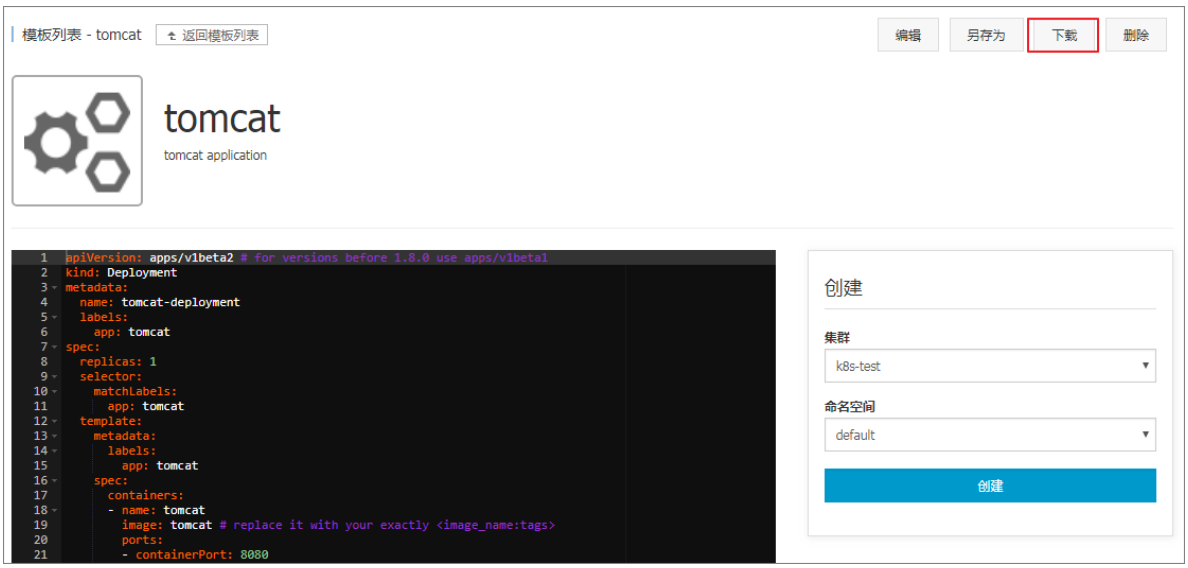
您已经创建一个编排模板，参见创建编排模板。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的市场 > 编排模板，进入模板列表页面，在我的模板下，您可看到已有的编排模板。
3. 选择所需的模板，单击详情，进入模板详情页面。



