阿里云
容器服务Kubernetes版

快速入门

文档版本：20191219

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. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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。
	该类警示信息可能会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，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意:** 权重设置为0，该服务器不会再接受新请求。
	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。	单击设置 > 网络 > 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击**确定**。
`Courier字体`	命令。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
*## *	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。	`switch {active|stand}`

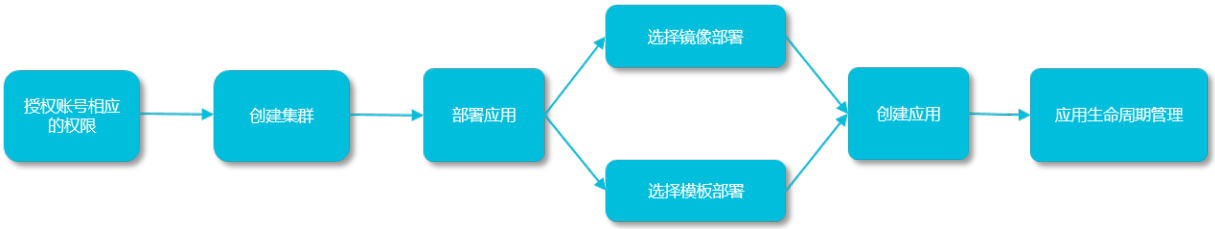
目录

1 使用流程

本文主要为您介绍完整的容器服务使用流程。

背景信息

完整的容器服务使用流程包含以下步骤。



操作步骤

1. 授予账号相应的角色。

   详细信息参见#unique_4。

2. 创建集群。

   - 如果您需要创建专有版Kubernetes（Dedicated Kubernetes），请参见#unique_5。
   - 如果您需要创建托管版Kubernetes（Managed Kubernetes），请参见#unique_6。
   - 如果您需要创建Serverless Kubernetes，请参见#unique_7。

3. 通过镜像或编排模板部署应用。

   您可以使用已有的镜像或编排模板部署应用。

   - 如果您选择镜像部署，请参见**镜像创建无状态 *Deployment* 应用**。
   - 如果您选择使用Yaml文件部署，请参见**编排模板**。

   > **说明:**
   > 如果您的应用由多个镜像承载的服务组成，建议选择使用Yaml文件部署应用。

4. 查看部署后应用的状态和**相应的服务**、**容器信息**。

2 基础入门

2.1 快速创建Kubernetes集群

本文以专有版 Kubernetes 集群为例，介绍如何使用控制台快速创建 Kubernetes 集群。

前提条件

您需要开通容器服务、资源编排（ROS）服务、弹性伸缩（ESS）服务和访问控制（RAM）服务。

更多限制和使用说明的信息，参见*#unique_14*。

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

背景信息

本例将演示如何快速创建一个Kubernetes集群，部分配置采用默认或最简配置。

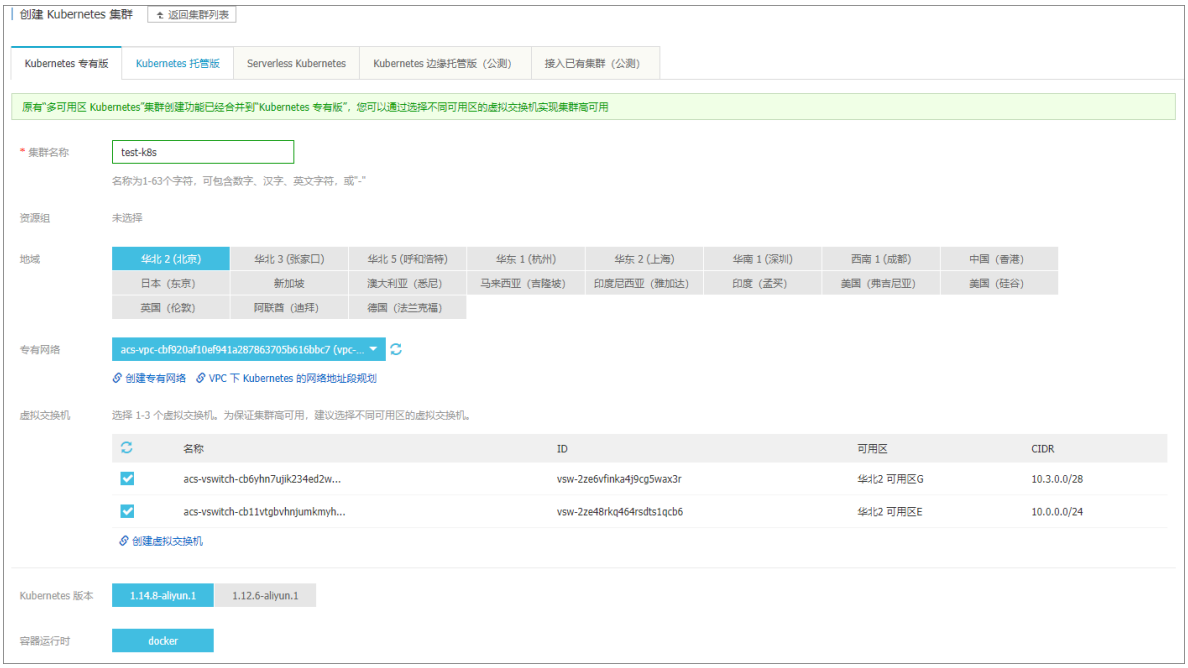
操作步骤

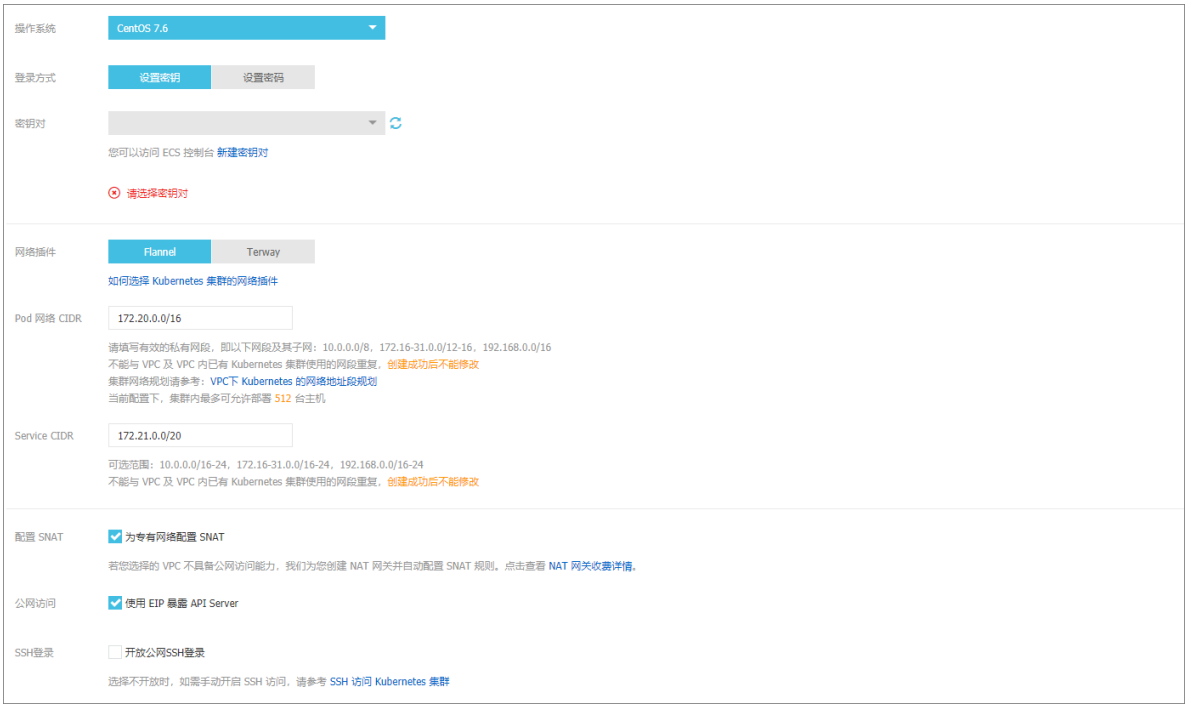
1. 登录**容器服务管理控制台**。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 集群，进入集群列表页面。
3. 单击页面右上角的创建 Kubernetes 集群，在弹出的选择集群模板中，选择标准专有集群页面，单击创建。

   默认进入Kubernetes 专有版集群配置页面。

4. 配置集群参数。

本例中大多数配置保留默认值，具体的配置如下图所示。





配置项	配置说明
集群名称	名称为1-63个字符，可包含数字、汉字、英文字符，或"-"。
资源组	集群所处的资源组。
地域	集群所处地域。

配置项	配置说明
专有网络	集群的专有网络。 您可以在已有 VPC 列表中选择所需的 VPC 。 · 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。
虚拟交换机	集群的虚拟交换机。您可以在已有 vswitch 列表中选择所需的 vswitch 。 您可以根据需要，选择1~3个 vswitch。推荐您选择3个 vswitch。
Kubernetes 版本和容器运行时	显示容器运行时和 Kubernetes 版本，您可根据需要选择 Kubernetes 版本。
节点类型	支持按量付费和包年包月。
购买时长和自动续费（可选项）	您可以选择购买的时长及自动续费的时长。 说明: 选择包年包月节点类型时可配置。
Master实例配置	选择实例数量、实例规格和系统盘。 · Master 实例数量：选择3个或5个实例。 · 实例规格：参见#unique_15。 · 系统盘：支持SSD云盘和高效云盘。

配置项	配置说明
Worker节点配置	您可选择新增实例或添加已有实例。若选择新增实例，可进行如下配置。 · 实例规格：支持选择多个实例规格。参见#unique_15。 · 已选规格：选中的规格呈现在这里。 · 数量：新增 Worker 实例的数量。 · 系统盘：支持 SSD 云盘和高效云盘。 · 挂载数据盘：支持 SSD 云盘、高效云盘和普通云盘。 说明: 挂载数据盘时，支持云盘加密。
操作系统	操作系统类型支持 CentOS 7.6 和 AliyunLinux 2.1903。
登录方式	支持设置密钥和密码，关于使用密钥登录的信息，参见#unique_16。
网络插件	支持Flannel和Terway，默认启用Flannel。具体可参见#unique_17。
Pod网络CIDR和Service CIDR	具体如何规划可参见#unique_18。
配置SNAT	可选，若不选择，需要自行配置NAT网关，或手动配置SNAT。
公网访问	· 如果选择开放，会创建一个EIP，并挂载到内网SLB上。此时，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。 · 若选择不开放，不会创建EIP，用户只能在VPC内部用kubeconfig连接/操作集群。
SSH登录（可选项）	· 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。您可手动进行配置，具体操作参见#unique_19 。 说明: 勾选公网访问时可配置。

配置项	配置说明
云监控插件	在节点上安装云监控插件，可以在云监控控制台查看所创建ECS实例的监控信息。
Ingress	在节点上安装 Ingress 组件。 说明: 如果您勾选创建 Ingress Dashboard，则需要同步安装日志服务组件。
日志服务	您可使用已有Project或新建一个Project。 勾选使用日志服务，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*#unique_20*。
存储插件	设置存储插件，支持 Flexvolume 和 CSI。
集群删除保护	为防止通过控制台或 API 误释放集群，默认启用集群删除保护。
RDS白名单	将节点 IP 添加到 RDS 实例的白名单。
实例保护	为防止通过控制台或 API 误删除释放集群节点，该选项默认勾选。
标签	为集群绑定标签。 说明: · 键是必需的，而 值 是可选的，可以不填写。 · 键 不能是 aliyun、http:// 、https:// 开头的字符串，不区分大小写，最多 64 个字符。 · 值 不能是 http:// 或 https://，可以为空，不区分大小写，最多 128 个字符。 · 同一个资源，标签键不能重复，相同标签键（Key）的标签会被覆盖。 · 如果一个资源已经绑定了 20 个标签，已有标签和新建标签会失效，您需要解绑部分标签后才能再绑定新的标签。

配置项	配置说明
高级选项	· 节点Pod数量：单节点可运行 Pod 数量的上限。 · 给节点设置安全组。请参见#unique_21。 · kube-proxy代理模式：支持 iptables 和 IPVS 两种模式。 - iptables：成熟稳定的kube-proxy代理模式，Kubernetes service的服务发现和负载均衡使用iptables规则配置，但性能一般，受规模影响较大，适用于集群存在少量的service。 - IPVS：高性能的kube-proxy代理模式，Kubernetes service的服务发现和负载均衡使用Linux ipvs模块进行配置，适用于集群存在大量的service，对负载均衡有高性能要求的场景。 · 自定义节点名称：为方便管理，您可以开启并自定义节点的名称。 · 节点服务端口范围：设置节点服务端口范围。默认端口范围为30000~32767。 · CPU Policy：设置CPU Policy。 - none：默认策略，表示启用现有的默认 CPU 亲和方案。 - static：允许为节点上具有某些资源特征的 Pod 赋予增强的 CPU 亲和性和独占性。 · 集群本地域名：默认域名为 cluster.local，可自定义域名。 · 集群 CA：设置是否开启集群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。 · 工作流引擎：设置是否使用AGS。 - 如果勾选 AGS，则创建集群时系统自动安装 AGS 工作流插件。 - 如果不勾选，则需要手动安装 AGS 工作流插件，请参见#unique_22。

5. 单击创建集群，启动部署。

   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。

2.2 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*#unique_5*。

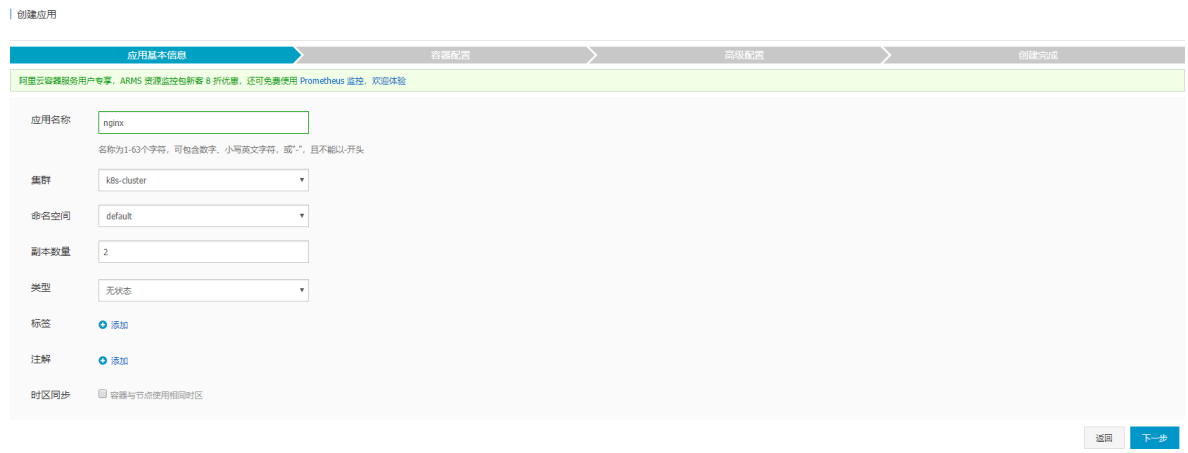
操作步骤

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 > 无状态，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、类型、注解和标签，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。

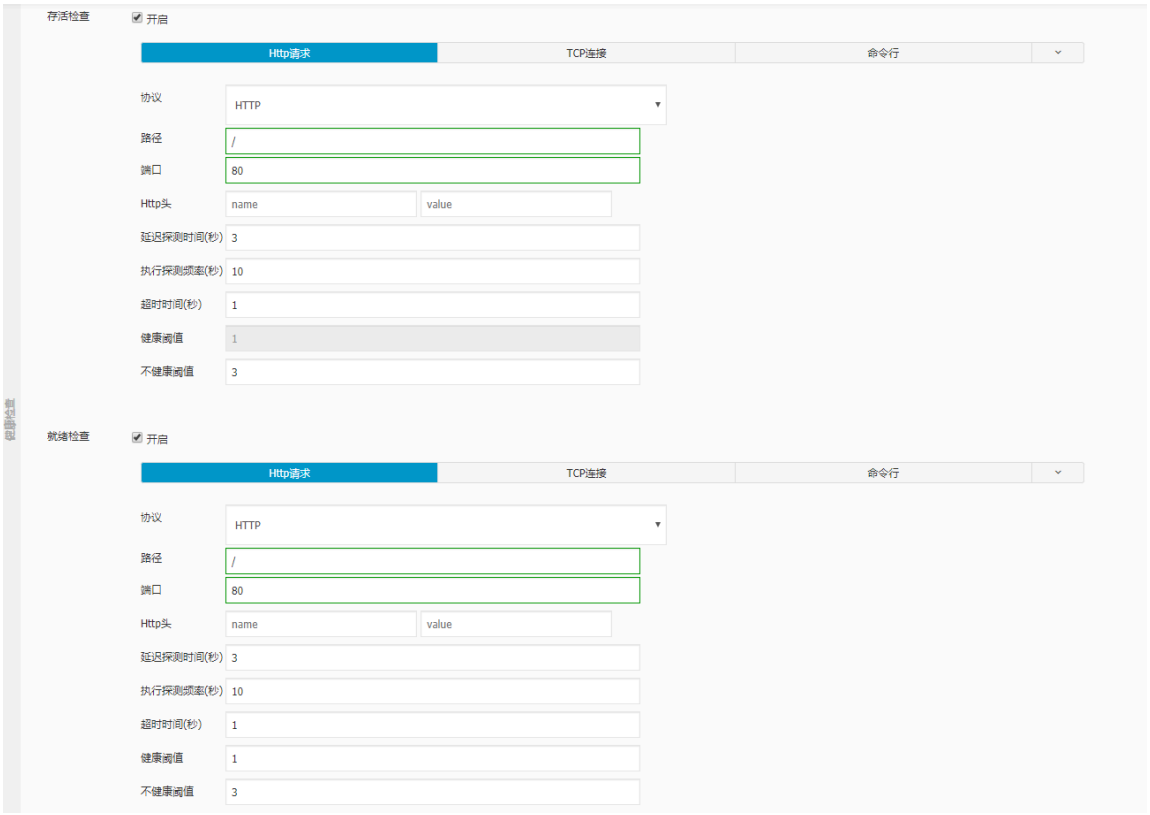
- 镜像密钥：单击设置镜像密钥设置镜像的密钥。对于私有仓库访问时，需要设置密钥，具体可以参见#unique_23
- 资源限制：可指定该应用所能使用的资源上限，包括 CPU 和内存两种资源，防止占用过多资源。其中，CPU 资源的单位为 cores，即一个核；内存的单位为 Bytes，可以为 Mi。
- 所需资源：即为该应用预留资源额度，包括 CPU 和内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争占资源，导致应用不可用。
- Init Container：勾选该项，表示创建一个 Init Container，Init Container 包含一些实用的工具，具体参见https://kubernetes.io/docs/concepts/workloads/pods/init-containers/。

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） 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。


存活检查
开启
Http请求
TCP连接
命令行
协议
HTTP
路径
/
端口
80
Http头
name
value
延迟探测时间(秒)
3
执行探测频率(秒)
10
超时时间(秒)
1
健康阈值
1
不健康阈值
3
就绪检查
开启
Http请求
TCP连接
命令行
协议
HTTP
路径
/
端口
80
Http头
name
value
延迟探测时间(秒)
3
执行探测频率(秒)
10
超时时间(秒)
1
健康阈值
1
不健康阈值
3

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 10 秒，最小为 1 秒。 · 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。

