阿里云
容器服务Kubernetes版

Kubernetes集群用户指南

文档版本：20190917

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。
*## *	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

法律声明.....I
通用约定.....I
1 简介..... 1
1.1 概述.....1
1.2 阿里云 Kubernetes vs. 自建 Kubernetes..... 1
2 授权管理..... 4
2.1 角色授权.....4
2.2 使用子账号.....7
2.3 Kubernetes 集群访问控制授权概述..... 9
2.4 自定义RAM授权策略.....11
2.5 子账号RBAC权限配置指导.....13
3 集群管理.....21
3.1 查看集群概览..... 21
3.2 创建集群..... 22
3.2.1 创建Kubernetes 集群..... 22
3.2.2 创建Kubernetes 托管版集群..... 35
3.2.3 创建Windows Kubernetes 集群..... 46
3.2.4 创建 Kubernetes 边缘托管版集群..... 53
3.3 管理与访问集群..... 60
3.3.1 通过 kubectl 连接 Kubernetes 集群..... 61
3.3.2 在CloudShell上通过kubectl管理Kubernetes集群..... 62
3.3.3 SSH访问Kubernetes集群..... 62
3.3.4 使用ServiceAccount Token访问Kubernetes集群.....67
3.3.5 SSH密钥对访问Kubernetes集群..... 69
3.4 升级集群..... 70
3.4.1 升级集群.....70
3.4.2 升级集群的GPU节点的NVIDIA驱动.....74
3.4.3 升级安装Kubernetes集群的metrics-server组件..... 79
3.4.4 升级系统组件.....82
3.5 扩容集群.....84
3.6 删除集群.....87
3.7 集群管理最佳实践..... 89
3.7.1 ECS选型.....89
3.7.2 高可靠推荐配置.....91
3.7.3 更新Kubernetes集群已过期的证书..... 96
3.7.4 更新Kubernetes集群即将过期的证书..... 97
3.7.5 VPC下 Kubernetes 的网络地址段规划..... 104
3.8 集群管理FAQ..... 107
4 多集群管理.....113

4.1 概述 113
4.2 部署集群联邦 114
4.3 部署多集群联邦应用 117
4.4 接入外部Kubernetes集群 118
5 节点管理 123
5.1 添加边缘节点 123
5.2 添加已有节点 126
5.3 查看节点列表 128
5.4 监控节点 129
5.5 管理节点标签 130
5.6 设置节点调度 131
5.7 设置节点自治 131
5.8 移除节点 132
5.9 查看节点资源请求量/使用量 135
5.10 Kubernetes GPU集群支持GPU调度 135
5.11 利用阿里云Kubernetes的GPU节点标签进行调度 140
5.12 为容器服务的Docker增加数据盘 144
5.13 集群节点挂载数据盘 146
6 应用管理 148
6.1 镜像创建无状态Deployment应用 148
6.2 镜像创建有状态StatefulSet应用 158
6.3 镜像创建Job类型应用 178
6.4 通过 Kubernetes Dashboard 创建应用 189
6.5 通过编排模板创建Linux应用 193
6.6 通过编排模板创建Windows应用 196
6.7 通过命令管理应用 199
6.8 利用 Helm 简化应用部署 200
6.9 使用应用触发器重新部署应用 207
6.10 指定节点调度 209
6.11 查看容器组 212
6.12 变更容器配置 215
6.13 手动伸缩容器应用 216
6.14 创建服务 218
6.15 查看服务 223
6.16 更新服务 225
6.17 删除服务 228
6.18 创建镜像 229
6.19 查看镜像列表 230
6.20 使用镜像密钥 231
6.21 免密拉取镜像 235
7 工作流 243
7.1 创建工作流 243
7.2 Workflow 示例模板 246
7.3 开启Workflow UI 262

7.4 AGS命令行帮助 265
7.5 AGS帮助示例 267
8 网络管理 278
8.1 概述 278
8.2 通过负载均衡（Server Load Balancer）访问服务 281
8.3 Ingress 支持 311
8.4 Ingress 访问日志分析与监控 317
8.5 路由配置说明 326
8.6 通过界面创建路由（Ingress） 329
8.7 查看路由 340
8.8 变更路由 341
8.9 删除路由 343
8.10 如何使用Terway网络插件 344
8.11 为容器（Pod）分配弹性网卡（ENI） 345
8.12 使用网络策略（Network Policy） 347
8.13 使用Host网络 352
9 配置项及密钥管理 353
9.1 创建配置项 353
9.2 在容器组中使用配置项 357
9.3 查看配置项 361
9.4 修改配置项 361
9.5 删除配置项 366
9.6 创建密钥 368
9.7 在容器组中使用密钥 369
9.8 查看密钥 372
9.9 编辑密钥 373
9.10 删除密钥 374
10 存储管理-Flexvolume 376
10.1 概述 376
10.2 安装插件 376
10.3 云盘存储卷 381
10.3.1 云盘存储卷使用说明 381
10.3.2 使用静态云盘卷 381
10.3.3 使用动态云盘卷 385
10.3.4 通过控制台使用动态云盘卷 388
10.4 NAS储存卷 392
10.4.1 NAS存储卷使用说明 392
10.4.2 Flexvolume驱动-静态NAS卷 394
10.4.3 Flexvolume驱动-动态NAS卷 398
10.4.4 NFS驱动-静态NAS卷 400
10.4.5 NFS驱动-动态NAS卷 402
10.5 OSS储存卷 403
10.5.1 OSS存储卷使用说明 404
10.5.2 使用OSS静态卷 404

10.6 创建持久化存储卷声明.....409
10.7 使用持久化存储卷声明.....411
10.8 存储卷常见问题..... 413
10.9 存储卷最佳实践..... 416
10.9.1 有状态服务-静态云盘使用最佳实践..... 416
10.9.2 如何使用BYOK创建加密云盘.....421
11 存储管理-CSI..... 424
11.1 概述..... 424
11.2 安装插件..... 424
11.3 云盘存储卷..... 431
11.3.1 云盘存储卷使用说明..... 431
11.3.2 静态云盘卷..... 432
11.3.3 动态云盘卷..... 434
11.4 NAS存储卷..... 436
11.4.1 NAS卷使用说明.....436
11.4.2 静态NAS卷.....437
11.5 OSS存储卷.....438
11.5.1 OSS卷使用说明..... 438
11.5.2 静态OSS卷..... 439
12 日志管理..... 442
12.1 概述..... 442
12.2 查看集群日志..... 442
12.3 使用日志服务进行Kubernetes日志采集..... 443
12.4 利用 Log-Pilot + Elasticsearch + Kibana 搭建 kubernetes 日志解决方案..... 454
12.5 为 Kubernetes 和日志服务配置 Log4JAppender..... 464
13 监控管理..... 470
13.1 基础资源监控..... 470
13.2 应用性能监控..... 473
13.3 架构感知监控..... 479
13.4 事件监控..... 481
13.5 开源Prometheus监控..... 489
14 安全管理..... 501
14.1 概述..... 501
14.2 Kube-apiserver审计日志..... 502
14.3 在Kubernetes中实现HTTPS安全访问.....514
14.4 更新即将过期证书.....523
14.5 漏洞修复公告..... 525
14.5.1 修复runc漏洞CVE-2019-5736的公告.....525
14.5.2 修复Kubernetes Dashboard漏洞*CVE-2018-18264*的公告.....527
14.5.3 修复Kubernetes漏洞*CVE-2018-1002105*公告..... 528
14.5.4 修复Kubectl cp漏洞*CVE-2019-11246*的公告..... 529
14.5.5 修复Kubectl cp漏洞*CVE-2019-11249*的公告..... 530
14.6 集群证书更新说明.....531

14.7 安全组常见问题..... 534
15 发布管理..... 536
15.1 基于Helm的发布管理..... 536
15.2 在阿里云容器服务Kubernetes上使用分批发布..... 540
16 命名空间管理..... 544
16.1 创建命名空间..... 544
16.2 设置资源配额和限制..... 546
16.3 编辑命名空间..... 549
16.4 删除命名空间..... 550
17 服务网格..... 552
17.1 Istio管理..... 552
17.1.1 概述..... 552
17.1.2 部署Istio..... 554
17.1.3 更新Istio..... 558
17.1.4 删除Istio..... 559
17.1.5 Istio组件升级..... 560
17.1.6 设置 Sidecar 自动注入..... 562
17.2 服务网关..... 563
17.2.1 部署服务网关..... 563
17.2.2 管理服务网关..... 565
17.3 基于Istio实现跨区域多集群部署..... 566
17.4 绑定虚拟服务..... 571
17.5 流量管理..... 572
17.6 Istio 常见问题..... 581
17.7 Istio最佳实践..... 584
17.7.1 在Kubernetes上基于Istio实现Service Mesh智能路由..... 584
17.7.2 启用Istio CoreDNS..... 591
17.7.3 使用阿里云容器服务部署Bookinfo示例..... 596
18 Knative 管理..... 609
18.1 概述..... 609
18.2 部署 Knative..... 610
18.3 部署组件..... 612
18.4 创建 Knative 服务..... 613
18.5 创建修订版本..... 616
18.6 部署 Serving Hello World 应用..... 619
18.7 在Knative 上实现 Tracing 分布式追踪..... 621
18.8 在Knative 上实现日志服务..... 624
18.9 在Knative 上实现监控告警..... 627
18.10 Knative 自定义域名..... 634
18.11 在 Knative 上实现 GitHub 事件处理..... 635
18.12 删除修订版本..... 639
18.13 删除 Knative 服务..... 640
18.14 卸载组件..... 641
18.15 卸载 Knative..... 642

19 模板管理..... 644
19.1 创建编排模板..... 644
19.2 更新编排模板..... 647
19.3 另存编排模板..... 650
19.4 下载编排模板..... 651
19.5 删除编排模板..... 653
20 应用目录管理..... 655
20.1 应用目录概述..... 655
20.2 查看应用目录列表.....656
21 服务目录管理..... 657
21.1 概述..... 657
21.2 开通服务目录..... 657
22 弹性伸缩..... 660
22.1 使用HPA弹性伸缩容器.....660
22.2 节点自动伸缩..... 664
22.3 虚拟节点..... 674

1 简介

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。

1.2 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

功能	说明
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。

2 授权管理

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#unique_8。
- 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#unique_9。

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

权限名称（Action）	权限说明
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

2.2 使用子账号

您可通过子账号使用容器服务。

前提条件

对于2018年12月7日后创建的集群，需要该子账号对应的主账号开通弹性伸缩服务 ESS，可参考#unique_11。

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

访问容器服务的权限分为 RAM 授权和 RBAC 授权两部分。详细请参见#unique_12。

说明:
进行子账号集群RBAC授权前，您需要确保目标集群至少已经被授予RAM只读权限。

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。

2.3 Kubernetes 集群访问控制授权概述

本文总体介绍容器服务 Kubernetes 集群访问控制授权的组成及如何进行授权。

容器 Kubernetes集群授权分为 RAM 授权和 RBAC 授权两部分。

RAM授权

RAM 授权作用于集群管理接口的访问控制，当您需要对集群进行可见性、扩缩容、添加节点等操作时，需要进行RAM授权。详细权限请参#unique_15/unique_15_Connect_42_table_w5v_xxv_93g。

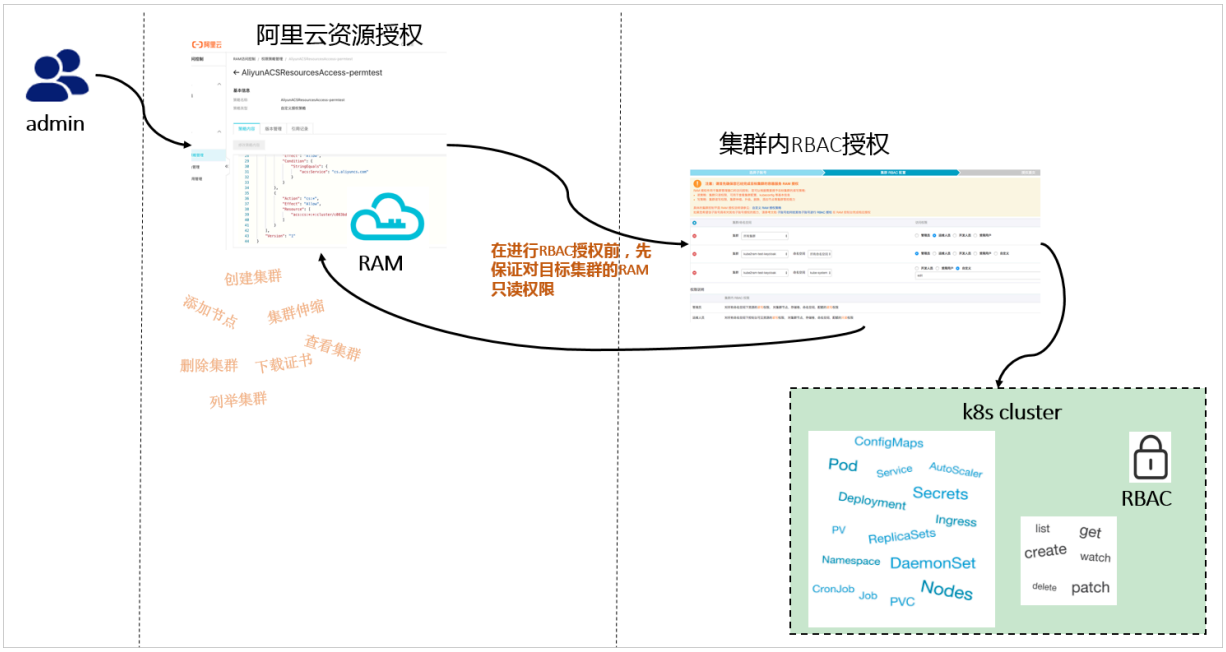
RBAC 授权

容器服务 Kubernetes 的 RBAC 授权是指用户所有调用 apiserver 访问 Kubernetes 集群内资源模型的访问控制，请参考授权概述，你可以通过容器服务给子账号授予如下预置角色：

表 2-1: 角色权限说明

角色	集群内RBAC权限
管理员	对所有命名空间下所有资源的读写权限
运维人员	对所有命名空间下控制台可见资源的读写权限，对集群节点，存储卷，命名空间，配额的只读权限
开发人员	对所有命名空间或所选命名空间下控制台可见资源的读写权限
受限用户	对所有命名空间或所选命名空间下控制台可见资源的只读权限
自定义	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

如何授权



- 在进行子账号集群RBAC授权前请首先完成对集群管控能力的RAM授权，您可以根据需要授予目标集群的读写策略：
  - 读策略：用于查看集群配置，kubeconfig 等基本信息
  - 写策略：包含集群伸缩、升级、删除，添加节点等集群管控能力

  在提交RBAC授权前，您需要确保目标集群已经被授予RAM只读权限，策略参考如下：

```
{
  "Statement": [
    {
      "Action": "cs:Get*",
      "Effect": "Allow",
      "Resource": [
        "acs:cs:*:*:cluster/<yourclusterID>"
      ]
    }
  ],
  "Version": "1"
}
```

  具体的RAM授权步骤请参见#unique_14。
- 当您完成RAM授权后，可参见#unique_16完成集群内 Kubernetes 资源模型访问的 RBAC 授权。

2.4 自定义RAM授权策略

本文档介绍如何创建自定义授权策略。下面以授予子账号查询、扩容和删除集群的权限为例进行说明。

前提条件

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。

背景信息

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。例如，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

操作步骤

1. 使用具有RAM权限的账号登录RAM管理控制台。
2. 单击左侧导航栏的权限管理 > 权限策略管理，进入权限策略管理页面。
3. 单击新建授权策略，进入新建自定义权限策略页面。
4. 填写策略名称，配置模式选择脚本配置，并在策略内容中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。

5. 编写完毕后，单击确定。

   预期结果

   返回权限策略管理页面，在搜索框中搜索策略名或备注，可以看到您授权的自定义的策略。

相关参考

表 2-2: 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

Action	说明
GetClusterEndpoint	查看集群接入点地址
GetClusterCerts	下载集群证书
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
UpdateClusterTags	编辑集群标签

2.5 子账号RBAC权限配置指导

本文主要介绍如何配置子账号对应的Kubernetes集群内RBAC权限的指导。

配置说明

- 您需要拥有一个阿里云主账号，并有一个或若干个子账号。
- 请先确保目标集群在RAM控制台中至少已被授予集群管理只读授权。
- RAM授权作用于集群管理接口的访问控制，您可以根据需要授予目标集群的读写策略：
  - 读策略：集群只读权限，可用查看集群配置，kubeconfig 等基本信息
  - 写策略： 集群读写权限，集群伸缩、升级、删除，添加节点等集群管控能力

  详细请参见#unique_19。

- 如果某子账号在目标集群或命名空间上被授予预置的管理员角色或自定义中的cluster-admin角色，同时满足#unique_19中的RAM授权配置， 则该子账号在对应的目标集群或命名空间上可以给其他子账号授权。
- 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。

公告

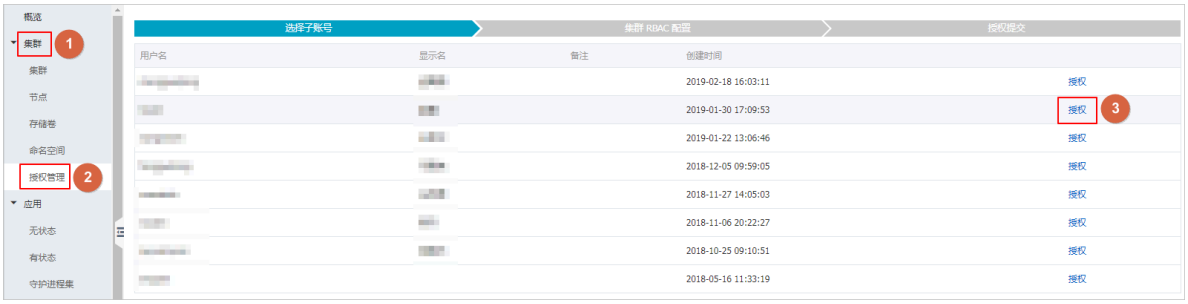
容器服务近期已完成升级集群授权管理安全策略：

- 禁止所有未授权的子账号访问集群资源，请及时对管理范围内的集群进行子账号应用权限设置及RAM授权操作。
- 子账号将只拥有授权域内集群的指定访问权限，在授权域外的原有兼容访问模式将被禁止，

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 > 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，进入集群RBAC配置页面。

> 说明：
> 如果您使用子账号授权，请先确保该子账号已完成集群RBAC配置#unique_19文档中的RAM授权，同时已被授予集群内RBAC管理员权限或cluster-admin角色。



4. 单击集群RBAC配置页面的加号，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，完成后单击下一步。

> 说明：

当前针对子账号的授权，支持在一个目标集群或命名空间上授予一个预置角色和多个自定义角色。

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 2-3: 角色权限说明

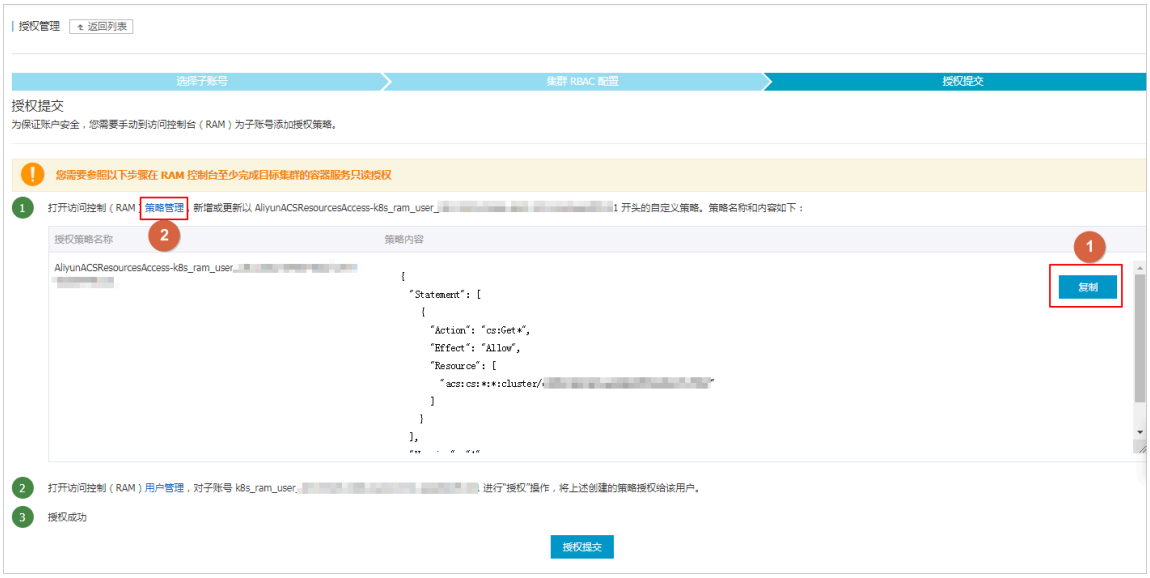
角色	集群内RBAC权限
管理员	对所有命名空间下所有资源的读写权限
运维人员	对所有命名空间下资源的读写权限，对集群节点，存储卷，命名空间，配额的只读权限
开发人员	对所有命名空间或所选命名空间下资源的读写权限
受限用户	对所有命名空间或所选命名空间下资源的只读权限
自定义	权限由您所选择的 ClusterRole 决定，请在确定所选 ClusterRole 对各类资源的操作权限后再进行授权，以免子账号获得不符合预期的权限。请参见自定义权限说明。

5. 在授权提交页签，如果出现授权成功，表示该子账号之前已被授予RAM权限，此时也已完成RBAC授权，操作结束。如果弹出错误提示对话框，表示该子账号未被授予RAM权限，请按

照页面提示，通过RAM控制台对子账号授予指定集群的只读权限。更多的权限请参见容器服务RAM Action。

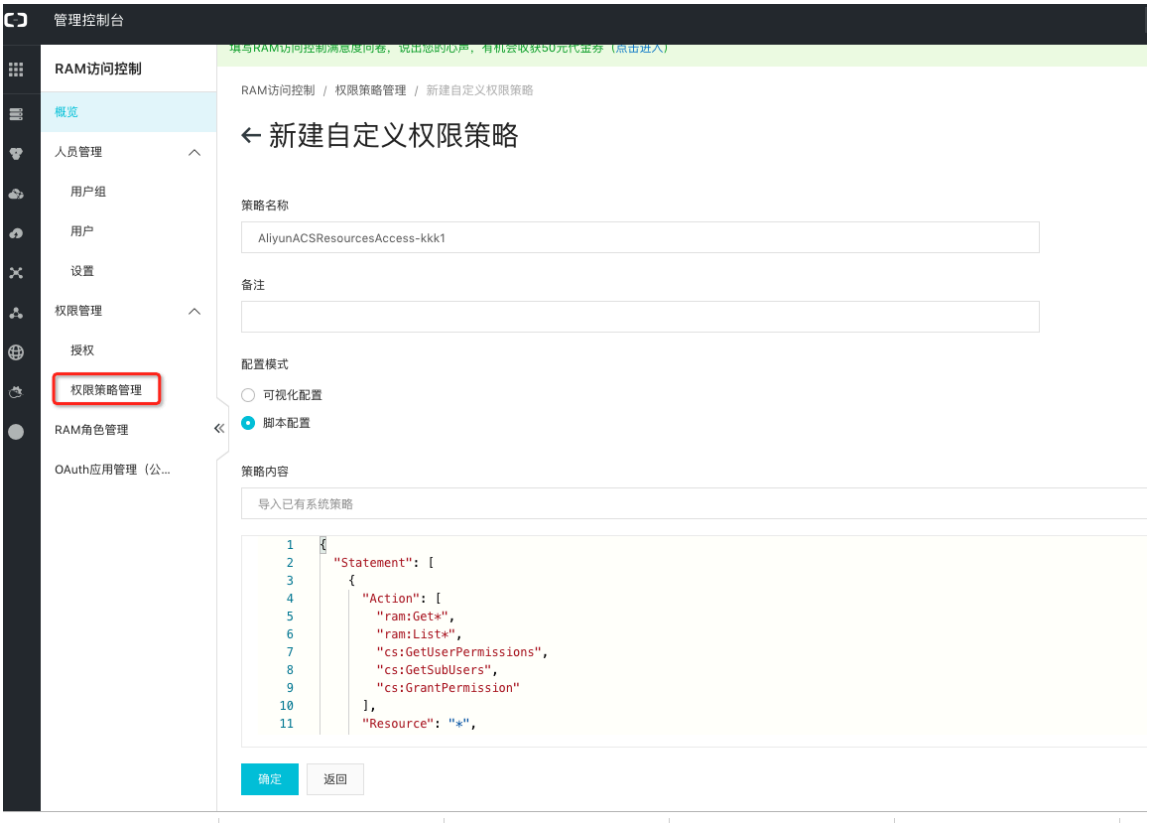
a. 请单击确定。

b. 在授权提交页面，单击复制后，再单击策略管理。

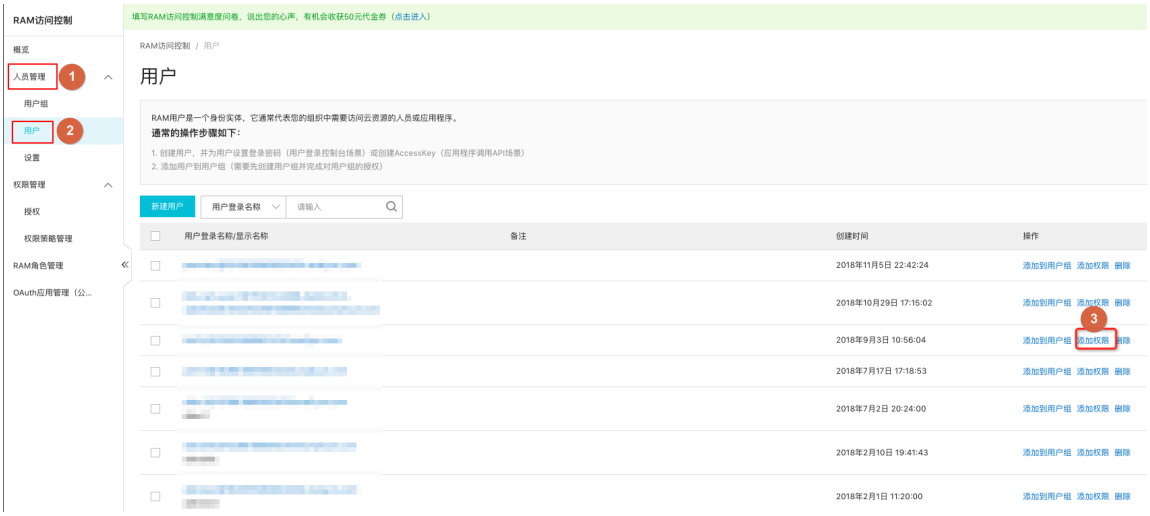


c. 跳转到RAM控制台页面，选择权限管理 > 权限策略管理，单击新建权限策略。

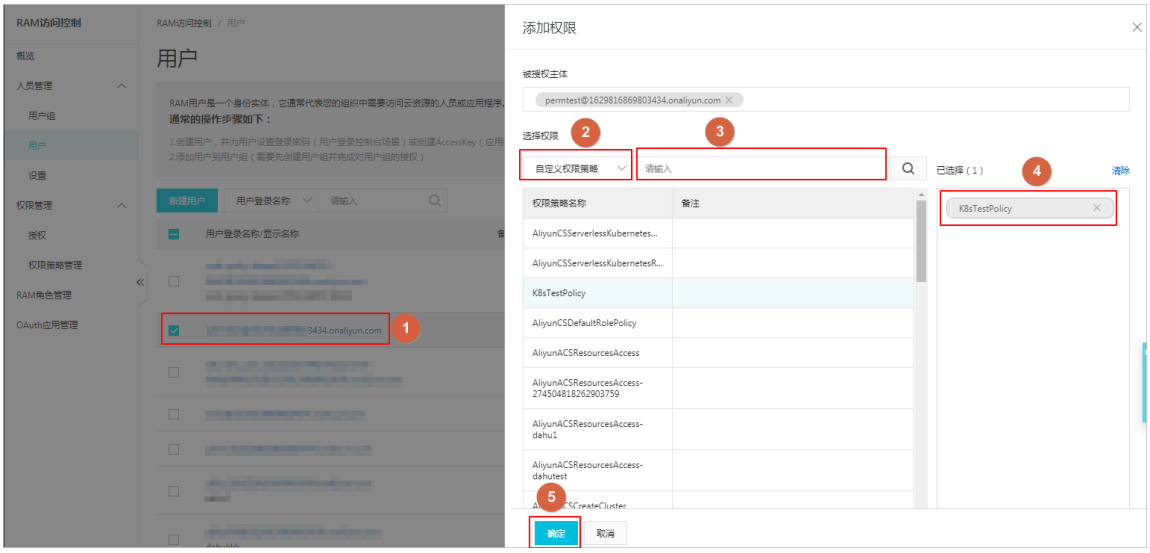
d. 进入新建自定义权限策略页面，自定义策略名称，配置模式选择脚本配置，使用Ctrl+V，将步骤6.b复制的内容粘贴到策略内容中，单击确定。详细的操作请参见#unique_8。



e. 选择人员管理 > 用户，在需要授权的用户的右侧，单击添加权限。



f. 在弹出的添加权限页面，权限选择自定义添加权限，搜索或者在下表中找到自定义的策略名称，单击移动到右侧已选择区域。单击确定，完成对子账号在指定集群的只读能力的授权。



g. 此时，返回容器服务控制台，在授权提交页面，单击授权提交，完成子账号RBAC的授权。

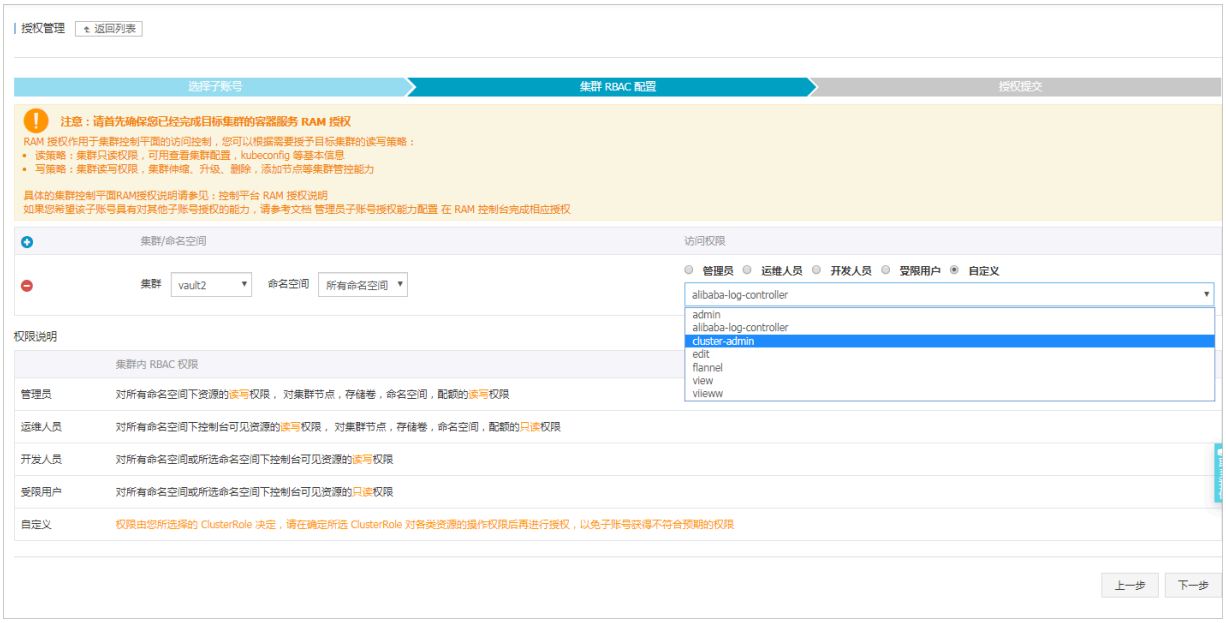
6. 配置完成后，您可以使用目标子账号登录容器服务控制台，并进行相关操作。

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。

```
# kubectl get clusterrole
```

```
# 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。

```
# kubectl get clusterrole cluster-admin -o yaml
```

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:
  - '*'
```

3 集群管理

3.1 查看集群概览

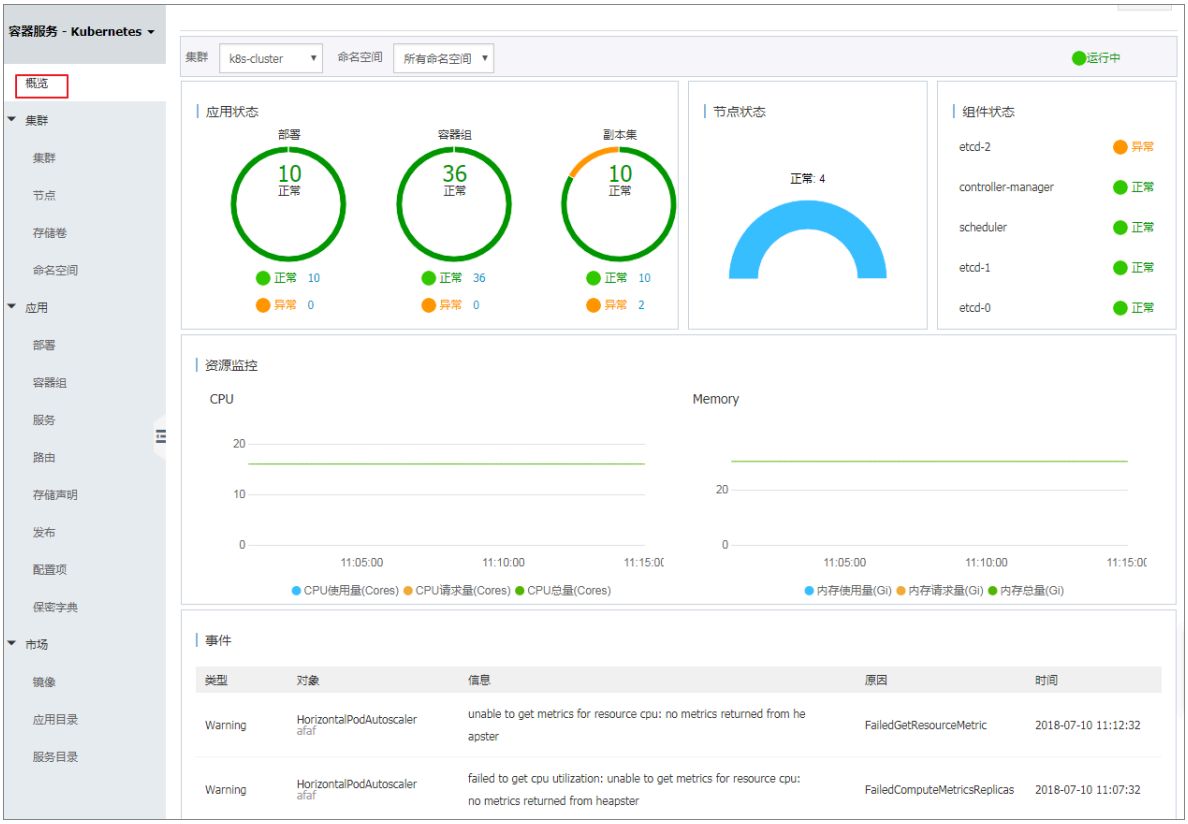
阿里云容器服务 Kubernetes 集群提供集群概览功能，提供应用状态、组件状态和资源监控等功能，方便您快速了解集群的健康状态信息。

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。



3.2 创建集群

3.2.1 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。

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。

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。

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。

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。
- 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。
- 给内网SLB挂载EIP，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）。

集群使用过程中，有如下限制：

- 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 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 > 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 集群配置页面。

4. （可选）选择集群所在的资源组。

5. 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（-）。

6. 选择集群所在的地域。

7. 设置集群的网络。Kubernetes 集群仅支持专有网络。

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。

8. 设置虚拟交换机。

您可以在已有 vswitch 列表中，根据可用区选择1~3个交换机 。如果没有您需要的交换机，可以通过单击创建虚拟交换机进行创建，请参见#unique_24。

9. 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。

10.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:

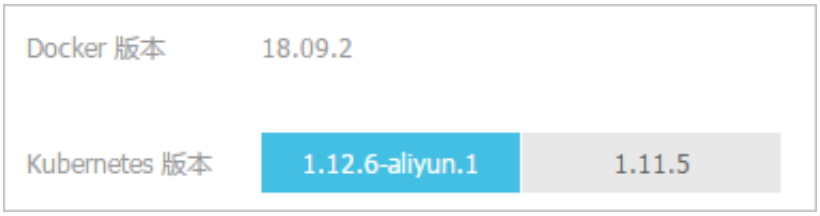
- 目前支持 CentOS、Windows 操作系统。
- 目前支持创建 3 个或者5个 Master 节点。
- 默认为Master节点挂载系统盘，支持SSD云盘和高效云盘。

11.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。

- 若您选择新增实例，则需要进行如下配置。

  - 实例规格：支持选择多个实例规格。参见#unique_25。
  - 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。
- 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。

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。



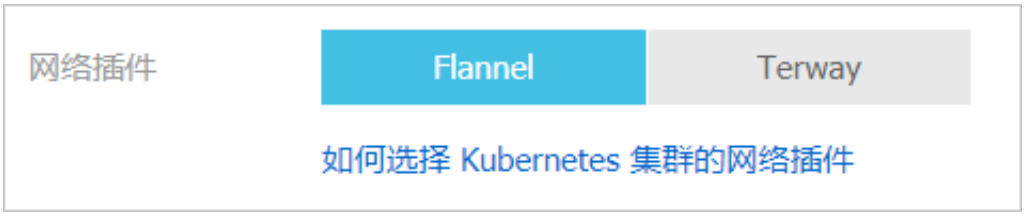
13.配置登录方式。

- 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#unique_26。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。

14.设置启用的网络插件，支持Flannel和Terway网络插件，具体可参考#unique_27。



- 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。

15.设置Pod网络 CIDR 和Service CIDR。

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#unique_28。

16.设置是否为专有网络配置 SNAT 网关。

说明:

若不勾选为专有网络配置 SNAT，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。

17.设置是否开放使用 EIP 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。

- 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。
- 如果选择不开放，则不会创建 EIP，用户只能在 VPC 内部用 kubeconfig 连接/操作集群。

18.设置是否开放公网 SSH 登录。

说明:

您需要开放使用 EIP 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#unique_29。

19.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。

20.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_30。

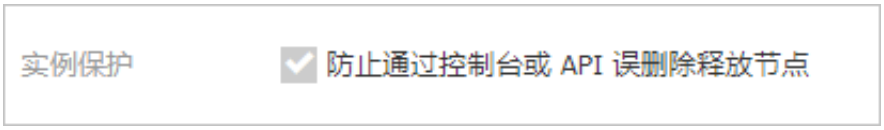
21.设置是否安装 Ingress 组件。

默认勾选安装 Ingress 组件，请参见Ingress 支持。

22.设置RDS白名单。

将节点 IP 添加到 RDS 实例的白名单。

23.设置是否启用实例保护。



说明:

为防止通过控制台或 API 误删除释放集群节点，默认启用实例保护。

24.为集群绑定标签。

输入键和对应的值，单击添加。

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。

25.是否启用高级选项。

a) 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。

b) 设置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。

c) 设置节点服务端口范围。

默认端口范围为30000~32767。

d) 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。

e) 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。

f）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_32。

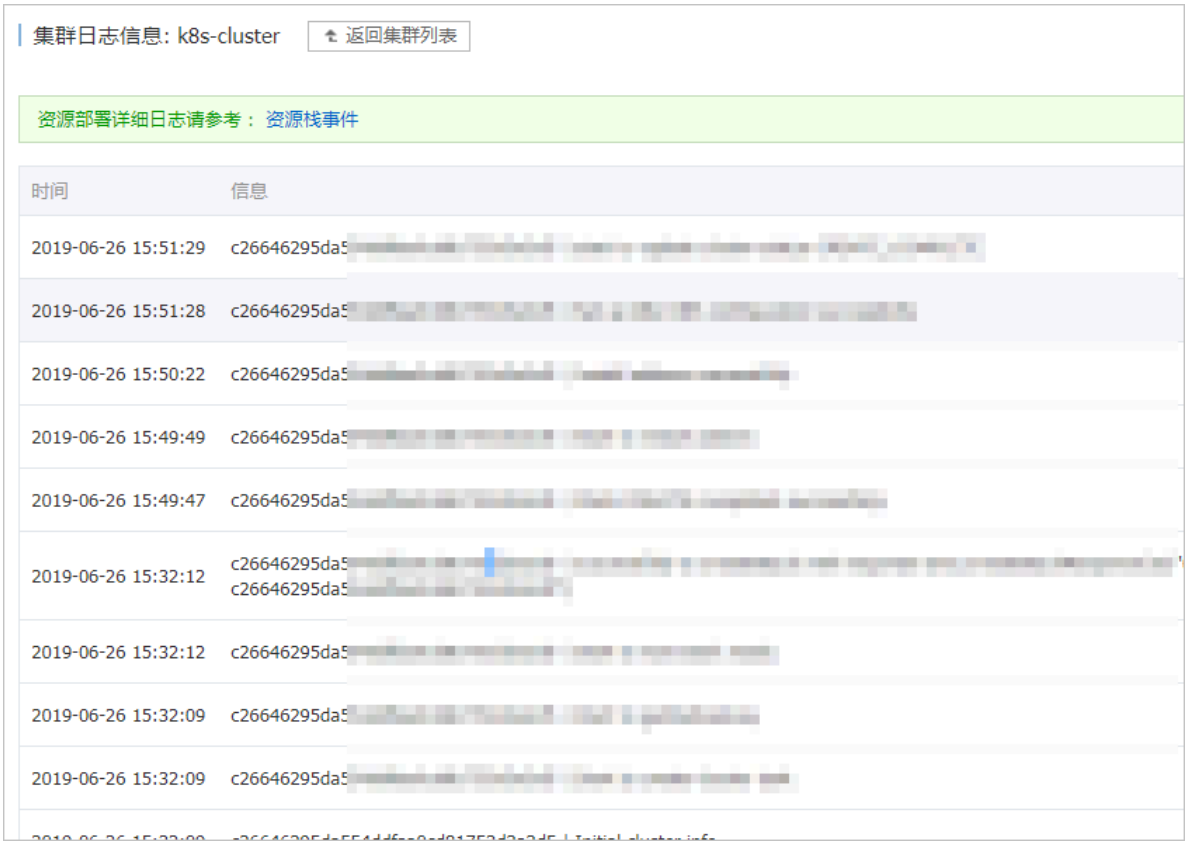
26.单击创建集群，在弹出的当前配置确认页面，单击创建，启动部署。

> **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。

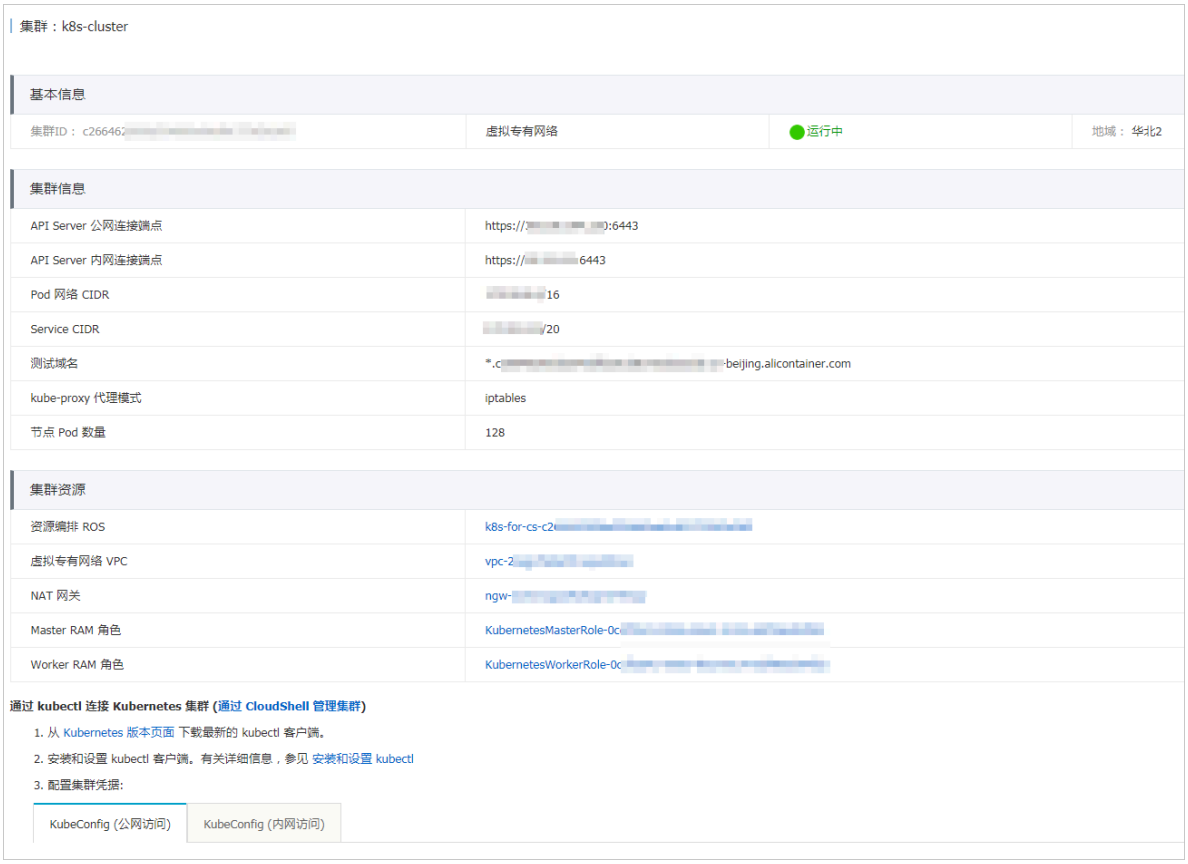
预期结果

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。
- Pod 网络 CIDR：每个 Pod 的 IP 地址段。创建成功后不能修改，且 Service 地址段也不能和 Pod 地址段重复。
- Service CIDR：Service 的 IP 地址段。创建成功后不能修改，且 Service 地址段也不能和 Pod 地址段重复。
- 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。
- 测试域名：为集群中的服务提供测试用的访问域名。测试域名后缀是`<cluster_id>.<region_id>.alicontainer.com`。
- kube-proxy 代理模式：Kubernetes service的服务发现和负载均衡需要通过服务代理进行配置，支持iptables和IPVS两种模式。
- 节点 Pod 数量：单个节点可运行 Pod 数量的上限，默认值为128。

例如，您可以#unique_33，执行`kubectl get node`查看集群的节点信息。

```
shell@Alicloud:~$ use-k8s-cluster

Type "kubectl" to manage your kubenetes cluster c50f6

shell@Alicloud:~$ kubectl get node
NAME                       STATUS   ROLES    AGE     VERSION
cn-hangzhou.192.168        Ready    master   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    master   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    master   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    master   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    master   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    <none>   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    <none>   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    <none>   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    <none>   6d23h   v1.12.6-aliyun.1
cn-hangzhou.192.168        Ready    <none>   6d23h   v1.12.6-aliyun.1
shell@Alicloud:~$
```

可以发现，一共有 10 个节点，包括 5 个 Master 节点和我们在参数设置步骤填写的 5 个 Worker 节点。

3.2.2 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、弹性伸缩（ESS）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 、 RAM 管理控制台 和 弹性伸缩控制台 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 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 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。

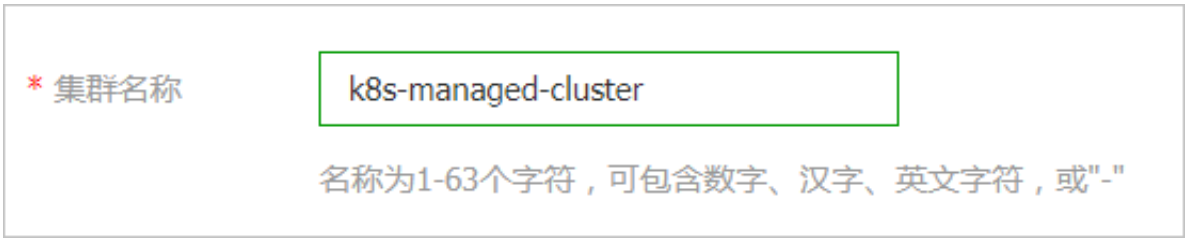
操作步骤

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 > 集群，单击页面右上角的创建 Kubernetes 集群。
3. 在弹出的选择集群模板页面，选择标准托管版集群页面，并单击创建，进入Kubernetes 托管版页面。

4. （可选） 选择集群所在的资源组。

5. 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（-）。



6. 选择集群所在的地域。

7. 设置集群的网络。Kubernetes 托管版集群仅支持专有网络。

   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。

8. 设置虚拟交换机。

   您可以在已有 vswitch 列表中，根据可用区选择1~3个交换机 。如果没有您需要的交换机，可以通过单击创建虚拟交换机进行创建，请参见#unique_24。

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。

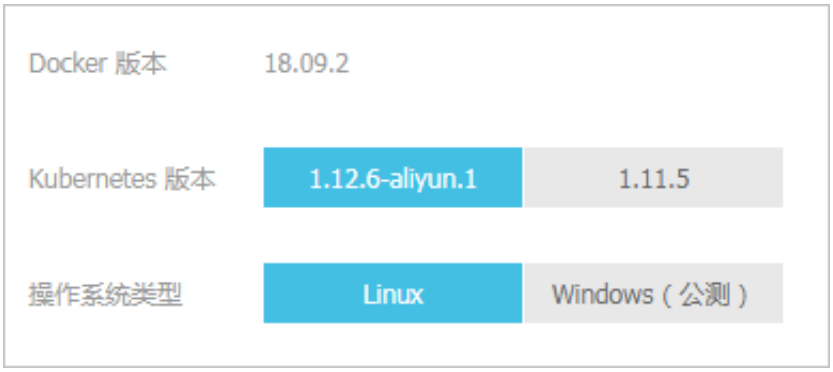
- 若您选择新增实例，则需要进行如下配置。

  - 节点类型：支持按量付费和包年包月两种节点类型。
  - 实例规格：参见#unique_25。
  - 已选规格：实例规格选中的规格。
  - 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。
- 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。

说明:

- 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。

10.显示当前支持的Docker版本、Kubernetes版本和操作类型系统，您可查看对应版本，并根据需要进行选择。本文以Linux为例进行介绍，关于Windows的操作请参见创建Windows Kubernetes 集群。



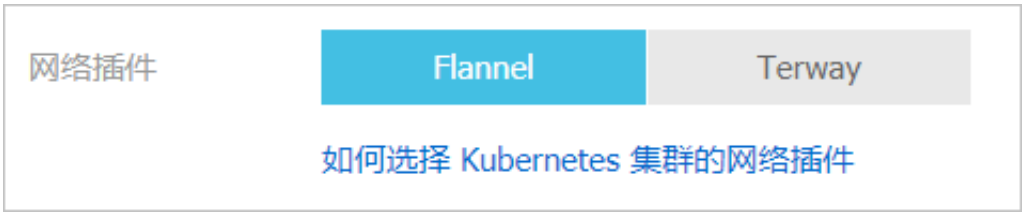
11.设置登录方式。

- 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#unique_26。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。

12.设置启用的网络插件，支持Flannel和Terway网络插件，具体可参考#unique_27。



- 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。

13.设置Pod 网络 CIDR 和Service CIDR。

说明:

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#unique_28。

14.设置是否为专有网络配置 SNAT 网关。

说明:

若不勾选为专有网络配置 SNAT，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。

15.设置是否启用公网访问。

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。

- 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。
- 如果选择不开放，则不会创建 EIP，用户只能在 VPC 内部用 kubeconfig 连接/操作集群。

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，详情参见使用日志服务进行Kubernetes日志采集。

18.设置是否安装 Ingress 组件。

默认勾选安装 Ingress 组件，请参见Ingress 支持。

19.设置RDS白名单。

将节点 IP 添加到 RDS 实例的白名单。

20.为集群绑定标签。

输入键和对应的值，单击添加。

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。

21.是否启用高级选项。

a. 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。

b. 设置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。

c. 是否启用工作流引擎。

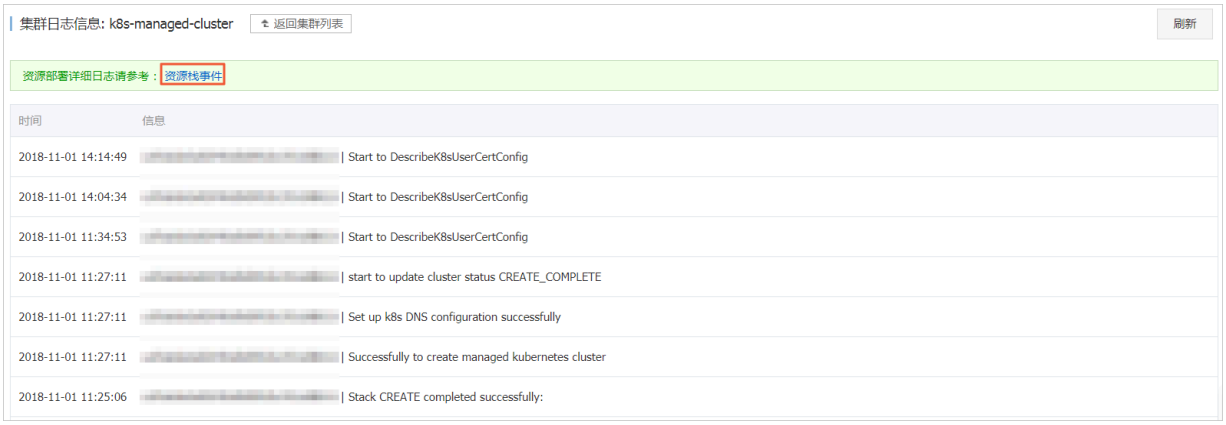
22.单击创建集群，在弹出的当前配置确认页面，单击创建，启动部署。

> **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。

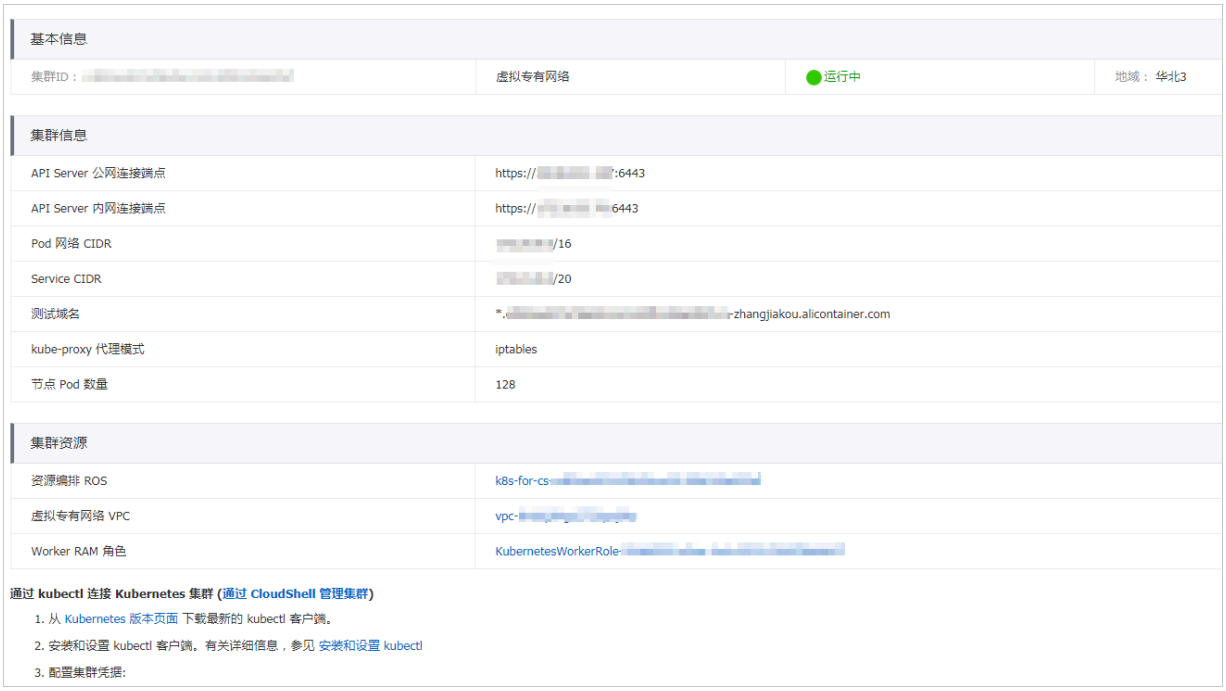
预期结果

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。

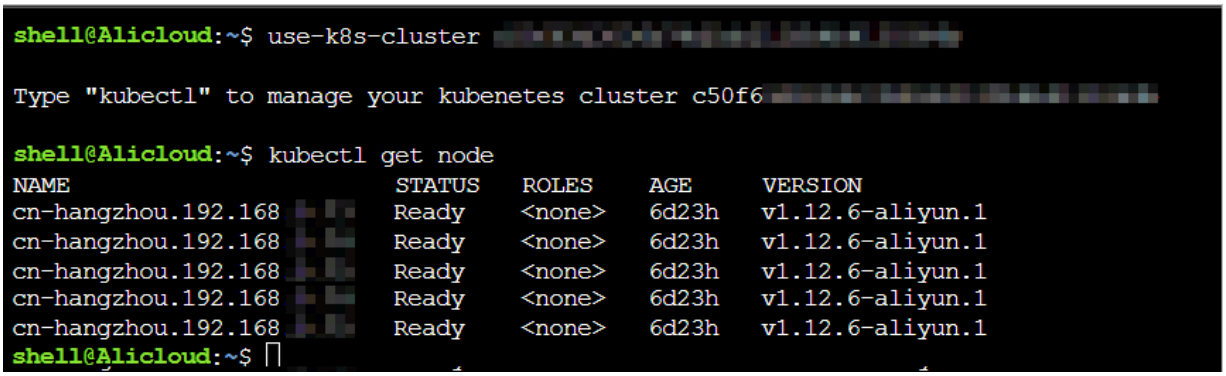


其中：

- 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。
- API Service 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。
- Pod网络CIDR：Kubernetes的Pod CIDR定义集群内Pod的网段范围。
- Service CIDR：Kubernetes的Service CIDR定义集群内暴露服务的网段范围。
- 测试域名：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`。
- kube-proxy 代理模式：Kubernetes service的服务发现和负载均衡需要通过服务代理进行配置，支持iptables和IPVS两种模式。

- 节点 Pod 数量：单个节点可运行 Pod 数量的上限，默认值为128。

您可以#unique_37，执行`kubectl get node`查看集群的节点信息。



3.2.3 创建Windows Kubernetes 集群

您可以通过容器服务控制台非常方便的创建Windows 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。

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。

> 说明:
> 容器服务Windows 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 Windows Kubernetes 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。
- Windows 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. 在弹出的选择集群模板页面，选择Windows 集群页面，并单击创建。
4. 进入创建 Kubernetes 集群页面，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。

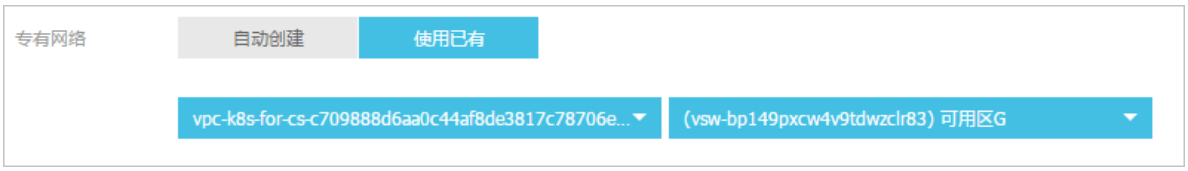
   说明:
   Kubernetes 集群仅支持专有网络。

   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:

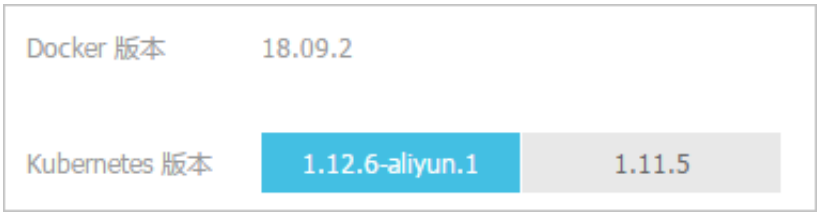
支持创建按量付费和包年包月两种节点类型。

8. 设置实例。

说明:

- 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. 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。



10.选择操作系统。

操作系统类型选择Windows。

11.设置登录方式。

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。

12.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#unique_28。

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.设置公网访问。

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。

- 如果选择开放，会创建一个EIP，同时把Master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，不会创建公网SLB，用户只能在VPC内部用kubeconfig连接/操作集群。

15.设置日志服务。

您可使用已有Project或新建一个Project。

您可以选择使用日志服务，从而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_30。

16.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。

17.是否启用高级选项。

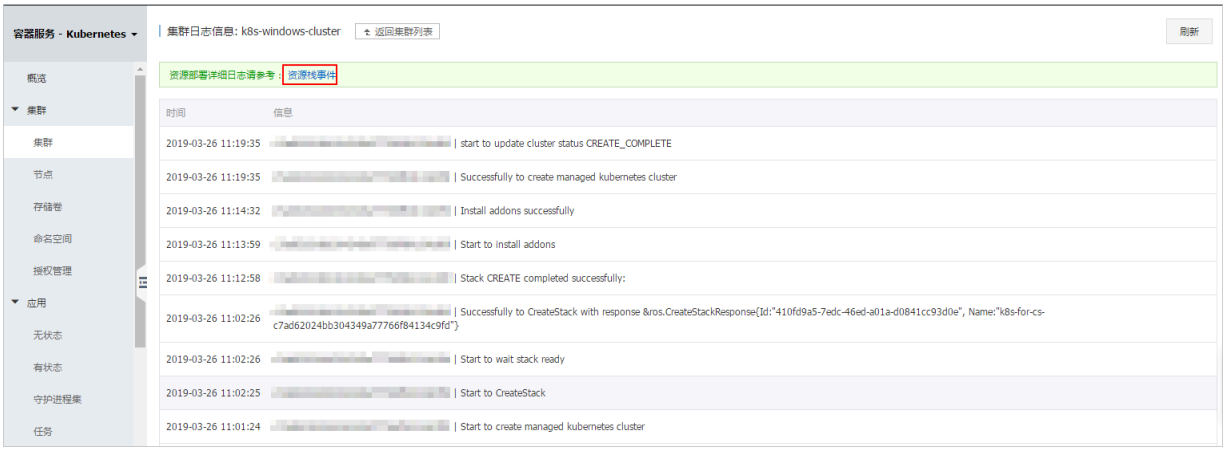
a. 设置节点 Pod 数量，是指单个节点可运行 Pod 数量的上限，建议保持默认值。

18.单击创建集群，在弹出的当前配置确认页面，单击创建，启动部署。

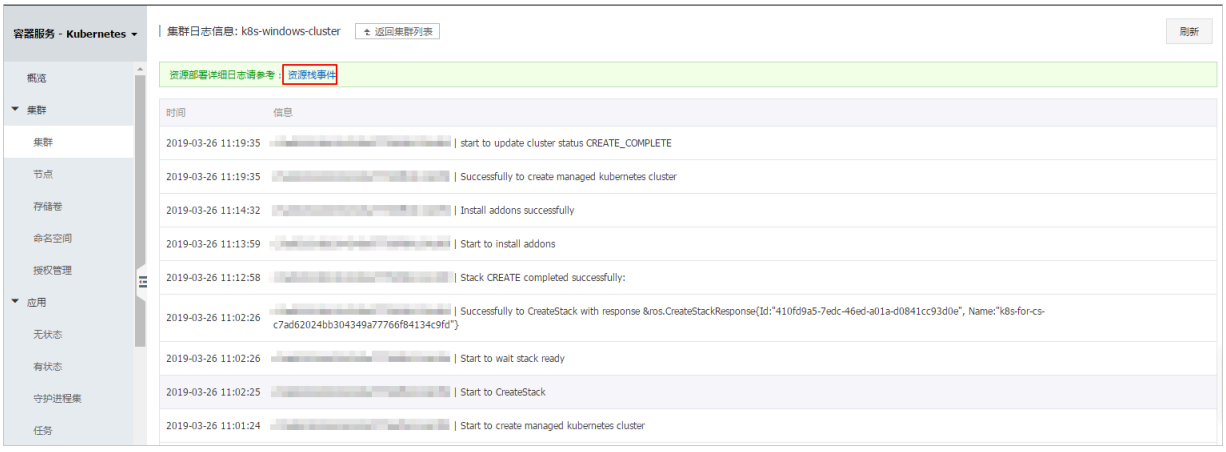
说明:
一个包含多节点的 Kubernetes 集群的创建时间一般约为 15 分钟。

预期结果

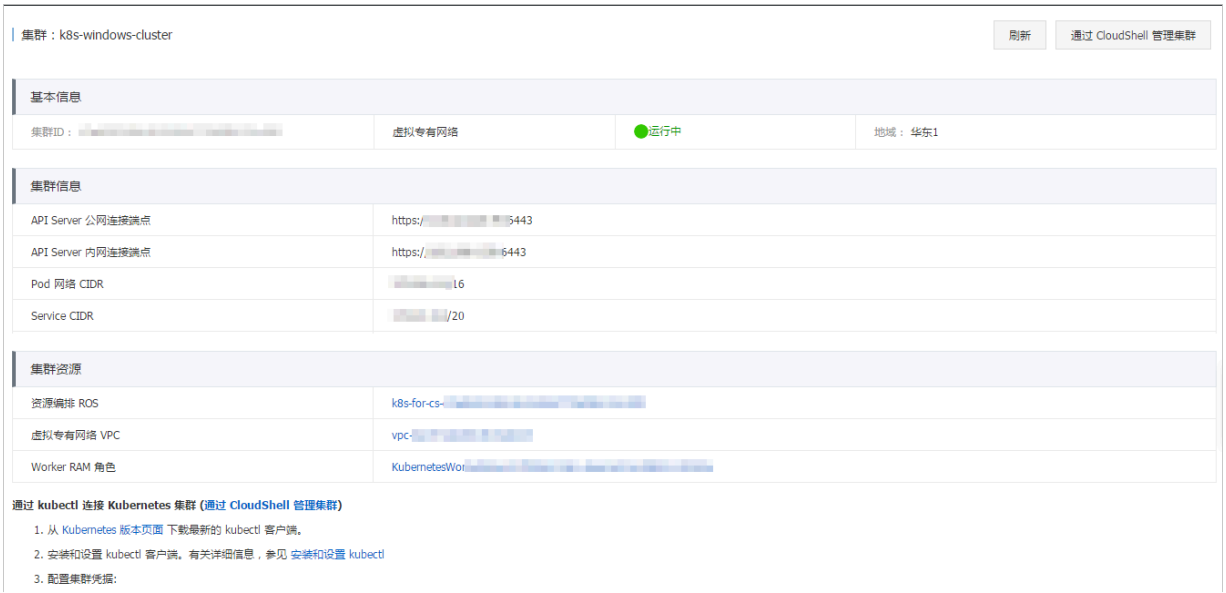
集群创建成功后，您可以在容器服务管理控制台的 Kubernetes 集群列表页面查看所创建的集群。



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。
- API Service 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。
- Pod网络CIDR：Kubernetes 的 Pod CIDR 定义集群内 Pod 的网段范围。
- Service CIDR：Kubernetes 的 Service CIDR 定义集群内暴露服务的网段范围。

您可以#unique_37，执行`kubectl get node`查看集群的节点信息。

```
shell@Alicloud:~$ kubectl get node
NAME                                STATUS   ROLES    AGE   VERSION
cn-hangzhou.[redacted]              Ready    <none>   26m   v1.12.6-aliyun.1
cn-hangzhou.[redacted]              Ready    <none>   26m   v1.12.6-aliyun.1
cn-hangzhou.[redacted]              Ready    <none>   26m   v1.12.6-aliyun.1
shell@Alicloud:~$
```

3.2.4 创建 Kubernetes 边缘托管版集群

您可以通过容器服务控制台非常方便的创建 Kubernetes 边缘托管版集群。

前提条件

您需要开通容器服务、资源编排（ROS）服务、弹性伸缩（ESS）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、 RAM 管理控制台 和 弹性伸缩控制台 开通相应的服务。

> **说明:**
> 容器服务 Kubernetes 边缘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 边缘托管版集群前，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 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。

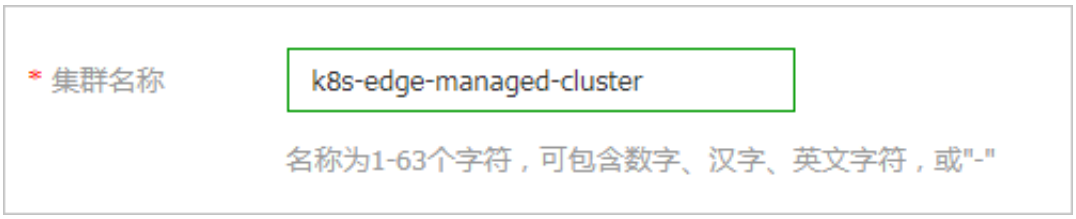
操作步骤

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 > 集群，单击页面右上角的创建 Kubernetes 集群。
3. 在弹出的选择集群模板页面，选择边缘托管集群并单击创建，进入Kubernetes 边缘托管版页面。
4. （可选） 选择集群所在的资源组。

5. 填写集群的名称。

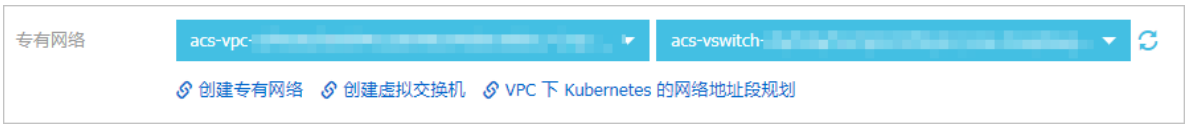
   集群名称应包含 1~63 个字符，可包含数字、汉字、英文字符或连字符（-）。



6. 选择集群所在的地域和可用区。

7. 设置集群的网络。Kubernetes 边缘托管版集群仅支持专有网络。

   您可以在已有 VPC 列表中选择所需的 VPC 和交换机。



   - 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。

8. 显示当前支持的Kubernetes版本。

9. 设置Pod 网络 CIDR 和Service CIDR。

   > **说明:**
   > 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，也不能和边缘节点所在网段重复，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#unique_28。

10.设置是否为专有网络配置 SNAT 网关。

说明:

若不勾选为专有网络配置 SNAT，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。

11.设置是否启用公网访问。

说明:

通常边缘节点需要通过公网和云端API server交互，因此若不勾选使用 EIP 暴露API Server，边缘节点将无法连接到云端集群，所创建集群也将无法在边缘场景下使用。

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。

- 如果选择开放，会创建一个EIP，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。
- 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。

12.设置是否启用云监控插件。

勾选在ECS节点上安装云监控插件，平台会自动帮助您购买ECS用于部署监控管控组件并会在集群中自动配置云监控插件。

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

勾选使用日志服务，平台会自动帮助您购买ECS用于部署日志管控组件并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.实例配置。

说明:

当您勾选了云监控插件和日志服务，此处的ECS配置才会呈现。

- 实例规格：支持选择多个实例规格。参见#unique_25。
- 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:

- 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。

15.设置登录方式。

说明:

当您勾选云监控插件或日志服务时，需要给ECS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#unique_26。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。

16.设置RDS白名单。

将节点 IP 添加到 RDS 实例的白名单。

17.为集群绑定标签。

输入键和对应的值，单击添加。

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。

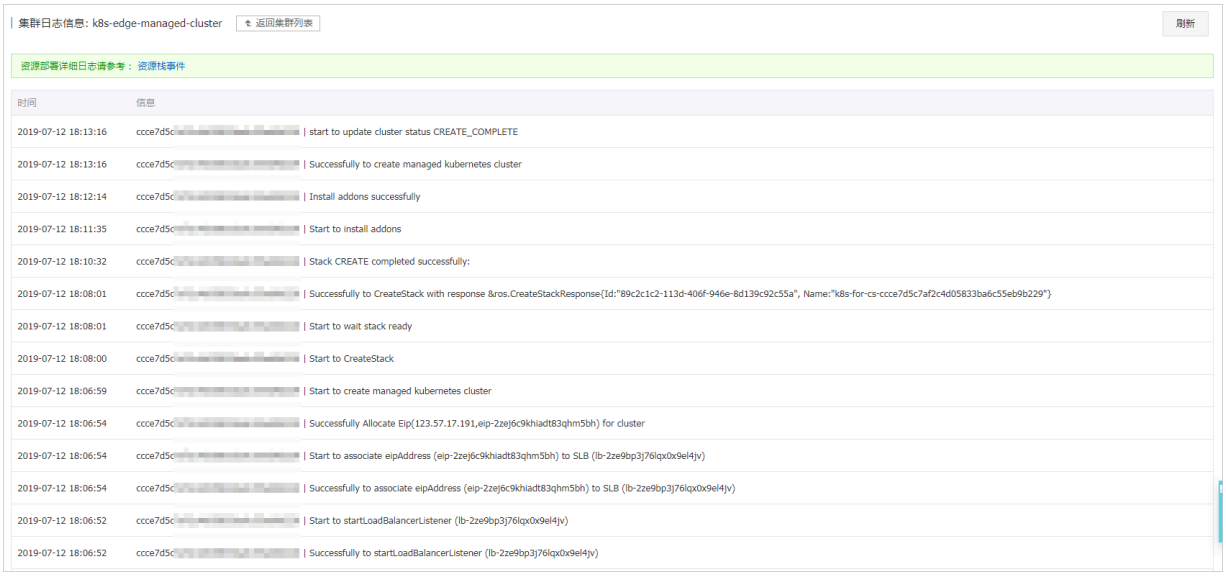
18.单击创建集群，在弹出的当前配置确认页面，单击创建，启动部署。

> **说明:**
> 一个 Kubernetes 边缘托管版集群的创建时间一般约为 15 分钟。

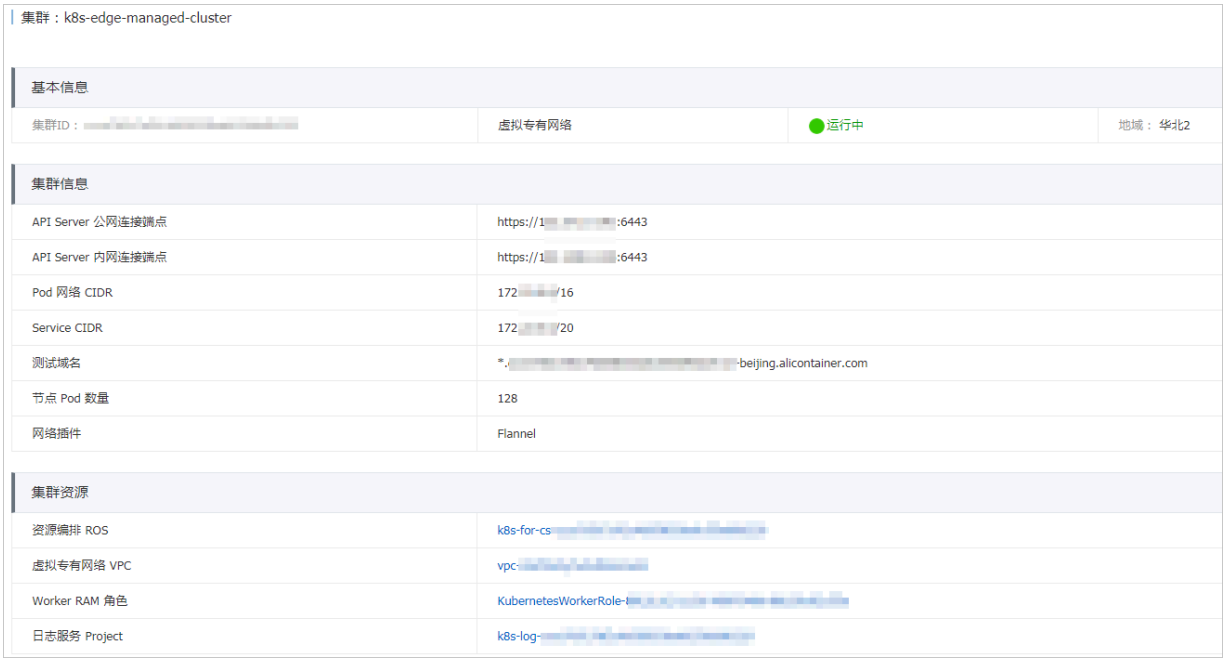
预期结果

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。
- API Service 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。
- Pod网络CIDR：Kubernetes的Pod CIDR定义集群内Pod的网段范围。
- Service CIDR：Kubernetes的Service CIDR定义集群内暴露服务的网段范围。
- 测试域名：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`。
- 节点 Pod 数量：单个节点可运行 Pod 数量的上限，默认值为128。
- 网络插件：支持Flannel和Terway网络插件。

您可以#unique_37，执行`kubectl get node`查看集群的节点信息。

```
Type "kubectl" to manage your kubenetes cluster

shell@Alicloud:~$ kubectl get node
NAME                                STATUS   ROLES    AGE   VERSION
cn-beijing.i-2zehvxttbua2aw800uin   Ready    <none>   11m   v1.12.6-aliyun.1
shell@Alicloud:~$
```

3.3 管理与访问集群

3.3.1 通过 kubectl 连接 Kubernetes 集群

本文主要为您介绍如何通过 kubectl 连接 Kubernetes 集群。

背景信息

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。

   - 如果您使用公网访问，请选择KubeConfig（公网访问）页签，并单击复制，将内容复制到本地计算机的 `$HOME/.kube/config`（`kubectl` 预期凭据所在的位置）。
   - 如果您使用内网访问，请选择KubeConfig（内网访问）页签，并单击复制，将内容复制到本地计算机的 `$HOME/.kube/config`（`kubectl` 预期凭据所在的位置）。

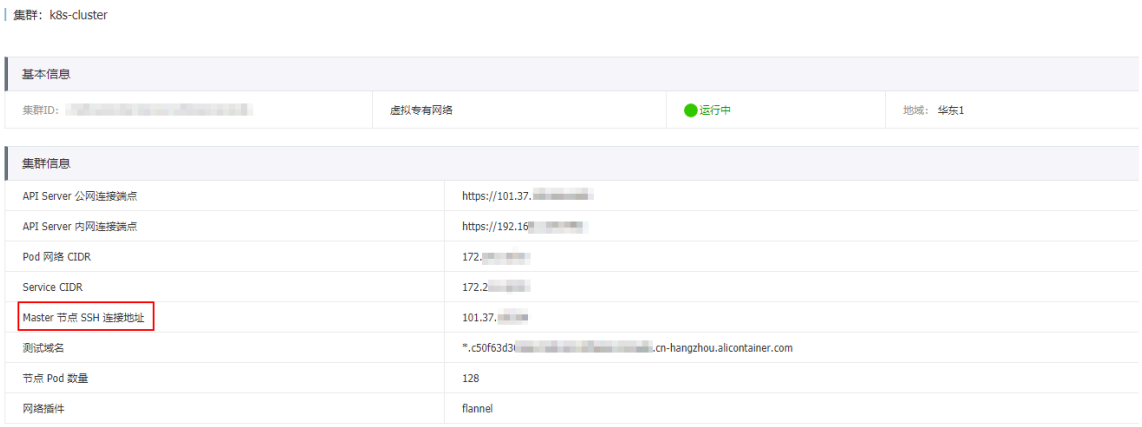
   配置完成后，您可以使用 kubectl 从计算机访问 Kubernetes 集群。

   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。



说明:

本操作仅适用专有版Kubernetes集群。

3.3.2 在CloudShell上通过kubectl管理Kubernetes集群

本文为您介绍如何在容器服务Kubernetes控制台上利用CloudShell通过kubectl管理Kubernetes集群。

前提条件

您已成功创建一个Kubernetes集群，参见#unique_42。

背景信息

若您希望通过kubectl工具管理容器服务Kubernetes集群，您可下载kubectl到客户端，具体安装方法，请参见#unique_43。您也可以直接在容器服务Kubernetes控制台上打开CloudShell，在CloudShell中通过kubectl管理集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 集群，进入集群列表页面。
3. 选择目标集群，单击操作列更多 > 通过CloudShell管理集群。

> 说明:
>
> 打开后，请执行以下操作：
>
> · 在授权页面单击确认，可获取一个临时的会话访问密钥，密钥有效期为1小时。
> · 在挂载存储空间页面，可根据您的实际情况单击创建并绑定或暂不创建。

4. 在CloudShell中可通过kubectl工具管理容器服务Kubernetes集群。

> 说明:
>
> 在打开集群关联的CloudShell时，系统会自动加载集群的`kubeconfig`文件。您可以通过kubectl直接管理您的集群。

3.3.3 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。

> 说明:

由于创建集群所使用的 ROS 编排模板的版本不同，不同时间点创建的集群开启 SSH 访问的操作会略有不同，请按照您的集群的创建时间选择不同的操作方法。

2017 年 12 月之前创建的集群

1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并查看集群 ID。
4. 登录 ROS 管理控制台。
5. 选择您的集群所在的地域，在搜索框中输入k8s-for-cs-clisterid搜索您的集群对应的资源栈，并单击右侧的管理。

6. 单击左侧导航栏中的资源，在资源列表中找到名为 `k8s_master_slb_internet` 的资源（该资源为您的集群的公网负载均衡实例）并单击资源 ID，页面跳转至您的公网负载均衡实例的详细信息页面。
7. 单击左侧导航栏中的实例 > 实例管理，单击添加监听。

8. 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。

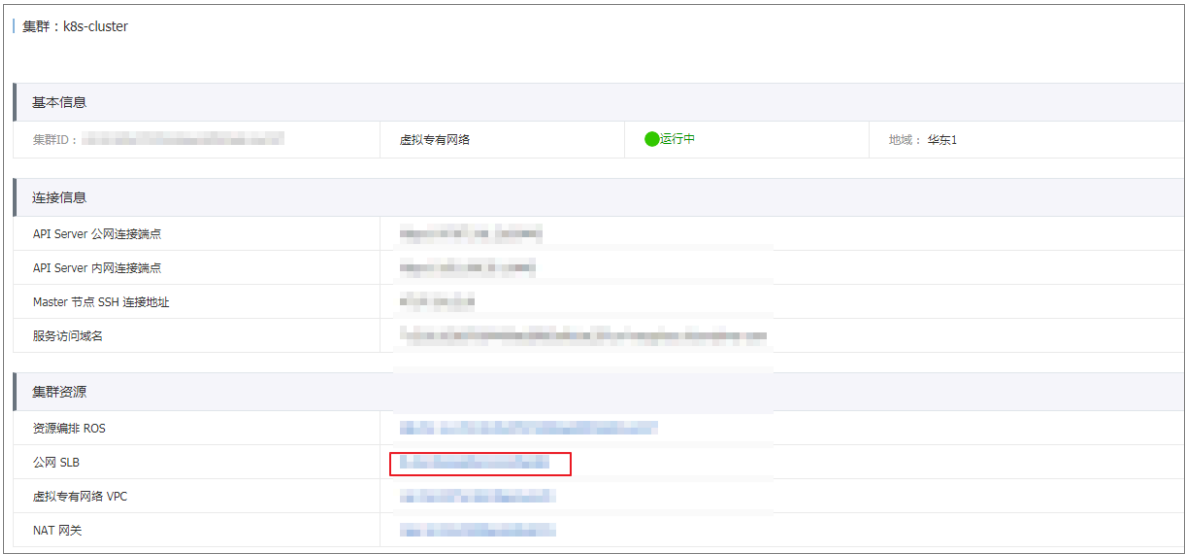
   d. 服务器组ID 选择 sshVirtualGroup。

   e. 单击 下一步 并单击确认，创建监听。

9. 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。

2017 年 12 月之后创建的集群

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。

3.3.4 使用ServiceAccount Token访问Kubernetes集群

本文介绍如何使用ServiceAccount Token访问Kubernetes集群，该功能适用于所有的Kubernetes集群，本例中为托管版集群。

背景信息

- 您已经成功创建一个 Kubernetes 托管版集群，参见#unique_46。
- 您可以通过Kubectl连接到Kubernetes 托管版集群，参见#unique_37。

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   [redacted]      [redacted]      443:[redacted]/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
[redacted]                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
```

3.3.5 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#unique_48，最后单击创建。

   a. 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。

   b. 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#unique_26。

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#unique_49。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`。

3.4 升级集群

3.4.1 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#unique_52。
- 集群升级 Kubernetes 过程中，1.8.1或1.8.4版本升级到1.9.3版本时，集群的所有Pod会被重启，对应用有影响。其他版本升级时，集群上的应用不会中断。如果应用强依赖于API Server可能会有短暂影响。
- 由于升级过程中网络重置，OSS存储卷会重新挂载，使用OSS存储卷的Pod在升级后需要重建。

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#unique_29和#unique_37。

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。

> **说明:**
> 所有节点只能Ready状态，不能包含其他状态。

```
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。

3.4.2 升级集群的GPU节点的NVIDIA驱动

本文介绍在Kubernetes集群的GPU节点有业务运行，及集群刚刚创建GPU节点没有业务运行两种场景下，如何升级集群GPU节点的NVIDIA驱动。

前提条件

- 您已经成功创建一个Kubernetes GPU集群，参见#unique_54。
- 您可以通过kubectl连接到Kubernetes GPU集群，参见#unique_37。

升级已有业务运行的GPU节点的驱动

1. 执行以下命令，把待升级驱动的GPU节点设置为不可调度。

```
kubectl cordon node-name
```

说明:

- 仅支持升级Worker节点的NVIDIA驱动。

- *node-name*格式为*your-region-name.node-id*。
  - *your-region-name*为您集群所在的地域名称。
  - *node-id*为节点所在的ECS实例ID。

  可执行以下命令查看*node-name*。

  ```
  kubectl get node
  ```

```
[root@gpu-test ~]# kubectl cordon cn-hangzhou.i-
node/cn-hangzhou.i-                                already cordoned
```

2. 执行以下命令，把步骤1节点上的Pod转移到其他节点。

```
kubectl drain node-name --grace-period=120 --ignore-daemonsets=true
```

```
[root@gpu-test ~]# kubectl drain cn-hangzhou.i-                --grace-period=120 --ignore-daemonsets=true
node/cn-hangzhou.i-                cordoned
WARNING: Ignoring DaemonSet-managed pods: flexvolume-, kube-flannel-ds-, kube-proxy-worker-, logtail-ds-
pod/domain-nginx-                evicted
pod/old-nginx-                evicted
pod/new-nginx-                evicted
pod/old-nginx-                evicted
```

3. 执行以下命令，登录步骤1的节点。

```
ssh root@xxx.xxx.x.xx
```

4. 在待升级驱动的节点上，执行以下命令，查看驱动版本。

```
nvidia-smi
```

```
[root@                ~]# nvidia-smi
Fri Jan 18 16:44:52 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                Driver Version: 384.111                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P4            On   | 00000000:00:08.0 Off |                    0 |
| N/A   24C    P8     6W /  75W |      0MiB /  7606MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

5. 执行以下命令，删除已有的驱动。

说明:

- 如果您驱动的版本是384.111，请按照以下步骤执行。

- 如果您驱动的版本不是384.111，请到NVIDIA官网下载对应版本的驱动包后，按照以下步骤执行卸载。

```
cd /tmp
```

```
curl -O https://cn.download.nvidia.cn/tesla/384.111/NVIDIA-Linux-x86_64-384.111.run
```

```
chmod u+x NVIDIA-Linux-x86_64-384.111.run
```

```
./NVIDIA-Linux-x86_64-384.111.run --uninstall -a -s -q
```

6. 执行以下命令，重启待升级驱动的节点。

```
reboot
```

7. 请到官网下载需要升级到的驱动，本文以410.79版本为例。
8. 执行以下命令，在步骤7下载驱动的路径安装NVIDIA驱动。

```
sh ./NVIDIA-Linux-x86_64-410.79.run -a -s -q
```

9. 执行以下命令，在NVIDIA驱动中增加以下配置。

```
nvidia-smi -pm 1 || true
```

```
nvidia-smi -acp 0 || true
```

10.执行以下命令，更新device-plugin。

```
mv /etc/kubernetes/manifests/nvidia-device-plugin.yml /
```

```
mv /nvidia-device-plugin.yml /etc/kubernetes/manifests/
```

11.在Master节点的任意路径下执行以下命令，将此节点设置为可调度。

```
kubectl uncordon node-name
```

执行结果

在Master节点执行以下命令，在升级驱动的GPU节点上验证升级后的驱动版本，可看到该GPU节点上NVIDIA驱动的版本为410.79，说明该节点的驱动已升级成功。

说明:

请替换您实际的*node-name*。

```
kubectl exec -n kube-system -t nvidia-device-plugin-node-name nvidia-smi
```

```
[root@gpu-test ~]#  kubectl exec -n kube-system -t nvidia-device-plugin-cn-[redacted] nvidia-smi
Mon Jan 21 03:14:48 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P4            On   | 00000000:00:08.0 Off |                    0 |
| N/A   21C    P8     6W /  75W |      0MiB /  7611MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

集群刚刚创建，升级没有业务运行的GPU节点的驱动

1. **执行以下命令，登录待升级驱动的GPU节点。**

   ```
   ssh root@xxx.xxx.x.xx
   ```

2. **在待升级驱动的节点上，执行以下命令，查看驱动版本。**

   ```
   nvidia-smi
   ```

   ```
   [root@[redacted] ~]# nvidia-smi
   Fri Jan 18 16:44:52 2019
   +-----------------------------------------------------------------------------+
   | NVIDIA-SMI 384.111                Driver Version: 384.111                   |
   |-------------------------------+----------------------+----------------------+
   | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
   | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
   |===============================+======================+======================|
   |   0  Tesla P4            On   | 00000000:00:08.0 Off |                    0 |
   | N/A   24C    P8     6W /  75W |      0MiB /  7606MiB |      0%      Default |
   +-------------------------------+----------------------+----------------------+

   +-----------------------------------------------------------------------------+
   | Processes:                                                       GPU Memory |
   |  GPU       PID   Type   Process name                             Usage      |
   |=============================================================================|
   |  No running processes found                                                 |
   +-----------------------------------------------------------------------------+
   ```

3. **执行以下命令，删除已有的驱动。**

   说明:

   - **如果您驱动的版本是384.111，请按照以下步骤执行。**

- 如果您驱动的版本不是384.111，请到NVIDIA官网下载对应版本的驱动包后，按照以下步骤执行卸载。

```
cd /tmp
```

```
curl -O https://cn.download.nvidia.cn/tesla/384.111/NVIDIA-Linux-x86_64-384.111.run
```

```
chmod u+x NVIDIA-Linux-x86_64-384.111.run
```

```
./NVIDIA-Linux-x86_64-384.111.run --uninstall -a -s -q
```

4. 执行以下命令，重启待升级驱动的节点。

```
reboot
```

5. 请到官网下载需要升级到的驱动，本文以410.79版本为例。
6. 执行以下命令，在步骤5下载驱动的路径安装NVIDIA驱动。

```
sh ./NVIDIA-Linux-x86_64-410.79.run -a -s -q
```

7. 执行以下命令，在NVIDIA驱动中增加以下配置。

```
nvidia-smi -pm 1 || true
```

```
nvidia-smi -acp 0 || true
```

执行结果

在Master节点执行以下命令，在升级驱动的GPU节点上验证升级后的驱动版本，可看到该GPU节点上NVIDIA驱动的版本为410.79，说明该节点的驱动已升级成功。

说明:

请替换您实际的*node-name*。

```
kubectl exec -n kube-system -t nvidia-device-plugin-node-name nvidia-smi
```

```
[root@gpu-test ~]#  kubectl exec -n kube-system -t nvidia-device-plugin-cn-[redacted] nvidia-smi
Mon Jan 21 03:14:48 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P4            On   | 00000000:00:08.0 Off |                    0 |
| N/A   21C    P8     6W /  75W |      0MiB /  7611MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

3.4.3 升级安装Kubernetes集群的metrics-server组件

本文档介绍如何在不升级Kubernetes集群的情况下升级安装metrics-server组件。

前提条件

- **您已成功创建一个Kubernetes集群。有关如何创建Kubernetes集群，参见#unique_42。**
- **确保Kubernetes集群的初始版本是1.12.6及以前的版本。**

Kubernetes集群组件的升级过程分为以下三个部分：切换数据采集组件、切换监控数据链路、变更组件兼容适配。

切换数据采集组件

创建并拷贝内容到*metrics-server.yaml*文件中，并执行`kubectl apply -f metrics-server.yaml`命令，将数据采集组件从Heapster切换到metrics-server。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin
  namespace: kube-system
---
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/
kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out
this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: metrics-server
  name: heapster
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    k8s-app: metrics-server
---
apiVersion: v1
kind: Service
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    kubernetes.io/name: metrics-server
spec:
  selector:
    k8s-app: metrics-server
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
---
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1beta1.metrics.k8s.io
spec:
  service:
    name: metrics-server
    namespace: kube-system
  group: metrics.k8s.io
  version: v1beta1
  insecureSkipTLSVerify: true
  groupPriorityMinimum: 100
  versionPriority: 100
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    k8s-app: metrics-server
spec:
  selector:
    matchLabels:
      k8s-app: metrics-server
  template:
    metadata:
      name: metrics-server
      labels:
```

```
        k8s-app: metrics-server
    spec:
      serviceAccountName: admin
      containers:
      - name: metrics-server
        image: registry.##REGION##.aliyuncs.com/acs/metrics-server:
v0.2.1-9dd9511-aliyun
        imagePullPolicy: Always
        command:
        - /metrics-server
        - '--source=kubernetes:https://kubernetes.default'
        - '--sink=socket:tcp://monitor.csk.##REGION##.aliyuncs.com:
8093?clusterId=
&public=true'
```

说明:

您需要将##REGION##与##CLUSTER_ID##替换为所选集群的地域名称（例如，华东1：cn-hangzhou）与集群ID。

切换监控数据链路

1. **单击左侧导航栏中的集群 > 节点，进入节点列表页面。**
2. **选择目标集群。**
3. **找到集群的三个master节点。单击master节点的实例ID，进入实例详情页面（本文档以master-01为例）。**
4. **单击远程连接。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。登录成功后，输入以下命令：**

```
sed -i 's/--horizontal-pod-autoscaler-use-rest-clients=false/--
horizontal-pod-autoscaler-use-rest-clients=true/' /etc/kubernetes/
manifests/kube-controller-manager.yaml
```

5. **重复步骤3-4，在master-02和master-03节点上执行该命令。**

执行完毕后，kube-controller-manager组件会被kubelet自动拉起更新。

组件兼容适配变更

1. **单击左侧导航栏中的路由与负载均衡 > 服务，进入服务页面。**
2. **选择目标集群和命名空间kube-system。单击服务heapster右侧的查看YAML。**
3. **在弹出的对话框中，修改selector中k8s-app的值为metrics-server。单击更新，完成修改。**

4. 单击左侧导航栏中的应用 > 无状态，进入无状态页面。
5. 选择目标集群和命名空间kube-system。
6. 选择Heapster相关组件（heapster和monitoring-influxdb），单击更多 > 删除，在弹出的对话框中，单击确定，完成链路切换。

> 说明:
> 删除monitoring-influxdb 组件时，在弹出的删除monitoring-influxdb提示框中，勾选移除关联的服务（Server）monitoring-influxdb，单击确定。

7. 验证链路切换状态。

   等待3分钟左右，数据链路会完成初始化。

   单击左侧导航栏中的应用 > 容器组。在容器组页面。如果CPU（核）与内存（字节）显示正常，则表示链路切换成功。

> 说明:
> 所有组件的CPU和内存值均为0则表示异常。

3.4.4 升级系统组件

本文为您介绍如何升级系统组件。

前提条件

您已成功创建一个Kubernetes集群，参见#unique_48。

背景信息

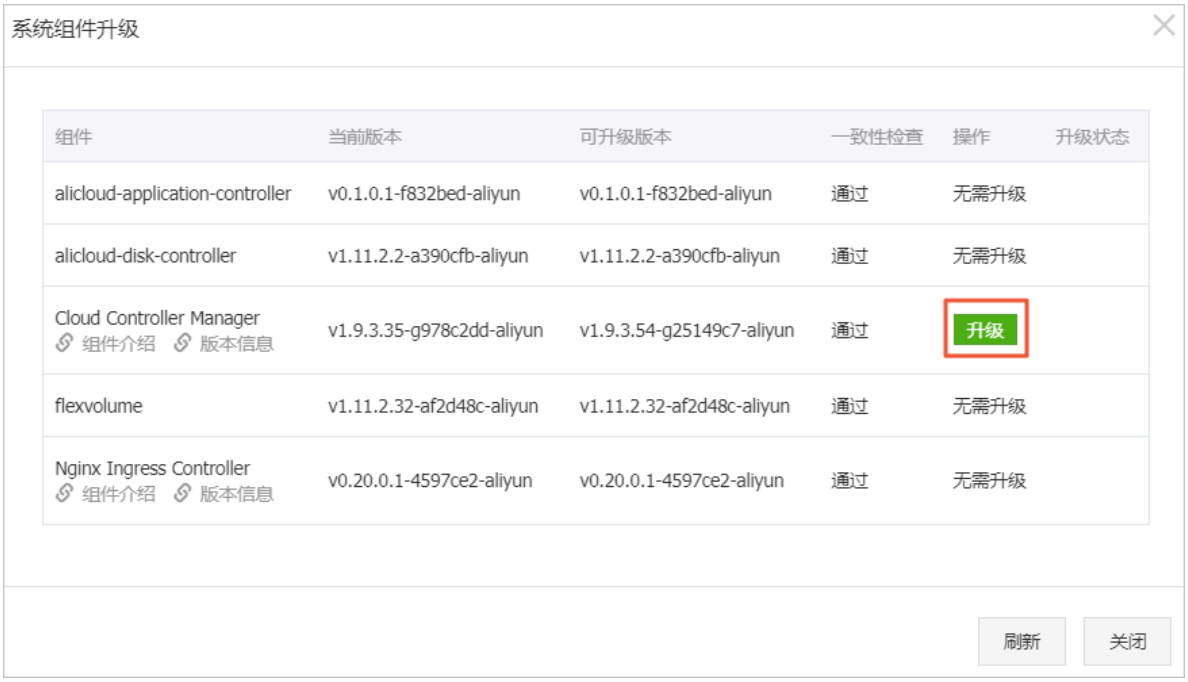
很多时候集群已经是最新版本，但某些系统组件需要进行更小粒度的版本升级。本文为您介绍如何升级系统组件。

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。

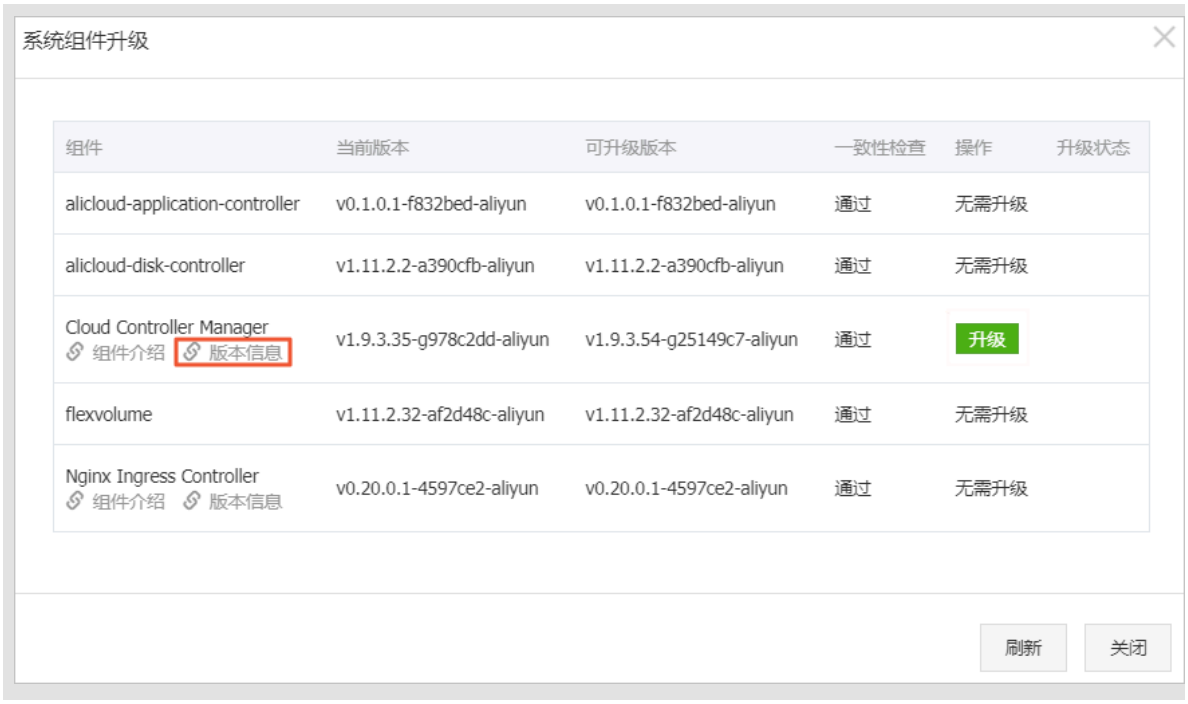
3. 选择目标集群，单击操作列更多 > 系统组件升级。

4. 选择需要升级的系统组件，单击操作列升级，升级状态显示升级中。



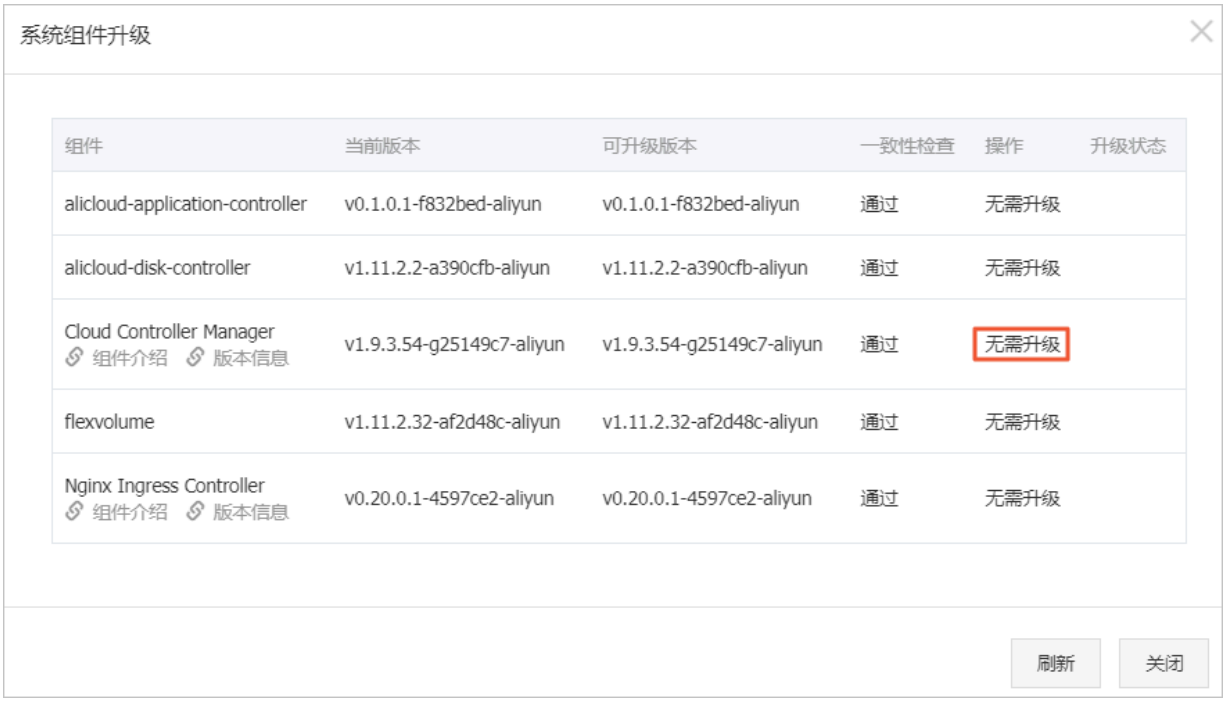
说明:

单击版本信息，可查看系统组件更新的具体内容。



预期结果

在系统组件升级页面，目标系统组件对应的操作列显示为无需升级。



3.5 扩容集群

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。

背景信息

目前不支持集群中 Master 节点的扩容。

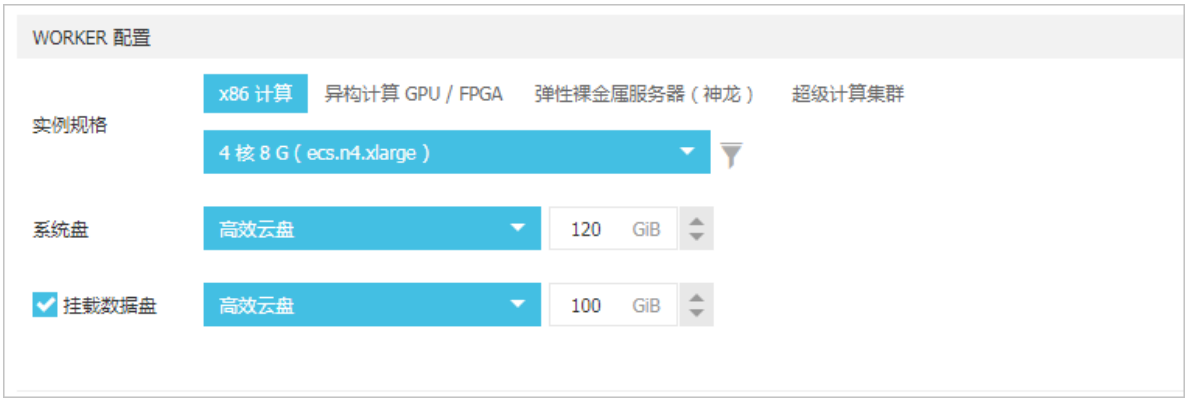
操作步骤

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 > 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扩容。

4. 进入集群扩容页面，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3 个扩容到 5 个。

5. 设置Worker节点的配置。



- 实例规格：支持选择的实例规格。参见#unique_25。
- 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。

6. 配置节点的登录方式。

- 设置密钥。

  您需要在扩容节点时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#unique_26。密钥对创建完毕后，设置该密钥对作为登录节点的凭据。

- 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。

7. 设置RDS白名单。

将节点 IP 添加到 RDS 实例的白名单。

8. 为节点绑定标签。

   输入键和对应的值，单击添加。

   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。

9. 单击提交。

后续步骤

扩容完成后，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 3 变成了 5。

3.6 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 > 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。

4. 弹出删除集群，确认内容无误后，勾选我已知晓以上信息并确认删除集群，单击确定。

下图为示例截图，删除集群时以实际截图为准。

后续步骤

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ROS创建的资源及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，单击 确定，即可删除集群和 ROS 资源栈。

说明:

如果在删除集群提示页面，勾选保留资源，则不会释放这些资源，您需要手动释放。有关如何如果排查不能释放的资源，可参见#unique_59。

3.7 集群管理最佳实践

3.7.1 ECS选型

本文介绍构建Kubernetes集群时推荐的ECS选型。

集群规划

目前在创建Kubernetes集群时，存在使用很多小规格ECS的情况，这样做存在以下弊端：

- 小规格Woker ECS的网络资源受限。
- 如果一个容器基本可以占用一个小规格ECS，此ECS的剩余资源就无法利用（构建新的容器或者是恢复失败的容器），在小规格ECS较多的情况下，存在资源浪费。

使用大规格ECS的优势：

- 网络带宽大，对于大带宽类的应用，资源利用率高。
- 容器在一台ECS内建立通信的比例增大，减少网络传输。

- 拉取镜像的效率更高。因为镜像只需要拉取一次就可以被多个容器使用。而对于小规格的ECS拉取镜像的次数就会增多。若需要联动ECS伸缩集群，则需要花费更多的时间，反而达不到立即响应的目的。

选择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	4核8G(不建议2核4G)
6-20个节点	4核16G
21-100个节点	8核32G
100-200个节点	16核64G

选择Worker节点规格

- 确定整个集群的日常使用的总核数以及可用度的容忍度。

  例如：集群总的核数有160核，可以容忍10%的错误。那么最小选择10台16核ECS，并且高峰运行的负荷不要超过160*90%=144核。如果容忍度是20%，那么最小选择5台32核ECS，并且高峰运行的负荷不要超过160*80%=128核。这样就算有一台ECS出现故障，剩余ECS仍可以支持现有业务正常运行。
- 确定CPU：Memory比例。对于使用内存比较多的应用例如java类应用，建议考虑使用1:8的机型。

选用裸金属神龙服务器

以下两种场景，建议选用裸金属神龙服务器：

- 集群日常规模能够达到1000核。一台神龙服务器至少96核，这样可以通过10～11台神龙服务器即可构建一个集群。

- 快速扩大较多容器。例如：电商类大促，为应对流量尖峰，可以考虑使用神龙服务来作为新增节点，这样增加一台神龙服务器就可以支持很多个容器运行。

使用神龙服务器构建集群，存在以下优势：

- 超强网络： 配备RDMA（Remote Direct Memory Access）技术。通过Terway容器网络，充分发挥硬件性能，跨宿主机容器带宽超过9Gbit/s。
- 计算性能零抖动：自研芯片取代Hypervisor，无虚拟化开销，无资源抢占。
- 安全：物理级别加密，支持Intel SGX加密，可信计算环境，支持区块链等应用。

3.7.2 高可靠推荐配置

为了保证应用可以稳定可靠的运行在Kubernetes里，本文介绍构建Kubernetes集群时的推荐配置。

磁盘类型及大小

磁盘类型

- 推荐选择SSD盘。
- 对于Worker节点，创建集群时推荐选择挂载数据盘。这个盘是专门提供给`/var/lib/docker`存放本地镜像。可避免后续如果镜像太多根磁盘容量不够的问题。在运行一段时间后，本地会存在很多无用的镜像。比较快捷的方式就是，先下线这台机器，重新构建这个磁盘，然后再上线。

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Kubernetes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。

Kubernetes集群中，ECS操作系统占用3G左右的磁盘空间，建议预留ECS操作系统8G的空间。剩余的磁盘空间由Kubernetes资源对象使用。

是否立即构建Worker节点

创建集群时，节点类型若选择：

- 按量付费，可在创建集群时，构建Worker节点。
- 包年包月，创建集群时，可先不构建Worker节点，根据后续需求，单独购买ECS添加进集群。

网络选择

- 如果需要连接外部的一些服务，如RDS等，则需要考虑复用原有的VPC，而不是创建一个新的VPC。因为VPC间是隔离的，您可以通过创建一个新的交换机，把运行Kubernetes的机器都放在这个交换机下，从而便于管理。

- 在Kubernetes集群创建时提供两种网络插件：Terway和Flannel，可参考#unique_27。
- Pod网络CIDR不能设置太小，如果太小，可支持的节点数量就会受限。这个值的设置需要与高级选项中的节点Pod数量综合考虑。例如：Pod网络CIDR的网段是/16，那么就有256*256个地址，如果每个节点Pod数量是128，则最多可以支持512个节点。

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区域的容灾能力。同时也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

在使用Kubernetes集群时，经常会遇到：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的问题。

为避免上述问题，在Kubernetes中部署Pod时，您可以指定这个Pod需要Request及Limit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Nginx这个Pod需要1核CPU，1024M的内存，运行中实际使用不能超过2核CPU和4096M内存。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources: # 资源声明
      requests:
        memory: "1024Mi"
        cpu: "1000m"
      limits:
        memory: "4096Mi"
        cpu: "2000m"
```

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。

日常运维

- 日志

  创建集群时，请勾选日志服务。

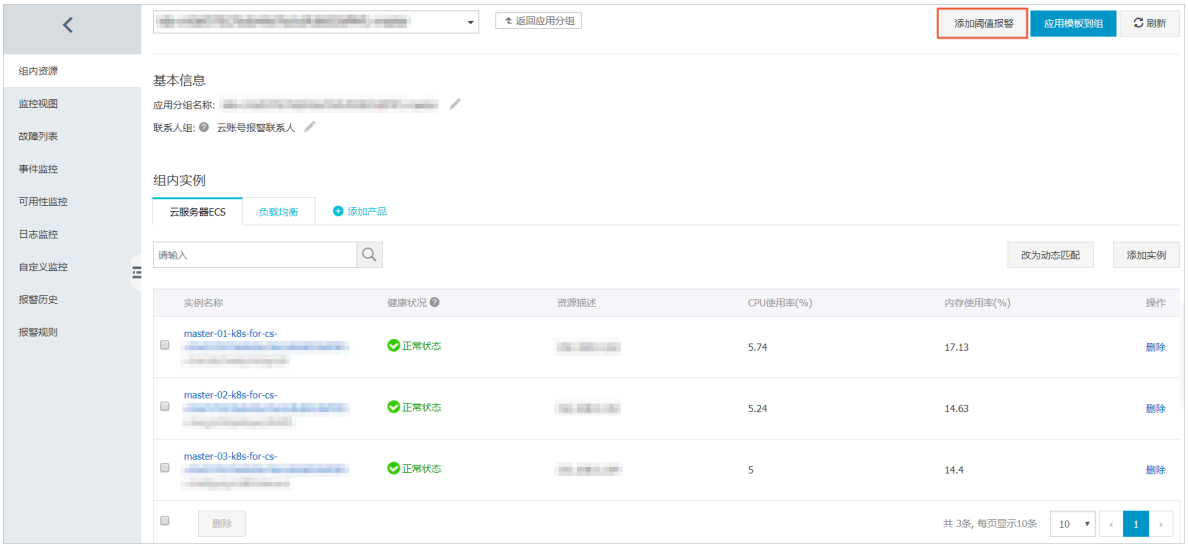
- 监控

  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。

> **说明:**
>
> - 对于ECS的监控，日常运维请设置cpu，memory，磁盘等的报警规则。且尽量将*/var/lib/docker*放在一个独立的盘上。



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。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，这种策略不再适用。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，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，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。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，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，在一个容器里放多个进程：监控进程、日志进程、sshd进程、甚至整个Systemd。这样操作存在两个问题：

- 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。
- 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有进程退出，也不会对容器做任何操作，但是实际上容器已经不能正常工作了。

如果有几个进程需要协同工作，在Kubernetes里也可以实现，例如：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（Single Point of Failure）

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。

3.7.3 更新Kubernetes集群已过期的证书

当集群证书已过期时，通过kubectl或api接口调用的方式与集群apiserver的通讯都将被禁止，因此我们无法通过模板部署的方式来自动更新集群各节点上的相应过期证书；此时集群管理员可以登录至集群各节点，通过如下`docker run`启动容器的方式执行目标节点的证书更新任务。

更新Master节点已过期的证书

1. 以root权限登录任意Master节点。
2. 在任意目录下执行以下命令，更新Master节点已经过期的证书。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role master
```

3. 在集群的每个Master节点，重复上述步骤，完成所有Master节点已过期的证书更新。

更新Worker节点已过期的证书

1. 以root权限登录任意Master节点。
2. 执行以下命令，获取集群rootCA私钥。

```
$ cat /etc/kubernetes/pki/ca.key
```

3. 执行以下命令，获取通过base64编码后集群的根私钥。
   - 若获取的集群rootCA私钥，有一行空行，执行以下命令：

     ```
     $ sed '1d' /etc/kubernetes/pki/ca.key| base64 -w 0
     ```

   - 若获取的集群rootCA私钥，无空行，执行以下命令：

     ```
     $ cat /etc/kubernetes/pki/ca.key | base64 -w 0
     ```

4. 以root权限登录任意Worker节点。
5. 在任意目录下执行如下命令，更新Worker节点已经过期的证书。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role node --rootkey ${base64CAKey}
```

说明：

${base64CAKey}为步骤3获取的通过base64编码后的集群根私钥。

6. 在集群的每个Worker节点，重复上述步骤，完成所有Worker节点已过期的证书更新。

3.7.4 更新Kubernetes集群即将过期的证书

本文介绍如何更新Kubernetes集群即将过期的证书。您可以通过控制台操作，也可以通过命令行自动化一键式更新所有节点证书，也可以手动更新master和worker节点证书。

前提条件

- 您已经成功创建一个Kubernetes集群，参见#unique_48。
- 您可以通过kubectl连接集群，参见#unique_37。

控制台更新所有节点证书

在容器服务管理控制台，单击目标集群的更新证书更新即将过期的证书，请参考#unique_66。

命令行自动更新所有节点证书

登录集群任意master节点，执行以下命令完成集群所有节点的证书更新。

```
$ curl http://aliacs-k8s-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/
public/cert-update/renew.sh | bash
```

执行结果

1. 执行以下命令，查看集群master和worker节点状态。

```
$ kubectl get nodes
```

```
[root@ ~]# kubectl get nodes
NAME                          STATUS    ROLES     AGE       VERSION
cn-hangzhou.                  Ready     <none>    23d       v1.11.2
cn-hangzhou.                  Ready     <none>    23d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     <none>    47d       v1.11.2
cn-hangzhou.                  Ready     <none>    47d       v1.11.2
[root@ ~]#
```

2. 执行以下命令，当master节点对应的SUCCESSFUL均为1，worker节点对应的SUCCESSFUL为集群worker节点数时，所有证书完成更新。

```
$ kubectl get job -nkube-system
```

```
[root@                    ~]# kubectl get job -nkube-system
NAME                          DESIRED   SUCCESSFUL   AGE
aliyun-cert-renew-master-1    1         1            6m
aliyun-cert-renew-master-2    1         1            5m
aliyun-cert-renew-master-3    1         1            5m
aliyun-cert-renew-worker      4         4            4m
cert-job-2                    1         1            22h
cert-job-3                    1         1            22h
cert-job-4                    1         1            22h
cert-node-2                   4         4            19h
```

手动更新master节点证书

1. 任意路径下，复制以下内容，创建*job-master.yml*文件。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: 1
  parallelism: 1
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                - ${hostname}
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - master
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
```

```
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
      volumes:
      - hostPath:
          path: /
          type: Directory
```

```
        name: ${jobname}
```

2. 获取集群master节点个数和hostname：

   · 方法一：

   执行以下命令：

   ```
   $ kubectl get nodes
   ```

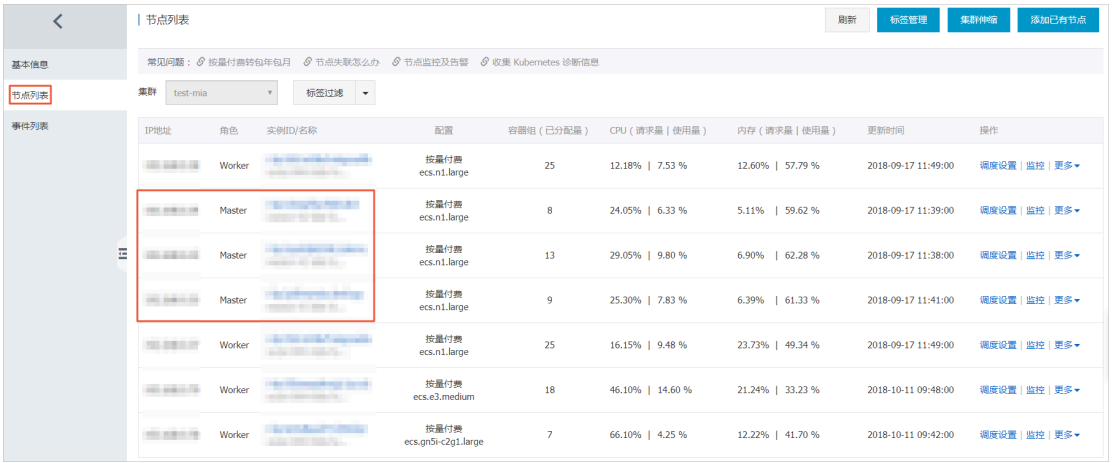
   ```
   [root@                    ~]# kubectl get nodes
   NAME                      STATUS    ROLES     AGE       VERSION
   cn-hangzhou.i             Ready     <none>    22d       v1.11.2
   cn-hangzhou.i             Ready     <none>    22d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     <none>    46d       v1.11.2
   cn-hangzhou.i             Ready     <none>    46d       v1.11.2
   [root@                    ~]#
   ```

   · 方法二：

   a. 登录容器服务管理控制台。

   b. 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。

   c. 选择目标集群，单击集群名称，查看集群详细信息，单击左侧导航栏的节点列表获取master个数和hostname。

3. 执行以下命令替换*job-master.yml*文件中指定的变量${jobname}和${hostname}。

```
$ sed 's/${jobname}/cert-job-2/g; s/${hostname}/hostname/g' job-
master.yml > job-master2.yml
```

其中：

- ${jobname}为Job和Pod的名称，此处设置为cert-job-2。
- ${hostname}为集群master节点的名称，此处请将hostname替换为步骤2中查看到的master名称。

4. 执行以下命令创建Job。

```
$ kubectl create -f job-master2.yml
```

5. 执行以下命令查看Job状态，当SUCCESSFUL均为1时，证书完成更新。

```
$ kubectl get job -nkube-system
```

6. 重复执行步骤3~5，完成所有master节点的证书更新。

```
[root@ ~]# kubectl get job -nkube-system
NAME         DESIRED   SUCCESSFUL   AGE
cert-job-2   1         1            22m
cert-job-3   1         1            2m
cert-job-4   1         1            1m
[root@ ~]#
```

手动更新worker节点证书

1. 任意路径下，复制以下内容，创建*job-node.yml*文件。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: ${nodesize}
  parallelism: ${nodesize}
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: job-name
                operator: In
                values:
                - ${jobname}
            topologyKey: kubernetes.io/hostname
      containers:
      - command:
```

```
        - /renew/upgrade-k8s.sh
        - --role
        - node
        - --rootkey
        - ${key}
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      volumes:
      - hostPath:
          path: /
          type: Directory
        name: ${jobname}
```

说明:

如果worker节点带有taint，需要在*job-node.yml*文件中增加对该taint的toleration s，即在securityContext: {}与volumes:之间增加以下内容（若有n个带有taint的worker节点，请复制n次）：

```
      tolerations:
      - effect: NoSchedule
        key: ${key}
        operator: Equal
        value: ${value}
```

获取${name}和${value}的方法如下：

a. 任意路径下，复制以下内容，创建*taint.tml*文件。

```
{{printf "%-50s %-12s\n" "Node" "Taint"}}
{{- range .items}}
    {{- if $taint := (index .spec "taints") }}
        {{- .metadata.name }}{{ "\t" }}
        {{- range $taint }}
            {{- .key }}={{ .value }}:{{ .effect }}{{ "\t" }}
        {{- end }}
        {{- "\n" }}
    {{- end}}
```

```
{{- end}}
```

b. 执行以下命令，查询带有taint的worker节点的${name}和${value}。

```
$ kubectl get nodes -o go-template-file="taint.tml"
```

```
[root@ ~]# kubectl get nodes -o go-template-file="taint.tml"
Node                                    Taint
cn-hangzhou.i-                          key1=value1:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
```

2. 执行以下命令，获取集群的CAKey。

```
$ sed '1d' /etc/kubernetes/pki/ca.key | base64 -w 0
```

3. 执行以下命令替换*job-node.yml*文件中指定的变量${jobname}、${nodesize}和${key}。

```
$ sed 's/${jobname}/cert-node-2/g; s/${nodesize}/nodesize/g; s/${key}/key/g' job-node.yml > job-node2.yml
```

其中：

- ${jobname}为Job和Pod的名称，此处设置为cert-node-2。
- ${nodesize}为worker节点个数，获取方法可参考手动更新master节点证书的步骤2。此处请将nodesize替换为集群的worker个数。
- ${key}为集群的CAKey，此处请将key替换为#unique_67/unique_67_Connect_42_section_k4z_skb_rfb步骤3获取到的CAKey。

4. 执行以下命令创建 Job。

```
$ kubectl create -f job-node2.yml
```

5. 执行以下命令查看Job状态，当SUCCESSFUL为集群worker节点数时，证书完成更新。