4. 在该模板的详情页中，您可单击右上角下载，会立即下载后缀为yml格式的模板文件。



1.16.5 删除编排模板

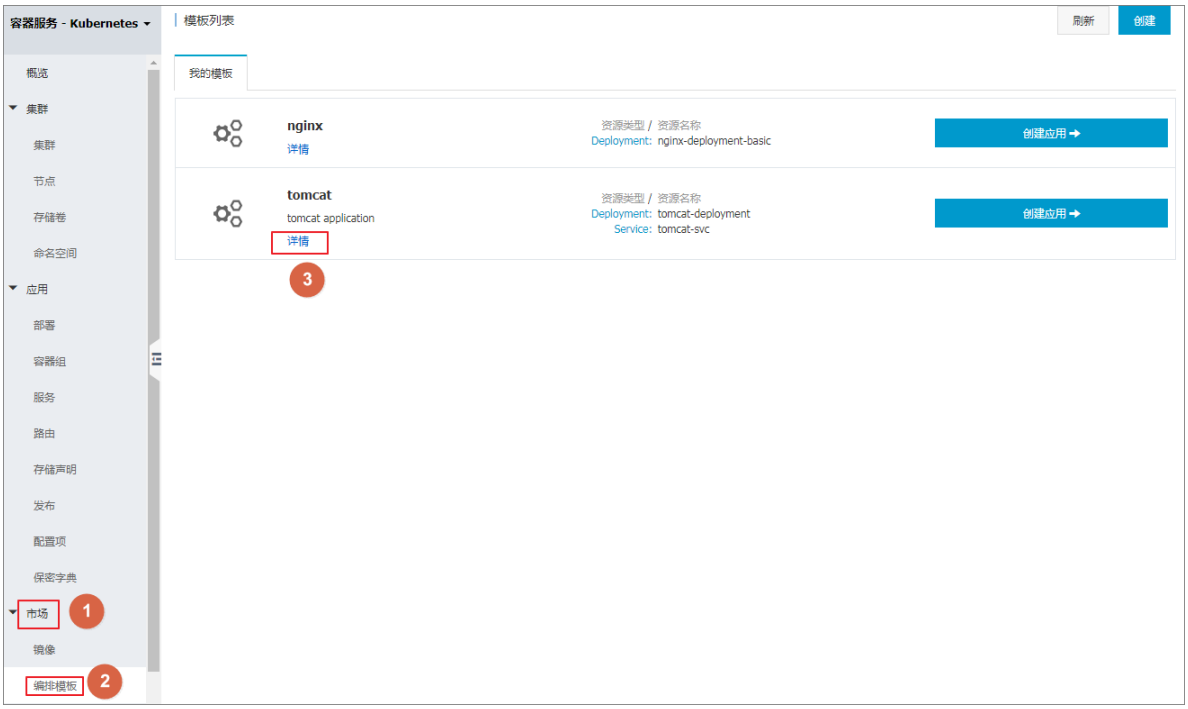
您可删除不再需要的编排模板。

前提条件

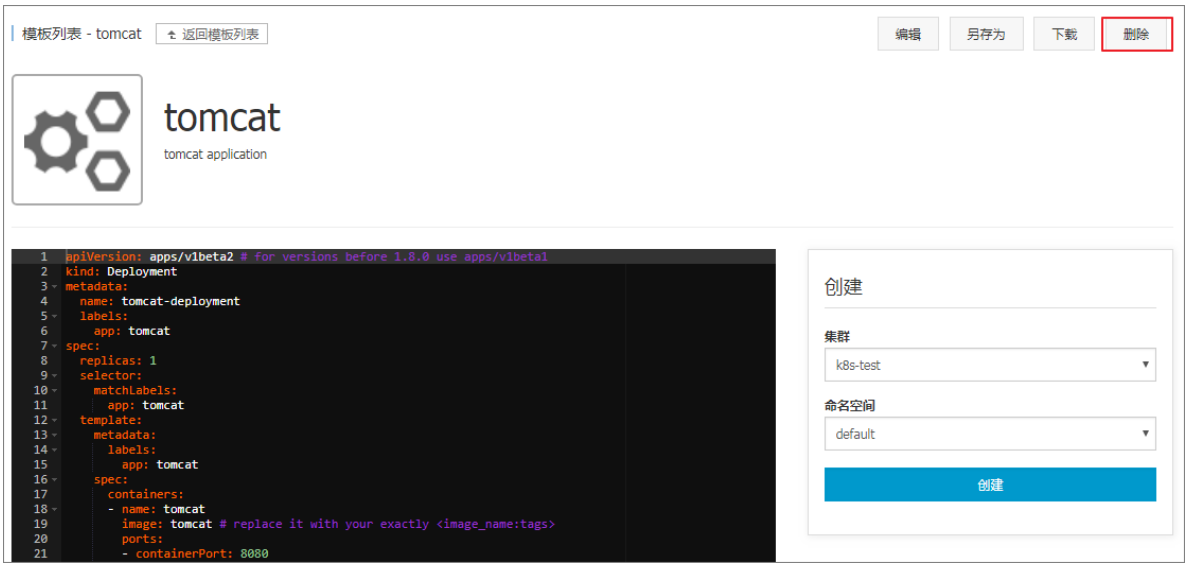
您已经创建一个编排模板，参见创建编排模板。

操作步骤

1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的**市场 > 编排模板**，进入**模板列表**页面，在**我的模板**下，您可看到已有的编排模板。
3. 选择所需的模板，单击**详情**，进入模板详情页面。