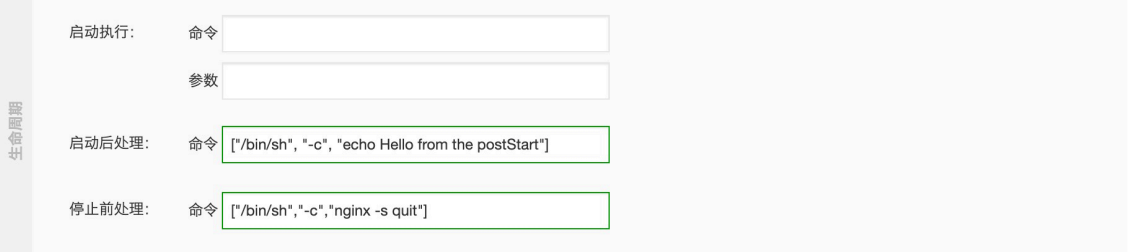
请求类型	配置说明
TCP连接	即向容器发送一个 TCP Socket，kubelet 将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： · 端口：容器暴露的访问端口或端口名，端口号必须介于 1~65535。 · 延迟探测时间（秒）：即 initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为 15 秒。 · 执行探测频率（秒）：即 periodSeconds，指执行探测的时间间隔，默认为 10 秒，最小为 1 秒。 · 超时时间（秒）：即 timeoutSeconds，探测超时时间。默认 1 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 10 秒，最小为1秒。 · 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) 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/*。

· 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。

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。

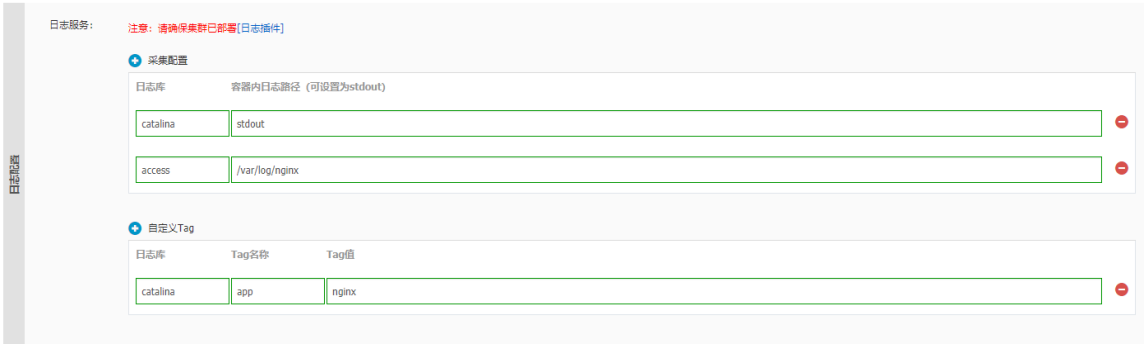
f) （可选）配置日志服务，您可进行采集配置和自定义 Tag 设置。

> **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。

a) 设置访问设置。

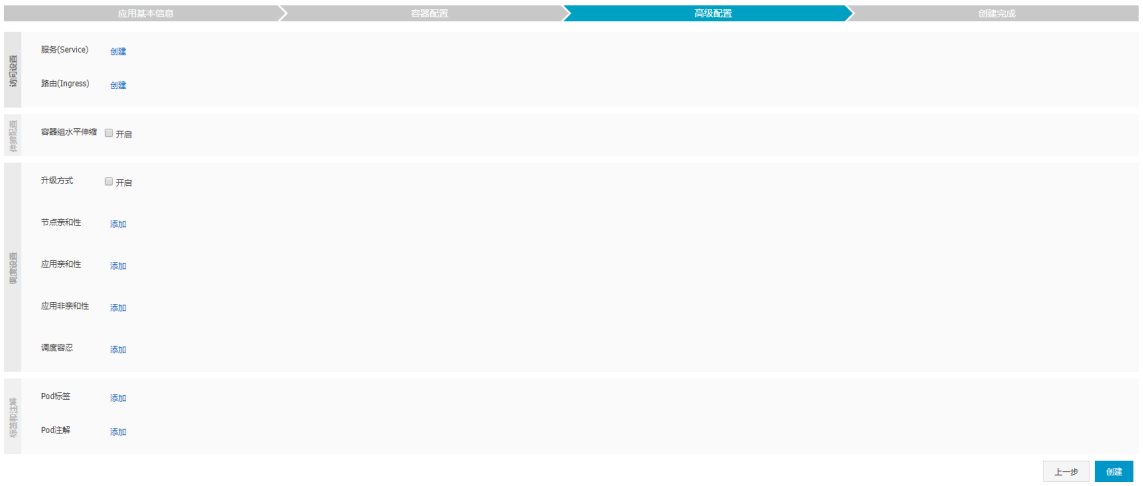
您可以设置暴露后端 Pod 的方式，最后单击创建。本例中选择 ClusterIP 服务和路由（Ingress），构建一个可公网访问的 nginx 应用。

说明:

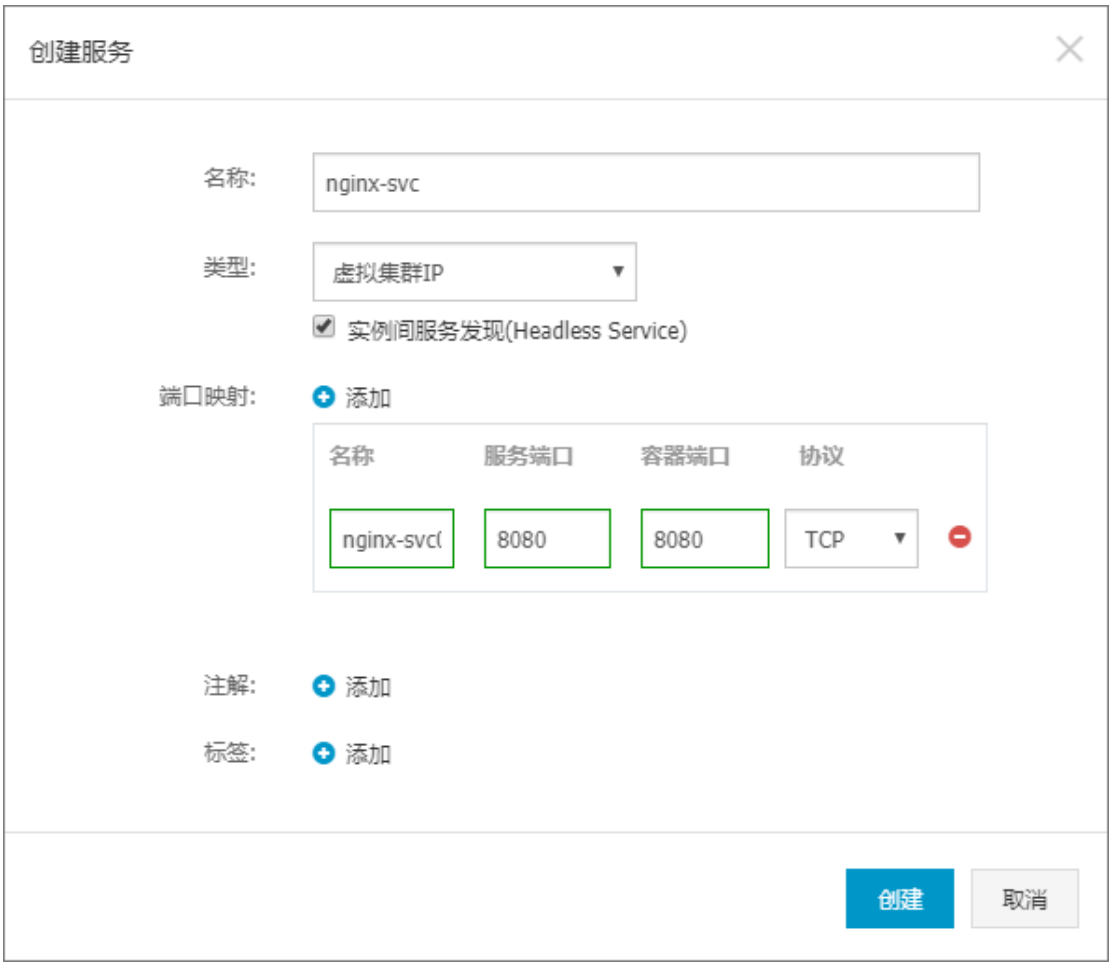
针对应用的通信需求，您可灵活进行访问设置：

- 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。**
  - **创建路由（Ingress）：通过路由（Ingress）提供公网访问能力，详情参见***https://kubernetes.io/docs/concepts/services-networking/ingress/*。



A. 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。



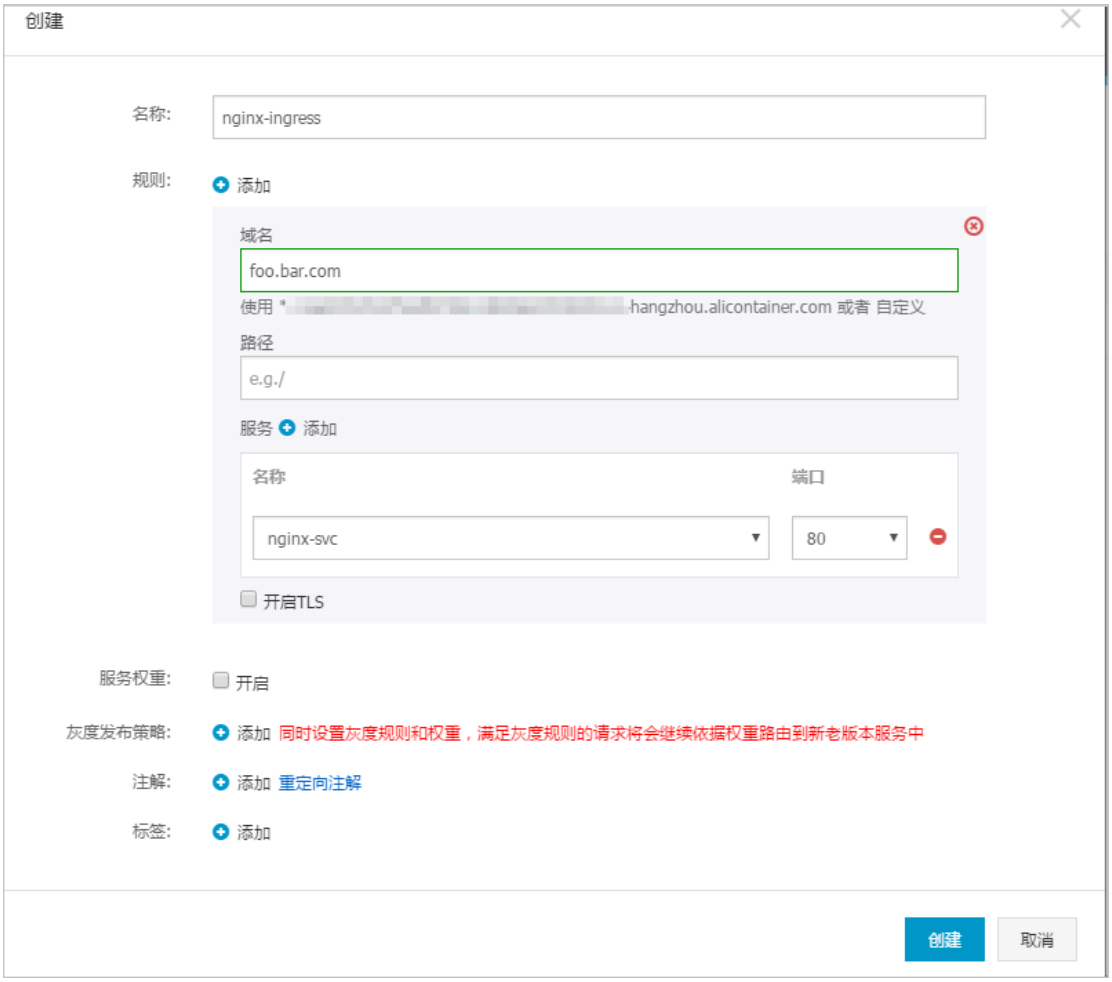
- 名称：您可自主设置，默认为 `applicationname-svc`。
- 类型：您可以从下面 3 种服务类型中进行选择。
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该项，服务只能够在集群内部可以访问。
  - 节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
  - 负载均衡：即 LoadBalancer，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*#unique_24*。
- 标签：您可为该服务添加一个标签，标识该服务。