```
$ kubectl get job -nkube-system
```

```
[root@ ~]# kubectl get job -nkube-system
NAME          DESIRED   SUCCESSFUL   AGE
cert-job-2    1         1            1h
cert-job-3    1         1            47m
cert-job-4    1         1            46m
cert-node-2   4         4            1m
[root@ ~]#
```

3.7.5 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

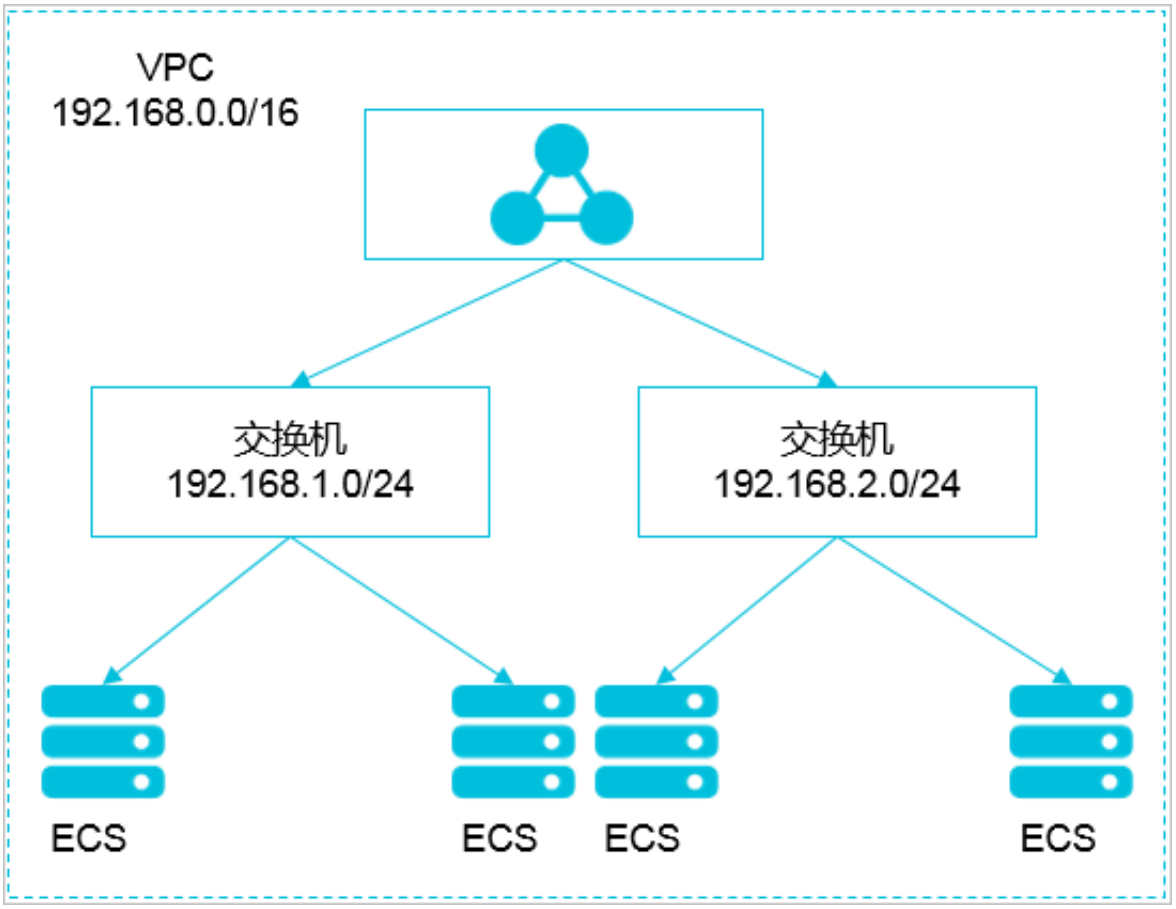
VPC 网段

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。



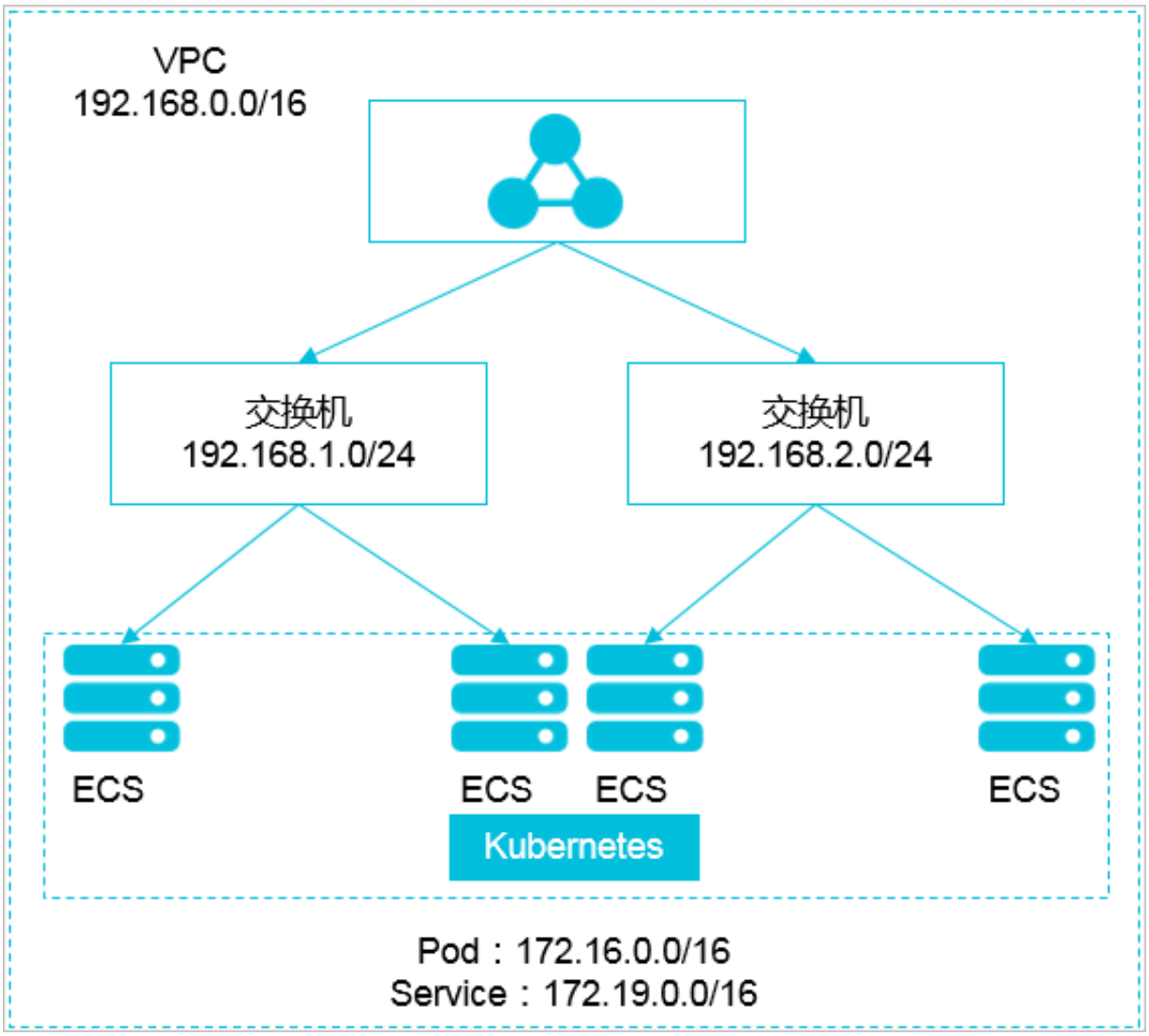
Pod 地址段

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 172.16.0.0/16，172.17.0.0/16等，因为这些地址都涵盖在 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

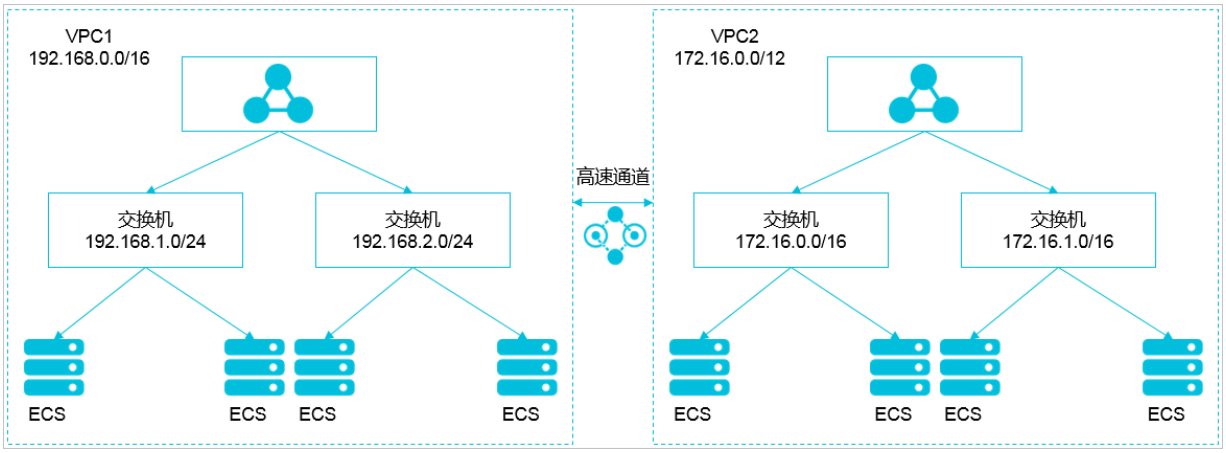
一个 VPC 下创建多个 Kubernetes 集群。在默认的网络模式下（Flannel），Pod 的报文需要通过 VPC 路由转发，容器服务会自动在 VPC 路由上配置到每个 Pod 地址段的路由表。所有 Kubernetes 集群的 Pod 地址段不能重叠，但 Service 地址段可以重叠。

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。

3.8 集群管理FAQ

本文主要为您介绍集群管理常见问题。

- 集群创建失败
- 删除 Kubernetes 集群失败：ROS stack 无法删除
- 收集 Kubernetes 诊断信息

集群创建失败

查看失败原因。

您可以通过查看集群的创建事件来查看集群创建失败的原因。

a) 登录 ROS 管理控制台。

b) 选择集群所在的地域，选择所需的集群并单击右侧的管理，单击左侧导航栏中的事件，将鼠标移动到失败事件上查看具体的失败报错信息。

上图中的报错信息显示由于 VPC 达到配额导致集群创建失败。

c) 您可以参考以下失败异常码及解决方法进行修复。

- Resource CREATE failed: ResponseException: resources.k8s_SNat_Eip: Elastic IP address quota exceeded Code: QuotaExceeded.Eip

  解决方法：

  释放多余的 EIP 或者提交 VPC 工单提高 EIP 限额。

- Resource CREATE failed: ResponseException: resources.k8s_master_slb_internet: The maximum number of SLB instances is exceeded. Code: ORDER.QUANTITY_INVALID

  解决方法：

  释放多余的 SLB 实例或者提交 SLB 工单提高 SLB 限额。

- Resource CREATE failed: ResponseException: resources.k8s_vpc: VPC quota exceeded. Code: QuotaExceeded.Vpc

  解决方法：

  释放多余的 VPC 或者提交 VPC 工单提高 VPC 限额。

- Status Code: 403 Code: InvalidResourceType.NotSupported Message: This resource type is not supported;

  解决方法：

  ECS 没有库存或者类型不支持，请选择其他 ECS 规格重试。

- Resource CREATE failed: ResponseException: resources.k8s_master_1: The specified image does not support cloud-init. Code: ImageNotSupportCloudInit

  解决方法：

  使用自定义镜像创建集群，自定义镜像必须是基于最新的 CentOS 镜像。

- Resource CREATE failed: ResponseException: resources.k8s_nodes: The resource is out of stock in the specified zone. Please try other types, or choose other regions and zones. Code: OperationDenied.NoStock

  解决方法：

  当前所选实例规格已售罄，请选择其他可用区或实例规格。
- Resource CREATE failed: ResponseException: resources.k8s_NAT_Gateway: A route entry already exists, which CIDR is '0.0.0.0/0' Code: RouterEntryConflict.Duplicated

  解决方法：

  当前 VPC 路由表内已存在默认路由，请移除默认路由，或取消勾选为专有网络创建SNAT后重试。
- Resource CREATE failed: ResponseException: resources.KubernetesWorkerRole: The number of role is limited to 200. Code: LimitExceeded.Role

  解决方法：

  RAM 角色数量已达到配额限制，请清理部分角色或提交 RAM 工单提高配额。
- Resource CREATE failed: ResponseException: resources.k8s_NAT_Gateway: The Account failed to create order. Code: OrderFailed

  解决方法：

  下单失败，请提交工单咨询。
- Resource CREATE failed: ResponseException: resources.k8s_master_1: This operation is forbidden by Aliyun RiskControl system. Code: Forbidden.RiskControl

  解决方法：

  您的账户出现异常，详情请联系客服
- Resource CREATE failed: ResponseException: resources.k8s_master_slb_internet: Your account does not have enough balance. Code: PAY.INSUFFICIENT_BALANCE

  解决方法：

  创建按量付费实例需要您账户余额大于100元，请先充值。

- Resource CREATE failed: ResponseException: resources.k8s_nodes: Your account does not have enough balance. Code: InvalidAccountStatus.NotEnoughBalance

  解决方法：

  创建按量付费实例需要您账户余额大于100元，请先充值。
- Resource CREATE failed: WaitConditionFailure: resources.k8s_node_cloudinit_wait_cond: See output value for more details.

  解决方法：

  配置集群出错，请稍后重试或提交工单咨询。
- Resource CREATE failed: WaitConditionTimeout: resources.k8s_master1_cloudinit_wait_cond: 0 of 2 received:

  解决方法：

  配置集群出错，请稍后重试或提交工单咨询。
- Resource CREATE failed: ResponseException: resources.k8s_master_1: The request processing has failed due to some unknown error. Code: UnknownError

  解决方法：

  未知错误，请稍后重试或提交工单咨询。
- Resource CREATE failed: ResponseException: resources.k8s_nodes: The request processing has failed due to some unknown error. Code: UnknownError

  解决方法：

  未知错误，请稍后重试或提交工单咨询。

删除 Kubernetes 集群失败：ROS stack 无法删除

用户在 ROS 创建的资源下手动添加了一些资源（比如在 ROS 创建的 VPC 下手动添加了一个 VSwitch），ROS 是没有权限删除这些资源的。这就会导致 ROS 删除 Kubernetes 资源时无法处理该 VPC，最终导致删除失败。

说明：

有关创建 Kubernetes 集群时 ROS 自动创建的资源，参见#unique_48。

1. 集群删除失败时（集群的状态显示删除失败），跳转到 ROS 管理控制台。

2. 选择集群所在的地域，找到集群对应的资源栈 `k8s-for-cs-{cluster-id}`，可以看到其状态为删除失败。

3. 单击资源栈的名称进入资源栈详情页面，单击左侧导航栏中的资源。

   您可以看到哪些资源删除失败了。本示例中负载均衡下的 VSwitch 删除失败。

4. 进入删除失败的资源所在产品的控制台，并找到该资源。

   本示例中，登录 VPC 管理控制台，找到集群所在的 VPC，并在该 VPC 下找到删除失败的 VSwitch。

5. 单击 VSwitch 右侧的删除 尝试手动删除。

   本示例中，由于 VSwitch 下还有资源未释放，所以删除失败。

   手动释放该 VSwitch 下的资源，然后再次尝试删除该 VSwitch。

6. 使用类似的方法手动删除 Kubernetes 集群下所有删除失败的资源，然后再次尝试删除 Kubernetes 集群。

收集 Kubernetes 诊断信息

当Kubernetes集群出现问题，或者节点异常时，您需要收集Kubernetes诊断信息。

> 说明:
>
> - 当集群异常时，需要在master节点完成收集。
> - 当worker节点异常时，则需要在master节点和异常的worker节点上完成收集。

完成以下步骤在master/worker节点收集诊断信息。

1. 在 master/worker节点下载诊断脚本，并增加运行权限。

```
curl -o /usr/local/bin/diagnose_k8s.sh http://aliacs-k8s-cn-hangzhou
.oss-cn-hangzhou.aliyuncs.com/public/diagnose/diagnose_k8s.sh
chmod u+x /usr/local/bin/diagnose_k8s.sh
```

2. 执行诊断脚本。

```
diagnose_k8s.sh
 ......
 + echo 'please get diagnose_1514939155.tar.gz for diagnostics'
 ##每次执行诊断脚本，产生的日志文件的名称不同
```

```
please get diagnose_1514939155.tar.gz for diagnostics
+ echo '请上传 diagnose_1514939155.tar.gz'
请上传 diagnose_1514939155.tar.gz
```

3. 列出并上传产生的日志文件。

```
cd /usr/local/bin
ls -ltr|grep diagnose_1514939155.tar.gz          ##注意替换为生成的日志文件名
```

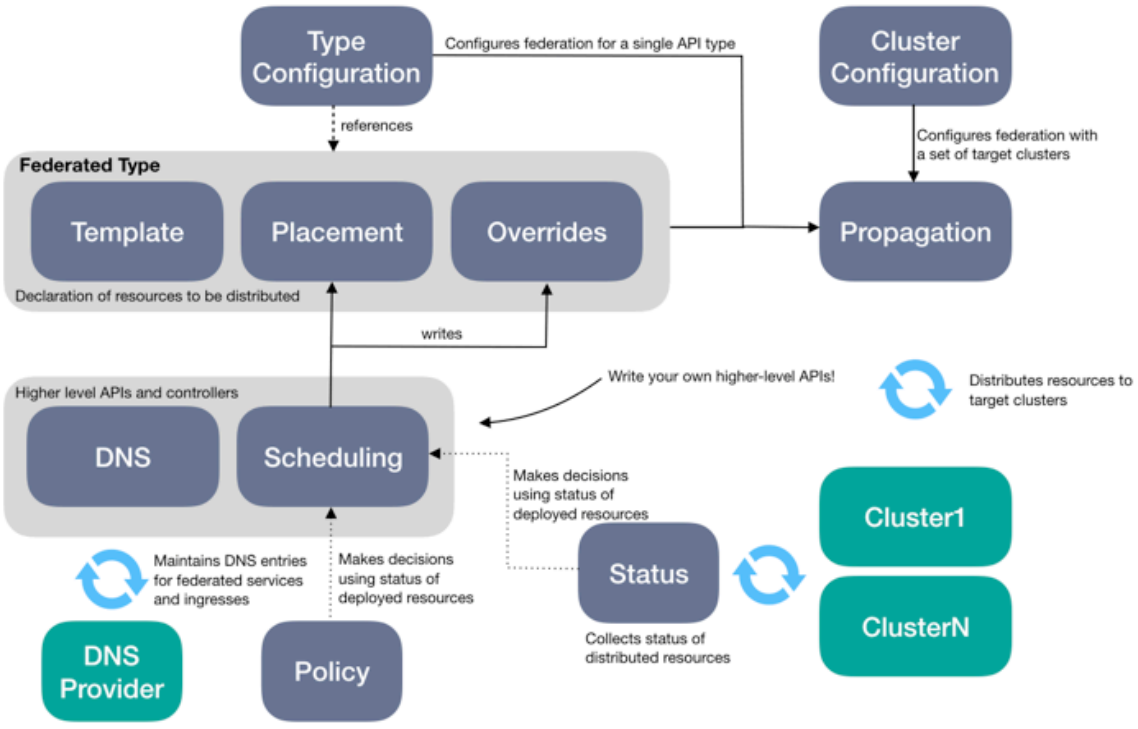
4 多集群管理

4.1 概述

在业务形态和云灾备场景下，需要服务进行多地域部署，这就需要对 Kubernetes 不同地域的集群进行管理。kubefed（集群联邦）是 Kubernetes 社区中重要的多云管理项目，实现了跨地区跨服务商管理多个 Kubernetes 集群的功能，通过 kubefed 主集群暴露一组 API，对多个集群的配置进行管理。

下面介绍如何快速在阿里云 Kubernetes 容器服务上搭建和使用 kubefed，详情请参见GitHub链接。

kubefed架构



kubefed 由两种类型的配置进行管理：

- Type configuration：说明哪些 API 类型 kubefed 需要管理。

  Type configuration 中有三个概念

  - Template：定义多集群资源模板。
  - Placement：定义多集群资源部署的集群信息。
  - Override：定义每个单独集群 Template 中特殊的字段属性。

- Cluster configuration：说明哪些集群 kubefed 需要管理。

因此，kubefed 可以定义多集群的 CRD 接口，例如，FederatedNamespace、FederatedService 和 FederatedIngress 等。

4.2 部署集群联邦

前提条件

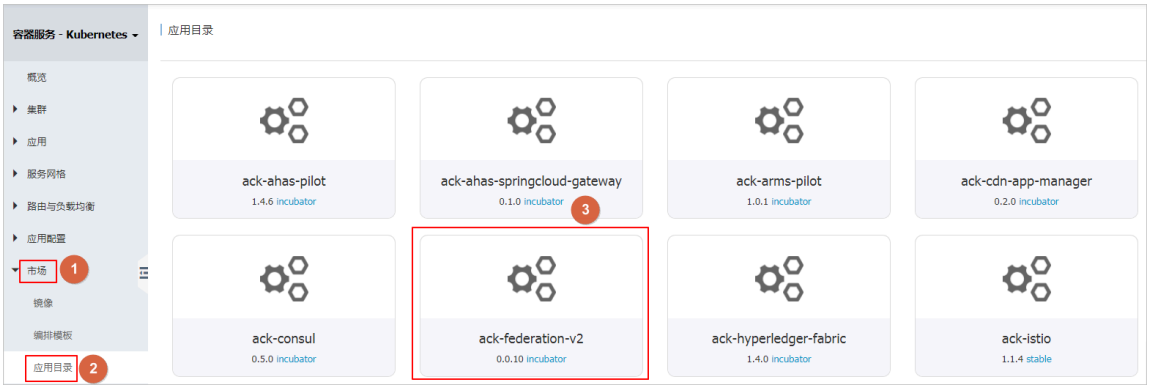
- 您已经成功创建两个 Kubernetes 集群，参见#unique_73。
- 通过CloudShell 连接 Kubernetes 集群。
- 您已经安装kuberfedctl工具，且 kubefedctl 工具的版本与联邦服务版本相配套。

操作步骤

1. 部署联邦服务。

   a) 登录容器服务管理控制台。

   b) 在 Kubernetes 菜单下，选择市场 > 应用目录，在右侧单击ack-federation-v2。



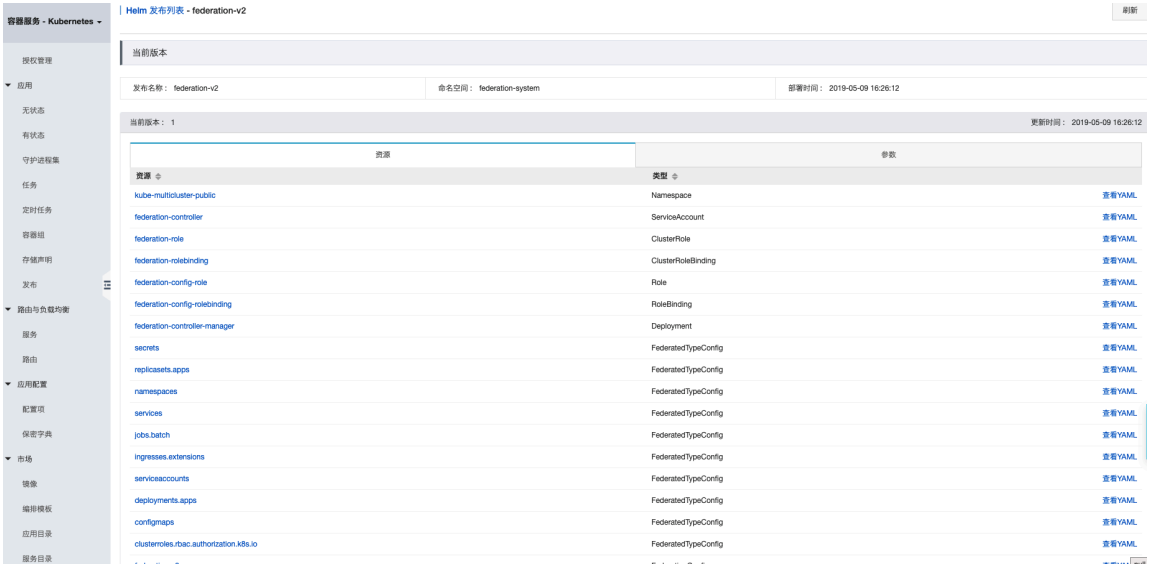
   c) 在应用目录 -ack-federation-v2 中，单击参数，可以通过修改参数配置进行定制化，如下所示。

说明:

resources下相关配置为federation-controller-manager服务资源限制配置，您可以根据接入集群大小进行配置，其他保持默认值即可。

d) 在右侧的创建页面，在右侧选择federation主集群cluster1，同时可以看到命名空间已设定为federation-system，发布名称已设定为federation-v2，然后单击创建。

几秒钟后，在弹出Helm 发布列表 - federation-v2页面，可以看到联邦服务在集群中被创建出来。



2. 配置多集群。

a) 以cluster1为主集群，执行`kubectl config get-contexts`命令，看到如下信息时，表示已配置完成。

```
CURRENT   NAME       CLUSTER    AUTHINFO             NAMESPACE
*         cluster1   cluster1   kubernetes-admin1
          cluster2   cluster2   kubernetes-admin2
```

b) 执行如下命令，组建federation集群。

```
kubefedctl join cluster1 --cluster-context cluster1 \
    --host-cluster-context cluster1 --add-to-registry --v=2
kubefedctl join cluster2 --cluster-context cluster2 \
    --host-cluster-context cluster1 --add-to-registry --v=2
```

说明:

如果有三个及以上集群，请参考上面第二条命令将集群加入到集群联邦中，以此类推。

```
kubefedctl join cluster1 --cluster-context cluster1 \
    --host-cluster-context cluster1 --add-to-registry --v=2
kubefedctl join cluster2 --cluster-context cluster2 \
    --host-cluster-context cluster1 --add-to-registry --v=2
kubefedctl join cluster3 --cluster-context cluster3 \
```

```
    --host-cluster-context cluster1 --add-to-registry --v=2
```

c) 执行`kubectl get federatedcluster -n federation-system`命令，看到如下信息说明多集群已经加入成功。

```
NAME       READY   AGE
cluster1   True    23s
cluster2   True    10s
```

4.3 部署多集群联邦应用

前提条件

- 您已经成功创建两个 Kubernetes 集群，参见#unique_73。
- 通过CloudShell 连接 Kubernetes 集群。
- 您已经成功部署集群联邦，参见部署集群联邦。

操作步骤

1. 创建并拷贝以下内容到*federated-namespace.yaml*文件中，并在主集群上`cluster1`执行`kubectl apply -f federated-namespace.yaml`命令，创建FederatedNamespace。

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
---
apiVersion: types.federation.k8s.io/v1alpha1
kind: FederatedNamespace
metadata:
  name: test-namespace
  namespace: test-namespace
spec:
  placement:
    clusterNames:
    - cluster1
    - cluster2
```

2. 创建并拷贝以下内容到*federated-deployment.yaml*文件中，并在主集群上`cluster1`执行`kubectl apply -f federated-deployment.yaml`命令，部署FederatedDeployment。

```
apiVersion: types.federation.k8s.io/v1alpha1
kind: FederatedDeployment
metadata:
  name: test-deployment
  namespace: test-namespace
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

```
      replicas: 2
      selector:
        matchLabels:
          app: nginx
      template:
        metadata:
          labels:
            app: nginx
        spec:
          containers:
          - image: nginx
            name: nginx
            resources:
              limits:
                cpu: 500m
              requests:
                cpu: 200m
  placement:
    clusterNames:
    - cluster1
    - cluster2
```

3. 执行如下命令，查看联邦集群中各个Deployment详情。

   当看到如下信息时，表示应用部署完成。

```
kubectl get deployment -n test-namespace --context cluster1
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
test-deployment   2         2         2            2           71s
kubectl get deployment -n test-namespace --context cluster2
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
test-deployment   2         2         2            2           77s
```

4.4 接入外部Kubernetes集群

您可以通过容器服务控制台接入现有的外部Kubernetes集群，并通过容器服务控制台进行管理。

前提条件

- 您需要开通容器服务、弹性伸缩（ESS）服务和访问控制（RAM）服务。

  登录 容器服务管理控制台 、 RAM 管理控制台 和 弹性伸缩控制台 开通相应的服务。
- 修改Kubernetes集群仍然必须在原有的集群中进行完成，包括添加与删除节点、升级Kubernetes集群版本以及更改Kubernetes组件参数等。
- 该功能目前只针对白名单客户开放。

操作步骤

1. 创建接入集群。

   a) 登录容器服务管理控制台。

   b) 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。

   c) 单击页面右上角的创建 Kubernetes 集群，在弹出的选择集群模板中，选择接入已有集群，单击创建。

   默认进入接入已有集群配置页面。

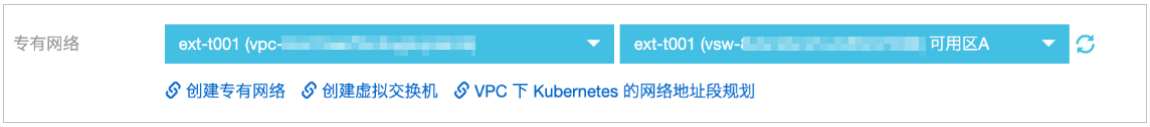
   d) 填写集群的名称。

   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（-）。

   e) 选择集群所在的地域和可用区。

   f) 设置集群的网络。Kubernetes 集群仅支持专有网络。

   您可以在已有 VPC 列表中选择所需的 VPC 和交换机。



   g) 设置是否启用绑定EIP。

   勾选此选项，会在集群中自动绑定EIP，用于建立集群链接。

   h) 为集群绑定标签。

输入键和对应的值，单击添加。

> **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。

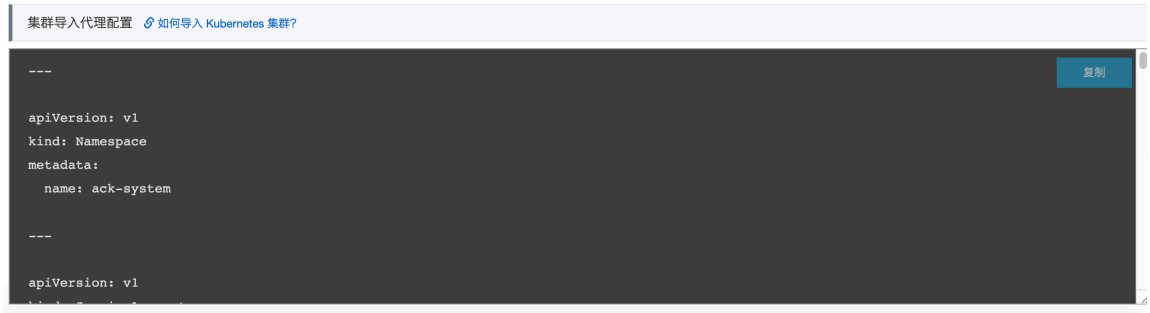
i) 单击创建集群，启动部署。

您可以在集群列表，看到新集群。

2. 将目标集群接入新集群中。

a) 在新集群（本例中为test-external-cluster1）右侧单击管理，进入基本信息页面。

b) 在集群接入代理配置区域单击复制，将以下yaml文件内容拷贝到*agent.yaml*文件中，并执行`kubectl apply -f agent.yaml`命令。



c) 在目标集群中执行`kubectl get all -n ack-system`命令，查看代理运行状况。

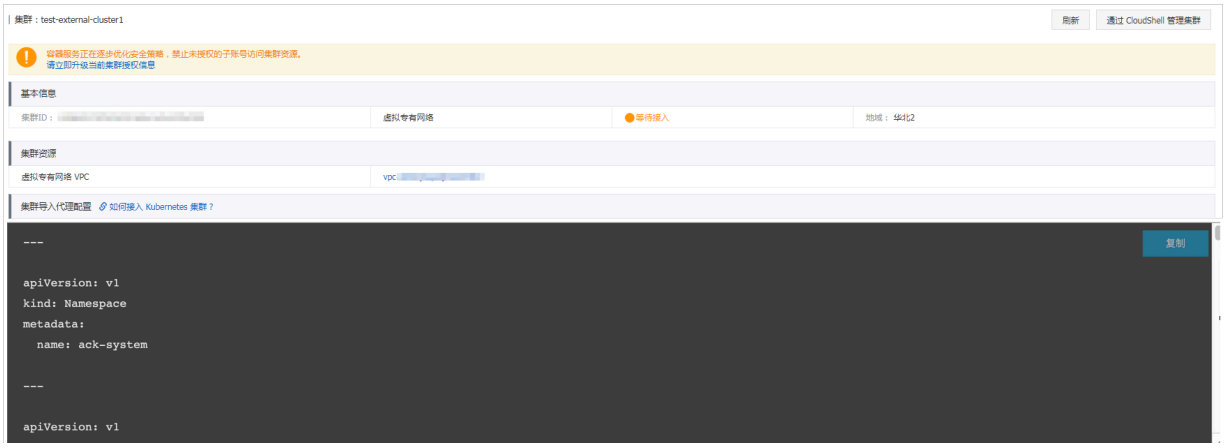
```
NAME                                        READY   STATUS
RESTARTS   AGE
pod/ack-cluster-agent-655b75c987-dwp6b   1/1     Running   0
    9s

NAME                                   DESIRED   CURRENT   UP-TO-DATE
   AVAILABLE   AGE
deployment.apps/ack-cluster-agent   1         1         1
   1           26m
```

```
NAME                                                  DESIRED   CURRENT
  READY   AGE
replicaset.apps/ack-cluster-agent-655b75c987   1         1
  1       26m
```

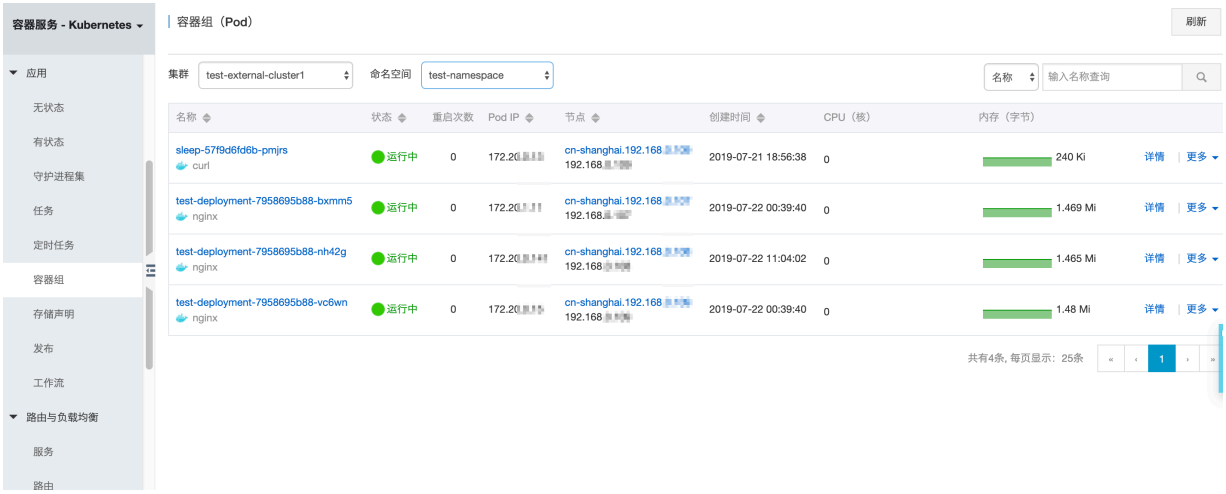
接入成功后，您可以在容器服务管理控制台的 Kubernetes 集群列表页面，看到新集群的状态为运行中。

预期结果

在集群列表页面中，找到新集群test-external-cluster1，单击操作列中的管理，查看新集群的基本信息和连接信息。



您可以#unique_37，执行`kubectl get node`查看新集群test-external-cluster1的节点信息。此时，您可以使用该kubeconfig连接远程的被接入集群，进行应用负载的部署。



您也可以在该接入的集群中通过使用Helm来发布管理应用。

5 节点管理

5.1 添加边缘节点

您可以向已经创建的边缘Kubernetes 集群中添加工作负载节点，工作负载节点需要能够保证和Kubernetes Apiserver的网络联通。边缘Kubernetes集群支持接入云上ECS节点，云上ENS节点，非云节点等。

前提条件

如果之前没有创建过集群，您需要先创建 Kubernetes 边缘托管版集群。

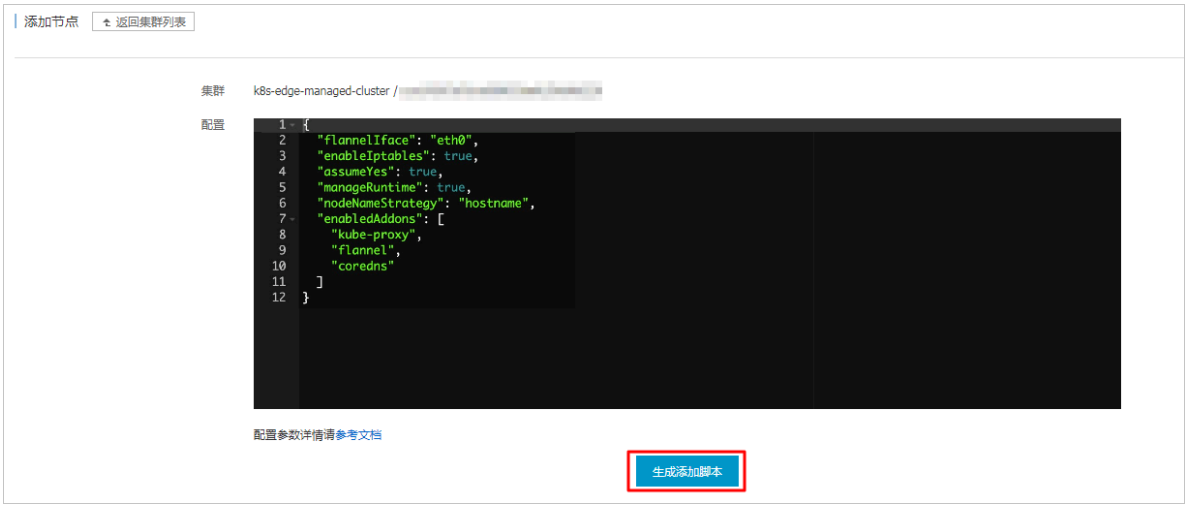
背景信息

- 边缘Kubernetes集群托管服务公测期间，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。
- 仅支持添加操作系统为 CentOS 7.4/7.6 的节点。

添加节点

1. 登录容器服务管理控制台。
2. 添加已有节点。您可以通过以下两个入口进行操作：
   - 入口一：
     a. 在Kubernetes菜单下，单击左侧导航栏中的集群 > 集群，进入 Kubernetes 集群列表页面。
     b. 选择所需的集群并单击右侧的更多 > 添加已有节点。
   - 入口二：
     a. 在Kubernetes菜单下，单击左侧导航栏中的集群 > 节点，进入 Kubernetes 节点列表页面。
     b. 选择所需的集群并单击右上角的添加已有节点。

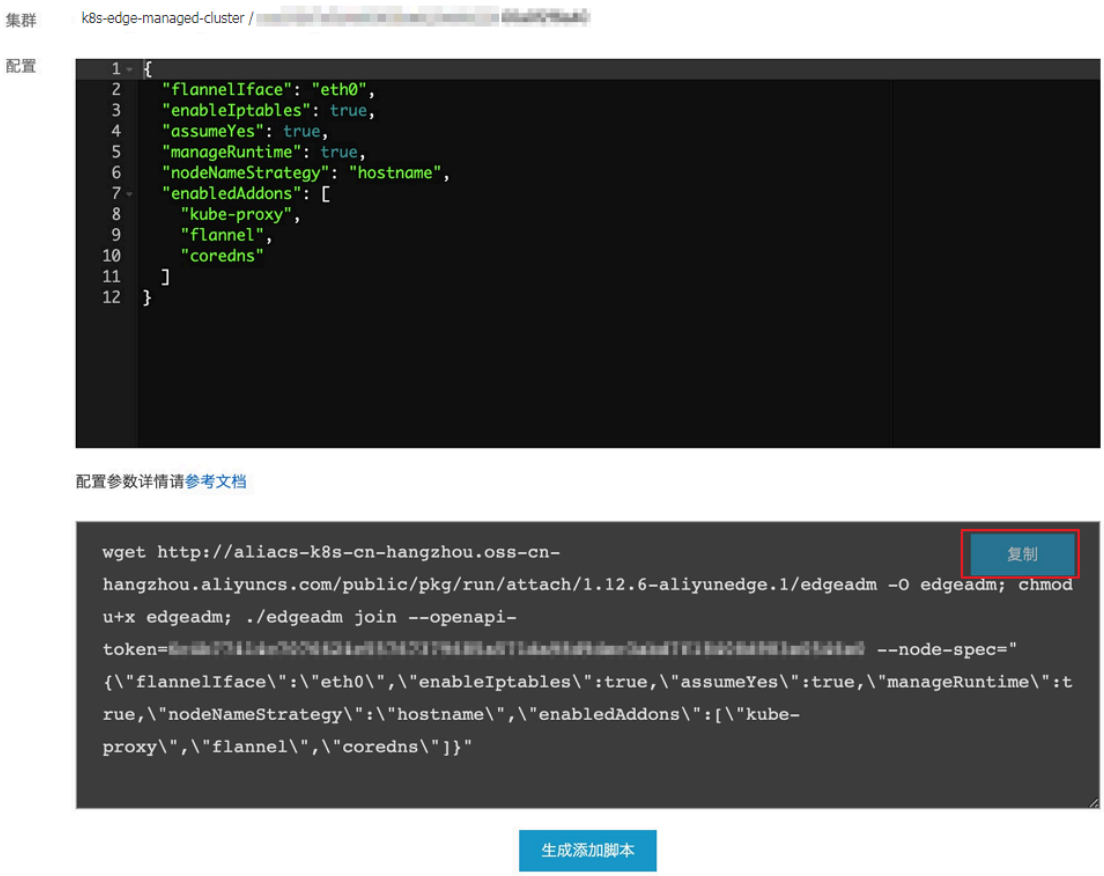
3. 进入添加节点页面，您可以填写节点接入配置，具体的配置参数参见参数列表。



4. 配置完成后，单击生成添加脚本。

5. 单击复制后，登录边缘节点并执行以下命令。

图 5-1: 复制命令




```
wget http://aliacs-k8s-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/
public/pkg/run/attach/1.12.6-aliyunedge.1/edgeadm -O edgeadm; chmod
u+x edgeadm; ./edgeadm join
```

```
--openapi-token=XXXXX --node-spec="{\"flannelIface\":\"eth0\",\"
enableIptables\":true,
\"assumeYes\":true,\"manageRuntime\":true,\"nodeNameStrategy\":\"
hostname\",\"enabledAddons\":[\"kube-proxy\",\"flannel\",\"coredns
\"]}"
```

如果添加边缘节点成功，如下图所示。

```
I0722 12:45:34.815382   29148 join-node_linux_amd64.go:138] [join-node] Checking kubelet, kubeadm, kubectl is exist or not.
I0722 12:45:35.331662   29148 join-node_linux_amd64.go:151] [join-node] Installing kubelet, kubeadm and kubectl.
I0722 12:45:42.532173   29148 join-node_linux_amd64.go:178] [join-node] Install kubelet, kubeadm and kubectl successfully.
I0722 12:45:42.532201   29148 join-node.go:70] [join-node] Resetting kubernetes cluster.
I0722 12:45:42.615758   29148 join-node.go:72] [join-node] Reset kubernetes cluster successfully.
I0722 12:45:42.615783   29148 join-node.go:76] [join-node] Installing cni plugins.
I0722 12:45:42.658745   29148 join-node.go:92] [join-node] Install cni plugins successfully.
I0722 12:45:42.658766   29148 join-node.go:125] [join-node] Setting the flannel interface configuration file.
I0722 12:45:42.658868   29148 join-node.go:134] [join-node] Set the flannel interface configuration file successfully.
I0722 12:45:42.658876   29148 join-node.go:140] [join-node] Setting the flannel ipsec configuration file.
I0722 12:45:42.658902   29148 join-node.go:153] [join-node] Set the flannel ipsec configuration file successfully.
I0722 12:45:42.658915   29148 join-node.go:159] [join-node] Configuring the kubelet service configuration.
I0722 12:45:42.726644   29148 join-node.go:202] [join-node] Config the kubelet service configuration successfully.
I0722 12:45:42.726710   29148 join-node.go:235] [join-node] Start to joining node to cluster.
I0722 12:45:44.330196   29148 join-node.go:250] [join-node] Join node to cluster successfully.
I0722 12:45:44.346703   29148 install.go:165] [install-edgehub] Stop edgehub successfully
I0722 12:45:44.346730   29148 install.go:98] [install-edgehub] Installing the edgehub.
I0722 12:45:44.577286   29148 install.go:209] [install-edgehub] Checking edgehub status
I0722 12:45:54.593565   29148 install.go:214] [install-edgehub] Edgehubb is ok
I0722 12:45:54.593588   29148 install.go:112] [install-edgehub] Install the edgehub successfully.
I0722 12:45:54.593593   29148 install.go:118] [install-edgehub] Reconfiguring the kubelet configuration files.
I0722 12:45:54.600822   29148 install.go:134] [install-edgehub] Reconfigure the kubelet configuration files successfully.
I0722 12:45:54.600842   29148 install.go:135] [install-edgehub] Restarting the kubelet.
I0722 12:46:09.862581   29148 install.go:148] [install-edgehub] Restart the kubelet successfully.
I0722 12:46:09.862696   29148 postcheck.go:72] [post-check] Checking docker status
I0722 12:46:09.865903   29148 postcheck.go:81] [post-check] docker is ok
I0722 12:46:09.865920   29148 postcheck.go:87] [post-check] Checking kubelet status
I0722 12:46:09.868620   29148 postcheck.go:95] [post-check] Kubelet is ok
I0722 12:46:09.868634   29148 postcheck.go:101] [post-check] Checking edgehub status
I0722 12:46:09.877993   29148 postcheck.go:107] [post-check] Edgehubb is ok
I0722 12:46:09.878009   29148 postcheck.go:123] [post-check] Checking addon kube-proxy status.
I0722 12:46:09.924835   29148 postcheck.go:127] [post-check] kube-proxy is OK.
I0722 12:46:09.924856   29148 postcheck.go:123] [post-check] Checking addon flannel status.
I0722 12:46:09.971854   29148 postcheck.go:127] [post-check] flannel is OK.
I0722 12:46:09.971876   29148 postcheck.go:123] [post-check] Checking addon coredns status.
I0722 12:46:10.020466   29148 postcheck.go:127] [post-check] coredns is OK.
I0722 12:46:10.020491   29148 postcheck.go:159] [post-check] Callback to the OpenAPI.
I0722 12:46:10.606890   29148 postcheck.go:172] [post-check] Callback to the OpenAPI successfully.

This node joined into the cluster successfully:  Please go to the console to see node detail information.
```

参数列表

参数	参数解释	默认值
flannelIface	flannel使用的网卡名	eth0
enableIptables	是否开启iptables	true
skipInstalled	表示已执行的步骤是否跳过	true
assumeYes	假设所有的问题回答自动回复yes	false
manageRuntime	是否由edgeadm安装并检测Runtime	true

参数	参数解释	默认值
nodeNameStrategy	节点名生成策略。目前支持四种策略： · hostname · random · randomWithPrefix · customized	hostname
nodeName	直接设置节点名。只有在`--nodename-strategy=customized`时生效	/
nodeNamePrefix	节点名称前缀；如果前缀为空，使用hostname；如果不为空prefix+UUID	/
enabledAddons	需要安装的组件列表；默认为空，不安装；普通节点需要配置为`["kube-proxy","flannel","coredns"]`	[]

5.2 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#unique_42。
- 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，单击左侧导航栏的集群 > 集群，进入集群列表页面。
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CPU Policy、登录密码和标签，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。

5.3 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#unique_37。

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
```

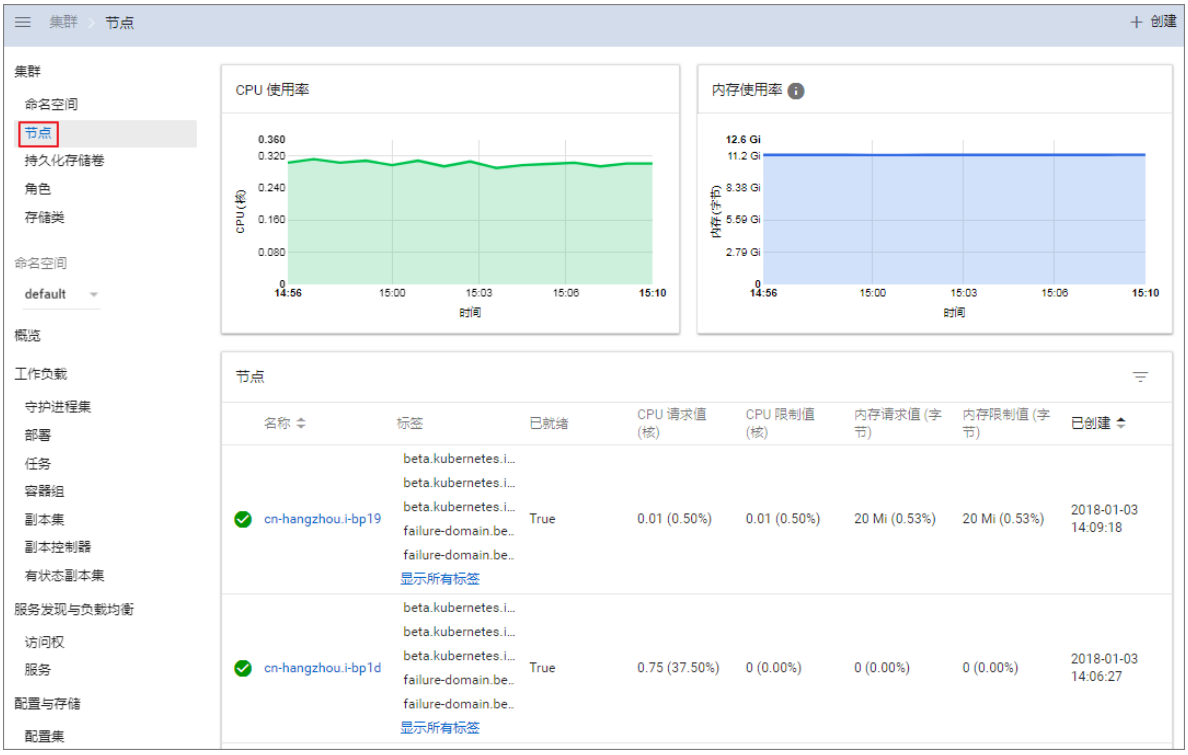
通过容器服务管理控制台查看

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。



5.4 监控节点

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。

后续步骤

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#unique_82。

Kubernetes 集群新增了关于应用分组的监控功能，您可以参见基础资源监控进行升级。

5.5 管理节点标签

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。

前提条件

您已经成功创建一个 Kubernetes 集群，请参见#unique_42。

背景信息

关于如何使用节点标签实现节点调度，请参见#unique_85。

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. 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。
2. 选择所需的集群，在页面右上角单击标签管理。
3. 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. 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。
2. 选择所需的集群，在页面右上角单击标签管理。

3. 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。

5.6 设置节点调度

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。

后续步骤

您在后续进行应用部署时，会发现Pod不会再调度到该节点。

5.7 设置节点自治

本文主要为您介绍如何在边缘节点中设置节点自治属性。

前提条件

- 您已经创建了一个Kubernetes边缘托管版集群，请参见创建 Kubernetes 边缘托管版集群。
- 您已经添加好一个边缘节点，请参见#unique_88。

背景信息

您可以通过Web界面设置节点自治属性，节点分为自治和非自治两种状态，当节点处于自治状态时，如果边缘节点和管控断连，管控将禁止节点操作以及避免节点上应用被驱逐，适用于边缘计算的弱网络连接场景。

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，单击右侧的更多 > 节点自治设置。

4. 在弹出的对话框中，单击确定。

说明:
本例中为开启节点的自治属性。您可以按照上述操作，关闭节点自治属性。

5.8 移除节点

对集群中的ECS实例进行重启或释放时，需要先从集群移除该ECS节点，本文为您介绍如何移除节点。

前提条件

- 您已成功创建一个Kubernetes集群，参见#unique_48。
- 您可以通过kubectl连接到Kubernetes集群，参见#unique_37。

背景信息

- 移除节点会涉及Pod迁移，可能会影响业务，请在业务低峰期操作。
- 操作过程中可能存在非预期风险，请提前做好相关的数据备份。
- 操作过程中，后台会把当前节点设置为不可调度状态。
- 移除节点仅移除Worker节点，不会移除Master节点。

操作步骤

1. 执行以下命令，把待移除节点上的Pod转移到其他节点。

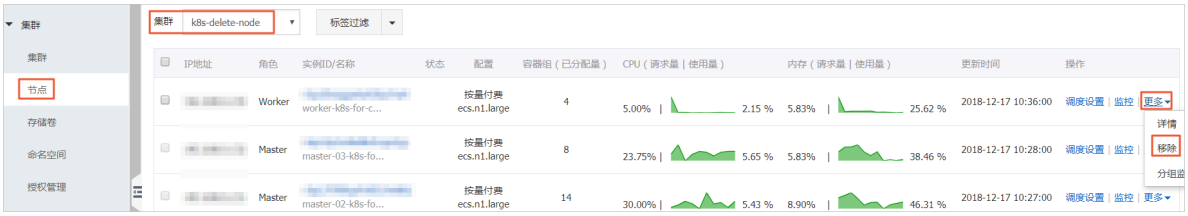
   说明:
   请确保集群其他节点的资源充足。

   ```
   kubectl drain node-name
   ```

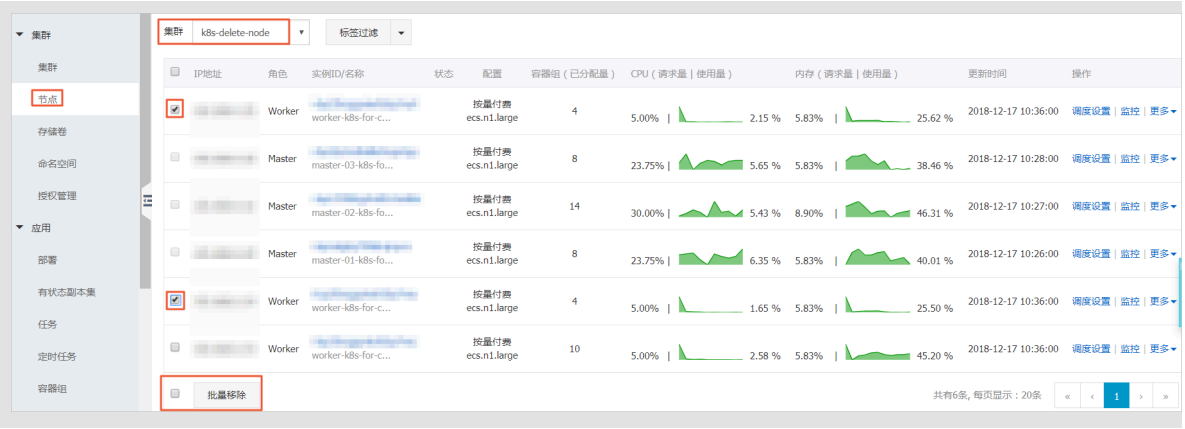
   说明:
   *node-name*格式为*your-region-name.node-id*。

   - *your-region-name*为您集群所在的地域名称。
   - *node-id*为待移除节点所在的ECS实例ID。如：cn-hanghzou.i-xxx。

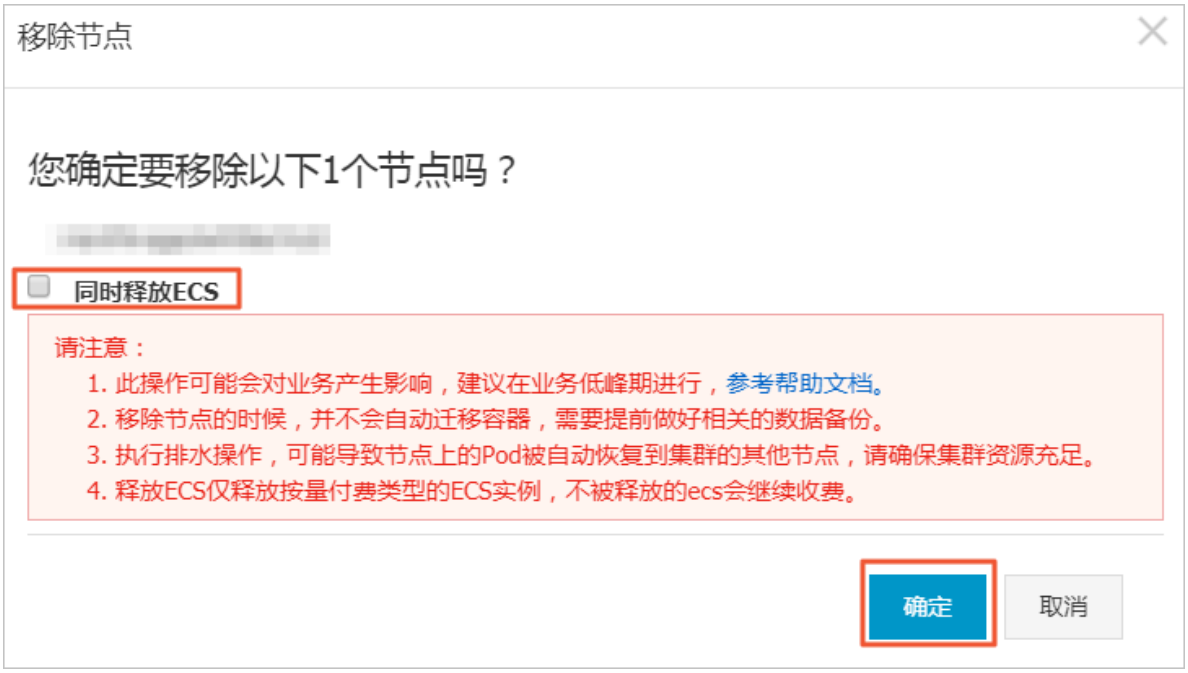
2. 在Kubernetes菜单下，单击左侧导航栏的集群 > 节点，进入节点列表页面。
3. 在目标集群下，选择目标节点，单击操作列更多 > 移除，进入移除节点页面。



   说明:
   若要同时移除多个节点，可在节点列表页面，目标集群下，同时勾选要移除的节点，单击批量移除。



4. （可选） 单击同时释放ECS，该节点所在的ECS实例将会被永久释放。



说明:

- 释放ECS实例仅释放按量付费的ECS实例。
- 对于预付费ECS实例，计费周期到期后，ECS实例会自动释放。
- 您也可以在ECS实例到期前：
  - 申请退款，提前释放实例，请参见退款规则及退款流程。
  - 将计费方式转为按量付费后释放实例，请参见#unique_90。
- 若不选择同时释放ECS，该节点所在的ECS实例会继续计费。

5. 单击确定，移除节点。

5.9 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#unique_48。

操作步骤

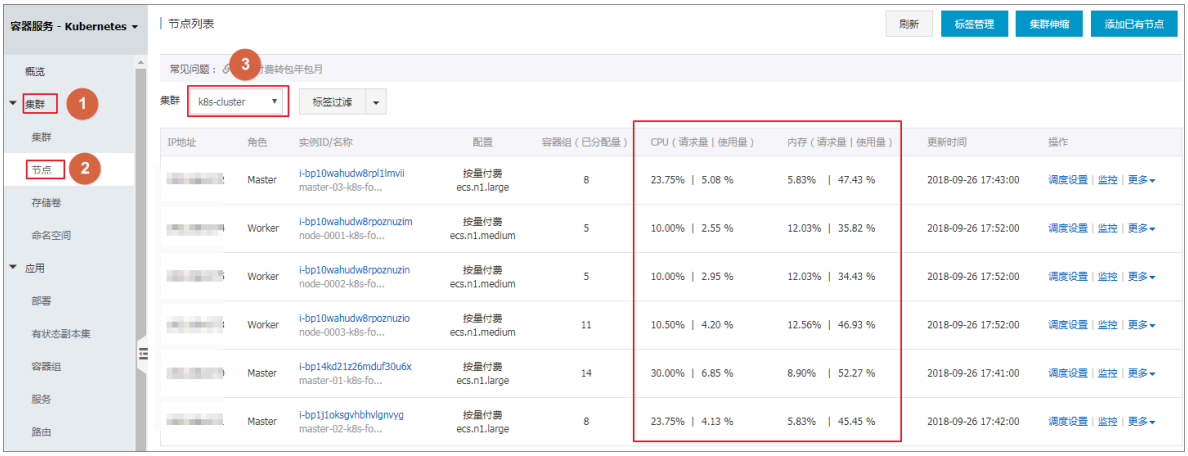
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 节点，进入节点列表页面。

   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#unique_92。
   - 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。



5.10 Kubernetes GPU集群支持GPU调度

本文主要为您介绍Kubernetes GPU集群可以支持GPU调度。

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。

背景信息

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 GPU，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。

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）。

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 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。

说明:
实例创建后，您可以通过 ECS 管理控制台将#unique_94。

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 > 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#unique_95。

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，单击操作列的更多 > 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. 在 Kubernetes 菜单下，单击左侧导航栏的应用 > 无状态，进入无状态（Deployment）页面。
2. 单击页面右上角的创建使用模板创建 。
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。

4. 单击左侧导航栏中的路由与负载均衡 > 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。

5. 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。

6. 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())
```

5.11 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#unique_54。
- 您已连接到Master节点，方便快速查看节点标签等信息，参见#unique_37。

背景信息

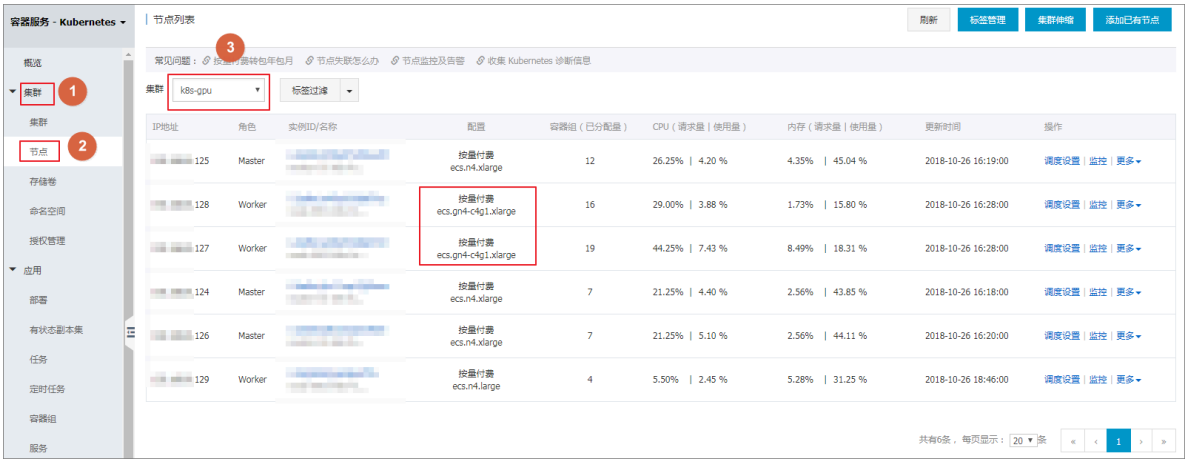
阿里云Kubernetes在部署Nvidia GPU节点的时候会发现GPU的属性，并且作为NodeLabel信息暴露给用户，拥有如下优势：

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

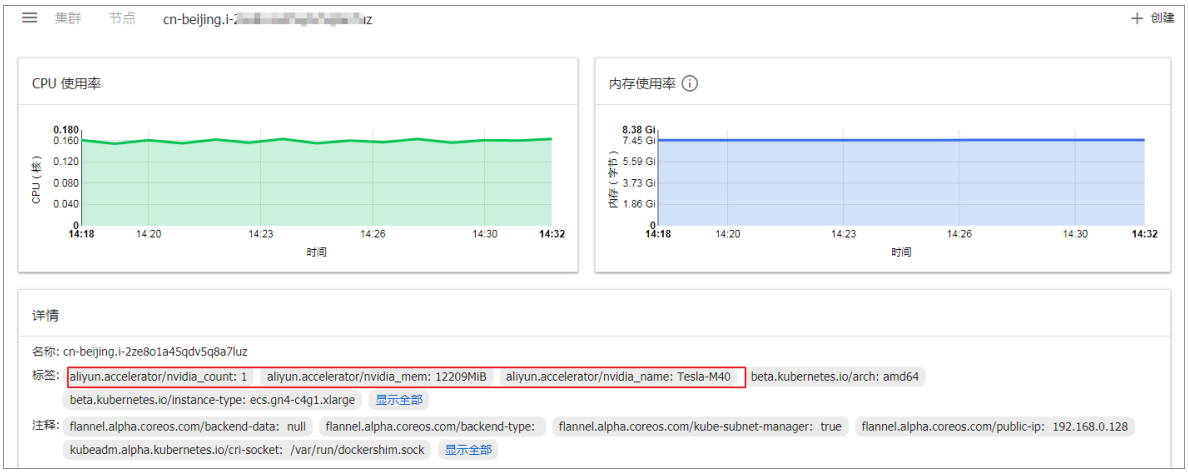
操作步骤

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）。

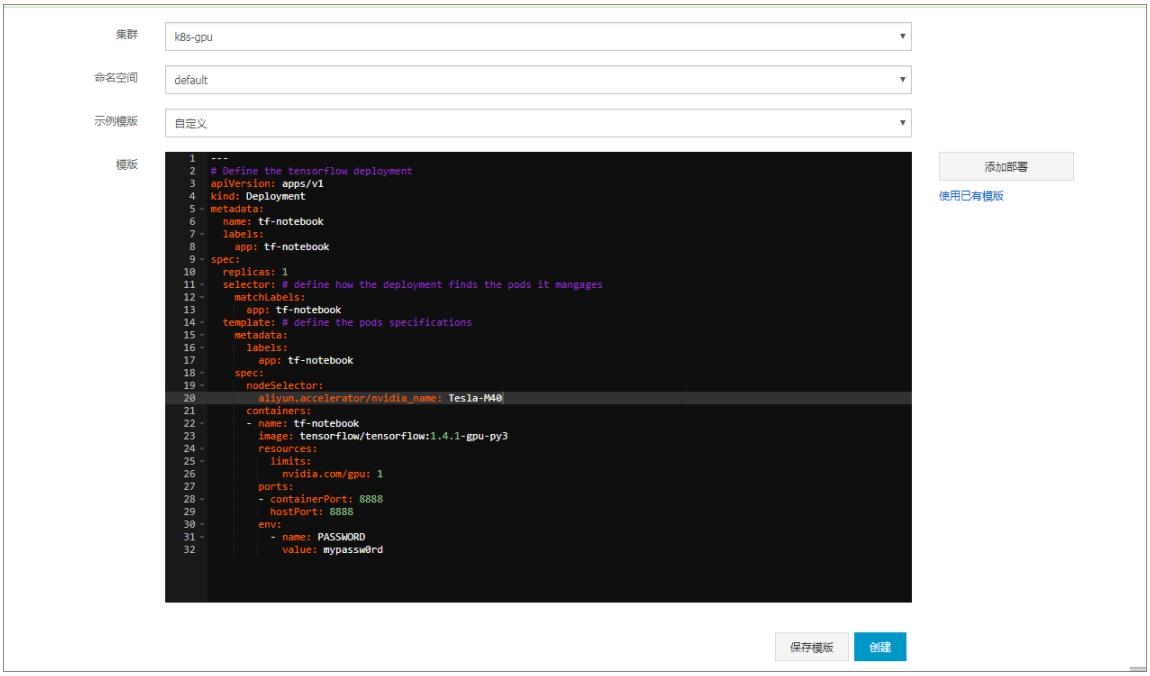
key	value
aliyun.accelerator/nvidia_count	GPU核心数量
aliyun.accelerator/nvidia_mem	GPU显存，单位为MiB
aliyun.accelerator/nvidia_name	nvida设备的GPU计算卡名称

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 > 无状态，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
```

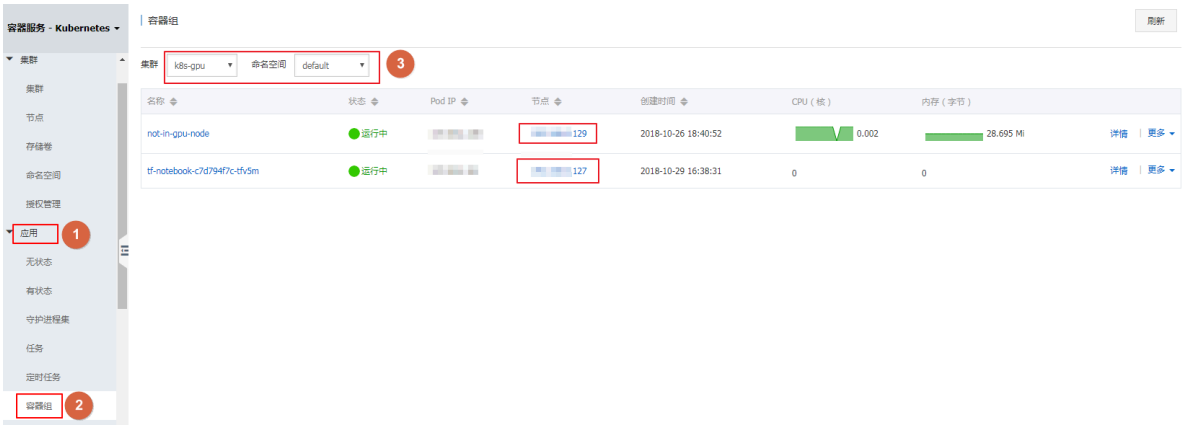
b) 您也可避免将某些应用部署到GPU节点。下面部署一个nginx的Pod，利用节点亲和性的特性进行调度，具体参见#unique_97中关于节点亲和性的说明。

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。

5.12 为容器服务的Docker增加数据盘

本文介绍如何为Docker 数据目录挂载数据盘。挂载数据盘可以扩容Docker数据目录，从而当在机器上运行的容器或者镜像数量不断增加时，保证有足够的磁盘空间可以满足使用需求。

Docker 数据目录

Docker的数据通过联合文件系统的方式存储到磁盘上。Docker默认的容器和镜像数据存储在*/var/lib/docker*目录下。您可以通过du命令查看这个目录目前占用的磁盘大小。

```
# du -h --max-depth=0 /var/lib/docker
7.9G    /var/lib/docker
```

更换Docker的数据盘

很多Docker镜像较大，因此可能几个镜像就会占用大量磁盘空间。较大的镜像或较多的容器，都会导致磁盘空间的不足。为了满足您增加镜像或容器的需求，您需要为Docker的数据目录增加数据盘。

增加数据盘

完成以下步骤，为Docker的数据目录增加数据盘：

1. 创建ECS数据盘，并挂载到需要扩容的机器上。
   a) 通过云服务器 ECS 控制台创建需要配置的云盘。
   b) 通过云服务器 ECS 控制台创建需要配置的云盘。
   c) 单击左侧导航栏中的实例。
   d) 单击目标ECS实例ID，进入实例详情页。
   e) 单击左侧导航栏中的本实例磁盘。
   f) 单击右上角的挂载云盘。
   g) 在弹出的对话框中，选择创建的磁盘作为目标磁盘。单击确定。
   h) 单击执行挂载，挂载新磁盘到目标ECS实例，并记录挂载点/dev/xvd*或者/dev/vd*。
2. 登录ECS实例，对刚才挂载的磁盘进行格式化。
   a) 执行ls -l /dev/xvd*或者ls -l /dev/vd*命令，验证是否和上述步骤中记录的挂载点一致。
   b) 通过fdisk命令对磁盘进行分区，然后使用mkfs.ext4命令格式化磁盘。

3. 移动Docker数据到新的磁盘。

   如果不希望中断节点上正在运行的应用，您需要对应用进行迁移。迁移swarm集群，请参见指定多节点调度。迁移Kubernetes集群，请参见Safely Drain a Node while Respecting Application SLOs。
   a) 停止Docker Daemon和kubelet，保证迁移时的数据完整。可以使用service kubelet stop和service docker stop 命令进行停止。
   b) 移动Docker的目录到一个备份的目录。例如：mv /var/lib/docker /var/lib/docker_data。
   c) 然后把新的格式化好的磁盘挂载到*/var/lib/docker*和*/var/lib/kubelet*目录。例如：

```
echo "/dev/xvdb1    /var/lib/container/     ext4    defaults
  0 0" >>/etc/fstab
echo "/var/lib/container/kubelet /var/lib/kubelet none defaults,
bind 0 0" >>/etc/fstab
echo "/var/lib/container/docker /var/lib/docker none defaults,bind
 0 0" >>/etc/fstab


mkdir /var/lib/docker

mount -a
```

   d) 把之前备份的Docker数据移动到新的磁盘上。例如：mv /var/lib/docker_data/* /var/lib/docker/。

4. 启动Docker Daemon和kubelet，并检查数据位置。

   a) 启动Docker Daemon和kubelet，命令分别是`service docker start`和`service kubelet start`。

   b) 执行`df`命令，可看到*/var/lib/docker*挂载到了新的磁盘上。如果需要启动Kubernetes集群，请跳过此步骤。

   c) 执行`docker ps`命令，查看容器是否丢失。根据需要，重启相关容器。如没有设置`restart:always`标签的容器。

5. 您可以通过调度的方式使被迁移走的容器回归到这个节点上来。

   更多容器服务的相关内容，请参见容器服务。

5.13 集群节点挂载数据盘

当需要在机器上运行的容器或者镜像数量不断增加时，磁盘的大小可能不再满足需求，您需要通过增加数据盘的方式对Docker的数据目录进行扩容。

前提条件

如需挂载数据盘，请确保您的集群是1.10.4以后版本。

挂载数据盘

您通常可以选择以下两种方案对集群已有节点进行数据盘扩容：

- 如果已有节点之前没有挂载盘，请参考为容器服务的Docker增加数据盘。
- 如果已有节点购买了数据盘，但是未能成功挂载，一般可以通过以下步骤进行挂载。

> **说明:**
>
> - 为降低操作过程的风险，建议操作前可以对节点做快照或者进行数据备份。
> - 需要评估集群的应用是否支持被调度到其他节点。
> - 请选择在业务低峰期间操作。
> - 节点排水会导致节点上的Pod被调度到其他节点，请确保集群有多余节点资源。如果资源不足，请考虑提前临时扩容。

在执行操作之前，您可以在worker节点上执行`df`命令，通过查看命令执行结果中`/var/lib/docker`是否成功挂载到`/dev/vdb1`来判断数据盘是否成功挂载。若挂载成功，则无需处理。若未成功挂载，您需要按照以下步骤完成挂载。

1. 设置节点为不可调度。

   详细操作请参考Mark node as unschedulable。

2. 节点排水。

   详细操作请参考Safely-Drain-Node。

3. 移除该节点。

   本文中仅就容器服务控制台中的操作进行说明。

   a) 登录容器服务管理控制台。

   b) 登录容器服务管理控制台。

   c) 单击左侧导航栏中的集群 > 节点。

   d) 选择要移除的节点，单击批量移除或者选择更多 > 移除。

   e) 在弹出的移除节点对话框中，单击确定。

   > 说明:
   > 不要选择同时释放ECS复选框。

4. 重新添加刚才移除的节点。

   a) 单击左侧导航栏中的集群 > 集群。

   b) 选择要添加节点的集群。单击更多 > 添加已有节点。

   c) 选择自动添加或手动添加节点。本文中以自动添加为例。

   d) 选择已有云服务器并单击下一步。

   e) 选择在添加节点过程中格式化数据盘。

   f) 按照界面提示完成其余操作。

成功添加节点后，您可以登录该节点并执行`df`命令查看数据盘的挂载情况。

如果数据盘挂载成功，如下图所示。

通过以上两种方案，您可以为已有节点添加或者重新挂载数据盘。

6 应用管理

6.1 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_95。

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_104
- 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。

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，最小为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。

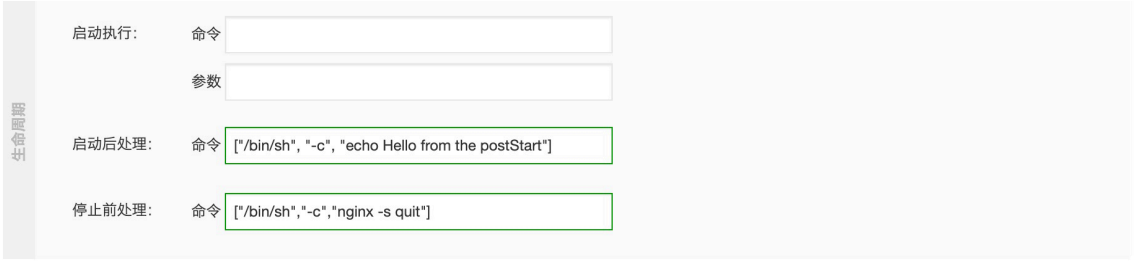
请求类型	配置说明
TCP连接	即向容器发送一个TCP Socket，kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： · 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 · 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 · 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10s，最小为1s。 · 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 · 健康阈值：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

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_105。
· 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_106。

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。

6.2 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_48。
- 您已成功创建一个云盘存储卷声明，参见#unique_108。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

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_109
- 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/。

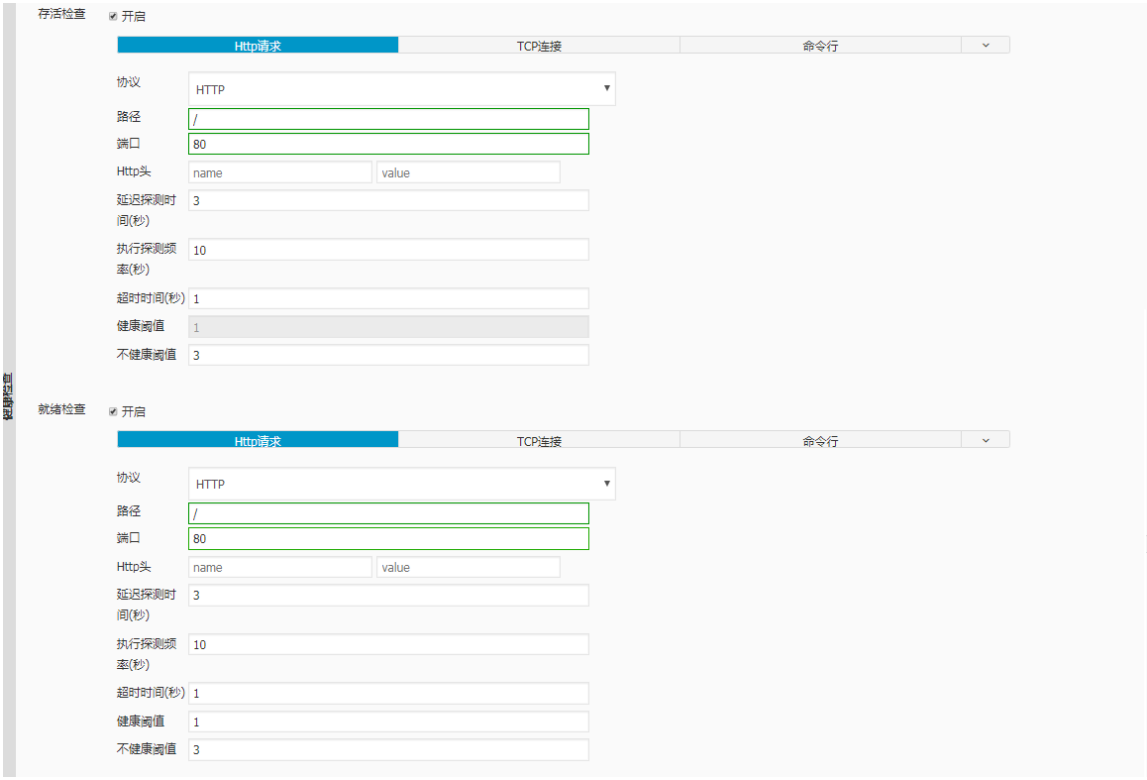
b) （可选）配置环境变量。

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。 · 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。

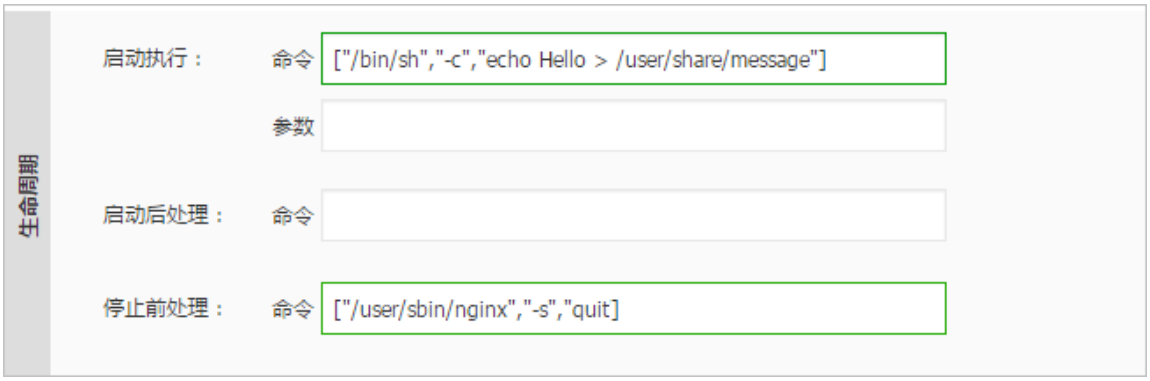
请求类型	配置说明
TCP连接	即向容器发送一个TCP Socket，kubelet将尝试在指定端口上打开容器的套接字。 如果可以建立连接，容器被认为是健康的，如果不能就认为是失败的。支持的参数包括： · 端口：容器暴露的访问端口或端口名，端口号必须介于1~65535。 · 延迟探测时间（秒）：即initialDelaySeconds，容器启动后第一次执行探测时需要等待多少秒，默认为15秒。 · 执行探测频率（秒）：即periodSeconds，指执行探测的时间间隔，默认为10s，最低为1s。 · 超时时间（秒）：即timeoutSeconds，探测超时时间。默认1秒，最小1秒。 · 健康阈值：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

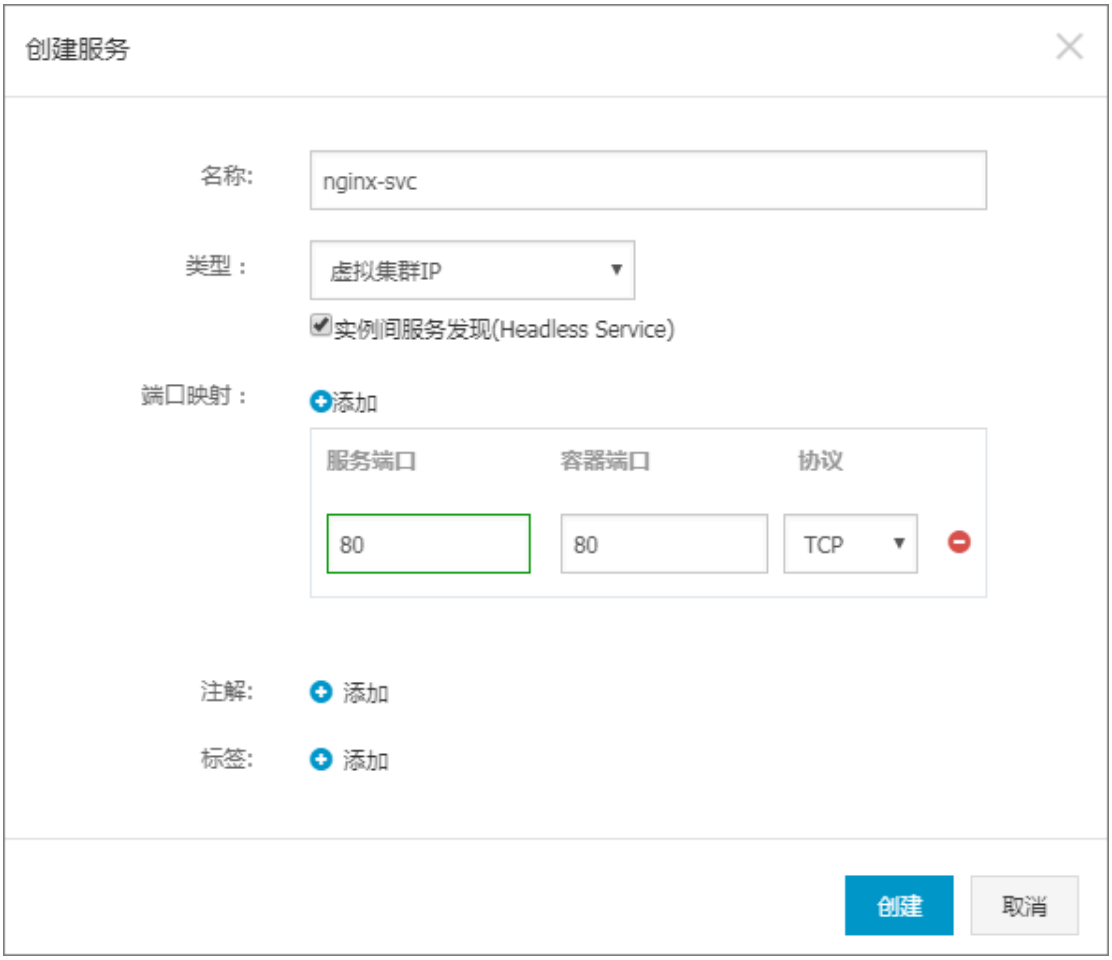
说明:

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/。

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_110。

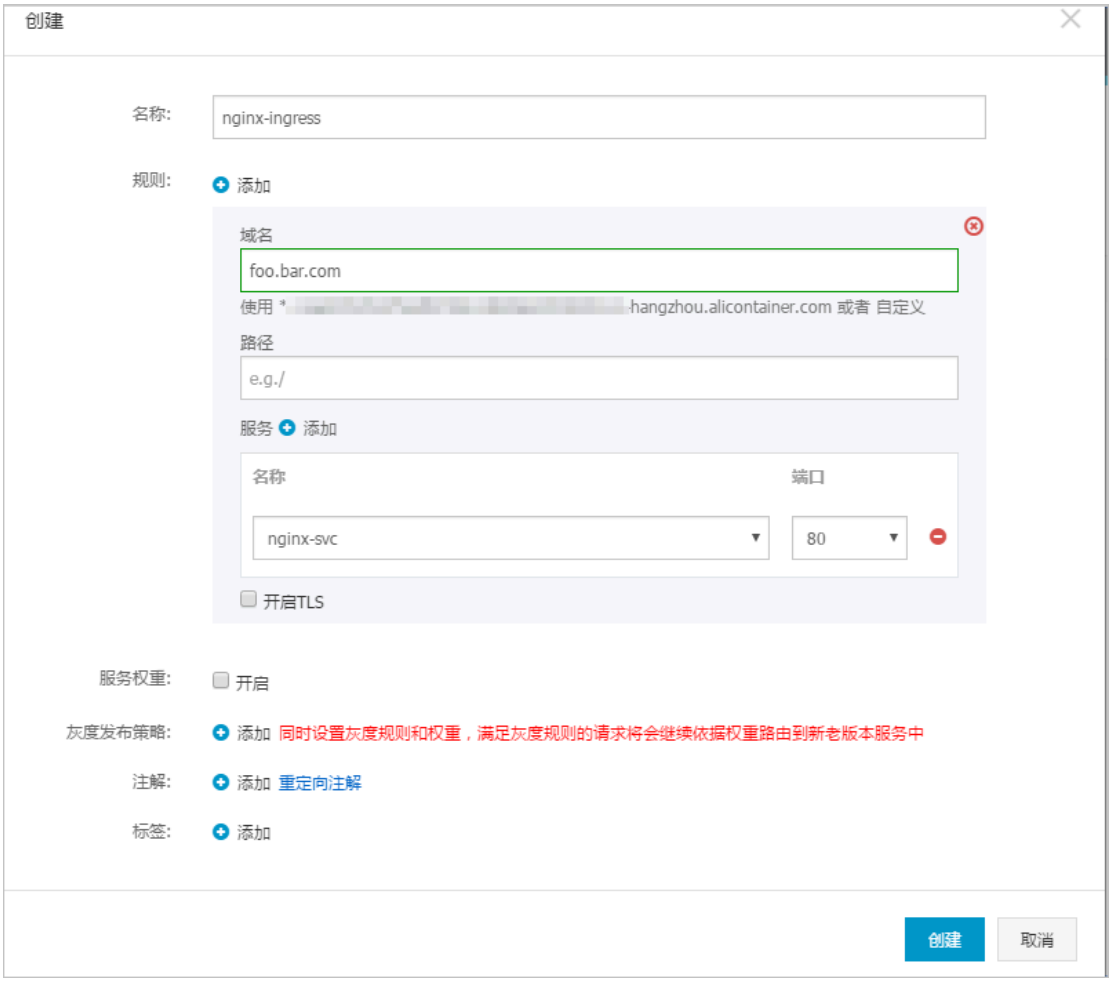
· 标签：您可为该服务添加一个标签，标识该服务。

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_111。

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。

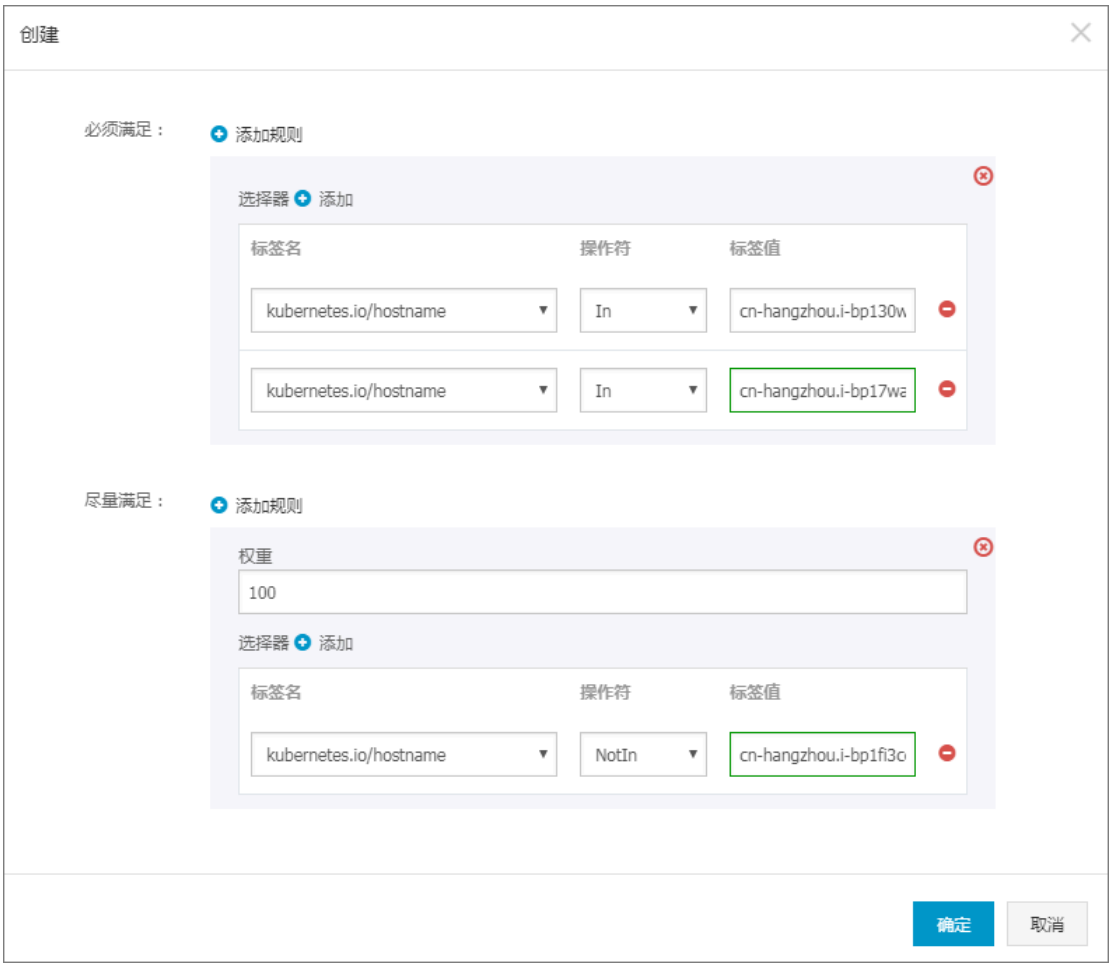
说明:

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。

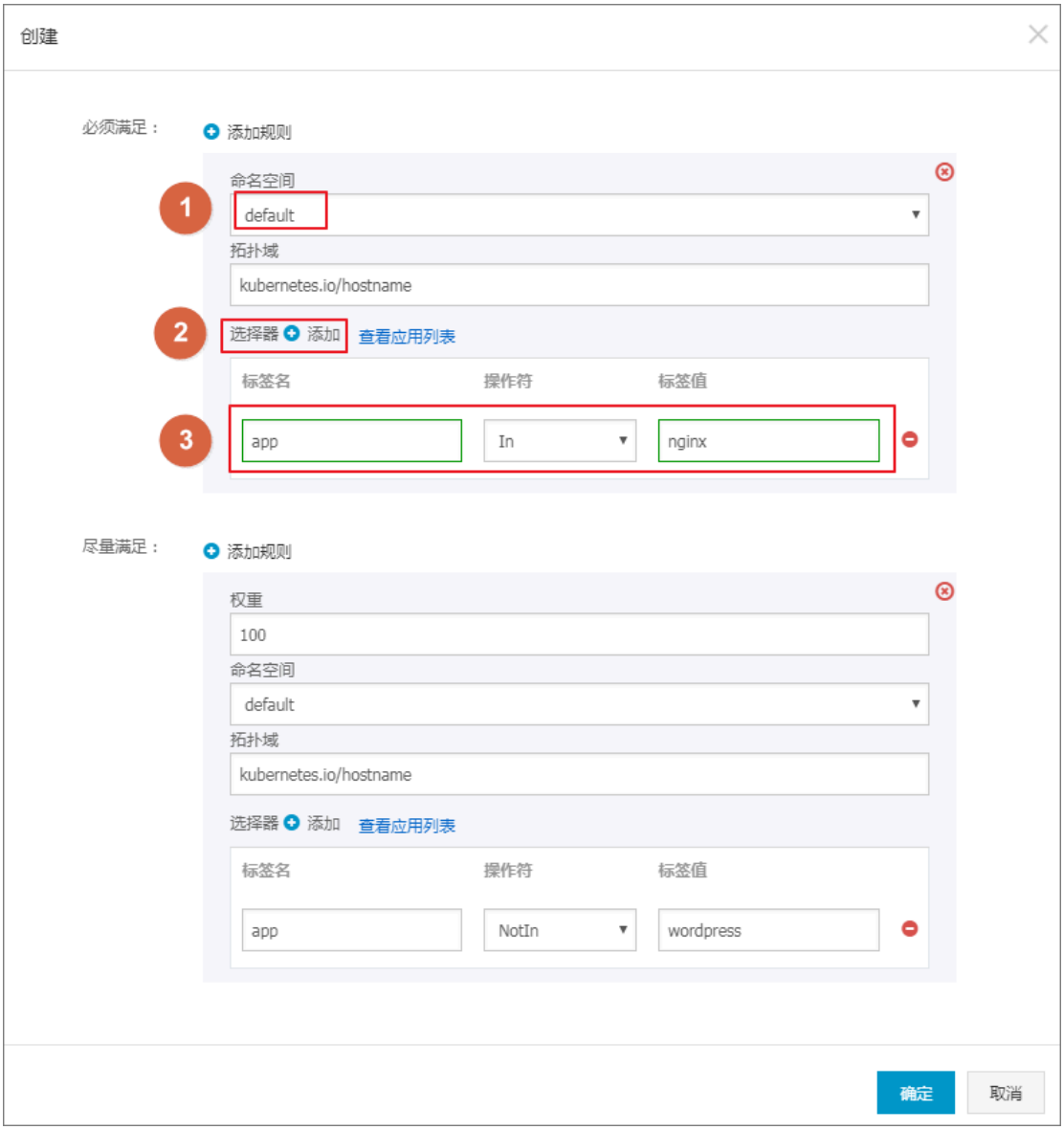


节点调度支持硬约束和软约束（Required/Preferred），以及丰富的匹配表达式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- 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

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。

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调度到权重最高的节点上。

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：

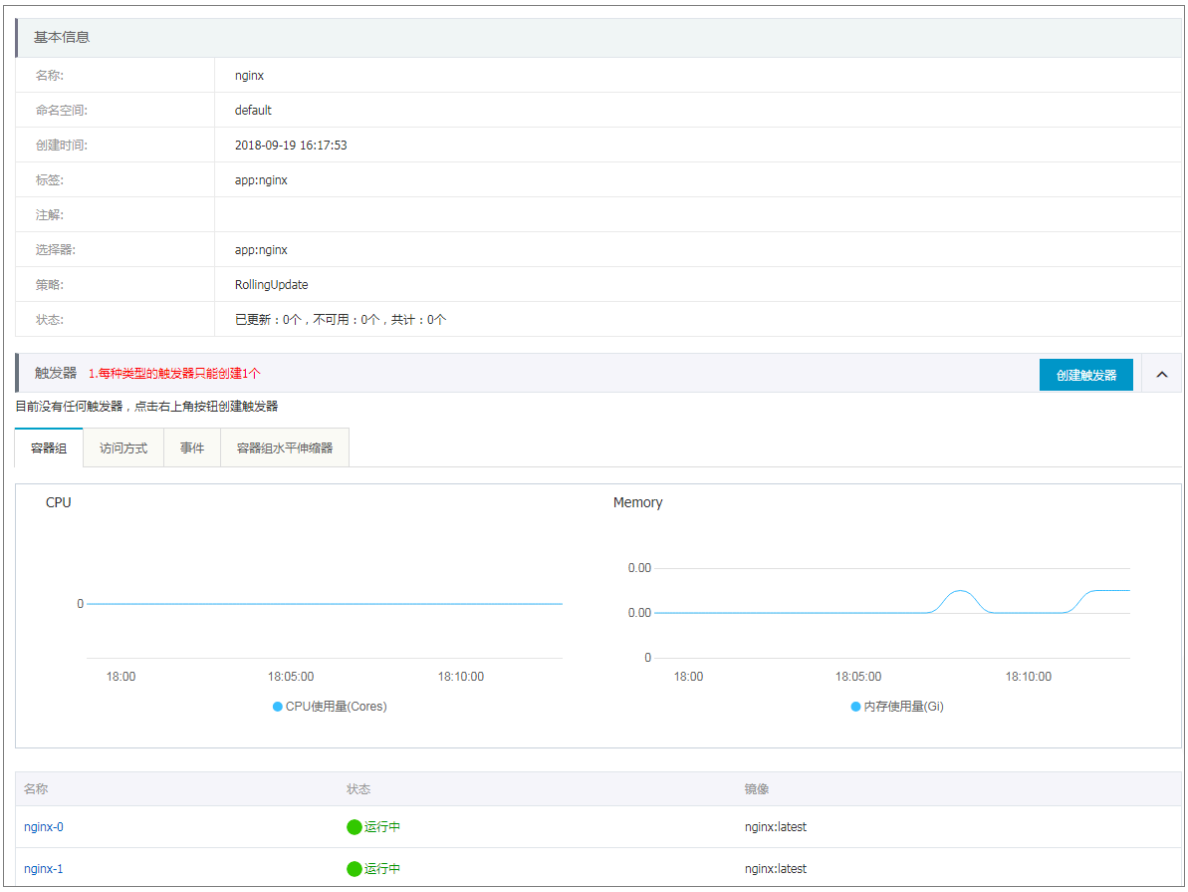
· 将一个服务的Pod分散部署到不同的拓扑域（如不同主机）中，提高服务本身的稳定性。
· 给予Pod一个节点的独占访问权限来保证资源隔离，保证不会有其它Pod来分享节点资源。
· 把可能会相互影响的服务的Pod分散在不同的主机上。

> 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。

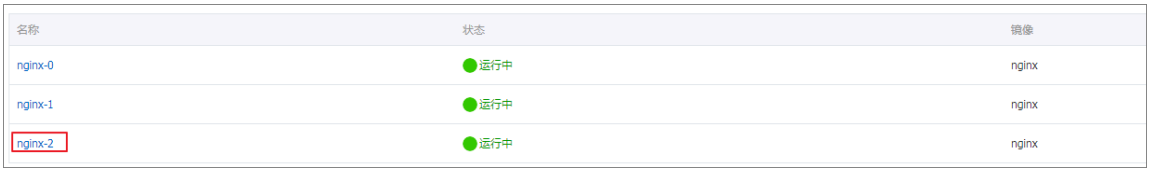
默认进入有状态副本集详情页面。



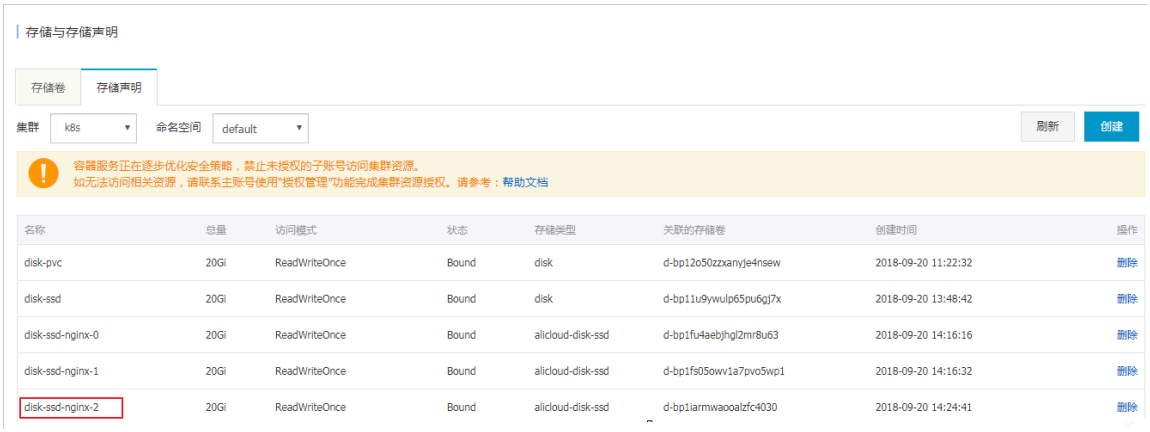
9. 然后单击左上角返回列表，进入有状态副本集列表页面，查看创建的StatefulSet应用。

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。

6.3 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#unique_48。

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

Job类型	使用示例	行为	completions	Parallelism
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completions个成功结束	2+	1
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。

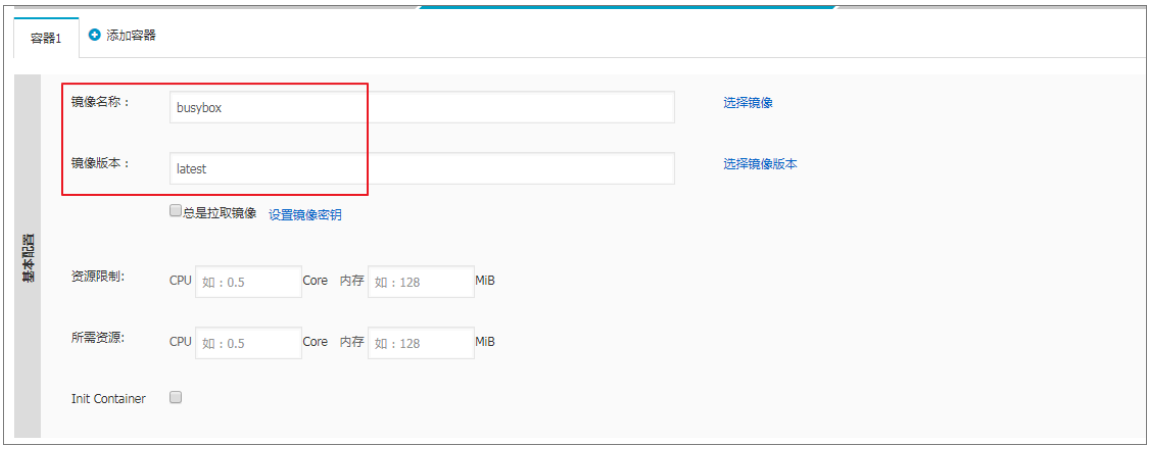
- **镜像名称：您可以单击选择镜像，在弹出的对话框中选择所需的镜像并单击确定，本例中为 busybox。**

  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#unique_113。
- 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/。



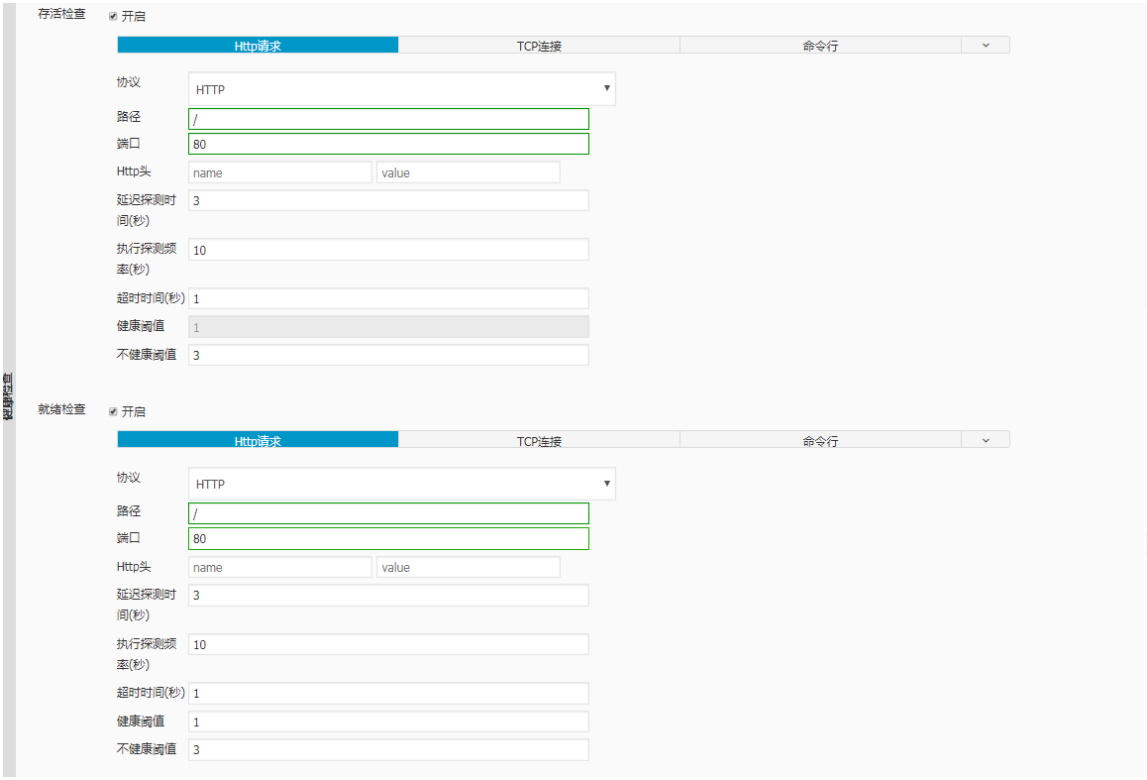
b) （可选）配置环境变量。

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。 · 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。

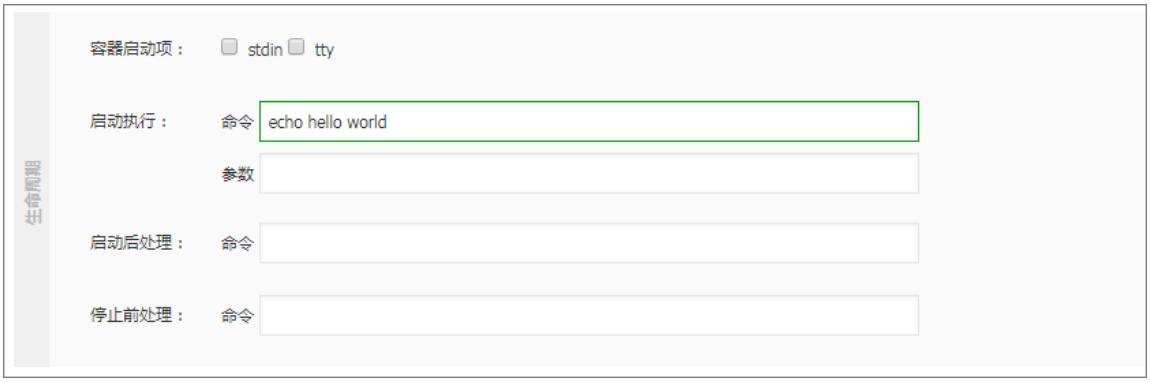
d) （可选）配置生命周期。

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。

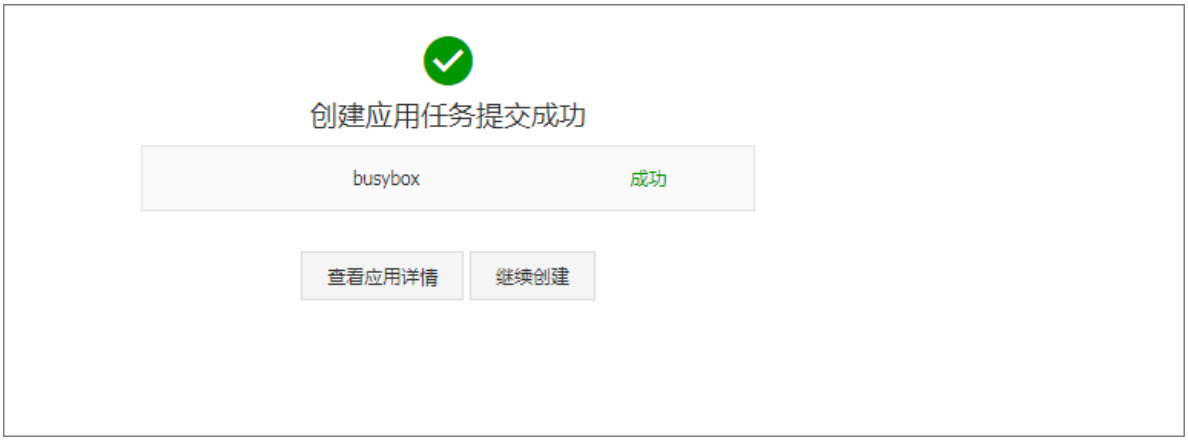
6. 进行高级设置。

   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
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）

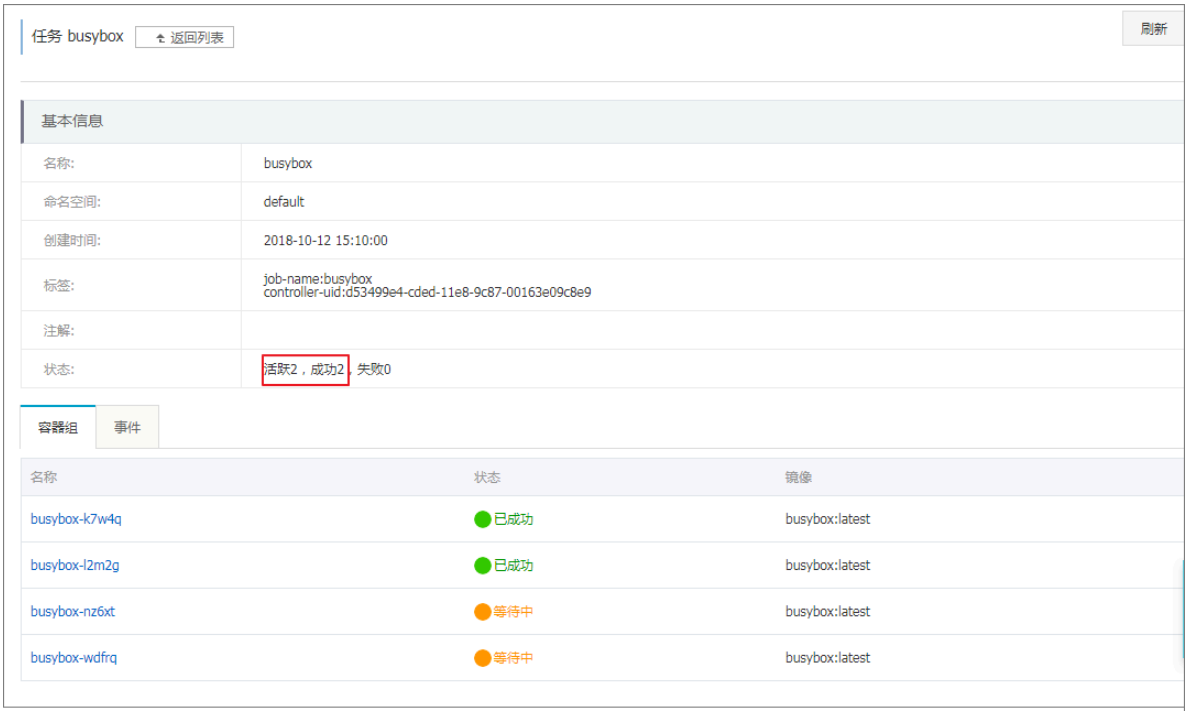
7. 最后单击创建。

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。

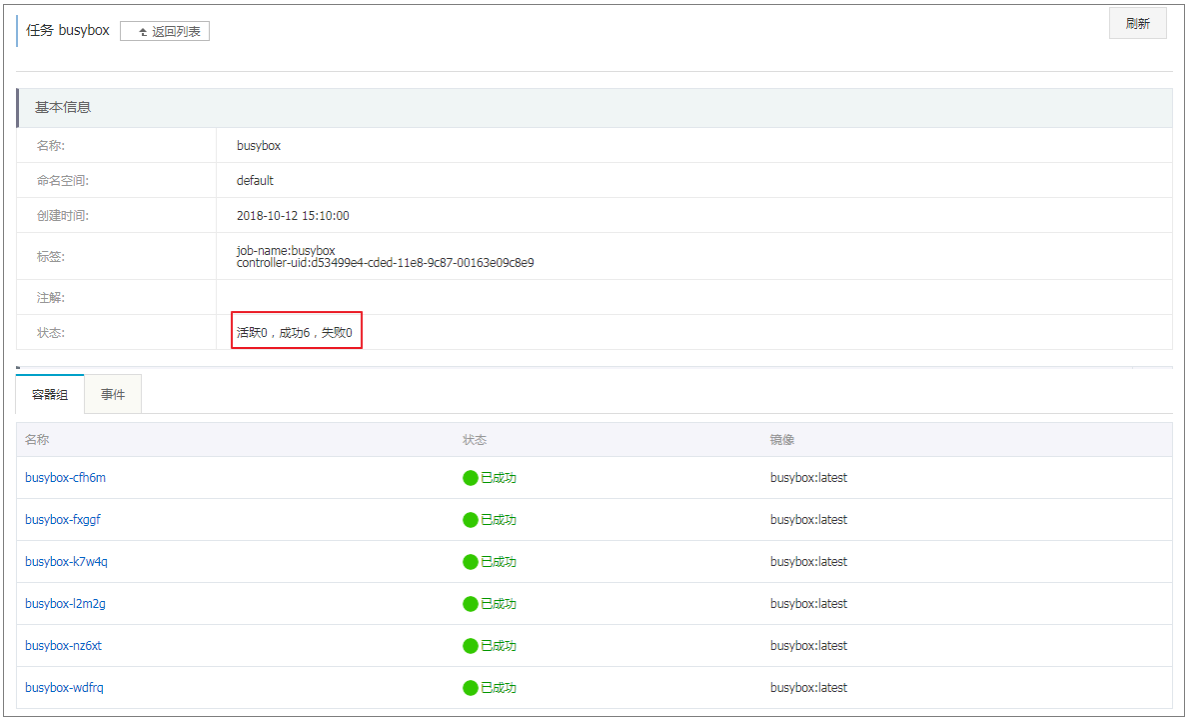


您可以单击查看应用详情，进入任务详情页面。

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。

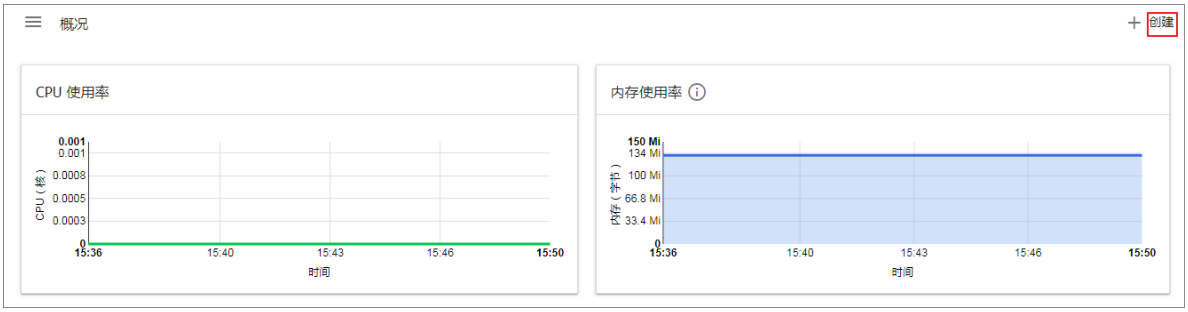
6.4 通过 Kubernetes Dashboard 创建应用

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。

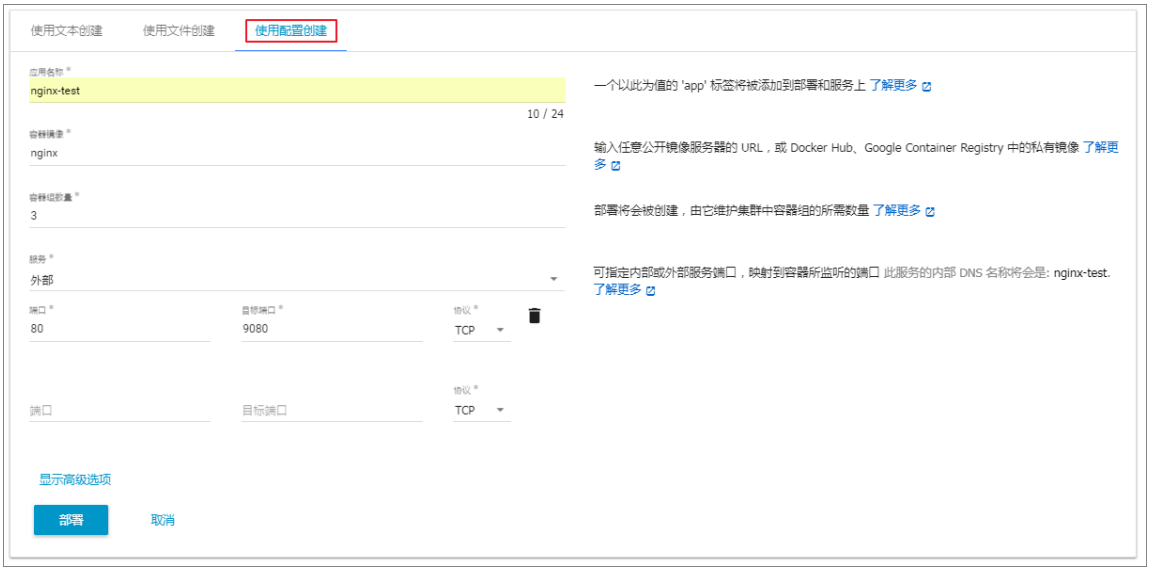
   您可以通过以下 3 种方法之一创建应用：

   · 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。



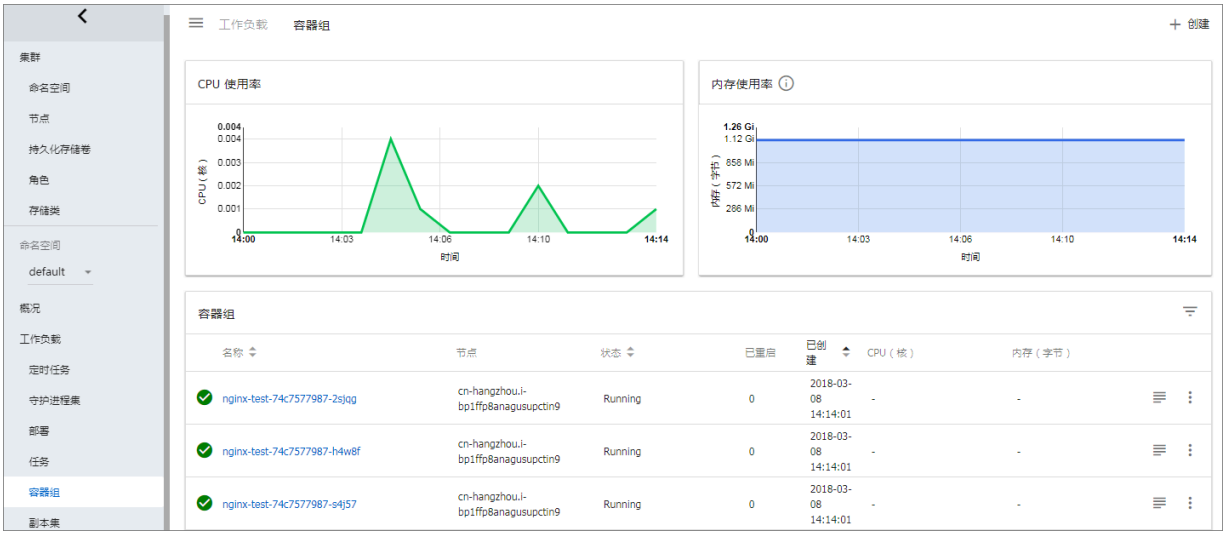
6. 单击部署 部署这些容器和服务。

   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。

后续步骤

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。



6.5 通过编排模板创建Linux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

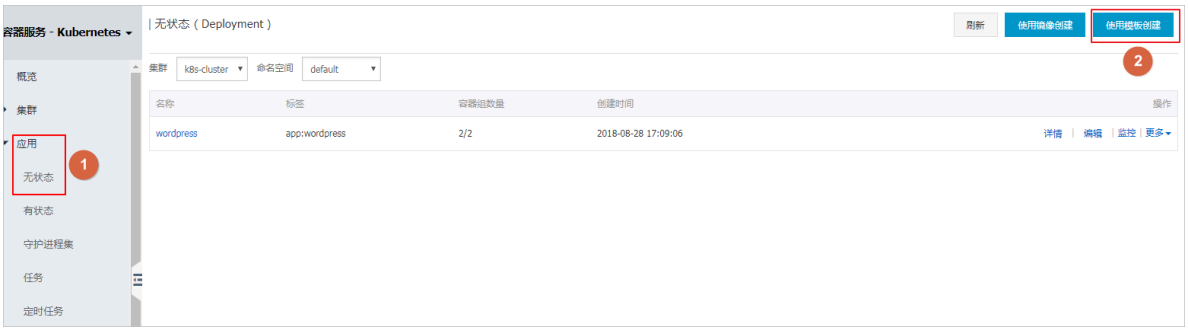
创建一个 kubernetes 集群，参见#unique_48。

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。

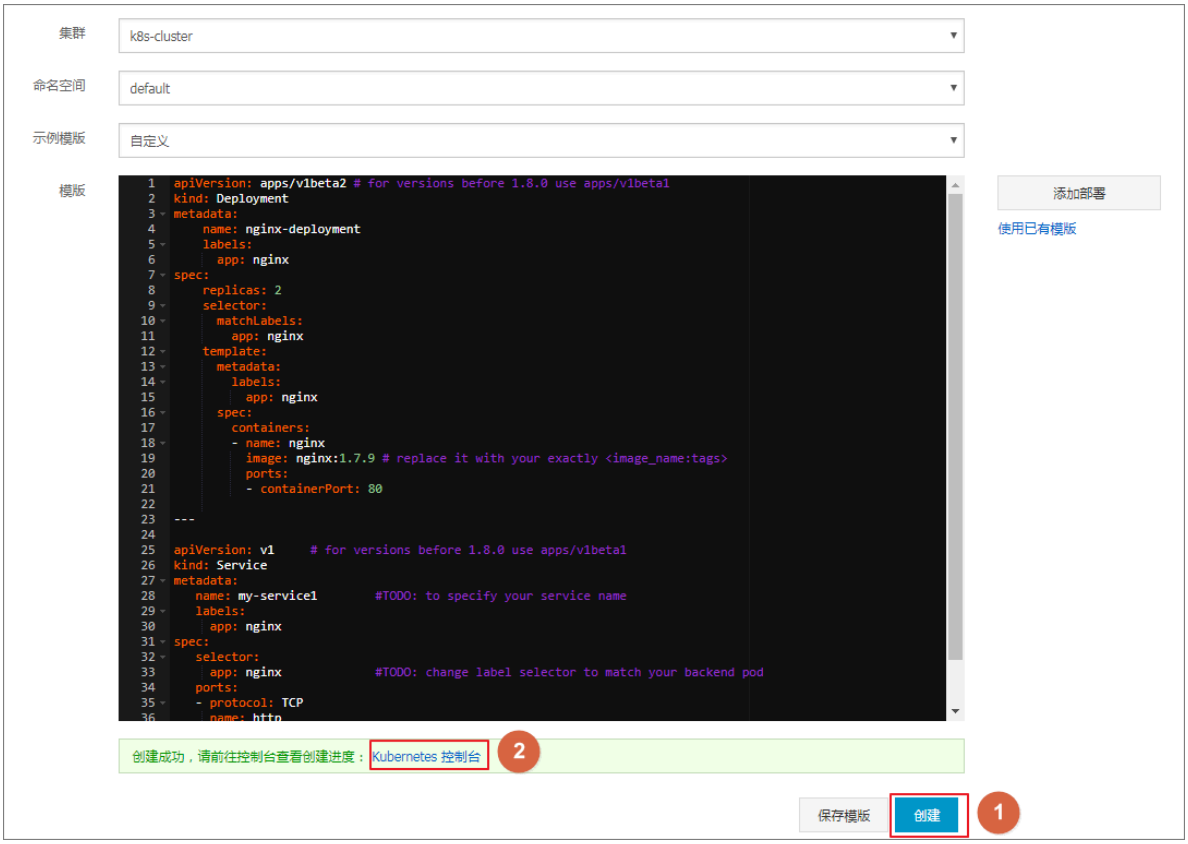
操作步骤

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 > 无状态，进入无状态（Deployment）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。



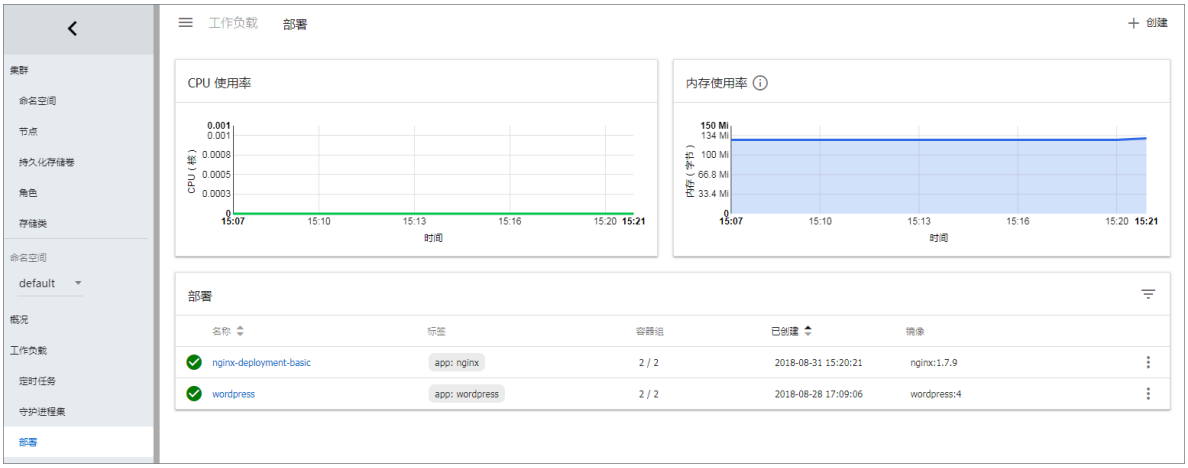
下面是一个 nginx 应用的示例编排，基于容器服务内置的编排模板。通过该编排模板，即可快速创建一个属于 nginx 应用的 deployment 。

> **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。

后续步骤

您也可返回容器服务首页，单击左侧导航栏中的路由与负载均衡 > 服务，查看该nginx的服务。

6.6 通过编排模板创建Windows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 Windows Kubernetes 集群，参见创建Windows Kubernetes 集群。

背景信息

本例演示如何通过一个编排模板创建 aspnet 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 aspnet 应用。

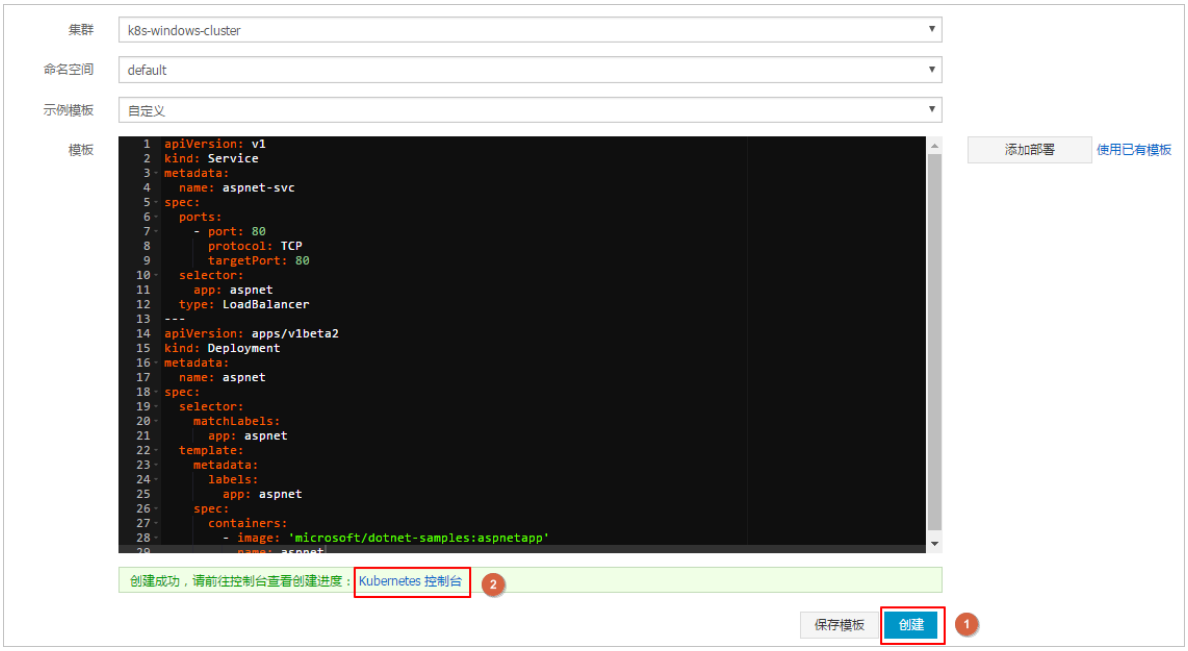
操作步骤

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 > 无状态，进入无状态（Deployment）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aspnet 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 aspnet 应用的 deployment 。

说明:

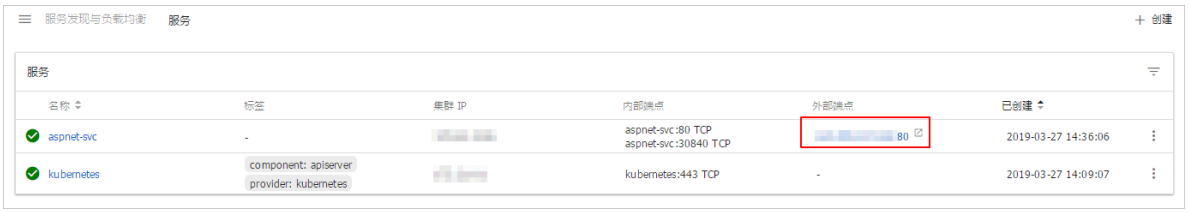
容器服务支持Kubernetes Yaml编排，支持通过---符号将资源对象分隔，从而通过一个模板创建多个资源对象。

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

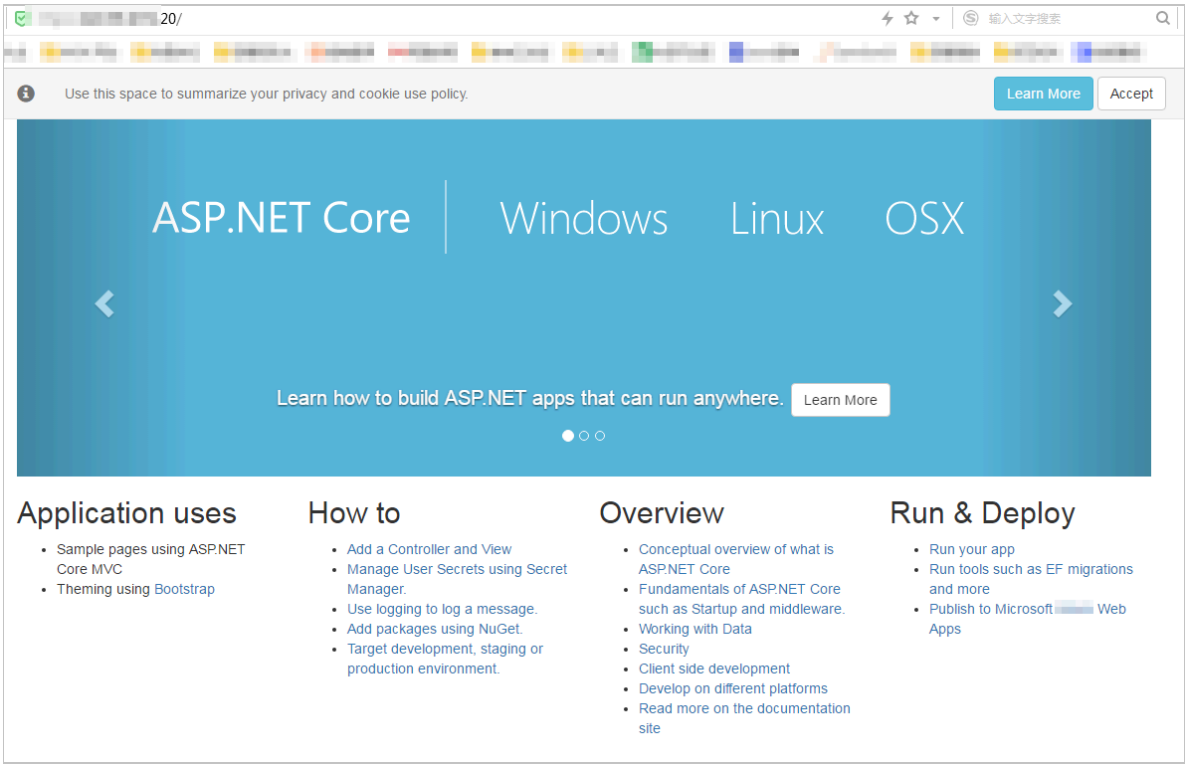
```
metadata:
  name: aspnet-svc
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: aspnet
  type: LoadBalancer
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: aspnet
spec:
  selector:
    matchLabels:
      app: aspnet
  template:
    metadata:
      labels:
        app: aspnet
    spec:
      containers:
        - image: 'microsoft/dotnet-samples:aspnetapp'
          name: aspnet
```

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 aspnet-svc服务已成功部署，并暴露了外部入口。单击外部端点的访问地址。



7. 您可以在浏览器中访问 aspnet 首页。



后续步骤

您也可返回容器服务首页，单击左侧导航栏中的路由与负载均衡 > 服务，查看该aspnet的服务。

6.7 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#unique_37。

通过命令创建应用

1. 执行如下命令启动容器（本示例中为 Nginx Web 服务器）。

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

2. 执行如下命令，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`kubectl get pods`命令，列出所有 default 命名空间里正在运行的容器。

```
NAME                                READY     STATUS    RESTARTS
AGE
nginx-2721357637-dvwq3              1/1       Running   1
9h
```

6.8 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#unique_48。

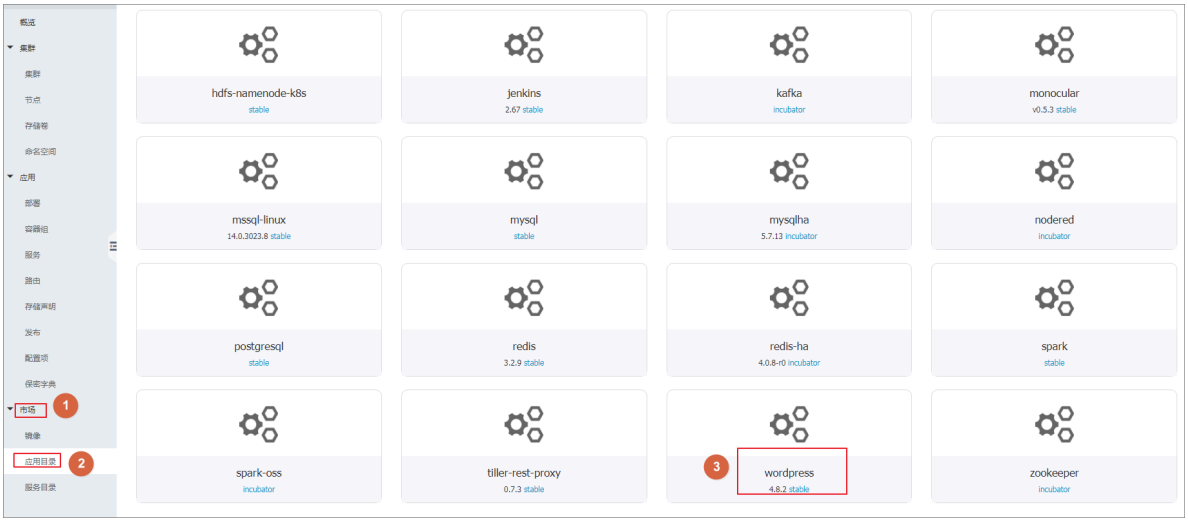
  在 Kubernetes 集群创建的同时，Tiller 将会被自动部署到集群之中，并且在所有的 master 节点上自动安装 Helm CLI 以及配置指向阿里云的 Chart 存储库。
- 查看您集群中 Kubernetes 的版本。