4. 在该模板的详情页中，您可单击右上角**删除**。



5. 在弹出的确认对话框中，单击确定。

1.17 应用目录管理

1.17.1 应用目录概述

微服务是容器时代的主题，应用微服务化给部署和管理带来极大的挑战。通过将庞大的单体应用拆分成一个个微服务，从而使各个微服务可被独立部署和扩展，实现敏捷开发和快速迭代。虽然微服务带来了很大的好处，但同时，由于应用拆分成许多组件，对应着庞大数量的微服务，开发者不得不面对这些微服务的管理问题，如资源管理、版本管理、配置管理等。

针对 Kubernetes 编排下微服务管理问题，阿里云容器服务引入 Helm 开源项目并进行集成，帮助简化部署和管理 Kubernetes 应用。

Helm 是 Kubernetes 服务编排领域的开源子项目，是 Kubernetes 应用的一个包管理工具， Helm 通过软件打包的形式，支持发布的版本管理和控制，简化了 Kubernetes 应用部署和管理的复杂性。

阿里云应用目录功能

阿里云容器服务应用目录功能集成了 Helm，提供了 Helm 的相关功能，并进行了相关功能扩展，如提供图形化界面、阿里云官方 Repository 等。

应用目录首页 chart 列表的信息包含：

- chart 名称： 一个 Helm 包，对应一个目标应用，其中包含了运行一个应用所需要的镜像、依赖和资源定义等。

- 版本：chart 的版本号。
- Repository：用于发布和存储 Chart 的仓库，例如官方仓库 stable、incubator 等。

各个 chart 详情页包含的信息不尽相同，例如，可能包含：

- chart 简介
- chart 详细信息
- chart 安装到集群的前提条件，如预先配置持久化存储卷(pv) 。
- chart 安装命令
- chart 卸载命令
- chart 参数配置项

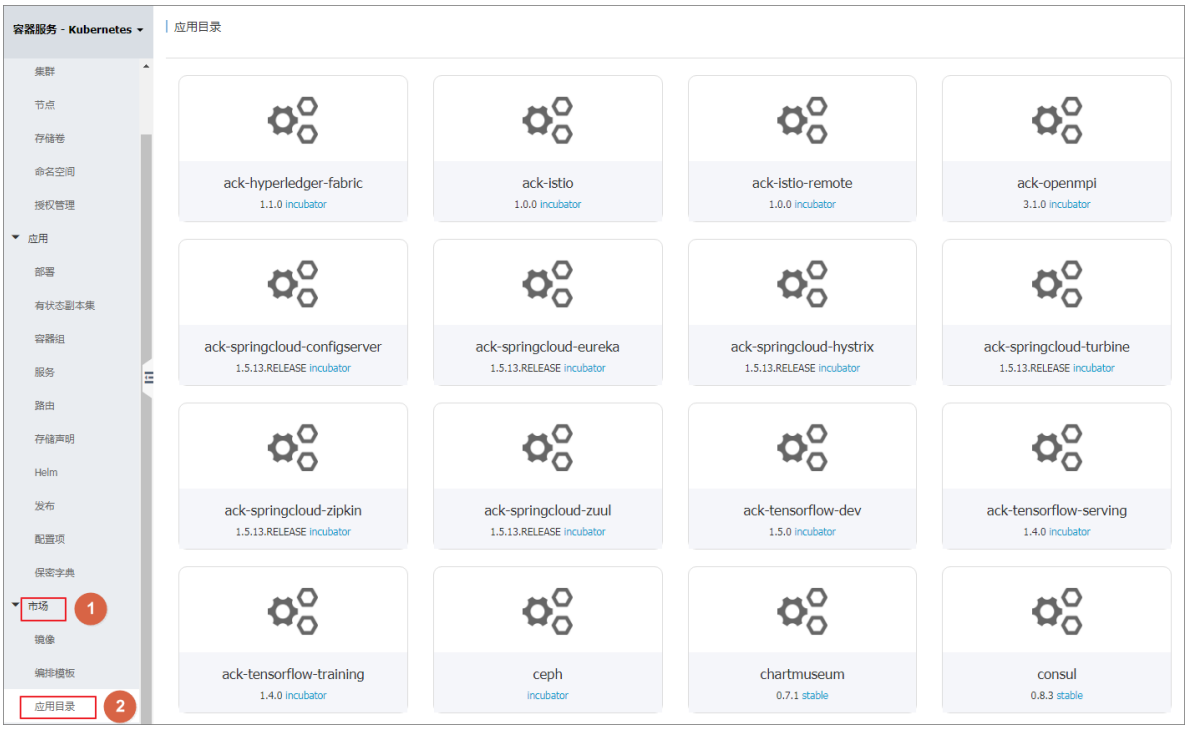
目前，您可以通过 helm 工具部署和管理应用目录中的 chart，具体请参见利用 *Helm* 简化应用部署。