B. 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*#unique_25*。

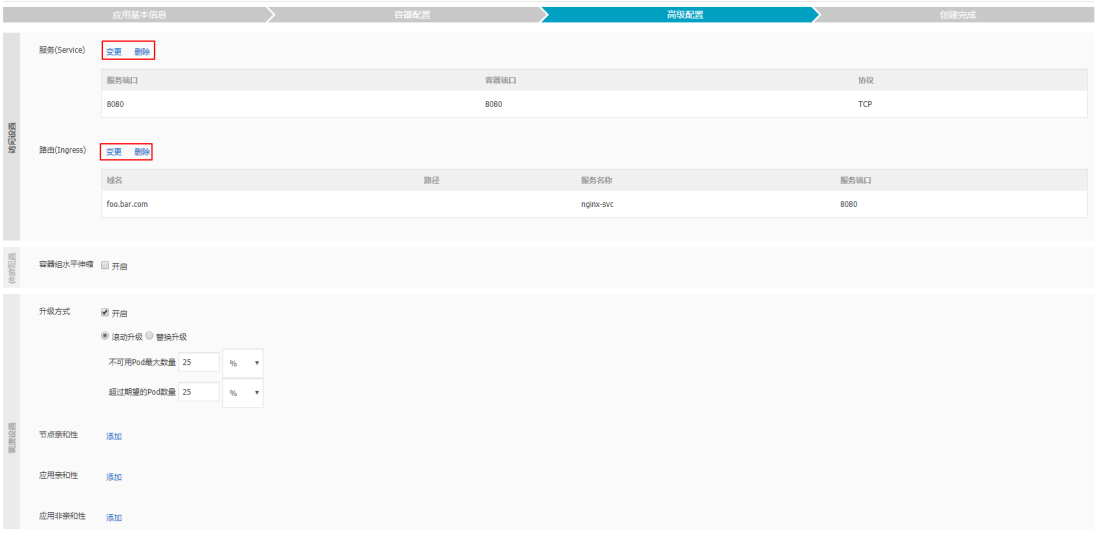
 **说明:**

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
```



C. 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） 设置调度设置。

您可设置升级方式、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*。

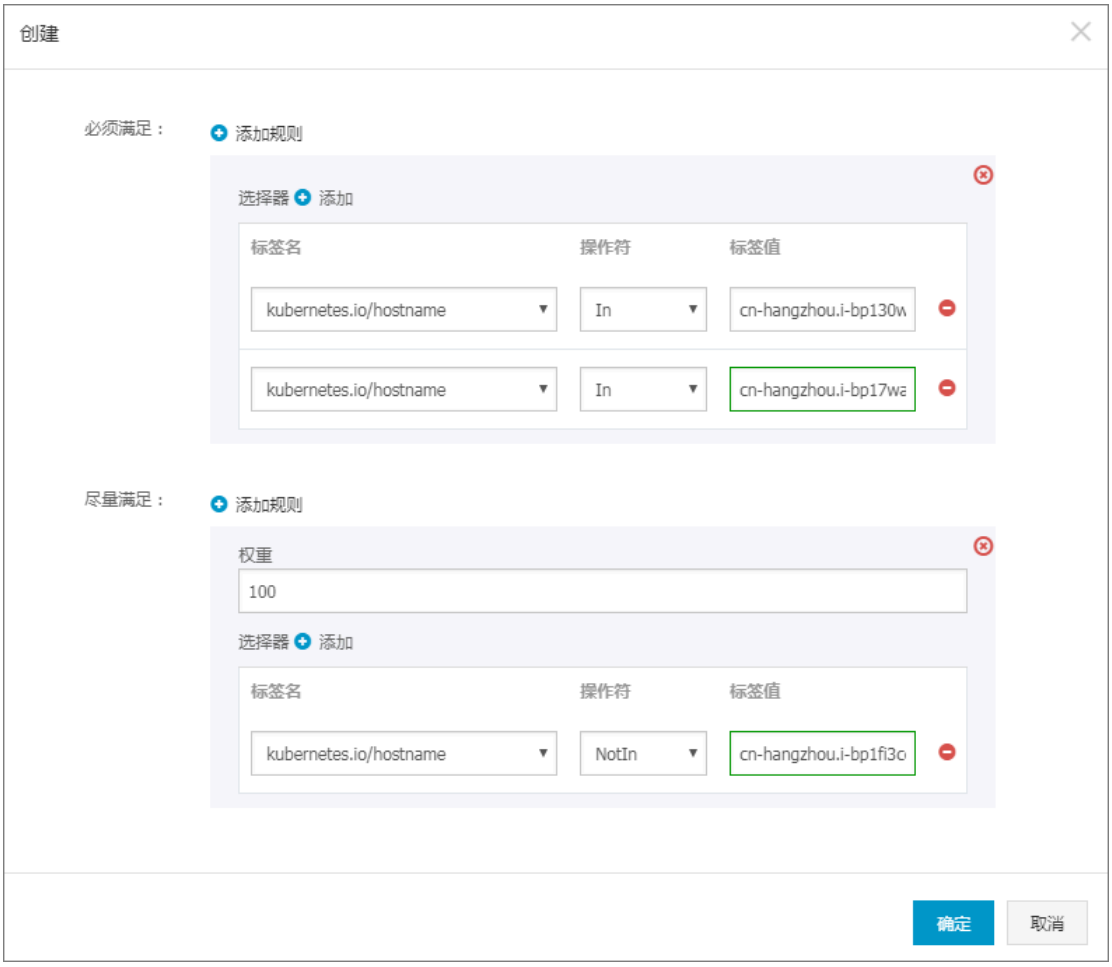
说明:

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。

A. 设置升级方式。

升级方式包括滚动升级（rollingupdate）和替换升级（recreate），详细请参见*https://kubernetes.io/zh/docs/concepts/workloads/controllers/deployment/*

B. 设置节点亲和性，通过 Node 节点的 Label 标签进行设置。

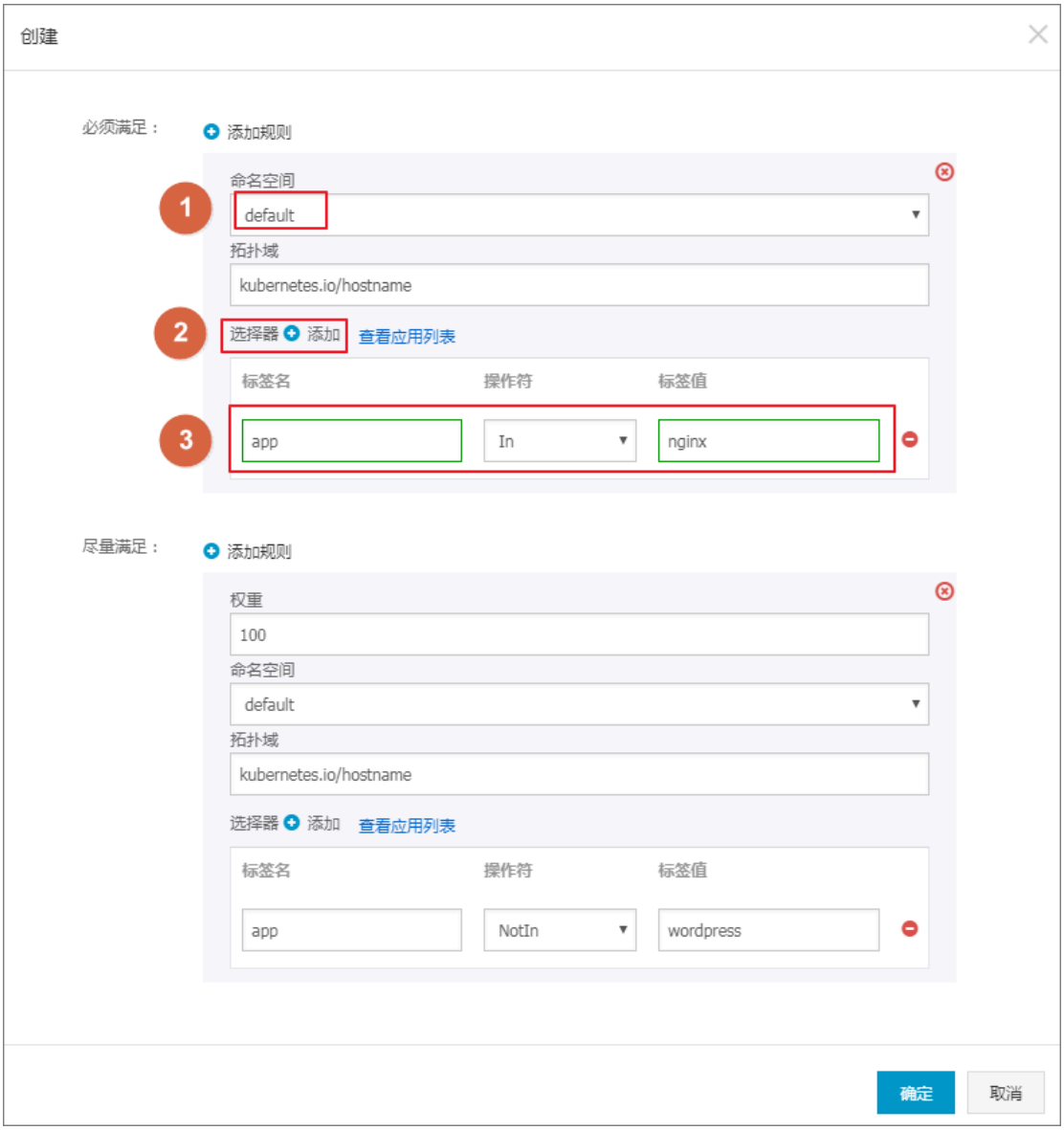


节点调度支持硬约束和软约束（Required/Preferred），以及丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- **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**

大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。

C. 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 调度到权重最大的节点上。

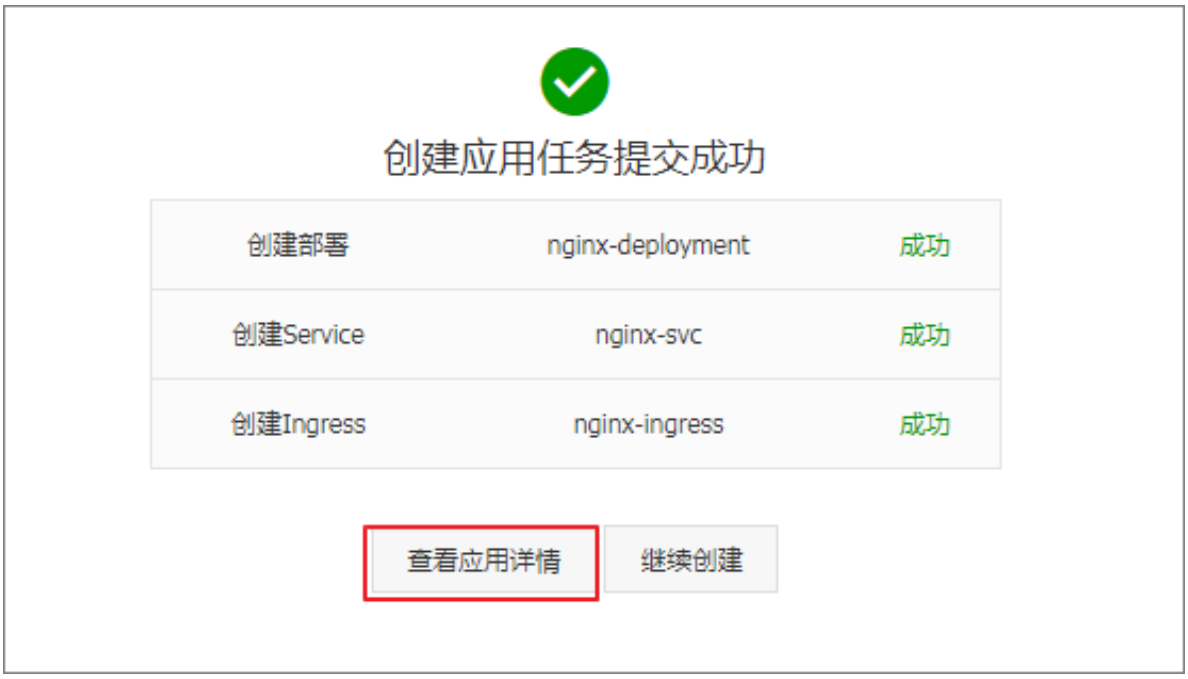
D. 设置应用非亲和性调度，决定应用的 Pod 不与哪些 Pod 部署在同一拓扑域。应用非亲和性调度的场景包括：

· 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。

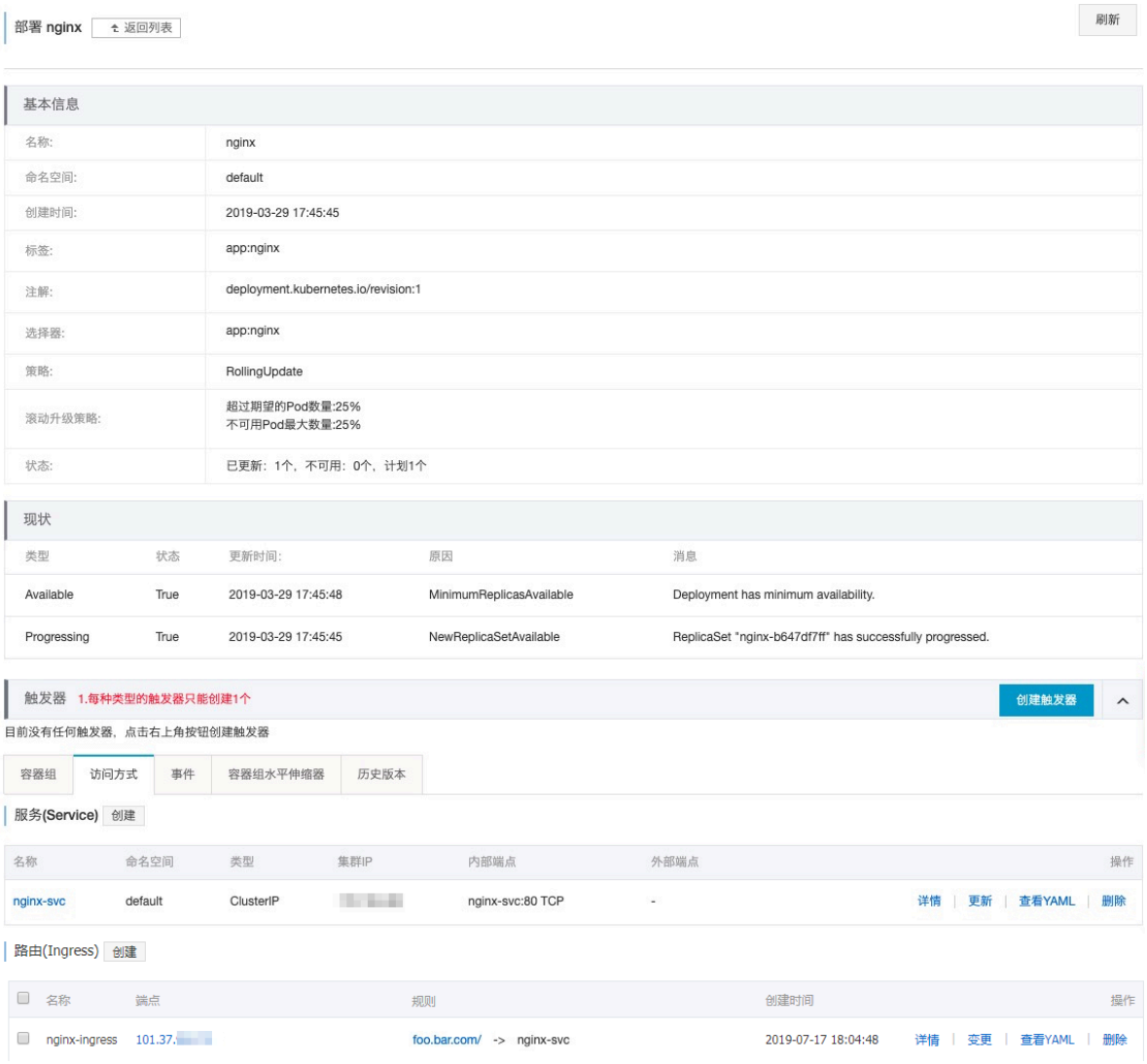
> 说明:
> 应用非亲和性调度的设置方式与亲和性调度相同，但是相同的调度规则代表的意思不同，请根据使用场景进行选择。

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。

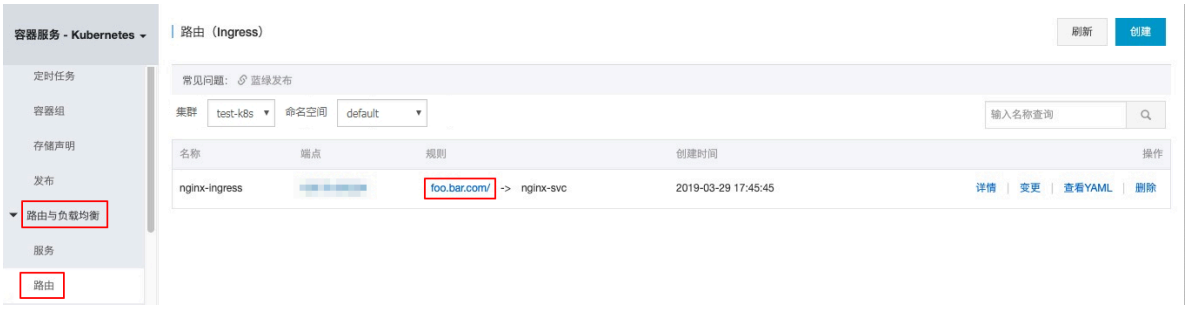


说明:

您也可以通过以下操作创建路由与服务。如上图所示，在访问方式页签。

- **单击服务右侧的创建，也可以进行服务创建，操作步骤同 6.i.a。**
- **您单击路由右侧的创建，进行路由的创建，操作同 6.i.b。**

9. 单击左侧导航栏的路由与负载均衡 > 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。

2.3 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*#unique_5*。
- 您已成功创建一个云盘存储卷声明，参见*#unique_27*。
- 您已连接到 Kubernetes 集群的 Master 节点，参见*#unique_28*。

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 > 有状态，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。
   - 标签：为该应用添加一个标签，标识该应用。
   - 注解：为该应用添加一个注解（annotation）。

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。

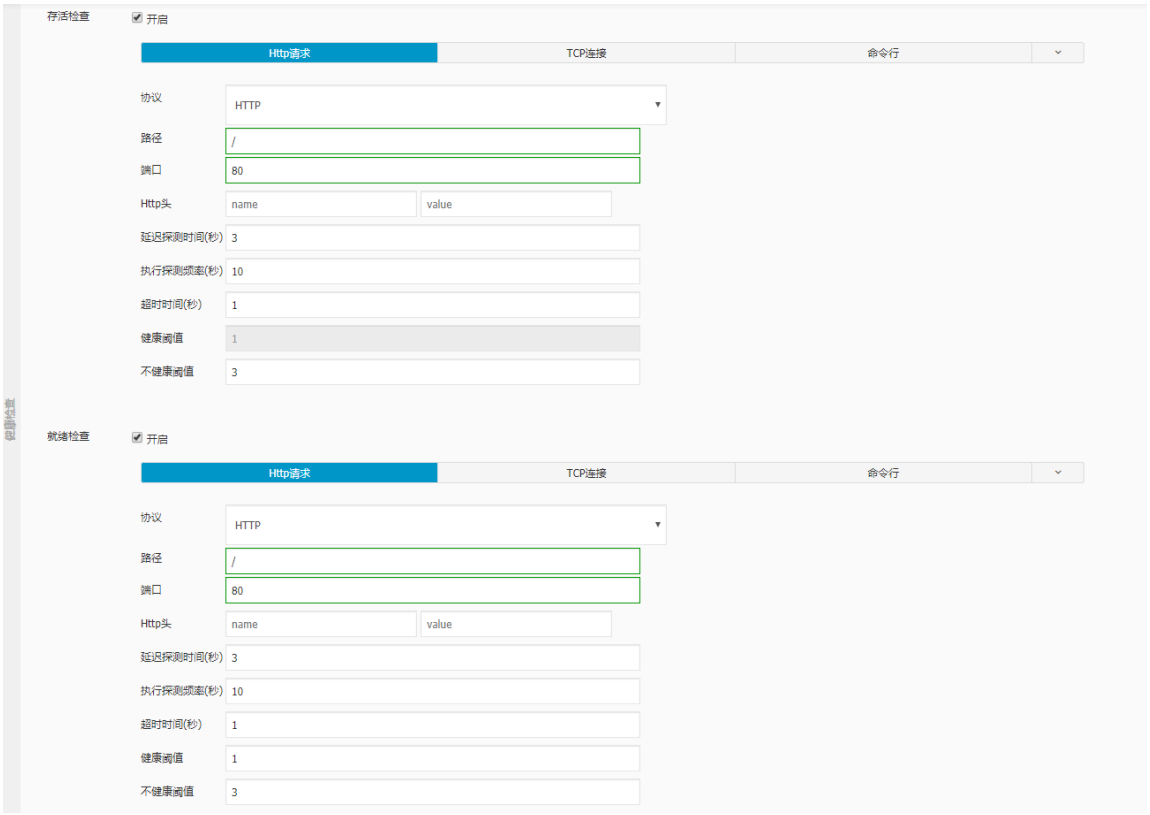
- 镜像密钥：单击设置镜像密钥设置镜像的密钥。对于私有仓库访问时，需要设置密钥，具体可以参见*#unique_23*。
- 资源限制：可指定该应用所能使用的资源上限，包括 CPU 和内存两种资源，防止占用过多资源。其中，CPU 资源的单位为 millicores，即一个核的千分之一；内存的单位为 Bytes，可以为 Gi、Mi 或 Ki。
- 所需资源：即为该应用预留资源额度，包括 CPU 和内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争夺资源，导致应用不可用。
- Init Container：勾选该项，表示创建一个Init Container，Init Container 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 10 秒，最小为 1 秒。 · 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。

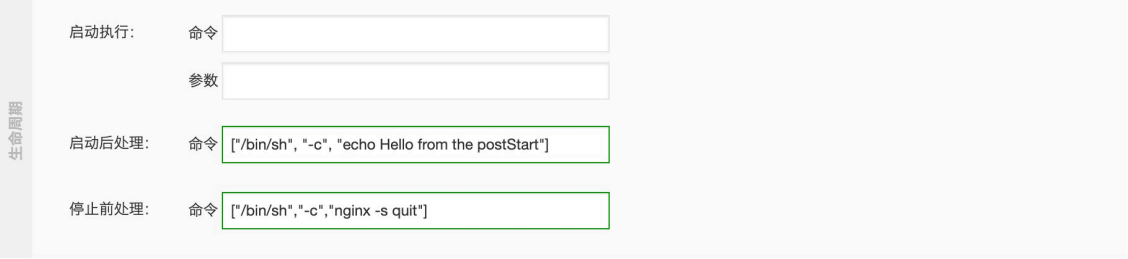
请求类型	配置说明
TCP连接	即向容器发送一个 TCP Socket，kubelet 将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： · 端口：容器暴露的访问端口或端口名，端口号必须介于 1~65535。 · 延迟探测时间（秒）：即 initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为 15 秒。 · 执行探测频率（秒）：即 periodSeconds，指执行探测的时间间隔，默认为 10 秒，最小为 1 秒。 · 超时时间（秒）：即 timeoutSeconds，探测超时时间。默认 1 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 10 秒，最小为 1 秒。 · 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/*。

· 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。

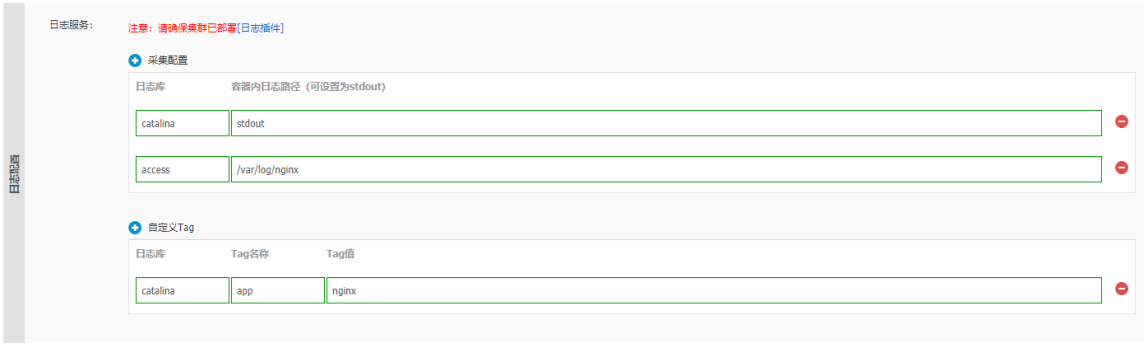
f) （可选）配置日志服务，您可进行采集配置和自定义 Tag 设置。

> **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。

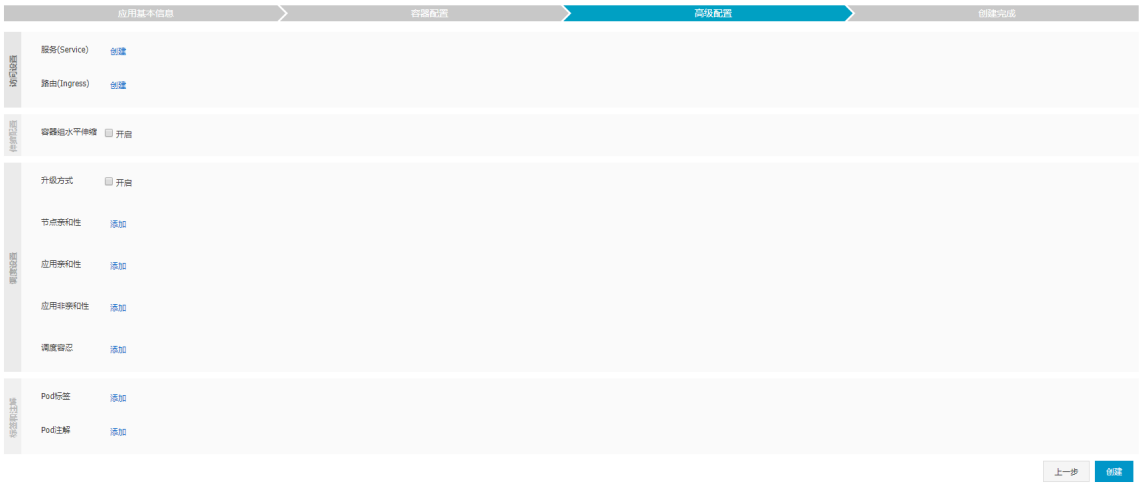
您可以设置暴露后端 Pod 的方式，最后单击创建。本例中选择 ClusterIP 服务和路由（Ingress），构建一个公网可访问的 nginx 应用。

> **说明:**

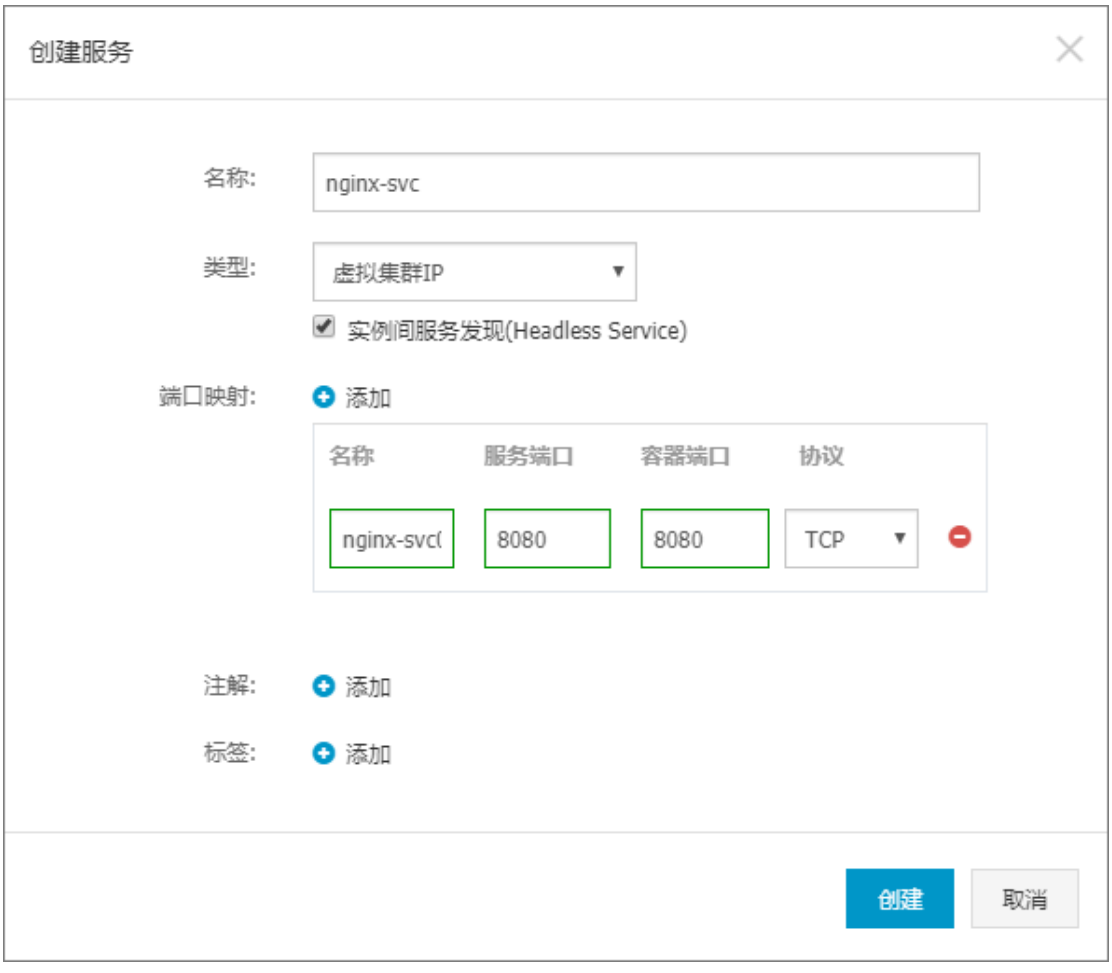
针对应用的通信需求，您可灵活进行访问设置：

- **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。**
  - **创建路由（Ingress）：通过路由（Ingress）提供公网访问能力，详情参见***https://kubernetes.io/docs/concepts/services-networking/ingress/*。



A. 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。



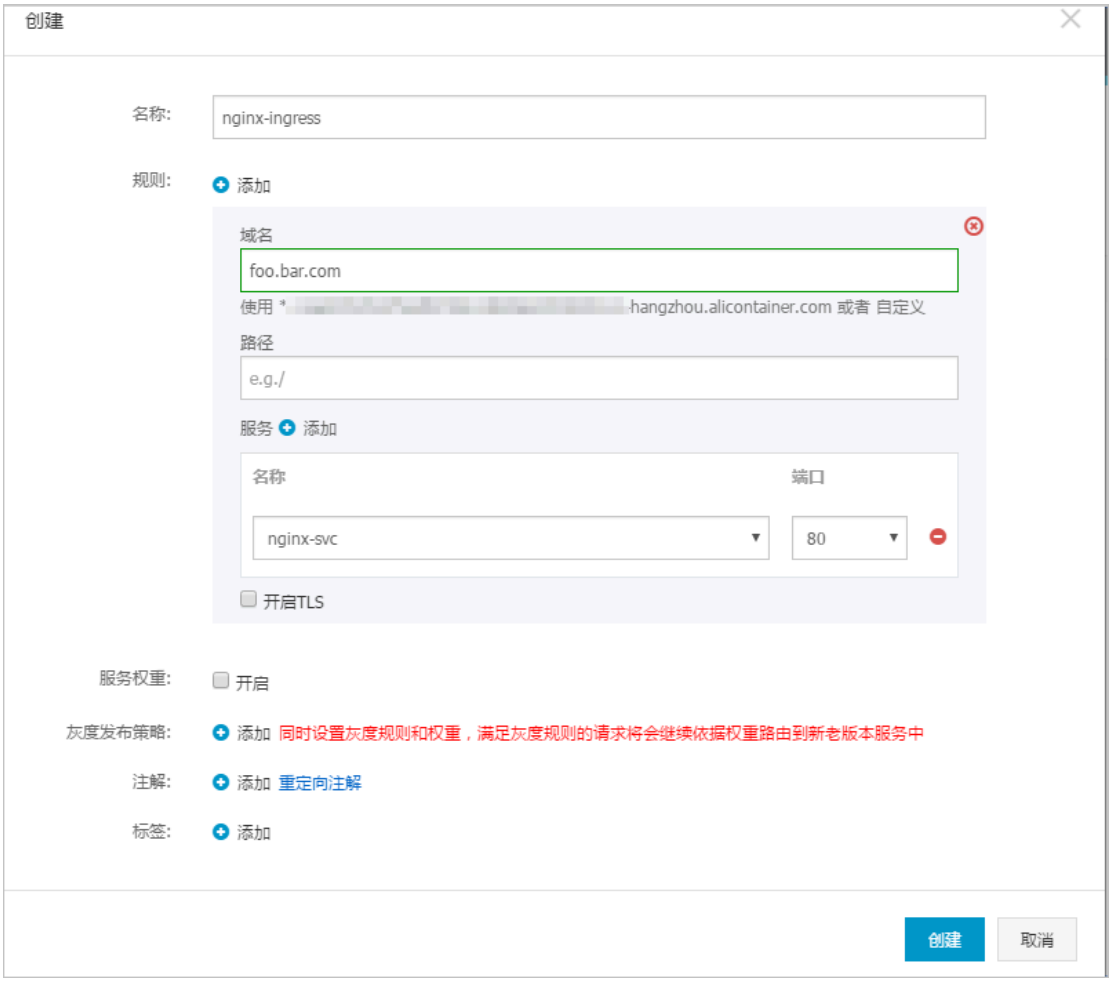
- 名称：您可自主设置，默认为 `applicationname-svc`。
- 类型：您可以从下面 3 种服务类型中进行选择。
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该项，服务只能够在集群内部可以访问。
  - 节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
  - 负载均衡：即 LoadBalancer，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*#unique_24*。
- 标签：您可为该服务添加一个标签，标识该服务。

B. 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*#unique_25*。

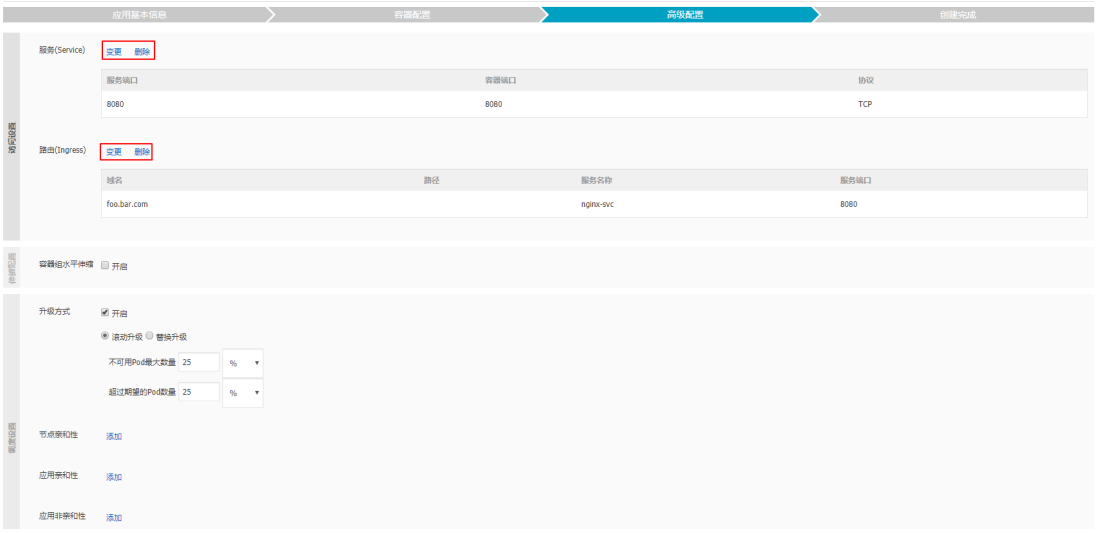
 **说明:**

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
```