  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

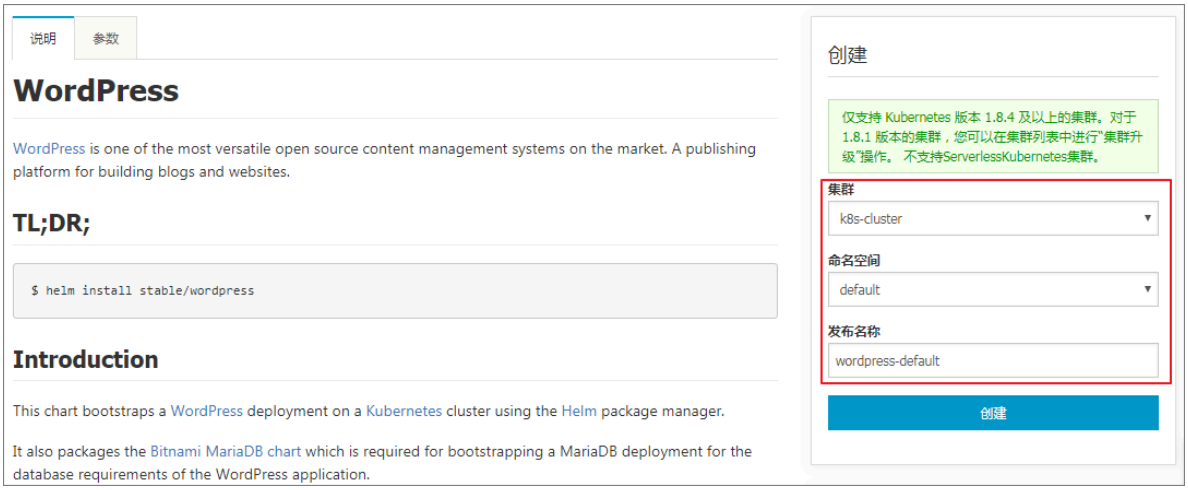
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的市场 > 应用目录，进入应用目录列表页面。

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。

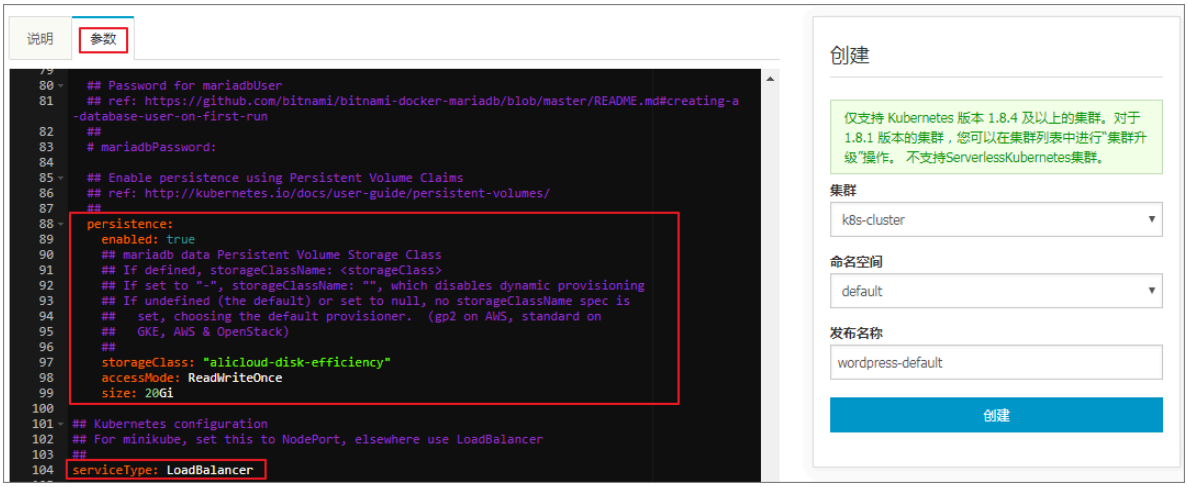


5. 单击参数，对配置进行修改。

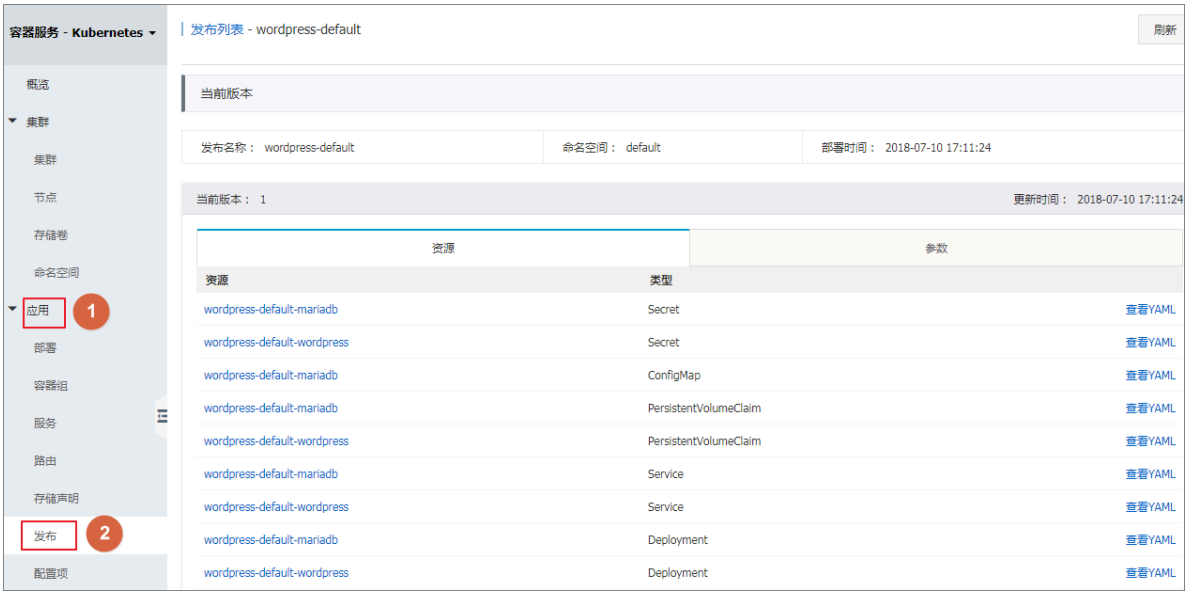
   本示例中使用云盘的动态数据卷绑定一个PVC，参见#unique_119。

说明:

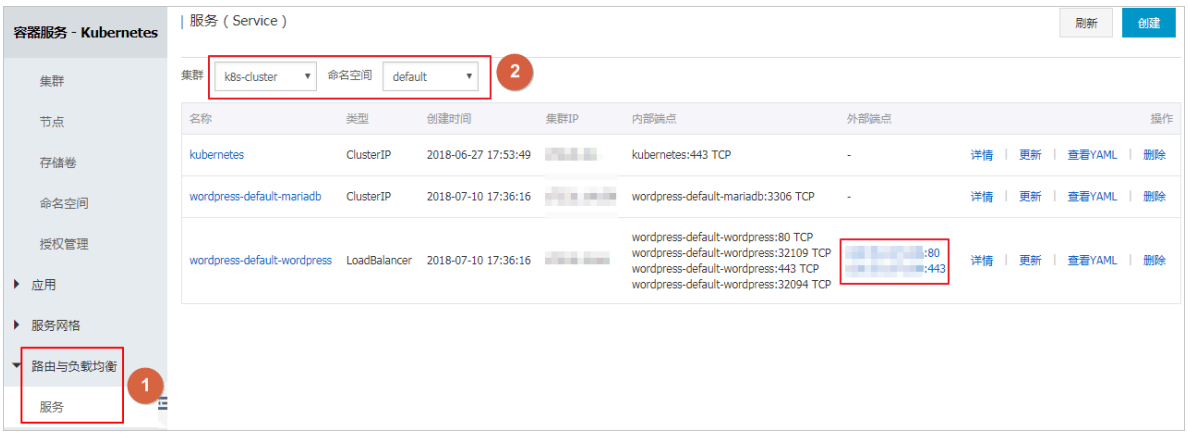
您需要预先创建一个云盘存储卷（PV），并且存储卷的容量不能小于PVC定义的数值。



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路由与负载均衡 > 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#unique_29 。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#unique_37。

   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。

6.9 使用应用触发器重新部署应用

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#unique_48。
- 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 > 无状态，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。

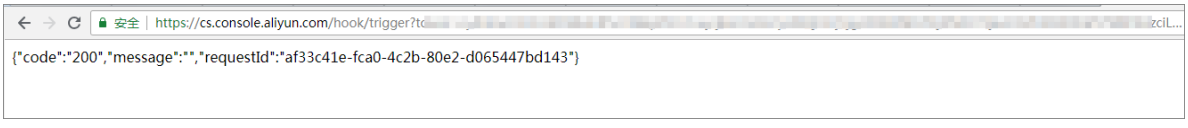
4. 在弹出的对话框中，选择重新部署触发器，然后单击确定。

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。

后续步骤

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
```

6.10 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。

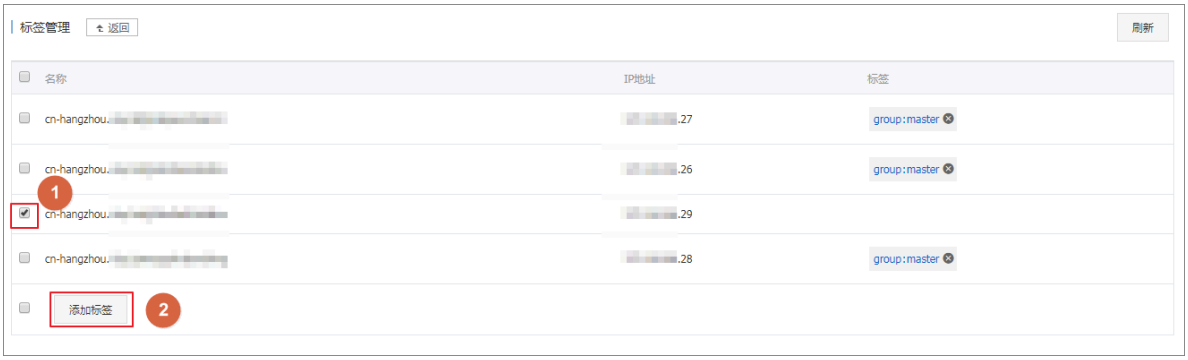
前提条件

您已经成功部署一个 Kubernetes 集群，参见#unique_48。

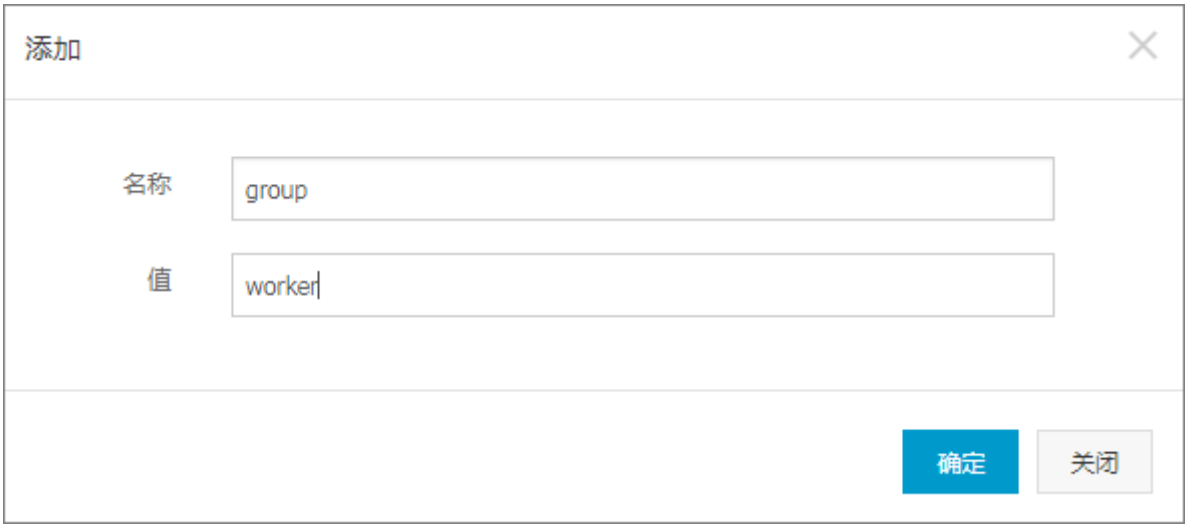
步骤1 为节点添加标签

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。

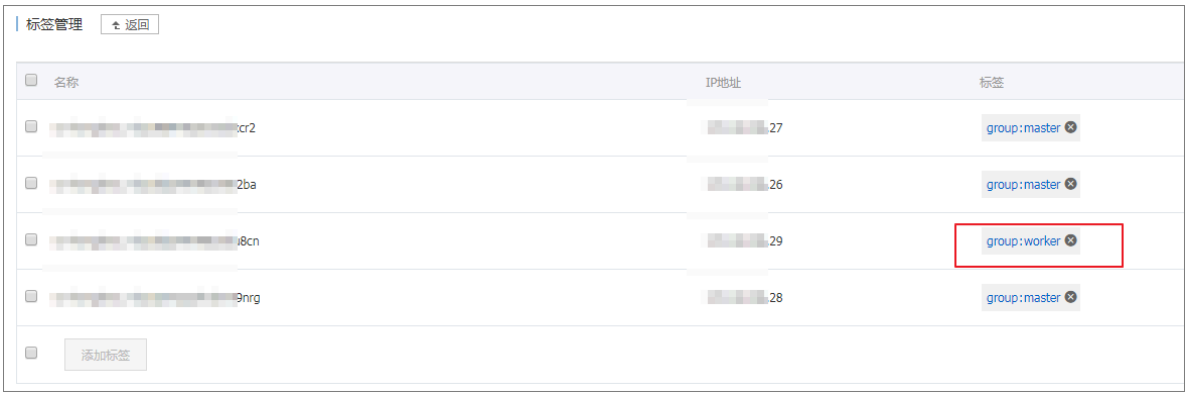
4. 在节点列表中，选择所需节点，然后单击添加标签。本例中选择一个 worker 节点。



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。

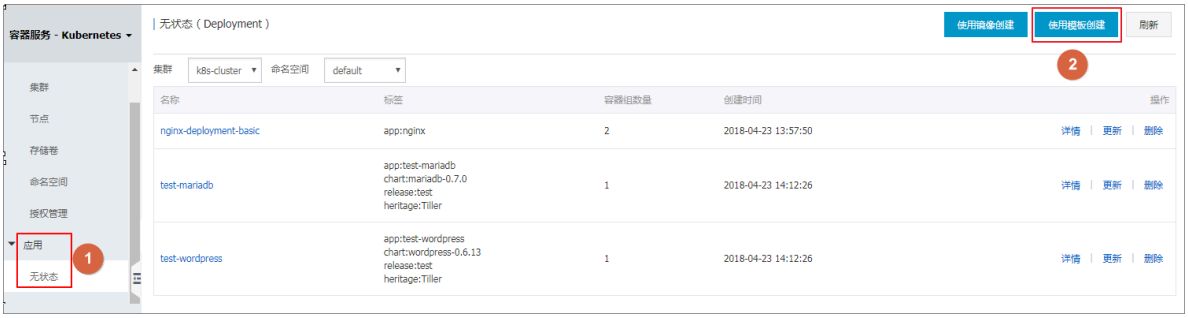


您也可以通过命令 `kubectl label nodes <node-name> <label-key>=<label-value>` 为节点添加标签。

步骤2 将 pod 部署到指定节点

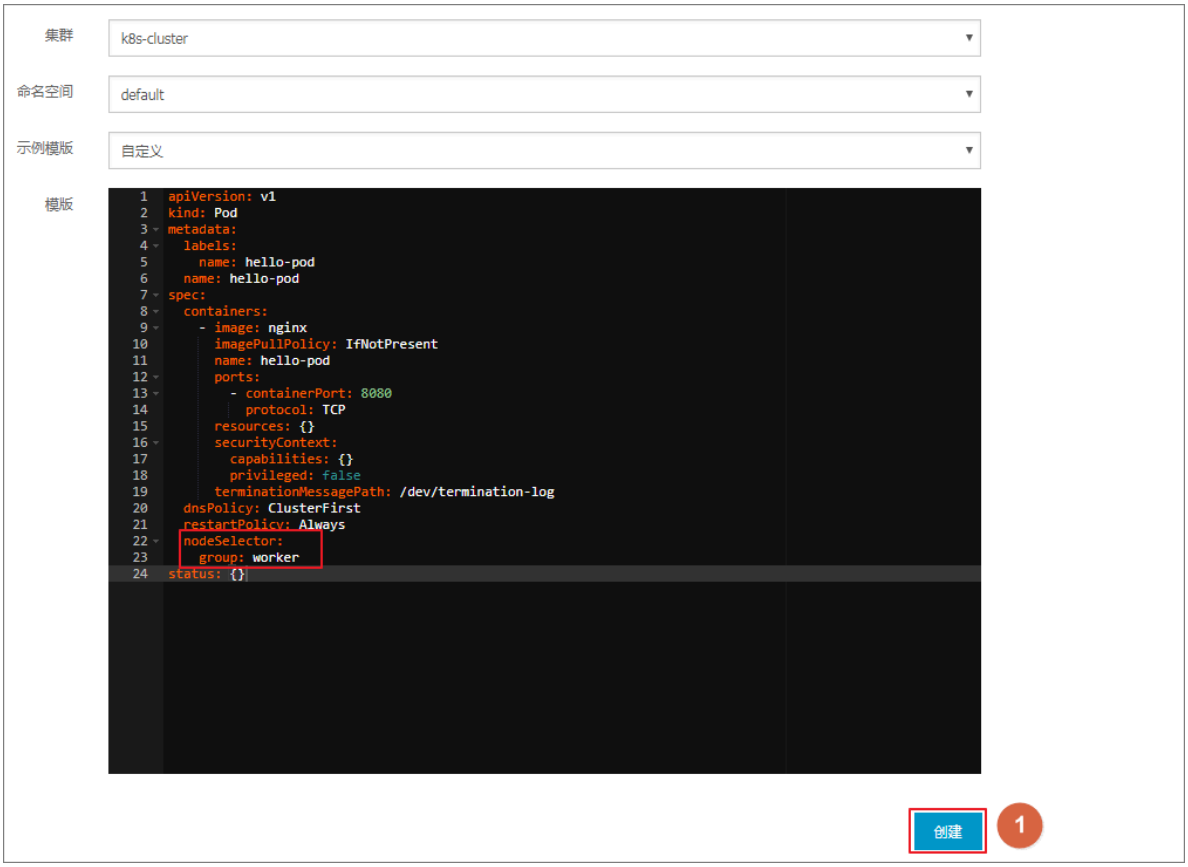
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，进入无状态（Deployment）页面。

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。

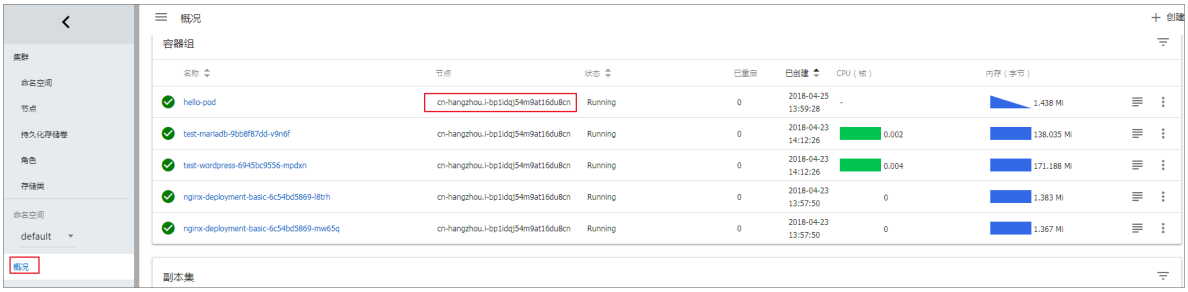


本示例的编排模板如下。

```
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。

6.11 查看容器组

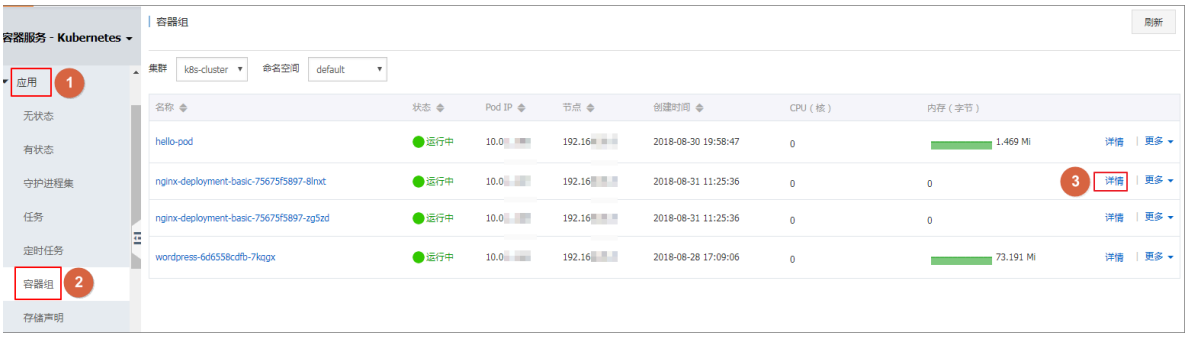
您可以通过Web界面查看容器组（Pod），并进行相关的操作。

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。



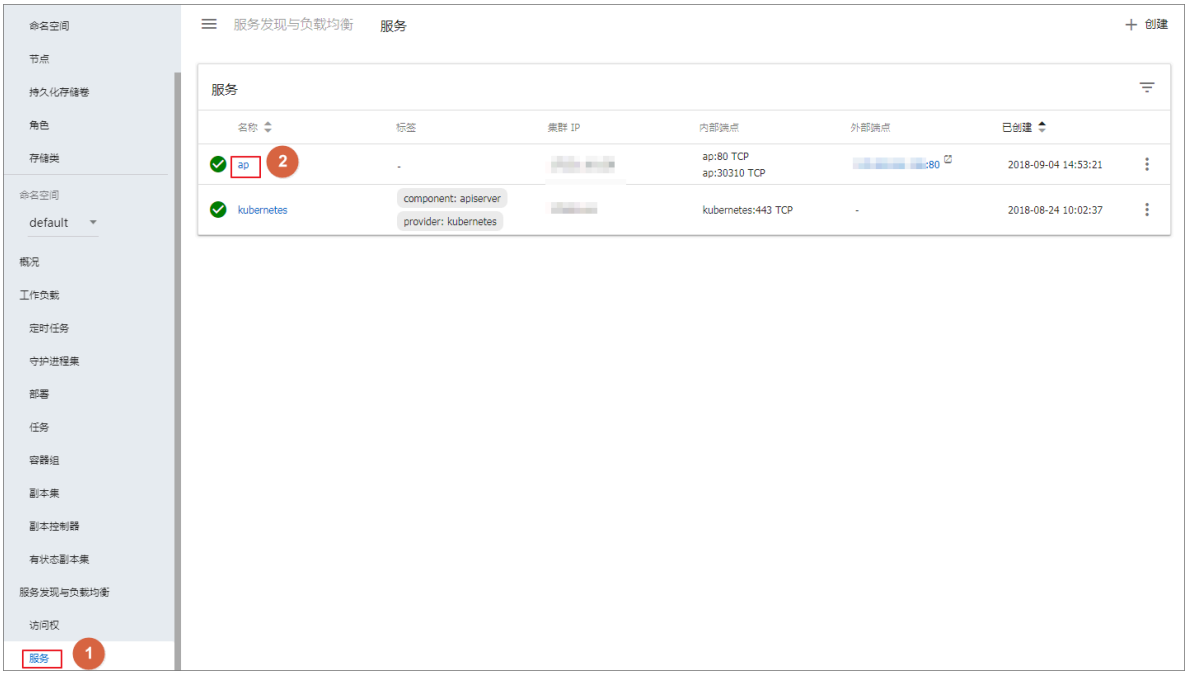
进入容器组的详情页，您可查看该容器组的详情信息。

通过 Kubernetes Dashboard 查看容器组

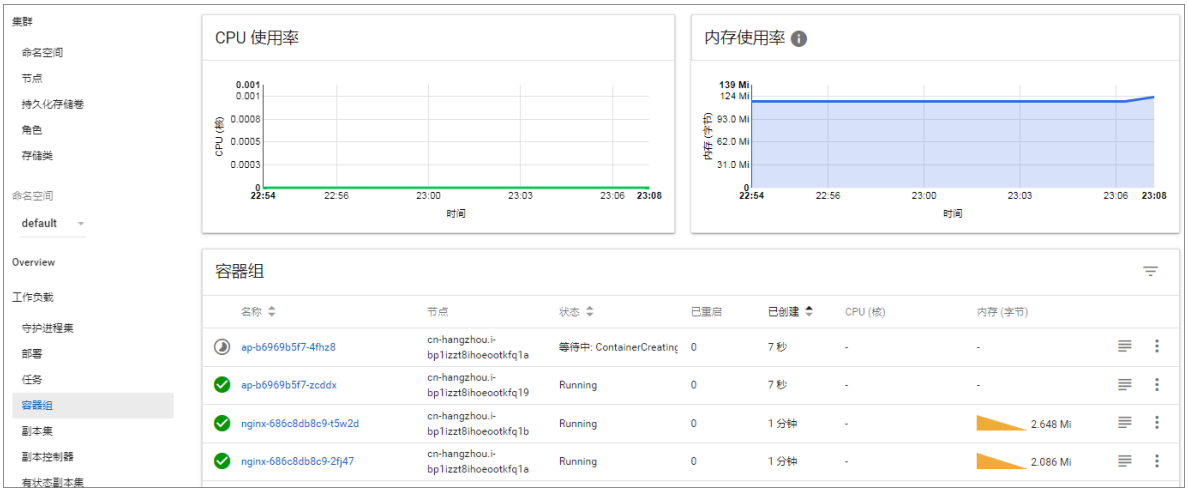
1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 集群，进入 Kubernetes 集群列表页面。
3. 选择所需的集群并单击右侧的控制台，进入 Kubernetes Dashboard。

4. 单击左侧导航栏中的容器组，查看集群中的容器组。

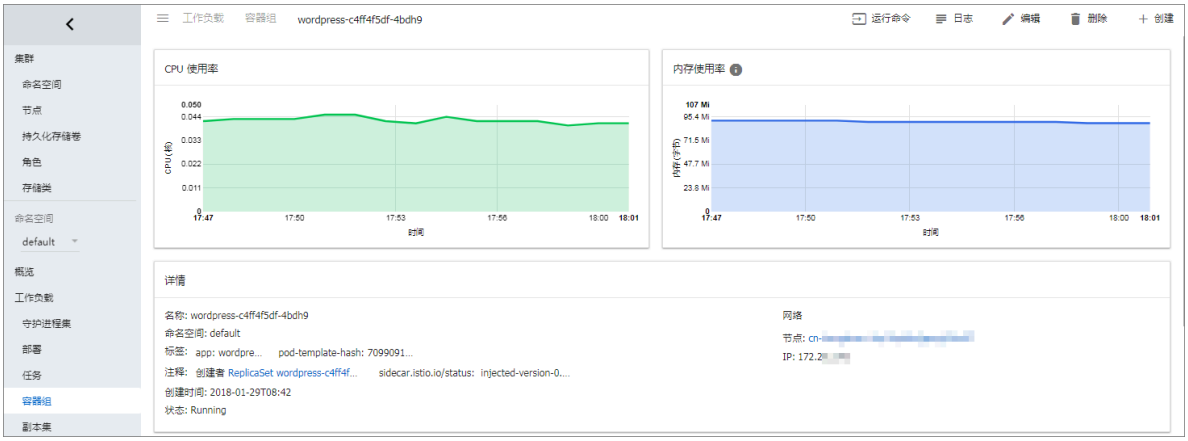
   或者，您可以单击左侧导航栏中的服务，选择所需的服务并单击服务的名称，查看该服务下的容器组。



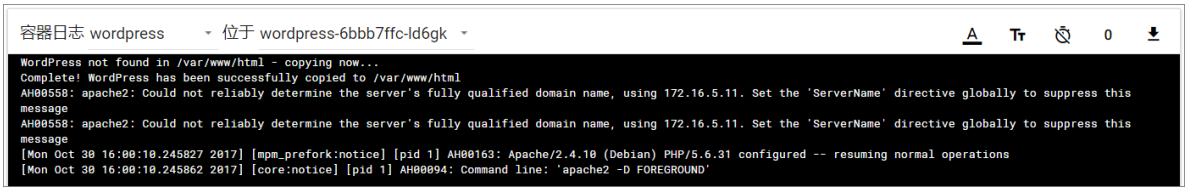
您可以通过左侧的图标查看每个 Kubernetes 对象的状态。 表示对象仍然处于部署状态。 表示对象已经完成部署。



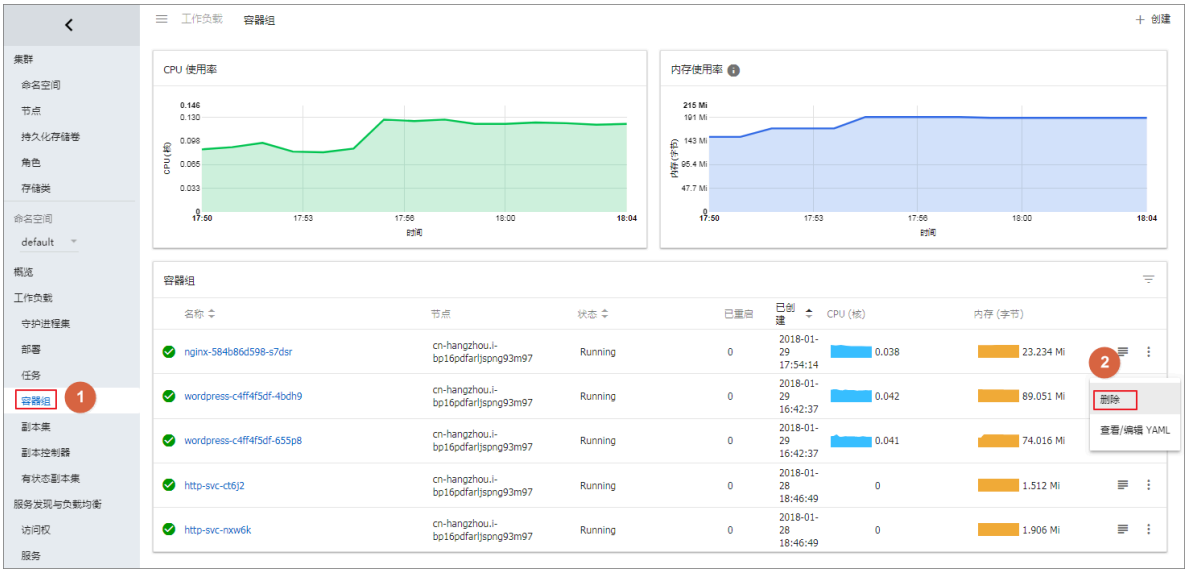
5. 选择所需的容器组，单击容器组的名称，您可以查看容器组的详细信息以及容器组使用的 CPU 和 Memory。



6. 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。



7. 您可以单击右侧的删除，删除该容器组。



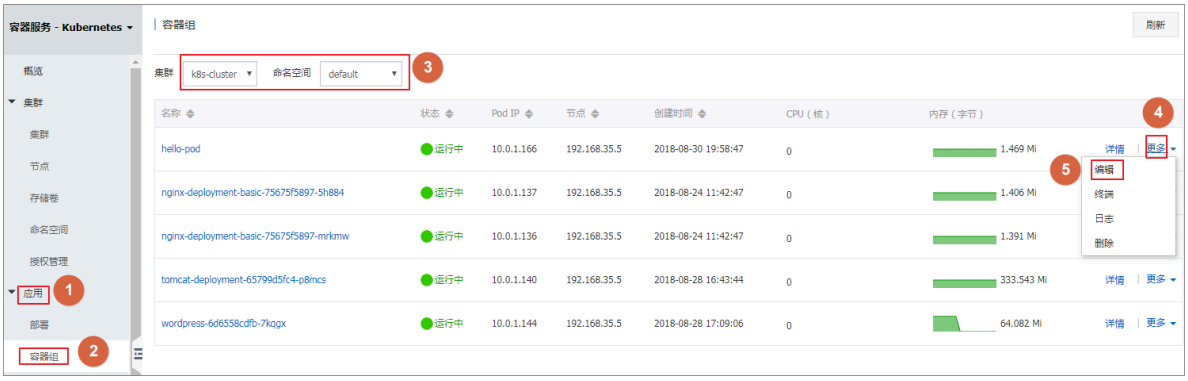
6.12 变更容器配置

您可以通过容器服务管理控制台变更容器组（Pod）的配置。

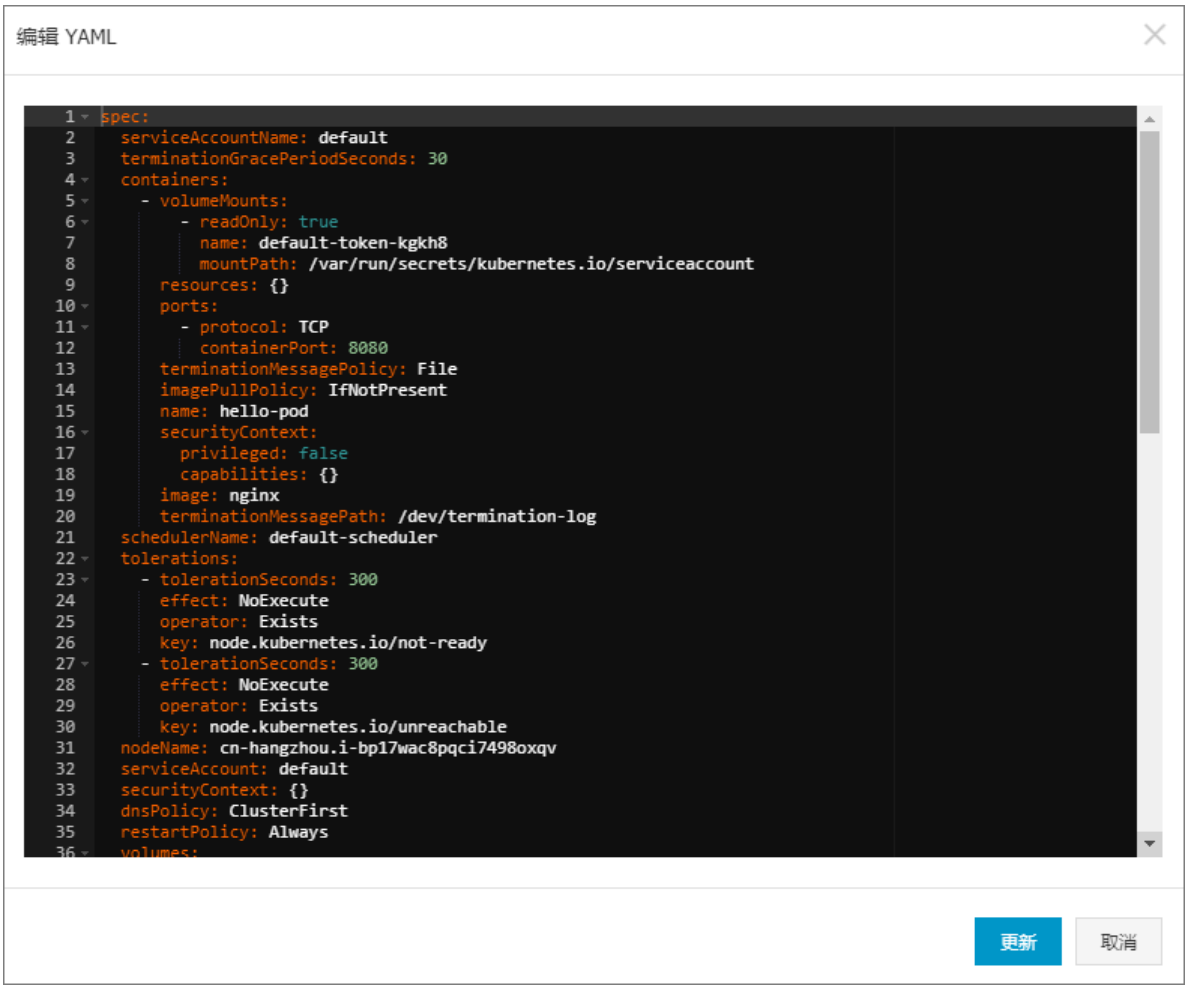
操作步骤

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。



6.13 手动伸缩容器应用

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。

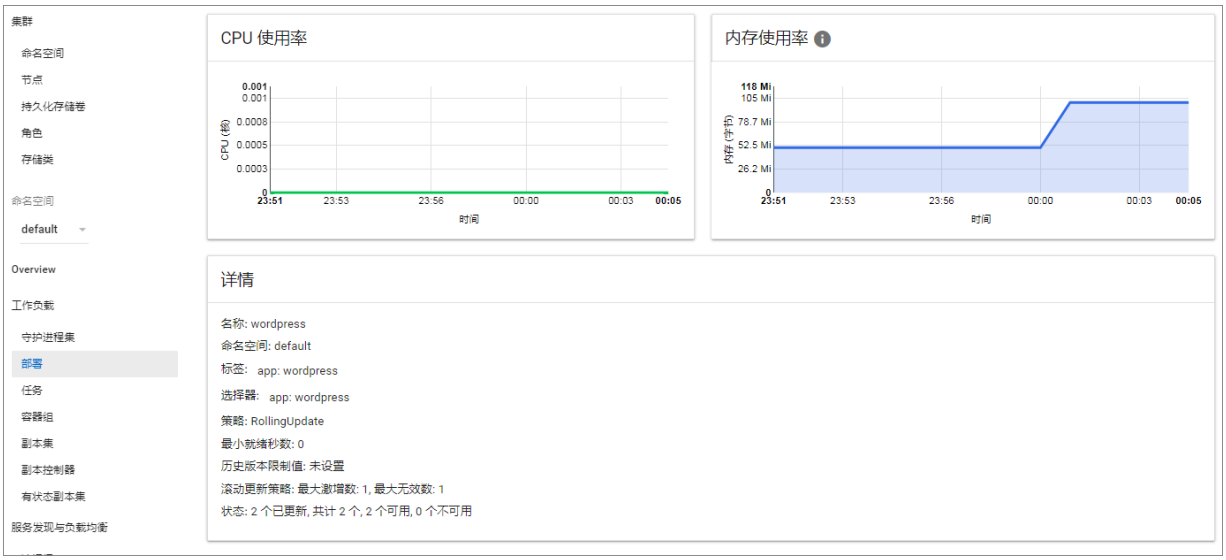
   此操作会扩容一个 Pod，将副本数升到 2。

后续步骤

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。



6.14 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

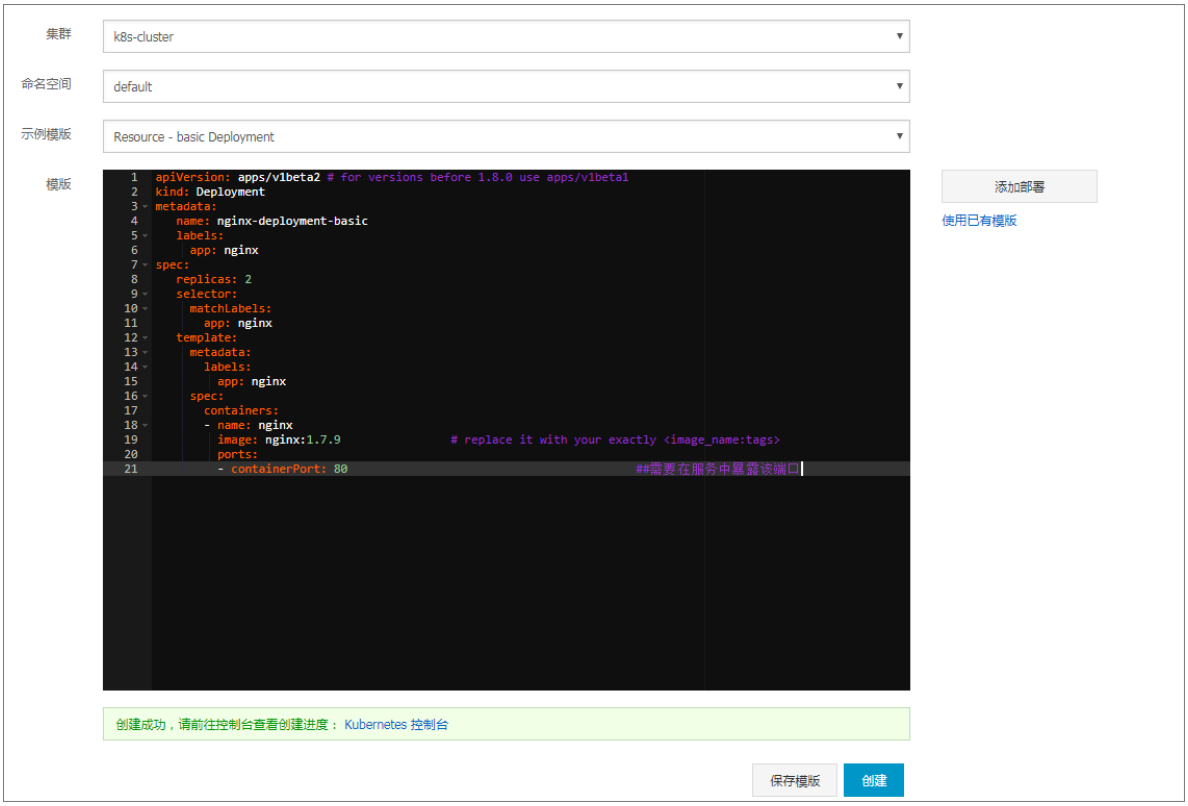
您已经成功创建一个 Kubernetes 集群，参见#unique_48。

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 > 无状态，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。

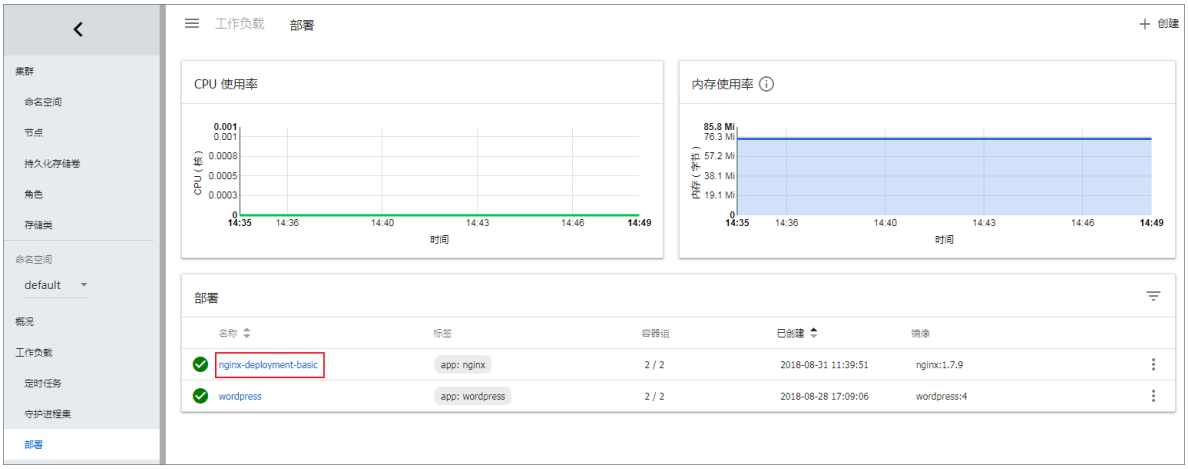


本例中，示例模板是一个 nginx 的 deployment。

```
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
```

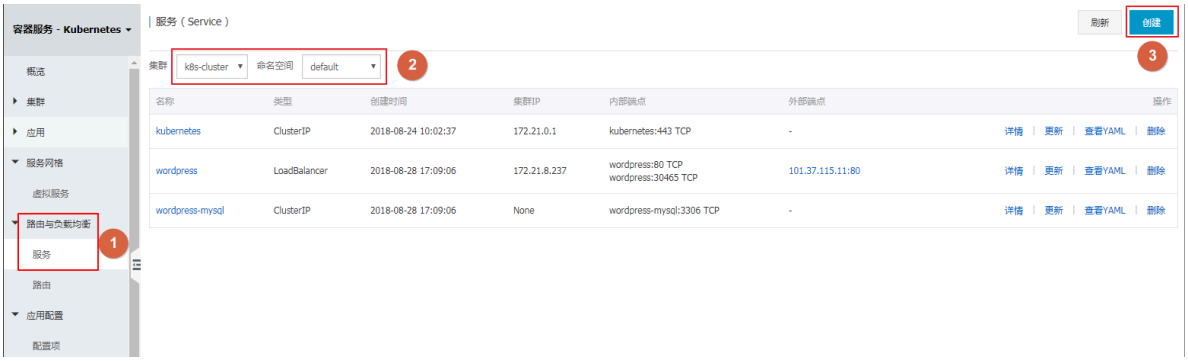
```
      spec:
        containers:
        - name: nginx
          image: nginx:1.7.9                    # replace it with your
exactly <image_name:tags>
          ports:
          - containerPort: 80
  ##需要在服务中暴露该端口
```

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路由与负载均衡 > 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。

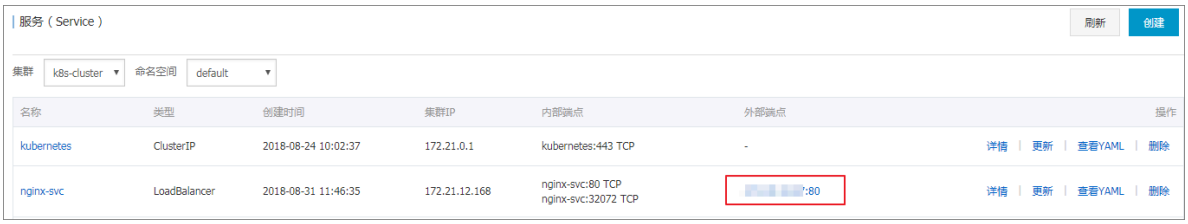
- 名称：输入服务的名称，本例中为 nginx-vc。
- 类型：选择服务类型，即服务访问的方式，包括：
  - 虚拟集群 IP：即 ClusterIP，指通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部可以访问，这也是默认的 ServiceType。

- 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。

> **说明：**
> 负载均衡类型支持使用已有 SLB，且多个 Kubernetes Service 可以复用同一个 SLB，但是存在以下限制：
>
> ■ 使用已有的负载均衡实例会强制覆盖已有监听。
> ■ Kubernetes 通过 Service 创建的 SLB 不能复用（会导致 SLB 被意外删除）。只能复用您手动在控制台（或调用 OpenAPI）创建的 SLB。
> ■ 复用同一个 SLB 的多个 Service 不能有相同的前端监听端口，否则会造成端口冲突。
> ■ 复用 SLB 时，监听的名字以及虚拟服务器组的名字被 Kubernetes 作为唯一标识符。请勿修改监听和虚拟服务器组的名字。
> ■ 不支持跨集群复用 SLB。

· 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#unique_110。
· 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。

6.15 查看服务

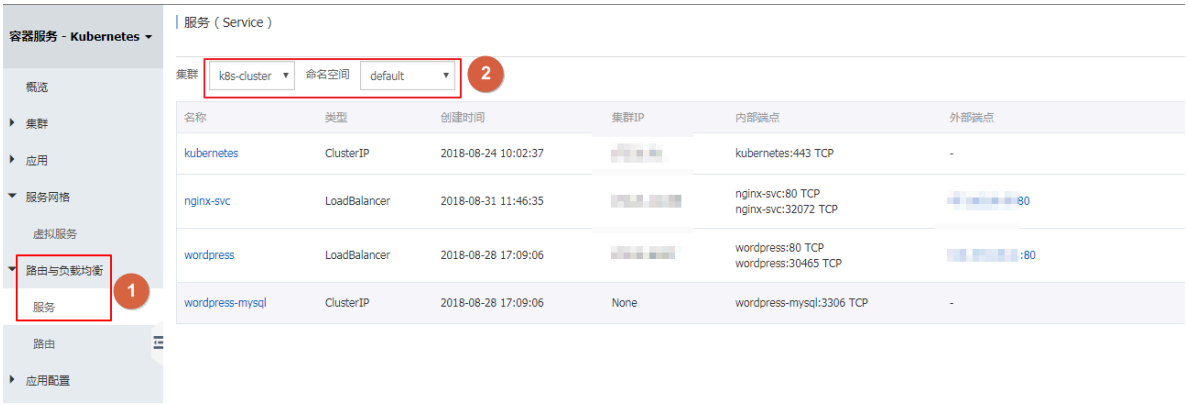
背景信息

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。

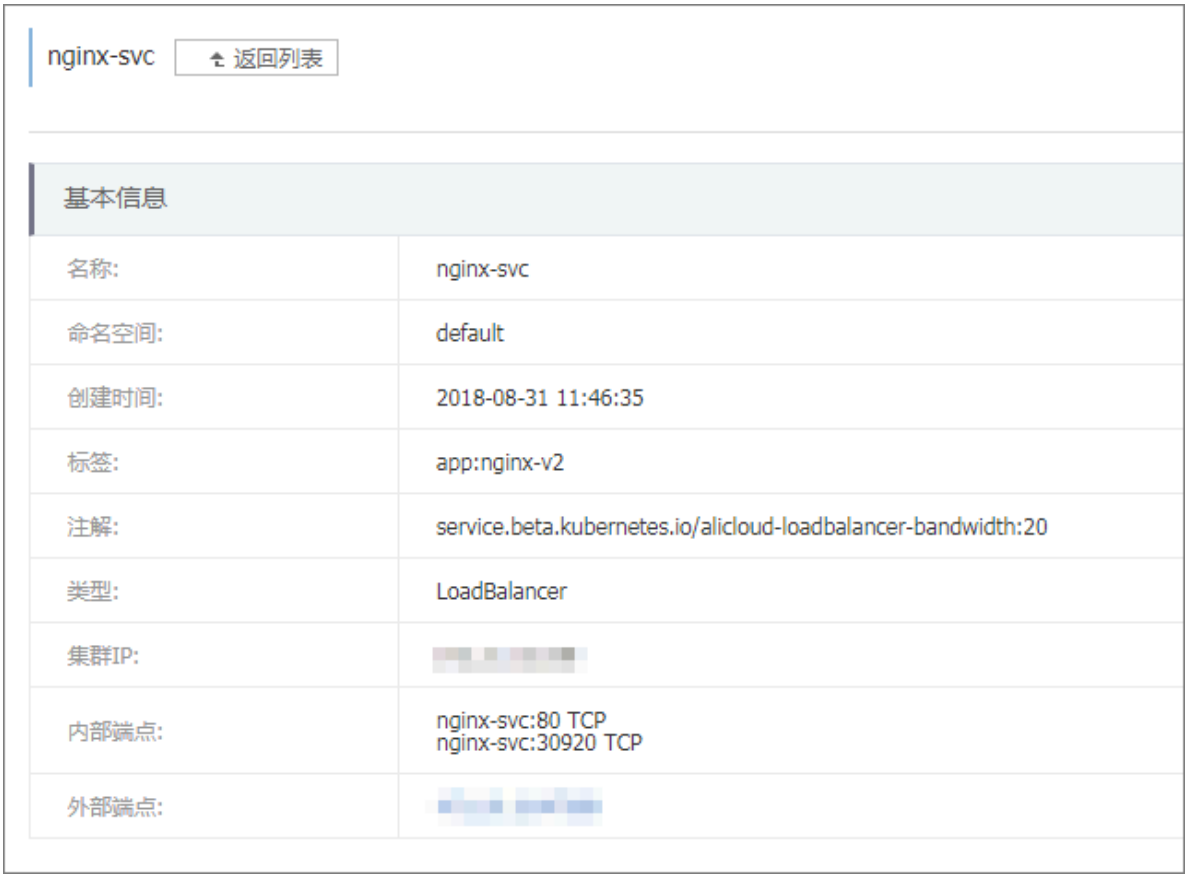
操作步骤

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路由与负载均衡 > 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。



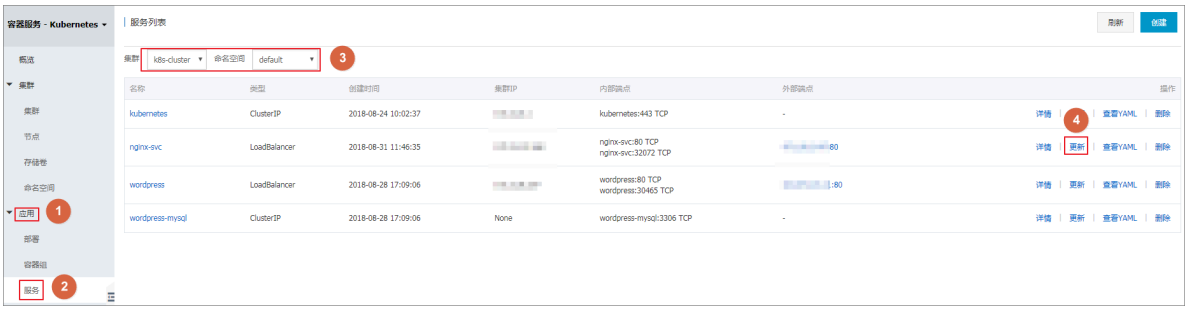
4. （可选） 您也可进入集群的 Kubernetes Dashboard。在左侧导航栏中单击服务，查看所有服务。

6.16 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。

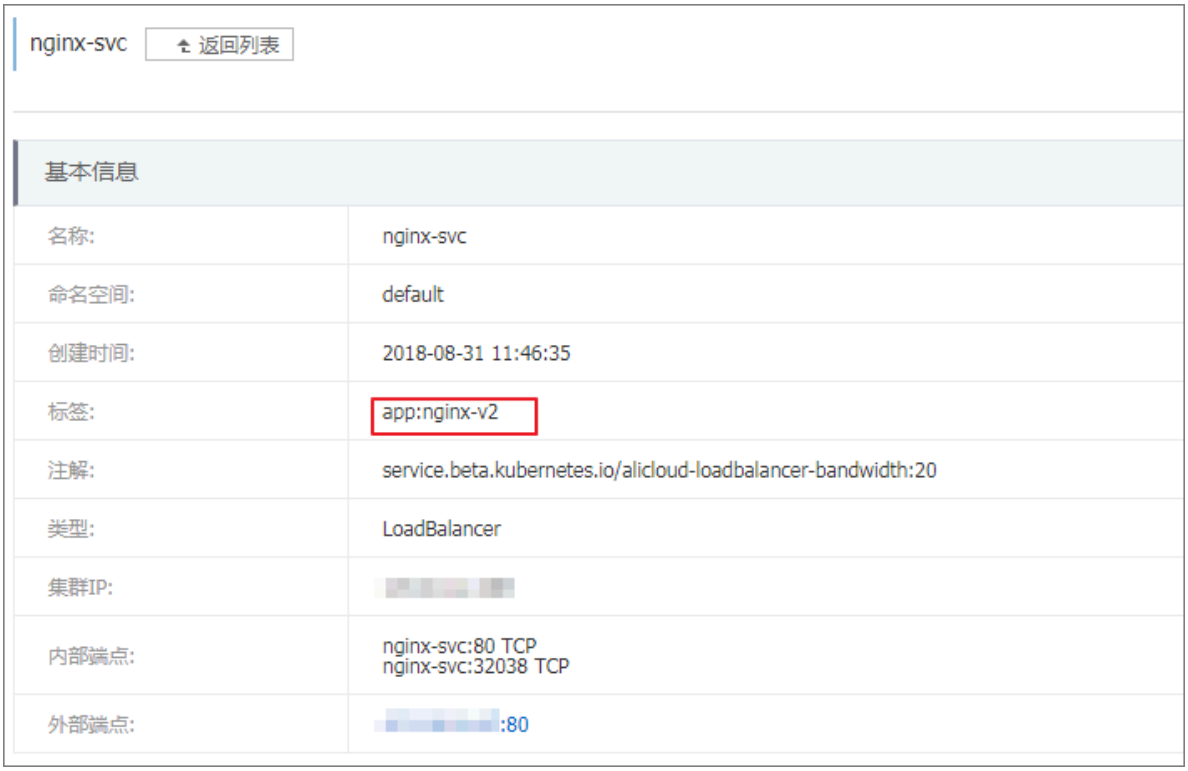
通过容器服务控制台服务列表页面

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。

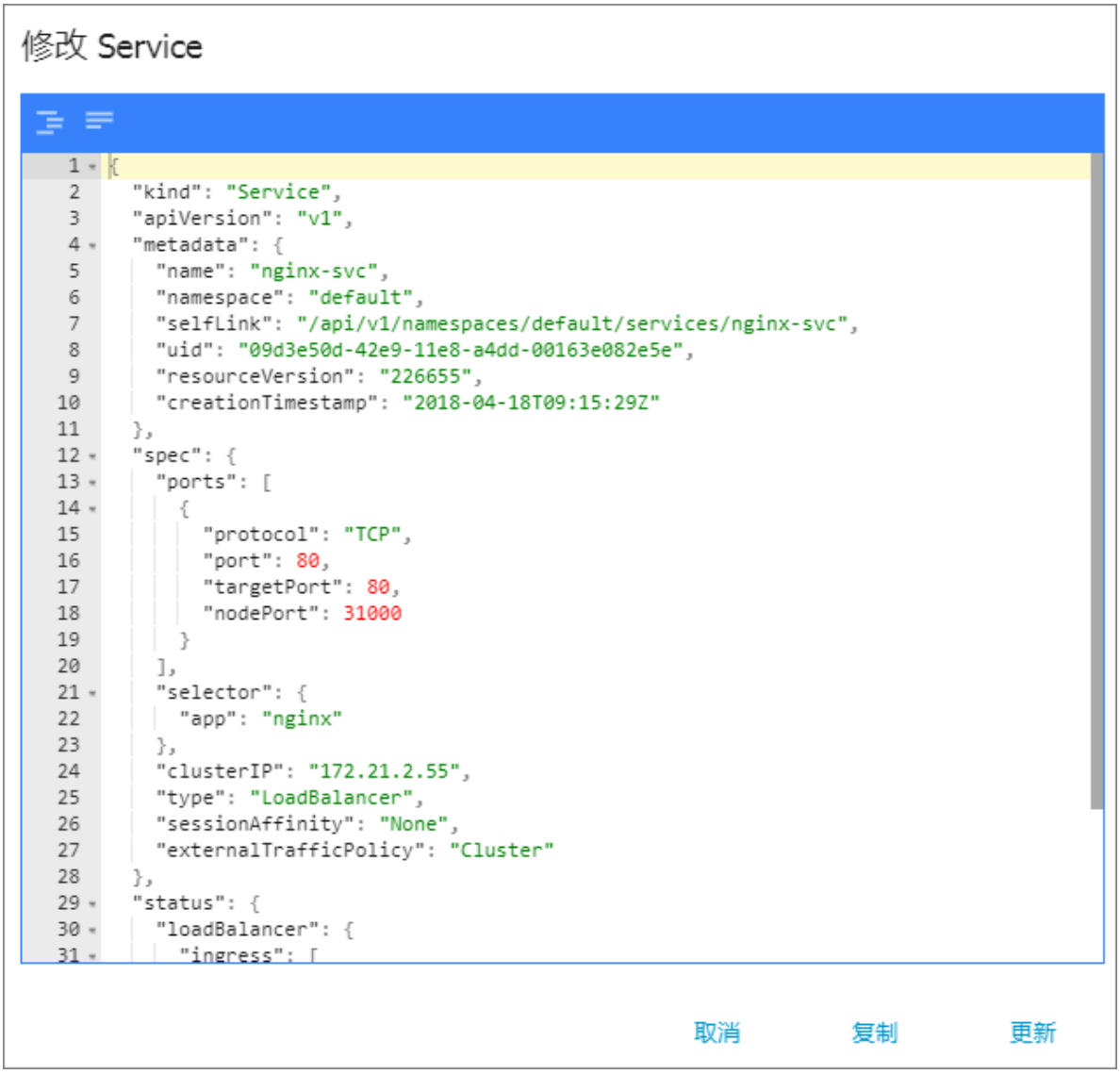


通过 Kubernetes Dashboard

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。



6.17 删除服务

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

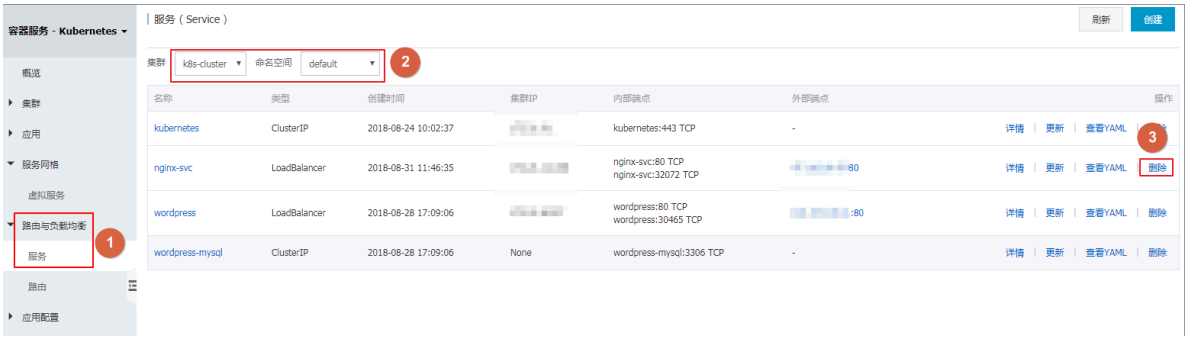
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已经成功创建一个服务，参见#unique_129。

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路由与负载均衡 > 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。

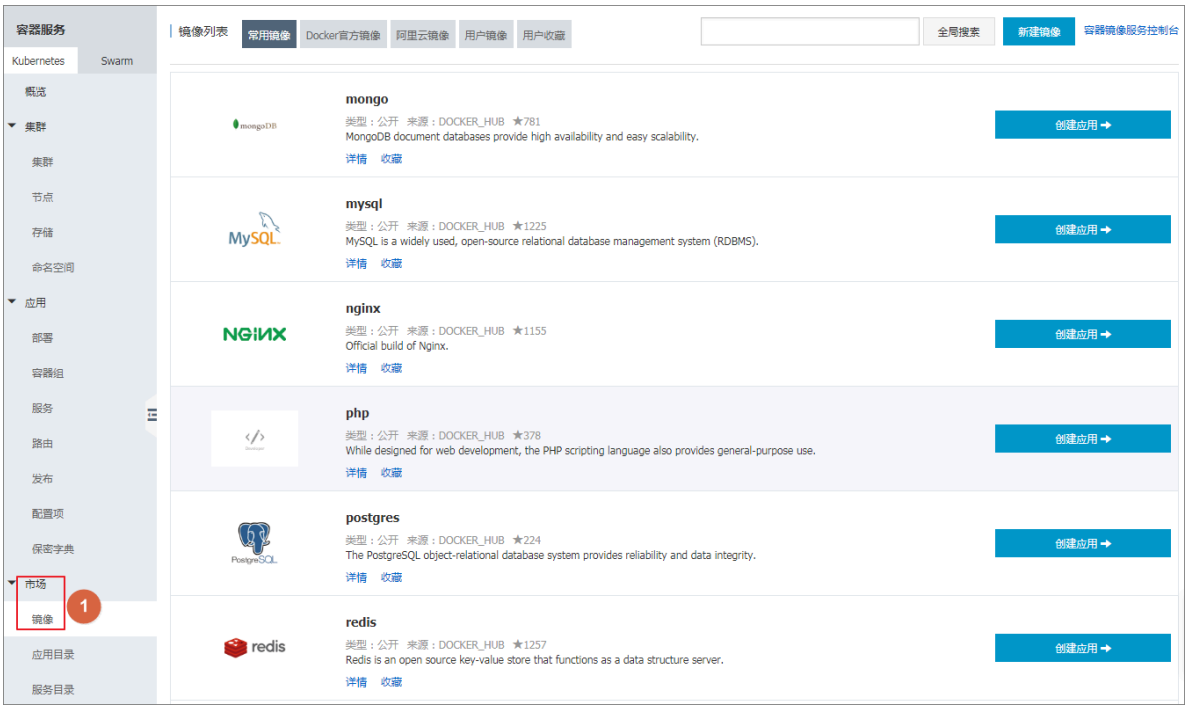


4. 在弹出的窗口中，单击确定，确认删除，该服务在服务列表中消失。

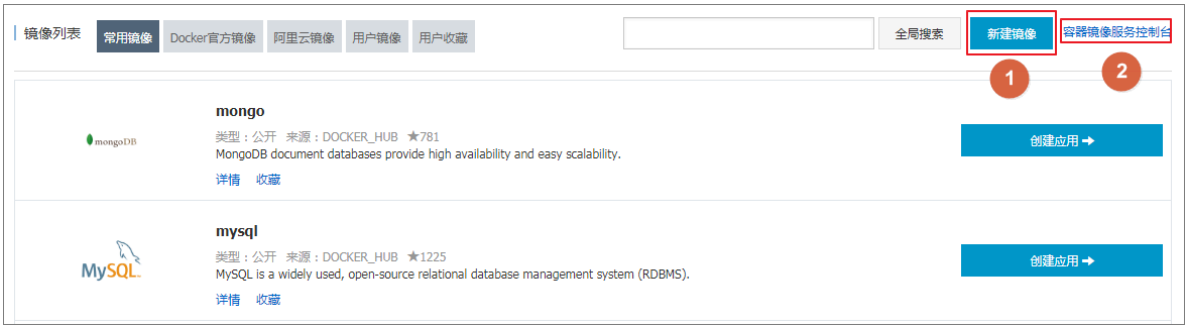
6.18 创建镜像

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。



3. 单击页面右上角的新建镜像，或直接单击容器镜像服务控制台。



4. 页面跳转到容器镜像服务管理控制台

   如果您是第一次访问，系统会提示您进行初始化设置。设置您的 Docker 登录密码并单击确定。

   然后，您可以申请一个仓库的命名空间，并单击右上角的创建镜像仓库进行镜像仓库的创建。

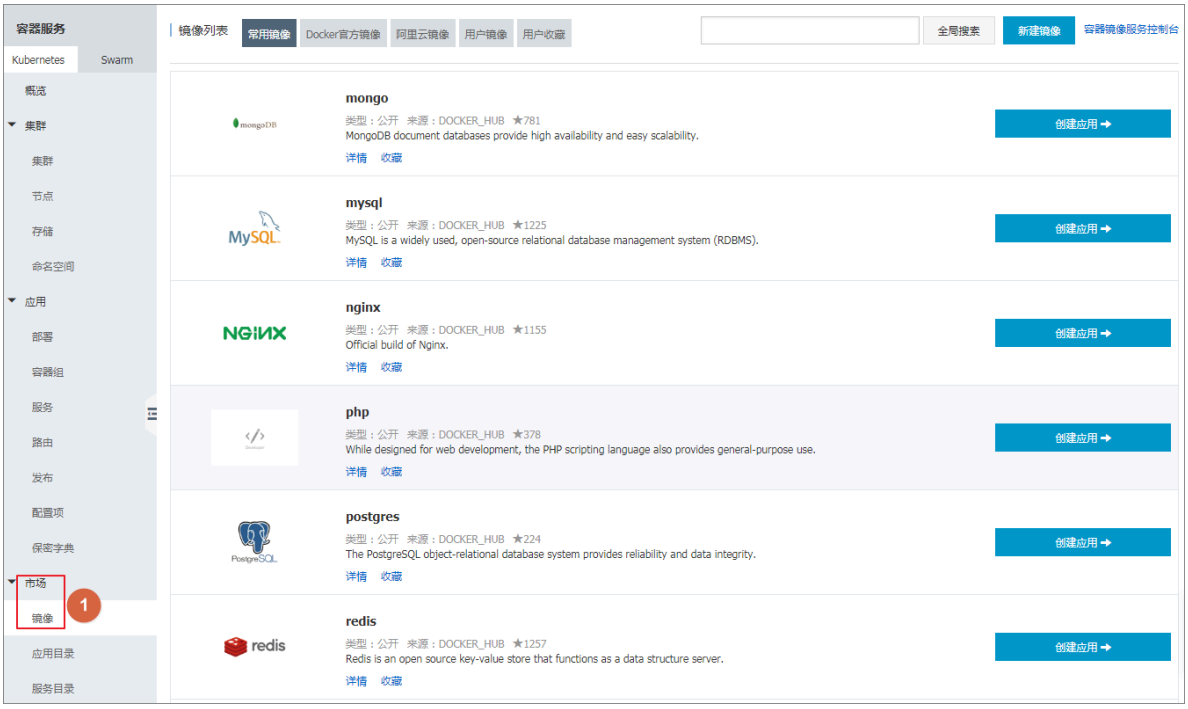
   您可以选择通过命令行上传自己的镜像（本地仓库），也可以选择第三方代码仓库进行自动构建（GitHub\Bitbucket）。

6.19 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或单击全局搜索通过镜像名称的关键字搜索镜像。

- 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。
- 阿里云镜像： 阿里云容器 Hub 提供的镜像，包含公开镜像和私有镜像。
- 用户镜像： 用户创建的镜像。
- 用户收藏： 用户可收藏其他页签下的镜像，也可以在 容器镜像服务下进行收藏，都会同步到该列表下。

6.20 使用镜像密钥

容器服务Kubernetes集群支持通过Web界面使用镜像密钥，包括新建镜像密钥和使用已有镜像密钥。

前提条件

- 您已成功创建一个Kubernetes集群，参见#unique_48。
- 您已构建一个私有镜像仓库，并将镜像上传到该仓库中。本例中使用阿里云容器镜像服务作为示例，参见#unique_133。

背景信息

您在使用私有镜像创建应用时，为了保障镜像安全，通常需要为镜像设置密钥。通过容器服务控制台，您可方便地将私有镜像库的身份认证信息通过docker-registry类型的secret传入到kubernetes中。

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 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:
   本例仅对容器镜像密钥的配置进行说明，详细的容器配置说明请参见#unique_97。

5. 在容器配置页面，首先配置镜像名称，在镜像名称栏中输入私有镜像地址，填写格式是`domainname/namespace/imagename`。

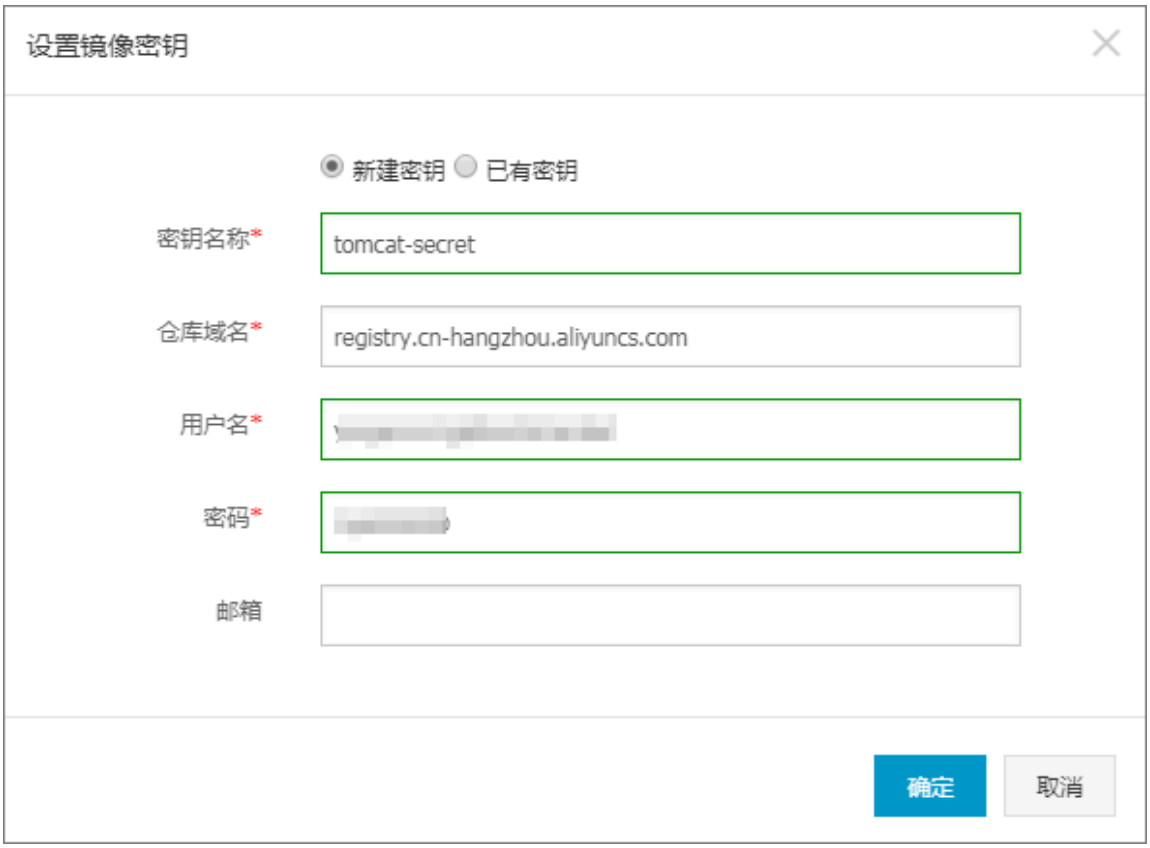
   说明:
   公共镜像无须配置镜像密钥。

6. 在镜像版本栏中，输入私有镜像地址版本，

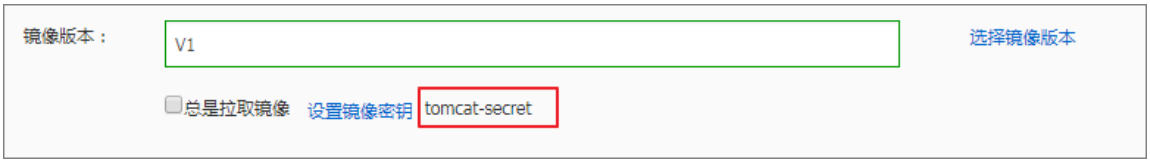
7. 单击设置镜像密钥，在弹出的对话框中进行设置。

   · 选择新建密钥，您需要配置以下参数。

   - 密钥名称（regsecret）： 指定密钥的键名称，可自行定义。
   - 仓库域名（docker-server）：指定 Docker 仓库地址。若输入阿里云容器服务镜像仓库，会默认补齐。
   - 用户名（docker-username）：指定 Docker 仓库用户名。若使用阿里云容器镜像服务，则为阿里云登录账号。
   - 密码（docker-password）：指定 Docker 仓库登录密码。若使用阿里云容器镜像服务，即容器registry独立登录密码。
   - 邮箱（docker-email）：指定邮件地址。非必选。



   单击确定，您可看到密钥已经成功创建。



   · 您也可单击已有密钥。您可通过命令行或Yaml文件等方式预先创建好容器镜像密钥，参见#unique_134和#unique_133。

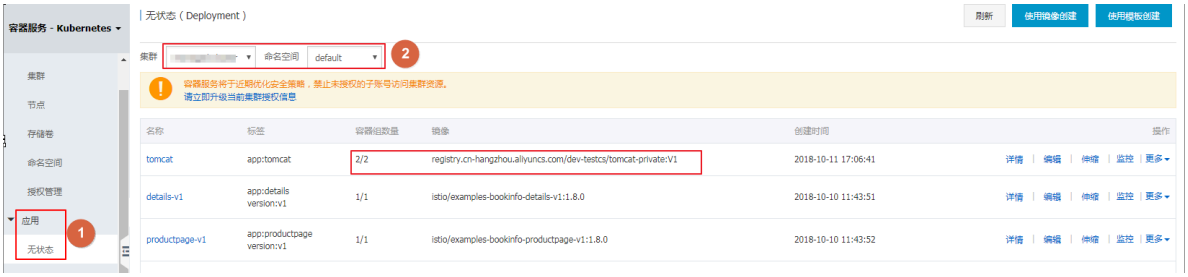
8. 完成配置后，单击下一步。

9. 根据页面引导，完成其他配置，最后单击创建。

10.单击左侧导航栏应用 > 无状态，选择集群和命名空间，查看Tomcat应用运行情况。

说明:

您可看到，Tomcat应用正常运行，成功通过密钥使用Tomcat私有镜像。



6.21 免密拉取镜像

本文介绍如何免密拉取阿里云容器镜像仓库中的私有镜像。

前提条件

您已经成功创建一个Kubernetes 集群，参见#unique_48。

背景信息

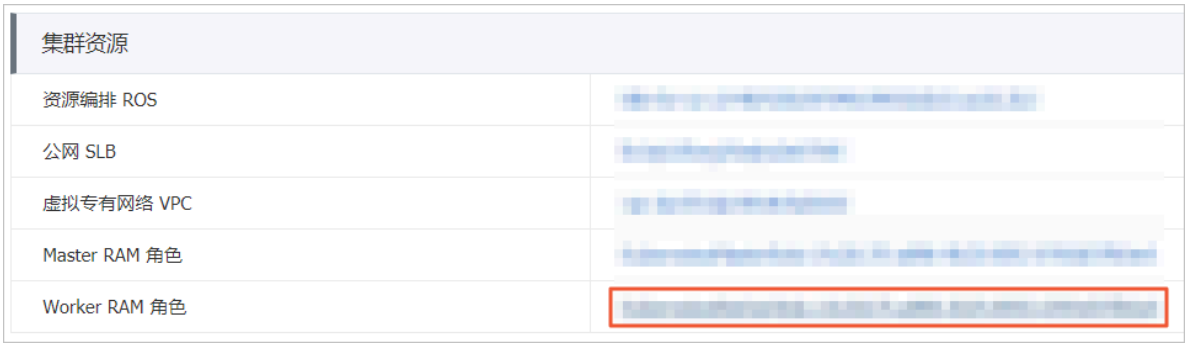
免密拉取功能：

- 仅支持拉取当前用户阿里云容器镜像仓库中的私有镜像，无法拉取其他用户的私有镜像。
- 支持跨地域拉取阿里云容器镜像仓库中的私有镜像。
- 支持多命名空间免密拉取。
- 支持配置免密拉取阿里云容器镜像服务企业版中的私有镜像。

- 支持的集群：
  - 专有版Kubernetes集群
  - 托管版Kubernetes集群
  - Serverless Kubernetes集群
- 支持的版本：
  - 专有版Kubernetes集群：高于或等于1.11.2的版本默认支持免密拉取镜像。版本低于1.11.2的版本请参考操作步骤。
  - 托管版Kubernetes集群：所有版本。
  - Serverless Kubernetes集群：所有版本。

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. 单击目标集群的名称，查看集群的详细信息。
4. 单击集群资源区域的Worker RAM角色，跳转到RAM访问控制台的RAM角色管理页面。



说明：

本文以新版RAM访问控制台为例进行介绍。

若您使用的是旧版的RAM访问控制台：

方法一

a. 单击左侧导航栏角色管理，在角色名中输入Worker RAM角色的名称进行搜索。单击角色名称。

b. 在基本信息区域，单击页面右上角的编辑基本信息。

方法二

单击页面右下角体验新版，切换到新版RAM访问控制台。在容器服务管理控制台单击Worker RAM 角色重新登录RAM访问控制台。

5. 在RAM角色管理页面，单击权限管理区域的权限策略名称，查看具体的权限策略。

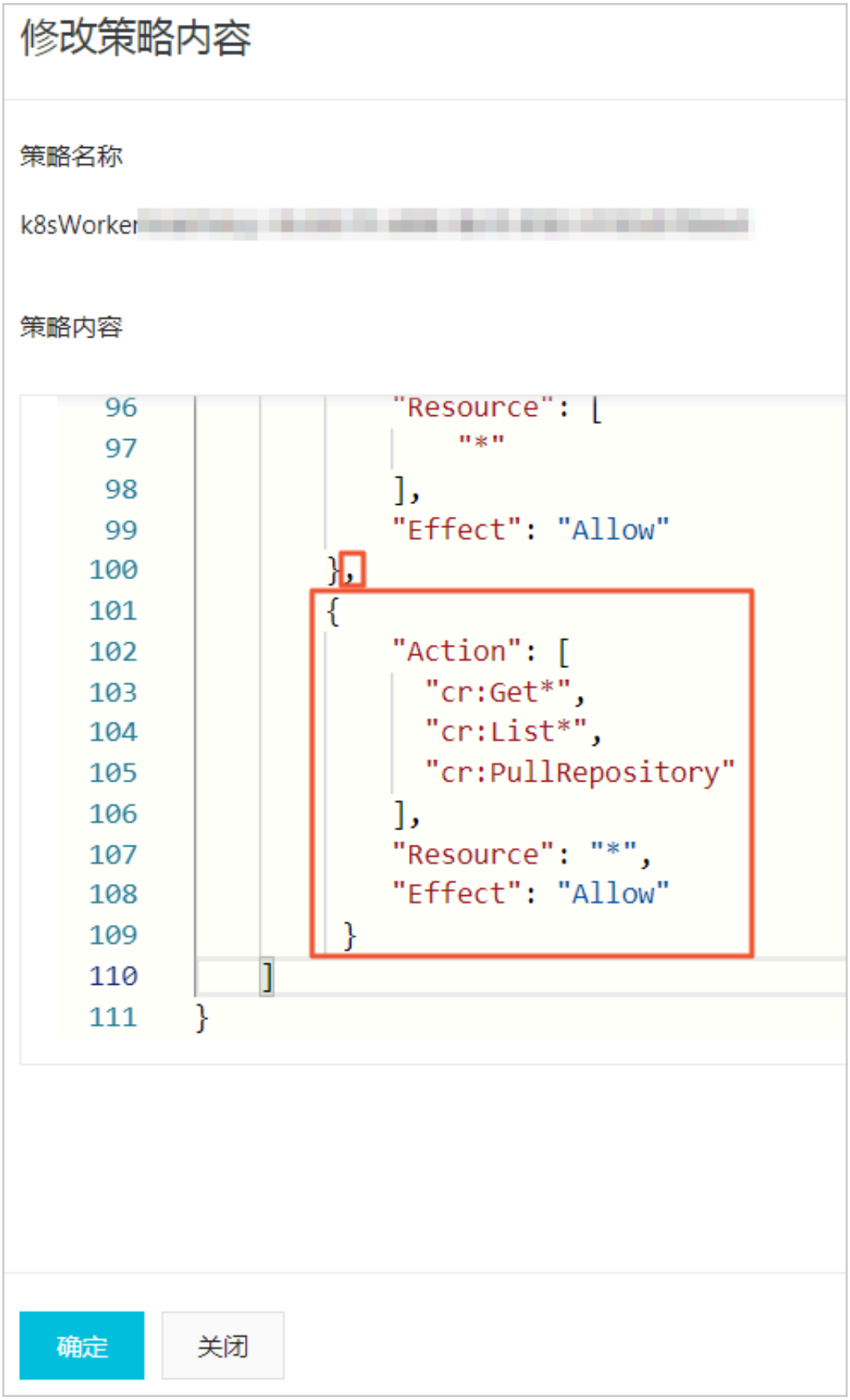
6. 在权限策略管理页面，单击策略内容区域的修改策略内容。

7. 在策略内容区域增加以下字段后，单击确定。

```
{
            "Action": [
              "cr:Get*",
```

```
                "cr:List*",
                "cr:PullRepository"
              ],
              "Resource": "*",
              "Effect": "Allow"
```

```
        }
```


修改策略内容
策略名称
k8sWorker
策略内容
96 "Resource": [
97 "*"
98],
99 "Effect": "Allow"
100 },
101 {
102 "Action": [
103 "cr:Get*",
104 "cr:List*",
105 "cr:PullRepository"
106],
107 "Resource": "*",
108 "Effect": "Allow"
109 }
110]
111 }
确定
关闭

8. 创建*aliyun-acr-credential-helper*服务，定时刷新容器镜像服务的临时token密码。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: acr-configuration
  namespace: kube-system
data:
    #具体配置说明参考下文
    acr-api-version: "2018-12-01"
    #acr-registry: "xxx-registry.*.cr.aliyuncs.com,xxx-registry-vpc
.*.cr.aliyuncs.com"
    #watch-namespace: "all"
    #expiring-threshold: "15m"
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: aliyun-acr-credential-helper
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: aliyun-acr-credential-helper
rules:
  - apiGroups:
      - ""
    resources:
      - namespaces
      - configmaps
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - serviceaccounts
      - secrets
    verbs:
      - create
      - update
      - patch
      - get
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: aliyun-acr-credential-helper
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: aliyun-acr-credential-helper
subjects:
  - kind: ServiceAccount
    name: aliyun-acr-credential-helper
    namespace: kube-system
---
apiVersion: apps/v1beta2
kind: Deployment
```

```
metadata:
  name: aliyun-acr-credential-helper
  namespace: kube-system
  annotations:
    component.version: "v19.01.28"
    component.revision: "v1"
  labels:
    app: aliyun-acr-credential-helper
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aliyun-acr-credential-helper
  template:
    metadata:
      labels:
        app: aliyun-acr-credential-helper
    spec:
      serviceAccountName: aliyun-acr-credential-helper
      containers:
      - name: aliyun-acr-credential-helper
        image: registry.cn-hangzhou.aliyuncs.com/acs/aliyun-acr-
credential-helper:v19.01.28.0-f063330-aliyun
        imagePullPolicy: Always
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        volumeMounts:
        - name: localtime
          mountPath: /etc/localtime
          readOnly: true
      volumes:
        - name: localtime
          hostPath:
            path: /etc/localtime
            type: File
      nodeSelector:
        beta.kubernetes.io/os: linux
```

ACR Credential Helper组件配置项可通过 ConfigMap 的方式来进行配置，并会自动实时生效。

表 6-1: 配置说明

配置项	配置项说明	默认值
acr-api-version	期望使用的CR OpenAPI Version	2018-12-01 （支持企业版Registy API）

配置项	配置项说明	默认值
acr-registry	期望能免密拉取镜像的Registry	registry.*.aliyuncs.com,registry-vpc.*.aliyuncs.com,xxx-registry.*.cr.aliyuncs.com,xxx-registry-vpc.*.cr.aliyuncs.com 说明: 如果需要配置多个Registry时，以逗号分隔。
watch-namespace	期望能免密拉取镜像的Namespace	default 说明: 当取值为All时，表示期望所有Namespace都能免密拉取。 如果需要配置多个Namespace时，以逗号分隔。
expiring-threshold	本地Cache Token快过期阈值	15m

7 工作流

7.1 创建工作流

本文主要为您介绍如何创建工作流。

背景信息

工作流是基于argo开发的，为Kubernetes提供容器化的本地工作流程。工作流程中的每个步骤都定义为容器。

工作流是作为 Kubernetes CRD（自定义资源定义）实现的。 因此，可以使用kubectl管理工作流，并与其他Kubernetes 服务本地集成，例如Volumes、Secrets 和 RBAC。 工作流控制器提供完整的工作流程功能，包括参数替换，存储，循环和递归工作流程。

前提条件

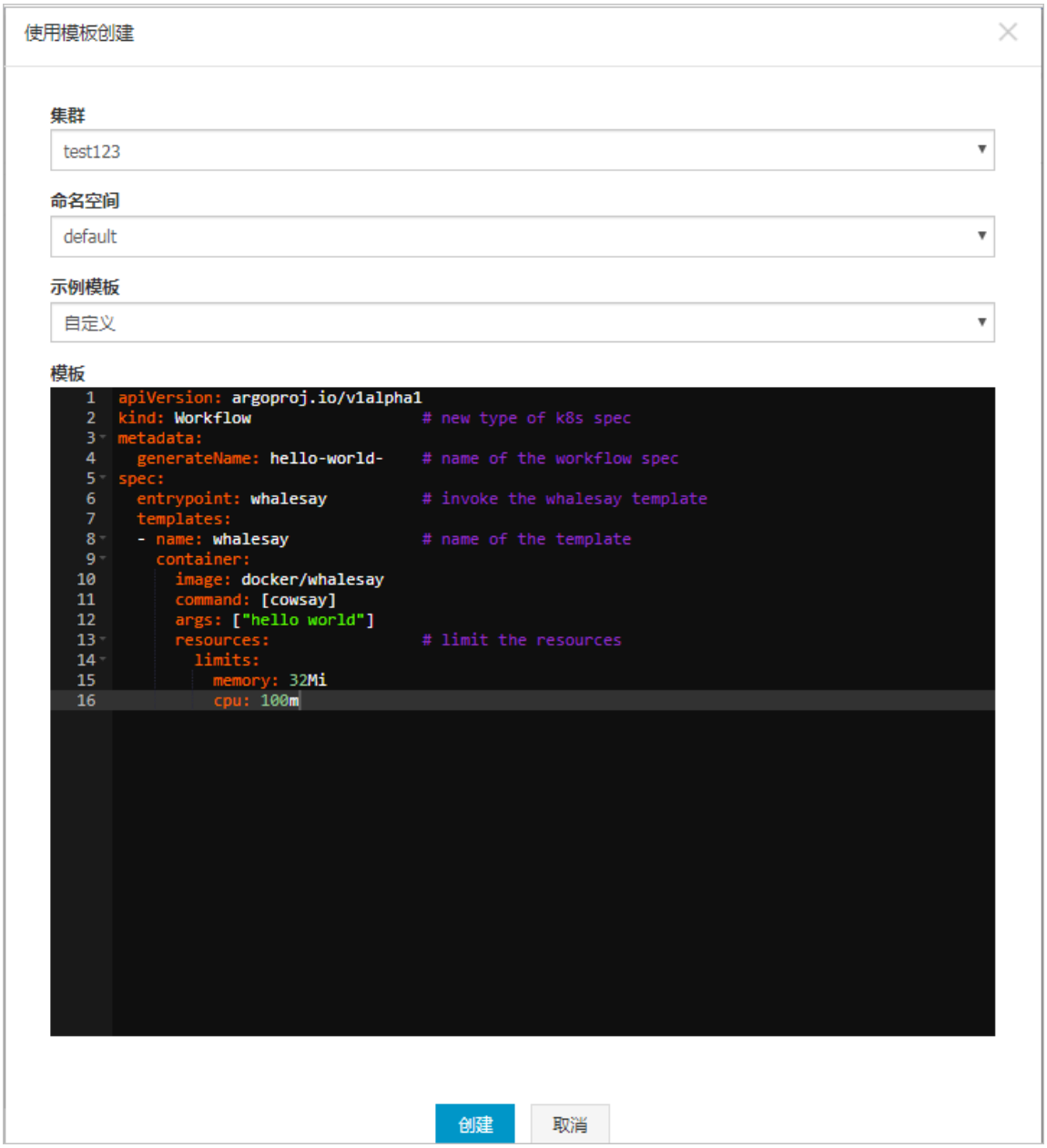
- 您已成功创建一个 Kubernetes 集群。参见#unique_48。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

操作步骤

您可以通过控制台和命令行的方式创建工作流。

- 通过控制台创建Hello World工作流

  1. 登录 容器服务管理控制台。
  2. 在 Kubernetes 菜单下，选择应用 > 工作流，单击右上角的创建。
  3. 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。工作流将部署在该集群内。
  - 命名空间：选择工作流所属的命名空间，默认是 default。
  - 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，让您快速部署工作流。您可以根据 Kubernetes Yaml 编排的格式要求自主编写，来描述您想定义的工作流。

下面是一个 Hello World 工作流的示例编排，基于容器服务自定义的编排模板。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow                  # new type of k8s spec
metadata:
  generateName: hello-world-    # name of the workflow spec
spec:
  entrypoint: whalesay          # invoke the whalesay template
  templates:
  - name: whalesay              # name of the template
    container:
      image: docker/whalesay
      command: [cowsay]
      args: ["hello world"]
      resources:                # limit the resources
        limits:
          memory: 32Mi
          cpu: 100m
```

4. 几秒钟后，返回工作流（Workflow）页面，您可以看到已经创建成功的工作流。

您可以单击详情，了解该工作流基本信息和器组信息。

- 通过命令行创建Parameters工作流

1. 创建并拷贝内容到*arguments-parameters.yaml*文件中，并执行`ags submit arguments-parameters.yaml -p message="goodbye world"`命令，创建 Parameters 工作流。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-parameters-
spec:
  # invoke the whalesay template with
  # "hello world" as the argument
  # to the message parameter
  entrypoint: whalesay
  arguments:
    parameters:
    - name: message
      value: hello world

  templates:
  - name: whalesay
    inputs:
      parameters:
      - name: message       # parameter declaration
    container:
      # run cowsay with that message input parameter as args
      image: docker/whalesay
      command: [cowsay]
```

```
      args: ["{{inputs.parameters.message}}"]
```

您也可以参考Workflow 示例模板，通过更新 yaml 文件的方式创建其他工作流。

Ags CLI 是兼容阿里云定制的兼容社区版 argo 的命令行工具，使用 Ags CLI 可以方便的提交、查看、修改、删除工作流。关于更多内容请参见#unique_32。

7.2 Workflow 示例模板

本文给您提供Workflow 示例模板，便于您创建需要的工作流。

Steps

本示例中，我们将了解如何创建多步骤工作流，如何在工作流规范中定义多个模板，以及如何创建嵌套工作流。 请务必阅读注释，以增强代码的可读性。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello

  # This spec contains two templates: hello-hello-hello and whalesay
  templates:
  - name: hello-hello-hello
    # Instead of just running a container
    # This template has a sequence of steps
    steps:
    - - name: hello1            # hello1 is run before the following
steps
        template: whalesay
        arguments:
          parameters:
          - name: message
            value: "hello1"
    - - name: hello2a           # double dash => run after previous
step
        template: whalesay
        arguments:
          parameters:
          - name: message
            value: "hello2a"
      - name: hello2b           # single dash => run in parallel with
previous step
        template: whalesay
        arguments:
          parameters:
          - name: message
            value: "hello2b"

  # This is the same template as from the previous example
  - name: whalesay
    inputs:
      parameters:
      - name: message
    container:
      image: docker/whalesay
```

```
      command: [cowsay]
      args: ["{{inputs.parameters.message}}"]
```

Steps 工作流模板打印出三种不同风格的hello。 hello-hello-hello模板由三个步骤组成。 名称为*hello1*的第一步将按顺序运行，而后两个名称为*hello2a* 和 *hello2b*的步骤将彼此并行运行。 使用ags CLI命令，我们可以以图形方式显示此工作流规范的执行历史记录，该规则显示名为*hello2a*和*hello2b*的步骤彼此并行运行。

结果显示：

```
STEP                                        PODNAME
 # arguments-parameters-rbm92
 ├---# hello1                    steps-rbm92-2023062412
 └-·-# hello2a                   steps-rbm92-685171357
   └-# hello2b                   steps-rbm92-634838500
```

DAG

作为指定步骤序列的替代方法，您可以通过指定每个任务的依赖关系将工作流定义为有向非循环图（DAG）。 对于复杂的工作流程，这可以更简单地维护，并且在运行任务时允许最大的并行性。

在以下工作流程中，步骤A首先运行，因为它没有依赖项。 A完成后，步骤B和C并行运行。 最后，一旦B和C完成，步骤D就可以运行了。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: dag-diamond-
spec:
  entrypoint: diamond
  templates:
  - name: echo
    inputs:
      parameters:
      - name: message
    container:
      image: alpine:3.7
      command: [echo, "{{inputs.parameters.message}}"]
  - name: diamond
    dag:
      tasks:
      - name: A
        template: echo
        arguments:
          parameters: [{name: message, value: A}]
      - name: B
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: B}]
      - name: C
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: C}]
      - name: D
```

```
        dependencies: [B, C]
        template: echo
        arguments:
          parameters: [{name: message, value: D}]
```

依赖图可以有多个根。 从DAG或步骤模板调用的模板本身可以是DAG或步骤模板。 这可以允许将复杂的工作流程拆分为可管理的部分。

Secrets

Template 支持与Kubernetes Pod规范相同的 Secret 语法和机制，允许访问 Secret 作为环境变量或 Volume mounts。

```
# To run this example, first create the secret by running:
# kubectl create secret generic my-secret --from-literal=mypassword=
S00perS3cretPa55word
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: secret-example-
spec:
  entrypoint: whalesay
  # To access secrets as files, add a volume entry in spec.volumes[]
and
  # then in the container template spec, add a mount using volumeMoun
ts.
  volumes:
  - name: my-secret-vol
    secret:
      secretName: my-secret     # name of an existing k8s secret
  templates:
  - name: whalesay
    container:
      image: alpine:3.7
      command: [sh, -c]
      args: ['
        echo "secret from env: $MYSECRETPASSWORD";
        echo "secret from file: `cat /secret/mountpath/mypassword`"
      ']
      # To access secrets as environment variables, use the k8s
valueFrom and
      # secretKeyRef constructs.
      env:
      - name: MYSECRETPASSWORD  # name of env var
        valueFrom:
          secretKeyRef:
            name: my-secret     # name of an existing k8s secret
            key: mypassword     # 'key' subcomponent of the secret
      volumeMounts:
      - name: my-secret-vol     # mount file containing secret at /
secret/mountpath
        mountPath: "/secret/mountpath"
```

Scripts & Results

通常，我们只需要一个模板来执行工作流规范中指定的脚本。 此示例显示了如何执行此操作：

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
```

```
metadata:
  generateName: scripts-bash-
spec:
  entrypoint: bash-script-example
  templates:
  - name: bash-script-example
    steps:
    - - name: generate
        template: gen-random-int-bash
    - - name: print
        template: print-message
        arguments:
          parameters:
          - name: message
            value: "{{steps.generate.outputs.result}}"  # The result
of the here-script

  - name: gen-random-int-bash
    script:
      image: debian:9.4
      command: [bash]
      source: |                                           # Contents of
the here-script
        cat /dev/urandom | od -N2 -An -i | awk -v f=1 -v r=100 '{
printf "%i\n", f + r * $1 / 65536}'

  - name: gen-random-int-python
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import random
        i = random.randint(1, 100)
        print(i)

  - name: gen-random-int-javascript
    script:
      image: node:9.1-alpine
      command: [node]
      source: |
        var rand = Math.floor(Math.random() * 100);
        console.log(rand);

  - name: print-message
    inputs:
      parameters:
      - name: message
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo result was: {{inputs.parameters.message}}"]
```

Script 关键字允许使用 source 标记指定脚本体。这将创建一个包含脚本主体的临时文件，然后将临时文件的名称作为命令的最终参数传递，该命令应该是执行脚本主体的解释器。

脚本功能的使用还将运行脚本的标准输出分配给名为 Result 的特殊输出参数。 这允许您在其余的工作流规范中使用运行脚本本身的结果。 在此示例中，结果仅由打印消息模板回显。

Output Parameters

Output Parameters 提供了将步骤的结果用作参数而不是使用外部存储的一般机制。 这允许您使用任何类型的步骤的结果，而不仅仅是脚本，用于条件测试，循环和参数。 Output Parameters 与脚本结果的工作方式类似，只是Output Parameters 的值设置为生成文件的内容而不是 stdout 的内容。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: output-parameter-
spec:
  entrypoint: output-parameter
  templates:
  - name: output-parameter
    steps:
    - - name: generate-parameter
        template: whalesay
    - - name: consume-parameter
        template: print-message
        arguments:
          parameters:
          # Pass the hello-param output from the generate-parameter
step as the message input to print-message
          - name: message
            value: "{{steps.generate-parameter.outputs.parameters.
hello-param}}"

  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [sh, -c]
      args: ["echo -n hello world > /tmp/hello_world.txt"]  # generate
 the content of hello_world.txt
    outputs:
      parameters:
      - name: hello-param   # name of output parameter
        valueFrom:
          path: /tmp/hello_world.txt  # set the value of hello-param
to the contents of this hello-world.txt

  - name: print-message
    inputs:
      parameters:
      - name: message
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["{{inputs.parameters.message}}"]
```

DAG 模板使用任务前缀来引用另一个任务，例如，`{{tasks.generate-parameter.outputs.parameters.hello-param}}`。

Loops

在编写Loops工作流时：

- **迭代一组输入（最为常用），如下例所示。**

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: loops-
spec:
  entrypoint: loop-example
  templates:
  - name: loop-example
    steps:
    - - name: print-message
        template: whalesay
        arguments:
          parameters:
          - name: message
            value: "{{item}}"
        withItems:              # invoke whalesay once for each item
 in parallel
        - hello world           # item 1
        - goodbye world         # item 2

  - name: whalesay
    inputs:
      parameters:
      - name: message
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["{{inputs.parameters.message}}"]
```

- **迭代多组项目，如下例所示。**

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: loops-maps-
spec:
  entrypoint: loop-map-example
  templates:
  - name: loop-map-example
    steps:
    - - name: test-linux
        template: cat-os-release
        arguments:
          parameters:
          - name: image
            value: "{{item.image}}"
          - name: tag
            value: "{{item.tag}}"
        withItems:
        - { image: 'debian', tag: '9.1' }       #item set 1
        - { image: 'debian', tag: '8.9' }       #item set 2
        - { image: 'alpine', tag: '3.6' }       #item set 3
        - { image: 'ubuntu', tag: '17.10' }     #item set 4

  - name: cat-os-release
    inputs:
      parameters:
      - name: image
      - name: tag
    container:
```

```
      image: "{{inputs.parameters.image}}:{{inputs.parameters.tag}}"
      command: [cat]
      args: [/etc/os-release]
```

· 将项目列表作为参数传递，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: loops-param-arg-
spec:
  entrypoint: loop-param-arg-example
  arguments:
    parameters:
    - name: os-list                                    # a list of
items
      value: |
        [
          { "image": "debian", "tag": "9.1" },
          { "image": "debian", "tag": "8.9" },
          { "image": "alpine", "tag": "3.6" },
          { "image": "ubuntu", "tag": "17.10" }
        ]

  templates:
  - name: loop-param-arg-example
    inputs:
      parameters:
      - name: os-list
    steps:
    - - name: test-linux
        template: cat-os-release
        arguments:
          parameters:
          - name: image
            value: "{{item.image}}"
          - name: tag
            value: "{{item.tag}}"
        withParam: "{{inputs.parameters.os-list}}"      # parameter
specifies the list to iterate over

  # This template is the same as in the previous example
  - name: cat-os-release
    inputs:
      parameters:
      - name: image
      - name: tag
    container:
      image: "{{inputs.parameters.image}}:{{inputs.parameters.tag}}"
      command: [cat]
      args: [/etc/os-release]
```

· 动态生成要迭代的项目列表，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: loops-param-result-
spec:
  entrypoint: loop-param-result-example
  templates:
  - name: loop-param-result-example
    steps:
```

```
    - - name: generate
        template: gen-number-list
    # Iterate over the list of numbers generated by the generate
step above
    - - name: sleep
        template: sleep-n-sec
        arguments:
          parameters:
          - name: seconds
            value: "{{item}}"
        withParam: "{{steps.generate.outputs.result}}"

  # Generate a list of numbers in JSON format
  - name: gen-number-list
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import json
        import sys
        json.dump([i for i in range(20, 31)], sys.stdout)

  - name: sleep-n-sec
    inputs:
      parameters:
      - name: seconds
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo sleeping for {{inputs.parameters.seconds}}
seconds; sleep {{inputs.parameters.seconds}}; echo done"]
```

Conditionals

我们还支持条件执行，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: coinflip-
spec:
  entrypoint: coinflip
  templates:
  - name: coinflip
    steps:
    # flip a coin
    - - name: flip-coin
        template: flip-coin
    # evaluate the result in parallel
    - - name: heads
        template: heads                 # call heads template if "
heads"
        when: "{{steps.flip-coin.outputs.result}} == heads"
      - name: tails
        template: tails                 # call tails template if "
tails"
        when: "{{steps.flip-coin.outputs.result}} == tails"

  # Return heads or tails based on a random number
  - name: flip-coin
    script:
      image: python:alpine3.6
```

```
      command: [python]
      source: |
        import random
        result = "heads" if random.randint(0,1) == 0 else "tails"
        print(result)

  - name: heads
    container:
      image: alpine:3.6
      command: [sh, -c]
      args: ["echo \"it was heads\""]

  - name: tails
    container:
      image: alpine:3.6
      command: [sh, -c]
      args: ["echo \"it was tails\""]
```

Recursion

模板可以递归地相互调用。 在上述硬币翻转模板的这种变体中，我们继续翻转硬币直到它出现在头部。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: coinflip-recursive-
spec:
  entrypoint: coinflip
  templates:
  - name: coinflip
    steps:
    # flip a coin
    - - name: flip-coin
        template: flip-coin
    # evaluate the result in parallel
    - - name: heads
        template: heads                 # call heads template if "
heads"
        when: "{{steps.flip-coin.outputs.result}} == heads"
      - name: tails                     # keep flipping coins if "
tails"
        template: coinflip
        when: "{{steps.flip-coin.outputs.result}} == tails"

  - name: flip-coin
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import random
        result = "heads" if random.randint(0,1) == 0 else "tails"
        print(result)

  - name: heads
    container:
      image: alpine:3.6
      command: [sh, -c]
```

```
      args: ["echo \"it was heads\""]
```

这是几次硬币翻转的结果，用于比较。

```
ags get coinflip-recursive-tzcb5

STEP                          PODNAME
MESSAGE
 # coinflip-recursive-vhph5
 ├---# flip-coin             coinflip-recursive-vhph5-2123890397
 └-·-# heads                 coinflip-recursive-vhph5-128690560
   └-○ tails

STEP                           PODNAME
MESSAGE
 # coinflip-recursive-tzcb5
 ├---# flip-coin              coinflip-recursive-tzcb5-322836820
 └-·-○ heads
   └-# tails
     ├---# flip-coin          coinflip-recursive-tzcb5-1863890320
     └-·-○ heads
       └-# tails
         ├---# flip-coin      coinflip-recursive-tzcb5-1768147140
         └-·-○ heads
           └-# tails
             ├---# flip-coin  coinflip-recursive-tzcb5-4080411136
             └-·-# heads      coinflip-recursive-tzcb5-4080323273
               └-○ tails
```

在第一次运行中，硬币立即出现在头部，我们停下来；在第二次运行中，硬币上升了三次，然后它终于出现了，我们就停了下来。

Exit handlers

Exit handlers 是在工作流结束时始终执行的模板，无论成功或失败。

Exit handlers 的一些常见用例。

- 工作流程运行后清理
- 发送工作流程状态通知（例如，电子邮件/ Slack）
- 将通过/失败状态发布到webhook结果（例如GitHub构建结果）
- 重新提交或提交其他工作流程

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: exit-handlers-
spec:
  entrypoint: intentional-fail
  onExit: exit-handler                  # invoke exit-hander template
at end of the workflow
  templates:
  # primary workflow template
  - name: intentional-fail
    container:
      image: alpine:latest
      command: [sh, -c]
```

```
      args: ["echo intentional failure; exit 1"]

  # Exit handler templates
  # After the completion of the entrypoint template, the status of the
  # workflow is made available in the global variable {{workflow.
status}}.
  # {{workflow.status}} will be one of: Succeeded, Failed, Error
  - name: exit-handler
    steps:
    - - name: notify
        template: send-email
      - name: celebrate
        template: celebrate
        when: "{{workflow.status}} == Succeeded"
      - name: cry
        template: cry
        when: "{{workflow.status}} != Succeeded"
  - name: send-email
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo send e-mail: {{workflow.name}} {{workflow.status
}}"]
  - name: celebrate
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo hooray!"]
  - name: cry
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo boohoo!"]
```

Timeouts

要限制工作流的已用时间，可以设置变量activeDeadlineSeconds。

```
# To enforce a timeout for a container template, specify a value for
activeDeadlineSeconds.
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: timeouts-
spec:
  entrypoint: sleep
  templates:
  - name: sleep
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo sleeping for 1m; sleep 60; echo done"]
    activeDeadlineSeconds: 10           # terminate container template
 after 10 seconds
```

Volumes

以下示例动态创建卷，然后在两步工作流中使用该卷。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
```

```
metadata:
  generateName: volumes-pvc-
spec:
  entrypoint: volumes-pvc-example
  volumeClaimTemplates:                     # define volume, same syntax
as k8s Pod spec
  - metadata:
      name: workdir                         # name of volume claim
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi                      # Gi => 1024 * 1024 * 1024

  templates:
  - name: volumes-pvc-example
    steps:
    - - name: generate
        template: whalesay
    - - name: print
        template: print-message

  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [sh, -c]
      args: ["echo generating message in volume; cowsay hello world |
tee /mnt/vol/hello_world.txt"]
      # Mount workdir volume at /mnt/vol before invoking docker/
whalesay
      volumeMounts:                         # same syntax as k8s Pod spec
      - name: workdir
        mountPath: /mnt/vol

  - name: print-message
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo getting message from volume; find /mnt/vol; cat /
mnt/vol/hello_world.txt"]
      # Mount workdir volume at /mnt/vol before invoking docker/
whalesay
      volumeMounts:                         # same syntax as k8s Pod spec
      - name: workdir
        mountPath: /mnt/vol
```

卷是将大量数据从工作流中的一个步骤移动到另一个步骤的非常有用的方法。 根据系统的不同，可以从多个步骤同时访问某些卷。

在某些情况下，您希望访问现有卷，而不是动态创建或销毁一个卷。

```
# Define Kubernetes PVC
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-existing-volume
spec:
  accessModes: [ "ReadWriteOnce" ]
  resources:
    requests:
      storage: 1Gi
```

```
---
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: volumes-existing-
spec:
  entrypoint: volumes-existing-example
  volumes:
  # Pass my-existing-volume as an argument to the volumes-existing-
example template
  # Same syntax as k8s Pod spec
  - name: workdir
    persistentVolumeClaim:
      claimName: my-existing-volume

  templates:
  - name: volumes-existing-example
    steps:
    - - name: generate
        template: whalesay
    - - name: print
        template: print-message

  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [sh, -c]
      args: ["echo generating message in volume; cowsay hello world |
tee /mnt/vol/hello_world.txt"]
      volumeMounts:
      - name: workdir
        mountPath: /mnt/vol

  - name: print-message
    container:
      image: alpine:latest
      command: [sh, -c]
      args: ["echo getting message from volume; find /mnt/vol; cat /
mnt/vol/hello_world.txt"]
      volumeMounts:
      - name: workdir
        mountPath: /mnt/vol
```

Daemon Containers

工作流可以启动在后台运行的容器（也称为守护程序容器），同时工作流本身继续执行。 请注意，当工作流退出调用守护程序的模板范围时，将自动销毁守护程序。 守护进程容器可用于启动要测试的服务或用于测试（例如，固定装置）。 我们还发现，在运行大型模拟以将数据库作为用于收集和组织结果的守护进程时，它非常实用。 守护进程与 sidecars 相比的最大优势在于它们的存在可以持续跨越多个步骤甚至整个工作流程。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: daemon-step-
spec:
  entrypoint: daemon-example
  templates:
  - name: daemon-example
```

```
    steps:
    - - name: influx
        template: influxdb              # start an influxdb as a
daemon (see the influxdb template spec below)

    - - name: init-database             # initialize influxdb
        template: influxdb-client
        arguments:
          parameters:
          - name: cmd
            value: curl -XPOST 'http://{{steps.influx.ip}}:8086/query'
 --data-urlencode "q=CREATE DATABASE mydb"

    - - name: producer-1                # add entries to influxdb
        template: influxdb-client
        arguments:
          parameters:
          - name: cmd
            value: for i in $(seq 1 20); do curl -XPOST 'http://{{
steps.influx.ip}}:8086/write?db=mydb' -d "cpu,host=server01,region=
uswest load=$i" ; sleep .5 ; done
      - name: producer-2                # add entries to influxdb
        template: influxdb-client
        arguments:
          parameters:
          - name: cmd
            value: for i in $(seq 1 20); do curl -XPOST 'http://{{
steps.influx.ip}}:8086/write?db=mydb' -d "cpu,host=server02,region=
uswest load=$((RANDOM % 100))" ; sleep .5 ; done
      - name: producer-3                # add entries to influxdb
        template: influxdb-client
        arguments:
          parameters:
          - name: cmd
            value: curl -XPOST 'http://{{steps.influx.ip}}:8086/write?
db=mydb' -d 'cpu,host=server03,region=useast load=15.4'

    - - name: consumer                  # consume intries from
influxdb
        template: influxdb-client
        arguments:
          parameters:
          - name: cmd
            value: curl --silent -G http://{{steps.influx.ip}}:8086
/query?pretty=true --data-urlencode "db=mydb" --data-urlencode "q=
SELECT * FROM cpu"

  - name: influxdb
    daemon: true                        # start influxdb as a daemon
    container:
      image: influxdb:1.2
      restartPolicy: Always             # restart container if it
fails
      readinessProbe:                   # wait for readinessProbe to
succeed
        httpGet:
          path: /ping
          port: 8086

  - name: influxdb-client
    inputs:
      parameters:
      - name: cmd
    container:
```

```
      image: appropriate/curl:latest
      command: ["/bin/sh", "-c"]
      args: ["{{inputs.parameters.cmd}}"]
      resources:
        requests:
          memory: 32Mi
          cpu: 100m
```

DAG模板使用任务前缀来引用另一个任务，例如`{{tasks.influx.ip}}`。

Sidecars

Sidecar是另一个容器，它与主容器在同一个容器中同时执行，在创建多容器容器时很实用。

在本示例中，我们创建了一个 Sidecar 容器，它将 nginx 作为简单的Web服务器运行。 容器出现的顺序是随机的，因此在此示例中，主容器轮询 nginx 容器，直到它准备好为请求提供服务。 在设计多容器系统时，这是一个很好的设计模式：在运行主代码之前，请始终等待所需的任何服务。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: sidecar-nginx-
spec:
  entrypoint: sidecar-nginx-example
  templates:
  - name: sidecar-nginx-example
    container:
      image: appropriate/curl
      command: [sh, -c]
      # Try to read from nginx web server until it comes up
      args: ["until `curl -G 'http://127.0.0.1/' >& /tmp/out`; do echo
 sleep && sleep 1; done && cat /tmp/out"]
    # Create a simple nginx web server
    sidecars:
    - name: nginx
      image: nginx:1.13
```

Kubernetes Resources

在多数情况下，您需要从工作流程管理 Kubernetes 资源。 资源模板允许您创建，删除或更新任何类型的Kubernetes 资源。

```
# in a workflow. The resource template type accepts any k8s manifest
# (including CRDs) and can perform any kubectl action against it (e.g
. create,
# apply, delete, patch).
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: k8s-jobs-
spec:
  entrypoint: pi-tmpl
  templates:
  - name: pi-tmpl
    resource:                   # indicates that this is a resource
template
      action: create            # can be any kubectl action (e.g.
create, delete, apply, patch)
```

```
      # The successCondition and failureCondition are optional
expressions.
      # If failureCondition is true, the step is considered failed.
      # If successCondition is true, the step is considered successful
.
      # They use kubernetes label selection syntax and can be applied
against any field
      # of the resource (not just labels). Multiple AND conditions can
 be represented by comma
      # delimited expressions.
      # For more details: https://kubernetes.io/docs/concepts/overview
/working-with-objects/labels/
      successCondition: status.succeeded > 0
      failureCondition: status.failed > 3
      manifest: |               #put your kubernetes spec here
        apiVersion: batch/v1
        kind: Job
        metadata:
          generateName: pi-job-
        spec:
          template:
            metadata:
              name: pi
            spec:
              containers:
              - name: pi
                image: perl
                command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(
2000)"]
              restartPolicy: Never
          backoffLimit: 4
```

以这种方式创建的资源独立于工作流程。 如果希望在删除工作流时删除资源，则可以将 Kubernetes 垃圾回收与工作流资源一起用作所有者引用。

说明:

更新Kubernetes资源时，资源将接受mergeStrategy属性，该属性的取值为strategy、merge或json。 如果未提供此属性，则默认为strategy属性。 需要注意的是，不能使用策略修补自定义资源，因此必须选择不同的策略。 例如，假设您已定义 CronTab CustomResourceDefinition，并且以下是 CronTab 的实例：

```
apiVersion: "stable.example.com/v1"
kind: CronTab
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

可以使用以下工作流修改此 Crontab。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: k8s-patch-
spec:
  entrypoint: cront-tmpl
  templates:
```

```
  - name: cront-tmpl
    resource:
      action: patch
      mergeStrategy: merge                    # Must be one of [
strategic merge json]
      manifest: |
        apiVersion: "stable.example.com/v1"
        kind: CronTab
        spec:
          cronSpec: "* * * * */10"
          image: my-awesome-cron-image
```

更多资源

- 更多资源展示，请参见https://github.com/argoproj/argo/blob/master/examples/README.md。
- 所有资料示例模板，请参见https://github.com/argoproj/argo/tree/master/examples。

7.3 开启Workflow UI

Workflow 提供了一套 UI 来展示目前工作流的状态，方便查看每个步骤的容器日志，下面为您介绍如何使用 Ingress 暴露UI访问端点。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

操作步骤

1. 执行 htpasswd 命令生成*auth*文件，用于存放用户名密码。

```
$ htpasswd -c auth workflow
New password: <workflow>
New password:
Re-type new password:
Adding password for user workflow
```

2. 执行如下命令，创建secret 来在 Kubernetes 集群中存放此加密文件。

```
$ kubectl create secret generic workflow-basic-auth --from-file=auth
 -n argo
```

3. 创建并拷贝内容到*ingress yaml*文件中，并执行`kubectl apply -f ingress.yaml`命令，创建workflow-ingress路由。

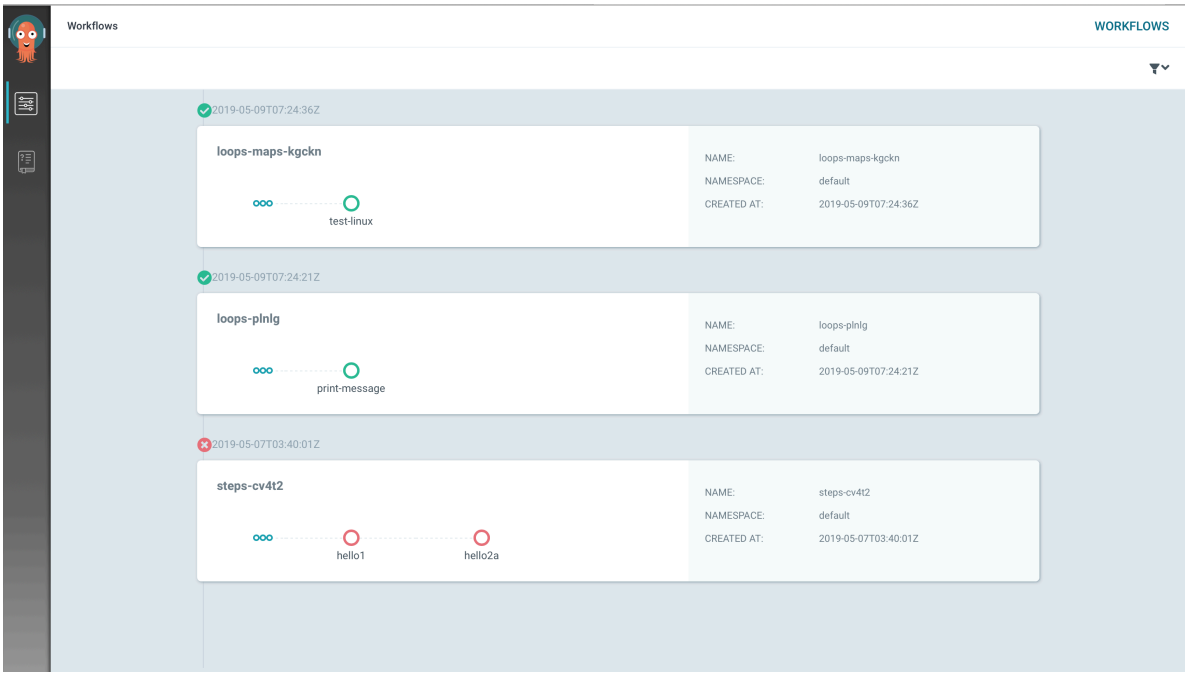
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: workflow-ingress
  namespace: argo
  annotations:
    # type of authentication
```

```
    nginx.ingress.kubernetes.io/auth-type: basic
    # name of the secret that contains the user/password definitions
    nginx.ingress.kubernetes.io/auth-secret: workflow-basic-auth
    # message to display with an appropriate context why the
authentication is required
    nginx.ingress.kubernetes.io/auth-realm: 'Authentication Required
 - workflow'
spec:
  rules:
  - host: workflow.<yourTestHost>
    http:
      paths:
      - path: /
        backend:
          serviceName: argo-ui
          servicePort: 80
```

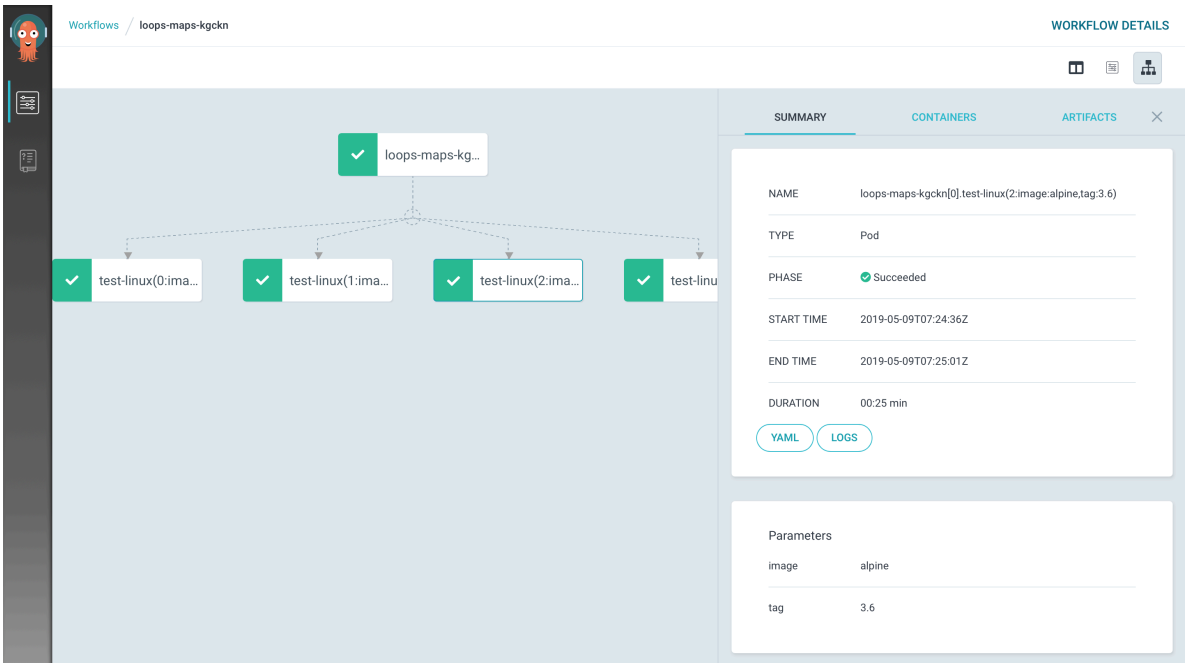
说明:

此处的host需要替换成您对应的集群地址（即为集群信息中的测试域名的值，例如：workflow.cfb131.cn-zhangjiakou.alicontainer.com）。

4. 在浏览器输入workflow.<yourTestHost>，按照提示输入密码就能看到如下界面。



你可以根据需要查看工作流的状态。



7.4 AGS命令行帮助

AGS 是阿里云基因服务的通用命令行工具， 目前主要集成 argo 功能并且适配阿里云各个产品的addon功能命令。

AGS 下载和安装

AGS 默认使用阿里云ak调用各个服务，目前集成了阿里云日志服务用于收集Pod日志。使用此功能请在集群创建的时候选择开启日志服务。

执行如下命令， 下载 AGS 工具并配置运行权限。

```
wget http://ags-hub.oss-cn-hangzhou.aliyuncs.com/ags-linux && chmod +x
 ags-linux && mv ags-linux  /usr/local/bin/ags
```

说明:

`ags config init`根据交互式命令行输入CLI需要的信息，初始化完成后，配置文件会被默认存储到*~/.ags/config*文件中。可以通过`ags config show`展示配置好的信息。其中`AccessKeySecret`会被加密存储。

如果您需要使用日志采集功能，则需要配置ags config。另外为了安全考虑，您可以给CLI单独创建一个ak，赋予日志服务权限即可。

如果您使用的是Kubernetes 托管版集群，您可以#unique_33，并执行如下命令，实现通过 CloudShell 使用 ags 命令行工具。

```
wget http://ags-hub.oss-cn-hangzhou.aliyuncs.com/ags-linux && chmod +x
 ags-linux && mv ags-linux  /usr/local/bin/ags
```

AGS 功能特性

- 完全兼容argo 命令
- Pod被删除后从阿里云日志服务拉取日志查看功能
- 集成kubectl 命令，用户可以直接使用kubectl命令操作集群
- 提供install uninstall 命令，一键安装或者卸载所需资源
- get workflow 命令提供资源使用量查看功能
- Yaml 模板允许下划线等特殊字符
- securityContext 安全支持
- Pod pending/fails 状态与workflow 状态同步
- Yaml定义自动retry功能
- 基于最近失败断点retry整个workflow

- 支持 ECI Serveless Kubernetes 架构

AGS 基本概览

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags
ags is the command line interface to Alibaba Cloud Genomics Compute
Service

Usage:
 ags [flags]
 ags [command]

Available Commands:
 completion  output shell completion code for the specified shell (
bash or zsh)
 config      setup ags client necessary info
 delete      delete a workflow and its associated pods
 get         display details about a workflow
 help        Help about any command
 install     install ags
 kubectl     kubectl command
 lint        validate a file or directory of workflow manifests
 list        list workflows
 logs        view logs of a workflow
 resubmit    resubmit a workflow
 resume      resume a workflow
 retry       retry a workflow
 submit      submit a workflow
 suspend     suspend a workflow
 terminate   terminate a workflow
 uninstall   uninstall ags
 version     Print version information
 wait        waits for a workflow to complete
 watch       watch a workflow until it completes

Flags:
     --as string                      Username to impersonate for the
operation
     --as-group stringArray           Group to impersonate for the
operation, this flag can be repeated to specify multiple groups.
     --certificate-authority string   Path to a cert file for the
certificate authority
     --client-certificate string      Path to a client certificate
file for TLS
     --client-key string              Path to a client key file for
TLS
     --cluster string                 The name of the kubeconfig
cluster to use
     --context string                 The name of the kubeconfig
context to use
 -h, --help                           help for ags
     --insecure-skip-tls-verify       If true, the server's certificat
e will not be checked for validity. This will make your HTTPS
connections insecure
     --kubeconfig string              Path to a kube config. Only
required if out-of-cluster
 -n, --namespace string               If present, the namespace scope
for this CLI request
     --password string                Password for basic authentica
tion to the API server
     --request-timeout string         The length of time to wait
before giving up on a single server request. Non-zero values should
```

```
contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero
means don't timeout requests. (default "0")
     --server string                  The address and port of the
Kubernetes API server
     --token string                   Bearer token for authentication
to the API server
     --user string                    The name of the kubeconfig user
to use
     --username string                Username for basic authentica
tion to the API server

Use "ags [command] --help" for more information about a command.
```

Install/Uninstall

您可以通过`ags install`命令，一键安装AGS所需资源，无需手动安装。

也可以通过`ags uninstall`命令， 一键卸载AGS所有资源。

7.5 AGS帮助示例

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

Log

1. 执行`ags config sls`命令，在AGS上配置并安装日志服务。

   原生Argo查看 Pod 日志只能从本地拉取，当Pod或者所在节点被删除后，日志也会随之丢失，这对查看错误分析以及原因等带来很大问题。 我们将日志上传到阿里云日志服务，即使节点消失也会从 日志服务重新拉取，将日志持久化。

2. 执行`ags logs`命令，查看工作流的日志。

   本例中执行`ags logs POD/WORKFLOW`，查看Pod/Workflow 的日志。

   ```
   [root@iZwz92q9h36kv8posr0i6uZ ~]# ags logs
   view logs of a workflow

   Usage:
    ags logs POD/WORKFLOW [flags]

   Flags:
    -c, --container string    Print the logs of this container (default
    "main")
    -f, --follow              Specify if the logs should be streamed.
    -h, --help                help for logs
    -l, --recent-line int     how many lines to show in one call (
   default 100)
        --since string        Only return logs newer than a relative
   duration like 5s, 2m, or 3h. Defaults to all logs. Only one of since
   -time / since may be used.
   ```

```
      --since-time string   Only return logs after a specific date (
RFC3339). Defaults to all logs. Only one of since-time / since may
be used.
      --tail int            Lines of recent log file to display.
Defaults to -1 with no selector, showing all log lines otherwise 10
, if a selector is provided. (default -1)
      --timestamps          Include timestamps on each line in the
log output
 -w, --workflow            Specify that whole workflow logs should
be printed
```

说明:

- **如果Pod存在本机，AGS 会从本地将Pod日志查询出来，所有flag兼容原argo命令。**
- **如果Pod已经被删除，AGS会从阿里云日志服务来查询日志，默认返回最近100条日志，可以通过`-l flag`来指定到底返回多少条日志。**

集成kubectl 命令

您可以执行如下命令，查看 Pod状态以及其他需要使用kubectl命令的情况。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test-v2
Name:                test-v2
Namespace:           default
ServiceAccount:      default
Status:              Running
Created:             Thu Nov 22 11:06:52 +0800 (2 minutes ago)
Started:             Thu Nov 22 11:06:52 +0800 (2 minutes ago)
Duration:            2 minutes 46 seconds

STEP           PODNAME             DURATION  MESSAGE
● test-v2
└---● bcl2fq  test-v2-2716811808  2m

[root@iZwz92q9h36kv8posr0i6uZ ~]# ags kubectl describe pod test-v2-
2716811808
Name:               test-v2-2716811808
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               cn-shenzhen.i-wz9gwobtqrbjgfnqxl1k/192.168.0.94
Start Time:         Thu, 22 Nov 2018 11:06:52 +0800
Labels:             workflows.argoproj.io/completed=false
                    workflows.argoproj.io/workflow=test-v2
Annotations:        workflows.argoproj.io/node-name=test-v2[0].bcl2fq
                    workflows.argoproj.io/template={"name":"bcl2fq","
inputs":{},"outputs":{},"metadata":{},"container":{"name":"main","
image":"registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2","command":
["sh","-c"],"ar...
Status:             Running
IP:                 172.16.1.152
Controlled By:      Workflow/test-v2
```