1.17.2 查看应用目录列表

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**市场** > 应用目录，进入应用目录列表。

   您可以查看该页面下 chart 列表，每个 chart 对应一个目标应用，包含一些基本信息，如应用名称、版本号和来源仓库等信息。



后续操作

您可进入单个 chart，了解具体的 chart 信息，可根据相关信息，利用 Helm 工具部署应用，具体请参见利用 *Helm* 简化应用部署。

1.18 服务目录管理

1.18.1 概述

云平台上运行的应用需要使用一些基础服务，如数据库、应用服务器等通用的基础软件。譬如一个 wordpress 应用，作为一个 web 应用，后端需要一个数据库服务，如 MariaDB。传统方式是在 wordpress 应用的编排中也创建应用依赖的 MariaDB 服务，并与 Web 应用进行集成。这种云上应用开发的方式，就需要开发者花费精力解决所依赖的基础设施软件的部署和配置，增加应用托管和迁移的成本。

阿里云容器服务支持并集成了服务目录的功能，该功能旨在接入和管理 Service Broker，使 kubernetes 上运行的应用可以使用 service broker 所代理的托管服务。服务目录功能将支持一系列基础设施软件，应用开发者可以不用关心这些软件的可用性和伸缩能力，也不用对其进行管理，开发者可以简单的将其作为服务使用，只需专注开发核心的应用程序。

服务目录通过 Kubernetes 的 Open Service Broker API 与 Service Broker 进行通信，并作为 Kubernetes API Server 的中介，以便协商首要规定（initial provisioning）并获取应用程序使用托管服务的必要凭据。关于服务目录的具体实现原理，请参考 *service catalog*。