C. 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。
- 最大副本数量：该 StatefulSet 可扩容的容器数量上限。
- 最小副本数量：该 StatefulSet 可缩容的容器数量下限。

c) （可选） 设置调度设置。

您可设置升级方式、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*。

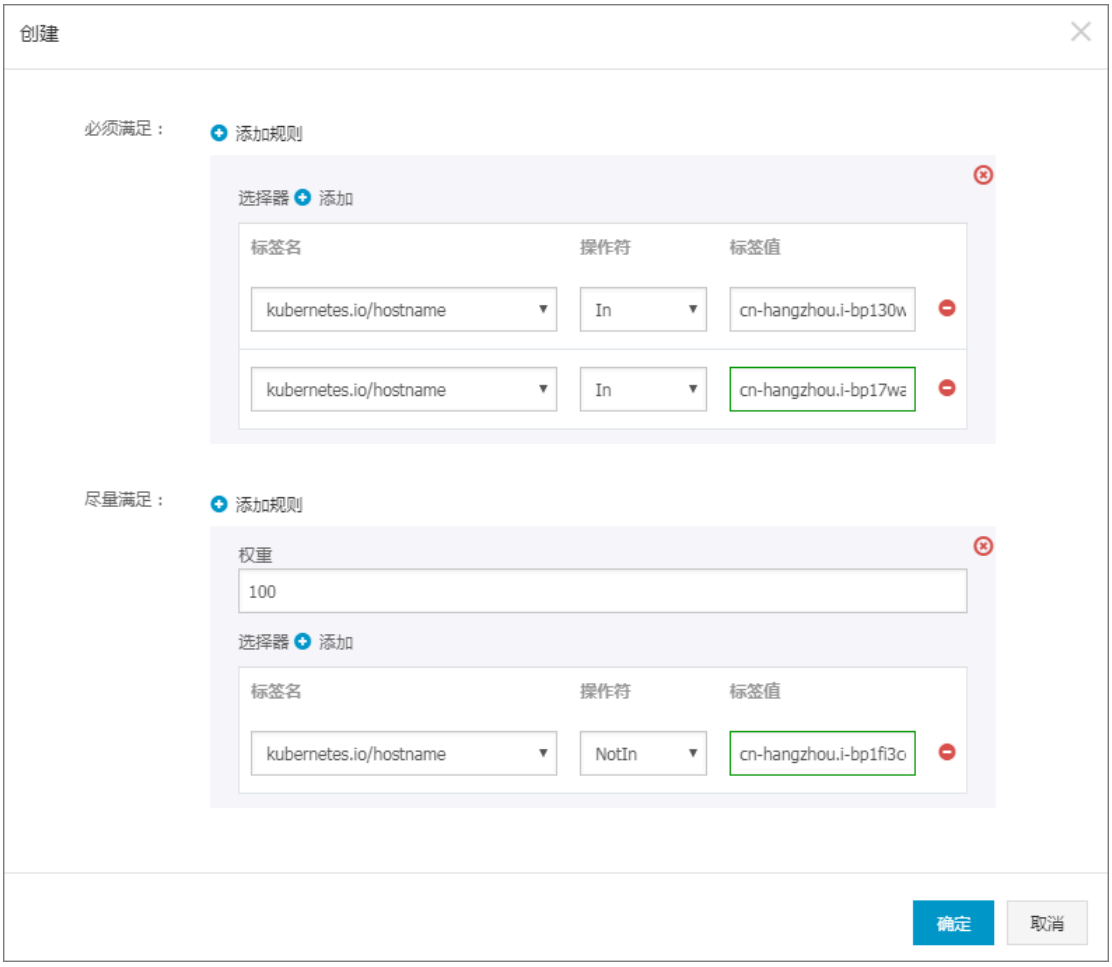
说明:

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。

A. 设置升级方式。

升级方式包括滚动升级（rollingupdate）和替换升级（recreate），详细请参见*https://kubernetes.io/zh/docs/concepts/workloads/controllers/deployment/*

B. 设置节点亲和性，通过 Node 节点的 Label 标签进行设置。

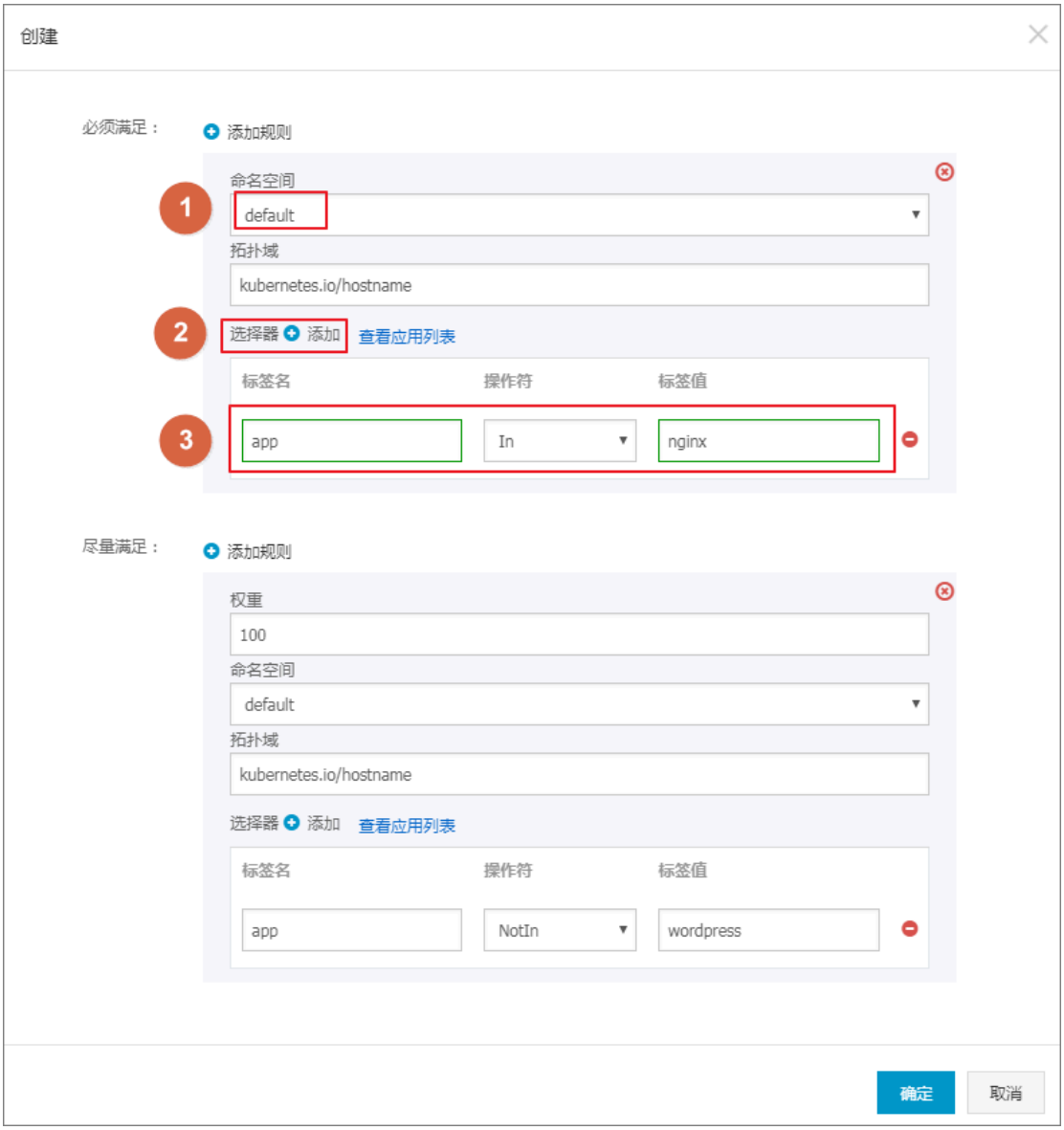


节点调度支持硬约束和软约束（Required/Preferred），以及丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- **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**

大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。

C. 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 调度到权重最大的节点上。

D. 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：

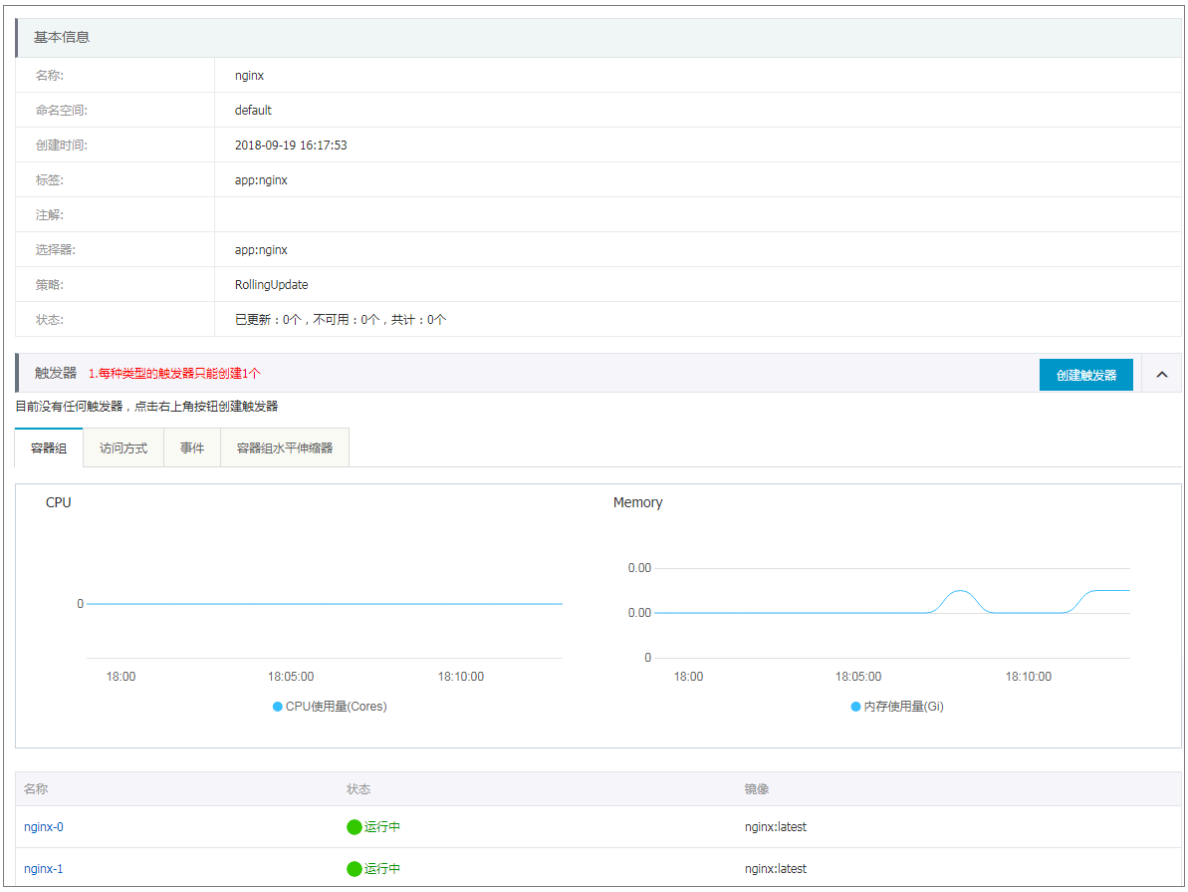
· 将一个服务的 Pod 分散部署到不同的拓扑域（如不同主机）中，提高服务本身的稳定性。
· 给予 Pod 一个节点的独占访问权限来保证资源隔离，保证不会有其它 Pod 来分享节点资源。
· 把可能会相互影响的服务的 Pod 分散在不同的主机上。