通过使用ags kubectl 命令，可以查看到describe pod的状态信息，所有kubectl 原生命令ags均支持。

查看workflow 资源使用量

1. 创建并拷贝内容到*arguments-workflow-resource.yaml*文件中，并执行`ags submit arguments-workflow-resource.yaml`命令，指定resource requests。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: test-resource
spec:
  arguments: {}
  entrypoint: test-resource-
  templates:
  - inputs: {}
    metadata: {}
    name: test-resource-
    outputs: {}
    parallelism: 1
    steps:
    - - arguments: {}
        name: bcl2fq
        template: bcl2fq
  - container:
      args:
      - id > /tmp/yyy;echo `date` > /tmp/aaa;ps -e -o comm,euid,fuid
,ruid,suid,egid,fgid,gid,rgid,sgid,supgid
        > /tmp/ppp;ls -l /tmp/aaa;sleep 100;pwd
      command:
      - sh
      - -c
      image: registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2
      name: main
      resources:                #don't use too much resources
        requests:
          memory: 320Mi
          cpu: 1000m
    inputs: {}
    metadata: {}
    name: bcl2fq
    outputs: {}
```

2. 执行`ags get test456 --show`命令，查看workflow 资源使用。

   本例中，结果显示的是 Pod和test456 使用的核/时。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test456 --show
Name:                test456
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Started:             Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Finished:            Thu Nov 22 14:43:30 +0800 (27 seconds ago)
Duration:            1 minute 41 seconds
Total CPU:           0.02806    (core*hour)
Total Memory:        0.00877    (GB*hour)

STEP           PODNAME             DURATION  MESSAGE  CPU(core*hour)
  MEMORY(GB*hour)
```

```
# test456                                                    0
  0
└---# bcl2fq  test456-4221301428  1m                        0.02806
  0.00877
```

securityContext 安全支持

创建并拷贝内容到*arguments-security-context.yaml*文件中，并执行`ags submit arguments-security-context.yaml`**命令，绑定对应的 psp来进行权限控制。**

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: test
spec:
  arguments: {}
  entrypoint: test-security-
  templates:
  - inputs: {}
    metadata: {}
    name: test-security-
    outputs: {}
    parallelism: 1
    steps:
    - - arguments: {}
        name: bcl2fq
        template: bcl2fq
  - container:
      args:
      - id > /tmp/yyy;echo `date` > /tmp/aaa;ps -e -o comm,euid,fuid,
ruid,suid,egid,fgid,gid,rgid,sgid,supgid
        > /tmp/ppp;ls -l /tmp/aaa;sleep 100;pwd
      command:
      - sh
      - -c
      image: registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2
      name: main
      resources:                #don't use too much resources
        requests:
          memory: 320Mi
          cpu: 1000m
    inputs: {}
    metadata: {}
    name: bcl2fq
    outputs: {}
    securityContext:
      runAsUser: 800
```

Yaml定义自动retry功能

有时一些bash命令会由于不明原因失败，重试就可以解决，AGS 提供一种基于yaml配置的自动重启机制，当Pod内命令运行失败后，会自动拉起重试，并且可以设置重试次数。

创建并拷贝内容到*arguments-auto-retry.yaml*文件中，并执行`ags submit arguments-auto-retry.yaml`**命令，配置Workflow的自动重启机制。**

```
# This example demonstrates the use of retries for a single container.
apiVersion: argoproj.io/v1alpha1
```

```
kind: Workflow
metadata:
  generateName: retry-container-
spec:
  entrypoint: retry-container
  templates:
  - name: retry-container
    retryStrategy:
      limit: 10
    container:
      image: python:alpine3.6
      command: ["python", -c]
      # fail with a 66% probability
      args: ["import random; import sys; exit_code = random.choice([0
, 1, 1]); sys.exit(exit_code)"]
```

基于最近失败断点retry整个workflow

在整个workfow运行中，有时候会出现某个step失败，这时候希望从某个失败的节点重新retry workflow，类似断点续传的断点重试功能。

1. 执行`ags get test456 --show`命令，查看Workflow test456 从哪个step断点。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test456 --show
Name:                test456
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Started:             Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Finished:            Thu Nov 22 14:43:30 +0800 (27 seconds ago)
Duration:            1 minute 41 seconds
Total CPU:           0.0572   (core*hour)
Total Memory:        0.01754    (GB*hour)

STEP           PODNAME              DURATION  MESSAGE  CPU(core*hour)
  MEMORY(GB*hour)
 # test456                                              0
  0
 └---# bcl2fq  test456-4221301428  1m                   0.02806
  0.00877
 └---X bcl2fq  test456-4221301238  1m                   0.02806
  0.00877
```

2. 执行`ags retry test456`命令，从最近失败断点处继续retry workflow test456。

使用 ECI 运行Workflow

ECI操作请参见弹性容器实例ECI。

配置使用 ECI 前，请先安装AGS，请参见#unique_32/unique_32_Connect_42_section_bs7_2zj_w60。

1. 执行`kubectl get cm -n argo`命令，获取Workflow对应的yaml文件名称。

```
[root@iZwz9f4ofes6kbo01ipuf1Z ~]# kubectl get cm -n argo
NAME                            DATA      AGE
```

```
workflow-controller-configmap   1         4d
```

2. 执行`kubectl get cm -n argo workflow-controller-configmap -o yaml`命令，打开*workflow-controller-configmap.yaml*文件，并使用如下内容覆盖当前yaml文件的内容。

```
apiVersion: v1
data:
  config: |
    containerRuntimeExecutor: k8sapi
kind: ConfigMap
```

3. 执行`kubectl delete pod <podName>`命令，重启 argo controller。

说明:

这里的*podName*为Workflow所在的Pod的名称。

4. 创建并拷贝内容到*arguments-workflow-eci.yaml*文件中，并执行`ags submit arguments-workflow-eci.yaml`命令，在 ECI 上运行的容器添加nodeSelector和Tolerations这两个标识。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay
      command: [env]
      #args: ["hello world"]
      resources:
        limits:
          memory: 32Mi
          cpu: 100m
    nodeSelector:                # 添加nodeSelector
      type: virtual-kubelet
    tolerations:                 # 添加tolerations
    - key: virtual-kubelet.io/provider
      operator: Exists
    - key: alibabacloud.com
      effect: NoSchedule
```

查看Workflow 实际资源使用量以及峰值

ags workflow controller 会通过metrics-server 自动获取Pod 每分钟的实际资源使用量，并且统计出来总量和各个Pod的峰值使用量。

执行`ags get steps-jr6tw --metrics`命令，查看Workflow 实际资源使用量以及峰值。

```
#  ags get steps-jr6tw --metrics
Name:                steps-jr6tw
```

```
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Tue Apr 16 16:52:36 +0800 (21 hours ago)
Started:             Tue Apr 16 16:52:36 +0800 (21 hours ago)
Finished:            Tue Apr 16 19:39:18 +0800 (18 hours ago)
Duration:            2 hours 46 minutes
Total CPU:           0.00275    (core*hour)
Total Memory:        0.04528    (GB*hour)

STEP            PODNAME                 DURATION  MESSAGE  CPU(core*
hour)  MEMORY(GB*hour)  MaxCpu(core)  MaxMemory(GB)
 # steps-jr6tw                                              0
     0                0             0
 └---# hello1   steps-jr6tw-2987978173  2h                 0.00275
     0.04528          0.000005      0.00028
```

设置Workflow 优先级

当前面有一些任务正在运行时，有一个紧急任务急需运行，此时，您可以给Workflow 设置高、中、低的优先级，高优先级抢占低优先级任务的资源。

- 您可以给某个Pod设置高优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-taskA.yaml*文件中，并执行`ags submit arguments-high-priority-taskA.yaml`命令，给任务A设置高优先级。

  ```
  apiVersion: scheduling.k8s.io/v1beta1
  kind: PriorityClass
  metadata:
    name: high-priority
  value: 1000000
  globalDefault: false
  description: "This priority class should be used for XYZ service
  pods only."
  ```

- 您可以给某个Pod设置中优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-taskB.yaml*文件中，并执行`ags submit arguments-high-priority-taskB.yaml`命令，给任务B设置中优先级。

  ```
  apiVersion: scheduling.k8s.io/v1beta1
  kind: PriorityClass
  metadata:
    name: medium-priority
  value: 100
  globalDefault: false
  ```

```
description: "This priority class should be used for XYZ service
pods only."
```

- 您也可以一个Workflow设置高优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-Workflow.yaml*文件中，并执行`ags submit arguments-high-priority-Workflow.yaml`命令，给Workflow中所有的Pod设置高优先级。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow                  # new type of k8s spec
metadata:
  generateName: high-proty-   # name of the workflow spec
spec:
  entrypoint: whalesay          # invoke the whalesay template
  podPriorityClassName: high-priority # workflow level priority
  templates:
  - name: whalesay              # name of the template
    container:
      image: ubuntu
      command: ["/bin/bash", "-c", "sleep 1000"]
      resources:
        requests:
          cpu: 3
```

下面以一个Workflow里面含有两个Pod，分别给一个 Pod 设置中优先级，另一个 Pod 设置高优先级，此时，高优先级的 Pod 就能抢占低优先级 Pod的资源。

1. 创建并拷贝内容到*arguments-high-priority-steps.yaml*文件中，并执行`ags submit arguments-high-priority-steps.yaml`命令，给Pod 设置优先级。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello

  templates:
  - name: hello-hello-hello
    steps:
    - - name: low
        template: low
    - - name: low-2
        template: low
      - name: high
        template: high

  - name: low
    container:
      image: ubuntu
      command: ["/bin/bash", "-c", "sleep 30"]
      resources:
        requests:
          cpu: 3

  - name: high
```

```
    priorityClassName: high-priority # step level priority
    container:
      image: ubuntu
      command: ["/bin/bash", "-c", "sleep 30"]
      resources:
        requests:
          cpu: 3
```

2. 运行结果为高优先级 Pod 会抢占并删除老的Pod。执行结果如下：

```
Name:                steps-sxvrv
Namespace:           default
ServiceAccount:      default
Status:              Failed
Message:             child 'steps-sxvrv-1724235106' failed
Created:             Wed Apr 17 15:06:16 +0800 (1 minute ago)
Started:             Wed Apr 17 15:06:16 +0800 (1 minute ago)
Finished:            Wed Apr 17 15:07:34 +0800 (now)
Duration:            1 minute 18 seconds

STEP            PODNAME                  DURATION  MESSAGE
 # steps-sxvrv                                     child 'steps-sxvrv
-1724235106' failed
 ├---# low      steps-sxvrv-3117418100  33s
 └-·-# high     steps-sxvrv-603461277   45s
   └-⚠ low-2    steps-sxvrv-1724235106  45s       pod deleted
```

说明：

这里高优先级任务会自动抢占低优先Pod所占用资源，会停止低优先级任务，中断正在运行的进程，所以使用时要非常谨慎。

Workflow Filter

在ags get workflow 中，针对较大的Workflow 可以使用 filter 列出指定状态的Pod。

1. 执行`ags get <pod##> --status Running`命令，列出指定状态的Pod。

```
[root@iZ8vb19036cvgu1tpxkzl1Z workflow-prioty]# ags get pod-limits-
n262v --status Running
Name:                pod-limits-n262v
Namespace:           default
ServiceAccount:      default
Status:              Running
Created:             Wed Apr 17 15:59:08 +0800 (1 minute ago)
Started:             Wed Apr 17 15:59:08 +0800 (1 minute ago)
Duration:            1 minute 17 seconds
Parameters:
  limit:             300

STEP                    PODNAME                        DURATION
MESSAGE
 ● pod-limits-n262v
   ├-● run-pod(13:13)  pod-limits-n262v-3643890604  1m
   ├-● run-pod(14:14)  pod-limits-n262v-4115394302  1m
   ├-● run-pod(16:16)  pod-limits-n262v-3924248206  1m
   ├-● run-pod(17:17)  pod-limits-n262v-3426515460  1m
   ├-● run-pod(18:18)  pod-limits-n262v-824163662   1m
   ├-● run-pod(20:20)  pod-limits-n262v-4224161940  1m
```

```
    ├─● run-pod(22:22)  pod-limits-n262v-1343920348  1m
    ├─● run-pod(2:2)    pod-limits-n262v-3426502220  1m
    ├─● run-pod(32:32)  pod-limits-n262v-2723363986  1m
    ├─● run-pod(34:34)  pod-limits-n262v-2453142434  1m
    ├─● run-pod(37:37)  pod-limits-n262v-3225742176  1m
    ├─● run-pod(3:3)    pod-limits-n262v-2455811176  1m
    ├─● run-pod(40:40)  pod-limits-n262v-2302085188  1m
    ├─● run-pod(6:6)    pod-limits-n262v-1370561340  1m
```

2. 执行ags get <*pod##*> --sum-info命令，统计当前Pod 状态信息。

```
[root@iZ8vb19036cvgu1tpxkzl1Z workflow-prioty]# ags get pod-limits-
n262v --sum-info --status Error
Name:                pod-limits-n262v
Namespace:           default
ServiceAccount:      default
Status:              Running
Created:             Wed Apr 17 15:59:08 +0800 (2 minutes ago)
Started:             Wed Apr 17 15:59:08 +0800 (2 minutes ago)
Duration:            2 minutes 6 seconds
Pending:             198
Running:             47
Succeeded:           55
Parameters:
  limit:             300

STEP                 PODNAME  DURATION  MESSAGE
 ● pod-limits-n262v
```

敏捷版Autoscaler 使用流程

在敏捷版使用autoscaler 需要用户提前创建或者已经具备以下资源：

- 您已经有一个VPC。
- 您已经有一个vSwitch。
- 您已经设置好一个安全组。
- 您已经获取到敏捷版的APIServer 内网地址。
- 您明确扩容节点的规格。
- 您已创建好一个ECS实例且拥有公网访问能力。

您可以在AGS命令行，按照界面提示进行如下操作：

```
$ ags config autoscaler    根据提示输入对应的值
Please input vswitchs with comma separated
vsw-hp3cq3fnv47bpz7x58wfe
Please input security group id
sg-hp30vp05x6tlx13my0qu
Please input the instanceTypes with comma separated
ecs.c5.xlarge
Please input the new ecs ssh password
xxxxxxxx
Please input k8s cluster APIServer address like(192.168.1.100)
172.24.61.156
Please input the autoscaling mode (current: release. Type enter to
skip.)
Please input the min size of group (current: 0. Type enter to skip.)
```

```
Please input the max size of group (current: 1000. Type enter to skip
.)
Create scaling group successfully.
Create scaling group config successfully.
Enable scaling group successfully.
Succeed
```

配置完成后，登录弹性伸缩控制台， 可以看到创建好的自动伸缩组。

配置使用 ags configmap

本例中，默认使用 hostNetwork。

1. 执行`kubectl get cm -n argo`命令，获取 Workflow 对应的yaml文件名称。

```
[root@iZ8vb19036cvgu1tpxkzl1Z ~]# kubectl get cm -n argo
NAME                           DATA   AGE
workflow-controller-configmap   1      6d23h
```

2. 执行`kubectl edit cm workflow-controller-configmap -n argo`命令，打开*workflow-controller-configmap.yaml*文件，将如下内容填入当前yaml文件中。

```
data:
  config: |
    extraConfig:
      enableHostNetwork: true
      defaultDnsPolicy: Default
```

填入完成后，*workflow-controller-configmap.yaml*全文如下：

```
apiVersion: v1
data:
  config: |
    extraConfig:
      enableHostNetwork: true
      defaultDnsPolicy: Default
kind: ConfigMap
metadata:
  name: workflow-controller-configmap
  namespace: argo
```

3. 配置完成后，新部署的Workflow 均会默认使用`hostNetwork`，且`dnsPolicy`为`Default`。
4. （可选）如果配置了psp，需要在psp中对应的yaml文件增加如下内容。

```
hostNetwork: true
```

说明:

如果该yaml文件中已有`hostNetwork`参数，需要将值改为`ture`。

完整的yaml文件示例点这里。

8 网络管理

8.1 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。

网络能力

Service

通常情况下，直接访问Pod会有如下几个问题：

- Pod会随时被Deployment这样的控制器删除重建，那访问Pod的结果就会变得不可预知。
- Pod的IP地址是在Pod启动后才被分配，在启动前并不知道Pod的IP地址。
- 应用往往都是由多个运行相同镜像的一组Pod组成，一个个Pod的访问也变得不现实。

Kubernetes中的Service对象就是用来解决上述Pod访问问题的。Service有一个固定IP地址，Service将访问他的流量转发给Pod，具体转发给哪些Pod通过Label来选择，而且Service可以给这些Pod做负载均衡。详细请参见创建服务。

Ingress

通常情况下，service 和 pod 的 IP 仅可在集群内部访问。集群外部的请求需要通过负载均衡转发到 service 在 Node 上暴露的 NodePort 上，然后再由 kube-proxy 通过边缘路由器 (edge router) 将其转发给相关的 Pod 或者丢弃，而 Ingress 就是为进入集群的请求提供路由规则的集合。

Ingress 可以给 service 提供集群外部访问的 URL、负载均衡、SSL 终止、HTTP 路由等。为了配置这些 Ingress 规则，集群管理员需要部署一个 Ingress controller，它监听 Ingress 和 service 的变化，并根据规则配置负载均衡并提供访问入口。

Ingress 的组成部分

- Nginx：实现负载均衡到pod的集合。
- Ingress Controller：从集群api获取services对应pod的ip到nginx配置文件中。
- Ingress：为nginx创建虚拟主机。

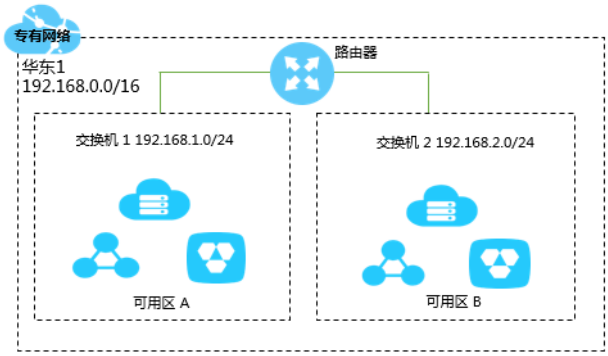
Ingress的创建与管理请参见通过界面创建路由（Ingress）。

网络底层基础设施

VPC

专有网络（Virtual Private Cloud）是基于阿里云创建的自定义私有网络，不同的专有网络之间彻底逻辑隔离。可以在专有网络内创建和管理云产品实例，例如，云服务器、云数据库RDS版和负载均衡等。

每个VPC都由一个私网网段、一个路由器和至少一个交换机组成。



SLB

负载均衡（Server Load Balancer）通过设置虚拟服务地址，将添加的ECS实例虚拟成一个高性能、高可用的应用服务池，并根据转发规则，将来自客户端的请求分发给云服务器池中的ECS实例。

负载均衡默认检查云服务器池中的ECS实例的健康状态，自动隔离异常状态的ECS实例，消除了单台ECS实例的单点故障，提高了应用的整体服务能力。此外，负载均衡还具备抗DDoS攻击的能力，增强了应用服务的防护能力。

负载均衡由以下三个部分组成：

- 负载均衡实例

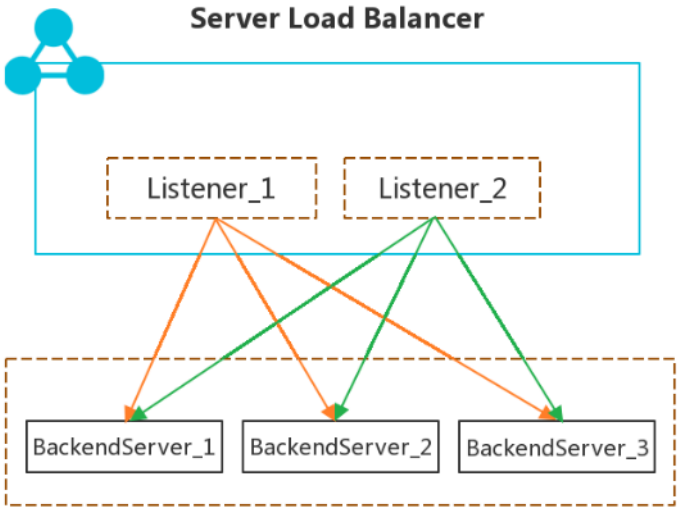
  一个负载均衡实例是一个运行的负载均衡服务，用来接收流量并将其分配给后端服务器。要使用负载均衡服务，您必须创建一个负载均衡实例，并至少添加一个监听和两台ECS实例。

- 监听

  监听用来检查客户端请求并将请求转发给后端服务器。监听也会对后端服务器进行健康检查。

- 后端服务器

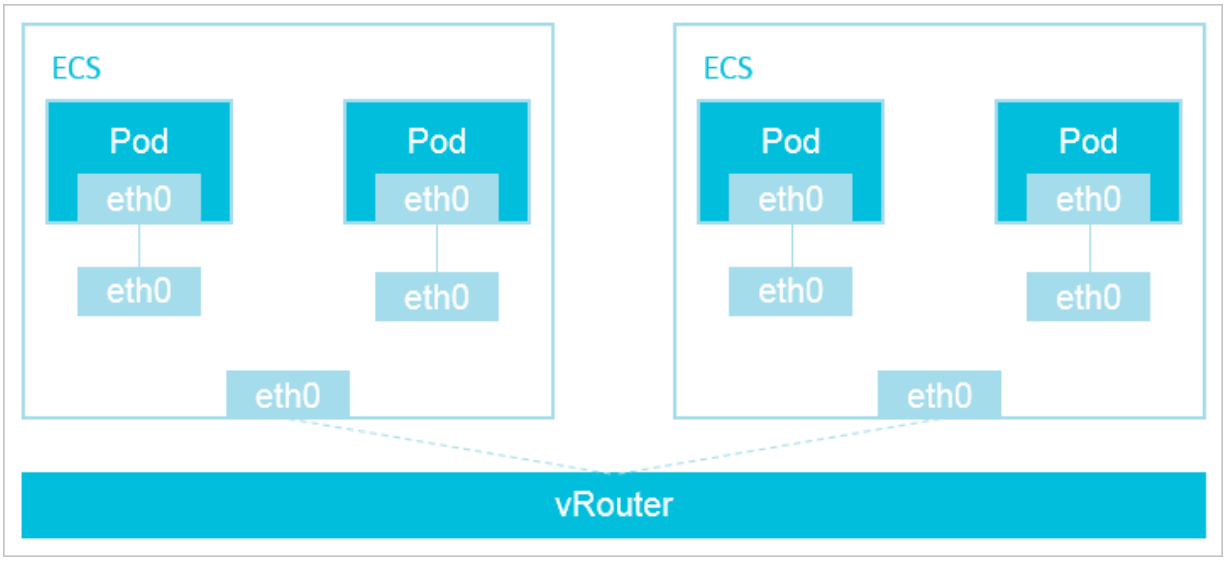
  一组接收前端请求的ECS实例。您可以单独添加ECS实例到服务器池，也可以通过虚拟服务器组或主备服务器组来批量添加和管理。



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。

网络策略

网络策略（NetworkPolicy）是一种关于pod间及pod与其他网络端点间所允许的通信规则的规范。

NetworkPolicy 资源使用标签选择pod，并定义选定pod所允许的通信规则。详细请参见使用网络策略（Network Policy）。

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，系统默认不再为该SLB处理监听，用户可以通过设置`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners: "true"`参数来显示启用监听配置，或者手动配置该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 pod -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
```

注意事项

- Cloud Controller Manager（简称CCM）会为Type=LoadBalancer类型的Service创建或配置阿里云负载均衡（SLB），包含SLB、监听、虚拟服务器组等资源。
- 对于非LoadBalancer类型的service则不会为其配置负载均衡，这包含如下场景：当用户将Type=LoadBalancer的service变更为Type!=LoadBalancer时，CCM也会删除其原先为该Service创建的SLB（用户通过service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id指定的已有SLB除外）。
- 自动刷新配置：CCM使用声明式API，会在一定条件下自动根据service的配置刷新阿里云负载均衡配置，所有用户自行在SLB控制台上修改的配置均存在被覆盖的风险（使用已有SLB同时不覆盖监听的场景除外），因此不能在SLB控制台手动修改Kubernetes创建并维护的SLB的任何配置，否则有配置丢失的风险。
- 同时支持为serivce指定一个已有的负载均衡，或者让CCM自行创建新的负载均衡。但两种方式在SLB的管理方面存在一些差异：

  指定已有SLB

  - 需要为Service设置annotation：service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id 。
  - SLB配置：此时CCM会使用该SLB做为Service的SLB，并根据其他annotation配置SLB，并且自动的为SLB创建多个虚拟服务器组（当集群节点变化的时候，也会同步更新虚拟服务器组里面的节点）。
  - 监听配置：是否配置监听取决于service.beta.kubernetes.io/alibaba-cloud-loadbalancer-force-override-listeners：是否设置为true。 如果设置为false，那么CCM不会为SLB管理任何监听配置。如果设置为true，那么CCM会尝试为SLB更新监听，此时CCM会根据监听名称判断SLB上的监听是否为k8s维护的监听（名字的格式为*k8s/Port/ServiceName/Namespace/ClusterID*），若Service声明的监听与用户自己管理的监听端口冲突，那么CCM会报错。
  - SLB的删除： 当Service删除的时候CCM不会删除用户通过id指定的已有SLB。
- CCM管理的SLB
  - CCM会根据service的配置自动的创建配置SLB、监听、虚拟服务器组等资源，所有资源归CCM管理，因此用户不得手动在SLB控制台更改以上资源的配置，否则CCM在下次Reconcile的时候将配置刷回service所声明的配置，造成非用户预期的结果。
  - SLB的删除：当Service删除的时候CCM会删除该SLB。

- 后端服务器更新
  - CCM会自动的为该Service对应的SLB刷新后端虚拟服务器组。当Service对应的后端Endpoint发生变化的时候或者集群节点变化的时候都会自动的更新SLB的后端Server。
  - `spec.externalTrafficPolicy = Cluster`模式的Service，CCM默认会将所有节点挂载到SLB的后端（使用BackendLabel标签配置后端的除外）。由于SLB限制了每个ECS上能够attach的SLB的个数（quota），因此这种方式会快速的消耗该quota，当quota耗尽后，会造成Service Reconcile失败。解决的办法，可以使用Local模式的Service。
  - `spec.externalTrafficPolicy = Local`模式的Service，CCM默认只会讲Service对应的Pod所在的节点加入到SLB后端。这会明显降低quota的消耗速度。同时支持四层源IP保留。
  - 任何情况下CCM不会将Master节点作为SLB的后端。
  - CCM默认不会从SLB后端摘除被kubectl drain/cordon的节点。如需移除节点，请设置`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-unscheduled-backend`为on。

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.XX.XX    101.37.XX.XX     80:31891/TCP
                  4s
```

3. 在浏览器中访问 `http://101.37.XX.XX`，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. 执行如下命令，创建一个 Nginx 应用。

```
kubectl apply -f nginx-svc.yml
```

3. 执行如下命令，向外网用户暴露 Nginx 服务。

```
root@master # kubectl get service
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(
S)         AGE9d
ngi-01nx        LoadBalancer   172.19.XX.XX    101.37.XX.XX     80:
32325/TCP   3h
```

4. 在浏览器中访问 `http://101.37.XX.XX`，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。

   此时，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.XX.XX:80`。

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。
2. 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。
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。
7. 或单击左侧导航栏路由与负载均衡 > 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。

- **创建一个公网类型的负载均衡**

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

- **创建一个私网类型的负载均衡**

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

- **创建HTTP类型的负载均衡**

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-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/alibaba-cloud-loadbalancer-protocol-
port: "https:443"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id:
 "${YOUR_CERT_ID}"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  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#unique_147。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-
type: "paybybandwidth"
    service.beta.kubernetes.io/alibaba-cloud-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**

  负载均衡规格可参见#unique_148。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-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/alibaba-cloud-loadbalancer-force-override-listeners`为ture

    说明:

    复用已有的负载均衡默认不覆盖已有监听，因为以下两点原因：

    ■ 如果已有负载均衡的监听上绑定了业务，强制覆盖可能会引发业务中断。

    ■ 由于CCM目前支持的后端配置有限，无法处理一些复杂配置。如果有复杂的后端配置需求，可以在不覆盖监听的情况下，通过控制台自行配置监听。

    如存在以上两种情况不建议强制覆盖监听，如果已有负载均衡的监听端口不再使用，则可以强制覆盖。

  - 使用已有的负载均衡暂不支持添加额外标签（`annotation: service.beta.kubernetes.io/alibaba-cloud-loadbalancer-additional-resource-tags`）

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: "${
YOUR_LOADBALACER_ID}"
  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，并强制覆盖已有监听

  强制覆盖已有监听，如果监听端口冲突，则会删除已有监听。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id: "${
YOUR_LOADBALACER_ID}"
    service.beta.kubernetes.io/alibaba-cloud-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/alibaba-cloud-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**

  - **参数`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-persistence-time`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/alibaba-cloud-loadbalancer-persistenc
e-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。
  - 配置insert cookie，以下四项annotation必选。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-
session: "on"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-
session-type: "insert"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie-
timeout: "1800"
    service.beta.kubernetes.io/alibaba-cloud-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。
  - 配置server cookie，以下四项annotation必选。
  - cookie名称（service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie）只能包含字母、数字、‘_’和‘-’。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-
session: "on"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-
session-type: "server"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie:
 "${YOUR_COOKIE}"
    service.beta.kubernetes.io/alibaba-cloud-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，例如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/alibaba-cloud-loadbalancer-master-
zoneid: "ap-southeast-5a"
    service.beta.kubernetes.io/alibaba-cloud-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**

  默认`externalTrafficPolicy`为Cluster模式，会将集群中所有节点挂载到后端服务器。Local模式仅将Pod所在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
```

- 创建私有网络类型（VPC）的负载均衡
  - 创建私有网络类型的负载均衡，以下两个annotation必选。
  - 私网负载均衡支持专有网络（VPC）和经典网络（Classic），两者区别参见#unique_149。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-
type: "intranet"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-network-
type: "vpc"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

- 创建按流量付费的负载均衡
  - 仅支持公网类型的负载均衡实例
  - 以下两项annotation必选

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-bandwidth:
 "45"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-
type: "paybybandwidth"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
```

```
  type: LoadBalancer
```

- **创建带健康检查的负载均衡**
  - **设置TCP类型的健康检查**
    - **TCP端口默认开启健康检查，且不支持修改，即**`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-flag annotation`**无效。**
    - **设置TCP类型的健康检查，以下所有annotation必选。**

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-type: "tcp"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-connect-timeout: "8"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-healthy-
threshold: "4"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-
unhealthy-threshold: "4"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-interval: "3"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

  - **设置HTTP类型的健康检查**

    设置HTTP类型的健康检查，以下所有的annotation必选。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-flag: "on"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-type: "http"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-uri: "/test/index.html"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-healthy-
threshold: "4"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-
unhealthy-threshold: "4"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-timeout: "10"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-
check-interval: "3"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol
-port: "http:80"
  name: nginx
```

```
    namespace: default
  spec:
    ports:
    - port: 80
      protocol: TCP
      targetPort: 80
    selector:
      run: nginx
    type: LoadBalancer
```

- 为负载均衡设置调度算法
  - wrr（默认值）：权重值越高的后端服务器，被轮询到的次数（概率）也越高。
  - wlc：除了根据每台后端服务器设定的权重值来进行轮询，同时还考虑后端服务器的实际负载（即连接数）。当权重值相同时，当前连接数越小的后端服务器被轮询到的次数（概率）也越高。
  - rr：按照访问顺序依次将外部请求依序分发到后端服务器。

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

- 创建带有访问控制的负载均衡
  - 需要先在阿里云控制台上创建一个访问控制并记录acl-id，然后使用如下 annotation 创建一个带有访问控制的负载均衡实例。
  - 白名单适合只允许特定IP访问的场景，black黑名单适用于只限制某些特定IP访问的场景。
  - 创建带有访问控制的负载均衡，以下三项annotation必选。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-status
: "on"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-id:
 "${YOUR_ACL_ID}"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-type:
 "white"
  name: nginx
  namespace: default
spec:
  ports:
```

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

- 为负载均衡指定虚拟交换机
  - 通过阿里云专有网络控制台查询交换机ID，然后使用如下的annotation为负载均衡实例指定虚拟交换机。
  - 为负载均衡指定虚拟交换机，以下两项annotation必选。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
   service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-
type: "intranet"
   service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vswitch-id:
 "${YOUR_VSWITCH_ID}"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

- 为负载均衡指定转发端口
  - 端口转发是指将http端口的请求转发到https端口上。
  - 设置端口转发需要先在阿里云控制台上创建一个证书并记录cert-id。
  - 如需设置端口转发，以下三项annotation必选。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-
port: "https:443,http:80"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id:
 "${YOUR_CERT_ID}"
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-forward-
port: "80:443"
  name: nginx
  namespace: default
spec:
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 443
  - name: http
    port: 80
```

```
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

- **为负载均衡添加额外标签**

  多个tag以逗号分隔，例如`"k1=v1,k2=v2"`。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-additional
-resource-tags: "Key1=Value1,Key2=Value2"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: LoadBalancer
```

- **移除SLB后端unscheduleable状态的节点**

  `kubectl cordon`**与**`kubectl drain`**命令会将节点置为unscheduleable状态，默认**`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-unscheduled-backend`**的取值为off，此时不会将处于unscheduleable状态的节点从SLB的后端服务器组移除。若需要从SLB的后端服务器组移除unscheduleable状态的节点，请将**`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-unscheduled-backend`**的的取值设置为on**。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-
unscheduled-backend: "on"
  name: nginx
spec:
  ports:
  - name: http
    port: 30080
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

```
  type: LoadBalancer
```

- **直接将Pod挂载到SLB后端**

  支持在Terway 网络模式下，通过`annotation: service.beta.kubernetes.io/backend-type: "eni"`**将Pod直接挂载到SLB后端，提升网络转发性能。**

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/backend-type: "eni"
  name: nginx
spec:
  ports:
  - name: http
    port: 30080
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

说明:

- **注释的内容是区分大小写的。**
- **自2019年9月11日起，annotation字段alicloud更新为alibaba-cloud。**

  例如:

  更新前: `service.beta.kubernetes.io/alicloud-loadbalancer-id`

  更新后: `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id`

  系统将继续兼容alicloud的写法，用户无需做任何修改，敬请注意。

注释	类型	描述	默认值
`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port`	string	多个值之间由逗号分隔，例如: `https:443,http:80`	无
`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type`	string	取值可以是`internet`或者`intranet`	`internet`

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-slb-network-type	string	负载均衡的网络类型，取值可以是classic或者vpc 取值为vpc时，需设置service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type为intranet。	classic
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-charge-type	string	取值可以是paybytraffic或者paybybandwidth	paybytraffic
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id	string	负载均衡实例的 ID。通过 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-id 指定您已有的SLB，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/alibaba-cloud-loadbalancer-force-override-listeners为true。	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-backend-label	string	通过 label 指定 SLB 后端挂载哪些worker节点。	无

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec	string	负载均衡实例的规格。可参见：#unique_150	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-persistence-timeout	string	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。 可参见：#unique_151	0
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session	string	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。 可参见：#unique_152和#unique_153	off

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session-type	string	cookie的处理方式。 取值： · insert：植入Cookie。 · server：重写Cookie。 说明： · 仅对HTTP和HTTPS协议的监听生效。 · 当service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session取值为on时，该参数必选。 可参见：#unique_152和#unique_153	无

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie-timeout	string	Cookie超时时间。取值：1-86400（秒） 说明：当service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session为on且service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session-type为insert时，该参数必选。 可参见：#unique_152和#unique_153	无

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cookie	string	服务器上配置的Cookie名称。 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/alibaba-cloud-loadbalancer-sticky-session为on且service.beta.kubernetes.io/alibaba-cloud-loadbalancer-sticky-session-type为server时，该参数必选。 可参见：#unique_152和#unique_153	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-master-zoneid	string	主后端服务器的可用区ID。	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-slave-zoneid	string	备后端服务器的可用区ID。	无

注释	类型	描述	默认值
externalTrafficPolicy	string	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/alibaba-cloud-loadbalancer-force-override-listeners	string	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/alibaba-cloud-loadbalancer-bandwidth	string	负载均衡的带宽，仅适用于公网类型的负载均衡。	50
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-cert-id	string	阿里云上的证书 ID。您需要先上传证书	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-flag	string	取值是on | off · TCP监听默认为on且不可更改。 · HTTP监听默认为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/alibaba-cloud-loadbalancer-health-check-type	string	健康检查类型，取值：tcp | http。 可参见：#unique_151	tcp

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-uri	string	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。 可参见：#unique_151	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-connect-port	string	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。 可参见：#unique_151	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-healthy-threshold	string	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 可参见：#unique_151	3
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-unhealthy-threshold	string	健康检查连续失败多少次后，将后端服务器的健康检查状态由success判定为fail。取值： 2-10 可参见：#unique_151	3

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval	string	健康检查的时间间隔。 取值：1-50（秒） 可参见：#unique_151	2

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-connect-timeout	string	接收来自运行状况检查的响应需要等待的时间，适用于TCP模式。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/alibaba-cloud-loadbalancer-health-check-connect-timeout的值小于service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval的值，则service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-connect-timeout无效，超时时间为service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval的值。 可参见：#unique_151	5

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-timeout	string	接收来自运行状况检查的响应需要等待的时间，适用于HTTP模式。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/alibaba-cloud-loadbalancer-health-check-timeout的值小于 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval的值，则 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-timeout 无效，超时时间为 service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-interval的值。 可参见：#unique_151	5

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-domain	string	用于健康检查的域名。 · $_ip：后端服务器的私网IP。当指定了IP或该参数未指定时，负载均衡会使用各后端服务器的私网IP当做健康检查使用的域名。 · domain：域名长度为1-80，只能包含字母、数字、点号（.）和连字符（-）。	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-health-check-httpcode	string	健康检查正常的HTTP状态码，多个状态码用逗号（,）分割。取值： · http_2xx · http_3xx · http_4xx · http_5xx 默认值为http_2xx。	http_2xx

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-scheduler	string	调度算法。取值wrr | wlc| rr。 · wrr：默认取值，权重值越高的后端服务器，被轮询到的次数（概率）也越高。 · wlc：除了根据每台后端服务器设定的权重值来进行轮询，同时还考虑后端服务器的实际负载（即连接数）。当权重值相同时，当前连接数越小的后端服务器被轮询到的次数（概率）也越高。 · rr：按照访问顺序依次将外部请求依序分发到后端服务器。	wrr
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-status	string	是否开启访问控制功能。取值： on | off	off
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-id	string	监听绑定的访问策略组ID。当AclStatus参数的值为on时，该参数必选。	无

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-acl-type	string	访问控制类型。 取值：white | black。 · white：仅转发来自所选访问控制策略组中设置的IP地址或地址段的请求，白名单适用于应用只允许特定IP访问的场景。设置白名单存在一定业务风险。一旦设名单，就只有白名单中的IP可以访问负载均衡监听。如果开启了白名单访问，但访问策略组中没有添加任何IP，则负载均衡监听会转发全部请求。 · black：来自所选访问控制策略组中设置的IP地址或地址段的所有请求都不会转发，黑名单适用于应用只限制某些特定IP访问的场景。如果开启了黑名单访问，但访问策略组中没有添加任何IP，则负载均衡监听会转发全部请求。当AclStatus参数的值为on时，该参数必选。	无

注释	类型	描述	默认值
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-vswitch-id	string	负载均衡实例所属的VSwitch ID。设置该参数时需同时设置addresstype为intranet。	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-forward-port	string	将HTTP请求转发至HTTPS指定端口。取值如80:443	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-additional-resource-tags	string	需要添加的Tag列表，多个标签用逗号分隔。例如："k1=v1,k2=v2"	无
service.beta.kubernetes.io/alibaba-cloud-loadbalancer-remove-unscheduled-backend	string	从slb后端移除SchedulingDisabled Node。取值on | off	off
service.beta.kubernetes.io/backend-type	string	支持在Terway eni网络模式下，通过设定改参数为"eni"，可将Pod直接挂载到SLB后端，提升网络转发性能。取值：eni	无

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: /$2
spec:
  rules:
  - http:
      paths:
      - path: /svc(/|$)(.*)
        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。

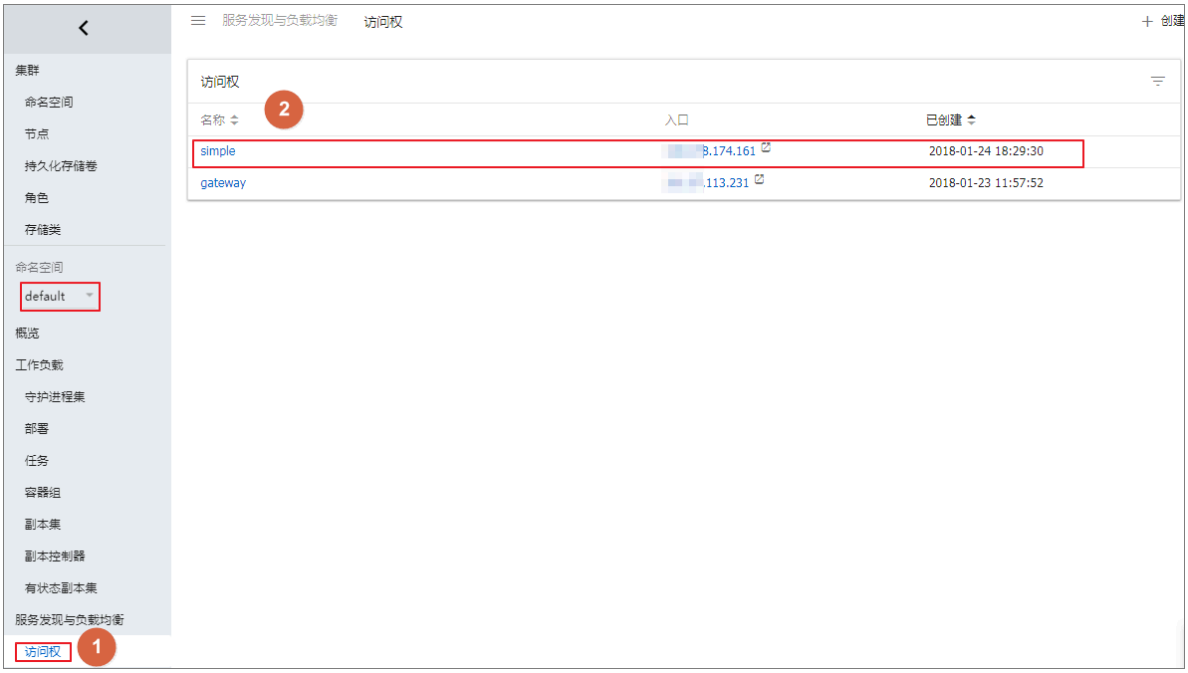
4. 单击使用文件创建。选择刚才保存的 `nginx-ingress.yml` 文件。

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。

8.4 Ingress 访问日志分析与监控

阿里云Ingress除了提供外部可访问的 URL、负载均衡、SSL、基于名称的虚拟主机外，还支持将所有用户的HTTP请求日志记录到标准输出中。同时Ingress访问日志与阿里云日志服务打通，您可以使用日志服务快速创建日志分析和监控大盘。

前提条件

1. 安装日志组件。

   集群创建时，默认会安装日志组件，如未安装，请参考 #unique_30进行手动安装。

2. 升级Log Controller。

   升级kube-system命名空间下的无状态应用（Deployment）alibaba-log-controller，替换以下内容：

   - 镜像名称：registry-vpc.{region-id}.aliyuncs.com/acs/log-controller，将其中的`{region-id}`替换为您集群所在Region ID，例如cn-hangzhou、cn-beijing、ap-southeast-1等。
   - 镜像版本（Tag）：版本不低于0.2.0.0-76648ee-aliyun。

   以下两种升级方法您可任选其一：

   - 通过执行kubectl命令`kubectl edit deployment alibaba-log-controller -n kube-system`进行升级。
   - 通过容器服务控制台升级。进入应用 > 无状态（Deployment），选择对应集群的命名空间kube-system，编辑alibaba-log-controller并保存。

部署Ingress采集配置

```
apiVersion: log.alibabacloud.com/v1alpha1
kind: AliyunLogConfig
metadata:
  # your config name, must be unique in you k8s cluster
  name: k8s-nginx-ingress
spec:
  # logstore name to upload log
  logstore: nginx-ingress
  # product code, only for k8s nginx ingress
  productCode: k8s-nginx-ingress
  # logtail config detail
  logtailConfig:
    inputType: plugin
    # logtail config name, should be same with [metadata.name]
    configName: k8s-nginx-ingress
    inputDetail:
      plugin:
        inputs:
        - type: service_docker_stdout
          detail:
```

```
            IncludeLabel:
              io.kubernetes.container.name: nginx-ingress-controller
            Stderr: false
            Stdout: true
        processors:
        - type: processor_regex
          detail:
            KeepSource: false
            Keys:
            - client_ip
            - x_forward_for
            - remote_user
            - time
            - method
            - url
            - version
            - status
            - body_bytes_sent
            - http_referer
            - http_user_agent
            - request_length
            - request_time
            - proxy_upstream_name
            - upstream_addr
            - upstream_response_length
            - upstream_response_time
            - upstream_status
            - req_id
            - host
            NoKeyError: true
            NoMatchError: true
            Regex: ^(\S+)\s-\s\[([^]]+)]\s-\s(\S+)\s\[(\S+)\s\S+\s"(\w
+)\s(\S+)\s([^"]+)"\s(\d+)\s(\d+)\s"([^"]*)"\s"([^"]*)"\s(\S+)\s(\S+)+
\s\[([^]]*)]\s(\S+)\s(\S+)\s(\S+)\s(\S+)\s(\S+)\s*(\S*).*
            SourceKey: content
```

日志采集配置针对Kubernetes进行了CRD扩展，可直接部署AliyunLogConfig的CRD配置，Log Controller会自动创建日志服务相关采集配置和报表资源，部署方式任选其一：

注意:

- 请确保日志组件 alibaba-log-controller版本不低于`0.2.0.0-76648ee-aliyun`。更新版本后，若已经应用了该CRD配置，请删除该配置并重新应用。
- 上述配置只针对阿里云Kubernetes默认Ingress Controller中的日志格式生效，若您修改过Ingress Controller的访问日志格式，请根据#unique_155修改上述CRD配置中的正则表达式提取`processor_regex`部分。

- 使用Kubectl命令部署

  将上述CRD配置保存成*nginx-ingress.yaml*，执行 `kubectl apply -f` 命令进行部署。

- 使用编排模板部署

  1. 登录容器服务管理控制台。
  2. 将上述CRD配置保存成编排模板。

     编排模板文档请参见#unique_156
  3. 基于该模板创建应用，选择您所在集群的default命名空间。

查看Ingress日志与报表

1. 登录日志服务控制台。
2. 单击左侧导航栏中的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（默认创建的project名称为 k8s-log-{cluster-id}）。
3. 在Project详情页面，默认进入日志库页面。名称为nginx-ingress的日志库（logstore）存放着所有的Ingress访问日志。单击左侧导航栏中的仪表盘进入仪表盘列表，可查看到所有Ingress的分析报表。

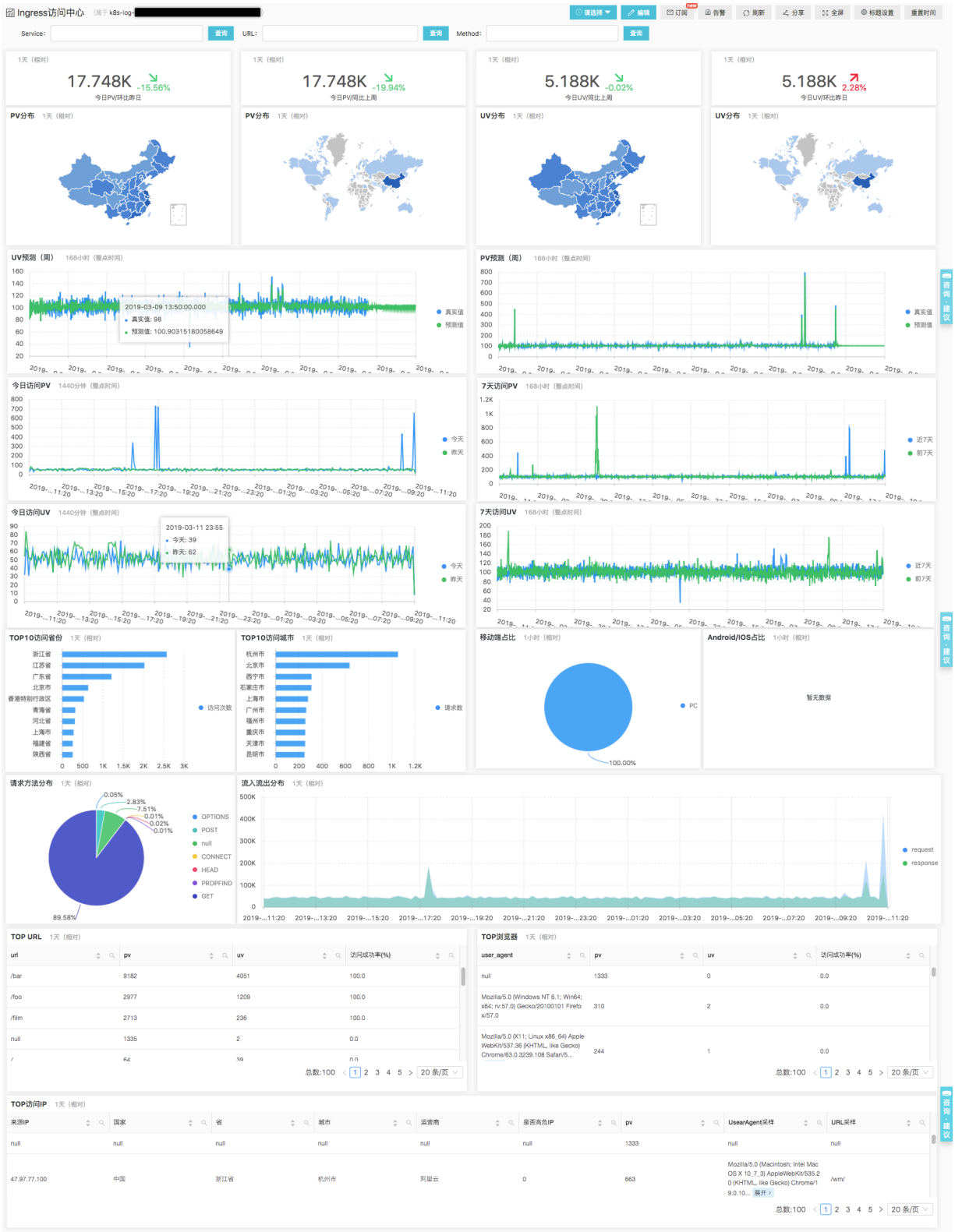
Ingress概览

Ingress概览报表主要展示当前Ingress的整体状态，主要包括以下几类信息：

- 整体架构状态（1天），包括PV、UV、流量、响应延迟、移动端占比和错误比例等。
- 网站实时状态（1分钟），包括PV、UV、成功率、5XX比例、平均延迟和P95/P99延迟等。
- 用户请求类信息（1天），包括1天/7天访问PV对比、访问地域分布、TOP访问省份/城市、移动端占比和Android/IOS占比等。
- TOPURL统计（1小时），包括访问TOP10、延迟TOP10、5XX错误TOP10和404错误TOP10。

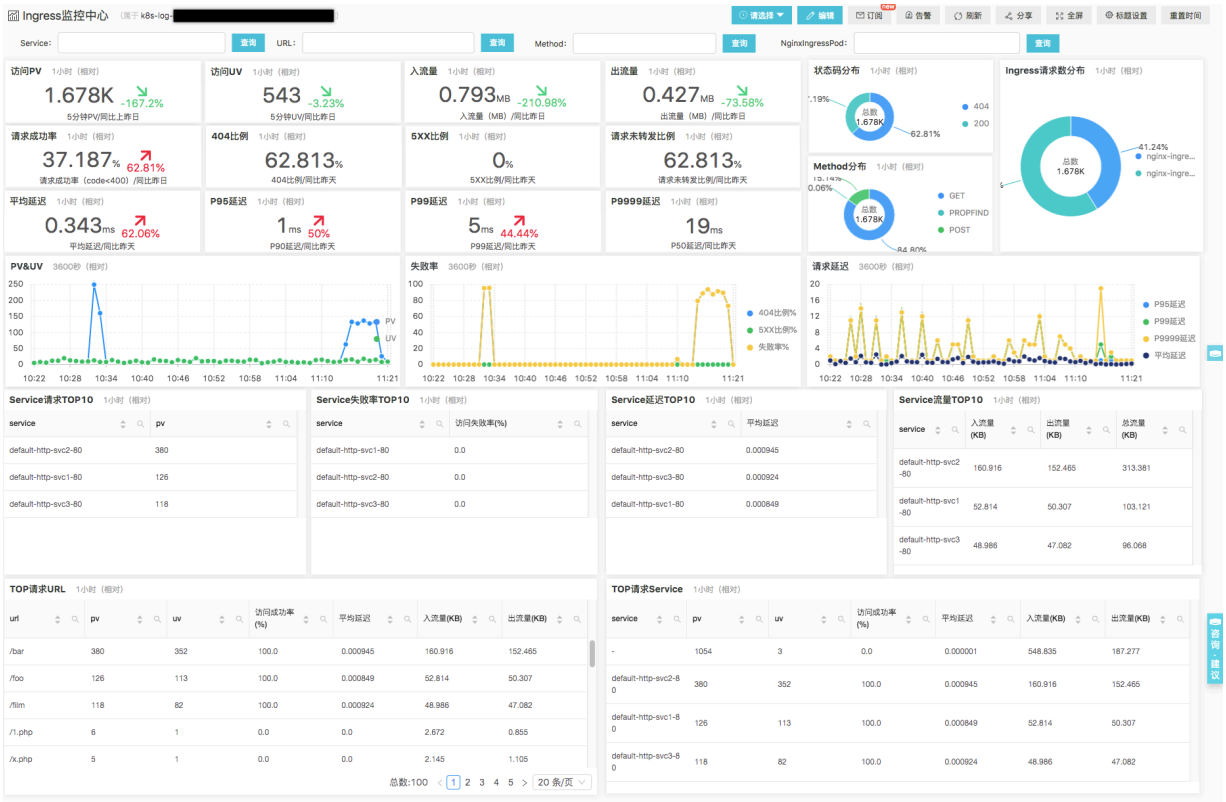
Ingress访问中心

Ingress访问中心主要侧重于用于访问请求相关的统计信息，一般用于运营分析，包括当日UV/PV、UV/PV分布、UV/PV趋势、TOP访问省份/城市、TOP访问浏览器、TOP访问IP、移动端占比和Android/IOS占比等。



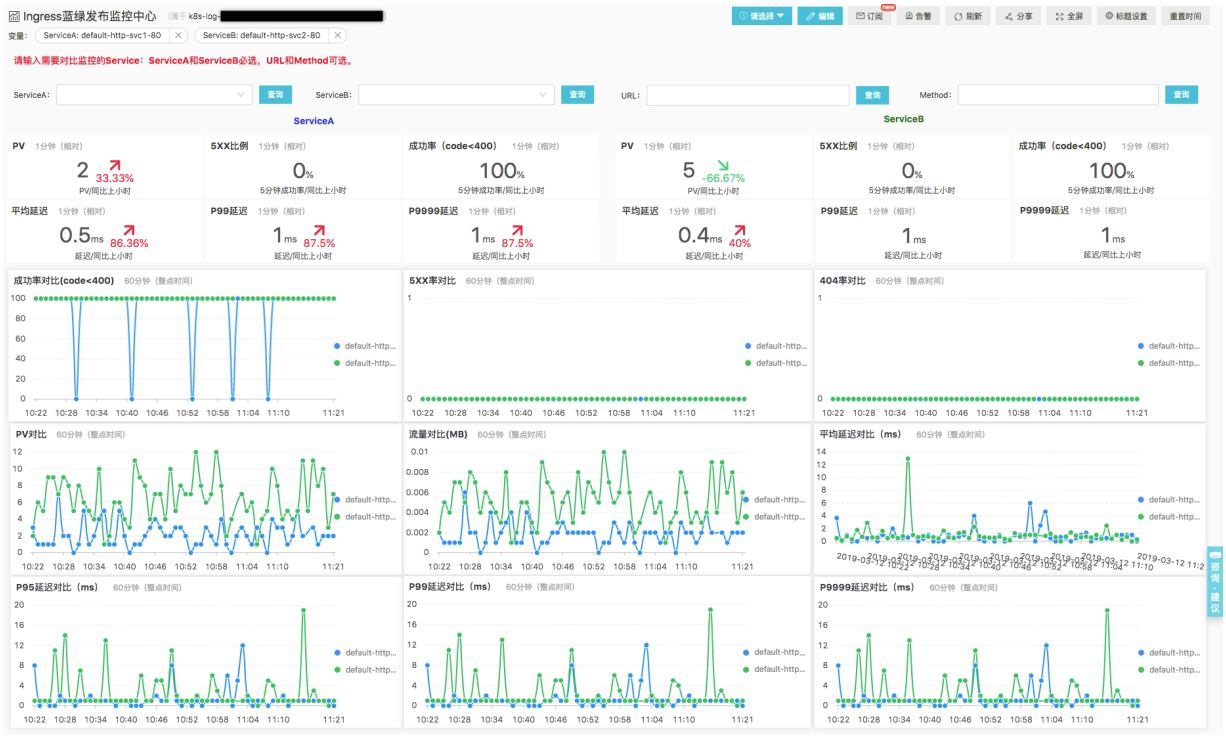
Ingress监控中心

Ingress监控中心主要侧重于网站实时监控数据，一般用于完整实时监控与告警，包括请求成功率、错误比例、5XX比例、请求未转发比例、平均延迟、P95/P99/P9999延迟、状态码分布、Ingress压力分布、Service访问TOP10、Service错误TOP10、Service延迟TOP10和Service流量TOP10等。



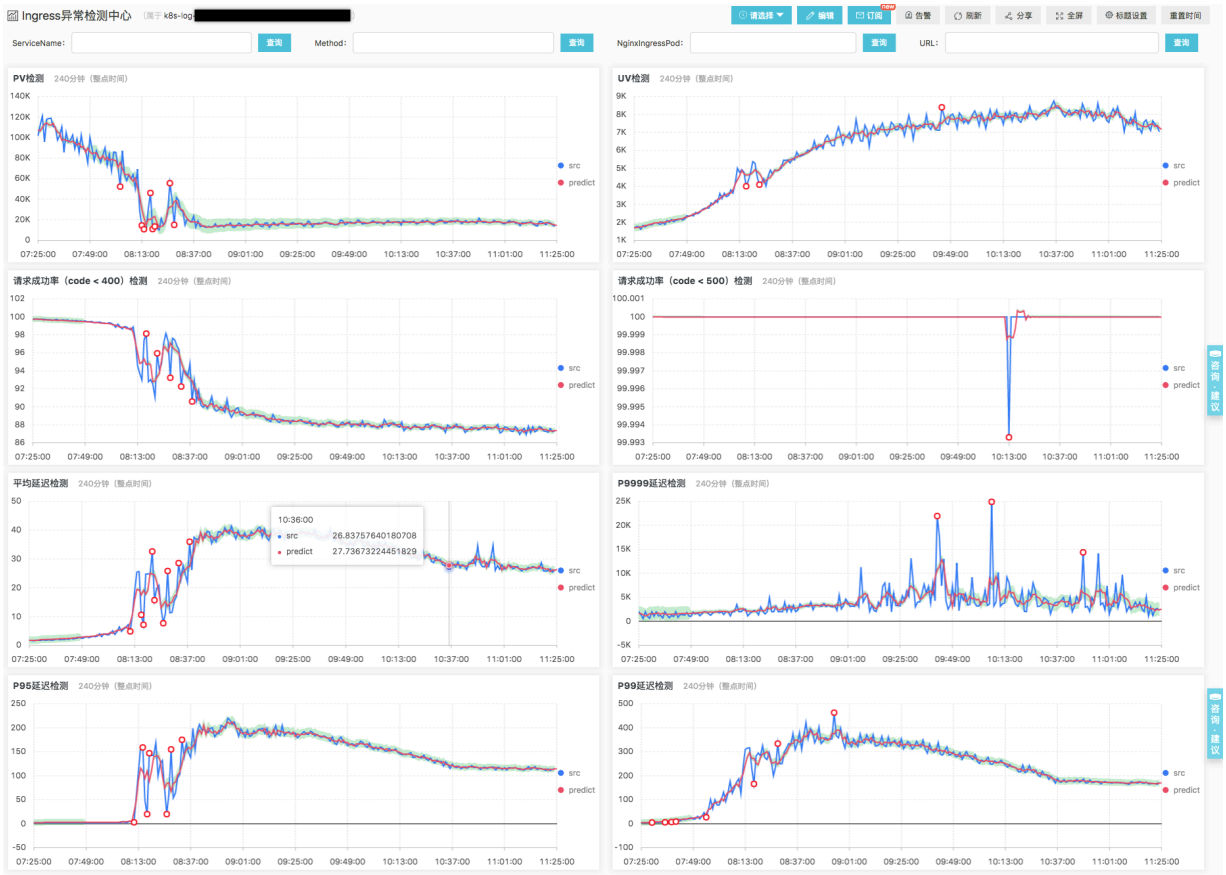
Ingress蓝绿发布监控中心

Ingress蓝绿发布监控中心主要用于版本发布时的实时监控与对比（版本前后对比以及蓝绿版本当前对比），以便您在服务发布时快速检测异常并进行回滚。在该报表中您需要选择进行对比的蓝绿版本（ServiceA和ServiceB），报表将根据您的选择动态显示蓝绿版本相关指标，包括PV、5XX比例、成功率、平均延迟、P95/P99/P9999延迟和流量等。



Ingress异常检测中心

Ingress异常检测中心基于日志服务提供的机器学习算法，通过多种时序分析算法从Ingress的指标中自动检测异常点，提高问题发现的效率。



配置告警

日志服务除提供交互式分析、可视化报表之外，您可直接基于上述报表快速配置告警，告警支持通知钉钉WebHook、短信、邮件和自定义WebHook等。

告警详细配置方法请参考#unique_157。

下述示例为Ingress配置5XX比例的告警，告警每5分钟执行一次，当5XX比例超过1%时触发。

1. **进入Ingress监控中心报表，鼠标滑动到图表5XX比例的右上角，在弹出的提示框中单击新建告警。**

2. 在告警页面中，填入告警名称、查询区间和执行间隔，查询语句中的`total`为5XX的百分比，因此触发条件填入：`total > 1`。

3. 在下一步的通知选项中，根据您的需求选择对应的通知方式，并填入对应参数即可完成告警创建。

订阅定时报告

日志服务除支持通过告警方式通知外，还支持报表订阅功能，您可使用该功能将报表定期渲染成图片并通过邮件、钉钉群等方式发送。

订阅功能详细使用方法请参考#unique_158。

下述示例为Ingress概览配置订阅功能，每天上午10点将报表发送到指定钉钉群：

1. 进入Ingress概览报表，单击报表右上角的订阅按钮。
2. 在弹出的配置页面中，频率选择每天、10:00，关闭添加水印选项。
3. 通知类型中选择钉钉机器人，填入钉钉机器人的WebHook地址（WebHook地址请参见自定义机器人获取）即可完成订阅。

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#unique_160。 也可以参考#unique_161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#unique_161/unique_161_Connect_42_section_j4d_jrs_vdb。

标签

您可为 ingress 添加标签，标示该 Ingress 的特点。

8.6 通过界面创建路由（Ingress）

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

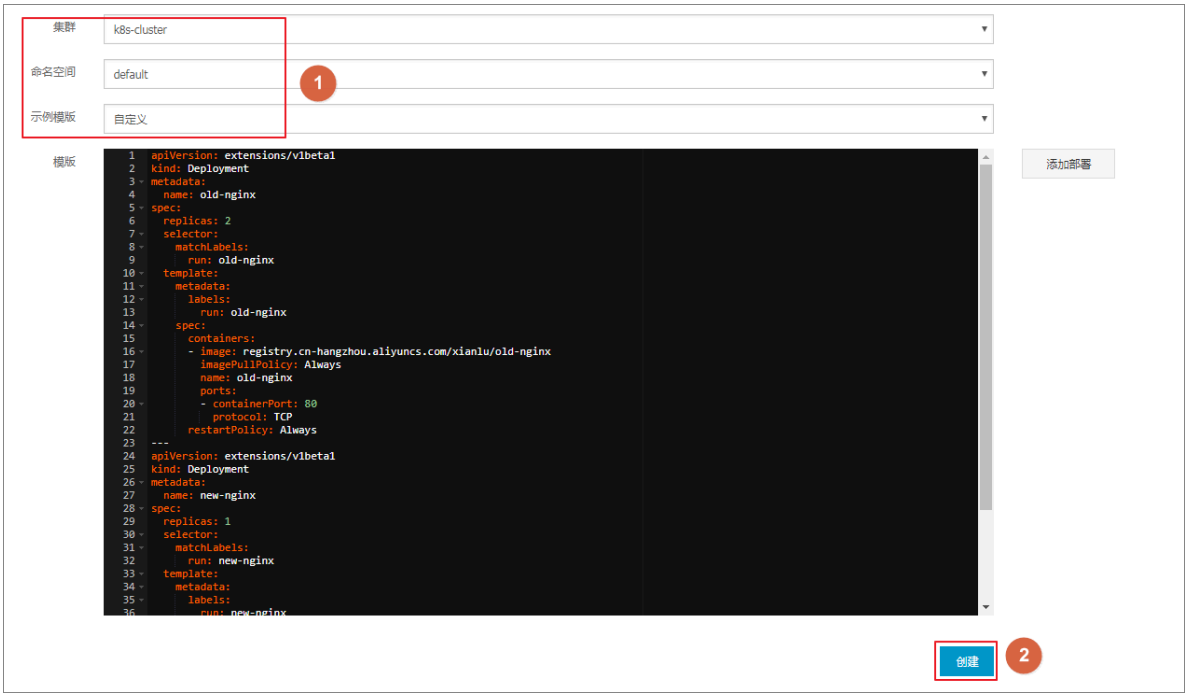
- 您已经成功创建一个 Kubernetes 集群，参见#unique_48，并且集群中 Ingress controller 正常运行。
- SSH 登录到 Master 节点，参见#unique_29。

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 > 无状态，进入无状态（Deployment）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路由与负载均衡 > 服务，进入服务列表页面。

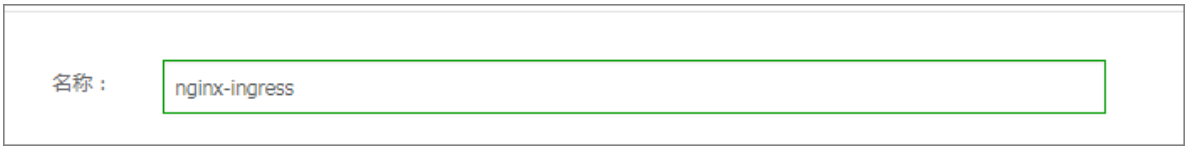
   等待服务创建完成后，在服务列表，您可看到本示例创建的服务。

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路由与负载均衡 > 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。

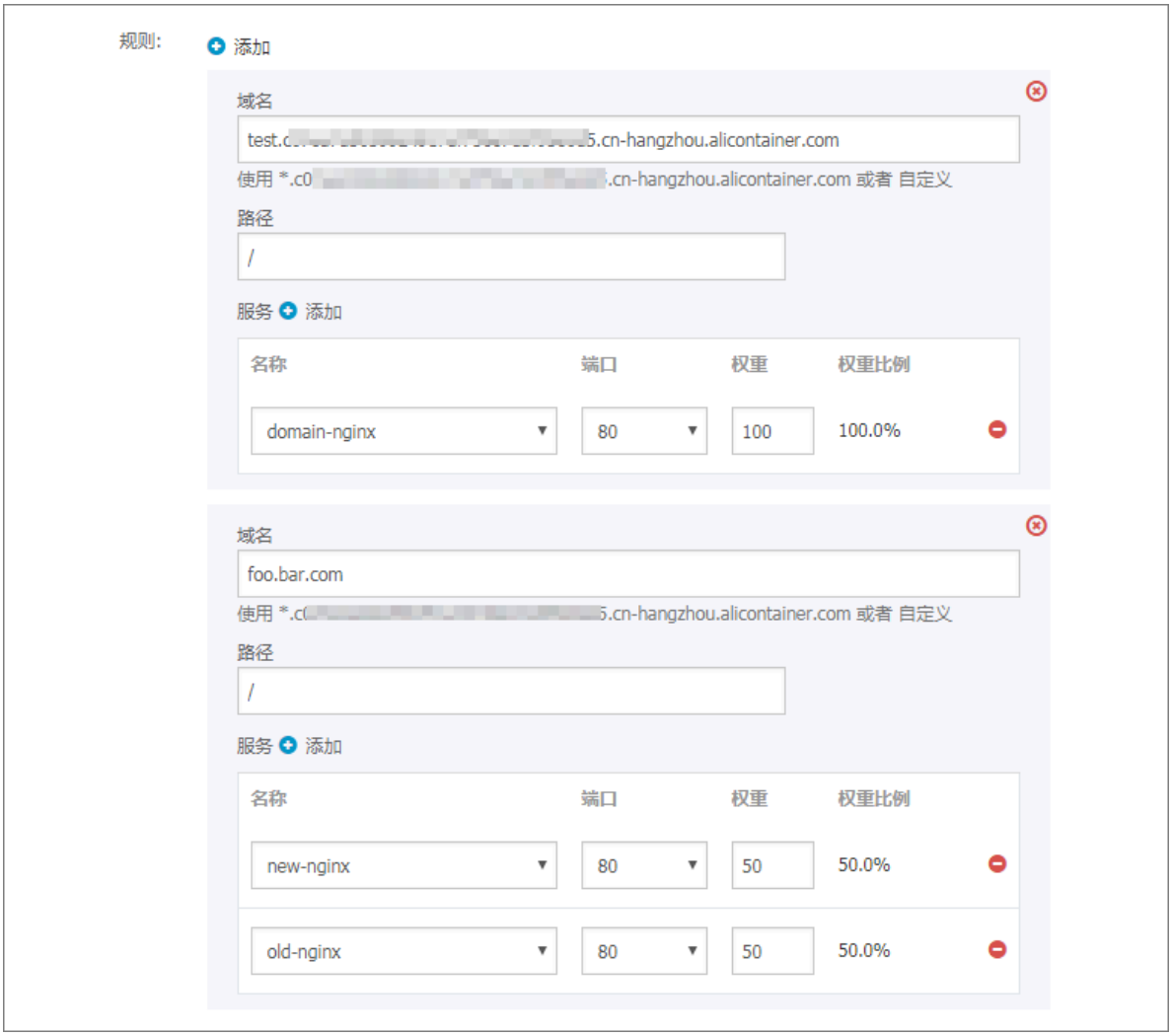
4. 在弹出的路由创建对话框中，首先配置路由名称，本例为 nginx-ingress。



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#unique_111。

   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。

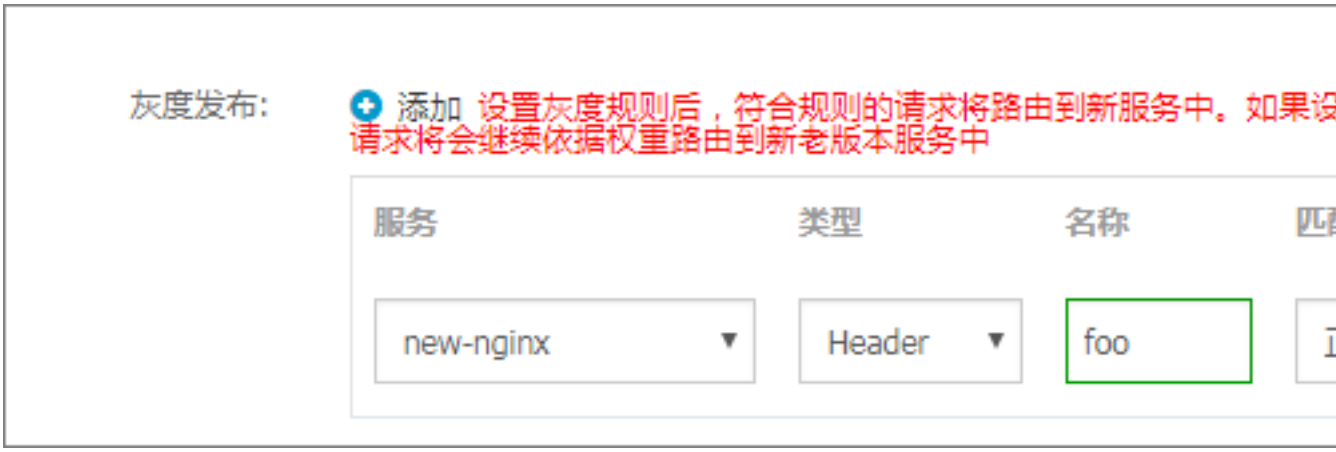
a. 基于Request Header的流量切分

b. 基于Cookie的流量切分

c. 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。

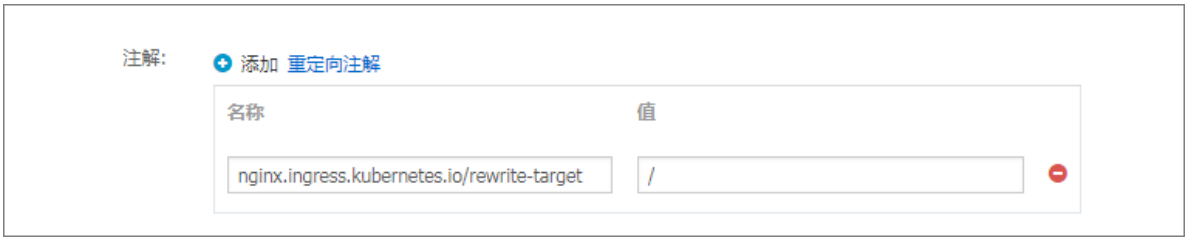
7. 配置注解。

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#unique_161/unique_161_Connect_42_section_j4d_jrs_vdb。

   · 您可选择使用已有密钥。

   a. 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"
```

   b. 创建一个 secret。

```
kubectl create secret tls foo.bar --key tls.key --cert tls.crt
```

   c. 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。

   a. 登录 master 节点，创建 *tls.key* 和 *tls.crt*。

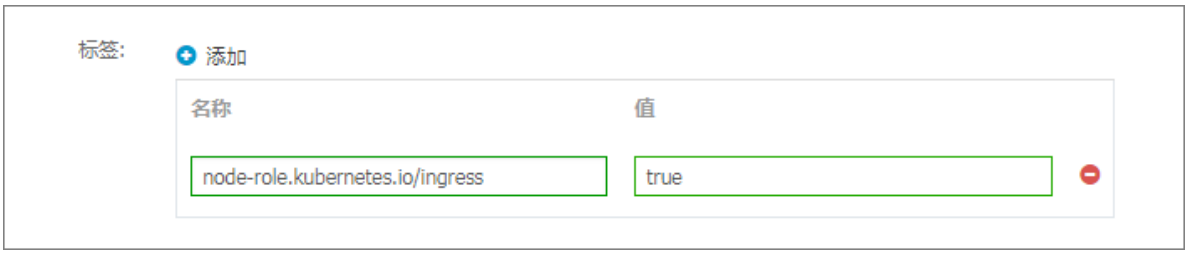
```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls
.key -out tls.crt -subj "/CN=foo.bar.com/O=foo.bar.com"
```

   b. 执行 `vim tls.key` 和 `vim tls.crt` 获取生成的私钥和证书。

   c. 将证书和私钥的内容复制到 TLS 新建密钥的面板。

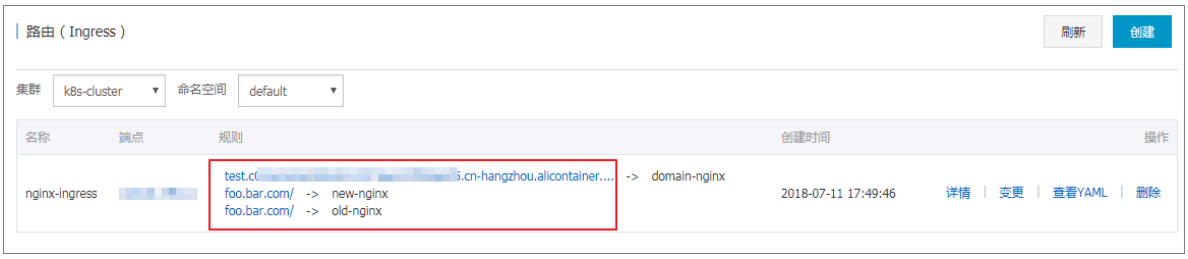
9. 添加标签。

   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
```

8.7 查看路由

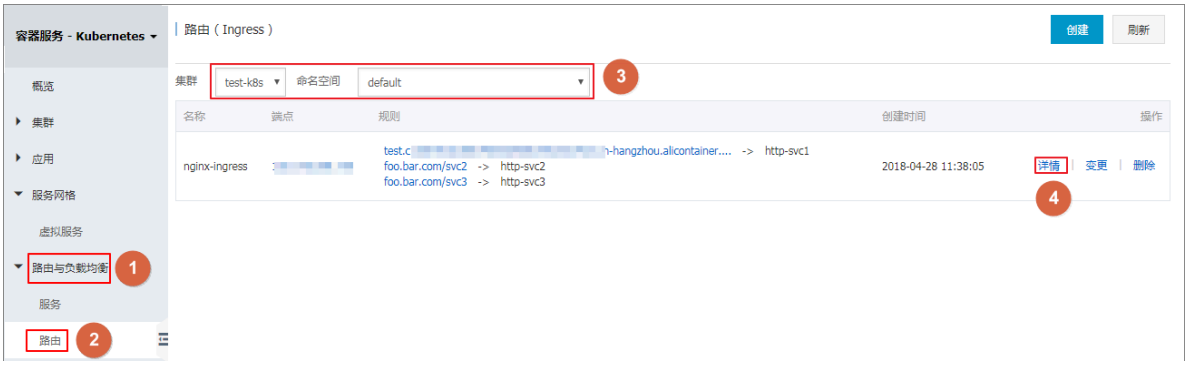
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见#unique_160。

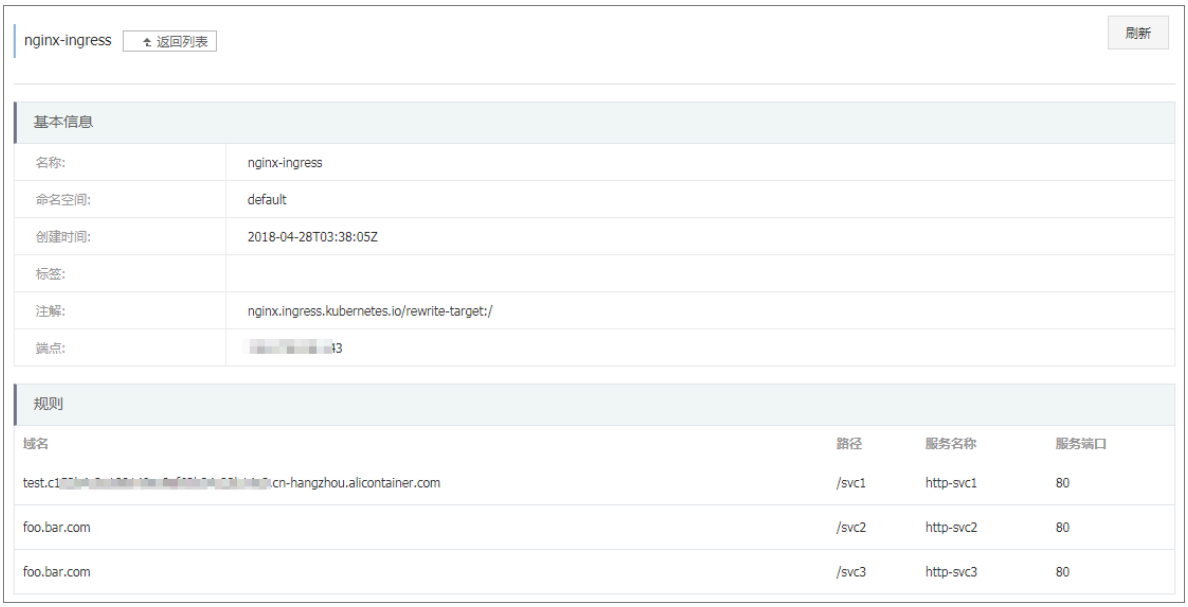
操作步骤

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 路由与负载均衡 > 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。



8.8 变更路由

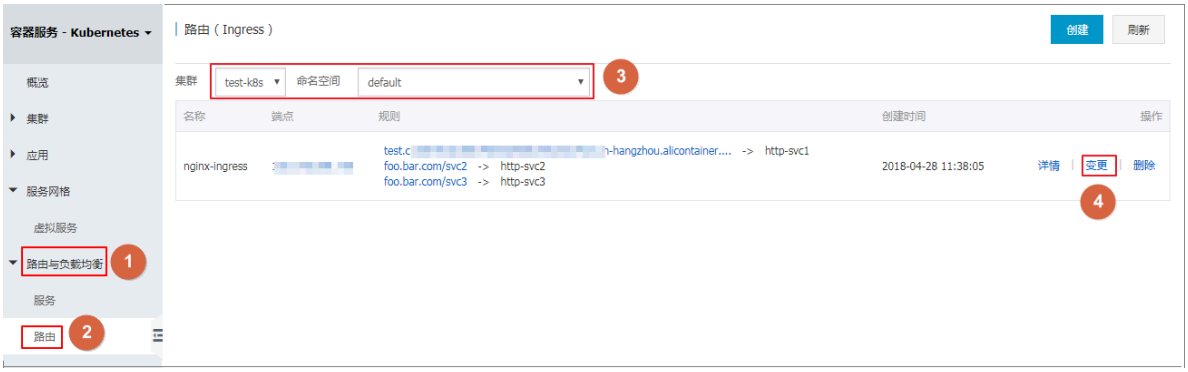
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见#unique_160。

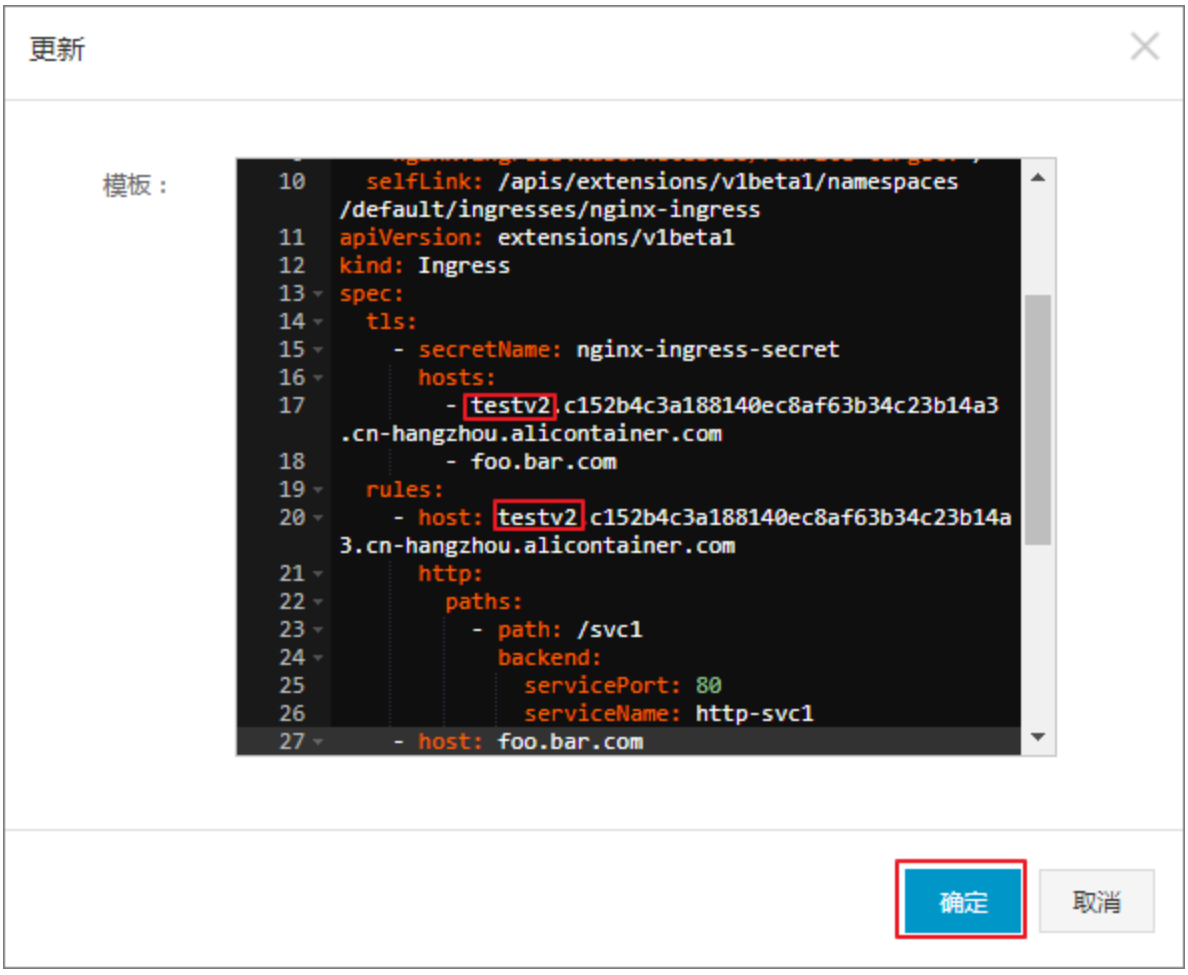
操作步骤

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路由与负载均衡 > 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。

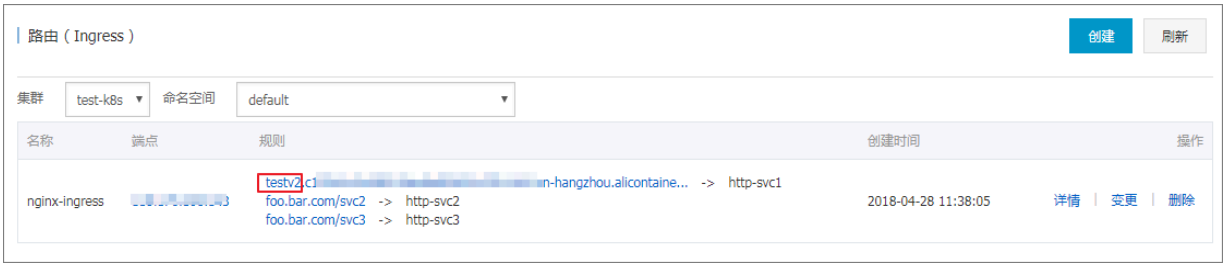


4. 在弹出的对话框中，对路由的相关参数进行变更，然后单击确定。本例中将 `test.[cluster-id].[region-id].alicontainer.com` 修改为 `testv2.[cluster-id].[region-id].alicontainer.com`。



后续步骤

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。



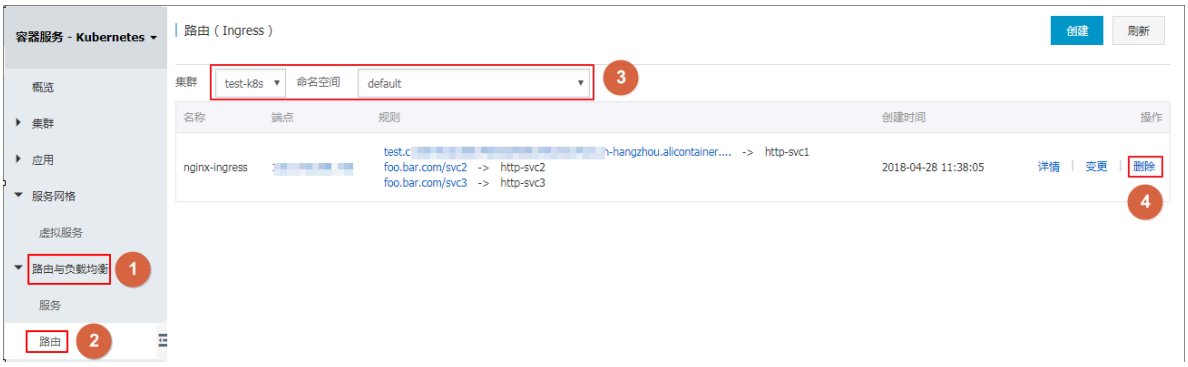
8.9 删除路由

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48，并且集群中 Ingress controller 正常运行。
- 您已经成功创建一个路由，参见#unique_160。

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路由与负载均衡 > 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。

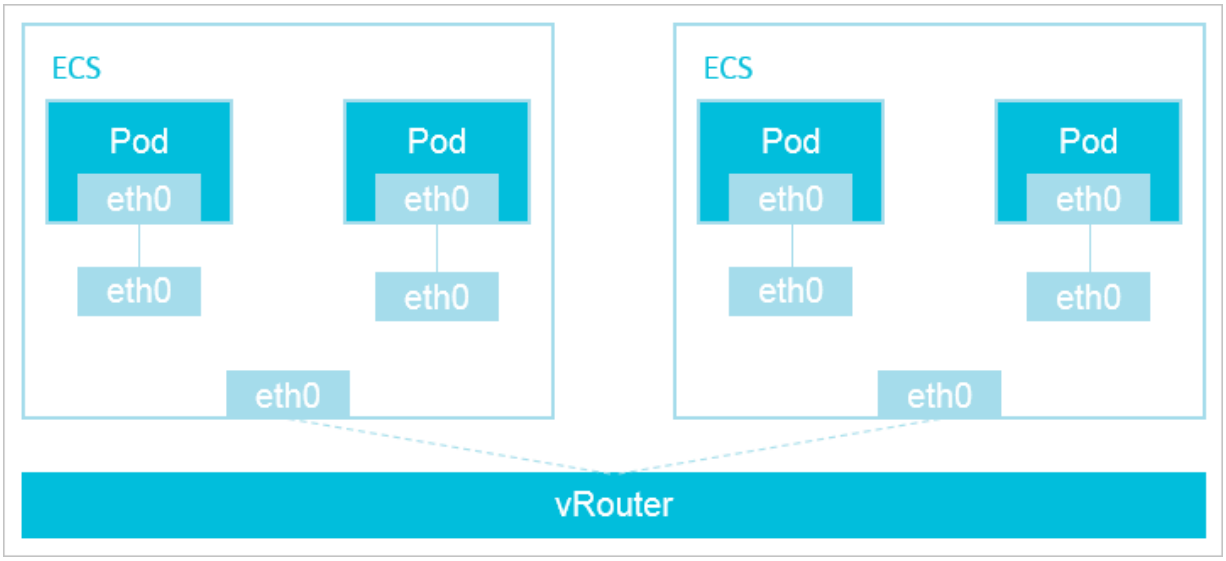
8.10 如何使用Terway网络插件

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#unique_48。

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#unique_27

- 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。

8.11 为容器（Pod）分配弹性网卡（ENI）

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#unique_48。
- 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 > 无状态，进入无状态（Deployment）页面。
3. 单击页面右上角的使用模板创建。

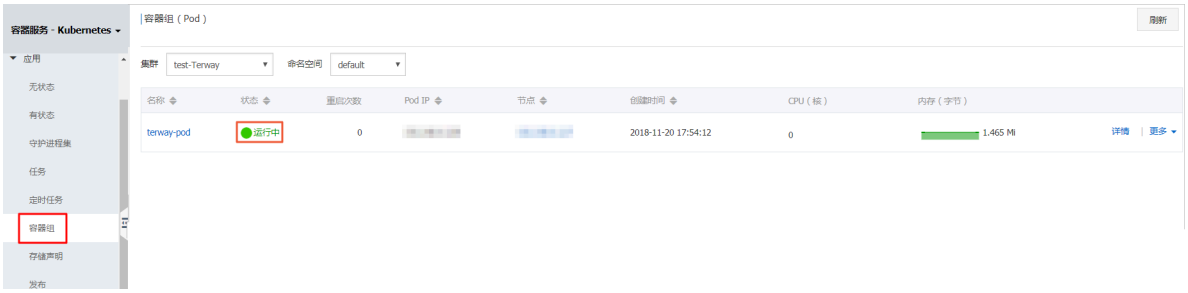
   您可以使用如下 yaml 示例模板创建Pod。

```
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
```

8.12 使用网络策略（Network Policy）

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48。
- 您在创建集群时，网络插件选择Terway，参见#unique_48。
- 您可以通过Kubectl连接到Kubernetes 集群，参见#unique_37。

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
```

```
/ #
```

8.13 使用Host网络

Pod使用Host网络，就是让Pod使用Node的网络空间和网络资源。配置成Host网络后，Pod可以直接访问到Node上loopback设备，可以在Node上监听地址，也可以用于监控Node上别的Pod的网络流量。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_95。
- 您可以通过Kubectl连接到Kubernetes 集群，参见#unique_43。

操作步骤

1. 打开您的Yaml文件*host-network.yaml*，在*spec*下指定`hostNetwork: true`。

   Yaml文件完整样例如下：

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: nginx
   spec:
     hostNetwork: true
     containers:
     - name: nginx
       image: nginx
   ```

2. 执行如下命令，创建一个Pod。

   ```
   kubectl apply -f host-network.yaml
   ```

3. 执行如下命令，查看Pod的网络和其所在的Node网络是否一致。

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

   当显示结果中，Pod的IP地址和宿主机的IP地址一致时，表示Host网络配置完成。

   ```
   NAME    READY   STATUS    RESTARTS   AGE   IP              NODE
                       NOMINATED NODE
   nginx   1/1     Running   0          29s   192.168.XX.XX   cn-
   zhangjiakou.192.168.XX.XX   <none>
   ```

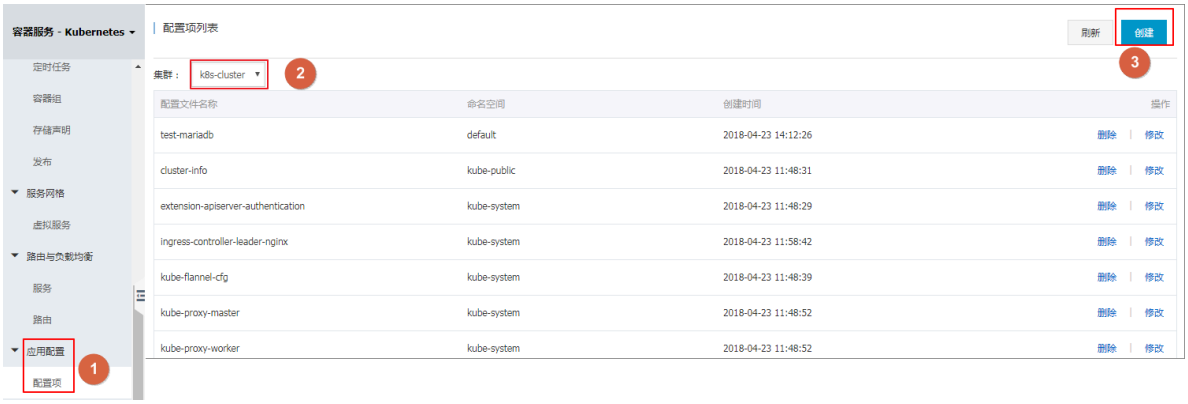
9 配置项及密钥管理

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。

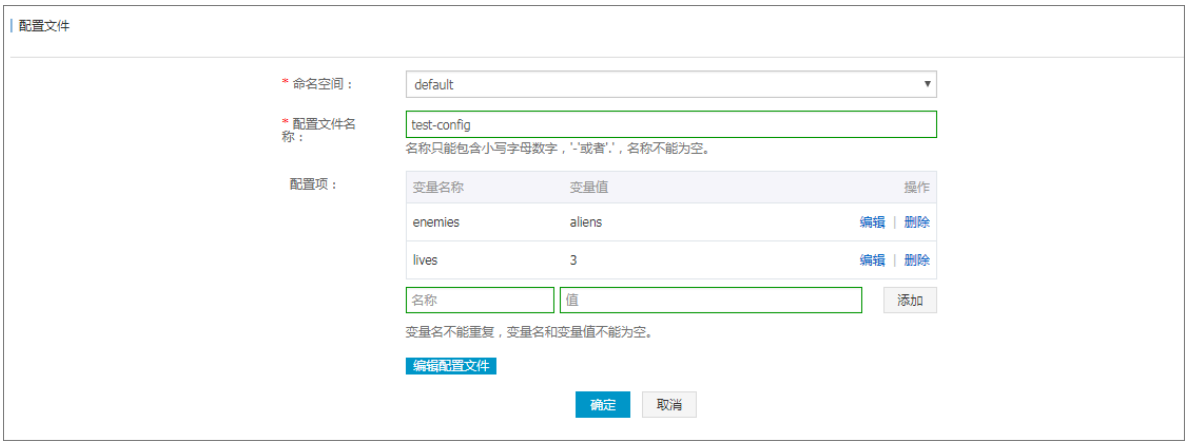
通过配置项创建

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 > 配置项，进入配置项列表。
3. 在配置项列表页面，选择需要创建配置项的集群，然后单击创建。

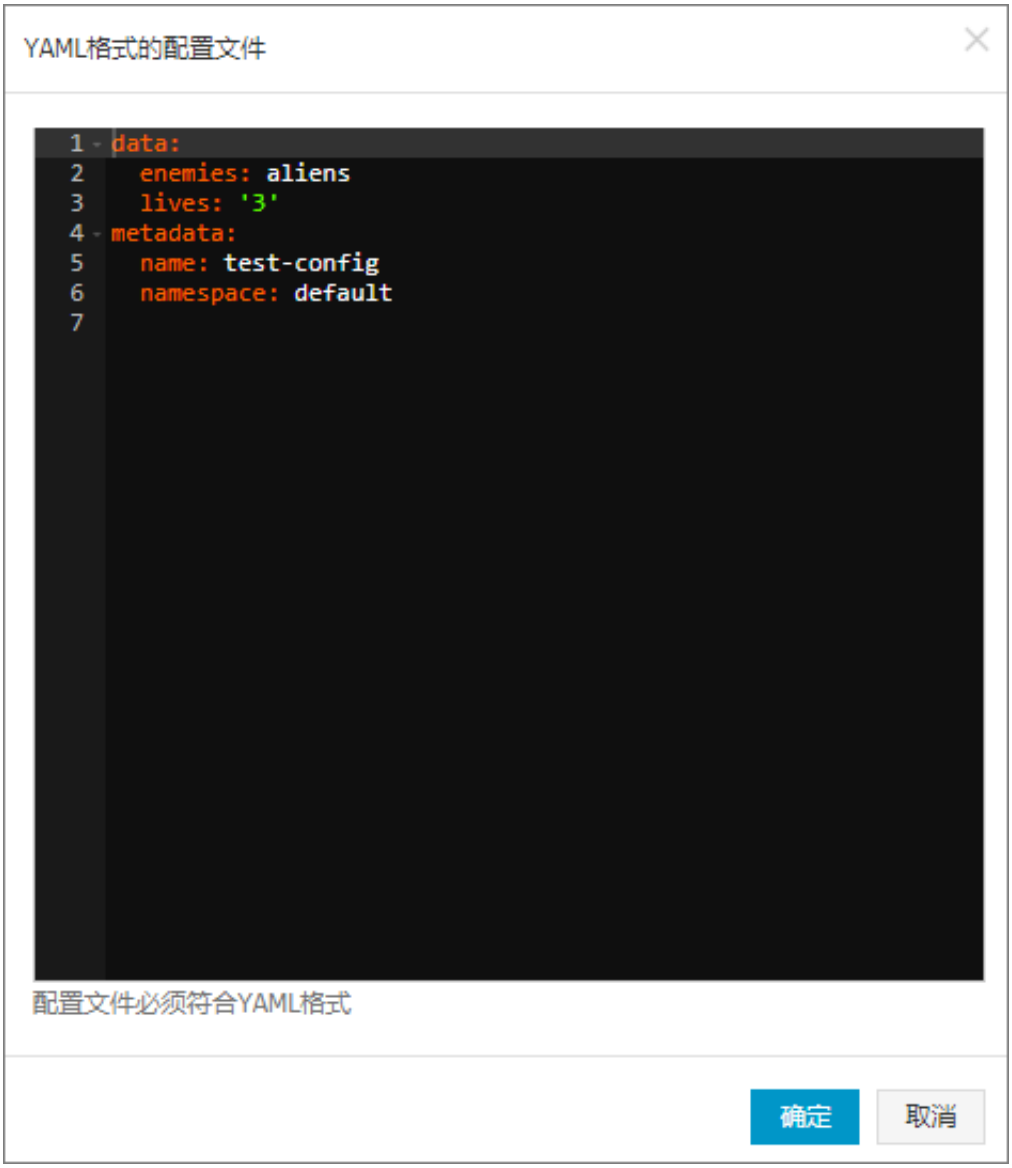


4. 填写配置文件的信息并单击确定。

   - 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，您可以在配置项列表中看到 test-config 配置文件。

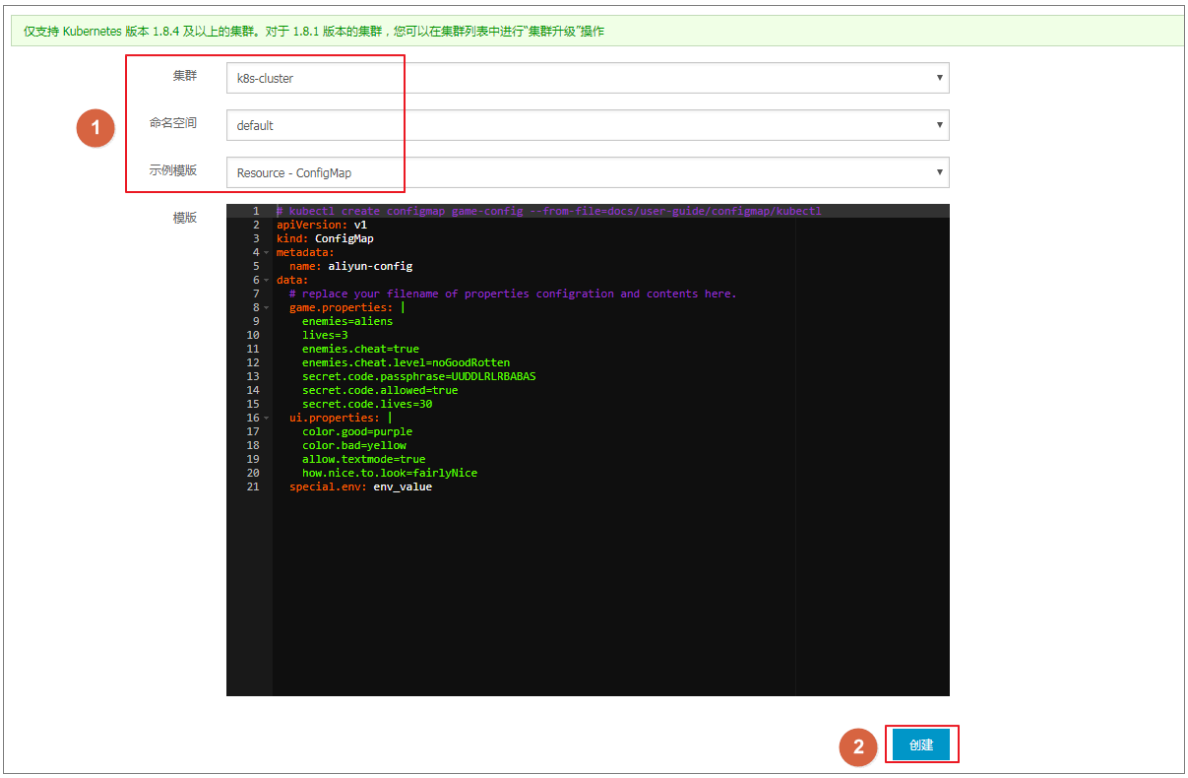
使用模板创建

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 > 无状态，进入无状态（Deployment）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创建。

   - 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，然后单击创建。



5. 部署成功后，您可以在配置项列表下看到 aliyun-config 配置文件。

9.2 在容器组中使用配置项

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 > 无状态，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。

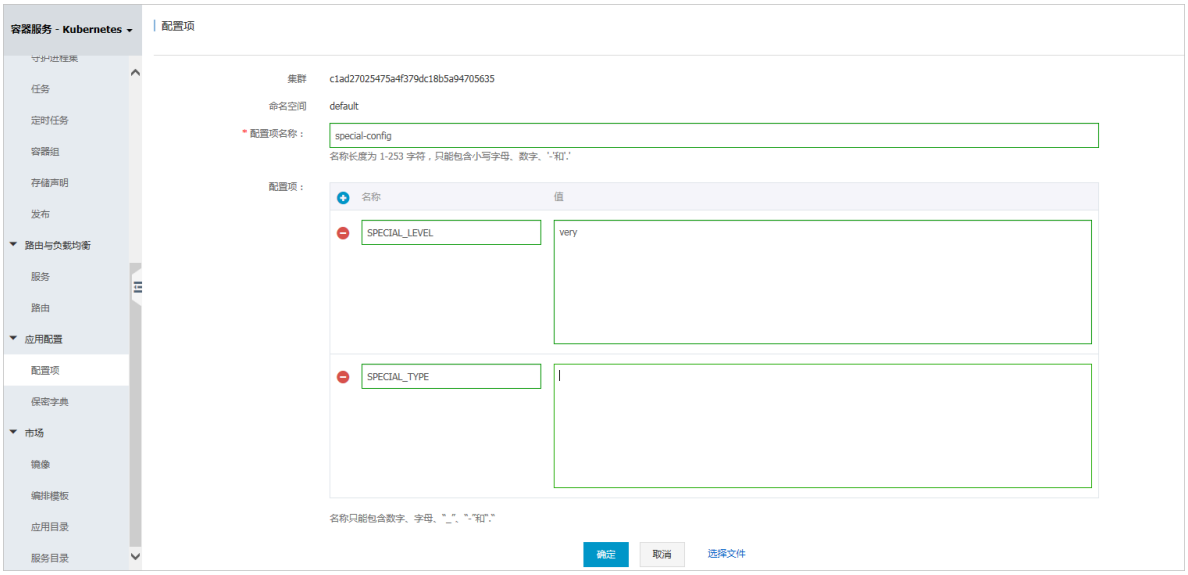
   您可以使用如下 yaml 示例模板创建配置项。

```
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 > 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，然后单击右上角的创建。
4. 输入配置项名称，然后单击 ，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 > 无状态，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 > 无状态，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 > 无状态，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 > 无状态，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
```

9.3 查看配置项

您可通过阿里云容器服务 Web 界面查看已创建的配置项。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已创建一个配置项，参见#unique_172。

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配置 > 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，选择所需的配置项，并单击右侧的详情。

4. 进入配置项的详情页面，您可查看该配置项的基本信息，以及配置项包含的数据信息。

9.4 修改配置项

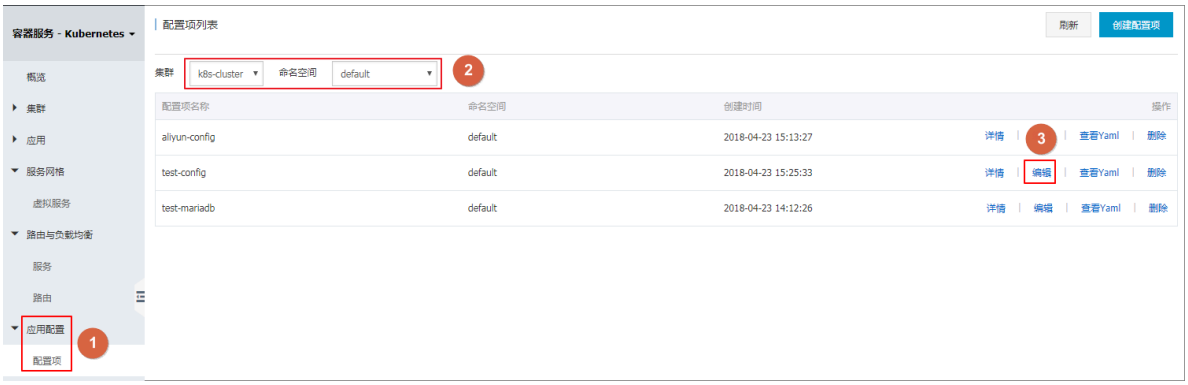
您可以修改配置项的配置。

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 > 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，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。

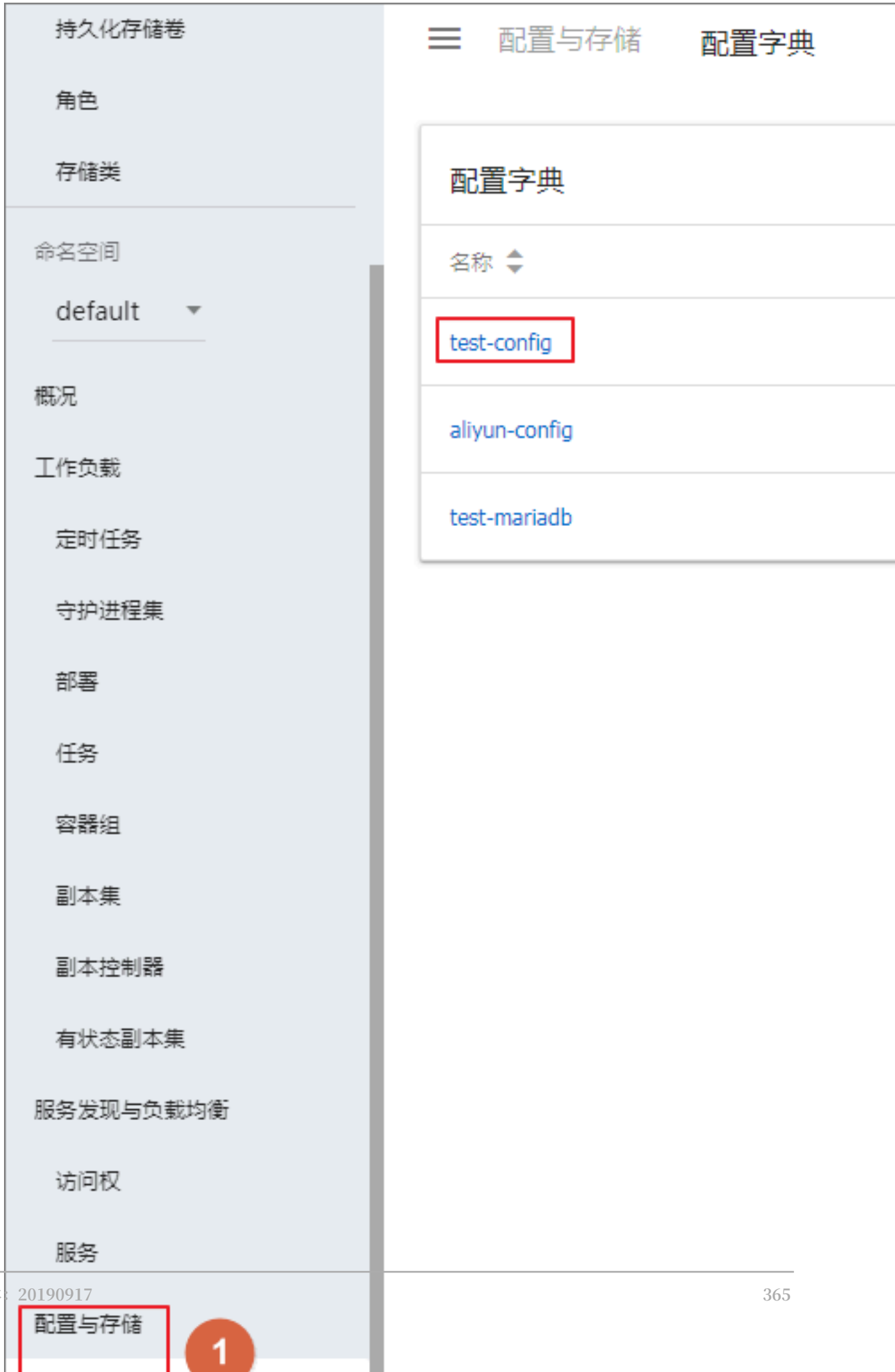
5. 修改配置项。
   - 选择需要修改的配置项并单击右侧的编辑，修改配置后单击保存。
   - 或者单击编辑配置文件，完成编辑后单击确定。

6. 完成配置项修改后，单击确定。

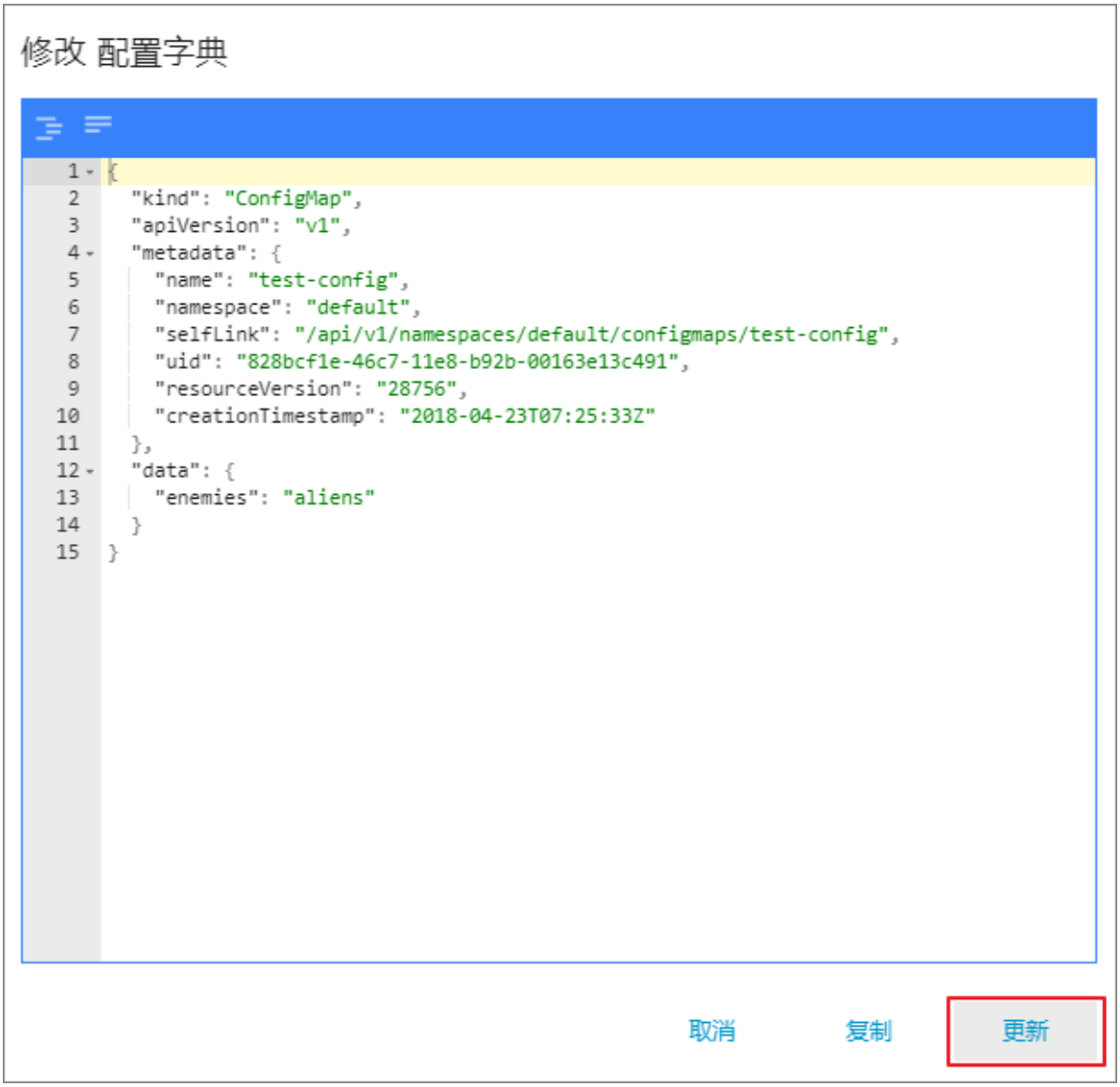
通过 Kubernetes Dashboard 进行修改

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 > 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
1
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。



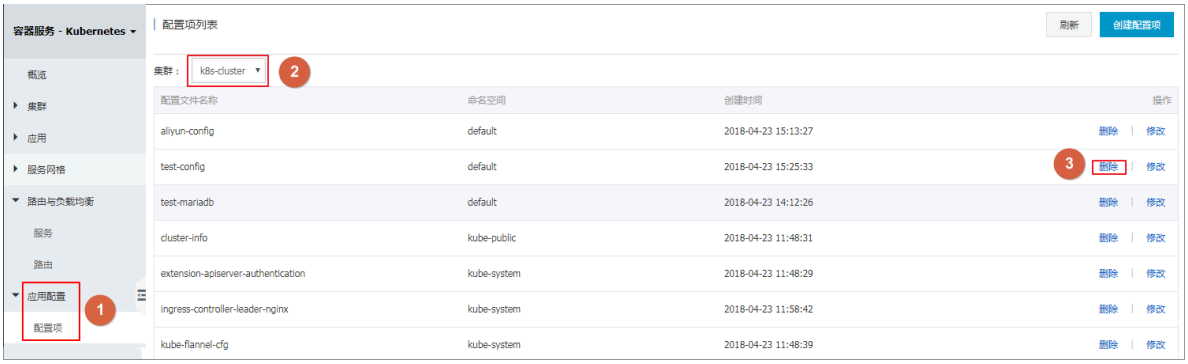
9.5 删除配置项

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 > 配置项，进入配置项列表页面。

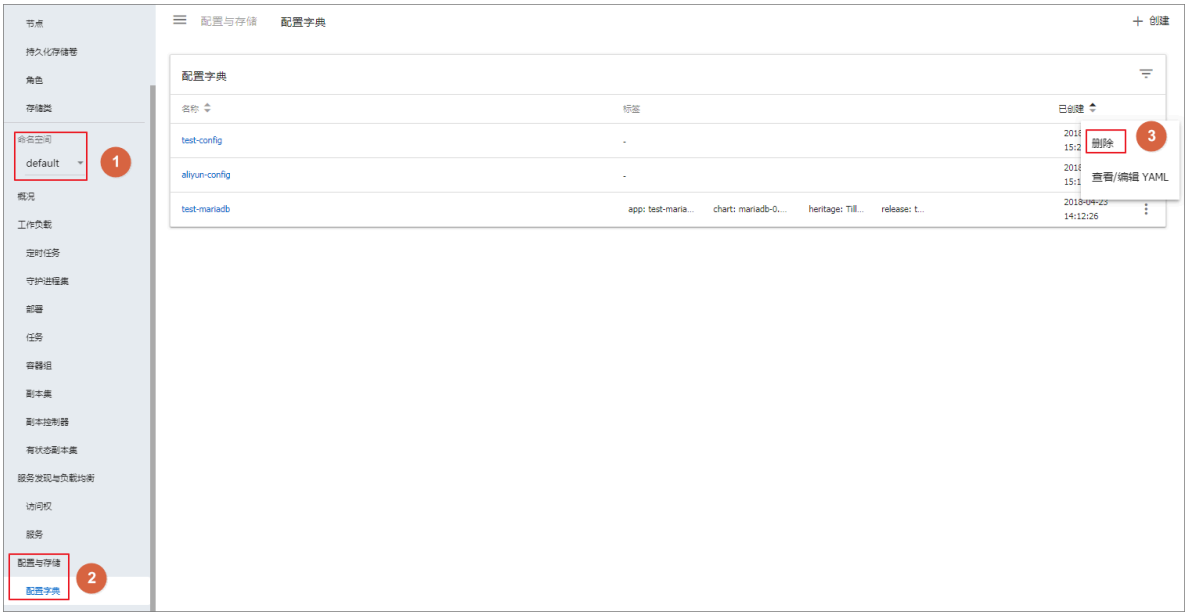
3. 选择目标集群，选择需要修改的配置项并单击右侧的删除。



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。

9.6 创建密钥

前提条件

您已成功创建一个 Kubernetes 集群，参见#unique_48。

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：

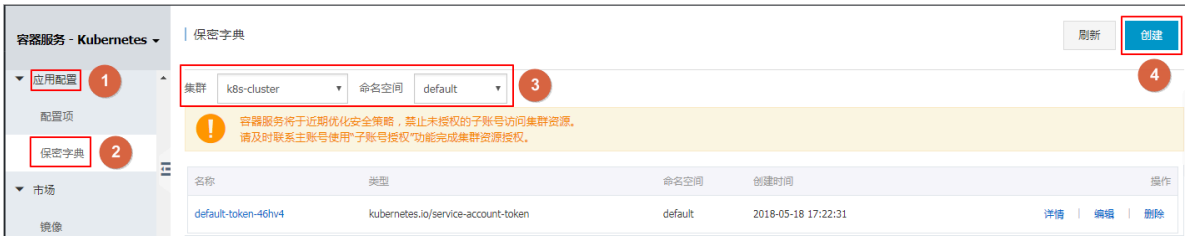
- Service Account：用来访问 Kubernetes API，由 Kubernetes 自动创建，并且会自动挂载到 Pod 的`/run/secrets/kubernetes.io/serviceaccount`目录中。
- Opaque：base64 编码格式的 Secret，用来存储密码、证书等敏感信息。

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

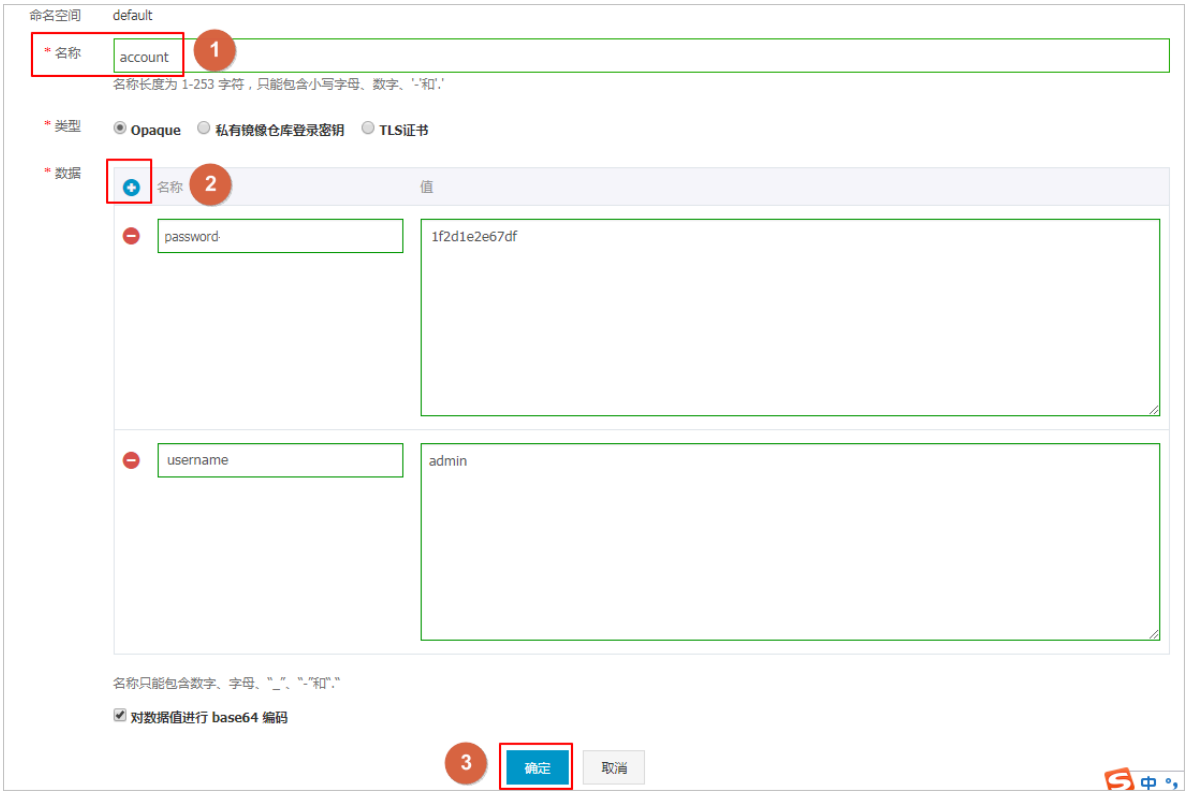
1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用配置 > 保密字典，进入保密字典页面。
3. 选择所需的集群和命名空间，单击右上角的创建。



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。



a. 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’ .’ 。

b. 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`。

c. 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。

9.7 在容器组中使用密钥

您可以在 Pod 中使用密钥，有多种使用场景，主要包括：

- 使用密钥配置Pod的数据卷
- 使用密钥设置Pod的环境变量

更多关于密钥的信息，可以参见 密钥。

前提条件

- 您在 Pod 里使用密钥时，需要两者处于同一集群和命名空间中。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

创建密钥

本示例创建密钥 secret-test。

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 示例模板创建密钥。

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-test
type: Opaque
data:
  username: admin
  password: 12345  #需要用Base64编码
```

您也可以通过 Web 界面创建密钥，请参见创建密钥。

使用密钥配置 pod 数据卷

以下两种配置方法您可任选其一进行配置。

密钥可以在Pod中作为文件使用。如示例所示， secret-test密钥的username和password以文件方式保存在*/srt*目录下。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod0
spec:
  containers:
  - name: redis
    image: redis
    volumeMounts:
    - name: srt
      mountPath: "/srt "
      readOnly: true
  volumes:
  - name: srt
    secret:
```

```
      secretName: secret-test
```

1. 通过执行kubectl命令`kubectl apply -f <example0.yaml>`进行配置。

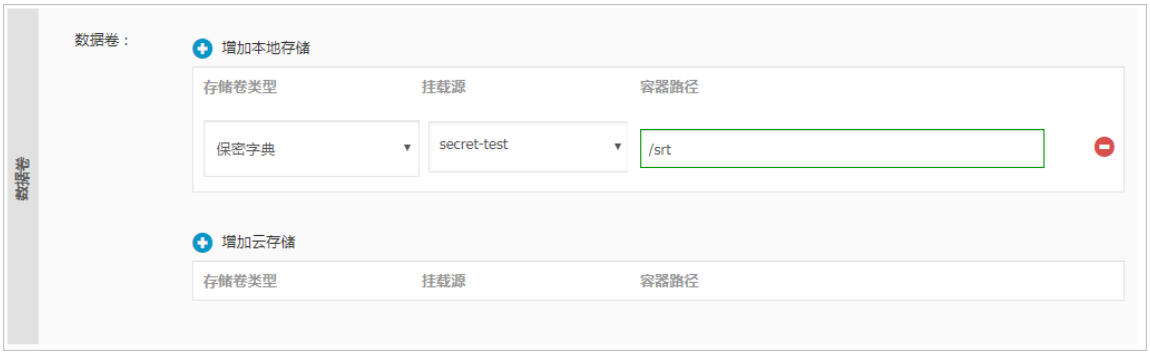
   > **说明:**
   > 此处的*example0.yaml*需要替换成实际yaml文件的名称。

2. 通过容器服务控制台进行配置。

   a. 登录容器服务管理控制台。

   b. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，单击右上角的使用镜像创建。请参考#unique_177。

   本例中，在容器配置页签，在数据卷区域，单击增加本地存储，存储卷类型为保密字典，挂载源为创建密钥中创建好的密钥，容器路径为在容器中访问的路径。



使用密钥设置Pod的环境变量

以下两种配置方法您可任选其一进行配置。

本例中， secret-test密钥中user和password设置为Pod的环境变量。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: redis
    image: redis
    env:
      - name: USERNAME
        valueFrom:
          secretKeyRef:
            name: secret-test
            key: username
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: secret-test
```