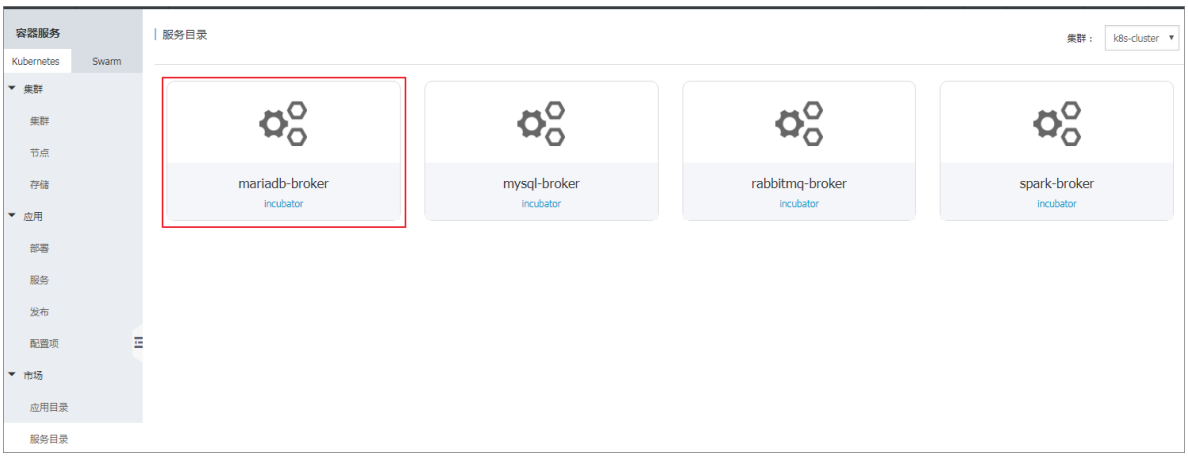
1.18.2 开通服务目录

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的**市场 > 服务目录**，在右上角选择目标集群。
3. 若您还未部署服务目录功能，界面上会提示您先进行安装。

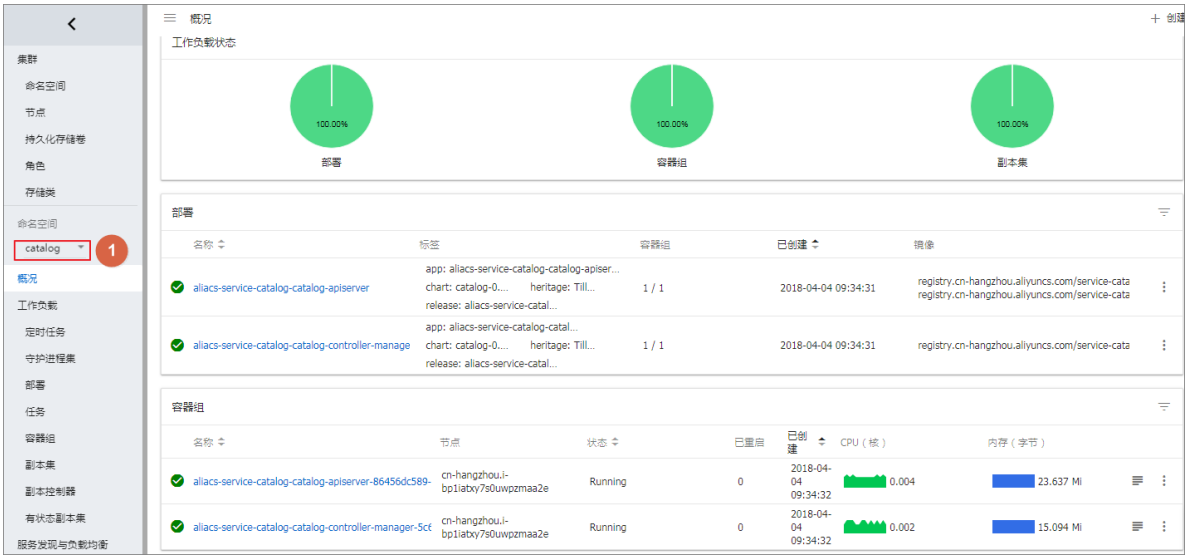
4. 安装完成后，服务目录页面中会显示默认安装的 Service Broker，您可以进入 mariadb-broker 了解详情。

> **说明：**
> 服务目录具体实现为一个扩展 API server 和一个 controller，阿里云容器服务安装服务目录功能后，会创建一个 catalog 命名空间。



5. 在左侧导航栏中单击集群，单击目标集群右侧的控制台。

6. 进入 Kubernetes Dashboard 页面，在命名空间中选择**`catalog`**，可以看到该命名空间下安装了 catalog apiserver 和 controller 相关的资源对象。



后续操作

至此，您已经成功开通服务目录功能。接下来可以通过服务目录下的 Service Broker 来创建托管服务的实例，并将其应用到您自己的应用程序中。