> 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。

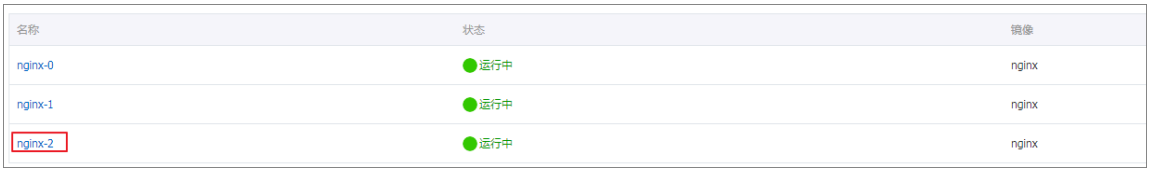
默认进入有状态副本集详情页面。



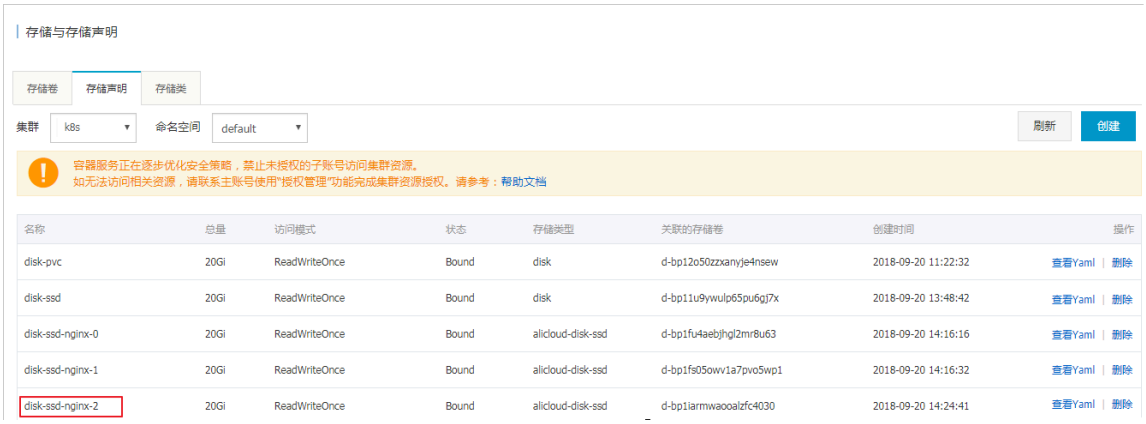
9. 然后单击左上角返回列表，进入有状态副本集列表页面，查看创建的 StatefulSet 应用。

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。



后续步骤

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。

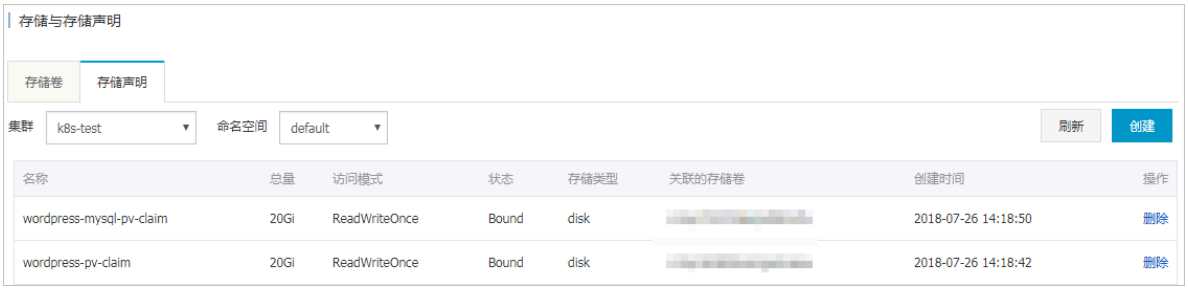
想要了解更多信息，参见*Kubernetes*有状态服务-*StatefulSet*使用最佳实践。

2.4 部署有依赖关系的wordpress应用

本文主要为您介绍如何部署有依赖关系的wordpress应用。

前提条件

- 创建一个 Kubernetes 集群。详情参见快速创建*Kubernetes*集群。
- 您需要创建存储卷和存储卷声明，关于如何创建存储卷，请参见*#unique_30*、*#unique_31*、*#unique_32*；关于如何创建存储卷声明，请参见*#unique_27*。在这里我们使用阿里云云盘作为存储卷；在示例中我们选择PV/PVC的方式进行存储卷挂载。在这里创建wordpress-pv-claim和wordpress-mysql-pv-claim两个存储声明，分别会在wordpress和wordpress-mysql的yaml文件中使用这两个声明来挂载相应的存储卷。



背景信息

本例主要演示如何通过编排模板中的自定义模板创建有依赖关系的应用。

主要组件有：

- wordpress
- mysql

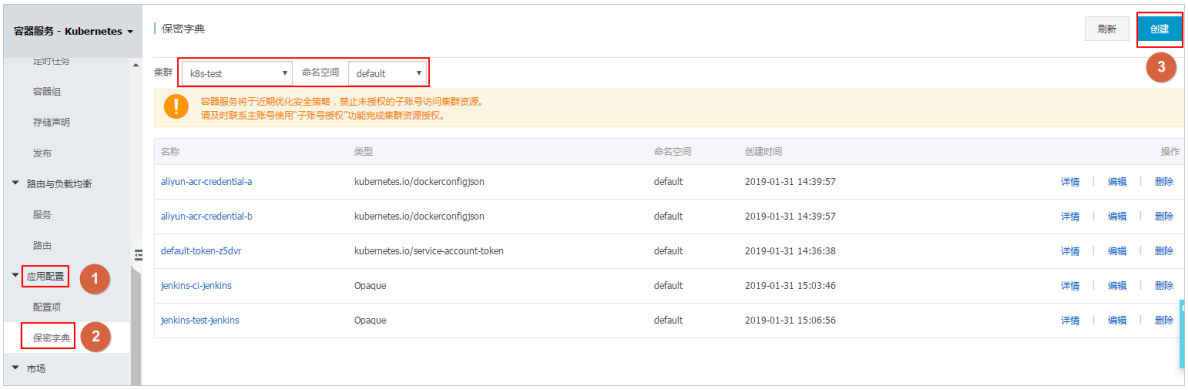
涉及到的资源：

- 存储卷管理
- secret管理
- 服务

操作步骤

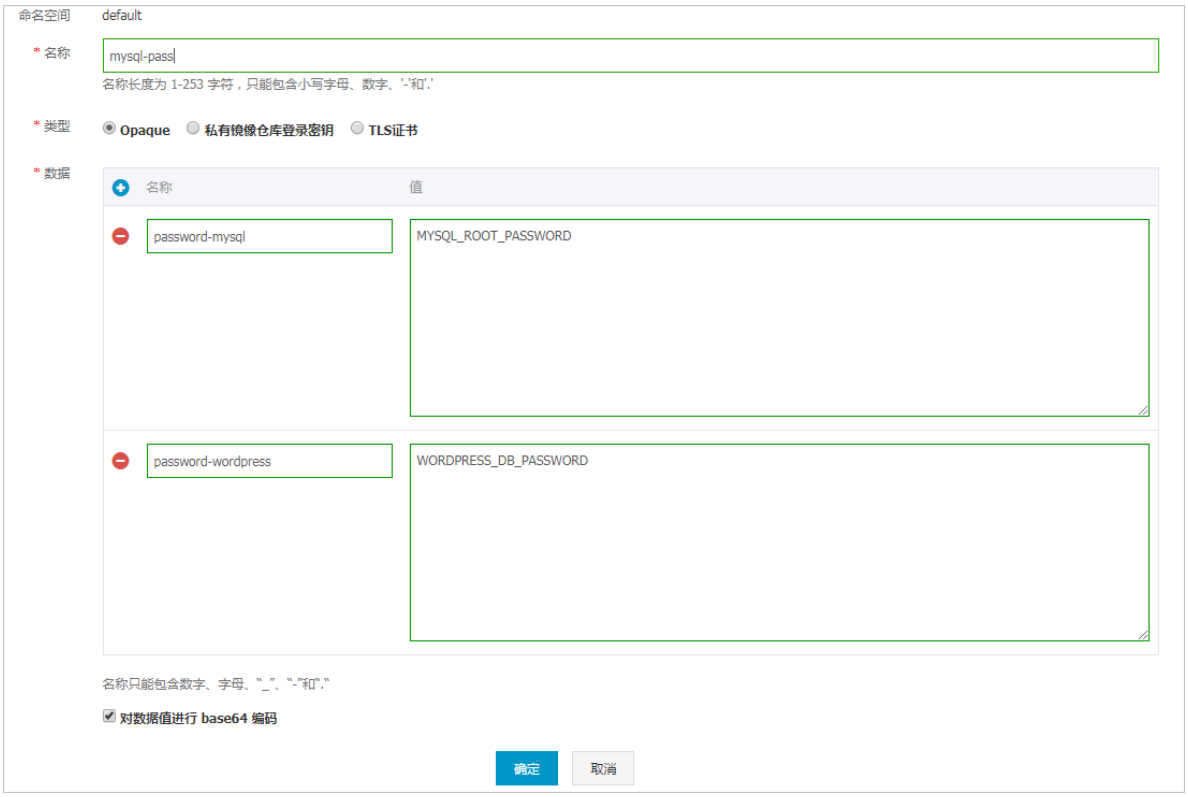
1. 登录容器服务管理控制台。
2. 使用准备好的存储卷声明。在这里创建wordpress-pvc和wordpress-mysql-pvc两个存储声明，分别在wordpress和wordpress-mysql的yaml文件中，用这两个声明挂载相应的存储卷。

3. 单击左侧导航栏中应用配置 > 保密字典，选择所需的集群和命名空间，单击右上角创建。创建过程请参考#unique_33。



由于创建和访问mysql数据库需要用户名密码，所以我们通过创建密钥的方式进行用户名密码的管理。

在使用secret前，您需要先将需要加密的secret在保密字典里进行创建，在本例中通过将mysql root的密码作为密钥进行创建，创建名称为mysql-pass，类型选择Opaque。该密钥会在后面的wordpress和wordpress-mysql的yaml文件中用到。



4. 单击左侧导航栏中的应用 > 无状态，单击右上角使用模板创建。

选择所需的集群和命名空间，创建wordpress deployment的yaml文件如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql  #通过名称指向需要访问的mysql，该名称与
mysql service的名称相对应。
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-pvc
          mountPath: /var/www/html
      volumes:
      - name: wordpress-pvc
        persistentVolumeClaim:
          claimName: wordpress-pv-claim
```

创建mysql deployment的yaml文件如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
```

```
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: wordpress-mysql-pvc
          mountPath: /var/lib/mysql
      volumes:
      - name: wordpress-mysql-pvc
        persistentVolumeClaim:
          claimName: wordpress-mysql-pv-claim
```

5. **为了使wordpress能够被外部访问，我们需要为wordpress创建service对外暴露访问方式，在这里使用LoadBalancer类型进行wordpress service的创建，容器服务会自动创建阿里云负载均衡，为用户提供外部访问。**

   wordpress mysql需要创建名为wordpress-mysql的service，以使在上面创建的wordpress deploymet可以访问到。由于该mysql只为wordpress内部调用，所以不需要为其创建LoadBalancer类型的service。

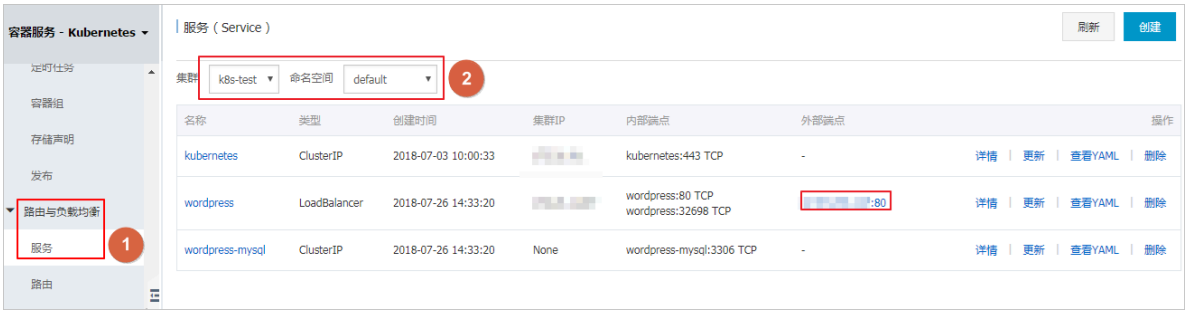
   创建service的方法请参考#unique_34。

   创建wordpress和mysql service的yaml文件如下：

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
```

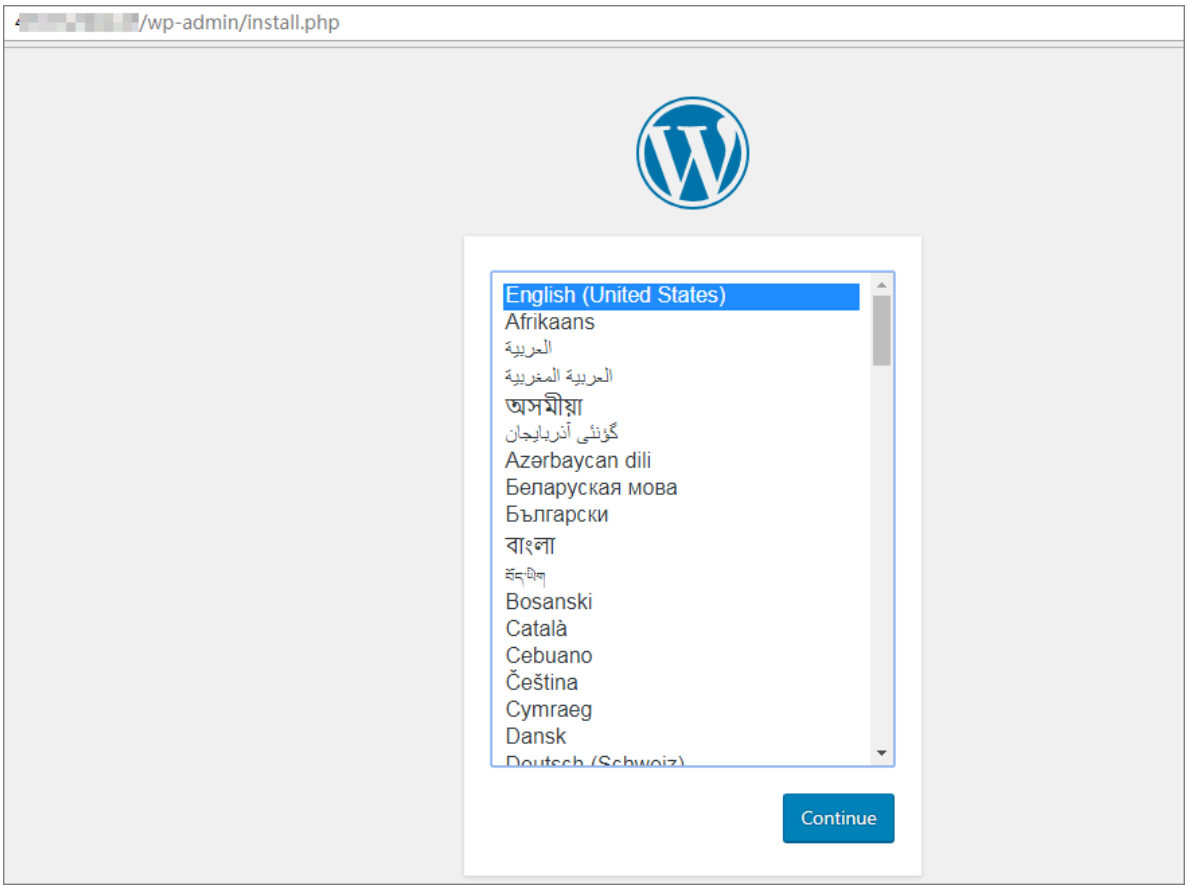
```
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
```

6. **当部署完成后，单击左侧导航栏中的路由与负载均衡 > 服务，找到wordpress服务并查看其外端端点。**



7. 在浏览器中输入*XX.XX.XX.XX/wp-admin/install.php*，访问wordpress服务的外部端点，您就可以通过负载均衡提供的IP地址进行wordpress应用的访问。

   此处的*XX.XX.XX.XX*为上步骤获取的外部端点的IP地址。



后续步骤

在wordpress应用的配置过程中，您可以使用密钥中配置的密码登录应用，此外，wordpress应用所属的容器产生的数据会保存数据卷中。

3 高阶入门

3.1 基于Java应用的DevOps示例

阿里云研发了开源持续交付工具 CodePipeline，致力于让代码的持续交付更简单、安全、高效。它提供多种语言的持续交付向导模板，通过模板快速填写进行持续集成，从而实现多平台、多环境的持续交付。

前提条件

- 准备阿里云 code 代码仓库

  首先登录 阿里云 *code* 管理控制台。然后 Fork 本示例使用的 CodePipeline 官方示例项目 Java-demo，项目地址是 *https://code.aliyun.com/CodePipeline/java-demo* 。

- 创建 kubernetes 集群

  登录容器服务管理控制台创建一个Kubernetes 集群，具体请参见快速创建*Kubernetes*集群。

背景信息

CodePipeline 与阿里云容器服务进行了深度集成，提供了容器化的完整解决方案，实现开发过程中代码构建、部署和验证管理等全生命周期操作。通过 CodePipeline 可以构建您的代码工作流模板，从代码编译/测试，到容器镜像的构建，再到容器环境的发布，打通容器环境下应用发布全过程自动化。更多的应用场景，请参见*CodePipeline* 应用场景。

本示例将演示一个 java 类型项目，执行镜像构建和发布，部署到 Kubernetes 这两个步骤的构建流程。需要用到阿里云code 代码仓库、容器镜像服务、容器服务 Kubernetes 集群，以及 CodePipeline，来实现一个 DevOps 实践案例。

创建阿里云容器镜像仓库

1. 登录 阿里云容器镜像服务。
2. 在左侧导航栏中，单击镜像仓库，再单击创建镜像仓库。

3. **配置镜像仓库。选择与容器集群相同的地域，设置代码源为前面准备的代码仓库** `java-demo`。**最后单击 创建镜像仓库。**

4. **镜像仓库创建成功后，会出现在镜像仓库列表下。**

说明:

如果您是首次使用阿里云容器镜像服务，需要设置 Registry 登录密码，请单击修改 Registry 登录密码 进行设置。

使用 CodePipeline 进行持续构建

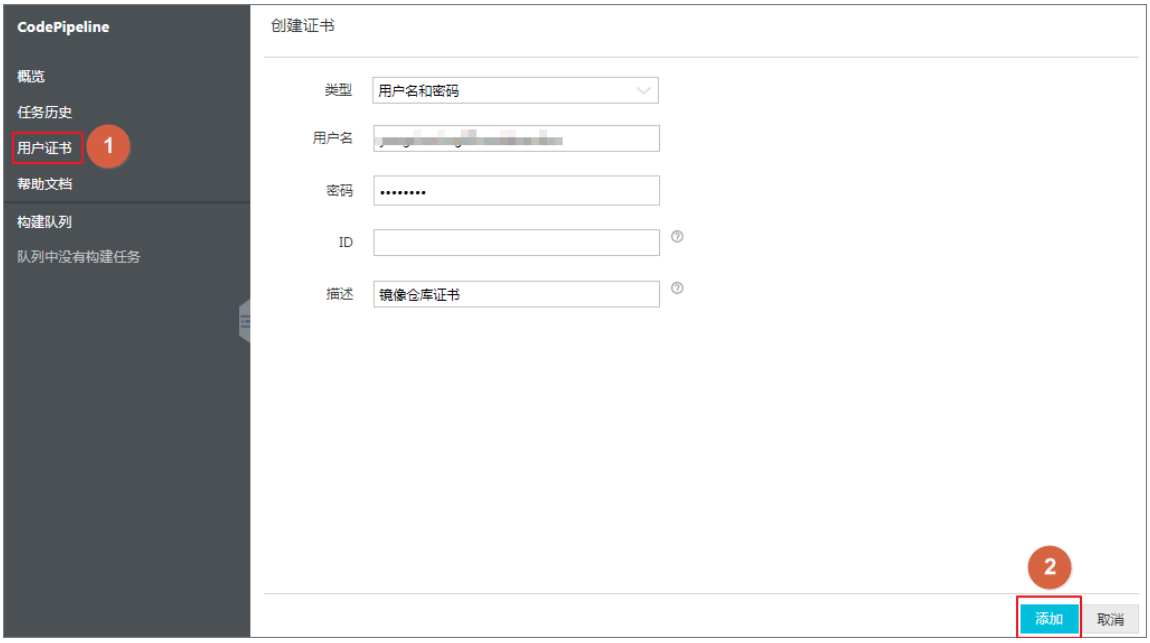
配置用户证书

1. **登录** *CodePipeline* **管理控制台。**
2. **单击左侧导航栏中的用户证书，然后单击右上角添加证书。**

3. 您需要配置用户证书，来对某些构建步骤进行授权。目前 CodePipeline 提供用户名和密码、Docker授权，Registry授权、SSH用户名和私钥四种类型的证书。

4. 本例中需要配置授权证书，具体如何配置可参见证书配置。在这里我们先准备两个证书以备后用。也可以在后续的构建配置中实时创建添加。

   · 第一个是镜像仓库的证书，选择用户名密码的方式；按下图填入以下内容，在这里最好添加描述信息，方便后续使用。ID不用填，在单击添加后会自动生成。



   · 第二个证书是kubernetes的证书，选择Docker授权的方式。

这里需要填入客户端key(key.pem)和客户端证书(cert.pem)，这两个内容可以在容器服务平台上获取。在容器服务平台上单击集群 > 管理，进入基本信息页面，在这里可以找到以上内容。如下图所示。

将上图中的client-certificate-data填入客户端证书(cert.pem)；client-key-data填入客户端key(key.pem)，单击确定完成配置。

修改 deployment.yaml 文件

在本例中，会执行两个构建步骤。

1. **通过镜像构建和发布，会将镜像推送到准备好的镜像仓库中，您可以修改 Dockerfile 文件，本例中不做修改。**
2. **部署到 Kubernetes 集群中。本例中使用 deployment.yaml 文件，为了验证镜像从构建、发布，再到创建应用的完整流程，要修改该示例项目中的 deployment.yaml 文件，将其镜像地址修改为本例中的镜像仓库地址** `registry.cn-hangzhou.aliyuncs.com/dev-testcs/codepipeline_registry`**。示例编排如下所示。**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: java-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: java-demo
  template:
    metadata:
      labels:
        app: java-demo
    spec:
      containers:
      - name: java-demo
        image: registry.cn-shanghai.aliyuncs.com/dev-testcs/java-
demo    #注意，修改为您自己镜像仓库地址
        imagePullPolicy: Always
        ports:
        - containerPort: 8080containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: java-demo
  namespace: default
spec:
  ports:
  - port: 80
    targetPort: 8080
```