```
            key: password
```

1. 通过执行kubectl命令`kubectl apply -f <example1.yaml>`进行配置。

   > **说明:**
   > 此处的*example1.yaml*需要替换成实际yaml文件的名称。

2. 通过容器服务控制台进行配置。

   a. 登录容器服务管理控制台。

   b. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，单击右上角的使用镜像创建。请参考#unique_177。

   本例中，在容器配置页签，在环境变量区域，单击 新增，类型为密钥，变量引用为创建密钥中创建好的密钥，在分别选择使用的key并填写好变量名称。

9.8 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。

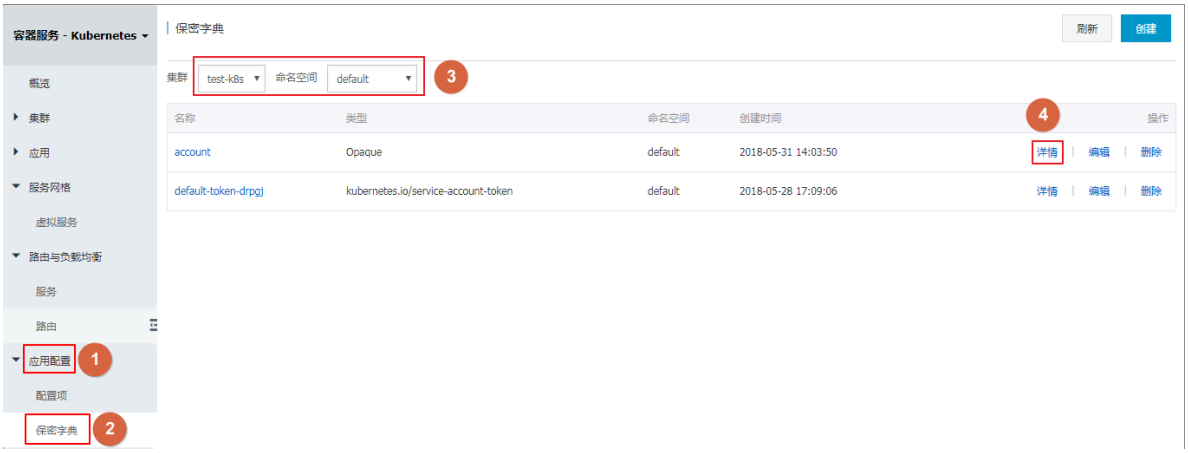
前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已创建一个密钥，参见#unique_179。

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配置 > 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。

9.9 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。

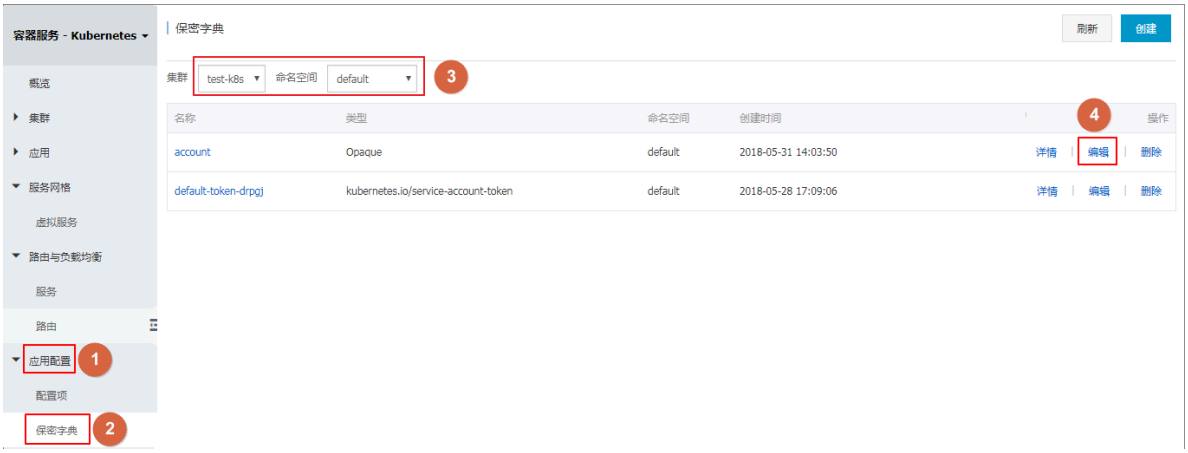
前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已创建一个密钥，参见#unique_179。

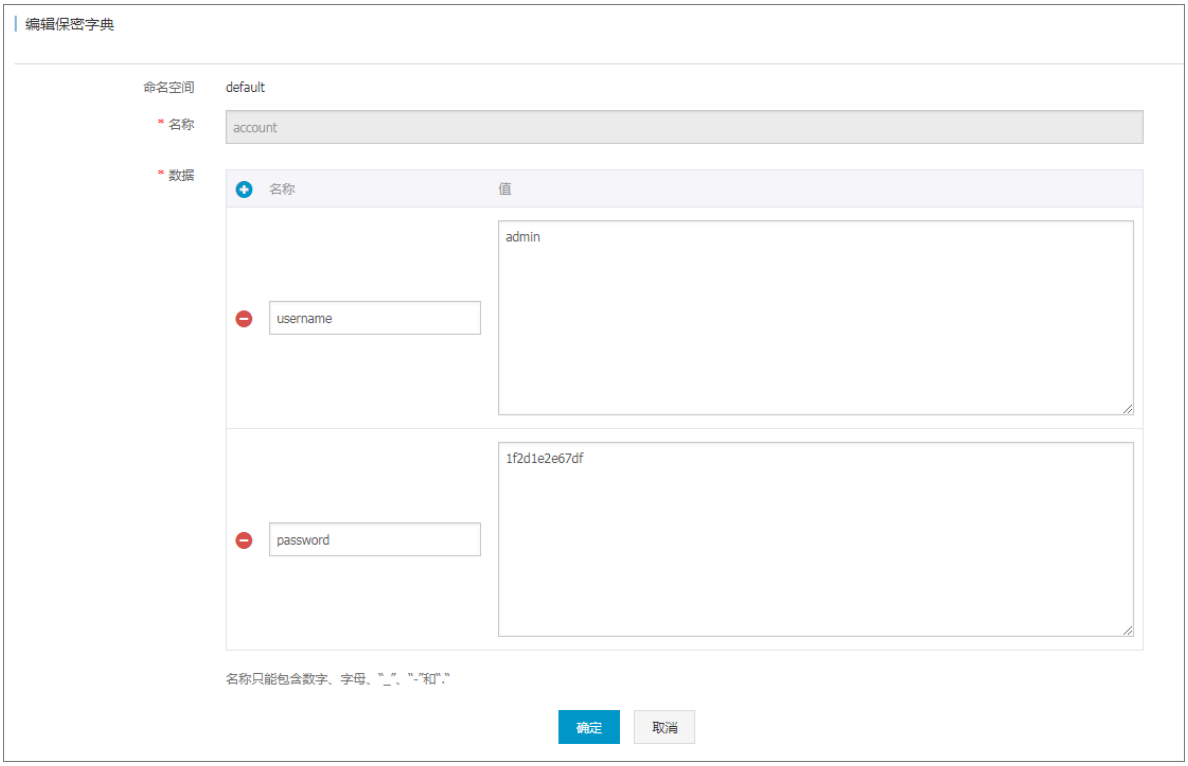
操作步骤

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配置 > 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。

9.10 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#unique_48。
- 您已创建一个密钥，参见#unique_179。

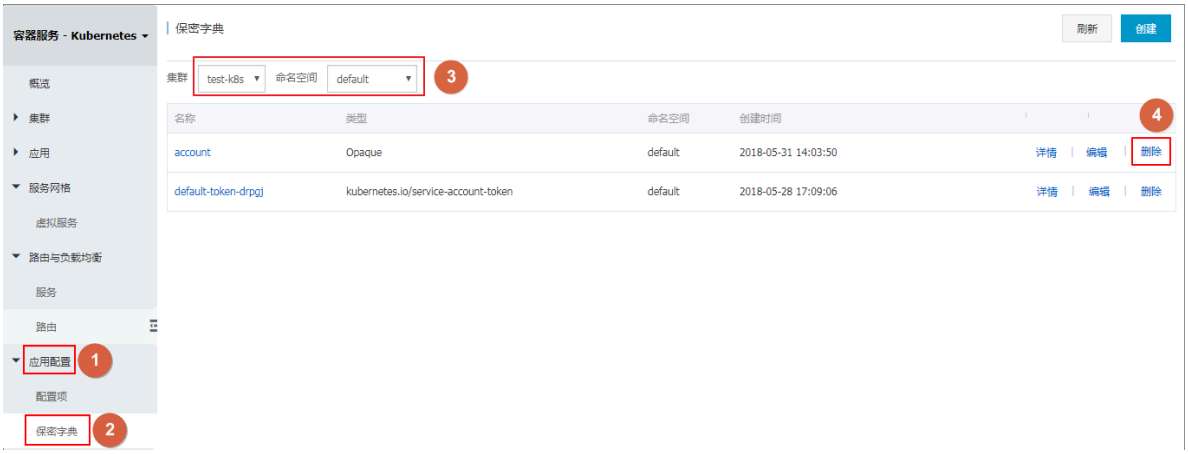
背景信息

说明:

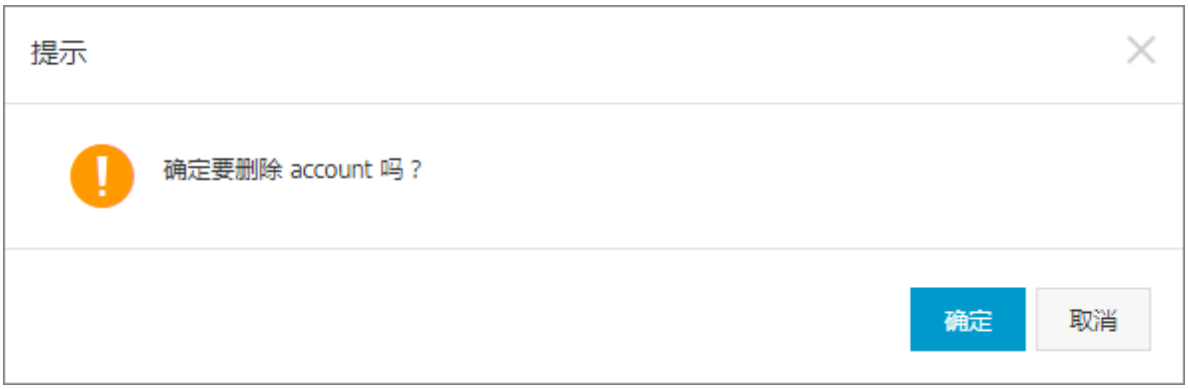
集群创建过程中生成的密钥请勿删除。

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配置 > 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。



10 存储管理-Flexvolume

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 静态存储卷支持以下两种使用方式： · 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	支持
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

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：

- 执行如下命令，输出若干（节点个数）Running 状态的 Pod 列表。

```
kubectl get pod -n kube-system | grep flexvolume
```

- 执行如下命令，输出一个 Running 状态的 Pod 列表。

```
kubectl get pod -n kube-system | grep alicloud-disk-controller
```

安装 Flexvolume

flexvolume在Kubernetes集群中已经默认安装。

如果您的集群中没有安装该插件，可以通过以下yaml文件手动安装。

```
apiVersion: 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:
      - operator: "Exists"
      priorityClassName: system-node-critical
      containers:
      - name: acs-flexvolume
        image: registry.cn-hangzhou.aliyuncs.com/acs/flexvolume:v1.12.
6.16-1f4c6cb-aliyun
        imagePullPolicy: Always
        securityContext:
          privileged: true
        env:
        - name: ACS_DISK
          value: "true"
        - name: ACS_NAS
          value: "true"
```

```
        - name: ACS_OSS
          value: "true"
        - name: ACS_CPFS
          value: "false"
        resources:
          limits:
            cpu: 1000m
            memory: 1000Mi
          requests:
            cpu: 100m
            memory: 100Mi
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - ps -ef |grep /acs/flexvolume | grep monitoring | grep -v
 grep
          failureThreshold: 8
          initialDelaySeconds: 15
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 15
        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/
  updateStrategy:
    type: RollingUpdate
```

安装Disk Controller

Disk Controller在Kubernetes集群中已经默认安装。

如果您的集群中没有安装该插件，可以通过以下yaml文件手动安装。

```
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: alicloud-disk-essd
provisioner: alicloud/disk
parameters:
  type: cloud_essd

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

```
  name: alicloud-disk-efficiency
provisioner: alicloud/disk
parameters:
  type: cloud_efficiency

---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: alicloud-disk-ssd
provisioner: alicloud/disk
parameters:
  type: cloud_ssd

---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: alicloud-disk-available
provisioner: alicloud/disk
parameters:
  type: available

---
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:
      - operator: Exists
      priorityClassName: system-node-critical
      nodeSelector:
         node-role.kubernetes.io/master: &amp;quot;&amp;quot;
      serviceAccount: admin
      serviceAccountName: admin
      containers:
        - name: alicloud-disk-controller
          image: registry.cn-hangzhou.aliyuncs.com/acs/alicloud-disk-
controller:v1.12.6.21-54d91d6-aliyun
          volumeMounts:
            - name: cloud-config
              mountPath: /etc/kubernetes/
            - name: logdir
              mountPath: /var/log/alicloud/
          resources:
            limits:
              cpu: 200m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 100Mi
          livenessProbe:
            exec:
              command:
```

```
              - sh
              - -c
              - ps -ef | grep /alicloud-disk-controller | grep -v grep
            failureThreshold: 8
            initialDelaySeconds: 15
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 15
      volumes:
        - name: cloud-config
          hostPath:
            path: /etc/kubernetes/
        - name: logdir
          hostPath:
            path: /var/log/alicloud/
```

安装Nas Controller

Nas Controller默认不安装，您可以在使用时，通过以下yaml文件手动安装。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: alicloud-nas-controller
  namespace: kube-system
spec:
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: alicloud-nas-controller
    spec:
      tolerations:
      - operator: &amp;quot;Exists&amp;quot;
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
      priorityClassName: system-node-critical
      serviceAccount: admin
      hostNetwork: true
      containers:
        - name: nfs-provisioner
          image: registry.cn-hangzhou.aliyuncs.com/acs/alicloud-nas-
controller:v1.14.3.8-58bf821-aliyun
          env:
          - name: PROVISIONER_NAME
            value: alicloud/nas
          securityContext:
            privileged: true
          volumeMounts:
          - mountPath: /var/log
            name: log
      volumes:
      - hostPath:
          path: /var/log
```

```
        name: log
```

10.3 云盘存储卷

10.3.1 云盘存储卷使用说明

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

使用说明

- 云盘为非共享存储，一个云盘（PVC/PV）只能同时被一个 pod 挂载。
- 推荐使用StatefulSet挂载使用云盘，Deployment挂载云盘时Replica需要为1。
- 使用云盘存储卷前需要先申请一个云盘，并获得磁盘 ID。#unique_187。

  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
  - ESSD云盘：最小20Gi
- 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。
- 待挂载的云盘类型必须是按量付费，包年包月的云盘无法被挂载。当您把集群中的ECS实例从按量付费转换成包年包月时，不可以把云盘一起变成包年包月，否则云盘将不能被挂载使用。

10.3.2 使用静态云盘卷

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，#unique_187。

直接通过 volume 使用

使用*disk-deploy.yaml*文件创建 Pod。

1. 创建并复制以下内容到*disk-deploy.yaml*中。

```
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"
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: d-wz9f6l6nm0uvazgh7y17
          mountPath: /data
      volumes:
        - name: "d-wz9f6l6nm0uvazgh7y17"
          flexVolume:
            driver: "alicloud/disk"
            fsType: "ext4"
            options:
```

```
                volumeId: "d-wz9f6l6nm0uvazgh7y17"
```

2. 执行如下命令，创建Pod。

```
kubectl apply -f disk-deploy.yaml
```

说明:

`volumes`中，`name`的取值建议和`volumeId`的取值保持一致。

如果您所使用的集群是多可用区的模式，使用云盘时需要进行可用区调度，即把 Pod 调度到与云盘相同的可用区。

具体配置为在 yaml 中对 pod 添加 nodeSelector，示例如下：

```
nodeSelector:
        failure-domain.beta.kubernetes.io/zone: cn-hangzhou-b
```

通过 PV/PVC 使用

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。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: d-bp1j17ifxfasvts3****
  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-bp1j17ifxfasvts3****"
```

说明:

pv的名称（name）要与阿里云盘 ID （volumeId）的取值保持一致。

- 通过控制台界面创建云盘数据卷

a. 登录容器服务管理控制台。

b. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 存储卷，进入存储与存储声明页面。

c. 在存储卷页签，选择所需的集群，单击创建。

d. 在创建数据卷对话框中，配置数据卷的相关参数。

- 存储卷类型：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。

e. 完成配置后，单击创建。

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
```

3. 创建 Pod

使用disk-pod.yaml文件创建 pod。

```
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"
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: pvc-disk
          mountPath: /data
      volumes:
        - name: pvc-disk
          persistentVolumeClaim:
            claimName: pvc-disk
```

10.3.3 使用动态云盘卷

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

1. **创建并复制以下内容到*storage-class.yaml*中。**

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
parameters:
  type: cloud_ssd
  regionid: cn-hangzhou
  zoneid: cn-hangzhou-b
reclaimPolicy: Retain
```

2. **执行如下命令，创建StorageClass。**

```
kubectl apply -f storage-class.yaml
```

说明:

参数说明：

- provisioner：配置为 alicloud/disk，标识StorageClass使用阿里云云盘 provisioner 插件创建。
- type：标识云盘类型，支持 cloud、cloud_efficiency、cloud_ssd、cloud_essd、available 四种类型；其中 available 会对高效、ESSD、SSD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。如果数据安全性要求高，推荐使用Retain方式以免误删。
- zoneid：期望创建云盘的可用区。

  如果是多可用区的情况，zoneid可同时配置多个，示例如下：

  ```
  zoneid: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c
  ```

- encrypted：可选参数。创建的云盘是否加密，默认情况是false，创建的云盘不加密。

创建PVC

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-ssd
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: alicloud-disk-ssd-hangzhou-b
  resources:
    requests:
      storage: 20Gi
---
kind: Pod
apiVersion: v1
metadata:
  name: disk-pod-ssd
spec:
  containers:
  - name: disk-pod
    image: nginx
    volumeMounts:
      - name: disk-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: disk-pvc
      persistentVolumeClaim:
        claimName: disk-ssd
```

默认选项：

在多可用区的集群中，需要您手动创建上述 StorageClass，这样可以更准确的定义所需要云盘的可用区信息。

集群默认提供了下面几种 StorageClass，可以在单可用区类型的集群中使用。

- alicloud-disk-efficiency：高效云盘
- alicloud-disk-ssd：SSD云盘。
- alicloud-disk-essd：ESSD云盘。
- alicloud-disk-available：提供高可用选项，优先创建高效云盘。

  如果相应可用区的高效云盘资源售尽，则优先创建SSD盘；如果SSD也售尽，则创建ESSD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: &amp;quot;nginx&amp;quot;
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-ssd
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-ssd
```

```
    spec:
      accessModes: [ &amp;quot;ReadWriteOnce&amp;quot; ]
      storageClassName: &amp;quot;alicloud-disk-ssd-hangzhou-b&amp;quot;
      resources:
        requests:
          storage: 20Gi
```

10.3.4 通过控制台使用动态云盘卷

本文主要为您介绍如何通过控制台使用动态云盘卷。

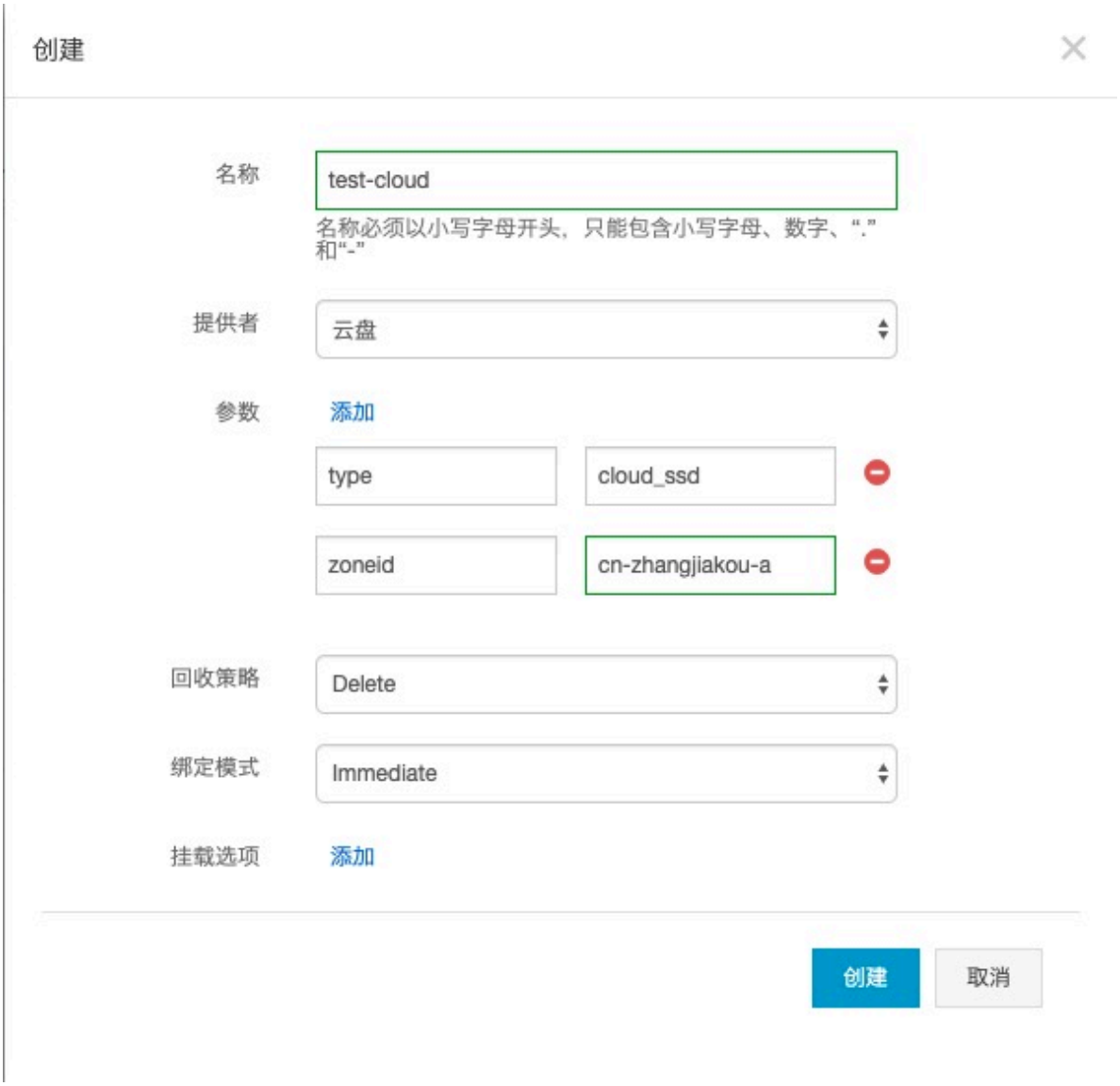
背景信息

您已经成功创建一个Kubernetes 集群，参见#unique_42。

创建StorageClass

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 > 存储卷，进入存储与存储声明页面。

3. 在存储类页签，单击右上角的创建。



- 名称：云盘的名称。
- 提供者：配置为云盘，标识StorageClass使用阿里云云盘 provisioner 插件创建。
- 参数：本例中，包含的示例参数为`type`和`zoneid`。
  - `type`：标识云盘类型，支持
    cloud、cloud_efficiency、cloud_ssd、cloud_essd、available 四种类型；其中 available 会对高效、ESSD、SSD依次尝试创建，直到创建成功。
  - zoneid：期望创建云盘的可用区。

    如果是多可用区的情况，zoneid可同时配置多个，示例如下：

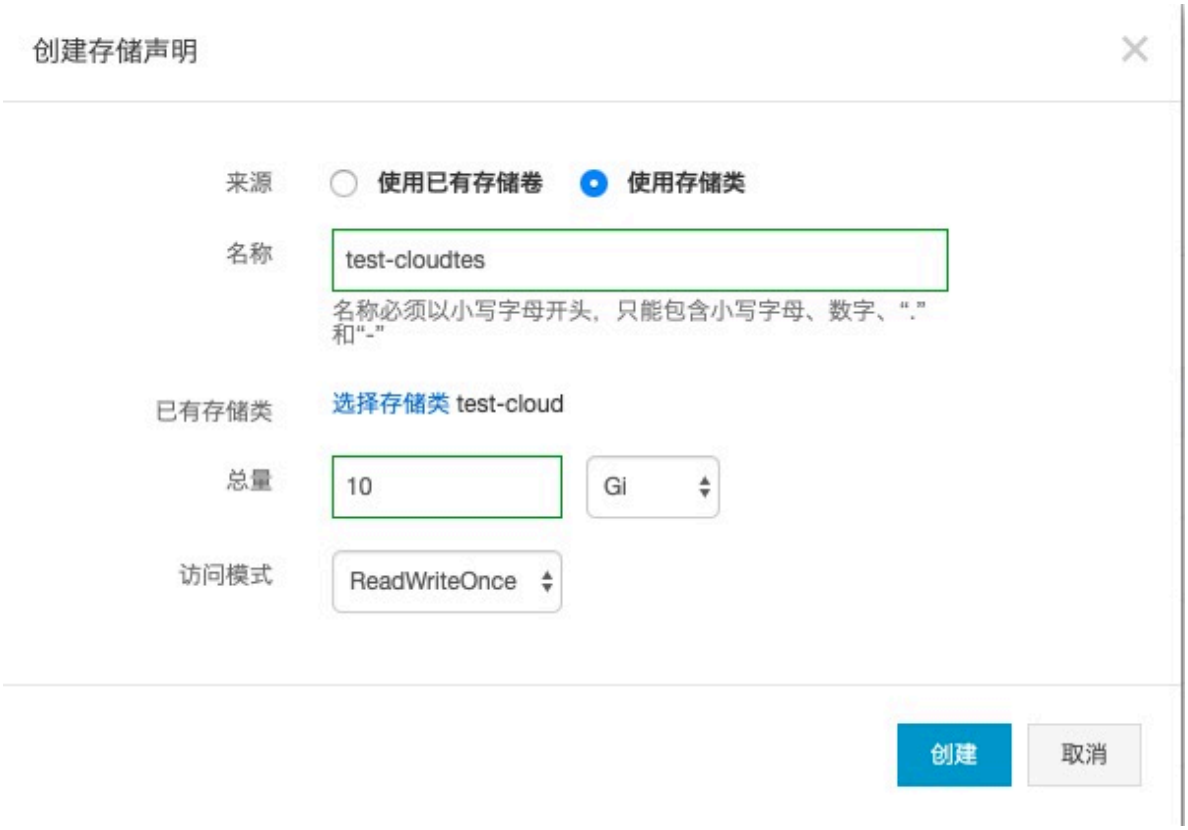
    ```
    zoneid: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c
    ```
  - `encrypted`：可选参数。创建的云盘是否加密，默认情况是false，创建的云盘不加密。

- 回收策略：云盘的回收策略，默认为Delete，支持Retain。如果数据安全性要求高，推荐使用Retain方式以免误删。
- 绑定模式：默认为Immediate，可选值为Immediate、WaitForFirstConsumer。
- 挂载选项：挂载volume时，可选择多种挂载选项。

4. 参数配置完成后，单击创建。

创建PVC

1. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 存储卷，进入存储与存储声明页面。
2. 在存储与声明页签，单击右上角的创建。
3. 在弹出的创建存储声明页面中，填写界面参数。



- 来源：包含使用已有存储卷和使用存储类两种来源。
- 存储声明类型：包含云盘、NAS、OSS三种类型。

说明：

当来源选择使用已有存储卷时，才会出现该选项。

- 名称：创建的数据卷的名称，数据卷名在集群内必须唯一。
- 分配模式：选择已有存储卷。

> 说明：
> 当来源选择使用已有存储卷时，才会出现该选项。

- 已有存储类：单击选择存储类，在目标存储卷右侧操作列单击选择，选择存储类。
- 总量：所创建存储卷的容量。

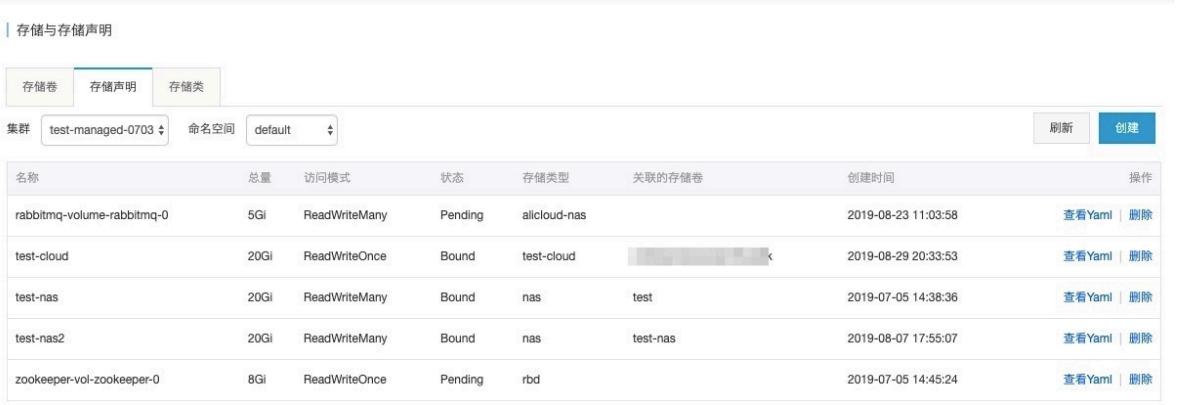
> 说明：
> 所创建的存储卷容量不能超过磁盘容量。

- 访问模式：默认为ReadWriteOnce。

> 说明：
> 当来源选择使用存储卷时，才会出现该选项。

4. 单击创建。

   创建成功后可以在列表中看到test-cloud，并且已绑定相应的存储卷。



使用数据卷

1. 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。
2. 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。

3. 设置容器配置。

   本例主要为您介绍数据卷的配置。其他界面参数请参见镜像创建无状态Deployment应用。

   数据卷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。

4. 所有的信息都配置完成后，单击创建。

   创建成功后，您就可以正常使用数据卷。

您也可以通过命令行的方式使用动态云盘卷，请参见#unique_191。

10.4 NAS储存卷

10.4.1 NAS存储卷使用说明

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 支持两种 Kubernetes 挂载驱动：

- flexvolume挂载驱动
  - 静态存储卷挂载
    - 通过volume方式挂载
    - 通过PV/PVC方式挂载
  - 动态存储卷挂载
- NFS挂载驱动
  - 静态存储卷挂载
    - 通过volume方式挂载
    - 通过PV/PVC方式挂载
  - 动态存储卷挂载

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挂载点。创建的 NAS 文件系统挂载点需要和您的集群位于同一 VPC。登录文件存储管理控制台，可以创建一个 NAS 文件系统和挂载点。

例如，您的挂载点为`055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com`格式。

注意事项

- NAS 为共享存储，可以同时为多个 Pod 提供共享存储服务，即一个PVC可以同时被多个POD使用。
- 在没有卸载NAS文件系统前，务必不要删除NAS挂载点，否则会造成操作系统Hang。
- NAS挂载点创建后，等待一定时间，待挂载点Ready后方可使用。
- 数据卷挂载协议推荐使用NAS v3。
- 使用NAS数据卷前，推荐升级Flexvolume到最新版本。

使用场景

- NFS驱动

  Kubernetes默认提供该驱动，当您的集群没有Flexvolume驱动时，可以使用NFS驱动。
- Flexvolume驱动

  Flexvolume驱动相比较NFS驱动，添加了一些优化配置，例如，子目录创建、挂载参数配置等。
- 静态存储卷

  NAS为共享存储，多数场景都可以通过静态存储卷挂载满足您的需求。

- 动态存储卷

  当为不同的用户（应用）提供不同的nas子目录进行挂载时，可以考虑使用nas动态存储卷。

  当使用SatefulSet部署应用，每个pod配置使用不同nas数据卷时，可以使用nas动态存储卷。

10.4.2 Flexvolume驱动-静态NAS卷

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。

背景信息

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。

参数说明：

- 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。极速NAS的Path需要以 */share* 开头。
- vers：定义 NFS 挂载协议的版本号，支持v3 和 v4.0，默认使用版本v3且推荐使用此版本。
- 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。
- options：定义挂载参数，不配置此项时，v3版本的挂载默参数为`nolock,tcp,noresvport`，v4.0版本挂载的默认参数为`noresvport`。

直接通过 volume 使用

使用*nas-deploy.yaml*文件创建 Pod。

1. 创建并复制以下内容到*nas-deploy.yaml*中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
```

```
        - containerPort: 80
        volumeMounts:
          - name: nas1
            mountPath: "/data"
      volumes:
      - name: "nas1"
        flexVolume:
          driver: "alicloud/nas"
          options:
            server: "0cd8b4a576-grs79.cn-hangzhou.nas.aliyuncs.com"
            path: "/k8s"
            vers: "3"
            options: "nolock,tcp,noresvport"
```

2. 执行如下命令，创建Pod。

```
kubectl apply -f nas-deploy.yaml
```

说明:

`volumes`中，`name`的取值建议和`volumeId`的取值保持一致。

如果您所使用的集群是多可用区的模式，使用云盘时需要进行可用区调度，即把 Pod 调度到与云盘相同的可用区。

具体配置为在 yaml 中对 pod 添加 nodeSelector，示例如下：

```
nodeSelector:
        failure-domain.beta.kubernetes.io/zone: cn-hangzhou-b
```

通过 PV/PVC 使用

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。

   - 通过 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: "3"
```

```
      options: "nolock,tcp,noresvport"
```

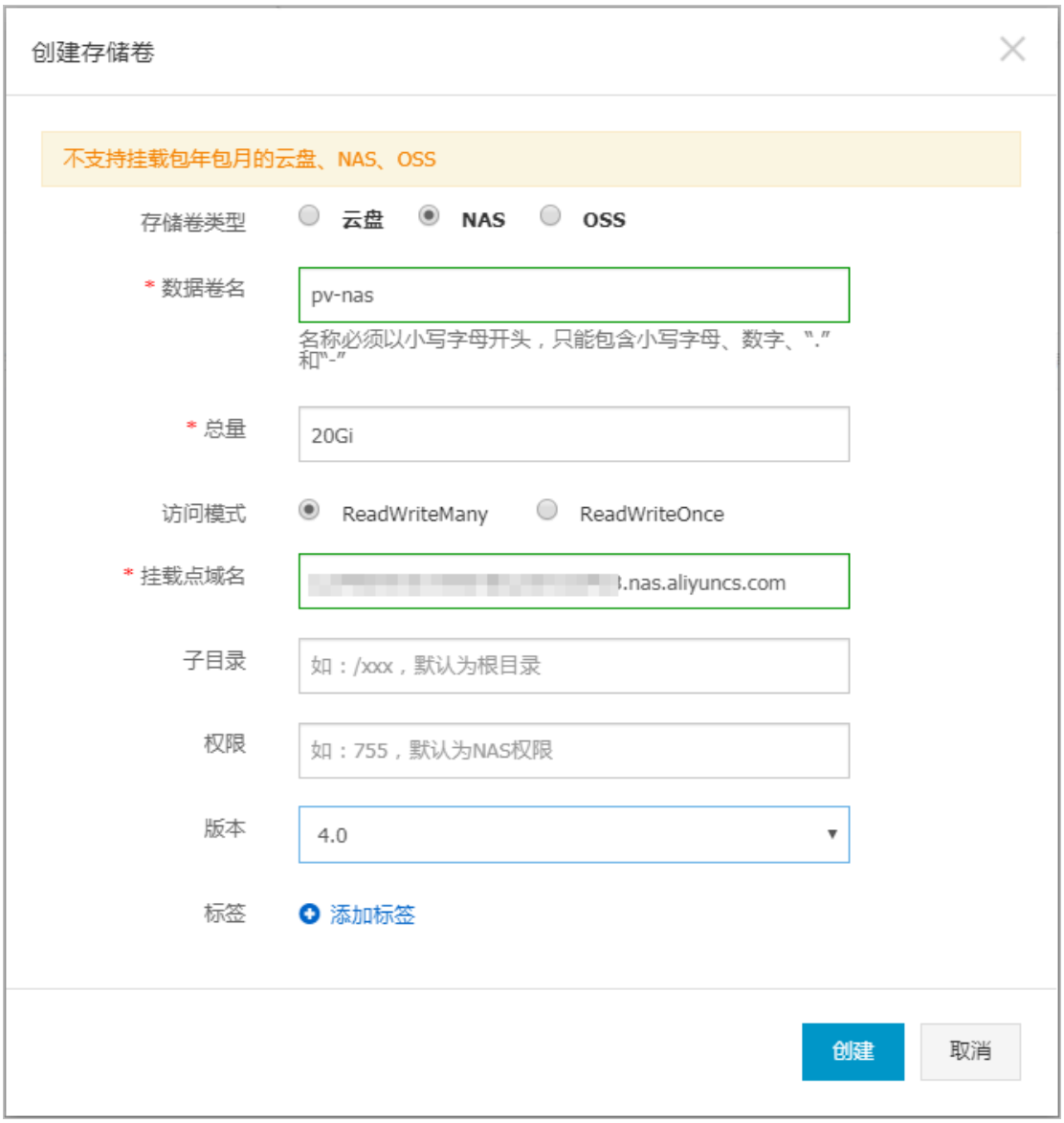
- 通过控制台界面创建云盘数据卷

a. 登录容器服务管理控制台。

b. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 存储卷，进入存储与存储声明页面。

c. 在存储卷页签，选择所需的集群，单击创建。

d. 在创建存储卷对话框中，配置存储卷的相关参数。



- 存储卷类型：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。
- 总量：所创建存储卷的容量。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。
  - 极速NAS需要以*/share*开头。
- 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。
  - 当挂载的目录文件量较大时不建议此配置，否则会出现chmod长时间执行。
- 版本：所创建存储卷的版本。
- 标签：为该存储卷添加标签。

e. 完成配置后，单击创建。

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
```

3. 创建 Pod

使用*nas-pod.yaml*文件创建 pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

```
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: &amp;quot;/data&amp;quot;
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: pvc-nas
```

10.4.3 Flexvolume驱动-动态NAS卷

本方案实现的动态NAS卷，是在某个NAS文件系统下通过创建子目录、并把子目录映射为一个动态PV提供给应用。

前提条件

使用此方案，需要在集群中部署Flexvolume驱动（Kubernetes集群默认已部署该驱动）。

您已经部署了alicloud-nas-controller。请参见#unique_196。

创建动态NAS卷

1. 配置StorageClass。

   配置示例如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-nas
mountOptions:
- nolock,tcp,noresvport
- vers=3
parameters:
  server: "23a9649583-iaq37.cn-shenzhen.nas.aliyuncs.com:/nasroot1/"
  driver: flexvolume
provisioner: alicloud/nas
reclaimPolicy: Delete
```

   说明：

   - mountOptions：表示生成的pv options 配置，挂载NAS卷时使用这个options挂载。
   - server：表示生成目标PV所使用NAS挂载点列表。格式为 *nfsurl1:/path1,nfsurl2:/path2*；当配置多个server时，通过此storageclass创建的PV会轮询使用上述server作为配置参数；极速NAS配置路径需要以*/share*开头。
   - driver：支持Flexvolume、NFS两种驱动，默认为NFS。

- reclaimPolicy：PV的回收策略，建议配置为Retain。
  - 当配置为Delete时，删除PV后NAS文件系统中的对应目录会默认修改名字（例如，*path-name*会被修改为*archived-path-name*）。
  - 如果需要删除文件系统中对应的存储目录，可在storageclass中配置`archiveOnDelete`为`false`。

2. 使用动态卷。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 5
  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: "/data"
```

```
            name: html
```

10.4.4 NFS驱动-静态NAS卷

使用 NFS 驱动，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。

通过 volume 方式使用

使用 *nas-deploy.yaml* 文件创建 Pod。

1. **创建并复制以下内容到*nas-deploy.yaml*中。**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: nas1
            mountPath: "/data"
      volumes:
      - name: "nas1"
        nfs:
          server: "0cd8b4a576-grs79.cn-hangzhou.nas.aliyuncs.com"
          path: "/k8s"
```

说明:

通过Volume方式使用NFS驱动挂载NAS，目前不支持配置Options，即不能控制挂载的版本和可选参数。

2. 执行如下命令，创建Pod。

```
kubectl apply -f nas-deploy.yaml
```

使用 PV/PVC方式挂载。

a) 创建 PV。

示例模板如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nas
spec:
  capacity:
    storage: 8Gi
  accessModes:
    - ReadWriteMany
  mountOptions:
    - nolock,tcp,noresvport
    - nfsvers=3
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /
    server: 055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com
```

b) 创建 PVC。

创建一个 PersistentVolumeClaim 来请求绑定该 PersistentVolume。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
     name: nasclaim
spec:
     accessModes:
       - ReadWriteMany
     resources:
       requests:
         storage: 8Gi
```

c) 创建Pod。

创建一个应用来申明挂载使用该数据卷。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

```
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: nasclaim
```

至此，您就将 NAS 远程文件系统挂载到您的 Pod 应用中了。

10.4.5 NFS驱动-动态NAS卷

本方案实现的动态nas卷，是在某个nas文件系统下通过创建子目录、并把子目录映射为一个动态pv提供给应用。

前提条件

您已经部署了alicloud-nas-controller。请参见#unique_196。

创建动态NAS卷

1. 配置StorageClass。

   配置示例如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-nas
mountOptions:
- nolock,tcp,noresvport
- vers=3
parameters:
  server: "23a9649583-iaq37.cn-shenzhen.nas.aliyuncs.com:/nasroot1/"
  driver: nfs
provisioner: alicloud/nas
reclaimPolicy: Retain
```

   说明：

   - mountOptions：表示生成的pv options 配置，挂载NAS卷时使用这个options挂载。
   - server：表示生成目标PV所使用NAS挂载点列表。格式为 *nfsurl1:/path1,nfsurl2:/path2*；当配置多个server时，通过此storageclass创建的PV会轮询使用上述server作为配置参数；极速NAS配置路径需要以*/share*开头。
   - driver：支持Flexvolume、NFS两种驱动，默认为NFS。

- reclaimPolicy：PV的回收策略，建议配置为Retain。
  - 当配置为Delete时，删除PV后NAS文件系统中的对应目录会默认修改名字（例如，*path-name*会被修改为*archived-path-name*）。
  - 如果需要删除文件系统中对应的存储目录，可在storageclass中配置`archiveOnDelete`为`false`。

2. 使用动态卷。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 5
  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: "/data"
          name: html
```

10.5 OSS储存卷

10.5.1 OSS存储卷使用说明

您可以在阿里云容器服务 Kubernetes 集群中使用阿里云 OSS 数据卷。

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，挂载节点和bucket相同region时，可使用内网地址。
- 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#unique_201。使用 OSS 数据卷必须在部署 Flexvolume 服务的时候创建 Secret，并输入AccessKey 信息。
- 早期的Flexvolume版本挂载的OSS卷，在容器服务Kubernetes集群升级、kubelet重启过程中会有ossfs重启操作，OSS存储卷会重新挂载，使用OSS存储卷的Pod在升级后需要重建，可以通过升级Flexvolume解决。

10.5.2 使用OSS静态卷

本文为您介绍使用OSS静态存储卷的几种方式。

直接使用 volume 方式

1. 创建并复制以下内容到*oss-deploy.yaml*中。

```
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"
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - cd /data
          initialDelaySeconds: 30
          periodSeconds: 30
      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"
```

2. **执行如下命令，创建Pod。**

```
kubectl apply -f oss-deploy.yaml
```

使用 PV/PVC

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 存储卷

  a. 登录容器服务管理控制台。

  b. 在 Kubernetes 菜单下，单击左侧导航栏中的集群 > 存储卷，进入存储与存储声明页面。

  c. 在存储卷页签，选择所需的集群，单击创建。

  d. 在创建数据卷对话框中，配置数据卷的相关参数。

  - 存储卷类型：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。
  - 总量：所创建存储卷的容量。
  - 访问模式：默认为 ReadWriteMany。
  - AccessKey ID、AccessKey Secret：访问 OSS 所需的 AccessKey。
  - 可选参数：您可以为OSS存储卷输入定制化参数，格式为：`-o *** -o ***`。
  - 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：为该存储卷添加标签。

  e. 完成配置后，单击创建。

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
```

3. 创建 Pod。

   使用*oss-deploy.yaml*创建 Pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oss-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-oss
            mountPath: "/data"
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - cd /data
          initialDelaySeconds: 30
          periodSeconds: 30
        volumes:
        - name: pvc-oss
          persistentVolumeClaim:
            claimName: pvc-oss
```

通过Secret方式使用AccessKey

1. 执行如下命令，创建Secret。

```
# kubectl create secret generic osssecret --from-literal=akId='
111111' --from-literal=akSecret='2222222' --type=alicloud/oss -n
default

akId: 为用户的Access Key ID;
akSecret: 为用户的Access Key Secret;
secret类型: 配置为alicloud/oss
名词空间: 需要与期望的pod相同在相同namespace;
```

```
osssecret: 为secret的名字，可以自行配置；
```

2. 在PV中使用Secret。

   在PV中通过secretRef字段定义Secret。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-oss
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  flexVolume:
    driver: "alicloud/oss"
    secretRef:
      name: "osssecret"
    options:
      bucket: "docker"
      url: "oss-cn-hangzhou.aliyuncs.com"
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

3. 在Volume中使用Secret。

   Volume中通过secretRef字段定义Secret。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-oss-deploy1
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-flexvolume-oss
        image: nginx
        volumeMounts:
          - name: "oss1"
            mountPath: "/data"
            subPath: "hello"
      volumes:
        - name: "oss1"
          flexVolume:
            driver: "alicloud/oss"
            secretRef:
              name: "osssecret"
            options:
              bucket: "aliyun-docker"
              url: "oss-cn-hangzhou.aliyuncs.com"
              otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

说明:

使用Secret方式配置AccessKey时，应用（Pod）所在的Namespace要和Secret所在的Namespace一致。

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#unique_48。
- 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#unique_119。

  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。

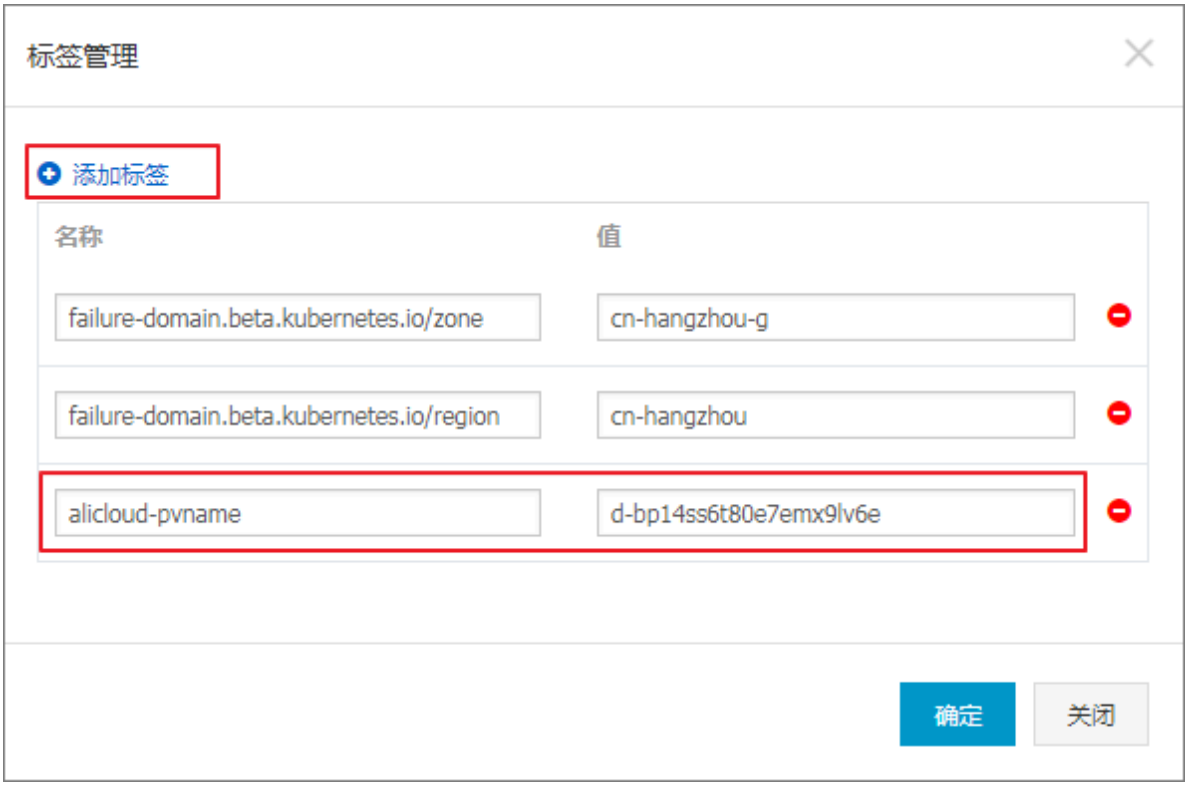
4. 在创建存储声明对话框中进行配置，最后单击创建。

- 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。

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#unique_119。

前提条件

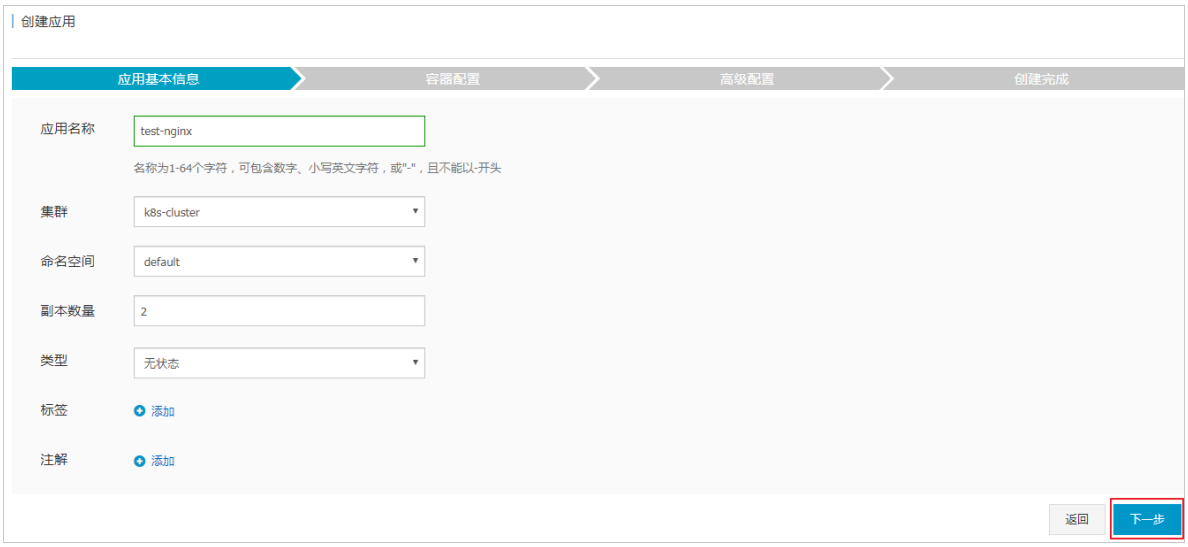
- 您已创建一个Kubernetes集群，参见#unique_48。
- 您已创建一个存储卷声明，本例中使用云盘创建一个云盘存储卷声明pvc-disk，参见#unique_108。

操作步骤

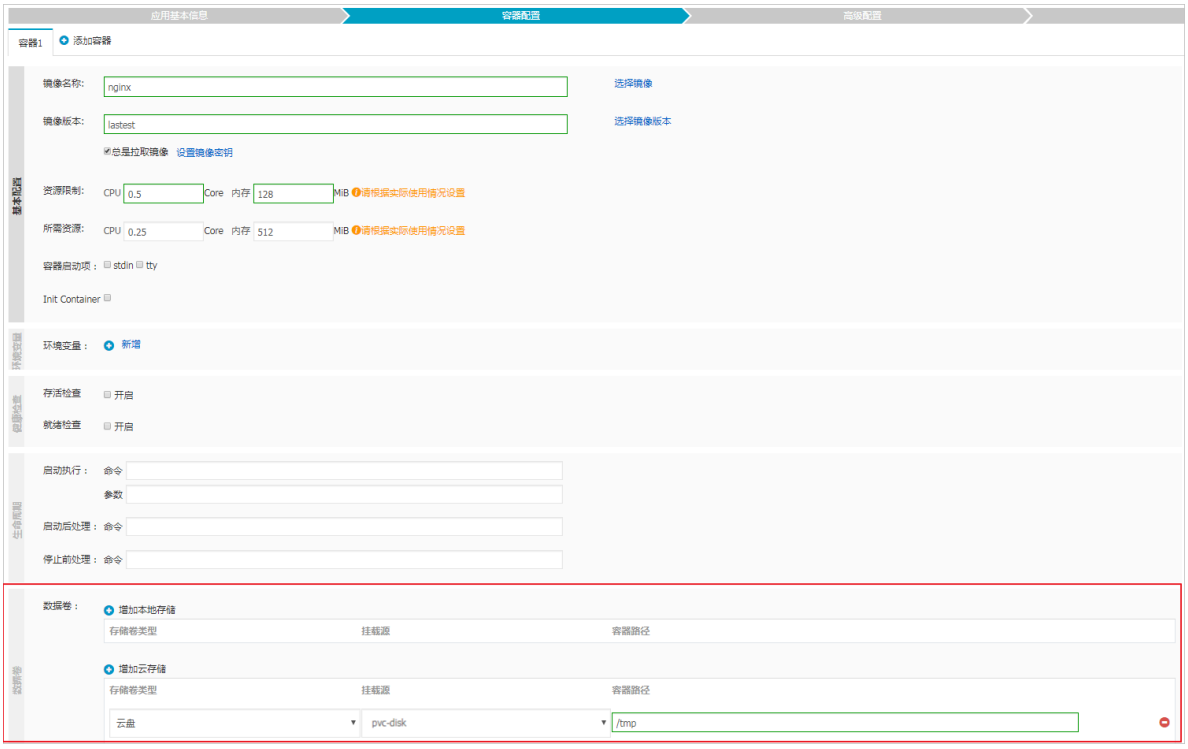
1. 登录容器服务管理控制台。

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



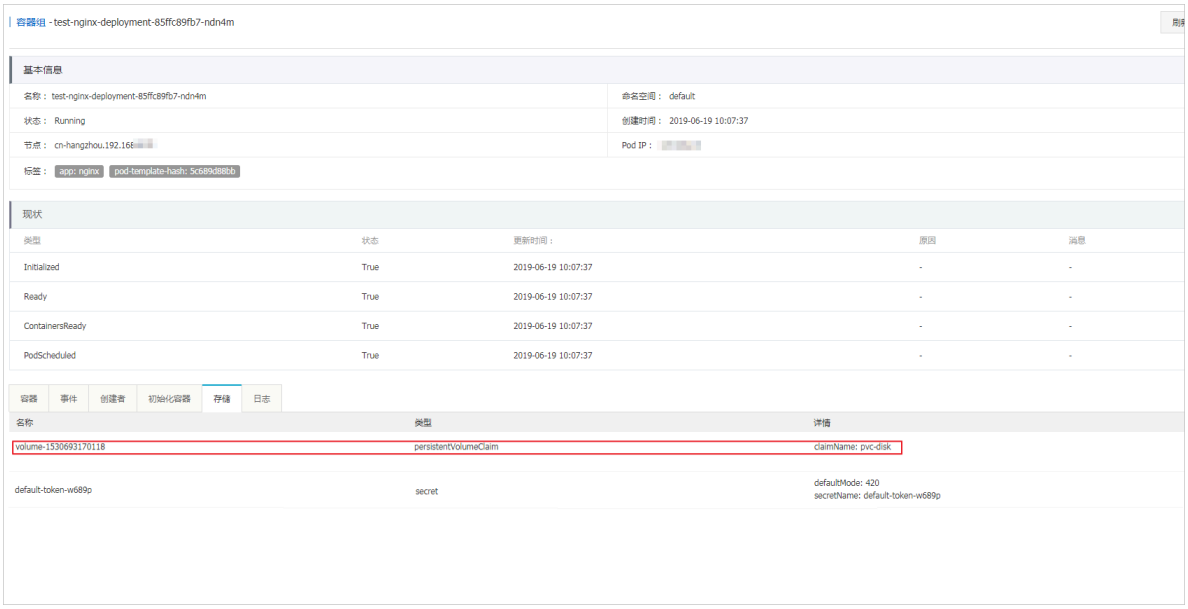
4. 在应用配置页面，选择镜像，然后配置云存储类型的数据卷，支持云盘/NAS/OSS 三种类型。本例中使用准备好的云盘存储卷声明，最后单击下一步。



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。



10.8 存储卷常见问题

存储卷挂载不上

检查Flexvolume是否安装

在master节点上执行下面命令。

```
# kubectl get pod -n kube-system | grep flexvolume

flexvolume-4wh8s              1/1       Running   0          8d
flexvolume-65z49              1/1       Running   0          8d
flexvolume-bpc6s              1/1       Running   0          8d
flexvolume-l8pml              1/1       Running   0          8d
flexvolume-mzkpv              1/1       Running   0          8d
flexvolume-wbfhv              1/1       Running   0          8d
flexvolume-xf5cs              1/1       Running   0          8d
```

查看flexvolume pod状态是否为Running，且运行的数量与节点数量相同；

如果没有安装，请参考#unique_206。

如果运行状态不对，请参考插件运行日志分析。

检查动态存储插件是否安装

如果使用云盘的动态存储功能，需要确认是否安装动态存储插件，执行下面命令。

```
# kubectl get pod -n kube-system | grep alicloud-disk

alicloud-disk-controller-8679c9fc76-lq6zb     1/1 Running   0   7d
```

如果没有安装，请参考#unique_206。

如果运行状态不对，请参考插件运行日志分析。

如何查看存储相关日志

Flexvolume日志（master1上执行）

执行get命令查看出错的pod：

```
# kubectl get pod -n kube-system | grep flexvolume
```

执行log命令，查看出错pod的日志：

```
# kubectl logs flexvolume-4wh8s -n kube-system
# kubectl describe pod flexvolume-4wh8s -n kube-system

#在pod描述最后若干行是pod运行状态的描述，可以根据描述分析错误；
```

云盘、NAS、OSS驱动日志查看：

```
# 查看host节点上持久化的日志；
# 如果某个Pod挂载失败，查看pod所在的节点地址：

# kubectl describe pod nginx-97dc96f7b-xbx8t | grep Node
Node: cn-hangzhou.i-bp19myla3uvnt6zihejb/192.168.247.85
Node-Selectors:  <none>

# 登录节点，查看日志：

# ssh 192.168.247.85
# ls /var/log/alicloud/flexvolume*
flexvolume_disk.log  flexvolume_nas.log  flexvolume_o#ss.log

可以看到云盘、nas、oss挂载的日志；
```

provsioner插件日志（master1上执行）

执行get命令查看出错的pod：

```
# kubectl get pod -n kube-system | grep alicloud-disk
```

执行log命令，查看出错pod的日志：

```
# kubectl logs alicloud-disk-controller-8679c9fc76-lq6zb -n kube-
system
```

```
# kubectl describe pod alicloud-disk-controller-8679c9fc76-lq6zb -n
kube-system

#在pod描述最后若干行是pod运行状态的描述，可以根据描述分析错误；
```

Kubelet日志

```
# 如果某个Pod挂载失败，查看pod所在的节点地址：

# kubectl describe pod nginx-97dc96f7b-xbx8t | grep Node
Node: cn-hangzhou.i-bp19myla3uvnt6zihejb/192.168.247.85
Node-Selectors:  <none>

# 登录节点，查看kubelet日志：

# ssh 192.168.247.85
# journalctl -u kubelet -r -n 1000 &> kubelet.log

# -n的值表示期望看到的日志行数；
```

上述为获取flexvolume、provsioner、Kubelet错误日志的方法，如果无法根据日志修复状态，可以附带日志信息联系阿里云技术支持；

云盘常见问题

云盘挂载失败，出现timeout错误

如果节点为手动添加，可能是由于sts权限的问题导致，需要手动配置RAM权限：#unique_207。

云盘挂载失败，出现Size错误

创建云盘对Size有如下要求。

说明:

- 普通云盘：最小5Gi
- 高效云盘：最小20Gi
- SSD云盘：最小20Gi

云盘挂载失败，出现zone错误

ECS挂载云盘时，必须在同一个region下面的相同zone内，否则不能挂载成功。

升级系统后，云盘有时报错：input/output error

1. 升级flexvolume到v1.9.7-42e8198或以后版本。
2. 对于已经出问题的Pod，需要重建。

升级命令：

```
# kubectl set image daemonset/flexvolume acs-flexvolume=registry.cn-hangzhou.aliyuncs.com/acs/flexvolume:v1.9.7-42e8198 -n kube-system
```

Flexvolume版本信息：可登录容器镜像服务控制台，单击左侧导航栏中的镜像搜索，搜索acs/flexvolume，获取Flexvolume最新版本信息。

NAS常见问题

NAS挂载时间太长

如果NAS卷包含的文件量很大，且在挂载模板中配置了chmod参数，可能导致挂载时间过长的问题；可以去掉chmod参数。

NAS挂载失败，出现timeouot错误

检查nas挂载点和集群是否在同一个vpc内，否则无法挂载。

OSS常见问题

oss挂载失败

- 检查使用的ak是否正确。
- 检查OSS挂载使用的URL是否网络可达。

集群升级后，容器内oss挂载目录不可用

升级集群、重启kubelet的时候，由于容器网络会重启，导致ossfs进程重启。

ossfs进程重启会导致主机与容器目录映射失效，这时需要重启容器，或重建Pod。

可以通过配置健康检查实现容器或Pod的自动重启，请参考#unique_208。

10.9 存储卷最佳实践

10.9.1 有状态服务-静态云盘使用最佳实践

本文为您介绍有状态服务-静态云盘的常见使用场景及方法。

背景信息

云盘的使用场景包括：

- 对磁盘I/O要求高的应用，且没有共享数据的需求，如MySQL、Redis等数据存储服务。
- 高速写日志。
- 持久化存储数据，不因Pod生命周期的结束而消失。

静态云盘的使用场景：

已经购买了云盘实例的情况。

静态云盘使用方式：

需手动创建PV及PVC。

前提条件

- 您已成功创建一个Kubernetes集群，参见#unique_48。
- 您已成功创建一个云盘，参见#unique_211。
- 您可以通过kubectl连接到Kubernetes集群，参见#unique_37。

使用限制

- 云盘为阿里云存储团队提供的非共享存储，只能同时被一个Pod挂载。
- 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。

创建PV

1. 创建*pv-static.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <your-disk-id>
  labels:
    alicloud-pvname: <your-disk-id>
    failure-domain.beta.kubernetes.io/zone: <your-zone>
    failure-domain.beta.kubernetes.io/region: <your-region>
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "alicloud/disk"
    fsType: "ext4"
    options:
      volumeId: "<your-disk-id>"
```

说明：

- `alicloud-pvname: <your-disk-id>`：PV的名称。需要与volumeID云盘ID一致。
- `failure-domain.beta.kubernetes.io/zone: <your-zone>`：云盘所在的可用区。例如：cn-hangzhou-b。
- `failure-domain.beta.kubernetes.io/region: <your-region>`：云盘所在的地域。例如：cn-hangzhou。

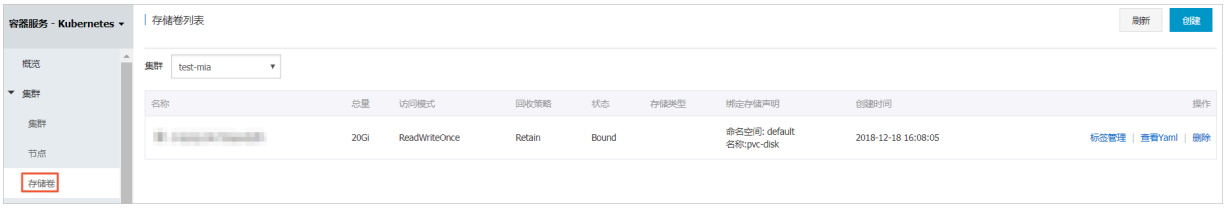
`failure-domain.beta.kubernetes.io/zone`和`failure-domain.beta.kubernetes.io/region`，在多可用区的情况下，必须配置，可保证您的Pod调度到云盘所在的可用区。

2. 执行以下命令，创建PV。

```
$ kubectl create -f pv-static.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的集群 > 存储卷，进入存储卷列表页面，选择目标集群，可以看到刚刚创建的PV。



创建PVC

创建云盘存储声明PVC，使用`selector`筛选PV，精确配置PVC和PV的绑定关系。

1. 创建*pvc-static.yaml*文件。

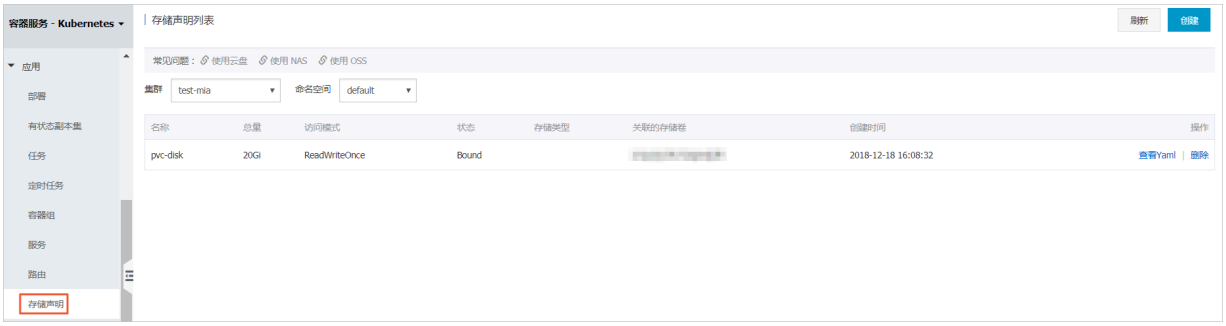
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  selector:
    matchLabels:
      alicloud-pvname: <your-disk-id>
```

2. 执行以下命令，创建PVC。

```
$ kubectl create -f pvc-static.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 存储声明，进入存储声明列表页面，选择目标集群和命名空间，可以看到刚刚创建的PVC。



创建应用

1. 创建*static.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-static
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
          persistentVolumeClaim:
            claimName: pvc-disk
```

2. 执行以下命令，创建Deployment。

```
$ kubectl create -f static.yaml
```

预期结果

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，选择目标集群和命名空间，可以看到刚刚创建的deployment。

静态云盘的持久化存储

1. **执行以下命令，查看部署的deployment所在Pod的名称。**

```
$ kubectl get pod | grep static
nginx-static-78c7dcb9d7-g9lll   2/2     Running     0          32s
```

2. **执行以下命令，查看*/data*路径下是否挂载了新的云盘。**

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll df | grep data
/dev/vdf        20511312    45080  20449848   1% /data
```

3. **执行以下命令，查看*/data*路径下的文件。**

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
lost+found
```

4. **执行以下命令，在*/data*路径下创建文件*static*。**

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll touch /data/static
```

5. **执行以下命令，查看*/data*路径下的文件。**

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
static
lost+found
```

6. **执行以下命令，删除名称为`nginx-static-78c7dcb9d7-g9lll`的Pod。**

```
$ kubectl delete pod nginx-static-78c7dcb9d7-g9lll
pod "nginx-static-78c7dcb9d7-g9lll" deleted
```

7. **同时在另一个窗口中，执行以下命令，查看Pod删除及Kubernetes重建Pod的过程。**

```
$ kubectl get pod -w -l app=nginx
NAME                            READY   STATUS    RESTARTS   AGE
nginx-static-78c7dcb9d7-g9lll   2/2     Running   0          50s
nginx-static-78c7dcb9d7-g9lll   2/2   Terminating   0     72s
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     0s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     73s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     5s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-h6brd   0/2   PodInitializing   0     6s
nginx-static-78c7dcb9d7-h6brd   2/2   Running   0     8sg   0     8s
```

8. **执行以下命令，查看Kubernetes重建的Pod名称。**

```
$ kubectl get pod
NAME                            READY   STATUS      RESTARTS   AGE
```

```
nginx-static-78c7dcb9d7-h6brd   2/2     Running     0          14s
```

9. 执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*static*并没有被删除，说明静态云盘的数据可持久保存。

```
$ kubectl exec nginx-static-78c7dcb9d7-h6brd ls /data
static
lost+found
```

10.9.2 如何使用BYOK创建加密云盘

阿里云容器服务Kubernetes集群如何使用BYOK创建加密云盘。

前提条件

使用BYOK创建加密云盘时，系统需要使用同一地域的密钥管理服务（KMS）提供的BYOK（Bring Your Own Key）。因此，首次通过控制台或者API使用云盘加密功能之前，您必须先开通密钥管理服务请参见#unique_213

背景信息

当您的业务因为安全需求或法规合规要求等原因，需要对存储在云盘上的数据进行加密保护时，您可以使用阿里云ECS云盘加密功能，无需构建、维护和保护自己的密钥管理基础设施，即可保护数据的隐私性和自主性。

本文中的云盘指代云盘和共享块存储中的数据盘。下文中统一简称为云盘，除非特别指出。

创建BYOK

创建BYOK并记录密钥ID。

1. 登录KMS 控制台。
2. 在控制台左上角处，单击创建密钥。
3. 在弹出的创建密钥对话框中，填写别名和描述。
4. 在高级选项下，选择密钥材料来源。
   - 选择阿里云KMS：密钥材料将由KMS生成。
   - 选择外部：KMS将不会生成密钥材料，您需要将自己的密钥材料导入。详情请参见#unique_214。
5. 单击确定。

创建StorageClass使用BYOK

在容器服务Kubernetes集群中新建StorageClass，并配置`encrypted: "true" kmsKeyId: <your BYOK>`。

1. 通过 kubectl 连接 Kubernetes 集群。
2. 执行如下命令，打开您的*storageclass-ssd-byok.yaml*文件。

```
$ cat storageclass-ssd-byok.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-disk-ssd-byok
parameters:
  type: cloud_ssd
  encrypted: "true"
  kmsKeyId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
provisioner: alicloud/disk
reclaimPolicy: Delete
```

说明:

kmsKeyId需要替换成创建BYOK中填写的密钥ID。

3. 执行如下命令，创建StorageClass。

```
kubectl apply -f storageclass-ssd-byok.yaml
```

4. 执行如下命令，查看StorageClass。

```
$ kubectl get sc

NAME                          PROVISIONER     AGE
alicloud-disk-available       alicloud/disk   19m
alicloud-disk-efficiency      alicloud/disk   19m
alicloud-disk-essd            alicloud/disk   19m
alicloud-disk-ssd             alicloud/disk   19m
alicloud-disk-ssd-byok        alicloud/disk   28s
```

创建示例应用并挂载云盘

1. 执行如下命令，打开您的*deploy.yaml*文件。

```
$ cat deploy.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-ssd-byok
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: alicloud-disk-ssd-byok
  resources:
    requests:
      storage: 20Gi
---
kind: Pod
apiVersion: v1
metadata:
  name: disk-pod-ssd-byok
spec:
  containers:
```

```
  - name: disk-pod-byok
    image: nginx
    volumeMounts:
      - name: disk-pvc-byok
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: disk-pvc-byok
      persistentVolumeClaim:
        claimName: disk-ssd-byok
```

2. 执行如下命令，创建应用。

```
kubectl apply -f deploy.yaml
```

3. 执行如下命令，查看PV。

```
$ kubectl get pv
NAME                      CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                     STORAGECLASS              REASON
AGE
d-j6c5jezzajt9ri47lvjs   20Gi       RWO            Delete
Bound    default/disk-ssd-byok   alicloud-disk-ssd-byok
8s
```

4. 登录ECS管理控制台，单击左侧导航栏的存储与快照 > 云盘，在磁盘列表中， 查看磁盘ID为 `d-j6c5jezzajt9ri47lvjs`的云盘是否已加密。

更多关于云盘加密以及BYOK的介绍请参见#unique_215。

11 存储管理-CSI

11.1 概述

容器服务持在Kubernetes集群通过Pod自动绑定阿里云云盘、NAS、 OSS 存储服务。

目前CSI驱动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	云盘静态存储卷支持使用CSI驱动进行挂载。 通过PV/PVC方式	支持
阿里云 NAS	NAS 静态存储卷支持使用CSI驱动进行挂载 通过PV/PVC方式	不支持
阿里云 OSS	OSS 静态存储卷支持使用CSI驱动进行挂载 通过PV/PVC方式	不支持

版本支持

- 当Kubernetes版本为1.14及以上版本时，支持选择使用Flexvolume/CSI插件类型。
- 新创建集群选择为CSI驱动时，kubelet参数`enable-controller-attach-detach`配置为true。

CSI驱动支持的Region同Kubernetes集群1.14所开放的Region。

11.2 安装插件

通过下面的 yaml 配置部署阿里云 Kubernetes 存储插件。

> 说明:
> CSI 插件需要您的集群版本大于1.14，且初始化集群的时候选择使用CSI插件。或者自行配置节点kubelet参数，确保`enable-controller-attach-detach`为true。

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 命名空间部署 CSI。

验证安装完成

在 Master 节点上：

- 执行如下命令，输出若干（节点个数）Running 状态的 Pod 列表。

```
kubectl get pod -n kube-system | grep csi-plugin
```

- 执行如下命令，输出一个 Running 状态的 Pod 列表。

```
kubectl get pod -n kube-system | csi-provisioner
```

安装 CSI-Plugin

CSI-Plugin 在Kubernetes集群中已经默认安装。

如果您的集群中没有安装该插件，可以通过以下yaml文件手动安装。

```
apiVersion: storage.k8s.io/v1beta1
kind: CSIDriver
metadata:
  name: diskplugin.csi.alibabacloud.com
spec:
  attachRequired: false
---
apiVersion: storage.k8s.io/v1beta1
kind: CSIDriver
metadata:
  name: nasplugin.csi.alibabacloud.com
spec:
  attachRequired: false
---
apiVersion: storage.k8s.io/v1beta1
kind: CSIDriver
metadata:
  name: ossplugin.csi.alibabacloud.com
spec:
  attachRequired: false
---
kind: DaemonSet
apiVersion: apps/v1beta2
metadata:
  name: csi-plugin
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: csi-plugin
  template:
    metadata:
      labels:
```

```
        app: csi-plugin
    spec:
      tolerations:
        - operator: "Exists"
      priorityClassName: system-node-critical
      serviceAccount: admin
      hostNetwork: true
      hostPID: true
      containers:
        - name: disk-driver-registrar
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver
-registrar:v1.1.0
          imagePullPolicy: Always
          args:
            - "--v=5"
            - "--csi-address=/var/lib/kubelet/plugins/diskplugin.csi.
alibabacloud.com/csi.sock"
            - "--kubelet-registration-path=/var/lib/kubelet/plugins/
diskplugin.csi.alibabacloud.com/csi.sock"
          volumeMounts:
            - name: kubelet-dir
              mountPath: /var/lib/kubelet/
            - name: registration-dir
              mountPath: /registration

        - name: csi-diskplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.
14.5.47-5577bcb-aliyun
          imagePullPolicy: "Always"
          args:
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--v=5"
            - "--driver=diskplugin.csi.alibabacloud.com"
          env:
            - name: CSI_ENDPOINT
              value: unix://var/lib/kubelet/plugins/diskplugin.csi.
alibabacloud.com/csi.sock
            - name: ACCESS_KEY_ID
              value: ""
            - name: ACCESS_KEY_SECRET
              value: ""
            - name: MAX_VOLUMES_PERNODE
              value: "15"
            - name: DISK_TAGED_BY_PLUGIN
              value: "true"
          volumeMounts:
            - name: kubelet-dir
              mountPath: /var/lib/kubelet
              mountPropagation: "Bidirectional"
            - name: container-dir
              mountPath: /var/lib/container
              mountPropagation: "Bidirectional"
            - mountPath: /dev
              mountPropagation: "HostToContainer"
              name: host-dev
            - mountPath: /var/log/
              name: host-log
            - name: etc
              mountPath: /host/etc
```

```
        - name: nas-driver-registrar
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver
-registrar:v1.1.0
          imagePullPolicy: Always
          args:
          - "--v=5"
          - "--csi-address=/var/lib/kubelet/plugins/nasplugin.csi.
alibabacloud.com/csi.sock"
          - "--kubelet-registration-path=/var/lib/kubelet/plugins/
nasplugin.csi.alibabacloud.com/csi.sock"
          volumeMounts:
          - name: kubelet-dir
            mountPath: /var/lib/kubelet
          - name: registration-dir
            mountPath: /registration

        - name: csi-nasplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.
14.5.47-5577bcb-aliyun
          imagePullPolicy: "Always"
          args:
          - "--endpoint=$(CSI_ENDPOINT)"
          - "--v=5"
          - "--driver=nasplugin.csi.alibabacloud.com"
          env:
          - name: CSI_ENDPOINT
            value: unix://var/lib/kubelet/plugins/nasplugin.csi.
alibabacloud.com/csi.sock
          volumeMounts:
          - name: kubelet-dir
            mountPath: /var/lib/kubelet/
            mountPropagation: "Bidirectional"
          - mountPath: /var/log/
            name: host-log

        - name: oss-driver-registrar
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver
-registrar:v1.1.0
          imagePullPolicy: Always
          args:
          - "--v=5"
          - "--csi-address=/var/lib/kubelet/plugins/ossplugin.csi.
alibabacloud.com/csi.sock"
          - "--kubelet-registration-path=/var/lib/kubelet/plugins/
ossplugin.csi.alibabacloud.com/csi.sock"
          volumeMounts:
          - name: kubelet-dir
            mountPath: /var/lib/kubelet/
          - name: registration-dir
            mountPath: /registration

        - name: csi-ossplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
```

```
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.
14.5.47-5577bcb-aliyun
          imagePullPolicy: "Always"
          args:
          - "--endpoint=$(CSI_ENDPOINT)"
          - "--v=5"
          - "--driver=ossplugin.csi.alibabacloud.com"
          env:
          - name: CSI_ENDPOINT
            value: unix://var/lib/kubelet/plugins/ossplugin.csi.
alibabacloud.com/csi.sock
          volumeMounts:
          - name: kubelet-dir
            mountPath: /var/lib/kubelet/
            mountPropagation: "Bidirectional"
          - name: etc
            mountPath: /host/etc
          - mountPath: /var/log/
            name: host-log
          - mountPath: /host/usr/
            name: ossconnectordir
      volumes:
        - name: registration-dir
          hostPath:
            path: /var/lib/kubelet/plugins_registry
            type: DirectoryOrCreate
        - name: container-dir
          hostPath:
            path: /var/lib/container
            type: DirectoryOrCreate
        - name: kubelet-dir
          hostPath:
            path: /var/lib/kubelet
            type: Directory
        - name: host-dev
          hostPath:
            path: /dev
        - name: host-log
          hostPath:
            path: /var/log/
        - name: etc
          hostPath:
            path: /etc
        - name: ossconnectordir
          hostPath:
            path: /usr/
  updateStrategy:
    type: RollingUpdate
```

部署 CSI-Provisioner

CSI-Provisioner 在Kubernetes集群中已经默认安装。

如果您的集群中没有安装该插件，可以通过以下yaml文件手动安装。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: alicloud-disk-available
provisioner: diskplugin.csi.alibabacloud.com
parameters:
    type: available
```

```
reclaimPolicy: Delete
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: alicloud-disk-essd
provisioner: diskplugin.csi.alibabacloud.com
parameters:
    type: cloud_essd
reclaimPolicy: Delete
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: alicloud-disk-ssd
provisioner: diskplugin.csi.alibabacloud.com
parameters:
    type: cloud_ssd
reclaimPolicy: Delete
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: alicloud-disk-efficiency
provisioner: diskplugin.csi.alibabacloud.com
parameters:
    type: cloud_efficiency
reclaimPolicy: Delete
---
kind: Service
apiVersion: v1
metadata:
  name: csi-provisioner
  namespace: kube-system
  labels:
    app: csi-provisioner
spec:
  selector:
    app: csi-provisioner
  ports:
    - name: dummy
      port: 12345

---
kind: StatefulSet
apiVersion: apps/v1beta1
metadata:
  name: csi-provisioner
  namespace: kube-system
spec:
  serviceName: "csi-provisioner"
  replicas: 1
  template:
    metadata:
      labels:
        app: csi-provisioner
    spec:
      tolerations:
      - operator: "Exists"
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
```

```
              matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
      priorityClassName: system-node-critical
      serviceAccount: admin
      hostNetwork: true
      containers:
        - name: csi-external-provisioner
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-provisioner
:v1.2.0
          args:
            - "--provisioner=diskplugin.csi.alibabacloud.com"
            - "--csi-address=$(ADDRESS)"
            - "--feature-gates=Topology=True"
            - "--volume-name-prefix=disk"
            - "--v=5"
          env:
            - name: ADDRESS
              value: /socketDir/csi.sock
          imagePullPolicy: "Always"
          volumeMounts:
            - name: disk-provisioner-dir
              mountPath: /socketDir
        - name: csi-diskprovisioner
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.
14.5.47-5577bcb-aliyun
          imagePullPolicy: "Always"
          args:
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--v=5"
            - "--driver=diskplugin.csi.alibabacloud.com"
          env:
            - name: CSI_ENDPOINT
              value: unix://socketDir/csi.sock
            - name: ACCESS_KEY_ID
              value: ""
            - name: ACCESS_KEY_SECRET
              value: ""
            - name: MAX_VOLUMES_PERNODE
              value: "15"
          volumeMounts:
            - mountPath: /var/log/
              name: host-log
            - mountPath: /socketDir/
              name: disk-provisioner-dir
            - name: etc
              mountPath: /host/etc
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
            requests:
              cpu: 100m
              memory: 100Mi
          livenessProbe:
            exec:
              command:
              - sh
              - -c
```

```
              - ps -ef | grep plugin.csi.alibabacloud.com | grep
diskplugin.csi.alibabacloud.com | grep -v grep
            failureThreshold: 8
            initialDelaySeconds: 15
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 15
      volumes:
        - name: disk-provisioner-dir
          emptyDir: {}
        - name: host-log
          hostPath:
            path: /var/log/
        - name: etc
          hostPath:
            path: /etc
  updateStrategy:
    type: RollingUpdate
```

11.3 云盘存储卷

11.3.1 云盘存储卷使用说明

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CSI插件支持通过PV/PVC方式挂载云盘：

- 静态存储卷

  您可以通过 PV/PVC 使用云盘静态存储卷。

- 动态存储卷

使用说明

- 云盘为非共享存储，一个云盘（PVC/PV）只能同时被一个 pod 挂载。
- 推荐使用有状态应用（StatefulSet）挂载使用云盘。无状态应用（Deployment）挂载云盘时Replica需要为1，不能保证挂载、卸载的优先顺序。
- 使用云盘存储卷前需要先申请一个云盘，并获得磁盘 ID。#unique_187。

  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
  - ESSD云盘：最小20Gi
- 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。

- 待挂载的云盘类型必须是按量付费，包年包月的云盘无法被挂载。当您把集群中的ECS实例从按量付费转换成包年包月时，不可以把云盘一起变成包年包月，否则云盘将不能被挂载使用。

11.3.2 静态云盘卷

本文主要为您介绍如何在CSI插件中挂载静态云盘卷。

前提条件

- 您已经创建好一个Kubernetes集群，并且在该集群中部署CSI插件。
- 您已经创建好一个按量付费的云盘。请参见#unique_222。

创建静态PV和PVC

通过云盘控制台创建云盘后，记录其DiskId为`d-wz92s6d95go6ki9xge6b`。

目前控制台只支持通过YAML模板创建CSI PV对象。

1. #unique_33。
2. 通过下面模板创建静态卷PV和PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 25Gi
  selector:
    matchLabels:
      alicloud-pvname: static-disk-pv
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: csi-pv
  labels:
    alicloud-pvname: static-disk-pv
spec:
  capacity:
    storage: 25Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: diskplugin.csi.alibabacloud.com
    volumeHandle: d-wz92s6d95go6ki9xge6b
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: topology.diskplugin.csi.alibabacloud.com/zone
          operator: In
          values:
```

```
          - cn-shenzhen-a
```

说明:

- **driver：定义驱动类型。**

  取值为`diskplugin.csi.alibabacloud.com`，表示使用阿里云云盘CSI插件。

- **volumeHandle：定义云盘ID。**
- **nodeAffinity：定义PV/PVC所属的Zone信息。**

  通过定义该参数，可以将PV/PVC所在的Pod调度到对应的Zone上。

创建应用

1. **创建并拷贝以下内容到*nginx-disk-dept.yaml*文件中。**

```
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"
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: pvc-disk
          mountPath: /data
      volumes:
        - name: pvc-disk
          persistentVolumeClaim:
```

```
              claimName: csi-pvc
```

2. 执行如下命令，创建一个应用。

```
kubectl apply -f nginx-disk-dept.yaml
```

11.3.3 动态云盘卷

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

1. 创建并复制以下内容到*storage-class-csi.yaml*中。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: alicloud-disk-ssd-b
provisioner: diskplugin.csi.alibabacloud.com
parameters:
  type: cloud_ssd
  regionid: cn-shenzhen
  zoneId: cn-shenzhen-b
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

2. 执行如下命令，创建StorageClass。

```
kubectl apply -f storage-class-csi.yaml
```

说明:

参数说明:

- `provisioner`：配置为 diskplugin.csi.alibabacloud.com，标识StorageClass使用阿里云云盘 provisioner 插件创建。
- `type`：标识云盘类型，支持 cloud、cloud_efficiency、cloud_ssd、cloud_essd、available 四种类型；其中 available 会对高效、ESSD、SSD依次尝试创建，直到创建成功。
- `regionid`：可选参数。期望创建云盘的区域。
- `reclaimPolicy`：云盘的回收策略，默认为Delete，支持Retain。如果数据安全性要求高，推荐使用Retain方式以免误删。
- zoneid：可选参数。期望创建云盘的可用区。如果是多可用区的情况，使用topology方式创建PV。
- `encrypted`：可选参数。创建的云盘是否加密，默认情况是false，创建的云盘不加密。

创建PVC/PV

下面为创建动态卷PV的示例YAML模板：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: disk-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 25Gi
  storageClassName: alicloud-disk-ssd
```

说明：

`storage`：定义申请云盘大小，最小为20Gi。

提供默认StorageClass：

在多可用区的集群中，需要您手动创建上述 StorageClass，这样可以更准确的定义所需要云盘的可用区信息。

集群默认提供了下面几种 StorageClass，可以在单可用区类型的集群中使用。

- alicloud-disk-efficiency：高效云盘。
- alicloud-disk-ssd：SSD云盘。
- alicloud-disk-essd：ESSD云盘。
- alicloud-disk-available：提供高可用选项，优先创建高效云盘。

  如果相应可用区的高效云盘资源售尽，则优先创建SSD盘；如果SSD也售尽，则创建ESSD盘。

创建应用

```
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"
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: pvc-disk
          mountPath: /data
      volumes:
        - name: pvc-disk
          persistentVolumeClaim:
            claimName: disk-pvc
```

11.4 NAS存储卷

11.4.1 NAS卷使用说明

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 支持两种 CSI 挂载方式：

- 静态存储卷挂载
- 动态存储卷挂载

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挂载点。创建的 NAS 文件系统挂载点需要和您的集群位于同一 VPC。登录文件存储管理控制台，可以创建一个 NAS 文件系统和挂载点。

例如，您的挂载点为`055f84ad83-ixxxx.cn-hangzhou.nas.aliyuncs.com`格式。

注意事项

- NAS 为共享存储，可以同时为多个 Pod 提供共享存储服务，即一个PVC可以同时被多个POD使用。
- 在没有卸载NAS文件系统前，务必不要删除NAS挂载点，否则会造成操作系统Hang。
- NAS挂载点创建后，等待一定时间，待挂载点Ready后方可使用。
- 数据卷挂载协议推荐使用NAS v3。
- 使用NAS数据卷前，推荐升级CSI到最新版本。

11.4.2 静态NAS卷

您可以通过 PV/PVC 方式使用阿里云 NAS 数据卷。

前提条件

- 您已经创建一个静态NAS卷。请参见#unique_227。
- 您已经创建NAS挂载点。请参见#unique_228。

  NAS挂载点需要和集群节点在同一个vpc内。

创建静态PV/PVC

以下为创建静态卷PV/PVC的yaml示例文件：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nas
  labels:
    alicloud-pvname: pv-nas
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: nasplugin.csi.alibabacloud.com
    volumeHandle: pv-nas
    volumeAttributes:
      server: "2564f49129-ysu87.cn-shenzhen.nas.aliyuncs.com"
      path: "/csi"
      vers: "3"
      options: "noresvport,nolock"
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-nas
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-nas
```

说明：

- `driver`：驱动类型。本例中取值为nasplugin.csi.alibabacloud.com，表示使用阿里云NAS CSI插件。
- `volumeHandle`：配置PV的名称。
- `host`：NAS挂载点。

- path：挂载子目录。
- vers：NAS目标版本。
- mode：挂载目录权限。

创建应用

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-nas
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        volumeMounts:
          - name: nas-pvc
            mountPath: "/data"
      volumes:
        - name: nas-pvc
          persistentVolumeClaim:
            claimName: nas-pvc
```

11.5 OSS存储卷

11.5.1 OSS卷使用说明

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。

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，挂载节点和bucket相同region时，可使用内网地址。

- 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 ***`。

11.5.2 静态OSS卷

本文为您介绍如何使用OSS静态存储卷。

前提条件

您已经创建静态OSS存储卷，请参见#unique_232。

> 说明:
> 相同region的OSS Bucket 可以使用内网地址进行挂载。

创建Secret

以下为通过Secret来配置用户的AccessKey信息的Yaml示例文件：

```
apiVersion: v1
kind: Secret
metadata:
  name: oss-secret
  namespace: default
stringData:
  akId: ***
  akSecret: ***
```

> 说明:
> 创建Secret选择的namespace需要和应用所在namespaces一致。
>
> `akId`、`akSecret`需要替换成用户的AccessKeyID和AccessKeySecret。

创建静态卷PV/PVC

以下为创建静态卷PV/PVC的yaml示例文件：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: oss-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: oss-csi-pv
spec:
```

```
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: ossplugin.csi.alibabacloud.com
    volumeHandle: data-id
    nodePublishSecretRef:
      name: oss-secret
      namespace: default
    volumeAttributes:
      bucket: "oss"
      url: "oss-cn-hangzhou.aliyuncs.com"
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

- `nodePublishSecretRef`：定义挂载PV时通过Secret对象来获取AccessKey信息。
- `volumeHandle`：配置PV的名称。
- `bucket`：需要挂载的OSS Bucket。
- `url`：挂载 OSS 的接入域名，挂载节点和Bucket相同region时，可使用内网地址。
- `otherOpts`：挂载 OSS 时支持输入定制化参数，格式为：`-o *** -o ***`。

配置PV/PVC的AccessKey

创建静态卷PV/PVC是通过Secret为plugin提供AccessKey信息，您也可以直接在PV中配置AccessKey信息。示例Yaml文件如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: oss-csi-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: ossplugin.csi.alibabacloud.com
    volumeHandle: data-id
    volumeAttributes:
      bucket: "oss"
      url: "oss-cn-hangzhou.aliyuncs.com"
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"
      akId: "***"
      akSecret: "***"
```

创建应用

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
```

```
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        volumeMounts:
          - name: oss-pvc
            mountPath: "/data"
      volumes:
        - name: oss-pvc
          persistentVolumeClaim:
            claimName: oss-pvc
```

12 日志管理

12.1 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#unique_30，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，#unique_235，您可以方便地搭建自己的应用日志集群。

12.2 查看集群日志

背景信息

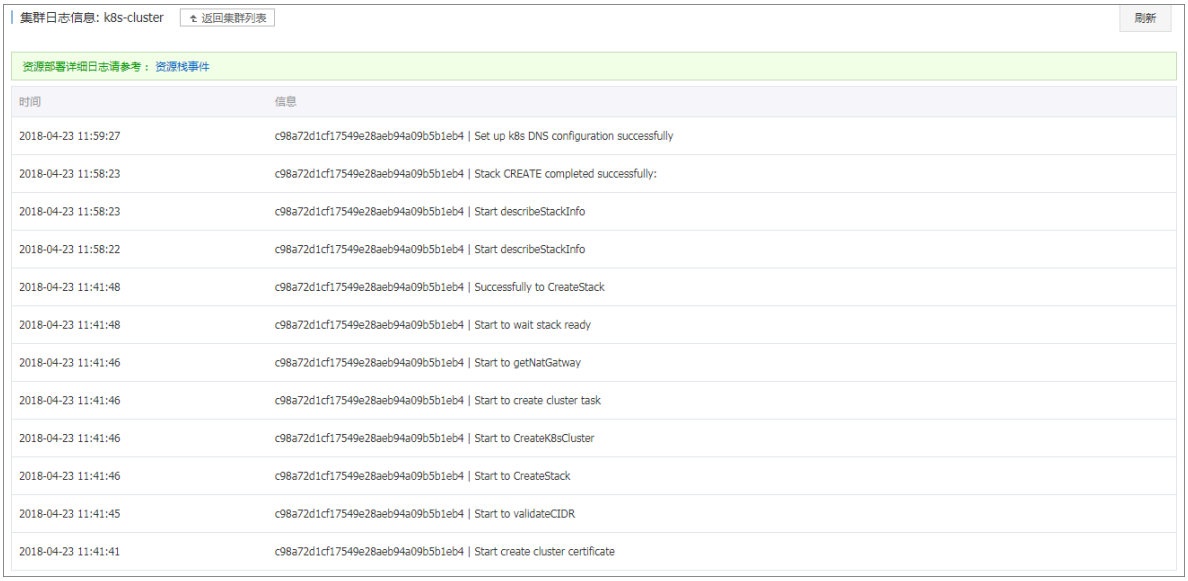
您可以通过容器服务的简单日志服务查看集群的操作日志。

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。



12.3 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，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。

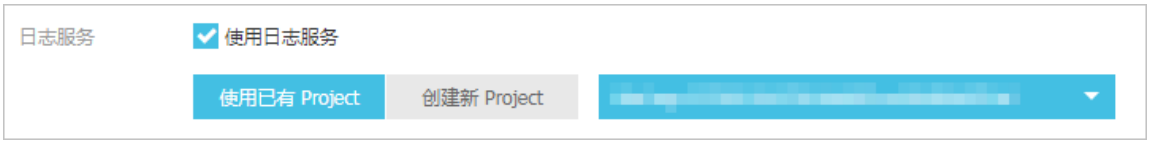
新建 Kubernetes 集群

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#unique_237进行配置。
4. 拖动到页面底部，勾选日志服务配置项，表示在新建的 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#unique_238）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. 通过cloudshell 连接 Kubernetes 集群。

   详细内容，可参考#unique_239 。

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. 通过cloudshell 连接 Kubernetes 集群。

   详细内容，可参考#unique_239 。

2. 在cloudshell中获取您的主账号aliuid，命令为 `echo $ALIBABA_CLOUD_ACCOUNT_ID`。

3. 替换`${your_k8s_cluster_id}`、`{your_ali_uid}`、`{your_k8s_cluster_region_id}`参数后执行以下安装命令。

```
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：您在步骤2中获取的aliuid。
- 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:

通过cloudshell 连接 Kubernetes 集群。

详细内容可参考#unique_239 。

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升级

若您不是使用helm的方式安装的日志组件，可通过修改DaemonSet模板的方式进行升级。若您的镜像账号为acs，请参考镜像仓库中最新的tag，将镜像中的tag更新至最新版本；若您的镜像账号为log-service，请参考镜像仓库中最新的tag，将镜像中的tag更新至最新版本。

说明：

- 用户升级tag后，如果Logtail没有滚动更新，您需要手动删除Logtail的Pod，触发强制更新。
- 查看Logtail是否运行在所有节点（包括master节点）上。若没有，请为Logtail设置tolerations。

```
tolerations:
- operator: "Exists"
```

更多最新配置方式，请参考最新helm包中的相关配置。

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 c12ba2028cxxxxxxxxxx6939f0b，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 c12ba2028cxxxxxxxxxx6939f0b --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 > 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#unique_97。
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#unique_97。

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 > 无状态，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。

环境变量高级配置

通过容器环境变量配置采集支持多种配置参数，您可根据实际需求设置高级参数来实现日志采集的特殊需求。

字段	说明	示例	注意事项

aliyun_logs_{key}	· 必选项。{key}由[a-z][a-z0-9-]组成，不能包含_。
	· 若不存在aliyun_logs_{key}_logstore，则默认创建并采集到名为{key}的logstore。
	· 当值为stdout表示采集容器的标准输出；其他值为容器内的日志路径。
		· `- name: aliyun_logs_catalina`
		`stdout`
		· `- name: aliyun_logs_access-log`
		`/var/log/nginx/access.log`
			· 默认采集方式为极简模式，如需解析日志内容，建议使用日志服务控制台并参考#unique_241、#unique_242行配置。
			· {key}需保持在K8s集群内唯一。

aliyun_logs_{key}_tags	可选。值为 {tag-key}={tag-value}类型，用于对日志进行标识。	`- name: aliyun_logs_catalina_tags` `app=catalina`	-
aliyun_logs_{key}_project	可选。值为指定的日志服务Project。当不存在该环境变量时为您安装时所选的Project。	`- name: aliyun_logs_catalina_project` `my-k8s-project`	Project需与您的Logtail工作所在Region一致。
aliyun_logs_{key}_logstore	可选。值为指定的日志服务Logstore。当不存在该环境变量时Logstore和{key}一致。	`- name: aliyun_logs_catalina_tags` `my-logstore`	-
aliyun_logs_{key}_shard	可选。值为创建Logstore时的shard数，有效值为1~10。当不存在该环境变量时值为2。	`- name: aliyun_logs_catalina_shard` `4`	-
aliyun_logs_{key}_ttl	可选。值为指定的日志保存时间，有效值为1~3650， · 当取值为3650时，指定日志的	`- name: aliyun_logs_catalina_ttl` `3650`	-

如果您需要将多个应用数据采集到同一Logstore，可以设置 aliyun_logs_{key}_logstore 参数，例如以下配置将2个应用的stdout采集到stdout-logstore中：

应用1设置的环境变量为：

```
######### 配置 环境变量 ###########
    - name: aliyun_logs_app1-stdout
      value: stdout
    - name: aliyun_logs_app1-stdout_logstore
      value: stdout-logstore
```

应用2设置的环境变量为：

```
######### 配置 环境变量 ###########
    - name: aliyun_logs_app2-stdout
      value: stdout
    - name: aliyun_logs_app2-stdout_logstore
      value: stdout-logstore
```

定制需求2：将不同应用数据采集到不同的Project

如果您需要将不同应用的数据采集到多个Project中，您需要进行以下操作：

1. 在每个Project中创建一个机器组，选择自定义标识，标识名为 k8s-group-{cluster-id}，其中{cluster-id}为您的集群ID，机器组名称您可以自定义配置。
2. 每个应用的环境变量中配置project、logstore、machinegroup信息，其中机器组名称为您在步骤1中创建的机器组名，例如：

```
######### 配置 环境变量 ###########
    - name: aliyun_logs_app1-stdout
      value: stdout
    - name: aliyun_logs_app1-stdout_project
      value: app1-project
    - name: aliyun_logs_app1-stdout_logstore
      value: app1-logstore
    - name: aliyun_logs_app1-stdout_machinegroup
      value: app1-machine-group
```

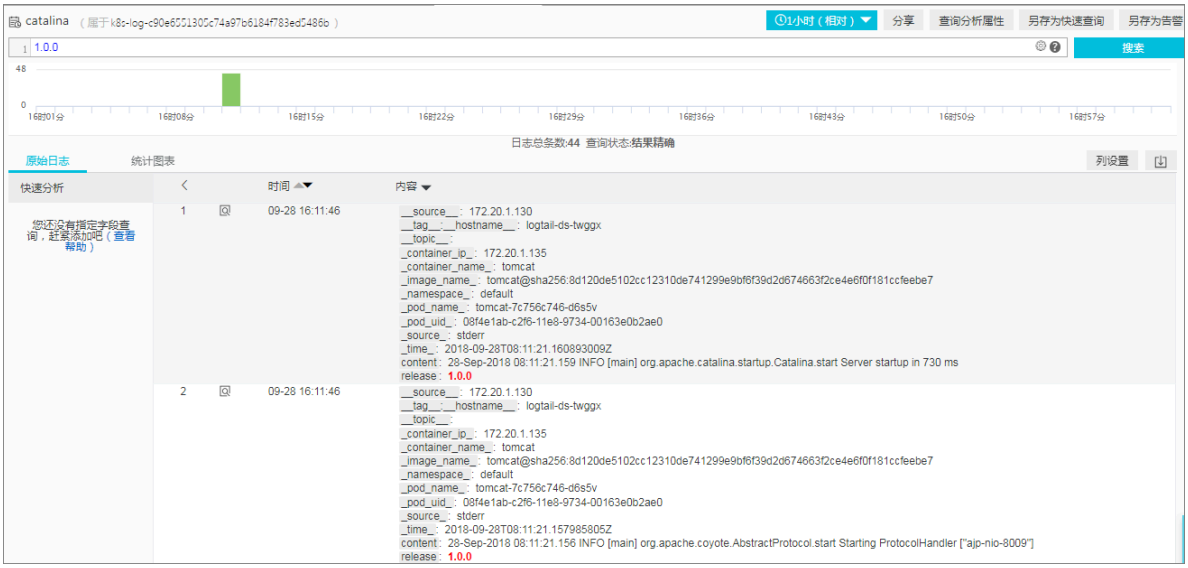
查看日志

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：
   - #unique_244
   - #unique_245
2. 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#unique_155。
3. 对于异常的排查，可以参考#unique_246。

12.4 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"`。

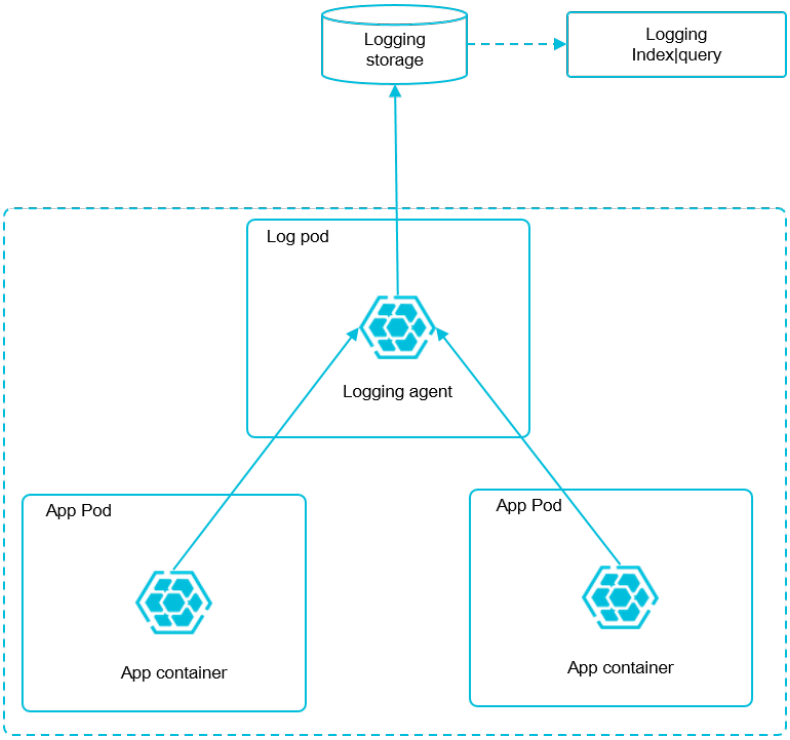
此外，Log-Pilot 还支持多种日志解析格式，通过 `aliyun_logs_$name_format=<format>` 标签就可以告诉 Log-Pilot 在采集日志的时候，同时以什么样的格式来解析日志记录，支持的格式包括：none、json、csv、nginx、apache2 和 regxp。

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。

请务必确保到 ElasticSearch 集群网络可达。

步骤1 部署ElasticSearch集群

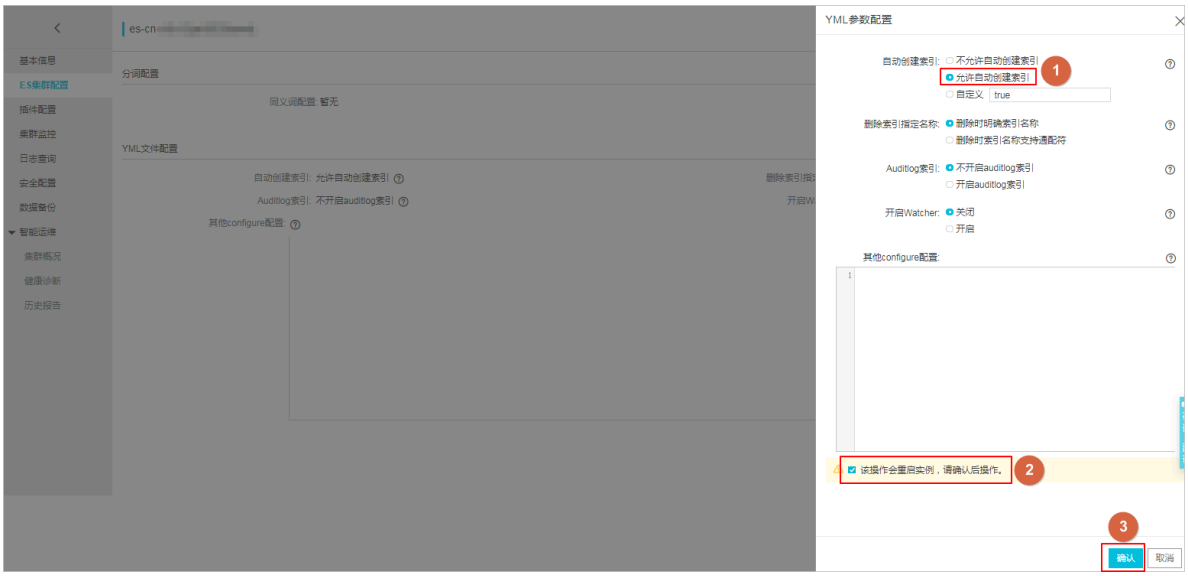
请参考#unique_248/unique_248_Connect_42_section_bpg_tjl_zgb完成ElasticSearch集群的部署，完成后，进行下一步操作。

> **说明:**
> 默认情况下，阿里云ElasticSearch集群（简称ES集群）当新增的文档发现没有索引时，不允许自动创建索引，而Log-Pilot在自动采集容器日志时需要依据日志采集配置来自动创建文档索引，因此我们这里需要开启自动创建索引功能。

您可通过如下操作，开启自动创建索引功能。

1. 单击目标集群右侧的管理，进入基本信息页面。
2. 选择ES集群配置，在YML文件配置区域框右侧单击修改配置，弹出YMAL参数配置页面。

3. 自动创建索引选择允许自动创建索引，勾选该操作会重启实例，请确认后操作。后，单击确认。



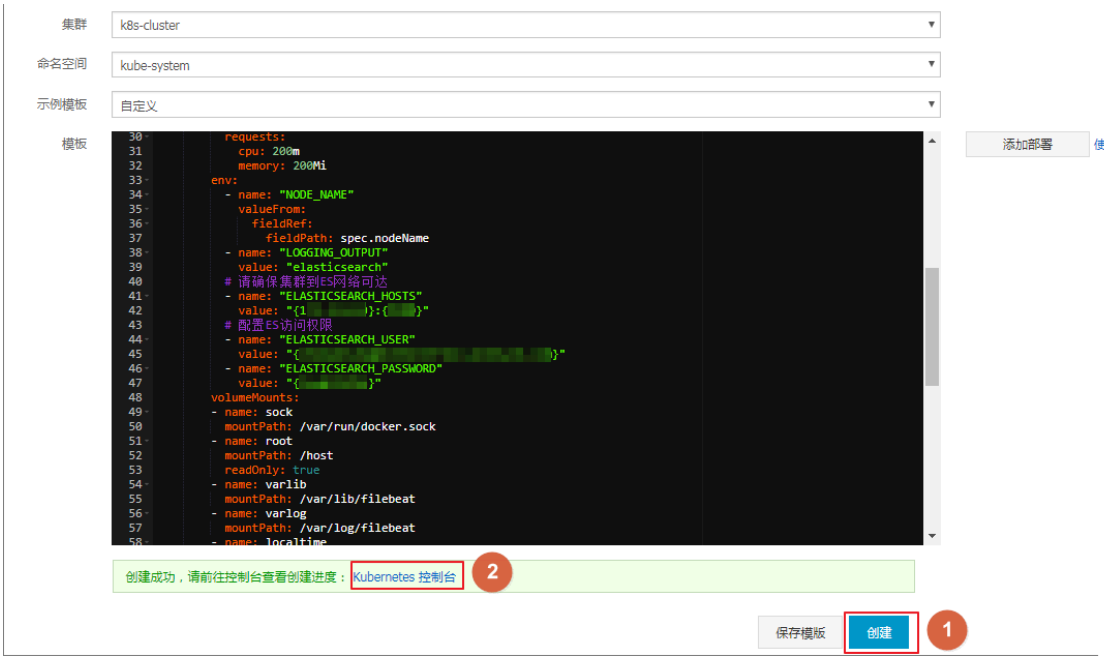
步骤2 部署 log-pilot组件

为降低节点的资源消耗，Log-Pilot 组件采用的是 DaemonSet 方式部署到每个集群的 Node 节点上，部署 Log-Pilot 日志采集工具，操作如下：

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，进入无状态页面。

3. 单击右上角的使用模板创建，进入使用模板创建页面。
4. 选择需要部署 log-pilot 组件的集群，并选择kube-system作为命名空间。

5. 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: extensions/v1beta1
kind: DaemonSet
metadata:
name: log-pilot
labels:
app: log-pilot
# 设置期望部署的namespace
namespace: kube-system
spec:
updateStrategy:
type: RollingUpdate
template:
metadata:
labels:
app: log-pilot
annotations:
scheduler.alpha.kubernetes.io/critical-pod: ''
spec:
# 是否允许部署到Master节点上
tolerations:
- key: node-role.kubernetes.io/master
effect: NoSchedule
containers:
- name: log-pilot
# 版本请参考https://github.com/AliyunContainerService/log-pilot/
releases
image: registry.cn-hangzhou.aliyuncs.com/acs/log-pilot:0.9.6-
filebeat
resources:
limits:
memory: 500Mi
requests:
cpu: 200m
memory: 200Mi
env:
- name: "NODE_NAME"
valueFrom:
fieldRef:
```

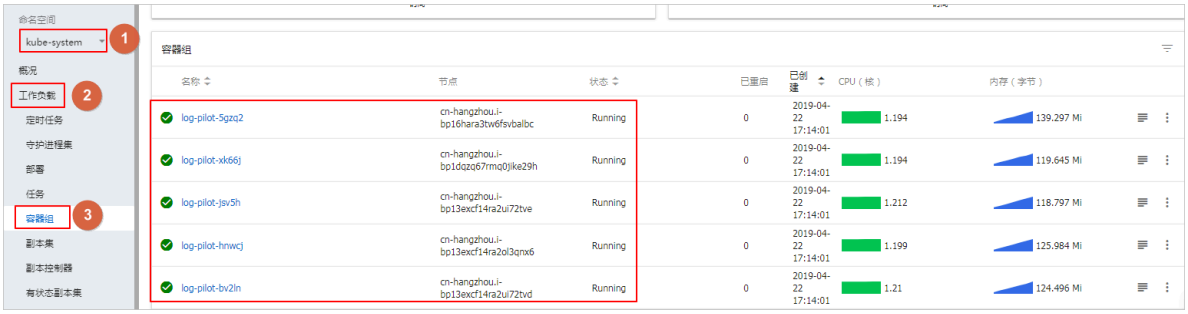
```
fieldPath: spec.nodeName
- name: "LOGGING_OUTPUT"
value: "elasticsearch"
# 请确保集群到ES网络可达
- name: "ELASTICSEARCH_HOSTS"
value: "{es_endpoint}:{es_port}"
# 配置ES访问权限
- name: "ELASTICSEARCH_USER"
value: "{es_username}"
- name: "ELASTICSEARCH_PASSWORD"
value: "{es_password}"
volumeMounts:
- name: sock
mountPath: /var/run/docker.sock
- name: root
mountPath: /host
readOnly: true
- name: varlib
mountPath: /var/lib/filebeat
- name: varlog
mountPath: /var/log/filebeat
- name: localtime
mountPath: /etc/localtime
readOnly: true
livenessProbe:
failureThreshold: 3
exec:
command:
- /pilot/healthz
initialDelaySeconds: 10
periodSeconds: 10
successThreshold: 1
timeoutSeconds: 2
securityContext:
capabilities:
add:
- SYS_ADMIN
terminationGracePeriodSeconds: 30
volumes:
- name: sock
hostPath:
path: /var/run/docker.sock
- name: root
hostPath:
path: /
- name: varlib
hostPath:
path: /var/lib/filebeat
type: DirectoryOrCreate
- name: varlog
hostPath:
path: /var/log/filebeat
type: DirectoryOrCreate
- name: localtime
hostPath:
path: /etc/localtime
```

说明:

参数说明:

- `{es_endpoint}`：ES 集群的访问地址。
  - 如果 Kubernetes 集群和部署的 ES 集群在同一个 VPC 下，则该地址为 ES 集群基本信息中的内网地址。
  - 如果 Kubernetes 集群和部署的 ES 集群在不同一个 VPC 下，则该地址为 ES 集群基本信息中的公网地址。
- `{es_port}`：ES 集群的访问端口，一般是9200。
- `{es_username}`：访问 ES 集群的用户名。默认用户名 elastic。
- `{es_password}`：访问 ES 集群的用户密码。即为创建 ES 集群时，设置的用户密码。

6. 单击Kubernetes 控制台前往控制台查看创建进度。
7. 选择命名空间为kube-system，在负载均衡 > 容器组下，看到如下图所示时，表示运行正常。



说明:

您也可以#unique_33，执行如下命令来查看 Log-Pilot 组件的运行状态。

```
~ kubectl apply -f log-pilot.yml
daemonset.extensions "log-pilot" created
~ kubectl -n kube-system get pod | grep log-pilot
log-pilot-458nj 1/1 Running 0 23s
log-pilot-8ld4n 1/1 Running 0 23s
log-pilot-b4kqv 1/1 Running 0 23s
log-pilot-gd588 1/1 Running 0 23s
log-pilot-k2ttk 1/1 Running 0 23s
```

步骤3 采集日志

这里以部署一个Tomcat为例，通过创建 Deploment 应用配置日志采集（配置方式同样适用 StatefulSet），来测试日志是否能正常采集、索引和显示。

1. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，进入无状态页面。
2. 单击右上角的使用模板创建，进入使用模板创建页面。
3. 选择集群为k8s-cluster（必须和Log-Pilot 组件部署在同一集群下），并选择目标命名空间。

4. 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。

   编排模板如下：

```
apiVersion: v1
kind: Pod
metadata:
name: tomcat
spec:
containers:
- name: tomcat
image: "tomcat:7.0"
env:
# 1、stdout为约定关键字，表示采集标准输出日志
# 2、配置标准输出日志采集到ES的catalina索引下
- name: aliyun_logs_catalina
value: "stdout"
# 1、配置采集容器内文件日志，支持通配符
# 2、配置该日志采集到ES的access索引下
- name: aliyun_logs_access
value: "/usr/local/tomcat/logs/catalina.*.log"
# 容器内文件日志路径需要配置emptyDir
volumeMounts:
- name: tomcat-log
mountPath: /usr/local/tomcat/logs
volumes:
- name: tomcat-log
emptyDir: {}
```

说明：

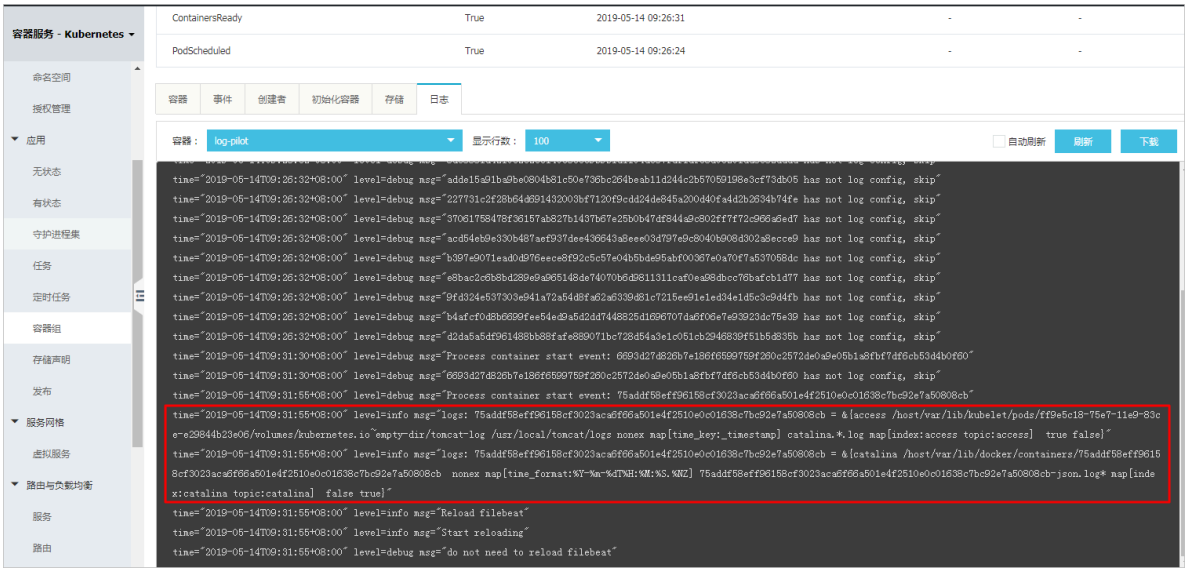
在上面的编排中，通过在 Pod 中定义环境变量的方式，动态地生成日志采集配置文件，环境变量的具体说明如下：

- `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 。

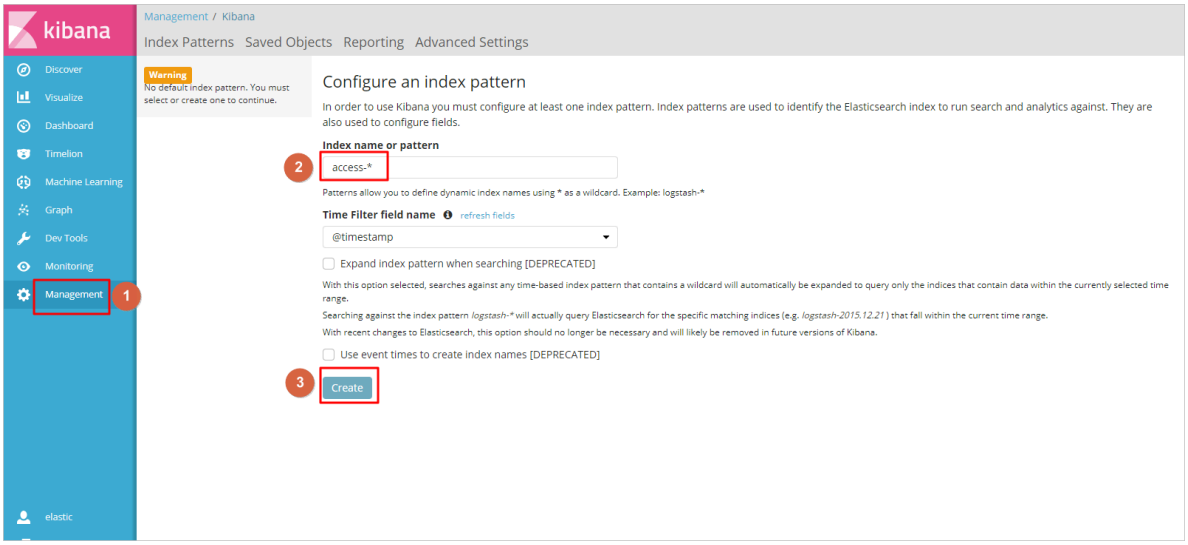
5. 选择应用 > 容器组，选择部署 Tomcat 的集群cluster-k8s和命名空间default，在目标 Tomcat 右侧选择更多 > 日志。

在容器组 - tomcat页面的日志页签下，看到如下内容时，表示 Tomcat 部署成功。



6. 在阿里云 ES 控制台，单击Kibana控制台，登录 Kibana。

7. 单击Management，在Configure an index pattern区域输入`access-*`，单击Create。

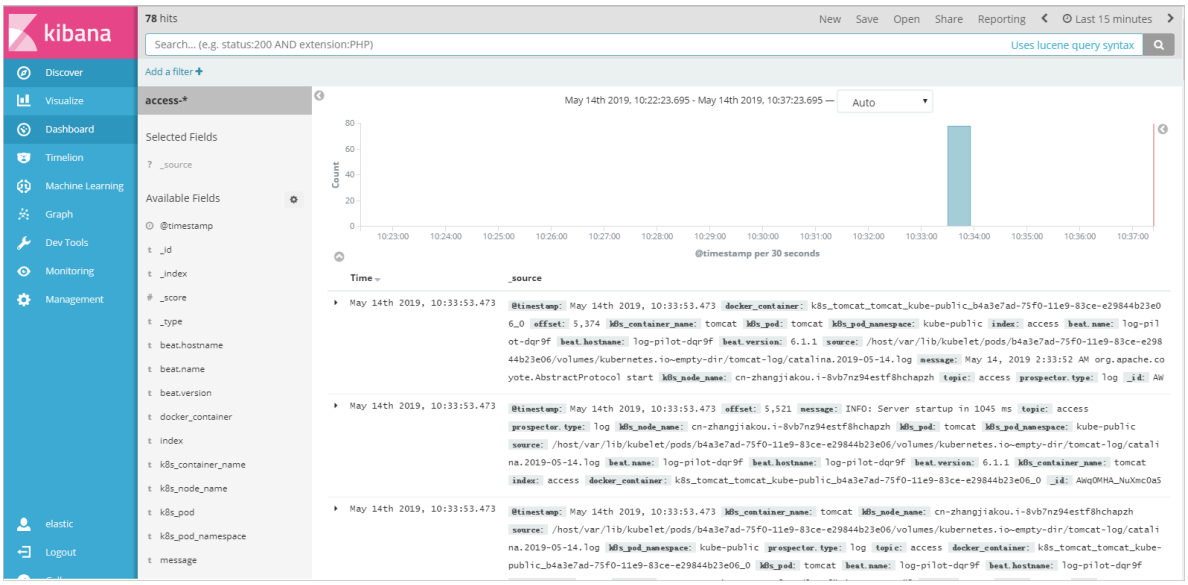


8. （可选）：如果您需要创建多个索引，按照如下步骤操作。

   a. 单击Management，在Kibana区域单击Index Patterns。

   b. 单击Create Index Patterns，在`Index name or pattern`中输入目标索引（索引格式为`XXX-*`），单击Create。

9. 单击Discover，您可以看到 Tomcat 的日志被采集到了指定的 ES 集群中。



> **说明:**
> Log-Pilot 默认采集日志到 ES 集群时会自动创建格式为 index-yyyy.MM.dd 的索引。

我们可以看到只需要通过上面简单的配置就可以很方便地将 Kubernetes 集群中Pod的标准输出日志和容器内文件日志采集到 ES 集群中。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。

12.5 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。

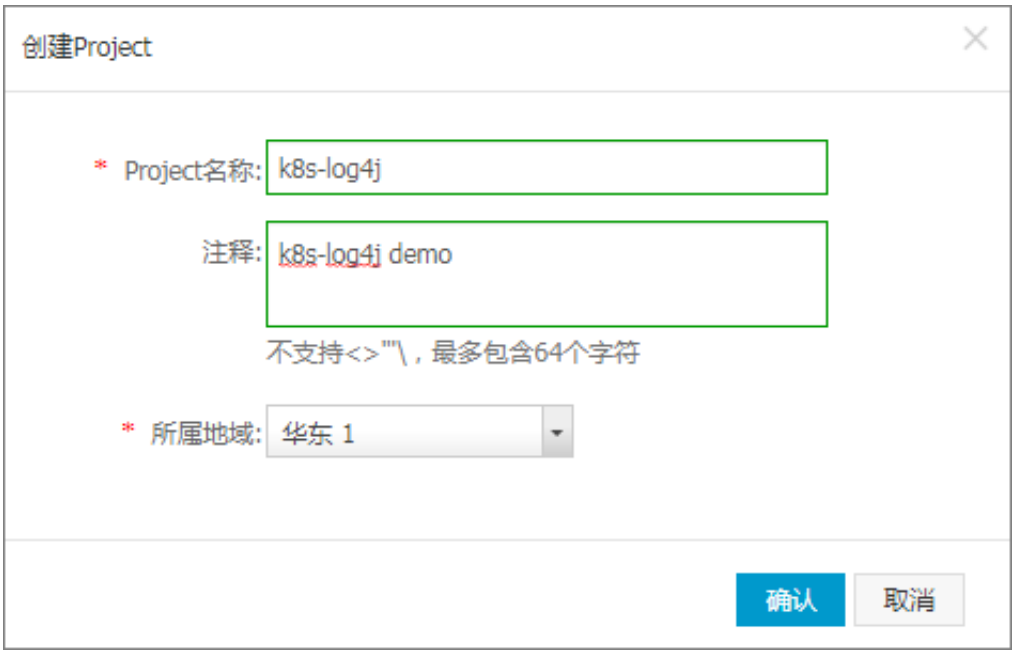
  本示例中，创建的 Kubernetes 集群位于华东 1 地域。
- 启用 AccessKey 或 RAM，确保有足够的访问权限。本例中使用 AccessKey。

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。

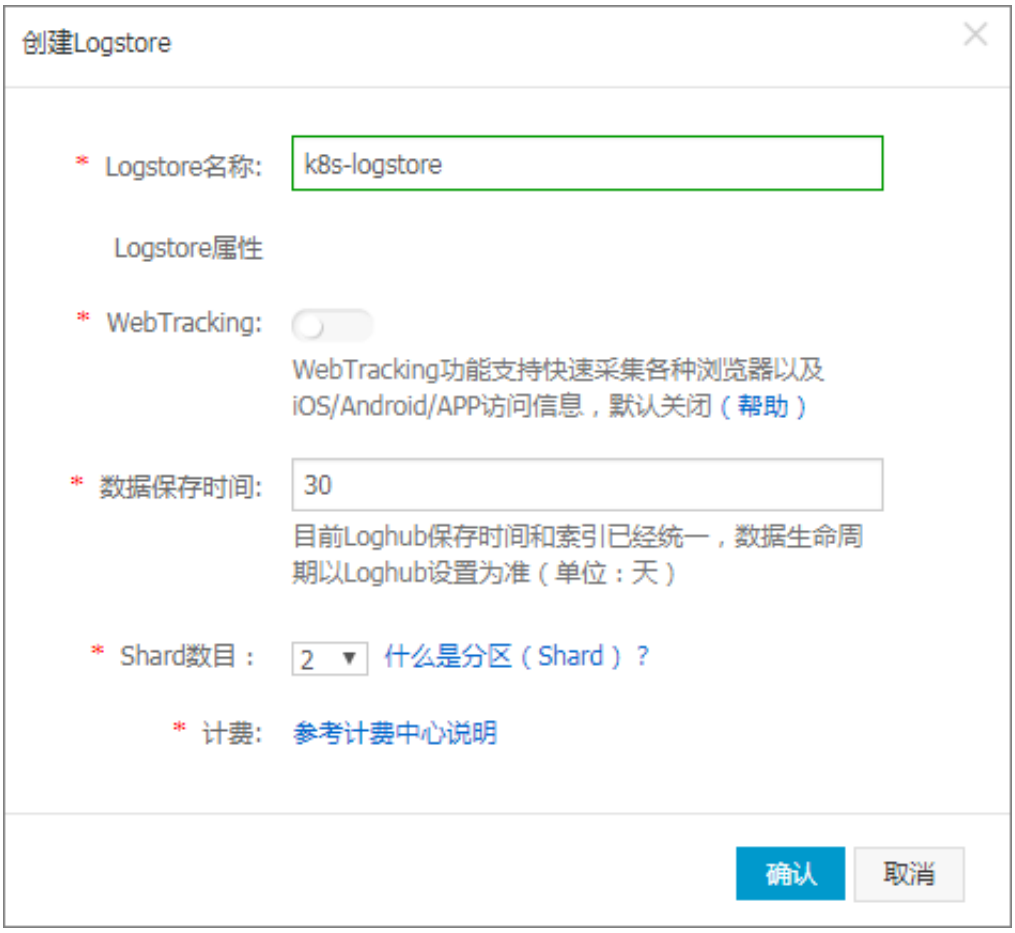


3. 创建完成后，k8s-log4j 出现在 project 列表下，单击该 project 名称，进入 project 详情页面。

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。

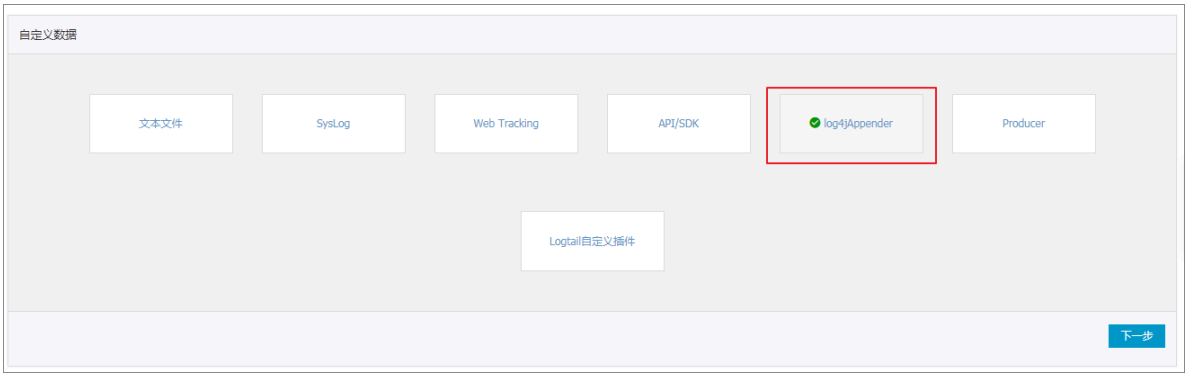
本示例创建名为 k8s-logstore 的日志库。



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#unique_29 或 #unique_37。

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。

13 监控管理

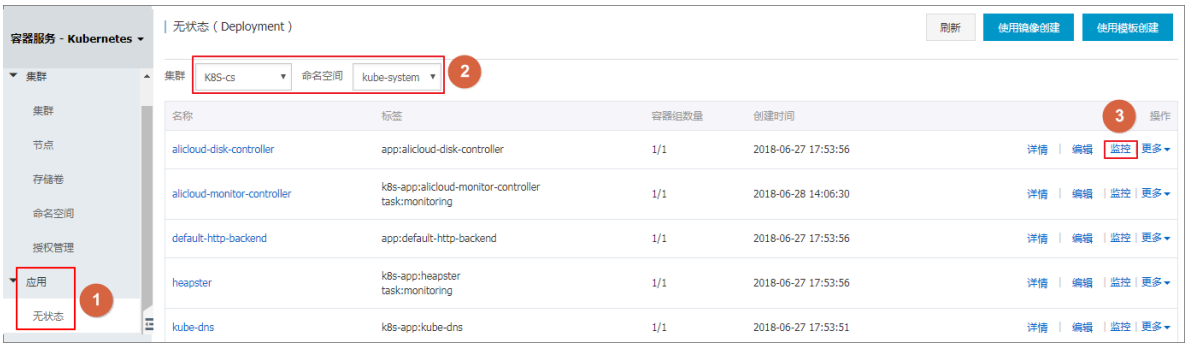
13.1 基础资源监控

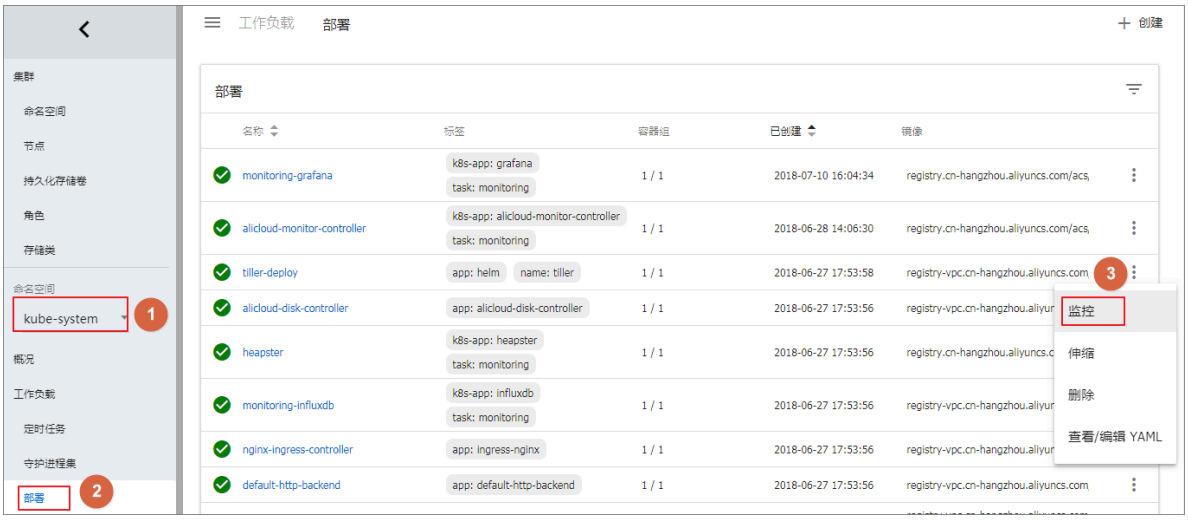
资源监控是Kubernetes中最常见的监控方式，通过资源监控可以快速查看负载的CPU、内存、网络等指标的使用使用率。在阿里云容器服务中，资源监控已经与云监控互通，新建的集群都默认安装与集成了云监控。

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，进入无状态（Deployment）页面。

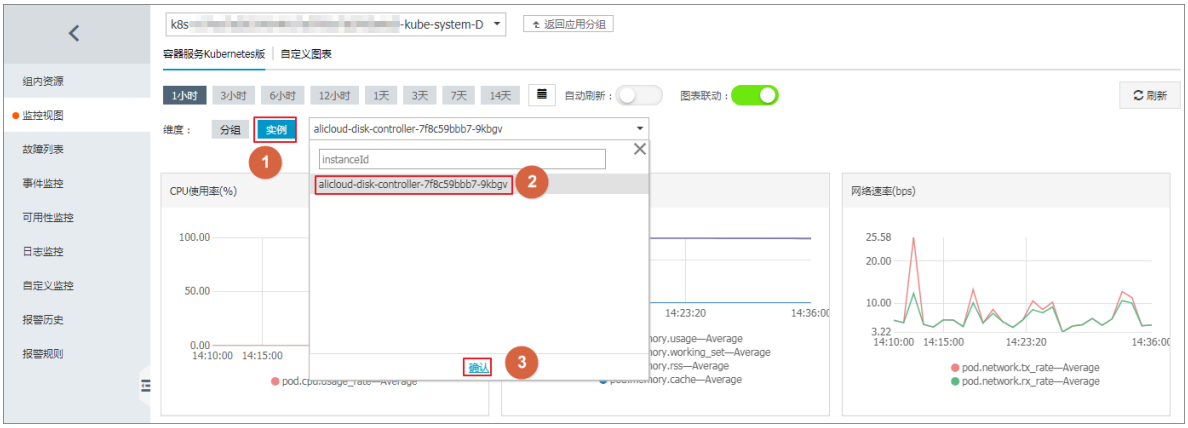
3. 选择所需的deployment，单击右侧的监控，或者在内置的Kubernetes Dashboard的部署页面中单击监控。





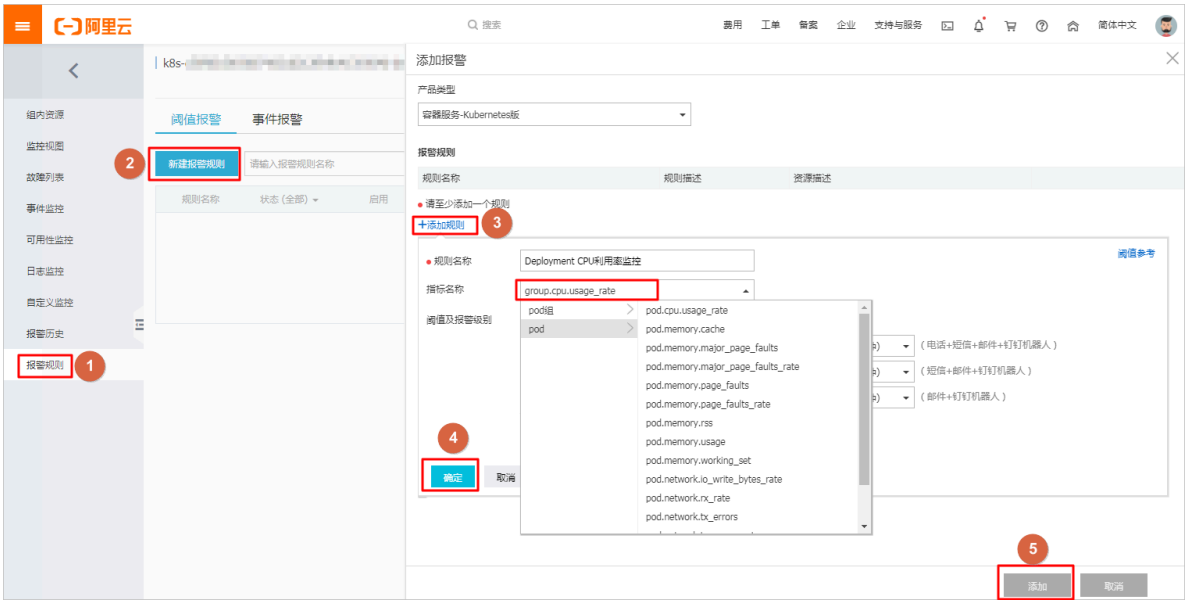
此时会跳转到云监控的相应的监控视图页面。

4. 您可以单击分组或实例，分别通过两个维度进行监控。



5. 如需告警设置，您可以在左侧导航栏单击报警规则并进行设置，详情请参见#unique_251。

分组级别的指标以group开头，实例级别的指标以pod开头。



13.2 应用性能监控

对于部署在容器服务 Kubernetes 版中的 Java/PHP 应用，您可以使用应用实时监控服务 ARMS 对其进行监控，实现自动发现应用拓扑、自动生成 3D 拓扑、自动发现并监控接口、捕获异常事务和慢事务，大幅提升线上问题诊断的效率。

前提条件

- #unique_237
- #unique_253：本文示例中的命名空间名称为 `arms-demo`
- #unique_254

> 说明:
> PHP 应用监控处于公测期，使用不会产生费用。

背景信息

应用实时监控服务 ARMS（Application Real-Time Monitoring Service）是一款阿里云应用性能管理（APM）类监控产品。只要为部署在容器服务 Kubernetes 版中的 Java 应用安装 ARMS 应用监控组件，您无需修改任何代码，就能借助 ARMS 对 Java 应用进行全方位监控，以便您更快速地定位出错接口和慢接口、重现调用参数、检测内存泄漏、发现系统瓶颈，从而大幅提升线上问题诊断问题的效率。ARMS 应用监控的详细信息请参见#unique_256。

安装 ARMS 应用监控组件

首先需要安装 ARMS 应用监控组件 ack-arms-pilot。

1. 登录容器服务 Kubernetes 版控制台。
2. 在左侧导航栏选择市场 > 应用目录，在右侧页面单击 ack-arms-pilot。
3. 在应用目录 - ack-arms-pilot 页面上，在右侧的创建面板中选择前提条件中创建的集群和命名空间，并单击创建。

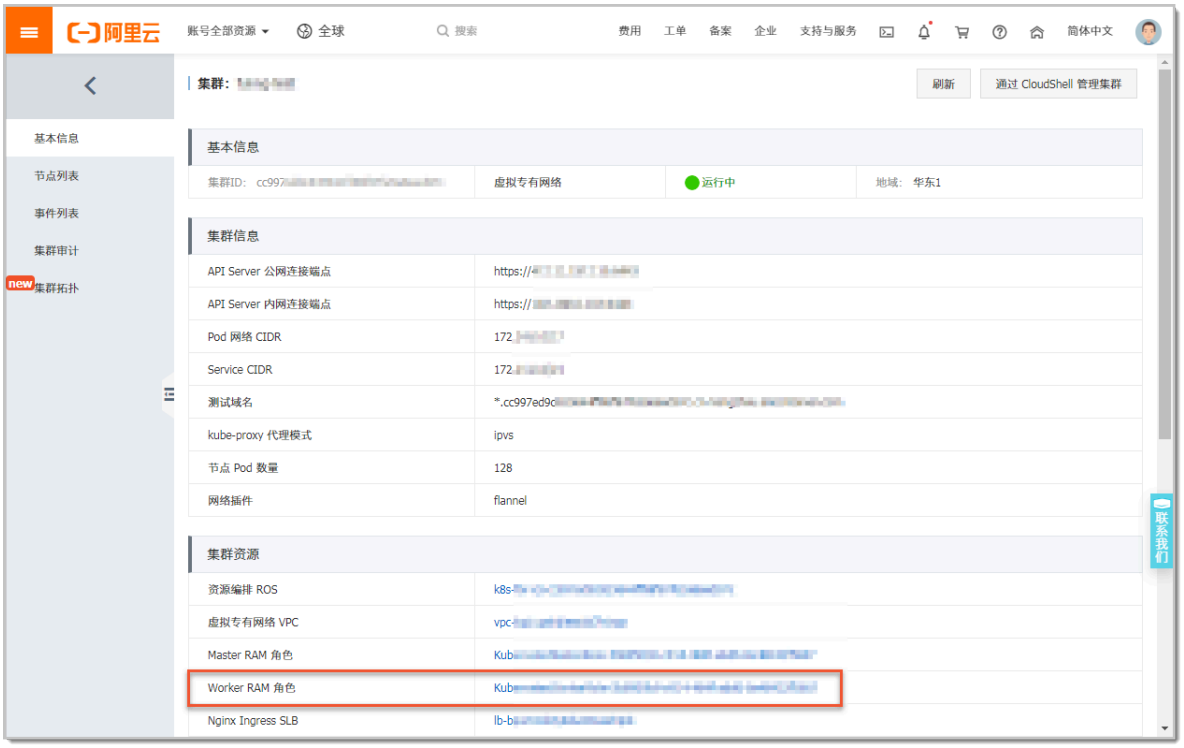
为容器服务 Kubernetes 版授权

接下来要为容器服务 Kubernetes 版授予 ARMS 资源的访问权限。

1. 使用主账号登录容器服务 Kubernetes 版控制台。

2. 在左侧导航栏选择集群 > 集群，在集群列表页面上的目标集群右侧操作列单击管理。

3. 在目标集群的基本信息页面上，单击集群资源区域的 Worker RAM 角色链接。

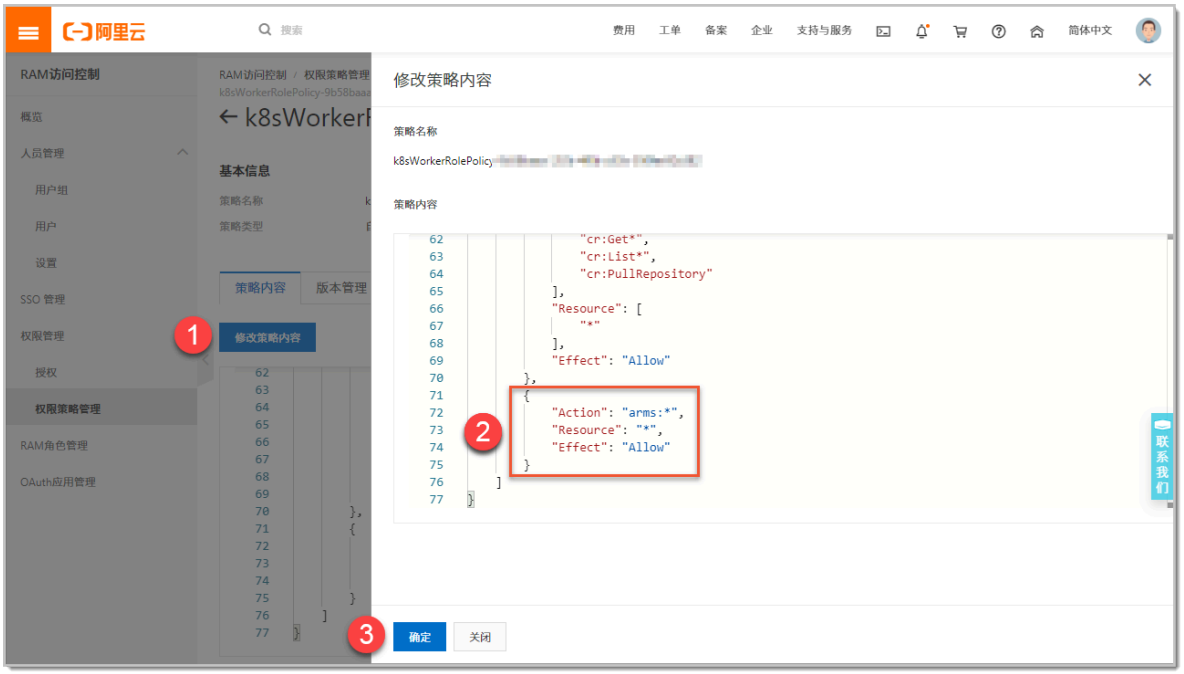


4. 在 RAM 访问控制控制台的 RAM 角色管理页面上，单击权限管理页签上的目标权限策略名称链接。

5. 在策略内容页签上单击修改策略内容，并在右侧的修改策略内容面板将以下内容添加到策略内容中，最后单击确定。

```
{
    "Action": "arms:*",
    "Resource": "*",
    "Effect": "Allow"
```

```
}
```



为 Java 应用开启 ARMS 应用监控

以下步骤分别对应创建新应用和已有应用这两种情况。

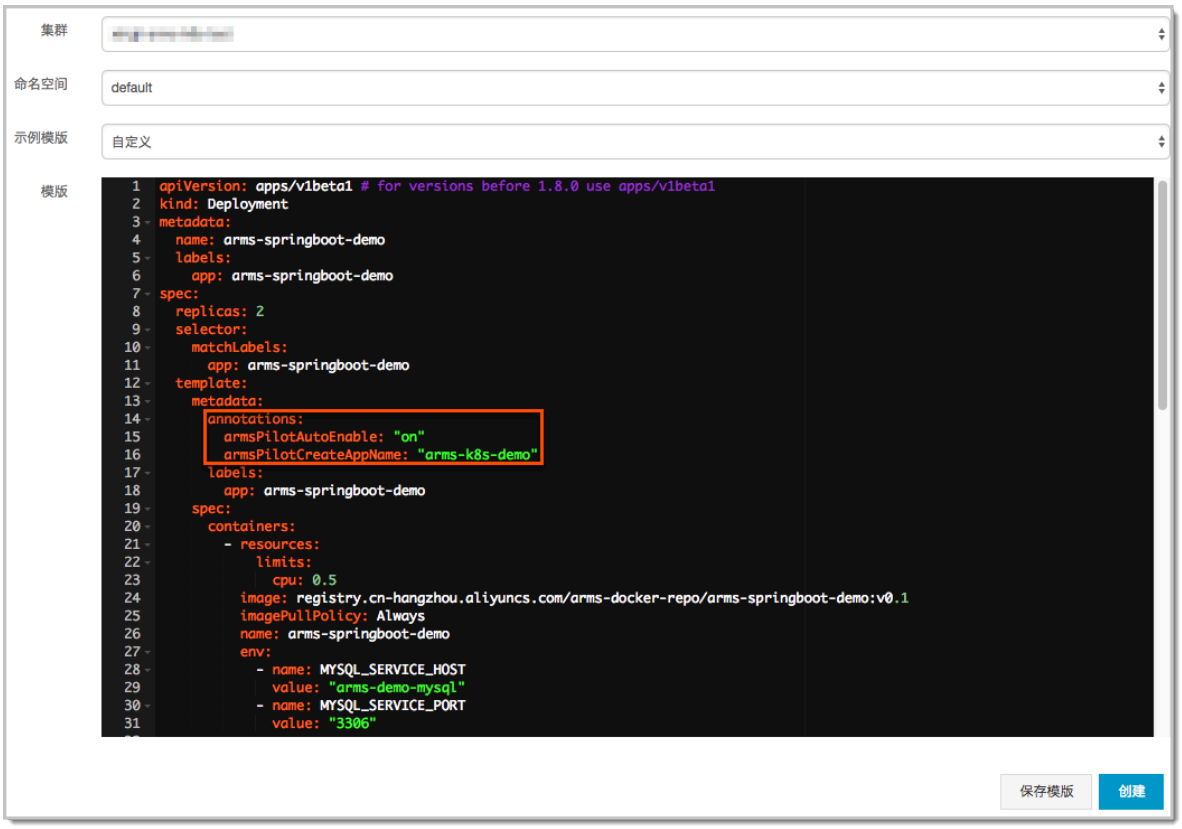
如需在创建新应用的同时开启 ARMS 应用监控，请按以下步骤操作：

1. **在容器服务 Kubernetes 版控制台左侧导航栏选择应用 > 无状态。**
2. **在无状态（Deployment）页面右上角单击使用模板创建。**
3. **在使用模板创建页面上选择集群、命名空间和示例模板，并在模板（YAML 格式）中将以下 `annotations` 添加到 *spec > template > metadata* 层级下。**

> **说明：**
> **请将 *<your-deployment-name>* 替换为您的应用名称。**

```
annotations:
  armsPilotAutoEnable: "on"
```

```
  armsPilotCreateAppName: "<your-deployment-name>"
```



创建一个无状态（Deployment）应用并开启 ARMS 应用监控的完整 YAML 示例模板如下：

如需为现有应用开启 ARMS 应用监控，请按以下步骤操作：

1. 在容器服务 Kubernetes 版控制台左侧导航栏选择应用 > 无状态或应用 > 有状态。
2. 在无状态（Deployment）或有状态（StatefulSet）页面上，选择集群和命名空间，并在目标应用右侧操作列中选择更多 > 查看 Yaml。
3. 在编辑 YAML 对话框中将以下 `annotations` 添加到 *spec > template > metadata* 层级下，并单击更新。

> **说明：**
> 请将 *<your-deployment-name>* 替换为您的应用名称。

```
annotations:
  armsPilotAutoEnable: "on"
```

```
  armsPilotCreateAppName: "<your-deployment-name>"
```

为 PHP 应用开启 ARMS 应用监控

以下步骤分别对应创建新应用和已有应用这两种情况。

如需在创建新应用的同时开启 ARMS 应用监控，请按以下步骤操作：

1. 在容器服务 Kubernetes 版控制台左侧导航栏选择应用 > 无状态。
2. 在无状态（Deployment）页面右上角单击使用模板创建。
3. 在使用模板创建页面上选择集群、 命名空间和示例模板，并在模板（YAML 格式）中将以下 `annotations` 添加到 *spec > template > metadata* 层级下。

   说明:

   请将 *<your-deployment-name>* 替换为您的应用名称。

   ```
   annotations:
     armsPilotAutoEnable: "on"
     armsPilotCreateAppName: "<your-deployment-name>"
     armsAppType: PHP
   ```

4. （本步骤仅限首次安装时）请修改安装 arms-pilot 的命名空间下的 *ConfigMap arms-php.ini* 文件，该文件内容为 *php.ini* 默认配置。

   说明:

   在 `extension=/usr/local/arms/arms-php-agent/arms-7.2.so` 配置中，*arms-7.2.so* 中的 7.2 为您的 PHP 版本，可使用的值为 5.4、5.5、5.6、7.0、7.1、7.2。

5. 将 *arms-php.ini ConfigMap* 配置项添加到 *php.ini* 文件的 *spec > template > spec > containers* 配置下，将 mountPath 设置为您的 *php.ini* 文件路径。

   ```
   volumeMounts:
           - name: php-ini
             mountPath: /etc/php/7.2/fpm/php.ini
             subPath: php.ini
   ```

   ```
   volumes:
         - name: php-ini
           configMap:
             name: arms-php.ini
   ```

   创建一个无状态（Deployment）应用并开启 ARMS 应用监控的完整 YAML 示例模板如下：

如需为现有应用开启 ARMS 应用监控，请按以下步骤操作：

1. 在容器服务 Kubernetes 版控制台左侧导航栏选择应用 > 无状态或应用 > 有状态。
2. 在无状态（Deployment）或有状态（StatefulSet）页面上，选择集群和命名空间，并在目标应用右侧操作列中选择更多 > 查看 Yaml。
3. 在编辑 YAML 对话框中将以下 `annotations` 添加到 *spec > template > metadata* 层级下，并单击更新。

> **说明:**
> 请将 *<your-deployment-name>* 替换为您的应用名称。

```
annotations:
  armsPilotAutoEnable: "on"
  armsPilotCreateAppName: "<your-deployment-name>"
  armsAppType: PHP
```

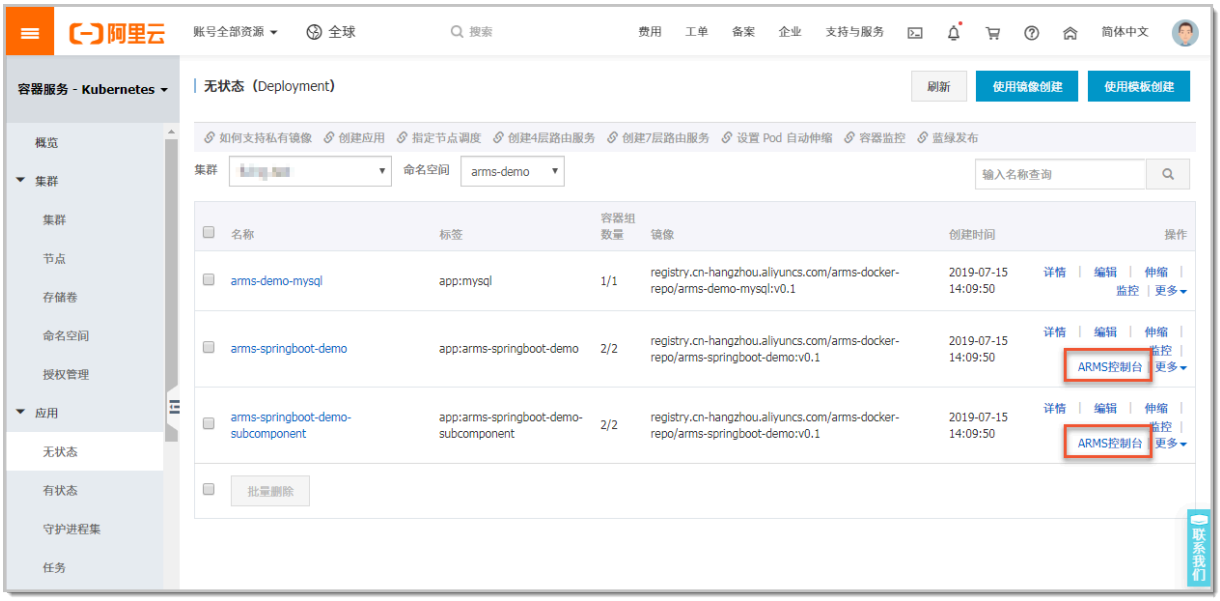
4. 挂载 *arms-php.ini ConfigMap* 项到 *php.ini* 文件 *spec > template > spec > containers* 下，将 mountPath 设置为您的 *php.ini* 文件路径。

```
volumeMounts:
        - name: php-ini
          mountPath: /etc/php/7.2/fpm/php.ini
          subPath: php.ini
```

```
volumes:
      - name: php-ini
        configMap:
          name: arms-php.ini
```

预期结果

在无状态（Deployment）或有状态（StatefulSet）页面上，目标应用的操作列将出现 ARMS 控制台按钮。



说明:

若操作列没有出现 ARMS 控制台，请检查您是否已授权容器服务访问 ARMS 资源。

后续步骤

完成上述步骤后，您就为部署在容器服务 Kubernetes 版中的应用开启了 ARMS 应用监控。在目标应用的操作列中单击 ARMS 控制台，将进入 ARMS 控制台的应用监控页面。ARMS 应用监控具备以下能力：

1. 展示应用总体性能关键指标，自动发现应用拓扑

2. 3D 拓扑图能立体展示应用、服务和主机的健康状况，以及应用的上下游依赖关系，帮助您快速定位诱发故障的服务、被故障影响的应用和关联的主机等，全方位地诊断故障根源，从而快速排除故障。

3. 捕获异常事务和慢事务，获取接口的慢 SQL、MQ 堆积分析报表或者异常分类报表，对错、慢等常见问题进行更细致的分析。

4. 自动发现和监控应用代码中常见的 Web 框架和 RPC 框架，并自动统计 Web 接口和 RPC 接口的调用量、响应时间、错误数等指标。

13.3 架构感知监控

本文为您介绍应用高可用性服务（Application High Availability Service，简称 AHAS）架构感知提供了针对容器服务 Kubernetes 环境的可视化展示能力。

前提条件

- 您已创建容器服务 Kubernetes 集群。具体步骤，参见#unique_258。
- 您已开通 AHAS。具体步骤，参见#unique_259。

背景信息

Kubernetes中的业务是运行在节点组成的资源池上，使得定位Pod的调用链路以及拓扑关系非常复杂。那么如何以可视化的方式监控Kubernetes中的负载状态，更好地可视化集群中流量的吞吐是非常重要的问题。阿里云推出了应用高可用服务（AHAS），是一款专注于提高应用高可用能力的云产品，提供应用架构自动探测，故障注入式高可用能力评测和一键流控降级等功能，可以快速低成本的提升应用可用性。AHAS 产品详细介绍请参考#unique_260。

操作步骤

1. 开通 AHAS 服务。

   检查是否开通了 AHAS：访问开通 AHAS 服务页面验证。如果已开通，会提示跳转到 AHAS 控制台。

2. 授权 AHAS 访问容器服务信息。

   访问云资源访问授权页面，单击同意授权。

3. 安装 AHAS Pilot。

   a) 登录容器服务管理控制台。

   b) 在 Kubernetes 菜单下，在左侧导航栏选择市场 > 应用目录，在右侧选中ack-ahas-pilot。

   c) 在应用目录 - ack-ahas-pilot页面，单击参数页签，可查看安装该组件的默认参数值。

   如需修改，请参考以下参数说明。

   表 13-1: 参数说明

参数	说明	默认值
`controller.region_id`	必选项。目标集群所在的地域，例如 cn-hangzhou、cn-beijing、cn-shenzhen、cn-shanghai。	cn-hangzhou
`resources.requests.cpu`	AHAS Pilot 占用的 CPU。	0.05
`resources.requests.memory`	AHAS Pilot 占用的内存。	200 Mi
`resources.limits.cpu`	AHAS Pilot 占用的 CPU 最高限制为0.2。	0.2
`resources.limits.memory`	AHAS Pilot 占用的内存最高限制为200 Mi。	200 Mi

   d) 在应用目录-ack-ahas-pilot 页面右侧的创建区域，选择集群、命名空间，并自定义发布名称，单击创建，添加 AHAS 应用高可用服务组件。

4. 查看 AHAS 服务数据。

   创建完成后，您可以登录AHAS 控制台，查看 AHAS 服务数据。

   图 13-1: pod下的视图

   图 13-2: service下的视图

   图 13-3: 主机层的可用区视图

   如果概览页中架构感知容器组数为 0，或者架构感知中数据为空，请 查是否选择了正确的地域（Region）：在 AHAS 控制台左上角选择的地域，需要与安装 AHAS Pilot 时参数`controller.region_id`配置的地域一致。

   查看参数`controller.region_id`的步骤如下：

   a. 在 Kubernetes 菜单下，在左侧导航栏选择应用 > 发布。
   b. 在发布页面，选择 Helm 页签。
   c. 找到发布名称为 ahas 的集群，单击操作列的详情。
   d. 单击参数页签，查看地域参数`env.region`的值。

13.4 事件监控

事件监控是Kubernetes中的另一种监控方式，可以弥补资源监控在实时性、准确性和场景上的缺欠。Kubernetes的架构设计是基于状态机的，不同的状态之间进行转换则会生成相应的事件，正常的状态之间转换会生成Normal等级的事件，正常状态与异常状态之间的转换会生成Warning等级的事件。开发者可以通过获取事件，实时诊断集群的异常与问题。

背景信息

kube-eventer是阿里云容器服务维护的开源Kubernetes事件离线工具，可以将集群的事件离线到钉钉、SLS等系统，并提供不同等级的过滤条件，实现事件的实时采集、定向告警、异步归档。更多内容请参见kube-eventer。

通过以下三种场景为您介绍事件监控。

场景1：使用钉钉实现Kubernetes监控告警

使用钉钉机器人监控并告警Kubernetes的事件是一个非常典型的ChatOps实现。具体的操作步骤如下：

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。
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 > 无状态，进入 无状态（Deployment）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。

配置	说明

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:`
	`  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=[可选参数：Normal或者Warning，默认值为：Warning]  #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。

场景2：使用SLS离线Kubernetes事件

SLS（Log Service）可以将Kubernetes的事件以更持久的方式进行存储，从而提供更多的事件归档、审计的能力。具体的操作步骤如下：

1. 创建project与logstore。
   a) 登录 日志服务管理控制台。
   b) 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。

   c) 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。
   d) 默认进入日志库页面，单击右上角的创建。
   e) 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。

   f) 创建完毕后，页面会提示您创建数据接入向导。
   g) 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。

2. 在 Kubernetes 集群中配置 log4j。
   a) 连接到您的 Kubernetes 集群。

      具体操作参见#unique_262 或 #unique_43。
   b) 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: extensions/v1beta1
      kind: Deployment
      metadata:
        name: kube-eventer
      ```

```
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: kube-eventer
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      serviceAccount: admin
      containers:
      - name: kube-eventer
        image: registry.cn-hangzhou.aliyuncs.com/acs/kube-eventer-
amd64:v1.0.0-d9898e1-aliyun
        imagePullPolicy: IfNotPresent
        command:
        - /kube-eventer
        - --source=kubernetes:https://kubernetes.default
        - --sink=sls:https://sls.aliyuncs.com?logStore=[your_logst
ore]&project=[your_project]
```

其中：

- `project`：**您所使用的阿里云日志服务 Project 的名称。本示例中为 k8s-log4j。**
- `logStore`：**您所使用的阿里云日志服务 Logstore 的名称。本示例中为 k8s-logstore。**

c) **在命令行中执行以下命令，创建 deployment。**

```
kubectl create -f demo-deployment.yaml
```

d) **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
```

3. **操作集群（例如，删除Pod或者创建应用等）产生事件后，登录日志服务控制台查看数据采集。**

请参见#unique_263/unique_263_Connect_42_section_uzx_4o9_uyc。

4. 设置索引与归档。请参见#unique_264。
   a) 在日志服务控制台单击Project名称。
   b) 在查询分析列，单击查询。
   c) 单击右上角的开启索引。
   d) 从右侧滑出查询分析页面，指定字段并进行配置。
   e) 单击确定。
      此时会出现日志查询与分析页面。

      说明:
      · 索引配置在1分钟之内生效。
      · 开启或修改索引后，新的索引配置只对新写入的数据生效。

   f) 如果您需要设置离线归档与计算的场景，可以在logstore上面将数据投递给MaxCompute或者OSS。请参见#unique_265和投递日志到OSS。

场景3：使用node-probelm-detector与eventer实现节点异常告警

node-problem-detector是Kubernetes节点诊断的工具，可以将节点的异常，例如：Docker Engine Hang、Linux Kernel Hang、网络出网异常、文件描述符异常转换为Node的事件，集合kube-eventer可以实现节点事件告警的闭环。具体的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，在左侧导航栏选择市场 > 应用目录，在右侧选中ack-node-probelm-detector。

3. 在应用目录 - ack-node-probelm-detector中，单击参数，可以看到node-problem-detector的默认配置。

您可以参考以下参数说明，配置kube-eventer的离线通道。

表 13-2: 参数说明

参数	说明	默认值
npd.image.repository	npd的镜像地址	registry.cn-beijing.aliyuncs.com/acs/node-problem-detector
npd.image.tag	npd的镜像tag	v0.6.3-16-g30dab97
alibaba_cloud_plugins	开启阿里云插件的列表	fd_check
eventer.image.repository	eventer的镜像地址	registry.cn-hangzhou.aliyuncs.com/acs/eventer
eventer.image.tag	eventer的镜像tag	v1.6.0-4c4c66c-aliyun
eventer.image.pullPolicy	eventer的镜像下载方式	IfNotPresent
eventer.sinks.sls.enabled	是否开启eventer sls的离线链路	false
eventer.sinks.sls.project	日志服务的project名称	-
eventer.sinks.sls.logstore	日志服务的project下logstore的名称	-
eventer.sinks.dingtalk.enabled	是否开始eventer 钉钉的离线链路	false
eventer.sinks.dingtalk.level	事件离线的等级	warning
eventer.sinks.dingtalk.label	打印事件的额外标签	-
eventer.sinks.dingtalk.token	过滤的资源类型	-
eventer.sinks.dingtalk.monitorkinds	过滤的资源命名空间	-

参数	说明	默认值
eventer.sinks.dingtalk.monitornamespaces	钉钉机器人的Token	-

4. 在右侧的创建页面，在右侧选择对应的集群，同时可以看到命名空间已设定为 istio-system ，发布名称已设定为ack-node-probelm-detector，然后单击创建。

   单击应用 > 容器组，选择目标集群和命名空间，在容器组列表中可以看到守护进程集中的ack-node-problem-detector-daemonset运行正常。

此时node-problem-detector与eventer都正常运行后，可以通过配置的eventer的离线通道进行数据的离线或者报警。

13.5 开源Prometheus监控

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。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_42。
- 您已连接到Master节点，方便快速查看节点标签等信息，参见#unique_43。

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。

操作步骤

1. 部署Prometheus监控方案。
   a) 登录容器服务管理控制台。
   b) 在 Kubernetes 菜单下，在左侧导航栏选择市场 > 应用目录，在右侧选中ack-prometheus-operator。
   c) 在应用目录-ack-prometheus-operator 页面右侧的创建区域，选择目标集群后，单击创建，添加 Prometheus-operator。

   查看部署结果：

   A. 执行以下命令，将集群中的Prometheus 映射到本地9090端口。

   ```
   kubectl port-forward svc/ack-prometheus-operator-prometheus
   9090:9090 -n monitoring
   ```

   B. 在浏览器中访问*localhost:9090*，即可查看Prometheus。
   C. 选择菜单栏Status > Targets，查看所有采集任务。

   如果所有任务的状态为UP，表示所有采集任务均已正常运行。

2. 查看与展示数据聚合。
   a) 执行以下命令，将集群中的Grafana映射到本地3000端口。

   ```
   kubectl -n monitoring port-forward svc/ack-prometheus-operator-
   grafana 3000:80
   ```

   b) 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。

3. 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-
operated 9093
```

上述步骤为您简单的介绍了如何部署Prometheus监控方案，下面将为您详细介绍一些不同场景下的配置。

告警配置

如果您想要设置告警的接收方式以及设置告警接收的模板，请参考如下对alertmanager的config进行配置。

· 设置告警接收方式

prometheus-operator支持钉钉告警和邮件告警，如果需要开启，请参见以下操作进行配置。

- 配置钉钉告警

如果需要添加钉钉告警功能，请在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到dingtalk字段将enabled设置为true，Token字段填入钉钉

的webhook地址；alertmanager的config字段找到receiver，填写您receivers中对应的钉钉告警名称（默认为 Webhook）。

例如您有两个钉钉机器人，操作示例如下：

1. 替换钉钉的token配置

   在您的钉钉机器人中，分别拷贝webhook地址替换为dingtalk1和dingtalk2的地址。即，使用webhook地址替换下图中的*https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxxxx*。

   图 13-4: 钉钉的token配置

```
##
podAnnotations: {}
service:
  ## Configuration for dingtalk service
  ##
  annotations: {}
clusterIP: ""
externalIPs: ""
loadBalancerIP: ""
type: ClusterIP
image:
  repository: timonwong/prometheus-webhook-dingtalk
  tag: latest
  pullPolicy: IfNotPresent
# pls fill your dingtalk rebot token
# ref: https://open-doc.dingtalk.com/microapp/serverapi2/qf2nxq
token:
- dingtalk1=https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxxx
- dingtalk2=https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxxx
```

2. 修改receivers

   alertmanager的config字段找到receiver，填写您receivers中对应的钉钉告警名称，本例中分别为webhook1和webhook2。

3. 修改url的值

   将url中的值替换成实际的dingtalk的值，本例中为dingtalk1和dingtalk2。

```
config:
  global:
    resolve_timeout: 5m
  route:
    group_by: ['job']
    group_wait: 1m
    group_interval: 1m
    repeat_interval: 2m
    receiver: "webhook1"
    routes:
    - match:
        alertname: Watchdog
      receiver: "webhook2"
  receivers:
  #- name: "null"
  - name: webhook1
    webhook_configs:
    - url: http://ack-prometheus-operator-dingtalk:8060/dingtalk/dingtalk1/send
      send_resolved: true
  - name: webhook2
    webhook_configs:
    - url: http://ack-prometheus-operator-dingtalk:8060/dingtalk/dingtalk2/send
      send_resolved: true
```

1

2

说明:

如果需要添加多个钉钉机器人，请将前面的webhook依次递增即可。

- **配置邮件告警**

如果需要添加邮件告警功能，请在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，将红色选框内的邮件信息补充完整，在alertmanager的config字段找到receiver，填写您receivers中对应的邮件告警名称（默认为 mail）。

```
config:
  global:
    resolve_timeout: 5m
  route:
    group_by: ['job']
    group_wait: 1m
    group_interval: 1m
    repeat_interval: 2m
    receiver: "null"
    routes:
    - match:
        alertname: Watchdog
      receiver: "null"
  receivers:
  - name: "null"
  #- name: webhook
  #  webhook_configs:
  #  - url: http://ack-prometheus-operator-alertmanager.monitoring:8060/dingtalk/ops_dingding/send
  #    send_resolved: true
  #- name: 'mail'
  #  email_configs:
  #  - to: 'xxxxxxx@qq.com'          #receive address
  #    smarthost: 'smtp.163.com:465' #stmp server address
  #    from: 'xxxxx@163.com'         #sender address
  #    auth_username: 'xxxxx@163.com'  #email-username
  #    auth_password: 'xxxxxxxx'       #email-password(Authorization code)
  #    require_tls: false              #tls switch
  #    send_resolved: true
```

- 设置告警接收模板

  您可以在alertmanager的templateFiles定义我们的告警模板，示例如下：

```
##
templateFiles: {}
#
# An example template:
#   template_1.tmpl: |-
#       {{ define "cluster" }}{{ .ExternalURL | reReplaceAll ".*alertmanager\\.(.*)" "$1" }}{{ end }}
#
#       {{ define "slack.myorg.text" }}
#       {{- $root := . -}}
#       {{ range .Alerts }}
#         *Alert:* {{ .Annotations.summary }} - `{{ .Labels.severity }}`
#         *Cluster:*  {{ template "cluster" $root }}
#         *Description:* {{ .Annotations.description }}
#         *Graph:* <{{ .GeneratorURL }}|:chart_with_upwards_trend:>
#         *Runbook:* <{{ .Annotations.runbook }}|:spiral_note_pad:>
#         *Details:*
#           {{ range .Labels.SortedPairs }} • *{{ .Name }}:* `{{ .Value }}`
#           {{ end }}
```

存储配置

支持通过TSDB和云盘存储，您可以通过如下操作设置prometheus的数据存放位置。

- TSDB配置

  如果需要将prometheus的数据存入TSDB，请在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到tsdb字段，将enabled设置为true，并把remoteRead和remoteWrite的url填写完整，url填写可以参见快速入门。

```
    ##  remoteWrite must Indicate.
    tsdb:
      #default is false
      enabled: false
      # the tsdb's specification
      specification: mlarge
      ## The remote_read spec configuration for Prometheus.
      ## ref: https://github.com/coreos/prometheus-operator/blo
.md#remotereadspec
      remoteRead:
        - url: ""
        # read_recent: true
      # - url: "http://ts-xxxxxxxxxxx.hitsdb.rds.aliyuncs.com:
      ## The remote_write spec configuration for Prometheus.
      ## ref: https://github.com/coreos/prometheus-operator/blo
.md#remotewritespec
      remoteWrite:
        - url: ""
        # - url: "http://ts-xxxxxxxxxxx.hitsdb.rds.aliyuncs.com:
```

- 云盘存储配置

  ack-prometheus-operator默认带了阿里云盘存储，可以选择在prometheus或者alertmanager中选择云盘存储，请在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到alertmanager的storage字段或者prometheus的storageSpec字段来

配置云盘存储，这里的storageClassName需要填云盘类型，accessModes需要配置访问模式，srorage需要填写需要容量。

```
# volumeClaimTemplate:
#   spec:
#     storageClassName: gluster
#     accessModes: ["ReadWriteOnce"]
#     resources:
#       requests:
#         storage: 50Gi
#   selector: {}
```

说明:

本例中，您需要在Prometheus中启用一个SSD云盘，则在Prometheus的storageSpec字段中，将`storageClassName`配置为`alicloud-disk-ssd`，`accessModes`配置为`ReadWriteOnce`，`srorage`配置为`50Gi`，示例如下。

```
storageSpec:
#  volumeClaimTemplate:
#    spec:
#      storageClassName: alicloud-disk-ssd
#      accessModes: ["ReadWriteOnce"]
#      resources:
#        requests:
#          storage: 50Gi
#    selector: {}
```

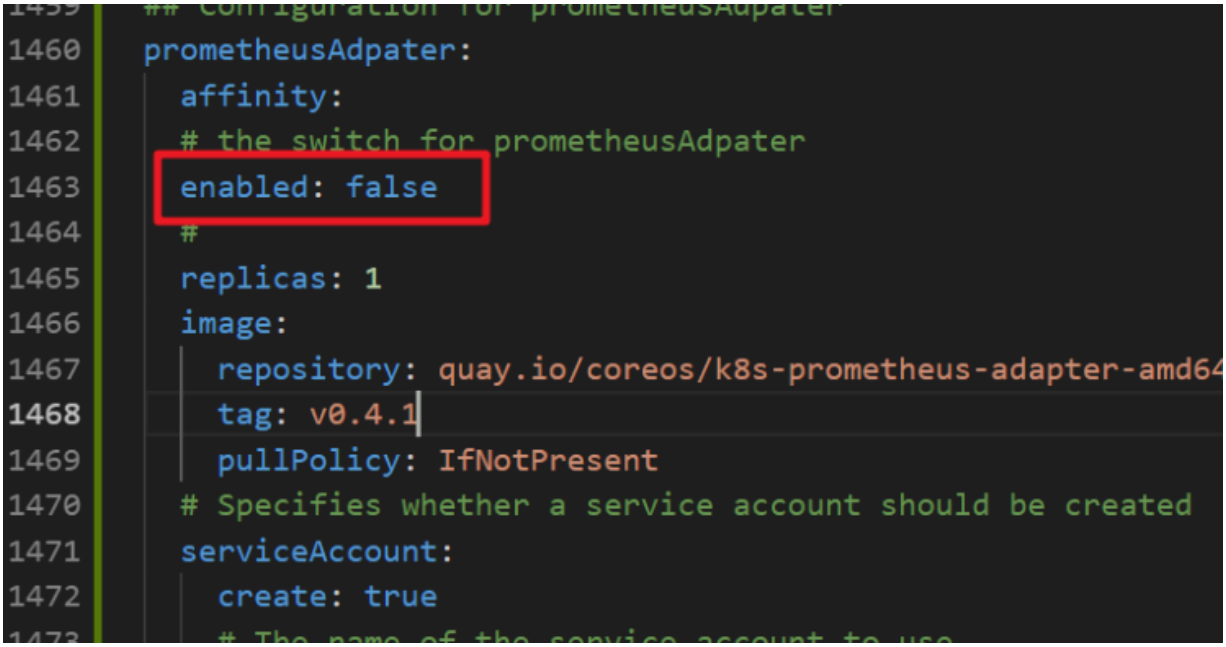
配置完成后，您可以在ECS控制台，选择存储与快照 > 云盘，查看您启用的SSD云盘。

如果您想要复用云盘，请参见云盘存储卷使用说明。

通过自定义指标来实现HPA

您可以在prometheus中使用prometheus-adapter自定义指标，通过开启adapter，使得集群中的pod根据指标，自动增加和减少，提升资源利用率。

您可以在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到prometheus-adapter字段，将enabled为true，即可启用adapter。



配置完成就，您可以通过执行如下命令，来查看是否正确开启adapter，如果您需要自定义指标，请参见prometheus-adapter。

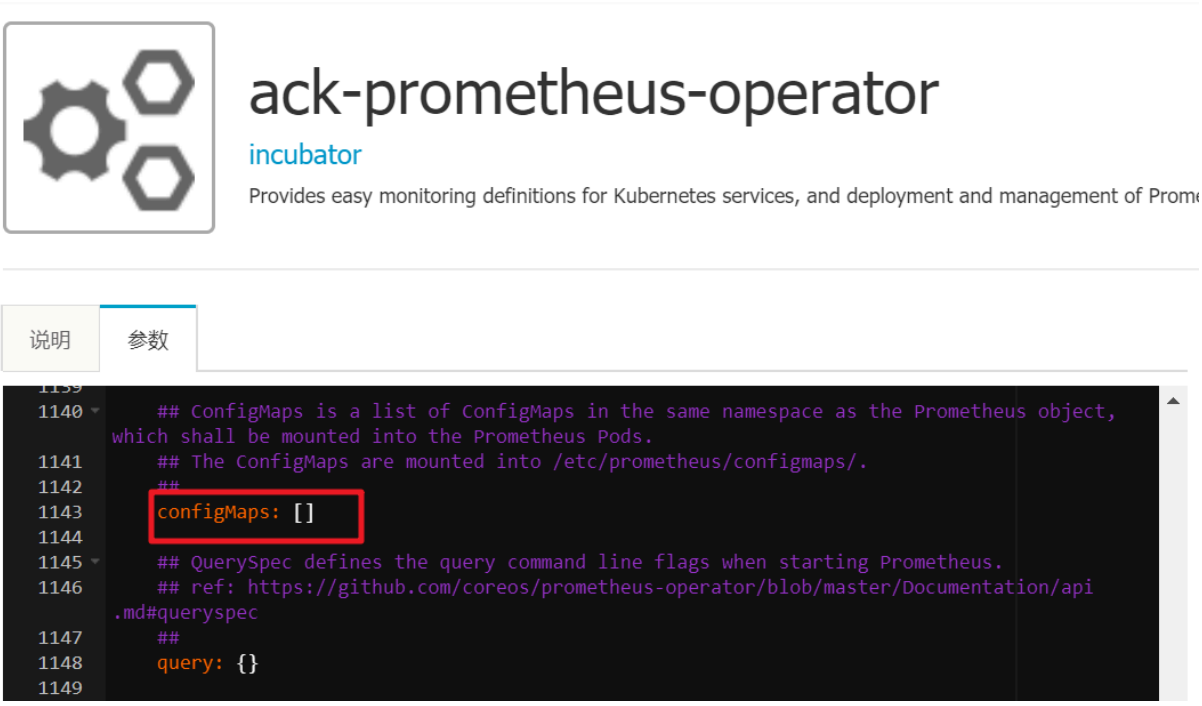
```
kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"
```

Prometheus挂载自定义configmap

以下介绍两种操作，将configmap挂载到pod中的*/etc/prometheus/configmaps/*路径下。

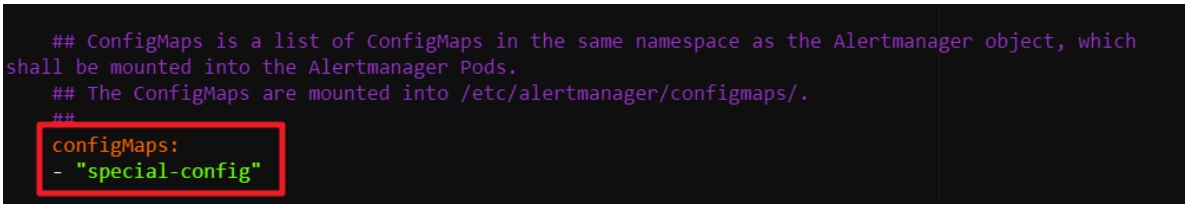
如果您是第一次部署prometheus-operator，请按照如下进行操作：

您可以在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到prometheus的configmaps字段，填入您自定义configmap名称。



如果集群中已有部署好的prometheus-operator，请按照如下进行操作：

1. 在 Kubernetes 菜单下，单击左侧导航栏的应用 > 发布，进入发布列表。
2. 单击Helm页签，在目标应用右侧单击更新，进入更新发布页面。
3. 在prometheus和alertmanager的configmaps字段中，填入您自定义configmap名称后，单击更新。



说明：

例如，您想要定义一个名称为`special-config`的configmap，里面包含prometheus的config文件。如果您想要将在prometheus的pod启动时，将其作为 --config.file的参数，那我们就可以在prometheus的configmaps字段添加如下字段，就可以将其挂载到我们的pod中了，其挂载路径为*/etc/prometheus/configmaps/*。

special-config的yml定义如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
data:
  config.yaml: |-
    global:
      # 抓取间隔，默认为 1m
      [ scrape_interval: <duration> | default = 1m ]

      # 抓取超时时间，默认为 10s
      [ scrape_timeout: <duration> | default = 10s ]

      # 规则评估间隔，默认为 1m
      [ evaluation_interval: <duration> | default = 1m ]

    # 抓取配置
    scrape_configs:
      [ - <scrape_config> ... ]

    # 规则配置
    rule_files:
      [ - <filepath_glob> ... ]

    # 告警配置
    alerting:
      alert_relabel_configs:
        [ - <relabel_config> ... ]
      alertmanagers:
        [ - <alertmanager_config> ... ]
```

prometheus的configmaps字段配置如下。

```
## ConfigMaps is a list of ConfigMaps in the same namespace as the Prometheu
## The ConfigMaps are mounted into /etc/prometheus/configmaps/.
##
configMaps:
- "special-config"
- "detail-config"
```

grafana配置

- Dashboard的外挂配置

  如果您想将Dashboard文件以configmap的方式挂载到grafana pod中去，您可以在应用目录-ack-prometheus-operator 页面，单击下方的参数页签，找到`extraConfigmapMounts`，您可以在下图的字段中进行挂载配置。

```
extraConfigmapMounts: []
# - name: certs-configmap
#   mountPath: /etc/grafana/ssl/
#   configMap: certs-configmap
#   readOnly: true
```

  说明:

  - 先确保dashboard以confgimap的形式存在于集群中。

    configmap的labels必须要和其他的configmap保持一致。
  - 在grafana的`extraConfigmapMounts`字段填入我们的configmap信息以及挂载信息。
  - mountPath是*/tmp/dashboards/*。
  - configmap是您定义的configmap的名称。
  - name是您Dashboard的json名称。

- Dashboard的持久化

  grafana支持dashboard的持久化操作，如果我们需要导出我们的dashboard到本地，我们可以将需要保存的dashboard通过导出为json文件的方式，将其保存到本地。请参见grafana导出。

常见问题

- 钉钉配置后，没有收到告警。

  1. 获取钉钉的webhook地址。请参见场景1：使用钉钉实现Kubernetes监控告警。
  2. 找到dingtalk字段，将enabled设置为true，将Token字段填入钉钉的webhook地址。请参见#unique_267/unique_267_Connect_42_section_kbc_qjj_hn3中的钉钉告警配置。

- 部署prometheus-operator时报错。

  报错信息如下

  ```
  Can't install release with errors: rpc error: code = Unknown desc =
  object is being deleted: customresourcedefinitions.apiextensions.k8s
  .io "xxxxxxxx.monitoring.coreos.com" already exists
  ```

  在卸载prometheus-operator的时候没有将上一次部署的自定义资源（crd）及时清理掉，执行如下命令，删除crd并重新部署。

  ```
  kubectl delete crd prometheuses.monitoring.coreos.com && kubectl
  delete crd prometheusrules.monitoring.coreos.com && kubectl delete
   crd servicemonitors.monitoring.coreos.com &&  kubectl delete crd
  alertmanagers.monitoring.coreos.com
  ```

- 邮件告警没有生效

  邮件告警没有生效，有可能是因为`smtp_auth_password`填写的是您的登录密码，而非授权码。另外stmp的服务器地址需要加端口号。

- 如果在单击yaml更新时，出现如下图所示的错误。

  当前集群暂时无法访问，请稍后重试或提交工单反馈。

  rpc error: code = ResourceExhausted desc = grpc: trying to send message larger than max (21329544 vs. 20971520)

  此问题原因是tiller的配置文件过大，导致的集群无法访问，您可以先将部分注释删除，再将配置文件以configmap形式，挂载到pod中，目前prometheus-operator只支持prometheus和alertmanager pod的挂载，步骤如您可以参见Prometheus挂载自定义configmap中的方法二。

- 部署prometheus-operator后，如何开启里面的功能?

  当我们已经部署好prometheus-operator后，如果有开启部分功能，我们可以发布 > Helm > 更新中找到我们的yaml，找到对应的开关，做相应的设置，然后单击更新，就可以开启您想要的功能。

- TSDB和阿里云云盘的选择

  TSDB支持的地域比较少，而阿里云云盘是全域支持，数据回收策略请参见以下配置。

  ```
  1212
  1213    ## How long to retain metrics
  1214    ##
  1215    retention: 10d
  1216
  ```

14 安全管理

14.1 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#unique_9。

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。

详细信息参见 给容器服务的 Kubernetes 集群部署 network policy 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#unique_29 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。

14.2 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，以及如何通过日志服务收集、分析审计日志，并根据您的需求为审计日志设置自定义的告警规则。

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: 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。

查看审计报表

容器服务Kubernetes版内置了3个审计日志报表。通过报表，您可以获取以下内容：

- 所有用户以及系统组件对集群执行的重要操作；
- 操作的源地址、源地址所属区域以及分布；
- 各类资源的详细操作列表；
- 子账号操作详细列表；
- 重要操作（登录容器、访问保密字典、删除资源等）的详细列表。

说明：

- 在2019年1月13日后创建的集群中，若已经选择日志服务，则会自动开通审计报表相关功能。若未开通，请参考手动开通审计报表。
- 请不要修改审计报表。如果您有自定义审计报表的需求，请在日志服务管理控制台创建新的报表。

您可以通过以下几种方式访问审计报表：

- 登录容器服务管理控制台。在集群列表的操作列表中，单击更多 > 集群审计。

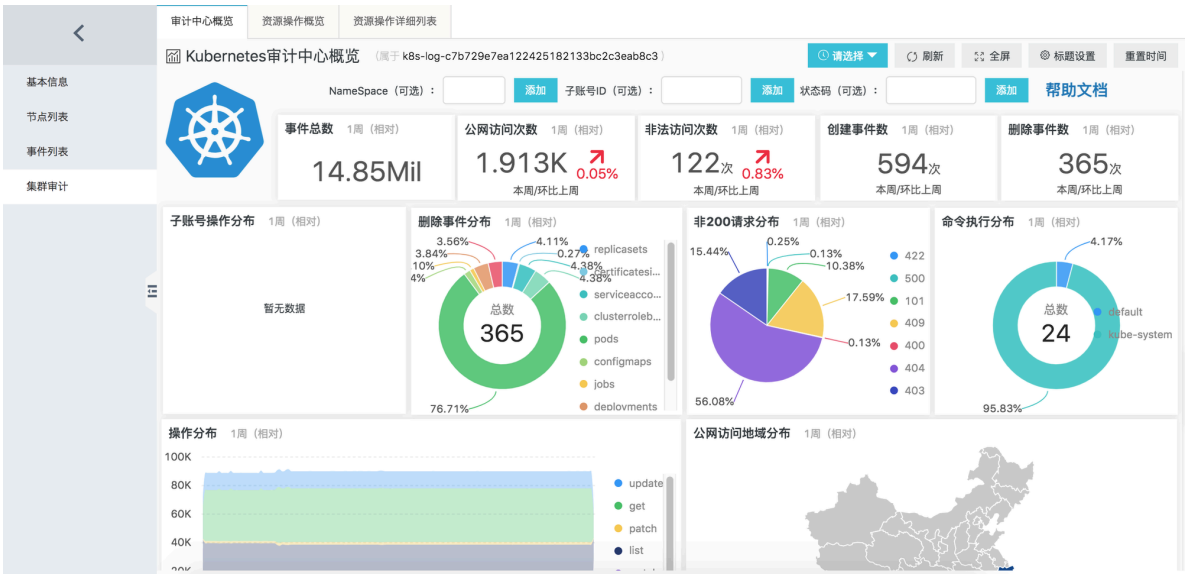
- 登录容器服务管理控制台。在集群列表中单击集群名称，进入到集群信息页面。在左侧导航栏列表中单击集群审计。

审计报表说明

apiserver审计共3个报表。分别是：审计中心概览、资源操作概览以及资源操作详细列表。

- **审计中心概览**

  审计中心概览展示Kubernetes集群中的事件整体概览以及重要事件（公网访问、命令执行、删除资源、访问保密字典等）的详细信息。



说明:

在该报表中，默认显示一周的统计信息。您可以自定义选择统计时间范围。此外，该报表支持指定Namespace、子账号ID、状态码进行筛选。您可以选择任一一项或多项组合筛选指定范围的事件。

- 资源操作概览

  资源操作概览展示Kubernetes集群中常见的计算资源、网络资源以及存储资源的操作统计信息。操作包括创建、更新、删除、访问。其中：

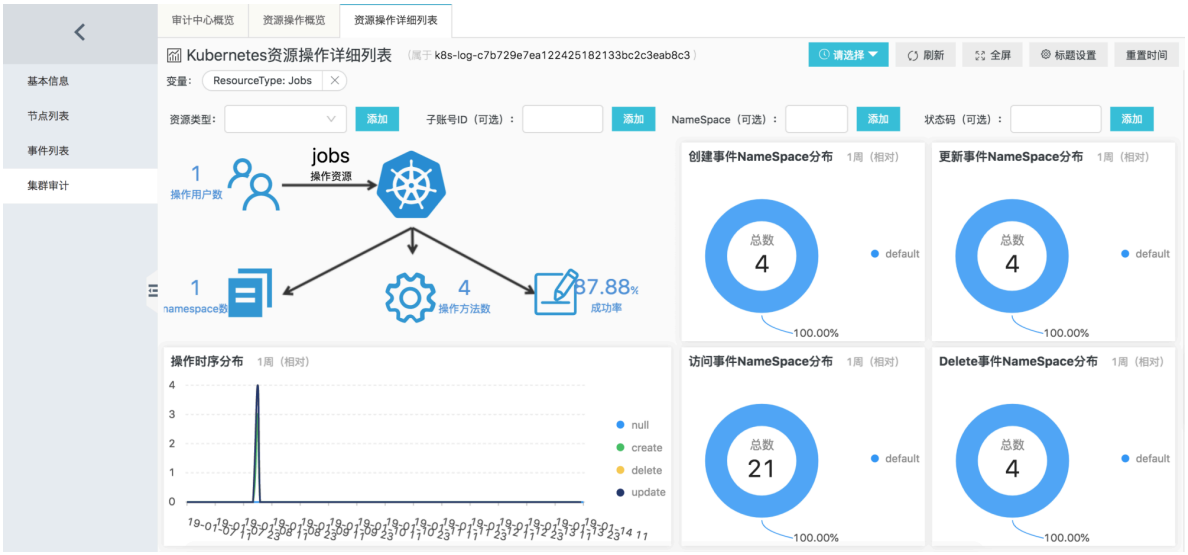
  - 计算资源包括：Deployment、StatefulSet、CronJob、DaemonSet、Job、Pod。
  - 网络资源包括：Service、Ingress。
  - 存储资源包括：ConfigMap、Secret、PersistentVolumeClaim。

说明：

- 在该报表中，默认显示一周的统计信息。您可以自定义选择统计时间范围。此外，该报表支持指定Namespace、子账号ID进行筛选。您可以选择任一一项或多项组合筛选指定范围的事件。
- 若您需要查看对应资源的详细操作事件，请使用资源操作详细列表。

- 资源操作详细列表

  该报表用于展示Kubernetes集群中某类资源的详细操作列表。您需要选择或输入指定的资源类型进行实时查询。该报表会显示：资源操作各类事件的总数、Namespace分布、成功率、时序趋势以及详细操作列表等。



> **说明：**
>
> - 若您需要查看Kubernetes中注册的CRD（CustomResourceDefinition）资源或列表中没有列举的其他资源，可以手动输入资源名的复数形式。例如CRD资源为AliyunLogConfig，则输入AliyunLogConfigs。
> - 在该报表中，默认显示一周的统计信息。您可以自定义选择统计时间范围。此外，该报表支持指定Namespace、子账号ID、状态码进行筛选。您可以选择任一一项或多项组合筛选指定范围的事件。

查看详细日志记录

如果您有自定义查询、分析审计日志的需求，可以进入日志服务管理控制台查看详细的日志记录。

1. 登录 日志服务管理控制台。
2. 单击左侧导航栏中的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：**
>
> - 在集群创建过程中，指定的日志Project中会自动添加一个名为audit-${clustereid}的日志库。
> - 审计日志的Logstore默认已经配置好索引。请不要修改索引，以免报表失效。

常见的审计日志搜索方式如下：

- 查询某一子账号的操作记录，直接输入子账号id，单击查询/分析。
- 查询某一资源的操作，直接输入资源名，单击查询/分析。
- 过滤掉系统组件的操作，输入`NOT user.username: node NOT user.username: serviceaccount NOT user.username: apiserver NOT user.username: kube-scheduler NOT user.username: kube-controller-manager`，单击查询/分析。

更多查询、统计方式，请参考日志服务查询分析方法。

设置告警

若您需要对某些资源的操作进行实时告警，可以通过日志服务的告警功能实现。告警方式支持短信、钉钉机器人、邮件、自定义WebHook和通知中心。详细操作方式请参考日志服务告警配置。

> **说明：**
>
> 对于审计日志的更多查询方式，可以参考审计报表中的查询语句。操作方式为：在日志服务Project详情页中，单击左侧导航栏中的仪表盘，进入指定的仪表盘（报表），展开指定分析图表右上角的折叠列表，并单击查看分析详情。详细操作方式请参考查看分析详情。

示例1：对容器执行命令时告警

某公司对于Kubernetes集群使用有严格限制，不允许用户登录容器或对容器执行命令，如果有用户执行命令时需要立即给出告警，并希望告警时能够显示用户登录的具体容器、执行的命令、操作人、事件ID、时间、操作源IP等信息。

- 查询语句为：

```
verb : create and objectRef.subresource:exec and stage:  ResponseSt
arted | SELECT auditID as "事件ID", date_format(from_unixtime(
__time__), '%Y-%m-%d %T' ) as "操作时间",  regexp_extract("requestURI
", '([^\?]*)/exec\?.*', 1)as "资源",  regexp_extract("requestURI",
 '\?(.*)', 1)as "命令" ,"responseStatus.code" as "状态码",
 CASE
 WHEN "user.username" != 'kubernetes-admin' then "user.username"
 WHEN "user.username" = 'kubernetes-admin' and regexp_like("
annotations.authorization.k8s.io/reason", 'RoleBinding') then
regexp_extract("annotations.authorization.k8s.io/reason", ' to User
 "(\w+)"', 1)
 ELSE 'kubernetes-admin' END
 as "操作账号",
CASE WHEN json_array_length(sourceIPs) = 1 then json_format(
json_array_get(sourceIPs, 0)) ELSE  sourceIPs END
as "源地址" limit 100
```

- 条件表达式为：`操作事件 =~ ".*"`。

示例2：apiserver公网访问失败告警

某集群开启了公网访问，为防止恶意攻击，需要监控公网访问的次数以及失败率，若访问次数到达一定阈值（10次）且失败率高于一定阈值（50%）则立即告警，并希望告警时能够显示用户的IP所属区域、操作源IP、是否高危IP等信息。

- 查询语句为：

```
* | select ip as "源地址", total as "访问次数", round(rate * 100, 2)
as "失败率%", failCount as "非法访问次数", CASE when security_check_ip
(ip) = 1 then 'yes' else 'no' end  as "是否高危IP",  ip_to_country(
ip) as "国家", ip_to_province(ip) as "省", ip_to_city(ip) as "市",
ip_to_provider(ip) as "运营商" from (select CASE WHEN json_array
_length(sourceIPs) = 1 then json_format(json_array_get(sourceIPs, 0
)) ELSE  sourceIPs END
as ip, count(1) as total,
sum(CASE WHEN "responseStatus.code" < 400 then 0
ELSE 1 END) * 1.0 / count(1) as rate,
count_if("responseStatus.code" = 403) as failCount
from log  group by ip limit 10000) where ip_to_domain(ip) != '
intranet'  having "访问次数" > 10 and "失败率%" > 50 ORDER by "访问次
数" desc limit 100
```

- 条件表达式为：`源地址 =~ ".*"`。

手动开通审计报表

若您还未开通审计报表，需手动开通审计报表。操作方式如下：

1. 开启apiserver审计日志。

   分别在三个master节点上，查看apiserver的POD配置，即查看启动参数、策略文件、环境变量和挂载目录是否配置了审计日志相关的内容：

   · 启动参数

   ```
   containers:
     - command:
       - kube-apiserver
       - --audit-log-maxbackup=10
       - --audit-log-maxsize=100
       - --audit-log-path=/var/log/kubernetes/kubernetes.audit
       - --audit-log-maxage=7
       - --audit-policy-file=/etc/kubernetes/audit-policy.yml
   ```

   · 策略文件*/etc/kubernetes/audit-policy.yml*。

   策略文件的内容请参见配置介绍中的配置策略文件。

   说明：

   如果*/etc/kubernetes/*目录下没有此文件，可以执行vi audit-policy.yml命令，新建一个文本，将配置策略文件中的内容拷贝并到该目录下即可。

   · 环境变量

   ```
   env:
       - name: aliyun_logs_audit-c12ba20***************9f0b
         value: /var/log/kubernetes/kubernetes.audit
       - name: aliyun_logs_audit-c12ba20***************9f0b_tags
         value: audit=apiserver
       - name: aliyun_logs_audit-c12ba20***************9f0b_product
         value: k8s-audit
       - name: aliyun_logs_audit-c12ba20***************9f0b_jsonfile
         value: "true"
   ```

   · 挂载目录

   ```
   volumeMounts:
       - mountPath: /var/log/kubernetes
         name: k8s-audit
       - mountPath: /etc/kubernetes/audit-policy.yml
         name: audit-policy
         readOnly: true
   volumes:
       - hostPath:
         path: /var/log/kubernetes
         type: DirectoryOrCreate
         name: k8s-audit
       - hostPath:
         path: /etc/kubernetes/audit-policy.yml
         type: FileOrCreate
   ```

```
    name: audit-policy
```

重启apiserver服务，可通过覆盖*/etc/kubernetes/manifests/kube-apiserver.yaml*执行，覆盖前请做好原yaml的备份工作。

若未包含上述内容，请将集群升级到最新版本。升级方式请参考升级集群。

2. 升级新版本日志服务组件Logtail。
   - 若您未安装日志服务组件，可以参考手动安装日志服务组件进行安装。
   - 若您已经安装日志服务组件，但还是显示未开通审计日志，您需要将日志服务组件升级到最新版本，确保Logtail版本在0.16.16及以上且能够运行在master节点。升级方式请参考升级日志服务组件。
3. 更新审计日志解析方式。
   a. 登录 日志服务管理控制台。
   b. 单击左侧导航栏中的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。
   c. 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），单击该日志库上的管理按钮。单击配置名称进入指定采集模式步骤。确认日志模式方式是否为JSON模式，若非JSON模式则修改为JSON模式。

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。

14.3 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#unique_48。
- 您已经通过SSH连接到Master节点，参见#unique_29。
- 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。

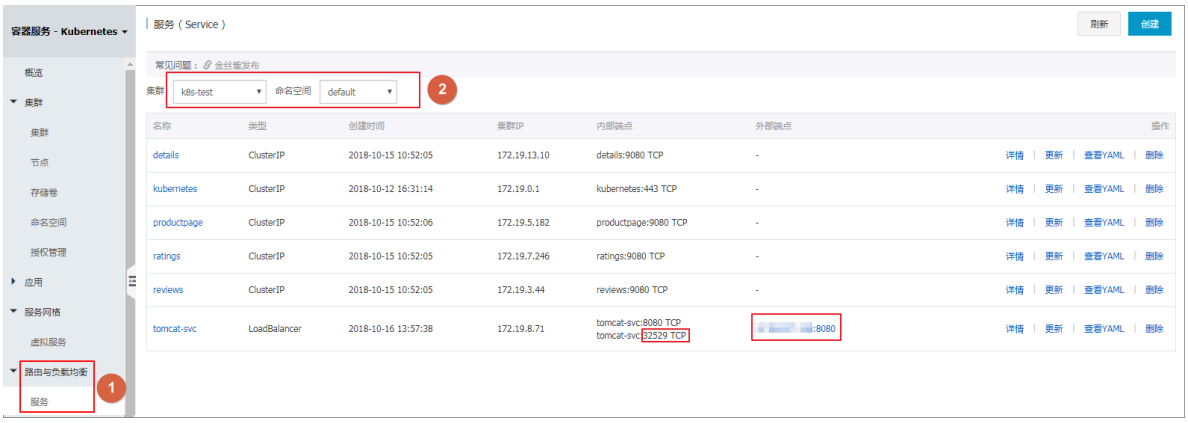
- 适用场景：应用不使用Ingress暴露访问方式，通过LoadBalancer类型的service进行应用访问的暴露。

准备工作

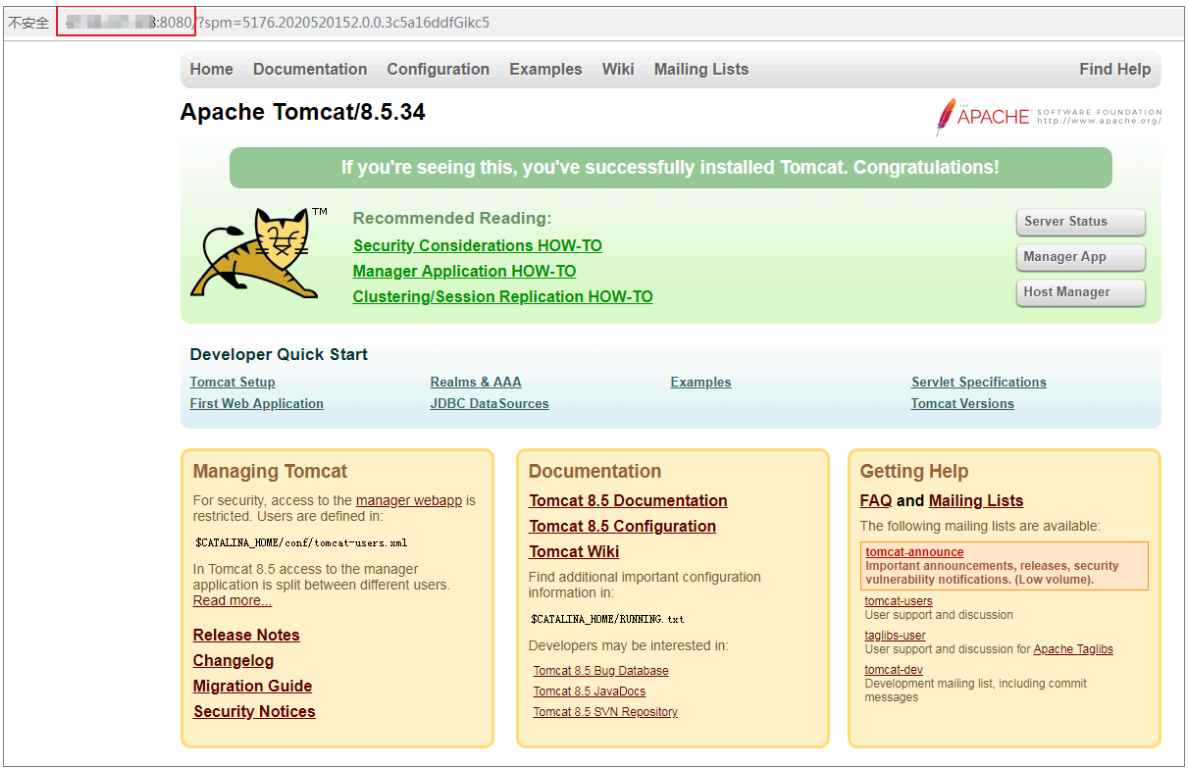
您已在该Kubernetes集群中创建一个Tomcat应用，该应用采用LoadBalancer类型的服务（service）对外提供访问。参见#unique_129。

示例

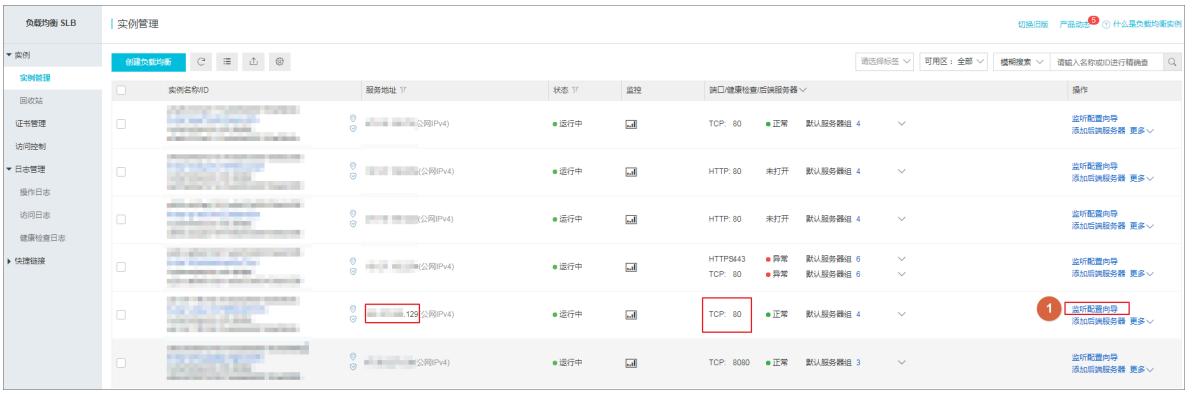
1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中路由与负载均衡 > 服务，选择所需集群和命名空间，查看预先创建的tomcat示例应用。如下图所示创建的tomcat应用名称为tomcat，服务名称为tomcat-svc。其中，服务类型为LoadBalancer，暴露的服务端口为8080。



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。

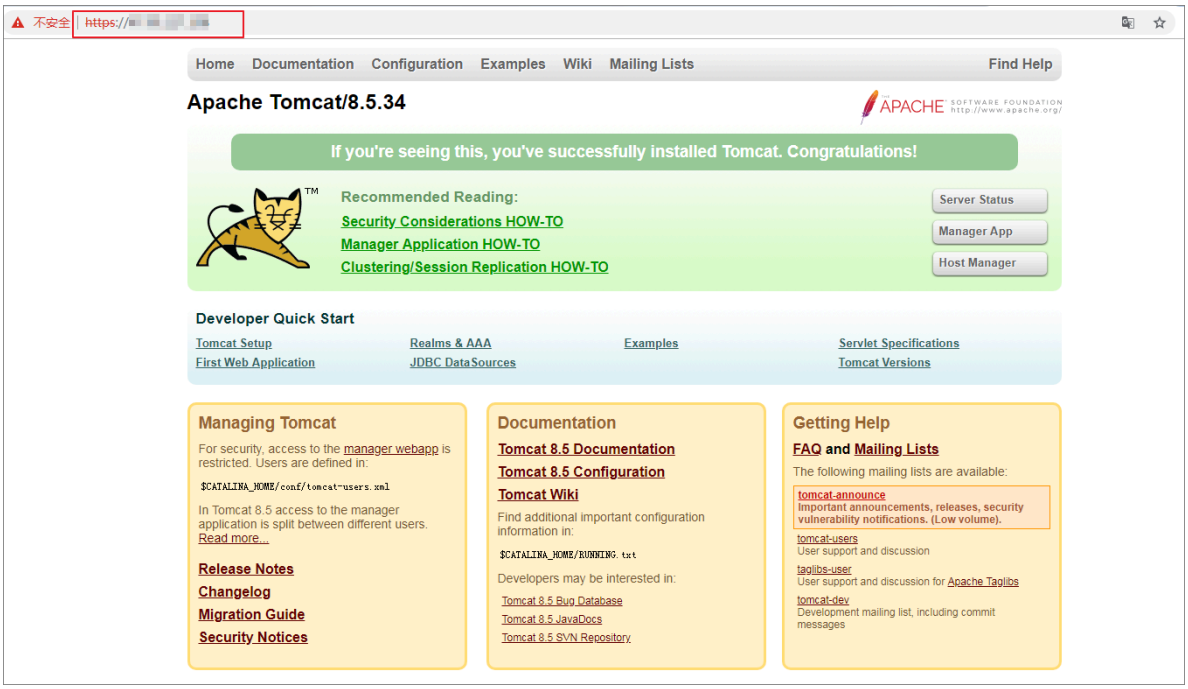
11.配置成功后，单击确定。

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。

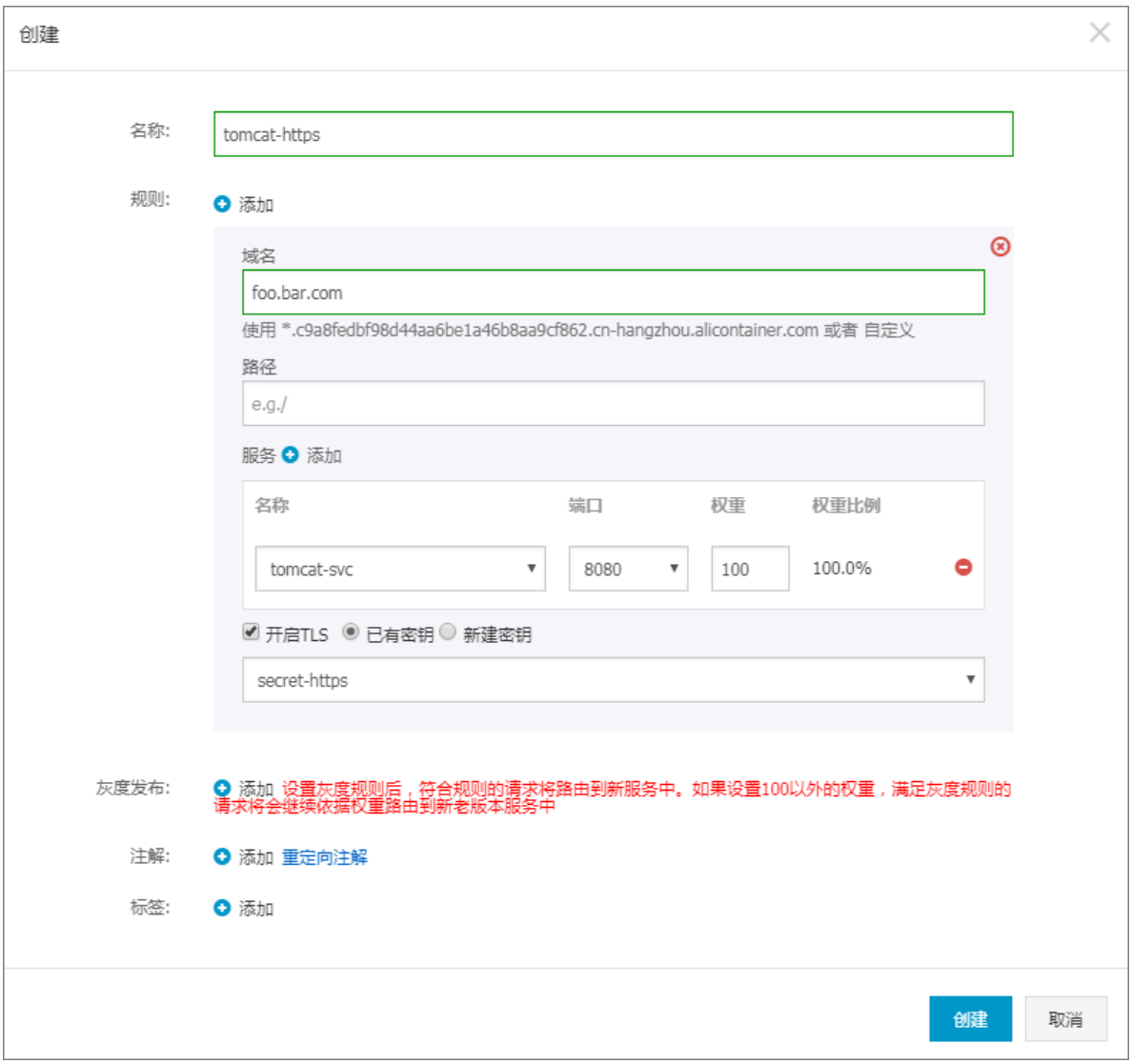
   ```
   kubectl create secret tls secret-https --key tls.key --cert tls.crt
   ```

2. 登录 容器服务管理控制台。
3. 单击左侧导航栏的路由与负载均衡 > 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#unique_160。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。

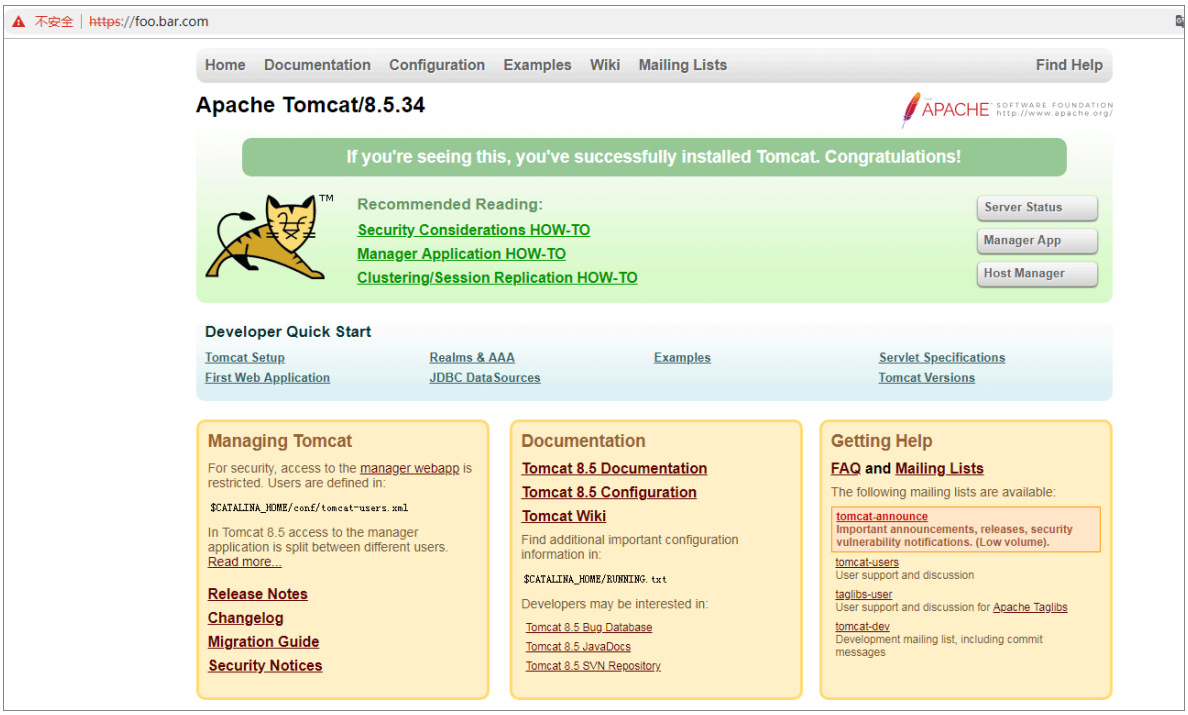
> **说明：**
> 本例中以`foo.bar.com`作为测试域名，您需要在hosts文件中创建一条记录。

```
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。



14.4 更新即将过期证书

本文介绍如何通过控制台更新Kubernetes集群即将过期的证书。

前提条件

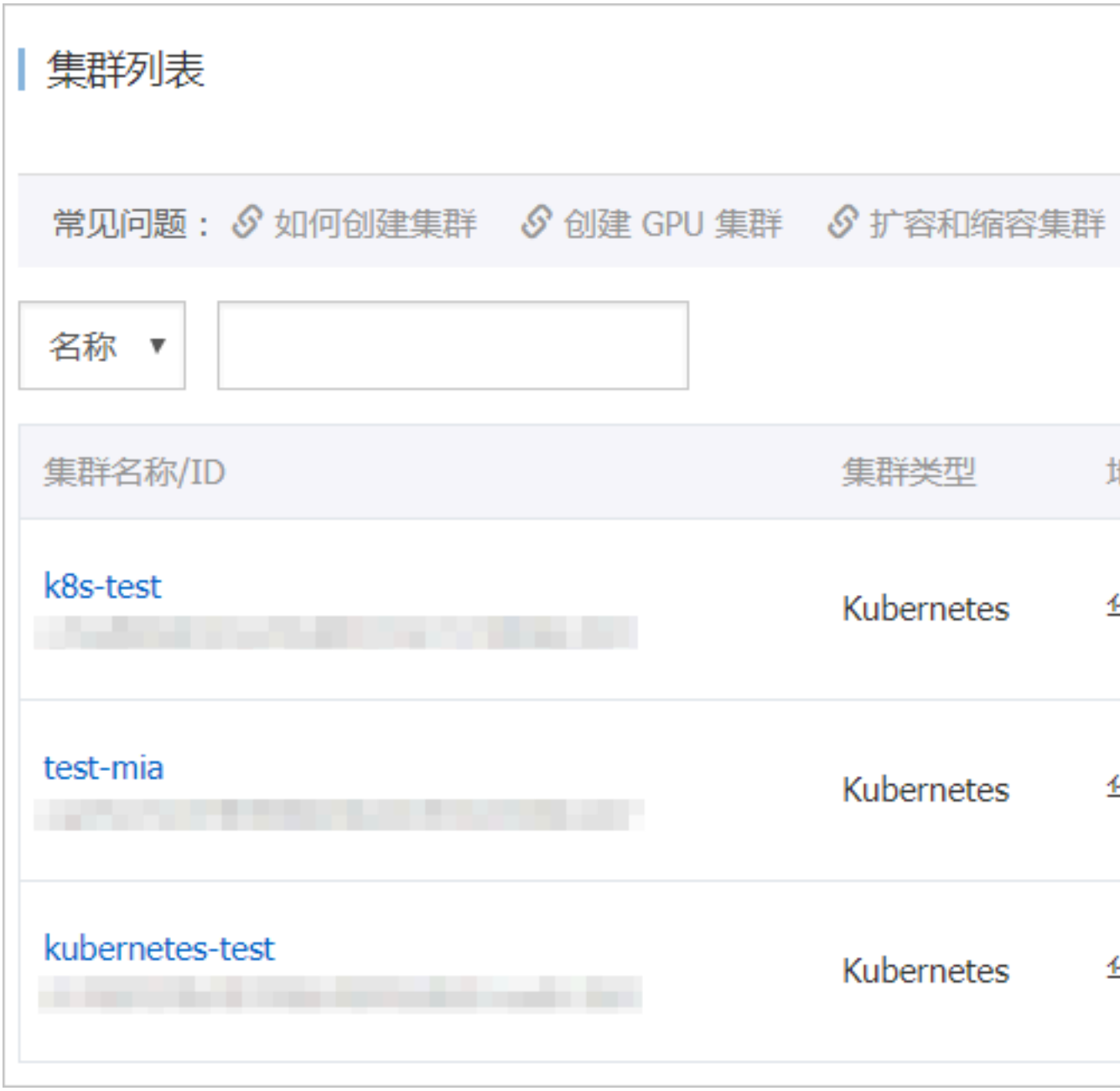
- 您已经成功创建一个Kubernetes集群，参见#unique_48，且集群证书即将过期。
- 您已经了解并做好更新证书前的准备工作，参见集群证书更新说明。

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集群 > 集群，进入集群列表页面。
3. 单击证书即将过期集群右侧的更新证书，进入更新证书页面。

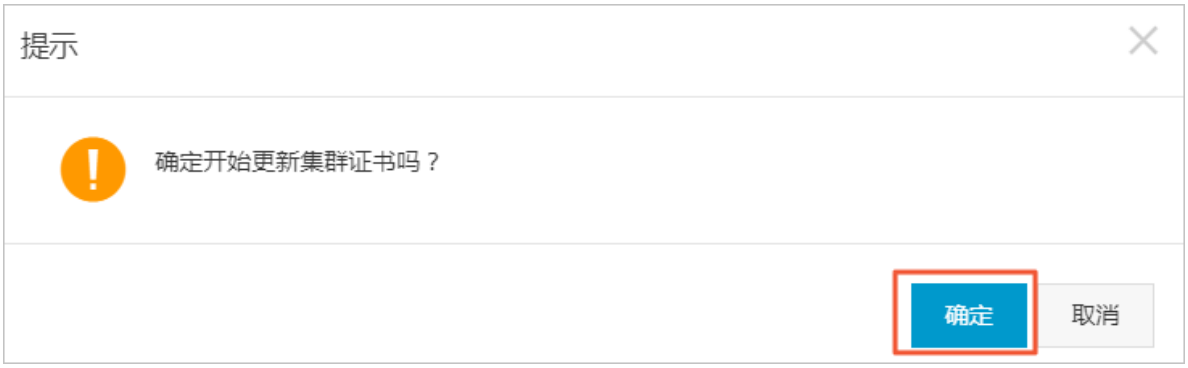
说明:

如果集群证书即将在两个月左右过期，会出现更新证书。



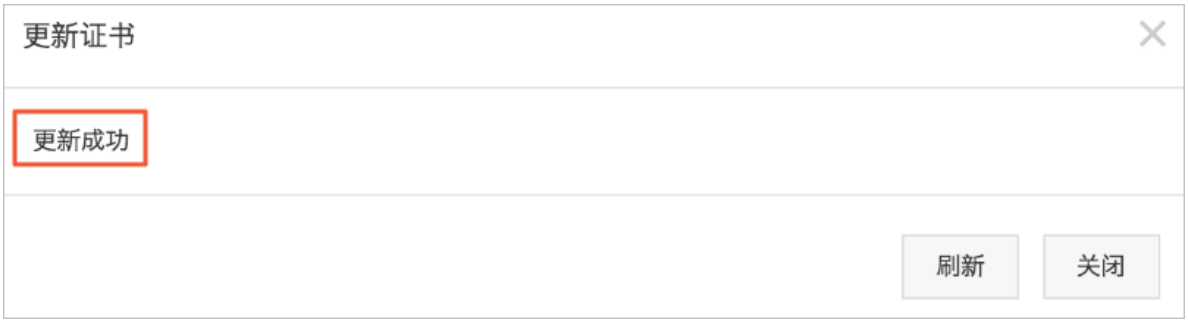
4. **单击更新证书，进入提示页面。**

5. 单击确定。



预期结果

- 更新证书页面，显示更新成功。



- 集群列表页面，目标集群无更新证书提示。

14.5 漏洞修复公告

14.5.1 修复runc漏洞CVE-2019-5736的公告

阿里云容器服务已修复runc漏洞CVE-2019-5736。本文介绍该漏洞的影响范围及解决方法。

背景信息

Docker、containerd或者其他基于runc的容器在运行时存在安全漏洞，攻击者可以通过特定的容器镜像或者exec操作获取到宿主机runc执行时的文件句柄并修改掉runc的二进制文件，从而获取到宿主机的root执行权限。

漏洞CVE-2019-5736的详细信息，请参见CVE-2019-5736。

影响范围

- 对于阿里云容器服务而言，影响范围如下：

  Docker版本 < 18.09.2 的所有Docker Swarm集群和Kubernetes集群（不包含Serverless Kubernetes集群）

- 对于用户自建的Docker/Kubernetes环境而言，影响范围如下：

  Docker版本 < 18.09.2 或者使用 runc版本 <= 1.0-rc6的环境。

解决方法

阿里云容器服务已经修复该漏洞，新创建的1.11或1.12版本的Kubernetes集群中的Docker版本已修复该漏洞。您可以通过以下方法修复已有集群中的漏洞：

- 升级Docker。升级已有集群的Docker到18.09.2或以上版本。该方案会导致容器和业务中断。
- 仅升级runc（针对Docker版本17.06）。为避免升级Docker引擎造成的业务中断，可以按照以下步骤，逐一升级集群节点上的runc二进制。

  1. 执行以下命令定位docker-runc。docker-runc通常位于*/usr/bin/docker-runc*路径下。

     ```
     which docker-runc
     ```

  2. 执行以下命令备份原有的runc：

     ```
     mv /usr/bin/docker-runc /usr/bin/docker-runc.orig.$(date -Iseconds
     )
     ```

  3. 执行以下命令下载修复的runc：

     ```
     curl -o /usr/bin/docker-runc -sSL https://acs-public-mirror.oss-cn
     -hangzhou.aliyuncs.com/runc/docker-runc-17.06-amd64
     ```

  4. 执行以下命令设置docker-runc的可执行权限：

     ```
     chmod +x /usr/bin/docker-runc
     ```

  5. 执行以下命令测试runc是否可以正常工作：

     ```
     docker-runc -v
     # runc version 1.0.0-rc3
     # commit: fc48a25bde6fb041aae0977111ad8141ff396438
     # spec: 1.0.0-rc5
     ```

```
docker run -it --rm ubuntu echo OK
```

6. 如果是Kubernetes集群中的GPU节点，还需要完成以下步骤继续安装下nvidia-runtime。

   a. 执行以下命令定位nvidia-container-runtime。nvidia-container-runtime通常位于*/usr/bin/nvidia-container-runtime*路径下。

   ```
   which nvidia-container-runtime
   ```

   b. 执行以下命令备份原有的nvidia-container-runtime：

   ```
   mv /usr/bin/nvidia-container-runtime /usr/bin/nvidia-container-
   runtime.orig.$(date -Iseconds)
   ```

   c. 执行以下命令下载修复的nvidia-container-runtime：

   ```
   curl -o /usr/bin/nvidia-container-runtime -sSL https://acs
   -public-mirror.oss-cn-hangzhou.aliyuncs.com/runc/nvidia-
   container-runtime-17.06-amd64
   ```

   d. 执行以下命令设置nvidia-container-runtime的可执行权限：

   ```
   chmod +x /usr/bin/nvidia-container-runtime
   ```

   e. 执行以下命令测试nvidia-container-runtime是否可以正常工作：

   ```
   nvidia-container-runtime -v
   #  runc version 1.0.0-rc3
   #  commit: fc48a25bde6fb041aae0977111ad8141ff396438-dirty
   #  spec: 1.0.0-rc5

   docker run -it --rm -e NVIDIA_VISIBLE_DEVICES=all ubuntu nvidia
   -smi -L
   #  GPU 0: Tesla P100-PCIE-16GB (UUID: GPU-122e199c-9aa6-5063-
   0fd2-da009017e6dc)
   ```

   说明：

   本测试运行在GPU P100机型中，不同GPU型号测试方法会有区别。

14.5.2 修复Kubernetes Dashboard漏洞*CVE-2018-18264*的公告

阿里云容器服务Kubernetes 已修复Dashboard漏洞*CVE-2018-18264*，本文介绍该漏洞的影响版本及解决方法。阿里云容器服务Kubernetes内建的Kubernetes Dashboard是托管形态且已进行过安全增强，不受此漏洞影响。

背景信息

Kubernetes社区发现Kubernetes Dashboard安全漏洞*CVE-2018-18264*：使用Kubernetes Dashboard v1.10及以前的版本有跳过用户身份认证，及使用Dashboard登录账号读取集群密钥信息的风险 。

阿里云容器服务Kubernetes内建的Kubernetes Dashboard是托管形态且已进行过安全增强，不受此漏洞影响。

安全漏洞*CVE-2018-18264*的详细信息，请参考：

- https://github.com/kubernetes/dashboard/pull/3289
- https://github.com/kubernetes/dashboard/pull/3400
- https://github.com/kubernetes/dashboard/releases/tag/v1.10.1

影响版本

如果您的Kubernetes集群中独立部署了Kubernetes Dashboard v1.10及之前版本（v1.7.0-v1.10.0），同时支持登录功能且使用了自定义证书。

解决方法

- 如果您不需要独立部署的Dashboard，请执行以下命令，将Kubernetes Dashboard从集群中删除。

```
kubectl --namespace kube-system delete deployment kubernetes-dashboard
```

- 如果您需要独立部署的Dashboard，请将Dashboard升级到v1.10.1版本，请参考https://github.com/kubernetes/dashboard/releases/tag/v1.10.1。
- 如果您使用阿里云容器服务Kubernetes版本托管的Dashboard，由于阿里云容器服务Kubernetes版本对此进行过安全增强，不受此漏洞影响，您仍可以直接在容器服务管理控制台上使用Dashboard。

14.5.3 修复Kubernetes漏洞*CVE-2018-1002105*公告

阿里云容器服务Kubernetes 已修复漏洞*CVE-2018-1002105*，本文介绍该漏洞的影响及解决方法。

背景信息

Kubernetes社区发现安全漏洞：*CVE-2018-1002105*。Kubernetes用户可通过伪造请求，在已建立的API Server连接上提升权限访问后端服务，目前阿里云容器服务Kubernetes 已第一时间修复此漏洞，请登录容器服务管理控制台升级您的Kubernetes版本。

漏洞*CVE-2018-1002105*详细信息，请参考：https://github.com/kubernetes/kubernetes/issues/71411。

影响版本

- Kubernetes v1.0.x-1.9.x

- Kubernetes v1.10.0-1.10.10 (fixed in v1.10.11)
- Kubernetes v1.11.0-1.11.4 (fixed in v1.11.5)
- Kubernetes v1.12.0-1.12.2 (fixed in v1.12.3)

影响配置

- 容器服务Kubernetes集群启用了扩展API Server，并且kube-apiserver与扩展API Server的网络直接连通。
- 容器服务Kubernetes集群开放了 pod exec/attach/portforward 接口，用户可以利用该漏洞获得所有的kubelet API访问权限。

阿里云容器服务Kubernetes集群配置

- 阿里云容器服务Kubernetes集群的API Server默认开启了RBAC，通过主账号授权管理默认禁止了匿名用户访问。同时Kubelet 启动参数为`anonymous-auth=false`，提供了安全访问控制，防止外部入侵。
- 对于使用子账号的多租户容器服务Kubernetes集群用户，子账号可能通过pod exec/attach/portforward 接口越权访问。如果集群只有管理员用户，则无需过度担心。
- 子账号在不经过主账号自定义授权的情况下默认不具有聚合API资源的访问权限。

解决方法

请登录容器服务管理控制台，升级您的集群，升级的注意事项及具体的操作步骤，请参考#unique_284。

- 如果您的集群版本为1.11.2请升级到1.11.5。
- 如果您的集群版本为1.10.4请升级到1.10.11或1.11.5版本。
- 如果您的集群版本为1.9及以下版本，请升级到1.10.11或1.11.5版本。在1.9版本升级1.10或1.11版本时，如集群使用了云盘数据卷，需在控制台先升级flexvolume插件。

> 说明:
> 在容器服务管理控制台上，选择目标集群，单击导航栏更多 > 系统组件升级，在系统组件升级页面，选择flexvolume组件，单击升级。

由于Serverless Kubernetes在此漏洞发生前已进行加固，用户不受影响。

14.5.4 修复Kubectl cp漏洞*CVE-2019-11246*的公告

Kubernetes 最近公布了另一个kubectl cp 相关漏洞 CVE-2019-11246 ，此漏洞可能允许攻击者利用kubectl cp命令，采用路径遍历（Path Traversal）的方式将容器tar包中的恶意文件写入所在主机上的任何路径，该过程仅受本地用户的系统权限限制。 详细信息请参见Kubernetes披露。

背景信息

该漏洞与不久前的CVE-2019-1002101漏洞影响相似，由于之前的相关漏洞修复。

kubectl cp命令用于用户容器和主机之间的文件拷贝，当从容器中拷贝文件时，Kubernetes 会首先在容器中执行tar命令创建相应的归档文件，然后发送给客户端，kubectl会在用户主机上进行相应解压操作。

如果容器tar包中包含恶意文件，当攻击者具有kubectl cp命令的执行权限时，可以利用路径遍历(Path Traversal。

官方修复pr请参见 https://github.com/kubernetes/kubernetes/pull/76788。

影响范围

- kubectl v1.11.x 及以前版本
- kubectl v1.12.1-v1.12.8（fixed in v1.12.9）
- kubectl v1.13.1-v1.13.5（fixed in v1.13.6）
- kubectl v1.14.1（fixed in v1.14.2）

说明:

您可以通过运行`kubectl version --client`命令，来查看 kubectl 版本。

解决方案

通过升级kubectl 的版本来修复该漏洞。请参考安装 Kubectl，升级kubectl客户端，安装成功后请再次确认客户端版本号。

- 如果您的 kubectl 版本为1.12.x 请升级到1.12.9。
- 如果您的 kubectl 版本为1.13.x 请升级到1.13.6。
- 如果您的 kubectl 版本为1.14.x，请升级到1.14.2。
- 如果您的 kubectl 版本为1.11及以下版本，请升级到1.12.9、1.13.6或1.14.2版本。

14.5.5 修复Kubectl cp漏洞*CVE-2019-11249*的公告

即前不久刚刚暴出的kubectl cp相关漏洞CVE-2019-11246，近日Kubernetes官方又公布了一个kubectl cp相关漏洞 CVE-2019-11249 ，此漏洞可能允许恶意攻击者利用目录遍历（Directory Traversal）的方式将容器tar包中的恶意文件写入或替换至所在主机上目标路径之外的其他位置，该过程仅受本地用户的系统权限限制。

背景信息

kubectl cp命令用于用户容器和主机之间的文件拷贝，当从容器中拷贝文件时，Kubernetes会首先在容器中执行tar命令创建相应的归档文件，然后发送给客户端，kubectl会在用户主机上进行相应解压操作。

如果容器tar包中包含恶意文件，当攻击者具有kubectl cp命令的执行权限时，可以利用目录遍历Directory Traversal。

在本次修复中，cp命令在执行untar过程中对所有子文件的目标路径执行了更严格的校验，禁止所有在cp目标路径外的解压后拷贝动作，以防止untar过程中的恶意攻击。

有关详细信息，请参见Kubernetes披露。

官方修复pr请参见 https://github.com/kubernetes/kubernetes/pull/80436/

影响范围

您可以通过运行`kubectl version --client`命令，来查看 kubectl 版本。

在下列范围内的版本均受次漏洞影响：

- kubectl 1.0.x-1.12.x
- kubectl 1.13.0-1.13.8（fixed in v1.13.9）
- kubectl 1.14.0-1.14.4（fixed in v1.14.5）
- kubectl 1.15.0-1.15.1（fixed in v1.15.2）

解决方案

通过升级kubectl 的版本来修复该漏洞。请参见安装Kubectl，升级kubectl客户端，安装成功后请再次确认客户端版本号。

- 如果您的 kubectl 版本为1.13.x 请升级到1.13.9。
- 如果您的 kubectl 版本为1.14.x，请升级到1.14.5。
- 如果您的 kubectl 版本为1.15.x，请升级到1.15.2。
- 如果您的 kubectl 版本为1.12.x及以下版本，请升级到1.13.9、1.14.5或1.15.2版本。

14.6 集群证书更新说明

本文档主要为您介绍证书更新相关自动化操作说明。

当集群证书过期前两个月左右，在容器服务控制台集群列表页面会出现更新证书的红色按钮提示您更新证书，同时您会收到相应的站内信或短信通知。

您可以通过单击该红色按钮进行集群证书的自动更新，整个更新时长依据集群节点个数而定，一般5~10分钟即可完成更新。更新成功后相应证书有效期会延长五年。

注意事项

- 在证书更新过程中集群 Kubernetes 原生组件会有短暂的重启过程，建议您在非业务高峰期执行更新操作。
- 更新操作不会影响已有的线上服务。

备份

节点类型	备份内容
Master	· /etc/kubernetes/ · /var/lib/kubelet/pki · /etc/systemd/system/kubelet.service.d/10-kubeadm.conf · /etc/kubeadm/ · 需要备份的业务数据 说明: 其中，*/var/lib/kubelet/pki*和需要备份的业务数据的路径下，如果不存在数据，则无需备份。
Worker	· /etc/kubernetes/ · /etc/systemd/system/kubelet.service.d/10-kubeadm.conf · /var/lib/kubelet/pki/* · 需要备份的业务数据 说明: 其中，*/var/lib/kubelet/pki/**和需要备份的业务数据路径下，如果不存在数据，则无需备份。

证书更新

表 14-1: Master节点

证书或 conf 名称	路径
· apiserver.crt · apiserver.key	/etc/kubernetes/pki
· apiserver-kubelet-client.crt · apiserver-kubelet-client.key	/etc/kubernetes/pki

证书或 conf 名称	路径
· front-proxy-client.crt · front-proxy-client.key	/etc/kubernetes/pki
· dashboard.crt · dashboard.key	/etc/kubernetes/pki/dashboard
· kubelet.crt · kubelet.key 说明: 如果不存在*kubelet.key*，则无需更新。	/var/lib/kubelet/pki 说明: 如果不存在数据，则无需更新。
admin.conf	/etc/kubernetes
kube.conf	/etc/kubernetes
controller-manager.conf	/etc/kubernetes
scheduler.conf	/etc/kubernetes
kubelet.conf	/etc/kubernetes
config	~/.kube/
· kubelet-client-current.pem 或 kubelet-client.crt\ · kubelet-client.key 说明: 如果不存在*kubelet-client.key*，则无需更新。	/var/lib/kubelet/pki 说明: 如果不存在数据，则无需更新。

表 14-2: Worker 节点

证书或 conf 名称	路径
· kubelet.crt · kubelet.key 说明: 如果不存在*kubelet.key*，则无需更新。	/var/lib/kubelet/pki 说明: 如果不存在数据，则无需更新。

证书或 conf 名称	路径
· kubelet-client-current.pem 或 kubelet-client.crt · kubelet-client.key 说明: 如果不存在*kubelet-client.key*，则无需更新。	/var/lib/kubelet/pki 说明: 如果不存在数据，则无需更新。
kubelet.conf	/etc/kubernetes

14.7 安全组常见问题

本文主要为您介绍安全组常见问题。

- 容器之间网络不通
- 如何给Kubernetes集群指定安全组

容器之间网络不通

下面为您介绍容器服务Kubernetes集群由于安全组导致网络不通的问题原因及解决方法。

- 入方向授权对象为Pod 网络 CIDR，且协议类型为全部的规则被删除。

  1. 登录容器服务管理控制台。
  2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 集群，进入集群列表页面。
  3. 单击目标集群名称，查看集群的详细信息。
  4. 单击集群资源区域的虚拟专有网络 VPC，跳转到专有网络管理控制台的专有网络详情页面。
  5. 单击网络资源区域的安全组右侧的数字，跳转到云服务器管理控制台的安全组列表页面。
  6. 单击目标安全组操作列的配置规则，查看安全组的详细信息。
  7. 在入方向页面，单击右上角添加安全组规则。
  8. 填写协议类型及授权对象。

  说明:

  - 协议类型请选择全部。

- 授权对象填写为Pod 网络 CIDR。
- Pod 网络 CIDR可在容器服务管理控制台集群详细信息页面，集群信息区域查看。

- 授权对象的设置，可参见#unique_288。

入方向授权对象为Pod 网络 CIDR，且协议类型为全部的规则已添加。

· 新增ECS实例的安全组与集群所在的安全组不同。

1. 登录容器服务管理控制台。
2. 单击左侧导航栏集群 > 集群，在集群列表页面，选择目标集群。
3. 单击目标集群操作列的集群名称。
4. 单击集群资源区域的虚拟专有网络VPC，跳转到专有网络管理控制台查看网络资源信息。
5. 在专有网络详情页面，单击网络资源区域安全组右侧的数字，跳转到云服务器ECS管理控制台的安全组列表页面查看安全组的详细信息。
6. 在安全组列表页面，查看安全组的名称。
7. 在云服务器ECS管理控制台，单击左侧导航栏实例。
8. 在实例列表页面，单击目标实例的操作列的更多 > 网络和安全组 > 加入安全组，进入ECS实例加入安全组页面。
9. 单击安全组复选框右侧的下拉箭头，输入步骤7查询到的集群安全组名称。
10. 单击确定。

执行结果：

1. 单击云服务器管理控制台左侧导航栏的实例，在实例列表页面，单击目标实例的名称。
2. 单击左侧导航栏的本实例安全组。
3. 在安全组列表区域，可看到ECS实例已加入集群所在的安全组。

如何给Kubernetes集群指定安全组

当前暂不支持给Kubernetes集群指定安全组。但是创建Kubernetes集群时，容器服务ACK会自动创建一个默认安全组，您可以通过修改默认安全组的规则，达到指定安全组的效果。

15 发布管理

15.1 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。

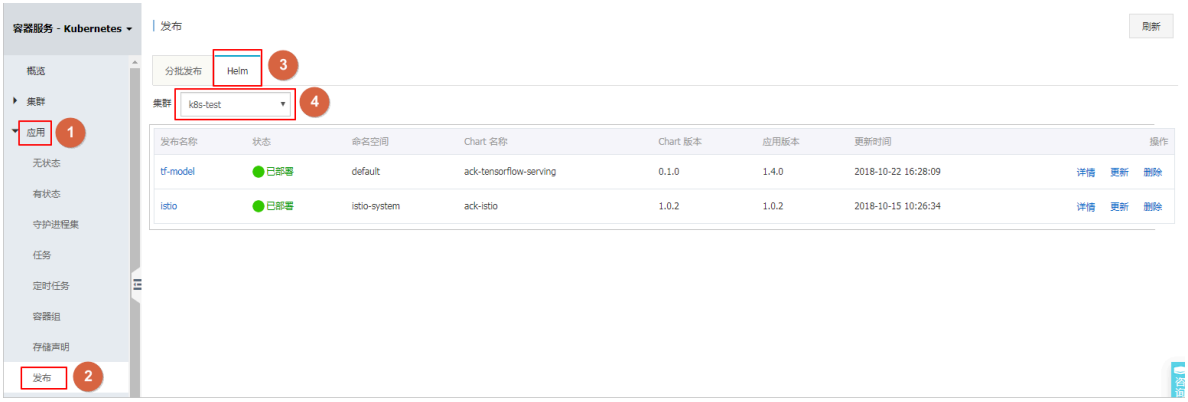
前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已经使用应用目录或服务目录功能，安装了 Helm 应用，参见#unique_291。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。

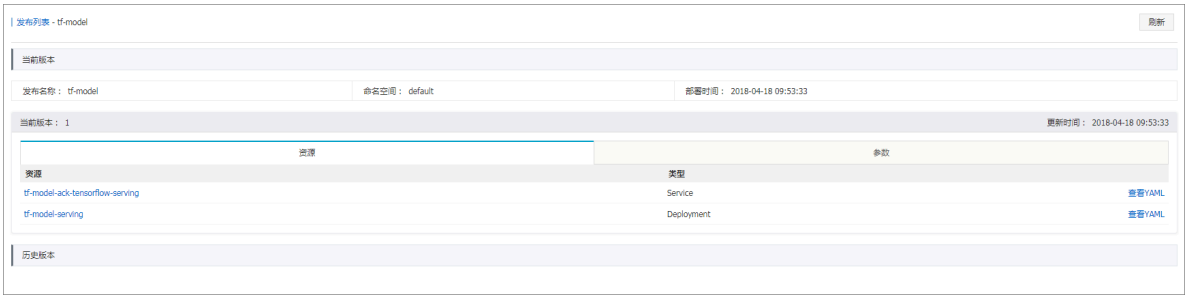


3. 以 tf-model 为例，您可查看发布的详情信息，单击右侧的详情，进入该发布的详情页面。

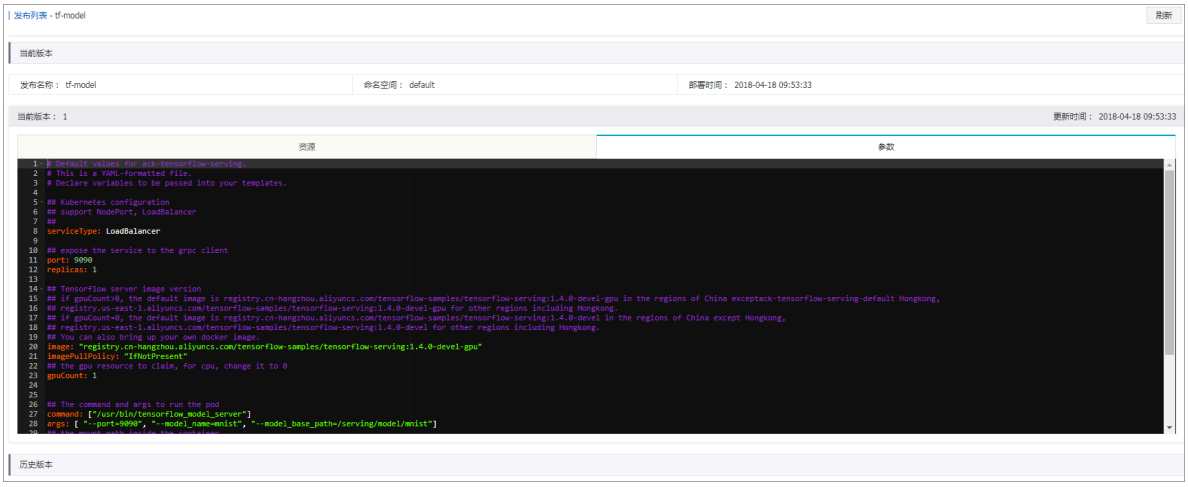
   你可以查看该发布的当前版本和历史版本等信息，当前版本为1，无历史版本。您还可查看 tf-model 的资源信息，如资源名称，资源类型，以及查看 YMAL 信息。

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。



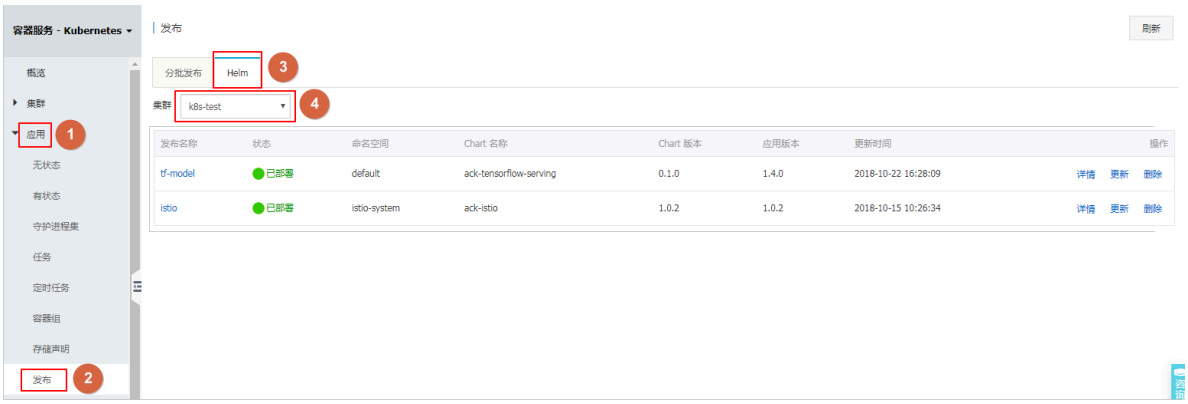
4. 单击参数，你可查看该 Helm 包安装的参数配置。



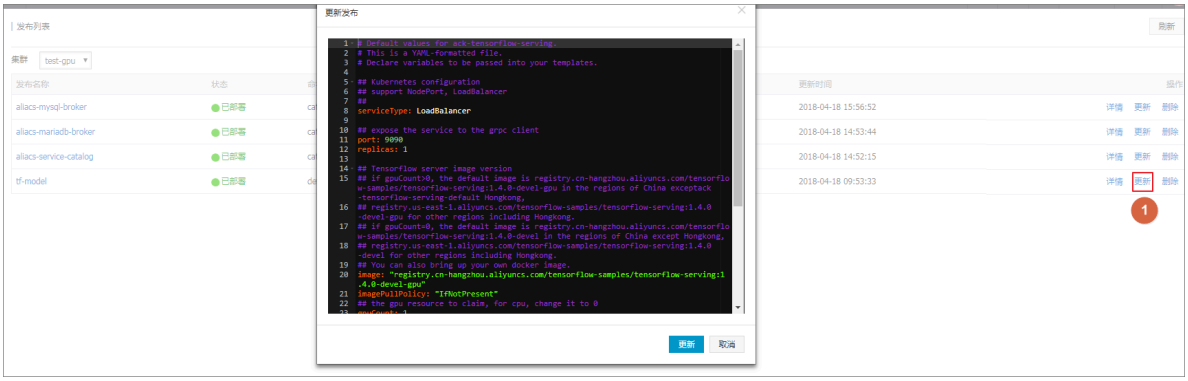
更新发布的版本

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。

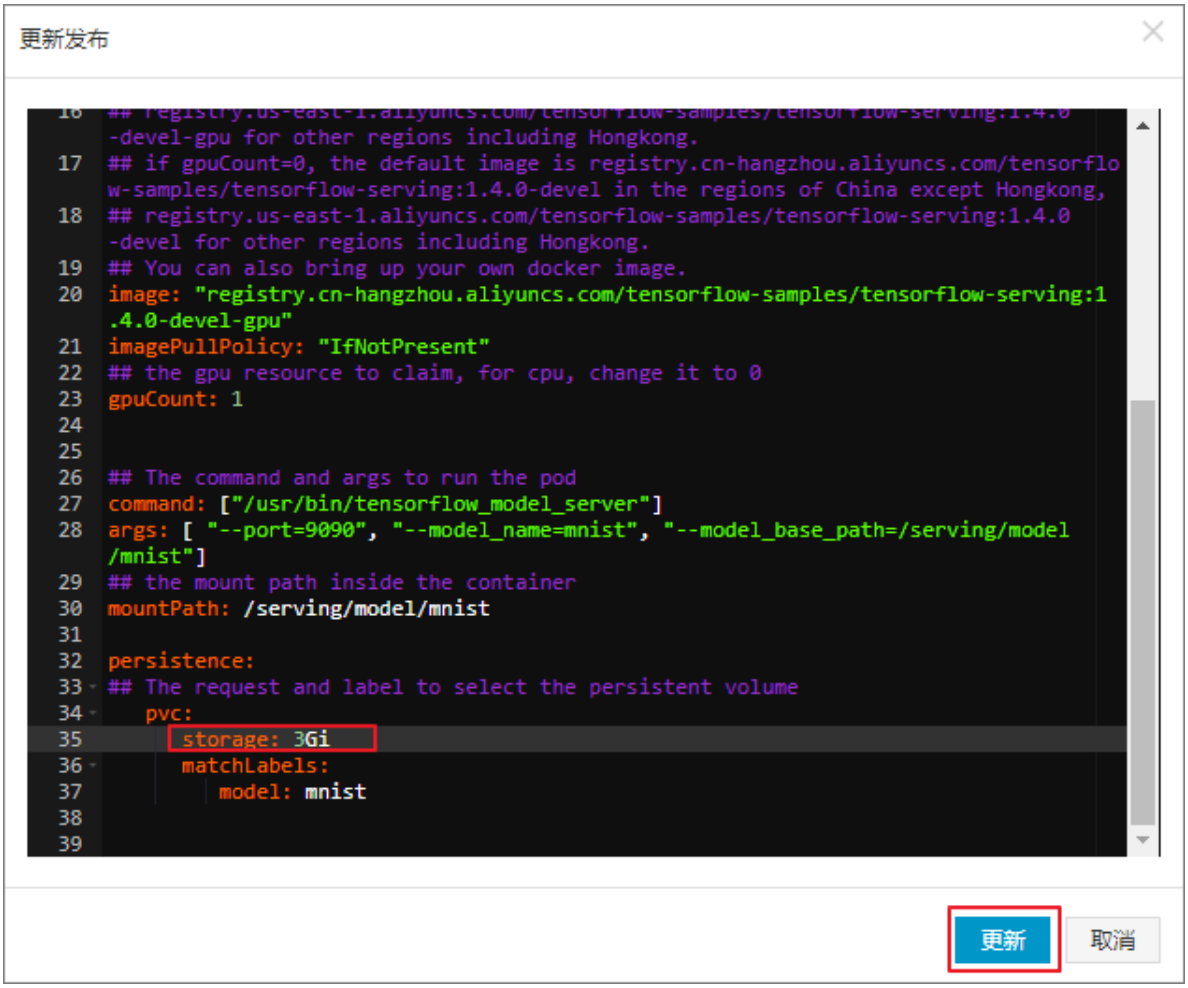
   您可看到该集群下通过 Helm 包管理工具发布的应用及服务等。



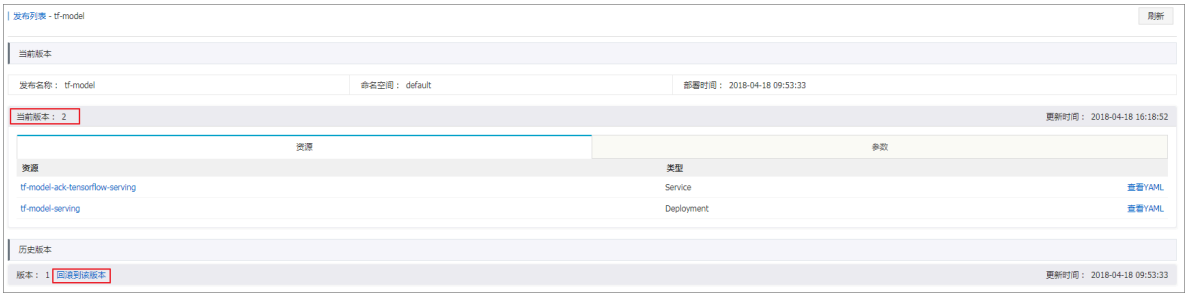
3. 以 tf-model 为例，您可更新该发布，单击右侧的更新，弹出更新发布对话框。



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。

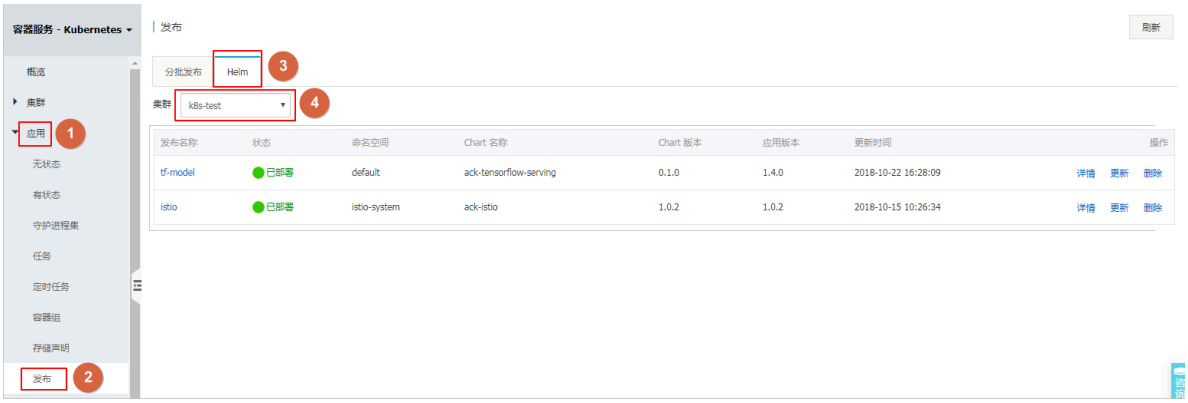


删除发布

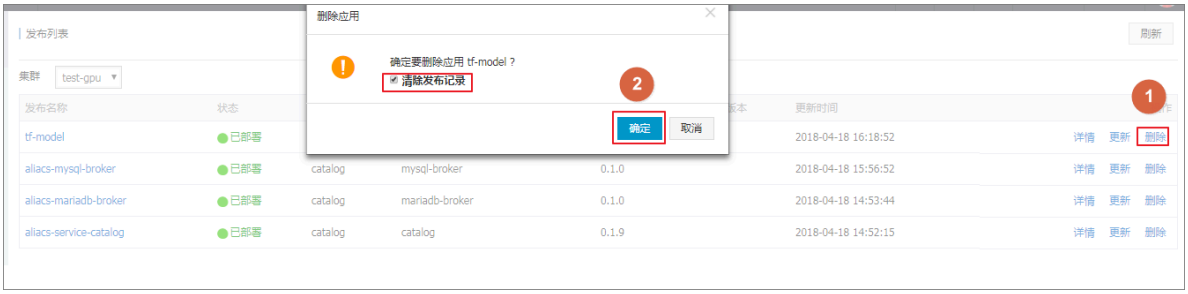
1. 登录容器服务管理控制台。

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。

15.2 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

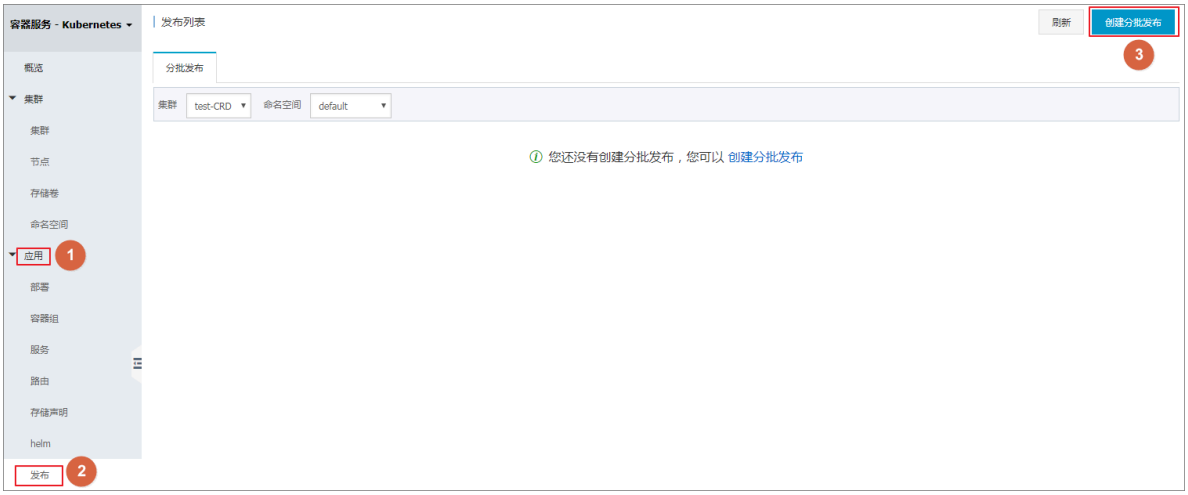
> **说明:**
> Kubernetes最新集群已经默认安装alicloud-application-controller；对于旧版本的集群，目前仅支持1.9.3及以上的版本，您可通过控制台上的提示链接进行升级。

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。

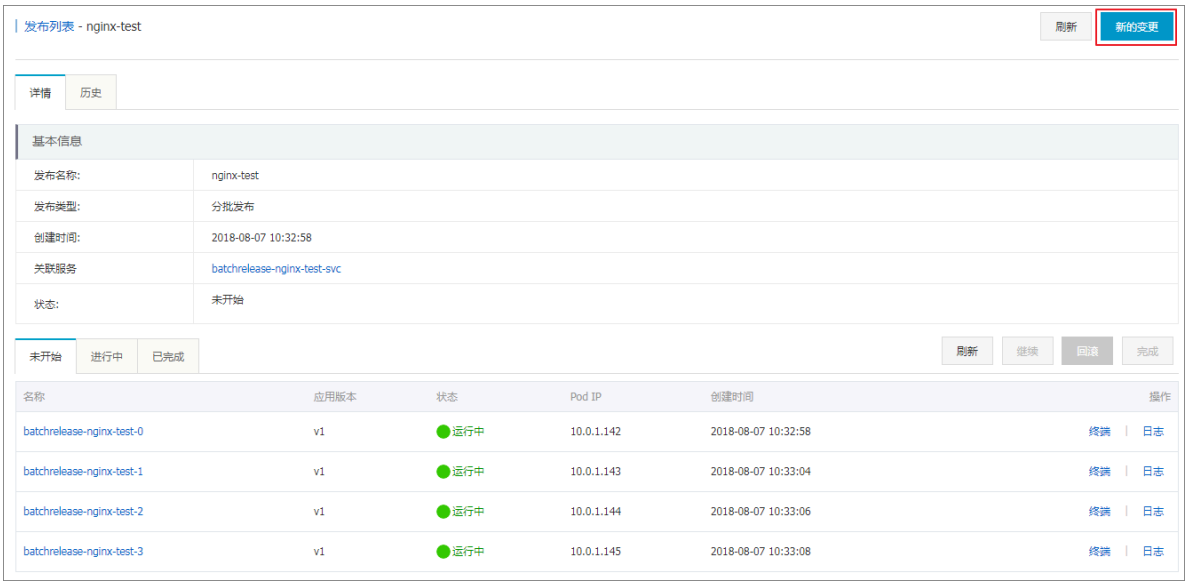


3. **首先配置分批发布信息，包括应用名称、部署集群、命名空间和发布选项。最后单击下一步。**

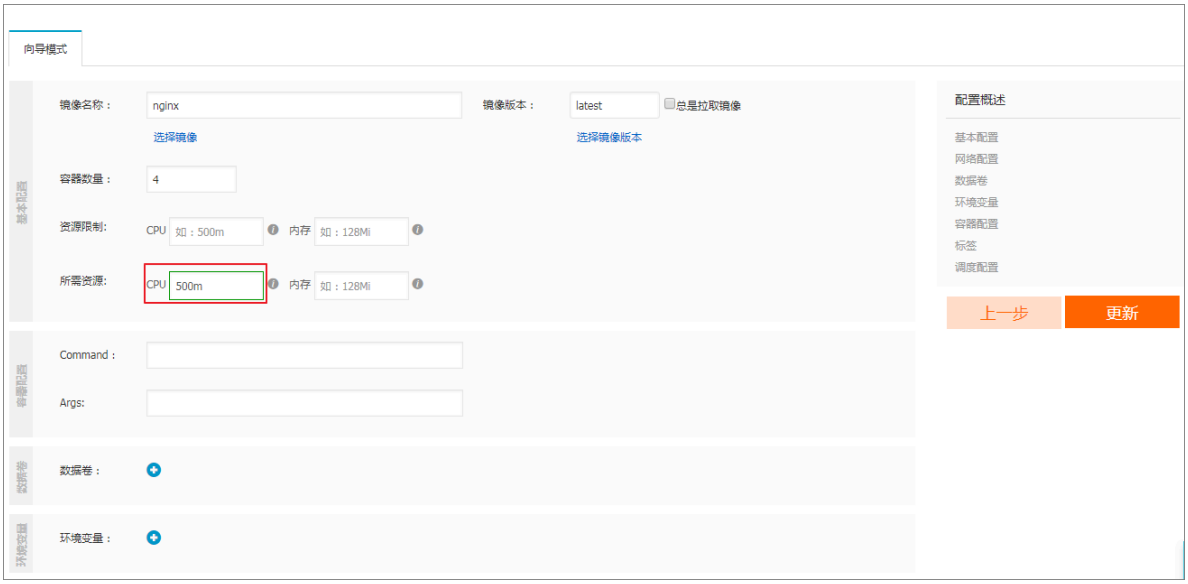
4. **在分批发布配置页面，配置后端的Pod和Service，最后单击更新，创建应用。**