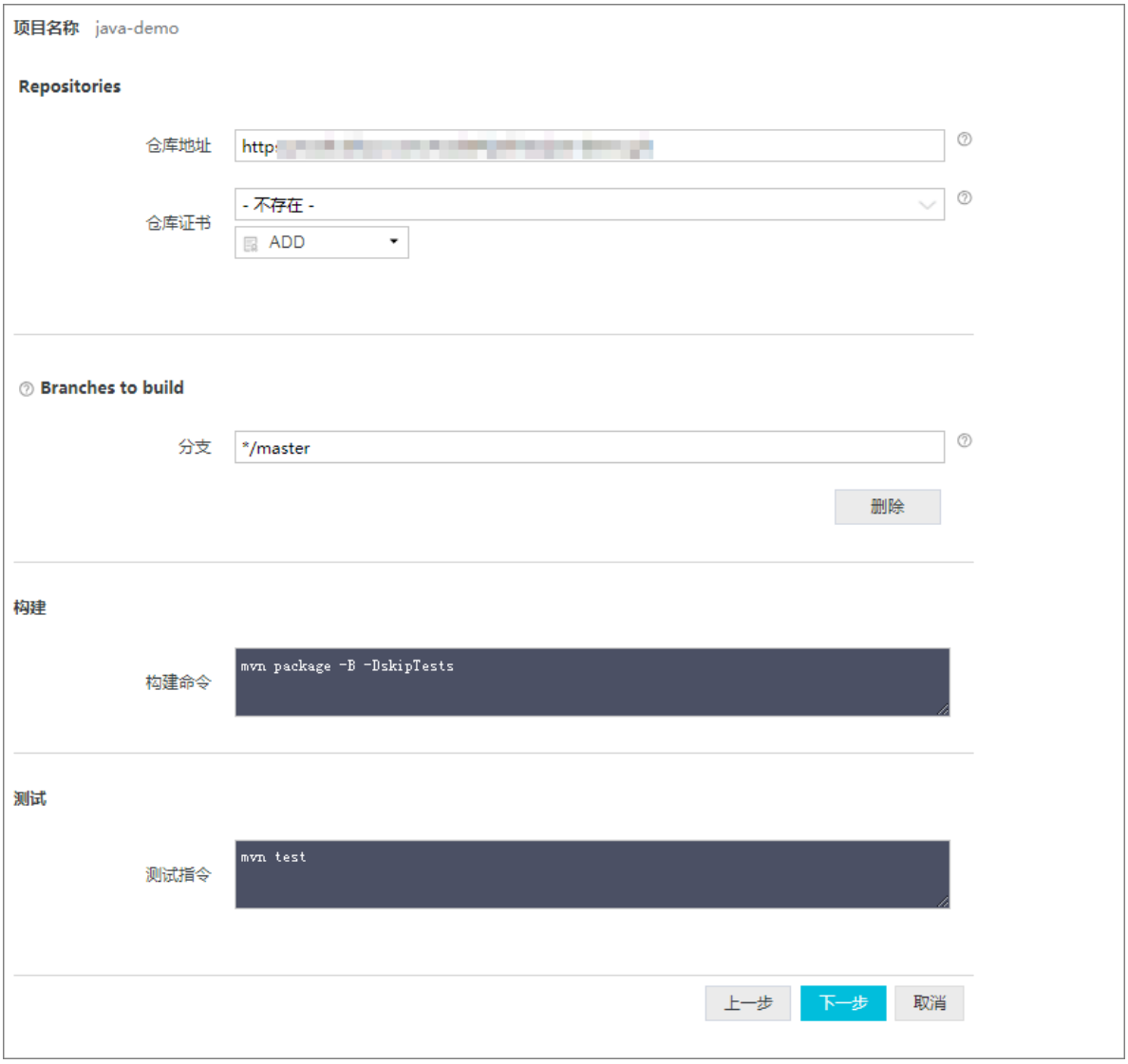
```
    name: java-demo
  selector:
    app: java-demo
  type: LoadBalancer
```

开始进行应用构建

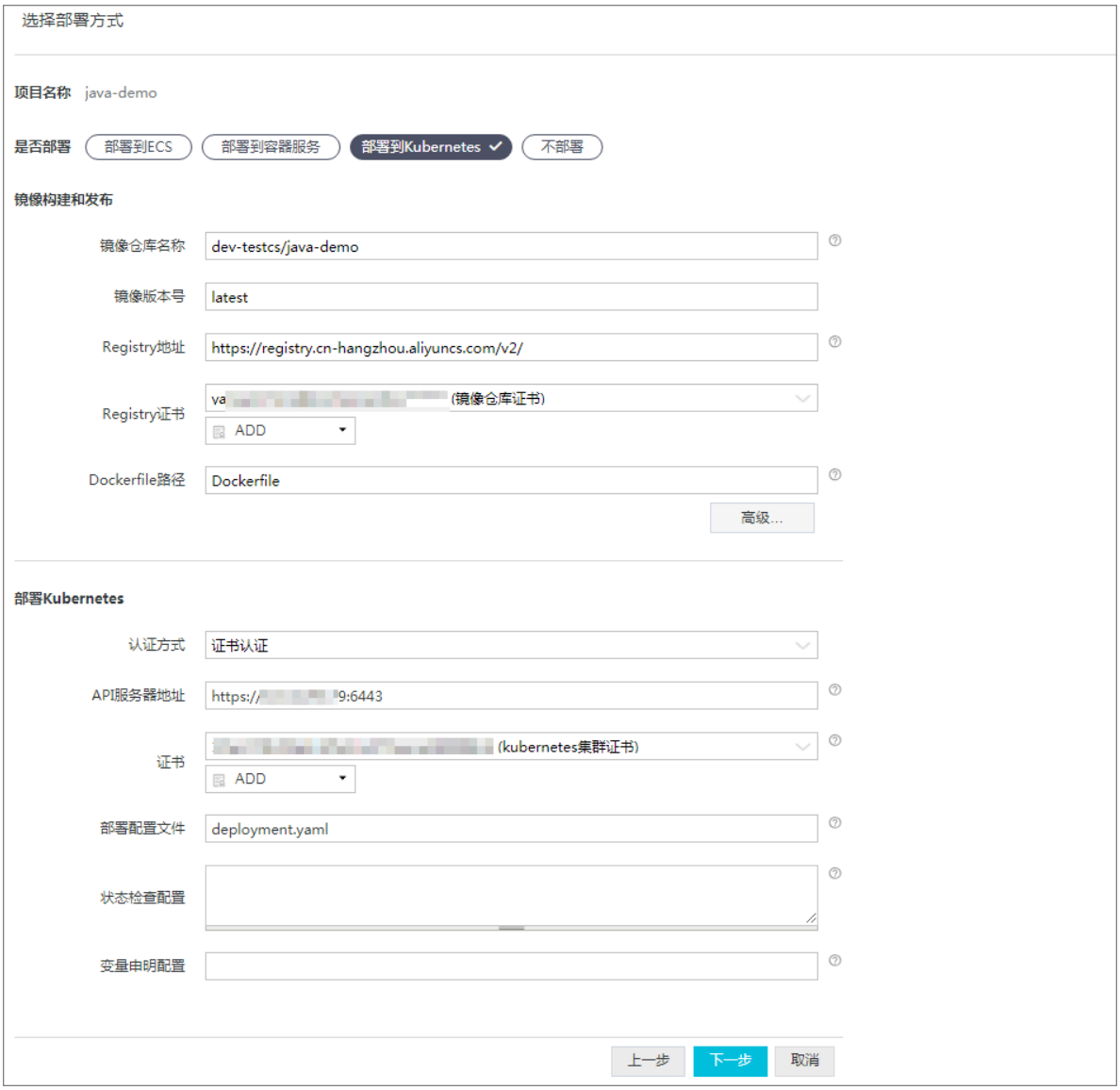
1. **登录 *CodePipeline* 管理控制台。**
2. **单击右上角 新建，开始构建一个新项目。本例中选择构建一个 java 类型的应用，名称为 java-demo，然后单击下一步。**
3. **配置项目参数，您可以直接输入 Repositories 地址，证书、分支、构建指令和测试指令。部分参数可以使用默认值。完成配置后，单击 下一步。**

说明:

如果您选择使用阿里云 code 进行源码管理，可跳过该步骤。后面可以直接绑定阿里云 code 代码仓库。CodePipeline 用户会直接绑定阿里云 Code 账户，自动列出您的代码仓库及其对应的分支或 Tag。



4. 选择部署到 kubernetes，对镜像构建与发布，部署 Kubernetes 这两个步骤进行参数配置，完成配置后，单击下一步。



- 镜像构建与发布。
  - 镜像仓库名称：镜像仓库名格式为`{namespace}/{image name}`。
  - 镜像版本号：自由设置，为会构建的镜像打上这个 tag，默认为latest。
  - Registry 地址：用来配置 Docker registry 地址。如果为空，默认使用 Docker hub registry；如果使用阿里云 registry， 需要根据您使用的镜像仓库的地域进行更改，其

他的都不需要更改，例如https://registry.cn-hangzhou.aliyuncs.com/v2/ 只需修改hangzhou这个地域。

- Registry 证书：您需要设置 Regostry 证书来添加授权信息。需要预先添加 Registry 授权类型的证书。
- Dockerfile 路径：填写 Dockerfile 文件在该项目工作空间的相对路径。如果该配置为空，则默认使用工作空间目录下命名为 Dockerfile 的文件。

· 部署 Kubernetes。

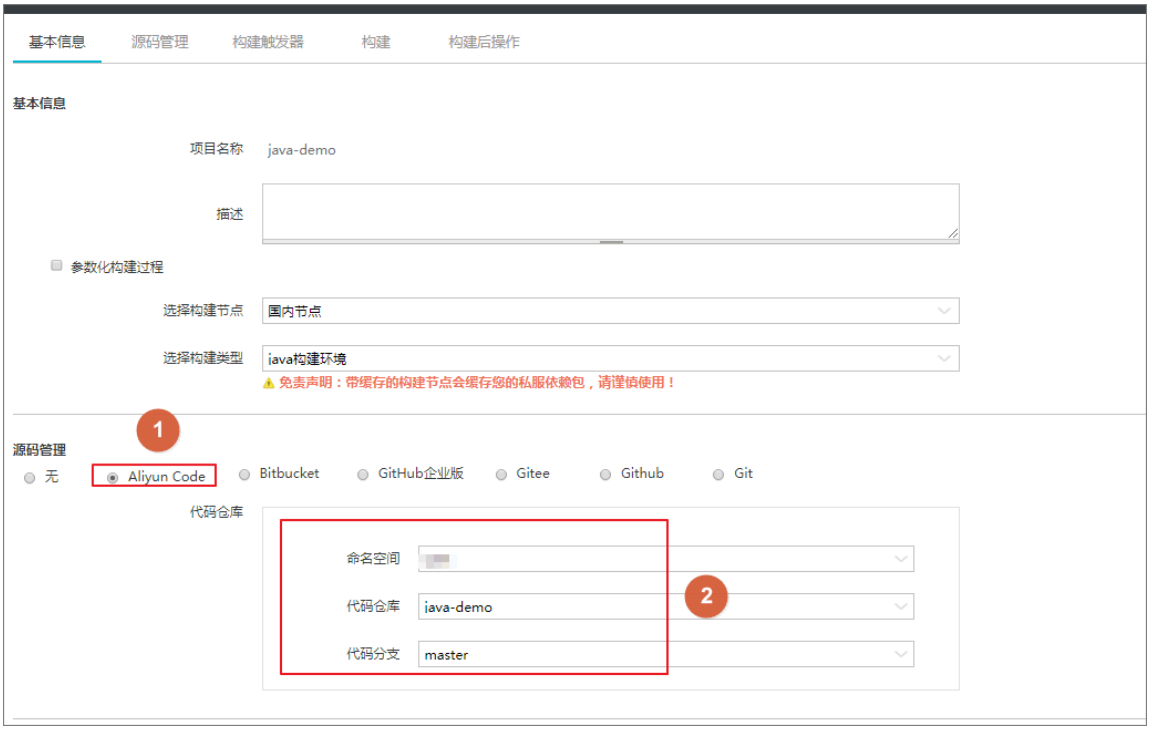
- 证书认证：CodePipeline 目前支持证书认证、用户名密码认证和 Token 认证三种认证方式。本例使用证书认证。
- API 服务器地址：即 API Server 公网连接地址，您可在容器服务控制台上获取，单击集群 > 管理，在连接信息中查看该地址。
- 证书：前面配置的 Docker 授权类型的证书。
- 部署配置文件：填写yaml格式的Kubernetes部署配置文件，本例中是根目录下的 deployment.yaml 编排文件。
- 状态检查配置：支持检验的 Kubernetes Kind：`pods,daemonsets,deployments,replicasets,replicationcontrollers,statefulsets`。如果检验的不是 `default` namespace 下的资源，请在首行填写 namespace 名称。
- 变量申明配置：支持系统环境变量，可以通过 ${JENKINS_URL}/env-vars.html/ 查看。如果使用了多个变量，请用”,”分隔。

5. 进入最终确认页面，在此页面中，大部分内容已经配置完毕，属于配置确认界面。还需要完成以下操作。

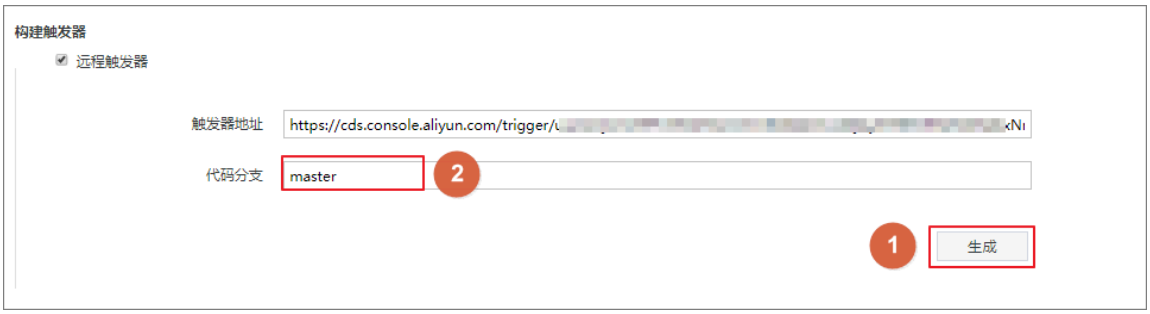
a) 在前面的步骤中，代码仓库管理没有配置，所以还需要在这里配置这一项，其他的都不需要配置。

说明:

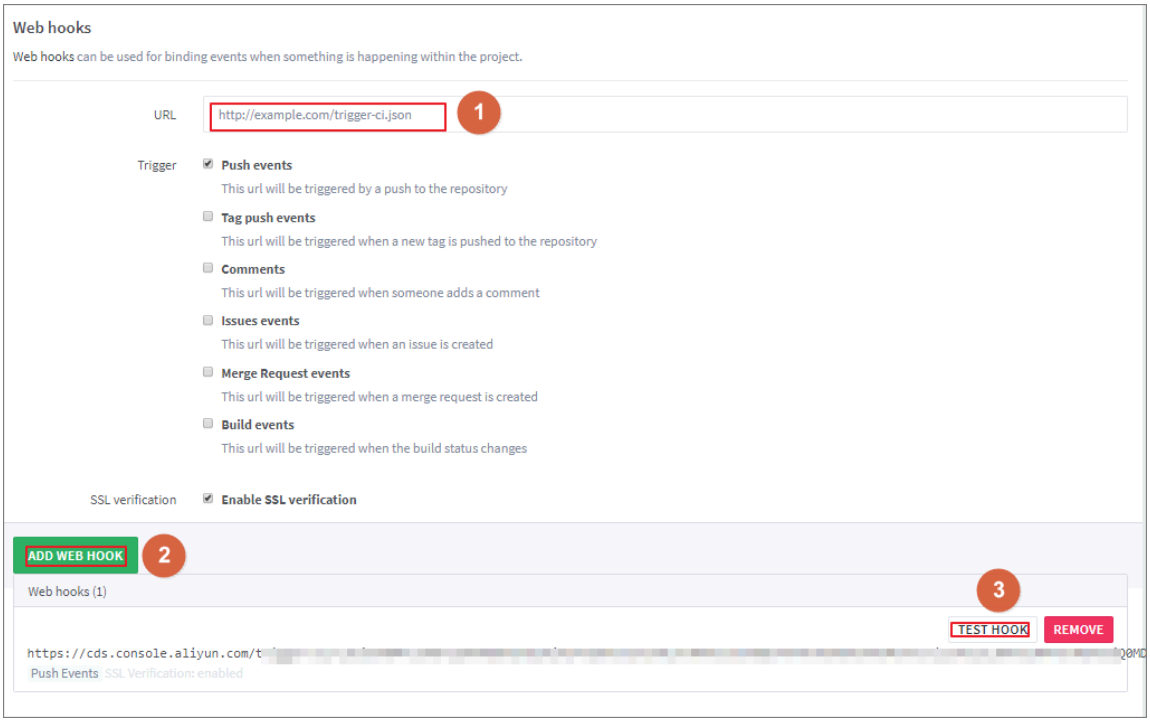
我们使用阿里云 code，会自动进行关联。



b) 您也可选择是否构建触发器，进行快速构建。生成并复制触发器地址，将其粘贴到浏览器中，按下回车键，浏览器窗口会提示您开始了一次构建。具体请参见 构建触发器。



这里，我们可以将该 url 填入到代码仓库的 webhook 中，当代码有变动时会通过该 url 触发 codepipeline 进行构建。Webhook 配置如下图所示。



6. 单击 提交。

7. 返回概览页面，在项目列表中，选择对应的项目，并单击右侧的执行构建。该项目会出现在构建队列下，单击该任务，进入项目详情页面。

8. 在构建历史下，单击对应的构建任务，然后单击左侧导航栏中的控制台输出，您可以查看该次构建过程输出的日志。

   本例中，输出日志最终提示成功创建对应的 deployment 对象。

```
08a01612ffca: Layer already exists
f715ed19c28b: Layer already exists
ceaed136256: Layer already exists
latest: digest: sha256:3bfb890d9fa366e0206a251620417825a511aafccf6d50b941aa98c7ecadd98e size: 3044
---------------------------------------------------------------

Apply config file.

kubectl apply -f /home/jenkins/workspace/java/deployment.yaml --kubeconfig=/home/jenkins/workspace/java/.kubeconfig
deployment "java-demo" created
service "java-demo" created
---------------------------------------------------------------

Start list nodes.

    addresses:
    - address: 192.168.0.177
    addresses:
    - address: 192.168.0.178
    addresses:
    - address: 192.168.0.176
    addresses:
    - address: 192.168.0.180
    addresses:

    - address: 192.168.0.181
    addresses:
    - address: 192.168.0.179
Finished: SUCCESS
```

9. 您可以登录 容器服务管理控制台，在目标集群右侧的应用 > 无状态 下看到成功部署了 java-demo 对象 。

3.2 使用Helm部署微服务应用

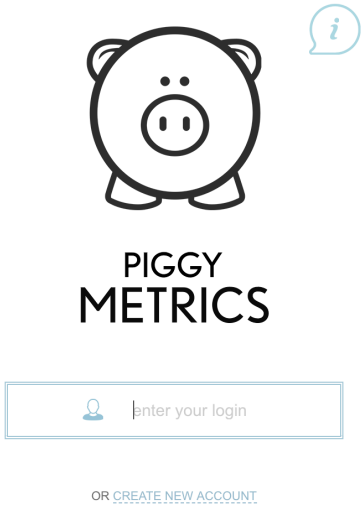
本文介绍如何将一个较复杂的应用部署到阿里云Kubernetes容器服务上，下面将从基础设施和应用部署的不同组合方式，来部署一个复杂的SpringCloud应用。

部署方式

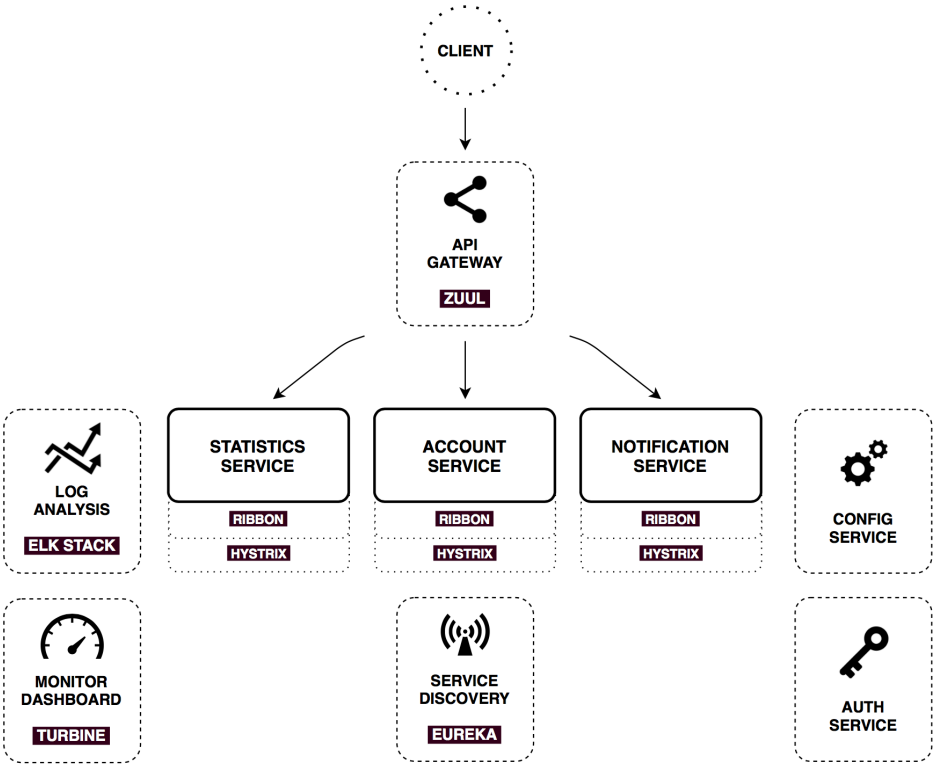
- 基础设施（Eureka，ConfigServer）和应用一起部署。
- 在容器服务上搭建好基础设施后，再部署应用。

示例应用PiggyMetrics

*PiggyMetrics*是github上的一个SpringCloud应用项目，Star数目3400多。这个项目主体采用Docker Compose部署，包含了完整的源代码以及构建好的容器镜像，是非常不错的SpringCloud容器化示例。



这个项目包含了3个业务微服务，分别是统计服务（Statistics Service）、账户服务（Account Service）和通知服务（Notification Service）。每个服务分别对应一个独立的MongoDB。微服务架构图示（采用作者原图）如下：



SpringCloud基础组件负责服务注册和registry服务（Eureka服务注册），config服务（配置管理），gateway（API网关，同时也是JavaScript Web界面），monitor服务（Hystrix Dashboard/Turbine）等。

场景一 用helm一键部署所有服务

1. **修改PiggyMetrics应用程序的Docker编写文件。**

   a) **切换配置文件版本。**

   PiggyMetrics的部署采用docker-compose YAML部署到单机，如果要部署到Kubernetes环境中，需要转换成为Kubernetes deployment YAML。

   说明:

   PiggyMetrics中的*docker compose*模版为2.1，kompose不支持该版本，所以需要把compose文件改为版本2。

   b) **在*docker-compose.yml*文件中。**

   去除kompose不支持的语法。

   ```
   depends_on:
         config:
           condition: service_healthy  # 不支持 condition
   ```

   增加Kubernetes server type annotation。

   ```
   depends_on:
     - config
   labels:
     kompose.service.type: loadbalancer
   ```

   c) **在*docker-compose.dev.yml*文件中，将PiggyMetrics应用程序使用的四个MongoDB数据库的外部端口更改为27017。**

   说明:

   PiggyMetrics应用包含四个MongoDB数据库，分别由 `auth-mongodb`、`account-mongodb`、`statistics-mongodb`和 `notification-mongodb`这四个字段定义。

   完整的YAML文件示例如下:

2. **使用Kompose生成Kubernetes配置文件来部署PiggyMetrics应用。**

   a) **设定PiggyMetrics部署所需的环境变量。**

   ```
   export NOTIFICATION_SERVICE_PASSWORD=passw0rd
   export CONFIG_SERVICE_PASSWORD=passw0rd
   export STATISTICS_SERVICE_PASSWORD=passw0rd
   export ACCOUNT_SERVICE_PASSWORD=passw0rd
   ```

```
export MONGODB_PASSWORD=passw0rd
```

b) 运行以下命令将compose文件转换为Kubernetes配置文件。

*kompose*工具可以一键将compose文件转换为Kubernetes配置文件。

```
kompose convert -f docker-compose.yml -f docker-compose.dev.yml -o
 piggymetrics -c
```

c) 运行`helm install`命令，在Kubernetes集群中部署PiggyMetrics应用。

例如，您可以运行以下命令，在命名空间`pm`中部署名为piggy的应用。

```
helm install --namespace pm --name piggy piggymetrics/
```

3. 确保本地的Helm客户端与远程的Helm客户端版本相同。

说明:

如果本地的Helm客户端与远程的Helm客户端版本不匹配时，会弹出如下错误提示：

```
Error: incompatible versions client[v2.14.1] server[v2.11.0]
```

如果没有显示此错误消息，您可以直接忽略此步骤。

- 如果是您是MAC系统，请运行以下命令获取Helm客户端的2.11.0版本。

```
brew unlink kubernetes-helm
brew install https://raw.githubusercontent.com/Homebrew/homebrew-
core/ee94af74778e48ae103a9fb080e26a6a2f62d32c/Formula/kubernetes-
helm.rb
```

- 如果您使用的是Windows或Linux系统，请安装相应的Helm客户端。

4. 部署完成后可以看输出成功信息。

   您可以在自己的Minikube上尝试，或者部署到阿里云容器服务Kubernetes版：*https://www.aliyun.com/product/kubernetes*。部署完成后进入服务列表页面，可以看到所有服务以及对应LoadBalancer类型Service对外暴露的访问地址及端口号。

名称	类型	创建时间	集群IP	内部端点	外部端点
account-mongodb	ClusterIP	2018-07-19 15:50:48		account-mongodb:27017 TCP	-
account-service	ClusterIP	2018-07-19 15:50:48		account-service:6000 TCP	-
auth-mongodb	ClusterIP	2018-07-19 15:50:48		auth-mongodb:27017 TCP	-
auth-service	ClusterIP	2018-07-19 15:50:48		auth-service:5000 TCP	-
config	ClusterIP	2018-07-19 15:50:48		config:8888 TCP	-
gateway	LoadBalancer	2018-07-19 15:50:48		gateway:4000 TCP gateway:30529 TCP	:4000
java-demo	LoadBalancer	2018-07-17 16:18:58		java-demo:80 TCP java-demo:30445 TCP	:80
kubernetes	ClusterIP	2018-07-02 14:43:47		kubernetes:443 TCP	-
monitoring	LoadBalancer	2018-07-19 15:50:48		monitoring:9000 TCP monitoring:31902 TCP monitoring:8989 TCP monitoring:31183 TCP	:9000 :8989
notification-mongodb	ClusterIP	2018-07-19 15:50:48		notification-mongodb:27017 TCP	-
notification-service	ClusterIP	2018-07-19 15:50:48		notification-service:8000 TCP	-
rabbitmq	NodePort	2018-07-19 15:50:48		rabbitmq:5672 TCP rabbitmq:30168 TCP rabbitmq:15672 TCP rabbitmq:32001 TCP	-
registry	LoadBalancer	2018-07-19 15:50:48		registry:8761 TCP registry:31563 TCP	:8761
statistics-mongodb	ClusterIP	2018-07-19 15:50:48		statistics-mongodb:27017 TCP	-
statistics-service	ClusterIP	2018-07-19 15:50:48		statistics-service:8888 TCP	-

- 单击registry service可以进入到PiggyMetrics的界面。

  PiggyMetrics是个人理财服务，用户输入收入和支出后可以展现漂亮的报表。
- 访问registry service，可以看到所有注册到Eureka Server上的服务。

DS Replicas

registry

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNT-SERVICE	n/a (1)	(1)	UP (1) - account-service-7fd4976bfc-2dq9p:account-service:6000
AUTH-SERVICE	n/a (1)	(1)	UP (1) - auth-service-7bdb99b5dc-kt7kd:auth-service:5000
GATEWAY	n/a (1)	(1)	UP (1) - gateway-77857d9c49-nhgsx:gateway:4000
NOTIFICATION-SERVICE	n/a (1)	(1)	UP (1) - notification-service-5d5859d7-w5hlz:notification-service:8
STATISTICS-SERVICE	n/a (1)	(1)	UP (1) - statistics-service-685fb8dc9f-6j7lv:statistics-service:7000

5. 您可以执行如下命令，删除PiggyMetrics。

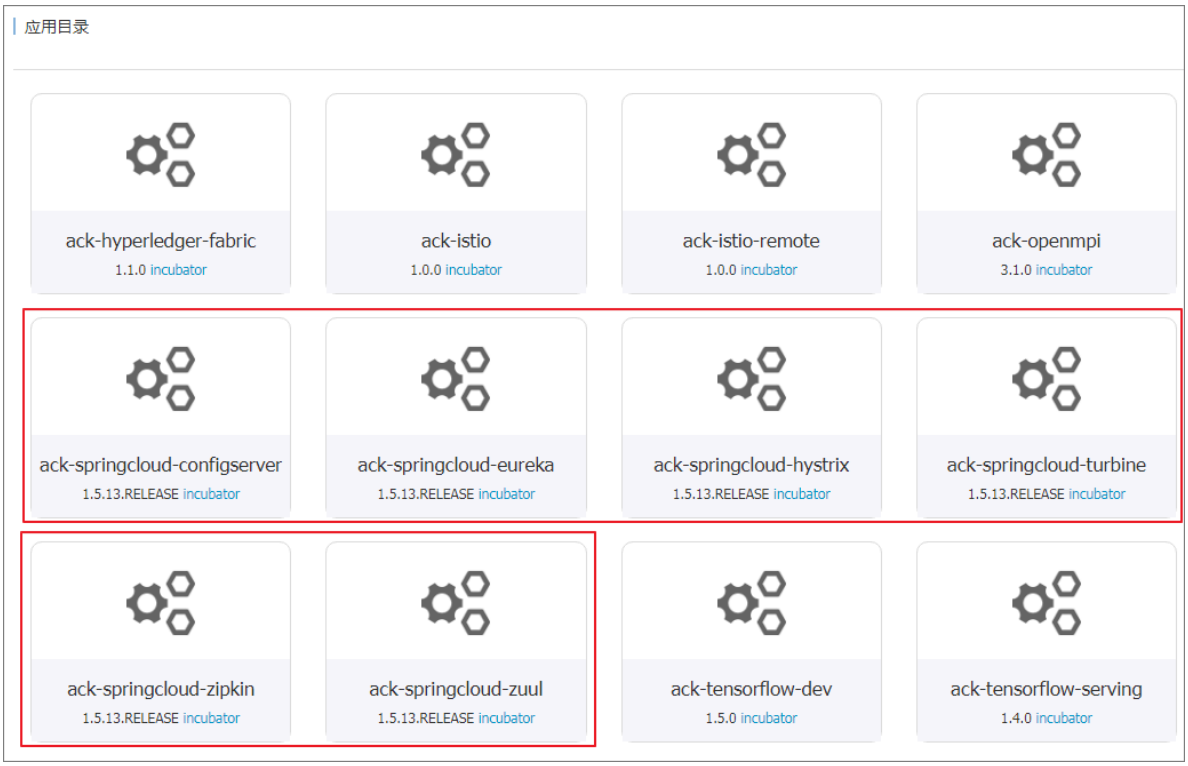
```
helm delete --purge piggymetrics
```

```
release "piggymetrics" deleted
```

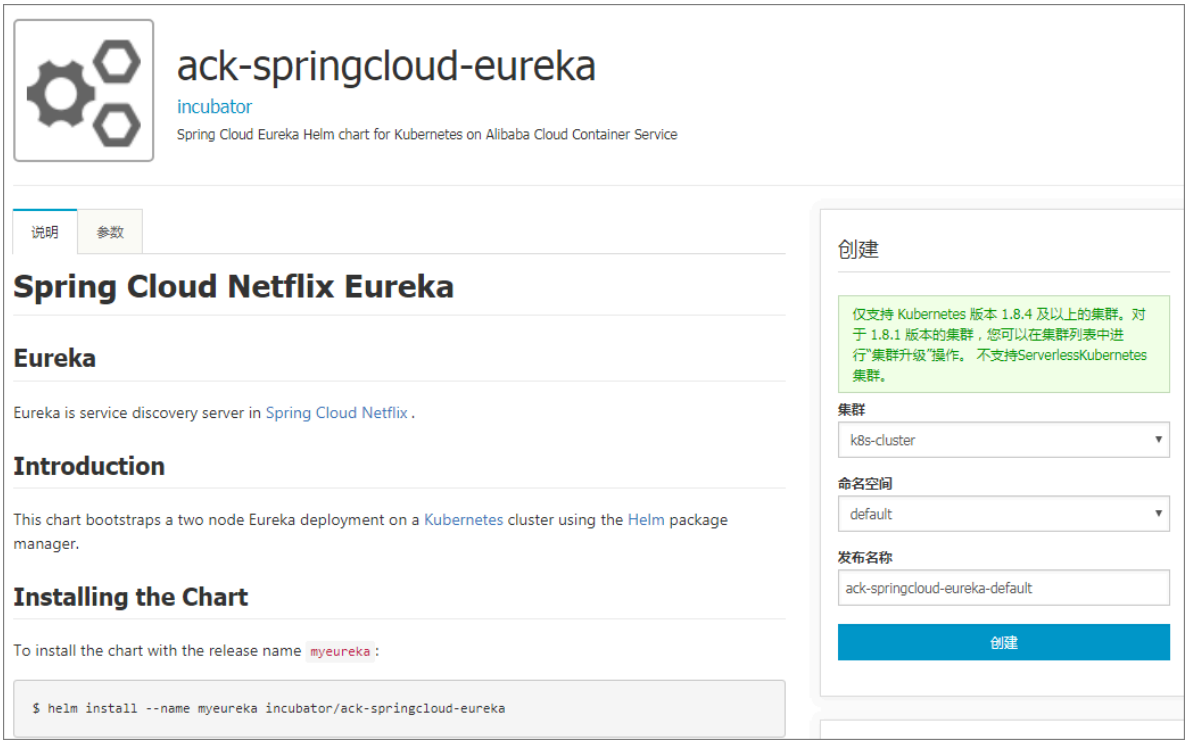
场景二 部署SpringCloud基础组件

上面您看到的是如何将全部基础组件（Eureka，Zuul，ConfigServer，Hystrix Dashboard）和业务应用（gateway，notification，statistics）都统一用一个helm chart部署成功。在实际工作中，更常见的情况是在集群中已经有了Eureka等基础组件，您只需部署和升级维护业务应用。