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。

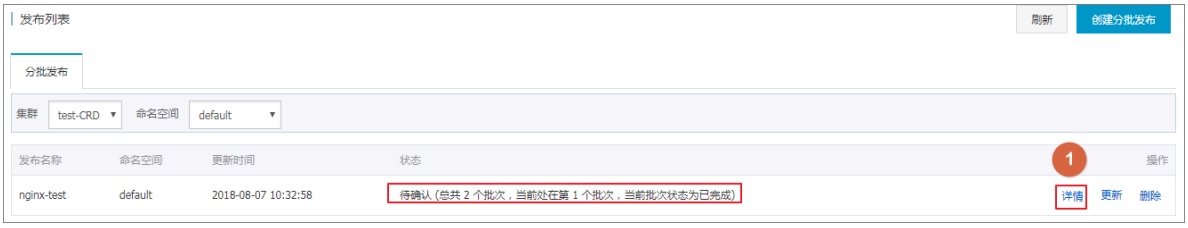
6. 在应用详情页面中，您可查看更多信息，单击页面右上角的新的变更，进行一次分批发布的变更。



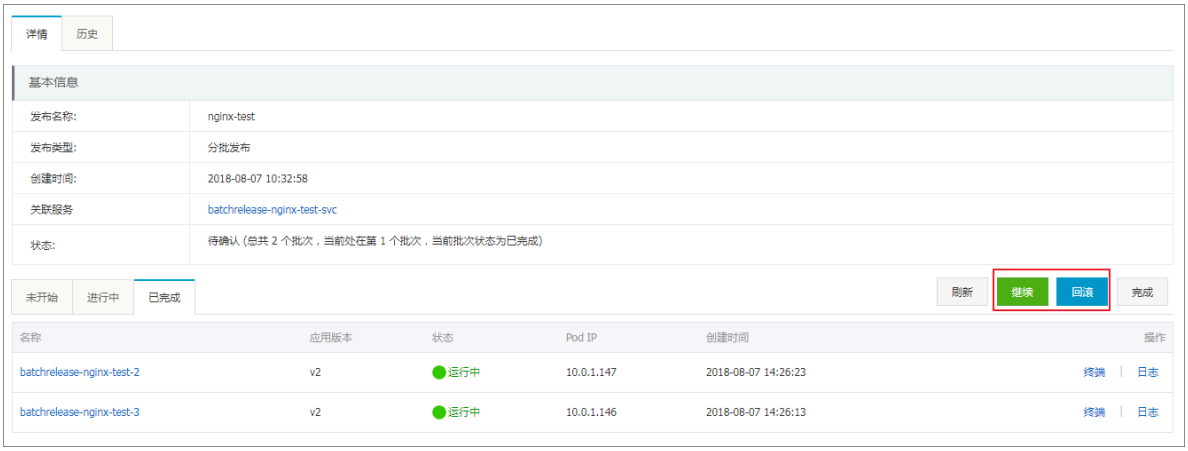
7. 在变更配置页面中，进行新版本应用的变更配置，完成后单击更新。



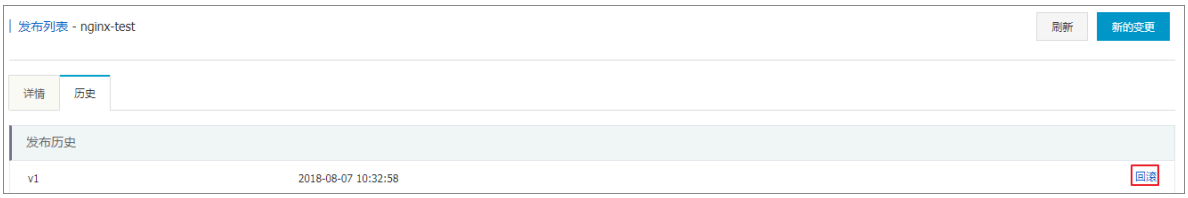
8. 默认返回发布列表页面，您可查看该应用的分批发布状态，完成第一批次部署后，单击详情。



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。



后续步骤

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。

16 命名空间管理

16.1 创建命名空间

前提条件

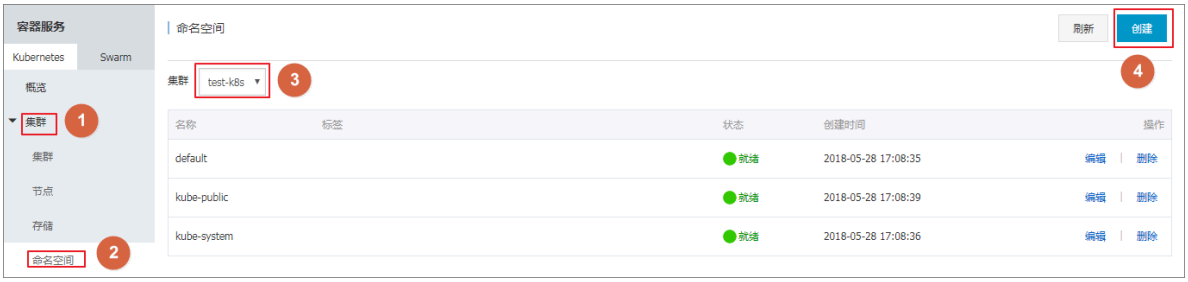
您已成功创建一个 Kubernetes 集群，参见#unique_48。

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。

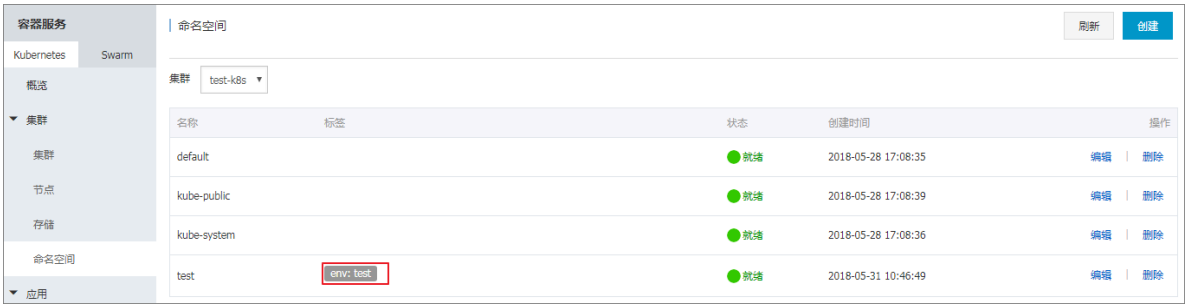


4. 在弹出的对话框中，配置命名空间。

- 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。



16.2 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#unique_48。
- 您已成功创建一个示例的命名空间test，参见#unique_296。
- 您已成功连接到集群的Master节点地址，参见#unique_37。

背景信息

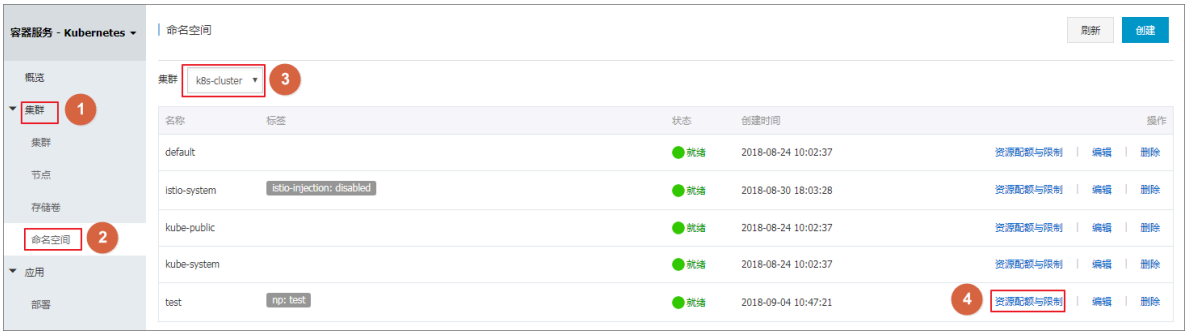
在默认的情况下，运行中的 Pod 可以无限制的使用 Node 上的 CPU 和内存，这意味着任意一个 Pod 都可以无节制地使用集群的计算资源，某个命名空间的 Pod 可能会耗尽集群的资源。

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。



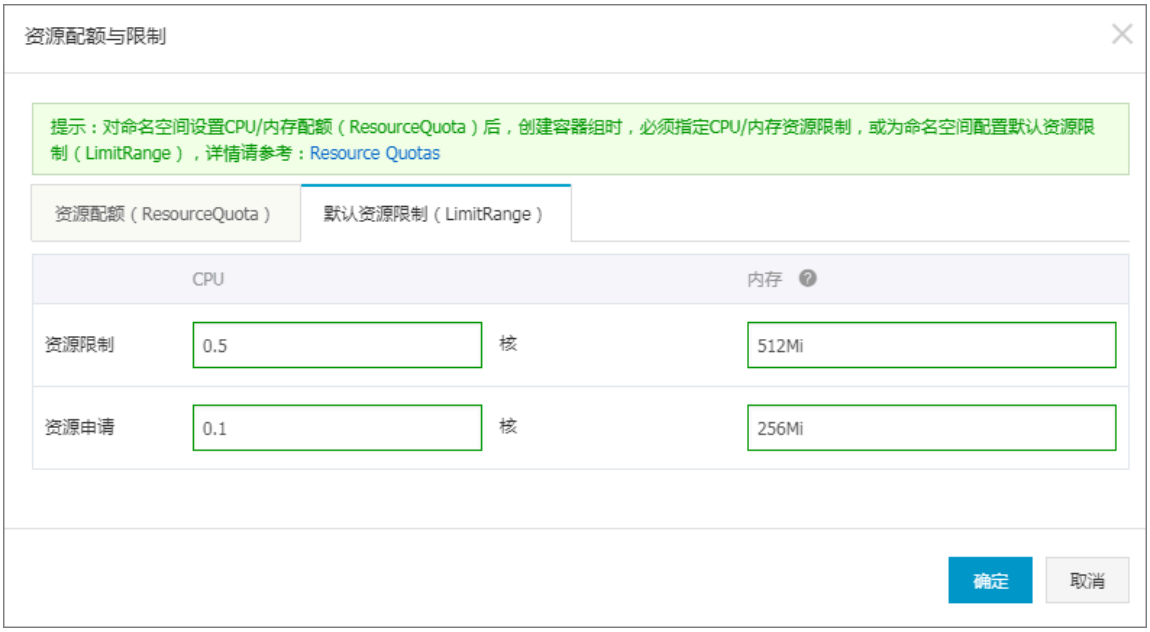
3. 在资源配额与限制对话框中，您可快速设置资源配额和默认资源限制。

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
```

16.3 编辑命名空间

前提条件

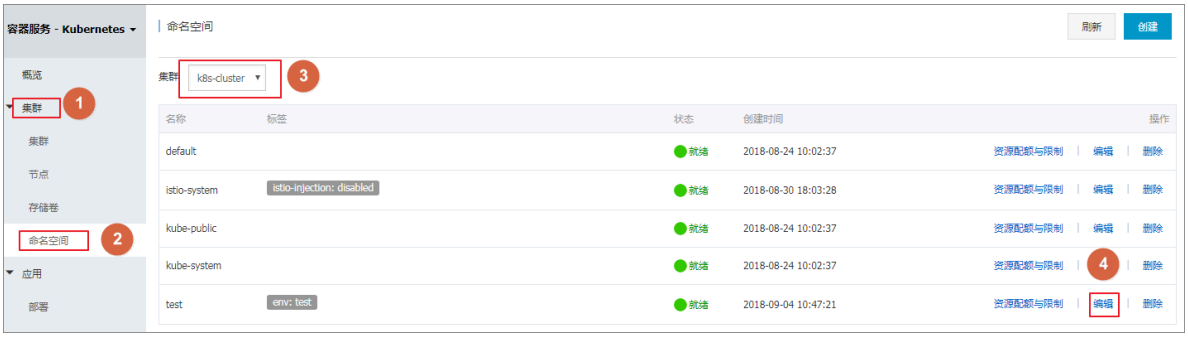
- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已成功创建一个示例的命名空间test，参见#unique_296。

背景信息

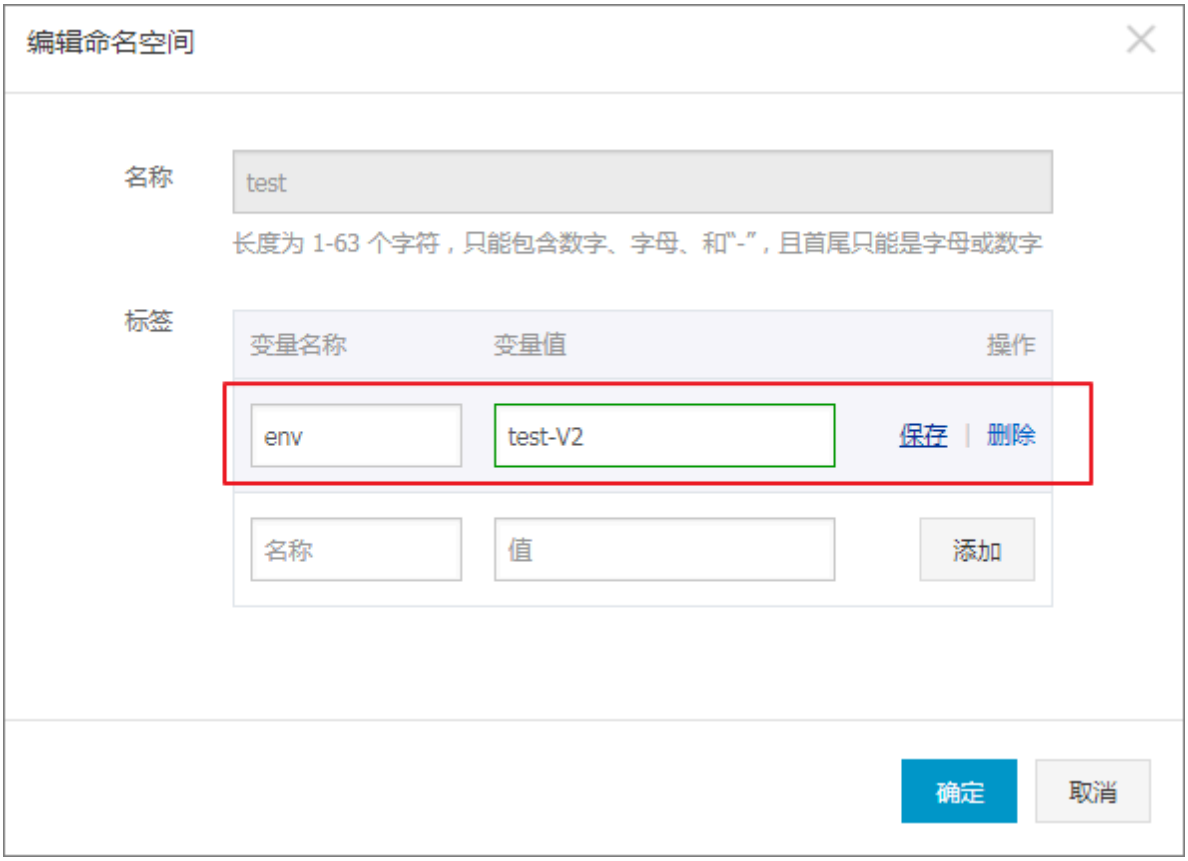
您可对命名空间进行编辑，对命名空间的标签进行增、删、改等操作。

操作步骤

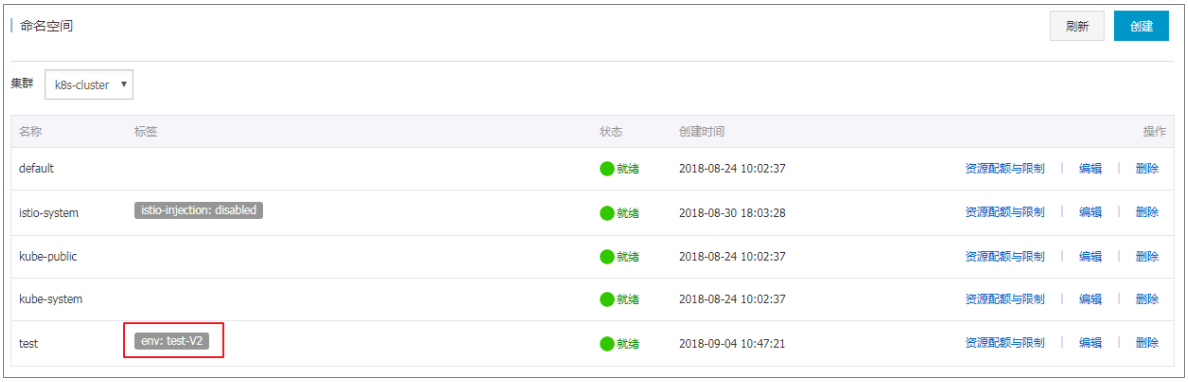
1. 登录 容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏的集群 > 命名空间，进入命名空间列表页面。
3. 选择所需的集群，选择所需的命名空间，单击右侧的编辑。



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。



16.4 删除命名空间

您可删除不再需要的命名空间。

前提条件

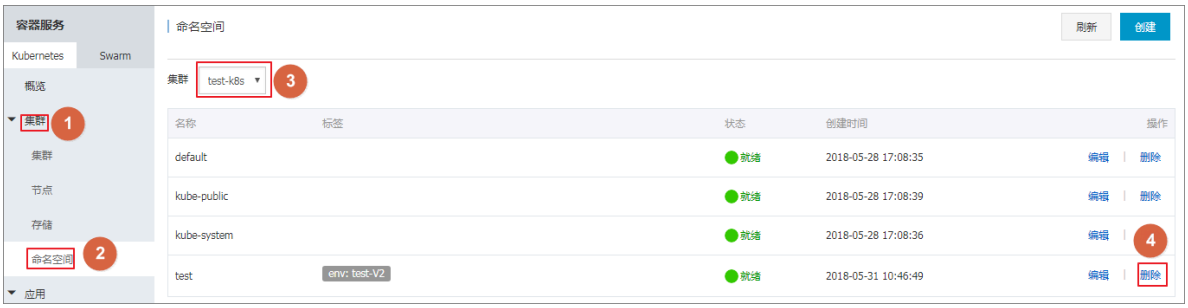
- 您已成功创建一个 Kubernetes 集群，参见#unique_48。
- 您已成功创建一个示例的命名空间test，参见#unique_296。

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。

17 服务网格

17.1 Istio管理

17.1.1 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。

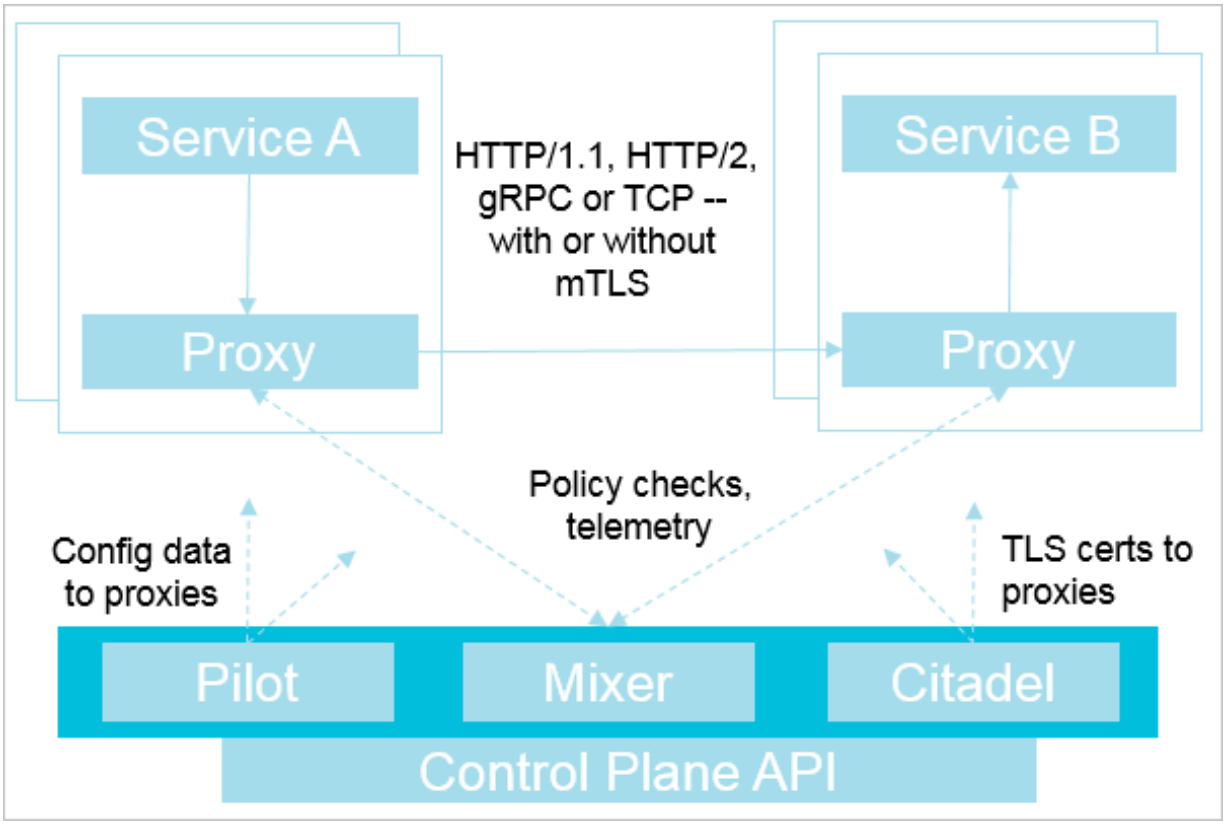
Istio提供如下功能：

- 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。

17.1.2 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#unique_42。
- 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#unique_303。

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

1. 部署istio 。
   a) 登录容器服务管理控制台。
   b) Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > Istio管理，进入部署Istio页面。
   c) 选择所需的集群并设置相应的配置参数。
   d) 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。

配置	说明
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。
链路追踪设置	启用链路追踪：启用该选项，则需要开通阿里云链路追踪服务。同时需要指定对应的接入点地址。例如，接入点地址为`http://tracing-analysis-dc-hz.aliyuncs.com/.../api/v1/spans`，表示启用该选项后，zipkin客户端根据v1版本的 API的公网（或者内网）接入点地址把采集数据传输到链路跟踪。 说明: 如果使用内网接入点，请确保Kubernetes集群与链路追踪服务在相同区域，保证网络互通。
Pilot设置	跟踪采样百分比（0-100）：默认取值为1。

配置	说明
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。 · 勾选全部选项则会拦截所有对外访问的地址。 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。

e) 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。

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，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。

· 单击左侧导航栏路由与负载均衡 > 服务，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。

2. 管理Istio Ingress Gateway。

   上述部署 Istio之后，Istio 1.1.4之后默认会创建一个Ingress Gateway。对于已有旧版本，建议升级到当前最新版本。如果需要对Ingress Gateway的配置进行调整，可以按照如下步骤进行更新。

   a) 在左侧导航栏选择应用 > 发布，单击Helm页签。

   b) 单击Istio右侧的更新，在弹出的更新发布页面中，可以看到如下示例参数定义。

```
gateways:
  enabled: true
  ingress:
    - enabled: true
      gatewayName: ingressgateway
      maxReplicas: 5
      minReplicas: 1
      ports:
        - name: status-port
          port: 15020
          targetPort: 15020
        - name: http2
          nodePort: 31380
          port: 80
          targetPort: 80
        - name: https
          nodePort: 31390
          port: 443
          targetPort: 0
        - name: tls
          port: 15443
          targetPort: 15443
      replicaCount: 1
      serviceType: LoadBalancer
  k8singress: {}
```

   c) 修改相应的参数后，单击更新。

   说明:

   - replicaCount：指定副本数。
   - ports：设置启用的端口。
   - serviceType：指定服务类型，可以设置为LoadBalancer、ClusterIP或者NodePort。
   - serviceAnnotations：当服务类型为设置LoadBalancer时，通过设置serviceAnnotations参数指定使用内网还是公网负载均衡、使用已有的SLB等；其他参数具体参见#unique_304。

3. 添加网关。

   作为Istio on ACK安装的一部分，提供了一个Istio入口网关。升级时，Istio和包括默认入口网关在内的所有默认资源都会自动升级。

   如果添加入口或出口网关，则它们在您的控制之下，并且在自动升级期间不会对其进行修改。请注意，在1.1及更高版本中，默认情况下不安装Istio egress网关。

   默认入口网关适用于已安装资源（RBAC、服务、部署），除了添加字段之外不需要自定义的部署。请勿更改默认入口网关配置中的任何值，因为在自动升级期间更改将恢复为默认值。对于需要自定义的更复杂方案，您需要创建新的入口网关。

17.1.3 更新Istio

您可以通过更新操作，编辑已部署的Istio。

前提条件

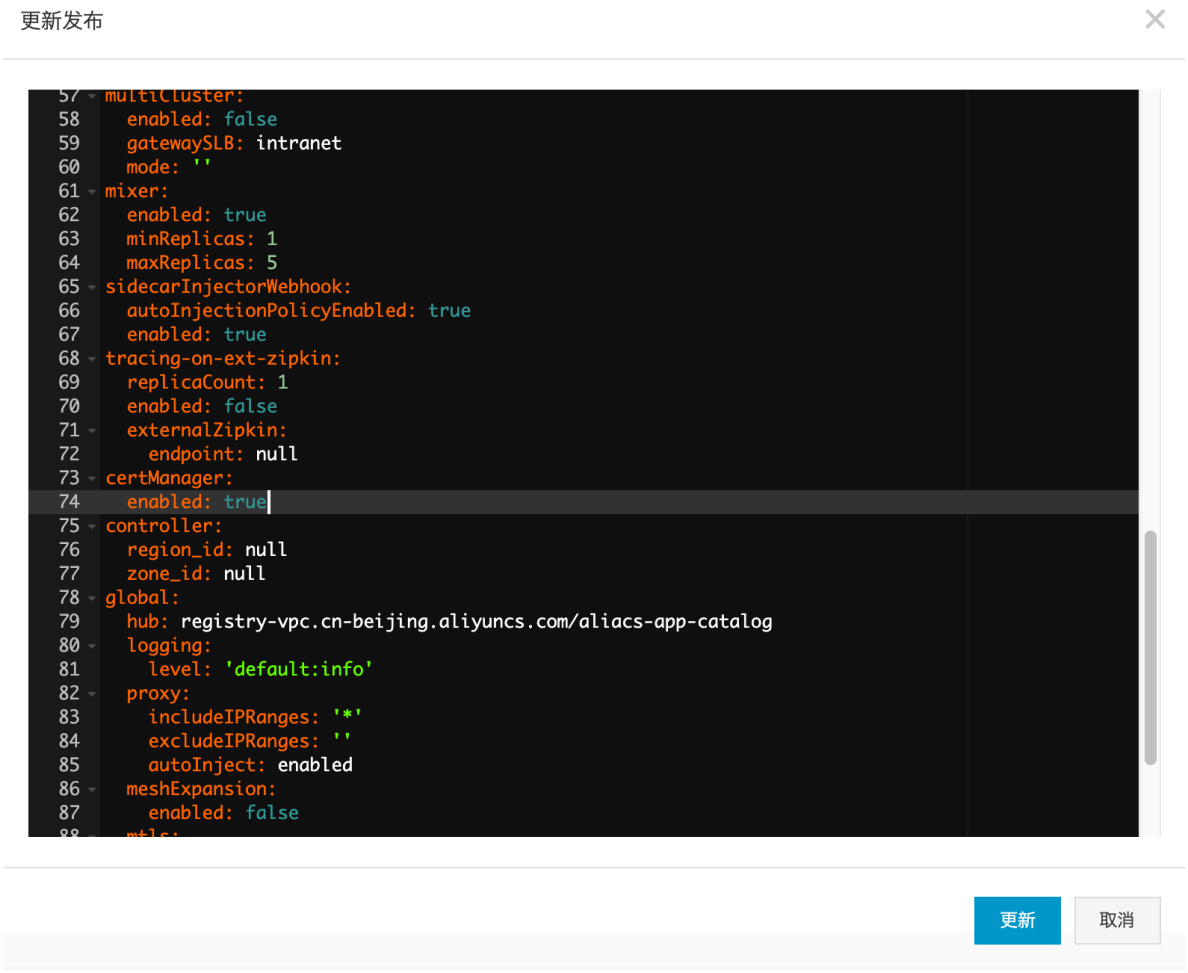
- 您已成功创建一个 Kubernetes 集群，参见#unique_95。
- 您已成功创建一个Istio，参见#unique_306。

操作步骤

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏的服务网格 > Istio管理，进入Istio管理页面。
3. 在页面的右上方，单击更新按钮。

4. 在弹出的对话框中，对Istio的参数进行编辑后，单击更新。

   如下图所示：



预期结果

可通过以下方式，查看更新后的内容：

- 在 Kubernetes 菜单下，单击左侧导航栏的应用 > 容器组，进入容器组（Pod）页面，选择目标集群和目标空间进行查看。

17.1.4 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#unique_95。
- 您已成功创建一个Istio，参见#unique_306。
- 在安装Istio的集群中，名称为istio-init的Helm发布正常运行中。在执行如下删除操作前不能删除istio-init。

- 您已连接到Kubernetes集群的Master节点，参见#unique_43。

操作步骤

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏的服务网格 > Istio管理，进入Istio管理页面。
3. 在页面的右上方，单击删除按钮。
4. 在弹出的对话框中，单击确定。
5. 单击左侧导航栏的应用 > 容器组，选择对应的集群和命名空间istio-sytem，确认Istio相关容器删除。

> 说明:
> 如果出现命名空间istio-system的istio-config资源出现无法删除的情况，请参考删除后custom resource对象 残留进行操作。

预期结果

单击左侧导航栏的应用 > 容器组，选择对应的集群和命名空间istio-sytem，确认Istio相关容器删除。

17.1.5 Istio组件升级

本文介绍Istio组件如何升级。

背景信息

- 升级过程可能涉及新的二进制文件以及配置和 API Schemas 等其他更改。
- 升级过程可能导致一些服务宕机 。
- 为了最大限度地减少宕机时间，请使用多副本以保证 Istio 控制平面组件和应用程序具有高可用性。
- 升级过程中可能会因为配置参数的变化对服务的SLB重新创建，建议通过设置SLB删除保护来保留已有SLB。

> 说明:
> 假设 Istio 组件在 istio-system namespace 中安装和升级。

升级过程可以分为3个部分：CRD升级、控制平面升级、数据平面Sidecar升级。

> 说明:
> 升级过程中可能会导致服务不可用，为了最大限度地减少宕机时间，请使用多副本以保证 Istio 控制平面组件和应用程序具有高可用性，包括控制平面和数据平面。

CRD升级及控制平面升级

1. 登录容器服务管理控制台。
2. 单击左侧导航栏中的服务网格 > Istio管理，进入Istio管理页面。如果版本低于1.1.4， 请工单联系进行升级。如果版本不低于1.1.4，并且提示升级，则单击升级链接进行自动升级。注意：升级过程会对之前做过的针对Kubernetes资源(如服务、部署等)的直接修改进行重置，因此建议所有的修改都通过Istio更新操作以保证所做的修改不会被重置。

数据平面Sidecar升级

控制平面升级后，已经运行 Istio 的应用程序仍将使用旧版本的 Sidecar。升级Sidecar，则需要重新注入。您可以选择以下两种操作升级Sidecar。

- 自动注入Sidecar

  如果使用自动注入Sidecar，可以通过对所有 pod 进行滚动升级来升级 Sidecar，这样新版本的 Sidecar 将被自动重新注入。

  可以使用如下脚本通过 patch 优雅结束时长来触发滚动更新：

```
NAMESPACE=$1
DEPLOYMENT_LIST=$(kubectl -n $NAMESPACE get deployment -o jsonpath
='{.items[*].metadata.name}')
echo "Refreshing pods in all Deployments: $DEPLOYMENT_LIST"
for deployment_name in $DEPLOYMENT_LIST ; do
    #echo "get TERMINATION_GRACE_PERIOD_SECONDS from deployment: $
deployment_name"
    TERMINATION_GRACE_PERIOD_SECONDS=$(kubectl -n $NAMESPACE get
 deployment "$deployment_name" -o jsonpath='{.spec.template.spec.
terminationGracePeriodSeconds}')
    if [ "$TERMINATION_GRACE_PERIOD_SECONDS" -eq 30 ]; then
        TERMINATION_GRACE_PERIOD_SECONDS='31'
    else
        TERMINATION_GRACE_PERIOD_SECONDS='30'
    fi
    patch_string="{\"spec\":{\"template\":{\"spec\":{\"terminatio
nGracePeriodSeconds\":$TERMINATION_GRACE_PERIOD_SECONDS}}}}"
    #echo $patch_string
    kubectl -n $NAMESPACE patch deployment $deployment_name -p $
patch_string
done
```

```
echo "done."
```

- 手动注入Sidecar

  执行以下命令，手动升级 sidecar。

  ```
  kubectl apply -f <(istioctl kube-inject -f $ORIGINAL_DEPLOYMENT_YAML
  )
  ```

  如果 Sidecar 以前被注入了一些定制的注入配置文件，请执行以下命令，手动升级 Sidecar：

  ```
  kubectl apply -f <(istioctl kube-inject --injectConfigFile inject-
  config.yaml --filename $ORIGINAL_DEPLOYMENT_YAML)
  ```

升级中的影响

- CRD升级

  对于集群内服务间的调用、gateway到服务间的调用均没有产生断连影响。

  集群内服务间的调用：

- 控制平面升级

  在启用HA（即pilot的replicas为2）的情况下，`istio-pilot/istio-policy/istio-telemetry`的HPA设置：`minReplicas: 2`。

  此时，在多次变更版本（升级、回滚）的情况下，测试结果显示为：服务之间的调用QPS基本不变，没有出现断连。

- 数据平面Sidecar升级

  无论是集群内服务之间的调用，还是通过gateway到服务的调用，QPS变化不大，但会出现断连情况。建议启动多副本的方式，降低影响。

17.1.6 设置 Sidecar 自动注入

本文为您介绍如何在某个命名空间中开启/关闭 Sidecar 自动注入功能。

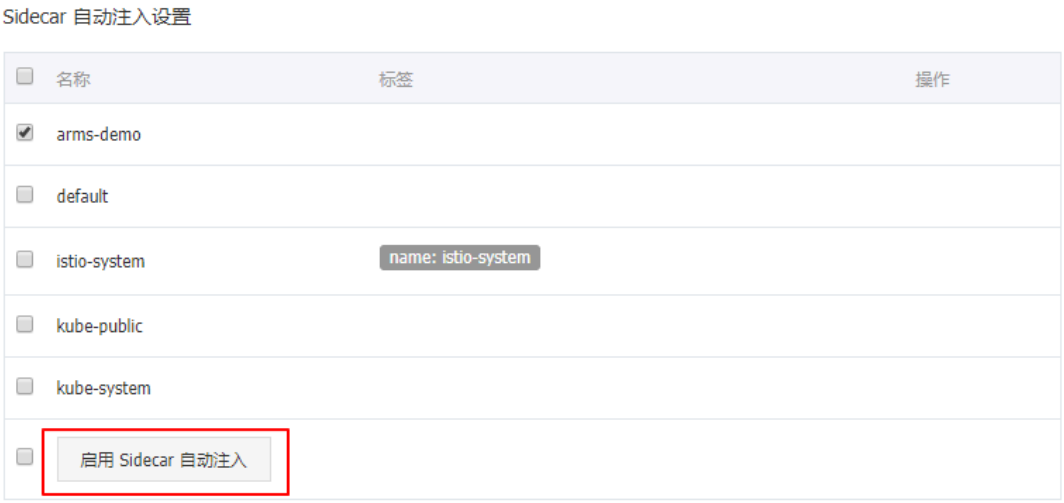
前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_42。
- 您已成功部署Istio，参见#unique_306。

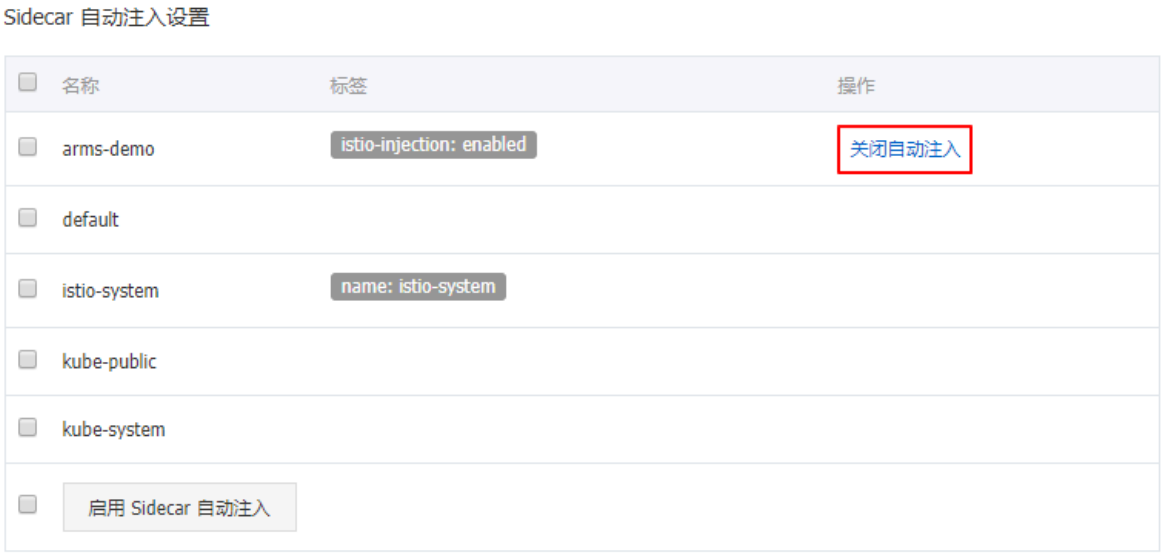
操作步骤

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服务网格 > Istio管理，进入部署Istio页面。

3. 在Sidecar 自动注入设置区域勾选一个或者多个命名空间，单击启用 Sidecar 自动注入。



4. 在弹出的提示框中单击确定。即在目标命名空间下启动Sidecar 自动注入功能。
5. （可选） 如果您想要关闭 Sidecar 自动注入功能，单击命名空间右侧的关闭自动注入，在弹出的对话框中单击确定即可。



17.2 服务网关

17.2.1 部署服务网关

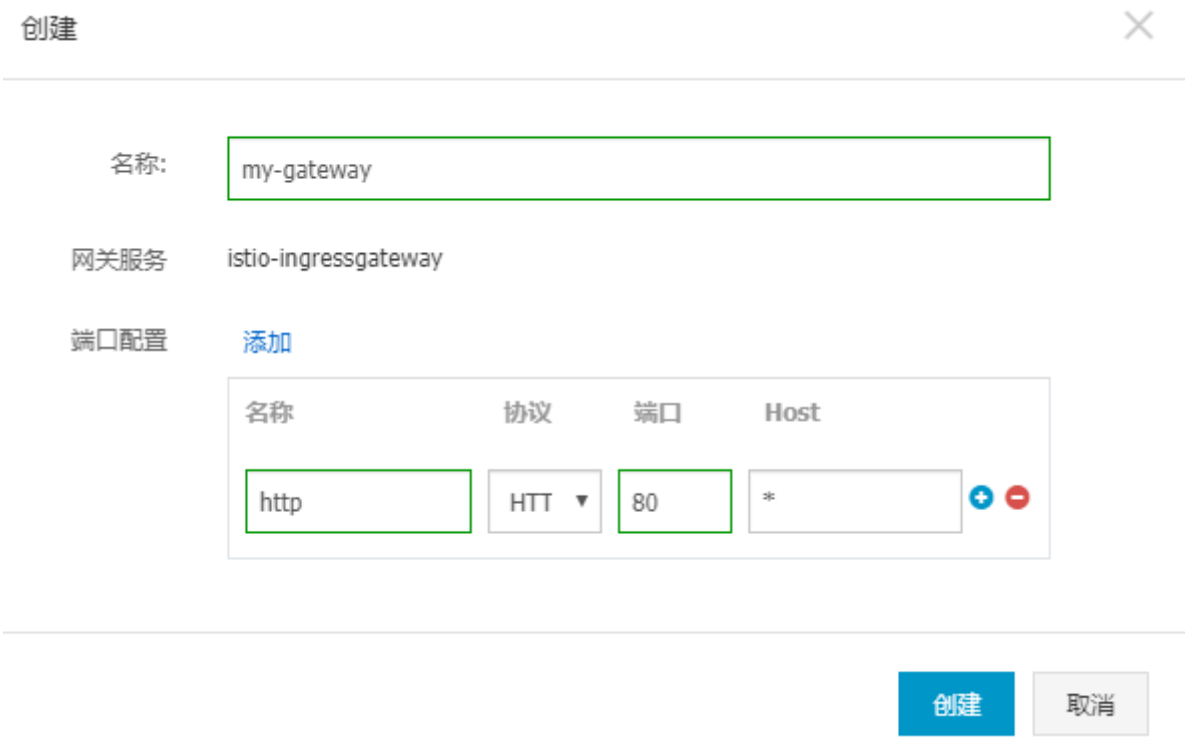
本文为您介绍如何部署服务网关。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_42。
- 您已成功部署Istio，参见#unique_306。

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服务网格 > 服务网关，进入服务网关页面。
3. 选择所需的集群及命名空间后，单击右上角新建。
4. 填写如下信息，单击创建，部署服务网关。



参数说明：

- 名称：新创建的服务网关的名称。
- 服务网关：服务代理网关。默认为istio-ingressgateway。代理网关必须和新创建的网关部署在同一命名空间。
- 端口配置：单击添加，可以增加多个端口的配置。
  - 名称：端口的名称
  - 协议：HTTP/TCP
  - 端口：1-65535
  - Host：支持多个host

预期结果

您可以在服务网关列表中，看到新创建的服务网关。

17.2.2 管理服务网关

本文主要为您介绍如何通过控制台修改或删除服务网关。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_42。
- 您已成功部署Istio，参见#unique_306。
- 您已经成功部署服务网关，参见部署服务网关。

更新服务网关

您可以通过更新yaml的方式修改服务网关配置。

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服务网格 > 服务网关，进入服务网关页面。
3. 在目标服务网关右侧单击查看Yaml。

   可以看到如下示例参数定义。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  creationTimestamp: '2019-08-23T06:50:25Z'
  generation: 1
  name: my-gateway
  namespace: default
  resourceVersion: '15172143'
  selfLink: /apis/networking.istio.io/v1alpha3/namespaces/default/
gateways/my-gateway
  uid: 492db126-c572-11e9-b65a-00163e13****
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: http
        number: 80
```

```
        protocol: HTTP
```

- `name`：服务网关的名称。本例为`my-gateway`。
- `hosts`：主机域名。
- `name`：服务网关端口名称。本例为`http`。
- `number`：服务网关端口号。
- `protocol`：服务端口协议。

4. 修改完成后，单击更新。

删除服务网关

1. 在 Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > 服务网关，进入服务网关页面。
2. 单击操作列的删除。
3. 在弹出提示对话框中，单击确定，删除服务网关。

17.3 基于Istio实现跨区域多集群部署

容器服务Kubernetes版（简称ACK）提供高性能可伸缩的容器应用管理能力，支持企业级 Kubernetes容器化应用的全生命周期管理。基于Istio，可以轻松地跨多个区域中的Kubernetes集 群实现流量统一管理，例如在阿里云容器服务中部署了多个ACK集群，每个集群位于不同的区域中。 如果最靠近用户的实例发生故障或过载，则流量将无缝地定向到另一个可用实例。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_48。
- 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#unique_315。
- 基于Flat网络或者VPN的多集群部署方式，多个Kubernetes集群应位于同一个VPC下网 络互通，或者通过云企业网、高速通道等打通跨VPC网络的访问；同时要求Pod/Service CIDR不能冲突。
- 以下文档使用基于Flat网络或者VPN的多集群Istio部署方式管理服务。

背景信息

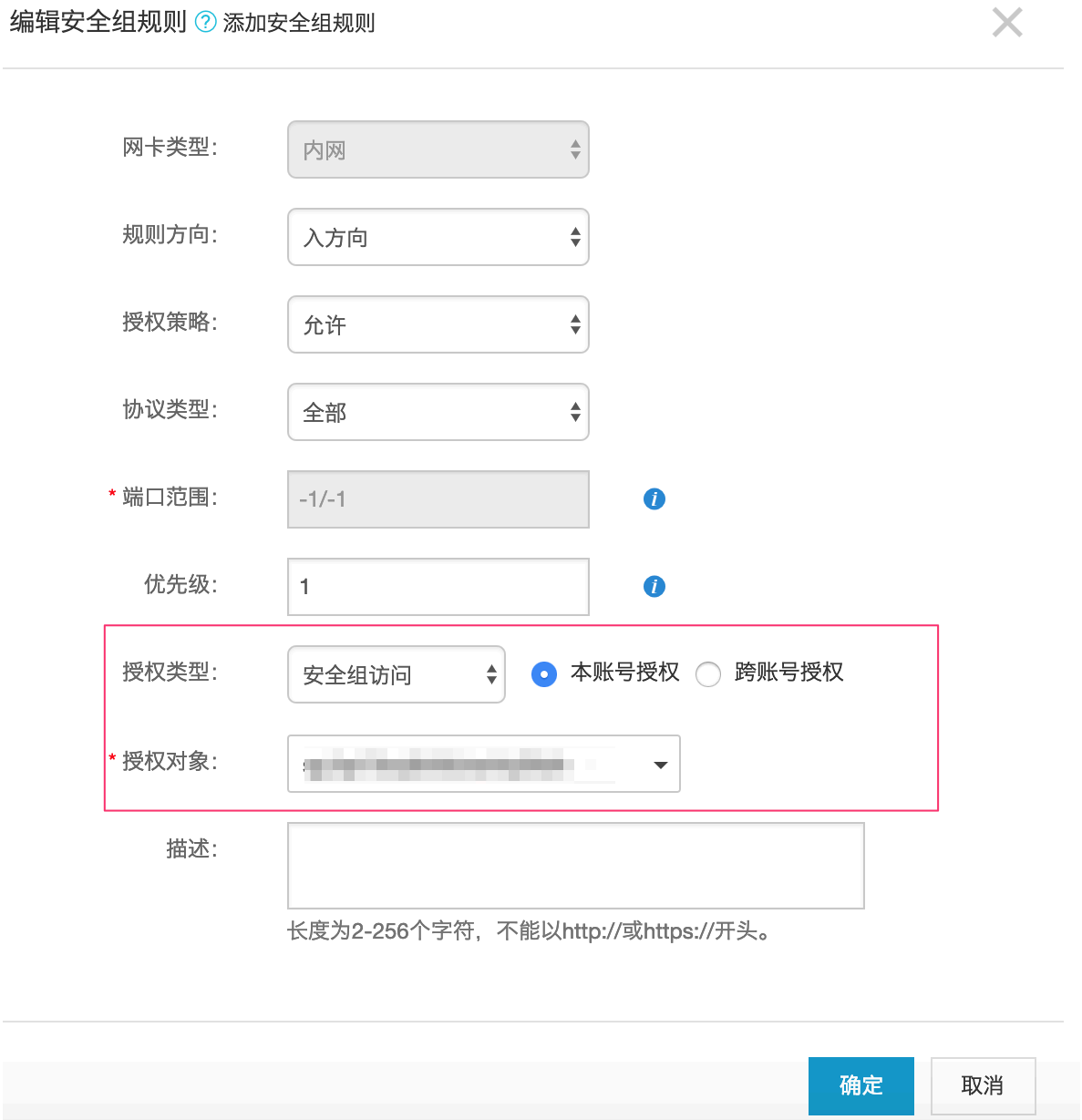
- 阿里云容器服务Kubernetes 1.10.4及之后版本支持部署Istio，如果是1.10.4之前的版本，请先升级到1.10.4或之后版本。
- 一个集群中，worker节点数量需要大于等于3个，保证资源充足可用。

操作步骤

配置多集群网络

- 同一VPC下

同一VPC下，无论集群是在同一交换机下还是多个交换机下，网络互通，多个安全组需要手工设置打通。不同安全组默认情况下不能互相访问，同一VPC下可以通过设置规则使其互通，请参见#unique_316。



- 不同VPC下

  不同VPC下，无论是否在相同地域、可用区，集群之间网络本身不能互通，需要设置VPC互通（云企业网、VPN网关、高速通道），云企业网的配置更加简单而且可以自动分发学习路由，因此，推荐您使用云企业网。

  > 说明:
  > VPC网络不能冲突，默认建立集群时创建的VPC网络段都是192.168.0.0/16；所以需要手工创建VPC，并指定不同的网段。

  此外，多个安全组需要手工设置打通，可以参考#unique_316。

  数据平面上的安全组添加对应的控制平面的网段：

  类似地，控制平面上的安全组也同时添加相应的网段。

  此外，网络规划需要提前设置好，保证2个集群内的service/pod CIDR、VPC CIDR都不能重叠。

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。

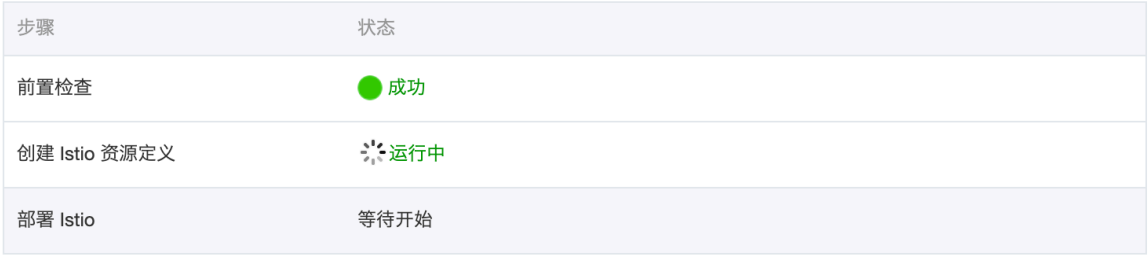
4. 针对多集群部署，根据如下信息进行配置：

   与部署Istio到单个集群的方式一样，可以使用阿里云提供的控制台一键部署，请参见#unique_318。

配置	说明
启用服务就近访问	在跨区域的多集群部署中是否启用服务就近访问。默认情况下不启用。
多集群同一控制平面模式设置	· 不启用：不启用多集群模式。 · 使用Flat网络或者VPN：基于Flat网络或者VPN方式实现多集群中的网络，多集群间 的Pod可以互通。 · 不使用Flat网络或者VPN：不使用Flat网络或者VPN，多集群之间仅通过网关进行交互。

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。

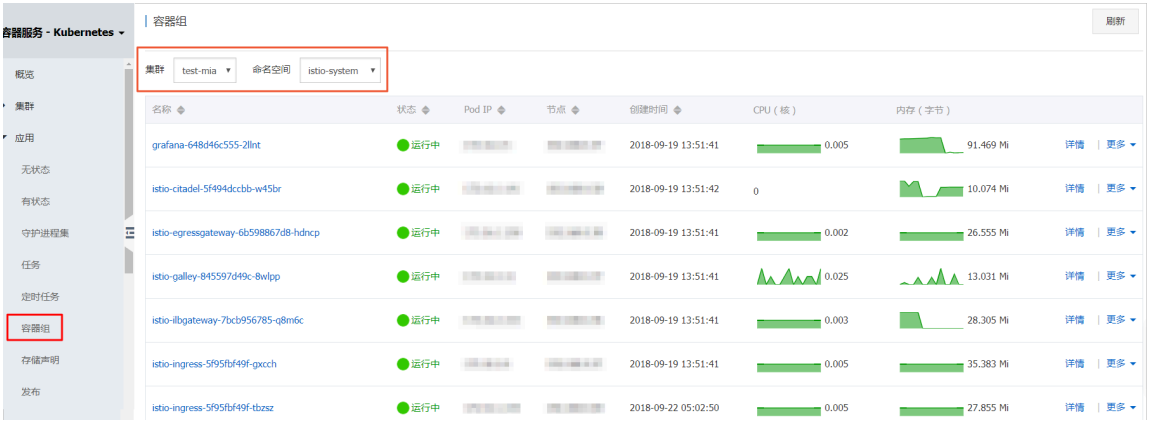


   预期结果

   可通过以下操作查看部署是否成功：

   a. 单击左侧导航栏应用 > 容器组，进入容器组页面。

   b. 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。



管理Istio Ingress Gateway。

1. 上述部署 Istio之后，包含一个入口网关Ingress Gateway，该网关使用公网IP的负载均衡。通过该网关将集群中应用服务暴露到集群之外。
2. 执行`kubectl get service -n istio-system -l istio=ingressgateway`，获取入口网关的公网IP地址，应用访问可以通过该Gateway进行。

```
 kubectl get service -n istio-system -l istio=ingressgateway
NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
                AGE
istio-ingressgateway   LoadBalancer   [redacted]      [redacted]      15020:31040/TCP,80:31380/TCP,443:31390/TCP,1
5443:30221/TCP   6d22h
```

将数据平面加入Istio管理

1. 在数据平面的Kubernetes集群上执行如下生成 kubeconfig 文件。

```
kubectl create namespace istio-system
kubectl apply -f http://istio.oss-cn-hangzhou.aliyuncs.com/istio-
operator/rbac.yml
wget http://istio.oss-cn-hangzhou.aliyuncs.com/istio-operator/
generate-kubeconfig.sh
chmod +x generate-kubeconfig.sh
./generate-kubeconfig.sh ${CLUSTER_NAME}
```

说明:

*${CLUSTER_NAME}*需要在整个服务网格内保证唯一。

2. 在Istio控制平面集群下执行如下命令。

```
wget http://istio.oss-cn-hangzhou.aliyuncs.com/istio-operator/create
-secret.sh
chmod +x create-secret.sh
./create-secret.sh ${CLUSTER_NAME}
```

3. 在Istio控制平面集群上，创建并拷贝内容到*${CLUSTER_NAME}.yaml*文件中，并执行`kubectl apply -n istio-system -f *${CLUSTER_NAME}*.yaml`。

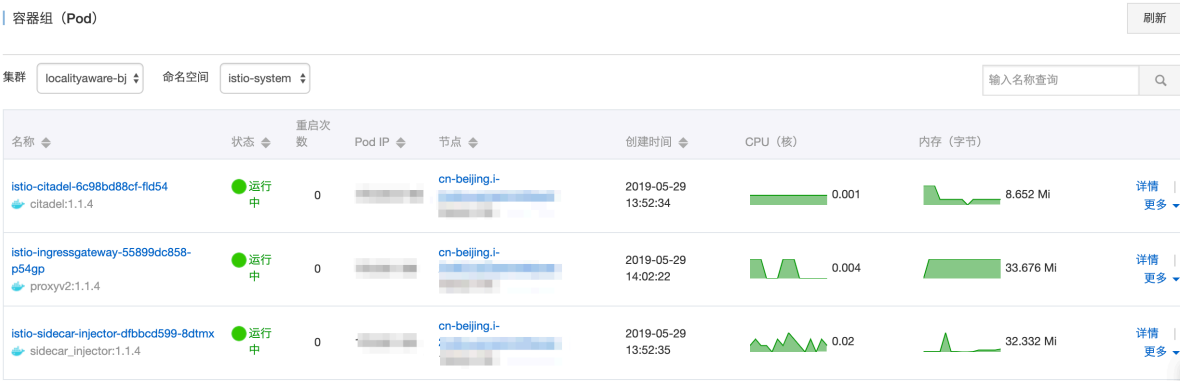
```
apiVersion: istio.alibabacloud.com/v1beta1
kind: RemoteIstio
metadata:
  name: ${CLUSTER_NAME}
  namespace: istio-system
spec:
  autoInjectionNamespaces:
  - default
  dockerImage:
    region: cn-beijing
  gateways:
    k8singress: {}
  hub: registry.cn-beijing.aliyuncs.com/aliacs-app-catalog
  includeIPRanges: '*'
  proxy: {}
  security: {}
  sidecarInjector:
    enabled: true
```

```
    replicaCount: 1
```

预期结果：

可通过以下操作查看远程集群的Istio部署是否成功：

1. **切换到远程集群，单击左侧导航栏应用 > 容器组，进入容器组页面。**
2. **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。**



17.4 绑定虚拟服务

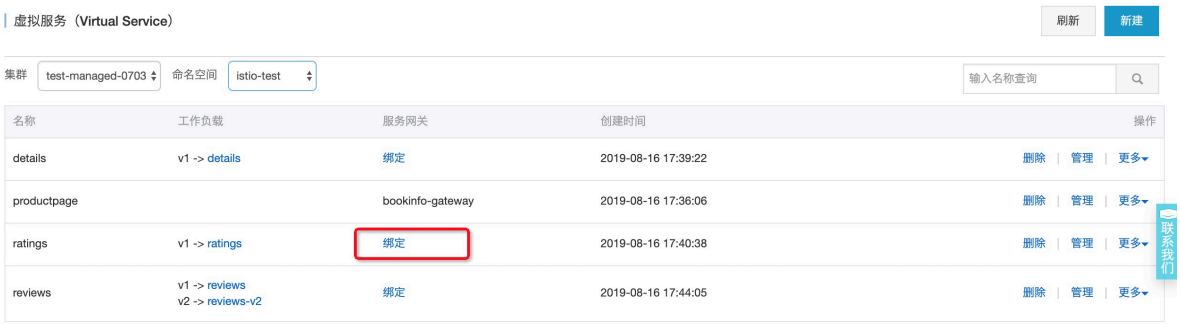
本文主要为您介绍如何绑定虚拟服务。

前提条件

- **您已成功创建一个 Kubernetes 集群，参见#unique_42。**
- **您已成功部署Istio，参见#unique_306。**
- **您已经创建了一个虚拟服务。**

操作步骤

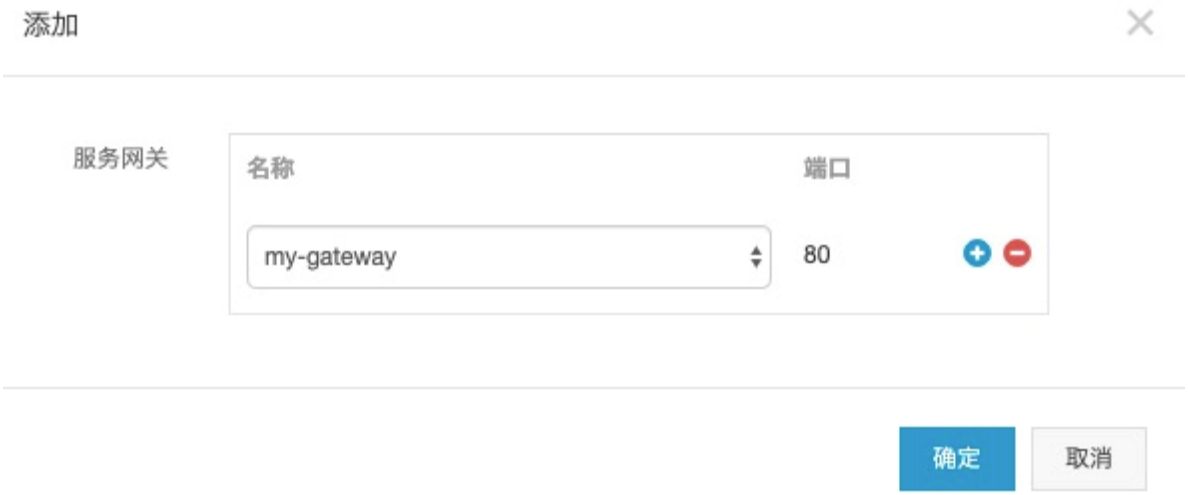
1. **登录容器服务管理控制台。**
2. **在 Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > 虚拟服务，可以看到虚拟服务（Virtual Service）列表。**



说明：

未绑定网关的虚拟服务在服务网关列会显示绑定按钮。

3. 在目标虚拟服务的服务网关列单击绑定，弹出添加页面。

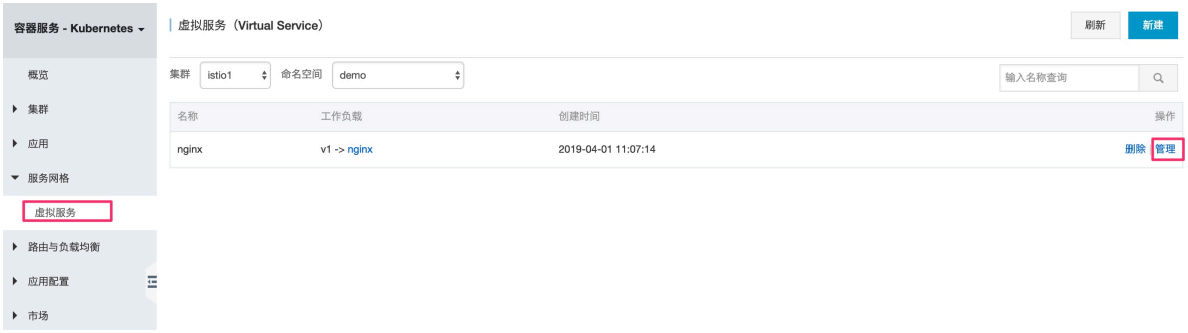


4. 在服务网关中选择对应的网关名称，单击确定，即完成虚拟服务的绑定。

17.5 流量管理

本文档为您介绍如何通过Istio管理流量。Istio提供的流量治理策略包含负载均衡算法设置、会话保持，连接池管理、熔断配置和故障注入等。

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。
3. 单击对应虚拟服务右侧的管理，进入详情界面，您可以根据需要，进行流量管理。



流量治理

在reviews详情页，单击流量治理，进入如下页面。

reviews
返回列表
刷新
基本信息
工作负载类型:
deployment
创建时间:
2019-03-26 16:19:58
版本管理
增加灰度版本
修改灰度策略
故障注入策略
版本号
工作负载
实例数
灰度策略
创建时间
操作
v2
reviews-v2
1
查看
2019-03-26 16:17:57
管理负载 | 删除版本 | 接管所有流量 | 流量治理
v3
reviews-v3
1
2019-03-26 16:17:57
管理负载 | 删除版本 | 接管所有流量 | 流量治理

- **负载均衡设置：Istio提供标准负载均衡算法和会话保持两种方式，两者只能取其一。**

图 17-1: 负载均衡算法

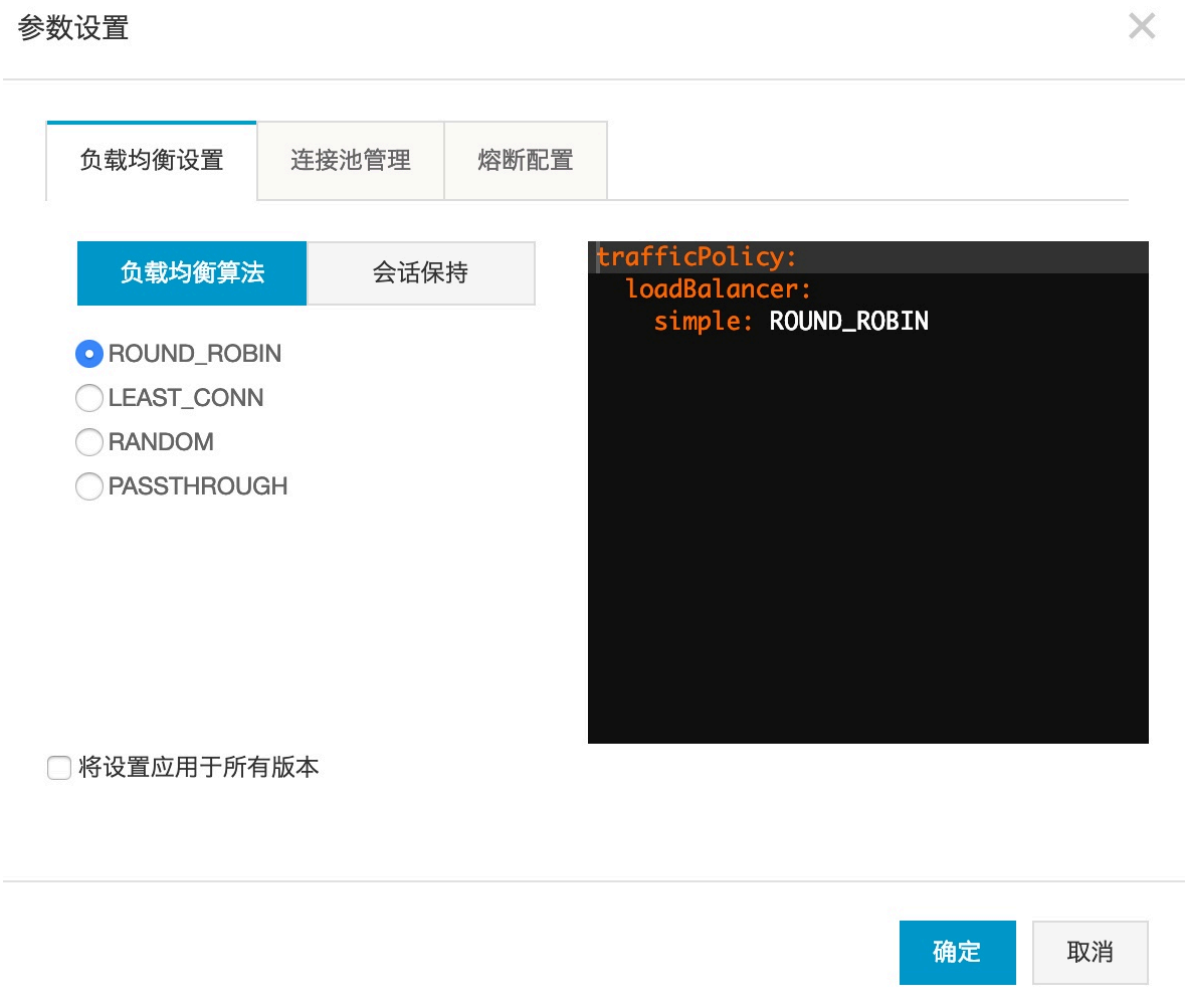


图 17-2: 会话保持

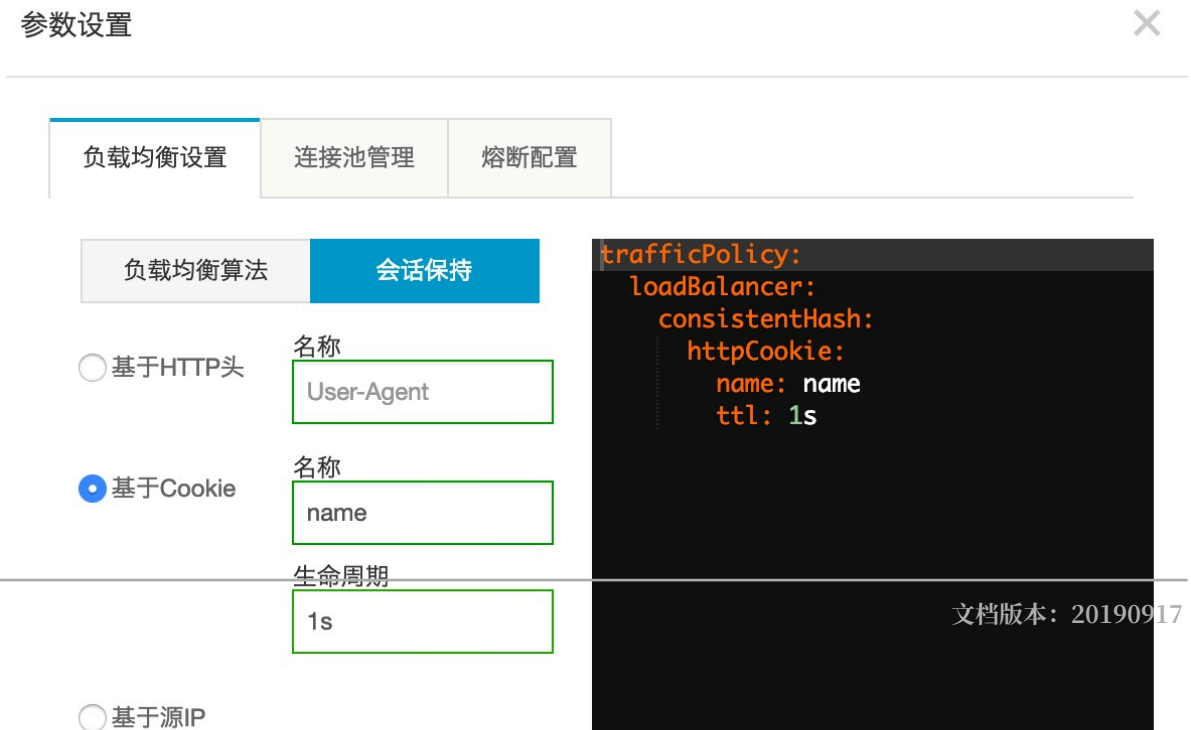


表 17-1: 负载均衡设置

参数	参数名称	参数说明
loadBalancer	负载均衡算法	- ROUND_ROBIN：轮询调度策略。是Istio代理的缺省算法。此算法按顺序在负载平衡池中的端点上均匀分配负载。 - LEAST_CONN：随机选择两个健康主机，从中选择一个较少请求的主机提供服务。 这是加权最小请求负载平衡的实现。 - RANDOM：随机的负载均衡算法。从所有健康的主机中随机选择一个主机，在负载平衡池的端点上均匀分配负载。在没有健康检查策略的情况下，随机策略通常会比轮询调度策略更加高效，但不会有任何顺序。 - PASSTHROUGH：这个策略会直接把请求发给客户端要求的地址上。请谨慎使用。
consistentHash	会话保持	- 基于HTTP头：根据HTTP header中的内容获取哈希。 - 基于Cookie：根据Cookie中的内容获取哈希。 - 基于源IP：根据源 IP 中的内容获得哈希。

- **连接池管理。**

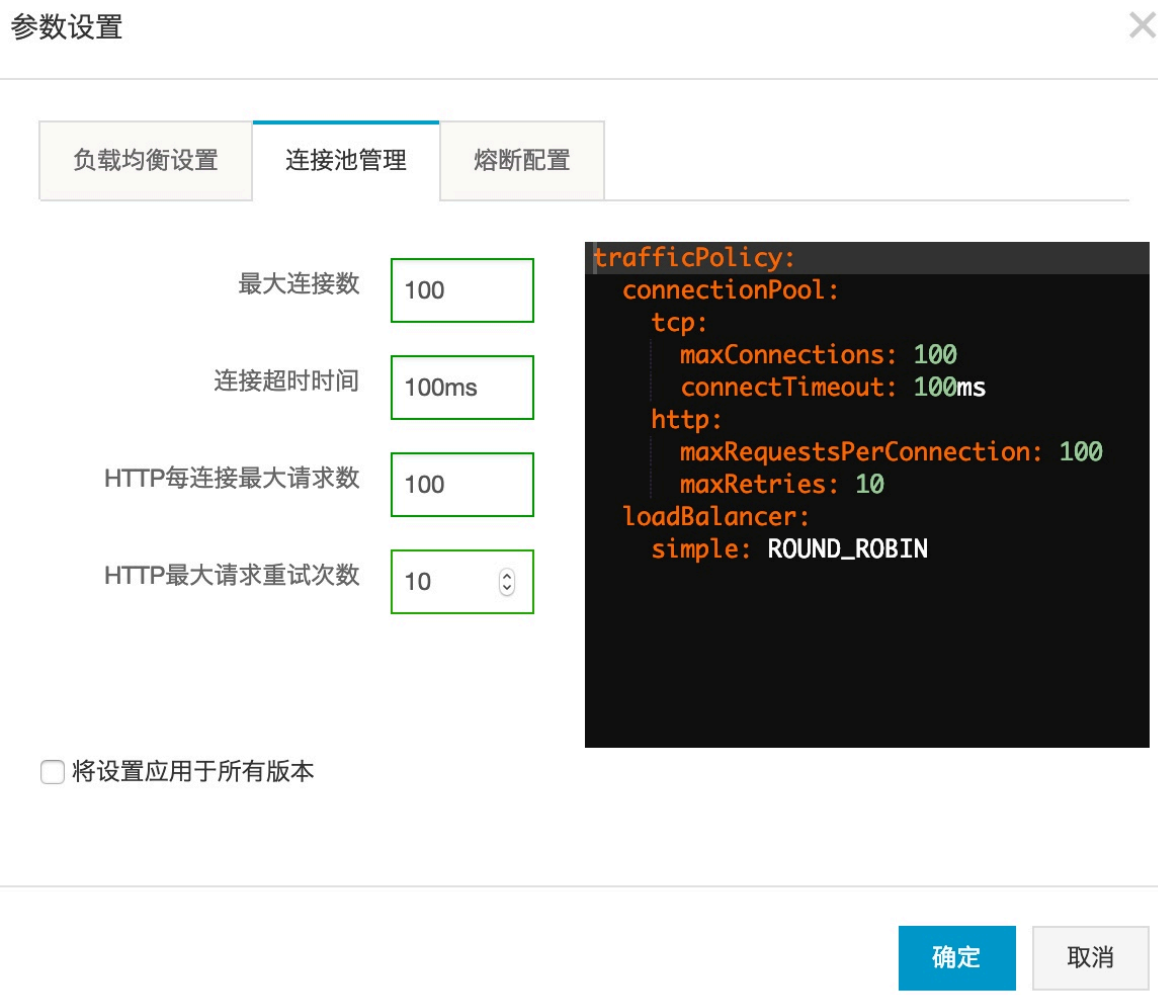


表 17-2: 连接池管理

参数	参数名称	参数说明
maxConnections	最大连接数	是指Envoy将为上游群集中的所有主机建立的最大连接数。适用于HTTP/1.1群集。 **说明:** 最大连接数和连接超时时间是对 TCP 和 HTTP 都有效的通用连接设置。

参数	参数名称	参数说明
connectTimeout	连接超时时间	TCP连接超时时间。最小值必须大于1ms。 说明: 最大连接数和连接超时时间是对 TCP 和 HTTP 都有效的通用连接设置。
maxRequestsPerConnection	HTTP每连接最大请求数	对某一后端的请求中，一个连接内能够发出的最大请求数量。对后端连接中最大的请求数量若设为1，则会禁止keep alive特性。 说明: HTTP每连接最大请求数和HTTP最大请求重试次数仅针对 HTTP1.1/HTTP2/GRPC 协议的连接生效。
maxRetries	HTTP最大请求重试次数	在指定时间内对目标主机最大重试次数。默认值为3。 说明: HTTP每连接最大请求数和HTTP最大请求重试次数仅针对 HTTP1.1/HTTP2/GRPC 协议的连接生效。

- **熔断设置：您可以在流量策略中选择进行熔断策略配置。**

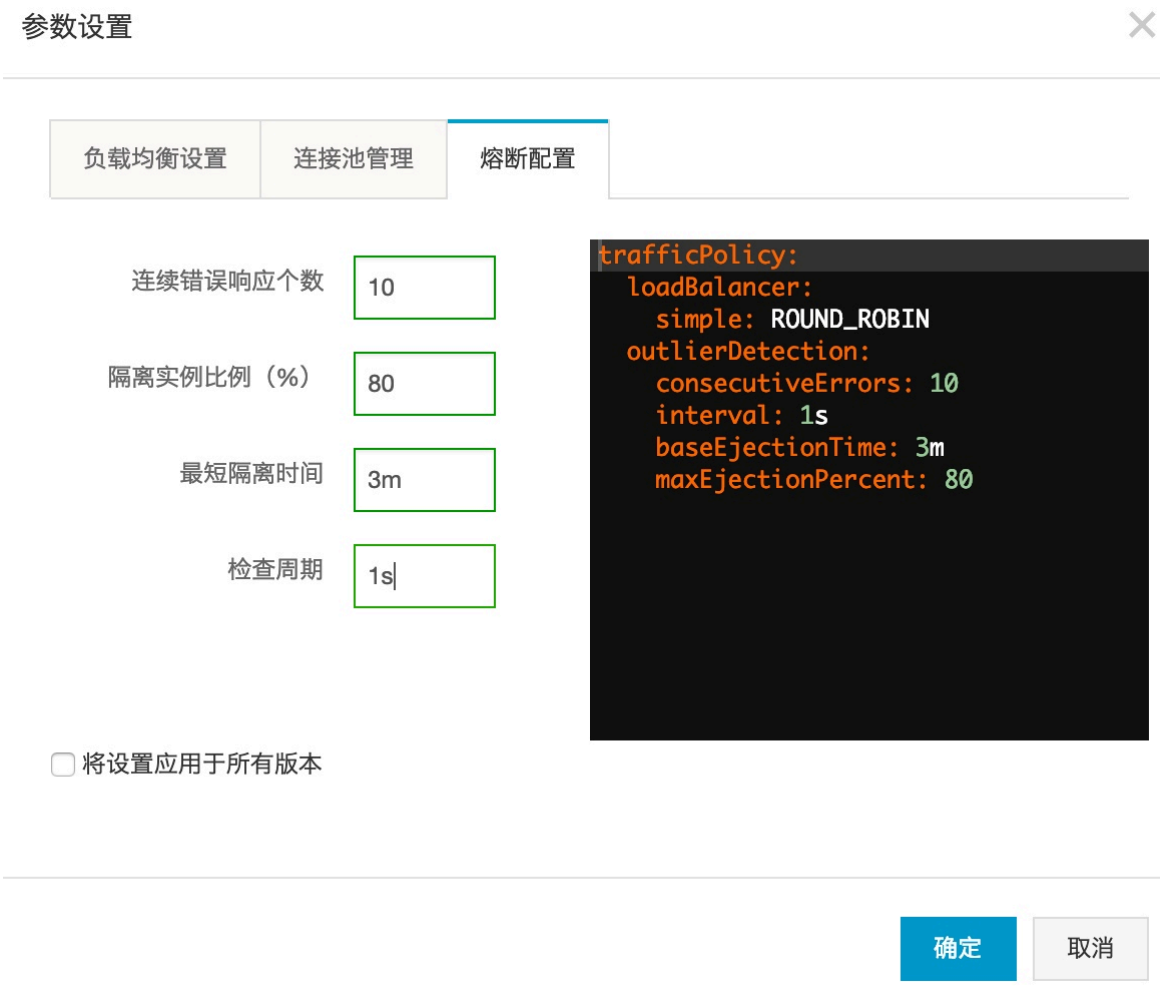


表 17-3: 熔断配置

参数	参数名称	参数解释
consecutiveErrors	连续错误响应个数	周期内连续出现错误的个数。超过该值后，主机将会被移出连接池。默认值是5。 **说明:** 当上游服务是 HTTP 服务时，5xx 的返回码会被记为错误；当上游主机提供的是 TCP 服务时，TCP 连接超时和连接错误/故障事件会被记为错误。

参数	参数名称	参数解释
maxEjectionPercent	隔离实例比例（%）	上游服务的负载均衡池中允许被移除的主机的最大百分比。默认值为10%。
baseEjectionTime	最短隔离时间	最小的移除时间长度。主机每次被移除后的隔离时间为被移除的次数和最小移除时间的乘积。这样的实现，让系统能够自动增加不健康上游服务实例的隔离时间。默认值为30s。
interval	检查周期	检测主机上一次被移除和这一次被移除之间的时间间隔。默认值为 10s，最小值为1ms。

说明:

勾选将设置应用于所有版本，上述设置在reviews服务的各个版本均生效。

故障注入策略

故障注入策略包含时延故障和中断故障策略，当前故障注入支持HTTP。

- 时延故障（delay）：转发之前加入延迟，用于模拟网络故障、上游服务过载等故障。

表 17-4: 时延故障

参数	参数名称	参数解释
percent	百分比	取值范围为0~ 100，用来指定注入延迟的比例。
fixedDelay	时延	必填项。在请求转发之前设定的时间延迟单位。取值单位包括小时（h）、分钟（m）、秒钟（s）和毫秒（ms），允许的最小值是 1ms。

- 中断故障（abort）：终止请求并向下游服务返回错误代码，模拟上游服务出错的状况。

表 17-5: 中断故障

参数	参数名称	参数解释
percent	百分比	取值范围为 0 ~ 100，用来指定中断请求的比例。
httpStatus	状态码	必填项。来定义返回给客户端服务的有效的 HTTP 错误代码。

17.6 Istio 常见问题

您可通过本文了解 Istio 的常见问题及解决方法。

集群内无法访问集群外的 URL

问题现象

集群内无法访问集群外的 URL。

问题原因

缺省情况下，Istio 服务网格内的 Pod，由于其 iptables 将所有外发流量都透明的转发给了 Sidecar，所以这些集群内的服务无法访问集群之外的 URL，而只能处理集群内部的目标。

解决方法

- 通过定义 ServiceEntry 来调用外部服务。
- 配置 Istio 使其直接放行对特定IP地址范围的访问。

详细内容，可参考Control Egress Traffic

Tiller 版本较低

问题现象

安装过程中遇到如下提示：

```
Can't install release with errors: rpc error: code = Unknown desc =
Chart incompatible with Tiller v2.7.0
```

问题原因

Tiller 版本较低，需要升级。

解决方法

执行以下任意一条命令，升级 Tiller 版本。

说明：

此处请升级到v2.10.0及以上版本。

升级到v2.11.0版本：

```
helm init --tiller-image registry.cn-hangzhou.aliyuncs.com/acs/tiller:
v2.11.0 --upgrade
```

升级到v2.10.0版本：

```
helm init --tiller-image registry.cn-hangzhou.aliyuncs.com/acs/tiller:
v2.10.0 --upgrade
```

说明：

Tiller版本升级后，建议将客户端也升级到相应版本，客户端下载地址，请参考https://github.com/helm/helm/releases。

CRD（Custom Resource Definitions）版本问题

问题现象

在第一次创建或升级 Istio 1.0 时遇到如下提示：

```
Can't install release with errors: rpc error: code = Unknown desc =
apiVersion "networking.istio.io/v1alpha3" in ack-istio/charts/pilot/
templates/gateway.yaml is not available
```

问题原因

CRD 不存在或版本较低，需要安装最新版本的 CRD。

说明：

仅Helm版本为2.10.0及之前版本，会遇到此问题。2.10.0之后的版本，系统自动升级 CRD。

解决方法

1. 下载新版本的 Istio，可参考Downloading the Release。
2. 执行以下命令，安装最新版本的 CRD。

```
kubectl apply -f install/kubernetes/helm/istio/templates/crds.yaml -
n istio-system
```

3. 如果启用了`certmanager`，需要执行以下命令安装相关的 CRD。

```
kubectl apply -f install/kubernetes/helm/istio/charts/certmanager/
templates/crds.yaml
```

子账号无法安装 Istio

问题现象

安装过程中遇到类似如下提示：

```
Error from server (Forbidden): error when retrieving current
configuration of:
Resource: "apiextensions.k8s.io/v1beta1, Resource=customresourcedefini
tions", GroupVersionKind: "apiextensions.k8s.io/v1beta1, Kind=
CustomResourceDefinition"
```

问题原因

当前使用账号不具有安装 Istio 的权限。

解决方法

- 切换为主账号登录。
- 对子账号赋予相应的权限，例如自定义角色中的 `cluster-admin`，可参考#unique_315。

卸载 Istio 后 CRD 残留

问题现象

卸载 Istio 后，CRD 残留。

问题原因

卸载 Istio，系统不会删除 CRD，需要执行命令删除。

解决方法

1. Helm为2.9.0及之前版本，需要执行以下命令，删除 Job 资源。

```
kubectl -n istio-system delete job --all
```

2. 执行以下命令，删除 CRD。

```
kubectl delete crd `kubectl get crd | grep -E 'istio.io|certmanager.
k8s.io' | awk '{print $1}'`
```

删除后custom resource对象 残留

问题现象

卸载 Istio 后，custom resource对象 残留。

问题原因

卸载 Istio，用户先删除了crd，但没有删除custom resource对象，需要执行命令删除。

解决方法

1. 执行`kubectl edit istio -n istio-system istio-config`命令。

2. 删除`finalizers`下的`- istio-operator.finializer.alibabacloud.com`。

17.7 Istio最佳实践

17.7.1 在Kubernetes上基于Istio实现Service Mesh智能路由

阿里云容器服务支持一键部署Istio，并支持多种扩展功能，本例中介绍如何通过Istio实现智能路由。Istio官方文档请参考intelligent-routing。

前提条件

- 您已成功部署一个Kubernetes集群，参见#unique_48。
- 您已成功部署了Istio，参见#unique_323。

说明:

本例中Isito的版本是1.0.2。

- 您需要拥有一个本地Linux环境，并已配置好Kubectl工具连接到集群，参见#unique_37。
- 您已下载对应Istio版本的项目代码，并在Istio文件目录下执行相关命令。参见https://github.com/istio/istio/releases。

安装官方示例

首先快速安装Bookinfo官方示例，具体可以参见： https://istio.io/docs/guides/bookinfo。

快速部署Bookinfo示例应用

1. 首先为`default`命名空间打上标签`istio-injection=enabled`。

   说明:
   阿里云容器服务Kubernetes集群支持一键部署Istio，并支持自动Sidecar注入。

   ```
   $ kubectl label namespace default istio-injection=enabled
   ```

2. 使用Kubectl命令部署Bookinfo示例应用。

   ```
   $ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
   ```

   上面的命令会启动全部的四个服务，其中也包括了`reviews`服务的三个版本（v1、v2 以及 v3）。

3. 确认所有的服务和 Pod 都已经正确的定义和启动。

   ```
   $ kubectl get svc,pods
   ```

4. 你需要从外部访问应用程序，例如使用浏览器。您需要创建一个Istio Gateway 。为应用程序定义入口网关。

   ```
   $ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
   ```

   确认网关创建完成：

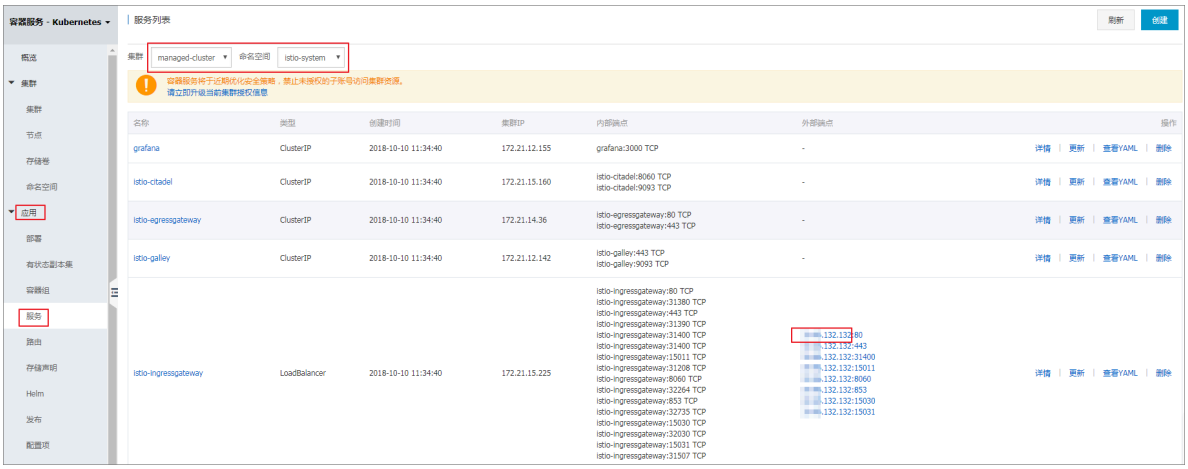
   ```
   $ kubectl get gateway
   NAME               AGE
   ```

```
bookinfo-gateway   32s
```

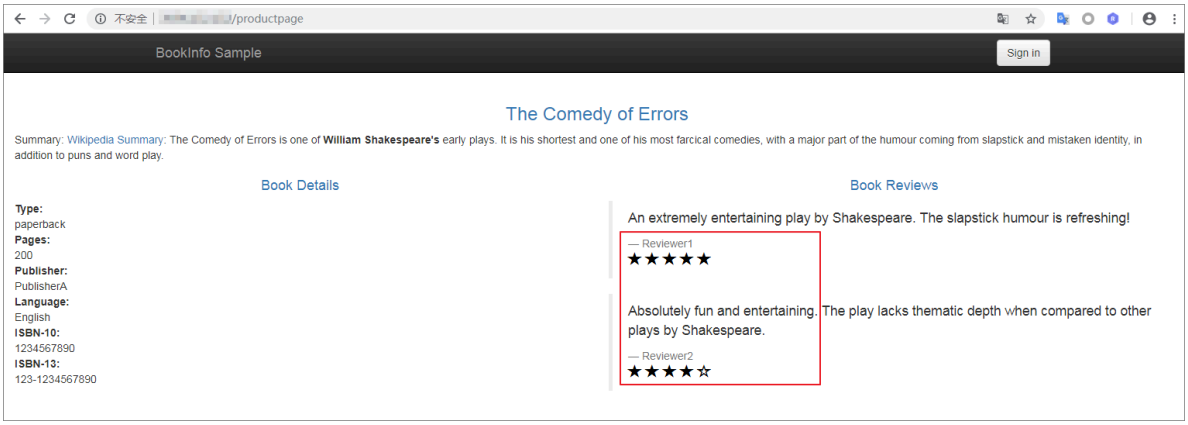
5. 然后确认istio-ingressgateway的访问地址。您可通过命令行快速查询。

```
$ kubectl get svc istio-ingressgateway -n istio-system
```

您也可通过容器服务管理控制台，在左侧导航栏单击应用 > 服务，选择集群和Istio-system命名空间，查看istio-ingressgateway服务的IP地址。



6. 访问BookInfo首页，地址是http://{EXTERNAL-IP}/productpage。



多次刷新浏览器，将在 productpage 中看到评论的不同的版本，它们会按照 round robin（红星、黑星、没有星星）的方式展现，因为目前为止还没有使用 Istio 来控制版本的路由。

请求路由

BookInfo示例部署了三个版本的reviews服务，因此需要设置一个缺省路由。否则当多次访问该应用程序时，会发现有时输出会包含带星级的评价内容，有时又没有。出现该现象的原因是当没有为应用显式指定缺省路由时，Istio会将请求随机路由到该服务的所有可用版本上。

在使用 Istio 控制 Bookinfo 版本路由之前，你需要在目标规则中定义好可用的版本 。

运行以下命令为 Bookinfo 服务创建的默认的目标规则：

- 如果不需要启用双向TLS，请执行以下命令：

```
$ kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml
```

- 如果需要启用双向 TLS，请执行以下命令：

```
$ kubectl apply -f samples/bookinfo/networking/destination-rule-all-mtls.yaml
```

等待几秒钟，等待目标规则生效。你可以使用以下命令查看目标规则：

```
$ kubectl get destinationrules -o yaml
```

将所有微服务的缺省版本设置为v1

通过运行如下命令，将所有微服务的缺省版本设置为v1：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml
```

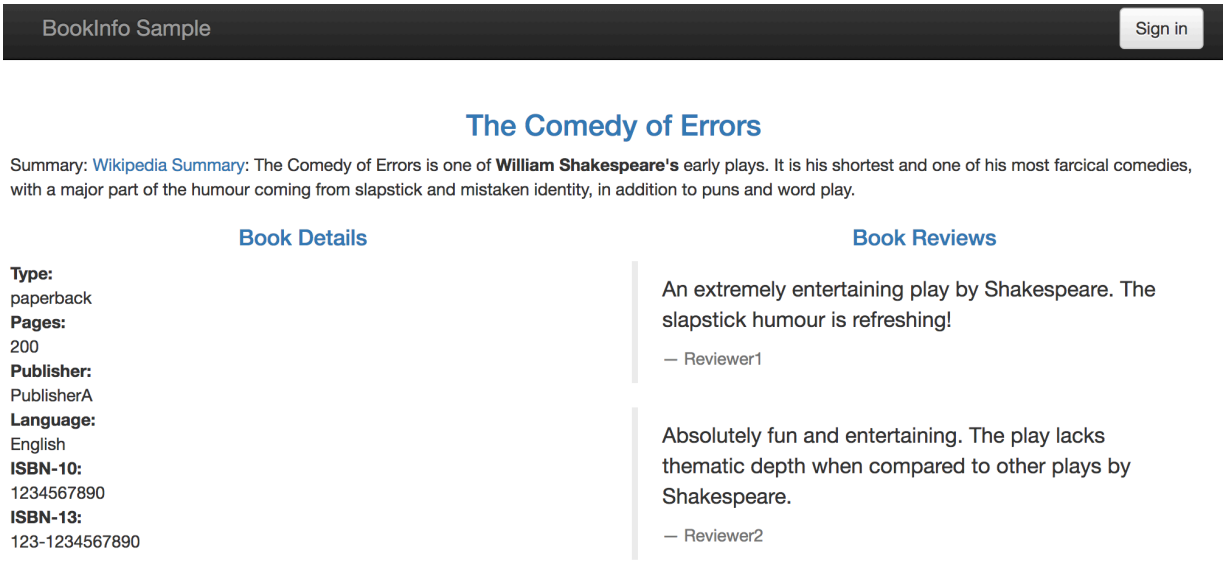
可以通过下面的命令来显示所有已创建的路由规则：

```
kubectl get virtualservices -o yaml
```

由于路由规则是通过异步方式分发到代理的，过一段时间后规则才会同步到所有pod上。因此需要等几秒钟后再尝试访问应用。

在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

可以看到BookInfo应用程序的productpage页面，显示的内容中不包含带星的评价信息，这是因为reviews:v1服务不会访问ratings服务。



将来自特定用户的请求路由到reviews:v2

通过运行如下命令，把来自测试用户"jason"的请求路由到reviews:v2，以启用ratings服务。

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews
-test-v2.yaml
```

可以通过如下命令确认规则是否创建：

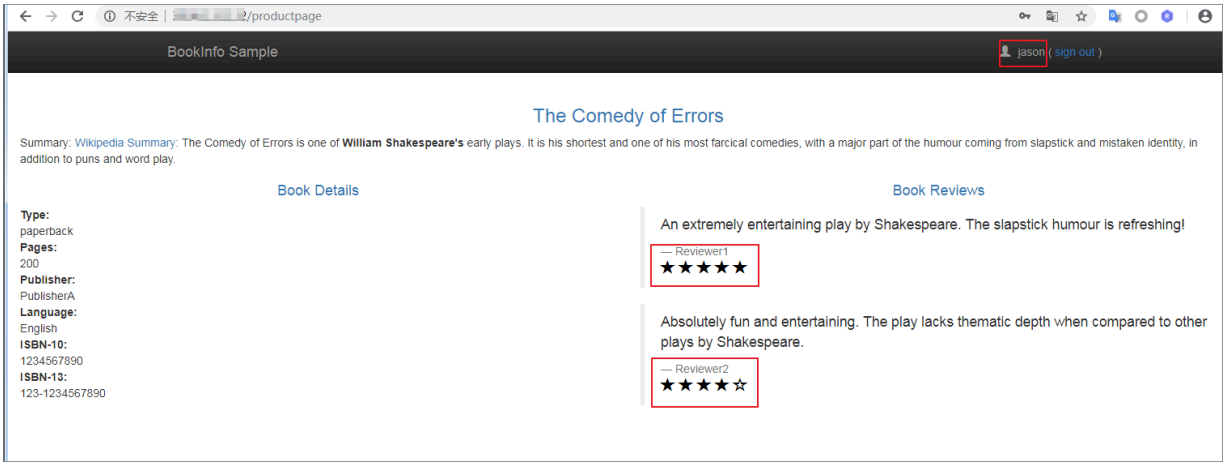
```
$ kubectl get virtualservice reviews -o yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，您可以在每条评价后面看到星级信息。

说明:

账号名称和密码都是`jason`。



说明:

本例中，首先使用Istio将100%的请求流量都路由到了BookInfo服务的v1版本；然后再设置了一条路由规则，路由规则基于请求的header（例如一个用户cookie）选择性地将特定的流量路由到了reviews服务的v2版本。

故障注入

为了测试BookInfo微服务应用的弹性，我们计划针对"jason"用户在`reviews:v2`和`ratings`服务之间 注入7秒的延迟 。由于 reviews:v2 服务针对调用ratings服务设置了10秒的超时，因此期望端到端的流程能无错持续。

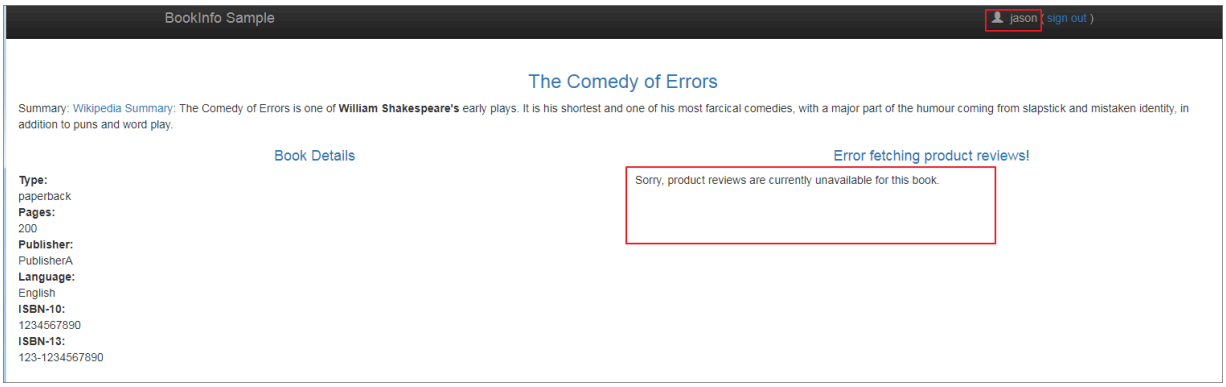
HTTP Delay

使用HTTP Delay创建一个故障注入规则，延迟来自用户jason的流量：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，您可看到如下画面：



说明:

整个review服务失败的原因：productpage和review服务之间的超时小于（3秒加上一次重试，总共6秒）review服务和rating服务之间的超时（10秒）。在由不同开发团队负责独立开发不同微服务的典型企业应用中，这类bug就会发生。Istio的故障注入规则有助于识别这些异常，而无需影响到最终用户。

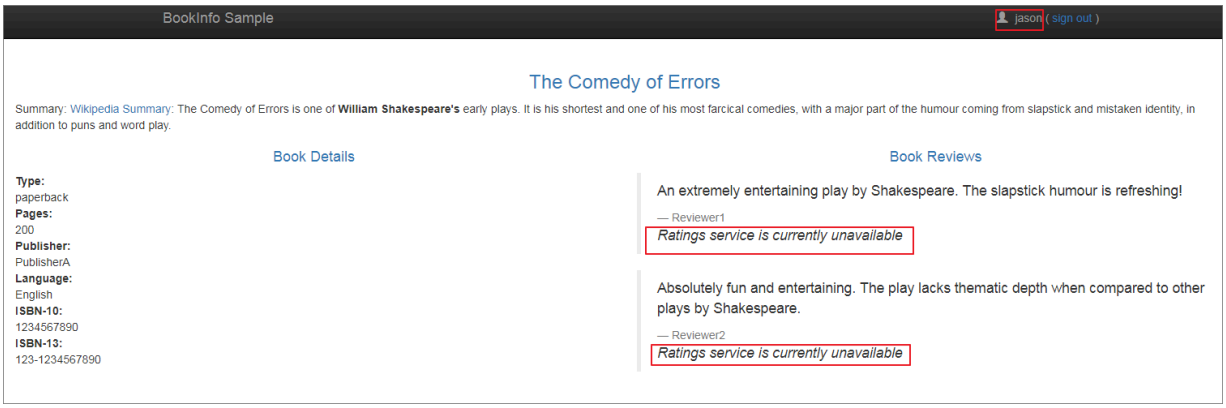
HTTP Abort

类似地，使用HTTP Abort创建一个故障注入规则：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings
-test-abort.yaml
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，应该可以看到如下画面：



流量转移

除了基于内容的路由，Istio还支持基于权重的路由规则。

首先，将所有微服务的缺省版本设置为v1：

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-all-
v1.yaml
```

其次，使用下面的命令把50%的流量从reviews:v1转移到reviews:v3:

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-
reviews-50-v3.yaml
```

在浏览器中多次刷新productpage页面，大约有50%的几率会看到页面中出现带红星的评价内容。

说明:

注意该方式和使用容器编排平台的部署特性来进行版本迁移是完全不同的。容器编排平台使用了实例scaling来对流量进行管理。而通过Istio，两个版本的reviews服务可以独立地进行扩容和缩容，并不会影响这两个版本服务之间的流量分发。

如果觉得 `reviews: v3` 微服务已经稳定，你可以通过以下命令， 将virtual service 100%的流量路由到 `reviews: v3`，从而实现一个灰度发布的功能。

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-
reviews-v3.yaml
```

总结

我们可以利用阿里云Kubernetes容器服务，快速搭建一套用于连接、管理以及安全化微服务的开放平台Istio，为应用引入和配置多个相关服务。本文通过一个官方示例来尝试了Istio 的流量路由、故障注入、流量转移等功能。欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

17.7.2 启用Istio CoreDNS

CoreDNS 及其Plugin扩展

CoreDNS 是一个 CNCF 下的孵化级项目，它的前身是 SkyDNS，主要目的是构建一个快速灵活的 DNS 服务器，让用户可以通过不同方式访问和使用 DNS 内的数据。基于 Caddy 服务器框架，CoreDNS 实现了一个插件链的架构，将大量逻辑抽象成插件 Plugin 的形式暴露给使用者，每个插件都执行 DNS 功能，例如 Kubernetes 的 DNS 服务发现、Prometheus 监控等。

除了插件化之外，CoreDNS 还有以下特性：

- 配置简单化： 引入表达力更强的 DSL，即 Corefile 形式的配置文件（也是基于 Caddy 框架开发）。
- 一体化的解决方案：区别于 kube-dns，CoreDNS 编译出来就是一个单独的二进制可执行文件，内置了 Cache、Backend Storage、Health Check 等功能，无需第三方组件来辅助实现其他功能，从而使得部署更方便，内存管理更为安全。

Corefile 介绍

Corefile 是 CoreDNS 的配置文件（源于 Caddy 框架的配置文件 Caddyfile），定义了以下内容：

- 服务器以什么协议监听在哪个端口（可以同时定义多个 server 监听不同端口）
- 服务器负责哪个 zone 的 authoritative DNS 解析
- 服务器将加载哪些插件

一个典型的 Corefile 格式如下所示。

```
ZONE:[PORT] {
    [PLUGIN] ...
```

```
}
```

Corefile 的各参数含义如下：

参数	含义
ZONE	用于定义服务器负责的 zone
PORT	监听端口，可选项，默认为 53
PLUGIN	定义服务器所要加载的插件plugin，每个 plugin 可以有多个参数

Plugin工作机制

当 CoreDNS 启动后，它将根据配置文件启动不同 server ，每台 server 都拥有自己的插件链。当有 DNS 请求时，它将依次经历如下 3 步逻辑：

- 如果有当前请求的 server 有多个 zone，将采用贪心原则选择最匹配的 zone。
- 一旦找到匹配的 server，按照 plugin.cfg 定义的顺序执行插件链上的插件。
- 每个插件将判断当前请求是否应该处理，将有以下几种可能：
  - 请求被当前插件处理

    插件将生成对应的响应并回给客户端，此时请求结束，下一个插件将不会被调用，如 whoami 插件。
  - 请求不被当前插件处理

    直接调用下一个插件。如果最后一个插件执行错误，服务器返回 SERVFAIL 响应。
  - 请求被当前插件以 Fallthrough 形式处理

    如果请求在该插件处理过程中有可能将跳转至下一个插件，该过程称为 fallthrough，并以关键字 fallthrough 来决定是否允许此项操作，例如 host 插件，当查询域名未位于 `/etc/hosts`，则调用下一个插件。

    ■ 请求在处理过程被携带 Hint。

    ■ 请求被插件处理，并在其响应中添加了某些信息（hint）后继续交由下一个插件处理。这些额外的信息将组成对客户端的最终响应，如 metric 插件。

CoreDNS 与 Kubernetes

从 Kubernetes 1.11开始，CoreDNS 作为 Kubernetes 的 DNS 插件进入GA状态，Kubernetes 推荐使用 CoreDNS 作为集群内的 DNS 服务。在阿里云容器服务 Kubernetes 1.11.5 及更新版本中默认安装使用 CoreDNS 作为 DNS 服务。

执行`kubectl -n kube-system get configmap coredns -oyaml`命令可以查看 CoreDNS 配置信息。

```
apiVersion: v1
data:
  Corefile: |-
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        proxy . /etc/resolv.conf
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2018-12-28T07:28:34Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "2453430"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: 2f3241d5-0a72-11e9-99f1-00163e105bdf
```

配置文件各参数的含义如下：

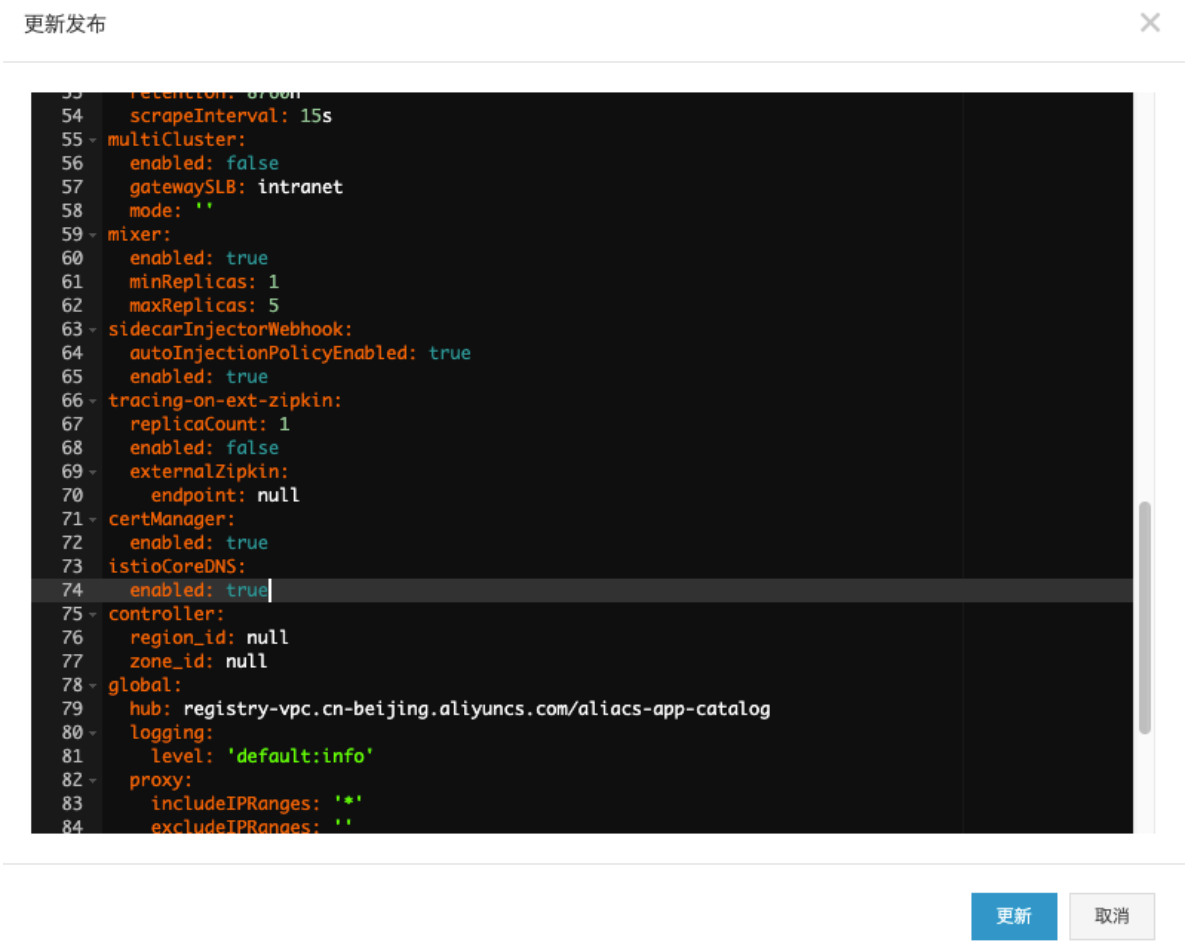
名称	含义
errors	错误会被记录到标准输出
health	健康状况，可以通过*http://localhost:8080/health*查看
kubernetes	根据服务的IP响应DNS查询请求，Cluster Domain默认为cluster.local
prometheus	可以通过*http://localhost:9153/metrics*获取prometheus格式的监控数据
proxy	本地无法解析后，向上级地址进行查询，默认使用宿主机的*/etc/resolv.conf*配置
cache	缓存时间

启用Istio CoreDNS

在Istio默认安装的配置中，没有启用Istio CoreDNS进行DNS解析。可以通过更新配置方式，在Istio安装之后重新启用Istio CoreDNS。

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服务网格 > Istio管理，进入部署Istio页面。

3. 单击右上角的更新按钮，弹出的更新发布 页面。



4. 将`istioCoreDNS`下`enabled`值修改为true，单击更新。

   如果没有以下参数，您可以在更新发布页面第一行添加，并保持Yaml的格式。

```
  istioCoreDNS:
    enabled: ture
```

查看集群 CoreDNS 配置

1. 可以通过执行`kubectl get svc istiocoredns -n istio-system`命令，来获取`istiocoredns`服务的Cluster IP。

```
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
     AGE
istiocoredns   ClusterIP   172.19.XX.XX    <none>        53/UDP,53/
TCP    44m
```

   查看集群的coredns的配置项configmap，可以看到`istiocoredns`服务作为`*.global`域的上游DNS服务，在此配置项configmap中已经被添加，如下所示。

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |-
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        proxy . /etc/resolv.conf
        cache 30
        reload
    }
    global:53 {
         errors
         cache 30
         proxy . {cluster IP of this istiocoredns service}
    }
```

2. 集群的 CoreDNS 容器会重新加载这些配置内容，可以通过如下命令行查看加载日志。

```
# kubectl get -n kube-system pod | grep coredns
coredns-8645f4b4c6-5frkg                                  1/1
  Running   0          20h
coredns-8645f4b4c6-lj59t                                  1/1
  Running   0          20h

# kubectl logs -f -n kube-system coredns-8645f4b4c6-5frkg
....
2019/01/08 05:06:47 [INFO] Reloading
2019/01/08 05:06:47 [INFO] plugin/reload: Running configuration MD5
 = 05514b3e44bcf4ea805c87cc6cd56c07
2019/01/08 05:06:47 [INFO] Reloading complete

# kubectl logs -f -n kube-system coredns-8645f4b4c6-lj59t
....
2019/01/08 05:06:31 [INFO] Reloading
2019/01/08 05:06:32 [INFO] plugin/reload: Running configuration MD5
 = 05514b3e44bcf4ea805c87cc6cd56c07
2019/01/08 05:06:32 [INFO] Reloading complete
```

创建 ServiceEntry 验证 DNS 解析

ServiceEntry 能够在 Istio 内部的服务注册表中加入额外的条目，从而让网格中自动发现的服务能够访问和路由到这些手工加入的服务。ServiceEntry 描述了服务的属性，即包括 DNS 名称、虚拟 IP、端口、协议以及端点等。这类服务可能是网格外的 API，或者是处于网格内部但却不存在于平台的服务注册表中的条目。例如，需要和 Kubernetes 服务沟通的一组基于虚拟机VM提供的服务。

本例中，使用了通配符定义 hosts，并指定了address，如下所示。

1. #unique_325。

2. 创建并拷贝以下内容到 *external-svc-serviceentry.yaml* 文件中，并执行`kubectl apply -f external-svc-serviceentry.yaml`命令，创建 ServiceEntry。

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-svc-serviceentry
spec:
  hosts:
  - '*.test.global'
  addresses:
  - 127.255.0.2
  ports:
  - number: 8080
    name: http1
    protocol: HTTP
  location: MESH_INTERNAL
  resolution: DNS
  endpoints:
  - address: 47.111.38.80
    ports:
      http1: 15443
```

3. 执行如下命令查看容器`istiocoredns`，可以看到上述 ServiceEntry 的域名映射关系已经被加载。

```
# kubectl get po -n istio-system | grep istiocoredns
istiocoredns-cdc56b67-ngtkr                  2/2     Running   0
     1h

# kubectl logs --tail 2 -n istio-system istiocoredns-cdc56b67-ngtkr
 -c istio-coredns-plugin
2019-01-08T05:27:22.897845Z    info    Have 1 service entries
2019-01-08T05:27:22.897888Z    info    adding DNS mapping: .test.
global.->[127.255.XX.XX]
```

4. 执行如下命令，使用镜像 tutum/dnsutils 创建测试容器。

```
# kubectl run dnsutils -it --image=tutum/dnsutils bash
```

5. 进入容器命令行后，执行 dig 查看相应的域名解析。

```
# dig +short 172.19.XX.XX (DNS服务的ClusterIP) A service1.test.global
127.255.XX.XX
# dig +short 172.19.XX.XX (DNS服务的ClusterIP) A service2.test.global
127.255.XX.XX
```

17.7.3 使用阿里云容器服务部署Bookinfo示例

本文介绍了使用阿里云容器服务部署运行bookinfo示例的过程。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见#unique_73，且集群版本为v1.12及以上。
- 您已经创建一个命名空间，参见#unique_327。

- 您已经成功部署Istio，参见部署Istio。
- 您已经启用 Sidecar 自动注入功能，参见设置 Sidecar 自动注入。

Bookinfo应用介绍

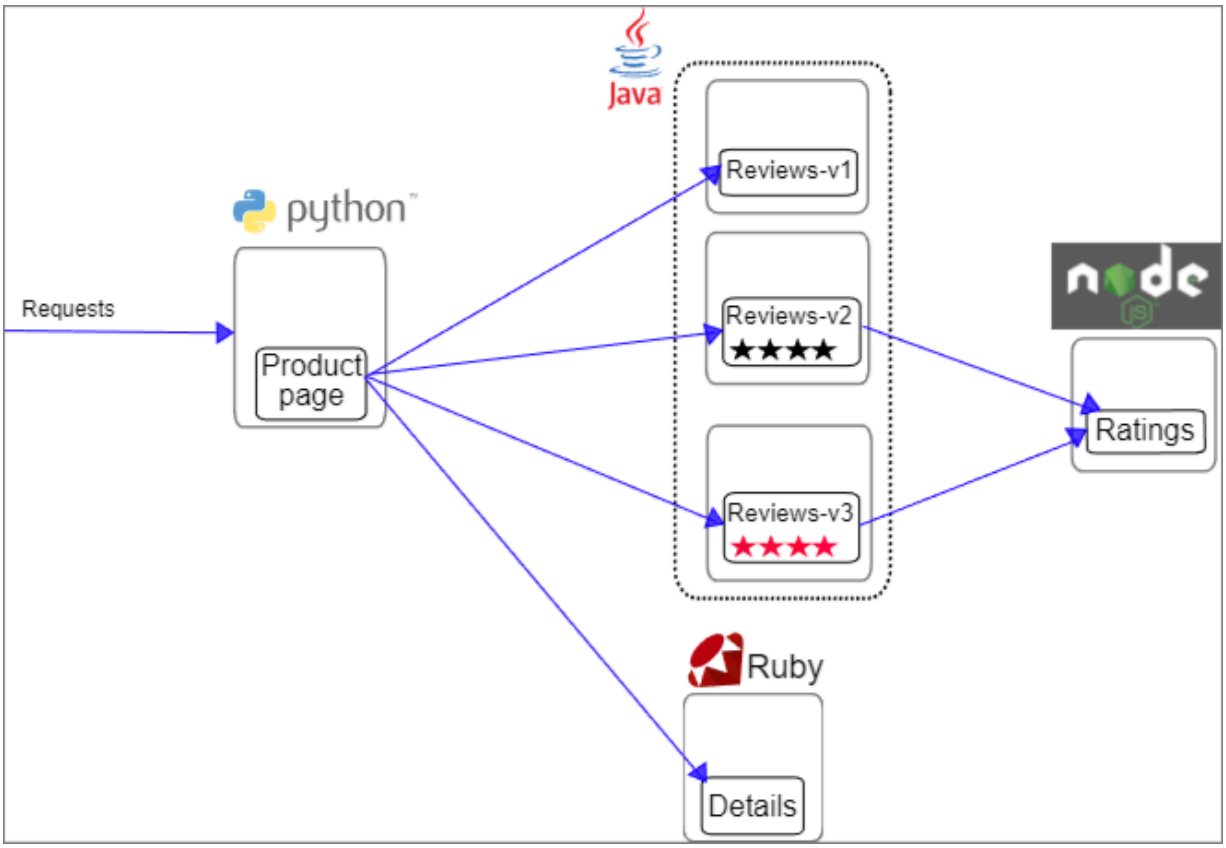
Bookinfo是一个样例应用，用来演示多种 Istio 特性，它由四个单独的微服务构成：

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

`reviews` 微服务有 3 个版本：

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

下图展示了这个应用的端到端架构。



部署Bookinfo应用

本例中，您已经完成部署Istio，并且istio-test命名空间中，开启Sidecar自动注入。

完成后，您可以看到istio-test命名空间下新增标签`istio-injection:enabled`。

您可以使用控制台分别部署productpage，detail，rating及reviews服务的两个版本（v1、v2）。

本例中以部署productpage服务为例进行介绍。

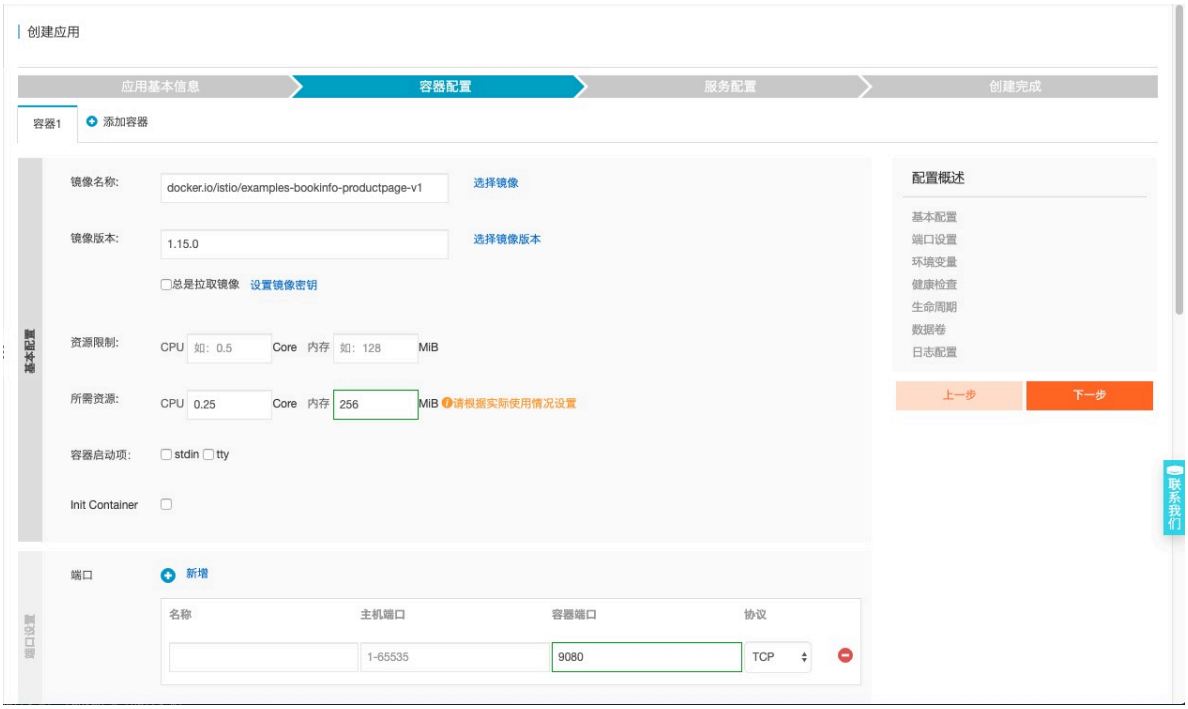
1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的服务网格 > 虚拟服务，然后单击页面右上角的创建。

3. 设置应用名称、应用版本 、 集群、命名空间和副本数量，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。

   本例中使用 istio-test 命名空间。

4. 设置容器配置。

   设置镜像名称、镜像版本、所需资源，容器端口等配置信息。



说明:

本示例中，界面参数取值参考如下：

- 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。您还可以填写私有 registry。本例中为*docker.io/istio/example-bookinfo-productpage-v1* 。

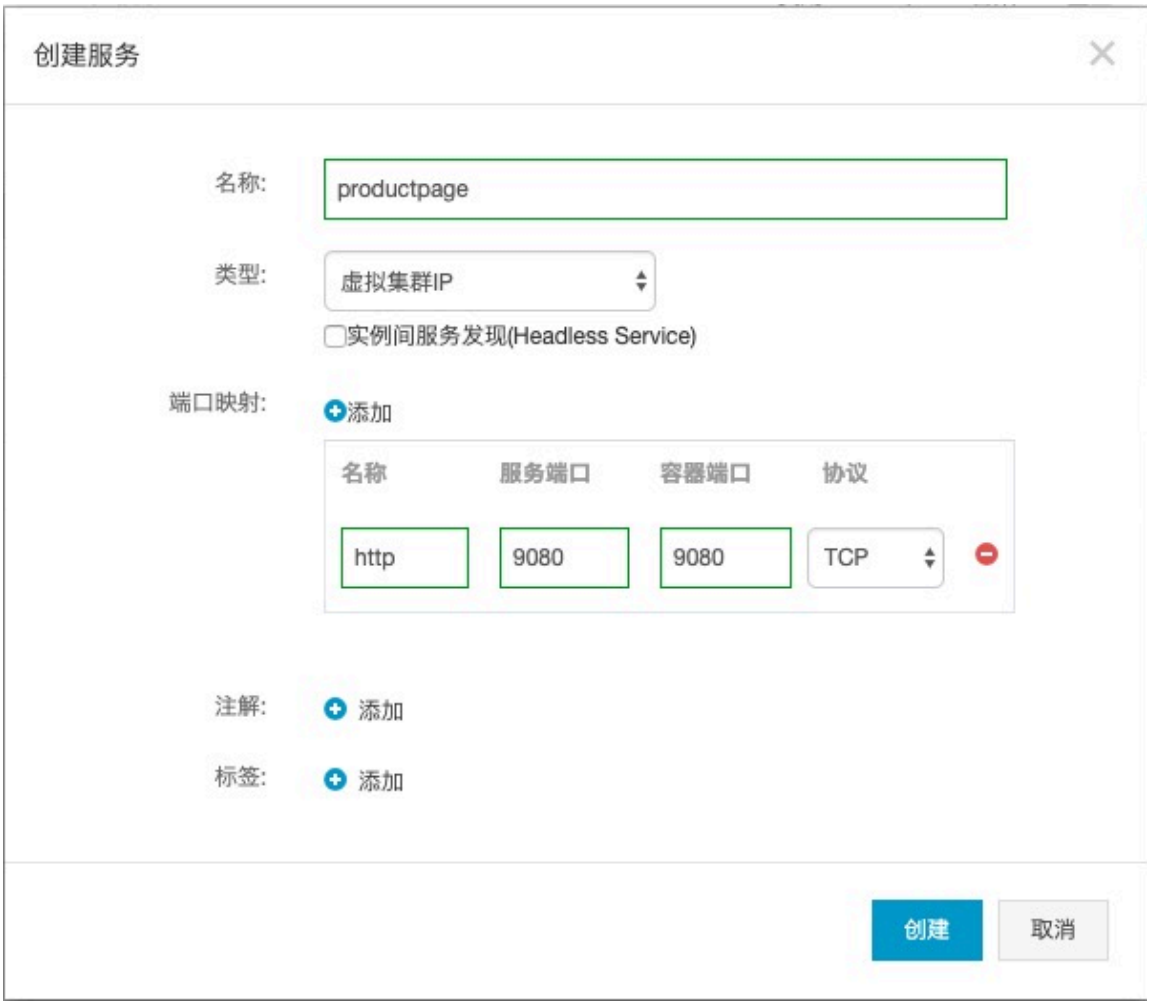
- 镜像版本：您可以单击选择镜像版本选择镜像的版本。若不指定，默认为 latest。本例中为1.15.0。
- 所需资源：即为该应用预留资源额度，包括 CPU 和 内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争占资源，导致应用不可用。本例中CPU为0.25Core，内存为256MiB。
- 端口映射：您需要添加主机端口和容器端口，支持 TCP/UDP 协议。本例中，容器端口为9080，协议为TCP。

界面其他参数详细信息请参见镜像创建无状态Deployment应用。

5. 完成容器配置后，单击下一步。

6. 进行服务配置。

a) 在服务栏单击创建，在弹出的对话框中进行配置。



- 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，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_328。
- 标签：您可为该服务添加一个标签，标识该服务。

b) 单击创建。

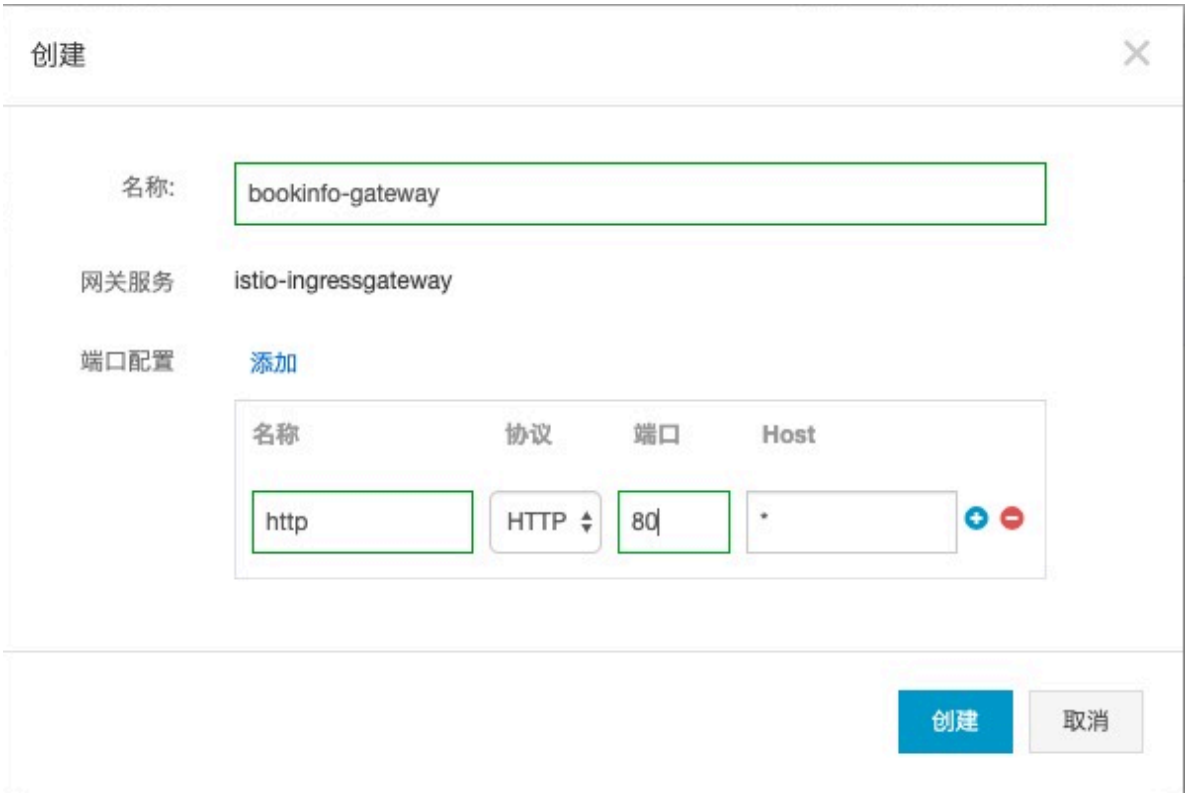
您可以参考上述步骤分别创建details，ratings，reviews（v1）这三个服务。

创建完成后，等待所有的service和 pod 都已经正常启动。

创建服务网关

1. 在 Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > 服务网关，进入服务网关页面。
2. 选择所需的集群及命名空间后，单击右上角新建。

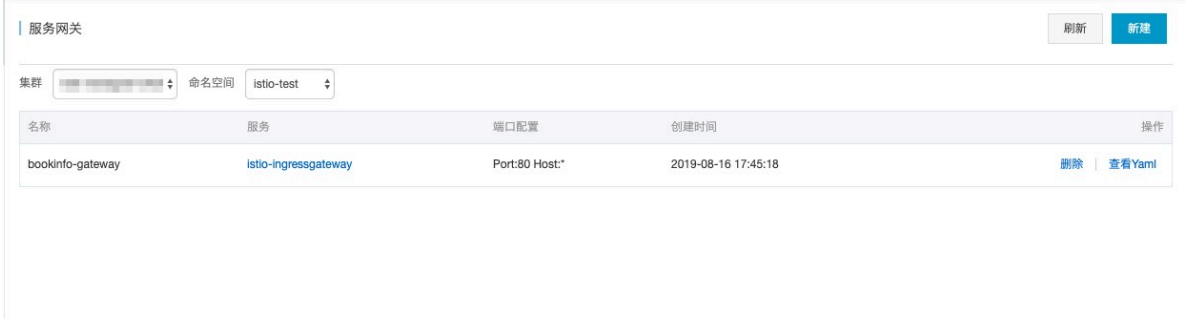
3. 填写如下信息，单击创建，部署服务网关。



参数说明：

- 名称：新创建的服务网关的名称。
- 服务网关：服务代理网关。默认为istio-ingressgateway。代理网关必须和新创建的网关部署在同一命名空间。
- 端口配置：单击添加，可以增加多个端口的配置。
  - 名称：端口的名称
  - 协议：HTTP/TCP
  - 端口：1-65535
  - Host：支持多个host

创建完成后，您可以在服务网关列表中，看到新创建的服务网关。

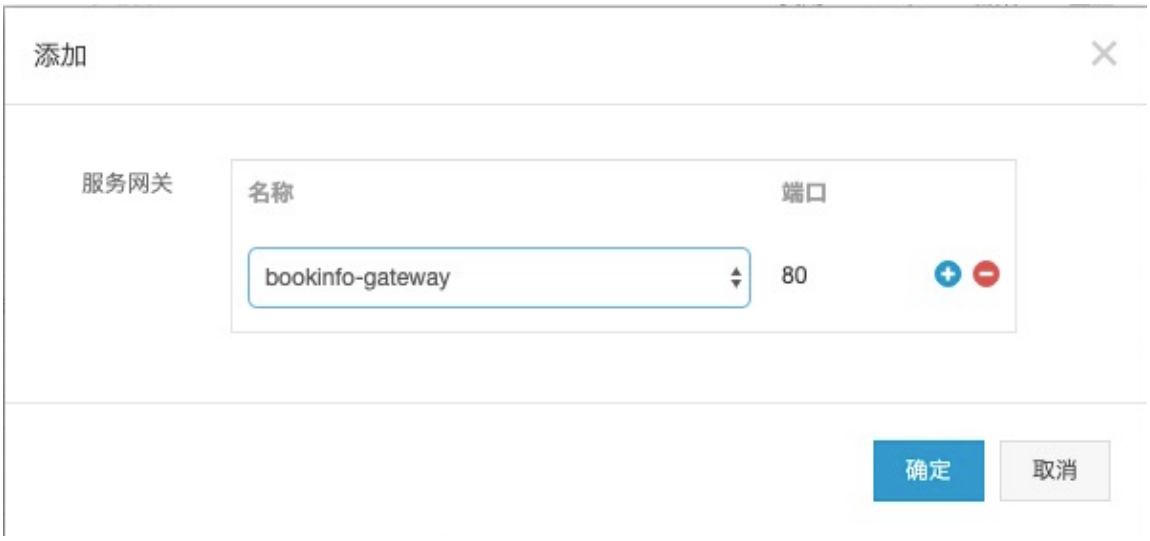


4. 将上述服务绑定到Bookinfo-gateway网关，请参见绑定虚拟服务。

   a) 在 Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > 虚拟服务，可以看到虚拟服务（Virtual Service）列表。