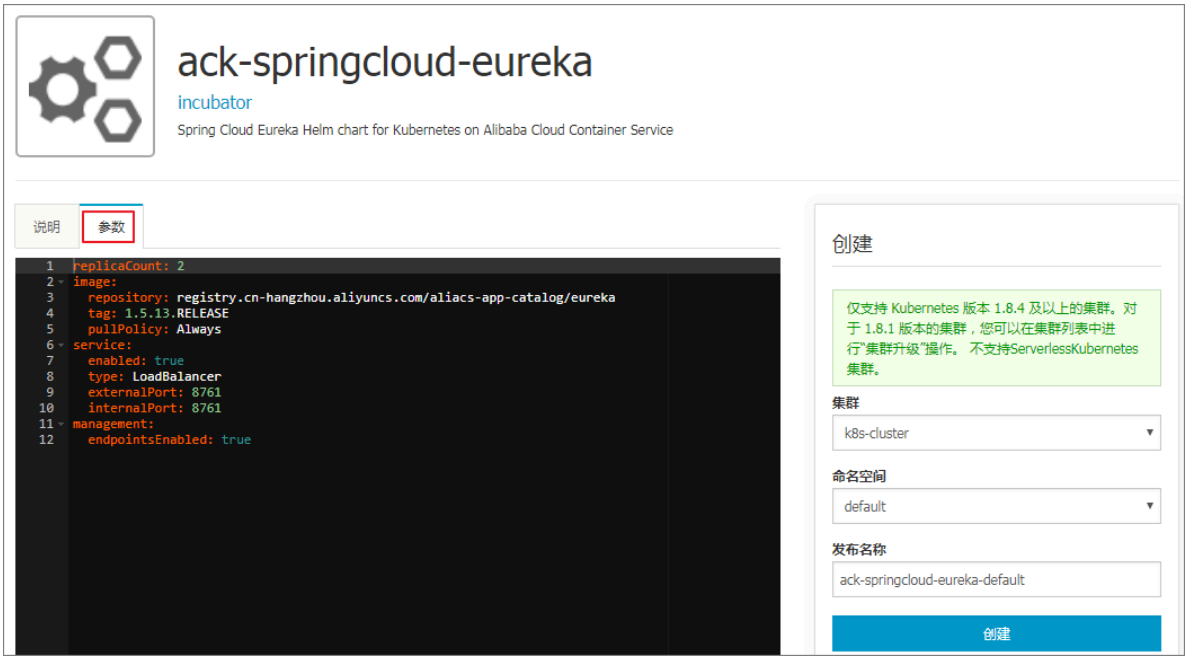
1. **在阿里云容器服务Kubernetes版中，单击左侧导航栏市场 > 应用目录，在右侧的应用目录中可以看到包含了SpringCloud基本组件。**



2. 单击ack-springcloud-eureka组件，进入如下界面：

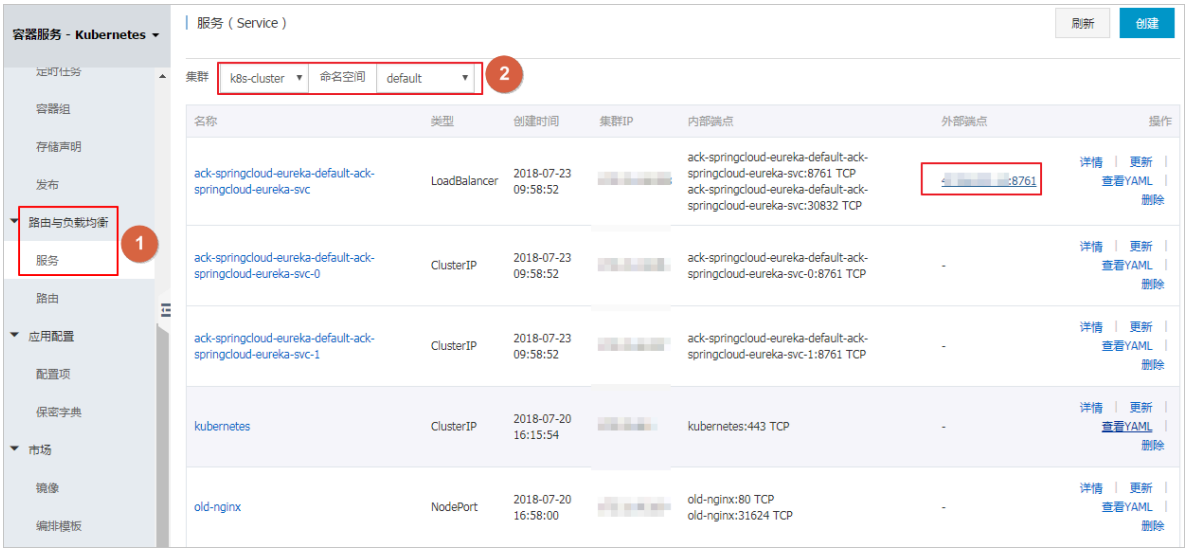


您也可以单击参数，进入参数页面，查看或更改配置：



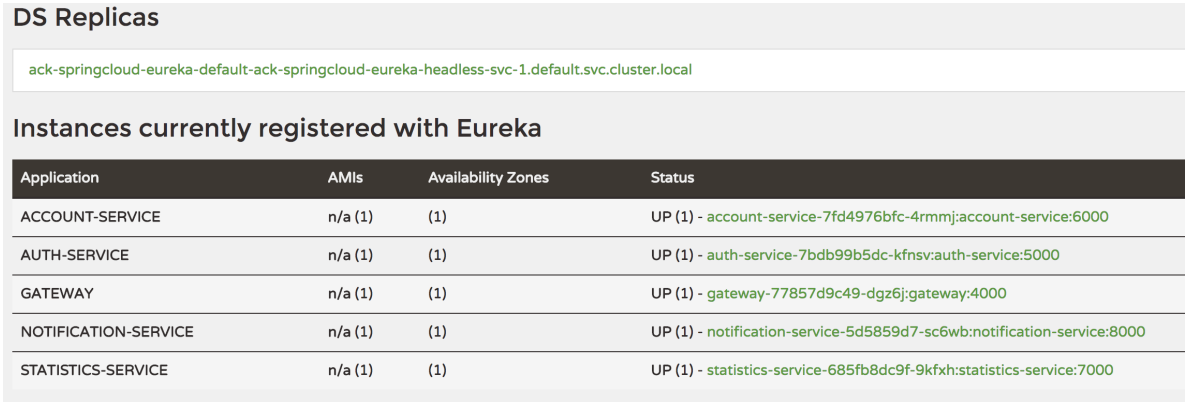
3. 在这里选择不改变任何参数，单击右侧创建，启动部署。

   部署成功后，在阿里云容器服务控制台中，单击左侧导航栏中路由与负载均衡 > 服务，选择集群和命名空间，进入服务列表页面。可以看到，EurekaServer有两个实例，对外暴露的服务地址为`ack-springcloud-eureka-default-ack-springcloud-eureka-svc`。



4. 您可以参见场景一部署一个没有Eureka的PiggyMetrics应用。

   所有服务启动成功后，访问Eureka，可以看到所有PiggyMetrics服务均已正确地注册到了EurekaServer中。



   PiggyMetrics应用已经部署到了包含EurekaServer的环境上。访问`GATEWAY`即可看到熟悉的登录界面。

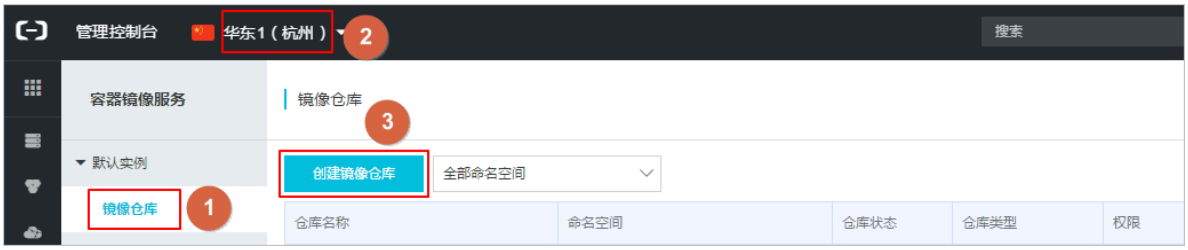
3.3 使用私有镜像仓库创建应用

在很多场景下，用户需要用到私有镜像仓库中的镜像进行应用的部署，在这篇文档中我们使用阿里云镜像仓库服务创建一个私有的镜像仓库，并且创建一个使用该私有镜像仓库的应用。

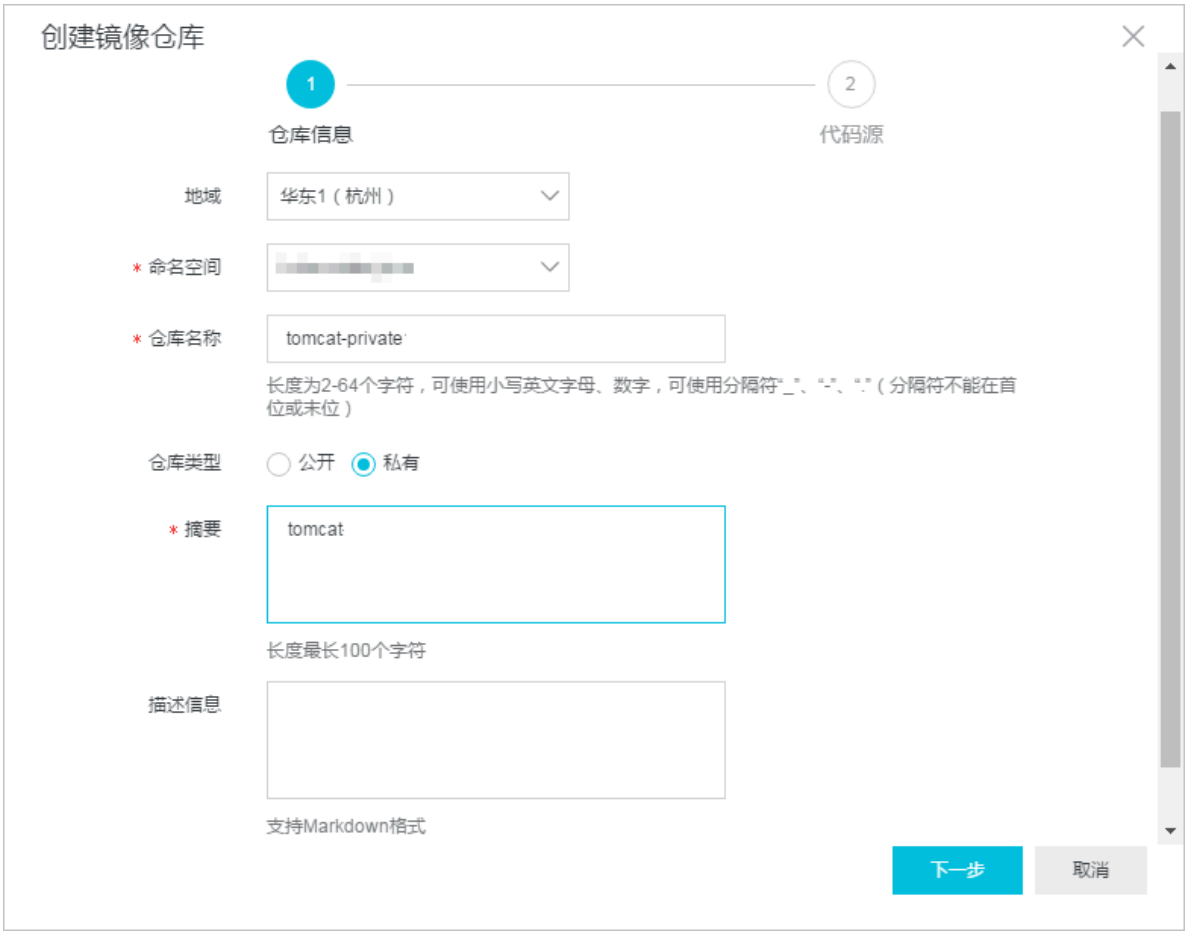
创建私有镜像库

如果您是首次使用阿里云容器镜像服务，会弹出提示需要您设置 Registry 登录密码，请单击前往开通，并根据界面提示，设置 Registry 登录密码。

1. **登录容器服务管理控制台。**
2. **在左侧导航栏中单击镜像仓库，选择所需的地域，然后单击创建镜像仓库。**

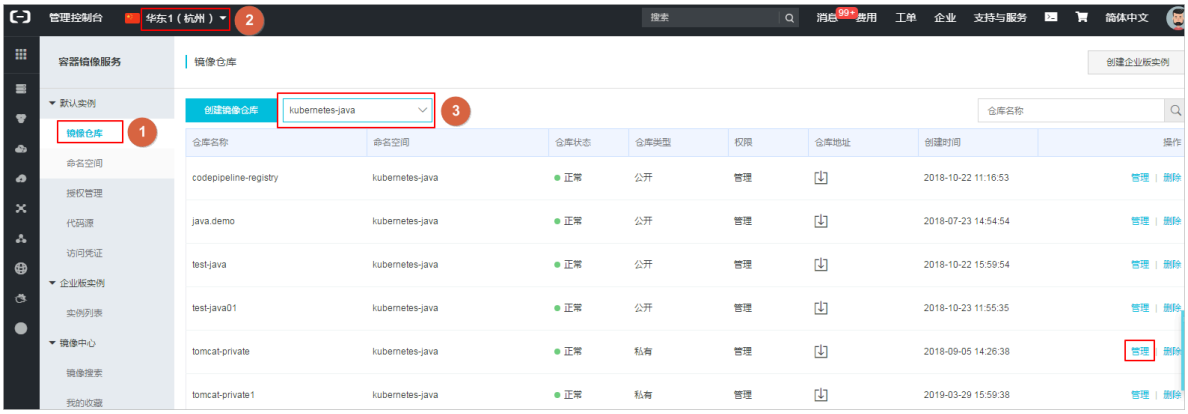


3. **在弹出的对话框中配置镜像仓库，设置命名空间、仓库名称、摘要和仓库类型，本例选择私有镜像仓库类型。然后单击下一步。**



4. 在设置代码源对话框中，将代码源设为本地仓库。

5. 在镜像仓库列表下，选择所需的地域和命名空间，您可看到创建完成的镜像仓库，单击右侧的管理。



6. 进入仓库管理界面，单击基本信息，您可以查看如何使用该私有镜像仓库。

7. 在Linux环境登录镜像仓库，执行如下命令，将本地的镜像上传到私有镜像仓库中。

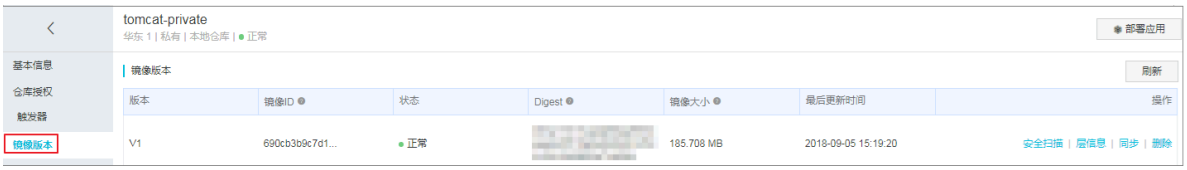
```
$ sudo docker login --username=abc@aliyun.com registry.cn-hangzhou.
aliyuncs.com
```

```
 Password
                          ## 镜像仓库独立登录密码：
 Login Succeed

$ docker images
                            #以tomcat为例
REPOSITORY
        TAG                 IMAGE ID            CREATED
SIZE
tomcat
        latest              2d43521f2b1a        6 days ago
463MB

$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/XXX/
tomcat-private:[镜像版本号]
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/XXX/tomcat-
private:[镜像版本号]
```

结果如下：

```
The push refers to a repository [registry.cn-hangzhou.aliyuncs.com/
XXX/tomcat-private]
9072c7b03a1b: Pushed
f9701cf47c58: Pushed
365c8156ff79: Pushed
2de08d97c2ed: Pushed
6b09c39b2b33: Pushed
4172ffa172a6: Pushed
1dccf0da88f3: Pushed
d2070b14033b: Pushed
63dcf81c7ca7: Pushed
ce6466f43b11: Pushed
719d45669b35: Pushed
3b10514a95be: Pushed
V1: digest: sha256:cded14cf64697961078aedfdf870e704a5227018
8c8194b6f70c778a8289**** size: 2836
```

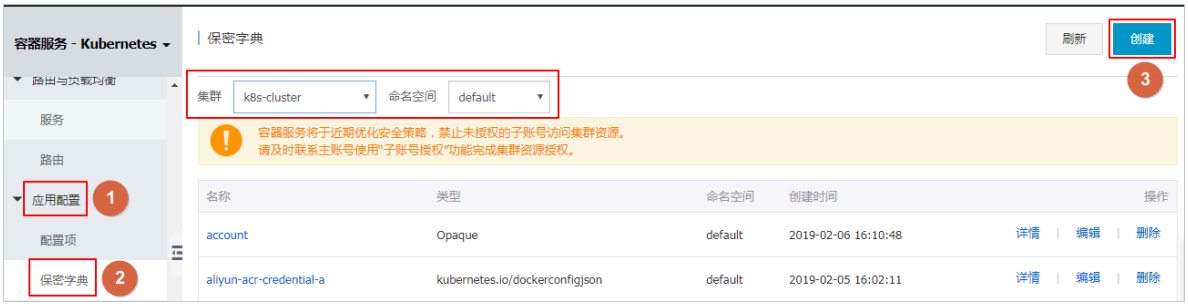
8. **返回该镜像仓库详情页，单击左侧导航栏中的镜像版本，您可以看到镜像已成功上传，并可查看镜像的版本信息。**



创建私有镜像仓库登录密钥类型的密钥

1. **在 Kubernetes 菜单下，单击左侧导航栏中的应用配置 > 保密字典，进入保密字典页面。**

2. 选择所需的集群和命名空间，单击右上角的创建。

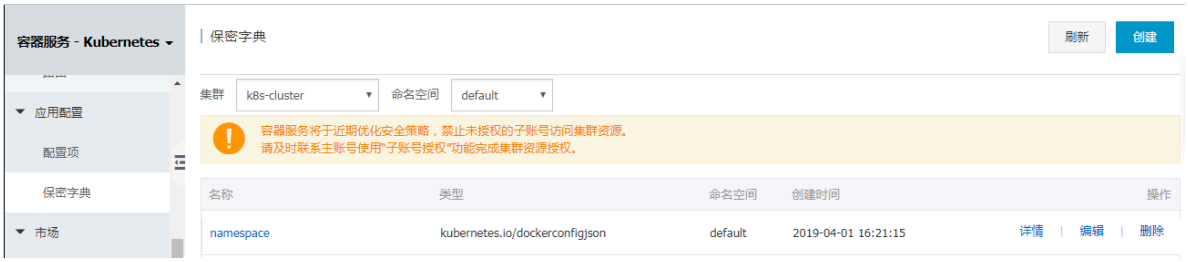


3. 配置新的保密字典。

说明:

在这里不能用容器服务控制台上的保密字典进行secret的创建。

4. 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。



您也可以*#unique_28*创建私有镜像仓库登录密钥类型的密钥。

通过私有镜像仓库创建应用

1. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，进入无状态（Deployment）页面。

2. 选择所需的集群和命名空间，单击右上角的使用模板创建。

说明:

您也可以通过单击使用镜像创建来创建应用。请参见*#unique_23*。

3. 示例模板选择自定义，并将以下内容复制到模板中，单击创建。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: private-image
  nameSpace: default
  labels:
    app: private-image
spec:
  replicas: 1
  selector:
    matchLabels:
      app: private-image
  template:
    metadata:
      labels:
        app: private-image
    spec:
      containers:
      - name: private-image
        image: registry.cn-hangzhou.aliyuncs.com/xxx/tomcat-private:
latest
        ports:
        - containerPort: 8080
      imagePullSecrets:
      - name: regsecret
```

更多内容请参见kubernetes官方文档使用私有仓库。