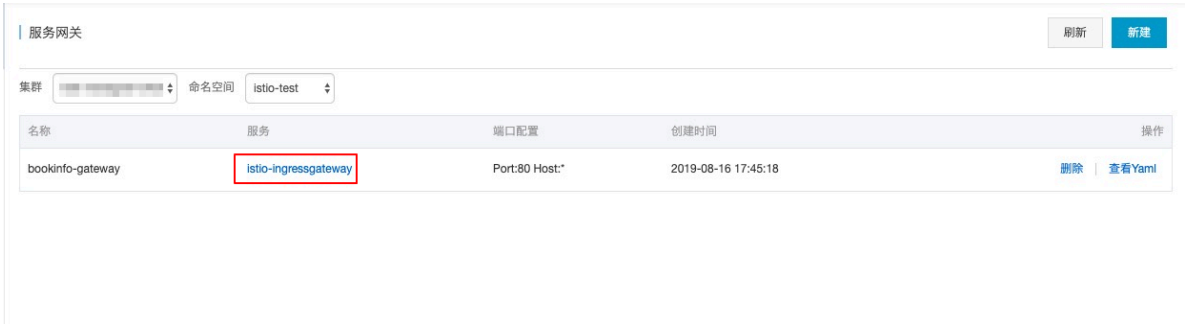
> **说明:**
> 未绑定网关的虚拟服务在服务网关列会显示绑定按钮。

   b) 在目标虚拟服务的服务网关列单击绑定，弹出添加页面。

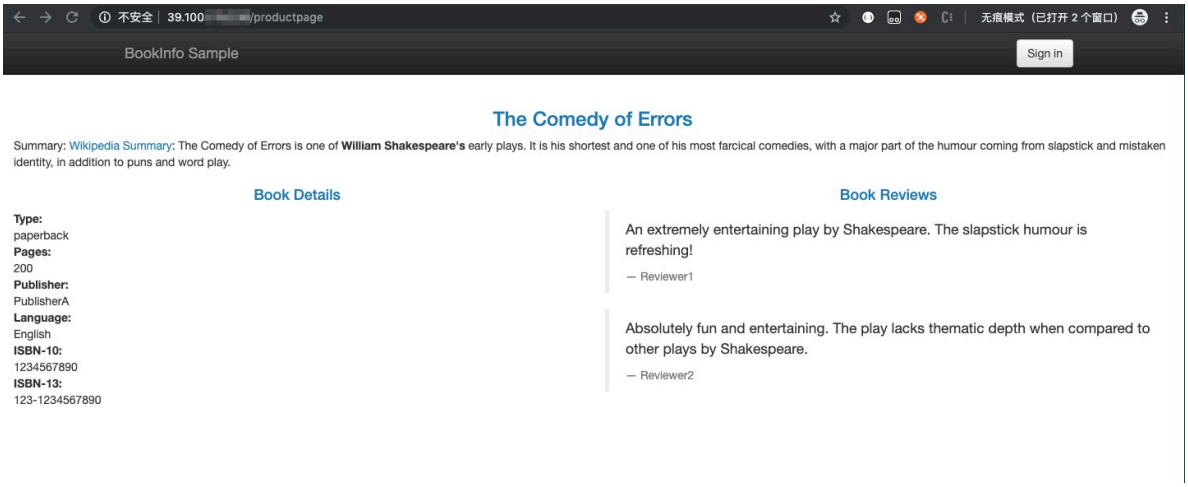


   c) 在服务网关中选择对应的网关名称，单击确定，即完成虚拟服务的绑定。

5. 在服务网关列表中，在目标服务网关对应的服务列中单击istio-ingressgateway，并输入##/productpage。



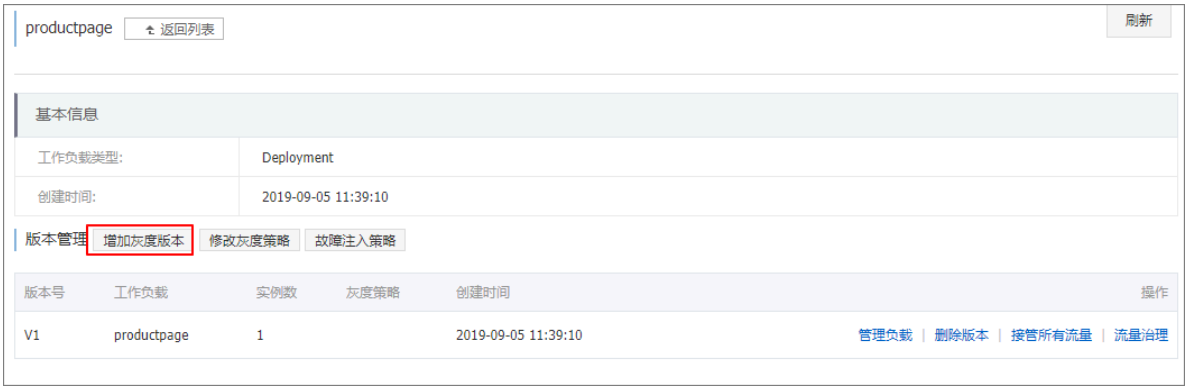
可以看到如下页面。



部署reviews-v2 版本

1. 在 Kubernetes 菜单下， 单击左侧导航栏中的服务网格 > 虚拟服务，可以看到虚拟服务（Virtual Service）列表。
2. 在目标服务的操作列单击管理，进入服务详情页面。

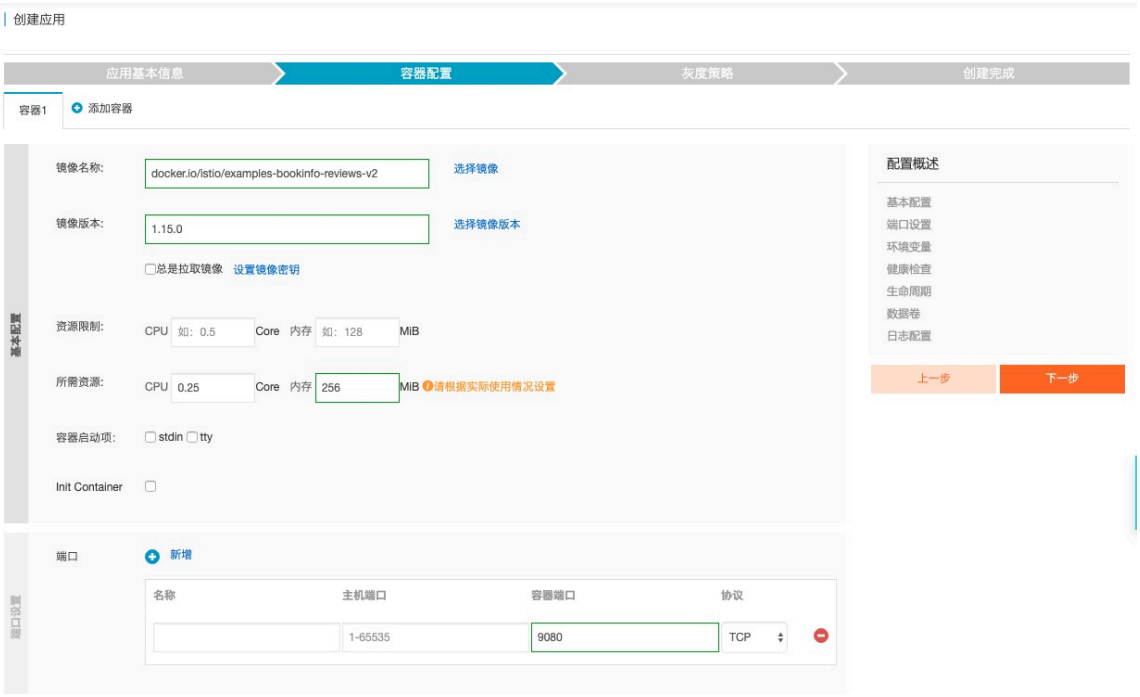
3. 单击增加灰度版本。



4. 进入创建应用界面。您可以根据需要配置界面参数。

a) 在应用基本信息页面，填写应用版本和副本数量后，单击下一步。

b) 在容器配置页面，设置镜像名称、镜像版本、所需资源，容器端口等配置信息后，单击下一步。



说明:

本示例中，界面参数取值参考如下：

- 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。您还可以填写私有 registry。本例中为*docker.io/istio/example-bookinfo-productpage-v2* 。
- 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。本例中为1.15.0。

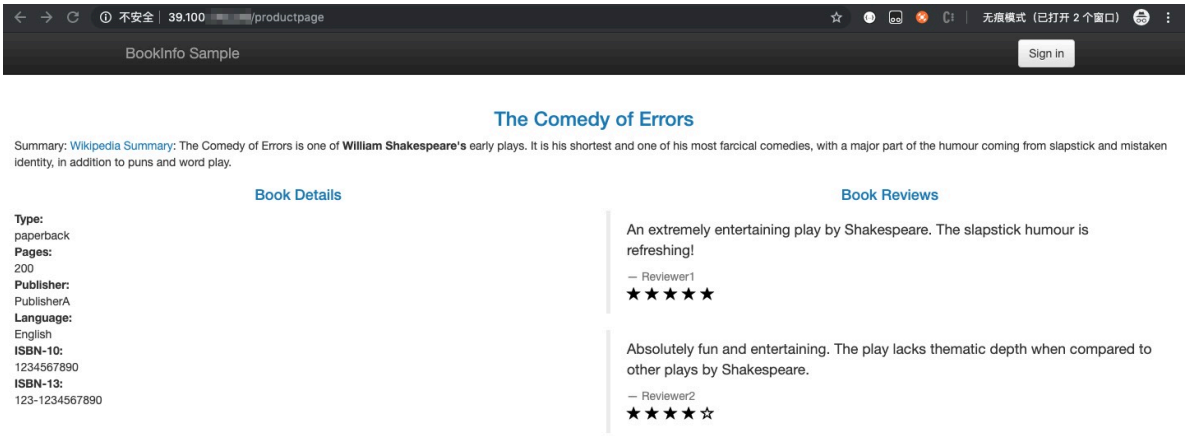
- 所需资源：即为该应用预留资源额度，包括 CPU 和 内存两种资源，即容器独占该资源，防止因资源不足而被其他服务或进程争占资源，导致应用不可用。本例中CPU为0.25Core，内存为256MiB。
- 端口映射：您需要添加主机端口和容器端口，支持 TCP/UDP 协议。本例中，容器端口为9080，协议为TCP。

界面其他参数详细信息请参见镜像创建无状态Deployment应用。

c) 在灰度策略页面，策略类型选择基于流量比例发布，流量比例中，V2流量比例设置为50%后，单击创建。

5. 待pod启动完成后，输入##/productpage，会看到v2版本的界面。

由于v1、v2流量比例各50%，可能要多刷新几次界面。



预期结果

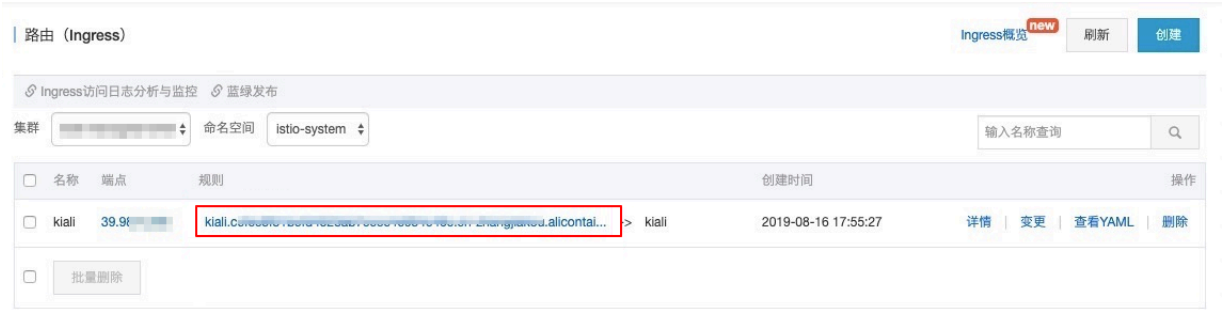
您可以使用压测工具（例如siege）模拟多次访问，再通过kiali观察流量拓扑。

在部署Istio时已经默认安装了kiali，您只需要新建一个Ingress就可以方便的进行访问。请参见#unique_329。

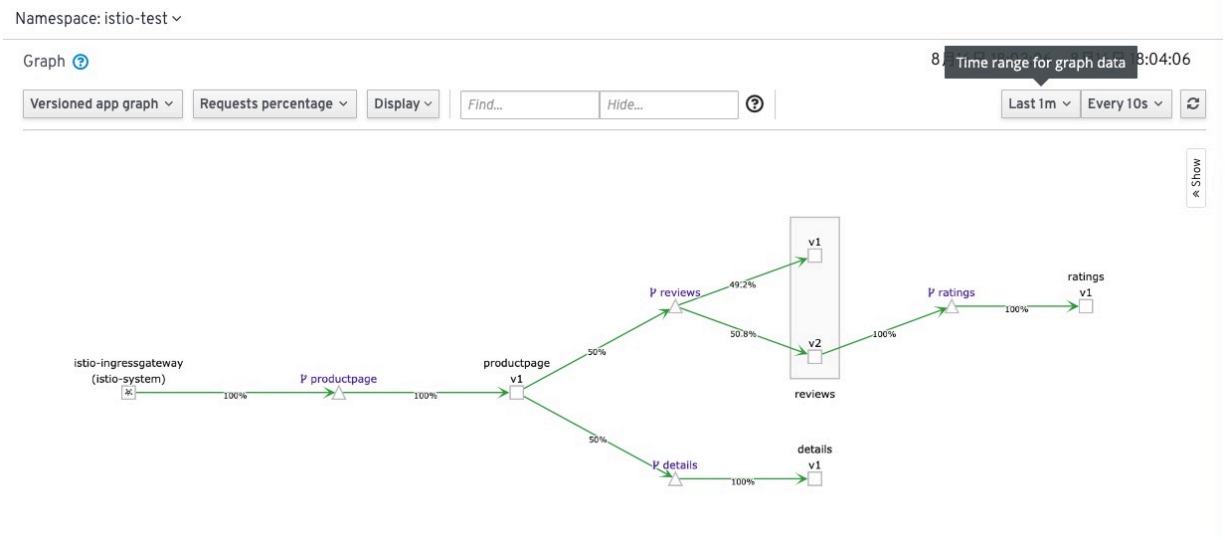
说明:

本例中，目标集群选择已部署Istio的集群，命名空间选择istio-system。

创建完成后，您可以在路由列表中，看到kiali的访问地址。



单击kiali的访问地址，打开Kiali控制台，可以看到如下调用效果。



说明:

用户名密码均默认为admin。

18 Knative 管理

18.1 概述

Knative 是一款基于 Kubernetes 的 Serverless 框架。其目标是制定云原生、跨平台的 Serverless 编排标准。Knative 通过整合容器构建（或者函数）、工作负载管理（动态扩缩）以及事件模型这三者来实现的这一 Serverless 标准。

Knative 体系架构

在 Knative 体系架构下各个角色的协作关系：

- Developers

  Serverless 服务的开发人员可以直接使用原生的 Kubernetes API 基于 Knative 部署 Serverless 服务。

- Contributors

  主要是指社区的贡献者。

- Operators

  Knative 可以被集成到任何支持的环境中，例如，云厂商、或者企业内部。目前 Knative 是基于 Kubernetes 来实现的，有 Kubernetes 的地方就可以部署 Knative。

- Users

  终端用户通过 Istio 网关访问服务，或者通过事件系统触发 Knative 中的 Serverless 服务。

Knative 核心组件

作为一个通用的 Serverless 框架 Knative 由三个核心组件组成：

- Build：提供从源码到镜像的通用构建能力
- Eventing：提供了事件的接入、触发等一整套事件管理的能力
- Serving：管理 Serverless 工作负载，可以和事件很好的结合并且提供了基于请求驱动的自动扩缩的能力，而且在没有服务需要处理的时候可以缩容到零个实例

Build 组件主要负责从代码仓库获取源码并编译成镜像和推送到镜像仓库。并且所有这些操作都是在 Kubernetes Pod 中进行的。

Eventing 组件针对 Serverless 事件驱动模式做了一套完整的设计。包括外部事件源的接入、事件注册和订阅、以及对事件的过滤等功能。事件模型可以有效的解耦生产者和消费者的依赖关系。生产者可以在消费者启动之前产生事件，消费者也可以在生产者启动之前监听事件。

Serving 组件的职责是管理工作负载以对外提供服务。对于 Knative Serving 组件最重要的特性就是自动伸缩的能力，目前伸缩边界支持从 0 到无限大。Serving 还有一个比较重要的功能就是灰度发布能力。

Knative add-on 组件

Knative 支持第三方 add-on 组件。当前支持 GitHub add-on组件，用于提供 GitHub 事件源支持。

18.2 部署 Knative

前提条件

- 您已经成功创建一个Kubernetes 集群，且集群中Worker节点的数量大于等于3个。参见#unique_73。
- 您已经成功部署 Istio，参见#unique_318。

  如果需要实现 Tracing 分布式追踪服务，需要额外设置 Istio 信息，请参考#unique_333。
- 仅支持 Kubernetes 版本 1.10 及以上的集群， 只支持标准 Kubernetes 托管版集群以及标准专有 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，选择Knative > 组件管理，单击右上角的一键部署。

3. 对模板进行相关配置，完成配置后单击部署。

- Bulid 组件：提供从源码到镜像的通用构建能力。
- Serving 组件：管理 Serverless 工作负载，可以和事件很好的结合并且提供了基于请求驱动的自动扩缩的能力，而且在没有服务需要处理的时候可以缩容到零个实例。
- Eventing 组件：提供了事件的接入、触发等一整套事件管理的能力。

执行结果

部署完成后，可以看到部署结果信息。

- 您可以单击进入组件管理，查看组件信息。
- 您可以单击进入应用管理，查看应用相关操作。

18.3 部署组件

本文以Eventing为例进行介绍如何部署组件。

背景信息

Knative 部署时，如果您未勾选需要部署的组件，您可以按照本文操作。

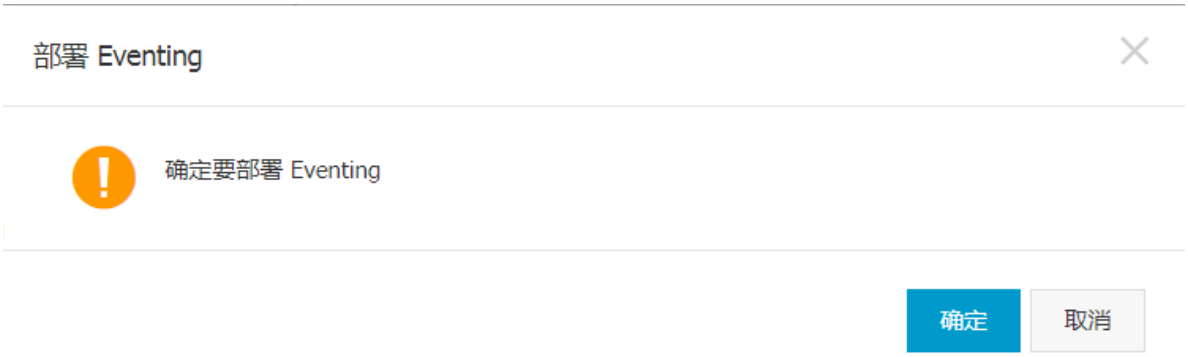
前提条件

- 您已经成功创建一个Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见 #unique_335。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 组件管理，进入knative 组件管理页面。
3. 在状态为未部署的组件的右侧，单击部署。

4. 弹出部署 Eventing页面，单击确定。



执行结果

部署完成后，单击Knative > 组件管理，在Knative 组件管理页面，可以看到当前组件状态为已部署。

18.4 创建 Knative 服务

本文主要为您介绍如何创建 Knative 服务。

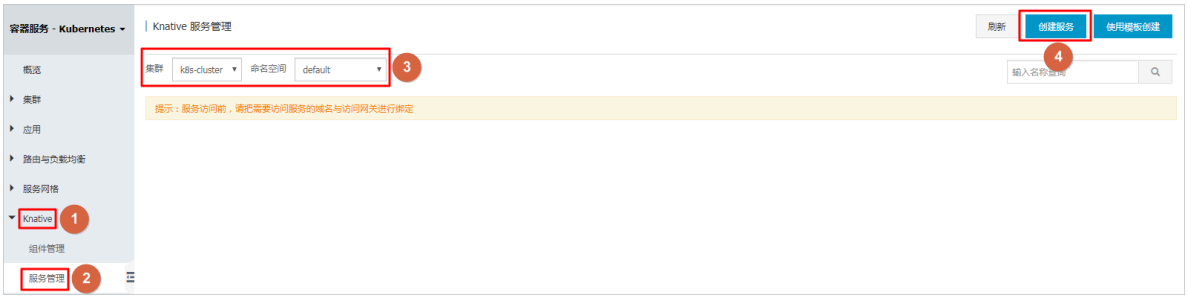
背景信息

- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见 #unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。

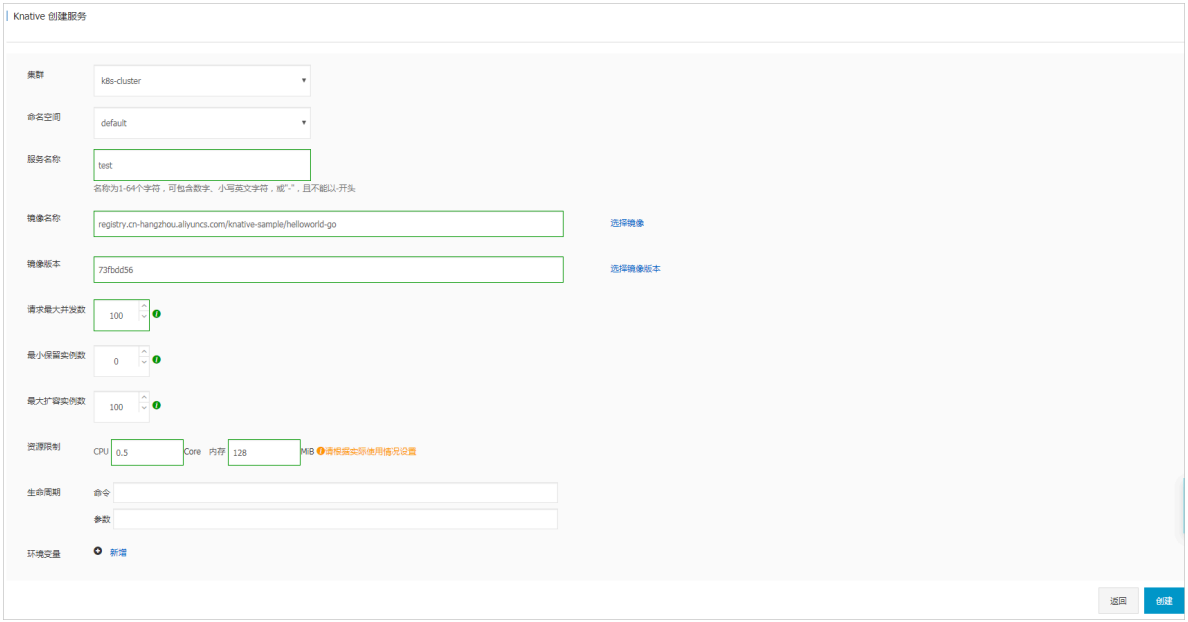
操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。

3. 单击右上角的创建服务。



4. 设置集群名称、命名空间、服务名称，选择所要使用的镜像和镜像版本等配置信息。



- 集群：选择需要创建服务的集群。
- 命名空间：选择该服务所属的命名空间。
- 服务名称：自定义该服务的名称。
- 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。您还可以填写私有 registry。填写的格式为*domainname/namespace/imagename:tag*。本例中为*registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go*。
- 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。本例中为73fbdd56。
- 请求最大并发数：容器允许的最大请求并发数。默认0，表示不限制并发数。
- 最小保留实例数：在无访问请求的情况下，最小保留的运行实例数。设置为0时，表示没有访问请求时，实例缩为 0。
- 最大扩容实例数：允许扩容出来的最多实例个数。
- 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 。
- 生命周期：包含命令（Command） 和参数（Args）。
  - 如果均不配置，则使用镜像默认的命令和参数。
  - 如果仅配置参数，则使用镜像默认的命令及新配置的参数。
  - 如果均配置，则会覆盖镜像默认的配置。
- 环境变量：支持通过键值对的形式配置环境变量。

5. 单击创建。

   创建完成后，您可以在Knative 服务管理列表中，看到新创建的服务。

预期结果

服务部署完成后，通过绑定Host域名与访问网关，然后可以直接访问服务URL。

1. 在 Kubernetes 菜单下，单击左侧导航栏的Knative > 组件管理，进入组件管理页面。可以看到访问网关。

2. 将访问网关地址与需要访问的域名进行Host绑定，在Hosts文件中添加绑定信息。

```
##+ ##
```

   样例如下：

```
47.95.XX.XX test.default.example.com
```

3. 完成Host绑定后，可通过域名*http://test.default.example.com*直接对服务进行访问。

18.5 创建修订版本

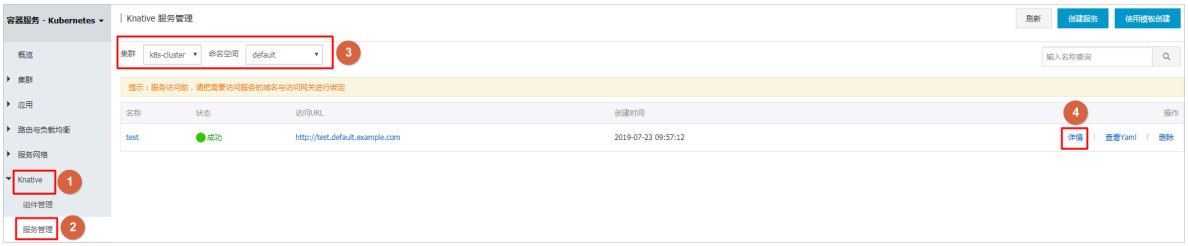
本文主要为您介绍如何创建修订版本。

背景信息

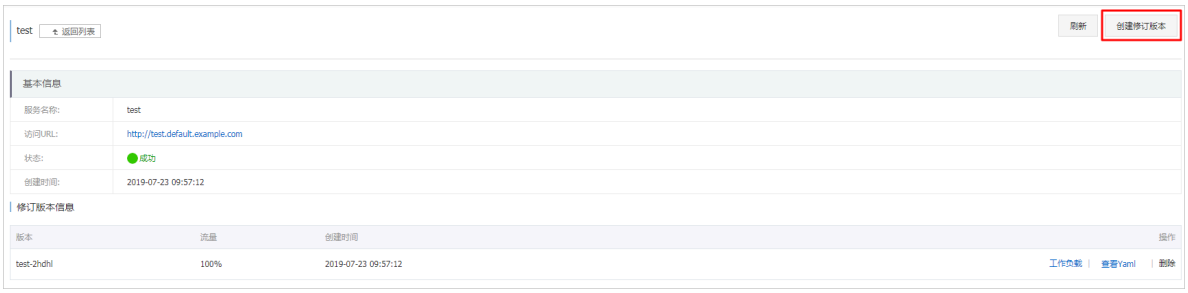
- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见 #unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。
- 您已经成功创建 Knative 服务，参见创建 Knative 服务。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。
3. 选择目标集群及命名空间，单击操作列的详情。

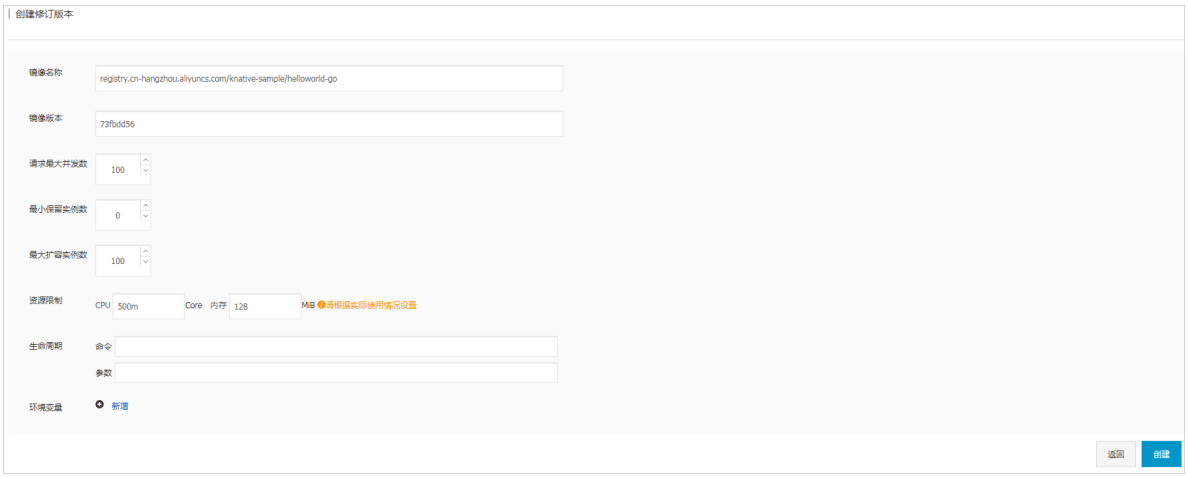


4. 进入目标服务详情页面，单击右上角的创建修订版本。



5. 进入创建修订版本页面，设置界面参数。

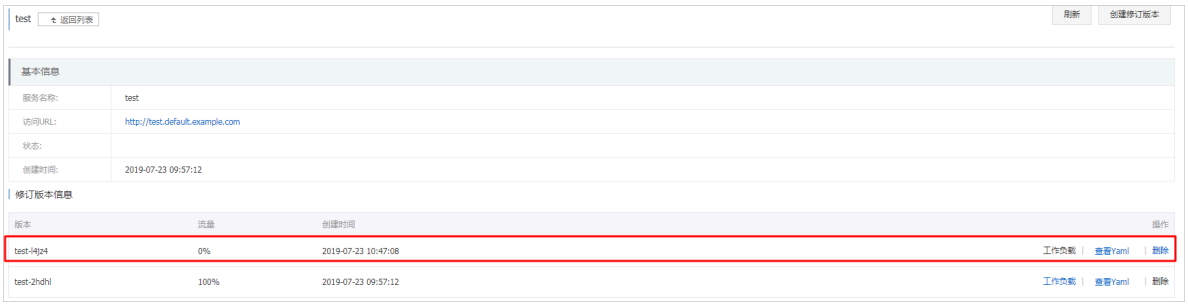
   该界面参数默认为创建服务时的参数取值。



- 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。您还可以填写私有 registry。填写的格式为*domainname/namespace/imagename:tag*。本例中为*registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go*。
- 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。本例中为73fbdd56。
- 请求最大并发数：容器允许的最大请求并发数。默认为0，表示不限制并发数。
- 最小保留实例数：在无访问请求的情况下，最小保留的运行实例数。设置为0时，表示没有访问请求时，实例缩为 0。
- 最大扩容实例数：允许扩容出来的最多实例个数。
- 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 。
- 生命周期：包含命令（Command） 和参数（Args）。
  - 如果均不配置，则使用镜像默认的命令和参数。
  - 如果仅配置参数，则使用镜像默认的命令及新配置的参数。
  - 如果均配置，则会覆盖镜像默认的配置。
- 环境变量：支持通过键值对的形式配置环境变量。

6. 单击创建，创建修订版本。

   此时可以在修订版本信息区域，可以看到新创建的修订版本信息。



18.6 部署 Serving Hello World 应用

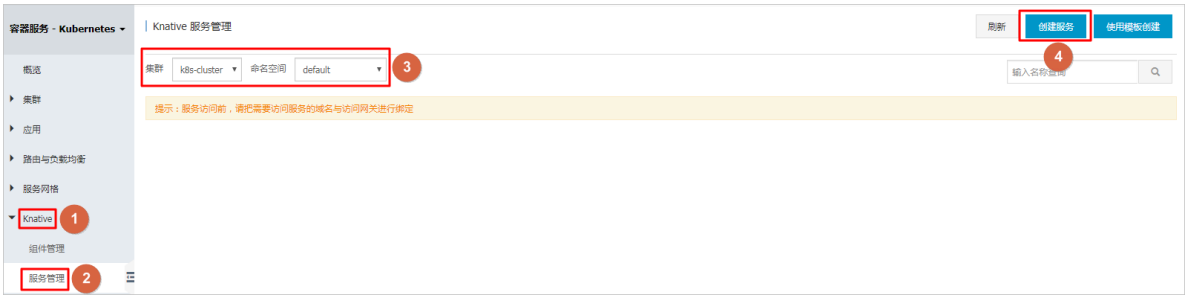
本文以 Hello World 示例为您介绍 Knative 。

背景信息

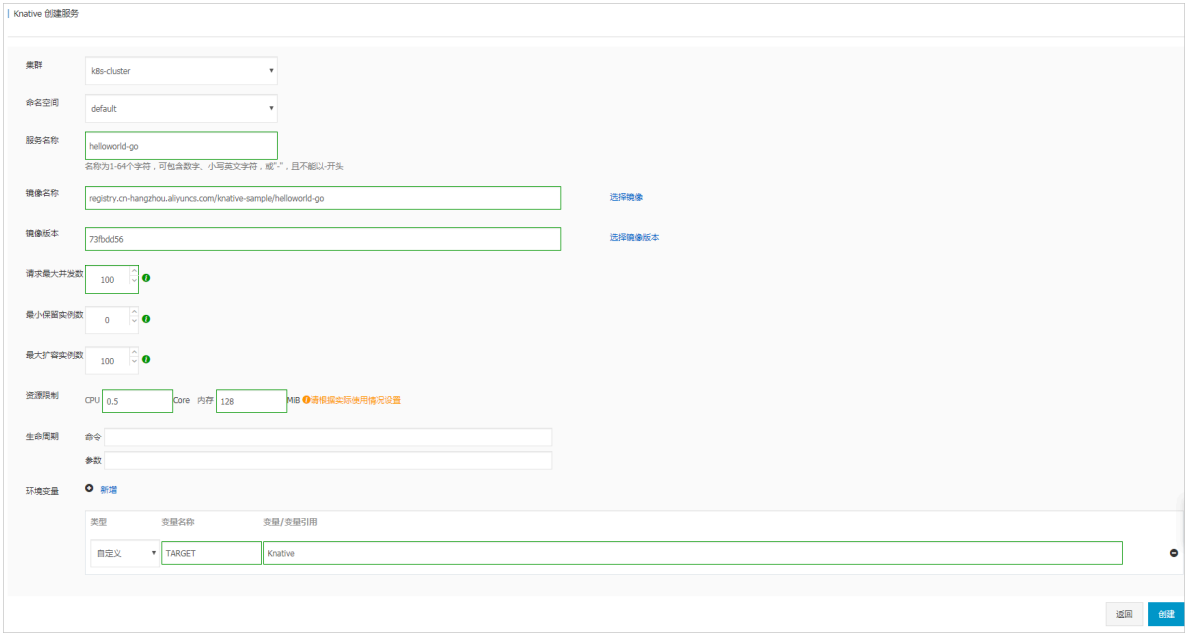
- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见 #unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。
3. 单击右上角的创建服务。



4. 设置集群名称、命名空间、服务名称，选择所要使用的镜像和镜像版本等配置信息。



说明:

本示例中，界面参数取值参考如下：

- 服务名称：自定义该服务的名称。本例为`helloworld-go`。
- 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。您还可以填写私有 registry。填写的格式为*domainname/namespace/imagename:tag*。本例中为 *registry.cn-hangzhou.aliyuncs.com/knative-sample/helloworld-go*。
- 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。本例中为`73fbdd56`。
- 环境变量：支持通过键值对的形式配置环境变量。本例中，`TARGET=Knative`。

界面其他参数详细信息请参见参数说明。

5. 单击创建。

创建完成后，您可以在Knative 服务管理列表中，看到新创建的服务。

预期结果

服务部署完成后，通过绑定Host域名与访问网关，然后可以直接访问服务URL。

1. 在 Kubernetes 菜单下，单击左侧导航栏的Knative > 组件管理，进入组件管理页面。可以看到访问网关。

2. 将访问网关地址与需要访问的域名进行Host绑定，在Hosts文件中添加绑定信息。

```
##+ ##
```

样例如下：

```
47.95.XX.XX helloworld-go.default.example.com
```

3. 完成Host绑定后，可通过域名*http://helloworld-go.default.example.com*直接对服务进行访问。

18.7 在Knative 上实现 Tracing 分布式追踪

链路追踪 Tracing Analysis 为分布式应用的开发者提供了完整的调用链路还原、调用请求量统计、链路拓扑、应用依赖分析等工具。本文介绍了如何在 Knative 上实现 Tracing 分布式追踪，以帮助开发者快速分析和诊断 Knative 中部署的应用服务。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。
- 您已经成功部署 Knative，参见#unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。

操作步骤

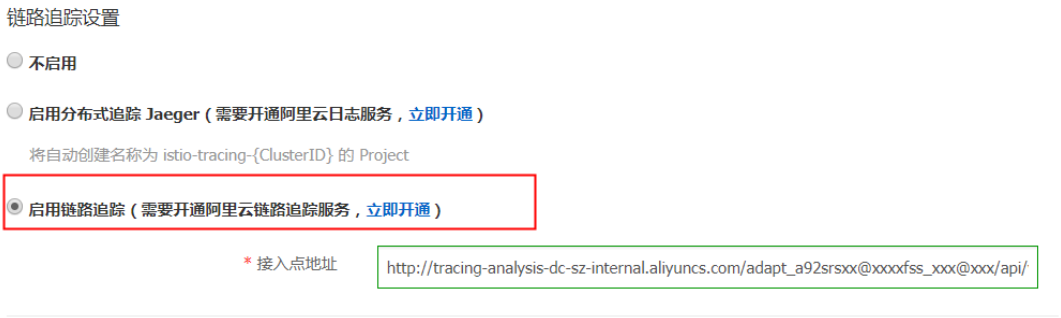
1. 部署Istio。参考#unique_318。

   部署时需要关注以下几点：

   - Pilot 设置跟踪采样百分比，推荐设置大一些（例如，100），便于数据采样。

   - 启用阿里云链路追踪服务，查看 token 选择为on , 客户端采集工具选择zipkin （目前仅支持使用 zipkin 的 /api/v1/spans 接口访问）， 选择集群所在 Region 的接入点（推荐使用内网接入地址），以华南1 Region 为例如下：

   - 选择启用链路追踪，设置链路追踪服务地址。



2. 执行如下命令，选择命名空间设置如下标签启用 Sidecar 自动注入`istio-injection=enabled`。

   通过这种方式就注入了 Istio 的 envoy 代理（proxy）容器， Istio 的 envoy 代理拦截流量后会主动上报 trace 系统。以设置 default 命名空间为例：

   ```
   kubectl label namespace default istio-injection=enabled
   ```

3. 部署 Knative Service 服务。参考#unique_340。

   完成后，您可以通过如下命令，访问Hello World 示例服务。

   ```
   $ curl -H "Host: helloworld-go.default.example.com" http://112.124.
   XX.XX
   Hello Knative!
   ```

4. 登录链路追踪服务控制台，选择应用列表，可以查看对应应用的 tracing 信息。

5. 选择应用，单击查看应用详情，可以看到服务调用的平均响应时间。

总结

在 Knative 中，推荐开启阿里云链路追踪服务。通过 Tracing 不仅有助于复杂应用服务之间的问题排查及效率优化，同时也是 Knative 的最佳实践方式。

18.8 在Knative 上实现日志服务

日志服务（Log Service，简称 LOG）是针对日志类数据的一站式服务。您无需开发就能快捷完成日志数据采集、消费、投递以及查询分析等功能。在 Knative 中结合日志服务，能有效提升对 Serverless 应用的运维能力。

前提条件

- 您已经开通日志服务，参见#unique_342。
- 您已经部署 Knative Service 服务。参考#unique_340。

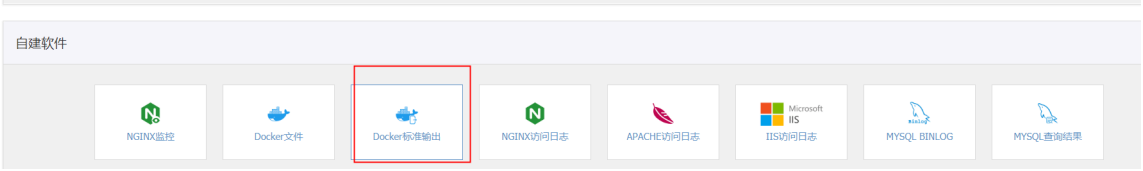
操作步骤

1. 为helloworld 接入日志采集。参考#unique_242。

   a. 登录日志服务管理控制台，单击目标Project名称，在日志库页签的Logstore列表中，可以看到步骤1创建的Logstore。

   b. 在目标Logstore右侧单击数据接入向导，弹出helloword-log页面。

   c. 在自建软件区域选择Docker标准输出，进入指定采集模式页面。



   d. 填写配置名称和插件配置完成后，单击下一步。

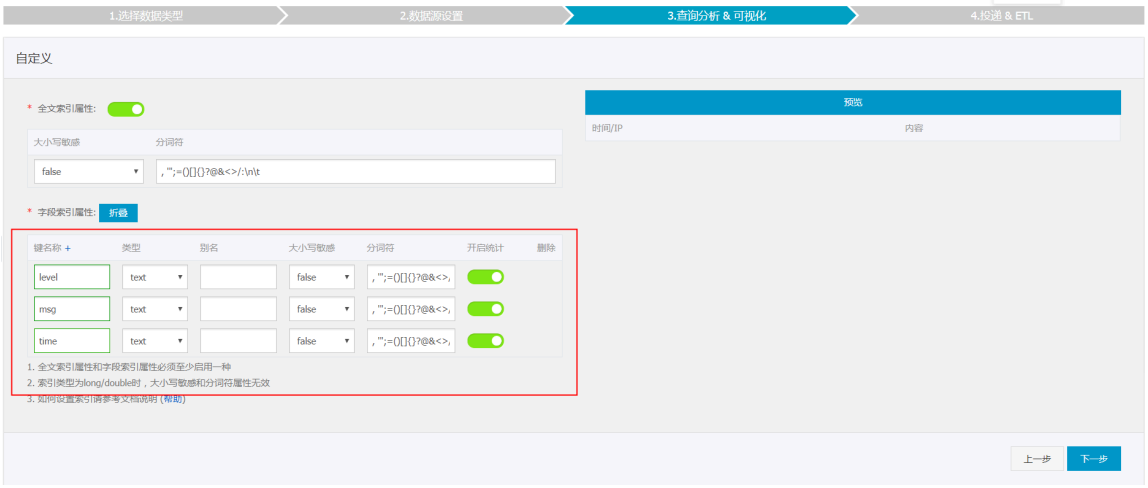
   插件配置这里我们针对 helloworld-go Service, 设置采集的环境变量为：`"K_SERVICE": "helloworld-go"`。并且通过 processors 分割日志信息，如这里`"Keys": [ "time","level", "msg" ]`。插件配置示例如下：

```
{
  "inputs": [
    {
      "detail": {
        "IncludeEnv": {
          "K_SERVICE": "helloworld-go"
        },
        "IncludeLabel": {},
        "ExcludeLabel": {}
      },
      "type": "service_docker_stdout"
    }
  ],
  "processors": [
    {
      "detail": {
        "KeepSource": false,
        "NoMatchError": true,
        "Keys": [
          "time",
          "level",
          "msg"
        ],
```

```
        "NoKeyError": true,
        "Regex": "(\\d+-\\d+-\\d+\\s+\\d+:\\d+:\\d+)\\s+(\\w+)\\s
+(.*)",
        "SourceKey": "content"
      },
      "type": "processor_regex"
    }
  ]
}
```

e. 进入应用到机器组页面，勾选目标Project名称并单击应用到机器组。

f. 进入自定义页面，开启全文索引属性，设置查询字段索引属性，并单击下一步。



g. 进入投递 & ETL页面，确认无误后，单击确认。

2. 执行如下命令，访问 Hello World 示例服务。

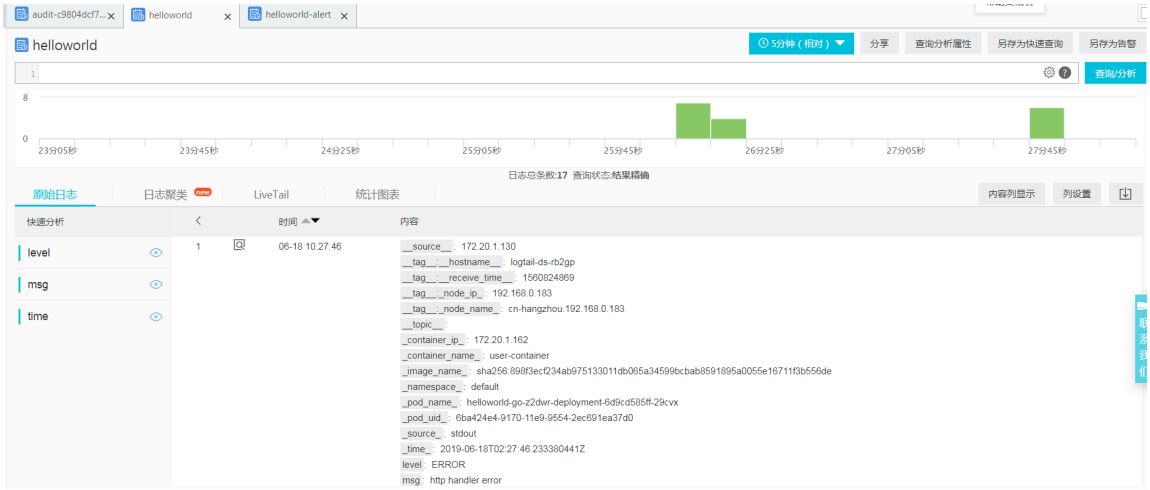
   此时会产生日志信息。

```
$ curl -H "Host: helloworld-go.default.example.com" http://112.124.
XX.XX
Hello Knative!
```

3. 在Logstore列表中，在目标Logstore右侧单击查询，可以看到该logstore的日志信息。

4. 设置查询分析。参考#unique_343。

   a. 为了便于查看，你可以通过列设置显示所需要的列。下图中的示例设置了level、msg和time这三列。



总结

通过上面的介绍，可以帮助您了解如何在 Knative 中使用日志服务收集 Serverless 应用容器日志。在 Knative 中采用日志服务收集、分析业务日志，满足了生产级别的 Serverless 应用运维的诉求。

18.9 在Knative 上实现监控告警

Knative 中可以通过日志服务收集日志信息，同时可以根据日志信息，在日志服务中快速方面的设置监控告警。

前提条件

- 您已经部署Hello World Service， 并实现日志采集，请参见#unique_345。
- 您已连接到Kubernetes集群的Master节点，参见#unique_37。

操作步骤

1. 设置查询分析。参见#unique_343。
   a. 登录日志服务管理控制台，单击目标Project名称，在日志库页签的Logstore列表中，可以看到步骤1创建的Logstore。
   b. 在目标Logstore右侧单击查询。
   c. 输入如下命令并单击查询/分析。

      如果您监控的原则是根据ERROR出现的次数，则设计统计 ERROR 的SQL语句：

```
* | select 'ERROR' , count(1) as total group by 'ERROR'
```

2. 告警设置。请参见#unique_346。
   a. 单击右上角的告警另存为，从右侧滑出侧边栏创建告警。

说明:

其中告警触发条件输入判断告警是否触发的条件表达式，可以参考#unique_347，我们这里设置查询区间为 1 分钟，执行间隔为 1 分钟，触发条件为 `total > 3`， 表示间隔 1 分钟检查，如果 1 分钟内出现 3 次 ERROR 信息，则触发告警。

b. 填写告警配置信息，完成后单击下一步。

配置项	说明
告警名称	告警的名称。名称长度为1~64个字符。
添加仪表盘	添加一个仪表盘。
图表名称	图表的名称。名称长度为1~64个字符。
查询语句	SQL查询语句。

配置项	说明
查询区间	查询区间为服务端每次执行查询时，读取的数据时间范围，支持相对时间与绝对时间。例如，执行时间点为14:30:06，设置查询区间为15分钟（相对），则查询区间为 14:15:06- 14:30:06；设置查询区间为15分钟（绝对），则查询区间为：14:15:00- 14:30:00。
执行间隔	服务端每次执行告警检查的时间间隔，即告警规则每隔多长时间执行一次。 告警执行间隔的配置范围为60~86400秒，即最小间隔为60秒，最大间隔为24小时。 说明: 目前服务端每次告警规则检查只会采样处理时间区间开始的前100条数据。
触发条件	判断告警是否触发的条件表达式，满足该条件时会根据执行间隔和通知间隔发送告警通知。 例如，您可以设置为`pv%100 > 0 && uv > 0`。 说明: 触发条件中，通过`$编号`区分不同的关联图表，例如`$0`表示编号为0的图表。 如何查看图表编号?
触发通知阈值	累计触发次数达到该阈值时根据通知间隔发送告警。不满足触发条件时不计入统计。 默认触发通知阈值为1，即满足一次触发条件即可检查通知间隔。 通过配置触发通知阈值可以实现多次触发、一次通知。例如，配置触发通知阈值为100，则累计触发次数达到100次时检查通知间隔。如果同时满足触发通知阈值和通知间隔，则发送通知。发送通知之后，累计次数会清零。如果因网络异常等原因执行检查失败，不计入累计次数。

配置项	说明
通知间隔	两次告警通知之间的时间间隔。 如果某次执行满足了触发条件，而且累计的触发次数已经达到触发通知阈值，且距离上次发送通知已经达到了通知间隔，则发送通知。如设置通知间隔为5分钟，则5分钟内至多收到一次通知。默认无间隔。 **说明:** 通过配置触发通知阈值和通知间隔可以实现告警抑制的功能，防止收到过多的告警信息。

c. 设置告警通知，完成后单击提交。

通知方式	通知方式
短信	短信形式发送告警通知，需要指定手机号码和发送内容。 多个手机号码之间通过逗号（，）分隔。发送内容为短信通知的内容，支持使用模板变量，长度为1~100个字符。
语音	语音形式发送告警通知，需要指定手机号码和发送内容。 多个手机号码之间通过逗号（，）分隔。发送内容为语音通知的内容，支持使用模板变量，长度为1~100个字符。
邮件	邮件形式发送告警通知，需要指定邮箱地址为收件人，并指定发送内容。 多个邮箱地址之间通过逗号（，）分隔。发送内容为邮件通知的内容，支持使用模板变量，长度为1~500个字符。
钉钉	钉钉机器人消息形式发送告警通知，当触发告警时，告警通知会以钉钉机器人消息的形式发送到钉钉群中。需要指定请求地址和发送内容。 发送内容为钉钉机器人消息的内容，支持使用模板变量，长度为1~500个字符。 如何设置钉钉机器人、获取请求地址，请查看#unique_346。 说明: 每个机器人每分钟最多发送20条。

通知方式	通知方式
WebHook	当触发告警时，告警通知会以指定形式发送到自定义WebHook地址中。需要指定请求地址、请求方法和发送内容。 请求方法可以设置为GET、PUT、POST、DELETE、和OPTIONS。发送内容为通知的内容，支持使用模板变量，长度为1~500个字符。
通知中心	通过阿里云通知中心中预设的通知方式向联系人发送告警通知。需要指定发送内容。发送内容为通知消息的内容，支持使用模板变量，长度为1~500个字符。 添加通知中心告警方式，需要在通知中心中设置联系人及通知方式。

3. 执行如下命令，访问 Hello World 示例服务。

   此时会触发告警通知。

```
$ curl -H "Host: helloworld-go.default.example.com" http://112.124.
XX.XX
Hello Knative!
```

4. 如果是设置的邮件通知，告警信息如下图所示。

总结

通过结合日志服务的监控告警方式，在 Knative 中可以第一时间监控到部署应用的异常状态并及时通知给运维、开发人员进行处理，保证服务持续运行。

18.10 Knative 自定义域名

本文介绍了如何在 Knative Serving 中使用自定义域名。

前提条件

- 您已经成功申请一个域名，参见阿里云域名服务。
- 您已经成功部署 Knative，参见 #unique_335。

背景信息

在 Knative Serving route 路由中默认使用 `example.com` 作为默认域名，route 完全定义的域名格式默认为：`{route}.{namespace}.{default-domain}`在 Knative Serving route 路由中默认使用。

通过 Kubectl 修改域名

下面以在命名空间knative-serving中将域名修改为自定义域名为例进行介绍。

1. #unique_33
2. 执行如下命令，编辑域名 config-map 配置文件 *config-domain*。

   ```
   kubectl edit cm config-domain --namespace knative-serving
   ```

3. 修改配置文件。

   使用自定义域名（如：mydomain.com），替换掉 example.com，保存配置。

   ```
   apiVersion: v1
   data:
     mydomain.com: ""
   kind: ConfigMap
   [...]
   ```

部署应用

如果你已经部署了应用，Knative 会根据域名配置 configmap 的修改，自动更新所有的 Service 和 Route。

1. 部署一个应用。请参见部署 Serving Hello World 应用。

   本例中部署一个命名为*helloworld-go*的应用。

2. 执行以下操作，查看部署结果。

   · 在Knative 0.7 版本下，执行如下命令，查看域名。

   ```
   kubectl get route ${SVC_NAME} --output jsonpath="{.status.url}"|
   awk -F/ '{print $3}'`
   ```

   · 在Knative 0.6 版本下，执行如下命令，查看域名。

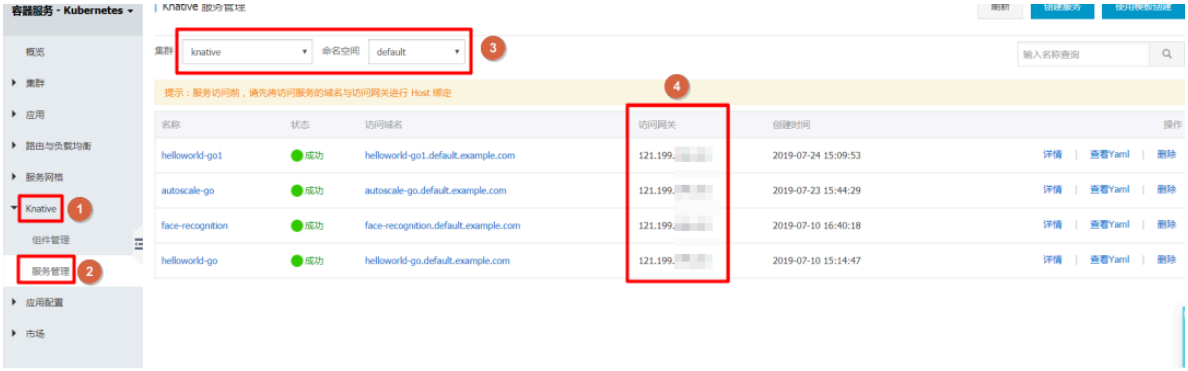
   ```
   kubectl get route helloworld-go --output jsonpath="{.status.domain
   }"
   ```

   当返回结果如下时，表示自定义域名已生效。

   ```
   helloworld-go.default.mydomain.com
   ```

域名发布

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。
3. 选择目标集群及命名空间，可以看到目标应用对应的访问网关。



4. 将Knative 网关 ip 设置到对应的域名解析，参见添加解析记录。

预期结果

执行如下命令，查看执行结果。

```
curl http://helloworld-go.default.mydomain.com
```

18.11 在 Knative 上实现 GitHub 事件处理

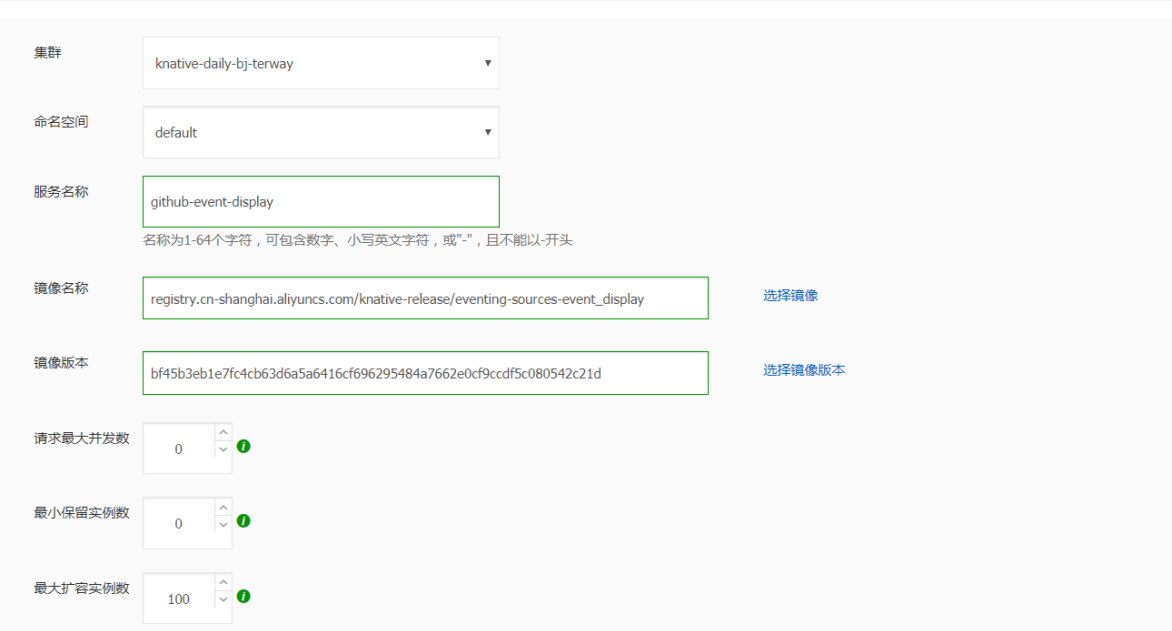
本文介绍了如何在 Knative 上实现 GitHub 事件处理。

前提条件

· 您已经成功部署 Serving 组件、Eventing组件及add-on组件，参见#unique_352。
· 您已经在Knative上自定义域名并完成域名配置。参见#unique_353。

创建服务

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。
3. 单击右上角的创建服务。
4. 设置集群名称、命名空间、服务名称，选择所要使用的镜像和镜像版本等配置信息。



说明:

本示例中，界面参数取值参考如下：

- 服务名称：自定义该服务的名称。本例为`github-event-display`。
- 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。您还可以填写私有 registry。填写的格式为*domainname/namespace/imagename:tag*。本例中为*registry.cn-shanghai.aliyuncs.com/knative-release/eventing-sources-event_display* 。
- 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。本例中为`bf45b3eb1e7fc4cb63d6a5a6416cf696295484a7662e0cf9ccdf5c080542c21d`。

界面其他参数详细信息请参见参数说明。

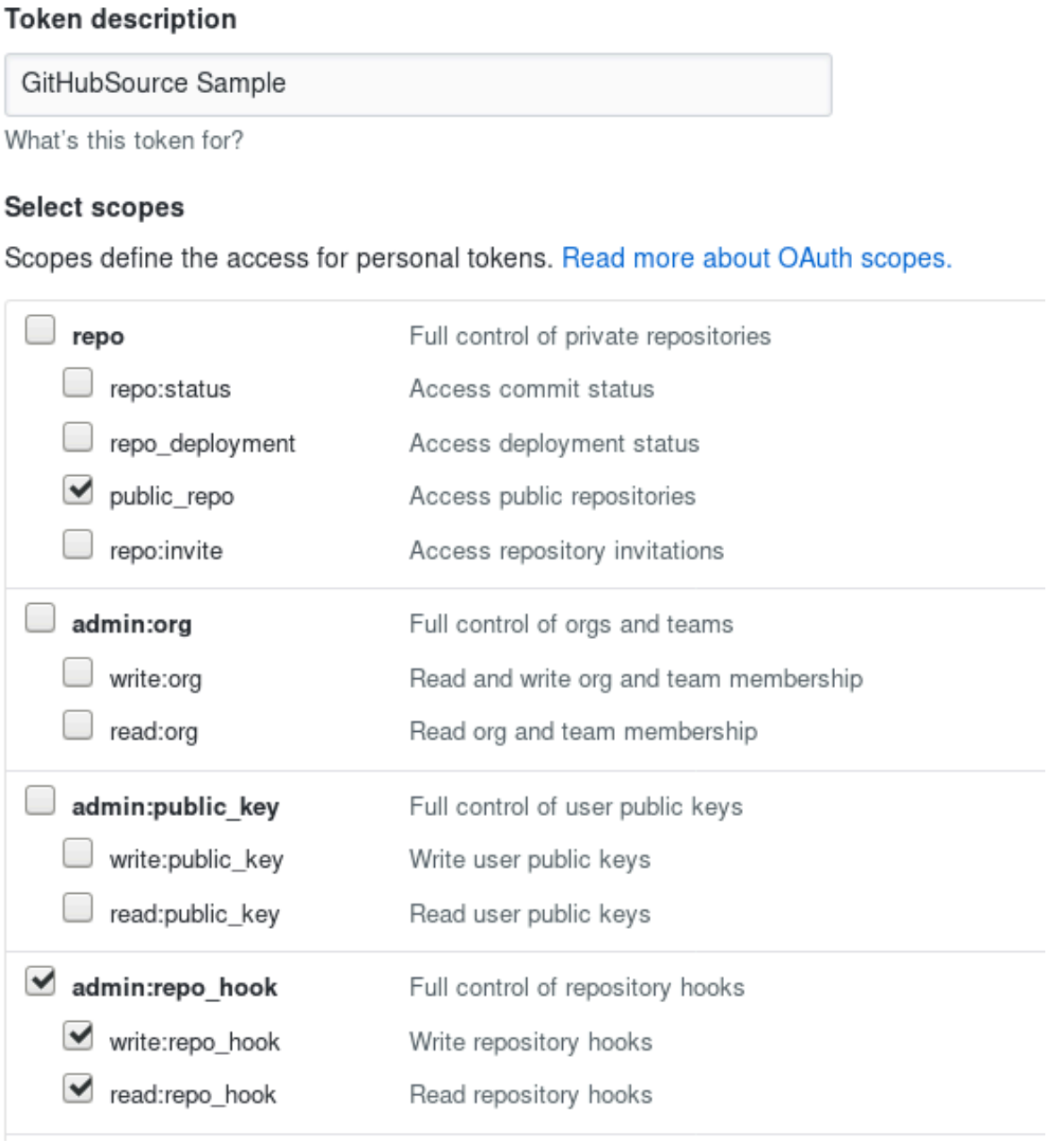
5. 单击创建。

创建 GitHub Token

1. 创建 Personal access token。

   Personal access token用于访问 GitHub API。请参见Personal access tokens。

   下面是设置一个 `GitHubSource Sample` token 的示例。



   说明:

   需要开启 `repo:public_repo` 和 `admin:repo_hook`，以便通过公共仓库触发 Event 事件，并为这些公共仓库创建 webhooks 。

   您可以自定义token 的名称。

2. 通过 kubectl 连接 Kubernetes 集群。

3. 更新*githubsecret.yaml*。

   如果生成的是 *personal_access_token_value* token，则需要设置 secretToken 。示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: githubsecret
type: Opaque
stringData:
  accessToken: personal_access_token_value
  secretToken: asdfasfdsaf
```

4. 执行如下命令，创建 GitHub Token。

```
kubectl --namespace default apply githubsecret.yaml
```

创建 GitHub 事件源

您可以创建 GitHubSource 用于接收 GitHub 产生的事件。

1. 通过 kubectl 连接 Kubernetes 集群。
2. 创建并拷贝以下内容到*github-source.yaml*文件中。

```
apiVersion: sources.eventing.knative.dev/v1alpha1
kind: GitHubSource
metadata:
  name: githubsourcesample
spec:
  eventTypes:
    - pull_request
  ownerAndRepository: <YOUR USER>/<YOUR REPO>
  accessToken:
    secretKeyRef:
      name: githubsecret
      key: accessToken
  secretToken:
    secretKeyRef:
      name: githubsecret
      key: secretToken
  sink:
    apiVersion: serving.knative.dev/v1alpha1
    kind: Service
    name: github-event-display
```

3. 执行以下命令，在命名空间default中创建 GitHubSource。

```
kubectl --namespace default apply github-source.yaml
```

预期结果

在 GitHub repository 中 Settings > Webhooks，查看会有一个验证成功的 Hook URL。

说明：

域名需要备案，否则无法进行访问。

在 GitHub repository 中执行以下命令，创建一个 `pull request`，此时会产生Event事件。

```
kubectl --namespace default get pods
kubectl --namespace default logs github-event-display-XXXX user-
container
```

在 Knative Eventing 可以看到类似下面的事件结果。

```
2018/11/08 18:25:34 Message Dumper received a message: POST / HTTP/1.1
Host: github-event-display.knative-demo.svc.cluster.local
Accept-Encoding: gzip
Ce-Cloudeventsversion: 0.1
Ce-Eventid: a8d4cf20-e383-11e8-8069-46e3c8ad2b4d
Ce-Eventtime: 2018-11-08T18:25:32.819548012Z
Ce-Eventtype: dev.knative.source.github.pull_request
Ce-Source: https://github.com/someuser/somerepo/pull/1
Content-Length: 21060
Content-Type: application/json
User-Agent: Go-http-client/1.1
X-B3-Parentspanid: b2e514c3dbe94c03
X-B3-Sampled: 1
X-B3-Spanid: c85e346d89c8be4e
X-B3-Traceid: abf6292d458fb8e7
X-Envoy-Expected-Rq-Timeout-Ms: 60000
X-Envoy-Internal: true
X-Forwarded-For: 127.0.0.1, 127.0.0.1
X-Forwarded-Proto: http
X-Request-Id: 8a2201af-5075-9447-b593-ec3a243aff52

{"action":"opened","number":1,"pull_request": ...}
```

18.12 删除修订版本

本文主要为您介绍如何删除修订版本。

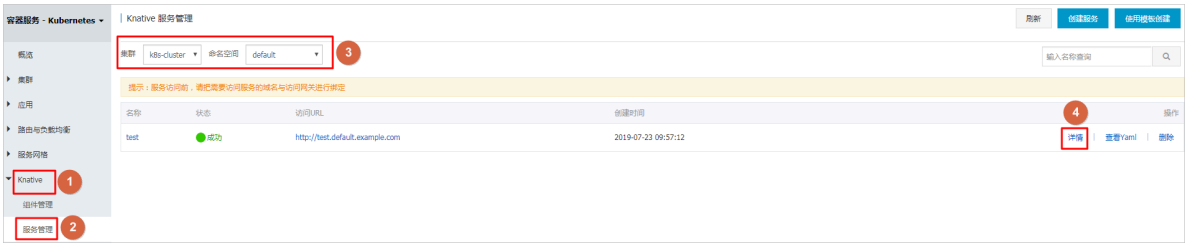
背景信息

- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见#unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。
- 您已经成功创建 Knative 服务，参见创建 Knative 服务。

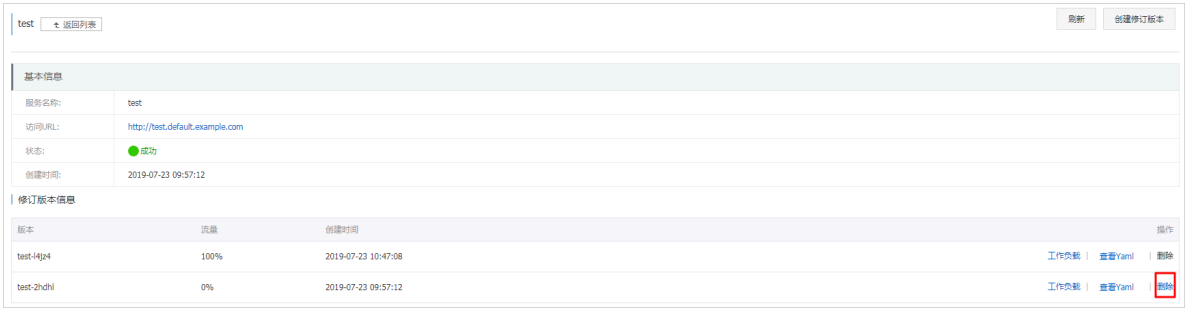
操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。

3. 选择目标集群及命名空间，单击操作列的详情，进入目标服务详情页面。



4. 在修订版本信息区域，选择待删除的修订版本，单击操作列的删除。



> **说明:**
> 流量为0表示该修订版本未提供服务请求，此时才可以删除。当该修订版本的流量不为零时，无法删除。

5. 在弹出的删除修订版本提示框中，单击确认，删除修订版本。

18.13 删除 Knative 服务

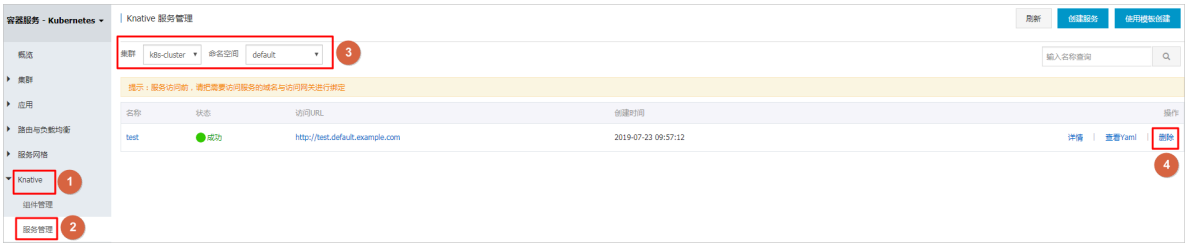
本文主要为您介绍如何删除 Knative 服务。

背景信息

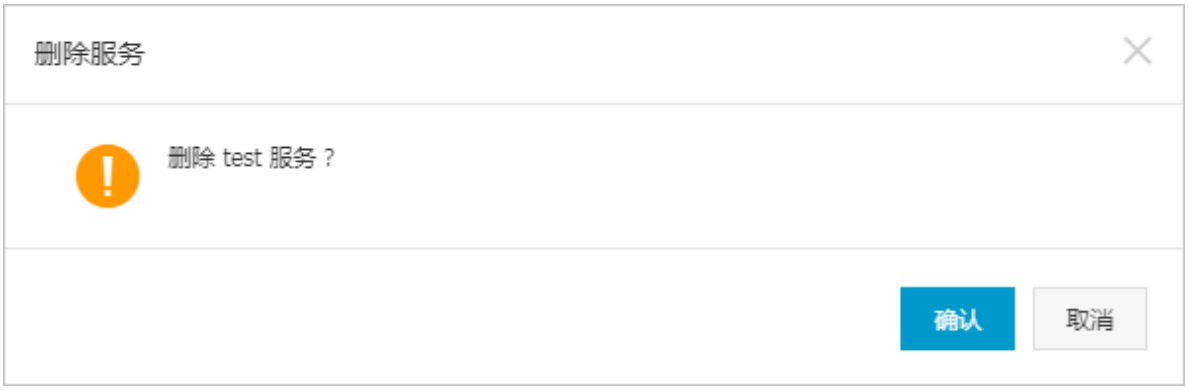
- 您已经成功创建一个 Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见#unique_335。
- 您已经成功部署 Serving 组件，参见部署组件。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 服务管理，进入服务管理页面。
3. 选择待删除的服务，单击操作列的删除。



4. 在弹出的删除服务提示框中，单击确认，删除服务。



18.14 卸载组件

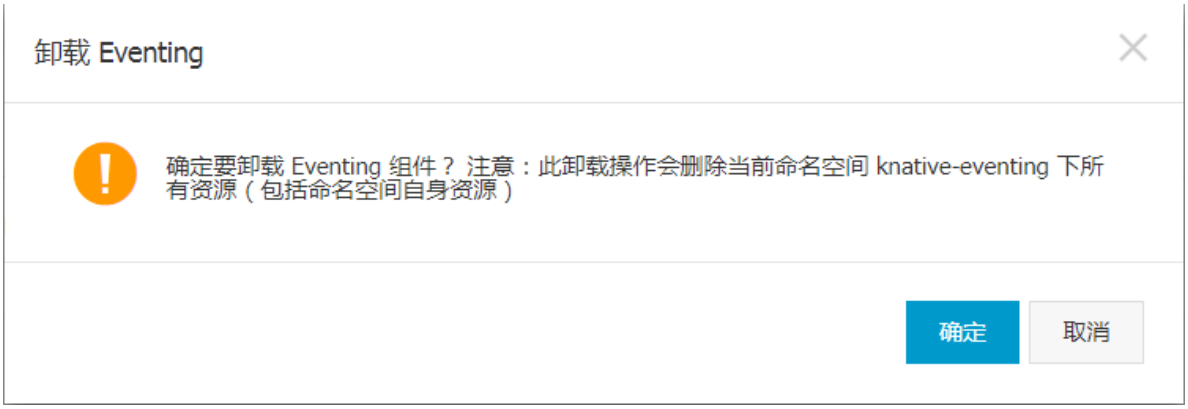
前提条件

- 您已经成功创建一个Kubernetes 集群，参见#unique_73。
- 您已经成功部署 Knative，参见#unique_335 。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 组件管理，进入Knative 组件管理页面。
3. 在目标组件右侧单击卸载。

4. 弹出卸载 Eventing页面，单击确定。



执行结果

卸载完成后，在Knative 组件管理页面，可以看到目标组件Eventing的状态为未部署。

18.15 卸载 Knative

前提条件

您已经成功部署 Knative，参见#unique_335。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择Knative > 组件管理，单击右上角的卸载。

3. 弹出卸载 Knative页面，勾选我已知晓并确认卸载 Knative，单击确认。

执行结果

卸载完成后，可以看到执行结果信息。

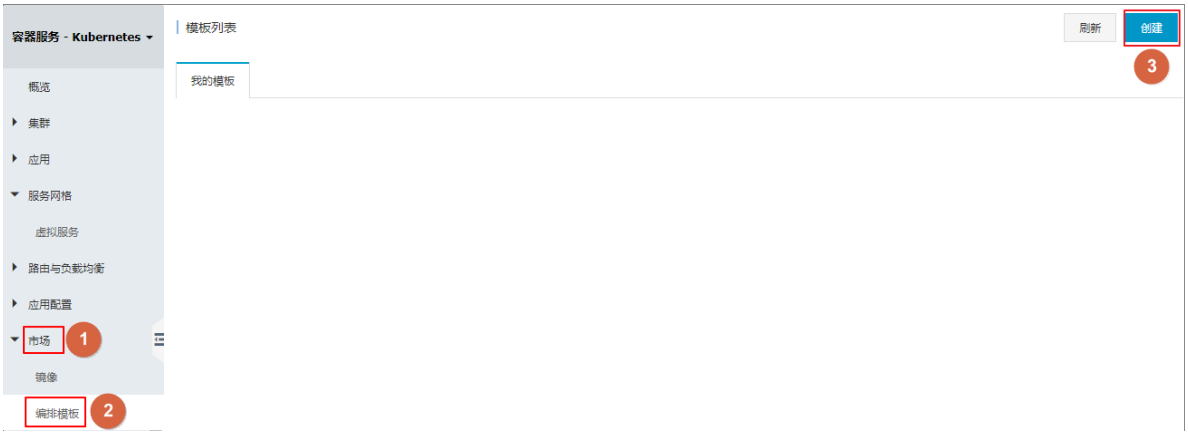
19 模板管理

19.1 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。

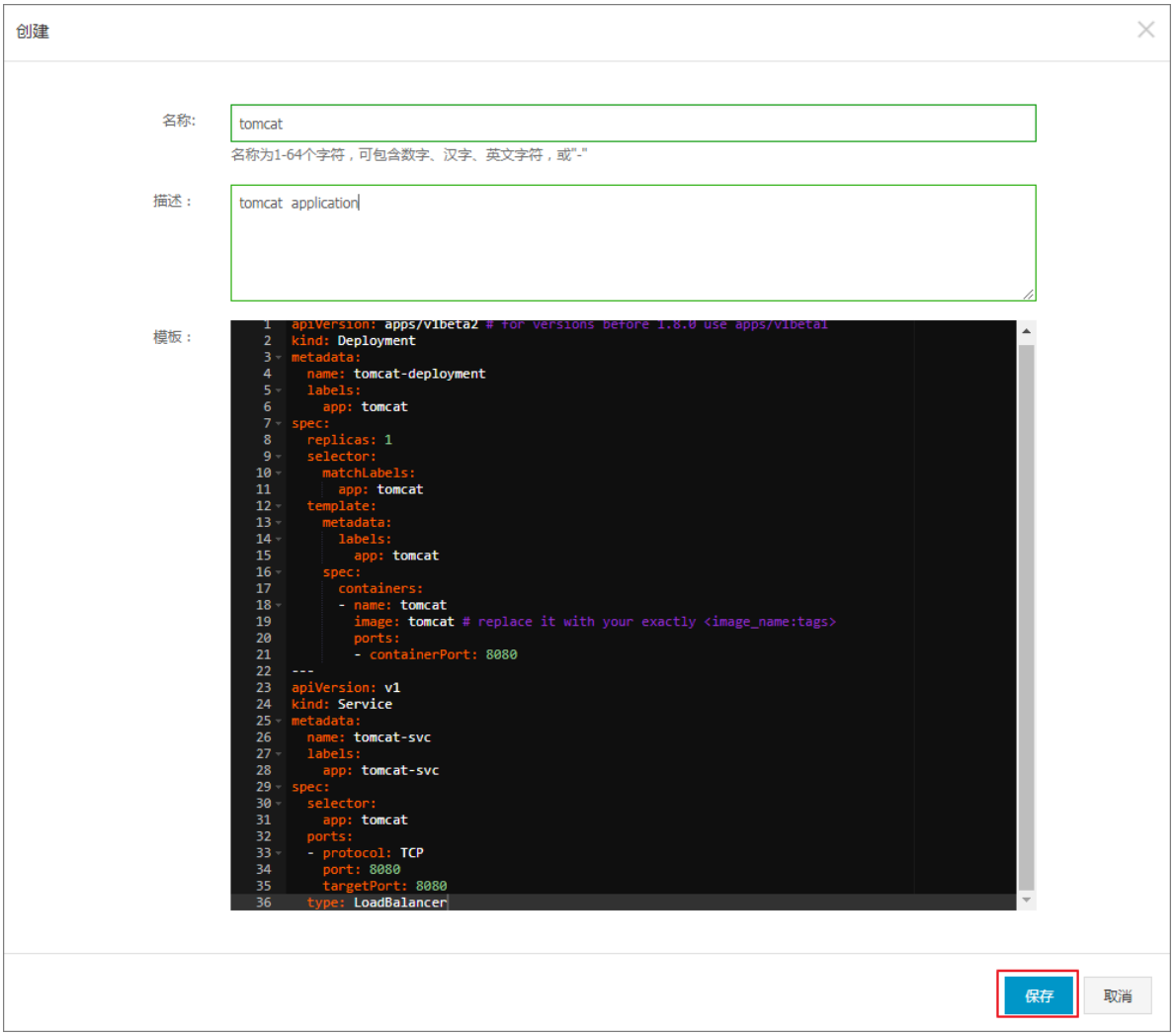
操作步骤

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。

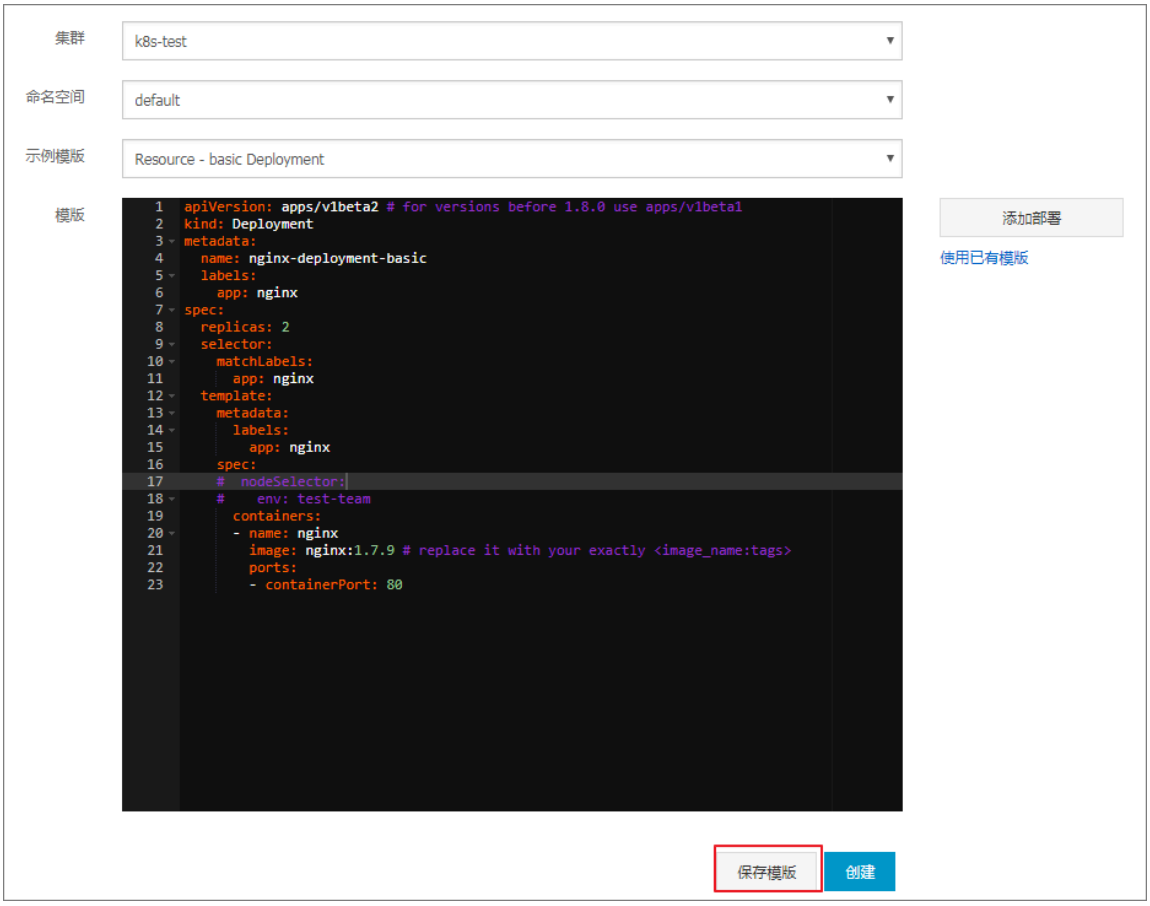
   - 名称：设置该模板的名称。
   - 描述：输入对该模板的描述，可不配置。
   - 模板：配置符合Kubernetes Yaml语法规则的编排模板，可包含多个资源对象，以---进行分割。



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 > 无状态，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。

后续步骤

您可使用我的模板下的编排模板，快速创建应用。

19.2 更新编排模板

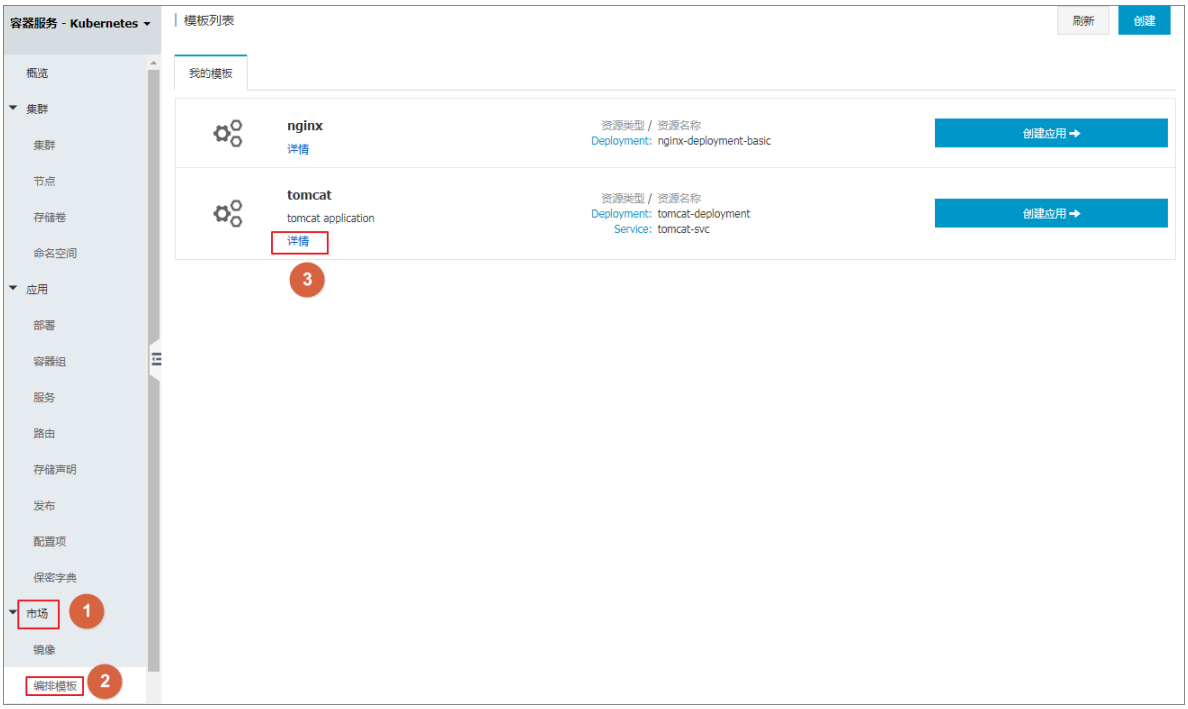
您可对已有的编排模板进行编辑，从而更新编排模板。

前提条件

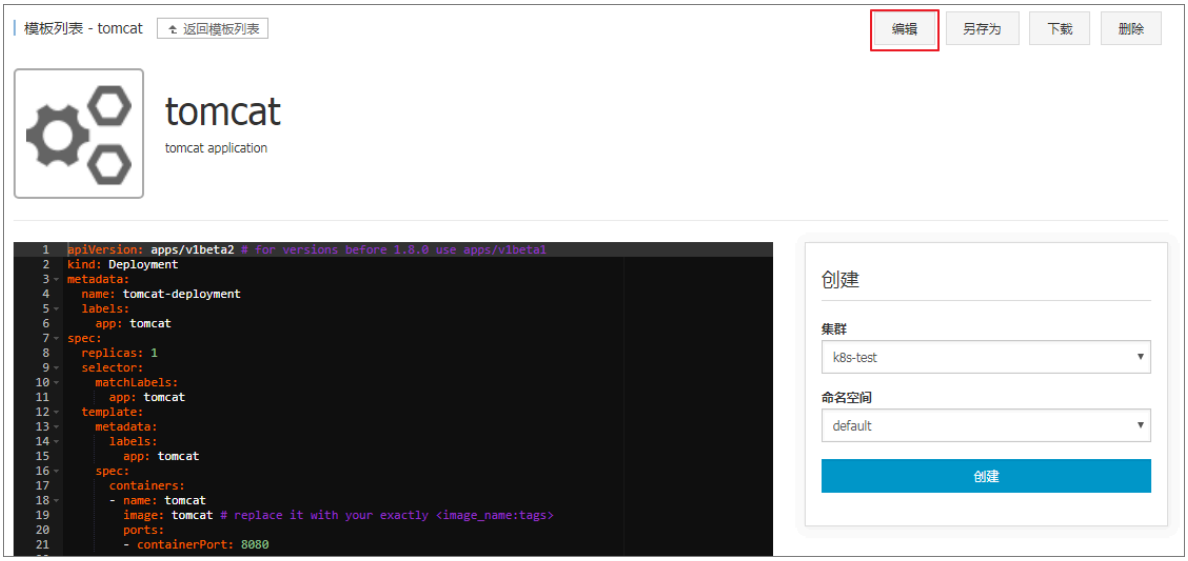
您已经创建一个编排模板，参见#unique_362。

操作步骤

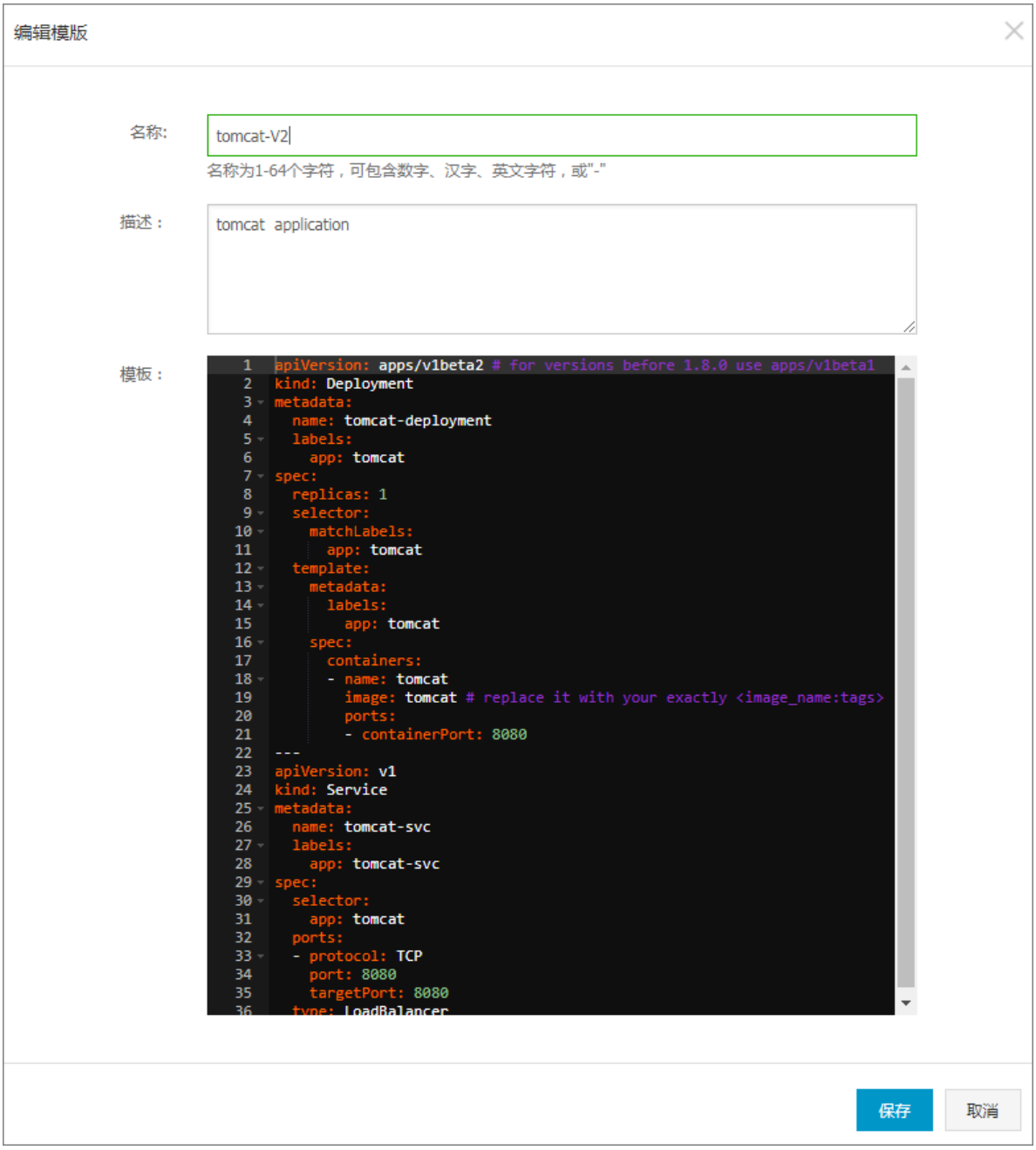
1. 登录容器服务管理控制台。
2. 在Kubernetes菜单下，单击左侧导航栏中的市场 > 编排模板，进入模板列表页面，在我的模板下，您可看到已有的编排模板。
3. 选择所需的模板，单击详情，进入模板详情页面。



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。

19.3 另存编排模板

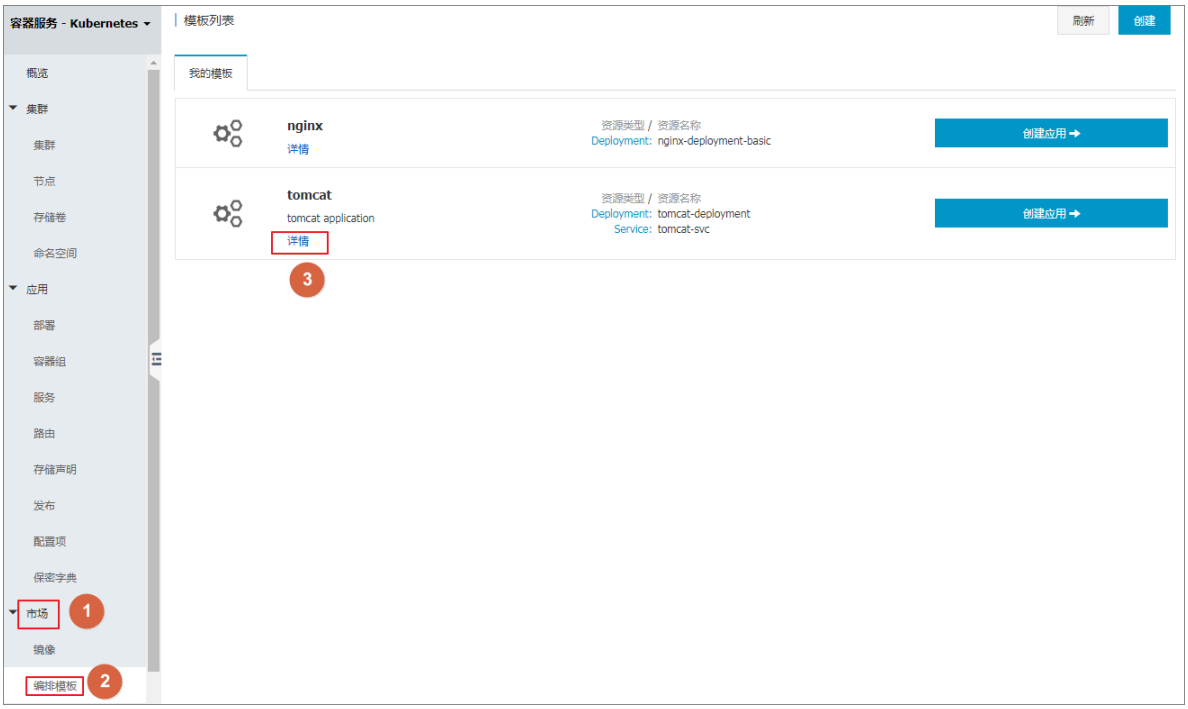
您可对已有模板进行另存，保存为一个新的编排模板。

前提条件

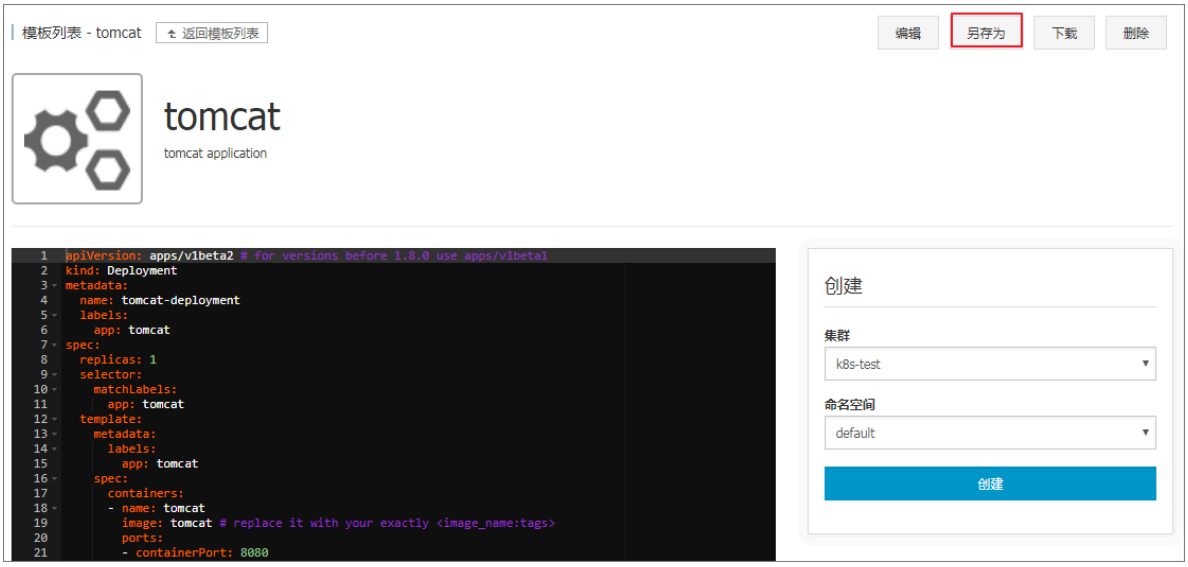
您已经创建一个编排模板，参见#unique_362。

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。

19.4 下载编排模板

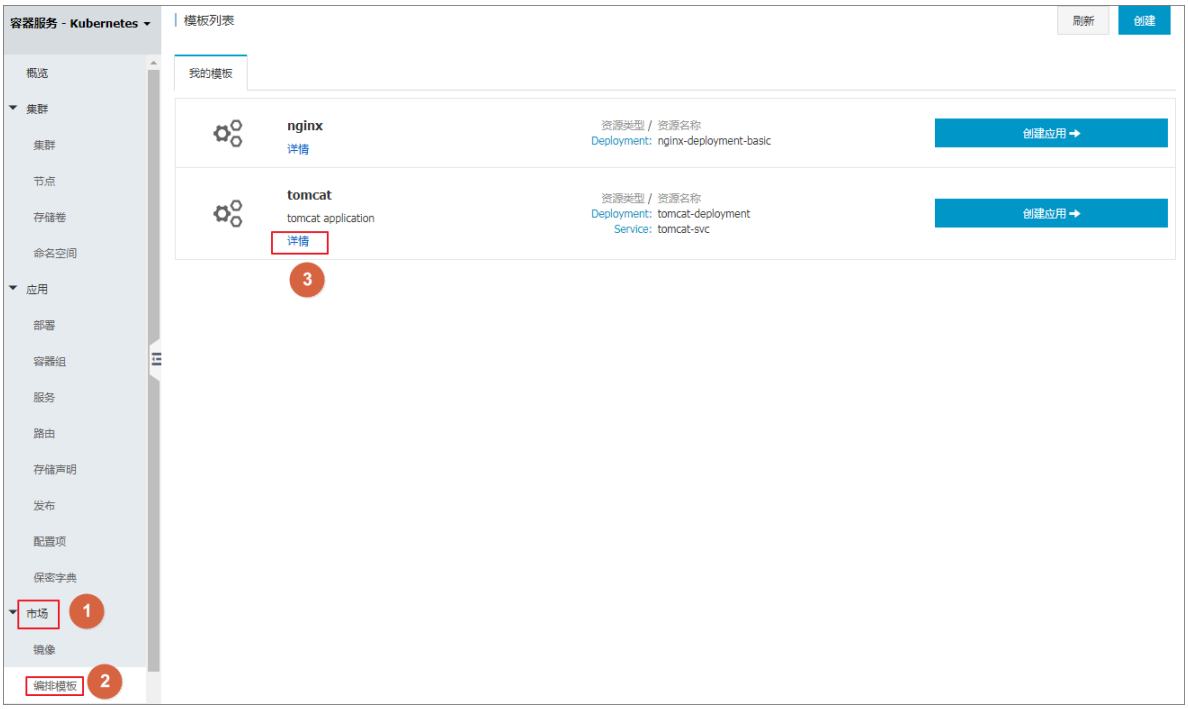
您可下载已有的编排模板。

前提条件

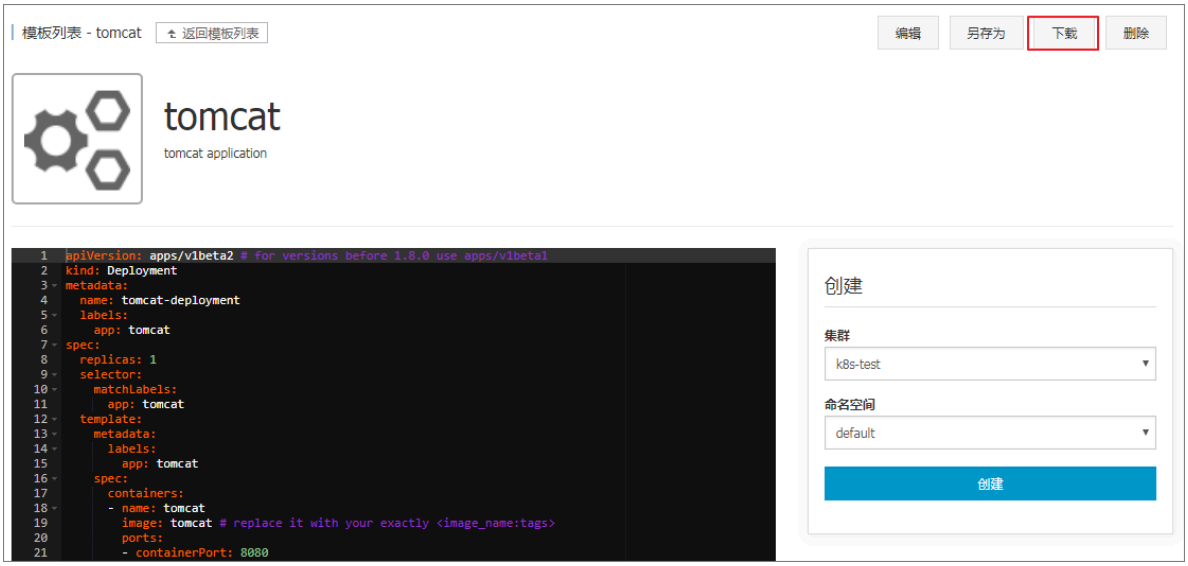
您已经创建一个编排模板，参见#unique_362。

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。



19.5 删除编排模板

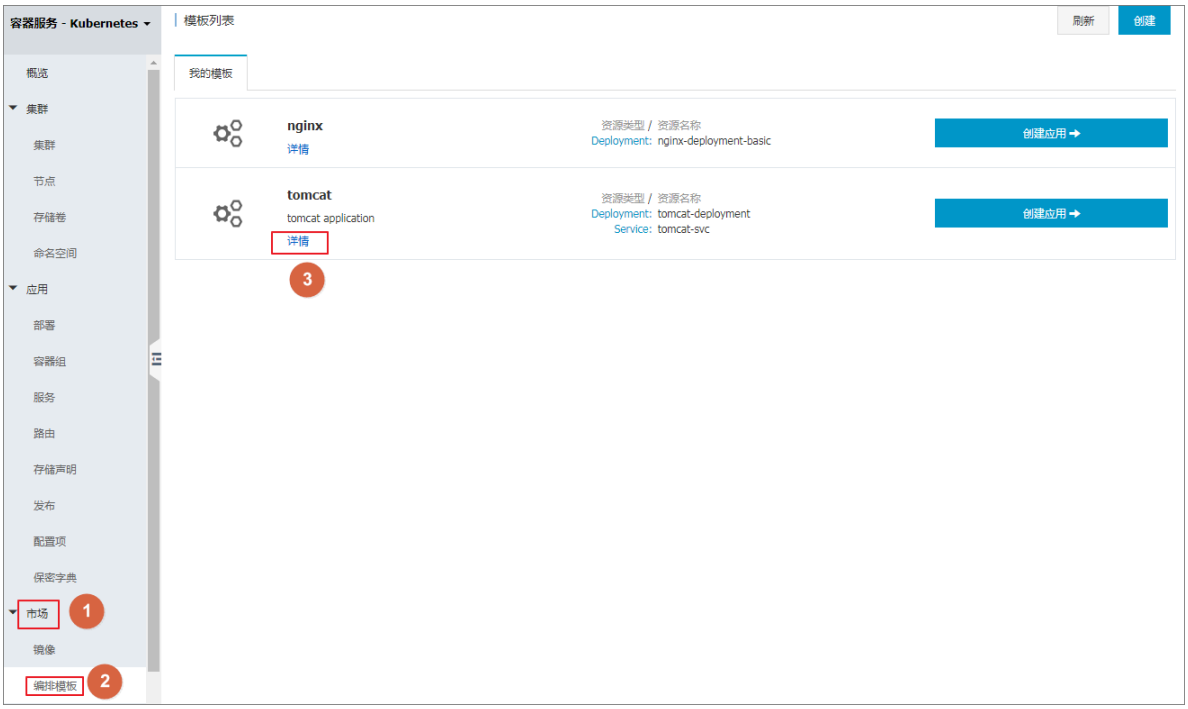
您可删除不再需要的编排模板。

前提条件

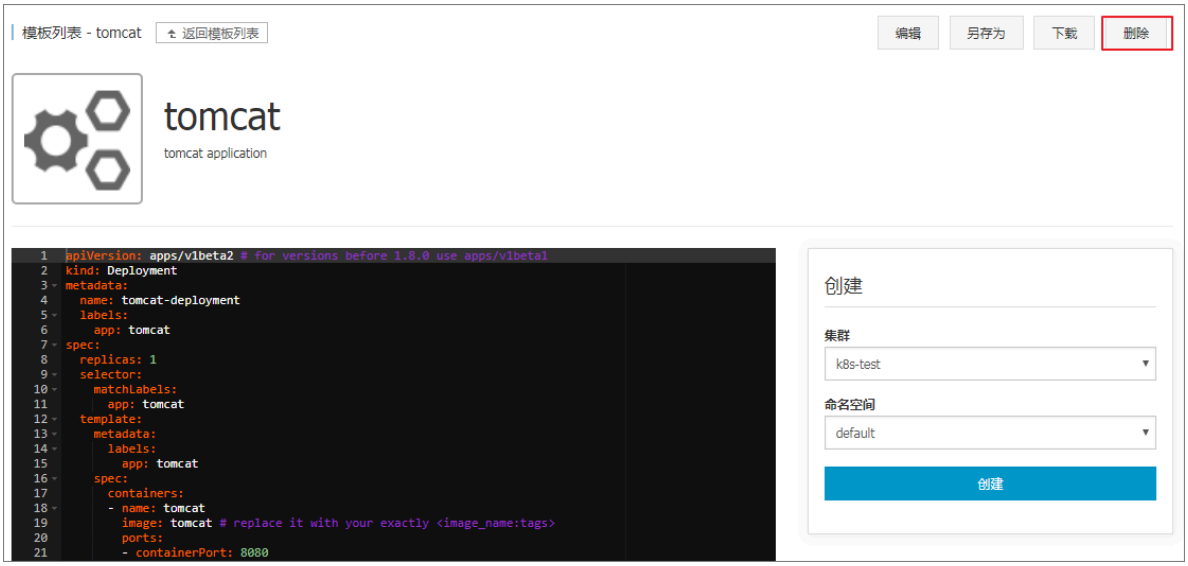
您已经创建一个编排模板，参见#unique_362。

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。

20 应用目录管理

20.1 应用目录概述

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，例如提供图形化界面、阿里云官方 Repository 等。

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

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，例如，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#unique_368。

20.2 查看应用目录列表

您可以通过控制台查看应用目录。

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 > 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。

后续步骤

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#unique_368 。

21 服务目录管理

21.1 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。

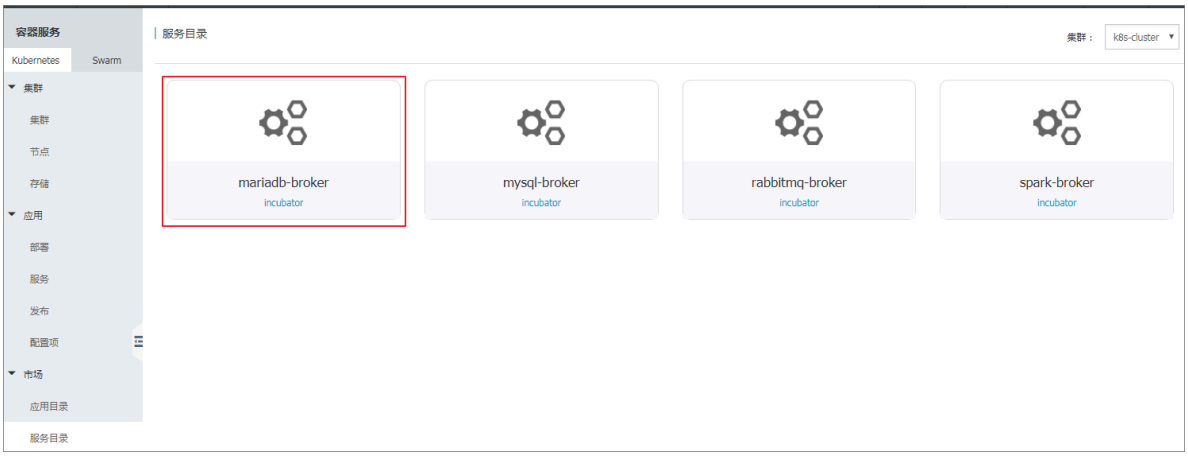
21.2 开通服务目录

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。

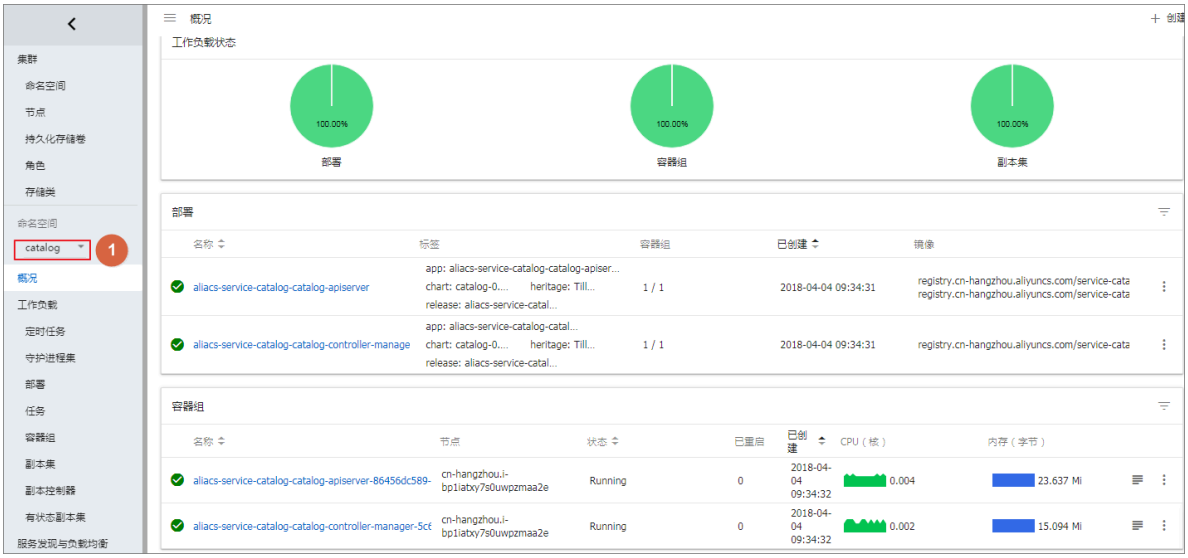
4. 安装完成后，服务目录页面中会显示默认安装的 Service Broker，您可以进入 mariadb-broker 了解详情。

> **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。



后续步骤

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。

22 弹性伸缩

22.1 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#unique_48。
- 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 > 无状态，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。

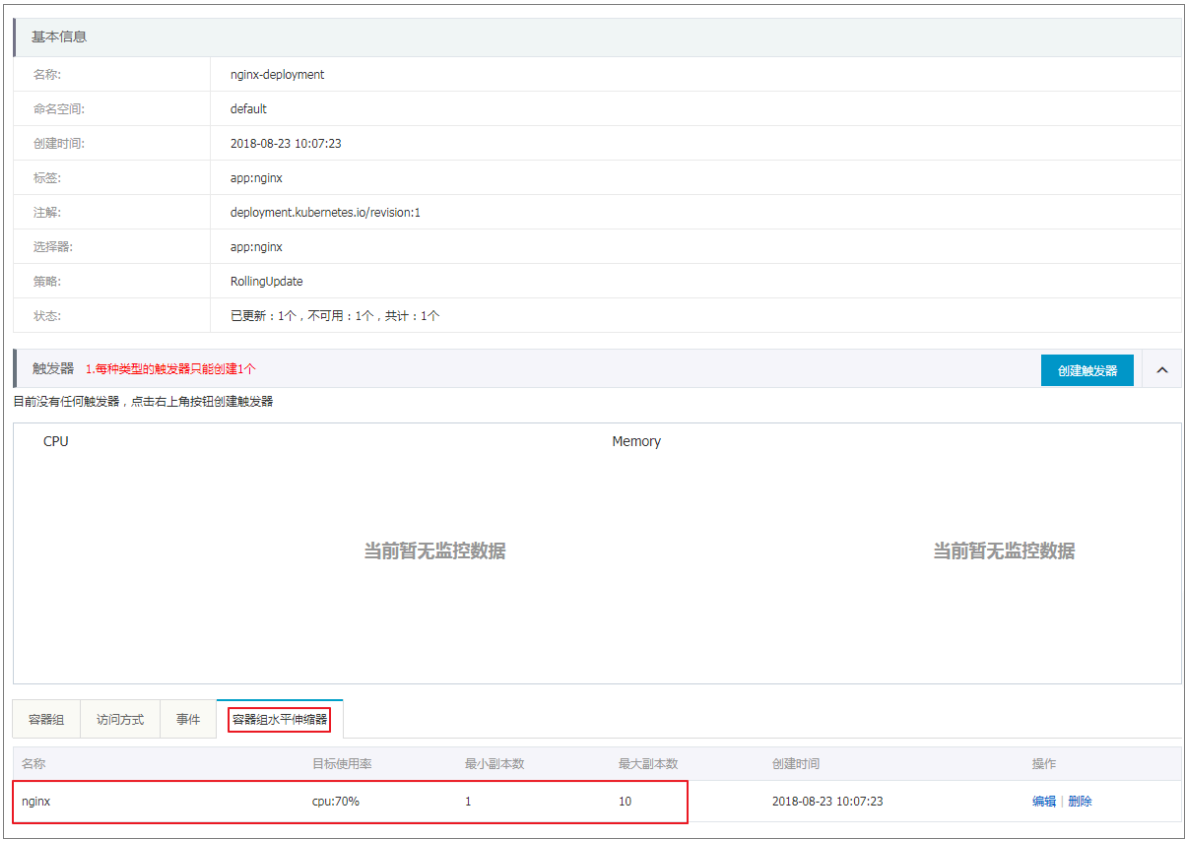
   - 指标：支持CPU和内存，需要和设置的所需资源类型相同。
   - 触发条件：资源使用率的百分比，超过该使用量，容器开始扩容。
   - 最大容器数量：该Deployment可扩容的容器数量上限。
   - 最小容器数量：该Deployment可缩容的容器数量下限。

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

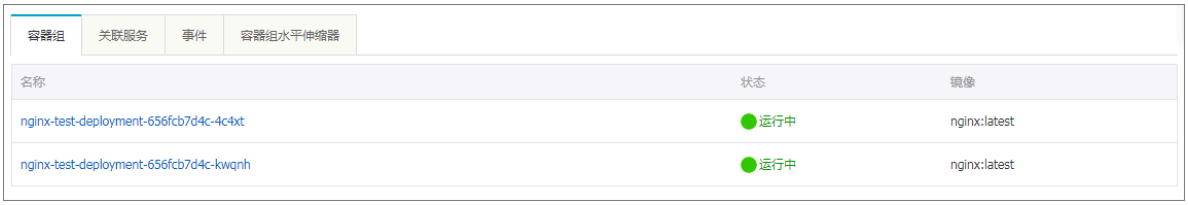
> **说明:**
> 您必须为Deployment设置所需资源，否则无法进行容器自动伸缩。

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。

22.2 节点自动伸缩

阿里云容器服务的自动伸缩能力是通过节点自动伸缩组件实现的，可以按需弹出普通实例、GPU实例、竞价付费实例，支持多可用区、多实例规格、多种伸缩模式，满足不同的节点伸缩场景。

工作原理

节点自动伸缩组件是基于kubernetes资源调度的分配情况进行伸缩判断的，节点中资源的分配是通过资源请求（Request）进行计算的。当Pod由于资源请求（Request）无法满足并进入等待（Pending）状态时，节点自动伸缩组件会根据配置的弹性伸缩组配置信息中的资源规格以及约束配置，计算所需的节点数目，如果可以满足伸缩条件，则会触发伸缩组的节点加入。当一个节点在弹性伸缩组中且节点上Pod的资源请求低于阈值时，节点自动伸缩组件会将节点进行缩容。因此资源请求（Request）的正确、合理设置，是弹性伸缩的前提条件。

注意事项

- 默认单个用户按量付费实例的配额是30台，单个VPC的路由表限额是50条。如需更大的配额，请提交工单申请。
- 单一规格的ECS库存容量波动较大，建议在伸缩组中配置多种同规格的实例类型，提高节点伸缩成功率。
- 极速弹出模式在节点进入停机回收状态时，节点将进行停机，并处在NotReady状态，当再次伸缩弹出时，节点状态会变为Ready。
- 极速弹出模式的节点处在停机回收状态时，只收取磁盘的费用，不收取计算费用（不包含拥有本地盘的机型系列，例如，ecs.d1ne.2xlarge），在库存充裕的前提下可以极速启动。

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 > 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

1. 开通ESS服务

   a. 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。

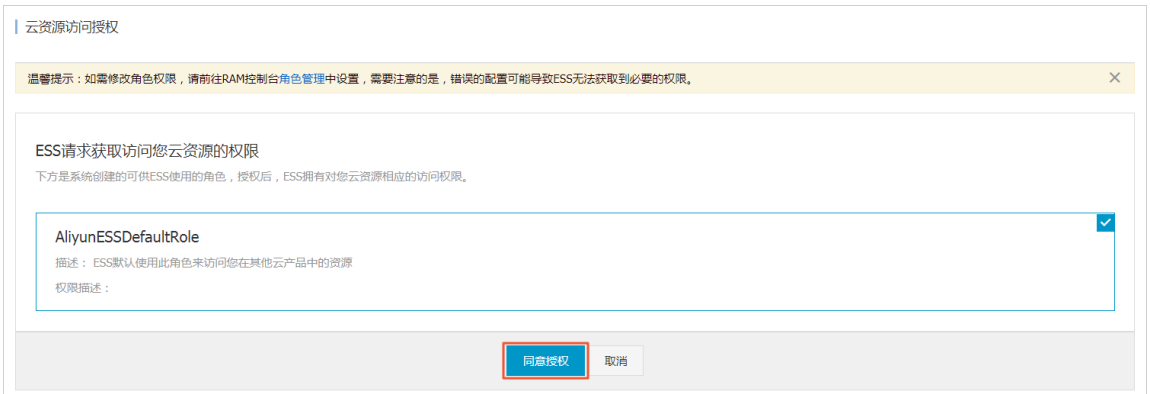
   b. 单击开通ESS服务，进入云产品开通页。

   c. 选中我已阅读并同意复选框，单击立即开通。

d. 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。

e. **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。**

f. **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。

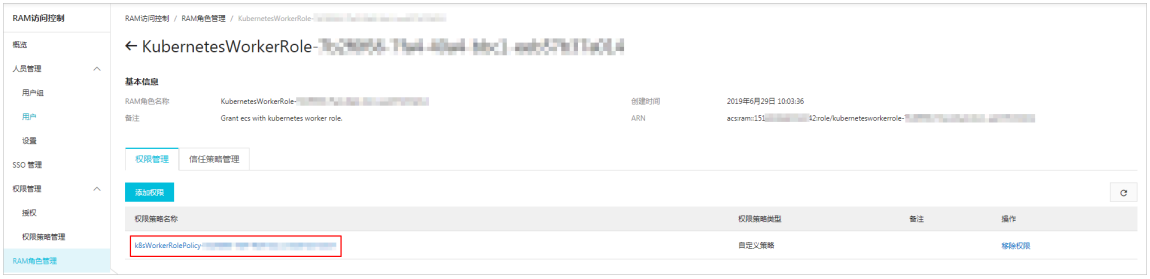
2. 授权角色

a. 单击弹出对话框中的第二个链接，进入RAM角色管理详情页面。

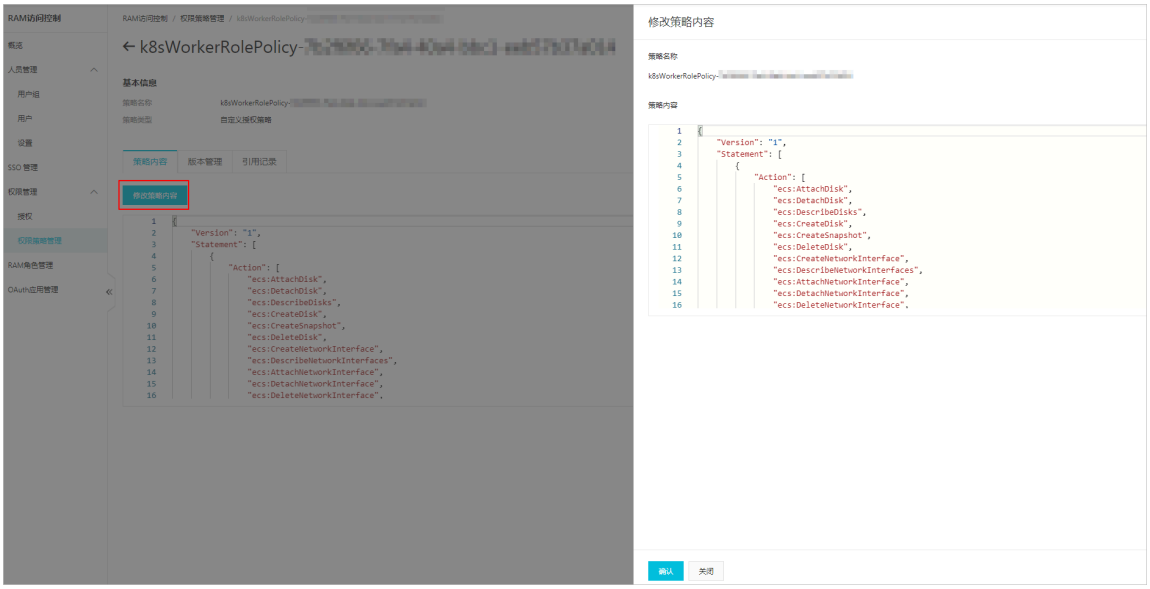
说明:

此处需要以主账号登录控制台。

b. 在权限管理页签，单击目标授权策略名称，进入授权策略详情页面。



c. 单击修改策略内容，从右侧滑出侧边栏修改策略内容页面。



d. 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:

需要在策略内容的任意一个`Action`字段的最后一行补充“，”，再添加以上内容。

e. 单击确认。

配置自动伸缩

1. 在自动伸缩页面，填写以下信息，并单击提交。

配置	说明
集群	目标集群名称。
缩容阈值	cluster-autoscaler管理的伸缩组中，每一个节点的资源申请值（Request）/ 每一个节点的资源容量。当低于配置的阈值时，节点会进行缩容。 **说明:** 弹性伸缩中，扩容会基于调度自动触发，只需设置缩容条件即可。
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（普通/GPU/抢占式），单击操作列创建。

3. 在伸缩组配置填写以下信息。

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。
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。
虚拟交换机	所创建伸缩组的虚拟交换机，包含伸缩组的可用区及Pod地址段。

4. 配置 Worker 节点的信息。

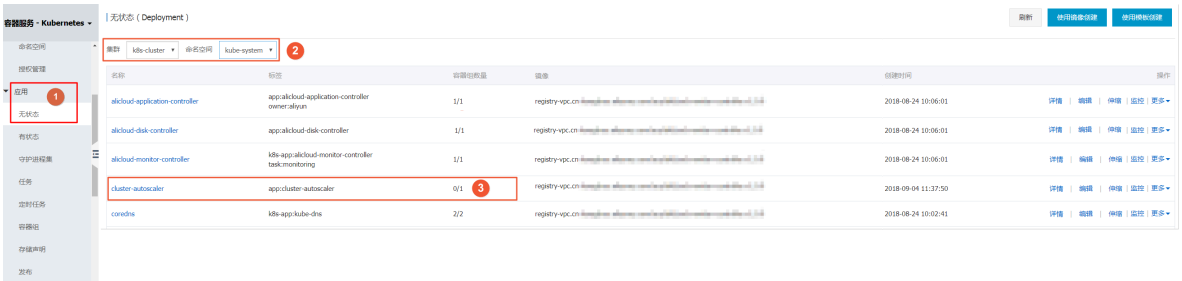
配置	说明
节点类型	所创建的伸缩组的节点类型，与创建集群时所选择的节点类型一致。
实例规格	伸缩组内实例的规格。
已选规格	所选择的伸缩组的实例规格，最多可以选择10种实例规格。
系统盘	伸缩组的系统盘。
挂载数据盘	是否在创建伸缩组时挂载数据盘，默认情况下不挂载。
实例数量	伸缩组所包含的实例数量。 说明: · 实例不包含客户已有的实例。 · 默认情况，实例的最小值是0台，超过0台的时候，集群会默认向伸缩组中添加实例，并将实例加入到伸缩组对应的Kubernetes集群中。
密钥对	登录伸缩后的节点时所使用的密钥对。可以在ECS控制台新建密钥对。 说明: 目前只支持密钥对方式登录。
伸缩模式	支持标准模式和极速模式。
RDS白名单	弹性伸缩后的节点可以访问的RDS实例。
节点标签	在集群中添加节点标签（Label）后，会自动添加到弹性伸缩扩容出的节点上。
污点（Taints）	添加污点后，Kubernetes将不会将Pod调度到这个节点上。

5. 单击确定，创建伸缩组。

预期结果

1. 在自动伸缩页面，可以在普通实例下看到新创建的一个伸缩组。

2. 单击左侧导航栏应用 > 无状态，进入无状态（Deployment）。
3. 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，表示伸缩组已创建成功。



常见问题

- 为什么节点自动伸缩组件无法弹出节点?

  请检查是否存在如下几种场景:

  - 配置伸缩组的实例类型无法满足Pod的资源申请（Request）。默认节点会安装系统组件，Pod的申请资源要小于实例的规格。
  - 是否完整按照步骤执行了授权操作。授权操作是集群维度的，需要每个集群操作一次。
  - 集群是否拥有出网能力。节点自动伸缩组件需要调用阿里云的OpenAPI，因此需要部署节点自动伸缩组件的节点具备出网能力。

- 为什么节点自动伸缩组件无法缩容节点?

  请检查是否存在如下几种场景:

  - 所有的节点Pod的资源申请（Request）阈值高于设置的缩容阈值。
  - 节点上运行`kube-system`undefined命名空间的Pod。
  - 节点上的Pod包含强制的调度策略，导致其他节点无法运行此Pod。
  - 节点上Pod的拥有PodDisruptionBudget，且到达了PodDisruptionBudget的最小值。

  您可以在开源社区得到更多关于节点自动伸缩组件的常见问题与解答。

22.3 虚拟节点

本文主要介绍虚拟节点及如何通过 Virtual Node Addon 插件部署虚拟节点等。

前提条件

- 您需要创建一个 Kubernetes 托管版集群。详情请参见#unique_377。
- 您需要开通弹性容器实例服务。登录弹性容器实例控制台开通相应的服务。

背景信息

虚拟节点来源于社区的 Virtual Kubelet 技术，其实现了 Kubernetes 与弹性容器实例 ECI 的无缝连接，让 Kubernetes 集群轻松获得极大的弹性能力，而不必受限于集群的节点计算容量。关于Virtual Kubelet的工作原理及其架构，请参见Virtual Kubelet。

图 22-1: 应用场景

基于ECI 的虚拟节点支持多种功能，如 GPU 容器实例、挂载 EIP、大规格容器实例等，不仅增强了 Kubernetes 集群的弹性，同时提供了丰富的能力扩展，让用户可以在一个 Kubernetes 集群中轻松管理多种计算 Workload，满足多种场景下的需求。

在混合集群中，真实节点上的 Pod 与虚拟节点上的 ECI Pod 互联互通。

说明:

虚拟节点上的 ECI Pod 是按需收费的，这与真实节点上的计费不同。

ECI 收费规则请参考计费概述。

ECI Pod 规格配置支持 0.25c 至 64c，请参考ECI 实例使用限制。

安装ack-virtual-node插件

1. 登录容器服务管理控制台。
2. 在Kubernetes 菜单下，单击左侧导航栏中的市场 > 应用目录，在右侧选中ack-virtual-node。
3. 在应用目录-ack-virtual-node页面，单击参数，配置虚拟节点参数。

参数	参数含义	获取路径
ECI_VSWITCH	虚拟交换机	您可以节点列表单击某个节点，在实例详情页签的配置信息区域中，获取虚拟交换机的值。
ECI_SECURITY_GROUP	安全组ID	您可以节点列表单击某个节点，在本实例安全组页签的安全组列表区域中，获取安全组ID的值。
ECI_ACCESS_KEY	用户AccessKey	请参见如何获取AccessKey ID和AccessKey Secre
ECI_SECRET_KEY	用户SecretKey	请参见如何获取AccessKey ID和AccessKey Secre

4. 配置完成后，在右侧的创建页面，选择对应的集群，可以看到命名空间已设定为kube-system，发布名称已设定为ack-virtual-node，单击创建。

5. 安装完成后，你可以通过集群 > 节点，在节点列表页面可以看到添加了一个节点virtual-kubelet。

   您可以通过 kubectl 命令，查看节点和helm部署状态，后续也可以通过helm对ack-virtual-node进行升级和管理。详情请参见#unique_378。

在虚拟节点上创建Pod

当集群中存在虚拟节点时，您可以把Pod调度到虚拟节点上，Virtual Kubelet将会创建出相应的ECI Pod。您可以通过以下三种方法操作：

- 通过设置nodeSelector和tolerations来创建Pod。

  1. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，单击右上角的使用模板创建。

  2. 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Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
  nodeSelector:
    type: virtual-kubelet
  tolerations:
  - key: virtual-kubelet.io/provider
    operator: Exists
```

- 通过设置nodeName的方式创建 Pod。

  1. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，单击右上角的使用模板创建。

  2. 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Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
```

```
    name: nginx
  nodeName: virtual-kubelet
```

- 通过配置namespace标签的方式创建Pod。

  1. 通过Cloudshell 连接 Kubernetes 集群。详细内容请参见#unique_378
  2. 执行如下命令，创建Pod。

     ```
     kubectl create ns vk
     kubectl label namespace vk virtual-node-affinity-injection=enabled
     kubectl -n vk run nginx --image nginx
     ```

在应用 > 容器组中，当出现如下界面时，表示 Pod 部署完成。

相关参考

- #unique_379
- #unique_380
- #unique_381